POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering

Academic year 2019/2020

Collegio di Ingegneria Informatica, del Cinema e Meccatronica (ICM)
Departement of Control and Computer Engineering (DAUIN)

Master's Degree Thesis

Asynchronous Embedded Model Control for Robotic Applications

Academic Supervisors

Prof. Carlo NOVARA
PhD Carlos Norberto PEREZ MONTENEGRO

Candidate

Luca NANU - S255351

July 2020

Abstract

A controlled physical system can be basically divided in two parts: the *controller* and the *physical plant* to be controlled. Controller output is called *control input*, which enters as input of plant to make it follow a desired behaviour of its *output measurements*.
In many controlled physical systems, the time needed to send control inputs/measurements is not fixed, but variable. In Networked Controlled systems (NCS) the control is made in the presence of a communication networks, where some data are subject to delays or losses. In a vehicle engine, fuel injection can be controlled with variable timing to increase performances/energy efficiency. In both cases (and others too), variable sampling time can be considered as plant disturbance, making more difficult controlling the system with desired behaviour.
To deal with this problem, the *Embedded Model Control (EMC)* well-established technique can be exploited. It is a model-based control technique, meaning that a simplified version of the plant to be controlled is built (the Embedded Model EM). With EMC the plant system can be controlled to reach a desired behaviour using a suitable controller (*control law*), with the possibility of reducing the plant disturbances.

Main objective of the thesis is to study asynchronous sampling time plants controlled by EMC technique (prosecuting the literature work), focusing in experimental tests with physical systems. In particular a ground differential-drive vehicle with two DC motors controlling the wheels is selected, equipped with a Raspberry Board.
2 main tests are performed: Model-in-the-Loop (MIL) and experimental tests (Robot Hardware Implementation Tests RHIT) where EMC is translated into software code (C++) and executed in robot Raspberry board.
With EMC technique are controlled, with asynchronous sampling time: robot DC motors to move the two wheels. When DC motors EMC are available can be controlled robot orientation only (make it turn in place or follow a circular trajectory) and robot longitudinal position only (control the robot to make it follow a straight line). Finally all previous EMC can be combined together to control both robot orientation and longitudinal position (the robot can move in 2D space till a certain longitudinal position and orientation angle).
A *non-conventional hierarchic structure* is used to connect control robot orientation/position with EMC. Indeed usually to control robot position in 2D space, motors and robot dynamics are combined together in a unique controller with a very complex structure, instead with hierarchic structure motors and orientation/position control blocks are separated: this lead to lower complexity of models, which are easier to be controlled.

The outputs of EMC are estimation of some plant variables (wheel speeds, robot yaw angle and position). Test results can be mainly judged first with *tracking error*, which is the difference between estimated state and a target reference to be tracked (i.e. reach a desired robot position and orientation in space or certain wheel speeds), second with *model error*, which is the difference between EM estimations and real plant measurements. The lower these errors are, better is the control.
For all tests made, model and tracking errors are sufficiently low, especially for DC motors where the difference between EMC and real plant is huge for its complex dynamics: it means that estimated disturbances are near to real ones.
It is also verified that EMC technique is able to reduce noises and disturbances of real robot plant (included the ones introduced by variable sampling time) not only in simulation MIL tests but also practically with RHIT, especially for DC motors. This means that with a simplified model (EM) even complex plants affected by many disturbances can be controlled with quite high precision.
Hierarchical structure is successfully verified, since with this method robot is able to reach almost precisely a desired position with a certain trajectory in space. This is a huge result because a robot can be controlled using simple linear models, avoiding complex non-linear terms and conversions.

Acknowledgements for thesis work

I would like to thank: my academic supervisors Prof. Carlo Novara and PhD Carlos Perez for giving me the opportunity of working with this thesis project and for their support, LaDiSpe and DAUIN Polytechnic of Turin laboratories for experimental material and assist

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview and main thesis objectives

In control theory, the actual behaviour of a physical dynamic system in continuous-time (CT) domain (*plant*) can be modified and corrected to follow a desired behaviour using a control model in discrete-time (DT) domain (*controller*). Looking at Figure 1.1, the plant outputs to be controlled are the *measurements* $y(k)$ and they enters the controller. A *control input* $u(k)$ is obtained as output and it enters the plant to modify properly its behaviour. To convert from DT to CT a digital-to-analogue converter (D/A) is needed, and for viceversa an analogue-to-digital converter (A/D). A system like that is also referred as *Closed loop control system.* The desired behaviour wanted for the plant is referred as *reference* signal $r(t)$: for example a robot following a trajectory to reach a certain position in space, eventually with obstacles on the way.

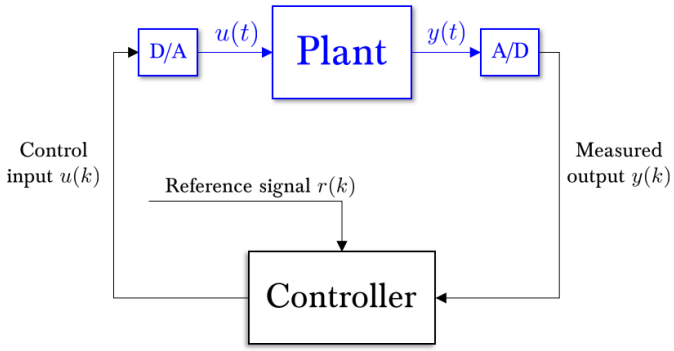


Figure 1.1: General control scheme, blue for CT and black for DT

Often control inputs/measurements are not exchanged between plant/controller with a fixed timing, but variable: looking at top figure in Figure 1.3 fixed timing is when the time needed to receive the next data is always the same, i.e. there is a fixed *inter-arrival time* (in this case $25\,\mathrm{ms}$). Instead in bottom figure the inter-arrival time is not fixed at $25\,\mathrm{ms}$, but can be variable in a certain range: we can only define, if possible, a *minimum* inter-arrival time.
There are many physical systems working with variable sampling time:

- In Networked Control system (NCS) one or more controllers and plants are connected together through a shared communication network. This is what is shown in Figure 1.2, where the network *closes the loop* between all the parts. Network is often *wireless*, i.e. it doesn't require physical connections between senders and receivers (*remote control*): an example can be robot drones moved by using another device like a computer.

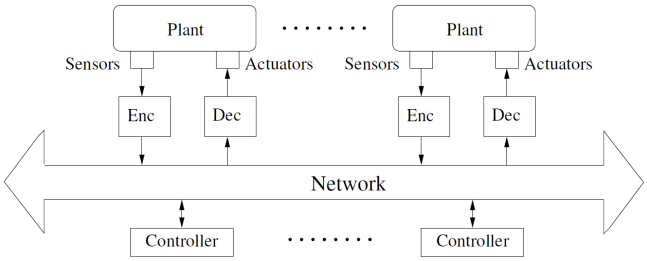


Figure 1.2: General NCS architecture

Source: [15]

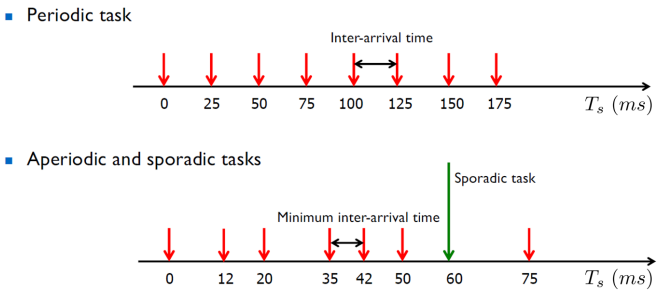


Figure 1.3: Example of fixed and variable sampling times (periodic/aperiodic tasks)

Source: [13]

Control inputs/measurements and other data are divided in units called *packets*, sampled and encoded/decoded in DT/CT to be transmitted. However in practice there are delays increasing the time needed for packets exchange. In this case the asynchronous timing is not intentional but comes from external factors. Main delays in NCS are due to, [15]: network access delays (time needed to accept data), transmission delays (time occurred in network between sending and receiving packets), package dropouts (data can be lost in transit through network).

- In other controlled systems the asynchronous timing is a desired condition (intentional): a common example can be a vehicle engine, where a variable to be controlled is the fuel injection inside the motor pistons. To increase performances and/or make the engine more energy efficient, injection can be performed with variable timing.

In these kind of systems, variable sampling time can be considered as disturbances affecting the plant, which make more difficult the system to be controlled with desired behaviour. In this regard, Embedded Model Control (EMC) is a well established control method. It belongs to model-based design methods class, i.e. requiring a simplified model (neglecting plant complex dynamics) of the plant to be controlled, usually in the form of DT *State-space equations (SS)*, the state variables are *estimations* of plant physical variables. In EMC case, the simplified model is called *Embedded Model (EM)*. This model is then connected with the *Control Law unit* part which finally is able to control physical quantities of a plant. In EMC field the plant is also called *fine model*. EMC fundamentals are summarized in [1], and example applications are treated in [3, 5, 6].
One handful property of EMC is to guarantee stability and desired behaviour of system even with presence of asynchronous sampling times of both command inputs and measurements. Indeed the model carries informations about disturbances of the system, and the control can be built to satisfy *active disturbance rejection*, i.e. trying to reduce and possibly remove disturbances at every time step. This feature is discussed in [4, 5], mainly tested in simulation.
This differentiate EMC from other model-based control methods, like internal model control (IMC) and model predictive control (MPC) where control models don't treat disturbance dynamics and consequently disturbance rejection [1].

Main objective of the thesis is prosecute the study of asynchronous sampling time plants controlled by EMC technique, [4, 5], focusing in experimental tests with physical systems. In particular a ground robot is selected, consisting in a differential drive vehicle, composed by 2 direct-current (DC) independent motors controlling the

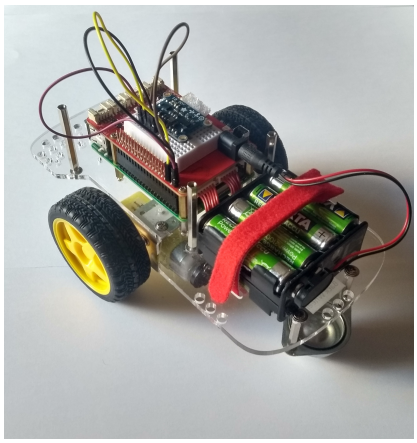


Figure 1.4: GoPiGo3 Differential-drive robot used for the thesis

wheels equipped with position encoders, and based on Raspbian operating system running inside a Raspberry Pi board. The robot is called GoPiGo3 and is produced by Dexter Industries company, [17]. Asynchronous EMC can be tested in 2 conditions:

- With Model-in-the-Loop (MIL) tests, making first EMC models to control the motors, and then to control robot orientation and position. This test requires both control models and robot plant to be run in simulation, in this case using MathWorks Simulink software (SW) application.

- Much more important, translate simulated EMC into SW code model (using C++ language) to be executed in Raspberry embedded system of the robot. Making physical tests allows better understand effectiveness of EMC control technique, because often the real plants present different behaviours with respect the ones tested in simulations. Since no specific name is given to experimental implementations of a control model, we will refer from now on as Robot Hardware Implementation Test (RHIT). Hardware (HW) Timers present in robot Raspberry board will be used to schedule sent and received data from control model and robot plant.

This thesis work tests is focused in studying and verifying EMC technique with asynchronous timing conditions, without considering for example network remote connections: indeed it was more important verify the control EMC technique.
Since no network connection is considered, to mimic asynchronous timing conditions will be used hardware timers already present in Raspberry board, which expiration time is defined by random numbers in a predefined range. This method can be considered more critical because with network control systems the timestamp variation is not so heavy (if network connection is reliable).
Trajectory reference followed by controlled robot is very simple: it needs to reach a certain target without obstacles on the way (wheel angular speed or robot position/orientation) only respecting the dynamic constraints of the system (geometrical constraints etc.). This allows the overall EMC to be run directly in Raspberry board, for RHIT tests. However, in many practical applications, reference trajectories are very complex, requiring obstacle avoidance and optimal path (for example to reduce supply/fuel consumption etc.). In these cases other techniques are required, exploiting machine learning and optimal problems to be solved, which are impossible to be loaded in a board like Raspberry. Hence network control come in help, allowing EMC part to be run in a powerful computer. This can be implemented in the future, which is not difficult if EMC control is already built.

1.2 Structure of the thesis

The thesis is structured in 6 main parts.

- **Part I**: Basic concepts of EMC are explained briefly following the existing literature.
  A differential drive robot ground vehicle called GoPiGo3 is used to make EMC practical RHIT tests. Basic robot construction properties are given, focusing on the main geometric and mass parameters needed to build EMC models.
  Finally are presented settings used to make simulation MIL using Simulink, and RHIT using Raspberry Pi board available in the robot.

Next parts are structured observing the time workflow of the thesis: at the beginning DC motors control is considered, then robot orientation and longitudinal position separately, finally the last 2 models are combined together to control 2D space robot position.
For every step, first the EMC model is theorized, then tested in MIL simulations using Simulink, at the end generated code is recovered from simulated model to be implemented in robot Raspberry board and make RHIT tests in real world.
Results of relevant tests are then saved, organized and discussed.

- **Part II**: Since very few datasheet informations are available for DC motors, first phase is *system identification* of the main parameters. After obtaining reliable parameters, EMC can be built and tested in MIL simulation. Every model must be connected with a simulated system (fine model) miming as close as possible physical motors, to establish similar real conditions.
  When results obtained in simulations become acceptable, physical RHIT tests can be done: the models are connected with the real plants (the 2 motors). No load conditions are considered first (i.e. the wheels not attached to the ground), for the left and right motors separately. Afterwards robot is placed to the ground, with load conditions, and the left/right motor models are tested separately and together.

Similar procedure used for motors is used for next control models too, treated in the following parts: theoretical analysis, MIL Simulink tests using a fine model of robot orientation/position, practical RHIT tests using robot, analysis of the results.
Specifically, a hierarchic structure is used to connect motors and robot orientation/position control models: estimated longitudinal and angular speeds are the outputs of orientation/position EMC, and they're converted into motor angular speeds with robot kinematic equations and used as reference for motors EMC. Then control inputs of DC motors EMC are used as input of robot plant, to control it physically. A simplified scheme (used in RHIT) of this hierarchic structure is shown in Figure 1.5: the non-conventional part is highlighted in red. The difference relies in the simplicity of the structure, because in commonly used structures motors and orientation/position controls are combined together in a unique model, requiring the presence of non-linear terms inside which usually lead to high complexity. In the way analysed in the thesis very simple and linear control models are exploited, and every non-linear term is let outside (the ones related to conversions for example).

- **Part III**: Robot orientation is then considered, allowing *only* control of robot yaw angle θ_z (i.e. angle around vertical axis) and z-axis angular speed ω_z. No position control is considered in this field. Angle and angular speed measurements are taken with an Inertial Measurements Unit (IMU) attached to robot. IMU is first calibrated and tested with already-built motors EMC.

- **Part IV**: Robot position EMC is studied in this part, allowing to control *only* robot longitudinal position

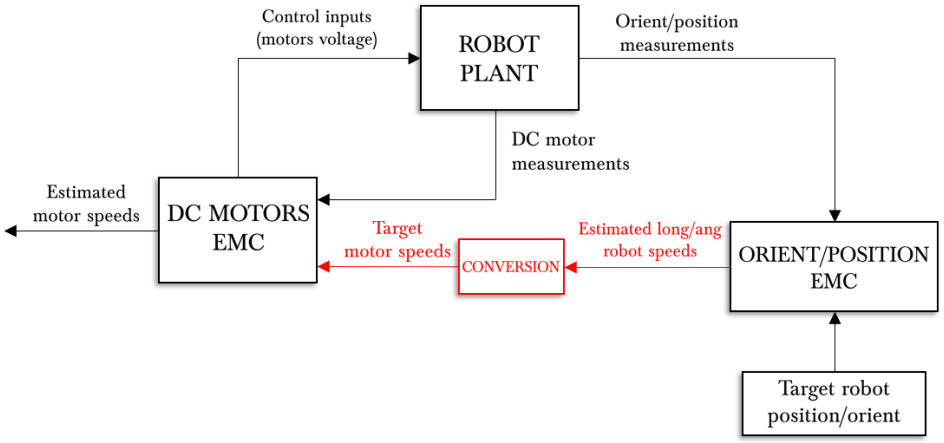


Figure 1.5: Orientation/position hierarchic structure simplified

ξ and longitudinal speed v_ξ. In reverse, now no orientation control is present, in other words the robot is allowed to go back and forth only. The position and longitudinal speed can be measured starting from DC motor encoders.

- **Part V**: Finally the previous 2 models are combined together to control both longitudinal and lateral position of robot. This allows the robot to move in planar space, but using polar-like coordinates θ_z, ξ instead of Cartesian coordinates ξ, ν, exploiting hierarchical structure.

- **Part VI**: Main objectives reached and final conclusions are the subject of this part, with references to possible improvements to be done in the future.

Notes: For many MIL and RHIT tests several plots are present: in these cases only the test settings are referenced in the *List of Figures* to avoid adding a long list (the number indicates the first figure of the test).
Unless specified, all figures are made by the author.
In the final pages after bibliography, a *Nomenclature* to help the reader is added summarizing the acronyms and main symbols used inside the thesis.

Part I

Overview

Chapter 2

EMC theory

2.1 EMC basic concepts

The basic concepts of the Embedded Model Control approach are first analysed, a short summary is presented taken from [1]. The main parts composing an EMC are shown in Figure 2.1.

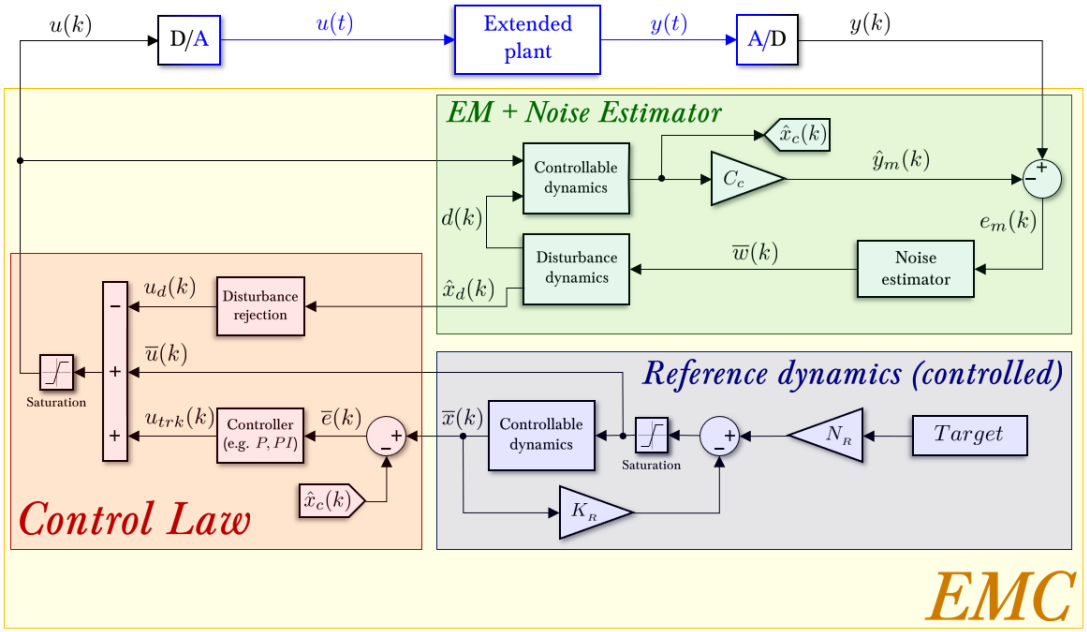


Figure 2.1: EMC complete scheme

The blocks in CT domain are coloured in blue, the ones in DT are in black. With respect [1], some parts of the scheme are modified according to what is needed as thesis objectives (e.g. reference dynamics part). Going in detail of each block, referencing to the figure:

- **Embedded Model (EM) + Noise estimator**: Since the physical plant often have unknown parameters and dynamics, one can build a model omitting them. This results in a simplified model called Embedded model. It can be decomposed in 3 main parts:
  - Controllable dynamics: Part containing the DT equations in which states are controllable $\hat{x}_c(k)$, i.e. an external control input can change internal states initial conditions to different final conditions. Hence

with a well suited control algorithm the behaviour of these states can be arbitrary decided.

- Disturbance dynamics: The physical plant can be affected by non-causal noises (i.e. they do not depend on their own past history) or causal disturbances (i.e. they are DT states).
  The last ones can be inserted in EM model disturbance dynamics part as non-controllable states $\hat{x}_d(k)$, i.e. they cannot be controlled by an arbitrary external control input. Then they also affect controllable dynamics, $d(k)$.

- Noise estimator: It's an output-to-state feedback estimator, which takes as input the difference of plant measurement $y(k)$ and estimated output $\hat{y}_m(k)$ (called ***Model error***), to estimate as outputs plant noises $\overline{w}(k)$.

- **Reference dynamics**: Inside there is only EM controllable dynamics part without disturbances. If a target output is imposed, the outputs are the reference states (without disturbances) to be tracked by EM controllable states (with disturbances). For thesis work, to obtain a desired reference trajectory a ***static-state feedback control*** can be added (matrices K_R and N_R) [14], and a saturation block limits the reference input $\overline{u}(k)$. The reference state is $\overline{x}(k)$. $\overline{u}(k)$ and $\overline{x}(k)$ are the used in Control law.
  More complex techniques can be considered, for example exploiting machine learning and optimization methods, but they are not needed for the objectives of the thesis.

- **Control Law**: Main aim of control part is to give as output a control input $u(k)$ to make physical plant behave as desired. Specifically we want estimated states in EM to follow a desired tracking reference input, hence reduce what is called ***Tracking error***: it's the difference between reference $\overline{x}(k)$ and estimated states $\hat{x}_c(k)$. For thesis work, to control tracking error is sufficient using either a proportional (P) or proportional-integral (PI) controller.
  Following suitable assumptions, the control is also able to reduce causal disturbances present in the physical plant (active rejection). This is one of the main differences with respect to other control methods.

The states of EMC model in DT domain are $x^T(k) = [\hat{x}_c, \hat{x}_d](k)$, with $\hat{x}_c(k)$ as controllable states and $\hat{x}_d(k)$ as non controllable states (disturbances). The equations in discrete time and matrix form are (state and output equations) (index is $k \geq 0$):

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) + G\overline{w}(k), & x(0) &= x_0 \\ \hat{y}_m(k) &= Cx(k), & z_m(k) &= Fx(k) \end{aligned} \tag{2.1}$$

In our case the performance channel and the EM output coincide, $z_m(k) = \hat{y}_m(k)$, $C = F$. The corresponding matrices are:

$$A = \begin{bmatrix} A_c & H_c \\ 0 & A_d \end{bmatrix}, \quad B = \begin{bmatrix} B_c \\ 0 \end{bmatrix}, \quad G = \begin{bmatrix} G_c \\ G_d \end{bmatrix}, \quad C = \begin{bmatrix} C_c & C_d \end{bmatrix}, \quad F = \begin{bmatrix} F_c & 0 \end{bmatrix} = C \tag{2.2}$$

Assumptions 2.1 *Pairs (A_c, B_c) are assumed to be controllable (indeed a controller like proportional (P) or proportional-integral (PI) can be exploited to track a reference), while (C_c, A_c) and (C, A) are assumed to be observable at least for one controllable state (hence at least one output from plant can be measured with a sensor), [1].*

The discrete time model can be decomposed into controllable and disturbance equations:

$$\begin{aligned} \hat{x}_c(k+1) &= A_c\hat{x}_c(k) + B_c u(k) + d(k), & \hat{x}_c(0) &= x_{c0} \\ \hat{x}_d(k+1) &= A_d\hat{x}_d(k) + G_d\overline{w}(k), & \hat{x}_d(0) &= x_{d0} \\ d(k) &= H_c\hat{x}_d(k) + G_c\overline{w}(k) \end{aligned} \tag{2.3}$$

The term $d(k)$ contains disturbance states $\hat{x}_d(k)$ and noises $\overline{w}(k)$ entering the controllable dynamics.
Reference dynamics: It is composed by controllable dynamics without disturbances, i.e. it has the same equation form of $\hat{x}_c(k+1)$ in Equation (2.3) but without $d(k)$ (the reference states and outputs are from now on referred with a "bar" above the symbol):

$$\begin{aligned} \overline{x}(k+1) &= A_c\overline{x}(k) + B_c\overline{u}(k), & \overline{x}(0) &= \overline{x}_0 \\ \overline{y}(k) &= C_c\overline{x}(k) \end{aligned} \tag{2.4}$$

Static-state FB can be implemented to Equation (2.4) if the input is considered as $\overline{u}(k) = -K_R\overline{x}(k) + N_R\overline{r}(k)$, where $\overline{r}(k)$ is the desired target reference:

$$\begin{aligned} \overline{x}(k+1) &= \underbrace{(A_c - B_cK_R)}_{A_R}\overline{x}(k) + \underbrace{B_cN_R}_{B_R}\overline{r}(k) \\ \overline{y}(k) &= \underbrace{C_c}_{C_R}\overline{x}(k) \end{aligned} \tag{2.5}$$

Figure 2.2 shows the closed loop system formed by reference dynamics and static-state FB control with K_R, N_R matrices.

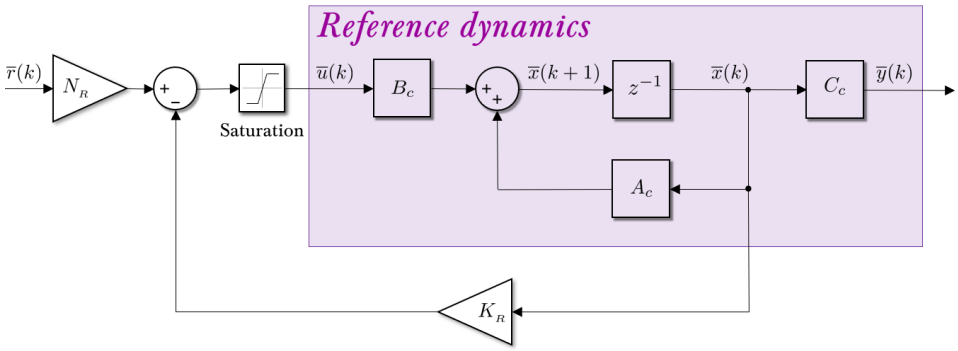


Figure 2.2: Orientation EM Reference Dynamics

Reference control input $\overline{u}(k)$ is ***saturated*** avoiding too high/low values which are impossible to be obtained physically.
K_R can be found with pole placement technique to control the CL matrix A_R. Instead N_R can be found by considering the input/output transfer function (in frequency domain) between $\overline{r}(z)$ and $\overline{y}(z)$, which is:

$$W(z) = \frac{\overline{y}(z)}{\overline{r}(z)} = \frac{C_R}{z\mathbb{I} - A_R}B_RN_R$$

N_R is needed to control the DC-gain K_{DC} of the overall reference dynamics system, and make it equal to 1 to

avoid an increase/decrease of output amplitude wrt input:

$$K_{DC} = \lim_{z\to 1} W(z) = \frac{C_R}{z\mathbb{I} - A_R} B_R N_R = 1$$

Solving the limit, N_R is equal to:

$$N_R = \left[C_R \left(\mathbb{I} - A_R\right)^{-1} B_R\right]^{-1} \tag{2.6}$$

Noise estimator: Used to estimate disturbance terms, starting from difference between real and simulated outputs, $e_m(k) = y(k) - \hat{y}_m(k)$. 2 main estimators can be build [1, 2]:

- **Static feedback noise estimator**: Can be exploited if dimension of total states $\boldsymbol{n_x}$ is exactly equal to the one of disturbance inputs $\boldsymbol{n_w}$. In this case the estimation of noises $\overline{w}(k)$ can be obtained using a simple static gain L:

  $$\overline{w}(k) = L e_m(k)$$

  L components can be found making equal the coefficients $a_{i,cl}$ of the characteristic polynomial of the closed loop matrix $A_{CL} = A - GLC$, computed with $\det[A_{CL} - \lambda\mathbb{I}]$, and the ones $a_{i,des}$ of a characteristic polynomial with discrete time eigenvalues p_i decided by us with $Re(\lambda) < 1$, to guarantee asymptotic internal/BIBO stability. For example, if A_{CL} is a 3×3 matrix, the 2 characteristic polynomials are:

  $$\begin{cases} \lambda^3 + a_{2,cl}\lambda^2 + a_{4,cl}\lambda + a_{3,cl} & \textit{Closed loop char. polynomial} \\ \lambda^3 + a_{2,des}\lambda^2 + a_{3,des}\lambda + a_{4,des} = (\lambda - p_1)(\lambda - p_2)(\lambda - p_3) & \textit{Desidered char. polynomial} \end{cases} \tag{2.7}$$

- **Dynamic feedback noise estimator**: This is the condition where $\boldsymbol{n_x} > \boldsymbol{n_w}$. This means that closed loop matrix $A_{CL} = A - GLC$ is not stabilizable using a simple static gain, every choice of matrix L is considered. Internal stability can only be recovered by adding a dynamic feedback of order $n - n_w$ (usually the order is 1). A new state $\hat{x}_e(k)$ is added in the CL system, and $\hat{x}_p^T(k) = [\hat{x}_c, \hat{x}_d](k)$. The new SS equations become:

  $$\begin{aligned} \hat{x}_p(k+1) &= A\hat{x}_p(k) + Bu(k) + G\overline{w}(k) \\ \hat{x}_e(k+1) &= A_e\hat{x}_e(k) + L_e\overline{e}_m(k) \\ \overline{w}(k) &= N\hat{x}_e(k) + L\overline{e}_m(k) \quad , \quad \overline{e}_m(k) = y(k) - \hat{y}_m(k) \end{aligned} \tag{2.8}$$

  In particular, the unknown matrices and variables are:

  $$N = \begin{bmatrix} N_w \\ N_d \end{bmatrix} \quad , \quad L = \begin{bmatrix} L_w \\ L_d \end{bmatrix} \quad , \quad A_e = 1 - \beta \tag{2.9}$$

  A_e usually has the form of a filter. L_e can be considered equal to 1.
  Making some arrangements:

  $$\hat{x}(k+1) = \begin{bmatrix} \hat{x}_p \\ \hat{x}_e \end{bmatrix}(k+1) = \underbrace{\begin{bmatrix} A - GLC & GN \\ -L_eC & A_e \end{bmatrix}}_{A_{CL}} \begin{bmatrix} \hat{x}_p \\ \hat{x}_e \end{bmatrix}(k) = \underbrace{\begin{bmatrix} B & GL \\ 0 & Le \end{bmatrix}}_{B_{CL}} \begin{bmatrix} u \\ y \end{bmatrix}(k) \tag{2.10}$$

  From CL matrix A_{CL} we can use again eigenvalue placement technique to find the unknown parameters $N_w, N_d, L_w, L_d, \beta$.

Using this method the estimated disturbance $\overline{w}$ in frequency domain will be:

$$\overline{w}(z) = L(z)\overline{e}_m(z)$$
$$L(z) = L + M\frac{1}{z - A_e}$$

In which it can be noticed the filter $z - A_e$ at denominator. Furthermore $M = NL_e$ (if $L_e = 1 \Rightarrow M = N$). In Figure 2.3 a general scheme of the dynamic feedback estimator is shown.

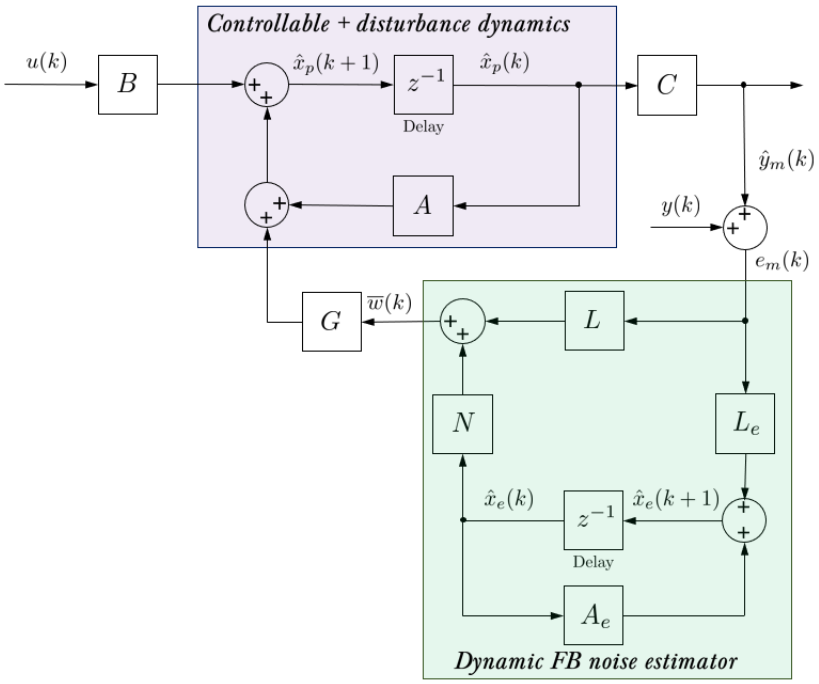


Figure 2.3: EM dynamic feedback noise estimator block scheme

Control law: It needs to respect the following requirements:

- Make the discrete control EM model follow a reference input.
- Cancel the disturbances of the real plant.

By assumption $\hat{x}_d(k)$ are not controllable by control input $u(k)$, in other words the matrices pair (A, B) is not controllable. Hence overall closed loop system results to be not stabilizable (internally). The only way to make the system internally stable is to cancel $x_d(k)$ terms and reduce (bound) effects of noises $\overline{w}(k)$, [1].
To this aim, the control law is made by considering the following LTI equation:

$$\begin{aligned} u(k) &= \overline{u}(k) + \underbrace{K\overline{e}(k)}_{u_{trk}(k)} - \underbrace{M\hat{x}_d(k)}_{u_d(k)} \\ \overline{e}(k) &= \overline{x} - (\hat{x}_c + Q\hat{x}_d) \end{aligned} \tag{2.11}$$

$\overline{e}(k) = e_{trk}(k)$ is the *tracking error* taking into account the difference between reference and EM dynamics (but in many practical cases $Q = 0_{ij}$), the term $u_{trk}(k) = K\overline{e}(k)$ is needed to make our model follow the reference, $u_d(k) = M\hat{x}_d(k)$ is the term needed to cancel the disturbance and to make the system stable, $\overline{u}(k)$ is the reference input.

Matrices K, Q and M need to be designed the following necessary and sufficient conditions:

$$\begin{aligned} &\textbf{1.} \quad A_c - B_c K \text{ asymptotically stable} \\ &\textbf{2.} \quad \begin{bmatrix} H_c + Q A_d \\ C_d \end{bmatrix} = \begin{bmatrix} A_c & B_c \\ F_c & 0 \end{bmatrix} \begin{bmatrix} Q \\ M \end{bmatrix} \end{aligned} \tag{2.12}$$

The 2^{nd} condition is also known as Davison-Francis relationship: if holds, the tracking error is bounded and the mean value tends to zero with control law equation, [1, 6].
Note: Reference, control law and noise estimator require eigenvalue tuning using pole placement technique. Since almost all of the tests will be performed with variable DT sampling time, the eigenvalues must be chosen properly. It is known that even if DT eigenvalues λ_i changes with sampling time T, CT ones μ_i does not: hence first CT eigenvalues are chosen, then a conversion to DT one is done using the relation $\lambda_i = e^{\mu_i T}$, at every time step.

2.2 EMC example - Mass-spring-damper system

A simple plant can be considered to learn about EMC technique, for example a *Mass-spring-damper (MSD) system*, where a mass m is connected with a spring of stiffness coefficient k and an ideal viscous damper with coefficient c, a force $F(t)$ is responsible of motion. Since it is required as one of the main objectives, EMC block is directly affected by variable step time: in this example the range is $T = T_s = 0.02\text{–}0.04\,\text{s}$ (cfr. Figure 2.5).
The governing CT differential equations for MSD are:

$$\begin{cases} \dot{x}_1(t) = x_2(t) \\ \dot{x}_2(t) = -\dfrac{k}{m} x_1(t) - \dfrac{c}{m} x_2(t) + \dfrac{1}{m} u(t) + \hat{x}_d(t) + w_1(t)\,, \quad y_1(t) = x_1(t), \quad y_2(t) = x_2(t) \\ \dot{x}_d(t) = w_2(t) \end{cases} \tag{2.13}$$

$u(t) = F(t)$ is the controlled input (force) coming from EMC (analogue converted), $w^T(t) = [w_1, w_2](t)$ are the noise error terms entering the system. State vector $x^T(t) = [x_1, x_2, x_d](t)$ is composed by position $x_1(t) = \xi(t)$, velocity $x_2(t) = v_\xi(t)$ of the mass, and disturbance state $\hat{x}_d(t)$. The last equation is added to simulate a non-controllable disturbance term, i.e a term that cannot be changed by modifying the input $u(t)$. $\hat{x}_d(t)$ enters the system as acceleration disturbance.
In matrix form (continuous time CT):

$$A_{CT} = \begin{bmatrix} 0 & 1 & 0 \\ -\frac{k}{m} & -\frac{c}{m} & 1 \\ 0 & 0 & 0 \end{bmatrix}, \quad B_{CT} = \begin{bmatrix} 0 \\ \frac{1}{m} \\ 0 \end{bmatrix}, \quad G_{CT} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix},$$
$$C_{CT} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad F_{CT} = C_{CT}$$

The Mass-spring-damper fine model is in Figure 2.4. Starting from Forward Euler discretization method, DT matrices can be recovered from CT ones ($\mathbb{I}$ is the identity matrix and T the sample time):

$$\begin{aligned} A_{DT} &= A_{CT} T + \mathbb{I} \\ B_{DT} &= B_{CT} T \\ C_{DT} &= C_{CT} \\ G_{DT} &= G_{CT} T \end{aligned} \tag{2.14}$$

For our system:

$$A_{DT} = \begin{bmatrix} A_c & H_c \\ 0 & A_d \end{bmatrix} = \left[\begin{array}{cc|c} 1 & T & 0 \\ -\frac{kT}{m} & -\frac{cT}{m}+1 & T \\ \hline 0 & 0 & 1 \end{array}\right], \quad B_{DT} = \begin{bmatrix} B_c \\ 0 \end{bmatrix} = \left[\begin{array}{c} 0 \\ \frac{T}{m} \\ \hline 0 \end{array}\right], \quad G_{DT} = \begin{bmatrix} G_c \\ G_d \end{bmatrix} = \left[\begin{array}{cc} 0 & 0 \\ T & 0 \\ \hline 0 & T \end{array}\right] \tag{2.15}$$
$$C_{DT} = \begin{bmatrix} C_c & C_d \end{bmatrix} = \left[\begin{array}{cc|c} 1 & 0 & 0 \\ 0 & 1 & 0 \end{array}\right], \quad F_{DT} = \begin{bmatrix} F_c & 0 \end{bmatrix} = C_{DT}$$

Since measured outputs are both position $\xi(k)$ and speed $v_\xi(k)$ there are 2 model errors, $e_{m1}(k) = \xi(k) - \hat{\xi}(k)$ and $e_{m2}(k) = v_\xi(k) - \hat{v}_\xi(k)$. Considering matrix related to disturbances G_{DT} in Equation (2.15), noise estimator is composed by a static gain matrix $L \in \mathbb{R}^{2\times 2}$. It can be used pole placement technique procedure starting from arbitrary decided poles p_{n1}, p_{n2}, p_{n3}, like in Equation (2.7):

$$\lambda^3 + a_{n2}\lambda^2 + a_{n3}\lambda + a_{n4} = (\lambda - p_{n1})(\lambda - p_{n2})(\lambda - p_{n3})$$

Noise estimation gain matrix will be, in function of characteristic polynomial terms a_{n2}, a_{n3}, a_{n4} and main parameters m, k, c:

$$L = \begin{bmatrix} l_{11} & l_{12} \\ l_{21} & l_{22} \end{bmatrix} \qquad \begin{aligned} l_{11} &= \frac{m\,(3 + 2a_{n2} + a_{n3}) - kT^2}{mT^2}, \qquad l_{12} = \frac{m\,(3 + a_{n2}) - cT}{mT} \\ l_{21} &= \frac{a_{n2} + a_{n3} + a_{n4} + 1}{T^3}, \qquad\qquad l_{22} = 0 \end{aligned}$$

For control law it can be considered a Proportional controller (P), and Equation (2.11), where matrices M, Q, K must be designed. From 2nd condition of Equation (2.12) $M = m$ and $Q = [0, 0]^T$.
Instead for design of matrix $K \in \mathbb{R}^{2\times 1}$ it can be applied again pole placement technique (Equation (2.7)) where p_{k1}, p_{k2} are arbitrary decided poles:

$$\lambda^2 + \lambda a_{k2} + a_{k3} = (\lambda - p_{k1})(\lambda - p_{k2})$$

and matrix K components are:

$$K = \begin{bmatrix} k_1 & k_2 \end{bmatrix} \qquad \begin{cases} k_1 = \dfrac{-kT^2 + m\,(1 + a_{k2} + a_{k3})}{T^2} \\ k_2 = \dfrac{m\,(2 + a_{k2}) - cT}{T} \end{cases}$$

For initial conditions position $\xi_0 = 0\,\mathrm{m}$ and velocity $v_{\xi,0} = 0\,\mathrm{m/s}$ for fine model plant are used. Target position to be reached is $\tilde{\xi} = 2\,\mathrm{m}$. The parameters in fine model are slightly different from EMC ones, to simulate parametric uncertainty (Table 2.1).

Parameter (meas. unit)	Fine model	EMC
m (kg)	1	1.5
k (N/m)	1	0.7
c (Ns/m)	1	1.4

Table 2.1: MSD fine model vs EMC parameters

CT Eigenvalue type	Values
Reference μ_R	-1.5230×2
Noise estimator μ_N	$-2.5648, -2.5646 \times 2$
Control μ_K	-2.5647×2

Table 2.2: Continuous eigenvalues MSD EMC

In addition noise disturbances are added to acceleration in fine model, with 2 uniform random numbers $w_1(t)$

and $w_2(t)$, the last integrated once (check Equation (2.13)). In particular their ranges are $\epsilon_{w1} = \epsilon_{w2} = 0\text{–}1\text{m/s}$. After some tests, CT eigenvalues considered for reference, control law and noise estimator blocks are shown in Table 2.2. Reference dynamics is quite slow, hence μ_R are near to zero, instead μ_N and μ_K are sufficiently fast to guarantee low tracking and model errors. At every time step CT eigenvalue must be converted in DT equivalent (λ_i), depending on variable sampling time T, using the relation $\lambda = e^{T\mu}$.

In the next set of figures, measured output $y(k)$ is compared with estimated $\hat{y}(k)$ and reference y_{ref} ones, and tracking and model errors $e_{trk}(k), e_m(k)$ are shown. First 2 plots are with disturbance rejection and the others without ($u_d(k) = 0\,\text{N}$). The presence of parametric and disturbance uncertainties does not lead to high tracking and model errors, but in absence of disturbance rejection position $\hat{\xi}(k)$ estimation cannot reach the steady state tracking conditions (2 or $-2\,\text{m}$), and there is constantly an error of some centimetres. Instead with disturbance rejection $e_{trk} \approx 0\,\text{m}$ for position at steady-state conditions.

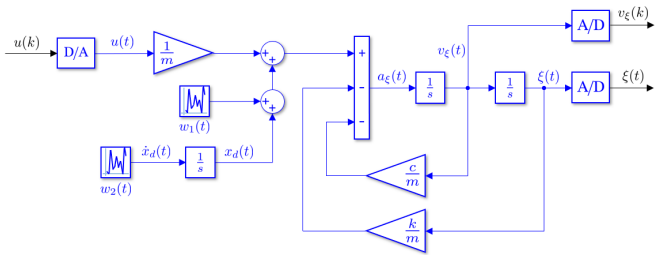


Figure 2.4: Mass spring damper fine model

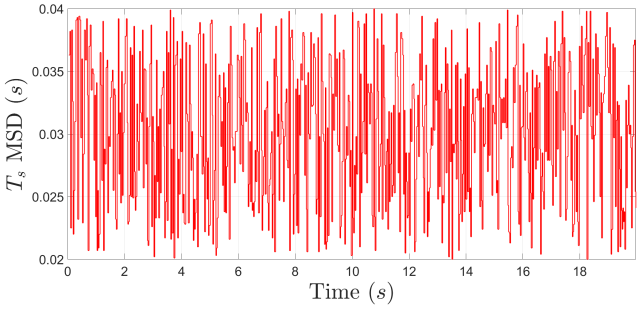


Figure 2.5: MSD variable Timestamp

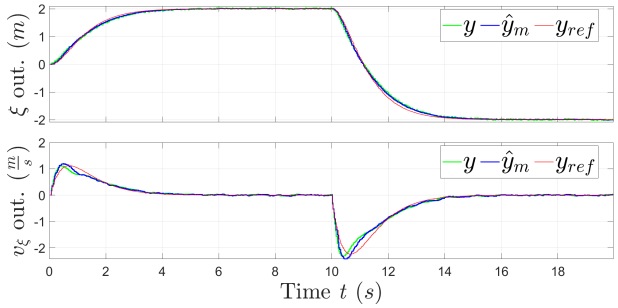


Figure 2.6: MSD $y, y_{ref}, \hat{y}_m$ - Disturbance rejection

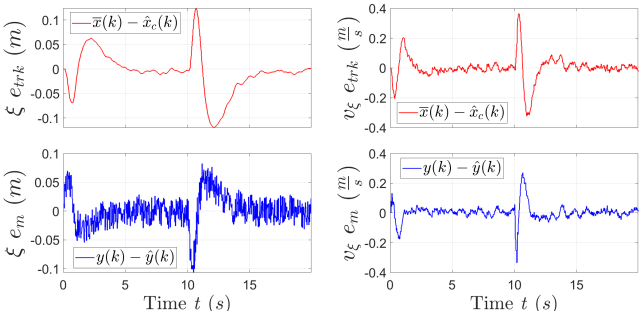


Figure 2.7: MSD tracking e_{trk} and model e_m errors - Disturbance rejection

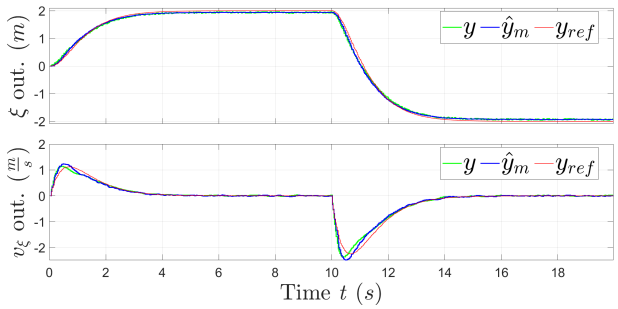


Figure 2.8: MSD $y, y_{ref}, \hat{y}_m$ - No disturbance rejection

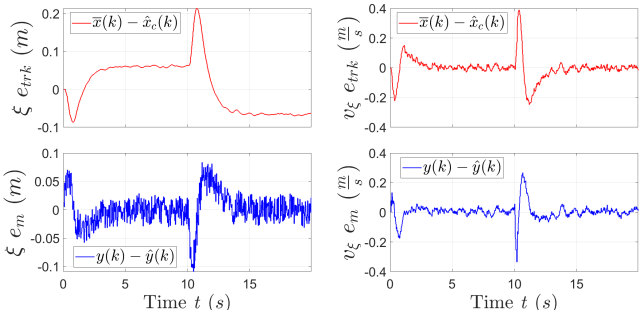


Figure 2.9: MSD tracking e_{trk} and model e_m errors - No disturbance rejection

Chapter 3

Robot GoPiGo3

For the thesis a differential drive (DD) robot car was considered. Among the possible choices it was selected one with quite precise encoders to measure wheel angular positions and, by differentiation, the wheel motor speeds. Dexter Industries company produces many kind of robots, mainly for educational purposes, and it offers an already build DD robot with 2 Magnetic Hall encoders called GoPiGo3, which is suitable for the purpose. Informations about it can be found in company website, [17].

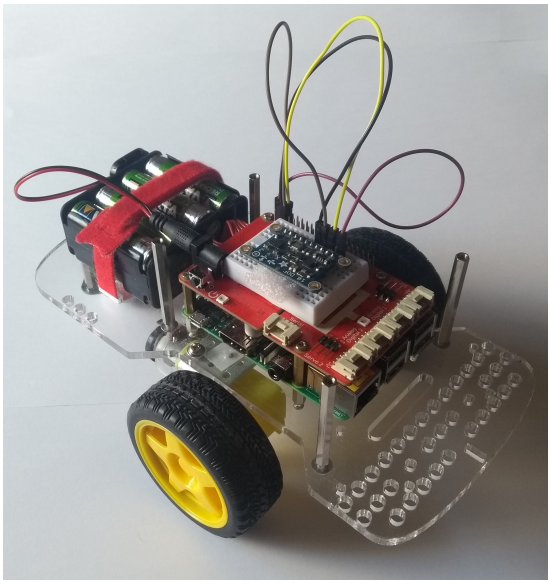


Figure 3.1: Front angle view GoPiGo3 DD robot

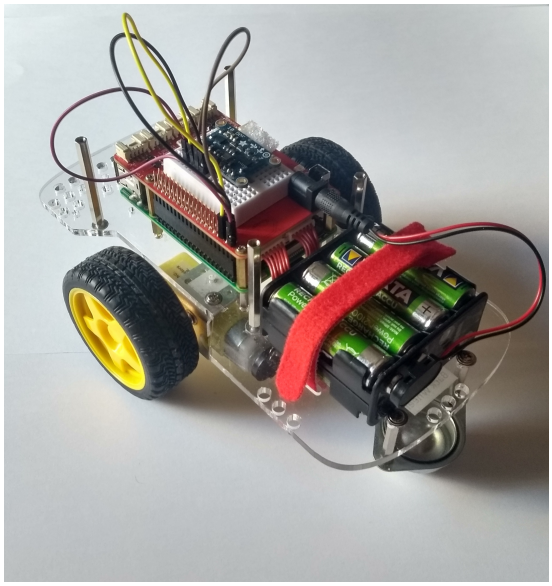


Figure 3.2: Back angle view GoPiGo3 DD robot

3.1 Construction properties

GoPiGo3 has two driving wheels and a caster wheel for equilibrium. It's a differential drive robot, it means that the wheels are driven independently from each other. It's mainly composed by:

- **2 direct current (DC) motors**: They move the wheels. The rotational output speed is reduced by a gearbox (protected by a plastic box) with has a ratio of $N = 120$, hence the motor speed is reduced by 120 times at load side.
  By construction they can range until a maximum voltage of $12\,\mathrm{V}$. In practice this voltage is lower because it depends on the supply (battery pack available voltage or a direct current power supply).

Motors are connected with main board using JST-XH 6 pin connectors.
Since no datasheet is given for the motors and gearbox inside, parameters were estimated using parameter identification (cfr. Section 5.1).

- **Magnetic encoders**: The motors are both equipped with 2 Hall Magnetic encoders with 6 pulse counts per rotation, to measure angular position (and speed by differentiation) of the wheels connected: with 120:1 motor gear reduction we obtain a total of 720 pulses per wheel rotation, which means an angular resolution of $0.5°$.

- **Main HW board**: It contains all HW elements to control the motors and other connection ports (I^2C, SPI, GPIO) for additional I/O devices. This board can be connected with a Raspberry Pi board directly using their I/O pins.
  Specifically for the motors the following components are available:

  - H-bridge drivers: They allow run the motors in both directions (clockwise and anticlockwise) and making them accelerating or decelerating.

  - PWM square wave generator: To generate the desired DC voltage for the motors a PWM modulation is used. It basically consists in generating a square wave with variable period (duty-cycle), so that we can control the average output voltage.
    PWM square wave in this case has $0 - V_{max}$ voltage amplitude, where V_{max} depends on the maximum voltage supply.
    The generated square wave was measured with an oscilloscope, with the probes across the 2 motor terminals (in parallel connection) imposing a known average voltage. PWM with 50% duty cycle (around 6 V) is shown in Figure 3.3.

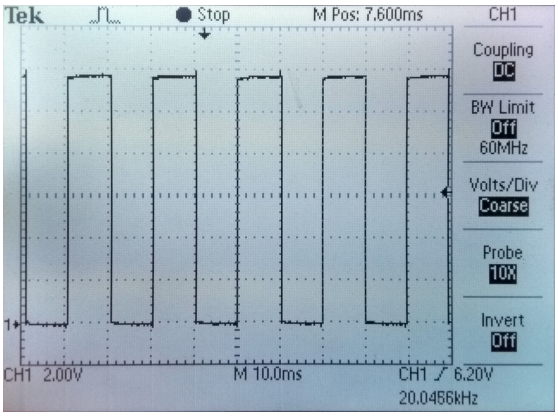


Figure 3.3: GoPiGo3 DC motor measured PWM square wave with 50% duty cycle

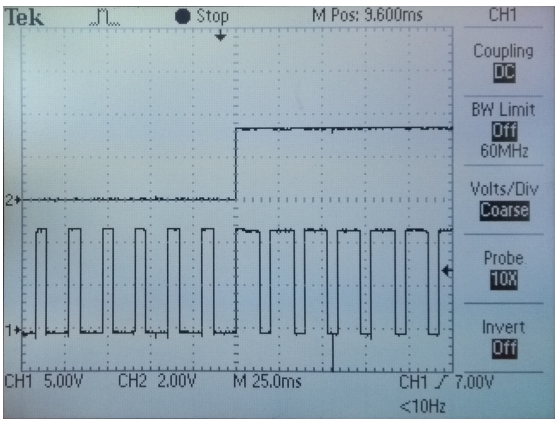


Figure 3.4: GoPiGo DC motor PWM duty cycle change - From 4 V to 8 V average voltage

Instead in Figure 3.4 a duty cycle change is imposed in one of the motors. The above signal shows a command in one of GPIO pins in the board, passing from LOW (0 V) to HIGH (3.3 V) logical condition when a change in duty cycle is caught: this pin is controlled by SW code, changing its logical condition when change in duty cycle of PWM is performed (again in SW code).
The signal below instead shows the PWM measured across motor terminals: after some time (half the horizontal scale) PWM duty cycle changes (HW change). It can be seen that a SW command to change

PWM (signal above) is converted almost immediately in HW command (change of PWM duty cycle), so it can be assumed that no delay exists in this operation.
In both figures it can be clearly seen that PWM signal is always generated at fixed time period, $T_{PWM} = 0.02\,\text{s}$.

- **Raspberry Pi Model 3B**: This board and its Raspbian OS are used to control all robot functionalities, at SW level.
  It is based on ARM Cortex-A53 Microprocessor 64-bit quad-core chipset with $1.2\,\text{GHz}$ clock speed, $1\,\text{GB}$ RAM, and allows Wi-fi and Ethernet connections. Raspbian OS is installed inside a microSD which acts as non-volatile memory (like Hard-disk). It has 40 I/O GPIO pins with different functionalities (for example there're I^2C reserved pins, with SDA and SCL connections), Ethernet and 4×USB ports, [18].
  Main robot board described before is directly connected with the Raspberry using some of I/O pins present.

- **Battery pack/power network supply**: The boards can be supplied using either a battery pack composed by $8 \times 1.2/1.5\,\text{V}$ AA batteries, or power supply converter to transform house network alternate $230\,\text{V}$ voltage into direct voltage until a maximum of $12\,\text{V}$. The second possibility is useful during writing of SW code phases when motors are still, in order to avoid consuming unnecessary battery power.

- **IMU**: An inertial measurement unit (IMU) is used to obtain robot orientation and position informations. It is produced by Adafruit company, model name "Adafruit 10-DOF IMU Breakout" [19], and consists in an embedded device composed by 3-axis accelerometer, 3-axis magnetometer (LSM303DLHC, measuring acceleration, linear acceleration and Earth magnetic field) and a 3-axis gyroscope (L3GD20, measuring angular speed). Also barometric and temperature sensors (BMP180) are present, but they're not important for thesis purposes.

3.2 Kinematic and dynamic properties

3.2.1 Geometric and mass parameters

For some control models built in the thesis, some geometric and mass informations are recovered.
To find some parameters (like total robot inertia I^t_{zz}), GoPiGo3 robot was reproduced using Autodesk Inventor CAD SW application (Figure 3.5). To define robot position in 3D space, Cartesian coordinates are considered using the symbol names ξ, ν, z (not x for longitudinal position because it is already reserved to EMC states): in Figure 3.6 are shown the Body RF (with origin in robot wheel axles, with the same distance from both wheels), subscript BF, and Inertial RF (origin fixed in one point in space), subscript IF.

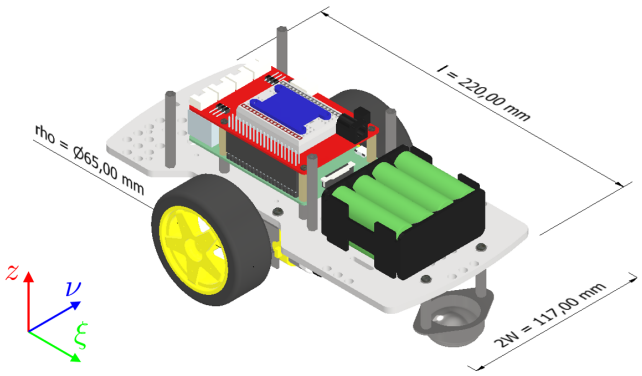


Figure 3.5: GoPiGo3 robot CAD model built with Autodesk Inventor

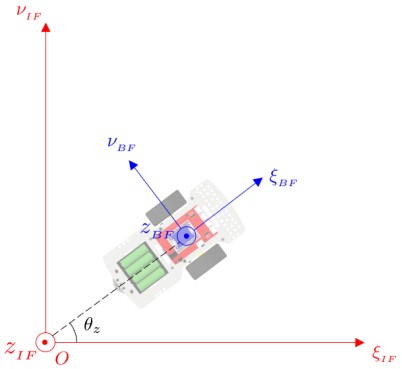


Figure 3.6: GoPiGo3 robot Body (blue) and Inertial (red) reference Frames

Main robot parameters (all approximated parameters are designated with "hat"):

	Symbol (meas. unit)	Value	Description
Geometric properties	W (m)	0.0585	*Half robot width*
	l (m)	0.2200	*Robot length*
	ρ (m)	0.0325	*Wheel radius*
	t (m)	0.0280	*Wheel width*
	$\hat{d}$ (m)	0.0200	*CoM distance from wheels axle*
Mass properties	m_w (kg)	0.0320	*Wheel mass*
	m_{batt} (kg)	0.2480	*Battery pack mass*
	m_b (kg)	0.6160	*Main body + battery pack masses*

Table 3.1: GoPiGo Differential Drive Robot parameters

Starting from these properties, total mass and inertia can be recovered. Total mass M_T is simply mass of the body plus battery pack and wheels:

$$M_T = m_b + 2m_w = 0.68\,\text{kg} \tag{3.1}$$

The main body without wheels can be approximated as a parallelepiped with length l and width $2W$, for which the moment of inertia around z axis can be computed as [11]:

$$\hat{I}^b_{zz} = m_B \frac{(2W)^2 + l^2}{12} = 0.0032\,\text{kg}\,\text{m}^2$$

where m_B is the main body mass. The other axes inertias are not needed.
Inertia tensor for the wheels is the diagunal matrix below:

$$\hat{I}_W = \begin{bmatrix} \hat{I}^w_{\xi\xi} & 0 & 0 \\ 0 & \hat{I}^w_{\nu\nu} & 0 \\ 0 & 0 & \hat{I}^w_{zz} \end{bmatrix} = \begin{bmatrix} \frac{m_b(3\rho^2+t^2)}{12} & 0 & 0 \\ 0 & \frac{m_w(R^2_{in}+R^2_{out})}{2} & 0 \\ 0 & 0 & \frac{m_b(3\rho^2+t^2)}{12} \end{bmatrix}$$

Since the wheel can be considered roughly as hollow cylinder it has external radius $R_{out} = \rho = 0.0325\,\text{m}$ and

internal radius (approximately) $R_{in} = \rho - 0.01 = 0.0225\,\mathrm{m}$. Numeric values for each wheel inertia are:

$$\hat{I}^w_{\xi\xi} = \hat{I}^w_{zz} = 1.0541\mathrm{e}{-5}\,\mathrm{kg\,m^2}$$
$$\hat{I}^w_{\nu\nu} = 2.5000\mathrm{e}{-5}\,\mathrm{kg\,m^2}$$

Knowing body and wheels inertia around z axis, the following formula for total inertia can be used:

$$\hat{I}^t_{zz} = \hat{I}^b_{zz} + m_b d^2 + 2\hat{I}^w_{zz} + 2m_w W^2 = 0.0037\,\mathrm{kg\,m^2} \tag{3.2}$$

It can be seen immediately that $\hat{I}_T$ and M_T mainly depends from main body platform and battery pack, and the wheels have very little impact because of the their low mass.
the terms $2m_wW^2$ and m_bd^2 in Equation (3.2) derives from Huygens-Steiner theorem (taken from [11]), which is resumed below:

Theorem 3.1 (Huygens-Steiner) *Inertia moment of a mass body m with respect to an axis placed at a distance from the same body center of mass (CoM) is:*

$$I = I_c + ma^2 \tag{3.3}$$

where I_c is inertia moment of body with respect to an axis parallel to body and passing through CoM.

In this case, the total inertia is computed using the rotational axis passing through the total robot CoM, which is slightly moved towards the caster wheel (it's not exactly on the wheel axles, condition verified by position of battery pack). Indeed it has a distance of W in lateral direction and d in longitudinal direction wrt origin of Reference BF in Figure 3.6. For this reason there are the terms m_wW^2 and m_bd^2. Caster wheel is considered as part of the main body.

3.2.2 Total robot inertia verification

Since GoPiGo3 robot has not exactly a parallelepiped shape, the inertia value in Equation (3.2) must be verified. This is done by building a model using Autodesk Inventor. Every part was created alone, then assembled. The interesting feature of this program is to compute total mass and geometric properties of the assembly (included the inertia and CoM), using a tool in the program (iProperties). There is some error inside these computations, but can be considered very small. Computed values are:

	Symbol (meas. unit)	Value	Description
Inertia properties	I^t_{zz} $(\mathrm{kg\,m^2})$	0.002235	*Total robot inertia, z axis*
	$I^w_{\nu\nu}$ $(\mathrm{kg\,m^2})$	22.9e−5	*Wheel robot inertia, ν-axis*
CoM properties	d (m)	0.0472	*CoM distance from wheel axles*

Table 3.2: GoPiGo properties using iProperties Inventor tool

The CoM distance d really depends on battery pack position, since they have important weight: it was estimated $\hat{d} \approx 0.02\,\mathrm{m}$ but in reality is near $0.05\,\mathrm{m}$. This is not a significant problem since the estimated total inertia $\hat{I}^t_{zz}$ doesn't differ so much from the real one I^t_{zz} computed with Inventor.
The real total inertia can be used to build fine model when robot orientation will be controlled by EMC, and to compute robot torque starting from wheel speeds (see Chapter 10 and next subsection).

3.2.3 From robot wheel speeds to robot torque/force relation

As we will see in Chapter 11 and Chapter 14, at least for MIL simulations is important to recover a relation between wheel angular speeds and robot total torque/force.
Starting from DC motor (and wheels) angular speeds $\omega_L = \dot{\phi}_L$ and $\omega_R = \dot{\phi}_R$, the angular accelerations $\dot{\omega}_L = \ddot{\phi}_L$ and $\dot{\omega}_R = \ddot{\phi}_R$ can be obtained with a simple derivative. Instead to obtain robot torques/forces from wheel angular accelerations some considerations need to be done.
First, we can assume that wheels have pure rolling motion, which means that contact point with ground is time by time equal to zero. In other words, in this conditions the wheels never slip.
Second, there are 3 contact points with the ground: 2 wheels and caster wheel. This last one does not exert horizontal forces to the ground (since it has negligible rolling inertia), so all the mass is driven by the 2 moving wheels.

- Each wheel moves half of total mass M_T.

- Pure rolling condition, in presence of torque moving the wheel, means that friction force developed at ground it's approximately equal to force moving the entire wheel, applied to wheel CoM.

By making forces and moments balance equations, for only 1 wheel (reference is [11]):

$$\begin{cases} \mathbf{R} + mg = m\mathbf{a}_{\mathbf{CM}} & \text{Vectorial Forces} \\ \boldsymbol{\tau} + \mathbf{r} \times \mathbf{f} = I^w_{\nu\nu} & \text{Vectorial Moments} \end{cases}$$

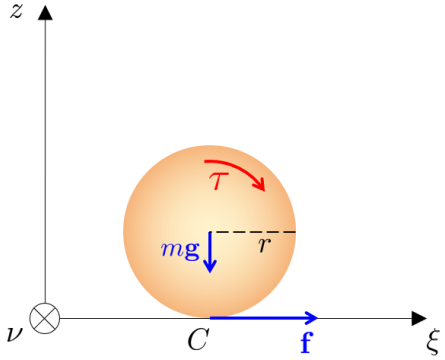


Figure 3.7: Forces and Moments acting on a generic wheel with pure torque motion

Source: [11] modified

$I^w_{\nu\nu}$ in this case is the wheel inertia around ν-axis in Table 3.2.
Balance equations can be decomposed in vertical and horizontal directions:

$$N = mg \quad , \quad f = ma_{CM} \quad , \quad \tau - rf = I^w_{\nu\nu}\frac{a_{CM}}{r} \tag{3.4}$$

From the last equations, friction force (and total force moving the wheel) is:

$$f = \frac{\tau}{r\left(1 + \dfrac{I^w_{\nu\nu}}{mr^2}\right)}$$

Ground contact point C velocity can be expressed as $\mathbf{v_c} = \mathbf{v_{CM}} + \boldsymbol{\omega_z} \times \mathbf{r}$, but since the motion is pure rolling $\boldsymbol{v_c} = 0\,\mathrm{m/s}$, the first equation reduces to $\mathbf{v_{CM}} = -\boldsymbol{\omega_z} \times \mathbf{r}$. In modulus, $v_{CM} = \omega_z r$. Derivating we obtain $a_{CM} = \alpha r$, where α is the wheel angular acceleration.

Combining with f in Equation (3.4), we obtain:

$$f = mr\alpha \tag{3.5}$$

To pass form wheel torque speeds τ_i to angular accelerations α_i the following relation can be used, recovered from the last 2 expressions:

$$\tau_i = m\rho^2 \left(1 + \frac{I^w_{\nu\nu}}{m\rho^2}\right) \alpha_i \tag{3.6}$$

Each wheel generates a robot torque which changes its vertical angle θ_z and angular velocity $\dot{\theta}_z = \omega_z$. In general the vectorial sum of torque gives the total torque exerted on robot $\boldsymbol{\tau_T}$:

$$\boldsymbol{\tau_T} = \sum_{i=1}^{2} \boldsymbol{\tau_i} = I^w_{zz} \ddot{\boldsymbol{\theta}}_{\boldsymbol{z}} = \boldsymbol{\tau_R} + \boldsymbol{\tau_L} = \boldsymbol{F_R} \times \boldsymbol{W_R} + \boldsymbol{F_L} \times \boldsymbol{W_L} = (F_R - F_L)\, W \tag{3.7}$$

where $\boldsymbol{W_R^T} = [0, 0, W]$, $\boldsymbol{W_L^T} = [0, 0, -W]$ are the vector distances of wheels from body CoM axis, which coincides with half robot width.
Combining Equation (3.5) and Equation (3.7) we obtain:

$$\ddot{\theta}_z = \underbrace{\frac{mrW}{I^t_{zz}}}_{b^\tau_{\alpha 1}} (\alpha_R - \alpha_L) \tag{3.8}$$

For robot specific case $m = M_T/2$ and $r = \rho$.
The last equation can be split in 2 linear ODE, with states $\theta_z, \dot{\theta}_z$:

$$\begin{cases} \dot{\theta}_z = \dot{\theta}_z \\ \ddot{\theta}_z = b^\tau_{\alpha 1} (\alpha_R - \alpha_L) \end{cases} \tag{3.9}$$

Instead to obtain total robot force F_T from wheel angular speeds ω_R, ω_L relation is very simple. The general Newton equation for total robot force $\boldsymbol{F_T}$ is:

$$\boldsymbol{F_T} = \sum_{i=1}^{2} \boldsymbol{F_i} = M_T \ddot{\boldsymbol{\xi}} = \boldsymbol{F_R} + \boldsymbol{F_L} \tag{3.10}$$

Using Equation (3.5) friction force can be related with left F_L and right F_R longitudinal robot forces acting on wheels, considering $m = M_T/2$ and $r = \rho$. Hence the relation between total robot force and wheels angular speeds is:

$$\ddot{\xi} = \underbrace{\frac{\rho}{2}}_{b^f_{\alpha 1}} (\alpha_R - \alpha_L) \tag{3.11}$$

And split in 2 linear ODE the equations are:

$$\begin{cases} \dot{\xi} = \dot{\xi} \\ \ddot{\xi} = b^f_{\alpha 1} (\alpha_R - \alpha_L) \end{cases} \tag{3.12}$$

To understand if the last Equation (3.8) and Equation (3.11) are coherent or not, a comparison with expression obtained using Differential Drive Lagrangian equations is considered (to pass form wheel torque/forces to longitu-

dinal/angular accelerations, Equation (3.6) is used):

$$\begin{cases} \ddot{\theta}_z = \underbrace{\dfrac{\rho}{2W}\dfrac{1}{I^w_{\nu\nu}+2I^t_{zz}\left(\frac{\rho}{2W}\right)^2}}_{b_\tau}(\tau_R-\tau_L) = b_\tau(\tau_R-\tau_L) \quad \Rightarrow \quad \ddot{\theta}_z = \underbrace{b_\tau\dfrac{M_T}{2}\rho^2\left(1+\dfrac{I^w_{\nu\nu}}{\frac{M_T}{2}\rho^2}\right)}_{b^\tau_{\alpha 2}}\alpha_i \\ \ddot{\xi} = \underbrace{\dfrac{\rho}{2}\dfrac{M_T^{-1}(I^w_{\nu\nu})^{-1}}{M_T^{-1}+2(I^w_{\nu\nu})^{-1}\left(\frac{\rho}{2}\right)^2}}_{b_f}(\tau_R+\tau_L) = b_f(\tau_R+\tau_L) \quad \Rightarrow \quad \ddot{\xi} = \underbrace{b_f\dfrac{M_T}{2}\rho^2\left(1+\dfrac{I^w_{\nu\nu}}{\frac{M_T}{2}\rho^2}\right)}_{b^f_{\alpha 2}}\alpha_i \end{cases} \tag{3.13}$$

The expressions are taken from [8], eqs (45) and (48). To make the direction of 2 wheel torque speeds equal, the equations are slightly modified (since in paper torques are in opposite direction). The procedure to find them is well explained in paper and it was verified its correctness. Main passages can be resumed as:

- Pure rolling motion condition (no velocity between ground and wheel contact point) results in non-holonomic constraints, basically they are the kinematic equations rewritten in a different way. Considering as generalized coordinates $\mathbf{q} = [\xi, \nu, \theta_z, \phi_R, \phi_L]$:

$$\begin{bmatrix} \cos\theta_z & \sin\theta_z & 0 \\ -\sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} \dot{\xi} \\ \dot{\nu} \\ \dot{\theta}_z \end{bmatrix} = \frac{\rho}{2}\begin{bmatrix} \dot{\phi}_R+\dot{\phi}_L \\ \dot{\phi}_R+\dot{\phi}_L \\ \dfrac{\dot{\phi}_R-\dot{\phi}_L}{W} \end{bmatrix}$$

  They can be rewritten using the generalized coordinates derivative $\dot{\mathbf{q}}$:

$$C^T(\mathbf{q})\boldsymbol{\lambda} = \begin{bmatrix} \cos\theta_z & \sin\theta_z & 0 & \frac{\rho}{2} & \frac{\rho}{2} \\ -\sin\theta_z & \cos\theta_z & 0 & \frac{\rho}{2} & \frac{\rho}{2} \\ 0 & 0 & 1 & \frac{\rho}{2W} & -\frac{\rho}{2W} \end{bmatrix}\dot{\mathbf{q}} = 0$$

  The last term can be added to Lagrangian equations:

$$\begin{aligned} &\frac{d}{dt}\left[\frac{\partial\mathcal{L}(\mathbf{q},\dot{\mathbf{q}})}{\partial\dot{\mathbf{q}}}\right] - \frac{\partial\mathcal{L}(\mathbf{q},\dot{\mathbf{q}})}{\partial\mathbf{q}} - C(\mathbf{q})^T\boldsymbol{\lambda} = \boldsymbol{\tau} \\ &M(\mathbf{q})\ddot{\mathbf{q}} + B(\mathbf{q},\dot{\mathbf{q}}) - C^T(\mathbf{q})\lambda = \boldsymbol{\tau} \end{aligned} \tag{3.14}$$

  $\boldsymbol{\lambda}$ terms are called *Lagrangian multipliers*.

- Lagrangian multipliers must be found. Since computation can be difficult using the complete Lagrangian equations, some assumptions are made: $d = 0\,\mathrm{m}$ (CoM is considered passing through the wheel axles) and $\theta_z = 0°$ (which means observing coordinate system to be parallel to the body-fixed coordinate system).
  Using these simplifications, starting from $C^T(\mathbf{q})\boldsymbol{\lambda} = 0$ and Equation (3.14), the expression for λ (simplified) will be:

$$\boldsymbol{\lambda} = -\left[C(\mathbf{q})M(\mathbf{q})^{-1}C^T(\mathbf{q})\right]^{-1}\left[C(\mathbf{q})M(\mathbf{q})^{-1}\boldsymbol{\tau} + \dot{C}(\mathbf{q})\right]$$

- $\boldsymbol{\lambda}$ can be substituted in Lagrangian equations to find $\ddot{\theta}_z$ and $\ddot{\xi}$ expressions in Equation (3.13).

$b^\tau_{\alpha 1}$ and $b^\tau_{\alpha 2}$ in Equation (3.8) and Equation (3.13) can be found knowing the inertia and mass parameters computed

before, leading to the following results:

$$b^{\tau}_{\alpha 1} = 0.2892 \quad , \quad b^{\tau}_{\alpha 2} = 0.2885$$

The 2 values are very near, hence they can be considered quite reliable.
From a deeper insight, Equation (3.8) is exactly equal to Equation (3.13) if we consider $I^{w}_{\nu\nu} = 0$.

The same can be done for longitudinal acceleration equation, comparing $b^{f}_{\alpha 1}$ and $b^{f}_{\alpha 2}$ in Equation (3.11) and Equation (3.13). After substitutions their values are the same:

$$b^{f}_{\alpha 1} = 0.0163\,\mathrm{m} \quad , \quad b^{f}_{\alpha 2} = 0.0163\,\mathrm{m}$$

Again Equation (3.11) and Equation (3.13) are exactly the same if we consider $I^{w}_{\nu\nu} = 0$.

3.2.4 Kinematic model

In some control model MIL and RHIT tests is important to understand the position and orientation of the robot in space. To this aim *Kinematic equations model* for DD robots can be exploited: they are equations which relates robot 2D position and vertical yaw angle $[\xi, \nu, \theta_z]$ with the left and right wheel angular positions ϕ_L and ϕ_R, using geometric properties only (ρ and W):

$$\begin{bmatrix} \dot{\xi} \\ \dot{\nu} \\ \dot{\theta}_z \end{bmatrix} = \begin{bmatrix} \frac{\rho}{2}\cos\theta_z & \frac{\rho}{2}\cos\theta_z \\ \frac{\rho}{2}\sin\theta_z & \frac{\rho}{2}\sin\theta_z \\ \frac{\rho}{2W} & -\frac{\rho}{2W} \end{bmatrix} \begin{bmatrix} \dot{\phi}_R \\ \dot{\phi}_L \end{bmatrix} \tag{3.15}$$

This is a standard model for DD robots, which can be easily found in literature (like in [7]). Making an integration of the equations robot position in 2D planar plane and orientation can be easily found, with good approximation.

If robot follows a circular trajectory of known mean radius r_m and total linear speed v, left and right motor angular speeds $\omega_L = \dot{\phi}_L$ and $\omega_R = \dot{\phi}_R$ can be easily found:

$$\begin{cases} v_L = (r_m - W)\dot{\theta}_z \\ v_R = (r_m + W)\dot{\theta}_z \\ v = r_m\dot{\theta}_z \end{cases}$$

From the desired total velocity v, $\dot{\theta}_z$ can be recovered, and from that the motor angular speeds ($\dot{\phi}_L = v_L/\rho$ for left wheel and $\dot{\phi}_R = v_R/\rho$ for right wheel):

$$\dot{\theta}_z = \frac{v}{r_m} \quad \Rightarrow \quad \begin{cases} \dot{\phi}_L = \dfrac{(r_m - W)\dot{\theta}_z}{\rho} \\ \dot{\phi}_R = \dfrac{(r_m + W)\dot{\theta}_z}{\rho} \end{cases} \tag{3.16}$$

Chapter 4

Control models software testing - General settings

To verify control models using EMC technique, 2 main kind of tests are performed:

- **Model-in-the-Loop (MIL) tests**: These kind of tests are used for EMC before making practical implementation in GoPiGo3 robot. Both robot plant and control scheme run in simulation using MathWorks Simulink application.
  It's obtained a preliminary analysis of the control model behaviour, which is then modified continuously comparing with SW implementation in the real robot (RHIT).

- **Robot Hardware Implementation Tests (RHIT)**: The control model is generated in *C++ language* SW code and directly implemented in Raspberry robot board, connecting with real plant devices.

Usually SW tests requires intermediate passages, which are not considered in this field:

- **Software-in-the-Loop (SIL)**: Is commonly performed after MIL, which consists in running again both plant and control model using simulation language (like Simulink), but control model is in form of generated code.

- **Processor-in-the-Loop (PIL)**: Executed after SIL, this test consists in running the plant in simulation (Simulink) and the control model in the target system (Raspberry).

- **Hardware-in-the-Loop (HIL)**: Precedes real implementation of both the 2 parts: controller SW code run in target system and plant is co-simulated usually in a rapid prototyping HW, which simulates it in real-time.

SIL and PIL are used to verify generated SW code for control models, first in simulation and then in target system: since EMC block is mainly composed by linear DT SS equations and requires simple computations, the overall generated SW code is very simple and short, thus these verification tests can be skipped.
HIL is not needed because the plant, the GoPiGo3 robot, is directly usable in almost every place for its little size. It's not a vehicle suspension which requires the entire vehicle to be drive for controller tests.
In the next sections the main passages to perform the 2 tests are explained, making reference to tools and SW/HW applications needed.

4.1 MIL configuration

Main SW applications used to perform MIL tests are MathWorks MATLAB and Simulink. They're composed of many toolboxes for different purposes, some needed for thesis work. For example MATLAB Symbolic Math toolbox is used to find control loop eigenvalues computing symbolically pole-placement expressions, to be used next in EMC.
In Simulink to build DT systems, state-space equations are preferred instead of Transfer functions, thus ***Matlab-functions*** blocks are better than LTI systems blocks and others. They also are useful when variable DT control systems are considered, where timestamp value always changes in DT system matrices at every time step: Matlab-functions allows to do that, which is impossible using elementary blocks.

4.1.1 Variable timestamp

The most important thesis objective is to verify EMC in presence of asynchronous sampling time, hence a way to implement variable Timestamps in simulation is needed. Since no already built Simulink block is available to this aim, a solution is find by the author.
In Simulink are available ***Stair generator blocks***, which allow to generate staircase functions (i.e.piecewise constant functions): they are used to produce a simulated PWM signal like the ones of robot DC motors.
First a signal (data + time) vector of random values in a predefined range is built, in MATLAB is sufficient to use a while conditional loop implementing the following recursive equations:

$$T_s(k) = r\epsilon_{min} + \epsilon \qquad \text{Timestamp data}$$
$$T(k) = T(k-1) + T_s(k) \qquad T(0) = 0\,\mathrm{s} \qquad \text{Timestamp time}$$

r is a uniform random number ranging 0–1, ϵ_{min} = minimum range value, ϵ = Timestamp range, $k \geq 0$.
In an alternative way, when a RHIT is already performed, the variable Timestamp implemented by SW code can be directly loaded in MATLAB to be used in simulations: this is useful to make comparison between RHIT and MIL.
Next for every sampled Timestamp data, another signal vector is built. Data values are 0 or 1: first Timestamp $T_s(k)$ value corresponds to 0, the second to 1, the third to 0, and so on. The time values are the same of $T(k)$. Resulting signal is like a Timestamp PWM, which can be directly loaded in staircase generator in Simulink.

In order to control the EMC in DT, in Simulink exist ***Triggered sub-systems***: in these sub-systems every block inside run only when an external trigger function edge is rising, falling or either, hence determining a DT timing. Trigger function in this case is exactly the timestamp PWM signal built before, where rising edges are passages from 0 to 1, and falling edges vice-versa. Putting triggered sub-system running for either edges, we obtain almost the same variable Timestamp desired.
Obviously stair generator edges in Simulink are really dependent on simulation time solver, because it's not possible to obtain an immediate sharp rising or falling of a signal function: for this reason it's better to use a Fixed/Variable Solver with low simulation time (for example $2\mathrm{e}{-4}\,\mathrm{s}$ is sufficient and simulations are not so long)

4.2 RHIT configuration

4.2.1 SW Tools and applications

Raspberry remote control applications

To make RHIT tests it's needed to work with Raspbian OS of Raspberry Pi board. Access and control it remotely using another OS is a more comfortable solution, for example using Windows in another Laptop.
There exists different SW applications to manage remotely other computer systems:

1. **PuTTY client**: This application client emulates the prompt terminal used by Raspbian OS remotely, using different connection protocols like SSH (Secure SHell, which uses an encrypted connection) or Telnet (no encrypted connection). Website is [20].
   This is not the best solution to develop the SW code for RHIT tests, because using a terminal many OS functionalities come out to be very uncomfortable, like administrate folders and files or write the code (an IDE application to write code is not available working in a terminal shell).

2. **VNC (Virtual Network Computing) Viewer and Server**: These 2 applications allow to control remotely a computer system OS with another one, but wrt PuTTY we can have access not only to commands prompt terminal but also to GUI Desktop. This is a huge convenience, especially when developing the code. Again security is guaranteed by using end-to-end connections encrypted using 128-bit AES, 2048-bit RSA keys (website is [21]). Basically VNC Viewer is the application to be installed in the device we want to control from (Windows for example), and VNC Server the one to be used in the device we want to control (Raspberry).

The only drawback controlling Raspberry remotely with a Desktop GUI may be running an RHIT test SW code: the presence of graphical interface may require other applications active at the same time of code program, and since the control models need to be stepped with a precise Timestamp this condition can lead to lags and timing errors.
If this is true PuTTY application may be better for running the code, since it does not require a Desktop GUI to be controlled. In reality it was experimented that GUI has very little influence on code timing requirements, and VNC remains the best solution.

Cross-platform IDE

Code development phases can be speeded up by using an Integrated development environment (IDE), which is an application program to write SW code and make cross-compilation flow of source/header files automatically.
Cross compilation represents all operations needed by SW compiler and linker to create executable files from source/header files. A schematic is in Figure 4.1: from source .c/.cpp and header .h/.hpp files a compiler converts them into object .o files: inside them there're all SW code present in .c/.cpp. Then the linker must connect all object files and other required static .a and dynamic .so libraries, to obtain a unique executable file. Executables can have different extensions (.out in older UNIX versions, now subtituted by .elf).
CodeBlocks SW application is finally selected as IDE for the purpose.

Simulink Embedded Coder

To pass from MIL to RHIT using Raspberry board, a tool for generating a SW code is needed. Simulink Embedded Coder is used to convert the Simulink EMC model into SW code. Basic settings are:

- Simulink Embedded Coder allows only 2 programming languages for generated code: C and C++. Since

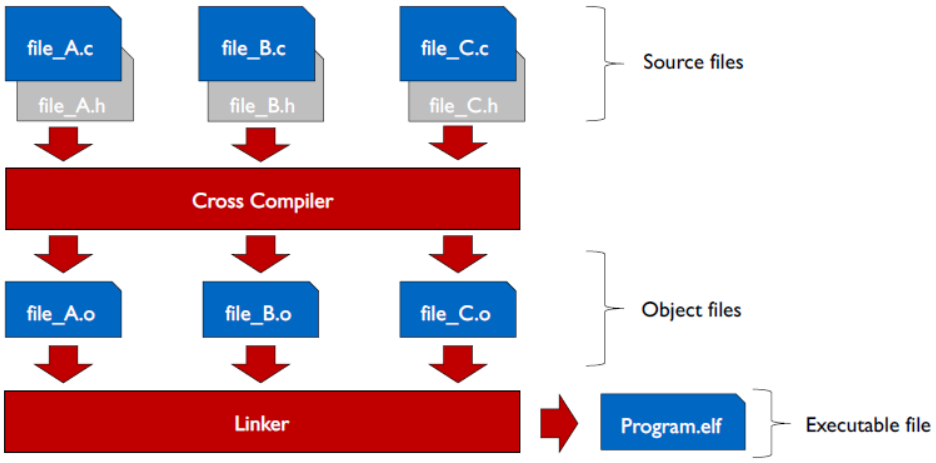


Figure 4.1: Cross compilation flow

Source: [13]

GoPiGo3 main library is already written in C++, the same choice is preferred for control model code. Another reason is that Object-oriented programming classes makes the code shorter and more understandable.

- Generated code is optimized, in the sense that part of the code is reduced in length combining variables, avoiding if possible conditional statements like for loops etc. This helps improve the speed performances when running the code, important in control field.
  In Simulink many options can be selected for optimization, for example one of them is the *"Loop unrolling threshold"* setting, which allows to decide the minimum width for a "for" loop: if the number of loops is lower than this width value, generated code tries to combine definitions of variables and avoid the presence of for loop.

- In Simulink settings among possible specific system target files it is chosen the one called *"ert.tlc"*: it means Embedded real-time target (with .tlc referring to the compiler extension used by coder tool), which optimizes for smaller memory model and execution speed. It's a good choice for Raspberry board.

- Raspberry board HW is based on ARM Cortex Microprocessor, hence also the generated code must be specific for this HW. In Simulink there's a setting to change device microprocessor vendor as desired.

Generated SW code results files are subdivided as:

- **ert_main.cpp** This is the main source file, it contains a main function with other sub-functions to run the code.

- **<model_name>.cpp** (model_name refers to the name given arbitrary to the model) This is the source file containing all informations about the generated Simulink model. It is subdivided in 3 class member functions (we can name the class as "Model"):

  - Model::initialize() Here the model parameters can be declared and initialized. Because the generated models are for control purposes, usually eigenvalue placement is used to control the system dynamic behaviour and stability. This member function is useful to initialize these ones, but also model states

and sampling time to zero.
Is not mandatory to use it, but is preferable since allows the code to be more clean and understandable.

- Model::step() This is the member function containing the Simulink control model translated in SW code: control inputs and sampling times enter as input variables and estimated states from EMC exit as output variables.
It's called at every discrete sampling time step by calling it from ert_main.cpp source file.

- Model::terminate() After the last execution of model, variables memory can be clean-up using this member function. Again its use is not mandatory, but preferable.

C++ classes usually have constructor and destructors to initialize the parameters (especially when one wants private parameters), but in this case they are substituted by member functions initialize() and terminate().

- **<model_name>.h** This header file (.h or .hpp file extension is indifferent, C++ works with both) contains the main libraries used by the model class and the declaration of the class member functions, constructors, destructors, states and variables and model-specific data types, to be used in .cpp source file. Usually the public states and variables used by the model are saved into structures, to avoid being declared as global variables (global variables need to be avoided if possible, since they can be accessed by every functions and sometimes modified without noticing).

- **rtwtypes.h** It translates Simulink specific data-types into HW-specific data-types, with different names in order to be distinguished: for example *"double"* in Simulink is translated into *"real_T"* data-type. This header is called by model_name.h file.

- **<model_name_data>.cpp** Sometimes the main model parameter names passed in Simulink from MATLAB workspace are saved in a different source file (designed by the _data suffix) as a structure.

4.2.2 Main SW libraries and functions

HW Timers

In order to step the model with a certain sampling time it is needed a quite precise HW timer: among the possibilities, considering that is not possible to use external HW timers, *signals* are used: a signal in UNIX OS platforms like Raspbian can be described as an asynchronous notification that an event occurred, and in this case it can call a signal handler to run a specific routine of functions, [22]. In particular the library header file used for the signal timers is called `signal.h`.
In this case the event is the expiration of a timer after an arbitrary elapsed time (our Timestamp), and the signal handler the functions needed to compute outputs of the control model and plant. After the expiration and function handler call, the timer is reset for a new step. Basic structure to run this kind of timers requires:

- Functions containing signal library functions to initialize and set the timer, for example taking as parameters its expire time decided by user and the number of signal (indeed there're a different number of signals to be used with different purposes, a part of them can be used freely by the user).

- Signal handler function, containing the functions we want to call. Every time a signal timer expires an

interrupt is generated and the functions inside signal handler are executed.

- A line of code function starting the signal timer, with parameters the signal number, expiration time and signal handler function.

In theory the signal handler function can be outside the main function, with the following simplified code flow:

```
#include <signal.h>
#include <sys/time.h>

void sig_handler ( int signo ) {
    // step functions to be called
}

void timer ( long int useconds ) {
    // initialize expiration time and start timers
}

int main() {
    signal (<SIGNAL NAME>, sig_handler );
    timer(useconds); // set the timer expiration in microseconds

    while(true){}

    return 0;
}
```

Listing 4.1: Signal timers SW code flow, interrupt mode

Signal library are old UNIX functions and one problem can be race-conditions if the signals are more than one: for example if the signal handlers share the same variables, one them can change the variable when the others don't want to, causing unexpected results.
For this reasons a better solution is to put in the signal handler only a flag (`volatile sig_atomic_t global_flag`), which can be only 1 or 0. Inside the main while function we put an if statement to verify value of the flag: if it is 1 enters the statement and run the code inside (containing the needed step functions), otherwise skip. At the end of if statement is needed to set `global_flag` = 0 to enter inside only once per timer expiration.
In practice the code is changed as:

```
#include <signal.h>
#include <sys/time.h>

volatile sig_atomic_t global_flag;

void sig_handler ( int signo )
{
    global_flag = 1;
}
void timer ( long int exp_time )
{
    // initialize and start timers
}
```

```
int main() {
    signal (<SIGNAL NAME>, sig_handler );
    timer(exp_time); // set the timer expiration in microseconds
    while(true){
        if (global_flag == 1) {
            // step functions to be called
            global_flag = 0;
        }
    }
    return 0;
}
```

Listing 4.2: Signal timers SW code flow, polling mode

This code can be considered running in polling condition, since it is needed to verify periodically the if statement, but the signals handlers are called in interrupt mode.
The only drawback of the last solution is that CPU is always used by the program, because every time the if statement needs to be verified. But apart from the main function to control the robot, no other processes are needed and this drawback sorts no effect.
This solution works with only 1 timer and it's based on the function `setitimer()` inside `timer()` above. To run more than 1 simultaneously, another function from standard C and signal libraries is needed to be called, `timer_settime()`.

Timestamp value

HW timers discussed before are based on expiration time which is the timestamp of DT models, hence is also needed a function to measure and verify it. In the next piece of code there is its implementation:

```
#include <sys/time.h>

struct timeval ts;

double timestamp() {
    gettimeofday(&ts, NULL);
    return (double)ts.tv_sec + (double)ts.tv_usec/1000000;
}
```

`gettimeofday(&ts, NULL)` is a subfunction that gets the seconds and μ-seconds since Epoch (which is the starting default date 1970-01-01 00:00:00 (UTC)) and save them into a structure called `ts`. Then they are returned in `timestamp()` function.
Note: `ts` can be casted to `double` only if there's a quite recent C++ compiler (for example in UNIX platforms if compiler is GCC, its version must be more recent than 7.1).

Variable Timestamp

One of the main thesis objective is to verify EMC using variable sampling time, because when control inputs are sent remotely the timestamp can be received with a certain delay.
Before making tests remotely (with a Networked Control system NCS), a simplification can be done to make variable sampling time tests: define in the code uniform random numbers in a predefined range, and add them to HW timers expiration time. If EMC is working with this simplification, it's almost sure that it can work with

a NCS too, because SW generated random timestamps present higher variations than remotely sended command input timestamps.
To generate random numbers the following piece of code can be implemented:

```
#include <stdlib.h>       // Library for srand, rand functions

srand (time(NULL));       // Random seed with current time number
double r;
double HI = 20;           // Highest value random number r (in milliseconds)
double LO = 0;            // Lowest value random number r (in milliseconds)
r = LO + static_cast <double> (rand()) / static_cast <double> (RAND_MAX/(HI-LO));
```

Indeed since uniform random number u_r using `rand()` function is limited from 0 to $RAND_{MAX}$ (the last is the maximum value returned by `rand()` function), to change the range from LO to HI one must do:

$$r = LO + \frac{u_r\,(HI - LO)}{RAND_{MAX}}$$

GoPiGo3 libraries

Dexter industries GoPiGo3 robot vendor provided a GitHub repository containing component datasheets, SW libraries and example codes in different languages, [23]. Among them `GoPiGo3.cpp` and `GoPiGo3.h` C++ source and header files are available, containing main classes to control GoPiGo3 robot functionalities. Source and header files are linked in a dynamic library called `libgopigo3.so`, which is then selected when running the code (by using CodeBlocks IDE settings).
The class member functions to be used for the thesis test are only the ones related to the DC motors, in particular to reset the encoder positions at the beginning of every test, to set PWM of the motors starting from batteries/supply voltage, to get motor status informations (encoder positions, PWM duty cycle, supply voltage) and to reset them all at the end of the test.

IMU libraries

Adafruit vendor for 10-DOF IMU provides SW code implementation for Arduino platforms only, using its own programming language. Since it is decided to write the overall code in C++, another solution must be found.
There exists a C++ library to get raw IMU data and to make sensor fusion of IMU sensors to recover angle pose using Euler angles or quaterniorns. The library is called *RTIMUlib2* and it's compatible with many commercial IMUs, included Adafruit 10-DOF one, [24].
The library has quite complex structure with many source/header files, but all functionalities can be provided by calling `RTIMULib.h` library and a dynamic library named `libRTIMULib.so` (which can be easily implemented by using whatever IDE, like CodeBlocks).
Again there's a main class containing the member functions needed to get raw sensors data and pose from sensor fusion. More informations in Section 4.3.2 and Section 9.1, where IMU settings are explained in detail to obtain measurements during practical implementations.

Save variables functions

To check RHIT results, at every DT step all variables needed are plugged into arrays and matrices and then saved into a text (.txt) file, organized in columns (the variables), and rows (value associated to each variable at every

sampling time step). Then, using MATLAB, the variables are converted into tables and plotted in figures.
For every test it cannot be known a priori the number of values associated to the variables, since they depend on sampling time (which may be variable) and test stop time. A solution can be define a very large array at the beginning and fill it with zeros. This is not a good solution because depending on the test we can waste a large amount of memory.
Nevertheless some C++ SW functions come in handy, they're based on `array.h` and `vector.h` C++ libraries:

```
#include <array>
#include <vector>

using namespace std;    // refers to the namespace where all C++ standard classes
    and functions are collected (from now on std is implicit)

// To create vectors of variables
array<data_type,array_dim> array_name}
vector<data_type> vector_name}
```

Difference between C++ arrays and vectors is that in arrays there's a single row of variables with predefined dimension, instead using vectors we don't need to define the dimension at the beginning, only the data-type (dynamic arrays). In addition can be created dynamic *Vectors of vectors* or *Vectors of arrays*, hence matrices without the need of specify row dimensions.
This is very helpful, and it's used finally to save the variables:

1. At every DT step, the variables are saved into an array with predefined dimensions, since their number is known a priori.

2. Then the array is plugged into a vector of arrays. like:

   ```
   #include <array>
   #include <vector>

   using namespace std;

   // To create matrices of variables
   vector<array<data_type,array_dim>> matrix_name;
   ```

3. At the next time step array can be filled with different values, and it can be *pushed back* into the vector of arrays: it means that the new array is placed exactly below the last saved, creating a new row (dynamically).

4. At the end of the test the vector of arrays has exactly the same row dimensions of the number of DT steps.

Other SW improvements

During work SW code is gradually improved for readability and optimization and to reduce dimensions (number of lines of code). For example almost all the functions are grouped using C++ class called `Main_functions` in a unique source/header file. In `ert_main.cpp` an instance of this class is added to call easily the member functions needed.

4.3 General Structure settings for EMC and robot plant

4.3.1 EMC

Code generation is used to convert the EMC Simulink files into SW code as explained in Section 4.2.1: 4 EMC models are build respectively for Left motor, Right motor, Orientation and Longitudinal Position, each of them have their own source and header pack files obtained by code generation.
In main source function (`ert_main.cpp`) different HW timers are associated to each EMC model. Then all operations needed to step the model and connecting it to the plant measurements are done, as already discussed in the if statement related to corresponding HW timer.

DC motors EMC

Two possible structures are selected for each DC motor EMC:

1. A unique class function is generated for both EM and Control model parts of DC motor EMC. In this case a unique HW timer is sufficient to step the model.

2. EM and Control parts of overall EMC are splitted in 2 generated codes, leading to 2 separated classes. This is done because, as already seen in Section 3.1 for GoPiGo3 main board specifications, PWM is sent to the motors with a fixed sampling time, $T_{PWM} = 0.02\,\text{s}$: using this separation we can run the Control part at the same PWM sampling time ($T_{ctrl} = T_{PWM} = 0.02\,\text{s}$), and EM part with fixed/variable one. This solution leads unavoidably to at least 2 HW timers, one for control part (T_{ctrl}) and the other for EM part (T_{EM}).
   With this implementation for simplicity EM part contains not only to Embedded Model and Noise estimator, but also to Reference dynamics controlled with static-state control.

If the second structure is used, we need to be careful with the connections between the parts, which need to be done manually in the code: regarding EM part, controllable/disturbance states $\hat{x}_c$ and $\hat{x}_d$, reference state $\overline{y} = y_{ref}$ and reference input $\overline{u}$ at every step must be passed to Control part; instead the overall control input u must be passed from Control to EM part at every DT step. A scheme is shown in Figure 4.2.

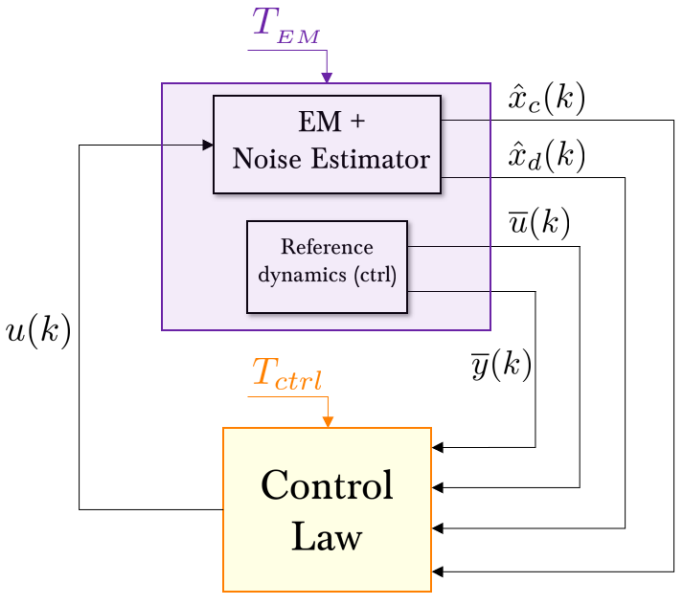


Figure 4.2: DC motor EMC splitted scheme for RHIT implementation

Orientation and position EMC

Both orientation and position EMC need the information about the IMU measurements, which are obtained in a fixed sampling time (around $T_{IMU} = 0.021\,\text{s}$). For this reason, one may think to use a separated timer with fixed sampling time for EMC Control part. However we'll see that performing the tests without splitting the 2 parts and using a unique HW timer is sufficient to get good results in terms of tracking and model errors.

4.3.2 Plant

In RHIT the control models mimic in a simplified way a certain plant, which is composed in this case by DC motors (to control wheel angular speeds) and IMU (to control robot orientation and position). SW code implementation of wheels angular speeds and IMU data are needed.

Motors

Motor encoder positions are the main measurements to be taken. Then with a simple discrete derivation (Forward-Euler is used) the wheel angular speeds are recovered.
Control input u coming from EMC is converted to PWM duty cycle, which is then applied physically to the motors, which changes corresponding position encoders status. `GoPiGo3.cpp` and `GoPiGo3.h` contains all member functions to do all these operations, the corresponding SW code is:

```
#include <GoPiGo3.h>

GoPiGo3 GPG;    // Main GoPiGo3 class instance

void GPG_measurements() {
    // Motor status parameters
    uint8_t state;              // Motor status flag, not used
    int8_t power;               // PWM duty cycle (%)
    int32_t pulses;             // Encoder position (degrees)
    int16_t dps;                // Wheel speed position (degrees per second), not
    used

    motor_voltage = GPG.get_voltage_battery();  // Recover battery/supply voltage
    pwm = u/ motor_voltage * 100;               // Proportion between voltage and
    PWM
    GPG.set_motor_power(MOTOR_LEFT, pwm);       // Apply PWM physically to motor
   (e.g. LEFT)
    GPG.get_motor_status(MOTOR_LEFT,state,power,pulses,dps);    // Get encoder
   pulses
    pos_m_d = (double) pulses / MOTOR_TICKS_PER_DEGREE;         /* Divide encoder
    pulses by MOTOR_TICKS_PER_DEGREE = 2 to obtain encoder position */
}
```

The motor power ranges from 0 to 100, and it's directly proportional to supply voltage, which ranges from 0 to V_{MAX}: knowing control input u and V_{MAX} = `motor_voltage` we can recover easily PWM value.
Because the encoder has a resolution of 720 pulses/revolution = 0.5° (means that with 720 pulses 2 revolutions of 360° are performed), n° pulses need to be divided by `MOTOR_TICKS_PER_DEGREE = 2`. Finally since the position is in degrees, it's needed to multiply by $\pi/180$ to obtain radians measure unit.

IMU

IMU is able to provide angular speeds in Cartesian ξ, ν, z directions and also sensor fusion pose (in Euler angles or quaternions). To measure them IMU library `RTIMULib.h` comes in handy.
First IMU classes are instantiated, then settings file `RTIMULib.txt`) is loaded (containing IMU model type, sensor settings like full scale range, calibration offsets, sample rate etc.), finally devices (among gyroscope, accelerometer and magnetometer) to be used for pose sensor fusion are decided. After this initialization, IMU measurements can be taken every DT step. For example to take IMU yaw angle θ_z and angular speed w_z basic SW code is:

```
#include <RTIMULib>      // IMU library
#include <stdlib.h>      // Standard library
#include <stddef.h>      // NULL macro

RTIMU *imu;
RTIMUSettings *settings;
RTIMU_DATA imuData;          // IMU instances of classes

double theta_z = 0;
double w_z = 0;

void IMU_initialize() {
    settings = new RTIMUSettings("IMU_info/","RTIMULib");   // Load settings file
    RTIMULib.txt

    imu = RTIMU::createIMU(settings);        // Load settings in imu class

    // Check if IMU is connected or not
    if ((imu == NULL) || (imu->IMUType() == RTIMU_TYPE_NULL)) {
        printf("No IMU found\n");
        exit(1);
    }

    imu->IMUInit();      //  set up IMU

    // Establish if pose sensor fusion uses or not gyro, accelerometer and
   magnetometer
    imu->setGyroEnable(true);
    imu->setAccelEnable(true);
    imu->setCompassEnable(true);
}

void IMU_measurements() {
    imuData = imu->getIMUData();

    // Get orientation yaw angle
    theta_z = imuData.fusionPose.z();

    // Get angular speed around z-axis
    w_z = imuData.gyro.z();
}
```

Part II

DC motors EMC

Chapter 5

EMC DC motors theory

In this part the Embedded Models of the 2 DC motors of the robot GoPiGo3 are build. The following steps are considered:

1 Since the main parameters (Armature resistance R_a, armature inductance L_a, inertia and friction) of the robot motors are unavailable, a parameter identification based on input (armature voltage) and output (angular speed) measurements was performed.

2 The estimated parameters are used to build the Fine Model of the DC motor for simulation purposes (MIL) and its Embedded Model Control (considering a simplified version of fine model, in order to be implemented in the robot board).

5.1 DC Motor Parameters identification and validation

Datasheets of the robot motors are not available, the only reliable informations are the Armature Voltage V_a, the angular speeds of the wheels ω_R, ω_L computed using 2 Hall encoders directly implemented, and the armature resistances R_a which can be easily measured across the motor terminals using a multimeter. Their values are $R_a \approx 19\,\Omega$ for the left motor and $R_a \approx 17.3\,\Omega$ for the right motor. However also R_a will be added as unknown parameters to be identified, to verify their values. For these reasons, parameter identification is performed in order to find the unknown parameters. 2 main steps are considered:

- **Identification phase**: A dataset of input and output measurements which has inside sufficient dynamic informations is used. In the case of DC motors step inputs are good, imposed for a certain amount of time, and the values of V_a and ω_r are collected. In some cases other input-output datasets are considered to improve the identification.

- **Validation phase**: The response of DC motor models with estimated parameters are compared with real response, using a dataset different from the one used for identification.

During work, different strategies are adopted to estimate the parameters, we can refer to them with a version number (v1.0, v1.1 etc.).

5.1.1 Main workflow

Identification phase

Starting from the usual DC motor tranfer fucntion in continous time, using the mechanical time constant τ_m, the electrical time constant τ_a and back Electromotive force constant k_v:

$$\frac{\omega(s)}{V_a(s)} = \frac{\frac{1}{k_v}}{1 + s\tau_m + s^2\tau_m\tau_a} \tag{5.1}$$

we can recover the transfer function in discrete time by considering the $\mathcal{Z}$ transform of s.
In general, every transfer function (TF) in Laplace domain $G(s)$ can be converted in $\mathcal{Z}$ domain transfer function $G(z)$ by considering the relation $G(z) = (1 - z^{-1})\mathcal{Z}\left\{\frac{G(s)}{s}\right\}$. For our TF ($T$ is a generic sampling time):

$$G(s) = \frac{1}{s} \quad \Rightarrow \quad G(z) = (1 - z^{-1})\mathcal{Z}\left\{\frac{1}{s^2}\right\} = \frac{z-1}{z}\frac{Tz}{(z-1)^2} = \frac{T}{z-1}$$
$$G(s) = \frac{1}{s^2} \quad \Rightarrow \quad G(z) = (1 - z^{-1})\mathcal{Z}\left\{\frac{1}{s^3}\right\} = \frac{z-1}{z}\frac{T^2 z(z+1)}{2(z-1)^2} = \frac{T^2(z+1)}{2(z-1)}$$

The discrete time TF of Equation (5.1) becomes:

$$\frac{\omega(z)}{V_a(z)} = \frac{\frac{1}{k_v}}{1 + \frac{(z-1)}{T}\tau_m + \frac{2(z-1)^2}{T^2(z+1)}\tau_m\tau_a} \tag{5.2}$$

After some computations we can write it in the following regressive form:

$$\frac{\omega(z)}{V_a(z)} = \frac{\beta_2 + \beta_1 z^{-1} + \beta_0 z^{-2}}{1 + \alpha_1 z^{-1} + \alpha_0 z^{-2}} \qquad \begin{aligned} \beta_2 &= 0 \\ \beta_1 = \beta_0 &= \frac{T^2}{k_v(\tau_m T + 2\tau_m\tau_a)} \\ \alpha_1 &= \frac{T^2 - 4\tau_m\tau_a}{\tau_m T + 2\tau_m\tau_a} \\ \alpha_0 &= \frac{T^2 - \tau_m T + 2\tau_m\tau_a}{\tau_m T + 2\tau_m\tau_a} \end{aligned} \tag{5.3}$$

Using the discrete TF model in Equation (5.3) initial guess of β_1, α_1 and α_0 can be obtained using different optimization methods. Due to simplicity ***Least Squares Method*** is the best, because we only need to find the approximate range of parameters (some error is accepted).
Using the backward shift operator q^{-1} we can write the output in time domain $\omega(t)$ as linear combination of input/output past values:

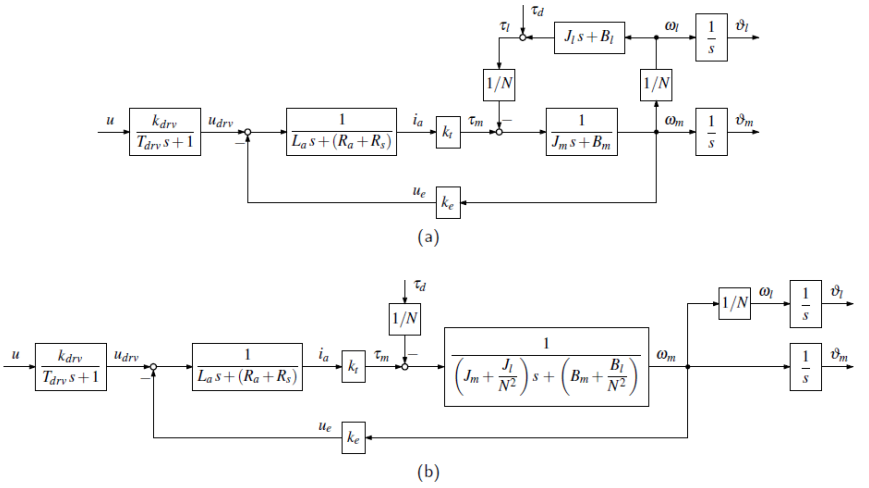
$$\omega(t) = -\alpha_1\omega(t-1) - \alpha_0\omega(t-2) + \beta_1\left[V_a(t-1) + V_a(t-2)\right], \quad t = 3, \ldots, N$$
$$\underbrace{\begin{bmatrix} \omega(3) \\ \vdots \\ \omega(N) \end{bmatrix}}_{b} = \underbrace{\begin{bmatrix} -\omega(2) & -\omega(1) & V_a(2) + V_a(1) \\ \vdots & \vdots & \vdots \\ -\omega(N-1) & -\omega(N-2) & V_a(N-1) + V_a(N-2) \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} \alpha_1 \\ \alpha_0 \\ \beta_1 \end{bmatrix}}_{x} \tag{5.4}$$

The Least square estimation problem can be solved as:

$$x = (A^\top A)^{-1} A^\top b \tag{5.5}$$

Since the TF DC motor model in Equation (5.1) is not considering the gearbox reduction N, the output value $w(s)$ obtained is not the one downstream at the wheels: we can correct the value with $\omega = N\omega'$.
Obviously this is an assumption, supposing that the gearbox does not modify significantly the output value. This can be explained by studying the inertia and friction effects of the gearbox in the entire DC motor model: basically the gearbox inertia and friction are combined with the ones of the DC motor to obtain a total inertia and friction J_{eq} and β_{eq} respectively.

What are exactly J_{eq} and β_{eq}? Inside them are present both the motor and gears inertia/friction. Since a reduction of the speed by a factor of $N = 120$ is done by the gears, these values changes consequently.
A laboratory activity of University of Padova about DC gear motor modelling is taken as reference [9]: in Figure 5.1 taken from this reference a dynamic block model of the motor with the addition of the gear reduction part is considered.
$\frac{k_{drv}}{T_{drv}s+1}$ is a voltage driver modelled as Low-pass filter (not important in this field). $k_e = k_v$ in this case.



J_m	*Motor inertia*
B_m	*Motor friction*
J_l	*Gear inertia*
B_l	*Gear friction*
ω_m	*Motor speed not reduced by gears*
$\omega_l = \omega$	*Motor speed reduced by gears*

Figure 5.1: DC gear motor J_{eq} and $\beta_{eq} = B_{eq}$ explanation

We can divide the mechanical dynamics in 2 parts, the rotor and gear load mech. dynamics. Their equations are (T_m motor torque, T_l' gear load torque motor side, T_l gear load torque gear side, T_d external loads torque, for simplicity = 0):

$$\begin{cases} J_m \frac{d\omega_m(t)}{dt} + B_m\omega_m(t) = T_m(t) - T_l'(t) \\ J_l \frac{d\omega_l(t)}{dt} + B_l\omega_l(t) = T_l(t) - T_d(t) \end{cases}$$

$$\begin{cases} \omega_l = \frac{\omega_m}{N} \\ T_l = NT_l' \end{cases}$$

Making equal the 2 equations considering $T_l = T_l'$ we obtain the equation of motor torque by considering all the inertia and friction at *motor side*:

$$\underbrace{\left(J_m + \frac{J_l}{N^2}\right)}_{J_{eq}} \frac{d\omega_m(t)}{dt} + \underbrace{\left(B_m + \frac{B_l}{N^2}\right)}_{B_{eq}} \omega_m(t) = T_m(t) \tag{5.6}$$

From now on $\beta_{eq} = B_{eq}$.

Assumptions 5.1 *Back-electromotive force k_v and torque constant k_t in a DC motor are supposed to be equal $k = k_v = k_t$. Parameters without index are the DC motors only parameters without gearbox effect. To insert the gearbox effect the last parameters are assumed to be multiplied by gear ratio N, the new parameters are defined by an index: hence $k' = kN$.*

This is because steady-state amplitude value of wheel speeds (related to k') is approximately the DC motor speeds value (related to k), but increased by a gearbox factor of N.
Then for a DC motor model mechanical and electrical time constants can be approximated as (the gearbox effect is inside parameter k now):

$$\tau_a = \frac{L_a}{R_a}, \quad \tau_m = \frac{J_{eq} R_a}{k^2} \tag{5.7}$$

and supposing that the friction coefficient $\beta_{eq} = 0\,\mathrm{Nm(rad/s)}^{-1}$ (it is a mild assumption, since we will verify after the estimation that is very small) and that the motor constants are equal $k = k_v = k_t$, we have to solve the 3 equations on right of Equation (5.3) to find the initial guess of parameters J_{eq}, L_a. In addition we add the parameters β_{eq} and k to verify our initial assumptions. In some cases also R_a was estimated even if it can be directly measured, because its measurement using a multimeter can lead to some parameter errors (for example the range for the left motor is 18–21 Ω).
Initial guess parameters are then used to make the real estimation using Simulink Design Optimization Toolbox from Matworks. Indeed starting from values near the real ones increases the convergence to reliable parameters. The method used by toolbox is Non linear Least Squares.

Validation Phase

In this phase we need to verify the model with final estimated parameters obtained during identification phase, using an input-output dataset different from the one used for identification.
The Root Mean Squared Error $RMSE = \sqrt{\frac{1}{N_{meas}} \|y(t) - \hat{y}_m(t)\|_2^2}$ (where $y(t)$ are the real measurements, $\hat{y}_m(t)$ the estimated ones, N_{meas} are the total number of measurements) is used as validation goodness parameter. Lower RMSE values indicate better estimation.

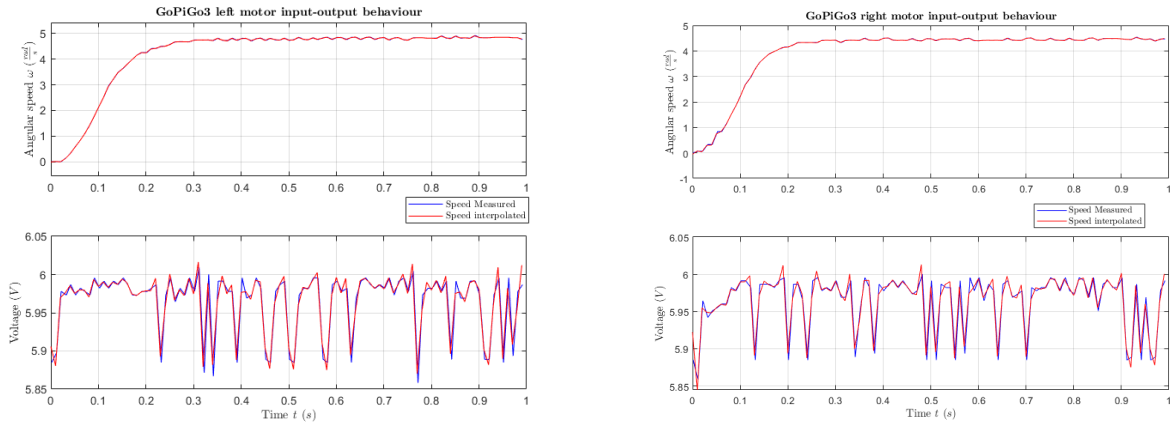
5.1.2 Version 0.0

Identification

The maximum supplying voltage of the motors is 12V, so a step of 6V is chosen for the identification experiment, to avoid non linearities.
The measurements are taken with a sampling time of $T = 0.01\,\mathrm{s}$ using Raspberry HW timers explained in Section 4.2.2. To overcome possible timer delays, in addition an interpolation of the measured data is performed, to obtain exactly $T = 0.01\,\mathrm{s}$.
Armature resistance values are not inserted as unknown parameters to be identified, and the measured values are taken: $R_a = 19\,\Omega$ for the left motor and $R_a = 17.3\,\Omega$ for the right motor.
For both the 2 motors transient behaviour is presented in Figure 5.2. Following the procedure in Section 5.1.1, the parameters of DC motor second order regressive model are found using LS estimation method, then using Equation (5.7) the initial guess parameters are recovered. Finally, using Simulink Optimization toolbox final parameters are estimated. All of them are resumed in Table 5.1.

Validation

For example there are considered steps with 5V and 9V, repeated alternatively for 2 s, for 2 cycles. The results are shown in Figure 5.3: RMSE was computed in both cases, giving the following results:



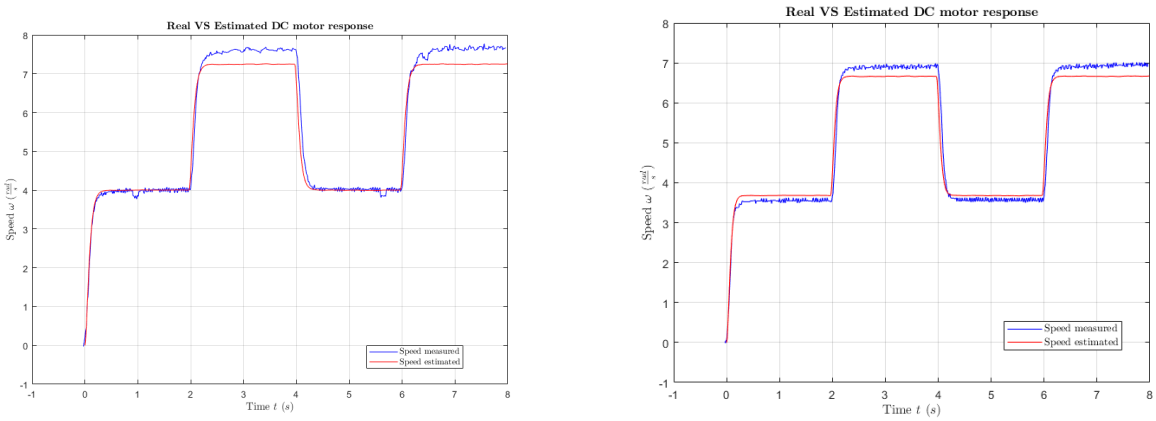
(a) *Left motor step response (*6V*)*

(b) *Right motor step response (*6V*)*

Figure 5.2: Left and Right DC motors v0.0 identification datasets

	Parameter	Unit measure	Left motor	Right motor
DC motor regressive form parameters	β_1	–	0.0176	0.02
	α_1	–	−1.3403	−1.2291
	α_0	–	0.3826	0.2809
Initial guess parameters	β_{eq}	$\mathrm{Nm(rad/s)}^{-1}$	0	0
	J_{eq}	kgm^2	7.9677e−7	9.6818e−7
	L_a	H	0.2025	0.1457
	k_v	V s	0.0100	0.0108
	τ_m	s	0.1510	0.1510
	τ_a	s	0.0107	0.0084
Final estimated parameters	β_{eq}	$\mathrm{Nm(rad/s)}^{-1}$	4.70e−11	4.51e−14
	J_{eq}	kgm^2	5.099e−7	5.625e−7
	L_a	H	0.428	0.375
	k_v	V s	0.0103	0.0112

Table 5.1: DC motor regressive form parameters and initial guess parameters - Left and Right motors v0.0



(a) *Left motor*

(b) *Right motor*

Figure 5.3: Left and Right DC motors v0.0 validation datasets

$$RMSE_L = 0.3988 \qquad RMSE_R = 0.3624$$

Seems that the time needed to reach the steady state conditions for the speed is more or less similar, problem is the value reached at steady state: this problem can be explained by assuming that some parameters have a dependence on the output speed w.
A problem arises with founded estimated parameters: the estimation was done using as dataset the output speed computed by robot producer, which is filtered to reject the high frequency dynamics. The filter always introduces a time delay, so at least the value of τ_m and τ_a are wrong, since they're higher than reality.
To avoid this problem in Section 5.1.3 another procedure is followed.

5.1.3 Version 1.0

The problem in version 0.0 (Section 5.1.2) can be overcome by measuring the position given by motor encoders, and then making a Forward-Euler discretization to obtain an approximation, as the following:

$$\dot{\phi}(t) \approx \frac{\phi(k+1) - \phi(k)}{T}$$

where ϕ is the encoder position, T is the sampling time used as increment value. Since the rotation is around only 1 axis, $\dot{\phi}(t) = \omega_i(t)$. With this procedure output speed is not filtered and all high frequency dynamics are present. In the next, parameters identification and validation are considered, only for left motor.

Identification

Table 5.2 presents the differences between the DC motor estimated parameters avoiding output filtering and changing identification dataset: open-loop (OL) I/O dataset Figure 5.2 for left motor v0.0, and closed loop (CL) I/O dataset, with input voltages and output speeds taken from a physical RHIT of DC motor EMC for left motor v1.0.
The difference wrt previous version can be seen looking at τ_m and τ_a parameters. τ_m is reduced, meaning that the delay due to output filtering is no longer present, also τ_a is reduced [1], but this value is used only for fine DC motor model in Simulink, for Model-in-the-Loop (MIL) tests. Indeed for the EMC of DC motor are sufficient only τ_m and k_v.
For the CL estimation parameters the values after using Simulink Design Optimization Toolbox are practically the same of LS initial guesses parameters (only $J_{eq} = 3.9197\text{e}{-7}\,\text{kgm}^2$ and $\beta = 0\ \text{N m}\,(\text{rad/s})^{-1}$ are different, but in a minimal way).

Parameter	Unit measure	Closed Loop	Open Loop
β_{eq}	$\text{Nm}(\text{rad/s})^{-1}$	2.2204e−14	4.7002e−11
J_{eq}	kgm^2	3.6873e−7	5.0983e−7
L_a	H	0.030687	0.42785
R_a	Ω	20.133	19
k_v	V s	0.0102	0.0103
τ_m	s	0.0714	0.0913
τ_a	s	0.0015	0.0225

Table 5.2: Final Estimated parameters Left motor v1.0, with comparison with v0.0

[1] In identification tests for all DC motor versions, the values of τ_a present an high variation range, because estimating precisely this parameter is very difficult, due to its very small physical effect on motor dynamics

The Figure 5.4 represent output speed ω_{LS} behaviour obtained with the estimated parameters using LS method (in red), compared with the measured speed ω_{meas} (in blue).

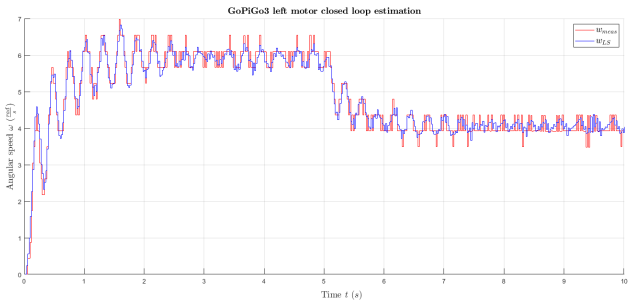


Figure 5.4: DC motor left v1.0 LS speed estimation, using closed loop dataset

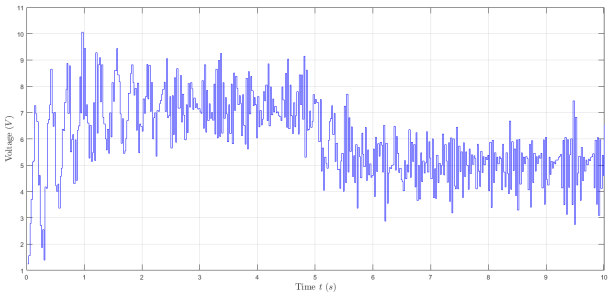


Figure 5.5: DC motor left v1.0 LS voltage, using closed loop dataset

Validation

Figure 5.6 and Figure 5.7 represent the validation of the estimated parameters using a different output dataset (multiple steps from 4 to 8 V), filtered (moving average filter) and not filtered.

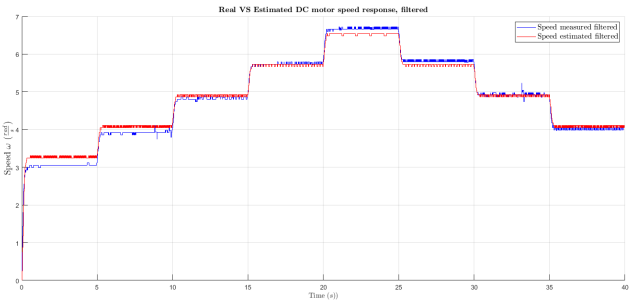


Figure 5.6: DC motor left v1.0 speed validation, filtered, using closed loop dataset

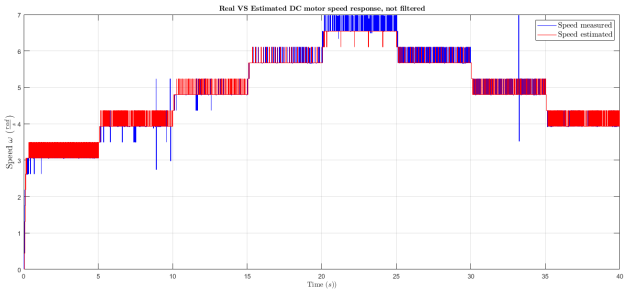


Figure 5.7: DC motor left v1.0 speed validation, NOT filtered, using closed loop dataset

RMSE value for validation is: $RMSE_L$ = 0.2813, which is better of Version 0.0.
From this validation, in Figure 5.6 we can notice that the second order model (Equation (5.1)) with fixed parameters has some limits.
Indeed seems that the estimated steady state output speed is near to the measured speed only for few input voltages, and in the other cases is lower or higher: a possible explanation is that the gearbox part slightly modifies the parameter values depending of motor speed. In the future the estimation can be improved by adding more informations to the motor model.

5.1.4 Version 1.1

For this version, parameters identification and validation are performed using 4 different types of input-output datasets, to obtain the highest possible dynamic information of the system. Again the identification is done only for left DC motor.

Identification

The parameters found using the LS solving equation and initial guess parameters are:

Parameter	Value
β_1	0.0701
α_1	−0.6204
α_0	−0.2060

Table 5.3: Parameters DC motor model in regressive form using LS method - Left motor v1.1

Parameter	Unit measure	Value
β_{eq}	$\mathrm{Nm(rad/s)^{-1}}$	0
J_{eq}	$\mathrm{kgm^2}$	8.3480e−7
L_a	H	0.1039
R_a	Ω	19
k_v	V s	0.0103
τ_m	s	0.1490
τ_a	s	0.0055

Table 5.4: Initial guess parameters Left motor v1.1

For the initial guess only 1 among 4 I/O datasets can be used, it is shown in Figure 5.8 and Figure 5.9:

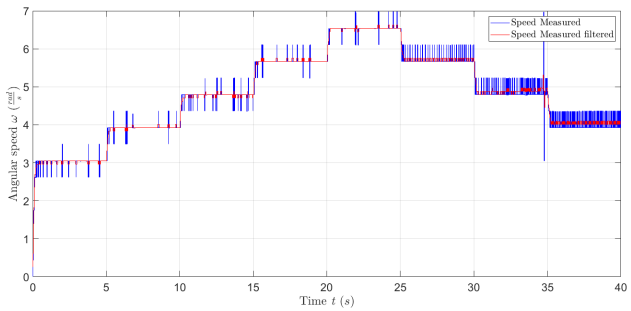


Figure 5.8: DC motor left v1.1 initial guess output speed dataset

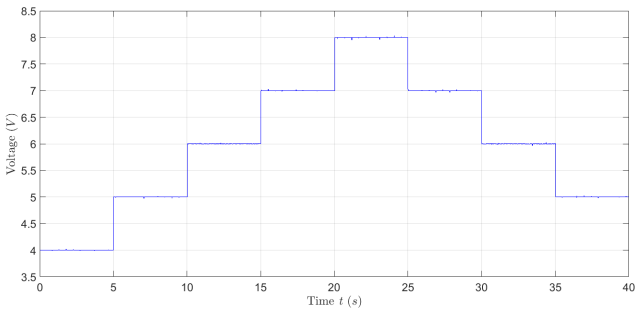


Figure 5.9: DC motor left v1.1 initial guess input voltage dataset

Then, in Simulink Design Optimization Toolbox the whole 4 I/O datasets are used.
In addition, the measured voltage inputs used for the estimation were filtered (rejecting the time delay the filter unavoidably brings), to have a smooth function and simplify the estimation.
The final estimated parameters are:

Parameter	Unit measure	Value
β_{eq}	$\mathrm{Nm(rad/s)^{-1}}$	4.3116e−14
J_{eq}	$\mathrm{kgm^2}$	4.6743e−7
L_a	H	0.2152
R_a	Ω	18.5730
k_v	V s	0.0102
τ_m	s	0.0839
τ_a	s	0.0116

Table 5.5: Final estimated parameters Left motor v1.1

The comparison with respect to all 4 datasets used is in the next 4 figures. It is added a third signal of filtered measured output (in red), useful to see the mean speed value during steady state conditions and compare with estimated output (in green), because the presence of high frequency peaks creates visual problems.

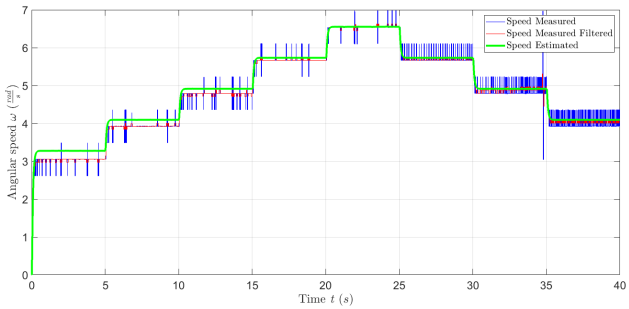


Figure 5.10: DC motor left v1.1 final estimated VS measured speed - Dataset 1

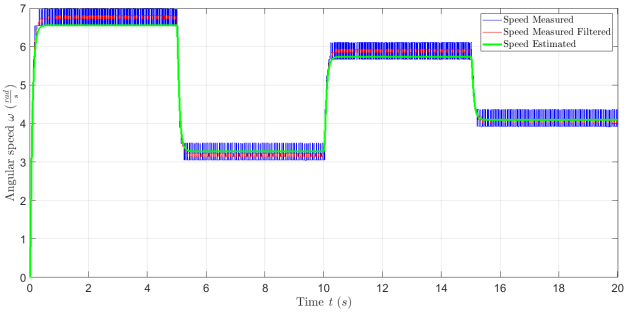


Figure 5.11: DC motor left v1.1 final estimated VS measured speed - Dataset 2

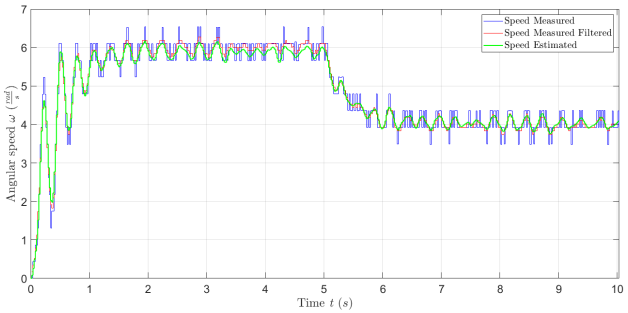


Figure 5.12: DC motor left v1.1 final estimated VS measured speed - Dataset 3

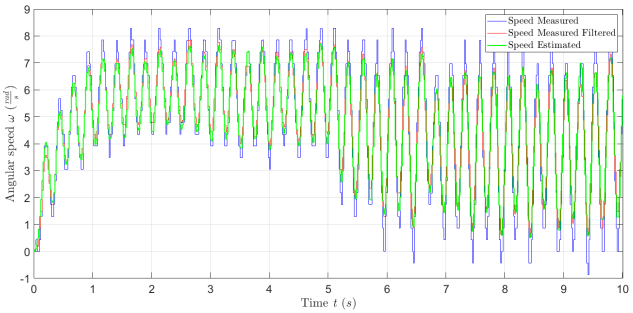


Figure 5.13: DC motor left v1.1 final estimated VS measured speed - Dataset 4

We can see immediately that also by using 4 I/O datasets, the estimated speed seems not to fit perfectly the measured one, as it can be clearly seen in Figure 5.10: as already explained in Section 5.1.3 for left motor v1.0, some parameters may change their values depending on the output speed, this can be caused by the effects of the gearbox.
However this doesn't affect the DC motor EMC, since we'll see in the next sections that for simplicity the model will be build based on fixed parameters τ_m and k_v, and their estimated values will be sufficient to guarantee good disturbance rejection and tracking during physical implementation.

Validation

To be done with another dataset, but almost all are used for the identification. However by considering the last figures with comparison of estimated and measured output speeds, the estimated parameters can be considered sufficiently reliable.

5.1.5 Version 1.2 (only for EMC)

Another parameter identification test is done using a dataset containing negative output speed measured values too. The identification and validation is done for both the 2 motors, left and right.
The set of parameters related to mechanical part are used only for EMC models, for fine models a different strategy is followed, explained in detail separately in Section 5.1.6, adding in model used for identification the informations about motor Coulomb frictions.

Identification

The parameters of motor regressive second order model (Equation (5.3)) using LS estimation method and the initial guess parameters are in the following tables:

Parameter	Left	Right
β_1	0.1609	0.1313
α_1	−0.3121	−0.4835
α_0	−0.2929	−0.1471

Table 5.6: Parameters DC motor model in regressive form using LS method - Left and Right DC motors v1.2 (for EMC only)

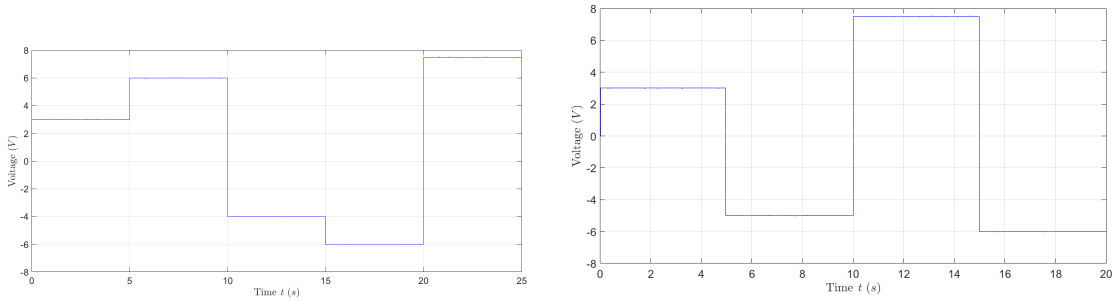
Parameter	Unit measure	Left	Right
β_{eq}	$\mathrm{Nm(rad/s)^{-1}}$	0	0
J_{eq}	$\mathrm{kgm^2}$	4.0925e−7	5.5084e−7
L_a	H	0.0660	0.0903
R_a	Ω	19	18
k_v	V s	0.0102	0.0117
τ_m	s	0.0755	0.0721
τ_a	s	0.0034	0.0050

Table 5.7: Initial guess parameters - Left and Right motor v1.2 (for EMC only)

For the identification 2 datasets are used, one using in-out measured values in open loop conditions (Dataset n° 1), the second using measured values from one of a closed loop test performed with one EMC DC motor version (Dataset n° 2).

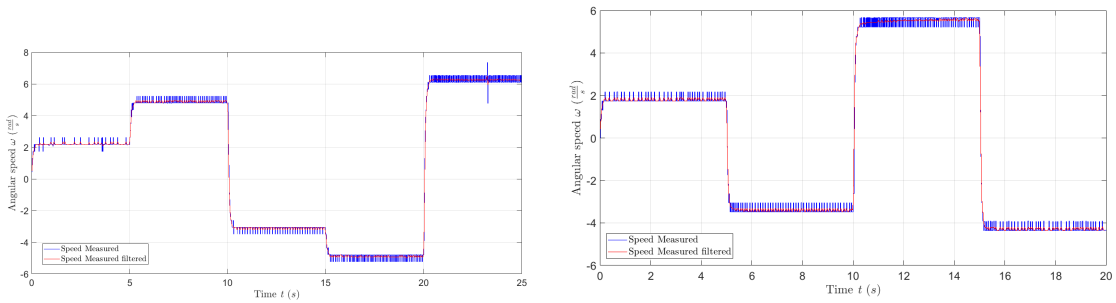
The Dataset n° 2 is characterized by step input voltages both positive and negative, and this one was used to find the first guess parameters (Figure 5.14 and Figure 5.15).

The 2 datasets of input voltages and output speeds are visible in the next figures: in the output measured speed ones, blue is the measured signal and red is the filtered signal (it's useful to make a comparison with the estimated speed, which is obtained by using a continuous time model).



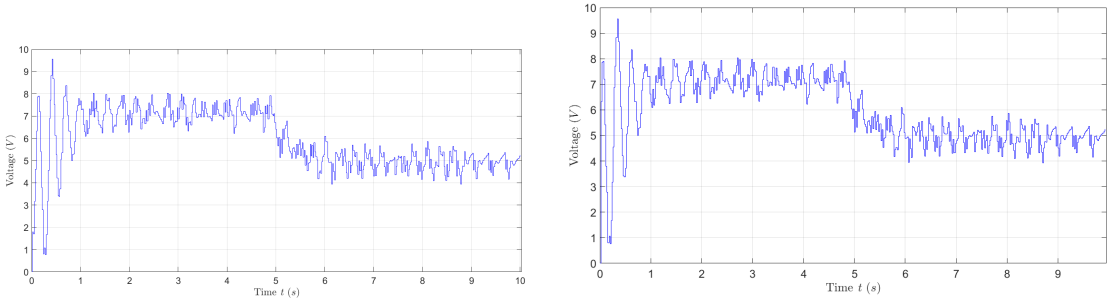
(a) *Left motor* (b) *Right motor*

Figure 5.14: DC motor v1.2 Dataset n° 1, input voltage



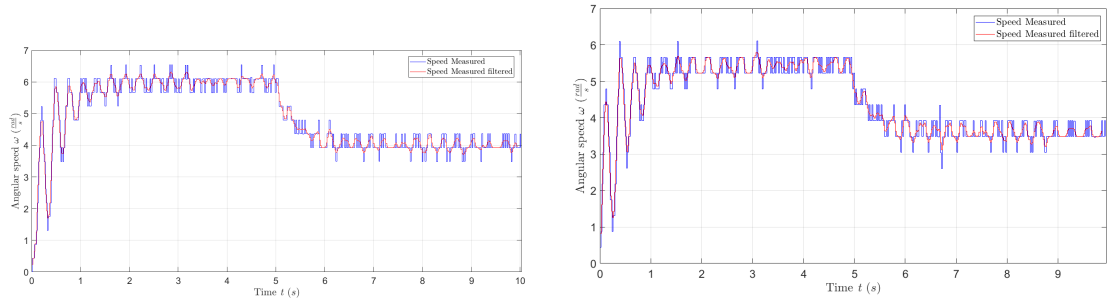
(a) *Left motor* (b) *Right motor*

Figure 5.15: DC motor v1.2 Dataset n° 1, output speed



(a) *Left motor* (b) *Right motor*

Figure 5.16: DC motor v1.2 Dataset n° 2, input voltage



(a) *Left motor* (b) *Right motor*

Figure 5.17: DC motor v1.2 Dataset n° 2, output speed

Using Simulink Design Optimization Toolbox both the last 2 datasets are used for the identification of parameters. The final estimated parameters are:

Parameter	**Unit measure**	**Left motor**	**Right motor**
β_{eq}	$\mathrm{Nm(rad/s)^{-1}}$	2.1032e−20	4.8623e−14
J_{eq}	$\mathrm{kgm^2}$	3.6898e−7	4.2876e−7
L_a	H	0.243 36	0.247 35
R_a	Ω	19.154	17.781
k_v	V s	0.010 179	0.011 553
τ_m	s	0.0682	0.0571
τ_a	s	0.0127	0.0139

Table 5.8: Final estimated parameters Left and Right motor v1.2 - For EMC models only

Resulting estimated output speeds are:

Validation

Validation is performed using a new dataset of input voltage steps from $[-8, 8]$ V (Figure 5.24 and Figure 5.25). RMSE values for both motors validations are:

$$RMSE_L = 0.1662 \qquad RMSE_R = 0.2416 \tag{5.8}$$

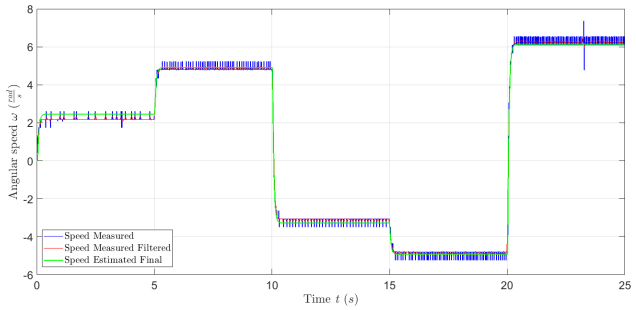


Figure 5.18: Estimated and measured output speed - Left motor v1.2, Dataset n° 1

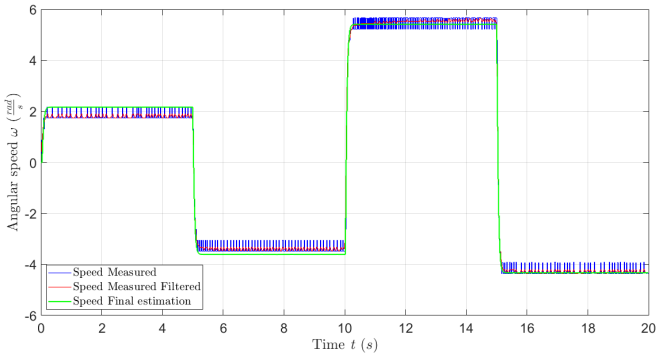


Figure 5.19: Estimated and measured output speed - Right motor v1.2, Dataset n° 1

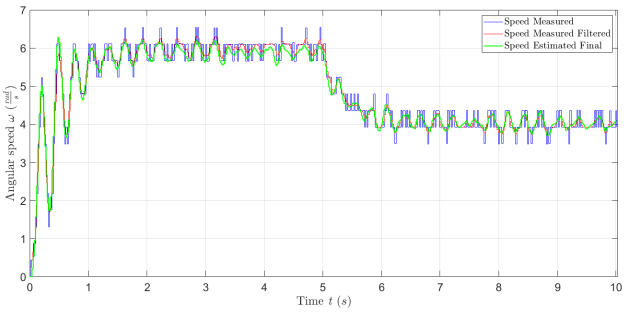


Figure 5.20: Estimated and measured output speed - Left motor v1.2, Dataset n° 2

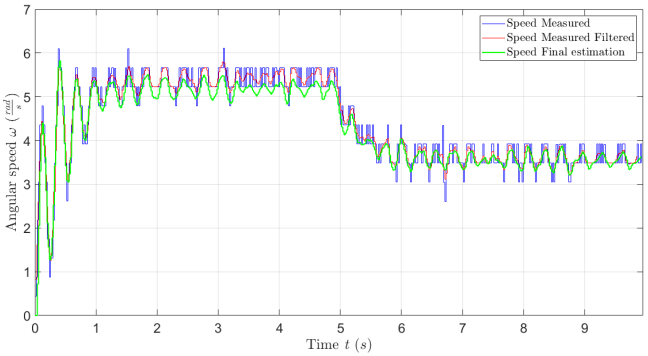


Figure 5.21: Estimated and measured output speed - Right motor v1.2, Dataset n° 2

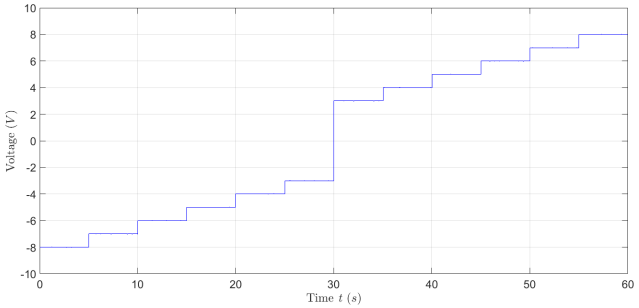


Figure 5.22: Validation dataset input voltage - Left motor v1.2

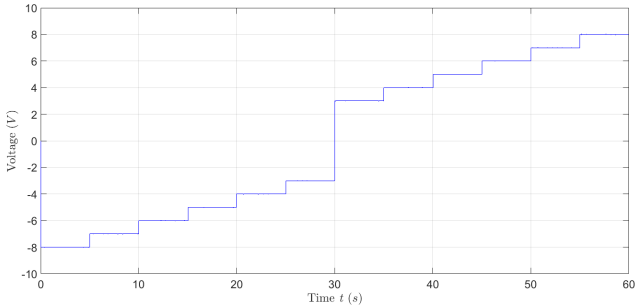


Figure 5.23: Validation dataset input voltage - Right motor v1.2

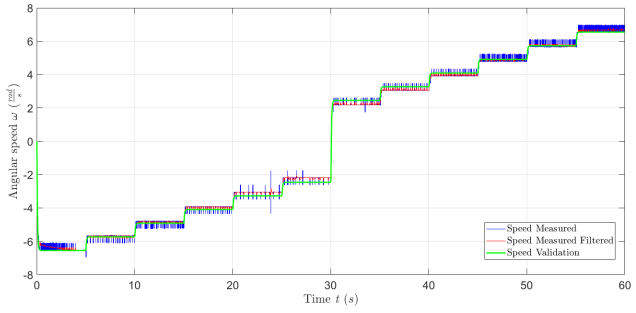


Figure 5.24: Estimated and measured output speeds - Validation dataset, Left motor v1.2 (only for EMC)

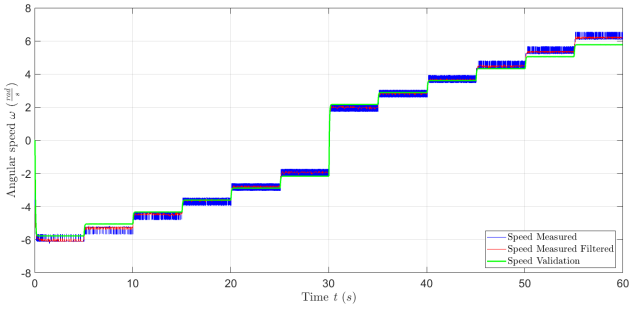


Figure 5.25: Estimated and measured output speeds - Validation dataset, Right motor v1.2 (only for EMC)

5.1.6 Version 1.2 (only for Fine models)

One of the main problems, at least to built a fine model for MIL simulations, is that the estimated parameters model not always lead to right angular speeds measured in open loop. For example looking at the validation plots of Figure 5.24 and Figure 5.25 it is clear that the estimated model output speed (in green) has a certain vertical offset wrt some measured output values.
This problem can be slightly overcome adding to estimation parameters model other informations about frictions: they can be simplified in a Coulomb friction always present and independent on the motor angular speed unless for its sign, [12]. This lead to a load torque acting on the motor of the following type:

$$T_C = \beta_c \, \mathrm{sgn}(\omega_i), \qquad \mathrm{sgn}(\omega_i) = \begin{cases} -1, & \text{if } \omega_i < 0 \\ 0, & \text{if } \omega_i = 0 \\ 1, & \text{if } \omega_i > 0 \end{cases} \tag{5.9}$$

Graphically the Coulomb friction torque appears as:

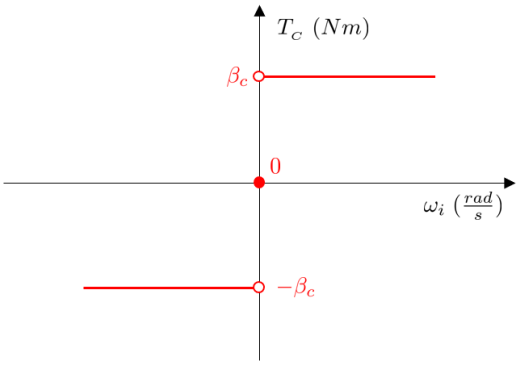


Figure 5.26: Coulomb friction torque model

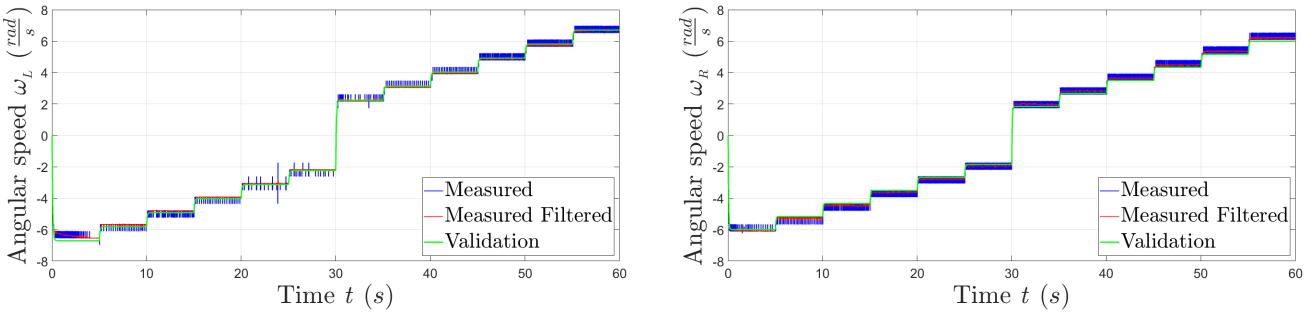
During parameters identification in the model is added a gain and a sign blocks, taking as input the output motor speed and giving as result a load Coulomb Torque T_C to be added in mechanical motor part.
The I/O datasets used for identification are the same of Figure 5.15 and Figure 5.17, the final estimated parameters for both right and left motor are (the new parameter β_c for Coulomb friction is added):

Parameter	Unit measure	Left motor	Right motor
β_{eq}	$\mathrm{Nm(rad/s)^{-1}}$	1.7525e−7	3.8736e−14
β_c	$\mathrm{Nm(rad/s)^{-1}}$	0.00023586	0.0004496
J_{eq}	$\mathrm{kgm^2}$	3.1487e−7	3.5903e−7
L_a	H	0.054 22	0.239 53
R_a	Ω	19.088	17.453
k_v	V s	0.0089386	0.010034
τ_m	s	0.0752	0.0622
τ_a	s	0.0028	0.0137

Table 5.9: Final estimated parameters Left and Right motor v1.2 - For fine models only

Comparing with estimated parameters in Table 5.8 the presence of a Coulomb friction reduces k_v value, which is related to output speed steady-state value: this means that now from Coulomb gain β_c depends part of motor steady-state dynamic behaviour.

Making a validation using the same dataset of v1.2 parameters for only EMC (Figure 5.24 and Figure 5.25), the difference is visible:



(a) *Left motor* (b) *Right motor*

Figure 5.27: Validation left and right motors v1.2 (only for Fine models)

Now in Figure 5.27 the green output speed related to estimated model is near to measured one for almost all the steps. Indeed also the RMSE values are lower than the ones found before (Equation (5.8)):

$$RMSE_L = 0.1196 \qquad RMSE_R = 0.1624$$

These parameters will helps to make DC motors MIL simulations with fine models nearest to reality.

5.2 DC Motors EMC

In this section the EMC of the 2 DC motors are build, and compared the results with a model which mimic the real motors present in GoPiGo3 robot (the fine model). All is done by using Simulink from Matworks.
First fine models are considered, trying to insert inside also the possible disturbances that can happen in reality.
Then a simplified model of the DC motor is created from the fine model (the embedded model EM), simple in such a way it can be easily implemented in Raspberry board of the robot.
Notations: In fine models, the output speed in SS equations is the one not reduced by gearbox $\omega_m(t)$. In EMC models the output speed is considered always reduced by gearbox $\omega(k)$, and the back-electromotive force parameter is the one related to wheel speeds $k'_v = k_v N$ (affected by gearbox reduction N as already explained in Assumptions 5.1).

5.2.1 Fine model

This model will be used only for simulation purposes (MIL tests), since when the control is applied in real there will be directly the real inputs/outputs of the plant.
In Simulink instead of using the Transfer Function (TF) representation, it was used the State Space (SS) one, with Matlab function blocks: indeed for EMC using SS equations is better for code generation and to deal with a variable timestamp (is simple modify the sampling time in the equations), the same solution is adopted for fine model. In this case DC motor is composed by 2 ODE, for the 2 states $i_a(t)$ and $\omega_m(t)$, which refer respectively to

electrical and mechanical parts. The equations are the following:

$$\begin{cases} \dot{x}_1(t) = A_1 x_1(t) + B_1 u_1(t) \Rightarrow \dfrac{di_a(t)}{dt} = -\dfrac{R_a}{L_a} i_a(t) + \dfrac{1}{L_a}\left(V_a(t) - e(t)\right) \\ \dot{x}_2(t) = A_2 x_2(t) + B_2 u_2(t) \Rightarrow \dfrac{d\omega_m(t)}{dt} = -\dfrac{\beta_{eq}}{J_{eq}} \omega_m(t) + \dfrac{1}{J_{eq}}\left(T_m(t) - T_r(t)\right) \end{cases} \tag{5.10}$$

$$e(t) = k_v \omega_m(t)$$

$$T_m(t) = k_t i_a(t)$$

T_r is the resistive torque due to friction/load/other disturbances. A sketch in terms of simple elementary blocks is shown in Figure 5.28.

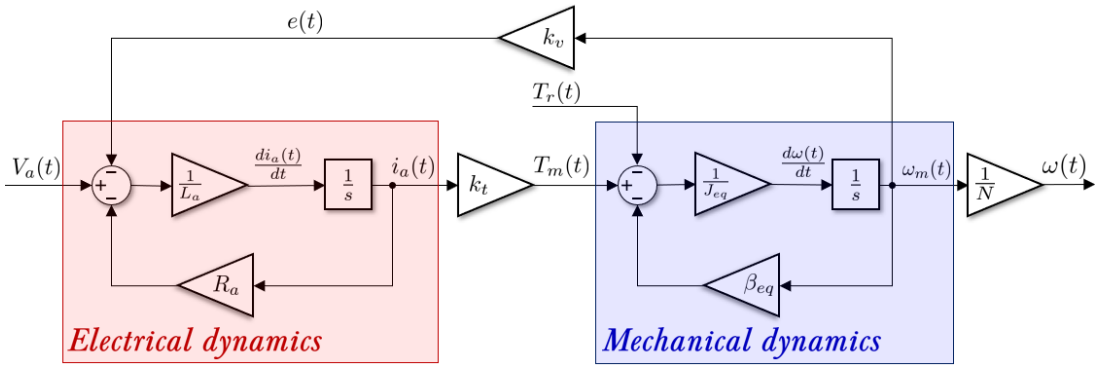


Figure 5.28: DC motor SS fine model

5.2.2 EMC model - 1st order disturbance

Controllable and disturbance dynamics

Since the current i_a (and consequently T_m) cannot be measured, it is convenient to treat the electrical part block of DC motor as neglected dynamics in EM model. In this way, only 1 state remains, the output speed of the motor $\omega(t)$.

A first order disturbance x_d is considered, like in Figure 5.29 taken from [6], including parametric uncertainties, neglected dynamics and other disturbances. In the figure w_u is the non-causal noise component and w_d is the disturbance state. The Σ block is a *Discrete time integrator*, composed by a delay with a positive unitary feedback, with transfer function $\frac{T}{z-1}$ (following Forward Euler integration method).

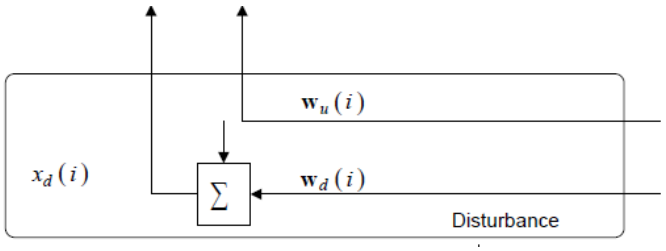


Figure 5.29: First order disturbance $\overline{w}$

Source: [6]

The SS mechanical equation in Equation (5.10) needs to be modified in discrete-time domain for EMC.

First step, since the real values β_{eq} and J_{eq} are the most difficult to be found, the usage of τ_m and k'_v is preferred. Hence the controllable and disturbance equations become, in continuous time domain:

$$\begin{aligned}\dot{\omega}(t) &= -\frac{1}{\tau_m}\omega(t) + \frac{1}{\tau_m k'_v}V(t) + \overline{w}_1(t) + x_d(t)\\ \dot{x}_d(t) &= x_d(t) + \overline{w}_2(t)\end{aligned} \tag{5.11}$$

In matrix form, after conversion in DT domain, SS equations of controllable and disturbance dynamics can be written following the form in Equation (2.2) and Equation (2.3), considering state vector as $x^T(k) = [x_c = \omega, x_d](k)$:

$$\begin{aligned}A &= \begin{bmatrix} A_c & H_c \\ 0 & A_d \end{bmatrix} = \begin{bmatrix} -\frac{1}{\tau_m}T + 1 & T \\ 0 & T+1 \end{bmatrix}, \quad B = \begin{bmatrix} B_c \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{T}{\tau_m k'_v} \\ 0 \end{bmatrix}, \quad G = \begin{bmatrix} G_c \\ G_d \end{bmatrix} = \begin{bmatrix} T & 0 \\ 0 & T \end{bmatrix},\\ C &= \begin{bmatrix} C_c & C_d \end{bmatrix} = \begin{bmatrix} 1 & 0 \end{bmatrix}, \quad F = \begin{bmatrix} F_c & 0 \end{bmatrix} = C\end{aligned} \tag{5.12}$$

Reference dynamics

CL loop equations Equation (2.5) can be used to find control reference dynamics for EMC DC motor, using pole-placement tecnique. K_R and N_R are the matrices to be designed.
$K_R = k$ can be found with pole placement technique by recovering the discrete time eigenvalues p_R (in this case only 1) that satisfies the following desired characteristic polynomial:

$$\det[\lambda\mathbb{I} - A_R] = (\lambda - p_R)$$

Hence we obtain:

$$k = \frac{A_c - p_R}{B_c}$$

The reference dynamics must be quite slow to allow the overall EM system to track the reference $\overline{r}$ without difficulties. So the p eigenvalue must be quite near to zero to obtain this behaviour: for example $p_R \approx -2.57$ (in CT domain) is a proper value.
N_R can be found when K_R is recovered by exploiting Equation (2.6).
In addition, since the real DC motors can work in a limited voltage range (of $V \approx \pm 11.5\,\mathrm{V}$ in the case of voltage taken from home network converted, and $V < 9.6\,\mathrm{V}$ using $8 \times 1.2\mathrm{V}$ batteries), a saturation on $\overline{u}$ is needed. For example, if the max voltage available for the motors is $V = \pm 11.5\mathrm{V}$:

$$\bar{u} = \begin{cases} 11.5\mathrm{V} & \text{if } \bar{u} > 11.5\mathrm{V} \\ -11.5\mathrm{V} & \text{if } \bar{u} < -11.5\mathrm{V} \end{cases}$$

Noise estimator

First we need to understand if a static or dynamic feedback Noise estimator L is needed: this depends on the dimensions of all the states $\dim x = n_x = 2$ and of disturbances $\dim w = n_w = 2$. By referring to [1] and Appendix of [2], if $n_x = n_w$ we fall in the case of static feedback L. So we can find the disturbances as $\overline{w} = Le_m = L(y - \hat{y}_m)$, with $\hat{y}_m$ as the output of EM and y as the fine model output.
$L^T = [l_1, l_2] \in \mathbb{R}^{2\times 1}$ matrix, the components can be found by considering the coefficients $a_{cl,i}$ of the characteristic polynomial of the closed loop matrix $A - GLC$, computed with $\det[(A - GLC) - \lambda\mathbb{I}]$, and equalizing them with the coefficients $a_{n,i}$ of a characteristic polynomial with discrete time eigenvalues $[p_{n1}, p_{n2}]$ decided by us with

$Re(\lambda) < 1$, to guarantee asymptotic internal stability. The 2 characteristic polynomials are:

$$\begin{cases} \lambda^2 + a_{cl,2}\lambda + a_{cl,3} & \textit{Closed loop char. polynomial} \\ \lambda^2 + a_{n,2}\lambda + a_{n,3} = (\lambda - p_{n1})(\lambda - p_{n2}) & \textit{Desidered char. polynomial} \end{cases}$$

The coefficients of L depends on sampling time T which is variable at every step. The components in function of model parameters k'_v, τ_m are:

$$\begin{cases} l_1 = \dfrac{(2 + T + a_{n,2})\,\tau_m - T}{T\tau_m} \\ l_2 = \dfrac{2T + T^2 + (1+T)\,a_{n,2} + a_{n,3} + 1}{T^2} \end{cases} \tag{5.13}$$

Control Law

For the control law we can follow the rules seen in Section 2.1, using Equation (2.11) and Equation (2.12). After some computations $Q = 0$ and $M = k'_v\tau_m$. Instead for K matrix used for tracking the controllable states to a reference one, it was experimented that only a proportional gain is not sufficient, since we never reach a zero tracking error. For this reason a discrete Proportional-Integral (PI) control was selected.
To find the right coefficients of $K = [k_p, k_i]$ we need to study the closed loop system including the controllable dynamics and the PI controller, like in Figure 5.30. $x_1(k) = \hat{x}_c(k)$ and $x_2(k)$ is the new state introduced by the integral action of PI controller.

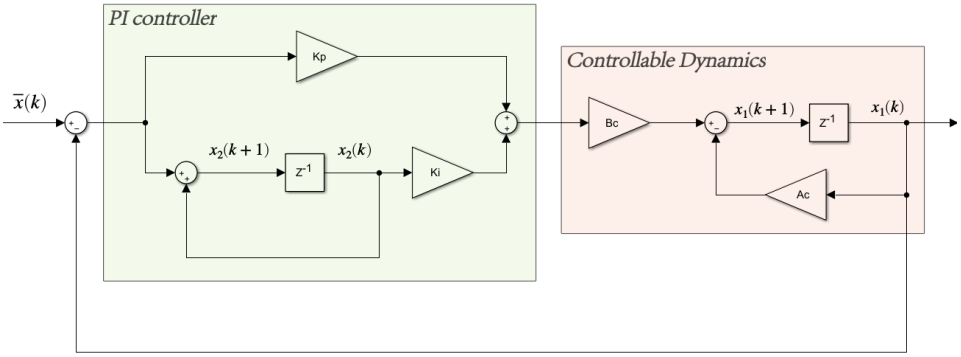


Figure 5.30: Closed Loop System with PI controller

In matrix form:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}(k+1) = \underbrace{\begin{bmatrix} -A_c - k_pB_c & k_iB_c \\ -1 & 1 \end{bmatrix}}_{A_{int}} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}(k) + \underbrace{\begin{bmatrix} k_pB_c \\ 1 \end{bmatrix}}_{B_{int}} V_r(k) \tag{5.14}$$

In order to get k_p and k_i coefficients we can use the pole placement technique and set the eigenvalues of A_{int} to some desired values inside the unit circle for stability.
If $a_{c1,des}$ and $a_{c2,des}$ are the coefficients of characteristic polynomial defined by desired eigenvalues $p_{c,i} = [p_{c1}, p_{c2}]$, finally:

$$\begin{aligned} k_p &= \frac{a_{c1,des} + A_c + 1}{B_c} \\ k_i &= \frac{B_c k_p - A_c + a_{c2,des}}{B_c} \end{aligned} \tag{5.15}$$

Like L coefficients, also k_p and k_i changes depending on the sampling time considered T.

To make the first condition of Equation (2.12) satisfied, eigenvalues p_i obtained with pole placement must be inside the unit circle.

5.2.3 EMC model - 2nd order disturbance

It's possible that in practice the plant has other disturbances in addition to the electrical and mechanical dynamics which may be not detected using 1st order disturbance: for this reason a 2nd order disturbance is build for the EMC DC motor model in this section.

Controllable and disturbance dynamics

As in Section 5.2.2 the electrical part of DC motor is considered as neglected dynamics in EM model. In this way, only 1 state remains, the output speed of the motor $\omega(k)$.
Even 2nd order disturbance model part includes parametric uncertainties, neglected dynamics and other disturbances, but now there are 2 delays and 3 sources of disturbance $\overline{w}(k)$, as it can be seen in Figure 5.31. $\overline{w}_0(k)$ enters directly in the controllable part of model and we cannot estimate it since it's not causal. Instead $\overline{w}_1(k)$ and $\overline{w}_2(k)$ are entering inside the 2 disturbance state equations, with states $x_{d1}(k)$ and $x_{d2}(k)$: we can estimate them with a noise estimator because they depend on past inputs for the presence of delays.
The $\sum$ block is a Discrete Integrator, i.e. a delay with a positive unitary feedback, with transfer function $\frac{T}{z-1}$.

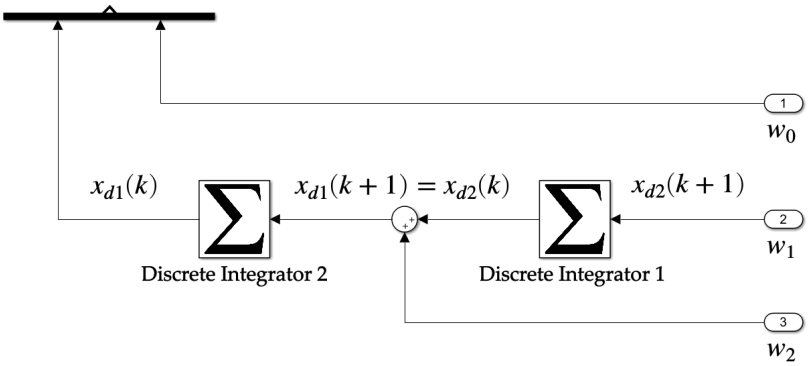


Figure 5.31: Second order disturbance $\overline{w}$

The SS mechanical equation is similar to Equation (5.11), but with the addition of a new disturbance state:

$$\begin{aligned}
\dot{\omega}(t) &= -\frac{1}{\tau_m}\omega(t) + \frac{1}{\tau_m k'_v}V(t) + \overline{w}_0(t) + x_{d1}(t) \\
\dot{x}_{d1}(t) &= x_{d1}(t) + x_{d2}(t) + \overline{w}_2(t) \\
\dot{x}_{d2}(t) &= x_{d2}(t) + \overline{w}_1(t)
\end{aligned} \tag{5.16}$$

The new states are $x = [x_c = \omega, x_{d1}, x_{d2}]^T$. Now it is needed to convert the state space equations from continous to discrete time domain, obtaining the following set of equations:

$$\begin{aligned}
&\omega(k+1) = \left(-\frac{1}{\tau_m}T + 1\right)\omega(k) + \frac{T}{\tau_m k'_v}V(k) + d(k)\ , \qquad d(k) = T\left(\overline{w}_0(k) + x_{d1}(k)\right) \\
&x_{d1}(k+1) = (T+1)x_{d1}(k) + T(x_{d2}(k) + \overline{w}_2(t)) \\
&x_{d2}(k+1) = T(x_{d2}(k) + \overline{w}_1(k))
\end{aligned} \tag{5.17}$$

The matrices using the form in Equation (2.1) are:

$$A = \begin{bmatrix} A_c & H_c \\ 0 & Ad \end{bmatrix} = \left[\begin{array}{c|cc} -\frac{T}{\tau_m}+1 & T & 0 \\ \hline 0 & T+1 & T \\ 0 & 0 & T+1 \end{array}\right], \quad B = \begin{bmatrix} B_c \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{T}{\tau_m k_v'} \\ 0 \\ 0 \end{bmatrix}, \quad G = \begin{bmatrix} G_c \\ G_d \end{bmatrix} = \left[\begin{array}{ccc} T & 0 & 0 \\ \hline 0 & T & 0 \\ 0 & 0 & T \end{array}\right] \tag{5.18}$$
$$C = \begin{bmatrix} C_c & C_d \end{bmatrix} = \left[\begin{array}{c|cc} 1 & 0 & 0 \end{array}\right], \qquad F = \begin{bmatrix} F_c & F_d \end{bmatrix} = C$$

Reference dynamics

The number of controllable states $x_c(k)$ is not changed passing from 1st to 2nd order disturbance model (it's only one, $x_c(k) = w(k)$): since the reference dynamics depends only on $x_c(k)$, it's the same of Section 5.2.2.

Noise estimator

In this case a static feedback noise estimator is sufficient to estimate the noise disturbances $\overline{w}_1$ and $\overline{w}_2$ ($\overline{w}_0$ is impossible to estimate due to causality). Indeed the dimensions of all the states is $\dim x = n_x = 3$ and the one of disturbances is $\dim w = n_w = 3$, so by referring to [1], since $n_x = n_w$, we fall in the case of static feedback L.
Another check can be made by following the Appendix on output feedback pole placement [2]. To have a static feedback a necessary (but not sufficient) condition is $n_w \times n_y \geq n$, where $n_y = \dim y = 1$ is the dimension of the known output, which in our case is the DC motor output speed.
So we can find the disturbances as $\overline{w} = Le_m = L(y - \hat{y}_m)$, with $\hat{y}_m$ as the output of EM and y as the fine model output. In this case $L^T = [l_1, l_2, l_3] \in \mathbb{R}^{3\times 1}$ matrix.
L components can be found making equal the coefficients $a_{cl,i}$ of the characteristic polynomial of the closed loop matrix $A_{CL} = A - GLC$, computed with $\det[A_{CL} - \lambda\mathbb{I}]$, and the ones $a_{n,i}$ of a characteristic polynomial with discrete time eigenvalues $[p_{n1}, p_{n2}, p_{n3}]$ decided by us with $Re(\lambda) < 1$, to guarantee asymptotic internal stability.
The 2 characteristic polynomials are:

$$\begin{cases} \lambda^3 + a_{cl,2}\lambda^2 + a_{cl,3}\lambda + a_{cl,4} & \textit{Closed loop char. polynomial} \\ \lambda^3 + a_{n,2}\lambda^2 + a_{n,3}\lambda + a_{n,4} = (\lambda - p_{n1})(\lambda - p_{n2})(\lambda - p_{n3}) & \textit{Desidered char. polynomial} \end{cases}$$

The coefficients of L depends on sampling time T which can be variable at every step. If they are highlighted τ_m, k_v DC motor model parameters they are:

$$\begin{cases} l_1 = \dfrac{(2T + a_{n,2} + 3)\,\tau_m - T}{T\tau_m} \\ l_2 = \dfrac{2(T+1)a_{n,2} + a_{n,3} + 6T + 3T^2 + 3}{T^2} \\ l_3 = \dfrac{\left(1 + 2T + T^2\right)a_{n,2} + (1+T) + a_{n,3} + a_{n,4} + 3T + 3T^2 + T^3 + 1}{T^3} \end{cases} \tag{5.19}$$

Control law

The control law form is again the same of Equation (2.11). In this particular case the matrix $K = [k_p, k_i]$ is exactly the same of Section 5.2.2 since it only depends on the controllable dynamics, which remains unchanged.
Instead the matrices Q and M need to be recomputed, since the number of disturbance states $x_d(k)$ passes from

1 to 2. The 2nd condition of Equation (2.12) needs to be satisfied to find these 2 matrices, it follows that:

$$\begin{bmatrix} Q \\ M \end{bmatrix} = \begin{bmatrix} q_1 & q_2 \\ m_1 & m_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ \tau_m k_v' & 0 \end{bmatrix} \tag{5.20}$$

Matrices Q, M change dimensions, with $Q \in \mathbb{R}^{1\times 2}$ but with components again equal to zero, $M \in \mathbb{R}^{1\times 2}$ but only the first component m_1 is important, since $m_2 = 0$.
It seems that nothing change from 1st order disturbance model, but it is not. Indeed m_1 depends only on $x_{d1}(k)$ state, but in the state equation $x_{d1}(k+1)$ in Equation (5.17) there is a dependence not only on $x_{d1}(k)$ but also on $x_{d2}(k)$: the effect of the second disturbance component is inside the first one.

Chapter 6

DC motors EMC Model-in-the-Loop (MIL) tests

Theoretical models explained in Chapter 5 are now translated in Simulink block models to be tested. Using MIL tests, both DC motor plant and EMC are run in simulation using Simulink. Before showing testing results, some general settings need to be discussed.
Notation: From now on, in all MIL and RHIT tests (also in the following chapters) the targets are referred with a "tilde" on the symbol.

6.1 General MIL settings on Simulink

DC motor EMC is quite easy to be implemented in Simulink: as already explained, whole EMC is a triggered sub-system, where discrete-time is decided by an external stair generator signal suitably built. Internal blocks are Matlab functions where Timestamp can be easily added as input.
CT eigenvalues obtained with pole-placement technique are the same for all tests:

Eigenvalue type	Continuous eigenvalues
Reference μ_R	-2.5647
Noise estimator μ_N	$-14.3844 \times 2, -14.3835$
Control μ_K	-2.5647×2

Table 6.1: Continous eigenvalues DC motor EMC - MIL tests

Instead regarding DC motor fine model, the implementation is more complicated.
A requirement for fine model is to mimic as much as possible the real motor behaviour. Real motor has quite complex dynamics due to inertia, frictions etc., especially due to gearbox part. In addition motors are driven by H-bridges to control their direction and speed, a PWM generator is responsible to convert battery/supply voltage in input voltage at the motor terminals, and encoders are based on Hall effect magnetic field sensors. Therefore, knowing only inputs/outputs (at wheel side) of the motors, it's very difficult to define in a precise way the motor behaviour in simulation. Parameters identification results to be not sufficient to exploit all dynamics, because it is only based on mechanical and electrical model of a DC motor.
As first attempt Simscape[2] built-in models for Hall effect encoders, PWM generation and H-bridge are added to

[2]Simscape is a MathWorks toolbox to add already built multidomain physical components, [25]

mechanical/electrical part of DC motors in simulation: they lead to very complex fine model, too heavy to be simulated in a short time and sometimes giving unexpected results. A simplified complete fine model is then exploited, using only quantizers and Matlab function blocks. Its structure is presented in Figure 6.1:

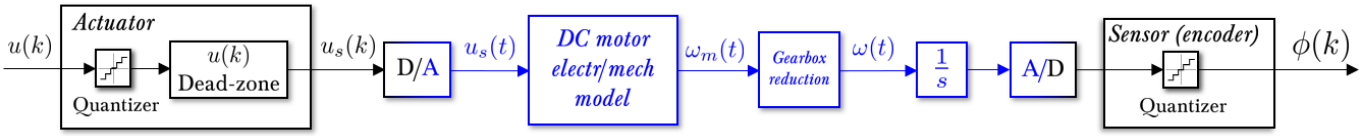


Figure 6.1: DC motor fine model Simulink scheme

Already built model is electrical and mechanical part of DC motor, with gearbox output speed reduction $1/N = 1/120$, in CT domain (highlighted in blue). The other blocks are simplified versions of actuator and motor encoder sensors, in DT (in black):

- The control input from DC motor EMC $u(k)$ passes through simulated actuator, composed by a *quantizer* and an *input dead-zone block*.
  Quantizer takes into account that input PWM in GoPiGo3 DC motors SW code is saved into an *unsigned 8-bit char*, or `uint8_t`, ranging 0–255. If P is the PWM duty cycle percentage, the values are divided in the following manner:

$$n = \begin{cases} 0\text{–}100 & \Rightarrow \quad P = [0, 100]\ \% \\ 101\text{–}127 & \Rightarrow \quad P = 100\ \% \\ 128\text{–}227 & \Rightarrow \quad P = [0, -100]\ \% \quad \text{(In decreasing order)} \\ 228\text{–}255 & \Rightarrow \quad P = -100\ \% \end{cases}$$

  Hence the actuator quantizer in Simulink divides in 100 parts the motor controlled voltage $u(k)$: for example if maximum voltage is given by network supply with a converter and equal to 11.5 V, the minimum input resolution will be 0.115 V.
  Dead-zone block is added because no output speed of DC motors at the wheel side is present when input voltage is too low, due to construction frictions and inertia.
  A comparison between input voltage and output speed is done in open-loop robot system configuration, for both left and right motors. Voltage V is linearly increased from 0 to $V_{max} \approx 11.5$ V, resulted output speed starts to increase only when $u > 2$ V approximately for both motors. This can be seen in the next 2 plots:

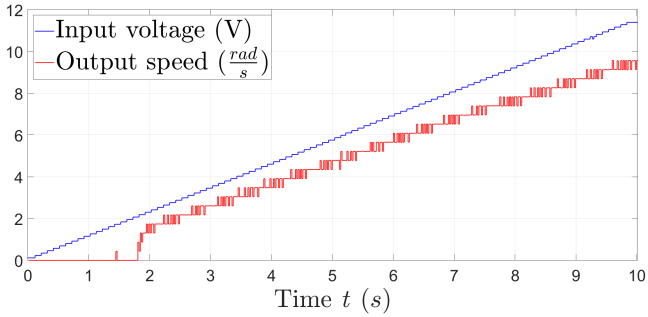


Figure 6.2: Left motor input voltage dead-zone

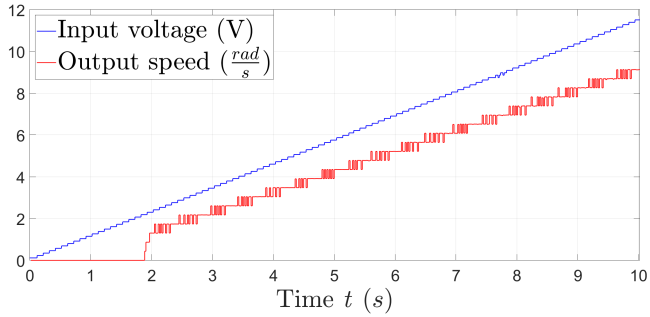


Figure 6.3: Right motor input voltage dead-zone

This is valid only when motors are still at the beginning. Instead if motors are already running, the voltage

when motors stop is lower than $2\,\mathrm{V}$, standing on a value of $1\text{–}1.5\,\mathrm{V}$. This is not experienced using an open-loop test but making practical RHIT.
These values might slightly change depending on output speed conditions, but they remains more or less in the above ranges. Implementation in Simulink is done using simple conditional "if" statements.

- The CT position, obtained by reducing $\omega(k)$ with the gearbox block and integrator, also needs to be quantized because encoder resolution is $0.5° = 0.0087\,\mathrm{rad}$. A quantizer Simulink block is sufficient to this aim.

6.2 General settings on Simulink MIL plot results

In MIL tests, the DC motor EMC must work at least in no-load conditions (i.e. robot wheels not touching the ground), because in real load conditions there's a load torque acting which leads to a more demanding voltage input to obtain the same output speed, which is difficult to be simulated.
The variable Timestamp used in MIL is exactly the same of corresponding RHIT test (done in Chapter 7), in order to make comparisons. In addition both left and right motors EMC are simulated splitting EM and Control parts, because in many RHIT it will be the best solution to reduce model and tracking errors. Further details are explained previously in Section 4.3.1.
Every results set is divided in 2 parts. The first plots set corresponds to MIL only tests using Simulink, with output speeds $y, y_{ref}, \hat{y}_m$ and relative model and tracking errors e_m, e_{trk}, control input voltage u and Timestamp for EM and Control blocks of EMC.
The second plots set is a comparison between MIL and RHIT: the output speeds $y, y_{ref}, \hat{y}_m$ for both tests shown in the same plot, the errors between measured outputs $e_y = y_{MIL} - y_{RHIT}$ and estimated outputs $e_{\hat{y}_m} = \hat{y}_m^{MIL} - \hat{y}_m^{RHIT}$, and the comparison of simulated EM/Control Timestamps. It'll be show that in the 2 tests output speeds follow almost the same behaviour, in the same condition of DT timestamp (the last means that the procedure explained in Section 4.1.1 to obtain a variable timestamp in simulation is reliable).

6.3 Simulink MIL results - Left motor

In Chapter 5 are discussed different versions of Left DC motor EMC, in which main parameters change on the basis of system identification outcomes. Version 1.2 is built only after RHIT on both 2 motors with robot moving on the ground, so all tests are done in load conditions. For this reason in simulation was considered only Version 1.1, which is basically the same but with slight parameter modifications.
All tests are made considering different target speeds $\tilde{\omega}_L = [6, 4]\ \frac{\mathrm{rad}}{\mathrm{s}}$ and $\tilde{\omega}_L = [8, 3, 5.5, 7]\ \frac{\mathrm{rad}}{\mathrm{s}}$, with presence or not of disturbance rejection.
From comparison between MIL and RHIT in every results set is immediate the similarities between y and $\hat{y}_m$ signals. The best comparison is when no disturbance rejection is present in Control Law, as it can be seen in Figure 6.8 and Figure 6.10: the peak reached by y and $\hat{y}_m$ is almost the same, meaning that the assumptions and simplifications done in Section 6.1 for Fine DC motor models are consistent.

6.3.1 No disturbance rejection, target output speed $\tilde{\omega}_L = [6, 4]$ $\frac{\text{rad}}{\text{s}}$

MIL simulation test results:

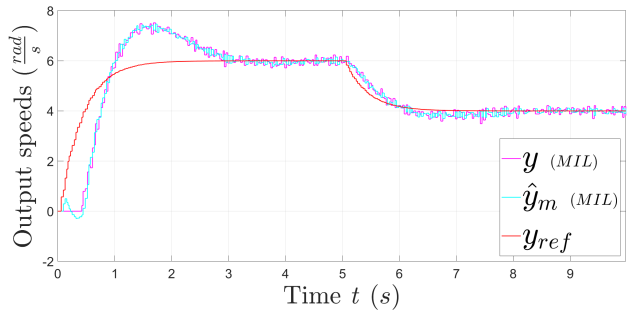


Figure 6.4: Left DC motor outputs y, y_{ref} and $\hat{y}_m$

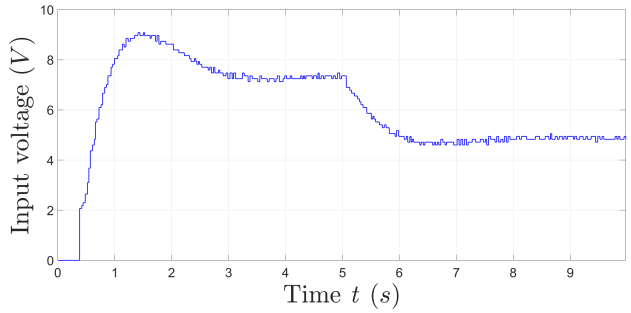


Figure 6.6: Left DC motor input voltage u

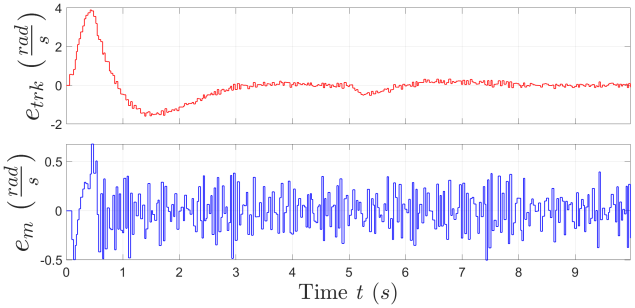


Figure 6.5: Left DC motor tracking error e_{trk} and model error e_m

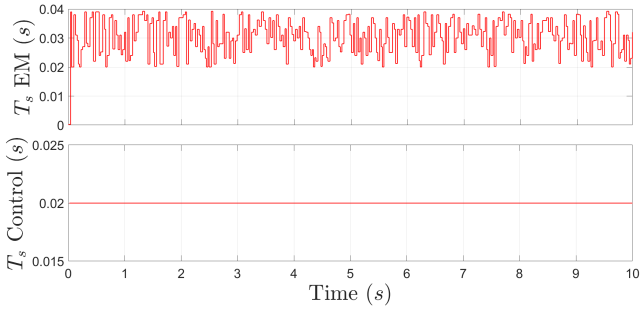


Figure 6.7: Left DC motor timestamp for EM and Control part of EMC

Comparison between MIL and RHIT:

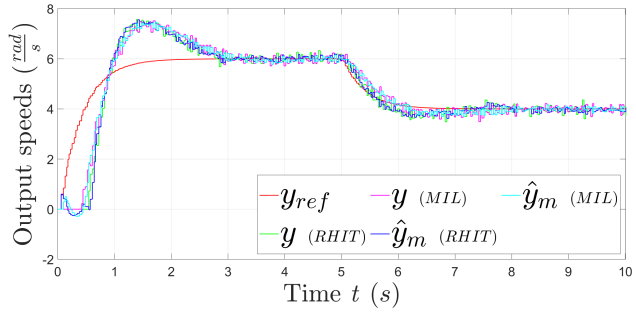


Figure 6.8: Left DC motor outputs y, y_{ref} and $\hat{y}_m$ - Comparison between MIL and RHIT

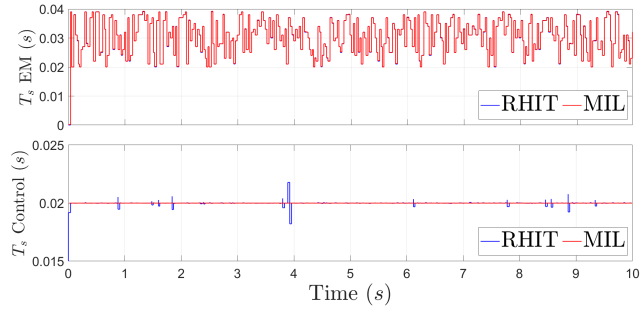


Figure 6.9: Left DC motor timestamp for EM and Control part of EMC - Comparison between MIL and RHIT

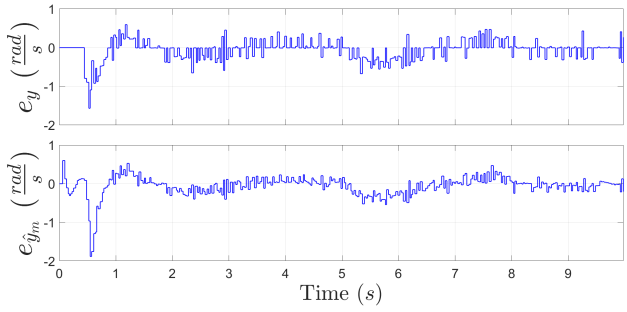


Figure 6.10: Left DC motor MIL and RHIT e_y (above) and $e_{\hat{y}_m}$ (below) error difference

6.3.2 Disturbance rejection, target output speed $\tilde{\omega}_L = [8, 3, 5.5, 7]$ $\frac{\text{rad}}{\text{s}}$

MIL simulation test results:

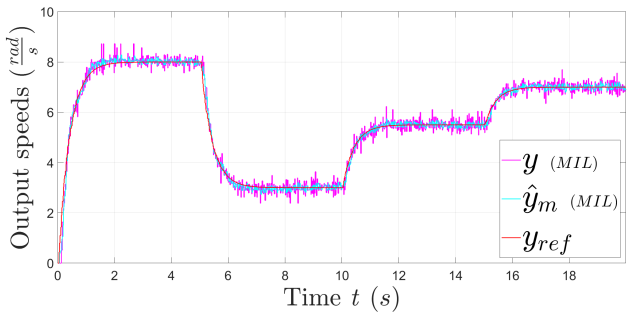


Figure 6.11: Left DC motor outputs y, y_{ref} and $\hat{y}_m$

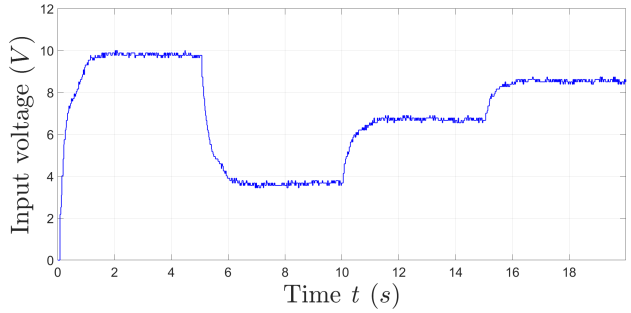


Figure 6.13: Left DC motor input voltage u

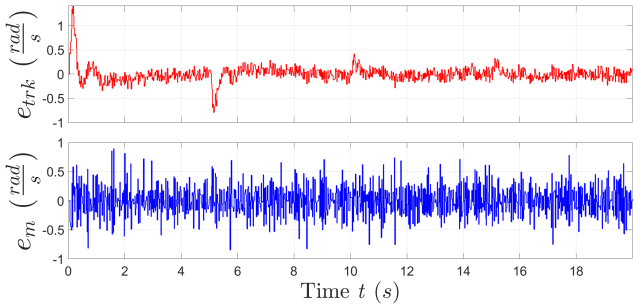


Figure 6.12: Left DC motor tracking error e_{trk} and model error e_m

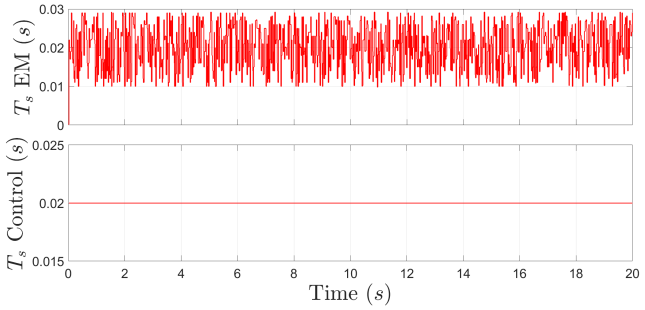


Figure 6.14: Left DC motor timestamp for EM and Control part of EMC

Comparison between MIL and RHIT:

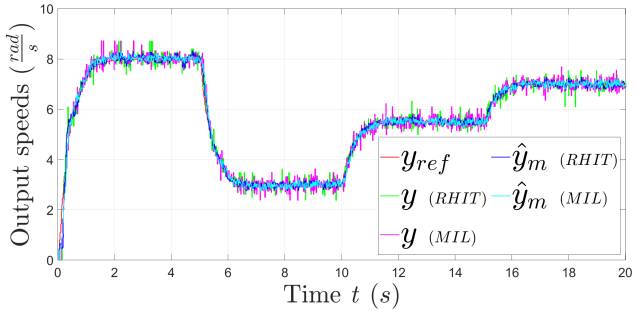


Figure 6.15: Left DC motor outputs y, y_{ref} and $\hat{y}_m$ - Comparison between MIL and RHIT

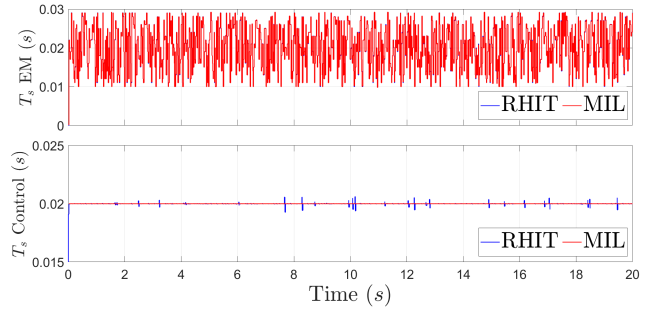


Figure 6.16: Left DC motor timestamp for EM and Control part of EMC - Comparison between MIL and RHIT

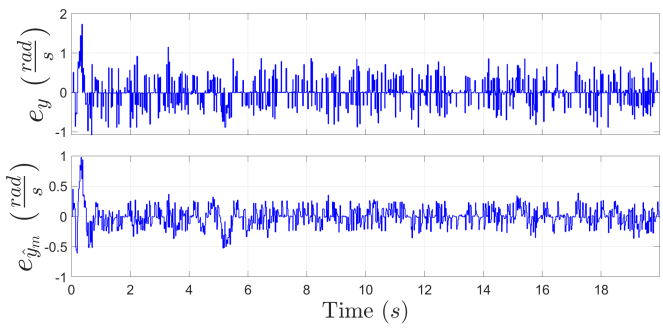


Figure 6.17: Left DC motor MIL and RHIT e_y (above) and $e_{\hat{y}_m}$ (below) error difference

6.4 Simulink MIL results - Right motor

For right motor it was used Version 1.2, which is the only one built from Chapter 5. Again different target speeds are considered, $\tilde{\omega}_R = [6, 4]\ \frac{\text{rad}}{\text{s}}$ and $\tilde{\omega}_R = [8, 3, 5.5, 7]\ \frac{\text{rad}}{\text{s}}$, and comparison with presence or not of disturbance rejection.

Also with right DC motor EMC is well evident the similarities with MIL and RHIT, then the assumptions Section 6.1 to make fine model in simulation are coherent.

Again as proof are taken the comparisons when no disturbance rejection in control law is present. First test is when target speed is $\tilde{\omega}_R = [6, 4]\ \text{rad/s}$ with outputs in Figure 6.36 and errors $e_y, e_{\hat{y}_m}$ in Figure 6.38: in this case all the peaks obtained with RHIT are simulated in a precise way.

Another test with different target output speeds ($\tilde{\omega}_R = [8, 3, 5.5, 7]\ \frac{\text{rad}}{\text{s}}$ in Figure 6.43 and Figure 6.45) show also the presence of saturation of input voltage when it reaches $\approx 11.5\,\text{V}$, which is successfully simulated with MIL tests.

6.4.1 Disturbance rejection, target output speed $\tilde{\omega}_R = [6, 4]$ $\frac{\text{rad}}{\text{s}}$

MIL simulation test results:

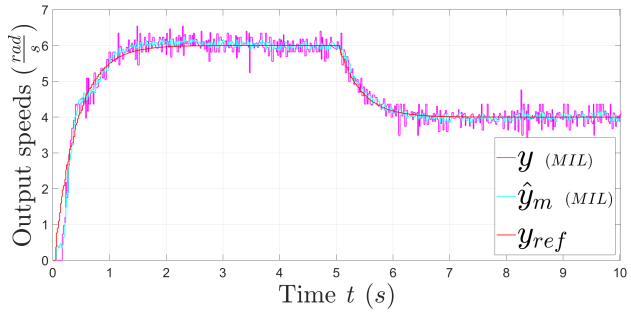


Figure 6.18: Right DC motor outputs y, y_{ref} and $\hat{y}_m$

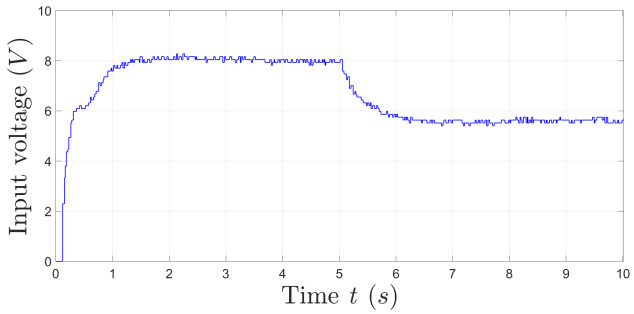


Figure 6.20: Right DC motor input voltage u

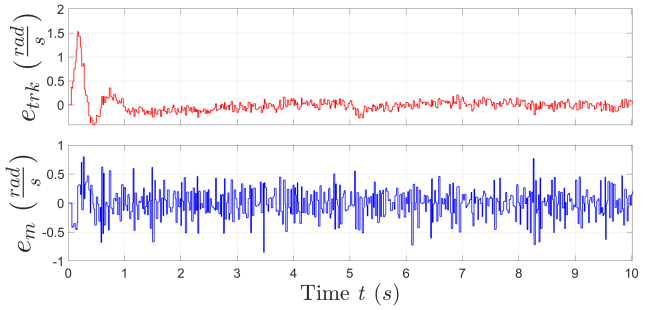


Figure 6.19: Right DC motor tracking error e_{trk} and model error e_m

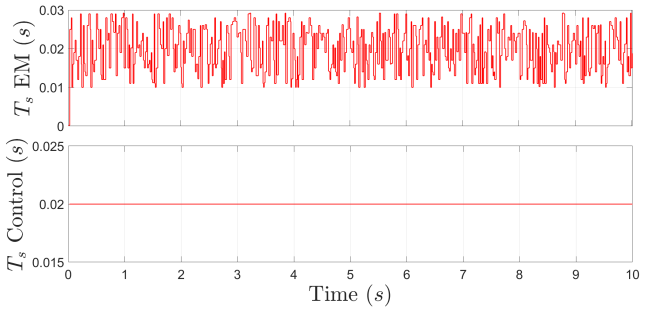


Figure 6.21: Right DC motor timestamp for EM and Control part of EMC

Comparison between MIL and RHIT:

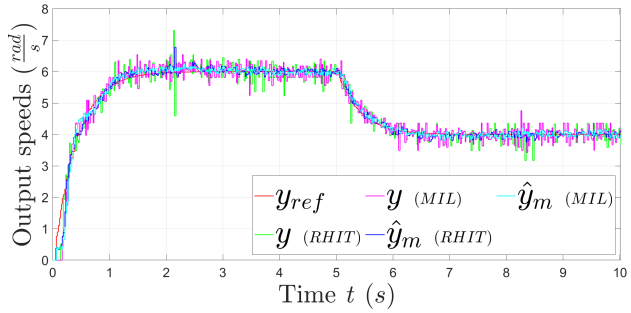


Figure 6.22: Right DC motor outputs y, y_{ref} and $\hat{y}_m$ - Comparison between MIL and RHIT

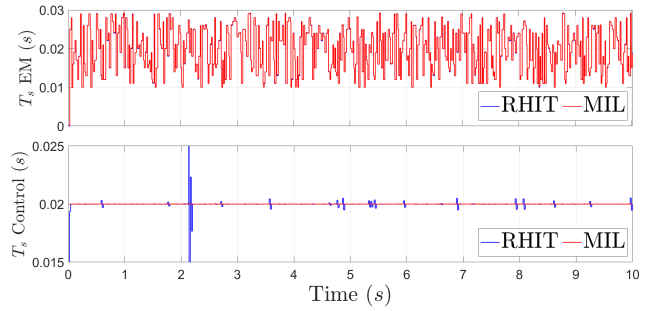


Figure 6.23: Right DC motor timestamp for EM and Control part of EMC - Comparison between MIL and RHIT

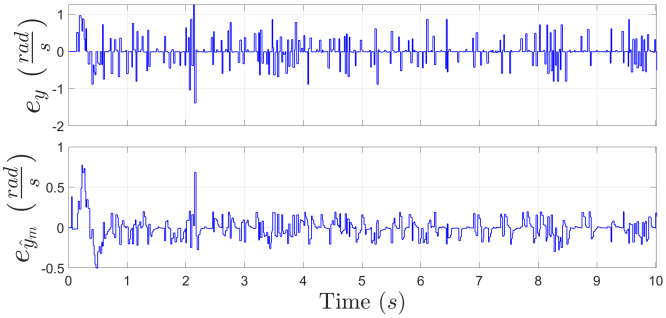


Figure 6.24: Right DC motor MIL and RHIT e_y (above) and $e_{\hat{y}_m}$ (below) error difference

6.4.2 Disturbance rejection, target output speed $\tilde{\omega}_R = [8, 3, 5.5, 7]$ $\frac{\text{rad}}{\text{s}}$

MIL simulation test results:

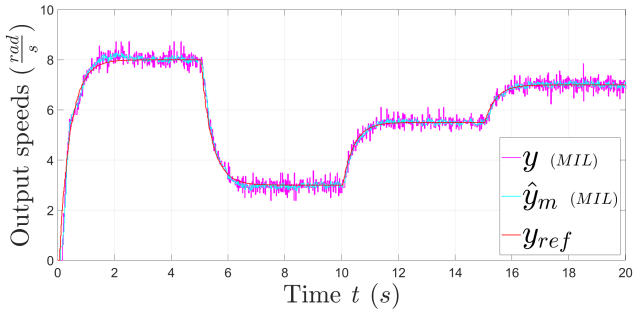


Figure 6.25: Right DC motor outputs y, y_{ref} and $\hat{y}_m$

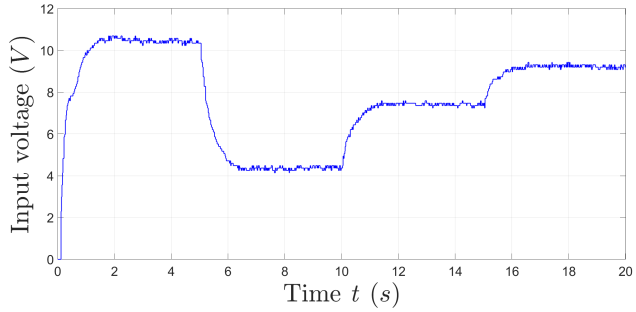


Figure 6.27: Right DC motor input voltage u

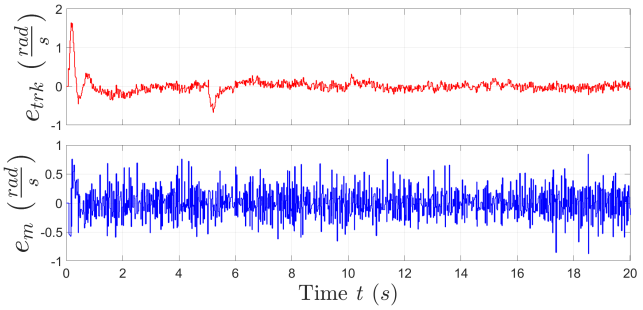


Figure 6.26: Right DC motor tracking error e_{trk} and model error e_m

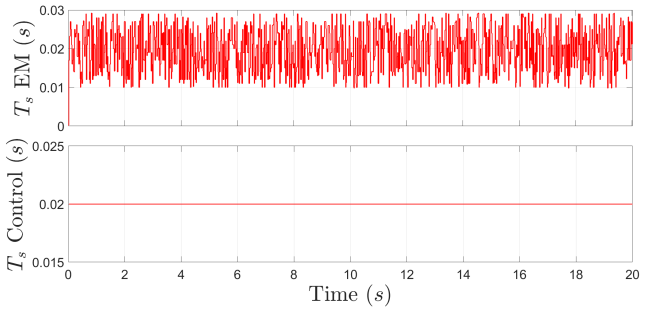


Figure 6.28: Right DC motor timestamp for EM and Control part of EMC

Comparison between MIL and RHIT:

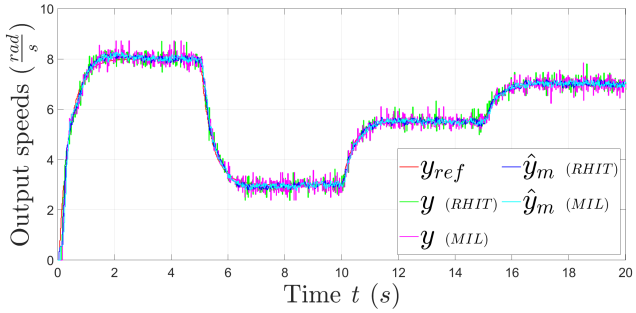


Figure 6.29: Right DC motor outputs y, y_{ref} and $\hat{y}_m$ - Comparison between MIL and RHIT

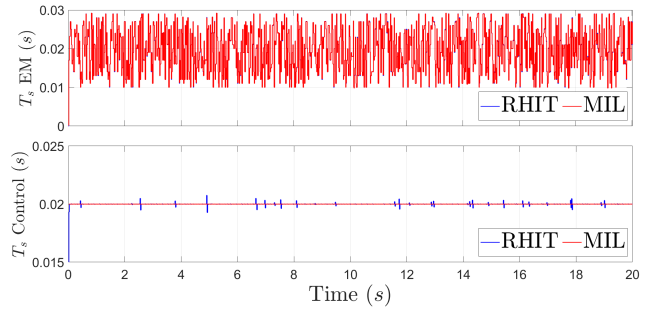


Figure 6.30: Right DC motor timestamp for EM and Control part of EMC - Comparison between MIL and RHIT

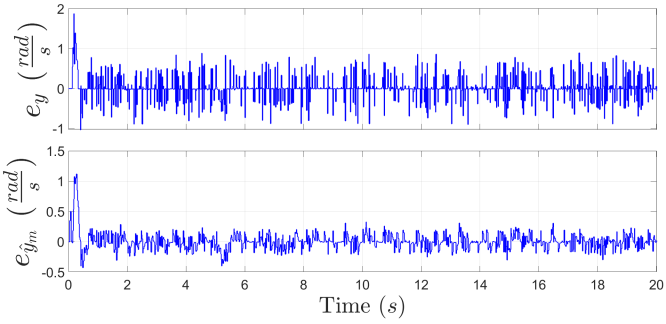


Figure 6.31: Right DC motor MIL and RHIT e_y (above) and $e_{\hat{y}_m}$ (below) error difference

6.4.3 No disturbance rejection, target output speed $\tilde{\omega}_R = [6, 4]$ $\frac{\text{rad}}{\text{s}}$

MIL simulation test results:

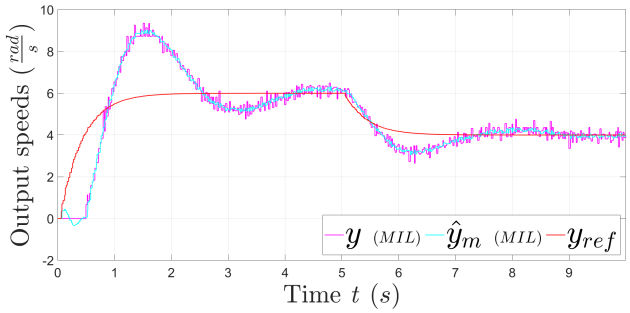


Figure 6.32: Right DC motor outputs y, y_{ref} and $\hat{y}_m$

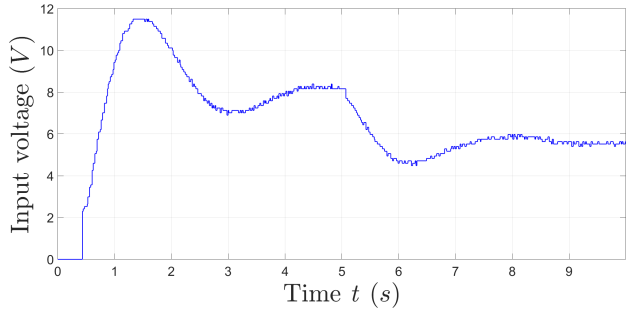


Figure 6.34: Right DC motor input voltage u

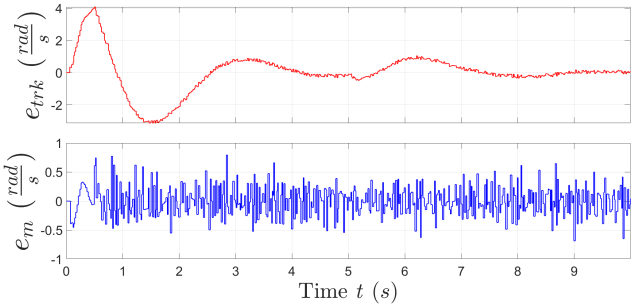


Figure 6.33: Right DC motor tracking error e_{trk} and model error e_m

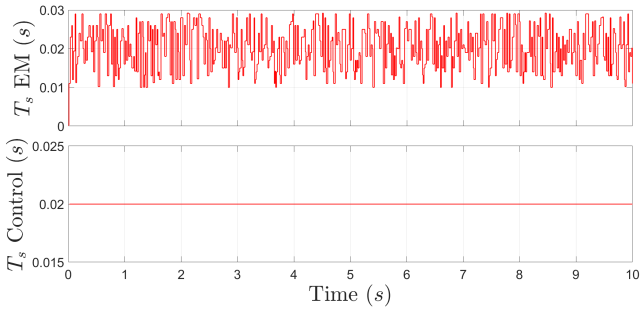


Figure 6.35: Right DC motor timestamp for EM and Control part of EMC

Comparison between MIL and RHIT:

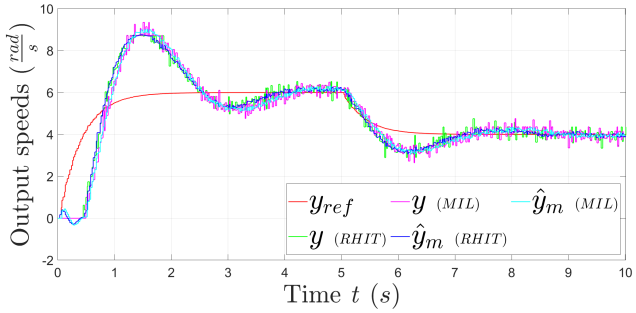


Figure 6.36: Right DC motor outputs y, y_{ref} and $\hat{y}_m$ - Comparison between MIL and RHIT

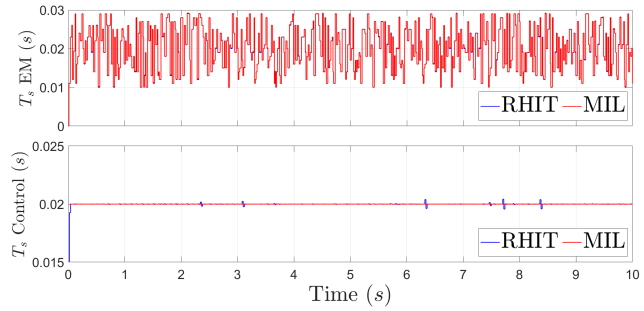


Figure 6.37: Right DC motor timestamp for EM and Control part of EMC - Comparison between MIL and RHIT

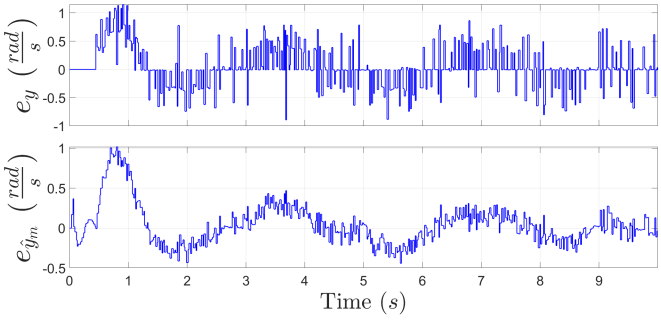


Figure 6.38: Right DC motor MIL and RHIT e_y (above) and $e_{\hat{y}_m}$ (below) error difference

6.4.4 No disturbance rejection, target output speed $\tilde{\omega}_R = [8, 3, 5.5, 7]$ $\frac{\text{rad}}{\text{s}}$

MIL simulation test results:

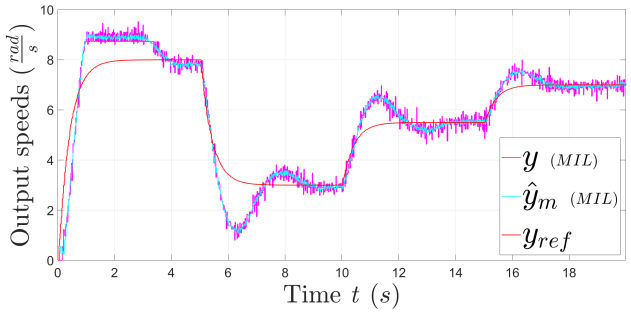


Figure 6.39: Right DC motor outputs y, y_{ref} and $\hat{y}_m$

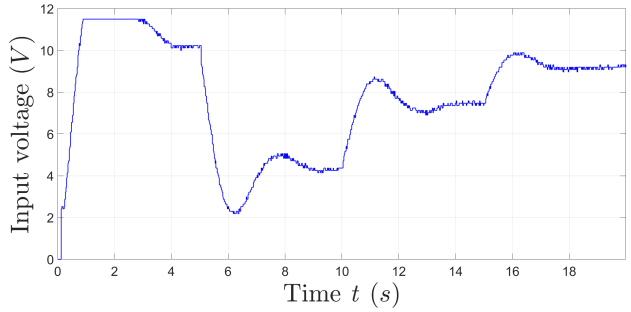


Figure 6.41: Right DC motor input voltage u

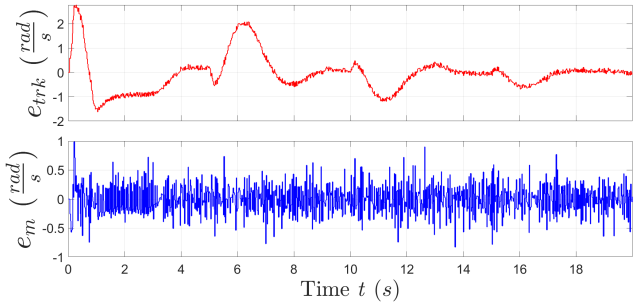


Figure 6.40: Right DC motor tracking error e_{trk} and model error e_m

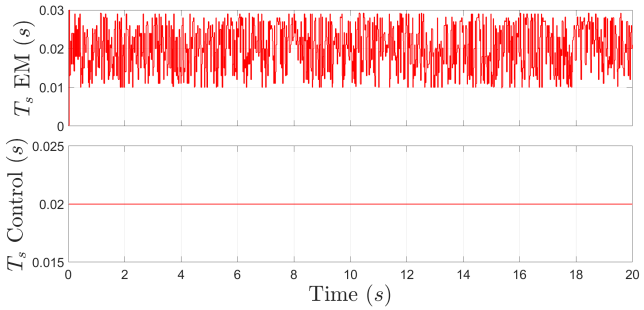


Figure 6.42: Right DC motor timestamp for EM and Control part of EMC

Comparison between MIL and RHIT:

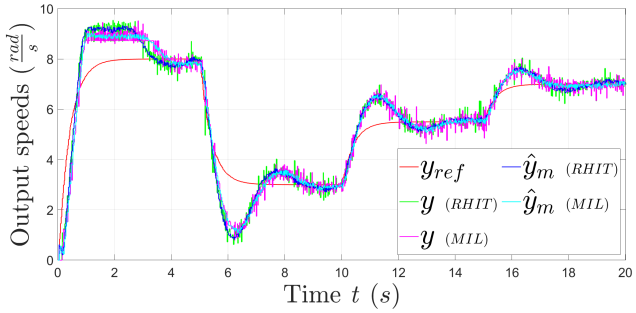


Figure 6.43: Right DC motor outputs y, y_{ref} and $\hat{y}_m$ - Comparison between MIL and RHIT

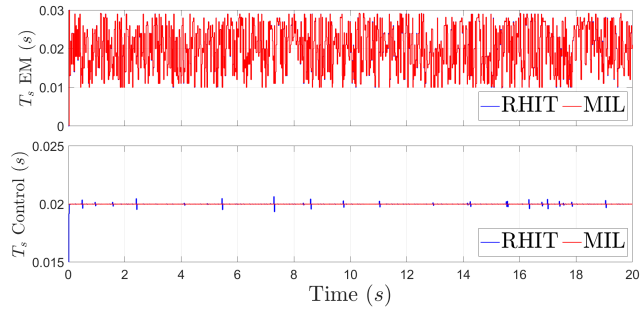


Figure 6.44: Right DC motor timestamp for EM and Control part of EMC - Comparison between MIL and RHIT

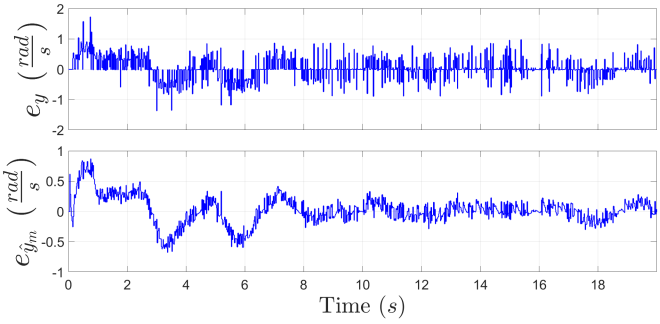


Figure 6.45: Right DC motor MIL and RHIT e_y (above) and $e_{\hat{y}_m}$ (below) error difference

Chapter 7

Separated Left and Right DC motors EMC - RHIT results

In this chapter SW/HW implementation (RHIT) of EMC for robot GoPiGo3 DC motors are described, showing the main results: the ability of estimated states to track the reference states (tracking error) and the difference of simplified EM model estimated outputs wrt the real robot plant outputs (model error). For theory about motors EMC check Chapter 5, the versions tested are v1.0 and v1.1 for left motor, and v1.2 for right motor.

7.1 Left motor version 1.0 - RHIT results

In the next sections the Hardware-in-the-loop (RHIT) results of EMC Left DC motor are presented, considering both 1st and 2nd order disturbances for $\hat{x}_d$ and $\overline{w}$. Instead for Right DC motor EMC version 1.2 is considered.

7.1.1 Main test settings

Since this is the first EMC built, many settings are experienced to obtain the best result in terms of tracking and model errors. First some tests using a fixed sampling time of $T_s = 0.02\,\mathrm{s}$ or $T_s = 0.04\,\mathrm{s}$ are performed, in order to see if EMC is working well in these conditions. These values are chosen properly, because it is known that motors PWM work only at a fixed sampling time of $T_{PWM} = 0.02\,\mathrm{s}$: therefore the fixed control sampling time must be the same of a multiple. After that EMC can be directly proven using variable sampling time (the ranges are explained afterwards).
EMC is then tested in load and no load conditions (robot placed or not on the ground), using or not disturbance rejection in control input voltage u, using a 1st or a 2nd order disturbance $\overline{w}$ model.
For each result the plots show:

- Outputs of control model, motor measured speed y, estimated motor speed $\hat{y}_m$, reference speed $\overline{y} = y_{ref}$ to be tracked by $\hat{y}_m$.
- Control input voltage u.
- Tracking error e_{trk} and model error e_m.
- The 2 components of u control input, disturbance rejection part u_d and tracking part u_{trk}.
  The reference part $\bar{u}$ is not of practical interest, because it depends on reference dynamics which is not affected by disturbances.
- Timestamp T.

Modifications applied during work on control model and its implementation

- Some modifications are applied on the control CT eigenvalues μ_K, since they're too fast for RHIT practical implementations, passing from ≈ -80 to ≈ -15.

- A filter was added for the command input u, using a *moving average (MA) filter*, which means that final command input u_{MA} will be the average sum of previous $m - n$ inputs (m is the number of total inputs, n is called the delay window length), [26]:

$$u_{MA}(k) = \frac{1}{n}\sum_{k=1}^{n-1} u(m-k) \tag{7.1}$$

  A filter of this type always introduces a time delay which cannot be overcome directly in RHIT implementations, therefore a small window length must be selected. In this case $n = 3$ is sufficient, so they are used 3 previous control inputs to build the moving average final input. Further informations can be found in every signal processing book, like [16].
  In DC motor EMC case, the presence of moving average filter causes a small time delay of overall system, hence increases stability in estimated motor speeds (i.e. CT eigenvalues can be faster without making system estimated outputs oscillating).

- In C++ code implementation there are 2 main parts to be executed at every time step, one is the EMC model and the other is the plant: during code execution the parts are run alternatively (plant → EMC → plant etc.). At the beginning, at every step, first the EMC was executed, then the plant: in this way the plant output speed enters the EMC model only in the next step, leading to bad results, especially using variable sample time: in particular y output resolution becomes very high.
  Solution is to run the plant first and use immediately the result output in the EMC model: in this case y output resolution reduces to reliable physical values. Comparing Figure 7.1, where the EMC is run first at every time step and Figure 7.2 where instead the plant is run first, the difference in output y resolution is visibly clear.

- The control law part is separated from EM and ran at fixed sample time. Instead the EM part (with noise estimator and reference dynamics blocks) run with variable sample time, further informations in Section 4.3.1. For RHIT experiments a timer running at $T_{ctrl} = 0.02\,\mathrm{s}$ is used to step the control part model, and another timer to step the EM part, at $T_{EM} = [0.01, 0.03]$ or $[0.02, 0.04]\mathrm{s}$. This separation is used trying to have better control results, since motor PWM always run at $T_{PWM} = 0.02\,\mathrm{s}$.

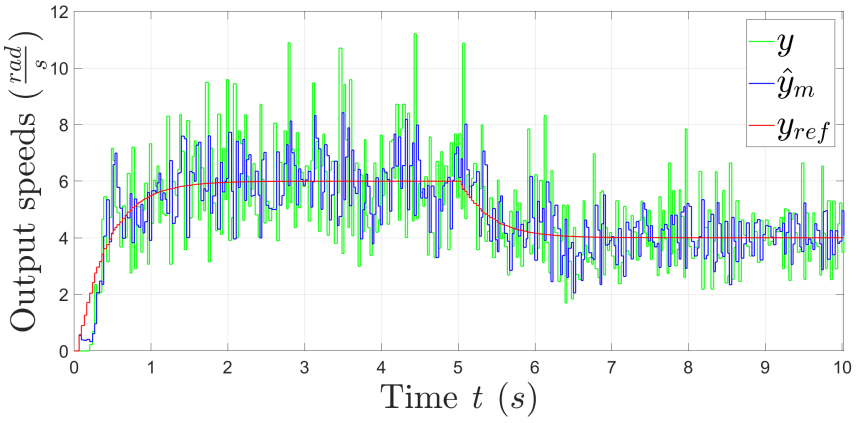


Figure 7.1: Measured output speeds $\hat{y}_m$ and y not filtered - EMC → plant implementation

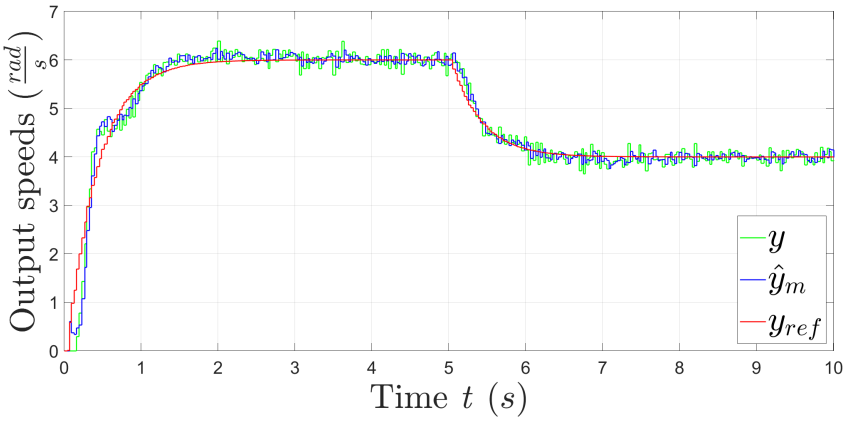


Figure 7.2: Measured output speeds $\hat{y}_m$ and y not filtered - Plant → EMC implementation

7.1.2 Summary on the results

In general the model error is low in every test (in many cases with a bound $< 0.5\,\text{rad/s}$), meaning that the noise estimator is working very well. This bound cannot be improved because it is almost near the encoder quantization resolution error for angular speed: indeed for $T_s = 0.01\,\text{s}$ the error is $\epsilon_{max} = 0.0087/0.01 = 0.8727\,\text{rad/s}$, and for $T_s = 0.04\,\text{s}$ the error is $\epsilon_{min} = 0.0087/0.04 = 0.2182\,\text{rad/s}$. Hence the bound is $\epsilon = [0.2182, 0.8727]$ rad/s. $0.5° = 0.8727\,\text{rad}$ is the encoder resolution position.

Instead for tracking error the results are really dependent on tests performed.

In the case of fixed sampling time, even with a simple 1st order disturbance model only, the tracking and disturbance rejection are very good. Comparing the 2 disturbance models (1st and 2nd), it can be seen that better results are obtained with the 1st order, because making the control loop faster the output tracks better the reference. This happens with continuous time control eigenvalues like $\mu_K = [-11.1572, 11.1572]$ using $T_s = 0.04\,\text{s}$, with EMC unified using only 1 HW timer (compare tests in Section 7.1.5 and Section 7.1.15).

This can be explained by noticing that $\hat{y}_m$ is not entirely able to follow y using 1st order disturbance, indicating that some dynamics are not estimated wrt 2nd order disturbance, look for example the output speeds in Section 7.1.5.

However, the advantage of 2nd order disturbance can be seen using variable sample time: the ability of tracking the reference is slightly better, check tests in Section 7.1.7 and Section 7.1.17, using unified EMC.

When the EMC is split in 2 parts, EM and control, the tracking error is a little bit lower than using a unified EMC, compare errors in Section 7.1.17 and Section 7.1.18. The difference is minimal, however conceptually is better to have a control part running with a fixed sampling time of $T_c = 0.02\,\text{s}$, because by construction PWM generator is giving motor voltage at that rate.

Both using 1st and 2nd order disturbance and concerning the cases where EMC is split in two parts (EM and Control parts), reducing the random time used for timer to step the Embedded model part from $T_{EM} = [0.02, 0.04]$ s to $T_{EM} = [0.01, 0.03]$ s and letting the control timer fixed to $T_c = 0.02\,\text{s}$, the estimated output speed $\hat{y}_m$ (and consequently measurement y) tracks in a better way the reference y_{ref}. The highest evidence is between tests using 2nd order disturbances in Section 7.1.18 ($T_{EM} = [0.02, 0.04]$ s) and Section 7.1.19 ($T_{EM} = [0.01, 0.03]$s).

Furthermore, especially in load conditions, advantage of disturbance term u_d in the command input is well evident comparing the results with and without this term: without u_d the tracking is very bad. Compare for example tests in Section 7.1.23 and Section 7.1.24.

Using both fixed or variable sampling time, best error conditions are obtained with very slow control CT eigenvalues, with $\mu_K = [-2.5647 \times 2]$.

7.1.3 First order disturbance, No load conditions - Result 1

Test settings: Disturbance rejection, Fixed sampling time $T = 0.02\,\text{s}$, timer Control and EM unified, target speed $[6, 4]$ rad/s.

Eigenvalue type	CT eigenvalues	Related DT at $T = 0.02\,\text{s}$
Control μ_K	$[-11.1572, -11.1572]$	$[0.8, 0.8]$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842]$	$[0.75, 0.75]$

Table 7.1: CT eigenvalues DC motor EMC Left motor v1.0 - 1st order disturbance, No load conditions, Result 1

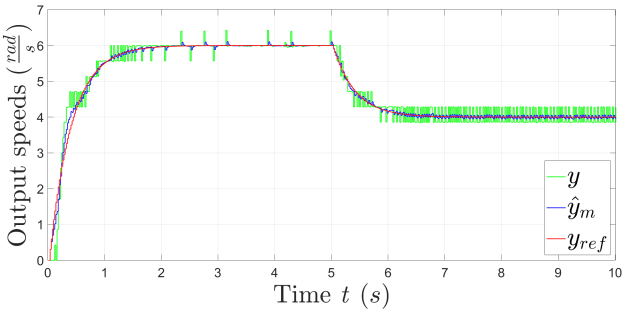


Figure 7.3: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

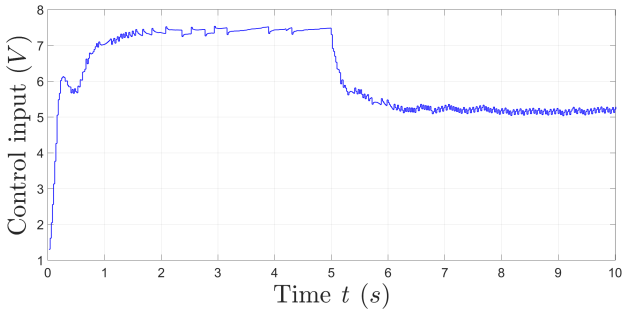


Figure 7.4: Control input voltage u

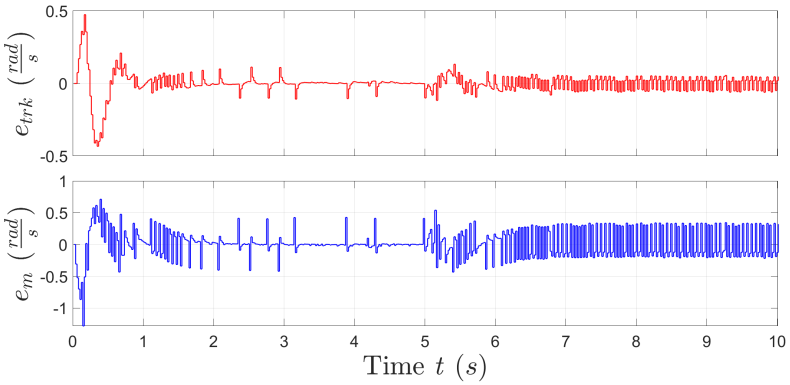


Figure 7.5: Tracking error e_{trk} and model error e_m

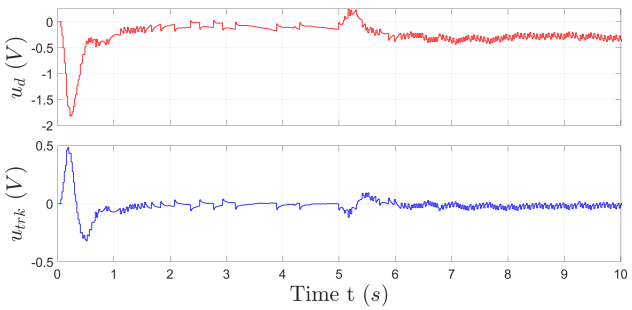


Figure 7.6: Control input parts u_d and u_{trk}

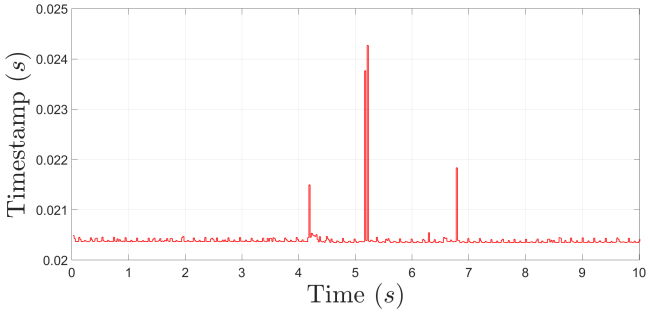


Figure 7.7: Measured Timestamp T

7.1.4 First order disturbance, No load conditions - Result 2

Test settings: Disturbance rejection, Fixed sampling time $T = 0.02\,\mathrm{s}$, timer Control and EM unified, target speed $[6, 4]$ rad/s.

Eigenvalue type	CT eigenvalues	Related DT at $T = 0.02\,\mathrm{s}$
Control μ_K	$[-5.2680, -5.2680]$	$[0.9, 0.9]$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842]$	$[0.75, 0.75]$

Table 7.2: CT eigenvalues DC motor EMC Left motor v1.0 - 1st order disturbance, No load conditions, Result 2

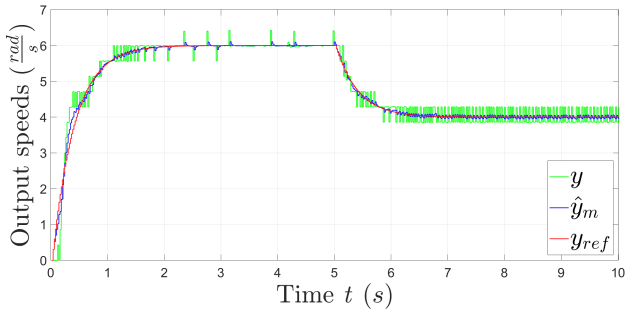


Figure 7.8: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

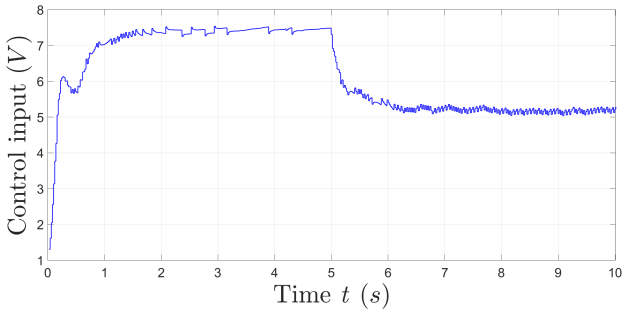


Figure 7.9: Control input voltage u

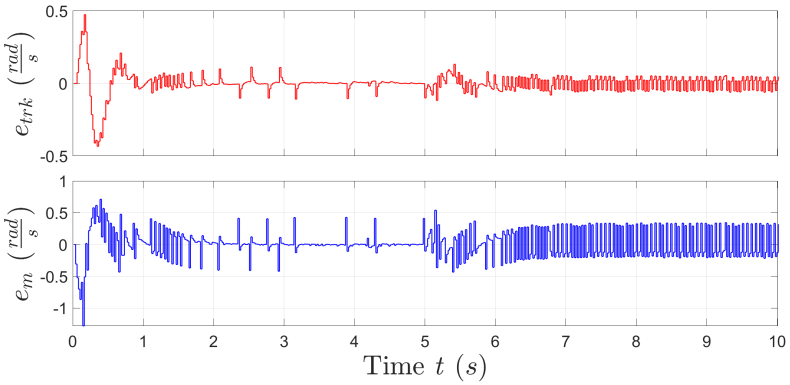


Figure 7.10: Tracking error e_{trk} and model error e_m

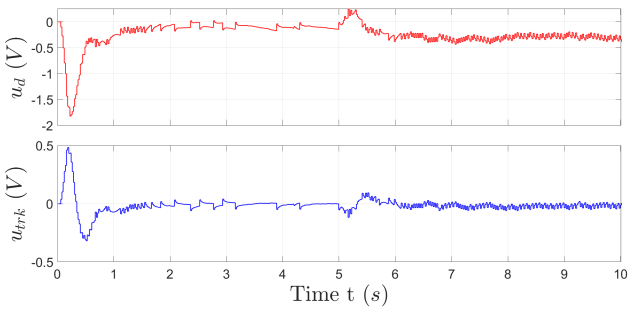


Figure 7.11: Control input parts u_d and u_{trk}

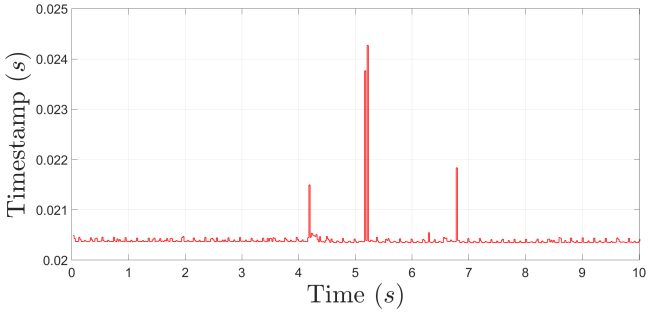


Figure 7.12: Measured Timestamp T

7.1.5 First order disturbance, No load conditions - Result 3

Test settings: Disturbance rejection, Fixed sampling time $T = 0.04\,\text{s}$, timer Control and EM unified, target speed $[6, 4]$ rad/s.

Eigenvalue type	CT eigenvalues	Related DT at $T = 0.02\,\text{s}$
Control μ_K	$[-11.1572, -11.1572]$	$[0.8, 0.8]$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842]$	$[0.75, 0.75]$

Table 7.3: CT eigenvalues DC motor EMC Left motor v1.0 - 1st order disturbance, No load conditions, Result 3

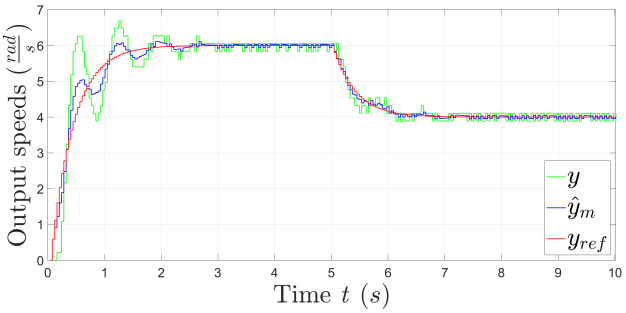


Figure 7.13: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

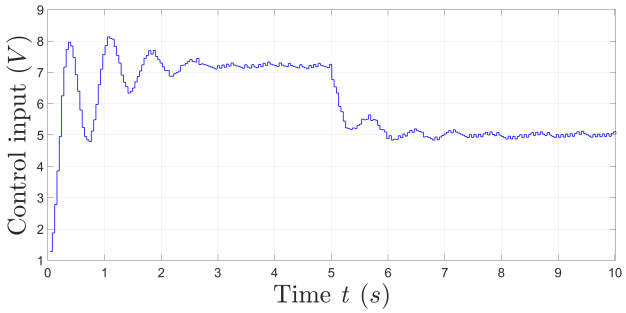


Figure 7.14: Control input voltage u

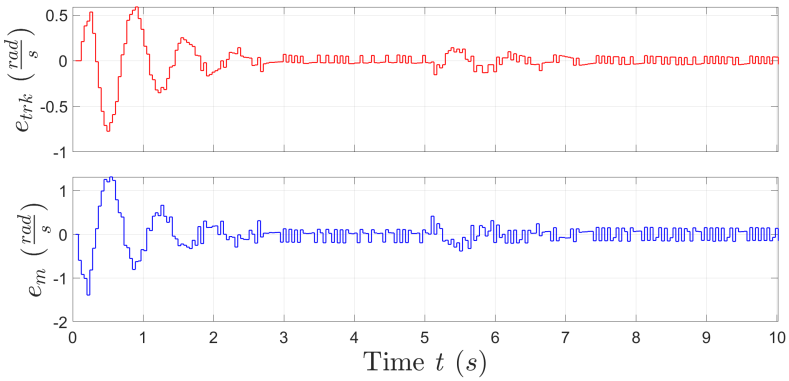


Figure 7.15: Tracking error e_{trk} and model error e_m

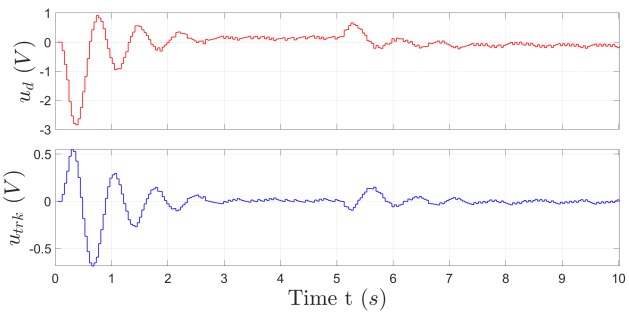


Figure 7.16: Control input parts u_d and u_{trk}

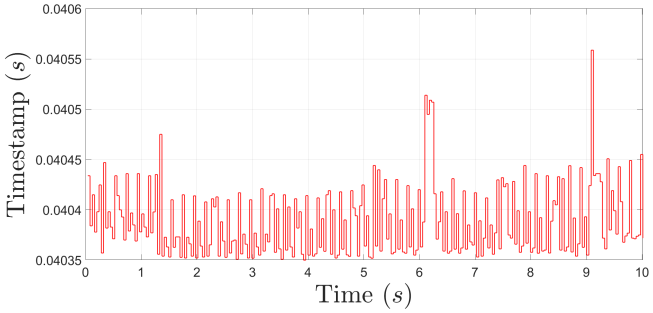


Figure 7.17: Measured Timestamp T

7.1.6 First order disturbance, No load conditions - Result 4

Test settings: Disturbance rejection, Fixed sampling time $T = 0.04\,\text{s}$, timer Control and EM unified, target speed $[6, 4]$ rad/s.

Eigenvalue type	CT eigenvalues	Related DT at $T = 0.02\,\text{s}$
Control μ_K	$[-5.2680, -5.2680]$	$[0.9, 0.9]$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842]$	$[0.75, 0.75]$

Table 7.4: CT eigenvalues DC motor EMC Left motor v1.0 - 1st order disturbance, No load conditions, Result 4

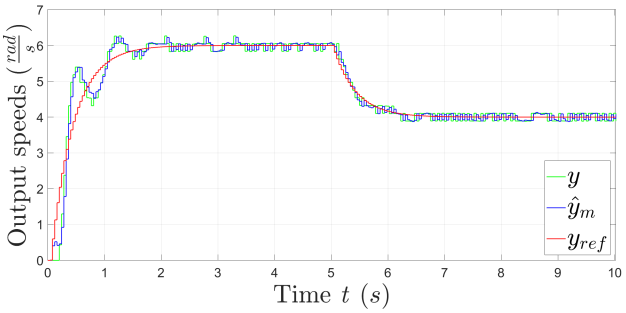


Figure 7.18: Measured output speeds $\hat{y}_m$ and y not filtered

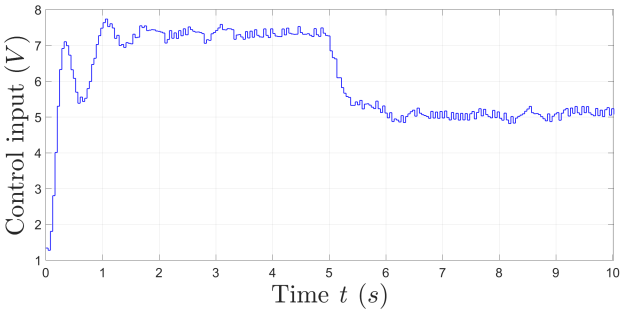


Figure 7.19: Control input voltage u

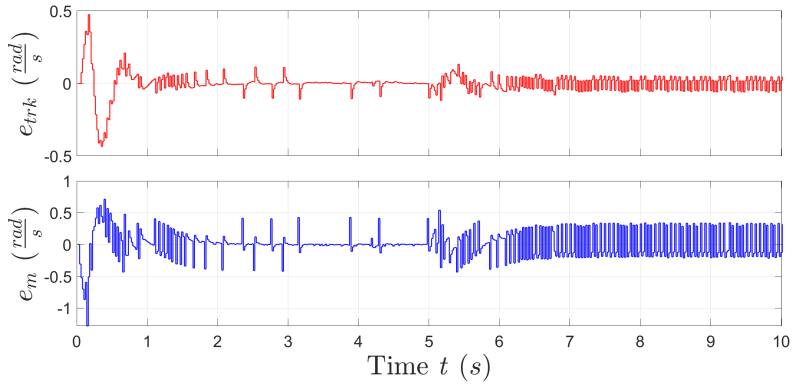


Figure 7.20: Measured tracking error e_{trk} and model error e_m

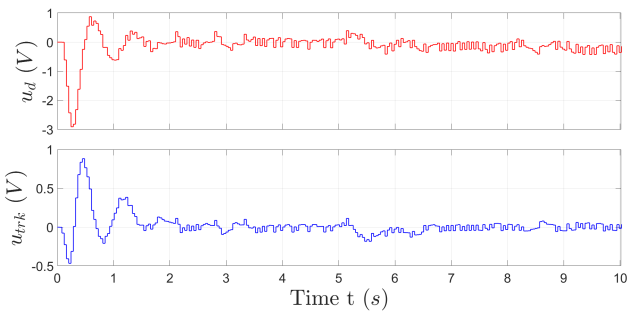


Figure 7.21: Control input parts u_d and u_{trk}

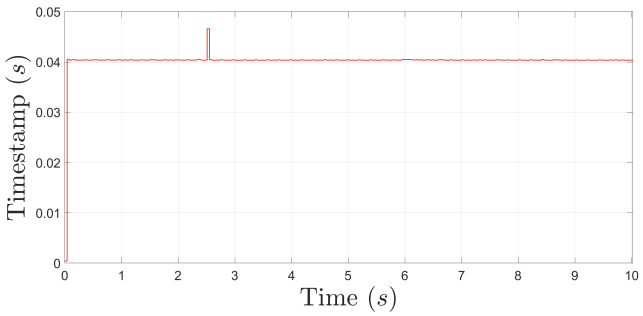


Figure 7.22: Measured Timestamp T

7.1.7 First order disturbance, No load conditions - Result 5

Test settings: Disturbance rejection, Variable sampling time $T = [0.02, 0.04]$ s, timer Control and EM unified, target speed $[6, 4]$ rad/s.

Eigenvalue type	CT eigenvalues	Related DT at $T = 0.02\,\mathrm{s}$
Control μ_K	$[-2.5647, -2.5647]$	$[0.95, 0.95]$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842]$	$[0.75, 0.75]$

Table 7.5: CT eigenvalues DC motor EMC Left motor v1.0 - 1st order disturbance, No load conditions, Result 5

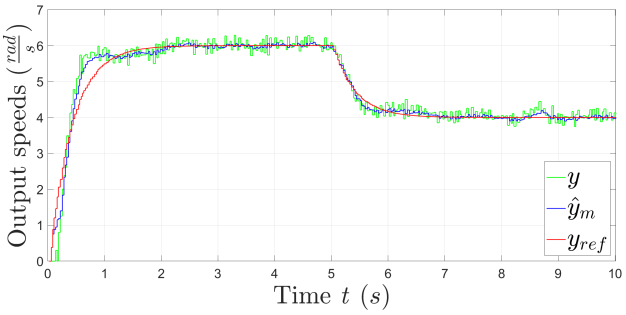


Figure 7.23: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

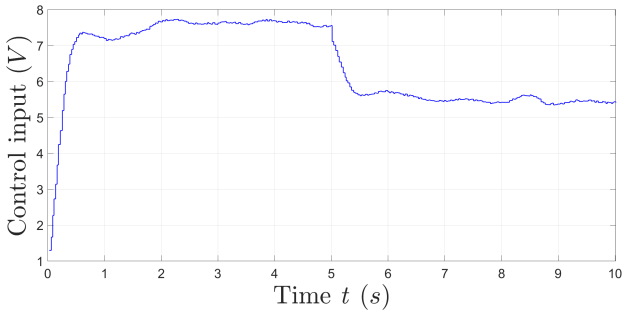


Figure 7.24: Control input voltage u

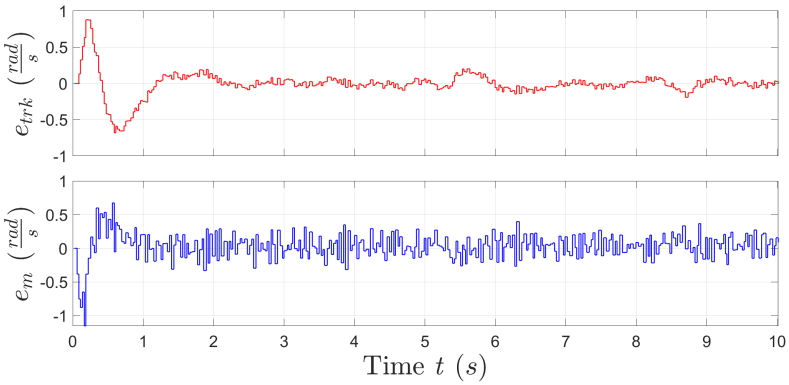


Figure 7.25: Measured tracking error e_{trk} and model error e_m

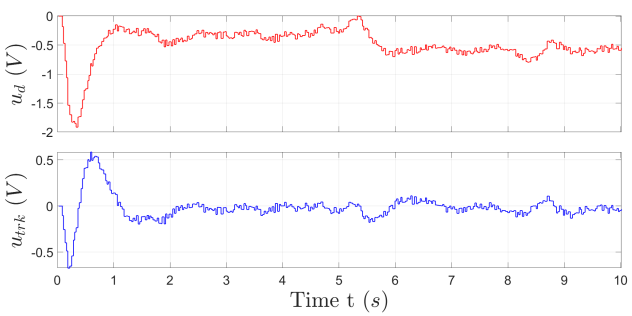


Figure 7.26: Control input parts u_d and u_{trk}

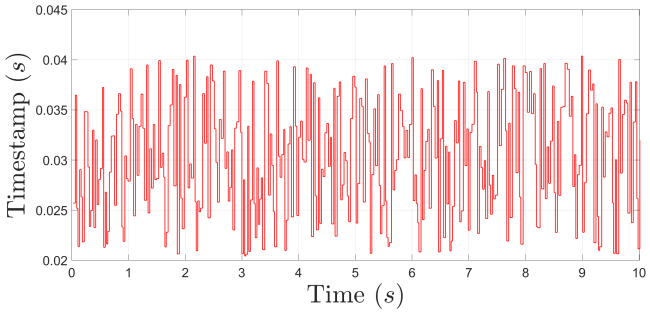


Figure 7.27: Measured Timestamp T

7.1.8 First order disturbance, No load conditions - Result 6

Test settings: Disturbance rejection, Variable sampling time $T = [0.02, 0.04]$ s, timer Control and EM separated, target speed $[6, 4]$ rad/s.

Eigenvalue type	CT eigenvalues	Related DT at $T = 0.02$ s
Control μ_K	$[-2.5647, -2.5647]$	$[0.95, 0.95]$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842]$	$[0.75, 0.75]$

Table 7.6: CT eigenvalues DC motor EMC Left motor v1.0 - 1st order disturbance, No load conditions, Result 6

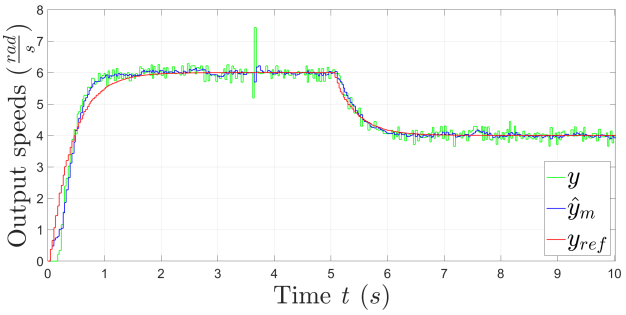


Figure 7.28: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

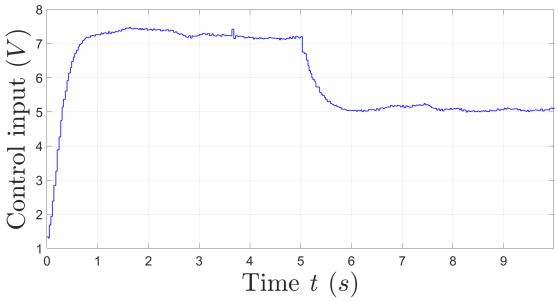


Figure 7.29: Control input voltage u

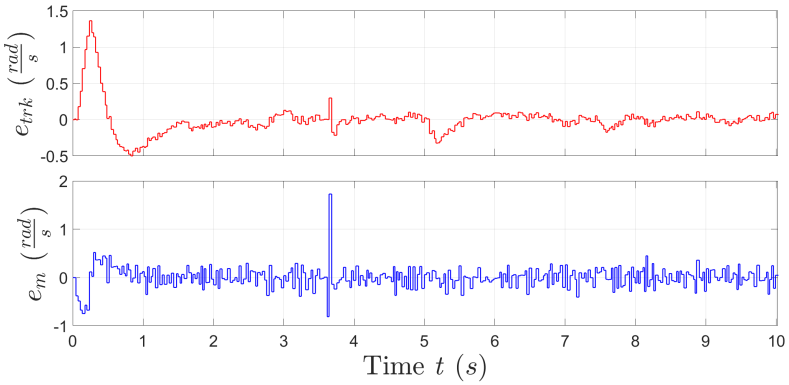


Figure 7.30: Tracking error e_{trk} and model error e_m

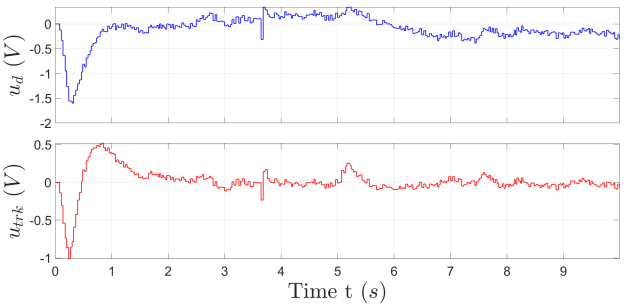


Figure 7.31: Control input parts u_d and u_{trk}

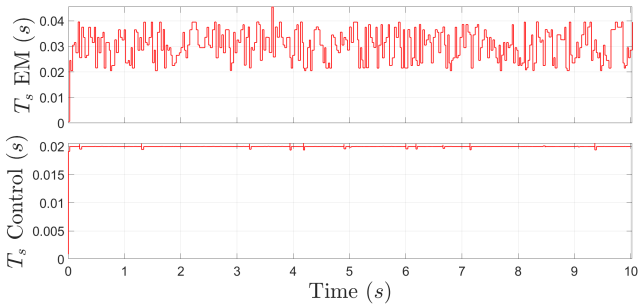


Figure 7.32: Measured Timestamp T

7.1.9 First order disturbance, No load conditions - Result 7

Test settings: Disturbance rejection, Variable sampling time $T = [0.01, 0.03]$ s, timer Control and EM separated, target speed $[6, 4]$ rad/s.

Eigenvalue type	CT eigenvalues	Related DT at $T = 0.02$ s
Control μ_K	$[-2.5647, -2.5647]$	$[0.95, 0.95]$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842]$	$[0.75, 0.75]$

Table 7.7: CT eigenvalues DC motor EMC Left motor v1.0 - 1st order disturbance, No load conditions, Result 7

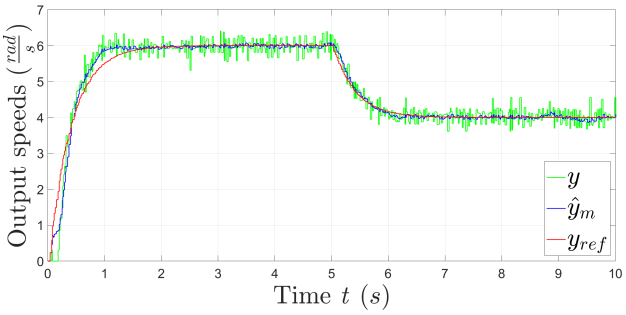


Figure 7.33: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

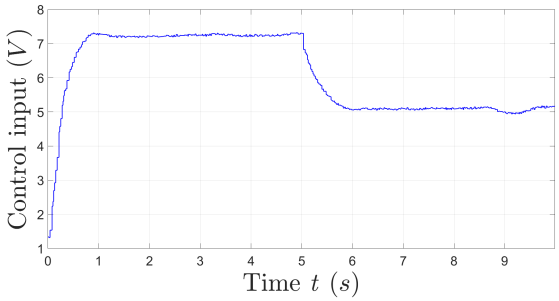


Figure 7.34: Control input voltage u

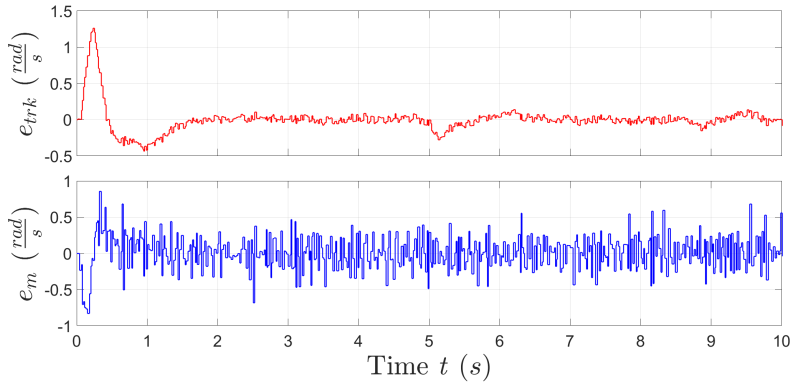


Figure 7.35: Tracking error e_{trk} and model error e_m

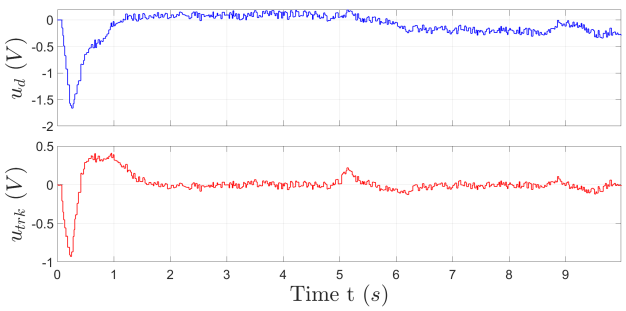


Figure 7.36: Control input parts u_d and u_{trk}

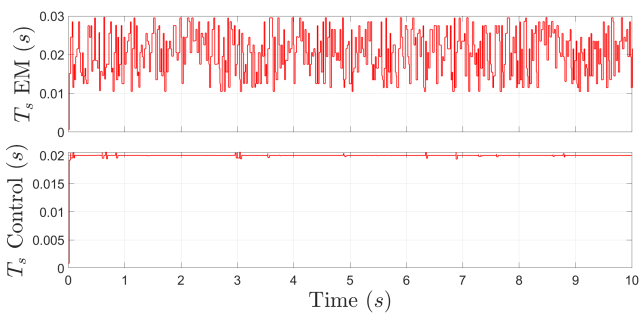


Figure 7.37: Measured Timestamp T

7.1.10 First order disturbance, No load conditions - Result 8

Test settings: No disturbance rejection, Variable sampling time $T = [0.02, 0.04]$ s, timer Control and EM separated, target speed $[6, 4]$ rad/s.

Eigenvalue type	CT eigenvalues	Related DT at $T = 0.02$ s
Control μ_K	$[-2.5647, -2.5647]$	$[0.95, 0.95]$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842]$	$[0.75, 0.75]$

Table 7.8: CT eigenvalues DC motor EMC Left motor v1.0 - 1st order disturbance, No load conditions, Result 8

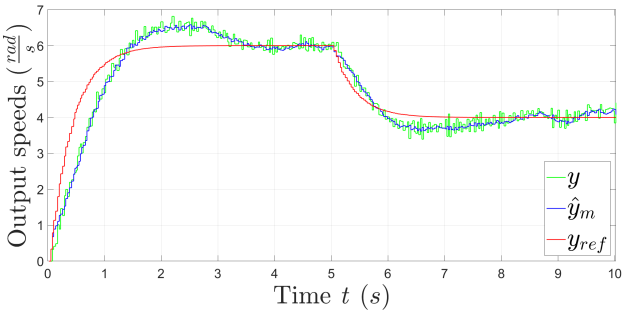


Figure 7.38: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

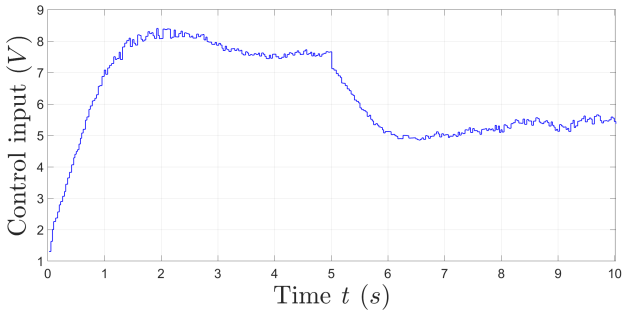


Figure 7.39: Control input voltage u

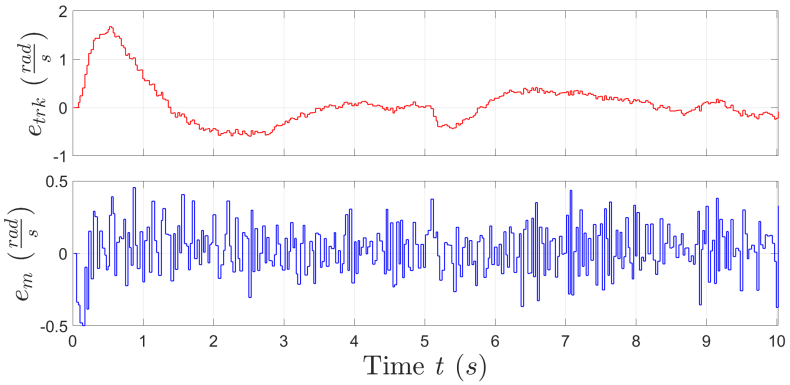


Figure 7.40: Tracking error e_{trk} and model error e_m

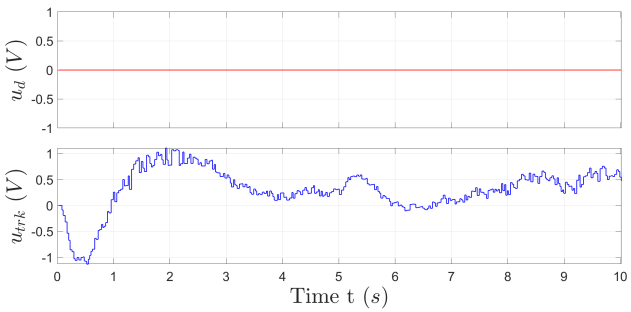


Figure 7.41: Control input parts u_d and u_{trk}

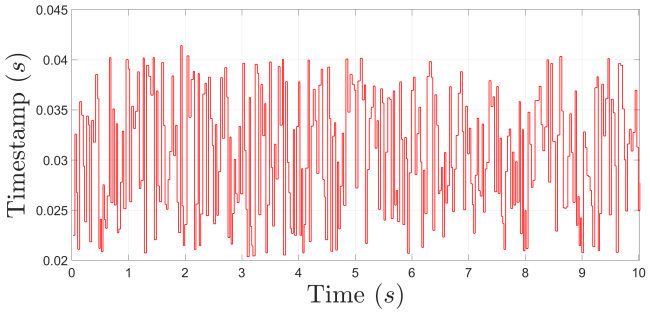


Figure 7.42: Measured Timestamp T

7.1.11 First order disturbance, Load Conditions - Result 1

Test settings: Disturbance rejection, Variable sampling time $T = [0.01, 0.03]$ s, timer Control and EM separated, target speed $[6, 4]$ rad/s.

Eigenvalue type	CT eigenvalues	Related DT at $T = 0.02$ s
Control μ_K	$[-2.5647, -2.5647]$	$[0.95, 0.95]$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842]$	$[0.75, 0.75]$

Table 7.9: CT eigenvalues DC motor EMC Left motor v1.0 - 1st order disturbance, Load conditions, Result 1

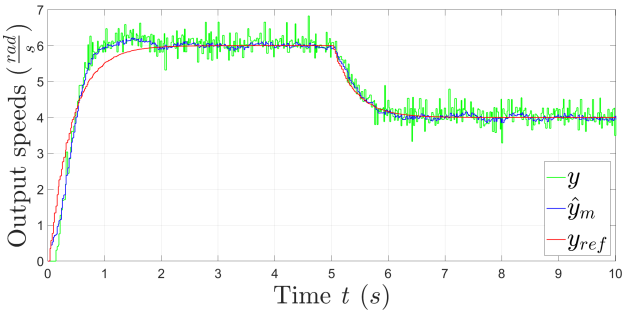


Figure 7.43: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

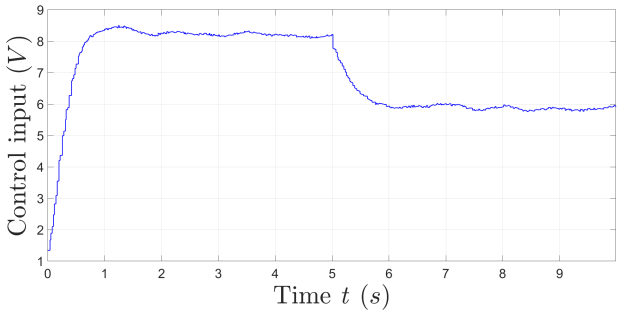


Figure 7.44: Control input voltage u

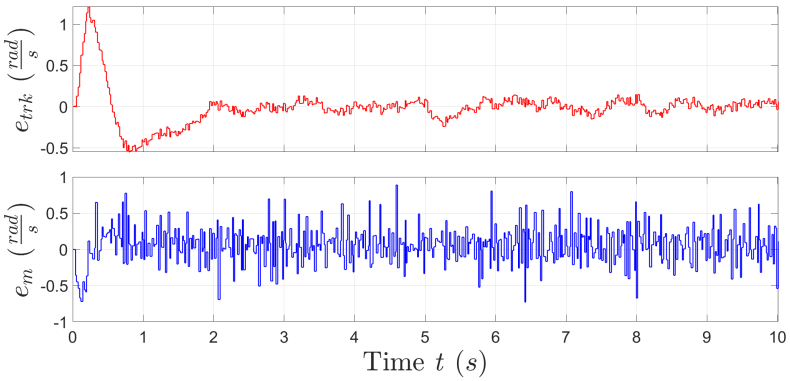


Figure 7.45: Tracking error e_{trk} and model error e_m

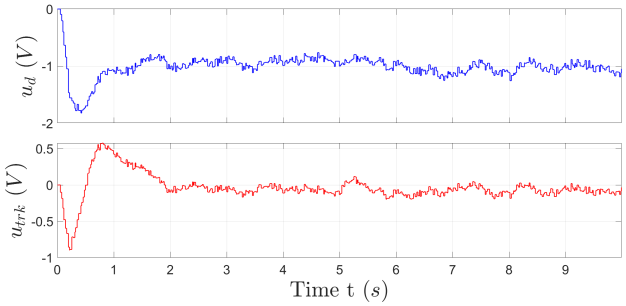


Figure 7.46: Control input parts u_d and u_{trk}

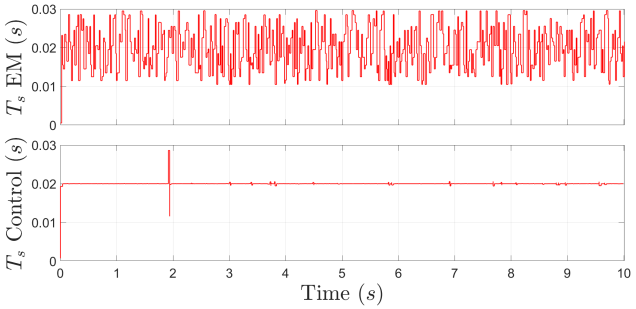


Figure 7.47: Measured Timestamp T

7.1.12 First order disturbance, Load Conditions - Result 2

Test settings: No disturbance rejection, Variable sampling time $T = [0.01, 0.03]$ s, timer Control and EM separated, target speed $[6, 4]$ rad/s.

Eigenvalue type	CT eigenvalues	Related DT at $T = 0.02$ s
Control μ_K	$[-2.5647, -2.5647]$	$[0.95, 0.95]$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842]$	$[0.75, 0.75]$

Table 7.10: CT eigenvalues DC motor EMC Left motor v1.0 - 1st order disturbance, Load conditions, Result 2

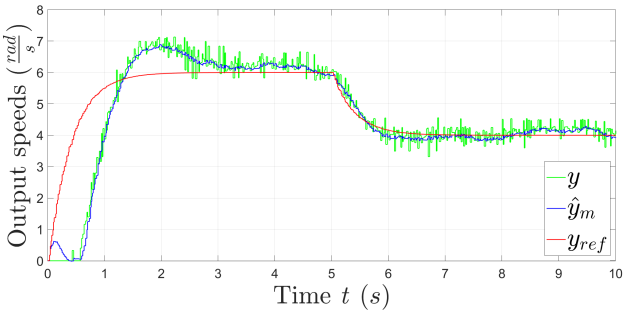


Figure 7.48: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

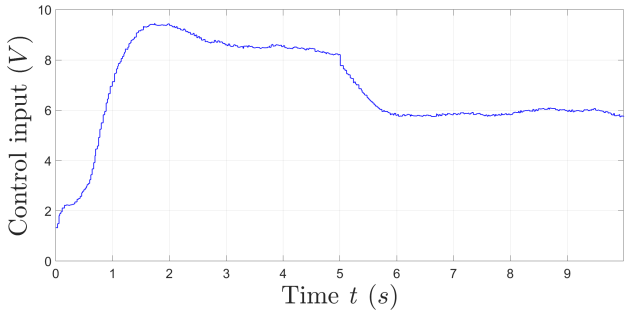


Figure 7.49: Control input voltage u

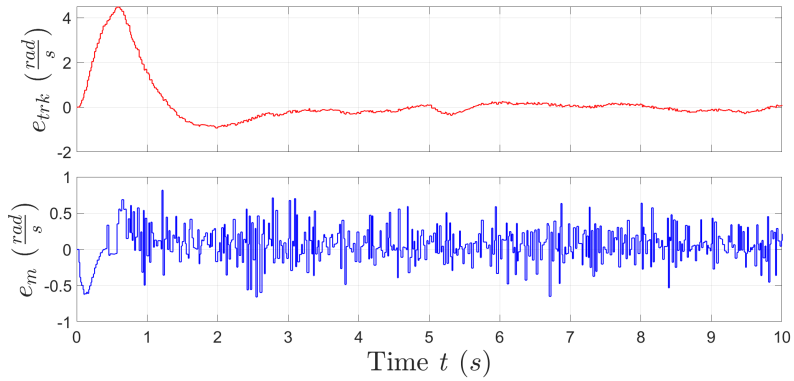


Figure 7.50: Tracking error e_{trk} and model error e_m

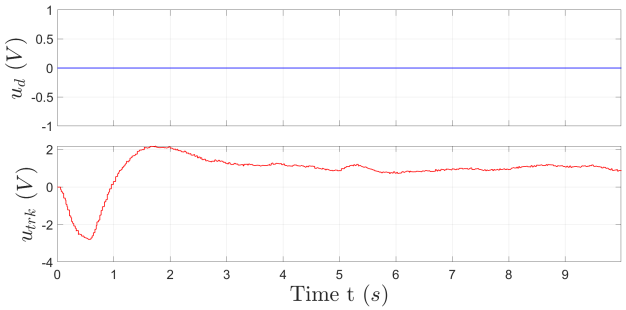


Figure 7.51: Control input parts u_d and u_{trk}

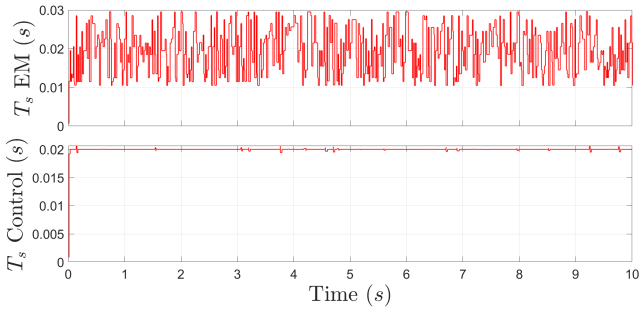


Figure 7.52: Measured Timestamp T

7.1.13 Second order disturbance, No load conditions - Result 1

Test settings: Disturbance rejection, Fixed sampling time T = 0.02 s, timer Control and EM unified, target speed $[6, 4]$ rad/s.

Eigenvalue type	CT eigenvalues	Related DT at $T = 0.02\,\mathrm{s}$
Control μ_K	$[-11.1572, -11.1572]$	$[0.8, 0.8]$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842]$	$[0.75, 0.75]$

Table 7.11: CT eigenvalues DC motor EMC Left motor v1.0 - 2nd order disturbance, No load conditions, Result 1

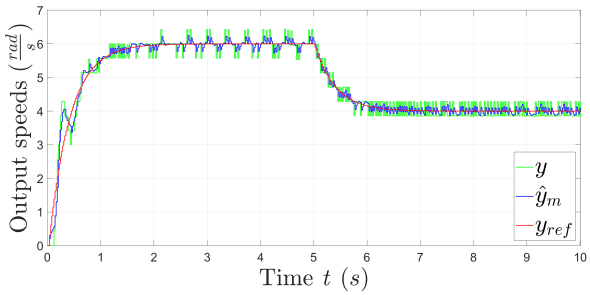


Figure 7.53: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

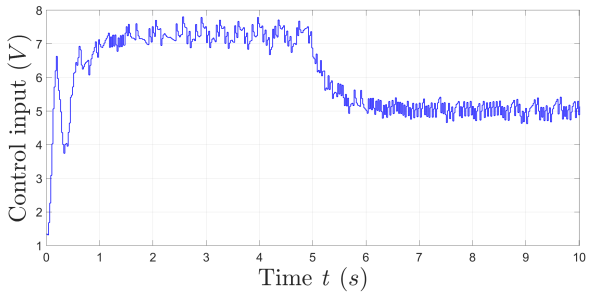


Figure 7.54: Control input voltage u

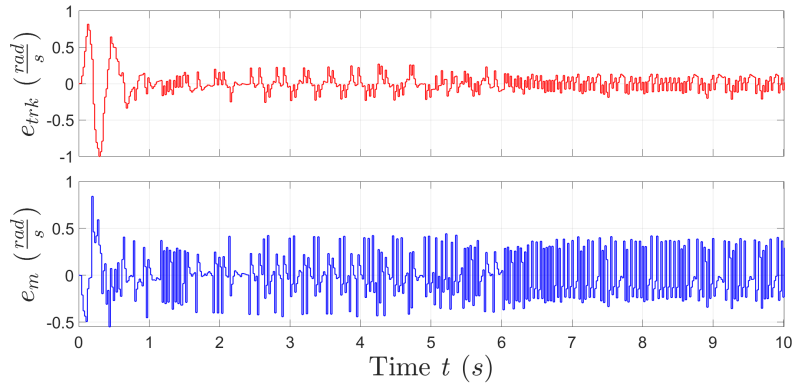


Figure 7.55: Measured tracking error e_{trk} and model error e_m

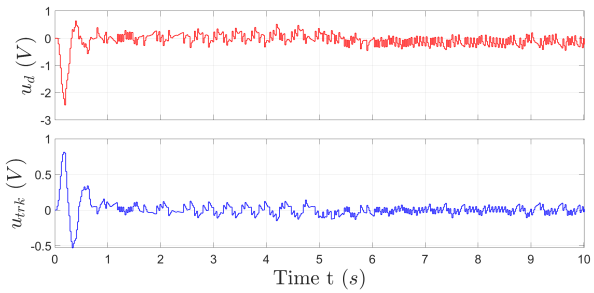


Figure 7.56: Control input parts u_d and u_{trk}

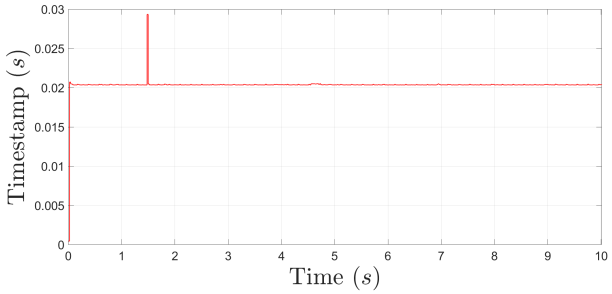


Figure 7.57: Measured Timestamp T

7.1.14 Second order disturbance, No load conditions - Result 2

Test settings: Disturbance rejection, Fixed sampling time $T = 0.02$ s, timer Control and EM unified, target speed $[6, 4]$ rad/s.

Eigenvalue type	CT eigenvalues	Related DT at $T = 0.02\,\mathrm{s}$
Control μ_K	$[-5.2680, -5.2680]$	$[0.9, 0.9]$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842]$	$[0.75, 0.75]$

Table 7.12: CT eigenvalues DC motor EMC Left motor v1.0 - 2nd order disturbance, No load conditions, Result 2

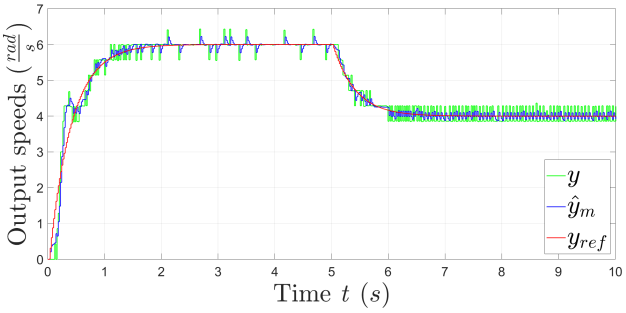


Figure 7.58: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

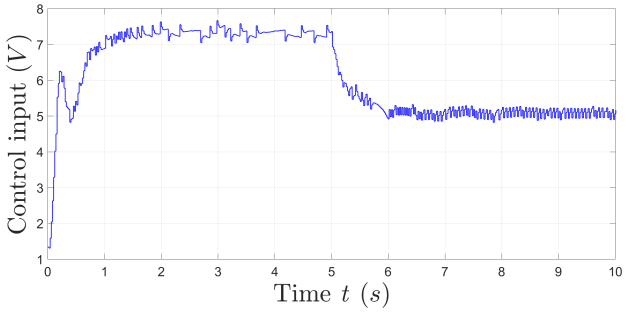


Figure 7.59: Control input voltage u

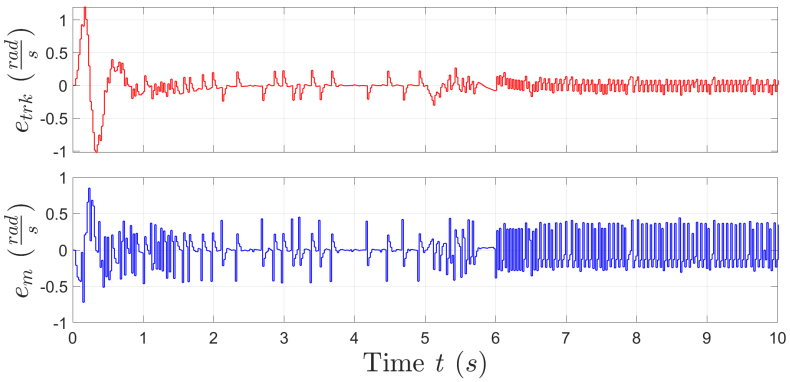


Figure 7.60: Tracking error e_{trk} and model error e_m

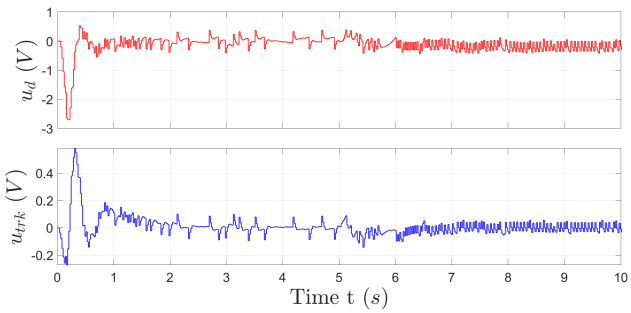


Figure 7.61: Control input parts u_d and u_{trk}

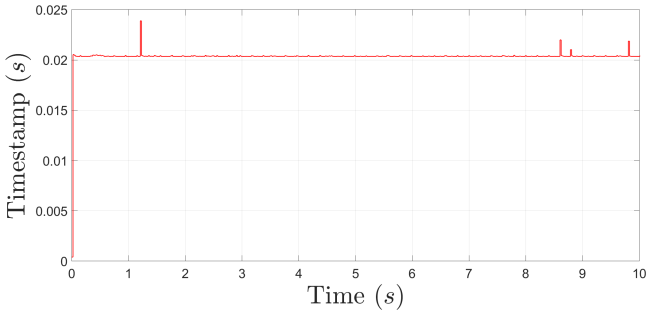


Figure 7.62: Measured Timestamp T

7.1.15 Second order disturbance, No load conditions - Result 3

Test settings: Disturbance rejection, Fixed sampling time T = 0.04 s, timer Control and EM unified, target speed $[6, 4]$ rad/s.

Eigenvalue type	CT eigenvalues	Related DT at $T = 0.02\,\mathrm{s}$
Control μ_K	$[-11.1572, -11.1572]$	$[0.8, 0.8]$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842]$	$[0.75, 0.75]$

Table 7.13: CT eigenvalues DC motor EMC Left motor v1.0 - 2nd order disturbance, No load conditions, Result 3

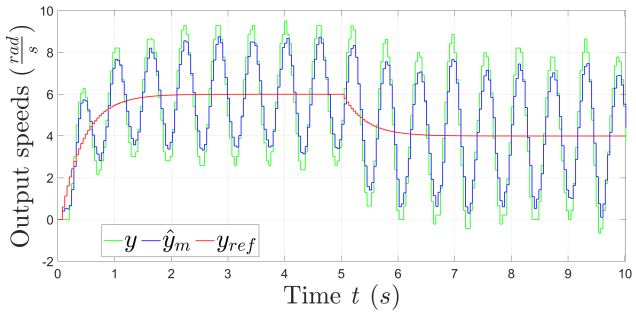


Figure 7.63: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

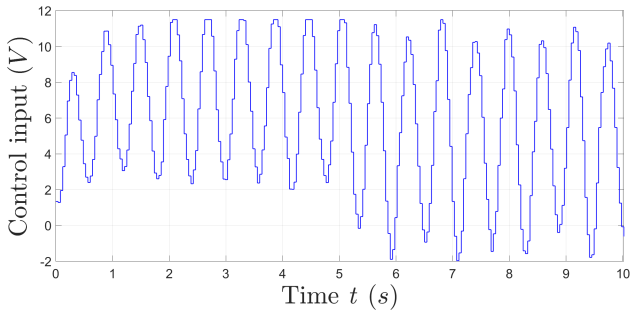


Figure 7.64: Control input voltage u

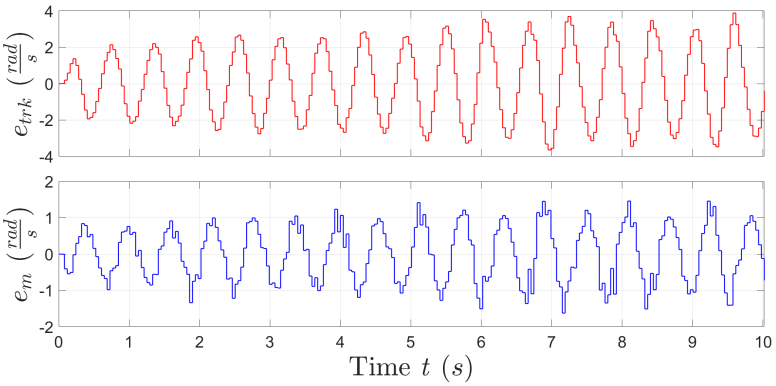


Figure 7.65: Tracking error e_{trk} and model error e_m

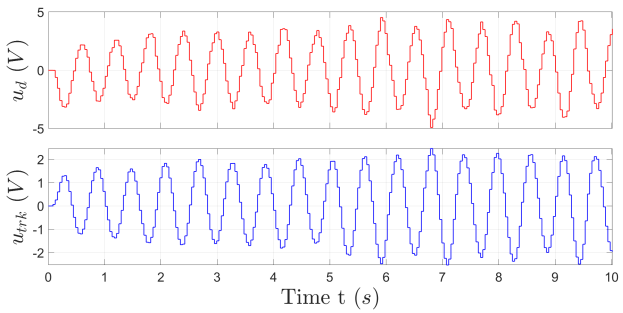


Figure 7.66: Control input parts u_d and u_{trk}

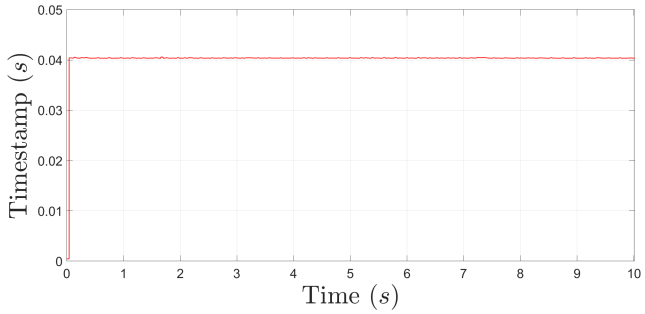


Figure 7.67: Measured Timestamp T

7.1.16 Second order disturbance, No load conditions - Result 4

Test settings: Disturbance rejection, Fixed sampling time $T = 0.04$ s, timer Control and EM unified, target speed $[6, 4]$ rad/s.

Eigenvalue type	CT eigenvalues	Related DT at $T = 0.02\,\text{s}$
Control μ_K	$[-5.2680, -5.2680]$	$[0.9, 0.9]$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842]$	$[0.75, 0.75]$

Table 7.14: CT eigenvalues DC motor EMC Left motor v1.0 - 2nd order disturbance, No load conditions, Result 4

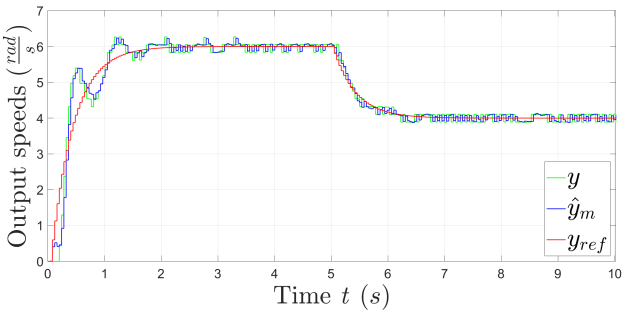


Figure 7.68: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

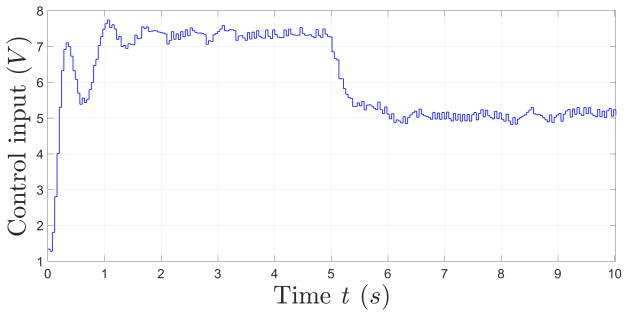


Figure 7.69: Control input voltage u

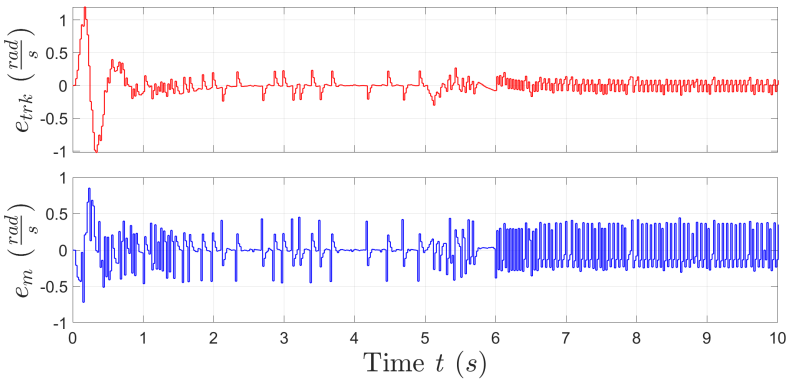


Figure 7.70: Tracking error e_{trk} and model error e_m

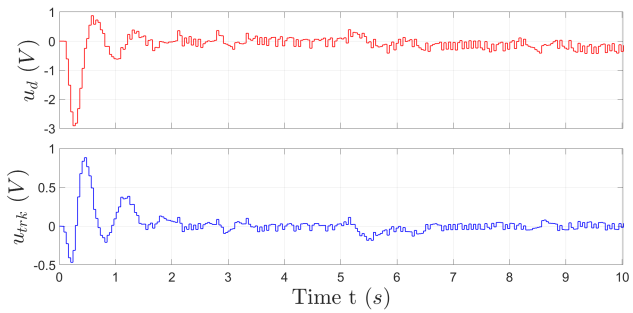


Figure 7.71: Control input parts u_d and u_{trk}

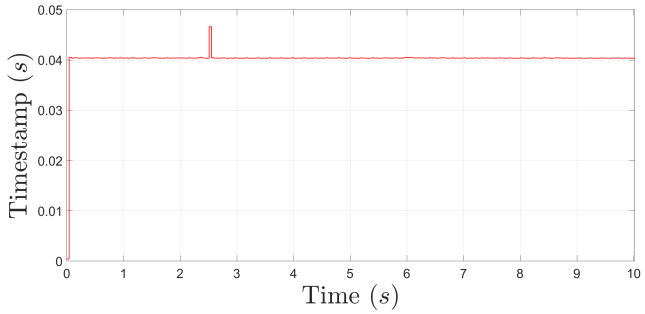


Figure 7.72: Measured Timestamp T

7.1.17 Second order disturbance, No load conditions - Result 5

Test settings: Disturbance rejection, Variable sampling time $T = [0.02, 0.04]$ s, timer Control and EM unified, target speed $[6, 4]$ rad/s.

Eigenvalue type	CT eigenvalues	Related DT at $T = 0.02\,\text{s}$
Control μ_K	$[-2.5647, -2.5647]$	$[0.95, 0.95]$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842]$	$[0.75, 0.75]$

Table 7.15: CT eigenvalues DC motor EMC Left motor v1.0 - 2nd order disturbance, No load conditions, Result 5

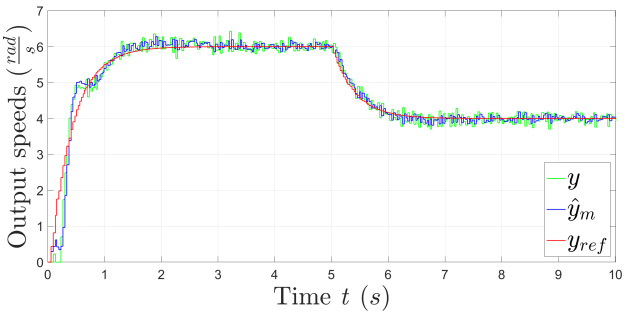


Figure 7.73: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

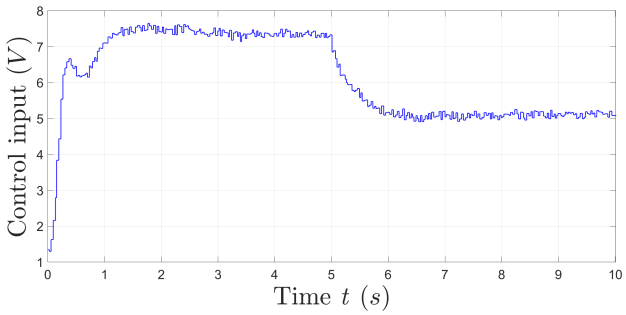


Figure 7.74: Control input voltage u

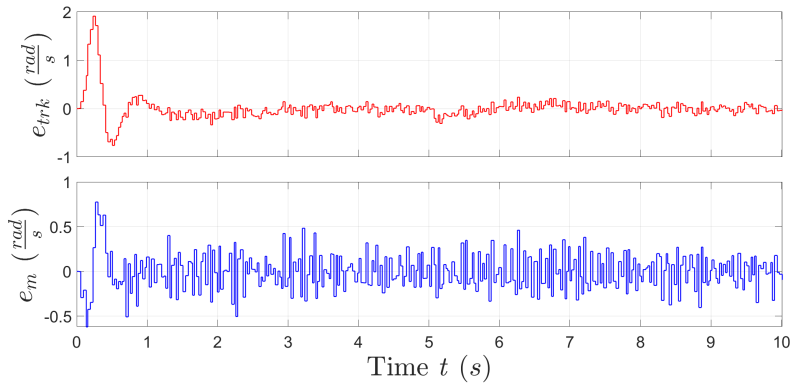


Figure 7.75: Tracking error e_{trk} and model error e_m

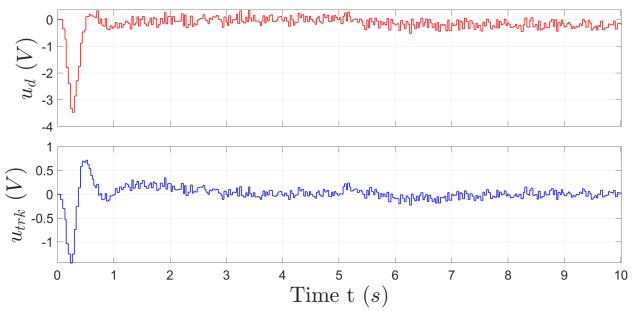


Figure 7.76: Control input parts u_d and u_{trk}

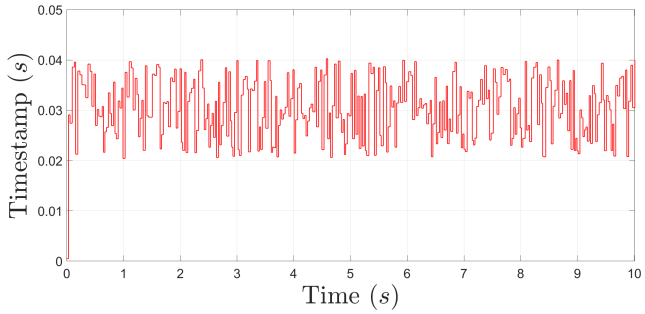


Figure 7.77: Measured Timestamp T

7.1.18 Second order disturbance, No load conditions - Result 6

Test settings: Disturbance rejection, Variable sampling time $T = [0.02, 0.04]$ s, timer Control and EM separated, target speed $[6, 4]$ rad/s.

Eigenvalue type	CT eigenvalues	Related DT at $T = 0.02$ s
Control μ_K	$[-2.5647, -2.5647]$	$[0.95, 0.95]$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842]$	$[0.75, 0.75]$

Table 7.16: CT eigenvalues DC motor EMC Left motor v1.0 - 2nd order disturbance, No load conditions, Result 6

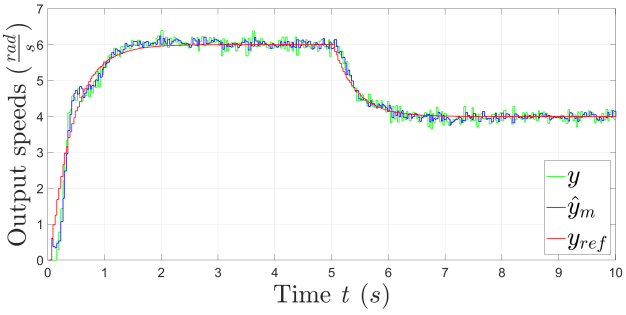


Figure 7.78: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

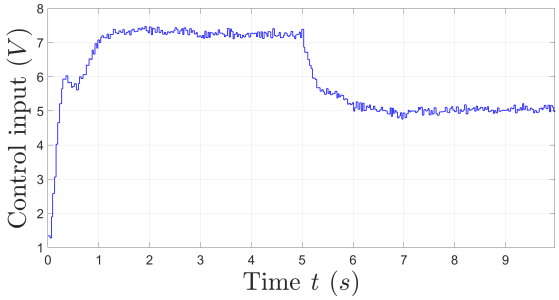


Figure 7.79: Control input voltage u

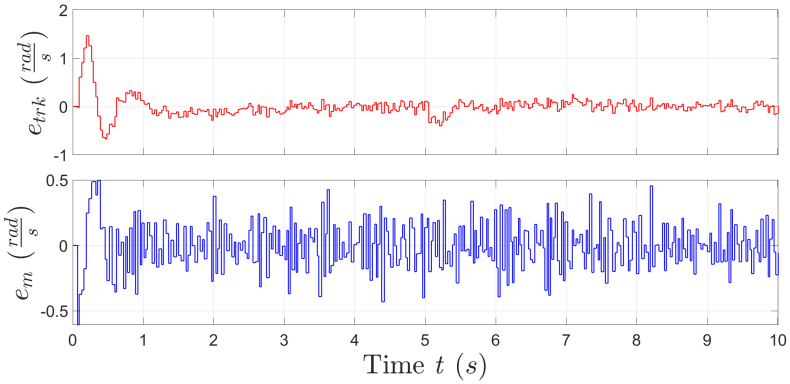


Figure 7.80: Tracking error e_{trk} and model error e_m

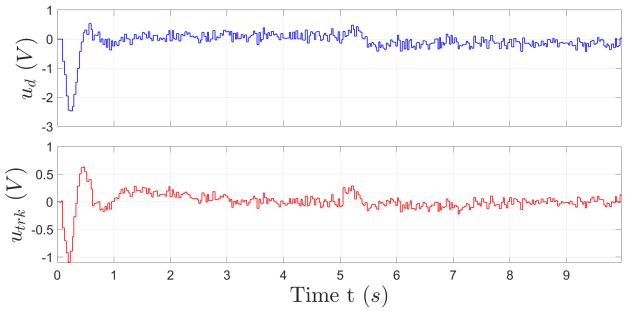


Figure 7.81: Control input parts u_d and u_{trk}

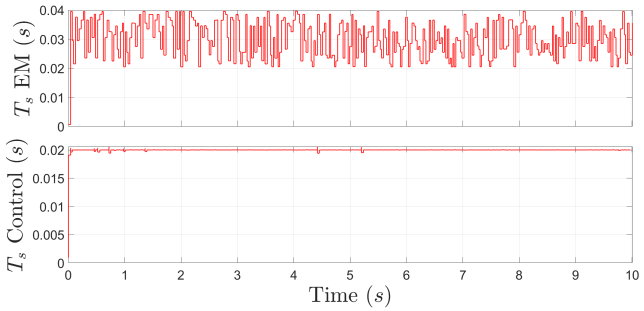


Figure 7.82: Measured Timestamp T

7.1.19 Second order disturbance, No load conditions - Result 7

Test settings: Disturbance rejection, Variable sampling time $T = [0.01, 0.03]$ s, timer Control and EM separated, target speed $[6, 4]$ rad/s.

Eigenvalue type	CT eigenvalues	Related DT at $T = 0.02$ s
Control μ_K	$[-2.5647, -2.5647]$	$[0.95, 0.95]$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842]$	$[0.75, 0.75]$

Table 7.17: CT eigenvalues DC motor EMC Left motor v1.0 - 2nd order disturbance, No load conditions, Result 7

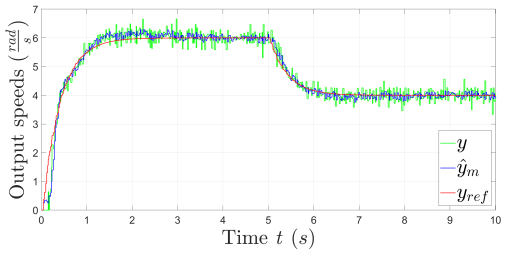


Figure 7.83: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

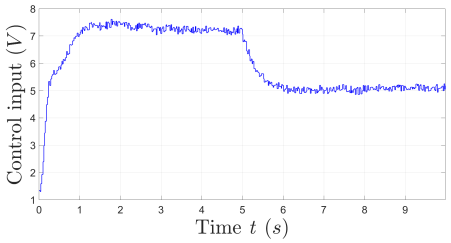


Figure 7.84: Control input voltage u

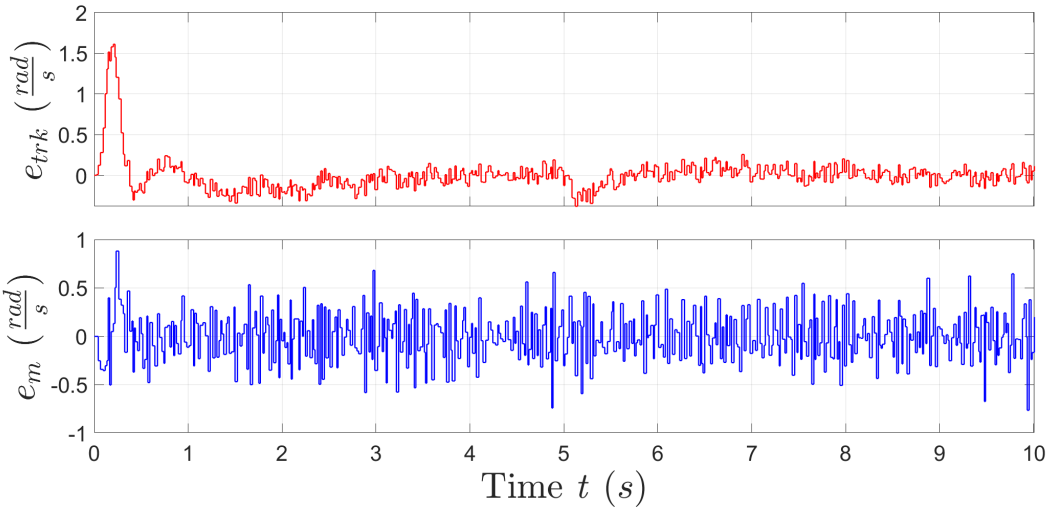


Figure 7.85: Tracking error e_{trk} and model error e_m

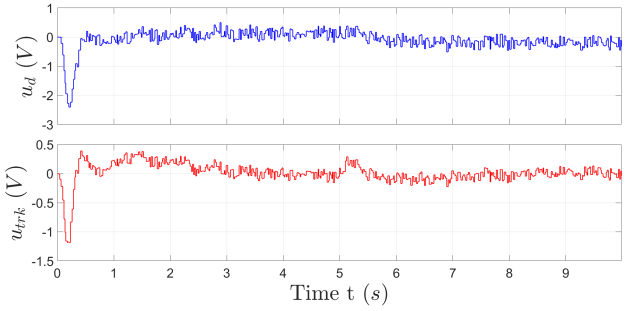


Figure 7.86: Control input parts u_d and u_{trk}

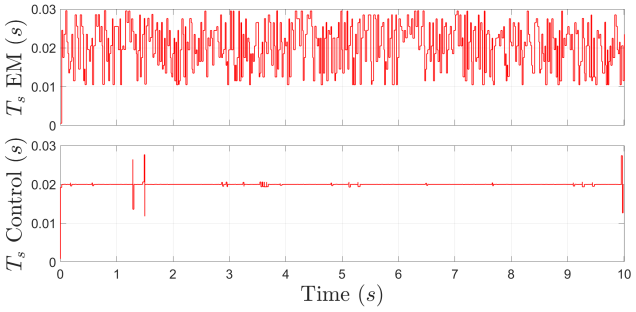


Figure 7.87: Measured Timestamp EM T_{EM}

7.1.20 Second order disturbance, No load conditions - Result 8

Test settings: Disturbance rejection, Variable sampling time $T = [0.01, 0.03]$ s, timer Control and EM separated, target speed $[8, 3, 5.5, 7]$ rad/s.

Eigenvalue type	CT eigenvalues	Related DT at $T = 0.02\,\text{s}$
Control μ_K	$[-2.5647, -2.5647]$	$[0.95, 0.95]$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842]$	$[0.75, 0.75]$

Table 7.18: CT eigenvalues DC motor EMC Left motor v1.0 - 2nd order disturbance, No load conditions, Result 8

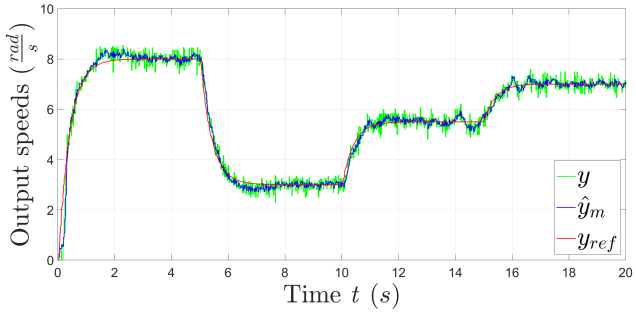


Figure 7.88: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

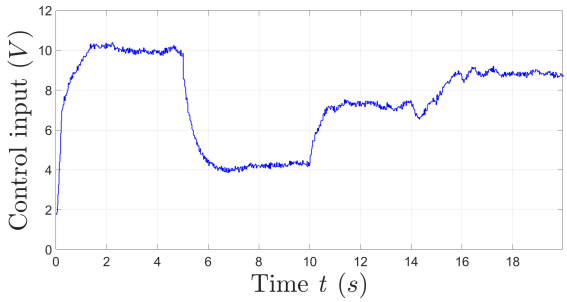


Figure 7.89: Control input voltage u

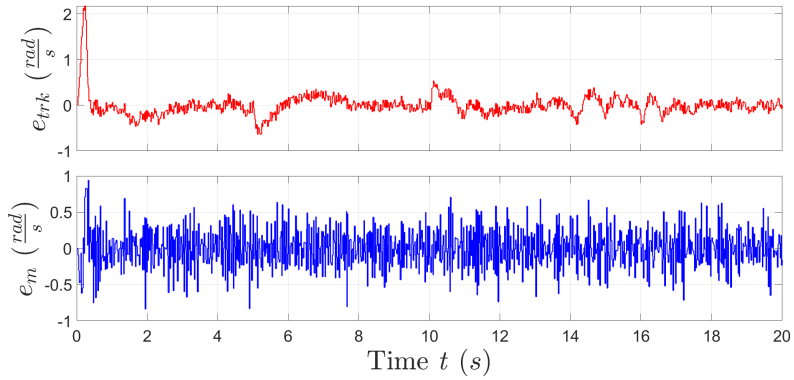


Figure 7.90: Tracking error e_{trk} and model error e_m

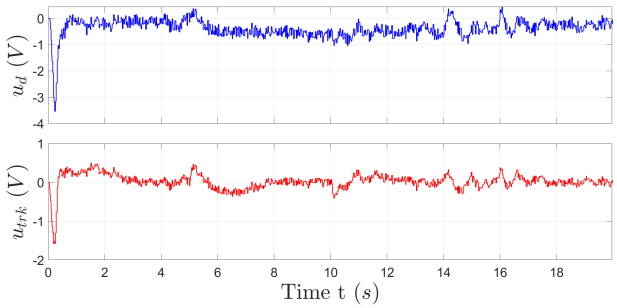


Figure 7.91: Control input parts u_d and u_{trk}

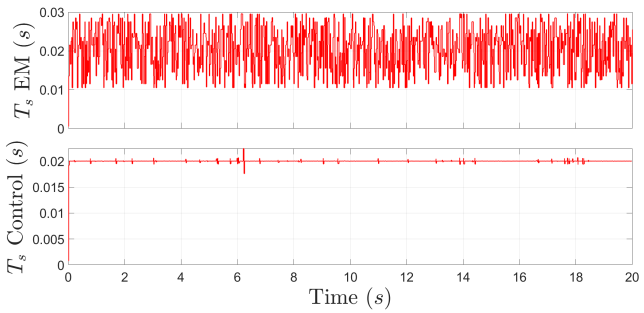


Figure 7.92: Measured Timestamp T

7.1.21 Second order disturbance, No load conditions - Result 9

Test settings: No disturbance rejection, Variable sampling time $T = [0.02, 0.04]$ s, timer Control and EM unified, target speed $[6, 4]$ rad/s.

Eigenvalue type	CT eigenvalues	Related DT at $T = 0.02$ s
Control μ_K	$[-2.5647, -2.5647]$	$[0.95, 0.95]$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842]$	$[0.75, 0.75]$

Table 7.19: CT eigenvalues DC motor EMC Left motor v1.0 - 2nd order disturbance, No load conditions, Result 9

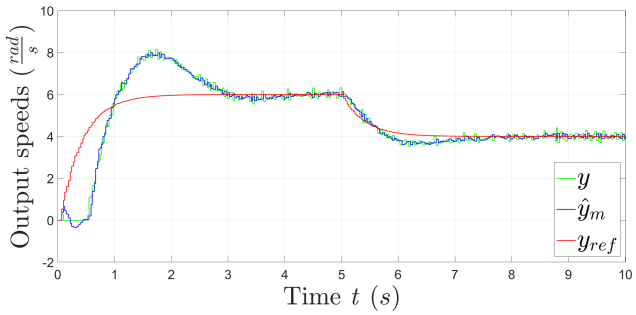


Figure 7.93: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

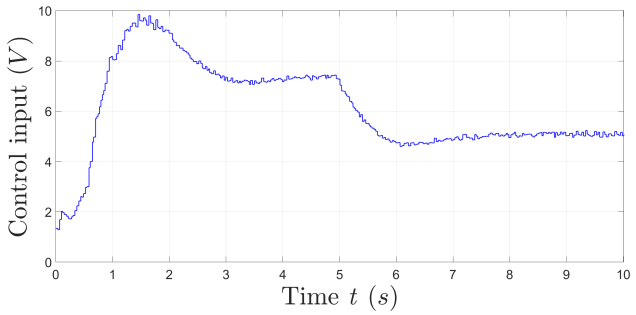


Figure 7.94: Control input voltage u

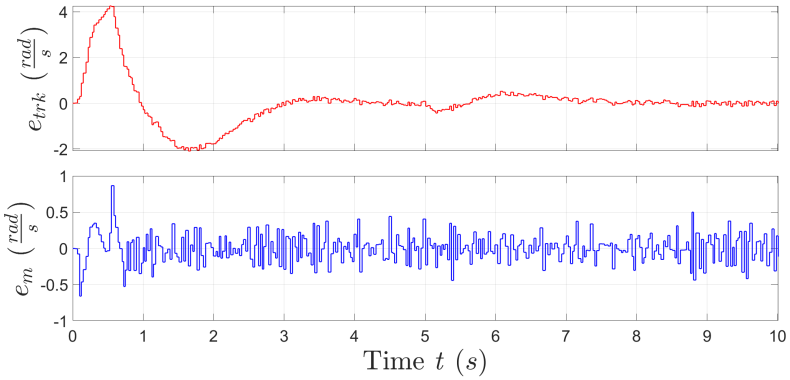


Figure 7.95: Tracking error e_{trk} and model error e_m

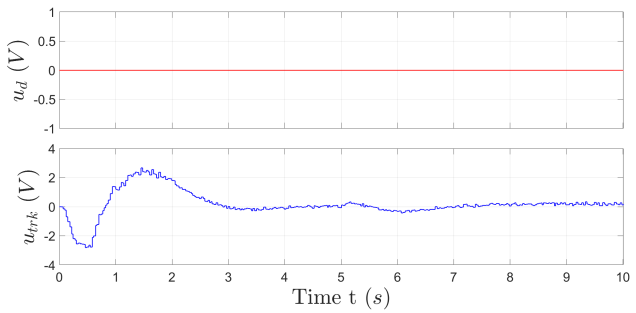


Figure 7.96: Control input parts u_d and u_{trk}

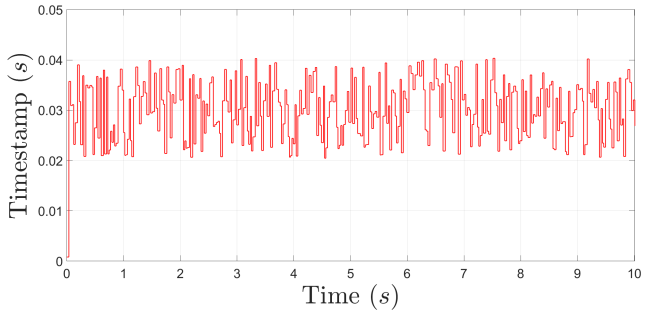


Figure 7.97: Measured Timestamp T

7.1.22 Second order disturbance, No load conditions - Result 10

Test settings: No disturbance rejection, Variable sampling time $T = [0.02, 0.04]$ s, timer Control and EM separated, target speed $[6, 4]$ rad/s.

Eigenvalue type	CT eigenvalues	Related DT at $T = 0.02$ s
Control μ_K	$[-2.5647, -2.5647]$	$[0.95, 0.95]$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842]$	$[0.75, 0.75]$

Table 7.20: CT eigenvalues DC motor EMC Left motor v1.0 - 2nd order disturbance, No load conditions, Result 10

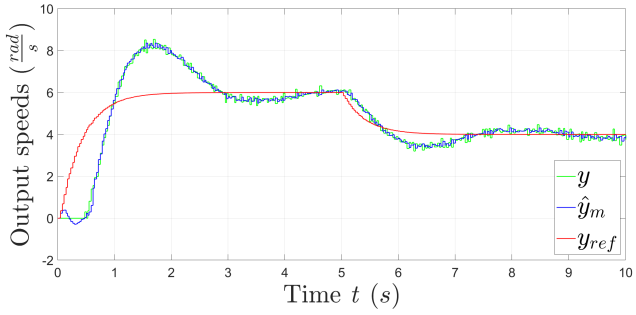


Figure 7.98: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

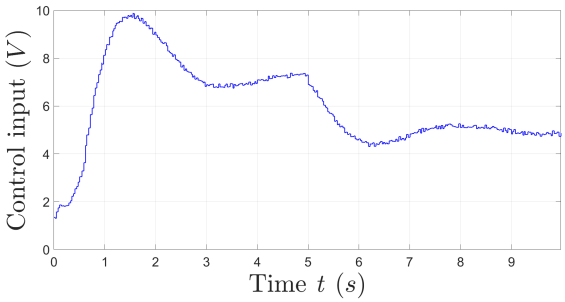


Figure 7.99: Control input voltage u

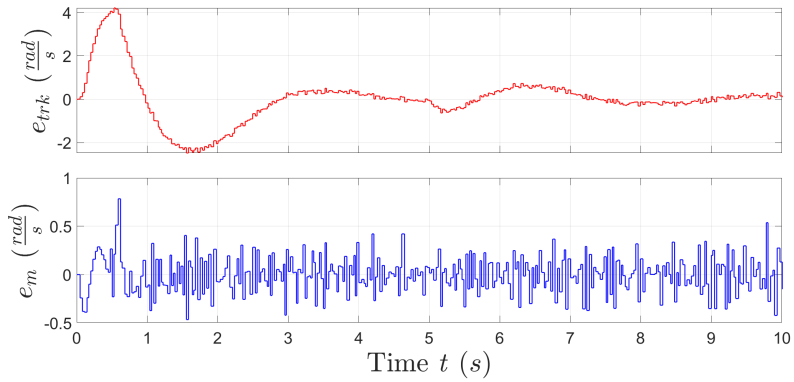


Figure 7.100: Tracking error e_{trk} and model error e_m

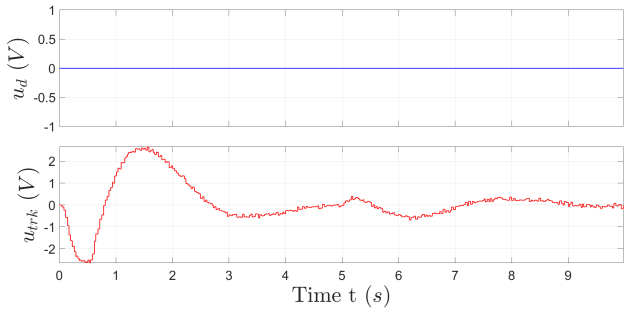


Figure 7.101: Control input parts u_d and u_{trk}

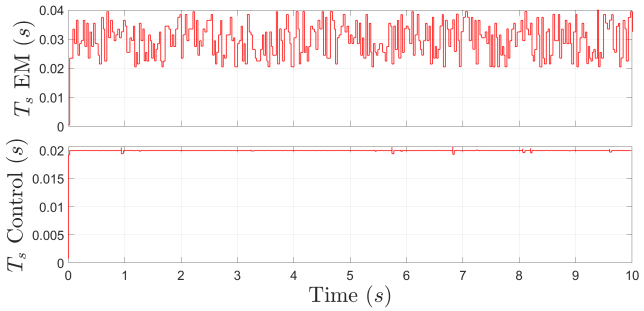


Figure 7.102: Measured Timestamp EM T_{EM}

7.1.23 Second order disturbance, Load conditions - Result 1

Test settings: Disturbance rejection, Variable sampling time $T = [0.01, 0.03]$ s, timer Control and EM separated, target speed $[6, 4]$ rad/s.

Eigenvalue type	CT eigenvalues	Related DT at $T = 0.02\,\mathrm{s}$
Control μ_K	$[-2.5647, -2.5647]$	$[0.95, 0.95]$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842]$	$[0.75, 0.75]$

Table 7.21: CT eigenvalues DC motor EMC Left motor v1.0 - 2nd order disturbance, Load conditions, Result 1

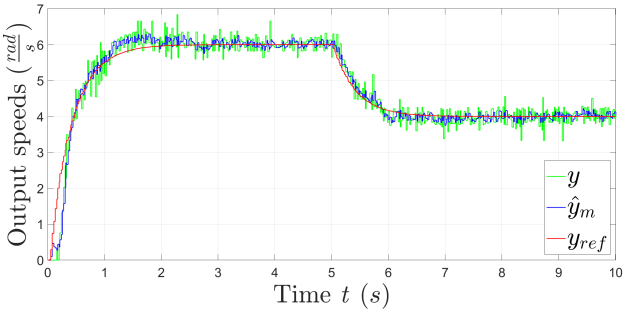


Figure 7.103: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

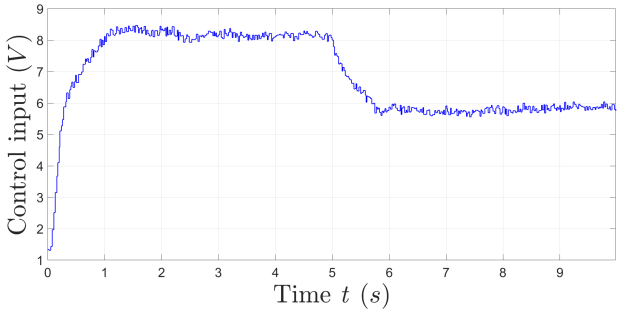


Figure 7.104: Control input voltage u

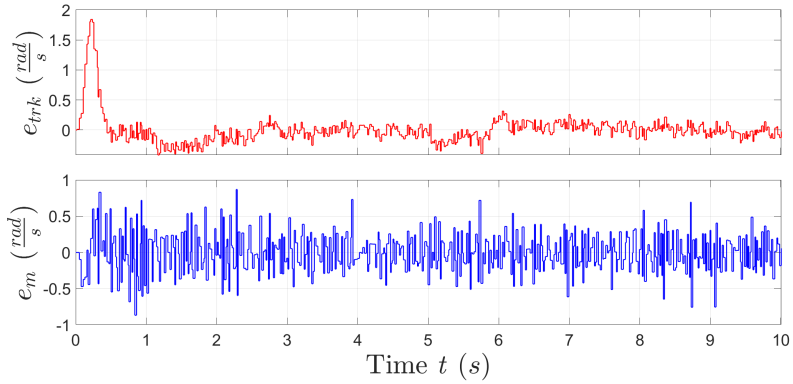


Figure 7.105: Tracking error e_{trk} and model error e_m

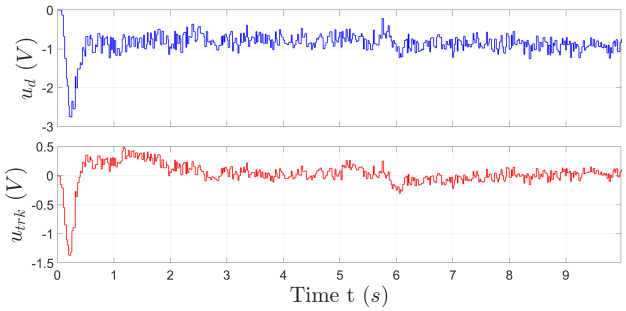


Figure 7.106: Control input parts u_d and u_{trk}

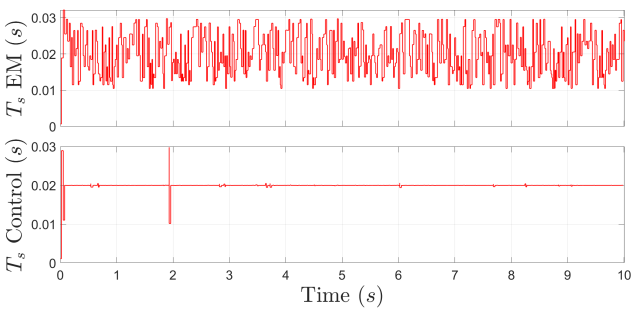


Figure 7.107: Measured Timestamp T

7.1.24 Second order disturbance, Load conditions - Result 2

Test settings: No disturbance rejection, Variable sampling time $T = [0.01, 0.03]$ s, timer Control and EM separated, target speed $[6, 4]$ rad/s.

Eigenvalue type	CT eigenvalues	Related DT at $T = 0.02$ s
Control μ_K	$[-2.5647, -2.5647]$	$[0.95, 0.95]$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842]$	$[0.75, 0.75]$

Table 7.22: CT eigenvalues DC motor EMC Left motor v1.0 - 2nd order disturbance, Load conditions, Result 2

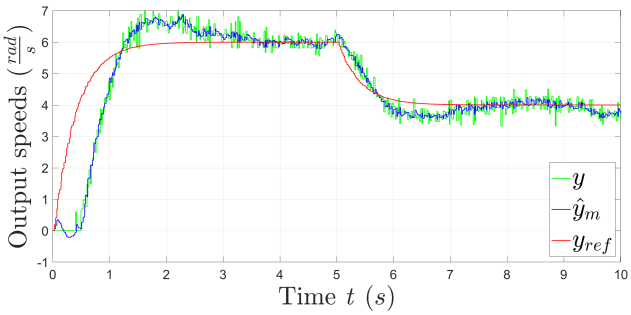


Figure 7.108: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

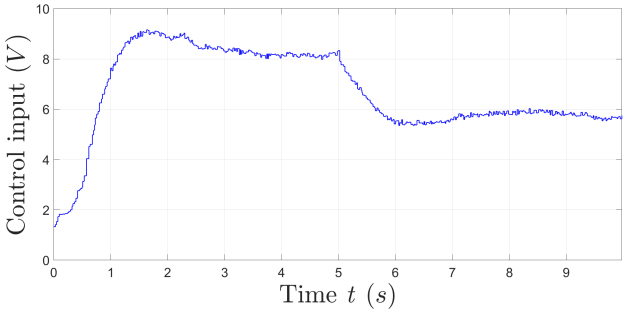


Figure 7.109: Control input voltage u

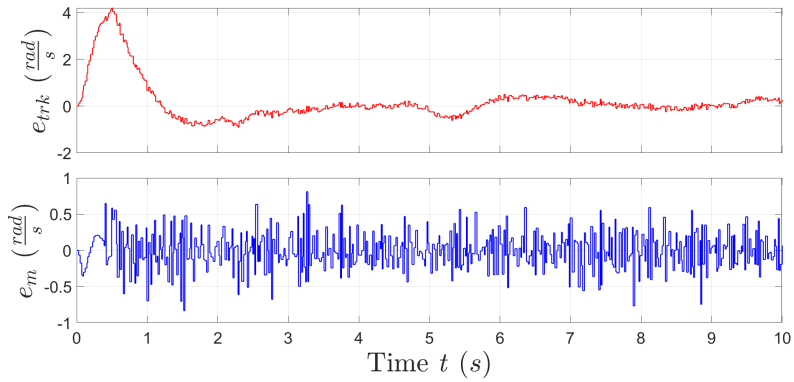


Figure 7.110: Tracking error e_{trk} and model error e_m

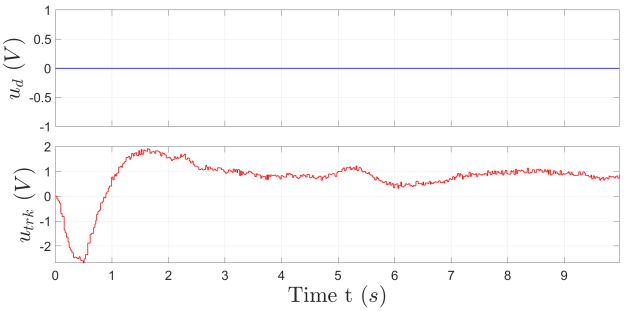


Figure 7.111: Control input parts u_d and u_{trk}

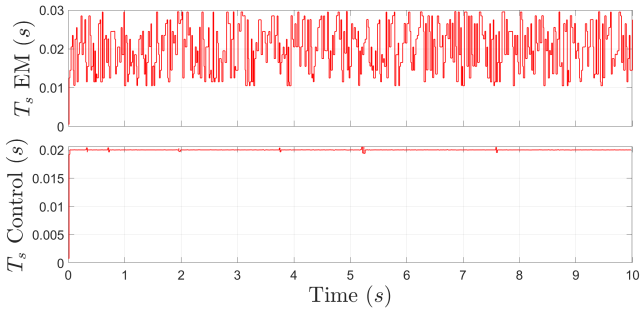


Figure 7.112: Measured Timestamp T

7.1.25 Second order disturbance, Load conditions - Result 3

Test settings: Disturbance rejection, Variable sampling time $T = [0.01, 0.03]$ s, timer Control and EM separated, target speed $[6.5, 3, 5]$ rad/s.

Eigenvalue type	CT eigenvalues	Related DT at $T = 0.02$ s
Control μ_K	$[-2.5647, -2.5647]$	$[0.95, 0.95]$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842]$	$[0.75, 0.75]$

Table 7.23: CT eigenvalues DC motor EMC Left motor v1.0 - 2nd order disturbance, Load conditions, Result 3

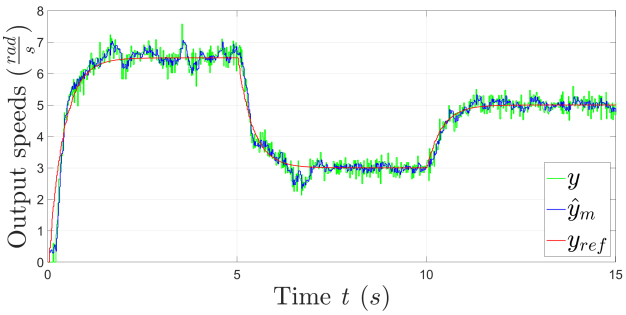


Figure 7.113: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

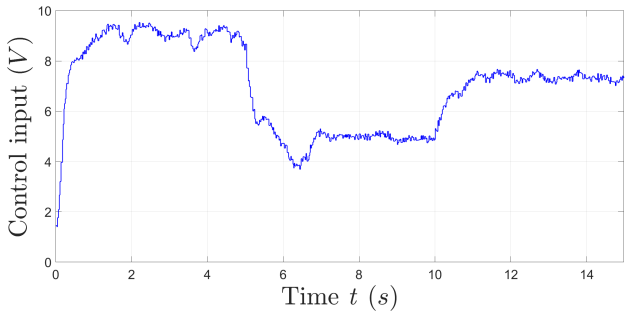


Figure 7.114: Control input voltage u

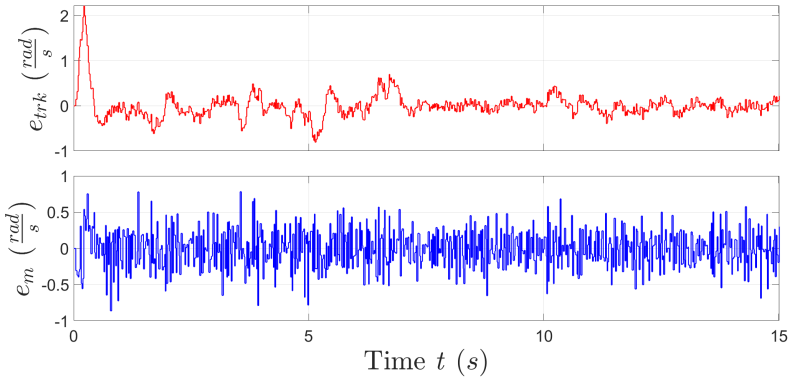


Figure 7.115: Tracking error e_{trk} and model error e_m

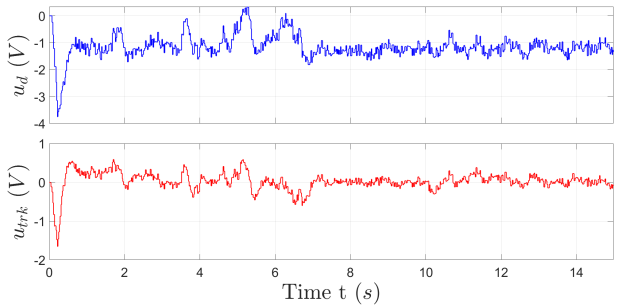


Figure 7.116: Control input parts u_d and u_{trk}

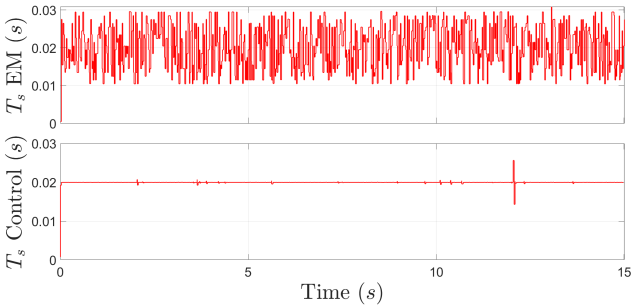


Figure 7.117: Measured Timestamp T

7.2 Left motor version 1.1 - RHIT results

In this part, some RHIT physical implementation tests for left motor EMc version 1.1 are performed.

7.2.1 Main settings

The same settings considered for Left motor v1.0 are considered for this version, Section 7.1.
Every set of plots is composed again by motor outputs, control input u with its components, tracking and model errors and Timestamp. EMC is compared in load and no load conditions, using or not disturbance rejection, splitting or not in EM and control parts. Different target speeds are also considered.
Now is not needed anymore verify the control model with fixed sampling time, hence all tests are performed using variable sampling time. The disturbance model for $\overline{w}$ is of 2^{nd} order. For other specific informations refers to Section 7.1.1.
For all tests the CT eigenvalues are:

Eigenvalue type	CT eigenvalues	Related DT eigenvalues at $T = 0.02\,\text{s}$
Control μ_K	$[-2.5647, -2.5647]$	$[0.95, 0.95]$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842]$	$[0.75, 0.75]$

Table 7.24: CT DC motor EMC left motor v1.1

7.2.2 Summary on the results

The main difference wrt left motor EMC v1.0 is the change of motor mechanical part parameters (τ_m and k_v'): this lead to lower difference of control model wrt robot plant, and better disturbance rejection. This is for example proven by looking at some load tests with difference reference speeds to be tracked, Section 7.2.7 and Section 7.1.25.
Apart from that, no other differences are found, the main result objectives are the same of Section 7.1.2 for EMC left motor v1.0.

7.2.3 No load conditions - Result 1

Test settings: Disturbance rejection, Variable sampling time $T = [0.02, 0.04]$ s, timer Control and EM unified, target speed $[6, 4]$ rad/s.

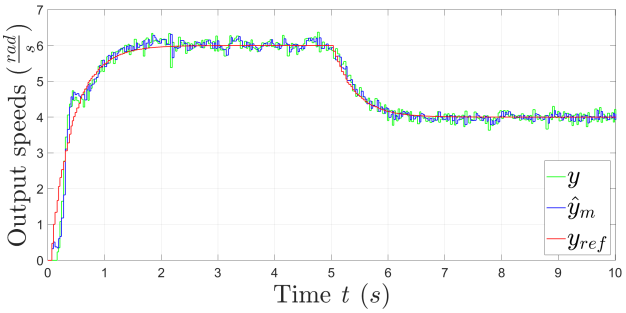


Figure 7.118: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

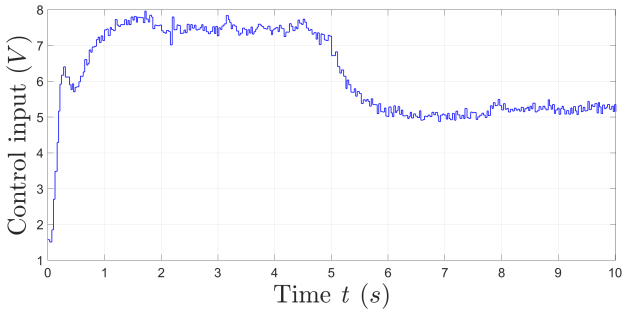


Figure 7.119: Control input voltage u

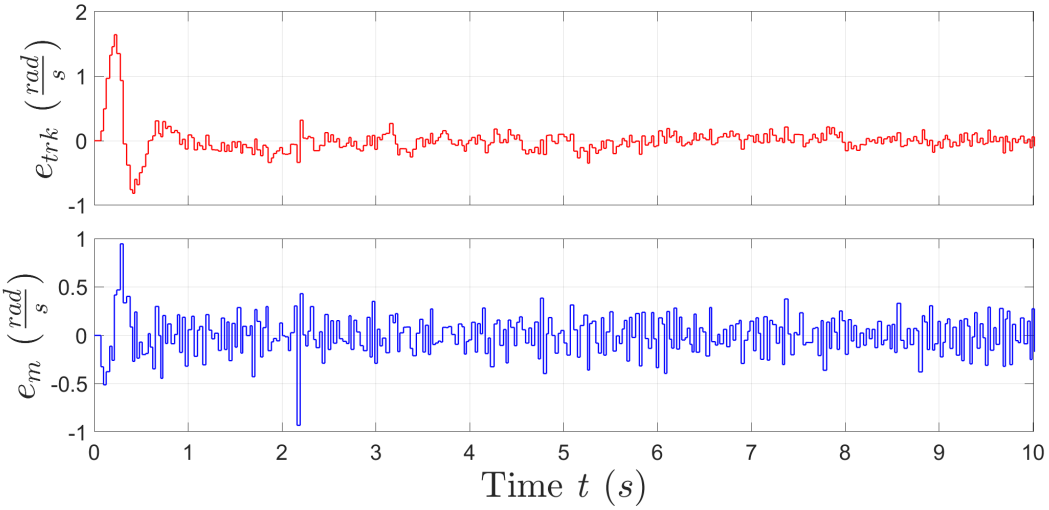


Figure 7.120: Tracking error e_{trk} and model error e_m

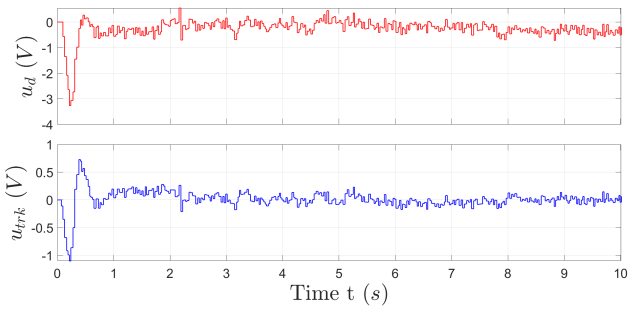


Figure 7.121: Control input parts u_d and u_{trk}

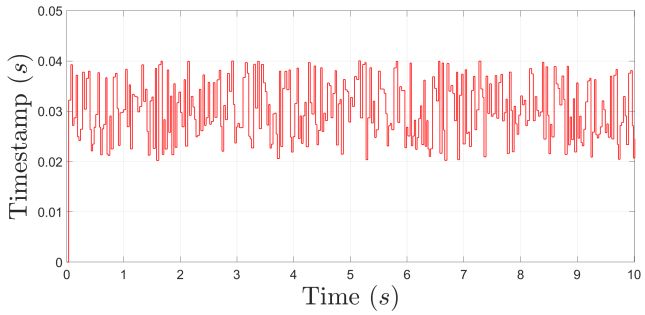


Figure 7.122: Measured Timestamp T

7.2.4 No load conditions - Result 2

Test settings: Disturbance rejection, Variable sampling time T = $[0.01, 0.03]$ s, timer Control and EM unified, target speed $[8, 3, 5.5, 7]$ rad/s.

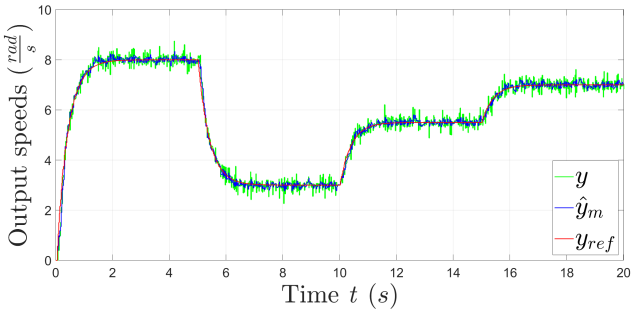


Figure 7.123: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

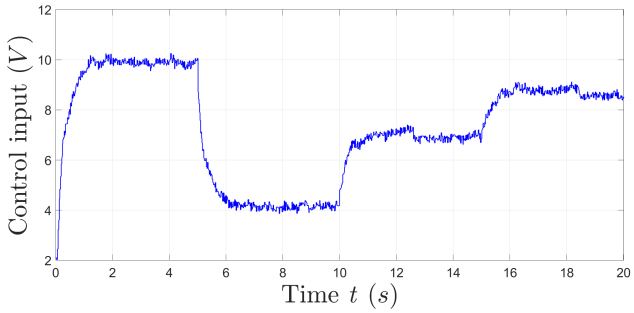


Figure 7.124: Control input voltage u

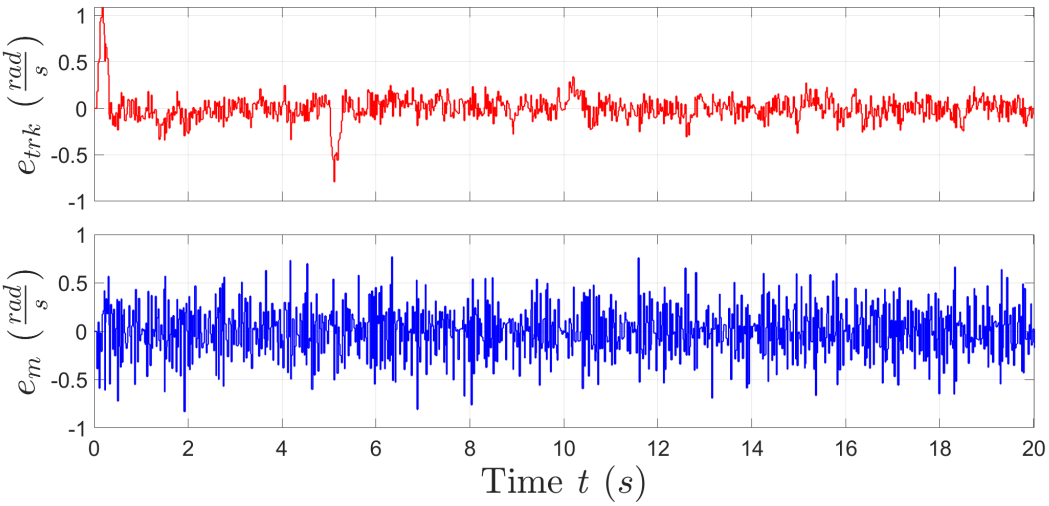


Figure 7.125: Tracking error $e_{_{trk}}$ and model error e_m

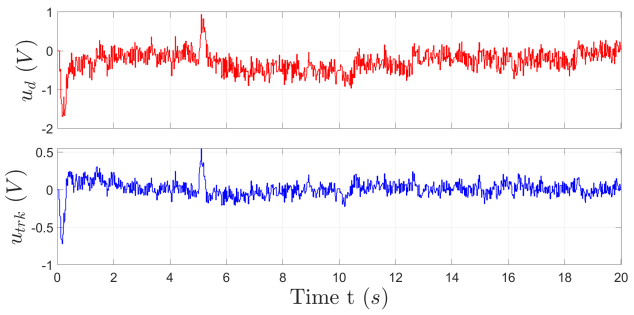


Figure 7.126: Control input parts u_d and $u_{_{trk}}$

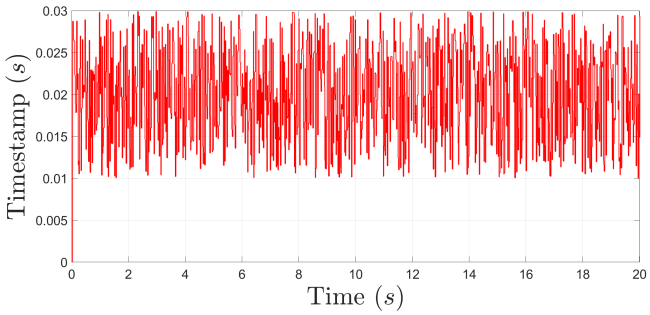


Figure 7.127: Measured Timestamp T

7.2.5 No load conditions - Result 3

Test settings: Disturbance rejection, Variable sampling time T_{EM} = $[0.01, 0.03]$ s, timer Control and EM separated, target speed $[8, 3, 5.5, 7]$ rad/s.

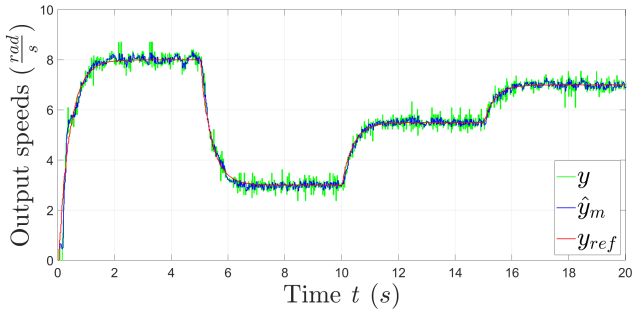


Figure 7.128: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

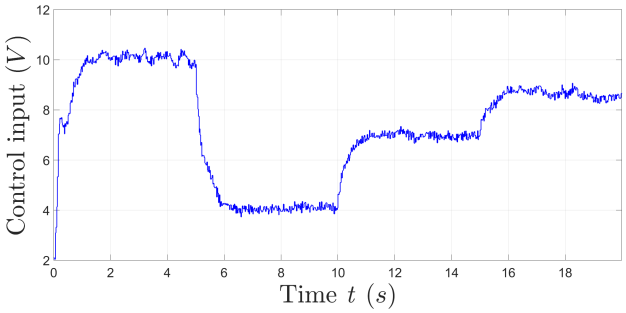


Figure 7.129: Control input voltage u

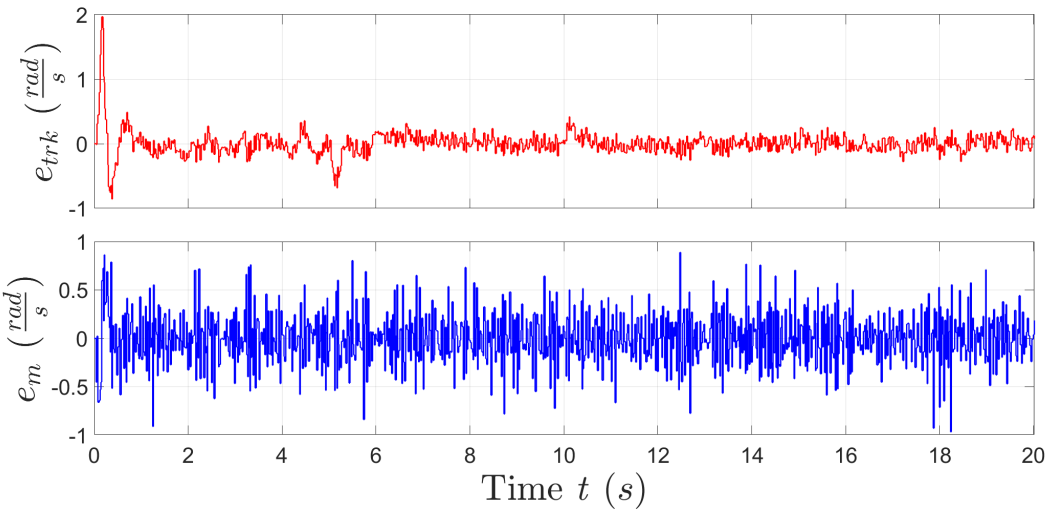


Figure 7.130: Tracking error e_{trk} and model error e_m

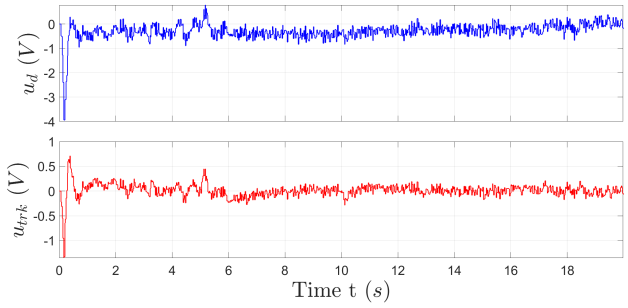


Figure 7.131: Control input parts u_d and u_{trk}

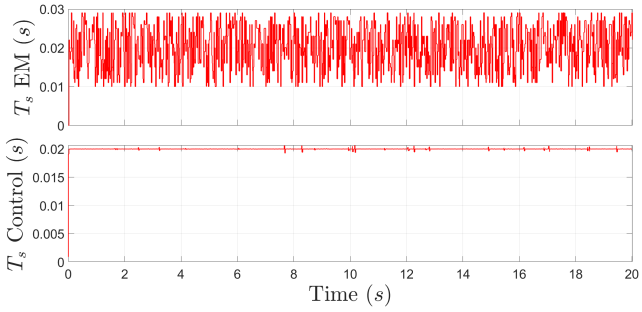


Figure 7.132: Measured Timestamp T

7.2.6 No load conditions - Result 4

Test settings: No disturbance rejection, Variable sampling time T_{EM} = $[0.02, 0.04]$ s, timer Control and EM separated, target speed $[6, 4]$ rad/s.

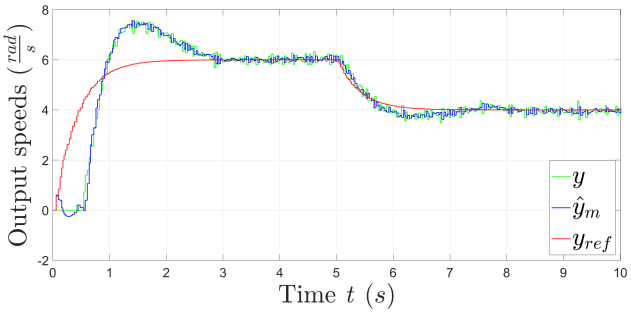


Figure 7.133: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

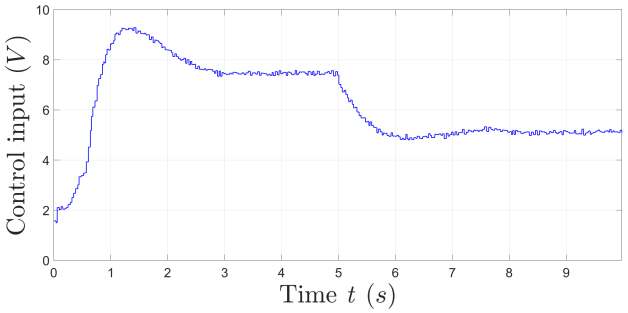


Figure 7.134: Control input voltage u

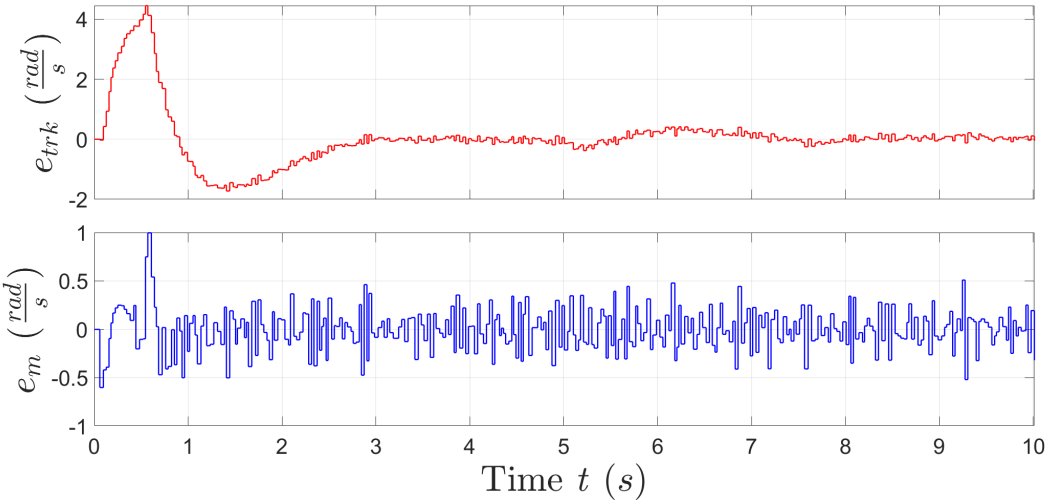


Figure 7.135: Tracking error e_{trk} and model error e_m

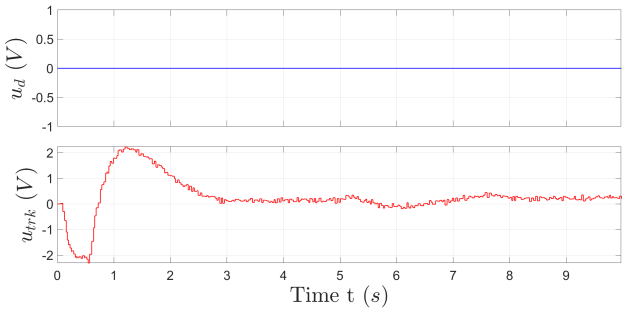


Figure 7.136: Control input parts u_d and u_{trk}

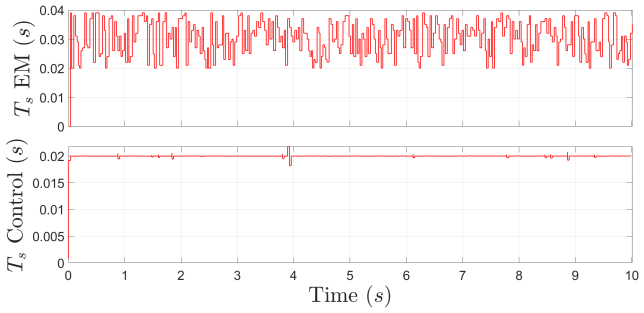


Figure 7.137: Measured Timestamp T

7.2.7 Load conditions - Result 1

Test settings: Disturbance rejection, Variable sampling time T_{EM} = $[0.01, 0.03]$ s, timer Control and EM separated, target speed $[6.5, 3, 5]$ rad/s.

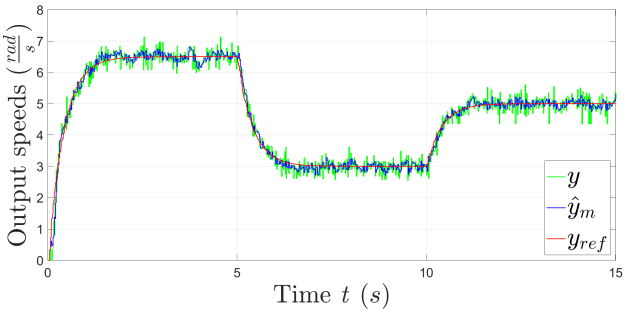


Figure 7.138: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

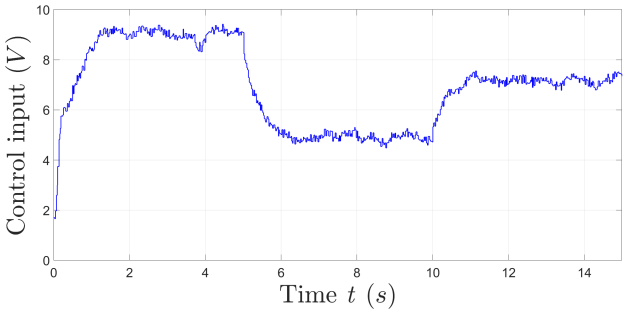


Figure 7.139: Control input voltage NOT filtered u

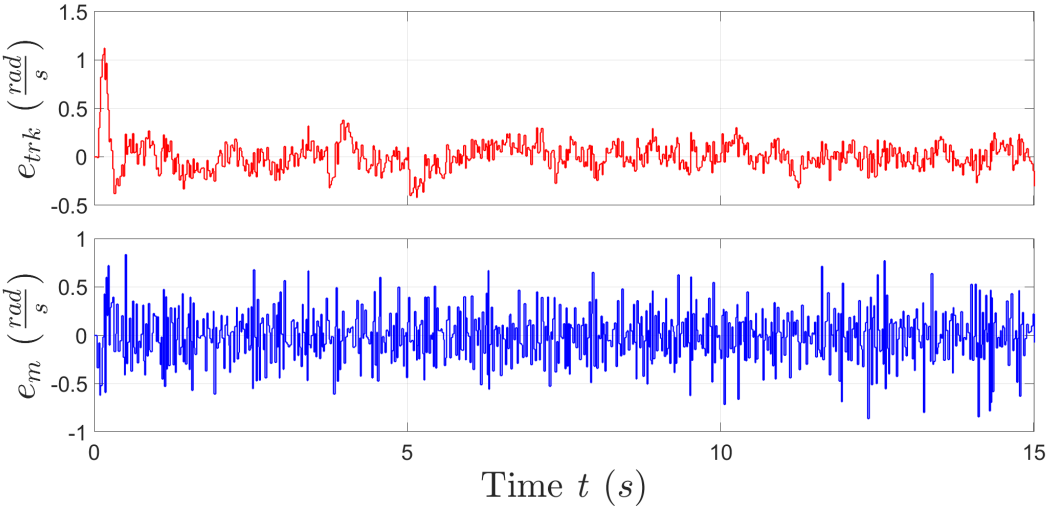


Figure 7.140: Tracking error e_{trk} and model error e_m

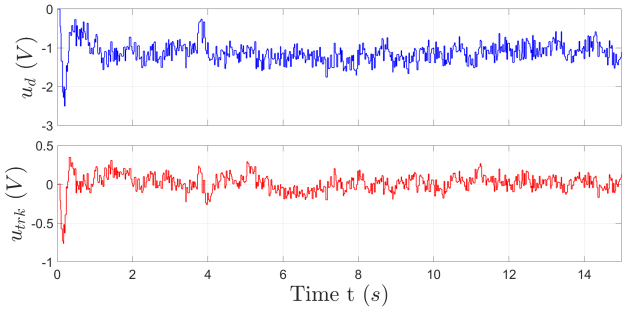


Figure 7.141: Control input parts u_d and u_{trk}

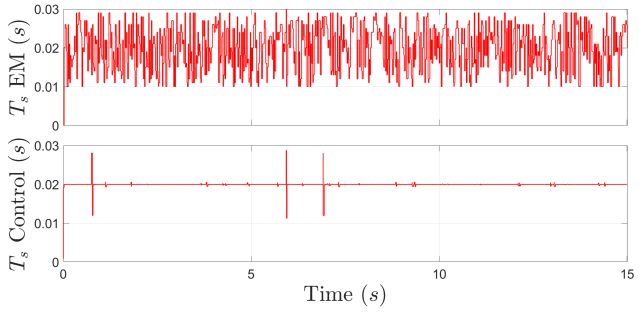


Figure 7.142: Measured Timestamp T

7.3 Right motor v1.2 - RHIT results

For the right motor tests, version 1.2 considering the parameters identification with best results.

7.3.1 Main settings

The very same implementation settings used for left motor EMC v1.0 are considered, Section 7.1.1. However the following settings are common for all tests:

- Continuous eigenvalues used:

Eigenvalue type	CT eigenvalues	Related DT eigenvalues at $T = 0.02\,\mathrm{s}$
Control μ_K	$[-2.5647, -2.5647]$	$[0.95, 0.95]$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842]$	$[0.75, 0.75]$

Table 7.25: CT eigenvalues DC motor EMC

- Control and EM parts of EMC DC motor are separated and their codes ran with different sampling times. Control part sampling time was maintained always fixed at $T = 0.02\,\mathrm{s}$. Instead EM part is repeated every $T = [0.01 - 0.03]$ s.

- 2^{nd} order model disturbance for $\overline{w}$ is used.

7.3.2 Summary on the results

The main results are almost the same as the ones already described in Section 7.1.2 for EMC left motor v1.0.
Even using a simplified model (EM) for the right motor the tracking and model errors e_{trk} and e_m are small and almost within the output speed resolution, as already explained in left motor EMC v1.0. In Load conditions robot is placed in floor, hence a load torque is added to the system as further disturbance. In this conditions robot DC motors are supplied by batteries, so the maximum voltage depend on their state of charge. A saturation is imposed in this case, and the output speed in some tests is limited.
Significant is the difference with and without the presence of disturbance rejection: even if e_m is almost the same, e_{trk} is very high in time transient phases of each output speed steps. This is mainly due to motors dynamic behaviour (voltage input dead-zones, check MIL fine model in Section 6.1). Important is to consider that this high difference is reduced when fastest control eigenvalues μ_K are considered (approximately $\mu_K < -5$).

7.3.3 No load conditions - Result 1

Test settings: Disturbance rejection, Target speed $[6, 4]$ rad/s.

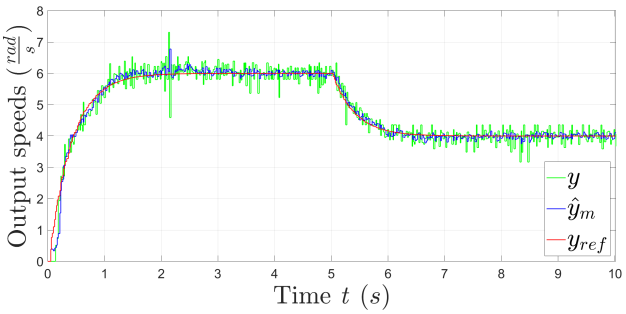


Figure 7.143: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

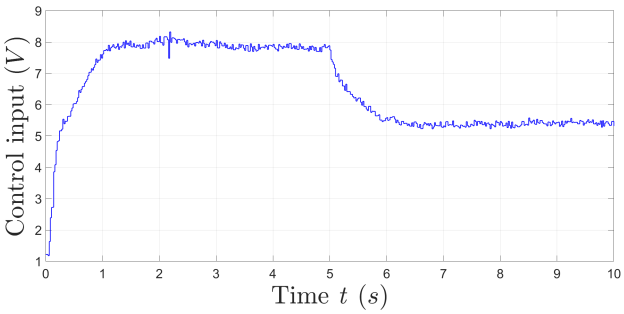


Figure 7.144: Control input voltage u

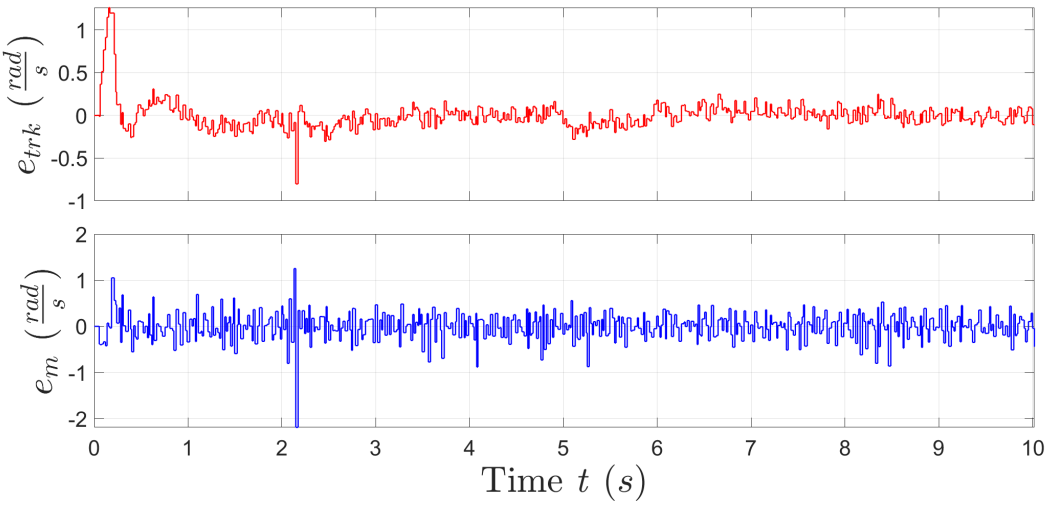


Figure 7.145: Tracking error e_{trk} and model error e_m

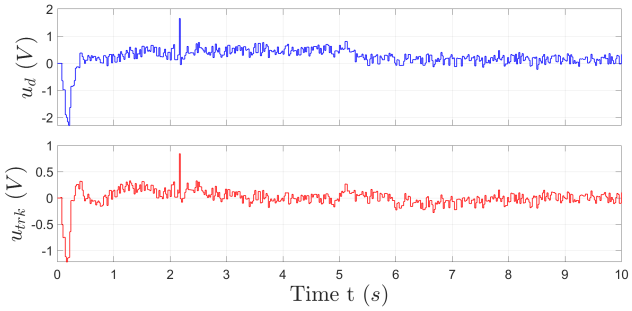


Figure 7.146: Control input parts u_d and u_{trk}

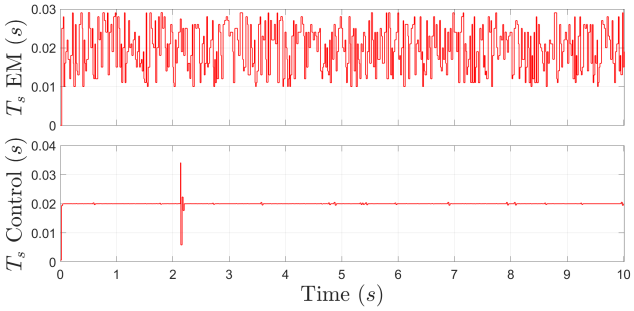


Figure 7.147: Measured Timestamp T

7.3.4 No load conditions - Result 2

Test settings: Disturbance rejection, Target speed $[8, 3, 5.5, 7]$ rad/s.

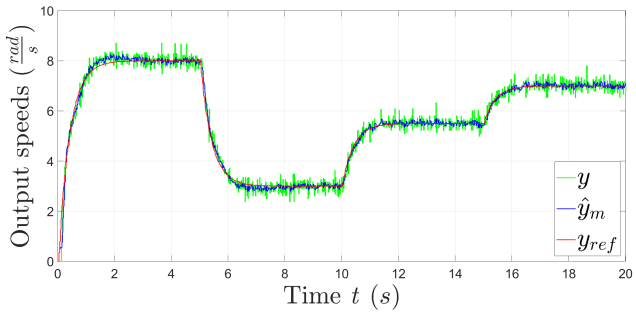


Figure 7.148: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

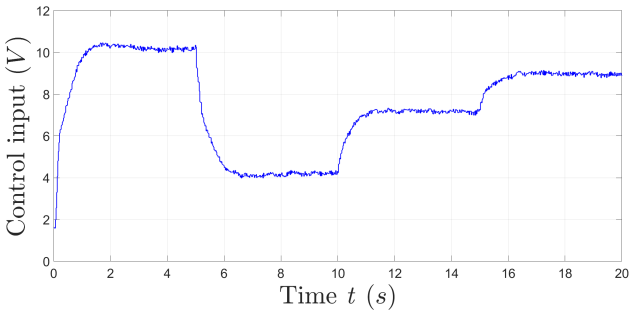


Figure 7.149: Control input voltage u

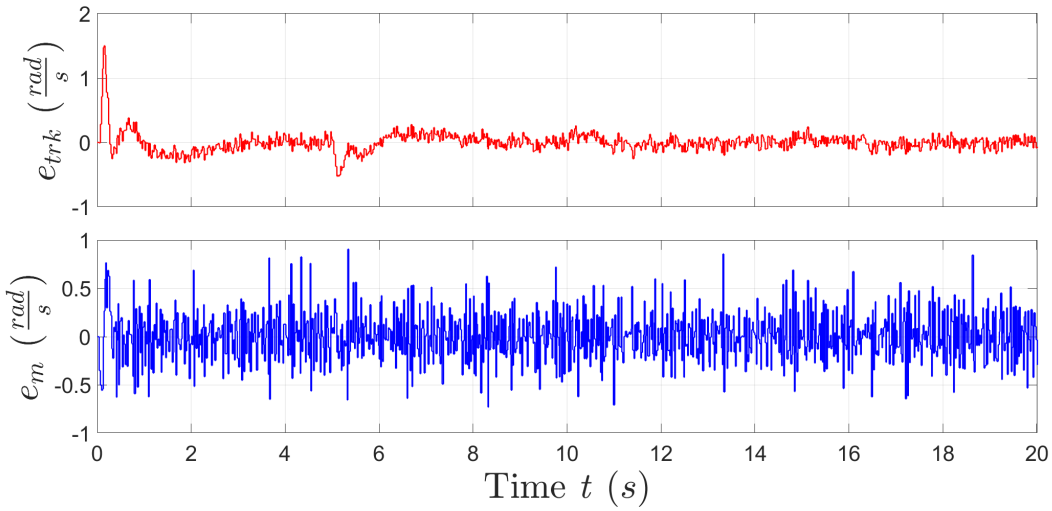


Figure 7.150: Tracking error e_{trk} and model error e_m

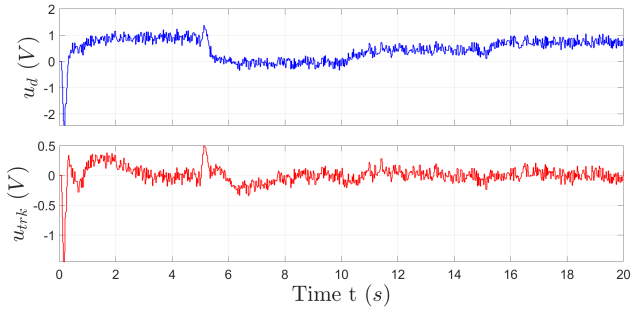


Figure 7.151: Control input parts u_d and u_{trk}

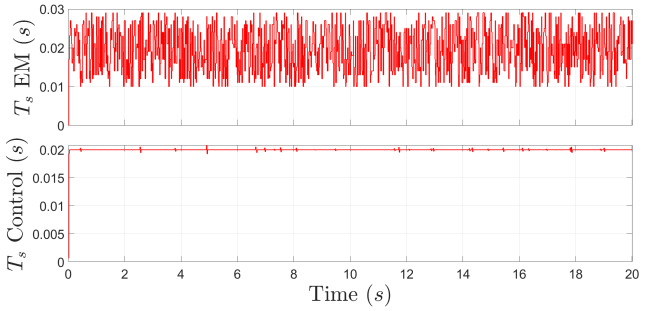


Figure 7.152: Measured Timestamp T

7.3.5 No load conditions - Result 3

Test settings: Disturbance rejection, Target speed $[7, -3, 4, -6.5]$ rad/s.

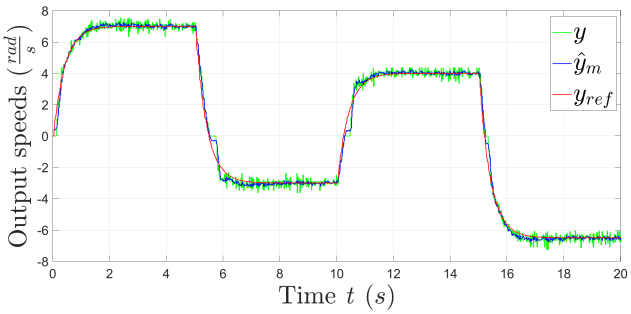


Figure 7.153: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

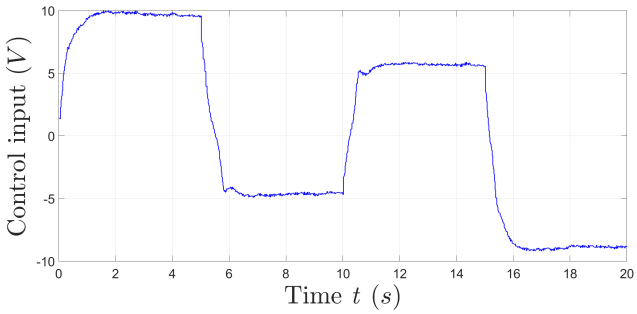


Figure 7.154: Control input voltage u

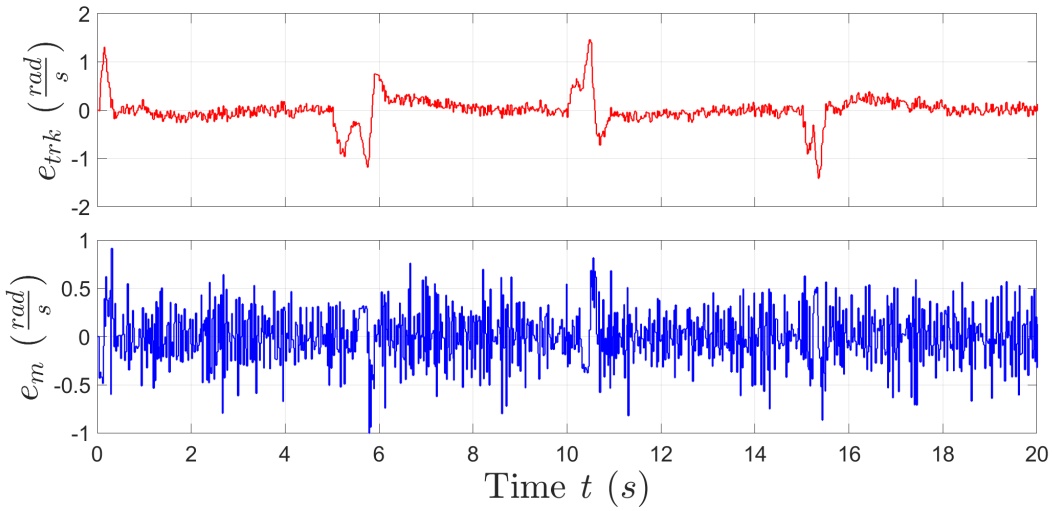


Figure 7.155: Tracking error e_{trk} and model error e_m

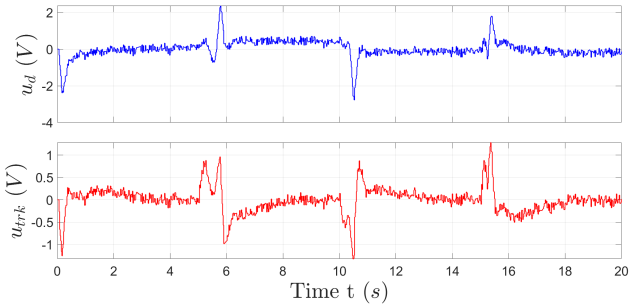


Figure 7.156: Control input parts u_d and u_{trk}

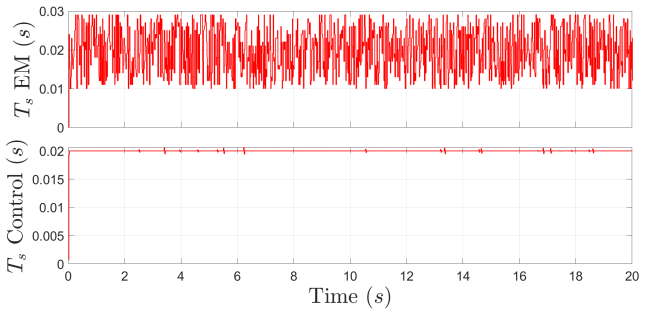


Figure 7.157: Measured Timestamp T

7.3.6 No load conditions - Result 4

Test settings: No disturbance rejection, Target speed $[6, 4]$ rad/s.

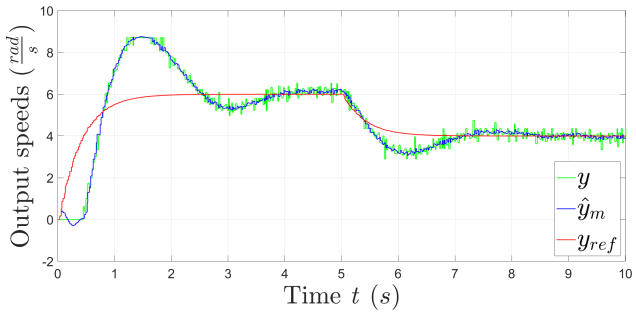


Figure 7.158: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

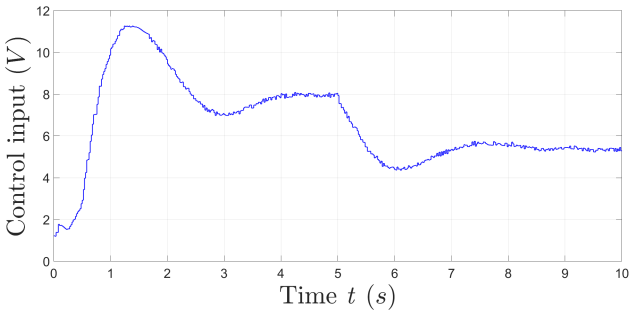


Figure 7.159: Control input voltage u

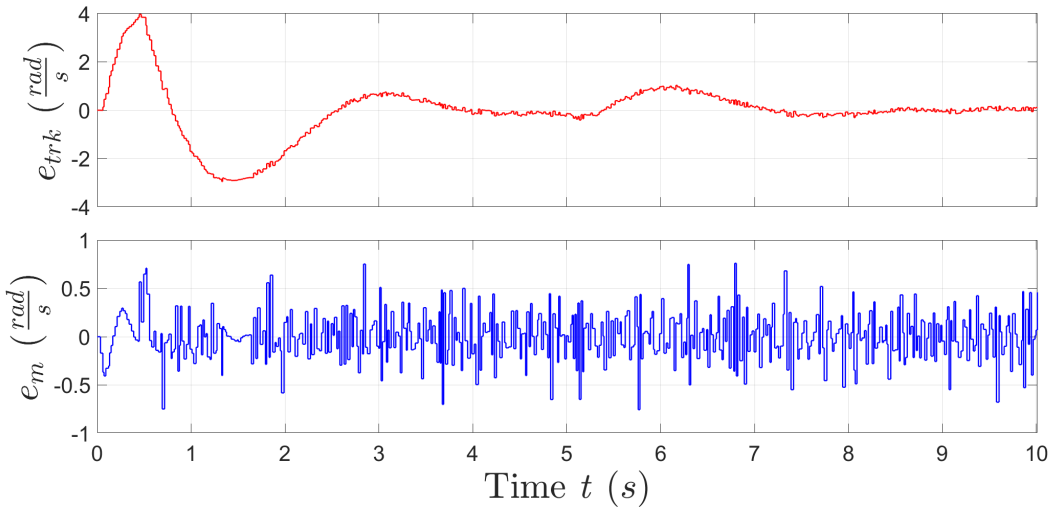


Figure 7.160: Tracking error e_{trk} and model error e_m

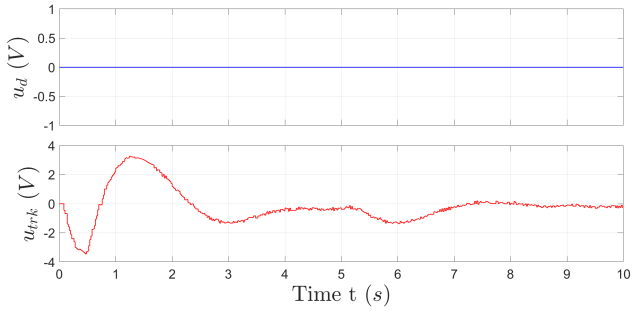


Figure 7.161: Control input parts u_d and u_{trk}

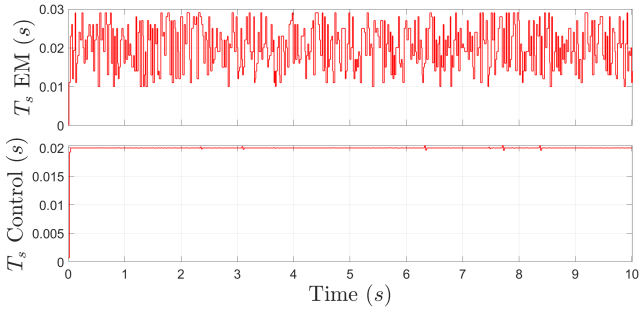


Figure 7.162: Measured Timestamp T

7.3.7 No load conditions - Result 5

Test settings: No disturbance rejection, Target speed $[8, 3, 5.5, 7]$ rad/s.

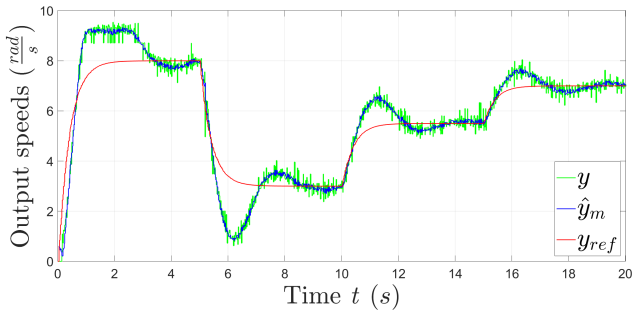


Figure 7.163: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

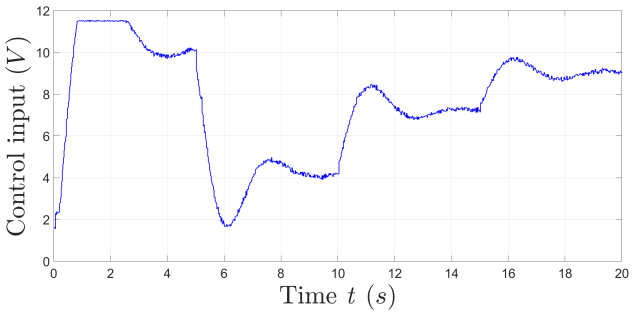


Figure 7.164: Control input voltage u

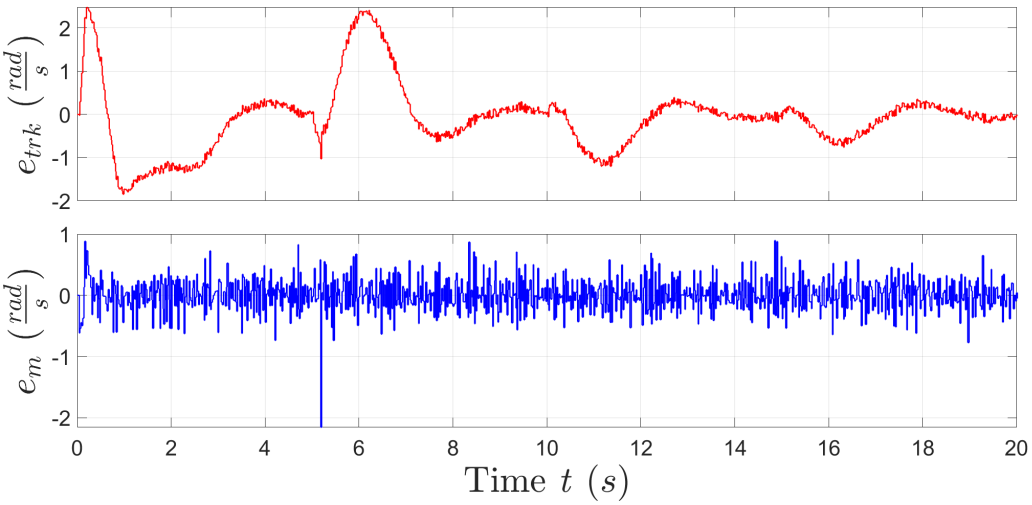


Figure 7.165: Tracking error e_{trk} and model error e_m

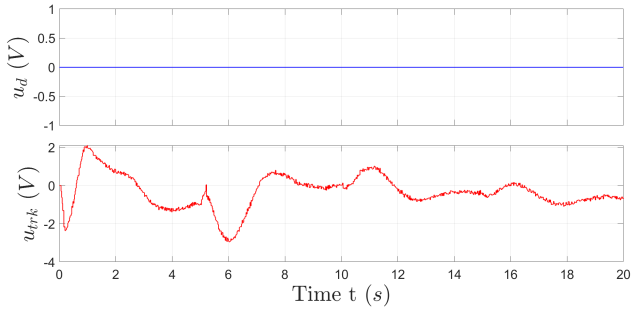


Figure 7.166: Control input parts u_d and u_{trk}

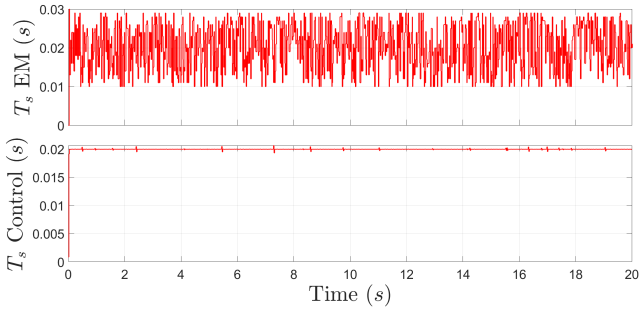


Figure 7.167: Measured Timestamp T

7.3.8 Load conditions - Result 1

Test settings: Disturbance rejection, Target speed $[8, 3, 5.5, 7]$ rad/s. Case of input saturation (because of low battery voltage supply).

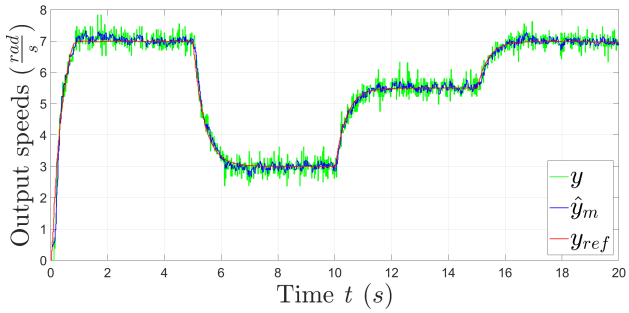


Figure 7.168: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

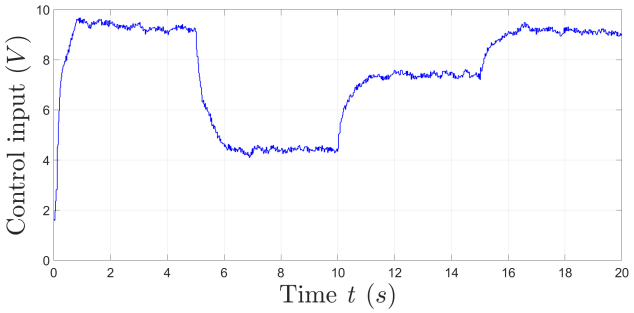


Figure 7.169: Control input voltage u

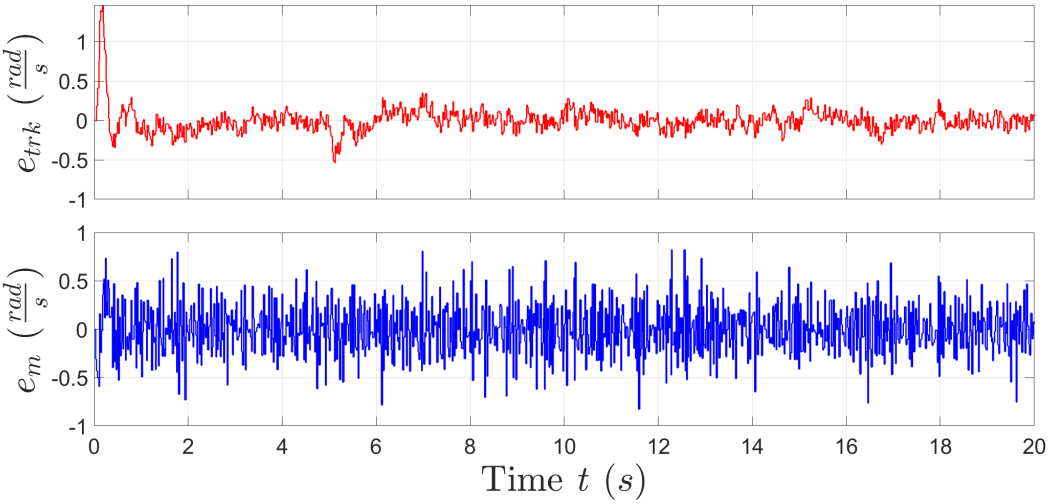


Figure 7.170: Tracking error e_{trk} and model error e_m

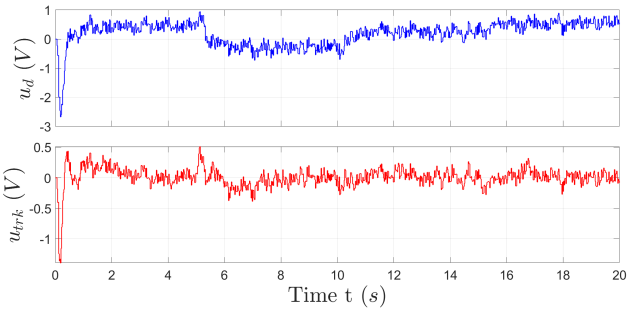


Figure 7.171: Control input parts u_d and u_{trk}

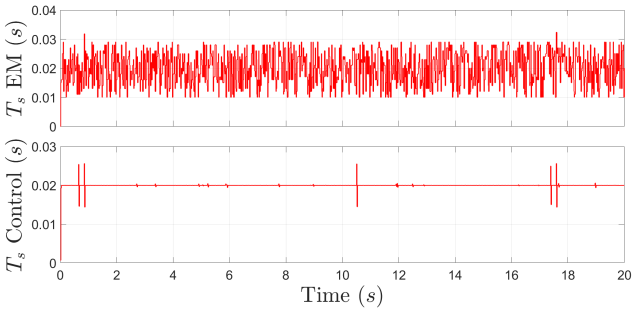


Figure 7.172: Measured Timestamp T

7.3.9 Load conditions - Result 2

Test settings: No disturbance rejection, Target speed $[8, 3, 5.5, 7]$ rad/s. Case of input saturation (because of low battery voltage supply).

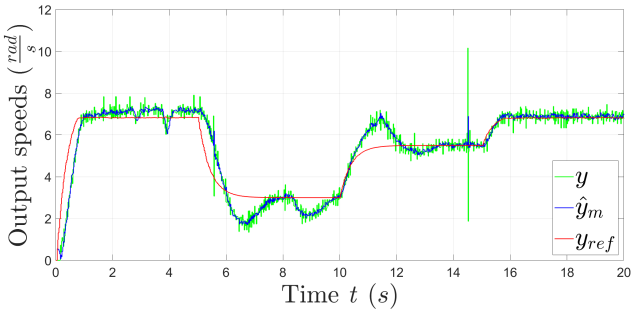


Figure 7.173: Output speeds $y_{ref}, \hat{y}_m$ and y not filtered

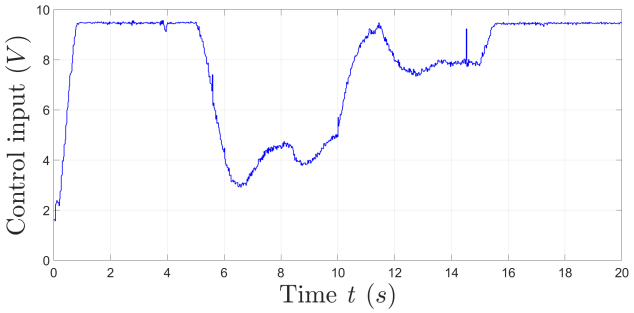


Figure 7.174: Control input voltage u

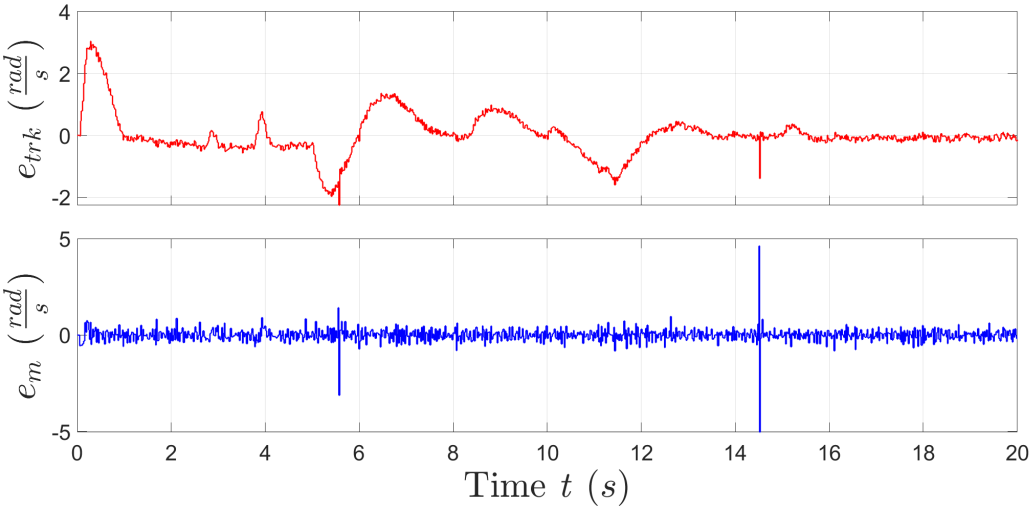


Figure 7.175: Tracking error e_{trk} and model error e_m

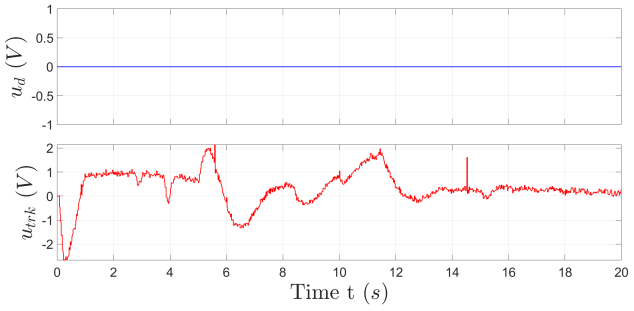


Figure 7.176: Control input parts u_d and u_{trk}

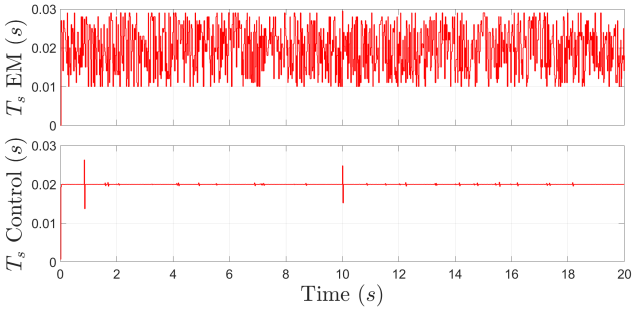


Figure 7.177: Measured Timestamp T

Chapter 8

Combined Left and Right DC motors EMC - RHIT results

RHIT tests are performed using both left and right motors EMC in combination, to see if the results obtained with motors EMC separated can be confirmed.

8.1 Main RHIT tests settings

The EMC versions used for both the motors are the ones giving the best results: v1.1 and v1.2 for the left motor (Section 5.1.4 and Section 5.1.5) and the only one found (v1.2) for the right motor (Section 5.1.5). Version 1.1 is considered with separated left motor EMC RHIT, however with tests using motors in combination is added the version 1.2, since v1.1 seems to be not entirely sufficient to obtain good tracking and model errors.
Common settings are (unless specified in each test):

- Left motor EMC version 1.2 (in some tests v1.1 to understand the differences), Right motor Version 1.2.

- CT eigenvalues used:

Eigenvalue type	CT eigenvalues	Related DT eigenvalues at $T = 0.02\,\mathrm{s}$
Control μ_K	$[-2.5647, -2.5647]$	$[0.95, 0.95]$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842, -14.3842]$	$[0.75, 0.75, 0.75]$

Table 8.1: CT eigenvalues DC motor EMC

  Control eigenvalues are changed in some test to see the effects of the designed model with and without disturbance rejection. When needed the new eigenvalues will be specified.

- Control and EM parts of EMC DC motor are separated and their codes run with different sampling times. Control part sampling time was maintained always fixed at $T_{ctrl} = 0.02\,\mathrm{s}$. Instead EM part is repeated every $T_{EM} = [0.01, 0.03]$ s, detailed informations in Section 4.3.1.

- At every step the plant measurements are taken first, and then the EM part begin. It is verified with the

previous RHIT that using this implementation order gives good result for output speed resolution.

- 2^{nd} order disturbance $\overline{w}$ is used.

- In all tests robot is placed in the floor, in load conditions, since it is important to understand its trajectory during motion.

For each set of plots are shown: the DC motor output speeds $(y, \hat{y}_m, y_{ref})$, input voltages u, Timestamp EM/Control T_{EM} and T_{ctrl}, tracking and model errors e_{trk}, e_m.
Since now the 2 motors work together in load conditions (robot is moving on the floor) it is important the whole robot position, speed and orientation. For this reason the trajectory followed by robot in Cartesian coordinates ξ, ν is computed. Accepting some approximations it can be used the simplified differential robot kinematic equations model explained in Equation (3.15).
Two main trajectories are considered, linear and circular, specific settings are written in correspondent test section. Concerning circular trajectory, since at the time of these tests no robot position and orientation control has been done yet, again kinematic properties of DD robot are exploited to find wheel speeds starting from a desired circular trajectory of known wheel mean radius r_m and total robot speed v, following Equation (3.16).
For each test is therefore added a figure with robot ξ, ν trajectory, in which the start and end points are highlighted, and the arrows are coloured depending on path completion (passing from blue near the start to red near the end).
Different tests are made with and without disturbance rejection, with different reference output values, both negative and positive, and also with different control eigenvalues and model parameters (left motor v1.1 and v1.2).

8.2 Summary on the results

During tests different parameters of EMC DC motor and CT control eigenvalues are used.
Right motor control model gives good model and tracking errors e_m and e_{trk} respectively, therefore the unique version found (Section 5.1.5) is sufficient.
Instead left motor control model have some problems to follow the reference speed (some oscillations around the reference are present), for this reason some tests using different versions are considered: comparing left motor EMC version 1.1 (Section 5.1.4) and 1.2 (Section 5.1.5), slightly better results are obtained with second version[3].
In one test control eigenvalues μ_K are changed from $[-2.5647, -2.5647]$ of Section 8.3.3 test to $[-5.2647, -5.2647]$ of Section 8.3.4 test (faster ones): the tracking error results are not improved (not perfect linear trajectory): this means that the selected control eigenvalues are fast enough for tracking the reference.
In linear trajectory tests, the robot sometimes didn't follow a perfect straight line: we need to remember that no position and orientation EMC have not been made yet: without this control, even if the EMC DC motor models work well, floor imperfections (bumps, slope etc.) can lead to trajectory errors.

[3]However we need to remember that the difference in model parameters are almost completely compensated by noise estimator, so if the tracking error is not good in theory is a matter of control eigenvalues (need to choose slower ones)

8.3 Linear trajectory

8.3.1 Disturbance rejection - Result 1

Specific test settings: EMC DC motor version 1.1. Targets $\tilde{\omega}_L = \tilde{\omega}_R = [6, 4]$ rad/s.

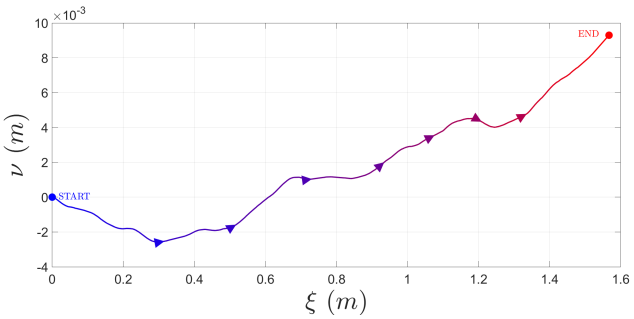


Figure 8.1: Robot position

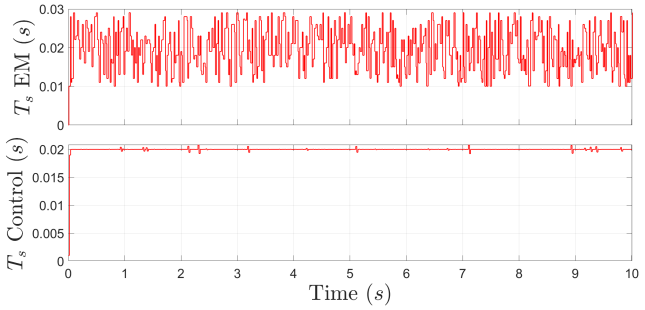


Figure 8.2: Measured Timestamp T - Left and Right motor

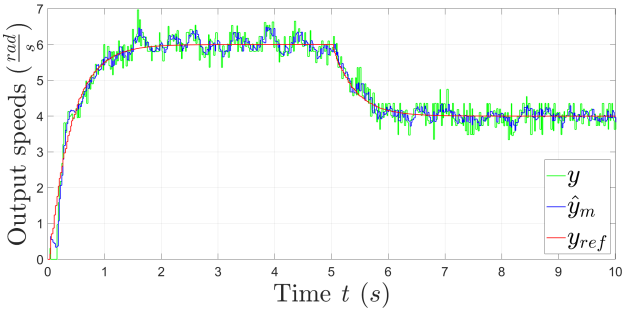


Figure 8.3: Output speeds reference y_{ref}, estimated $\hat{y}_m$ and measured y - Left motor

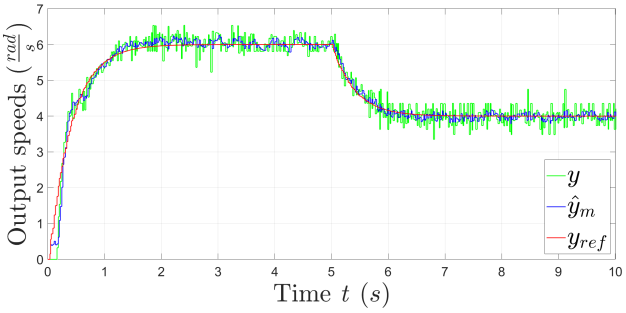


Figure 8.4: Output speeds reference y_{ref}, estimated $\hat{y}_m$ and measured y - Right motor

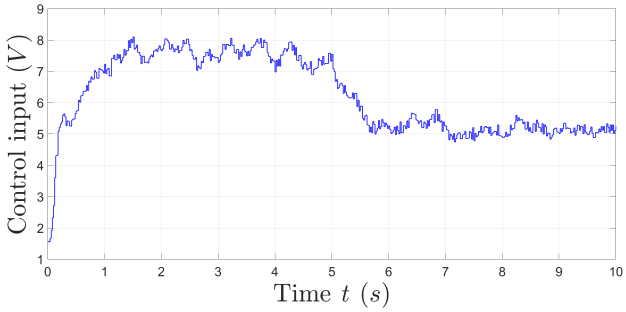


Figure 8.5: Control input voltage u - Left motor

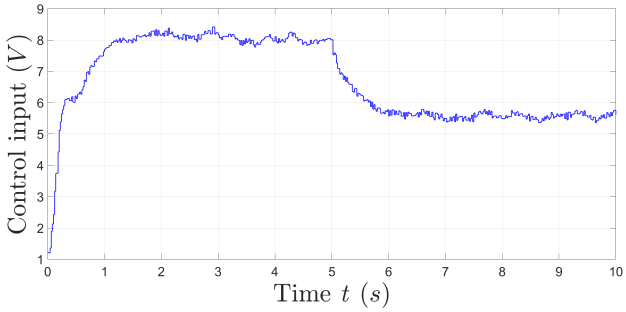


Figure 8.6: Control input voltage u - Right motor

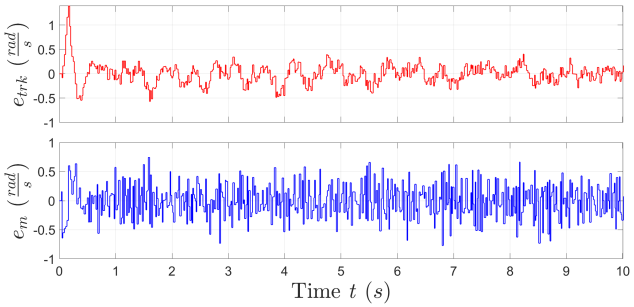


Figure 8.7: Tracking error e_{trk} and model error e_m - Left motor

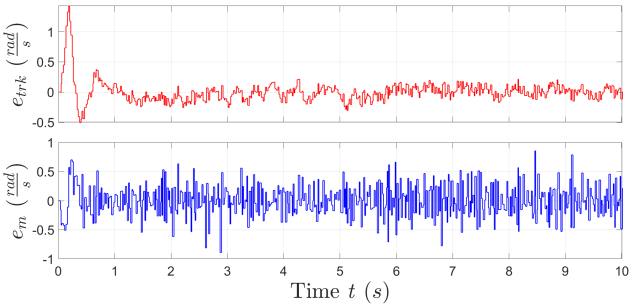


Figure 8.8: Tracking error e_{trk} and model error e_m - Right motor

8.3.2 Disturbance rejection - Result 2

Specific test settings: EMC DC motor version 1.1. Targets $\tilde{\omega}_L = \tilde{\omega}_R = [6, -4, 5]$ rad/s

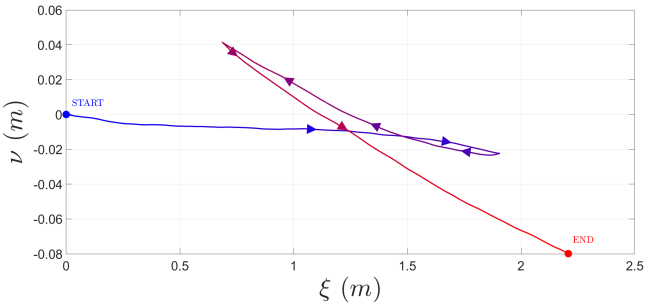


Figure 8.9: Robot position

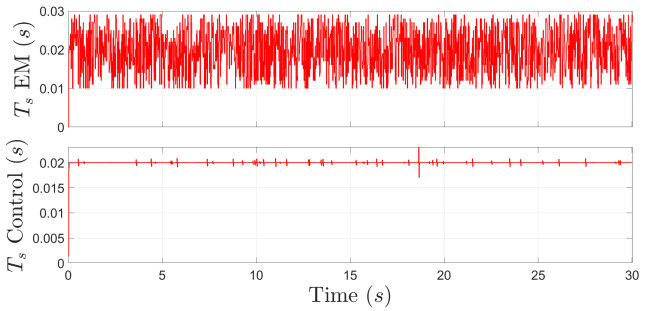


Figure 8.10: Measured Timestamp T - Left and Right motor

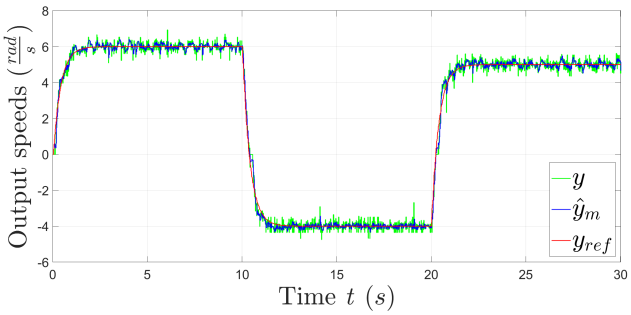


Figure 8.11: Output speeds reference y_{ref}, estimated $\hat{y}_m$ and measured y - Left motor

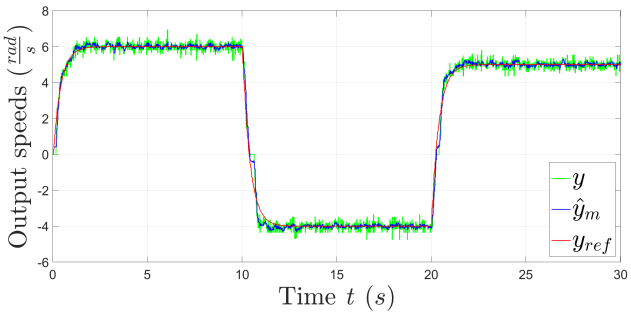


Figure 8.12: Output speeds reference y_{ref}, estimated $\hat{y}_m$ and measured y - Right motor

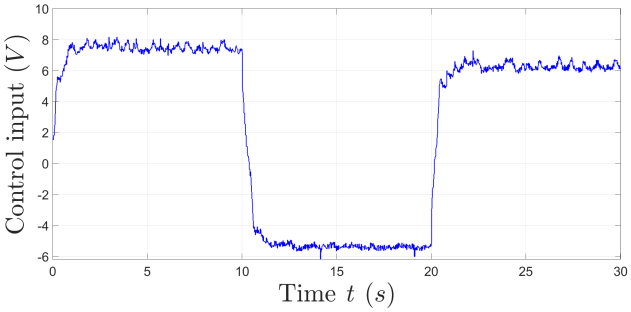


Figure 8.13: Control input voltage u - Left motor

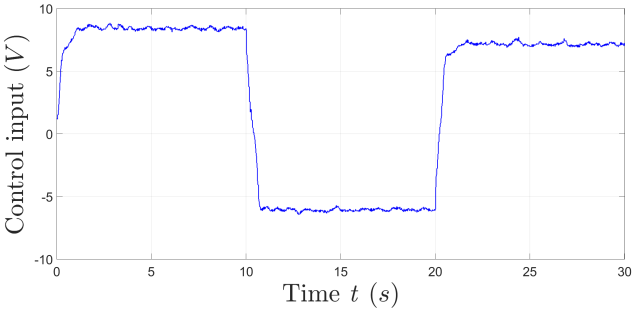


Figure 8.14: Control input voltage u - Right motor

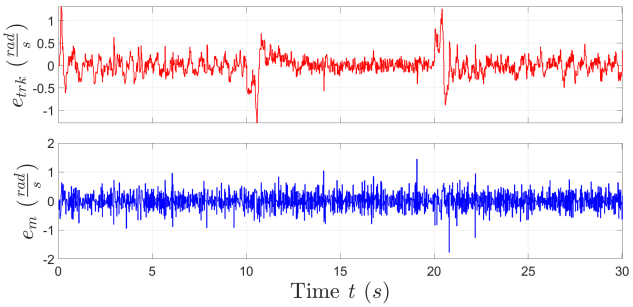


Figure 8.15: Tracking error e_{trk} and model error e_m - Left motor

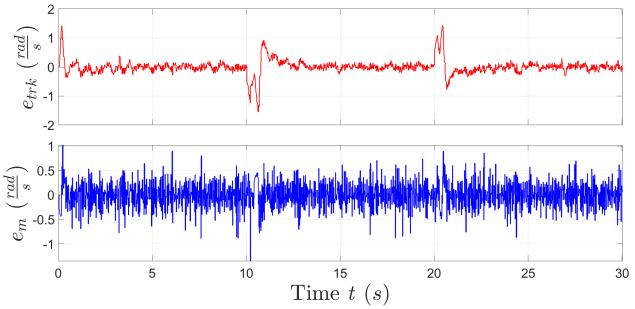


Figure 8.16: Tracking error e_{trk} and model error e_m - Right motor

8.3.3 Disturbance rejection - Result 3

Specific test settings: Targets $\tilde{\omega}_L = \tilde{\omega}_R = [6, -4, 5]$ rad/s

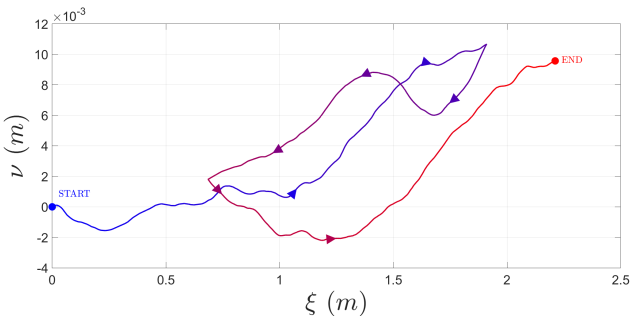


Figure 8.17: Robot position

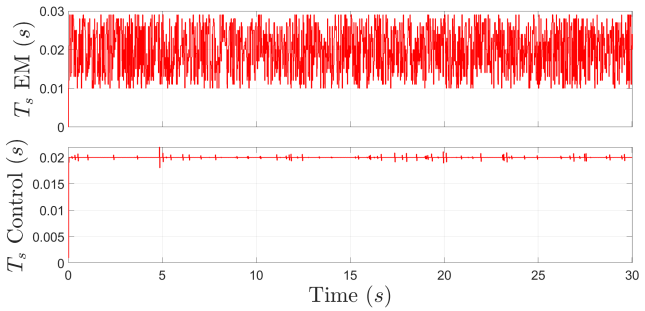


Figure 8.18: Measured Timestamp T - Left and Right motor

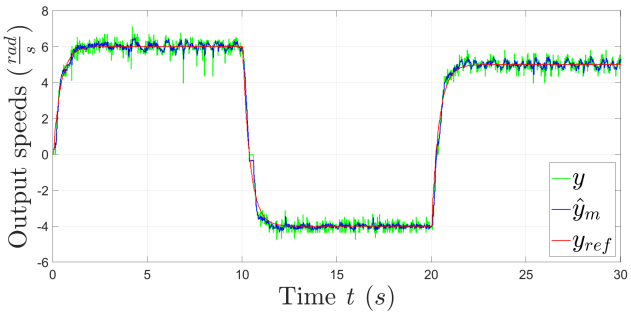


Figure 8.19: Output speeds reference y_{ref}, estimated $\hat{y}_m$ and measured y - Left motor

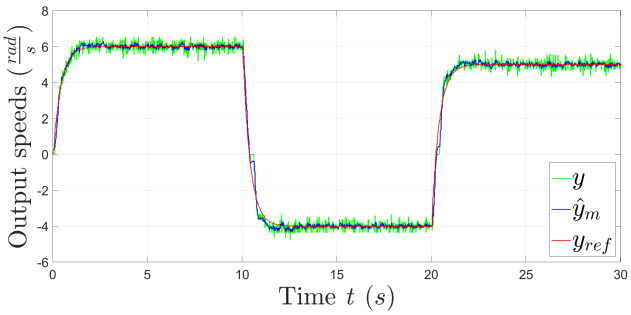


Figure 8.20: Output speeds reference y_{ref}, estimated $\hat{y}_m$ and measured y - Right motor

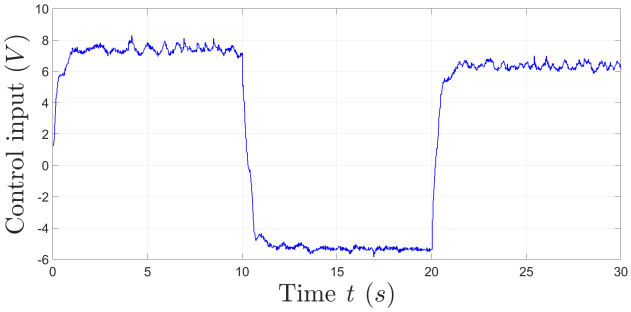


Figure 8.21: Control input voltage u - Left motor

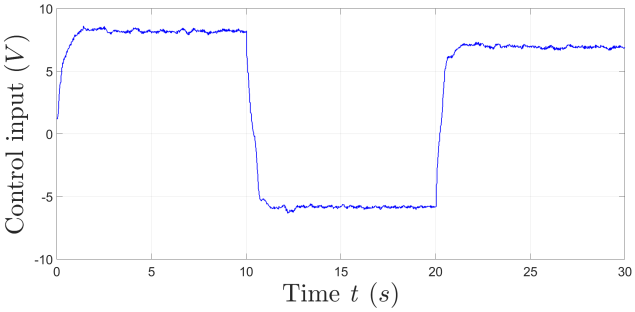


Figure 8.22: Control input voltage u - Right motor

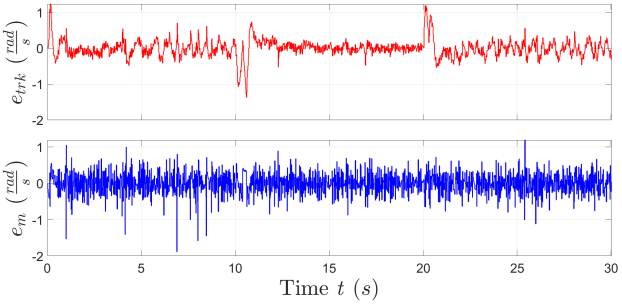


Figure 8.23: Tracking error e_{trk} and model error e_m - Left motor

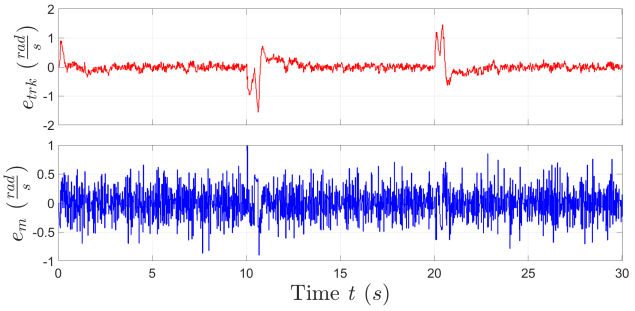


Figure 8.24: Tracking error e_{trk} and model error e_m - Right motor

8.3.4 Disturbance rejection - Result 4

Specific test settings: CT control eigenvalues $\mu_K = [-5.2647, -5.2647]$. Using faster control eigenvalues the drift of robot during its motion is present anyway (i.e. the trajectory is not perfectly linear). Targets $\tilde{\omega}_L = \tilde{\omega}_R = 6\,\mathrm{rad/s}$

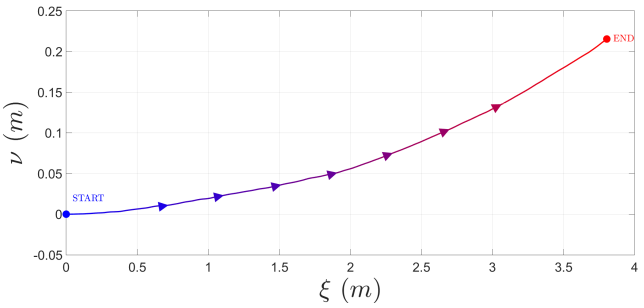


Figure 8.25: Robot position

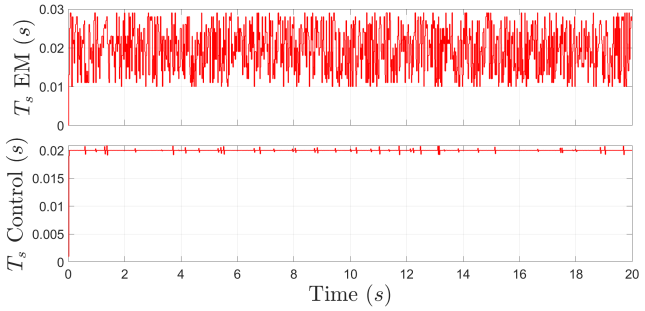


Figure 8.26: Measured Timestamp T - Left and Right motor

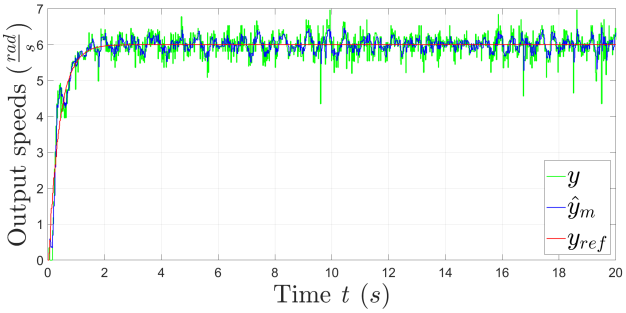


Figure 8.27: Output speeds reference y_{ref}, estimated $\hat{y}_m$ and measured y - Left motor

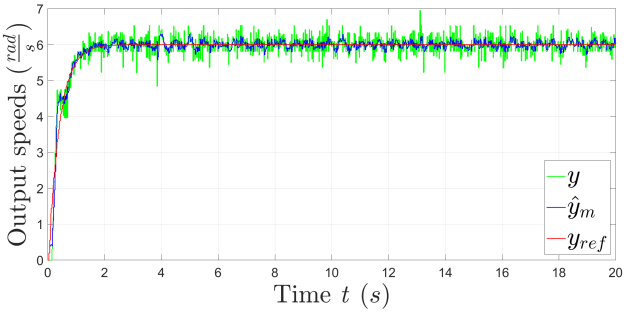


Figure 8.28: Output speeds reference y_{ref}, estimated $\hat{y}_m$ and measured y - Right motor

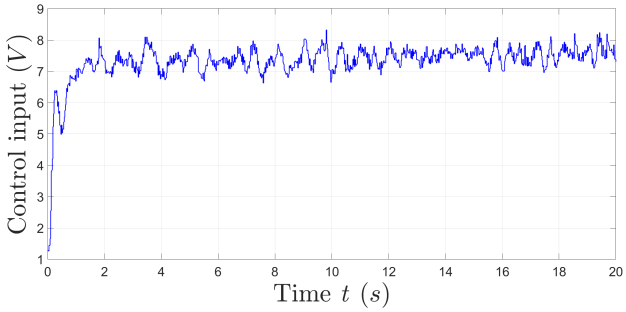


Figure 8.29: Control input voltage u - Left motor

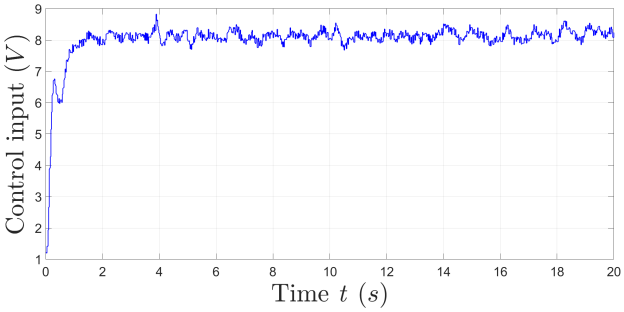


Figure 8.30: Control input voltage u - Right motor

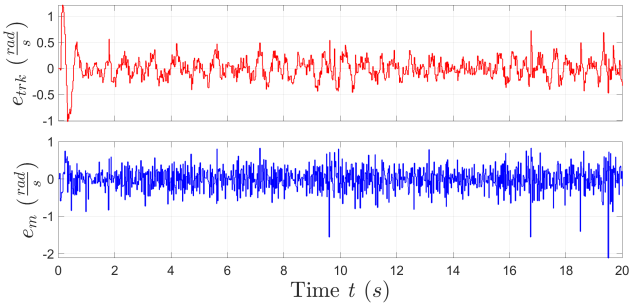


Figure 8.31: Tracking error e_{trk} and model error e_m - Left motor

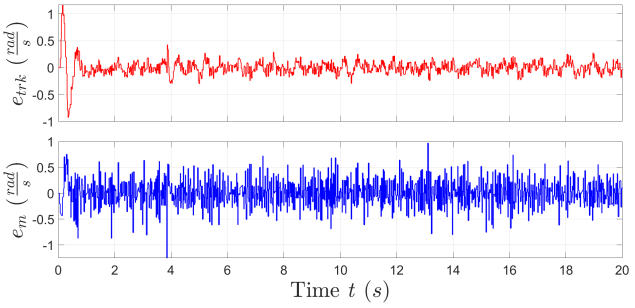


Figure 8.32: Tracking error e_{trk} and model error e_m - Right motor

8.3.5 No disturbance rejection - Result 1

Specific test settings: Targets $\tilde{\omega}_L = \tilde{\omega}_R = [6, -4, 5]$ rad/s

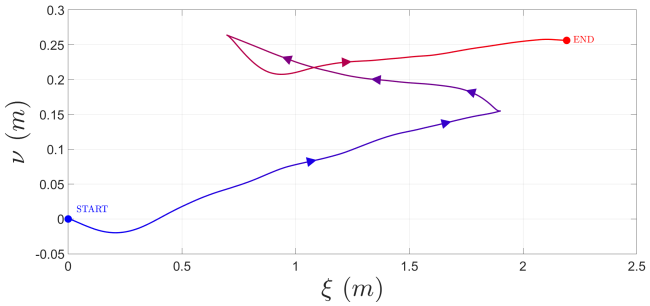


Figure 8.33: Robot position

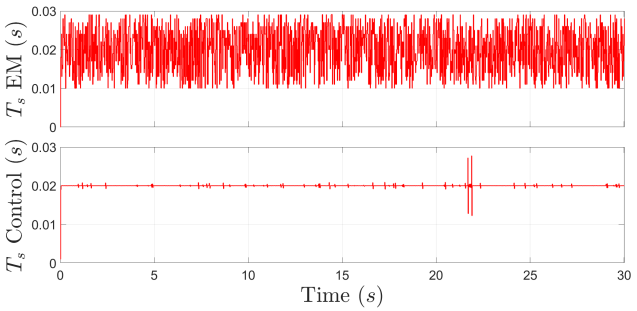


Figure 8.34: Measured Timestamp T - Left and Right motor

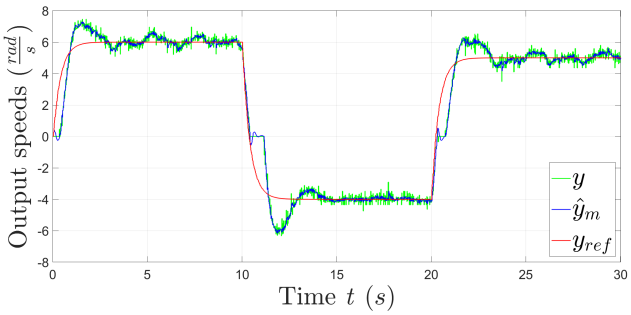


Figure 8.35: Output speeds reference y_{ref}, estimated $\hat{y}_m$ and measured y - Left motor

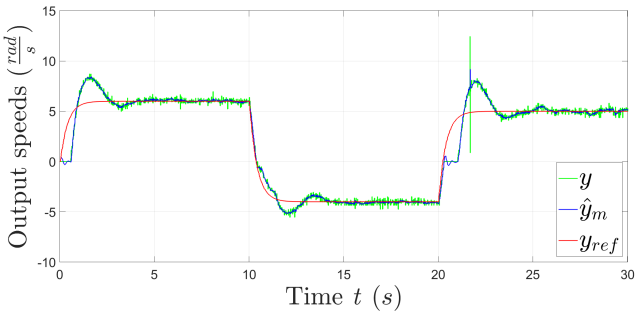


Figure 8.36: Output speeds reference y_{ref}, estimated $\hat{y}_m$ and measured y - Right motor

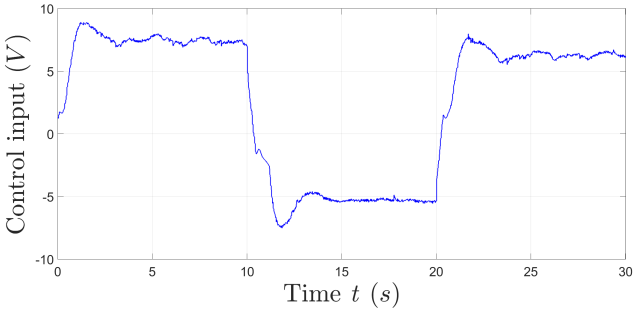


Figure 8.37: Control input voltage u - Left motor

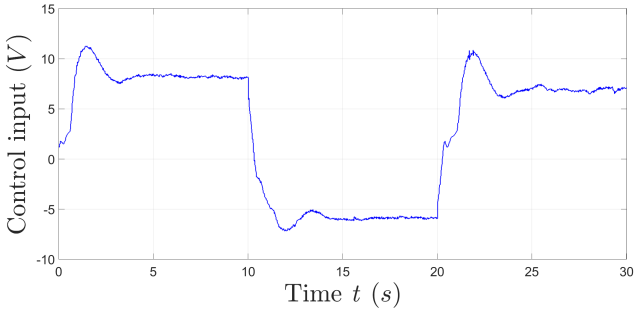


Figure 8.38: Control input voltage u - Right motor

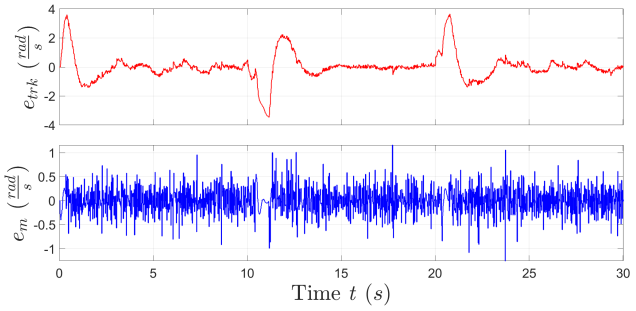


Figure 8.39: Tracking error e_{trk} and model error e_m - Left motor

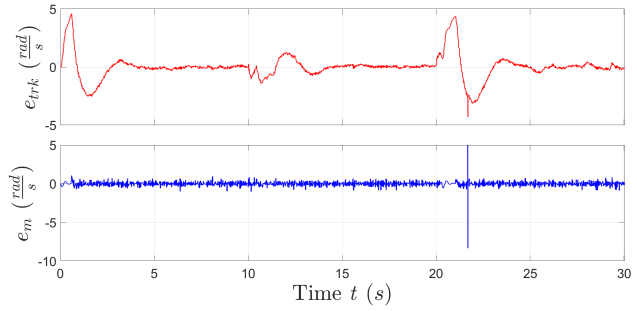


Figure 8.40: Tracking error e_{trk} and model error e_m - Right motor

8.3.6 No disturbance rejection - Result 2

Specific test settings: Targets $\tilde{\omega}_L = \tilde{\omega}_R = 6\,\mathrm{rad/s}$.

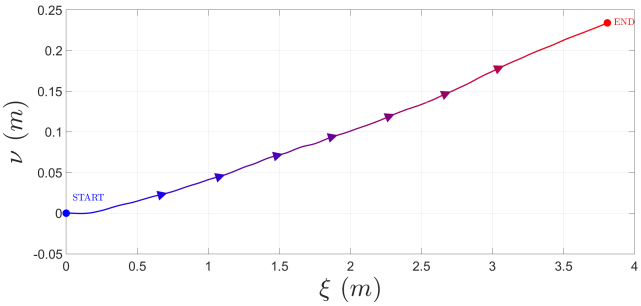


Figure 8.41: Robot position

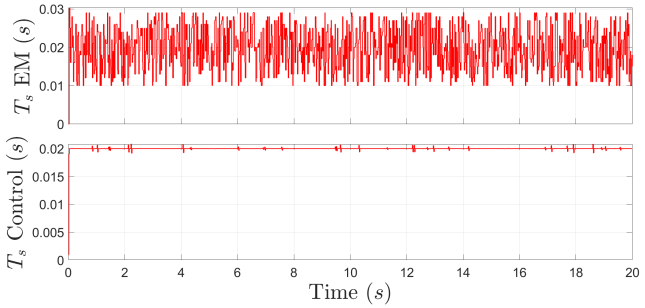


Figure 8.42: Measured Timestamp T - Left and Right motor

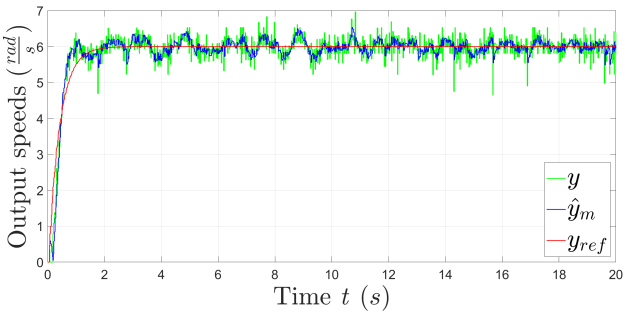


Figure 8.43: Output speeds reference y_{ref}, estimated $\hat{y}_m$ and measured y - Left motor

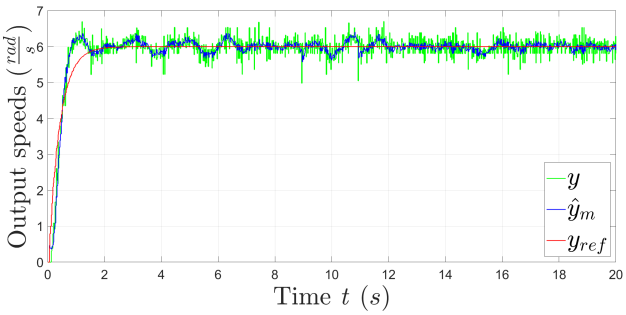


Figure 8.44: Output speeds reference y_{ref}, estimated $\hat{y}_m$ and measured y - Right motor

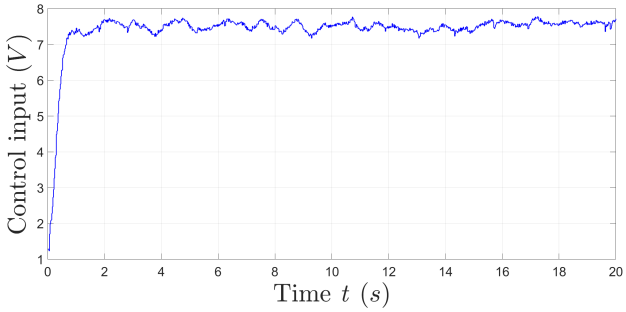


Figure 8.45: Control input voltage u - Left motor

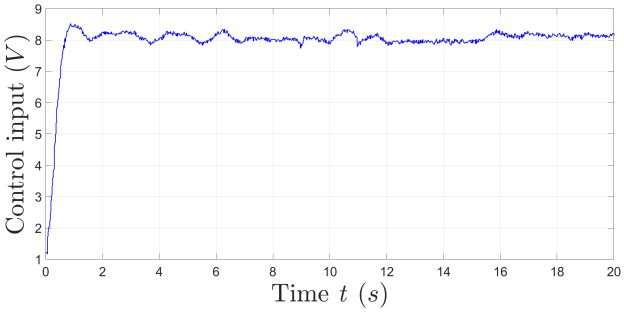


Figure 8.46: Control input voltage u - Right motor

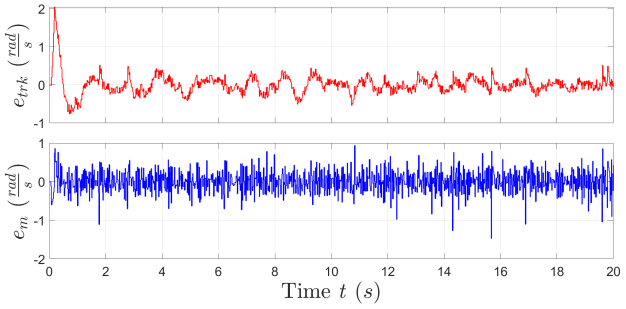


Figure 8.47: Tracking error e_{trk} and model error e_m - Left motor

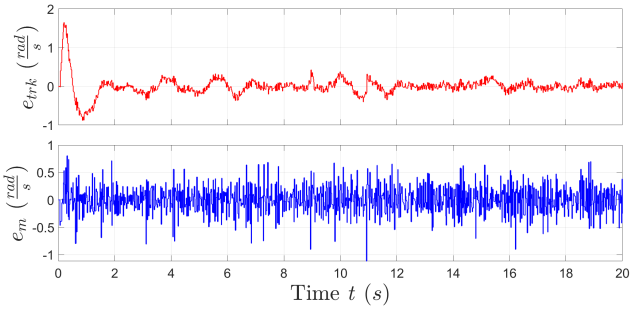


Figure 8.48: Tracking error e_{trk} and model error e_m - Right motor

8.4 Circular trajectory

8.4.1 Disturbance rejection - Result 1

Specific test settings: The desired radius and mean linear velocity of robot used are $r_m = 0.3\,\mathrm{m}$ and $v_\xi = 4\,\frac{\mathrm{m}}{\mathrm{s}}$.

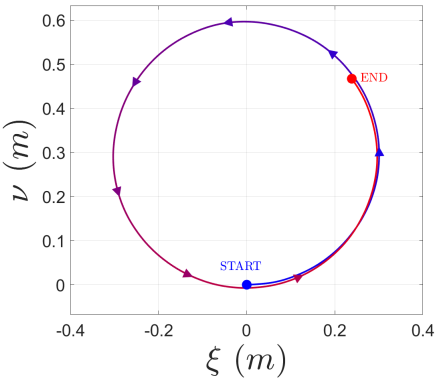


Figure 8.49: Robot position

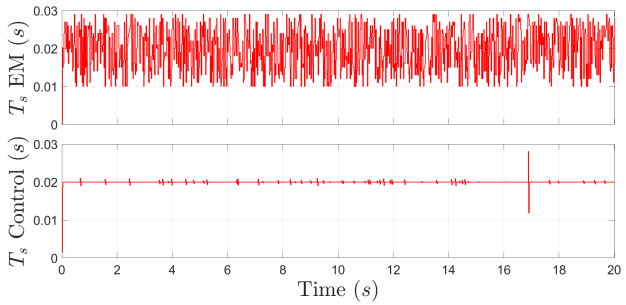


Figure 8.50: Measured Timestamp T - Left and Right motor

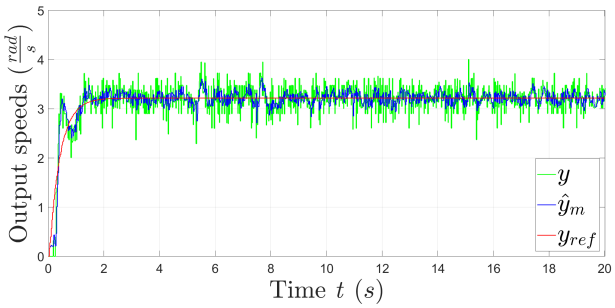


Figure 8.51: Output speeds reference y_{ref}, estimated $\hat{y}_m$ and measured y - Left motor

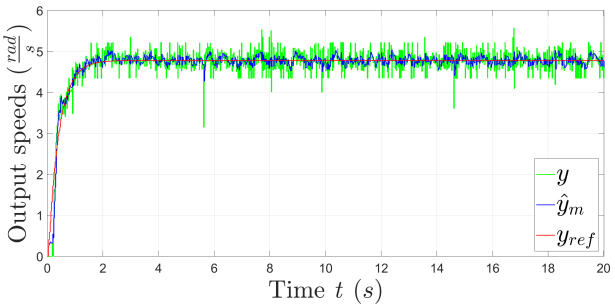


Figure 8.52: Output speeds reference y_{ref}, estimated $\hat{y}_m$ and measured y - Right motor

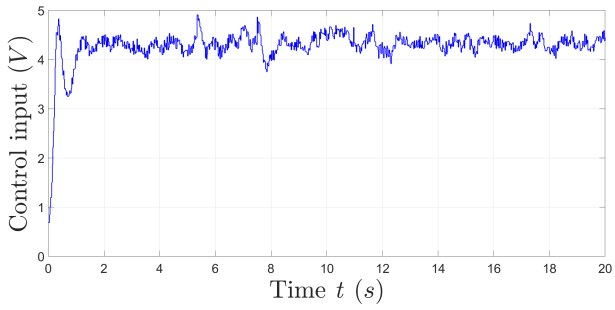


Figure 8.53: Control input voltage u - Left motor

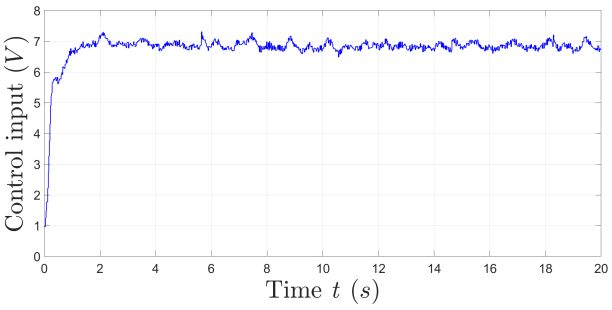


Figure 8.54: Control input voltage u - Right motor

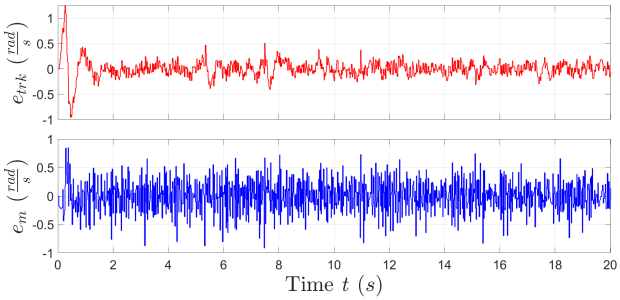


Figure 8.55: Tracking error e_{trk} and model error e_m - Left motor

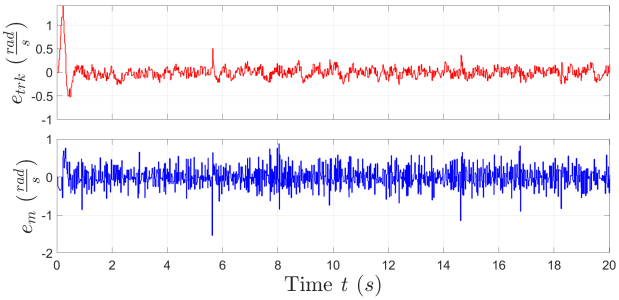


Figure 8.56: Tracking error e_{trk} and model error e_m - Right motor

8.4.2 Disturbance rejection - Result 2

Specific test settings: The desired radius and mean linear velocity of robot used are $r_m = 0.3\,\mathrm{m}$ and $v_\xi = 4\,\frac{\mathrm{m}}{\mathrm{s}}$.

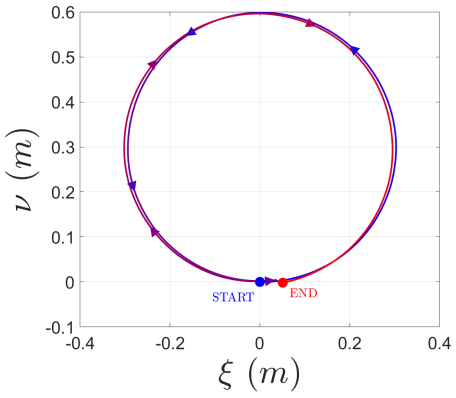


Figure 8.57: Robot position

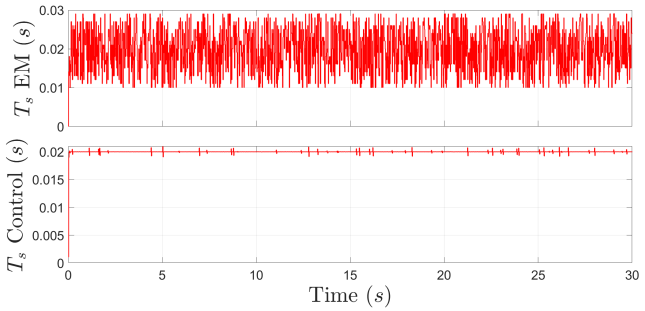


Figure 8.58: Measured Timestamp T - Left and Right motor

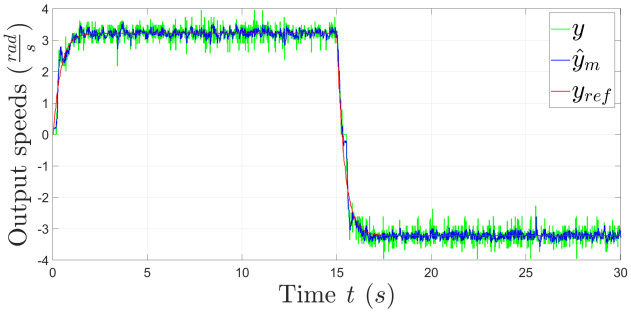


Figure 8.59: Output speeds reference y_{ref}, estimated $\hat{y}_m$ and measured y - Left motor

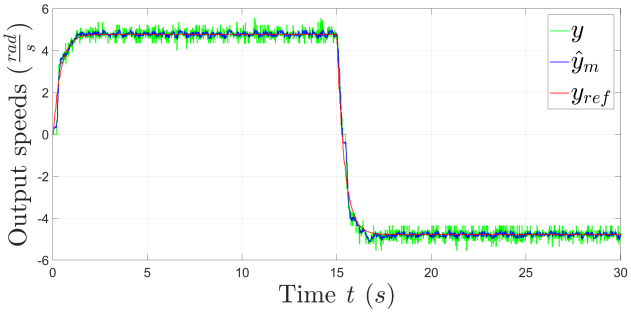


Figure 8.60: Output speeds reference y_{ref}, estimated $\hat{y}_m$ and measured y - Right motor

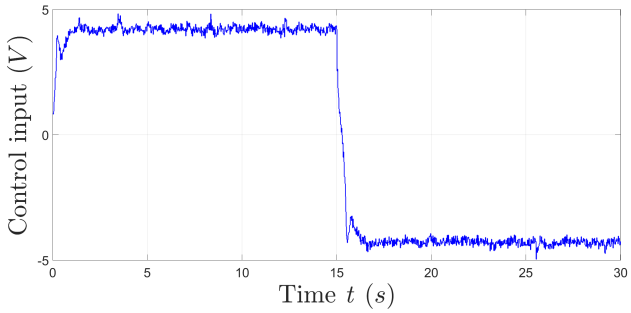


Figure 8.61: Control input voltage u - Left motor

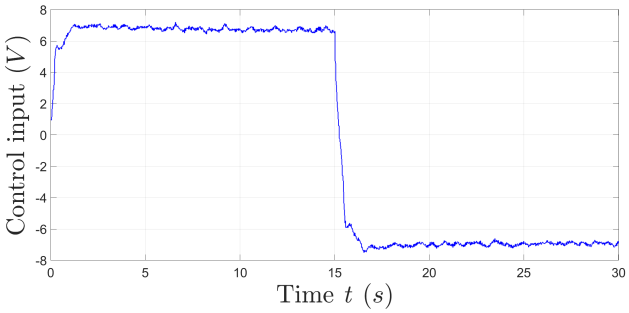


Figure 8.62: Control input voltage u - Right motor

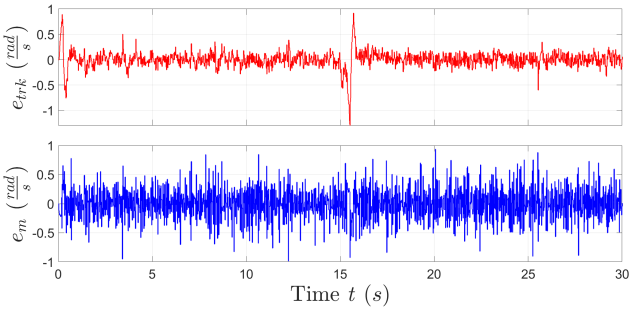


Figure 8.63: Tracking error e_{trk} and model error e_m - Left motor

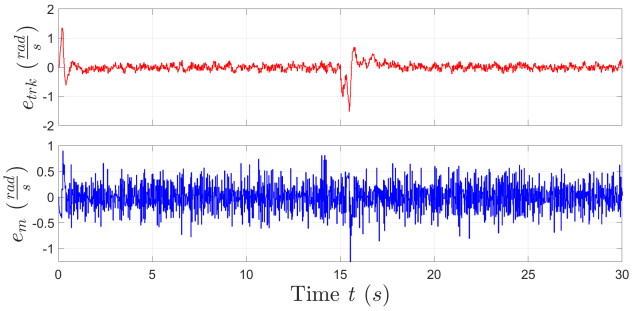


Figure 8.64: Tracking error e_{trk} and model error e_m - Right motor

Chapter 9

Inertial Measurement Unit (IMU) measurements with Two motors EMC

EMC until now is verified using both 2 motors in load conditions (robot placed on the ground). Objective in the next chapters will be to built an EMC to control robot orientation/longitudinal position. Orientation of the robot is defined mainly by angular speed ω_z and yaw angle θ_z around z-axis, and longitudinal position by ξ coordinate. To obtain these measurements a IMU device is selected. Hence in this chapter, using already controlled DC motors with EMC, IMU data are taken and post-processed to understand if measurements are reliable to be considered for orientation/position control.

9.1 Main measurements settings

9.1.1 IMU

To obtain data from IMU, a library called RTIMULib2 in C++ code is used (freely downloadable from GitHub repository, [24]). The library contains already prepared functions to get raw accelerometer, magnetometer and gyroscope data.

Before starting using IMU, calibration of all sensors is necessary: common procedure consists in computing bias/offsets of all data and subtract them from actual values of acceleration, angular speed and magnetic field at runtime. For example acceleration bias B_a can be computed by taking minimum and maximum values for all coordinate components ξ, ν, z, make the mean values and subtract them from actual accelerations, at every measurement step:

$$\begin{cases} B_{a,\xi} = \dfrac{a_{\xi,min} + a_{\xi,max}}{2} \\ B_{a,\nu} = \dfrac{a_{\nu,min} + a_{\nu,max}}{2} \\ B_{a,z} = \dfrac{a_{z,min} + a_{z,max}}{2} \end{cases} \Rightarrow \begin{cases} a_\xi^{corr} = a_\xi - B_{a,\xi} \\ a_\nu^{corr} = a_\nu - B_{a,\nu} \\ a_z^{corr} = a_z - B_{az} \end{cases}$$

A more precise procedure to remove bias is to take many measurements putting IMU in different positions and compute the bias and sensitivity errors as explained in [10]. Similar procedure can be done for angular speed and magnetic field.

A tool for calibration is already present in the RTIMULib2 C++ library to make the calibration: it saves all offsets/biases in a setting file (*RTIMULib.ini*) that can be used in every C++ code, uploading it with a function. The calibration for accelerometer/gyroscope is done moving IMU around space with almost all possible orientations, and the tool saves the minimum and maximum values for each Cartesian direction. Instead for magnetometer first

an *ellipsoid* of values is saved, moving IMU in all possible directions: from this ellipsoid magnetic field offsets are computed and then removed.
In this file also other settings are present: sensors full scale range, sample rates (for fusion algorithms). Specifically:

Sensor	Setting name	Value	Notes
Gyroscope	Sample rate	$95\,\mathrm{Hz}$	
	Full scale range	$\pm 250\,^{\circ}/\mathrm{s} \approx \pm 4.36\,\frac{\mathrm{rad}}{\mathrm{s}}$	
Accelerometer	Sample rate	$50\,\mathrm{Hz}$	
	Full scale range	$\pm 2\,\mathrm{g}$	Minimum value possible (others are $\pm 4, \pm 8, \pm 16\,\mathrm{g}$)
Magnetometer	Sample rate	$30\,\mathrm{Hz}$	
	Full scale range	$\pm 130\,\mu\mathrm{T}$	

Table 9.1: IMU sensors settings

There is also a sensor fusion algorithm already implemented to compute orientation (roll, pitch and yaw Euler angles) based on Kalman filter or RTQF estimation methods. Obviously orientation results are sensitive to sensor measurement errors, for this reason good calibration is fundamental.
Magnetometer is the most difficult to be calibrated, since the magnetic field changes a lot depending on its location in robot platform: this depends on other magnetic fields generated not only by DC motors but also by embedded HW board.
Considering this problem, 2 possibilities for fusion algorithm can be used:

- Using all 3 sensors data inside fusion algorithm.
  To calibrate the magnetometer, it was fixed with adhesive in a precise position of robot platform and then calibrated with RTIMULib2 tool. In this way, the motors/HW magnetic fields don't interfere since both robot and magnetometer are in the same reference frame.
  This method leads to absolute orientation, i.e. depending directly on earth magnetic field.

- Use only accelerometer and gyroscope for sensor fusion. In this case roll, pitch and yaw angles start from 0 at every IMU reset.

IMU results considering both the 2 conditions are shown, to understand which is the best solution.
For accelerometer, library contains also a C++ function to compute linear accelerations (also called acceleration residuals), which are acceleration values without the effect of gravity. The algorithm multiply total gravity (Euclidean 2-norm of all acceleration components) with orientation matrix and quaternion pose, and subtract it from actual acceleration.

9.1.2 DC Motor EMC

EMC versions 1.2 for the left motor and the only one found for the right motor are used for motor parameters. Common EMC motor settings are:

- Continuous eigenvalues used:

Eigenvalue type	CT eigenvalues	Related DT eigenvalues at $T = 0.02\,\mathrm{s}$
Reference μ_R	-2.5647	0.95
Noise estimator μ_N	$[-14.3842, -14.3842, -14.3842]$	$[0.75, 0.75, 0.75]$
Control μ_K	$[-2.5647, -2.5647]$	$[0.95, 0.95]$

Table 9.2: CT eigenvalues DC motor EMC - 2 motors load conditions

- Control and EM parts of EMC DC motor are separated and their codes run with different sampling times. Control part one is maintained always fixed at $T_{ctrl} = 0.02\,\mathrm{s}$. Instead EM part is repeated every $T_{EM} = [0.01, 0.03]$ s.

- At every step the plant measurements are taken first, and then the EM part begin.

- 2^{nd} order disturbance $\overline{w}$ is used.

- Robot placed in floor, load conditions.

9.1.3 Kinematic model

In order to verify position and orientation of the robot recovered by IMU in each test, they are computed using the simplified differential robot kinematic model in Equation (3.15). If robot in RHIT tests follows a circular trajectory with mean radius r_m and total robot speed v, the wheel speeds are computed at runtime with Equation (3.16).

9.2 Measurement plots

- 2 EMC DC motors figures:
  - Measured, estimated and reference output speed y, $\hat{y}_m$ and y_{ref}, in $\mathrm{rad/s}$.

- Kinematic equation figures:
  - Robot trajectory, in which the start and end points are highlighted, and trajectories and arrows are coloured depending on path completion (passing from blue near the start to red near the end).
    2 main trajectories are considered, linear and circular.
    For linear trajectories reference wheel speeds $\tilde{\omega}_L = \tilde{\omega}_R$ are: $0\,\mathrm{rad/s}$ for $[0, 5]$ s, $6\,\mathrm{rad/s}$ for $[5, 15]$ s and finally $4\,\mathrm{rad/s}$ for $[15, 20]$ s. Robot reaches approximately a longitudinal position of $\xi \approx 2.5\,\mathrm{m}$.
    For circular trajectory the desired radius and mean linear velocity of robot used are $r_m = 0.3\,\mathrm{m}$ and $v = 4\,\mathrm{m/s}$. From these values angular speed can be computed following Equation (3.16), $\omega_z = \dot{\theta}_z \approx 0.43\,\mathrm{rad/s}$. This is the value expected by gyroscope measurements. Instead reference wheel speeds are $\tilde{\omega}_L \approx 3.2\,\mathrm{rad/s}$ and $\tilde{\omega}_R = 4.7\,\mathrm{rad/s}$.
  - θ_z $(^\circ)$ and ω_z $(\mathrm{rad/s})$ around z axis which is the most important in planar motion.

- IMU measurements and estimation figures:
  - Orientation: roll, pitch, yaw angles $(^\circ)$, obtained using Kalman filter sensor fusion.

- Angular speed around all 3 axis $\omega_\xi, \omega_\nu, \omega_z$ $(\mathrm{rad/s})$.

- Magnetic field in all 3 axis $(\mu\mathrm{T})$.

- Acceleration (with gravity g component), split in 3 axis components a_ξ, a_ν, a_z (g).

- Linear acceleration (without gravity g component), split in 3 axis components $a_{l,\xi}, a_{l,\nu}, a_{l,z}$ (g).

For accelerations, linear accelerations and angular speeds plots also filtered signals are shown, to better understand their variations (since the sensor noise is reduced). Moving average filter of 15 sample delay window is used like in left DC motor RHIT, Equation (7.1).

9.2.1 Summary on the results

The best orientation fusion algorithm seems to be the one without magnetometer, as it gives better pose results (high output resolution, lower noise). However we will see in Chapter 12 making orientation EMC RHIT that the absence of magnetometer leads to pose measurement drifts, especially when robot angular speed is very low.
Since we have planar motion, only yaw angle θ_z orientation component is important, either with linear and circular trajectory the values are coherent with real conditions, with errors around some degrees.
Also gyroscope seems to give coherent values: for circular trajectories $\omega_z \approx 0.43\,\mathrm{rad/s}$, as expected by kinematic equations computations.
However, since accelerometer is a very sensible to noise sensor, it is impossible to understand if the acceleration and linear acceleration values are good or not, some manipulations on data are needed to clean the noise (which are not treated in this thesis work). At least, when robot is still, there is very few bias error ($< 0.1\,\mathrm{g}$), meaning that sensor calibration is done correctly.

9.2.2 No magnetometer data in sensor fusion, Linear trajectory - Result 1

Trajectory estimated with kinematic equations:

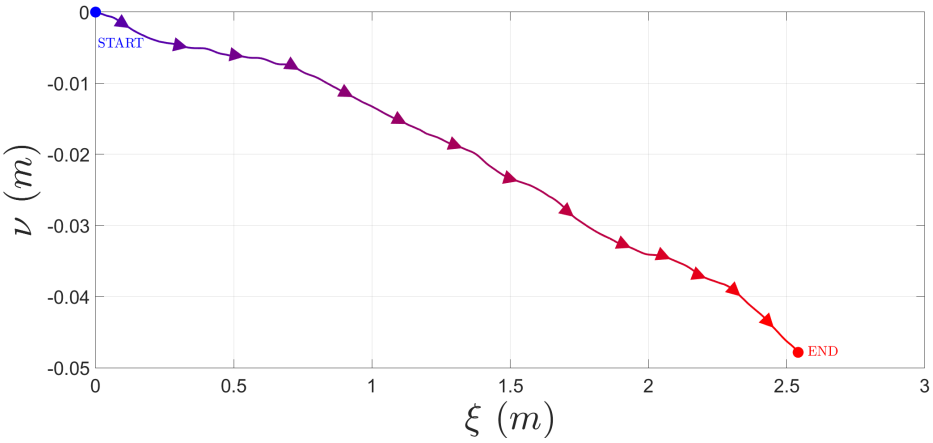


Figure 9.1: Robot position

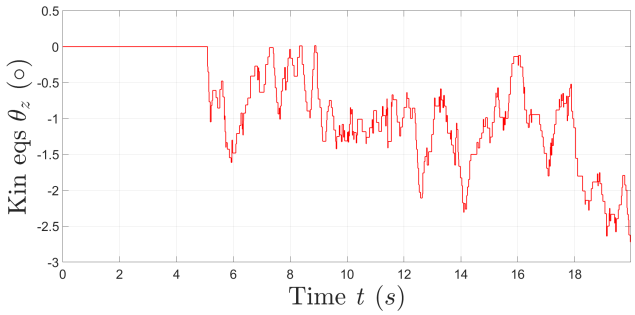


Figure 9.2: Robot orientation θ_z

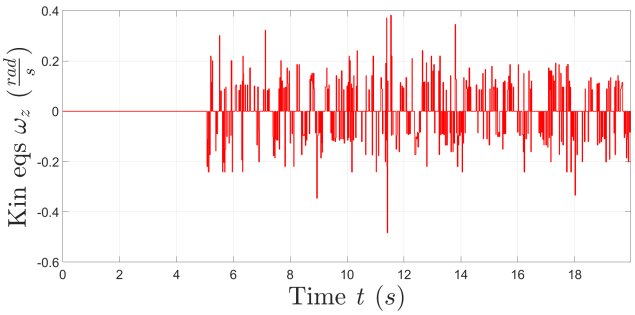


Figure 9.3: Angular speed ω_z

IMU measurements and estimations:

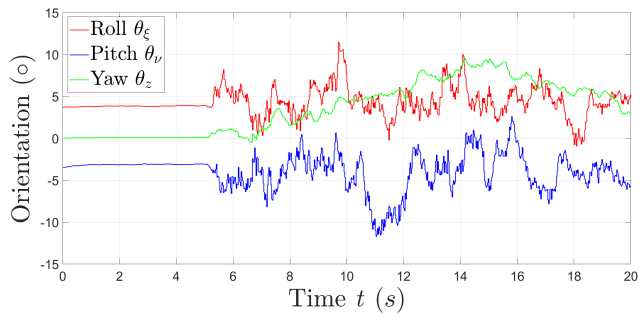


Figure 9.4: Robot orientation - IMU estimation using Kalman filter sensor fusion

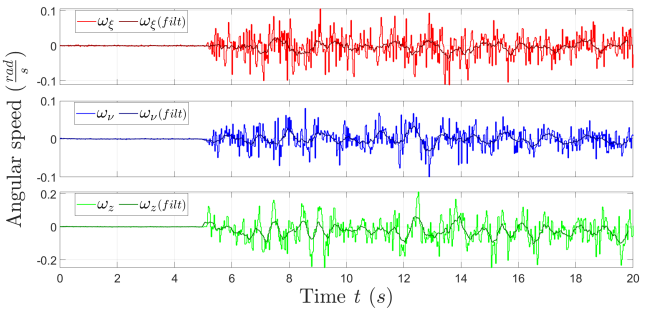


Figure 9.5: Robot angular acceleration - IMU measurements raw and filtered (moving average - 15 samples window)

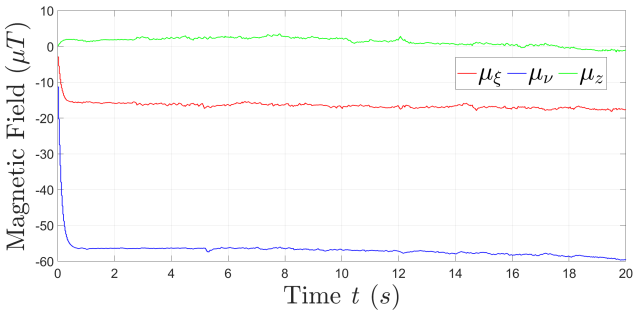


Figure 9.6: Magnetic field - IMU raw measurements

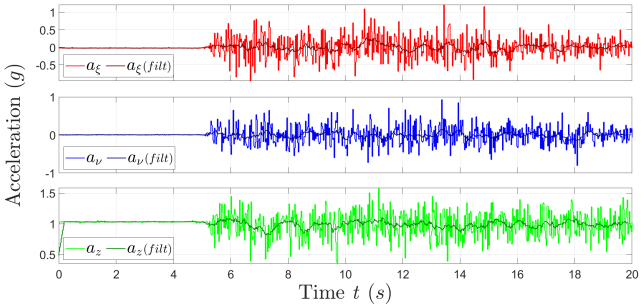


Figure 9.7: Robot Acceleration - IMU measurements raw and filtered (moving average - 15 samples window)

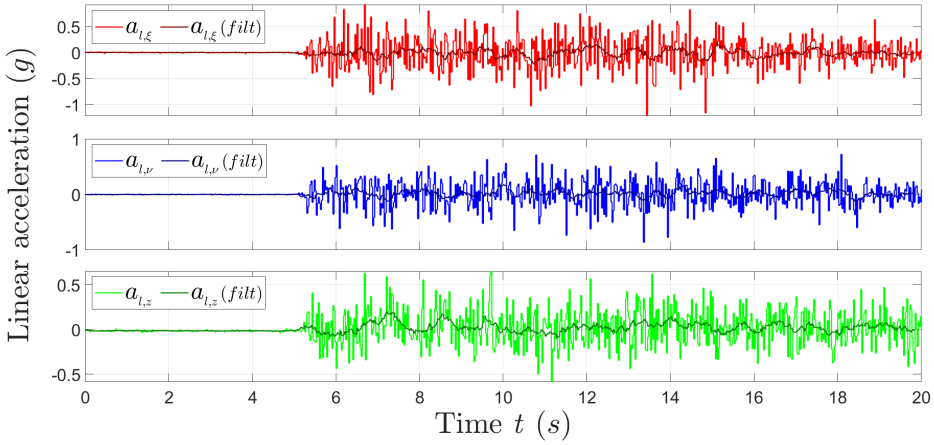


Figure 9.8: Robot linear acceleration - IMU estimation using orientation matrix, raw and filtered (moving average - 15 samples window)

Left and Right DC motors EMC output speeds:

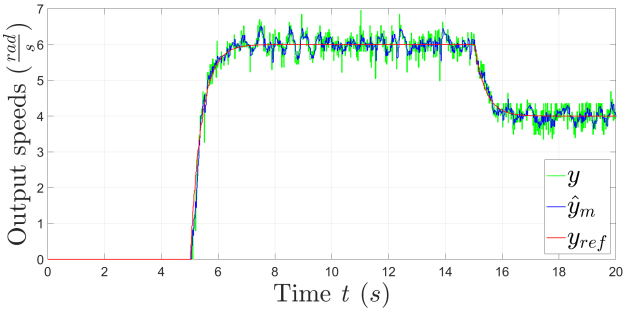


Figure 9.9: Output speeds reference y_{ref}, estimated $\hat{y}_m$ and measured y - Left motor

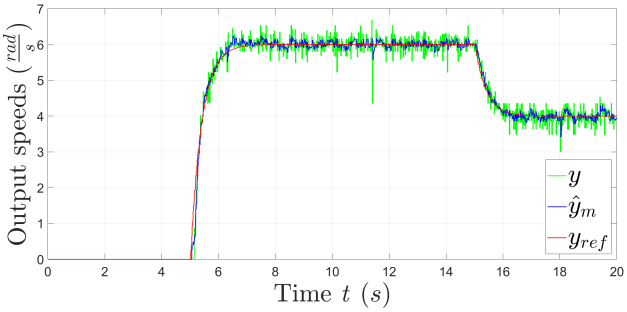


Figure 9.10: Output speeds reference y_{ref}, estimated $\hat{y}_m$ and measured y - Right motor

9.2.3 No magnetometer data in sensor fusion, Circular trajectory - Result 1

Trajectory estimated with kinematic equations:

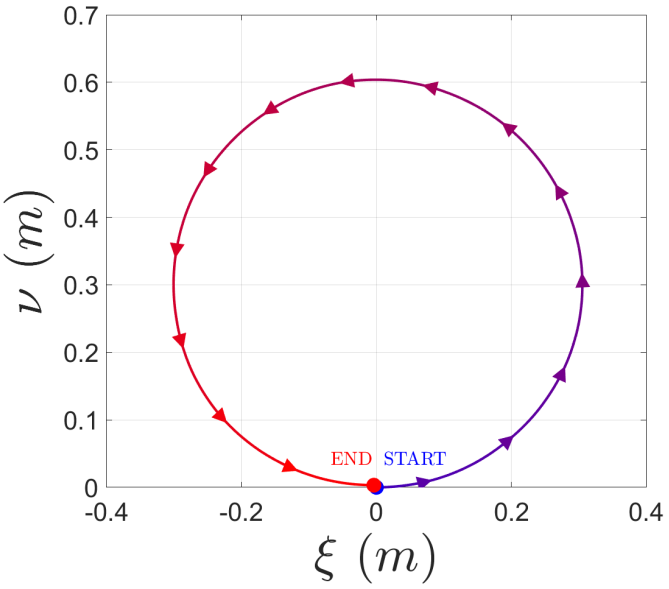


Figure 9.11: Robot position

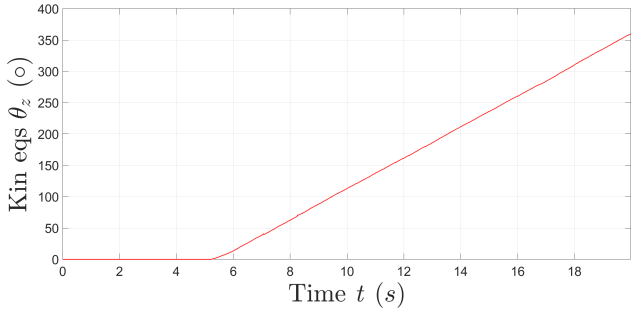


Figure 9.12: Robot orientation θ_z

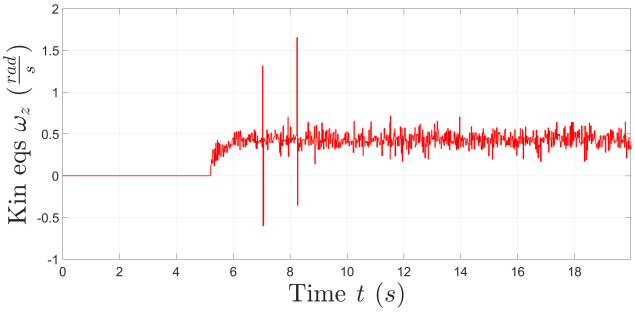


Figure 9.13: Angular speed ω_z

IMU measurements and estimations:

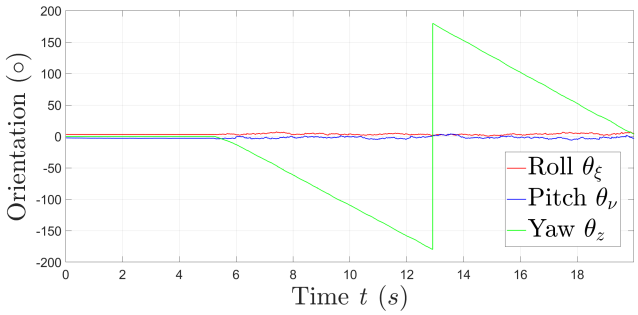


Figure 9.14: Robot orientation - IMU estimation using Kalman filter sensor fusion

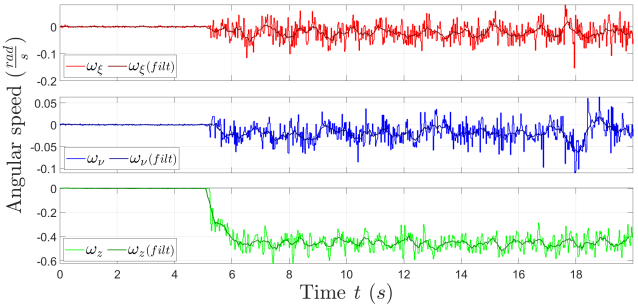


Figure 9.15: Robot angular acceleration - IMU measurements raw and filtered (moving average - 15 samples window)

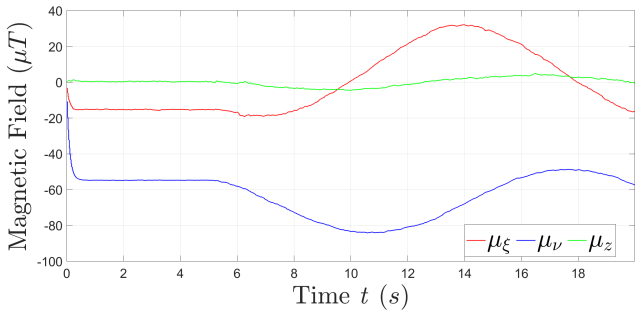


Figure 9.16: Magnetic field - IMU raw measurements

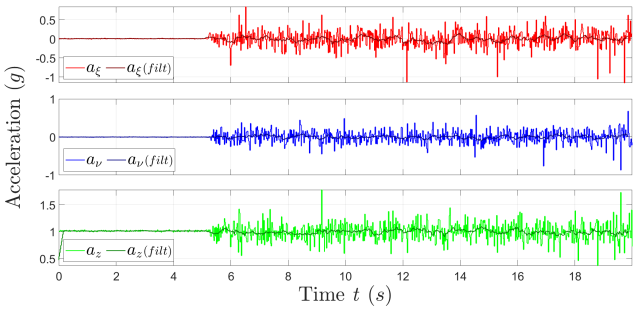


Figure 9.17: Robot Acceleration - IMU measurements raw and filtered (moving average - 15 samples window)

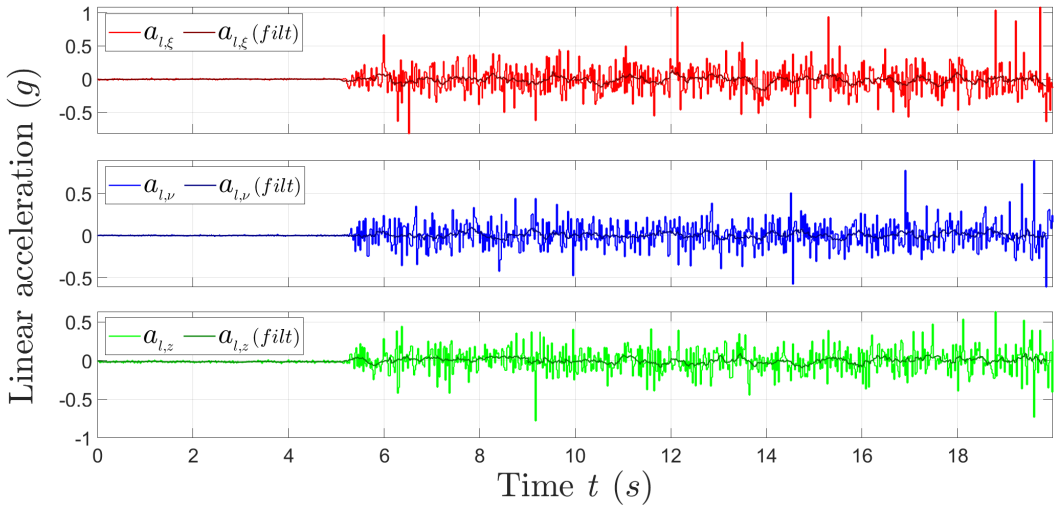


Figure 9.18: Robot linear acceleration - IMU estimation using orientation matrix, raw and filtered (moving average - 15 samples window)

Left and Right DC motors EMC output speeds:

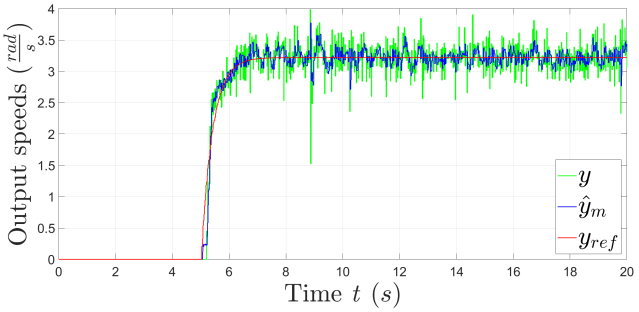


Figure 9.19: Output speeds reference y_{ref}, estimated $\hat{y}_m$ and measured y - Left motor

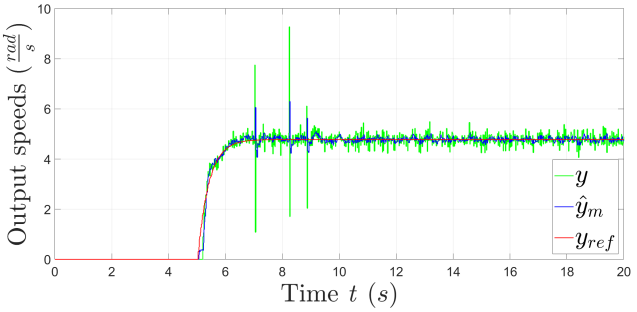


Figure 9.20: Output speeds reference y_{ref}, estimated $\hat{y}_m$ and measured y - Right motor

9.2.4 Magnetometer data in sensor fusion, Linear trajectory - Result 1

Trajectory estimated with kinematic equations:

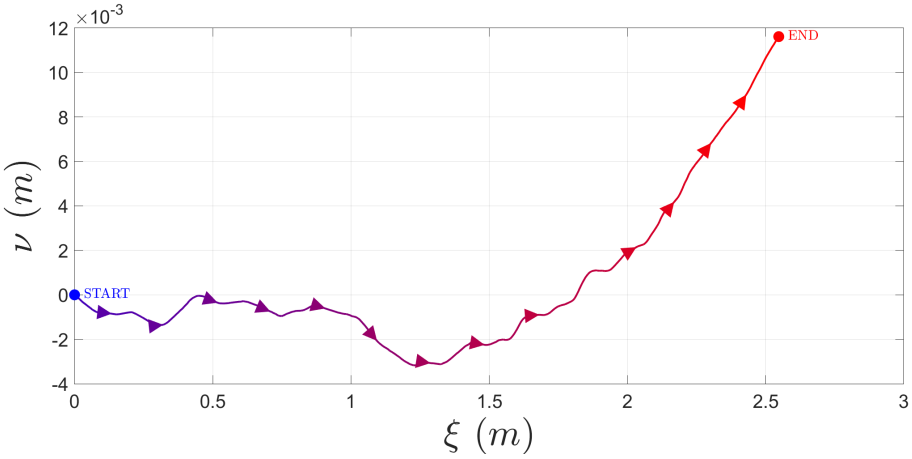


Figure 9.21: Robot position

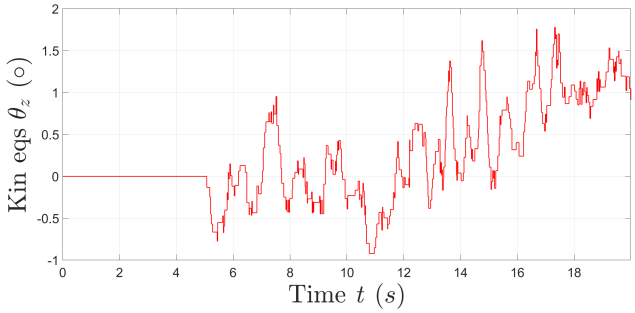


Figure 9.22: Robot orientation θ_z

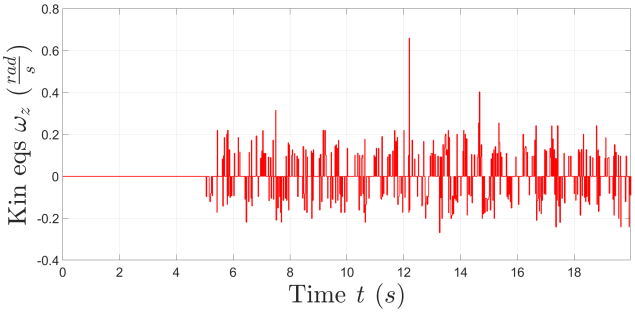


Figure 9.23: Angular speed ω_z

IMU measurements and estimations:

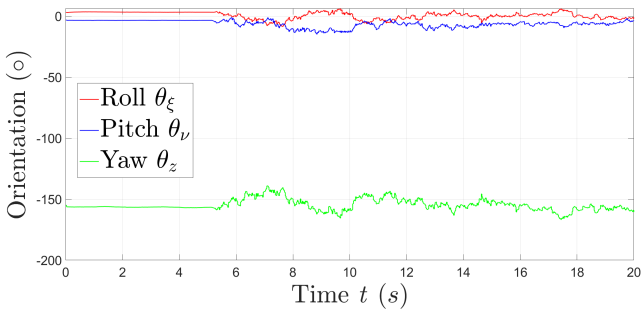


Figure 9.24: Robot orientation - IMU estimation using Kalman filter sensor fusion

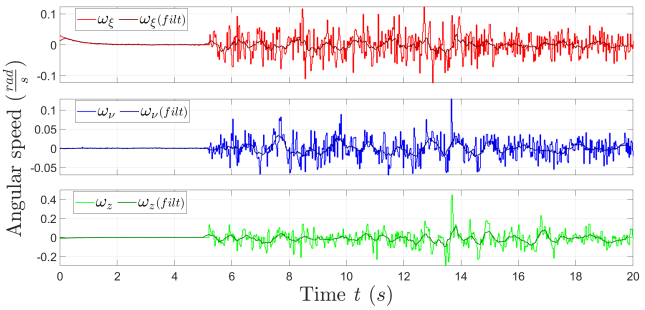


Figure 9.25: Robot angular acceleration - IMU measurements raw and filtered (moving average - 15 samples window)

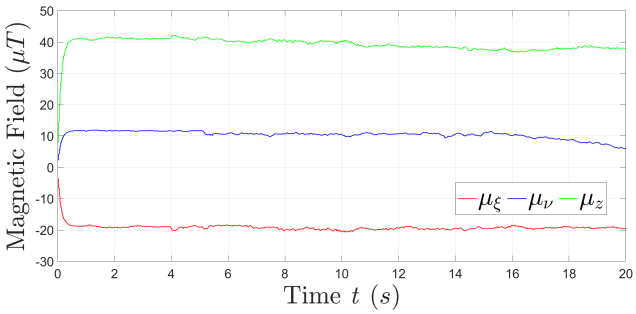


Figure 9.26: Magnetic field - IMU raw measurements

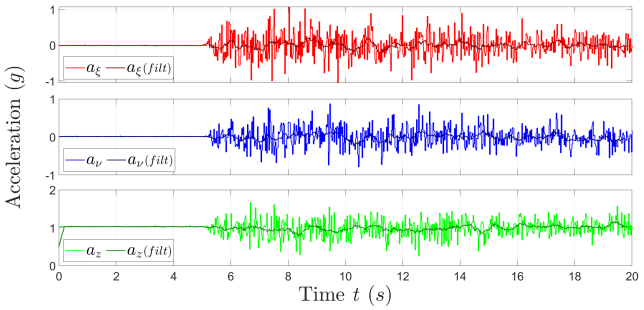


Figure 9.27: Robot Acceleration - IMU measurements raw and filtered (moving average - 15 samples window)

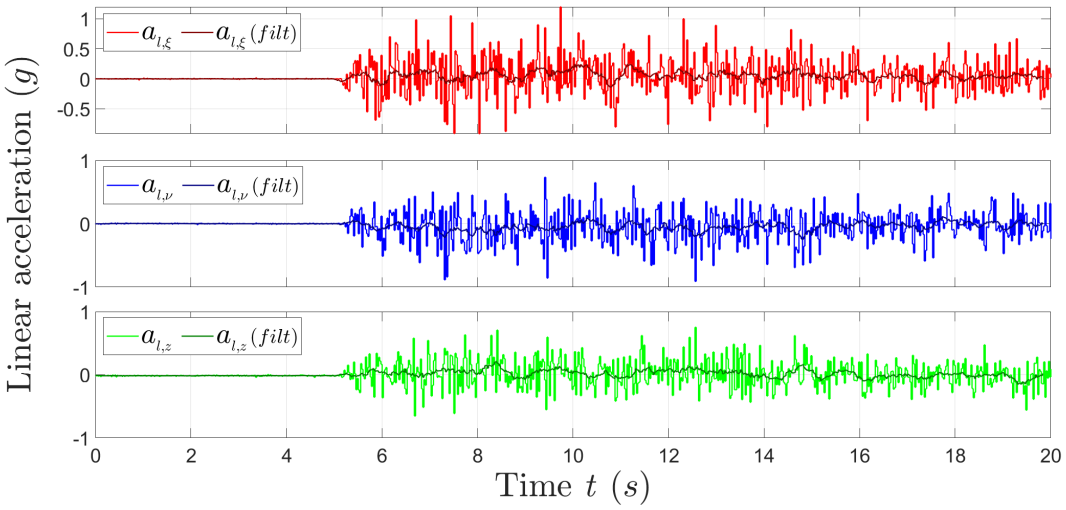


Figure 9.28: Robot linear acceleration - IMU estimation using orientation matrix, raw and filtered (moving average - 15 samples window)

Left and Right DC motors EMC output speeds:

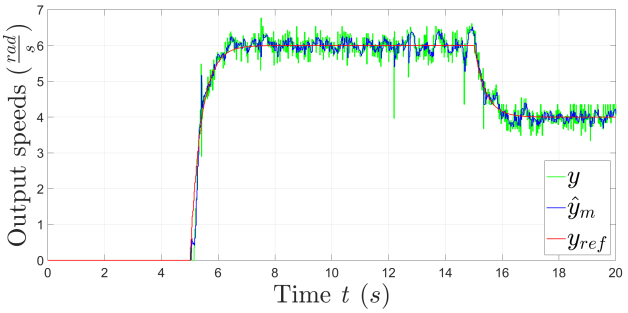


Figure 9.29: Output speeds reference y_{ref}, estimated $\hat{y}_m$ and measured y - Left motor

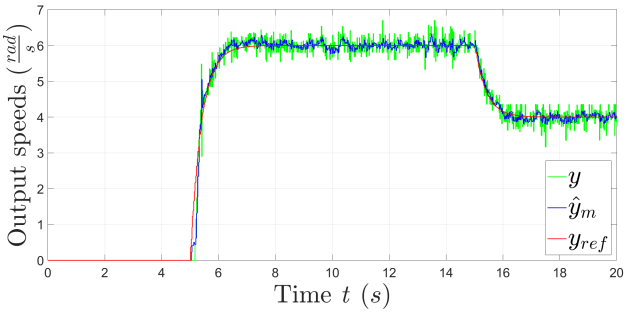


Figure 9.30: Output speeds reference y_{ref}, estimated $\hat{y}_m$ and measured y - Right motor

9.2.5 Magnetometer data in sensor fusion, Circular trajectory - Result 1

Trajectory estimated with kinematic equations:

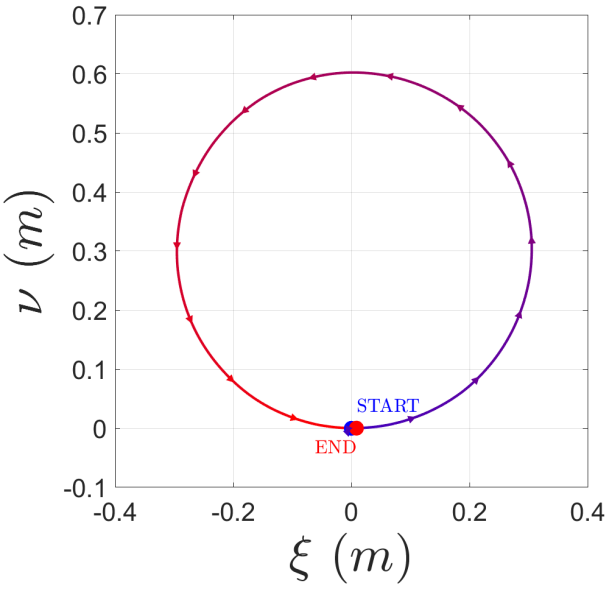


Figure 9.31: Robot position

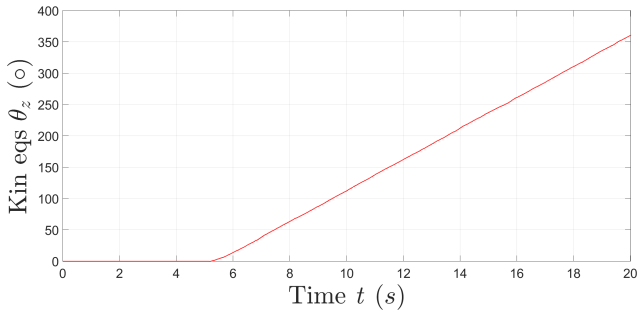


Figure 9.32: Robot orientation θ_z

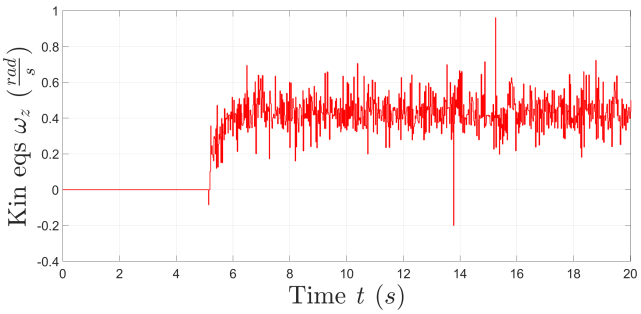


Figure 9.33: Angular speed ω_z

IMU measurements and estimations:

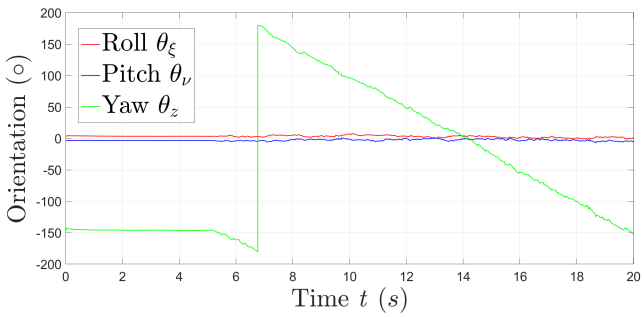


Figure 9.34: Robot orientation - IMU estimation using Kalman filter sensor fusion

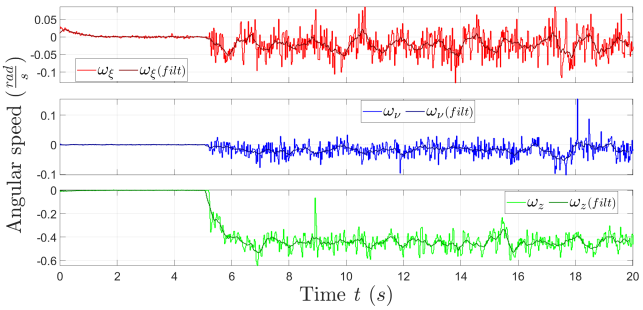


Figure 9.35: Robot angular acceleration - IMU measurements raw and filtered (moving average - 15 samples window)

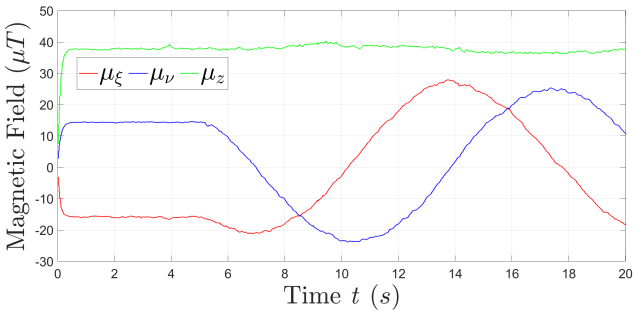


Figure 9.36: Magnetic field - IMU raw measurements

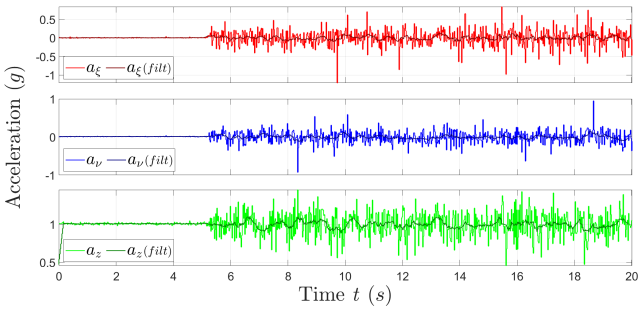


Figure 9.37: Robot Acceleration - IMU measurements raw and filtered (moving average - 15 samples window)

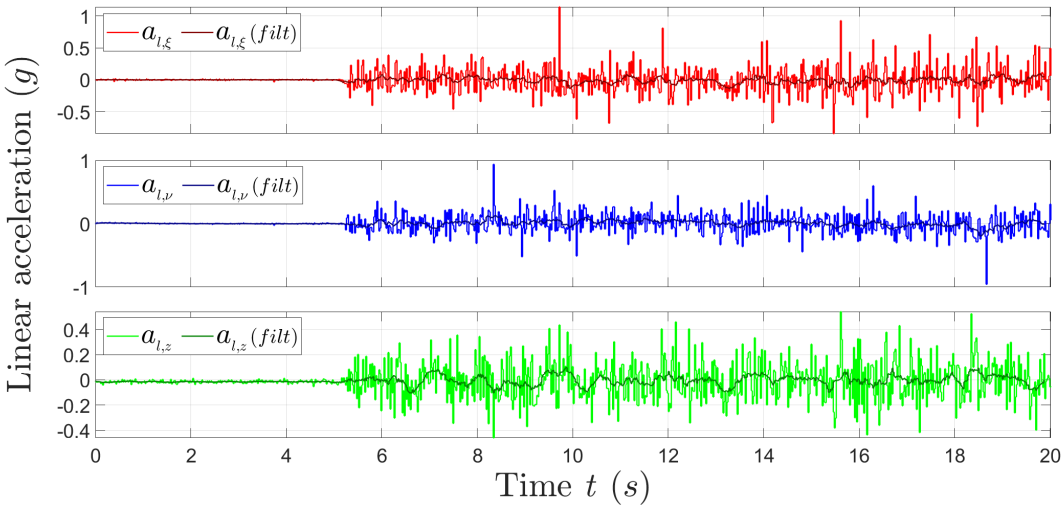


Figure 9.38: Robot linear acceleration - IMU estimation using orientation matrix, raw and filtered (moving average - 15 samples window)

Left and Right DC motors EMC output speeds:

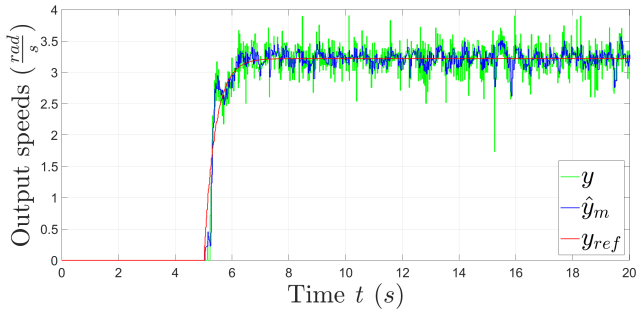


Figure 9.39: Output speeds reference y_{ref}, estimated $\hat{y}_m$ and measured y - Left motor

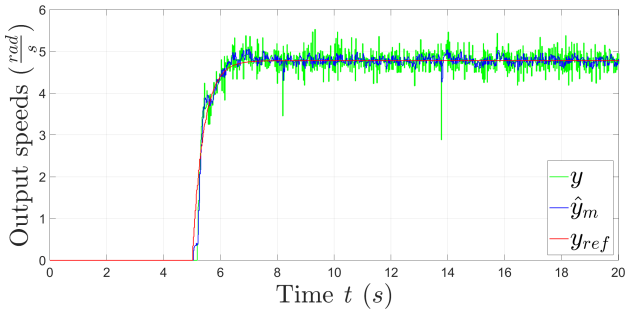


Figure 9.40: Output speeds reference y_{ref}, estimated $\hat{y}_m$ and measured y - Right motor

Part III

Robot orientation EMC

Chapter 10

EMC Differential drive robot theory - Orientation only

The DC motors EMC are already found and tested in Part II, hence they can be combined together with another EMC to control GoPiGo3 orientation angle θ_z and angular speed $\omega_z = \dot{\theta}_z$: with this control robot will be able to rotate in place (with the wheels moving with equal speed but in opposite directions), or to perform a circular trajectory of predefined mean radius wrt robot center of wheel axles.
In this chapter will be explained the theoretical analysis beyond the EMC and fine models regarding robot orientation.
Notation: From now on total robot inertia symbol will be I_T instead of I_{zz}^t (initially considered in Section 3.2), for visual simplicity. If this parameter is estimated is referred as $\hat{I}_T$.

10.1 Fine model

Dynamic equation of robot orientation in CT domain is the following (taken from Equation (3.7)):

$$\boldsymbol{\tau_T} = \sum_{i=1}^{2} \tau_i(t) = \tau_L(t) + \tau_R(t) = I_T \ddot{\theta}_z(t) \tag{10.1}$$

where I_T is the robot total inertia parameter, $\tau_T(t)$ is the total torque applied on robot, $\tau_L(t)$ and $\tau_R(t)$ are respectively the torques depending on the forces acting on the left and right wheels with respect to the robot wheel axles, $\ddot{\theta}_z(t)$ is the total angular acceleration of the robot.
Hence general CT ODE equations, without disturbances are:

$$\dot{\theta}_z(t) = \dot{\theta}_z(t)$$
$$\ddot{\theta}_z(t) = \frac{\tau_T(t)}{I_T}$$

For fine models the real total inertia parameter is $I_T = 0.002235\ \mathrm{kgm}^2$, computed with an Inventor robot CAD model and shown in Table 3.2.

10.2 EMC model

10.2.1 Embedded Model EM

Orientation robot EM can be build considering 2 integrators to recover θ_z and $\dot{\theta}_z$ from Equation (10.1): these will be the controllable states of the system. However, total robot inertia parameter changes to $\hat{I}_T = 0.0037\,\mathrm{kgm}^2$, found in Equation (3.2), because the robot is simplified as a parallelepiped. The difference between fine and EMC parameters lead to a small parametric uncertainty. CT state equations for EM with presence of disturbance states and noises are:

$$\begin{cases} \dot{\theta}_z(t) = \dot{\theta}_z(t) + \overline{w}_1(t) \\ \ddot{\theta}_z(t) = x_d(t) + \dfrac{\tau_T(t)}{I_T} + \overline{w}_2(t) \\ \dot{x}_d(t) = \overline{w}_3(t) \end{cases}$$

In a first EM model solution $\dot{\theta}_z(t)$ is not measured (and estimated), thus there is not any affecting disturbance $\overline{w}_1(t)$ (last equation red term vanishes). In a second solution also $\dot{\theta}_z(t)$ is measured and $\overline{w}_1(t)$ is added to related CT equation (in red).
Disturbance states $x_d(t)$, in a first attempt model, can be considered of 1st order, entering the system as acceleration perturbation. In CT, there is only 1 integrator and a non-causal disturbance. In DT integrator changes in a delay with a unitary feedback, with the input multiplied by sampling time T_s, if Forward Euler discretization method considered. Hence the CT integral can be approximated with a Forward Euler DT integrator:

$$\frac{1}{s} \approx \frac{T_s}{z-1} \tag{10.2}$$

Solution 1 - θ_z estimation only

In this case only θ_z is the measured state, hence no noise component $\overline{w}$ is present in its related SS equation.
If $\omega_z(k) = \theta_z(k+1)$, the state vector is $x^T(k) = [\theta_z, \omega_z, x_d](k)$ and input vector is $u(k) = \tau_T(k)$, following the EM form in Equation (2.1) and Equation (2.2), in matrix form the equations are:

$$\begin{cases} x(k+1) = \underbrace{\begin{bmatrix} 1 & T & 0 \\ 0 & 1 & T \\ 0 & 0 & 1 \end{bmatrix}}_{A_{DT}} x(k) + \underbrace{\begin{bmatrix} 0 \\ \frac{T}{I_T} \\ 0 \end{bmatrix}}_{B_{DT}} u(k) + \underbrace{\begin{bmatrix} 0 & 0 \\ T & 0 \\ 0 & T \end{bmatrix}}_{G_{DT}} \overline{w}(k) \quad , \quad x_0(k) = 0_{3\times 1} \\ y(k) = \underbrace{\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}}_{C_{DT}}(k) \end{cases} \tag{10.3}$$

All matrices are decomposed in the following submatrices:

$$A_{DT} = \begin{bmatrix} A_c & H_c \\ 0 & Ad \end{bmatrix} = \left[\begin{array}{cc|c} 1 & T & 0 \\ 0 & 1 & T \\ \hline 0 & 0 & 1 \end{array}\right] \quad , \qquad B_{DT} = \begin{bmatrix} B_c \\ 0 \end{bmatrix} = \left[\begin{array}{c} 0 \\ \frac{T}{I_T} \\ \hline 0 \end{array}\right]$$

$$G_{DT} = \begin{bmatrix} G_c \\ G_d \end{bmatrix} = \left[\begin{array}{cc} 0 & 0 \\ T & 0 \\ \hline 0 & T \end{array}\right] \quad , \qquad C_{DT} = \begin{bmatrix} C_c & C_d \end{bmatrix} = \left[\begin{array}{cc|c} 1 & 0 & 0 \end{array}\right]$$

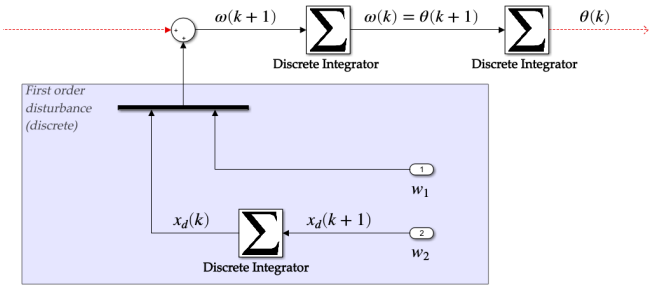


Figure 10.1: First order disturbance w entering orientation EM - Solution 1

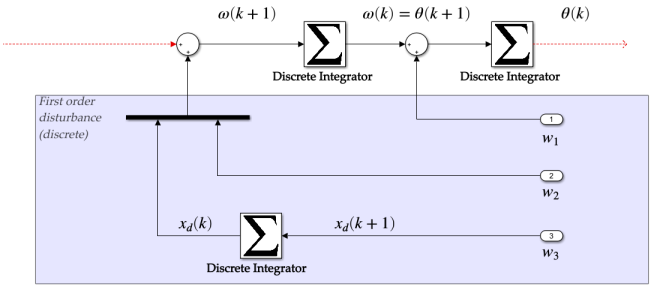


Figure 10.2: First order disturbance w entering orientation EM - Solution 2

Solution 2 - Both θ_z and ω_z estimation

In this case both θ_z and ω_z can be measured with sensors, hence in all SS equations $\overline{w}$ is present. Wrt Equation (10.3) it is needed to change DT SS equations, modifying C_{DT} and G_{DT} matrices (highlighted in red):

$$\begin{cases} x(k+1) = \underbrace{\begin{bmatrix} 1 & T & 0 \\ 0 & 1 & T \\ 0 & 0 & 1 \end{bmatrix}}_{A_{DT}} x(k) + \underbrace{\begin{bmatrix} 0 \\ \frac{T}{I_T} \\ 0 \end{bmatrix}}_{B_{DT}} u(k) + \underbrace{\begin{bmatrix} T & 0 & 0 \\ 0 & T & 0 \\ 0 & 0 & T \end{bmatrix}}_{G_{DT}} \overline{w}(k) \quad , \quad x_0(k) = 0_{3\times 1} \\ y(k) = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}}_{C_{DT}} (k) \end{cases} \tag{10.4}$$

All matrices are decomposed in the following submatrices:

$$\begin{aligned} A_{DT} &= \begin{bmatrix} A_c & H_c \\ 0 & Ad \end{bmatrix} = \left[\begin{array}{cc|c} 1 & T & 0 \\ 0 & 1 & T \\ \hline 0 & 0 & 1 \end{array}\right] \quad , \qquad B_{DT} = \begin{bmatrix} B_c \\ 0 \end{bmatrix} = \left[\begin{array}{c} 0 \\ \frac{T}{I_T} \\ \hline 0 \end{array}\right] \\ G_{DT} &= \begin{bmatrix} G_c \\ G_d \end{bmatrix} = \left[\begin{array}{ccc} T & 0 & 0 \\ 0 & T & 0 \\ \hline 0 & 0 & T \end{array}\right] \quad , \qquad C_{DT} = \begin{bmatrix} C_c & C_d \end{bmatrix} = \left[\begin{array}{cc|c} 1 & 0 & 0 \\ 0 & 1 & 0 \end{array}\right] \end{aligned} \tag{10.5}$$

A noise component $\overline{w}_1(k)$ is added to θ_z SS equation because both the controllable states are measured with an output sensor. Now 2 noise estimators are needed to estimate 2 state disturbances (combined in a proper way), and inevitably the matrices in Equation (10.4) need to be decomposed in 2 parts, using *matrix driven decomposition* explained in [1] and used for example in [3].
In Figure 10.1 and Figure 10.2, violet area shows the discrete time blocks used for first order disturbance, for both solution 1 and 2.

10.2.2 Control law

Equation (2.11) in Chapter 2 will be used for control law. K, M, Q matrices need to be found.
$Q^T = [q_1, q_2], M = m$ matrices can be univocally determined using 2^{nd} condition in Equation (2.12):

$$\text{Solution 1 and 2} \quad \begin{cases} q_1 = q_2 = 0 \\ m = I_T \end{cases} \tag{10.6}$$

Regarding matrix K, a proportional control only (P) is considered, to track the references states θ_z and ω_z. In this case $K = K_p = [k_{p1}, k_{p2}]$, which components are static gains.

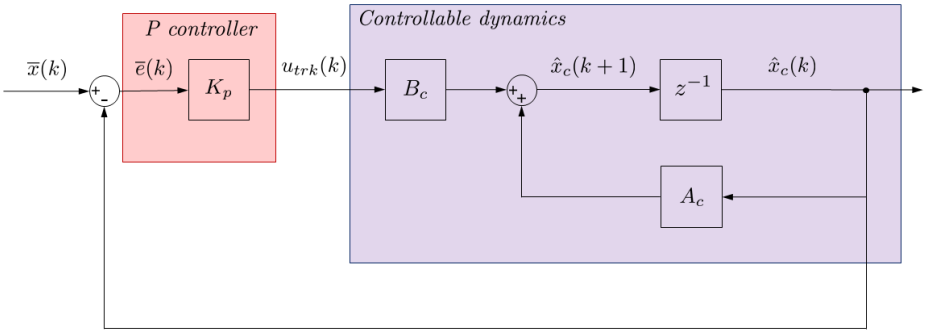


Figure 10.3: Orientation EMC - Proportional controller (P) Closed Loop scheme

Starting from the scheme in Figure 10.3, the CL equation becomes:

$$\begin{aligned}\hat{x}_c(k+1) &= A_c\hat{x}_c(k) + B_cK_p\overline{e}(k)\\ &= A_c\hat{x}_c(k) + B_cK_p\left(\overline{x}(k) - \hat{x}_c\right) =\\ &= \underbrace{\left[A_c - B_cK_p\right]}_{A_K}\hat{x}_c(k) + \underbrace{B_c}_{B_K}\overline{x}(k)\end{aligned}$$

where $\overline{x}$ is the reference state to be tracked. $\overline{e}(k) = \overline{x}(k) - \hat{x}_c(k)$ without the matrix Q (cfr. Equation (2.11)) because it has null components $q_1 = q_2 = 0$, as it is already found in Equation (10.6).
Pole placement technique can be used to find the 2 unknown values of k_{p1}, k_{p2}, to arbitrary decide the eigenvalues of $A_K = A_c - B_cK_p$, and make them stay in the unitary circle, to obtain closed loop asymptotic internal stability:

$$\begin{cases}\det\left(A_K - \lambda\mathbb{I}\right) = \lambda^2 + a_{cl,2}\lambda + a_{cl,3} & \textit{Closed loop characteristic polynomial}\\ \lambda^2 + a_{k2}\lambda + a_{k3} = (\lambda - p_{k1})(\lambda - p_{k2}) & \textit{Desired characteristic polynomial}\end{cases}$$

$p_K = [p_{k1}, p_{k2}]$ are arbitrary eigenvalues decided by us with $Re(\lambda) < 1$. Finally k_{p1} and k_{p2} can be computed as (in function of main orientation robot parameter I_T):

$$\begin{aligned}k_{p1} &= \frac{I_T(1 + a_{k2} + a_{k3})}{T^2}\\ k_{p2} &= \frac{I_T(a_{k2} + 2)}{T}\end{aligned} \tag{10.7}$$

10.2.3 Reference Dynamics

Static state FB control is selected to have θ_z and ω_z reference states, starting from desired targets $\tilde{\theta}_z, \tilde{\omega}_z$, to be tracked by Embedded Model estimated states (following CL equations in Equation (2.5) and controlling the CL matrix A_R through pole placement technique): the matrices to be designed are K_R to make the closed loop system asymptotically stable and a matrix N_R to make overall system DC-gain equal to 1.
In this case $r(k) = \tilde{\theta}_z$, since the most important state to be controlled is θ_z. ω_z is indirectly controlled since it depends on the other (θ_z is the integral of ω_z). $C_R = \begin{bmatrix}1 & 0\end{bmatrix}$ since only θ_z is considered at output.
$K_R = [k_{r1}, k_{r2}]$ can be found by making equal the closed loop characteristic polynomials (defined by $\det\left(A_K - \lambda\mathbb{I}\right)$)

and a desired characteristic polynomial, with eigenvalues $p_R = [p_{r1}, p_{r2}]$ staying inside the unitary circle to obtain asymptotic internal stability:

$$\begin{cases} \det(A_K - \lambda\mathbb{I}) = \lambda^2 + a_{cl,2}\lambda + a_{cl,3} & \textit{Closed loop characteristic polynomial} \\ \lambda^2 + a_{r2}\lambda + a_{r3} = (\lambda - p_{r1})(\lambda - p_{r2}) & \textit{Desired characteristic polynomial} \end{cases}$$

k_{r1} and k_{r2} are finally (in function of main orientation robot parameter I_T):

$$\begin{aligned} k_{r1} &= \frac{I_T(1 + a_{r2} + a_{r3})}{T^2} \\ k_{r2} &= \frac{I_T(a_{r2} + 2)}{T} \end{aligned} \tag{10.8}$$

After K_R is known, N_R matrix can be computed with Equation (2.6).

10.2.4 Noise Estimator

Solution 1 - Dynamic FB filter for θ_z estimation only

Depending on disturbances and states dimensions, a static or dynamic noise estimator can be used: since Equation (10.3) have $\dim \overline{w} = n_w = 2$ lower than dimension of total states $\dim x = n_x = 3$, a dynamic filter is needed, cfr. [1, 2]. We can follow the procedure explained in Section 2.1, designing eigenvalues of CL matrix A_{CL} in Equation (2.10). The unknowns of Equation (2.9) become in this case:

$$N = \begin{bmatrix} n_0 \\ n_1 \end{bmatrix} \quad , \quad L = \begin{bmatrix} l_0 \\ 0 \end{bmatrix} \quad , \quad A_e = 1 - \beta \tag{10.9}$$

Using the usual pole placement technique, and considering the desired characteristic polynomial as (starting from desired eigenvalues $p_N = [p_{n1}, p_{n2}, p_{n3}, p_{n4}]$):

$$\lambda^4 + a_{n2}\lambda^3 + a_{n3}\lambda^2 + a_{n4}\lambda + a_{n5} = (\lambda - p_{n1})(\lambda - p_{n2})(\lambda - p_{n3})(\lambda - p_{n4})$$

The final unknowns are:

$$\begin{aligned} l_0 &= \frac{3a_{n2} + a_{n3} + 6}{T^2} \\ n_0 &= -\frac{15a_{n2} + 2a_{n3} - a_{n4} + a_{n2}a_{n3} + 3a_{n2}^2 + 20}{T^2} \\ n_1 &= \frac{a_{n2} + a_{n3} + a_{n4} + a_{n5} + 1}{T^3} \\ \beta &= a_{n2} + 4 \end{aligned} \tag{10.10}$$

We can notice that β is the unique unknown without sampling time T. It's not a problem, indeed is possible that the dependence on T vanishes after computations.

Solution 2 - Two static FB noise estimators for both θ_z and $\dot{\theta}_z$ estimation

At least 2 noise estimators are needed to estimate the noises of θ_z and ω_z and combine them to obtain $\overline{w}$. A *measure-driven decomposition* is performed following an example of web winding EMC control in [1, 3].
Since A_{DT} and G_{DT} matrices are upper triangular, they can be divided into diagonal sub-matrices, again upper triangular. How many row divisions depend on number of measured outputs: for our equations, $\dim y = n_y = 2$,

so 2 rows:

$$A_{DT} = \left[\begin{array}{c|cc} 1 & T & 0 \\ \hline 0 & 1 & T \\ 0 & 0 & 1 \end{array}\right] \quad , \quad G_{DT} = \left[\begin{array}{c|cc} T & 0 & 0 \\ \hline 0 & T & 0 \\ 0 & 0 & T \end{array}\right] \quad , \quad C_{DT} = \left[\begin{array}{cc|c} 1 & 0 & 0 \\ \hline 0 & 1 & 0 \end{array}\right] \tag{10.11}$$

To build the 2 noise estimators only the diagonal sub-matrices are relevant. For the 1st noise estimator (θ_z) they are:

$$A^0_{DT} = 1 \quad , \quad G^0_{DT} = T \quad , \quad C^0_{DT} = 1 \tag{10.12}$$

hence a static FB estimator can be exploited, using pole placement technique as in Equation (2.7). The noise $\overline{w}_0$ can be estimated as:

$$\begin{aligned} \overline{w}_0(k) &= L_0 \overline{e}_{m0}(k) \\ L_0 &= l_0 = \frac{a^0_n + 1}{T} \end{aligned} \tag{10.13}$$

where $a^0_n = -p_{n0}$, and p_{n0} is the desired eigenvalue arbitrary decided by us. In this case model error is $\overline{e}_{m0} = \theta_z - \hat{\theta}_z$. Instead for the 2nd noise estimator (ω_z) the sub-matrices are:

$$A^1_{DT} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \quad , \quad G^1_{DT} = \begin{bmatrix} T & 0 \\ 0 & T \end{bmatrix} \quad , \quad C^1_{DT} = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{10.14}$$

hence another static FB estimator can be exploited. The noise $\overline{w}_1$ can be estimated as:

$$\begin{aligned} \overline{w}_1(k) &= L_1 \overline{e}_{m1}(k) \\ L_1 &= \begin{cases} l_{10} = \dfrac{a^1_{n2} + 2}{T} \\ l_{11} = \dfrac{a^1_{n2} + a^1_{n3} + 1}{T^2} \end{cases} \end{aligned} \tag{10.15}$$

With a^1_{n2} and a^1_{n3} recovered with the following desired characteristic polynomial, considering p_{n1} and p_{n2} decided arbitrary (pole placement technique):

$$\lambda^2 + a^1_{n2}\lambda + a^1_{n3} = (\lambda - p_{n1})(\lambda - p_{n2})$$

In this case model error is $\overline{e}_{m1} = \omega_z - \hat{\omega}_z$. Total estimated noise is:

$$\overline{w} = \begin{bmatrix} \overline{w}_0 \\ \overline{w}_1 \end{bmatrix}$$

Solution 3 - Static FB noise estimator for θ_z and Dynamic for ω_z estimation

Using this method again 2 noise estimators are combined together to estimate the noises $\overline{w}$, following *measure-driven decomposition* procedure as Solution 2: the matrices are divided again as Equation (10.11).
For 1st noise estimator (related to θ_z) the sub-matrices are exactly as Equation (10.12). Hence the estimation can be obtained with static FB and the noise $\overline{w}_0$ can be estimated exactly as Solution 2, Equation (10.13).
Instead for the 2nd noise estimator (ω_z) the sub-matrices are the same as Equation (10.14): however instead of using another static estimator as Solution 2, a static plus dynamic FB noise estimators combined will be used, like shown in Figure 10.4. The static FB part is composed by $L^T_1 = [l_{10}, l_{11}] \in \mathbb{R}^{2\times 1}$ matrix. Using pole placement technique the component expressions are the same of Equation (10.15). Instead the dynamic FB part has matrix unknowns as

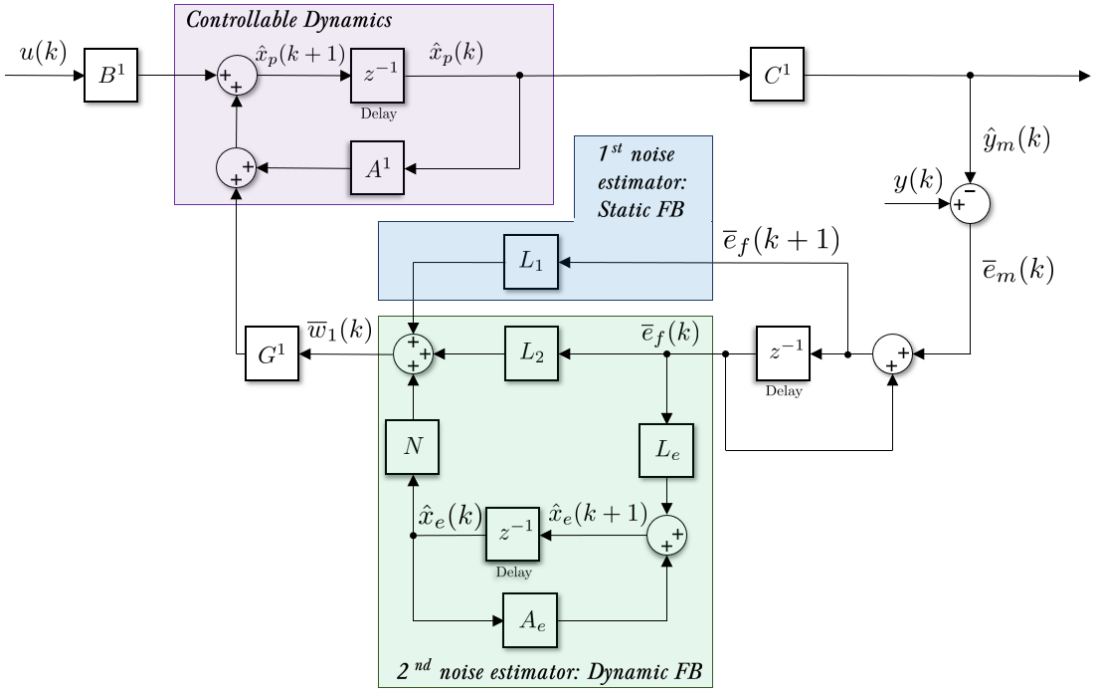


Figure 10.4: Noise estimator Orientation EMC Solution 3 - Static + dynamic FB parts

Equation (10.9):

$$N = \begin{bmatrix} n_0 \\ n_1 \end{bmatrix} \quad , \quad L_2 = \begin{bmatrix} l_{10} \\ 0 \end{bmatrix} \quad , \quad A_e = 1 - \beta$$

Only L_2 matrix has the non-null component l_{10} equal to the one of static part noise estimator for $\omega_z(k)$, already known. The other unknowns are:

$$\begin{aligned} n_0 &= \frac{(2 - 20T)a_{n2} - 60T + a_{n3} + 400T^2 + 3}{T} \\ n_1 &= \frac{a_{n2} + a_{n3} + a_{n4} + 1}{T^2} \\ \beta &= a_{n2} - 20T + 3 \end{aligned} \tag{10.16}$$

With a_{n2}, a_{n3}, a_{n4} are obtained using desired characteristic polynomial as below (at the beginning for pole placement we decide p_{n1}, p_{n2}, p_{n3} values inside unitary circle for asymptotic internal stability):

$$\lambda^3 + a_{n2}\lambda^2 + a_{n3}\lambda + a_{n4} = (\lambda - p_{n1})(\lambda - p_{n2})(\lambda - p_{n3})$$

The noise $\overline{w}_1$ can be estimated as:

$$\overline{w}_1(k) = L_1\overline{e}_{m1}(k) + L_2\overline{e}_f(k) + Nx_e(k) \tag{10.17}$$

The presence of a dynamic FB part in noise estimator adds a new state in the system, $\overline{e}_f$, which filters the noise estimation. Total estimated noise is again:

$$\overline{w} = \begin{bmatrix} \overline{w}_0 \\ \overline{w}_1 \end{bmatrix}$$

Comparison between the 3 noise estimator solutions

To have an insight of which one between Solution 1 (dynamic FB noise estimator), Solution 2 (2 static FB noise estimators combined with measure-driven decomposition method) and Solution 3 (2 static and dynamic FB noise estimators using measure-driven decomposition method) can be the best, simulations with complete EMC and fine models are done in open-loop, without considering the DC motors EMC.
Parameter settings for Fine and EMC models are explained in previous sections. Noise errors are added to θ_z and ω_z fine model measurements: ϵ_{θ_z} = 0–0.1 for Solution 1, and $\epsilon_{\theta_z} = \epsilon_{\omega_z}$ = 0–0.1 for Solutions 2 and 3, these values are provisional before making closed-loop MIL and RHIT tests. CT eigenvalues used for each solution test are:

CT eigenvalue type	Sol. 1 (Dyn FB)	Sol. 2 (Two Static FB)	Sol. 3 (Static + Dyn. FB)
Control μ_K	$-1.0101, -17.8337$	$-2.5647, -11.1572$	$-2.5647, -11.1572$
Reference μ_R	-1.0101×2	-1.0101×2	-1.0101×2
Observer μ_N	$-2.5702, -2.5647 \times 2, -2.5591$	-17.8337 (θ_z est)	-17.8337 (θ_z est)
		-17.8337×2(ω_z est)	-11.1572×2 (ω_z est 1)
			-34.6582×3 (ω_z est 2)

Table 10.1: Provisional Eigenvalue settings Orientation EMC - Solution comparisons between Noise estimators

In the next figures (Figures 10.5 to 10.7) are shown reference, estimated and measured outputs related to θ_z and ω_z, using the 3 noise estimator solutions.

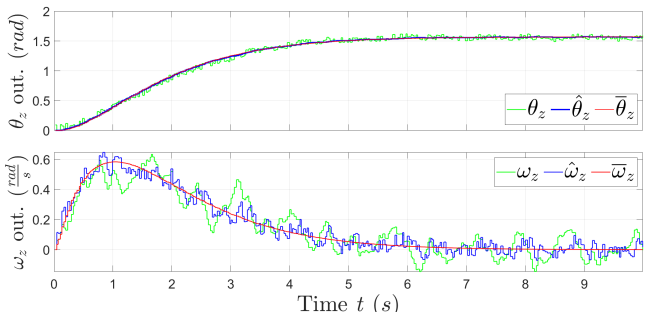


Figure 10.5: Orientation EMC Solution 1 outputs - Dynamic FB noise estimator

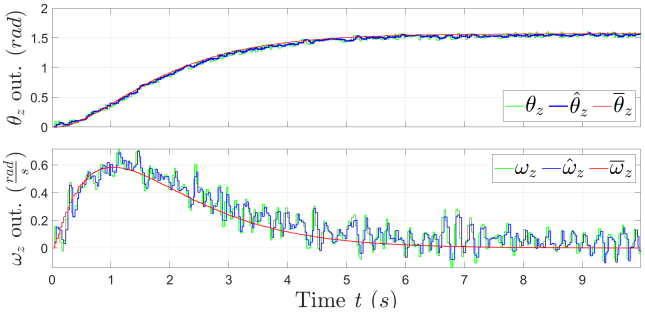


Figure 10.6: Orientation EMC Solution 2 outputs - Static FB noise estimators with measure-driven decomposition

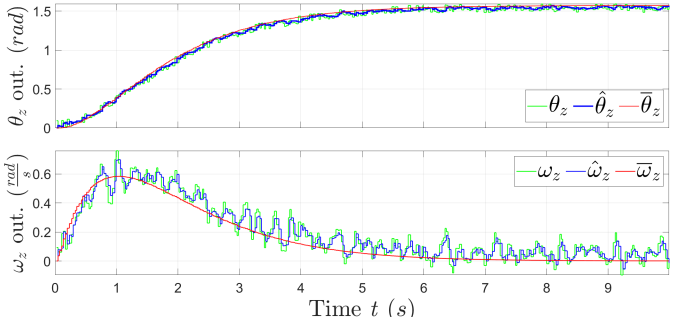


Figure 10.7: Orientation EMC Solution 3 outputs - Static and dynamic FB noise estimators with measure-driven decomposition

We can immediately see that for all solutions model and tracking errors concerning θ_z are very low, instead for ω_z:

- Using the dynamic feedback noise estimator (Solution 1, Figure 10.5), tracking error is acceptable, but model error is quite high: however if we take into account that observer was build only for θ_z, it's a good result.

- Using 2 static feedback noise estimators with measure-driven decomposition method (Solution 2, Figure 10.6), both tracking and model errors are low, since observers are build for both states.

- No significant improvements are obtained using measure-driven decomposition with both static and dynamic estimators (Solution 3, Figure 10.7) wrt to only static FB measure-driven decompsition (Solution 2, Figure 10.6), because both tracking and model errors are not improved so much.

After RHIT we'll see that Solution 2 is the best, and some comparisons with the other solutions will be shown.

Chapter 11

Orientation EMC - Model-in-the-Loop (MIL) simulation tests

Theoretical models explained in Chapter 10 for controlling robot orientation are now translated in Simulink block models to be tested with MIL procedure, where both plants and EMC are executed in simulation.

11.1 General MIL settings on Simulink

11.1.1 Simulated orientation EMC and fine models

Orientation fine model is only constituted by 2 integrators and a gain (total robot inertia I_T) to obtain robot angular position θ_z and speed ω_z simulation measurements from Equation (10.1).
2 uniform random noise disturbances are added to θ_z and ω_z, replicating the measurement sensor errors. By looking at IMU measurements it can be only recovered noise and disturbance informations when robot is still, because they change in an unpredictable way when robot is moving, for presence of other dynamics disturbing the sensors.
In simulation these errors are approximated as uniform random numbers added to angle and angular speed fine model outputs (***not*** directly inside the model between integrators), their bounds are:

$$e_{\theta_z} = [-1, 1]^\circ \approx [-0.0175, 0.0175] \text{ rad}$$
$$e_{\omega_z} = [-0.005, 0.005] \frac{\text{rad}}{\text{s}}$$

The bounds are low because it was experimented that using too high errors lead to very different MIL simulations wrt RHIT, when making comparisons.
Instead Orientation EMC is exactly the same explained in Section 10.2, using static FB measure-driven decomposition noise estimator (it represents the best solution among noise estimators). Virtual torque is saturated at $1\,\text{N}\,\text{m}$.

11.1.2 Orientation and DC motors Hierarchical structure

First it is necessary to understand how to connect the motors EMC with new orientation control model. Among the possible solutions, it is decided to use a *hierarchical structure*. Prerequisite is to have DC motors EMC build and tested with MIL and RHIT procedures (done in Part II). In Figure 11.1 there is a sketch structure of hierarchical structure to be implemented in Simulink for MIL tests, main features can be resumed as follows:

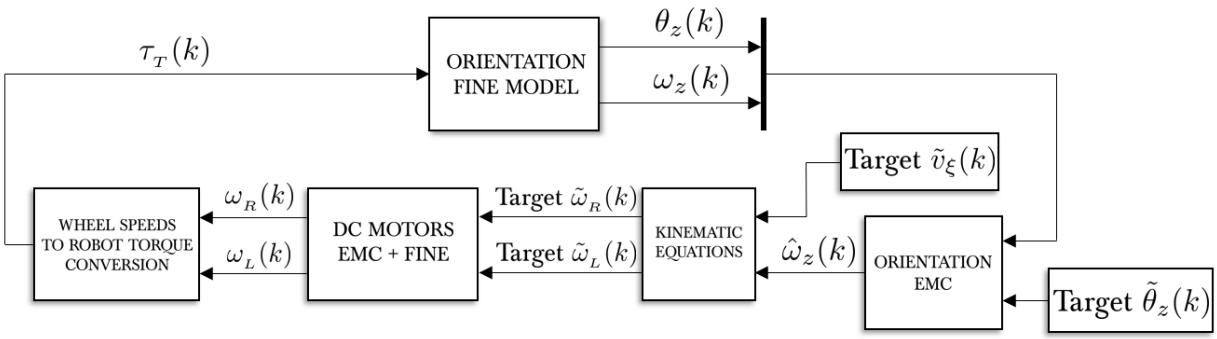


Figure 11.1: Hierarchic structure for MIL tests - Orientation and DC motors EMC

- At the beginning a robot target angle $\tilde{\theta}_z(k)$ and a reference trajectory inside Orientation EMC are decided. Orientation EMC will be able to make the estimates $\hat{\theta}_z(k), \hat{\omega}_z(k)$ as near as possible to desired trajectory reference $\overline{\theta}_z(k), \overline{\omega}_z(k)$ states and measurement $\theta_z(k), \omega_z(k)$ states, at the same time.

- Orientation EMC output will be the estimate $\hat{\omega}_z(k)$. Now it's needed to convert angular and linear robot speeds into DC motor wheel speeds: kinematic equations are used (Equation (3.15)), but since no position control is built yet, a target longitudinal speed $\tilde{v}_\xi(k) = 0\,\mathrm{m/s}$ is imposed, hence the robot is only allowed to round still in place. Another possibility is to impose a circular trajectory of the robot, defining its *mean radius*: from it wheel angular speeds can be recovered exploiting kinematic equations in a different way (Equation (3.16)).

- Wheel speeds coming from kinematic equations are used as *targets* for DC motors EMC ($\tilde{\omega}_R(k), \tilde{\omega}_L(k)$). This is the most interesting point of this structure: estimates coming from another control model (Orientation EMC) will be used as targets for the DC motors EMC.

- In Figure 11.1, there is a block containing CL EMC and fine models for both left and right motors. The outputs of DC motor fine models will be the measured wheel speeds $\omega_R(k), \omega_L(k)$, they are derivated with Forward-Euler discretization to obtain angular accelerations $\alpha_R(k), \alpha_L(k)$ and robot torque $\tau_T(k)$ is obtained exploiting Equation (3.8), $\tau_T(k) = I_T \alpha_z(k)$. Then torque is converted in CT domain and enters as input of Orientation fine model, to simulate robot real dynamic behaviour in terms of orientation.

- The loop is closed by taking the measurements $\theta_z(t), \omega_z(t)$ from Orientation fine model, which are then converted in DT to enter Orientation EMC.

The structure is hierarchical because the outputs of Orientation EMC will be used in DC motors EMC.
This is also a *non-conventional scheme*: first for the presence of estimated state $\hat{\omega}_z$ coming out from Orientation control model, and second for kinematic conversion *outside* the EMC.
Indeed usually the conversions are done inside the control models, with motor and orientation dynamics combined together. This unavoidably leads to the presence of non linearities inside DT SS (trigonometric functions) which are difficult to be handled.
Instead, using the structure above, all control models are *linear*, and equations are drastically simplified. We will

see in MIL tests that this kind of structure gives satisfactory results.

11.2 General settings on Simulink MIL plot results

DC motors EMC CT eigenvalues for DC motors are the same as Table 6.1, for both right and left motors. Instead for Orientation EMC the same final CT eigenvalues giving the best results in RHIT are considered, as described in Table 12.2 and Table 12.3.
Main trajectories followed are:

- Robot CoM still (robot linear speed $\tilde{v}_\xi = 0\,\frac{\mathrm{m}}{\mathrm{s}}$) and $\tilde{\theta}_z = [0, \pm 2\pi]\mathrm{rad}$ angle trajectory.

- Robot in movement following a circle trajectory (robot linear speed $\tilde{v}_\xi \neq 0\,\frac{\mathrm{m}}{\mathrm{s}}$ and mean radius $r_m = 0.2\,\mathrm{m}$), $\tilde{\theta}_z = [0, \pm 2\pi]\mathrm{rad}$ angle trajectory.

MIL simulations are not done once and all before RHIT, indeed the final structure is the result of continuous comparison between MIL and RHIT.
For this reasons every set of results is divided in 2 parts: the first plots are related to MIL simulations only, showing $y, \hat{y}_m$ and y_{ref} for angle θ_z and angular speed ω_z, with relative tracking and model errors e_m, e_{trk}, for Orientation control model. In addition are shown measurements and estimations of DC motor output speeds $y, \hat{y}_m$ and y_{ref}.
The second set of plots instead shows a comparison between MIL and RHIT testing procedures, where some RHIT results are taken from Chapter 12.
Disturbance rejection term is present in control input for all tests, unless for one of them (Section 11.3.2).

11.3 Summary on the obtained results

Regarding only MIL simulations, the selected EMC for orientation seems to work efficiently, reducing model and tracking errors as low as possible, at least for robot orientation angle θ_z. The angle trajectory is followed in a quite precise way. Angular speed ω_z trajectory is followed partially, and also model error is not very low, but remember that the 2 states are not independent and orientation angle is the most important to be controlled.
Only in the final section of trajectory the measurement and estimation states start oscillating around the reference signal: this is coherent with reality, because robot stops after reaching the first time the final position, but then restarts and re-stop many times because of DC motor dead-zones (explained in Section 6.1).
Looking at the comparison between MIL and RHIT, the Timestamp obtained with RHIT is reproduced quite exactly in MIL simulation, for all control models, meaning that also in this case the variable timestamp Simulink block model built in Section 4.1.1 is reliable. Also a comparison between the 2 virtual control inputs is evidenced, to have an indication of model reliability and control behaviour. For example if control torque is too high or if is not near to zero when robot is stopping, it probably means that EMC is forcing to reduce model and tracking errors without success (for example in Figure 11.23 and Figure 11.34).
The behaviour of MIL vs RHIT estimations and measurements are quite the same, but with some slight differences. For example the simulated and IMU measurements are not exactly the same, and the behaviour when trajectory is near to be reached changes rapidly. In particular the last difference is due to the DC motor fine model, which is not precisely the same as real, especially for voltage input dead-zones. Finally control inputs in the 2 tests present some differences: this is again dependent on disturbances and noises acting on real robot not present in simulated fine models.
In Section 11.3.2 MIL test the absence of disturbance rejection does not lead to worse tracking and model errors, since the orientation is not affected by too many disturbances like the DC motor (indeed there is only robot total inertia I_T parameter). This will be proven in orientation RHIT tests too (Section 12.4.2).

11.3.1 Target angle $\tilde{\theta}_z = 2\pi$ rad (360°), target long. speed $\tilde{v}_\xi = 0$ $\frac{\mathbf{m}}{\mathbf{s}}$, dist. rej.

MIL simulation test results:

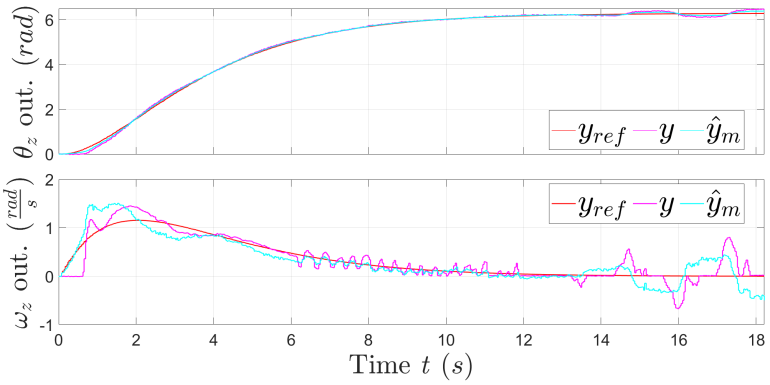


Figure 11.2: MIL Orientation EMC outputs y_{ref} (red), y (magenta) and $\hat{y}_m$ (cyan)

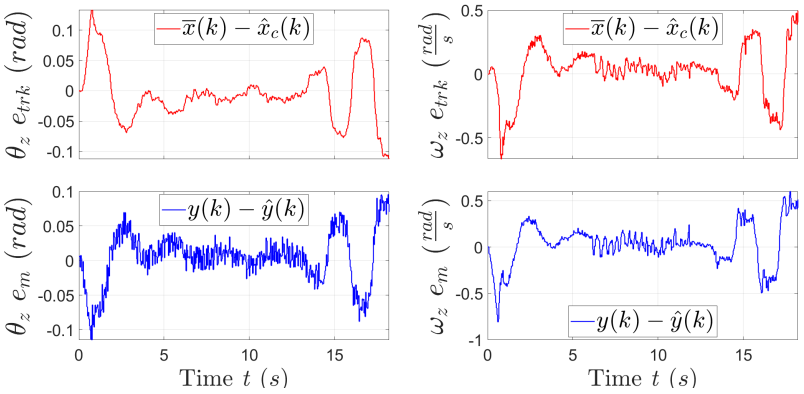


Figure 11.3: MIL Orientation EMC tracking e_{trk} and model e_m errors

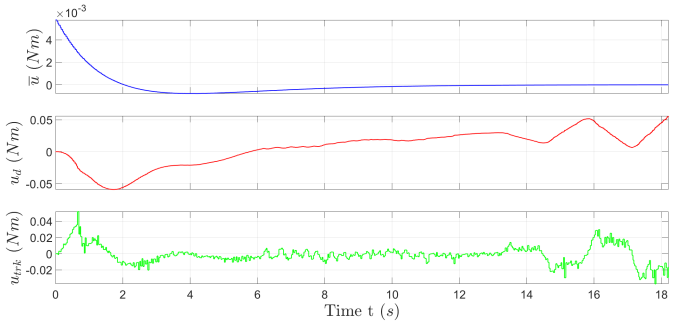


Figure 11.4: MIL Orient. EMC control input terms $\overline{u}, u_d$ and u_{trk} (from top to bottom) - Disturbance rejection

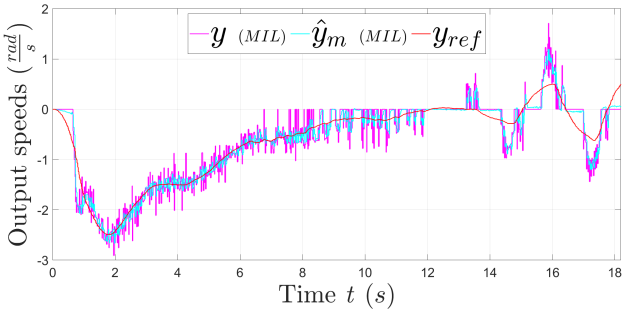


Figure 11.5: MIL Left motor EMC outputs y (magenta), $\hat{y}_m$ (cyan) and y_{ref} (red)

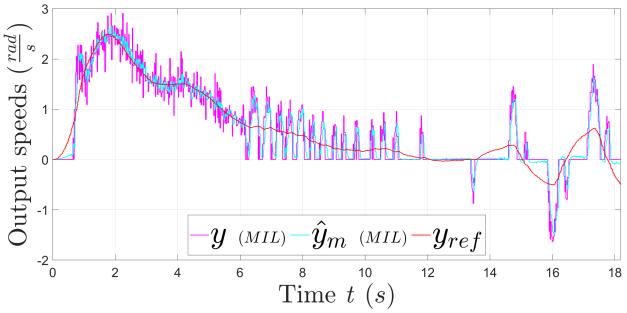


Figure 11.6: MIL Right motor EMC outputs y (magenta), $\hat{y}_m$ (cyan) and y_{ref} (red)

Comparison between MIL and RHIT:

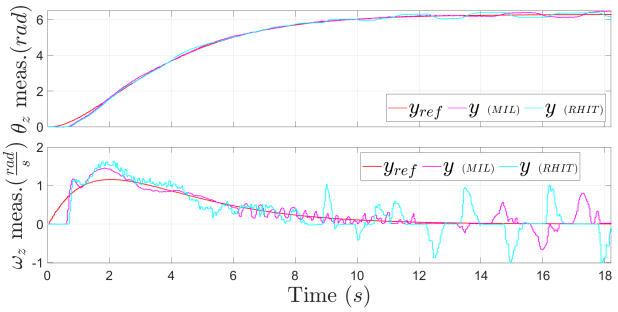


Figure 11.7: Orientation EMC measured outputs y (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

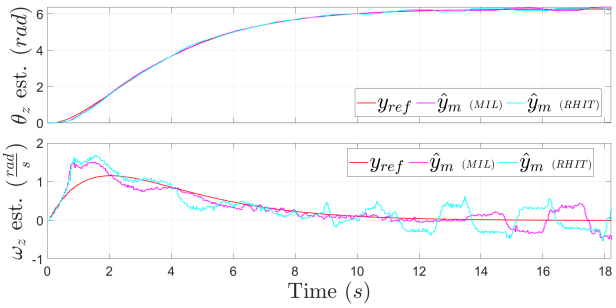


Figure 11.8: Orientation EMC estimated outputs $\hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

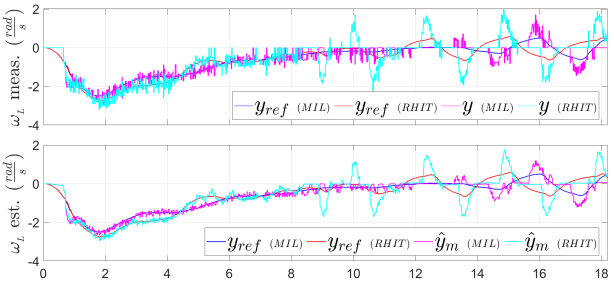


Figure 11.9: Left DC motor EMC measured outputs $y, \hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

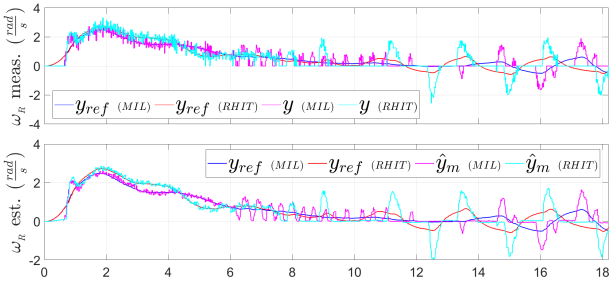


Figure 11.10: Right DC motor EMC measured outputs $y, \hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

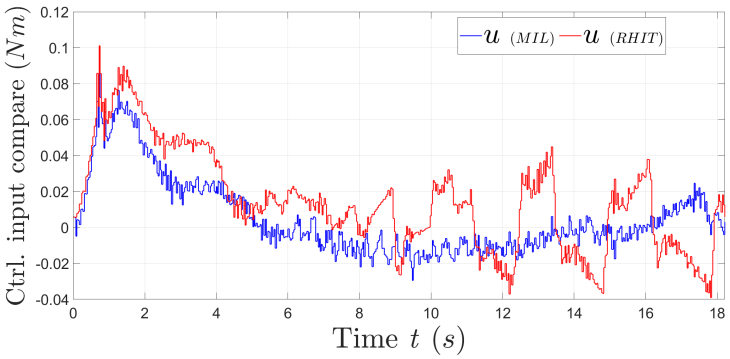


Figure 11.11: Orientation EMC virtual control input u - Comparison MIL and RHIT

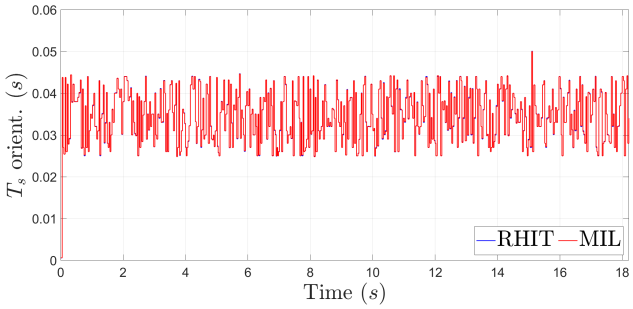


Figure 11.12: Orientation EMC Timestamp - Comparison MIL and RHIT

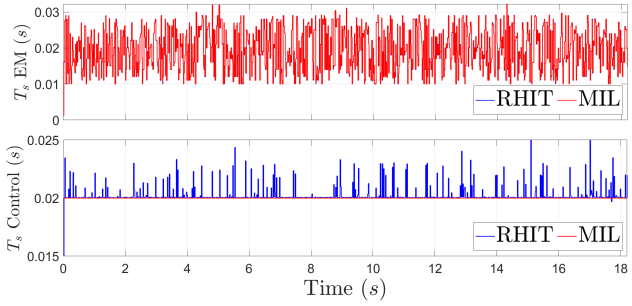


Figure 11.13: (Left and right) DC motors EMC Timestamp EM and Control part - Comparison MIL and RHIT

11.3.2 Target angle $\tilde{\theta}_z = 2\pi$ rad (360°), target long. speed $\tilde{v}_\xi = 0$ $\frac{\mathbf{m}}{\mathbf{s}}$, No dist. rej.

MIL simulation test results:

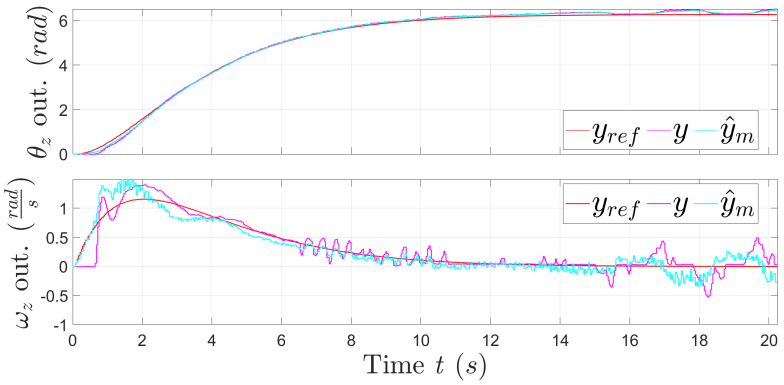


Figure 11.14: MIL Orientation EMC outputs y_{ref} (red), y (magenta) and $\hat{y}_m$ (cyan)

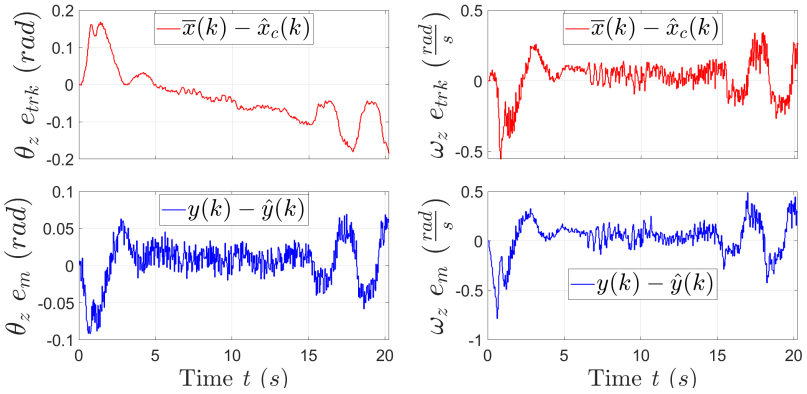


Figure 11.15: MIL Orientation EMC tracking e_{trk} and model e_m errors

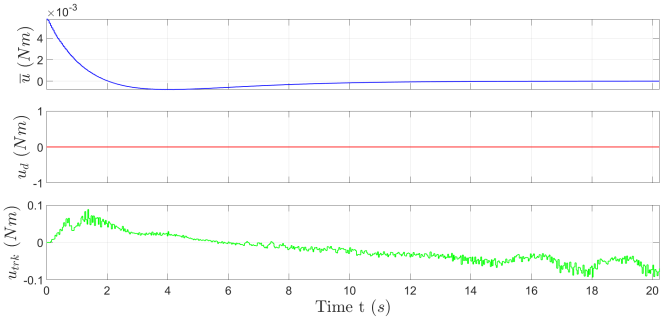


Figure 11.16: MIL Orient. EMC control input terms $\overline{u}, u_d$ and u_{trk} (from top to bottom) - No disturbance rejection

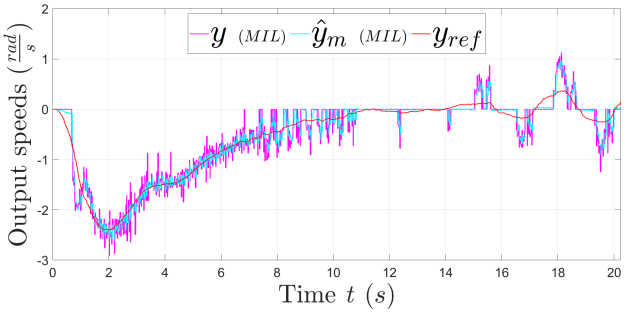


Figure 11.17: MIL Left motor EMC outputs y (magenta), $\hat{y}_m$ (cyan) and y_{ref} (red)

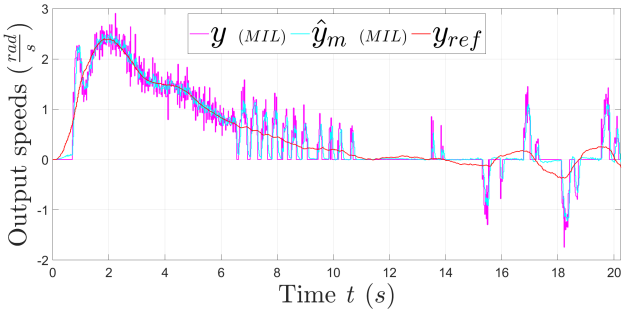


Figure 11.18: MIL Right motor EMC outputs y (magenta), $\hat{y}_m$ (cyan) and y_{ref} (red)

Comparison between MIL and RHIT:

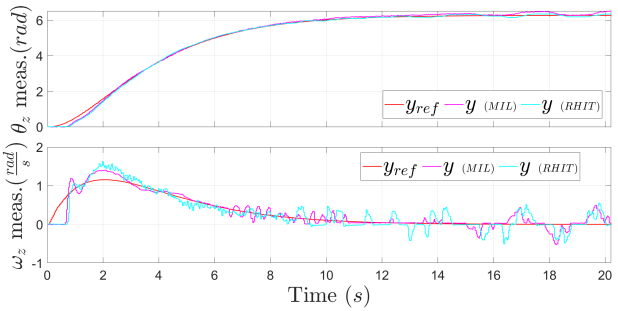


Figure 11.19: Orientation EMC measured outputs y (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

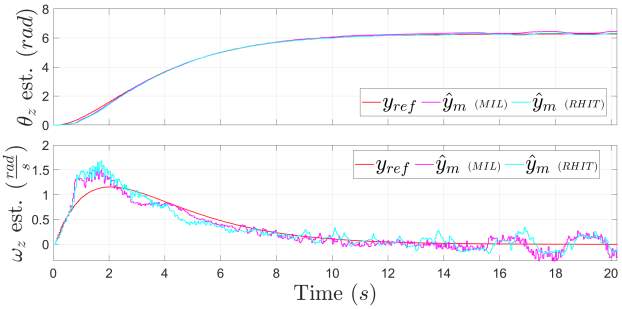


Figure 11.20: Orientation EMC estimated outputs $\hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

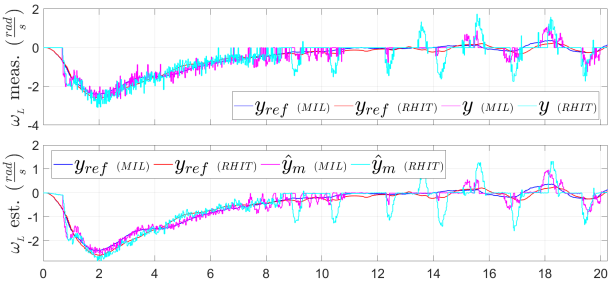


Figure 11.21: Left DC motor EMC measured outputs $y, \hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

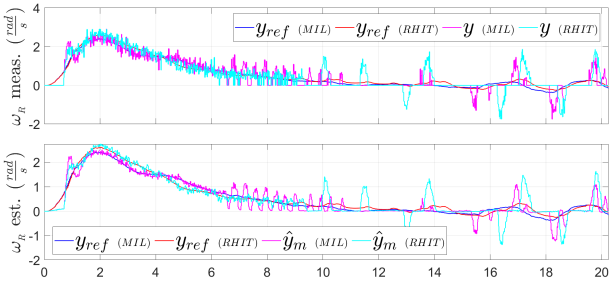


Figure 11.22: Right DC motor EMC measured outputs $y, \hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

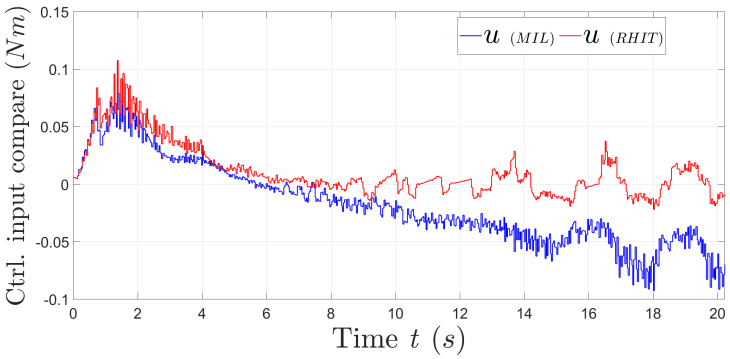


Figure 11.23: Orientation EMC virtual control input u - Comparison MIL and RHIT

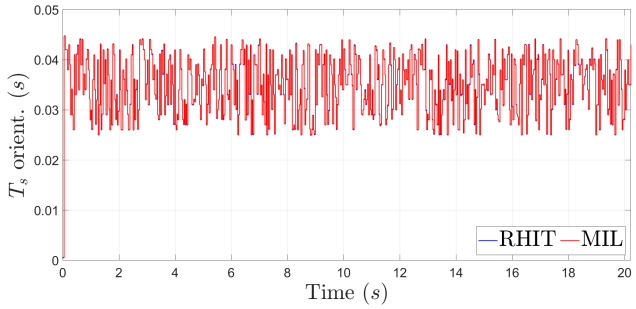


Figure 11.24: Orientation EMC Timestamp - Comparison MIL and RHIT

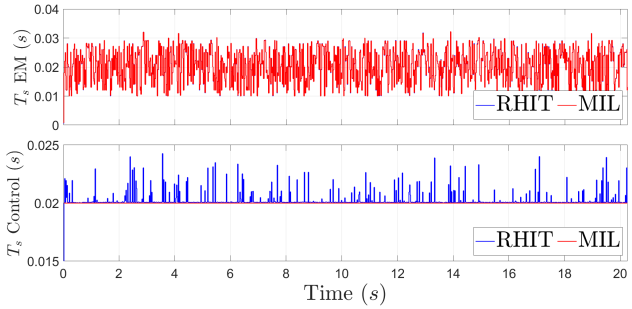


Figure 11.25: (Left and right) DC motors EMC Timestamp EM and Control part - Comparison MIL and RHIT

11.3.3 Target angle $\tilde{\theta}_z = -2\pi$ rad $(-360°)$, target long. speed $\tilde{v}_\xi = 0$ $\frac{\mathbf{m}}{\mathbf{s}}$, dist. rej.

MIL simulation test results:

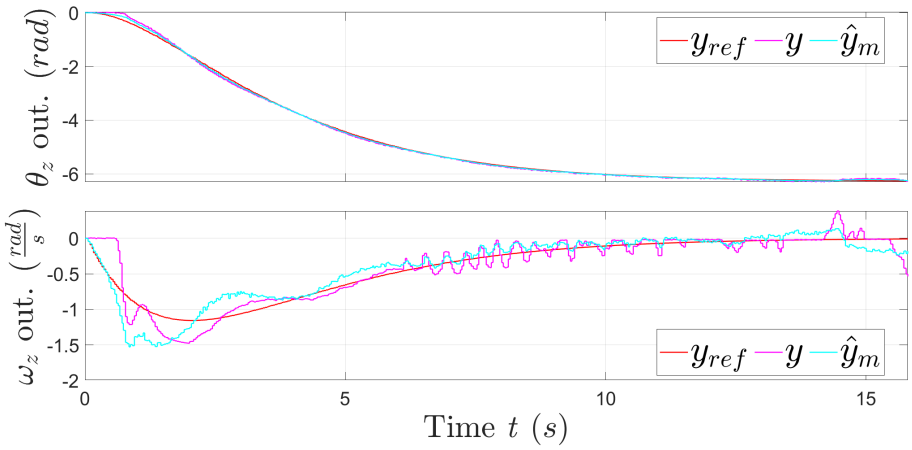


Figure 11.26: MIL Orientation EMC outputs y_{ref} (red), y (magenta) and $\hat{y}_m$ (cyan)

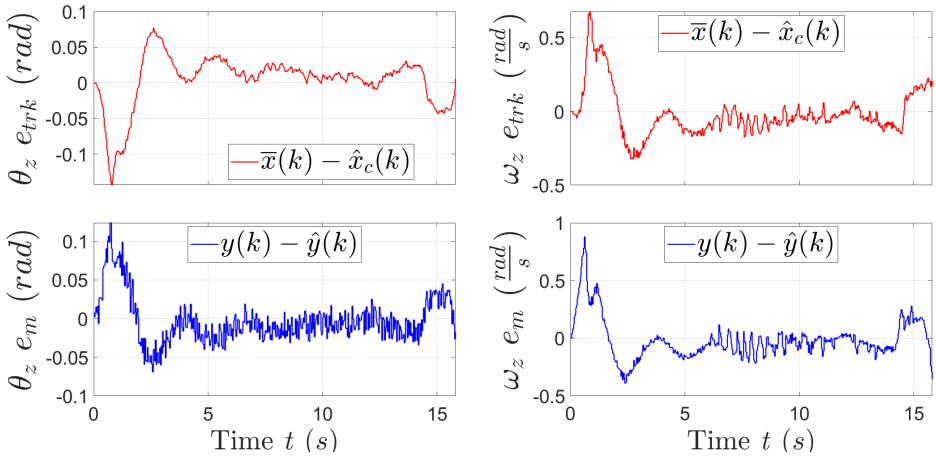


Figure 11.27: MIL Orientation EMC tracking e_{trk} and model e_m errors

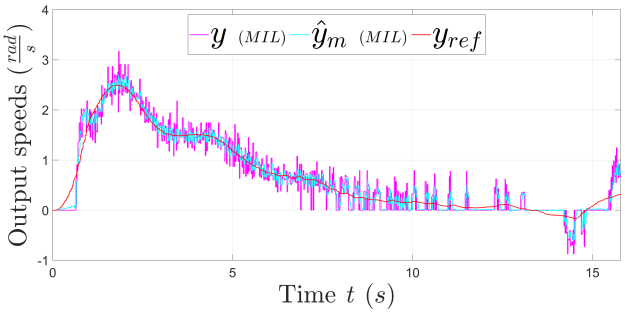


Figure 11.28: MIL Left motor EMC outputs y (magenta), $\hat{y}_m$ (cyan) and y_{ref} (red)

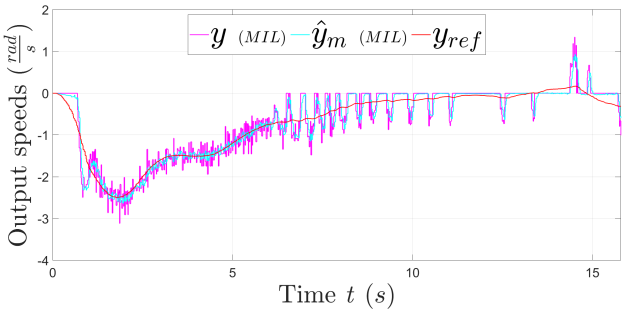


Figure 11.29: MIL Right motor EMC outputs y (magenta), $\hat{y}_m$ (cyan) and y_{ref} (red)

Comparison between MIL and RHIT:

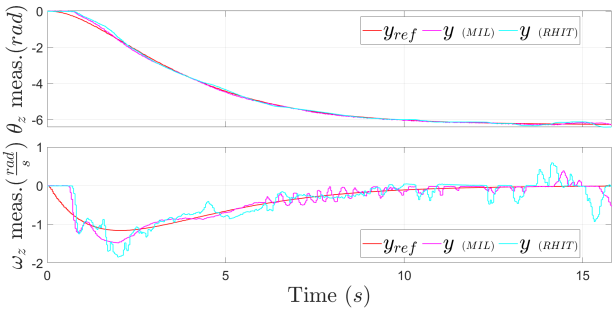


Figure 11.30: Orientation EMC measured outputs y (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

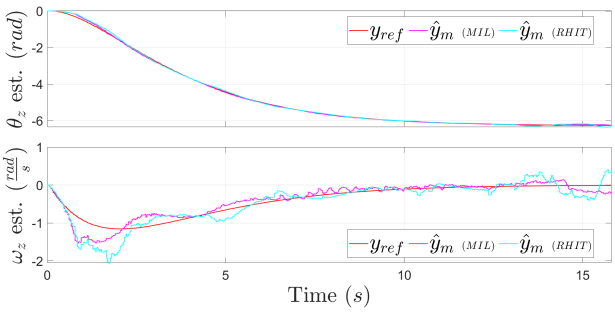


Figure 11.31: Orientation EMC estimated outputs $\hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

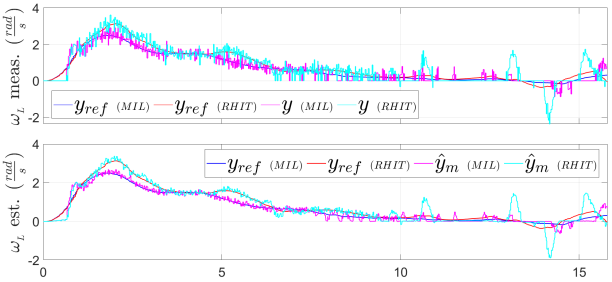


Figure 11.32: Left DC motor EMC measured outputs $y, \hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

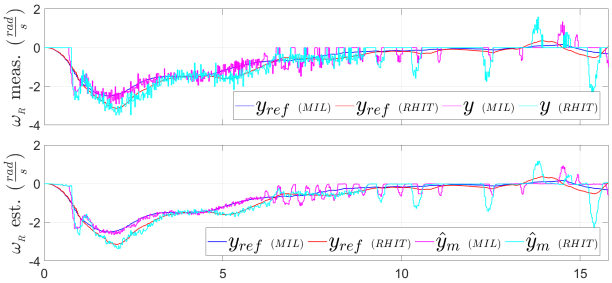


Figure 11.33: Right DC motor EMC measured outputs $y, \hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

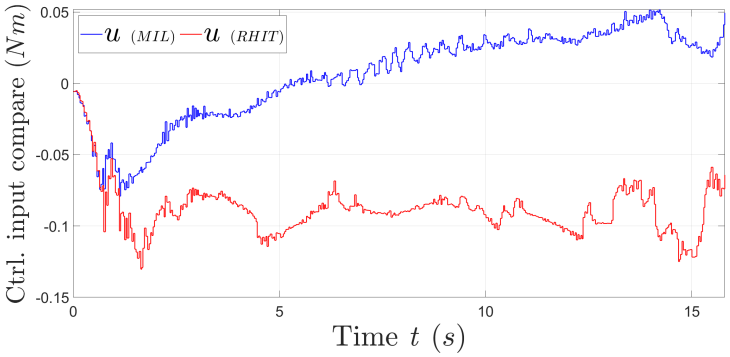


Figure 11.34: Orientation EMC virtual control input u - Comparison MIL and RHIT

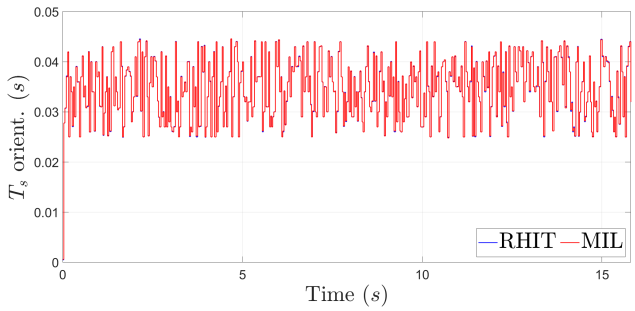


Figure 11.35: Orientation EMC Timestamp - Comparison MIL and RHIT

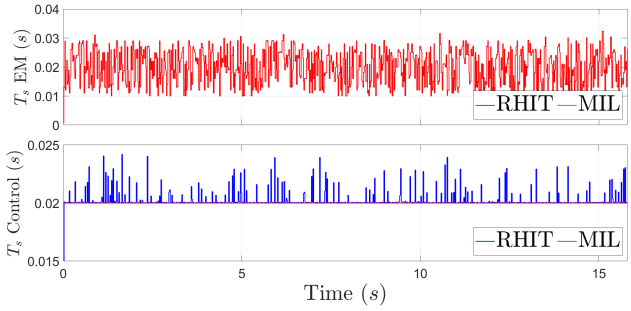


Figure 11.36: (Left and right) DC motors EMC Timestamp EM and Control part - Comparison MIL and RHIT

11.3.4 Target angle $\tilde{\theta}_z = 2\pi$ rad $(360°)$, target long. speed $\tilde{v}_\xi \neq 0$ $\frac{\mathbf{m}}{\mathbf{s}}$ $(r_m = 0.2\ m)$, dist. rej.

MIL simulation test results:

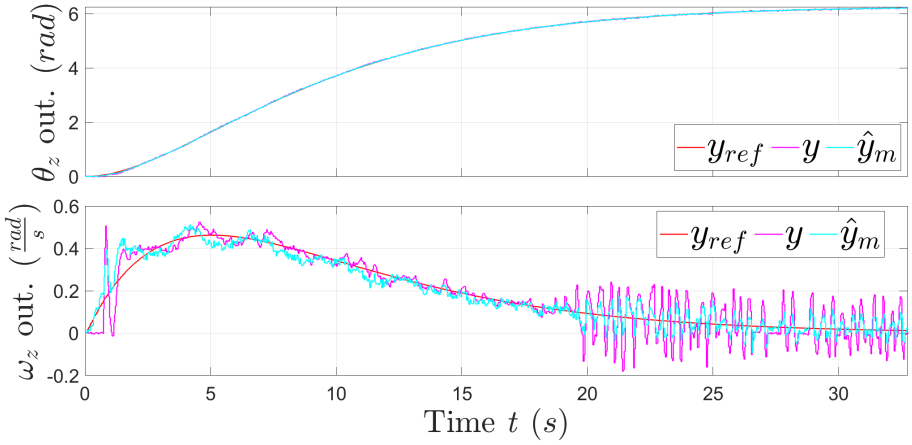


Figure 11.37: MIL Orientation EMC outputs y_{ref} (red), y (magenta) and $\hat{y}_m$ (cyan)

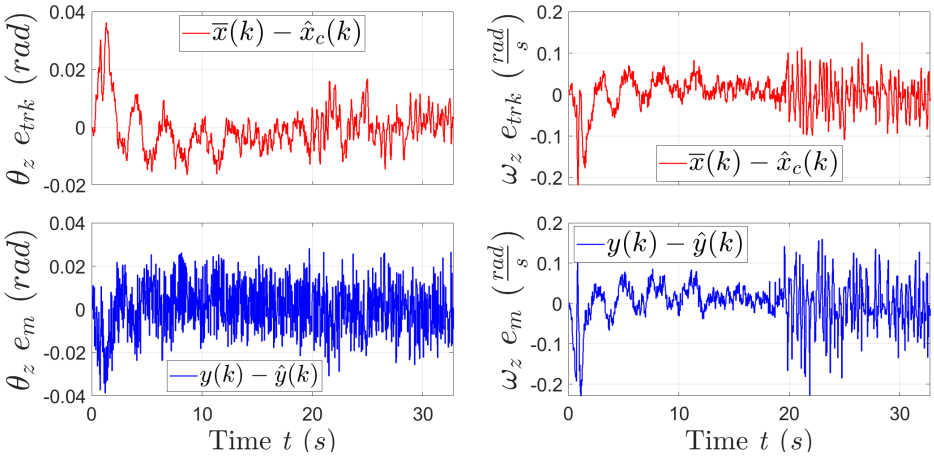


Figure 11.38: MIL Orientation EMC tracking e_{trk} and model e_m errors

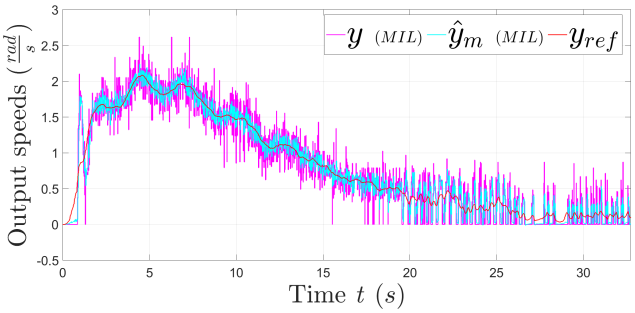


Figure 11.39: MIL Left motor EMC outputs y (magenta), $\hat{y}_m$ (cyan) and y_{ref} (red)

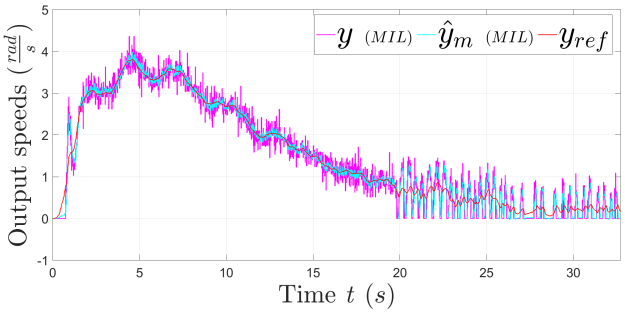


Figure 11.40: MIL Right motor EMC outputs y (magenta), $\hat{y}_m$ (cyan) and y_{ref} (red)

Comparison between MIL and RHIT:

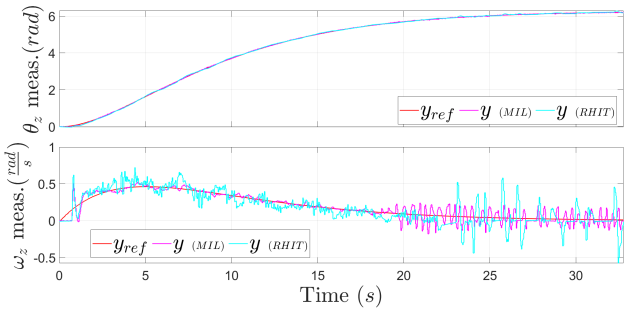


Figure 11.41: Orientation EMC measured outputs y (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

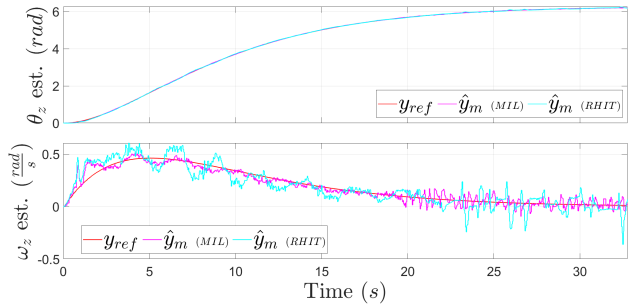


Figure 11.42: Orientation EMC estimated outputs $\hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

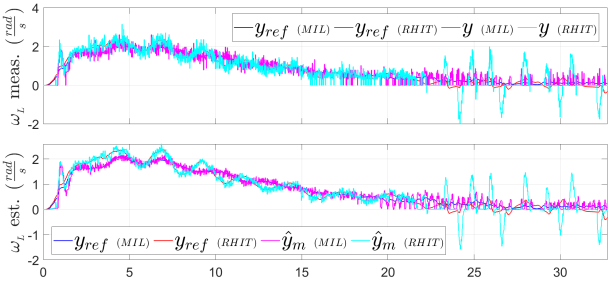


Figure 11.43: Left DC motor EMC measured outputs $y, \hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

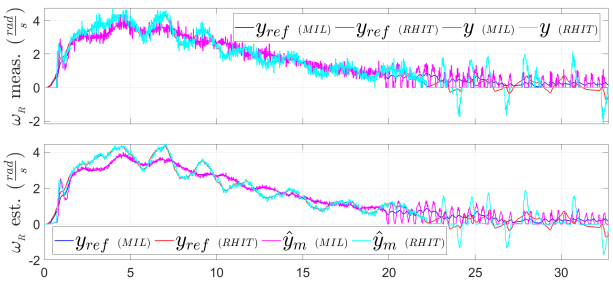


Figure 11.44: Right DC motor EMC measured outputs $y, \hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

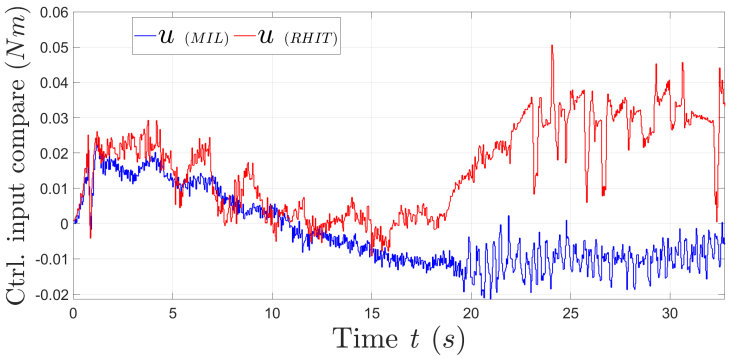


Figure 11.45: Orientation EMC virtual control input u - Comparison MIL and RHIT

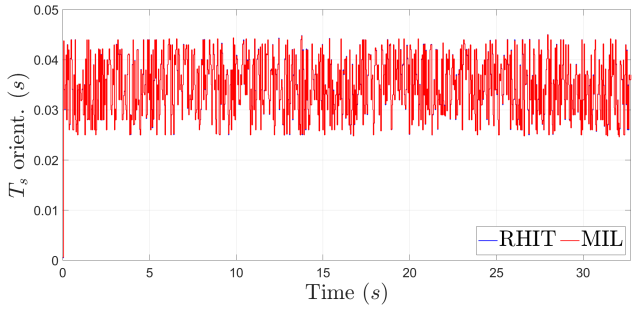


Figure 11.46: Orientation EMC Timestamp - Comparison MIL and RHIT

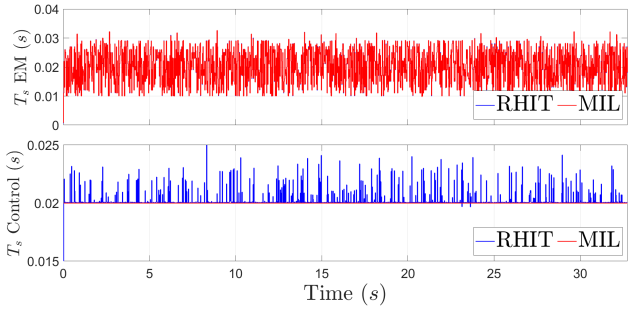


Figure 11.47: (Left and right) DC motors EMC Timestamp EM and Control part - Comparison MIL and RHIT

11.3.5 Target angle $\tilde{\theta}_z = -2\pi$ rad $(-360°)$, target long. speed $\tilde{v}_\xi \neq 0$ $\frac{\mathbf{m}}{\mathbf{s}}$ $(r_m = 0.2\ m)$, dist. rej.

MIL simulation test results:

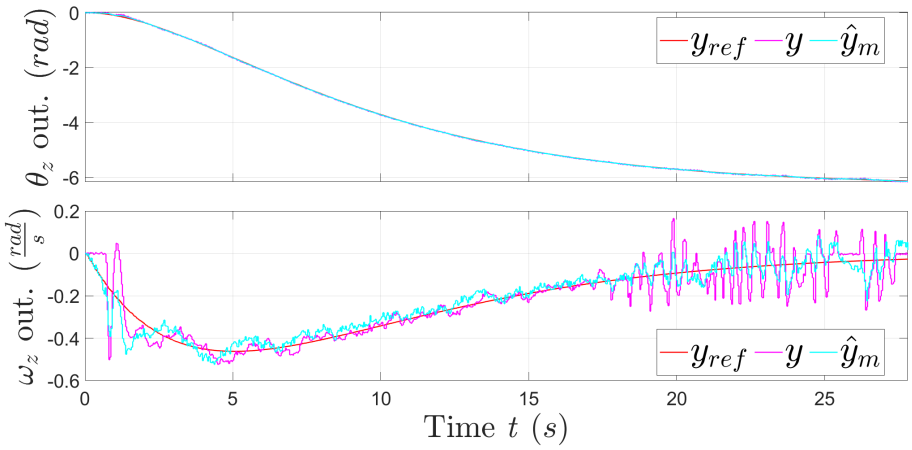


Figure 11.48: MIL Orientation EMC outputs y_{ref} (red), y (magenta) and $\hat{y}_m$ (cyan)

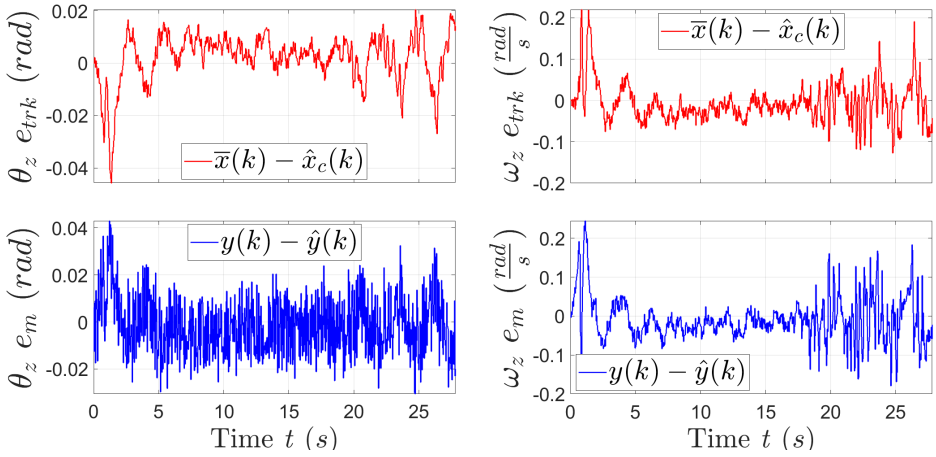


Figure 11.49: MIL Orientation EMC tracking e_{trk} and model e_m errors

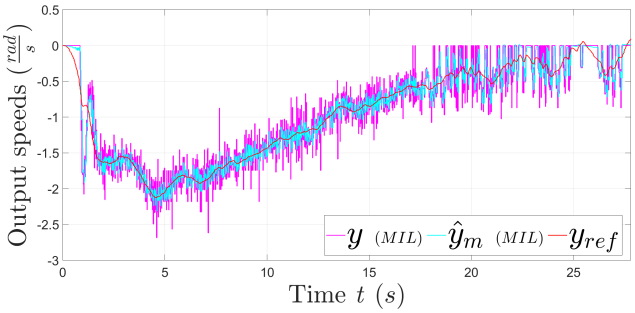


Figure 11.50: MIL Left motor EMC outputs y (magenta), $\hat{y}_m$ (cyan) and y_{ref} (red)

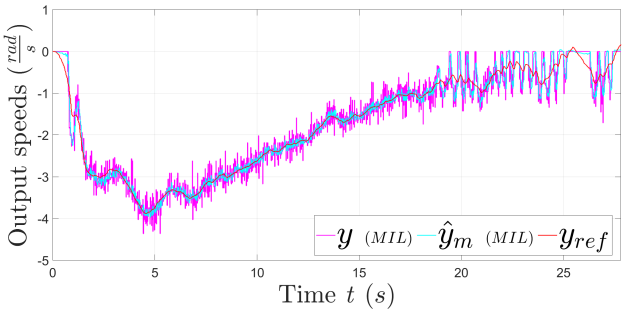


Figure 11.51: MIL Right motor EMC outputs y (magenta), $\hat{y}_m$ (cyan) and y_{ref} (red)

Comparison between MIL and RHIT:

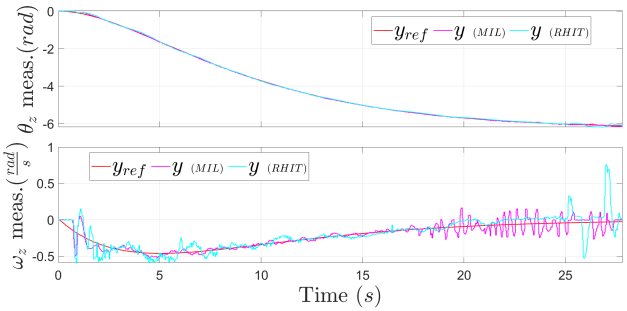


Figure 11.52: Orientation EMC measured outputs y (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

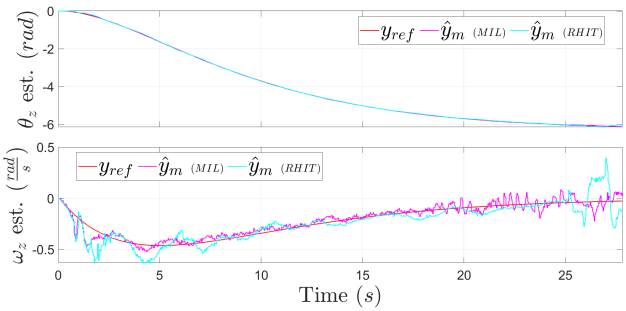


Figure 11.53: Orientation EMC estimated outputs $\hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

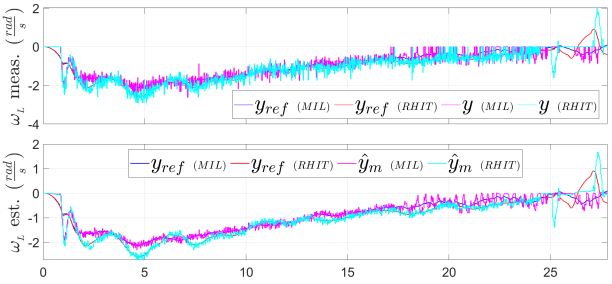


Figure 11.54: Left DC motor EMC measured outputs $y, \hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

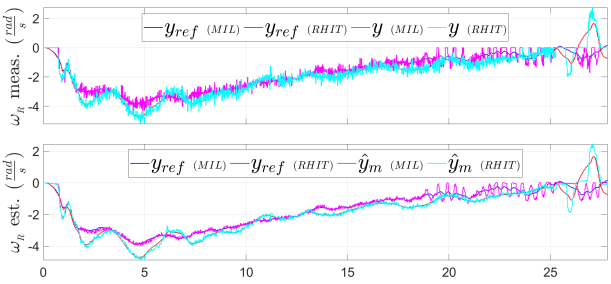


Figure 11.55: Right DC motor EMC measured outputs $y, \hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

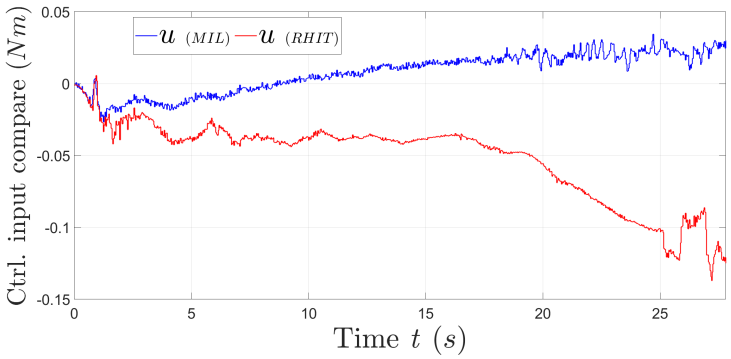


Figure 11.56: Orientation EMC virtual control input u - Comparison MIL and RHIT

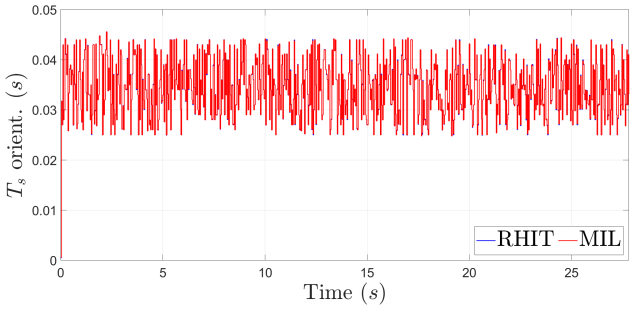


Figure 11.57: Orientation EMC Timestamp - Comparison MIL and RHIT

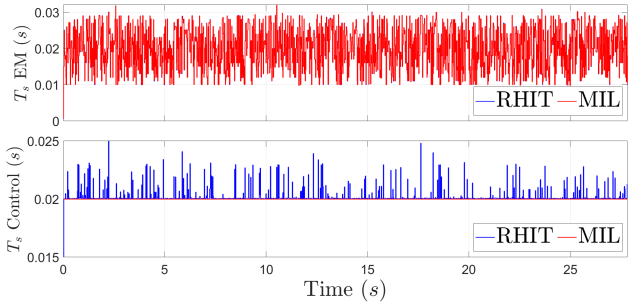


Figure 11.58: (Left and right) DC motors EMC Timestamp EM and Control part - Comparison MIL and RHIT

Chapter 12

Motors and orientation EMC - RHIT results

In this section, practical implementation of robot orientation EMC is considered.
3 possible orientation control models are selected depending on noise estimator structure used to estimate the controllable states θ_z angle and ω_z angular speed; theoretical details about them are explained in Section 10.2.4:

1. Dynamic feedback noise estimator.
2. Measure driven decomposition using static FB noise estimators only for both states
3. Measure driven decomposition using static FB noise estimator for θ_z and a dynamic one for ω_z.

A hierarchic structure is used to combine the orientation EMC and already built DC motors EMC. The scheme is slightly modified wrt the one used for MIL tests (Section 11.1.2) because fine models are not needed anymore. Indeed, as it can be seen in Figure 12.1, they are substituted by real robot plant, which accept the motor control input voltages as inputs (u_R, u_L) and gives all needed orientation and motor measurements to close the loop with EMCs.

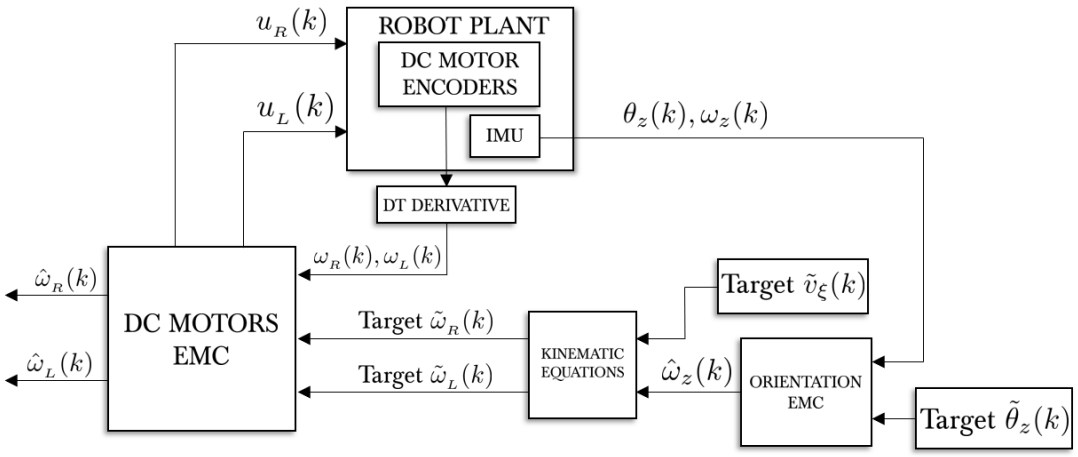


Figure 12.1: Hierarchic structure for RHIT - Orientation and DC motors EMC

The loop structure is briefly recalled below, making reference to the figure (for more detailed informations make also reference to Section 11.1.2):

- Starting from an input reference target $\tilde{\theta}_z$, from orientation control model, $\hat{\theta}_z$ and $\hat{\omega}_z$ are estimated.

- Conversion block allows to pass from estimated $\hat{\omega}_z$ to angular motor speeds. Conversion consists in kinematic equations. In this case the inputs are $\hat{\omega}_z$ and $\tilde{v}_\xi$ imposed equal to zero since we are controlling only the orientation. Outputs are the target motor wheel speeds $\tilde{\omega}_{_R}, \tilde{\omega}_{_L}$ entering DC motors EMC.

- From DC motors EMC control input voltages $u_{_R}, u_{_L}$ are the physical voltages to be applied to real motors, moving the robot. Measurement θ_z is then computed using IMU sensor fusion, angular speed measurement ω_z using IMU gyroscope. Instead the wheel angular positions $\omega_{_R}, \omega_{_L}$ are recovered by motor encoders.

- DC motors EMC give as output estimated wheels angular speeds. If these are not sufficient to obtain the desired orientation, there is an error between IMU measurements and estimated states in orientation EMC, the model tries to reduce it closing the loop.

12.1 General RHIT test settings

Practical RHIT tests must show if orientation EMC gives low model and tracking errors e_m and e_{trk} respectively, at least for $\hat{\theta}_z$ estimated state. Instead concerning $\hat{\omega}_z$ estimate is not crucial having very low errors, since we are interested mainly in final orientation angle and not in robot speed trajectory. Obviously too high errors are not allowed.
Main trajectories followed are:

- Robot CoM still (robot linear speed $\tilde{v}_\xi = 0\,\mathrm{m/s}$) and $\tilde{\theta}_z = [0, \pm 2\pi]\mathrm{rad}$ angle trajectory.

- Robot in movement following a circle trajectory (robot linear speed $\tilde{v}_\xi \neq 0\,\frac{\mathrm{m}}{\mathrm{s}}$ and mean radius $r_m = 0.2\,\mathrm{m}$), $\tilde{\theta}_z = [0, \pm 2\pi]\mathrm{rad}$ angle trajectory.

For every test the following plots are shown: for Orientation EMC estimated, measured and reference outputs related to θ_z and ω_z, measured and tracking errors e_m, e_{trk}, virtual torque control inputs u; for DC motors EMC only $y, y_{ref}, \hat{y}_m$ outputs. In some tests are also shown the robot trajectory (when robot is following a circular trajectory) and the virtual control input components u_d, u_{trk} (when comparing tests with and without disturbance rejection).
The robot angular speed ω_z can be measured by IMU gyroscope with enough accuracy, instead for angle θ_z measurement different solutions are possible:

- Use the motor encoders and convert the angular position of the wheels to robot angular speed ω_z (using the kinematic model), then make a discrete integration (directly using SW code) to obtain the angle.

- Use IMU and RTIMULib2 sensor fusion algorithm (RTQF or Kalman) combining ***only*** the gyroscope and accelerometer data.

- Use IMU and RTIMULib2 sensor fusion algorithm (RTQF or Kalman) combining ***all*** data from gyroscope, accelerometer and magnetometer

RTQF fusion algorithm shows better measurements results and it's selected instead of Kalman filter.
For orientation model the CT eigenvalues used in control, noise estimator and reference EMC blocks are explained in each test section. Instead concerning the EMC motors version 1.2 is used for both.

2 HW timers work for both the 2 motors (EM and control part) and only 1 for orientation control model.
In every test variable sample time is considered for both control models: in particular the already tested random sampling times $T_{EM} = [0.01, 0.03]$ s and $T_{ctrl} = 0.02\,\text{s}$ for EM and control EMC block parts are used.
Instead for orientation control model, variable sampling time is selected on the basis of IMU sampling rate: this is equal to $[47, 47.5]$ Hz, which means approximately $T_{IMU} = 0.021\,\text{s}$. A good choice for variable sampling time turned out to be $T_o = [0.025, 0.045]$ s, with minimum value $0.025 > 0.02$ s. In the next figures are presented the Timestamp ranges used in all tests (some differences occur among tests, but the range remains still the same):

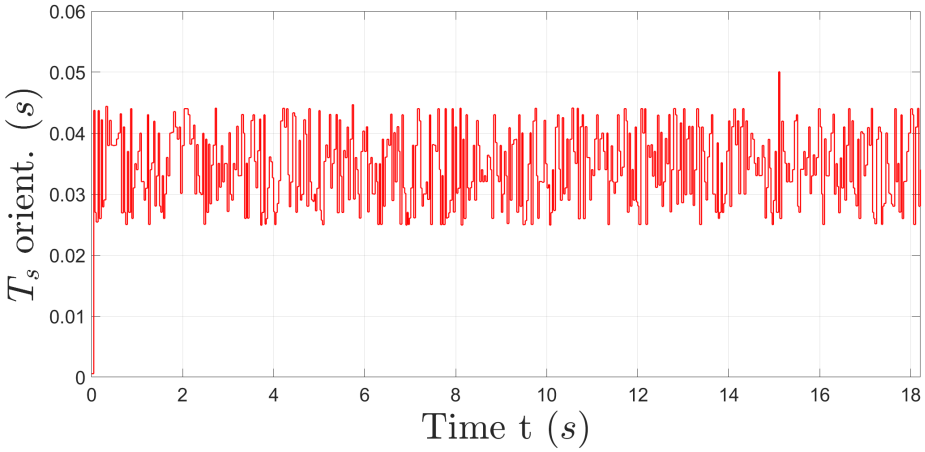


Figure 12.2: Orientation EMC Timestamp

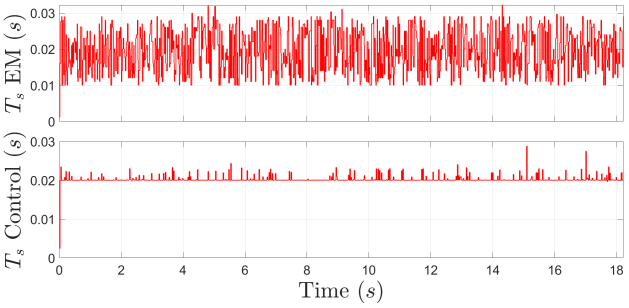


Figure 12.3: Left motor EMC Timestamp

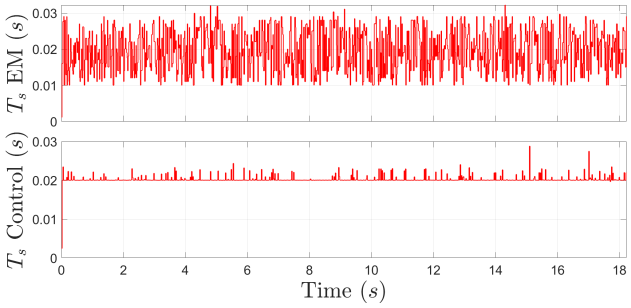


Figure 12.4: Right motor EMC Timestamp

In Figure 12.3 and Figure 12.4, Timestamp of control block parts for both the motors needs to be maintained fixed at $0.02\,\text{s}$ but some spikes are present: this can be derived from the presence of 3 different timers running simultaneously in SW code (2 for the both the motors and 1 for orientation part) and also by implementation issues. This is not a problem since the DC motors EMC work well anyway.
Some SW implementation problems occurred when making these tests. First IMU fusion algorithm computes the robot angle within the limited range $[-\pi, \pi]$, instead the control model is built allowing $\theta_z \in \mathbb{R}$: so estimated angle state does not follow the measurement when $|\theta_z| > \pi$. So the IMU measured angle after fusion must be *unwrapped*: the domain for θ_z must be $\mathbb{R}$. Matworks Simulink program gives a build-in block to do this operation (unwrap block), using code generation it can be easily implemented in the SW code.
Second, when using magnetometer in sensor fusion algorithm, the computed angle value is dependent on the earth magnetic orientation: for construction properties in this case the magnetometer north is the earth magnetic south. Hence if for example robot is pointing north the computed angle is $\theta_z = 2\pi\,\text{rad}$. This starting offset creates unexpected results since the control model always start with $\theta_z = 0\,\text{rad}$. For this reasons a SW code is implemented to compute the offset and remove it at run-time from the computed angle.

12.2 Short summary on RHIT results

The best noise estimator model for orientation EMC is the one with 2 static FB estimators combined together following measure-driven decomposition. In general this solution lead always to very low tracking and model errors for orientation angle θ_z and angular speed ω_z.
As it will be seen in the corresponding sections the other methods, dynamic FB and measure-driven decomposition using static plus dynamic FB noise estimators are not suitable.
Using static FB measure-driven decomposition noise estimators, different angle θ_z measurement techniques are compared: IMU sensor fusion with and without magnetometer, motors encoder. The best solution results to be IMU with magnetometer sensor fusion.
Furthermore, DC motors dead zones creates some oscillations when robot approaches the target orientation, the problem may be easily solved by stopping SW code when a certain error bound is reached for the first time. However the code is let running intentionally even the target is almost reached: indeed in some tests with a circle trajectory when IMU without magnetometer in sensor fusion is not used (Section 12.4.1), orientation target angle is never reached, because IMU angle is wrong.
A test with orientation EMC without disturbance rejection is performed, making a circular trajectory when robot is still, with IMU angle measurement using magnetometer sensor fusion: the errors difference wrt using disturbance rejection is minimal, this is because the only physical parameter for orientation is the robot total inertia I_T, and the one of EMC is very near to reality, leading to small noises and disturbances (the most depending on asynchronous sampling time). This is not like DC motors EMC where many neglected dynamics and disturbances are present.
Other specific results are discussed in corresponding next sections.

12.3 RHIT tests - Dynamic FB noise estimator

For first tests only dynamic FB noise estimator is considered to estimate θ_z controllable state, since it is supposed to be the only one measured. ω_z estimate can be recovered consequently, since its dependence by integration on the angle.
The IMU is used to measure θ_z, with presence of magnetometer data in fusion algorithm.
Only a turn in place trajectory of $[0, 2\pi]$ rad test is sufficient to understand that this is not a suitable model for controlling orientation. Indeed even if for angular speed appears quite good tracking and model errors, the ones of angle are not acceptable: for example, seeing the test in top plot of Figure 12.5 the estimate $\hat{y}_m$ is in the middle of reference y_{ref} and measurement y, with high difference. It means that the orientation EMC tries without success to reduce both model error e_m and tracking error e_{trk}.
Max torque allowed is $\tau_{max} = 1\,\mathrm{N\,m}$.
For orientation control model the following CT eigenvalues are selected:

Eigenvalues type	CT eigenvalues
Reference μ_R	-0.2×2
Noise estimator μ_N	-2.5×4
Control μ_K	$-1, -17$

Table 12.1: CT eigenvalues Orientation EMC

12.3.1 Plot results

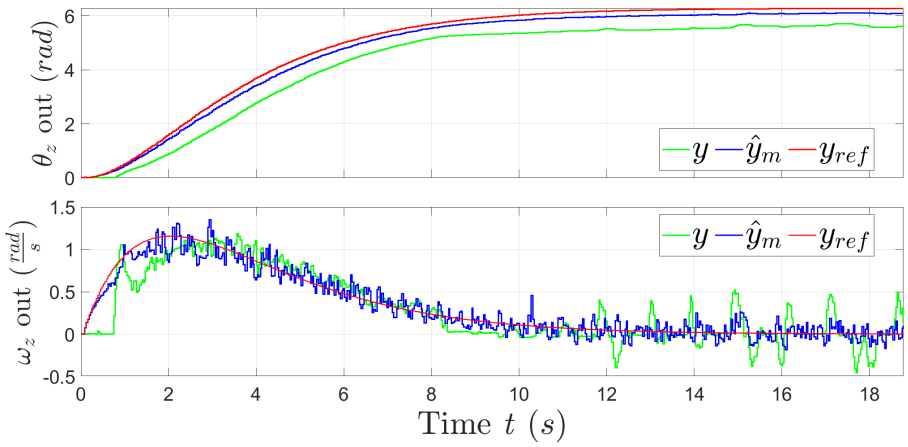


Figure 12.5: Orientation EMC y, $\hat{y}_m$ and y_{ref}

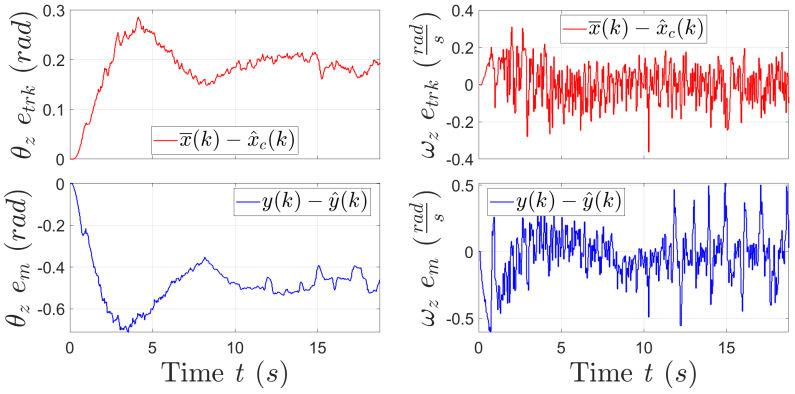


Figure 12.6: Tracking e_{trk} and model e_m errors for θ_z (1st column) and ω_z (2nd column)

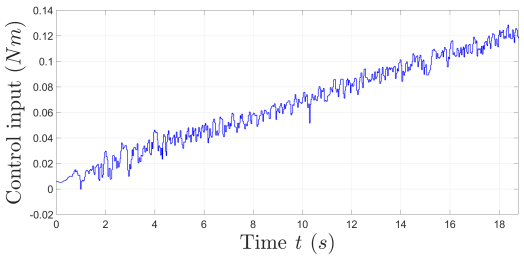


Figure 12.7: Robot torque control input (virtual)

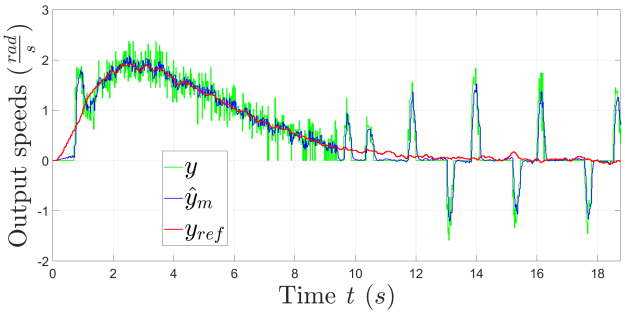


Figure 12.8: Right motor EMC y, $\hat{y}_m$ and y_{ref}

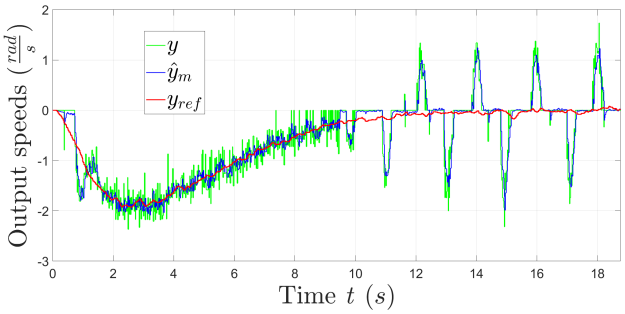


Figure 12.9: Left motor EMC y, $\hat{y}_m$ and y_{ref}

12.4 RHIT tests - Static feedback measure driven decomposition noise estimator

From practical tests results the best method is the one with noise estimator with 2 static FB. The tracking and model errors are the lowest for both controllable states and outputs.
The CT eigenvalues when robot is turning around in place from 0 to $2\pi\,\mathrm{rad}$ are:

Eigenvalues type	CT eigenvalues
Reference μ_R	-0.5×2
1^{st} noise estimator (static) $\mu_{N,1}$	-6
2^{nd} noise estimator (static) $\mu_{N,2}$	-6×2
Control μ_K	$-30, -10$

Table 12.2: CT eigenvalues Orientation EMC - Robot turning on the spot

Instead with a circle trajectory from 0 to $2\pi\,\mathrm{rad}$ and mean radius $r_m = 0.2\,\mathrm{m}$, the CT eigenvalues are:

Eigenvalues type	CT eigenvalues
Reference μ_R	-0.2×2
1^{st} noise estimator (static) $\mu_{N,1}$	-6
2^{nd} noise estimator (static) $\mu_{N,2}$	-6×2
Control μ_K	$-30, -10$

Table 12.3: CT eigenvalues Orientation EMC - Robot following a circle trajectory

In the last case reference eigenvalues are slower, since with more fast values the DC motors can easily reach voltage saturation ruining the result tests.
Virtual control input torque is in any case saturated when it reaches $1\,\mathrm{N\,m}$.

12.4.1 IMU θ_z measurement, NO magnetometer in sensor fusion

Using only accelerometer and gyroscope in θ_z measurement sensor fusion and when robot is turning around without linear velocity, the orientation is controlled in a quite precise way (first 2 tests). Instead with a circle trajectory with linear velocity $\neq 0$, IMU angle measurement from fusion algorithm becomes inconsistent, as it can be seen in Figure 12.22: the angle computed using the encoder overcome $8\,\mathrm{rad}$ which represent the effective orientation, instead IMU angle is $\approx 6.2\,\mathrm{rad}$ (more or less 360°) which is wrong. So also the green curve representing θ_z measurement on top of Figure 12.24 is wrong and misleading.
In practice the robot overcome 2π angle and continue moving for more than a quarter of a circle. This can be due to wrong IMU sensor fusion outcomes when the angular speed gyroscope information is very low (i.e. when robot is approaching the target angle).
For this reasons, best choice is to add magnetometer information to sensor fusion, as it can be seen in next section results.

Robot still, $[0, 2\pi]$ angle trajectory

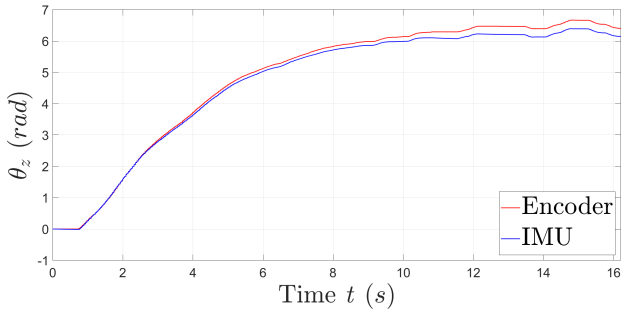


Figure 12.10: Encoder VS IMU θ_z measurement

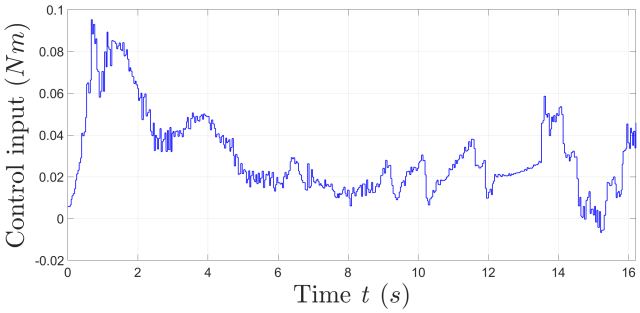


Figure 12.11: Robot torque control input (virtual)

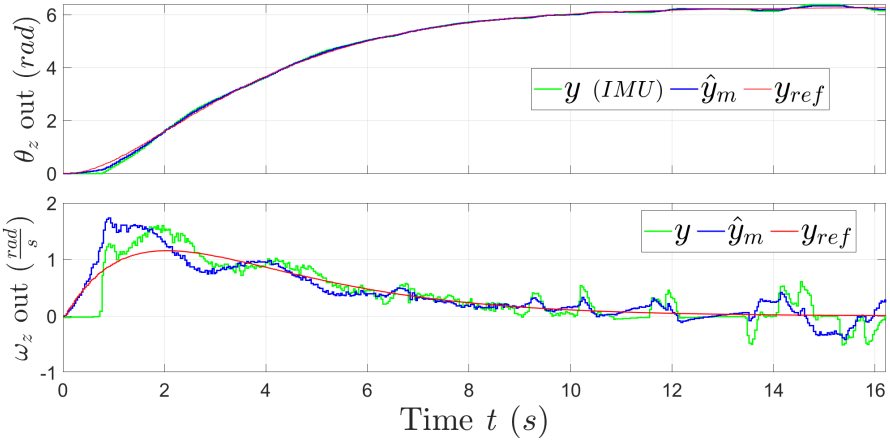


Figure 12.12: Orientation EMC y, $\hat{y}_m$ and y_{ref}

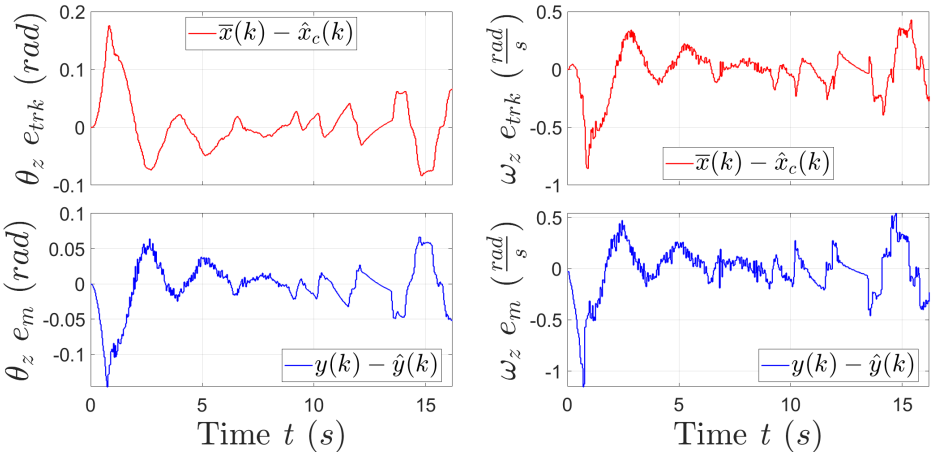


Figure 12.13: Tracking e_{trk} and model e_m errors for θ_z (1st column) and ω_z (2nd column)

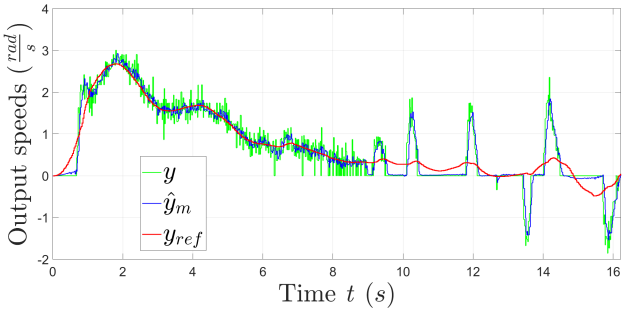


Figure 12.14: Right motor EMC y, $\hat{y}_m$ and y_{ref}

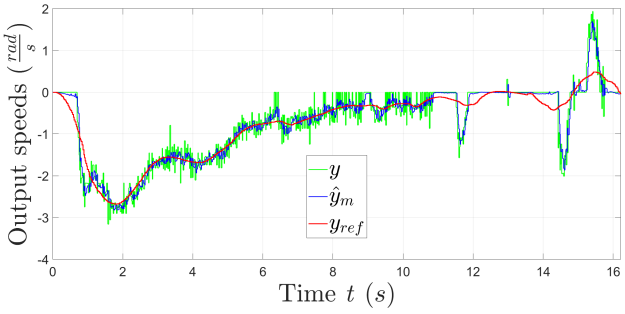


Figure 12.15: Left motor EMC y, $\hat{y}_m$ and y_{ref}

Robot still, $[0, -2\pi]$ angle trajectory

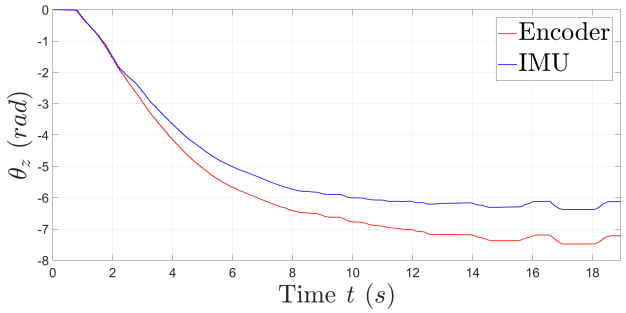


Figure 12.16: Encoder VS IMU θ_z measurement

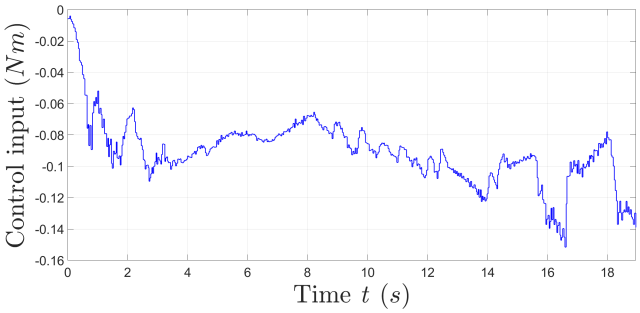


Figure 12.17: Robot torque control input (virtual)

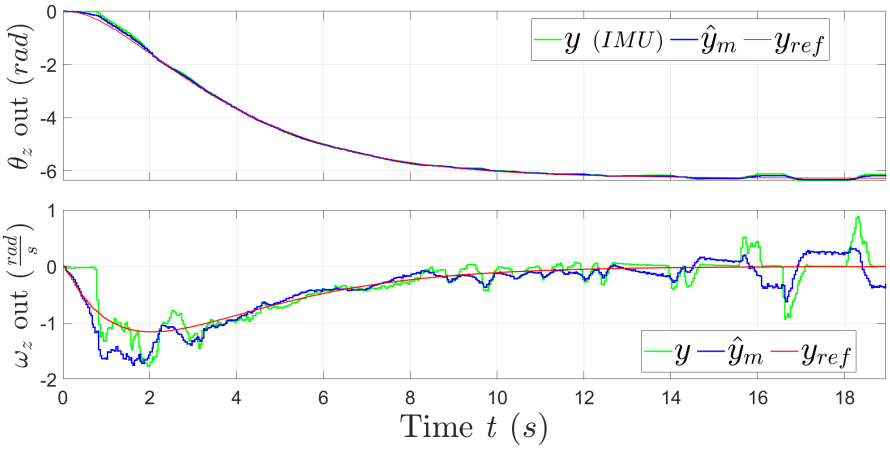


Figure 12.18: Orientation EMC y, $\hat{y}_m$ and y_{ref}

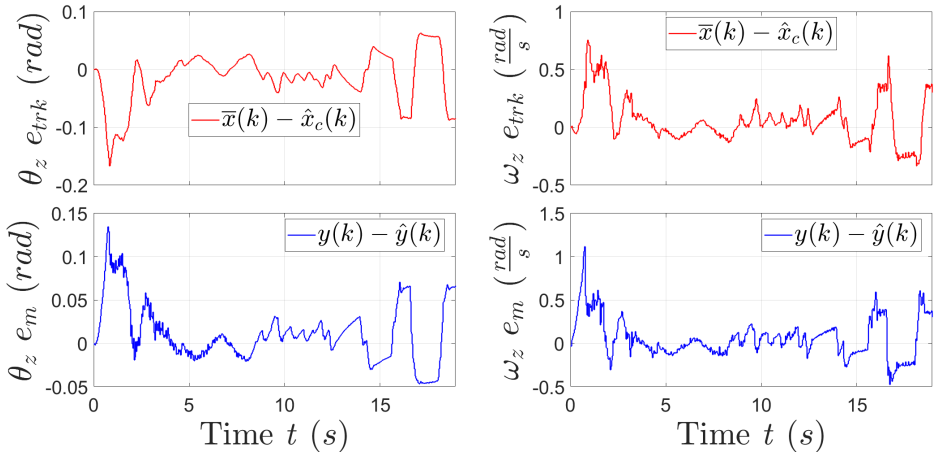


Figure 12.19: Tracking e_{trk} and model e_m errors for θ_z (1st column) and ω_z (2nd column)

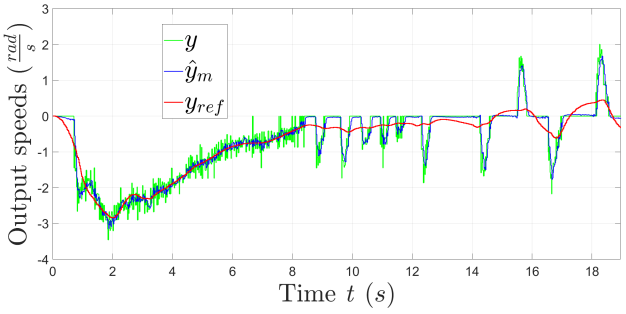


Figure 12.20: Right motor EMC y, $\hat{y}_m$ and y_{ref}

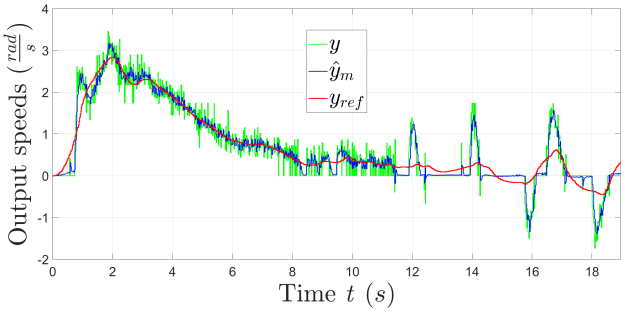


Figure 12.21: Left motor EMC y, $\hat{y}_m$ and y_{ref}

Robot in movement, circle r_m= 0.2 m, $[0, 2\pi]$ angle trajectory

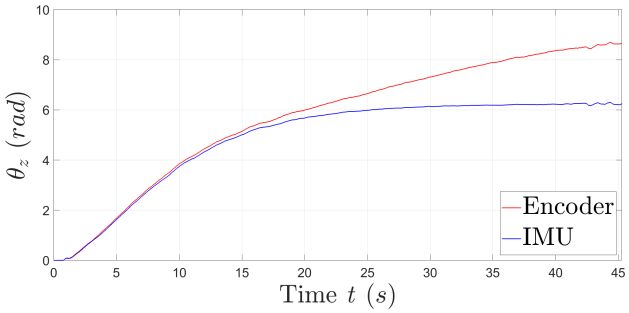


Figure 12.22: Encoder VS IMU θ_z measurement

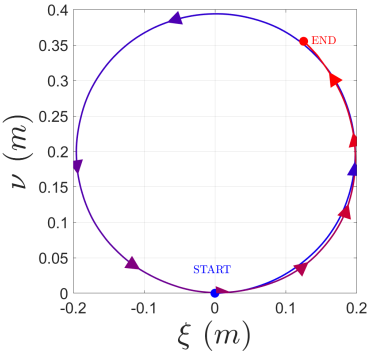


Figure 12.23: Robot position (using motor encoders and kin. eqs)

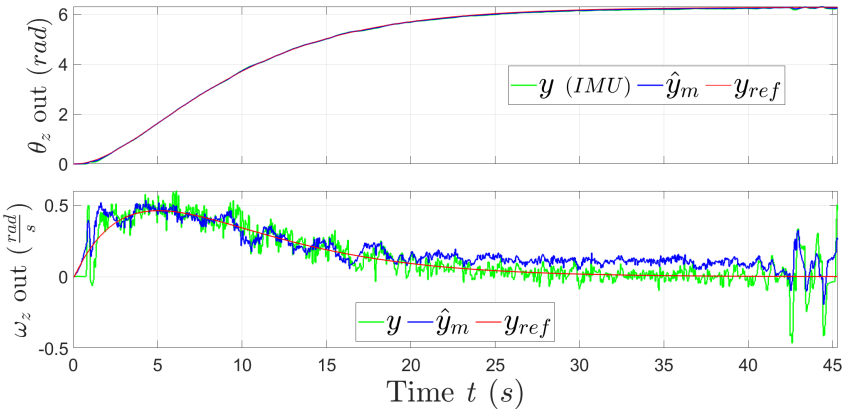


Figure 12.24: Orientation EMC y, $\hat{y}_m$ and y_{ref}

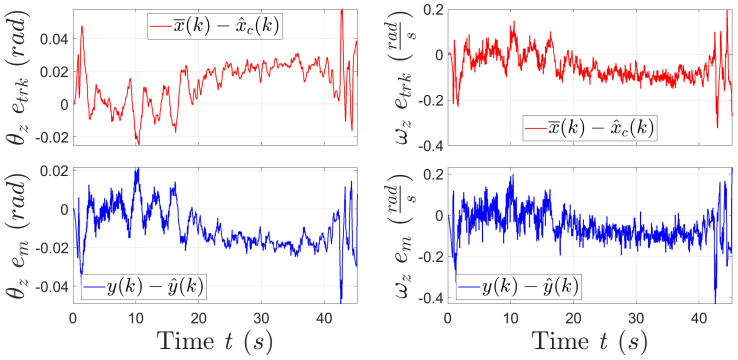


Figure 12.25: Tracking e_{trk} and model e_m errors for θ_z (1st column) and ω_z (2nd column)

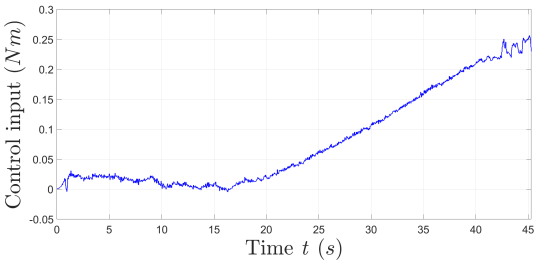


Figure 12.26: Robot torque control input (virtual)

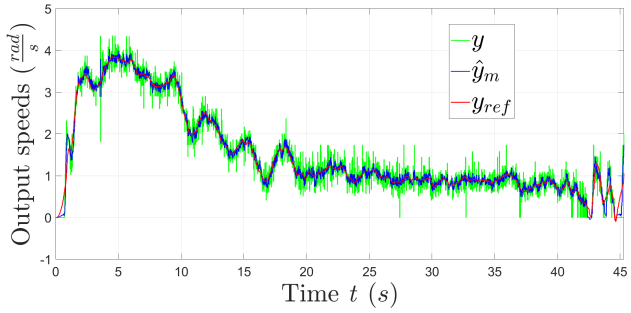


Figure 12.27: Right motor EMC y, $\hat{y}_m$ and y_{ref}

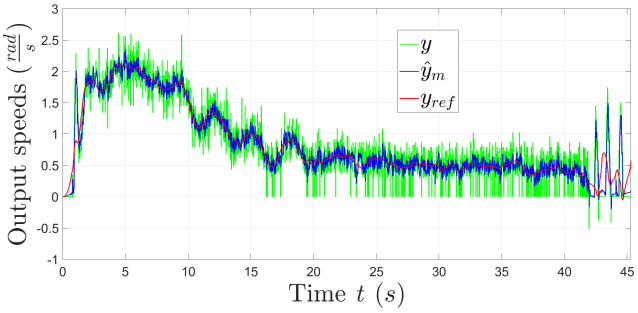


Figure 12.28: Left motor EMC y, $\hat{y}_m$ and y_{ref}

12.4.2 IMU θ_z measurement, magnetometer in sensor fusion

In addition to gyroscope and accelerometer data, also magnetometer information is added to IMU sensor fusion algorithm, in order to avoid inconsistent angle measurements when gyroscope angular acceleration is low.
However now the angle does not start anymore from $0\,\mathrm{rad}$ at every robot run-up, but it depends on robot orientation with respect to earth magnetic field: for example when robot is pointing earth magnetic north, IMU starting angle will be $\pi\,\mathrm{rad}$. As already explained this problem can be eluded by adding a piece of SW code removing this angle offset at every test start.
In all test cases tracking and model errors are very low, confirming that this is the best solution. No more measurement problems arises when robot approaches the target angle (like IMU without magnetometer case).
The first 2 results sets compare orientation EMC with and without disturbance rejection: orientation tracking and model errors are almost the same, and the fact that $u_d = 0\,\mathrm{N\,m}$ is compensated by an increase of u_{trk} values, which highest peak passes from 0.05 to $0.1\,\mathrm{N\,m}$ (which difference is almost the max peak of u_d when disturbance rejection is present), check Figure 12.30 and Figure 12.37.

Robot still, $[0, 2\pi]$ angle trajectory - Disturbance rejection

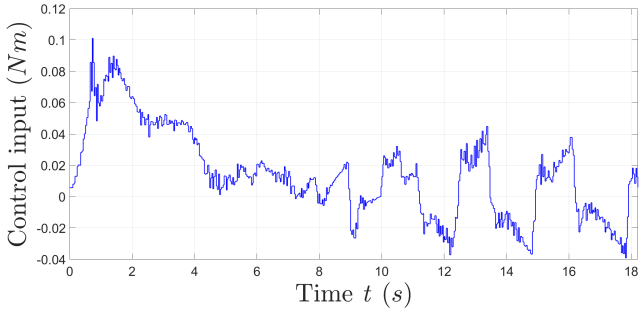


Figure 12.29: Robot torque control input (virtual)

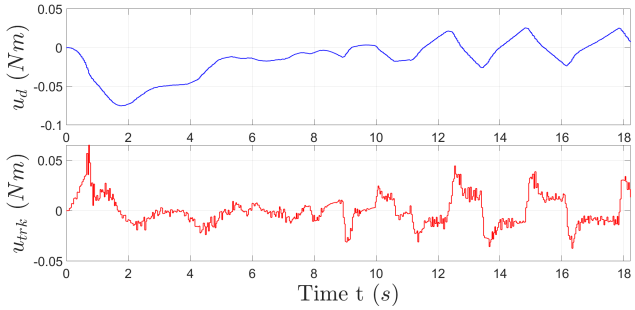


Figure 12.30: Control input components u_d and u_{trk}

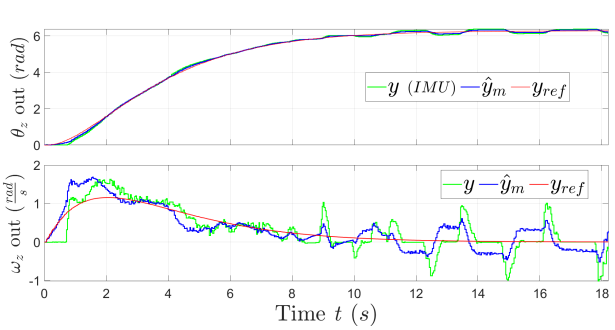


Figure 12.31: Orientation EMC y, $\hat{y}_m$ and y_{ref}

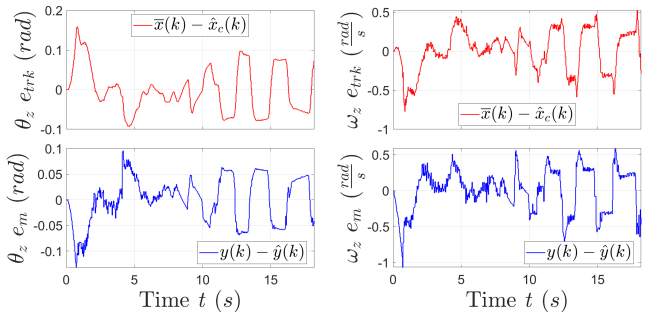


Figure 12.32: Tracking e_{trk} and model e_m errors for θ_z (1^{st} column) and ω_z (2^{nd} column)

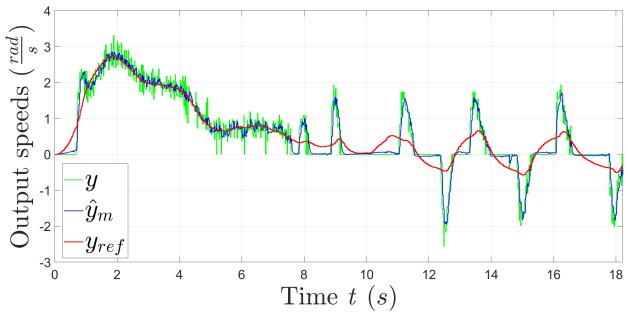


Figure 12.33: Right motor EMC y, $\hat{y}_m$ and y_{ref}

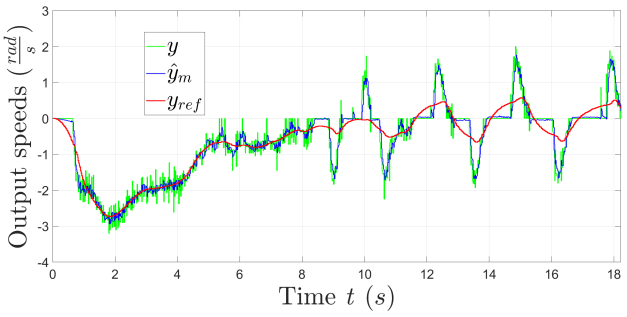


Figure 12.34: Left motor EMC y, $\hat{y}_m$ and y_{ref}

Robot still, $[0, 2\pi]$ angle trajectory - No disturbance rejection

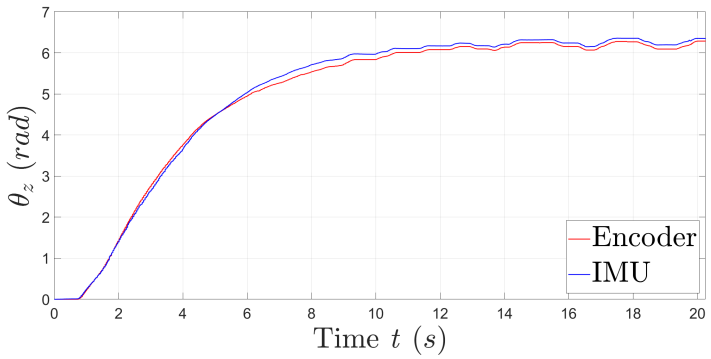


Figure 12.35: Encoder VS IMU θ_z measurement

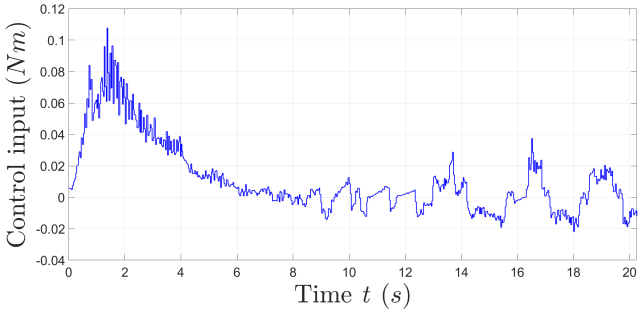


Figure 12.36: Robot torque control input (virtual)

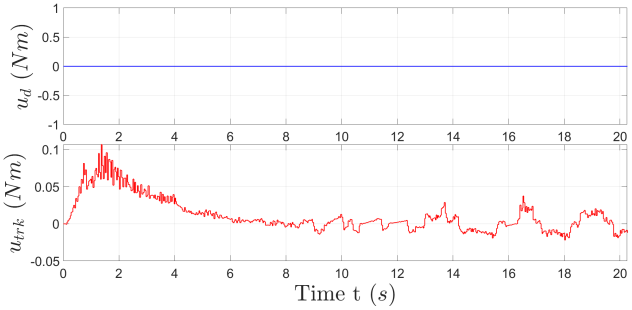


Figure 12.37: Control input components u_d and u_{trk}

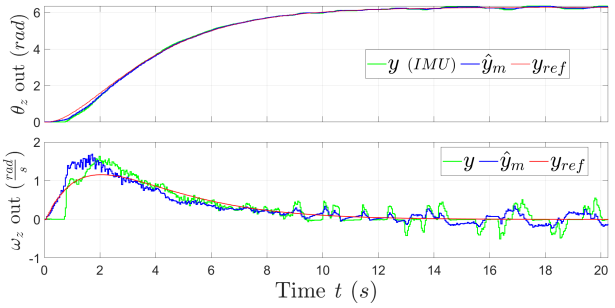


Figure 12.38: Orientation EMC y, $\hat{y}_m$ and y_{ref}

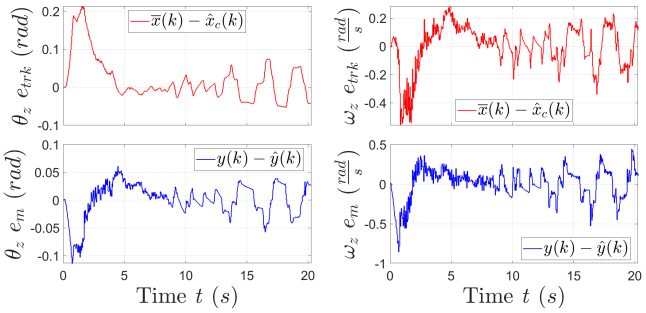


Figure 12.39: Tracking e_{trk} and model e_m errors for θ_z (1st column) and ω_z (2nd column)

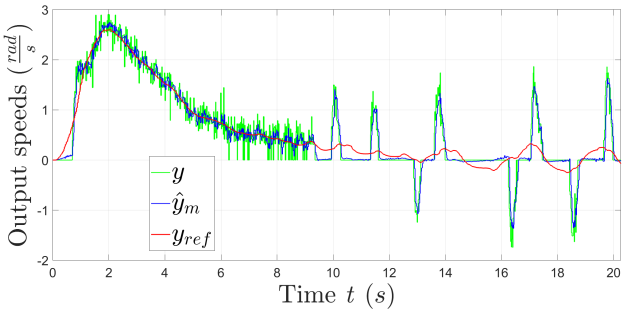


Figure 12.40: Right motor EMC y, $\hat{y}_m$ and y_{ref}

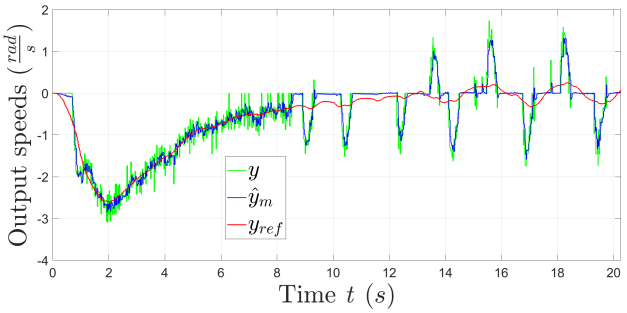


Figure 12.41: Left motor EMC y, $\hat{y}_m$ and y_{ref}

Robot still, $[0, -2\pi]$ angle trajectory

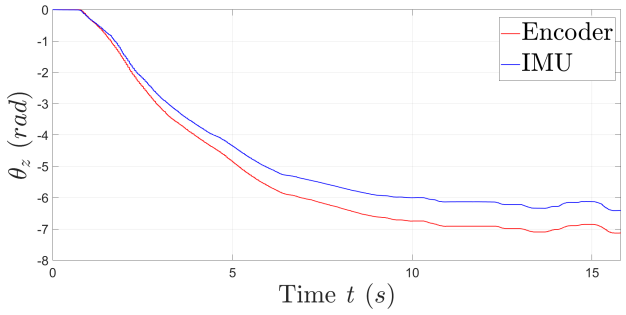


Figure 12.42: Encoder VS IMU θ_z measurement

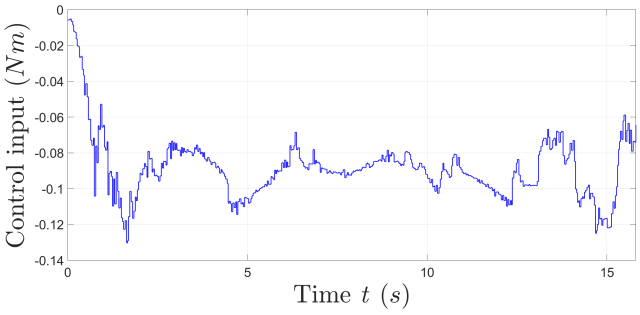


Figure 12.43: Robot torque control input (virtual)

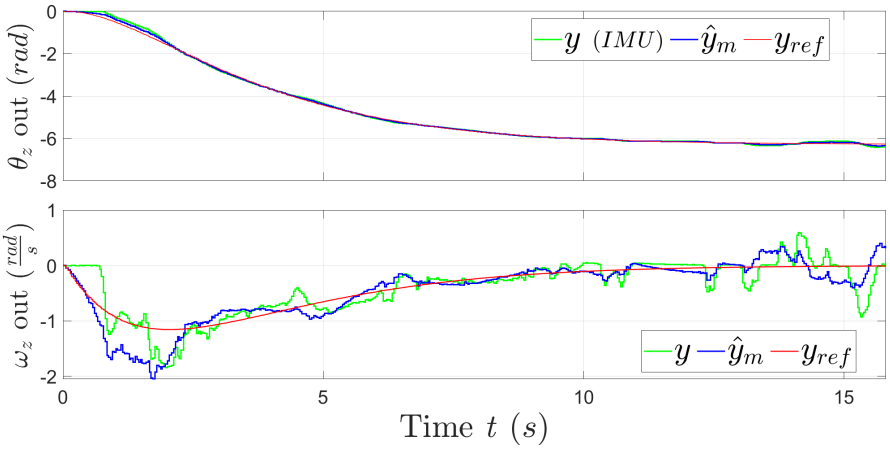


Figure 12.44: Orientation EMC y, $\hat{y}_m$ and y_{ref}

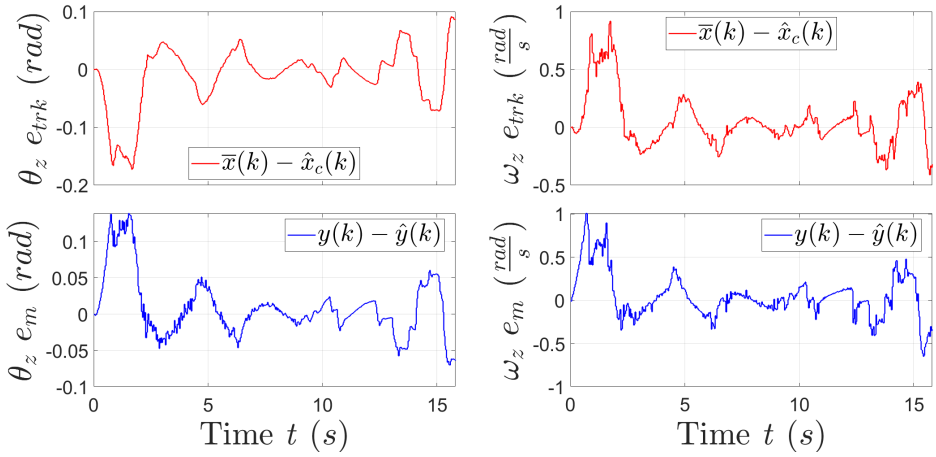


Figure 12.45: Tracking e_{trk} and model e_m errors for θ_z (1st column) and ω_z (2nd column)

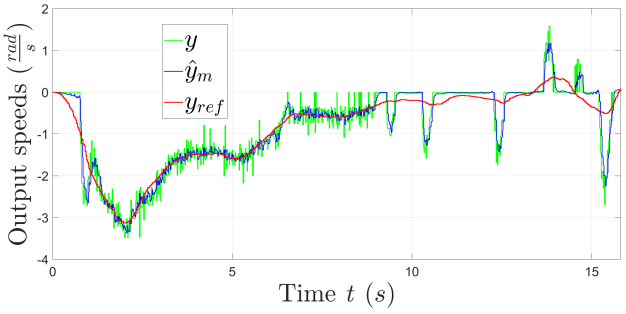


Figure 12.46: Right motor EMC y, $\hat{y}_m$ and y_{ref}

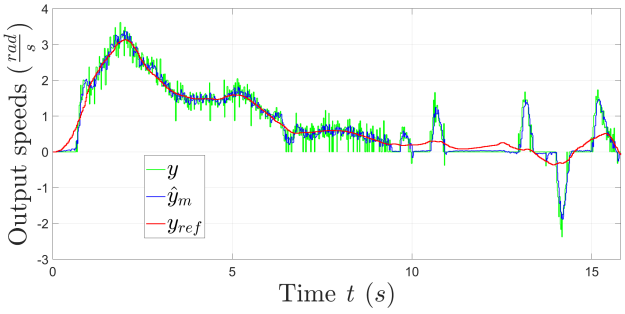


Figure 12.47: Left motor EMC y, $\hat{y}_m$ and y_{ref}

Robot in movement in circle r_m = 0.2 m, $[0, 2\pi]$ angle trajectory

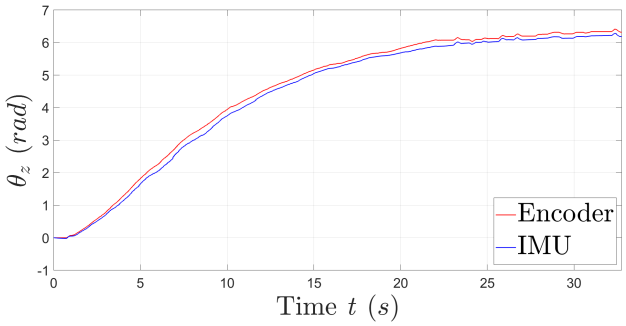


Figure 12.48: Encoder VS IMU θ_z measurement

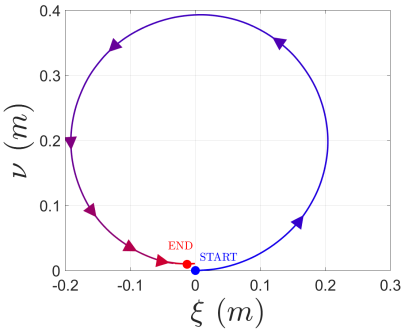


Figure 12.49: Robot position (using motor encoders and kin. eqs)

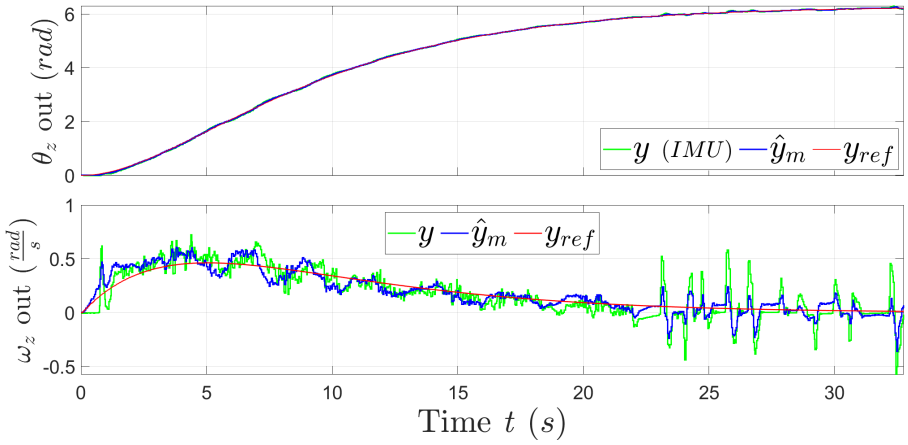


Figure 12.50: Orientation EMC y, $\hat{y}_m$ and y_{ref}

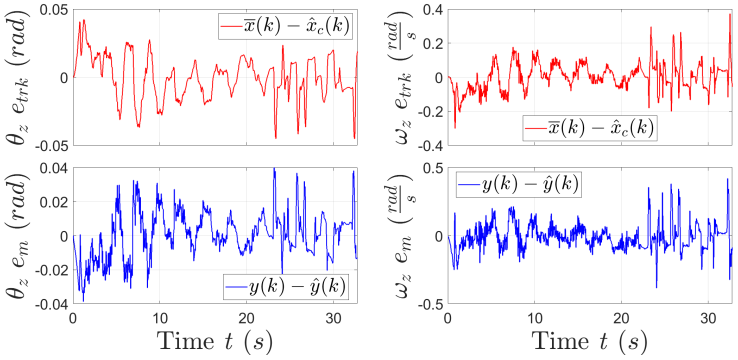


Figure 12.51: Tracking e_{trk} and model e_m errors for θ_z (1st column) and ω_z (2nd column)

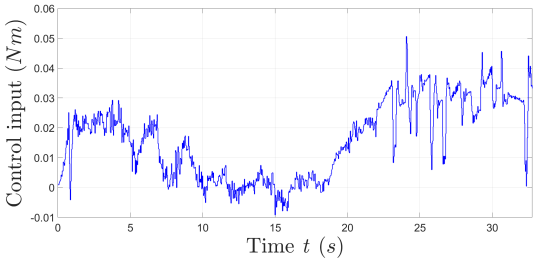


Figure 12.52: Robot torque control input (virtual)

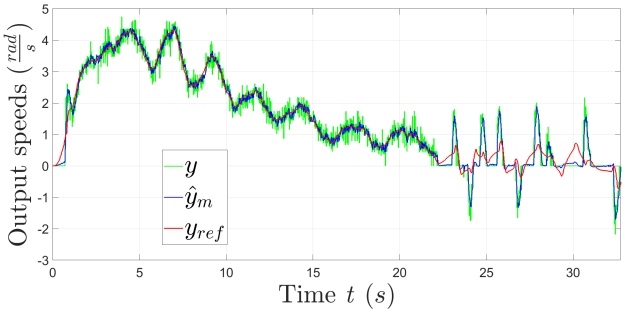


Figure 12.53: Right motor EMC y, $\hat{y}_m$ and y_{ref}

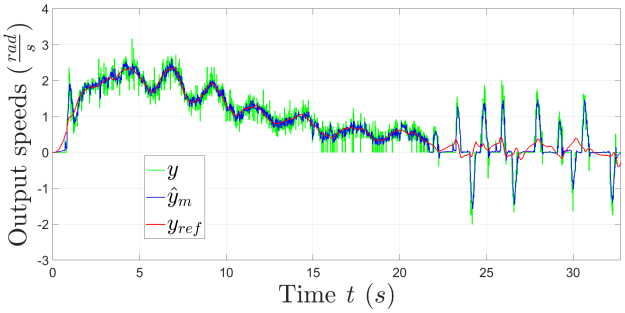


Figure 12.54: Left motor EMC y, $\hat{y}_m$ and y_{ref}

Robot in movement, circle r_m= 0.2 m, $[0, -2\pi]$ angle trajectory

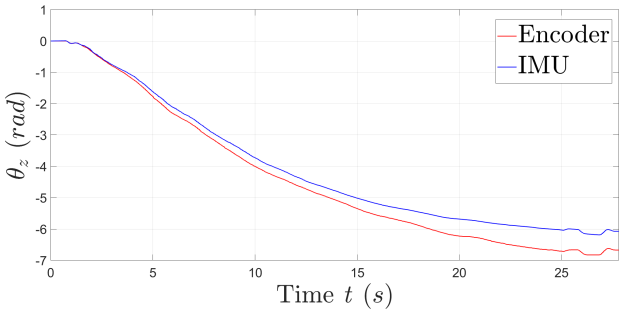


Figure 12.55: Encoder VS IMU θ_z measurement

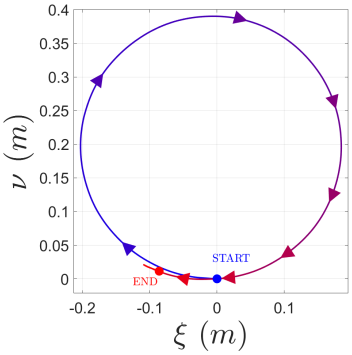


Figure 12.56: Robot position (using motor encoders and kin. eqs)

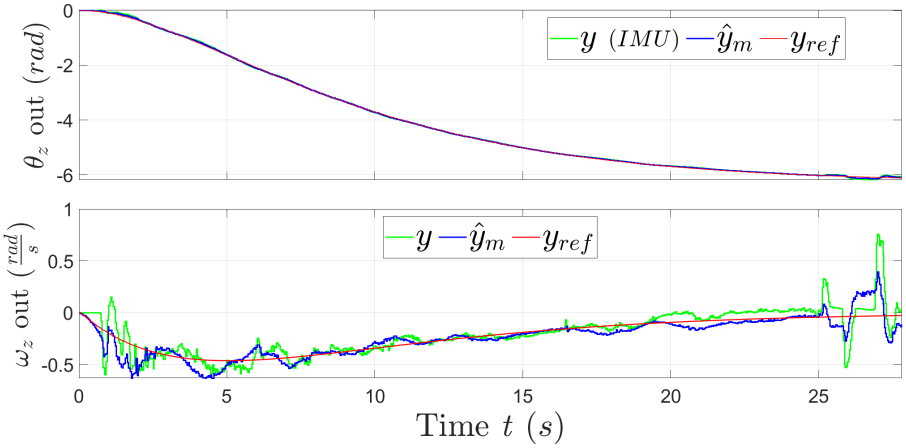


Figure 12.57: Orientation EMC y, $\hat{y}_m$ and y_{ref}

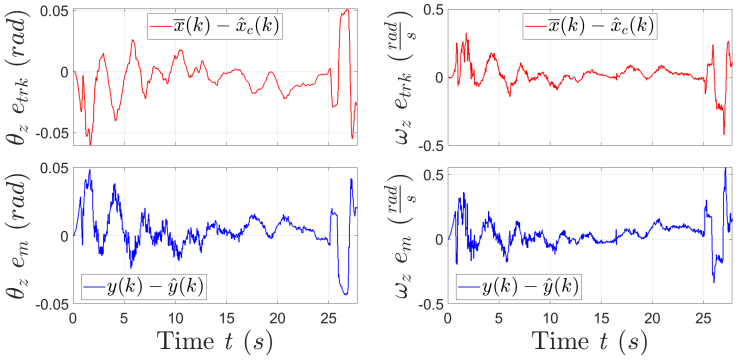


Figure 12.58: Tracking e_{trk} and model e_m errors for θ_z (1st column) and ω_z (2nd column)

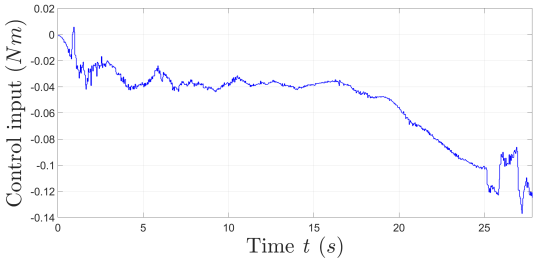


Figure 12.59: Robot torque control input (virtual)

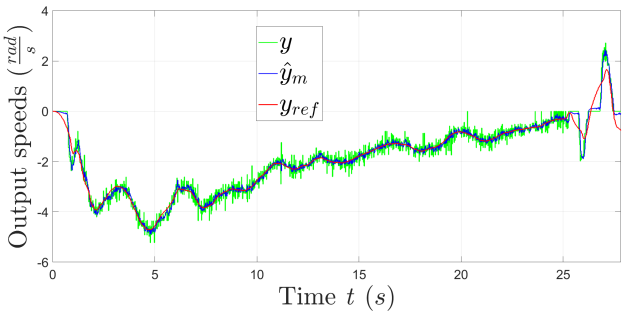


Figure 12.60: Right motor EMC y, $\hat{y}_m$ and y_{ref}

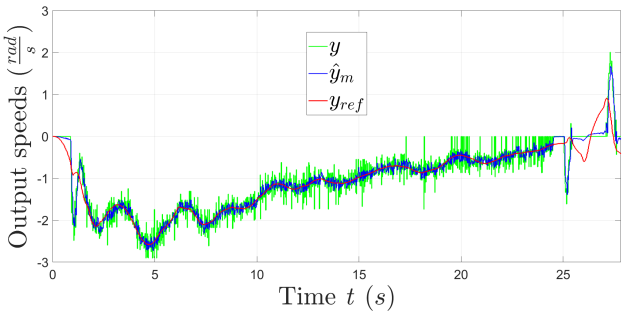


Figure 12.61: Left motor EMC y, $\hat{y}_m$ and y_{ref}

12.4.3 Encoder θ_z measurement

Using encoder motor position to obtain the robot yaw angle measurement gives reliable values when robot starts from $0\,\text{rad}$ and reaches $2\pi\,\text{rad}$ target angle (either when robot has or not linear velocity), but drifts when robot is rotating in opposite directions (from $2\pi\,\text{rad}$ to $0\,\text{rad}$), as it can be seen in Figure 12.68 and especially Figure 12.81. This leads to avoid motor encoders for θ_z measurements.

Robot still, $[0, 2\pi]$ angle trajectory

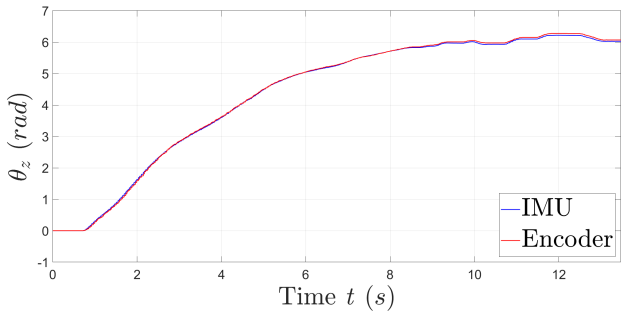


Figure 12.62: Encoder VS IMU θ_z measurement

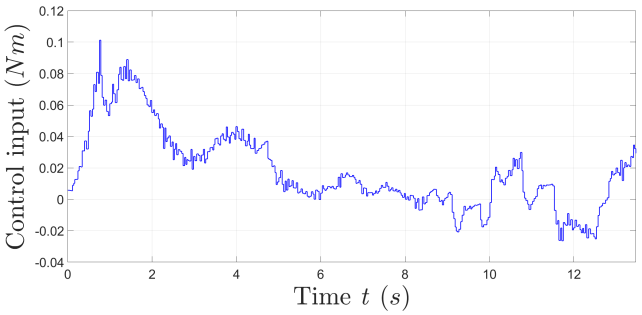


Figure 12.63: Robot torque control input (virtual)

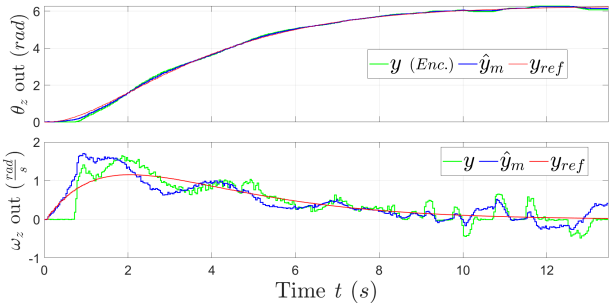


Figure 12.64: Orientation EMC y, $\hat{y}_m$ and y_{ref}

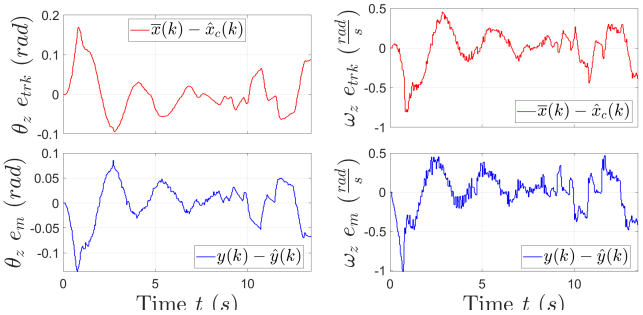


Figure 12.65: Tracking e_{trk} and model e_m errors for θ_z (1st column) and ω_z (2nd column)

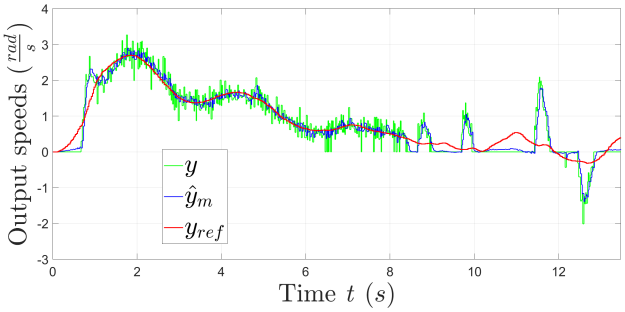


Figure 12.66: Right motor EMC y, $\hat{y}_m$ and y_{ref}

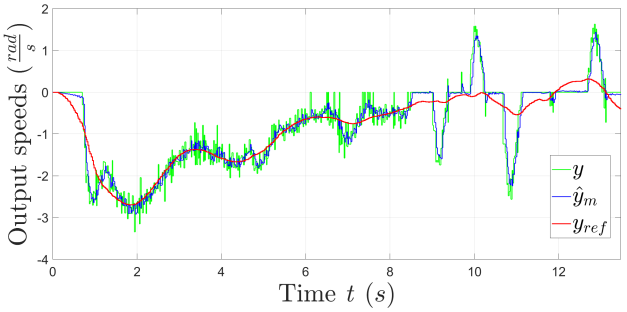


Figure 12.67: Left motor EMC y, $\hat{y}_m$ and y_{ref}

Robot still, $[0, -2\pi]$ angle trajectory

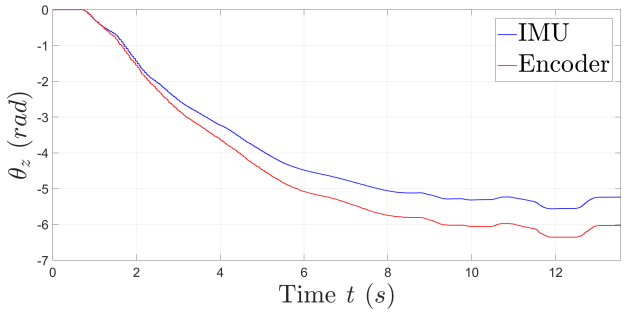


Figure 12.68: Encoder VS IMU θ_z measurement

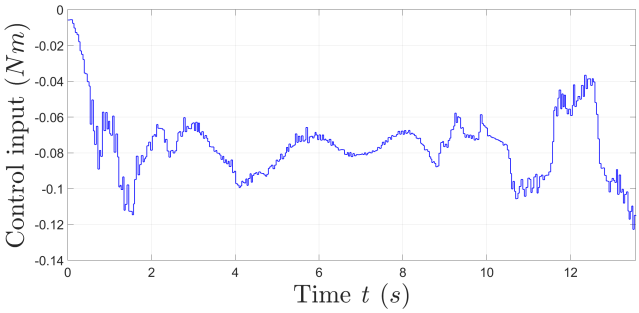


Figure 12.69: Robot torque control input (virtual)

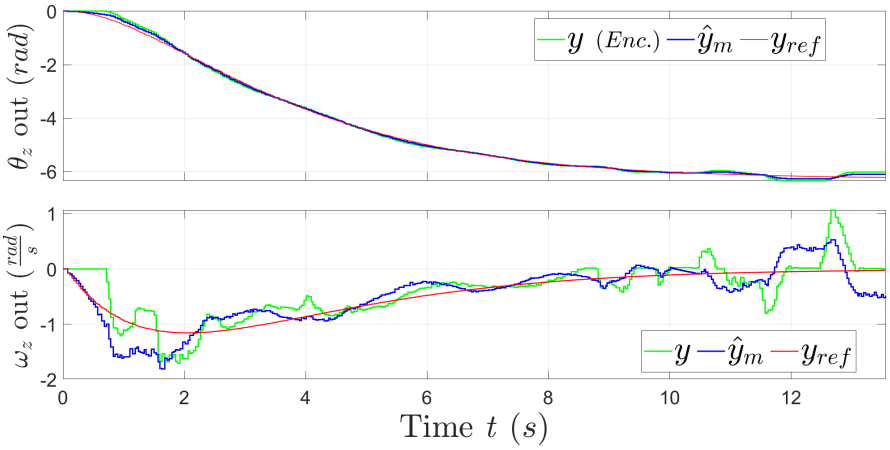


Figure 12.70: Orientation EMC y, $\hat{y}_m$ and y_{ref}

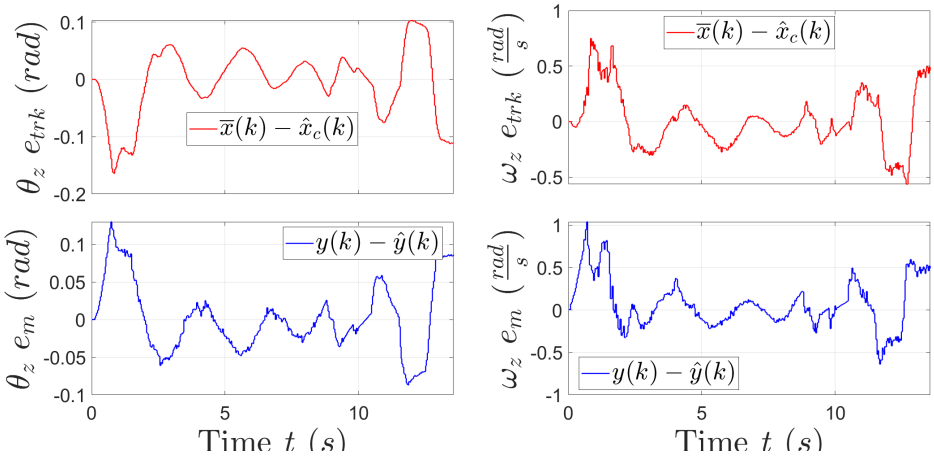


Figure 12.71: Tracking e_{trk} and model e_m errors for θ_z (1st column) and ω_z (2nd column)

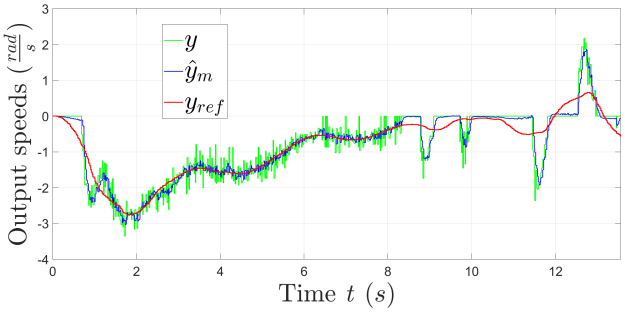


Figure 12.72: Right motor EMC y, $\hat{y}_m$ and y_{ref}

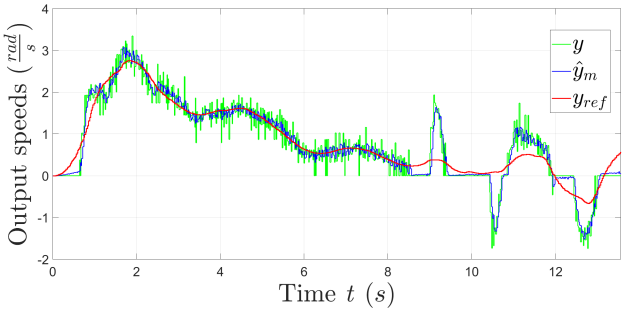


Figure 12.73: Left motor EMC y, $\hat{y}_m$ and y_{ref}

Robot in movement in circle r_m = 0.2 m, $[0, 2\pi]$ angle trajectory

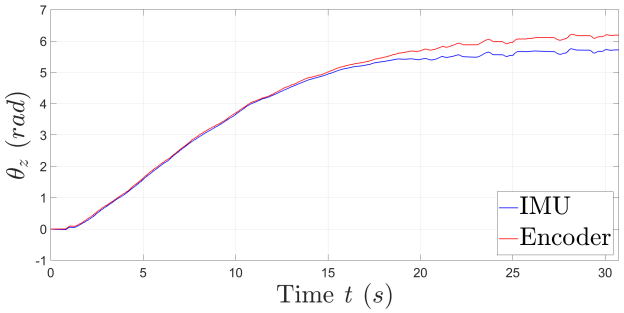


Figure 12.74: Encoder VS IMU θ_z measurement

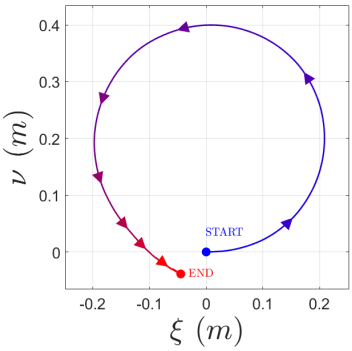


Figure 12.75: Robot position (using motor encoders and kin. eqs)

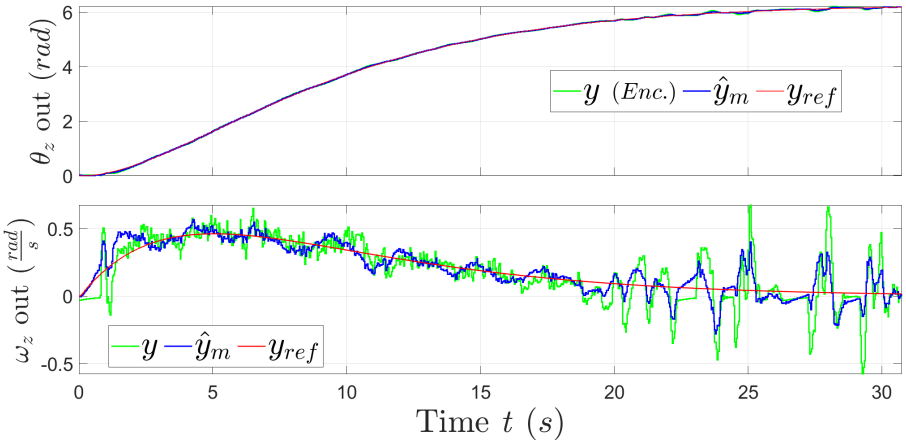


Figure 12.76: Orientation EMC y, $\hat{y}_m$ and y_{ref}

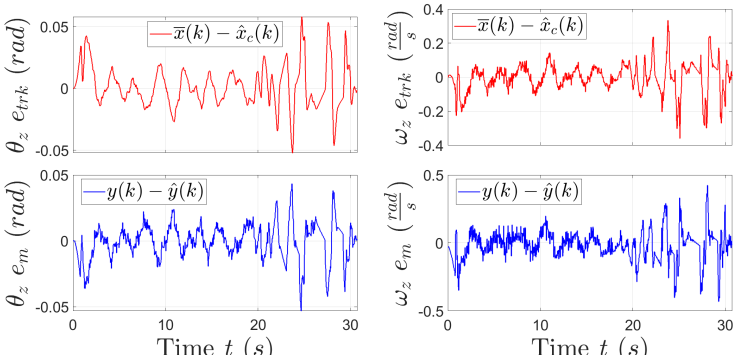


Figure 12.77: Tracking e_{trk} and model e_m errors for θ_z (1st column) and ω_z (2nd column)

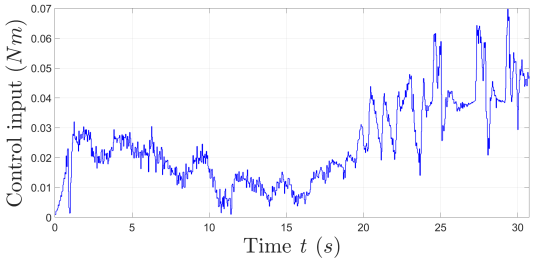


Figure 12.78: Robot torque control input (virtual)

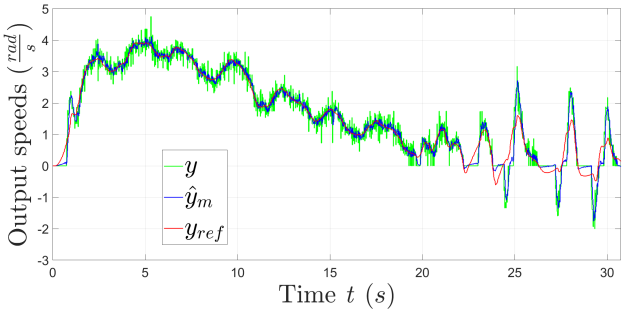


Figure 12.79: Right motor EMC y, $\hat{y}_m$ and y_{ref}

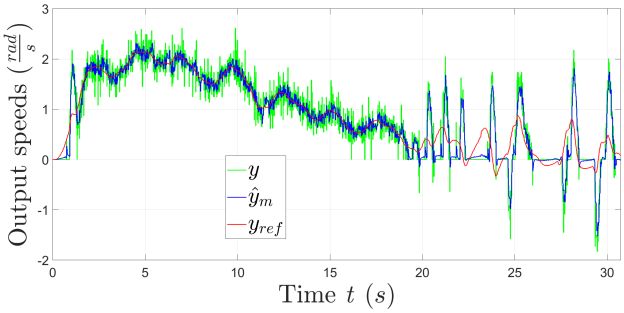


Figure 12.80: Left motor EMC y, $\hat{y}_m$ and y_{ref}

Robot in movement, circle r_m= 0.2 m, $[0, -2\pi]$ angle trajectory

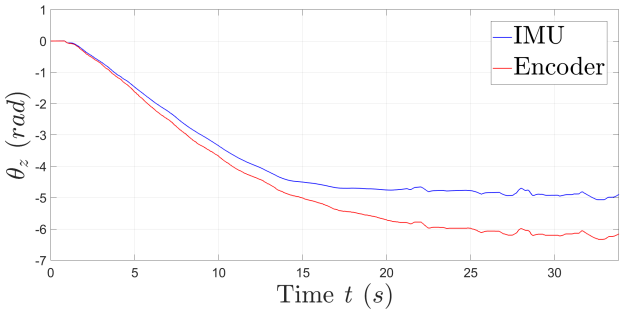


Figure 12.81: Encoder VS IMU θ_z measurement

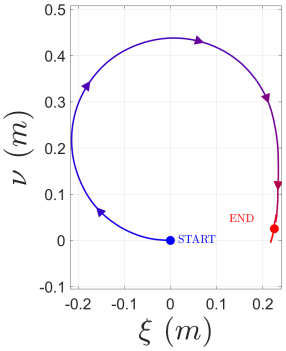


Figure 12.82: Robot position (using motor encoders and kin. eqs)

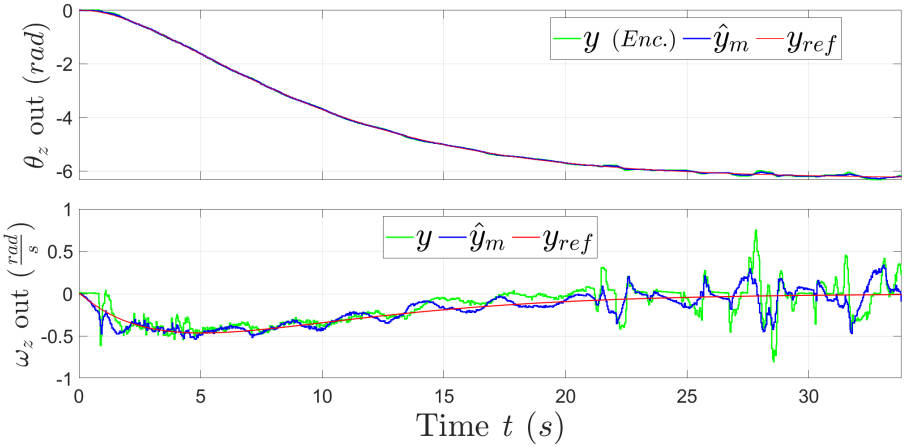


Figure 12.83: Orientation EMC y, $\hat{y}_m$ and y_{ref}

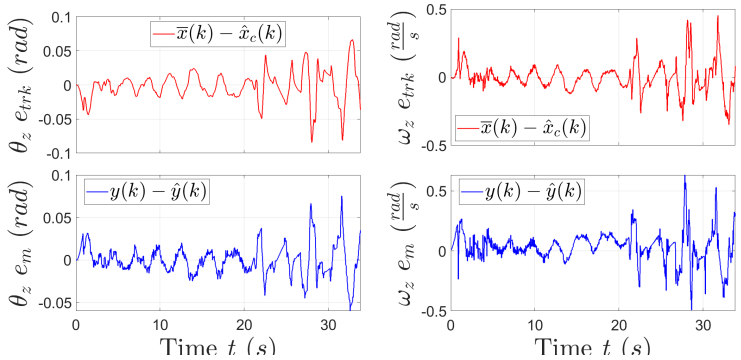


Figure 12.84: Tracking e_{trk} and model e_m errors for θ_z (1st column) and ω_z (2nd column)

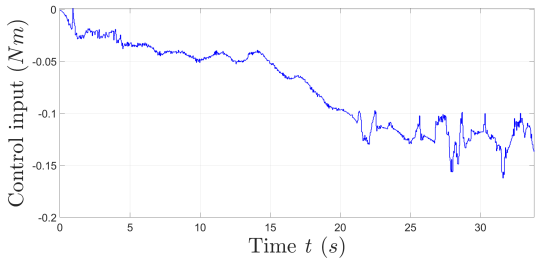


Figure 12.85: Robot torque control input (virtual)

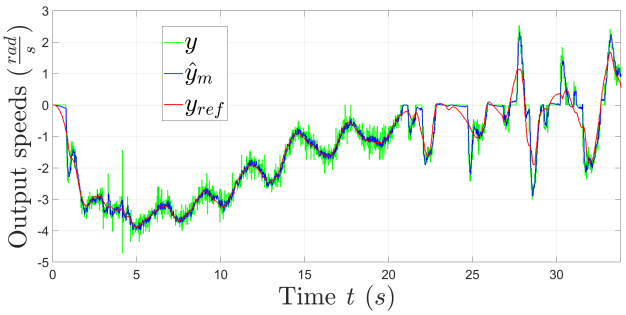


Figure 12.86: Right motor EMC y, $\hat{y}_m$ and y_{ref}

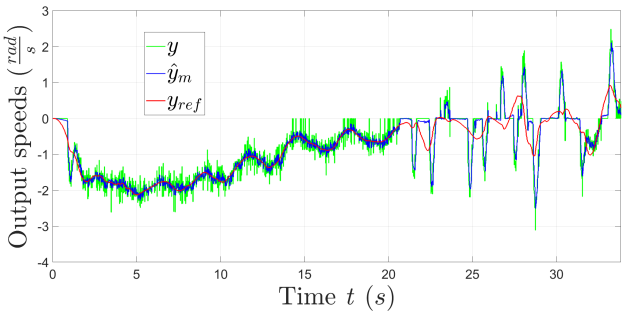


Figure 12.87: Left motor EMC y, $\hat{y}_m$ and y_{ref}

12.5 RHIT tests - Dynamic feedback measure driven decomposition noise estimator

Another possibility is to use 2 noise estimators for both controllable states, static FB for θ_z and dynamic FB for ω_z, exploiting measure-driven decomposition .
The IMU is used to measure θ_z, with presence of magnetometer data in fusion algorithm.
An angle trajectory of $[0, 2\pi]$ rad is done, and no further improvements on tracking and model errors are reached with respect to the control model with only 2 static noise estimators. This is because the demanding input torque is very high and the control too aggressive, as we can see in Figure 12.90: it reaches values higher than $1.5\,\mathrm{N\,m}$ when the target is almost reached, which is impossible in practice (because the torque must be near to zero). For this reason, this solution is rejected even if the estimated orientation angle follows quite well the reference and measurement.
To avoid saturations max torque allowed is heavily increased $\tau_T = 100\,\mathrm{N\,m}$, which is obviously physically too high for a robot trajectory like this one. For orientation control model the following CT eigenvalues are selected:

Eigenvalues type	CT eigenvalues
Reference μ_R	-0.5×2
1st noise estimator (static) $\mu_{N,1}$	-5
2nd noise estimator (static part) $\mu_{N,21}$	-10×2
2nd noise estimator (dynamic part) $\mu_{N,22}$	-10×3
Control μ_K	$-50, -30$

Table 12.4: CT eigenvalues Orientation EMC

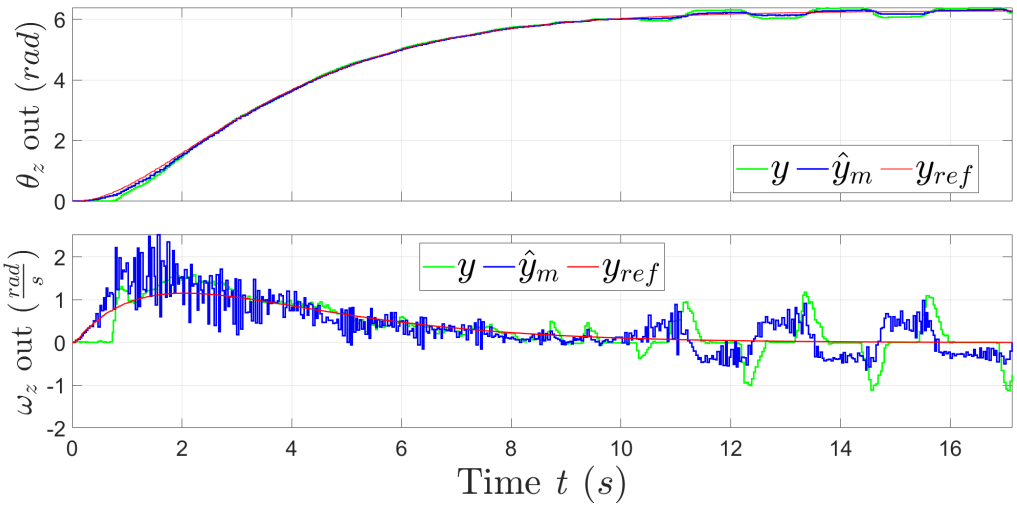


Figure 12.88: Orientation EMC y, $\hat{y}_m$ and y_{ref}

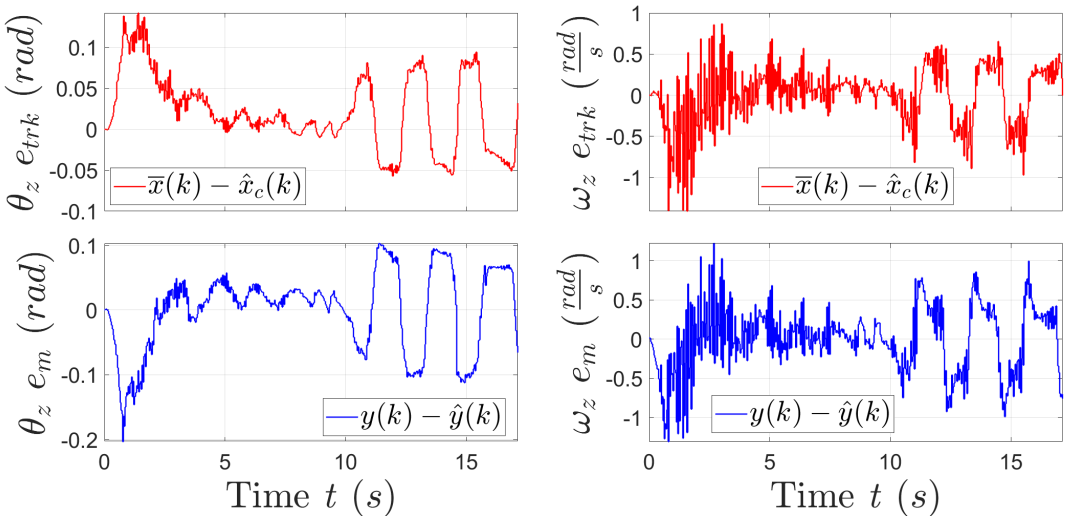


Figure 12.89: Tracking e_{trk} and model e_m errors for θ_z (1st column) and ω_z (2nd column)

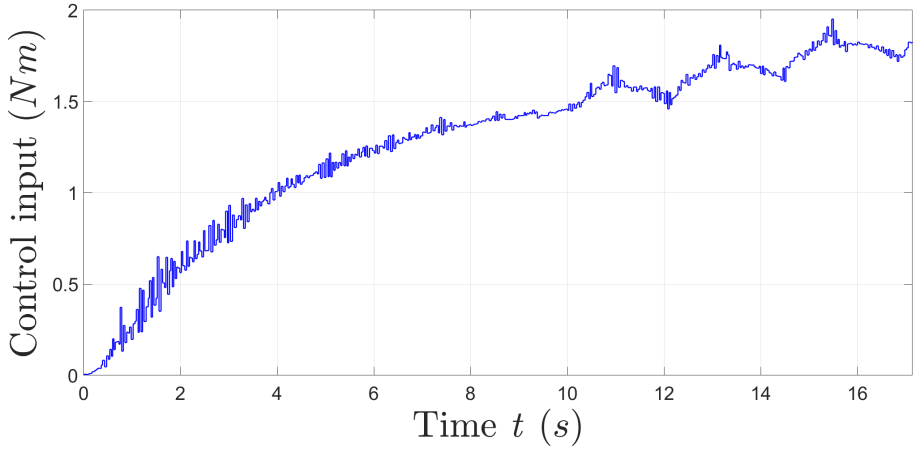


Figure 12.90: Robot torque control input (virtual)

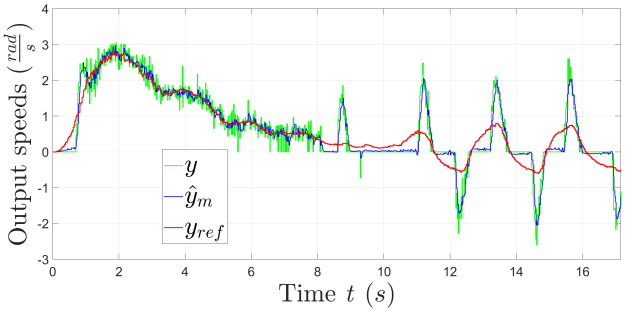


Figure 12.91: Right motor EMC y, $\hat{y}_m$ and y_{ref}

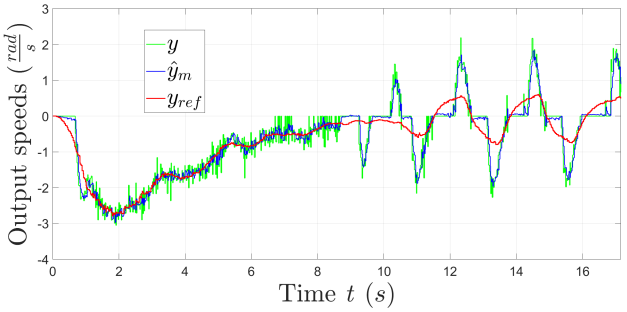


Figure 12.92: Left motor EMC y, $\hat{y}_m$ and y_{ref}

Part IV

Robot longitudinal position EMC

Chapter 13

EMC Differential drive robot theory - Longitudinal position only

The chapter is focused in theoretically building EMC to control only longitudinal position and speed of robot. Hence the robot will be allowed to move only in one direction, longitudinal along one axis.
To define axes reference, for simplicity the inertial RF is considered equal to robot RF in rest position, with the origin placed in robot wheels axle: in this way the robot position is always zero in both 2D axes, for every test. In Figure 3.6 this is obtained overlapping the 2 inertial and body reference frames (IF and BF).
Both fine (for MIL simulations) and EMC models are very similar to orientation ones discussed in Chapter 10. Indeed for orientation the robot total torque is the command input, for position control is the robot total force: hence if the only parameter for orientation was the total robot inertia I_T, for position will be the *total robot mass* M_T. This is a huge benefit, because many settings are already defined.
Notation: From now on, since x symbol is already used to define EM states, robot longitudinal positions will be referred as ξ and longitudinal velocity as $v_\xi = \dot{\xi}$.

13.1 Fine model

Dynamic equation of robot position is:

$$F_T = \sum_{i=1}^{2} F_i(t) = F_L(t) + F_R(t) = M_T \ddot{\xi}(t) \tag{13.1}$$

Where M_T is the robot total mass parameter, $F_L(t)$ and $F_R(t)$ are the longitudinal forces acting on the 2 wheels and their sum results in total force $F_T(t)$. Total robot acceleration is defined as $\ddot{\xi}(t)$. The last equation can be split in 2 ODE:

$$\begin{aligned} \dot{\xi}(t) &= \dot{\xi}(t) \\ \ddot{\xi}(t) &= \frac{F_T(t)}{M_T} \end{aligned} \tag{13.2}$$

The total mass is well defined (it is sufficient to weight the robot) and its value is $M_T = 0.68\,\text{kg}$, Equation (3.1).
The error is already implemented in the motor fine model with quantization needed for position encoders, so disturbances in position fine model are not needed.

13.2 EMC model

13.2.1 DT EM equations

Similarly to Orientation EMC, also EM for position control can be build considering 2 integrators. Starting from CT state equations in Equation (13.2) disturbances can be added:

$$\begin{aligned}\dot{\xi}(t) &= \dot{\xi}(t) + \overline{w}_1(t)\\ \ddot{\xi}(t) &= x_d(t) + \frac{F_T(t)}{M_T} + \overline{w}_2(t)\\ \dot{x}_d(t) &= \overline{w}_3(t)\end{aligned}$$

Where $\overline{w}_i(t)$ are noises affecting the states.
From orientation control model it's experienced that is possible to measure both controllable states, thus even in the previous CT equations noises $\overline{w}_i(t)$ affect both position and speed states.
For disturbance state equation a first order is sufficient, and the state is added to the system as acceleration perturbation. In CT is sufficient only 1 integrator and a not-causal disturbance.
In discrete time, CT integrator transforms into a Forward-Euler DT intergrator as in Equation (10.2). Vector of states will be $x^T(k) = \left[\xi, v_\xi, x_d\right](k)$ and the EM model DT SS equations in matrix form become:

$$\begin{aligned}x(k+1) &= \underbrace{\begin{bmatrix}1 & T & 0\\ 0 & 1 & T\\ 0 & 0 & 1\end{bmatrix}}_{A_{DT}} x(k) + \underbrace{\begin{bmatrix}0\\ \frac{T}{I_T}\\ 0\end{bmatrix}}_{B_{DT}} u(k) + \underbrace{\begin{bmatrix}T & 0 & 0\\ 0 & T & 0\\ 0 & 0 & T\end{bmatrix}}_{G_{DT}} \overline{w}(k) \quad , \quad x_0(k) = 0_{3\times 1}\\ y(k) &= \underbrace{\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}}_{C_{DT}}(k)\end{aligned} \tag{13.3}$$

All matrices are decomposed in the following submatrices:

$$\begin{aligned}A_{DT} &= \begin{bmatrix}A_c & H_c\\ 0 & Ad\end{bmatrix} = \left[\begin{array}{cc|c}1 & T & 0\\ 0 & 1 & T\\ \hline 0 & 0 & 1\end{array}\right] \quad , \qquad B_{DT} = \begin{bmatrix}B_c\\ 0\end{bmatrix} = \left[\begin{array}{c}0\\ \frac{T}{I_T}\\ \hline 0\end{array}\right]\\ G_{DT} &= \begin{bmatrix}G_c\\ G_d\end{bmatrix} = \left[\begin{array}{ccc}T & 0 & 0\\ 0 & T & 0\\ \hline 0 & 0 & T\end{array}\right] \quad , \qquad C_{DT} = \begin{bmatrix}C_c & C_d\end{bmatrix} = \left[\begin{array}{cc|c}1 & 0 & 0\\ 0 & 1 & 0\end{array}\right]\end{aligned} \tag{13.4}$$

The matrices in Equation (13.3) are decomposed in 2 parts using matrix driven decomposition, exactly in the form used for orientation EMC matrices in Equation (10.11).
In Figure 13.1, violet area shows the discrete time first order noise disturbances $\overline{w}_i(k)$.

13.2.2 Control law

Reference theory for control law is Equation (2.11) in Chapter 2: K, M, Q matrices need to be found.
Concerning $K = K_p = [k_{p1}, k_{p2}]$ matrix, the very same proportional control only (P) used for orientation EM is considered even for this EM, to track the reference for both states ξ and $v_\xi = \dot{\xi}$. Theory to find k_{p1} and k_{p2} components is already explained in Section 10.2.2 with pole placement technique, hence only final expressions are

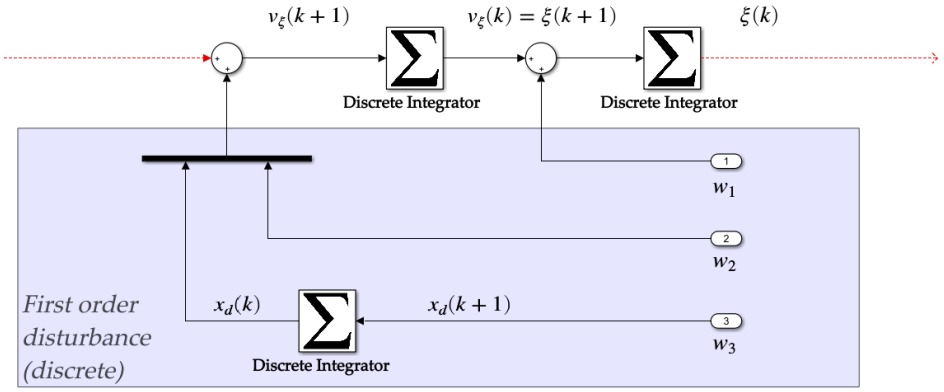


Figure 13.1: First order disturbance w entering position EM

reported.
CT eigenvalues are decided arbitrary by us and then converted into DT ones $p = [p_{k1}, p_{k2}]$, every time step. Desired characteristic polynomial can be defined as:

$$\lambda^2 + a_{k2}\lambda + a_{k3} = (\lambda - p_{k1})(\lambda - p_{k2})$$

K_p matrix components are:

$$\begin{aligned} k_{p1} &= \frac{M_T(1 + a_{k2} + a_{k3})}{T^2} \\ k_{p2} &= \frac{M_T(a_{k2} + 2)}{T} \end{aligned} \tag{13.5}$$

$Q^T = [q_1, q_2], M = m$ matrices can be univocally determined using 2^{nd} condition in Equation (2.12):

$$\begin{cases} q_1 = q_2 = 0 \\ m = M_T \end{cases}$$

13.2.3 Reference dynamics

Theory about reference dynamics is already discussed in Section 10.2.3, following a static-state FB reference controller. Again starting from desired CT eigenvalues $p_R = [p_{r1}, p_{r2}]$, desired characteristic polynomial will be of second order:

$$\lambda^2 + a_{r2}\lambda + a_{r3} = (\lambda - p_{r1})(\lambda - p_{r2})$$

Components for matrix $K_R = [k_{r1}, k_{r2}]$ are finally:

$$\begin{aligned} k_{r1} &= \frac{M_T(1 + a_{r2} + a_{r3})}{T^2} \\ k_{r2} &= \frac{M_T(a_{r2} + 2)}{T} \end{aligned} \tag{13.6}$$

Static-state FB requires also a matrix N_R to control overall closed loop system DC-gain, which can be computed with Equation (2.6) if K_R is known.

13.2.4 Noise estimator

2 noise estimators are needed to estimate the disturbances $\overline{w}$ of ξ and v_ξ. Since the only difference between orientation and position EM is the parameter of B_c matrix in Equation (13.4) (mass M_T instead of inertia I_T), and

the estimator expression do not depend on this parameter ⇒ noise estimator for EMC long. position model is exactly the same of Orientation EMC (cfr. Section 10.2.4, solution 2).
First static FB estimator (ξ) expression can be found using pole placement technique, considering the desired polynomial as $a_n^0 = -p_{n0}$, with p_{n0} the desired eigenvalue arbitrary decided by us inside unitary circle for internal stability. Hence, noise $\overline{w}_0$ can be estimated as:

$$\overline{w}_0(k) = L_0\overline{e}_0(k)$$
$$L_0 = l_0 = \frac{a_n^0 + 1}{T}$$

This estimator is related to model error $\overline{e}_0 = \xi - \hat{\xi}$.
The 2nd noise estimator (v_ξ) is again with static FB with $L_1^T = [l_{10}, l_{11}]$, the noise vector $\overline{w}_1 \in \mathbb{R}^{2\times 1}$ and can be estimated as:

$$\overline{w}_1(k) = L_1\overline{e}_1(k)$$
$$L_1 = \begin{cases} l_{10} = \dfrac{a_{n2}^1 + 2}{T} \\ l_{11} = \dfrac{a_{n2}^1 + a_{n3}^1 + 1}{T^2} \end{cases}$$

With a_{n2}^1 and a_{n3}^1 recovered with the following desired characteristic polynomial, considering p_{n1} and p_{n2} decided arbitrary:

$$\lambda^2 + a_{n2}^1\lambda + a_{n3}^1 = (\lambda - p_{n1})(\lambda - p_{n2})$$

In this case model error is $\overline{e}_1 = v_\xi - \hat{v}_\xi$.
Total estimated disturbance is:

$$\overline{w} = \begin{bmatrix} \overline{w}_0 \\ \overline{w}_1 \end{bmatrix}$$

Chapter 14

Longitudinal position EMC - Model-in-the-Loop (MIL) simulation tests

In Chapter 13 control and fine models for controlling robot longitudinal position are explained in theory, now MIL simulation tests can be performed with both models translated and run in Simulink.

14.1 General MIL settings on Simulink

14.1.1 Simulated long. position EMC and fine models

Longitudinal position fine model is composed by gain (total robot mass M_T) plus 2 integrators, as already explained in Equation (13.1).
As opposed to Orientation fine model for MIL tests, no additional noises or disturbances are added to the position and longitudinal speed measurements of longitudinal position fine model: this is because they're recovered using kinematic equations starting from wheel encoders position measurements, hence all the errors are present in DC motors fine model outputs ω_R and ω_L (check loop structure path explained in Section 14.1.2).
Regarding longitudinal position EMC nothing changes from model blocks explained in Section 13.2.

14.1.2 Long. position and DC motors Hierarchical structure

Hierarchical structure is exploited for Longitudinal position control model too, to be combined with DC motors EMC. Because it's practically the same as the one of Orientation EMC (for structure features refer to Section 11.1.2) here will be presented only the differences. Taking a look at Figure 14.1, they are:

- Orientation EMC is substituted by Longitudinal position EMC, which now tries to control a target position $\tilde{\xi}(k)$ (reference dynamics inside the model gives as output the desired trajectory).

- Kinematic conversion block is the same, but since now it is the orientation control supposed to be absent, a target $\tilde{\omega}_z(k) = 0\,\mathrm{rad/s}$ is imposed (plus the longitudinal position EMC estimated output $\hat{v}_\xi(k)$).

- The estimated wheel speeds from DC motors EMC are converted not in a total torque but a total force $F_T(t)$ entering longitudinal position fine model. The conversion can be performed using the transformation in Equation (3.11), to pass from wheel speeds to robot longitudinal acceleration/force:

$$F_T(t) = M_T\ddot{\xi}(t) = \frac{M_T\rho}{2}\left(\ddot{\phi}_R(t) + \ddot{\phi}_L(t)\right) \tag{14.1}$$

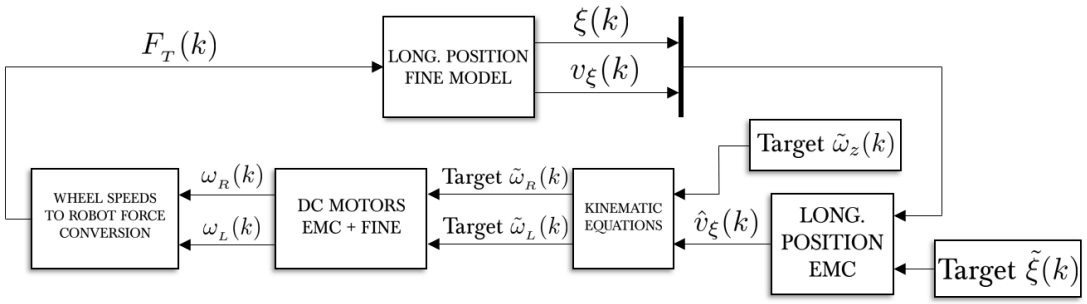


Figure 14.1: Hierarchic structure for MIL tests - Longitudinal position and DC motors EMC

- Longitudinal position fine model measurements are now $\xi(t)$ and $v_\xi(t)$.

DC motors EMC CT eigenvalues for DC motors are the same as Table 6.1, for both right and left motors.
Instead for Longitudinal position EMC the same final CT eigenvalues giving the best results in RHIT are considered, the ones in Table 15.1 (when robot is covering $2\,\mathrm{m}$ distance the reference CT eigenvalues changes from -0.5×2 to -0.2×2).
Virtual force is saturated at $10\,\mathrm{N}$.

14.2 General settings on Simulink MIL plot results

Longitudinal trajectories are of $1\,\mathrm{m}$ (one way and round trip) and $2\,\mathrm{m}$, with 0° angle trajectory. MIL plot simulations are the result of continuous comparison with RHIT. Only best MIL tests are shown in the plots.
Every set of results is divided in 2 parts: the first plots are related to MIL simulations only, showing $y, \hat{y}_m$ and y_{ref} for long. position ξ and speed v_ξ, with relative tracking and model errors e_m and e_{trk}, related to longitudinal position control model. Other two plots are for measurements and estimations of DC motor output speeds.
The second set of plots instead shows a comparison between MIL and RHIT testing procedures, where RHIT are taken from Chapter 15: the Timestamp obtained with RHIT is reproduced quite exactly in simulation, for all control models (the timestamp is the same for the first set of plots too). Also a comparison between the 2 virtual control inputs is evidenced, to have an indication of its reliability and control model behaviour.
Disturbance rejection term is present in control input for all tests, except from one of them (Section 14.3.4), to make a comparison with and without.

14.3 Summary on the obtained results

Regarding only MIL simulations tracking and model errors are very low (at max some cm for both e_m, and e_{trk} related to ξ, and lower than $0.1\,\mathrm{m/s}$ for errors related to v_ξ).
For all tests comparison between MIL and RHIT reveals that the simulations are very similar to reality. Making a comparison between tests with and without disturbance rejection, in Section 14.3.3 and Section 14.3.4, we can see that the difference is very low. Indeed making a comparison between long. position EMC errors e_m and e_{trk} with disturbance rejection (Figure 14.25) and without (Figure 14.37), they're almost the same apart from e_{trk} for state related to ξ: maximum peak is reached at $|0.02|\ \mathrm{m}$ with disturbance rejection, however an higher peak at $|0.03|\mathrm{m}$ is obtained without disturbance rejection. This is due to the fact that the total robot mass parameter used in EMC model is very near to reality (because is simple to weight the robot). Comparing virtual force control inputs in Figure 14.26 and Figure 14.38, when no disturbance rejection is present, the input component related to tracking error u_{trk} is increased from 1 to 2 N, to compensate the absence of $u_d = 0\,\mathrm{N}$.

14.3.1 Target position $\tilde{\xi}$ = 1 m, target ang. speed $\tilde{\omega}_z$ = 0 $\frac{\text{rad}}{\text{s}}$, dist. rej.

MIL simulation test results:

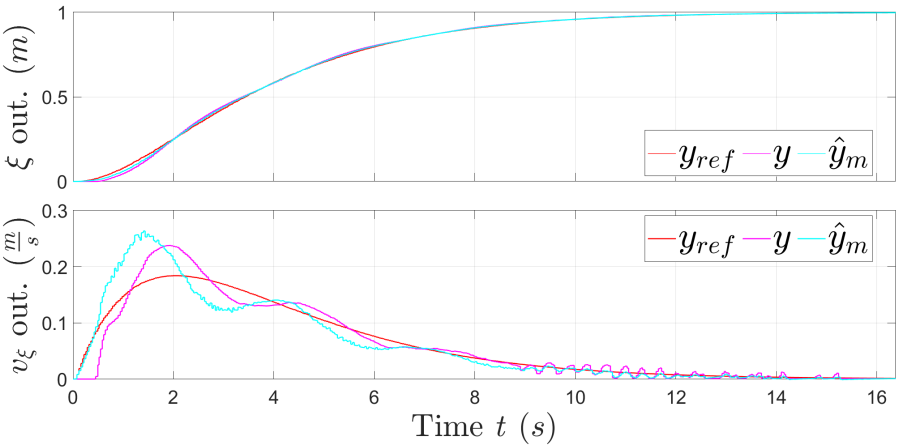


Figure 14.2: MIL Long. position EMC outputs y_{ref} (red), y (magenta) and $\hat{y}_m$ (cyan)

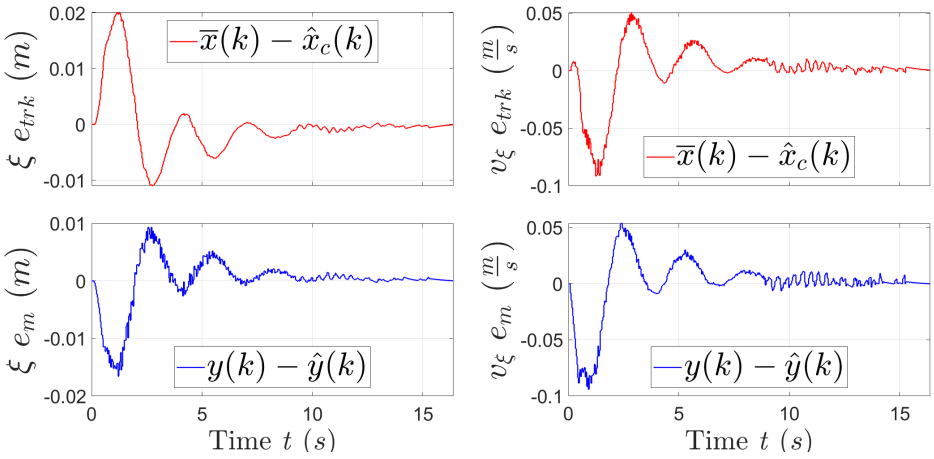


Figure 14.3: MIL Long. position EMC tracking e_{trk} and model e_m errors

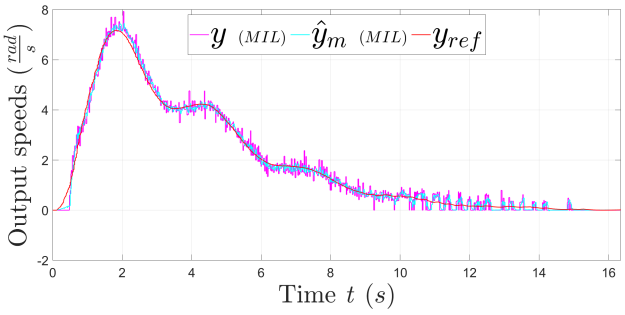


Figure 14.4: MIL Left motor EMC outputs y (magenta), $\hat{y}_m$ (cyan) and y_{ref} (red)

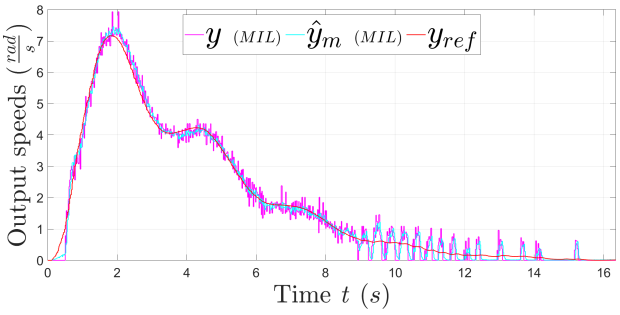


Figure 14.5: MIL Right motor EMC outputs y (magenta), $\hat{y}_m$ (cyan) and y_{ref} (red)

Comparison between MIL and RHIT:

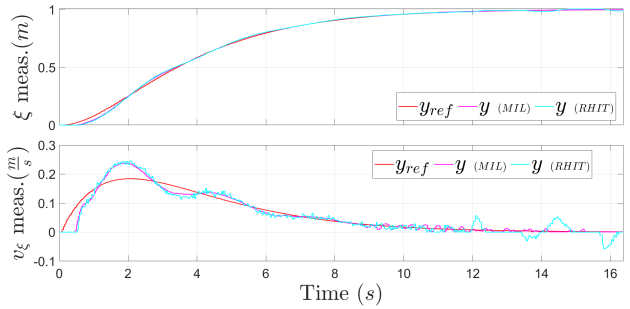


Figure 14.6: Long. position EMC measured outputs y (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

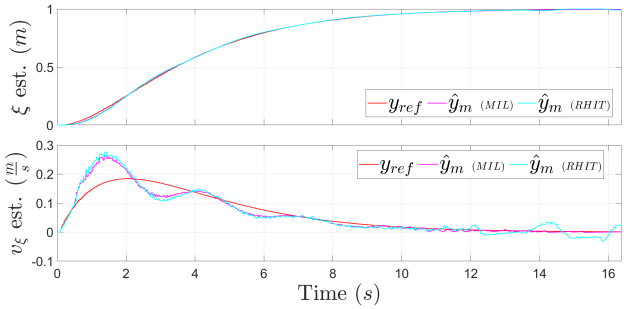


Figure 14.7: Long. position EMC estimated outputs $\hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

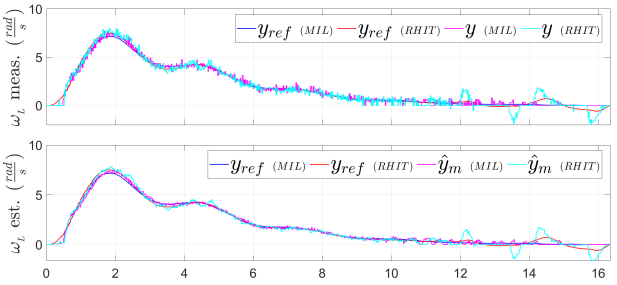


Figure 14.8: Left DC motor EMC measured outputs $y, \hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

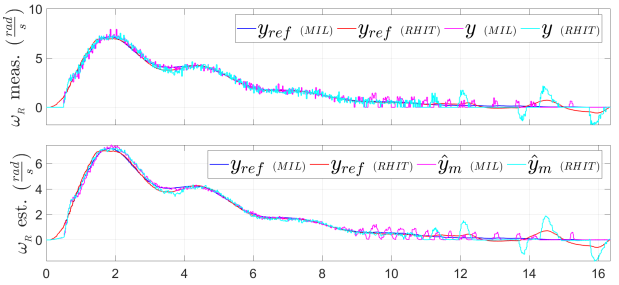


Figure 14.9: Right DC motor EMC measured outputs $y, \hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

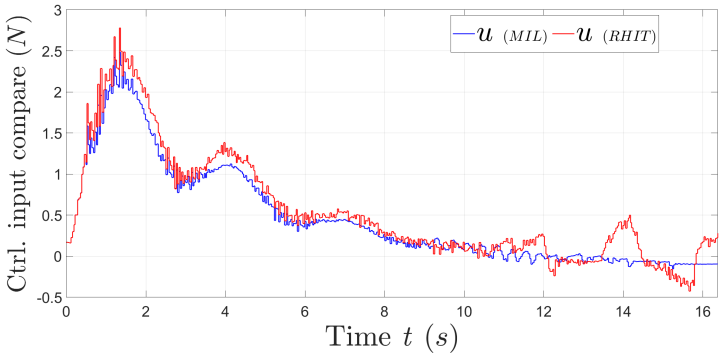


Figure 14.10: Long. position EMC virtual control input u - Comparison MIL and RHIT

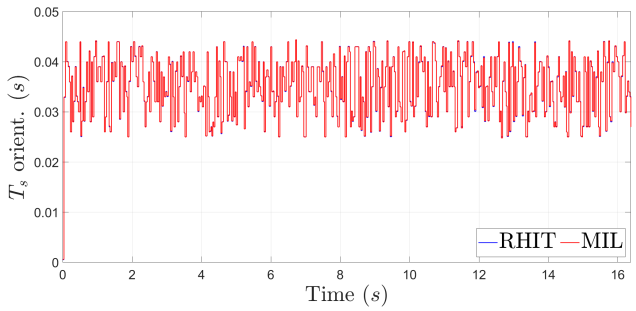


Figure 14.11: Long. position EMC Timestamp - Comparison MIL and RHIT

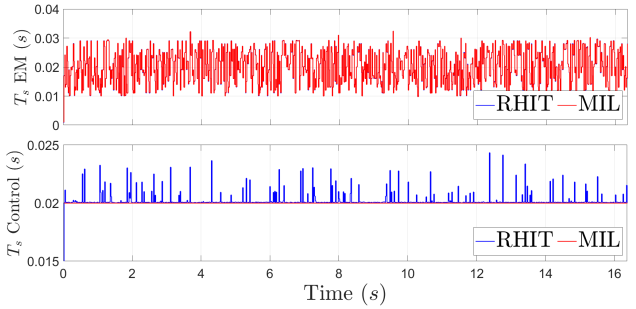


Figure 14.12: (Left and right) DC motors EMC Timestamp EM and Control part - Comparison MIL and RHIT

14.3.2 Target position $\tilde{\xi}$ = 2 m, target ang. speed $\tilde{\omega}_z$ = 0 $\frac{\text{rad}}{\text{s}}$, dist. rej.

MIL simulation test results:

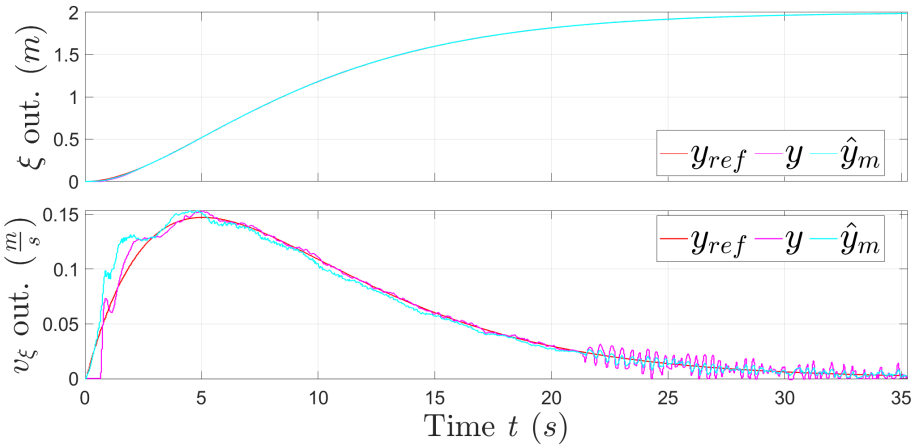


Figure 14.13: MIL Long. position EMC outputs y_{ref} (red), y (magenta) and $\hat{y}_m$ (cyan)

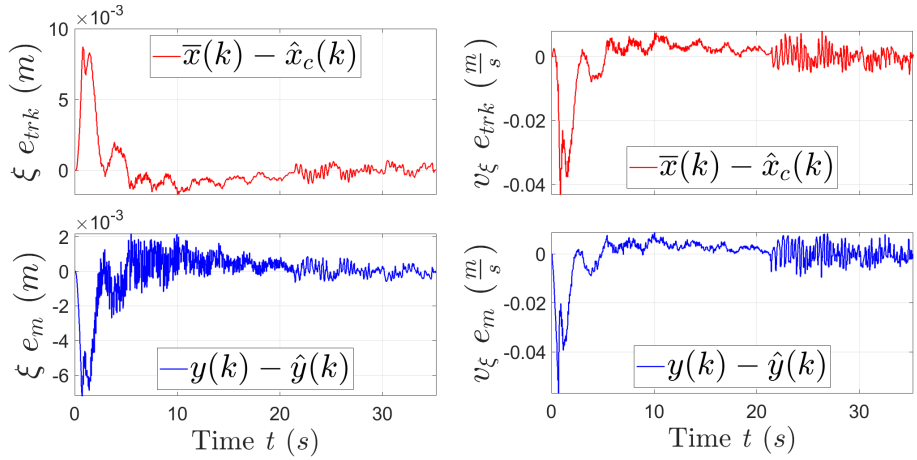


Figure 14.14: MIL Long. position EMC tracking e_{trk} and model e_m errors

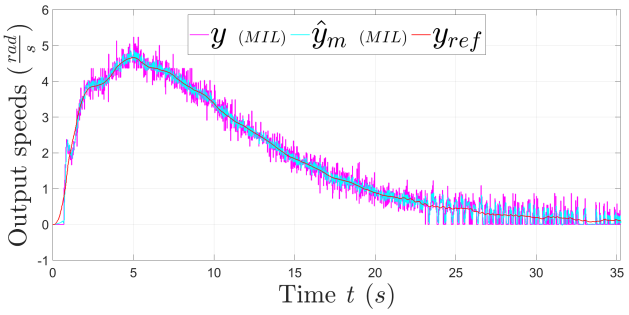


Figure 14.15: MIL Left motor EMC outputs y (magenta), $\hat{y}_m$ (cyan) and y_{ref} (red)

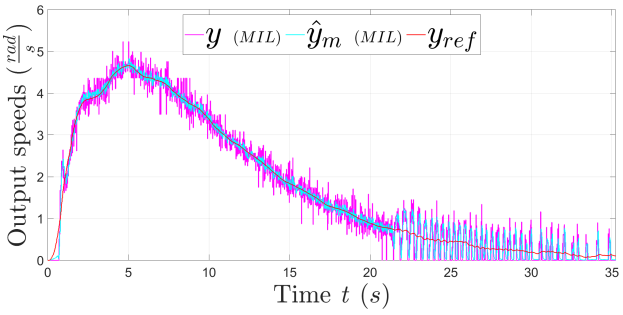


Figure 14.16: MIL Right motor EMC outputs y (magenta), $\hat{y}_m$ (cyan) and y_{ref} (red)

Comparison between MIL and RHIT:

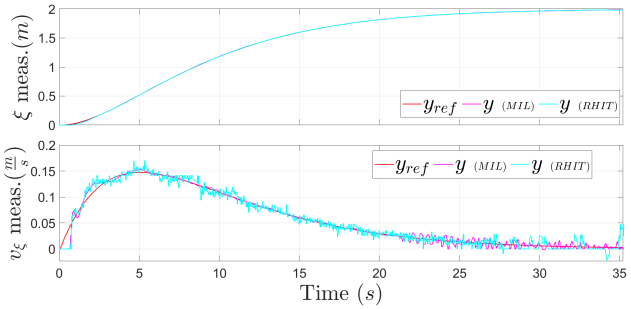


Figure 14.17: Long. position EMC measured outputs y (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

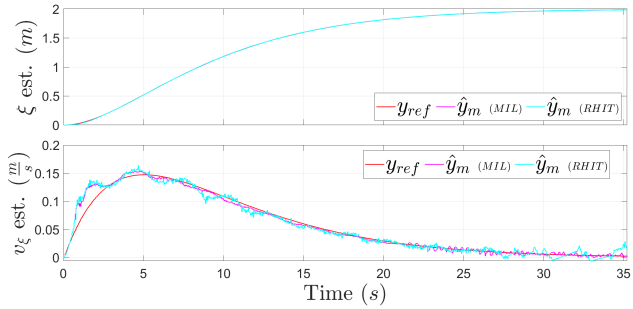


Figure 14.18: Long. position EMC estimated outputs $\hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

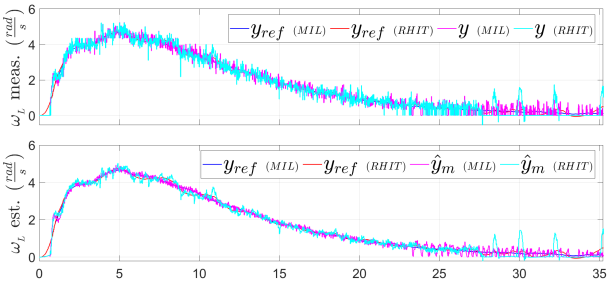


Figure 14.19: Left DC motor EMC measured outputs $y, \hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

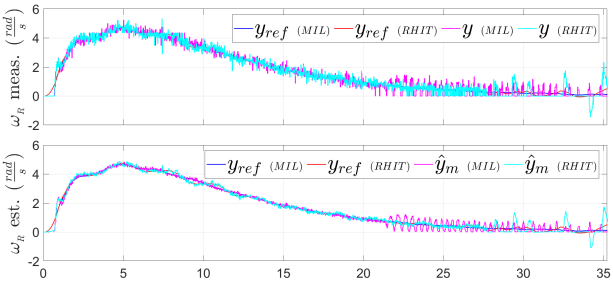


Figure 14.20: Right DC motor EMC measured outputs $y, \hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

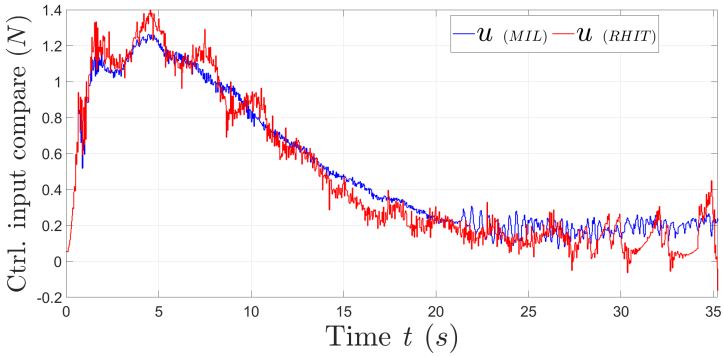


Figure 14.21: Long. position EMC virtual control input u - Comparison MIL and RHIT

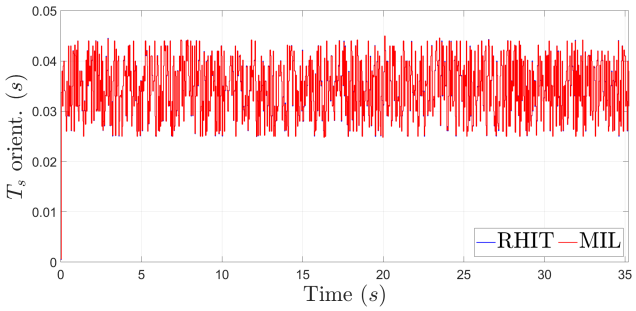


Figure 14.22: Long. position EMC Timestamp - Comparison MIL and RHIT

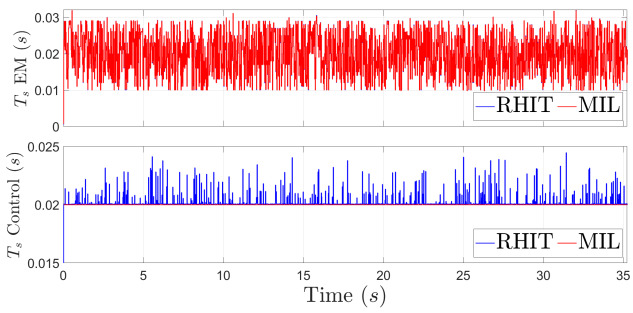


Figure 14.23: (Left and right) DC motors EMC Timestamp EM and Control part - Comparison MIL and RHIT

14.3.3 Target position $\tilde{\xi}$ = 1 m (round trip), target ang. speed $\tilde{\omega}_z$ = 0 $\frac{\text{rad}}{\text{s}}$, dist. rej.

MIL simulation test results:

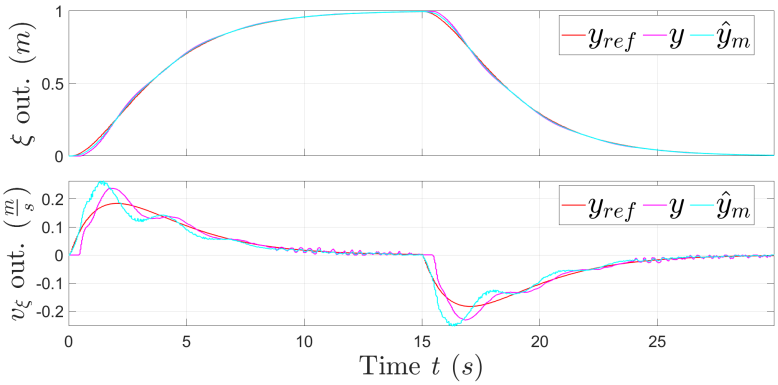


Figure 14.24: MIL Long. position EMC outputs y_{ref} (red), y (magenta) and $\hat{y}_m$ (cyan)

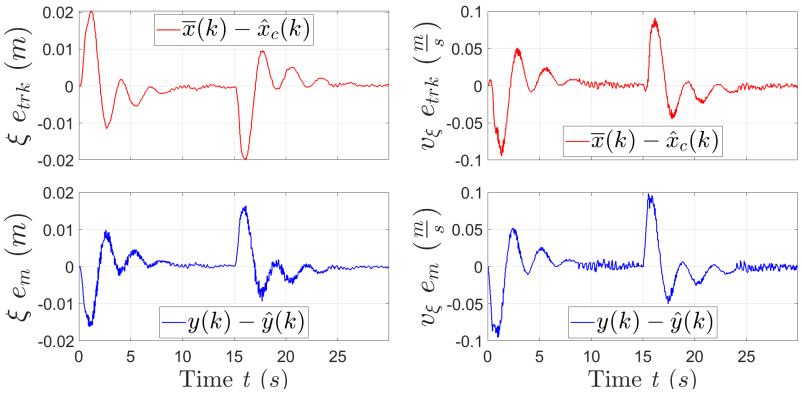


Figure 14.25: MIL Long. position EMC tracking e_{trk} and model e_m errors

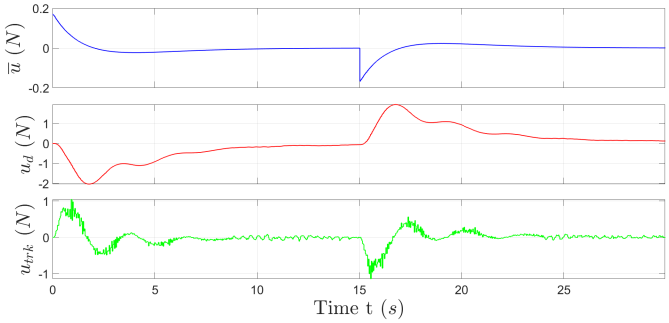


Figure 14.26: MIL Long. position EMC control input terms $\overline{u}, u_d$ and u_{trk} (from top to bottom) - Disturbance rejection

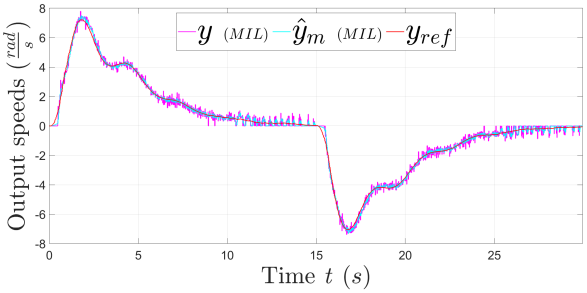


Figure 14.27: MIL Left motor EMC outputs y (magenta), $\hat{y}_m$ (cyan) and y_{ref} (red)

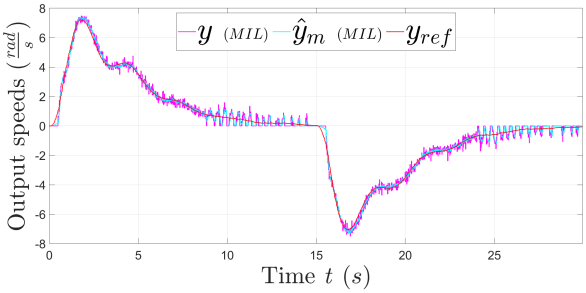


Figure 14.28: MIL Right motor EMC outputs y (magenta), $\hat{y}_m$ (cyan) and y_{ref} (red)

Comparison between MIL and RHIT:

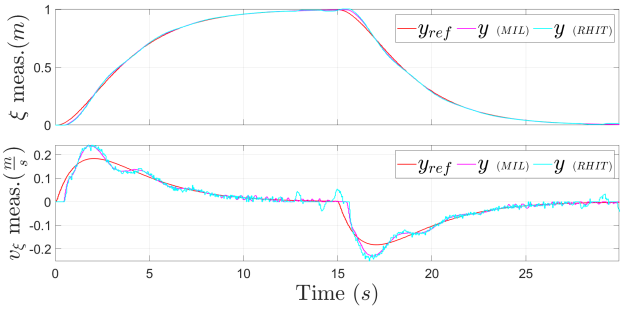


Figure 14.29: Long. position EMC measured outputs y (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

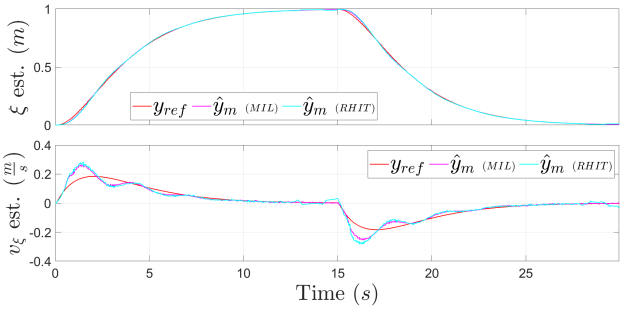


Figure 14.30: Long. position EMC estimated outputs $\hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

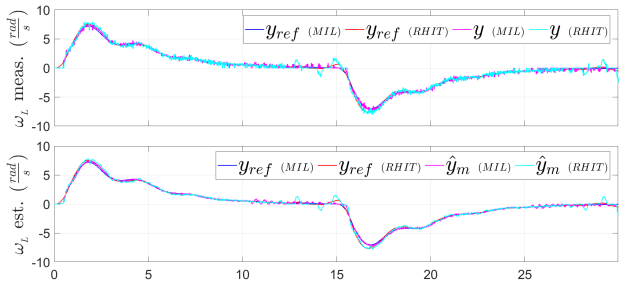


Figure 14.31: Left DC motor EMC measured outputs $y, \hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

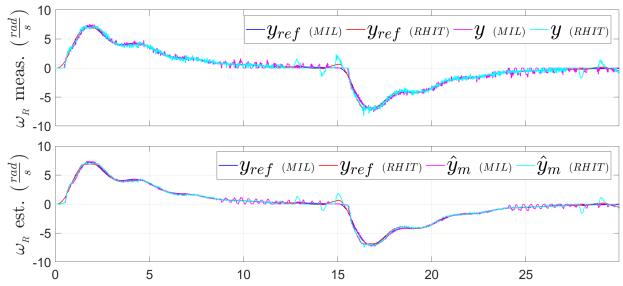


Figure 14.32: Right DC motor EMC measured outputs $y, \hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

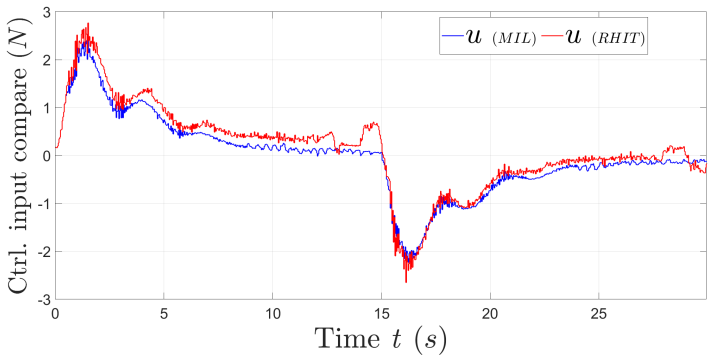


Figure 14.33: Long. position EMC virtual control input u - Comparison MIL and RHIT

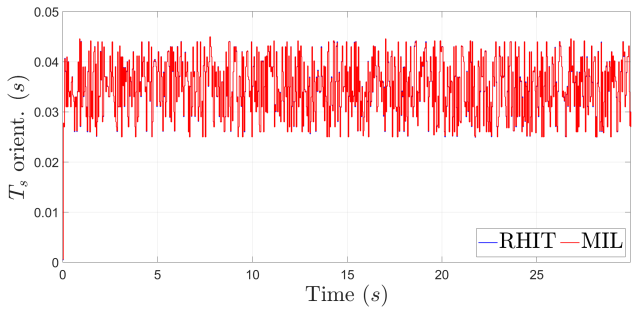


Figure 14.34: Long. position EMC Timestamp - Comparison MIL and RHIT

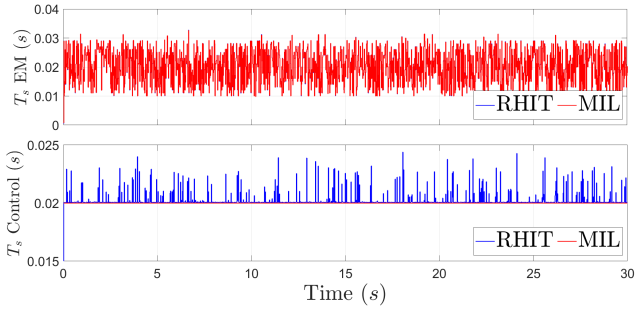


Figure 14.35: (Left and right) DC motors EMC Timestamp EM and Control part - Comparison MIL and RHIT

14.3.4 Target position $\tilde{\xi}$ = 1 m (round trip), target ang. speed $\tilde{\omega}_z$ = 0 $\frac{\text{rad}}{\text{s}}$, No dist. rej.

MIL simulation test results:

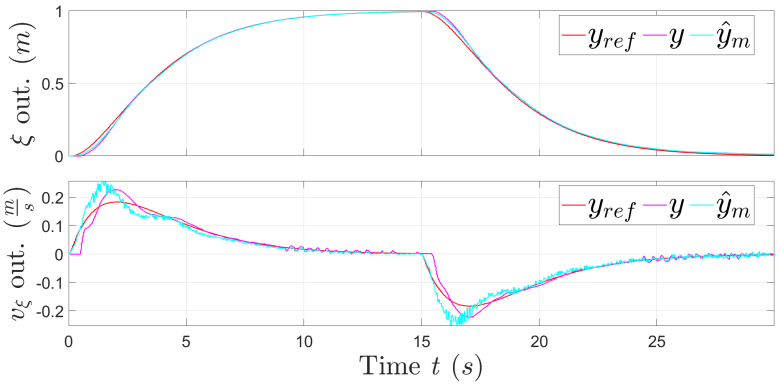


Figure 14.36: MIL Long. position EMC outputs y_{ref} (red), y (magenta) and $\hat{y}_m$ (cyan)

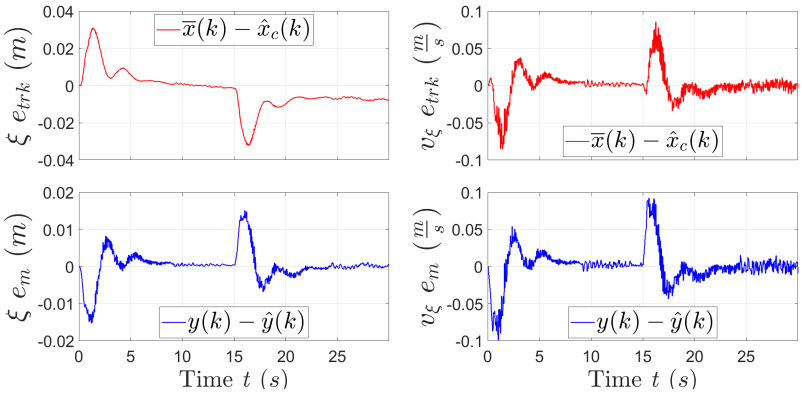


Figure 14.37: MIL Long. position EMC tracking e_{trk} and model e_m errors

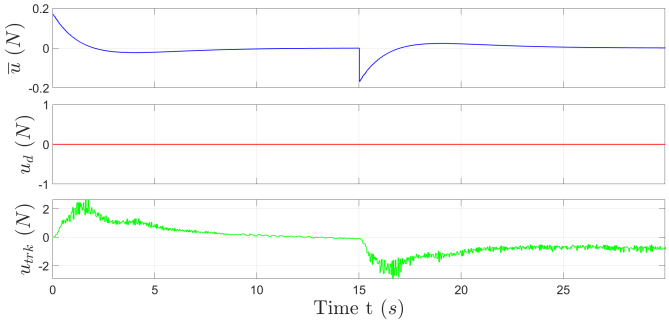


Figure 14.38: MIL Long. position EMC control input terms $\overline{u}, u_d$ and u_{trk} (from top to bottom) - No disturbance rejection

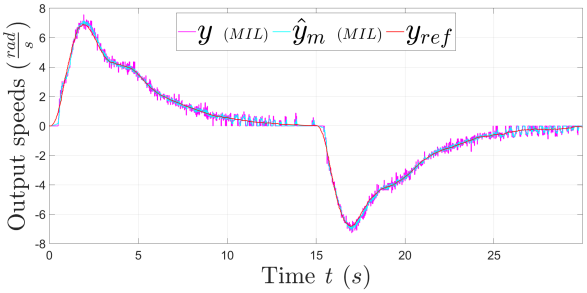


Figure 14.39: MIL Left motor EMC outputs y (magenta), $\hat{y}_m$ (cyan) and y_{ref} (red)

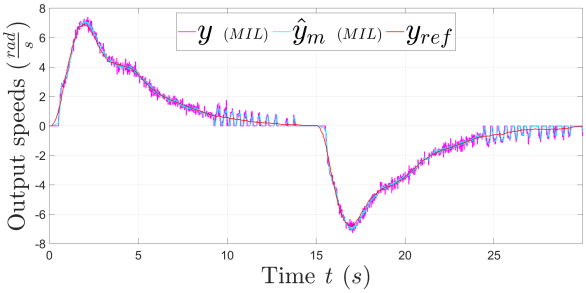


Figure 14.40: MIL Right motor EMC outputs y (magenta), $\hat{y}_m$ (cyan) and y_{ref} (red)

Comparison between MIL and RHIT:

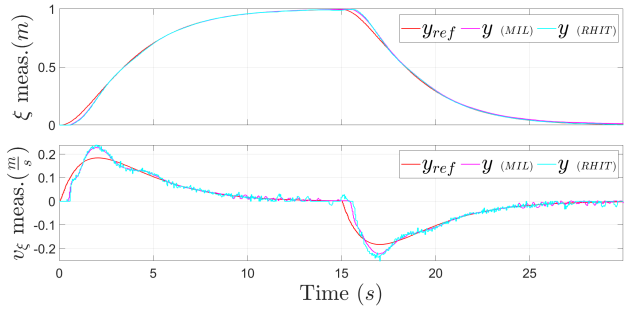


Figure 14.41: Long. position EMC measured outputs y (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

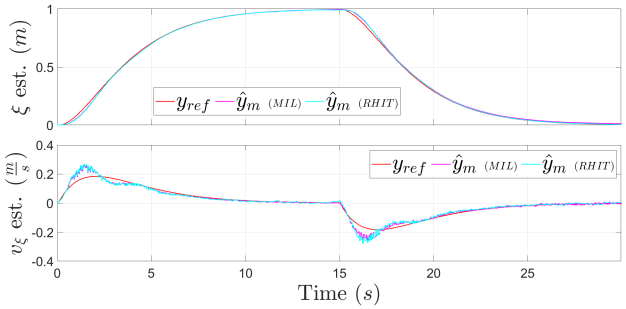


Figure 14.42: Long. position EMC estimated outputs $\hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

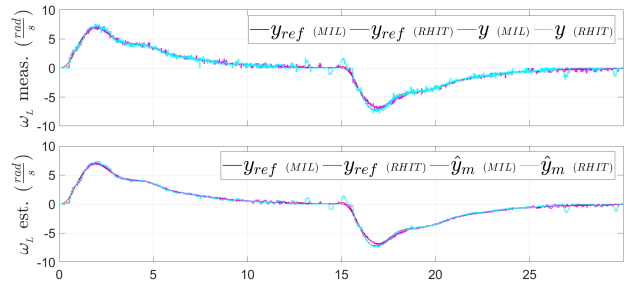


Figure 14.43: Left DC motor EMC measured outputs $y, \hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

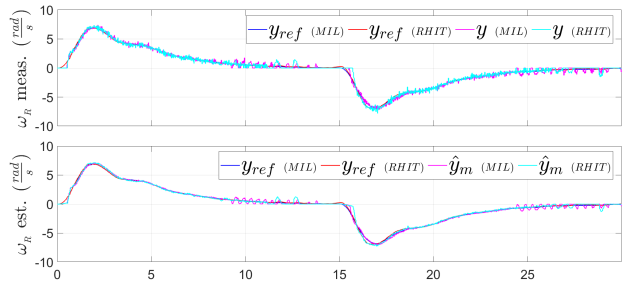


Figure 14.44: Right DC motor EMC measured outputs $y, \hat{y}_m$ (magenta MIL, cyan RHIT) - Comparison MIL and RHIT

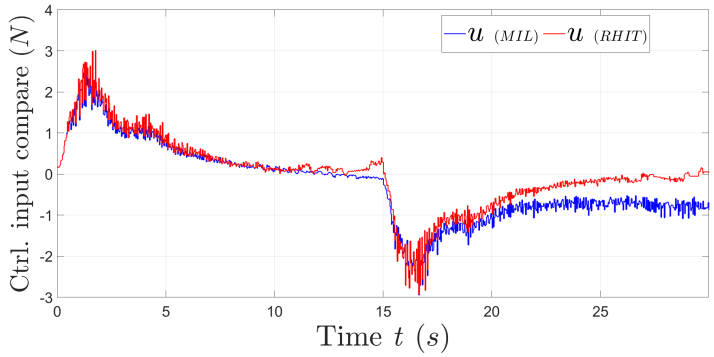


Figure 14.45: Long. position EMC virtual control input u - Comparison MIL and RHIT

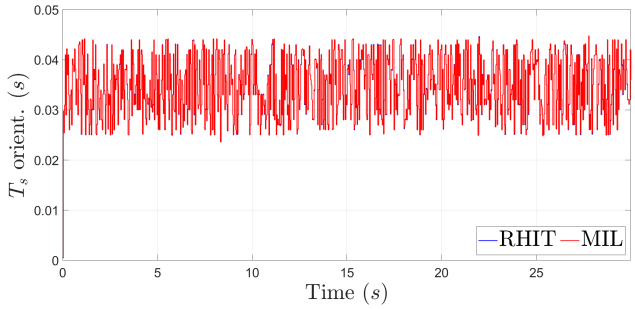


Figure 14.46: Long. position EMC Timestamp - Comparison MIL and RHIT

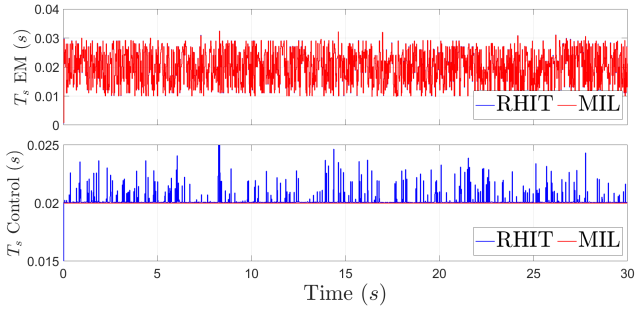


Figure 14.47: (Left and right) DC motors EMC Timestamp EM and Control part - Comparison MIL and RHIT

Chapter 15

Motors and longitudinal position EMC - RHIT results

The EMC system to control the longitudinal position is very similar to the one for orientation EMC only, basically with a gain difference (total robot mass instead of total inertia). Hence the generated SW code to be implemented in the Raspberry is quite the same.
2 possibilities are available to measure the robot position ξ and speed v_ξ in longitudinal direction:

1. The IMU accelerometer may be used to recover speed and position by double integration. However raw accelerometer data is affected by large frequency spectrum noise, and 2 integrations unavoidably leads to wrong values. To avoid this problem a sensor fusion with another sensor can be used, for example using the motor encoders, but this is not treated in this thesis work.

2. (This is the procedure used) Using only the motor encoders to find the angular motor speeds, then kinematic equations to recover the long. speed and position by integration (this procedure is also called *odometry*). Since it is made an integration to pass from speed to position, also in this case the derivation and integration errors accumulate over time: however the distance longitudinally covered for tests is not very high (only some meters), and we can assume that the estimation is sufficiently precise.

Again a hierarchic structure very similar to the one used for orientation EMC is exploited (cfr. Figure 12.1), however robot plant now requires only DC motor encoders to obtain the measurements: to obtain the longitudinal position and speed ξ, v_ξ kinematic equations are used a second time (cfr. Equation (3.15)), and a DT derivative to obtain wheel angular speeds ω_R, ω_L. Making reference to Figure 15.1, we can recall the main passages of the entire hierarchic system loop:

- From long. position control model, long. position $\hat{\xi}$ and speed $\hat{v}_\xi$ are estimated. Since only longitudinal distance is controlled, total velocity coincides with the one along ξ axis, $\hat{v} = \hat{v}_\xi$.

- Conversion block allows to pass from estimated $\hat{v}$ to angular motor speeds, which are used as *target* speeds entering the DC motor control models. Conversion again is based on kinematic equations, one input is $\hat{v}$, the other is $\hat{\omega}_z = 0\,\mathrm{rad/s}$, since only position control is considered.

- DC motors EMC give as output the wheel angular speeds and voltage control inputs u_R, u_L for real motors.

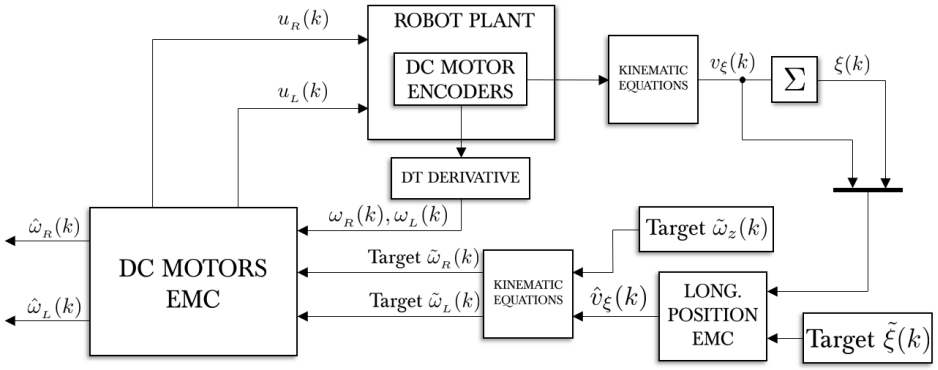


Figure 15.1: Hierarchic structure for RHIT - Longitudinal position and DC motors EMC

- Position and speed measurements from long. position fine model are computed at every sampling time step by using the wheel encoder angular positions and kinematic equations, to enter long. position EMC. At the same time from motor encoders also the real wheel speed measurements ω_R, ω_L are computed to enter DC motors EMC.

15.1 General RHIT test settings

As for orientation EMC RHIT tests, model and tracking errors e_m and e_{trk} are very important for estimated position $\hat{\xi}$, but for $\hat{v}_\xi$ estimate they are not crucial since we are interested mainly in final longitudinal position. Only warning is to maintain the linear estimated speed in a bounded error range.
Main trajectories followed are straight lines till $1\,\mathrm{m}$ or $2\,\mathrm{m}$ (sometimes coming back to initial position), and 0 rad angle trajectory. Predictable orientation errors may arise during tests because only position is controlled.
2 HW timers work for both the 2 motors (EM and control part) and only 1 for position control model.
In every test variable sample time is considered for both control models: in particular the already tested random sampling times $T_{EM} = [0.01, 0.03]$ s and $T_{ctrl} = 0.02\,\mathrm{s}$ for EM and control EMC block parts are used. Instead for position control model $T_p = [0.025, 0.045]$ s variable sampling time is considered. Now encoder is used to estimated the position but in the future IMU can be integrated, so with this timestamp range we guarantee minimum IMU sampling rate $T_{IMU} = 0.021\,\mathrm{s}$.
In the next figures are presented the Timestamp ranges used in all tests (some differences occurs among tests, but the range remains still the same):

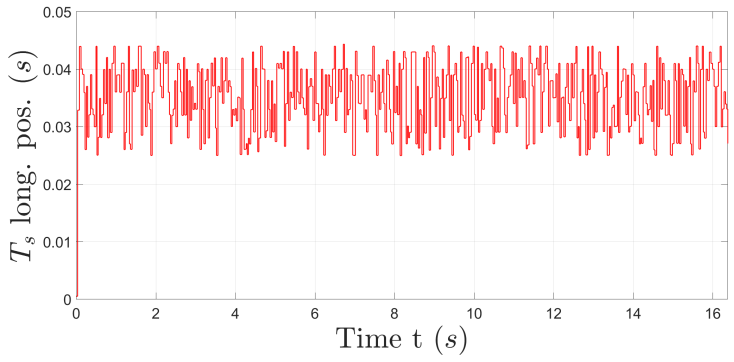


Figure 15.2: Position EMC Timestamp

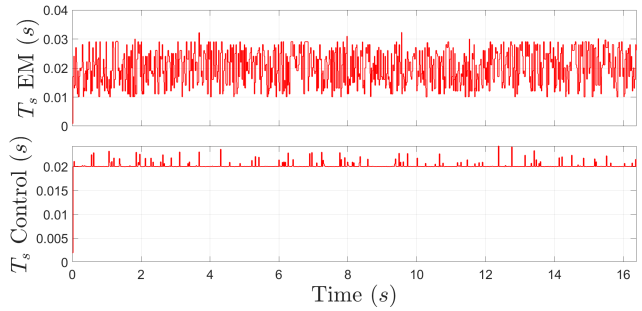


Figure 15.3: Left motor EMC Timestamp

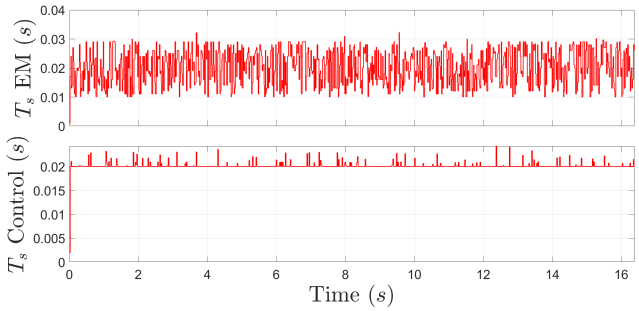


Figure 15.4: Right motor EMC Timestamp

In Figure 15.3 and Figure 15.4, Timestamp of Control part is $0.02\,\mathrm{s}$ with some spikes in the measurements due to the 3 HW timers running simultaneously for the tests. As for orientation only control model, they don't affect the DC motors EMC results.

There are considered again 2 static FB measure-driven decomposition noise estimators. CT eigenvalues when the robot is covering a $1\,\mathrm{m}$ distance are:

Eigenvalue type	Continuous eigenvalues
Reference μ_R	-0.5×2
1st noise estimator (static) $\mu_{N,1}$	-6
2nd noise estimator (static) $\mu_{N,2}$	-6×2
Control μ_K	$-30, -10$

Table 15.1: CT eigenvalues Longitudinal position EMC

When the distance covered is $2\,\mathrm{m}$, only the reference eigenvalues changes from -0.5×2 to -0.2×2, in order to reduce the speed required for the DC motors and avoid saturations.

Virtual force input is saturated when reaching $10\,\mathrm{N}$, it's a reasonable value as maximum values of 1.5–$3\,\mathrm{N}$ are seen in every test. Version 1.2 EMC is used for both left and right DC motors.

15.2 Plot results

In all tests the tracking and model errors are very low for position, with a maximum of some cm. The ones for longitudinal speeds are not low (maximum of $\approx 0.15\,\mathrm{m/s}$), but at the same time not too high guaranteeing a quite smooth robot speed trajectory behaviour.

The last 2 plot tests are made considering a one meter robot round-trip, the first when disturbance rejection of position control model is active, and the second when it is not active. There are no significant differences in both position and velocity tracking and model errors, only a slightly improvement in favour of the disturbance rejection. Only more variable virtual input force (but in the same range) is found when the disturbance rejection is not present, because since disturbance rejection input component u_d is not present anymore the other input tracking component u_{trk} becomes higher to compensate.

This can be seen in Figure 15.22 and Figure 15.30 where the maximum peak for u_{trk} passes from $1\,\mathrm{N}$ to $3\,\mathrm{N}$, which difference is more or less the maximum peak of u_d when the disturbance rejection is present.

15.2.1 Trajectory distance of 1 m

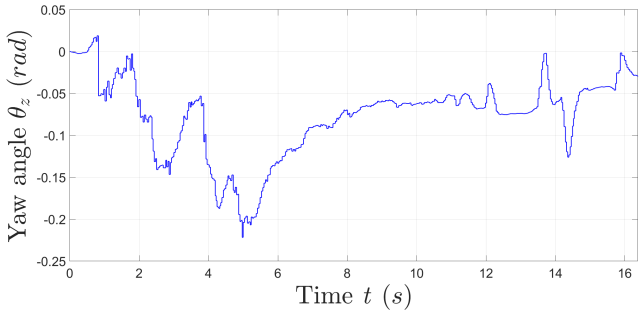


Figure 15.5: IMU θ_z measurement

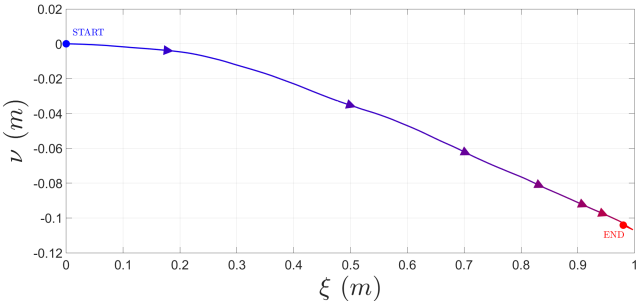


Figure 15.6: Robot position (using motor encoders and kin. eqs)

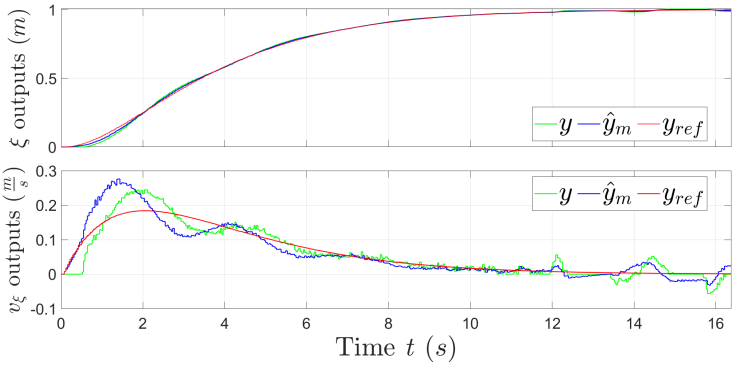


Figure 15.7: Longitudinal position and speed EMC y, $\hat{y}_m$ and y_{ref}

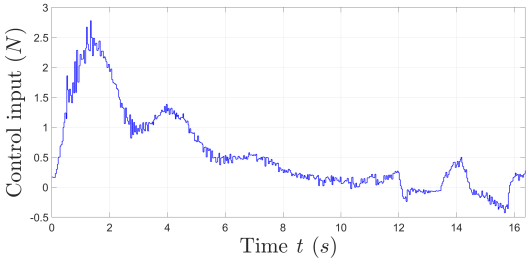


Figure 15.8: Robot force control input (virtual)

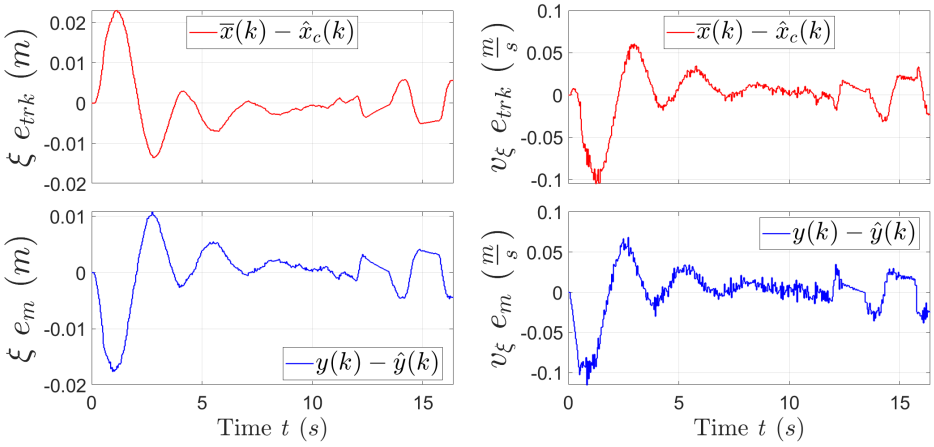


Figure 15.9: Tracking e_{trk} and model e_m errors for x (1st column) and v_x (2nd column)

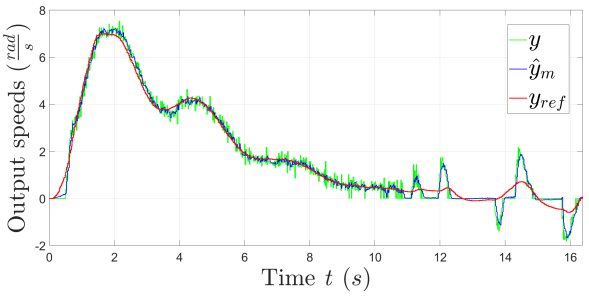


Figure 15.10: Right motor EMC y, $\hat{y}_m$ and y_{ref}

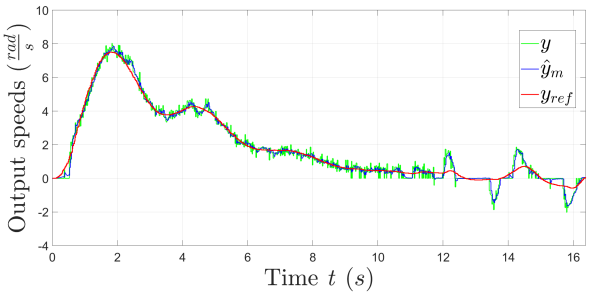


Figure 15.11: Left motor EMC y, $\hat{y}_m$ and y_{ref}

15.2.2 Trajectory distance of 2 m

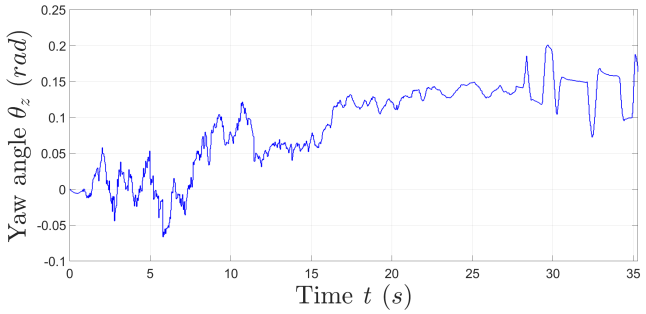


Figure 15.12: IMU θ_z measurement

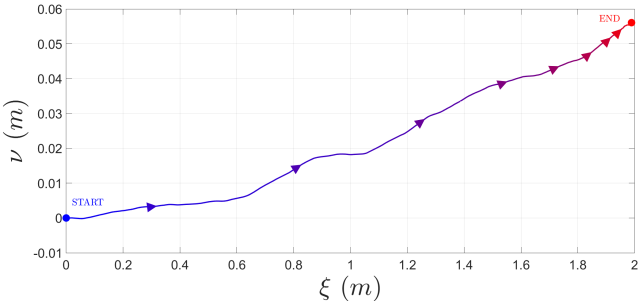


Figure 15.13: Robot position (using motor encoders and kin. eqs)

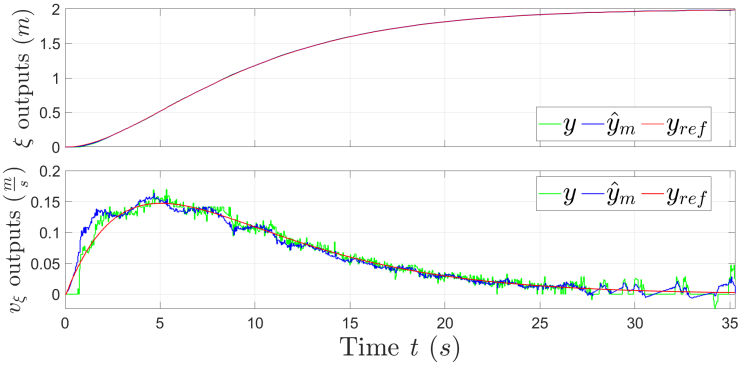


Figure 15.14: Longitudinal position and speed EMC y, $\hat{y}_m$ and y_{ref}

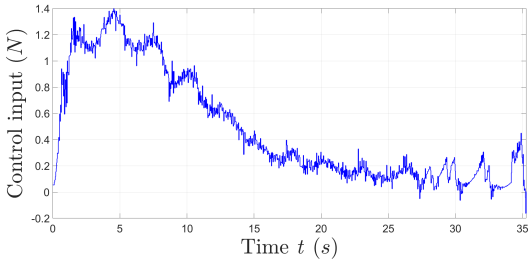


Figure 15.15: Robot force control input (virtual)

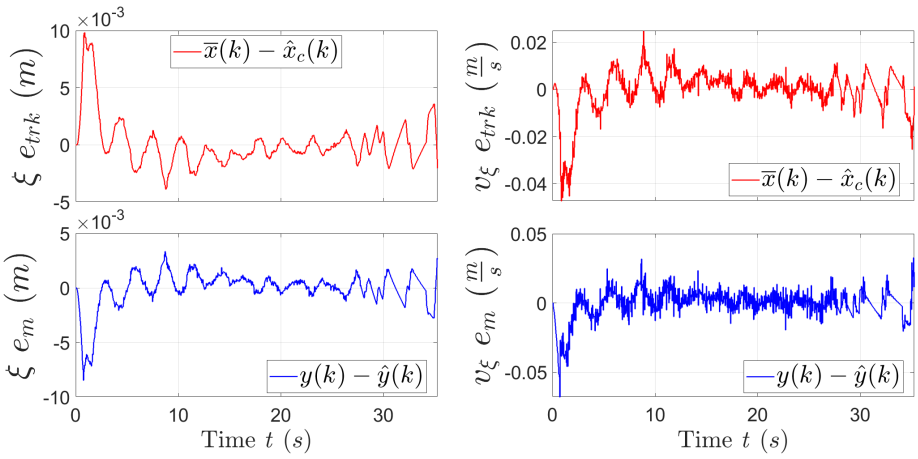


Figure 15.16: Tracking e_{trk} and model e_m errors for x (1st column) and v_x (2nd column)

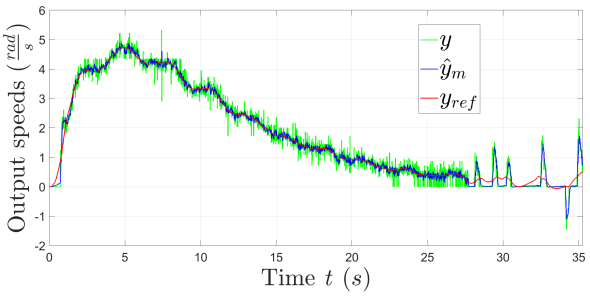


Figure 15.17: Right motor EMC y, $\hat{y}_m$ and y_{ref}

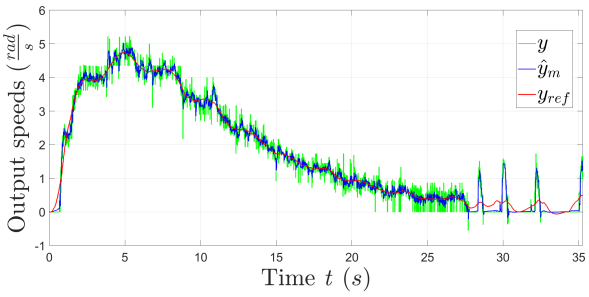


Figure 15.18: Left motor EMC y, $\hat{y}_m$ and y_{ref}

15.2.3 Trajectory distance of 1 m and return to initial position - Disturbance rejection

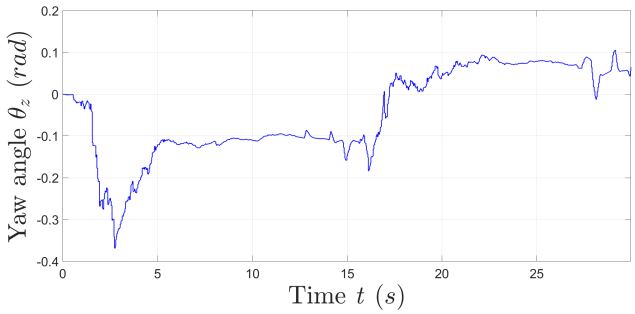


Figure 15.19: IMU θ_z measurement

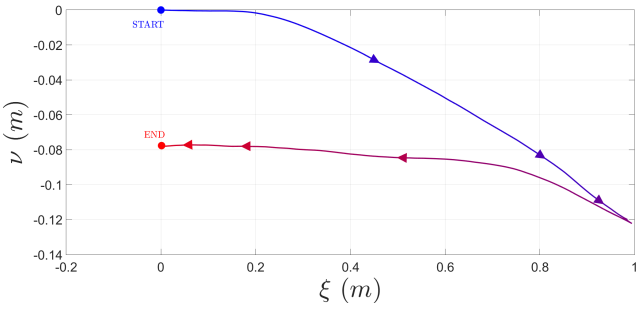


Figure 15.20: Robot position (using motor encoders and kin. eqs)

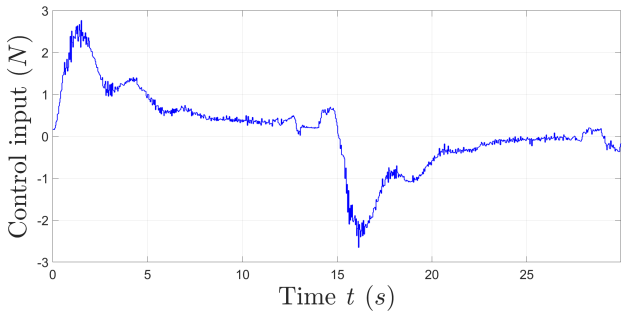


Figure 15.21: Robot force control input (virtual)

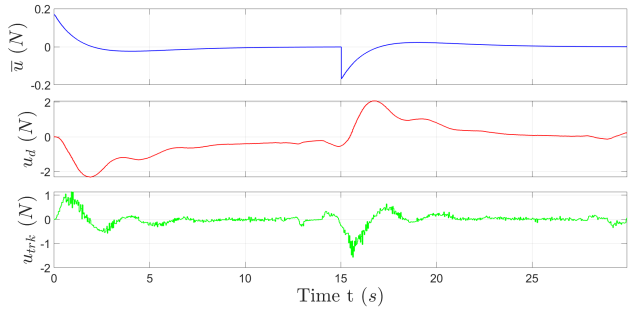


Figure 15.22: Robot force control input components (virtual): $\overline{u}$, u_d and u_{trk} (respectively from top to bottom)

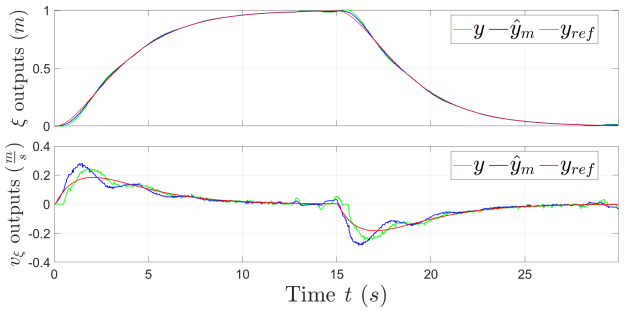


Figure 15.23: Longitudinal position and speed EMC y, $\hat{y}_m$ and y_{ref}

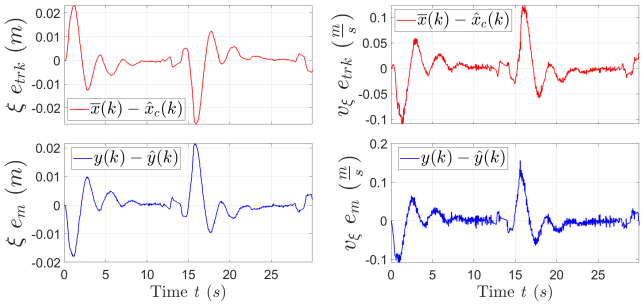


Figure 15.24: Tracking e_{trk} and model e_m errors for x (1st column) and v_x (2nd column)

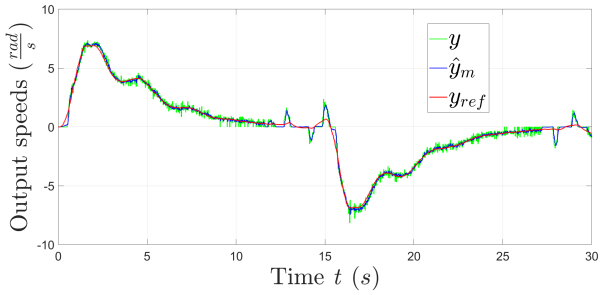


Figure 15.25: Right motor EMC y, $\hat{y}_m$ and y_{ref}

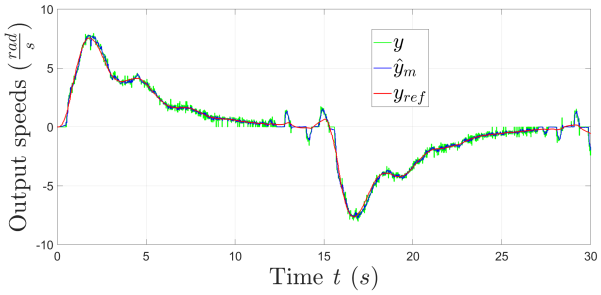


Figure 15.26: Left motor EMC y, $\hat{y}_m$ and y_{ref}

15.2.4 Trajectory distance of 1 m and return to initial position - No disturbance rejection

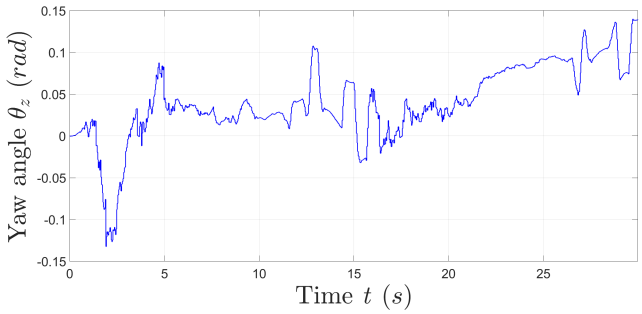


Figure 15.27: IMU θ_z measurement

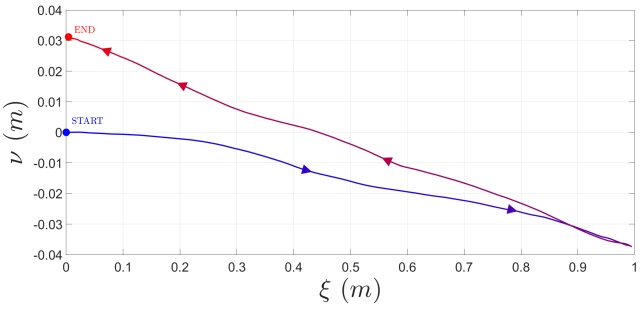


Figure 15.28: Robot position (using motor encoders and kin. eqs)

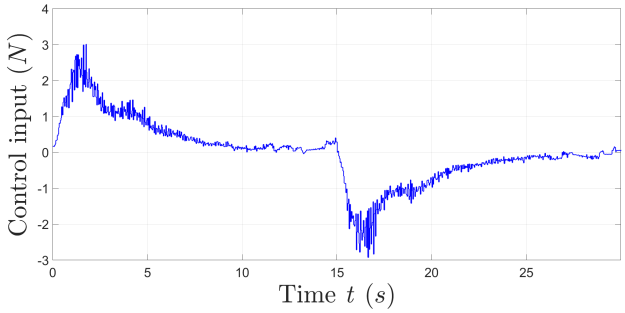


Figure 15.29: Robot force control input (virtual)

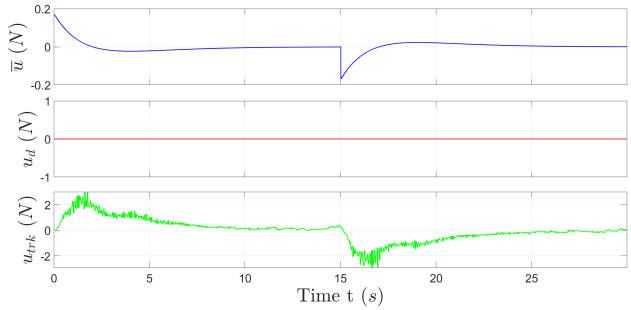


Figure 15.30: Robot force control input components (virtual): $\overline{u}$, u_d and u_{trk} (respectively from top to bottom)

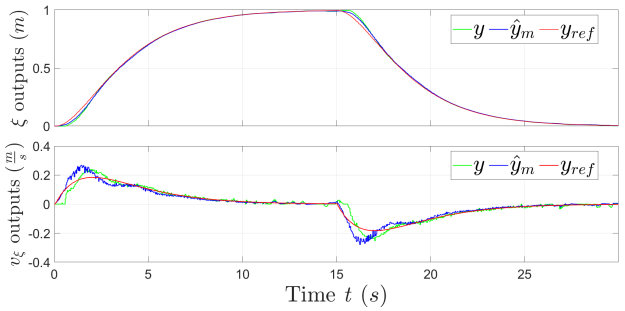


Figure 15.31: Longitudinal position and speed EMC y, $\hat{y}_m$ and y_{ref}

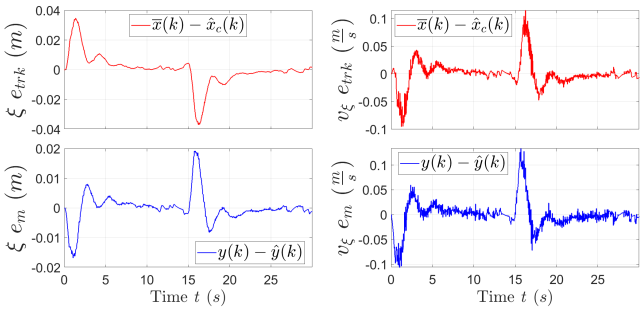


Figure 15.32: Tracking e_{trk} and model e_m errors for x (1st column) and v_x (2nd column)

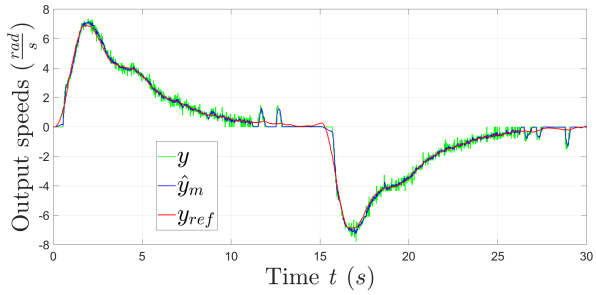


Figure 15.33: Right motor EMC y, $\hat{y}_m$ and y_{ref}

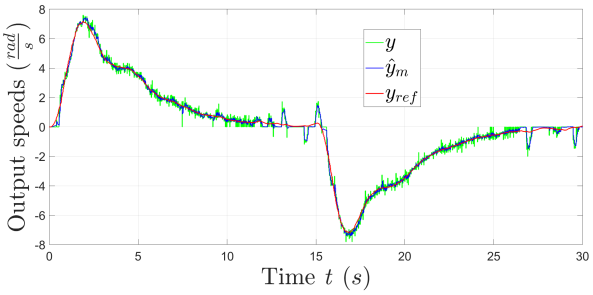


Figure 15.34: Left motor EMC y, $\hat{y}_m$ and y_{ref}

Part V

Robot 2D space position EMC

Chapter 16

Motors, orientation and long. position robot EMC - Theory and RHIT results

The 2 EMC for orientation and longitudinal position for GoPiGo3 discussed in Chapter 10 and Chapter 13 are combined together to control robot in longitudinal and lateral position at the same time.
Regarding how previous EMC models are built, most simple way to control robot in 2D space is to use longitudinal position ξ (and speed v_ξ) and *orientation angle* θ_z *(and angular speed* ω_z*)* as robot coordinate targets: it's quite similar to use polar coordinates (with position in one axis instead the radius). In this way already discussed control models work independently, we need only to make a conversion outside the control models to pass from inertial RF to body RF and viceversa. This is the structure used to make EMC tests in this thesis.
Conversely, selecting Cartesian coordinates as targets (ξ, ν) more complex structure must be selected, which is not covered in this thesis work.
Notation: Symbols ξ, ν are used as Cartesian coordinates to avoid misunderstandings with x symbol, used as state vector for EMC.

16.1 Orientation and long. position EMC hierarchical structure scheme

The last 2 robot orientation and long. position EMC can be easily combined together to make a unique structure, and move robot in every point of 2-D plane.
Only the scheme for RHIT is presented (Figure 16.1), since no MIL tests are done for this control structure: indeed to make a simulation quite complex fine model combining both orientation and position robot dynamics must be built, and this is not covered in this thesis field.
The simplest procedure is to run directly the 2 control models in independent way, one controlling the angle θ_z for ν-axis lateral position and the other ξ-axis longitudinal position. The other way is to control ν position instead of angle θ_z in a more intuitive way, but this condition requires more complex control models (therefore also for RHIT and not only for MIL tests this structure is not considered).
Since the 2 orientation and long. position EMC work independently, explanation for the most parts of this 2D position EMC hierarchic structure can be found at beginning of Chapter 12 and Chapter 15.
The specific settings for the control models derives from best results obtained when they are tested alone:

- **Orientation EMC**: Magnetometer information in sensor fusion to measure the orientation yaw angle θ_z, noise estimator splitted in 2 static FB estimators, using measure-driven decomposition procedure.

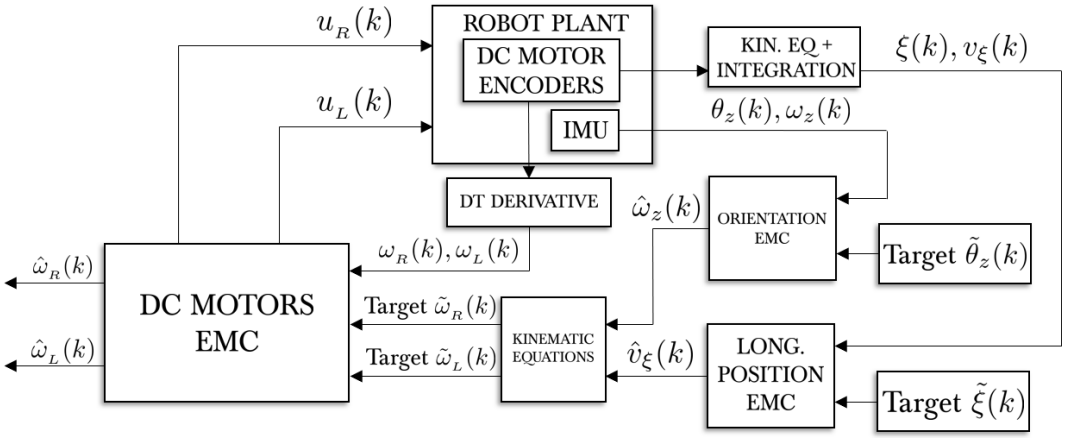


Figure 16.1: Hierarchic structure for RHIT - Longitudinal position, orientation and DC motors EMC

- **Position EMC**: Linear position and speed obtained using DC motors encoders, 2 static FB noise estimators (measure-driven decomposition).

- **DC motors**: Version 1.2 for both left and right.

16.2 General RHIT settings

Main trajectories followed:

- Longitudinal position $\tilde{\xi} = 2\,\mathrm{m}$, maintaining $\tilde{\theta}_z = 0\,\mathrm{rad}$ angle orientation.

- Longitudinal position $\tilde{\xi} = 2\,\mathrm{m}$ and $\tilde{\theta}_z = 50^\circ \approx 0.87\,\mathrm{rad}$ angle orientation.

2 HW timers work for both the 2 motors (EM and control part) and only 1 for both orientation and position control models (in order to avoid a 4^{th} HW timer running simultaneously with the others, which more likely can add interferences to timestamps).
Timestamp for both is again variable to understand working conditions of EMC, for DC motors they are $T_{EM} = [0.01, 0.03]$ s and $T_{ctrl} = 0.02\,\mathrm{s}$, instead for orientation and position $T_o = T_p = [0.025, 0.045]$ s (to make sure that at least one IMU measurement is taken at every step, since its sample rate is $[47, 47.5]$ Hz). An example taken from one of the tests (straight line till $2\,\mathrm{m}$ robot trajectory):

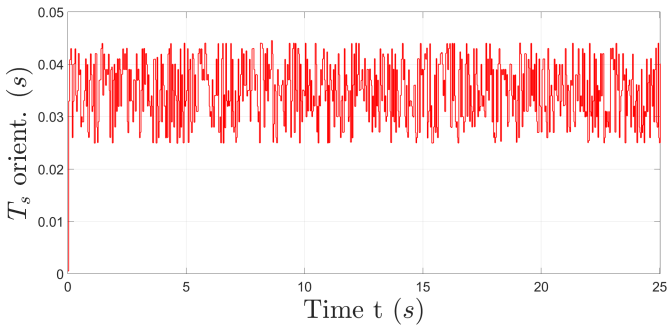


Figure 16.2: Orientation and Long. position EMC Timestamp

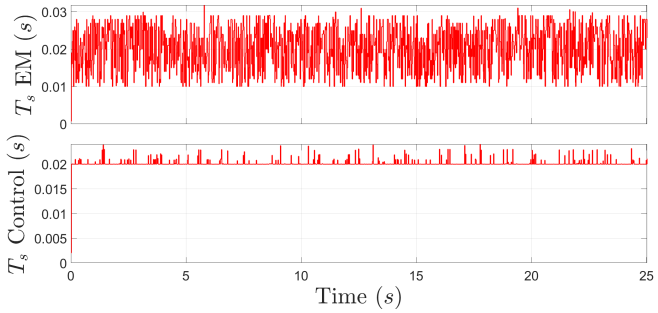


Figure 16.3: Left motor EMC Timestamp

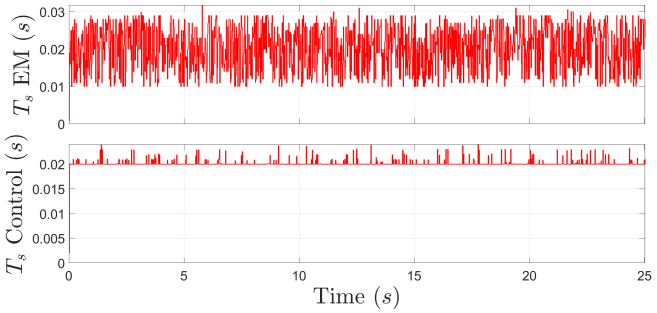


Figure 16.4: Right motor EMC Timestamp

The spikes in DC motor timestamps (Figure 16.3 and Figure 16.4) do not interfere so much with the results. They are due to the presence of many timers running simultaneously.
Regarding EMC control input disturbance rejection, it is already verified testing orientation and position EMC separately that the absence of disturbance rejection term does not influence badly the tracking and model errors (because the disturbances for robot orientation/position are small and related only to total mass and inertia parameters). Since the control models work independently even with 2D position control, there is no interest in repeating the tests without disturbance rejection.
In this tests the robot has also a lateral movement (i.e. when the desired angle $\theta_z \neq 0$), hence is necessary to pay attention to the reference frames (RF) used. As long as the robot goes only in longitudinal direction the longitudinal velocity v_ξ and position in ξ are exactly the total ones, but if lateral motion is present a transformation is needed to pass from body RF to inertial RF, and viceversa.
Making references to already explained reference frames for GoPiGo3 robot in Figure 3.6 (with BF coincident with IF at the beginning of every test, for simplicity) and to hierarchical structure in Figure 16.1, transformation is done as follows:

- In longitudinal position EMC the computed virtual force input and the states are in Robot Inertial RF.

- When the passage from estimated longitudinal speed to wheels angular speeds is needed, a conversion is performed to obtain the Body RF longitudinal speed $\hat{v}_\xi^{BF}$ from Inertial RF one $\hat{v}_\xi^{IF}$, estimated by long. position EMC. It can be computed as:

$$\hat{v}_\xi^{BF} = \frac{\hat{v}_\xi^{IF}}{\cos\hat{\theta}_z} \tag{16.1}$$

  where $\hat{\theta}_z$ is the estimated angle robot position obtained from orientation EMC. The DC motors EMC target angular speeds are now in Body RF.

- The robot position measurements can be found integrating the following equation:

$$v_\xi^{IF} = \frac{\rho}{2}\left(\omega_R + \omega_L\right)\cos\theta_z \tag{16.2}$$

  Since it is needed the position in inertial RF and wheel speeds measurements ω_L, ω_R are in Body RF, in the last equation there is the multiplication by $\cos\theta_z$ to convert speed in Inertial RF.

Until now IMU sensor fusion for angle θ_z works using all 3 devices present, gyroscope, magnetometer and accelerometer. It is experimented that using the accelerometer is not a good choice because it leads to increased tracking and model errors.

Control model	Eigenvalue type	Value
Orientation	Reference μ_R	-0.3×2
	1st noise estimator (static) $\mu_{N,1}$	-5
	2nd noise estimator (static) $\mu_{N,2}$	-5×2
	Control μ_K	$-30, -20$
Long. Position	Reference μ_R	-0.25×2
	1st noise estimator (static) $\mu_{N,1}$	-5
	2nd noise estimator (static) $\mu_{N,2}$	-5×2
	Control μ_K	-20×2

Table 16.1: CT eigenvalues Orientation and position EMC

The CT eigenvalues considered are: for the 2 DC motors exactly the same used in Table 9.2, instead for position and orientation control models: Max virtual control input force is $F_{v,max} = 10\,\mathrm{N}$, the max virtual control input torque is $\tau_{v,max} = 1\,\mathrm{N\,m}$.

16.3 Plot results

2 tests are made with robot going in longitudinal direction till $2\,\mathrm{m}$ trying to maintain $0\,\mathrm{rad}$ orientation angle, one with and the other without accelerometer presence in angle sensor fusion.
For each test are shown the robot trajectory in ξ, ν Cartesian coordinates, $y, y_{ref}, \hat{y}_m$ outputs for orientation and longitudinal position EMC, the correspondent errors e_m, e_{trk}, torque and force virtual control inputs (orientation and long. position EMC) and $y, y_{ref}, \hat{y}_m$ outputs for left and right DC motors EMC.
Comparing the estimated angles and angular speeds of robot in Figure 16.6 and Figure 16.16 and the relative tracking and model errors (Figure 16.7 and Figure 16.17), it can be seen that IMU measurements are smoother and consequently the estimates more precise with lower e_m and e_{trk} when the accelerometer is NOT used in sensor fusion.
This can be explained either by uncalibrated accelerometer device, for example due to mean value offsets at rest (as it can be seen in Figure 16.15 where the offset respectively for a_ξ, a_ν and a_z are $\epsilon_{a,\xi} = 0.0567\,\mathrm{g}$, $\epsilon_{a,\nu} = 0.0691\,\mathrm{g}$ and $\epsilon_{a,z} = 0.0160\,\mathrm{g}$), or by high noise with high frequency bandwidth (almost like white noise with important amplitude) affecting the accelerometer data when robot moves.
Instead looking the 3rd test when targets are $2\,\mathrm{m}$ for longitudinal position and $50\,^\circ$ for angle trajectory, the problem of making correct conversions between Inertial RF and Body RF arises. We can verify if the conversion is good by looking at the robot force control input of longitudinal position EMC: it must be present a peak of some N at beginning which then reduce to zero when robot approaches position target.
This is almost what happens, by looking Figure 16.30: only problem happens when approaching the target position $F_v =\approx 1/2\ \mathrm{N}$ (different from zero), but it not crucial because the control input is virtual and not directly used, and trajectory can be followed anyway.
Apart from problems discussed above the targets positions are reached with low tracking and model errors. Instead the errors regarding the estimated angles might be improved, for example making the IMU angle measurements more precise.

16.3.1 Trajectory targets $\tilde{\xi}$ = 2 m, $\tilde{\theta}_z$ = 0° - Fusion algorithm WITHOUT accelerometer

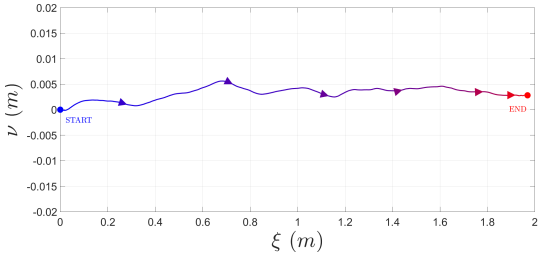


Figure 16.5: Robot position (using motor encoders and kin. eqs)

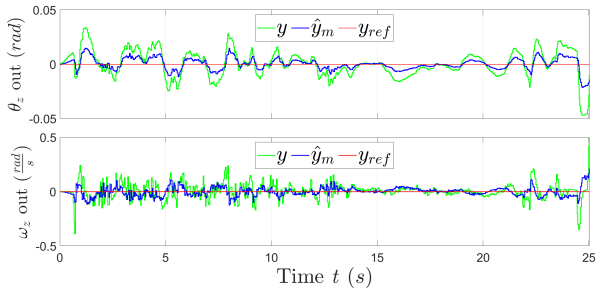


Figure 16.6: Orientation EMC y (green), $\hat{y}_m$ (blue) and y_{ref} (red)

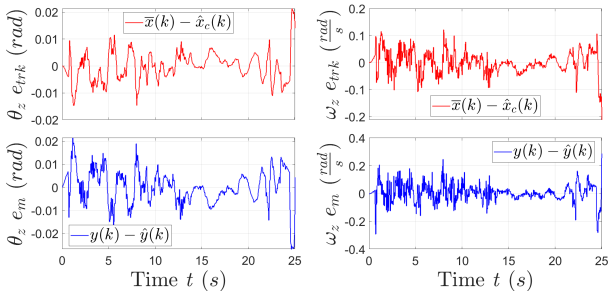


Figure 16.7: Orientation tracking e_{trk} and model e_m errors for θ_z and ω_z

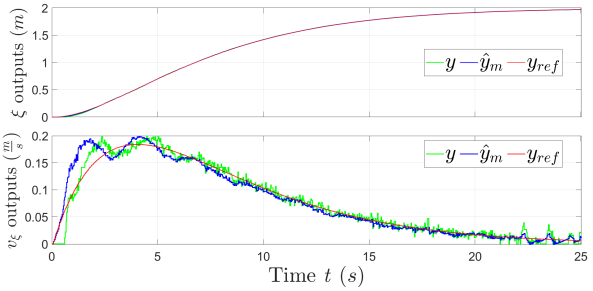


Figure 16.8: Longitudinal position and speed EMC y (green), $\hat{y}_m$ (blue) and y_{ref} (red)

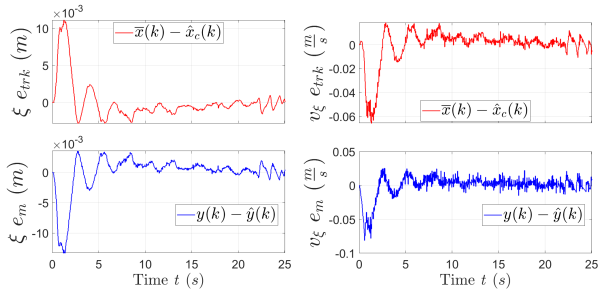


Figure 16.9: Longitudinal position and speed tracking e_{trk} and model e_m errors for ξ and v_ξ

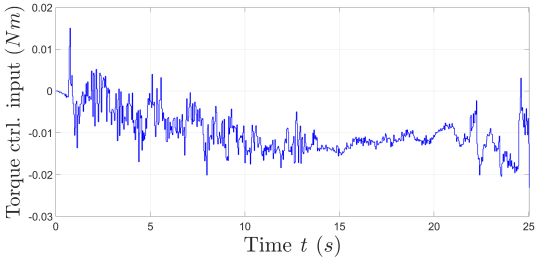


Figure 16.10: Torque control input (virtual)

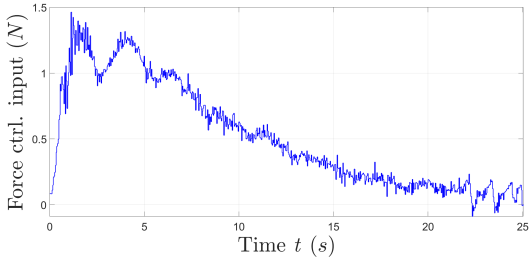


Figure 16.11: Force control input (virtual)

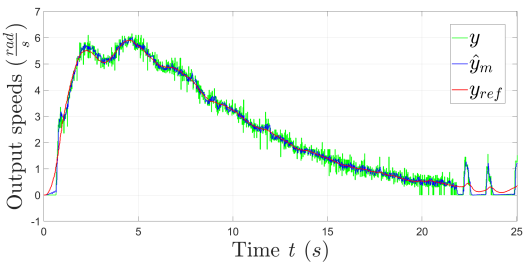


Figure 16.12: Right motor EMC y, $\hat{y}_m$ and y_{ref}

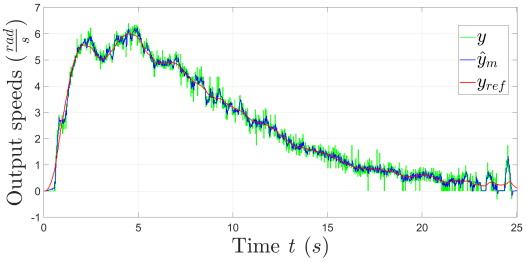


Figure 16.13: Left motor EMC y, $\hat{y}_m$ and y_{ref}

16.3.2 Trajectory distance of 2 m, $\tilde{\theta}_z$ = 0° - Orientation fusion algorithm WITH accelerometer

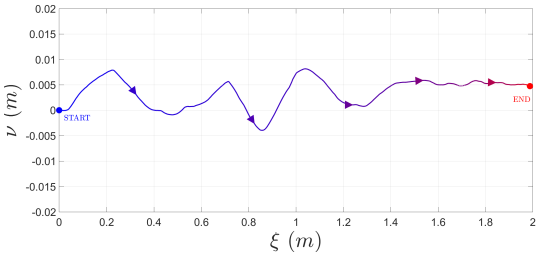


Figure 16.14: Robot position (using motor encoders and kin. eqs)

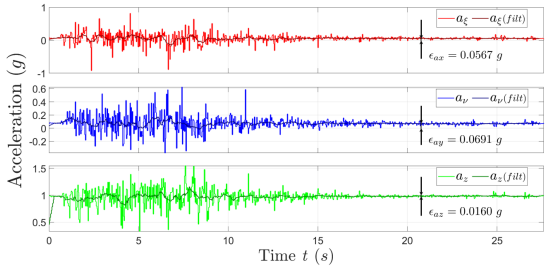


Figure 16.15: Acceleration IMU measurements for sensor fusion, with offsets highlighted

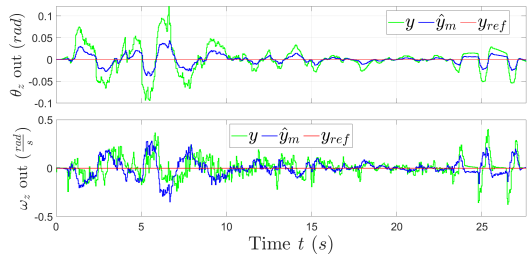


Figure 16.16: Orientation EMC y (green), $\hat{y}_m$ (blue) and y_{ref} (red)

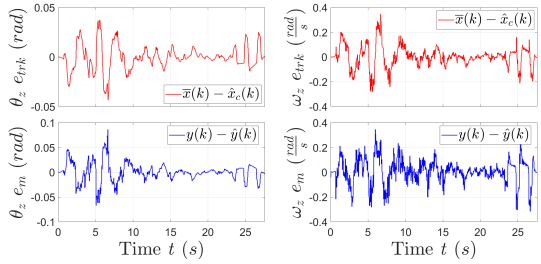


Figure 16.17: Orientation tracking e_{trk} and model e_m errors for θ_z and ω_z

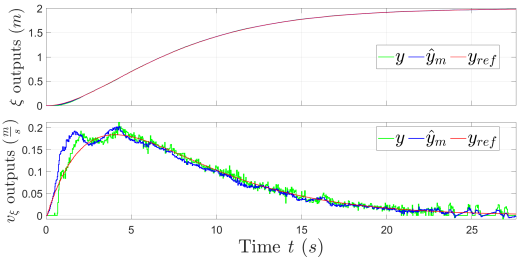


Figure 16.18: Longitudinal position and speed EMC y (green), $\hat{y}_m$ (blue) and y_{ref} (red)

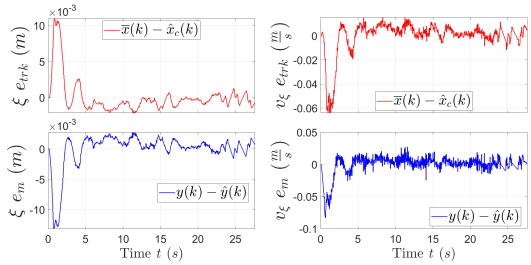


Figure 16.19: Longitudinal position and speed tracking e_{trk} and model e_m errors for ξ and v_ξ

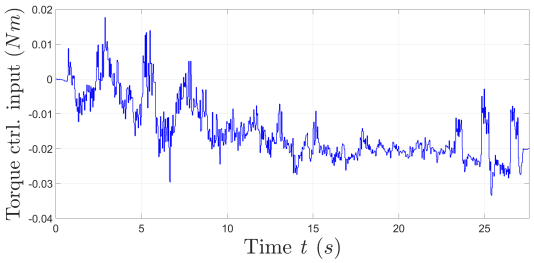


Figure 16.20: Torque control input (virtual)

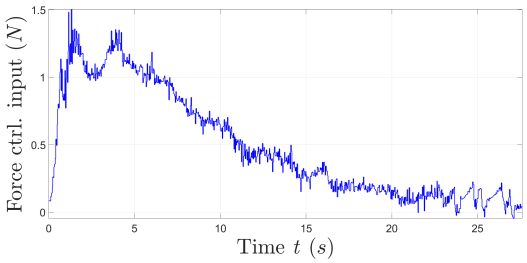


Figure 16.21: Force control input (virtual)

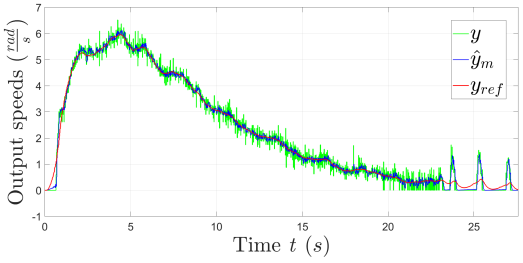


Figure 16.22: Right motor EMC y, $\hat{y}_m$, y_{ref}

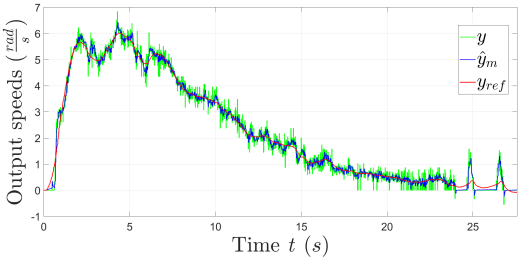


Figure 16.23: Left motor EMC y, $\hat{y}_m$, y_{ref}

16.3.3 Trajectory distance of 2 m, $\tilde{\theta}_z = 50°$ - Fusion algorithm WITHOUT accelerometer

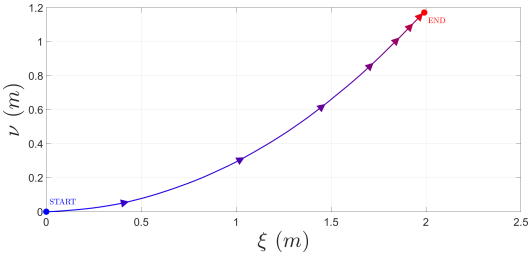


Figure 16.24: Robot position (using motor encoders and kin. eqs)

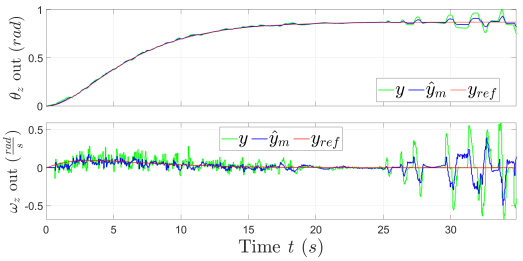


Figure 16.25: Orientation EMC y (green), $\hat{y}_m$ (blue) and y_{ref} (red)

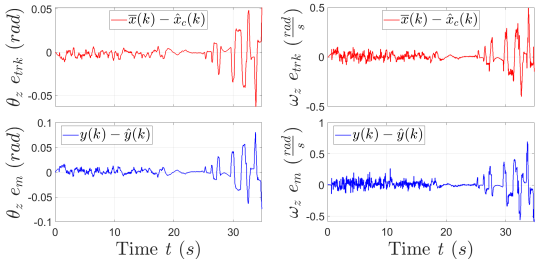


Figure 16.26: Orientation tracking e_{trk} and model e_m errors for θ_z and ω_z

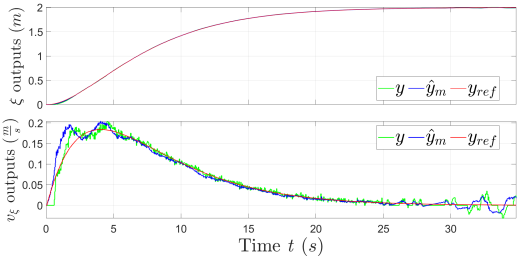


Figure 16.27: Long. position and speed EMC y (green), $\hat{y}_m$ (blue) and y_{ref} (red)

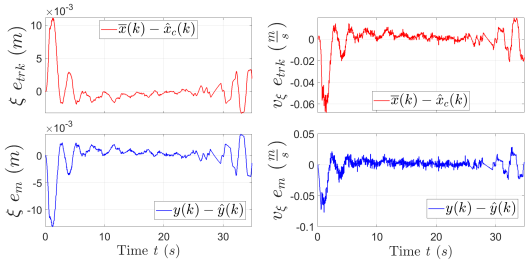


Figure 16.28: Long. position and speed tracking e_{trk} and model e_m errors for ξ and v_ξ

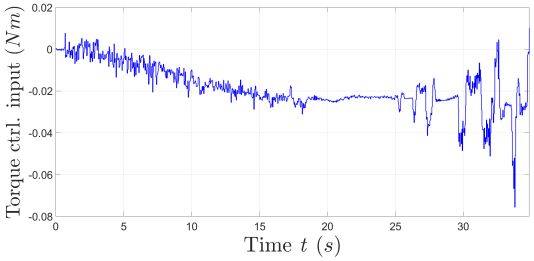


Figure 16.29: Torque control input (virtual)

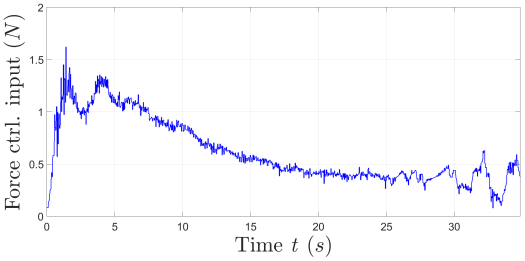


Figure 16.30: Force control input (virtual)

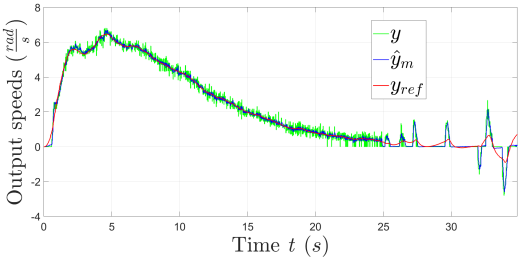


Figure 16.31: Right motor EMC y, $\hat{y}_m$ and y_{ref}

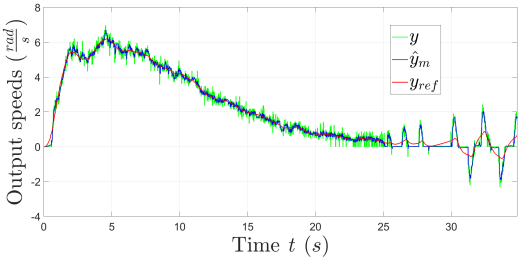


Figure 16.32: Left motor EMC y, $\hat{y}_m$ and y_{ref}

Part VI

Conclusions

Chapter 17

Summary on the results and future work

Real world physical controlled systems are often not synchronous, meaning that their measurements and control inputs are not given in a fixed time, but variable. For example Networked Controlled systems (NCS) are usually characterized by the presence of wireless network between controllers and real plants to be controlled. The presence of delays and package dropouts in data network access or transmission lead to disturbances added to controlled system.
In this field comes in handy the model-based technique called *Embedded Model Control (EMC)*, based on exploiting a simplified version of the physical system to be controlled, the Embedded Model (EM). The peculiarity of this method with respect others is the presence of a disturbance dynamics related to physical plant. This control is also able to reduce the disturbances using plant measurement output informations.
Objective of the thesis is to verify EMC technique effectiveness with asynchronous time physical systems, using a real robot plant system: a differential-drive robot called GoPiGo3 is used to this aim, produced by Dexter industries company, [17].
Two main kind of tests are performed:

- The first is a simulation of both the robot plant and control models, Model-in-the-Loop (MIL) test done using MathWorks Simulink.

- The second is direct experimental implementation of the control model using GoPiGo3 robot, called by the author Robot Hardware Implementation Test (RHIT). It is made basically generating C++ code from Simulink models and loading it in robot GoPiGo3 (which is equipped with a Raspberry board).
  To mimic asynchronous timing conditions, a SW code to generate random timestamps in a desired range is build, then Raspberry HW timers are used to step controllers/robot plant basing on the obtained variable sampling time.

With EMC technique are controlled the robot DC motors to move the 2 wheels, then robot longitudinal position and orientation first separately to make robot follow a straight line or a circular trajectory, finally both orientation and position to move the robot in 2D space. A non-conventional hierarchic structure is used to connect EMC of DC motors and orientation/position, exploiting the EMC estimated states as robot movement targets.
The outputs of EMC are estimation of some plant states (wheel speeds, robot vertical angle and position in space). Results can be mainly judged comparing the EMC estimation states with the reference trajectory one wants to track (*tracking error*), and the EMC estimated outputs with real plant output measurements (*model error*, since it's necessary that EM model is near to real plant). The lower the errors are, better is the control.

For all tests they are sufficiently low, especially for DC motors because the difference between EMC and real plant is quite huge, for the presence of many neglected dynamics related to inertia and frictions not present (but estimated) in the control model. For orientation and longitudinal position EMC the disturbances are minimal and differences are only in total robot mass and inertia parameters (which are near enough to reality).
The ability of EMC to reduce plant disturbances is effectively verified for all control models, especially for DC motors: the presence of disturbance rejection allows to reduce significantly the tracking errors, making the wheel robot speeds follow a desired trajectory quite precisely. This can be seen for example from Figure 17.1 and Figure 17.2, which are the results for right DC motors (for left motor they are similar) taken from RHIT tests in Section 8.3.3 and Section 8.3.5, where EMC is used to control both left and right GoPiGo3 DC motors, with asynchronous control sampling time. Objective for estimated wheel output speed $\hat{y}_m$ (blue) is to follow the reference y_{ref} (red) and reduce the error with respect to measurement y (green): if no disturbance rejection is present is well evident that the red trajectory is not followed precisely anymore (Figure 17.2), instead this is successfully done adding disturbance rejection (Figure 17.1).

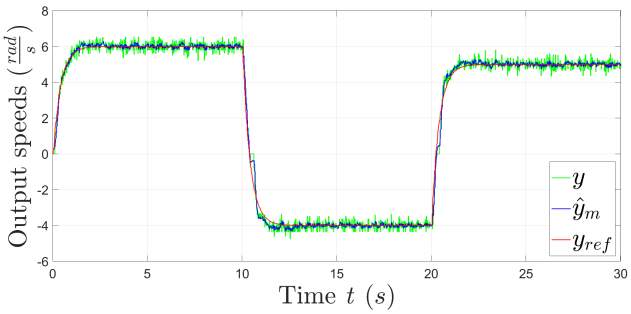


Figure 17.1: Combined DC motors EMC RHIT output speeds reference y_{ref}, estimated $\hat{y}_m$ and measured y - Right motor ***with*** disturbance rejection

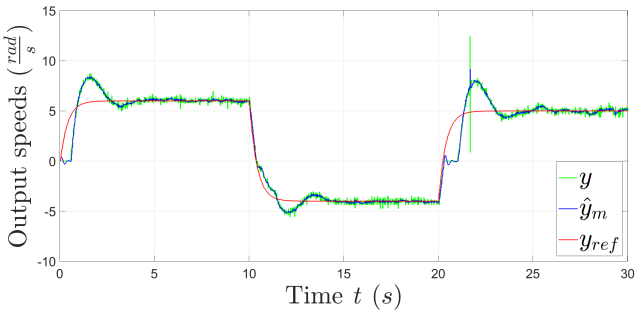


Figure 17.2: Combined DC motors EMC output speeds reference y_{ref}, estimated $\hat{y}_m$ and measured y - Right motor ***without*** disturbance rejection

Hierarchical structure requiring the combination of DC motors and orientation/position control models in a non-conventional way is successfully verified, since robot is able to finally reach a desired position in space following almost precisely the desired trajectory. This is a huge result because a robot can be controlled using very simple linear models, avoiding complex conversions and non-linear terms.
The next figures show the best result obtained moving GoPiGo3 robot with longitudinal trajectory till $\tilde{\xi} = 2\,\mathrm{m}$ position, and assuming a final orientation angle $\tilde{\theta}_z = 50°$ $(\approx 0.87\,\mathrm{rad})$, using EMC technique to control DC motors, robot position and orientation with hierarchical structure: the estimated outputs $\hat{y}_m$ (blue) successfully track the references y_{ref} (red), low tracking errors, and they are near to measurements y (green), low model errors, using asynchronous sampling time (plots taken from Section 16.3.3) (the oscillations at the end are due to unavoidable motor dynamics, they have been left in plot results to be sure that the target is effectively reached):

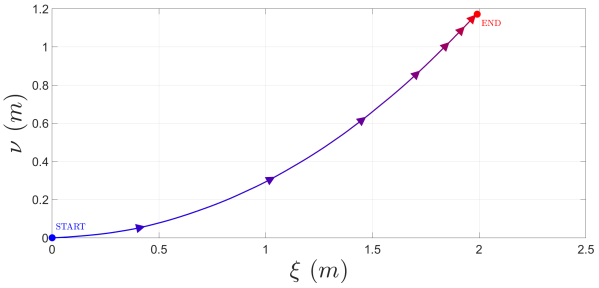


Figure 17.3: Robot position

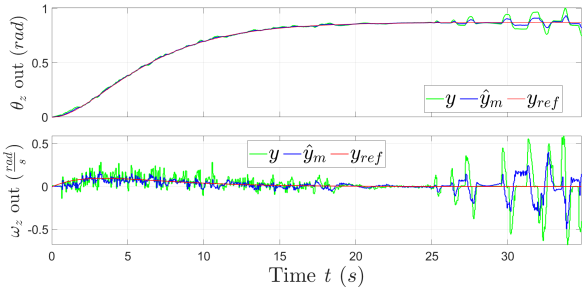


Figure 17.4: Orientation EMC y (green), $\hat{y}_m$ (blue) and y_{ref} (red)

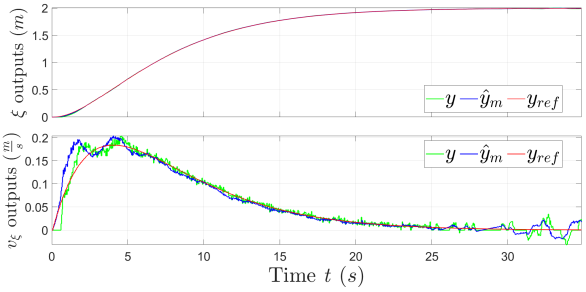


Figure 17.5: Longitudinal position and speed EMC y, $\hat{y}_m$ and y_{ref}

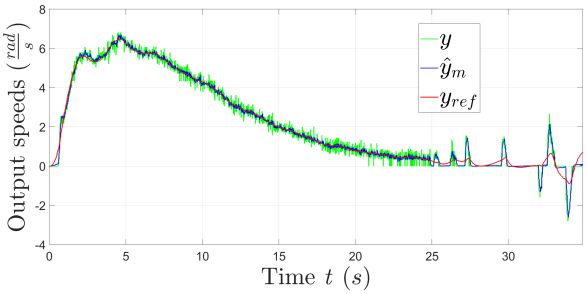


Figure 17.6: Right motor EMC y, $\hat{y}_m$ and y_{ref}

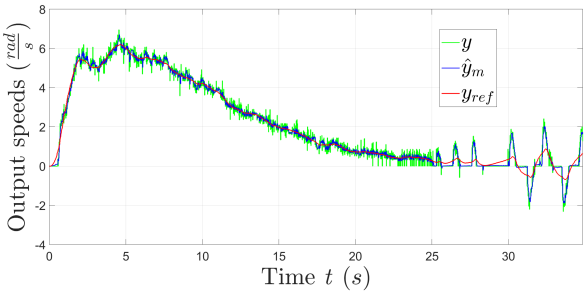


Figure 17.7: Left motor EMC y, $\hat{y}_m$ and y_{ref}

Future work may be concentrated in:

- Make control behaviour even better. For example the motors don't work if the voltage is below 2 V approximately, and this affects the first part of every test: the reference trajectory is rising but measured and estimated speeds remain zero for some time, and when they start a disturbance peak can be present. EMC is able to mitigate the most of this peak, but sometimes not at all. A solution can be starting the reference trajectory only when also the measurement speed starts.

- Build and test a more complex hierarchical structure to make robot able to move in a Cartesian ξ, ν 2D space, instead of following robot longitudinal position ξ and orientation yaw angle θ_z as already done.

- Send the input targets for EMC remotely (Networked controlled system NCS) connecting robot Raspberry board with another device (e.g. laptop with Windows/Linux OS). The last device run all EMC and send the commands inputs and targets to robot Raspberry. Conversely Raspberry is used only to run the motors on the basis of command inputs, recover encoder and IMU measurements and send them back to the other device. A NCS is mandatory when reference dynamics becomes more complex (e.g requiring obstacle avoidance and optimal path), because in this case Raspberry board will not be powerful enough to run the entire EMC.
  For NCS communication protocol to be used, an example is the MQTT, which allows to easily send the lightweight data needed for GoPiGo3 control.

- Other physical plants in different environments can be used to test asynchronous EMC, for example drones.

Bibliography

[1] Enrico Canuto - *Embedded Model Control: Outline of the theory.* ISA Transactions 46, pagg. 363–377, 2007 pag 2, pag 7, pag 8, pag 10, pag 11, pag 12, pag 52, pag 55, pag 132, pag 134

[2] Enrico Canuto - *On dynamic uncertainty estimators.* American Control Conference, Chicago (IL, USA), 2015 pag 10, pag 52, pag 55, pag 134

[3] Enrico Canuto, Fabio Musso - *Embedded model control: Application to web winding.* ISA Transactions 46, 379–390, 2007 pag 2, pag 132, pag 134

[4] Carlos Norberto Perez Montenegro, Luigi Colangelo, José María Pardo Álvarez , Alessandro Rizzo, Carlo Novara - *Asynchronous Multi-rate Sampled-data Control: an Embedded Model Control Perspective.* IEEE 58th Conference on Decision and Control (CDC), 2019 pag 2

[5] José María Pardo Álvarez, Carlo Novara, Carlos Norberto Perez Montenegro - *Embedded Model Control for Networked Control Systems.* Thesis, Politecnico di Torino, 2019 pag 2

[6] Wilber Acuña-Bravo, Andrés Molano-Jiménez, Enrico Canuto - *Embedded model control, performance limits - A case study.* DYNA 84(201), pagg. 267-277, 2017 pag 2, pag 12, pag 51

[7] Rached Dhaouadi, Ahmad Abu Hatab - *Dynamic Modelling of Differential-Drive Mobile Robots using Lagrange and Newton-Euler Methodologies: A Unified Framework.* Advances in Robotics & Automation, 2013. pag 23

[8] California Institute of Technology (Caltech) - *Handouts of ME72 Engineering Design Laboratory*, 2010-2011. Site: https://robotics.caltech.edu/~me72/class/ pag 22

[9] Riccardo Antonello, Angelo Cenedese - *Laboratory activity 1: DC gearmotor modelling.* University of Padova, 2016. Site: https://elearning.dei.unipd.it/mod/resource/view.php?id=26813 pag 39

[10] Kionix website (https://www.kionix.com) - Application notes *AN 012 - Accelerometer Errors*, 2015 pag 117

[11] Mazzoldi P., Nigro M., Voci C. - *Elementi di Fisica - Meccanica, Termodinamica (Seconda Edizione).* EdiSES, 2014 (8^{th} reprint) pag 18, pag 19, pag 20

[12] Ivan Virgala, Michal Kelemen - *Experimental Friction Identification of a DC Motor.* International Journal of Mechanics and Applications, 2013 pag 49

[13] M. Violante - *Operating Systems for Embedded Systems.* Polytechnic of Turin MSc course, 2018/2019 pag 2, pag 27

[14] M. Canale - *Digital control technologies and architectures.* Polytechnic of Turin MSc course, 2017/2018 pag 8

[15] Joao P. Hespanha, Payam Naghshtabrizi, Yonggang XuA - *Survey of Recent Results in Networked Control Systems.* Proceedings of the IEEE, 2007 pag 2

[16] Bernard Mulgrew, Peter Grant, John Thompson - *Digital Signal Processing, Concepts and Applications.* First publication MACMILLAN PRESS LTD, 1999 pag 68

[17] Dexter industries GoPiGo3 DD robot website - https://www.dexterindustries.com/store/gopigo3-base-kit/ pag 3, pag 15, pag 203

[18] Raspberry Pi 3 Model B website - https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ pag 17

[19] Adafruit 10-DOF IMU Breakout IMU website - https://learn.adafruit.com/adafruit-10-dof-imu-breakout-lsm303-l3gd20-bmp180 pag 17

[20] PuTTY client website - https://www.chiark.greenend.org.uk/~sgtatham/putty/ pag 26

[21] VNC Viewer and Server website - https://www.realvnc.com/en/connect/ pag 26

[22] Signals in UNIX platforms - https://en.wikipedia.org/wiki/Signal_(IPC) pag 28

[23] GitHub repository for GoPiGo3 robot material - https://github.com/DexterInd/GoPiGo3 pag 31

[24] GitHub repository for RTIMULib2 library, used for IMU measurements - https://github.com/RPi-Distro/RTIMULib or https://github.com/RTIMULib/RTIMULib2 pag 31, pag 117

[25] MathWorks Simscape toolbox - https://uk.mathworks.com/products/simscape.html pag 57

[26] Moving average filter - https://en.wikipedia.org/wiki/Moving_average pag 68

Nomenclature

Acronyms

CT	Continuous Time
DT	Discrete Time
CoM	Center of Mass
EMC	Embedded Model Control
EM	Embedded Model
HW	Hardware
SW	Software
SS	State space (equations)
LTI	Linear time invariant (system)
IMU	Inertial measurement unit
HIL	Hardware-in-the-Loop (test)
SIL	Software-in-the-Loop (test)
MIL	Model-in-the-Loop (test)
RHIT	Robot Hardware Implementation test: implementation of control models in a real robot
DC	Direct current (motor)
OS	Operating system
PWM	Pulse-width modulation
GPIO	General purpose input-output (pin)
I/O	Input/output
RF	Reference frame
BF	Body (reference) frame
IF	Inertial (reference) frame
IDE	Integrated development environment
wrt	With respect to
GUI	Graphical User Interface
ODE	Ordinary differential equation
DD	Differential-drive (robot)
MSD	Mass-spring-damper (physical system)
NCS	Networked Control System
MQTT	Message Queue Telemetry Transport (communication protocol)
MA	Moving Average (FIR filter)
FB	Feedback

GoPiGo3 DD robot main parameters

M_T	Total mass
I_T or I_{zz}^t	Total inertia z-axis
F_T, F_R, F_L	Total, left, right forces acting on axle and wheels
τ_T, τ_R, τ_L	Total, left, right torques acting on axle and wheels
ρ	Robot wheel mean radius
W	Half robot width
l	Robot longitudinal length

EMC general notations

x	All states
x_c	Controllable states

x_d	Disturbance states
$\hat{x}$	Estimated states
$\overline{x}$	Reference states
u	Control/command input (Control law)
$\overline{u}$	Reference input (component of u)
u_d	Disturbance rejection input (component of u)
u_{trk}	Tracking error input (component of u)
y	Output measurement/s
$\overline{y}, y_{ref}$	Reference output (Reference dynamics)
$\hat{y}_m$	Estimated output (EM part)
$\overline{w}, w$	Noises and disturbances (EM part)
T, T_s	Discrete sampling time
e_m	Model error
$\overline{e}, e_{trk}$	Tracking error
$\lambda_{_K}, \lambda_{_N}, \lambda_{_R}$	DT eigenvalues (control, noise est., ref.)
$\mu_{_K}, \mu_{_N}, \mu_{_R}$	CT eigenvalues (control, noise est., ref.)

DC motor parameters and EMC

$\omega_{_R}, \omega_{_L}$	Measured wheel angular speeds, left and right
$\phi_{_R}, \phi_{_L}$	Measured wheel angles, left and right
$\hat{\omega}_{_R}, \hat{\omega}_{_L}$	Estimated wheel angular speeds, left and right
$\tilde{\omega}_{_R}, \tilde{\omega}_{_L}$	Reference targets wheel angular speeds, left and right
R_a	DC Motor armature resistance
L_a	DC Motor armature inductance
V_a, V	DC Motor armature voltage
k_v	DC Motor back electromotive force constant
k_t	DC Motor torque constant
J_{eq}	DC Motor total equivalent inertia (motor side)
β_{eq}	DC Motor total equivalent friction (motor side)
τ_m	DC Motor mechanical time constant
τ_a	DC Motor electrical time constant
N	DC Motor gearbox reduction

Orientation EMC

θ_z	Robot yaw angle measurement, z-axis
$\hat{\theta}_z$	Robot yaw angle estimated with EMC
$\tilde{\theta}_z$	Robot yaw angle reference target
ω_z	Robot angular speed measurement, z-axis
$\hat{\omega}_z$	Robot angular speed estimated with EMC
$\tilde{\omega}_z$	Robot angular speed reference target

Longitudinal position EMC

ξ	Robot long. position measurement, ξ-axis
ν	Robot lateral position measurement, ν-axis
v_ξ	Robot long. speed measurement
$\hat{v}_\xi$	Robot long. speed estimation
$\hat{\xi}$	Robot long. position, EMC estimation
$\hat{v}_\xi$	Robot long. speed, EMC estimation
$\tilde{\xi}$	Robot long. position target
$\tilde{v}_\xi$	Robot long. speed target

This MSc thesis is written using LaTeX markup language, *book* document class, *baskervald* font, *11 pt* size