Bachelor final project

Bachelor’s degree in Industrial Technology Engineering

Control design and implementation
of a twin robot

Author: Mireia Perna Vila
Director: Arnau Dòria Cerezo
Víctor Repecho del Corral
Convocatòria: June 2018

Escola Tècnica Superior
d’Enginyeria Industrial de Barcelona
SUMMARY

The purpose of this project is the creation of a controller for an autonomous robot. To achieve this, the accelerometer of the STM32F4-Discovery board will be used which will be coupled to the robot.

This project is an extension of previous projects referenced below which worked with the same robot implementing the following application:

- Design and implementation of a line tracking system for the robot.
- Design and implementation of a WiFi communication system for the robot.

The extension will be the design and implementation of a PD controller for the line tracking robot. The innovation will be the use of the acceleration data obtained directly from the accelerometer MEMS of the discovery board in order to design the derivative controller.

First step will be the theoretical design of the controller using the Routh-Hurwitz theorem and the following simulations using Matlab and Simulink, to prove the response of the robot that will be expected in the experimental tests.

Once the controller is designed, the discovery board will be programmed in order to implement the controller. To condition the data acquired for the accelerometer it is going to be designed a first order low-pass filter and an axis rotation system in order to use this data for the controller.

Finally, using a communication system and the line tracking system it is going to be proved the real response of the robot and the controller by experimental tests in a closed circuit.
CONTENTS

SUMMARY .......................................................................................................................... 3
CONTENTS .......................................................................................................................... 4
1. Introduction......................................................................................................................... 6
   1.1. Objectives and scope of the project .......................................................................... 6
2. Control design..................................................................................................................... 7
   2.1. Model ......................................................................................................................... 7
   2.2. Analysis and control tuning ....................................................................................... 8
      2.2.1. Stability .............................................................................................................. 10
      2.2.2. Pole location ..................................................................................................... 11
   2.3. Simulations ............................................................................................................... 12
      2.3.1. Implication of the parameters in the response ................................................ 13
      2.3.2. Tuning of $K_P$ and $K_D$ according to the system requirements ................ 16
3. Hardware and software description .................................................................................. 19
   3.1. STM32F4-Discovery Board ..................................................................................... 19
      3.1.1. Accelerometer LIS3DSH ................................................................................ 20
      3.1.2. User and reset buttons .................................................................................... 21
   3.2. WiFi access point and network router ........................................................................ 21
   3.3. Software .................................................................................................................... 22
      3.3.1. Eclipse and System Workbench (for STM32) ................................................ 22
      3.3.2. Viewers .............................................................................................................. 22
      3.3.2.1. STMStudio .................................................................................................. 22
      3.3.2.2. Python application - Python (x,y) ................................................................. 23
4. Signal conditioning............................................................................................................ 24
   4.1. Low-pass filter design .............................................................................................. 24
   4.2. Axes rotation system ............................................................................................... 28
5. Experimental tests ............................................................................................................ 31
CONCLUSIONS ..................................................................................................................... 33
BUDGET ................................................................................................................................. 34
1. Introduction

1.1. Objectives and scope of the project

The main objective of the project is to design and implement a PD controller in an autonomous vehicle using the development board STM32FA-Discovery. The idea is to use the acceleration data from the accelerometer MEMS as the input signal of the derivative controller.

In order to implement the controller in the board, because of the predetermined conditions of the accelerometer, values obtained directly are not correct since is affected by disturbances of the environment (noise), the offset on each axis and the orientation of the board respect the vehicle, as it could not be completely parallel to this giving incorrect values. Therefore, it is necessary to implement a low pass filter to remove noise signal and a rotation system for the axis so the z component is always aligned with gravity.

Once the controller is designed and the simulations has been done, the objective of the project is to compare the response of the system obtained in the simulations with the real response of the robot obtained in experimental tests.

In order to achieve it previous projects have been used from where it has been used the kinematic and dynamic studies of the system and the line tracking system from [1] in order to design the new controller and the experimental test, and from [2] the communication system in order to prove the response of the robot.

The final point of the work is the comparison of the response that is to be obtained according to the calculated parameters and the actual response, since the model of system used is a very simplified model that does not take into account frictions, parameters of the motor and others that alter the behaviour of the vehicle.
2. Control design

2.1. Model

As mentioned above, the model proposed is according to the line tracking study from [1] where a PI controller is used to perform this control. The kinematic model used is the following

\[ u_1 = \frac{r}{2} (\omega_R + \omega_L) \]  \hspace{1cm} (1)

\[ u_2 = \frac{r}{2} \cdot \frac{r}{R} (\omega_R - \omega_L) \]  \hspace{1cm} (2)

Where \( u_1 \) is the linear speed, \( u_2 \) is the angular speed of the vehicle and \( \omega_R \) and \( \omega_L \) are both the angular speed of the right (R) and left (L) wheels. The parameters \( r \) and \( R \) are from the vehicle design and presented in the following table

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>MEANING</th>
<th>MEASUREMENT [SI]</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R )</td>
<td>Distance between the wheels</td>
<td>0.17</td>
</tr>
<tr>
<td>( r )</td>
<td>Radius of the driving wheels</td>
<td>0.035</td>
</tr>
<tr>
<td>( l )</td>
<td>Distance from the axle to the tracking sensor</td>
<td>0.068</td>
</tr>
<tr>
<td>( l_o )</td>
<td>Distance from the axle to the accelerometer</td>
<td>0.09</td>
</tr>
</tbody>
</table>

TABLE 1 – VALUES OF VEHICLE PARAMETERS

Other variable parameters that must be taken into account are \( d \) which is the distance of the sensor to the line, \( \theta \) which is the angle formed by the axes of the vehicle with the real axes \( o'x'y' \), \( \sigma(q) \) which is the curvature of the line in each point and \( \theta_q \) which is the angle formed by the direction of the line and the real axes.

Figure 3 – Scheme of the vehicle and the line that follows
On the other hand, the points $P$, $B$, $A$ and $Q$ correspond respectively to the line tracking sensor, the middle point of the axis of the wheels, the accelerometer and the point of the line to which it is directed.

It should be taken into account that throughout the project it has worked with a linear speed $u_1$ of $0.18 \text{m/s}$.

As a two-wheeled vehicle is used, the rotation centre is located in the middle of the wheel axle so the kinematic model of it with respect to this point $B$ is given by the following system

$$\begin{align*}
\dot{x}' &= \cos(\theta) u_1 \\
\dot{y}' &= \sin(\theta) u_1 \\
\dot{\theta} &= -u_2
\end{align*}$$

(3)

After determining the desired trajectory followed by the vehicle and linearizing the system, calculated in [1] the following transfer function of the plant of the system is obtained

$$G(s) = \frac{D(s)}{U_2(s)} = \frac{ls + \nu \alpha}{s^2 + v^2 c^2}$$

(4)

Where $\alpha = \sqrt{1 - l^2 c^2}$, the input signal is the angular speed $u_2$ and the controlled parameter is the distance to the line $d$. The parameter $l$ is defined in the table above, $v$ is the linear speed $u_1$ and $c$ is the curvature defined as $\frac{1}{R_{\text{curvature}}}$, where $R_{\text{curvature}}$ is the radius of the curve of the path followed.

### 2.2. Analysis and control tuning

Parting from the system model from [1] showed as follow

![Figure 4 – Scheme of control of the trajectory taken from [1]](image)
The line tracking control is a PI controller that takes as the control parameter the distance between the vehicle sensor to the line of the circuit, so as it is considered the reference distance $d_{ref}$ equal to zero, the error signal $e_1$ corresponds to

$$ e_1 = d_{ref} - d_{measured} = -d_{measured} \quad (5) $$

As it is known that the derivative action gives a better response it is been decided to implement a PD controller, where the derivative signal will be the data take it directly from the accelerometer so a causality problem it is avoid. The proportional action will take as input signal the same as before.

It must be taken into account that we need speed data in order to implement the derivative part so as the accelerometer gives acceleration data, it is necessary to incorporate an integrator element.

The idea is that the speed in y axis $v_y$ (angular speed), given by the accelerometer data is transferable to the speed of the tracking sensor since the vehicle is working as a rigid as follows

$$ \dot{d} = - \frac{l}{l_a} v_y \quad (6) $$

![Figure 5 – Scheme of the kinematic relation (6)](image)

Where $l_a$ the distance between the accelerometer and the rotation is $B$, and $l$ is the distance between the rotation centre and the sensor of the line tacking. These values are determined in Table 1.

So as it is considered the reference value of it equal to zero $\dot{d}_{ref} = 0$, the error signal $e_2$ corresponds to

$$ e_2 = \dot{d}_{ref} - \dot{d} = -\dot{d} \quad (7) $$

So $u_2$ will be determined by the following signal

$$ u_2 = K_P \cdot e_1 + K_D \cdot \frac{de_2}{dt} = K_P \cdot (-d_{measured}) + K_D \cdot \frac{d(-d)}{dt} \quad (8) $$
Where $K_P$ is the proportional gain and $K_D$ the derivative gain.

Finally, the transfer function of the controller and the final model is the following

$$C(s) = K_P + K_D s$$

$$W(s) = \frac{C(s) \cdot G(s)}{1 + C(s) \cdot G(s)} = \frac{(K_P + K_D s)(l s + \nu \alpha)}{(s^2 + \nu^2 c^2) + (K_P + K_D s)(l s + \nu \alpha)}$$

(9)

(10)

And the bloc diagram of the system

![Bloc diagram of the control system](image)

**Figure 6 – Bloc diagram of the controller**

### 2.2.1. Stability

Next step is to determine the values of the gains $K_P$ and $K_D$ in order to impose a stable system. For this it is used the Routh-Hurwitz theorem to find the ranges of the gains that meet this need.

The characteristic polynomial is given by

$$Q(s) = 1 + C(s) \cdot G(s) = (1 + K_D l)s^2 + (K_P l + K_D \nu \alpha)s + (\nu^2 c^2 + K_P \nu \alpha)$$

$$= s^2 + \frac{(K_P l + K_D \nu \alpha)}{(1 + K_D l)} s + \frac{(\nu^2 c^2 + K_P \nu \alpha)}{(1 + K_D l)}$$

(11)

The parameters that accompany each grade are the ones needed to implement this method which ordered from higher to lower grade are $a_2 = 1$, $a_1 = \frac{(K_P l + K_D \nu \alpha)}{(1 + K_D l)}$, and $a_0 = \frac{(\nu^2 c^2 + K_P \nu \alpha)}{(1 + K_D l)}$. 

The theorem says that the number of sign changes between the coefficients of the first column corresponds to the number of positive roots of the system, so there must be no sign changes since for a system to be stable all roots must have a negative real part

\[
\begin{array}{c|cc}
  s^2 & a_2 & a_0 \\
  s^1 & a_1 & 0 \\
  s^0 & b_1 & b_2 \\
\end{array}
\]  

(12)

Where \( b_1 = -\frac{1}{a_1} \left| \begin{array}{cc} a_2 & a_0 \\ a_1 & 0 \end{array} \right| = a_0 \) and \( b_2 = -\frac{1}{a_1} \left| \begin{array}{cc} a_2 & 0 \\ a_1 & 0 \end{array} \right| = 0 \)

Imposing that parameters from the first column must be positive, the following relation is found

\[
K_p l + K_D v \alpha > 0 
\]

(13)

\[
l^2 c^2 + K_p v \alpha > 0 
\]

(14)

As it is known that the rest of parameters \( l, \alpha, v, c \) defined before are always positive, the following restrictions are the ones to determine the stability of the system

\[
K_p > -\frac{v c^2}{\alpha} 
\]

(15)

\[
K_D > l c^2 \alpha^2 
\]

(16)

In case of following a straight line defined by \( c = 0 \), the restriction is \( K_p > 0 \) and \( K_D > 0 \).

### 2.2.2. Pole location

To obtain a stable system the poles of it must have negative real part. Since the characteristic equation is of second order, there are two imaginary poles conjugated according to

\[
\begin{align*}
  p_1 &= -\sigma + \omega j \\
  p_2 &= -\sigma - \omega j
\end{align*}
\]

(17)

Where \( \sigma = \frac{4}{t_s} \) and \( \omega = \frac{-\pi \sigma}{\ln(M_p)} \)

The parameter \( t_s \) is the settling time defined as the time where the response reaches the 98% of its value and \( M_p \) is the maximum overshoot defined as the maximum peak value of the response measured from the desired response of the system. These parameters are imposed depending on the desired response of the system.
In order to determine the values of $K_p$ and $K_D$, as it is known that the condition of stability is determined by its poles, it is imposed a desired characteristic polynomial $Q'(s)$ as follows

$$Q'(s) = (s - p_1)(s - p_2) = s^2 + (-p_1 - p_2)s + (p_1p_2)$$ \hspace{1cm} (18)

Imposing $Q'(s) = Q(s)$ and equalizing the parameters that accompany each grade as follows

$$a_1 = -p_1 - p_2 = 2\sigma$$ \hspace{1cm} (19)
$$a_0 = p_1p_2 = \sigma^2 + \omega^2$$ \hspace{1cm} (20)

It is obtained a relation between the value of the gains and the parameters of the poles determined by

$$K_D = \frac{2\sigma - K_pl}{\nu\alpha - 2\sigma l}$$ \hspace{1cm} (21)
$$K_p = \frac{2\sigma l\gamma + \beta(1 - v^2c^2)}{\gamma l^2 + \beta\nu\alpha}$$ \hspace{1cm} (22)

Where $\gamma = \sigma^2 + \omega^2$ and $\beta = \nu\alpha - 2\sigma l$

With this relation it is possible to determine the optimum value of the gains while the stability condition is met.

It must be taken into account that these values of $K_p$ and $K_D$ are considering that there are no zeros (roots in the numerator) which is not the case. When doing the simulations with these values, a different response to the one imposed with $t_s$ and $M_p$ will be observed due to the influence of the zeros, so the gains will have to be adjusted in order to meet the desired requirements.

### 2.3. Simulations

In the following section different simulations are done in order to adjust the value of the gains of the controller, using Matlab and Simulink.

To model the plant of the system in Matlab it has been used the linearized form from [1], to make it easier to work with. This model has the following structure

$$G(s) = C(sI - A)^{-1}B$$ \hspace{1cm} (23)

Where $A = \frac{v}{a} \begin{pmatrix} -c^2 & -1 \\ c^2 & -1 \end{pmatrix}$, $B = \begin{pmatrix} l \\ -1 \end{pmatrix}$ and $C = (1 \quad 0)$
Once the plant and all the parameters are defined in Matlab with $K_P$ and $K_D$ as in (21) and (22), the system is modelled in Simulink adding visualizers in time, $u_2$ and $d$ response it is possible to graph these signals in order to study them.

The following is the Simulink model

![Simulink model of the system](image1)

**Figure 7 – Simulink model of the system**

![Simulink model of the controller](image2)

**Figure 8 – Simulink model of the controller**

![Simulink model of the plant](image3)

**Figure 9 – Simulink model of the plant**

### 2.3.1. Implication of the parameters in the response

In the first place, a study has been carried out to see how the variation of the variable parameters of the system affects the response obtained.

These are the curvature of the line followed $c$, the settling time $t_s$ and the overshoot $M_p$ of the system, but the settling time and the overshoot are parameters imposed at the beginning according to the conditions of the system and the proposed requirements, however the curvature can vary throughout the study depending on the path followed. Therefore to carry out the simulations different values of $t_s$ and $M_p$ have been imposed and from there it has been observed how the curvature affects the system.

The maximum value of the curvature that Matlab accept is 14.7, so the simulations performed took five different values of $c$ between 0 (straight line) and 14.
- $t_s = 0.5s$ and $M_p = 5\%$

Figure 10 – Simulation (left) and zoom of the stationary state (right)

- $t_s = 0.5s$ and $M_p = 80\%$

Figure 11 – Simulation (left) and zoom of the stationary state (right)

- $t_s = 2s$ and $M_p = 5\%$

Figure 12 – Simulation (left) and zoom of the stationary state (right)
- $t_s = 2s$ and $M_p = 80\%$

Figure 13 – Simulation (left) and zoom of the stationary state (right)

- $t_s = 5s$ and $M_p = 5\%$

Figure 14 – Simulation (left) and zoom of the stationary state (right)

- $t_s = 8s$ and $M_p = 5\%$

Figure 15 – Simulation (left) and zoom of the stationary state (right)
- \( t_s = 8s \) and \( M_p = 80\% \)

These simulations show that the effect of the curvature of the path followed it is more significant as the settling time and the overshoot increase, but as this parameter is an external factor it is not possible to control, so adjusting the values of \( t_s \) and \( M_p \) according to the requirements of the system, the design of the controller will be done.

### 2.3.2. Tuning of \( K_P \) and \( K_D \) according to the system requirements

To meet the requirements of the system it has been determined the maximum speed that can reach the wheels of the vehicle, since it is equivalent to the maximum power of the engine.

It has been measured with the charged batteries (5V) and when the vehicle is both in contact and not in contact with the ground.

In the following graphs it can be observed that there is not an important difference between the vehicle being in contact or in non-contact, but since the studies are carried out with the vehicle in contact with the ground, the maximum speed is sought in this state.

![Figure 16 – Simulation (left) and zoom of the stationary state (right)](image)

![Figure 17 – Wheels speed in contact with the ground](image)
It must be taken into account that when making the tests in contact with the ground, the data collection is limited to the path that the vehicle can follow. As the tests were done in the laboratory, the duration of the test is shorter because when encountering obstacles, the robot has a device that makes it stop before crashing, so at the end of each test there is a peak fall in speed.

The maximum values obtained in the contact test are 11.8 rad/s for left wheel and 12.39 rad/s for the right wheel.

Once the maximum speed that each wheel can reach has been found, it can be limited the range of angular speed \( u_2 \) in which the vehicle can work according to the kinematic relation of the system (1) and (2) as follows

\[
\begin{align*}
    u_2 &< \frac{1}{R} (r \omega_{R_{\text{max}}} - u_1) \\
    u_2 &> \frac{1}{R} (u_1 - r \omega_{L_{\text{max}}})
\end{align*}
\]  

(24)

The maximum speed values of each wheel are different but in motion both wheels must have the same speed to follow a straight line, therefore the angular speed range \( u_2 \) of the vehicle is determined taking the minimum between the maximum speeds measured as the maximum that can be achieved at the same time. So \( \omega_{R_{\text{max}}} = \omega_{L_{\text{max}}} = \min(11.8; 12.39) = 11.8 \text{ rad/s} \)

Knowing these restrictions, the appropriate value of the gains \( K_P \) and \( K_D \) can be determined by checking the response through simulations. The main point of these simulations is to verify that the maximum value of the response, which is the angular speed, is in the rank determined by

\[-1.3706 \text{ rad/s} < u_2 < 1.3706 \text{ rad/s}\]

With the previous simulations it has been possible to determine that the angular speed \( u_2 \) decreases as the overshoot decreases and increases the settling time.
In the following simulations it is imposed \( c = 0 \) and the maximum angular speed \( u_2 = 1.3706 \text{rad/s} \), so it is easy to see the behaviour of the response when varying the overshoot (between 0 and 90%) and the settling time (between 0 and 5s) by separate.

Testing different values, doing the relevant simulations and taking into account that the overshoot is usually between 5 and 10% and that it is not desired to have a high settling time, it has been proposed the values \( t_s = 3.6s \) and \( M_p = 5\% \).

With these parameters the maximum speed required by the wheels is within the determined range. Varying the parameters minimally it is possible to get a speed closer to the maximum imposed but considering that these changes are hundreds of seconds it has been worked with these values, obtaining the following values for the gains \( K_P = 24.4731 \) and \( K_D = 19.3172 \).
3. Hardware and software description

3.1. STM32F4-Discovery Board

It is a development board aimed at both beginner and experienced users that allow creating and developing applications through a high performance microcontroller.

It offers the following features:

- STM32F407VGT6 microcontroller featuring:
  - 32-bit ARM Cortex-M4 with FPU core
  - 1 Mb Flash memory
  - 192 Kb RAM in a LQFP100 package
  - 168 MHz clock rate
- ST-LINK embedded tool for programing and debugging
- USB ST-LINK with re-enumeration capability and three different interfaces: Virtual COM port, mass storage and debug port.
- Board power supply through USB cable or an external 3V-5V power supply.
- LIS3DSH ST MEMS 3-axis accelerometer.
- MP45DT02 ST MEMS audio sensor omni-directional digital microphone.
- CS43L22 audio DAC with integrated class D speaker driver.
- Eight LEDs:
  - LD1 (red/green) for USB communication.
  - LD2 (red) for 3.3 V power on.
  - Four user LEDs: LD3 (orange), LD4 (green), LD5 (red), LD6 (blue).
  - 2 USB OTG LEDs LD7 (green) VBUS and LD8 (red) over-current.
- Two push buttons: user and reset.
- USB OTG FS with micro-AB connector.
- Extension header for all LQFP100 I/Os for quick connection to prototyping board and easy probing.
- Comprehensive free software including a variety of examples, part of the STM32CubeF4 package or STSW-STM32068 for legacy standard library usage.

In this project the main peripheral used is the accelerometer, still others are used such as the push buttons and the LEDs to control the operation of the device and the turning on and off of the system, and others used in [1] and [2].
3.1.1. Accelerometer LIS3DSH

The accelerometer is the component LIS3DSH an ultra-low-power high-performance three-axis lineal accelerometer, which is able to pick up the acceleration in the three prefixed axis according to the Figure 22, and can be programmed to implement autonomous applications as motion-controlled user interface, gaming and virtual reality, pedometers, intelligent power saving for handheld devices, display orientation, click/double-click recognition, impact recognition and logging, vibration monitoring and compensation.

The features of the device are:

- Wide supply voltage 1.71V-3.6V
- Independent IOs supply (1.8V) and supply voltage compatible
- Ultra-low power consumption
- \( \pm 2g/\pm 4g/\pm 6g/\pm 8g/\pm 16g \) dynamically selectable full scale
- I²C/SPI digital output interface
- 16-bit data output
- Programmable embedded state machines
- Embedded temperature sensor
- Embedded self-test
- Embedded FIFO buffer
- 10000g high shock survivability
- ECOPACK, RoHS and ‘Green’ compliant

The units of measurement used are mg, where gravity is 1000mg. For this application the dynamically rank it is set to ±2g in the three axes, as it is the lowest rank possible and the vehicle is not able to reach these values.

It must be taken into account that during the project all the kinematic relations have been made defining that the linear speed \( u_1 \) is on the x axis, so at the moment of coupling the plate to the vehicle, this has to be oriented so that the x axis of the accelerometer is in the direction of rectilinear movement of the vehicle.

### 3.1.2. User and reset buttons

User button is connected to I/O PA0 pin of the microcontroller and reset button to NRST which function is to reset the microcontroller.

The user button can be given different uses according with the purpose. In this project it controls the start-up of the vehicle and the axes calibration system of the accelerometer which is explained later.

### 3.2. WiFi access point and network router

The WiFi control system comes from [2] which consisted of implementing a communication via WiFi between the vehicle and the computer. This communication allows carrying out tests with the vehicle without having to be connected by cable to the computer. The communication allows you to send orders from the computer and at the same time receive the desired information acquired by the board.

The WiFi module ESP8266 is used, which is directly coupled to the vehicle and allows WiFi connection. It is a low cost device that uses a serial port IP for sending data, and therefore can be implemented in many more devices. An important point is that the work of the device does not interfere with that of the microcontroller.

It is also necessary to use a WiFi access point to create the network to which the vehicles and the control computer will connect.

![Figure 23 – WiFi module ESP8266](image)
All this part is implemented and explained in [2], where the structure of the system is the following

![Diagram](image)

*Figure 24 – WiFi connections from [2]*

### 3.3. Software

#### 3.3.1. Eclipse and System Workbench (for STM32)

Eclipse is an integrated development environment of code that allows developing projects in different languages as C, C++, Python and others as long as the connectors needed for each language are installed.

System Workbench is one of the Eclipse environments that provide a software development, compile and debug tools for STM32 boards and allow creating an integrated project for the development board used.

About this software it has been able to implement the project using code written in C language the previous projects [1] and [2].

In the annexes is the entire code.

#### 3.3.2. Viewers

##### 3.3.2.1. STMStudio

This program allows viewing and recording the behaviour of the parameters desired on the screen in real time but with the disadvantage that it can only work with the vehicle connected by cable to the computer.
It is a useful tool when determining parameters that do not require the continuous movement of the vehicle as the noise detected by the accelerometer and the deviation of the axes thereof.

It is an easy software to use and understand. It consists of a manual [6] which explains step by step the installation and configuration of the software as well as the use, study and visualization of the variables. Also it is possible to change the behaviour of the variables in real time using mathematical expressions, but these changes are not made in the code, that is to say they are momentary variations to be able to do different tests.

3.3.2.2. Python application - Python (x,y)

Python(x, y) is a free scientific and engineering development software for numerical calculations, data analysis and visualization based on the Python programming language, Qt user graphical interfaces and the interactive scientific development environment Spyder.

In [2] was already implemented a python application that allowed visualization of parameters by WiFi. This application facilitates the study of the parameters at a more advanced level, since it allows receiving data of the vehicle when it is in movement without needing to be connected to the computer, as well as to send orders to him. It is a very versatile application since it can be modified according to the needs as adding other parameters or graphics modifying the code.
4. Signal conditioning

As previously mentioned, to implement the derivative controller the signal emitted by the accelerator is used directly, so it must be conditioned in order to obtain good results.

For this, a low-pass filter has been designed to avoid possible spikes and erroneous values, and also an axes rotation system to always obtain the desired signal.

4.1. Low-pass filter design

The response will just be better as long as there is no noise signal, in case of having noise the derivative controller will amplify it, that is why a first order filter has been implemented.

As discussed above, the data obtained by the accelerometer come from variations of its environment, therefore not only detects acceleration changes on the board, which is the useful information, but that part of the data obtained are erroneous.

The signal known as the noise signal is the one that gives these erroneous data. In the electrical and electronic field, noise is considered to all electrical disturbances that interfere with the transmitted or processed signals. An irregular signal complicates the implementation of many operations and on the other hand, being in motion this can give amplified values.

The magnitude of the noise that detects the accelerometer can be checked by taking data for a certain time, keeping it as quiet as possible. Then the value of this signal is shown graphically for the three axes.

![Accelerometer data without filter](image)

*Figure 25 – Accelerometer data without filter*
The accelerometer works in \( mg \) so that at rest only the force of gravity appears on the \( z \) axis around 1000\( mg \) and in sight, it can be seen that the value of this signal does not seem significant. Knowing that the measurement range is between \( \pm 2000mg \) and the noise ranges in a range of 30\( mg \) the signal, it can be analytically verified that it only represents 0.75% of the signal. This may mean that the noise is due to the sampling time of the accelerometer itself.

To obtain a cleaner signal, a discrete low-pass filter is implemented, which eliminates the highest frequency signals. These types of filters have the following iterative form

\[
G(s) = \frac{U(s)}{V(s)} = \frac{\omega_0}{s + \omega_0} \tag{25}
\]

\[
(s + \omega_0) \cdot U(s) = \omega_0 \cdot V(s) \tag{26}
\]

Where \( U(s) \) is the output signal and \( V(s) \) the input signal of the parameter is desired to filter, in this case will be the acceleration.

Applying the inverse transform of Laplace the continuous form in time is obtained as shown below

\[
\frac{du}{dt} + \omega_0 \cdot u(t) = \omega_0 \cdot v(t) \tag{24}
\]

The device does not work continuously, that is, it does not collect data continuously but rather it does it every so often. For this reason, it is necessary to go from continuous time to discrete time applying the definition of the derivative, with which the following model is obtained

\[
\frac{u_{k+1} - u_k}{T_S} = \omega_0 \cdot [v_k - u_k] \tag{28}
\]

\[
u_{k+1} = u_k \cdot [1 - T_S \cdot \omega_0] + v_k \cdot T_S \cdot \omega_0
\]

In it, the parameter \( T_S \) is the signal period of the accelerometer, known as the sampling time, which is the time between each data collection and \( \omega_0 \) is the cut-off frequency so, therefore the maximum frequency that the filter will allow to pass.

The noise signal does not follow any regular model from which its period can be obtained directly, then a series of possible periods are approximated graphically with which the means will be made to determine this value. It is determined to the three axes, a frequency of noise between 8 and 10 Hz, therefore \( \omega \) is around 50rad/s.

In order to determine the cut-off frequency of the filter, a spectral analysis must be carried out with which the highest frequencies that are to be eliminated by the filter are obtained.

The sampling time of the accelerometer is 1ms but in order to pick up the data, one of the two viewers must be used that have another sampling time. This means that the data obtained finally
would have been sampled twice at different times, which is not correct and although a sampling was imposed on the viewers equal to that of the accelerometer, they would not have to be synchronous so it would continue to be incorrect.

Consequently, a more experimental method has been used to make a sampling of the data using an oscilloscope and a Dac at a lower speed (50μs) than that of sampling (1ms). The answer obtained for the x axis noise at rest is as follows

The signal amplitude (y axis) is somewhat above 100mV, a very small value compared to the measurement range of the accelerometer which is between ±2V. The time division (x axis) is 1ms where a peak corresponding to the accelerometer data jack is observed. With this, it can already be verified that the noise comes from the accelerometer itself.

A FFT spectral analysis has been done with the oscilloscope trying to produce a sinusoidal signal (this signal has been done manually so it is an approximation since it does not follow a correct sinew form).
The bottom screen shows the two frequency spectrum of the signal centred in the middle of the screen and with a frequency division (x axis) of 2.5kHz. The centre peak is due to the continuous signal and as it has been supposed previously, there is no significant noise signal, not even the sampling one that is 1kHz (sampling of 1ms). All other signals are due to white noise and that the signal is an approximation.

With all this, it has been proposed to also implement the filter to work with cleaner signals and eliminate the small noise set around 50Hz. A cut-off frequency of 10rad/s has been proposed and as it is known that the data collection period $T_s$ is 1ms, the algebraic shape of the filter is as follows

\[
x_{k+1} = (1 - 0.01) \cdot x_k + 0.01 \cdot x
\]
\[
y_{k+1} = (1 - 0.01) \cdot y_k + 0.01 \cdot y
\]
\[
z_{k+1} = (1 - 0.01) \cdot z_k + 0.01 \cdot z
\]  

(29)

Where $x$, $y$ and $z$ are the values obtained by the accelerometer and the values with subscript $k$ and $k+1$ are the previous and current filtered values.

An important addition is that for the data obtained in the $y$ and $z$ axis of the accelerometer the same previous study has been done, obtaining very similar results to those of the $x$ axis, for that reason the same filter has been implemented to the three axes.

Once the filter is applied to the oscilloscope, a small reduction in the amplitude of the signal is observed.

![Figure 28– Comparaison of the response with the Dac without using filter (left) and with it (right)](image)

Although it seems a change almost non-existent, below is shown by the viewer the improvement of the response to each axis when it is at rest and when a movement is made wherever $x$, $y$ and $z$ are the unfiltered signals and $x_{k+1}$, $y_{k+1}$ and $z_{k+1}$ are the leaked signals.
4.2. Axes rotation system

For the study carried out, it is necessary that the x and y axes are always parallel to the ground, or in other words, that the axis z is always perpendicular to the ground. Therefore, as it is assumed that the vehicle will only move parallel to the ground (without changes of height) the z acceleration component must constantly show the action of gravity (1g), the component y the acceleration rectilinear and the component x will appear when rotating (it also appears if there is movement perpendicular to the axis and but it is not taken into account since the vehicle cannot make this movement).
As the accelerometer itself has determined the orientation of the axes, as shown in section 3.1.1, when coupling the board to the vehicle due to irregularities in contact and ground surfaces, these will not be well-oriented. To solve this, a rotation of the axes is carried out.

Below is how the real axis and the accelerometer axis would be available when the board is oriented in different ways.

![Figure 31 – Orientation of the accelerometer axes and the real axes](image)

The rotation system used is very common in navigation systems and is based on the rotation independent roll, pitch and yaw of the axes x, y and z respectively. This system determines that the orientation of an object with respect to the axes of the earth, can be described according to a rotation of each axis x, y and z.

In this way, once the board is attached to the vehicle and in a state of rest, it can be determined which deviation have the axes of the accelerometer relative to those of the earth.

The rotation on the z axis is not considered since the imposed condition is that this axis is linear with gravity, and it is assumed that the orientation of the x and y axis once coupled the plate to the vehicle, is correct. The reference of the rotation system is therefore the component of gravity since it is always constant.

In the first place the angles of rotation with respect to axis x and y are determined when the vehicle is at rest, since thus all the acceleration that detects the accelerometer \((x_g, y_g, z_g)\) in any of the three axes will be part of the gravity.
\[ g = \sqrt{x_g^2 + y_g^2 + z_g^2} \]

\[ \phi = \tan^{-1}\left(\frac{y_g}{x_g} \right) \]

\[ \theta = \tan^{-1}\left(\frac{x_g}{\sqrt{y_g^2 + z_g^2}} \right) \]

\[ \psi = 0 \]

Once these parameters have been determined, the change of axes can be carried out. It has been taken into account that in motion, the accelerometer detects both the acceleration, which is the part that is wanted to be obtained, and the action of gravity therefore, it must be eliminated.

\[
\begin{align*}
\text{Accelerations in motion captured by the accelerometer} & \begin{cases} x \\ y \\ z \end{cases} \\
\text{Acceleration at rest} & \begin{cases} x_g \\ y_g \\ z_g \end{cases} \\
\text{Desired accelerations} & \begin{cases} x' = x - x_g \\ y' = y - y_g \\ z' = z - z_g \end{cases}
\end{align*}
\]

It has been decided to make the rotation in order \( x, y, z \), therefore the rotation matrix \( R = R_x \cdot R_y \cdot R_z \) remains as follows:

\[
R = \begin{bmatrix}
\cos \theta \cos \psi & \cos \theta \sin \psi & -\sin \theta \\
\cos \psi \sin \theta \sin \phi - \cos \phi \sin \psi & \cos \phi \cos \psi + \sin \theta \sin \phi \sin \psi & \cos \phi \sin \theta \sin \psi - \cos \psi \sin \phi \\
\cos \phi \cos \psi \sin \theta + \sin \phi \sin \psi & \cos \phi \sin \theta \sin \psi - \cos \psi \sin \phi & \cos \theta \cos \phi
\end{bmatrix} \tag{30}
\]

Therefore, the system that determines the final accelerations after the filter has passed, eliminated the gravity component and correctly oriented, is as follows:

\[
\begin{bmatrix}
\dot{a}_x \\
\dot{a}_y \\
\dot{a}_z
\end{bmatrix} = R \begin{bmatrix}
x' \\
y' \\
z'
\end{bmatrix} = \begin{bmatrix}
x' \cdot \cos \theta - z' \cdot \sin \theta \\
x' \cdot \sin \theta \sin \phi + y' \cdot \cos \phi + z' \cdot \cos \theta \sin \phi \\
x' \cdot \cos \phi \sin \theta - y' \cdot \sin \phi + z' \cdot \cos \theta \cos \phi
\end{bmatrix} \tag{31}
\]
5. Experimental tests

The last part of this work is that once the gains have been determined and the controller has been implemented, the following circuit is followed to see how the robot really responds.

As discussed above, the real answer does not have to look like the simulated since there are many parameters of the robot that are not taken into account.

Two tests have been done giving random values to $K_P$ and $K_D$ and a last one with the values obtained theoretically in section 2.3.2. These values are the following:

<table>
<thead>
<tr>
<th>TEST</th>
<th>$K_P$</th>
<th>$K_D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>19</td>
<td>24</td>
</tr>
</tbody>
</table>

Table 2 - Gains in each test

The first parameter studied in these tests has been the distance between the sensor and the line of the circuit $d$, determining the error that is made by varying the gains.
The graph clearly shows the pattern of each round and with very similar values between each test. The fact that for different values of the gains this parameter varies so little is not the answer that was expected and suggests that something does not work as it should.

Then the angular speed of the vehicle has been studied since on one hand the angular speed desired by the controller is known and on the other hand, by means of the kinematic relation (2) the actual speed at which the vehicle is operating can be calculated.

![Figure 35 – Response for parameter $u_2$](image)

In this case, an expected response is obtained, since the speed desired by the controller is an ideal speed that does not take into account frictions and external factors that affect the real speed, making it always lower. In addition, it can be observed that the real speed is within the working range determiner in section 2.3.2.
CONCLUSIONS

The conclusions of this work have not been all expected. It has worked with a very simplified model which is affected by different factors to which the supply of work did not reach, therefore the response of the controller has not been the ideal.

Still, another of the objectives of the work was the theoretical design of a derivative controller that directly uses the acceleration detected by the accelerometer, which has been carried out throughout the study of the system and once achieved has been validated by simulations in Matlab, where it has been observed that the response of the system was correct and concordant with the specifications of the system.

Therefore, the echo of the implementation is left for later projects in which the factors that affect its functioning are studied.

Finally, another intermediate objective of the work, was the realization and implementation of a low pass filter to obtain a cleaner signal of the acceleration, and a system of rotation of the axes of the accelerometer. Both have been validated experimentally giving good results and proving that they were correct.
BUDGET

This work corresponds to 12 ECTS credits equivalent to 300 hours of which has been divided into 150 work in the laboratory, 100 autonomous work of learning and information search, and 50 in the preparation of documentation.

As it is an end-of-grade project, no salary has been received nor has it been necessary to purchase the necessary work devices, but taking into account that a salary has been received and that the necessary tools are not available, the budget it would be like this:

<table>
<thead>
<tr>
<th>CONCEPT</th>
<th>UNIT RATE [€/UT]</th>
<th>UNITS [UT]</th>
<th>COST [€]</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORKING HOURS</td>
<td>25</td>
<td>300</td>
<td>7500</td>
</tr>
<tr>
<td>PERSONAL COMPUTER</td>
<td>700</td>
<td>1</td>
<td>700</td>
</tr>
<tr>
<td>MICROCONTROLLER</td>
<td>10</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>SOFTWARE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>SUBTOTAL</strong></td>
<td></td>
<td></td>
<td><strong>8210</strong></td>
</tr>
<tr>
<td>VAT (21%)</td>
<td></td>
<td></td>
<td><strong>1724</strong></td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td></td>
<td></td>
<td><strong>9934</strong></td>
</tr>
</tbody>
</table>

*TABLE 3 - BREAKDOWN OF THE BUDGET*
ACKNOWLEDGMENTS

I would like to thank my tutors Arnau Dòria Cerezo and Victor Repecho del Corral for all the help they have offered me to carry out this project and for paying attention to me when there has been some inpreview. Thanks to this I have learned about a subject in which I had never moved and which I have enjoyed a lot.

I would also like to thank the IOC laboratory team for the good reception I received and of course my family and friends, with whom I have been able to tell at any time and situation and with whom without them I could not have reached where I am.
BIBLIOGRAPHY


ANNEXE: Software Code

```c
/* File Name: main.c */
#include "stm32f4xx_hal.h"
#include "stdio.h"
#include "stm32f4xx_hal.h"
#include "stm32f4_discovery.h"
#include "comunicaciones.h"
#include "defines.h"
#include le llibraria math per fer proves
#include <math.h>

/* Variables acelerometre */
int8_t Buffer[3];
IO float x, y, z, xx, yk, yk1, zk, zk1, xg, zg, xx, yy, zy, ay, oz, Ax, B, C, day, day1, vy, vy1;

/* Accelerometre */
extern void ESP_ACCELERO_Init (void);
extern void ESP_ACCELERO_GetXZY (int16_t *pDataXZY);
static void accel_comp (void);
static void angle_comp (void);

/* Buffers comunicacio */

extern uint8_t paquetSortida[NIDA_MAXDATA_PAQUET];
extern BufferFIFO BufferEntradaUART;
int8_t characterRx;
int8_t start = 0;

/* #**********************************************************************
63 /* USER CODE BEGIN Includes */
64 //**Variables que inicialitzem al programa**
65
66 // **SEÑALES DE REFERENCIA
67 _20 float wref=0.0; //velocitat que volen tancar m/s; wref=w2D*u-wref;
68 _20 float wref=0.0; //velocitat que volen tancar m/s; wref=w2D*u-wref;
69 _20 float wref=0.0;
70 _20 float wref=0.0;
71
72 // **CONTROLADORES PROPORTIONAL I INTEGRAL
73
74 // senyal controladors
75
76 _20 float dy[2] = [0, 100];
77 _20 float dy[2];
78
79 // diferencia measured amb referencia
80 _20 float error[2];
81
82 // senyal del control integral dreta a la right wheel
83 _20 float SenIntD[2];
84 // senyal del control integral dreta a la left wheel
85 _20 float SenIntD[2];
86
87 // senyal del control dreta; duty en X ya de 8 a 100 un cop aplicat senyal de control
88 _20 float dy[2];
89 // contents dels controladors
90 _20 float dy[] = [0, 100];
91 _20 float dy[2];
92
93 // **ENCODERS
94
95 // mesura
96
97 //_20 uint1_t uWfrequency[2]; // comptes fletes entre dos flans del encoder (d'esquerra, es per a cada roda), inicialitzem en un numero molt gran
98 _20 uint1_t PrescalerDec=100; // amb prescaler de 1200 obtindrem la mateix resolucio podem mesurar algunes de 5 segons sense escalfar flans
99 _20 uint1_t uW; 100
101 _20 float uWmeasured;
102 _20 float uWmeasured;
103 _20 float uWmeasured;
104 _20 float uWmeasured;
105 _20 float uWmeasured;
106 _20 float uWmeasured;
107 // wevsCalculat(nom) = number; //nombre = "+" **"C12S16" / 26;wCCValue=Preescaler; i 26 son els formats del encoder i 2" r es diametre roda 0,063m.
108 _20 float FLUX[10]; // 0[Hx] seves de comptes que pot fer el timer del encoder per segons
109 _20 float Nom[2] = [200, 0]; // Nombre de voltetes per cada interruccio: aquest cas per cada roda dreta
110 _20 float Don[2] = [100, 0]; // nombre de voltetes per cada interruccio: aquest cas per cada roda dreta
111 _20 float radsDrets[2]; // roda dreta
112 _20 float radsDrets[2]; // roda dreta
113
114 // uA = velocitat angular * w2D/r = w2D*... r=0,063
115 _20 float uA; // velocitat angular roda dreta
116 _20 float uA; // velocitat angular roda dreta
117 _20 float uA; // velocitat angular roda dreta
118
119 // uA; // velocitat angular = w2D/r = w2D*... r=0,063
120 _20 float uA; // velocitat angular roda dreta
121 _20 float uA; // velocitat angular roda dreta
122
```

---

**Control design and implementation of a twin robot**

---

**ANNEXE: Software Code**

---

**User code includes**: Includes the variables that initialize the program.

---

**Variables**: Defines references and control signals.

---

**Control Proportional and Integral**: Defines the control signals for each motor.

---

**Encoders**: Measures the encoder counts.

---

**Calculations**: Calculates the angular velocity based on encoder counts.

---

**Results**: Calculates results for both wheels based on encoder data.
```c
// variables

// I/O bit 0 cont0: // contador por indicar code LIFT parada
// I/O bit 1 cont0: // contador por indicar code HIGH parada

// **ACER AL MOTOR

// Valors introduits als motors

// **SENSOR DETECTOR DE OBSTACLES

// I/O bit 0 Value = 0;
// I/O bit 1 Value = 0;
// I/O bit 2 Value = 0;
// I/O bit 3 Value = 0;

// **SIGNALS DEL CONTROL DE LÍNEA**

// I/O bit 4 result:10000\[2]; // alguna variable es un nombre de 12 bits [0-4096] equivalent al\n// Volts [0-5V] que transmiten los dos sensors [3V

// I/O bit 5 Value;

// I/O bit 6 measured;

// I/O bit 7 measured2;

// I/O bit 8 w=8.18; //velocidad de crecer

// I/O bit 9 u;

// I/O bit 0 error0=0.0, error0 enter=0.0;

// ** CONTROLADOR Proporcional Distancia

// I/O bit 10 prop=0.0;

// I/O bit 11 kp=254.26; // tenint en compte el canvi d'unitats -- ambra 4.5 pero en canviar la calibració del\n// up així canvia per línies de 14mm

// ** CONTROLADOR Derivativo\n
// kp=0.5;

// ** CONTROLADOR Integral\n
// kp=0.000001, (0.000001 V/0 E)

// Temporizadores

// I/O bit 12 timer1_crono=0;

// USER CODE END Includes

// Private variables

// UART_HandleTypeDef huart;

// USER CODE FOREIGN PV */

// Private variables

// 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_TIM2_Init(void);

// static void MX_TIMER_Init(void);

// static void MX_TIM3_Init(void);

// void HAL_TIM_MspPostInit(TIM_HandleTypeDef *htim);

// USER CODE FOREIGN PPF */

// USER CODE FOREIGN OP */

// int main(void)
```
/* USER CODE BEGIN 1 */

/* USER CODE END 1 */

/* MCU Configuration-----------------------------*/

/* Reset of all peripherals, Initializes the flash interface and the Systick. */

HAL_Init();

/* Configure the system clock */
SystemClock_Config();

/* Systick end of count event each 10ms */
// SystemCoreClock = HAL_RCC_GetHCLKFreq();
// Systick_Config(SystemCoreClock / 100);

/* Initializar acelerometre */
BSP_ACCELEROM_Init();

/* Initialize all configured peripherals */
BSP_LED_Init(LED5);
BSP_LED_Init(LED4);
BSP_LED_Init(LED3);
MX_GPIO_Init();
MX_DMA_Init();
MX_ADC_Init();

// HAL_TIM_PWM_Start(T1, TIM_CHANNEL_1);  HAL_TIM_PWM_Start(T1, TIM_CHANNEL_1);
// HAL_TIM_PWM_Start(T1, TIM_CHANNEL_1);  HAL_TIM_PWM_Start(T1, TIM_CHANNEL_1);
// HAL_TIM_PWM_Start(T2, TIM_CHANNEL_1);  HAL_TIM_PWM_Start(T2, TIM_CHANNEL_1);
// HAL_TIM_PWM_Start(T3, TIM_CHANNEL_1);  HAL_TIM_PWM_Start(T3, TIM_CHANNEL_1);
// HAL_TIM_PWM_Start(T4, TIM_CHANNEL_1);  HAL_TIM_PWM_Start(T4, TIM_CHANNEL_1);
// HAL_TIM_PWM_Start(T5, TIM_CHANNEL_1);  HAL_TIM_PWM_Start(T5, TIM_CHANNEL_1);
// HAL_TIM_PWM_Start(T6, TIM_CHANNEL_1);  HAL_TIM_PWM_Start(T6, TIM_CHANNEL_1);
// HAL_TIM_PWM_Start(T7, TIM_CHANNEL_1);  HAL_TIM_PWM_Start(T7, TIM_CHANNEL_1);

//fin loop

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

// **INICIALIZACION DEL SOFTWARE DE COMUNICACIONES**
BSP_IO_Init();
InitUART1();
//UART5->SR = 0x10;

reboot(T1)(caracterNo. 1);
configurarConexioWiFi();
establirConexio();
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);
    COM1_gestionarComunicacions();
    //HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,GPIO_PIN_RESET);
    if (BSP_PB_GetState(BUTTON_KEY) != KEY_NOT_PRESSED)
    {
        HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,GPIO_PIN_SET);
        angle_comp();
        HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,GPIO_PIN_RESET);
    }
}

303\void HAL_SYSTICK_Callback(void)
304\{
305\  HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,GPIO_PIN_SET);
306\  BSP_ACCELEROM_GetXYZ(Buffer);
307\  x = Buffer[0];
308\  y = Buffer[1];
309\  z = Buffer[2];
310\  /* Filtre x */
311\  xk1 = (float) (1-(TS_N0X))*xk + (x*TS_N0X);
312\  xk=xk1;
313\  /* Filtre y */
314\  yk1 = (float) (1-(TS_N0Y))*yk + (y*TS_N0Y);
315\  yk=yk1;
316\  /* Filtre z */
317\  zk1 = (float) (1-(TS_N0Z))*zk + (z*TS_N0Z);
318\  zk=zk1;
319\  accel_comp();
320\  HAL_GPIO_WritePin(GPIOB,GPIO_PIN_15,GPIO_PIN_RESET);
321\}

325\void angle_comp(void)
326\{
327\  yg = yk1;
328\  zg = zk1;
329\  xg = xk1;
330\  A = atan( yg / zg );
331\  B = atan( xg / (sqrt(pow(yg, 2) + pow(zg, 2))));
332\  C = 0;
333\}
void accel_comp (void)
{
    xx = xk1 - xg;
    yy = yk1 - yg;
    zz = zk1 - zg;
    ax = xx*cos(c)*cos(C) + yy*cos(b)*sin(C) - zz*sin(b);
    ay = xx*cos(c)*sin(C) + yy*cos(b)*cos(C) - zz*sin(b);  
    az = xx* cos(a)*cos(c)*sin(b) + (cos(a)*sin(b) + sin(a)*sin(c)) + zz*cos(b)*cos(a);  
    /* Controlador derivativo de l’acceleració */
    vy1 = vy1 + (float)(1.0/16)*vy1;
    vxy1 = vxy1 + (float)(1.0/16)*vxy1;
    vy = vy1;
}

/** System Clock Configuration */

void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct;
    RCC_ClkInitTypeDef RCC francais_plljabi;

    __HAL_RCC_PLL_CLK_ENABLE();

    __HAL_RCC_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.OscillatorFreqHSE = 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_ClockInitConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5);

    HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/10);

    HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

    /* SysTick IRQn interrupt configuration */
    HAL_NVIC_SetPriority(SystemTick_IRQn, 0, 0);

    /* ADCl int function */
    void MX_ADC1_Init(void) // capta els volts del sensor de línia
{
    
}
```c
 ADC_ChannelConfigTypeDef sConfig;

// Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)
/

HAL_ADC_Instance = ADC1;
HAL_ADC_Init.ClockPrescaler = ADC_CLOCKPRESCALER_PCLK2_D2V2;
HAL_ADC_Init.Resolution = ADC_RESOLUTION_12B;
HAL_ADC_Init.ScanConvMode = ENABLE;
HAL_ADC_Init.ContinuousConvMode = DISABLE;
HAL_ADC_Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_1EDGE_NONE;
HAL_ADC_Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_1T1_ED1;
HAL_ADC_Init.DataAlign = ADC_DATAALIGN_RIGHT;
HAL_ADC_Init.NbrOfConversion = 1;
HAL_ADC_Init.NbrOfSamplesRequests = ENABLE;
HAL_ADC_Init.DCCSelection = DISABLE;
HAL_ADC_Init(&handling1);
/

// Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

/*

  sConfig.Channel = ADC_CHANNEL_8;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  HAL_ADC_ConfigChannel(&handling1, &sConfig);
/

// Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

/*

  sConfig.Channel = ADC_CHANNEL_9;
  sConfig.Rank = 2;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  sConfig.Offset = 0;
  HAL_ADC_ConfigChannel(&handling1, &sConfig);
*/

// TIM1 init function */
void TIM1_Init(void) // per a la interrupcicio i controlar els motors
{
  TIM1_ClockConfigTypeDef sClockSourceConfig;
  TIM1_MasterConfigTypeDef sMasterConfig;
  TIM1_BreakCallbackTypeDef sBreakCallback;
  TIM1_OC_InitTypeDef sConfigOC;

  html1.Instance = TIM1;
  html1.Init.Prescaler = 0;
  html1.Init.CounterMode = TIM_COUNTERMODE_UP;
  html1.Init.Period = 50000;
  html1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  html1.Init.RepetitionCounter = 0;
  HAL_TIM_Base_Init(&html1);

  // the modes of both channels have to be opposite to turn the wheels on the same sense
  sConfigOC.OCPMode = TIM_OCPMODE偓;
  sConfigOC.OCPulse = 12000;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCPNPolarity = TIM_OCPNPOLARITY_HIGH;
  sConfigOC.OCCallbackMode = TIM_OCCOCALLBACK_DISABLE;
  sConfigOC.OCDivideState = TIM_OCDIVSTATE_RESET;
  sConfigOC.OCPINState = TIM_OCPINSTATE_RESET;
  sConfigOC.OCPINState = TIM_OCPINSTATE_RESET;
  HAL_TIM_PWM_ConfigChannel(&html1, &sConfigOC, TIM_CHANNEL_1);
```
vehp que en tim3
445  sSlaveConfig.SlaveMode = TIM_SLAVEMODE_RESET;
446  sSlaveConfig.InputTrigger = TIM_TS_TIMPO;
447  sSlaveConfig.TriggerPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
448  sSlaveConfig.TriggerFilter = 0;
449  HAL_TIM_SlaveConfigSynchronization(&htim4, &sSlaveConfig);
450  
451  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
452  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
453  HAL_TIMx_MasterConfigSynchronization(&htim4, &sMasterConfig);
454  
455  sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
456  sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
457  sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
458  sConfigIC.ICFilter = 0;
459  HAL_TIM_IC_ConfigChannel(&htim4, &sConfigIC, TIM_CHANNEL_2);
460  
461  //TIM4->CRL = TIM_CRL_UDIS;
462  
463 } /* TIM9 init function */
464  
void MX_TIM10_Init(void)
465  {
466    TIM_OC_InitTypeDef sConfigOC;
467    
468    htim10.Instance = TIM10;
469    htim10.Init.Prescaler = 1;
470    htim10.Init.CounterMode = TIM_COUNTERMODE_UP;
471    htim10.Init.Period = 65535;
472    htim10.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
473    HAL_TIM_Base_Init(&htim10);
474    
475    TIM_OC_Init(&htim10);
476    sConfigOC.OCMode = TIM_OCMODE_PWM1;
477    sConfigOC.Pulse = 65535;
478    sConfigOC.OCpolarity = TIM_OCPOLARITY_HIGH;
479    sConfigOC.OCresetMode = TIM_OCRreset_DISABLE;
480    HAL_TIM_PWM_ConfigChannel(&htim10, &sConfigOC, TIM_CHANNEL_1);
481    
482    HAL_TIM_PWM_Init(&htim10);
483  } /* TIM11 init function */
484  
void MX_TIM11_Init(void)
485  {
486    TIM_IC_InitTypeDef sConfigIC;
487    
488    htim11.Instance = TIM11;
489    htim11.Init.Prescaler = 255;
490    htim11.Init.CounterMode = TIM_COUNTERMODE_UP;
491    htim11.Init.Period = 65535;
492    htim11.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
493    HAL_TIM_Base_Init(&htim11);
494    
495    HAL_TIM_IC_Init(&htim11);
496    sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
497    sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
498    sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
499    sConfigIC.ICFilter = 0;
500  
501  

Control design and implementation of a twin robot

```c
685  HAL_TIM_IC_ConfigChannel(&htim1, &configIC, TIM_CHANNEL_1);
686 
687  }
688 /* TIM12 Init function */
689 void MX_TIM12_Init(void)
690 {
691  TIM_SlaveConfigTypeDef sSlaveConfig;
692  TIM_IC_InitTypeDef sConfigIC;
693 
694  htim2.Instance = TIM2;
695  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
696  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
697  htim2.Init.Period = 0x3F3F;
698  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
699  MX_TIM_IC_Init(&htim2);
700 
701  sSlaveConfig.SlaveMode = TIM_SLAVEMODE_RESET;
702  sSlaveConfig.GPIOInput = TIM_GPIOINPUT_DISABLE;
703  sSlaveConfig.TriggerMode = TIM_TRIGMODEseysen;
704  sSlaveConfig.Trigger = TIM_TRIGSOURCE_GTRF;
705  sSlaveConfig.TriggerFilter = 0;
706  sConfigIC.IOClockSource = TIM_CLOCKSOURCE_INTERNAL;
707  sConfigIC.IOPolarity = TIM_INPUTPOLARITY_HIGH;
708  sConfigIC.IOSequence = TIM_IOSERIALIZE_NONINVICT;
709  sConfigIC.IOPreference = TIM_IPRIORITY_HARDI;
710  sConfigIC.IOPriority = TIM_IPRIORITY_HIGHI;
711  MX_TIM_IC_ConfigChannel(&htim2, &sConfigIC, TIM_CHANNEL_1);
712 
713  /* Enable TIM2 controller clock */
714  void MX_TIM2_Init(void)
715  {
716    /* TIM2 controller clock enable */
717    __HAL_RCC_TIM2_CLK_ENABLE();
718    /* TIM2 interrupt init */
719    HAL_NVIC_SetPriority(TIM2_IRQn, 0, 0);
720    HAL_NVIC_EnableIRQ(TIM2_IRQn);
721  }
722 
723  /* TIM2 controller clock enable */
724  __HAL_RCC_TIM2_CLK_ENABLE();
725 
726  /* Configure GPIO pin: PA8 */
727  GPIO_InitStruct.Pin = GPIO_PIN_0;
728  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
729  GPIO_InitStruct.Pull = GPIO_NOPULL;
730  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
731 
732  /* EXTI interrupt init */
733  HAL_NVIC_SetPriority(EXTI_IRQn, 0, 0);
734  HAL_NVIC_EnableIRQ(EXTI_IRQn);
735 
736  /* Configure GPIO pin: PA8 */
737  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, GPIO_PIN_RESET);
738  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
739  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, GPIO_PIN_RESET);
740 
741  /* Configure GPIO pin: PA8 */
742  GPIO_InitStruct.Pin = GPIO_PIN_11 | GPIO_PIN_13 | GPIO_PIN_15;
743  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
744  GPIO_InitStruct.Pull = GPIO_NOPULL;
745  GPIO_InitStruct.Speed = GPIO_SPEED_LOW;
746  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
```

```c
void MK_DAC_Init(void)
{
    // DAC Channel Configuration
    stConfig = DAC_ChannelTypeDef::DAC_CHANNEL1;
    HAL_DAC_Init(&hdac);  
    stConfig.DAC_Channel = DAC_CHANNEL1;  
    stConfig.DAC_PautBuffer = DAC_OUTPUTBUFFER_DISABLE;  
    stConfig.DAC_ConfigureChannel(&hdc, &stConfig, DAC_CHANNEL1);
}

// DIO CONTROL
// INTERFACCIA. Ejecuta cada 80 microsegundos el ADC y mantiene valores disponibles
// ADCCorvertedValue[0-7] buffer de 8 posiciones
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)
{
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, GPIO_PIN_SET);  
    // TCN Initialization();
    // cronometra
    if (crono == 0xFFFFF){
        crono=0;
    }else{
        crono=crono+1;
    }
    if (ADCCorvertedValue[0] == 0){
        // MSGURA X MODE, SIENTE QUIÉN DETECTÓ ANIMAL
        // RODA ALEAT.
        if (cont < 20000) {
            cont=cont+1;
            dmeasured = (float)(cont/DWight);  
            DWeight = (float)(cont/DWight);  
            // Pregunta sobre la N SGLOS EL DUT que ESTÉ SUMINISTR.
        }else{
            dmeasured=0.0;
        }
        if (DWeight < left_weight) {
            // extensión rodó parado, ajusta ser la medida final
            dmeasured = (float)(cont/DWight);
        }else{
            dmeasured=0.0;
        }
    }else{
        // extensión rodó parado, ajusta ser la medida final
        dmeasured = (float)(cont/DWight);
    }
    // PARÁMETROS PARAodal
    // TIPO DE GRASA
    if ((HAL_TIMDelegateFlag != (HAL_TIM(delegate_fc1) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1))))
    // Ejecuta a 'o' para el software el flag que indica
    // TIM_CHANNEL.0
    // (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1))
    // TIM_CHANNEL.1
    // (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1))
    // OBTENIENDO EL VALOR MEDIDO
    // DMEASURED = (float)(cont/DWight);
    // DECRECión
    // (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1))
    // DETECTING DISTANCE
    // UValue = HAL_TIM_ReadCapturedValue(halgt2, TIM_CHANNEL1);  
    // valor caso el sensor de proximidad, si es 0 no detecta res, de lo contrario, calcular la
    // InvValue = (float)1 / (1 / UValue);
    // US Distance = (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1))
    // DECRECión
    // (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1)) | (HAL_TIM(delegate_fc1))
    // OBTENIENDO EL VALOR MEDIDO
    // VValue = (float)invADCConvertedValue[0];  
    // invADCConvertedValue[1];
```
Control design and implementation of a twin robot
if (dutyRcorr < 0) {dutyRcorr=0;}

// pass to duty [0-100] w duty [0-500]

//dutyRcorr = 100.0; per ficar velocitat màxima de la roda dreta

PWMSpeedR=(dutyRcorr*0.5); //PWMSpeedR=(float)(dutyRcorr*(500/100));

dutyLcorr= dutyLref+(error1*(XpMotor)+SensInt);

// ajustar la senyal de control

if (dutyLcorr < 0) {dutyLcorr=0;}

if (dutyLcorr > 200) {dutyLcorr=200;}

// pass to duty [0-100] w duty [0-500]

//dutyLcorr = 100.0; per ficar velocitat màxima de la roda esquerra

PWMSpeedL=(dutyLcorr*0.5); //PWMSpeedL=(float)(dutyLcorr*(500/100));

Thesis