Design and Implementation of PID-Based Steering Control for 1/10-Scale Autonomous Vehicle

Victor Robila
Hunter College High School
New York City, USA
victor.robila@gmail.com

Laura Paulino
Department of Computer Science
Montclair State University
Montclair, USA
adamespaull1@montclair.edu

Mihir Rao
Chatham High School
Chatham, USA
mihirraov@gmail.com

Iris Li
Millburn High School
Millburn, USA
iris.li6285@gmail.com

Michelle Zhu
Department of Computer Science
Montclair State University
Montclair, USA
zhumi@montclair.edu

Weitian Wang
Department of Computer Science
Montclair State University
Montclair, USA
wangw@montclair.edu

***Abstract*—Due to the rising popularity and interest in autonomous vehicles, it is beneficial to build a low-cost prototype autonomous vehicle for educational purposes. Thus, more students would be able to acquire hands-on experiences in designing and implementing critical smart navigational functionalities for autonomous vehicles. In this study, Proportional-Integral-Derivative (PID) control and steering control algorithms are developed to maneuver a 1/10-scale autonomous vehicle in a real-world scaled-down driving environment. An obstacle detection system is also designed. The state-of-the-art Robot Operating System (ROS) is employed in the software development and vehicle control to communicate between components. This paper, as part of a few related papers on autonomous vehicle from our research group, focuses on the control algorithms design and implementation, which incorporates continuous real-time feedback to generate a correction value to keep the vehicle on track. The algorithm combines the lane tracking, stop sign detection, and obstacle detection of the vehicle and sends data values to motors. Experimental results suggest the efficacy of our developed approaches. The future work of this study is discussed.**

***Index Terms*—autonomous vehicle, computer vision, obstacle detection, PID Control**

I. Introduction

Autonomous vehicles are at the forefront of the next transportation model. As technology progresses, there are opportunities to reduce pollution and congestion in traffic [1]. Widespread use of autonomous vehicles communicating with each other can lead to less car crashes as wells as an overall decrease in pollution due to the reduction of traffic congestion [2]. These factors served as motivation for our research.

For successful navigation, autonomous vehicles need to be able to respond to driving environments using lane tracking, obstacle detection, and sign recognition. Software components used to build the obstacle detection and lane tracking systems are thus crucial for the function of an autonomous vehicle and have been extensively investigated. However, the independent software components also need efficient coordination through central modules [3]. Therefore, the development of a PID (Proportional-Integral-Derivative) program to manage the control loop where data are converted from sensory inputs (cameras and sensors) into numerical data that are then used to adjust steering and speed, is vital to the success of a vehicle.

Further motivation for this study involves the low cost of the vehicle. Creating small-scale autonomous vehicles can broaden the participation of students in research and learning experiences. Such platforms can further serve as devices to engage the broader society to explain autonomous navigation and thus increase the acceptance of the technology. The scaled-down vehicles are also vastly cheaper than a full-size autonomous vehicle, yet they are still able to provide a multitude of research opportunities [4], [5].

This project is a continuation of previous work that focused on the design of an autonomous vehicle and the overall identification of the software and hardware models [6], [7]. A plan of work on the obstacle detection as well as a framework for steering control was conducted and described. In addition, the lane tracking and stop sign detection as described in [6] were done. This study focuses on the PID-based steering control of the vehicle. The track, where the autonomous vehicle ran in simulated situations in the real world, could be scaled up for full-size vehicles. Autonomous driving can be achieved by using the vehicle's cameras and sensors to make decisions based on data produced by the environment.

II. Literature Review

There are several approaches for the creation of autonomous vehicles and different ways to run them. One of the main methods to do this with smaller scale vehicles is to run the autonomous vehicle on an Arduino device that controls the physical actions (turning/speed, etc.) of the vehicle [8]–[10]. Another way to utilize the Arduino device is through a multi-objective pipeline approach that takes inputs and directly sends them to separate motors based on the type of input given. This can help increase the efficiency of a vehicle by eliminating the need to sort through data and reducing the overhead of signal

transmission [9]. These methods are usually simpler but remove the possibility of choosing between multiple inputs and making decisions based on a wider variety of data [9]. For example, this strategy works well for a vehicle that has an ultrasonic sensor that stops the car at a certain distance but is harder to implement in a vehicle with multiple types of data such as angles and conflicting or overlapping data (e.g., having an obstacle detection program which stops the vehicle and some other software that also stops the vehicle but does so due to different data). Another method for the steering and car control PID is an individual pipeline approach that takes different data inputs and converts them into signals that communicate with the motors on the vehicle, either wireless or wired. This method ensures that all the data is received at the same rate and that the machine can react to the data in a timely manner [5]. The data inputs received vary among projects and can range from obstacle detection sensors to cameras to assist with lane tracking; a decision is made based on this data as to the exact direction and speed that the motors should move [5], [11], [12].

In addition to the structures of the other papers' autonomous vehicle software, there are also various software/hardware differences. Previous studies used an Arduino device to run code due to how it incorporated powerful and complex programming functions while being easy-to-use, making it suitable for a classroom environment [8]. One team used just an Arduino [11] to run their autonomous vehicle but most used a combination of Arduino and the Robot Operating System (ROS). ROS made it easy for them to implement other libraries in Arduino and to communicate between codes. Otherwise, the software was very similar to our model, but the approaches for steering and speed control varied significantly. For the hardware components, other teams used their own cameras and motors to run their vehicle, but the most important differences are within the construction, namely the number or location of the wheels and the size of the vehicles.

Different approaches can be pursued in building the autonomous platforms, including different wheel configurations such as three-wheels [9], [11]. A three-wheel approach renders steering easier. Other designs were based on six-wheel platforms or treads instead of actual wheels [9], [13]. Another key distinction found in the research was that many approaches had a much smaller vehicle than ours, simultaneously decreasing price and increasing steering control and speed. While this smaller size may seem beneficial, the downside is that it leaves less space for other hardware components, so the larger size of our vehicle made it more powerful than several of those mentioned [9].

In this study, we developed a more realistic autonomous platform that is closer in design to a real-size vehicle than those described in other papers. The vehicle had the same number of wheels and shape as vehicles used today by consumers. It also incorporated features such as obstacle detection, lane tracking, and sign detection, all features that are similar to those used in full-scale autonomous vehicles. Unlike full-scale vehicles, our platform was more cost-convenient and used mostly off-the-shelf hardware, thus allowing for further customization and expansion. In line with previous research, the processing pipeline was built using ROS and Arduino and facilitates the inclusion of extensions as well as supporting reproducibility.

III. Overview of Hardware System

The 1/10-scale autonomous vehicle had a variety of hardware components and was built on a short-course racing truck due to how its size allowed for the other hardware to fit on top (see Fig. 1) . The truck used had a Magnum 272 transmission with 2-wheel Drive and a torque-control™ slipper clutch. We used a waterproof Titan 12-Turn 550 motor equipped with XL-5 electronic speed control and comprehensive steering capabilities. This model is very similar to real-world driving experiences and allows the vehicle to drive in a variety of weather conditions without breaking. Full specifications can be found in [14].

For the physical computing system, we used an NVIDIA Jetson Xavier [15], [16] which runs the Ubuntu 18.04 Operating System to incorporate our software [17]–[19]. The computing platform was chosen for its small size and lightweight while still being very powerful.

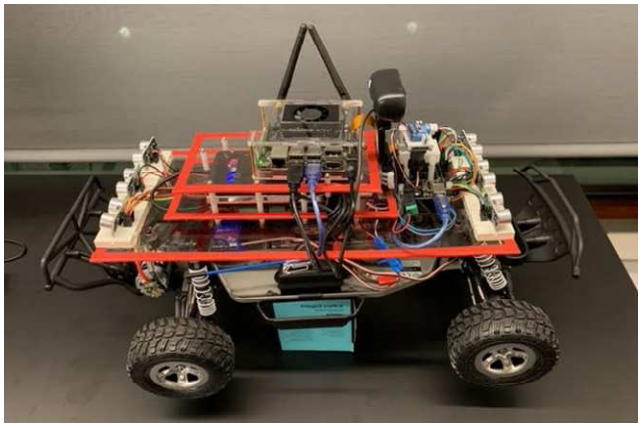

Fig. 1. Image of the vehicle showing its components. The distance sensors are visible at the front and back of the vehicle as well as the computing system, camera, and wheels.

An Arduino board is used to run our PID-based steering control algorithms. The one we chose was an Arduino MEGA 2560 (see Fig. 2) [20]. The board has 54 digital input/output pins, 16 analog inputs, and 4 serial ports. The board sends commands to the motors to make the vehicle function in our driving environment. For our vehicle to be able to see the road and run the lane-tracking and road sign recognition features, we needed a high-resolution camera. This enabled us to collect real time data which was converted into instructions that told the vehicle to accelerate, decelerate, and turn. The image data is sent from the camera to NVIDIA Jetson Xavier, where calculations are made and steering/speed values are sent to the Arduino.

Fig. 2. Arduino Mega2560. Image credit [21].

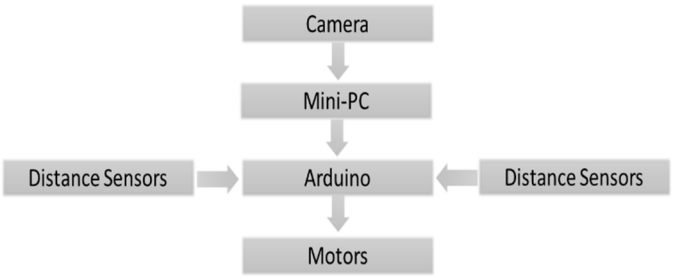

Fig. 3. The component architecture of the Autonomous vehicle.

As shown in Fig. 3, the camera and the distance sensors communicate with the Arduino code running on the mini-PC NVIDIA Jetson Xavier. The PID algorithms in the Arduino code convert the data collected from the outside world into speed and data values which are communicated with the motors. The high-resolution camera is mounted on top of the chassis while distance sensors are placed in the front and back of the chassis.

Finally, to test the obstacle detection we employed HC-SR04 ultrasonic sensors. These detect an obstacle within the range of 0.02m to 4m (see Fig. 4). The sensor sends out ultrasonic waves and measures the time from when the wave is sent out to when the receiver detects the reflection of the wave. This is then converted to the distance from the obstacle that reflected the wave. Three sensors were placed in front and three were placed in the back. In this work, only the front sensors were used.

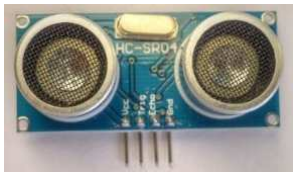

Fig. 4. HC-SR04 ultrasonic sensor. Image credit [22].

The experimental setup also included a driving environment (see Fig. 5) created by putting white tape on the floor of our lab and modeling it after the lane markings used in the real world. The entire environment is built to scale, including the vehicle and several stop signs and obstacles which can be added to simulate the vehicle running in the real world. This was used as a training platform to fine-tune aspects of the code and to run various experiments.

Fig. 5. The driving environment.

IV. Overview of Software System

The main software components of the autonomous vehicle are Robotics Operating System (ROS), Python, and Arduino, which run simultaneously on an Ubuntu Operating System. Ubuntu is an open-source operating system (OS) that is based on Linux, and has uses in a variety of fields, including autonomous vehicles and machine learning. For the operating system used on the mini-PC, we used Ubuntu version 18.04 [19].

We used the Arduino Integrated Development Environment (IDE) to write the code for the vehicle and the Arduino board to run it. The combination of hardware and software was used to compute the angle and speed for the vehicle and to communicate with the motors in the vehicle. Arduino can be used in Windows, Linux, and Mac and provides a simple and easy-to-use programming environment. In addition to ROS, Ubuntu, and Arduino, our vehicle also used Microsoft Visual Studio to run the python programs for road sign detection and lane tracking. ROS was implemented into these programs and connected them with the Arduino PID [5], [10], [25], [26].

ROS is a collection of open-source robotics libraries and software. It is a platform where various entities can provide their software and libraries for people interested in robotics to use. For our project, we used ROS to communicate between the sensors and programs related to them and the Arduino PID to run the robot's mechanical components (steering and speed). We made use of ROS publishers and subscribers to do this, publishing the data in specific channels that the Arduino code listened to [23], [24]. The publisher sends data values to the ROS channel from which the subscriber in the Arduino code pulls the data. From there it is converted into speed and steering values.

V. Approach

To run the platform, two sensor data streams generated from the ultrasonic sensors and camera were used to adjust the speed and steering of the vehicle. Such adjustment was done in stages. First, camera data was processed to provide estimates for the deviation from the lane (see Fig. 3). Additional work where camera data are also used for traffic sign detection and interpretation is ongoing. In this project, simulated data for sign information was used. Finally, distance sensor data provided estimates for how far potential obstacles were located and were also used to adjust the speed.

The lane tracking module analyzes video data and generates a stream of values that correspond to correction angles (in the range of -180 to 180 degrees). The module transformed the perspective to see the lines and used trigonometry to calculate the deviation from the center of the path. The deviation stream was then sent using an ROS publisher as an integer value and was received by a subscriber in the Arduino code that ran on the same channel. The offset angle was then added to the angle where the vehicle is straight and thus acted as a correction. If the angle created as a result of the calculation was greater than 180 or less than 0, it was rounded to 180 and 0 respectively (see Fig. 6). This angle would then be input in a function that controlled the speed and steering of the vehicle.

The second camera data stream was produced by a module that estimated the distance from a stop sign. The module generated a percent certainty that there was a stop sign in the

view of the camera and would output the distance from that sign once a threshold certainty value was reached. This, like the lane tracking program, would then be published on a different ROS channel. The corresponding subscriber in the Arduino code receives the data and converts it to a speed value based on this distance. For example, if the distance was less than 60 cm and greater than 50 cm, the speed would be changed to the set value of 130 cm for that interval. The vehicle would gradually slow down until it reached a distance less than 20 cm from the stop sign, after which it would stop for 20 seconds before resuming normal operations.

```
New Steering Computation
    - Inputs: steering_value (received by ROS subscriber)
    - Updates: newsteering

newsteering = (int)steering_value*1.3+90

if newsteering < 0
    // minimum steering value is 0
    newsteering = 0
else if newsteering > 180
    // maximum steering value is 180
    newsteering = 180
```

Fig. 6. Pseudocode for computing the direction based on the steering value received.

Next, data generated by the distance sensors were used to detect obstacles. Although the platform had sensors at the front and back of the vehicle (six in total, three in front, three in back), the vehicle was only moving forward, so only the front sensors were used. To compute the distance, we averaged the measurement of all three sensors. Then, based on the distance from the obstacles, similar to the stop sign distance, the car would slow down and stop at the fixed distance of 40 cm. This was all done directly through the Arduino board and did not need another ROS publisher or subscriber.

```
Adjusting Vehicle Speed and Steering
  - Inputs: frontdistance
  - Updates: steering and velocity (data used to control the
      platform)

if frontdistance ≥ 40
    //steering but stop vehicle
    velocity = 90

if 70 ≥ frontdistance ≥ 40
    //change to new steering and reduce speed
    steering=newsteering
    velocity=120

if frontdistance > 70
    //change to new steering and go full speed
    steering=newsteering
    velocity= 180
```

Fig. 7. Pseudocode for adjusting the speed and direction.

Figs. 6 and 7 show the pseudocode for the PID functionality that controls the speed based on obstacles and steering changes based on the lane tracking. As noted earlier, the speed values can range from 0 to 180, and in our case, based on the obstacle detection we either stop the vehicle, slow it down or run the vehicle at the maximum speed (Fig. 7). The steering adjustment factor (1/3) was empirically determined based on repeated measurements that determined that our platform had a slight deviation in steering.

VI. On-road Application Results and Analysis

The testing for our autonomous vehicle was done in three parts: obstacle detection, stop sign detection, and lane tracking. All of these parts involved the PID algorithm and the connections between sections using ROS.

A. Obstacle Detection

The criteria for this section of testing were whether the car detected an obstacle, and the platform reduced the speed promptly. Fig. 8 maps our results aggregated across the three sensors across 10 data points taken from a test of the vehicle. A board was placed in front of the vehicle to simulate an obstacle. The distance to the obstacle was measured. Simultaneously, the speed of the vehicle was also observed. The data was taken from an array being repeatedly updated which included the distance data from the 3 sensors (floats), the speed, and the steering values (integers).

The blue line in the figure maps the aggregated distance while the red dashed line maps the motor power. As noted earlier, a power level of 90 means that the vehicle is stopped. The obstacle detection sensors did exhibit a change in distance and the change varied correctly based on the distance from the board. This graph also displays the speed at which the vehicle reacts to obstacles and demonstrates that our vehicle was very successful in adjusting to obstacles.

In addition to this, the wheels correctly changed their turning speed based on the thresholds of 100 cm, 70 cm, and 40 cm, where 40 cm is when the vehicle stopped, and 70 cm-40 cm is when the vehicle began to slow down.

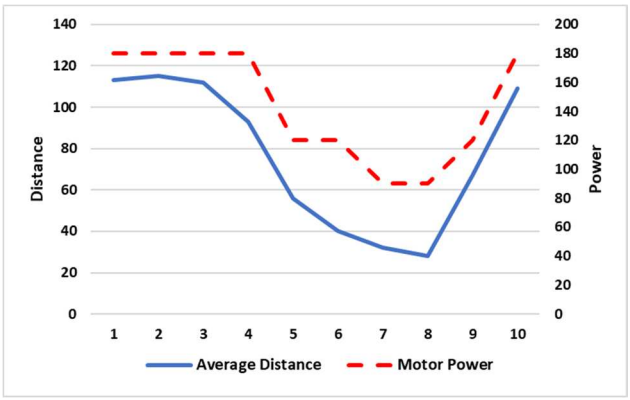

Fig. 8. The plot of the average distance measured by the ultrasonic sensors (blue line) and the adjusted motor power (red dashed line).

Therefore, both the obstacle detection and PID portions of the Arduino code were fully functioning for this part of the experiment.

B. Stop Sign Detection

The stop sign detection code is yet to be incorporated on our platform due to version incompatibility between the computer vision libraries and the software installed on the mini-PC. As

such, rather than using real data, we generated random sequences to be sent by the ROS publishers. In our experiments, we noticed that the speed changed accordingly, and the wheels also slowed down or sped up depending on the distance received. The speed ranged from 90 to 180, where the vehicle stopped at 20 cm from the sign and went at full speed 50 cm before the sign. In addition, the vehicle did stop for 20 seconds once the value received was less than or equal to 20. This showed that the ROS publisher and subscriber communication worked and that the PID was functioning correctly.

C. Lane Tracking

For lane tracking, the vehicle changed its steering value based on the value received from the lane tracking module and turned the wheels accordingly. The code was tested on the track, and the lane tracking code sent out values while the PID received and converted them accordingly. The PID also rounded the values to 180 and 0 for certain situations as mentioned above. Therefore, the PID and ROS publisher and subscriber communications were also working for this part of the experiment. The vehicle was successful in completing a lap around the track and staying in the lane.

VII. Conclusions and Future Work

In this study, we have developed PID-based steering control algorithms to maneuver a 1/10-scale autonomous vehicle in a real-world scaled-down driving environment. The state-of-the-art ROS has been employed in the software development and vehicle control. The PID and ROS implementation functioned correctly and was able to respond in a timely manner in our experiments. The integration of sensor data modules with control modules meant that values were being sent from the programs for lane tracking and stop sign detection and were being received by the Arduino PID and being converted. There are a variety of things that can be improved about this specific aspect of the project, namely the speed of the conversions and the accuracy of the values received. Currently, every value used is an integer and has to be in order for the program to work. In the future, we can potentially change the values to floats in order to use decimals and make the vehicle more exact. The increased speed of the vehicle's operations could also help the vehicle make decisions quicker. To make the model even more realistic and closer to a production car, complete additions of stop sign detection and even obstacle avoidance need to be added. In addition, detecting crosswalks, other lane markings, and other vehicles are part of our next steps for research. Gathering data by testing the obstacle detection sensors and comparing that to the actual distance will help us get a better idea of how well the model is performing compared to other similar autonomous vehicle designs. Speed tests can also provide useful data and insights. In future work, we will also develop new and faster approaches to improve and compare the performance of our current solutions when related to production vehicles with autonomous features.

References

[1] S. Pettigrew, Z. Talati, and R. Norman, "The health benefits of autonomous vehicles: public awareness and receptivity in Australia," *Australian and New Zealand journal of public health*, vol. 42, no. 5, pp. 480–483, 2018.

[2] R. E. Stern *et al.*, "Quantifying air quality benefits resulting from few autonomous vehicles stabilizing traffic," *Transportation Research Part D: Transport and Environment*, vol. 67, pp. 351–365, 2019.

[3] S. D. Pendleton *et al.*, "Perception, planning, control, and coordination for autonomous vehicles," *Machines*, vol. 5, no. 1, p. 6, 2017.

[4] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An open approach to autonomous vehicles," *IEEE Micro*, vol. 35, no. 6, pp. 60–68, 2015.

[5] "PID control explained. PID control is a mathematical approach… | by Mattia Maldini | Medium." https://maldus512.medium.com/pid-control-explained-45b671f10bc7 (accessed Oct. 29, 2021).

[6] L. Paulino, M. Zhu, and W. Wang, "Learning Autonomous Driving in Tangible Practice: Development and On-Road Applications of a 1/10-Scale Autonomous Vehicle," *in 2021 IEEE Frontiers in Education Conference (FIE)*, 2021: IEEE, pp. 1-4.

[7] W. Wang and L. Paulino, "Instill Autonomous Driving Technology into Undergraduates via Project-Based Learning," *in 2021 IEEE Integrated STEM Education Conference (ISEC)*, 2021: IEEE, pp. 1-3.

[8] Y. A. Badamasi, "The working principle of an Arduino," in *2014 11th international conference on electronics, computer and computation (ICECCO)*, 2014, pp. 1–4.

[9] K. S. Alli *et al.*, "Development of an Arduino-based obstacle avoidance robotic system for an unmanned vehicle," *ARPN Journal of Engineering and Applied Sciences*, vol. 13, no. 3, pp. 1–7, 2018.

[10] M. Fezari and A. Al Dahoud, "Integrated Development Environment 'IDE' For Arduino," *WSN applications*, pp. 1–12, 2018.

[11] M. C. De Simone and D. Guida, "Identification and control of a unmanned ground vehicle by using Arduino," *UPB Sci. Bull. Ser. D*, vol. 80, pp. 141–154, 2018.

[12] S. Evripidou, K. Georgiou, L. Doitsidis, A. A. Amanatiadis, Z. Zinonos, and S. A. Chatzichristofis, "Educational Robotics: Platforms, Competitions and Expected Learning Outcomes," *IEEE Access*, vol. 8, pp. 219534–219562, 2020.

[13] F. M. Lopez-Rodriguez and F. Cuesta, "An Android and Arduino Based Low-Cost Educational Robot with Applied Intelligent Control and Machine Learning," *Applied Sciences*, vol. 11, no. 1, p. 48, 2021.

[14] "Slash: 1/10-Scale 2WD Short Course Racing Truck with TQ^{TM} 2.4GHz radio system," Nov. 25, 2013. https://traxxas.com/products/models/electric/58024slash (accessed Oct. 29, 2021).

[15] "Deploy AI-Powered Autonomous Machines at Scale," *NVIDIA*. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/ (accessed Oct. 29, 2021).

[16] H. A. Abdelhafez, H. Halawa, K. Pattabiraman, and M. Ripeanu, "Snowflakes at the Edge: A Study of Variability among NVIDIA Jetson AGX Xavier Boards," in *Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking*, 2021, pp. 1–6.

[17] "Index of /releases/18.04.5." https://old-releases.ubuntu.com/releases/18.04.5/ (accessed Oct. 29, 2021).

[18] "Six reasons why developers choose Ubuntu Desktop | Ubuntu." https://ubuntu.com/engage/developer-desktop (accessed Oct. 29, 2021).

[19] M. Helmke, *Ubuntu Unleashed 2019 Edition: Covering 18.04, 18.10, 19.04*. Addison-Wesley Professional, 2018.

[20] "Arduino Mega 2560 Rev3," *Arduino Official Store*. https://store.arduino.cc/products/arduino-mega-2560-rev3 (accessed Oct. 29, 2021).

[21] A. Dingley, *Arduino Mega2560*. 2011. Accessed: Oct. 29, 2021. [Online]. Available: https://commons.wikimedia.org/wiki/File:Arduino_Mega2560.jpg

[22] N. Dilmen, *Ultrasonic sensor*. 2014. Accessed: Oct. 29, 2021. [Online]. Available: https://commons.wikimedia.org/wiki/File:HC_SR04_Ultrasonic_sensor_1480322_3_4_HDR_Enhancer.jpg

[23] "ROS: The ROS Ecosystem." https://www.ros.org/blog/ecosystem/ (accessed Oct. 29, 2021).

[24] "ROS: Why ROS?" https://www.ros.org/blog/why-ros/ (accessed Oct. 29, 2021).

[25] "Getting Started with Arduino products." https://www.arduino.cc/en/Guide (accessed Oct. 29, 2021).

[26] "Servo - Arduino Reference." https://www.arduino.cc/reference/en/libraries/servo/ (accessed Oct. 29, 2021).