Department of
Electrical & Electronics Engineering
Abdullah Gül University

EE3002 Embedded Design System Capsule

Line Follower Project Report

Submitted on: 12.01.2021

Submitted by:

Muratcan Yazıcı 110110156

Mahmut Efil 110110157

Grade: / 100

How It Works?

We divided this part into 7 sub-sections to explain what is included in this project, the components, error calculation algorithm, CubeMx configuration, and system design. In this part of the report, these sub-sections are explained in detail.

1) QTR-8RC Sensor Working Principle and Code Algorithm

QTR-8RC is an array of 8 reflectance sensors. The pins are for simply the VCC, GND, and 8 sensors. Also, there is a pin called “LEDON”, which turns the LEDs HIGH. We did not connect this pin, because it is internally pulled HIGH if nothing is connected to it. The schematic of the QTR-8RC sensor is shown in Figure 1.

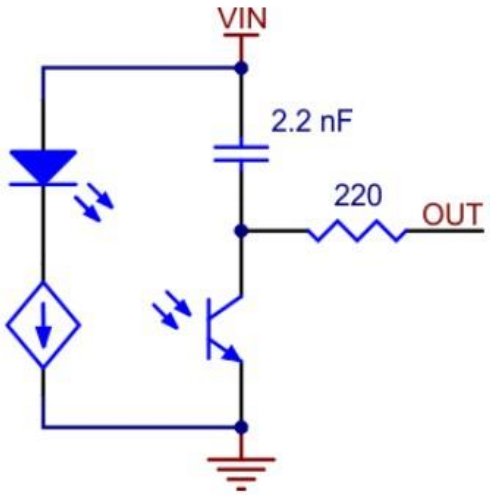

Figure 1: Schematic of QTR-8RC sensor (1 sensor only)

Each of these sensors has an LED/Phototransistor pair on them to read the values in parallel. To get the data from this sensor, we first pulled each LED high and started reading its values. When we give a HIGH voltage to the sensors, it automatically charges the capacitor on it. Then, the capacitor starts to discharge. Indeed, the complete discharge time determines whether the surface has high reflectance or not. If the surface has a high reflectance (Whiter), the capacitor discharges fast. Furthermore, if the surface has a lower reflectance (Darker), the capacitor discharges slowly. Thus, we measured the complete discharge time and assigned a threshold value to determine whether the surface is Black or White. Here, we do not return an Analogue signal. We return a Digital signal which returns only “1” and “0”, which represents black and white respectively.

2) L298 Full Bridge Motor Driver

These components are required to run the motors with the desired PWM signals. We aim to turn the motors depending on the error signals coming from the QTR-8RC sensor. This component can run two motors independently, which perfectly suits our application. The block diagram and the connection of the L298 Motor driver are shown in Figure 2.

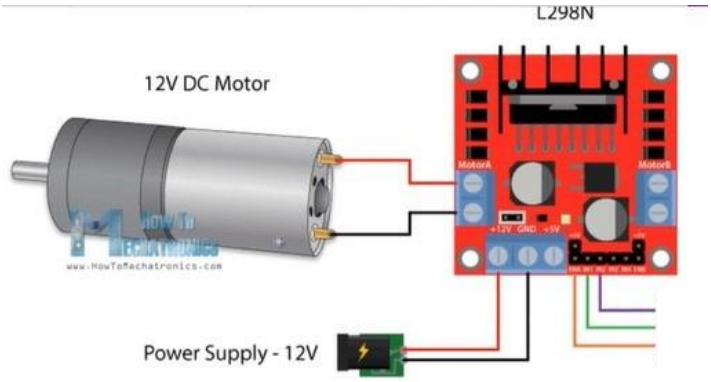

Figure 2: Motor Driver Block diagram and Connections

In the block diagram above, connections for motor 1 is shown. The modular requires a DC Voltage source that is connected to the "+12V" and "GND" terminal. We used two Li-ion 18650 batteries in series to power the module. This power configuration supplies between 6V to 8V depending on the charge level. Additionally, motor connections are also shown. There are 3 Pins to control each motor (Enable Pin, Input 1, Input 2). Depending on the Input 1 and Input 2 configurations, the motor either goes forward or backward. We applied PWM to Enable pins to control each motor. The required code for the configurations depending on the sensor data is available in the source code. Finally, a PWM signal is applied to each Enable pins to control the motor to follow the black line.

3) Vehicle Design

The design of the car belong to Muratcan Yazıcı is shown in Figure 3. The design of the car belong to Mahmut Efil is shown in Figure 4.

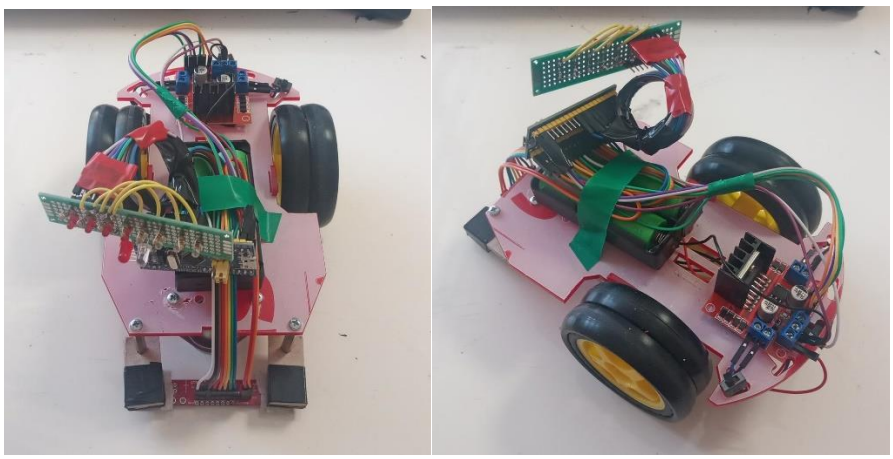

Figure 3: Vehicle design

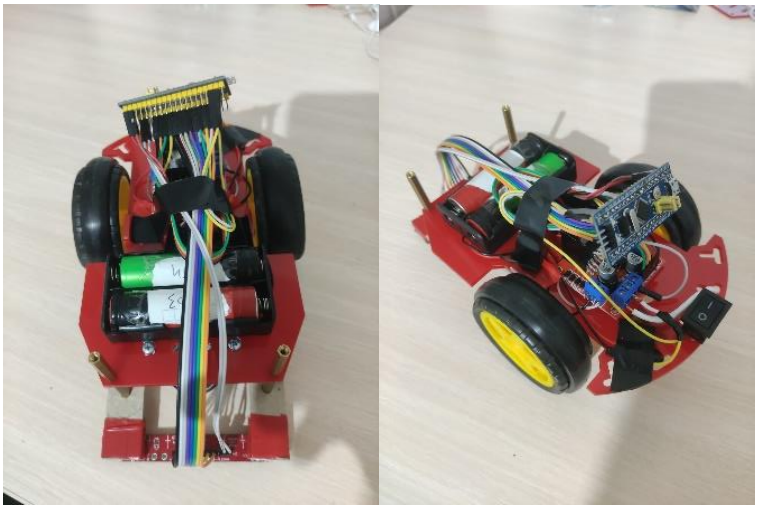

Figure 4: Vehicle design

As shown in Figure 3 and 4, the sensor is put close to the ground to read the values in better accuracy. In our design, the motor driver is put behind, and batteries in the middle front. Also, we both inserted a switch to run the motor whenever we want, not every time.

4) CubeMx Configuration

The CubeMx configuration for Muratcan and Mahmut's design are shown in Figure 5 and Figure 6, respectively.

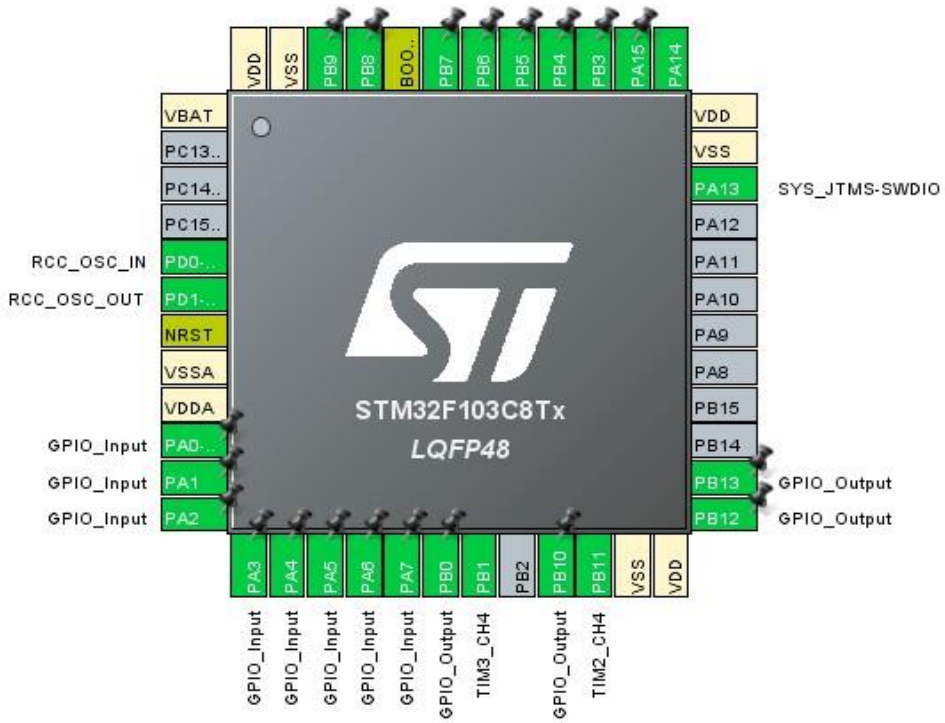

Figure 5: CubeMx Configuration for Muratcan's Design

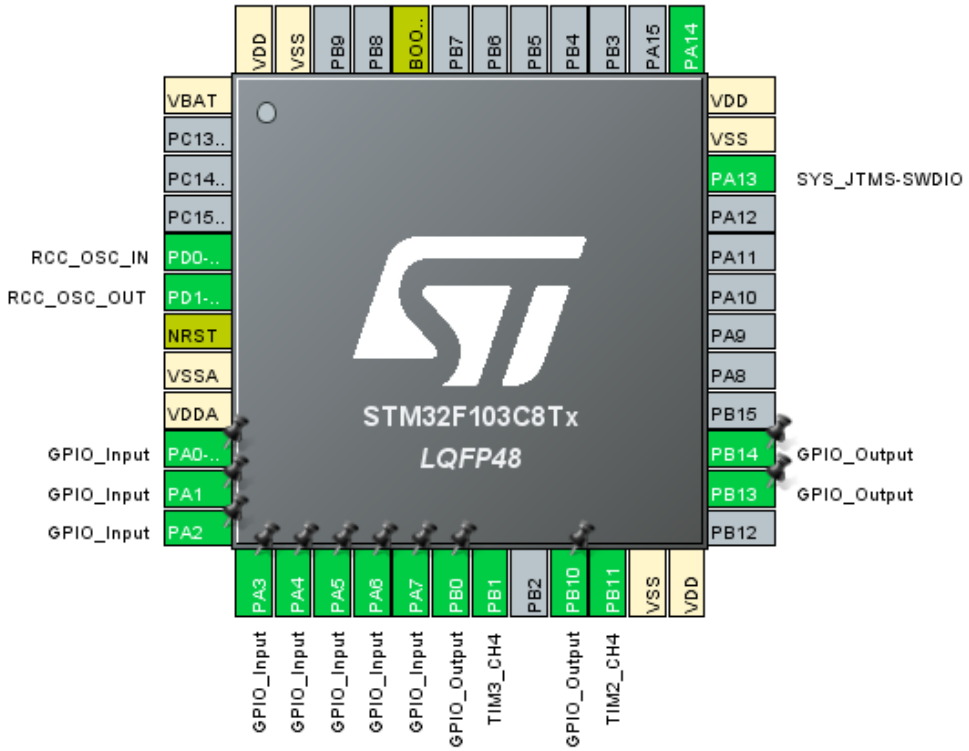

Figure 6: CubeMx Configuration for Mahmut's Design

We connected the QTR-8RC sensor from GPIOA Pin0 to Pin 7 (8 sensors in total). To provide the motors with PWM signals, We used Timer 2 Channel 4 and Timer 3 Channel 4. Here, important parameters are Prescaler (PSC) and Counter Period (Auto Reload). These parameters determine the frequency of the PWM signal depending on the following formula. We used 8MHz internal clock.

$$f = \frac{8MHz}{(PSC + 1)(Period\ + 1)}$$

Our PSC and Period values are 79 and 99 respectively. In this configuration, the frequency of the PWM signal is calculated as 1kHz.

We connected the PWM outputs to Enable Pins of the motor driver, and GPIO Output Pins to Input 1 and Input 2 Pins of the motor driver. These connections are done for both motors.

As shown in Figure 5, for Muratcan's design, the pins B9 to A15 are used to test the sensor. These pins are set as GPIO Output and they are connected to LEDs. Whenever the sensor detected a black surface, the corresponding LED turned on. LED array addition is explained in section 6.

5) Error Table and Error Calculation

For this project, the error calculation is critical. To determine the error signals, we first created an Error table which is illustrated in Table 1.

QTR0	QTR1	QTR2	QTR3	QTR4	QTR5	QTR6	QTR7	ERROR

0	0	0	0	0	0	0	1	4
0	0	0	0	0	0	1	1	3
0	0	0	0	0	1	1	0	2
0	0	0	0	1	1	0	0	1
0	0	0	0	1	1	1	0	2
0	0	0	1	1	1	0	0	1
0	0	0	1	1	0	0	0	0
1	0	0	0	0	0	0	0	-4
1	1	0	0	0	0	0	0	-3
0	1	1	0	0	0	0	0	-2
0	0	1	1	0	0	0	0	-1
0	1	1	1	0	0	0	0	-2
0	0	1	1	1	0	0	0	-1
0	0	0	0	0	1	1	1	4.5
0	0	0	0	1	1	1	1	4.5
0	0	0	1	1	1	1	1	4.5
0	0	1	1	1	1	1	1	4.5
1	1	1	0	0	0	0	0	-4.5
1	1	1	1	0	0	0	0	-4.5
1	1	1	0	0	0	0	0	-4.5
1	1	1	1	1	1	0	0	-4.5

Table 1: The Error Table

A) Slight Right and Left Turns

The slight turn cases are illustrated on Table 1 as “Right” and “Left”. In these cases, one motor is slowed down and other motor is speeded up depending on the error signal. This way, slight turns are completed.

B) 90 Degree Right and Left Turns

The 90-degree turn cases are illustrated in Table 1 as “90 Left” and “90 Right”. For these cases, particularly three, four, or five sensors from the most right or the most left sensors are checked. If this sensor results in Black, a 90-degree turn is required. For these cases, one motor is completely turned off, and the other is significantly speeded up. I assigned a big error value for one motor to stop and the other to speed up. This way, the sharp turns are completed.

C) Acute Angle Turns

We did not define Acute angle turns. These cases are special and need to be treated differently. For the robot to be able to have this turn, we kept the last read most right or most left sensor value. This value determined the direction of the motor’s next movement when the sensor is completely out of the track (All White Case). When the case is All White, the error signal is huge “8”. This big value caused the PWM signal to significantly increase for one motor, and significantly decrease for the other motor. However, turning the acute angle accurately is not possible when one motor is completely at a halt and the other turns fast. To turn back to the track quickly, one motor should turn backward, and the

other should go forward. Thus, we wrote another algorithm that turns the corresponding motor backward in such cases. In this way, the sharp turns are successfully fulfilled.

6) LED Sensor Indicators

We implemented one of our cars (Muratcan's design) a soldered array of 8 LEDs where each one corresponds to one sensor. This system turned on the corresponding LEDs of the sensors that read a low reflectance (Black Surface). This way, we were able to see what the robot sees, and how it acts accordingly. This method helped us a lot to debug the problems. A picture of the system implemented on the robot is shown in Figure 7.

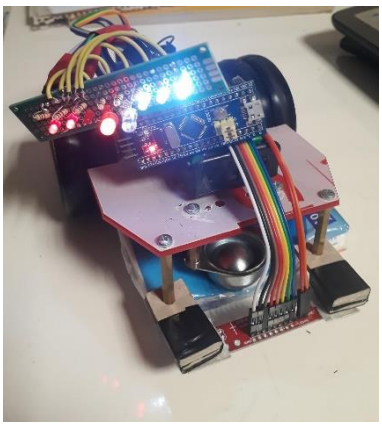

Figure 7: LED array on the Line Follower

6) PID Controller

In the control part, we basically designed a control logic in other words an ''artificial intelligence'' to keep our Robot to follow the line. This logic controller is called a PID controller which is a complex system. PID has a variety of usage in our lives and as you can see in Figure 8 below, there are three parts including Proportional, Derivative, and Integral gain control. The working principle of the PID controller is that error or deviation in the system is calculated by measuring instant value continuously with the help of QTR-8RC Sensor and the required response is given accordingly to obtain the set point that is the initial position with no error status as seen in Figure 8.

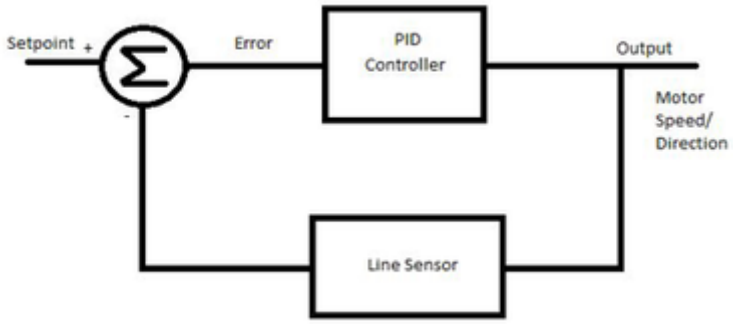

Figure 8: Block Diagram of PID Controller

The error term is defined in our code as a difference between the setpoint and the current value of the sensors. The P term in the PID represents the proportional term and it is equivalent to the error value.

The I term in the PID represents the integral term and it is equivalent to the sum of entire previous error values.

The D term in the PID represents the derivative term and it is equivalent to the difference between an instant error from a set point and an error from a previous moment. These quantities define the PID value for the controller and the calculation can be observed as follows:

$$PIDvalue = (Kp * P) + (Kd * D) + (Ki + I)$$

In the equation:

- Kp is used to adjust the magnitude of the change.
- Ki is used to adjusting the rate at which the change and eliminates the steady-state error.
- Kd is used to arrange stability, it reduces the overshoot.

We selected Kp=4, Kd=10, Ki=0. We arranged the values by trial error to obtain the best response. Low Kd values caused huge oscillations in our system. Thus, we selected a big Kd value. Also, the Kp is arranged to obtain a smooth movement. We did not use the Integral term because this term always sums the error values and causes instability after some time. We could have put a higher limit for the Integral term, yet the system was already working fine with the PD control mechanism.

In Figure 9, you can see the expressions and gain effects for the representative refence signal for P, I and D.

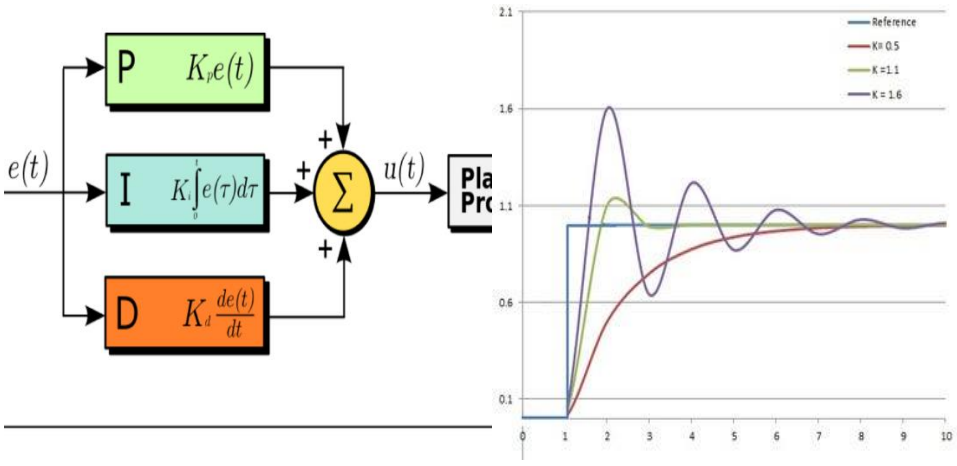

Figure 9: PID Controller Block Diagram and Effects of P, I, and D Gains

In our Robot design, we preferred to use a PD controller instead of a PID. First, we determined the P-value by trial error. After finding the appropriate P-value that is fast enough, we wanted to add the D term as well. Indeed, only P control is enough for this project, yet the system would have a huge

overshoot, that is robot would oscillate even in a straight path. P term lowers the overshoot by calculating the difference between the current error and the previous error. After adding the D coefficient, we spotted a significant overshoot decrease. The reason why we did not use the Integral coefficient is that it caused instability for our case. Integral term continuously sums all the previous errors, and after a certain time, its value causes one motor to turn fast and the other motor to stop. We could have eliminated this effect by adding a higher limit to the Integral term, yet we thought it is not necessary to create complexity on a system that works fine.

Responsibility Distribution

In this project, we worked as a group in the design and implementation parts since we both are in the same city and neighborhood. We decided on our design and each member designed his part by himself. In the algorithm part, the sensor value readout is implemented by Mahmut Efil and the PD Controller and motor control algorithm is implemented by Muratcan Yazıcı. The improvement part which includes value determination and essential arrangements is made by the contribution of both members. After we succeeded in all the parts, we explained the working principles to each other. Now, we are both able to design the car individually.

Challenging Parts

At the beginning of this project, we faced a lot of issues such as sensor readout problems, speed, and maneuvering problems, some design application problems, etc. Regarding the sensor readout problems, the determined threshold value in our design was problematic and we couldn't have the desired stability. To overcome this problem, we increased the threshold and the maximum number of repetitions while the sensor is reading suitable values. We also faced some unstable deflective situation while running our motor in the designed path. To get rid of this problem, we have designed a special code algorithm for acute-angled parts of the path. We have applied a PD controller and tried to obtain a stable system. Thanks to that way, we were also able to overcome the speed issue and we decreased the completion time from around 30 seconds to 13 seconds. In the first stage of the implementing part, we were using 4 serial alkaline batteries that provide 6V output. Then we realized that we cannot supply the motors and cannot get the desired speed. Therefore, we decided to use 2 Li-ion 18650 batteries in series to obtain a better voltage output. The nominal capacity of these batteries is 3400mAh and the C-Rate limit is 2C, meaning that a peak current of ~6.8A can be provided to the motors.

How to Improve More?

After completing our project, we thought that if we had more time to design and develop our project, we would choose completely different design materials for the body of our motors. We would first change the wheels of our motor and place a wheel that has a slim design with a better grip to overcome instability. It would be better to use rigid and lightweight hard plastic materials for the body. Additionally, we would choose our motors with more robust and high voltage input so that the speed could be increased. For the software part, the algorithm can be developed more. We can use a PID controller and we can eliminate time delay and steady-state errors.

References:

https://www.instructables.com/Line-Follower-Robot-PID-Control-Android-Setup/

https://www.pololu.com/docs/0J13/all

https://www.instructables.com/Robot-Line-Follower/

https://medium.com/@TowardInfinity/pid-for-line-follower-11deb3a1a643

https://www.st.com/resource/en/datasheet/stm32f205rb.pdf