COEX-1

Cooperative Explorer

Aurélien Werenne
Advisor: Prof. B. Boigelot

University of Liège

Link to code: Github

March 16, 2019

Overview

1. General
2. Components
3. Control
4. Exploration & Mapping
5. Code

Overview

1. General
2. Components
3. Control
4. Exploration & Mapping
5. Code

The goal of the robot, named COEX-1, is to perform a mapping of an unknown environment. The robot cooperates in a centralized multi-agent setting.

The main features of the robot are:

- Following a black line
- Computing the travelled distance
- Detecting, classifying and handling intersections
- Avoiding obstacles
- Communication with the central unit

Non-exhaustive list of used components:

- Microcontroller (Arduino Nano)
- Reflectance sensor array (Pololu QTR-MD-06A)
- Digital distance sensor (Sharp GP2Y)
- Magnetic encoders
- DC Motor (Pololu 150:1 micro metal gearmotor MP)
- Motor driver (L298N dual H-bridge)
- Battery (NiMH 7.2V)
- Bluetooth module (HC-05)

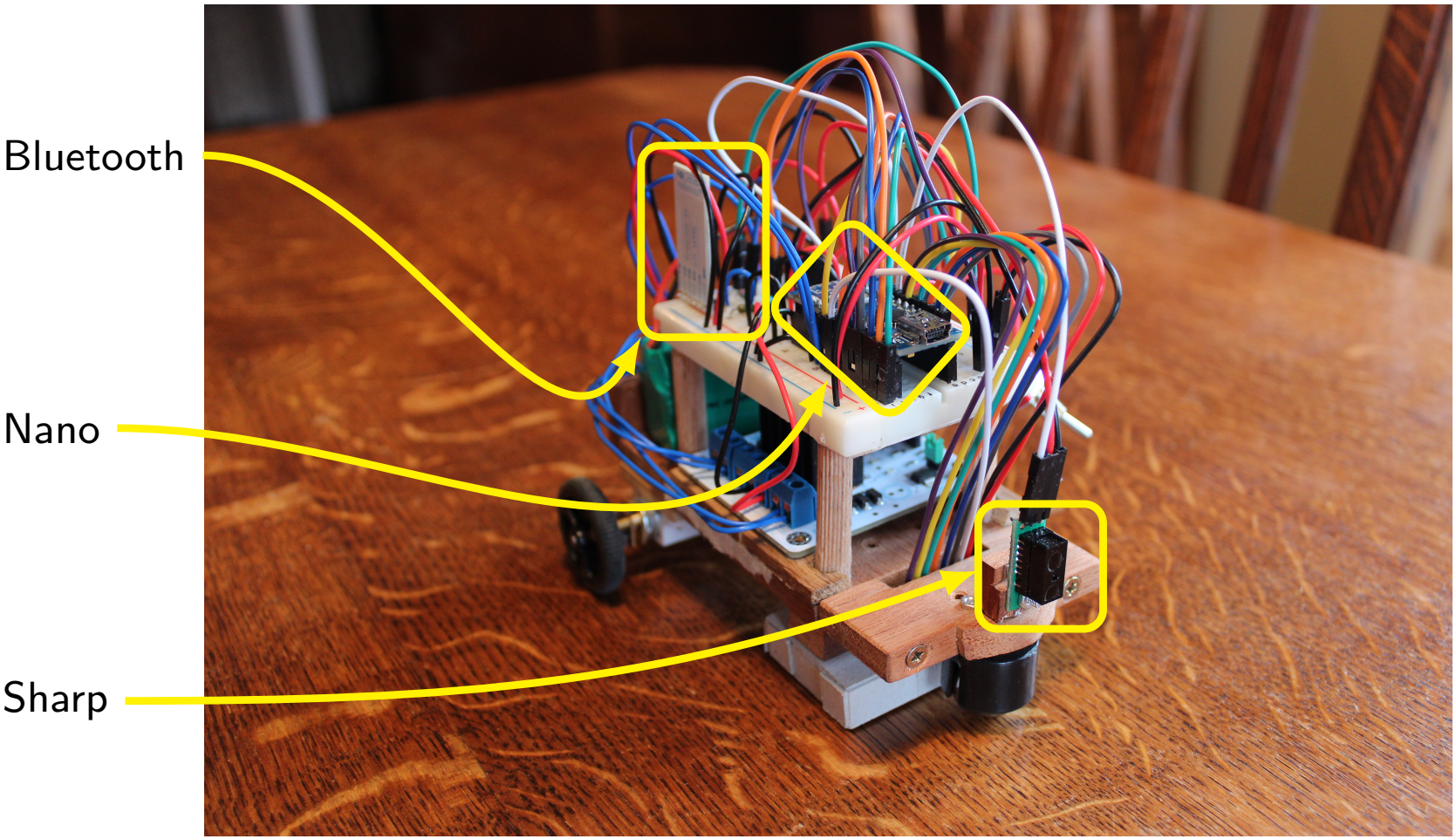


Figure: Front view of COEX-1

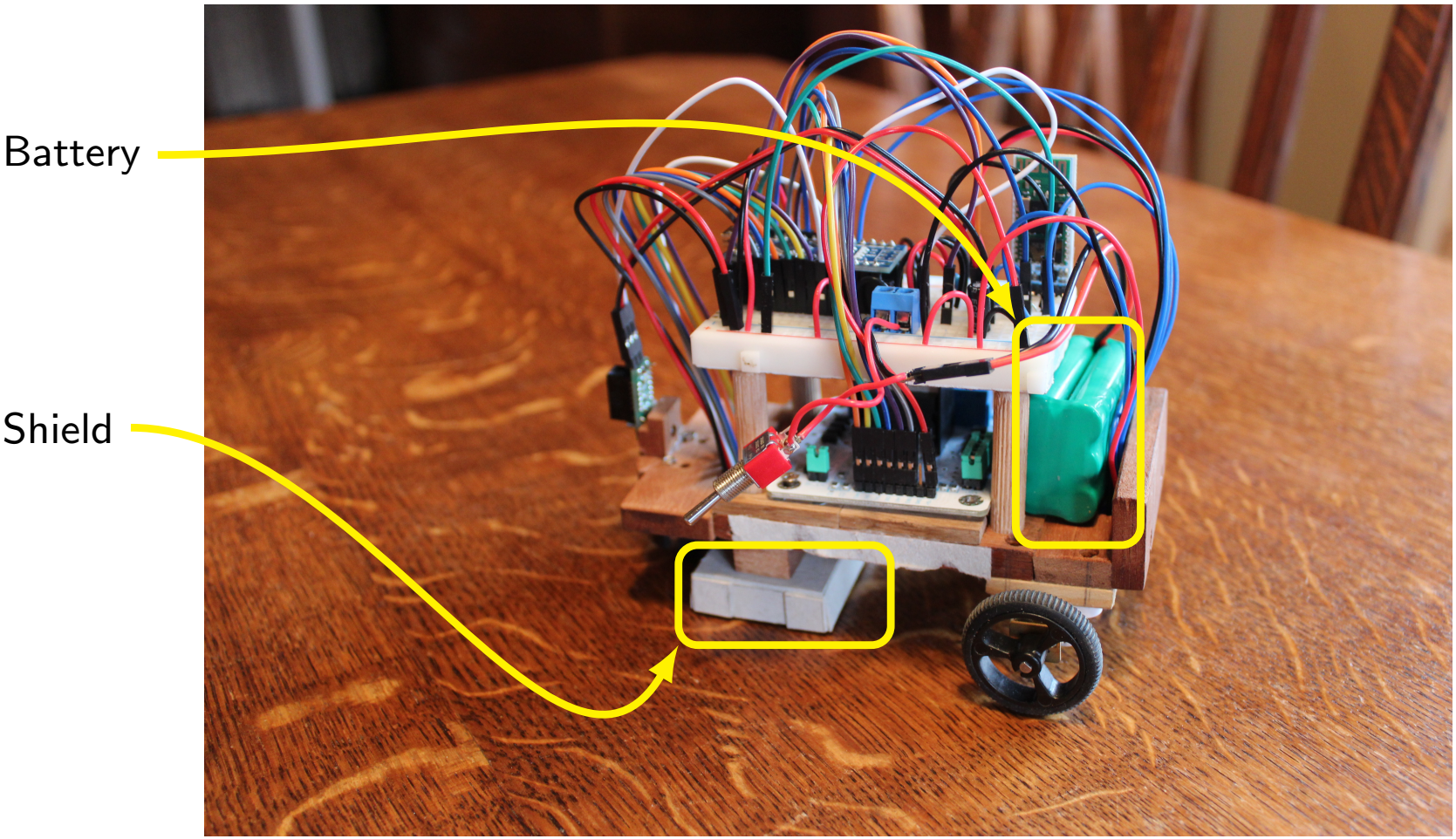


Figure: Side view of COEX-1

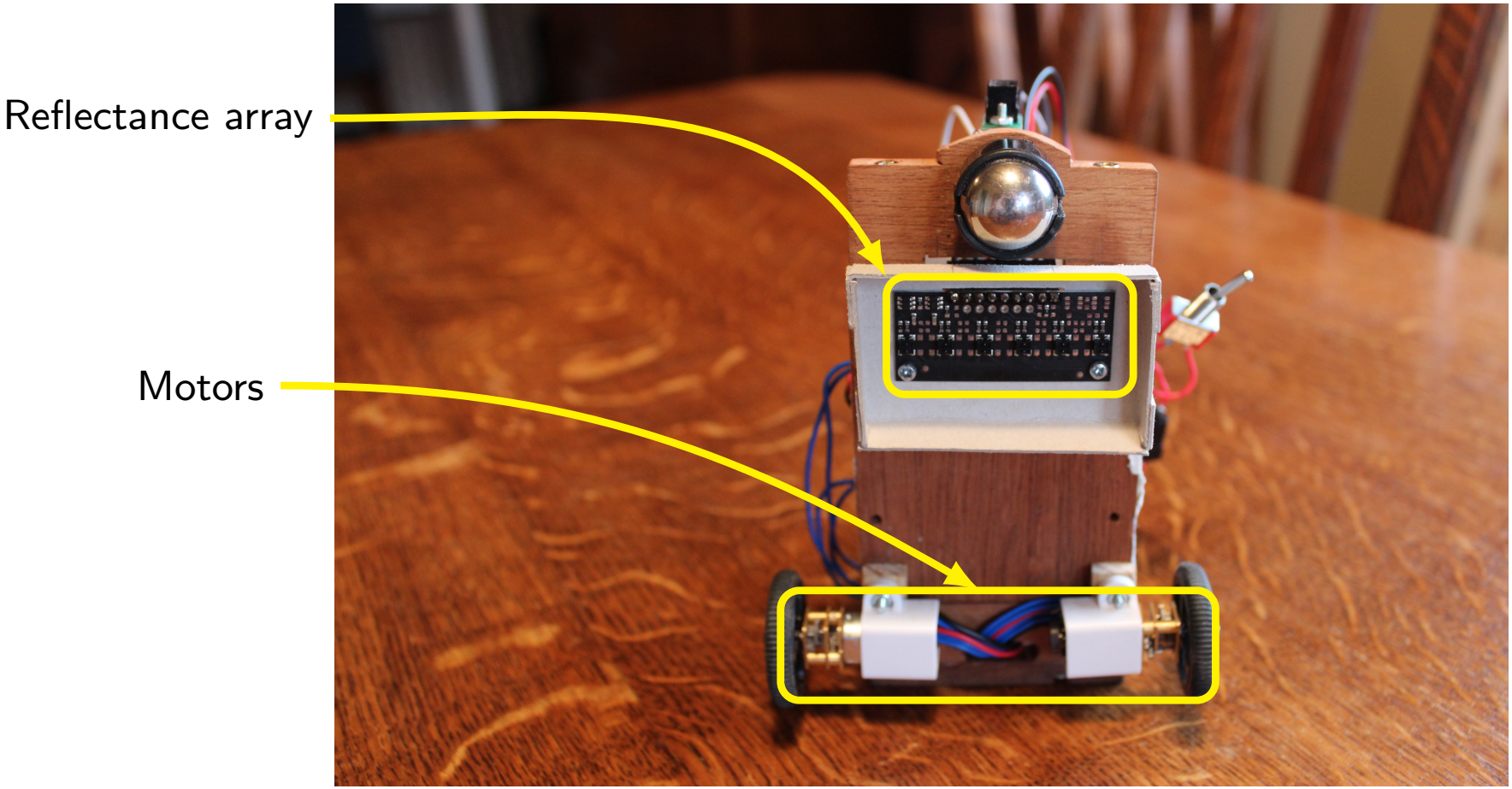


Figure: Bottom view of COEX-1

Let us define some terminology,

- *Error line*: the deviation (in cm) of the detected black line from the center of the sensor array ($err_{line} \in [-2; 2]$).
- *Target speed*: the speed the robot wants to achieve.
- *Progress speed*: an intermediate value of speed whose value is in between the previous and the current target speed. Serves as a proxy to smoothly control acceleration.
- *Measured speed*: the speed measured by the encoder, i.e. the actual speed of COEX-1.

Overview

1 General

2 Components

3 Control

4 Exploration & Mapping

5 Code

Adding a bypass capacitor was necessary to make the Sharp sensor work. On Fig.**??** the detection of obstacles is read perfectly. The threshold is set at 2 Volt instead of the mean in order to avoid False Negatives.

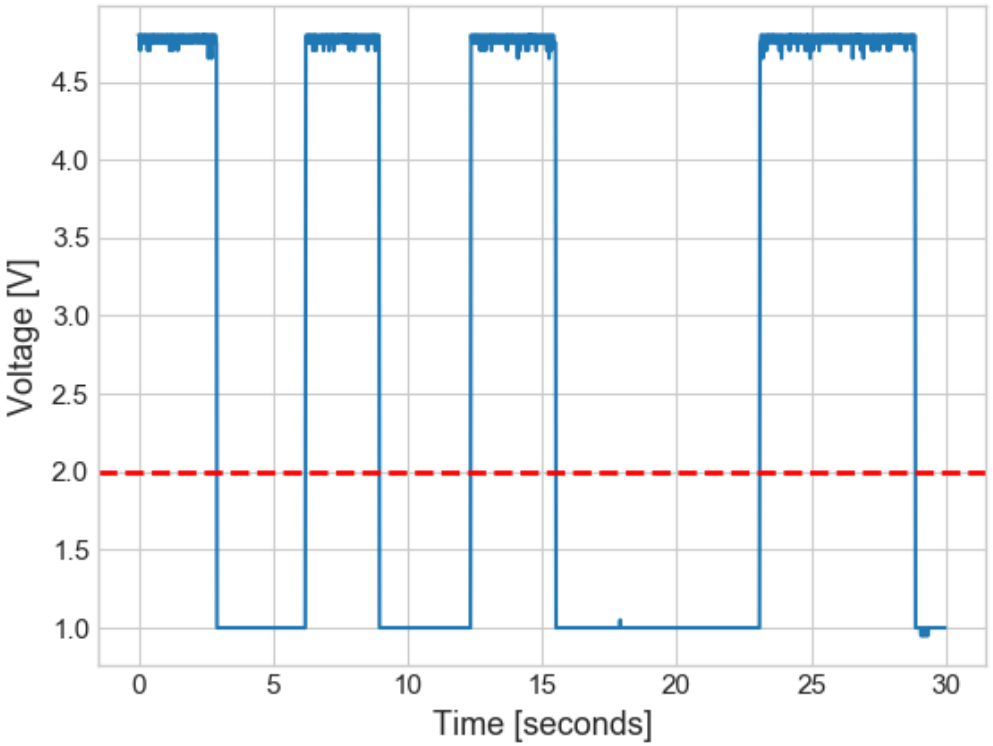


Figure: Sharp sensor output, $f = 20Hz$. Test performed with back and forth movement of obstacle, going in and out of reach of the sensor.

Experimentally, I found it is best to average four readings of the sensor to reduce noise. By doing so, we can observe the sensor's output varying smoothly.

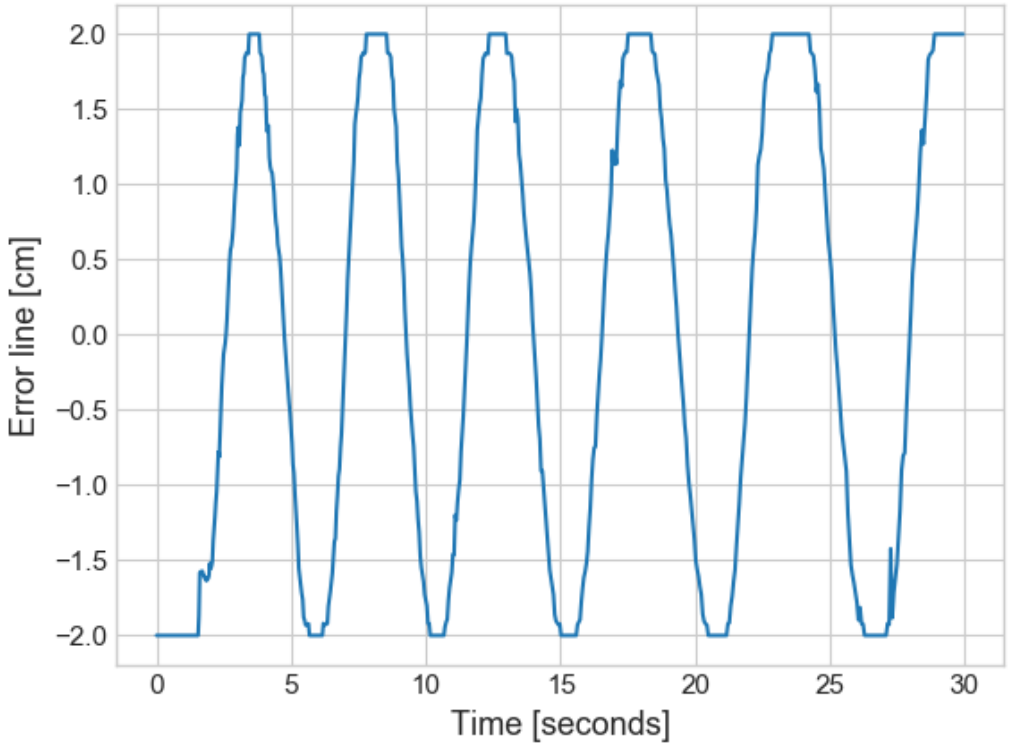


Figure: QTR sensor output, $f = 100Hz$. Test performed by moving the black line back and forth between left and right side of the sensor.

The robot is designed such that it is able of going up and downhill (at moderate inclinations). To do so the reflectance array is fixed relative to the ground well above the recommendations of the manufacturer.

This decision made the sensor particularly sensitive to ambient light interferences. As a solution a shield was constructed around the sensor (Fig.8).

To evaluate the limits of the Bluetooth module and the communication protocol, a simple test is performed as described in Fig.**??**.

Te function send $f(t) = A\sin(2\frac{\pi}{T}t) + b$, is of the sinus form because it enable us to easily deduce delay, distortion by regressing.

We conclude that 5Hz works well and is enough for our use case.

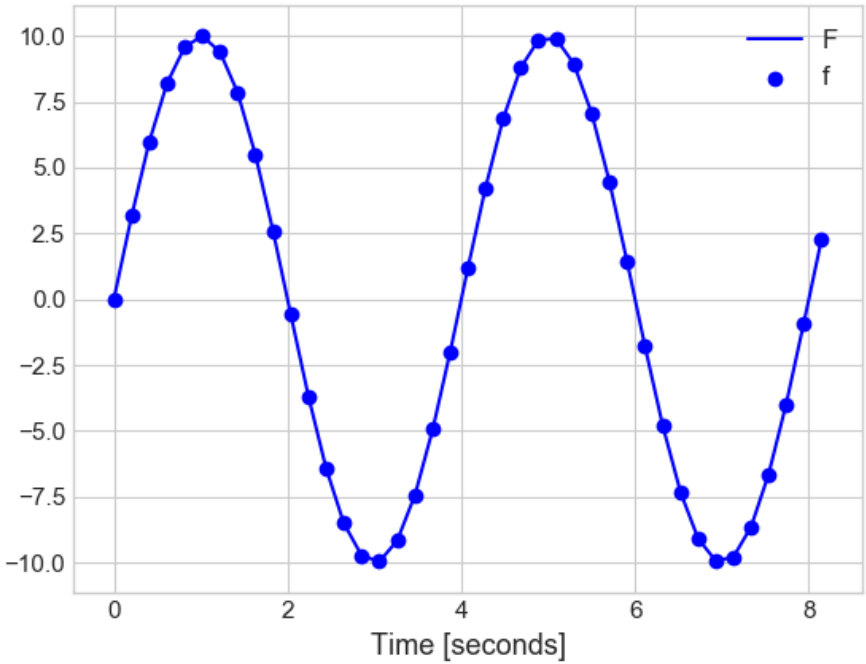


Figure: Sinus test at 5Hz, input signal

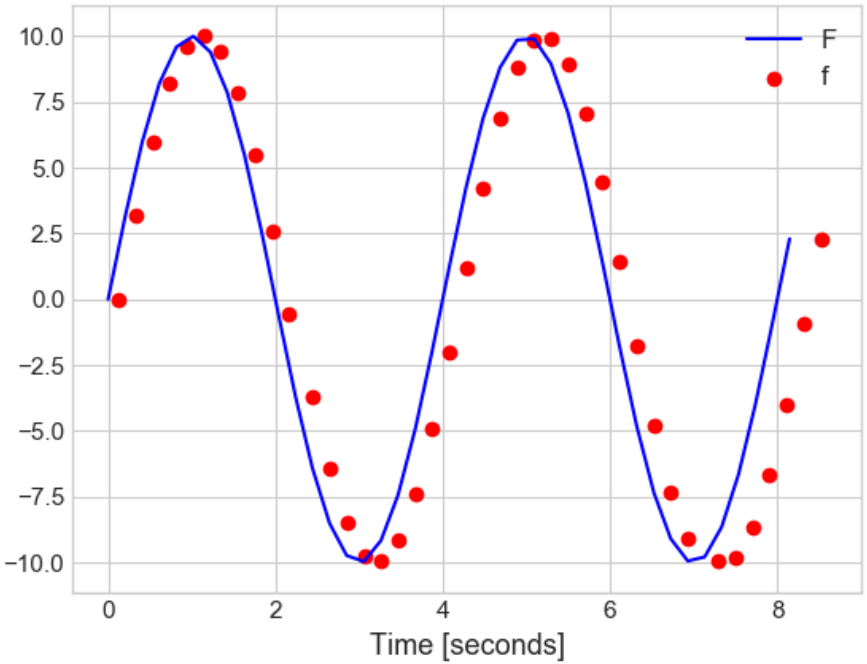


Figure: Sinus test at 5Hz, output signal

Some loss is observed at 25Hz.

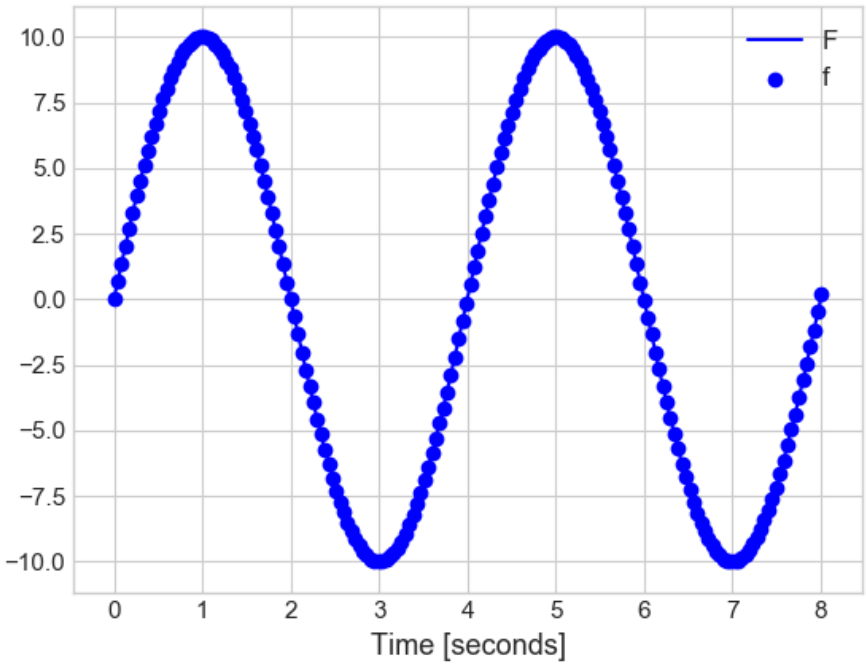


Figure: Sinus test at 25Hz, input signal

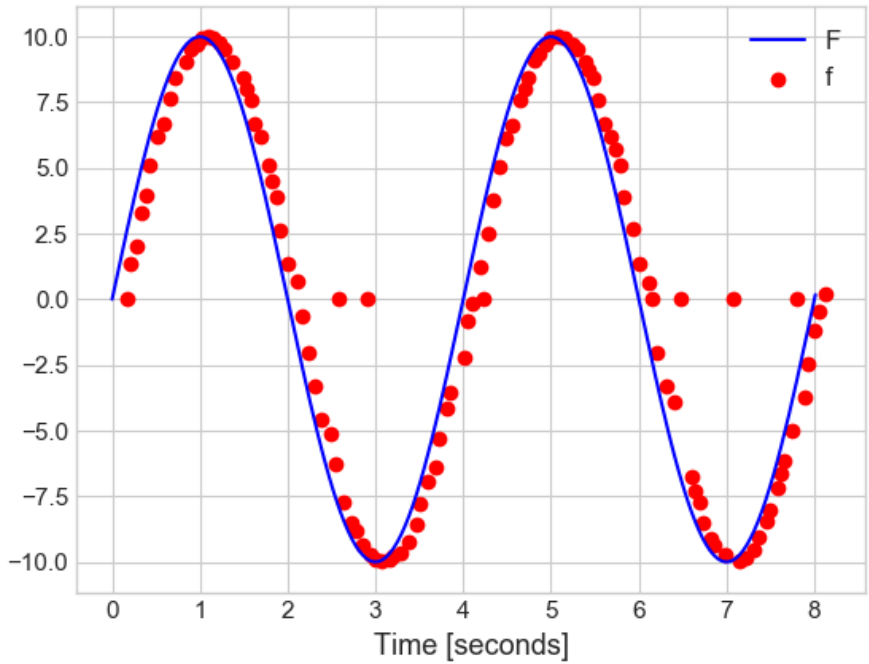


Figure: Sinus test at 25Hz, output signal

At 40Hz, our protocol completely breaks down.

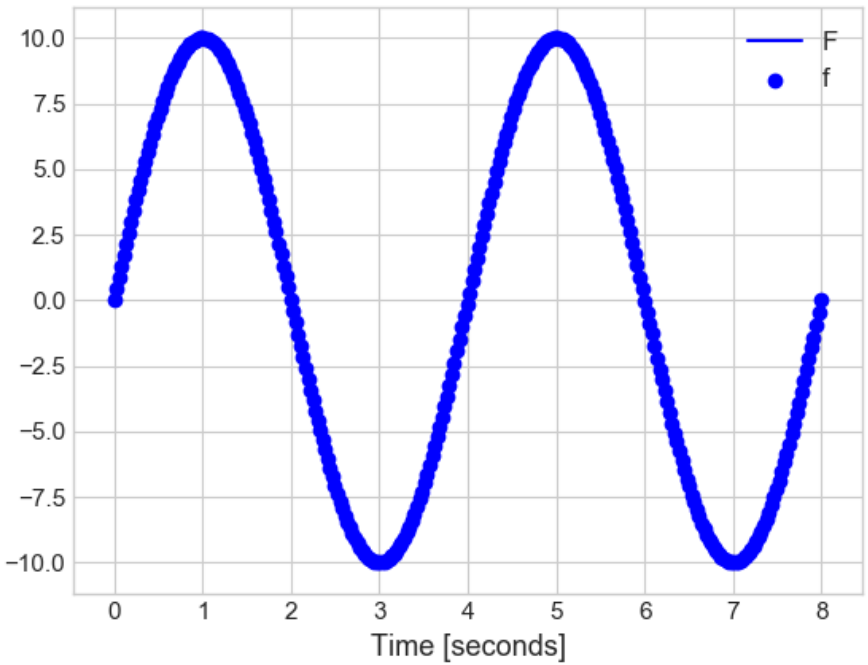


Figure: Sinus test at 40Hz, input signal

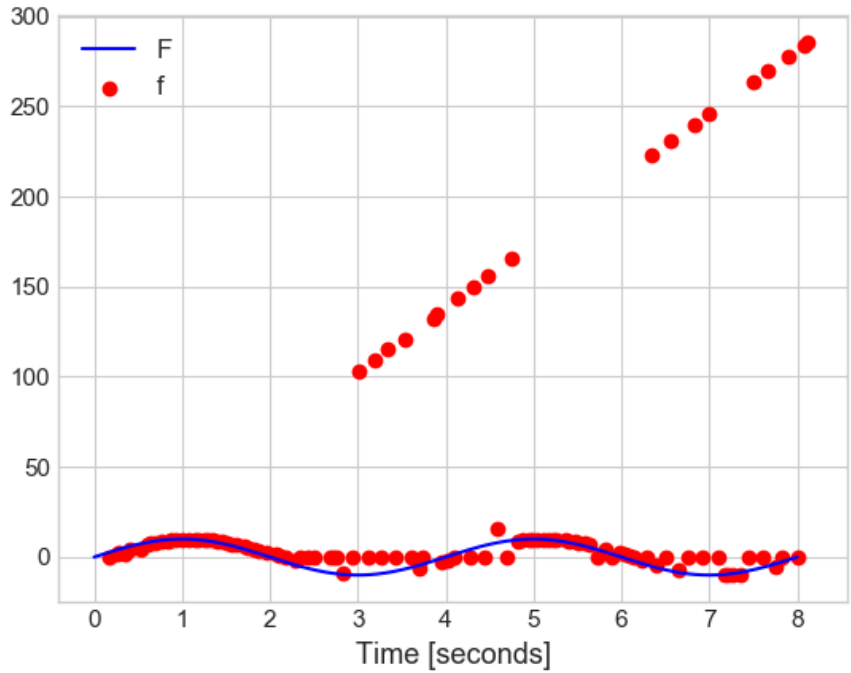


Figure: Sinus test at 40Hz, output signal

A voltage divider was added to the circuit because of the difference between the logic level voltage of the Arduino ($0 - 5$ Volt) and the bleutooth module ($0 - 3.3$ Volt).

$$V_{arduino} = \frac{R_2}{R_1 + R_2} V_{bleutooth}$$

$$= \frac{2.2}{3.2} V_{bleutooth}$$

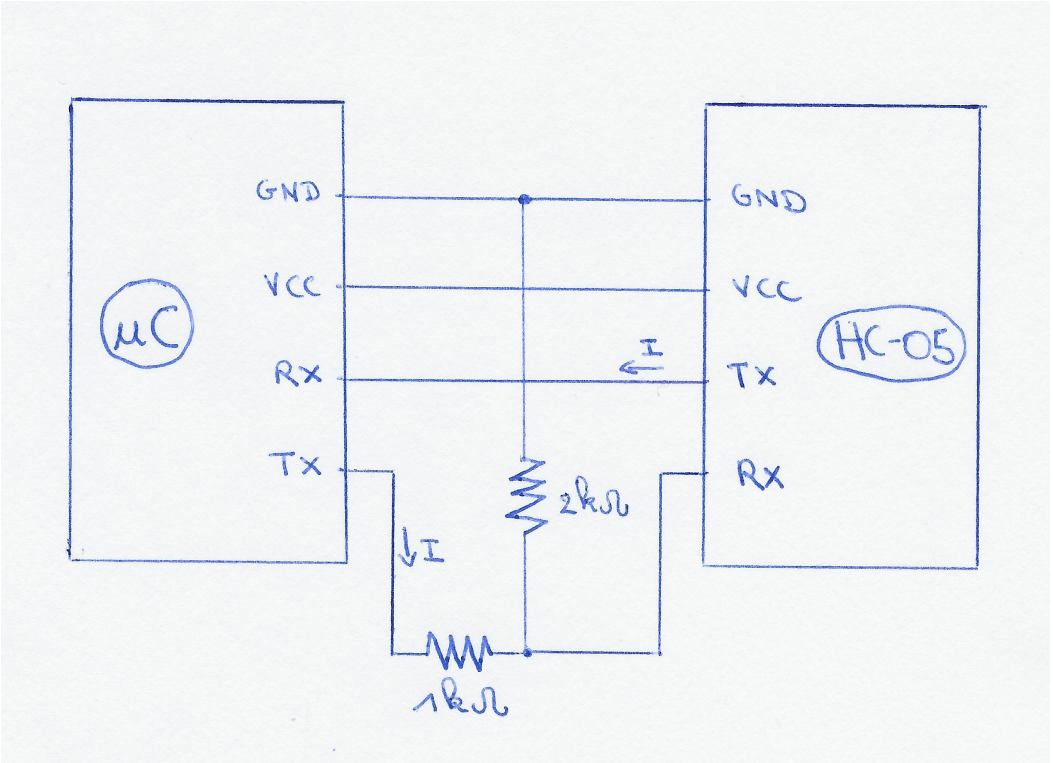


Figure: Voltage divider

Making use of the Hall effect, the encoders counts the number of revolutions of each motor shaft, noted n_L and n_R. The angular velocity of the wheel is computed as follows:

$$w_L = 2\pi f_L \qquad \text{with} \qquad f_L = \frac{n_L}{N\, G_b\, \Delta t}$$

with N the number of counts per revolution on one channel[1], and G_b the gearbox ratio. The robot can be modelled as a differential steering wheel such that,

$$v = \frac{v_L + v_R}{2} = \frac{R(w_L + w_R)}{2}$$

[1]Having only 2 interrupt pins available on the Arduino Nano, only one channel of each encoder was used.

The PWM is the signal send to the motors to control the speed. This relationship is visualized in the plot below. Small differences appear between the motors. We also note the relations varies with the charge of the battery. These imperfections force us to design a control-feedback system.

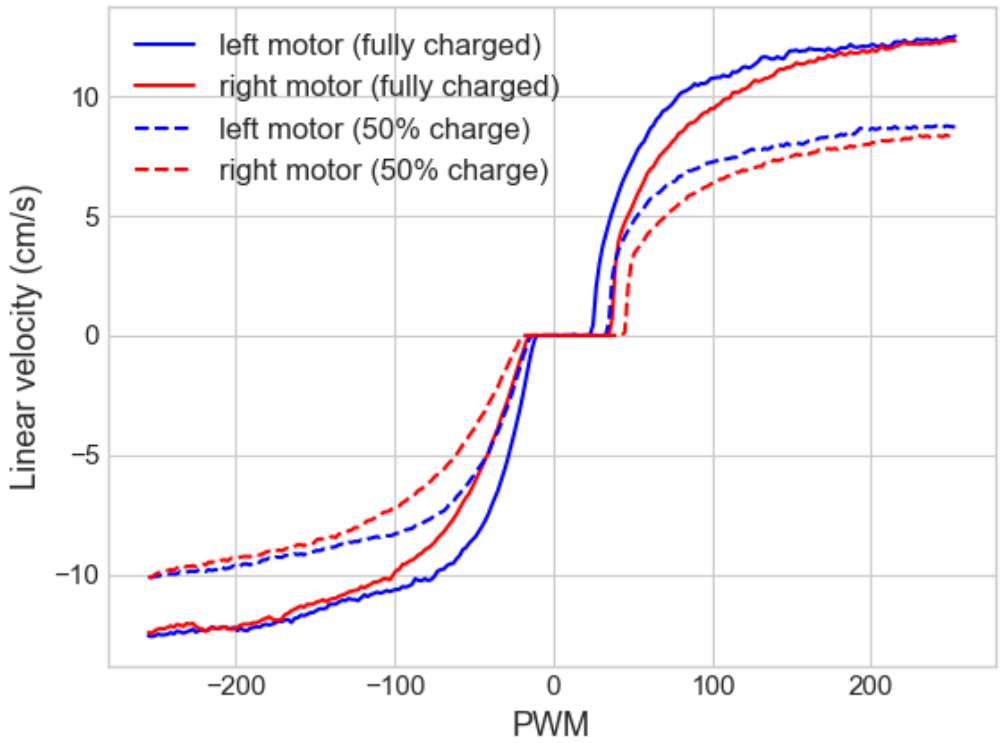


Figure: Relationship between PWM (input) and measured speed (output)

Overview

1 General

2 Components

3 Control

4 Exploration & Mapping

5 Code

A self-regulated system is acheived by using several PID controllers. The output of a PID controller can be expressed as

$$o_n(e_0 \ldots e_n) \leftarrow K_p e_n + K_d \frac{e_n - e_{n-1}}{\Delta t_{n-1:n}} + K_e \sum_{i=0}^{n} e_i \Delta t_{n-1:n}$$

with anti-windup,

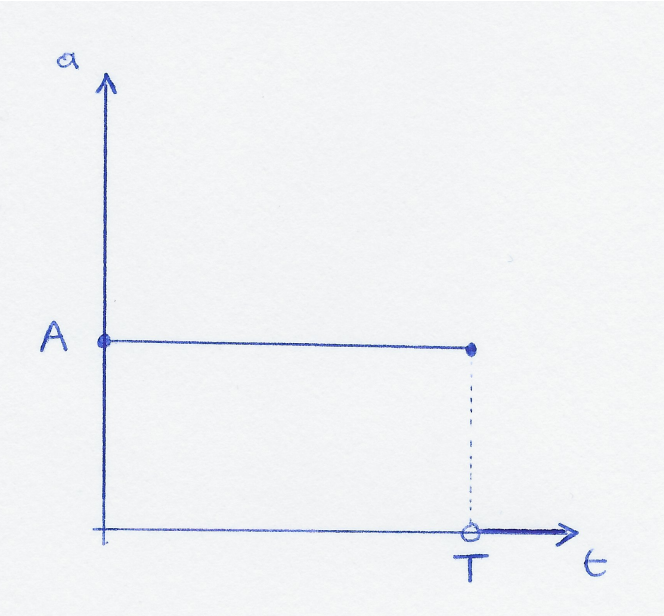
$$o_n = \begin{cases} max & \text{if } o_n > max \\ min & \text{if } o_n < min \\ o_n & \text{otherwise.} \end{cases}$$

The three main controllers are used to control the speed, direction and turning (discussed respectively on slides 24, 29 and **??**)

The pwm value is computed from the three controllers outputs as

$$\begin{cases} \alpha = o(e_{direction}) \\ \beta = o(e_{speed}) \\ \gamma = o(e_{align}) \end{cases} \Rightarrow \begin{cases} pwm_L = (\beta - \gamma) + \alpha \\ pwm_R = (\beta - \gamma) - \alpha \end{cases}$$

During speed control the systrem objective is to minimize e_{speed}, the difference between progress and measured speed. For a unifrom acceleration (shown in Fig.24) the update rule for the progress speed is Eq.1



$$v_{n+1} \leftarrow v_n + A\,\Delta t_{n:n+1} \quad \text{with} \quad A = \frac{v_{target}}{T} \tag{1}$$

Figure: Profile function of a uniform acceleration.

For a smoother transition[2], the robot accelerates following the profile as in Fig.30. However, we want the robot to achieve the target speed in the same amount of time T. This constraint is developed in Eq.2.

$$\begin{aligned} &\Leftrightarrow \int_0^T a(t) = \int_0^T a'(t) \\ &\Leftrightarrow A\,T = \underbrace{d\,B}_{\textit{triangles}} + \underbrace{(T-2d)B}_{\textit{rectangle}} \end{aligned} \tag{2}$$

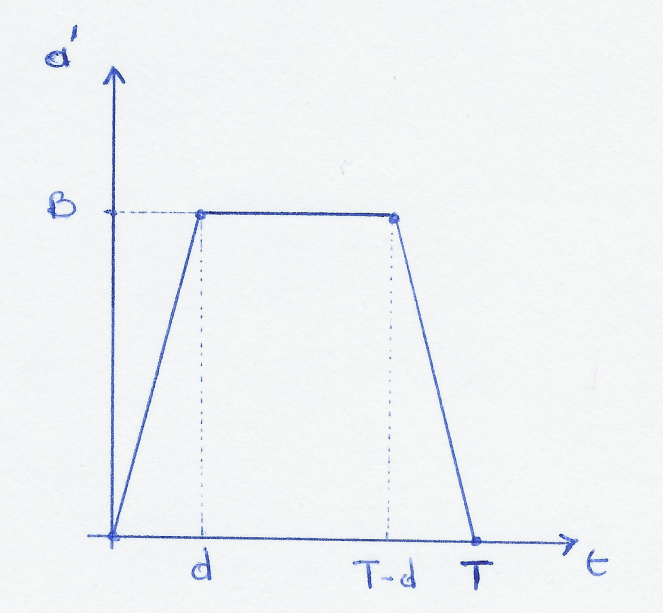


Figure: Profile function of a "smooth" acceleration

[2]Design purely based on instinct, effectiveness of method needs to be confirmed by experiments.

By substituting the parameter $\psi = \frac{d}{T}$ in equation 2, we obtain

$$\boxed{B = \frac{A}{1 - \psi}}$$

The update rule for the progress speed becomes

$$v_{n+1} \leftarrow v_n + \int_n^{n+1} a'(t) = v_n + \int_0^{n+1} a'(t) - \int_0^n a'(t)$$

Note that with the right hand side of the above expression, we can make use of precomputed values of the geometric surfaces.

The system achieves good results. Still, future tuning is necessary.

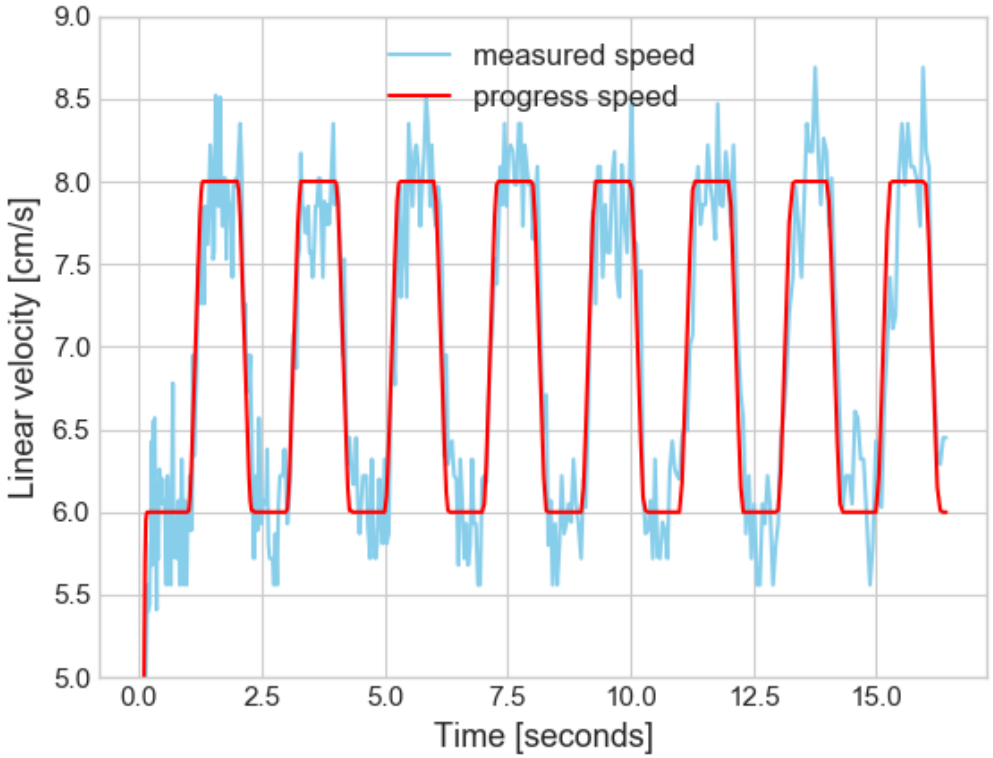


Figure: Evolution of measured with changing progress speed.

To capture the general tendency, an average over all the cycles of Fig.27 on the previous slide is done.

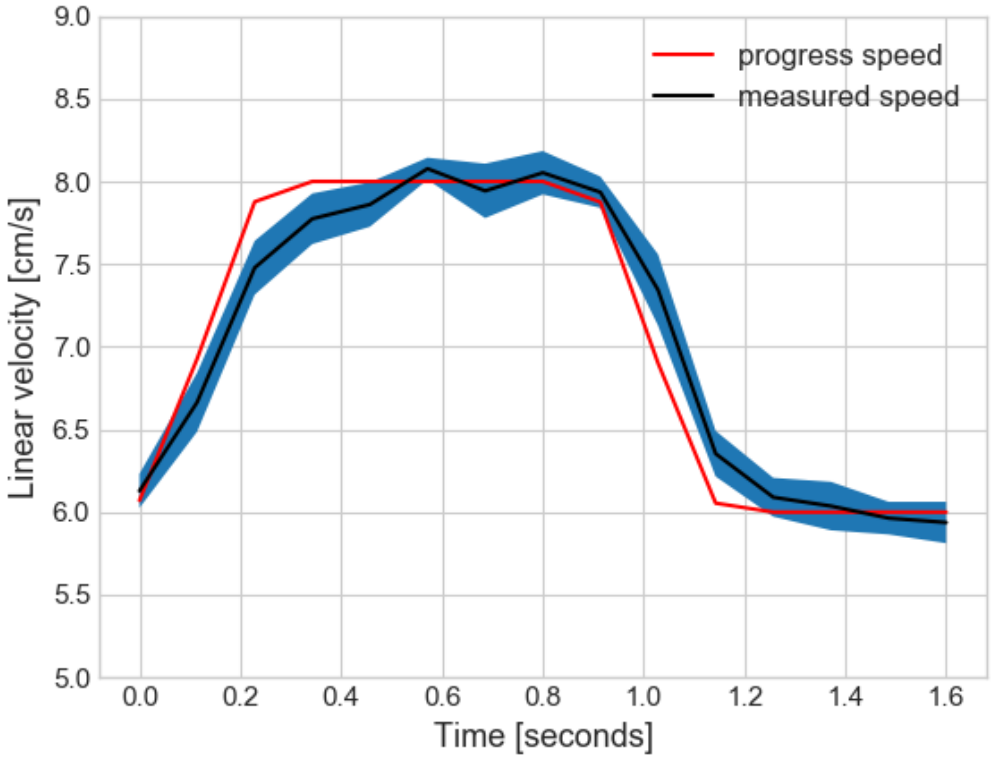


Figure: Mean measured speed with error band (± 2 standard deviation).

We define three types of direction controls,

- *forward control*: moving in a straight direction ($v_L = v_R \Rightarrow e_{direction} = v_R - v_L$).
- *line-following control*: adapts direction in order to keep the black line centered ($err_{line} = 0 \Rightarrow e_{direction} = err_{line}$).
- *turning control*: pure rotation can be approximated if we satisfy the constraint $v_L = -v_R$ ($e_{direction} = v_L + v_R$).

I inspired from setpoint weighting developped variable error weighting. It solves the problem: at initial when $v = 0$, if error line than takes too much importance compared to speed control. Z can be seen as a regularizer to control the relative imporatnce direction speed (β, α) correction term. Hyperbolic tangent.

$$Z(i) = \tanh(\zeta i)$$
$$e_i = Z(i)e_i'$$

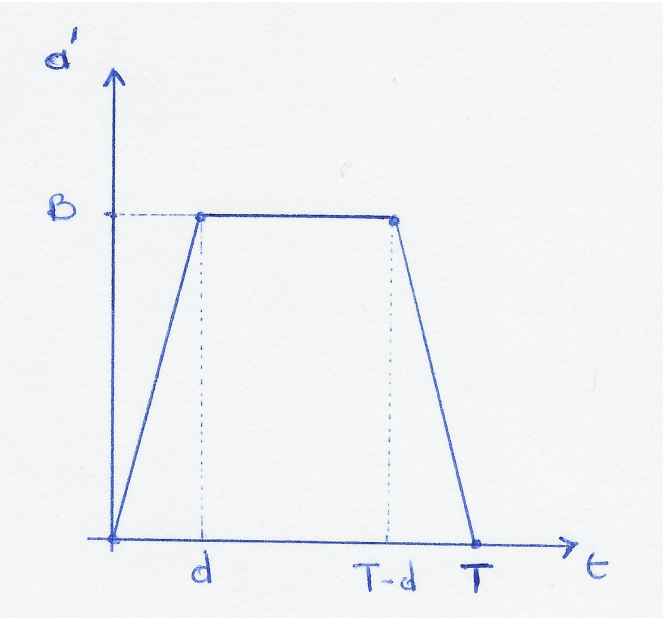


Figure: Tanh function with different zeta

On Fig.31, different trajectories of line-following on a straight line with initial disalignments are plotted.

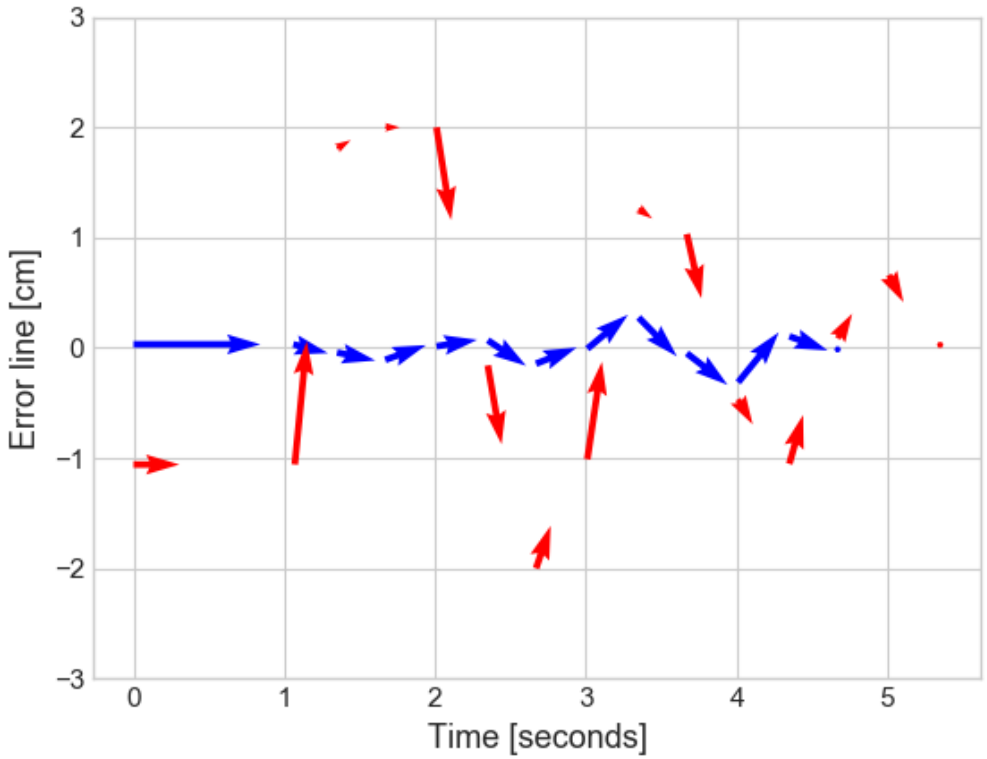


Figure: Profile of uniform acceleration

The COEX-1 is modelled as a two-wheel robot (see figure below). By integrating the angular velocity under the assumption that $v_R = -v_L$, rotational displacement can be deduced

$$\theta(t) = \frac{2vt}{b} + \theta_0 \tag{3}$$

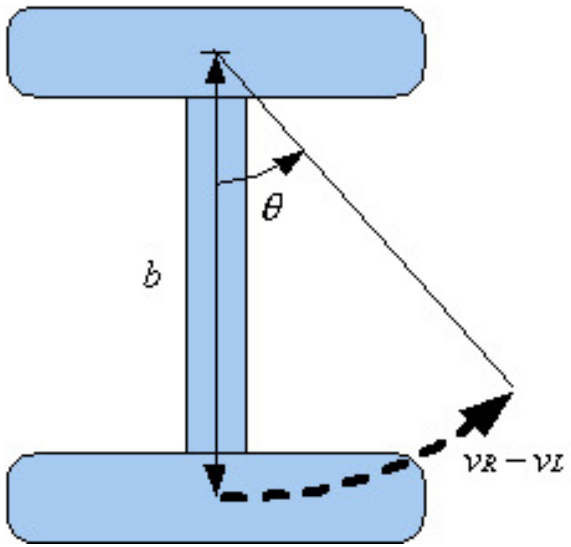


Figure: Rotation of COEX-1

Using Eq.3 an algorithm is designed to make the robot turn Θ radians.

1: **procedure** TURN(Θ, v_{target})
2: $\theta \leftarrow 0$
3: TURN(v_{target})
4: **while** $\theta < \Theta$ **do**
5: $\theta_{n+1} \leftarrow \frac{2\, v_n\, \Delta_{n:n+1}}{b}$
6: **end while**
7: STOPMOTORS
8: **end procedure**

The margin error observed is roughly between -10 and 10 radians, which is surprisingly small. However not good enough for a robust turn when exploring environments. Another method is developed (see slide 43).

Overview

1. General
2. Components
3. Control
4. Exploration & Mapping
5. Code

To explore an unknown environment structured as a maze it needs to be able to:

- Compute the distance travelled since the last intersection (slide ref)
- Detect an intersection (slide ref)
- Turn to the desired intersection (slide ref)

Here under is a plot of the computed distance ($v\Delta t$) versus ground truth. Mean on training set.

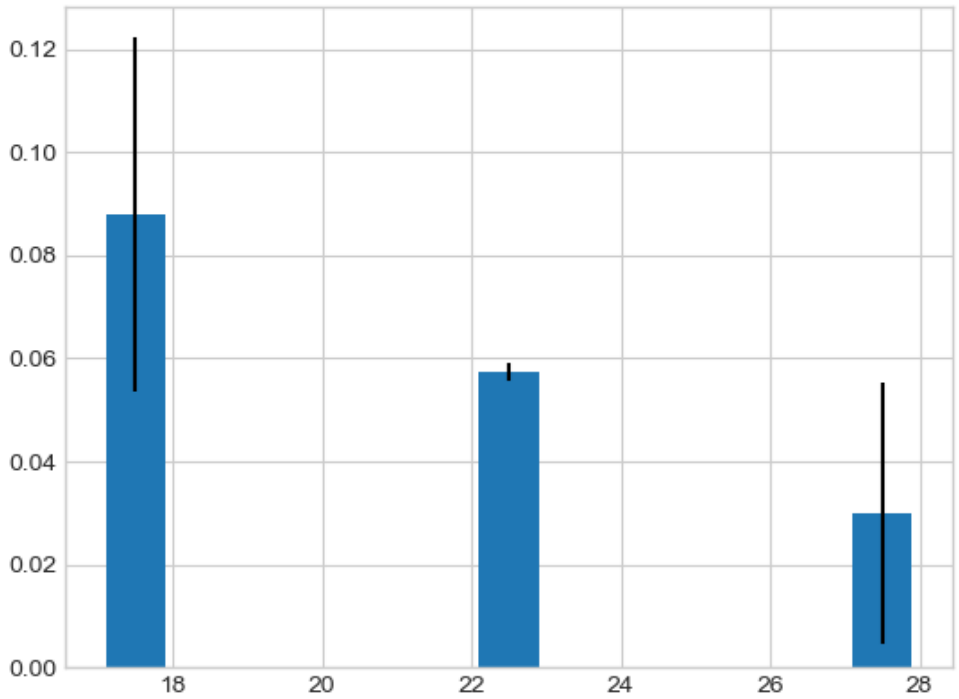


Figure: Architecture of the code

A second method deduced from the Fig above is to remove a term $bb\dot{\theta}$. These

Plot comparison with error bars of two methods on test set.

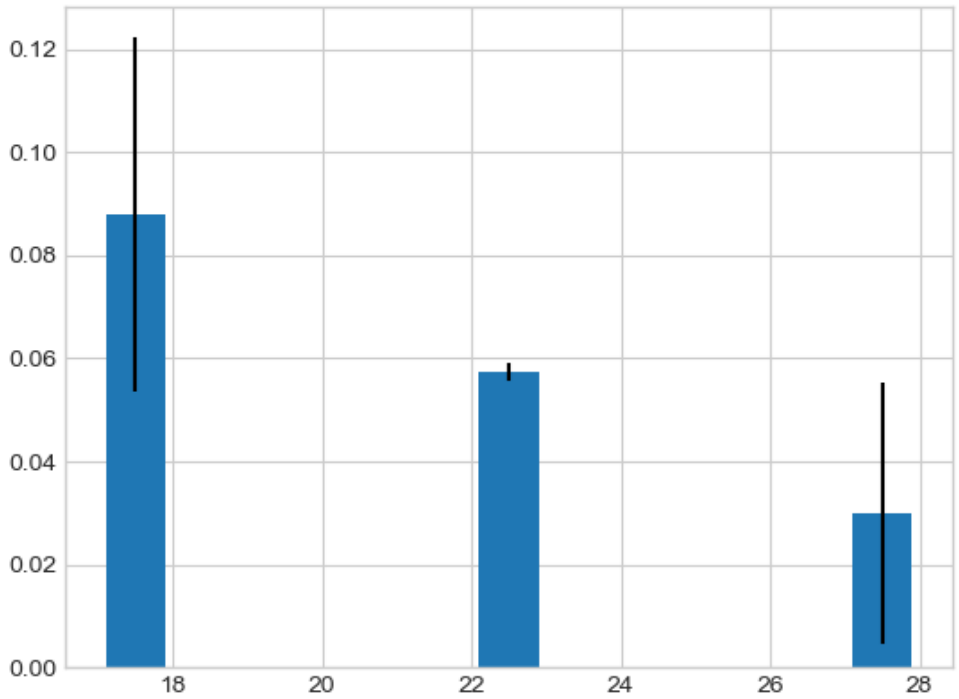
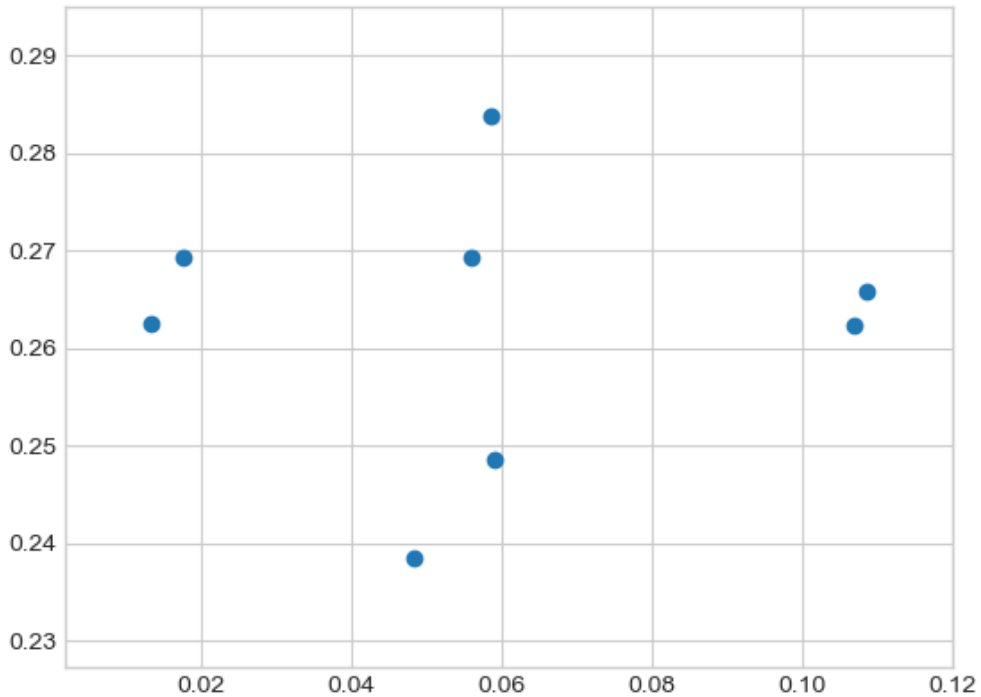


Figure: Architecture of the code

We could try to go further by quantifying uncertainty. For example if we analyze MSE see eq.

$$MSE = \sum_{i=0}^{N} err_{line}^{2}$$

obtained iteratively by rolling mean method. We can see a clear trend.



However a simpler choice is to discretize the result of the second. Moreover having separate distributions with no overlapping, the method seems to be sufficient. If we fix the length 7.5 12.5 etc. Plot discretization on test set.

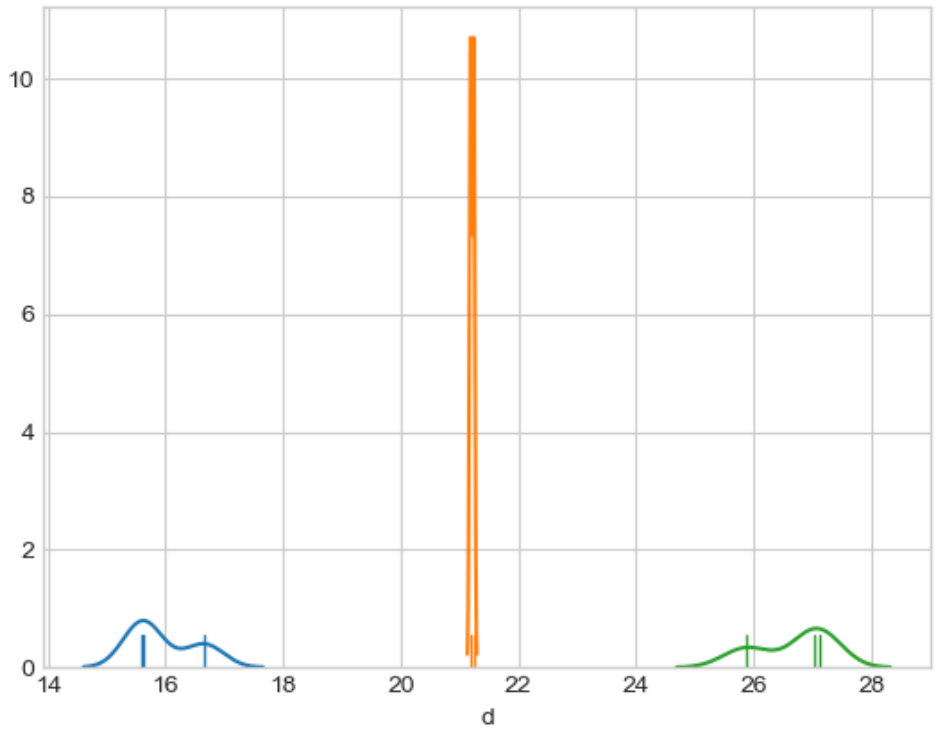


Figure: Architecture of the code

Map + caption: describing second method is applied + discretization.

Approach developped to intersection detection is based on majority vote idea. x_{loc} takes value 1 if a black line is detected at location loc. From moment anomalie detected we start following procedure.

$$y_{loc} = \underset{x \in \{0,1\}}{\text{argmax}}\, \text{mode}(x_{loc})$$

Boolean value for intersection based on decision rule as follows

$$(y_{center} = 0) \vee (y_{left} = 1) \vee (y_{right} = 1)$$

We use the values computed during intersection detection, and we make the robot go a bit further forward. The type of intersection is determined and takes values in following set:

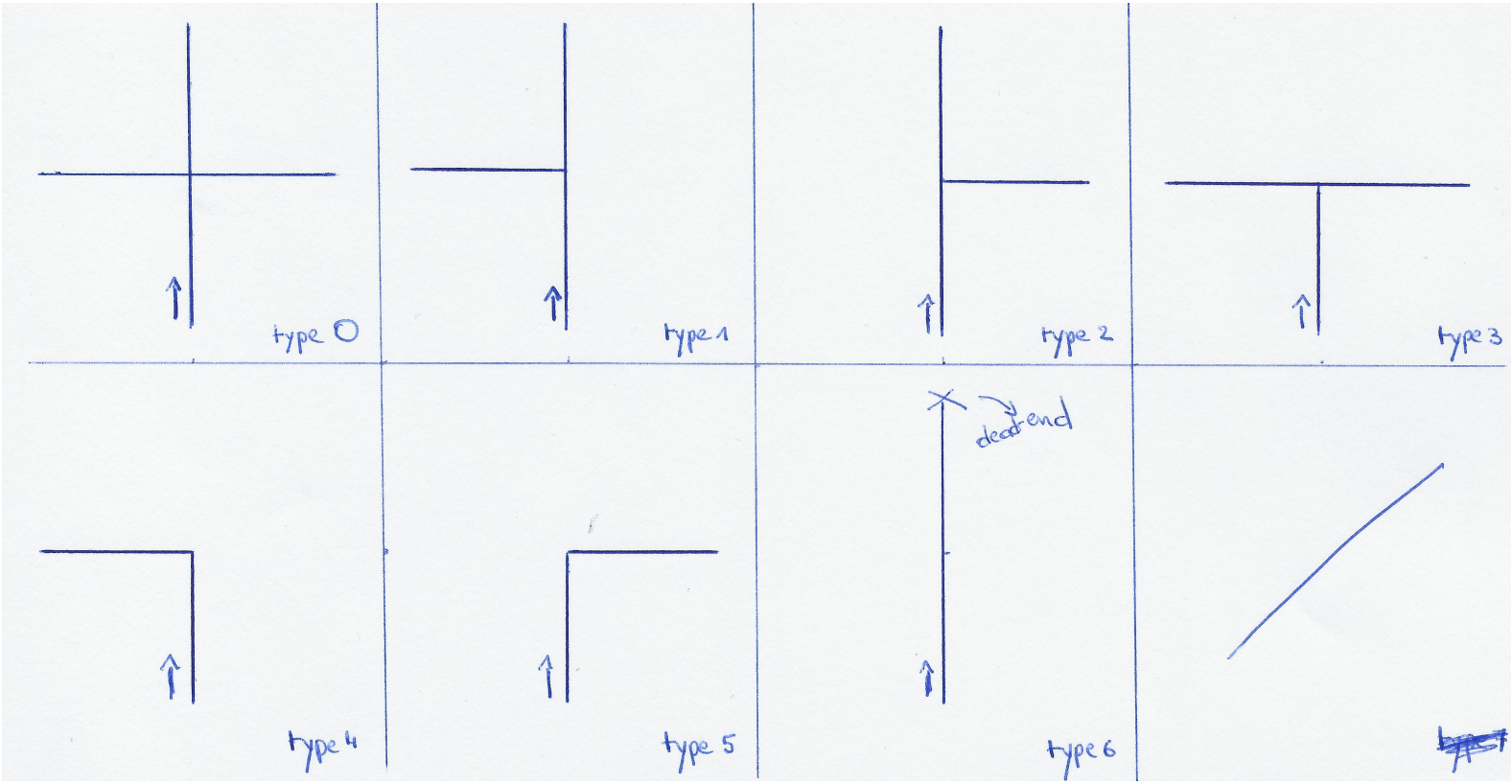


Figure: The 7 types of intersections

Explain simulated for less tedious tuning. Compute quadratic interpolation to fi points $(\pm 2, v_{init})$ and $(\pm\epsilon, v_{min})$.

$$v(x; a, b) = a(x - \epsilon)^2 + b$$

with initial conditions $v(x_{init}) = v_{init}$ and $v(\epsilon) = v_{min}$. Thus

$$a = \frac{v_{init} - v_{min}}{(2 - \epsilon)^2} \qquad \text{with} \qquad b = v_{init}$$

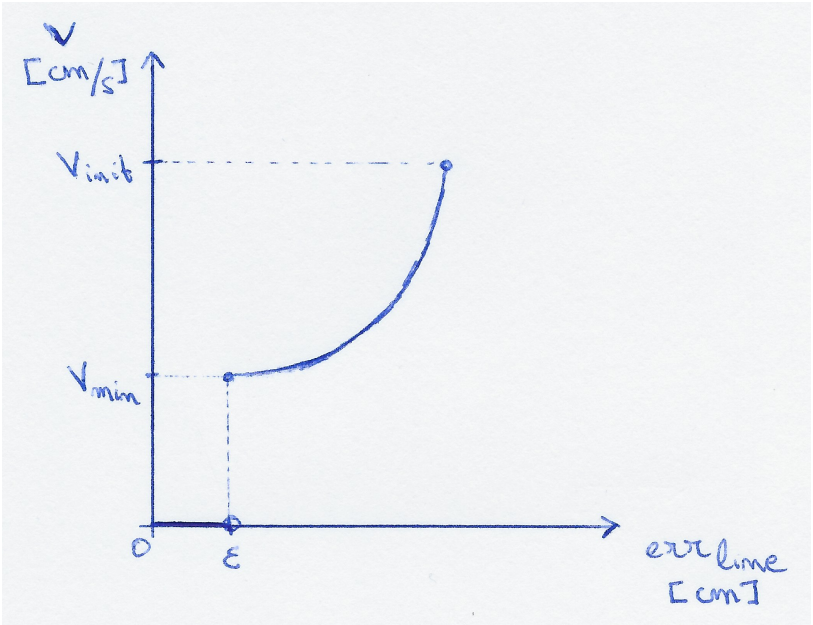


Figure: The 7 types of intersections

Quadratic is too short time to be applied. Better setpoint would be x instead of v.

Overview

1. General
2. Components
3. Control
4. Exploration & Mapping
5. Code

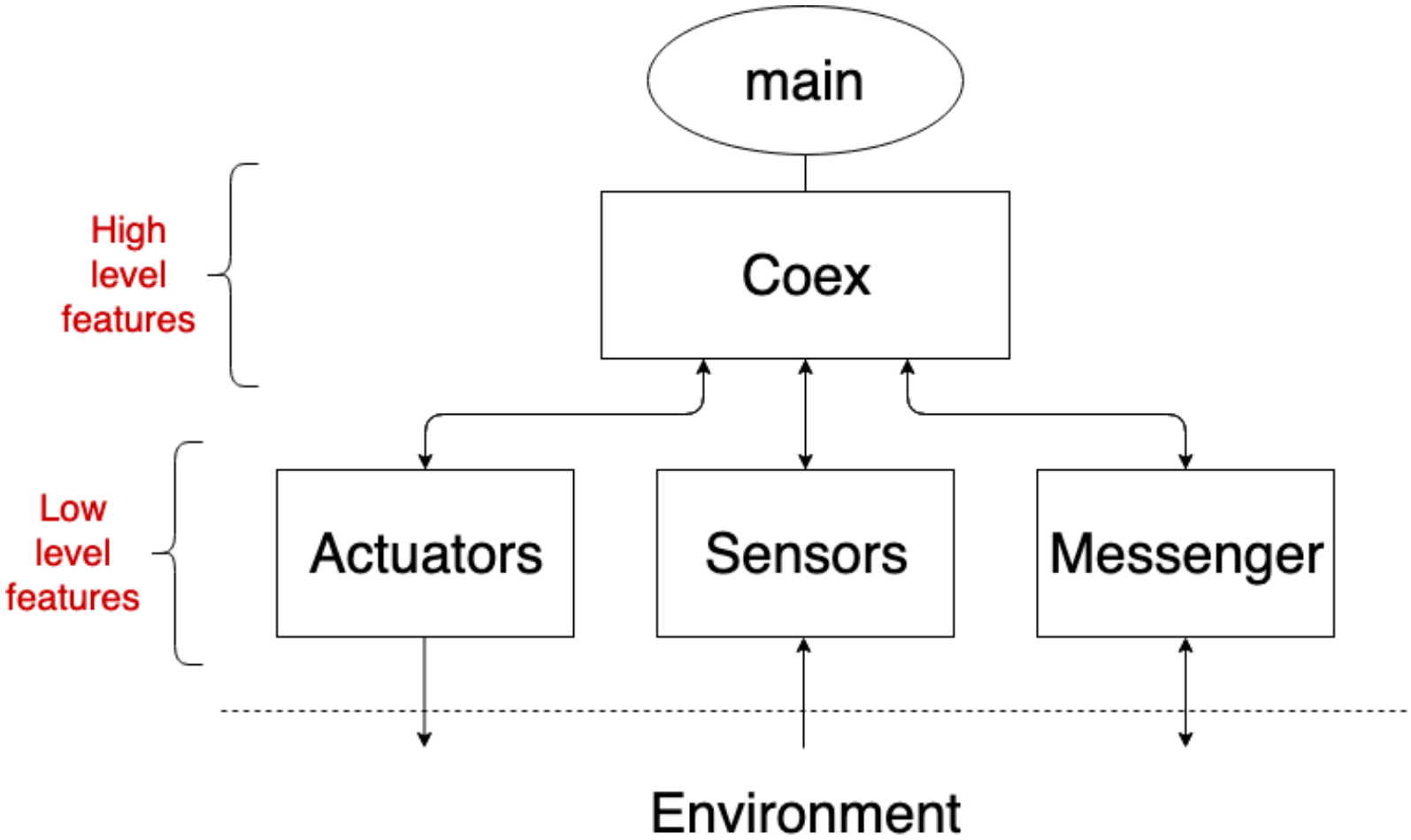


Figure: Architecture of the code