

## Project Final Report

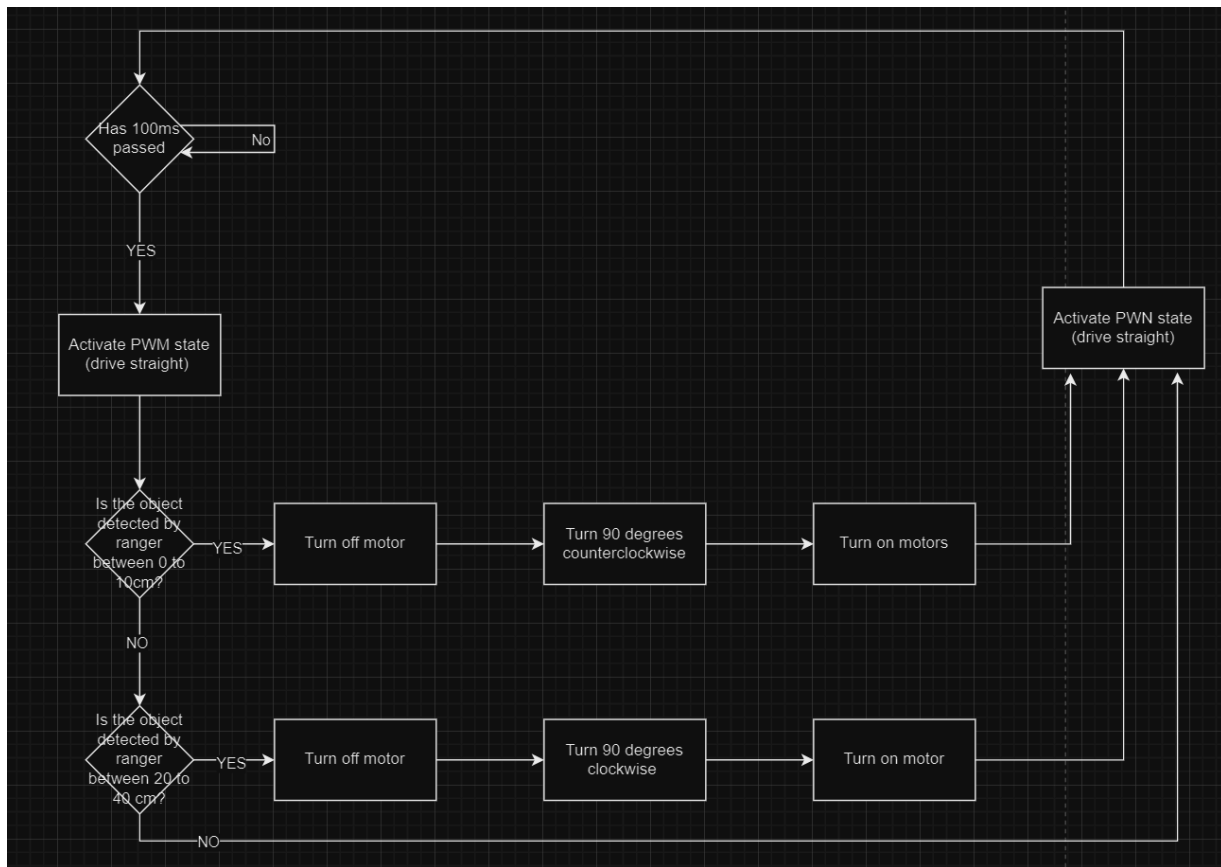
### Project Introduction and Description:

Problem: The racetrack contains a bunch of inner barriers that cars must drive around and complete the track.

### Proposed and Final Solutions:

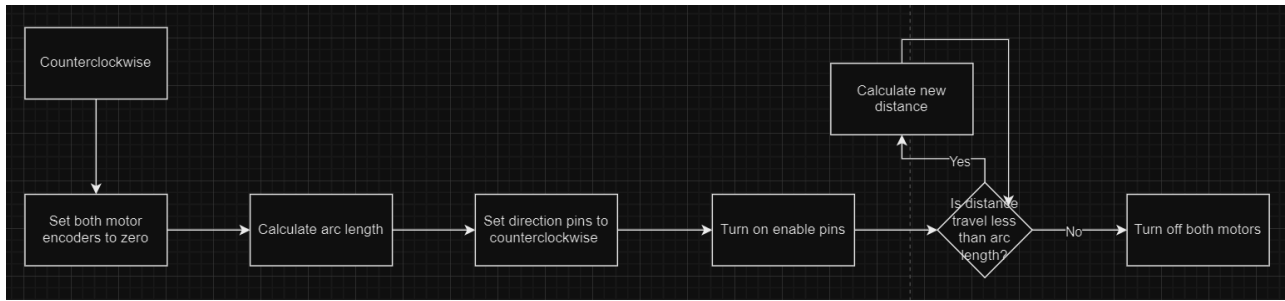
Solution: We will implement a ranger to the car that is facing upward. Since the ranger is facing upward, we will have a ruler as the “object” to block the wave giving us a distance value. When the ranger detects a large value (greater than 40cm), it would signal the car to drive straight. Otherwise, there are 2 if statements that will signal the car to turn 90° clockwise or counterclockwise. For the car to turn counterclockwise, the ranger value needs to be less than 10 (the rule in front is less than 10 cm away), otherwise if the ranger value is within 20 to 40 cm, then it will turn clockwise. Overall, we are guiding the car to turn when it needs to, otherwise just go in a straight line.

## High-Level Design:



### Main:

- After every 100ms, it will start the main code for this project that is within an infinite while loop. Once 100ms has passed, it will run the Wheel Speed Control that is basically the same as lab 3 implementations.
- For the wheel speed control for the PWM, we already set a desired speed for the whole system so the desired speed is a constant, while the differential speed is set to 0, since we don't need to turn at an angle.
- After applying the compare value, the system will record the ranger value and then it will face the first if statement condition (is the ranger value  $\leq 10$ ).
  - If yes, then it will turn off the motor in order to let the counterclockwise function to run. Then resume the system to turn on the motor and apply the wheel speed control state.
  - If no, then go to the seconds if condition
- Run the second if statement condition (is the ranger value  $> 20$  and  $< 40$ ).
  - If yes, then it will turn off the motor in order to let the clockwise function to run. Then resume the system to turn on the motor and apply the wheel speed control state.
  - If no, then go to the wheel speed control state



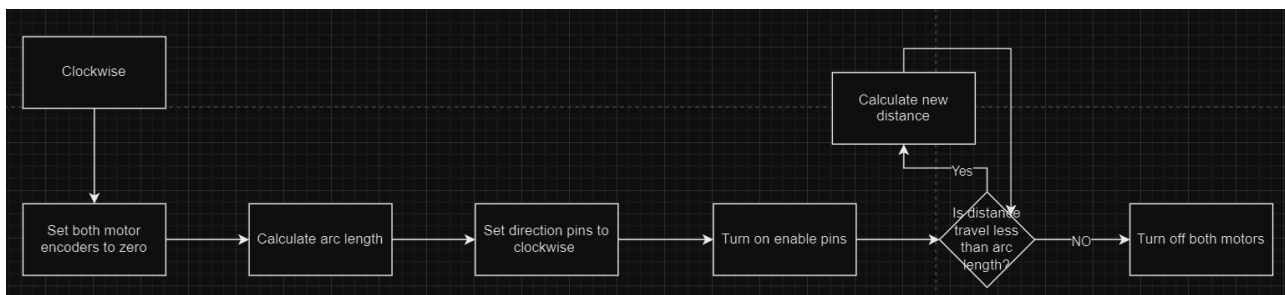
This is the counterclockwise function, used in the main.

Set the distance variable as well as reset the encoder to 0. Then calculate the arc length from the wheel radius and circumference. From the H-bridge implementation, set the GPIO ports to be counterclockwise and then turn on the enable pins.

Then it will run the while loop to make the car turn to 90 degrees based on the condition of if the distance is less than the arc length.

- If yes, then calculate the new distance based on the circumference and encoders
- If no, then it's done turning 90 degrees counterclockwise

Lastly, set the motors off.



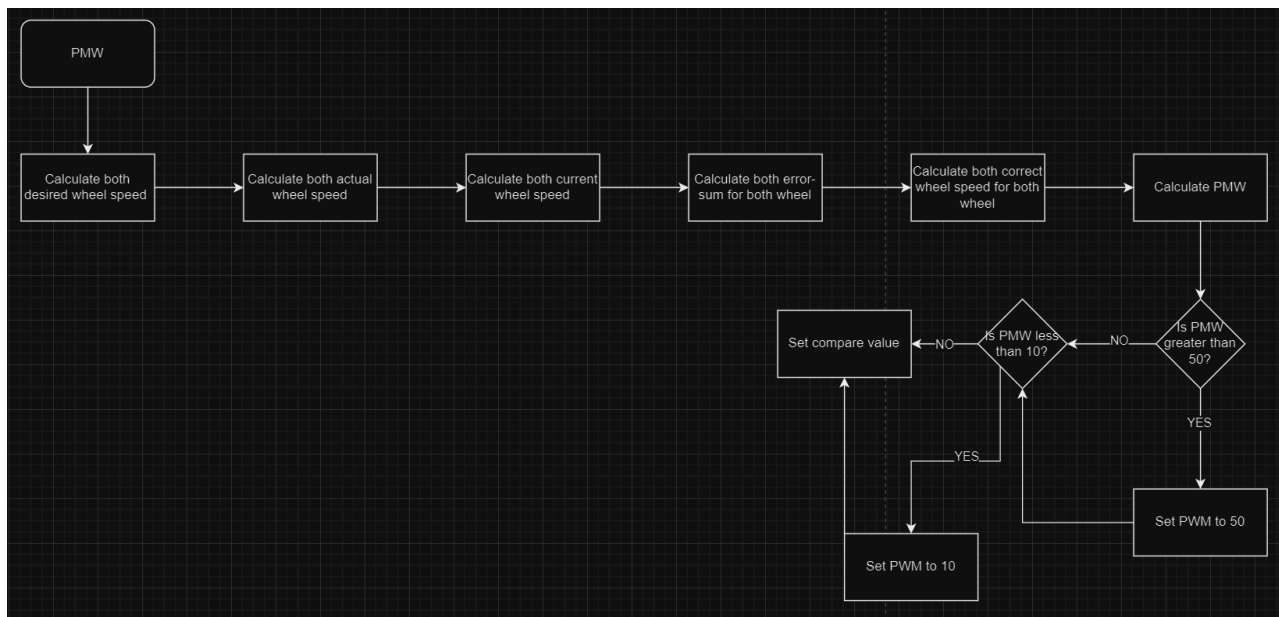
This is the clockwise function, used in the main.

Set the distance variable as well as reset the encoder to 0. Then calculate the arc length from the wheel radius and circumference. From the H-bridge implementation, set the GPIO ports to be clockwise and then turn on the enable pins.

Then it will run the while loop to make the car turn to 90 degrees based on the condition of if the distance is less than the arc length.

- If yes, then calculate the new distance based on the circumference and encoders
- If no, then it's done turning 90 degrees clockwise

Lastly, set the motors off.



This is the Wheel Speed Control or the PWM function.

This is the same implementation used throughout lab 3-5 that uses capture mode and PWM for the encoders. Not going to go too detailed on it as it calculates the desired and actual wheel speed to find the current speed and calculate the sum of errors. Then find the corrected speed into the PWM value and enforce the min and max limitations on it. Lastly, set the compare value.

## Low-Level Design:

Global variables/prototypes/etc.

main:

Initializations

Enforce a startup delay of 1 second

Turn on the motors

Set desired speed to a certain speed

Set differential speed to 0

Start 100 ms timer here!! <-- Important to start AFTER 1 second delay

Infinite while loop:

If 100 ms has passed:

<-- Start Wheel Speed Control -->

For Each Wheel:

calculate desired wheel speed as desired speed +/- differential speed

calculate actual wheel speed use trendline equation from figure for current speed

calculate current speed error from the difference of the desired & actual wheel speed

add current speed error to an "error sum" variable

calculate the corrected speed using the discrete equation given

convert the corrected speed (duty cycle) into a compare value

enforce minimum and maximum limits on the compare value

apply the compare value

<-- End Wheel Speed Control -->

Read the ranger value (and request a new measurement)

If ranger value <= 10

Turn motors off

<-----Start turning counterclockwise----->

Set distance and encoder for both wheels to 0

Calculate the arc length

Set the H-bridge to turn the pins to counter-clockwise direction

Turn the motor on

While loop (distance <= arc length)

Calculate the new distance with the circumference & encoder from both wheels

Turn motor off

Set motor to forward

<-----End turning counterclockwise----->

Set ranger value to a high number to indicate driving straight

If ranger value > 20 and <= 40

Turn motors off

<-----Start turning clockwise----->

```
    Set distance and encoder for both wheels to 0
    Calculate the arc length
    Set the H-bridge to turn the pins to clockwise direction
    Turn the motor on
    While loop (distance <= arc length)
        Calculate the new distance with the circumference & encoder from both wheels
    Turn motor off
    Set motor to forward
    <-----End turning clockwise----->
    Apply delay_cycle to wait for couple of second
```

Other Functions:

```
//might turn the counter, clockwise, and PWM into a function, but it's still the same code as above
```

GPIO Init (usual stuff)

Timer Init:

```
PWM Timer+CCRs (30 kHz, OUTPUT_RESET_SET <- makes control easier)
Encoder Timer+CCRs
100 ms Timer
```

I2C Init:

readRanger:

```
All same from Activity 13
```

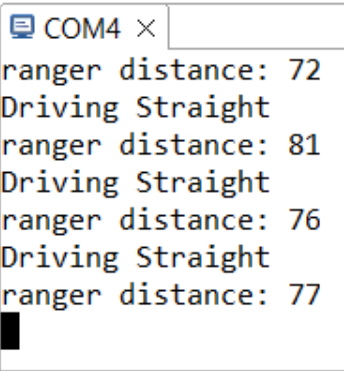
Encoder ISR:

```
Same as in Lab 4
```

100 ms Timer ISR:

```
Mark that 100 ms has passed using a global flag
```

### Verification of Operation:

 <pre>COM4 x ranger distance: 72 Driving Straight ranger distance: 81 Driving Straight ranger distance: 76 Driving Straight ranger distance: 77 █</pre>	<pre>ranger distance: 104 ranger distance: 12 ranger distance: 9 Turning counter-clockwise ranger distance: 101</pre>	<pre>ranger distance: 53 ranger distance: 21 Turning clockwise ranger distance: 107</pre>
Since the ranger didn't detect any object in the way, the value would be high (more than 40 cm away), indicating the car will only drive straight and does not perform any turns.	The ranger detects an object less than 10 cm away (9 cm) which triggers the counter-clockwise turning function. After completing the function, the ranger detects something greater than 40 cm (101 cm) resulting in the car to resume driving straight.	When the ranger detects an object between 20 to 40 cm (21 cm), the clockwise turning function is triggered. After completing the function, the ranger detects something greater than 40 cm (107 cm) resulting the car to resume driving straight.

### Conclusion:

The second solution (ranger) works as intended overall, but there was one slight unexpected behavior that we have noticed. There was a slight issue with what we had with the ranger as after it turned either counterclockwise or clockwise, it would then repeat itself one more time. When I was trying to figure out why, I used a print statement on the ranger to see what was happening. The problem was that after the reader read the ranger once and performed the function, the reader's response was too fast and it read another value similar to the previous for the next run. So the solution that I proposed was to use a while loop that will read the ranger 3 times, but will take the last value to ensure that it acts as a delay, making sure that it runs the function once. So once the problem was fixed, the car began to work as intended. Overall, what I think the code could be improved on is readings of the ranger so it can act faster and more fluently for the race without any delays. The delay is there to add a 1 second for the car before making a decision and lets the car actually turn 90 degrees. Another thing that I think could be improved is the car moving straight because at a higher speed (desired\_speed = 50), the car does drive at a slight angle so I would like to improve that but overall at a reasonable speed, the car does drive straight. The turning function worked as intended since it's the same code used during Lab 3 and the pwm function and timers was the same as lab 5.

We would like to point out that our first solution was using the bumper but we had an issue on figuring out how to use the bumpers to turn 90 degrees clockwise or counterclockwise without touching the bumpers. We didn't know that we could place makers beforehand and because of this we decided to use the ranger as a way for us to turn the car 90 degrees clockwise or counterclockwise.