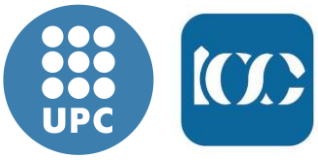


End of Degree Project of Physics Engineering

Control design and implementation for a line tracker vehicle

Ivan Prats Martinho

Domingo Biel

Arnau Dòria

June 2016

Index:

Summary

The main goal of this work is to build a two-wheeled robot emulating a vehicle in a road. The aim of the overall project is to apply control techniques to different scenarios of communicated vehicles, such as: platooning, autonomous vehicles, overtaking maneuvers, automatic vehicle lane change, or shockwave traffic jam reduction.

The first step, and the aim of this BSc thesis, is: i) to obtain the dynamic model of the robot tracking a path, ii) to design control algorithms for obtaining a desired performance, iii) to assemble the robot, and iv) to test the robot in a real path.

From the robot kinematics, a nonlinear dynamics is derived. The used approach was not reported in the literature, and is the first contribution of the work. Then, using linear approximation, a proportional-integral (PI) controller is designed and validated under numerical simulations using Matlab/Simulink software.

The last task of this work consists in assembling the robot hardware. This prototyping part included the robot mechanics (chassis, wheels,...), sensor implementation (encoders, reflective optical sensor,...) and coding the Digital Signal Processor (DSP). Finally, some real experiments of a robot tracking a path (a straight line with a sudden change of angle, a circle, and a sinusoid trajectory) has been performed to test the designed system.

Objectives

The objectives (with their main inner tasks) of this Project are:

1. Model the line tracker vehicle.
   1.1. Review the modelling of the line tracker vehicle in the bibliography.
   1.2. Choose the proper model taking into account the accuracy and the complexity of the model.
   1.3. Simulate the model in Matlab-Simulink.
   1.4. Validate the model with experimental measures.
2. Design and validate the control to track a path by numerical simulations
   2.1. Design several controllers to the track the line.
   2.2. Develop a set of numerical tests to study the stability of the controlled system.
   2.3. Study of the robustness of the chosen models in front of the variation of the parameters used in the models.
3. Build the line tracker vehicle
   3.1. Choose and evaluate the vehicle motors
   3.2. Learn the programming and compiler software that works with the microcontroller STM32F4-Discovery.
   3.3. Implement the different sensors (infrared and proximity ultrasound sensors) and build the signal acquisition chains.
   3.4. Program the preliminary software to test the motor drivers.
4. Program the control in the microprocessor
   4.1. Develop the program in the microprocessor code.
   4.2. Tune the parameters of the control according to the real measures.
5. Evaluate the performance of the control
   5.1. Develop a set of experimental tests in order to validate the control design.
   5.2. Test different controllers and compare their performances.

Chapter 1: Modeling and Control Design

After looking among dozens of articles for a proper kinematic model of a dual wheeled line tracker vehicle, we only found two articles that used a model similar to the one we wanted. But neither of the articles presented a model that could be applied to our line tracker vehicle.

At the beginning we tried following the model presented in the first article[1], but we did not like the fact the model did not pay enough attention to the trajectory and how it varies the dynamics of the vehicle.

From the second article[2] we get the idea of parametrizing the trajectory that the vehicle has to follow in order to obtain the kinematic model of the vehicle, and the definition of the state variables we will use. But in order to adapt it to the sensors of the vehicle, we had to change them slightly.

We then had to create our own model and find its dynamical equations on our own.

[1] Yulin Zhang, Daehie Hong, Jae H. Chung, and Steven A. Velinsky, *Dynamic Model Based Robust Tracking Control of a Differentially Steered Wheeled Mobile Robot* (1998)
[2] Pascal Morin, Claude Samson. *34. Motion Control of Wheeled Mobile Robots*

1. 1 The dual wheeled vehicle

In the following illustration is presented our dual wheeled vehicle and the most important parameters and variables we will be using.

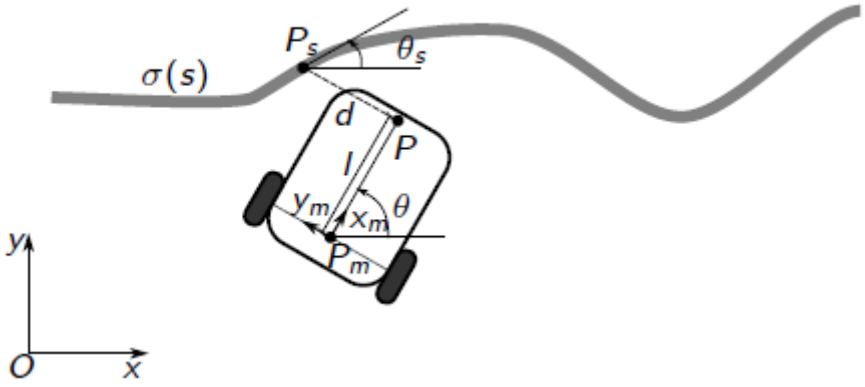


Figure I.1: Illustration of the chosen model and its parametrization.

The system we are going to study is made of two elements, each one having its own coordinate system: the trajectory and the vehicle.

The kinematic model of the unicycle-type mobile robot point P_m, where the middle of the wheel axis is, is given by the equations:

$$\begin{cases} \dot{x} = \cos(\theta)\, u_1 \\ \dot{y} = \sin(\theta)\, u_1 \\ \dot{\theta} = u_2 \end{cases} \quad \text{(I.1)}$$

With:

$$u_1 = \frac{r}{2}(\omega_r + \omega_l);\ u_2 = \frac{r}{2R}(\omega_r + \omega_l); \quad \text{(I.2)}$$

Where *r* is each wheel's radius, *R* is the distance between the two wheels, and ω_r (and respectively ω_l) are the angular velocity of the right (and left) wheels.

The purpose of the control is to make the robot follow the trajectory; for this we will define the following elements of the system:

- P_m: the middle point of the wheel axis of the robot as explained above.
- Axes $(x_m; y_m)$: a coordinate system whose origin is the point P_m. The axis $\overrightarrow{x_m}$ is perpendicular to the wheel axis, and $\overrightarrow{y_m}$ is parallel to it, and its direction is towards the left wheel of the vehicle. Also, θ is the angle between the axes $\vec{x}$ of the global coordinate system, and $\overrightarrow{x_m}$. This coordinate system will be usually referred to as the "relative coordinate system" in this paper.
- $P(x, y)$: the measure point of the robot, where the infrared sensor is located. This point is located at an *l* distance from P_m in direction $\overrightarrow{x_m}$. This is the point that has to be over the trajectory.
- $P_S(x, y)$: the point in the trajectory that the vehicle is meant to follow. It is defined by the point in the trajectory $\sigma(s)$ that is in the coordinates: $(0, d)$ of the relative coordinate system of the vehicle. More details of this point will be given in the next section *I.ii.*
- $\theta_S(s)$: the angle between the tangent to the trajectory $\sigma(s)$ in the point $P_S(x, y)$, and the $\vec{x}$ axis of the global coordinate system.

1.2 The model

In order to safely say that the robot is following the trajectory, two conditions must be met:

- $P_S = P$ or in other words: $d = 0$
- $\theta \sim \theta_S$ This condition guaranties that the vehicle is properly aligned with the trajectory. For now we will leave it like they have to be approximately equal, later in this section the exact relation will be calculated.

We are then interested in knowing the dynamics of these two state variables: *d* and the relation between θ and θ_S. But we will also need the dynamics of the sate variable *s*, in order to make the vehicle move at a constant velocity.

The coordinates of the point P_S can be given by two equations:

$$P_S = P_m + R(\theta)\binom{l}{d} = \binom{\sigma_x(s)}{\sigma_y(s)} \tag{I.3}$$

Where $R(\theta)$ is the 2D coordinate rotation matrix:

$$R(\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$$

Let us differentiate the equation (I.3):

$$\dot{P_S} = \dot{P_m} + R(\theta)\begin{pmatrix} 0 \\ \dot{d} \end{pmatrix} + \frac{\partial R(\theta)}{\partial \theta}\dot{\theta}\begin{pmatrix} l \\ d \end{pmatrix} = \dot{s}\begin{pmatrix} \frac{\partial \sigma_x(s)}{\partial s} \\ \frac{\partial \sigma_y(s)}{\partial s} \end{pmatrix}$$

Now we substitute $\dot{P_m}$ and $\dot{\theta}$ with the equations of (I.1):

$$\binom{\cos(\theta)}{\sin(\theta)}\mathrm{u}_1 + R(\theta)\begin{pmatrix} 0 \\ \dot{d} \end{pmatrix} + \frac{\partial R(\theta)}{\partial \theta}\binom{l}{d}\mathrm{u}_2 = \dot{s}\begin{pmatrix} \frac{\partial \sigma_x(s)}{\partial s} \\ \frac{\partial \sigma_y(s)}{\partial s} \end{pmatrix}$$

We will write $\frac{\partial \sigma_x(s)}{\partial s}$ as $\partial\sigma_x$ (and the respective$\partial\sigma_y$) in order to get cleaner equations from now on. Rearranging to isolate $\dot{d}$:

$$\begin{pmatrix} 0 \\ \dot{d} \end{pmatrix} = R^{-1}(\theta)\left(\binom{\partial\sigma_x}{\partial\sigma_y}\dot{s} - \binom{\cos(\theta)}{\sin(\theta)}\mathrm{u}_1 - \frac{\partial R(\theta)}{\partial \theta}\binom{l}{d}\mathrm{u}_2\right)$$

Where:

$$R^{-1}(\theta) = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix};\ \frac{\partial R(\theta)}{\partial \theta} = \begin{pmatrix} -\sin(\theta) & -\cos(\theta) \\ \cos(\theta) & -\sin(\theta) \end{pmatrix}$$

We get:

$$\begin{pmatrix} 0 \\ \dot{d} \end{pmatrix} = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix}\begin{pmatrix} \partial\sigma_x \\ \partial\sigma_y \end{pmatrix}\dot{s} - \binom{1}{0}\mathrm{u}_1 - \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}\binom{l}{d}\mathrm{u}_2$$

And finally:

$$\begin{cases} \dot{d} = -lu_2 - (\partial\sigma_x \sin(\theta) - \partial\sigma_y \cos(\theta))\dot{s} \\ \dot{s} = \frac{u_1 - d*u_2}{\partial\sigma_x \cos(\theta) + \partial\sigma_y \sin(\theta)} \end{cases} \quad \text{(I.4)}$$

We now know the dynamics of *d*, but we do not have anywhere θ_S and we still have the terms $\frac{\partial\sigma_x(s)}{\partial s}$ and $\frac{\partial\sigma_y(s)}{\partial s}$. So we will substitute the following equations in (I.4):

$$\frac{\partial\sigma_x(s)}{\partial s} = \cos(\theta_S)\,;\ \frac{\partial\sigma_y(s)}{\partial s} = \sin(\theta_S)$$

$$\begin{cases} \dot{d} = -lu_2 - (\cos(\theta_S)\sin(\theta) - \sin(\theta_S)\cos(\theta))\dot{s} \\ \dot{s} = \dfrac{u_1 - d * u_2}{\cos(\theta_S)\cos(\theta) + \sin(\theta_S)\sin(\theta)} \end{cases}$$

And we apply the trigonometric relations:

$$sin(a-b) = sin(a)\,cos(b) - cos(a)\,sin(b)$$

$$cos(a-b) = cos(a)\,cos(b) + sin(a)\,sin(b)$$

$$\begin{cases} \dot{d} = -lu_2 - (\sin(\theta - \theta_S))\dot{s} \\ \dot{s} = \frac{u_1 - d*u_2}{\cos\,(\theta - \theta_S)} \end{cases} \quad \text{(I.5)}$$

Knowing that:

$$\dot{\theta}_s = \frac{\partial\theta_S(s)}{\partial s}\dot{s}$$

We define: $\theta_e = \theta - \theta_s$ and substituting it in (I.5) we get the final dynamics of the model we will use:

$$\begin{cases} \dot{d} = -\mathrm{l}u_2 - \tan(\theta_e)(u_1 - du_2) \\ \dot{s} = \frac{u_1 - du_2}{\cos(\theta_e)} \\ \dot{\theta}_e = u_2 - \frac{c(s)}{\cos(\theta_e)}(u_1 - du_2) \end{cases} \quad \text{(I.6)}$$

Where c(s) is the curvature of σ(s) :

$$c(s) = \frac{\partial\theta_s}{\partial s} \quad \text{(I.7)}$$

Working trajectory

As we previously explained, in order for the vehicle to properly follow the trajectory; two conditions must be met:

$$d^* = 0\ ;\ \theta_e^* \sim 0\ ;$$

But, we also would like that the vehicle goes at a constant linear velocity, hence:

$$\dot{s}^* = v\ ;$$

Being v a constant. We will have 2 control signals: u_1 and u_2, and the aim is to find the working trajectory that makes: $d^* = 0\ ;\ \dot{s}^* = v\ ;\ \dot{\theta}_e = 0$

Applying these conditions to the equations of (I.6) we get:

$$\begin{cases} 0 = -lu_2^* - \tan(\theta_e^*)(u_1^* - d^* u_2^*) \\ v = \dfrac{u_1^* - d^* u_2^*}{\cos(\theta_e^*)} \\ 0 = u_2^* - \dfrac{c(s)}{\cos(\theta_e^*)}(u_1^* - d^* u_2^*) \end{cases}$$

We will leave the term $c(s)$ unchanged for now and it will be treated as an unknown parameter later.

Applying $d^* = 0$: we get a 3x3 system with the following variables: $(u_1^*\ ;\ u_2^*\ ;\ \theta_e^*)$. We solve it and get:

$$\begin{cases} u_1^* = v\sqrt{1 - \big(lc(s)\big)^2} \\ u_2^* = c(s)v \\ \theta_e^* = \operatorname{asin}\big(-lc(s)\big)\ ; \end{cases} \tag{I.8}$$

Linearization

We linearize the three dynamic equations around the working trajectory defined in (I.8):

$$u_1^* = v\sqrt{1-\left(lc(s)\right)^2}\,;\; u_2^* = \; c(s)v\,;\; d^* = 0\,;\; \theta_e^* = \operatorname{asin}\left(-lc(s)\right); \qquad \text{(I.9)}$$

$$\dot{d} \;\cong \dot{d}^* + \left(\frac{\partial \dot{d}}{\partial d}\right)^* \Delta d + \left(\frac{\partial \dot{d}}{\partial u_2}\right)^* \Delta u_2 + \left(\frac{\partial \dot{d}}{\partial u_1}\right)^* \Delta u_1 + \left(\frac{\partial \dot{d}}{\partial \theta_e}\right)^* \Delta \theta_e$$

Deriving the first equation of (I.6) and substituting its variables for the values in (I.9) we get:

$$\dot{d} \;\cong\; -\frac{vlc(s)^2}{\sqrt{1-\left(lc(s)\right)^2}}\tilde{d} - \frac{v}{\sqrt{1-\left(lc(s)\right)^2}}\widetilde{\theta_e} + \frac{lc(s)}{\sqrt{1-\left(lc(s)\right)^2}}\widetilde{u_1} - l\widetilde{u_2} \qquad \text{(I.10)}$$

Where:

$$\begin{cases} \tilde{d} = d - d^* = d \\ \widetilde{\theta_e} = \theta_e - \theta_e^* = \theta_e - \operatorname{asin}\left(-lc(s)\right) \\ \widetilde{u_1} = u_1 - u_1^* = u_1 - v\sqrt{1-\left(lc(s)\right)^2} \\ \widetilde{u_2} = u_2 - u_2^* = u_2 - c(s)v \\ \tilde{s} = s - s^* \end{cases} \qquad \text{(I.11)}$$

Doing the same for the rest of the equations in (I.6) we find:

$$\dot{s} \;\cong \frac{-c(s)v\,\tilde{d} - lc(s)v\,\widetilde{\theta_e} + \widetilde{u_1}}{\sqrt{1-\left(lc(s)\right)^2}} + v \qquad \text{(I.12)}$$

$$\dot{\theta_e} \;\cong\; \frac{vc(s)^2}{\sqrt{1-\left(lc(s)\right)^2}}\,\tilde{d} + \frac{vlc(s)^2}{\sqrt{1-\left(lc(s)\right)^2}}\,\widetilde{\theta_e} - \frac{c(s)}{\sqrt{1-\left(lc(s)\right)^2}}\widetilde{u_1} + \widetilde{u_2} \qquad \text{(I.13)}$$

Where (I.11) also applies.

Seeing as there is no dependence with the state variable *s* and that we will only be interested in controlling the state variables: $(\tilde{d}\,;\widetilde{\theta_e})$, we will only use the subsystem that we obtain doing the variable change in (I.11) with the equations: (I.10), (I.12) and (I.13), and putting it in matrix form:

$$\begin{pmatrix} \dot{\tilde{d}} \\ \dot{\widetilde{\theta_e}} \end{pmatrix} = \frac{v}{\sqrt{1-\left(lc(s)\right)^2}} \begin{pmatrix} -lc(s)^2 & -1 \\ c(s)^2 & lc(s)^2 \end{pmatrix} \begin{pmatrix} \tilde{d} \\ \widetilde{\theta_e} \end{pmatrix} + \frac{1}{\sqrt{1-\left(lc(s)\right)^2}} \begin{pmatrix} lc(s) \\ -c(s) \end{pmatrix} \tilde{u}_1 + \begin{pmatrix} -l \\ 1 \end{pmatrix} \tilde{u}_2 \qquad \text{(I.14)}$$

Maximum Curvature

The term $\sqrt{1-\left(lc(s)\right)^2}$ is present in some of the past equations, for example:

$$u_1^* = v\sqrt{1-\left(lc(s)\right)^2}$$

Where u_1^* is the equilibrium value of the sum of the two linear wheel speeds. A sum of linear wheel speeds cannot be imaginary, therefore:

$$1 > \left(lc(s)\right)^2$$
$$\frac{1}{l} > c(s)$$

This makes a lot of sense: supposing that the wheels cannot go backwards: the two wheel vehicle cannot follow a line that turns faster than the distance from the measuring point *P* to the wheel axis. In the case of our vehicle: $l = 0{,}06m$, so:

$$16{,}67 > c(s)$$

$$c_{MAX} = 16{,}67\frac{1}{m} \qquad \text{(I.15)}$$

We can assume that this model fails for curvatures bigger than c_{MAX}

Transfer function

In order to make a proper theoretical analysis, we will suppose that:

$$\tilde{u}_1 = 0\ ; u_1 = u_1^*$$

Our control signal will be the input $\tilde{u}_2$. And we wish to control the state variable $\tilde{d}$. The transfer function of the linearized system can be obtained by:

$$G(p) = C(pI - A)^{-1}B$$

Note that we will use the letter "p" as the usual "s" variable for the Laplace realm because we have been using the letter "s" for the parametrization of the curve.

Where:

$$\mathrm{C} = (1 \quad 0)\ ;\ \mathrm{B} = \begin{pmatrix} -\mathrm{l} \\ 1 \end{pmatrix};\ A = \frac{\mathrm{v}}{\sqrt{1 - \left(\mathrm{lc(s)}\right)^2}} \begin{pmatrix} -\mathrm{lc(s)}^2 & -1 \\ \mathrm{c(s)}^2 & \mathrm{lc(s)}^2 \end{pmatrix}$$

Giving us:

$$G(p) = (1\ 0)\left(\begin{pmatrix} p & 0 \\ 0 & p \end{pmatrix} - \frac{v}{\propto}\begin{pmatrix} -lc(s)^2 & -1 \\ c(s)^2 & lc(s)^2 \end{pmatrix}\right)^{-1}\begin{pmatrix} -l \\ 1 \end{pmatrix}$$

$$G(p) = (1 \quad 0)\ \frac{\frac{\propto}{v}}{\left(\frac{\propto p}{v}\right)^2 - (lc(s)^2)^2 + c(s)^2}\begin{pmatrix} \frac{p \propto}{v} - lc(s)^2 & -1 \\ c(s)^2 & \frac{p \propto}{v} + lc(s)^2 \end{pmatrix}\begin{pmatrix} -l \\ 1 \end{pmatrix}$$

$$\frac{\tilde{D}(p)}{\tilde{U}_2(p)} = G(p) = -\frac{lp + v\alpha}{p^2 + c^2v^2} \tag{I.16}$$

The transfer function is only negative because of the direction in which we defined u_2. If we were to change the sign of u_2 in all the analysis we would get to the same $G(p)$ but with positive sign.

1.3 Control Design

In this section we will proceed to analyze some different controllers in order to have a better idea of the type of controller that would benefit more the vehicle. The aim of the vehicle's controller is to make $d \to 0$, and for this we will use the control variable $\tilde{u}_2$.

Proportional Controller

Replacing the control law $\tilde{u}_2 = k_p \tilde{d}$ in the equations (I.14), the closed loop dynamics yields:

$$\begin{pmatrix} \dot{\tilde{d}} \\ \dot{\tilde{\theta}_e} \end{pmatrix} = \frac{v}{\alpha} \begin{pmatrix} -lc(s)^2 & -1 \\ c(s)^2 & lc(s)^2 \end{pmatrix} \begin{pmatrix} \tilde{d} \\ \tilde{\theta_e} \end{pmatrix} + \begin{pmatrix} -l \\ 1 \end{pmatrix} (k_p \tilde{d})$$

$$\begin{pmatrix} \dot{\tilde{d}} \\ \dot{\tilde{\theta}}_e \end{pmatrix} = \frac{v}{\alpha} \begin{pmatrix} -lc(s)^2 - k_p l \frac{\alpha}{v} & -1 \\ c(s)^2 + k_p \frac{\alpha}{v} & lc(s)^2 \end{pmatrix} \begin{pmatrix} \tilde{\mathrm{d}} \\ \tilde{\theta}_{\mathrm{e}} \end{pmatrix}$$

With:

$$\alpha = \sqrt{1 - \big(lc(s)\big)^2}$$

If the vehicle follows a straight line: $c(s) = 0$, so: $\alpha = 1$, then:

$$\begin{pmatrix} \dot{\tilde{d}} \\ \dot{\tilde{\theta}}_e \end{pmatrix} = \begin{pmatrix} -k_p l & -v \\ k_p & 0 \end{pmatrix} \begin{pmatrix} \tilde{d} \\ \tilde{\theta}_e \end{pmatrix}$$

The eigenvalues of the closed loop matrix are

$$\lambda = \frac{-lk_p \pm \sqrt{l^2 k_p^2 - 4k_p v}}{2} = -\sigma \pm j\omega_d$$

In order to get the real part of the eigenvalues negative:

$$0 > -lk_p + \sqrt{l^2 k_p^2 - 4k_p v}$$

$$\big(lk_p\big)^2 > l^2 k_p^2 - 4k_p v$$

$$0 > -4k_p v$$

For a positive v: the eigenvalues of the system always have a negative real part. Getting the transfer function of the system from (I.16) and $\mathrm{C(s)} = \mathrm{k_p}$ we get the Open Loop transfer function:

$$G_{OL}(s) = -k_p \frac{lp + v}{p^2}$$

Which has two pure integrators. The static error is then zero. However, the open loop transfer function for a constant curvature $c(s) = c$ is:

$$G_{OL}(p) = k_p \frac{lp + v\sqrt{1 - (lc)^2}}{p^2 + c^2 v^2}$$

Which causes a static position error constant of: $\mathrm{k_{ssp}} = \lim_{\mathrm{p}\to 0} \mathrm{G_{OL}(p)} = \frac{\mathrm{k_p}\sqrt{1-(lc)^2}}{\mathrm{c^2 v}}$, that turns to a steady state (static position) error of:

$$\mathrm{e_{ssp}} = \frac{\mathrm{c^2 v}}{\mathrm{k_p}\alpha}$$

From the settling time value approximation: $\mathrm{t_s} \approx \frac{4}{\sigma}$

$$\mathrm{t_s} = \frac{4}{\sigma} = \frac{8}{\mathrm{lk_p}} = 53{,}33ms$$

This may seem unacceptable, but if we try for example with the value: $\mathrm{k_p} = 2500$. Parameters for this simulation: $c(s) = 14m^{-1}$; $l = 0{,}06m$; $v = 0{,}3\frac{m}{s}$. The Root locus of the open loop transfer function $k_p \frac{lp+v\sqrt{1-(lc)^2}}{p^2+c^2v^2}$ is calculated and shown in the next figure.

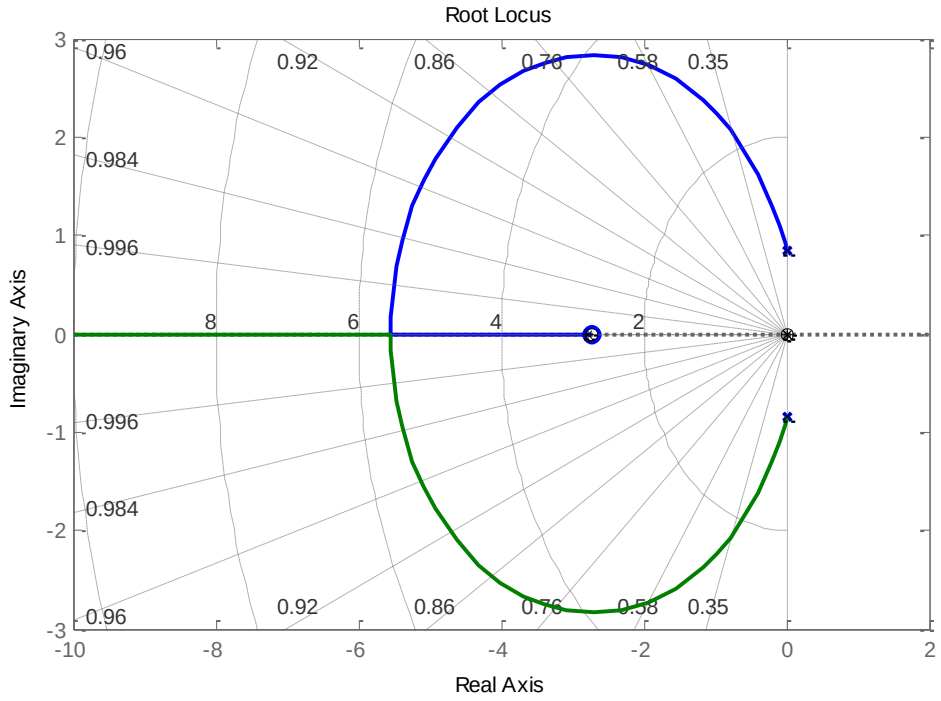


Figure I.2: Root Locus of the proposed proportional controller. The poles have almost reached the zeros at the current k_p. Calculations done with Matlab.

And the step response of this transfer function at the current k_p is:

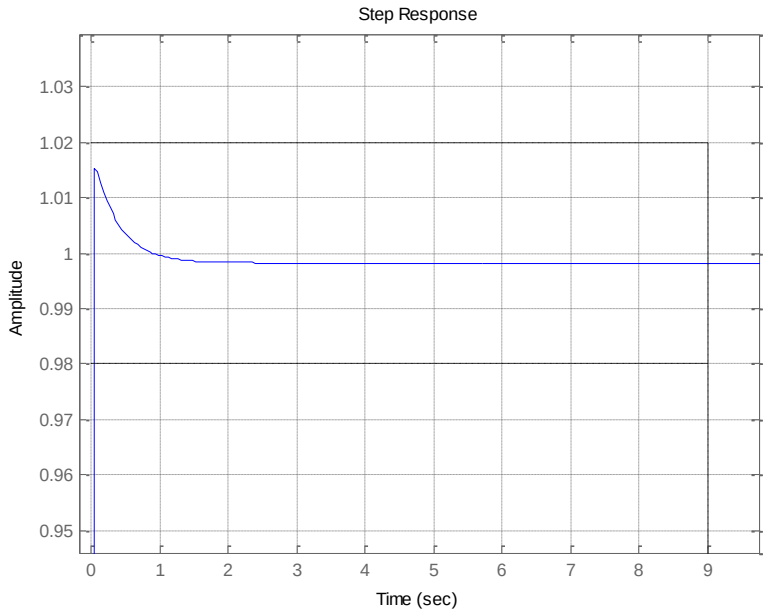


Figure I.3: Step response of the system with the same proportional controller.

The steady state error does not converge to 0, but with this gain it is almost negligible. We would not be able to see it with the vehicle. The overshoot (almost nonexistent) and the settling time are also acceptable.

We will try this controller with the vehicle in the chapter III: Experimentation.

Proportional Integral Controller

The transfer function for a PI controller is:

$$C(p) = \frac{k_p p + k_i}{p}$$

We will use the Routh Criteria to find the possible values of k_i, as a function of k_p. The characteristic equation of the closed loop system is given by: $D(s) = 1 + C(s)G(s) = 0$, then

$$D(p) = p(p^2 + l^2 c(s)^2) + \left(k_p p + k_i\right)(lp + v\alpha)$$

$$D(p) = p^3 + a_2 p^2 + a_1 p + a_0$$

Where: $a_2 = k_p l$, $a_1 = l^2 c^2 + k_p v\alpha + k_i l$, and $a_0 = k_i v\alpha$, and $\alpha = \sqrt{1 - \left(lc(s)\right)^2}$

$$\begin{array}{c|cc} 3 & 1 & a_1 \\ 2 & a_2 & a_0 \\ 1 & a_2 a_1 - a_0 & 0 \\ 0 & a_0 & 0 \end{array}$$

$$\begin{aligned} a_2 &= k_p l > 0 \\ a_1 &= l^2 c^2 + k_p v\alpha + k_i l > 0 \\ a_0 &= k_i v\alpha > 0 \end{aligned}$$

Which, assuming that $v, l > 0$ and $k_p, k_i > 0$:

$$a_2 a_1 - a_0 = k_p l(l^2 c^2 + k_p v\alpha + k_i l) - k_i v\alpha > 0$$

For any controller that we design, the fixed parameters are:

$$l = 0{,}06m\,;\, v = 0{,}3\frac{m}{s}$$

We will also be limited in the values that *d* can acquire:

$$d \in [-8; 8]mm$$

This limitation comes from the characterization of the Line sensor: more information on chapter (*II.iii*).

The open loop transfer function with the PI controller will then be:

$$G_{OL}(p) = \frac{k_p(p + a)}{p}\frac{lp + v\alpha}{p^2 + c^2 v^2}$$

We try with the same value *of* $k_p = 2500$, $k_i = 100$ *and* $c(s) = 14m^{-1}$ as before to be able to compare with the proportional control. With this value we will Matlab to plot the Root Locus of this open loop transfer function.

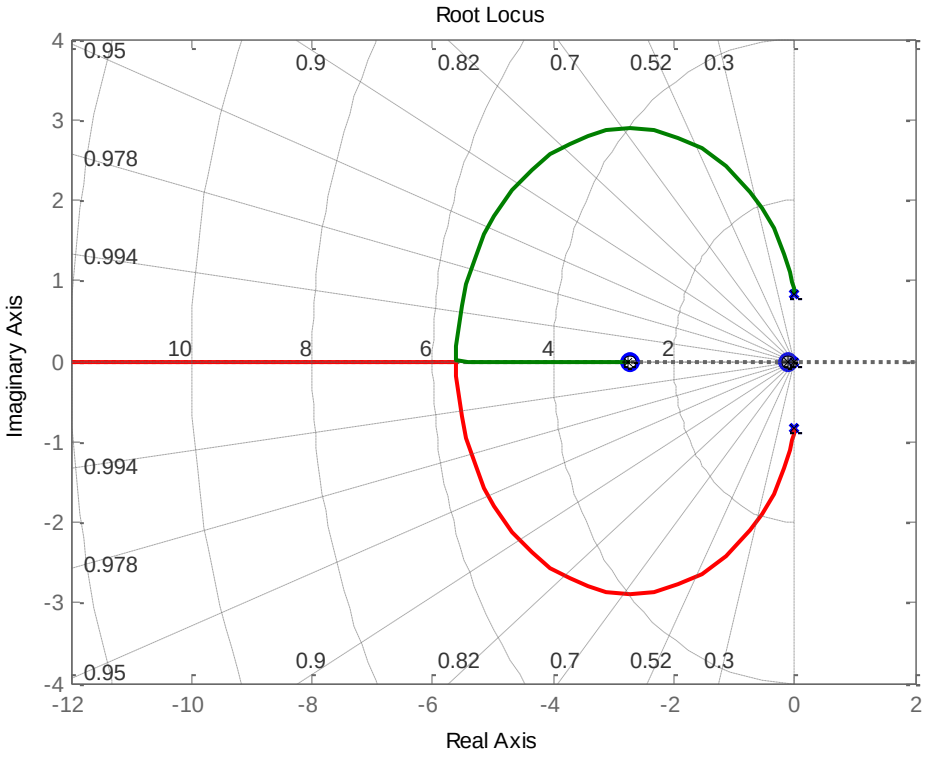


Figure I.4: Root Locus of the PI controller. The poles have almost reached the zeros at the current : k_p.

And the step response of this transfer function at the current k_p is:

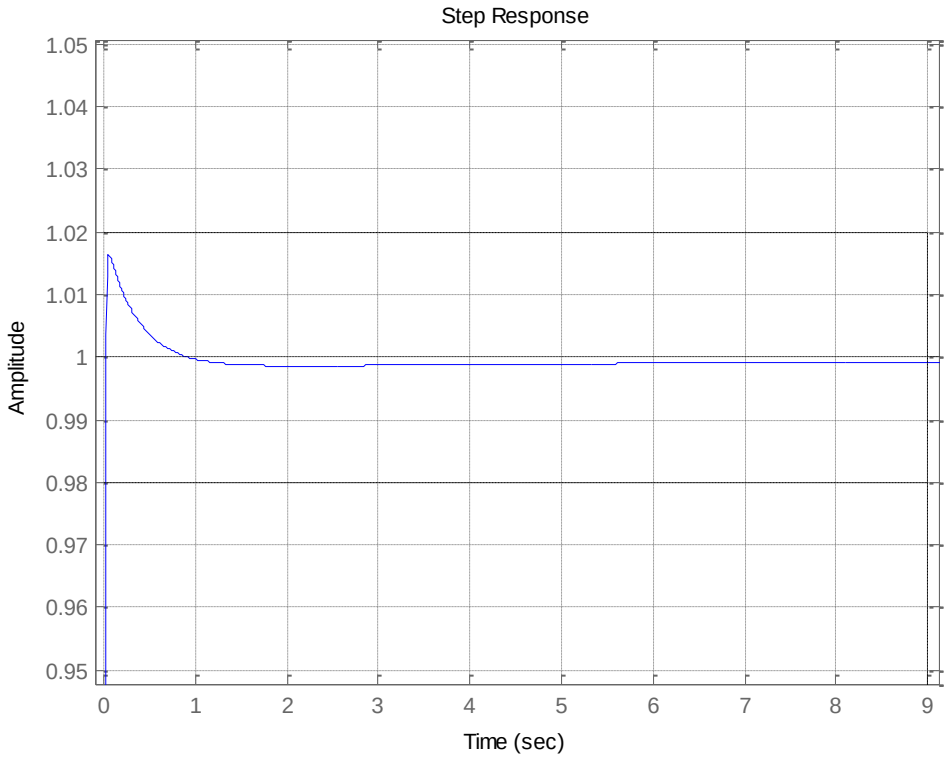


Figure I.5: Step response of the system with the same PI controller.

There are no clear improvements in the PI control when comparing it to the proportional one. Yes, the steady state error tends to zero, but it was not a big problem in the proportional controller anyway, and the step response is almost identical in both cases.

Comparing the two controllers with the information we have up to this point we get:

1)If we have a curvature of the trajectory constant and 0 ($c(s) = 0$): both controllers appear to be identical.
2)If we have a constant curvature, but bigger than 0 and smaller than c_{MAX} (a trajectory of a circle): the PI seems a bit better for making the steady state error 0, but the difference between the PI and the proportional could not be appreciated in the vehicle.
3)If we have a non-constant curvature ($c(s) = f(s)$): we do not know which one would be better at this stage of the project.

Chapter 2: Implementation

This chapter is devoted to present all the hardware needed to make the vehicle work (sensors, batteries, chassis, wheels…) and its implementation and characterization. In addition, the Digital Signal Processor (DSP), the software and the peripherals we used will be commented.

2.1 DC motors, driver, used sensors and hardware.

For this project we bought the Arduino kit "*Kit Robot LRE-EO2*" from www.leantec.es, without the included *Arduino UNO* board. This pack, see *FigureIV.1*, included the following elements used in this project:

- 1xChassis robot 2WD (2 wheels, 2 DC bidirectional motors, 2 encoder wheels, 1 chassis)
- USB cable required to compile into the board
- Motor driver L298N
- Ultrasonic sensor HC-SR04
- Color cables
- Battery holder for 4 AA batteries

Figure II.1: Photo of the Kit Robot LRE-EO2, taken from www.leantec.es. We did not buy the Arduino board shown in the image.

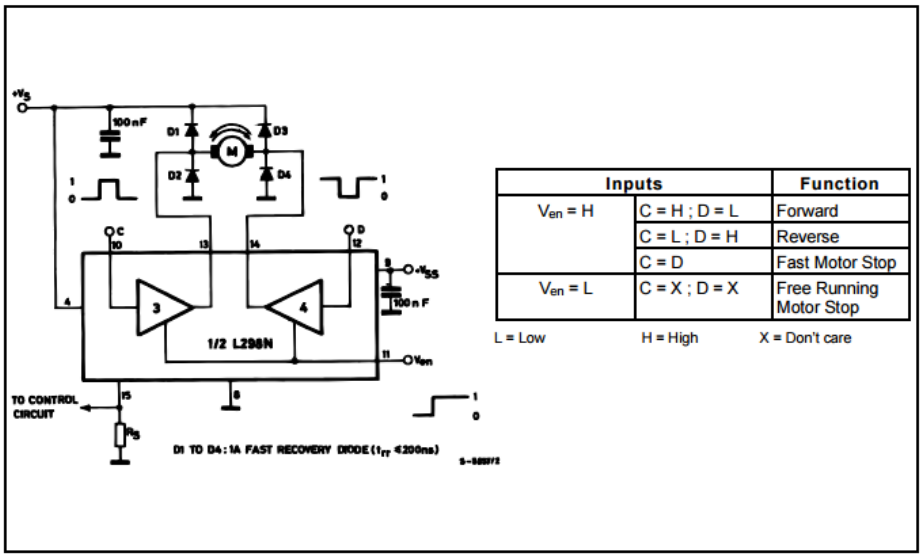
We will now explain how we have assembled some of these hardware parts and a few more that were not included in the pack.

DC motors

We were not able to find much technical information about the DC bidirectional motors. So we ran some tests in order to know the motor's response to adapt the model for the simulations and control design. For more information about the motor's characterization go to (*IV.iii*).

Driver

The two DC motors of the wheels are not connected directly to the board; they are connected to a L298N driver. The driver has six connections: one to ground, another to a high voltage (5V in our case), and the others to four control signals that regulate the speed of each wheel (2 signals for each wheel).



Inputs		Function
V_{en} = H	C = H ; D = L	Forward
	C = L ; D = H	Reverse
	C = D	Fast Motor Stop
V_{en} = L	C = X ; D = X	Free Running Motor Stop

L = Low H = High X = Don't care

Figure II.2: Schematic of the control of one of the two bidirectional DC motors. The two control signals go to ports C and D, and V_{en} is always at 5V. It can be found in the L298N driver datasheet online.

In the previous image the circuit to one of the two outputs of the driver is shown. This output will be directly connected to one of the wheels. In order to have the highest dynamic range, control signals C and D (the two signals that control one of the driver outputs) must operate in a complementary way.

We get the fastest forward speed of a motor when signal C is always high and signal D is always low. If we define Δ as the duty cycle of signal C, then the aforementioned situation happens when Δ=100%. Now if Δ=0%, that means that signal C is always low, while signal D is always high, we get the fastest backwards speed of the wheel. The wheel will be stopped if Δ=50%. The variable Δ will be the one we use to control the speed of the wheels, one for each wheel.

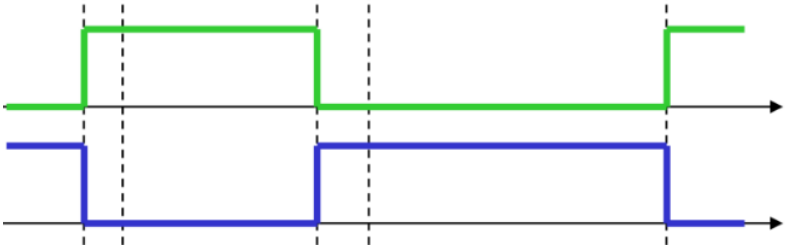


Figure II.3: Let the upper (green) signal be C, and the lower (blue) D. In this case Δ is roughly 40%, the wheel attached to the DC motor would then run really slowly and backwards.

Encoders and photointerrupters:

Two encoder circuits are implemented to know the speed of the wheels. Each one of these encoders will send one signal with the vehicle's wheel speed to the STM board. Therefore, each encoder is attached to each one of the vehicle's wheel's axis and both of them will turn at the same rotational speed. A RPI-574 phototinterrupter is used to generate the signal from the encoder wheel. Their setup in the vehicle can be seen in the following images.

Figure II.4: The image on the left is the encoder wheel, and the one of the right is a photo of how it has been implemented in the vehicle. The photointerrupter RPI-574 can be seen above the encoder wheel in the right photography.

Line sensor:

We used the LRE-F22 sensor to know not only if the vehicle is in the line or not, but also measure how far away it is.

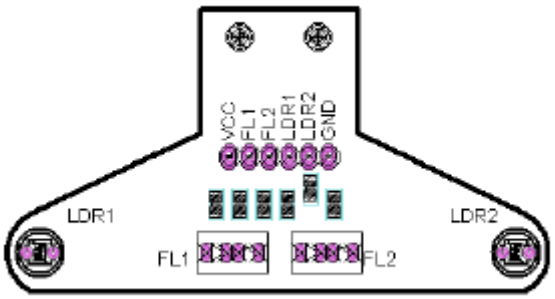


Figure II.5: Scheme of the LRE-F22 sensor with all its pins. Taken from its datasheet available at leantec.es

This sensor has two infrared emitters and two infrared receivers (FL1 and FL2 in the image above) that detect how much of the infrared emission comes back to the sensor. If the sensor FLX receives all the infrared light back (we are in a reflecting surface, outside of the line) then its pin will be at VCC.

There are then two output signals from this sensor: V_FL1 and V_FL2, that go from 0[V] to "VCC" [V] depending on how black (or white) is the surface below the two infrared receivers.

Ultrasonic sensor:

The HC-SR04 sensor does not affect directly the controller of the vehicle, its purpose is to detect any obstacles in front of the vehicle to make it stop in case it gets really close.

Figure II.6: Photo of the HC-SR04 ultrasonic sensor. It has four different pins: VCC requires a supply voltage of 5V, "GND" is the ground, "Trig" will receive a precise trigger square signal and "Echo" will be the sensor's output.

The following figure shows a schematic time diagram of the sensor's performance. The sensor receives a trigger input signal with a 10us pulse from the microprocessor. Then the sensor emits 8 ultrasonic pulses and as soon as the last pulse is emitted the echo pin goes high until the echo pulse is received or the trigger sends the next pulse.

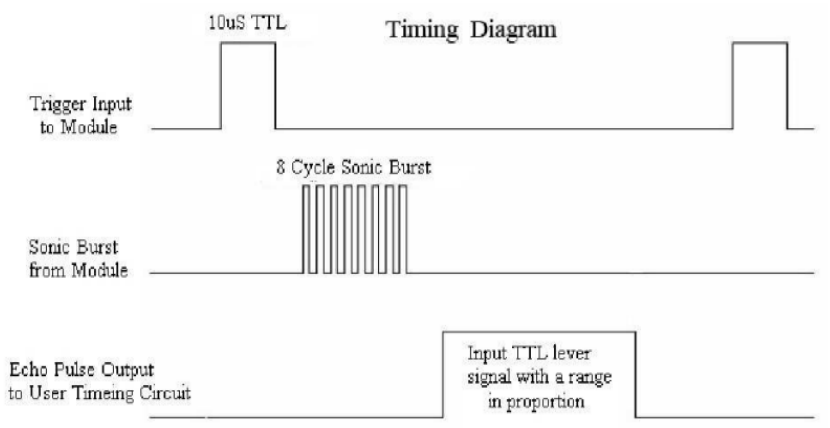


Figure II.7: General Timing Diagram of the sensor's behavior. Extracted from its datasheet.

If the echo is received, the period of time when the echo pulse output is at high level determines the distance between the obstacle and the sensor:

$$d[cm] = t\ [\mu s] * 0{,}01715$$

being *d* is the distance between the sensor and the object, and *t* is the time the echo signal is high.

2.2 Programming environment, used peripherals: STM32F4-Discovery board

Eclipse (https://eclipse.org/) has been chosen as software for the development of the project. We used the OpenOCD debugging Eclipse plug-in to be able to debug our codes (http://gnuarmeclipse.github.io/openocd/).

The control and managing of the mobile robot requires a high performance microcontroller. In this project we select the STM32F4-Discovery kit with STM32F407VG. Some of the key features that made us decide on it are:

- Peripherals such as: Timers, PWM channels, Analog to Digital Converter (ADC), Digital to Analog Converter (DAC)
- Integrated Floating Point Unit (FPU) that allows fast hardware calculations (sums, subtractions, multiplications, divisions…)
- Its low cost and easy access
- Microcontroller based on ARM architecture
- Many freeware tools available online to program it.

We also used the software *STM32 Cube,* and the *Hal Libraries* as freeware. This program generates C projects for our board, but easily programming all the peripherals we want beforehand. It is by no means necessary, but it saved us time programming the peripherals. The *Hal Libraries* have a similar purpose; they make easier the use of peripherals.

For more information on the board, *STM32 Cube* and *HAL libraries*: visit the official website www.st.com.

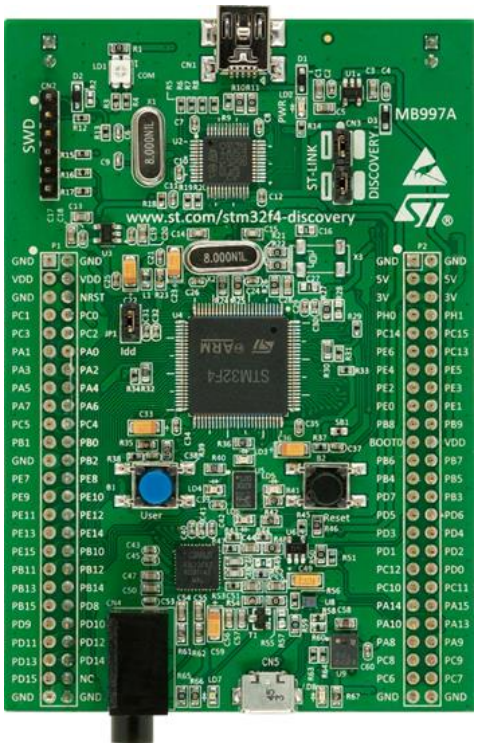


Figure II.8: The STM32F4Discovery board, with the STM32F407VG microcontroller on the center.

From all the peripherals and tools the board offers, we used the following ones on our project:

- 1 Advanced-control timer (TIM1) for the generation of the 2 complementary signals (4 in total) for the DC motors, and to generate the main interruption.
- 2 General-purpose timers (TIM3 and TIM4) in Input Capture direct mode to read the encoder's frequencies.
- Another 2 General-purpose timers (TIM10 and TIM 12) one to generate the trigger signal for the ultrasonic sensor, and the other one to read its response.
- 1 Analog to Digital Converter (ADC1) with 2 input channels to read the infrared line sensor. In order to improve the acquisition time of the data, such acquisitions have been made through a Direct Memory Access (DMA).

All the following data on the peripherals has been extracted from the "*Manual of STM32f4 peripherals*" found in the board's official website (stated above).

Advanced-control Timer (TIM1)

This timer sends the complimentary signals to the driver that allow the wheel's motors to work. We use 4 of its channels: Channel 1, channel 1N (1's complimentary), channel 2 and channel 2N (same as channel 1 and 1N). The channels 1 and 2 are completely independent, whereas the "N" channels depend on their complimentary channels like:

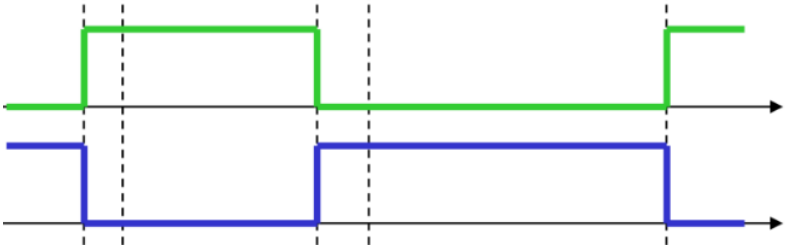


Figure II.9: If the green signal is the output signal of Channel 1, then the blue one would be the output signal of Channel 1N.While the Channel 1 signal is high, the Channel1N signal is low and vice versa.

This timer also is responsible of triggering an interruptive routine. All the control calculations and the collection of the sensor data will be in this interruptive routine. The sample time will be decided by this timer's frequency.

The chosen frequency for this timer is 20kHz, because that is the fastest frequency that the Driver allowed and we fix the sample time as low as possible. This will be the frequency of the complementary signals and the frequency at which the interruption triggers.

General-purpose timers (TIM3, TIM4, TIM10 and TIM12)

These four timers can divide in two groups: TIM3, TIM4 and TIM12 in different "input" modes and TIM10 in "single PWM Generation mode".

The timer 10 in single PWM Generation mode is always generating a 1,315kHz signal with a 1,32% Duty in order to obtain a 10µs trigger for the HC-SR04 sensor and be able to read when the robot is at about 8cm of an obstacle. (For more information refer to the chapter (*IV.i)*, and the ultrasound sensor part).

The input mode function used in the other 3 timers allows them to measure the frequency of the signal in each of the timer's input pin. Timers 3 and 4 are equally configured in order to read each encoder equally, but TIM12's configuration is a little bit different to the other two timers.

This timer is in "PWM Input mode", which allows it to also read the duty cycle of the input signal using two channels for the same signal. In this mode: one of the channels is configured to count when the input signal goes from Low to High. The other channel is configured to count when the input signal goes from Low to High, and it finishes counting when it returns to Low. With both of these channels: we get how long the input signal takes to do one full cycle, and how long it is High during that cycle: the frequency and duty cycle.

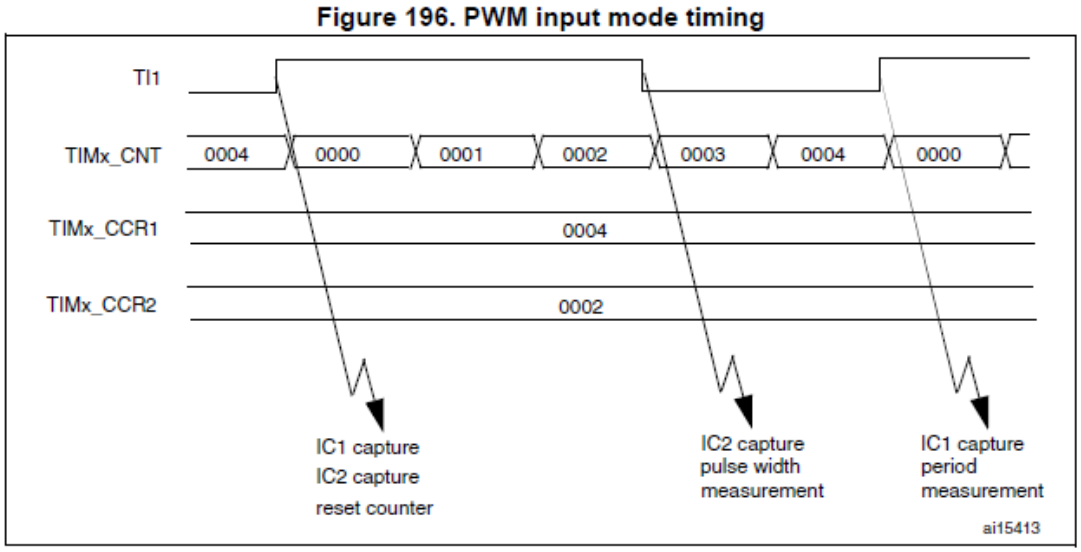


Figure II.10: Illustration on how the PWM input mode works, taken from the "RM0090 Reference manual" from st.com.

In the previous image: TI1 is the input signal, it is Low when it is at 0V and High at 3V. TIMx_CNT are the counts of the TimerX. TIMx_CCR1 is how many counts the timer has counted between IC1 measurements (Low to High). TIMx_CCR2 is how many counts the timer has counted between IC2 measurements (Low to High, and High to Low)

For a more detailed explanation of each Timer mode: refer to the Manual of STM32f4 peripherals, from www.st.com.

ADC:

Two low pass filters are implemented, before the ADC pins, to avoid any possible short-circuit effects from the ADC pins, see *Figure IV.11*. They are simple 1^{st} order analogic filters with $\mathrm{R} = 100\Omega$ and $\mathrm{C} = 47\mathrm{nF}$.

The ADC peripheral is in charge of reading the voltage of the two pins of the line sensor. Its two input signals are the two signals that come from the line sensor. Thanks to these two voltages we are able to calculate the distance *d* from the line center to the vehicle's measurement point as follows: the two input voltages from the line sensor are subtracted to get the voltage difference: "ΔV". Then using the Line sensor model characterized in the section (*II.iii*) we determine the distance *d.*

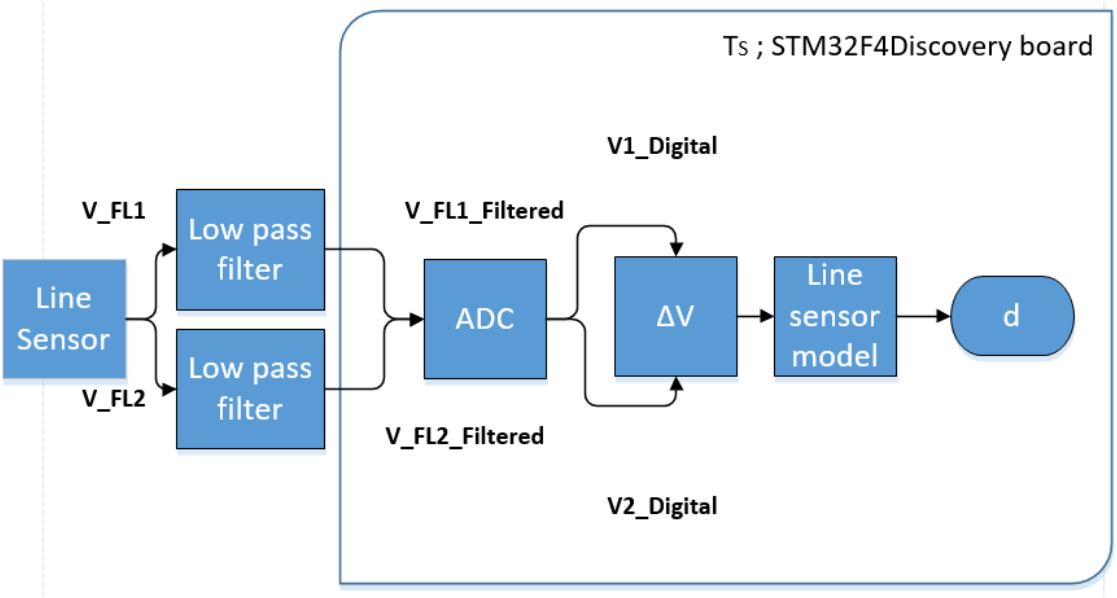


Figure II.11: Block diagram of the ADC implementation.

For more information on the "*Line sensor model"* head to the section (*II.iii)*.

2.3 Characterization and implementation of the hardware

Regulator circuit:

The regulator circuit provides a 3V supply voltage to the line sensor. We use the LM317T device from www.st.com to get the 3V voltage from the 5V of the batteries of the vehicle.

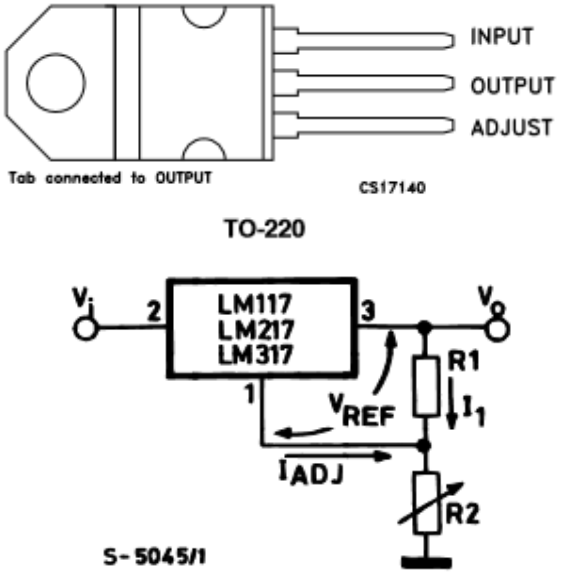


Figure II.12: Schematic of the LM317T device (up) and the schematic of the implementation circuit proposed by its datasheet (down). "Vi" refers to the Input pin at 5V, "Vo" to the output pin at 3V, and the port 1 to the "Adjust" pin.

We follow the circuit recommended by the picture above. In order to decide on a R1 and R2, the datasheet also gives us the equation:

$$V_O = V_{REF}\left(1 + \frac{R_2}{R_1}\right) + I_{ADJ}R_2$$

Knowing that V_{REF} is of 1,25V (given by the datasheet), and that I_{ADJ} can be up to 100µA; getting a small enough R_2 would allow us to neglect the effects of I_{ADJ}. Therefore, we need a relation of: R_2/R_1 = 1,4, so we finally set R_1= 1,5kΩ and R_2=2,2kΩ (R_2/R_1 = 1,47), resulting in Vo=3,2V in the board (verified with the laboratory's multimeter).

Encoder implementation circuit:

We use two photointerrupters RPI-574, a Digital Buffer SN74HC4N, various resistances and capacitors to measure the speed of each wheel. One of photointerrupters generates a signal, whose frequency is directly related to the rotational speed of the vehicle wheel. This signal is then treated: passing through a 1st order low-pass filter and a Digital Buffer. It will finally be captured by the DSP board where one timer (either TIM3 or TIM4 depending on which wheel it is) will get its frequency to the control system.

We got the photointerrupter circuit by following the suggested circuit by the its datasheet, and adjusting the resistances in order to maximize the precision.

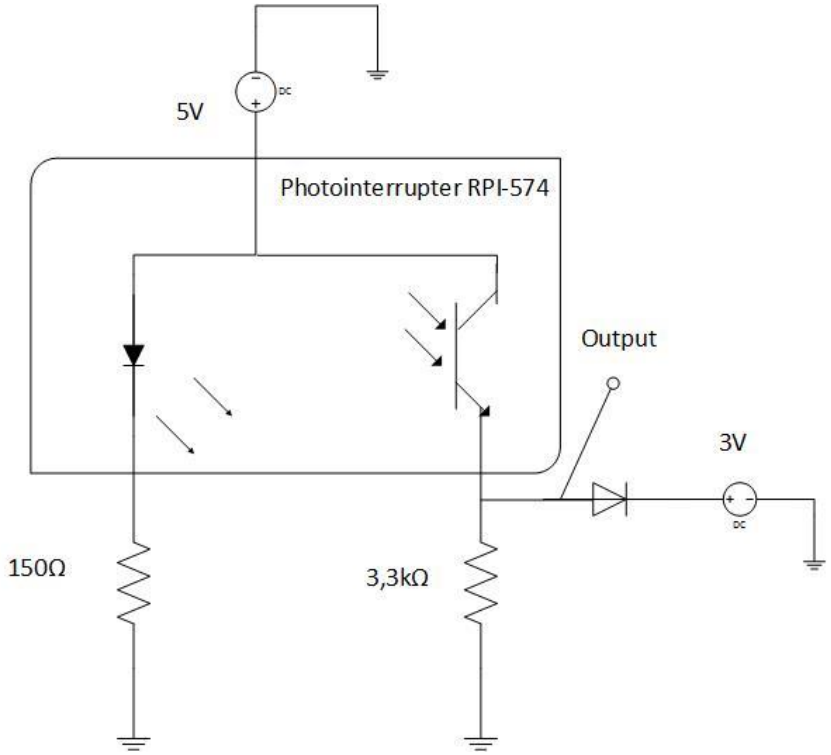


Figure II.13: Schematic of the "Photointerrupter circuit".

The Output is where the circuit will be linked to the "Encoder Implementation Circuit", see figure *II.15*. The diode is a security measure to ensure that there is no voltage higher than 3V that could harm the board. Since the signal waveform that the "Photointerrupter Circuit" gave was too different from a square signal, thus the DSP could not read it correctly, a digital buffer and one low-pass filter were implemented in the "*Encoder Implementation Circuit".* The low pass filters parameters are of 1kΩ resistances and 100nF capacitors.

The whole process of measuring one of the wheel's frequency is illustrated in the following image. Where the "Photointerrupter circuit" block is the one in figure *II.13*.

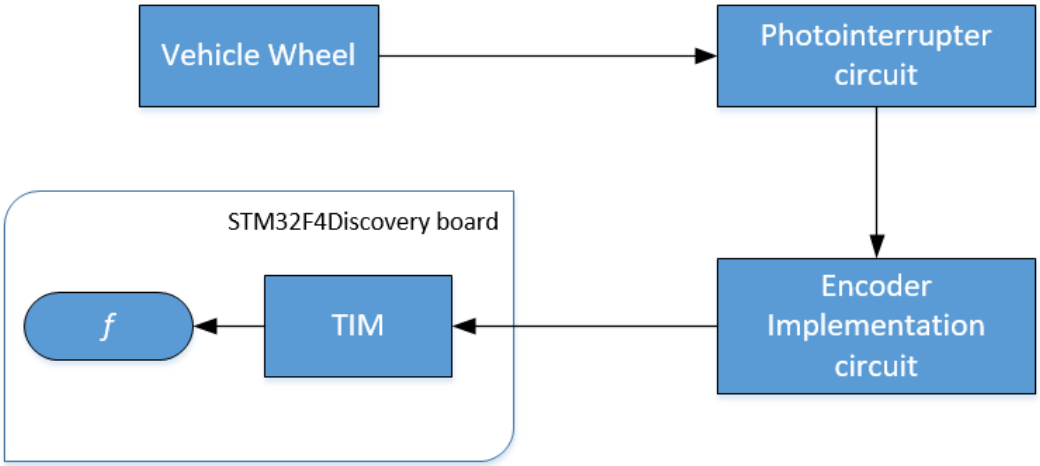


Figure II.14 – Block diagram to illustrate the process of measuring the frequency "f" of one of the vehicle's wheels, from the moving vehicle wheel.

In the following figure we can see an electric schematic of the process stated in figure *II.14.*

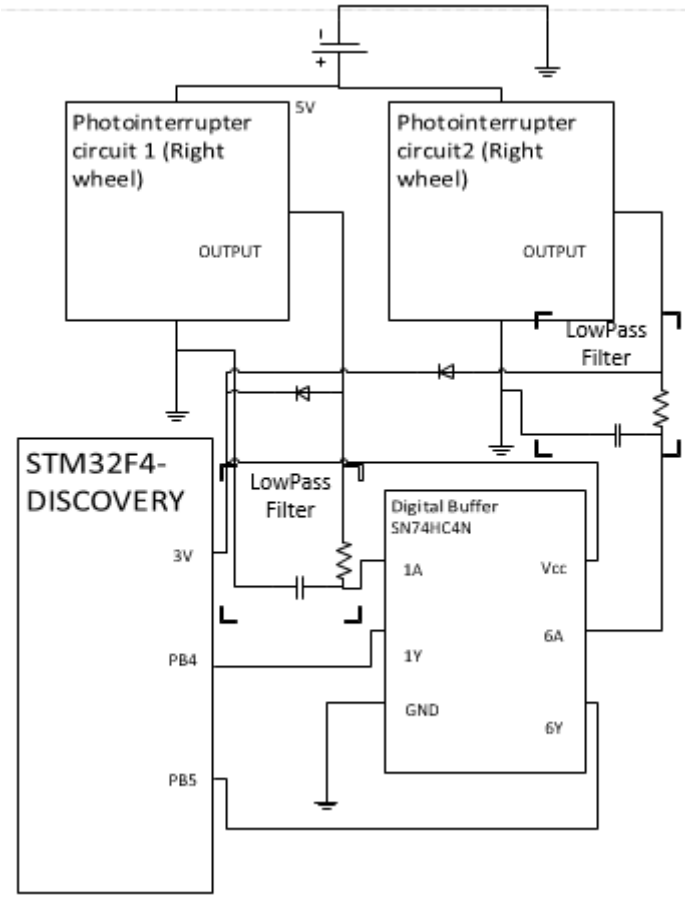


Figure II.15: Electric schematics of the Encoder Implementation circuit, from the Photointerrupter Circuit to the DSP

For more detailed information of the Digital Buffer SN74HC4N, refer to its datasheet.

Line sensor model:

We use the Analog to Digital Converter (ADC) of the board to convert the two signals from the line sensor to the digital realm, operate with their values, and finally get the distance *d* from the line to the vehicle's measuring point.

In order to know this distance *d*: a characterization of the Line sensor LRE-F22 model is required.

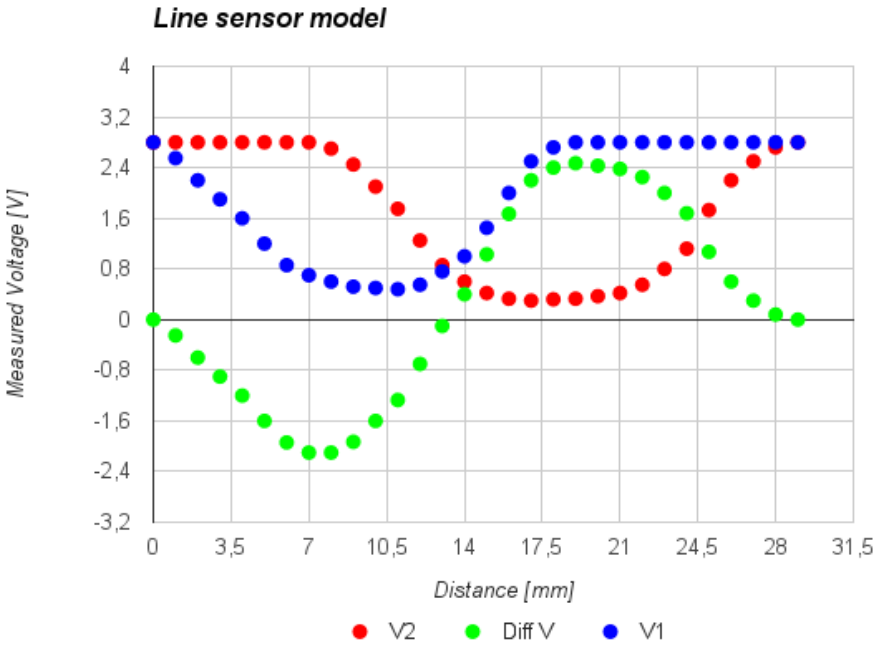


Figure II.16: Characterization experiment of the measured voltages of the two pins of the Line sensor LRE-F22: FL1 and FL2 (V1 and V2 data in the figure) and their difference: Diff V = V1 – V2 = ΔV. The supply voltage for these measurement was 3V.

The *Figure II.16* shows an experimental measure of the two outputs of the line sensors by moving the robot perpendicularly across a fixed straight line. At the beginning of the experiment, where *Distance=0*, the line sensor is above the table and both of its output signals are saturated at their maximum. Then we begin moving really slowly the line sensor towards the line, when the first infrared receiver of the sensor ("FL1" in the figure, whose output voltage is represented by the blue dots) starts to get close to the line, its output voltage goes towards 0V. When the line sensor has been moved 14mm, both of the receivers are perfectly above the line, we want the vehicle to go towards this point, hence here we will define *d = 0.* If the line sensor keeps moving past this point, the infrared receiver FL1 starts to get near the end of the line: which makes the tension of V1 go towards 3V again. The same process applies to the second infrared receiver FL2 (its output is represented by the red dotes in the figure), but only 7mm after.

We need a function: $d = f(\Delta V)$ to know the distance *d* from the measure point in the vehicle to middle of the line. This function can only be zero once, if not the vehicle could converge to the other zeros when $d \to 0$. In order to have such function, the range of values of *d* will be

the relative maximums of ΔV: from ΔV=-2V to ΔV=2,4V. These values of ΔV correspond to $d \in [-8\,;8]mm$ approximately.

There is a small difference in the two Line sensors pins, which results in having more dynamic range of *d* in one direction than the other.

From rearranging the data from the previous test: we find the function $\mathrm{d}(\Delta \mathrm{V})$, in *Figure II.17* ,which now is linear and can be used to calculate *d*. The polynomial function that will be used in the Line Sensor Model is also shown in the figure legend.

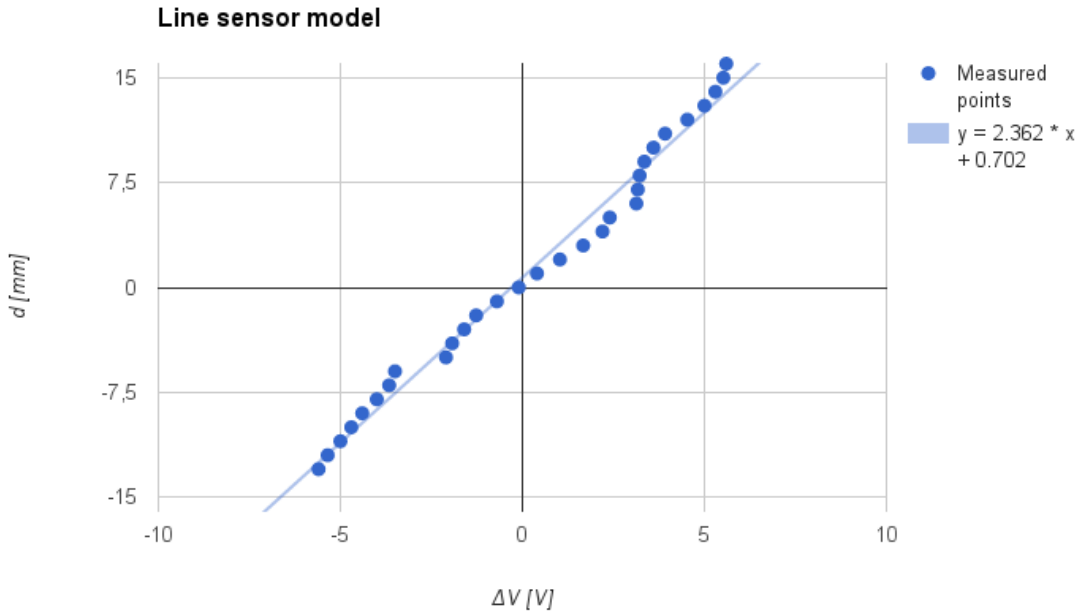


Figure II.17: The "Measured points" refer to the "Diff V" points in the Figure II.16, reorganized in order to linearize it.

We lost some dynamic range: now $d \in [-8\,;8]mm$, in order for the line sensor to measure properly the distance *d*. When *d* is beyond this range, the ΔV voltage goes down again, which in our model implies that the vehicle is returning to the line, when in reality it's getting even more far away.

We would recommend looking for another infrared sensor that has a wider dynamic range for future projects.

Duty signal – speed characterization

In order to find the relation between the duty cycle of the control signal and the wheel speed that signal will produce, without any information from the motor's manufacturer, we have to calculate it experimentally.

First we program the DSP to send to the driver two pairs of complementary signals with a given duty cycle: Δ%. Then, we will measure the frequency "*f*" of the signals that come from the encoders of the wheels in the oscilloscope, calculate at which speeds the wheels are going, and then find the relation between the duty signal and the wheel speed.

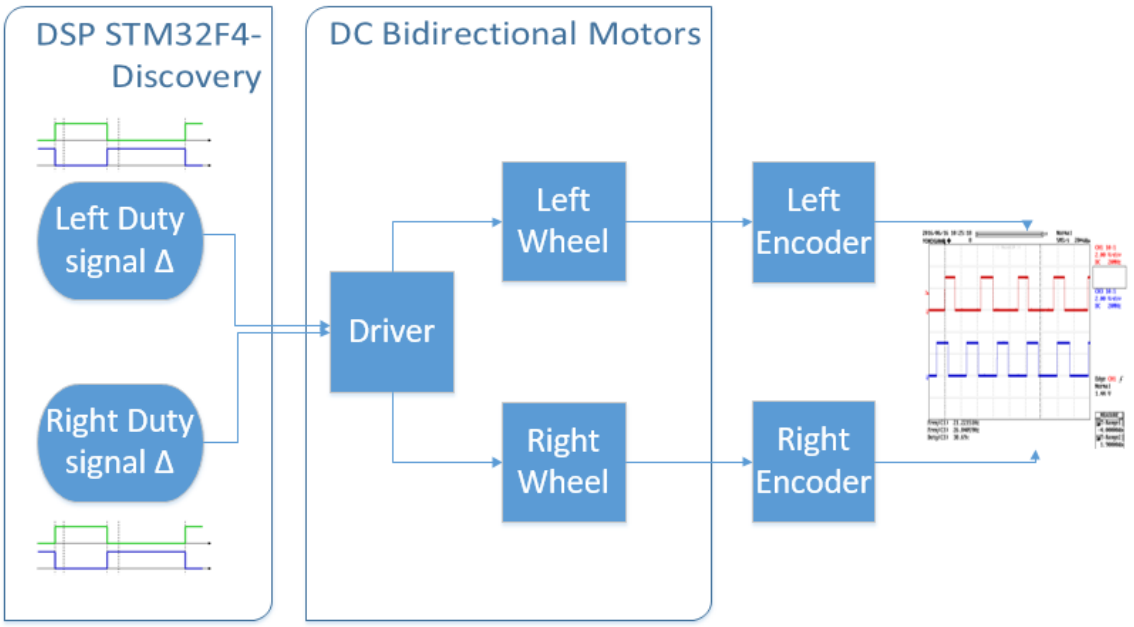


Figure II.18: Block diagram of the experiment to find the relation between a control signal with duty: Δ, and its corresponding wheel speed.

In the following figure: the oscilloscope capture of the test illustrated in *Figure II.18* is shown. The red (upper) signal is the output signal of the Right Wheel Encoder, and the blue (lower) one is the one from the Left Wheel Encoder. For these values, the duty cycle of both of the control signal was Δ=90%, and we measure a frequency "*f*" of 27Hz from the left wheel and 21Hz from the right wheel.

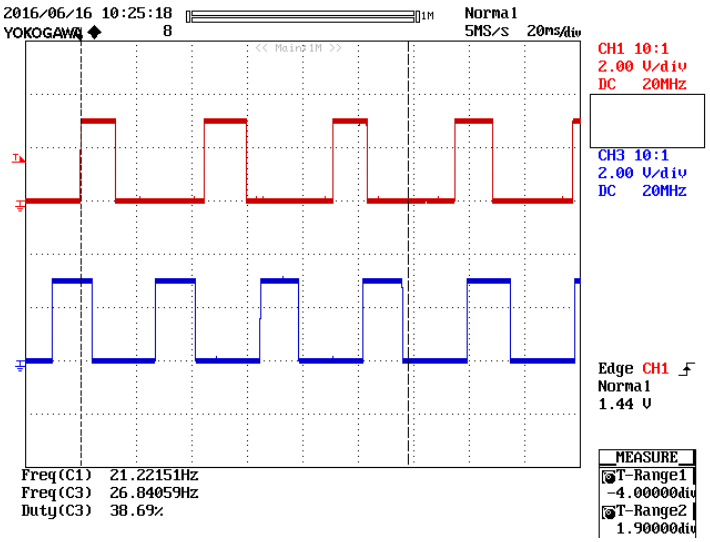


Figure II.19: Screen capture of the oscilloscope, during the test explained in Figure II.18

To find the wheel speed from the frequency of the output signal of each encoder we apply the equation:

$$v = \frac{f}{n} * 2\pi R \qquad \text{(II.1)}$$

Where v is the linear speed of the wheel, f is the frequency from the output signal of the encoder, n is the number of holes in the encoder wheel, and R is the radius of the wheel. In our vehicle: $n = 20$; $R = 6{,}5cm$

During this test, we also realized the existence of a “Dead Zone” of the DC bidirectional motors. The wheel does not move only if the duty cycle of the complementary signals that control it is Δ=50%. In this “Dead Zone” of Δ=[40%; 60%], the friction effects caused by the weight of the whole vehicle make the wheel not move, when it should be moving at least really slowly.

DC bidirectional motor's response

Now that we know the relation between the control signal and the speed of the wheels, we want to measure the motor's response. We will then conduct another test to find an approximation of the motor's response.

We will adjust the duty cycle Δ of the control signal for each wheel. This control signal will go to both wheels, and the encoder will send back to the DSP the frequency at which they turn. A timer (TIM) will receive the output signal of the encoder. Thanks to the equation (II.1), the DSP will calculate the speed of the wheel and send it out of the DSP to the oscilloscope for us to read it, through a Digital to Analogic Channel. Thanks to the oscilloscope we will be able to see an approximation of the motor's response. This test is illustrated in the next figure.

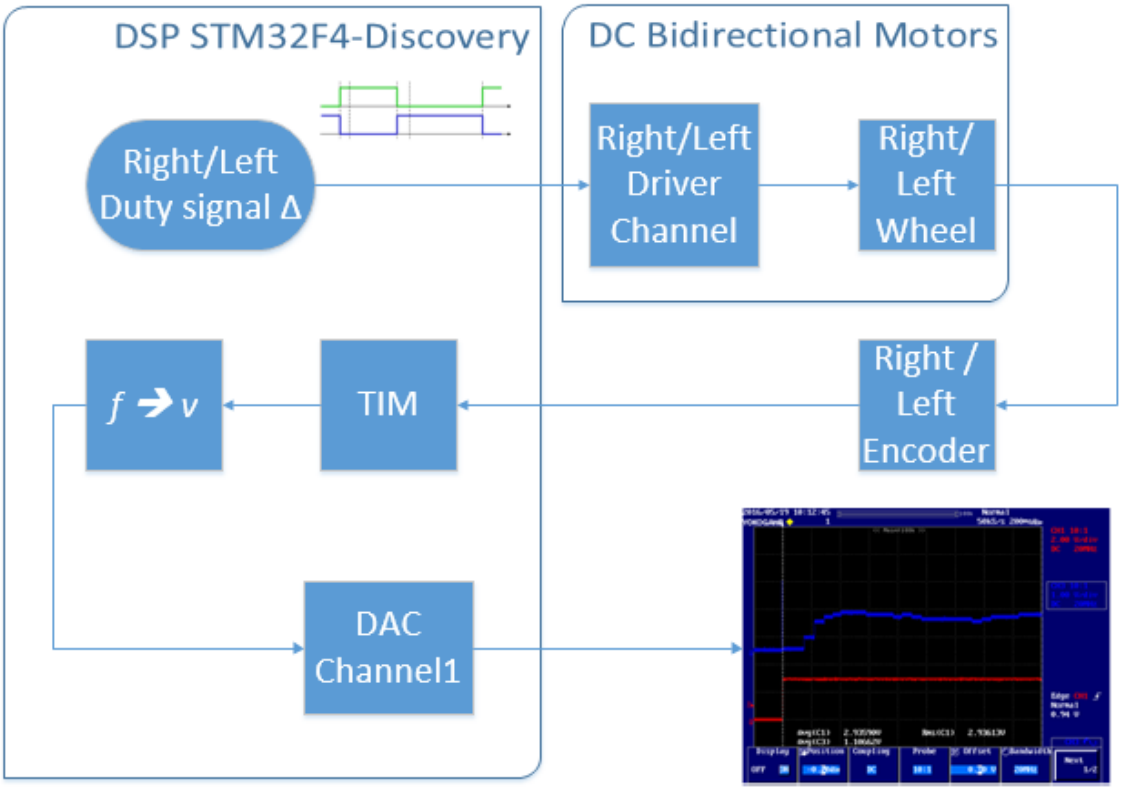


Figure II.20 – Block diagram of the test to characterize the Motor's response.

The signal we see in the oscilloscope has to reach 1.5V in order to get to the speed we want: "v_{nom}".
We will note the time difference (Δt) between: the time where the DSP sends the PWM signal to the motor, and the time where the DAC pin reaches 1.5V.
Test conditions:

- At first the wheels are stopped; they go from 0 [m/s] to v_{nom} [m/s].
- The wheels are on the ground during the test, carrying the whole vehicle: inertia and friction effects are expected.

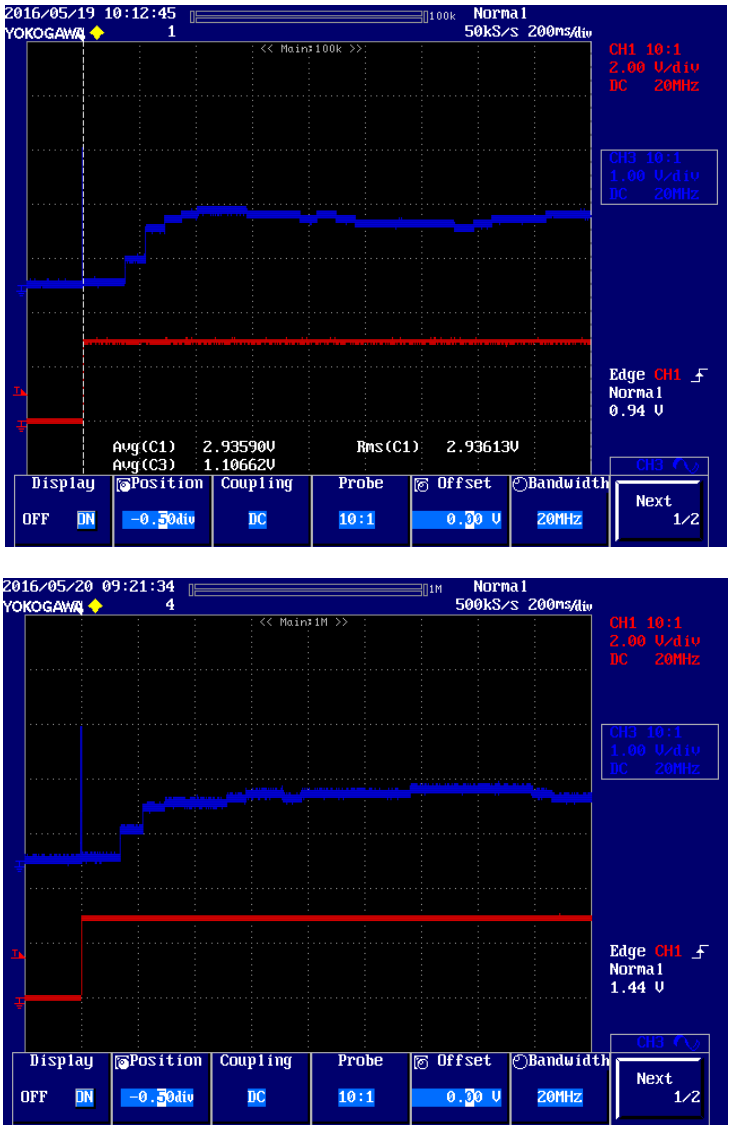


Figure II.21: Test results shown in the oscilloscope.

The blue signal represents the wheel speed (upper image: right wheel, lower image: left wheel). The wheel speed reaches v_{nom} when the blue signal reaches 1,5V. The red signal switches to 5V when the microcontroller sends the control signal to the DC motor. The period of time: Δt is the time between the instant when the control signal is sent by the DSP, and when the blue signal reaches its stationary value of 1,5V.

From these tests we conclude three things: the left wheel seem to go faster than the right one with the same duty signal Δ, the average time (Δt) needed to reach the desired speed is of 350ms approximately for both wheels, and both of them seem to not be able to reach the desired wheel speed (1,5 V).

A controller for the wheel speeds is implemented to ensure that both wheels are going at the speed defined by the control signal.

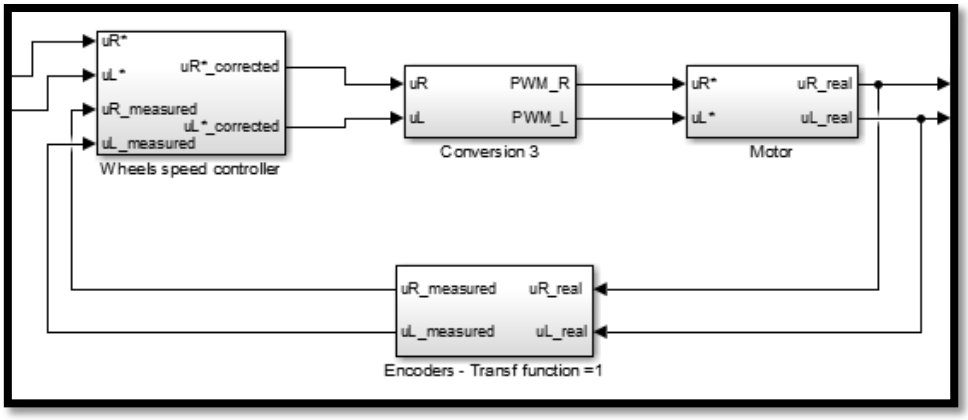
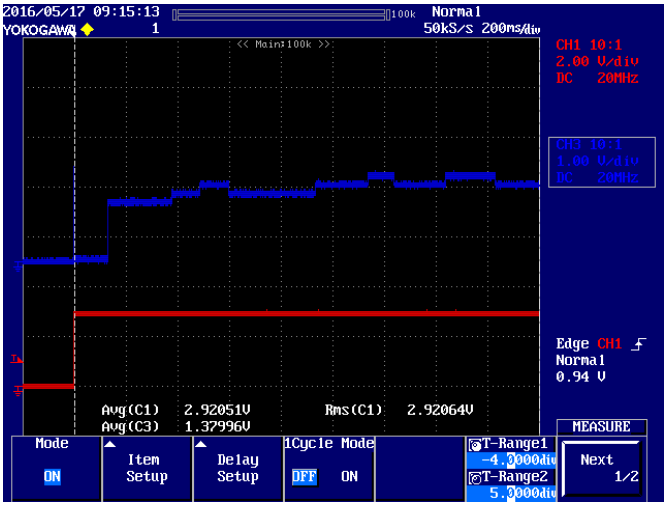


Figure II.22: Block diagram of the inner wheel speed controller. Inside the block "Motor" will be the motor's response.

We assume that the sensor's (encoder) transfer function is 1, in other words: that the encoder reads perfectly the frequencies from the two wheels.

As previously commented in section (*II.i*), we do not know the mechanical parameters of the DC motors, so in order to tune the speed controller we repeat the previous test until the blue signals tend to converge to 1,5V.



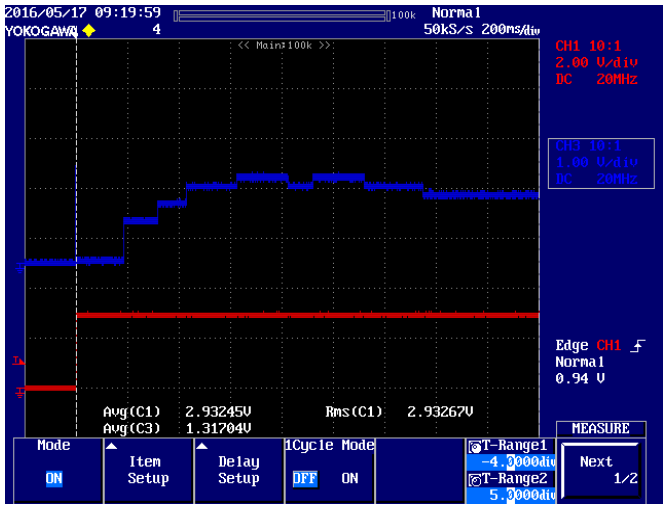


Figure II.23: Screen capture of the oscilloscope at the end of the test. For more information on what is each signal: see the Figure II.21 and the description below. The image of the top is from the right wheel, and the lower one is from the left wheel.

Now that we tuned the wheel speeds controller, we see that both wheels reach the desired speed (blue signal reaches and stays at around 1,5V), and that both have a similar behavior.

From all the tests with the different wheel speeds, the average settling time: Δt that the DSP needs to make a one of the wheels go at a certain speed and stabilize it is:

$$t_{s\ Average} = 300ms$$

We will then model the motor response with a 1st order filter with low gain with a transfer function like:

$$M(p) = \frac{k}{\frac{t_s p}{4} + 1}$$

Where t_s is the settling time found during the test and k is the experimental gain.

These values are purely experimental, so they may get tuned to a certain extent in the simulations but always respecting the order of magnitude. Moreover, this is the average time that one wheel takes to go from 0m/s to a certain *v*. These had to be the test conditions due to our lab conditions, but it is safe to assume that this value of $t_{s\ Average}$ is lower if we do not start with fully stopped wheels.

We plan a little test in order to ensure that the speed controller works properly. Before the speed controller was set up, if we set the same control signal for each wheel: Δ, the vehicle would not go in a straight line. The wheels or the DC motors are not identical. We try again the same test with the speed controller on. Now if we put the same control signal Δ for both wheels, the vehicle goes in a straight line, at the speed corresponding to the duty cycle Δ.

These are good approximations, but we have no way of getting information from the vehicle while it is moving and tracking the line. So we cannot do the tests that would ensure that the speed controller and the model of the motor response work properly when the control signals of the wheels vary.

2.4 General structure of the Robot's firmware and code

In this section we assemble all the circuits and hardware previously introduced in section *II.1* and *II.3* with the Discovery board.

The final vehicle:

The "Line Sensor Implementation circuit", the "Encoder circuit" and the Regulator circuit have already been explained in section *II.3*. There is no need for any implementation circuit for neither the Ultrasonic sensor nor the Driver.

The following figure shows the general schematic electric structure of the vehicle.

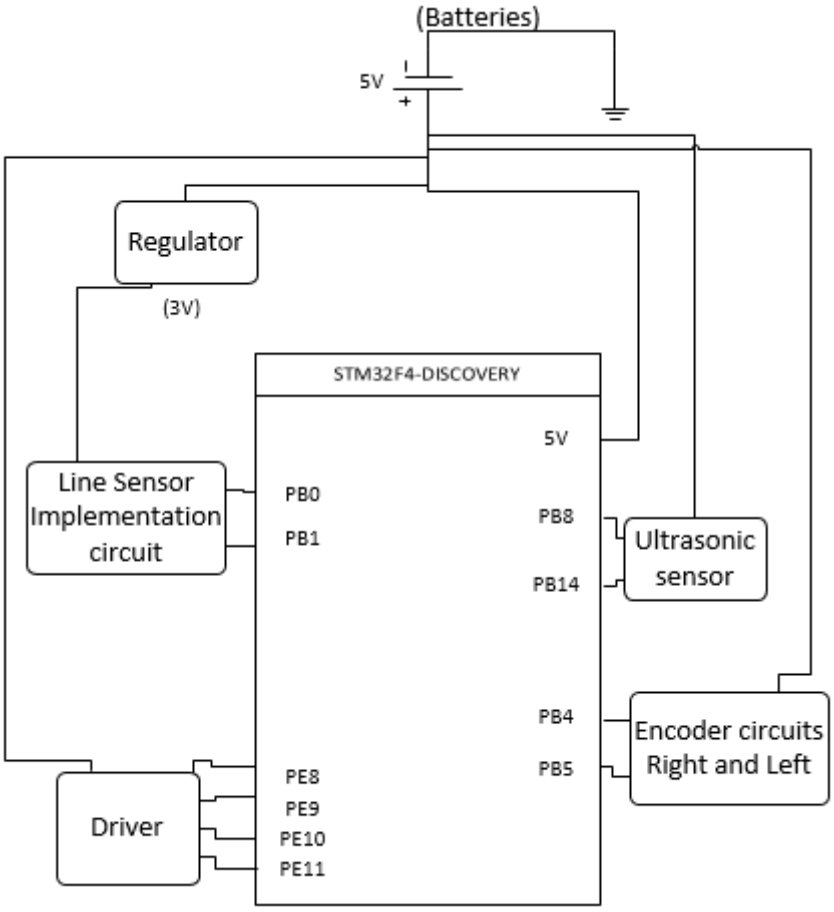


Figure II.24: Electric schematic of the whole vehicle's circuitry.

For more information on the subsystems in the *Figure II.*24: head to their respective section in the chapter *II.iii.* The final vehicle we built is showed in the next pair of images:

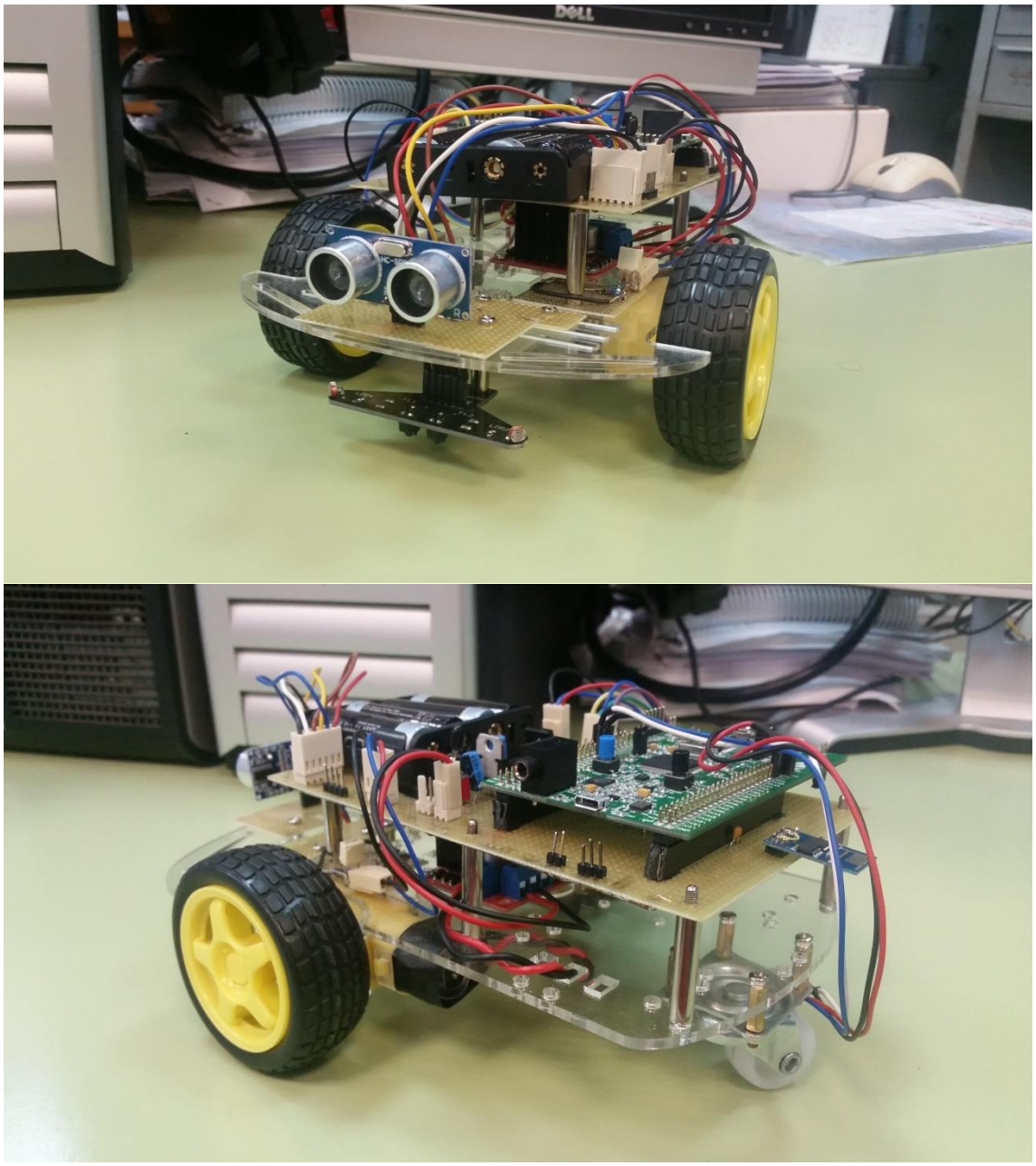


Figure II.25: Final photos of the front and side of the vehicle.

General Structure of the code:

As previously explained in (*III.ii)*: there is an interruptive routine programmed every 50µs (20kHz frequency) where the microcontroller has to do all the controller's calculations and read all the sensor values. This interruptive routine runs every:

$$T_S = 50\mu s$$

Being *Ts* the sample time. In this routine, the DSP has to:

1. Initialize the interruptive routine.
2. Record into in its memory the 2 values from the ADC channels (which come from the line sensor).
3. Convert them to the distance *d*.
4. Calculate the control signal *u2.*
5. From *u2* and a constant *u1* (defined from the beginning as the linear constant speed of the vehicle) calculate each wheel speed references: *uR** (Right) and *uL** (Left).
6. Calculate the real control signals that will be sent to the driver (and then to both wheels) from: the references *uR** and *uL** and the measured wheel speeds: *uR_measured* and *uL_measured.*
7. Reconfigure the timer TIM1 to send the real control signals in the form of 4 square signals (2 complementary signals for each wheel) to the Driver.

This process is illustrated in the following block diagram:

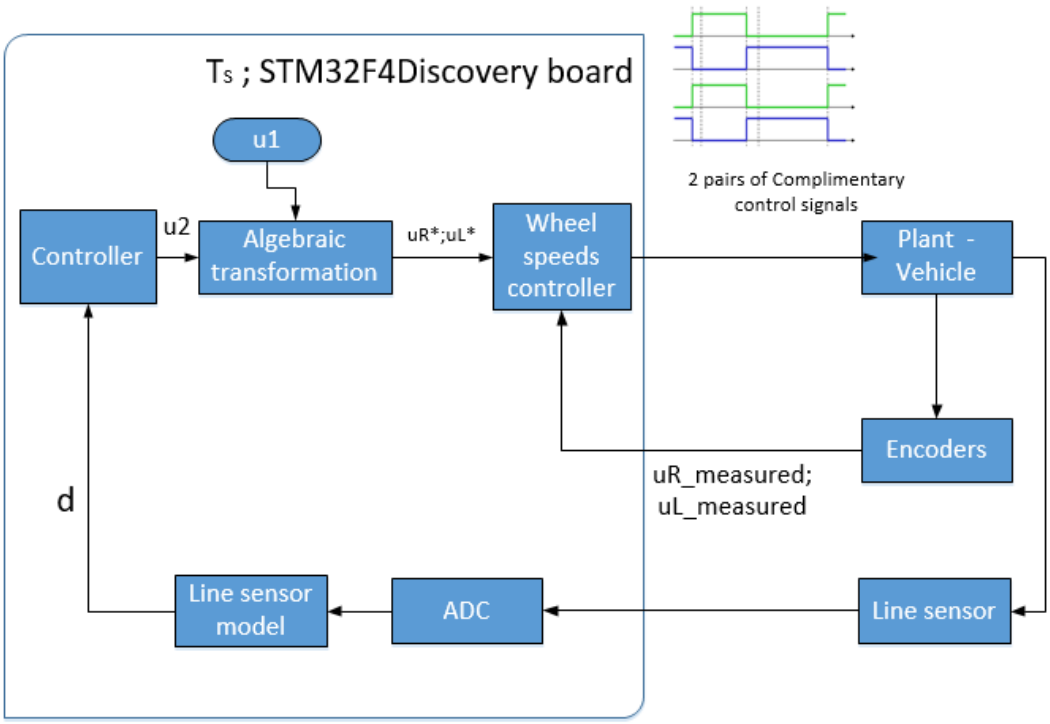


Figure II.26: Block diagram of the code's general code structure. The plant is constantly sending data to the Encoders and the Line Sensor, but the DSP only processes them every Ts=50µs.

We need to make sure that the total code processing time does not go beyond 50µs. If it was any longer: we cannot be certain that the controller is being implemented correctly. The code itself could be divided in two general steps:

1. The time the DSP needs to start the interruptive routine generated by the timer TIM1, and the time to get the values of the two ADC channels and record them into the DSP memory.
2. The time to do all the calculations for the controller, algebraic transformations, duty cycles of the control signals… And to send them to the Driver.

We conduct a test in the laboratory where we program two signals to raise during the execution of each of the two steps above, and to go low when the step is finished. When measuring these two signals in the laboratory's oscilloscope we get the following image:

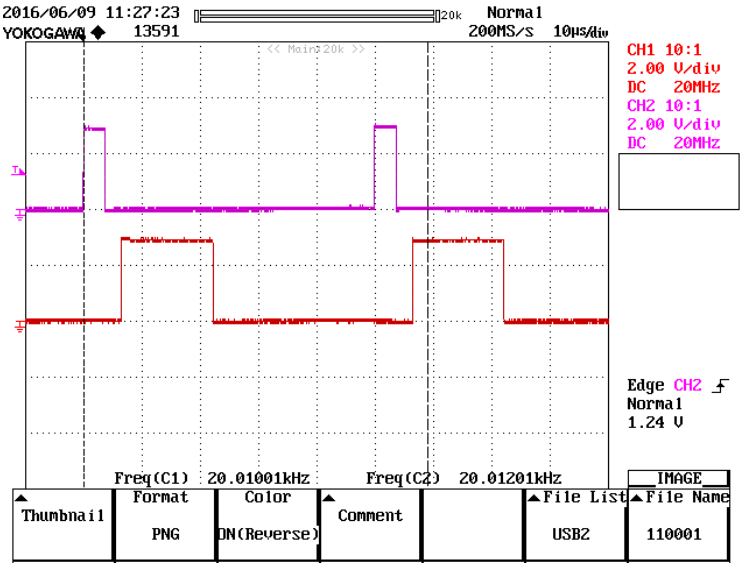


Figure II.27: Screen capture of the laboratory oscilloscope during the processing time test.

The pink signal represents the time the DSP needs to do the steps 1 and 2 listed at the beginning of this section. And the red signal represents the time the DSP needs to do the other 3 to 7 steps. These two signals restart every *Ts*=50µs.

We still have more than 20µs to fill before the code consumes too much of the processing time of the DSP microcontroller.

Chapter 3: Simulation and experimental results.

This chapter will present the results obtained in both the real and the modelled vehicle in three different tests with the aim to test the capacity of adaptation of the robot to different types of trajectory, namely, tracking a straight line, tracking a circle trajectory and tracking a trajectory with a sinusoid pattern. The tests will validate both the linearized model and control previously designed. Furthermore, the implemented robot should follow the trajectory to a certain extent in every test. Conclusions about the overall real performance will be also derived.

Testing the linearized model:

To see how good the linearized model is: we will also change the dynamic equations in the "Vehicle" block from the equations in (*III.4*) to:

$$\dot{d} \cong -\frac{vlc(s)^2}{\sqrt{1-(lc(s))^2}}\tilde{d} - \frac{v}{\sqrt{1-(lc(s))^2}}\widetilde{\theta_e} + \frac{lc(s)}{\sqrt{1-(lc(s))^2}}\widetilde{u_1} - l\widetilde{u_2} \tag{I.10}$$

$$\dot{s} \cong \frac{-c(s)v\,\tilde{d} - lc(s)v\,\widetilde{\theta_e} + \widetilde{u_1}}{\sqrt{1-(lc(s))^2}} + v \tag{I.12}$$

Where:

$$\begin{cases} \tilde{d} = d - d^* = d \\ \widetilde{\theta_e} = \theta_e - \theta_e^* = \theta_e - \operatorname{asin}(-lc(s)) \\ \widetilde{u_1} = u_1 - u_1^* = u_1 - v\sqrt{1-(lc(s))^2} \\ \widetilde{u_2} = u_2 - u_2^* = u_2 - c(s)v \\ \tilde{s} = s - s^* \end{cases} \tag{I.11}$$

Then we will analyze the differences in the evolution of the variable "*d*" between the non-linear model and the linearized one. This linearized model can only work when the vehicle is near the working trajectory (see *I.ii.Working trajectory* for more information). Considering that the vehicle may start at initial conditions far away from this working point: in the simulations we will give some time to the vehicle to get to the working point (using the non-linear dynamic equations) and then we will switch to the linearized ones. This shift will be properly signaled in the simulations.

In all of the following tests:

$$l = 0{,}06[m]\,;\, v = 0{,}3\ [m\,/\,s]$$

The initial conditions of the vehicle will be really close to the trajectory in each test, and the curvature of each trajectory is previously calculated.

We will use a Proportional-Integral type of controller properly tuned for all three simulations, based on the one designed in the section *I.iii*.

3.1 Numerical simulations setup

Simulink is a known mathematical framework to develop numerical simulations of dynamical systems. *Figure III.1* shows the complete Simulink block diagram used to perform the numerical simulations and validate both the model of the vehicle and the controller previously designed.

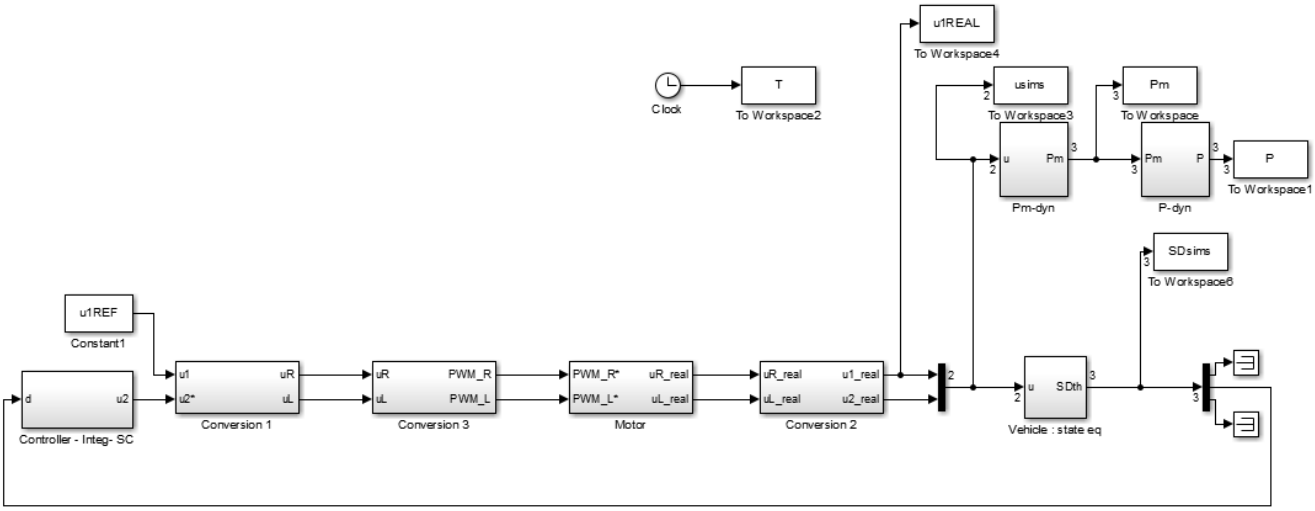


Figure III.1: Simulink block diagram of the complete dynamical system

The block diagram is composed of different subsystems which are explained in the following.

"Conversions" subsystems:

The "conversion" subsystems, presented in *Figure III.2*, of the Simulink block diagram perform the following algebraic transformations of variables:

- The "Conversion 1" block is the variable change from (u_1, u_2) to (u_R, u_L):

$$u_R = r\omega_r, \; u_L = r\omega_l \tag{I.17}$$

  where $\omega_{r/l}$ and r are taken from the equations in (I.2). We get the algebraic transformation to (u_1, u_2) from the same equations, applying (I.17) and isolating (u_R, u_L) we get:

$$\begin{cases} u_R = u_1 + Ru_2 \\ u_L = u_1 - Ru_2 \end{cases}$$

- The "Conversion 2" block directly uses the equations in (I.2) applying the variable change of (I.17).

- The “Conversion 3” block performs the relation between the speed of the vehicle’s wheel in [m/s] and the integer value required to apply the complementary control signals so that the vehicle gets the desired wheel speeds (for more information on the PWM control signals: refer to section *II.ii*). This relation has been found empirically while characterizing the DC bidirectional motors (more information on chapter *II.iii*).

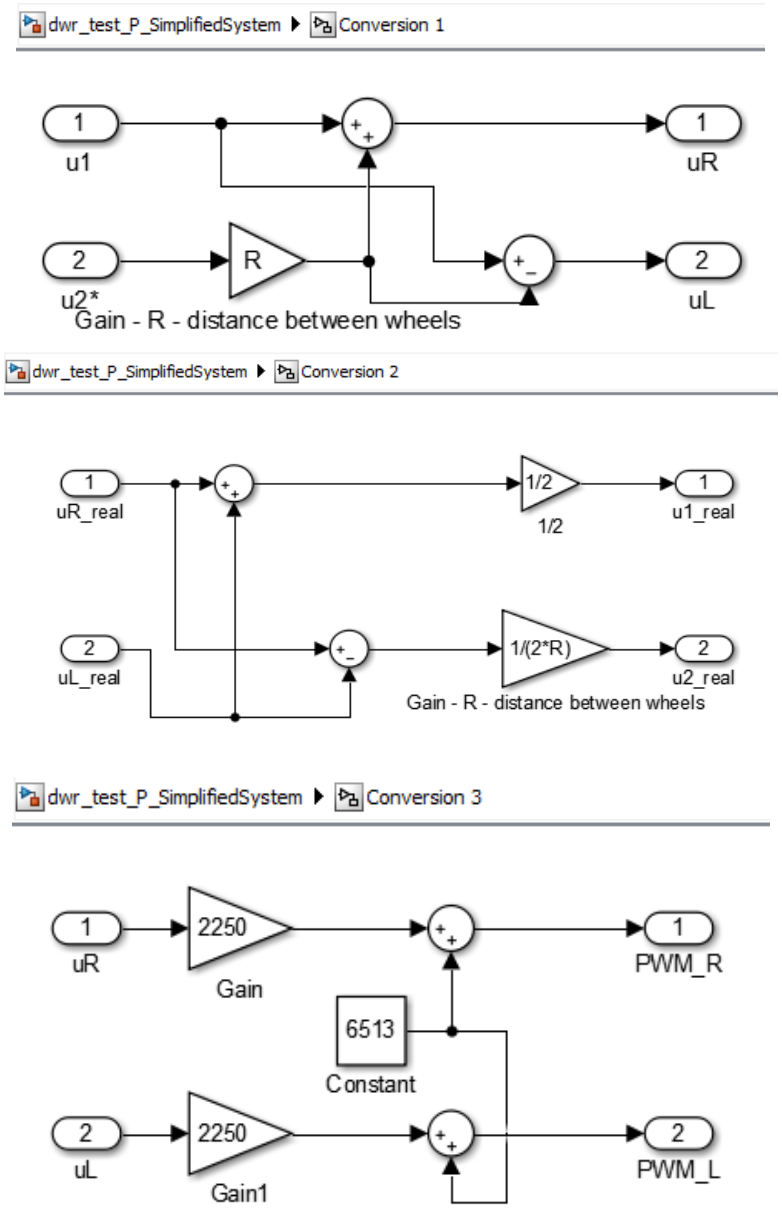


Figure III.2: Block diagrams of the three different conversions used in the project’s numerical simulations.

“Controller” Subsystem:

This is the block where the controller calculates the control signal from the state variable *d.* This subsystem is dependent on the type of control simulated. For example, *Figure III.8* shows the implementation for a Proportional plus Integral controller.

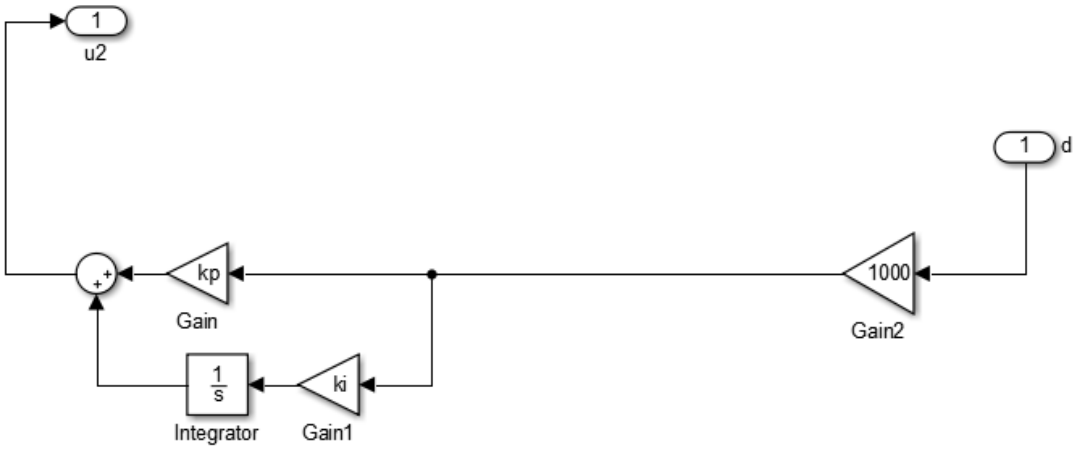


Figure III.3: Block diagram of block "Control" of the Simulink block diagram. In this case it represents a Proportional Integrator controller.

"Motor" Subsystem:

This block implements the transfer function of the motor with a first order filter with a low gain, see *Figure III.iv*. The input is the PWM control signal. The parameters of the filter have been adjusted considering the measured response of the motor presented in section *II.iii*.

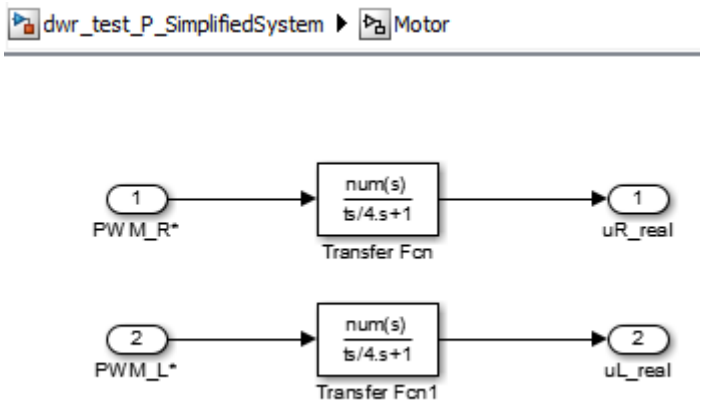


Figure III.4: "Motor" subsystem. The two transfer functions are identical, and represent a 1st order filter.

“Vehicle” Subsystem:

This Simulink subsystem implements the equations III.4 of the dynamical vehicle model presented in the chapter *I*.

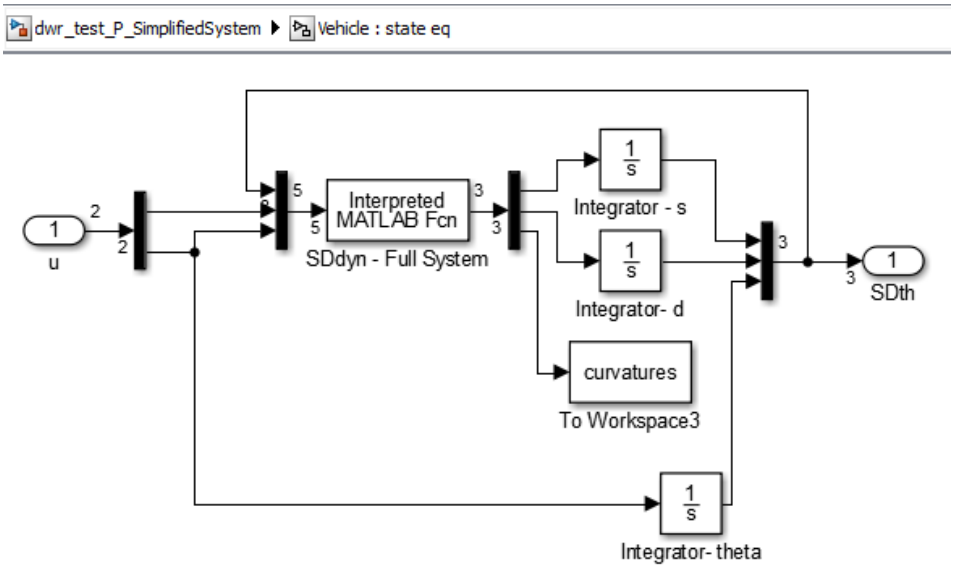


Figure III.5: “Vehicle” Subsystem. All three integrators ($(s; \dot{d}; \theta)$ variables) have predefined Initial Conditions which are calculated before starting the simulation.

In order to be able to calculate $\dot{d}$ and $\dot{s}$: we must know the values of $(\frac{\partial \sigma_x}{\partial s}; \frac{\partial \sigma_y}{\partial s})$ at each value of s. This implies that in order to simulate a trajectory, it has to be defined by a function of the kind:

$$(\sigma_x; \sigma_y) = (f(s); g(s))$$

Where $f(s)$ and $g(s)$ have to be perfectly defined for all *s.* These functions as well as the dynamic equations are included in the “I*nterpreted MATALAB Fcn*” block.

3.2 Test 1: The straight line trajectory

The aim of this first test is to evaluate how the robot responds to a sudden discontinuous change of trajectory.

Simulations results:

The results of the numerical simulations of the straight line trajectory are displayed in the following images. We used a PI controller, based on the results found in the section *I.iii.*

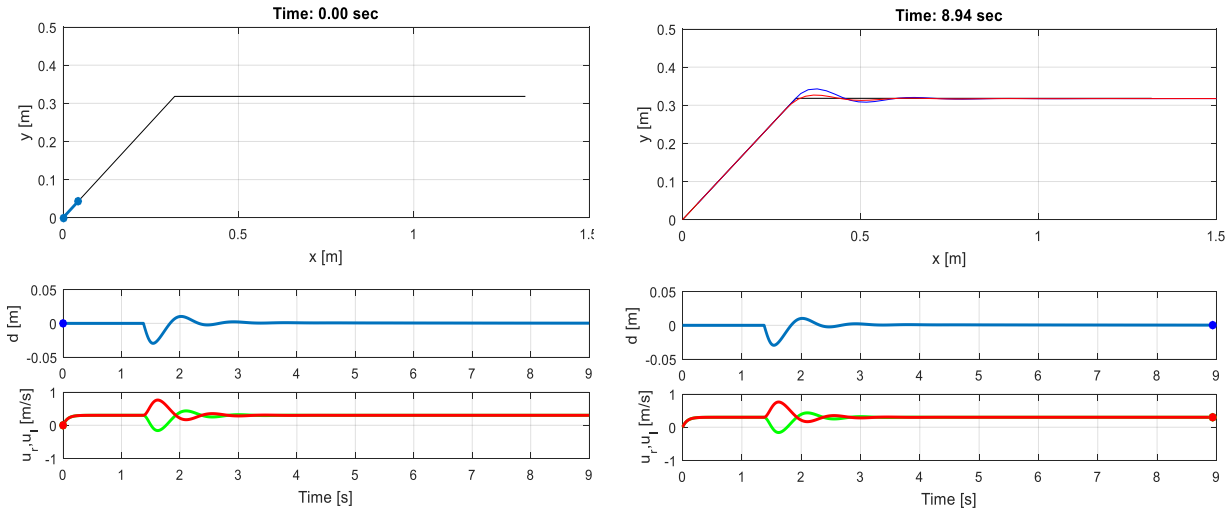


Figure III.6: Screen capture of the beginning (on the left) and end (on the right) of the simulation (with the non-linear dynamic equations).

The black line in the first plot (the upper left one) is the trajectory to follow, in this plot the vehicle has not started moving yet. The red and blue lines are the path where the vehicle has been (red one is the path of the middle of the vehicle's wheel axis, and the blue one is the measuring point path). The 4 small plots are the evolution of the error *d* and the evolution of each linear wheel speed through time.

The vehicle is able to track the line from start to finish. A 2nd order response can be seen in the angle change of the trajectory. This is due to the settling time of the motor response we modeled in section *II.iii*. The theoretical model accordingly predicts this result: when considering a perfect motor, the simulation presents a 1st order response, exactly like the response seen in the theoretical analysis of the PI controller in section *I.iii.*

The following images show the same simulation results if the dynamical model was replaced at some point (marked by the magenta line) during the simulation for the linearized dynamic model.

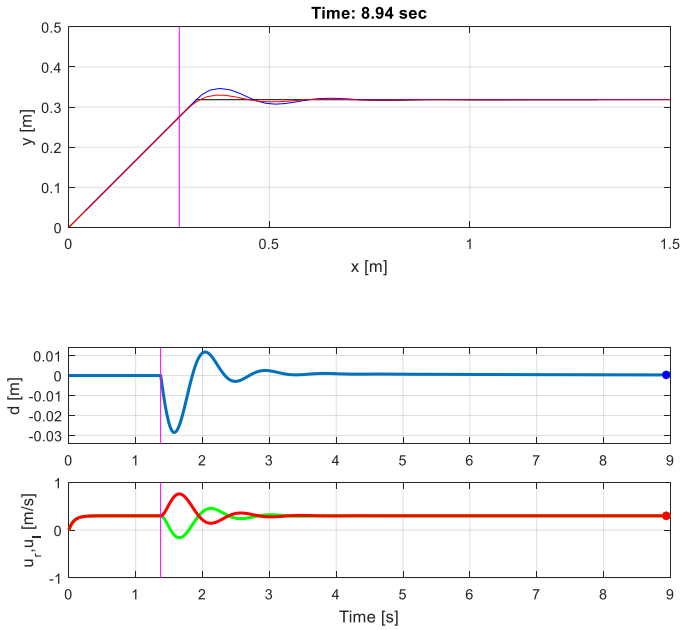


Figure III.7: Simulation of the trajectory Test1 with the linearized dynamic equations from the magenta line, before this line the model uses the same non-linearized dynamics as the previous simulation.

Comparing it to the non-linearized model results, we cannot see any visible differences between the two plots of the state variable *d.* In order to see if there are any differences, we save the vector of errors "*d*" of both simulations and we evaluate their differences:

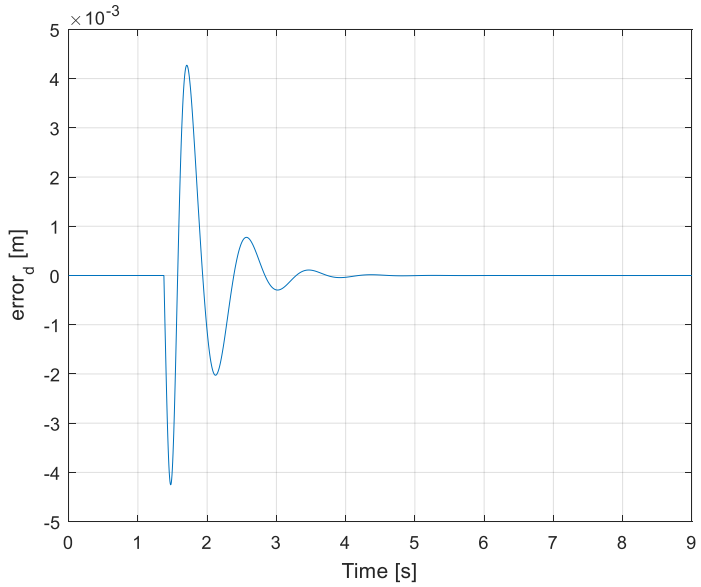


Figure III.8: $error_d$ is the difference between the "d" in the non-linearized dynamic model and the linearized one.

$$error_d = d_{non-linear} - d_{linearized}$$

We can safely say that, in this test at least, the linearized model converges to the non-linear one when the vehicle approaches the working point. And that the system responds accordingly to what we expected in the theoretical analysis.

Experimental results:

When testing the vehicle in the same line trajectory and with the same controller: we do appreciate a 2^{nd} order response like the one in the simulation. The vehicle does not get lost and is able (at least from what we can see) to make the steady state error tend to 0.

It is worth noting that in the simulation: the variable *d* reaches the value *d = 20mm*. In the section (*II.iii. Line sensor model):* we said the vehicle gets lost for values of *d* beyond 8mm. But, in the laboratory the vehicle is able to track the line correctly, which means that *d* does not reach the value 8mm in the real vehicle. In this case the simulation does not correctly represent the real vehicle model.

3.3 Test 2: The circle trajectory

The radius of the circle is: $\mathrm{R} = 0.17[\mathrm{m}]$, hence its curvature: $c(s) = \frac{1}{R} = 5{,}88\ [\frac{1}{m}]$.

The aim of this test is to see if the vehicle converges to $d = 0$ with a constant curvature and a PI controller like it is supposed.

Simulations results:

The results of the numerical simulations of the circle trajectory are displayed in the following images. We used a PI controller, based on the results found in the section *I.iii.*

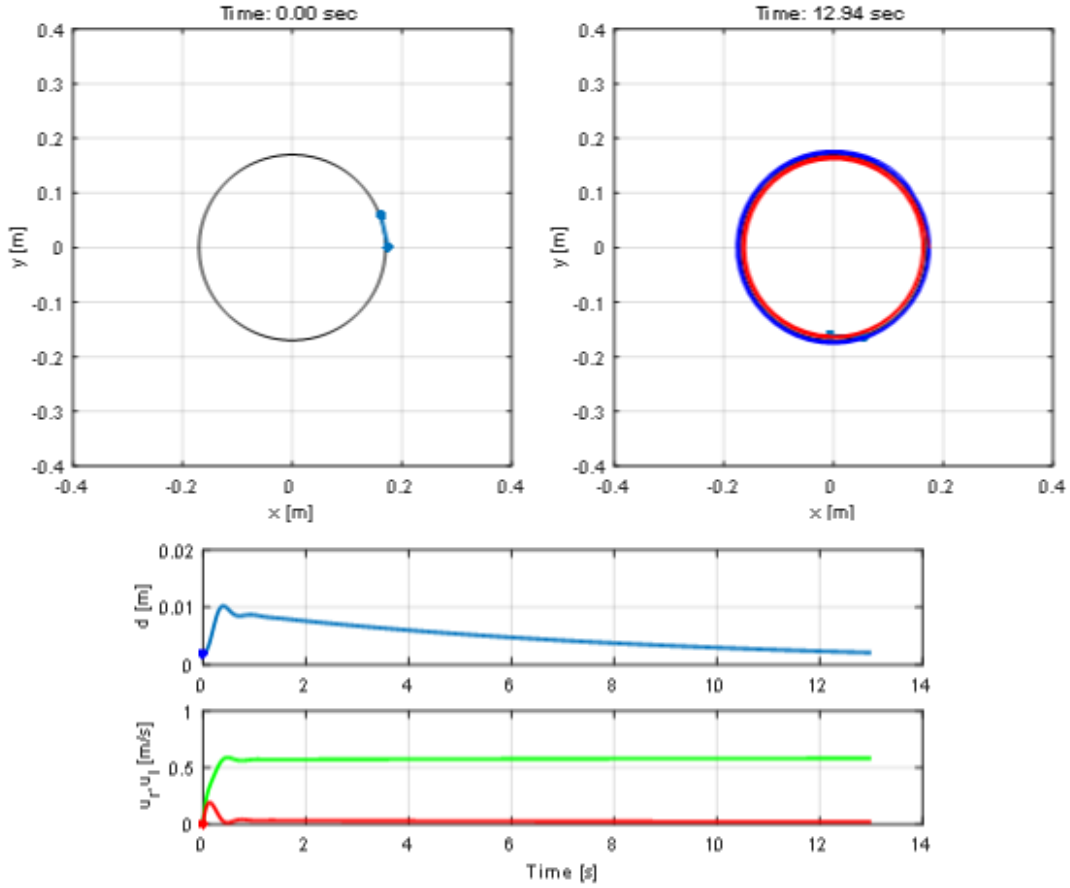


Figure III.9: Screen capture of the beginning (on the left) and the end (on the right) of the simulation (with the non-linear dynamic equations). For more details on the colors and information of the plots: refer to the first figure in (III.ii).

The state variable "*d*" converges to 0 when the curvature is constant. This is what we expected from the Controller Design of the Proportional Integral controller. Now we make sure that the linearized model acts like the full dynamical model in a circular trajectory. The following image shows the same simulation if the dynamical model was replaced at some point (marked by the magenta line) during the simulation for the linearized dynamic model.

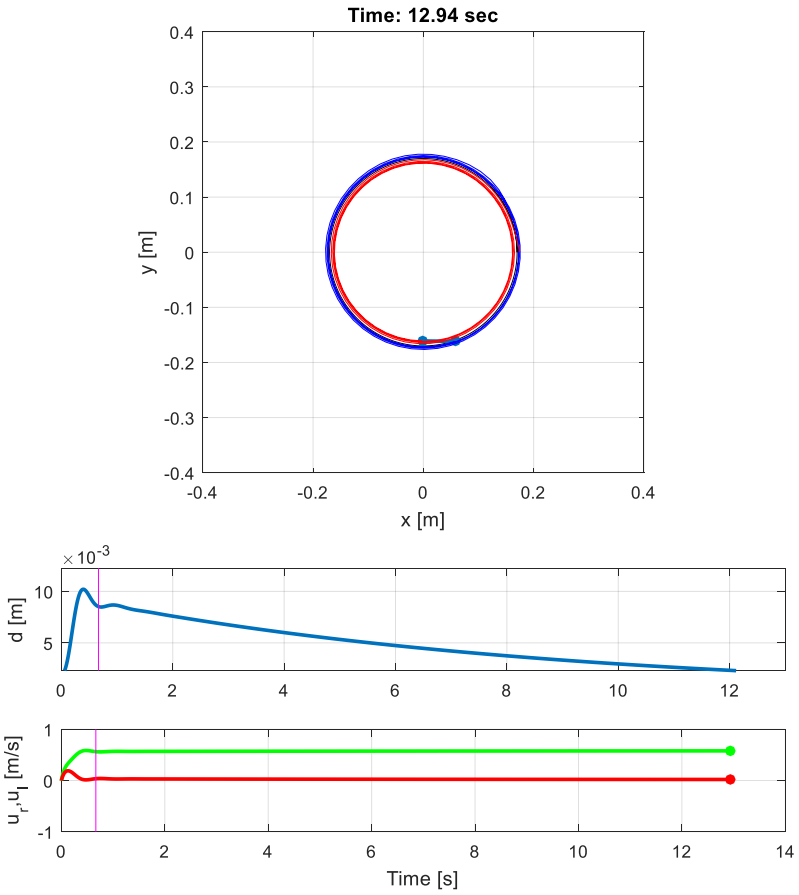


Figure III.10: Simulation of the trajectory Test2 with the linearized dynamic equations from the magenta line.

As we previously had in Test1: it is hard to see any differences between the non-linearized and linearized models with this plots. We will now show in the next figure the difference between the vector of the state variable *d* in the non-linearized dynamic model (*Figure III.9*) and the one simulated with the linearized model (*Figure III.10)*.

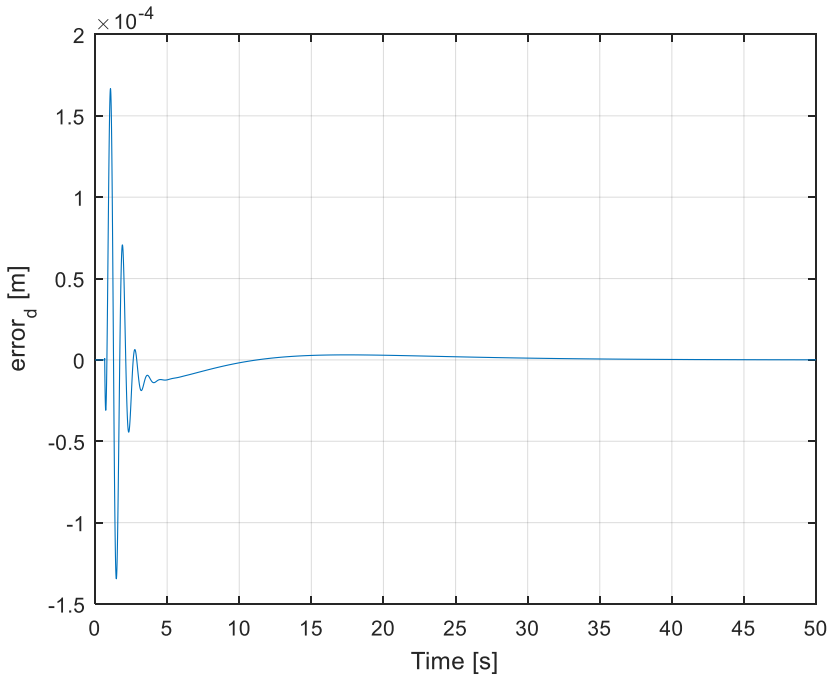


Figure III.11: "error$_d$" is the difference between the distance "d" in the non-linearized dynamic model, and the linearized one.

The time limit of the simulation had to be augmented in order to clearly show the convergence to zero. The linearized system clearly converges to the full dynamic system when $t \to \infty$, if the curvature of the trajectory remains constant, and is under the value $c_{MAX} = 16.67\frac{1}{m}$. Like in Test1: the linearized model tends to the non-linearized one, even though the curvature is not 0. The simulations have been confirming our theoretical analysis so far.

Experimental results:

When we test this trajectory in the laboratory with a similar PI controller in the vehicle: the response appears to be the same as the one in the simulation: the vehicle follows the trajectory and we can see how the steady state error tends to 0 (how the measuring point of the vehicle is every time closer to the trajectory).

3.4 Test 3: The sinusoid pattern trajectory

This trajectory is a sinusoid of the form:

$$y = A * sin\left(\frac{x}{T}\right)$$

Where: A=0,1 [m]; $T = 0{,}5/2\pi$ [m/rad]

Simulations results:

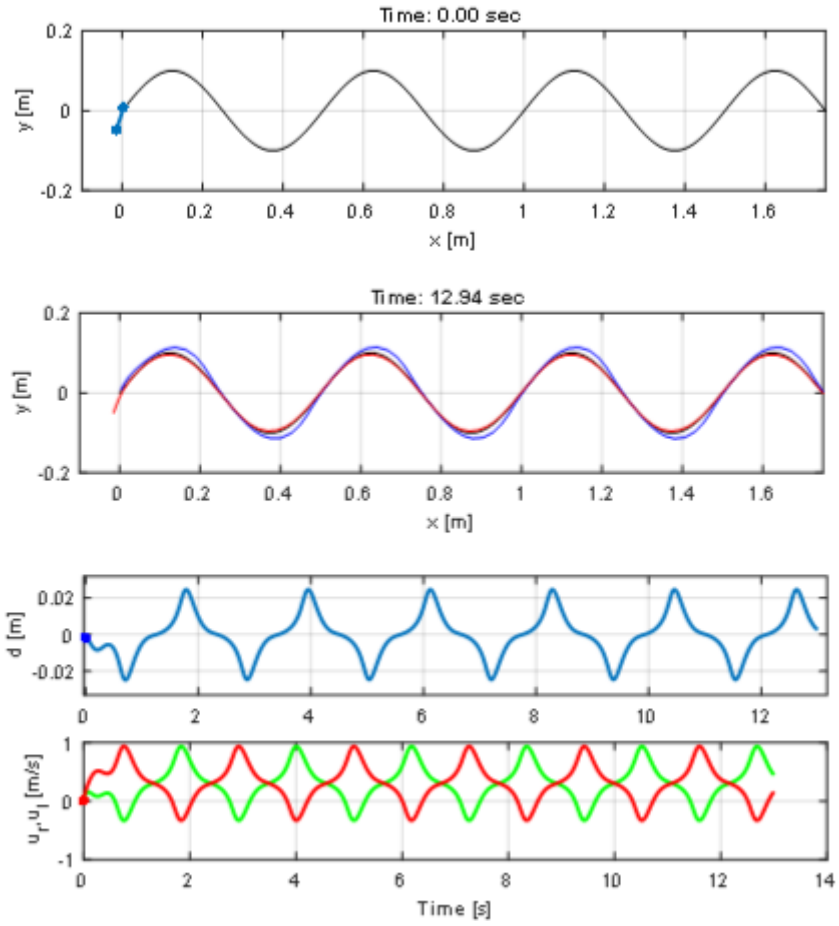


Figure III.12: Screen capture of the beginning (upper plot) and the end (middle plot) of the simulation (with the non-linear dynamic equations) on the trajectory of Test3. For more details on the colors of the upper plots: refer to the first figure in III.ii.

This result was also to be expected. If the curvature of the trajectory $c(s)$ is not constant, and the applied controller is not tuned to counter it, the error "*d*" does not converge to zero.

If the non-linear model does not converge to the working trajectory (if "*d*" does not converge to zero), then it is safe to assume that the linearized model will not be able to converge to the

non-linear model. In the following simulations the vehicle should not be able to follow the trajectory like it did in the other two tests when using the linearized dynamical equations.

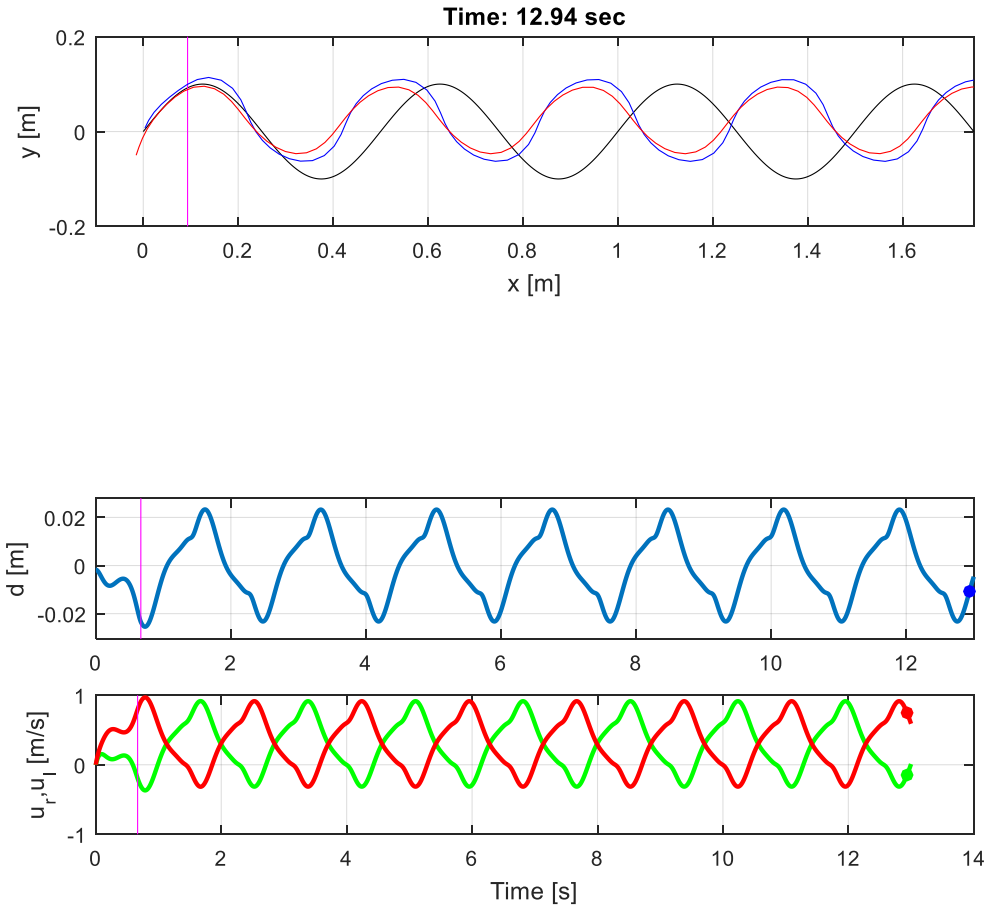


Figure III.13: Simulation of the sinusoid pattern trajectory using the linearized dynamic equations from the magenta line.

Here we see that: if the vehicle's dynamics change to the linearized model, it cannot keep up tracking the trajectory. This result, as stated before, is to be theoretically expected when the system does not converge to the working trajectory (where: $d \to 0$) using the non-linearized dynamical model. We will now show the difference between the state variable *d* between the two simulation in *Figures III.12 and III.13*.

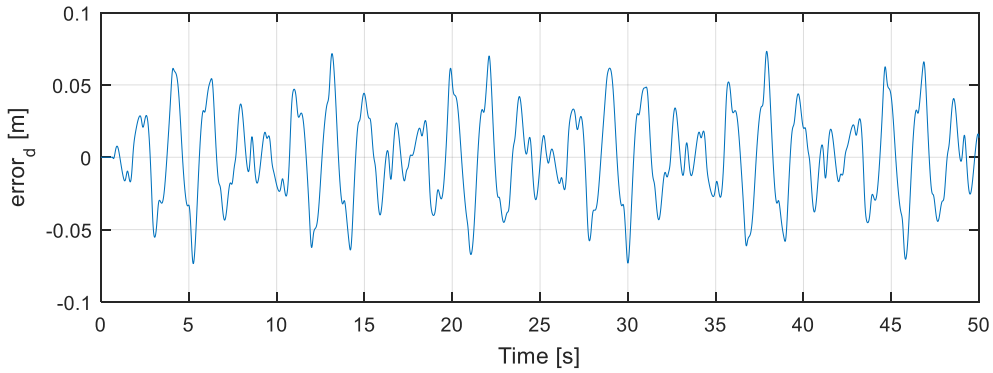


Figure III.14: "$error_d$" is the difference between the "d" in the non-linearized dynamic model and the linearized one.

The difference between the two models clearly does not converge to 0. This could be seen in *Figure III.14* where the vehicle's trajectory does not follow the black line.

Experimental results:

In the laboratory, the vehicle is able to correctly track the line with a controller similar to the one simulated. We are able to tell just by looking that, like in the simulation, the steady state error does not converge to 0 through the test.

We conclude that the vehicle is able to track trajectories where the curvature is not constant. But it is not able to follow them as precisely as the trajectories with constant curvatures, at least using the current simulated PI controller.

Chapter 4: Conclusions

As previously stated, this project is only the first one of a series of projects around the line tracker vehicle. We completed all the objectives we settled at the beginning of the project, the robot itself is working quite well at the moment and it responds accordingly to our model predictions, but there are still some subjects to be solved.

The biggest setback we had doing the project was the fact that we had no way of getting information from the vehicle while it moves and is tracking the line. All the tests and tuning had to be done while the robot was wired, using the laboratory's oscilloscope or the computer's debug tool. This fact severely limits us when testing the vehicle's general performance, and as a consequence it also limits the optimization of the hardware and the control. It should be pointed out that while this project was being done, another student was working on the vehicle's communication via Wi-Fi with the computer for his own project, but due to external circumstances: it could not be finished on time. The main idea behind it was to be able to monitor the vehicle's variables while it was tracking the line to make sure that the vehicle was responding accordingly to the model. This communication tool would have allowed us to study the live steady state error of the vehicle, and see if the wheel speeds match the control signals

The next step in this series of projects would be to finish the communication part, and then making sure that the wheel speeds match the control signal properly, in other words: making sure the speed controller is working properly. In the model we are using it is implied that each of the vehicle wheels is perfectly controllable, if this was not true in the real vehicle: then the model (and designed controllers from the model) cannot be expected to work properly. Then, the following step would be to properly characterize the DC motors with the help of the communication software.

New and advanced controllers could be designed, once the hardware is completely characterized, in order to improve the performance of the tracking for a wider range of trajectories, especially trajectories where the curvature is not constant.