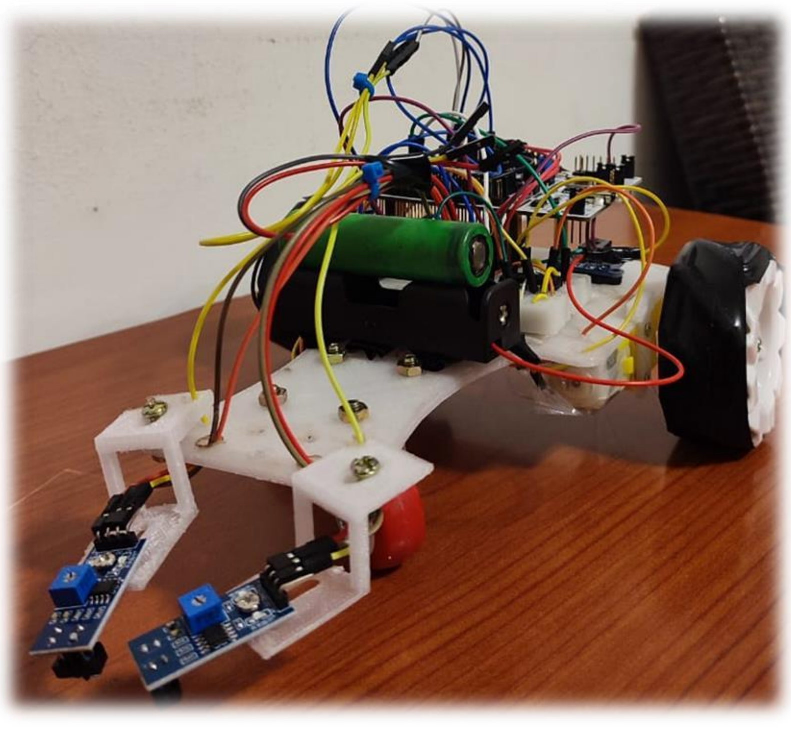
UNIVERSITÀ DELLA CALABRIA

Dipartimento di INFORMATICA , MODELLISTICA , ELETTRONICA e SISTEMISTICA

Master In Robotics and Automation

Line - Following Robot using ROS

Laboratory of Mechatronic Final Project



Professor:
Prof. Marco Lupia

Student:
Sofia Sanchez
(232620)

Academic year 2023

Introduction

The project consists of the construction of line-following robot able to move automatically and follow a detected black line that is drawn on the surface. The measure of the sensors used to detect the line and the encoders which are used for building the control are read by an ESP board, this information is sent to ROS via socket communication by a publisher node created on ESP. Then, in ROS a subscriber-publisher node receives the information related to the encoders sensors and process it to obtain the needed control actions. As ESP is development board that cannot afford complex programs it was incorporated a second board which are going to receive the values before explained using a subscriber node on it and execute all the logic to move the robot. On the first ESP it was also create another subscriber that will receive a message to know when the robot has to start to move and stop.

First, it is presented the logic under which the system will base its behavior, the electronic elements used for the development of the model, and the reason for its choice. Then, the logic and form of programming used are explained. Finally, the interchange of information between nodes, while the robot is working, is presented.

1. Objectives

The aim of this project consists of built a model, read its information and send control actions to an ESP via nodes communication to construct a following-liner robot that work with the logic shown in fig1.

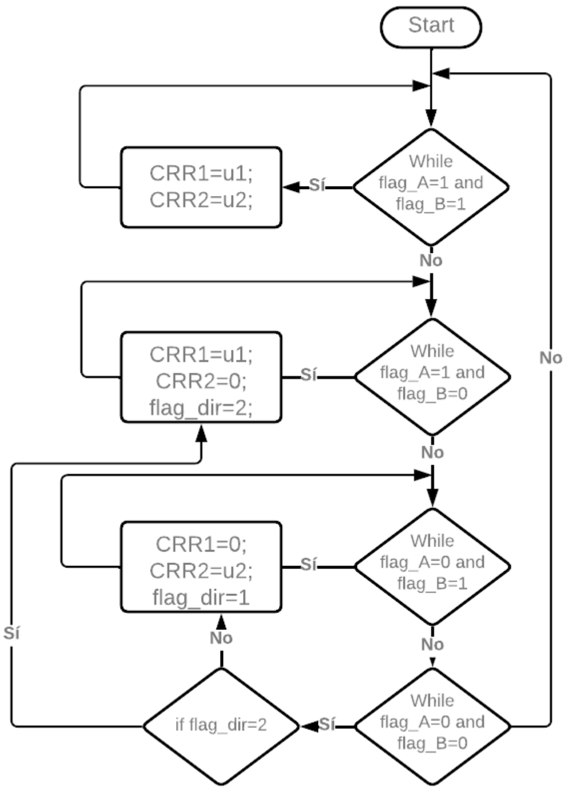

Start
While flag_A=1 and flag_B=1
Sí
CRR1=u1; CRR2=u2;
No
While flag_A=1 and flag_B=0
Sí
CRR1=u1; CRR2=0; flag_dir=2;
No
While flag_A=0 and flag_B=1
Sí
CRR1=0; CRR2=u2; flag_dir=1
No
While flag_A=0 and flag_B=0
Sí
if flag_dir=2
Sí
No
No

Fig.1 Programming logic Flow chart

2. Configuration and Implementation

The inputs, outputs and their configuration for the first ESP board are shown in table 1 and for the second one in table2. The code is developed in the Arduino programming environment (compatible with ESP board) following the logic of the flow chart shown in fig.1.

Table 1. configuration of I/O ports for 1 ESP

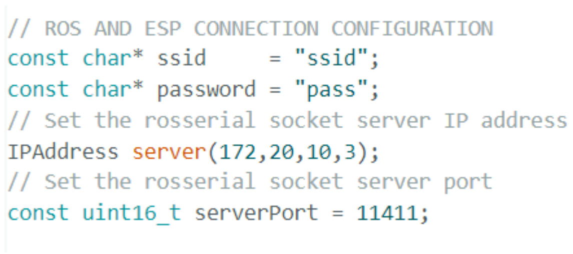
Name	Description	Pin	Configuration	Mode
LED 1	Start/Stop	2	Output	--
Sensor A	Left Infrared Sensor	34	Digital Input	Interruption
Sensor B	Right Infrared Sensor	35	Digital Input	Interruption
Encoder 1	Encoder for the right wheel	32	Digital Input	Interruption
Encoder 2	Encoder for the left wheel	33	Digital Input	Interruption

Table 2. configuration of I/O ports for 2 ESP

Name	Description	Pin	Configuration	Mode
Start	Start/Stop	32	Input	--
PWM 1	PWM for the right wheel	12	Alternate Function	--
PWM 2	PWM for the left wheel	13	Alternate Function	--

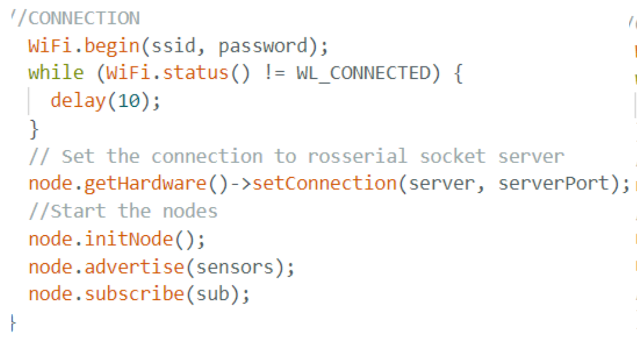
2.1. ROS and ESP board communication

The first step that will allow us to transmit and receive data between our real robot and ROS is to configure and establish socket communication. To do this first the configuration for the connection were set such as IP address, name of user, password, and port, as follows.

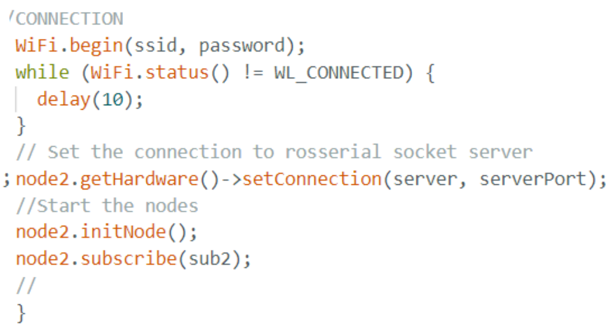


```
// ROS AND ESP CONNECTION CONFIGURATION
const char* ssid     = "ssid";
const char* password = "pass";
// Set the rosserial socket server IP address
IPAddress server(172,20,10,3);
// Set the rosserial socket server port
const uint16_t serverPort = 11411;
```

Then to establish the connection the follow code was used inside the "void setup" function in Arduino for the first and second ESP boards. Here we can notice that when the connection starts the nodes were started too.



```
'/CONNECTION
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(10);
  }
  // Set the connection to rosserial socket server
  node.getHardware()->setConnection(server, serverPort);
  //Start the nodes
  node.initNode();
  node.advertise(sensors);
  node.subscribe(sub);
}
```



```
'CONNECTION
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
  delay(10);
}
// Set the connection to rosserial socket server
node2.getHardware()->setConnection(server, serverPort);
//Start the nodes
node2.initNode();
node2.subscribe(sub2);
//
}
```

These nodes are used to send the information of the sensors to ROS to compute the control actions using a node inside of it. Then, to send back the output of the PID a subscriber in the second ESP board was made. The last node created on Arduino is a subscriber node in the first ESP which will receive the order of start and stop the movement of the robot. The following code were used to create the nodes:

- ESP 1:

Publisher Node:

```
//NODES
ros::NodeHandle node;
// PUBLISHER NODE
std_msgs::Int16MultiArray str_msg;
ros::Publisher sensors("sensor", &str_msg);   //"sensor" name of the topic and "str_msg" type of mess
```

Subscriber Node:

```
// SUSCRIBER NODE
//Calback
void messageCb( const std_msgs::Empty& toggle_msg){ //calback name "messageCb"
      digitalWrite(2, HIGH-digitalRead(2));   // blink the led
      START=!START;
    }
ros::Subscriber<std_msgs::Empty> sub("toggle_led", &messageCb );// SUSCRIBE TOPIC "toggle_led"
```

- ESP 2:

Subscriber Node:

```
//NODES
ros::NodeHandle node2;
// SUSCRIBER NODE
//Calback
void messageCb( const std_msgs::Int16MultiArray& arr){ //calback name "messageCb"
  for (int i=0; i<arr.data_length;i++){
   uu[i]=arr.data[i];
  }
 }
ros::Subscriber<std_msgs::Int16MultiArray> sub2("pid", &messageCb ); //subscibe to topic pid
```

It is also important to mention that it is necessary to add the dependencies that allow us to work with a specific type of data. In this case were:

```
//Type of message that are going to be published or recived
#include <std_msgs/Int16MultiArray.h>
#include <std_msgs/Empty.h>
```

```
//Type of message that are going to be recived
#include <std_msgs/Int16MultiArray.h>
```

2.2. On/off the system

As we can see the subscriber node of the ESP 1 is used to simulate an open push button that performs the start and stop of the robot. The callback of this node is call messageCb and will allow us to change the state of the variable START which is used to turn on and off the system. This node is subscribed to a “toggle_led” topic and it will execute the callback when the node receives the message “toggle_msg”. As we have to start also the second ESP the pin 2 will also send a signal with this purpose.

```
// SUSCRIBER NODE
//Calback
void messageCb( const std_msgs::Empty& toggle_msg){ //calback name "messageCb"
      digitalWrite(2, HIGH-digitalRead(2));   // blink the led
      START=!START;
    }
ros::Subscriber<std_msgs::Empty> sub("toggle_led", &messageCb );// SUSCRIBE TOPIC "toggle_led"
```

2.3. Infrared Sensor (Line detectors)

The TCRT5000 consists of an infrared LED and a phototransistor (that is sensitive to light). This sensor has a coating on it to filter out light that is not within the infrared spectrum to help reduce the chance of environmental interference (this is what gives the input side of the TCRT5000 its black colour).

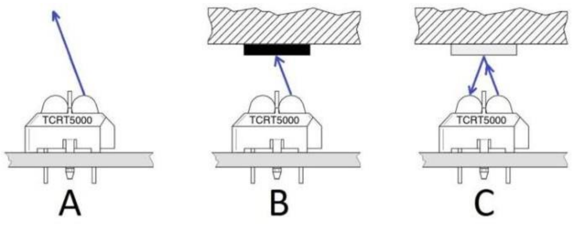


Fig.2.Infrared sensor Fig.3.Infrared sensor operation

Configuration of PIN 34 and PIN 35 as a line sensors:

```
//PINS for the line sensors
const int L_line = 34;
const int R_line = 35;
//Variables used in the interruptions of the line sensors.
short int FlagA;
short int FlagB;
```

Both sensors are working as interrupts because it is needed to know their states only when they change. These sensors are used to detect line and we must know when the robot is not more able to detect it. In other words, when the robot sees and does not see theline there will be an action to execute to allow the robot follows the line.

```
//INTERRUPTION CALLBACKS
//LINE SENSOR INTERRUPTION CALLBACK
void IRAM_ATTR isr1(){
  if (digitalRead(L_line)==LOW)
  FlagA=1;
  else
  FlagA=0;
}
void IRAM_ATTR isr2(){
  if (digitalRead(R_line)==LOW)
  FlagB=1;
  else
  FlagB=0;
}
```

```
//INTERRUPTIONS
//INTERRUPTION ASSOCIATED TO LINE_SENSORS
pinMode(L_line, INPUT_PULLUP);
attachInterrupt(L_line, isr1, CHANGE);
pinMode(R_line, INPUT_PULLUP);
attachInterrupt(R_line, isr2, CHANGE);
```

Each time when the interruption occurs two flags called *flagA* and *flagB* change their value between 0 and 1. These flags are used for the logic of the movement of the robot while it is following the line this is going to be explained later in the section on the logic of programming.

2.4. Gear motor

In the project, the motor that have been used are - more specifically - DC gear motors, with a gearbox ratio equal to 1:48; this means that the internal DC motor's velocity is reduced by a factor of 48.

Fig.5. gear motor

The motors are controlled whit a H-bridge founded in L293D chip. This is an electronic component that is provided by two H-bridge for that it allows to drive two DC motors in both of direction. The chip has 16 pins as it is shown in the following figure:

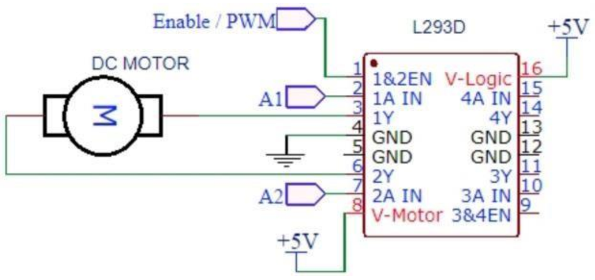


Fig.6 L293D H-bridge

How it was explained before the second ESP is going to receive the necessary control action to move the motors. The PINS 12 and 13 are used to this end, as it is show below:

```
int LED_PIN2=13;
int LED_PIN3=12;
```

We define a precision for the timer, in this case is 8 bits. It means that the duty cycle can be modify from 0 to 100 per cent with a precision of 1/256. Here also is define a frequency, in this case it was 1000 Hz.

```
// use 8 bit precission for  timer
#define LEDC_TIMER_8_BIT  8
// use 1000 Hz as a LEDC base frequency
#define LEDC_BASE_FREQ     1000
```

Then, the PWM was defined in the channels 2 and 3 and attached to the PINS 12 and 13. Finally, with the function *ledcWrite* we can give a duty cycle to a PWM. In this case it was 0 because the robot is stop.

```
//PWM

ledcSetup(LEDC_CHANNEL_2, LEDC_BASE_FREQ, LEDC_TIMER_8_BIT);
ledcSetup(LEDC_CHANNEL_3, LEDC_BASE_FREQ, LEDC_TIMER_8_BIT);
ledcAttachPin(LED_PIN2, LEDC_CHANNEL_2);
ledcAttachPin(LED_PIN3, LEDC_CHANNEL_3);
ledcWrite(LEDC_CHANNEL_2, 0);
ledcWrite(LEDC_CHANNEL_3, 0);
```

The PINS 12 and 13 are connected to the H-bridge to move the robot and its duty cycle will be modify with the values given by the PID outputs.

2.5. Encoders (Photo Interrupter Module)

It applies the principle that light is interrupted when an object passes through the sensor, soit's a switch that will trigger a signal when light between its gap is blocked. Therefore, photo-interrupters are widely used in many applications, like speed measurement.

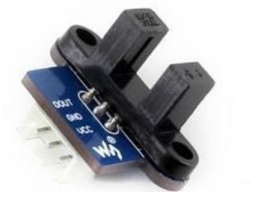


Fig.8. Encoder

The pins 32 and 33 were configured as an input. The operatingprinciple refers to a pull-up configuration because the sensor when a hole is detected, the digital input becomes high, otherwise it's equal to zero.

```
//PINS for encoder sensors
const int L_encoder = 32;
const int R_encoder = 33;
//Variables used in the interruptions of the encoders sensors.
int count_l=0; //count how many times the encoder read a hole
int count_r=0;
short int c1; //maintain the final count and it is given to the PID control
short int c2;
```

The digital input, of each encoder, are working as interrupts and become high when a hole is detected for that reason rising edges were considered. The used interruptions and callbacks were:

```
//INTERRUPTION ASSOCIATED TO ENCODERS_SENSORS
pinMode(L_encoder, INPUT_PULLUP);
attachInterrupt(L_encoder, isr3, RISING);
pinMode(R_encoder, INPUT_PULLUP);
attachInterrupt(R_encoder, isr4, RISING);
```

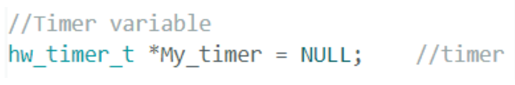
```
//ENCODER SENSOR INTERUPTION CALLBACK
void IRAM_ATTR isr3(){
  count_l+=1;
}
void IRAM_ATTR isr4(){
  count_r+=1;
}
```

When an interruption of an encoder occurs there a flag increases its value. In the case of the encoder on the right wheel the flag name is *count_r* and on the other case the flag is named *conunt_r*.

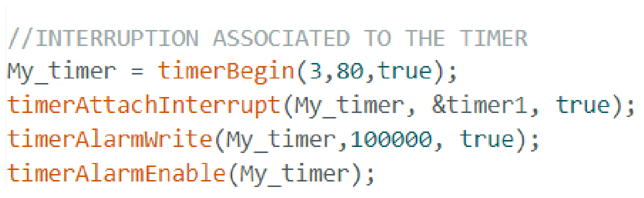
In fact, a printed wheels with 20 holes are in the sensor's gap and spin at the same velocityof the wheels. This velocity is which we want to measure. If the measurement process is performed in a minute, we can get the number of rpm.

$$rpm = \frac{count}{20} * 60$$

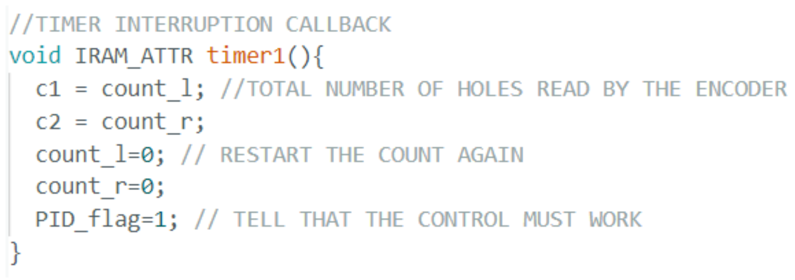
Then, the timer was configured to allows us to check the encoder each 0.1 seconds. As it is show in the following code lines:



```
//Timer variable
hw_timer_t *My_timer = NULL;    //timer
```



```
//INTERRUPTION ASSOCIATED TO THE TIMER
My_timer = timerBegin(3,80,true);
timerAttachInterrupt(My_timer, &timer1, true);
timerAlarmWrite(My_timer,100000, true);
timerAlarmEnable(My_timer);
```



```
//TIMER INTERRUPTION CALLBACK
void IRAM_ATTR timer1(){
  c1 = count_l; //TOTAL NUMBER OF HOLES READ BY THE ENCODER
  c2 = count_r;
  count_l=0; // RESTART THE COUNT AGAIN
  count_r=0;
  PID_flag=1; // TELL THAT THE CONTROL MUST WORK
}
```

When the interruption of the *timer* occurs the value of the flags used on the encoders *count_l* and *count_r* are stored (for be used on the PI control) and after are reset to zero to **measure the speed of the wheel each** 0.1 seconds.
The variable *PID_flag* is set to one in the interrupt each 0.1 secondto allows to use PI controller each sampling time. For do it that variable is checked inside the main *while* of the code.

2.6. Gear motor control (PI)

Since a DC motor has first order dynamics, a PI controller is enough to correct the error in space state and have an adequate system response. With the proportional action it is tried to minimize the system error. When the error is large, the control action is large and tends to minimize this error. While the integral action is achieved to reduce the error of the system in permanent regime.

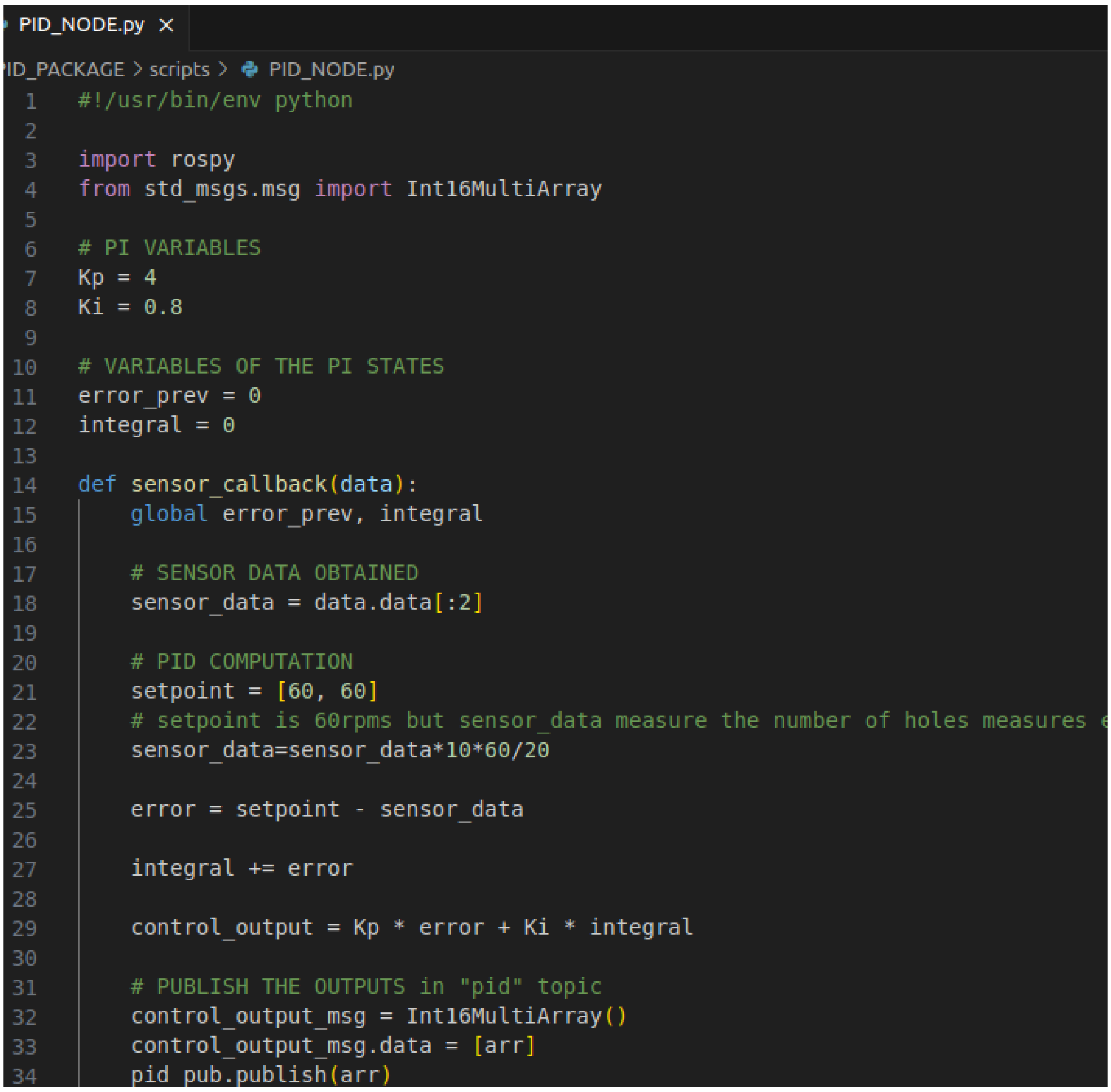
In order the to follow the reference of speed a PI control was introduced. The speed reference is: 60 rpm.

Due to not having the dynamics of the plant the tuning of the PI was done manually. The proportional and integrator gains was chosen by empirical way with this purpose. The proportional gain chosen was kp = 4 and the integral gain was

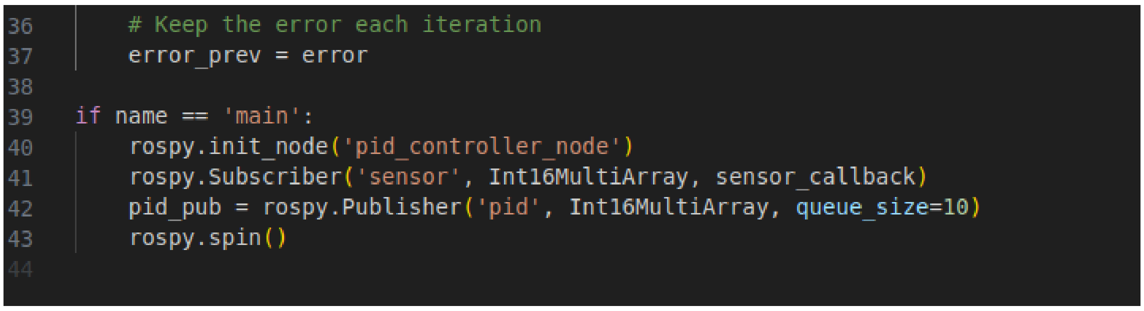
ki = 0.8.

Another important thing to remember is that the number of holes that has the printed wheel used to measure the rpm of the motors is 20 for that reason the sampling time Ts = 0.1 [seconds] is used to have the enough time to have more accuracy in rpm measure which is fundamental to able the motor to follow the different reference signal.

The PID was computed in ROS and then published to the second ESP. The node used to this purpose was the following.



```python
#!/usr/bin/env python

import rospy
from std_msgs.msg import Int16MultiArray

# PI VARIABLES
Kp = 4
Ki = 0.8

# VARIABLES OF THE PI STATES
error_prev = 0
integral = 0

def sensor_callback(data):
    global error_prev, integral

    # SENSOR DATA OBTAINED
    sensor_data = data.data[:2]

    # PID COMPUTATION
    setpoint = [60, 60]
    # setpoint is 60rpms but sensor_data measure the number of holes measures e
    sensor_data=sensor_data*10*60/20

    error = setpoint - sensor_data

    integral += error

    control_output = Kp * error + Ki * integral

    # PUBLISH THE OUTPUTS in "pid" topic
    control_output_msg = Int16MultiArray()
    control_output_msg.data = [arr]
    pid_pub.publish(arr)
```



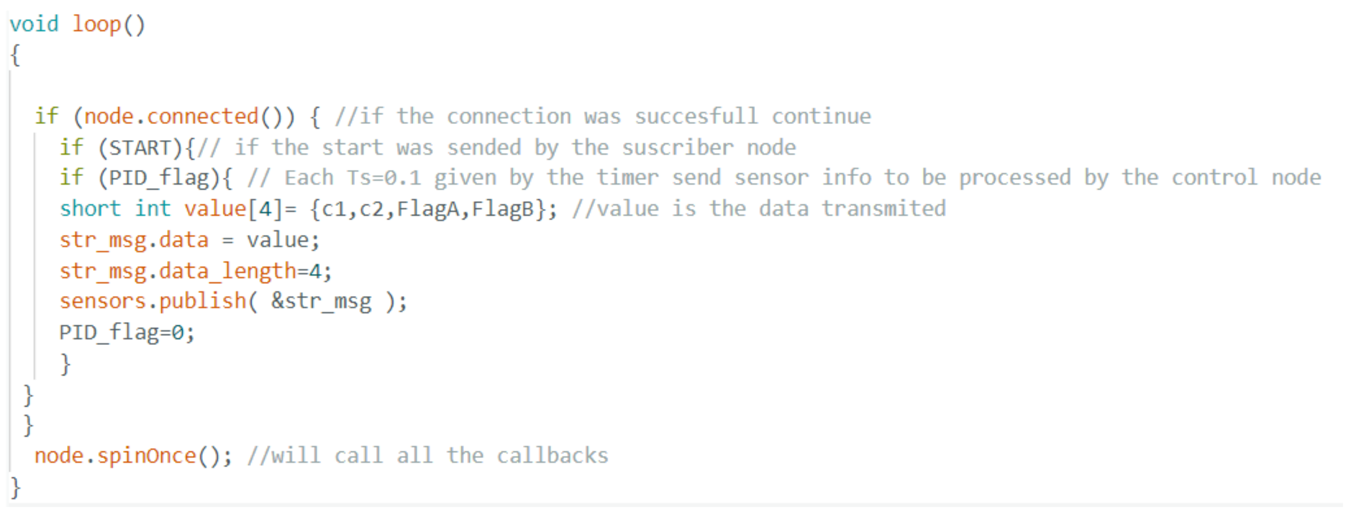
```python
    # Keep the error each iteration
    error_prev = error

if name == 'main':
    rospy.init_node('pid_controller_node')
    rospy.Subscriber('sensor', Int16MultiArray, sensor_callback)
    pid_pub = rospy.Publisher('pid', Int16MultiArray, queue_size=10)
    rospy.spin()
```

2.7. Void Loop

Finally, the programming logic that follows the configurations shown in fig.1 was implemented on the infinite *void loop* of each ESP board.

ESP1:

The first ESP has the goal of publish the data obtained by the sensor to ROS, it was done with the following code:



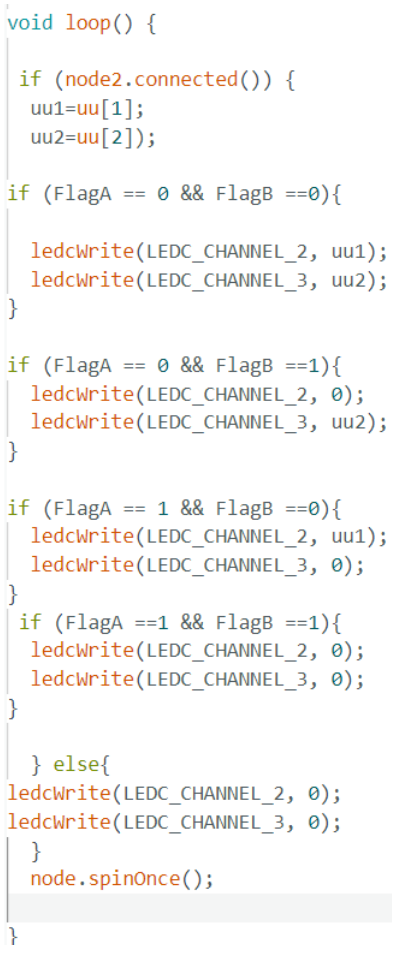
```cpp
void loop()
{

  if (node.connected()) { //if the connection was succesfull continue
    if (START){// if the start was sended by the suscriber node
    if (PID_flag){ // Each Ts=0.1 given by the timer send sensor info to be processed by the control node
    short int value[4]= {c1,c2,FlagA,FlagB}; //value is the data transmited
    str_msg.data = value;
    str_msg.data_length=4;
    sensors.publish( &str_msg );
    PID_flag=0;
    }
 }
 }
  node.spinOnce(); //will call all the callbacks
}
```

ESP2:

The robot must be able to follow a black line knowing the states of its sensors. Is important to know that the sensors will give a high level (1) as an output when they detect a black line, and a low level (0) otherwise. The robot will take different decisions depending on the following table:

Table 3. configuration of the sensor states

	Sensor A	**Sensor B**	**Description**
Case 1	1	1	Both sensors can see the black line.
Case 2	1	0	Only the sensor A is able to see the black line.
Case 3	0	1	Only the sensor B is able to see the black line.
Case 4	0	0	Any sensor can see the black line.



```
void loop() {

 if (node2.connected()) {
  uu1=uu[1];
  uu2=uu[2]);

if (FlagA == 0 && FlagB ==0){

  ledcWrite(LEDC_CHANNEL_2, uu1);
  ledcWrite(LEDC_CHANNEL_3, uu2);
}

if (FlagA == 0 && FlagB ==1){
  ledcWrite(LEDC_CHANNEL_2, 0);
  ledcWrite(LEDC_CHANNEL_3, uu2);
}

if (FlagA == 1 && FlagB ==0){
  ledcWrite(LEDC_CHANNEL_2, uu1);
  ledcWrite(LEDC_CHANNEL_3, 0);
}
 if (FlagA ==1 && FlagB ==1){
  ledcWrite(LEDC_CHANNEL_2, 0);
  ledcWrite(LEDC_CHANNEL_3, 0);
}

  } else{
ledcWrite(LEDC_CHANNEL_2, 0);
ledcWrite(LEDC_CHANNEL_3, 0);
  }
  node.spinOnce();

}
```