UNIVERSITÀ DEL SALENTO

Facoltà di Ingegneria

Corso di Laurea in Computer Engineering

Project documentation

for the course

ESTIMATION AND DATA ANALYSIS WITH APPLICATIONS

Position and velocity estimation of a moving vehicle with respect to a reference obstacle using Kalman filter

Professor

Prof. Daniela De Palma

Student

Lamberto Preite (20068180)

Contents

Figures

1. Introduction

The purpose of this project is to develop a robotic vehicle that can move and take measurements of its position and velocity with reference to a certain object (typically a wall or a nonmoving obstacle). The vehicle can move freely via direct command (provided via IR remote control) or autonomously, in a straight line, to a provided position. The design involves movement on a flat surface.

The main objective is to implement a robust state estimation algorithm that can filter out noise from the noisy measurements. The state is composed by position and velocity. This process requires the definition of a dynamic model that define the evolution of the state vector in relation to certain physical quantities and the measurements. After this definition and with some assumptions it is possible to apply the Kalman filter algorithm. More details will be provided in Chapter 4.

The vehicle is equipped with two sensors. An ultrasonic sensor takes the distance measure between the vehicle and the reference obstacle. An optical sensor, used in combination with a wheel encoder, can measure the number of light pulses per second, useful for counting the round per second of the wheel. More assumptions and details will be provided in Chapter 3.

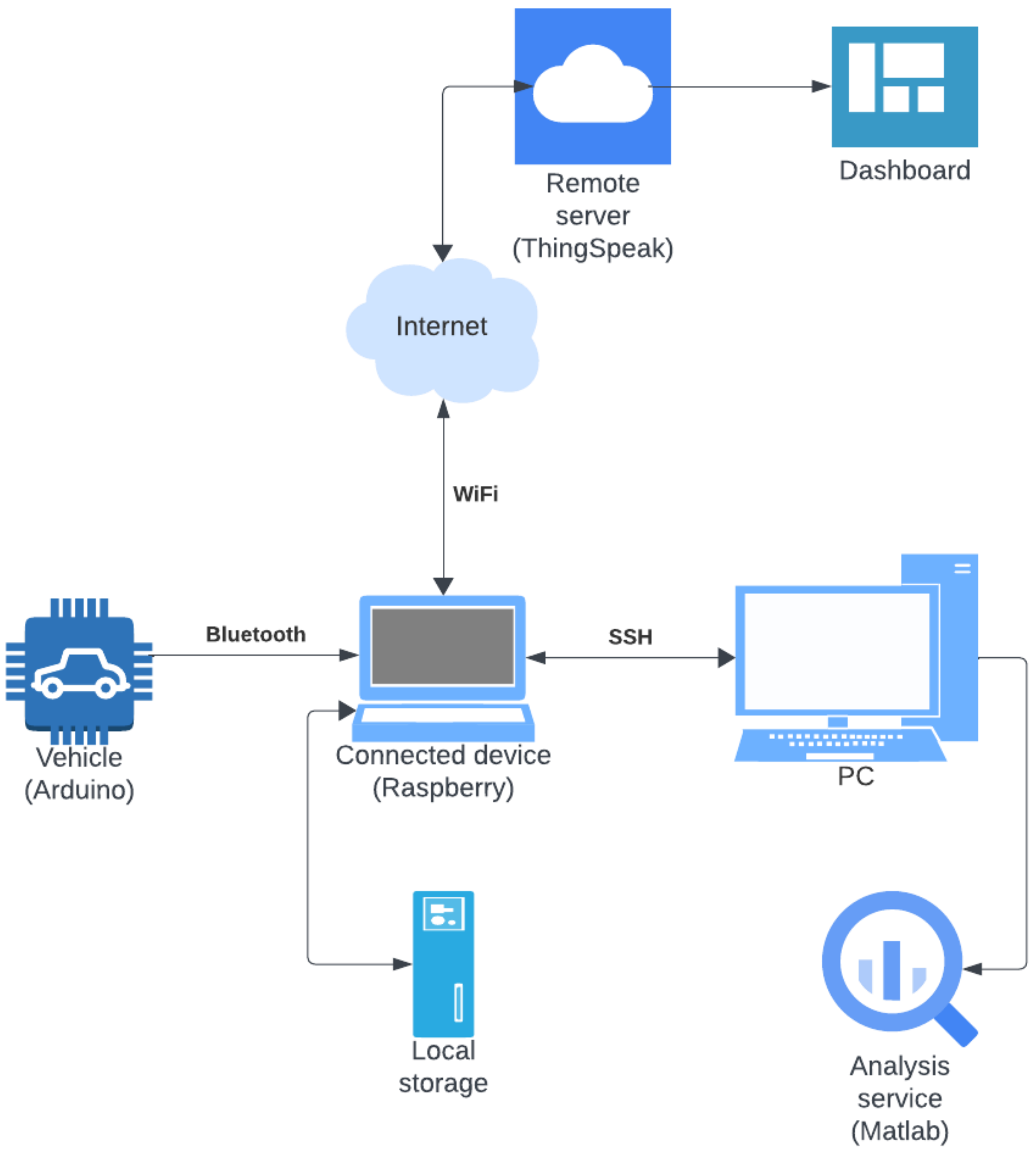
The vehicle data is available in real time as it is sent to a connected device which has the task of processing the data. The data is sent again to a remote server, to have a dashboard view, and is written to a csv file for further analysis.

The software is developed using a combination of programming languages: C++ for vehicle software, Python for the connected device and some MATLAB's script for simulation and analysis. The software is designed to control the movement of the vehicle and to process the data from the sensors.

The project is divided into several phases, including the design and construction of the vehicle, the development of the software, and the testing of the vehicle. The testing phase involve evaluating the vehicle's ability to move autonomously and to take accurate measurements of its position and distance.

Overall, this project aims to develop a robotic vehicle that can move autonomously and take accurate measurements of its position and distance. The use of a Kalman filter will ensure that the measurements are accurate and reliable, making the vehicle suitable for a wide range of applications.

To have a clear understanding of the implemented architecture, before going to the next chapter is useful to look at the Figure 1.1 – Designed architecture representing the designed architecture.



Remote server (ThingSpeak)
Dashboard
Internet
WiFi
Vehicle (Arduino)
Bluetooth
Connected device (Raspberry)
SSH
PC
Local storage
Analysis service (Matlab)

Figure 1.1 – Designed architecture

2. Requirements

In this chapter will be illustrated the hardware and software requirements for all the devices.

2.1. Vehicle

Hardware

For the vehicle control it was chosen to use the Arduino UNO R3[1] microcontroller board. For the implementation has been used a compatible board with the same characteristics.

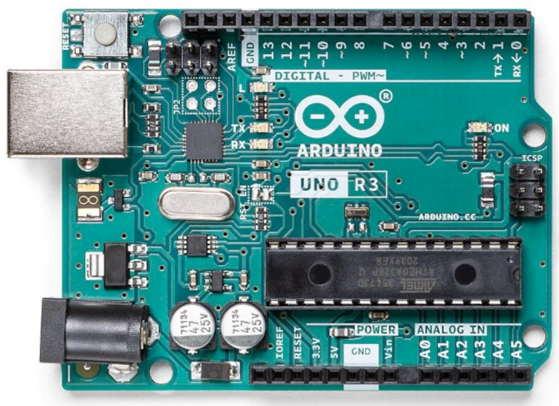


Figure 2.1 - Arduino Uno R3 microcontroller board

The board, based on the chip ATmega328P, allow to connect several devices via its 14 digital pins and 6 analog pins as shown in Figure 2.2 - Arduino UNO R3 pinout.

[1] For further details check the official documentation here: https://docs.arduino.cc/hardware/uno-rev3.

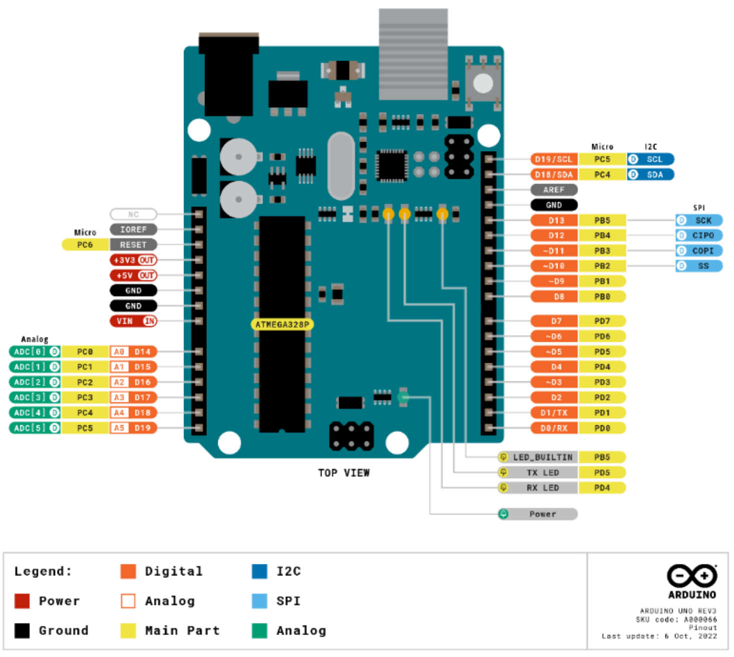


Figure 2.2 - Arduino UNO R3 pinout

For the implementation, the following devices has been connected and used:

- HC-05 module to provide Bluetooth connectivity.
- FC-03 module (optical sensor) and wheel encoder to measure round per seconds of the wheel.
- HC-SR04 module (ultrasonic sensor) to measure distance between the vehicle and an obstacle.
- L298 motor shield and four geared motors with wheel to move the vehicle.
- SG90 servo motor to move and center the ultrasonic sensor.
- VS1838 IR receiver (38kHz) to receive commands via an IR remote controller.
- MB102 Breadboard power supply module to provide power, necessary due to different voltage requirements from components.
- Battery holder that allows to insert two 18650 rechargeable batteries to provide power.

In Figure 2.3 the circuit scheme with the devices descripted above.

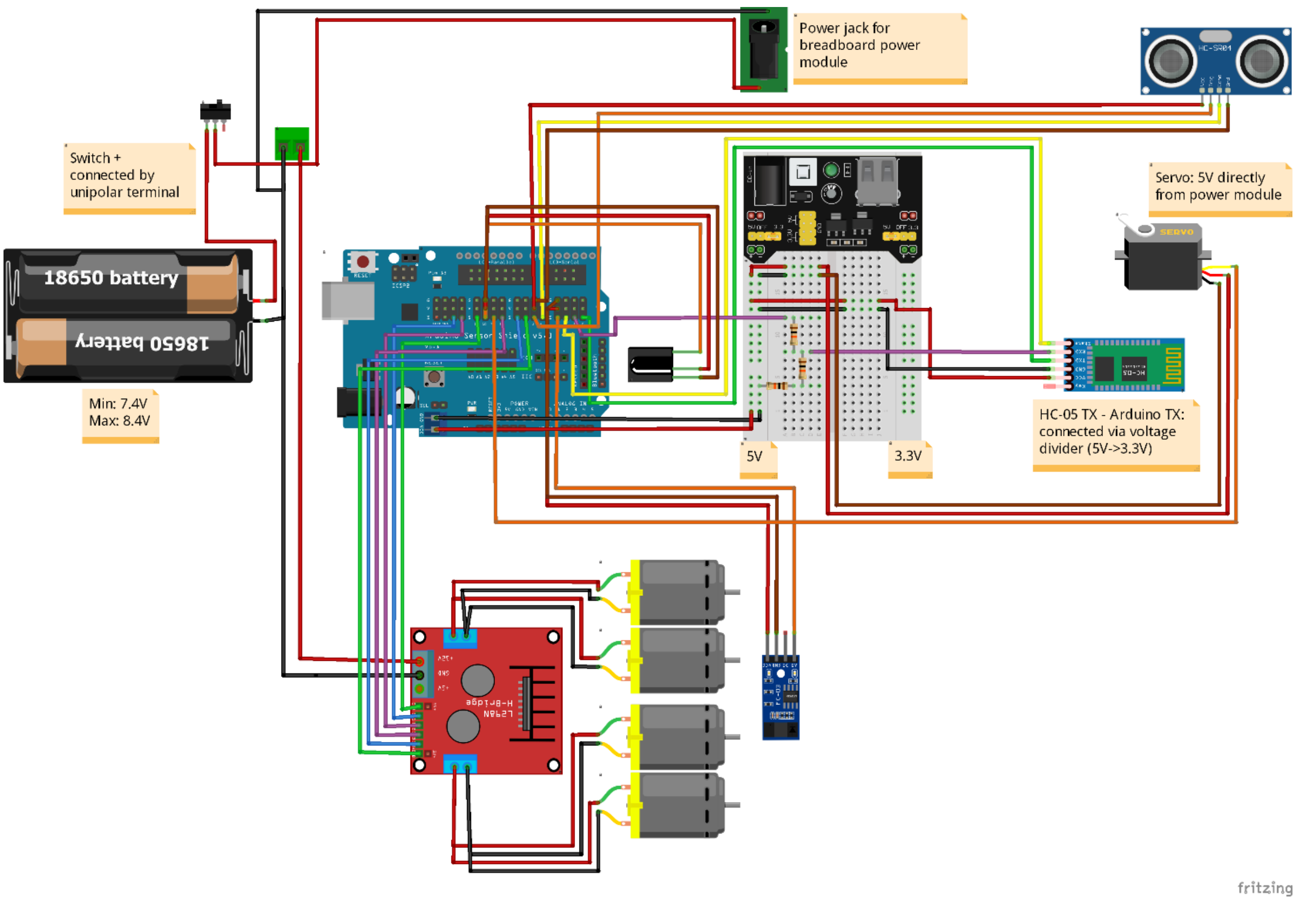

Power jack for breadboard power module
Switch + connected by unipolar terminal
18650 battery
18650 battery
Min: 7.4V
Max: 8.4V
Servo: 5V directly from power module
5V
3.3V
HC-05 TX - Arduino TX: connected via voltage divider (5V->3.3V)
fritzing

Figure 2.3 - Project circuit scheme

Software

To compile and upload to the microcontroller the C++ source code it has been used the Arduino IDE 2.2.1.

The following libraries has been installed and used to write the main sketch:

- IRremote (4.2.0) (just TinyIRReceiver) to receive the IR commands.
- Servo (1.2.1) to control the SG90 servo motor.
- BasicLinearAlgebra (4.1) to perform the matrix computation in an easier way.

2.2. Connected device

Hardware

As connected device it was chosen a Raspberry Pi 3 B[2]. This device is a single-board computer with minimal hardware but with few advantages that fits the project scenario: good portability and an extremely low power consumption. This model has a built-in Bluetooth WiFi connectivity and so it does not need any additional module to work. For

[2] For further details check the official documentation here https://www.raspberrypi.com/documentation and the product page for specific information here https://www.raspberrypi.com/products/raspberry-pi-3-model-b.

commissioning it requires only an external microSD that contains a compatible OS and a micro-USB power supply to power on the device.

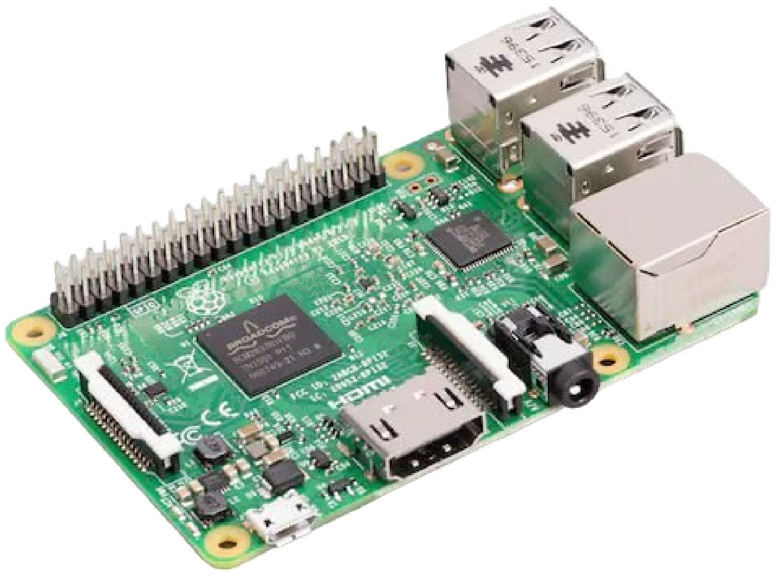


Figure 2.4 - Raspberry Pi 3 B

Software

The OS installed is Debian GNU/Linux 11 (bullseye). When installing the OS through the suggested software Raspberry Pi Imager it is possible to configure the WiFi connection and enable the SSH connection. With this configuration it is possible to use the Raspberry Pi remotely from other device that supports the SSH connection.

The scripts are written in Python and Bash languages.

The Python version used is the 3.9.2 and the following libraries has been installed and used:

- pyserial 3.5 to manage the serial connection between Arduino and Raspberry through Bluetooth.
- Requests 2.31.0 to manage HTTP requests.
- Python-decouple 3.8 to manage parameters passed to the script.

2.3. MATLAB

For the data analysis and simulation, it was used MATLAB R2023b Update 5. MATLAB allows a simple data management due its matrix-based language that allows to perform complex operation in a quite simple and natural language.

Furthermore, for certain commands, has been necessary to install the following packages:

- Control System Toolbox (23.2).
- Image Processing Toolbox (23.2).

- Optimization Toolbox (23.2).

2.4. ThingSpeak

As said in introduction, the connected device sends the data to a remote server. This server is ThingSpeak[3]. ThingSpeak is an IoT analytics platform service that allows to aggregate, visualize, and analyze live data streams in the cloud. This service is well integrated with MATLAB, in fact it is possible to do analytics inside ThingSpeak writing and executing MATLAB code to perform preprocessing, visualizations, and analyses.

To collect data with ThingSpeak it is necessary to have an account and to create and configure a channel. A channel is used to put together the data about the same context and to create a view (dashboard) to allow users to easily check the data. During the creation phase is necessary to define the fields that represent the data. In the end, to write and read data from the channel is necessary to get the Channel ID and the read/write API Key. These steps will be clearer when the implementation will be discussed in Chapter 5.

[3] For further details check the official documentation here https://ch.mathworks.com/help/thingspeak.

3. Dynamic model

In this chapter will be discussed how the continuous-time dynamic system was derived, and how it was discretized. A description of how the physical parameters characterizing the system were obtained will also be provided.

3.1. Continuous-time dynamic model

The dynamic model is described by two vectorial equations: a state equation and a measure equation. The first one binds the derivative of the state $\dot{x}(t) \in \mathbb{R}^2$ with its value $x(t) \in \mathbb{R}^2$ and the value of an input $u(t) \in \mathbb{R}$. In this case the state vector is composed of position $p(t)$ [cm] and velocity $v(t)$ [cm/s]. All the quantities refer to the vehicle with respect to a reference object. The second one binds the measure $z(t) \in \mathbb{R}^2$ with the state.

State equation

As known from physics, the derivative of the position is the velocity, and the derivative of the velocity is the acceleration (denoted by $a(t)$ [cm/s^2]). So, the state equation will be of this form:

$$\begin{bmatrix} \dot{p}(t) \\ \dot{v}(t) \end{bmatrix} = \begin{bmatrix} v(t) \\ a(t) \end{bmatrix} \tag{3.1-1}$$

To obtain an expression of $a(t)$ it is necessary to understand which forces act on the vehicle.

First, it must be understood how the input interacts with the vehicle. The input $u(t) \in [-255,255]$ represents a PWM[4] of a voltage V_p $[V]$. This voltage is sent to the four geared motors causing an application of a force $f_{in}(t)$ that will cause a rotatory acceleration. This acceleration has a direction dependent of the sign of the input and an intensity proportional to the absolute value. The acceleration of the motors implies a linear acceleration that is still be proportional to the input $u(t)$. So, it is possible to consider that:

$$f_{in}(t) \propto V_{in}(t) \propto \frac{V_p}{255} u(t) \Rightarrow f_{in}(t) = \eta_V \frac{V_p}{255} u(t) \tag{3.1-2}$$

Where η_V $[N/V \times 10^{-2}]$ is a proportionality coefficient between the force $f_{in}(t)$ and the input voltage $V_{in}(t)$.

It is necessary to also consider a friction force due to several factors (air drag, motor friction, and similar) that cause the vehicle to achieve a maximum operating speed. This force, proportional to the velocity, is modeled as follow:

$$f_{frict}(t) = -bv(t) \tag{3.1-3}$$

[4] Stands for Pulse-width modulation, it is a technique used to the control the average power delivered by an electrical signal. The PWM control the duty cycle of a square wave from 0% to 100%.

Where $b\ [kg/s]$ represents the overall friction coefficient.

So, in this model the following forces is considered:

- Force proportional to input $f_{in}(t) = \eta_V V_{in}(t) = \eta_V \frac{V_p}{255} u(t)$.
- Friction force $f_{frict}(t) = -bv(t)$.

So, the Newton's second law equation is:

$$\sum F(t) = Ma(t) \Rightarrow f_{in}(t) - bv(t) = Ma(t) \Rightarrow a(t) = \frac{f_{in}(t)}{M} - \frac{bv(t)}{M} \qquad (3.1\text{-}4)$$

Where $M\ [kg]$ represents the mass of the vehicle.

Considering the results obtained in the equation (3.1-4), it is possible to write the state equation, separating the contribute given by the state and the one given by the input:

$$\begin{bmatrix} \dot{p}(t) \\ \dot{v}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{b}{M} \end{bmatrix} \begin{bmatrix} p(t) \\ v(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{M}\eta_V \frac{V_p}{255} \end{bmatrix} u(t) + v_{PR}(t) \qquad (3.1\text{-}5)$$

This equation, written in form $\dot{x}(t) = Ax(t) + Bu(t) + v(t)$, where $A \in \mathbb{R}^{2\times 2}$ is the state matrix that represents how the state evolve in function of itself, and $B \in \mathbb{R}^2$ is the input matrix that represents the evolution in function of the input $u(t)$ and $v_{PR}(t) \in \mathbb{R}^2$ represents the process noise (more details will be provided later).

Measure equation

The vehicle can take two measures:

- Vehicle-object distance $d_o(t)\ [cm]$ taken by the ultrasonic sensor.
- Pulses per second $pulse(t)\ [s^{-1}]$ taken by the optical sensor and wheel encoder.

In vector notation results the following measurements vector:

$$z(t) = \begin{bmatrix} z_1(t) \\ z_2(t) \end{bmatrix} = \begin{bmatrix} d_o(t) \\ pulse(t) \end{bmatrix} \qquad (3.1\text{-}6)$$

The measure equation is of the form $z(t) = Cx(t)$, where $C \in \mathbb{R}^{2\times 2}$ is the measure matrix that represents the bond between the state and the measures.

The measure taken by the ultrasonic sensor corresponds to the position of the vehicle, so $d_o(t) = p(t)$. This measure is taken by the HC-SR04 sensor that can emit and receive an ultrasonic signal. The sensor emits a sound pulse and receive the same signal after bouncing to the reference object. Since the sound pulse travel at sound speed $v_s(T)[m/s]$[5] (about 343,4 m/s in air with $T = 20°\ C$) and considering that the pulse travels two time the distance, it is possible to take the measure $d_0(t)$ in the following way:

[5] The speed of sound depends on medium and temperature. The considered value is chosen considering air as medium and a temperature of about 20° C.

- Emit a sound pulse and receive the same signal.
- Measure the round-trip time $t_{rt}(t)$ $[s]$.
- Measure the distance with the formula $d_o(t) = v_s \cdot 10^{-2} \cdot \frac{t_{rt}(t)}{2}$.

Where factor 10^{-2} is necessary to convert the speed of sound in cm/s.

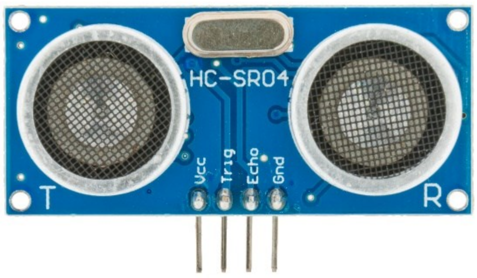


Figure 3.1 - Ultrasonic sensor HC-SR04

The measure taken by the optical sensor consists of number of pulses per second. The sensor has an IR led and a phototransistor and can reveal if there is an obstacle between the two components.

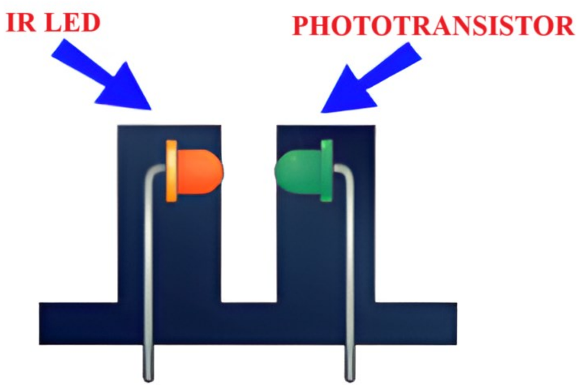


Figure 3.2 - Photo interrupter embedded in FC-03 sensor

The sensor provides as output a HIGH signal when the IR led light cannot reach the phototransistor (so if there is an obstacle in the middle), and a LOW signal instead. A pulse is counted by the vehicle every time the signal rise.

Using a wheel encoder is possible to compute the rounds per second of the wheel connected to a motor. The encoder is simply a little circle with a specific number of holes that can be attached on the other side of motor, so the encoder round as the wheel. If the encoder is inserted in the middle of the optical sensor the result is an alternating HIGH/LOW signal with a frequency proportional to the rotation speed.

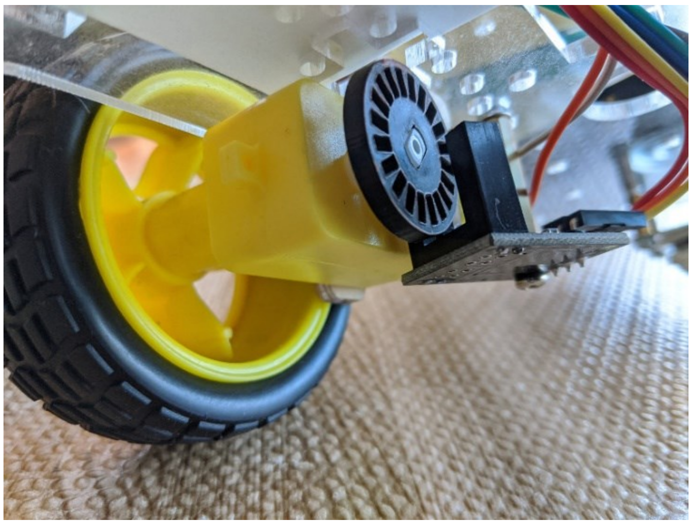


Figure 3.3 - Motor with attached wheel and encoder inserted in the middle of optical sensor

Known the number of pulses $pulse(t)$ and the number of holes PPR (pulses per revolution), is possible to compute the rounds per second $rps(t)$ $[s^{-1}]$ with the following formula:

$$rps(t) = \frac{pulse(t)}{PPR} \tag{3.1-7}$$

This value, supposing that all the wheel round at the same speed and direction, can be converted in linear velocity $v(t)$ $[cm/s]$ using the following formula:

$$v(t) = rps(t) \cdot \pi D \tag{3.1-8}$$

Where D is the diameter of the wheel.

So, looking at (3.1-7) and (3.1-8), it is possible to express the relationship between the measure and the state as follows:

$$pulse(t) = v(t) \cdot \frac{PPR}{\pi D} \tag{3.1-9}$$

After these considerations is possible to express the measure equation in vectorial notation as follows:

$$\begin{bmatrix} d_o(t) \\ pulse(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & \frac{PPR}{\pi D} \end{bmatrix} \begin{bmatrix} p(t) \\ v(t) \end{bmatrix} + w(t) \tag{3.1-10}$$

Where $w(t) \in \mathbb{R}^2$ represents the measure noise (more details will be provided later).

Summary

The continuous time dynamic model, defined in equations (3.1-5) and (3.1-10) is the following:

$$\begin{cases} \begin{bmatrix} \dot{p}(t) \\ \dot{v}(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\dfrac{b}{M} \end{bmatrix} \begin{bmatrix} p(t) \\ v(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \dfrac{1}{M}\eta_V \dfrac{V_p}{255} \end{bmatrix} u(t) + v_{PR}(t) \\ \begin{bmatrix} d_o(t) \\ pulse(t) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & \dfrac{PPR}{\pi D} \end{bmatrix} \begin{bmatrix} p(t) \\ v(t) \end{bmatrix} + w(t) \end{cases} \tag{3.1-11}$$

Where:

$$A = \begin{bmatrix} 0 & 1 \\ 0 & -\dfrac{b}{M} \end{bmatrix} \tag{3.1-12}$$

$$B = \begin{bmatrix} 0 \\ \dfrac{1}{M}\eta_V \dfrac{V_p}{255} u(t) \end{bmatrix} \tag{3.1-13}$$

$$C = \begin{bmatrix} 1 & 0 \\ 0 & \dfrac{PPR}{\pi D} \end{bmatrix} \tag{3.1-14}$$

3.2. Discretization

To implement the filter, it is necessary to discretize the continuous time system. The result of this process is a discrete-time dynamic model defined as follows:

$$\begin{cases} x(k+1) = Fx(k) + Gu(k) + \Lambda v_{PR}(k) \\ z(k) = Hx(k) + w(k) \end{cases} \tag{3.2-1}$$

Where the matrices F, G, H are the discrete counterparts of matrices A, B, C respectively and Λ is a matrix that maps the process noise.

To discretize the continuous model is necessary to choose a discretization step T sufficiently small. Using an integral approach, it is possible to approximate the matrices characterizing the discrete model as follows:

$$F = e^{AT} \cong \begin{bmatrix} 1 & T\left(1 - \dfrac{b}{M}\dfrac{T}{2}\right) \\ 0 & 1 - \dfrac{b}{M}T\left(1 - \dfrac{b}{M}\dfrac{T}{2}\right) \end{bmatrix} \tag{3.2-2}$$

$$G = (e^{AT} - I)A^{-1}B \cong \frac{1}{M}\eta_V \frac{V_p}{255} \begin{bmatrix} \dfrac{T^2}{2} \\ T\left(1 - \dfrac{b}{M}\dfrac{T}{2}\right) \end{bmatrix} \tag{3.2-3}$$

$$\Lambda = (e^{AT} - I)A^{-1} \cong \begin{bmatrix} T & \dfrac{T^2}{2} \\ 0 & T\left(1 - \dfrac{b}{M}\dfrac{T}{2}\right) \end{bmatrix} \tag{3.2-4}$$

$$H = C = \begin{bmatrix} 1 & 0 \\ 0 & \frac{PPR}{\pi D} \end{bmatrix} \tag{3.2-5}$$

Where the approximation is given by the fact that has been considered only the first three terms for the matrix exponential, that is:

$$e^{AT} = \sum_{k=0}^{\infty} \frac{(AT)^k}{k!} \underset{k=0,1,2}{\cong} I + AT + \frac{A^2T^2}{2} \tag{3.2-6}$$

And where I is the identity matrix, chosen with the right dimension (2×2).

To simplify the notation, it is useful to define the following constant:

$$K_{bMT} = T\left(1 - \frac{b}{M}\frac{T}{2}\right) \tag{3.2-7}$$

So, the discrete-time system is given by:

$$\begin{cases} \begin{bmatrix} p(k+1) \\ v(k+1) \end{bmatrix} = \begin{bmatrix} 1 & K_{bMT} \\ 0 & 1 - K_{bMT} \end{bmatrix} \begin{bmatrix} p(k) \\ v(k) \end{bmatrix} + \frac{1}{M}\eta_V \frac{V_p}{255} \begin{bmatrix} \frac{T^2}{2} \\ K_{bMT} \end{bmatrix} u(k) + \Lambda v_{PR}(k) \\ \begin{bmatrix} d_o(k) \\ pulse(k) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & \frac{PPR}{\pi D} \end{bmatrix} \begin{bmatrix} p(k) \\ v(k) \end{bmatrix} + w(k) \end{cases} \tag{3.2-8}$$

3.3. Noise

As seen in the previous equations, there are noises terms in state and measure equations. There is process noise $v_{PR}(k)$ and measure noise $w(k)$. The first one is related to the uncertainty of the model that is not perfect, so it appears in the state equation. The second one concerns the intrinsic noise of the sensors and appears in the measure equation.

In general, the noise is modeled as a stochastic process and it is supposed to be additive, white[6], Gaussian with zero mean and certain covariance. In this case these assumptions hold for both the noises and results that:

$$\begin{gathered} v_{PR}(k) \sim \mathcal{N}(0, Q) \\ w(k) \sim \mathcal{N}(0, R) \end{gathered} \tag{3.3-1}$$

Where Q and R represents the covariance matrices associated to process and measure noise, respectively. More assumptions will be made in Chapter 4.

Before dealing with process noise, it is necessary to get more information about the measure noise. In fact, the process noise is chosen in relation to the other one and usually this choice is known as the Kalman Filter calibration. Perhaps the discussion about the process noise is postponed to the Chapter 4.

[6] A stochastic process is defined as white when it is uncorrelated with itself at different times.

The measure noise instead can be obtained considering more information about the sensors. Usually in the datasheet there are plenty of information about the noise and uncertainty.

For the HC-SR04, looking at the datasheet, it results a standard deviation equals to $\sigma_{US} = 0.3\ [cm]$.

For the FC-03 sensor and the wheel encoder, no precise information has been found. So, it has been chosen to model the noise considering the quantization error due to a limited number of holes in the wheel encoder. In fact, the sensor can count a pulse only if the following hole has been reached. This implies an error that can be modeled as noise.

In terms of pulses, this means that the noise is bounded in the interval [0,1] with an expected value equal to $\mu_p = \frac{1}{2}$.

Since it is necessary to have a zero mean noise it is possible to consider the interval $\left[-\frac{1}{2}, \frac{1}{2}\right]$ by shifting forward the measure taken by the optical sensor by μ_p.

So, it makes sense to consider that the noise w_{OPT} is normally distributed with zero mean and a variance such that it is very improbable to obtain realization outside the interval considered before.

Considering that the most probable values of a Gaussian are located at a maximum of 3 standard deviations from the mean, it is reasonable to choose the standard deviation as follows:

$$0 + 3\sigma_{OPT} = \frac{1}{2} \Rightarrow \sigma_{OPT} = \frac{1}{6} \qquad (3.3\text{-}2)$$

Observe that, the consequences of this error, in terms of velocity, is that, in the worst case, supposing that the sensor is missing exactly one pulse, the measured velocity is lower than real by a factor of

$$w_{speed_{max}} = \frac{1}{PPR} \cdot \pi D \cong 1{,}0210\ \frac{cm}{s} \qquad (3.3\text{-}3)$$

This computation is done just to give an idea of the impact of the noise to velocity. In fact, the noise must refer to the pulse's measurements.

This information give as results the following covariance matrix:

$$R = \begin{bmatrix} \sigma_{US}^2 & 0 \\ 0 & \sigma_{OPT}^2 \end{bmatrix} = \begin{bmatrix} 0{,}3^2 & 0 \\ 0 & \left(\frac{1}{6}\right)^2 \end{bmatrix} \qquad (3.3\text{-}4)$$

3.4. Parameters

As seen previously, the system described in *(3.2-8)* is characterized by several physical parameters that need to be evaluated. The parameters that must be determined are the following:

- $M\ [kg]$: mass of the vehicle.
- $V_p\ [V]$: peak voltage sent to the motors corresponding to max value of input.
- $b\ [kg/s\ ]$: overall friction coefficient.
- $\eta_V\ [N/V \times 10^{-2}]$: proportionality coefficient between input voltage and input force.
- PPR: pulses per revolution, the number of holes in wheel encoder.
- $D\ [cm]$: diameter of the wheel.

These parameters can be estimated from experimental results or obtained from datasheet or direct measure.

From datasheet or direct measure

These parameters do not need any assumptions and its values are reported in the following list:

- $M = 0{,}731\ kg$
- $V_p = 6{,}0\ V$
- $PPR = 20$
- $D = 6{,}5\ cm$

Note that V_p depends on supply voltage V_+[7]. It is assumed to be constant and equal to $V_+ = 8.36\ V$. This assumption holds for short usage with batteries with a good state of charge.

Estimated from experimental results

The two remaining parameters can be estimated indirectly doing calculus and some specific experiment.

First, it is necessary to obtain some useful results from the differential equation characterizing the evolution of velocity. This equation appears in equation (3.1-5) and it is the following:

$$\dot{v}(t) = -\frac{b}{M}v(t) + \frac{K_V}{M}u(t), v(0) = v_0 \tag{3.4-1}$$

Where $K_V = \eta_V \frac{V_p}{255}\ [N \times 10^{-2}]$ is a constant defined for simplicity of notation.

It is useful to inspect two cases:

Case 1

$$u(t) = C_{in}, v(0) = 0$$

[7] It is the voltage supplied by the two 18650 batteries and depends on the charge state of them.

The equation becomes the following:

$$\dot{v}(t) = -\frac{b}{M}v(t) + \frac{K_V C_{in}}{M}, v(0) = 0 \qquad (3.4\text{-}2)$$

The solution can be found using the separation of variables method and it is the following:

$$v(t) = \frac{K_V C_{in}}{b}\left(1 - e^{-\frac{b}{M}t}\right) \qquad (3.4\text{-}3)$$

It is possible to observe that:

$$\lim_{t\to\infty} v(t) = \frac{K_V C_{in}}{b} \qquad (3.4\text{-}4)$$

That is equal to the max velocity corresponding to an input equal to C_{in}. Observe that the result is correct from a dimensional perspective. In fact,

$$\frac{[N \times 10^{-2}]}{\left[\frac{kg}{s}\right]} = \left[kg \times \frac{m}{s^2}\right] \cdot \left[\frac{s}{kg}\right] \times 10^{-2} = \left[\frac{m}{s} \times 10^{-2}\right] = \left[\frac{cm}{s}\right] \qquad (3.4\text{-}5)$$

Case 2

$$u(t) = 0, v(0) = \frac{\mathrm{K_V C_{in}}}{\mathrm{b}}$$

In this case the equation becomes the following:

$$\dot{v}(t) = -\frac{b}{M}v(t), v(0) = \frac{K_V C_{in}}{b} \qquad (3.4\text{-}6)$$

The solution can be found as the previous case, and it is the following:

$$v(t) = \frac{K_V C_{in}}{b} e^{-\frac{b}{M}t} \qquad (3.4\text{-}7)$$

Obviously, it results that:

$$\lim_{t\to\infty} v(t) = 0 \qquad (3.4\text{-}8)$$

So, the vehicle appears to be stationary after a certain period. It is useful to know after how much time it is sufficiently stationary[8]. This can be done resolving the following inequality:

$$v(t) - \lim_{t\to\infty} v(t) < \varepsilon \Leftrightarrow \frac{K_V C_{in}}{b} e^{-\frac{b}{M}t} < \varepsilon \qquad (3.4\text{-}9)$$

Where $\varepsilon > 0$ $[cm/s]$ is a threshold with a very small value. The result, assuming a positive velocity is the following:

[8] The term sufficiently refers to the fact that the vehicle reaches the idle state only for an infinite value of t.

$$t_{stop}(C_{in},\varepsilon)=\frac{M}{b}ln\left(\frac{1}{\varepsilon}\frac{K_VC_{in}}{b}\right)\underset{\varepsilon=\epsilon_{KC}\cdot\frac{K_VC_{in}}{b}}{=}\frac{M}{b}ln\frac{1}{\epsilon_{KC}} \qquad (3.4\text{-}10)$$

Where $\epsilon_{KC}>0$ it is a particular choice of epsilon that set a threshold relative[9] to the starting velocity value. The absolute error committed will be equal to ε.

A value of $\epsilon_{KC}<0{,}01$ it is reasonable and so the choice done is $\epsilon_{KC}=e^{-5}\cong 0{,}0067$. This choice leads to the following result:

$$\tilde{t}_{stop}=5\cdot\frac{M}{b} \qquad (3.4\text{-}11)$$

Friction coefficient b

This result it is useful to estimate the friction coefficient b when are known the mass M and the time $\tilde{t}_{stop}$ that can be determined experimentally.

Observe that this result can be achieved with similar assumptions done in case 1 (reasoning about time to reach max speed) and leads to the same results. Since it is harder to determine with experiments it is preferred to use this method.

The expression of b can be obtained by the equations (3.4-10) (3.4-11) and is the following:

$$b=\frac{1}{\epsilon_{KC}}\cdot\frac{M}{\tilde{t}_{stop}}\underset{\epsilon_{KC}\cong e^{-5}}{=}5\cdot\frac{M}{\tilde{t}_{stop}} \qquad (3.4\text{-}12)$$

Where it is necessary to determine $\tilde{t}_{\text{stop}}$ experimentally.

The results are the following:

- $\tilde{t}_{stop}=0{,}615\ s$
- $b=5{,}9431\ kg/s$

Voltage coefficient η_V

Known the coefficient b and considering that the max velocity corresponding to an input equal to C_{in} is given by $v_{max}(C_{in})=\frac{K_VC_{in}}{b}$, and remembering that $K_V=\eta_V\frac{V_p}{255}$, it results that

$$v_{max}(C_{max})=\frac{K_VC_{max}}{b}=\frac{\eta_VV_p}{255}\cdot\frac{255}{b}=\frac{\eta_VV_p}{b} \qquad (3.4\text{-}13)$$

And so

$$\eta_V=v_{max}(C_{max})\cdot\frac{b}{V_p} \qquad (3.4\text{-}14)$$

Where it is necessary to determine $v_{max}(C_{max})$ experimentally.

The results are the following:

[9] In fact, the term ϵ_{KC} results to be adimensional.

- $v_{max}(C_{max}) = 71{,}471\ cm/s$
- $\eta_V = 70{,}7931\ N/V \times 10^{-2}$

3.5. Model summary

This section reports the results obtained in this chapter[10]:

Discrete-time model

$$K_{bMT} = T\left(1 - \frac{b}{M}\frac{T}{2}\right)$$

$$\begin{cases} \begin{bmatrix} p(k+1) \\ v(k+1) \end{bmatrix} = \begin{bmatrix} 1 & K_{bMT} \\ 0 & 1 - K_{bMT} \end{bmatrix} \begin{bmatrix} p(k) \\ v(k) \end{bmatrix} + \frac{1}{M}\eta_V \frac{V_p}{255} \begin{bmatrix} \frac{T^2}{2} \\ K_{bMT} \end{bmatrix} u(k) + \Lambda v_{PR}(k) \\ \begin{bmatrix} d_o(k) \\ pulse(k) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & \frac{PPR}{\pi D} \end{bmatrix} \begin{bmatrix} p(k) \\ v(k) \end{bmatrix} + w(k) \end{cases}$$

$$v_{PR}(k) \sim \mathcal{N}(0, Q)$$
$$w(k) \sim \mathcal{N}(0, R)$$

$$\sigma_{US} = 0{,}3 \text{ cm}$$

$$\sigma_{OPT} = \frac{1}{6} \text{ pulses}$$

$$R = diag(\sigma_{US}^2, \sigma_{OPT}^2)$$

$$\sigma_{Q_p} = 0.1$$

$$\sigma_{Q_v} = 0.1$$

$$Q = diag\left(\sigma_p^2, \sigma_v^2\right)$$

With the following parameters:

- $M = 0{,}731\ kg$
- $b = 5{,}9431\ kg/s$
- $V_p = 6{,}0\ V$
- $\eta_V = 70{,}7931\ N/V \times 10^{-2}$
- $PPR = 20$
- $D = 6{,}5\ cm$

[10] Also reports the process noise choice obtained in Chapter 4.2 for completeness.

3.6. Fixed position model

For some experimentation it is useful to use a model in which the velocity does not affect the position of the vehicle. This happens when the motors spin while the vehicle is not touching the floor. In this case the matrix A, described in equation (3.1-5), of the continuous time dynamic model, change because the position is constant and result that $\dot{x}(t) = 0$, so

$$A = \begin{bmatrix} 0 & \mathbf{0} \\ 0 & -\frac{b}{M} \end{bmatrix} \tag{3.6-1}$$

This implies the following changes in the discrete time model[11]:

$$F = \begin{bmatrix} 1 & \mathbf{0} \\ 0 & 1 - \frac{b}{M} K_{bMT} \end{bmatrix} \tag{3.6-2}$$

$$G = \frac{1}{M} \eta_V \frac{V_p}{255} \begin{bmatrix} \mathbf{0} \\ K_{bMT} \end{bmatrix} \tag{3.6-3}$$

$$\Lambda = \begin{bmatrix} T & \mathbf{0} \\ 0 & K_{bMT} \end{bmatrix} \tag{3.6-4}$$

All the other characteristics and assumptions remain the same and are still valid.

[11] The changes are denoted in bold.

4. Filtering algorithm: the Kalman filter

As seen in equations (3.1-8), there is noise that contaminates the measurements. To filter out the noise it is necessary to implement an appropriate estimation technique. The algorithm chosen to dd this is the Kalman filter (KF).

4.1. Algorithm description

This algorithm, which takes its name from Rudolf E. Kalman, is a recursive algorithm based on two steps:

- Prediction: predict the state at time $k+1$.
- Correction: adjust the prediction using the added information came at time $k+1$.

The first step is done using the model, the second one is done using the measurements. The output of this algorithm is not only the estimated state but also the covariance of the estimate that is updated at every step.

As any recursive algorithm, it needs an initialization. It requires the initial estimate $\hat{x}(0|0)$ and the initial covariance $P(0|0)$. The initial state $x(0)$, modeled as a stochastic process, can be assumed Gaussian with mean $\hat{x}(0|0)$ and covariance $P(0|0)$.

Note that the algorithm can be applied to systems described by linear equations (w.r.t. the state and the measures) and in which the noise is modeled as white stochastic process. For non-linear systems can be used the EKF (Extended Kalman Filter) that is based on linearization.

A simple diagram representing how the KF works is the following:

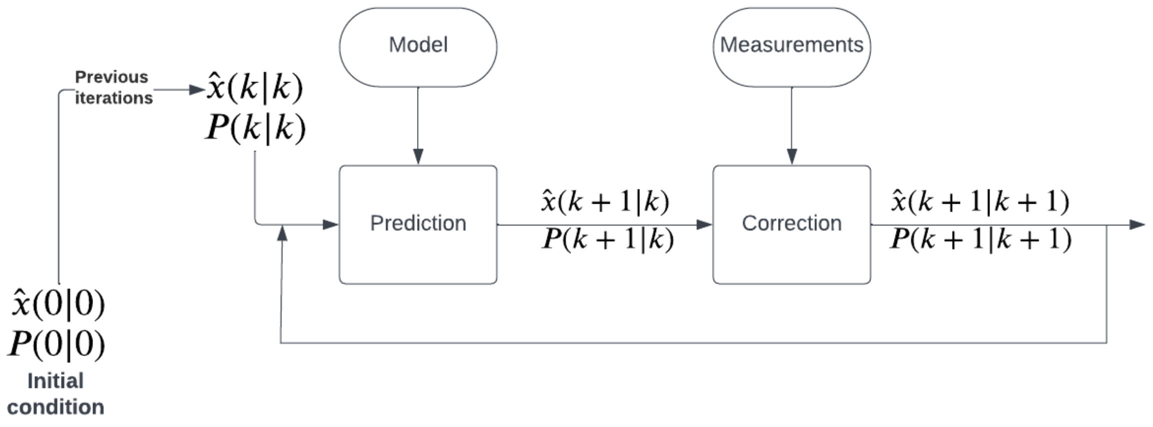


Figure 4.1 - Kalman filter algorithm diagram

Where the notation $\hat{x}(k+1|k)$ represents an estimate of the state at time $k+1$ using only the information available at time k. Usually, this estimate is called prediction, while $\hat{x}(k+1|k+1)$ is called correction or estimate. The same notation also applies to the covariances.

The equations that characterize the algorithm are the following:

Prediction step (Predictor)

State prediction

$$\hat{x}(k+1|k) = F(k)\hat{x}(k|k) + G(k)u(k) \tag{4.1-1}$$

Prediction covariance

$$P(k+1|k) = F(k)P(k|k)F(k)^T + Q(k) \tag{4.1-2}$$

Correction step (Corrector)

Filter gain

$$W(k+1) = P(k+1|k)H(k+1)^T[H(k+1)P(k+1|k)H(k+1)^T + R(k+1)]^{-1} \tag{4.1-3}$$

State correction

$$\hat{x}(k+1|k+1) = \hat{x}(k+1|k) + W(k+1)[z(k+1) - H(k+1)\hat{x}(k+1|k)] \tag{4.1-4}$$

Correction covariance

$$P(k+1|k+1) = [I - W(k+1)H(k+1)]P(k+1|k) \tag{4.1-5}$$

Where:

- $F(k), G(k), H(k)$ are the matrices that describe the model as in equations (3.2-2), (3.2-3), (3.2-5) that can be in general time variant but are constant in this scenario.
- $u(k)$ is the input of the system.
- $Q(k), R(k)$ are the covariance matrices relative to process and measure noise, respectively. Also, in this case these matrices can be time variant but are constant in this scenario.

Requirements

To apply the KF it is necessary that the dynamic system satisfies some conditions.

As said before, the first condition is that the system needs to be linear w.r.t. the state and the measure.

The second condition refers to the noise. The process and measure noises $v(k), w(k)$ need to be additive, white, Gaussian[12] with zero mean and a certain covariance, so:

$$\begin{gathered} E[v(k)] = 0 \\ E[v(k)v(j)^T] = Q(k)\delta_{kj} \end{gathered} \tag{4.1-6}$$

$$\begin{gathered} E[w(k)] = 0 \\ E[w(k)w(j)^T] = R(k)\delta_{kj} \end{gathered} \tag{4.1-7}$$

[12] The Gaussianity requirement ensures that the KF is an optimal estimator, however if not satisfied, it can still be applied still obtaining a BLUE estimator.

Furthermore, known the initial state $x(0)$, it is also necessary that there is a mutual uncorrelation between the initial state, the process noise, and the measure noise, so:

$$\begin{gathered} E[x(0)v(k)^T] = 0 \\ E[x(0)w(k)^T] = 0 \\ E[v(k)w(j)^T] = 0 \end{gathered} \tag{4.1-8}$$

4.2. Initialization and process noise

Initialization

As seen before in Chapter 4.1, to apply the KF algorithm is necessary to initialize it with an initial estimate with an associated covariance. Since the ultrasonic sensor works good in range between 4 and 400 centimeters, it makes sense that the initial position is included in this interval. A reasonable choice for the initial position estimate is the following:

$$\hat{x}_p(0|0) = \frac{400+4}{2} = 202 \tag{4.2-1}$$

Furthermore, since it is typical that the most probable values of a Gaussian are located at a maximum of 3 standard deviations from the mean, it results that:

$$\hat{x}_p(0|0) + 3\sigma_p = 400 \Rightarrow \sigma_p = \frac{198}{3} = 66 \tag{4.2-2}$$

As regards speed, however, the vehicle is programmed to start the filtering process from a standstill, so the initial velocity estimate is the following:

$$\hat{x}_v(0|0) = 0 \tag{4.2-3}$$

The standard deviation can be chosen supposing a small movement proportional to the max speed $v_{max}(C_{max})$ defined in (3.4-13). Supposing as maximum reasonable value just a 3% value of maximum speed, it results:

$$\hat{x}_v(0|0) + 3\sigma_v = v_{max}(C_{max}) \cdot \frac{3}{100} \Rightarrow \sigma_v = \frac{v_{max}(C_{max})}{100} \tag{4.2-4}$$

With the previous assumptions, the initialization is the following:

$$\hat{x}(0|0) = \begin{bmatrix} \hat{x}_p(0|0) \\ \hat{x}_v(0|0) \end{bmatrix} = \begin{bmatrix} 66 \\ 0 \end{bmatrix} \tag{4.2-5}$$

$$P(0|0) = \begin{bmatrix} \sigma_p^2 & 0 \\ 0 & \sigma_v^2 \end{bmatrix} = \begin{bmatrix} 66^2 & 0 \\ 0 & \left(\frac{v_{max}(C_{max})}{100}\right)^2 \end{bmatrix} \tag{4.2-6}$$

Process noise

As said in Chapter 3.3., the choice of the process noise is usually made in relation to the measure noise: this procedure is known as Kalman Filter calibration.

Remembering the covariance matrix R described in equation (3.3-4) and considering that the model is made by 6 parameters, derived by experiments which are themselves subject to noise, the process noise is modeled as follows:

$$\sigma_{Q_p} = 0.1 \tag{4.2-7}$$

$$\sigma_{Q_v} = 0.1 \tag{4.2-8}$$

Giving the following covariance matrix as result:

$$Q = \begin{bmatrix} \sigma_{Q_p}^2 & 0 \\ 0 & \sigma_{Q_v}^2 \end{bmatrix} = \begin{bmatrix} 0.1^2 & 0 \\ 0 & 0.1^2 \end{bmatrix} \tag{4.2-9}$$

4.3. Asymptotic behavior

A particularly useful characteristic of the KF is that it is possible to study the asymptotic behavior without taking any measurements. In fact, as seen in equations (4.1-2), (4.1-3), (4.1-5), the covariance matrices $P(k+1|k), P(k+1|k+1)$ and the gain $W(k+1)$ do not depend on measurements or the state but only depend on the matrices that define the model. This allows to analyze the values taken by these 3 matrices before the effective implementation.

Furthermore, it is possible to demonstrate if, given an LTI[13] system, the covariance, and so the gain, will converge. This is possible thanks to the asymptotic convergence theorem that gives the sufficient[14] conditions to state the convergence. The theorem states that:

- If F is stable

  Or if

- The couple (F, H) is completely detectable.
- The couple (F, G) is completely stabilizable.

So, it results that:

$$\lim_{k \to \infty} P(k+1|k) = \bar{P} \tag{4.3-1}$$

Where $\bar{P} > 0$ exists and it is independent by the initialization and is the solution of the algebraic Riccati equation.

Applying this theorem to the system obtained for this scenario in Chapter 3.5 gives the following results (supposing $T = 0.1$):

$$F = \begin{bmatrix} 1 & 0.0593 \\ 0 & 0.5175 \end{bmatrix} \tag{4.3-2}$$

[13] LTI stands for Linear Time Invariant system.

[14] If the conditions are not satisfied it is not possible to say anything about convergence.

$$G = \begin{bmatrix} 0.0114 \\ 0.1352 \end{bmatrix} \tag{4.3-3}$$

$$H = \begin{bmatrix} 1 & 0 \\ 0 & 0.9794 \end{bmatrix} \tag{4.3-4}$$

And it results that:

- F is not stable.

But results that:

- The couple (F, H) is observable and so it is detectable.
- The couple (F, G) is reachable and so it is stabilizable.

So, it is granted the convergence of covariance and gain.

5. Implementation

This chapter provides implementation details. First, the simulation conducted in MATLAB will be discussed. Subsequently, details will be provided relating to the software installed on the vehicle (only the part relating to the KF), the one on the connected device and the one used to analyze the results. This chapter is not meant to provide a full description of the software but just a description of the main characteristics of it.

5.1. Scenario

Before talking about the implementation, it is useful to describe better the scenario.

The vehicle, in general, can move freely in an environment, read an input position received by IR remote, reach the specified position and transmit data to the connected device. These actions are divided into 4 operating states:

- **Free walk**: move freely in an environment.
- **Read**: read custom position from IR remote.
- **Explore**: reach custom position specified in Read mode.
- **Data transmission**: same as Free walk mode but with data transmission.

Considering the names of these 4 states, the vehicle takes the name of F.R.E.D.

A particular attention must be given to the Explore mode, which is the state in which the vehicle performs the filtering while moving towards the custom position. This is the scenario considered by the model.

Given this scenario, considering the scenario in which the vehicle placed in front of an obstacle, and considering that the vehicle can move only in a straight line, this is the reference system:

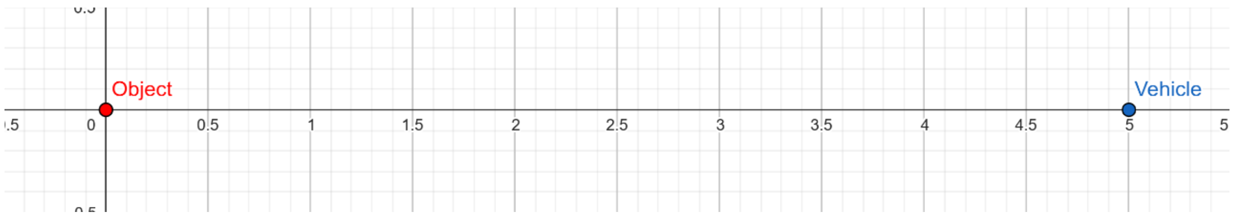


Figure 5.1 – Reference system of the scenario

The real representation of the scenario is shown in the figures below:

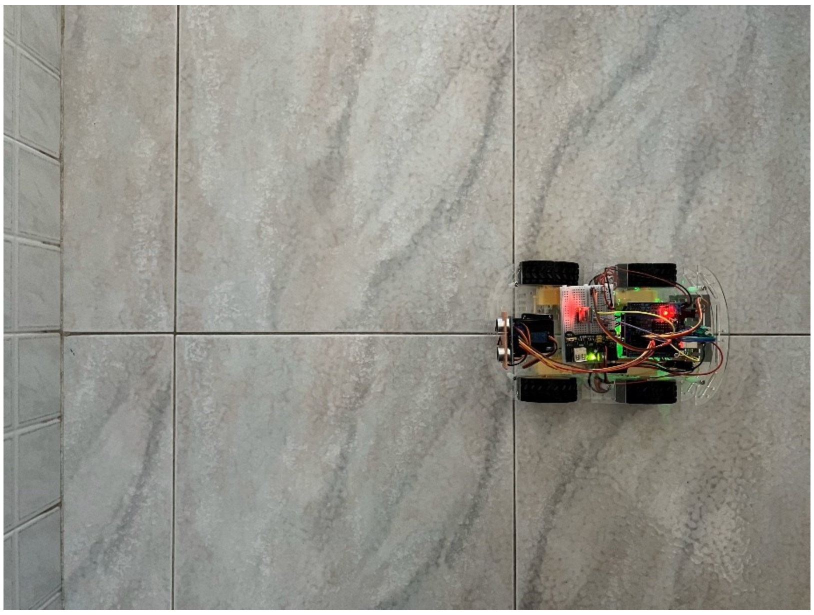


Figure 5.2 - Real scenario (upper view)

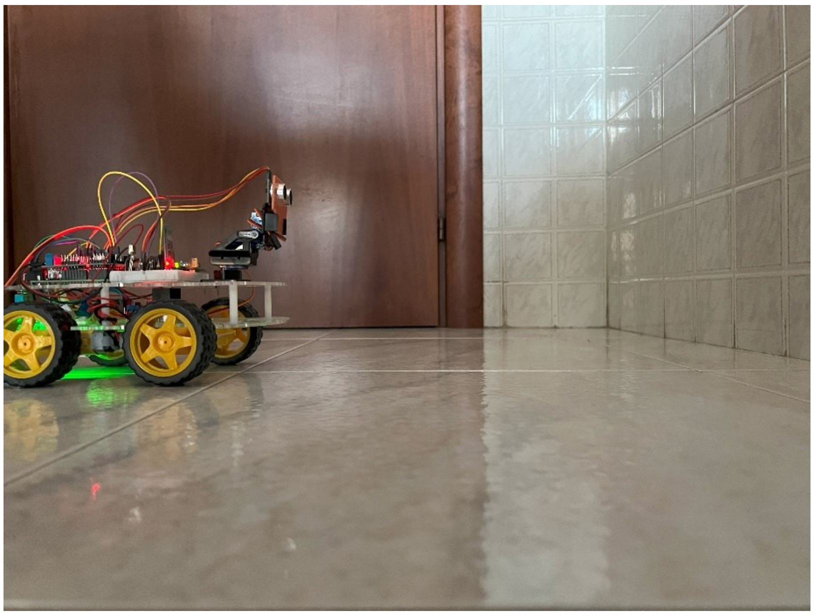


Figure 5.3 - Real scenario (side view)

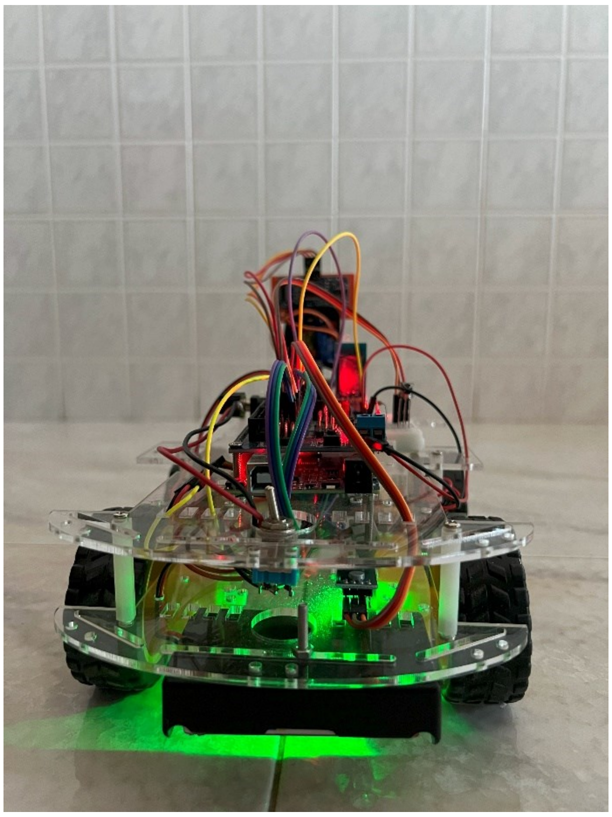


Figure 5.4 - Real scenario (back view)

5.2. MATLAB simulation

To simulate the real scenario, the following MATLAB scripts has been written:

- FRED_simulation.m: script that simulates the Explore mode. Given an objective position the simulated vehicle starts moving toward it while performing measurements and filtering. The script can also simulate the model descripted in Chapter 3.6.
- FRED_covariance.m: script that performs the off-line computations of the KF (covariance and gain).
- check_covariance.m: MATLAB custom defined function that check if a given matrix P satisfies the condition of being a covariance matrix (symmetric and semidefinite positive).

Simulation

The simulation script takes as input:

- All the physical parameters described in Chapter 3.4.
- The objective position obj.
- The discretization step T.
- The standard deviations of the noises.
- The initialization values of the model.
- Simulation time in seconds sec.

Performs the following operations:

- Compute the matrices of the dynamic model.
- Initialize the model as defined in Chapter 4.2.
- Simulate the real evolution of the vehicle (movement and noisy measurements).
- Perform the KF algorithm (predictor and corrector).
- Choose the input to provide based on the state of the robot.

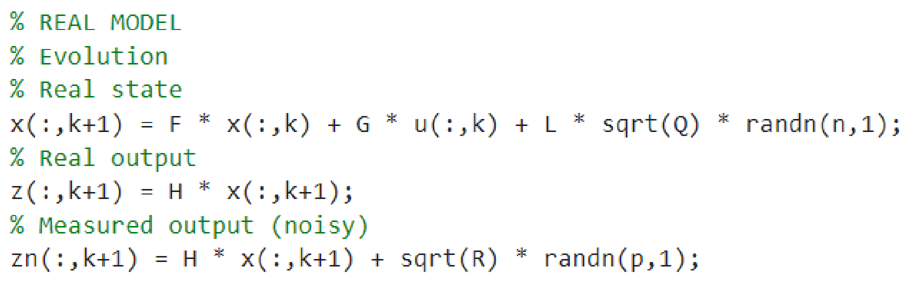
Provide the following textual outputs:

- Final simulated state.
- Final estimated simulated state.
- Final position error with respect to the objective position (real and estimated).

And provide the following graphs:

- Input over time.
- State over time (measured, estimated, real) in 2 graphs (position and velocity).
- Full trajectory of the vehicle in the reference system, with indication of the obstacle and the objective position.
- Distance thresholds[15] over time.
- Covariance over time in 2 graphs (main diagonal elements)
- State estimation error over time in 2 graphs (position and velocity).

Here also reported the code that simulate the trajectory and that performs the filtering:



```
% REAL MODEL
% Evolution
% Real state
x(:,k+1) = F * x(:,k) + G * u(:,k) + L * sqrt(Q) * randn(n,1);
% Real output
z(:,k+1) = H * x(:,k+1);
% Measured output (noisy)
zn(:,k+1) = H * x(:,k+1) + sqrt(R) * randn(p,1);
```

Figure 5.5 - MATLAB model simulation

[15] These thresholds are used to choose the correct input to provide. This topic is not treated in this document.

```
% KALMAN FILTER ESTIMATION
% Predictor
% Prediction: x(k+1|k)
x_pred(:,k+1) = F * x_hat(:,k) + G * u(:, k);
% Prediction covariance: P(k+1|k)
P_pred = F * P * F' + Q; % P[k+1|k]

% Corrector
% Gain
W = P_pred*H'/(H*P_pred*H'+R);
% Correction: x(k+1|k+1)
x_hat(:,k+1) = x_pred(:,k+1) + W*(zn(:,k+1)-H*x_pred(:,k+1));
% Correction covariance: P(k+1|k+1) (formulation 2)
P = (eye(n)-W*H)*P_pred;
```

Figure 5.6 - MATLAB implementation of KF

Off-line computations

The script called FRED_covariance.m performs just the off-line computations with a particular focus to the results inherent to the asymptotic behavior of the KF relative to the characteristics of the model. In fact, this script is like the previous one, but without all the commands inherent to the trajectory simulation but implements some additional checks to provide more information about covariance and gain.

Considering this, this script takes almost the same input as the previous one and performs these additional operations:

- Check the observability and controllability of the dynamic system[16].
- Check the integrity of the covariance matrix at each step[17].

Provide the following textual outputs:

- Final prediction covariance.
- Final estimation covariance.
- Final gain.

And provide the following graphs:

- Gain over time in 2 graphs (main diagonal elements)
- Covariance over time in 2 graphs (main diagonal elements)
- State estimation error over time in 2 graphs (position and velocity).

[16] This is done by applying the PBH test.

[17] This check, implemented in function checkCovariance, is done by checking that the matrix is symmetric and semidefinite positive. This is necessary due to potential problems related to numerical calculation.

5.3. Vehicle Kalman filter implementation

In this chapter will be provided more details about the code that is installed in the vehicle and that performs the filtering.

The sketches that can be uploaded to the vehicle are 2:

- EDA2023-Project-FRED.ino: main sketch that implement the true behavior of F.R.E.D.
- FRED-Params.ino: sketch used to perform experiments to estimate the parameters as described in Chapter 3.4.

Since the objective of this documentation is not the explanation of how to deal with a microcontroller, here it will only be described the way in which the Kalman filter has been implemented.

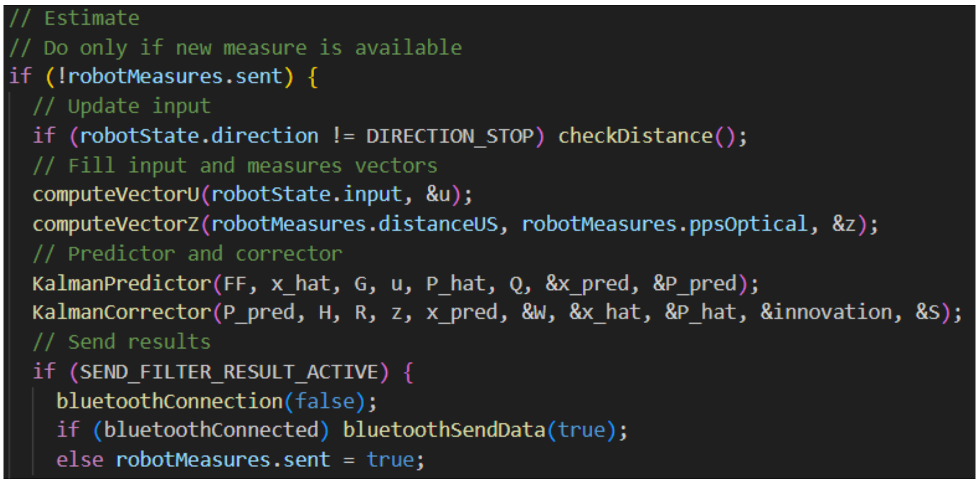
The sketch takes as input:

- All the physical parameters described in Chapter 3.4.
- The discretization step T.
- The standard deviations of the noises.
- The initialization values of the model.

The results obtained are sent to the connected device at each iteration and are the following:

- Input (scalar value).
- Measurements (vector of 2 elements).
- State (vector of 2 elements).
- Covariance (just the main diagonal)

The code that performs the filtering in the Explore state is shown in the figure below:



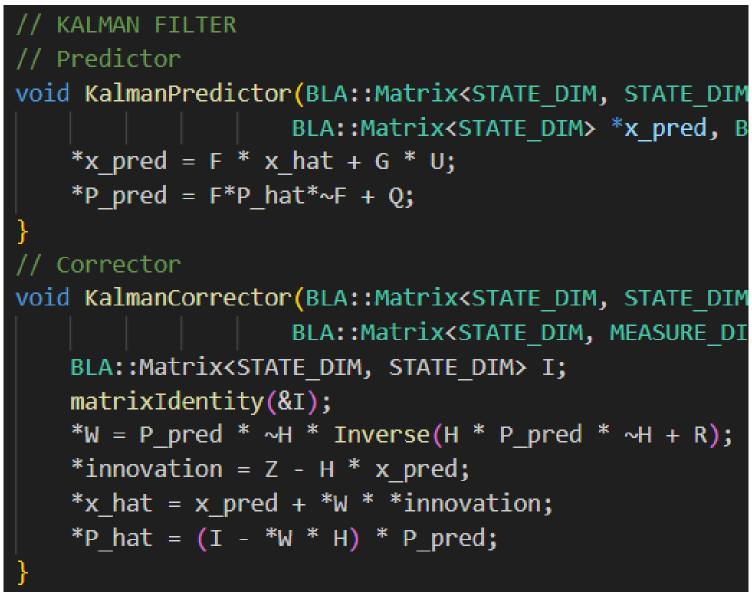
```
// Estimate
// Do only if new measure is available
if (!robotMeasures.sent) {
  // Update input
  if (robotState.direction != DIRECTION_STOP) checkDistance();
  // Fill input and measures vectors
  computeVectorU(robotState.input, &u);
  computeVectorZ(robotMeasures.distanceUS, robotMeasures.ppsOptical, &z);
  // Predictor and corrector
  KalmanPredictor(FF, x_hat, G, u, P_hat, Q, &x_pred, &P_pred);
  KalmanCorrector(P_pred, H, R, z, x_pred, &W, &x_hat, &P_hat, &innovation, &S);
  // Send results
  if (SEND_FILTER_RESULT_ACTIVE) {
    bluetoothConnection(false);
    if (bluetoothConnected) bluetoothSendData(true);
    else robotMeasures.sent = true;
```

Figure 5.7 - Filtering in Explore state

Notation: the matrix FF in the code shown in Figure 5.7 represents the matrix F (the name F generates a problem related to a macro defined for the microcontroller).

The functions computeVectorU and computeVectorZ just put the values of input and measurements in the right variables.

The functions that perform the prediction and correction steps are defined in a custom library called Estimation.h as follows and according to the definition provided in Chapter 4.1:



```
// KALMAN FILTER
// Predictor
void KalmanPredictor(BLA::Matrix<STATE_DIM, STATE_DIM
                    BLA::Matrix<STATE_DIM> *x_pred, B
    *x_pred = F * x_hat + G * U;
    *P_pred = F*P_hat*~F + Q;
}
// Corrector
void KalmanCorrector(BLA::Matrix<STATE_DIM, STATE_DIM
                    BLA::Matrix<STATE_DIM, MEASURE_DI
    BLA::Matrix<STATE_DIM, STATE_DIM> I;
    matrixIdentity(&I);
    *W = P_pred * ~H * Inverse(H * P_pred * ~H + R);
    *innovation = Z - H * x_pred;
    *x_hat = x_pred + *W * *innovation;
    *P_hat = (I - *W * H) * P_pred;
}
```

Figure 5.8 - Kalman Filter implementation

Where the notation is the following:

- $\sim A = A^T$, where A is a generic matrix.
- $Inverse(B) = B^{-1}$, where B is a nonsingular matrix.

All the information gathered by the vehicle are sent to the connected device at each step as can be seen in Figure 5.7.

5.4. Connected device and communication

The connected device can connect to the vehicle at startup or when the vehicle is already navigating. The Bluetooth connection and the handling of all the received messages are done using Python. To use the script, it is convenient to use the Bash script that performs some required operations[18].

The files are the following:

- FRED.sh: the Bash script that safely starts the Python script.

[18] To communicate with the vehicle, it is necessary to involve and configure the Bluetooth protocol RFCOMM (Radio Frequency Communication)

- FRED_data_transmission.py: the main Python script that establish the Bluetooth connection and handle[19] all the received data.

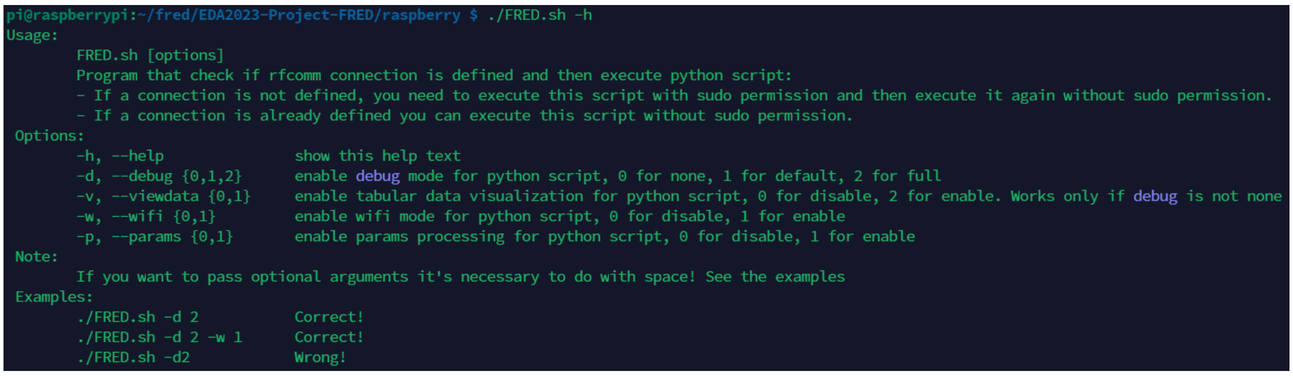
The Python script, which can be used for both filtering and experimentation scenarios, provides as output a csv file containing all the information received by the vehicle.

Considering the filtering scenario, the csv file is formatted with the following columns:

- created_at: contains the time in epoch (milliseconds).
- field1: values of the input.
- field2: values of the ultrasonic sensor measurements.
- field3: values of the optical sensor measurements.
- field4: values of the position estimate.
- field5: values of the velocity estimate.
- field6: values of the position covariance.
- field7: values of the velocity covariance.
- field8: values of the objective position.
- status: values of the vehicle mode (with reference to F.R.E.D.).

The column names have been chosen to match the names required by ThingSpeak. In fact, the script, in addition to providing the csv file, sends the data to the remote server.

The Bash script can be executed with -h option and gives as output a help message, reported here to give a few other information about the capabilities of the script:



```
pi@raspberrypi:~/fred/EDA2023-Project-FRED/raspberry $ ./FRED.sh -h
Usage:
        FRED.sh [options]
        Program that check if rfcomm connection is defined and then execute python script:
        - If a connection is not defined, you need to execute this script with sudo permission and then execute it again without sudo permission.
        - If a connection is already defined you can execute this script without sudo permission.
 Options:
        -h, --help              show this help text
        -d, --debug {0,1,2}     enable debug mode for python script, 0 for none, 1 for default, 2 for full
        -v, --viewdata {0,1}    enable tabular data visualization for python script, 0 for disable, 2 for enable. Works only if debug is not none
        -w, --wifi {0,1}        enable wifi mode for python script, 0 for disable, 1 for enable
        -p, --params {0,1}      enable params processing for python script, 0 for disable, 1 for enable
 Note:
        If you want to pass optional arguments it's necessary to do with space! See the examples
 Examples:
        ./FRED.sh -d 2          Correct!
        ./FRED.sh -d 2 -w 1     Correct!
        ./FRED.sh -d2           Wrong!
```

Figure 5.9 - FRED.sh help message

5.5. Visualization of results

The visualization of the results can happen in two different ways:

- Using MATLAB to read the csv file provided by the connected device or to access directly to the remote server.
- Using the dashboard provided by the remote server.

[19] The handling of the messages uses a tiny custom defined protocol that defines 5 types of messages that are structured in a specific way.

MATLAB

To view the results the following MATLAB scripts have been written:

- view_data_source.m: script that shows the results of the filtering reading the data from a local csv or from ThingSpeak.
- view_params.m: script that shows the results of the experimentation done to estimate the parameters.
- Constants.m: file used to define the customized class Constants to make it easier to insert options into the main script.

The view_data_source.m script can read data from a csv generated by the connected device (SOURCE_LOCAL), a csv exported from the remote server (SOURCE_EXPORTED) or directly from ThingSpeak (SOURCE_REMOTE).

Regardless of the source, the output will be always the same and is given by the following textual output:

- Starting estimated state.
- Final estimated simulated state.
- Final position error with respect to the objective position (real and estimated).

And the following graphs:

- Input over time.
- State over time (measured vs estimated).
- Full trajectory of the vehicle in the reference system, with indication of the obstacle and the objective position.
- State covariances.

ThingSpeak

Using the ThingSpeak platform as remote server allows to provide a user-friendly visualization of the data gathered from the vehicle, without the direct use of any MATLAB script.

As said in Chapter 2.4, ThingSpeak allows to create a channel that holds all the data for a specific context. The channel created for this project is called FRED Data (Channel ID 2391724).

ThingSpeak has three main limitations:

- It has a minimum time resolution equal to one second. The implementation involves considering only the first data received for each second.
- It can accept a new update message only each 15 seconds. This limitation can be solved using the bulk update API[20] (that allows to send a bulk of data in a single

[20] More information in the official API Reference here https://ch.mathworks.com/help/thingspeak/channels-and-charts-api.html?s_tid=CRUX_lftnav.

message), but it reduces the speed at which the data is displayed (since it needs to be processed).

- Each channel can hold just 8 different data (scalar value) called field. Furthermore, the API requires a JSON formatted as seen when talking about the csv in the Chapter above.

The dashboard is characterized by the following graphs:

- Input over time.
- Ultrasonic measure over time.
- Optical measure over time.
- Position estimate over time.
- Velocity estimate over time.
- Position covariance over time.
- Velocity covariance over time.
- Vehicle location (considered fixed and located at University of Salento).
- Channel status Updates.
- Comparison between measured and estimated position over time.
- Comparison between measured and estimated velocity over time.
- Input and position over time in the same graph.
- Input and velocity over time in the same graph.

Where the last four graphs have been generated using the MATLAB Visualizations app feature[21] provided by ThingSpeak.

[21] This feature allows to write small MATLAB scripts that can display one graph for each apps. More information at the official guide here: https://ch.mathworks.com/help/thingspeak/matlab-visualizations-app.html.

6. Results

In this chapter the results obtained will be provided and commented. First the results obtained in MATLAB simulation and then those obtained in the real scenario.

6.1. MATLAB simulation

In the FRED_simulation.m MATLAB script the following input has been provided:

- Physical parameters as described in Chapter 3.4.
- Objective position $obj = 10$.
- Discretization step $T = 0.1$.
- Standard deviations of the noises as described in Chapters 3.3 and 4.2.
- Initialization values of the model as described in Chapter 4.2.
- Simulation time in seconds $sec = 10$.

These choices of input leads to the following results:

```
Objective position: 10
Real starting position: 202
Real starting velocity: 0
Real final position: 9.7423
Estimated final position: 9.8542
Real position error (wrt obj): 0.2577
Estimated position error (wrt obj): 0.1458
Real final velocity: -0.0064346
Estimated final velocity: 0.03776
```

Figure 6.1 - MATLAB simulation textual results

And to the following graphs:

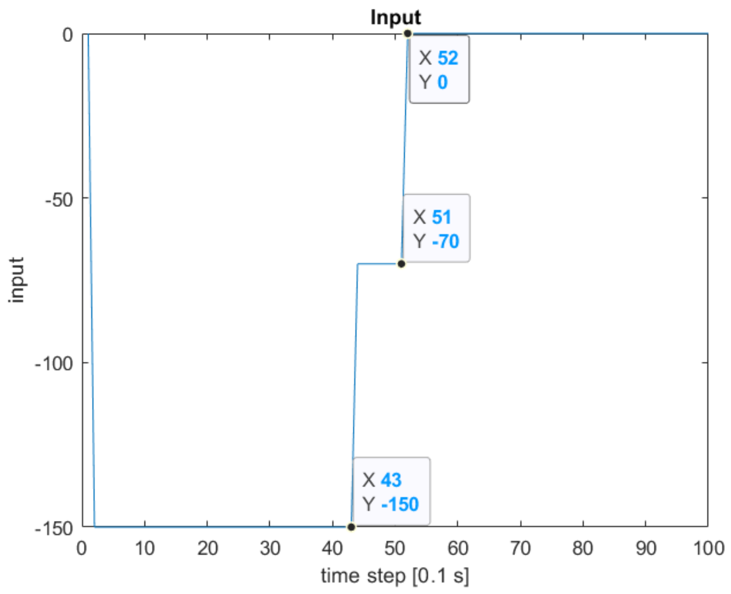


Figure 6.2 - MATLAB simulation input over time

As seen in the figure above, the simulation controls the input with 3 values: one higher (in module) value to approach the objective faster, one lower (in module) to reach the desired position with more precision and the null input to stop.

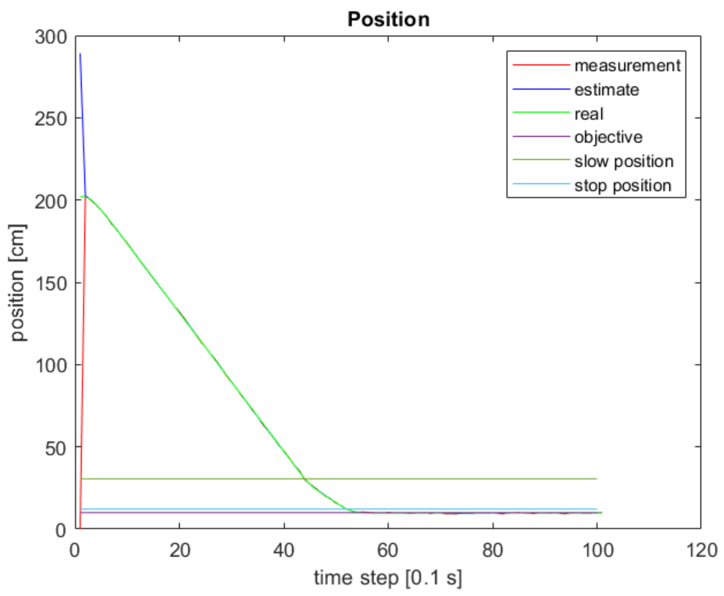


Figure 6.3 - MATLAB simulation position over time

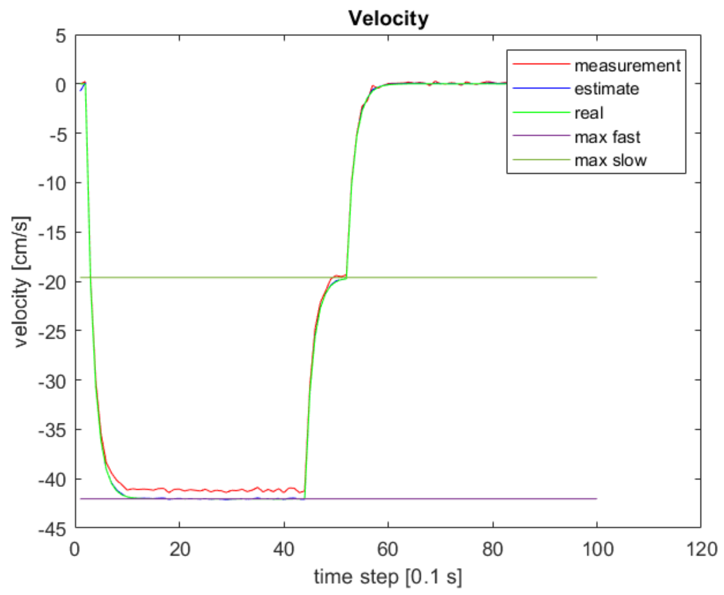


Figure 6.4 - MATLAB simulation velocity over time

In the position over time graph there are two lines parallel to the x axis that indicate the position in which the vehicle changes its input from high to low (slow position) and then to null (stop position).

Also, in the velocity over time graph there are two lines as the position graph. In this case they indicate the max value of velocity for higher (max fast) and lower input (max slow).

In both graphs it is possible to see the real (simulated) state, the estimated state and the measurement taken. Note that, in the velocity graph, the measure refers to the pulses and so there is a gap between the measure and the real/estimate due to the effective difference of magnitude[22] between these two quantities.

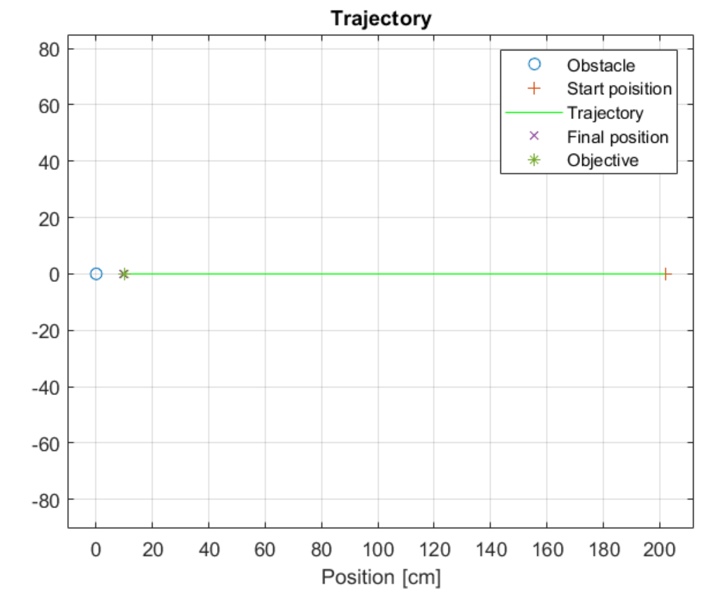


Figure 6.5 - MATLAB simulation vehicle trajectory

The trajectory graph refers to the real value of position.

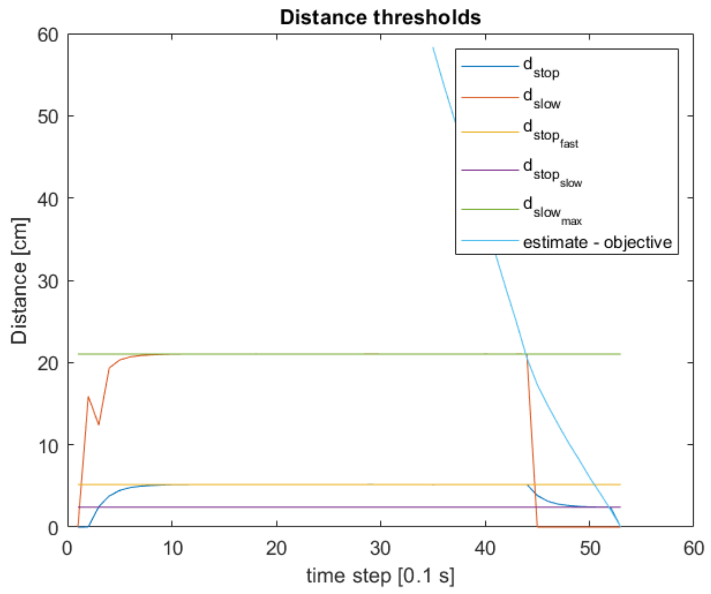


Figure 6.6 - MATLAB simulation distance thresholds over time

The distance thresholds graph shows how the vehicle choose the input in relation to the difference between the actual position and the objective. These thresholds change over time in proportion to velocity (intuitively, higher the velocity, before the vehicle needs to provide the null input) and reach a stable value when the velocity does not change.

[22] The difference is given by the terms $H_{22} = \frac{PPR}{\pi D}$

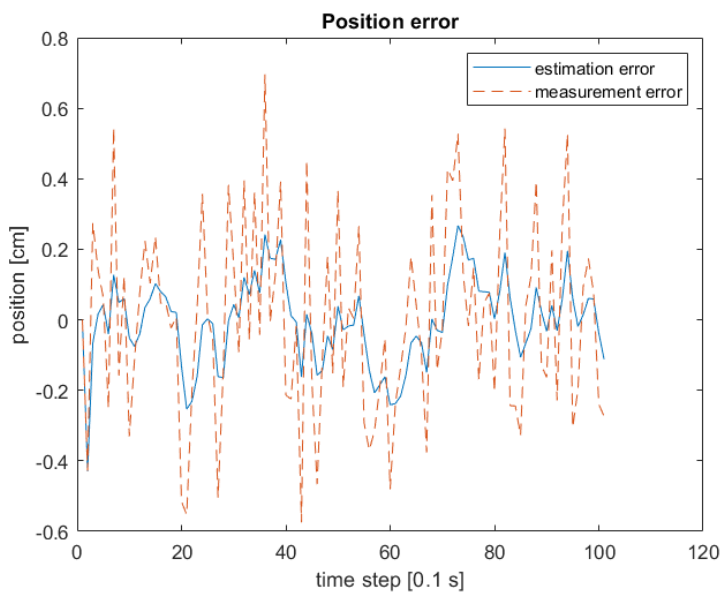


Figure 6.7 - MATLAB simulation position error over time

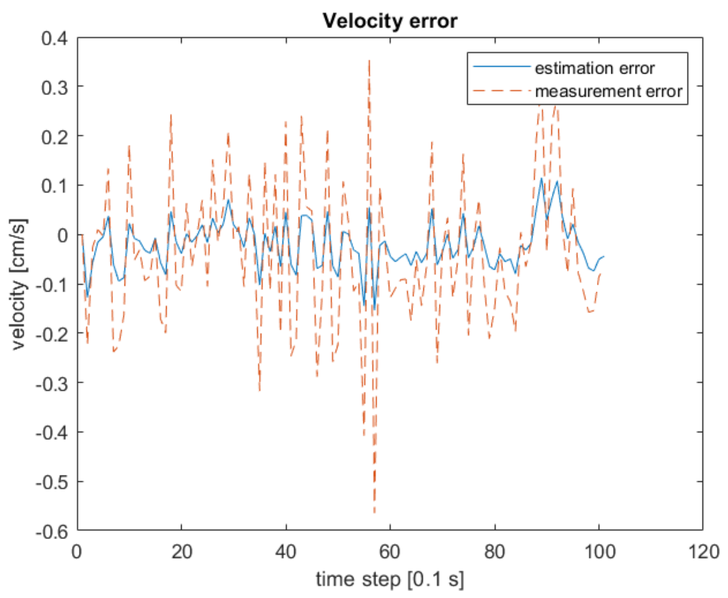


Figure 6.8 - MATLAB simulation velocity error over time

In the last two graphs it is possible to see the estimation and the measurement error. The first one is referred to the difference between the real state and the estimate. The second one is referred to the difference between the clean measure and the noisy one.

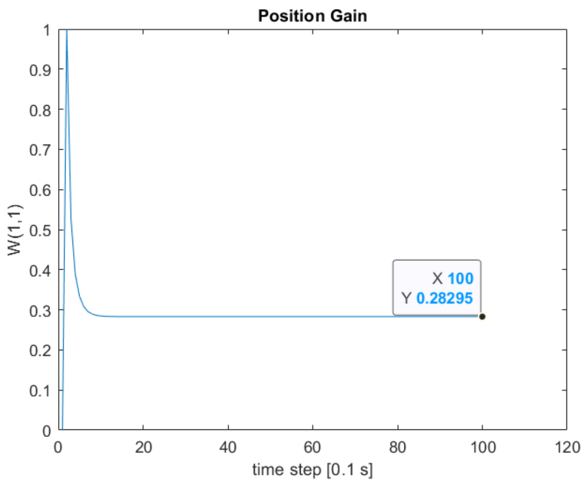
It is possible to see how the Kalman filter reduce the error done by the measurements.

To provide the results about the estimation covariance and gain it has been used the script FRED_covariance.m that, using the same parameters as before, provides the following textual results:

```
F matrix not stable
System observable
System reachable

Covariance and gain converge

Final prediction covariance
    0.0355    0.0004
    0.0004    0.0123

Final estimation covariance:
    0.0255    0.0002
    0.0002    0.0086

Final gain:
    0.2829    0.0064
    0.0020    0.3046
```

Figure 6.9 - MATLAB simulation covariance and gain textual results

And the following graphs:



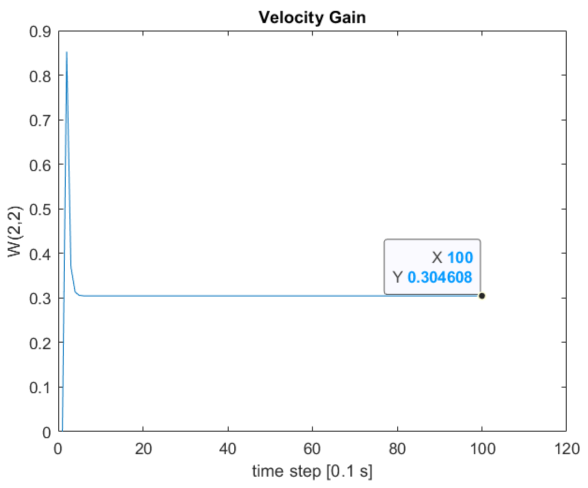
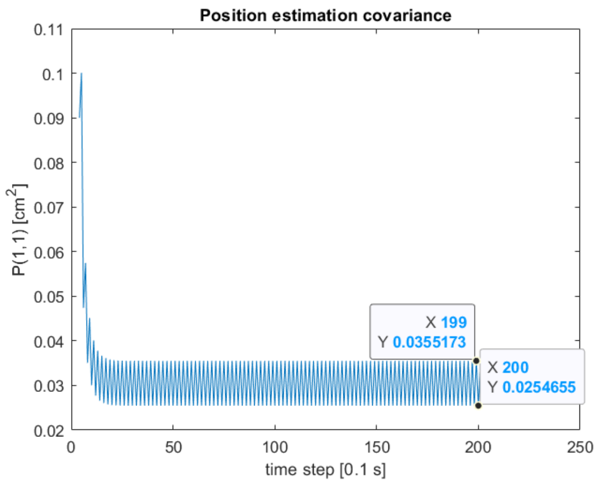


Figure 6.10 - MATLAB simulation gain



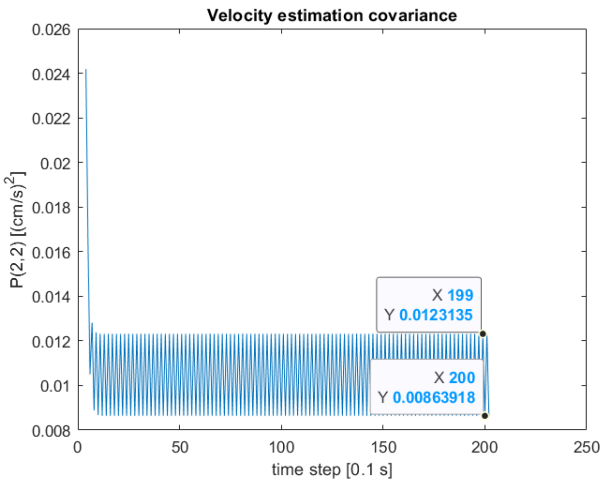


Figure 6.11 - MATLAB simulation estimation covariance

As seen in Chapter 4.3, the gain and the covariance converge. The oscillation visible in the covariance graphs is caused by the prediction step that increases the covariance until the correction step.

Observe that the graphs shown in Figure 6.10 and Figure 6.11 do not start from the first instant of time because the values are too high at the beginning and do not allow to appreciate the results.

6.2. Real scenario

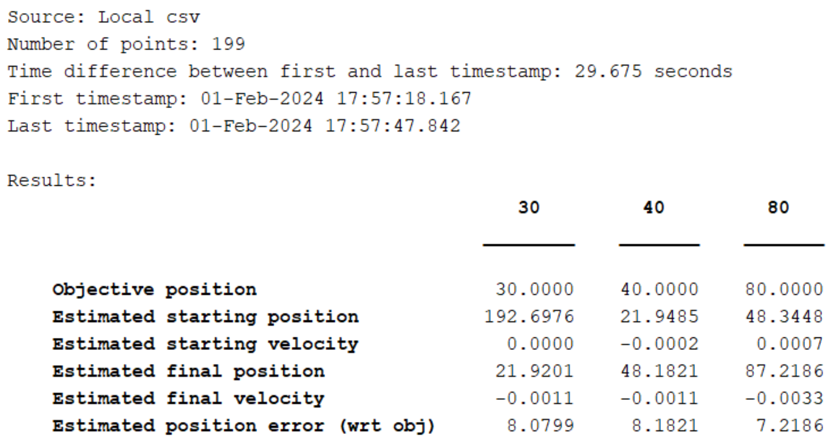
In the EDA2023-Project-FRED.ino sketch the following inputs has been provided:

- Physical parameters as described in Chapter 3.4.
- Discretization step $T = 0.1$.
- Standard deviations of the noises as described in Chapters 3.3 and 4.2.
- Initialization values of the model as described in Chapter 4.2.

The experiment is done providing 3 different objective positions in sequence: 30, 40, 80.

Local view

The results obtained analyzing the local csv file, generated by the connected device, are the following:



```
Source: Local csv
Number of points: 199
Time difference between first and last timestamp: 29.675 seconds
First timestamp: 01-Feb-2024 17:57:18.167
Last timestamp: 01-Feb-2024 17:57:47.842

Results:
                                          30         40         80
                                       ________   ________   ________

    Objective position                  30.0000    40.0000    80.0000
    Estimated starting position        192.6976    21.9485    48.3448
    Estimated starting velocity          0.0000    -0.0002     0.0007
    Estimated final position            21.9201    48.1821    87.2186
    Estimated final velocity            -0.0011    -0.0011    -0.0033
    Estimated position error (wrt obj)   8.0799     8.1821     7.2186
```

Figure 6.12 - Real scenario textual results

And the following graphs:

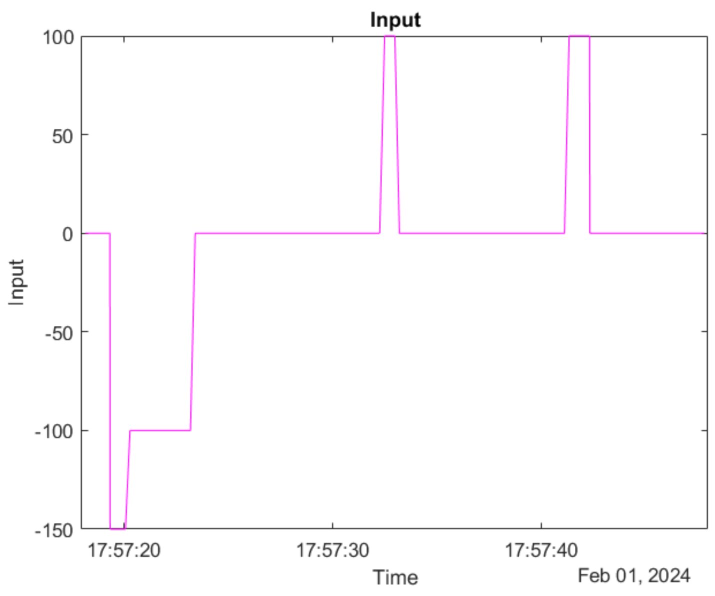


Figure 6.13 - Real scenario input over time

The input is chosen in an equivalent way as the MATLAB simulation.

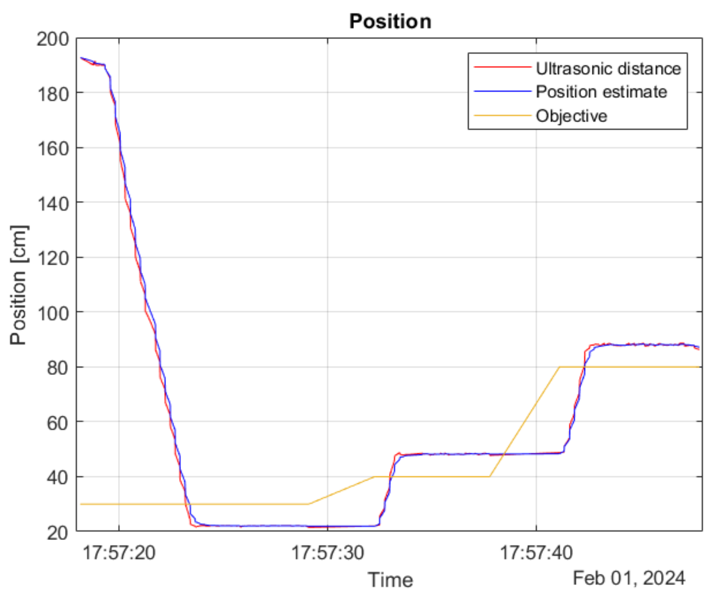


Figure 6.14 - Real scenario position over time

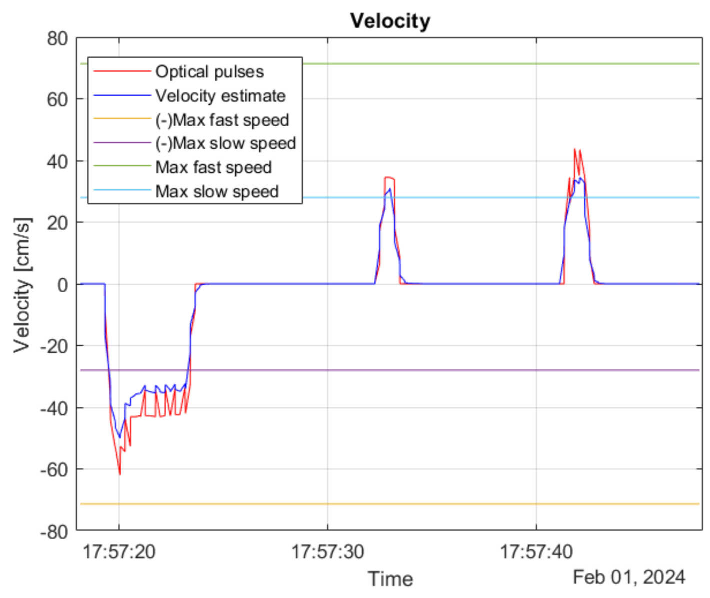


Figure 6.15 - Real scenario velocity over time

In the position over time graph the yellow line represents the objective position over time.

In the velocity over time graph the lines parallel to the time axes represents the theoretical max velocity corresponding to a high and low input (respectively, starting from the green to the yellow line, the values corresponding to high positive input, low positive input, low negative input, and high negative input).

In both graphs it is possible to see the estimated state and the measurement taken. Also in this case, in the velocity graph, the measure refers to the pulses and so there is a gap. In this case is not possible to plot the real value of the state because in a real scenario it is not known.

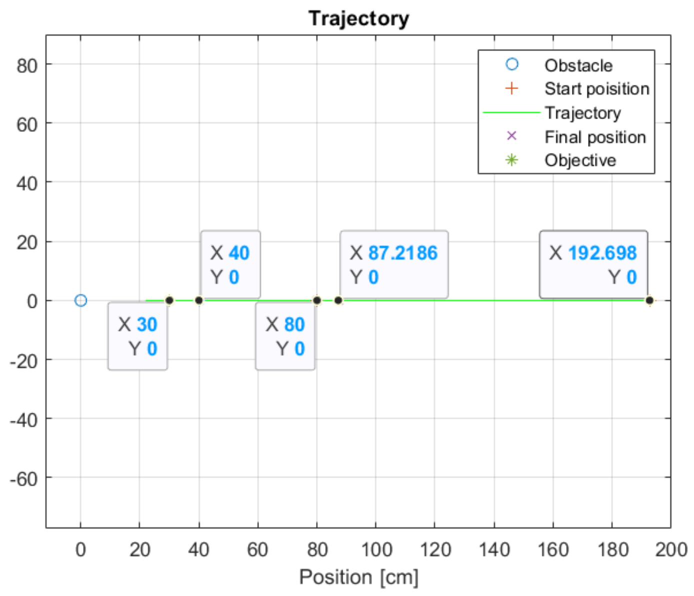


Figure 6.16 - Real scenario trajectory

In the trajectory graph it is possible to see the starting position (the rightest one) the three objective positions and the final position.

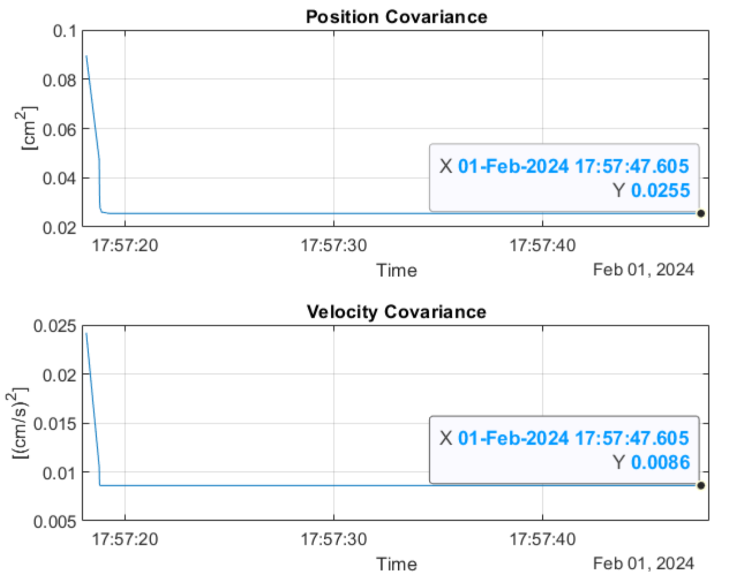


Figure 6.17 - Real scenario estimation covariance

The covariance graph shows the estimation covariance of the state (just the diagonal). The results are correctly similar as the simulation.

The results obtained relating to vehicle control are not very satisfactory but can be improved applying a better control technique (that is not an objective of this project.

Instead, the results obtained with the implementation of the Kalman filter, can be considered good. In fact, looking at Figure 6.3, Figure 6.4, and at Figure 6.7,Figure 6.8, it is possible to see how the estimate follows better the real state.

Regarding the real scenario it is harder to comment the results since the real state is not available. In general, it is possible to say that the estimate appears to be less noisy. The results can be improved enhancing the model doing better estimation of the parameters.

ThingSpeak dashboard

The same results can be seen using the ThingSpeak dashboard available here: https://thingspeak.com/channels/2391724.

Here just some graphs for completeness:

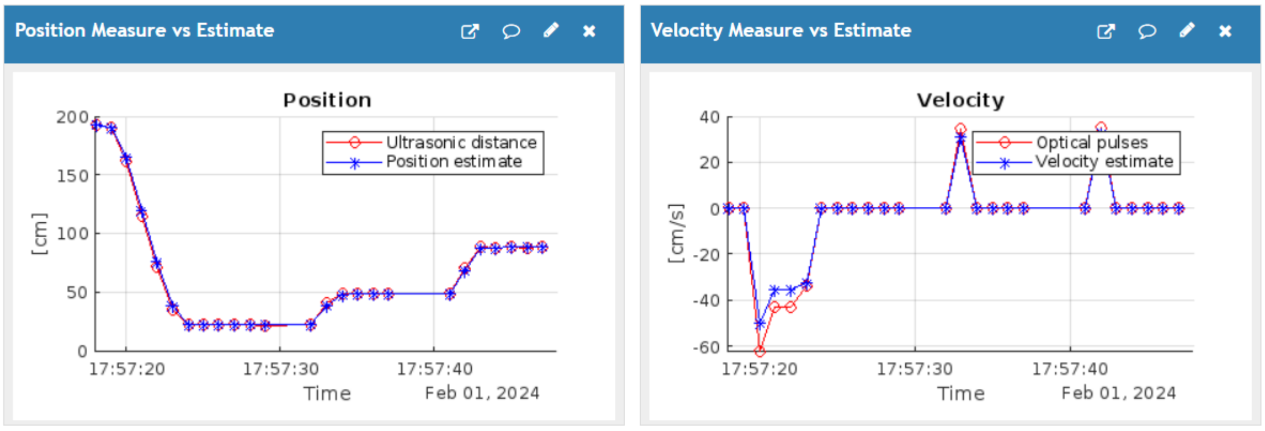


Figure 6.18 - ThingSpeak state visualization

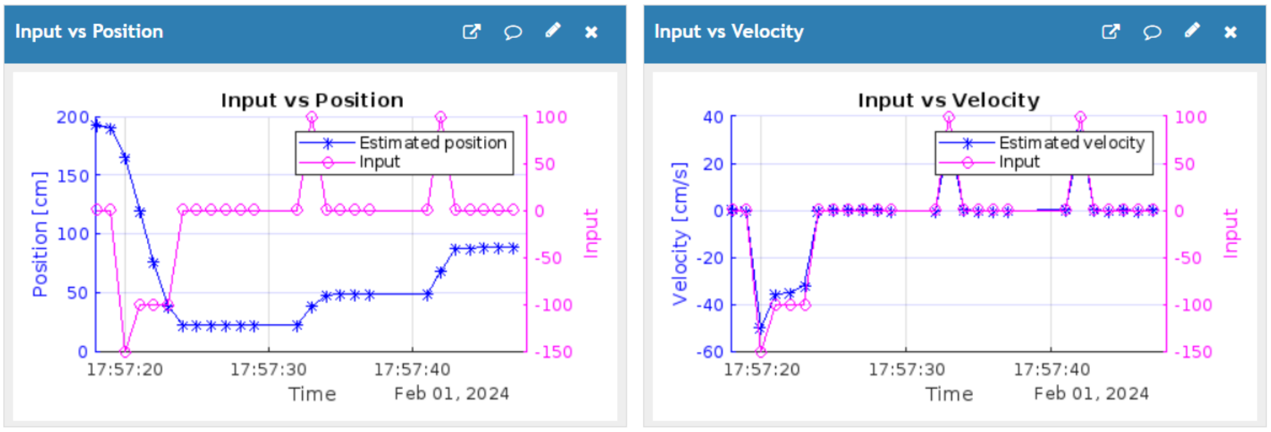


Figure 6.19 - ThingSpeak input vs state

Note that, in the ThingSpeak visualization, the data is taken at a rate of one second. So each point corresponds to one second.