End of Degree Project

Bachelor's degree in Industrial Technology Engineering

Design of controllers and its implementation for a line tracker vehicle

Author: Albert Costa Ruiz
Director: Arnau Dòria Cerezo
Víctor Repecho del Corral

Call: January 2017

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Summary

This End of Degree Project is the result of joining two previous projects and its continuation. One was about the building of a two-wheeled line tracker vehicle (*'Control design and implementation for a line tracker vehicle'* by Ivan Prats Martinho) and the other one dealt with implementing a communication system which allows exchange of information between devices *('Disseny i implementació d'un Sistema de comunicacions WiFi per a una xarxa de vehicles autònoms'* by Antoni Riera Seguí).

The first part of the project consists of a motor control enhancement of the line tracker vehicle. It is used a feed forward signal to make a first attempt to control the motors, and right after a Proportional Integral controller is implemented to adjust the response. The second part is about the wheel speed measurement system. In that section it is dealt with the encoder's limitations and how to solve them to get an appropriate performance. The third part is addressed to the line sensor and the trajectory control. An improvement of the line sensor data acquisition is exposed and it is explained how the trajectory control operates.

Apart from all the tasks done with the vehicle's control, there is a section in which it is worked on the WiFi communication system which monitors the vehicle from a computer. It is explained how it works, how to get a suitable usage and it is shown an interface to manage and see the data transmitted.

Finally it is presented an experimental test set in which it is shown how the vehicle's behaviour is. In these tests it is determined which control parameters are the most suitable to get the best behaviour of the vehicle.

Contents

1. Glossary

All along this project words, abbreviations and expressions in italics are defined in this section.

A **motor driver** is a little current amplifier; the function of it is to take a low-current control signal and then return it a higher current signal that can drive the motor. L298N it is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors.

A **feed forward signal** is pathway within a control system which passes a controlling signal from a source in its external environment, often a command signal from an external operator, to a load elsewhere in its external environment.

A **Digital Signal Processor (DSP)** is a specialized microprocessor, with its architecture optimized for the operational needs of digital signal processing.

An **encoder** is a sensing device that provides feedback.

A **prescaler** is an electronic counting circuit used to reduce a high frequency electrical signal to a lower frequency by integer division.

A **watchdog timer** is an electronic timer that is used to detect and recover from computer malfunctions.

A **phototransistor** is a semiconductor light sensor formed from a basic transistor with a transparent cover that provides much better sensitivity than a photodiode

An **Analog-to-Digital Converter** (ADC, A/D, A–D, or A-to-D) is a system that converts an analog signal into a digital signal.

WiFi is a technology for wireless local area networking with devices based on the IEEE 802.11 standards.

TPC/IP is suite of protocols for communication between computers, specifying standards for transmitting data over networks and used as the basis for standard internet protocols.

A **Central Processing Unit (CPU)** is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output operations specified by the instructions.

Random-Access Memory (RAM) is a form of computer data storage which stores frequently used program instructions to increase the general speed of a system. A random-

access memory device allows data items to be read or written in almost the same amount of time irrespective of the physical location of data inside the memory.

A **Universal Asynchronous Receiver/Transmitter (UART)** is a computer hardware device for asynchronous serial communication in which the data format and transmission speeds are configurable. A UART is usually an individual (or part of an) integrated circuit (IC) used for serial communications over a computer or peripheral device serial port.

A **voltage regulator** automatically maintains a constant voltage level. A voltage regulator may be a simple "feed-forward" design or may include negative feedback control loops.

A **modem (modulator-demodulator)** is a network hardware device that modulates one or more carrier wave signals to encode digital information for transmission and demodulates signals to decode the transmitted information.

IEEE 802.11 is a set of media access control (MAC) and physical layer (PHY) specifications for implementing wireless local area network (WLAN) computer communication

Flow control is the process of managing the rate of data transmission between two nodes to prevent a fast sender from overwhelming a slow receiver.

Multiplexing is a method by which multiple analog or digital signals are combined into one signal over a shared medium.

A **Comma-Separated Values (CSV)** file stores tabular data in plain text. Each line of the file is a data record. Each record consists of one or more fields, separated by commas. The use of the comma as a field separator is the source of the name for this file format.

Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java.

A **router** is a networking device that forwards data packets between computer networks.

A **microprocessor development board** is a printed circuit board containing a microprocessor and the minimal support logic needed for an engineer to become acquainted with the microprocessor on the board and to learn to program it. It also served users of the microprocessor as a method to prototype applications in products.

A **microcontroller** is a small computer on a single integrated circuit. It contains one or more CPUs (processor cores) along with memory and programmable input/output peripherals.

A **debugger or debugging tool** is a computer program that is used to test and debug other programs (the "target" program). The code to be examined might alternatively be running on an instruction set simulator (ISS), a technique that allows great power in its ability to halt when specific conditions are encountered.

2. Preface

This project is a continuation of two previous projects, as it was said in the Summary, and more globally it is one of the first steps to build a set of two-wheeled robots emulating vehicles in a road. These vehicles will be in communication between them and they will be used to simulate different scenarios such as platooning, autonomous vehicles, overtaking manoeuvres, automatic vehicle lane changes and shockwave traffic jams.

In one of the previous projects, called '*Control design and implementation for a line tracker vehicle',* the vehicle was assembled (all the vehicle mechanic parts and the implementation of the sensors). Also it was coded a first version of the robot's firmware which allowed the robot to follow a track.

In the other previous project called *'Disseny i implementació d'un Sistema de comunicacions WiFi per a una xarxa de vehicles autònoms'* a communication system was created to exchange data between different devices. It is told in detail the practical case of communication between a line tracker vehicle and a computer.

These two projects were done during the same period of time and when the one that dealt with the communication system finished, there was no time left to properly test it in the vehicle. That is the reason why it was not possible getting information from the vehicle while it was tracking the line. That meant a great handicap to design the vehicle's control, so one of the reasons of doing this project is taking advantage of the communication system to have a feedback and design a good vehicle's control.

3. Introduction

3.1. Objectives

This project starts on the basis of all the work done in the two previously mentioned projects [1] [2] and it has two basic objectives: join the essential elements of these two projects and improve the performance of the vehicle's control.

Improving the vehicle's motor control involves getting a reliable measurement system of the wheel speeds and a fast and accurate response of the motors. About the part addressed to the trajectory control, it is pursued to enhance the data acquisition captured by the photodetector and evaluate the vehicle performance in different tracks.

Thanks to the communication system it will be possible to check the motors response and the vehicle behaviour following a track. Regarding this communication system, it will be done an overview of it and the main aspects required to get a suitable operation will be explained.

3.2. Starting point

3.2.1. The vehicle

All the pieces and elements that constitute the vehicle had been got by buying the Arduino kit 'Kit Robot LRE-EO2' at 'leantec' website. As it was said, the vehicle was already assembled and it is formed by the following components:

- A chassis robot 2WD (2 wheels, 2 DC bidirectional motors, 2 encoder wheels and 1 chassis).
- An USB cable required to compile into the board.
- A motor driver L298N.
- An ultrasonic sensor HC-SR04.
- Colour cables.
- A battery holder for four AA batteries.

Other components that were not included in the kit were:

- A STM32F4-Discovery board.
- A WiFi Module ESP8266, version ESP-1.
- A sensor LRE-F22 (line sensor).
- A regulator circuit LM317T to get the 3V from the 5V of the batteries.

Pictures in Figures 3.1 and 3.2 show the line tracker vehicle:

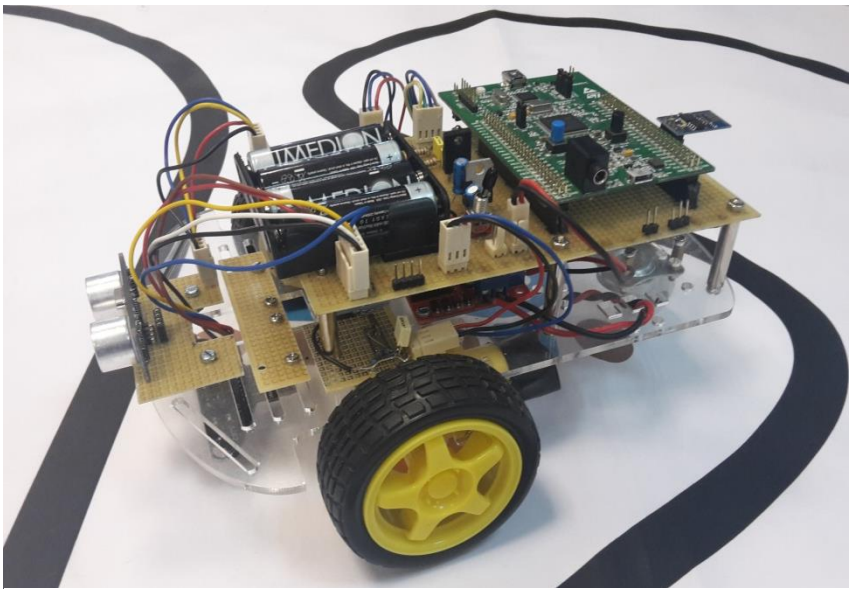

Fig. 3.1 Photo 1 of the line tracker vehicle.

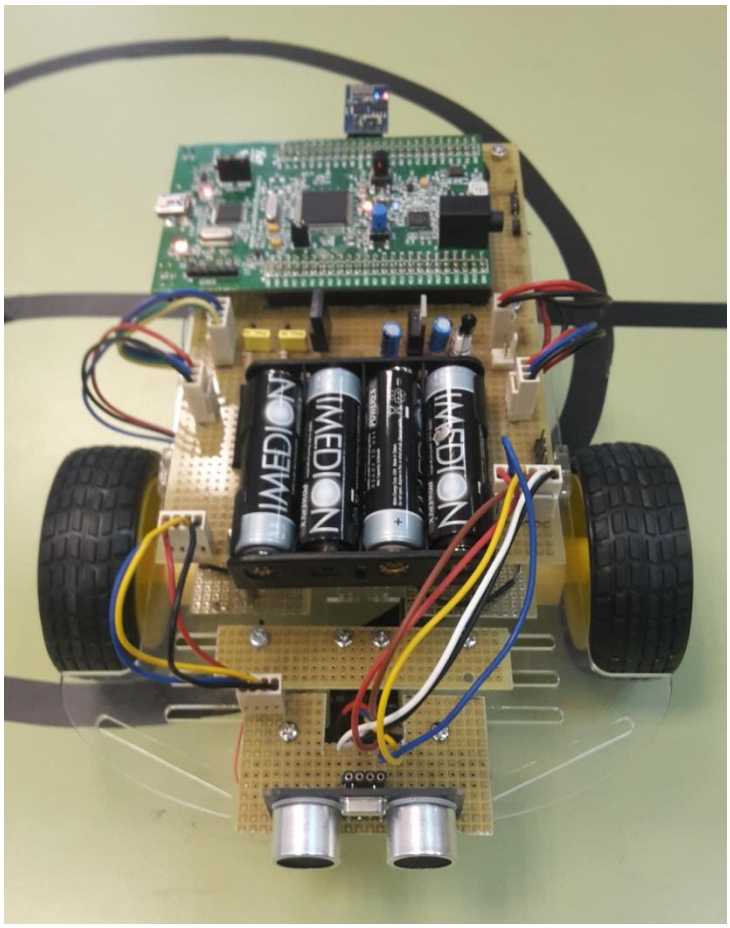

Fig. 3.2 Photo 2 of the line tracker vehicle.

4. Modelling and control design of a dual wheeled vehicle

4.1. Overall control scheme

Figure 4.1 shows the system that is pursued to control. It represent the elements that make up the line tracker vehicle, the input and output signals that are involved and the location where the controllers will act.

This system can be devided in two subsytems: a internal one that is formed by the DC motors and an external one that is the entire vehicle. Both present a control loop that proveides feedback data captured by a sensor. The internal one is a feedback loop in charge of controlling the wheel speeds of the vehicle. Its input signals are the wheel speeds arranged and its feedback signal is the measurement of the wheel speeds provided by two rotary encoders. While the external one is responsible of setting the wheel speeds with the aim that the vehicle followed the path. Its feedback signal is a value in proportion to how far the vehicle is from the line, and it is provided by two photodetectors.

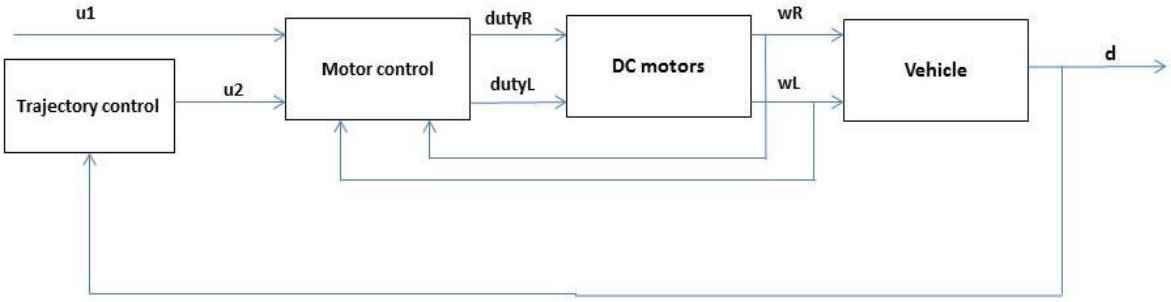

Fig. 4.1 Illustration of the model and its parametrization.

In the upper figure, $u1$ represents a prearrange speed that defines how fast the vehicle follows the line (its meaning will be detailed in next sections); $u2$ is the control signal that corrects the vehicle's trajectory ($u1$ and $u2$ define the wheel speeds using a kinematic relation); $dutyR$ and $dutyL$ are the duty cycle of each motor; wR and wL are the wheel speed measurements and d is the distance up to the track.

These two subsystems will be treated in isolation due to they have a great different dynamic behaviour. So each one will be studied and implemented independently which means that once the motor control is stablished, the trajectory control will be settled on the basis of the perfect operation of the motors.

4.2. DC motors

4.2.1. Modelling of DC motors

Two DC motors are the actuators that rotate the vehicle's wheels. The system structure chosen to model them is the common one, which includes an armature resistance R_a and a winding leakage inductance L_a [6].

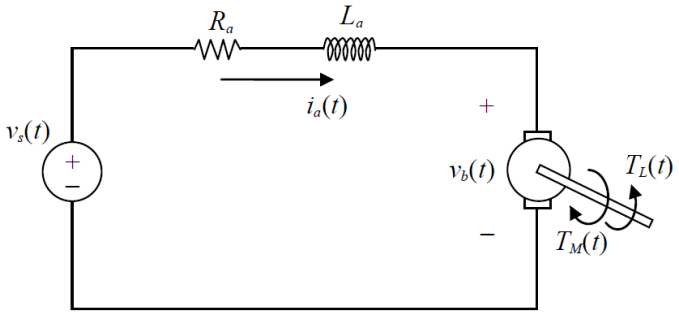

Fig. 4.2 Scheme of the DC motor.

According to the Kirchhoff's voltage law, the electrical equation of the DC motor is described as

$$R_a i_a(t) + L_a \frac{di_a(t)}{dt} + v_b(t) = v_s(t) \qquad \textit{(Eqn.4.1)}$$

where $i_a(t)$ is the armature current, $v_b(t)$ is the emf (electromotive force voltage) and $v_s(t)$ is the voltage source. The back emf voltage $v_b(t)$ is proportional to the angular velocity $w(t)$ of the rotor in the motor, expressed as

$$v_b(t) = k_b w(t) \qquad \textit{(Eqn.4.2)}$$

where k_b is the back emf constant. In addition, the motor generates a torque T_m proportional to the armature current, given as

$$T_m(t) = k_t i_a(t) \qquad \textit{(Eqn.4.3)}$$

where k_t is the torque constant. Besides, if we want to describe its mechanical behaviour, and $T_L(t)$ is defined as an external torque of payload, then we have the following equation

$$B_w w(t) + J_M \frac{dw(t)}{dt} = T_M(t) - T_L(t) \qquad \textit{(Eqn.4.4)}$$

where B_w is the frictional coefficient and J_M is the rotor moment of inertia.

Based on Equations 4.1, 4.2 and 4.4, the dynamic equation of the DC motor can be expressed as:

$$\begin{cases} R_a i_a(t) + L_a \frac{di_a(t)}{dt} + v_b(t) = v_s(t) \\ B_w w(t) + J_M \frac{dw(t)}{dt} = T_M(t) - T_L(t) \end{cases} \quad \textit{(Eqn.4.5)}$$

If this model is implemented in a negative feedback system where the input signal is the applied voltage $v_s(t)$ and the output signal is the angular velocity $w(t)$ of the rotor in the motor, it is obtained the following transfer function

$$\frac{W(s)}{V_s(s)} = \frac{K_t}{(L_a s + R_a)(J_M s + B_w) + K_b K_t} \quad \textit{(Eqn.4.6)}$$

4.2.2. Control of the motors

Control action is given by a Proportional Integral controller which implements the following signal

$$u(t) = u_p(t) + u_i(t) = K_p \cdot e(t) + K_i \int_0^t e(t)\, dt \quad \textit{(Eqn.4.7)}$$

where K_p is the proportional motor gain; K_i is the integrator motor gain factor and $e(t)$ is the error $e(t) = w^* - w(t)$.

Figure 4.3 shows how the motor control that will be implemented. We have to point out that a feed forward signal will be implemented in order to improve the control's characteristics. This feed forward signal is a relation between the duty applied and the rotation seed developed by the motors. This relation is empirical and it will be explained with more detail in a next section.

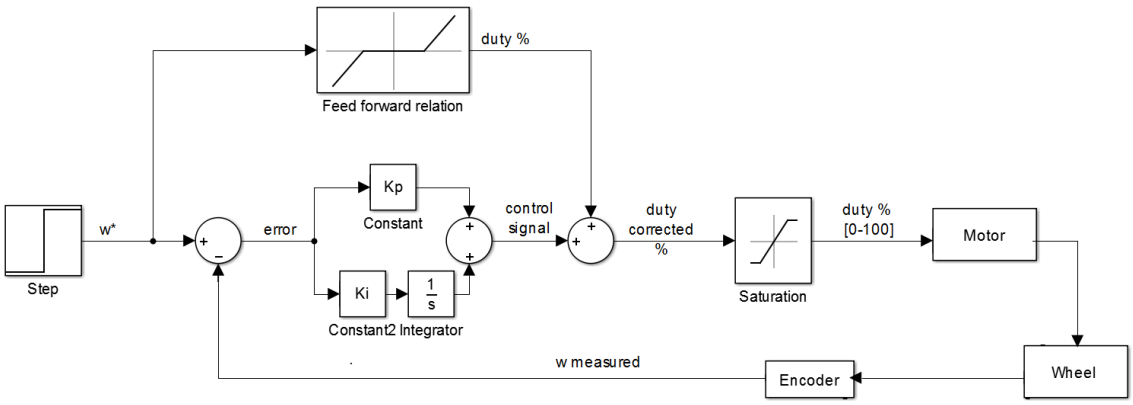

Fig. 4.3 Scheme of the DC motor control.

4.3. The vehicle

4.3.1. Model of a two-wheel vehicle tracking a path

The idea of vehicle model was taken from two articles [3] [4]. It consisted of parametrizing the trajectory that the vehicle has to follow in order to obtain the kinematic model. Figure 4.4 shows the dual wheeled vehicle and a tracking path that is persuaded to be followed with a certain speed v.

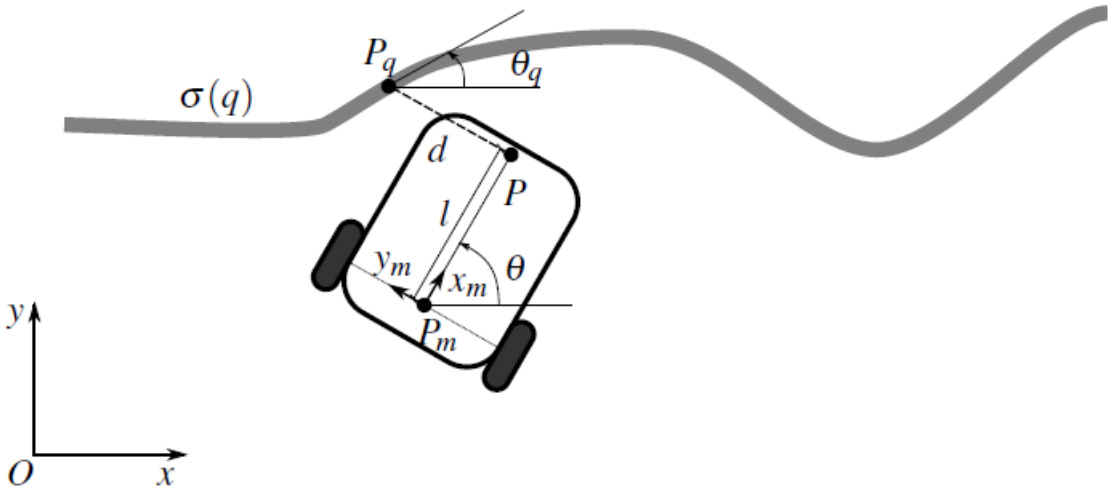

Fig. 4.4 Illustration of the model and its parametrization.

The path and the vehicle are the two elements that are going to be studied in our system; each one has its own coordinate system. The kinematic model of the unicycle-type mobile robot with respect to point P_m (located in the middle of the wheels axel) is given by the following equations:

$$\begin{cases} \dot{x} = \cos(\theta)\, u_1 \\ \dot{y} = \sin(\theta)\, u_1 \\ \dot{\theta} = -u_2 \end{cases} \qquad \textit{(Eqn.4.8)}$$

with

$$u_1 = \frac{r}{R} \cdot (w_r + w_l)\,; \;\; u_2 = \frac{r}{2R} \cdot (w_r - w_l) \qquad \textit{(Eqn. 4.9)}$$

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 others element are:

- 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 x_m is perpendicular to the wheel axis, and 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 x of the global coordinate system, and x_m. This coordinate system will be usually referred to as the "relative coordinate system" in this paper.
- (x,): 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 x_m. This is the point that has to be over the trajectory.
- $P_q(x,y)$: the point in the trajectory that the vehicle is meant to follow. It is defined by the point in the trajectory (q) that is in the coordinates: (0,) of the relative coordinate system of the vehicle.
- $\theta_q(q)$: the angle between the tangent to the trajectory $\sigma(q)$ in the point $P_q(x,y)$, and the x axis of the global coordinate system.

The conditions than must be met to say the vehicle follows a track are:

- $P_q = P$ or equivalently $d=0$.
- $\theta \sim \theta_q$ which means the vehicle is aligned with the trajectory.

It will be studied the dynamics of these two state variable (d and θ), and the state variable q in order to make the vehicle move at a constant velocity.

From the Figure 4.1 can be seen that

$$P_q = P_m + P = P_m + R(\theta)\binom{l}{d}; \qquad \textit{(Eqn.4.10)}$$

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

$$R(\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \qquad \textit{(Eqn.4.11)}$$

Parametrising $\sigma(q)$ with respect to q, differentiating *(Eqn. 4.3)* and using (*Eqn. 4.1*), is possible to determine the motion equations in terms of d, q and θ

$$\begin{cases} \dot{d} = lu_2 - \tan(\theta - \theta_q)(u_1 + du_2) \\ \dot{q} = \frac{(u_1 + du_2)}{\cos(\theta - \theta_q)} \\ \dot{\theta} = -u_2 \end{cases} \qquad \textit{(Eqn.4.12)}$$

where $\frac{\partial \sigma x}{\partial q} = \cos(\theta_q)$ and $\frac{\partial \sigma y}{\partial q} = \sin(\theta_q)$.

Finally, defining a deviation angle θ_e, the dynamics simplifies to

$$\theta_e = (\theta - \theta_q)$$

$$\begin{cases} \dot{d} = lu_2 - \tan(\theta_e)(u_1 + du_2) \\ \dot{q} = \frac{(u_1 + du_2)}{\cos(\theta_e)} \\ \dot{\theta}_e = -u_2 - \frac{c(q)}{\cos(\theta_e)}(u_1 + du_2) \end{cases} \quad \textit{(Eqn.4.13)}$$

where $c(q) = \frac{\partial \theta_q}{\partial q}$ is the curvature of (q).

Desired working trajectory

As mentioned above the regulation point in order that the vehicle follows the trajectory is

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

Also it is required that the vehicle goes at a constant linear velocity

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

Applying these conditions to the expression (*Eqn. 4.8)*, the required control values u_1 and u_2, and the corresponding deviation angle are

$$\begin{cases} u_1^* = v\sqrt{1 - l^2 c(q)^2} \\ u_2^* = -c(q) \cdot v \\ \theta_e^* = arcsin(-c(q) \cdot l) \end{cases} \quad \textit{(Eqn.4.14)}$$

Linearized dynamics

If the working trajectory defined in *(Eqn. 4.13)* is linearized it is got the next matrix expression ($c(q)$ is abbreviated by c)

$$\begin{pmatrix} \dot{d} \\ \dot{\tilde{\theta}}_e \end{pmatrix} = \frac{v}{\alpha}\begin{pmatrix} -lc^2 & -l \\ c^2 & lc^2 \end{pmatrix}\begin{pmatrix} d \\ \tilde{\theta}_e \end{pmatrix} + \frac{1}{\alpha}\begin{pmatrix} lc \\ -c \end{pmatrix}\tilde{u}_1 + \begin{pmatrix} l \\ -1 \end{pmatrix}\tilde{u}_2;$$

where $\dot{\tilde{\theta}}_e = \theta_e - \theta_e^*$; $\tilde{u}_i = u_i - u_i^*$ (for i=1 ,2) and $\alpha = \sqrt{1 - l^2 c^2}$. *(Eqn.4.15)*

Transfer function

In order to make a proper theoretical analysis it will be supposed that

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

The control signal applied to the line tracker vehicle will be the input $\tilde{u}_2$, and $\tilde{d}$ the state variable it is wished to control. The transfer function of the linearized system can be obtained by

$$G(s) = C(s\,I - A)^{-1}B \qquad \textit{(Eqn.4.16)}$$

where

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

Finally the transfer function obtained is G(s)

$$G(s) = \frac{D(s)}{U_2(s)} = \frac{ls + v\alpha}{s^2 + l^2c^2} \qquad \textit{(Eqn.4.17)}$$

$$D(s) = \frac{ls + v\alpha}{s^2 + l^2c^2}U_2(s) \qquad \textit{(Eqn.4.18)}$$

4.3.2. Design of the trajectory control

In this section the controller that will be implemented in the vehicle is analysed. The aim of the vehicle's controller is to make the variable d becomes zero, and the variable $\tilde{u}_2$ will be used to get that.

It was decided in one of the previous projects [1] that a Proportional Integral controller was the most suitable to implement. Transfer function for a PI controller is

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

Routh criteria will be used 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

$$D(s) = 1 + C(s)G(s) = 0$$

$$D(s) = s(s^2 + l^2c^2) + (k_p s + k_i)(ls + v\alpha) \qquad \textit{(Eqn.4.19)}$$

$$D(s) = s^3 + a_2 s^2 + a_1 s + a_o$$

Once the Routh criteria have been employed the next condition is got

$$a_2 a_1 - a_0 = k_p l\left(l^2c^2 + k_p v\alpha + k_i l\right) - k_i v\alpha > 0 \qquad \textit{(Eqn.4.20)}$$

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

$$G_{OL}(s) = \frac{k_p(s+a)}{s}\frac{ls + v\alpha}{s^2 + c^2v^2} \qquad \textit{(Eqn.4.21)}$$

The same way it was done in the motor control, a proportional integral controller will be implemented in the same way. Error signal is $-d$ considering that $d^* = 0$. That means u_2 will implement the following signal

$$u_2(t) = K_p \cdot (-d(t)) + K_i \int_0^t -d(t)\, dt \qquad \textit{(Eqn.4.22)}$$

where K_p is the proportional gain; K_i is the integrator gain and $-d(t)$ is the error made. Figure 4.5 shows how this control will be implemented.

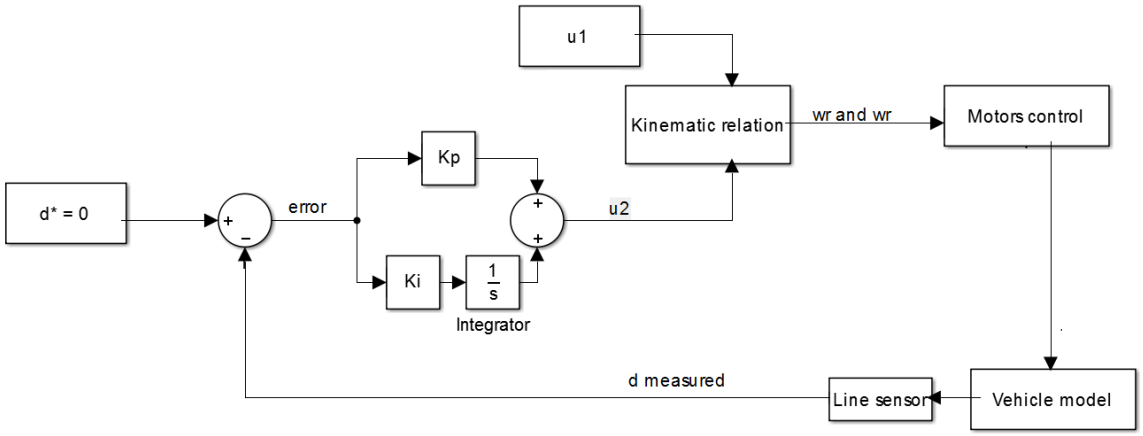

Fig. 4.5 Scheme of the trajectory control.

5. Hardware implementation

5.1. The STM32F4-Discovery board

The line tracker vehicle is managed and controlled by a microprocessor development board created by STMicroelectronics manufacturer. The STM32F4DISCOVERY kit leverages the capabilities of the STM32F407 high performance *microcontroller*, to allow users to easily develop applications.

5.1.1. Discovery board's main features

The STM32F4DISCOVERY kit offers the following features:

- A STM32F407VGT6 microcontroller based on the high-performance ARMCortex-M4 32-bit RISC core operating at a frequency of up to 168 MHz. In the following table its main features and the peripherals that will be employed are shown.

STM32F407 microcontroller's features	
CPU	ARMCortex-M4 32-bit with Floating Point Unit (FPU) single precision.
RAM memory	192 Kb
Flash memory	1 Mb
Clock rate	168 MHz
Timmers	Twelve general-purpose input/output 16-bit timers (GPIO) including two PWM timers for motor control and two general-purpose 32-bit timers.
Communication interfaces	2 UARTs (universal asynchronous receiver / transmitter)
Other peripherals	Three 12-bit ADCs (Analog to Digital Converter) Two DACs (Digital to Analog Converter) A low-power RTC (Real Time Clock)

Table. 5.1 STM32F407 features.

- A ST-LINK embedded debug tool which is an in-circuit *debugger* and programmer for the STM32 microcontroller families.
- An USB OTG FS with micro-AB connector.
- Eight LEDs: LD1 (red/green) for USB communication, LD2 (red) for 3,3V power, four user LEDs; LD3 (orange), LD4 (green), LD5 (red) and LD6 (blue); and 2 USB OTG LEDs LD7 (green) VBUS and LD8 (red) over-current.
- Two push-buttons (user and reset).
- The board power supply is through an USB bus or from an external 5 V supply voltage.
- A LIS302DL 3-axis accelerometer
- An MP45DT02 ST-MEMS audio sensor omni-directional digital microphone.
- A CS43L22 audio DAC with integrated class D speaker driver.

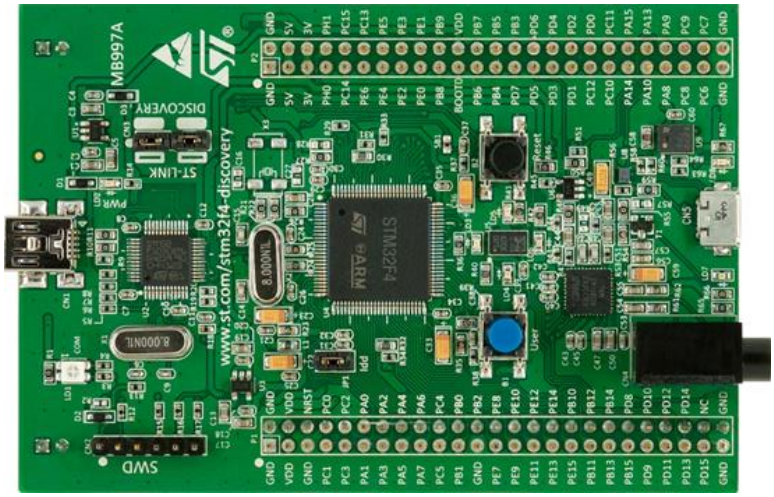

Fig. 5.1 A photo of the STM32F4-Discovery board.

5.1.2. Development of the software's project

To develop this project in C, it has been employed the integrated development environment Eclipse. For code debugging it has been used the Open On-Chip Debugger (OpenOCD), which aims to provide debugging in-system programming for embedded target devices.

Also, it has been used a software tool called STM32Cube that allows configuring STM32 microcontrollers very easily and generating the corresponding initialization C code through a step-by-step process. In this project it was used to configure all the discovery board's peripheries, clocks and middleware setup.

5.1.3. Peripherals and tools employed in this project

In the following list it is shown the peripherals used and what purpose has each one:

- One Advanced-control timer (TIM1) that is used to generate PWM signals to drive the DC motors. Also it is in charge of causing the interruption in which the vehicle's control is executed. This timer has a 168 MHz counter frequency and this interruption is triggered when it reaches 8400, which means the interruption has a 50 µs period.

- Two General-purpose timers (TIM3 and TIM4) to read encoder's frequency. They read pulse signals generated by the encoder's photodetector and assign them the corresponding counter clock value (TIM3 is linked with the left encoder and TIM4 with the right encoder). They have an 84 MHz counter frequency.

- Two General-purpose timers (TIM10 and TIM12) to manage the ultrasonic sensor. The first one generates the trigger signal and the second one reads its response.

- One 12 bit Analog to Digital Converter (ADC1) with two input channels to read each line sensor voltage signal. This data acquisition is done through a Direct Memory Access (DMA) that allows the ADC's data capture while the CPU is doing other operations. Once the operation is done the CPU receives an interrupt from the DMA controller. In this way the data time acquisition is lower.

5.2. Motor control

The main target of this section is controlling the motors mechanical response to get a wished angular velocity in each wheel with great accuracy. An encoder for each wheel will be used to measure the output angular velocity so we will be able to make corrections toward desired performance.

5.2.1. Wheel speed measurement system

The first step to obtain a suitable control of the motors is implementing a feedback loop of the velocity measurements. That will allow the motors to adjust its performance to meet a desired output response.

The rotary encoders

The line tracker vehicle is provided with a rotary *encoder* in each wheel to measure the angular velocity. They use optical technology that allows conversion from motion to digital code. It is said, light beams sent out from a LED are interrupted by the opaque lines on a holey disk before being picked up by a photodetector. This produces a pulse signal that is sent to the DSP and it is processed by timers 3 and 4 according to belonging to the right or left wheel.

Fig. 5.2 The encoder wheel and how it is assembled in the vehicle.

These timers consist of a 16-bit up auto-reload counters driven by a programmable 16-bit *prescaler*. They are capable to count up from 0 to 2^{16} (cMax) with 84 MHz (f_{tim}) counter frequency. The prescaler can be used to divide the counter clock frequency by any factor between 1 and 2^{16} (65536) and it is controlled through a 16-bit register that can be changed on the fly as this control register is buffered.

The counter register (TIMx_CNT), the prescaler register (TIMx_PSC), and the auto-reload register (TIMx_ARR) can be written or read by software, even when the counter is running. The auto-reload register is preloaded at each update event (UEV). This event is sent when channel 1 (timer 3, right wheel) or channel 2 (timer 4, left wheel) receives an input capture coming from the pulse signal produced by the photodetector. Also the update event is produced when the counter reaches the overflow. In following sections (*'Stop detection'* and '*Encoder's lecture*') it is exposed a corresponding prescaler to avoid the overflow.

Figure 5.3 shows encoder and counter register's behaviour.

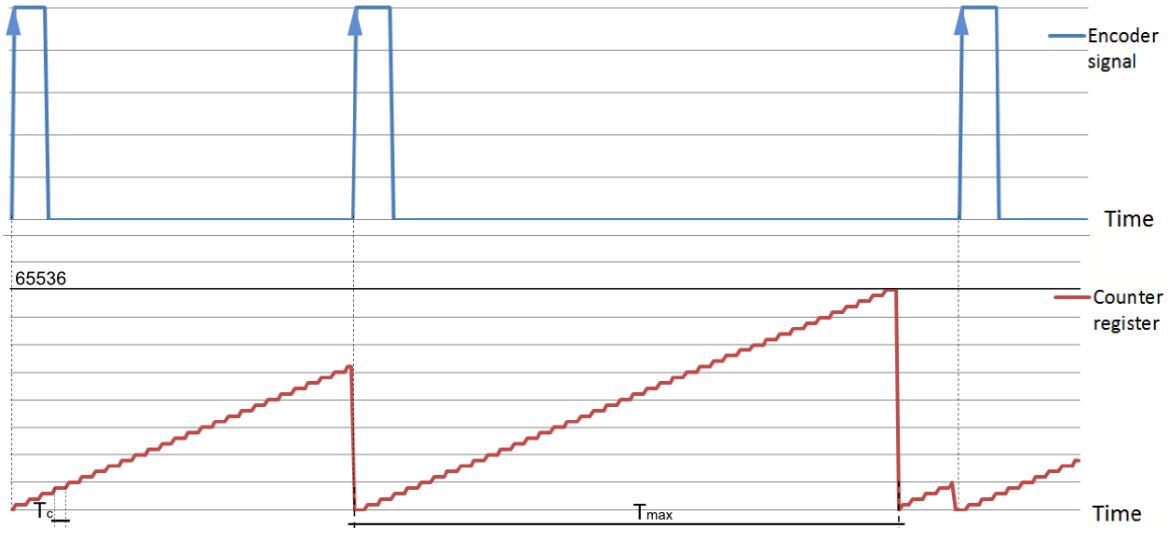

Fig. 5.3 Encoder signal and couter register.

In the first input capture, counter will show a value in proportion to the angular velocity of the wheel. Whereas in the second captured it is reached the overflow and the counter will show a value lower in relation to this angular speed (it will be though wheel is rotating in a higher angular velocity than it does).

Stop detection

As it was mentioned in the previous encoders' introduction, Timers 3 and 4 are responsible of processing encoders signal. These timers have a counter register in which it is loaded the number of the counter when an update event is produced.

The line tracker vehicle can be stopped by the WiFi communication system, by the ultrasonic sensor or by an external disturbance. In these cases the counter register would have been loaded with the counter of the last update event, and that means the measurement system would actually have a wrong speed measurement value.

To solve this problem it was decided to implement a *watchdog* signal in the microcontroller's code. When a trigger event occurs a flag is set by hardware. Every time the main interruption is executed and this flag is not set, a counter is increased. If this counter reaches $2 \cdot 10^4$, it means the vehicle has been stooped for one second (the main interruption is executed every 50 µs). Therefore the corresponding wheel will be assigned a zero value angular velocity whenever the vehicle is stopped for more than one second. The code where the watchdog is implemented can be seen in the annex at '*Code of the control'* section page 66.

Rotation direction

One of the encoder's limitations is the incapability to measure the rotation direction. That is to say wheel's angular velocity is measurable but it is not possible to distinguish if the wheel is rotating backward or forward.

To figure out this it is considered positive rotation when the duty value introduced belongs to the range of the duty's forward relation (that relation can be seen in the section 5.2.2 *Experimental relation between duty signal and motor's response*). When the value belongs to the other range it is considered negative relation. We are conscious that an error is being made since it is considered wheels are already rotating at the desired direction just after the order is given. The code where the direction speed is assigned can be seen in the annex at '*Code of the control'* section page 65.

Encoder's lecture

Firstly it is going to be defined the counter period and maximum time the timer is able to count. These elements are represented in Figure 5.3 and their expressions are

$$T_c = prescaler \cdot \frac{1}{f_{tim}} \qquad \textit{(Eqn.5.1)}$$

$$T_{max} = cMax \cdot \frac{1}{f_{tim}} \cdot prescaler \qquad \textit{(Eqn.5.2)}$$

Prescaler value defines the features of our system of measurement. A low prescaler value gives more accuracy measuring high speeds (the time per count is lower), although the time it can be counting is lower.

Angular velocity of each wheel from the signal of the encoders is calculated using the next equation

$$w = \frac{2 \cdot \pi \cdot f_{tim}}{n \cdot counter \cdot prescaler} \qquad \textit{(Eqn.5.3)}$$

Where f_{tim} is the counter frequency and n is the number of holes of each encoder.

The prescaler value is such that the stop detection is measurable ($T_{max} = 1\ s$) and we got the maximum precision. So the prescaler value is given by the next expression.

$$prescaler = \frac{T_{max} \cdot f_{tim}}{cMax} \qquad \textit{(Eqn.5.4)}$$

$$T_{max} = 1s; \quad f_{tim} = 84MHz; \quad cMax = 2^{16} \rightarrow prescaler = 1281{,}74 \cong 1300$$

It is chosen 1300 as the prescaler's value in order to avoid the risk of reaching the overflow before the stop detection.

It is estimated the motors provide a maximum lineal speed around 12,3 rad/s (duty at 100%). At this speed the period of the output signal of each encoder is approximately 0,025 s. Encoders signal for each wheel at maximum speed can be seen in Figure 5.4.

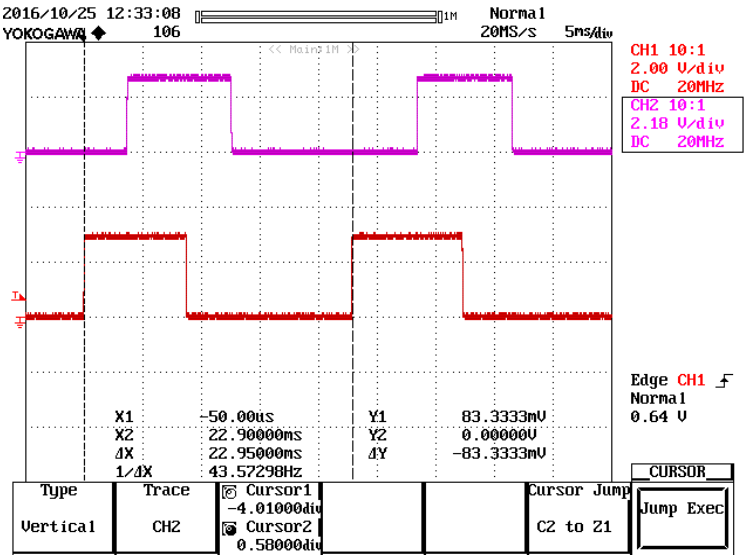

Fig. 5.4 Encoders signal at maximum speed (red line is the right one and the purple line is the left one).

At this maximum speed and with the chosen prescaler, the counter value will be

$$counter = \frac{T_{min} \cdot f_{tim}}{prescaler} \qquad \textit{(Eqn.5.5)}$$

$$T_{min} = 0{,}025s;\ \ f_{tim} = 84MHz;\ \ prescaler = 1300;\quad \rightarrow counter \cong 1615$$

Now it will be made a study of the encoder's lecture precision. Applying the angular velocity equation (Eqn. 5.3) we get the following maximum angular velocity

$$f_{tim} = 84MHz;\ \ n = 20;\ \ \ prescaler = 1300;\ \ \ counter = 1615;$$

$$wMax = 12{,}56936332\ rad/s$$

If the same calculation is done for a counter's value below it and upper it, it is obtained the following results

speed c-1	speed	speed c+1
12,57715103	12,56936332	12,56158525

That means at maximum speed we have the following accuracy

$$a = \left(\frac{Speed_{c-1} - Speed}{Speed}\right) \cdot 100 \cong \left(\frac{Speed - Speed_{c-1}}{Speed}\right) \cdot 100 \cong 0{,}062\ \%$$

It is obvious this accuracy is quite good for the feedback loop of the motor controller.

In order to enhance the encoder's lecture precision, a dynamic scale could be implemented. A prescaler changeable on fly would provide a dynamic behaviour in the measurement. In high speeds a low prescaler value would be used to increase accuracy, while in low speeds it would be used a sufficient value to not reach the overflow. In our case it was checked this implementation wasn't worthy because the accuracy in high speeds almost didn't improve.

Zone of zero speed and maximum speed

The motors have an improper operation at low speeds. When duty signals between speeds of 1,5 rad/s and almost 0 rad/s are applied, wheels rotate in an irregular and non-constant way. The same occurs in (0 , -1,5] rad/s range. On the other hand it was realized that the maximum speed that motors can reach is 18,46 rad/s in both rotating directions. To avoid these malfunction ranges, the angular velocity of the motors is limited in the following working ranges: [18,4 ; 1,5), [0] and (-1,5 ; -18,4] (rad/s).

If in the future of the vehicle's development if it is required higher accuracy in the motors response and a wider working speed range, a current control motor must be implemented as a replacement of the voltage control. With a current control we could control the torque of the motors. The code where the wheel speed values are limited can be seen in the annex at *'Code of the control'* section pages 67 and 68.

5.2.2. Experimental relation between duty signal and motor's response

In order to obtain a *feed forward signal* to make a first attempt to control the motors and to improve the specifications of the controller that will be implemented next, trials testing responses for a wide variety of duties values were performed. In the following images this relation can be seen:

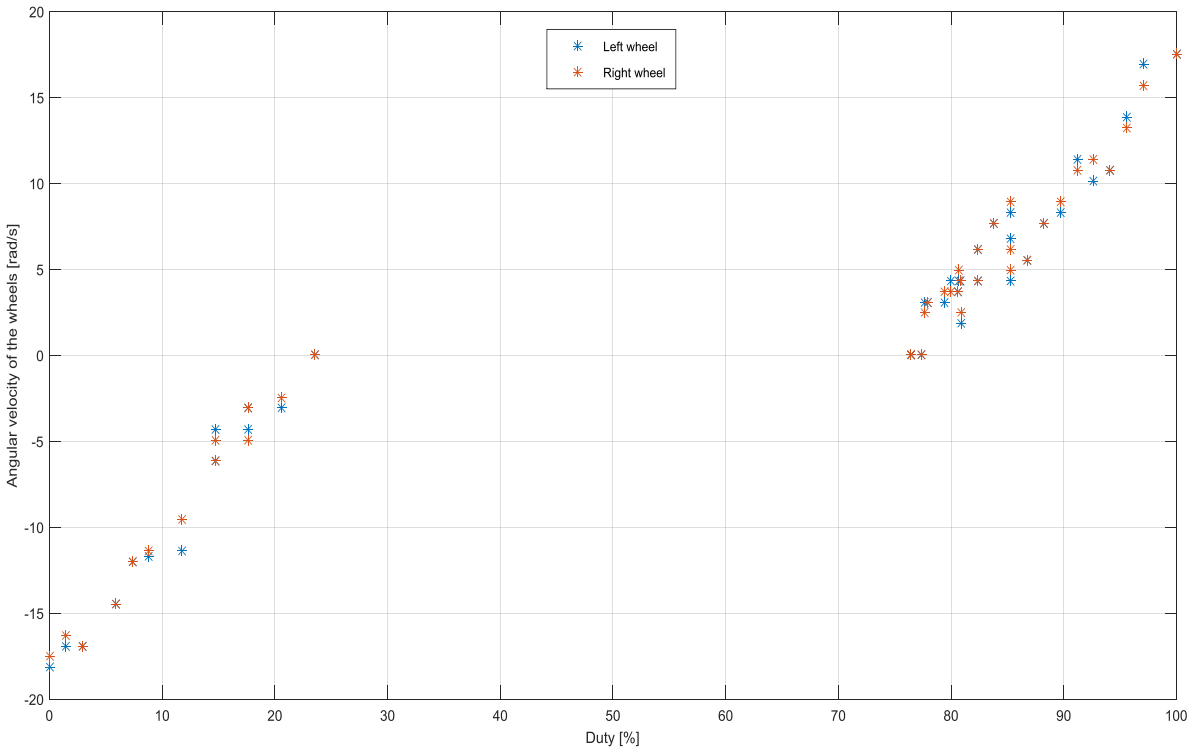

Fig. 5.5 Representation of the angular velocity of the wheels and the corresponding duty supplied.

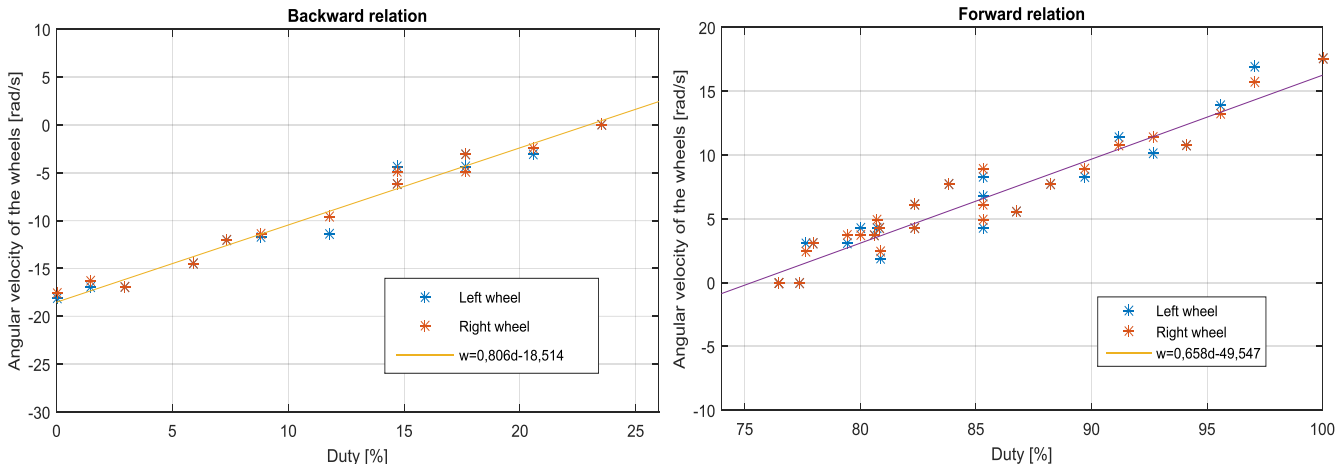

Fig. 5.6 Feed forward relation in backward rotation (w=0,806d-18,514) and in forward rotation (w=0,658d-49,547).

Figure 5.5 shows a significant range of duty values (from 20% to 76%) in which motors do not develop torque enough to rotate the wheels, which means a working dead zone of the DC bidirectional motors. The figure of below (Figure 5.6) shows equations taken to get this approximate relation, in forward rotation direction and in backward rotation direction. Besides it seems clear that the two DC motors have a similar behaviour.

Tests were performed imposing different duty values on the motors. Then while the vehicle was moving in a straight line, speed wheels values were being recorded from a computer using the WiFi communication system.

5.2.3. Proportional Integral controller implementation

In this part a feedback loop will be implemented. It will take the system output (encoder's measurement of wheel speed) and it will be used to know the angular velocity of the wheels and calculate error it is being made ($e = w^* - w\_measured$). That enables the system to adjust its performance to meet a desired output response, it is said, the feed-forward signal will be modified in order to approach the desired wheel speed.

Error signal is multiplied by control actions and the controller output resultant is added up to the percent duty signal coming from the feed forward relation. The duty signal once corrected is transformed to a percent duty signal using the respective linear relation (Figure 5.6); it will be used the forward or backward equation depending on rotation is positive or negative. Right after, this signal is saturated between 0 and 100; it is done an equivalent transformation in [0-8500] range with integer values and it is written in the 'Capture/Compare Register' (CCR) of the Timer 1. Eventually Timer 1 provides the signal to the Driver L298

which drives the motors.

Error signal is updated every T_s (interruption period) and it is used a Discrete-Time Integrator with a Forward Euler method (also known as forward rectangular approximation) to calculate the integral action signal $u_i(t)$.

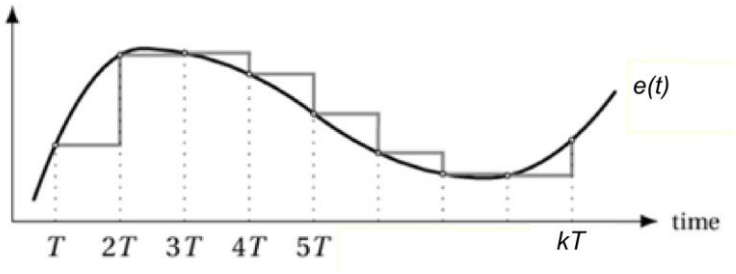

Fig. 5.7 Error discretization.

For this method, the integrator is approximated by $T_s/(z-1)$. The resulting expression for the output integral action signal at step k is

$$u_i(kT) = u_i\big((k-1)T\big) + T_s K_i e((k-1)T) \qquad \textit{(Eqn.5.6)}$$

Since this controller is being implemented in accumulation mode, sampling period is considered one second. Figure 5.8 shows the entire implementation.

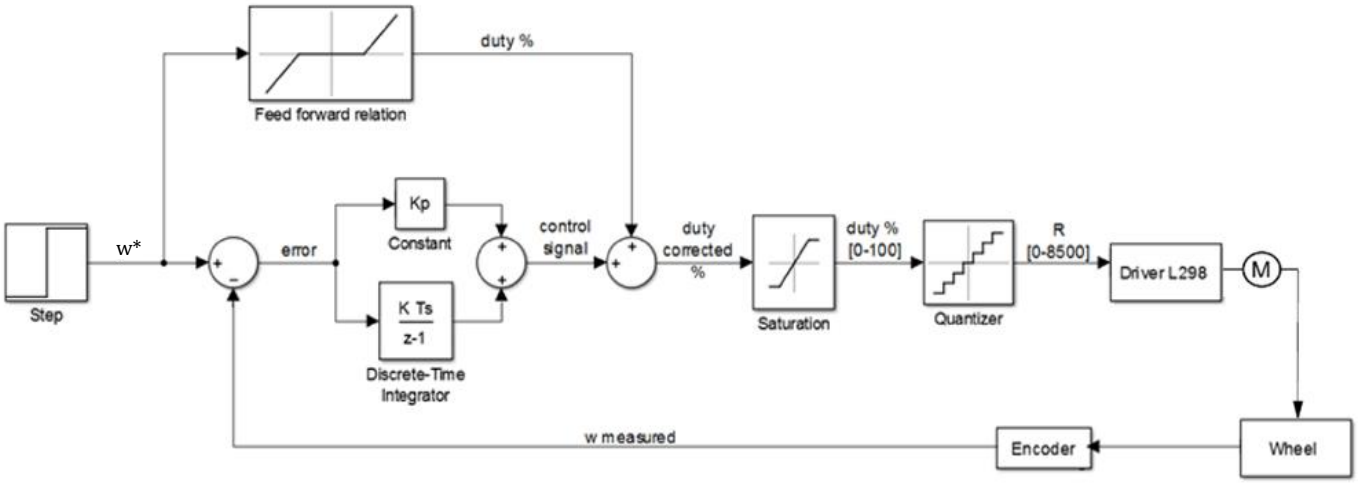

Fig. 5.8 Motor control implementation.

5.3. Trajectory control

The aim of this part is to design a control algorithm that will provide the wheel speeds in order to follow a defined track. A line sensor provides a signal relating the vehicle position with respect to the line. Depending on this value a specific angular velocity for each wheel will be established.

5.3.1. Photodetector

Two photodetectors placed in front of the vehicle are used to determine its relative position. Each photodetector consists of an infrared LED and a *phototransistor*. The infrared LED sends out light beams that strikes the surface and gets reflected back. If the surface is white, more intensity of light gets reflected and for black surface very less it is reflected. The phototransistor is used to detect the intensity of light reflected and the corresponding analogue voltage is induced.

The voltage signals of each photodetector are processed by a 12 bit *Analog-to-digital converter (ADC)* of the DSP. The supply voltage of the photodetector is 3V and the conversion is done to a 0-4096 range (12 bits).

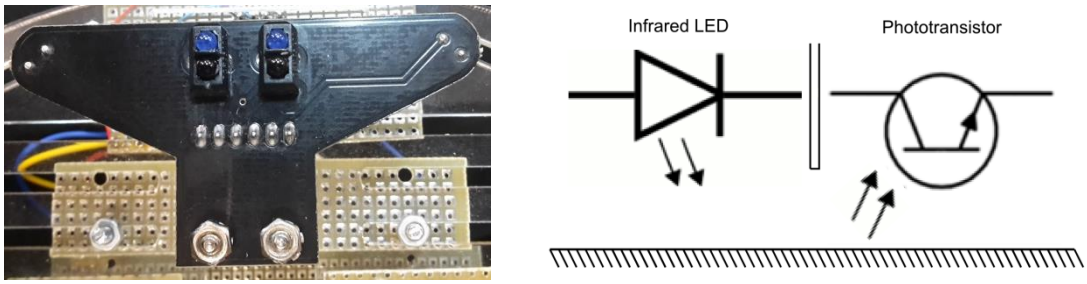

Fig. 5.9 Vehicle's photodetectors and a drawing of how they work.

These converted values are subtracted and the result (V_{diff} = *left photodetector's value - right photodetector's value*) is treated as a proportional number related to the relative position of the vehicle.

The Figure 5.11 shows the captured values when de vehicle is moved perpendicularly across a black 14 mm long line above a light green surface, beginning on the line's right side (Figure 5.10).

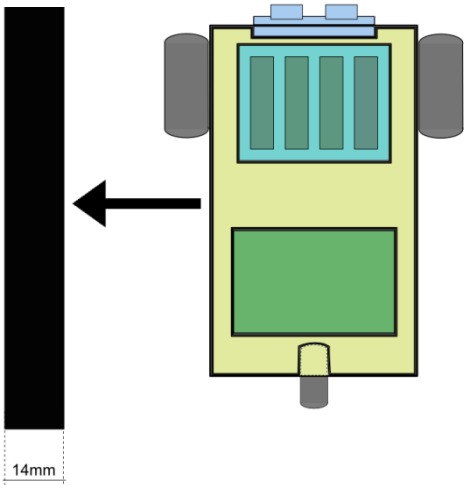

Fig. 5.10 A drawing of how the line sensor's values were captured.

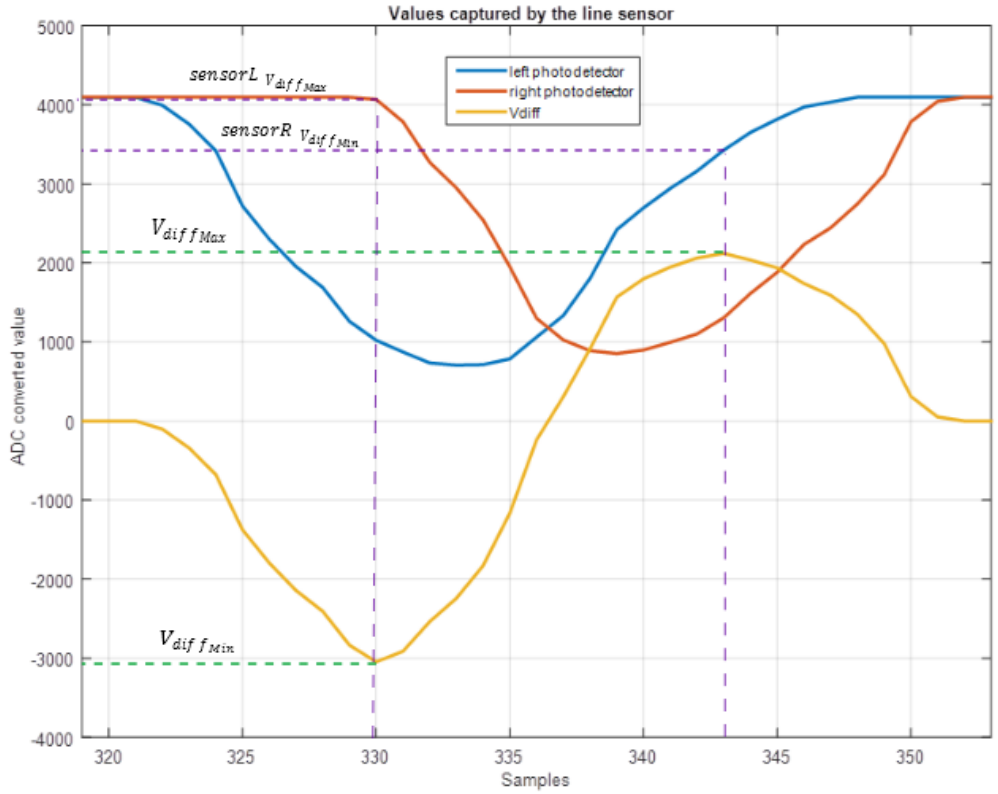

Fig. 5.11 Captured values by the two photodetectors and its subtraction Vdiff.

Improvement of the sensor's detection range

To enlarge the line sensor's detection range and not just working in the linear section of V_{diff}, Boolean conditions with empirical constants were introduced. This way more values of the photodetector are used and some dynamic range is gained. Equation 5.7 shows how photodetector's values are treated now.

$$V_{diff} = \begin{cases} sL + sR - V_{diff_{Max}} & ; \quad if \ \ sL > sR \ \ and \ \ sL > sL\rfloor_{V_{diff_{Max}}} \\ -(sL + sR) - V_{diff_{Min}} & ; \quad if \ \ sR > sL \ \ and \ \ sR > sR\rfloor_{V_{diff_{Min}}} \\ sL - rL & ; \quad otherwise \end{cases} \qquad \textit{(Eqn.5.7)}$$

where V_{diff} is the value in proportion to the distance to the line; sL is the left photodetector's value; sR is the right photodetector's value; $V_{diff_{Max}}$ is the maximum value V_{diff} reaches in the normal configuration; $V_{diff_{Min}}$ is the minimum; $sL\rfloor_{V_{diff_{Max}}}$ is the value that the left sensor

takes when V_{diff} reaches its maximum value and $sR]_{V_{diff_{Min}}}$ is the value the right sensor takes when V_{diff} reaches its minimum value. The first expression with Boolean conditions is linked to the line's left side and the second is linked to the line's right side. This expression is introduced in the code and it can be seen in the annex at the '*Code of the control*' section page 66.

For a suitable behaviour, every time conditions changed (line's thickness, light conditions, etc.) these empirical values must be recalculated.

In Figure 5.11 V_{diff} maximum and minimum values are indicated with green lines and $sL]_{V_{diff_{Max}}}$ and $sL]_{V_{diff_{Max}}}$ are marked with purple lines. This enhancement can be seen in the Figure 5.12 (measurements were taken with the same conditions as in Figure 5.11).

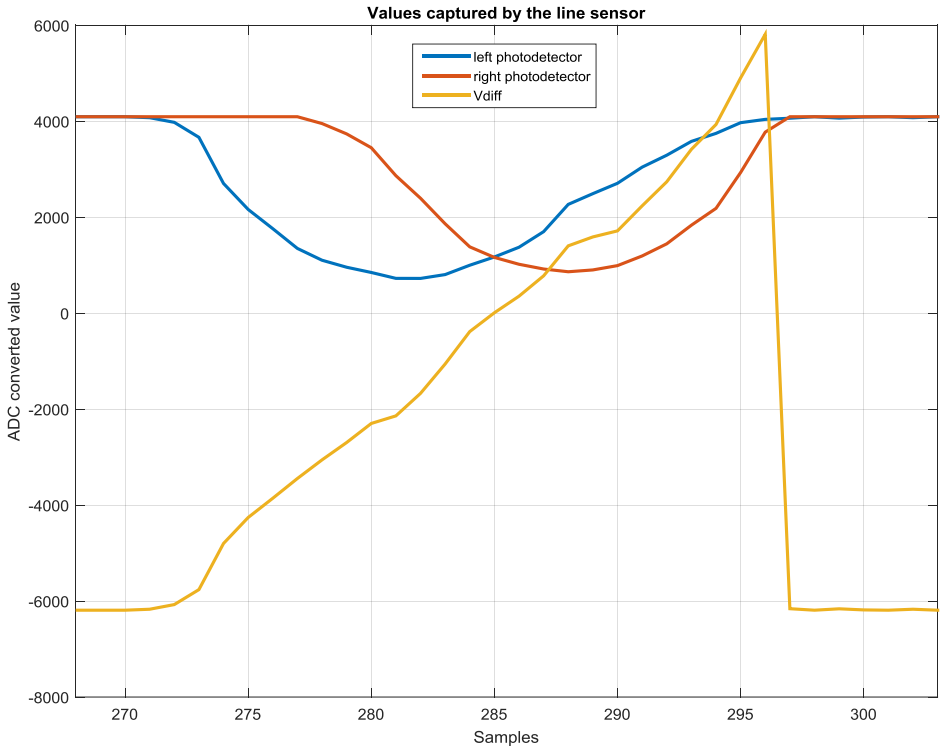

Fig. 5.12 Captured values by the two photodetectors and its subtraction V_{diff} once values are treated with the previous expression.

The range of values V_{diff} that can be taken has increased from a range of [-3000, 2000] to the range [-6000, 6000]. Now the vehicle can detect a wider distance from the line's symmetric axle. However there is a discontinuity when the vehicle reaches the full white surface of the line's left side. In this area the vehicle would take a wrong value in relation to its position.

Finally it is important to emphasise the line sensor is not able to measure a distance, but it provides a value in proportion to how far the vehicle is from line's symmetric axle. In addition the values shown in Figures 5.11 and 5.12 are for a specific case (a black 14 mm long line above a light green surface). If line's thickness or surface's colour changes these values would vary. It is not advisable using a line too wide because the variable V_{diff} would have a big dead zone in the line's middle and the trajectory's control could not work properly.

5.3.2. Wheel speed calculation

As it is shown in the fourth section '*Model of a two-wheel mobile robot tracking a path'*, the kinematic model of the vehicle is given by the following equations

$$u_1 = \frac{r}{R} \cdot (w_r + w_l)$$

$$u_2 = \frac{r}{2 \cdot R} \cdot (w_r - w_l) \qquad \textit{(Eqn.5.8)}$$

Where r is each wheel's radius, R the distance between wheels, $w's$ are the angular velocities of each wheel, u_1 the longitudinal component of the vehicle's linear speed and u_2 the transverse component.

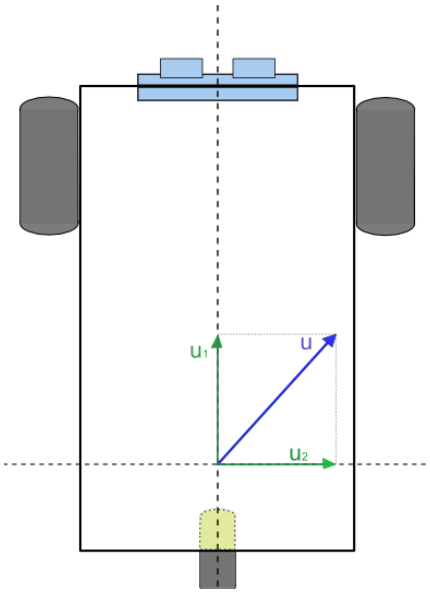

Fig. 5.13 Representation of u_1 and u_2 vectors in a drawing of the vehicle.

If we consider the angular velocities of the wheels depending on u_1 and u_2, we get the expressions which define the vehicle's kinematic model

$$w_l = u_2 + \frac{u_1}{R}$$

$$w_r = \frac{2 \cdot u_1}{R} - w_l \qquad \textit{(Eqn.5.9)}$$

where u_1 is a constant value that determines the linear velocity reached by the vehicle when it follows a straight line. The higher is u_1's value, more quickly the vehicle will follow the trajectory. Meanwhile u_2 is a variable that depends on the trajectory control, it allows for changing vehicles trajectory and it affects both wheel's speed at the same time but in an opposing way (while one wheel speed is increased the other one decreased in the same value). Values w_r and w_l will be the input signals of the motor control.

5.3.3. Proportional Integral controller implementation

In this section through the input signal of the line sensor, a proportional distance feedback loop will be implemented. The u_2's value will be used to adjust the trajectory of the vehicle and it will be modified according to photodetectors' signals (V_{diff} value).

The vehicle must always work in the linear section of the ADC converted value, as it was shown in Figure 5.12. If the line sensor does not work above the linear section, the control will not act proportionally in relation with V_{diff} signal or it will not work in the right way. The desired value that V_{diff} must take is 0, which means both photodetectors are measuring the same value; therefore the vehicle is above the line's symmetric axle.

The same way it was done in the motor control, a proportional integral controller will be implemented. The integral part will be approximated by $T_s/(z-1)$ due to the error signal (the distance to the line) is discretized. The code where this control is implemented can be seen in the annex at *'Code of the control'* section page 67.

Figure 5.14 shows the trajectory's control operation.

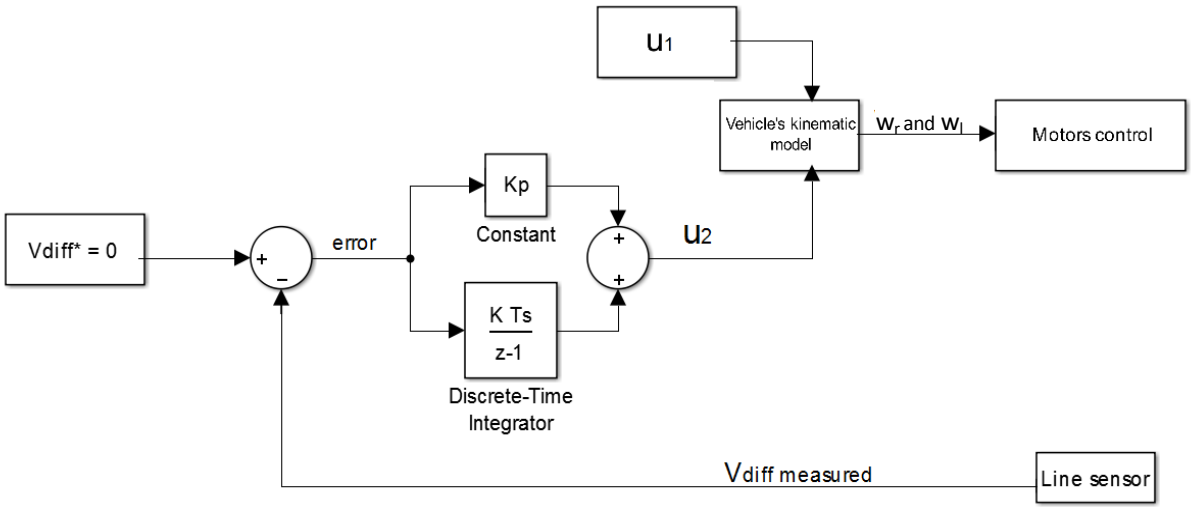

Fig. 5.14 Tajectory control implementation.

6. Communication System

This section is mainly about a project of the student Antoni Riera titled *'Disseny i implementació d'un Sistema de comunicacions WiFi per a una xarxa de vehicles autònoms'* [2]. That project deals with implementing a communication system which allows exchange of information between vehicles. In this work it is going to be done an overview of it and the main aspects required to get a suitable operation will be explained.

6.1. WiFi Module

Firstly it is going to be described the component that allows communication between the vehicle and a computer. The ESP8266 is a *WiFi* chip with *TCP/IP* protocol stack and Micro Controller Unit capability that can give access to a WiFi network. It is capable of either hosting an application or offloading all WiFi networking from another application processor. It is cheap, it has small dimensions (25mm×15mm×1mm) and it was developed by a Shanghai-based Chinese manufacturer, Espressif Systems. It has been commercialized since 2014 and nowadays different versions are sold, in this case we have the ESP-1 version.

The main components that form the WiFi Module are a *CPU*, a *RAM memory*, an aerial WiFi and a *UART* port. The last one is in charge of transmitting and receiving the data (bits) in a sequential way.

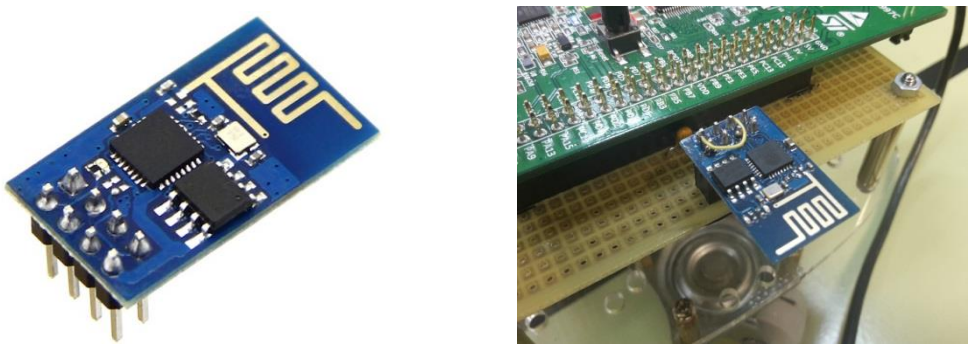

Fig. 6.1 WiFi Module ESP8266, version ESP-1.

The WiFi Module ESP8266 is installed in the high-performance discovery board STM32F4 DISCOVERY as it can be seen at Figure 6.1. It needs an output voltage of 3,3 V that is

supplied by a *voltage regulator* connected to the batteries (a voltage higher than 4,5 V is dangerous). The main features of the ESP8266 module are in Table 6.1.

WiFi module's features	
CPU	Low power 32-bit CPU.
RAM memory	64 Kb for instructions. 96 Kb for data.
Flash memory	1 Mb
WiFi Connectivity	Compatible with IEEE 802.11 b/g/n standard. Compatible with WEP, WPA, WPA2. Integrated TCP/IP protocol stack.
Inputs and Outputs	16 GPIO pins. UART controller. ADC of 10 bits.

Table. 6.1 WiFi Module's features.

6.1.1. Package management system and other specifications

The factory settings of the ESP8266 are not suitable for a reliable usage of this device: transmission time is too long (100 ms) and the package transmission tax (2,5 seconds) not high enough. In the previous project [2] it was decided to use the earliest version of the firmware available (version 1.4.1) which improves these features. To ensure specifications, the features of the ESP8266 were analysed and some trials were carried out in that project. The following list is a brief conclusion of the results that were extracted:

- The highest tax of 30 bytes length package transmission, with no packages lost, was 25 packages/second. Packages temporally equidistant with an error lower than 5%.
- The highest tax of 30 bytes length package reception, with no packages lost, was 50 packages/second. Packages temporally equidistant with an error lower than 5%.
- The highest tax of 30 bytes length package in reception and transmission simultaneously, with no packages lost, was 25 packages/second.

- Time to detect an available network: between 1000 and 3000 ms.
- Time to connect to a network: between 2000 and 5000 ms.
- Time to set a TCP connection: between 500 and 2000 ms.

For these trials, the communication with the computer was done setting a data rate in bits per second (baud) of 112500 bauds (a byte arrives every 71 ms). For more information about these trials, consult the aforementioned project in the beginning of this section [2].

6.2. Computer

A computer is made use of monitoring the vehicle in order to know its performance on-line. Data like wheel speeds, values captured by its sensors or information about the control's execution can be collected and represented. Also it is possible sending commands to the vehicle while it is running. For example, it can be stopped and started, or speed wheels can be changed. In a coming section called '*Firmware and code's structure: Communication routine'*, it is explained how these functions work.

6.2.1. Application for data visualization

An application for a computer allows for monitoring the vehicle, sending commands, collect vehicle's data and represent it graphically. Figure 6.2 shows the application in operation while a test speed was being carried out.

It is made up of three main windows; the two on the right always show reference speed (red line) and the speed measured by the encoder (green line); whereas the upper-middle one represents a chosen state variable. A list on the left shows the state variables the vehicle is sending and the value that they have. If we make double click on one of these variables, it will be plotted on the upper-middle window. It is possible changing the scale of the plots by introducing a maximum and minimum value in the corresponding boxes; if any change is made the variables are plotted with an automatic scale.

There is a command-line interface below the upper-middle window to interact with the vehicle. A window shows the command history, and every time the vehicle receives an order a confirmation is sent back. For example, in Figure 6.2, it has been sent an 'S' to order the vehicle start working and 'WR=0,2' to change linear speed wheels to 0,2 m/s. As it can be seen, every time a command is sent it is shown in the command window with '@' plus the vehicle's IP plus the command sent; and the reception confirmation is represented with the vehicle's IP plus 'K'. Since in future projects more than one vehicle will be monitored, the box next to the command line is used to write on it the vehicle's IP.

The state variables that the vehicle sends are written on a *CSV file* (a file example can be seen at Table 6.2). This file is named by default with the current date and the first file's column always indicates the instants that the state variable values have been received. Due to the fact that the package of state variables is received after it has been built, and receptions are no equidistant in time, a clock was implemented in the control routine of the DSP in order to link a time with each variable captured. In this way it is possible to show these variables on a graph dependent on the time. In the middle windows of Figure 6.2 can be seen the aforementioned clock. Its value represents how many times the main interruption of 50 µs has been executed. The code where this clock is implemented can be seen in the annex at '*Code of the control'* section page 64.

This application was developed by the programming language *Python* and the graphic library Pygraph.

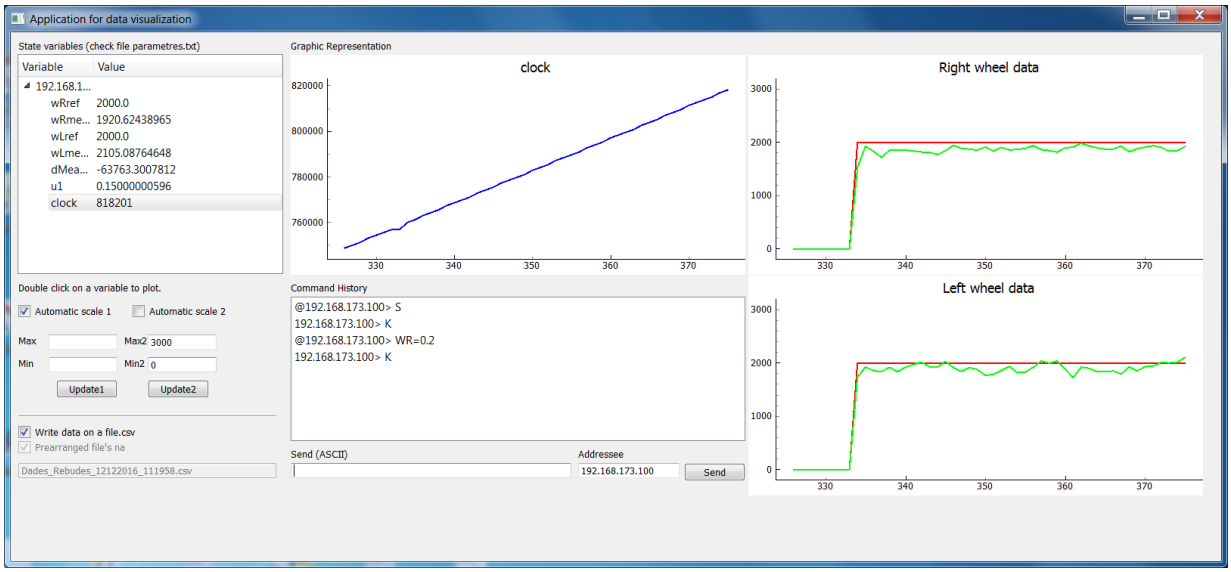

Fig. 6.2 Application for data visualization.

TIMESTAMP	wRref	wRmeasured	wLref	wLmeasured	Vdiff	dist	clock
2016-12-21 11:47:48.602000	1336.31	1336.31	1663.69	1432.34	-238.00	362.00	303043.00
2016-12-21 11:47:48.665000	1331.52	1327.17	1668.48	1395.97	-239.00	377.00	304283.00
2016-12-21 11:47:48.727000	1284.05	1327.17	1715.95	1409.69	-265.00	377.00	305523.00
2016-12-21 11:47:48.790000	1292.48	1390.09	1707.52	1542.88	-259.00	377.00	306762.00
2016-12-21 11:47:48.883000	1313.41	1356.36	1686.59	1528.22	-242.00	387.00	308622.00
2016-12-21 11:47:48.946000	1338.29	1342.01	1661.71	1495.32	-224.00	381.00	309862.00
2016-12-21 11:47:49.008000	1344.76	1409.09	1655.24	1534.27	-218.00	388.00	311102.00
2016-12-21 11:47:49.102000	1373.77	1475.26	1626.23	1596.65	-197.00	398.00	312962.00
2016-12-21 11:47:49.258000	1373.77	1475.26	1626.23	1596.65	-197.00	398.00	312962.00
2016-12-21 11:47:49.289000	1429.42	1456.37	1570.58	1445.83	-159.00	392.00	316681.00
2016-12-21 11:47:49.382000	1525.99	1450.60	1474.01	1496.00	-97.00	426.00	318541.00
2016-12-21 11:47:49.445000	1513.21	1411.50	1486.79	1430.16	-106.00	412.00	319781.00
2016-12-21 11:47:49.507000	1569.02	1446.79	1430.98	1513.85	-71.00	447.00	321021.00
2016-12-21 11:47:49.601000	1550.18	1419.09	1449.82	1407.58	-85.00	433.00	322880.00
2016-12-21 11:47:49.663000	1556.68	1414.53	1443.32	1282.53	-82.00	421.00	324120.00
2016-12-21 11:47:49.710000	1602.35	1398.33	1397.65	1314.74	-54.00	426.00	325360.00
2016-12-21 11:47:49.819000	1645.74	1352.75	1354.26	1380.78	-29.00	447.00	327220.00

Table. 6.2 CSV file in which the variables sent by the vehicle are written.

6.3. Operation of the transmission

6.3.1. Package transmission

An universal asynchronous receiver/transmitter (UART) peripheral integrated in the STM32F407 microcontroller allows an output/input series communication. It sends and receives data, but it does not save it. Bytes are sent one by one, and only the last byte is stored in a register; when the next one arrives the previous one is lost. Therefore each byte is copied to the RAM memory of the microcontroller before the UART peripheral receives the next one. The same occurs when data is sent.

Figure 6.3 shows the aforesaid process, where T_x represents next byte to be sent and R_x represents the last byte received.

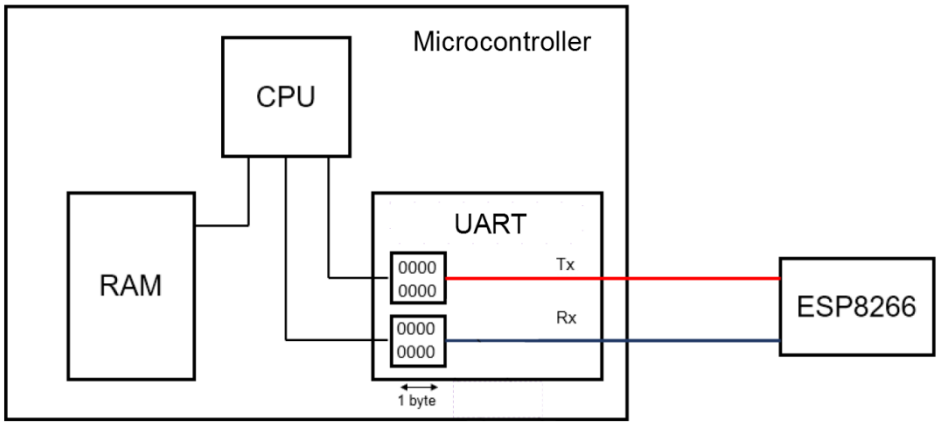

Fig. 6.3 Process of the package transmission.

6.3.2. Communication by AT commands

AT commands (abbreviation of ATtention commands) are instructions used to control a *modem*. They belong to the Hayes command set which is a command language consisting of a series of short text strings, in ASCII code, that produce operations such as dialling, hanging up, and changing the parameters of the connection.

In this project these commands are sent by a series communication channel. Each command makes a specific action, for instance: scan for available networks, connect to a network, prepare a package to be sent, etc. The WiFi Module ESP8266 process these commands sequentially, therefore it does not admit new commands until the current command has not been processed.

6.3.3. Internet Protocol and transport layer

The ESP8266 device uses *IEEE 802.11 b/g/n standard* for the wireless communication and an Internet Protocol (IP) to create the network. Each device participating in this network has a numerical label assigned that is called IP address. The IP address is defined as a 32-bit number and this system is known as Internet Protocol Version 4 (IPv4). Internal Protocol provides a communication between the network's clients using packages called datagrams. Each datagram has two components, a header that includes source and destination addresses, and a data payload that is the data to be transported.

The transport layer is a conceptual division of methods in the layered architecture of protocols in the network stack in the Internet Protocol. It allows communication between two devices of the network and it provides services such as connection-oriented data stream support, reliability, *flow control* and *multiplexing*. There are two transport layer protocols: Transmission Control Protocol (TCP) and User Datagram Protocol (UPD). First one uses a retransmission strategy to insure that data will not be lost in transmission, whereas the second one uses a simple connectionless transmission model with a minimum of protocol mechanism (it gives no guarantee of delivery, ordering, or duplicate protection).

In this project communications are done through a User Datagram Protocol (UDP) because in this way packages arrive immediately to the recipient. Using a Transmission Control Protocol involves a long time period of verification which delays the communication.

6.4. Router

It is made use of a *router* to create a wireless network and connect the vehicle with a computer. It is in charge of identifying the devices that will be the network's clients, in this case the vehicle and the computer, and allowing communication between them. The router is connected to two data lines (from the vehicle and from a computer), when a data packet comes in one of the lines, the router reads the address information in the packet to determine the ultimate destination following an IP protocol.

Fig. 6.4 Router employed.

It is also possible create a wireless network and a wireless access point with a computer. Name's network and password's network have to coincide with the connectivity parameters of the vehicle configuration.

For this project configuration the computer's IP is 192.168.173.10 and the vehicle's IP is 192.168.173.100.

7. Firmware and code's structure

This section has as an objective to explain which routine the microcontroller does while the vehicle is in operation. It is important to differentiate two main functions which are the data transmission (Communication System) and the vehicle's control (Motors control and Trajectory control). Being the second one more important than the first one as it will be shown next.

7.1. Communication routine

This routine is executed continuously, it is within an infinite loop of the microcontroller's code, but it has low priority so it can be interrupted anytime by the control routine. Below there are the main steps this routine follows when the WiFi Module is receiving or sending data:

a) Data reception

1. The data that the vehicle receives are commands sent by the computer's application. These packages are characterized by its first byte position that contains the character '>'. The rest is the order's name and in some cases a value preceded by an equal symbol '='. The las byte is always a line break '\n' to isolate different commands.
2. When data are received an interruption is generated and the CPU copies each byte of the UART peripheral's entrance register.
3. These bytes are copied into a buffer to be treated in an asynchronous way (independent of the arrival time).
4. The buffer is parsed and compared with the prearranged order's names. If it coincides with one, the corresponding instruction is carried out. For example if it receives '>S\r\n', the Boolean variable 'start' takes 1, which means the vehicle is started.
5. Right after, a confirmation message is sent back to confirm the instruction has been carried out. It appears on the command history windows of the application for data visualization.

b) Data delivery

1. The data that the vehicle sends are values from different variables such as wheel speeds or sensor's information. It sends it in an almost continuous way (in this project there is a 70 ms period of data delivery).
2. These packages are characterized by its first byte which is the 'E' character. Each variable is identified by the positions its bytes hold in the package. Also the microcontroller and the recipient have to know which positions the variables occupy into the package and which kind of data each one represents (float, enter, Boolean, etc).

3. When a package is build it is assigned into a buffer. Buffer's size can be changed according to the number of bytes that have to be sent, and it is made up of 1 byte elements.
4. The bytes of the buffer are moved to the UART's register to be sent. This copy is done by the CPU which moves bytes from the memory RAM to the UART peripheral after an interruption is generated. Byte packages are queued in an intermediate memory while the CPU is doing tasks of higher priority (the control routine), packages will be sent as soon as the CPU had finished these high priority tasks.

7.2. Control routine

Control routine is executed periodically every 50 µs with high priority. Its final purpose is driving the motors so the vehicle could follow the line at the desired speed. For that, it is required a process in which data's sensors are captured, these values are processed and the corresponding duty signals are used to drive the motors. Down below the main steps followed are exposed:

1. A first interruption is generated in the microcontroller by the Timer 1. It is used to capture the voltage signals of each photodetector and these ones are processed by a 12 bit Analog-to-digital converter. Then they are recorded into the microcontroller's memory.
2. A second interruption, scheduled right after the previous one, is generated.
3. A chronometer, consisting of a 32 bit unsigned enter, is started. Its functionality was explained in the Communication System section.

a) Data acquisition

4. The data recorded into the encoder's counter registers are processed. Some calculations are done to know wheel angular speeds. Also in this part it is detected if the vehicle is not moving.
5. The values coming from the ultrasonic sensor are taken. They give information about the presence of a near obstacle to decide if the vehicle must be stopped.
6. The two values of the line sensor are processed to know the distance of the vehicle to the line (as it was explained in the Trajectory control section). The detection range enhancement of the line sensor is implemented in this step.

b) Vehicle's stop check

7. The vehicle is stopped if the corresponding command is sent from the Application for data visualization or if the ultrasonic sensor captured values require it. Motors are stopped immediately by introducing a duty of 0 speed and all control parameters are reset.

c) Controller's calculation

8. The trajectory control establishes the angular velocity of the wheels depending on the distance measured, the cruising speed u_1 and the control signal u_2.
9. These angular velocity values are transformed with a feed forward relation (it can be seen at Motor control section) to make a first attempt to calculate the duty values associated.
10. The motors control corrects these duty signals using the measurements done by the encoders and by the implementation of a Proportional Integral controller.

d) Drive the motors

11. Finally timer 1 provides the pertinent signal to the Driver L298 which drives the motors.

In Figure 7.1 the control routine is shown.

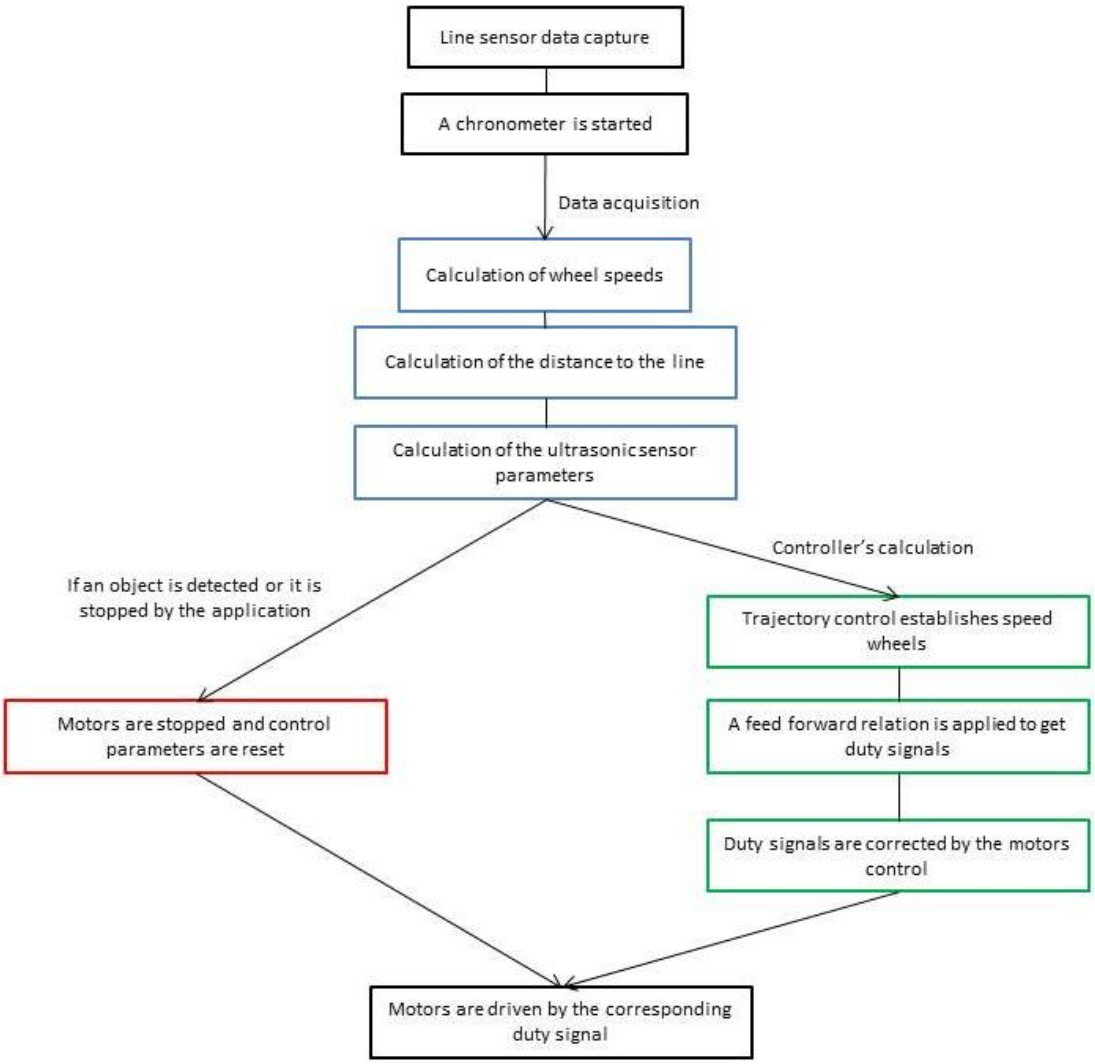

Fig. 7.1 Diagram of the control routine.

7.3. Execution time

The execution time of the control routine is an important fact to analyze due to it is executed periodically every 50 µs and if it goes beyond this time the control operation would not work properly. We need to make sure that the code execution in the control routine does not take longer than 50 µs.

In the current configuration the control routine takes 12 µs when the vehicle is stopped (the vehicle is only capturing and recording data from the sensors), and it takes 27 µs when it is running. So in the most critical case, there are 23 µs left to execute the communication routine.

For the future of the vehicle's development it is important to take care of the functions and operations that are implemented. Divisions and other complex calculations increase a lot the microcontroller's working time.

Figures 7.2 and 7.3 show a test carried out with an oscilloscope in which three signals were represented. The blue one is set when the first interruption is generated and it lasts until each photodetector has captured the voltage signals. The red one represents how much the control routine lasts. And finally the pink one represents the communication system and it becomes an impulse every time a package of information is sent.

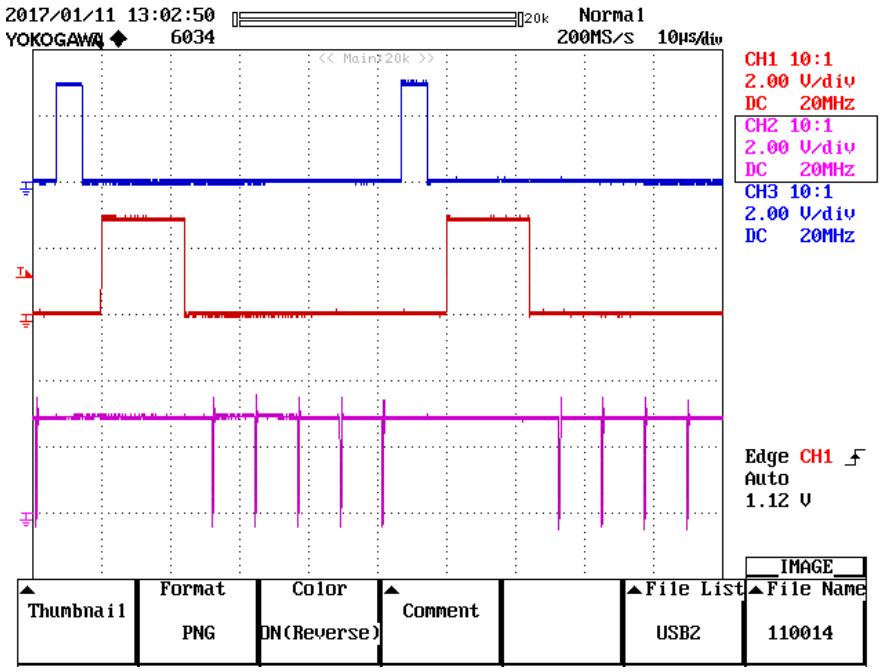

Fig. 7.2 Communication routine and control routine when the vehicle is stopped.

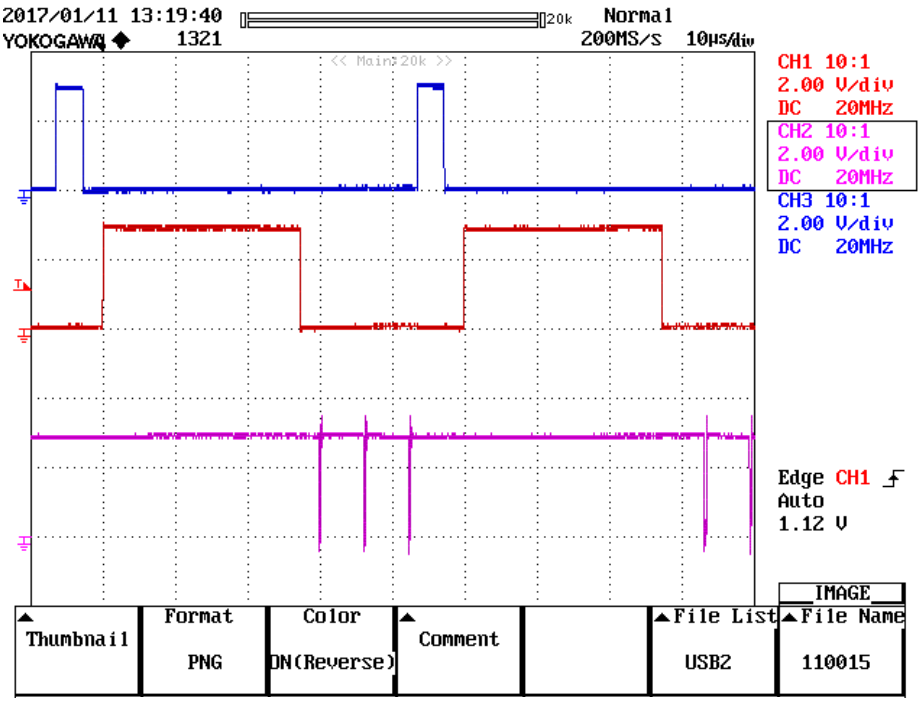

Fig. 7.3 Communication routine and control routine when the vehicle is running.

8. Experimental tests of the vehicle

A series of tests were performed in order to check the vehicle operation and to fit the control parameters of the vehicle. Two kinds of tests were run; the first trials were carried out to try the motors behavior and the latter to see how the vehicle responded following different tracks.

All the data that will be shown in this section has been obtained from the Communication System and the Application for data visualization. Graphs were created with the numerical computing environment MATLAB.

8.1. Motor tests

Tests were carried out to see the response time and the speed accuracy of the motors. Motors were set to run with different angular velocities. The computer application was employed to change motor angular speed and recording encoder's captured values for each wheel.

Methodology used:

- Both wheels were always above the floor, so the friction produce by the weight of the vehicle did not affect.
- The angular velocity of both wheels was changed at the same time and to the same value.
- Changes from one speed to another were done abruptly like a step input signal.
- It was enough time between speed changes to see the complete motor response.
- The proportional gain K_p of the trajectory control took different values and the integral gain K_i was always 10^{-3}.

In the next two pages some of these tests are shown. Results changing the proportional gain K_p are compared in the first two; meanwhile the last test is done for the chosen proportional gain. After them there are some notes about the results obtained.

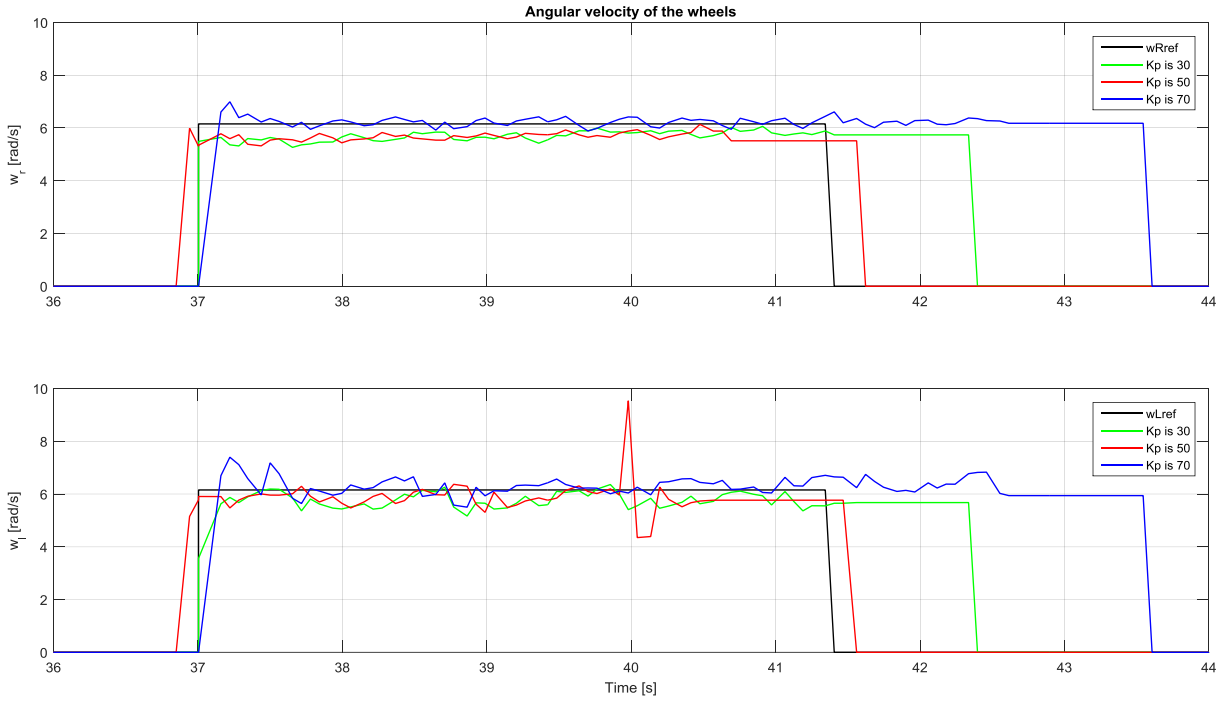

Fig. 8.1 Tests of the motors at 6,15 rad/s with Kp's of 30 (green line), 50 (red line) and 70 (blue line).

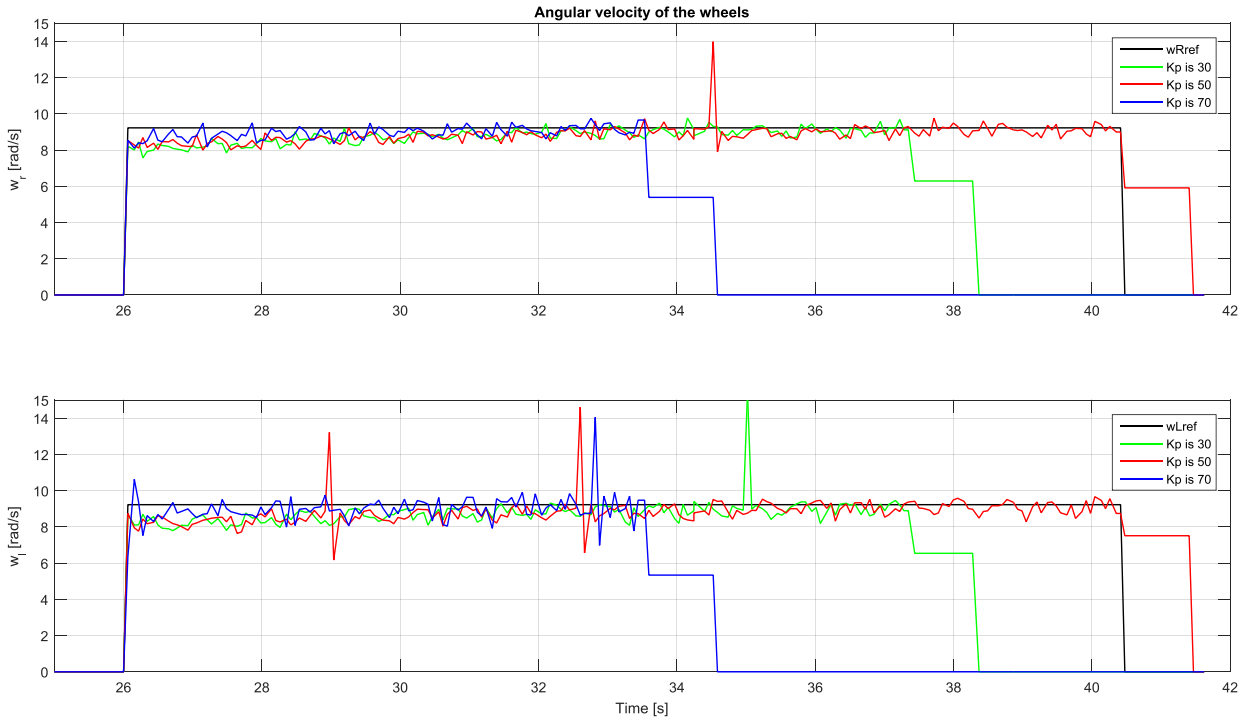

Fig. 8.2 Tests of the motors at 9,2 rad/s with Kp's of 30 (green line), 50 (red line) and 70 (blue line).

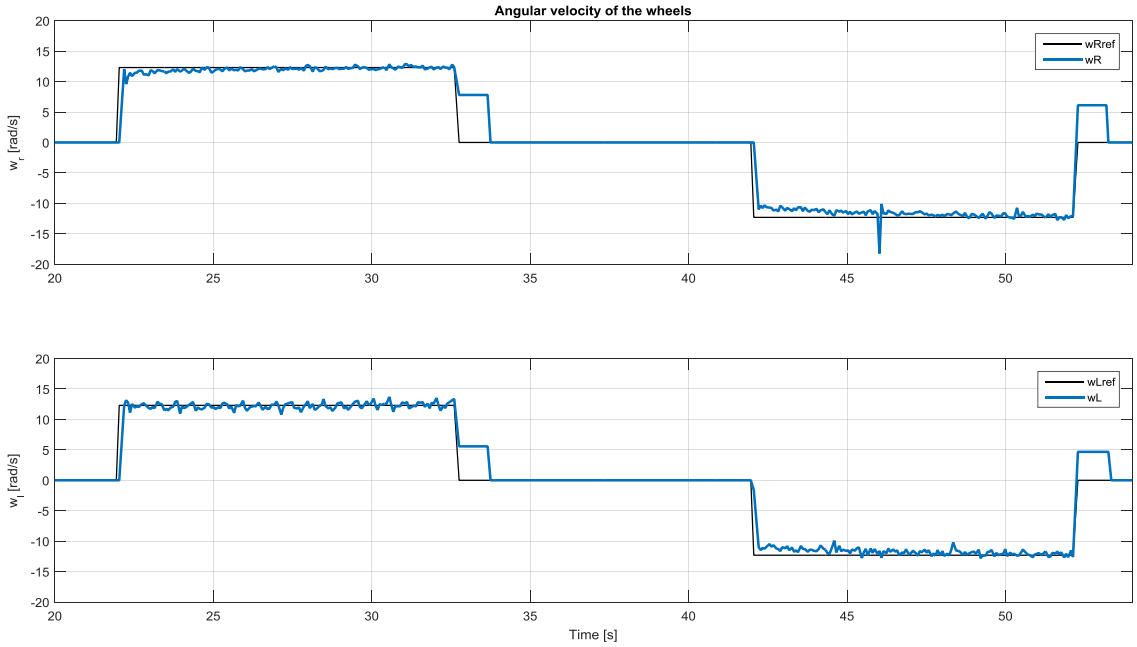

Fig. 8.3 Tests of the motors at 12,3 rad/s and -12,3 rad/s with a Kp of 70 (blue line).

Remarks of the results obtained

Figures 8.1 and 8.2 show clearly that the response corresponding with a $\mathrm{Kp}=70$ reaches the reference speed faster than the other two. It has a little overshoot but it probably won't affect to the vehicle's performance. In the three cases the oscillations are similar. Having said this, the chosen value of the proportional gain of the motors will be 70.

In the last test, shown in Figure 8.3, angular velocity of the wheels were changed from 0 to 12,3 rad/s and from 0 to -12,3 rad/s with the chosen K_p. As we can see the time response is quite fast and the accuracy obtained is also good. When the car is stopped we can see that the measurement captured by the encoder's is erroneous for 1 second (in the trial once 0 rad/s was assigned to be the angular velocity, wheels stopped rotating immediately). This fact was explained in the *Rotary encoders* section (*Stop detection* part).

8.2. Track tests

In these tests the vehicle run in three different tracks with the goal of testing the vehicle's behavior and finding the most suitable control parameters for each track (Figure 8.4). For these trials it was recorded the approximate distance of the vehicle to the line's symmetric axle all along the circuit.

As it was said in the section about the trajectory control, the line sensor does not measure a distance; it provides a value in proportion to how far the vehicle is from the line's symmetric axle. To get a distance from this value, it was done an empirical relation for the line's circuit. This relation is not perfect and also the line is not exactly the same in each circuit, so the distance value given by the sensor is just an approximate value.

The methodology used in these tests was:

- Every time a test was carried out, the vehicle did ten laps: five laps clockwise and other five in an anti-clockwise direction.
- Tests were done at two different cruising speeds u_1, at 0,15 m/s and at 0,20 m/s.
- Tests were done for different proportional gains K_p of the trajectory control. The integral gain K_i wasn't changed and it took a value of 10^{-5}.
- It was sought all the trials were performed with any irregularity or disturbance.
- The distance to the line symmetric axle was the factor taken into consideration to decide the best control's configuration. Also it was taken into account that line tracking was gentle and not quivering.
- The line was less or more 24 mm wide in each circuit.
- Tests were carried out in three different tracks: circuit A has a constant curvature all along the circuit (circuit A is a circumference); circuit B is formed by straight parts and circular parts; and finally circuit C has a non-constant curvature.

The circuits employed are shown in Figure 8.4.

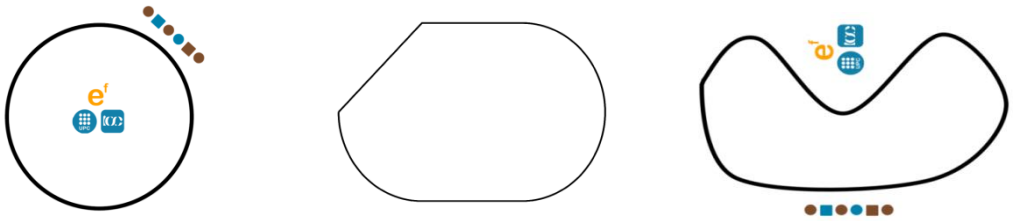

Fig. 8.4 Circuits employed: A, B and C.

In the following pages there are the results of some tests (at 0,20 m/s cruising speed u_1 and the vehicle tracking the circuit in an anti-clockwise direction) taken in the circuits aforementioned; results changing the proportional gain K_p are compared in each one. After them, there are some notes about the results obtained.

- Tests at 0,2 m/s (u_1) in the circuit A.

Fig. 8.5 Circuit A.

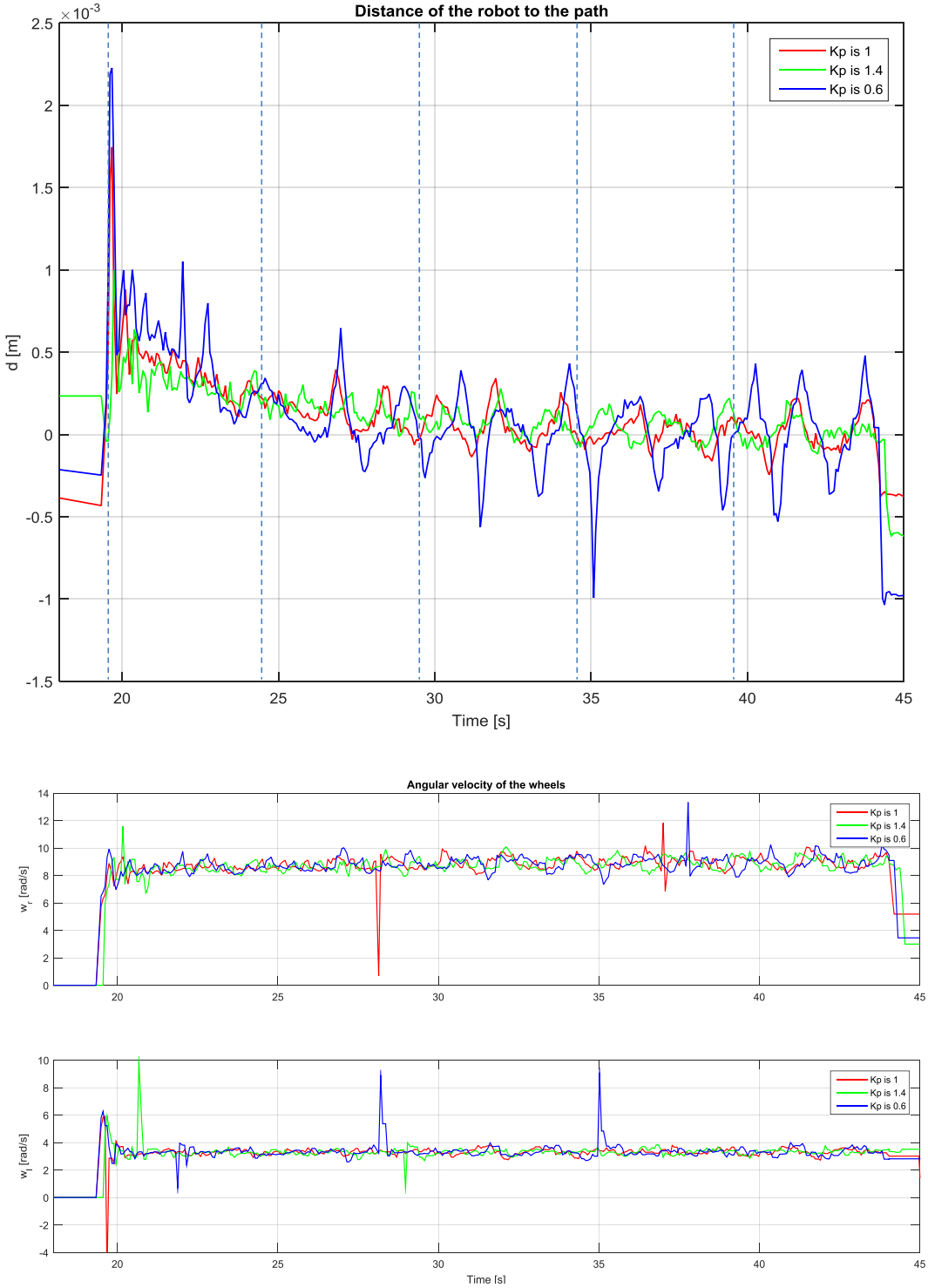

Fig. 8.6 Results of circuit A. Laps are divided by discontinuous blue lines.

- Tests at 0,2 m/s (u_1) in the circuit B.

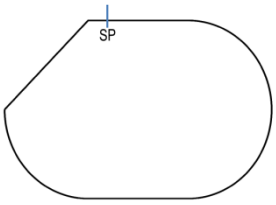

Fig. 8.7 Circuit B.

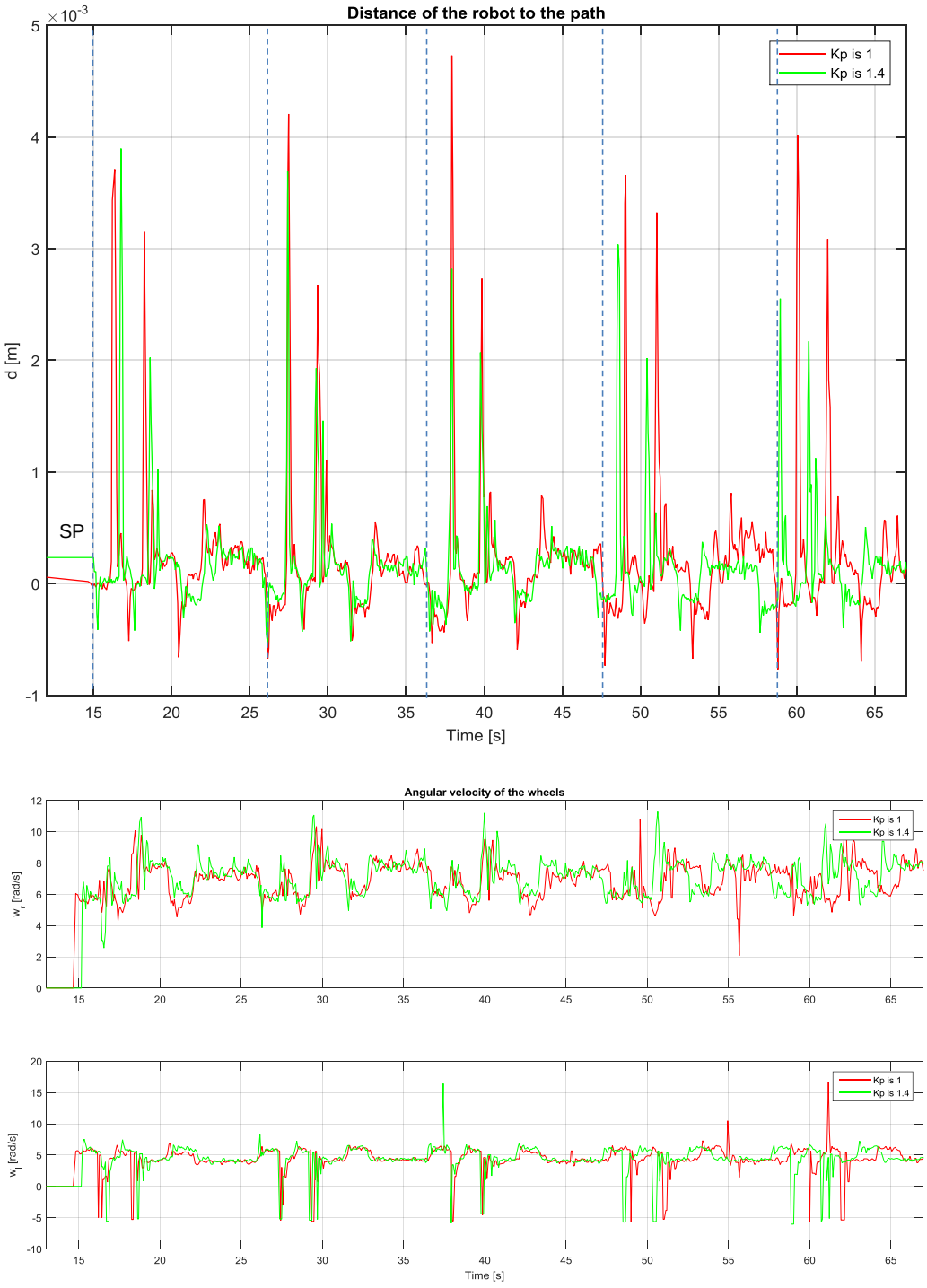

Fig. 8.8 Results of circuit B. Laps are divided by discontinuous blue lines. SP is the starting point where the vehicle started to run.

- Tests at 0,2 m/s (u_1) in the circuit C.

Fig 8.9 Circuit C.

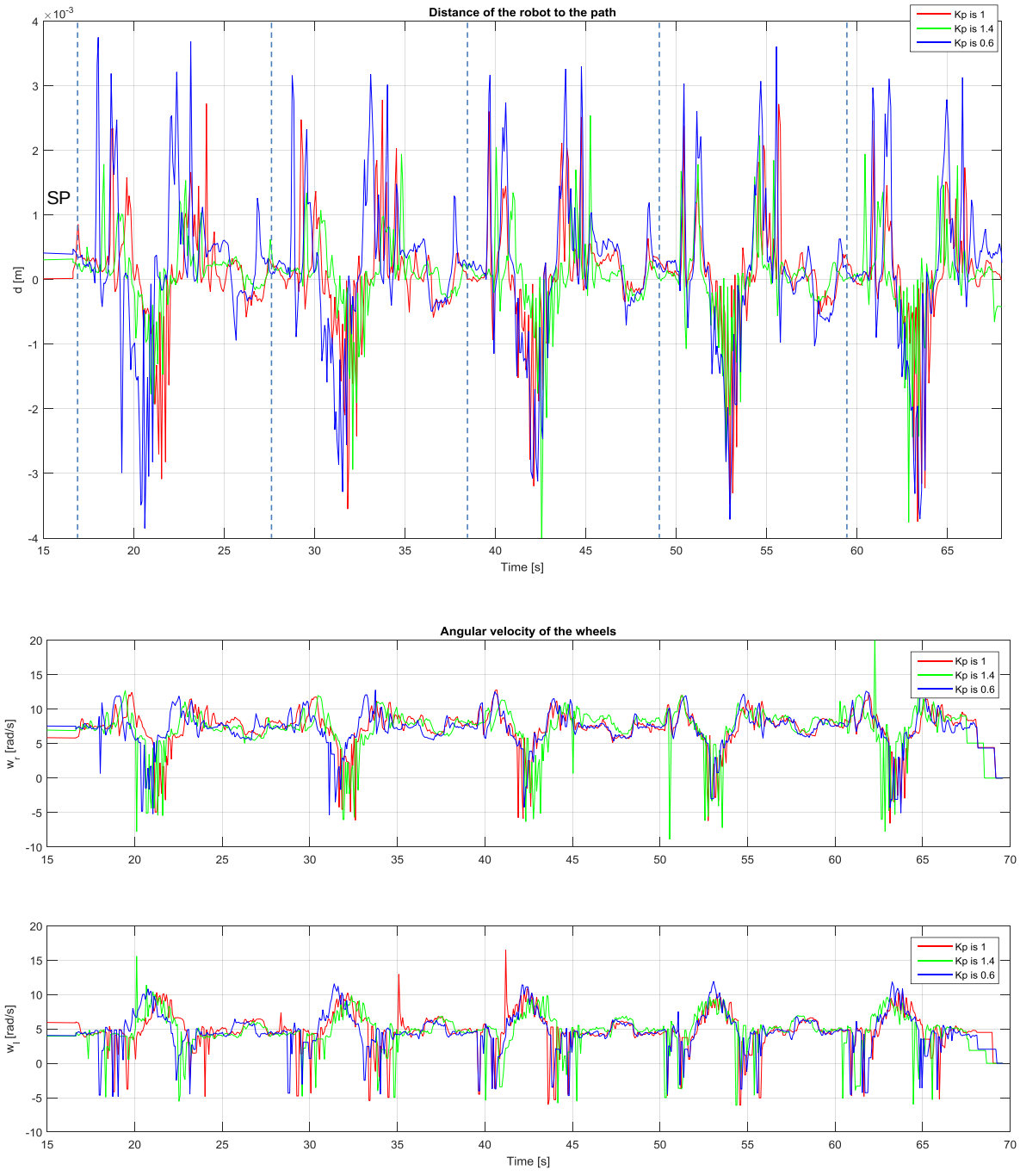

Fig 8.10 Results of circuit C. Laps are divided by discontinuous blue lines. SP is the starting point where the vehicle started to run.

Remarks of the results obtained in each track

Circuit A

In this circuit the curvature is always the same, so once the vehicle has been stabilized the distance to the line reaches a value around 0. That means that the Proportional Integral controller works properly since the error (the distance to the line) is removed. In addition, it can be seen that the lowest proportional gain ($K_p = 0{,}6$) produces the highest oscillations once the system is settled, meanwhile the other proportional gains cause a similar performance and these oscillations are smaller.

It is also interesting to mention that the vehicle lasts more or less 5 seconds (approximately 1 lap) to stabilize, it is said, distance value reaches 0 in this period of time. Regarding this last fact, once the vehicle is stabilized almost all the control signal is provided by the integral gain, so if the integral gain K_i is increased this setting time would be reduced.

Circuit B

Firstly it must be said that at a 0,2 m/s speed and with a proportional gain of 0,6 the vehicle couldn't follow the track when it reached the straight. At this point the proportional gain is not high enough to correct the car's trajectory.

If we look at the sections where the distance to the line reaches zero, this value is reached only in the straight line part (between the two sudden changes of trajectory). In the circular parts the vehicle does not have time enough to stabilize. This fact is understandable considering that in circuit A, which is perfectly circular, the vehicle lasts approximately 5 seconds to remove the distance to the line.

Finally if we compare the performance according to the proportional gain, it seems that the highest one produces less error in the sudden trajectory changes, although the difference is not really big.

Circuit C

The las circuit presents a curvilinear outline, the curvature is variable and it is only constant in short stretches. The vehicle is not stabilized in any circuit's section and the distance value is nearly never removed. The vehicle's control has been designed for tracks with constant curvatures, that's why this one is the most hard to follow.

Conclusions

All the objectives settled at the beginning of the projects have been achieved. The two previous projects have been joined together and the communication system was leveraged to improve the vehicle's control.

The PI controller is able to control the DC motors with a good behaviour; they reach the derived speed with accuracy and its response is fast. The wheel speed measurement system can measure low speeds and it can detect when the vehicle is stopped. However the motors cannot rotate in a continuos way when the arranged speed is too low. If in the furure it is required a wider working speed range, a current control motor must be implemented as a replacement of the voltage control. With a current control we could control the torque of the motors.

About the part addressed to the trajectory control, currently the photodetector can detect a wider distance from the line's symmetric axle than it did before. That means the control can act proportionally to tracks that present more sudden changes of trajectory. Although it captures a wrong value when the vehicle reaches the full white surface of the line's left side. It doesn't pose a problem but it could be corrected in the future to get a better performance.

The vehicle can follow a track gently and it doesn't shake as it used to do. Also it was found a good configuration of the control parameters once some experimental tests were set. In these experiments the good performance of the vehicle was demonstrated.

Regarding the communication system, it operates effectively with the vehicle and now the application for data visualization is more interactive and practical.

This project has bright future ahead with lots of possibilities to keep improving. In my opinion the next step would be assembling another vehicle and start to work with the communication between them.

Acknowledgements

I would like to say thanks to all the assistance and cooperation received from my tutors Arnau Dòria Cerezo and Víctor Repecho del Corral. They have spent a lot of time helping me to overcome the obstacles found during this project. Thanks them I have learnt a lot in different fields and I have really enjoyed the months while I have been developing the vehicle. In the same way I want to say thanks to the IOC's team for the assistance provided.

Also I would like to appreciate all the previous work done by the students Ivan Prats Martinho and Antoni Riera Seguí. They established the foundations of this project and did a really well job.

Finally I would like to express my gratitude for the support received from my family and friends, not just for this project but also during the degree.

Bibliography

Bibliography references

[1] Prats Matinho, Ivan. *Control design and implementation for a line tracker vehicle*. End of Degree Project of Physics Engineering (2016).

[2] Riera Seguí, Antoni. *Disseny i implementació d'un sistema de comunicacions WiFi per a una xarxa de vehicles autònoms.* End of Degree Project UPC (2016).

[3] Yulin Zhang, Daehie Hong, Jae H. Chung, and Steven A. Velinsky, *Dynamic Model Based Robust Tracking Control of a Differentially Steered Wheeled Mobile Robot,* Proc. of the American Control Conference, (1998)

[4] P. Morin and C. Samson. *Motion control of wheeled mobile robots*. In B. Siciliano and O. Khatib, editors. Handbooks of Robotics, chapter 34, pages 779–826. Springer, (2008).

[5] Prof. Yon-Ping Chen. *Dynamic System Simulation and Implementationon*. NCTU Department of Electrical and Computer Engineering (2015).

[6] Farreny Garcia, Alex. *Modeling and simulation of the transmission System for a chainless electric bicycle* End of Degree Project UPC (2014).

Annex

Code of the vehicle's control

```
#include <stdio.h>
#include "stm32f4xx_hal.h"
#include "stm32f4_discovery.h"
#include <stm32f4_hal/stm32f4xx_hal.h>
#include <comunicacions.h>
//inclouem la llibreria math per fer probes
#include <math.h>
/* Buffers comunicacio */

extern uint8_t paquetSortida[MIDA_MAXIMA_PAQUET];
extern BufferFIFO bufferEntradaUART;
uint8_t caracterRx;

uint8_t start = 0;



// **SENYALS DE REFERENCIA

__IO float wRref=0.0; // velocitats que volem tenir m/s ; wRref=2*u1-wLref
__IO float wLref=0.0; // velocitats que volem tenir m/s ; wLref=R*u2-u1

__IO float wRrefV=0.0;

__IO float wLrefV=0.0;
// **CONTROLADORS PROPORCIONAL I INTEGRAL
// senyals controladors
__IO float dutyRref; //duty en % va de 0 a 100
__IO float dutyLref;

__IO float errorR=0; // diferencia Wmeasured amb Wreferencia
__IO float errorL=0;

__IO float SenyIntR=0; // Senyal del control integral actuara a la right wheel
__IO float SenyIntL=0; // Senyal del control integral actuara a la left wheel

__IO float dutyRcorr; // duty en % va de 0 a 100 un cop aplicat senyal de
control
__IO float dutyLcorr;

// constants dels controladors

__IO float KpMotor=70; // control proporcional, per cada motor es igual, amb

__IO float KiMotor=1;
```

```
// **ENCODERS

//mesura

//__IO uint16_t uwFrequency[2];
__IO uint16_t uwIC2Value[2]={35000 ,35000}; // comptes fetes entre dos flancs del encoder (dos valors, un per a cada roda), inicialitzem en un numero molt gran per a considerar que esta parat
__IO uint16_t PrescalerEnc=1300; // amb presacaler de 1300 obtenim la maxima resolucio podem mesurar algo mes de 1 segon de temps enrtre flancs
//__IO uint32_t u=1;
__IO float wLmeasured;
__IO float wRmeasured;

__IO float wLmeasuredV;
__IO float wRmeasuredV;

// w=velocitatLineal = NumW/DenW = 2*r*pi*fCLKtim23 / 20*uwIC2Value*PrescalerEnc  ; 20 son els forats del encoder i 2*r es diametre roda 0,065m.
__IO float fCLKtim34=84; // 84[MHz] numero de comptes que pot fer el timer del encoder per segon
__IO float NumW=17153095.88; // NumW=2*r*pi*fCLKtim23, sera igual per el encoder de les dues rodes
__IO float DenWright;// DenW=20*uwIC2Value*prescalerEnc; es calculara per a cada interrupcio (aquest cas per roda dreta)
__IO float DenWleft; //roda esquerra

// mesura w0
__IO uint16_t contL=0; // comptador per indicar roda LEFT parada

__IO uint16_t contR=0; // comptador per indicar roda RIGHT parada
// **ACCIO ALS MOTORS

// Valors introduits als motros
__IO float PWMSpeedR;
__IO float PWMSpeedL;

// **SENSOR DETECTOR DE OBSTACLES
__IO float USValue = 0;
__IO float InvUSValue;
__IO float USFreq = 0;
__IO float USDutyCycle = 0;
__IO uint16_t USLimit=61; //61% of the duty at 1,315kHz =465us

__IO float USd = 0;
// **SENYALS DEL CONTROL DE LINIA**
__IO uint16_t uhADCxConvertedValue[2]; // aquesta variable es un nombre de 12 bits [0-4096] equivalent als Volts [0-3V] que trasnmeten els dos sensors (3V sobre blanc, 0v sobre negre)
__IO float Vdiff;
__IO float d_measured;
__IO float d_measuredV;

__IO float u1=0.2; //velocitat de creuer
```

```
__IO float u2; //velocitat de gir

__IO float errorD=0.0;
__IO float Kp_dist=1; // abans 0.3 pero en canviar la calibracio del KP aixo
canvia per linia de 14mm // per linia de 25 mm 0,6

__IO float senyPropD=0.0;
__IO float senyIntD=0.0;
__IO float Ki_dist=1; //Ki_dist=0.000001, (0.000001 VA BE)

// temps comunicacions
__IO uint32_t crono=0;

/* USER CODE END Includes */

/* Private variables ---------------------------------------------------------
--*/
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;
TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim4;
TIM_HandleTypeDef *htim30, *htim40;
TIM_HandleTypeDef htim10;
TIM_HandleTypeDef htim11;

TIM_HandleTypeDef htim12;
/* Private function prototypes -----------------------------------------------
--*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM1_Init(void);
static void MX_TIM3_Init(void);
static void MX_TIM4_Init(void);
static void MX_TIM10_Init(void);
static void MX_TIM11_Init(void);

static void MX_TIM12_Init(void);
int main(void)

{
  HAL_Init();
SystemClock_Config();

  /* Initialize all configured peripherals */

  BSP_LED_Init(LED3);
  BSP_LED_Init(LED4);
  BSP_LED_Init(LED5);

  BSP_LED_Init(LED6);
```

```
MX_GPIO_Init();
  MX_DMA_Init();
  MX_ADC1_Init();

  MX_TIM1_Init();
  MX_TIM3_Init();
  MX_TIM4_Init();
  MX_TIM10_Init();
  MX_TIM11_Init();
  MX_TIM12_Init();
  //
  HAL_TIM_PWM_Start_IT(&htim1,TIM_CHANNEL_1);
HAL_TIMEx_PWMN_Start(&htim1,TIM_CHANNEL_1);
  HAL_TIM_PWM_Start(&htim1,TIM_CHANNEL_2);
HAL_TIMEx_PWMN_Start(&htim1,TIM_CHANNEL_2);
  HAL_TIM_PWM_Start(&htim10,TIM_CHANNEL_1);
  /* USER CODE END 2 */
  //tocado para capture mode
  HAL_TIM_IC_Start(&htim11, TIM_CHANNEL_1);
  HAL_TIM_IC_Start(&htim12, TIM_CHANNEL_1);
  HAL_TIM_IC_Start(&htim12, TIM_CHANNEL_2);

  HAL_TIM_IC_Start(&htim3, TIM_CHANNEL_1);
  HAL_TIM_IC_Start(&htim4, TIM_CHANNEL_2);
  //fin tocado

  HAL_Delay(1000); //Esperar que arranqui l'esp8266 o saltaran errors del
port UART

  // **INICIALITZACIO DEL SOFTWARE DE COMUNICACIONS**
  BSP_LED_On(LED6);
  inicialitzarUart();
  rebreUartIT(&caracterRx, 1);
  configurarConnexioWifi();
  establirConnexio();
  HAL_Delay(100);
  inicialitzaBufferFIFO(&bufferEntradaUART); /* Una altra vegada */
  BSP_LED_Off(LED6);

  //HAL_GPIO_WritePin(GPIOB,GPIO_PIN_13,GPIO_PIN_RESET)


  // **INFINITE LOOP*/
while (1)
  {
        HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,GPIO_PIN_SET);
        COM_gestionarComunicacions();
        //enviarBufferUART();
        HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,GPIO_PIN_RESET);
  }

}
```

```
void SystemClock_Config(void)
{
       RCC_OscInitTypeDef RCC_OscInitStruct;
         RCC_ClkInitTypeDef RCC_ClkInitStruct;

         __PWR_CLK_ENABLE();

         __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

         RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
         RCC_OscInitStruct.HSEState = RCC_HSE_ON;
         RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
         RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
         RCC_OscInitStruct.PLL.PLLM = 4;
         RCC_OscInitStruct.PLL.PLLN = 168;
         RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
         RCC_OscInitStruct.PLL.PLLQ = 4;
         HAL_RCC_OscConfig(&RCC_OscInitStruct);

         RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK

|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
         RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
         RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
         RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
         RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
         HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5);

         HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

         HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

         /* SysTick_IRQn interrupt configuration */
         HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);

}
// *CODI CONTROLS */
// INTERRUPCIO,  Execucio cada 50 microsegons el ADC ja ha pactat valors i
disponibles uhADCxConvertedValue[] buffer dos posicions
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
       HAL_GPIO_WritePin(GPIOB,GPIO_PIN_11,GPIO_PIN_SET);
       //COM_gestionarComunicacions();
       // cronometre

       if (crono == 0xFFFFFFFF){ crono=0;}


       else{crono=crono+1;}
```

```
// **CAPTACIO DADES**            //

// * ENCODERS
       uwIC2Value[0]= HAL_TIM_ReadCapturedValue(&htim3,TIM_CHANNEL_1); //
encoder right wheel
       if (uwIC2Value[0] == 0){uwIC2Value[0]=30000;}
       DenWright=20.0*uwIC2Value[0]*PrescalerEnc;

       uwIC2Value[1]= HAL_TIM_ReadCapturedValue(&htim4,TIM_CHANNEL_2); //
ecoder left wheel
       if (uwIC2Value[1] == 0){uwIC2Value[1]=30000;}

       DenWleft=20.0*uwIC2Value[1]*PrescalerEnc;
// * MESURA W RODES, SENTIT GIR I DETECTOR W=0
       //RIGHT WHEEL
                     if (contR<20000) // ha passat 1 segon sense moviment a
R, si no detectem esta parat, calculem w de la roda R
                                    {
                                    contR=contR+1;
                                    wRmeasured =
(float)(NumW/DenWright);//(uwFrequency[0]*0.065*0.314159);
                                    if (PWMSpeedR<4200){
wRmeasured=wRmeasured*(-1.0);} //PROBLEMA amb Wref=0!!! PRESUPOSEM SIGNE DE
LA W SEGONS EL DUTY QUE LI ESTEM SUBMINISTRANT i no la Wref (ja que no podem
detectar sentit contrari de gir)
                                    }
                     else   {wRmeasured=0.0;}


                      wRmeasuredV=wRmeasured*10000;
//LEFT WHEEL
                     if (contL<20000) // ha passat 1 segon sense moviment a
L, si no detectem esta parat, calculem w de la roda L
                                    {
                                    contL=contL+1;
                                    wLmeasured = (float)(NumW/DenWleft);
//uwFrequency[1]*0.065*3.14159/10;
                                    if (PWMSpeedL<4200){
wLmeasured=wLmeasured*(-1);}
                                    }
                     else   {wLmeasured=0.0;} // detectem roda parada, aquesta
sera la mesurada

                     wLmeasuredV=wLmeasured*10000;
//FLANCS PER CADA RODA
                     //left wheel
                     if ((__HAL_TIM_GET_FLAG(&htim4, TIM_FLAG_TRIGGER ) == 1)
)
                                    {

                                    //HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,1);
                                    __HAL_TIM_CLEAR_FLAG(&htim4,
TIM_FLAG_TRIGGER );
                                    //HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,0);
                                    contL=0;
```

```
if ((__HAL_TIM_GET_FLAG(&htim3, TIM_FLAG_TRIGGER ) == 1) ) //cada vegada que
el encoder pasa per un flanc '__HAL_TIM_GET_FLAG()' es posa 1, despres
sempre el tornem a 0 per tornarlo a visualitzar en una altre pasada
                                {
                                                // TIM_FLAG_CC1 correspon al
SR-TIM_CCR1 pg 634 manual
                                //HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,1);
                                __HAL_TIM_CLEAR_FLAG(&htim3,
TIM_FLAG_TRIGGER ); //__HAL_TIM_CLEAR_FLAG(&htim4, TIM_FLAG_CC1 ); // posem
a 0 per software el flag que indicaba la rebuda o no dun flanc del encoder
                                //HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,0);
                                contR=0;

                                }                               }
// * DETECTOR OBSTACLE
      USValue = HAL_TIM_ReadCapturedValue(&htim12, TIM_CHANNEL_1); //
      InvUSValue= (float) (1 / USValue);
      USDutyCycle = HAL_TIM_ReadCapturedValue(&htim12, TIM_CHANNEL_2) * 100
* InvUSValue; //
      USFreq = HAL_RCC_GetHCLKFreq()*InvUSValue*0.0035714; //USFreq =
(float)(HAL_RCC_GetHCLKFreq())/2 / USValue/140;

      USd=(USDutyCycle/USFreq)*171.5;                                   //
(float)((USDutyCycle/USFreq)*0.01715*10000);
// DISTANCIA A LINIA

      Vdiff= (float)(uhADCxConvertedValue[0]-uhADCxConvertedValue[1]);

      //AUGMENT DEL RANG DE VISUALITZACIO DEL ADC LINIA 14 mm(TREC!)
      if ((uhADCxConvertedValue[0] >= uhADCxConvertedValue[1]) &&
(uhADCxConvertedValue[0] > 3000)){ //ZONA L
            //Vdiff=uhADCxConvertedValue[0]+uhADCxConvertedValue[1]-2000.0;

      }
      if ((uhADCxConvertedValue[1] >= uhADCxConvertedValue[0]) &&
(uhADCxConvertedValue[1] > 3200)){ //ZONA R
                  Vdiff=-
(uhADCxConvertedValue[0]+uhADCxConvertedValue[1])+2000.0;


      }
d_measured=(0.00173*Vdiff)+0.234; // nova relacio de distancia en milimitres
dist=0,0001684·Vdiff-0,1309 ABANS d_measured=(0.00173*Vdiff)+0.234;

      d_measuredV=d_measured*10000;
// **COMPROBEM SI PAREM COTXE**

if((start==0) || ((USd<7) && (USValue != 0))) // STOP; a traves de la
aplicacio, o objecte a distancia prixima, o sensor de proximitat no rep
senyal,
      {
      TIM1->CCR1=(uint16_t)(4200); //left wheel
      TIM1->CCR2=(uint16_t)(4200); //right wheel


      //*REINICIALITZEM (les constants dels controladors NO)
```

```
wRref=0.0;
       wLref=0.0;
       wRrefV=0.0;
       wLrefV=0.0;

       // No eliminem errors, vull que continui mesurant la señal
       //errorL=0.0;
       //errorR=0.0;

       //errorD=0.0;

       SenyIntR=0.0;
       SenyIntL=0.0;
       PWMSpeedR=4200;
       PWMSpeedL=4200;
       senyPropD=0.0;
       senyIntD=0.0;
       }

// **CALCULS I EXECUCIO**
else {

       //CONTROL DISTANCIA A LINIA PI
       errorD=-d_measured; // la distancia ideal de la linia es 0mm, mig
linia negre, ERROR=0.0-d_measured
       senyPropD=Kp_dist*errorD;
       senyIntD= senyIntD+(0.00001*Ki_dist*errorD);

       u2=senyPropD+senyIntD;

       //CALCUL VELOCITATS RODES
       wLref=0.15*u2+u1; //R=0.15m es distancia entre rodes en metres
       wRref=2*u1-wLref;

       wLrefV=10000*wLref;
       wRrefV=10000*wRref;


       //CONTROL RIGHT Wheel   // definim una zona de velocitat zero entre
0.05 i -0.05.
                           if (wRref>0.05){
                                  dutyRref= (wRref+1.6103)*46.729; //model
experimental planta
                                                        }
                           if (wRref<-0.05){
                                  dutyRref= (wRref+0.5888)*39.0625;
//dutyRref= (wRref+0.5888)/0.0256; AIXI EVITEM DIVISIONS
                                                       }
                           if ((wRref>=-0.05) && (wRref<=0.05)) {
                                  dutyRref= 50; //interval entre 0.05 i -0.05
m/s en que considerarem nula la velocitat, pararem el cotxe, no podrem
portar el cotxe a aquestes ws
                                  wRref=0.0;
        }
```

```
//CONTROL LEFT Wheel
                        if (wLref>0.05){
                              dutyLref= (wLref+1.6103)*46.729;  // model
experimental planta, a partir de consigna wLref dona el duty en %
                                                    }
                        if (wLref<-0.05){
                              dutyLref= (wLref+0.5888)*39.0625;
                                                    }
                        if ((wLref>=-0.05) && (wLref<=0.05)) {
                              dutyLref= 50; //interval entre 0.05 i -0.05
m/s en que considerarem nula la velocitat, pararem el cotxe, no podrem
portar el cotxe a aquestes ws
                              wLref=0.0;
       }

            // definim els errors dels dos motors
                  errorR= wRref-wRmeasured;
                  errorL= wLref-wLmeasured;

                  SenyIntR= SenyIntR+(errorR*(KiMotor)*0.001); // KiMotor
es 0,001
                  SenyIntL= SenyIntL+(errorL*(KiMotor)*0.001);

                  dutyRcorr= dutyRref+(errorR*(KpMotor)+SenyIntR);
                  // saturem la senyal de control
                  if (dutyRcorr > 100) {dutyRcorr=100;}
                  if (dutyRcorr < 0) {dutyRcorr=0;}
                  // pasem duty [0-100] a duty [0-8500]
                  PWMSpeedR=dutyRcorr*85;
//PWMSpeedR=(float)(dutyRcorr*(8500/100));

                  dutyLcorr= dutyLref+(errorL*(KpMotor)+SenyIntL);
                  // saturem la senyal de control
                  if (dutyLcorr > 100) {dutyLcorr=100;}
                  if (dutyLcorr < 0) {dutyLcorr=0;}
                  // pasem duty [0-100] a duty [0-8500]
                  PWMSpeedL=dutyLcorr*85;
//PWMSpeedL=(float)(dutyLcorr*(8500/100));


// **INTRODUIM VALORS als motors**

                  TIM1->CCR1=(uint16_t)(PWMSpeedL); //left wheel// TIM1-
>CCR1=(uint16_t)(PWMSpeedL)
                  TIM1->CCR2=(uint16_t)(PWMSpeedR); //right wheel// TIM1-
>CCR2=(uint16_t)(PWMSpeedR);


      }
HAL_GPIO_WritePin(GPIOB,GPIO_PIN_11,GPIO_PIN_RESET);

// *END CODI CONTROLS */
}
```

```
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *UartHandle)
{
  BSP_LED_Toggle(LED3);
}
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *UartHandle)
{
  BSP_LED_Toggle(LED4);
  escriuBufferFIFO(&bufferEntradaUART, &caracterRx, 1);
  rebreUartIT(&caracterRx, 1);
}
void HAL_UART_ErrorCallback(UART_HandleTypeDef *UartHandle)
{
  BSP_LED_On(LED5);
}

/* USER CODE END 4 */

#ifdef USE_FULL_ASSERT
**
   * @brief Reports the name of the source file and the source line number
   * where the assert_param error has occurred.
   * @param file: pointer to the source file name
   * @param line: assert_param error line source number
   * @retval None
   */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
/* User can add his own implementation to report the file name and line
number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
  /* USER CODE END 6 */

}

#endif
```

Code of the communications

```
#include "comunicacions.h"
#include "string.h"
#include <stdlib.h>
int8_t paquetSortida[MIDA_MAXIMA_PAQUET] = {0};

int entradaA, entradaB;

int temporitzador = 0;

/* Variables globals a transmetre o modificar */

//extern float u2;

// **SENSOR DE LINIA** i **MOTORS**
extern float wRrefV;
extern float wLrefV;

extern float wLmeasuredV;
extern float wRmeasuredV;

extern float d_measuredV;
// ARRENCADA
extern uint8_t start;

/* BUFFERS */

BufferFIFO bufferEntradaUART;
BufferFIFO bufferSortidaUART;
/* BUFFERS */

BufferFIFO bufferEntradaUART;
BufferFIFO bufferSortidaUART;

/* Definit al fitxer de capcelera */
int COM_gestionarComunicacions() {
	/* Executar aquesta funció contínuament però amb prioritat baixa. Pot
ser interrompuda en qualsevol moment.
	 * HAL_GetTick() retorna el valor del rellotge intern en milisegons
(contant a partir del HAL_Init() a l'engegada)	*/

	static uint32_t tickEnviamentDadesUART = 0;
	static uint32_t tickEnviamentEstat = 0;


	/* Tractar un element del buffer de dades rebudes */
while(tractarDadesRebudesUART() == 99);	/* Tractar dades. Retorna 99
quan el buffer conté paquets que no hem de tractar. */


	/* Enviar un element del buffer de comandes pendents per enviar. */
	if((HAL_GetTick() - tickEnviamentDadesUART) >
PERIODE_ENVIAMENT_DADES_UART) {
		enviarBufferUART();
		tickEnviamentDadesUART = HAL_GetTick();
	}
```

```
	}
if((HAL_GetTick() - tickEnviamentEstat) > PERIODE_ENVIAMENT_ESTAT) {
		posarCuaEnviamentEstat();
		tickEnviamentEstat = HAL_GetTick();
	}
return(0);
}
int posarCuaEnviamentEstat() {
	/* TODO nom funció provisional */

	// Per a enviar l estat, posem al buffer de sortida "+++ESTAT    \n"
amb la mida del paquet i acabat en \n.
	// Fem aixo perque al buffer nomes podem emmagatzemar caràcters.
	// Si emmagatzemem altres coses al buffer, un byte que fós 0x0A
(salt de línia) (p. ex. un enter amb el número 10) tallaria el paquet.

	uint8_t arrayEstat[MIDA_PAQUET_ESTAT] = "+++ESTAT";
	/* Se substituira per les variables d'estat en el moment precís de
l'enviament */
	arrayEstat[MIDA_PAQUET_ESTAT - 1] = '\n';
		//Acabem el paquet amb un salt de línia
	enviarASCII(&arrayEstat,MIDA_PAQUET_ESTAT,15);
	return(0);
}
int enviarEstatUart() {
	/* ***** Envia l'estat directament per UART. No posa el paquet en
cua. ***** */

	// Definir MIDA_PAQUET_ESTAT al fitxer comunicacions.h

	uint8_t dadesPaquet[MIDA_PAQUET_ESTAT] = {'E'}; // 'E' caràcter de
control (estat)

	// HAN DE COINCIDIR AMB ELS DEL fitxer parametre.txt, nom i ordre
	//int int1 = arrodonirFloat(wRrefV);
	// No tocar el byte 0. Conté un caràcter de control.
	sensor1=(float)(uhADCxConvertedValue[0]);
	sensor2=(float)(uhADCxConvertedValue[1]);
memcpy(&dadesPaquet[1], &wRrefV, sizeof(float)); // sizeof te da el numero
de bytes del tipo entre () --> 4
	memcpy(&dadesPaquet[5], &wRmeasuredV, sizeof(float));
	memcpy(&dadesPaquet[9], &wLrefV, sizeof(float));
	memcpy(&dadesPaquet[13], &wLmeasuredV, sizeof(float));
	memcpy(&dadesPaquet[17], &Vdiff, sizeof(float));
	memcpy(&dadesPaquet[21], &sensor1, sizeof(float));
	memcpy(&dadesPaquet[25], &crono, sizeof(uint32_t));


	// MIDA_PAQUET_ESTAT	40	maxim podem transmetre 40 bytes

	/* Aqui ja senvia directament el paquet. NO es posa en cua. (Ja
estava en cua!) */
	enviaUart(&dadesPaquet, MIDA_PAQUET_ESTAT);

	return(0);
}
```

```
int tractarDadesRebudesUART() {

	/* Obtenim una nova dada del buffer UART (fins el següent salt de
línia) */

	uint8_t comanda[MIDA_MAXIMA_ENTRADA] = {0};
	int mida = 0;
	if (llegirBufferFIFO(&bufferEntradaUART, &comanda, &mida) != 0){
		/* No hi ha noves comandes per llegir */
		return(1);
	}
if (comanda[0] == '+') {
		// Síntaxi d'un paquet de dades:
		// "+IPD,a,b:xxxxx" on a = id connexió, b = número de
caràcters, xxxxx bytes del paquet.
		// Cal utilitzar les comes i els dos punts per saber a partir
de quin caràcter llegir. (a i b tenen llargaria variable)
		int i = 1;
		while (comanda[i] != ':')
		{
			i++;
			if (i == 40) { //ESTAVA A 20, es perque es la mida
maxima del paquet?
				/* No són les dades que estavem esperant. */
				return(99);
					}
		}

		i++;
		int longitud = 0;

		/* Paquet contindrà els bytes que s'han enviat per TCP/UDP */
		uint8_t paquet[MIDA_MAXIMA_PAQUET] = {0};

		while( (longitud < MIDA_MAXIMA_PAQUET - 1) && (i <
MIDA_MAXIMA_ENTRADA)) { // -1 perque acabi en \0 (null)
			paquet[longitud] = comanda[i];
			longitud++;					/* TODO
canviar nom variable */
		    i++;
		}

		/* Funcio de parsing, desempaquetament i tractament de les
dades, al fitxer de funcions d'usuari */
		tractarPaquet(&paquet);
	}
	else {

		/* TODO Tractar dades connexió */
		return(99);
	}
	return(0);
}
```

```
Llegeix un paquet de xarxa, desempaqueta el seu contingut i efectua les
accions ne
 *  *paquet: Punter al primer element d'un array de bytes que contingui el
paquet.
 */
int tractarPaquet(uint8_t *paquet) {

	if(paquet[0] == 'E') {
		/* Paquet d'estat. Desempaquetar el contingut i assignar-ho
com a variable local. */

	}
	else if (paquet[0] == '>'){ // ORDRES
		char strComparar[MIDA_MAXIMA_PAQUET];

		memcpy(strComparar,">LED6\r\n", 7);
		if( !memcmp(paquet, strComparar, 7)) {

			BSP_LED_Toggle(LED6);

			char resposta[3] = ">K";
			enviarASCII(&resposta,2,15);
			return(0);
		}

		memcpy(strComparar,">S\r\n", 4); // CANVIAT 'START' PER S,
memcpy(strComparar,">S\r\n", 8);
		if( !memcmp(paquet, strComparar, 4)) {  // if(
!memcmp(paquet, strComparar, 8)) {

			start = 1;

			char resposta[3] = ">K";
			enviarASCII(&resposta,2,15);
			return(0);
		}

		memcpy(strComparar,">P\r\n", 4); //CANVIAT 'STOP' PER T, aixi
em de enviar menys bytes, memcpy(strComparar,">P\r\n", 7);
		if( !memcmp(paquet, strComparar, 4)) { //if( !memcmp(paquet,
strComparar, 7)) {

			start = 0;

			char resposta[3] = ">K";
			enviarASCII(&resposta,2,15);
			return(0);
		}
```

```
int inicialitzaBufferFIFO(BufferFIFO *buffer) {
	//Inicialitza un struct BufferFIFO.
	buffer->longitud = MIDA_BUFFER;
	buffer->pEntrada = 0;
	buffer->pSortida = 0;

	for(int i = 0; i < MIDA_BUFFER; i++) {
		buffer->dades[i] = 0;
	}

	return(0);
}
void incrementaPunterEntrada(BufferFIFO *buffer) {
	buffer->pEntrada++;
	if (buffer->pEntrada == buffer->longitud) {
		buffer->pEntrada = 0;
}
void incrementaPunterSortida(BufferFIFO *buffer) {
	buffer->pSortida++;
	if (buffer->pSortida == buffer->longitud) {
		buffer->pSortida = 0;
}
int escriuBufferFIFO(BufferFIFO *buffer, uint8_t *entrada, uint8_t mida) {
	/* Primer escriu el valor, i despres incrementa el punter. */
	for (int i=0; i < mida; i++) {

		/* Aquest if només és per quan hi ha overflow. Si no, mirar
més avall */
		if( (buffer->pEntrada + 1 == buffer->pSortida) || ( (buffer-
>pEntrada + 1 == buffer->longitud) && (buffer->pSortida == 0) ) ) {
int pSaltLinia = buffer->pSortida;

			while (buffer->dades[pSaltLinia] != '\n') {
				pSaltLinia++;
				if (pSaltLinia == buffer->longitud)
{pSaltLinia = 0;}   //Recorrer circularment
				if (pSaltLinia == buffer->pEntrada) {
					/* Error. El text que estem
intentant escriure ocupa mes que tot el buffer sencer.  */
					/* La comanda es tallarà i no es
podrà parsejar bé. Després el sistema es recupera. */
					inicialitzaBufferFIFO(buffer);
					return(99);
				}
				/* Si s'aconsegueix sortir del while,
pSaltLinia apunta al seguent '\n' */
			}
			/* Avancem el punter de lectura, descartant la
primera comanda que s'havia de processar */
			buffer->pSortida = pSaltLinia + 1;
			if (buffer->pSortida == buffer->longitud)
{buffer->pSortida = 0;}
		}
		buffer->dades[buffer->pEntrada] = entrada[i];
		incrementaPunterEntrada(buffer);
	}
```

```
int establirConnexio() {
	//ANTIC
	//Establir la connexio bloqueja durant uns segons el vehicle.
	//Cal activar les interrupcions per UART despres de fer-ho.

	HAL_Delay(1000);

	uint8_t comandaCipmux[] = "AT+CIPMUX=1\r\n";
	//TODO definir les comandes a part
	enviaUart(&comandaCipmux, COUNTOF(comandaCipmux) -1);

	HAL_Delay(1000);    //TODO Delay

	uint8_t comandaNetejarConnexioUDP[] = "AT+CIPCLOSE=0\r\n"; //TODO
	enviaUart(&comandaNetejarConnexioUDP,
COUNTOF(comandaNetejarConnexioUDP) -1);

	HAL_Delay(1000);

	uint8_t comandaConnexioUDP[] =
"AT+CIPSTART=0,\"UDP\",\"192.168.173.1\",6789,6789,2\r\n"; //TODO
	enviaUart(&comandaConnexioUDP, COUNTOF(comandaConnexioUDP) -1);

	HAL_Delay(1000);    //TODO Delay

	return(0); //TODO temporal
}
int enviaUart(uint8_t *pDadesTx, int mida) {
	/* Cada byte s'enviara per interrupcio. Cal copiar les dades a una
variable global. */
	memcpy(&paquetSortida, pDadesTx, mida);
	return(HAL_UART_Transmit_IT(&UartHandle, &paquetSortida, mida));
}

int rebreUartIT(uint8_t *pDadesRx, int mida) {
	//Nota: Retorna 0 (HAL_OK) o el codi d'error.
	return(HAL_UART_Receive_IT(&UartHandle, pDadesRx, mida));
}

int enviarBufferUART() {
	/* Enviar una comanda a l'esp8266. Hi ha d'haver una separació de xx
ms entre comandes successives*/

	uint8_t paquet[MIDA_MAXIMA_PAQUET] = {0};
	int mida;

	/* Obtenir la primera comanda de la cua. Si no retorna 0, no n'hi
ha. */
	if( llegirBufferFIFO(&bufferSortidaUART, &paquet, &mida) != 0) {
		/* No hi ha noves comandes per llegir */
		return(1);
	}

	/* Enviem la comanda a l'esp8266 pel port UART */

	if(paquet[0] == '+'){
		enviarEstatUart();
	}
```

```
	else {
		enviaUart(&paquet, mida);
	}
if(connexio == 15) {

		int midaComandaCipsend;
		if(mida<10) {
			midaComandaCipsend = 17;
		} else if(mida<100) {

			midaComandaCipsend = 18;
		} else {
			midaComandaCipsend = 19;
		}

		uint8_t comandaCipsend[midaComandaCipsend];
		sprintf(&comandaCipsend, "AT+CIPSEND=0,%d\r\n", mida);
		/* Posa en cua l'ordre CIPSEND i el paquet en sí
consecutivamenet */
		escriuBufferFIFO(&bufferSortidaUART, &comandaCipsend,
COUNTOF(comandaCipsend) -1 );
		escriuBufferFIFO(&bufferSortidaUART, pDadesTx, mida);
	}


	return(0);
}
```

Code of the application for data visualization

```
import pyqtgraph as pg
from pyqtgraph.Qt import QtCore, QtGui
import numpy as np
import time
import socket
import struct
import datetime
import baseInterficieGrafica

#Parametres
PERIODE_RECEPCIO_DADES = 25 # (1ms. es el minim possible AMB UNA
GRAFICA!!. amb el qual es poden rebre 1000 paquets/segon) amb tres
grafiques no ho soporta -->minim 15 (con 30 y resto sin tocar va
bien)
PERIODE_ACTUALITZACIO_GRAFIC = 0.20  # segons estava a 0.020
ESCALA_EIX_TEMPORAL_GRAFIC = 50    # En numero d'actualitzacions del
grafic. estava a 400, mes petit mes espaiat es veura el grafic

#Parametres xarxa
DIRECCIO_IP_ORDINADOR = "192.168.173.10" #si unsem el router ser de
10, sino de 1 al final
TIMEOUT_UDP = 0.10

PORT_UDP = 6789
variablesEntrada = []           # llista de variables d'entrada.
(Contingut: objectes tipus VariableEstatEntrada)
valorActual = ''                #String que contindra els bytes
rebuts en cada paquet de xarxa
tempsActualitzacioGrafic = 0    #Float que representa l'instant (en
segons d'execucio) de l'ultima actualitzacio del grafic (obtenir amb
time.clock())

variableDibuixar = 0            #Variable que s'esta dibuixant
actualment. El numero correspon a la posicio segons l'ordre del
fitxer parametres.txt

'''
Cada objecte creat amb aquesta classe representa una variable de les
que venen en un paquet d'estat.
El metode self.parsejarPaquetEstat(paquetEstat) actualitza el valor
de la variable, sent l'unic parametre el paquet d'estat complet (tots
els bytes)
'''
class VariableEstatEntrada(object): #classe que tindran les variables
un cop llegides del fitxer.txt
    def __init__(self, nom, tipus, posicioInicial, posicioFinal,
dividir = 1):
        self.nom = nom
        self.tipus = tipus
        self.posicioInical = posicioInicial
        self.posicioFinal = posicioFinal
        self.dividir = dividir
        self.valor = 0.0 # tocat era 0
        self.historialDades = [0] * ESCALA_EIX_TEMPORAL_GRAFIC
```

```
    def parsejarPaquetEstat(self, paquetEstat):
        try:
            if self.tipus <> 'zero':
                self.actualitzarValor( desempaquetaBytes(
paquetEstat[self.posicioInical:self.posicioFinal] , self.tipus,
self.dividir) )
            else:
                self.actualitzarValor(0)
        except:
            self.actualitzarValor(0)

    def actualitzarValor(self, nouValor):
        #Actualitzar el valor de la variable d'entrada
        self.valor = nouValor

    def actualitzarHistorialDades(self):
        #Actualitzar l'historial per a poder representar graficament
la variable.
        #No es fa en la mateixa crida a actualitzarValor(), perque la
arribada dels valors i la representacio grafica son asincrones
        np.roll(self.historialDades, 1)
        self.historialDades = np.roll(self.historialDades, -1)
        self.historialDades[-1] = self.valor  #abans
self.historialDades[-1] = self.valor*1000

    def valorActual(self):

        return self.valor
'''
Llegir el fitxer parametres.txt per a obtenir la sintaxi dels paquets
de xarxa
'''
def obtenirParametres():    #llegim wl fitxwr parametre.txt
    global variablesEntrada  #variablesEntrada es una llista!!!
    try:
        fitxerParametres = open('parametres.txt', 'r')
        for lc in fitxerParametres:    #lc: linia concatenada
            linia = lc.split()  #linia es una llista amb les paraules
que conte

            if linia == [] or linia[0][0] in '# \n':    #descartar les
linies que comencen amb '#'
                continue

            try:
                nomVariable = linia[0]
                posicioInicial = int(linia[1])
                posicioFinal = int(linia[2])
                tipusDades = linia[3]
                dividir = linia[4]

                #Creem un objecte que representa la variable d'entrada
(instancia de la classe VariableEstatEntrada) i l'afegim a la llista.
                objecteVariableEstatEntrada =
VariableEstatEntrada(nomVariable, tipusDades, posicioInicial,
posicioFinal, dividir) #emprem la classe definida al principi
```

```
    variablesEntrada.append(objecteVariableEstatEntrada)

            except:
                print "Error llegint el fitxer parametres.txt.
Linia:\r\n"
                print '    ' + lc

        fitxerParametres.close()
    except:
        print "No s'ha trobat el fitxer parametres.txt\n"
    if len(variablesEntrada) == 0:
        #Creem un objecte dummy ja que la llista no pot estar buida
        objecteVariableEstatEntradaBuit = VariableEstatEntrada("(Cap
variable d'estat)", 'zero', 0, 0, 1)
        print "No hi ha cap variable d'estat correctament definida al
fitxer parametres.txt\n"
        variablesEntrada.append(objecteVariableEstatEntradaBuit)

'''
Tractar els paquets de xarxa que han arribat a l'ordinador.
'''
def tractarDadesRebudes():
    try:
        paquetDades, addr = socket1.recvfrom(1024)   #Consultem al
sistema operatiu si te paquets de xarxa per a nosaltres
    except:
        #No hi havia cap paquet en cua
        return
    emissor = addr[0]

    if paquetDades[0]=='E':
        #Paquet d'estat
        desempaquetarEntradaEstat(paquetDades)
    elif paquetDades[0]=='>': # aixo es per quan enviem ordres
        #comanda
        tractarComandaRebuda(paquetDades, emissor)
'''
Desempaquetar un paquet d'estat.
Entrada es un string que conte el paquet (array de bytes).
Cada objecte VariableEstatEntrada de la llista variablesEntrada conte
la informacio necessaria per a extreure el seu valor del paquet
d'estat
S'utilitza el metode parsejarPaquetEstat de la classe
VariableEstatEntrada
'''
def desempaquetarEntradaEstat(entrada):
    global variablesEntrada
    #Desempaquetem la "entrada", que es un array de bytes, separant-la
en fragments i processant-los amb desempaquetaBytes(entrada, tipus,
multiplicador)

    for i in range(len(variablesEntrada)):
        variablesEntrada[i].parsejarPaquetEstat(entrada)
```

```
    '''
Tractar un paquet rebut, del tipus ordre.
El paquet es mostra a l'indicador d'ordres.
'''
def tractarComandaRebuda(paquetDades, emissor):
    comanda = paquetDades[1:] # aixi treiem el simbol >
#descartar el caracter de control
    form.listWidgetHistorialComandes.addItem(emissor + '> ' + comanda)
    form.listWidgetHistorialComandes.scrollToBottom()


'''
Enviar una comanda (paquet del tipus ordre), al destinatari indicat.
'''
def enviarComanda(comanda, destinatari):
    paquet = '>' + comanda
    try:
        paquet = bytes(paquet)  #Convertir el paquet a ASCII
        socket1.sendto(paquet, (destinatari, PORT_UDP))
        form.listWidgetHistorialComandes.addItem('@' + destinatari + '> '
+ comanda)
        form.listWidgetHistorialComandes.scrollToBottom()
    except:
        print 'Error enviant paquet. Comproveu caracters especials i
adreca IP.'
'''
Escriure el valor de cada variable d'estat al fitxer, en l'ordre en que
apareixen a parametres.txt, separats per comes
'''
def escriuVariablesEstatFitxer():

    if form.checkBoxEscriureFitxer.isChecked():
        f1.write(str(datetime.datetime.now()))
        for i in range(len(variablesEntrada)):
            f1.write(";%.2f" % variablesEntrada[i].valor)
        f1.write('\n')
''' DesempaquetaBytes
Entrada: String (probablement no imprimible) els bytes del qual continguin
un numero.
Tipus: 'h' signed short (2 bytes, usualment anomenat int en c), 'f' float
estandard (4 bytes)
Altres tipus: https://docs.python.org/2/library/struct.html
Multiplicador: Si s'havia multiplicat el nombre previament per algun
numero (p. ex. per enviar un nombre fixe de posicions decimals)
'''
def desempaquetaBytes(entrada, tipus, multiplicador=1):
    try:
        return struct.unpack(tipus, entrada)[0] #ABANS: return
struct.unpack(tipus, entrada)[0]/float(multiplicador)
    except:
        print "Error de desempaquetat: " + entrada + ' ' + tipus
        return 0
```

```
'''
Acabar de preparar la interficie grafica.
Afegir les variables a l'arbre de l'esquerra.
'''
def inicialitzarInterficieGraficaUsuari():
    for i in range(len(variablesEntrada)):
        QtGui.QTreeWidgetItem(form.llista1)    #Afegir una nova fila a
l'arbre de l'esquerra
        form.treeWidgetVariablesEstat.topLevelItem(0).child(i).setText(0,
_translate("MainWindow", variablesEntrada[i].nom, None))    #Nom de la
variable a la 1era columna
        form.treeWidgetVariablesEstat.topLevelItem(0).child(i).setText(1,
_translate("MainWindow", "0", None))
'''
Classe InterficieGrafica heredada. La base es troba al fitxer
baseInterficieGrafica.py
'''
class InterficieGrafica(QtGui.QMainWindow,
baseInterficieGrafica.Ui_MainWindow):
    def __init__(self, parent=None):
        super(InterficieGrafica, self).__init__(parent)
        self.setupUi(self)


        #Associar la funcio funcioDobleClickVariableEstat a un doble click a
l'arbre de variables d'estat

self.treeWidgetVariablesEstat.doubleClicked.connect(self.funcioDobleClickVar
iableEstat)

        #Associar altres events amb altres funcions

self.checkBoxEscalaAutomatica.stateChanged.connect(self.funcioCheckBoxEscala
Automatica)

self.checkBoxEscalaAutomatica2.stateChanged.connect(self.funcioCheckBoxEscal
aAutomatica2)

self.checkBoxActualitzarEscala.clicked.connect(self.funcioBotoActualitzarEsc
ala)    #Atencio, posa checkbox pero es un boto.

self.checkBoxActualitzarEscala2.clicked.connect(self.funcioBotoActualitzarEs
cala2)
        self.botoEnviar.clicked.connect(self.funcioEnviarComanda)

self.lineEditEnviarComanda.returnPressed.connect(self.funcioEnviarComanda)
def funcioDobleClickVariableEstat(self):
        #Aquesta funcio s'executa en fer doble click a qualsevol element de
l'arbre de variables d'estat.
        #Cal detectar quina s'ha premut exactament. form.llista1 es tota la
llista. S'han de recorrer tots els objectes subordinats a form.llista1
        for i in range(form.llista1.childCount()):
            if form.llista1.child(i).isSelected():
                seleccionarVariableDibuixar(i)
                break
```

```
def funcioCheckBoxEscalaAutomatica(self):
        if self.checkBoxEscalaAutomatica.isChecked():
            p1.enableAutoRange(axis=pg.ViewBox.YAxis)


        else:
            p1.disableAutoRange(axis=pg.ViewBox.YAxis)


    def funcioCheckBoxEscalaAutomatica2(self):
        if self.checkBoxEscalaAutomatica2.isChecked():

            p2.enableAutoRange(axis=pg.ViewBox.YAxis) #afegit
            p3.enableAutoRange(axis=pg.ViewBox.YAxis) #afegit

        else:

            p2.disableAutoRange(axis=pg.ViewBox.YAxis) #afegit
            p3.disableAutoRange(axis=pg.ViewBox.YAxis) #afegit

    def funcioBotoActualitzarEscala(self):
        try:
            self.checkBoxEscalaAutomatica.setChecked(False)
            YMin = int(self.lineEditYMin.text())
            YMax = int(self.lineEditYMax.text())
            p1.getViewBox().setYRange(YMin, YMax)

        except:
            print "No es pot canviar l'escala grafic 1 amb els valors
introduits."
            self.checkBoxEscalaAutomatica.setChecked(True)

    def funcioBotoActualitzarEscala2(self):
        try:
            self.checkBoxEscalaAutomatica2.setChecked(False)
            YMin2 = int(self.lineEditYMin2.text())
            YMax2 = int(self.lineEditYMax2.text())

            p2.getViewBox().setYRange(YMin2, YMax2)
            p3.getViewBox().setYRange(YMin2, YMax2)
        except:
            print "No es pot canviar l'escala grafic 2 i 3 amb els valors
introduits."
            self.checkBoxEscalaAutomatica2.setChecked(True)

    def funcioEnviarComanda(self):
        enviarComanda(self.lineEditEnviarComanda.text(),
self.lineEditDestinatari.text())
        self.lineEditEnviarComanda.clear()
```

```
'''
Funcio que fa replot del grafic. S'executa cada PERIODE_ACTUALITZACIO_GRAFIC
(20ms).
'''
def actualitzaGrafic():
    global curve1, llegendax
    global curve2
    global curve3

    #Escriure el valor de totes les variables en aquest instant
    #escriuVariablesEstatFitxer() REPETIT??-->SI..

    #Actualitzar l'historial de totes les variables.
    for i in range(len(variablesEntrada)):
        variablesEntrada[i].actualitzarHistorialDades()

    #Avancar el temps (eix x)
    llegendax += 1
    curve1.setPos(llegendax, 0)

    #Wref de les Rodes
    curve2.setPos(llegendax, 0)
    curve3.setPos(llegendax, 0)

    #Ws mesurades de les Rodes
    curve4.setPos(llegendax, 0)
    curve5.setPos(llegendax, 0)

    #Representar graficament el valor historic de la varible (eix y) en el temps
(eix x)
    curve1.setData(variablesEntrada[variableDibuixar].historialDades)

    curve2.setData(variablesEntrada[0].historialDades) #TOCADO/ DIBUIXAREM AQUI
DADES RIGHT WHEEL Wref (pos 0 de la llista)
curve2.setData(variablesEntrada[variableDibuixar].historialDades)
    curve3.setData(variablesEntrada[2].historialDades) #DIBUIXAREM AQUI DADES
LEFT Wref (pos2 de la llista) DONA PROBLWMWA

    curve4.setData(variablesEntrada[1].historialDades) #DIBUIXEM Wmes Right
wheel
    curve5.setData(variablesEntrada[3].historialDades) #DIBUIXEM Wmes Left Wheel


'''
def seleccionarVariableDibuixar(numeroVariable):
    #permet dibuixar als grafics les variables seleccionades

    global variableDibuixar
    variableDibuixar = numeroVariable           #Aquesta variable s'utilitza a
la funcio actualitzaGrafic()

    p1.setTitle(variablesEntrada[variableDibuixar].nom)
```

```
'''
Actualitza les etiquetes de l'arbre de l'esquerra. (Valors de totes les
variables)
'''
def actualitzaEtiquetes():

    for i in range(len(variablesEntrada)):
        form.treeWidgetVariablesEstat.topLevelItem(0).child(i).setText(1,
str(variablesEntrada[i].valorActual()))


def funcioPeriodica():
    global tempsActualitzacioGrafic
    #Aquesta funcio es crida cada PERIODE_RECEPCIO_DADES (1) ms. Es necessari
treballar amb un unic timer.

    #Consultem si hi ha noves dades i les desempaquetem. Si no n'hi ha, les
dades d'aquest periode seran les del periode anterior.
    tractarDadesRebudes()

    #Escrivim una nova fila al fitxer amb els valors actuals i actualitzem el
grafic, cada PERIODE_ACTUALITZACIO_GRAFIC (20 ms)
    if tempsActualitzacioGrafic < ( time.clock() +
PERIODE_ACTUALITZACIO_GRAFIC):
        tempsActualitzacioGrafic = time.clock()

        escriuVariablesEstatFitxer()
        actualitzaGrafic()
        actualitzaEtiquetes()
timerReplot = pg.QtCore.QTimer()

#Temporitzador que cridara la funcio periodica
def inicialitzarTemporitzadors():
    global timerReplot
    timerReplot.timeout.connect(funcioPeriodica)
    timerReplot.start(PERIODE_RECEPCIO_DADES)


_encoding = QtGui.QApplication.UnicodeUTF8
def _translate(context, text, disambig):
        return QtGui.QApplication.translate(context, text, disambig, _encoding)
'''
Inicialitzacio del programa
'''
if __name__ == '__main__':
    import sys
    if (sys.flags.interactive != 1) or not hasattr(QtCore, 'PYQT_VERSION'):

        #creacio socket UDP
        socket1 = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        try:
            socket1.bind((DIRECCIO_IP_ORDINADOR, PORT_UDP))
        except:
            raise Exception("\n\nError obrint un socket de xarxa. Probablement
no heu iniciat correctament l'adaptador virtual/router. Son les adreces IP i
port correctes? (revisar la capcalera de main.py) \n")
        socket1.settimeout(0.1)
```

```
        #Definim estil dels recuadres dels grafic
        #inicialitzacio grafic
        pg.setConfigOption('background', 'w')
        pg.setConfigOption('foreground', 'k')

        #definicio linies que dibuixem
        estilLinia1 = pg.mkPen(color='b', width=2)#definim linia 1,
        estilLinia2 = pg.mkPen(color='r', width=2)#definim linia 2,
        estilLinia3 = pg.mkPen(color='r', width=2)#definim linia 3,

        #per les Ws mesurada de les rodes
        estilLinia4 = pg.mkPen(color='g', width=2)#definim linia 2,
        estilLinia5 = pg.mkPen(color='g', width=2)#definim linia 3,

        #Obertura fitxer
        #no volem es crei excel
        nomFitxer = "Dades_Rebudes_"+time.strftime("%d%m%Y_%H%M%S")+".csv"
        f1 = open(nomFitxer,'w')
#Llegir el fitxer parametres.txt i crear els objectes variableEstatEntrada
(instancies de la classe VariableEstatEntrada)
        obtenirParametres()

        #Inicialitzacio interficie grafica
        app = QtGui.QApplication(sys.argv)
        form = InterficieGrafica()
        form.show()
        inicialitzarInterficieGraficaUsuari()    #adaptar la intericie segons el
fitxer parametres.txt
#GRAFIC 1
        #Inicialitzacio fienstra plot
        p1 = form.graphicsView_2.getPlotItem()
        p1.setTitle(variablesEntrada[variableDibuixar].nom)
        curve1 = p1.plot( [0]*ESCALA_EIX_TEMPORAL_GRAFIC, pen=estilLinia1)
        p1.setMouseEnabled(False,False)
        #p1.disableAutoRange(axis=pg.ViewBox.YAxis)
        p1.hideButtons()
        llegendax = 0
#GRAFIC 2 AFEGIT!! RIGHT WHEEL
        #Inicialitzacio fienstra plot
        p2 = form.graphicsView_3.getPlotItem()
        p2.setTitle("Right wheel data")
        curve2 = p2.plot( [0]*ESCALA_EIX_TEMPORAL_GRAFIC, pen=estilLinia2)
        curve4 = p2.plot( [0]*ESCALA_EIX_TEMPORAL_GRAFIC, pen=estilLinia4)
        p2.setMouseEnabled(False,False)
        #p1.disableAutoRange(axis=pg.ViewBox.YAxis)
        p2.hideButtons()
        llegendax = 0
#GRAFIC 3 AFEGIT!! LEFT WHEEL
        #Inicialitzacio fienstra plot
        p3 = form.graphicsView_4.getPlotItem()
        p3.setTitle("Left wheel data")
        curve3 = p3.plot( [0]*ESCALA_EIX_TEMPORAL_GRAFIC, pen=estilLinia3)
        curve5 = p3.plot( [0]*ESCALA_EIX_TEMPORAL_GRAFIC, pen=estilLinia5)
        p3.setMouseEnabled(False,False)
        #p1.disableAutoRange(axis=pg.ViewBox.YAxis)
        p3.hideButtons()
        llegendax = 0
```

```
#inicialitzacio del temporitzador
        inicialitzarTemporitzadors()

        #Escriure la capcalera del fitxer de sortida
        #
        f1.write("TIMESTAMP")
        for i in range(len(variablesEntrada)):
            f1.write(';' + variablesEntrada[i].nom)
        f1.write('\n')

        form.lineEdit_nomFitxer.setText(_translate("MainWindow", nomFitxer,
None))


        #Inicialitzar la aplicacio
        app.exec_()
```