Embedded Systems: Robot Project

Johnson Domacasse(4471709)
Harm Nieuwland(4926749)
Date: 31/12/2023
Teacher: Suzana Andova

List of Figures

List of Tables

Document history

Version	Date	Status	Author	Description
0.1	2023-11-06	Draft	Johnson	Document creation
0.2	2023-14-11	Draft	Johnson	Phase 1 documentation

Terms, Abbreviations

Abbreviation	Description
IR sensor	Infrared red sensor
US sensor	Ultrasonic sensor
MCU	Micro controller unit
ARR	Auto-reload register
PSC	Pre scaler
FCS	Feedback Control Systems

1. Introduction

The robot platooning project is one of three projects that were assigned to each project group to work on. The primary goal is to make a robot that can smartly navigate a course of black tape on the floor and in later parts of the project, apply FCS for a smarter robot.

2. System Description

This section of the report is dedicated to give a brief description of what our team did to accomplish the robot project. A brief description is given of what the project entails and how it was solved.

The robot project has 3 main components that function together to form one entire system. These components are the ultrasonic sensor, the 2 Servo motors, and the infrared sensors. Each one of these components code, were compiled and analyzed to then be combined into one final machine. Below is a small description of each sensor and their purpose:

1. ***Ultrasonic sensor:*** The ultrasonic sensor acts as our feedback "giver" later on in the project when the feedback control systems are implemented as well. In the first design of the robot, it will simply act as a stop mechanism for the robot. Additional details will be given in the implementation phase 1 section.
2. ***Infrared sensors:*** The infrared sensors act as the steering wheel for the robot. Based on if the sensor is detecting a black line both straight or curved, it will alter send this information back to the program so the robot can steer accordingly based on the results.
3. ***Servo motors:*** The servo motors act as the wheels for the robot. It will start driving at a specific speed and based on the information it gets from both the ultrasonic – and the infrared sensors. More information on their calibration can be found in the implementation section.

Each sensor is connected on specific pins that are exclusive to the code that will be provided along wit this document. See table 1 for PIN configuration of the robot project. Be advised, some pins have the wire color attached to them because the wires are attached to the robot itself. **[5]**

	Actuator	Pin Name	Wire Color
1.	Ultrasonic Trigger	PB5	-
2.	Ultrasonic Echo	PB6	-
3.	Right-side Servo(clockwise rotation)	PA0	-
4.	Left-side Servo(counter clockwise rotation)	PA1	-
5.	IR OUT1 (leftmost)	PB10	White
6.	IR OUT2	PA8	Orange
7.	IR OUT3	PA10	Green
8.	IR OUT4	PB3	Orange
9.	IR OUT5 (rightmost)	PB4	Green

Table 1. Pin configuration

3. Design Phase 1

This section of the report will be dedicated to how the design of the robot is structured. For this project, different diagrams will be used to show what is happening through out the robot processes. See figure 1 below for complete state machine.

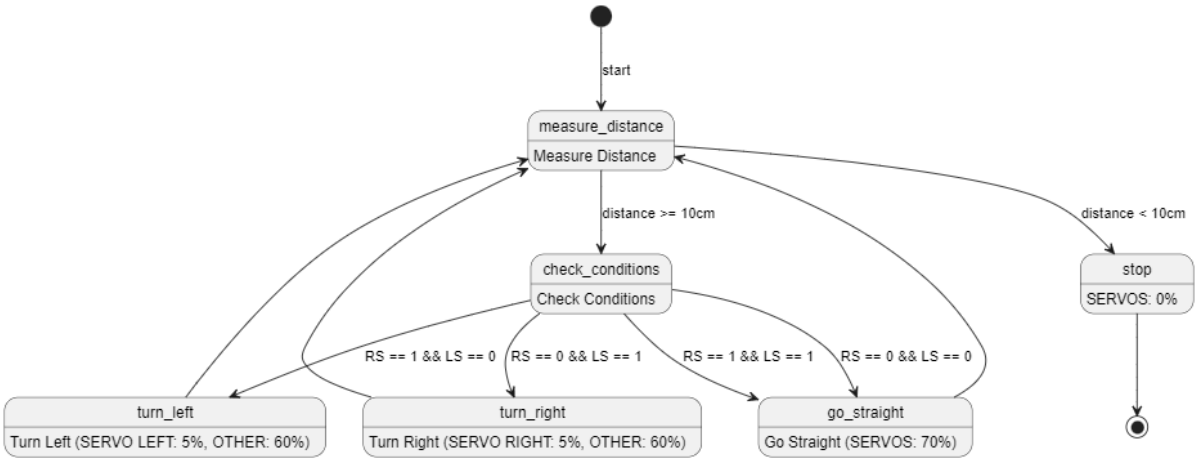


Figure 1: Phase 1 state machine.

4. Implementation Phase 1

This section of the report will be dedicated to explaining the choices made behind certain parts of the robot project implementation. These can include for example which timers were used, or why certain values for certain registers were chosen and more. Note: For the remainder of the robot project, the clock speed of each implementation will be set to 16MHz.

4.1 Servo implementation

For the servo implementation, the knowledge that was gained from the timers output assignment was applied first. Using the alternate function mapping table from the MCU datasheet**[6]** and the knowledge of setting up and using the timers**[4],** the servo implementation was then configured correctly PIN-wise.

The reasoning behind the usage of the values 32 and 45000 for the PSC and the ARR respectively was determined again using the following formula:

$$PWM\ Frequency => \frac{Clock\ Frequency}{Desired\ Frequency} = value(PSC \times ARR)$$

Since the desired frequency was a frequency of 50Hz (for a 20ms period)**[3]**, we end up with a value of 320000. We thought initially that our PSC and ARR values would be 32 and 10000 then but when read with the logic Analyzer, we noticed that these signals were not that of a 20ms period. In the end we manually calibrated it to approximately 20ms by having the ARR value be at 45000. This is all done on one timer with two separate channels.

With that being said the values that are being sent to the servo also has to be changed. Take the clockwise rotation for now. To spin the servo at 100% we need to pulse it with 1280 and 0% would be 1470. Through trial and error and using the analyser, we came to a conclusion that for the servo to spin clockwise at 100%, we would need to pulse it with the value of 2768. So this value we divided by the initial 1280. We get a constant of 2.16217.

Additionally we made another function so we can control the servo by passing the desired percentage value in the function. This is done by implementing the “map” function**[7].**

Finally, since both servos need to be pulsed by different values to act as wheels, they would need to operate both with counter clockwise and clockwise. For that two separate functions were made to control them. Next to this, we have another function so that we can stop the servos, they need to be pulsed with 1500. So both our servos will be post with that equivalent to stop them.

4.2 Ultrasonic implementation

For the ultrasonic, the knowledge that was gained from the timers input assignment was applied first. Using the alternate function mapping table from the MCU datasheet**[6]** and the knowledge of setting up and using the timers**[4]**, the ultrasonic was then configured pin wise.

The timers setup was the same as the previous assignment as well. We slow down the clock to 1MHz so we can get values of 1 microsecond. From there we can easily pulse the ultrasonic with at least 10 microseconds on the trigger pin. The output from the echo pin is then read using another timer set to input mode. From there we have an interrupt handler, constantly reading the rising and falling edges to get the final distance value. For some context, signals on the ultrasonics can be seen in the following image.

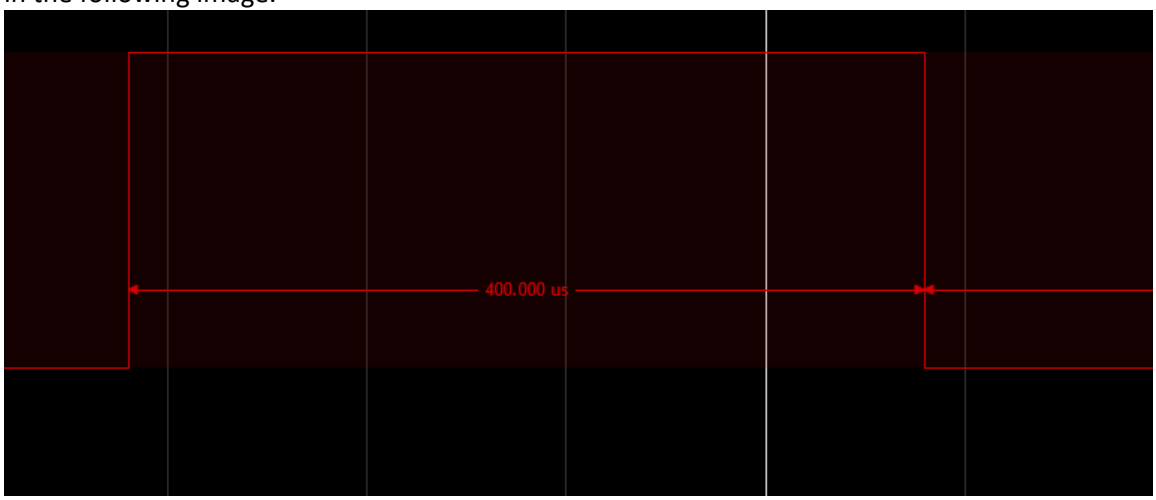


Figure 2: pulse width that is being read on the input pin

The distance is then calculated using the following formula:

$$Distance\ (in\ cm) = \frac{Time\ (microseconds)}{58}$$

4.3 infrared implementation

The infrared code is relatively the easiest one out of all of the sensors to calibrate. The general idea is when the ultrasonic is on a flat service the color sensor would “detect” something. When a black object comes across the sensor would not detect anything anymore. This is because the black object absorbs the infrared color so it cannot be reflected in order to be reached by then sensor. So with that logic, and the fact that we use black tape makes our implementation design easier. Using the knowledge we got from the pushbutton assignment, we can just use the IDR register to read the value of the pin it is connected to. We can then put in in an if-statement to confirm if the sensor isn’t detecting something. In this case it would do something.

4.4 combined implementation

The combined implementation to form the entirety of the robot was done in parts. First we combine the ultrasonic and the servo motor implementations together. You will notice an immediate change in your distance outputs. This can be recalibrated. Now that we have a servo to functions normally, and can stop at an object 10 centimeters away, it was time to work on the steering. According to the state machine design that we have, the code for the line following was implemented using the IR sensor. However there are complications with both reading the sensor and the height of the sensor on the robot.

5. Testing Phase 1

This section of the report will be dedicated to explaining the steps that were taken in order to test the completed phase 1 of the robot project.

Scenario 1:
The robot is able to follow a straight line.

Scenario 2:
The robot is able to make slight turns on bends.

Scenario 3:
The robot is able to stop when an object is detected at 10 centimeters or less from the front.

6. Design Phase 2

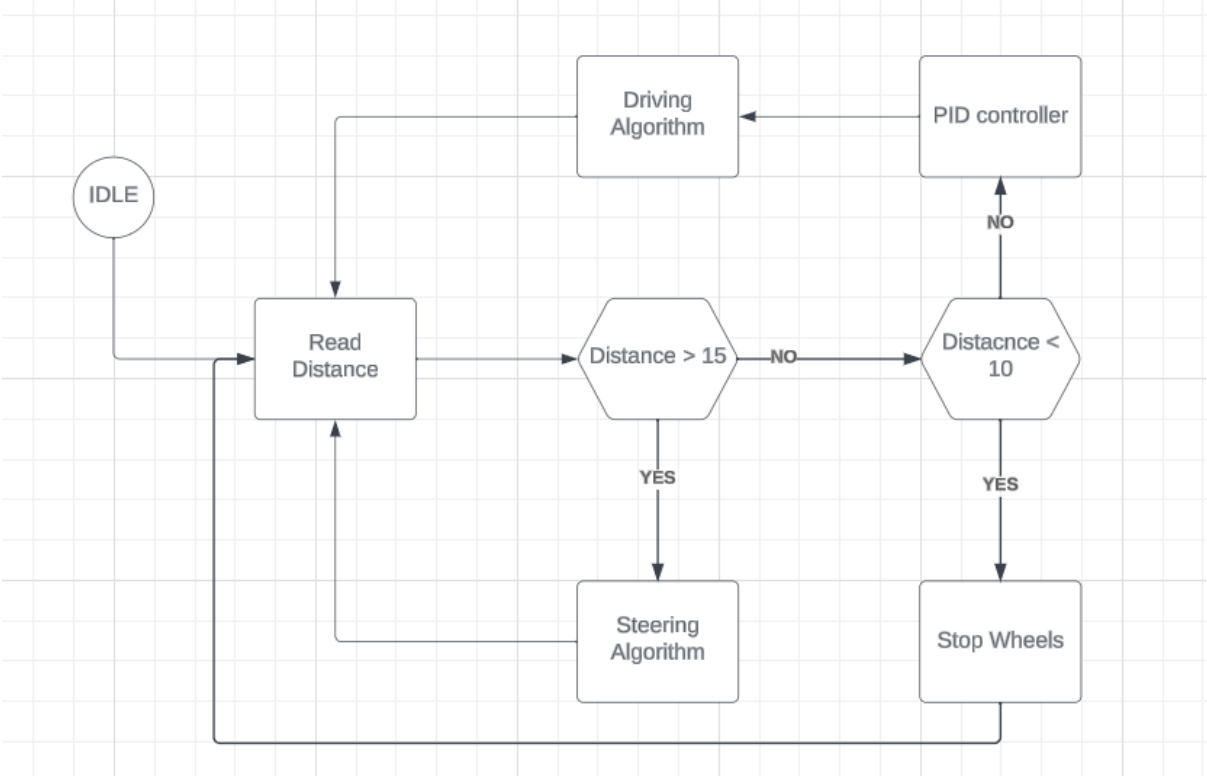


Figure 3: Flowchart representing the basic logic of the system.

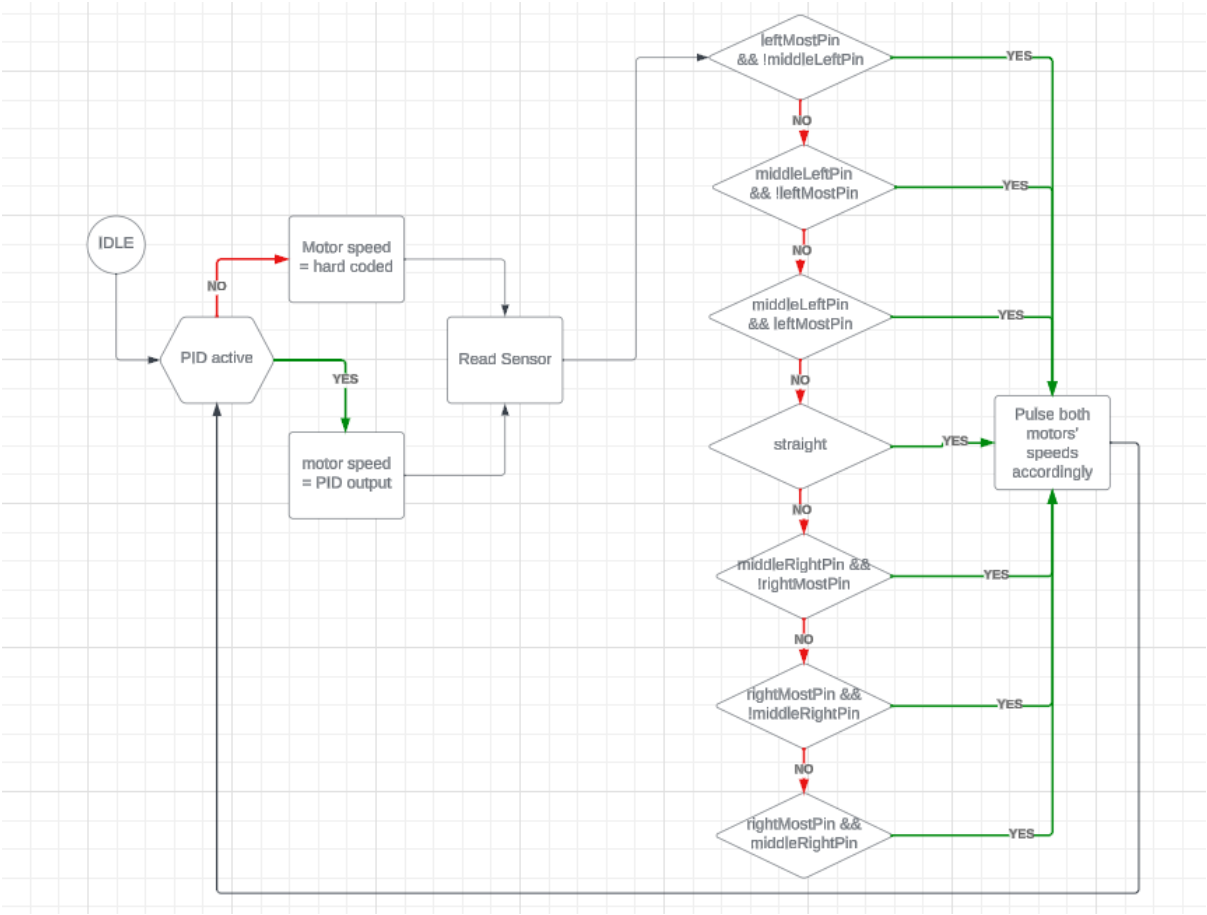


Figure 4: Flowchart representing the driving algorithm

7. Implementation Phase 2

This section of the report will be dedicated to explaining the choices mad behind certain parts of the robot project implementation. These can include why we chose a specific logic or why certain registers were chosen and more. ***Note:*** *It was decided to use a project with a clock speed of 72 MHz, since this solved a majority of the problems found.*

7.1 Progress from Sprint 1

A continuation of the first sprint brings the project to the following points:

- The robot is able to drive in a straight line.
- The robot is able to drive through a course of turns.
- The robot is able to stop when an object is detected at 10 centimeters.

7.2 Improvements made

Now that all of the problems from the previous sprint were solved, a majority of the work went into using the PID controller loop to now control the robot speed when an object is detected in front of the robot. There project is still based on a 72MHz clock. The project

7.3 PID implementation

A few aspects of the code have changed ever since introducing PID to control the servo motors. Instead of pulsing the servos manually, the servos are now pulsed with values that are derived from the PID loop that is implemented. The error within the loop is calculated with the distance that is received from the distance sensor.

A new servo function has also been declared to handle these new values. The lowest values the servos need to be pulsed with are both 1480 and 1520 for both counterclockwise and clockwise. This function will then take a PID output value of maximum 200 (the max the servos need for full speed are 1280 and 1720) to drive.

The PID loop itself will calculate the output using the derivative, integral and proportional and return this. These values are calculated using the constants for each of these variables. In order to get to these variables was trial and error. The one notable thing is that, when the P constant is changed, it will change the based speed that the servos begin to drive with. The I constant when changed, changes the rate of acceleration of the servos. The D value will handle overshooting when handling turning.

Finally, within the loop, the distance is then taken, passed to the PID controller and the output that is returned will be passed in to the new servo function. Which will take the (mapped to 200 value) and pulse the servo with the new speed based on the distance.

7.4 Applying the PID within steering.

Now that it is certain that the PID loop works as intended, it was time for integration with steering. Applying the PID to control the servo in one straight line was done. Regarding steering was a bit more complicated. The Requirement was that under a certain distance the robot should ***start*** applying PID to control the speed.

The steering was done with hard-coded values in the previous implementation that were deemed the most “Smooth” when running through a course. A solution needed to be found in order to control the servo steering when PID is applied. The max speed that was taken notice that the servo was with 60% (using the mapped servo function). The idea behind this was to use percentages when applying the speed. For example, assume in this situation PID is not being applied. To turn the robot with a hard left, then the right wheel would need to apply significant more power than the left wheel. The value used for the right wheel were 60(%) and the left wheel 0(%).

In the steering function, when not applying PID, the maximum speed variable will be set to 60. Based on the situation ,the new percentage is applied to the value. For example if the motor needs only 30% power then what is done is the max motor speed is taken and then multiplied by the desired percentage which is (50% or 0.5).

With this implementation, it was easier to apply the PID output (which is mapped from 0 to 60) as a direct value to the function now as well while keeping the old percentages. In the case where PID is being applied, the motor speed value is not set to the maximum speed value (60) but with the refined PID value that is received after mapping it. With this, the robot is now able to steer using the PID controller as well.

8. Testing Phase 2

This section of the report will be dedicated to explaining the steps that were taken in order to test the completed phase 2 of the robot project

Scenario 1:
The robot is able to perform the tasks when there is no vehicle/object in front of it. Meaning it is able to both drive, steer and stop and object of 10 centimetres. This was done by putting the robot on a hand made course at home and letting it drive with no vehicles in front of it and then putting an object in front of it to stop it.

Scenario 2:
The robot is able to apply PID to smoothly hold a distance between itself and the object/vehicle in front of it. This was done, by putting a hand as the robot is driving similarly to scenario 1 to stop it. The hand will then move so that the robot can follow it.

Scenario 3:
The robot is able to apply PID **and** drive smoothly along the course. This was testing similarly to situation two by placing the hand in front of it while also going along the course.

Scenario 4:
The robot is able to stop behind the object/vehicle both when it is applying PID and isn't. This was tested similarly situation one but now when it is applying pid the hand will also stop so the robot will also stop.

9. Reflections

***Johnson**:*

- Implemented a PID controller and loop skeleton.
- Improved/applied the PID controller and loop in the actual implementation.
- Responsible for motor controls.
- Responsible for PID steering logic.

This was a fun project to work on, in the sense that personally I love working with hardware, no matter how they are programmed. It was rewarding to see everything working according to tests in the end. Above are my contributions to this project.

The start of this project was a bit difficult since for the first part of the project my group mate was sick. When he got better, we ran into more problems not because of ourselves but faulty hardware. Once we fixed those then there were more problems which were the result of poor configuration. In the end when we were able to solve all of these issues. This project also taught me that integration in a large scale project is one of the most difficult parts of working on a project like this. In the future I plan to work on large scale projects with a bit more consideration for integration along the project cycle.

***Harm**:*

- Responsible for the steering algorithm.
- Responsible for testing the robot on hand-made course at home.
- Responsible for fixing the infrared sensors.
- Responsible for distance algorithm.

The start of this project was very troubling. At the start we had all our PoC's working separately but not together. This caused delays in delivering the required product for phase 1. During phase 2 a lot of progress was made on the problems that were holding us back as they were resolved and we could now start testing the car with all the POCs combined. This was my responsibility. At home a

course was built for the car to drive on. Using this course the values for how hard the car should turn and drive were tuned. This course was then also used to test the PID together with driving and turning around this course.

This project showed me how difficult it can be to integrate multiple PoC's as these PoC's can block each other and cause big problems. These problems cost a lot of time, the problem ended up being 2 minor details in the code. This means that during integration we need to make sure that the codes we have do not interfere with the functionality of the other code. Overall the project was good at combining all the aspects learned during this course as all the points we talked about came forward during this project.

10. Problems within the project

Throughout the initial phase 1 sprint there were some problems. In this section of the report, all of these problems are mentioned and potential solutions that we found.

Note: the way the signals are divided is the following:
BLUE CHANNEL: ultrasonic trigger pin
YELLOW CHANNEL: ultrasonic echo pin
RED CHANNEL: servo motor 1
GREEN CHANNEL: servo motor 2

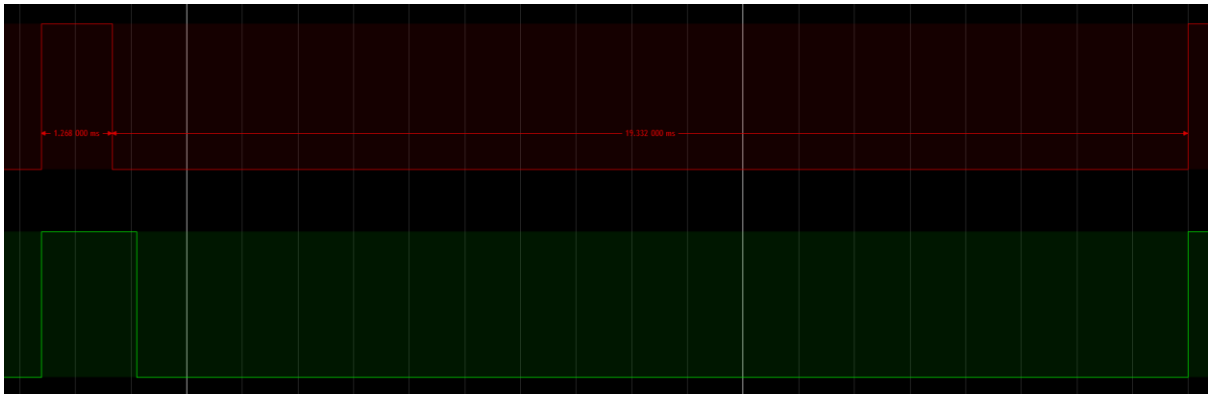
10.1 Ultrasonic inaccuracies:

The First issue that was encountered when working with ultrasonic is the fact that there is a 2 centimeter difference between the desired distance and the acquired one. For example, instead of receiving 10 centimeters in the terminal we would receive 12 centimeters. This issue was **solved** by using the 72 MHz timer instead of the 16 MHz timer like the previous assignment. This is done using the same configuration and values as the previous implementation.

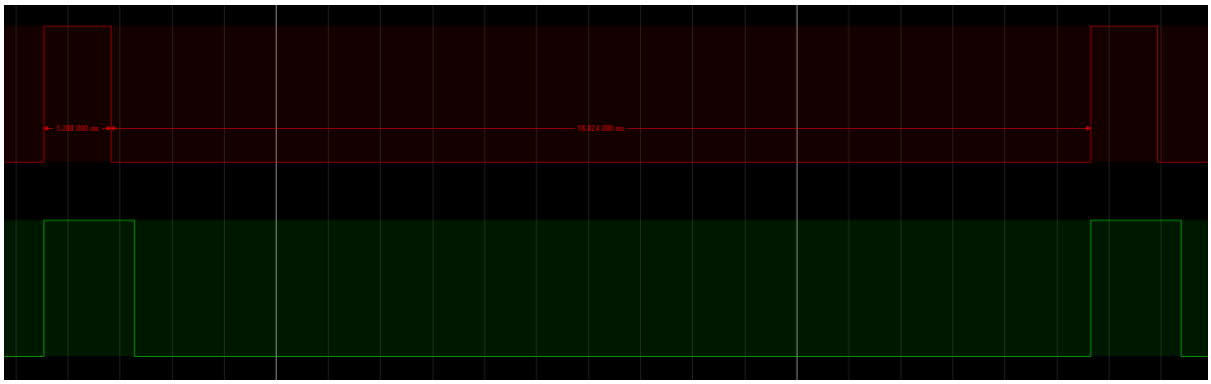
10.2 Servo calculation inaccuracy:

As mentioned in section 4.1, the same formula was used as in the previous assignment to calculate the ARR and PSC values and came with 10000 and 32. This didn't make out system pulse with our desired values either. We had to adjust the ARR to 45000 instead of 10000 and only then would the system work when it was pulsed it with the correct datasheet values. This is also after adjusting the values by multiplying them with the number 2.16217.

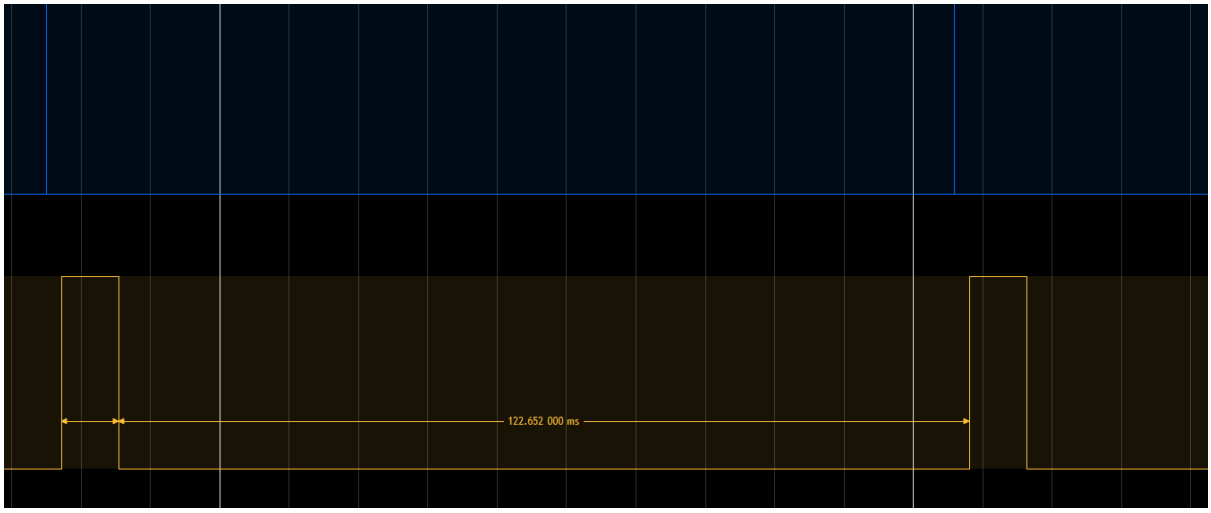
This was **solved** with the 72MHz clock, the newly calculated values for the ARR and PSC. The received values were 10000 and 144. These values made pulsing the servos much more precise. All that needed to be done was dividing the datasheet values by half and it worked perfectly. As seen on the below images.



Servo motors on 16MHz clock



Servo motors on 72MHz clock



Ultrasonic sensor under normal circumstances (72MHz)

10.3 infrared sensor inaccuracy:

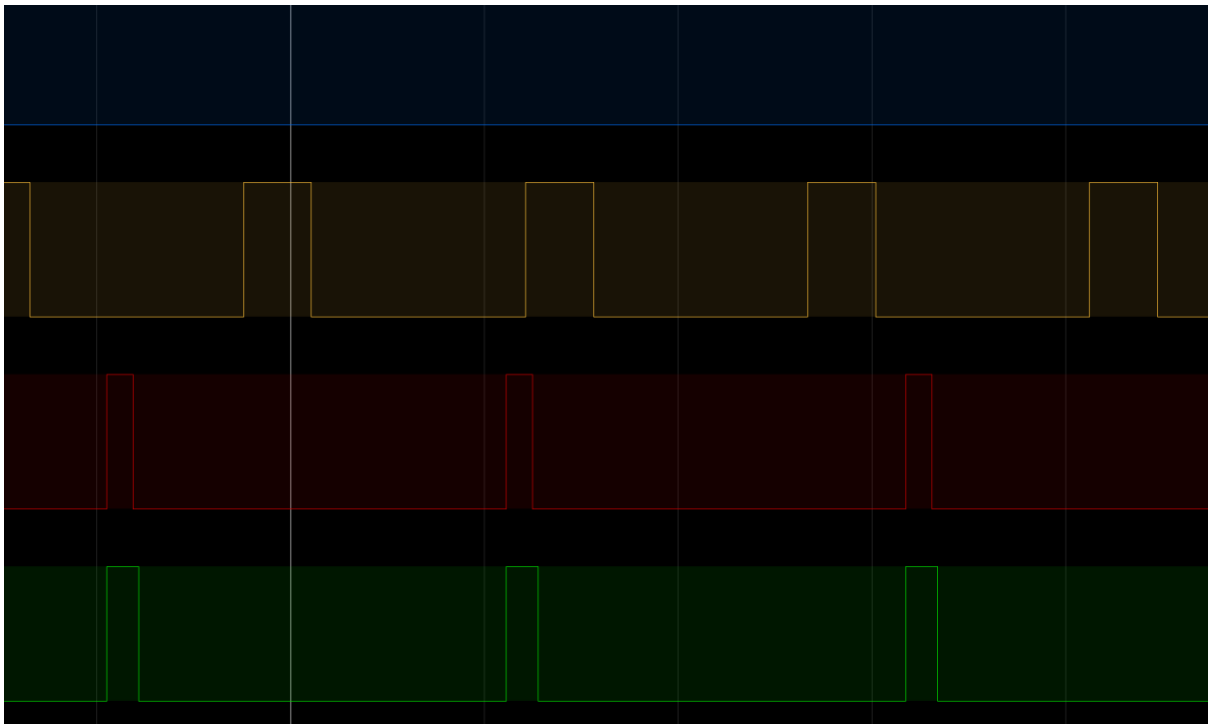
Initially we thought the problem of the IR sensor was shorting and it being to far off the ground. This ended up being a robot car specific problem as we got another car with the same sensor where this problem was not found. This problem has thus been solved by simply using a different car.

The next problem that was found that one of the pins did not detect the IR sensor pin go to zero, the problem was not found on this as the method of detecting this was the exact same as the other 4 pins on the sensor. To get around this problem as different pin was used, this was then changed in the code and the problem was solved as this pin perfectly detects the pin going zero.

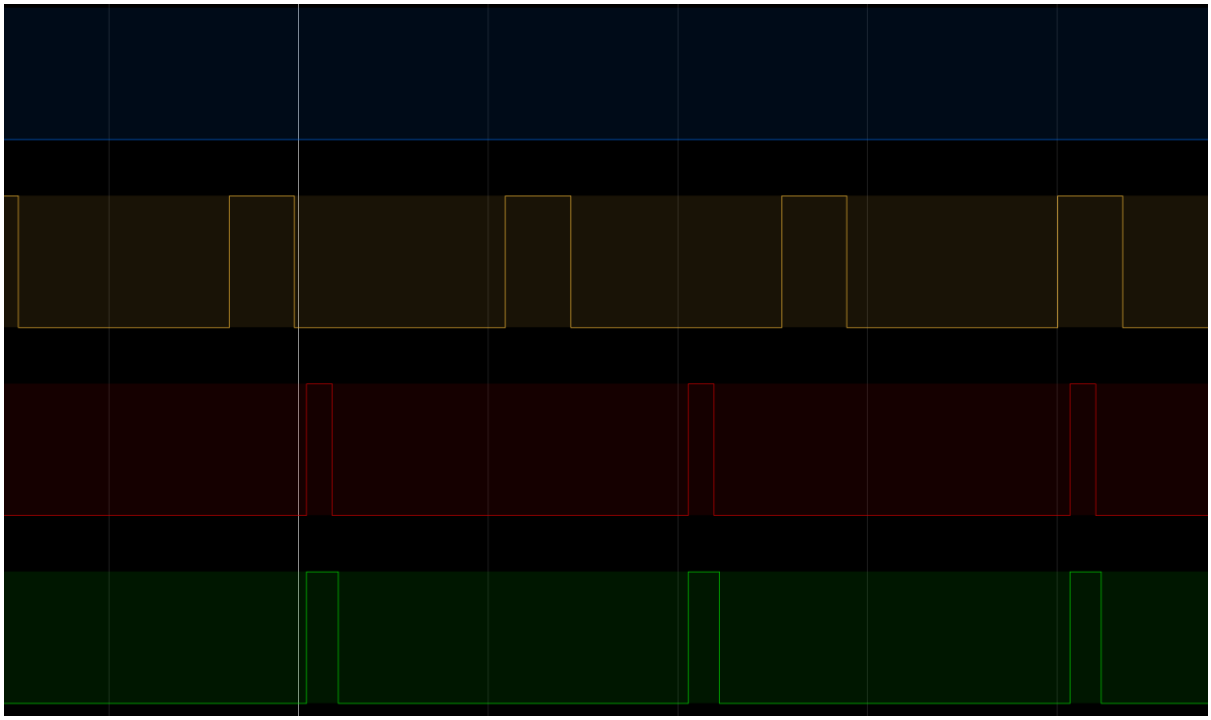
The last problem that we have with the IR sensor is now that one of the pins is not seeing black far enough, the tape was lifted around 5mm which caused the pin to go to zero. For this the problem has not been found. Incase the sensor does short we have placed washers on the screw that mounts the sensor to the car, this creates space between the metal of the car and the sensors pins, this should assure us that the pins are not accidently shorted.

10.4 Combined prototype:

This is where the big problems of the entire project began. The motors were working but the timers of the ultrasonic sensor are not. Separately each sensor works perfectly on 72MHz without problems. Together, they do not. Below, the measurements can be found of both the implementations working. Both the 16MHz and the 72MHz clock frequencies. We believe that somehow, something is block a register and in doing so the ultrasonic sensor timers do not function properly. It can be seen that although the motors are working correctly, the ultrasonic does not. The only reason that it is getting signals in the first place is because of something within the hardware itself. It was noticed that when powered on, you can already read values from the echo pin. The only issue is these values are inaccurate. That is why the yellow channel has signals on it. This problem was **solved** with the use of the 72MHz clock.



16MHz clock combined prototype.



72MHz clock combined prototype.

10.5 Volatile distance variable:

At first the implementation worked as intended where it was possible to use the distance variable by printing it to the terminal using the interrupt itself. In an ideal implementation the distance variable needs to be used in the main loop to be used in the PID controller loop. This was not possible. The variable could not be printed in the main loop. Despite the variable being volatile, it was only able to be used in the interrupt handler. This issue was **solved** by instead of setting "TIM_DIER_UIE" into the DIER register (which is updating the interrupt handler), the "TIM_DIER_CC1IE" macro is set into that DIER register. This way we are enabling the interrupt on channel 1 rather than updating it**[4]**.

10. Bibliography

[1] – *Ultrasonic Sensor Datasheet. (n.d.).* https://www.handsontec.com/dataspecs/HC-SR04-Ultrasonic.pdf

[2] - *Reflective optical sensor with transistor output - Vishay Intertechnology. (n.d.).* https://www.vishay.com/docs/83760/tcrt5000.pdf

[3] - *Parallax feedback 360° high-speed servo (#900-00360). (n.d.).* https://cdn.sparkfun.com/assets/8/5/e/4/e/DS-16047.pdf

[4] – *STMicroelectronics Reference Manual. (n.d.).* https://www.st.com/resource/en/reference_manual/rm0316-stm32f303xbcde-stm32f303x68-stm32f328x8-stm32f358xc-stm32f398xe-advanced-armbased-mcus-stmicroelectronics.pdf

[5] - *UM1724 user manual - stmicroelectronics. (n.d.).*
https://www.st.com/resource/en/user_manual/um1724-stm32-nucleo64-boards-mb1136-stmicroelectronics.pdf

[6] - *ARM® cortex®-STMicrocontrollers Datasheet. (n.d.).*
https://www.st.com/resource/en/datasheet/stm32f303re.pdf

[7] - *Map function. map() - Arduino Reference. (n.d.).*
https://www.arduino.cc/reference/en/language/functions/math/map/