Team Apollo Advanced Robotics Final Report

Michael Becerra
Dept. of Mechanical Engineering
University of Colorado Boulder
Boulder, United States
michael.becerra@colorado.edu

Tsuening Lee
Dept. of Aerospace Engineering
University of Colorado Boulder
Boulder, United States
tsuening.lee@colorado.edu

Tyler L'Hotta
Dept. of Mechanical Engineering
University of Colorado Boulder
Boulder, United States
tyler.lhotta@colorado.edu

Thanushraam Suresh Kumar
Dept. of Robotics
University of Colorado Boulder
Boulder, United States
thanushraam.sureshkumar@colorado.edu

Andrey Lototskiy
Dept. of Computer Science
University of Colorado Boulder
Boulder, United States
andrey.lototskiy@colorado.edu

***Abstract*—This paper explains Team Apollo's approach, results, and challenges to the Autonomous Vehicle Competition. As a result of hardware damage, Team Apollo's approach to the challenge features as well as navigation relied soley on the DeepRacer's camera sensor and OpenCV. Team Apollo was able to achieve an average track time of 2:08 minutes. The results for each split are as follows: 1:58, 2:15, and 2:12. Additionally, Team Apollo were able to complete the stop sign, speed limit signs, and vehicle telemetry interface challenge features. Lastly, challenges faced by the team included hardware, software, and team effort.**

I. Introduction

Autonomous vehicle racing has a rich and storied history, starting as far back as the early 1980s with line-following robotic competitions. As technology has improved, robotic competitions have been able to expand beyond this premise to a multitude of avenues. For the purposes of this class, teams of students make a return to tradition and race an indoor course. In addition to autonomous navigation, other challenge features are to be implemented to further test the capabilities of the DeepRacer.

During the initial project proposal stage, two possible packages of challenge features were discussed. This was done to combine features that had synergistic elements with the main overarching five-point feature chosen. Any progress made towards the main navigation goal would also then trickle down to the lower point features, allowing the team to make parallel progress. This was especially important given varying levels of engagement in the team, which forced efficiency as a main focus. For both packages and just in general, the telemetry data interface seemed like a fairly simple and valuable thing to have, so it ended up making an appearance in both approaches.

The first package's five-point challenge was the SLAM package implementation. This was initially looked at due to interest on the team in learning and implementing SLAM due to its widespread use in the field of robotics. In addition to this challenge, avoiding an obstacle placed on the track was the two-point challenge to be implemented. With the use of LiDAR in mind for the SLAM approach, the team felt that avoiding an obstacle placed on the track would be fairly synergistic.

The second package's five-point challenge was the camera navigation. This was looked at because it would allow the team to gain some experience with computer vision, which is also a big aspect in the field of robotics. This would allow the team to learn and use OpenCV, a widely used open source library for computer vision and image processing. The accompanying challenge proposed was the target finding feature, which would have the DeepRacer identify and drive over pieces of green tape on the track. As both features relied heavily on computer vision, this felt like a fairly reasonable combination of features to help reach the eight-point goal.

The plan following the approval of the proposal of the team was to split the six-person team into two three-person subteams, where both teams would examine the difficulty and probability of implementing either package. However, due to time constraints and uneven participation, only one package was initially explored, though this was ultimately not pursued.

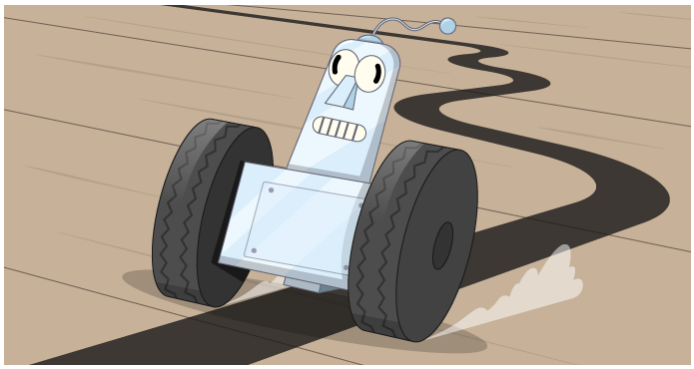

Fig. 1. Line Following Robot [1]

Due to an incident in which the LiDAR ended up breaking, the team was forced to pivot toward challenge features that

could be done soley through the use of the camera. Following a discussion on what features should be done instead, the list of features settled on were camera navigation, stop signs, speed limit signs, and telemetry data interface. This package of features is fairly similar to the the second package mentioned above with the main change being made to the accompanying features totaling two points. At this point, really understanding the level of effort available from the team, a simple line following navigation approach seemed the most possible for the five-point challenge. As a result, having to juggle line following with driving over targets on the track seemed like further complexity which was beyond the bandwidth of the team. To compensate for the two points needed, the stop sign and speed limit sign features were chosen as they seemed fairly similar and approachable to implement.

The rest of the paper describes the DeepRacer system, Team Apollo's implementation approaches, results, and challenges faced.

II. Description of System

A. Raspberry Pi 4

Fig. 2. Raspberry Pi 4 [2]

For the DeepRacer's central processing unit, we opted to use a Raspberry Pi 4. To facilitate seamless collaboration across our engineering team, we established secure SSH connections to the Raspberry Pi through a VPN server, using Tailscale. This setup allowed one member to directly interface with the hardware, while enabling other team members to remotely connect and collaborate in real time. This approach ensured efficient communication and synchronized efforts, even when working from different locations.

B. Raspberry Pi Camera Module

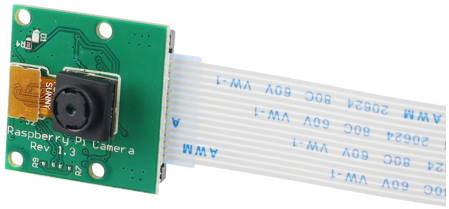

Fig. 3. Raspberry Pi Camera Module (5mp OV5647) [3]

For the visual capabilities of the DeepRacer, we integrated an RPi Camera Module. The camera was mounted in a way that positioned the field of view (FOV) with the horizon line near the center. This placement provided an optimal balance between accurate object identification in the near field and the ability to capture distant visuals, ensuring that the vehicle could effectively navigate both close obstacles and broader track features.

C. LiDAR

Fig. 4. Slamtech RPLIDAR A1M8 LiDAR Scanner [4]

The LiDAR system typically operates by emitting laser pulses and measuring the time it takes for them to bounce back after hitting objects in the environment. This data is used

to create a detailed 3D map of the surroundings, allowing the system to detect obstacles and navigate the environment with precision. The LiDAR sensor generates a point cloud, which is processed to detect distances and shapes, providing the vehicle with real-time spatial awareness. However, due to hardware issues that arose during the project, the LiDAR system was ultimately not utilized during the final runoff. Instead, the team relied solely on the DeepRacer's camera sensor and OpenCV to achieve navigation and obstacle detection, bypassing the LiDAR functionality for the competition.

D. Steering Servo

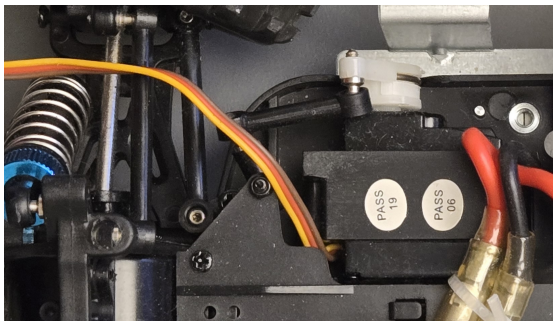

Fig. 5. 15 kg Servo and Steering Linkage

A single 15 kg servo is responsible for controlling the steering of the DeepRacer. The servo rotates a shaft that drives a simple two-bar steering linkage, which is directly connected to the tie rods. This system allows the servo to adjust the angle of the front tires, enabling precise steering control. The servo is powered and controlled directly by the Raspberry Pi (RPi), which sends the necessary control signals to ensure responsive and accurate steering adjustments during operation.

E. Drive Motor

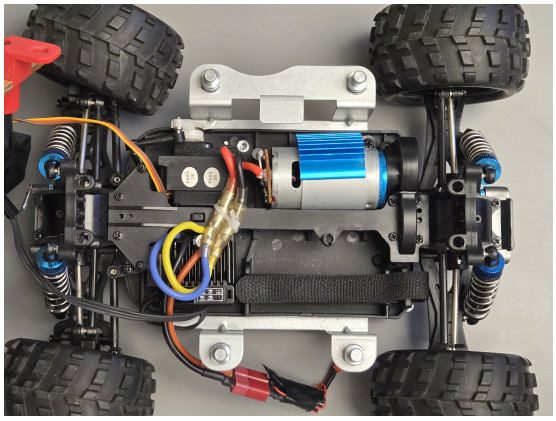

Fig. 6. Motor and Drive System in Chassis)

The DeepRacer system is powered by a single drive motor, which is connected to a central axle that distributes power to both the front and rear differentials. This configuration provides the system with all-wheel-drive (AWD) capabilities. Both differentials are mechanical open differentials, which direct power to the wheel with the least resistance, ensuring optimal traction in various conditions.

The drive motor is powered by a separate battery system, which includes a buck converter that regulates the voltage to the motor driver. The motor driver is directly controlled by the Raspberry Pi (RPi), which allows for precise motor control during operation. Additionally, the motor is equipped with an anodized aluminum heat sink to dissipate heat generated during use. This heat sink is actively cooled through the robot's movement, ensuring that the motor remains within a safe operating temperature range throughout its performance.

III. Implementation Details

A. Overall ROS2 Structure

During the beginning of the class, a variety of ROS 2 topics were introduced and shown. The structure of our team's code never progressed beyond a simple publisher/subscriber model. In which the publisher just published camera readings, specifically values related to colors we deemed were important for the challenge features, such as blue, red, and yellow/green. The subscriber took these readings and used information such as contour area or centroid position to control the vehicle's motor functions.

B. Camera Navigation

For the camera navigation around the track, the code developed for hardware check-off three was adapted to achieve line following. In check-off three, the challenge was to detect an orange cone and control the vehicle so it would turn towards the cone, approach it, and stop once within a certain range. The implementation for this worked well, and modifying it for line following involved adjusting the detected color range in OpenCV and refining the motor control logic. Unlike check-off three, where distance from the target was critical, line following simplified the logic to detect the presence of a specific color in the field of view (FOV). The software operates by selecting a predetermined HSV color range as the target. Once a matching color region is detected, it identifies the largest contour and estimates proximity based on its area. A smaller area suggests the object is farther away, while a larger area indicates it is closer. Steering is then controlled by calculating the centroid of the largest detected contour and determining its horizontal offset from the center of the frame. This offset is used as the error input for a PID controller, which adjusts the servo angle to keep the target centered in the field of view.

While check-off three required more distance-based considerations, line following needed to detect the color and then move the motors accordingly. If the line color was detected, the vehicle's motor was set to a predefined forward speed. If not, it either slowed down or stopped, depending on how long the line had been missing. This was handled using a timeout counter and fallback steering logic to help the robot recover the line. While a simple proportional controller can be effective for basic tasks, our implementation used a full PID

controller for more stable steering during line following. The PID algorithm would calculate the error between the detected line's centroid and the center of the camera frame, and adjust the steering angle to minimize that error over time. One of the primary challenges in this setup was fine-tuning the HSV color ranges in OpenCV to reliably detect the line under varying lighting conditions. Thankfully, the original black tape was swapped to blue, which significantly improved detection due to the improved contrast. Tuning the PID variables also proved challenging, especially when trying to reduce overcorrection or get consistent behavior during turns.

C. Stop Sign

Fig. 7. Stop Sign Challenge Feature Given PNG

In a similar fashion to the camera navigation above, the code written for hardware check-off three was leveraged to implement the stop sign feature. Using OpenCV, the robot monitored for red regions such as the given sign's color [Fig. 7] using a predefined HSV range. When the red contour area exceeded a set threshold, the driver node stopped the motors for four seconds before continuing. This threshold was determined through a fair amount of testing/tuning to ensure the car reliably stopped within an acceptable distance from the sign, while minimizing false negatives. Another approach to handle proper stopping would have been to use LiDAR readings and stopping the car once the stop sign came into distance, but as a result of the broken LiDAR, the team used a threshold-based approach.

D. Speed Limit Signs

Fig. 8. Speed Limit Sign Challenge Feature Given Red Zone PNG

Fig. 9. Speed Limit Sign Challenge Feature Given Green Zone PNG

In similar fashion to the stop sign implementation, OpenCV was used to detect either the red [Fig. 8] or green [Fig. 9] colors from the two school zone signs. When detecting the red school zone sign, the max speed was set to half its value in order to slow down the vehicle. Once the green school zone sign was detected, the max speed was doubled to return it back to its original value, hence speeding back up to normal speeds. With both of these in play, the challenge feature was completed. No timers or anything like that were needed for this challenge feature as opposed to the stop sign feature.

E. Vehicle Telemetry Visualizer

The team chose to implement the telemetry visualizer using *ROS 2 WebSockets* to enable real-time communication between the robot and the user interface. This architecture allows telemetry data such as speed, steering angle, and other sensor outputs to be transmitted to a browser-based dashboard. For the live video stream, the team utilized a ROS2 node capable of publishing image frames. These frames were accessed through a web frontend, providing a continuous visual feed from the onboard camera.

The telemetry interface was built using HTML, CSS, and JavaScript as a responsive web app. It subscribes to ROS 2 topics such as /robot/speed and /robot/steering_angle to display the vehicle's current throttle and steering commands in real time. This gives the user a live view of how the robot is responding to detected visual cues.

The dashboard combines live video and telemetry gauges into one interface, making it easy to monitor the robot from any device on the same network. This setup is especially useful for debugging, as it allows the team to see the colors and contours detected based on the current HSV values.

IV. Performance Analysis

A. Timed Trial

TABLE I
TIMED TRIAL LAP RESULTS

Trial #	Lap Time
1	1:58
2	2:15
3	2:12
Average	2:08

Team Apollo was able to achieve an average lap time of 2:08 across three separate splits [Tab. I]. Our fastest split was 1:58 and our slowest was 2:15. The standard deviation was 7.41 seconds. We believe we'd be able to achieve faster times by increasing the speeds along the straightaways, but with the persistent motor drift issue, a slower speed that allowed for consistent laps was preferable.

B. Stop Sign

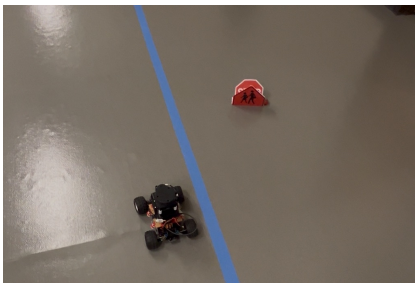

Fig. 10. Car Stopping at Stop Sign

When testing the stop sign feature, our vehicle was able to stop fairly consistently at a reasonable distance from the sign. However, performance varied depending on lighting conditions and the presence of other objects with similar colors. During five trial runs under optimal conditions, the car successfully stopped at a reasonable distance from the sign, waited for three seconds, and then resumed driving at its normal motor speed in 80% of our tests. The main challenge was using area thresholding to determine when to stop. Depending on the angle of the sign, lighting conditions, and how our algorithm processed the image, the detected area varied a lot. This made it difficult to define a threshold that would consistently stop the car at the desired distance we wanted. An improvement here could be detecting the shape of the sign in addition to its color, as well as further tuning the HSV ranges to reduce false positives.

C. Speed Limit Signs

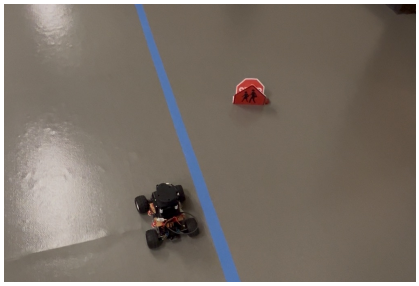

Fig. 11. Car Entering School Zone and Reducing Speed by Half

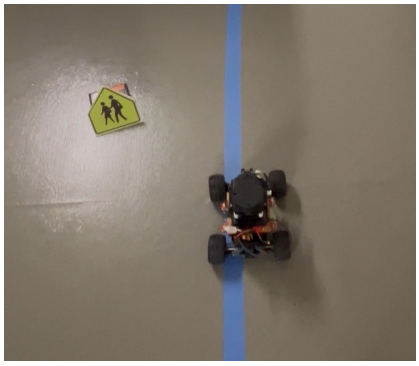

Fig. 12. Car Exiting School Zone and Returning to Normal Speed

Similar to the stop sign feature, the school zone compliance worked reliably in most cases, with a success rate of around 80% under optimal conditions. We faced similar thresholding challenges, where detection could trigger the logic to reduce speed by half because the area of the largest contour exceeded our threshold. In some cases, it also increased the speed too early, even before the car had fully passed the yellow sign. This again could be improved by refining the contour detection and HSV ranges, as well as incorporating shape-based detection to minimize false positives.

D. Vehicle Telemetry Visualizer

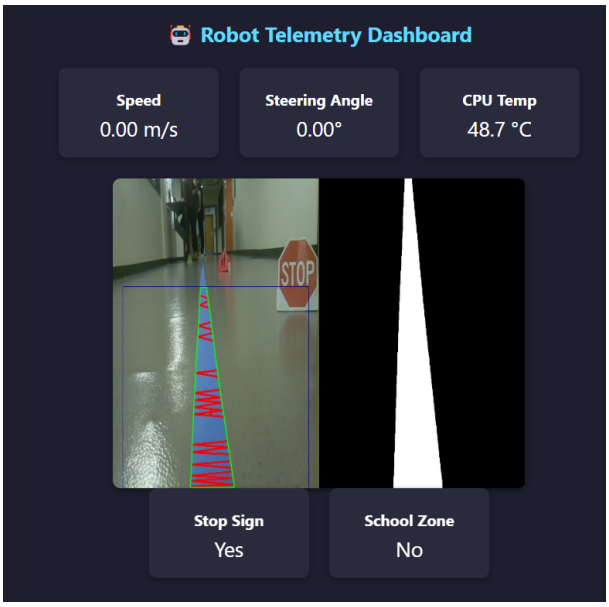

Fig. 13. Dashboard

The team implemented a suite of ROS subscribers to capture and process all of their robot's real-time data streams [Fig. 13]:

- **Motor & Steering Feedback:** Subscribed to `/robot/speed` and `/robot/steering_angle` to receive continuous updates on wheel velocity and servo position.
- **Vision Input:** Subscribed to `my_camera/image_raw` for the raw camera feed, then used OpenCV (via `CvBridge`) in callback functions to extract both contour outlines of the guiding line and higher-level lane detections.
- **System Monitoring:** Subscribed to `/pi/cpu_temperature` to monitor the Raspberry Pi's CPU temperature in real time, enabling proactive thermal management.
- **Traffic Sign Detection:** Subscribed to `/apollo_red_info` and `/apollo_yellow_info` to recognize stop signs and school-zone entry/exit events by evaluating contour area thresholds.

V. Challenges Faced

A. Hardware

1) Raspberry Pi 4 Fan: Throughout the semester, the team encountered two Raspberry Pi 4 fan failures. The first fan burnt out and became unusable. Unlike other hardware issues, the team believes this particular failure was not entirely within our control. The default placement of the hardware components, as specified in the setup instructions, did not provide adequate spacing for the fan to rotate freely. While the CAD design accounted for pin heights, it did not consider the wires emerging from the pins, which led to the plates being positioned too closely together. This resulted in the PCB making contact with the fan, obstructing its movement. As the Raspberry Pi 4 heated up, the software instructed the fan to increase its speed. However, due to the restricted space, the fan was unable to spin, and its motor continued to push, eventually causing it to burn out. To address this issue with the second fan, the team used a custom-printed plastic component placed atop the cooler, creating the necessary clearance for the fan to function properly.

2) LiDAR: At the beginning of the project, the team successfully operated the LiDAR sensor, receiving telemetry through the Slamtech RP LiDAR package. During the process of diagnosing hardware issues related to the fan, it was discovered that the micro USB connection on the LiDAR PCB had become deformed. This occurred when the top half of the robot was manipulated while the micro USB cable was still connected. The deformation caused pads on the PCB to lift, breaking the traces. Fine wires were used to restore the connections; however, the telemetry received after the repair was unusable. We suspect this issue may stem from delays in data transmission to the integrated chip over the copper wire.

3) DeepRacer Structural Base: During the testing of hardware check-off three, the team lost control of the robot which led to it crashing and breaking the structural base which held the camera and LiDAR sensors. As will be mentioned later, this loss of control occurred as a result of the team trying to solve the motor drift issue. When ranging values to determine the point at which the motors no longer ran, the values were between -0.3 and 0.3. For context, the full range of the motor values are from -1 to 1, so 0.3 was assumed to be fairly conservative. The team was quickly proven wrong, as 0.3 is actually quite fast. This speed combined with an already running directional control node resulted in a scene right out of a movie and a broken structural base. Thanks to a quick response from Professor Beuken, CAD files were obtained and the broken structure was able to be reprinted in a much more attractive black finish.

4) Motor Battery Lockout: Leading into the final stretch of the semester, the team came across an issue in regards to the motor battery locking out. This happens when the motor battery drops below a certain voltage threshold as it locks itself to prevent damage. Even if the battery is to be charged back up to full power, in order to be usable, it has to be unlocked using a special cable. Some mishaps led to this problem hindering the team's progress quite a bit, as we were unable to source another battery for testing. We were eventually able to obtain another battery and test our code for the challenge features.

B. Software

1) SD Card Ubuntu Corruption: At some point prior to hardware check-off number one, our installation of Ubuntu on the SD card got corrupted. The team never really figured out why that occurred exactly, but the solution was fairly straightforward and was just reformatting and re-flashing the SD card. Initially, the team was fairly hesitant to do this because it would result in a fair amount of progress being

lost, but it had to be done. Besides this instance, it has not happened again.

2) Shared ROS Topic names: During testing, we found that we would randomly get ghost images from different locations around the course. We rebooted our robot several times and worked to clear any caches that may contain these ghost images. It was only after seeing the same ghost image appear rapidly, almost every other frame, that we chose to investigate. What we found is that we were intermittently receiving camera data from another team's robot. The saving grace was that the ghosted images contained a wire dangling in front of the camera, this led us to deduce that the image could not be our own.

3) Motor Drift: One issue faced by the team during the course of the semester was an issue regarding the speed at which the motor ran at given a particular value. For example, setting the motor speed to zero should in theory stop the motor. This was not the case as the semester went on, it got to the point where -0.10 was the value the motor needed to be set at in order to stop the motor. Hence, the motor drift term. It is possible that the motor drift was actually a hardware issue, but the team is not sure we have the resources/understanding to determine that.

C. Team Issues

Throughout the semester, the team faced challenges related to individual effort and communication. Despite the graduate-level nature of the course, there were times when initiative was lacking, and absences from team meetings became somewhat frequent. The meeting times had been agreed upon by all team members as available, but absences were sometimes not communicated in advance. This led to an uneven distribution of workload, with some team members being less involved than others, which impacted the overall team dynamic and progress.

ACKNOWLEDGMENT

Team Apollo would like to acknowledge the efforts put forth by Leopold Beuken, Destin Woods, Brendan Crowe, and Praneeth Nemani. Thanks for making this project possible and thanks for the semester. Additionally, the team would like to thank all the folks who have contributed to OpenCV as we would not have been able to complete this project without their contributions. Lastly, Team Apollo would like to thank the authors of the references below.

REFERENCES

[1] Raspberry Pi Foundation, "Raspberry Pi line-following robot with Python," *Raspberry Pi Projects*, [Online]. Available: https://projects.raspberrypi.org/en/projects/rpi-python-line-following. [Accessed: Apr. 29, 2025].

[2] Adafruit Industries, "Adafruit Reflective IR Sensor - 5mm IR Transmitter and Receiver," *Adafruit*, [Online]. Available: https://www.adafruit.com/product/4292. [Accessed: Apr. 29, 2025].

[3] Dothecamera, "Raspberry Pi Camera Module 5mp OV5647 for RASPBERRY PI 3 / 2 / B+," *Dothecamera*, [Online]. Available: https://dothecamera.com/product/raspberry-pi-camera-module-5mp-ov5647-for-raspberry-pi-3-2-b/. [Accessed: Apr. 29, 2025].

[4] Slamtec, "Slamtec RPLIDAR A1M8 2D 360 Degree 12 Meters Scanning Radius LIDAR Sensor Scanner for Obstacle Avoidance and Navigation of Robots," *Amazon.com*, [Online]. Available: https://www.amazon.com/Slamtec-RPLIDAR-Scanning-Avoidance-Navigation/dp/B07TJW5SXF. [Accessed: Apr. 29, 2025].

APPENDIX

Challenge Features

Challenge	Points
Camera Navigation	5
Stop Signs	1
Speed Limit Signs	1
Telemetry Visualizer	1
Total Points	8

GitHub Repo Link

https://github.com/Apollo-Adv-Robo/Apollo

Participation

Student	Contribution
Michael Becerra	Hardware + Code + Report + Testing
Masoumeh Ghanbarpour	N/A
Thanushraam Suresh Kumar	Code + Report + Testing
Tsuening Lee	Hardware + Code + Report + Testing
Tyler L'hotta	Hardware + Report + Testing
Andrey Lototskiy	Code + Testing