POLITECNICO DI TORINO

POLITECNICO DI TORINO

Master Degree course in Computer Engineering

Master Degree Thesis

Super-Twisting Sliding Mode Control and Observation for a mobile ground robot

Supervisors
Prof. Elisa Capello
Iris David Du Mutel
Enza Incoronata Trombetta

Candidate
Luca Orsini

Academic Year 2022-2023

Abstract

An Observer and a Controller based on the Sliding Mode theory have been developed for a mobile ground robot. After an introduction about the main control theory concepts, including classical control techniques such as Proportional Integral Derivative (PID), additional considerations about the utility and use of observers in practical cases are provided. The theory of sliding mode is presented and analyzed in terms of performance, stability and convergence. The mathematical stability of the control and observation algorithm is studied making use of Lyapunov theory. The Observer and Controller design is based on the kinematic model of the unicycle. Matlab and Simulink have been used as simulation environment in which a pre-existing vehicle model is present. An initial tuning in simulation has been performed for both the navigation and control blocks. For the experimental case, a practical trial-and-error phase has been addressed to assess performance issues, such as sensor noise, non-linearities, data sampling frequencies, making it possible to deal with such limitations. An overview of the nodes and topics already on the robot board has been addressed, including a description of the Extended Kalman Filter (EKF). The filter performance is compared with the new sliding mode super-twsting observer. The observer estimates the body velocity and orientation angle in an inertial reference frame using as inputs acceleration and angular velocity, from IMU sensors, and inertial position from encoders. The controller takes as input the inertial orientation and body velocity from the observer and a reference trajectory from a ROS node containing the Artificial Potential Field (APF) algorithm. The algorithm from simulation are translated to Python code to be deployed on the real Unmanned Ground Vehicle (UGV). The Observer is tested on the real robot, initially in an open loop configuration. Eventually, the Observer is implemented in feedback with the Controller. The influence of the nonlinear closed loop system on the robot dynamic is then discussed. The results of the thesis enhance the robustness of the Sliding Mode technique, showing optimal performances from both the Observer and Controller.

Contents

Chapter 1

Introduction

A mobile ground robot is a vehicle that operates on surfaces and can move in space to reach a target and accomplish a goal. Mobile robots have numerous applications, such as emergency rescue operations, manufacturing, logistics, agriculture, and exploration, among others [1]. The Mars rovers are an example of exploratory ground robots that allow remote vehicle control and perform in hostile environments for humans. Typically, a mobile robot is designed to move autonomously and must be capable of determining the actions needed to accomplish a task. For autonomy, a mobile robot needs batteries for energy, boards with CPUs, databases, and algorithms for intelligence, and sensors, such as cameras, to obtain knowledge about the surrounding environment. Perception and locomotion are two of the most important areas of robotics. Perception refers to the ability of a robot to acquire and interpret sensory information about its environment. This can include information from cameras, lidar, sonar, or other sensors that allow the robot to understand its surroundings. Locomotion refers to the ability of a robot to move from one place to another. From a mechanical point of view a mobile robot consists of a rigid body (chassis) with a Locomotion system. The Locomotion structure is influenced mainly by the environment in which the robot has to perform, it can be on ground, underwater or on air. This obviously influence the mechanical components needed. For a ground mobile robot generally a wheeled structure is preferred, but it could be a "legged" structure, like in humanoid robots. A robot can be mathematically modeled as a system, which is a non-trivial concept. A system can be described as a collection of interacting components that evolve in time in response to an input or perturbation, which is detected by sensors to produce an output. For instance, a car is an example of a system composed of various components that generate friction with air or ground, each of which has a physical weight, and these interactions impact the evolution of the car's velocity, which is considered as the output in this case. A control system is a system with additional components to control its behavior. In modern control theory, the first step in defining a control system is to obtain or construct a mathematical model that describes the evolution of variables called states in response to a perturbation. Building a model is not an easy task, as it involves defining a first-order differential equation. The definition of a mathematical model can be based on a physical knowledge of the system, and this is the case of the kinematic unicycle model that will be introduced in chapter 4, or it can be done by sampling a large number

of data to fit a model that generates a so called black box, a function that links inputs and outputs, this process is called System Identification. This second method may allow to achieve great performances, but its very hard to be applied to control algorithms, because the information about physical meaning of variables is then lost. Obviously the mathematical model has a fundamental importance, because it defines the degree of correlation between the real, "ideal", evolution of the system, and its approximation. In short words no matter how the model is defined, some informations will always be lost, it is an approximation of the real world, Nonlinearities and other sources of disturbances can be difficult to model with precision, which can lead to errors in control or estimation. Additionally, there may be unmodeled dynamics or uncertainties in the system that can also impact the accuracy of the model. Therefore, it's important to carefully consider the limitations of the model and use techniques such as robust control and estimation to handle uncertainties and disturbances. The control of a mobile ground robot must then consider the presence of this uncertainties, so robust control algorithms must be chosen such that this kind of non-linearities are compensated. A good candidate for this purpose is the Sliding Mode technique, a non-linear algorithm with a switching function, which is considered very robust to compensate modeling uncertainties, and is also computationally very efficient [2]. Considering the experimental robot used for this thesis, since not all the control variables are available from the sensors, one of the aim of this thesis is the design of an Observer to estimate the body velocity and orientation of the robot with the knowledge of the inertial position from encoders. For this purpose a second order sliding mode technique has been chosen, because the kinematic relation between position and velocities contains second order degree uncertainties, so in this case a simple first order sliding mode observer would not compensate the uncertainties effect. As explained here [3], a very good candidate to deal with this kind of problem is the super-twisting sliding mode algorithm [4]. Then a controller has been designed, and since the control variable, for a kinematic reason, has a relative degree of one, a super-twisting control algorithm has been chosen for this purpose, because it avoid the so called "chattering". The sliding mode theory will be explained in chapter 3. Instead others chapters contain:

- Chapter 2 describes the basics of Guidance Navigation and Control system. It introduces important mathematical tools and concepts such as stability, controllability and observability, very importatnt to explain the algorithms design.
- Chapter 4 is used to introduce the kinematic model of the unicycle, which is the physical approximation used for the models design.
- Chapters 5 is where the design of the two algorithms occurs.
- chapter 6, is the result section, it contains an explanation of the experimental robot characteristics, the simulated and real results of the algorithms.

Chapter 2

Guidance, Navigation and Control Systems

Guidance, Navigation and control systems (GNC) are critical modules generally related to aerospace and missile fields, that deals with the control movement of objects, and can also be extended to the case of a ground Robot. The three modules can be represented in a closed loop as in Fig. 2.1.

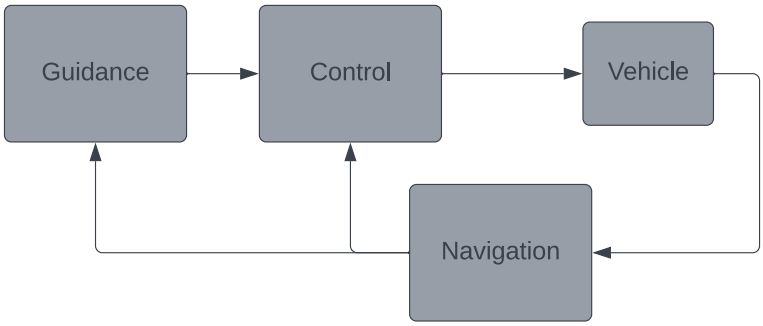

Figure 2.1. A closed loop representation of a GNC system

2.1 Guidance

The Guidance module is involved in the problem of defining a trajectory for the vehicle considering the required task goal, and the possible presence of obstacles. Is defined by algorithms that taking into account sensors data define desired reference points. A path planning algorithm, can work:

- Off-line: Requieres a structured environment, it means that a "map" of the surrounding enviroenment is known and so is the position of all the potential obstacles.
- On-line: An on-line path planning algorithm, instead, does not know the surrounding environment. It needs sensors to acquire informations of the space, and then

takes on-line decisions to reach a point. A very common example of an on-line guidance algorithm is the **Artificial Potential Field** [5].

2.1.1 Artificial Potential Fields

The idea is to build potential fields in the space, so that the robot is affected by two types of potentials:

- **Attractive potentials**
- **Repulsive potentials**

The **attractive potential** generates an attractive force to guide it to the goal position. Given $q = \begin{bmatrix} x \\ y \end{bmatrix}$ the inertial position of the robot, $q_g = \begin{bmatrix} x_g \\ y_g \end{bmatrix}$ the goal position and $e(q) = q_g - q$ an attractive potential can be defined as a:

- **Paraboloidal potential**:

$$U_a = \frac{1}{2} k_a e(q)^T e(q) = \frac{1}{2} k_a ||e(q)||^2$$

Then the attraction force is linear in e:

$$f_a = -\nabla U_a = k_a e(q)$$

- **Conical**:

$$U_a = k_a ||e(q)||$$

In this case the attractive force is constant:

$$f_a = k_a \frac{e(q)}{||e(q)||}$$

Obviously the first one works better in proximity of the goal, but increases linearly with e, so a good approach consist of an hybrid between the two. The number of **repulsive field** depend on the number of obstacles. The intensity of the repulsive force depend on distance from the obstacles, generally, to simplify, a range of influence is defined. A graphical representation of a APF scenario is showed on Fig. 2.2.

2.2 Navigation

A navigation system is a system of devices, technologies, and techniques that enable the determination of the location, orientation, and movement of an object. Navigation systems are commonly used in transportation, including aviation, maritime, and land-based vehicles, as well as in military, scientific, and recreational applications. These systems typically use a combination of sensors, such as GPS, accelerometers, gyroscopes, magnetometers, and cameras, to determine position, velocity, and direction, and provide guidance to reach a desired destination.

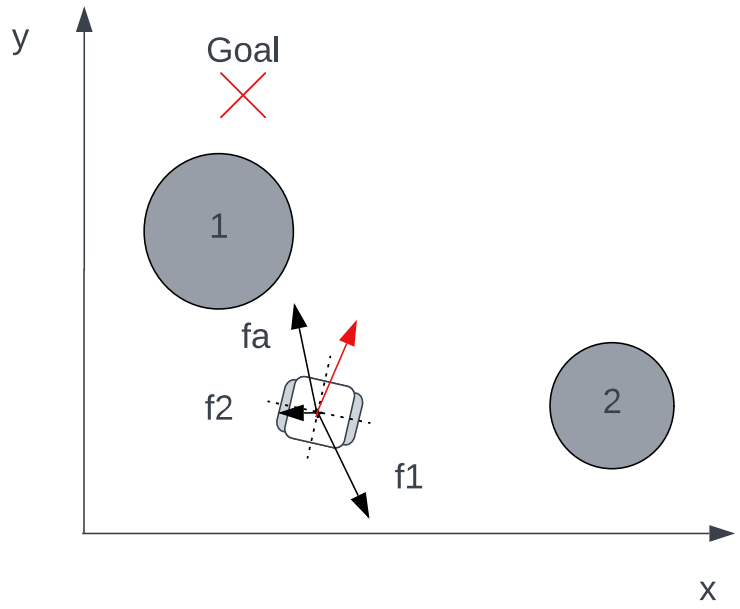

Figure 2.2. An example of the forces involved in an APF algorithm, with two obstacles and a mobile robot. The red line is the result force that generate a reference to get nearer to the goal and at the same point avoid obstacle.

2.3 Control systems main concepts

The main components that define a control system are the Controller, the Plant, the actuators and Sensors. A controller, roughly speaking, is a device that act to minimize the error between a variable and its desired reference. An example of a controller is the cruise control, which is a component that regulate the throttle position to control the air mass that enter in the manifold of the engine, to generate a torque and maintain a constant velocity of the vehicle [6]. In this case the error is defined by the actual velocity (variable) and constant velocity desired (reference). Another example is the Anti Blocking System (ABS), which aim is to avoid wheel blocking while braking both to minimize braking distance and to maintain steer ability to avoid obstacles, it is based on the measure of wheel velocity. The plant is the physical system to be controlled, while the actuators are the devices that allow a physical interaction with the plant, for example the car pedal. There are two main control system structures, the open loop control system and the closed loop system, also called feedback loop. The open loop structure is more simply, has a lower cost, it does not need measurements, to put it simply it is useful for example in manufacturing processes where output changes are not required, like a conveyor belt. The closed loop control system 2.3, instead, need sensors measurements to constantly define the error between the output and desired reference values. Moreover the feedback loop influences the dynamic of the system, so a stability study in this case is needed. The Plant can be represented with a mathematical model of differential equations, a very simple representation is the Linear Time Invariant (LTI) state space model:

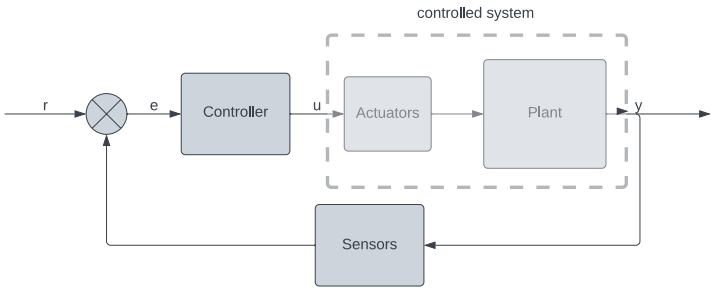

Figure 2.3. Closed loop control system

$$\begin{cases} \dot{x} = Ax + Bu \\ y = \mathrm{C}x \end{cases} \tag{2.1}$$

It can be read as, the variation in time of the states x linearly depend on the relations defined in matrix A, and by the inputs relations in B. Instead the matrix C defines the outputs (y), in short words, what is known from the sensors.
To clarify, from the LTI model is possible to make a very simple and intuitive control example:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = u_1 \\ \dot{x}_3 = x_4 \\ \dot{x}_4 = u_2 \end{cases} \tag{2.2}$$

It can be seen as the kinematic relation of a point in the 2D space, where x_2 is the velocity among the x-axis and x_4 is the velocity among the y-axis, while x_1 is the inertial position (x) and x_3 the y coordinate. The control could aim to drive the point from an initial inertial position (x_0, y_0) to the origin (0,0), the control variables are u_1 and u_2, that can represent accelerations on the axis. In this case the state space matrix are defined as:

$$\mathrm{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad \mathrm{B} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \qquad \mathrm{C} = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$$

The full rank C matrix tells that in this simple case all states are measured. An easy control law can be defined as:

$$\begin{cases} u_1 = -k_1 \cdot x_1 - k_2 \cdot x_2 \\ u_2 = -k_3 \cdot x_3 - k_4 \cdot x_4 \end{cases} \tag{2.3}$$

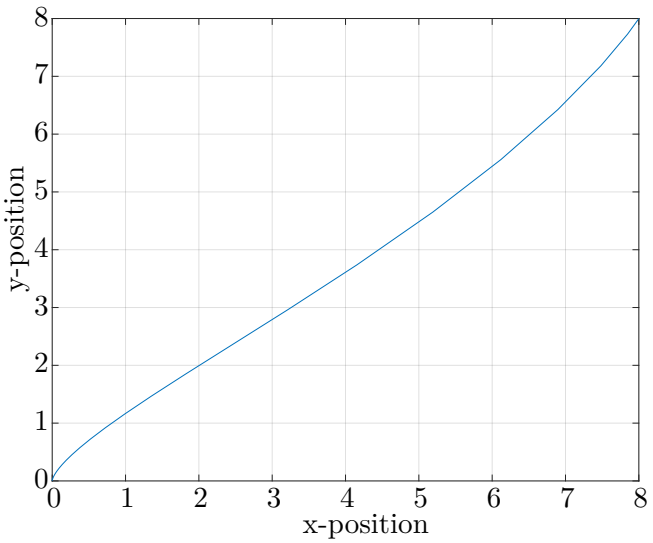

Figure 2.4. simple control, initial positions (8,8)

The following values $k_1 = 1$ $k_2 = 2$ $k_3 = 2$ $k_4 = 4$ are chosen, and the initial position is (8,8), Fig. 2.4. The LTI model is in many cases a too low degree approximation of a system. Real dynamics are more precisely represented by non-linear systems. Non-linear systems are generally written in this form:

$$\begin{cases} \dot{x} = f(x,u) \\ y = g(x,u) \end{cases}$$

A classic example used in a scholastic environment is the pendulum, where $x = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix}$:

$$\begin{cases} \dot{x_1} = x_2 \\ \dot{x_2} = -\frac{K}{J} \cdot \sin x_1 - \frac{\beta}{J} \cdot x_2 + \frac{1}{J} \cdot u \end{cases}$$

According to the 2^{nd} principle of dynamics (Newton's Law), from a physical point of view, K is the elastic constant, J is the moment of inertia and β is the friction coefficient. From the figure is easy to understand that the states represent the angle and the angular velocity, the control variable is the Torque. Since the control of this non-linear system is more complex, will be treated in the second Chapter (2) of Sliding Mode theory as an example.

2.3.1 Stability

There are many definitions for stability, the mostly considered are the Bounded Input Bounded Output (BIBO) and the internal stability. Having a dynamical system, by

defining an initial state and integrating the differential equations is possible to define an evolution of the system called trajectory of the system. Now by defining two different initial states, and integrating, two trajectories are generated $x_1(t)$ and $x_2(t)$, by analyzing the two trajectory is possible to conclude that a system is [7]:

- **internally asymptotically stable** if $\lim_{t\to\infty} ||x_1(t) - x_2(t)|| = 0$
- **internally marginally stable** if $\exists\sigma, \exists\epsilon : ||x_1(0) - x_2(0)|| < \sigma \implies ||x_1(t) - x_2(t)|| < \epsilon, \forall t$
- **internally unstable** otherwise

The BIBO stability concept, can be expressed as, for any bounded initial conditions, and for any bounded input $||u(t)|| < M_u < \infty$, also the output of the system is bounded $|||y(t)|| < M_y < \infty$. In the case of LTI systems, stability is an internal property of the system, it doesn't depend on the inputs, so it is veryfied for every possible input signal. Because of this propriety, only the matrix A has to be analyzed. In particular, assuming A is diagonalizable, from the matrix is possible to calculate the equilibrium point and the eigenvalues. The equilibrium point in LTI system (if A is diagonalizable) is only one, and is the point where $\dot{x} = 0$, the states does not move without any perturbation. The eigenvalues can be calculated solving the $[m \cdot n]$ system:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

After calculating the eigenvalues, a theorem defines the system internal stability.
Theorem. The LTI system is internally:

- **asymptotically stable** iff $Re(\lambda_i) < 0, \forall i$
- **marginally stable** iff $Re(\lambda_i) <= 0$, and $k_l = 1$ for $\forall l : Re(\lambda_l) = 0$
- **unstable** iff $\exists i : Re(\lambda_i) > 0$ or $\exists i : Re(\lambda_i) = 0$ and $k_i > 1$

Two simple 2x2 matrix are given to make some example of theorem application:

$$A_1 = \begin{bmatrix} -3 & -2 \\ 1 & 0 \end{bmatrix} \qquad A_2 = \begin{bmatrix} -3 & 1 \\ 1 & 0 \end{bmatrix}$$

For both the two matrix the equilibrium point can be founded by solving the simple system $A\bar{x} = 0$, and is equal to $\begin{bmatrix} \overline{x_1} \\ \overline{x_2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$. The eigenvalues of matrix A_1 are equal to $\lambda_1 = -2$ $\lambda_2 = -1$, both with real part strictly < 0, so the first system is asymptotically stable, the system's trajectory converges to it's equilibrium point 2.5. The eigenvalues associated to the second matrix are $\lambda_1 = -3.3028$ and $\lambda_2 = 0.3028$, so the second system is unstable, the trajectory diverges 2.6. An important implication links internal stability and BIBO stability in LTI systems, the inverse implication do not hold generally:

$$\textbf{asymptotically stable} \implies \textbf{BIBO stable}$$

If a system is not stable, it's possible to stabilize the system by manipulating the inputs. Let's assume that a system is controllable, the concept will be defined in next section 2.3.2, it is possible then to implement a linear state feedback stabilization, which is also a form of control as will be shown, that consist on defining a K matrix to change the system dynamic. As example the same model of (3.1) is used. The system is clearly unstable, but as will be shown is controllable, so is possible to define input values to make the system stable. A way is to design a K matrix so that $u = Kx$ so that the simple dynamic (2.1) becomes:

$$\begin{cases} \dot{x} = (A + BK)x \\ y = Cx \end{cases}$$

K can be chose as in the equation (2.3), $\mathrm{K} = \begin{bmatrix} -1 & -2 & 0 & 0 \\ 0 & 0 & -2 & -4 \end{bmatrix}$, so that the matrix $(A + BK)$ becomes asymptotically stable. This method can be considered a control technique because if a reference value r is added to the input, $u = Kx + r$ the states converge to r and not zero. The stability analisys discussed so far works for LTI models, but real world systems are non-linear. A characteristic of non-linear systems is that the number of equilibrium points is greater than one, so is hard to define a global concept of stability. **Lyapunov's** methods are mostly used in non-linear stability study, tehy are two:

- **Linearization Method:** it's a local analysis of the stability properties of it's linear approximation around an equilibrium point.
- **Direct Method:** it's a global definition of stability obtained by the analysis of a energy like function $V(x)$ associated to the dynamic of the system. The chosen candidate function have to satisfy some relations:
  - $\dot{V} < 0$ for $x \neq 0$
  - $\lim_{|x| \to \infty} V = \infty$

  Generally $V(x)$ is chosen similar to a kinetic energy like function:

$$V(x) = \tfrac{1}{2}x^2$$

The first method is a sufficient but not necessary condition. Is very immediate, for example given the system:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -x_2 - \sin x_1 \end{cases}$$

The linearization around the origin equilibrium point can be computed with the jacobian matrix:

$$\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

After the linearizaion teh Kalman method discussed for linear system can be simply used, and it allow to conclude that the previous non-linear system is stable at the origin. If the Linearization method does not allow any conclusion, it is possible then to try use the Lyapunov Direct method.

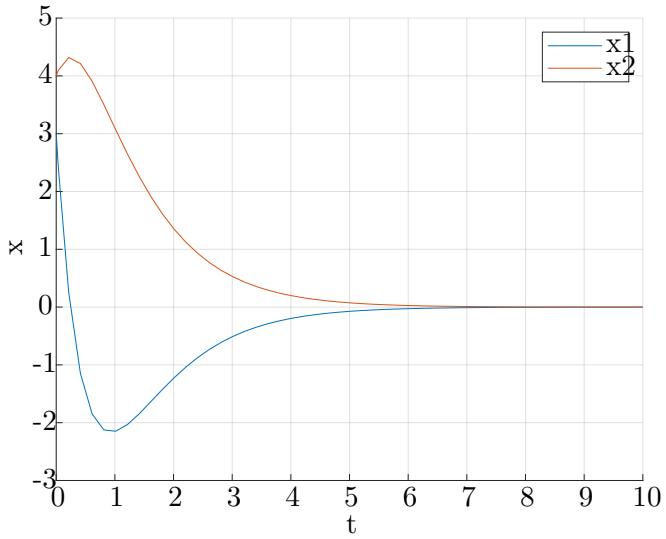

Figure 2.5. initial states $x_1 = 3$ $x_2 = 4$

2.3.2 Controllability

Controllability describes the possibility of action of the $u(\cdot)$ function to influence the states trajectory. It can be portrayed as the possibility to move the state from an initial value to a desired goal by acting on the command $u(\cdot)$. In LTI systems it can be easily proved by performing an easy matrix computation:

$$\begin{bmatrix} B & AB & A^2B & \cdots & A^{n-1}B \end{bmatrix} \tag{2.4}$$

Where n is the system dimension, it depend on the A matrix. If the matrix (2.4) is full rank, so that each row of the matrix is linearly independent from the others, then the system is said to be controllable, this is the so called kalman method. An example of a controllable system is the one in Eq. (3.1). In the case of nonlinear systems, is hard to define a global concept of controllability. A local evaluation can be done by linearizing the nonlinear system around an equilibrium point x_0. As an example the following nonlinear system form is taken:

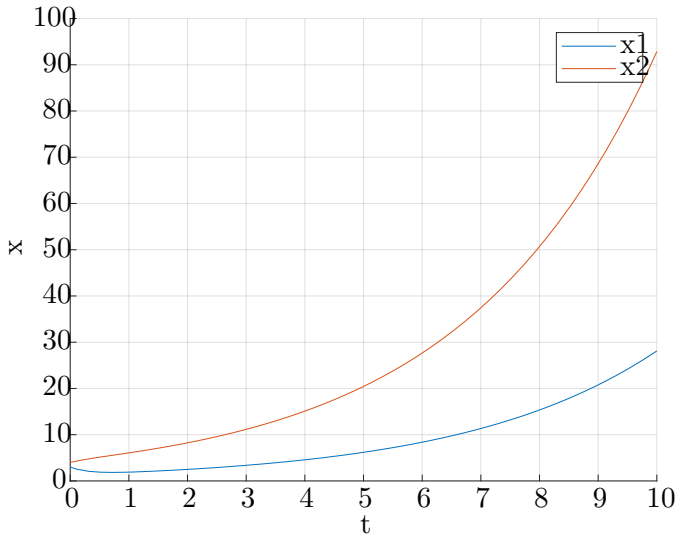

Figure 2.6. initial states $x_1 = 3$ $x_2 = 4$

$$\dot{x} = f(x) + \sum_{i=1}^{m} g(x)u_i \tag{2.5}$$

Where $f(x)$ is called drift vector field while $g(x)$ is the control input vector field. It can be linearized around an x_0 point performing partial derivatives:

$$\Delta\dot{x} = \left[\frac{\partial f}{\partial x}\right] \Delta x + \sum_{i=1}^{m} g(x_0)u_i \tag{2.6}$$

After linearization the kalman method can be applied to the matrix:

$$\left[g_1, \quad \cdots, \quad g_m, \quad \left[\frac{\partial f}{\partial x}\right] g_1, \cdots, \quad \left[\frac{\partial f}{\partial x}\right] g_m, \quad \left[\frac{\partial f}{\partial x}\right]^{n-1} g_1, \quad \cdots, \quad \left[\frac{\partial f}{\partial x}\right]^{n-1} g_m\right]$$

Is important to state, as explained here [8], that this method represent a sufficient but not necessary condition, so if the kalman requirement is not meet, the system could still be controllable, another way to analyze it is with the so called Lie Algebraic Rank Condition (LARC). It defines the concept of accessibility, which is similar to controllability. Local accessibility means that a system can be steered from x_0 to a non-empty set of reachable point around x_0. For driftless systems where $f(x) = 0$, for example the case of non-holonomic kinematic models, accessibility is equivalent to controllability, the only difference is that inputs could be strictly positive, in that case, accessibility contain the

controllability set. Accessibility can be evaluated calculating the so called lie brackets. For two V_1 and V_2 vectors, lie brackets are defined as:

$$\Big[V_1, V_2\Big] = \frac{\partial V_2}{\partial x} V_1 - \frac{\partial V_1}{\partial x} V_2 \tag{2.7}$$

For the Eq. (2.5) accessibility can be calculated by applying the kalman method to the matrix (2.8):

$$\Big[g_1, \cdots, g_m, [f, g_1], \cdots, [f, g_m], \cdots, [ad^{n-1} f, g_1], \cdots, [ad^{n-1} f, g_m]\Big] \tag{2.8}$$

2.3.3 Observability

Until now only systems with all known states have been considered, like in case (3.1), where the matrix C was considered of all ones. In real world cases, not all states are always measured, for many reasons, for example sensors may be expensive and used for large scale production. If some states are not measured the interdependence between the states variables of the model may provide the possibility to reconstruct the non measured states. This is the concept of Observability, a state is observable if is possible to reconstruct its value by the knowledge of other states. The reconstruction of a state can be also used for ensuring fault tolerance in a devices. As for the Controllability concept, in LTI systems the observability is very simple to verify, again the Kalman method can be used:

$$O_{bs} = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix}$$

It depend only on A and C, and if the rank(O_{bs}) matrix is equal to n the system is fully observable, so is possilbe to reconstruct all the states. The easiest example in LTI systems is the Luenberg observer/estimator, it is implemented as a simple L matrix. In Fig. 2.7 a description of its positioning in control scheme block. The hat simbol is to recall that the state is estimated:

$$\dot{\hat{x}} = A\hat{x} + Bu + LC[x - \hat{x}] \tag{2.9}$$

L matrix is defined analyzing the error estimation $\tilde{x} = x - \hat{x}$, which time derivative becomes $\dot{\tilde{x}} = \dot{x} - \dot{\hat{x}}$, by substituting then Eq. (2.9) and (2.1):

$$\dot{\tilde{x}} = (A - LC)\tilde{x} \tag{2.10}$$

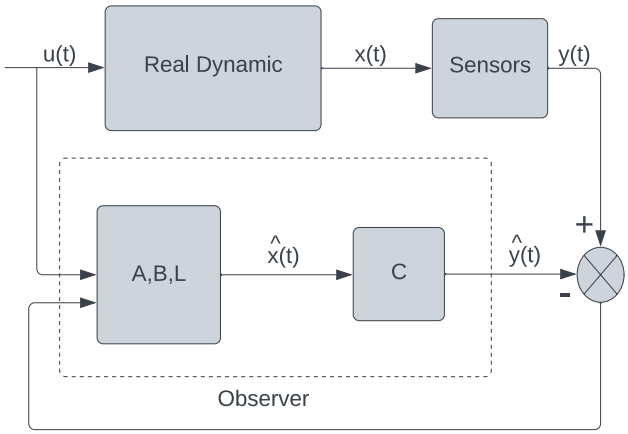

Figure 2.7. Representation of a Luenberg Observer in a control system

If the system is observable, the eigenvalues, as discussed for stability, can be placed arbitrarly to be negative so that the error $\tilde{x}$ goes to zero in a finite time, and $\hat{x} = x$. A simple example is proposed, the folowing LTI system is used as example:

$$\dot{x} = \begin{bmatrix} -1 & 1 \\ 0 & -3 \end{bmatrix} x + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u \tag{2.11}$$

The system is observable, so is possible to chose arbitrary eigenvalues, in this example $\lambda_1 = -5$,$\lambda_2 = -8$. The L matrix is then chosen as $L = \begin{bmatrix} 9 & 10 \end{bmatrix}$. Simulations are shown on Fig. 2.8, where can be seen as $\hat{x}_1$ reach x_1 after neither 1 second, and then $\hat{x}_2$ reach x_2. For non-linear systems, as discussed for controllability, the situation is way more complex. In very low cases is possible to say that a system is observable in a global definition, most of the times concepts of locality are used. Given a nonlinear system in the generic form:

$$\begin{cases} \dot{x} = f(x,t) + g(x,t)u \\ \text{y} = \text{h(x,t)} \end{cases} \tag{2.12}$$

Since observability depend on states relations and measured states $g(x,t)$ can be put to zero, and then it is possible to verify observability aound some x_0 states. For nonlinear systems a general method to verify controllability does not exist. There are many theorems and methods for specific nonlinear system forms, it's all about choosing which is applicable to a desired system. As explained in [9], a system is locally weakly observable if taken the lie derivatives with respect of the output function:

$$H = \frac{d}{dx} \begin{bmatrix} \mathcal{L}_f^0 h \\ \vdots \\ \mathcal{L}_f^{k-1} h \end{bmatrix}$$

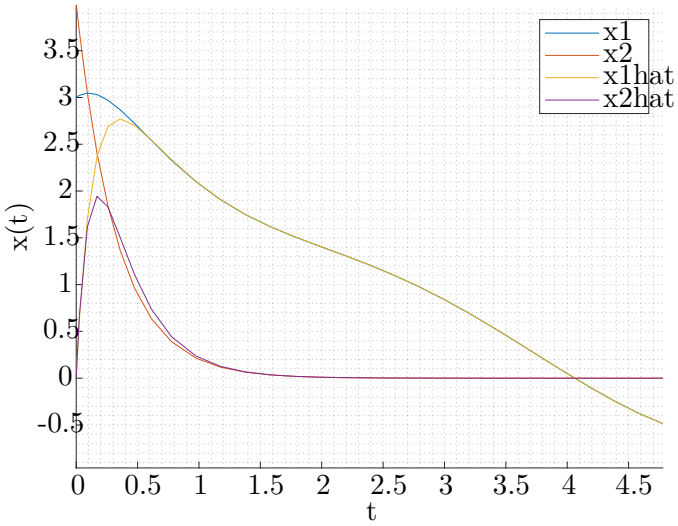

Figure 2.8. Luenberg observer simulation, x1hat is $\hat{x}_1$ x2hat is $\hat{x}_2$

If the rank$(H) = n$ then the system (2.12) is locally weakly observable at x_0.

2.4 Classical control algorithm

In this section some largely used control methods are described.

2.4.1 PID

Proportional-Integral-Derivative control is probably the most common control algorithm used in industrial application. It's popularity can be attributed to it's low cost and simplicity still allowing good performances. The closed loop application of the PID can be represented as in Fig. 2.9. The control law $u(t)$ is defined as [10]:

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt} \tag{2.13}$$

The aim is to chose the three values to obtain:

- Fast rise time
- Minimal overshoot
- Zero steady-state error

A way to achieve that is to define the three constants K_p,K_i,K_d by analyzing the closed loop transfer function, so that the poles are < 0 in absolute value as discussed in stability section 2.3.1. In a practical case a that is not enough, a trial-and-error process is necessary.

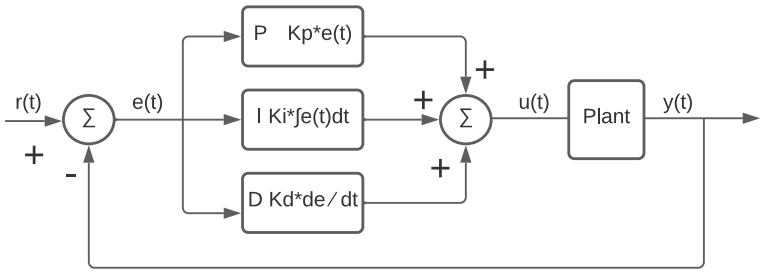

Figure 2.9. Representation of a closed loop PID application

2.4.2 Extended Kalman Filter

Is a non-linear example of a states estimator. Until now only ideal systems have been considered, a more precise representation of a system account also the influence of disturbances, both from the sensors and specific of the system model. For the description of the EKF a discrete time explanation is made, it can be easily obtained with a simple Euler's method:

$\dot{x}$ can be written as $\frac{x_{k+1}-xk}{Ts}$ Ts is the sampling time.

A non linear system in a discrete-time generic form can be written as:

$$\begin{cases} x_{k+1} = f(x_k, u_k) + d^x \\ y_k = h(x_k, u) + d^y \end{cases} \tag{2.14}$$

d^x is a disturbance while d^y is the measurement noise. Both can be considered as Gaussian noises with a certain variance. In many cases the sensor noise is known a priory. The EKF algorithm is based on two phases [7]:

- Prediction phase, the model is used to make a prediction of the x_k^p value

$$x_k^p = f(\hat{x}_{k-1}, u_{k-1})$$
$$P_k^p = F_{k-1} P_{k-1} F_{k-1}^T + Q^d$$

  The function $f(\hat{x}_{k-1}, u_{k-1})$ represent the known model, P_k is the so called covariance matrix. F_k is the Jacobian $\frac{\partial f}{\partial x}$ matrix, it represent the linearization of the f function. Q^d is the covariance matrix of d^x, generally it is defined with d^x values in its diagonal.

- Update phase, the prediction is corrected using output informations and noise consideration to improve the predicted value

$$
\begin{aligned}
S_k &= H_k P_k^p H_k^T + R^d \\
K_k &= P_k^p H_k^T S_k^{-1} \\
\Delta y_k &= y_k - h(x_k^p) \\
\hat{x}_k &= x_k^p + K_k \Delta y_k \\
P_k &= (I - K_k H_k) P_k^p
\end{aligned}
$$

  In this case R^d is the covariance matrix of d^y, is defined as the Q^d matirx for the output disturbance, instead H_k is the jacobian matrix of the output function. The updated value is then again used for the next prediction. The value $\Delta y_k = y_k - \hat{y}_k$ is the estimation error, the aim is to bring it to zero.

2.4.3 Low-pass filters

Low-pass filters are not properly, controller devices, but their use in a control systems are very important, so they can find a space in this chapter. A low-pass filter is a device that is used to pass signals with a frequency lower that a chosen cutoff frequency while attenuates higher frequencies. A very simple example is the first order LPF, it is defined by the following transfer function:

$$
H = \frac{1}{(1 + T_f s)} \tag{2.15}
$$

The transfer function can be discretized and rewritten as:

$$
\begin{gathered}
y(k) = (1 - \alpha) y(k-1) + \alpha u(k) \\
\alpha = \frac{h}{T_f + h}
\end{gathered}
$$

Where h is the sampling time u is the input noisy signal, and y the output of the filter. The aim is to chose α given h to cut a certain frequency:

$$
\frac{1}{T_f} = f_c = \frac{\alpha}{h(1-\alpha)}
$$

Considering a signal with mean value equal to zero, given α it is possible to calculate the ratio between the input variation and output variation as:

$$
\frac{\sigma_y}{\sigma_u} = \sqrt{\frac{\alpha}{2-\alpha}}
$$

σ is the standard deviation, so for a zero mean value signal, Var $= \sigma^2$, then:

$$
\frac{Var(y)}{Var(u)} = \frac{\alpha}{2-\alpha}
$$

To make an example, given a normally (Gaussian) distributed random signal with variance equal to 10^{-2}, sampling time equal to 0.1, choosing $\alpha = 0.1$, the filtering result is showed in Fig. 2.10. The output signal variance becomes then:

$$
Var(y) \approx 0.0526 \cdot 10^{-2} \approx 5.2 \cdot 10^{-4}
$$

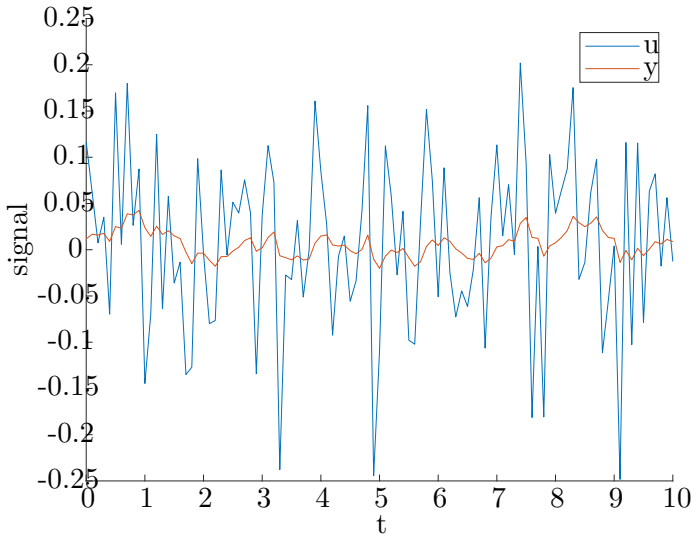

Figure 2.10. y is the filtered signal

The filter generates an output time delay, so in general is important to balance cut-frequency and time delay. Generally variance and mean value of the noise sensor is known, in any case is possible to evaluate those values, by collecting a large number of data of the sensor.

Chapter 3

Sliding mode theory

Sliding Mode Control (SMC) is a non-linear control technique that owns remarkable properties of robustness, accuracy, and is known to be easy to implement. This method consist in defining a so called sliding surface, to drive the states along this surface and maintaining them in its neighbour. The sliding surface is defined so that a relation between states and inputs explicitly appear. One of the first studies of the sliding method can be found here [11].

3.1 First Order Sliding Mode

A very simple example of how it works can be made by considering again the control problem (3.1). In this case a non-linear bounded disturbance is considered:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = f(x,t) + u_1 \\ \dot{x}_3 = x_4 \\ \dot{x}_4 = f(x,t) + u_2 \end{cases} \tag{3.1}$$

The function $f(x,t)$ is a sinusoidal bounded disturbance $2sin(t) <= |2|$. The Fig. 3.4 shows that the first control method used (2.3) does not compensate the non-linear disturbance effect. The position oscillates around the origin without reaching it Fig 3.5. To apply the sliding mode algorithm [3], first of all a **sliding variable** has to be chosen. In this case $s_1 = x_1$ and $s_2 = x_3$ are the two sliding variables, then two constrain functions are defined as $\sigma_1 = s_1 + \dot{s}_1$ and $\sigma_2 = s_2 + \dot{s}_2$. In order to achieve asymptotic convergence to zero of all the states, the two sliding variables s_1, s_2 must converge to zero in a finite time by means of the two control variables u_1, u_2. The stability with a non-linear disturbance can be analyzed with Lyapunov theory, so a lyapunov function must be chosen such that, given:

$$V = \frac{1}{2}s_1^2 \tag{3.2}$$

The following two rules are satisfied:

- $\dot{V} < 0$ for $s_1 \neq 0$
- $\lim_{|s_1| \to \infty} V = \infty$

The second rule is obviously satisfied, the first one can be rewritten as:

$$\dot{V} <= -\alpha V^{\frac{1}{2}}, \alpha > 0 \tag{3.3}$$

Its integration between time $0 <= \tau <= t$ gives:

$$V^{\frac{1}{2}}(t) <= -\frac{1}{2}\alpha t + V^{\frac{1}{2}}(0)$$

The "energy" dissipates in a finite time:

$$t <= 2\frac{V^{\frac{1}{2}}(0)}{\alpha}$$

To achieve this, u_1 (same for u_2, the control problem is mirrored) can be chosen, from $\dot{s}_1$ as:

$$u_1 = -x_2 - \rho sign(s_1)$$

The time derivative of (3.2) becomes:

$$\dot{V} = s_1\dot{s}_1 = s_1(x_2 + f(x,t) + u_1) = s_1(x_2 + f(x,t) - x_2 - \rho sign(s_1)) \tag{3.4}$$

It is not important to know the function $f(x,t)$, what is important is its bounds limits. In this case its limit is defined as $<= 2$, so is possible to approximate the dynamic (3.4) as:

$$\dot{V} <= 2|s_1| - s_1\rho sign(s_1)$$

Where

$$sign(x) = \begin{cases} 1 & x > 0 \\ -1 & x < 0 \end{cases}$$

Then the disequation becomes:

$$\dot{V} <= 2|s_1| - |s_1|\rho$$

And taking into account (3.3), ρ has to be chosen as:

$$\rho >= 2 + \frac{\alpha}{\sqrt{2}} \tag{3.5}$$

The equation (3.5) assert that, to stay in the sliding surface, and attenuate the non-linear effect of the disturbance the value of ρ has to be chosen greater than the maximum bounded value of the disturbance. To make a simulation, the disturbance is chosen as a sinusoidal function, but it is not important to know how it is modeled. The Fig. 3.1 shows that the disturbance effect can be compensated by choosing $\rho_1 = 2.1$ and $\rho_2 = 2.1$, the control law is then:

$$\begin{cases} u_1 = -x_2 - 2.1 sign(s_1) \\ u_2 = -x_4 - 2.1 sign(s_2) \end{cases}$$

The sliding variables s_1 and s_2 in this case, stay near zero Fig. 3.6. This other figure 3.2, instead, shows what happen if the disequation (3.5) is not verified. In this case is clear that the non-linear disturbance effect is not compensated, so the sliding variables does not stay close to zero. The sliding surface, instead can be seen as the straight line between the states, when the sliding variable goes to zero. By increasing the values ρ the sliding surface convergence is faster, with a side effect, it increases a "zig-zag" dynamic, called chattering that depend on the switching *sign* function Fig. 3.3.

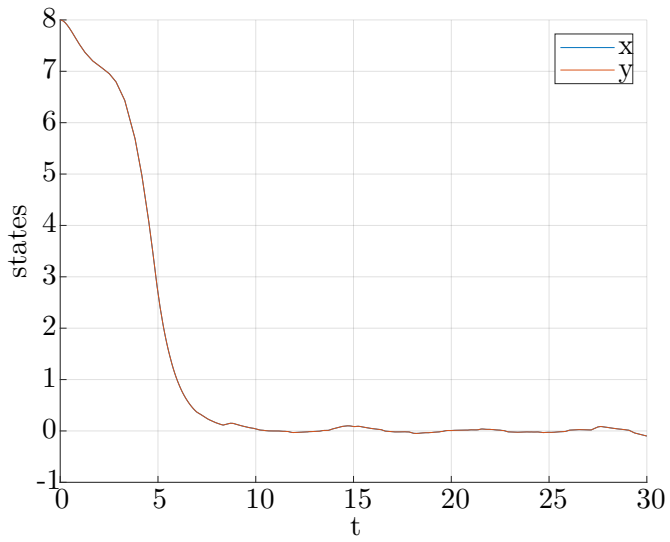

Figure 3.1. With the sliding mode control, choosing $\rho_1 = 2.1$, $\rho_2 = 2.1$ the states converge to zero, compensating the sinusoidal disturbance.

3.2 Higher Order Sliding Mode

As explained in the first order, sliding mode is a technique that allows to maintain to zero a certain constrain. Given a non-linear system in the form:

$$\begin{aligned} \dot{x} &= f(x,t) + g(x,t)u \\ y &= h(x,t) \end{aligned} \tag{3.6}$$

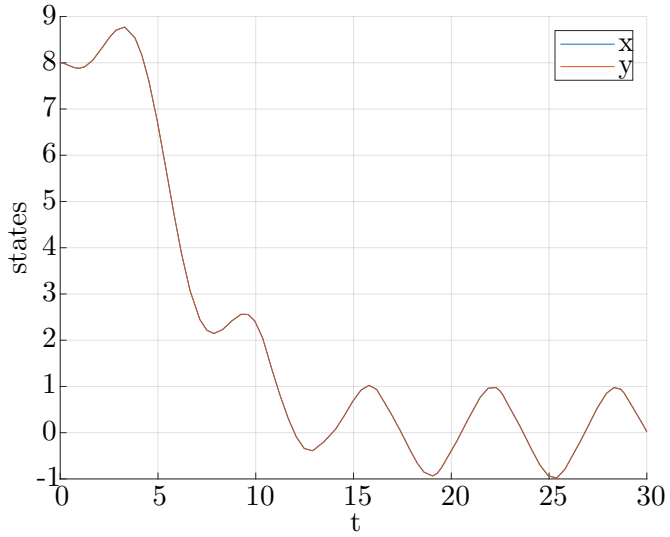

Figure 3.2. With the sliding mode control, choosing $\rho_1 = 1$, $\rho_2 = 1$ the states does not stay near zero.

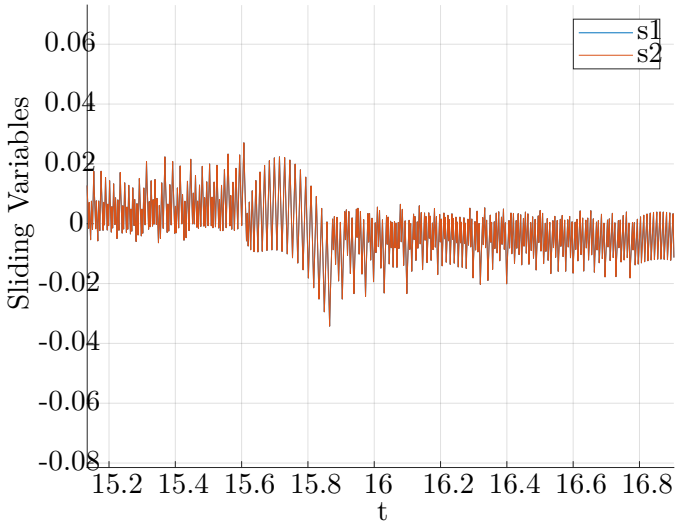

Figure 3.3. Example of chattering phenomenon.

In many cases the aim is to define an error $e = h(x,t) - h_r(x,t)$ (r means reference) so the sliding variable can be chosen as $s = e$. The constrain then is s, and driving it to zero allows to reach the required reference so that $e = 0$. To explain the difference between a first order sliding mode and higher order sliding modes, two important definitions are described:

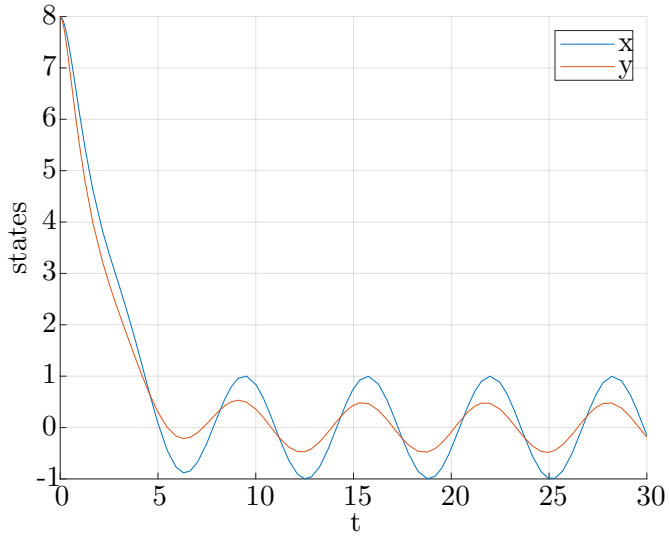

Figure 3.4. The states oscillates around the zero value without compensating the sinusoidal disturbance.

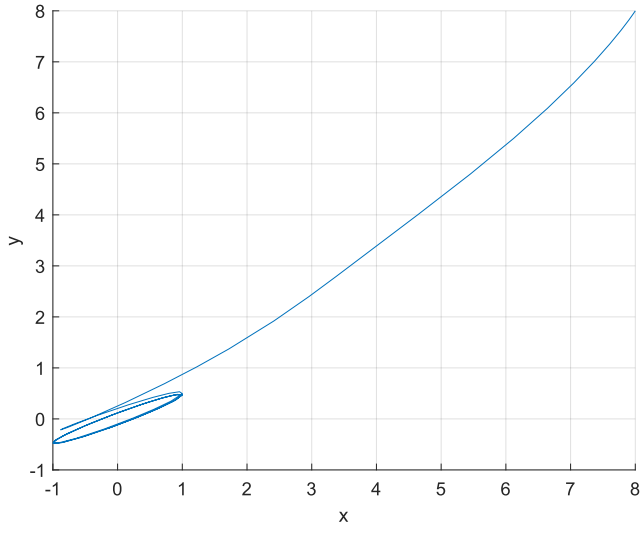

Figure 3.5. The (0,0) position canno't be reached with this linear control method.

- **Sliding Order**: given the sliding variable s, by computing n time derivatives of s, $[\dot{s}, \ddot{s}, \ldots, s^n]$. The sliding order is the derivative order n of s that contains a "sliding discontinuity", for example the *sign* function.
- **Relative degree**: the relative degree, in general, is a propriety of the system to control. It depend on the states relationships of the system. By computing the

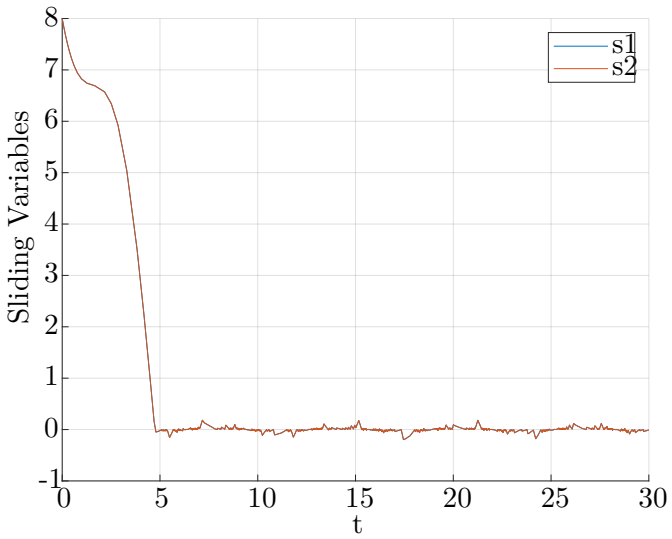

Figure 3.6. The sliding variables s_1 and s_2 stay close to zero.

Lie Derivatives of (3.6), with respect to the output $h(x,t)$ function, the following relation is obtained:

$$\frac{d(L_f^{n-1}h(x))}{dt} = L_f^n h(x) + L_g L_f^{n-1} h(x) u$$

The relative degree is the first n where $L_g L_f^{n-1} h(x) u \neq 0$ In short words, is the first n value of time derivatives of the outputs $[\dot{h}, \ddot{h}, \ldots, h^n]$ in which the command explicitly appears.

The first order sliding mode allows to drive, but not maintain exactly equal to zero, sliding variables of relative degree equal to one. The example (3.1) was chosen on purpose, in that case is easy to verify that the relative degree of the constrain is 1:

$$\begin{aligned} s_1 &= x_1 + \dot{x}_1 = x_1 + x_2 \\ \dot{s}_1 &= x_2 + u_1 \\ s_2 &= x_3 + \dot{x}_3 = x_3 + x_4 \\ \dot{s}_1 &= x_4 + u_2 \end{aligned}$$

The commands u_1 and u_2 appear in the first time derivative. A first order sliding mode allows to drive to zero s but not it's derivative $\dot{s}$, this causes the chattering of Fig. 3.6, where the sliding variables oscillates very close to zero. To ensure that the constrain converge to zero, and stays there, without chattering, the sliding order must be greater

than the relative degree of the constrain. For the previous example at least a second order sliding mode algorithms is necessary.

$$\dot{\sigma}$$

3.2.1 Supertwisting algorithm

The super-twisting algorithm is a second order sliding mode technique. It allows to drive to zero $s = \dot{s} = 0$, avoiding chattering, in a system with relative degree 1.

$$u = -\lambda|\sigma|^{\frac{1}{2}} sign(\sigma) + v \tag{3.7}$$

$$\dot{v} = \begin{cases} -u & |u| > U_m \\ -\alpha sign(\sigma) & |u| <= U_m \end{cases} \tag{3.8}$$

U_m comes from teh following relations from the time derivative of the function constrain:

$$\dot{\sigma} = h(x) + g(x)u \tag{3.9}$$

By considering $h(x)$ and $g(x)$ as bounded functions, then:

$$|\dot{h}| + U_m|\dot{g}| <= C, \quad 0 < K_m <= g(x) <= K_M, \quad |h/g| < qU_m, \quad 0 < q < 1$$

From these relations, then λ can be chosen as [3]:

$$\lambda > \sqrt{\frac{2}{(K_m\alpha - C)}} \frac{(K_m\alpha + C)K_M(1+q)}{K_m^2(1-q)}$$

A super-twisting algorithm is now applied to the example (3.1), to show that in this case, not only $s_1 = 0$, but also its time derivative $\dot{s}_1 = 0$, and chattering is absent. The control law is defined as:

$$\begin{cases} \mathrm{u}_1 = -x_2 - 1.5\sqrt{2|s_1|}sign(s_1) + v_1 \\ \mathrm{u}_2 = -x_4 - 1.5\sqrt{2|s_2|}sign(s_2) + v_2 \\ \dot{\mathrm{v}}_1 = -2.2sign(s_1) \\ \dot{\mathrm{v}}_2 = -2.2sign(s_2) \end{cases}$$

The results are shown in Fig. 3.7.

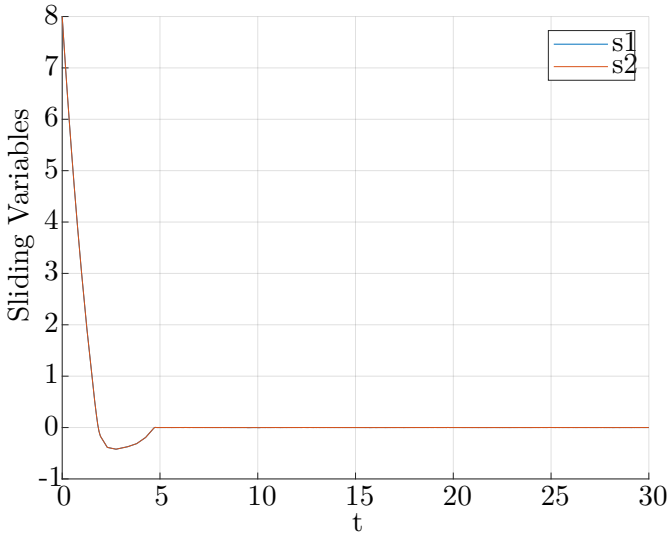

Figure 3.7. With super-twisting algorithm, s_1 and s_2 converge to zero and stays there without chattering.

3.3 Sliding Mode Observers

As discussed, the purpose of an observer is to estimate the unmeasured states using the states relationships of the model. The linear Luenberger observer defined in Eq. (2.9), usually fails estimating the states in the presence of an unknown signal or disturbances. Sliding mode observers, as in the control case, make use of a discontinuity function, like the $sign(x)$ function, to drive the estimation error along a sliding surface to make the estimate states converge to the real values. Generally this types of observers are defined considering dynamical systems in a so called triangular input form [4]. A triangular input form can be represented as:

$$\begin{cases} \dot{\mathrm{x}}_1 = x_2 \\ \dot{\mathrm{x}}_2 = x_3 \\ \vdots \\ \dot{\mathrm{x}}_{n-1} = x_n \\ \dot{\mathrm{x}}_n = f(x) + g(x)u \end{cases}$$

Generally if a system is not in this form, there is a way to put it in a triangular form performing some transformations. Given a generic non-linear system:

$$\begin{cases} \dot{\mathrm{x}} = f(x) + \sum_{i=0}^{m} g_i(x)u_i \\ \mathrm{y} = \mathrm{h(x)} = [\mathrm{h}_1, h_2, \ldots, h_p]^T \end{cases}$$

if all the following three rules are satisfied, then it can be made a transformation into a triangular form:

- $k_1 >= k_2 >= \cdots >= k_p >= 0$
- $\sum_{i=0}^{p} k_i = n$
- $$rank \begin{bmatrix} dh_1(x) \\ dL_f h_1(x) \\ \vdots \\ dL_f^{k_1-1} h_1(x) \\ \vdots \\ dh_p(x) \\ \vdots \\ dL_f^{k_p-1} h_p(x) \end{bmatrix} = n$$

It means that the system is Locally Weakly Observable, and exists p integers $(k_1, k_2, \ldots, k_p)$ that form the smallest p-tuple with respect to the lexicographic ordering. In that case the transformation can be performed as:

$$\begin{aligned} \dot{\eta}_i &= A_i \eta_i + F_i(\eta) + G_i(\eta) u \\ y_i &= \begin{bmatrix} 1 & 0 & \ldots & 0 \end{bmatrix} \eta_i \end{aligned}$$

Where:

$$A_i = \begin{bmatrix} 0 & 1 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \ldots & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \in R^{k_i \cdot k_i}$$

$$F_i = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ L_f^{k_i} h_i(x) \end{bmatrix} \in R^{k_i}$$

$$G_i = \begin{bmatrix} 0 & 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 0 \\ L_{g_1} L_f^{k_i-1} h_i(x) & L_{g_2} L_f^{k_i-1} h_i(x) & \ldots & \ldots & L_{g_m} L_f^{k_i-1} h_i(x) \end{bmatrix} \in R^{k_i \cdot m}$$

Since Higher Order Sliding Mode concept has already been discussed, this chapter will directly jump to the definition of a second order sliding mode observer the **Super-Twisting Observer**.

3.3.1 Super-Twisting Observers

Is one of the most popular second-order sliding mode observers technique [3]. Given the relation:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = f(t, x_1, x_2, u) \\ y = h(x) = x_1 \end{cases}$$

It means x_1 is measured, and x_2 is its velocity. This structure is in a triangular input form, so a simple super-twisting algorithm can be written as:

$$\begin{cases} \dot{\hat{x}}_1 = \hat{x}_2 + \lambda |x_1 - \hat{x}_1|^{1/2} sign(x_1 - \hat{x}_1) \\ \dot{\hat{x}}_2 = f(t, x_1, \hat{x}_2, u) + \alpha sign(x_1 - \hat{x}_1) \end{cases}$$

Where the "hat" symbol means that the state is an estimation. Now taking $\tilde{x}_1 = x_1 - \hat{x}$ and $\tilde{x}_2 = x_2 - \hat{x}_2$, as the estimation errors, and performing the time derivative:

$$\begin{cases} \dot{\tilde{x}}_1 = \tilde{x}_2 - \lambda |\tilde{x}_1|^{1/2} sign(\tilde{x}_1) \\ \dot{\tilde{x}}_2 = f(, t, x_1, x_2, u) - f(t, x_1, \hat{x}_2, u) + \epsilon(t, x_1, x_2, y) - \alpha sign(\tilde{x}_1) \end{cases}$$

The function ϵ is a disturbance, then if all the states are bounded, then the existence of a constant f^+ is ensured, such that:

$$|f(, t, x_1, x_2, u) - f(t, x_1, \hat{x}_2, u) + \epsilon(t, x_1, x_2, y)| < f^+$$

Then is the system is Bounded Input Bounded States (BIBS), as demonstrated on this work [12], the parameters α and λ can be chosen as:

$$\lambda = 1.5\sqrt{f^+}$$
$$\alpha = 1.1 f^+$$

This values ensure the convergence of the estimated states to the real states. In mechanical systems, generally, f^+ is defined as $2 \sup |acceleration|$.

Chapter 4

Kinematic Model

Kinematics a is field of physics that describes the motion of a rigid body in the space, without any consideration of the forces involved. The Kinematic relations depend on the body structure, and this influence the space of possible movement solutions for a fixed time coordinate. The kinematic model of a mobile robot is determined by its locomotion structure, in this case only a "wheeled" structure is considered. The main wheeled structures are [5]:

- Fixed
- Steering
- Caster
- Omnidirectional

The combination of this types of wheel in a rigid body structure generates some typical Kinmatic structures, which are:

- **Unicycle**: can be seen as a chassis with a single wheel, from a physical point of view there are obviously stability problems, but it will be shown that it can approximate more complex structures Fig. 4.1.
- **Bicycle**: is a vehicle with a fixed wheel and a turning wheel Fig. 4.2.
- **Tricycle**: is a vehicle with two fixed wheels in the rear axle, and a steering wheel in the front axle.
- **Differential drive**: is a vehicle with two rear differential actuated wheels, and a front caster wheel.
- **Synchronized drive**: approximate a vehicle with three steering wheel actuated synchronously with two engines, one for traction and one for wheel orientation.
- **Car**: approximate the structure of a car, with two front steering wheel and two rear fixed wheels.

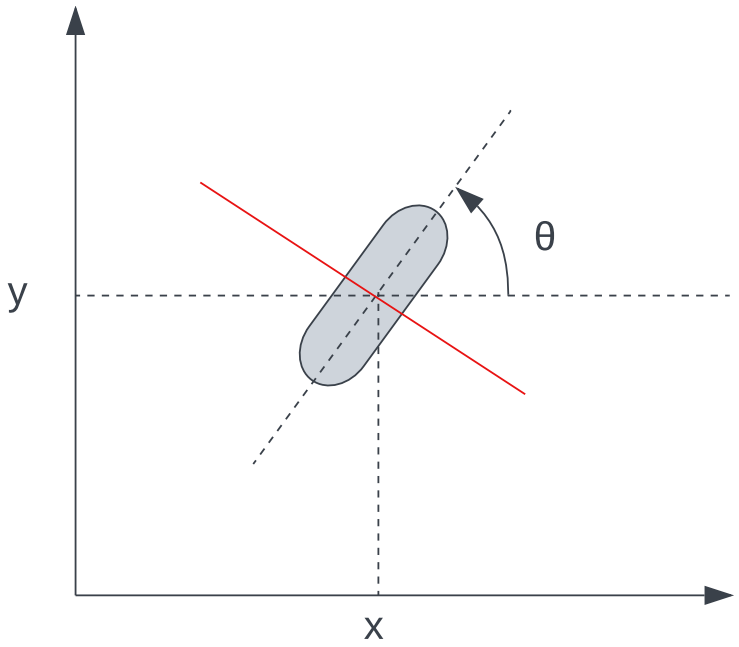

Figure 4.1. Unicycle representation

Wheeled vehicles are subjects to constraints to local mobility. Is easy to understand that a car can't move orthogonally, so the space of reachable points is locally limited. This tho does not preclude the possibility of reaching any point of the space. This constraints are classified as:

- Equalities called also **bilateral** constraints
- Disequlities called also **Unilateral** contraints

And can depend on time, in that case are called **Rehonomic** otherwise **Scleronomic**. In this case only **Bilateral Scleronomic** constraints are considered. Another classification is between **Holonomic** and **Non-holonomic** constraints:

- **Holonomic**: given a system with generaziled coordinates q, a constaint is defined as Holonmic if it is integrable and writtable as:

  $$h_i(q) = 0$$

  It does not depend on time derivatives of q. This type of constraint reduce the accessible space of possible configurations. A system is Holonomic if all the constraints are holonomic.

- **Non-holonomic**: a non-holonomic constaint, instead, is written as:

  $$a(q, \dot{q}) = 0$$

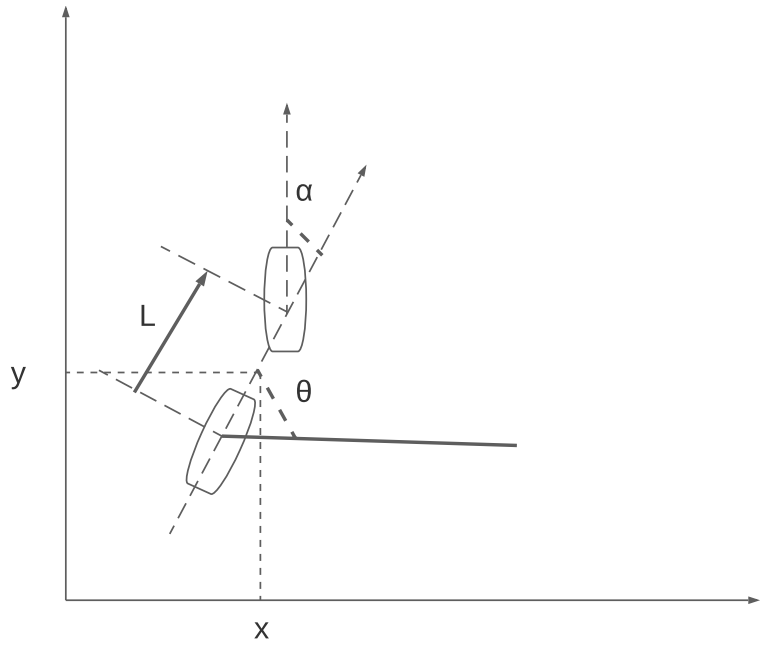

Figure 4.2. Bicycle representation, L is the distance between the two wheels, and the two angles are with respect to the inertial space. The position (x,y) can be chosen as the point in L/2.

So depend on velocities, ans can be brought to a so called Pfaffian form:

$$\frac{dh(q)}{dq}\dot{q} = 0$$

It is not integrable, so can't be brought back to a holonomic form. This type of constraints does not reduce the accessibility configuration space. It means that each point of the space can be reached, also if not instantaneously. An example is the **pure rolling constaint**.

The **pure rolling constraint** can be applied to the unicycle model. Given $q = \begin{bmatrix} x & y & \theta \end{bmatrix}$ It assert that the wheel can't move orthogonally to its asset, obviously in absence of slipping force, it is represented in Fig. 4.1 by the red line. It is wrtitten as:

$$\frac{dy}{dx} = \tan\theta$$

And can be brought to a Pfaffian form this way:

$$\frac{dy}{dx} = \frac{dy}{dt} \cdot \frac{dt}{dx} = \frac{\dot{y}}{\dot{x}} = \frac{\sin\theta}{\cos\theta} \implies -\dot{y}\cos\theta + \dot{x}\sin\theta = 0$$

It can be rearranged in a matrix form as:

$$\begin{bmatrix} \sin\theta & -\cos\theta & 0 \end{bmatrix} \dot{q} = 0$$

A nullspace of the previows matrix can be defined by two vectors:

$$g_1(q) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \end{bmatrix}$$

$$g_2(q) = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$$

It is possible then to define a kinematic model, for the unicycle "vehicle", that represent the space of possible locally reachable points as:

$$\dot{q} = \sum_{j=1}^{m} g_j(q) u_j = G(q)u$$

$$G(q) = \begin{bmatrix} \cos(\theta) & 0 \\ \sin\theta) & 0 \\ 0 & 1 \end{bmatrix}$$

So m inputs u are defined. In the case of the unicycle $m = 2$ so:

$$\dot{q} = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \\ 0 \end{bmatrix} u_1 + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u_2 \tag{4.1}$$

Where $u = \begin{bmatrix} v & w \end{bmatrix}$ that are the body linear velocity and angular velocity of the vehicle. Now one can question, how can a "wheel" model be useful for a more complex mobile robot? The unicycle kinematic model can be extended to approximate more complex kinematic structures, for example a two wheeled differential robot as in Fig. 4.3. It is possible to approximate the two wheeled vehicle kinematic by choosing:

$$v = \frac{r(w_r + w_l)}{2}$$
$$w = \frac{r(w_r - w_l)}{d}$$

Where w_r is the right wheel velocity and w_l is the velocity of the left wheel. For a control approach, it is useful to rewrite this kinematic relation to enhance a reference trajectory error [13]. Given a reference trajectory, defined by the system:

$$\begin{cases} \dot{\mathrm{x}}_r = v_r \cos(\theta_r) \\ \dot{\mathrm{y}}_r = v_r \sin(\theta_r) \\ \dot{\theta}_r = w_r \end{cases}$$

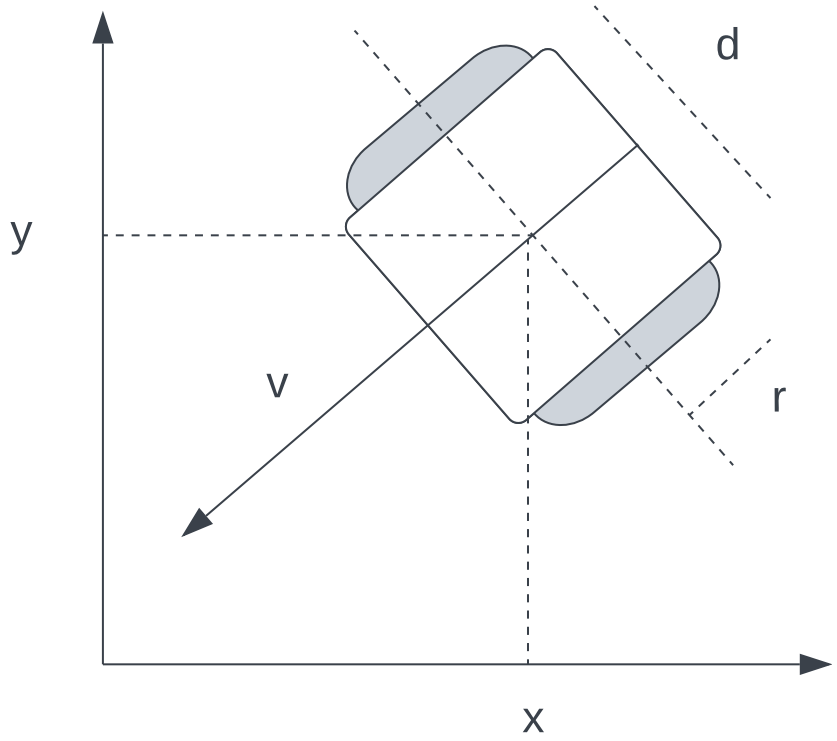

Figure 4.3. A simple two wheeled robot representation

A graphical representation is showed in Fig. 4.4. From the scheme of the previous figure, the error coordinates can be expressed, by performing a rotation with respect to z axis, and a translation:

$$\begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r - x \\ y_r - y \\ \theta_r - \theta \end{bmatrix}$$

Then by performing a time derivative of the states, and making some trigonometric substitutions, the error dynamic differential equation can be expressed as:

$$\begin{bmatrix} \dot{x}_e \\ \dot{y}_e \\ \dot{\theta}_e \end{bmatrix} = \begin{bmatrix} w y_e - v + v_r \cos\theta_e \\ -w x_e + v_r \sin\theta_e \\ w_r - w \end{bmatrix}$$

From this form is possible to design a controller to drive to zero the error states, so that the reference is reached.

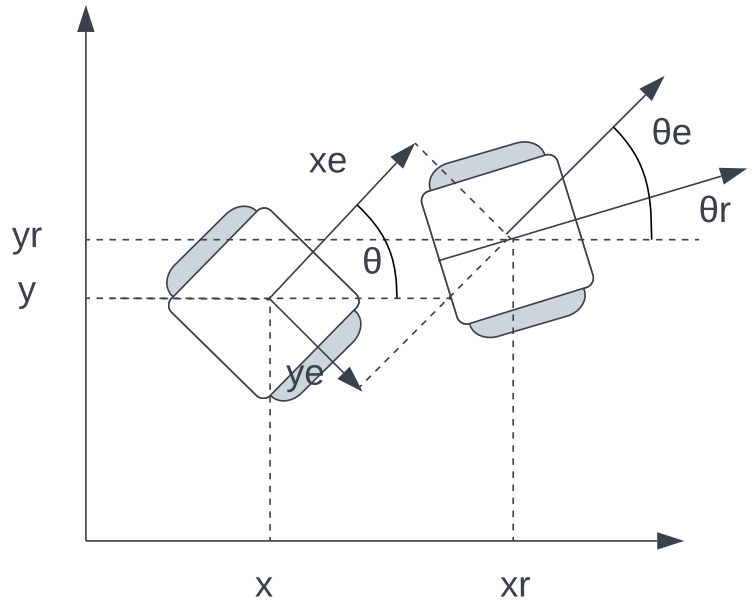

Figure 4.4. Reference trajectory representation

Chapter 5

Observer and Controller design

This chapter show the two algorithms design from a mathematical point of view, then the translation in python code will be shown in Appendix.

5.1 Observer Model

The Observer was designed to estimate two states, the velocity of the robot and its orientation. To achieve this, the kinematic model defined in Eq. 4.1 has been considered. It can be rewritten in a more explanatory form as:

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = w \end{cases} \tag{5.1}$$

As already explained, the super-twisting sliding mode algorithm was selected for this purpose, it is generally defined starting from a triangular form model. By performing two simple time derivatives of $v_x = \dot{x}$ and $v_y = \dot{y}$ the kinematic relations presented, can be rearranged in two triangular relations:

$$\begin{cases} \dot{x} = v_x = v\cos\theta \\ \dot{v}_x = a\cos\theta - vw\sin\theta \\ \dot{y} = v_y = v\sin\theta \\ \dot{v}_y = a\sin\theta + vw\cos\theta \end{cases} \tag{5.2}$$

Where a is the body velocity of the robot. The velocity can be considered as $v = \sqrt{{v_x}^2 + {v_y}^2}$. The acceleration a and the angular velocity in this case are considered as known from sensors, while θ and v_x, v_y and consequence v are estimated. Since the angle θ never appears isolated from cosine and sinus functions, $\cos\theta$ and $\sin\theta$ are considered as two state to estimate, then the angle θ is computed as $\arctan(\sin\theta, \cos\theta)$. From now

on, the following change of variable is considered $C = \cos\theta$, $S = \sin\theta$. At this point the model can be written as:

$$\begin{cases} \dot{x} = v_x = \sqrt{{v_x}^2 + {v_y}^2}C \\ \dot{v}_x = aC - \sqrt{{v_x}^2 + {v_y}^2}wS \\ \dot{y} = v_y = \sqrt{{v_x}^2 + {v_y}^2}S \\ \dot{v}_y = aS + \sqrt{{v_x}^2 + {v_y}^2}wC \\ \theta = \arctan(S, C) \end{cases} \tag{5.3}$$

To be more clear, the states are chosen as:

$$\begin{bmatrix} x \\ vx \\ C \\ y \\ vy \\ S \end{bmatrix} \text{ and the known outputs are } \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} \text{ from encoders}$$

The observability is reached by writing the relations in the following form:

$$\begin{cases} \dot{x} = vx \\ \dot{v}_x = aC - \sqrt{{v_x}^2 + {v_y}^2}wS \\ \dot{y} = vy \\ \dot{v}_y = aS + \sqrt{{v_x}^2 + {v_y}^2}wC \\ \dot{z}_1 = \sqrt{{v_x}^2 + {v_y}^2}C \\ \dot{z}_2 = \sqrt{{v_x}^2 + {v_y}^2}S \end{cases} \tag{5.4}$$

Obviously $z_1 = x$ and $z_2 = y$ are redundancies to respect the rank condition explained in the Observability section, and some examples of applications are here [4]. Those two redundant relations increase the states relations. This way the rank of the jacobian matrix associated to the lie derivatives is equal to 6, which is the states dimension, and this can be achieved by adding two fictitious outputs $h_3 = \dot{x}$ and $h_4 = \dot{y}$:

$$\begin{bmatrix} h_1 \\ \dot{h}_1 \\ \ddot{h}_1 \\ h_2 \\ \dot{h}_2 \\ \ddot{h}_2 \\ h_3 \\ \dot{h}_3 \\ h_4 \\ \dot{h}_4 \end{bmatrix} \implies rank \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & \frac{v_x C}{\sqrt{{v_x}^2+{v_y}^2}} & -\sqrt{{v_x}^2 + {v_y}^2}S & 0 & \frac{v_y C}{\sqrt{{v_x}^2+{v_y}^2}} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{v_x S}{\sqrt{{v_x}^2+{v_y}^2}} & 0 & 0 & \frac{v_y S}{\sqrt{{v_x}^2+{v_y}^2}} & \sqrt{{v_x}^2 + {v_y}^2}C \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} = 6$$

Now from Eq. 5.3 the super-twisting observer has been developed as follows:

$$\begin{cases} \dot{\hat{x}} = \hat{v}_x + \lambda_{11}|x - \hat{x}|^{1/2} sign(x - \hat{x}) \\ \dot{\hat{v}}_x = a\hat{C} - \sqrt{\hat{v}_x{}^2 + \hat{v}_y{}^2} w\hat{S} + \lambda_{12} sign(x - \hat{x}) \\ \dot{\hat{y}} = \hat{v}_y + \lambda_{21}|y - \hat{y}|^{1/2} sign(y - \hat{y}) \\ \dot{\hat{v}}_y = a\hat{S} + \sqrt{\hat{v}_x{}^2 + \hat{v}_y{}^2} w\hat{C} + \lambda_{22} sign(y - \hat{y}) \end{cases} \tag{5.5}$$

$$\begin{cases} \dot{\hat{z}}_1 = \sqrt{\hat{v}_x{}^2 + \hat{v}_y{}^2}\hat{C} + \lambda_{z1}|x - \hat{z}_1|^{1/2} sign(x - \hat{z}_1) \\ \dot{\hat{C}} = -w\hat{S} + \alpha_1 sign(x - \hat{z}_1) \\ \dot{\hat{z}}_2 = \sqrt{\hat{v}_x{}^2 + \hat{v}_y{}^2}\hat{S} + \lambda_{z2}|y - \hat{z}_2|^{1/2} sign(y - \hat{z}_2) \\ \dot{\hat{S}} = w\hat{C} + \alpha_2 sign(y - \hat{z}_2) \end{cases} \tag{5.6}$$

Now a stability analysis will be made, using lyapunov theory. Tendentially the two brackets above can be seen as to observers, 5.5 is the velocity observer, and 5.6 is the orientation observer. For the stability analysis a Lyapunov technique is used. The equation 5.5 can be divided in two triangular mirrored sub-relations, so analysis is performed along the x axis relation. The estimation errors are defined as:

$$\begin{aligned} \tilde{x} &= x - \hat{x} \\ \tilde{v}_x &= v_x - \hat{v}_x \end{aligned} \tag{5.7}$$

The time derivatives allow to analyze the error dynamic:

$$\begin{aligned} \dot{\tilde{x}} &= \tilde{v}_x - \lambda_{11}|\tilde{x}|^{1/2} sign(\tilde{x}) \\ \dot{\tilde{v}}_x &= a(C - \hat{C}) + \hat{v}w\hat{S} - vwS - \lambda_{12} sign(\tilde{x}) \end{aligned} \tag{5.8}$$

The stability analysis requires that the function $F(t, X, \hat{X}, u) = a(C - \hat{C}) + \hat{v}w\hat{S} - vwS$ is bounded. In this case capital X refers to the states, not inertial position. To simplify the stability validation from now on, the equation 5.5 is modified considering $C = \cos\theta$ and $S = \sin\theta$ so that the influence of the orientation observer is considered bounded, because cosine and sinus are bounded no matter how θ oscillates, while the same hypothesis can't be made about S and C treated as states values. Then a is the acceleration, so can be assumed bounded for a physical reason, same consideration for w (angular velocity). For the body velocity a very rough assumption is used, if $|\hat{v}_x| \leq 2 \sup |v_x|$ and $|\hat{v}_y| \leq 2 \sup |v_y|$ holds [13], and as said all other inputs are bounded, then exist for sure a constant value f^+:

$$|F(t, X, \hat{X}, u)| < f^+ \tag{5.9}$$

Then λ values can be taken as:

$$\begin{aligned}\lambda_{11} &= 1.5\sqrt{f^+}\\ \lambda_{12} &= 1.1f^+\\ \lambda_{21} &= 1.5\sqrt{f^+}\\ \lambda_{22} &= 1.1f^+\end{aligned}$$

Tendentially, the same upper value can be considered for both x, y dynamics. This tuning values, ensures that the Lyapunov functions associated to the estimation errors, have first time derivative with negative sign, so that the energy of the error dynamics goes to zero in a finite time:

$$\begin{aligned}V_1 &= \frac{1}{2}\tilde{x}\\ V_2 &= \frac{1}{2}\tilde{v_x}\\ \dot{V}_1 &= \tilde{x}(\tilde{v}_x - \lambda_{11}|\tilde{x}|^{1/2}sign(\tilde{x}))\\ \dot{V}_2 &= \tilde{v}_x(F(t, X, \hat{X}) - \lambda_{12}sign(\tilde{x}))\end{aligned}$$

This mean that after a time t, $\hat{v}_x \to v_x$ and $v_y \to v_y$, this implies that $\hat{v} \to v$. With the same reasonings already made, choosing:

$$\begin{aligned}\lambda_{z1} &= 1.5\sqrt{{f_z}^+}\\ \alpha_1 &= 1.1{f_z}^+\\ \lambda_{z2} &= 1.5\sqrt{{f_z}^+}\\ \alpha_2 &= 1.1{f_z}^+\end{aligned}$$

The orientation error dynamics becomes:

$$\dot{\tilde{z}}_1 = vC - \hat{v}\hat{C} - \lambda_{z1}|\tilde{z}_z|^{1/2}sign(\tilde{z}_1)$$

The tuning value ensure to drive $\dot{\tilde{z}}_1 = 0$ and $\tilde{z}_1 = 0$, and after a finite time t when $\hat{v} = v$:

$$0 = vC - v\hat{C} \implies C = \hat{C}$$

Same for $\hat{S}$, all is specular. The model can be seen as two nonlinear Globally Asymptotically Stable (GAS) cascaded observers Fig. 5.1, and as explained in this study [14], nonlinear cascaded GAS systems are stable. The model can be then discretized using Euler's method, it can be rearranged as:

$$\begin{cases} \hat{x}_{k+1} = \hat{x}_k + T_f(\hat{v}_{xk} + \lambda_{11}|x - \hat{x}|^{1/2} sign(x - \hat{x}_k)) \\ \hat{v}_{xk+1} = \hat{v}_{xk} + T_f(a\cos\hat{\theta}_k - \sqrt{\hat{v}_{xk}{}^2 + \hat{v}_{yk}{}^2} w \sin\hat{\theta}_k + \lambda_{12} sign(x - \hat{x}_k)) \\ \hat{y}_{k+1} = \hat{y}_k + T_f(\hat{v}_y + \lambda_{21}|y - \hat{y}_k|^{1/2} sign(y - \hat{y}_k)) \\ \hat{v}_{yk+1} = \hat{v}_{yk} + T_f(a\sin\hat{\theta}_k + \sqrt{\hat{v}_{xk}{}^2 + \hat{v}_{yk}{}^2} w \cos\hat{\theta}_k + \lambda_{22} sign(y - \hat{y}_k)) \end{cases} \tag{5.10}$$

$$\begin{cases} \hat{z}_{1k+1} = \hat{z}_{1k} + T_f(\sqrt{\hat{v}_{xk}{}^2 + \hat{v}_{yk}{}^2}\hat{C}_k + \lambda_{z1}|x - \hat{z}_{1k}|^{1/2} sign(x - \hat{z}_{1k})) \\ \hat{C}_{k+1} = \hat{C}_k + T_f(-w\hat{S}_k + \alpha_1 sign(x - \hat{z}_1)) \\ \hat{z}_{2k+1} = \hat{z}_k + T_f(\sqrt{\hat{v}_{xk}{}^2 + \hat{v}_{yk}{}^2}\hat{S}_k + \lambda_{z2}|y - \hat{z}_{2k}|^{1/2} sign(y - \hat{z}_{2k})) \\ \hat{S}_{k+1} = \hat{S}_k + T_f(w\hat{C}_k + \alpha_2 sign(y - \hat{z}_{2k})) \end{cases} \tag{5.11}$$

$$\hat{\theta}_{k+1} = \arctan\hat{S}_k, \hat{C}_k \tag{5.12}$$

T_f is the working step time of the observer, for 100Hz it means $T_f = 0.01s$.

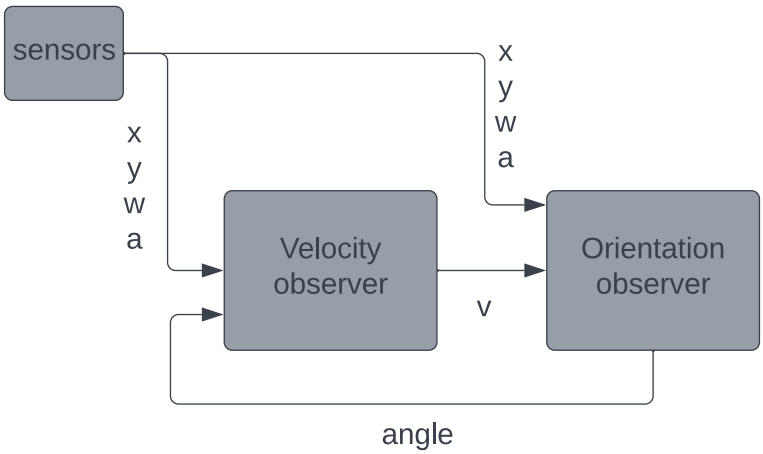

Figure 5.1. Cascaded observers

5.2 Controller Model

Also in this case a second order super twisting sliding mode algorithm was chosen. The reference kinematic model is the trajectory error dynamic:

$$\begin{bmatrix} \dot{x}_e \\ \dot{y}_e \\ \dot{\theta}_e \end{bmatrix} = \begin{bmatrix} wy_e - v + v_r \cos\theta_e \\ -wx_e + v_r \sin\theta_e \\ w_r - w \end{bmatrix}$$

As explained in Chapter 3, the first thing to do consist in defining the sliding variables. The following three sliding variables are chosen:

$$s_1 = x_e$$
$$s_2 = y_e$$
$$s_3 = \theta_e$$

Those three variables defines the error states between actual and reference trajectory, as showed in Fig. 2.2. The sliding variables are of relative degree of 1, it means that the control variables appear in the first time derivatives. The aim is to drive s_i to zero and keep them there. The control inputs of the kinematic model are two, so two sliding surfaces are chosen to explicitly derives relations between states and inputs. The sliding surfaces are defined as:

$$\sigma_1 = \dot{s}_1 + k_1 s_1$$
$$\sigma_2 = s_2 + k_2 s_3$$

Computing the time derivatives:

$$\dot{\sigma}_1 = (\dot{w}y_e + w(-wx_e + v_r \sin\theta_e) - a + \dot{v}_r \cos\theta_e - v_r(w - w_r)sin\theta_e) + k_1(wy_e - u_1 + v_r \cos\theta_e)$$
$$\dot{\sigma}_2 = (-u_2 x_e + v_r \sin\theta_e) + k_2(u_2 - w_r)$$

Usually this form is chosen $\sigma_1 = \dot{s}_1 + k_1 s_1$, but it is more adapt to acceleration control, because in this case acceleration appears in the first time derivative. In this thesis a velocity control is instead considered. So σ_1 is changed:

$$\sigma_1 = s_1$$
$$\sigma_2 = s_2 + k_2 s_3$$

$$\dot{\sigma}_1 = wy_e - u_1 + v_r \cos\theta_e$$
$$\dot{\sigma}_2 = (-u_2 x_e + v_r \sin\theta_e) + k_2(u_2 - w_r)$$

The algorithm input is defined as:

$$\begin{cases} u = -\lambda|\sigma|^{1/2} sign(\sigma) + v \\ \dot{v} = -\alpha sign(\sigma) \end{cases}$$

Then to drive $\dot{\sigma_1} \to 0, \dot{\sigma_2} \to 0, \sigma_1 \to 0, \sigma_2 \to 0$ the control commands are:

$$\begin{cases} u_1 = wy_e + v_r \cos\theta_e + \lambda_1 |\sigma_1| sign(\sigma_1) + v_1 \\ \dot{v}_1 = -\alpha_1 sign(\sigma_1) \\ u_2 = \frac{v_r}{(x_e + k_2)} \sin\theta_e + \frac{k_2}{(x_e + k_2)} w_r + \lambda_2 |\sigma_2| sign(\sigma_2) + v_2 \\ \dot{v}_2 = -\alpha_2 sign(\sigma_2) \end{cases} \tag{5.13}$$

Also in this case the differentiator tuning variables used for observer is considered:

$$\begin{aligned} \lambda_1 &= 1.5\sqrt{{f_1}^+} \\ \alpha_1 &= 1.1{f_1}^+ \\ \lambda_2 &= 1.5\sqrt{{f_2}^+} \\ \alpha_2 &= 1.1{f_1}^+ \end{aligned} \tag{5.14}$$

This helps because the problem is then just to find three good values of ${f_1}^+$,${f_2}^+$ and k_2.

Chapter 6

Results

This chapter will introduce the experimental robot used, then a first simulation section will show the results obtained from the algorithms implemented in a simulink/matlab environment. The observer and the controller will be tested singularly, then a closed loop result is shown. The last section consider the experimental results obtained from the real on board implementation.

6.1 Devastator robot

The robot present two boards:

- **FRDM-K64F**: This board contains a firmware that manages sensor data and communicate with the other board via a User Data Protocol (UDP).
- **LattePandaDelta432**: this board contain the Robotic Operator System (ROS) that is a largely used robotic programming environment. All the control algorithms are defined in this board, it contain the Artificial Potential Field (APF), used to generate a reference to reach a goal and at the same time avoid obstacles. A ROS package called *robot_localization* that is an implementation of an Extended Kalman Filter. A controller to substitute. The output of the controller (PWMs) is sent via UDP to the FRDM board that interact with actuators.

For data acquisition, the robot hold the following sensors:

- **Realsense d435** a depth and RGB camera that can be used for visual odometry. It is not used for the aim of this thesis.
- **encoders** are used to measure the inertial x,y position.
- **IMUk64** an IMU (Inertial Measurement Unit) sensor that works as accelerometer and magnetometer, inside the FRDM-K64 board.
- **IMUext** an external IMU, that connects thtough an I2C protocol to the FRDM-K64 board, it works as Gyroscope and accelerometer.

The ROS system works with **nodes**, that are coded modules which communicate with **topics**. The overall already built in structure in ROS is showed in Fig. 6.1. The main goal of this thesis is to remove the `Robot_localization` algorithm (EKF) and substitute it with a super-twisting sliding mode observer to estimate θ (angle) and v the body velocity of the robot. Also the controller is substituted with a new super-twisting sliding mode controller. Changes are represented in Fig. 6.2.

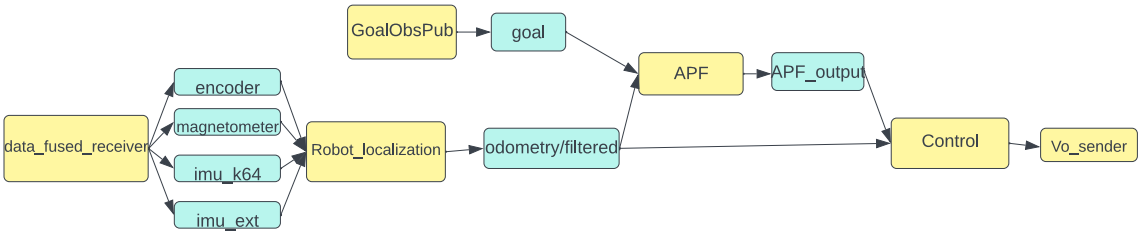

Figure 6.1. The yellow blocks are teh nodes, the blue one are the topics

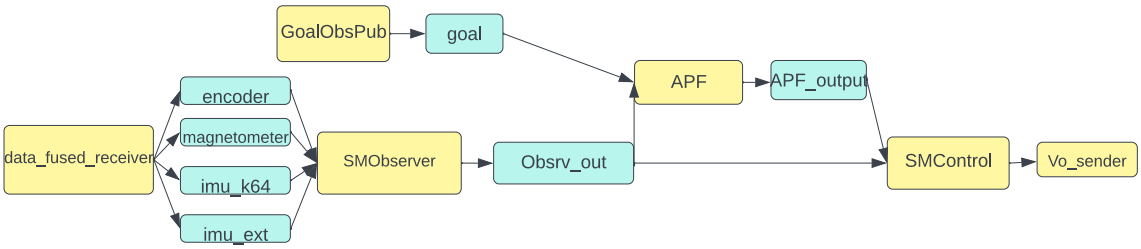

Figure 6.2. This is the final block diagram idea, the observer substitute the `Robot_localization` node and the controller is substituted with a new Sliding Mode Controller

6.2 Simulated results

For the simulations, a model of the robot dynamic was used. It contains three main modules, an APF to define a trajectory reference, a plant of the robot, simulated with a system identification module and a sensor module to create a more realistic environment. The sensor module applies a random Gaussian noise to the input variables with the following variance values (mean value equal to zero):

$$\begin{bmatrix} x \\ y \\ w \\ a \end{bmatrix} = \begin{bmatrix} 1e-04 \\ 1e-04 \\ 1e-06 \\ 1e-06 \end{bmatrix}$$

The **Controller** is implemented as in the Eq. 5.13, of the previous chapter:

$$\begin{cases} u_1 = wy_e + v_r \cos\theta_e + \lambda_1 |\sigma_1| sign(\sigma_1) + v_1 \\ \dot{v}_1 = -\alpha_1 sign(\sigma_1) \\ u_2 = \frac{v_r}{(x_e+k)} \sin\theta_e + \frac{k_2}{(x_e+k)} w_r + \lambda_2 |\sigma_2| sign(\sigma_2) + v_2 \\ \dot{v}_2 = -\alpha_2 sign(\sigma_2) \end{cases} \tag{6.1}$$

The tuning variables are chosen as:

$$\begin{aligned} \lambda_1 &= 1.5\sqrt{0.001} \\ \alpha_1 &= 1.1(0.001) \\ \lambda_2 &= 1.5\sqrt{0.005} \\ \alpha_2 &= 1.1(0.005) \\ k &= 0.8 \end{aligned}$$

The controller is tested to drive the robot from an initial position (0,0) to a desired goal position (4,8), while avoiding some obstacles. The trajectory results are showed in Fig. 6.3. The robot reach the final destination while avoiding the obstacles. The velocity

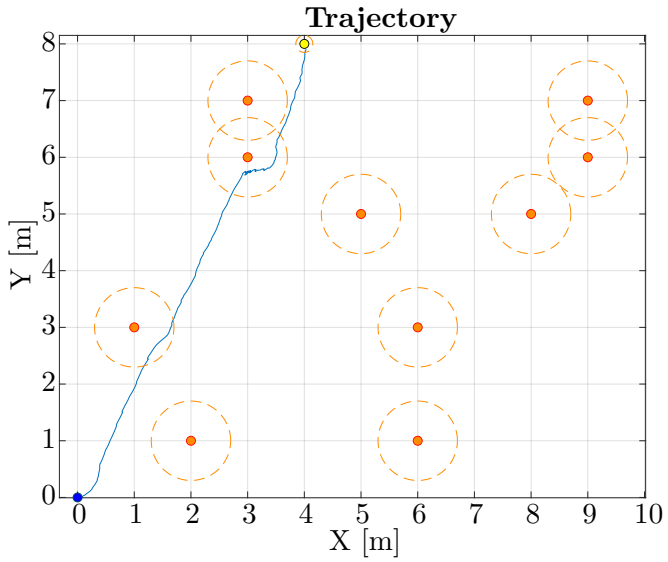

Figure 6.3. The blue line is the robot trajectory, while the orange point are the obstacles and the circunference is the obstacles range of influence for the APF algorithm

and angle reference chase is showed in Fig. 6.4. The chase of references is not perfect probably because the APF gives very brusque reference changes that makes it hard to the controller to align. A second simulation shows the effect of the tuning variables, this

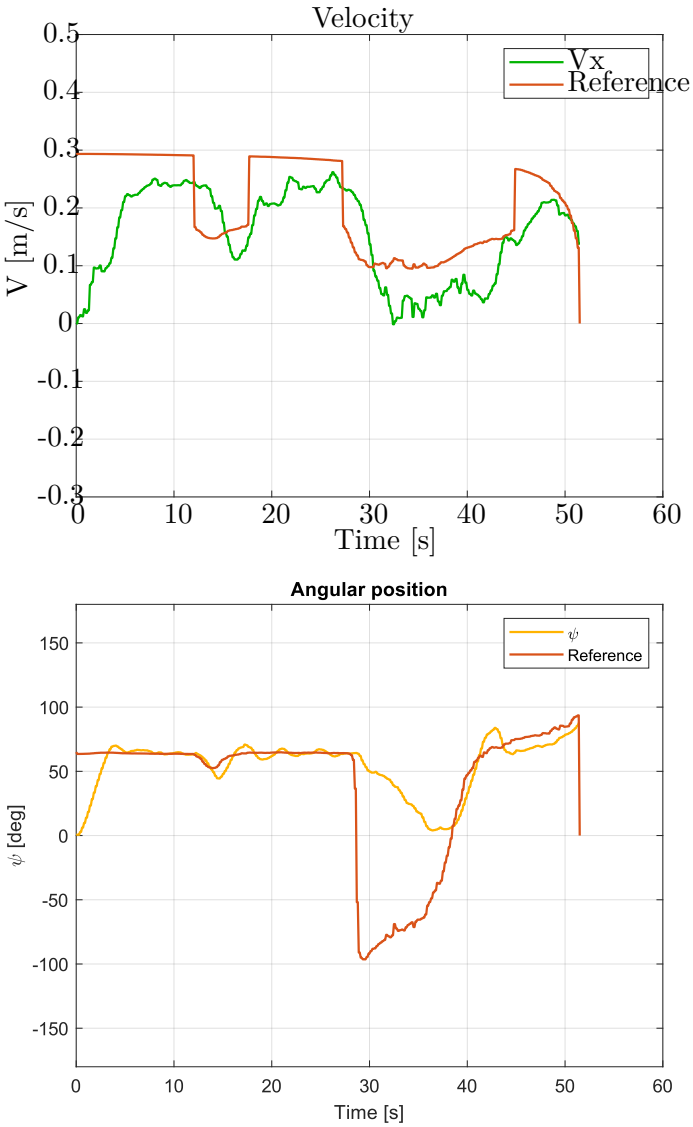

Figure 6.4. In the first image V_x is the linear velocity of the robot, while the second shows the angle trajectory chase

time the values are set as:

$$\lambda_1 = 1.5\sqrt{0.01}$$
$$\alpha_1 = 1.1(0.01)$$
$$\lambda_2 = 1.5\sqrt{0.15}$$
$$\alpha_2 = 1.1(0.15)$$
$$k = 0.8$$

The values λ and α are increased, the results shows that the goal is still reached but with a more oscillating trajectory Fig. 6.5. Those are limit values, a further increase on

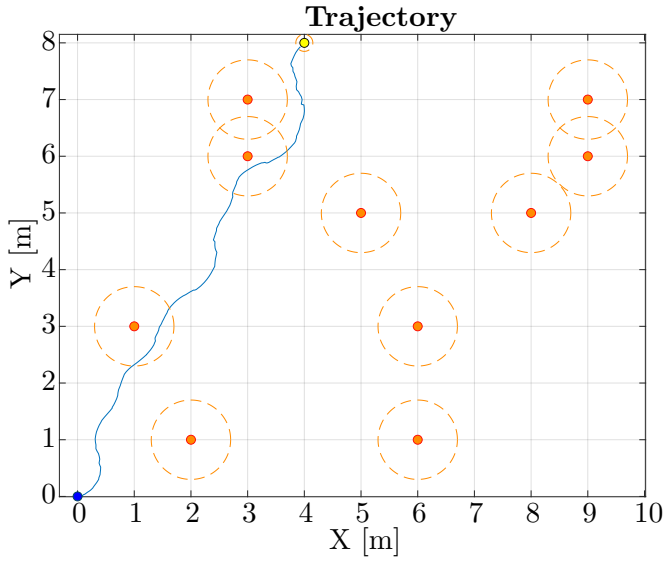

Figure 6.5. A more uncontrolled dynamic with respect to the first simulation.

tuning variables does not allow to reach the final goal. As already explained, a non-linear closed loop system is hard to analyze. For this reason the **Observer** implementation approach, was to first of all test it outside of the closed loop, then to run it in feedback with the APF and Controller. As explained in the previous chapter, the **Observer** is implemented as:

$$\begin{cases} \dot{\hat{x}} = \hat{v}_x + \lambda_{11}|x - \hat{x}|^{1/2} sign(x - \hat{x}) \\ \dot{\hat{v}}_x = a\cos\hat{\theta} - \sqrt{\hat{v}_x{}^2 + \hat{v}_y{}^2}\, w \sin\hat{\theta} + \lambda_{12} sign(x - \hat{x}) \\ \dot{\hat{y}} = \hat{v}_y + \lambda_{21}|y - \hat{y}|^{1/2} sign(y - \hat{y}) \\ \dot{\hat{v}}_y = a\sin\hat{\theta} + \sqrt{\hat{v}_x{}^2 + \hat{v}_y{}^2}\, w \cos\hat{\theta} + \lambda_{22} sign(y - \hat{y}) \end{cases} \tag{6.2}$$

$$\begin{cases} \dot{\hat{z}}_1 = \sqrt{\hat{v}_x{}^2 + \hat{v}_y{}^2}\hat{C} + \lambda_{z1}|x - \hat{z}_1|^{1/2} sign(x - \hat{z}_1) \\ \dot{\hat{C}} = -w\hat{S} + \alpha_1 sign(x - \hat{z}_1) \\ \dot{\hat{z}}_2 = \sqrt{\hat{v}_x{}^2 + \hat{v}_y{}^2}\hat{S} + \lambda_{z2}|y - \hat{z}_2|^{1/2} sign(y - \hat{z}_2) \\ \dot{\hat{S}} = w\hat{C} + \alpha_2 sign(y - \hat{z}_2) \end{cases} \tag{6.3}$$

$$\hat{\theta} = \arctan(S, C)$$
$$\hat{v} = \sqrt{\hat{v}_x{}^2 + \hat{v}_y{}^2}$$

Where $\hat{v}$ and $\hat{\theta}$ are the two estimated states. The sensors variables a, w, x, y are simulated with a frequency of 20Hz, while the observer works at a frequency of 100Hz. The first simulation shows the results with the following initial conditions and tuning variables:

$$\begin{bmatrix} \hat{x} \\ \hat{y} \\ v_x \\ \hat{v}_y \\ \hat{C} \\ \hat{S} \\ \hat{z}_1 \\ \hat{z}_2 \\ \hat{\theta} \end{bmatrix} = \begin{bmatrix} 0.2 \\ 0.2 \\ 0.2 \\ 0.2 \\ 1 \\ 1 \\ 0.1 \\ 0.1 \\ 0.2 \end{bmatrix}$$

$$\begin{bmatrix} \lambda_{11} \\ \lambda_{12} \\ \lambda_{21} \\ \lambda_{22} \\ \lambda_{z1} \\ \alpha_1 \\ \lambda_{z2} \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} 1.5\sqrt{0.06} \\ 1.1(0.06) \\ 1.5\sqrt{0.06} \\ 1.1(0.06) \\ 1.5\sqrt{0.3} \\ 1.1(0.3) \\ 1.5\sqrt{0.3} \\ 1.1(0.3) \end{bmatrix}$$

The figure 6.6 shows the velocity estimation with respect $Vreal$ that represent the real velocity dynamics. Initial states of the observer are different from the real ones, so it takes some time for the algorithm to stabilize to the real states. Then the Fig. 6.7 shows the angle estimation and Fig 6.8 the known states (x,y) estimation. The idea is that taking initial conditions different from the real ones, the integrative model part won't drive to the real dynamics. This can be visualize in Fig. 6.9, that consider the same previous case but with tuning variables equal to zero, this mean the observer is "turned off". As can be seen the two dynamics are translated, and the (x,y) position reconstruction fails 6.10, so the idea behind the observer algorithm is to chase the known variables (x,y) to correct the integrative part of the model and drive it back to the real dynamic. To reduce the

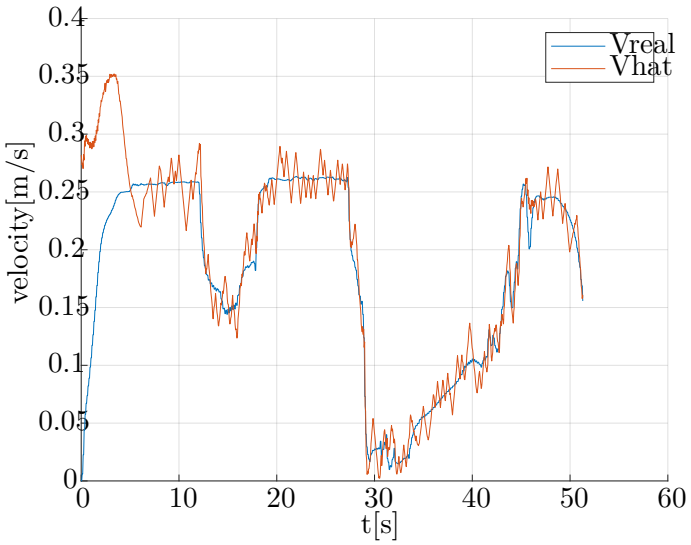

Figure 6.6. The estimated velocity reaches the real one.

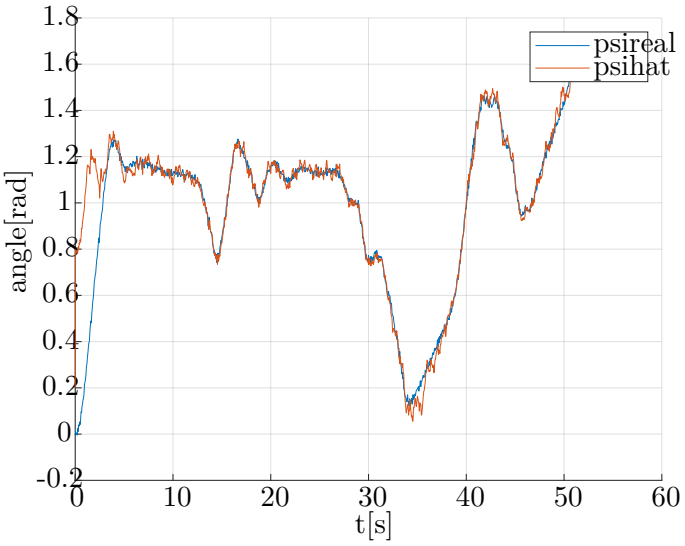

Figure 6.7. After some time the estimation follow the real state.

convergence time of the figure 6.6, it is possible to increase the tuning variables as:

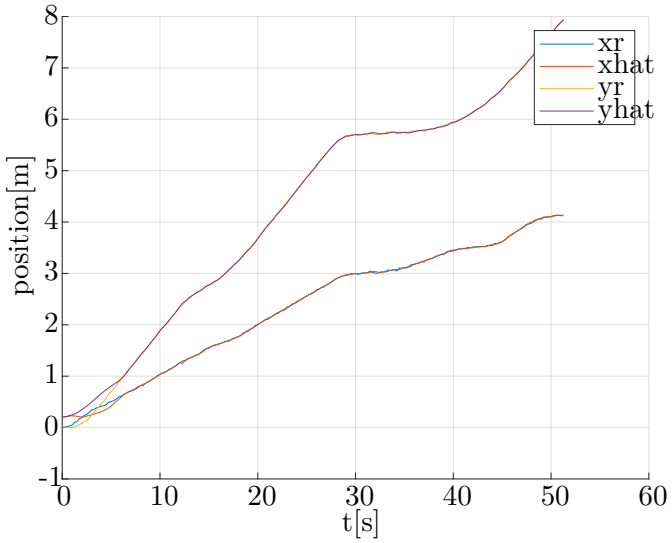

Figure 6.8. The inertial position estimation reaches the real values.

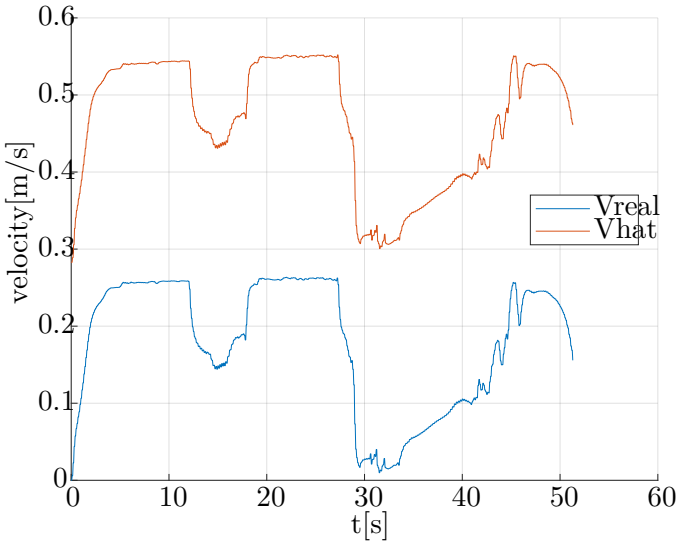

Figure 6.9. This figure shows what happen without using the observer, with non-zero initial conditions. The role of the observer is then to drive Vhat to the real function.

$$\begin{bmatrix} \lambda_{11} \\ \lambda_{12} \\ \lambda_{21} \\ \lambda_{22} \\ \lambda_{z1} \\ \alpha_1 \\ \lambda_{z2} \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} 1.5\sqrt{0.3} \\ 1.1(0.3) \\ 1.5\sqrt{0.3} \\ 1.1(0.3) \\ 1.5\sqrt{0.5} \\ 1.1(0.5) \\ 1.5\sqrt{0.5} \\ 1.1(0.5) \end{bmatrix}$$

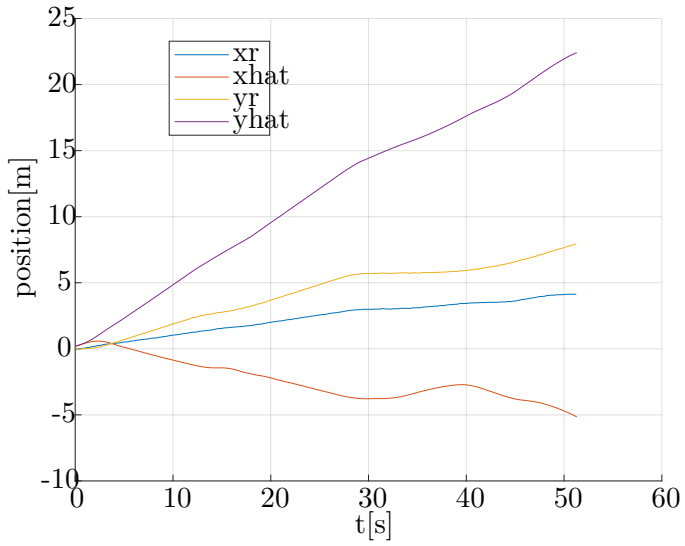

Figure 6.10. With non-zero initial conditions the solutions of the dynamics evolves differently.

Unfortunately this increases the chattering phenomenon, Figs. 6.11, 6.12. The oscil-

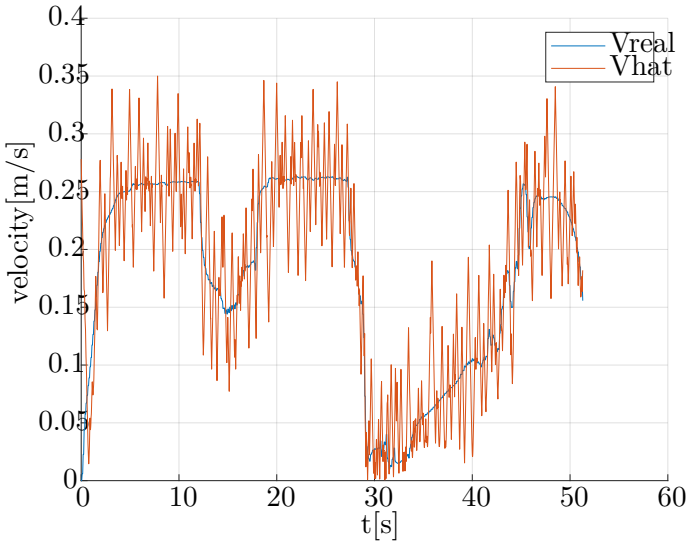

Figure 6.11. Increasing tuning values force a faster convergence, but increases the chattering.

lations (chattering) can be reduced using low-pass filters or by changing the switching $sign(\cdot)$ function with the tangent hyperbolic function $\tanh(\cdot)$, and by defining a new constant n inside the function $\tanh(n * (e))$. Using $n = 10$ in the previous example this is the new velocity estimation Fig. 6.13. The next simulation consider The whole dynamic with

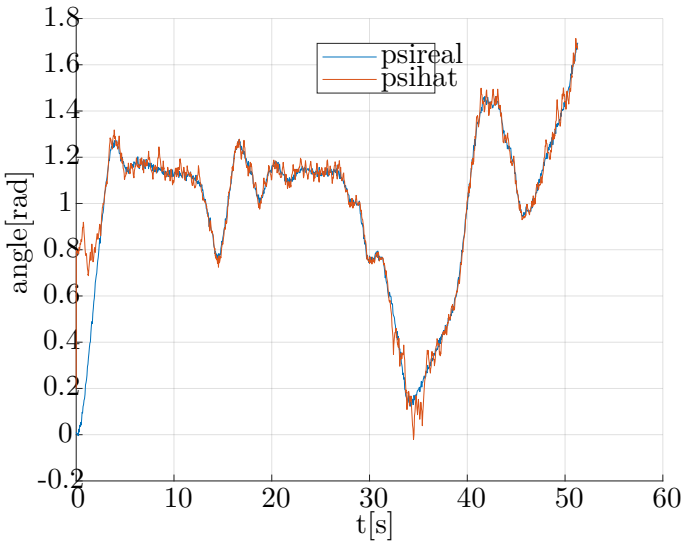

Figure 6.12. higher oscillations from previous simulation.

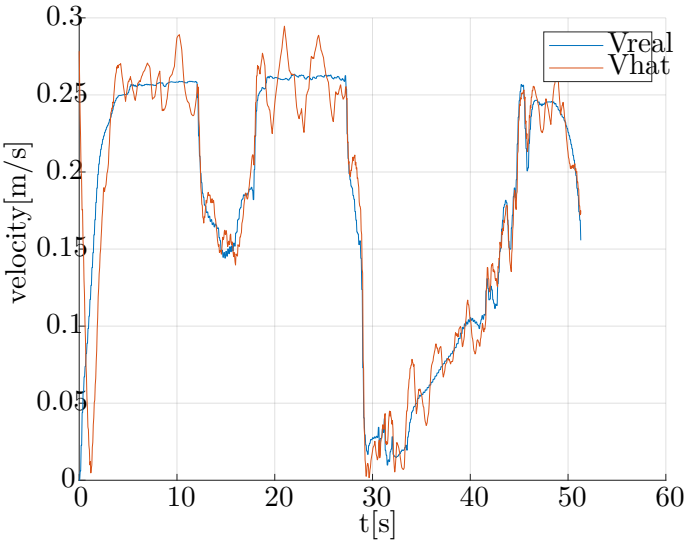

Figure 6.13. Oscillations are smaller, but quality of estimation is worse.

the Observer in feedback with APF and Controller, with initial conditions and tuning variables of the observer chosen as:

$$\begin{bmatrix} \hat{x} \\ \hat{y} \\ v_x \\ \hat{v}_y \\ \hat{C} \\ \hat{S} \\ \hat{z}_1 \\ \hat{z}_2 \\ \hat{\theta} \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 1 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}$$

$$\begin{bmatrix} \lambda_{11} \\ \lambda_{12} \\ \lambda_{21} \\ \lambda_{22} \\ \lambda_{z1} \\ \alpha_1 \\ \lambda_{z2} \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} 1.5\sqrt{0.03} \\ 1.1(0.03) \\ 1.5\sqrt{0.03} \\ 1.1(0.03) \\ 1.5\sqrt{0.1} \\ 1.1(0.1) \\ 1.5\sqrt{0.1} \\ 1.1(0.1) \end{bmatrix}$$

The tuning variables of the controller are the same of the first example. Despite the non-zero initial conditions, the observer's states reaches the real states and the robot arrive at the final goal in position (10,7) while avoiding all obstacles. The results are in Figs. 6.14, 6.15.

6.3 Experimental results

The Matlab/simulink models are translated directly into python, the translated code is showed in the Appendix section. A first order Low-pass filter was applied to the acceleration and angular velocity input states (from IMU). Firstly mean value and variance are evaluated from sensor data sampling. The obtained values are:

$$\begin{aligned} Var(a) &= 1.5e-04 \\ Mean(a) &= -0.2924 \\ Var(w) &= 4.31e-06 \\ Mean(w) &= -0.0170 \end{aligned}$$

The acceleration is more "problematic" with higher variance and a Mean value not negligible. Those values was used to define a very simple low-pass filter, defined as:

$$\begin{aligned} a_f &= (1-\alpha_1)a_{old} + \alpha_1 a_{new} \\ w_f &= (1-\alpha_2)w_{old} + \alpha_2 w_{new} \end{aligned}$$

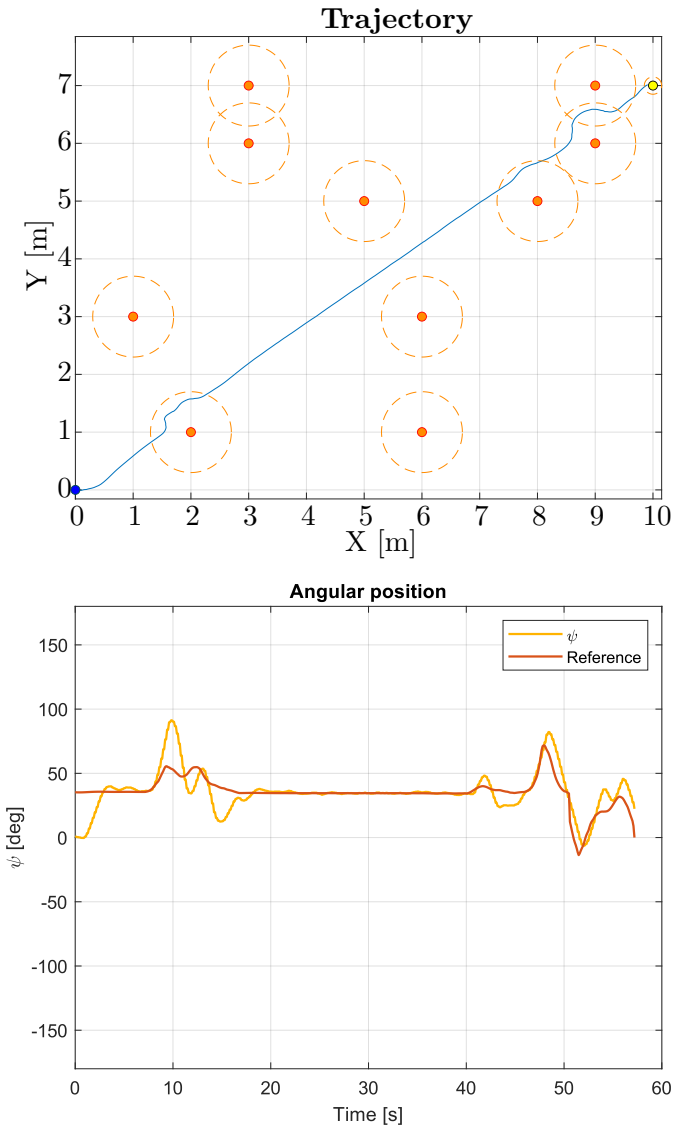

Figure 6.14. Caption

The filtering results are showed in Fig. 6.16. As should be clear the data refers to the robot stopped. For the **Controller** test a very simple experiment has been performed. The goal is set to the position (1,1) (meters), and no obstacles are considered. The results refers to the data obtained from the encoders. The next figure shows the trajectory followed by the robot, and it is made a comparison between real case and the simulative one Fig. 6.17. The robot doesn't get to the final point becasuse the APF has a tolerance value of 10cm, this means that when the robot get inside a 10cm ray from the goal it stops. But is clear the tendency of the robot to reach the final point, the experiment can be considered successfully completed. The states estimation of the **Observer** refers to

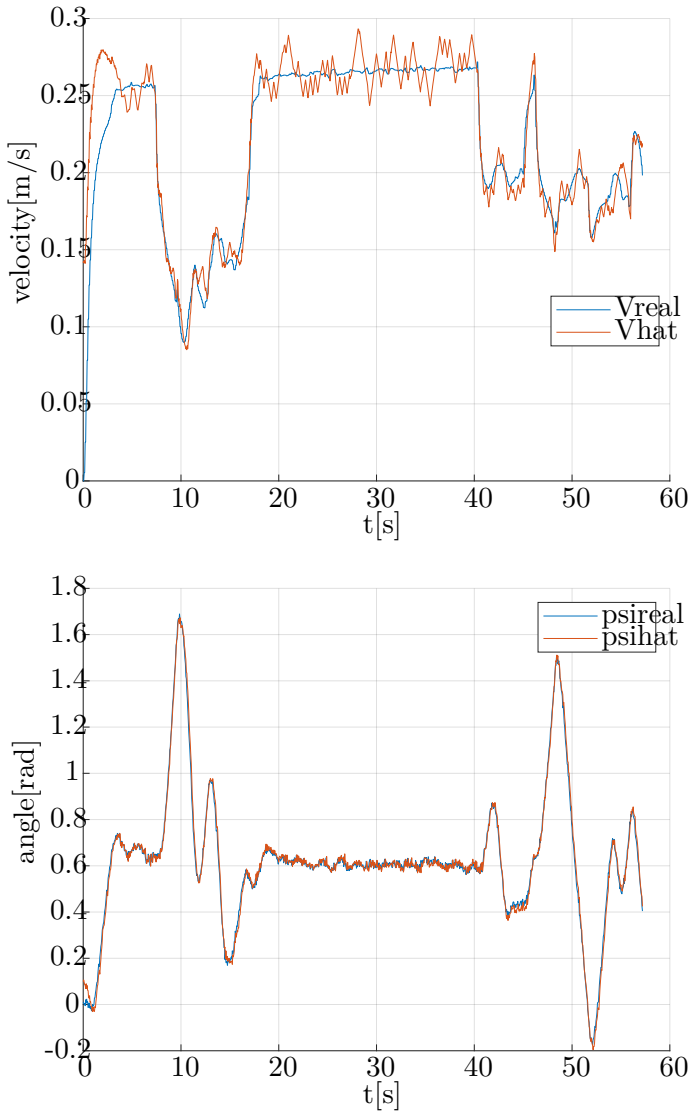

Figure 6.15. Caption

the same experiment. Since no sensor for body velocity and orientation was available on the robot, the observer estimation is compared with an Extended Kalman Filter already built in the robot. The following figure shows the comparison results 6.18. The $sign(\cdot)$ function was substituted with the $\tanh(\cdot)$ function to try reduce chattering. The velocity estimations are comparable, the observer unfortunately present the chattering problem. For the angle the two estimation differ remarkably. But analyzing the trajectory on Fig. 6.17, assuming as zero degree the x axis direction, the observer estimation seems to be more realistic.

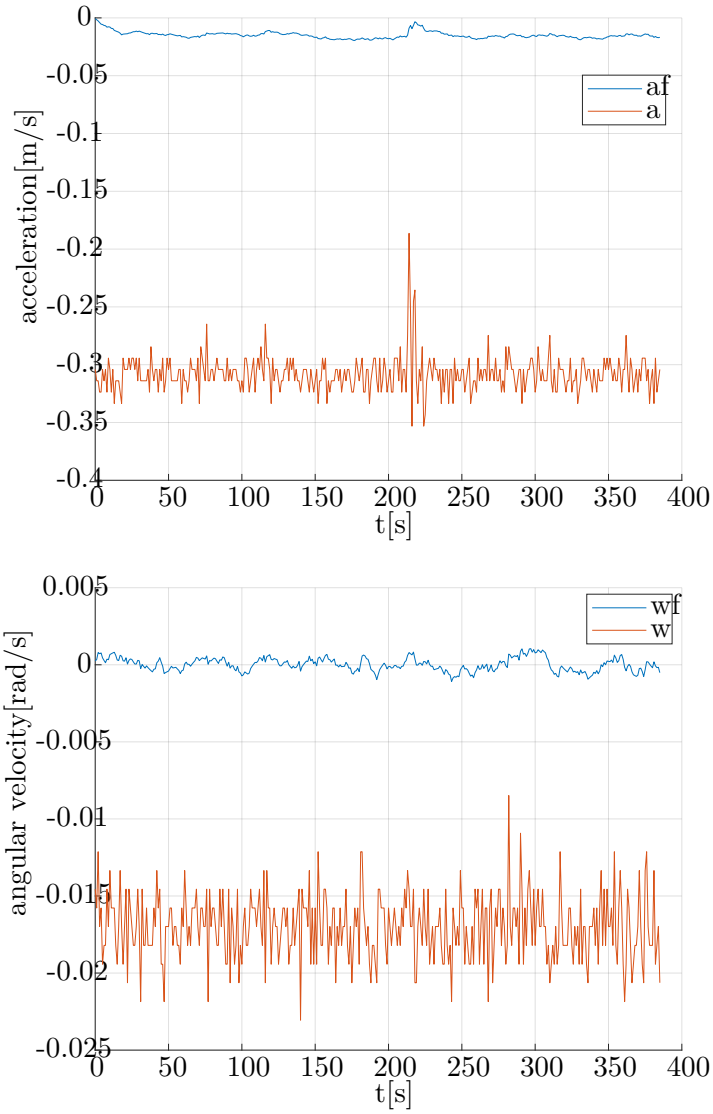

Figure 6.16. w_f and a_f are the filtered values, while a and w are those from the sensors. The oscillations are reduced and the functions stay close to zero.

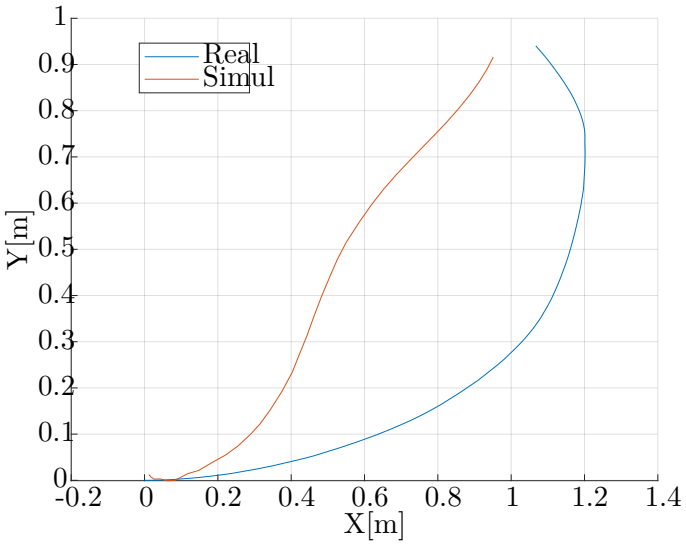

Figure 6.17. Caption

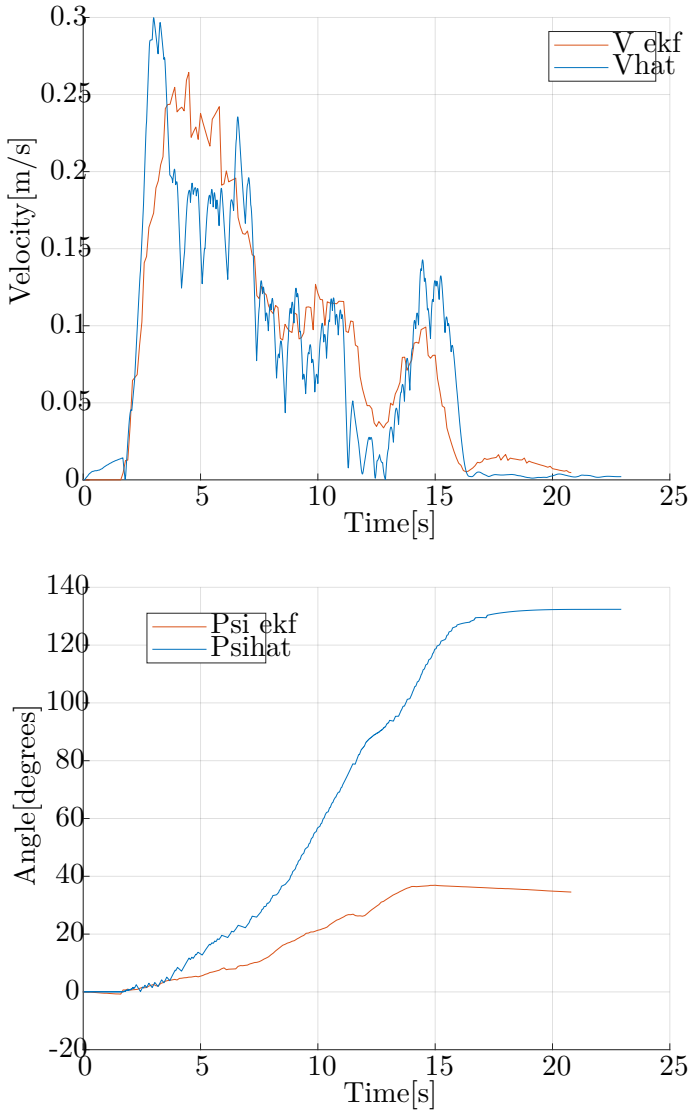

Figure 6.18. Comparison between Observer and EKF estimations. On top body velocity of the robot. Bottom the orientation estimation.

Chapter 7

Conclusions

The aim of this work was to challenge the robustness of the sliding mode technique in a real mobile robotic context, in particular it was used to develop two important modules in the robotic context, the **Controller** and the **Observer**. The two algorithms showed remarkable results in a simulated environment. The experimental results proof the algorithms work also in the real context, but with more evident fragility. It is important to emphasize that the algorithms are applied to a very simple and approximate kinematic model, the unicycle. The model does not consider any force involved in the dynamic such as friction with ground or slipping. This approximation, and data noise makes it challenging to obtain very robust solutions. To reduce the noise a first order low-pass filter is implemented. The Observer was designed with the aim of substitute the already built in Extended Kalman Filter (EKF). Unfortunately due to some technical problems, the work was not completed, the observer was tested in the robot, but not in closed loop with the controller. This could be one of the reasons why the controller performance wasn't so good. Analyzing the two figures 6.17 and 6.18, is clear that the estimated angle from the EKF does not match up with the trajectory, this means that, probably the controller try to drive to zero the reference angle from the APF while the EKF estimation opposes. This could be why the controller draw a large curve to get to the goal. The angular estimation of the Observer, instead, seems to be more coherent with the encoders trajectory, but is hard, without having tried the feedback dynamic, to conclude that this observer is a better candidate. Overall the sliding mode technique can be considered a very strong option for vehicle control systems. It is very versatile, and easy to implement also for more complex mathematical models. As cons, the problem of chattering could become instead a great limitation in more precise context, such as surgical robots where any sort of small oscillation must be avoided. A further work could consider the application of previous techniques to a dynamical model of the robot, that consider stronger mathematical relations. Furthermore would be interesting to test the controller in an environment with obstacles.

Appendix A

Observer Code

The observer in this case is showed with message filters implementation. It means all the sensors are assumed to work at the same velocity. For a stronger consistency on data access, some locks are used. The idea is that the observer function constantly iterates at 100Hz, when sensors data are collected, an interrupt is triggered and the callback function is called.

```
#!/usr/bin/env python3

from multiprocessing import Lock
import threading
import message_filters
import rospy
import math
import numpy
from prova_one.msg import Obsrv_cmd
from sensor_msgs.msg import Imu
from nav_msgs.msg import Odometry

class Obsrv():
    def __init__(self):
        rospy.init_node("SMObserver",anonymous = True)
        self.hz = 100
        self.rate = rospy.Rate(self.hz)
        self.Init_var()
        sub_Enc = message_filters.Subscriber("encoder",Odometry,queue_size = 10)
        sub_ImuExt = message_filters.Subscriber("imu_ext",Imu,queue_size = 10)
        sub_Imuk64 = message_filters.Subscriber("imu_k64",Imu,queue_size = 10)
        ts = message_filters.ApproximateTimeSynchronizer([sub_Enc, sub_ImuExt,sub_Imuk64],queue_size=10,slop=0.01)
        ts.registerCallback(self.callback)
        pub_Obsrv = rospy.Publisher("Obsrv2_out",Odometry,queue_size = 10)
        while not rospy.is_shutdown():
            self.Observer(pub_Obsrv)
            self.rate.sleep()

```

```
29          rospy.spin()
30
31      def Observer(self,pub_Obsrv):
32          F = 0.18
33          F2 = 0.35
34          h11 = 1.5*math.sqrt(F)
35          h21 = 1.1*F
36          h12 = 1.5*math.sqrt(F2)
37          h22 = 1.1*F2
38          F1 = 0.3
39          hz11 = 1.5*math.sqrt(F1)
40          hz21 = 1.1*F
41          hz12 = 1.5*math.sqrt(F1)
42          hz22 = 1.1*F
43          Ts = 0.01
44          n1 = 250
45          n2 = 150
46          self.lock.acquire()
47          ## Vxhat
48          zhat1_new = self.zhat1 + Ts*(self.Vxhat + h11*math.sqrt(abs(self.
    xreal-self.zhat1))*numpy.tanh(n1*(self.xreal-self.zhat1)))
49          Vxhat_new = self.Vxhat + Ts*(self.a*math.cos(self.psihat) -self.
    Vhat*self.w*math.sin(self.psihat) + h21*numpy.tanh(n1*(self.xreal-self
    .zhat1)))
50
51          ## Vyhat
52          zhat2_new = self.zhat2 + Ts*(self.Vyhat + h12*math.sqrt(abs(self.
    yreal-self.zhat2))*numpy.tanh(n2*(self.yreal-self.zhat2)))
53          Vyhat_new = self.Vyhat + Ts*(self.a*math.sin(self.psihat) + self.
    Vhat*self.w*math.cos(self.psihat) + h22*numpy.tanh(n2*(self.yreal-self
    .zhat2)))
54
55
56          ## Cos(psi)
57          xhat_new = self.xhat + Ts*(self.Vhat*self.Chat + hz11*math.sqrt(
    abs(self.xreal-self.xhat))*numpy.sign((self.xreal-self.xhat)))
58          Chat_new = self.Chat + Ts*(-self.w*self.Shat + hz21*numpy.sign((
    self.xreal-self.xhat)))
59
60          ## Sin(psi)
61          yhat_new = self.yhat + Ts*(self.Vhat*self.Shat + hz12*math.sqrt(
    abs(self.yreal-self.yhat))*numpy.sign((self.yreal-self.yhat)))
62          Shat_new = self.Shat + Ts*(self.w*self.Chat + hz22*numpy.sign((
    self.yreal-self.yhat)))
63
64
65          self.Vhat = math.sqrt(Vxhat_new**2 + Vyhat_new**2)
66          self.psihat = numpy.arctan2(Shat_new,Chat_new)
67
68          self.Vxhat = Vxhat_new
69          self.Vyhat = Vyhat_new
70          self.zhat1 = zhat1_new
71          self.zhat2 = zhat2_new
72          self.Chat = Chat_new
73          self.Shat = Shat_new
```

```
74          self.xhat = xhat_new
75          self.yhat = yhat_new
76
77          self.msg.twist.twist.linear.x = Vxhat_new
78          self.msg.twist.twist.linear.z = self.Vhat
79          self.msg.twist.twist.linear.y = Vyhat_new
80          self.msg.pose.pose.position.x = zhat1_new
81          self.msg.pose.pose.position.y = zhat2_new
82          self.msg.pose.pose.orientation.z = self.psihat
83          pub_Obsrv.publish(self.msg)
84          self.lock.release()
85
86      def callback(self,sub_Enc,sub_ImuExt,sub_Imuk64):   ##posso sostituire
        con la camera, che usa rtabmap
87          self.lock.acquire()
88          alpha1 = 0.1
89          alpha2 = 0.1
90
91          self.xreal = sub_Enc.pose.pose.position.x
92          self.yreal = sub_Enc.pose.pose.position.y
93
94          a_new = sub_Imuk64.linear_acceleration.x + 0.2924
95          w_new = sub_ImuExt.angular_velocity.z + 0.0170
96
97          self.a = (1-alpha1)*self.a_old + alpha1*a_new
98          self.w = (1-alpha2)*self.w_old + alpha2*w_new
99
100         self.a_old = self.a
101         self.w_old = self.w
102         self.lock.release()
103
104
105     def Init_var(self):
106
107         self.lock = Lock()
108         self.a_old = 0
109         self.w_old = 0
110         self.xreal = 0
111         self.yreal = 0
112         self.Vxhat = 0
113         self.Vyhat = 0
114         self.xhat = 0
115         self.w = 0
116         self.a = 0
117         self.yhat = 0
118         self.zhat1 = 0
119         self.zhat2 = 0
120         self.psihat = 0
121         self.Vhat = 0
122         self.Chat = 1
123         self.Shat = 0
124         self.msg = Odometry()
125
126
127 if __name__ == '__main__':
```

```
128     try:
129         Obsrv()
130     except rospy.ROSInterruptException:
131         pass
```

Appendix B

Controller Code

```
    #py_controll.py


from numpy import linalg as LA
import numpy as np
import math
import queue
import rospy
import message_filters
from nav_msgs.msg import Odometry
from geometry_msgs.msg import Pose
from sensor_msgs.msg import Imu
from visual_odometry.msg import PWM_cmd,APF_cmd
from scipy.spatial.transform import Rotation as R



class SMC():


    def __init__(self):
        rospy.init_node('SMC',anonymous=True)
        self.rate = rospy.Rate(10) # 10hz
        self.init()
        sub_APFout = message_filters.Subscriber("/APF_output", APF_cmd,
    queue_size = 10)
        sub_goal = message_filters.Subscriber("/goal", Pose, queue_size =
    10)
        sub_odometry    = message_filters.Subscriber("/odometry/filtered",
    Odometry, queue_size = 10)
        self.PIDpub = rospy.Publisher('SMC_super_cmd', PWM_cmd, queue_size
    =10)
        self.msg = PWM_cmd()
        ts = message_filters.ApproximateTimeSynchronizer([sub_APFout,
    sub_odometry,sub_goal], queue_size=10, slop=0.5, allow_headerless=True
    )
        ts.registerCallback(self.Control)
        rospy.spin()
```

```
33
34      def get_rotation(self,Odom):
35          orientation_q = Odom.pose.pose.orientation
36          orientation_list = [orientation_q.x, orientation_q.y,
    orientation_q.z, orientation_q.w]
37          r = R.from_quat(orientation_list)
38          EuAn = r.as_euler('zyx', degrees=False)
39          return EuAn
40
41      def init(self):
42          self.v1 = 0
43          self.v2 = 0
44          self.psid_prec = 0
45
46
47      def Control(self,sub_APFout, sub_odometry, sub_goal):
48          k = 0.8
49          Ts = 0.1
50          F1 = 0.01
51          F2 = 0.05
52          h1 = 1.5*math.sqrt(F1)
53          alfa1 = 1.1*F1
54          h2 = 1.5*math.sqrt(F2)
55          alfa2 = 1.1*F2
56
57          vx = sub_odometry.twist.twist.linear.x
58          vy = sub_odometry.twist.twist.linear.y
59          xr = sub_odometry.pose.pose.position.x
60          yr = sub_odometry.pose.pose.position.y
61          wr = sub_odometry.twist.twist.angular.z
62          vd = sub_APFout.Vref
63          psid = sub_APFout.Psiref
64          wd = (psid-self.psid_prec)/0.1
65
66          xd = xr + vd*0.1*math.cos(psid)
67          yd = yr + vd*0.1*math.sin(psid)
68          [psir, _, _] = self.get_rotation(sub_odometry)
69          vr = (vx+vy)/(math.cos(psir)+math.sin(psir))
70
71
72
73
74          #errors definition
75
76          xe=math.cos(psir)*(xd-xr)+math.sin(psir)*(yd-yr)
77          ye=-math.sin(psir)*(xd-xr)+math.cos(psir)*(yd-yr)
78          psie=psid-psir
79
80          if math.fabs(psie) > math.pi:
81              psie=psie-2*math.pi*np.sign(psie)
82
83          # smc - surfaces
84          s1=xe
85          s2=ye + k*psie
86
```

```
87          v1_new = self.v1 + Ts*(-alfa1*np.sign(s1))
88          U1 = -h1*math.sqrt(abs(s1))*np.sign(s1) + v1_new
89          u1 = -U1 + wr*ye + vd*math.cos(psie)
90
91
92          v2_new = self.v2 + Ts*(-alfa2*np.sign(s2))
93          U2 = -h2*math.sqrt(abs(s2))*np.sign(s2)
94          u2 = -U2 + vd/(xe + k)*math.sin(psie) + k/(xe + k)*wd
95
96
97
98          if u1>0.45:
99              u1 = 0.45
100
101         if u1<0:
102             u1=0
103
104         if u2 > 0.1:
105             u2 = 0.1
106
107         if u2 < -0.1:
108             u2 = -0.1
109
110         PWM_R = 20000*(u1+u2)
111         PWM_L = 20000*(u1-u2)
112
113         if PWM_R>20000:
114             PWM_R = 20000
115
116         if PWM_R<-20000:
117             PWM_R = -20000
118
119         if PWM_L>20000:
120             PWM_L = 20000
121
122         if PWM_L<-20000:
123             PWM_L = -20000
124
125         self.psid_prec = psid
126
127         self.msg.u1 = u1
128         self.msg.u2 = u2
129         self.msg.PWM_left = PWM_L
130         self.msg.PWM_right = PWM_R
131         self.PIDpub.publish(self.msg)
132
133
134 if __name__ == '__main__':
135     try:
136         SMC()
137     except rospy.ROSInterruptException:
138         pass
```

Bibliography

[1] F. Rubio, F. Valero, and C. Llopis-Albert, "A review of mobile robots: Concepts, methods, theoretical framework, and applications," *International Journal of Advanced Robotic Systems*, vol. 16, no. 2, pp. 172988141983959–14, 2019.

[2] K. Adamiak and A. Bartoszewicz, "Novel power-rate reaching law for quasi-sliding mode control," *Energies*, vol. 15, no. 15, 2022.

[3] Y. Shtessel, C. Edwards, L. Fridman, A. Levant, *et al.*, *Sliding mode control and observation*, vol. 10. Springer, 2014.

[4] T. Floquet and J. P. Barbot, "Super twisting algorithm-based step-by-step sliding mode observers for nonlinear systems with unknown inputs," *International Journal of Systems Science*, vol. 38, no. 10, pp. 803–815, 2007.

[5] M. Indri, "Sistemi robotici," *Master Degree Computer Engineering*, 2021-2022.

[6] S. Malan, "Automotive control systems," *Master Degree Computer Engineering*, 2022.

[7] C. Novara, "Nonlinear control and aerospace applications," *Master Degree Computer Engineering*, 2022.

[8] A. M. Hassan and H. E. Taha, *Geometric Nonlinear Controllability Analysis for Airplane Flight Dynamics*.

[9] "Nonlinear observability via koopman analysis: Characterizing the role of symmetry," *Automatica (Oxford)*, vol. 124, pp. 109353–, 2021.

[10] K. J. Åström and B. Wittenmark, "Computer-controlled systems: Theory and design," 1984.

[11] V. Utkin, "Variable structure systems with sliding modes," *IEEE Transactions on Automatic Control*, vol. 22, no. 2, pp. 212–222, 1977.

[12] J. Davila, L. Fridman, and A. Levant, "Second-order sliding-mode observer for mechanical systems," *IEEE transactions on automatic control*, vol. 50, no. 11, pp. 1785–1789, 2005.

[13] N. M'Sirdi, A. Rabhi, L. Fridman, J. Davila, and Y. Delanne, "Second order sliding-mode observer for estimation of vehicle dynamic parameters," *Int. J. Vehicle Design Int. J. Vehicle Design*, vol. 48, pp. 190–207, 07 2008.

[14] Zs. Lendek, R. Babuška, and B. De Schutter, "Stability of cascaded fuzzy systems and observers," *IEEE Transactions on Fuzzy Systems*, vol. 17, pp. 641–653, June 2009.