Robot Lab Report

TU Delft

Ariel Potolski Eilat
Student number: 5482526

Note: in all the assignments, I use different speeds in the analogWrite() function to the motor pins. This is a way to compensate for the speed rotation difference between the two motors that was noticed during the implementation.

Assignment 3.1: Variable Speed

The code for this assignment was written in arduino, and therefore no timer settings on the STM32 were used. The arduino code for this assignment was really simple, as it was basically a sequence of analogWrite() calls to each pin (two pins per motor) of the motors followed by a delay. The speed was changed accordingly, such that the robot started running at a certain speed, increased that speed for a few seconds, and then decreased the speed until stopping.

Assignment 3.2: Obstacle Detection and Stopping

The goal of the second assignment was to start the robot and make it run. Once the robot detected an object - using an ultrasonic sensor placed on the front of the robot - and the distance to that object was less than 15 centimeters, the robot should stop. I completed this assignment by using arduino library interrupts.

In the setup method, I set the pin modes as usual, according to the function of the pins I used. Besides that, since I used an interrupt in this code, I also attached the interrupt to the desired pin by using the “attachInterrupt(pin, interrupt, state)” function. The way that function works is the following: the function attaches an interrupt to a given pin, and this interrupt is called whenever the pin goes to the desired state. In the case of this assignment, I made the call “attachInterrupt(echoPin, echoISR, CHANGE) ” to this function. This means that every time the signal received in the echo pin of the sensor changes, the echoISR interrupt (function) should be called.

The “echoISR” function goal is to calculate the distance the robot is from the object. In order to do that, I used the ultrasonic sensor. The way this works is the following. Whenever the echoPin of the sensor has a HIGH signal/state, I know that this is the start of the echo pulse, and save the time when that happens in a variable. Whenever the echoPin of the sensor has a HIGH signal/state, I know that this is the end of the echo pulse. With these two pieces of information, I can calculate the time difference between the start and the end of the pulse. Furthermore, once I have this difference, I can calculate the distance to the object which is given by the equation “distance = duration * 0.034 / 2”.

In the loop function, we start the motors of the robot to run in a straight line. After that, we emit the waves from the ultrasound sensor through the “trigPin”, which will cause the echoISR to be called in case the signal of “echoPin” changes. Finally, if the distance calculated in the ISR function is less than 15 centimeters, we stop the robot.

Assignment 3.3: Line Follower

The goal of this assignment was to code the robot to follow a black line using two infrared sensors. The code for this assignment was written in arduino. For this assignment, aside from the motors' pins, I used one pin for each IR sensor and one global variable - namely threshold - which value varies according to the ambient where the sensors will be used. In the case of this lab, the ideal threshold varied between 50 and 60. For this assignment, specifically, the chosen threshold was 50.

In the "setUp()" function, as is common, I setted up the pin modes of each pin. In the "loop()" function is where all the "dirty" work for this assignment can be found. First, I read the values from the IR sensors using the "analogRead()" function, and store each value in one variable. Then, I have one if statement, followed by two if else statements. The first case I considered (if statement), is what to do in case the values read from both sensors were smaller than or equal to the threshold. If this was the case, that would mean that our sensors were over the white part, and therefore the line would be between the sensors. Hence, the robot should continue moving straight forward. In the second case, I considered the left sensor to read a value above 50, and the right sensor a value smaller than or equal to 50. This would mean that the left side IR sensor of the robot is on the line, and consequently we would have to adjust the robot to the left. Thus, in that case, I had to set the right motor to spin forward, and the left motor to speed backwards. The last case I considered was the right IR sensor reading being greater than 50, and the left IR sensor reading being smaller than or equal to 50. The reasoning here is the same as described above for the left sensor, but for the right sensor instead. Accordingly, in that case, the left motor is set to spin forward, while the right motor is set to spin backwards.

One last note about this assignment's code, is that it is noticeable that I added some really small delays after the "analogWrite" calls for each case. The reason why I did that is because during the implementation, I found these delays to be a great help when making sharp turns specially (i.e. 90 degrees turns).

Assignment 3.4: Object Avoidance

The goal of this assignment was to write a code that makes the robot follow a black line, in such a way that if the robot senses an object on that line, it goes around it and continues following the line afterwards.

The reasoning and the code for following the line is exactly the same as described in the previous assignment, and therefore I will not go over it again. The object detection is made basically in the same way as I did in assignment two. The only difference here is that I don't use the interrupt to calculate the time difference between emitting and receiving the pulse. Instead, I use a function provided by

arduino called “pulseIn()”, that already returns that difference for you. The distance to the object is calculated in the same way as in assignment 3.2.

Once the distance detected from the object is less than 15 centimeters, I make a call to a function that I named “goAround()”. This function basically contains hard coded instructions for the robot to go around the object and back to the line, keeping a fair and reasonable estimation of the size of the object used in the lab. Hence, it is important to notice that, if applied to a bigger object than the one used in the lab, this function would not work, and it would have to be changed. At first, I tried using the front and side sensors, but I ran into some problems. Since the manual stated that it was okay to hard code this function, I decided to go for this easier approach.

Assignment 3.5: Fixed Distance

This last assignment’s goal was to make the robot travel a fixed distance, using speed encoders and slotted wheels. It is important to notice that I chose to make my robot follow the black line instead of just having it going straight. The code for following the line is the same as described in assignment 3.3 and used in assignments 3.3 and 3.4, with the difference that the threshold here was set to 60 instead of 50.

The first important thing that should be measured is that I keep two global variables that serve as counters to the number of slots read by the encoder. In the “setup()” function, I included the usual pin mode selection, as well as two “attachInterrupt()” calls, one for each speed encoder. Both interrupts, one for the left encoder and one for the right, do only one thing: increment the counter of its correspondent encoder by one. The interrupts are called every time there is a rising edge in the signal of the encoders, that is, whenever a slot on the slotted wheels passes by the encoder.

In the “loop()” function, before the if statements related to following the line, I read the values received by the encoders’ pins. After that, there is one if statement that checks if the robot has reached the target distance. To do that, however, I would need to either convert the target distance to counts, or the number of counts to distance. I decided to implement the former. In order to do that, I implemented the function “getDistanceInCounts” that does one simple calculation:

counts = (distance * CPR) / (wheelCircumference * 100),

where “distance” is the target distance, “CPR” is the acronym for Counts per revolution of the encoder, “wheelCircumference” is the circumference of the robot’s wheels, and the 100 is because the distance is given in centimeters. Given that, the condition I evaluate is whether the counter for both the right and left encoders is greater than or equal to the counts calculated for the target distance. If that is the case, the robot stops.