2023 Annual Conference & Exposition
Baltimore Convention Center, MD | June 25 - 28, 2023
The Harbor of Engineering Education for 130 Years
ASEE
Paper ID #37668

Developing a Modular Package for Autonomous Driving and the Experiences Gained while Implementing this System in a Sophomore Level Design Course

Pradeep Radhakrishnan, Worcester Polytechnic Institute

Developing a Modular Package for Autonomous Driving and the Experiences Gained while Implementing this system in a Sophomore Level Design Course

Abstract

Radio-Controlled (RC) scaled cars are a great way for prospective engineers to learn real-world technical skills including the basics of autonomous driving. Current self-driving RC car modules require hours of setup configurations and track-specific training before driving autonomously. We developed the Modular Package for Autonomous Driving (MPAD) to allow Mechanical Engineering students to assess their work in a complete system without requiring coding or electrical design background. MPAD consists of two development boards, a microprocessor (Raspberry Pi 4 8GB) and a microcontroller (Elegoo Mega 2560). Various sensors (Ultrasonic, inertial measurement unit, temperature, and hall-effect) are connected to the microcontroller, which then interfaces with the microprocessor. The sensor package uses a camera for lane recognition and ultrasonic sensors for collision avoidance. The package was specifically designed to be easily transferable from one RC car to another. MPAD was implemented in an introductory engineering design course at XYZ University, where groups of students designed a custom scaled car with gearbox, steering linkage, and a chassis that can accommodate MPAD. The aim was to check the modularity of MPAD and to verify if the provided documentation can lead to the creation of a successful self-driving RC car. Eight teams of mechanical engineering students each received documentation in the form of two guides to create their own version of MPAD-enabled RC car. While there were challenges and learnings, the integration was successful, and the students were able to fully utilize MPADs capabilities with their own RC car design and demonstrate a self-driving scale car. This article will discuss the implementation, testing details, experiences gained and future work.

1. Introduction

Autonomous vehicles have gained considerable interest in recent years due to their potential for disruption. As they advance in capability and increase in adoption, studies have shown autonomous vehicles can reduce accidents and traffic congestion [1]. The race to achieve full autonomy is undoubtedly here and many companies are taking part in it. Although the closest purchasable fully autonomous vehicles are those with adaptive cruise control (ACC), there are now driverless, fully autonomous commercial vehicles on the roads in the United States. Waymo [2], a subsidiary of Google entirely focused on self-driving, recently released a driverless ride-hailing service in Phoenix, named Waymo One [3]. This service is fully available to the public but is not yet ready for denser city driving with a larger population and more obstacles. As self-driving technology cements itself in our society, it becomes more important to provide exposure to the technologies involved in autonomous driving to students in mechanical engineering. Developing a car with these capabilities requires computer vision and a variety of sensors to recognize its surroundings and navigate obstacles.

Due to their size and scope, Radio Controlled (RC) cars are a great way for prospective engineers to learn real-world technical skills. The low cost and shorter turnaround time allow for rapid development and testing, and can effectively teach many of the same principles as real cars. At XYZ University, ABC is an introductory design course where remote-controlled scale cars are designed, analyzed, built, and tested by groups of students. Each group was allocated a budget with which they can acquire all the necessary parts. The project was divided into three segments. First, the students construct custom powertrains that can be housed in a gearbox. Then, a steering component made up of a servo and a parallelogram linkage mechanism is designed and assembled. Finally, students manufacture a custom chassis that can accommodate and support the weight of all the electronics (including sensors), the gearbox, and the steering mechanism. Towards the end of the course, the RC Cars designed and built by these groups are evaluated and tested on indoor tracks. During this process, groups are tasked with implementing MPAD on their designed RC cars. They are provided with a sensor and AI guide that is easy to follow, one that requires no prior knowledge of sensors or AI programming. Emphasis on ease of use ensures that students are spending time developing their car rather than worrying about configuring sensors or software. This was a guiding principle throughout the project and pivotal in making sure the module could be used by students with any skill level. By introducing MPAD to classrooms, students can discover and integrate electrical engineering and computer science principles with mechanical engineering to deliver a comprehensive final product. Autonomous driving is the ideal way to test a custom RC car's mechanical integrity and avoid human-induced driving bias.

The paper is organized as follows. Section 2 presents background on currently available RC car modules followed by a detailed description of MPAD in Section 3. Section 4 will discuss the implementation in course, outcomes and feedback from students in addition to various challenges that were encountered. Finally, concluding remarks will be presented in Section 5.

2. Related Work

Plans to introduce an add-on self-driving module to the student-designed RC cars were met with difficulties as current self-driving RC car modules, such as Donkey Car [4], SunFounder PiCar [5], and Deep Pi Car [6], require hours of setup configurations and course-specific training data set before driving autonomously. Even after training, most modules did not provide the capability to avoid obstacles or traverse non-standard terrain. Furthermore, additional steps would need to be taken by the group to configure the self-driving module to the specific design of the RC Car. A particularly useful self-driving module that can be used by students is the AWS DeepRacer [7]. This package focuses more on the deployment of a custom machine learning model to a prebuilt RC car that contains the required sensors. Therefore, AWS DeepRacer advertises itself as a tool to allow prospective users to practice and deploy machine learning and renders insufficient for a course such as ABC designed for mechanical engineering students.

Donkey Car uses Machine Learning to drive. Given that it does not call upon a previously established model, the user would need to create their own model. Creating your own model is a lengthy and complicated process that requires starting over if the model is flawed. Collecting the training data requires manually driving the car around. The driving skill of the model will be

dependent on the driving skill of the user. The driving data then has to be transferred to the user's personal computer where the training can take hours. The end product is track-specific and such an RC Car cannot stay in the lane if changes are made to the track. The driving itself of both DonkeyCar and AWS DeepRacer influenced the training process and was not universal to any RC car. Interfacing with the car was not essentially intuitive. Therefore, it was quite unreliable to use DonkeyCar or AWS DeepRacer in a mechanical engineering course such as ABC at XYZ University.

3. Modular Package for Autonomous Driving

The intent of the Modular Package for Autonomous Driving (MPAD) is to allow users to convert their Radio-Controlled (RC) cars into fully autonomous ones. Such mechanisms will need to have similar technology like autonomous vehicles. Most autonomous vehicles are equipped with sensors that enable them to detect objects and avoid obstacles. This is carried out by computer software that takes physical readings of the car's surroundings and makes sense of not only where the car currently is, but where it should be moving towards. Sensors can also call attention to data about the car's performance and can be useful in designing safer cars. The current sensors implemented through MPAD are cameras, ultrasonic sensors, IMUs, temperature sensors, and hall effect sensors. Readings from these sensors are displayed on a dashboard and can be exported for further analysis. Users can engage with MPAD through this dashboard.

3.1 Dashboard

The dashboard was intended to provide the user with real-time driving controls and a display for the sensor data. After assembling the sensor package, students can simply navigate to the MPAD Web Application's unique URL on any web browser to view documentation and access their car's driving dashboard. When the RC Car's Raspberry Pi is powered on, it will automatically register on a server and begin streaming camera and sensor data. Users can monitor and export sensor data, watch the camera feed, and control their car's driving in real-time through the dashboard (seen in FIG 1).

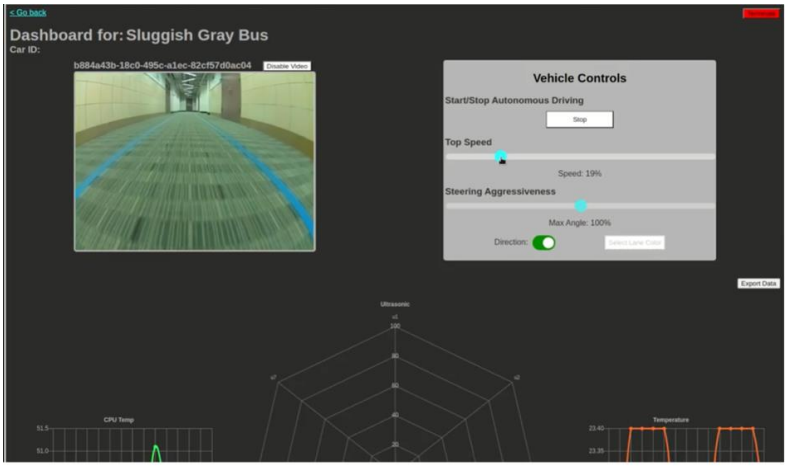

FIG 1: Driving dashboard on MPAD's web application

Within the car, there are numerous sensors that are wired through a breadboard to the microcontroller. Starting with the sensor module (in blue on FIG 2), the sensor data and camera

feed are processed on the Arduino and forwarded to the Raspberry Pi, which then sends that information to the users (in pink on FIG 2) on the web dashboard through the Heroku [8] server. The Raspberry Pi can then receive driving requests from the users and send them to the mechanical driving components (in green on FIG 2). Essentially, the server acts as the gateway between the car and the web interface page.

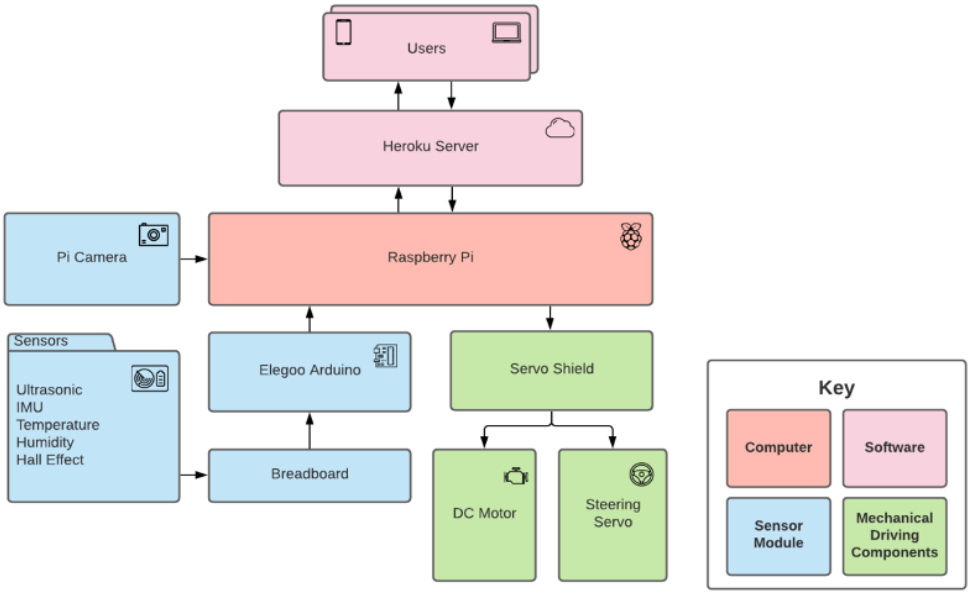

FIG 2: Abstracted system block diagram

All communication and data go through Heroku as seen in FIG 3. The start/stop commands from the webserver trigger the car's self-driving program. When a user makes a change to any of the driving controls, the dashboard will send an appropriate application programming interface (API) request to the server with the car's id. The server will then retrieve the current configurations of the car, apply the requested change, and send the updated information to the car. From here, it continually polls the server every 500ms to check if any control updates are necessary. The server stores all its information in Redis [9], a storage system that works well with Heroku, and updates it appropriately as changes are made.

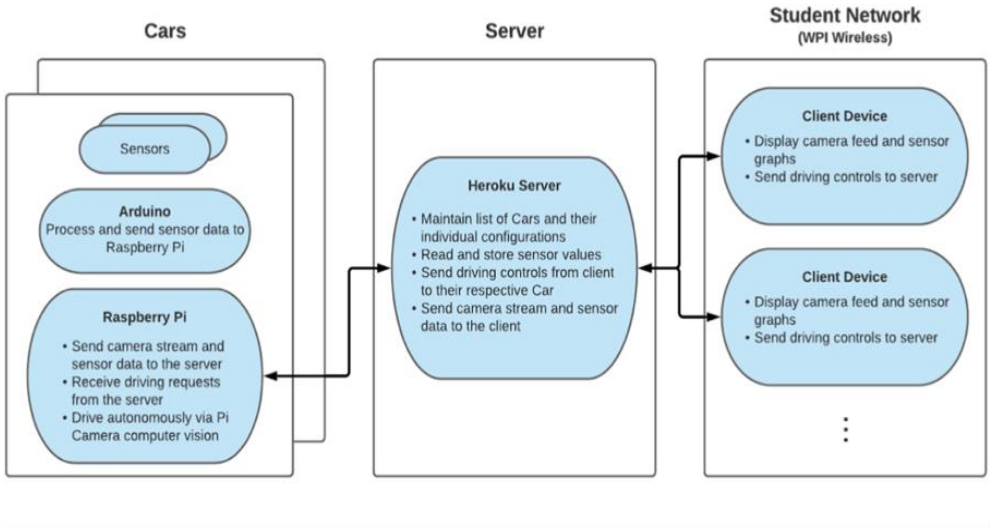

FIG 3: Web application system architecture

The dashboard pulls in data in real-time from the car's sensor array. This is done while providing a means for users to control the car's self-driving algorithm. The system utilizes a wide array of tools including Flask [10], Heroku, bash scripting, Chart.js [11], Socket.io [12], and general web development tools (HTML, CSS, JS). These tools were used in tandem to provide the user an intuitive connection with the car.

MPAD will allow a vehicle to successfully drive tracks with lanes made up of any color. In order to achieve this, a 'Lane Selection Tool' was designed to allow users to dynamically select the color of the lane and be able to view how the car sees it. To use this tool, a user must simply click on different parts of the lane in the frame on the left (seen in FIG 4). Upon each click, the right frame will begin to highlight the lane in white (seen in FIG 4).

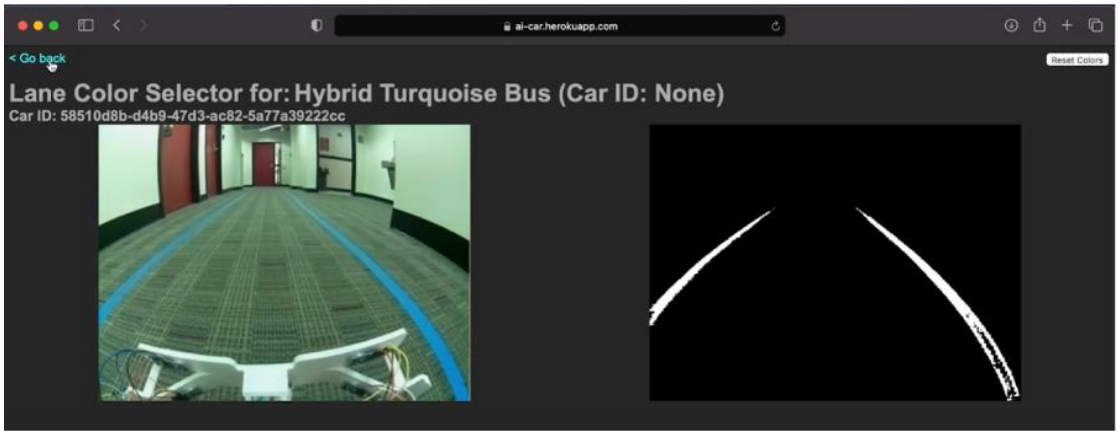

FIG 4: Lane color selection tool on the web dashboard

There is a button to reset the color channels if the user selects a pixel that they did not intend to. Any selection that is not on the lane will create noise and highlight background elements that will

disrupt the driving algorithm since the color channels are output to the car and used in the driving code. Once the lane colors are set, the car is ready to be driven autonomously.

3.2 Autonomous Driving Implementation

Control of the self-driving algorithm is accessible through the dashboard. Autonomous Driving is achieved by video capture, lane following, and obstacle avoidance. The camera is the main input required for detecting lanes. This makes it the single most important sensor for autonomous driving. Because a camera has a wide field of view and captures content in color, it is ideal for aiding a self-driving algorithm in a cost-effective way. Mounted at a 60-degree angle relative to the face of the car, the camera captures most of the car's forward perspective at 8in above the ground.

The primary goal of lane following is to stay in the middle of the two lanes. To do this, the algorithm looks at the bottom half of the image to determine which side of the screen has the least lane color in it. Because this side is assumed to be furthest to the car, the servo is set to steer toward it. FIG 5(left) shows a section of the frame used by the algorithm to determine the distance from the lane on either side. Getting to the middle of the lane is done by attempting to equalize the car's distance to both lanes.

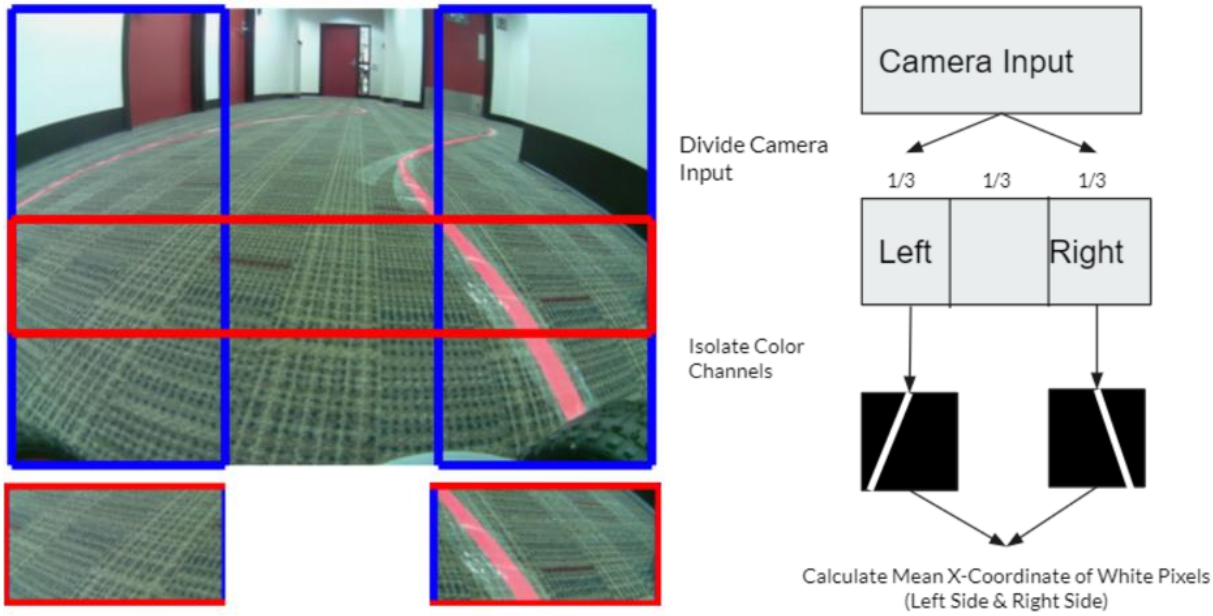

FIG 5: (left) Image Selection on Frame to determine the amount of Lane Color, (right): Lane Detection Flow Diagram

Given, for example, the two final cropped images in FIG 5 (left), the algorithm will suggest a sharp left as the cropped area on the left does not contain a lane to detect. A schematic of this Lane Detection process is shown in FIG 5(right). Principally, the difference (distance) of the left lane is

subtracted from the difference (distance) of the right lane. This is called the offset and it determines where the car should be going to stay in the middle of the lane. Based on the value of the offset, the algorithm determines not only which side to turn but also the sharpness with which it has to make the turn.

Obstacle avoidance builds on top of lane following. This means that obstacle avoidance will still follow lanes. Obstacle avoidance works with a fusion of ultrasonic sensors and the camera. Autonomous driving is only achieved by looping the above process of video capture, lane following, and obstacle avoidance process hundreds of times a minute. This approach is the most minimally taxing on the Raspberry Pi's CPU.

3.3 Sensor Implementation

Choosing the appropriate sensors to record the data patterns is an essential part of any car's design. A combination of sensors allows the car to recognize and interpret what is occurring around it.

Sensors are placed in specific areas of the car. The sensor package contains two temperature sensors (one near the battery and another close to the motor) to avoid overheating issues, one IMU to allow for drivability through ramps, one hall effect sensor to collect data on the rpm of the motor, and seven ultrasonic sensors (one in each side and five in the front) to enable the obstacle avoidance feature of the car. The locations of these sensors are depicted in FIG 6.

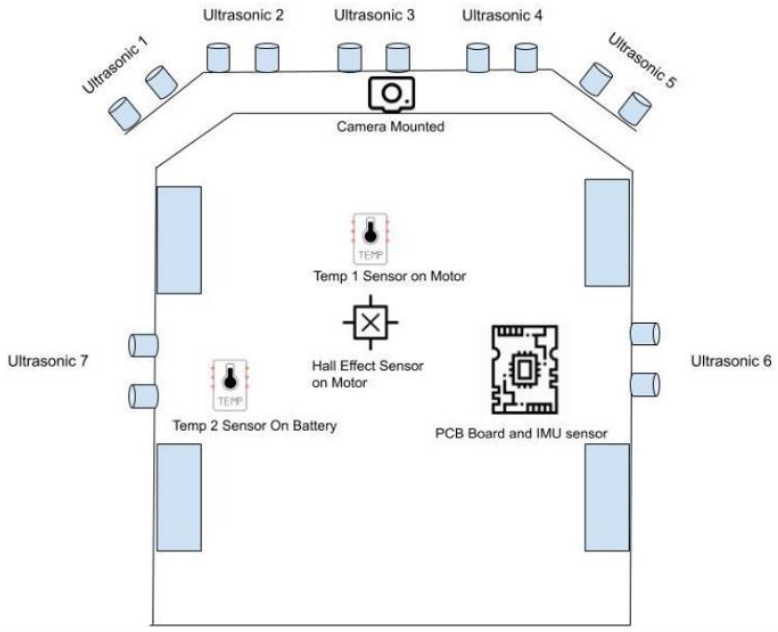

FIG 7: Sensor Layout in an RC Car

The final sensors' block diagram can be seen below in FIG 7. Starting from the Elegoo Mega (Arduino, microcontroller) and the sensors, the sensors mentioned above in the Sensor Layout section of this chapter are wired to an on-car breadboard (or printed circuit board). From there the sensor data is compiled in the Elegoo's software platform and sent serially to the Raspberry Pi.

The Raspberry Pi takes three other inputs: the camera for lane following, the portable battery as a power supply, and the server for user-input driving conditions. From these inputs, the Raspberry Pi controls the Servo Shield. The Servo Shield then sends those controls to two outputs: the Steering Servo for steering and the ESC for the velocity of the car. The ESC also requires a LiPo Battery to have a battery sensor for the car. This makes the Raspberry Pi the main processor responsible for providing the commands to the mechanical parts of the car. The temperature data, acceleration, and orientation data gathered from the sensors by the Elegoo Mega will be sent to the Raspberry Pi, which in turn will forward it to an online UI dashboard alongside the camera's live feed. All these components combined create MPAD.

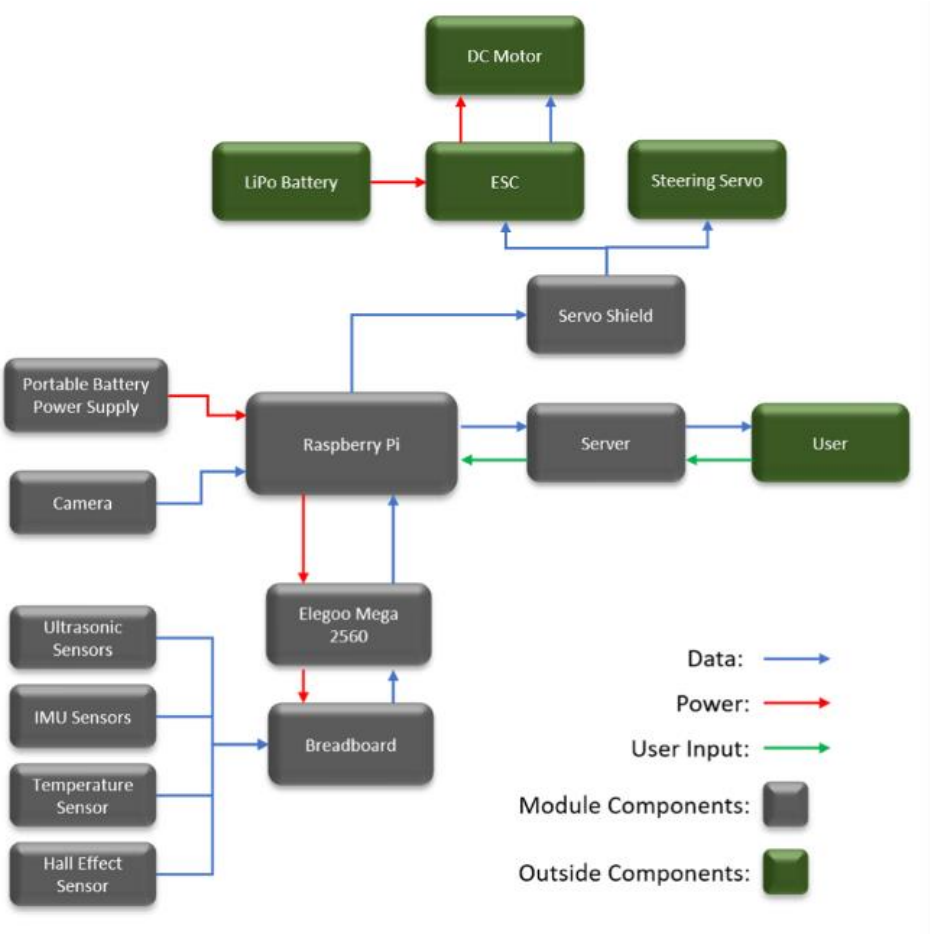

FIG 7: System Block Diagram for MPAD

The sensor package was an essential part of the fully integrated MPAD providing sensor data for display in the dashboard and allowing for features enabled by the AI aspect of the project. All of this information can also be found in the main guides for sensor setup, which are the Sensor and AI Setup Guide and the Arduino IDE Setup Guide.

3.4 Technical Testing

Complete testing was carried out on a test car before MPAD was introduced to ABC Course in XYZ University. This included the sensor accuracy testing of each sensor in the system. Each of

the sensors is presented on a standard line graph, with the exception of the ultrasonics which is shown on a web graph. Please see FIG 8 for a breakdown of the data graph from each sensor.

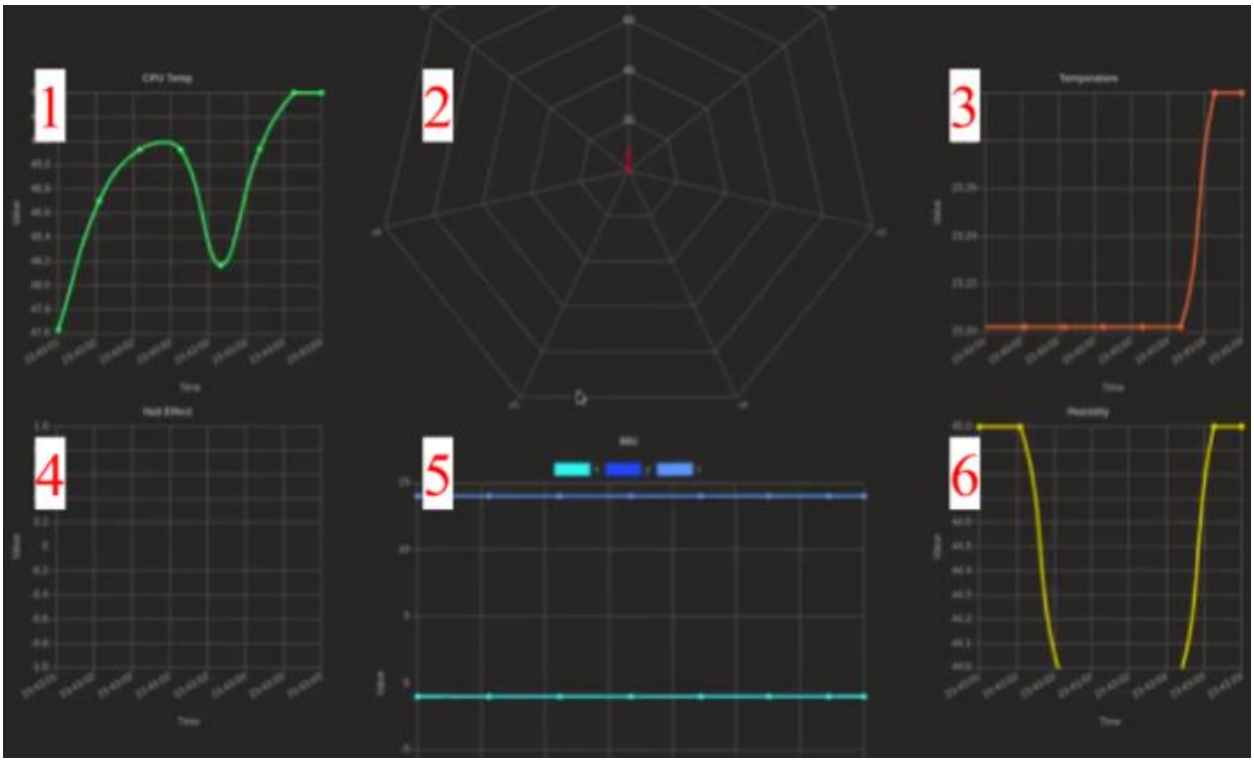

FIG 8: Sensor Readings being Displayed on Dashboard

The Web Application was found to be accessible through any web browser and even worked on mobile devices. The startup script which includes the sensor reading script and the master script that handles the car's controls is more reliable on the Raspberry Pi 4 (4-8GB RAM) than Raspberry Pi 3 (1GB RAM). The upgrade to Raspberry Pi 4 decreased latency and allowed the cars to be more responsive at higher speeds. With around 35 ms runtime, the driving essentially ran at 28.5 fps. This means the driving code itself was fairly efficient.

The RC car used for testing is 27cm wide, 41cm long, and 18 cm high. It is a 3-D printed RC car made from mostly plastic parts with some aluminum elements (FIG 9). The RC car is rear-wheel drive only and is capable of reversing direction. In terms of steering, each wheel can turn 46 degrees in both directions. This gives the RC test car a turning radius of roughly 90 degrees. Given the dimensions and size, the car was tested on multiple tracks with the full modular package. The car was also tested for its speed on the track and its ability to maneuver around obstacles and stay between lanes.

FIG 9: RC Car used for Testing

It was found that when the speed of the car increases, the latency stays the same but the opportunity to make up for it decreases. Sharp turns also have less room for error so during high-speed testing they tend to be the point of failure. Obstacle avoidance was tested by having the car complete the track continuously while increasing the size of the obstacle blocking the lane. Testing showed that as long as the obstacle had a surface from which ultrasonic sensors could bounce rays off, the size of the obstacle was not as important as the total road blocked. The car was able to avoid obstacles and still stay within the lane as long as at least 60% of the road was unblocked. Sharper turns were needed on the bigger blocks so not every car might have the steering capabilities to avoid it.

As a total reset, if something were to go wrong with the algorithm, the user can start and stop the driving to 'restart' the process.

4. Implementation in course

MPAD was introduced in a 2021 spring semester ABC Course at the XYZ University. The goal of the course was to help mechanical engineering students understand mechatronic systems. In the course, there were eight groups, each consisting of 4-5 students. The groups designed, analyzed, manufactured, and assembled RC Cars that could not only carry a payload but also take full advantage of an add-on self-driving module to drive autonomously. In the process, they became familiar with a structured product design process that included challenges in design thinking, analytical and computational analysis of the design, points of failure analysis, and integration of electronic components to a mechanical product.

After students designed and built their RC Cars, they were given "How to Setup" guides. These guides were written with the intention of providing them to Mechanical Engineering students. As mentioned earlier, there were two main guides: the Arduino IDE Setup Guide and the Sensors and AI Setup Guide. The Arduino IDE Setup Guide provides step-by-step instructions on uploading the Arduino IDE and sensor code to the Elegoo Mega 2560 microcontroller. It contains images to support and assist students as they go through the whole process by themselves. The Sensors and AI Setup guide aids the students with setting up the hardware required to fully utilize MPAD. The guide goes over the main components of the sensor package as well as the implementation and

best places to position the various sensors. It also covers the AI and UI systems overview and the limitations of MPAD.

Since the course was composed of mostly freshmen and sophomore year students, a significant challenge faced by the curators of the setup guides was to convey the complexities of sensor and software integration to an audience that had limited knowledge about the subjects. Nevertheless, the setup guides were successful in assisting the students with the process of converting their RC Cars into autonomous ones. This is to say that students were first able to manually drive their scale cars around before attaching the self-driving component to it and then having it drive autonomously.

After the manual driving phase, each team decided on locations and methods to mount their Elegoo 2650 Board, Raspberry Pi, and Breadboard on the RC car. This helped them organize wiring across the RC car before attaching all the sensors. MPAD is designed to take advantage of the Electronic Speed Control (ESC), which regulates power to the brushed DC motor in the gearbox, and the Servo Shield, which is the principal steering component of the RC car. Wires from these two controllers are first attached to the Raspberry Pi. Then all teams 3-D printed their bumper and camera mount. The bumper held the ultrasonic sensors and the camera mount in place and was crucial in the self-driving aspect of the project. The ultrasonic sensors were wired to the breadboard and the camera was directly connected to the Raspberry Pi as seen above in FIG 8. It was worth noting that five of the seven ultrasonic sensors were placed on the bumper while the other two were supported and connected on edge of the chassis between the rear and front wheels. Placement for and wiring from the hall effect sensor and temperature sensors were also carried out in a similar fashion. The IMU was soldered to the breadboard itself. Once this was done, the Elegoo 2650 Board itself was wired up onto the Raspberry Pi. Housing for batteries and payload which were previously manufactured during chassis design could also be used now to power the Raspberry Pi, Servo, and the Motor. A fully connected and set-up self-driving RC Car designed by a student team can be seen in FIG 10(a) and FIG 10(b) below.

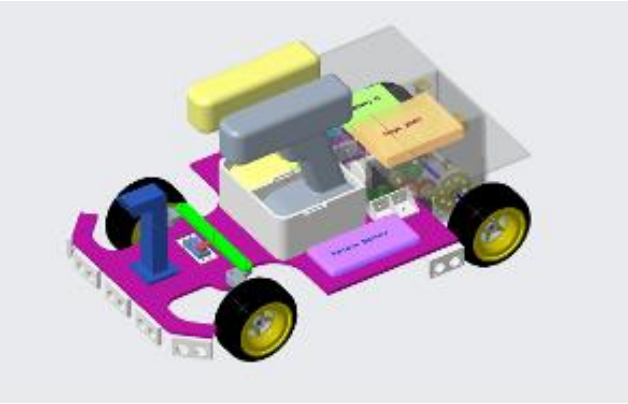

FIG 10(a) – Example CAD Model of Final Car

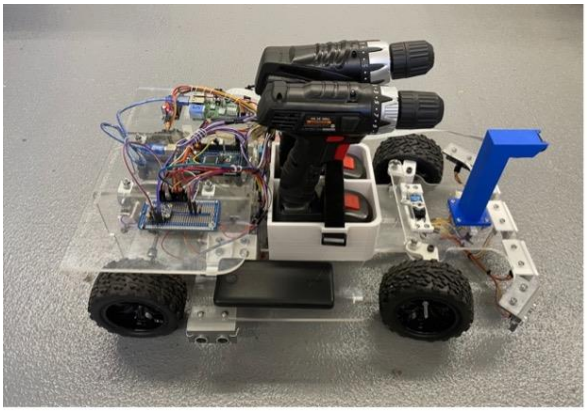

FIG 10(b) – Example Final Autonomous RC Car

Most groups were able to accomplish these tasks with help just from the guides. From here on out the students can connect to and view their RC car through the online dashboard. In order to test the cars' ability to drive autonomously, it was important to test whether the camera was turning on

and connecting to Bluetooth correctly. The guides also served as a resource to view solutions to common problems during troubleshooting. Despite sometimes breaking apart and losing functionality during testing, all teams were able to make modifications and improvisations to their self-driving RC Cars before submitting the cars on race day. This is also to say that the teams had successfully tested, raced, and submitted working models of their self-driving RC cars. However, to submit a final product, most teams initially required assistance with connection issues and irregularities with their self-driving RC car. The issues broadly fell under these categories: connectivity, unreliable parts, component orientations, manufacturing defects, and server overload issues.

The most common issue was connecting the Raspberry Pi to a Wi-Fi network or hotspot. This step is essential as it will allow the groups to Remote Desktop into the Raspberry Pi and upload the sensor code. However, the campus Wi-Fi at XYZ University has multiple firewalls and procedural steps that hindered ease of use and connection to the online dashboard. To avoid this lengthy and troublesome procedure, the MPAD team pre-configured all of the Raspberry Pi's to the XYZUNIVERSITY-WIRELESS network and developed a startup script that automatically enrolls onto the Web Application's server. Through initial testing, it also became clear that students could quickly pick up the dashboard and learn the functionality of the controls. The issues mostly came up with loading the dashboard across various devices for the first time. For instance, initial testing in ABC Course revealed that "https://" at the beginning of the URL is necessary for the dashboard to work. If not, the camera's live feed will not be displayed, since the WebSocket requires an encrypted connection to the server.

Apart from issues with defective components, some groups, initially, were provided with Raspberry Pi 3's instead of Raspberry Pi 4's which resulted in slow response and irregular driving by the RC cars. Through testing, it was seen that there were significant differences in driving performance on Raspberry Pi 3's because they had only 1GB RAM. As latency increases, the car can oscillate more. At high speeds, this can cause the car to drive off the track. This is why it is recommended that a Raspberry Pi 4 with at least 4GB of RAM be used. In addition to running the driving code, the Raspberry Pi is responsible for sending the sensor data and camera feed to the server and receiving driving requests from a user.

While testing their autonomous RC, one team was faced with an unusual issue with driving. The code for driving was designed such that servo is mounted upside down. This meant the servo placed in a different orientation could steer toward a different direction than the intended. It also resulted in some overheating issues. To rectify this, an edit to the 'angleset' needed to be made in the drive.py file. Once this change in the Raspberry Pi code was registered the RC car began to steer correctly again. Another team realized a similar issue with their servo. This team's servo was centered at a 15-degree angle to the left causing the car to shift left and swerve intentionally from a straight path. This was a relatively easy fix and required a simple adjustment of the servo.

Manufacturing issues also crept in during the testing phase. One team realized that their laser-cut chassis did not account for the margin of error resulting from the manufacturing process itself. This issue prevented certain components from hooking onto the chassis such as the side-view

ultrasonics. This issue was resolved by remaking this section of the chassis and incorporating the margin of error. Material properties greatly affect the continued use of the self-driving RC Car. While groups tested their autonomous cars, some crashed into walls, resulting in broken bumpers or misplaced gears and shafts. For example, one team had their ESC connected in the opposite direction making the car reverse backward instead of moving forward and thus crashing the car. This was a frequent issue and groups recognized that they needed to test their vehicles in an environment where crashes could be easily stopped.

Throughout the testing phase, it was seen that various issues with components and miswiring contributed to difficulties with self-driving. Much of this resulted in a greater need to highlight the exact configuration required for MPAD to run effectively. Feedback from the students in the course provided valuable insight as to where the guides overestimated student abilities or under described certain procedures. These updates to the guide were added after MPAD was tested in a classroom setting. Some of these issues include the distance of the camera above ground level for maximum view, the requirement of certain sensors like the temperature sensor to be placed within the vicinity of another component, checking the working of individual components before the whole system was put together, etc.

During the course, many groups carried out late-night testing to get their RC cars working before the deadline. Affairs resulting from a collective need by all the groups to carry out these assessments at the same time resulted in overloading of the server before race day. The application can handle any number of cars but is limited by the server's request handling capabilities and as a result, it is recommended for groups of two to four to test the service at a given time.

It is safe to say that most students left the course with valuable experiences. Many of them have formed good friendships as well. Most of all, students were able to comprehend complex information and use it to design, build and create complex electromechanical systems in a team setting. During course evaluations, it was found that the course was rather difficult as it was time-consuming but otherwise it felt "rewarding" on completion with students rating over 4.2/5 for the course content and the learning achieved. The link presents videos of the students testing their RC cars using MPAD: https://1drv.ms/u/s!AtHIJxxntmsWyXj28wetKGoTgwdh?e=9bcUID.

5. Conclusion and Future Work

MPAD was created to be a modular package that allows RC cars to autonomously drive through tracks. The final package contained features such as lane following, obstacle avoidance, sensor data gathering and display, and interactive dashboard. The package does not require any training and is not track specific, increasing the efficiency, practicality, and value of the final product. This project was able to complete its main educational goal with the help of two guides that allowed the students in ABC Course at XYZ University to be exposed to the concepts of electrical engineering and computer science by integrating MPAD into their own custom RC cars. This MPAD implementation was also proof of its modularity aspect, where different RC cars were all able to utilize MPAD to add autonomous driving features to their design. This process exposed

project participants to various challenges in developing a system and scaling that to be used in a course. In terms of future work, there are several additions planned such as utilizing additional sensors such as LiDARs for obstacle detection and switching to local hotspot and a local server-based system for enhanced connectivity and restricted-free access.

6. Acknowledgments

The authors would like to thank the students in the ABC course for utilizing MPAD and providing valuable suggestions. The authors would also like to thank the Mechanical Engineering Department for financing the purchases of important equipment.

7. References

[1] P. Coppola and D. Esztergár-Kiss, *Autonomous Vehicles and Future Mobility*. Elsevier, 2019.

[2] "Home," *Waymo*. https://waymo.com/ (accessed Feb. 28, 2023).

[3] "Waymo One – Waymo." https://waymo.com/waymo-one/ (accessed Feb. 28, 2023).

[4] "Donkey® Car - Home." https://www.donkeycar.com/ (accessed Feb. 28, 2023).

[5] "Raspberry Pi Video Car Kit (Picar-V) for Intermediate," *SunFounder*. https://www.sunfounder.com/products/smart-video-car (accessed Feb. 28, 2023).

[6] M. G. Bechtel, E. Mcellhiney, M. Kim, and H. Yun, "DeepPicar: A Low-Cost Deep Neural Network-Based Autonomous Car," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Aug. 2018, pp. 11–21. doi: 10.1109/RTCSA.2018.00011.

[7] "AWS DeepRacer - the fastest way to get rolling with machine learning," *Amazon Web Services, Inc.* https://aws.amazon.com/deepracer/ (accessed Feb. 28, 2023).

[8] "Cloud Application Platform | Heroku." https://www.heroku.com/ (accessed Feb. 28, 2023).

[9] "Redis," *Redis*. https://redis.io/ (accessed Feb. 28, 2023).

[10] "Welcome to Flask — Flask Documentation (2.1.x)." https://flask.palletsprojects.com/en/2.1.x/ (accessed Apr. 27, 2022).

[11] "Chart.js." https://www.chartjs.org/ (accessed Feb. 28, 2023).

[12] "Socket.IO." https://socket.io/ (accessed Feb. 28, 2023).