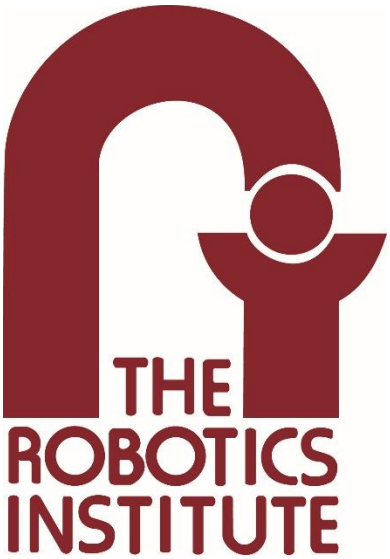

MRSD Project Course

Team I – AIce

Autonomous Zamboni Convoy

Individual Lab Report 03

Author

Nick Carcione

Teammates

Rathin Shah

Yilin Cai

Jiayi Qiu

Kelvin Shen

March 3, 2022

Contents

1. Individual Progress

My progress these past two weeks has been largely focused on getting the RC car that we plan to use for initial testing up and running. This included two main activities: controlling the steering angle of the wheels and controlling the speed of the wheels/car.

The first of these two activities that I tackled was the control of the steering angle. Originally, the plan was to develop an analytic mapping that would convert a desired slip angle into the steering angle of one of the front wheels using formulas derived from standard Ackermann geometry. This steering angle would then be converted into the angle that the servo motor that actuates the steering linkage needs to be at. However, I quickly ran into issues trying to accurately measure and model the physical steering system on the RC car, so instead I settled for a mapping between the desired steering angle and the necessary servo angle. This map was created by measuring the maximum angles that the front right tire could reach. These angles were measured by suspending the car on its side, physically moving the tire to a neutral (straight forward) position, and securing and zeroing an electronic level to the tire. With this setup, a sketch of which is shown in Figure 1 below, the maximum angles that the front right wheel could move to could be found. These values are reproduced in Table 1 on the next page. When defining and using these angles in the Arduino software, steering angles that cause the vehicle to turn left were defined to be negative and those that caused the vehicle to turn right were defined as positive. The map from steering angle to servo angle was split so that the positive and negative steering angles had their own mappings.

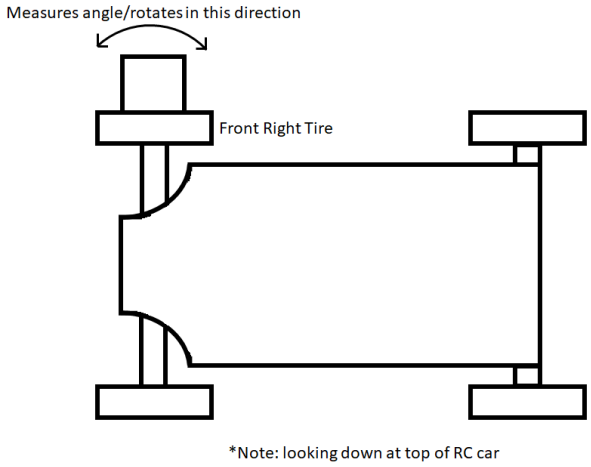

Figure 1: Setup used for measuring maximum steering angles

Table 1: Maximum steering angle values for the front right wheel and the corresponding servo angles

Angle Defn.	Angle Value [°]	Servo Angle [°]
Max. Turn Left	(-) 43	90
Straight	0	135
Max. Turn Right	33	180

The second major task involved in getting the RC car up and running was learning how to control the speed of the wheels. There were plenty of challenges associated with this and they are detailed in the "Challenges" section later in this report. The main challenge I faced was learning how to interface with the ESC that the vehicle came with. Even after trying the code that the teams from 2016 and 2019 used to control this RC car, I was either unable to get the wheels to move at all or, at best, move in a strange and unpredictable way. I eventually noticed a pattern that the wheels would only rotate when a non-zero speed command was sandwiched between zero-speed commands. After trying out some other tutorials found online and doing further testing, I learned that the ESC is only armed after it receives a zero-speed command (i.e., it must receive a zero-speed command before executing non-zero speed commands). I further realized that the PWM signals the previous teams were sending to the ESC were causing it to draw too much power from the PSU. When this would happen, the voltage would suddenly drop below the constant 11.1 V it was set at, and the system would shut off. To avoid this from happening, I found the signals that corresponded to the shut off points and bounded the PWM signals to be within these values. The important PWM values found during testing are included in Table 2 below.

Table 2: Important PWM values and the physical phenomena they relate to

Variable	PWM Value
Max. Speed - Reverse	1390
Min. Speed - Reverse	1456
Brake (Speed = 0)	1500
Min. Speed - Forward	1534
Max. Speed - Forward	1600

The meaning behind these values should also be quickly explained. All of these PWM signals are used with the writeMicroseconds() command from the Arduino Servo library. Sending a PWM signal of 1500 causes the wheels to stop turning and is also the signal required to arm the ESC. Around 1500 is a dead ban of signals that do not provide enough power to reliably actuate the motor. This dead ban stretches from 1456-1534. To move the vehicle forward, commands greater than 1500 need to be sent. As just mentioned, the vehicle begins to creep forward at a PWM signal of 1534 and reaches maximum forward speed at a PWM signal of 1600. To move the vehicle backward, commands less than 1500 need to be sent. The vehicle begins to creep backward at a PWM signal of 1456 and reaches maximum backward speed at a

PWM signal of 1390. Desired speeds are mapped within these ranges (1534-1600 for positive speeds and 1456-1390 for backward speeds) to avoid the dead band around 1500.

The final circuit used that integrated both steering and speed control is shown below in Figure 2. Using this circuit, it is possible to both steer the wheels and control their speed using either the potentiometers (to simulate the Zamboni being manually driven) or by sending in commands via the serial monitor (to simulate getting commands from the low-level controllers). The button is used to switch between these two modes.

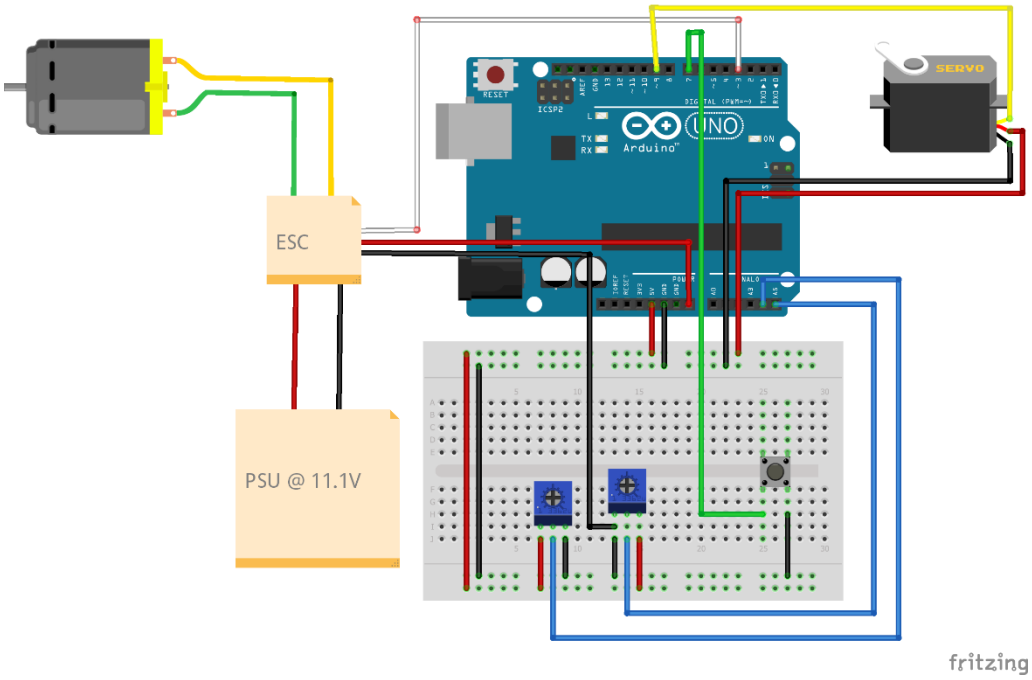

Figure 2: Circuit schematic for steering and speed control

It is also worth noting that, from my experience with testing the car, a specific "start-up" sequence should be followed to ensure that the ESC functions as intended. The recommended steps are included in Figure 3 below.

If Testing Code/Using Serial Monitor

1. Make sure ESC manual switch is set to OFF
2. Connect ESC power lines to PSU/battery
3. Connect Arduino to laptop/battery
4. Turn PSU on (if using)
5. Upload code
6. Switch ESC manual switch to ON
    1. Should hear 3 short beeps followed by 1 long beep
7. Open Serial monitor (if using)
8. System starts in "Manual Mode" (using potentiometers)
9. Press button to switch to "DBW Mode' (using Serial monitor commands)

General Use

1. Make sure ESC manual switch is set to OFF
2. Connect ESC power lines to battery
3. Connect Arduino to battery
4. Switch ESC manual switch to ON
    1. Should hear 3 short beeps followed by 1 long beep
5. ESC is now armed and ready to operate

Figure 3: Recommended steps for starting the RC car

2. Challenges

The first challenge that I faced in learning to control the RC car centered around the steering angles of the wheels. Like the Zamboni, the RC car has an Ackermann steering geometry, so the two steered (front) wheels have different but related steering angles when turning. The angles of the front wheels are controlled by a single servo motor that actuates the steering linkages. The original goal was to develop an analytic function that mapped a desired vehicle slip angle to a desired steering angle for one of the wheels and then mapped the desired steering angle into a desired servo angle. However, upon examining the RC car, I found that the links where of rather arbitrary lengths and that many of the links connected at difficult to measure angles. I was unable to find any drawings or documentation for this RC car online, so developing an accurate analytic steering-angle-to-servo-angle function became unrealistic.

Controlling the speed of the wheels also posed a considerable challenge. At first, I tried to control the ESC using code from the previous two MRSD teams that used this platform. However, in both cases, the wheels would never begin to spin no matter what command I sent to the ESC. Eventually, I began to get the wheels to spin but only if I sandwiched the speed command in between brake (or 0 speed) commands (the wheels would only begin to rotate after the second brake command). Even then, the visually observed speed did not seem accurate to the desired speed. After some digging online, we found a tutorial where someone used the same ESC we had, and we were able to use their circuit and code to learn how to successfully control the motor speed. Another challenge that we faced with the wheel speed is that the DC motor currently on the car does not output encoder pulses, so we have no way of measuring the current speed for feedback control or getting encoder odometry data for our velocity estimation/localization package.

Finally, a logistical challenge that we are currently facing with the car is that the battery it came with (leftover from its last use in 2019) quickly began to swell and bulge. To avoid a battery runaway event, we decided it was best to remove the battery and order a new one, which prevents the car from driving around freely until the new battery arrives.

3. Teamwork

<u>Rathin Shah</u>

Rathin worked with Yilin and Jiayi to get the follower Zamboni to follow a path in the Gazebo simulation. His efforts focused on developing the path following logic for the Zamboni utilizing the waypoints generated by Jiayi and a vehicle controller that he developed in Simulink. He also assisted me in finding the maximum steering angles of the RC car so that we could create a map between steering angle and servo angle. Additionally, Rathin designed and 3D printed a mount for the RealSense camera.

Yilin Cai

Yilin worked with Rathin and Jiayi to get the follower Zamboni to follow a path in the Gazebo simulation. His efforts focused on making the reaction/performance of the Zamboni realistic. He modified the dynamics parameters of the Zamboni in URDF to better reflect their real values. He also tuned the values of the PID controller to improve the Zamboni's reaction when it would receive new waypoints. Yilin integrated the waypoints generated by Jiayi into the simulation and visualized the results.

Jiayi Qiu

Jiayi worked with Rathin and Yilin to get the follower Zamboni to follow a path in the Gazebo simulation. Her efforts focused on estimating the velocity of the leader and generating the waypoints for the follower to follow. She developed, tuned, and integrated an Extended Kalman Filter (EKF) to estimate the velocity of the leader based on its position and heading relative to the follower. She also created and published waypoints containing position, heading, and velocity values for the follower to follow.

Kelvin Shen

Kelvin continued developing the perception subsystem with a focus on the RealSense camera. In simulation, he remade the URDF of the ArUco marker board using custom Gazebo textures to achieve better, more consistent detection in simulation. He then mounted the marker board to the back of the leader Zamboni and tested the accuracy of the pose estimation algorithm when both the leader and follower Zambonis are moving and when the board is partially occluded. He also tested the pose estimation algorithm on the physical RealSense using a printed-out version of the marker board.

4. Plans

My goals for the next Progress Review are to build upon my previous work on the RC car and get it to the point where it can follow waypoints. Achieving this benchmark would allow us as a team to begin testing some of our algorithms and controllers as well as some of the hardware/sensors. I also plan on helping with the testing and validation of the RealSense camera and IMU on the RC car. Since I have spent the most time with the RC car, my role in these tests will likely be running and maintaining the vehicle during data collection.