COEX-1

Cooperative Explorer

Aurélien Werenne
Advisor: Prof. B. Boigelot

University of Liège

Link to code: Github

March 17, 2019

Overview

Overview

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)

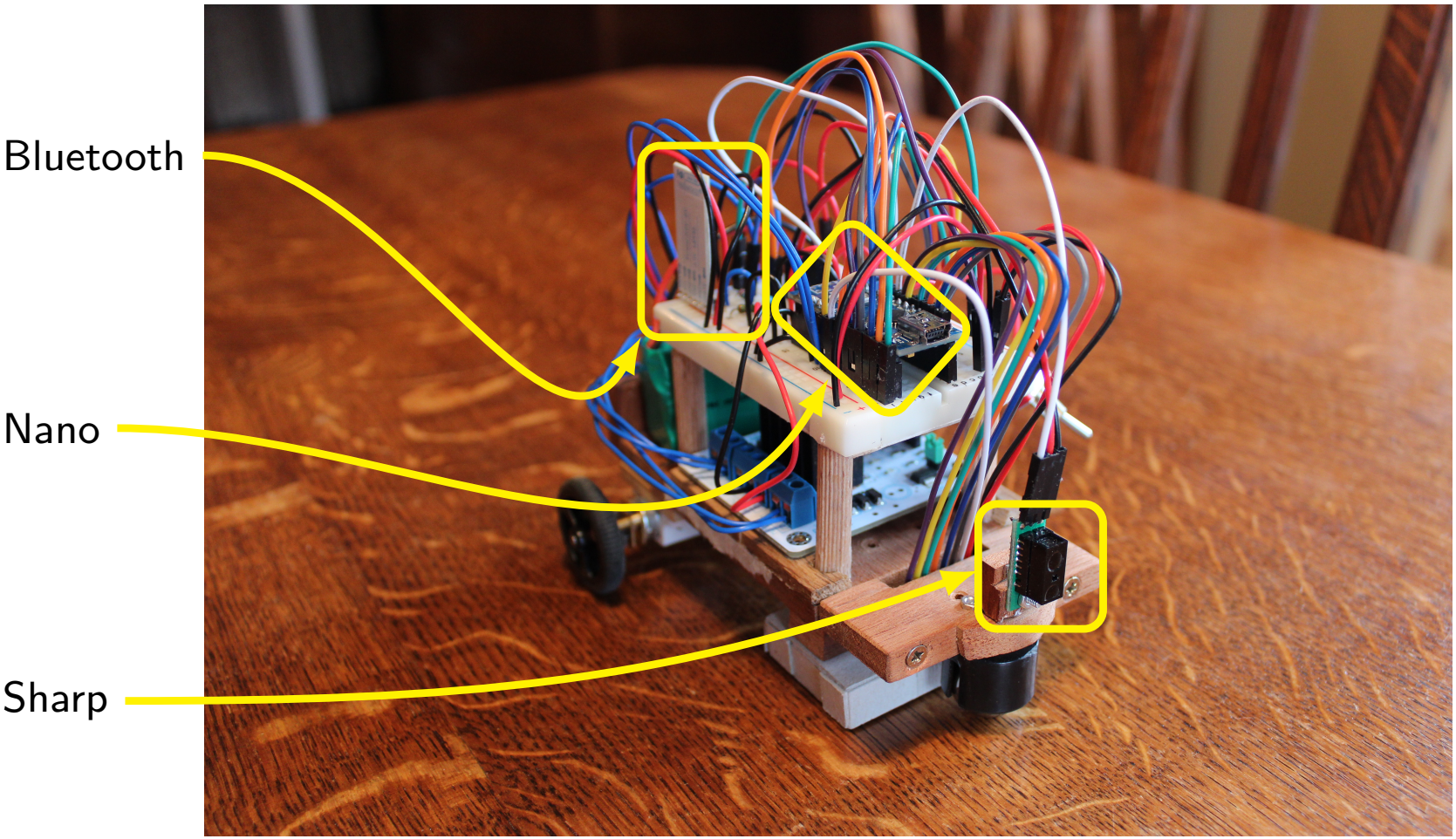

Figure 1: Front view of COEX-1

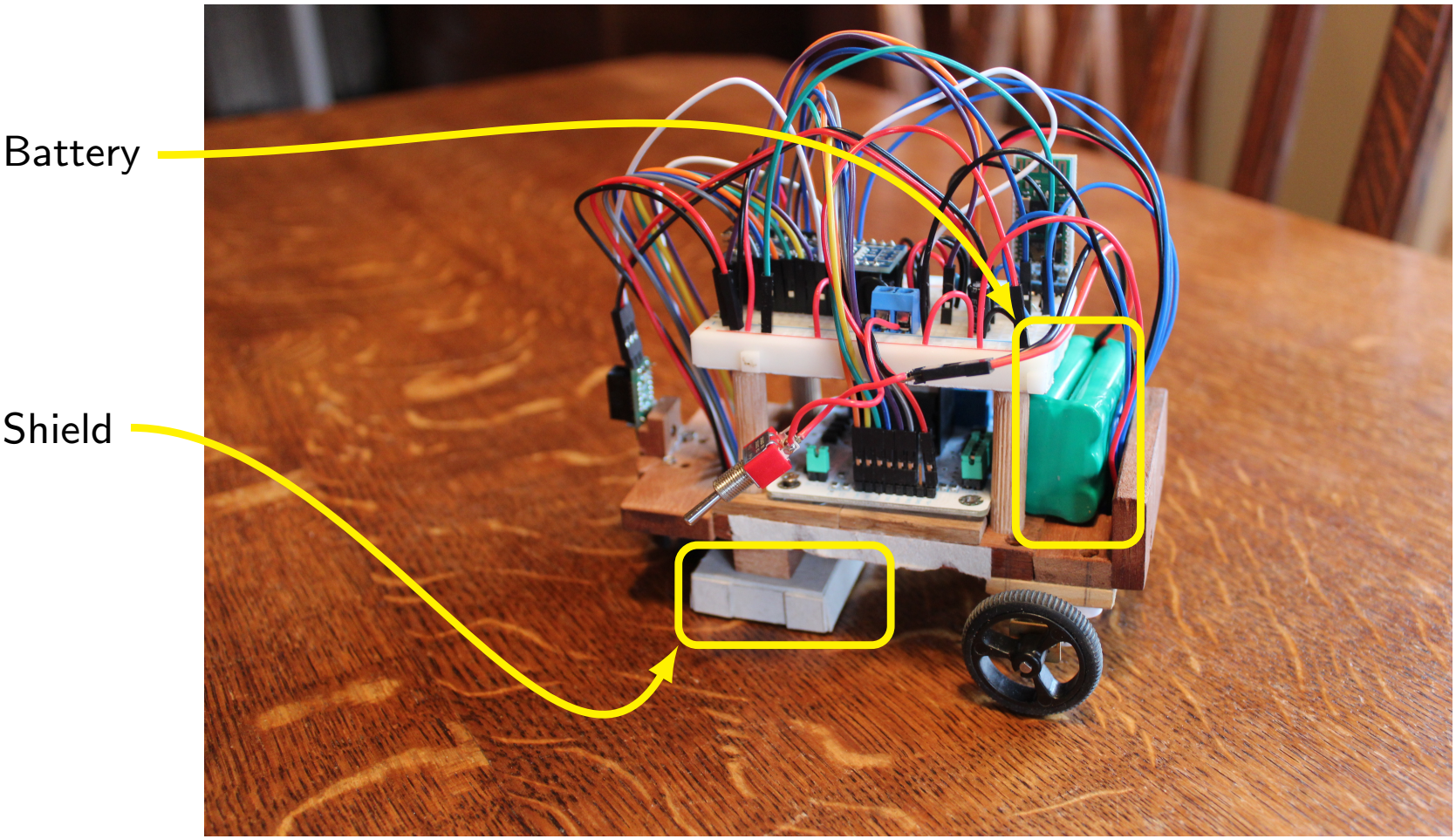

Figure 2: Side view of COEX-1

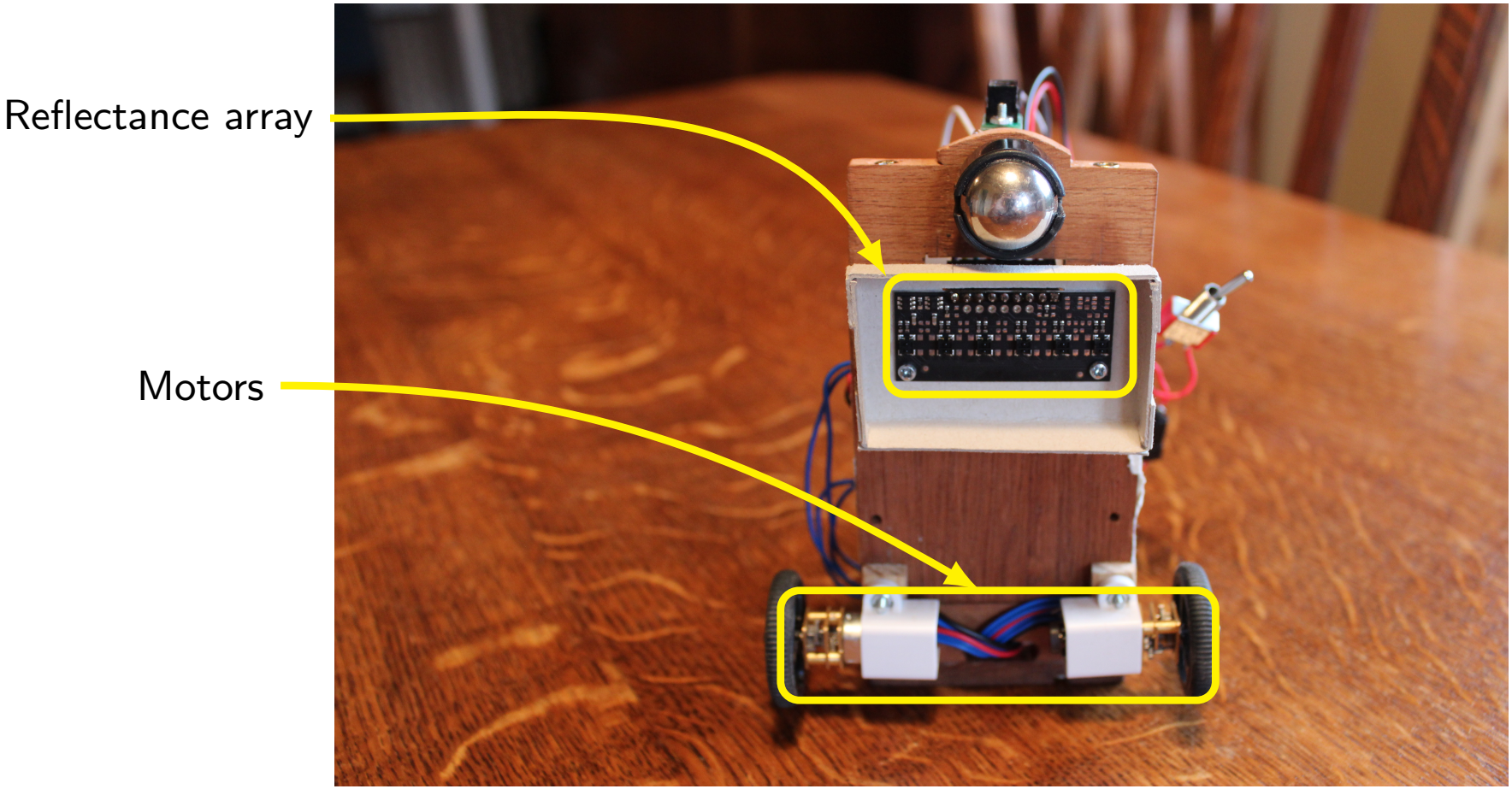

Figure 3: Bottom view of COEX-1

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

Adding a bypass capacitor was necessary to make the Sharp sensor work. On Fig.4 the detection of obstacles is read perfectly. The threshold is set at 2 Volt instead of the mean (3 Volt) in order to avoid False Negatives.

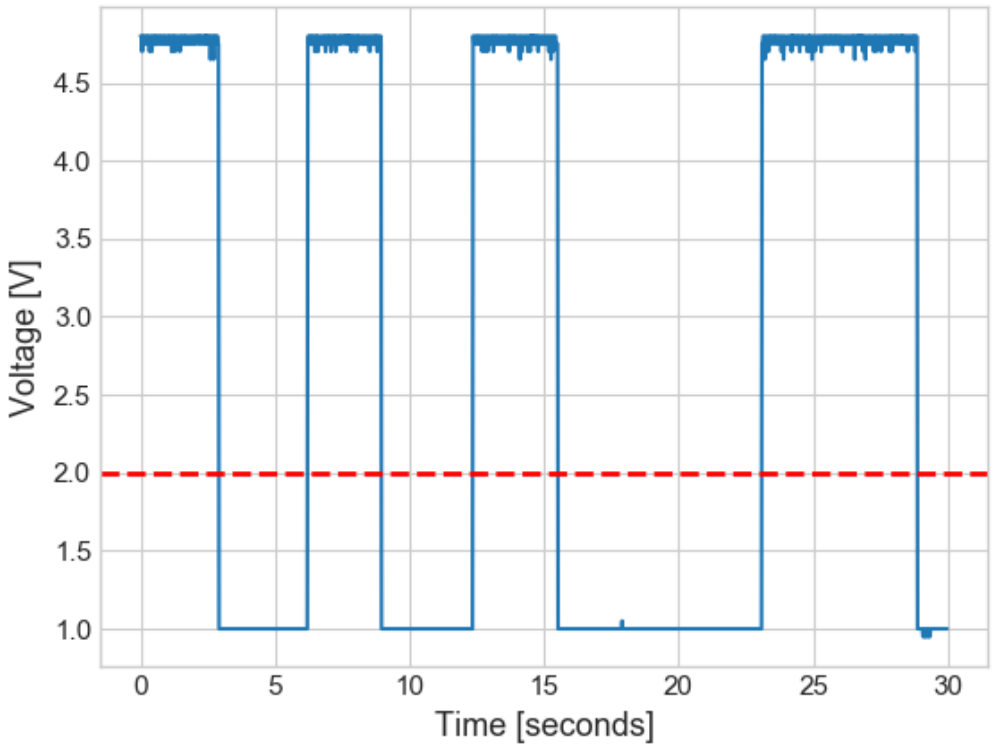

Figure 4: Sharp sensor output, $f = 20Hz$. Test performed with back and forth movement of obstacle, going in and out of reach of the sensor.

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. The sensor is particularly sensitive to ambient light interferences. As a solution a shield was constructed around the sensor (Fig.3).

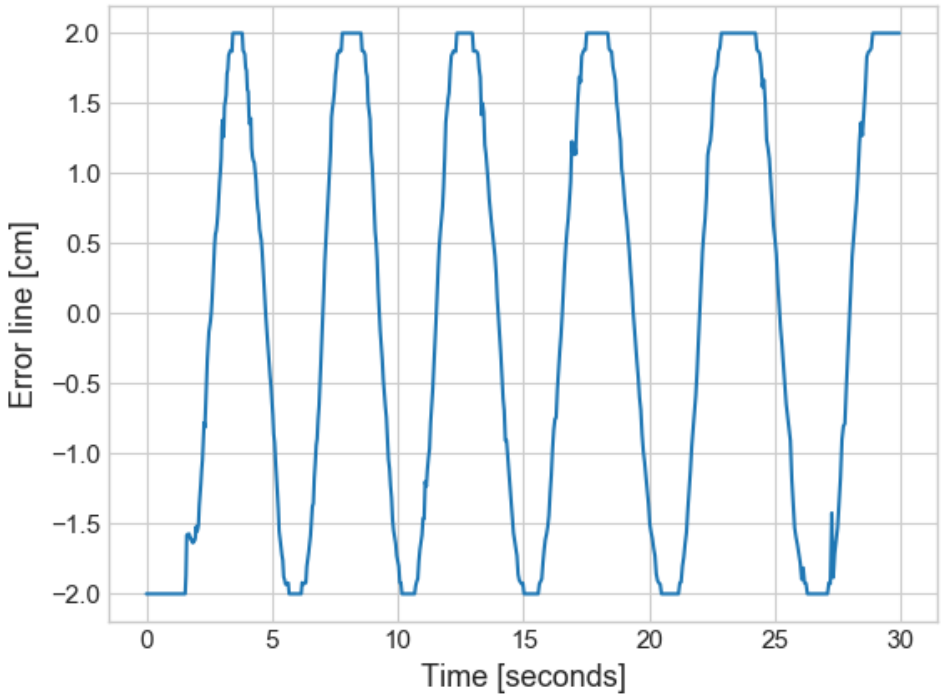

Figure 5: QTR sensor output, $f = 100Hz$. Test performed by moving the black line back and forth between left and right side of the sensor.

To evaluate the robustness of the communication protocol, a simple test is performed which I call the *sinus test*. The principle is to send a message at time t with content $f(t)$ to the robot, and receive its response at time t' with content $f(t)$ unchanged. The function $f(t) = A\sin(2\frac{\pi}{T}t) + b$, enables us to easily deduce delay, distortion, ...

On Figs14, 15, 16 we plot $(t, f(t))$ and on Figs14, 15, 16 we plot $(t', f(t))$. We conclude that 5Hz works well and is enough for our use case.

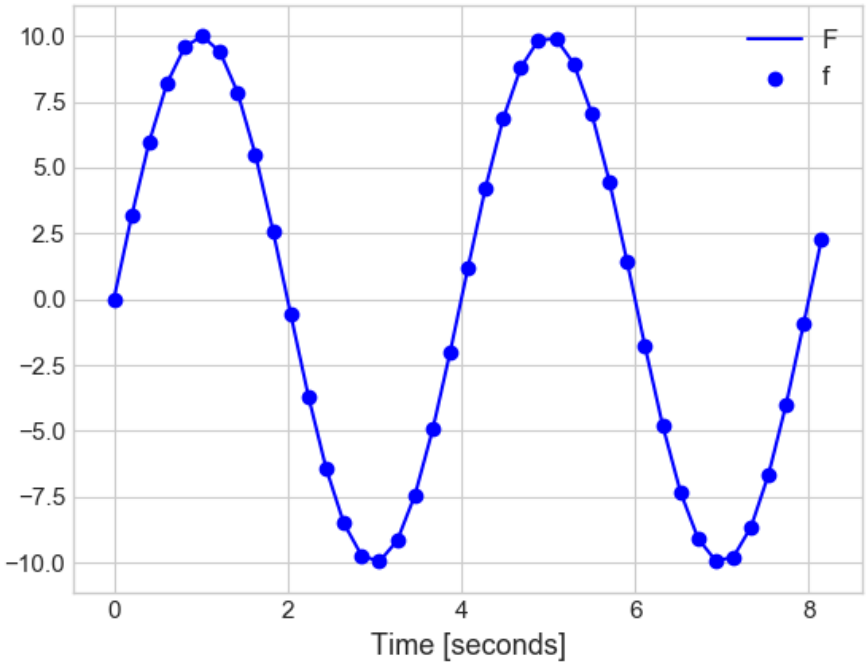

Figure 6: Sinus test at 5Hz, input signal

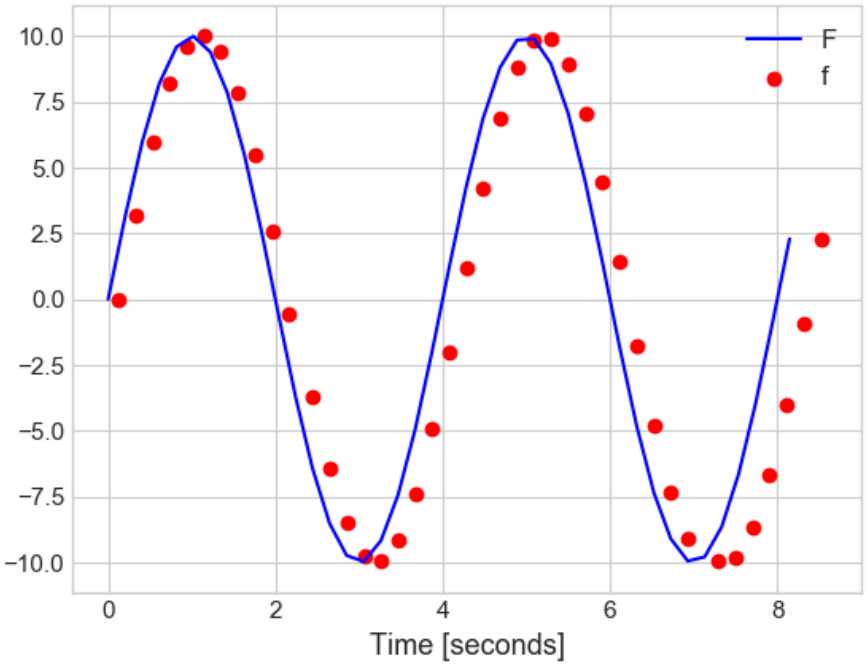

Figure 7: Sinus test at 5Hz, output signal

Some loss is observed at 25Hz.

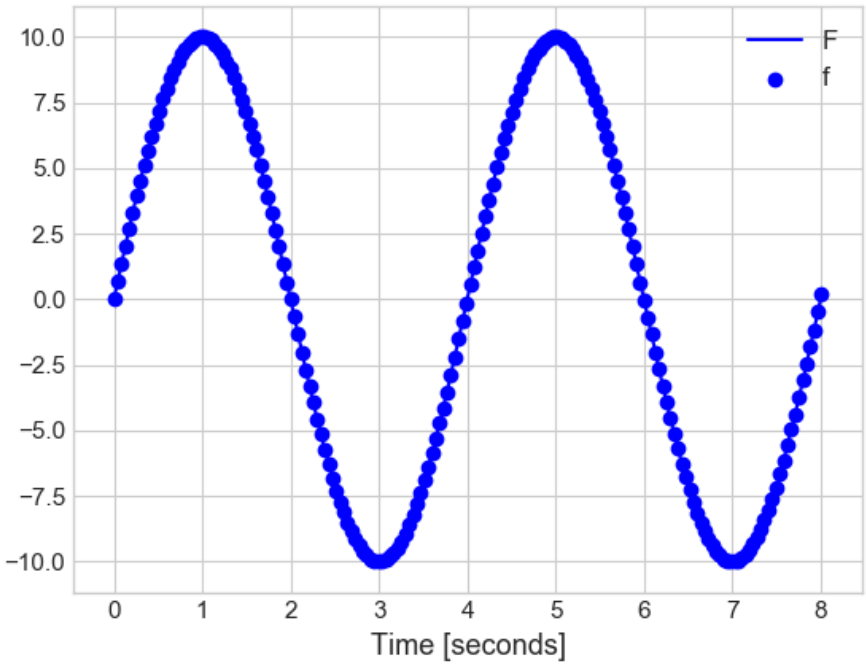

Figure 8: Sinus test at 25Hz, input signal

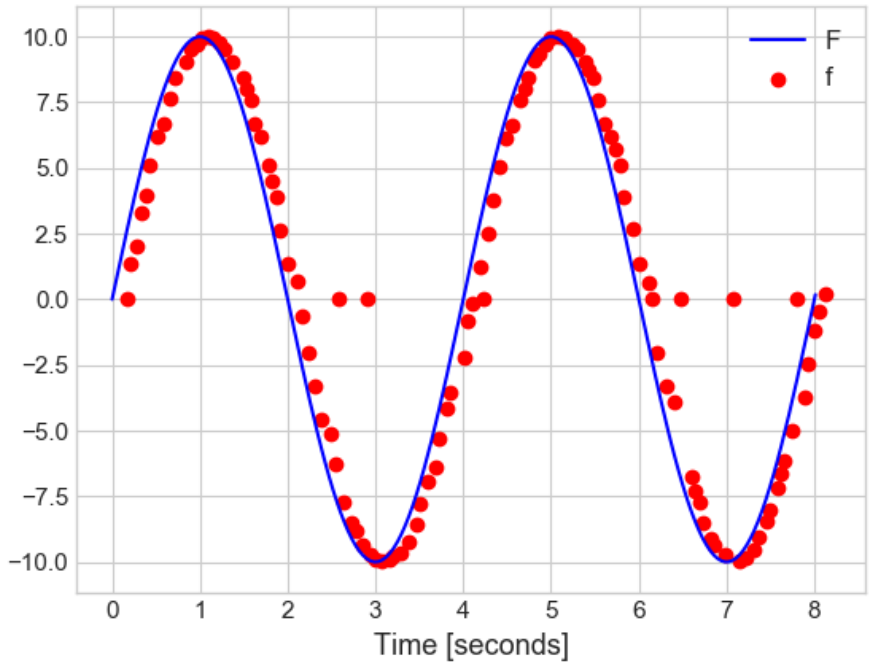

Figure 9: Sinus test at 25Hz, output signal

At 40Hz, the protocol completely breaks down.

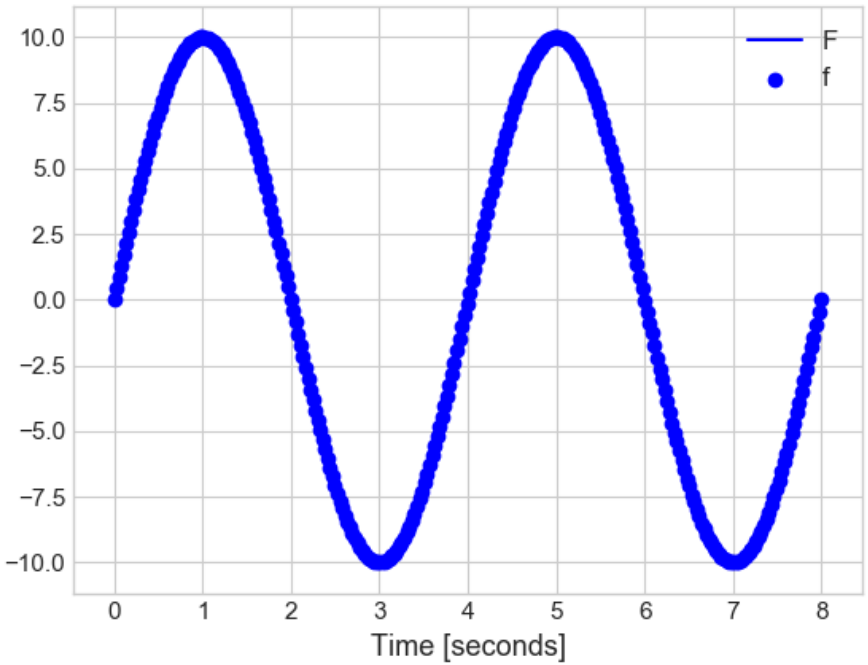

Figure 10: Sinus test at 40Hz, input signal

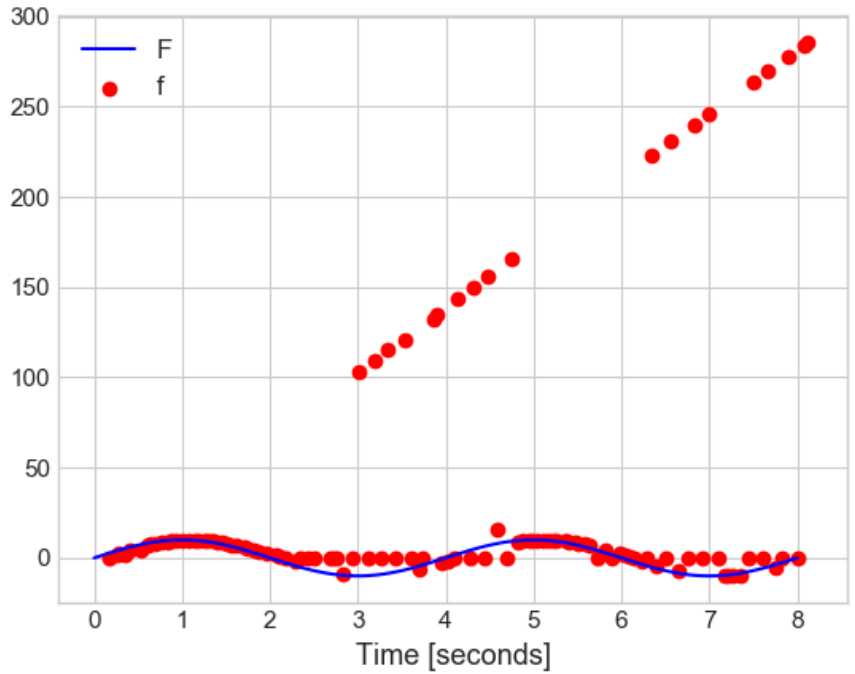

Figure 11: Sinus test at 40Hz, output signal

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).

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

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

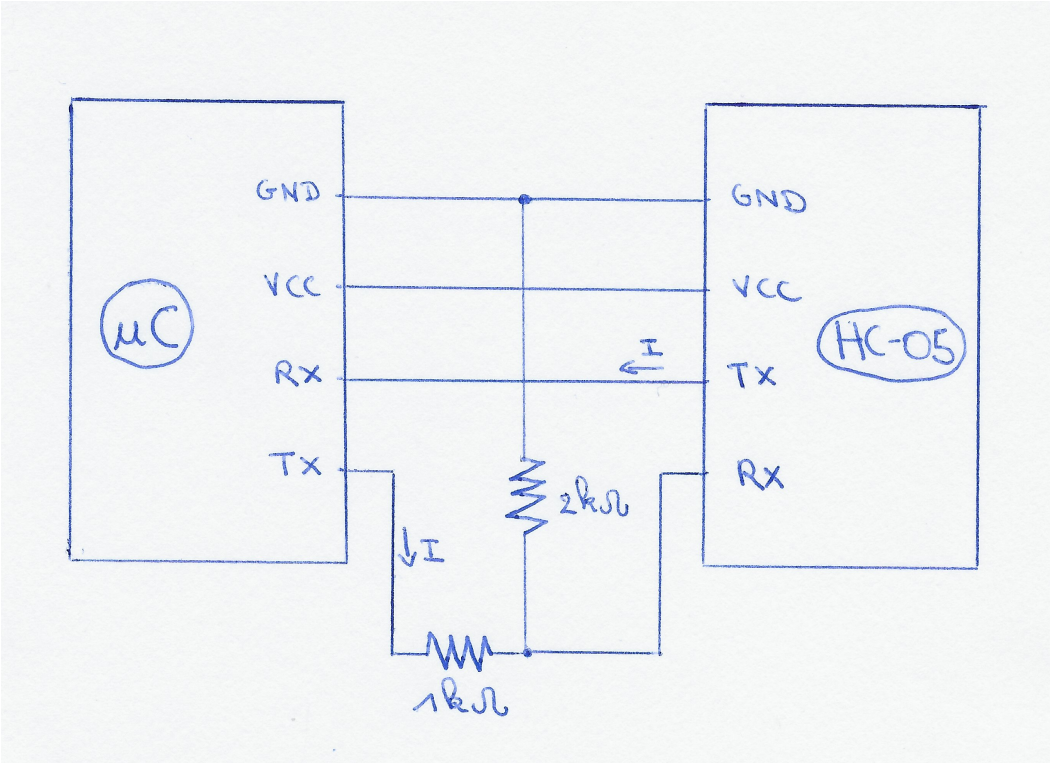

Figure 12: Voltage divider

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 variation with the charge of the battery. These imperfections force us to design a control-feedback system.

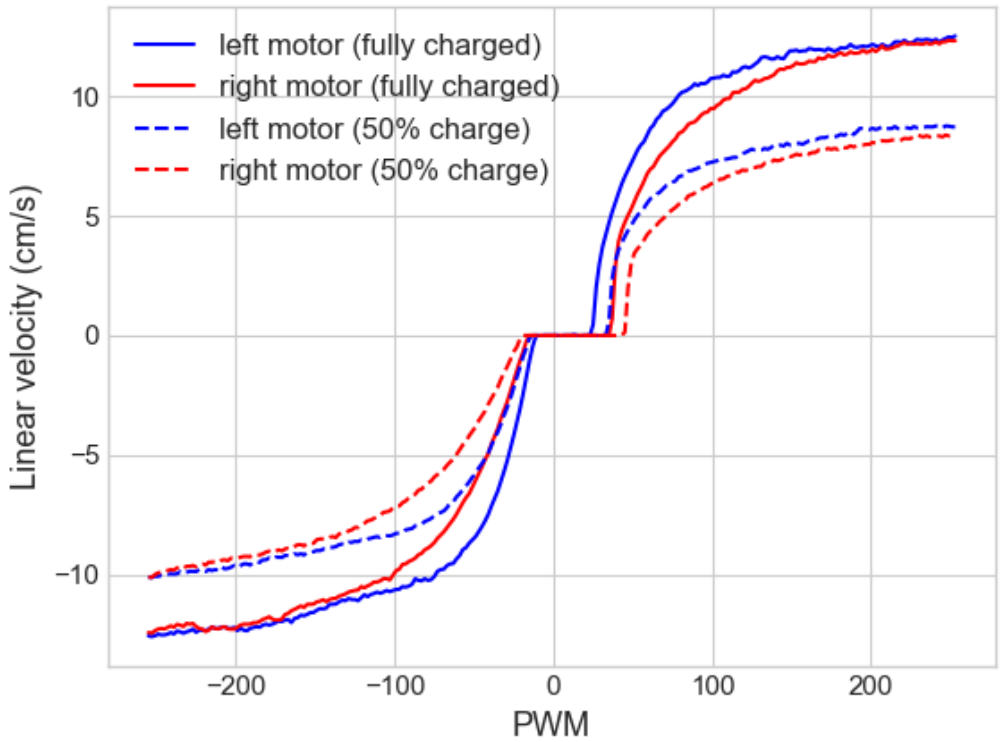

Figure 13: Relationship between PWM (input) and measured speed (output)

Overview

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

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

with anti-windup,

$$I_{out}^{(n)} = \begin{cases} max & \text{if } I_{out}^{(n)} > max \\ min & \text{if } I_{out}^{(n)} < min \\ I_{out}^{(n)} & \text{otherwise.} \end{cases}$$

The three main controllers are used to control speed, direction and turning (discussed respectively on slides 23, 28 and 32)

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.23) the update rule for the progress speed is Eq.1

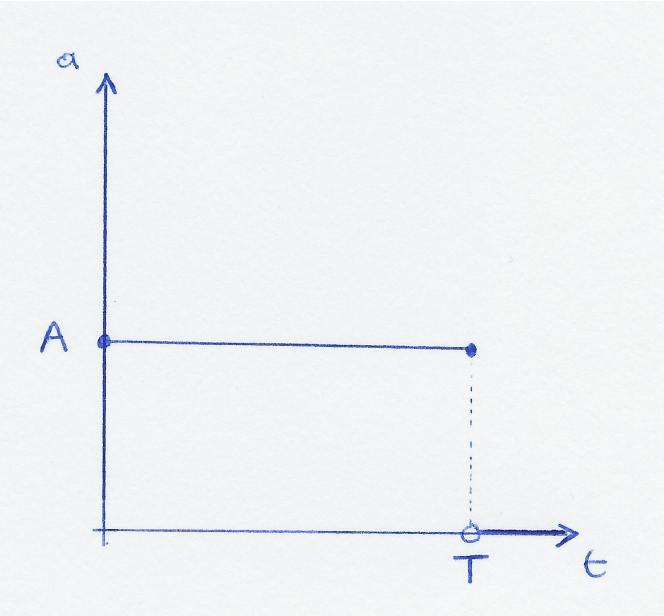

Figure 14: Profile function of a uniform acceleration.

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

For a smoother transition[2], the robot accelerates following the profile as in Fig.24. However, we want the robot to achieve the target speed in the same amount of time T. This constraint is developed in Eq.2.

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

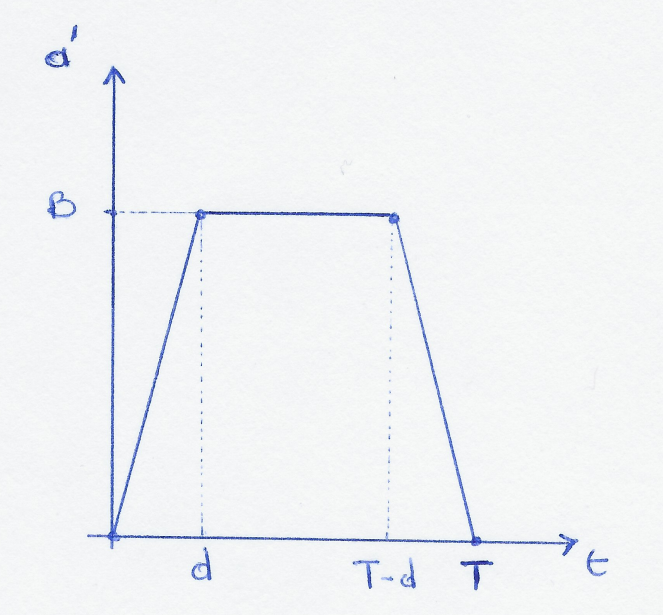

Figure 15: Profile function of a "smooth" acceleration

[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.

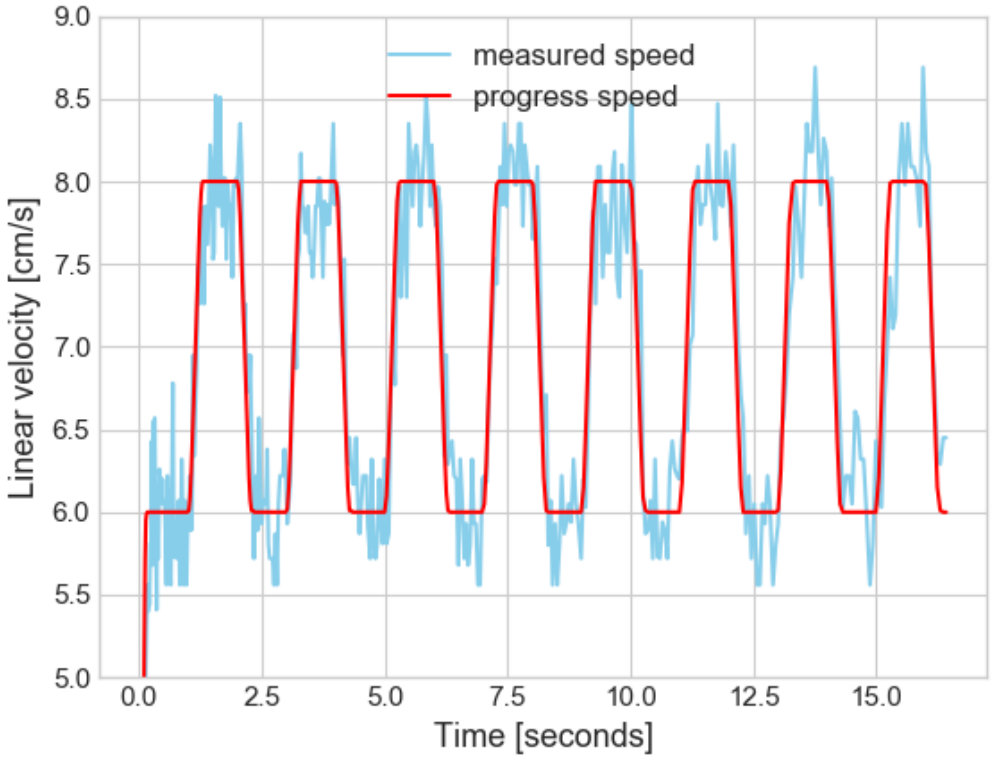

Figure 16: Evolution of measured with changing progress speed.

To capture the general tendency, an average over all the cycles of Fig.26 on the previous slide is done.

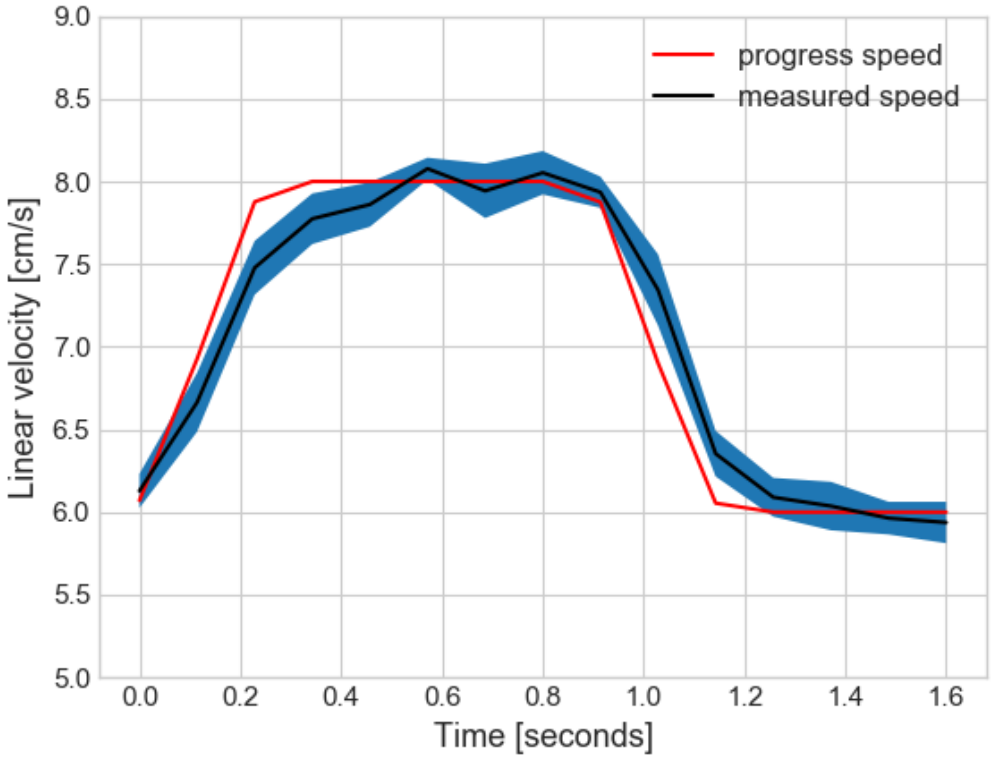

Figure 17: Mean measured speed with error band (± 2 standard deviation).

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 $v_L = -v_R$ ($e_{direction} = v_L + v_R$).

When the motionless robot starts following a line, the correction term α takes too much importance relative to β (speed), causing oscillations. I introduce *variable error weighting* (see Eq. 3) to solve this problem.

The error $e_{direction}$ is weighted:

$$e^{(t)}_{direction} = Z(t) err^{(t)}_{line} \quad (3)$$

with $Z(t) = \tanh(\zeta t)$ and ζ a tuned parameter.

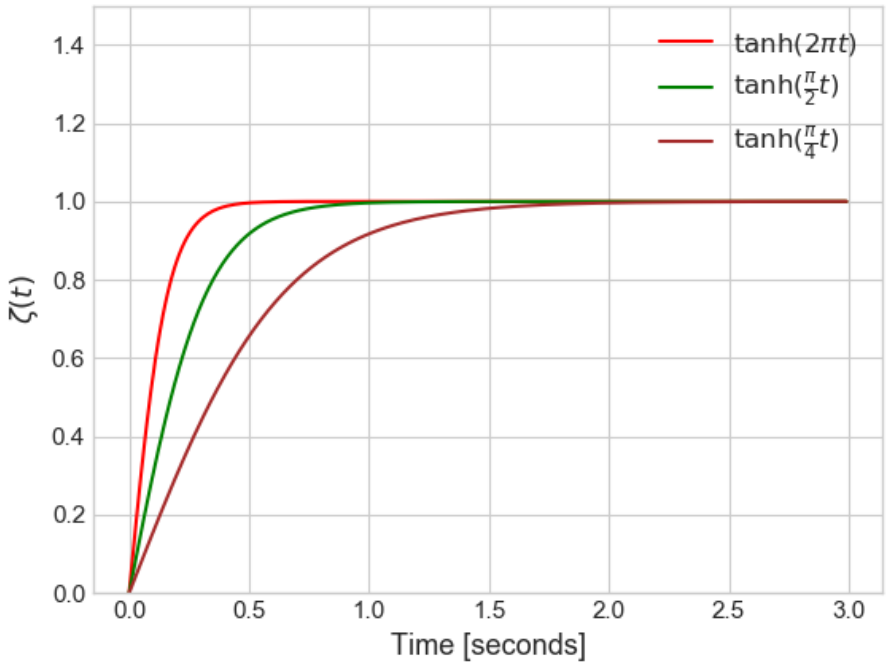

Figure 18: Hyperbolic tangent for different zeta values.

On Figures 30, 30, 31, 31, different trajectories of the robot following a straight line with initial misalignments are shown. By using VEW, overshoots are minimal.

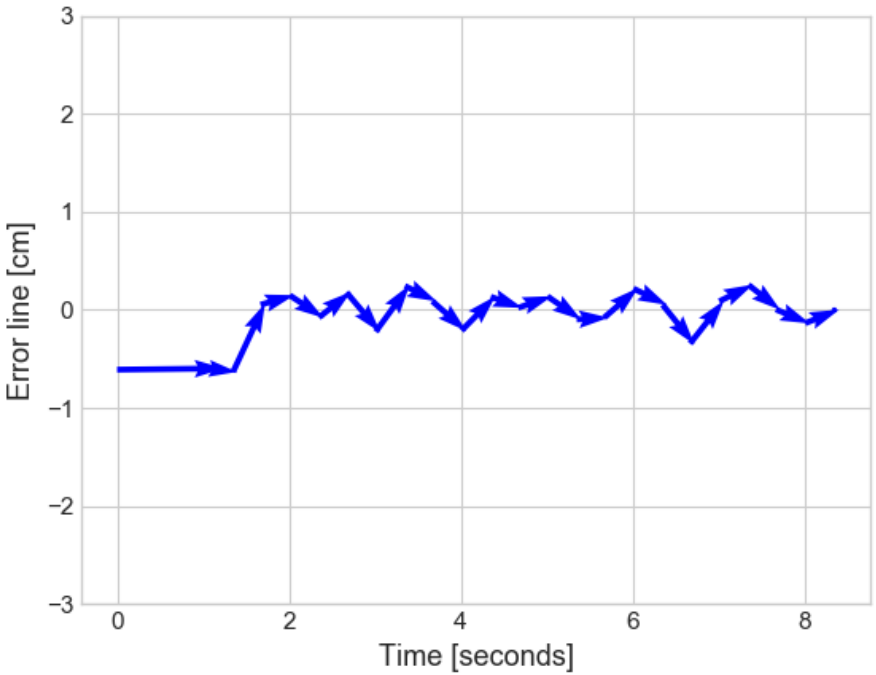

Figure 19: Small initial misalignment of the line to the left.

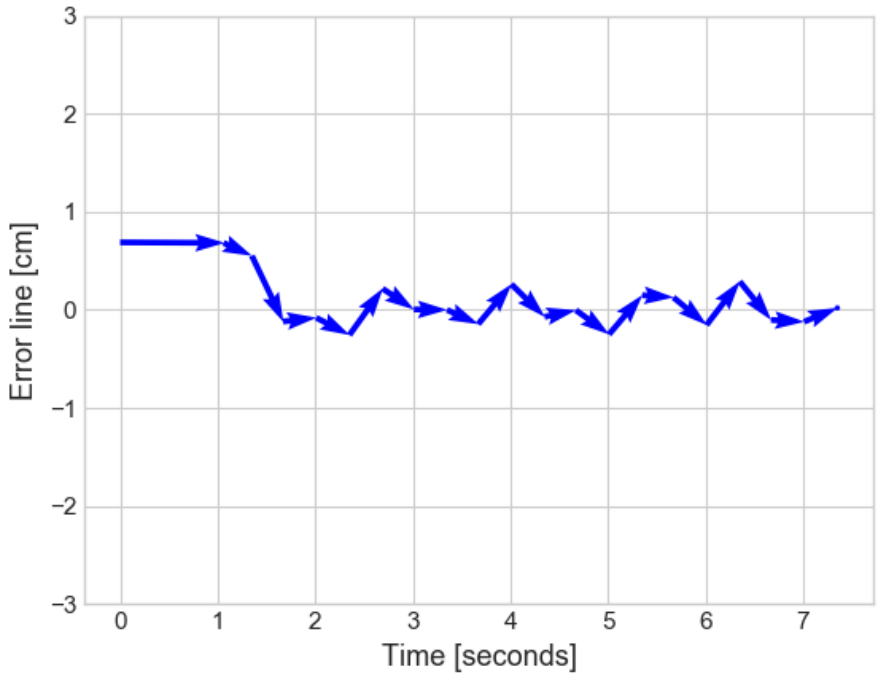

Figure 20: Small initial misalignment of the line to the right.

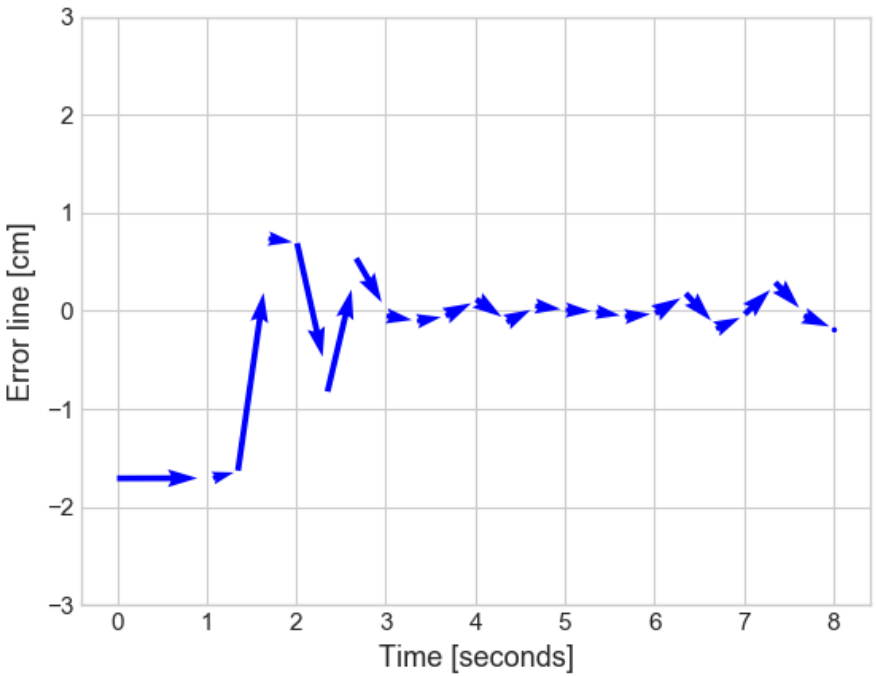

Figure 21: Big initial misalignment of the line to the left.

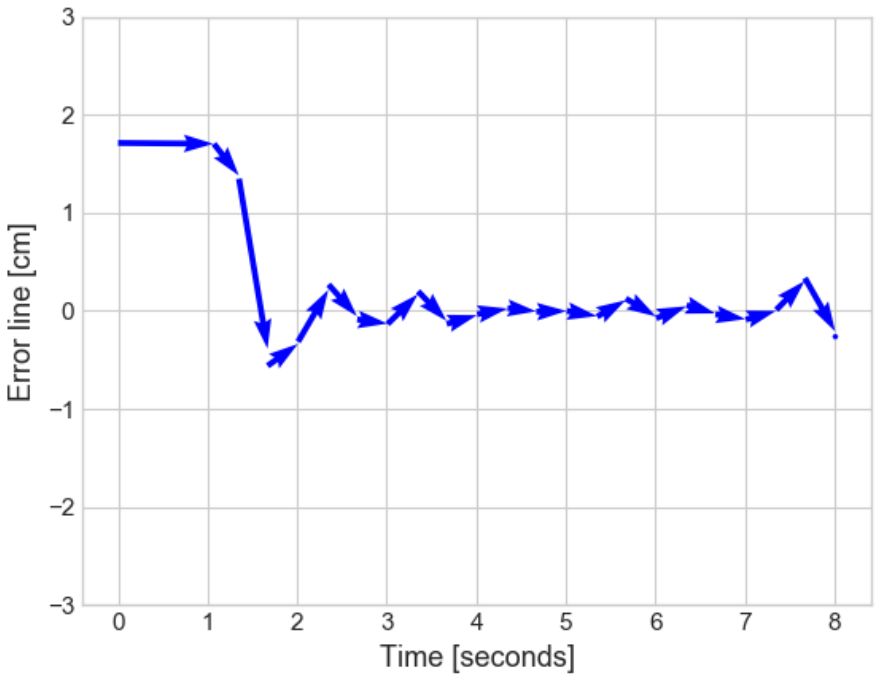

Figure 22: Big initial misalignment of the line to the right.

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

$$\theta(t) = \frac{2vt}{b} + \theta_0 \tag{4}$$

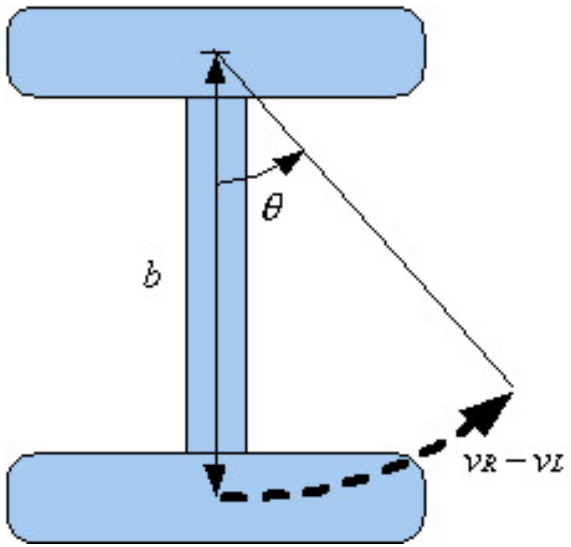

Figure 23: Rotation of COEX-1

Using Eq.4 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 41).

Overview

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

- Compute the distance travelled since the last intersection (slide 36)
- Detect an intersection (slide 39)
- Turn to the desired intersection (slide 41)

Distance is naively computed by summing $v_{i-1:i}\,\Delta t_{i-1:i}$. However it is found experimentally that errors are sufficiently small (see Fig.36).

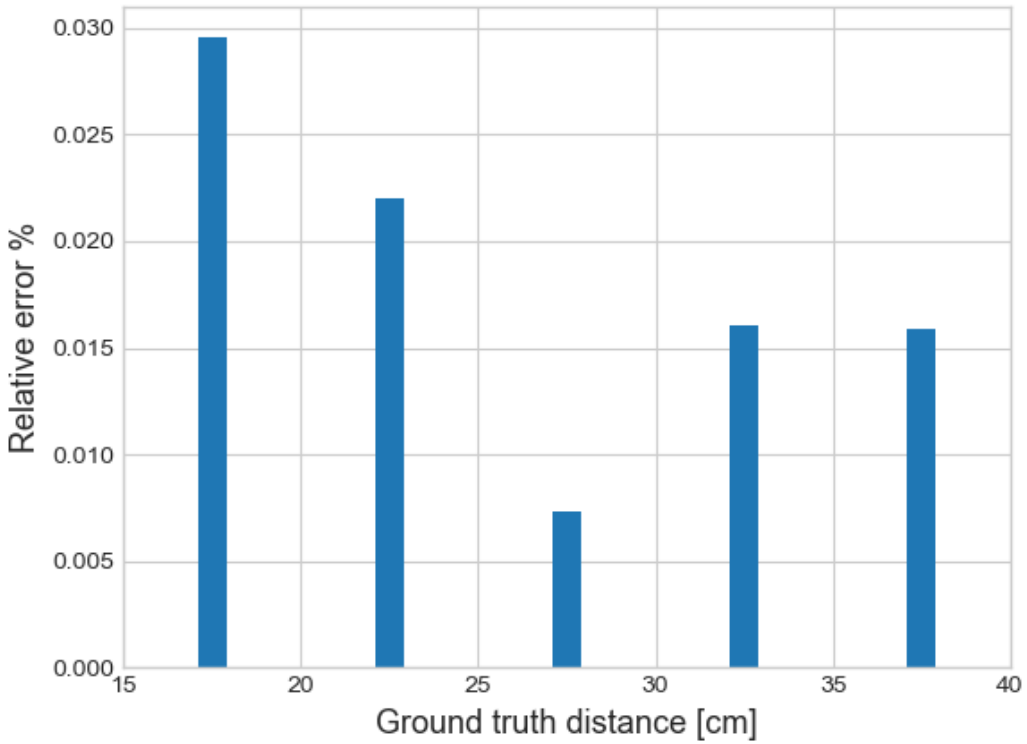

Figure 24: Relative error on measure (mean on a sample per ground truth, sample size = 8).

Having a small mean error and no overlap between measures (see below), discretization with a chuck size of 5 cm seems to be a reasonable solution to remove uncertainty.

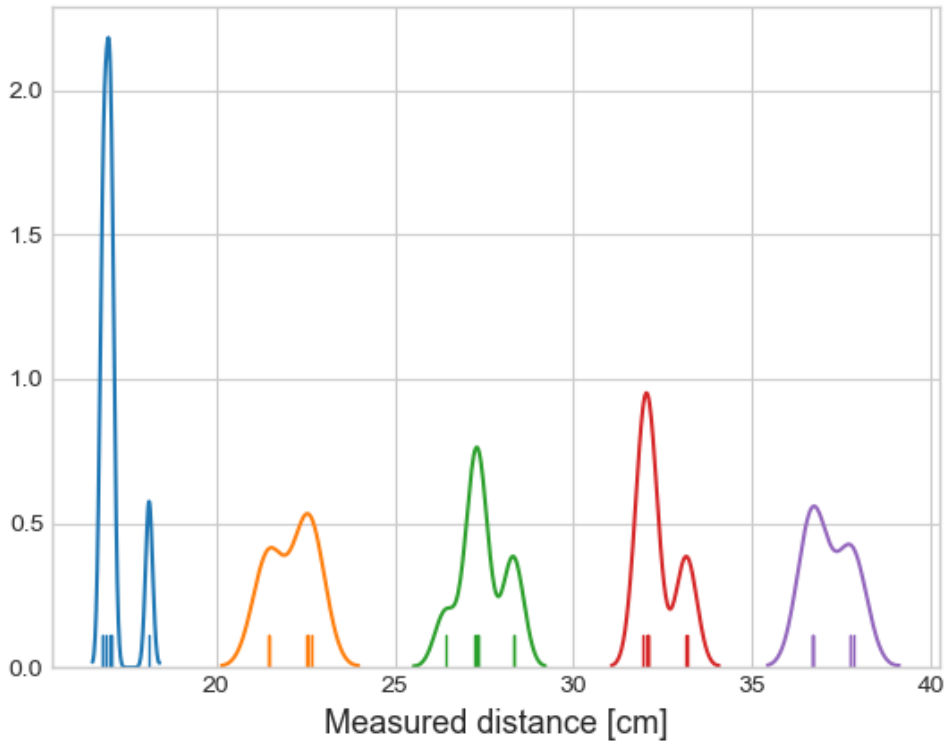

Figure 25: Distributions of sample of measures for each ground truth distance.

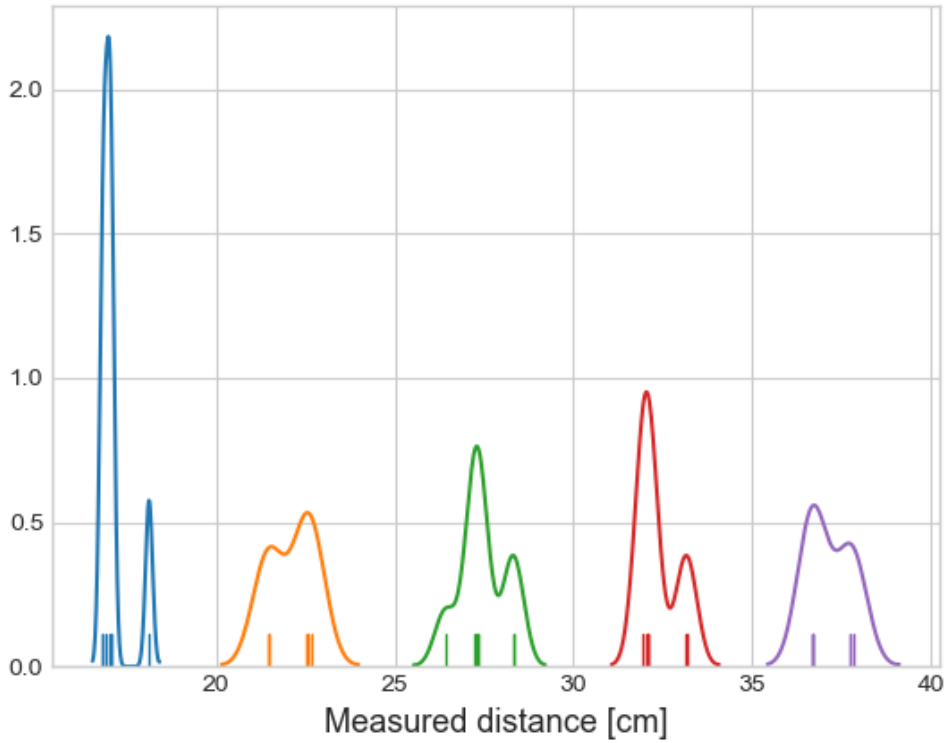

Figure 26: Map + explain.

To avoid false positives when detecting intersections, COEX-1 uses a solution based on a majority vote idea. From the moment the sensor reads an intersection, we compute all (5) for a predetermined distance ($\approx$ width of the black line)

$$y_{loc} = \underset{x \in \{0,1\}}{\text{argmax}}\, \text{mode}(x_{loc}) \tag{5}$$

At the end of that distance, the decision rule is given by

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

The type of intersection is inferred from set of x_{loc}. The different types:

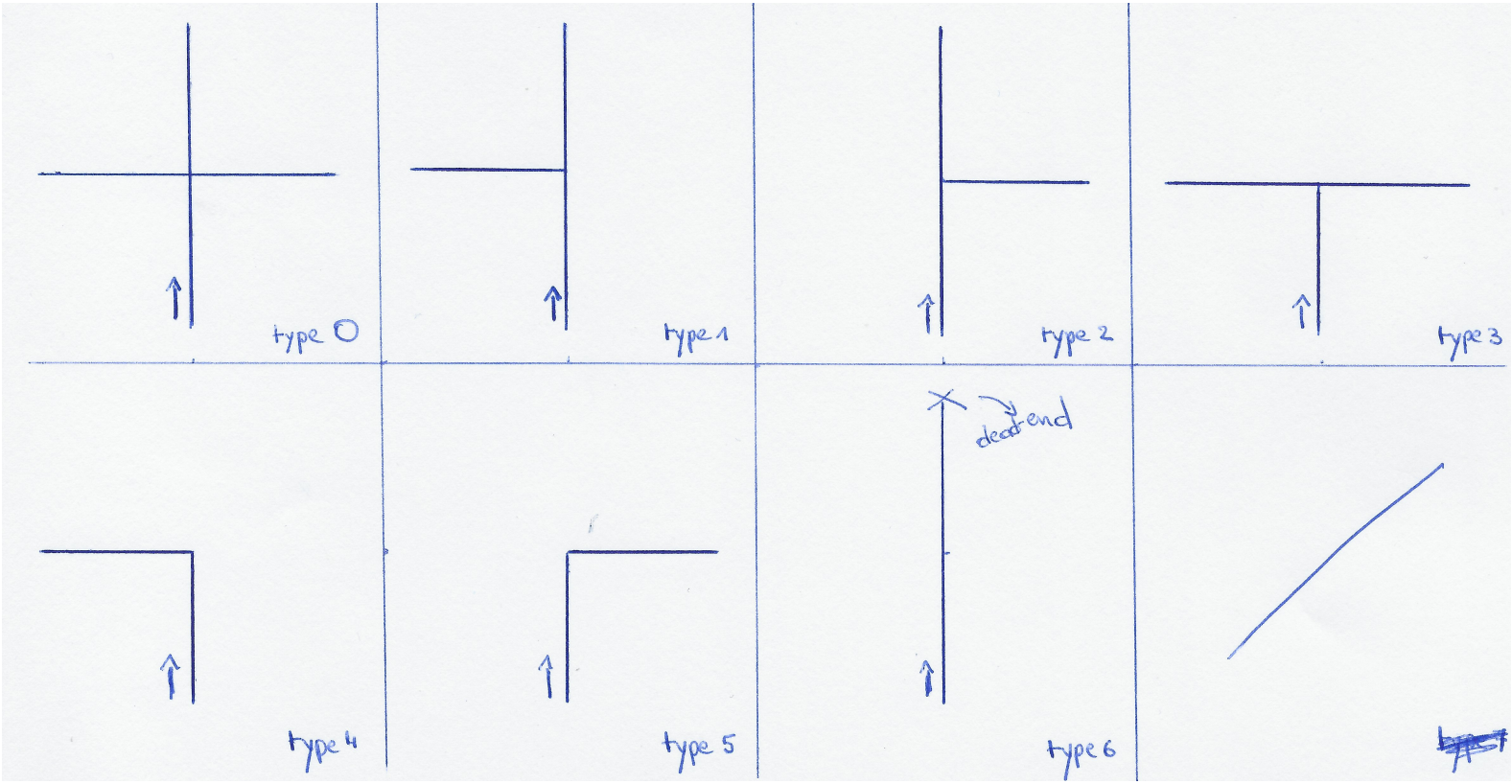

Figure 27: The 7 types of intersections

(TODO: write explanation) 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

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

(TODO: write explanation)

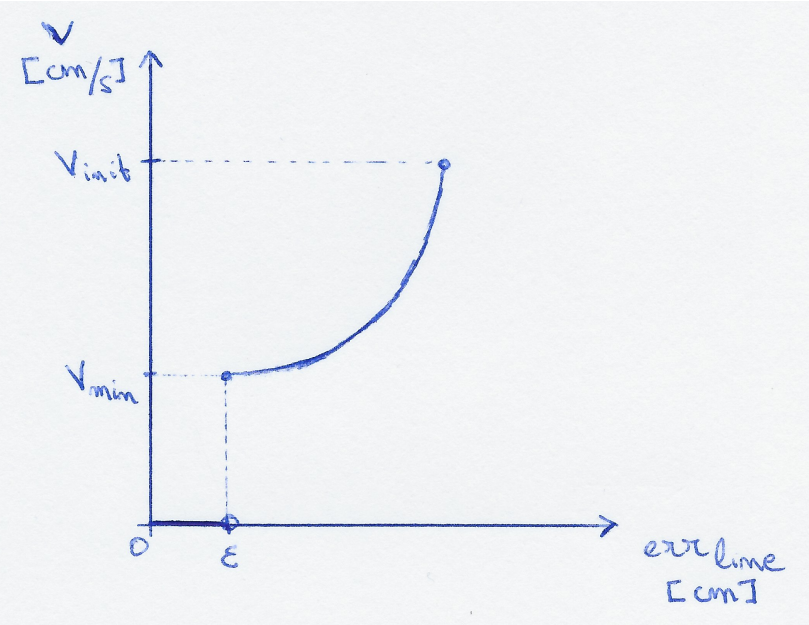

Figure 28: The 7 types of intersections

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

Overview

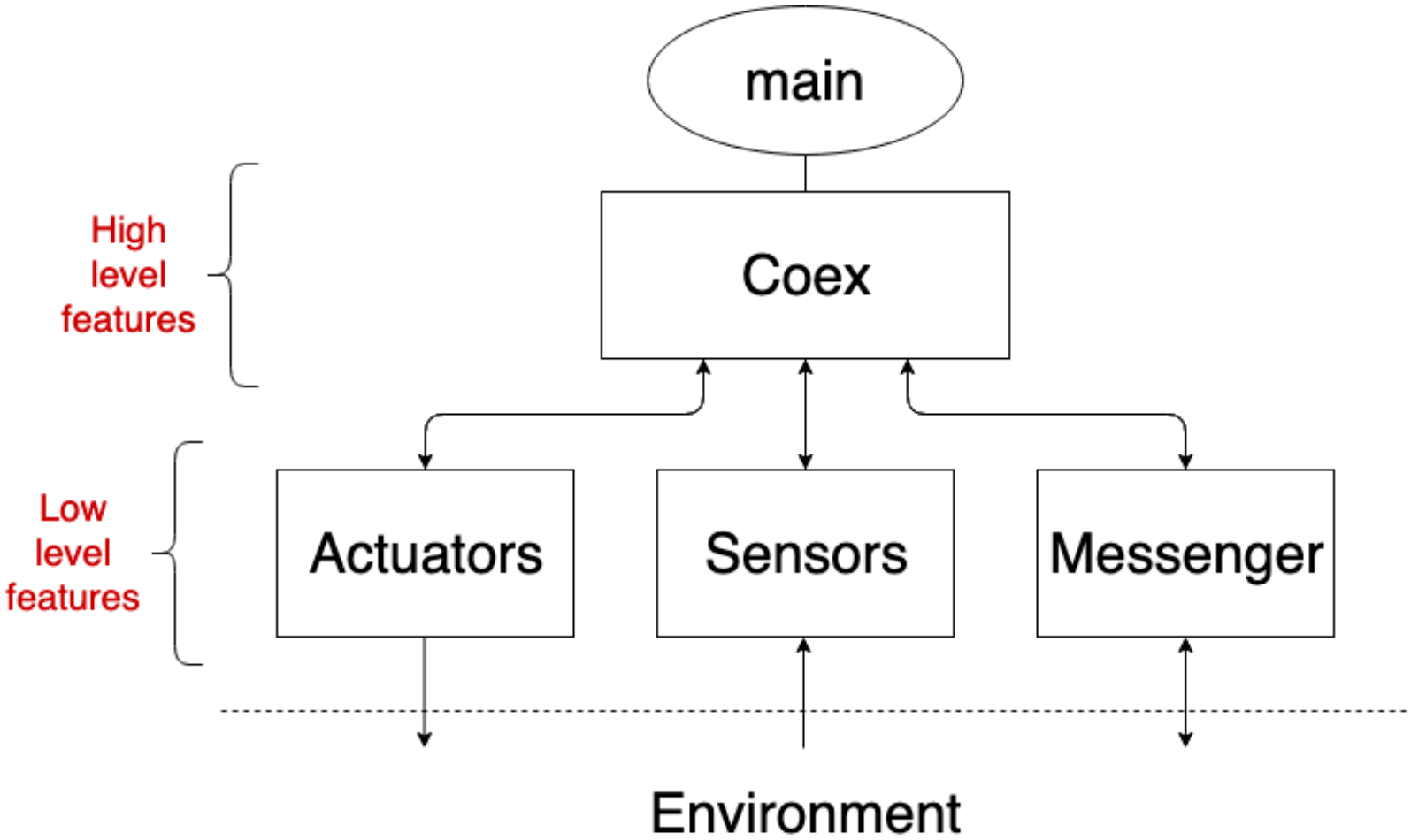

Figure 29: Architecture of the code