POLITECNICO DI TORINO
MSc Degree in Mechatronic Engineering

Final work

Considerations about the development of a Ballbot system

Tutors:

Prof. Stefano Mauro
Prof. Stefano Paolo Pastorelli

Candidate:

Roberta Iemolo

Academic year 2018/2019

Abstract

In this document are developed considerations about the design and control of a particular system located in the field of mobile robotics called Ballbot. A Ballbot has a particular structure that allows to present a new concept of mobility, it is a dynamically stable mobile robot that must balance itself on a sphere that guarantees the omnidirectionality movement. In particular, it was analyzed a Ballbot with a ball drive-mechanism composed of three omnidirectional wheels. Firstly, a 2D modeling of the system, considered as an inverted pendulum, was studied on the two decoupled XZ, YX planes. Congruently, to achieve the system stability, a control system was developed through the use of *Matlab* and its Simulink tool. In particular was employed an optimal control theory implemented by an LQR (Linear Quadratic Regulator). Using *Simscape*, a Matlab tool that allows the modeling and the simulation of a multidomain physical system, a simplification of the physical model of the thought ballbot has been built. Various tests on balancing of the simplified Ballbot prototype were performed by varying the initial state-target of the ballbot, giving an impulsive stimuli and evaluating the balancing capacity of the system. At a later time, the attention was dedicated on the Electronics that allows the control of the sphere. An Arduino-MKR1000 has been chosen as microcontroller, therefore it follows a description of the affiliated sensors necessary for the ballbot system. As a consequence, using appropriate support packages and libraries, it was conceived a Simulink-Arduino interface on Simulink tool.

Contents

List of Figures

List of Tables

1 Introduction

One of the main fields of mobile robotics application is the human environment. In recent years, in fact, the mobile robotics sector has had a considerable expansion due above all to the invention of an increasingly advanced and precise sensors, such as to develop their ability to assist the human in carrying out various and, more or less, complicated physical tasks. The current mobile robotics is increasingly enriched by prototypes with the ability to move in limited spaces, to struggle between obstacles and, indeed, also to interact with the human. In this context it is however important to pay attention on the structure of the robots. A structure that is congruent to the work environment is in fact necessary and, in the case of mobile robots that have to interface with a human environment, it is certainly convenient that a structure is closer to the human structure, therefore a tall and thin robot. On the other hand, a conventional mobile robot that does not care about having a particular physical structure and characterized by a large, heavy base would make it unfit for movement in a crowded environment. A first attempt at a mobile robot that respects the attributes listed above is the *Segway*, a two-wheeled mobile robot with a structure similar to an inverted pendulum. The operating principle of the *Segway* is based on center of mass (CM) position control of the system. To move forward or backward, the rider must bend slightly forward or backward according to the desired movement; a manual mechanism is used to control the bar in order to allow the vehicle to steer. However, Two-wheeld mobile robot does not result an ideal solution because its restricted motion has still unsolved.
More adequate to the aim is the Ballbot system with a new concept of mobility, its structure is also similar to that of an inverse pendulum but it passes from the balance on two points of support on the ground to only one.

1.1 Ballbot concept

A Ballbot is a new and recent type of mobile robot. Its points of strength, if compared to other robots, are the omnidirectional motion and the rotation about its vertical axis. It is a mobile robot that maintains balance on a ball and has a single contact point on the ground and its main purpose is to maintain meta-stable equilibrium and offer the possibility of movement based on user input. As much as it is an unstable system, even when motionless, it belongs to the area of research of the nonholonomic system. These are characterised by nonintegrable rate constraints arising out of rolling contact or momentum conservation. So, one of the main issues of the Ballbot is about the control. Modern Ballbot robots use fuzzy systems, genetic algorithms and other artificial intelligent tools, but more extensively used are the PID (Proportional-Integral-Derivative) and LQR (Linear Quadratic Regulation) control laws. In order

to control the stabilization of the robot, there are different possible answers. The classical implementation, consists of four motors, characterized by the generator force perpendicular on the axis of each one. In this structure opposite motors would receive equal value commands of opposite signs. Three-motors models have a more complex computational aspect. The command force will be located at different angles from the gyroscope axis. Moreover, it has more degrees of freedom than a simple inverted pendulum as the sphere allows travel in any given direction.

2 State of the Art

Successfully tested Ballbot implementation are the CMU Ballbot, BallIP Ballbot, REZERO Ballbot, that uses a three omni-directional wheels available to control the sphere's movement while keeps the robot in the meta-stable position, but also the NXT LEGO Ballbot, which uses also it three common wheels and represent the simplest ballbot model, with motors perpendicular to the OX and the OY plane and the AIT Ballbot developed by the Asia institute of Technology that, unlike the previous two mentioned above, uses four omni-directional wheels which make it certainly easier to model than to compute resources.[1]

2.1 CMU Ballbot

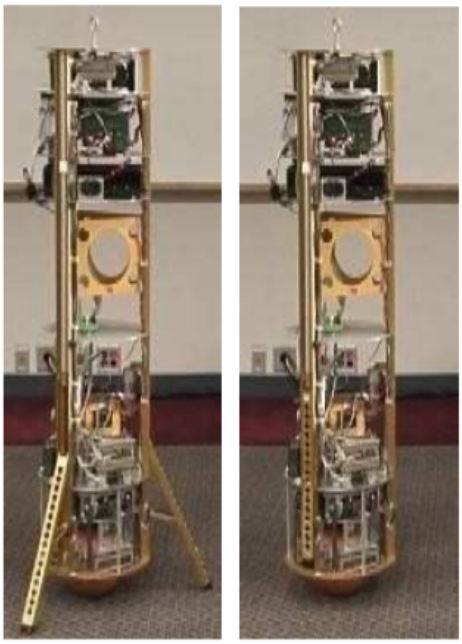

Figure 1: CMU Ballbot: (a) statically stable, (b) dynamically stable

CMU Ballbot is the pioneer robot of this type of system. It is a person-sized mobile robot developed at Carnegie Mellon University, USA, in 2005. Its structure is part of a construction with one body and one sphere with a stabilization methods similar of a simplified planar inverted pendulum[2]. It represents an evolved omni-directional mobile robot able to balance both dynamically and statically on a single spherical wheel. The design of the considered Ballbot is characterized by four principle drive

system:
'control system', 'leg drive' that allows controlled transition from SSS (Static Stable State) to DSS (Dynamically State Space), 'yaw drive' to allow rotation around its vertical axis(yaw-motion) and 'four-motor inverse mouse-ball' drive that enables backward and forward motion.

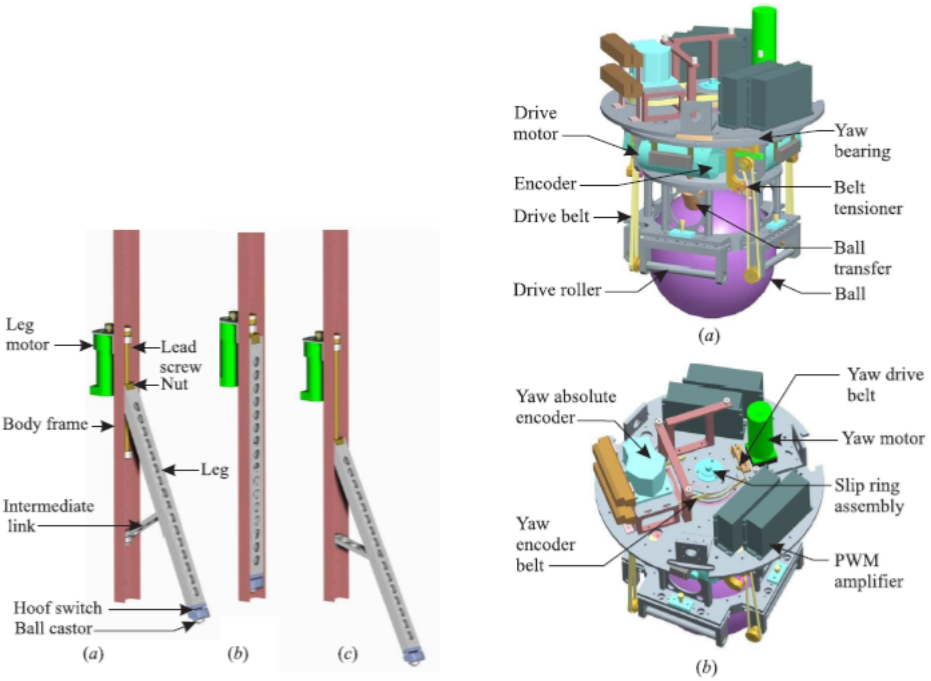

Figure 2: (a) View showing main drive arrangement, (b) View showing yaw drive mechanism

Figure 3: (a) View showing main drive arrangement, (b) View showing yaw drive mechanism

To guarantee a statically stable state when powered off a triad system is used.

2.1.1 CMU Ballbot Control

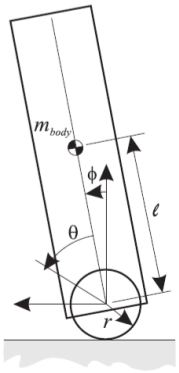

Figure 4: Planar simplified ballbot model.

- Balancing Controller:
  the 3D CMU Ballbot balancing is reached using two independent Proportional Integral Derivative (PID) controllers acting one in each of the vertical planes. PID gains were tuned manually. These two controllers try to move the ball directly below the center mass of the body and make Balancing controller good at balancing about zero body angles.

- Station Keeping Controller:
  Station Keeping allows balancing at a desired angle position Φ_d even when disturbed. It is realized adding an outer loop to Balancing Controller and is regulated by a Proportional-Derivative (PD) controller as shown in the following block diagram:

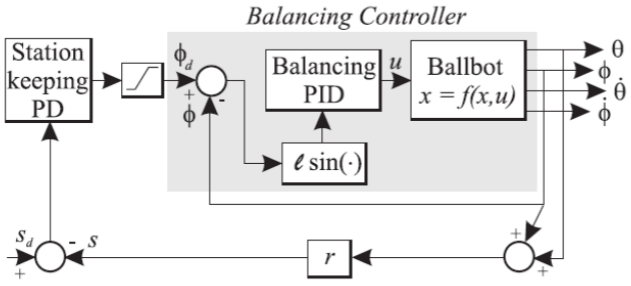

Figure 5: Station Keeping block diagram with Balancing control block.

- Yaw Controller:
  It is an independent controller composed by an inner Proportional Integral (PI) control loop whose task is feed back the yaw angular velocity, and an PD controller that create an outer loop working with yaw angle Ψ and yaw angular velocity $\dot{\Psi}$:

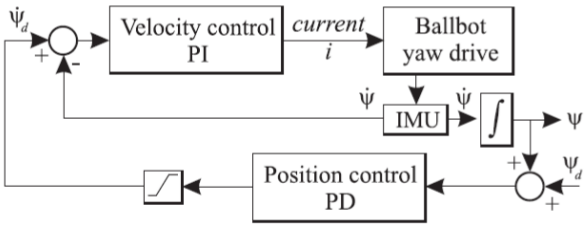

Figure 6: Yaw controller block diagram

- Legs Controller:
  Leg controller is composed by three decoupled controller, one for each leg, to allow the rising up and the putting down of the legs. An inner Proportional-Integrative (PI) control loop guarantees the legs-down controller instead, legs-up controller is created through a PI speed controller that stops the legs speed is lower than a set low threshold so, when the legs collide the body.

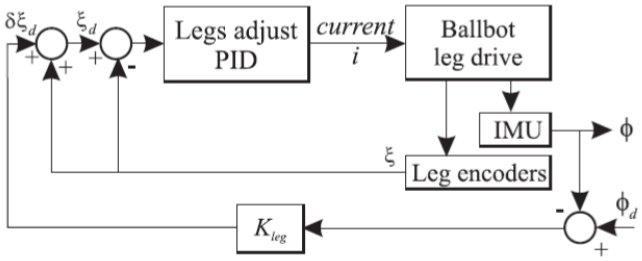

Figure 7: Legs-adjust controller block diagram

2.2 BallIP-TGU Ballbot

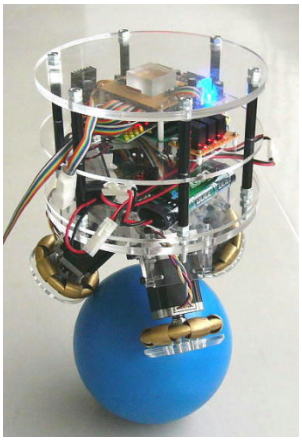

Figure 8: BallIP-TGU

BallIP-TGU Ballbot is developed at Tohoku Gakuin University, Japan, in 2008. This project is expanded to test the capability of robots to carry loads in order to promote these modern systems also for cooperative transportation. Some load transport examples are shown below:

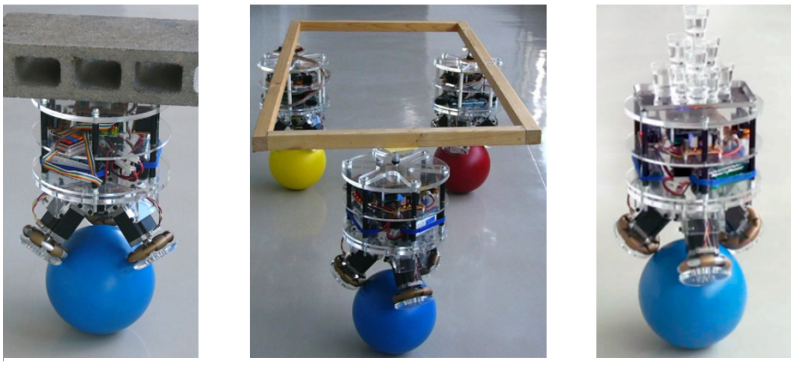

Figure 9: BallIP-TG load transport examples

Actuator mechanism of BallIP is provided by three stepper motors, placed at an angle of 120 degrees one each other, whose take off the staff needed for shaft encoders and make the system more accurate.
Wheels and motor's shaft are directly connected. Otherwise, there is not belt in the mechanism transmission and, for this reason, in BallIP-TGU Ballbot there is much less backlash with respect to CMU Ballbot [3].

2.2.1 BallIP-TGU Controller

The controller to balance the BallIP-TPU is developed starting from a simply Proportional-Derivative(PD) controller.
In constrast to the CMU Ballbot that implement a torque control, BallIP uses a velocity control strategy on their omniwheels for two principle reason: first one is due to the use of stepper motors to actuate the omnidirectional wheels, and the other one is because this type of control better handles the scenario where omniwheel and the ball loses the contact [4].

2.3 Rezero Ballbot

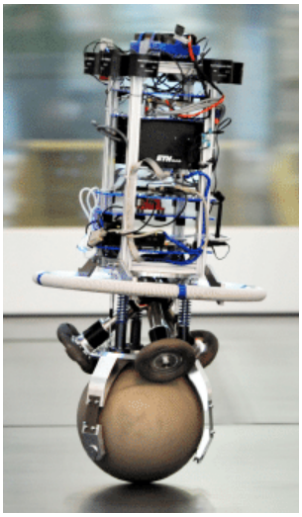

Figure 10: Rezero Ballbot

Rezero Ballbot developed by a team of 10 students at ETH Zurich, Switzerland, in 2010. The project was born as a challenge: create a new product or prototype in not over one year.
The development of Rezero is mainly inspired by the two previous CMU and BallIP-TCU ballbot projects. Unlike the previous two, it is designed also for high acceleration, fast movement, and with a particular attention on external structure and design in order to make the most of all possible abilities of a Ballbot system.
One of successes envisaged and obtained by the Rezero is the capability to move spontaneously, smoothly, in every direction.
Furthermore it demonstrates its potential, even in a restricted way, being able to interact with a bounded group of people: it can even recover from the thrusts, reacting on attraction and being able to follow a path, or a person at a constant distance.

2.3.1 Rezero Technics

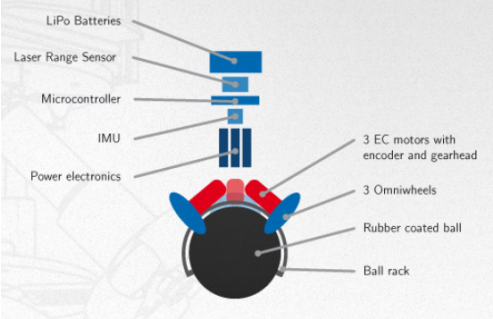

Figure 11: Rezero Technics

The project is thinking to create more propulsion possible with the lowest weight possible. Rezero's drive system, as BallIP-TGU Ballbot, is composed by three omni-directional wheels able to rotational motion without restriction in the axial direction. Omniwheels that allow the control of the sphere without paying attention to unwanted friction.
Rezero Electronic system is composed by high-tech sensors: 'Distance and Sonic' sensors, to capture the presence of an obstacle, with microphones to make Rezero able to perceive and to react to its environment, gyroscope and accelerometer integrated on the IMU (Inertial Measurements Unit) shield, to acquire angular position: Rezero is programmed to measure the incline 160 times per second.

2.3.2 Rezero Control

In order to develop a controller robustness against the noise , the ETH team use a linear control with a low pass filters joined to a Kalman controllers [3].
Kalman Filter Control - Concept:
Sensors give information about the system state but with uncertainty or inaccuracy, and not directly. Kalman Filter goal is to squeeze as much possible information from the measurements to obtain a better answer with respect to an estimate could be give. Looking at the current state (at time (N-1)),with particular and adequate assumptions, Kalman Filter predict the next state at time N. One of the advantage is that it is light on memory, in fact it do not need to hold any history about the previous state, and it very fast, making it well suited for real time problems and embedded systems. It also called Linear Quadratic Estimation (LQE).

2.3.3 Rezero Design

The design of Rezero is thinking to communicate ballbot agility.
Its envelop is able to vary in height, it can be let down over the ball during park scenario and it is able to remain stable.

Figure 12: Rezero with design casing

Table 1: Comparison of the characteristics of four Ballbot robots

University	CMU	BallIP-TGU	ETH
Year	2006	2008	2010
Weight (Kg)	45	11	14.5
Height (cm)	150	50	80
Max deviation angle (deg)	Less than 1	Less than 5	Less than 1
Rotation	yes	yes	yes
Controller	LQR/PID	PD	LQR/gain scheduling

2.4 NXT LEGO Ballbot

Figure 13: NXT Lego Ballbot

NXT LEGO Ballbot is a particular ballbot system developed at University of Adelaide, Australia, using LEGO pieces and blocks.
It has a better stability compared to CMU Ballbot and it is characterized by the possibility to be controlled via remote control [3]. The project is tested using a plastic ball as spherical wheel, its motion is regulated through three wheels connected to the LEGO parts attached diagonally. The wheels are covered with two rubber tires attached to DC motors and one freely-spinning wheel. NXT Ballbot movement is allowed by rotating DC motors [5].

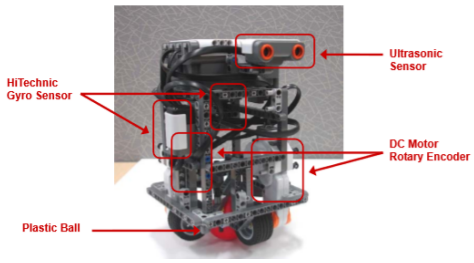

Figure 14: NXT-Lego Ballbot Sensoring

Figure 15: NXT-Lego Ballbot remote control

3 Model Based Design (MBD) - Overview

Intelligent systems are at the base of the technology used in everyday life. These systems use the built-in software to provide intelligent communications, transport and automation: "Today there are more intelligent systems on our planet than human beings".
As these systems add new features, the software they contain grows and incorporates increasingly powerful algorithms..To develop such a complex code, it is increasingly necessary to use system models throughout the design process. *MathWorks*, company manufacturer of the MATLAB software, has perfected this "model-based approach" by creating software tools to achieve faster design, better quality, greater flexibility and reduced costs. Model-Based Design starts from a system model in which ideas and requirements are collected. The model is a specific executable that allows you to collaborate on different engineering roles and modeling domains: it provides graphical modeling environments which contains block diagrams and state machines. You can simulate the model or perform rapid prototyping on it to explore design alternatives in a short time. At each iteration it is possible to refine the model of the system to optimize the project, create test benches for system verification and saving time. When it is ready for implementation it is possible to automatically generate the code (*code-generation button*, for the working of both microelectronic (Arduino, Raspberry PI) and embedded devices, FPGA (Field Programmable Gate Array), thus eliminating errors related to manual coding. So, Model-Based offers the opportunity to simplify projects by avoiding problems of design, automation or different phases in advance, as the generation of code and the acceleration of the entire development process that actuate a significantly reduction of physical prototypes and experimental tests thanks to the use of the simulation.

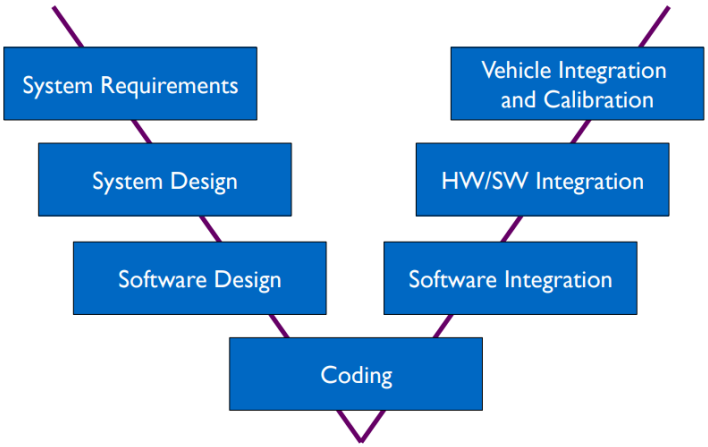

Figure 16: Model Based Design V-flow

- MODEL IN THE LOOP (MIL):
  It is a purely functional simulation. The plant and the controller running both on your PC to be sure that the model works properly. In fact, in this development process it is possible simulate and modify the design until you are satisfy. The model exists entirely in the simulation tool: *Simulink* in this project.

- SOFTWARE IN THE LOOP (SIL):
  Durign the SIL process the plant exists on the tool, while the controller is translate into C-Code. So, code is plant depender but the simulation is still on PC: this is a simulation in the simulation domain. Time is not real and logic signals are exchanged between controller and plant. In order to avoid high time of running, optimization is necessary.

- PROCESSOR IN THE LOOP (PIL):
  PIL goal is to validate the control on the target hardware. In this development process there is still the *Simulink model* of the plant. So, part on the model running on PC (plant) and the other part, that is a C-code, runs on the target-hardware or rapid prototyping hardware.

- HARDWARE IN THE LOOP (HIL):
  This phase is simulated in real time (physical domain): physical signals are exchanged between controller and plant. The firts one runs on the target-HW and the plant on the emulation hardware. For these reasons, HIL is good for testing interactions with hardware and real-time performance.

In this document is developed the first part of V-model on a preliminary ballbot design.

4 Preliminary design

4.1 2D Geometric parameters

Table 2: Geometric parameters of the 2D preliminary design

Parameter	Symbol	Value
Distance between CM and ball center	L	0.6 [m]
Ball radius	R_s	0.16 [m]
Ball mass	M_s	2.5 [Kg]
Ball inertia	J_s	0.016 [Kg m^2]
Inertia Pitch moment about CM	J_α	12.48 [Kg m^2]
Inertia Roll moment about CM	J_β	12.48 [Kg m^2]

4.2 Drive mechanism

Starting from an analysis about the different types of arrangements which allows spherical omnidirectional motion of the functional prototypes ballbot described in the previous sections, it was conclued that 'three omniwheels over ball center' results the best drive mechanism considering a trade-off among construction, controllabilty, encumbrance and friction .
In particular, Ballbot drive mechanism proposed in this section is based on the work carried out by engineers at Faculty of Automatic Control and Computers University Potitechnica of Bucharest, about precisely Three Omni-wheel Ballbot Optimum Implementation; In Figure17 it is shown control functionality block diagram developed:

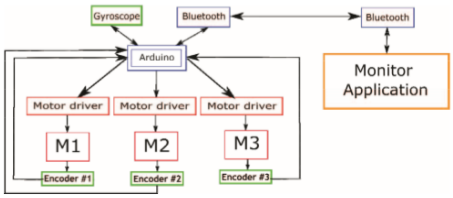

Figure 17: Three-ommniwheel Drive mechanism block diagram

A Bluetooth module communicate the robot's operating parameters to a computer in order to monitor the behavior of the robot. The Arduino microcontroller reads orientation data detected by the gyroscope in order to compute adequate commands must be executed by each of the three motors to maintain the robot stability. After the motors performed the microcontroller commands, the econders send motor monitoring data to the computer application through Bluetooh module to restart robot's behavior

control. In this project the microcontroller conceived is an Arduino MKR1000 so:

1. Its Wi-fi integrated module is considered, instead of a Bluetooth module, to communicate the robot's operating status;
2. The microcontroller reads the detected data from a Gyroscope sensor through BNO0555 IMU (Inertia Measurement Unit).

The implementation of the "Three Omni-wheel Ballbot implementation" is shown in the *Figure17*, the three motors are arranged in an equilateral triangle manner with motor3(M3) placed on OY axis and the other two motors disposed at 120 degrees one to the left side of it and the other one to the right side [1].

5 2D Model Design

In this chapter is described the mathematical model of a Ballbot robot following the modeling of NXT Ballbot: a Mindstorms NXT version of Ballbot[5]. The first Ballbot modeling approach do not consider the omniwheels, the Ballbot model is a spherical wheeled inverted pendulum so the structure looks like a multy-body system in which involve two rigid bodies: the sphere and the body.[cit NXT LEGO]

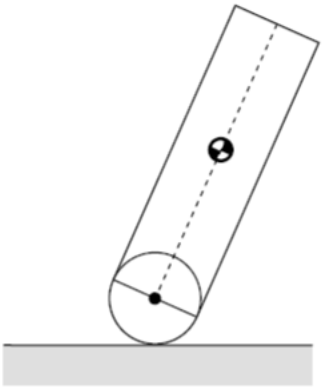

Figure 18: Ballbot modeling: Inverted spherical pendulum

5.1 Planar system Model

According to this work [5] it is needed to actuate different assumptions and particular simplifications.
3D modeling of an omnidirectional spherical robot interests research area of the non-holonomic system so, to simplify the study of the motion equations of the system, it is analyzed the motion on XZ plane an YZ plane: the two planes are considered decoupled and are characterized of the same motion equations. These assumptions reduce the three dimensional problem in the study of two separate and identical spherical wheeled inverted pendulum.

5.1.1 Assumptions and Simplifications:

1. The motion in the Pitch-XZ plane and in the Roll-YZ plane are decoupled;
2. Since the device present a revolution symmetry, the dynamic equations for the two planes are identical.

Simplifying hypotheses:

1. No slip between sphere and the ground;
2. As a first approach rolling friction is considered negligible.

5.1.2 XZ Planar System:

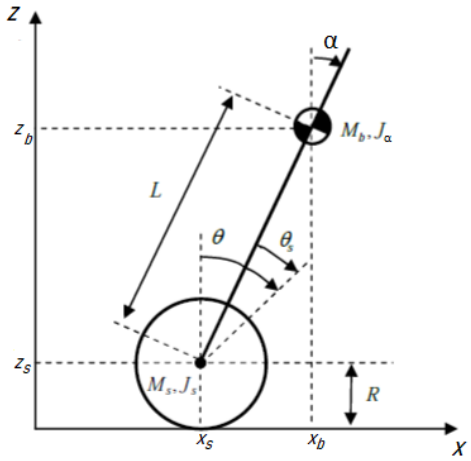

Figure 19: XZ Planar System

α : *body pitch angle*; θ : *spherical wheel angle*; θ_s : *spherical wheel angle driven by the DC motor*;

Ballbot generalized coordinates and generalized velocities on XZ plane based on the coordinate system of the Figure, assuming that at time t=0 $\rightarrow$ θ=0.

$(x_s, z_s) = (R\theta, R)$;

$(\dot{x}_s, \dot{z}_s) = (R\dot{\theta}, 0)$;

$(x_b, z_b) = (x_s + Lsin(\alpha), z_s + Lcos(\alpha)) = (R\theta + Lsin(\alpha), R + Lcos(\alpha))$;

$(\dot{x}_b, \dot{z}_b) = (R\dot{\theta} + L\dot{\alpha}cos(\alpha), -L\dot{\alpha}sin(\alpha))$;

5.2 YZ Planar System

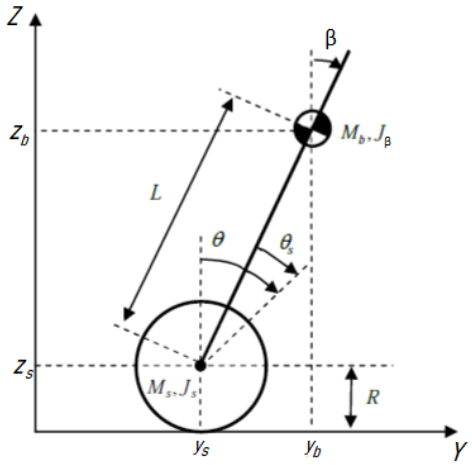

Figure 20: YZ Planar System

β : *body roll angle*; θ : *spherical wheel angle*; θ_s : *spherical wheel angle driven by the DC motor*;

Ballbot generalized coordinates and generalized velocities on YZ plane based on the coordinate system of the Figure, assuming that at time t=0 $\rightarrow$ θ=0. $(y_s, z_s) = (R\psi, R)$;

$(\dot{y}_s, \dot{z}_s) = (R\dot{\psi}, 0)$;

$(y_b, z_b) = (y_S + Lsin(\beta), z_S + Lcos(\beta)) = (R\psi + Lsin(\beta), R + Lcos(\beta))$;

$(\dot{y}_b, \dot{z}_b) = (R\dot{\psi} + L\dot{\beta}cos(\beta), -L\dot{\beta}sin(\beta))$;

5.3 Equations of Motion

Dynamic equations of motion of the 2D multy-body ballbot system can be derived through the Lagrange method.
Defined the state variables of the system, considering as states the set of generalized angular positions q(t) and the generalized angular velocities $\dot{q}(t)$, the state vector is 2n-dimensional vector (with n= number of generalized coordinates defined as:

$$\left[\theta \quad \alpha, \quad \dot{\theta}, \quad \dot{\alpha}\right] : \; XZ \; plane \; state \; vector; \tag{1}$$

$$\begin{bmatrix}\theta & \beta, & \dot{\theta}, & \dot{\beta}\end{bmatrix} : \ YZ \ plane \ state \ vector; \tag{2}$$

Having hired only holonomic constraints, Lagrange approach generates n-differential equations:

$$\frac{d}{dt}(\frac{\delta L}{\delta \dot{q}_i}) - \frac{\delta L}{\delta q_i} = F_i; \qquad i = 1, \ldots, n \tag{3}$$

where:
L is the *Lagrange state function* defined as the difference between the kinetic co-energy C^* and its potential energy P :

$$L(q, \dot{q}) = C^*(q, \dot{q}) - P(q) \tag{4}$$

and the term $\frac{\delta L}{\delta \dot{q}_i}$ is indicated as *generalized momentum.*

5.3.1 Lagrangian Modeling- XZ Planar system

Considering the 2D Ballbot modeling on XZ plane (Figure.19), the goal of this section is to propose a relationship between the torque applied to the sphere and the response of α and θ variables, using the Lagrange approach. The same procedure is developed for YZ-plane on Appendix *YZ-Plane Math.* As said in subsection *5.1.1 Assumptions and Simplifications*, same results of XZ-plane are obtained.

$$\begin{cases} \frac{d}{dt}\frac{\delta L}{\delta \dot{\theta}} - \frac{\delta L}{\delta \theta} = F_\theta \\ \frac{d}{dt}\frac{\delta L}{\delta \dot{\alpha}} - \frac{\delta L}{\delta \alpha} = F_\alpha \end{cases} \quad \text{XZ-plane Lagrangian differential equations;} \tag{5}$$

$$L = T_1 + T_2 - U \quad \text{XZ-plane Lagrange state function} \tag{6}$$

where:

$T_1 = \frac{1}{2} M \left\|x\right\|^2$; Translational kinetic energy;

$T_2 = \frac{1}{2} I \left\|w\right\|^2$; Rotational kinetic energy;

$U = Mgh$; Potential energy;

$$T_1 = \frac{1}{2} M_s(\dot{x}_s^2 + \dot{z}_s^2) + \frac{1}{2} M_b(\dot{x}_b^2 + \dot{z}_b^2) \tag{7}$$

$$T_2 = \frac{1}{2} J_s \dot{\theta}^2 + \frac{1}{2} J_\alpha \dot{\alpha}^2; \tag{8}$$

$$U = M_s g z_s + M_b g z_b; \tag{9}$$

Substituting generalized coordinatees and velocities defined in *5.1.2 XZ Planar System* section and doing the math,Appendix *XZ Plane Math*, the following T_1, T_2, U, are computed:

$$T_1 = \frac{1}{2} M_s R_s^2 \dot{\theta}^2 + \frac{1}{2} M_B R_s^2 \dot{\theta}^2 + \frac{1}{2} M_B L^2 \dot{\alpha}^2 + M_B R_s L \dot{\theta} \dot{\alpha} cos(\alpha); \tag{10}$$

$$T_2 = \frac{1}{2} J_s \dot{\theta}^2 + \frac{1}{2} J_\alpha \dot{\alpha}^2; \tag{11}$$

$$U = M_s g R + M_b g (R + L cos(\alpha)); \tag{12}$$

So, from equation (4):

$$L = \frac{1}{2} R_s^2 \dot{\theta}^2 (M_s + M_b) + \frac{1}{2} M_b L^2 \dot{\alpha}^2 + M_b R_s L \dot{\theta} \dot{\alpha} cos(\alpha) + \frac{1}{2} J_s \dot{\theta}^2 + \frac{1}{2} J_\alpha \dot{\alpha}^2 - M_s g z_s - M_b g z_b; \tag{13}$$

Deriving congruently from (3) (as shown in the Appendix section) the following XZ-plane Lagrangian differential equations are obtained:

$$\begin{cases} R_s^2 (M_s + M_b) \ddot{\theta} + M_b R_s L cos(\alpha) \ddot{\alpha} + J s \ddot{\theta} - M_b R_s L sin(\alpha) \dot{\alpha}^2 = F_\theta \\ M_b R_s L cos(\alpha) \ddot{\theta} + M_b L^2 \ddot{\alpha} + J_\alpha \ddot{\alpha} - M_b R_s L \dot{\theta} \dot{\alpha} sin(\alpha) - M_b g L sin(\alpha) + M_b R_s L \dot{\theta} \dot{\alpha} sin(\alpha) = F_\alpha \end{cases} \tag{14}$$

5.3.2 2D Linearized System - State-Space formulation

In this subsection is proposed the study of the behavior of Lagrangian system in proximity of an equilibrium configuration. For this purpose it will apply to the equations of Lagrange a linearization procedure: the equilibrium point ($\alpha = 0$) is chosen as point of linearization, so:

- $\sin(\alpha) \rightarrow \alpha$;
- $\cos(\alpha) \rightarrow 1$;
- Second order term $\dot{\alpha}^2$ is neglected;

With the above considerations, Lagrangian equations (5) became:

$$\begin{cases} [(M_s + M_b)R_s^2 + J_s]\ddot{\theta} + [M_b R_s L]\ddot{\alpha} = F_\theta \\ [M_b L R_s]\ddot{\theta} + [M_b L^2 + J_\alpha]\ddot{\alpha} - M_b g L\alpha = F_\alpha \end{cases} \tag{15}$$

Once is obtained a linear formulation of the Lagrangian differential equations, it can be expressed as a second order differential vector equation [7]:

$$M\ddot{q}(t) + (D + G)\dot{q}(t) + (K + H)q(t) = F \tag{16}$$

Where:

- $M = M^T$: mass or inertia matrix, positive definite symmetric;
- $D = D^T$: viscous damping matrix, symmetric;
- $G = -G^T$: gyroscopic matrix, skew matrix;
- $K = K^T$: stiffness (elasticity) matrix, symmetric;
- $H = -H^T$: circulatory matrix (constrained damping), skew matrix;
- F is the generalized force vector.

Not all the terms described above are always present in the dynamical equations of a multibody system. In our case :

$$M\begin{bmatrix}\ddot{\theta}\\ \ddot{\alpha}\end{bmatrix} + (D + G)\begin{bmatrix}\dot{\theta}\\ \dot{\alpha}\end{bmatrix} + (K + H)\begin{bmatrix}\theta\\ \alpha\end{bmatrix} = \begin{bmatrix}F_\theta\\ F_\alpha\end{bmatrix} \tag{17}$$

and comparing linear modeling of the system (15) the following matrices are obtained:

$$M = \begin{bmatrix}(M_b + M_s)R_s^2 + J_s & M_b L R_s\\ M_b L R_s & M_b L^2 + J_\alpha\end{bmatrix} \tag{18}$$

$$(D + G) = \begin{bmatrix}0\end{bmatrix} \tag{19}$$

$$(K + H) = \begin{bmatrix}0 & 0\\ 0 & -M_B g L\end{bmatrix} \tag{20}$$

A system with this structure, with (D+G) term equal to 0 is called *circulatory system.* H matrix contains non conservative terms that can cause instability [6]. Considering the state variables defined in the state vector (1):

$$x = [\theta, \alpha, \dot{\theta}, \dot{\alpha}] \tag{21}$$

and the following state space description:

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) \end{cases} \tag{22}$$

- A $\in R^{\text{nxn}}$ is the State matrix;
- B $\in R^{\text{nxm}}$ is the Input matrix;
- C $\in R^{\text{pxn}}$ is the Output matrix;
- u(t) $\in R^m$ is the input vector, also called *"control signals"* or *"commands"*;
- y(t) $\in R^p$ is the output vector, it is composed by the signals generated by the user as control actions;

it is possible to write the state equations:

$$x = [\theta, \alpha, \dot{\theta}, \dot{\alpha}] \tag{23}$$

$$\dot{x}(t) = \begin{bmatrix} 0_{nxn} & I_{nxn} \\ -M^{-1}(K+H) & -M^{-1}(D+G) \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ M^{-1} \end{bmatrix} u(t) \tag{24}$$

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & A_{(3,2)} & 0 & 0 \\ 0 & A_{(4,2)} & 0 & 0 \end{bmatrix} \qquad \textit{State matrix;} \tag{25}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ \frac{M_{(2,2)}+M_{(1,2)}}{detM} \\ \frac{M_{(2,1)}+M_{(1,1)}}{detM} \end{bmatrix} \qquad \textit{Input matrix;} \tag{26}$$

with:

$$A(3,2) = \frac{M_B g L M_{(1,2)}}{detM} \tag{27}$$

$$A(4,2) = \frac{M_B g L M_{(1,1)}}{detM} \tag{28}$$

$$x = \begin{bmatrix} \theta \\ \alpha \\ \dot{\theta} \\ \dot{\alpha} \end{bmatrix} \qquad \textit{state vector;} \qquad u = [F_\theta] \qquad \textit{control signal;} \tag{29}$$

5.4 Modeling with non-zero floor slope

In this subsection is presented a 2D-ballbot modeling in an environment closer to the real human one: the perfomance analysis is executed considering a floor slope. Following the study [11] about a Ballbot on a non completely smooth surface and exploiting again a Lagrangian approach, the dynamic equations of the Ballbot behavior are obtained. Unlike the 2D modeling of *chapter5*, in this case it is needed to add a coordinate reference system $X'Z'$ with X' results coincident with the ground, θ and ψ are used as system degrees of freedom, as shown in figure below.

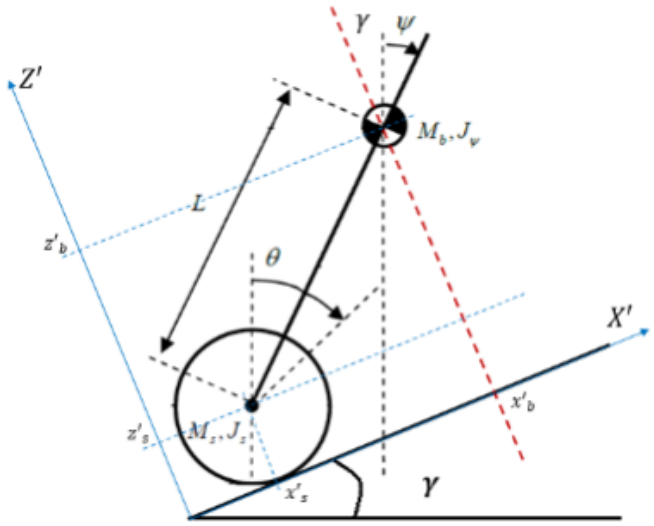

Figure 21: XZ-plane with floor slope - 2D model

Analyzing the system the total translational/rotational kinetic energy and potential energy of the system it is derived:

$$T_{trasl} = \frac{1}{2}M_s(R\dot{\theta})^2 + \frac{1}{2}M_b((R_s\dot{\theta} + L\dot{\psi}\cos(\gamma-\psi))^2 + (L\dot{\psi}\sin(\gamma-\psi))^2); \tag{30}$$

$$T_{rot} = \frac{1}{2}J_s\dot{\theta}^2 + \frac{1}{2}J_\psi\dot{\psi}^2 \tag{31}$$

$$U = M_s g z'_s + M_b g L\cos(\gamma-\psi) \tag{32}$$

Accordingly to relations (5) and (17) are obtained the following matrices:

$$M = \begin{bmatrix} M_{1,1} & M_{1,2} \\ M_{2,1} & M_{2,2} \end{bmatrix} \tag{33}$$

where:
$M_{1,1} = R_s(M_s + M_B) + I_B$

$M_{1,2} = R_s L M_B \cos(\gamma - \psi)$
$M_{2,1} = E_{1,1} + E_{1,2}$
$M_{2,2} = gR_s(M_s + M_B) + E_{1,2}$

$$(D+G) = \begin{bmatrix} (D+G)_{1,1} & (D+G)_{1,2} \\ (D+G)_{2,1} & (D+G)_{2,2} \end{bmatrix} \tag{34}$$

where:
$(D+G)_{1,1} = 0$
$(D+G)_{1,2} = -R_s L M_B \sin(\gamma - \psi)\dot{\psi}$
$(D+G)_{2,1} = 0$
$(D+G)_{2,2} = -R_s L M_B \sin(\gamma - \psi)\dot{\psi}$

$$(K+H) = \begin{bmatrix} (K+H)_{1,1} \\ (K+H)_{1,2} \end{bmatrix} \tag{35}$$

where:
$(K+H)_{1,1} = -gR_s(M_s + M_B)\sin\gamma$
$(K+H)_{1,2} = -gR_s(M_s + M_B)\sin\gamma - gLM_S sin\phi$
In this representation of the system, the equations of motion have been transformed in order to obtain the second equation unforced, so non-potential force vector $\vec{f_n}$ (17), in this case became $\vec{f_n} = [T \quad 0]^T$ Starting from the definition of the state-vector (36), the state-space representation of the ballbot system in a canonic form can be derived from the equations below:

$$x = \theta \quad \psi \quad \dot{\theta} \quad \dot{\psi} \tag{36}$$

$$\ddot{\theta} = \frac{1}{ac - b^2\cos\gamma - \psi}(cT + bT\cos\gamma - \psi) + cd\sin\gamma - bgLM_S\sin(\psi)\cos(\gamma - \psi); \tag{37}$$

$$\ddot{\psi} = \frac{1}{ac - b^2\cos(\gamma - \psi)}(agLM_s\sin\psi - bd\sin\gamma\cos(\gamma - \psi) - aT + b^2\sin(\gamma - \psi)\cos(\gamma - \psi)\dot{\psi}^2) \tag{38}$$

with:
$a = R_s(M_s + M_B) + I_B$
$b = R_s L M_B$
$c = I_B + L^2 M_B$
$d = gR_s(M_s + M_B)$

6 3D Model design

This section contains the information needed to develop an appropriate control for a 3D Ballbot model. In particular it is used the 3D modeling developed in [15] where, like 2D modeling of the *chapter5*, after analyzing the system, the Lagrangian method is used to derive the equations of motion. In this case it is considered a full three-dimensional geometry and all coupling effects. The proposed geometry for this 3D model consists of five solid bodies:

- 1 sphere;
- 3 omniwheels;
- 1 body with 3 motors;

consequently:

- the sphere is always in contact with the ground in an arbitrary point;
- the three omniwheels link the upper part of the sphere in three point at any time;
- The three torques T_1, T_2, T_3 generated by the motors and transformed by the gear are considered as the inputs of the system.
- It is supposed a tilt angles range from $-20°$ to $20°$, in order to operate in a scenario in which singularities do not to be considered.

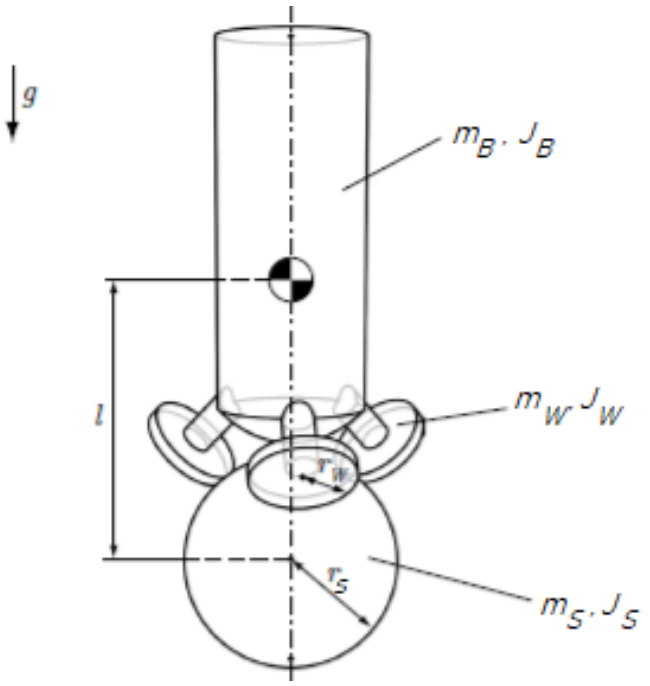

Figure 22: 3D-Ballbot modeling

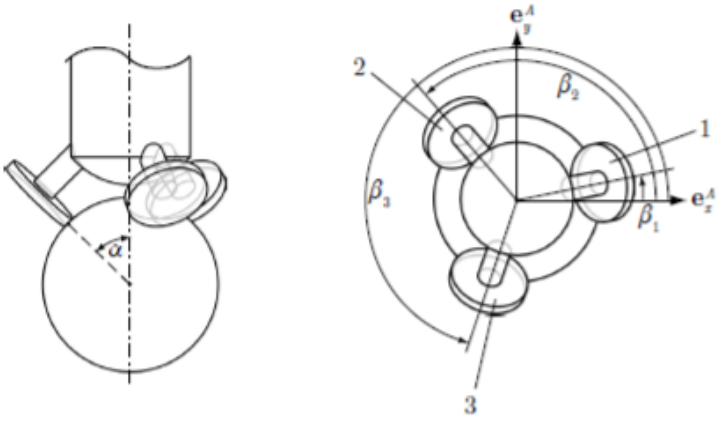

Figure 23: Omniwheel position on 3D-Ballbot modeling

6.1 3D Geometric parameters

Table 3: Geometric parameters of the 3D preliminary design

Parameter	Symbol	Value
Mass of the ball	m_S	2.29 [Kg]
Radius of the ball	r_S	0.125 [m]
Mass of the virtual actuating wheel	m_W	3 [Kg]
Mass of the body	m_B	9.2 [Kg]
Radius of the body(cylinder)	r_B	0.1 [m]
Inertia of the sphere	J_S	0.0239[Kgm^2]
Inertia of the body	$J_{B,x}$	2.026 [Kg m^2]
Inertia of the body	$J_{B,y}$	2.025 [Kg m^2]
Inertia of the body	$J_{B,z}$	0.092 [Kg m^2]
Inertia of one motor and wheel	J_W	0.00315 [Kgm^2]
Angle of the omniwheel contact point	α	45°
Angles of the omniwheel direction	$\beta_1, \beta_2, \beta_3,$	0°,120°,240°

6.2 Equations of Motion

As in the *subchapter5.3*, it is defined a set of variables as minimal coordinates q to describe the system through Lagrangian method and, for each body, it is analyzed its translational, rotational and potential energy . In this procedure it is chosen that the center of the ball is concurred with the plane zero potential energy.

$$\begin{bmatrix}\theta_x & \theta_y & \theta_z & \phi_x & \phi_y\end{bmatrix} \text{minimal coordinates } q \tag{39}$$

- Sphere:
  The sphere is only moving in the horizontal directions so, its potential energy U_S is equal to zero. Instead the sphere kinetic energy $T_S$is given by a contribute due to the translation and another one due to the rotation.

$$T_S = \frac{1}{2} m_S \left\| {}_I v_S \right\|^2 + \frac{1}{2} J_S \left\| {}_L w_S \right\|^2 \tag{40}$$

$$U_S = 0 \tag{41}$$

- Body:
  The moment of inertia of the omniwheels and of the motors are included as contribute on the moment of inertia of the body in order to consider their rotation about the inertial reference frame. Therefore, body potential energy contains motors and omniwheels. Body kinetic energy is calculated with respect to the center of the sphere P and not to its center of mass so, it is needed to

add a coupling term.

$$T_B = \frac{1}{2}\cdot m_B \cdot \|_I\vec{v}_{B,P}\|^2 + \frac{1}{2}\cdot J_B \cdot \|_A\vec{w}_{B,A}\|^2 + m_B \cdot (R_{AI}\cdot \dot{\vec{I}r}_{B,P})\cdot ({}_A\vec{w}_{B,A} \times {}_A\vec{r}_{PSA}) \tag{42}$$

$$U_B = -m_B \cdot G \cdot R_{IA} \cdot {}_A\vec{r}_{PSA} \tag{43}$$

where R_{IA} is a rotation matrix between Inertial reference frame I and the body fixed reference system A, ${}_A\vec{r}_{PSA}$ is the vector from the center of the sphere P to the center of gravity in the body fixed reference system A and G is the gravity vector defined as G=[0 0 -g]$^{\mathrm{T}}$.

- Omniwheels:
  As a consequence of the simplification done above for the wheels, it is considered only its additional Kinetic rotational energy around the respective motor axis.

$$T_{Wi} = \frac{1}{2}\cdot {}_AJ_{Wi}\cdot {}_Aw_{Wi}^2 \qquad \text{with i= 1,2,3} \tag{44}$$

As the 2D-procedure, the equations of motion are derived applying Lagrange equation:

$$\frac{d}{dt}\left(\frac{\delta T}{\delta \dot{\vec{q}}}\right) - \frac{\delta T}{\delta \vec{q}} + \frac{\delta U}{\delta \vec{q}} - \vec{f_{NP}} = 0; \tag{45}$$

where:

- $T = T_S + T_B + T_{W_1} + T_{W_2} + T_{W_3}$;
- $U = U_B$;
- f_{NP}: represent the non-potential forces of the system that include both the torques T_1, T_2, T_3 and the counter torques $T_{C_1}, T_{C_2}, T_{C_3}$ acting on the body[15].

6.3 3D Linearized System - State-Space formulation

The equation of motion derived are a big set of non linear differential equations that is not possible to obtain directly so, it is necessary to use an adequate software for breaking down the calculations and applying simplification on each step. A huge size reduction of the system it is done analyzing only a linear version of the system. In this way, considering:

$$\begin{cases} \dot{\vec{x}} = A\vec{x} + B\vec{u} \\ \vec{y} = C\vec{x}) + D\vec{u} \end{cases} \quad \textit{state space representation} \tag{46}$$

with:

$$\vec{x} = [\theta_x, \dot{\theta}_x, \theta_y, \dot{\theta}_y, \theta_z, \dot{\theta}_z, \psi_x, \dot{\psi}_x, \psi_y, \dot{\psi}_y]^T \qquad \textit{state vector} \tag{47}$$

$$\vec{u} = [u_1, u_2, u_3]^T = [T_1, T_2, T_3]^T \qquad \textit{control signal} \tag{48}$$

$$\vec{y} = \vec{x} \tag{49}$$

Below are shown the state space matrices derived in [15] from a system with a linearization on zero equilibrium point and, subsequently, with a linearization on an arbitrary operation point.

$$A_0 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 37.69 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 37.73 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ -73.02 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -73.09 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \textit{State matrix} \tag{50}$$

$$B_0 = \begin{bmatrix} 0 & 0 & 0 \\ 4.02 & -2.01 & -2.01 \\ 0 & 0 & 0 \\ 0 & 3.485 & -3.485 \\ 0 & 0 & 0 \\ -10.76 & -10.76 & -10.76 \\ 0 & 0 & 0 \\ -13.48 & 6.738 & 6.738 \\ 0 & 0 & 0 \\ 0 & -11.68 & -11.68 \end{bmatrix} \qquad \textit{Input matrix} \tag{51}$$

According with assumptions (40):

$$C_0 = \begin{bmatrix} I_{10} \end{bmatrix} \qquad \textit{Output matrix} \tag{52}$$

$$D_0 = \begin{bmatrix} 0_{10\text{x}3} \end{bmatrix} \tag{53}$$

The arbitrary linearization point chosen is:

$$\vec{p} = \begin{bmatrix} \theta_x \\ \dot{\theta}_x \\ \theta_y \\ \dot{\theta}_y \\ \dot{\theta}_Z \\ \dot{\phi}_x \\ \dot{\phi}_y \end{bmatrix} = \begin{bmatrix} 0.32rad \\ -0.5rad/s \\ 0.1rad \\ 0.35rad/s \\ -0.2rad/s \\ 6.3rad/s \\ 2.3rad/s \end{bmatrix} \tag{54}$$

$$A_p = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 12.85 & -0.5234 & 64.1 & 1.164 & 0 & 2.361 & 0 & 0.09295 & 0 & 0.07759 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -62.57 & 3.008 & 32.44 & -5.953 & 0 & -3.789 & 0 & -0.1806 & 0 & -0.04806 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -38.94 & -8.147 & 4.429 & 21.06 & 0 & 6.915 & 0 & 1.066 & 0 & 1.703 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ -13.2 & -1.236 & -28.15 & 1.551 & 0 & -2.268 & 0 & 0.03377 & 0 & 0.1916 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 45.18 & 0.3581 & -42.06 & -3.098 & 0 & 3.54 & 0 & -0.3492 & 0 & -0.206 \end{bmatrix} \quad \textit{State matrix} \tag{55}$$

$$B_p = \begin{bmatrix} 0 & 0 & 0 \\ 2.327 & -2.863 & -3.038 \\ 0 & 0 & 0 \\ 3.925 & 6.862 & 1.02 \\ 0 & 0 & 0 \\ -10.17 & -7.92 & -11.51 \\ 0 & 0 & 0 \\ -11.38 & 6.901 & 6.871 \\ 0 & 0 & 0 \\ -2.231 & -13.96 & -8.24 \end{bmatrix} \quad \textit{Input matrix} \tag{56}$$

also in this case, according with assumption (40) :

$$C_p = \begin{bmatrix} I_{10} \end{bmatrix} \qquad \textit{Output matrix} \tag{57}$$

$$D_p = \begin{bmatrix} 0_{10\text{x}3} \end{bmatrix} \tag{58}$$

A_p and B_p matrices evince strong coupling terms and point out how it is confining a modeling based on a decoupled system. In the *subchapter9.3* it is developed a 3D system control, also in this case it is chosen an LQR controller to obtain a robust control, with a plant implemented using A_0,B_0,C_0,D_0 matrices for reasons of simplification.

7 LQR - Linear Quadratic Regulator Control

The LQR controller, as part of the optimal controls and, more in general, of the automatic controls and LTI (Linear Time-Invariant) dynamic systems, is a dynamic compensator obtained by following the minimization of a cost index J(x,u) which depends on the state x(t) and on the control u(t) :

$$\min_{u(t),t\in[0,t_f]} J(u,x) = \min_{u(t),t\in[0,t_f]} x^T(t_f)Su(t_f) + \int_{t=0}^{t=t_f} (x^T(t)Qx(t) + u^T(t)Ru(t))\,dt \tag{59}$$

Where:

$$Q, S \in R^n : Q = Q^T \geq 0, \quad S = S^T \geq 0; \in R^p : Q = Q^T > 0;$$

For each Q symmetric positive semi-definite matrix and for each R symmetric positive definite matrix always exists an $u_{ott}(t)$ solution of the optimum control LQR that minimize the cost index J(x,u). If the LTI system is *reachability*and *observability* minimizing the cost index J(x,u) , make it bounded, also the system will be stable.
A time-invariant controller can be realized when t_f tends to infinity. As a consequence of the infinite time horizon, the final cost:

$$x^T(t_f)Su(t_f)$$

has been removed from finite time cost index, so:

$$\min_{u(t),t\in[0,\infty]} J(u,x) = \int_{t=0}^{\infty} (x^T(t)Qx(t) + u^T(t)Ru(t))\,dt \tag{60}$$

Essentially, the obtained control $u_{ott}(t)$ is a linear function of the state and of several matrix including P(t) solution of DRE (Riccati Differential Equation) if the control is a finite-time control, or of P (constant) solution of ARE (Ricatti Algebric Equation) otherwise.

- Finite horizon time:
  - Control:

$$u_{ott}(t) = -K_{ott}(t)x(t)$$

  - State Feedback Controller:

$$K_{ott}(t) = R^{-1}B^T P(t)$$

  - DRE whose solution provides P(t):

$$\frac{dP(t)}{dt} = A^T P(t) + P(t)A + Q - P(t)BR^{-1}B^T P(t)$$

- Infinite horizon time:

  - Control:

$$u_{ott}(t) = -K_{ott}(t)x(t)$$

  - State Feedback Controller:

$$K_{ott}(t) = R^{-1}B^T P$$

  - ARE whose solution provides P:

$$[0] = A^T P + PA + Q - PBR^{-1}B^T P$$

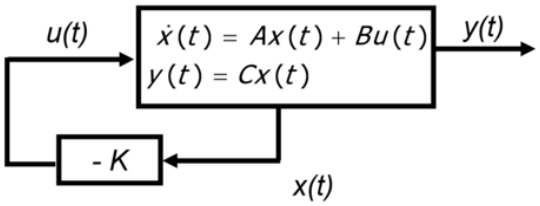

Figure 24: Static State feedback control architecture.

7.1 System Reachability:

Considering the matrix M_r as defined:

$$M_r = \begin{bmatrix} B & AB \cdots & A^{n-1}B \end{bmatrix}$$

If $\rho(M_r)$ is maximum, so if:

$$\rho(M_r) = n;$$

the dynamic system:

$$\dot{x}(t) = Ax(t) + Bu(t)$$

is said *reachable*.
In other terms the couple (A,B) is reachable for short.

7.2 System Observability

Considering the matrix M_o as defined:

$$M_o = \begin{bmatrix} C & CA \cdots & CA^{n-1} \end{bmatrix}^T$$

If $\rho(M_o)$ is maximum:

$$\rho(M_o) = n;$$

the dynamic system:

$$\dot{x}(t) = Ax(t) + Bu(t)$$

$$y(t) = Cx(t)$$

is said *Observable*.
In other terms the couple (A,C) is observable for short.

7.3 Tuning LQR Control Laws

1. Derive a possible State-Space Description of the system to control:

   $$\dot{x}(t) = Ax(t) + Bu(t)$$

   With:

   $A \in R^{\text{nxn}}$ is the State matrix; $B \in R^{\text{nxp}}$ is the Input matrix;

   As said before, the design of the control system is transformed into a minimization problem of the cost index J(u,x) so the goal is to minimize the energy of the system by commands over time.

2. Tuning the LQ Regulators implies choosing the weight matrices Q and R.
   Q and R are typically chosen as diagonal matrices so, for a system of n states and p controls there will be (n+p) parameters to choose;

3. The diagonal value q_{ij} and r_{ij} of Q and R respectively , are chosen according to the relative importance of each state control variable, bearing in mind that it

is needed to have $q_{jj} \geq 0$ $r_{jj} > 0$;

(a) As first approach, it is possible to choose q_{ij} and r_{ij} so that all the state and input components values appear in the cost function with almost the same order of magnitude;

(b) After this first iteration , values are modified to impose the desired performance;

(c) Anyhow, the important aspect is the relative value of a single weighting coefficients with respect to the others.

In general:
Q matrix defines the weight on the states;
R matrix defines weights on the control input in the cost function.

- If $R \gg Q$, cost function is weighted by the control effort u(t) and an expansive control solution is obtained.

- If $Q \gg R$, cost function is weighted by the state errors and the system's response becomes arbitrarily fast.

Since there is a trade-off between the two, you may want to keep Q as an identity matrix and after choose R. It is possible to choose a large R, if there is a limit on the control output signal (for instance, if large control signals introduce sensor noise or cause actuator's saturation) and choose a small R if having a large control signals is not a problem for the analyzed system.
Once the weight matrices have been defined , K_{LQR} matrix will be compute through numerical techniques using an apposite MATLAB commands:

» `[K_LQR]=LQR(A,B,Q,R);`

Where:
A is the State matrix of the state-space system representation;
B is the Input matrix of the state-space system representation;
Q,R are the previous defined weight matrices of the LQR Control.

or:
» `[K_LQR,P,E]=LQR(A,B,Q,R);`

to obtain also the associated algebric Riccati equation solution `P` and the closed-loop eigenvalues `E=EIG(A-BK`$_{\texttt{LQR}}$`)`.

8 MATLAB implementation

In an apposite *file.m* are defined all the necessary parameters to calculate the numeric value of the state space matrices A,B,C,D computed on *chapter5* and, accordingly, to evaluate the transfer function that describes the linearized 2D and 3D modeling of the ballbot, in order to use it as *plant t.o.f.* on the Simulink feedback control for simulating the behavior of the system. It is chosen to use an LQR control with an integrative stage to track the error of the variable chosen to be measured, therefore two other matrices A_{tot}(square matrix), B_{tot}, are defined coherently with the matrices A, B and with a new state given by the integrative error on the selected measure. The size of A_{tot}, B_{tot} obviously changes with respect to the number of variables chosen to be measured. The K_{LQR} gain will be compute using A_{tot}(square matrix), B_{tot} in order to obtain also the relative integrative K_{LQR} gain, K_{e_i}. The latter part of the `MATLAB code` of each *file.m* is dedicated to the implementation of a `System Observer` which, exploiting the Observability proper of the system and using a pole placement method to develop a stable system,it is able to estimate a real state vector using as input: the input of the plant and the plant output variables that make it observable, so:

$$u = -K_{tot} \cdot x_{tot} = -\begin{bmatrix} K_i & K_{LQR} \end{bmatrix} = \begin{bmatrix} q \\ x \end{bmatrix} \tag{61}$$

$$\hat{x}(k+1) = (A_d - L \cdot C_d)\hat{x}(k) + \begin{bmatrix} B_d & L \end{bmatrix} \begin{bmatrix} u \\ y \end{bmatrix} \tag{62}$$

where $\hat{x}(k)$ is the estimated state vector from the observer system.
On the MATLAB code Appendix are shown several *file.m* respectively for each simulation test. Inside it also contained how `L matrix` and `observer system` are defined.

9 Simulink

In this chapter are presented Simulink schemes block of the 2D and 3D Ballbot modeling based on consideration both about type of reference and variables chosen to be measured.

9.1 2D- Position control

The figure below shown the Simulink feedback control with a step input reference, amplitude=4[deg], on the sphere position *theta* and with a block structure following the premises done on the *MATLAB Implementation chapter*.

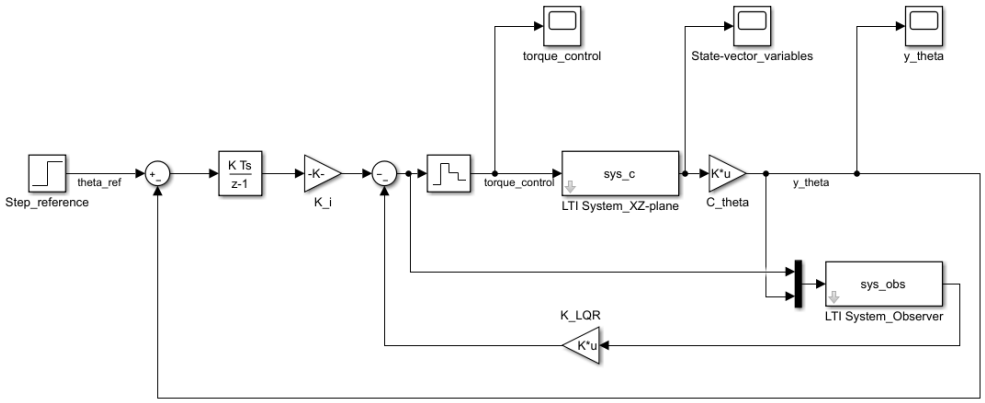

Figure 25: Simulink system control block with reference on the sphere position *theta*.

9.1.1 Simulation

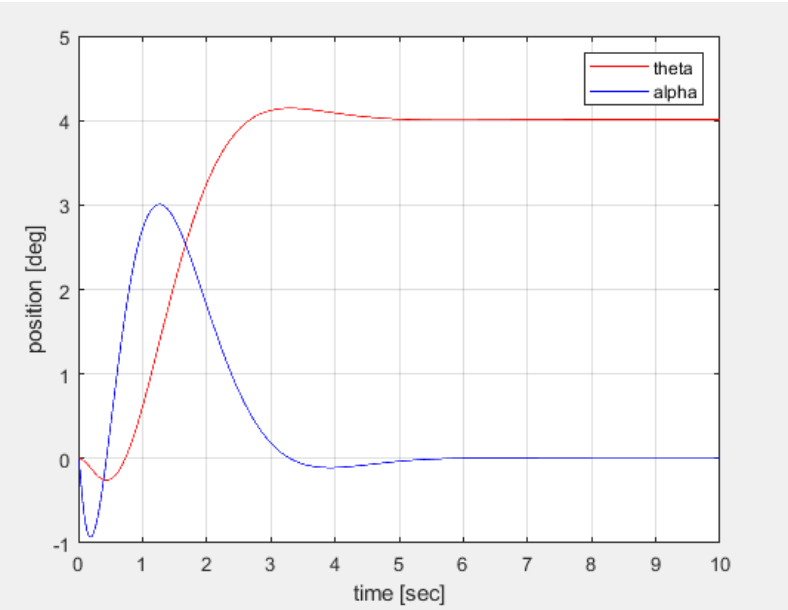

Figure 26: Measured *theta* and output *alpha*

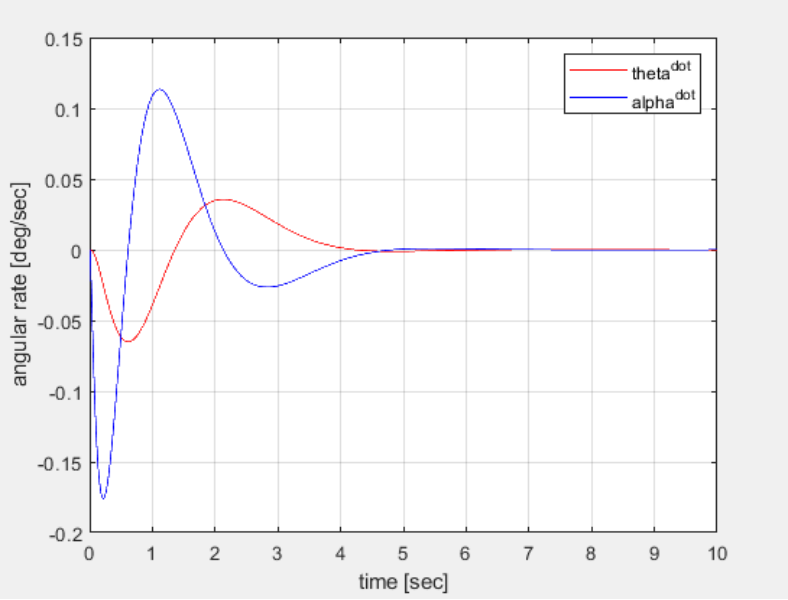

Figure 27: Angular rate of the sphere *theta-dot* and of the body *alpha-dot*

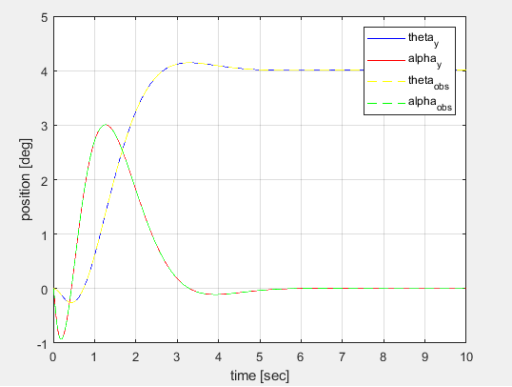

Figure 28: Comparison between plant-output and observer-output of system position

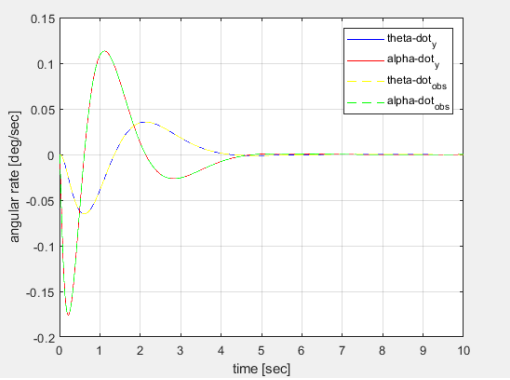

Figure 29: Comparison between plant-output and observer-output of system angular rate

9.2 2D - reference on body position

The following Simulink feedback control shown a step input reference with amplitude=4[deg], on the body position *alpha*. Also in this case it is used a block structure coherently with the assumptions of *chapter8*. Firstly, it is simulated with a pulse input reference signal with a period of 7.5[sec] and an amplitude of 4[deg] and, in a second moment, with a signal manually generated using a "signal builder" with 0[deg] amplitude from 0[sec] to 2[sec], 4[deg] from 2[sec] to 5[sec] , and 0[deg] amplitude from 5[sec] to the end of the simulation.

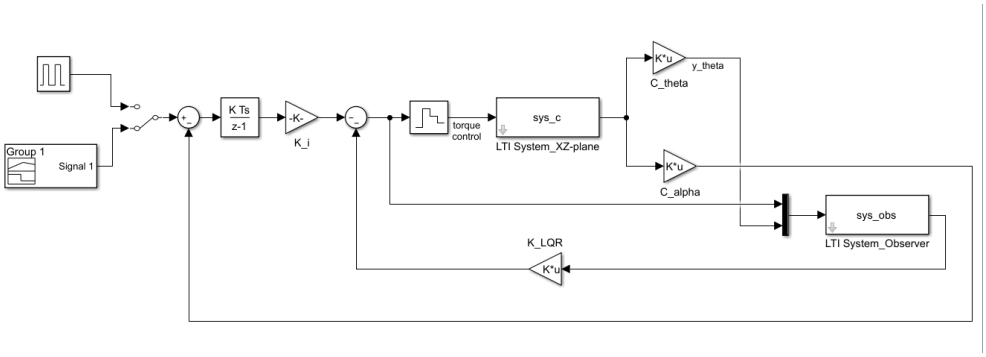

Figure 30: Simulink system control block with reference on the body position *alpha*.

9.2.1 Simulation

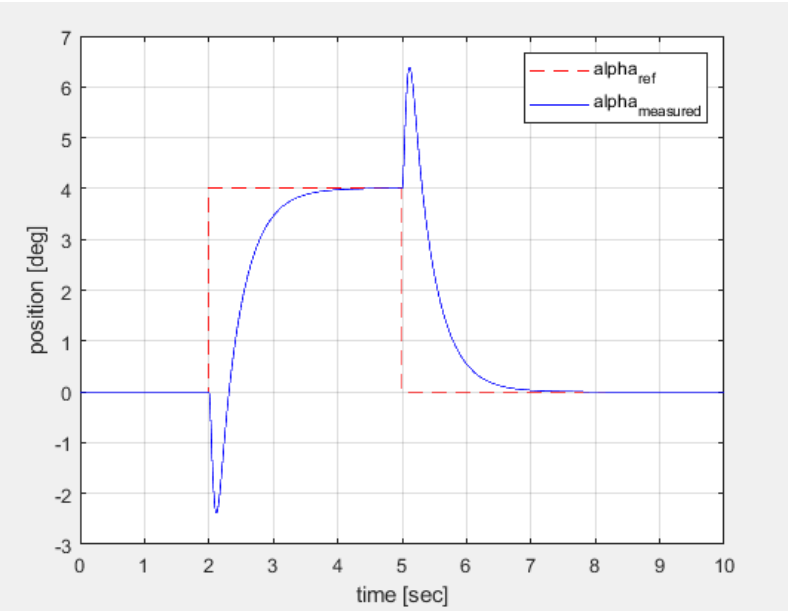

Figure 31: Comparison between measured *alpha* and signal built reference on *alpha*

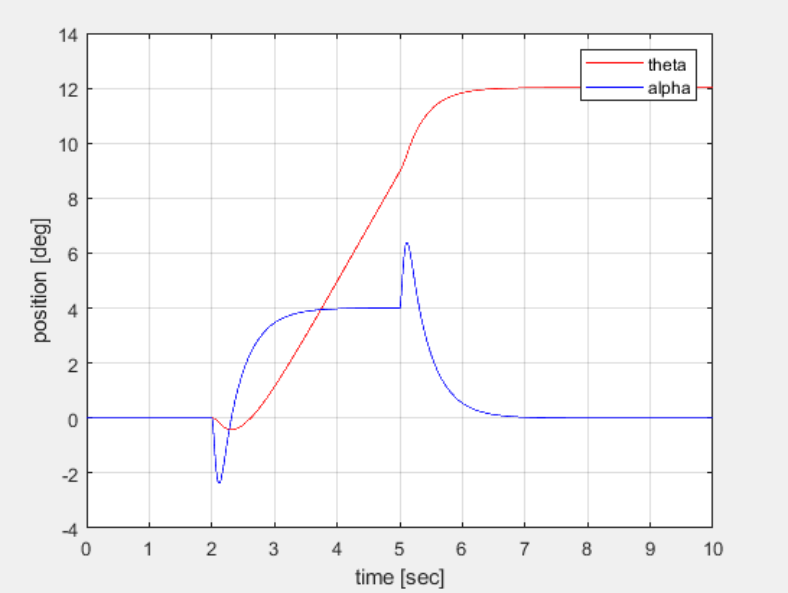

Figure 32: Measured *alpha* and output *theta* - signal built reference on *alpha*

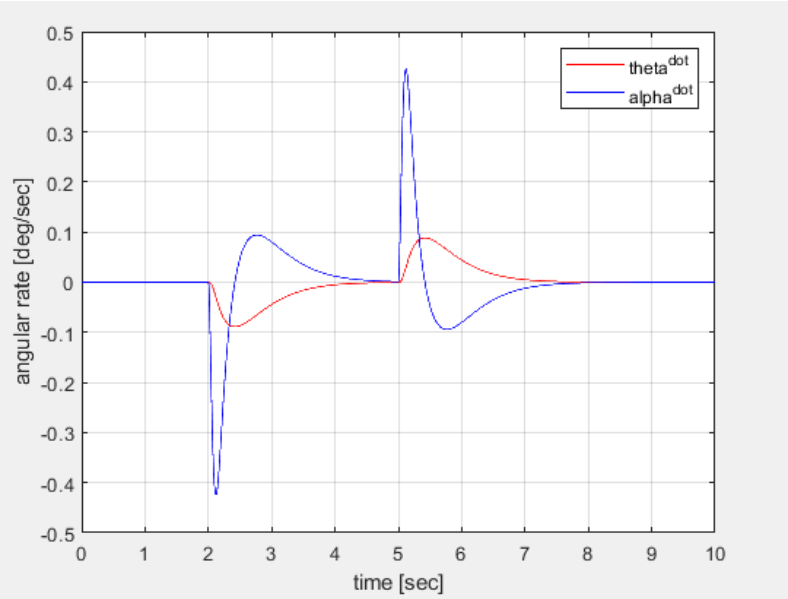

Figure 33: Angular rate *theta-dot* and angular rate *alpha-dot* - signal built reference on *alpha*

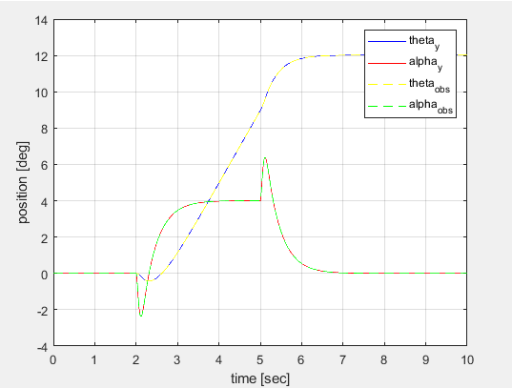

Figure 34: Comparison between plant-output and observer-output of system position - signal built reference on *alpha*

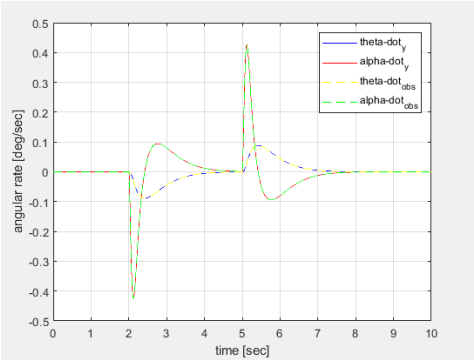

Figure 35: Comparison between plant-output and observer-output of system angular rate - signal built reference on *alpha*

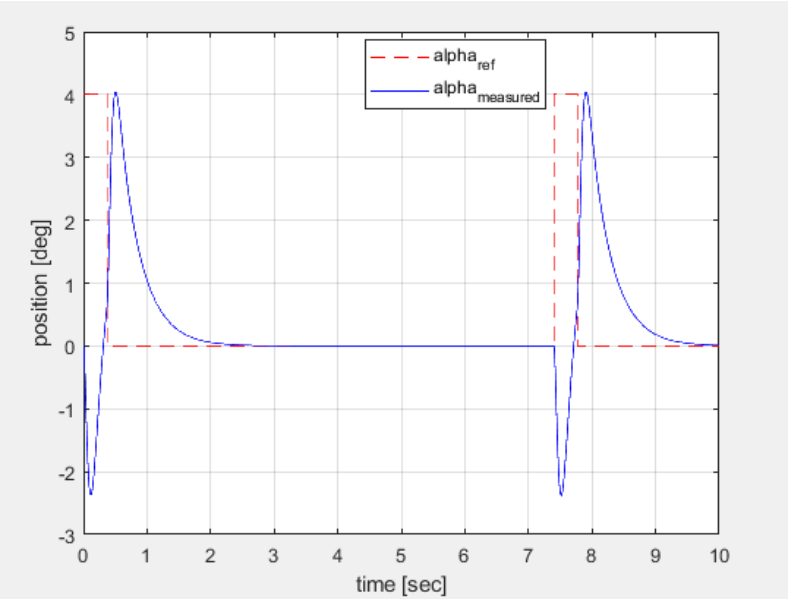

Figure 36: Comparison between measured *alpha* and pulse reference on *alpha*

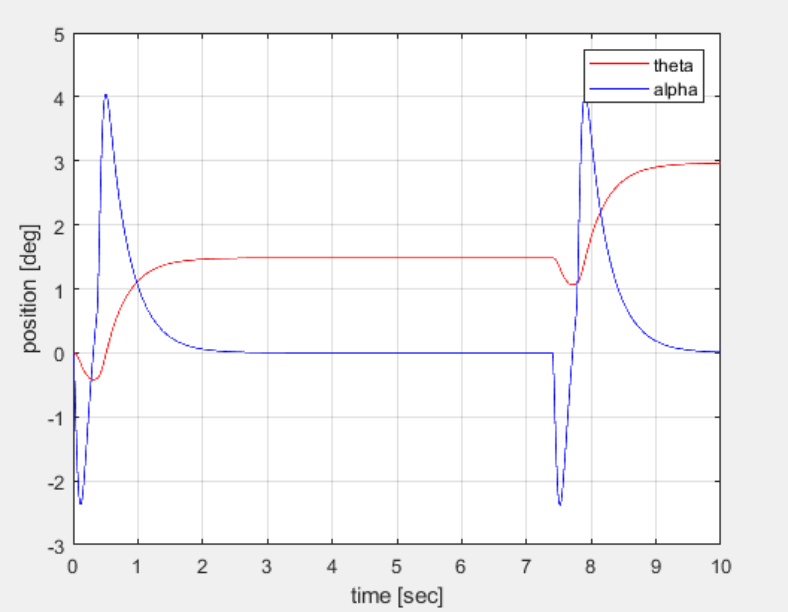

Figure 37: Measured *alpha* and output *theta* - pulse reference on *alpha*

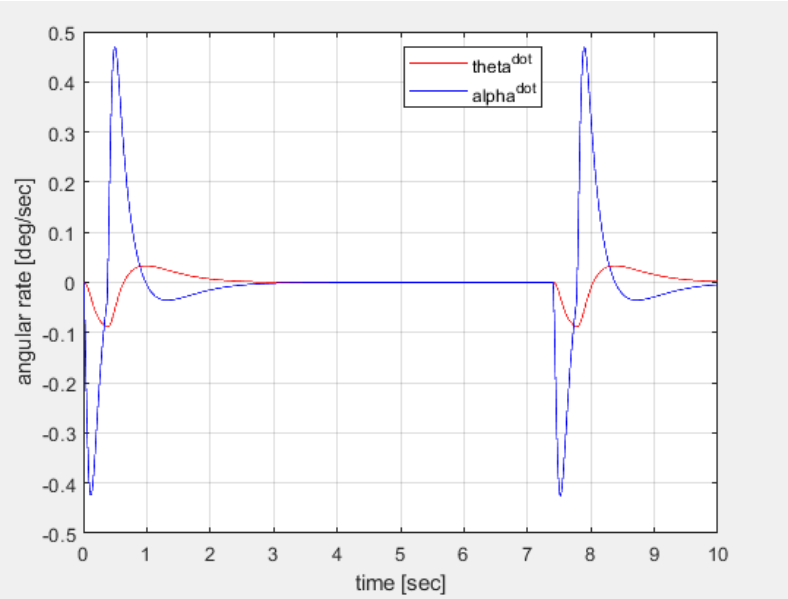

Figure 38: Angular rate *theta-dot* and angular rate *alpha-dot* - pulse reference on *alpha*

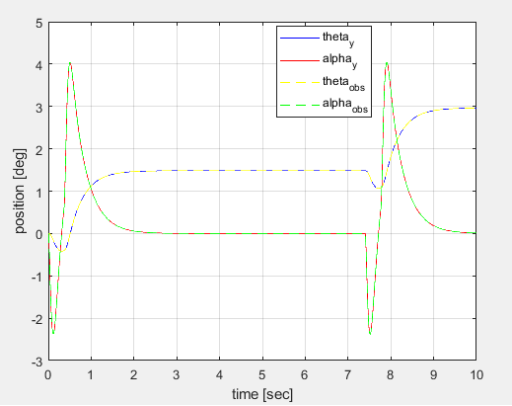

Figure 39: Comparison between plant-output and observer-output of system position -pulse reference on *alpha*

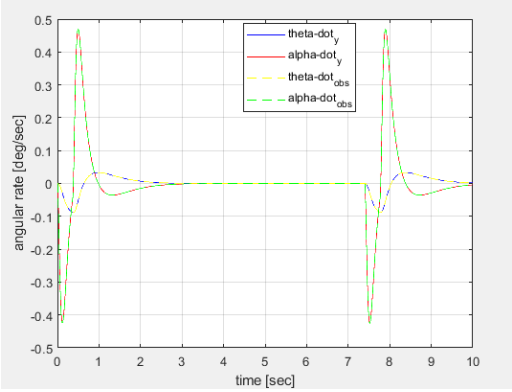

Figure 40: Comparison between plant-output and observer-output of system angular rate - pulse reference on *alpha*

9.3 2D- Velocity control

The figure below shown the Simulink feedback control with a step input reference of an amplitude=1.5 from 2[sec] to 5[sec] on the sphere velocity variable *thetadot*:

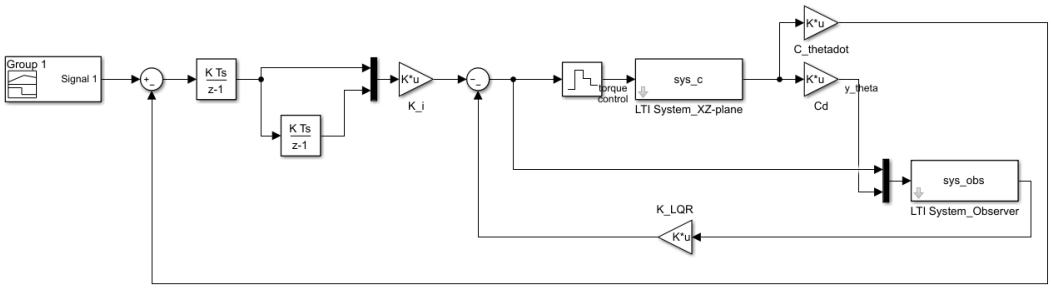

Figure 41: Simulink 2D-system control block with step reference from 2[sec] to 5[sec] on sphere velocity *thetadot*

9.3.1 Simulation

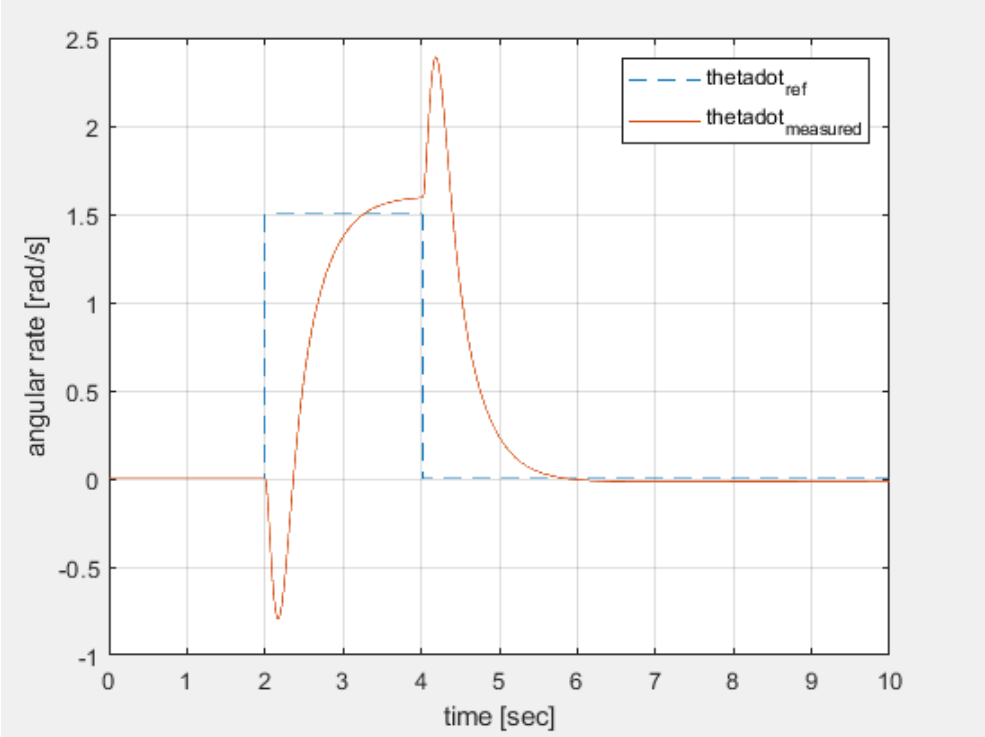

Figure 42: Comparison between measured sphere velocity *thetadot* and velocity reference *thetadot-ref*

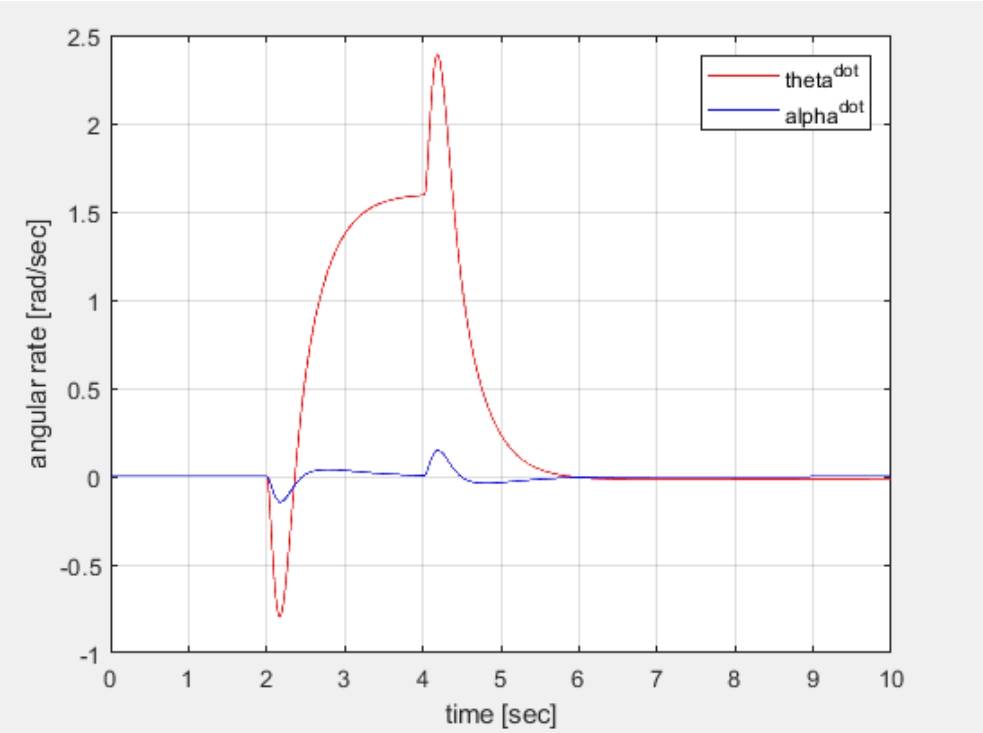

Figure 43: Angular rate *alphadot* - step sphere velocity reference

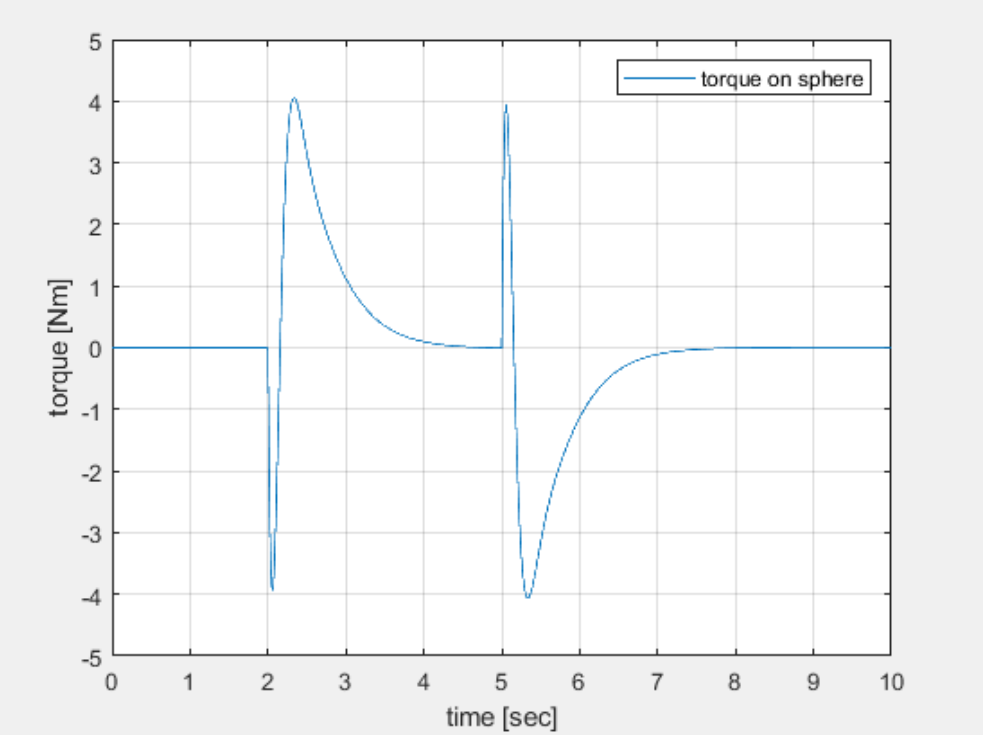

Figure 44: Sphere torque - sphere velocity step reference

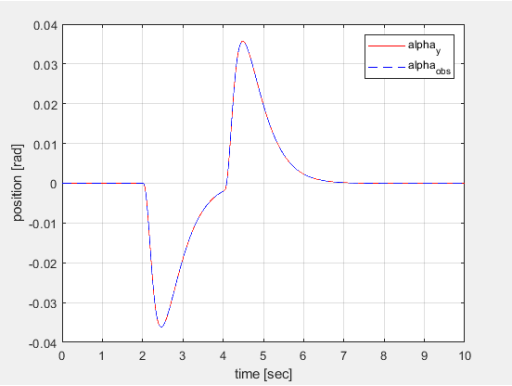

Figure 45: Comparison between plant-output and observer-output of body position *alpha* - step reference on velocity

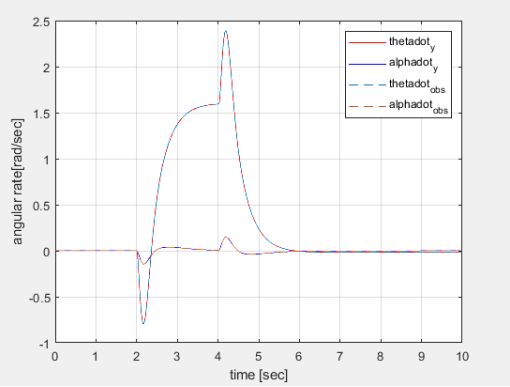

Figure 46: Comparison between plant-output and observer-output of system angular rate - step reference on velocity

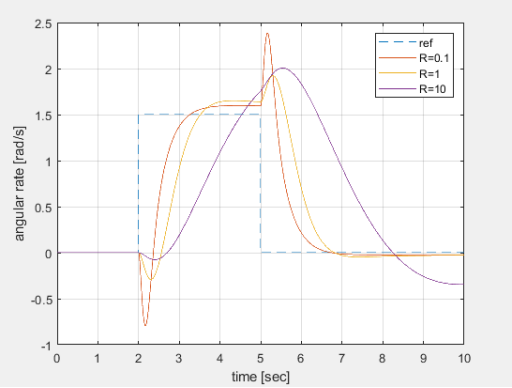

Figure 47: Comparison between *thetadot-measured* for different R matrix weight values

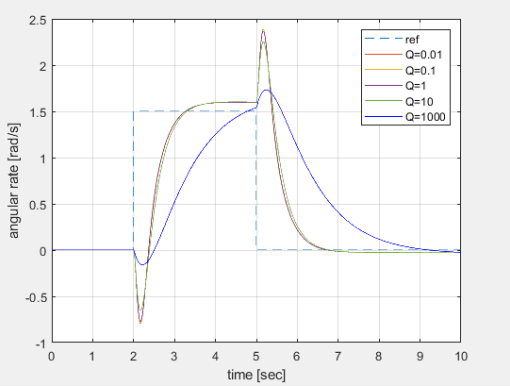

Figure 48: Comparison between *thetadot-measured* for different Q matrix weight values

9.4 3D-Position control

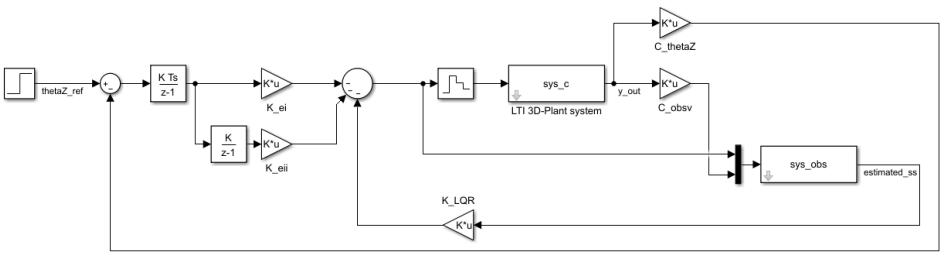

Figure 49: Simulink 3D-system control block with step reference on position *thetaZ*

```
K_lqr =

  82.8501    0.1218  145.5569   26.5481  139.6796    1.5452  -30.3167   -3.5308   10.1815    5.5989   10.6715   -0.4405
  -8.1275   -0.0384   44.5652   15.0800  125.8768    8.0225   -3.3253   -0.1602   28.9769    5.2951   -6.4013    0.2603
 -43.1641   -0.0538  -89.6903   -5.4401   45.7369   10.3186   15.2668    2.3846    2.6424   -0.2534   28.5198    4.5205
```

9.4.1 Simulation

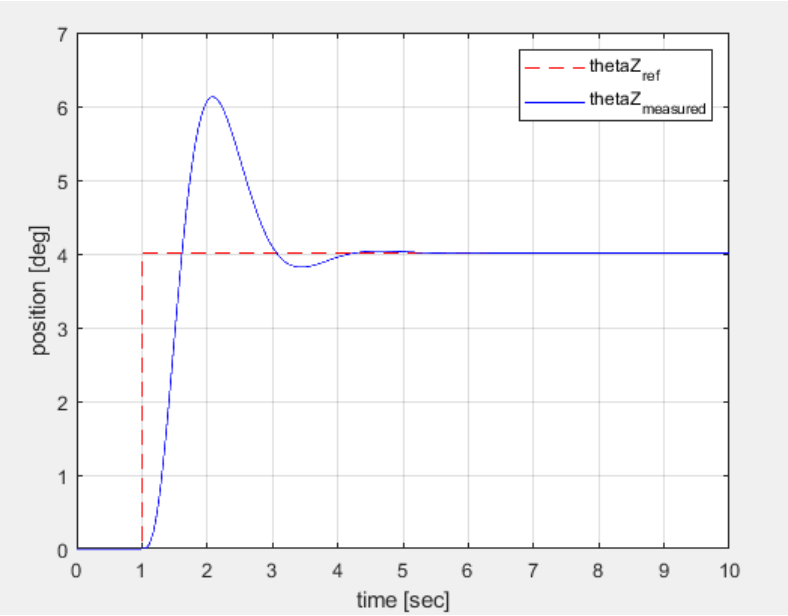

Figure 50: Comparison between measured *thetaZ* and step reference on *thetaZ*

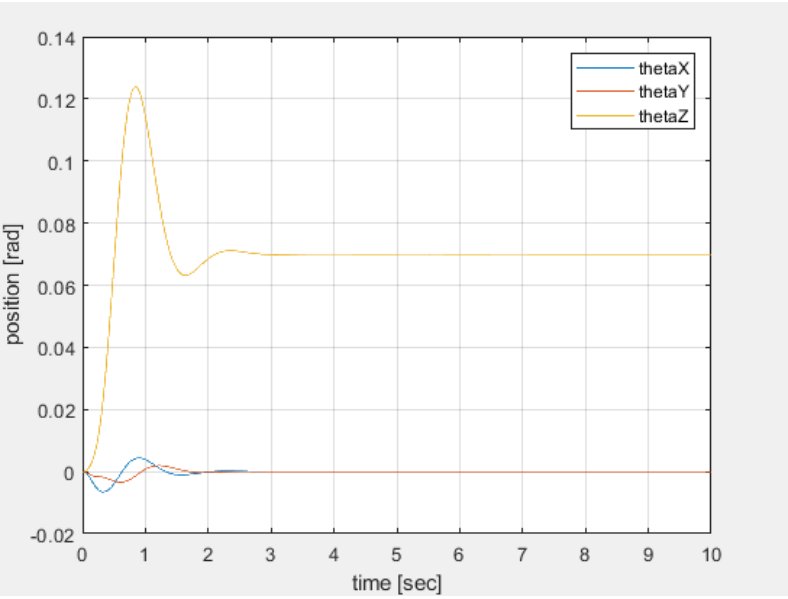

Figure 51: Output position of the sphere *thetaX*, *thetaY*,*thetaZ* with step reference on *thetaZ*

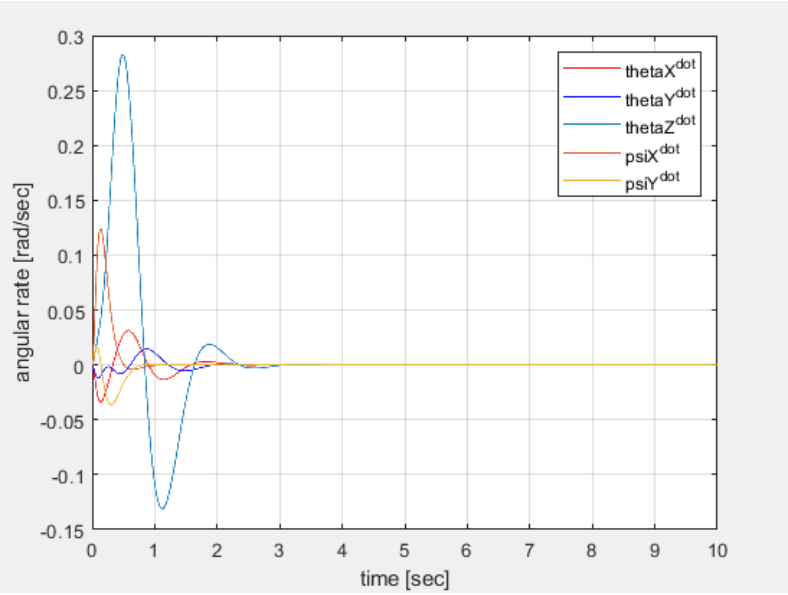

Figure 52: Output angular rates with step reference on the sphere position *thetaZ*

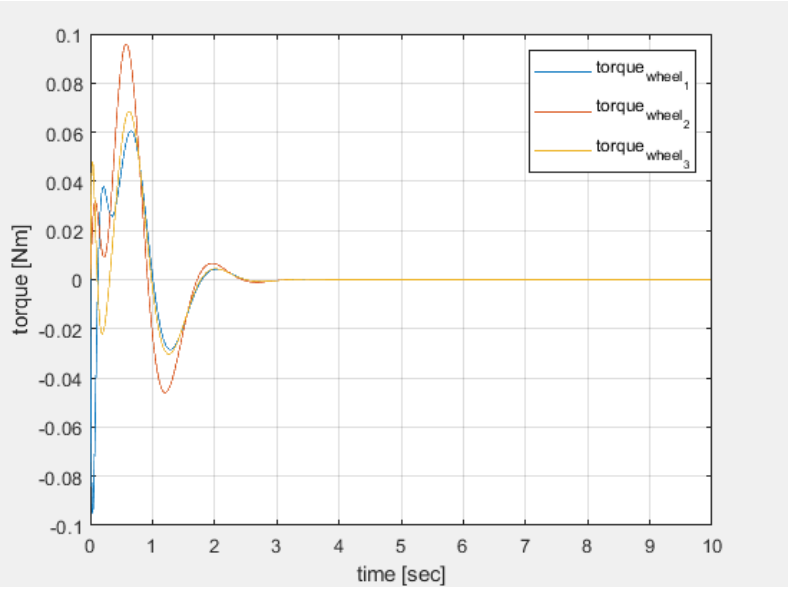

Figure 53: Torque signal on the $omniwheel_{1,2,3}$ respectively with a step reference on the sphere position *thetaZ*

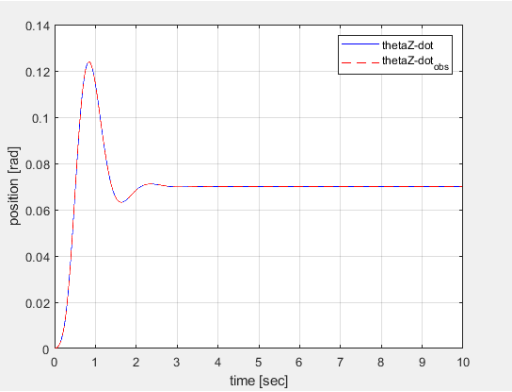

Figure 54: Comparison between plant-output and observer-output of system position *thetaZ* - step reference on *thetaZ*

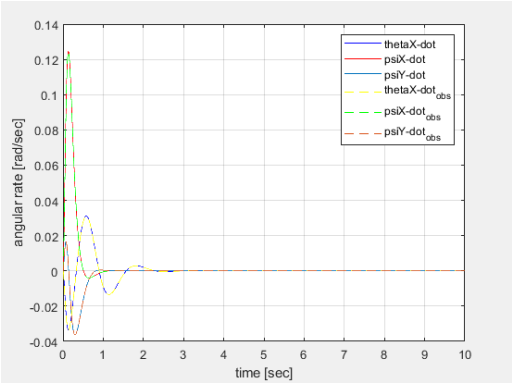

Figure 55: Comparison between plant-output and observer-output of system angular rate - step reference on *thetaZ*

10 Simscape

Simscape is an extension of the *Simulink software* that allows the simulation of multidomain physical systems. In this section it is shown a Simscape block scheme of the Ballbot system. Using the 3D geometric characteristic described on *subchapter6.1*, each elementary body component of the Ballbot it is created in Solidworks software and imported on Simscape tool where they were assembling as in *Figure47*. The contact between sphere and each wheel it is performed through the Simscape block "Sphere to Tube Force" while the contact plane-sphere with the Simscape block "Sphere to Plane Force". These two contact-blocks belong to an external library that must be downloaded called "Simscape Multibody Contact Forces Library" that contains contact force models and force laws for multibody modeling in Simscape Multibody.
In *Figure50* it is shown the 3D-Ballbot model obtained.

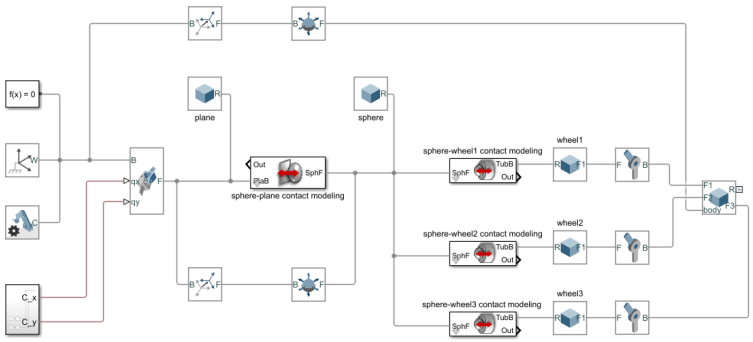

Figure 56: Simscape 3D-Ballbot model scheme

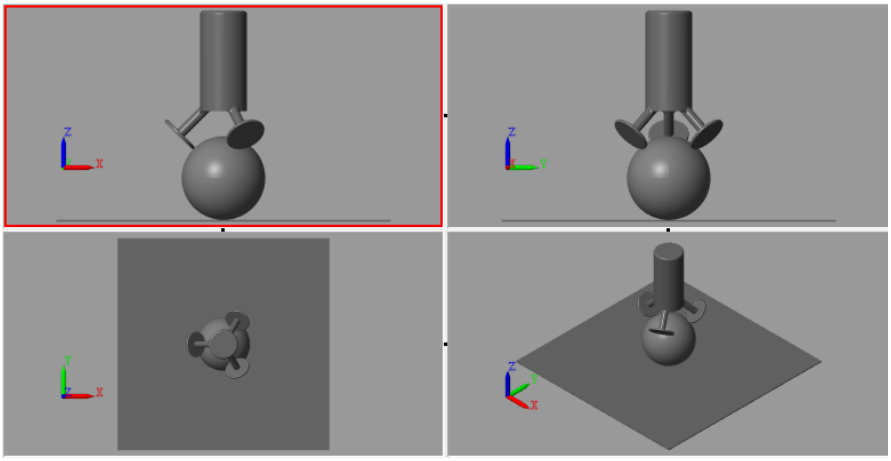

Figure 57: Simscape 3D-Ballbot model view

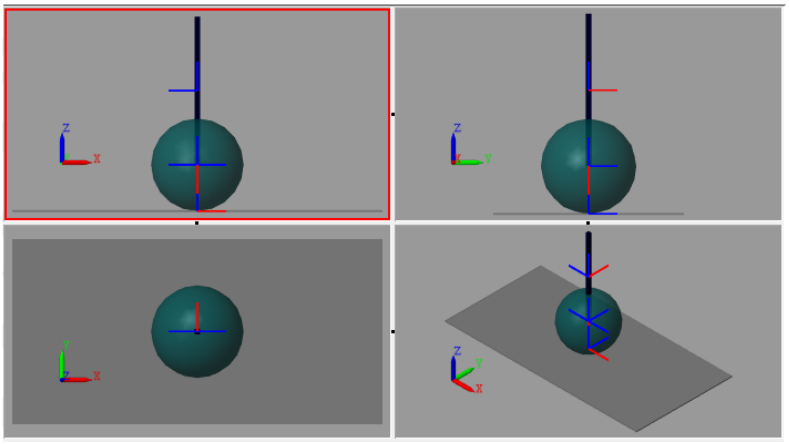

Figure 58: Simscape 2D-Ballbot model view

11 ARDUINO electronic platform

Arduino is an open-source hardware platform developed in 2003 at the Interaction Design institute of Ivrea, a post-graduate school created by the collaboration between Olivetti and Telecom Italia. The name of the platform derives from the name of a bar of Ivrea frequented by the founding members.
Arduino is composed of a series of electronic boards equipped with a microcontroller, supported by an integrated development environment, *Aduino IDE (Integrated Development Enviroment)*, that allows its programming. The programs developed in the *Arduino IDE* are commonly called *sketches* and are developed in a simple programming language, derived from `C` and `C++`, called `Wiring`. All The software is free and this is exactly what has created the strength of Arduino: an online community of developers that continuously discusses the development of projects with an Arduino electronic, increasingly expanding the community and making it accessible even to the less experienced. Arduino makes possible in fact to create a range of projects and device, in a relatively simple and rapid way, both for personal and scientific use. The basic structure of an Arduino microcontroller consists of pins connected to the I/O ports (Analogue and Digital), a voltage regulator and, when necessary, a USB interface that allows connection with the computer used to program it. Through the input channels the card receives information, data collected by external sensors, the developed program reworks the data and through the output channels provides the instructions that allow interaction with the outside through suitable actuators. To date, more than 17 versions of Arduino, more and more avant-garde, have been marketed, to which almost every electronic object can be connected, from computers, sensors, actuators and displays. This development has led Arduino to position itself in a good place in the IoT(Internet of Things) market in which objects can acquire an optimal position thanks to the connection to the networks.

11.1 Arduino MKR1000

In this project is considered an Arduino MKR1000 version, a board that combines the features of Arduino Zero and WIfi-Shield, integrating directly into a Wifi module that allows you to connect the Arduino board to the Internet wirelessly. It is based on a microcontroller developed by Atmel, ATSAMSW25 Soc and it is equipped with a USB port that can be used to supply power (5V) to the board and allow the connection of the hardware platform to the computer. Unlike other versions, I/O pins operate at 3.3V : by applying a voltage greater than 3.3V to any I/O pins, the microcontroller may be damaged. In addition, the Arduino MKR1000 is able to operate even without the connected Li-Po battery and has a limited power consumption. Alternatively, output to 5V digital devices is possible, but bidirectional communication with 5V

devices requires an appropriate level shift in order to adequate the ArduinoMKR100 voltage dynamic to the input signal voltage.The Arduino-MKR1000 interaction with the outside takes place through MKR MotorCarrier shield, a dedicated piloting board, connected on it using apposite pins, that allows the driving of various actuators.

11.2 Technical specifications

Table 4: Arduino MKR1000 - Technical specifications

Microcontroller	SAMD21 Cortex- M0 + 32bit low power ARM MCU
Board Power Supply (USB/VIN)	5V
Circuit Operating Voltage	3.3V
Digital I/O pins	8
PWM pins	12(0,1,2,3,4,5,6,7,8,10,A3-or18,A4-or19)
UART	1
SPI	1
I2C	1
Analog Input Pins	7 (ADC 8/10/12bit)
Analog Output Pins	1 (DAC 10bit)
External Interrupts	8 (0,1,4,5,6,7,8,A1-or16,A2-or17)
DC current per I/O pin	7mA
Flash Memory	256KB
SRAM	32KB
EEPROM	no
UART	1
Clock Speed	32.768KHz (RTC), 48MHz
LED-BUILTIN	6
Full-Speed USB Device and embedded host	
Length	61.5[mm]
Width	25[mm]
Weight	32[gr]

11.3 Sensors

Sensors are devices or dedicated modules used to estimate the condition of the robot and to give information about its surrounding environment. The data collected by the sensors, are passed to a controller to enable appropriate behavior. All the sensors on the market need an adequate electronic circuit that allows the interface to the microcontroller it is chosen to use.
In this section are described some Arduino compatible sensors selected for the Sensoring stuff of the Ballbot model to develop.

11.3.1 Ultrasonic Sensor:

Figure 59: Ultrasonic sensor

The Ultrasonic sensor compatible with Arduino is a device able to capture the presence of an object in front of it detecting its distance through two transducers: one send out a high frequency sound burst (typically out of the auditory range for humans or domestic animals) and the other one feels for whatever is bouncing back (Figure 23). The elapsed time between the emitted sound burst and the one received gives indication to estimate the distance to the object in front of the sensor. In the Figure.22 is shown a typical low-cost ultrasonic sensor, the two transducer appears look like small cylinders :the sensors are marked "T" for transmitter on the left and "R" for receiver on the right. The sensor compute the distance from the object in front of it using the following relation below:

$$distance = \frac{(time\,of\,flight \;\cdot\; speed\,of\,sound)}{2} \tag{63}$$

where:
time of flight is the elapsed time from the emitter sending the burst to the receiver getting it back;
speed of sound is sound travels speed considered equal to 332 meters per second (1,087 ft/s) in normal environmental conditions;
$\frac{1}{2}$ considering that the signal goes and comes back.
So, to guarantee an adequate sensor collected data is needed to pay attention on the environment conditions because variances in humidity or temperature could change the accuracy of the sensor. Furthermore, if multiple ultrasonic sensors are used in close proximity to each other, the accuracy of the sensor may be affected by the interference between the different signals emitted by each ultrasonic sensor. In the market there are also high-tech ultrasonic sensor with an only one transducer able to switch from sound emitters to sound receivers and viceversa.

Figure 60: Ultrasonic sensor

The sensor presents four terminal pins: PWR, GND, Trig (trigger), and Ech (echo). The measurement is performed through a change of PULSE signals among the MKR1000 and the sensor. The microcontroller send a pulse with a width of a couple of microseconds to the sensor via the Trig pin. From that moment, the sensor activates the Ech pin setting it to HIGHT, in this configuration-set sound burst is emitted, flying towards an object, and bouncing back. The Ech pin is kept on activate mode until the front of the sound burst is echoed back into the receiver transducer. The microcontroller can obtain the time of flight by measuring the difference between the time when the Echo pin receives a rising edge, so the time when the Echo pin is in active state, and the time when Echo pin receives the first opposite activating edge.
In this preliminary ballbot project the ultrasonic sensor it thinking to be used as obstacle avoidance systems.

11.3.2 Hall sensor module

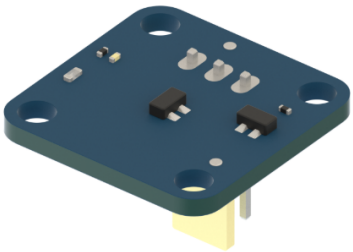

Figure 61: Hall Sensor

A hall sensor is a linear transducer that works on the principle of the Hall effect varying its output voltage depending on the magnetic field density surrounding the device. Usually adopted to time speed of wheels or shaft, such as a tachometer, it has many applications including electronic switch, and current sensing applications. The module in *Figure51* is the SL353HT Hall Sensor from Honeywell compatible with

Arduino MKR1000: this particular device is characterized by high switching speed and low current consumption. Arduino micro controller is able to detect the output voltage change of the Hall sensor through its interrupt pin and determine when the magnet is near the sensor or not. Hall effect sensor has three terminal pins: V_{cc}, GND , and V_{out} (voltage output signal)

11.3.3 Inertial Measurement Unit (IMU)

An IMU (Inertial Measurement Unit) is an electronic device that measures module behaviour under changes in linear acceleration, angular rotation, and, in some cases, the magnetic field around the module. IMU measurements are possible because it is a self-contained system that includes a triad of different sensors, accelerometers, gyroscopes, and magnetometers. In fact, the data not jet processed coming from the sensors on the IMU, are then elaborate and combined into other information that is easier to use in our projects. The measurements extract are indicated as pitch, roll, and yaw and respectively correlates to X, Y, or Z axes.

- Acceletometer:
  The accelerometer measures the linear velocity and the changes in the object to which it is attached. The accelerometer measures the linear velocity and the changes in the object to which it is attached. The sensor generate diefferent amounts of voltage depending on stress scenario or intese vibration due to the movement and it provides motion data as an x, y, and z value set.

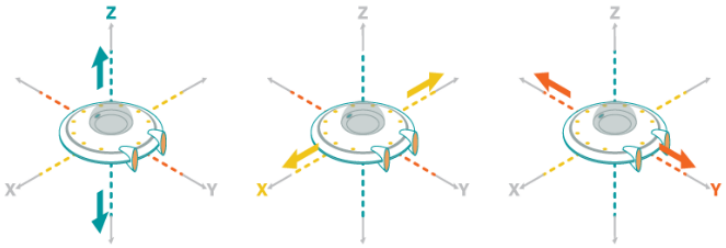

Figure 62: Accelerometer linear velocity measurements

- Gyroscope:
  The gyroscope catches orientation with respect to the gravity. It provides rotation data using x, y, and z values used to measure rotation rate (angular velocity).

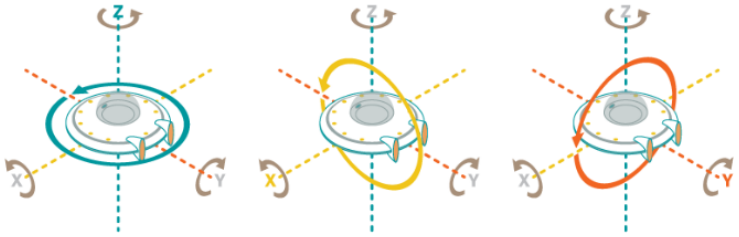

Figure 63: Gyroscope angular velocity measurements

- Magnetometer:
  Despite its serviceability, the magnetometer is not present in all IMU shield. This sensor measures the magnetism so it is generally used from the IMU to detect the relative change in a magnetic field at a particular direction. Magnetometers are used to provide a direction reference for our electronic device. It gives a North Magnetic Pole reference point and improves the accuracy of the measurements.

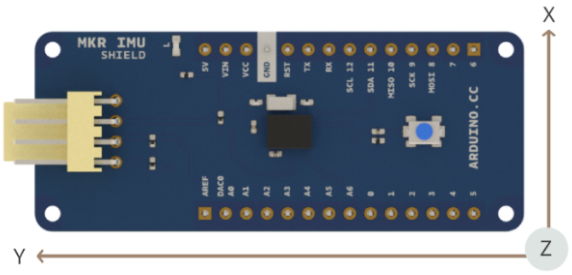

Figure 64: BNO055 IMU shield

In *Figure54* is shown BNO055 IMU from Bosch, a 9-axis (accelerometer and gyroroscope and magnetometer) orientation sensor, compatible with Arduino MKR1000: it is needed the manual calibration every time is powered on to calibrate magnometer inner sensor. The communication of the MKR100 with the sensor happens through the BNO055 IMU library: it is possible download and install it using the `library manager` on the `Arduino IDE (Integrated Development Environment)` → `Sketch` → `Include Library` → `Manage Library`.

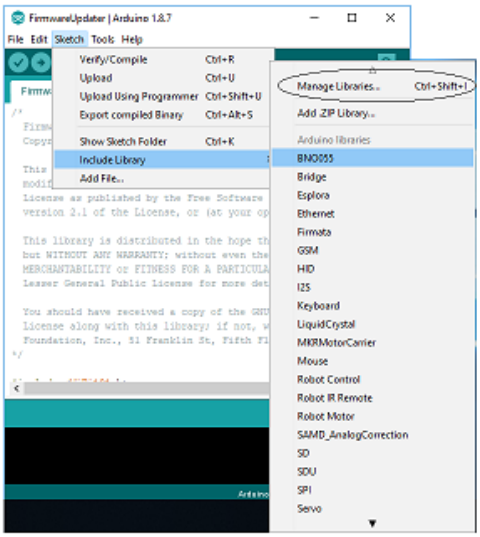

Figure 65: Guidelines IMU BNO055-library download

In this preliminary study of Ballbot robot, IMU shield is used for tracking the orientation of the ballbot.

11.4 Aduino MKR1000 - Matlab Interface

To program the Arduino MKR1000 through MATLAB and Simulink are needed two support interface packagese: `MATLAB Support Package for Arduino` and `SimulinkSupport Package for Arduino` . These packages make MATLAB able to communicate with Arduino directly and extend Simulink library with characteristic blocks to configure sensors, writing and reading data from them.

- `MATLAB Support Package for Arduino` :
  This package allows MATLAB to acquire inputs and send outputs to Arduino boards and connected devices.
  It also installs `Arduino IDE 1.8.1 (Arduino Integrated Development Environment)` and a few associated libraries.
- `Simulink Support Package for Arduino` :
  Thanks this package is possible to run Simulink models on Arduino boards;

The two packages can be easily downloaded as `Adds-Ons` from the `MATLAB Toolstrip` clicking on `Get Add-Ons`: it is also important to follow the instructions on the

`Adds-Ons` themselves.

12 Arduino MKR100-Simulink interface

In this section a first modeling of the ballbot control system is developed paying attention on the designed operating electronics. A basic interface between Arduino MKR100 and Simulink tool is allowed by a system of sensors and actuators accessible through the interface libraries discussed in the previous chapter.
This control system must be understood as a guideline on a first approach to the development of a control that can be downloaded on the chosen Arduino shield so, it is yet needed to test and modify it accordingly, adding also the safety conditions that the test result highlights.

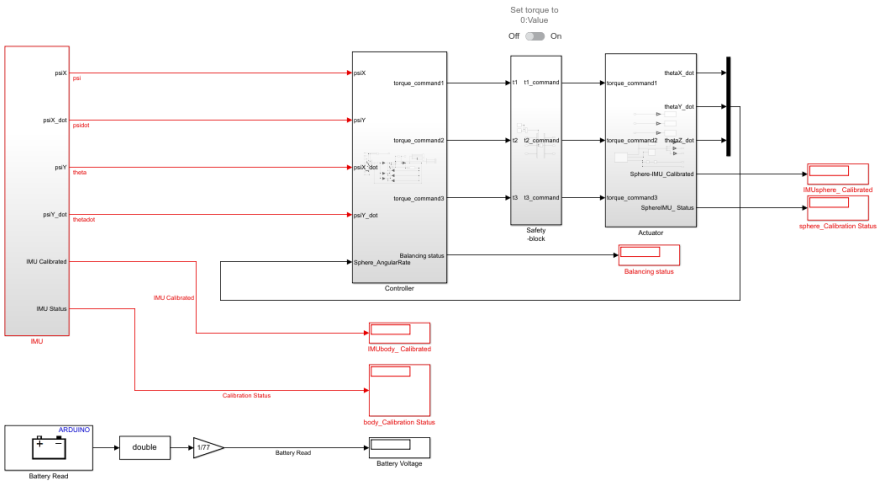

Figure 66: Simulink system control-block with Arduino inteface

12.1 IMU BNO055 block

In the figure below it is shown the enlarged IMU interface block.

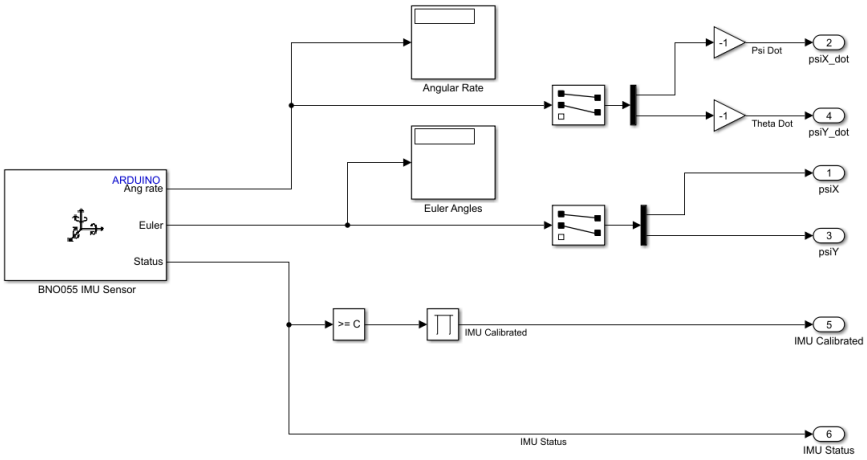

Figure 67: Arduino IMU block

`BNO055 IMU Sensor block` is located in *Simulink Library Browser* → *Simulink Support for Arduino Sensors library*. It is added in the Simulink project with the setting of *Figure59* in order to get the desired output signals : angular rate [degree per second] and Euler angles [degree].
The output number 6, *IMU status*, gives information about the calibration status respectively of IMU system, Gyroscope, Accelerometer and Magnetometer through a vector shown on "Calibration Status" display. A basic requirement to obtain accurate outputs is to initialize each sensor for locating the gravity vector and magnetic field and offset the coordinates in an apposite way. The elements of the calibration status vector consist of a values included between 0 and 3, evaluating the degree to which the sensor is calibrated. In our case it is fundamental to have at least a fully calibrated (3) gyroscope and magnetometer, a way to obtain this scenario is to leave the body which support the IMU shield completely at rest, while the model runs in External mode to calibrate the gyroscope, and to pick up the body and rotate it at least 90 degrees along each of the 3 spatial axes to obtain a calibration of the magnetometer: repeat this procedure until the respective vector elements does not show a fully calibrate status. The calibration process must be done every time the BNO055 IMU is powered on.
This model is configured to be executed in `External mode` (It is needed to set *Simulation mode* to `External` in the Simulink toolbar) so, it will run directly on the Arduino board but, thanks to the Display blocks "Angular Rate" and "Euler Angles", it is still possible to send information back to Simulink and visualize it on the screen.

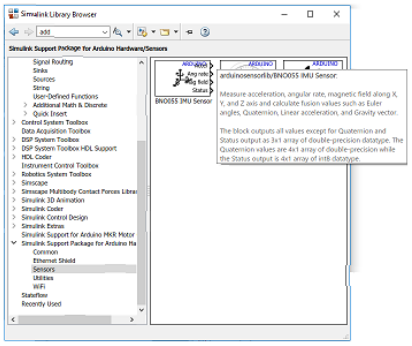

Figure 68: Simulink IMU block

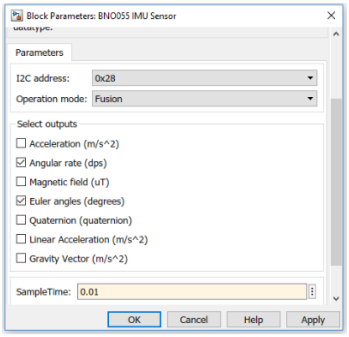

Figure 69: Simulink IMU setting

12.2 Safety Block

The safety block contains a basic logic function implemented with an AND logic block for giving the possibility to manually set the torque control signals on 0, in the event of anomalous system behavior during the hardware simulation. It is easily to enable 0 AND-input value directly through the slider switch labeled "Set torque on 0" *Figure61*.

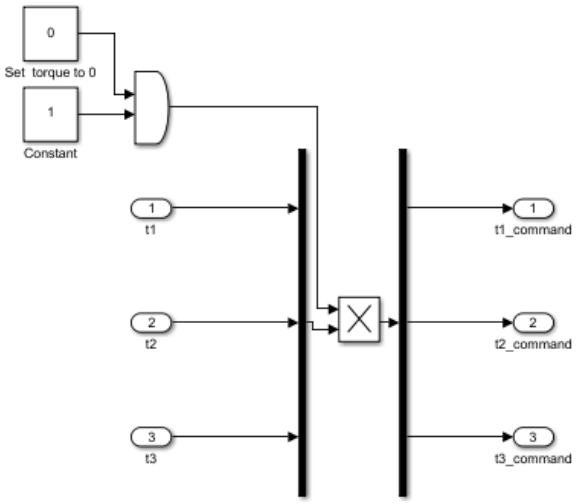

Figure 70: Simulink-Arduino safety-block

12.3 Actuator block

Actuator block it used to command torque values to the omnidirectional wheels motors. In this way it is needed normalized the torque commands generated by the control block in terms of signed fractional duty cycle (-1 to 1).

The interface with the conceived DCmotor is implemented through M3M4DCMotors block *Figure64*. As indicated in its setting window, this block admit a value included between -255 and 255, the block translate its input value both in voltage to be applied across the motor and the direction in which it has to be applied so, consider that a positive torque to the motor it is applied if the value is positive and a negative torque viceversa. Since the torque commands are values normalized between -1 and 1 , a gain value to 255 is added in order to generate a proper input format for the DC Motor block.

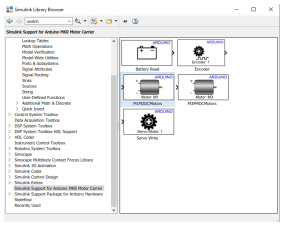

Figure 71: Simulink DC-Motor block

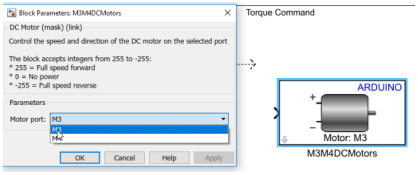

Figure 72: Simulink DC-Motor setting

To detect the ball speed it is conceived to add another IMU shield on the center of the sphere with an apposite support mechanism. The communication between this second IMU and the Arduino could be happen through a WIfi communication thanks the Wi-fi integrated chip on the Arduino MKR1000.

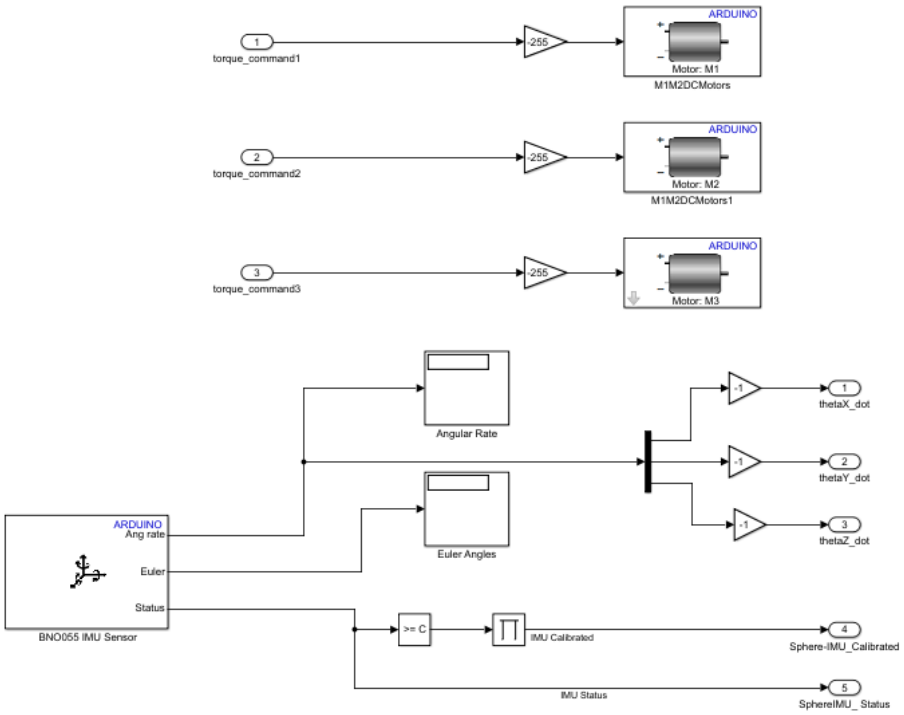

Figure 73: Simulink-Arduino actuator block

13 Conclusion

This thesis aim is to develop considerations on the design and control of a Ballbot system: a modular and spherical based robot with a structure that makes it suitable for interacting with the human environment.
During the design phase, the system's mechanism of operation and its principle of motion were carefully examined. A Ballbot with a ball drive-mechanism composed of three omnidirectional wheels was chosen. To understand the Ballbot dynamic, the physical model was analysed starting with simplified two-dimensional model of the structure under suitable assumptions. The structure has been considered as a reverse pendulum and a modeling has been carried out on the two XZ YZ planes.The mathematical modelling is completed by introducing a linearized three-dimensional model.
Analysing the 2D control results, the project has generated very satisfying results. As it can be perceived in the *chapter7*, the plots obtained from the simulations of the LQR control show how effectively the model is implemented. In fact, although an Observer was used as a system for estimating the states, there are no significant differences with respect to the direct measurements coming out from the plant. Working on the respective weights of the matrices Q and R, the system control implemented results are robust and stable: both on the position control and on the speed control, the output measurement can track the chosen reference and give a stable system state variable according to the dynamics of the system.
In the three-dimensional case, instead, the system must work with a plant state vector of ten elements. Therefore, finding the compromise and the appropriate weight for each variable results in a more complicated scenario than the two-dimensional case. In any case, the 3D controller developed as a control system on the position is able to generate a measured output able to track the assigned reference but needs to continue to work carefully on the weights of the Q and R matrices in order to obtain a three-dimensional control system that also reflects the dynamics of the analysed system.
The future goal, that will succeed the one carried out by this research, will be to obtain a physical-stable 3D controller in order to be able to interface it with the Simscape Ballbot scheme developed, to simulate then the physical behaviour of the Ballbot modeling.

14 Appendix

14.1 XZ Plane Math

$$T_1 = \frac{1}{2}M_s(\dot{x}_s^2 + \dot{z}_s^2) + \frac{1}{2}M_b(\dot{x}_b^2 + \dot{z}_b^2) =$$

$$= \frac{1}{2}M_s(R_s\dot{\theta})^2 + \frac{1}{2}M_s((R_s\dot{\theta} + L\dot{\alpha}cos\alpha)^2 + (-L\dot{\alpha}sen\alpha)^2)$$

$$= \frac{1}{2}M_sR_s^2\dot{\theta}^2 + \frac{1}{2}M_BR_s^2\dot{\theta}^2 + \frac{1}{2}M_BL^2\dot{\alpha}^2 + M_BR_sL\dot{\theta}\dot{\alpha}cos(\alpha)$$

$$= \frac{1}{2}M_sR_s^2\dot{\theta}^2 + \frac{1}{2}M_BR_s^2\dot{\theta}^2 + \frac{1}{2}M_BL^2\dot{\alpha}^2cos(\alpha)^2 + M_BR_sL\dot{\theta}\dot{\alpha}c_\alpha + \frac{1}{2}M_BL^2\dot{\alpha}^2sin(\alpha)^2 =$$

$$= \frac{1}{2}M_sR_s^2\dot{\theta}^2 + \frac{1}{2}M_bR_s^2\dot{\theta}^2 + \frac{1}{2}M_bL^2\dot{\alpha}^2 + M_bR_sL\dot{\theta}\dot{\alpha}cos(\alpha);$$

$$T_1 = \frac{1}{2}M_s(\dot{x}_s^2 + \dot{z}_s^2) + \frac{1}{2}M_b(\dot{x}_b^2 + \dot{z}_b^2) =$$

$$= \frac{1}{2}M_s(R_s\dot{\theta})^2 + \frac{1}{2}M_s((R_s\dot{\theta} + L\dot{\alpha}cos\alpha)^2 + (-L\dot{\alpha}sen\alpha)^2) =$$

$$= \frac{1}{2}M_sR_s^2\dot{\theta}^2 + \frac{1}{2}M_BR_s^2\dot{\theta}^2 + \frac{1}{2}M_BL^2\dot{\alpha}^2 + M_BR_sL\dot{\theta}\dot{\alpha}cos(\alpha) =$$

$$= \frac{1}{2}M_sR_s^2\dot{\theta}^2 + \frac{1}{2}M_BR_s^2\dot{\theta}^2 + \frac{1}{2}M_BL^2\dot{\alpha}^2cos(\alpha)^2 + M_BR_sL\dot{\theta}\dot{\alpha}c_\alpha + \frac{1}{2}M_BL^2\dot{\alpha}^2sin(\alpha)^2 =$$

$$= \frac{1}{2}M_sR_s^2\dot{\theta}^2 + \frac{1}{2}M_bR_s^2\dot{\theta}^2 + \frac{1}{2}M_bL^2\dot{\alpha}^2 + M_bR_sL\dot{\theta}\dot{\alpha}cos(\alpha);$$

$$T_2 = \frac{1}{2}J_s\dot{\theta}^2 + \frac{1}{2}J_\alpha\dot{\alpha}^2;$$

$$U = M_sgz_s + M_Bgz_B; U = M_sgR + M_Bg(R + Lcos(\alpha));$$

$$L = \frac{1}{2}R_s^2\dot{\theta}^2(M_s+M_B)+\frac{1}{2}M_bL^2\dot{\alpha}^2+M_bR_sL\dot{\theta}\dot{\alpha}cos(\alpha)++\frac{1}{2}J_s\dot{\theta}^2+\frac{1}{2}J_\alpha\dot{\alpha}^2-M_sgR-M_Bg(R+Lcos(\alpha));$$

$$\begin{cases} \frac{d}{dt}\frac{\delta L}{\delta \dot{\theta}} - \frac{\delta L}{\delta \theta} = F_\theta \\ \frac{d}{dt}\frac{\delta L}{\delta \dot{\alpha}} - \frac{\delta L}{\delta \alpha} = F_\alpha \end{cases} \tag{64}$$

$$\frac{\delta L}{\delta \dot{\theta}} = R_s^2\dot{\theta}(M_s + M_b) + M_b R_s L\dot{\alpha}cos(\alpha) + J_s\dot{\theta}; \tag{65}$$

$$\frac{d}{dt}\frac{\delta L}{\delta \dot{\theta}} = R_s^2(M_s + M_b)\ddot{\theta} + M_b R_s L\ddot{\alpha}cos(\alpha) - M_B R_s L\dot{\alpha}^2 sin(\alpha) + J_s\ddot{\theta}; \tag{66}$$

$$\frac{\delta L}{\delta \theta} = 0; \tag{67}$$

$$\frac{\delta L}{\delta \dot{\alpha}} = M_b L^2\dot{\alpha} + M_b R_s L\dot{\theta}cos(\alpha) + J_\alpha\dot{\alpha}; \tag{68}$$

$$\frac{d}{dt}\frac{\delta L}{\delta \dot{\alpha}} = M_b L^2\ddot{\alpha} + M_b R_s L\ddot{\theta}cos(\alpha) - M_b R_s L\dot{\theta}\dot{\alpha}sin(\alpha) + J_\alpha\ddot{\alpha}; \tag{69}$$

$$\frac{\delta L}{\delta \dot{\alpha}} = M_b gLsin(\alpha) - M_b R_s L\dot{\theta}\dot{\alpha}sin(\alpha); \tag{70}$$

$$\begin{cases} R_s^2(M_s + M_b)\ddot{\theta} + M_b R_s Lcos(\alpha)\ddot{\alpha} + J_s\ddot{\theta} - M_b R_s Lsin(\alpha)\dot{\alpha}^2 = F_\theta \\ M_b R_s Lcos(\alpha)\ddot{\theta} + M_b L^2\ddot{\alpha} + J_\alpha\ddot{\alpha} - M_b R_s L\dot{\theta}\dot{\alpha}sin(\alpha) - M_b gLsin(\alpha) + M_b R_s L\dot{\theta}\dot{\alpha}sin(\alpha) = F_\alpha \end{cases} \tag{71}$$

$$\begin{cases} (R_s^2(M_s + M_b) + J_s)\ddot{\theta} + M_b R_s Lcos(\alpha)\ddot{\alpha} - M_b R_s Lsin(\alpha)\dot{\alpha}^2 = F_\theta \\ M_b R_s Lcos(\alpha)\ddot{\theta} + (M_b L^2 + J_\alpha)\ddot{\alpha} - M_b R_s L\dot{\theta}\dot{\alpha}sin(\alpha) - M_b gLsin(\alpha) + M_b R_s L\dot{\theta}\dot{\alpha}sin(\alpha) = F_\alpha \end{cases} \tag{72}$$

$$\begin{cases} [(R_s^2(M_s + M_b) + J_s)]\ddot{\theta} + [M_B R_s Lcos(\alpha)]\ddot{\alpha} - M_b R_s Lsin(\alpha)\dot{\alpha}^2 = F_\theta \\ [M_B R_s Lcos(\alpha)]\ddot{\theta} + [M_b L^2 + J_\alpha]\ddot{\alpha} - M_b gLsin(\alpha) = F_\alpha \end{cases} \tag{73}$$

Linearized state equations of 2D-spherical wheeled inverted pendulum on YZ plane:

$$\begin{cases} [(M_s + M_b)R_s^2 + J_s]\ddot{\theta} + [M_b R_s L]\ddot{\alpha} = F_\theta \\ [M_b L R_s]\ddot{\theta} + [M_b L^2 + J_\alpha]\ddot{\alpha} - M_b gL\alpha = F_\alpha \end{cases} \tag{74}$$

Inverse Inertia matrix:

$$M^{-1} = \frac{1}{detM}\begin{bmatrix} M_{(2,2)} & M_{(1,2)} \\ M_{(2,1)} & M_{(1,1)} \end{bmatrix} \tag{75}$$

14.2 YZ-Plane Math

$$T_1 = \frac{1}{2}M_s(\dot{y}_s^2 + \dot{z}_s^2) + \frac{1}{2}M_b(\dot{y}_b^2 + \dot{z}_b^2) = \tag{76}$$

$$= \frac{1}{2}M_s R_s^2\dot{\psi}^2 + \frac{1}{2}M_b R_s^2\dot{\psi}^2 + \frac{bL^2\dot{\beta}^2}{2} + M_b R_s L\dot{\psi}\dot{\beta}cos(\beta); \tag{77}$$

$$T_2 = \frac{1}{2}J_s\dot{\psi}^2 + \frac{1}{2}J_\beta\beta^2; \tag{78}$$

$$U = M_s g z_s + M_B g z_b = \quad M_s gR + M_b g(R + Lcos(\beta)); \tag{79}$$

$$\frac{d}{dt}\frac{\delta L}{\delta\dot{\theta}} - \frac{\delta L}{\delta\theta} = F_\theta; \tag{80}$$

$$\frac{d}{dt}\frac{\delta L}{\delta\dot{\alpha}} - \frac{\delta L}{\delta\alpha} = F_\alpha; \tag{81}$$

$$L = \frac{1}{2}R_s^2\dot{\psi}^2(M_s+M_B)+\frac{1}{2}M_bL^2\dot{\beta}^2+M_bR_sL\dot{\psi}\dot{\beta}cos(\beta)++\frac{1}{2}J_s\dot{\psi}^2+\frac{1}{2}J_\beta\dot{\beta}^2-M_sgR-M_bg(R+Lcos(\beta)); \tag{82}$$

$$\begin{cases} \frac{d}{dt}\frac{\delta L}{\delta\dot{\psi}} - \frac{\psi L}{\delta\psi} = F_\psi \\ \frac{d}{dt}\frac{\delta L}{\delta\dot{\beta}} - \frac{\delta L}{\delta\beta} = F_\beta \end{cases} \tag{83}$$

$$\begin{cases} R_s^2(M_s + M_b)\ddot{\psi} + M_bR_sLcos(\beta)\ddot{\beta} + J_s\ddot{\beta} - M_bR_sLsin(\beta)\dot{\beta}^2 = F_\psi \\ M_bR_sLcos(\beta)\ddot{\psi} + M_bL^2\ddot{\beta} + J_\beta\ddot{\beta} - M_bR_sL\dot{\psi}\dot{\beta}sin(\beta) - M_bgLsin(\beta) + M_bR_sL\dot{\psi}\dot{\beta}sin(\beta) = F_\beta \end{cases} \tag{84}$$

$$\begin{cases} (R_s^2(M_s + M_b) + J_s)\ddot{\psi} + M_bR_sLcos(\alpha)\ddot{\beta} - M_bR_sLsin(\beta)\dot{\beta}^2 = F_\psi \\ M_bR_sLcos(\beta)\ddot{\psi} + (M_bL^2 + J_\beta)\ddot{\beta} - M_bR_sL\dot{\psi}\dot{\beta}sin(\beta) - M_bgLsin(\beta) + M_bR_sL\dot{\psi}\dot{\beta}sin(\beta) = F_\beta \end{cases} \tag{85}$$

$$\begin{cases} [(R_s^2(M_s + M_b) + J_s)]\ddot{\psi} + [M_BR_sLcos(\beta)]\ddot{\beta} - M_bR_sLsin(\beta)\dot{\beta}^2 = F_\psi \\ [M_BR_sLcos(\beta)]\ddot{\psi} + [M_bL^2 + J_\beta]\ddot{\beta} - M_bgLsin(\beta) = F_\beta \end{cases} \tag{86}$$

Linearized state equations of 2D-spherical wheeled inverted pendulum on XZ-plane:

$$\begin{cases} [(M_s + M_b)R_s^2 + J_s]\ddot{\psi} + [M_bR_sL]\ddot{\beta} = F_\psi \\ [M_bLR_s]\ddot{\psi} + [M_bL^2 + J_\beta\ddot{\beta} - M_bgL\beta = F_\beta \end{cases} \tag{87}$$

14.3 Matlab code

controller2D-Theta.m

```
%Geometric Parameters
L=0.6;              %[m]
Rs=0.16;            %[m]
Ms=2.5;             %[Kg]
Js=0.016;           %[kg*m^2]
Ja=12.48;           %[kg*m^2]
Mb=50;              %[kg]
g=9.81;             %[m/s^2]

M_11=[(Mb+Ms)*Rs^2]+Js;
M_12=Mb*L*Rs;
M_21=M_12;
M_22=(Mb*L^2)+Ja;

detM=(M_11*M_22)-(M_21*M_12);

%State Matrix coefficients S1 S2
A_32=(Mb*g*L*M_12)/detM;
A_42=(Mb*g*L*M_11)/detM;

A=[0,0,1,0; 0,0,0,1; 0,A_32,0,0; 0,A_42,0,0];
B=[0;0;(M_22+M_12)/detM;(M_21+M_11)/detM];
C=eye(4);
D=zeros(4,1);

%System Reachability
Mr=ctrb(A,B);                   %create Reachability Matrix;
rho_Mr=rank(Mr);                %compute Mr rank;

%System Observability
Mo=obsv(A,C);                   %create Observability Matrix;
rho_Mo=rank(Mo);                %compute Mo rank;

sys_c=ss(A,B,C,D);

[NUM,DEN]=ss2tf(A,B,C,D);
tf_sys1=tf(NUM(1,1:5),DEN);
tf_sys2=tf(NUM(2,1:5),DEN);
tf_sys3=tf(NUM(3,1:4),DEN);
tf_sys4=tf(NUM(4,1:4),DEN);

Ts=0.001;
```

```
sys_d=c2d(sys_c,Ts,'zoh');

A_d= sys_d.A;
B_d=sys_d.B;

C_d=[1 0 0 0];
Cd_alpha=[0 0 1 0];

D_d=0;

A_tot=[1 -Ts*C_d ;zeros(4,1) A_d];
B_tot=[0; B_d];

%LQR Design Parameters: weight matrices Q and R
R=0.1
Q=diag([1000 10 100 100 100]);
[K_dlqr]=dlqr(A_tot,B_tot,Q,R)

lambda_c = [0.995 0.995 0.995 0.995 0.995];
[K_dlqr1]=acker(A_tot,B_tot,lambda_c)
lambda_obsv_des=[0.5 0.5 0.5 0.5];
L=acker(A_d',C_d',lambda_obsv_des)';

sys_obs=ss(A_d-L*C_d, [B_d L], eye(4),0,Ts);
```

controller2D-ThetaAlpha.m

```
%Geometric Parameters
L=0.6;              %[m]
Rs=0.16;            %[m]
Ms=2.5;             %[Kg]
Js=0.016;           %[kg*m^2]
Ja=12.48;           %[kg*m^2]
Mb=50;              %[kg]
g=9.81;             %[m/s^2]

M_11=[(Mb+Ms)*Rs^2]+Js;
M_12=Mb*L*Rs;
M_21=M_12;
M_22=(Mb*L^2)+Ja;

detM=(M_11*M_22)-(M_21*M_12);

%State Matrix coefficients S1 S2
A_32=(Mb*g*L*M_12)/detM;
A_42=(Mb*g*L*M_11)/detM;
```

```
20
   A=[0,0,1,0;
22      0,0,0,1;
        0,A_32,0,0;
24      0,A_42,0,0];
   B=[0;0;(M_22+M_12)/detM;(M_21+M_11)/detM];
26 C=eye(4);
   D=zeros(4,1);
28
   %System Reachability
30 Mr=ctrb(A,B);                    %create Reachability Matrix;
   rho_Mr=rank(Mr);                 %compute Mr rank;
32
   %System Observability
34 Mo=obsv(A,C);                    %create Observability Matrix;
   rho_Mo=rank(Mo);                 %compute Mo rank;
36
   sys_c=ss(A,B,C,D);
38

40 [NUM,DEN]=ss2tf(A,B,C,D);
   tf_sys1=tf(NUM(1,1:5),DEN);
42 tf_sys2=tf(NUM(2,1:5),DEN);
   tf_sys3=tf(NUM(3,1:4),DEN);
44 tf_sys4=tf(NUM(4,1:4),DEN);

46 x0=[pi;pi;0;0];

48 Ts=0.001;
   sys_d=c2d(sys_c,Ts,'zoh');
50
   A_d= sys_d.A;
52 B_d=sys_d.B;
   C_d=[1 0 0 0; 0 1 0 0];
54 D_d=0;

56 A_tot=[eye(2) -Ts*C_d ; zeros(4,2) A_d];
   B_tot=[0;0; B_d];
58 C_tot=[zeros(2,2) C_d];
   D_tot=0;
60
   %LQR Design Parameters: weight matrices Q and R
62  R=0.1;
    Q1=diag([1000 1000 10 100 100 100]);
64 [K_dlqr]=dlqr(A_tot,B_tot,Q1,R)

66 %Observer system:
```

```
lambda_obsv_des=[0.5 0.45 0.55 0.5];
L=place(A_d',C_d',lambda_obsv_des)';

sys_obs=ss(A_d-L*C_d, [B_d L], eye(4) ,0,Ts);
```

controller3D.m

```
%Linearized model on zero equilibrium point
A=zeros(10,10);
A(1,:)=[0,1,zeros(1,8)];
A(2,:)=[37.69, zeros(1,9)];
A(3,:)=[zeros(1,3) 1 zeros(1,6)];
A(4,:)=[0 0 37.73 zeros(1,7)];
A(5,:)=[zeros(1,5) 1 zeros(1,4)];
A(6,:)=[zeros(1,10)];
A(7,:)=[zeros(1,7) 1 zeros(1,2)];
A(8,:)=[-73.02 zeros(1,9)];
A(9,:)=[zeros(1,9) 1];
A(10,:)=[0 0 -73.09 zeros(1,7)];

B=zeros(10,3);
B(1,:)=[0 0 0];
B(2,:)=[4.02 -2.01 -2.01];
B(3,:)=[0 0 0];
B(4,:)=[0 3.485 -3.485];
B(5,:)=[0 0 0];
B(6,:)=[-10.76 -10.76 -10.76];
B(7,:)=[0 0 0];
B(8,:)=[-13.48 6.738 6.738];
B(9,:)=[0 0 0];
B(10,:)=[0 -11.68 -11.68];


C=eye(10);
D=zeros(10,3);

%System Reachability
Mr=ctrb(A,B);                    %create Reachability Matrix;
rho_Mr=rank(Mr);                 %compute Mr rank;

%System Observability
Mo=obsv(A,C);                    %create Observability Matrix;
rho_Mo=rank(Mo);                 %compute Mo rank;

%Continuous-time system
sys_c=ss(A,B,C,D);
```

```
40
   %Transfer functions 3D system:
42 [NUM,DEN]=ss2tf(A,B,C,D,:);
   tf_sys1=tf(NUM(1,:),DEN);
44 tf_sys2=tf(NUM(2,:),DEN);
   tf_sys3=tf(NUM(3,:),DEN);
46 tf_sys4=tf(NUM(4,:),DEN);
   tf_sys5=tf(NUM(5,:),DEN);
48 tf_sys6=tf(NUM(6,:),DEN);
   tf_sys7=tf(NUM(7,:),DEN);
50 tf_sys8=tf(NUM(8,:),DEN);
   tf_sys9=tf(NUM(9,:),DEN);
52 tf_sys10=tf(NUM(10,:),DEN);

54 %Sampling time
   Ts=0.001;
56
   %Discrete-time system
58 sys_d=c2d(sys_c,Ts,'zoh');
   %
60 A_d= sys_d.A;

62 B_d=sys_d.B;

64 C_thetaX=[1 zeros(1,9)];
   C_thetaXdot=[0 1 zeros(1,8)];
66 C_thetaZ=[0 0 0 0 1 zeros(1,5)];
   C_d=[1 zeros(1,9); 0 0 1 zeros(1,7);0 0 0 0 1 zeros(1,5);
68     zeros(1,6) 1 zeros(1,3); zeros(1,8) 1 0]; %Measures on system position.
   C_obsv=[0 1 zeros(1,8); zeros(1,4) 1 zeros(1,5);zeros(1,6) 1 0 0 0;zeros(1,7) 1 0 0; z
70
   D_d=0;
72
   A_tot=[1 0 -Ts*C_thetaZ ;1 1 zeros(1,10) ; zeros(10,2) A_d];
74 B_tot=[zeros(2,3); B_d];

76 %LQR Design Parameters: weight matrices Q and R:

78 %    Q,R weighted for C_d:
   %    R=diag([0.1 0.1 0.1]);
80 %    Q=diag([100 100 100 100 100 1 0.01 0 1 0.001 1 0.001 0.1 0.01 0.1]);

82 %    Q,R weighted for C_thetaX:
   %    R=10*diag([0.1 0.1 0.1]);
84 %    Q=diag([10 .02 0.1*ones(1,10)]);

86 %    Q,R weighted for C_thetaZ:
```

```
      R=10*diag([0.1 0.1 0.1]);
88    Q=diag([10 .02 1*ones(1,10)]);

90  %   Q,R weighted for C_thetaZ-Simscape. It is needed to work with weigth of
    %   Q and R
92  %   R=1e16*diag([0.1 0.1 0.1]);
    %   Q=diag([1000 1e6, 1e4 1e10 1e4 1e10 1000 1e10 1e6 1000 1e4 1e20])
94
      [K_dlqr]=dlqr(A_tot,B_tot,Q,R)
96
    %Observer:
98    lambda_obsv_des=[0.5 0.5 0.5 0.5 0.55 0.55 0.55 0.45 0.45 0.5];
      L=place(A_d',C_obsv',lambda_obsv_des)';
100
      sys_obs=ss(A_d-L*C_obsv,[B_d L], eye(10),0,Ts);
```

References

[1] Iulian Calciu, Laura Mihaela Vasilescu, Adriana Draghici,Daniel Rosner, Monica Patrascu. Facultu of Automatic Control and Computers University Politehina of Bucarest. Three Omni-wheel Ballbot Optimum implementation

[2] Su, B., Gong, Y. (2017). Euler-Lagrangian Modeling and Exact Trajectory Following Controlling of Ballbot-Like Robot, 2325–2330.

[3] Navabi, H., Sadeghnejad, S., Ramezani, S., Baltes, J. (2017). Position Control of the Single Spherical Wheel Mobile Robot by Using the Fuzzy Sliding Mode Controller, 2017.

[4] (Nagarajan, Kantor, Hollis, 2006)Nagarajan, U., Kantor, G., Hollis, R. (2006). The Ballbot: An Omnidirectional Balancing Mobile Robot.

[5] Nxt, L. M. (n.d.). NXT Ballbot Model-Based Design.

[6] G.Genta. Dynamics of Rotating Systems,Springer,2005.

[7] B.Bona. Dynamic Modelling of Mechatronic Systems, Celid.

[8] H. S.R. Kwakernaak. Linear Optimal Control Systems, Wiley-Interscience, 1972.

[9] M.S.R.H a G.K., Bhashar Vaidya. "Operation of the Ballbot on Slopes and with Center-of-Mass Offsets", IEEE International Conference on Robotics and Automation (ICRA),2015.

[10] J. U.S. C.B.FONG. 899: BALLBOT, University of Adelaide, 2009

[11] Ivanova, T. B., Kilin, A. A., Pivovarova, E. N. (2018). Controlled Motion of a Spherical Robot with Feedback . I, 497–510.

[12] A, A. H. J. (2002). Introducing August: A Novel Strategy for An Omnidirectional Spherical Rolling Robot, (May), 3527–3533.

[13] Negi, M., Johnson, J., Senthilnathan, R. (2018). ScienceDirect ScienceDirect A Fuzzy Fuzzy Logic-in-Loop Logic-in-Loop Control Control for for a a Novel Novel Reduced Reduced Height Height Ballbot Ballbot Prototype Prototype. Procedia Computer Science, 133, 960–967. https://doi.org/10.1016/j.procs.2018.07.081

[14] Bhattacharya, S. (2000). Design , Experiments and Motion Planning of a Spherical Rolling, (April), 1207–1212.

[15] Thesis, B. ETH . Modeling and Control of a Ballbot, 2010.

[16] Navabi, H., Sadeghnejad, S., Ramezani, S., Baltes, J. (2017). Position Control of the Single Spherical Wheel Mobile Robot by Using the Fuzzy Sliding Mode Controller, 2017.

[17] Li, Y., Yang, M., Sun, H., Liu, Z., Zhang, Y. (2018). with Flywheel , Pendulum , and Propeller, 485–501. https://doi.org/10.1007/s10846-017-0558-x