Developing an A.I. Self-Driving Power Wheels Car

Parker W. Lauders

B.S. in Computer Engineering Technology (Est. May 2026)

B.S. in Computer Science (Starting in Fall 2024)

Minoring in CADD

Department of Engineering Technologies

Shawnee State University

Portsmouth, Ohio 45662

November 28, 2023

Author Note:

This project was personally funded by Parker Lauders with donation of the Power Wheels Jeep from Jake Holbrook and the donation of the Arduino Mega 2560 from Lorie Miracle during the summer of 2023.

Table of Contents

Table of Contents (Continued)

Table of Figures

Table of Figures (Continued)

Table of Figures (Continued)

Abstract

This research project comprehensively explores a self-driving AI-powered vehicle's design, development, and evaluation. Beginning with the selection and integration of hardware components, the transformation of a Power Wheels car into an autonomous platform is detailed. Control and vision systems, combining mechanical and electrical engineering with software development, form the project's backbone. Integrating sensors, cameras, and a control system showcases the synergy of these disciplines. Utilizing the Jetson Nano Developer Kit and Arduino Mega underscores the incorporation of embedded systems into autonomous vehicles. The A.I., trained on a meticulously prepared dataset, is at the core of self-driving capabilities. Data preprocessing, encompassing image compression, scaling, color rendering, and standardization, ensures a robust dataset for efficient machine learning model training. The Convolutional Neural Network (CNN) demonstrates the AI's proficiency in interpreting visual data from a camera and translating it into actionable information for real-world navigation. Evaluation of self-driving capabilities highlights achievements in obstacle detection and response, consistent stopping and resuming of motion. However, imprecise steering control, mainly during turns, indicates the need for further refinement in the A.I. model. Future improvements, including a computer-vision program for path detection and trajectory prediction and an upgraded object detection sensor setup for comprehensive awareness, illustrate a commitment to safety and continuous enhancement. As A.I. technologies evolve, this project lays the groundwork for future advancements in autonomous vehicle innovation.

Keywords: self-driving vehicle, A.I.-powered, hardware integration, Power Wheels transformation, autonomous platform, control systems, vision systems, mechanical engineering, electrical engineering, software development, sensor integration, camera integration, Jetson Nano Developer Kit, Arduino Mega, embedded systems, A.I. training, dataset preparation, data preprocessing, image compression, scaling, color rendering, standardization, machine learning model, Convolutional Neural Network (CNN), visual data interpretation, real-world navigation, obstacle detection, steering control refinement, computer vision program, path detection, trajectory prediction, object detection sensor, safety enhancement, continuous improvement, A.I. technologies, autonomous vehicle innovation.

Objectives

This project aims to design, develop, and evaluate the accuracy of an A.I. self-driven vehicle model capable of autonomously navigating its environment. By utilizing the latest versions of technologies that focus on artificial intelligence, robotics, and computer vision, this project aims to achieve the following:

- Auto-Braking System for Safety:

  Develop and implement an auto-braking system that utilizes sensors and algorithms to detect objects in the vehicles' path of concern. The system should be capable of accurately recognizing obstacles in the vehicle's environment, such as pedestrians, vehicles, or stationary objects. The system should then be able to automatically brake appropriately based on the vehicle's position to these obstacles.

- Self-Decided Vehicle Control:

  Design and integrate intelligent vehicle control mechanisms that enable the vehicle to autonomously maintain a particular position in a lane. Develop and integrate A.I. algorithms for steering and speed regulation to facilitate smooth and accurate lane control. The self-driving vehicle control system should adapt to the changing environment and road conditions.

- Real-Time Performance:

  Optimize the overall system performance to achieve as close to real-time performance as possible. Minimize latency and ensure quick response times between decision-making and control actions. This requires efficient algorithm design, optimized code implementation, and proper hardware utilization.

- Seamless System Integration:

  Ensure seamless integration of the A.I. self-driven vehicle's various software and hardware components. Integrate object detection sensors, processing units, control mechanisms, and controls to work uniformly together. Develop effective communication protocols and enable reliable and efficient data flow between system modules.

Introduction

Combining the rapidly growing artificial intelligence industry with the booming electric car industry, artificial intelligent vehicles have begun to be presented as the method of travel in the upcoming years on roads around the globe. Car companies such as Tesla, one of many companies competing in the private consumer market for electric vehicles, or Waymo, a company known as a strong pioneer in the rideshare industry, have aided in pushing this ideology of self-driving autonomous vehicles on the roads to reality. The introduction of self-driving cars is drastically revolutionizing the methods used in everyday transportation, like the boom sparked by the original automobiles nearly a century ago. Through this study/development, this project aims to learn from creating an A.I. self-driven power wheels car while exploring the crossings of modern A.I. technology and mobility with embedded system integration.

Theoretical Background

Artificial intelligence, commonly known as A.I., is a prominent field in computer science that incorporates many technologies and applications. One of the earliest pioneers in the field was Alan Turing, renowned for coinventing the *Bombe* machine (The National Museum of Computing, n.d.), which cracked the German *Enigma* code and his foundational contributions to computer science. In his 1950 published journal, "*Computer Machinery and Intelligence,"* Turing introduced the fundamental question: "Can machines think?" He introduced a test known as "The Imitation Game," commonly known as "The Turing Test," which has become a famous benchmark in artificial intelligence. The Turing Test aimed to determine if a machine's behavior could demonstrate intelligent behavior that a human being is capable of. To test this, Turing created a game with three players: Player A, player B (the opposite gender of Player A), and an interrogator who can be of either gender. Players A and B are held in a separate room from the interrogator, with the objective of the game being for the interrogator to determine which player is male and which player is female by only questioning one player via writing. From this, Turing believed that "in about fifty years' time it will be possible to programme computers, with a storage capacity of about 10^9, to make them play the imitation game so well that an average interrogator will not have more than 70 per cent, chance of making the right identification after five minutes of questioning" (Turing, 1950, pp. 433-442).

Turing knew little that this idea of programming machines to think in similar objective-based ways as humans would revolutionize the way things are accomplished less than 100 years later. This is drastically accomplished with machine learning, a sub-area of artificial intelligence "concerned with intelligent systems that learn." Machine learning is one of several areas of artificial intelligence this paper mainly focuses on. Machine learning, with its origin being credited to psychologist Frank Rosenblatt from the University of Cornell, was created based on the "ideas about the work of the human nervous system." The first artificial neural network was created from this foundation of ideas in 1957. Called "perceptron" by Rosenblatt, the neural network was able to recognize the letters of the alphabet by using a combination of "analog and discrete signals that included a threshold element that converted analog signals into discrete ones" (Fradkov, 2020, p. 1385). Machine learning artificial neural networks are made with one goal in mind: to mimic the human neural network via electronic systems and algorithms. With this, machine learning is the closest form that bridges how humans learn in their everyday lives with artificial intelligence. This is done by "learning results from the interaction between the agent and the world and from observation of the agent's own decision-making process" (Boden, 1996, p. 89).

The sub-section of machine learning used in the development of this project is known as deep learning; this type of machine learning "enables computers to solve perceptual problems such as image and speech recognition." Although the two types of artificial intelligence work towards the previously mentioned goal of mimicking the human brain, the data used by the deep learning method are more drastic. While both still use artificial neural networks, deep learning "use multiple processing layers to discover patterns and structure in very large data sets" these layers start simple and get more complex as the layers increase (Rusk, 2016). An example relevant to this project would be a deep neural network in charge of getting the data on a car's placement on a road. The first layer would detect the edge of the road, where the pavement meets the material surrounding the road. The next layer could pick up the white lines on the edge of the road if present, while the next layer gets the yellow lines and determines if it is dashed or solid. Then, the following layers could use various techniques and algorithms to see the shape of the road the car is in and measure its position based on the center of the car to the center of the road.

The second sub-area of artificial intelligence this paper will focus on is machine vision. Machine vision began its journey only a few years after Turing's revolutionizing question. It has continued to improve and develop in advanced ways and shows no sign of slowing down soon. Machine vision aims "

to segment the world into objects, to classify these objects into categories, and to describe their geometric structure" (Boden, 1996, p. 183). With the combination of machine learning and machine vision, the ability to have a computer recognize and learn based on its environment or objects presented at hand and adapt to them is limitless. This is the primary foundation of the artificial intelligence system this project is built upon.

Methods and Procedures

This research project paper presents the design, development, and evaluation of a self-driving A.I.-powered vehicle by using a Power Wheels children's car as the platform. The primary objectives of this study/development are to explore the integration of hardware components, control and vision systems, data collection techniques, machine learning model development, and methods to enhance the vehicles' overall autonomous capabilities.

Design and Development of Hardware Components

One of the most critical aspects of designing and developing an autonomous A.I.-powered vehicle is designing and developing hardware components. The vehicle needs a base platform, with dimensions, weight, power requirements, and compatibility with extra parts all being important considerations. I will move forward with integrating required parts, including but not limited to sensors, actuators, microcontrollers, single board computers (SBC), power management systems, and communication modules, after the optimum option for the vehicle platform has been determined.

When it comes to the sensors, I choose to employ a combination of sensors to gather various sorts of data crucial for the coding and training the autonomous A.I. model for operations. This includes, but is not limited to, cameras for perception based on computer vision and distance sensors for accurate distance calculations of nearby objects. I regard the integration process for actuators—like motors and servos—as making it possible for the vehicle to have a microcontroller or SBC to control the speed and steering. The

evaluation of the control system integration, the mechanical installation of the selected actuators to the vehicle's chassis, and the discussion of the actuators' specifications. A description of how microcontrollers and onboard SBC are used to manage data processing, sensor integration, system control, indicator control, and data processing. The choice of capable microcontrollers and SBCs, the programming environment, and the communication protocols are used to integrate data across various components and systems.

Control and Vision Systems

The control and vision systems play a vital role in this project by enabling the self-driving A.I. to be aware of its surrounding environment, collect informative data to feed the A.I. and move as the A.I. intends. This is vital to the vehicle's A.I. system due to the inability of machines to perceive surrounding areas in the way a human or animal would, meaning that all computer vision must be done accurately, and controls must be done accordingly to ensure that the movements are executed adequately to the desired action. The integration of these two must also maintain a seamless connection to address the limitations the A.I. faces in perceiving and responding to the world. I will outline the methodologies and procedures used to ensure the total effectiveness of coordination between the perceptively gathered data and the performed control actions. I will also detail how these two systems communicate to ensure that what is desired becomes a reality and that what is reality is considered when making the following calculation toward the desired action.

The computer vision system is responsible for interpreting visual data captured, enabling the vehicle to "see" and fully understand its surrounding environment. To better understand how the perception is accurately achieved and acted on, the process of the calculations for the desired actions based on captured visual data will be outlined along with these methodologies and procedures. The choice of sensor for the vision system, consisting of cameras, is based on the camera's ability to achieve foundational solid factors. Factors such as the camera's resolution, sensitivity to outside conditions, field of view, and ability to autofocus will be examined. Captured image data undergoes a preprocessing procedure, including image cropping, color conversions, blurs, resizing, and lastly ending with normalization are applied. Following this, the image data is fed to a deep learning model trained on a specific dataset, enabling the car to maintain

a position centered on the lane while understanding its environment, such as where the road surface is and is not. The "*Machine Learning and Model Development"* section details the model architecture and training procedures.

The control system is responsible for interpreting the data from the vision system's A.I. model outputs and applying them to real-world vehicle movements. The following methodologies and procedures ensure this is completed precisely and consistently. The integration of actuators mentioned in the previous *"Design and Development of Hardware Components"* section enables the translation of electronic control signals into physical actions that can be carried out. The control system, built via a microcontroller, is a translator between the A.I. model and the power wheels components. The sole purpose of the microcontroller is to receive the desired movement output via the A.I. model and then translate it into the corresponding physical actions, such as steering for actuators or speed control for drive motors. A robust communication protocol that can send data with a low loss or distortion rate is employed to ensure that the data exchange between the two systems is accurate and that the exchange can be completed as close to real-time. Integrated into the control system are sensors that collect data for the system to function independently from the vision system to ensure that integrated safety features are held at a processing speed closest to real-time.

Data Collection Techniques

Collecting a robust and compatible dataset is a fundamental step in preparing for the neural network, the brains of the self-driving operations. To accomplish this, several processes and methods must be executed to ensure the accuracy and reliability of data collection. The first process is where the choice of camera for the vision system comes into play; the front-facing camera is required to provide a clear and accurate perspective of the road ahead and the surrounding environment, such as the grass beside a country road. To capture accurate and usable data, a manual operation method is required to ensure the data is to the desired output expected by the neural network, which will operate the self-driving ability in the future. To ensure the full potential capabilities of the neural network, diverse data is needed to show various factors

of road conditions, such as weather and lighting conditions; an extensive dataset is also required to provide the neural network with a reliable amount of data.

The commonly used data points in self-driving neural networks for cars include but are not limited to camera images with values of throttle and steering. A procedure is needed to record these required data points accurately. With this comes the need for a method to ensure that the captured data is accurate in correspondence to one another to provide an accurate overall dataset. Also, a method for saving the data points to their directories with the proper labeling of each type is needed. By doing this, the dataset will be accurate, and the data will be usable and will not be contaminated with inaccurate data. Then, data preparation techniques will need to be used to ensure optimal performance and that the data obtained is beneficial when training the neural network.

Machine Learning Model Development

At the core of achieving autonomous driving abilities lies the development of a machine learning model. Before training the neural network, a method is needed to preprocess and augment the images captured during data collection. This form of data preprocessing is different from what may be for numerical values, such as the throttle or steering data. Data preprocessing for image data consists of image edits known as "data augmentation," which brings more diversity to the original dataset. The reason for this is that "one of the main advantages of preprocessing data is that it helps to improve the accuracy of the model. By cleaning and formatting the data, we can ensure that the algorithm is only considering relevant information and that it is not being influenced by any irrelevant or incorrect data" (Singh S. , 2023). This means that the neural network training program must incorporate several data augmentations best suited for a self-driving car model while maintaining a process in place to complete a successful model training.

The proper choice of model architecture also plays a vital role in training the neural network. Neural networks rely on various forms of architectures; these architectures contain different "abilities" that, in return, allow the neural network to complete function in various ways. Selection of the correct architecture for the model is drastic as it controls what the neural network can accomplish, meaning that without the correct one, the AI can be irrelevant when it comes to the goals upon it. This matters because neural

networks are composed of multiple interconnected layers, with the architecture playing a role in shaping the connected layers. With this factoring into the shape of the layers, it also means that it factors into the overall speed and ability for the A.I. model. Based on the architecture also lays the side factor of model complexity, this can result in the models' required resources in both the training process and the real-world implementation process(s). This means a procedure for choosing the correct architecture must be in place to ensure the overall structure and performance of the A.I. model align with the goals of this project.

To ensure the model produces the desired number of outputs based on the desired number of inputs, the number of nodes per layer and the total number of layers in the neural network play a role in structuring the model to predict in the desired ways. Like a blueprint, this maps the model from the beginning to the end, ensuring that everything desired becomes reality. The way these layers are shaped, like the selection of the model's architecture, plays a vital role in the model's implementation speed, along with the resources used. For this reason, a process must be in place to ensure that the model is not only reaching or aiding towards the project's desired goals but also that the model can be run on the selected hardware.

Results and Discussion

This section thoroughly overviews how my self-driving AI-powered vehicle was designed and developed. It covers hardware component selection and integration, control and vision system integration, data gathering methods, machine learning model construction, testing and evaluation procedures, and ability enhancement methods.

Design and Development of Hardware Components

With the donated Power Wheels vehicle being used as the base platform for this research project, only modifications to the vehicle's chassis were required to house additional hardware components. This is a solid foundation for a vehicle's platform because the total size, weight, and power needs match the size of a conventional Power Wheels car. The sensors chosen were the HC-SR04 ultrasonic sensor module based on the low current draw while maintaining a high working frequency and range; a front-viewing Logitech C920 Pro webcam with 1920 x 1080 pixels at 60 frames per second capabilities was installed to control the computer vision system allowing for precise and smooth visual input. To manage the power wheels steering

of the front tires, an 80KG (176.37 lbs.) max torque rated servo was added to the steering rod for high torque situations and to ensure the steering is never an issue. BTS7960 high current (43 amps) motor driving H-Bridge modules were connected individually to the original power wheels' 12V D.C. motors to enable an external system to power them in various directions and speeds using a pulse width modulation (PWM) signal. The Arduino Mega 2560 REV 3 was selected for the control systems microcontroller based on the 54 GPIO ports and the operating speed/power provided by the ATmega2560 microcontroller. For the A.I. system, the SBC used is the Jetson Nano Developer Kit; this features the low-cost yet powerful Jetson Nano module on a breakout board, allowing for the A.I. application to be powered by the 128-core maxwell GPU while having access to easy usage of the GPIO pins. Two separate power systems, each consisting of a 12V D.C. 9Ah rechargeable lead-acid battery, were installed into the vehicle, one powering the processing systems and steering servo via voltage step-down converters.

In contrast, the other battery powers the two 12V D.C. drive motors via the BTS7960 motor drivers. Together, these batteries are estimated to allow for the full power of both systems for roughly two to three hours of continuous usage on a full battery. The two batteries are recharged via a cigarette lighter socket installed on the back of the power wheels, in place of the original fake gas cap, by a 12V tringle charger. Communication is handled by an I2C protocol connection between the control system and the A.I. processing system via a two-channel logic level shifter housed on the PCB breakout board for the Arduino Mega 2560 REV 3. Functioning safety indicators such as front and back left and right turn signals, back brake lights, and an LED status light on top to notify any nearby humans of the vehicle's desired movements were also installed onto the vehicle. Integrating the FS-iA6B remote control (R.C.) receiver allows a user to control the vehicle via remote control, allowing the power wheels vehicle to be driven around a determined path/route to collect the necessary data for the deep learning A.I. model.

Control and Vision Systems

This project's heart is the integration between the control and vision systems, which allow the power wheels to navigate their environment with accuracy and safety. To allow a secure and accurate connection between the systems, communication protocols were thoroughly researched and reviewed to ensure that the

data packets being sent/received do not encounter any unwanted changes between the two systems. Factors of functionality are what the decision of communication protocol was based upon, such as a low number of needed connections, reliability, and speed. The communication protocol I chose to develop this connection with is the I2C protocol; this was chosen due to meeting the desired choosing factors. The primary deciding factor was due to I2C communication being a synchronous protocol, meaning that the data bits being sent are synced to an internal clock that both devices use when communicating; this ensures that both devices receive the bits in the correct order, resulting in accurate data packages when reading (Kalinsky & Kalinsky, 2001) (Wikimedia Foundation, 2023) (kangalow, 2023) (Mankar, Darode, Trivedi, Kanoje, & Shahare, 2014).

In the case of this project, the vision system is the controller device (commonly called master) due to the A.I. being the central aspect in control, with the control system being the peripheral device (commonly called slave) due to it executing the desired A.I. outputs in the real world. To allow for this protocol, a logic level shifter was required to shift the voltage levels between the two systems due to the control system running on 5V and the vision system running on 3.3V. To accomplish this, placed on the control systems breakout shield is a two-channel logic level shifter based on the same schematic as Adafruit's four-channel shifter, which utilizes BSS138 MOSFETs in pair with 10kΩ pull-up resistors (Adafruit, 2014). This then means that to enable this communication, all that is needed is the control system's 5V VCC signal, the vision systems' 3.3V VCC signal, the GND to be shared between devices, the SDA lines, which is the line used to transfer data, and lastly the SCL liens, which is the line the clock is on to ensure the devices are in sync.

A Logitech webcam is located on the hood of the power wheels, which allows the vision system to gather a screenshot of the environment in front of the vehicle. This camera was used due to its compatibility with the Jetson Nano, along with the features the camera held. These features are the full high-definition resolution, auto-focus, and auto adjustment to the environment's brightness changes. Once this image goes through a small preprocessing step, which crops the image to the used area, followed by a color change and image ratio conversion with a slight Gaussian blur, the image is fed into the trained A.I. model. The model

then takes this preprocessed image and produces a command array from the image. Taking this command array, the data is separated into a bit for the direction of travel and two bytes, one for the steering angle and the other for the throttle speed. This is then run through a process that converts the command array from a scale of negative one to zero to positive one for the converted steering byte, then zero to one for the converted throttle byte into the usable ranges set in the control systems program; the direction data acts more in place as a Boolean variable for if the reverse direction is needed. Once completed, the vision system sends the newly formatted command array to the control system via the I2C bus connection. Then, it repeats the process as shown in the diagram below.

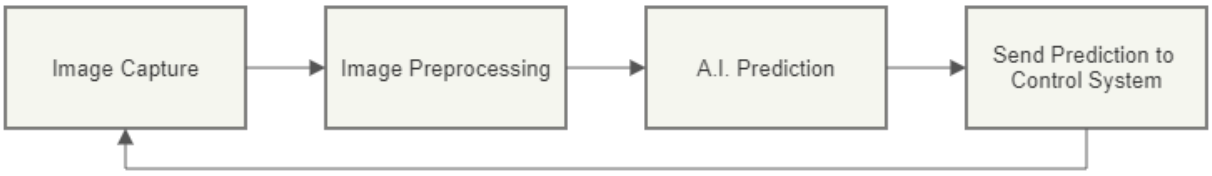


Figure 1. Vision System Real-Time Implementation Process Diagram.

As shown in the graph below, once the control system receives the command array via the Wire class, the control system acts upon the received data by setting the motors and steering servo angle to the vision system's desired values. Using the direction data, which consists of a single bit, if the direction is zero, the motor controller sends throttle data for the forward direction, with reverse throttle data sent if the direction bit is one. When the turn angle is more significant than half the entire range in either direction, the appropriate turning indicator will turn on a blink in a set interval on both the front and back of the vehicle. The brake lights functionality is handled differently; for the brake indicators to be activated, the car either must be stopped with the direction data set to one or physically moving in a reverse direction, which is indicated by the direction data being set to a value of one. For the safety of surrounding humans, the power wheel's original LED light was repurposed to act as an LED strobe light when the A.I. mode is activated, alerting humans in its surrounding area. All these features work with other functionalities in the control system with an operation speed as close to real-time as the equipment allows based on no delays besides Arduino's Servo class, which uses microsecond delays.

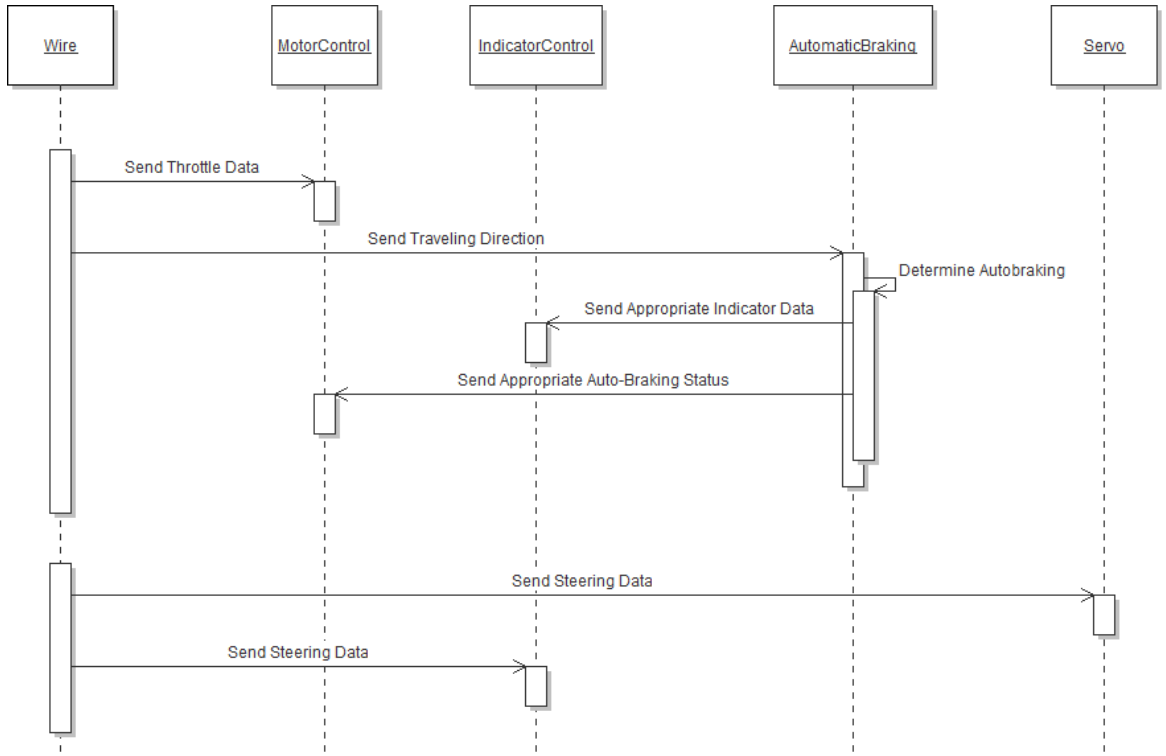


Figure 2. Sequence Diagram for the Control System When in A.I. Mode.

To ensure that the communication bus between the two systems is active is a function placed in the control system that checks the time between the last command array received and the current time when the difference between these two results in an amount greater than a specified interval the motors are disabled as this signifies that a disconnection of the communication bus has been made. A kill switch is another safety feature implemented in the two systems' communication. The kill switch, triggered by moving the right stick on the R.C. receiver into different corners to toggle it ON/OFF, allows a human to manually interrupt the communication process between the vision and control systems. When the kill switch is activated, the two systems are communicated. However, the actuators and signals are disabled from acting on the receiving command array data, causing the power wheels to be disabled. This was implemented instead of an emergency stop button for optimized safety and usability. It ensures that a human can turn off the power wheels from anywhere (within the R.C. receivers' range) without walking with or running after it to press the emergency stop.

To allow the power wheels to have a sense of location for items in the surrounding environment, located on the front and back of the vehicle, are HC-SR04 ultrasonics sensors. Using these sensors, the control system can detect objects from the left tire's edge to the correct tire's edge in either direction. This means that anything the power wheels could roll over is detectable. Using this ability, in place is an auto-braking system that functions by getting a scan of the HC-SR04 sensor in the direction of travel and then comparing it to a safety zone value. This value, set to four feet in this iteration, controls the distance an object must be to the front or back of the vehicle before the auto-braking is activated. Once the auto-braking is activated, the motors are disabled, and the braking indicators are activated. This remains for five seconds before checking to ensure the object has left the safety zone. This is done on the control system with no communication to the vision system due to the unmet need for the system currently to control the auto-braking. This decision was based on limiting the work the Jetson Nano would have to complete to help achieve the goal of real-time processing.

A small study on the accuracy of the auto braking feature was conducted using a human subject. To ensure the results were accurate, the human subject was asked to stand still while the vehicle approached them in a forward-moving direction. They were then asked not to move after the vehicle came to a complete stop; this allowed for the distance between the bumper and the subject to be measured. After repeating this ten times, a graph of the results was made to show these trials' test results.

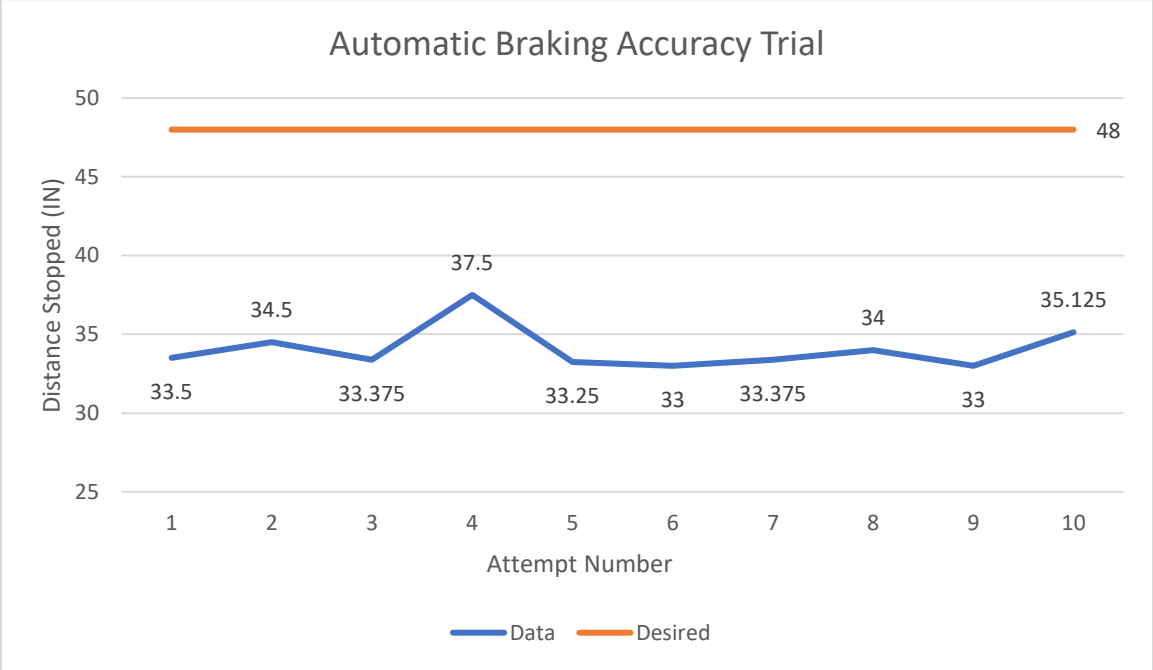


Figure 3. Automatic Braking Accuracy Trial Result Graph.

Depicted on the graph by the orange horizontal line is the targeted braking distance used in the automatic braking code, as the results from the trial show that the vehicle's ability to stop at this distance correctly was off by roughly twelve inches. This is due to three factors in place; the first is that the sensor is oriented at a 15-degree angle upwards on the front of the power wheels to prevent being set off by uneven road surfaces. This means that objects the sensor detects at this angle are closer than if the sensor was orientated to scan parallel to the driving surface. The second is the slight delay between the initial detection scan and the line of code that acts upon the result due to the processing speed at which the control systems microcontroller processes it. Although this delay is in the milli-seconds range, it still provides inaccuracy in the results based on the slight delay. The last factor is one of uncontrollable nature, this being the vehicle sliding due to the sudden stop. Like a real car, the power wheels car tires cannot gain enough friction with the road surface at first due to the traveling speeds, thus causing a few inches to be lost in the braking distance.

Data Collection Techniques

A reliable and extensive dataset was required before the neural network in charge of operating the self-driving operations could be trained. Data of the power wheels being driven manually via an R.C. controller was required to gain this dataset. To accomplish this, the control system was given an R.C. mode, which allowed the FlySky RC transmitter to completely control the power wheels, throttle, and steering abilities, as shown in the graph below.

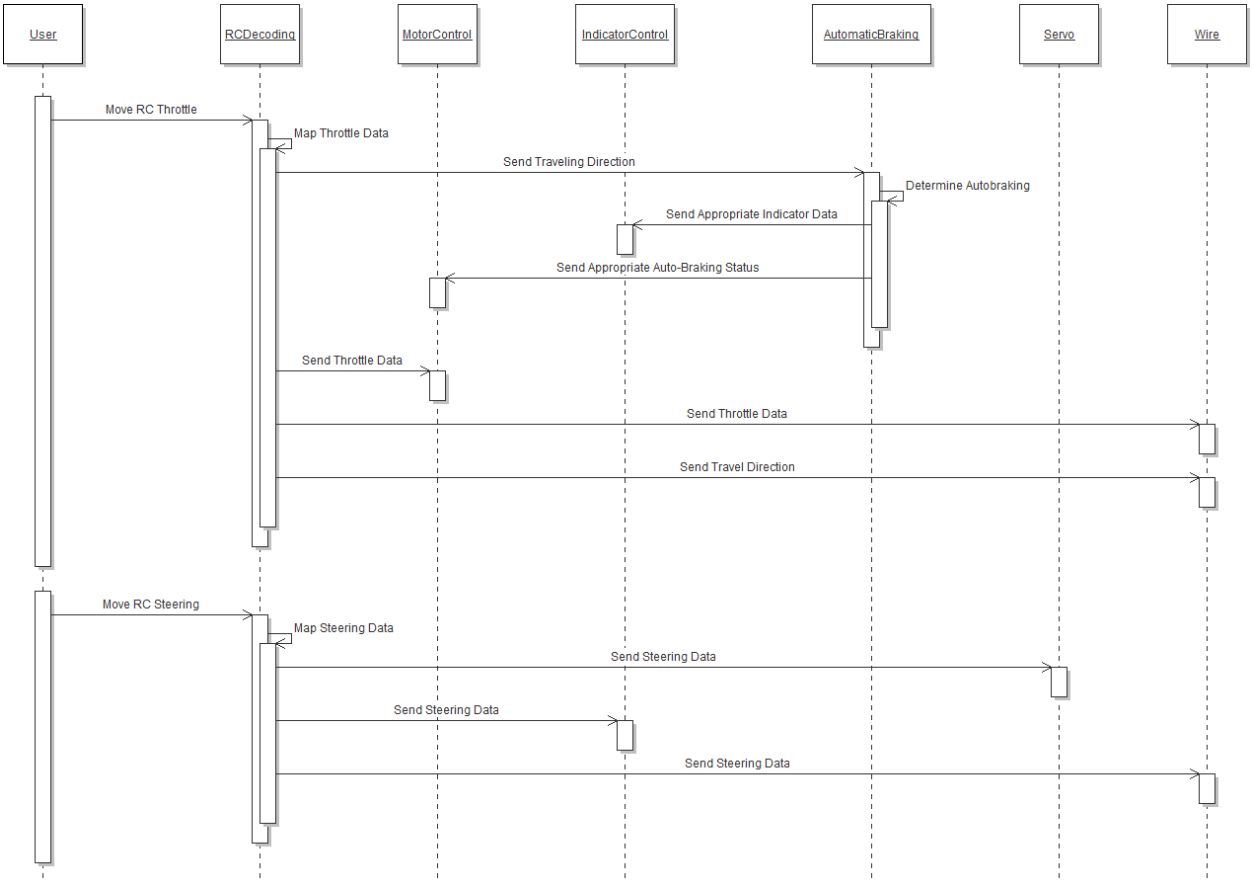


Figure 4. Sequence Diagram for the Control System's RC Mode.

Using a mapping method, the pulse width modulation signal of the R.C. receiver is read and then mapped appropriately to the corresponding instruction to be carried out via actuators (Tsuge, 2022b). Once the R.C. receiver's data is mapped, the throttle data is applied to the MotorControl class, the steering data is applied to the Servo and IndicatorControl class, and the direction is given to the AutomaticBraking class

to ensure a level of safety while in the R.C. mode. This allows the vehicle to be remotely controlled to the fullest of its abilities while incorporating the indicator functions presented in the A.I. mode.

A program is run on the vision system to collect data, allowing it to collect an image with its corresponding throttle and steering values. To accurately capture this, the vision system will capture an image from the webcam mounted on the hood using OpenCV and then request data from the control system (Intel Corporation, 2016). Before the control system sends the steering and throttle data to the vision system, the throttle data is manipulated to factor in the direction of travel. To do this, the throttle value is multiplied by a negative one if the direction bit is one, while the throttle value is left as a positive number if the direction bit is equal to zero. Once done, the control system places the steering and throttle data into an array and sends it via the I2C communication bus. Once the vision system receives the data from the control system, the data is split into the appropriate variables. Once split, the image taken is saved to a "captured_images" directory as a ".jpg" format, with the steering and throttle saved as a ".txt" format in similar separated directories. This is done at a rate of eight data points a minute, meaning that within an hour of data collection, roughly 500 data points were collected. Once this robust dataset was collected, a validation process was performed on the throttle and steering data to ensure that the values saved were in the proper range and that each ".txt" file had only one data point.

Once this is completed, a log is made as a ".csv" file. Inside the log file are three columns: a list of collected image paths, a list of the steering values, and a list of throttle values. While this log file is being produced, the program performs data scaling, which takes the collected steering data and remaps it to -1 to 0 for data in the 75 to 90 range, then 0 to 1 for data in the 90 to 140 range. It is mapped in this format due to 90 being the value at which the wheels are centered, while 75 is the maximum degree allowing for a left turn and 140 being the maximum for a right turn. The exact process is completed on the throttle data but on a range of 0 to 1 for the collected throttle data from 0 to 127. Data scaling is performed to make "data points generalized so that the distance between them will be lower," which presents a better neural network due to the minor difference being more accessible to process across the nodes from the input to the output

(Verma, 2021). The images' preprocessing (including data augmentation) occurs during model training, unlike the preprocessing and data scaling completed on the steering and throttle data.

Machine Learning Model Development

A big focus was spent on data preprocessing and A.I. model architecture to train the A.I. model for efficient use in this project. Due to the importance of preparing the data for the A.I. model, several sources were relied on to ensure the data was processed correctly and efficiently. After researching several sources, I found that the typical data augmentations performed during image preprocessing included normalization, image cropping and rotating, color conversion, and image scaling (Fredrick, 2021) (Sagar, 2019) (Singh A. , 2019) (Verma, 2021). Due to the images undergoing physical edits, the preprocessing with the data augmentations is completed during the model training process. This is due to the images being converted to matrix held via arrays for the TensorFlow model trainer to read, meaning a 1.12MB image file is converted into an 80 M.B. NumPy matrix file, not storing the arrays locally allowed for the hosting device not to need a high size storage device since the matrix files alone would have taken up roughly 400 GB of storage space.

The first step of image preprocessing was to crop the image to only use the valuable parts. This is one of the most crucial processes to optimize the A.I. model. This is because by removing the unnecessary scenery from the training and implementation images, the A.I. was given an image with the highest percentage of needed detail. With the unnecessary scenery included, the A.I. could get confused and interpret unwanted image data into consideration when it comes to the training and implementation process of self-driving ability. To do this, the range of pixels on the Y-axis from 875 to 1150 was kept, and the rest of the image was removed, resulting in an image such as the following:

Figure 5. Cropped Image Example for A.I. Training/Implementation.

After this, the following image preprocessing step was to scale down the image. This allows for the image to be increased on the Y-axis, which in return makes the image look as though it is at a greater angle looking down. This is important because it allows the A.I. to interpret the road better since we are passing an image that makes it look longer than it is. To compensate for this, the image is shrunk roughly six times in the X-axis, making the Y-axis stretch appear more significant. This results in an image such as the following:

Figure 6. Resized Image Example for A.I. Training/Implementation.

After this, the edits consisted of physically changing the image itself. The first physical change was randomizing the image flipped left to right. This also meant that the steering values were flipped to correspond to the appropriately flipped image. This was done to diversify the images' dataset better and ensure that the model needed to train more on more than complete loop images. This made it so that the images were not in series while training and appeared randomly given to the A.I. model during building. Following the image resizing/random rotation, the image's color values were changed. This is done by converting the RGB (Red, Green, Blue) to YUV where:

- Y (luma): Represents the brightness level or intensity of the color.
- U (chroma): Represents the color information along the blue–yellow axis.
- V (chroma): Represents the color information along the red–cyan axis.

(OpenAI, 2023). This is done because A.I. models can better understand YUV versus the standard RGB image. After the conversion, the following image is created:

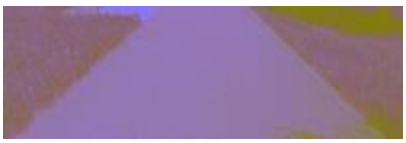


Figure 7. YUV Color Converted Image Example for Training/Implementation.

After this was completed, a Gaussian Blur was applied to the image, resulting in a noise reduction and smoothness compared to the freshly converted YUV image.

The last step of image preprocessing was to normalize the image; this is the process of "adjusting all the values measured in the different scales, in a common scale. In statistics, normalization is the method of rescaling data where we try to fit all the data points between the range of 0 to 1 so that the data points can become closer to each other" (Verma, 2021). This means we are taking the image's pixel values, which are 8-bits (0-255), and turning them into a floating value point from 0 to 1. This is done because the computer can compute the smaller range of values more efficiently than if the range was kept in 8 bits. To accomplish the following equation is performed:

$$x_{\text{normalized}} = \frac{x - \min(x)}{\max(x) - \min(x)} = \frac{x}{255}$$ (Verma, 2021)

The equation is simplified to the image values divided by 255 due to dealing with standard RGB images, which have a minimum value of zero and a max of 255 on each color layer. This means that even if the minimum/maximum values are absent in an image, all images are the same to prevent unintentional differences. This is done to each color layer, meaning that red, green, and blue get this process.

A deep learning model was made after the image preprocessing steps were completed to the image being inputted into the model training program. Specifically, a Convolutional Neural Network (CNN) model, this strategically chosen model type was based on its ability to recognize relationships and learn from the patterns the relationships hold and act upon them within the images used in the model's training. The input consists of image data from the camera being passed through the trained model, which will then perform feature extraction and relationship recognition. This was the center of the entire project since, with the understanding of relationships and patterns of images and numerical values, the car could properly make its own decisions to move in the desired way within a changing environment. CNNs are typically used in vision-based applications, making them the most robust choice for interpreting and making decisions based on the data received via the preprocessed images.

During the training of the model, the CNN network took the shape of the following image:

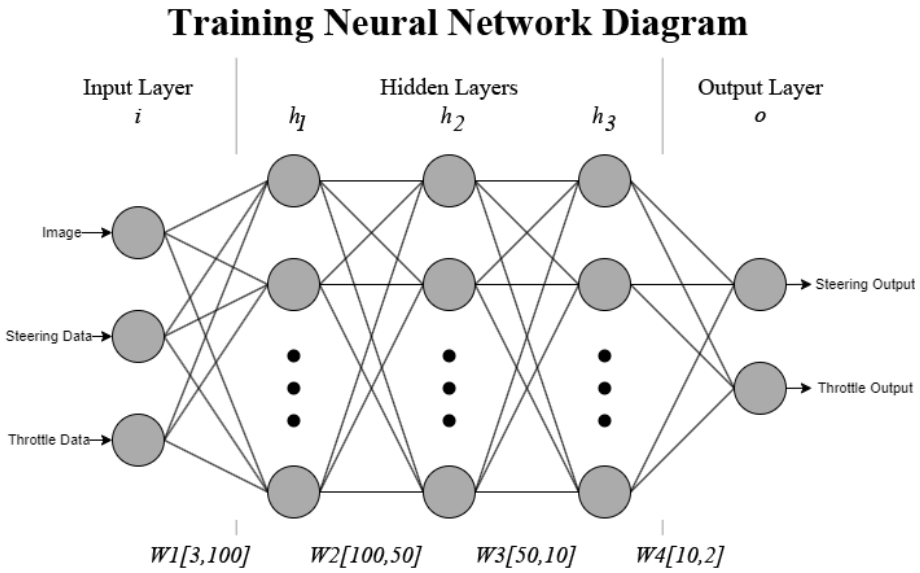


Figure 8. Training CNN Neural Network Diagram.

This shows how the model takes the image data along with the steering and throttle data and then webs them together to make a neural network capable of outputting the desired steering and throttle values. The two values are then outputted by the implementation neural network, which is shown below:

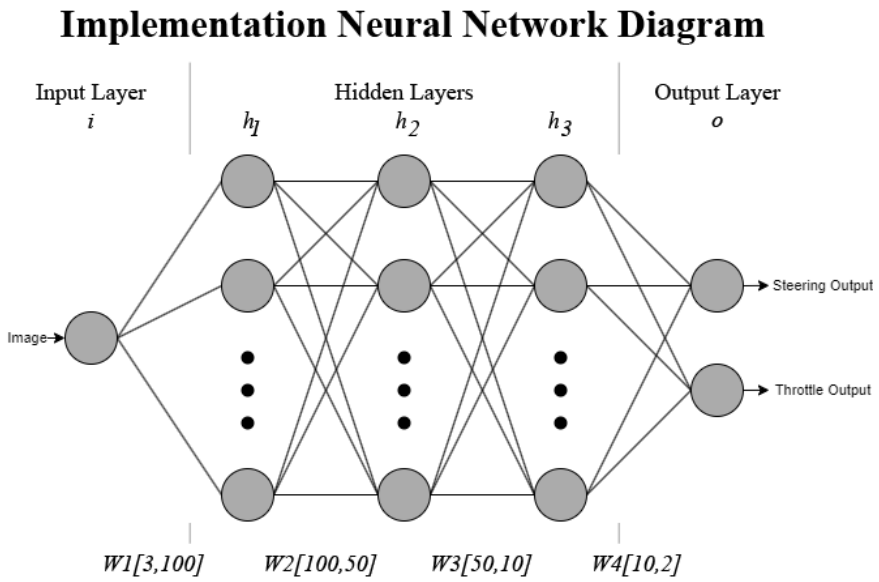


Figure 9. Implementation CNN Neural Network Diagram.

Here, with this model diagram, the image from the onboard camera is given to the network. In response, the model breaks down the relationships within and runs them throughout the neural network "web" to make a pattern-based decision on what the steering and throttle outputs should be. After the model sent the steering and throttle data via the neural network outputs, the desired values were fed to the control system, turning them into real-world motions on both the driving speed and steering angle.

Evaluation

This paper will not mention the testing process in depth due to the simplicity of what was performed to test the vehicles' ability. With that said, the way the vehicle was tested was as simple as taking it back to the original path the A.I. data was collected on and starting it in different positions to see how it would perform. This was done a dozen times before the battery for the motors died, meaning that enough data could be gathered from the single testing session to conclude the result of this project. This section will mention these results and then mention future enhancements that will be made to the second iteration of this self-driving model.

The results of the self-driving ability were more impressive than initially expected for the first attempt at something of this nature. The vehicle itself could stop when objects were correctly detected; this is discussed on page 22 and safely reactivated once the object was no longer detected. Along with this, the throttle values were, unfortunately, max throttle or close enough to be considered max throttle based on having little to no difference in speed. This is based on how the motors handled the small percentage change; the percentage of throttle variation resulted in roughly -5% of change from the maximum allowed throttle value. The steering, however, is the disappointing aspect of this project. Although the steering values were greatly diversified from the throttle data given to the model when training the neural network, the results for steering followed the throttle value issue. This resulted in the vehicle only managing to drive successfully on the straighter sections of the path, meaning that when it came to any turn in the path, the car would not correctly turn as intended.

Future Enhancements

The first enhancement that will be made to the second version of this self-driving car model will be to fix this issue with the steering. This will be done by incorporating a computer vision program to detect the edges of the path or road and find its natural curvature. If a curvature is detected, then the curvature will be used to determine the amount of steering needed to complete the turn. Along with this vision system, the vehicles' moving trajectory can also be predicted based on the steering value the car receives from the A.I. model, meaning that future additions, such as parking, would be possible. The addition of this computer vision system will also be paired with a much more diverse dataset, allowing for the A.I. model to be significantly improved in terms of the values it generates for both the throttle and steering.

The second enhancement is to upgrade the object detection sensor setup. In this version, the vehicle only has a singular sensor on the front and backside. This means the object must be in front of the vehicle to be detected correctly. This can be an issue in the case of an object coming at the vehicle from the side of the vehicle. To solve this problem, my goal is to incorporate a new sensor setup, which will make it so the vehicle has sensors around it entirely; with this, a new program will be developed that will be able to predict when braking is needed. This should allow for an even safer breaking feature, higher accuracy, and repeatability in performance. When writing this, the plan is to keep the original ultrasonic sensor as the object detection sensor but have a separate system running the detection prediction separately from the control system. This will also increase detection speeds as the only process running will be deciding whether braking is needed.

The evaluation section mentioned the third enhancement, upgrading the A.I. model itself. This will begin with adding more data to the dataset and making sure that the model architecture itself is in the most efficient form. This will be the focus of the second iteration of this project since the main center was gaining an understanding of A.I. models themselves. I also plan to optimize the model to perform more efficiently and quickly on the Jetson Nano. A look at changing the implementation code to C or C++ will also be taken to ensure that the second iteration of this project will run even closer to the objective of real-time performance.

Conclusion

This research project offers a comprehensive overview of designing, developing, and evaluating a Self-Driving AI-powered vehicle. It began by selecting and integrating hardware components, which enabled the transformation of a Power Wheels car into a platform capable of autonomous operation. Control and vision systems were the driving forces behind the project, regulating the communication and making decisions between the A.I. module and the physical components of the car. Integrating sensors and cameras and a control system demonstrated the combination of mechanical engineering, electrical engineering, and software development. The utilization of the Jetons Nano Developer Kit, which serves as the core of the A.I. system, and the accompanying Arduino Mega demonstrate the incorporation of embedded systems into autonomous vehicles.

At the heart of self-driving capabilities was the A.I., trained on a pre-trained dataset. The data preprocessing, which included image compression, image scaling, color rendering, and standardization, ensured that a robust dataset was prepared for efficient training. The machine learning model, the CNN, demonstrated the ability of A.I. to interpret a camera's visual data and convert it into usable data that the car can use to navigate in the real world.

The self-driving capabilities were evaluated to identify significant achievements and areas of improvement. The vehicle could detect and respond to obstacles accurately and consistently stop and resume motion when encountering them. However, there was a lack of precision in steering control, especially during turns, suggesting a need for further refinement in the Artificial Intelligence (AI) model. Looking forward, the outlined future improvements demonstrate a dedication to continual improvement. Including a computer-vision program to identify path edges and predict trajectories is intended to address steering issues and improve the vehicle's navigation features. Additionally, upgrading the object detection sensor setup to cover every aspect of the vehicle demonstrates a commitment to safety and comprehensive 360-degree awareness.

Developing the A.I. model remains a central objective for future iterations, intending to augment the dataset, optimize the model's architecture, and explore enhancements in real-time performance. As the

project acknowledges its limitations and outlines areas for growth, it becomes evident that this self-driving prototype signifies a final point and a fundamental step in an ongoing path of innovation.

In essence, this research project dove into the complexities of developing a self-driving vehicle, emphasizing the need for system collaboration, data preparation, and a commitment to continuous improvement. Although this project resulted in a functional starting prototype of a self-driving car, the room for improvement will forever rapidly grow in parallel with the continuous growth of A.I. technologies.

References

Adafruit. (2014, December 10). *4 Channel Level Shifter PCB*. Retrieved June 27, 2023, from GitHub: https://github.com/adafruit/4-Channel-Level-Shifter-PCB

Arduino S.r.l. (2020, September 29). *Arduino Mega 2560 Rev3 Datasheet.* Retrieved May 30, 2023, from Arduino Documentation: https://docs.arduino.cc/static/0b60b2282500b2cbd7b7c2b14611fe3b/A000067-datasheet.pdf

Arduino S.r.l. (2022, April 12). *Arduino Nano*. Retrieved from Arduino Documentation: https://docs.arduino.cc/static/6e28cbbb307b926ab9682f28113e7880/A000005-datasheet.pdf

Arduino S.r.l. (2023, May 16). *Blink Without Delay*. Retrieved from Arduino: https://docs.arduino.cc/built-in-examples/digital/BlinkWithoutDelay

Benchoff, B. (2013, February 4). *DIY Solder Stencils from Soda Cans*. Retrieved July 13, 2023, from Hackaday: https://hackaday.com/2013/02/04/diy-solder-stencils-from-soda-cans/

Boden, M. A. (1996). *Artificial Intelligence.* San Diego: Academic Press.

Chanzon. (n.d.). *Chanzon 5mm Orange LED Diode Data Sheet.* Retrieved from Amazon: https://www.amazon.com/gp/product/B01AUI4VWY/ref=ppx_yo_dt_b_search_asin_title?ie=UTF8&psc=1

Chollet, F. (2023, July 24). *The Sequential Model*. Retrieved August 3, 2023, from TensorFlow: https://www.tensorflow.org/guide/keras/sequential_model

Davies, D. W. (1999). The Bombe a Remarkable Logic Machine. *Cryptologia, 23*(2), 108-138. doi:10.1080/0161-119991887793

Dejan. (n.d.). *L298N Motor Driver - Arduino Interface, How It Works, Codes, Schematics*. Retrieved May 21, 2023, from How to Mechatronics: https://howtomechatronics.com/tutorials/arduino/arduino-dc-motor-control-tutorial-l298n-pwm-h-bridge/

Edgelec. (n.d.). *Edgelec 5mm Red LED (White Lens) Diode Data Sheet.* Retrieved from Amazon: https://www.amazon.com/gp/product/B077XB7NY2/ref=ppx_yo_dt_b_search_asin_title?ie=UTF8&th=1

Elec Freaks. (n.d.). *Ultrasonic Ranging Module HC-SR04.* Retrieved from Sparkfun: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

Error: ImportError: /usr/lib/aarch64-linux-gnu/libgomp.so.1: cannot allocate memory in static TLS block (i looked through available threads already). (2021, January 21). Retrieved August 9, 2023, from NVIDIA Developer Forums: https://forums.developer.nvidia.com/t/error-importerror-usr-lib-aarch64-linux-gnu-libgomp-so-1-cannot-allocate-memory-in-static-tls-block-i-looked-through-available-threads-already/166494

Fairchild Semiconductor. (2005, October). *BSS138 Datasheet (PDF) - Fairchild Semiconductor*. Retrieved June 26, 2023, from AllDataSheet: https://pdf1.alldatasheet.com/datasheet-pdf/view/50815/FAIRCHILD/BSS138.html

Fly Sky. (n.d.). *FS-iA6B.* Retrieved from FkySky RC: https://www.flysky-cn.com/ia6b-canshu

Fradkov, A. L. (2020). Early History of Machine Learning. *IFAC-PapersOnLine, 53*(2), 1385-1390. Retrieved June 23, 2023, from https://doi.org/10.1016/j.ifacol.2020.12.1888

Fredrick, A. (2021, August 31). *Getting Started with Image Preprocessing in Python.* Retrieved July 23, 2023, from Section.io: https://www.section.io/engineering-education/image-preprocessing-in-python/

HandsOn Technology. (n.d.). *BTS7960 High Current 43A H-Bridge Motor Driver*. Retrieved from HandsOn Tec: https://www.handsontec.com/dataspecs/module/BTS7960%20Motor%20Driver.pdf

HandsOn Technology. (n.d.). *L298N Dual H-Bridge Motor Driver*. Retrieved from HansOn Tec: http://www.handsontec.com/dataspecs/L298N%20Motor%20Driver.pdf

Hello AI World, choose Epochs and Batch-Size. (2022, October 27). Retrieved August 9, 2023, from NVIDIA Developer: https://forums.developer.nvidia.com/t/hello-ai-world-choose-epochs-and-batch-size/232008

How to solve OpenCV Green Screen. (2019, October 5). Retrieved July 3, 2023, from NVIDIA Developer Forums: https://forums.developer.nvidia.com/t/how-to-slove-opencv-green-screen/82803

Illegal Instruction (Core Dumped) Error on Jetson Nano. (2021, January 8). Retrieved August 9, 2023, from Stack Overflow: https://stackoverflow.com/questions/65631801/illegal-instructioncore-dumped-error-on-jetson-nano

Intel Corporation. (2016, September 3). *OpenCV - Python*. Retrieved July 2, 2023, from GitHub: https://github.com/opencv/opencv-python

Intel Corporation. (n.d.). *Install OpenCV-Python in Ubuntu*. Retrieved July 2, 2023, from OpenCV: https://docs.opencv.org/3.4/d2/de6/tutorial_py_setup_in_ubuntu.html

Intel Corporation. (n.d.). *Operations on Arrays.* Retrieved July 24, 2023, from OpenCV: https://docs.opencv.org/3.4/d2/de8/group__core__array.html

Jetson Nano as I2C Slave. (2020, June 2). Retrieved July 20, 2023, from NVIDIA Developer Forums: https://forums.developer.nvidia.com/t/jetson-nano-as-i2c-slave/125937

JST Sales America Inc. (2010). *XH Connector Datasheet.* Retrieved July 14, 2023, from Digikey: https://www.digikey.com/en/resources/datasheets/jst/xh-connector

Kalinsky, D., & Kalinsky, R. (2001). Introduction to I2C. *Embedded Systems Programming, 14(8)*, 1101-1105. Retrieved June 23, 2023

kangalow. (2023, February 7). *jtop: The Ultimate Tool for Monitoring NVIDIA Jetson Devices.* Retrieved August 9, 2023, from JetsonHacks: https://jetsonhacks.com/2023/02/07/jtop-the-ultimate-tool-for-monitoring-nvidia-jetson-devices/

Kashif. (2022, September). *Serial UART Communication between Two Arduino Boards*. Retrieved from linuxhint: https://linuxhint.com/serial-uart-communication-between-two-arduino/

Kurniawan, D. (2022, May 13). *Several Types of Artificial Neural Networks Architecture that You Should Know*. Retrieved August 15, 2023, from Medium: https://medium.com/mlearning-ai/several-types-of-artificial-neural-networks-architecture-that-you-should-know-c169a5e22ec7#:~:text=Each%20architecture%20of%20the%20neural,realistic%20and%20high%2Dresolution%20images.

Laddha, N. R., & Thakare, A. P. (2013, January). A Review on Serial Communication by UART. *International Journal of Advanced Research in Computer Science and Software Engineering, 3*(1), 366-369. Retrieved June 23, 2023

Mankar, J., Darode, C., Trivedi, K., Kanoje, M., & Shahare, P. (2014, January). Review of I2C Protocol. *International Journal of Research in Advent Technology, 2*(1), 474-479.

murtazahassan. (2020, October 15). *Neural Networks Self Driving Car Raspberry Pi.* Retrieved August 1, 2023, from GitHub: https://github.com/murtazahassan/Neural-Networks-Self-Driving-Car-Raspberry-Pi/tree/main

NumPy Developers. (2023, July 8). *Input and output.* Retrieved from Numpy API Reference: https://numpy.org/doc/stable/reference/routines.io.html

NVIDIA. (2020, December 18). *Index of/compute/redist/jp/v45/tensorflow.* Retrieved August 8, 2023, from NVIDIA Developer Download: https://developer.download.nvidia.com/compute/redist/jp/v45/tensorflow/

NVIDIA. (2023, April 25). *Initial Setup Headless Mode.* Retrieved July 27, 2023, from NVIDIA Developer: https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit#setup-headless

NVIDIA. (2023, August 3). *Installing TensorFlow for Jeston Platform.* Retrieved August 8, 2023, from NVIDIA Docs Hub: https://docs.nvidia.com/deeplearning/frameworks/install-tf-jetson-platform/index.html

OpenAI. (2023). *ChatGPT [Large language model]*. Retrieved from OpenAI: https://chat.openai.com

PCB Trace Width Calculator. (n.d.). Retrieved from DigiKey: https://www.digikey.com/en/resources/conversion-calculators/conversion-calculator-pcb-trace-width

Roscoe, W., & Conway, A. (n.d.). *About Donkey*. Retrieved June 29, 2023, from Donkey Car: http://docs.donkeycar.com/

Rusk, N. (2016, January). Deep Learning. *Nature Methods, 13*(1), 35. Retrieved July 1, 2023, from https://doi.org/10.1038/nmeth.3707

Sagar, A. (2019, August 28). *How to Train Your Self Driving Car Using Deep Learning.* Retrieved August 1, 2023, from Towards Data Science: https://towardsdatascience.com/how-to-train-your-self-driving-car-using-deep-learning-ce8ff76119cb

Singh, A. (2019, September 19). *9 Powerful Tips and Tricks for Working with Image Data using skimage in Python.* Retrieved July 31, 2023, from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2019/09/9-powerful-tricks-for-working-image-data-skimage-python/

Singh, S. (2023, February 20). *Importance of Pre-Processing in Machine Learning*. Retrieved August 14, 2023, from KDnuggets: https://www.kdnuggets.com/2023/02/importance-preprocessing-machine-learning.html

Sparkfun. (2020, January 20). *NVIDIA Jetson Nano Developer Kit Data Sheet.* Retrieved May 6, 2023, from Sparkfun: https://cdn.sparkfun.com/assets/0/7/f/9/d/jetson-nano-devkit-datasheet-updates-us-v3.pdf

Susumu International U.S.A. (n.d.). *RR1220P-103-D Data Sheet.* Retrieved June 26, 2023, from Susumu Category: https://www.susumu.co.jp/common/pdf/n_catalog_partition05_en.pdf

TensorFlow. (2023, 07 06). *tf.test.is_gpu_available*. Retrieved August 9, 2023, from TensorFlow API Docs: https://www.tensorflow.org/api_docs/python/tf/test/is_gpu_available

The National Museum of Computing. (n.d.). *The Turing-Welchman Bombe*. Retrieved May 4, 2023, from https://www.tnmoc.org/bombe

Tsuge, B. (2022a, January 21). *Reading Values from an RC Receiver using Arduino*. Retrieved May 23, 2023, from The Bored Robot: https://theboredrobot.com/blogs/blog/reading-values-from-an-rc-receiver-using-arduino

Tsuge, B. (2022b, May 15). *Simultaneously Reading Two PWM Signals from an RC Receiver with Arduino*. Retrieved May 24, 2023, from The Bored Robot:

https://theboredrobot.com/blogs/blog/simultaneously-reading-two-pwm-signals-from-an-rc-receiver-with-arduino

Turing, A. (1950, October). Computing Machinery and Intelligence. *The Essential Turing, LIX*(236), 433-460. Retrieved May 4, 2023, from https://doi.org/10.1093/mind/LIX.236.433

Verma, Y. (2021, August 29). *Why Data Scaling is important in Machine Learning & How to effectively do it*. Retrieved August 14, 2023, from Analytics India Magazine: https://analyticsindiamag.com/why-data-scaling-is-important-in-machine-learning-how-to-effectively-do-it/

Wikimedia Foundation. (2023, May 4). *I2C*. Retrieved June 22, 2023, from Wikipedia: https://en.wikipedia.org/wiki/I%C2%B2C

Wikimedia Foundation. (2023, Jun 10). *Serial Peripheral Interface*. Retrieved June 23, 2023, from Wikipedia: https://en.wikipedia.org/w/index.php?title=Serial_Peripheral_Interface&action=history

Wikimedia Foundation. (2023, June 11). *Universal Asynchronous Receiver-Transmitter*. Retrieved June 23, 2023, from Wikipedia: https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter

Zambetti, N. (2023, June 26). *A Guide To Arduino & the I2C Protocol (Two Wire)*. Retrieved from Arduino Documentation: https://docs.arduino.cc/learn/communication/wire

Engineering Journal

May 4, 2023 – Today I sat down and started this report by setting up the structure. I also completed the Objective section and started the Introduction section with the beginning of a running reference page.

May 6, 2023 – Today I sat down and picked the parts to use for the project. The chosen parts are NVIDIA Jetson Nano for the A.I. operations and an Arduino Uno for the indicator and automatic braking system. For the automatic braking system, two HC-SR04 ultrasonic sensors were chosen; one in the front and back. CAT-6 ethernet cable was chosen for the wiring due to its neatness for cable management. I also ordered orange clear-lens LEDs for the turn signals.

May 7, 2023 – Today I received the Power Wheels car donated by Jake Holbrook.

May 8, 2023 – Today I disassembled and thoroughly cleaned the Power Wheels car. To thoroughly clean the Power Wheels, I cut the wires going to the back motors and wrapped the motors in plastic. I then used a power washer to clean away any dirt. Lastly, I cut the plastic away where the turn signal stickers were to mount the custom functional turn signals.

May 9, 2023 – Today I designed and manufactured the turn signal prototype using FDM and DLP 3D printers. Mounting the turn signal was not a focus but instead the housing for the LEDs and the clear lens. At first, the tolerance was too big, so I reprinted the parts with half the 0.25mm tolerance used before.

May 10, 2023 – Today I reassembled the Power Wheels car and removed all the decal stickers. I also removed the decorative front winch and removed the water hose from the car.

May 11, 2023 – Today I test-ran the camera ribbon cable in the car to ensure I had enough length. I also ran the CAT-6 cable to the front of the car for the safety light, turn signals, and HC-SR04 sensor. I then designed the wiring diagram to better visualize how the connections will be made throughout the car.

May 12, 2023 – Today I designed and manufactured the mount method for the turn signal. This was done by modifying the top piece of the housing. At first, the mounting standoffs were too long causing the turn signal to mount too low. After fixing this the turn signal sat at a weird angle toward the ground so the mounting method was changed. After changing the mounting method, the turn signal's final prototype was produced, and the final prototype of the turn signal was mounted. The finalized version has not been printed yet. I also added the Author's Note to the title page.

May 13, 2023 – Today I designed and manufactured the housing and lens for the back indicators. The housing had printed FDM on the Bambu X1-Carbon and the lens was resin printed on the Photon. When resin printing the lens, I tried to print both at one time, only one fully printed and was warped due to supports failing to adhere to previous layers.

May 14, 2023 – Today I got the back lens to finally print correctly. This was done by removing the magnetic flex plate on the build plate, causing adhesive issues. I also wired JST XH connectors on the indicators for them to be easily removable from the custom wiring harnesses. I then made the back wiring harness by changing how the ethernet cable is braided so that the proper cables are running to the proper places; these had the appropriate male JST connectors.

May 15, 2023 – Today I printed the final version of the housings for all indicators in Bambu Lab's black PLA-CF filament and transferred the LEDs into the final housings and remounted them onto the Power Wheels car body. I also completed the front wiring harness like the process done on the back harness.

May 16, 2023 – Today I coded and tested the braking system code on the Tinker CAD website using their circuit builders and an Arduino UNO. I first started by setting the HC-SR04 sensing up then implemented the auto-braking feature. Then I added the input signal reading from the other systems and finished with the turn signal functionality. The first time the code did not function properly, this was due to the turn signals not being able to be properly turned on/off. To fix this I simply restructured the way the code was to have fewer voids and completely recoded the turn signal functionality and was able to get it to work properly.

May 17, 2023 – Today I made the braking systems control board. For this, I decided to go with a clone of the Arduino Nano due to the lower operating voltage as the originally planned to use Arduino Uno. To make the circuit board I simply used the appropriate JST XH female connectors, female header pins for the microcontroller integration, and a prototyping board. After finishing that I designed and manufactured the front and back HC-SR04 housings.

May 18, 2023 – Today I made modifications to the sensor housings. The first change was to the back housing, this was simply making the mounting holes wider as they didn't properly line up with the preexisting holes. The second modification was made to both the front and back housing; this was making the fit tighter for the HC-SR04, I then also made it, so the barrels of the ultrasonic sensor didn't stick out as much from the housing. After successfully prototyping the housings, I printed them in the PLA-CF filament for the finalized version. I also rewrote the original introduction into the now-beginning Theoretical Background paragraph and rewrote the objective section to add length and more in-depth explanations. I also added the Jetson Nano and Arduino Nano data sheets to Appendix A.

May 19, 2023 – Today I added the LED data sheets along with the HC-SR04 modules data sheets. I also modified the Power Wheels bumper for the cable to run properly to the HC-SR04, after this all the indicators and sensor housings were mounted to the vehicle. I then designed and manufactured the safety light insert to go inside the original fake siren light; wiring was not done for this. Testing of the braking systems code was then done. The findings were that the turn signals, sensors, and object detection functioned properly during testing, but two issues were found. The first is that the braking can "flicker" meaning that it does not have a hold period to ensure the objects have fully moved out of the path when it disengages the sensor's safety zone. The second is considering future processes, this is that the back sensor could make the vehicle brake when driving forward when something gets in the safety zone and vice versa for reverse. This is an issue as only the safety zone in the direction of the path of the vehicle is needed for proper functionality.

May 20, 2023 – Today I sat down and added an implementation to the two issues found yesterday during the testing. To have the hold period, I simply made it so that instead of setting the braking state straight back to false, it waits for 5 seconds to pass and then goes to false. While coding this I found a few repeating chunks of code and decided to go through and straighten up the code by adding them as functions to the methods they were originally in. I also replaced the simpler if-else statements with conditional operators. This also allowed for fewer global variables since static variables could be held in the functions and methods.

May 21, 2023 – Today I tested version 2.2 of the code for the braking system without the communication signals connected. During testing, the system was detecting false input signals causing the turn signals and brakes to be activated randomly. I then connected a GND signal to all the input signals and tested the system and found this prevents them from happening. I then looked at the systems control board for any type of sort circuits and did not find anything. The false positives did show that the braking via the communication inputs did not need the 5-second wait after activation due to the brakes being used with a type of vision involved.

May 22, 2023 – Today I updated the code for the braking system to include the bypass of the 5-seconds for the brakes. I accomplished this by having a separate function for the brakes and having the code that writes the brake lights to off check if bit 3 of the communication inputs is on and if so then it skips writing the brakes off. I then also started to research how to use an Arduino with an RC receiver to make the Power Wheels remote controllable with an L298N motor driving module.

May 23, 2023 – Today I coded and tested a program I found on the website *The Bored Robot* to read the RC receiver signal with an Arduino Uno using the ”pulseIn()” advanced I/O feature. When testing, the range of data I received was from 993 through 1992, with the range being from 1486 through 1493 when the stick was centered on the transmitter. This is a good first step, but some tinkering will need to be completed to get the proper range of 0 through 255 for the PWM to the L298N motor driver. I also discovered that the “pulseIn()” temporarily blocks the program meaning other code cannot run alongside it making a newer method to be needed.

May 24, 2023 - Today I coded a second test code on the Arduino Uno for reading the RC receiver signal but assigned the value of the signal to ranges of 0 through 255 using the "map()" feature. I also coded this new test with a new method using external interrupts instead of the "pulseIn()" feature found on *The Bored Robot* website. It was also found that the number of needed GPIO pins is 13 allowing for the Arduino Uno, Nano, or Micro as options for the microcontroller. I also decided to see the time high for the signal coming from the RC receiver and found that the range of time is 0.7 through 1.7 microseconds, with the stick centered being at 1.2 microseconds. The signal at low is 0V (GND) with the high peaks being 3.2V.

May 25, 2023 – Today I combined the RC test code with code to incorporate the L928N motor driver using a 5V DC motor with a fan attached to visualize the motor's spin rotation. To do this I looked at the "pulse_width" and saw if it was above or below the value of the stick centered. Then I made it set the H-bridge depending on this check through the related "IN" pins.

May 27, 2023 – Today I took the L298N integrated test code and applied it to the left motor on the actual Power Wheels car instead of the 5V DC motor. This worked as intended and presented no issues, so the code was updated to integrate the right motor as well. This was done by adding a function for setting the h-bridge to forward/reverse and then setting speed by using "analogWrite()" to send the same PWM signal.

May 28, 2023 – Today I modified the code from yesterday's test to incorporate the right wheel as well. To do this I took repeated code such as writing directions for the motors and then made it into functions for use inside the methods while applying the same code to the left motor inputs. Upon testing, this was found to be successful, and the RC controls for channel 2 worked as intended. I also began looking at the steering method and decided to purchase a high-torque servo to use for steering over the original pick of a linear actuator due to the overall cost.

May 29, 2023 – Today I took a break from coding and created the drawing files for the brake, turn, and safety indicators and added them under Appendix B of this paper. Only the manufactured parts were shown in the drawing files. I also purchased a rechargeable 12V 9AH battery to power the drive motors. I also ran and placed connectors on the end of the 14 AWG wire to power the L298N, I then used a buck converter to boost the 5V to 9V to power the Arduino Uno with the test code.

May 30, 2023 – Today I decided that combining the RC Control system with the Indicator/Safety system would be best to cut down on the number of power supplies needed. In doing this the system can be cut down to an Arduino Mega 2560 and the Jetson Nano. I also decided to add a motor control feature to this all-in-one type of system so the Jetson Nano will only have to run the A.I. and output bits of data to the Arduino; to do this I will add a 3-position switch to toggle "RC Mode" and "A.I. Mode". I also ordered an authentic Arduino Mega 2560 Rev 3 for the system to ensure its quality and functionality are at the highest for the all-in own system.

May 31, 2023 – Today I started to code the all-in-one system. The approach taken was to have a main program that controls library calls based on the position of the "Mode" switch. This way the functionalities (such as automatic braking, RC functionality, etc.) are separated and not cluttering up the main program like before. I ended up coding the automatic braking system, motor control, and RC decoding but only made the RC decoding and motor control work to test the idea of having a motor control feature that can be shared.

June 1, 2023 – Today I tested the code first with a power supply regulated at 12V 5A and found that the code worked as intended. I then tested the car driving with the purchased battery and found that sometimes the motors would not move when trying to drive forwards or reverse. This is believed to be caused by the battery having a max initial current of 2.7A while the motors drew 5A on spin-up. To prevent this overload, I added a small delay between starting the motors, which seems to have fixed the issue; the back of the power wheels was hoisted in the air for this test.

June 2, 2023 – Today I tested driving the car on the ground and found that the motors would first function fine but over time they would stop working as intended. When this happened, the left wheel would only spin in the forward direction and the right wheel would only spin in the reverse direction. Still thinking this was due to the step-up current maxing the battery's output amps, I decided to solder a 10uF capacitor to each motor output to help. Doing this helped the motors spin up faster without drawing as many amps as before but the issue of the motors randomly stopping the normal functionality is still present; The issue is believed to be related to the delay in the motor startups or some other code-related issue.

June 3, 2023 – Today I coded the turn/brake indicators into the RC functionality code. This was done by making the class “Indicator.h” and setting up methods and functions that will be able to be shared with both operation modes. I also removed the delay in the motor control code and found that the motor issue still occurred meaning the source of the issue was not the small delay.

June 4, 2023 – Today I went ahead and took a break from the motor issue to hopefully come back tomorrow with a fresh view of the issue. While doing this, I installed the key on/off switch in the spot the fake ignition key was on the Power Wheels. I also mounted and wired a waterproof cigarette lighter socket at the location of the fake gas cap on the back of the Power Wheels to recharge the battery under the hood.

June 5, 2023 – Today I started back at debugging the issue with the motors and found some possible issues. My first thought when debugging was that the Arduino Mega 2560 was not properly sending the data to the L298N, so I decided to hook LEDs to the logical inputs and connected the enable (PWM) pins to my oscilloscope to see the analog data being sent. This test showed that the Arduino Mega 2560 was indeed sending the data properly and that the issue was not related to this. The second test was hooking the oscilloscope up to the output side of the added capacitors to see the motor’s voltage input. This showed that when only one motor spun (left in forwards, right in reverse) that the charge and discharge of the capacitors were differently shaped; it also showed that there was no attempt to charge or discharge the capacitor for the stopped motor. Shortly after this discovery, after unhooking the oscilloscope probes a concerning hot smell started to come from the L298N making me think that the L298N may be bad. To test this, I allowed it to cool and then tested the motors with only the Arduino connected to the L298N and test equipment. As half predicted the motors functioned properly until the L298N chip got hot on the module then the occurring issue of only one motor working resurfaced. At this point, the belief of the motor issue is due to the L298N module. To try and fix this I decided to replace the L298N module with two BTS7960 43A motor controllers with one attached to each wheel since they are single H-Bridge controllers. With this replacement, the Arduino Mega will now need to be attached to the Jetson Nano power supply system, while only the DC motors are connected to the current-in-place power system.

June 6, 2023 – Today I went through the code for the control system and added comments to variables and comments on functions/methods without them. I also made the coding more visually appealing for any sort of future debugging. I went ahead and removed the L298N motor control as well in preparation for the BTS7960 Modules arriving tomorrow.

June 7, 2023 – Today I hooked up the two BTS7960 motor controllers to the motors and Arduino Mega 2560. I also had to use 14AWG wire splicers to power the second BTS7960 motor controller. After wiring the two motor controllers up I looked through the datasheets example code and programmed the Arduino with a simple program to spin the motors forwards and then in reverse. The motors spin up almost at full speed instantly with no issue signifying that the DC motors were maxing out the amperage of the L298N motor controller. When trying to measure the amperage drawn from the new motor system my digital multi-meter, the probes were not able to carry more than 3A of current making the motors spin slower than when the DMM was disconnected. This proves that the two BTS7960 are allowing the motors to draw more amperage than the L298N due to the principle that the DMM did not disturb the 2.5A reading I was getting with the L298N or the DC motor's speed.

June 8, 2023 – Today I took yesterday's very basic example code and replaced the old L298N code with it. The enable pins for forward and reverse for both controller modules can be connected to 5V the entire time with just the forward and reverse PWM pins being connected to the Arduino Mega 2560; meaning that only four connections (two each) are needed to connect the motor controllers to the Arduino Mega 2560 not counting the connection of GND and 5V from the Arduino. After the motors' functioned properly with the RC controls I found that there was a small false pulse of reverse data occasionally which I fixed by updating the range for the sticks centered; this was most likely caused by changing from a clone Arduino Mega 2560 to an authentic one. I went ahead and tested the RC functionality of the Power Wheels on a blacktop driveway and found no issues with the motors spinning up when powered by the originally purchased 12V 9Ah battery. During the testing, I found that the BTS7960 motor controllers have a built-in braking feature for when the PWM input signal is "0" meaning that the car stops in under a foot or so meaning the auto braking should be very responsive.

June 9, 2023 – Today I designed and prototyped the servo mount to start working on adding the steering abilities to the RC functionality. This included purchasing the required mounting hardware to mount the servo to the mount and then the mount to the car. I also went ahead and made the bill of materials spreadsheet for this project as very few additional parts will be needed from here on out.

June 10, 2023 – Today I designed and printed the mounting jig for the servo mount. I also went through the documentation and updated all datasheets to include the first page and any material that describes the usage of the component. In doing so I also started using the Word source manager to auto-update the reference page and any in-text citations made. I then started designing the control system PCB in Eagle CAD by creating the schematic for the Arduino Mega 2560. After this, I created the PCB outline and then played around with the position of the components where I wanted them.

June 11, 2023 – Today I designed the rest of revision A for the control system PCB and placed the production order through JLCPCB. The PCB is estimated to arrive in 10 - 16 days unfortunately but this will allow enough time to finish the rest of the control system and any sort of polishing to the RC functionality. The PCB has JST XH connectors to provide easy connection to the control system, alongside LEDs for the bit outputs and power status.

June 12, 2023 – Today I mounted the servo to the power wheels car and assembled the steering assembly for the servo to control the front tires' direction. I also then started to code the library to make the servo steer the wheels based on the data received from channel 1 of the RC receiver. At first, I tried to code my own using only one delay in microseconds, but the servo did not function as intended, so I decided to use the Arduino open-source servo library. When trying to power the servo via the Arduino Mega 2560's 5V source, the servo could not move the steering assembly. At this point, I decided to stop testing the servo steering functionality and resume later.

June 14, 2023 – Today I tested the servo steering functionality in various ways. The first thing I did was connect the servo to an actual GND pin of the Arduino Mega 2560 instead of the mini breadboard I am using temporarily to supply GND to various things due to the lack of them on the Arduino itself. This functioned when the connecting bar from the servo to the steering rod was disconnected but it could not provide enough torque when it was connected. This gave me the idea to instead provide the servo with 6.8V, its max rated voltage for max rated torque, via a benchtop power supply to test and see if this improved the operating torque any. After doing this the steering functionality worked as intended but the range of the servo needed to be adjusted to stop the servo from going further than the tires could turn. To do this I connected the Arduino Mega 2560 to my computer and watched the serial plotter as I turned the steering left and right via the RC controller, this gave me a range of 115 through 91 for the left turn, 90 for the centered stick, and 89 through 55 for right turns. Although this was tested with the wheel up off the ground, it was 100% successful after providing 6.8V to the servo. However, this means that a new PCB will need to be ordered to change the servo from being powered by the Arduino Mega 2560 5V regulator to a buck converter that's stepping 12V down to 6.8V while connecting the grounds of the two. I decided to wait on updating and sending revision B for production in case of any future changes needing to be made and have decided to just use the JST connector plug without the VCC wire connected.

June 15, 2023 – Today I tested the steering with the wheels on the ground, added Appendix C: PCB Design, and finished the theoretical background portion of this project report. Before I was able to test this, I planned on hooking the control system and servo up to separate 9V batteries with buck step-down converters to convert the 9V to the needed 7.5V and 6.8V but realized that I only had step-up converters. I then decided to go ahead and order a second 12V 9Ah battery while ordering the buck step-down converters. While testing I found that the steering worked well when steering to the right but the angle to the left seemed like too little. Tomorrow, I plan to troubleshoot this in-depth to find a solution to this issue.

June 16, 2023 – Today I measured the amperage of the servo to start preparing for the design of the control system REV B PCB. I also 3D printed some Velcro cable holders I found on printables.com designed by @MartinS_196727 and mounted them in the power wheels car to organize and hold the cables. I then tested to find a root cause for the issue with the left steering. The first thing I attempted to solve the issue with was changing the angle the servo motor is limited to for left turns from the original 120 degrees to 140. This seemed to improve the angle of steering to the left, but I felt as if the angle could still be improved upon, so I decided to see if making the plastic I attached to the servo mount more rigid would prevent any unwanted play in the steering. Doing this presented no change in the left-hand steering and while observing I realized the steering rod was limiting out due to the design of the Power Wheel's steering. I also completed the "Design and Development of Hardware Components" section of the "Methods and Procedures" due to the hardware systems almost being completed.

June 17, 2023 – Today I installed the second 12V battery and wired two steps down converters to convert the 12V to 6.8V and 7.5V needed for the servo and the Arduino Mega. I did not wire the 12V to 5V converter yet due to not being completely positive about where the Jetson Nano will be mounted. With the installation of these, I was able to fully drive the car via remote control. When driving the car around the indicators worked as intended and the motors showed no issue with power. However, the steering seemed weak and would slightly turn to the desired direction then after driving for a few seconds would finally go to the desired angle. This is believed to be a servo-based issue but tomorrow I will test an idea before ordering a higher torque servo. I also Velcro strapped the wires into place for organization, and the computing area's overall appearance looks better.

June 18, 2023 – Today I tested to improve the steering ability of the Power Wheels. My first thought was to check the joints where the tires rotate to see if there was any undesired friction and found that this was not the case and that the tires moved smoothly. After this I tried slowing the servo down in the code to try and allow the servo to have more time with the applied torque, this showed no improvement and interfered with the response of the system due to the small delays needed to attempt this. In the end, I decided to go ahead and order an 80 kg torque servo.

June 19, 2023 – Today I received a 1-channel 12V relay module to use to power on and off the computing systems power source with the motors power source key switch. I did this to keep the key switch as the master on/off switch for the entire system, the 12V source from the motor triggers the relay which closes the loop for the computing power source. I also began researching what data the Jetson Nano will need to be trained for the self-driving feature; from what was found data such as motor direction and speed, along with the servo angle position and the video camera footage are the key source of information. I also decided that the Jetson Nano should not need access to the HC-SR04 sensors due to the ability to enable the auto-braking system even when in the A.I. mode. I also began brainstorming on how to connect the second battery to the already installed charger port, the idea is to use a two-position switch (the same for selecting A.I. and RC mode) to select which battery is getting charged.

June 20, 2023 – Today I received the new 80 kg steering servo in the mail. This allowed me to design, prototype, and print the final carbon fiber version of the servo mount. Due to the much greater size, the preexisting mounting holes were no longer usable, but this was the only modification that had to be made to accommodate the new bigger torque servo. After installation, I upped the voltage from the servo power step-down buck converter from the previous 6.8V to the newly suggested 7.4V. After this, I wired the stepper motor to the control system, powered it on, and tested it. The mapping position values did need to be changed to accommodate the longer servo horn but other than that easy change the servo installation and integration went smoothly. The tires now move like a hot knife through butter compared to the old servo. Tomorrow, I plan to test drive the car outside.

June 22, 2023 – Today I researched different ways to communicate between the control system and the Jetson Nano. The I2C method of communication seems like it could be the best option but due to the lack of knowledge on I2C communication, let alone communicating between two different type systems, I feel as using regular 8-bit binary communication with a single wire for each bit may be easier at this time with possible upgrades with later features. Although this has not been officially decided on, if the 8-bit method was to be used then 4-bits can be used for the motor speed values using 0-10 to translate into 0-100% speed percentage. The other 4 bits will be used for steering, the steering range moved total is 65 degrees meaning that if you divide 65 degrees by 5 then 13 multiplied by 5 equals 65. This means that 0-13, with a multiplication of 5 after receiving the data, can translate into the desired steering angle since the minimum and max range of the stepper motor is permanently set. An additional wire will be needed to give a direction signal to the control system to know which direction to spin the motors. I also test-drove the car via the RC functionality today and had promising results. The only potential issue is sometimes the servo is slow or delayed getting to the position desired, but it is not catastrophic so until it presents a problem the code will stay the same.

June 23, 2023 – Today I decided to further research communication protocols to use to communicate between the Arduino Mega 2560 Rev 3 and the Jetson Nano. The three main protocols I looked at were the previously mentioned I2C, SPI, and UART. Based on the research I feel that UART would be the best-starting ground as it uses the preexisting “Serial” features by writing packets of bytes. It also only requires two wires between the two devices while maintaining the ability to send two bytes and a single bit; this means that the actual values of the steering angle and motor speed can be sent at once. Today I also split what was the second paragraph of the *Theoretical Background* chapter, which mentioned Machine Learning and Machine Vision, into two separate paragraphs by adding the history of machine learning. I also added more datasheets to Appendix A.

June 24, 2023 – Today I created the drawing file for the latest servo mount and added it to Appendix B. I also flashed the OS onto the Jetson Nano board and installed the required software for Secure Shell and Remote Desktop Connection; this will be used to remote into the Jetson Nano from any computer. I also created a test for the communication of the control system and Jetson Nano by using two Arduino Uno microcontrollers in a simulator. To get an understanding of how UART code worked on Arduino I decided to use preexisting code at first as a test, the code I used was written by Kashif and found on LinuxHint. Shortly after testing the UART communication protocol, I realized the true importance of synchronous protocols, due to UART being asynchronous the order of the bits would slowly start to get shifted after a few seconds of sending data. This would not be an issue if it was not for trying to send two bytes with a single bit of data. However, A longer delay fixed this, but since I want the communication to be as real-time as possible, a different communication protocol must be used. Although using the preexisting "Serial" features was easy to get it set up there needs to be a clock to ensure the byte is constructed with the bits in the order they were sent. To fix the issue I have decided to go with the I2C communication protocol as it allows for the two-wire connection while being able to communicate with more devices while maintaining the same two wires if later features are added.

June 25, 2023 – Today I researched and purchased single-channel optocouplers. These will be used to allow for the proper voltage needed for the GPIO pins for communication between the Jetson Nano and the Arduino Mega; this is used instead of regulating the Arduino Mega GPIO pins voltage to 3.3V to the Jetson Nano and the Jetson Nano GPIO pins voltage to 5V for the Arduino Mega. The optocouplers will also allow for the power connectivity between the two microcontrollers to be separated in case any shorts were to happen. After picking the PC817C optocouplers, I designed and tested a single optocoupler circuit, a single optocoupler connecting two Arduino Uno, and a four optocoupler circuit. The goal is to use two optocouplers, one with the input being the Arduino Mega and one as the Jetson Nano, for each way of communication so that each signal can go to and from. Further testing on this will be done tomorrow using two Arduinos in a simulation.

June 26, 2023 – Today I sat down and tried to configure a circuit that would allow optocouplers to perform bidirectional communication between the control system and the Jetson Nano via I2C communication. This was unsuccessful due to the optocoupler having too slow of a response time meaning that the second microcontroller was not able to get the proper data signals needed for communication. The method of testing was using TinkerCAD's circuit builder with code from an example found on Arduino Documentation by Nicholas Zambetti on a guide for using the I2C library for Arduino called Wire. To fix this I decided to turn back to the originally avoided voltage conversions between the two microcontrollers. This will be done between a bidirectional level shifter which will convert 5V to 3.3V and 3.3V to 5V. Originally, I was going to purchase a four-channel module from Adafruit but after the cost of the module plus shipping, I was able to order the parts from Digi-Key needed for the entire control system shield. Using Adafruit's published schematic for the 4-channel level shifter, I will now integrate two channels of level shifting (SDA and SCL) into the shield itself by using two BSS138 MOSFET for the SDA and SCL signals between the microcontrollers. The Adafruit schematic was added to Appendix C along with the BSS138 MOSFET and the chosen 10 kilo-ohm resistors datasheet to Appendix A.

June 27, 2023 - Today I redesigned the control systems schematic and PCB to incorporate the I2C logic level shifter based on the previously mentioned Adafruit 4-channel logic level shifter. I also included connections for the added power inputs for the Arduino Mega 2560 and the servo so that everything is contained on the shield itself. Along with this I decided to redo the layout of the JST connectors and made sure to label them and place them in sections that are labeled for appearance and ease of use. The JST connectors connecting the motors to the shield were also changed from B6B-XH connectors to B4B-XH connectors since three of the previous pins were connected to VCC, to fix this I will CNC two PCBs to fit on the header pins on the motor controllers to connect the enable pins to VCC. I also went ahead and ordered the PCBs through JLCPCB.

June 28, 2023 – Today I renamed Appendix C to "PCB Design", that way I could add the images of the PCB and the math used for trace width calculations without having to add a new appendix. The next thing I did was go through and make the solved equations for the PCB width calculations. Afterward, I added the new schematic alongside the PCB board images, and the PCB trace calculations used for designing. I also added resources used yesterday to the reference page. Along with this, I began preparing the Jetson Nano for the A.I. part of this project. This involved starting a game plan for training the A.I. and updating everything on the Jetson Nano.

June 29. 2023 – Today I began installing the needed software for the Jetson Nano A.I. system. I have decided to start with the open-source software *Donkey Car* for training the A.I. model. This software will read the incoming signals from the throttle and the steering while capturing images from the front camera as data. The software will then take this data and train a convolutional neural network to mimic the behavior of the driver's footage. Donkey Car's guide recommends having 5,000 – 20,000 images to train the model on. After the model is trained the car will then use this trained CNN to take a picture and decide from it 20 times a second while driving. This makes it the best option to start with as the software is already developed and completely open-source along with it being used by many others in smaller-scale self-driving cars. While installing I ran into multiple issues regarding permissions of root folders and newer versions that the walk though says to install not being found by my Jetson. Unfortunately, I think this has wasted a day's work, but I plan to pick up where I finished tomorrow and see if I can successfully get this installed.

June 30, 2023 – Today I attempted to install the Donkey Car software again and had the same fate as yesterday. Although this time I did make it further than yesterday, I still was unable to get TensorFlow to properly install which is needed for the Donkey Car software. After fighting with it for some more time I decided to reinstall the SD card image for Jetpack. At first, I installed the version listed on the Donkey Car instructions but after rebooting it would not load up properly, so I then went and got the latest release off NVIDA's website. After doing this I followed the instructions for setting Jetpack up and Donkey Car. In the end, I was able to get everything properly installed.

July 1, 2023 – Today I created the Donkey Car application I will use for the CNN training. I also added deep learning into the Theoretical Background section as this is the mode of A.I. used by Donkey Car. After looking more into the way Donkey Car learns for the CNN training a possible issue has been made aware of. The issue is due to having the Arduino Mega 2560 the desired way of communication to the Jetson Nano is I2C, meaning the Jetson Nano doesn't directly receive the data from the motors and steering servo as it originally is designed for. To work around this there is the option of Donkey Car that uses a Robohat. The Robohat acts as the Arduino Mega 2560 does and sends the Jetson Nano data between -1, 0, and 1 for the training over a serial connection. The goal is to make Arduino mimic the Robohat to have this properly work; the goal is to still maintain the I2C communication between the two systems. If this cannot work, plan B is to create a custom A.I. training application as a time-saving precaution. I researched the needed software to complete this and found it may be easier than messing with Donkey Car. For plan B the required software would be a program for data collection, preprocessing program, training the A.I. model in TensorFlow then making a program to do real-time operations based on the trained model.

July 2, 2023 – Today I researched Donkey Car's data collection portion to start preparing for training the deep learning A.I. model. Unfortunately, the Donkey Car software is written so that the device controls the motors and steering servo or completes this via a serial connection to another device. This means that I do not believe the data collection software made by them will be usable while gathering data for training the model. I did, however, find that once the A.I. model has been trained, I should be able to use the Donkey Car Autopilot software for driving the car in real time. To get the data needed to train the A.I. model I will use OpenCV to capture an image from the CSI camera while saving the incoming data via the I2C line at the same time. The rest of the day was spent fighting the installation of OpenCV as I could not get it to successfully install until about the fifth attempt. After installation, I created a simple program in Python to show a window with the camera feed, but the software would not properly run during testing. Tomorrow I will research the issue and hopefully find a solution.

July 3, 2023 – Today I successfully got the camera to work after a few hours of research followed by trial and error. The issue at hand was caused by the V4L API I was attempting to use with OpenCV. Although I had installed and updated the API no matter what I did the camera still would not function. After reading through a few forums for the Jetson Nano discussing these Pi Camera v2's not working I found that there is another approach to using GStreamer. After using the GStreamer approach, I got the camera to work with OpenCV-Python programs. After testing out the straight video streaming program I played around with the different image ratios along with frames per second to get a perfect blend of quality and smoothness. I then coded a program that would get a single image from the camera and save it. This worked flawlessly so I then added a photo counter so the program loops until a certain number of photos are taken and saves them as "image_<number of photos taken>".

July 4, 2023 – Today I decided to change course as the second revision of the PCB has not arrived meaning the control system cannot yet send I2C data to the Jetson Nano due to the voltage difference; I also do not have any other 3V microcontrollers to temporarily use to develop the I2C communication/saving feature of the A.I. data gathering program yet. This allowed me to start finalizing the code for the control system where I quickly realized that I had coded the entire control system to use C formatted libraries and not the desired object-oriented style; I had originally planned to only use the libraries to prototype the approach of achieving the desire function with the final code being a class. So, after learning from my fast-moving mistake, I redid the code from libraries to be classes and redid the main program to use these classes. After this, I made a class diagram to insert to show the relationship between the classes in the program. In going through this process, I was able to fully learn how to design and create classes in C++ and was able to better the program by cutting down on the number of voids that were called in the main program for the indicators. Before, the main program called "useBrakes()" if it was activated 'true', with "useLeftTurnSignal()" and "useRightTurnSignal()" called if the 'pwm_side' (steering servo data) was more than half the range but with the new class structure in place the only void needed to be called is "useIndicators()" where all this is already nested in the void. Having the classes also allowed me to make voids private that did not need the ability to be directly called from an outside program, like the three voids mentioned above.

July 5, 2023 – Today I decided to test version 4 of the control systems code before moving on with the project. When testing the code, the back motors would slowly spin when trying to test just the steering controls. Also, when testing the forward and reverse controls of the car the motors would slowly spin in the direction before completely stopping as well. This was caused by the attempt of trying to have the void which saves the pulse width of the signals coming in from the RC receiver call the PWM mapping void instead of having the main program call it. After making the "readReciever()" void public in the class and adding "RC.readReciever()" to the main program at the beginning of the looping fold, the code worked as intended. I then updated the class diagram and added it to the newly created *Appendix D: Code.* Unfortunately, due to time restraints, I was unable to incorporate the automatic braking feature as I had originally planned, meaning that this will be the starting point for tomorrow.

July 6, 2023 – Today I decided to go ahead and design the camera mount for the A.I. computer vision part of the project. Before doing any designing, I took the camera and set it to the aspect ratio I plan to use for training the A.I. driving model and tested the view at different heights using a ruler. I found while using the chosen aspect ratio that 2.5", from the hood to the bottom of the camera module, was perfect. I also found that the camera did not need to be angled in any special way and that it could be mounted perpendicular to the hood with no issues. I took this and designed a two-piece enclosure that consists of the mounting bracket with a cover shell. Both pieces will be held together by the mounting screws that mounted a plastic shovel to the hood originally. Although the prototype was a rough-shaped idea of my desired end goal, after spending too much time on this I went ahead and printed the prototype.

July 7, 2023 – Today I tested the prototype and found that it worked as desired. I have decided to change the back shape of the shell to add to the overall appearance and reduce material usage as I am getting low on the PLA-CF material I have printed the rest of the parts in. I also went ahead and added a small brace to ensure any vibrations would be less if on rougher surfaces. After these design changes, I went ahead and created the drawing files and added them to *Appendix B* while the prototype was printing. The final version ended up printing successfully and looks way more visually appealing mounted to the power wheels car than the prototype, and due to the small design change the roll of PLA-CF had just enough on it to print it.

July 8, 2023 – Today I ran the cable for the front camera and tested the layout of the Jetson Nano and the Arduino Mega in the electronics bay. After finding the desired location for the Jetson Nano I went to drill holes in the vehicle's chassis to install the Wi-Fi antennas and realized the 12-inch extension cables I had purchased were SMA and not RP-SMA like I needed. Shortly after realizing this, I went ahead and purchased two 24-inch male RP-SMA to female RP-SMA cables and should have them by Monday allowing for the Jetson Nano to be mountable in the power wheels vehicle. After this, I decided to read through this paper as a check and found that the *Methods and Procedures* section for the *Design and Development of Hardware Components* sub-section was more appropriate under the *Results and Discussion* section for that sub-section. After moving it I decided to properly write the *Methods and Procedures* sub-section and added to the *Results and Discussion* for that sub-section regarding the reason for selecting the hardware components based on the data I referenced in the data sheets listed under *Appendix A*. I then created the introduction paragraph for the *Results and Discussion* section making sure to deliver the reasoning for the section while keeping it short and sweet.

July 9, 2023 – Today I integrated the auto-braking feature into the RC functionality code as well as coded the A.I. mode safety indicator. The way I was able to get the auto-braking working was by having it call the function to determine if the auto-braking is needed or not then right after call a function that returns the result of this. The reason the first function doesn't just return the result is due to errors when attempting to have it return the value of 'brake_activated'. The only way I could properly get this accomplished was by having the first function void while the result returner function was an int. I then made it so if the returned value was '1' (brakes needed) the 'use_indicators' function received '(1, pwm_side)' meaning the function gets a reverse direction signal turning on the brake lights while allowing turn signals to function normally; if the returned value was '0' (no brakes needed) the 'use_indicators' function received the normal '(direction, pwm_side)'. As far as stopping the motors the same idea as the indicators were used but with the 'driveMotors' function, if brakes were needed then the function received '(0, !direction)' meaning speed was set to zero while the motor direction was flipped; if no brakes are needed then the function receives '(pwm_fwd, direction)'.

July 10, 2023 – Today I tested the braking system with more challenging tests than yesterday, designed the motor controller breakout PCB and prepared the wiring for them. While testing the braking system I found that objects skinny like humans and poles were able to be recognized by the braking sensors from tire to tire meaning the complete front environment around the car is detectable within the three-foot set range. When going full speed, the braking stops the vehicle just in time to provide a few inches between the object and the front bumper, which for now is sufficient. When first trying to make the motor controller breakout board, I had attempted to use just a standard prototype PCB board but after struggling for an hour I realized the board looked messy and was shorted. Instead of trying to troubleshoot the short and make the appearance look better, I decided to just CNC mill a single-sided PCB. Unfortunately, due to time restraints, I was only able to design and prepare the G-CODE for the PCB.

July 11, 2023 – Today I machined two of the motor controller breakout boards on my SanSmart Genmitsu 3018-PROVer. This was done by using a V-cut bit with a 0.8mm drill bit for the through holes and a 3mm endmill to cut the profile of the board. After fabrication, I assembled the PCB by soldering two rows of 4-pin female header pin connectors to make a 4x2 female header connection and soldering the brown and green colored (striped and solid) wires of the ethernet cable to the corresponding pads. I also added the PCB schematic and board design to *Appendix C* regardless of the simplest design; the four-pin connector was not used instead I simply soldered the wires directly to the pads. I also made the computerized version, using Violet UML Editor, of the sequence diagrams for the RC Mode and A.I. Mode of the control system.

July 12, 2023 – Today I designed, and 3D printed mounts for the BTS7960 motor controllers and the 12V relay module board. At first, I had planned to use M3 threaded inserts for the screws to thread into but due to the small diameter I needed for the BTS7960 mounting standoffs when trying to heat them in place on the prototype, it was obvious the posts were too small and melted. I then decided to have them simply thread into the plastic. Using the same method with the relay mount the prototype was functional and no issues seem to be present from threading straight into the plastic. Unfortunately, due to running out of PLA-CF, the mounts were printed in Bambu Lab's Matte Gray PLA, this will not present issues as they are mounted inside the vehicle and not on the exterior.

July 13, 2023 – Today I decided to make the mount for the Jetson Nano Developer Kit and the Arduino Mega 2560 Rev 3. After printing the prototype, I realized that the distance needed between the two computing systems needed to be greater to plug the power input into the Jetson Nano. It was also noted that the Jetson Nano mounting holes did not line up properly. When trying to mount the Arduino Mega due to the female header pins being closer to the mounting holes, the head-on by M3 screws were too big meaning I needed to use one of the center holes on the board. After rotating the Jetson Nano so all the connections faced the same orientation as the Arduino Mega's I and attempting to fix the mounting hole issue I decided to simply print just the mounting standoffs instead of the full two-hour print. After multiple tries to get the holes lined up, I then realized it was caused by my making the Jetson Nano mounting holes the size of the screw I wanted to use (2.5mm) and not the actual dimension of 2.75mm. After fixing and testing this the alignment of everything was fixed and the mount was made. Right after fixing this issue, I received notice that the control system PCB boards were to be delivered tomorrow so I decided to find a way to make a SMT stencil for the logic level shifter SMD parts. After researching, I found that people tend to use aluminum soda cans to create D.I.Y. SMT stencils, so following the tutorial in Brain Benchoff's Hackaday article, I fabricated a stencil using my CNC machine instead of the traditional metal etching method. The results came out decent, while not perfect for the requirements needed it will work just fine.

June 14, 2023 – Today I received the revision B PCB for the control system shield. I then assembled a board today by using my recently purchased reflow plate for the logic level shifter SMT parts and a regular Weller solder station for all the through-hole components. After assembly, I made my best attempt at cleaning all the flux off using isopropyl alcohol but after a while of cleaning, I determined that the board was clean enough and that it was time to move on. I then mounted the Jetson Nano and Arduino Mega 2560 REV 3 to the computing systems mount and mounted it to the base of the power wheels vehicle. After it was mounted, I installed the shield and connected everything to the shield. I then decided to start finishing up part documentation by making the drawing file for the BTS7960 motor controller mount. I then added the JST datasheet and solder mask image to the appropriate appendix and finished the day by designing and printing the mode switch mount which then was added to *Appendix B.*

July 15, 2023 – Today I finished up the computing system documentation by creating and adding the drawing files for the 12V relay module mount and the computing system mount. After making and adding these I designed the 12V 9Ah battery mount and prototyped it. While the battery mount was printing, I went ahead and soldered and attached the JST B3B-XH connector to the mode switch and mounted it with the printed mount from yesterday. The battery holder prototype properly functioned so I mounted it in place and reconnected the computing system's battery. After this, I decided to go ahead and make the custom cable for the RC receiver to connect to the shield and once this was done, I mounted the RC receiver with some double-sided foam tape and connected it to the shield. Following this I decided to power on the power wheels for the first time with the shield and found that several things occurred. The first issue was that I accidentally got the Arduino Mega's power JST connection plugs GND and VCC flip-flopped which was causing a short circuit with the steering servo's power JST connector due to the shared GND. The second was that the left brake light stayed on faintly while the other indicators would blink like normal, this is believed to be due to code and was not attempted to be fixed yet. After this I realized that the mode switch was not functioning properly so after spending over an hour trying different things, such as serial outputs in the code on the high/low status of the "A.I._MODE" and "RC_MODE", I found that the issue was both sides of the switch would read as high. This was caused by forgetting to add pull-down resistors to the PCB shield meaning that the non-selected side would not get properly pulled down to GND. Luckily due to the open space between the shield and the Arduino Mega I was able to solder two 1K ohm resistors from the ground of the safety indicator light to each mode pin. This fixed the mode selection issue, but another problem was presented when in A.I. mode the safety indicator light did not function as expected. Instead of strobing as it was designed to do it simply would have all LEDs on then when turning the mode switch to center position it would light up only two sides, but when on RC mode the light was off. After these bugs in what is assumed to be the code, I decided to end the day by installing the WIFI antennas for the Jetson Nano's WIFI module. To have the antennas mounted on the back of the power wheels as desired I used two 24" RP-SMA extension cables and drilled two holes symmetrically on the back of the power wheels beside the back taillights. I then drilled two more into the computing system area and fed the SMA cables through.

July 16, 2023 – Today I resolved the issues found during yesterday's testing; the driving of the vehicle has not yet been tested due to the pre-driving occurring issues to keep the number of bugs being solved at one time low. The A.I. safety strobe indicator was the first issue tackled, for debugging the issue I used several serial outputs to state data such as the 'current_time', 'previous_time', etc., and found that the interval had a value of zero. This meant that when the code tested to see if an 'interval' amount of time had passed it would also be true due to the time passed being greater than zero. To solve this issue, I moved the setup of the variable 'interval' from the other class variables inside the 'strobe' function. The second issue solved was the left turn indicator constantly being powered on. This was not so easy of a bug to fix and in total took roughly three hours to troubleshoot. The first thing I tested was the continuity of the connector pins to ensure that no shorts occurred on both the shield and the wiring harness. During this, I found that the 'left indicator' pin had a 500-ohm lower reading than the other indicator pins. I then decided to see if the issue resulted in the PCB shield by removing the shield from the Arduino Mega and providing the 5V pin with the appropriate voltage and connecting GND. During this test, the indicator functioned as desired, and no issues occurred. This led me to believe that it was code related so I decided as my first check in the code I would disable every call to the indicator class and see if the problem still occurred, doing this showed that the problem still occurred. Finally, I decided to flash the Arduino Mega with a simple code to flash the left indicator on and off, and after uploading the program the left indicator always remained on. This then proved that it was not in fact code related but hardware related. So, using my benchtop multimeter I decided to read the voltages of each indicator pin and found that the left indicator would get a constant voltage of 4.65V while the other pins received 50mV when not activated. My first attempt to solve this was looking for any pieces of conductive material that could have possibly been connecting 5V and the indicator pin on the Arduino. After spending an hour referencing Arduino Megas's schematic looking for shorts and cleaning every joint on the ATMega 2560 chip it was obvious that this issue had to result somewhere inside the AMega2560 chip. To save on both time and money I decided to solve this by not ordering a new Arduino Mega 2560 but instead changing the 'left_indicator' to pin 31 instead of 30 which then allowed me to bridge the connection hole from pin 30 to pin 31 on the shield solving this issue.

July 17, 2023 – Today I began the last finishing items for the control system. I first started by finishing the drawing file for the battery mount then added it to this document. After this I connected and installed the Jetson Nano's 5V 3A micro-USB converter to power the Jetson Nano off the 12V 9Ah computing system battery. After this, I tested the logic level shifter on the PCB by providing 3V with a power supply to the proper pin on the JST connector and flashing the Arduino Mega 2560 with a test I2C sending code. The results of the test passed with the Arduino Mega sending I2C signals at 4.65V and the logic level shifter converting it properly to 3V. The final thing I did for the day was take the test I2C sending code and integrate it into the code for the RC functionality; this code was not tested yet.

July 18, 2023 – Today I started the final few steps left before finalizing the control system by integrating the I2C receiving code into the A.I. functionality. Following this I designed the Jetson Nano I2C breakout board and produced the GCODE files for fabricating the PCB. While the breakout PCB was being CNC milled, I prepared the cable for the I2C communication by crimping the proper connectors for a JST four-pin connector. After it was completely machined, I solder four female header connectors to the board (the board was designed for a 3x2 female connector but positions two and four are unneeded, so these were left unconnected) and soldered the wire straight to the board similarly to the motor controller breakout board. After this, I made the servo extension cable and ran both wires inside the power wheels vehicle completing the wiring for this project. After this, I decided to test out driving the power wheels after adding the computing system shield and found that the RC functionality did not function properly. At first, the steering servo would not move at all, and the brake lights constantly stayed on, after a close look I noticed that the VCC wire powering the servo had broken free of the JST connector, so I fixed this. To try and fix the brake issue, guessing what was caused by the auto-braking sensors, I went through the code to ensure that the pins listed on the code were the same numbers used for the shield and found that all numbers lined up as they were designed for. So, I used a few serial outputs in the code to print out the distance being read by the sensor and the value of the auto braking status. The issue was that the echo and trigger pins in the connector were backward so after flipping them the auto braking system functioned properly; due to rainy weather, the driving was not tested but will be tomorrow as part of finalizing the control systems code.

July 19, 2023 – Today I started to finalize the control system code and hardware. To do this I retested everything again from the RC functionality (I2C from the A.I. mode will come fall under the A.I. coding). During the testing for the RC functionality, I found that the left motor did not spin properly. To resolve this, I first checked the voltages coming in and out of the motor controller and found that the voltage going out to the motor was constantly at 0V. This was found to be due to the GND wire of the motor coming out of the motor controller's screw terminal when mounting into the power wheels. After fixing this the left motor still had no response so testing was done on the cable from the motor controller to the Arduino Mega, during this a broken pad connection for the forward PWM on the motor controller breakout board was found. To fix this I CNC milled a new breakout board and soldered it together. After this the motor worked as intended and the RC functionality was tested. During testing everything worked as intended and the steering servo showed to have a better steering ability than before. The believed reason for this is the made servo wire instead of the previous jumper wires allowing for a higher transfer of amperage to the servo. The last thing that needed to be tested was the I2C communication from the Arduino Mega to the Jetson Nano. To test this, I made a simple Python program to read the data and print it in the terminal. During this testing, I had several unsuccessful attempts, with the first being when trying to send the throttle, steering, and direction data. When trying to send all three data values the Jetson Nano would read the data being sent (0, 90, 0) as 0, 255, 255 then the Arduino Mega would stop communicating via I2C and the Jetson Nano could no longer find the I2C address of the Arduino Mega (the address was set as 0x04 to prevent any interference with other I2C addresses). After spending close to five hours testing different sample codes found in articles on I2C communication with the Jetson Nano I was unable to get this working. To ensure the completion of this project, if a solution is not found tomorrow a different form of communication will have to be chosen.

July 20, 2023 – Today I resumed the I2C communication issue and finalized the control system code, the A.I. mode code for the control system was not finalized as I am considering it part of the A.I. code. After researching to try and find a solution for the I2C issue I found an article on the NVIDIA developer forum that gave me the information about the Jetson Nano's stock kernel only allowing for the Jetson Nano to be the master's in communication with a required modification to the kernel to make it act as a slave device. After considering this and the way I was attempting to have the devices communicate via I2C, I decided to give one last attempt where the Jetson Nano would be the master device and the Arduino Mega would be the slave device. In this attempt, the Jetson Nano would request three bytes of data from the Arduino, which the Arduino would then send as a slave device when the Jetson requests it; this would also allow for better data collection as the program can request data with better timing sync instead of getting constant data reads that could be outdated by a few milliseconds. To test this idea, I reviewed the Arduino wire library example titled 'slave_sender' and modified it to send two variables of type byte and one of type bool. As far as the Python script I used the same method as yesterday that uses the SMBus library with the "read_i2c_block_data()" function, this allows me to control the address of the device I am reading and the number of bits that are being read (in this case requested). Following the read I added a few more lines of code to split the data array into the proper variables, which will also make the throttle data negative if the direction is 'True' or reverse. The first test of this passed with flying colors and no issues were presented to my surprise after the attempts of yesterday. Along with this working flawlessly the program for the control system was able to run as normal at normal speeds even when being requested for data. After this I added the code for the classes used in the control system, I also included the latest version of the main program. I decided to use subheadings under the appendix to section of the code for easy reference, this was technically not necessary as the code will be uploaded in the appropriate file and format to GitHub after this project. I realized today that I still need to add a switch between the two batteries VCC to control which one is being charged via the cigarette lighter port that was installed previously. Tomorrow, I plan to do this along with creating and adding a diagram for the A.I., creating and adding the complete wire diagram for the project, and beginning coding the data collection program for the Jetson Nano.

July 21, 2023 – Today I created the A.I. diagram to show how the process will work, created the wiring diagram, and began coding the data collection program. Unfortunately, I was unable to purchase the needed three-position switch needed for the recharging battery selection today. To create the A.I. diagrams I used a simple flow chart maker and created the A.I. training diagram along with the real-time A.I. diagram to show the processes. After this I created the wiring diagram inside of EagleCAD using the module option for the components for visual appearance, this does not show connections that can be found in the control system schematic as I wanted to keep it to the main connections from both batteries. With this additional, I decided to change the label of *Appendix C* to "Electronics" to cover the topics of PCB design and schematics for electronic systems. After this, I started programming the data collection software for the A.I. model training. To do this I combined the camera test code from July 3 with the I2C test code from yesterday and unfortunately could not get the camera to properly function once again. Unlike before the GStreamer backend in the OpenCV code did not fix the issue but made it so the camera was unable to save the image from the camera entirely. Using the OpenCV video capture feature the image would save a green image, this is assumed to be due to OpenCV only being able to read 8-bit data while the camera sends 10-bits hence the reason for GStreamer working in the test script. However, after a few tries the GStreamer and video capture methods worked but unfortunately, they are not reliable enough with the camera I originally planned to use. To fix this, I reinstalled OpenCV and built it for functionality with GStreamer and the problems still occur. The device will also list the camera as a device but will not open it in any application that uses a webcam regardless of which port it is plugged into. The concerning part is I can open the camera feed via the terminal using GStreamer, but programs cannot. To try and see if this was just a bad camera module, I attempted to use a Raspberry Pi camera I had scraped from an old project and found that even this doesn't work regardless of the advertisement that it should with no extra drivers on the Jetson Nano. This gave me the idea to try a USB webcam, which functioned properly 100% of the time using the video-capturing method. To try and still use the original camera I made a post in a Jetson Nano developing group in hopes someone may have some experience on this issue that could help, if not plans to switch to the USB camera will be made to ensure the finishing of this project before returning to college.

July 22, 2023 – Today I decided to go ahead and switch over to the Logitech USB camera after one last attempt at the CSI camera. Unfortunately, everything that was suggested to me in the Jetson Nano Developer group was things I had already attempted with no luck in them functioning so due to this the USB camera is the next best thing. After purchasing an ¼" screw I designed and manufactured the prototype for the USB camera mount. Once this was printed, I was able to check the mount on the power wheels and found that the mount worked as intended so I decided to print the final version with the little amount of PLA-CF I had left. I decided to print the top piece first to ensure that the visible part would be printed in full PLA-CF and then once it was completed, I printed the bottom. Unfortunately, I ran out of PLA-CF on the last six layers and had to finish the rest of the print with regular black PLA; fortunately, the two-tone won't be noticeable as it is mounted under the hood, and when the hood is up it's not the bottom side. After the final versions were completed, I mounted the camera to the hood and rand the USB cable to the Jetson Nano. After this, I finished the data collection program by adding the code that saves the I2C throttle and steering data to the designated output folders in the "Data Collection" folder. After this was done, I took 152 data packets to test the preprocessing on for tomorrow.

July 23, 2023 – Today I prepared the preprocessing programs for the A.I. model training code series. When hand reviewing the test data, I collected I found that the throttle and steering data sometimes had two sets of data saved to one file, this is assumed to be caused by when the I2C requests data right when it changes from one value to another. To fix this I made a Python program that would check the throttle and steering data to ensure they are in the allowed range as well as catch and fix doubled data. If it detected the double data (data consisting of four through six digits of data) then it would take the front half of the data; if five digits are found the assumption, is it is the moment the data value crosses into the three-digit values and the first two digits are kept. After this, I started researching how to process the images for model training. Since this project is not touching on real-time training the methods used for that were ignored as the data will be preprocessed after collection. After finding multiple articles on preprocessing data used for applications such as object identification, I found an article that is relevant to the project. In the article, the author Adhinga Fredrick discusses grayscale conversion, normalization, data augmentation, and image

standardization of images for A.I. model training. For this project grayscale conversion is not a desired processing process as the A.I. needs to learn the difference between road and surroundings, plus this can provide the ability to add additional features such as traffic light abilities in the future. The need for standardization of the capture images was deemed unneeded due to the images all coming from the same camera meaning cropping and area selection of an image is not required as they are already the same aspect ratio; this can however allow for future developments to this project such as adaptions of extra cameras.

July 24, 2023 – Today I began the process of creating the image processing program. I first started by installing all the libraries needed/used in Fredrick's article; this article will be the starting ground of the program. After spending an hour on the installation of the required libraries and running the code used to show just an image, I quickly realized that this was meant to be done in Google Colab, so I migrated the code over to Colab and everything ran smoothly. After following the article and using the normalization and data augmentation (changing brightness) I successfully had a program to preprocess an image. Unfortunately, after getting it to work I noticed that the colors of the processed images were being changed, the most noticeable was orange colors turning blue. Unfortunately, after a few hours of trying to get the processed images to save with the correct coloring, I decided to abort using the Colab method and go to a traditional Python script. After spending an hour or more researching how to properly use the normalization function in OpenCV I began programming my method of preprocessing the images. As hoped, the normalization method successfully worked, and this was confirmed by having it output the images data before and after normalization is applied. The last thing to do is make it so that the program will run through every image and output them in the desired folders; due to time, this will be completed tomorrow.

July 25, 2023 – Today I corrected and finished the normalization code for the data preprocessing and added the *Control & Vision Systems* portion of the *Methods and Procedures*. After thinking back to yesterday's issue of the images not properly saving, I decided to program a Python script that would load an image and print the pixel value to ensure that the values were correct. By doing this I found that the output images from the preprocessing program remained the same as the original image, this was quickly found to be due to a mistake in my code where I saved 'image' and not 'norm_image'. After fixing this the images reverted

to saving as black screens just as the Google Colab method did even though printing the pixel values before saving showed the proper values ranging from 0 to 1 not either 0 or 1. After a thorough search for a solution I was unable to find a solution to fix the OpenCV save issue, so I decided to try a method of preprocessing and saving using NumPy. To try out this method I decided to go ahead and add the image normalization code to the data processing program and just have one program for the complete preprocessing of the data. To use the NumPy method the program loads the image data using OpenCV, this is due to the data being an image and not a ".npy" file, then this data is processed through the formula for "norm_image" on Fredrick's Google Colab method article. This is done with every image in the data collection folder, data augmentation was not included yet as I believe the natural change of the images collected for data will be sufficient; this will be finalized once data collection is completed.

July 26, 2023 – Today I finalized the data validation and image normalization portions of the preprocessing program. To do this I ran numerous stress tests on the data validation for the steering and throttle validation functions. After this I decided to add a file checker to the image normalization function to check and ensure that the file saved by the program is the same in pixel value as the normalized image to ensure that the data is correct for A.I. model training; if it is not valid the filename is saved into a ".txt" file like the unsolved errors list for data validation. When going to text this I found it very time-consuming to have to sit and wait for the data validation portion to even begin the normalization, so I decided to add input options for the user to skip, perform the section, or exit the program. Doing this saved a good amount of time when stress testing the program. Once this was completed, I created the drawing file for the Logitech webcam mount and added it to the documentation.

July 27, 2023 – Today I went to collect data for the A.I. model training and hit several issues while doing so. The first issue I had was that I could not properly remote into the Jetson Nano off my phone's hotspot via my windows computer. The second issue was that the motors on the power wheels car would not work. After returning to my workshop, I successfully used SSH to remote into the Jetson Nano via a computer running Linux on my phone's hotspot. When reattempting to do the same on my Windows computer I was unsuccessful even though the IP address was correct. I then found that the motors did in fact work but the

auto braking feature was being triggered by the grass that was in front of the car and that the brakes weren't functioning. To test this, I found a ~1.5" pipe and laid it in front of the car and slowly rolled it forward until the front sensor was triggered, although no brake lights were turned on the auto braking feature was triggered. In the end, I found that the HC-SR04 sensor was able to successfully detect a one-and-a-half-inch object on the ground at roughly two and a half feet in front of the vehicle. After this I decided to test running the Python script and then disconnecting it from the display for data collection and found that the program seemed to stop after doing this, this was noticeable due to the blue light on the webcam not showing like when the display was connected. Based on this I decided that the best fix would be to reinstall the OS onto the Jetson Nano but instead use the headless mode option.

July 28, 2023 – Today I was only able to partially begin the headless option for the Jetson Nano due to slower Wi-Fi issues caused by a storm. I decided to go ahead and reinstall the OS for the Jetson Nano from the NVIDIA website to ensure that the installation was the latest install, after installation I flashed the OS file to the micro-SD card using Etcher. The next step was finding a 5V 4A standard barrel power supply since the headless mode requires setup via the micro-USB connector for serial connection. Unfortunately, I could not find anything that met these requirements, so I chose to take an unused barrel connector power supply and cut the wires furthest from the connector. After this, I stripped and tinned the wire with solder and used my DC power supply to supply power to the Jetson Nano with exactly 5V at the 4A limit. This allowed me to set the Jetson Nano up and begin the installation of OpenCV. Unfortunately, when I tried updating/upgrading Jetson's Linux files and libraries before installation of OpenCV the Wi-Fi went out for a few hours. After the Wi-Fi connection was restored, I was able to successfully install the updates/upgrades along with the files needed for OpenCV, however, I was not able to build OpenCV due to the slower downloading speeds. While the Wi-Fi was out, I did come up with a plan for if the mobile hotspot did not work again. This plan consists of purchasing a 5V 4A barrel connector power converter with a male-to-female micro-USB cable that has a built-in mount. This will allow me to remote into the Jetson Nano through the serial port allowing for the control of the Jetson Nano if the hotspot gives troubles, however, this will not be implemented until the next data collection attempt to ensure it is truly needed.

July 29, 2023 – Today I built OpenCV using the same Linux guide on the OpenCV website as before. This time however I noticed that it seemed to take several hours longer than when I was using a GUI. However, after several hours OpenCV was completely installed and ready to use. After this was done, I installed "smbus" and "pip" back onto the Jetson Nano as they require libraries that do not come already installed on the OS. While all this was installing and building, I decided to troubleshoot the brake indicator issue. My first assumption was that the connecting piece inside the JST connector may have broken off from the wire but after a quick inspection and test, the connector was deemed not part of the issue. After testing the continuity of the traces for the brake indicator on the PCB shield showed that the trace was connected properly, I decided to measure the voltage being sent from the Arduino Mega's pin. After flashing a program to the Arduino Mega that held the GPIO pin for the brake indicator high I measured the voltage between it and ground and found that the pin never left 0V. So, marking it the same issue as the left turn indicator issue a while back but this time the voltage was stuck on 0V and not 5V. To fix this I repeated the same bridging technique as for the left turn indicator making the brake indicator now pin 35 instead of 36. This was all I was able to accomplish today due to time restraints and the roughly four-hour building process of OpenCV.

July 30, 2023 – Today I was able to successfully obtain data for the A.I. model training. Before leaving for the local park, I made sure to do a good inspection of the hardware components along with testing the vehicle's functions. I also made sure to connect the Jetson Nano to my iPhone's mobile cellular hotspot as a means of remoting in with my laptop while not having to worry about a cable connection. During data collection, I was able to run the power wheels car for roughly one hour continuously with only a few breaks to save data sets. I decided to save the data set into four separate sets to ensure the entire data set wouldn't be lost if some technical difficulty happened. One issue I did have was that the data collection was not saving at the rate of 20 data points a second as desired, to somewhat aid in the more data points per second I changed the image format after data set one to PNG images from the original jpg format. This allowed me to get roughly 50 more data points in the same amount of time as before. Unfortunately, due to the high current draw of the motors the drive battery quickly died leaving only 484 collected data points. After

returning to the workshop, I installed the recently received three-way toggle switch, this will allow for the selection of which battery to charge meaning there is only a need for the already installed single charging port. After plugging the battery charger in I was curious to see the percentages of the battery power left on each battery, to my surprise the computing systems battery only used 5% of its capacity meaning that it draws roughly 500mAh. The motor battery, however, used 85% of its capacity meaning that the motors draw roughly 7.7Ah. Due to me saving the data into four different data sets during collection the data sets now needed to be combined into one giant data set. After spending a couple of hours, I was able to get a program coded that combined the data sets together and converted the PNG files to jpg; this was done to prevent having to edit the data processing code. By programming this Python script, I was able to combine the data sets into one in 46.1 seconds with the throttle data combining in 0.24 seconds and the throttle data in 0.23 seconds. By creating this program, I can now use it to combine any data sets in the future within a small amount of time. This was also programmed without multithreading meaning that the ability to increase the speed is very much possible. During data collection, I did find two items of concern that will be changed. The first was an issue that occurred at the first attempt at data collection, this being the issue with the auto braking going off when nothing is truly in the way. This was once again concluded to be caused by the grass surrounding the pathway, to prevent this I simply unscrewed the front sensor and tapped it so that it was pointing in more of an upwards angle. I believe that the height at which the sensor is mounted is the issue as the sensor's 15-degree scan radius can pick up on the grass so with the sensor angled upwards the auto-baking is still correctly functional, but the bottom of the scanning radius is now higher than the grass. The second was already in place for the data collection process, this being the max speed being set to 50% of the max speed output from the motors. By permanently having the speed at this limit a human can comfortably walk beside or behind the power wheels while it is in operation. This will allow for the vehicle to also be safer for the surrounding people or animals. I also had the idea to add a kill switch for when the A.I. is driving the vehicle for the human to disable any movement from the power wheels. The idea is to use the RC remote by making it so that if the control stick is in the bottom left the power wheels will stop and then to regain movement the user will have to move the stick to the upper right corner.

July 31, 2023 – Today I went through some of the images to see if any data augmentation techniques would be needed during the preprocessing process before sending the collected data through. While going through the images I noticed that many of them were close in image brightness so I have determined the brightness change augmentation will be applied. I have decided to do 20 randomized augmentations per image with the brightness varying from 50% to 200% as I found this to be the most ideal range while testing on image zero. To do this I unfortunately reprogrammed the entire data preprocessing Python script leaving techniques like the original. This was needed due to the desire to produce 20 images per photo since there will be nested loops so that the 20 images are saved before the next image is loaded. This is also performed on the data for the throttle and steering values before they are mapped to a 0-1 range for the A.I. training model. To also ensure that the data being read is similar I changed from the "for file in the data set" method to the "while data_number != len(dataset)" method, the method I used for the combination script programmed yesterday, with the path then being fully constructed ("processed_image_") with just the current image number and file extension being added to it. This ensures that the data is processed numerically and sorted for everything so that the data properly lines up. In total using the now included data augmentation method the A.I. model will get 9,420 points of data for training from the original 471 data points. The method for data augmentation itself was first attempted with the method shown in Fredrick's article but I was unable to get the proper results when using his *Changing Brightness* method, the issue was the colors of the image would be altered as if inverted. To fix this I was able to find an article by Aishwarya Singh on image data augmentation processing using skimage in Python. Using skimage the image brightness is changed by altering the image's gamma, which thankfully kept the image colors correct allowing for the normalization of the image (still using Fredrick's method) functional. After the software was completed, I began preprocessing the data, with the throttle and steering data only taking 6.65 seconds to complete the validation and duplication of 18,840 total files (9,420 for each). The augmentation and normalization process, however, unfortunately, hit an out-of-memory error 50.96% of the way through outputting the image files. An issue I would not have imagined occurring but unfortunately did, out of the 4,808 outputted .npy image array files it takes up a total of 380 GB of storage at about 81 MB for each file.

August 1, 2023 – Today I found a 1TB external hard drive and ran the data preprocessing program on it to ensure enough storage for the output files. Before I could do this, I realized that the mapping of the steering values was incorrect from my original plan, the original plan being to have the mapping be from -1 to 1. Before rerunning the program, I added this in so that the left mode turn is values -1, steering straight is 0, and the rightmost turn is 1. After this, I decided to completely redo all the data preprocessing, this time the throttle and steering data took 50.87 seconds. The image preprocessing was completed and was able to process the data at an estimated 2X faster speed for unknown reasons. In total saving the 9,420-image array .npy files took only 45.59 minutes to complete with all the files taking up 745GB of storage. After this, I began slowly researching how to train an A.I. model for the self-driving portion. To my surprise, the number of articles discussing the topic of self-driving A.I. model training was slim with most articles titled in a way to seem helpful only being on the techniques or methods used and not an actual discussion on how to accomplish just a topic. Due to my little knowledge of this topic or A.I. model training the need for an article with some sort of guidance (such as example code, like the previously used websites/articles) is a must to accomplish the desired goal. After a few hours of reading through articles in search of something helpful to point me in the right direction, or the foundation on how to just train the model I came across an article by Abhinav Sagar an undergrad from the University of Maryland published on Towards Data Science. The article titled *How to Train Your Self-Driving Car Using Deep Learning*, was the first article to list code to back the topic up for discussion. The issue with the code is that the code is made to train an A.I. model using three separate camera views, these being the left side of the vehicle, the center, and the right of the vehicle. After looking through the code, however, and seeing the techniques used allowed me to better understand how the process is completed when attempting something in this manner. After reviewing the code, I made histograms for both the throttle and steering data to show the repeating occurrences of each possible value like the ones shown in the article, this was done by creating a quick program using NumPy's histogram function. After a little more research just to find a second source of information I came across a GitHub project by Murtaza Hassan labeled *Neural Networks Self Driving Car Raspberry Pi;* however, it should be noted that the presence of training with throttle data in this approach

is not found only steering values are given. After reviewing the repository, I found a link to Hassan's website where he posts courses on Computer Vision and A.I. applications. Here I found a free course over the GitHub repository where Hassan goes through each step of the program and explains what the code is doing. This helped greatly as it gave me the idea of how the processing software functioned as showed me that my attempt at processing the data and saving it ahead of time may not be as efficient as I believed it would be. For a better understanding, I plan to follow this course to the end to see the full process a source for my research into A.I. programming. After completing the course, I then plan to reference his code and make it work for this project by including the throttle data by modifying the code. I feel this will provide a good attempt and functional foundation for me to later build on after further learning how to better program A.I. models and further the abilities of the vehicle itself in the future. I chose to reference the GitHub reference as this is where the complete code is listed, instead of the course landing page where you can only obtain videos of the code explanation.

August 2, 2023 – Today I sat down and started comparing the two codes found yesterday during research and found that much of the code was very similar. For this reason, I decided to instead merge parts of Sagar's code into Hassan while adding the ability to train for throttle values. From Sagar's code, I added the training and validation data histograms while maintaining Hassan's histogram displaying method as Sagar's was not clear. I had also planned to add the *Loss vs. Epoch* graph generator code but before starting seen that Hassan already had this in place. To implement the use of throttle data I looked through the code along with his course videos where he is explaining the code. The first step was setting up the third column and labeling it *"Throttle" and* then updating the step one function caller in the *Training.py* file. Once this was done, I worked on having the visualize and balance histograms display both the steering and throttle data, this is done by showing the data as normal and then allowing the steering data to be balanced. After this the balanced histogram for both data points are generated, this is done this way due to too much data being removed if the throttle then balanced itself. Following this I then searched through the code and found anywhere that steering was mentioned and duplicated the functionality for the addition of throttle use. The last thing to do was properly crop the images for the data augmentation. Unlike my program this program

performs data augmentation at a 50% chance of happening so that the same augmentations are not applied to every image; augmentation methods include panning, zooming, changing brightness, and flipping the image. The surprise to me was how no images were saved throughout the code, instead, it would hold them as variables throughout each process which was then turned into a NumPy array. This gave me the thought that the data I did preprocess may be able to work with model training. After a few trials and errors fixing random errors, such as maxing out the index due to not resetting it between the jump from steering to throttle data, I was able to get a model to train. After the model was trained, I used Hassan's implementation code as a test to see the data the A.I. model would output from the images collected during data collection. To do this I simply made a program that would loop through all the images and run them through the model and save the output. To my surprise, the A.I. model only produced a single value instead of the expected two values. Although the throttle data mostly consisted of the max speed (127) used in the training, a throttle value was still anticipated.

August 3, 2023 – Today I was able to get the A.I. model outputting two outputs, I also made the neural network diagrams as a visual to see the node segments and retrained the model for dual outputs. To get the two nodes on the output I used the method shown on the sequential model page on TensorFlow's official website where the create layers to a variable and then use the variable as the reference of the previous layer to the next layer, this links all the layers together over time and then allows for one model build at the end. For testing the training of the model for two outputs I trained the model with only two epochs just as a quick and easy testing base. The first time around the model was still only outputting value so I had the idea to add a second dense layer at the end with a value of one and added names to the dense layers. In the model build I then tried having the outputs be an array with both the variables I assigned the dense layers to as the output. To my surprise this was functional on the first attempt, so I allowed the model to be trained with the original 10 epochs with 100 steps per epoch. After roughly 95 minutes the model was completely trained and showed way better results than any previous model that was trained with the single value output. Running it through the testing program I noticed that a handful of values were out of range and were none usable, to make these usable I made all out-of-range data the max value allowed. After doing this I noticed

the output data for the throttle was about three times lower than expected with the max value only being 0.336. To fix this I made the throttle sensitivity three while increasing the steering sensitivity to one as I felt the steering value was lower than desired. After this everything worked as intended and the values seemed reasonable for the images being run through the tester. After this, I researched how to create an A.I. neural network graph and found a website I have used to create flow charts for previous projects called draw.io. For the number of nodes in each layer, I decided to use the number of nodes listed for the dense layers as the conventional layers get flattened before the addition of the dense layers. As seen in multiple examples I decided instead of making the graph 100 nodes in height that I'd use a shortened graph that uses the three dots to show there are more nodes than shown. Since I did this, I made sure to label the weight of each layer in between layers in the form of "W1[3, 100]" where three is the number of previous nodes and 100 is the number of nodes in the next layer. After this, I coded the Python program to implement the A.I. model in real-time operation. For this, I used Hassan's *implementation.py* program as a reference for how it should be structured. Obviously due to the communication being done via the I2C protocol with the A.I. model also outputting two variables the way the implementation is different while maintaining the same ideology.

August 4, 2023 – Today I unfortunately spent several hours troubleshooting issues relating to the Jetson Nano developer board. When installing the battery selection switch for charging the other day, I had forgotten to disconnect the front battery and the end of the wire managed to contact the ethernet ports shield on the Jetson Nano creating a small spark. Not thinking much of this I dismissed it and thought nothing of it, that was until today when I went to test the implementation code on the Jetson Nano. When powering on the device it seemed to be functioning as normal using the serial connection through the micro-USB port with power being supplied via the barrel connector. The first issue I encountered however was that no matter what the Wi-Fi card no longer functioned, meaning I was unable to use Wi-Fi; it should be noted that when testing via wired ethernet connection the Wi-Fi functioned properly. After this, I assumed that the Wi-Fi card had just been burnt out in the accident, so I decided to try another I had from a scraped laptop. Unfortunately, this one did not function as hoped, this led me to try a USB Wi-Fi adapter which

also did not function. In fact, upon further review, none of the USB ports functioned. I then checked the GPIO pins and found that the +3V3 and +5V pins supplied the correct voltage although none of the other pins were tested for functionality. In total all this testing took up roughly two and a half hours before I finally concluded that the carrier board for the Jetson Nano was burnt up, regardless of the Linux functionality still working. A new Jetson Nano has been ordered and is expected to be delivered in two days, allowing time to finish this project before returning to school. Fortunately, to get some work accomplished on the project today I found a Raspberry Pi 3 B+ which I was able to use to develop the proper I2C communication between what will be the Jetson Nano and the Arduino Mega 2560. Although the A.I. model was not trained on any reverse driving data I coded the implementation program to incorporate sending the value of the direction for future developments. I now plan to go ahead and develop the code which the control system will use to control the power wheels by receiving the data from the future in-place Jetson Nano that's temporarily a Raspberry Pi; Randomized data in the allowed range will be sent from the Raspberry Pi to the Arduino Mega every two seconds to test that the controls work while the power wheels are up on stands in the workshop to prevent unwanted movements.

August 5, 2023 – Today I updated the A.I. diagrams and made a few modifications to the A.I.-related programs. The first thing I did was update the A.I. diagram to incorporate the new process used for the A.I. portion of the project, this includes the new programs that were not originally planned for like the data set combination and data logging programs. I also made changes to the preprocessing program to allow the user to input the number of data duplications they desire; this will allow for the use of the data validation feature for the current A.I. process while leaving the ability to use the image augmentation feature for future projects/developments on this project. Along with this edit, I decided to redo the model testing program to display the image it is testing from while also displaying the value of steering on the image itself. Originally, I had tried programming this to incorporate a feature where it would read the actual data it was trained on and compare it to the predicted number; from this, I had planned for it to output a sensitivity number to allow for the lowest difference range between the real and predicted data. Unfortunately, I was unable to find an accurate and consistent method for making this feature in the software. Fortunately, after adding the

text display to the OpenCV image windows, I noticed that the steering angles seemed to be close to the real-world data meaning no sensitivity will be compensated in the A.I. output prediction. I also emailed Murtaza Hassan seeking permission to publish my modified version of his training code in this paper. While reviewing my modified code before sending it to Hassan to show the modifications I made, I noticed I had made an error related to the data I fed the A.I. model while training. For the training I had given the correct image and steering data but had unfortunately given the model steering data for the throttle input. This explains why the data was roughly three times smaller than expected when originally testing the model; luckily this was an easy fix and retraining the model can be completed overnight. After this was done, I reviewed and finalized the logging code used to take the path of the images collected and the steering/throttle data and place it inside a spreadsheet file which is then used by the A.I. training model. The code is now placed in *Appendix D,* along with the updated data preprocessing code. At the end of the day, I received an email from Hassan granting permission to publish my modified version of his original code in the paper.

August 6, 2023 – Today I received the new Jetson Nano replacement and finished preparing for the real-world test. The first thing I did today was unbox the new Jetson Nano and insert the micro-SD card along with the Wi-Fi card. To my surprise, the Wi-Fi card no longer worked meaning it too was shorted out along with the old Jetson Nano. After attempting two other Wi-Fi cards I had along with two USB Wi-Fi adapters I had, I was unable to get the wireless Wi-Fi functioning. I went ahead and ordered a replacement Wi-Fi chip and began looking for a temporary fix so I can still test the A.I. model on the set timeline. The method I found via the read-me files from the serial connection method for the Jetson Nano, was using an ethernet cable connection between the Jetson Nano and the host computer (in this case my windows laptop). This is done by connecting the two via the ethernet connection ports and then using SSH, in my case PuTTY, to remote into the Jetson Nano the same as if you were using the micro-USB connection besides this not being through COMs. After this I tested the A.I. functionality code with an implementation test code, the implementation test code randomizes data in the allowed ranges and then sends them to the control system for it to act on. This control system's A.I. functionality code functioned correctly the first time but one issue

I quickly realized that needed to be addressed was that when the Jetson Nano stopped transmitting data the control system would act on the last set of data sent. This meant that it would never turn off and continuously keep running. To fix this I created a check to see the amount of time between the runtime and the time the last data packet was received, if this time interval is over one and a half seconds (this will be lowered for real-world implementation to 750ms) then the sent data is overwritten with off data. After this, everything functioned perfectly and the confidence of a smooth first test was developed. While this was going on I trained another A.I. model as a test with twice as many epochs along with steps per epoch. This model surprisingly performed with a greater average difference between real-world and predicted data than the model which was trained with only 10 epochs with 100 steps per epoch. To visualize this, I reattempted the original desired feature I had planned for the "*model testing v2.py"* this being the difference average along with proper visualizations for the comparison of real-world to predicted data. I was finally able to accomplish this which now means the model tester gives the average difference and shows graphs to visualize this result. This is where the notice of the "less" trained model performing better was observed as I was able to compare the results between the two easier than before. The less trained model had an average difference for steering data of 0.564755 and a throttle difference of 3.917197; the more trained model had a steering difference of -1.123142 with a throttle difference of 2.059447. Although the more trained model seems like it would be a better model due to the "meet in the middle" type of difference between steering and throttle, the less trained is more desirable since the steering difference is lower and due to the throttle not being as devastating as steering accuracy at this time. Although the A.I. functionality code works as intended there still is one feature that I need to put in place for safety, this being the kill switch via the RC remote. After the time taken from model training today, I have decided that any future trained models will be trained via a rented server to free up computer resources allowing for work to be completed while model training. At the end of the day when I sat down to write this journal entry, I realized that I have not yet created the drawing files for the front and back HC-SR04 ultrasonic sensor mounts, meaning this will be completed tomorrow.

August 7, 2023 – Today I had originally planned to test the self-driving portion of this project at my local park but due to unforeseen weather this plan was postponed till tomorrow. Instead, I went ahead and created the drawing files for the front and back HC-SR04 ultrasonic sensor mounts. After completing this I decided to go ahead and rent a server to perform a few more test trains to have various options on selections of models for testing. Using DigitalOcean I was able to set up a server with it being activated in less than five minutes from the creation point. Once the server was live, I began uploading the combined data which was collected along with the log maker program. After this, I transferred the A.I. model training programs and began installing the needed modules needed for them to function. After this was all done, I ran the log file maker, this ensured that the path of the images for training was correctly set for the server paths instead of the host computer. Before I did anything I went through the codes to ensure that anything that would regularly display a figure was turned off and set to "plt.savefig(<path>)", this allows for the graphs to still be obtained from the terminal. After this, I went to train a standard 10 epochs with 100 steps per epoch, and to my surprise, I was getting an error that the image path did not exist when trying to open the images. Unfortunately, after spending two hours trying to find where the code was failing I found that this was due to how the Linux server handled the "myImagePath = os.path.join(myImagePathL[0],myImagePathL[0])" unlike Windows where it creates the "captured_image/image_<image number>.jpg" path used to open the image the server was creating a "captured_images/captured_images" path. To fix this I changed it to "myImagePath = os.path.join("IMG", myImagePathL[1])", this changes the folder to the image data set folder and grabs the "image_<image number>.jpg" portion of "myImagePathL". After this *Training.py* successfully worked and model training began. After the model was finished training, I was able to transfer it from the server to the host PC by using the "scp -r" command on the created "Figure" directory. The first model trained had a steering difference of 3.388535 and a throttle difference of 7.585987, these are way higher than the favored model stated yesterday but training took roughly 25% less time with better server options available for future A.I. needs. During this test, I noticed that on the result graph, both losses are minimum at seven epochs, so I then went to look at the other A.I. model's result graphs and noticed similar results with the lowest loss being around six to seven epochs, this allowed me enough backing information

to justify training one last model this time with seven epochs and 100 steps per epoch. After training I ran the test of the model and found that the steering difference was 2.564755 while the throttle was 2.042462. To better compare I decided to get the lowest and highest difference from both this model and the favored one from yesterday, in the end, this just-trained model has a lower range of difference with steering being from -17 to 25 and throttle being from -57 to 8; the favored model has a steering range of -16 to 35 with throttle of -57 to 22. In the end, I have decided to first test with the most recent model and then test with the now previously favored model. After this, I added the kill switch function to the A.I. functionality code so that it was in place for tomorrow for testing. This consists of still reading the RC receiver data when in the A.I. mode but only acting if it is in the position of bottom left or top right on the control stick. For future developments, the goal is to assign this to a switch on top of the transmitter. Following this I decided to go ahead and create the HC-SR04 ultrasonic sensor mount drawing files for both the front and back sensor, both were then added to *Appendix B;* it should be noted I first designed a wedge to allow for the sensor to be angled like it was taped in during data collection before doing the drawing file. After this, I decided to review the implementation code one last time before the test day. During the review, I found that I had failed to put in an out-of-range data fix, so I went ahead and added that in place. Once this was done the implementation code was completed and I added it to *Appendix D.* While adding the implementation code I decided to go ahead and add the model tester code and both scripts for training the A.I. models. As an experiment before shutting down the server rental, I wanted to test what would happen if I trained a model with 40 epochs with 100 steps per epoch, although this will probably result in an overfitted model out of personal curiosity I have decided to run this; the total runtime of the server for project use was ten hours not counting this model. After returning to see if the model was trained and ready to test, I found that the model did not finish training. After looking at the DigitalOcean server dashboard I noticed that the time around when I closed the remote window the CPU usage dropped meaning that this action was probably what stopped the model training.

August 8, 2023 – Today I prepared for the real-world testing items and trouble shot unexpected errors. The first thing I did was install the recently received Wi-Fi replacement card to be able to remote in via my windows laptop. After this, I tested the kill switch code that was added to the A.I. functionality in the control system. After this I went to test the implementation code and kept receiving errors that the "tensorflow.keras.models import load_model" was not found. After trying to reinstall TensorFlow twice with no success I finally was able to get it installed properly using NVIDIA's *Installing TensorFlow for Jetson Platform*. With this being the first method, it ultimately failed due to an error that the used version in the command was not found, so following this, I tried using the "pip3 install tensorflow" which resulted in a requirement already satisfied message for everything. After this I decided to try going to the URL in the NVIDIA method by removing the version at the end, this resulted in a page with multiple versions. In the end, I was able to get TensorFlow 2.3.1 installed and functioning, the issue turned out to be due to the version of Cuda which is installed with Jetpack not supporting the latest versions of TensorFlow obtained by other methods. After this issue the *Implementation.py* program was able to run without the "Illegal Instruction" error, however, this introduced another issue. While training the A.I. models on the host computer I failed to maintain the same version of TensorFlow along with the models being trained to use a CPU and not a GPU; this is due to TensorFlow not recognizing the GPU on the host computer. To get around this I decided to try and train a model on the Jetson Nano itself. To do this I transferred the appropriate files to the Jetson Nano like what I did with the server yesterday. After this, I started installing all the prerequisites such as modules and libraries the *Training.py* program uses. Unfortunately, I was unable to properly get the "imgaug" package installed meaning I then resorted back to installing the proper software on my host machine. After spending two hours installing the proper versions of Cude, CuDNN, and Visual Studio Code C++, I installed them and was ready to move on to installing TensorFlow. Like everything else today installing TensorFlow was unsuccessful due to oldest version available to be located is 2.8.0. Feeling as though today was a complete waste of valuable time I decided to finish the day on a good note by completing the *Control & Vision Systems* section for *Results & Discussion* along with adding all A.I. system codes to *Appendix D.*

August 9, 2023 – Today I decided to reimage the OS for the Jetson Nano on a new 64GB micro-SD card and try reinstalling the requirements for the model training and implementation programs. To do this I followed the same steps as before which consisted of formatting the card and then writing the OS. While the required modules were installing, I decided to go ahead and create the bill of material for this project; this only includes items used in the finalized version. After spending roughly eight hours getting everything reinstalled on the Jetson Nano, I wanted to test the modules and output the version to ensure that the module was made correctly. To do this I made a Python script that simply imported them and then printed the version for each one. When running this I received "Illegal Instruction (Core Dumped)" for the module "pandas". This was fixed via a quick search which led to a Stack Overflow forum posting on the same issue. In this forum I learned that this is caused by the installed version of NumPy 1.19.5, downgrading to NumPy 1.19.4 solved this; also mentioned in this was adding "export OPENBLAS_CORETYPE=ARMV8" to the Jetson Nano's *"bashrc'* file. After this I restarted the Jetson Nano and the issue seemed to be fixed, however, this returned another issue. This time the issue was "cannot locate memory in static TLS block" caused by the "*libgomp"* package. To fix this I did a quick search like the previous issue and found a forum on the NVIDIA developer site that gave a command to run before running the script since it was an "export" command I decided to add it to the *"Bashrc"* file; it is unknown if adding this to the file helps any or not. After this, I was able to get the test program to not return any errors, which meant I felt ready to move on. After transferring the files over the same as for the server and the previous time I ran the *Training.py* script. The program returned an error that was related to the "*matplotlib"* module not being able to show the graphs made, to solve this I simply changed the "plt.show()" lines to "plt.savefig(<destination>)". This fixed this issue and allowed the program to get to the A.I. model training portion. However, once I started training a replicated model like the ones favored that were trained on the host computer, six epochs with 100 steps per epoch, after two steps during the first epoch I got a "Killed" message and the training program stopped. This made me question if it too was trying to train with a CPU, so I decided to use a test program found on TensorFlow's Python API documentation page. This output that a GPU was recognized meaning that TensorFlow should automatically recognize this and utilize it during training. The reason I believed that it

was using the CPU was due to it killing the program itself, leaving me with the thought that it could have maxed out its resources. In hopes of finding out if this is true or not, I researched for a way to monitor resources on the Jetson Nano leaving me to come across a program called "*jtop"*, this program is the closest thing to a resource monitor on Windows where I can monitor all resources at the same time in one place. Using *jtop* I opened another SSH terminal using PuTTY and executed the needed command to open the program and was then able to watch the resource usage. While viewing the resource usage I ran the training program and to my surprise found that the issue was not maxing out the GPU but the memory. Due to the Jetson Nano only having 4GB of memory the model architecture must be too big meaning that it maxes out causing the error. To try and test this assumption I decided to cut the number of nodes in the model, after running the training program this showed to help as it made more progress than the time before. However, I noticed that the time per epoch still took the same time even though the layer sizes were smaller making me wonder if the batch size (steps per epoch) were still too big. After some research I found an NVIDIA developer forum where a user was having an issue with the accuracy of the trained model when using different amounts of epochs and batch sizes, commented on this was a solid recommendation which stated that for his dataset size of 300 images, 5 epochs would be a good number as this will prevent overfitting in the A.I. model. Taking this and turning it into one epoch per 60 images I used this to get a rounded-up number of eight epochs, so I changed the number of epochs. Also stated was that the recommended data set size is two to three times the number of classes in the model, this meaning that the dataset should be six to nine. This number seemed low to me as the original A.I. training program on my host computer used a dataset of 100 but using the knowledge from someone more experienced on this topic I decided to test it out. This too resulted in the training program freezing but the max out was during the translation from the fifth epoch to the sixth. I then decided to cut the arcature down by half once more leaving it a quarter of the original size, and to my surprise, this model was able to be successfully trained with results better than any model from the host computer; the maximum memory usage was 3.8GB.

August 10, 2023 – Today I was finally able to perform the real-world test at a local park. Going in this test was probably the most nervous moment I had as this test could be the final step (besides documentation) to this project. Before testing, however, things were very rocky, when trying to accomplish the execution of the *Implementation.py* program I received an error message. The first issue I received was regarding OpenCV not liking the cropping preprocess line of code. After reviewing the program, I discovered that the issue was the capture was never being released meaning the return of the image was not properly functioning, this was due to the "cap.release()" being commented out due to being accidentally placed on the same line as the comment on what it was used for. After moving it to the line under the comment where it originally belonged, this solved this issue. The next error I received was due to the third argument in the "write_i2c_block_data()' function being a list of more than 32 bytes. After reviewing the entire implementation code, I could not locate any issues which could be causing this in any way, shape, or form. It wasn't until I was looking through the *Model Testing v2.py* code that I realized the issue was I never rounded the data converted from the A.I.'s prediction changing it from a type of float to int meaning it was over the byte limit. After this, I added a simple test line of code to output the data wanting to be sent before sending it to the control system and saw that the results for steering were 90 less than what was expected. This was due to having the lines which mapped it from the [-1, 0, 1] range to [75,140] range using "*=", to my surprise this function was not doing what I originally had thought it would do. Originally, I had thought that it would times the steering prediction to the "(90 – 75)" and then after this add 90, however, it instead multiplied it to the full "(90 – 75) + 90"; this made it -0.5 times smaller than what was the desired number. To fix this I simply turned it into "steering = steering * (90 – 75) + 90" which works as I intended the original method to. After spending roughly 30 minutes troubleshooting and correcting these errors it was finally time to test the self-driving portion. IT WORKED! Well sort of, the A.I. performed well with the straighter parts of the park's pathway, I would like to note that on a few occurrences, the A.I. would drive the power wheels off the path after some runtime. Unfortunately, when it came to the curved parts the A.I. did not perform as I hoped. The A.I. would turn the tires in the right direction but the angle at which the tires turned was not as drastic as they needed to be, meaning at roughly halfway through the turn the vehicle

would run off the path. This is believed to be due to two reasons, the first being the size of the trained dataset playing into this. Due to the limited sizing of the dataset, very few data points in the dataset occurred at turns, this means that for future A.I. model training, more data on turns need to be included. The second reason is more theory-based than as factual as the first reason, this being that the A.I. doesn't properly recognize that it is heading in a turn. This is believed due to how the image is preprocessed before being sent to the A.I. model, the image which is cropped to reduce unneeded clutter in the images is resized to stretch the image to almost a top-down view of the path. Due to this, I believe that the curvature is not noticed as it would be if the image was not cropped. At this time, I believe that the best way to solve this issue is to either change the way the image is cropped in the preprocessing function or add more computer vision applications that can detect the radius of the turn and then use this to factor into the steering ability. The last solution could in the long term provide overall better self-driving performance due to this detection of the lane (or path) being what the steering is based upon with the DNN's main purpose to identify when to autobrake or stop at indicators such as stop signs or traffic lights. If this was accomplished this would transition the auto-braking from only being based on the HC-SR04 sensors to the addition of braking prediction based on a camera data visual perspective. After this, I decided to test the accuracy of the auto-braking function accuracy with the newly 3D-printed 15-degree sensor wedge to see if the distance at which it stops is still as desired. To test this, I had a human stand in front of the vehicle and while in the RC mode, I drove the power wheels towards them, once the auto-braking stopped the power wheels I then measured the distance between the front bumper and the human. On average the power wheels stopped 34 inches in front of the human when the safety zone was set to four feet or 48 inches. At first, this was believed to be due to the sensor not measuring straight in front of the power wheels due to the 15-degree wedge but after doing the math using trigonometry the hypotenuse is only 1.16 to 1.33 inches more than the "adjacent" side that was measured. One factor that was observed that went unnoticed was the power wheels skidding a few inches when the motors are set to straight zero. With these issues noted future developments in the future can now be made to improve the accuracy of this while making the distance skid less.

August 11, 2023 – Today I worked on the documentation portion as I have concluded that due to moving back to college the improvements desired will have to be postponed to iteration two of this project. The first thing I did was review the already written sections of the paper, during which I noticed the *Control and Vision Systems* portions seemed underwritten based on what all went into them. To fix this I decided to slightly modify the portion that mentioned both as a whole then following it added a separate mention to both the control and vision systems individually. After this I transferred all the modified code to my computer from the Jetson Nano and updated it in *Appendix D.* After this I decided to change the *Control & Vision Systems* section of the *Methods and Procedures* section to better mention the two as a separate system along with how they are used together. Following this I went through *Appendix D* and double-checked that all the programs were attached and if so that they were the latest versions. After this, I did the same for all the drawing files in *Appendix B.*

August 12, 2023 – Today I took and added the build photos to *Appendix E,* after this I modified the *Computer and Vision Systems* section of *Results & Discussion* to better cover the additional changes made to the *Methods & Procedures* section. Unfortunately, due to time restraints this was all I was able to accomplish.

August 13, 2023 – Today I got back into the documentation and reviewed the work done yesterday for the *Computer and Vision Systems* section of *Results and Discussion.* During which I realized I forgot to mention the kill switch feature that was added last minute. After adding this as I reread it one more time to ensure the new mention of this fit, I realized that some of the diagrams housed in *Appendix D* would fit as in text visuals based on what is being mentioned. After this I decided to move the diagram which shows the processes the *Implementation.py* program uses to under the section which mentions this along with the sequence diagram to under the section which introduces how the control system breaks down the command array data which is sent from the vision system.

August 14, 2023 – Today I wrote the *Data Collection Techniques* section for both the *Methods and Procedures* and the *Results and Discussion.* After this I went ahead and added any sources that were not yet added to the references page and replaced the placeholder with the correct citation for them. After this I decided to go ahead and start on the *Machine Learning Model Development* section for the *Methods and Procedures.* After writing the first paragraph of the section I decided to change having the "Methods to Enhance the Vehicle's Overall Autonomous Capabilities" in both the *Methods and Procedures* along with the *Results and Discussion* to having just one "chapter" titled *Future Enhancements*.

August 15, 2023 – Today I began writing the *Machine Learning Model Development* section of the *Methods and Procedures* section.

September 26, 2023 – Today I finished the *Machine Learning Model Development* section of the *Methods and Procedures*. After this I began writing the *Machine Learning Model Development* section for the *Results and Discussion* section.

November 16, 2023 – Today I finished the *Machine Learning Model Development* section of the *Results and Discussion* section. I also added the source to OpenAI's ChatGPT as it was used for a few ideas needed based situations throughout the project.

November 19, 2023 – Today I added an *Evaluation* section separate of the *Methods and Procedures* and *Results and Discussion*. This was done due to only having two possible paragraphs for this section and since the following section is *Future Enhancements* that ties into the results of the testing. I also completed the *Future Enhancements* section and started to update the formatting on a few pages to visually make them more appealing.

November 20, 2023 – Today I wrote the *Conclusion* and *Abstract*, after this I began grammatically checking the paper using Grammarly.

November 21, 2023 – Today I finished the grammar check and began reviewing the paper formats for finalization of the paper.

November 28, 2023 – Today I finished reviewing the paper and have determined it suitable to get reviewed by professors/engineers.

Appendix A: Datasheets

Join the Revolution and Bring the Power of AI to Millions of Devices

The NVIDIA® Jetson Nano™ Developer Kit delivers the compute performance to run modern AI workloads at unprecedented size, power, and cost. Developers, learners, and makers can now run AI frameworks and models for applications like image classification, object detection, segmentation, and speech processing.

The developer kit can be powered by micro-USB and comes with extensive I/Os, ranging from GPIO to CSI. This makes it simple for developers to connect a diverse set of new sensors to enable a variety of AI applications. It's incredibly power-efficient, consuming as little as 5 watts.

Jetson Nano is also supported by NVIDIA JetPack™, which includes a board support package (BSP), Linux OS, NVIDIA CUDA®, cuDNN, and TensorRT™ software libraries for deep learning, computer vision, GPU computing, multimedia processing, and much more. The software is even available using an easy-to-flash SD card image, making it fast and easy to get started.

The same JetPack SDK is used across the entire NVIDIA Jetson™ family of products and is fully compatible with NVIDIA's world-leading AI platform for training and deploying AI software. This proven software stack reduces complexity and overall effort for developers.

KEY FEATURES

Jetson Nano Module
- 128-Core NVIDIA Maxwell™ GPU
- Quad-Core ARM® A57 CPU
- 4 GB 64-Bit LPDDR4
- 10/100/1000BASE-T Ethernet

Power Options
- Micro-USB 5V 2A
- DC Power Adapter 5V 4A

I/O
- USB 3.0 Type A
- USB 2.0 Micro-B
- HDMI/DisplayPort
- M.2 Key E
- Gigabit Ethernet
- GPIOs, I^2C, I^2S, SPI, UART
- MIPI-CSI Camera Connector
- Fan Connector
- PoE Connector

Kit Contents
- NVIDIA Jetson Nano Module with Heatsink and Reference Carrier Board
- Quick Start Guide and Support Guide

Figure 10. Jetson Nano Developer Kit Datasheet Page 1 of 2.

NVIDIA JETSON NANO DEVELOPER KIT
TECHNICAL SPECIFICATIONS

DEVELOPER KIT

GPU	128-Core Maxwell
CPU	Quad-Core ARM A57 @ 1.43 GHz
Memory	4 GB 64-bit LPDDR4 25.6 GB/s
Storage	microSD (Not Included)
Video Encoder	4K @ 30 | 4x 1080p @ 30 | 9x 720p @ 30 (H.264/H.265)
Video Decoder	4K @ 60 | 2x 4K @ 30 | 8x 1080p @ 30 | 18x 720p @ 30| (H.264/H.265)
Camera	2x MIPI CSI-2 DPHY lanes
Connectivity	Gigabit Ethernet, M.2 Key E
Display	HDMI 2.0 and eDP 1.4
USB	4x USB 3.0, USB 2.0 Micro-B
Others	GPIO, I²C, I²S, SPI, UART
Mechanical	100 mm x 80 mm x 29 mm

*Please refer to NVIDIA documentation for what is currently supported.

NVIDIA

Figure 11. Jetson Nano Developer Kit Datasheet Page 2 of 2.

Arduino® Nano

Product Reference Manual
SKU: A000005

Description

Arduino® Nano is an intelligent development board designed for building faster prototypes with the smallest dimension. Arduino Nano being the oldest member of the Nano family, provides enough interfaces for your breadboard-friendly applications. At the heart of the board is **ATmega328 microcontroller** clocked at a frequency of 16 MHz featuring more or less the same functionalities as the Arduino Duemilanove. The board offers 22 digital input/output pins, 8 analog pins, and a mini-USB port.

Target Areas

Maker, Security, Environmental, Robotics and Control Systems

Figure 12. Arduino Nano Datasheet Page 1 of 13.

Features

- **ATmega328** Microcontroller
  - High-performance low-power 8-bit processor
  - Achieve up to 16 MIPS for 16 MHz clock frequency
  - 32 kB of which 2 KB used by bootloader
  - 2 kB internal SRAM
  - 1 kB EEPROM
  - 32 x 8 General Purpose Working Registers
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
- **Power**
  - Mini-B USB connection
  - 6-20V unregulated external power supply (pin 30)
  - 5V regulated external power supply (pin 27)
- **Sleep Modes**
  - Idle
  - ADC Noise Reduction
  - Power-save
  - Power-down
  - Standby
  - Extended Standby
- **I/O**
  - 22 Digital
  - 8 Analog
  - 6 PWM Output

Figure 13. Arduino Nano Datasheet Page 2 of 13.

5.1 Analog

Pin	Function	Type	Description
1	+3V3	Power	5V USB Power
2	A0	Analog	Analog input 0 /GPIO
3	A1	Analog	Analog input 1 /GPIO
4	A2	Analog	Analog input 2 /GPIO
5	A3	Analog	Analog input 3 /GPIO
6	A4	Analog	Analog input 4 /GPIO
7	A5	Analog	Analog input 5 /GPIO
8	A6	Analog	Analog input 6 /GPIO
9	A7	Analog	Analog input 7 /GPIO
10	+5V	Power	+5V Power Rail
11	Reset	Reset	Reset
12	GND	Power	Ground
12	VIN	Power	Voltage Input

5.2 Digital

Pin	Function	Type	Description
1	D1/TX1	Digital	Digital Input 1 /GPIO
2	D0/RX0	Digital	Digital Input 0 /GPIO
3	D2	Digital	Digital Input 2 /GPIO
4	D3	Digital	Digital Input 3 /GPIO
5	D4	Digital	Digital Input 4 /GPIO
6	D5	Digital	Digital Input 5 /GPIO
7	D6	Digital	Digital Input 6 /GPIO
8	D7	Digital	Digital Input 7 /GPIO
9	D8	Digital	Digital Input 8 /GPIO
10	D9	Digital	Digital Input 9 /GPIO
11	D10	Digital	Digital Input 10 /GPIO
12	D11	Digital	Digital Input 11 /GPIO
13	D12	Digital	Digital Input 12 /GPIO
14	D13	Digital	Digital Input 13 /GPIO
15	Reset	Reset	Reset
16	GND	Power	Ground

Figure 14. Arduino Nano Datasheet Page 9 of 13.

Lens			Color	Forward Voltage Min. (DC V)	Forward Voltage Max. (DC V)	IF (Forward Current) = 20mA		Viewing Angle
Size	Style	Type				Luminous Intensity (mcd)	Wavelength	
5mm	Round	White (Diffused)	White	3.0	3.2	2000-3000	6000K	120°
			Warm White	3.0	3.2	2000-3000	3500k	
			Red	2.0	2.2	1000-2000	620-625nm	
			Yellow	2.0	2.2	1000-2000	585-595nm	
			Green	3.0	3.2	4000-5000	520-525nm	
			Yellow-Green	2.0	2.2	500-600	570-575nm	
			Blue	3.0	3.2	5000-7000	460-470nm	
			Orange	2.0	2.2	1000-2000	602-610nm	
			Pink	3.0	3.2	500-600	/	

Figure 15. EDGELEC LED Datasheet.

5mm Clear Lens (IF=20mA)	Wavelength	Luminous intensity	Forward voltage	Viewing Angle
White	6000-9000K	12000-14000mcd	3.0-3.2V	30°
Red	620-625nm	2000-3000mcd	2.0-2.2V	30°
Green	515-525nm	15000-18000mcd	3.0-3.2V	30°
Blue	450-455nm	7000-8000mcd	3.0-3.2V	30°
Yellow	588-592nm	1500-2000mcd	2.0-2.2V	30°
Warm White	2800-3000K	14000-16000mcd	3.0-3.2V	30°
Yellow-green	570-575nm	500-700mcd	2.0-2.2V	30°
Orange	602-610nm	1500-2000mcd	2.0-2.2V	30°
Purple (UV)	395-400nm	300-400mcd	3.0-3.4V	30°
Pink	/	7000-8000mcd	3.0-3.2V	30°

Figure 16. CHANZON LED Datasheet.

Ultrasonic Ranging Module HC - SR04

Product features:

Ultrasonic ranging module HC - SR04 provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:

(1) Using IO trigger for at least 10us high level signal,

(2) The Module automatically sends eight 40 kHz and detect whether there is a
pulse signal back.

(3) IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic to returning.
Test distance = (high level time×velocity of sound (340M/S) / 2

Wire connecting direct as following:

- 5V Supply
- Trigger Pulse Input
- Echo Pulse Output
- 0V Ground

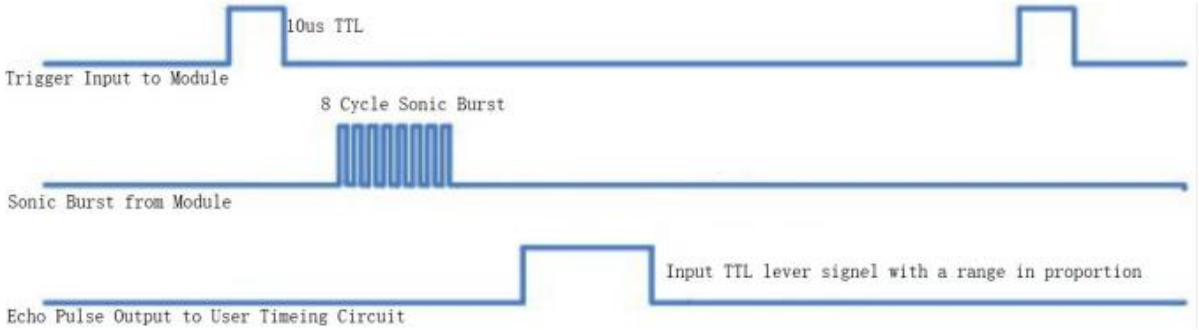
Electric Parameter

Working Voltage	DC 5 V
Working Current	15mA
Working Frequency	40Hz
Max Range	4m
Min Range	2cm
MeasuringAngle	15 degree
Trigger Input Signal	10uS TTL pulse
Echo Output Signal	Input TTL lever signal and the range in proportion
Dimension	45*20*15mm

Figure 17. HC-SR04 Ultrasonic Module Datasheet Page 1 of 2.

Timing diagram

The Timing diagram is shown below. You only need to supply a short 10uS pulse to the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. The Echo is a distance object that is pulse width and the range in proportion .You can calculate the range through the time interval between sending trigger signal and
receiving echo signal. Formula: uS / 58 = centimeters or uS / 148 =inch; or: the range = high level time * velocity (340M/S) / 2; we suggest to use over 60ms measurement cycle, in order to prevent trigger signal to the echo signal.



Attention:

●The module is not suggested to connect directly to electric, if connected electric, the GND terminal should be connected the module first, otherwise, it will affect the normal work of the module.
● When tested objects, the range of area is not less than 0.5 square meters and the plane requests as smooth as possible, otherwise ,it will affect the results of measuring.

Figure 18. HC-SR04 Ultrasonic Module Datasheet Page 2 of 2.

Product Model: FS-iA6B

PWM channel: 6

Wireless frequency: 2.4GHz

Wireless protocol: AFHDS 2A

Range: 500 ~ 1500m (in the air)

Antenna type: Dual copper tube antenna (150mm * 2)

Power: 4.0-8.4V

RSSI: Supported

Data port: PWM / PPM / i.bus / s.bus

Temperature range: -10 ℃ — + 60 ℃

Humidity range: 20% -95%

Online Update: Yes

Dimensions: 47 * 26.2 * 15 mm

Weight: 10g

Certification: CE, RCM, FCC ID: N4ZFLYSKYIA10

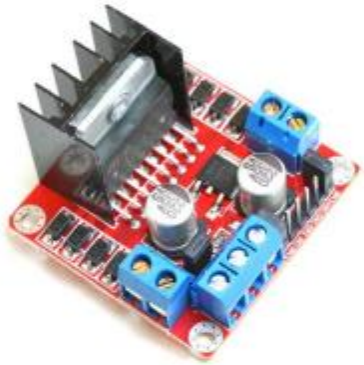
Figure 19. FlySky iA6B Datasheet.

HT **Handson Technology**

User Guide

L298N Dual H-Bridge Motor Driver

This dual bidirectional motor driver, is based on the very popular L298 Dual H-Bridge Motor Driver Integrated Circuit. The circuit will allow you to easily and independently control two motors of up to 2A each in both directions.It is ideal for robotic applications and well suited for connection to a microcontroller requiring just a couple of control lines per motor. It can also be interfaced with simple manual switches, TTL logic gates, relays, etc. This board equipped with power LED indicators, on-board +5V regulator and protection diodes.



SKU: MDU-1049

Brief Data:

- Input Voltage: 3.2V~40Vdc.
- Driver: L298N Dual H Bridge DC Motor Driver
- Power Supply: DC 5 V - 35 V
- Peak current: 2 Amp
- Operating current range: 0 ~ 36mA
- Control signal input voltage range :
- Low: -0.3V ⩽ Vin ⩽ 1.5V.
- High: 2.3V ⩽ Vin ⩽ Vss.
- Enable signal input voltage range :
  - Low: -0.3 ⩽ Vin ⩽ 1.5V (control signal is invalid).
  - High: 2.3V ⩽ Vin ⩽ Vss (control signal active).
- Maximum power consumption: 20W (when the temperature T = 75 ℃).
- Storage temperature: -25 ℃ ~ +130 ℃.
- On-board +5V regulated Output supply (supply to controller board i.e. Arduino).
- Size: 3.4cm x 4.3cm x 2.7cm

Figure 20. L298N Module Datasheet Page 1 of 7.

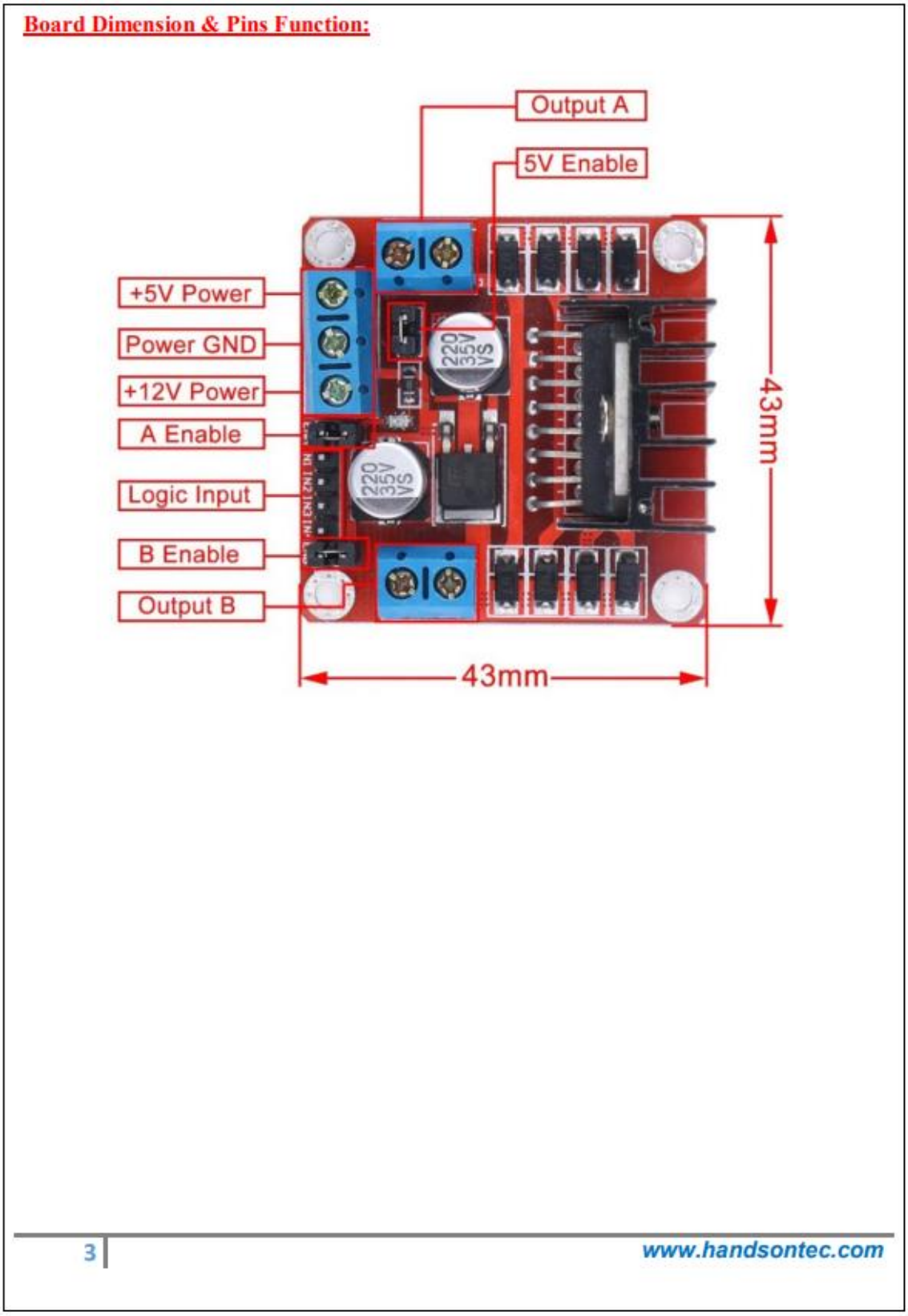


Figure 21. L298N Module Datasheet Page 3 of 7.

Arduino® MEGA 2560 Rev3

Product Reference Manual
SKU: A000067

Description

Arduino® Mega 2560 is an exemplary development board dedicated for building extensive applications as compared to other maker boards by Arduino. The board accommodates the ATmega2560 microcontroller, which operates at a frequency of 16 MHz. The board contains 54 digital input/output pins, 16 analog inputs, 4 UARTs (hardware serial ports), a USB connection, a power jack, an ICSP header, and a reset button.

Target Areas

3D Printing, Robotics, Maker

Figure 22. Arduino Mega 2560 Datasheet Page 1 of 17.

Features

- **ATmega2560 Processor**
  - Up to 16 MIPS Throughput at 16MHz
  - 256k bytes (of which 8k is used for the bootloader)
  - 4k bytes EEPROM
  - 8k bytes Internal SRAM
  - 32 × 8 General Purpose Working Registers
  - Real Time Counter with Separate Oscillator
  - Four 8-bit PWM Channels
  - Four Programmable Serial USART
  - Controller/Peripheral SPI Serial Interface
- **ATmega16U2**
  - Up to 16 MIPS Throughput at 16 MHz
  - 16k bytes ISP Flash Memory
  - 512 bytes EEPROM
  - 512 bytes SRAM
  - USART with SPI master only mode and hardware flow control (RTS/CTS)
  - Master/Slave SPI Serial Interface
- **Sleep Modes**
  - Idle
  - ADC Noise Reduction
  - Power-save
  - Power-down
  - Standby
  - Extended Standby
- **Power**
  - USB Connection
  - External AC/DC Adapter
- **I/O**
  - 54 Digital
  - 16 Analog
  - 15 PWM Output

Figure 23. Arduino Mega 2560 Datasheet Page 2 of 17.

5.1 Analog

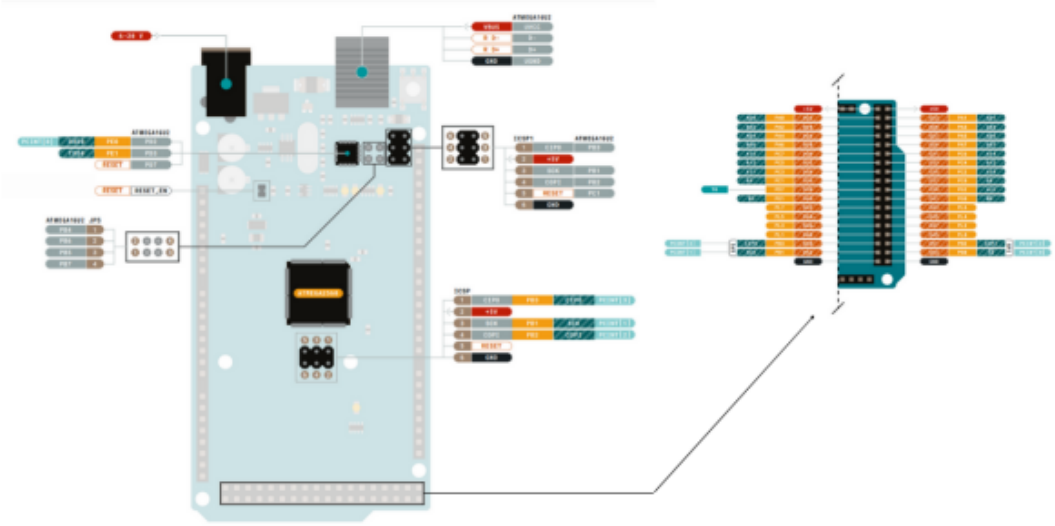
Pin	Function	Type	Description
1	NC	NC	Not Connected
2	IOREF	IOREF	Reference for digital logic V - connected to 5V
3	Reset	Reset	Reset
4	+3V3	Power	+3V3 Power Rail
5	+5V	Power	+5V Power Rail
6	GND	Power	Ground
7	GND	Power	Ground
8	VIN	Power	Voltage Input
9	A0	Analog	Analog input 0 /GPIO
10	A1	Analog	Analog input 1 /GPIO
11	A2	Analog	Analog input 2 /GPIO
12	A3	Analog	Analog input 3 /GPIO
13	A4	Analog	Analog input 4 /GPIO
14	A5	Analog	Analog input 5 /GPIO
15	A6	Analog	Analog input 6 /GPIO
16	A7	Analog	Analog input 7 /GPIO
17	A8	Analog	Analog input 8 /GPIO
18	A9	Analog	Analog input 9 /GPIO
19	A10	Analog	Analog input 10 /GPIO
20	A11	Analog	Analog input 11 /GPIO
21	A12	Analog	Analog input 12 /GPIO
22	A13	Analog	Analog input 13 /GPIO
23	A14	Analog	Analog input 14 /GPIO
24	A15	Analog	Analog input 15 /GPIO

5.2 Digital

Pin	Function	Type	Description
1	D21/SCL	Digital Input/I2C	Digital input 21/I2C Dataline
2	D20/SDA	Digital Input/I2C	Digital input 20/I2C Dataline
3	AREF	Digital	Analog Reference Voltage
4	GND	Power	Ground
5	D13	Digital/GPIO	Digital input 13/GPIO
6	D12	Digital/GPIO	Digital input 12/GPIO
7	D11	Digital/GPIO	Digital input 11/GPIO
8	D10	Digital/GPIO	Digital input 10/GPIO
9	D9	Digital/GPIO	Digital input 9/GPIO
10	D8	Digital/GPIO	Digital input 8/GPIO
11	D7	Digital/GPIO	Digital input 7/GPIO
12	D6	Digital/GPIO	Digital input 6/GPIO
13	D5	Digital/GPIO	Digital input 5/GPIO
14	D4	Digital/GPIO	Digital input 4/GPIO

Figure 24. Arduino Mega 2560 Datasheet Page 10 of 17.

Pin	Function	Type	Description
15	D3	Digital/GPIO	Digital input 3/GPIO
16	D2	Digital/GPIO	Digital input 2/GPIO
17	D1/TX0	Digital/GPIO	Digital input 1 /GPIO
18	D0/Tx1	Digital/GPIO	Digital input 0 /GPIO
19	D14	Digital/GPIO	Digital input 14 /GPIO
20	D15	Digital/GPIO	Digital input 15 /GPIO
21	D16	Digital/GPIO	Digital input 16 /GPIO
22	D17	Digital/GPIO	Digital input 17 /GPIO
23	D18	Digital/GPIO	Digital input 18 /GPIO
24	D19	Digital/GPIO	Digital input 19 /GPIO
25	D20	Digital/GPIO	Digital input 20 /GPIO
26	D21	Digital/GPIO	Digital input 21 /GPIO



Arduino Mega Pinout

Figure 25. Arduino Mega 2560 Datasheet Page 11 of 17.

5.3 ATMEGA16U2 JP5

Pin	Function	Type	Description
1	PB4	Internal	Serial Wire Debug
2	PB6	Internal	Serial Wire Debug
3	PB5	Internal	Serial Wire Debug
4	PB7	Internal	Serial Wire Debug

5.4 ATMEGA16U2 ICSP1

Pin	Function	Type	Description
1	CIPO	Internal	Controller In Peripheral Out
2	+5V	Internal	Power Supply of 5V
3	SCK	Internal	Serial Clock
4	COPI	Internal	Controller Out Peripheral In
5	RESET	Internal	Reset
6	GND	Internal	Ground

5.5 Digital Pins D22 - D53 LHS

Pin	Function	Type	Description
1	+5V	Power	Power Supply of 5V
2	D22	Digital	Digital input 22/GPIO
3	D24	Digital	Digital input 24/GPIO
4	D26	Digital	Digital input 26/GPIO
5	D28	Digital	Digital input 28/GPIO
6	D30	Digital	Digital input 30/GPIO
7	D32	Digital	Digital input 32/GPIO
8	D34	Digital	Digital input 34/GPIO
9	D36	Digital	Digital input 36/GPIO
10	D38	Digital	Digital input 38/GPIO
11	D40	Digital	Digital input 40/GPIO
12	D42	Digital	Digital input 42/GPIO
13	D44	Digital	Digital input 44/GPIO
14	D46	Digital	Digital input 46/GPIO
15	D48	Digital	Digital input 48/GPIO
16	D50	Digital	Digital input 50/GPIO
17	D52	Digital	Digital input 52/GPIO
18	GND	Power	Ground

Figure 26. Arduino Mega 2560 Datasheet Page 12 of 17.

5.6 Digital Pins D22 - D53 RHS

Pin	Function	Type	Description
1	+5V	Power	Power Supply of 5V
2	D23	Digital	Digital input 23/GPIO
3	D25	Digital	Digital input 25/GPIO
4	D27	Digital	Digital input 27/GPIO
5	D29	Digital	Digital input 29/GPIO
6	D31	Digital	Digital input 31/GPIO
7	D33	Digital	Digital input 33/GPIO
8	D35	Digital	Digital input 35/GPIO
9	D37	Digital	Digital input 37/GPIO
10	D39	Digital	Digital input 39/GPIO
11	D41	Digital	Digital input 41/GPIO
12	D43	Digital	Digital input 43/GPIO
13	D45	Digital	Digital input 45/GPIO
14	D47	Digital	Digital input 47/GPIO
15	D49	Digital	Digital input 49/GPIO
16	D51	Digital	Digital input 51/GPIO
17	D53	Digital	Digital input 53/GPIO
18	GND	Power	Ground

6 Mechanical Information

6.1 Board Outline

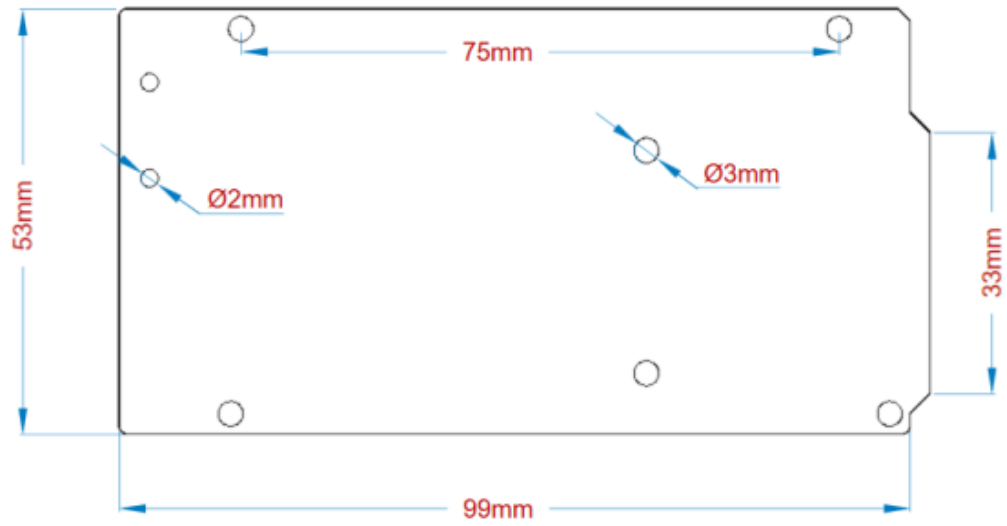


Figure 27. Arduino Mega 2560 Datasheet Page 13 of 17.

HT **Handson Technology**

User Guide

BTS7960 High Current 43A H-Bridge Motor Driver

The BTS7960 is a fully integrated high current H bridge module for motor drive applications. Interfacing to a microcontroller is made easy by the integrated driver IC which features logic level inputs, diagnosis with current sense, slew rate adjustment, dead time generation and protection against overtemperature, overvoltage, undervoltage, overcurrent and short circuit. The BTS7960 provides a cost optimized solution for protected high current PWM motor drives with very low board space consumption.

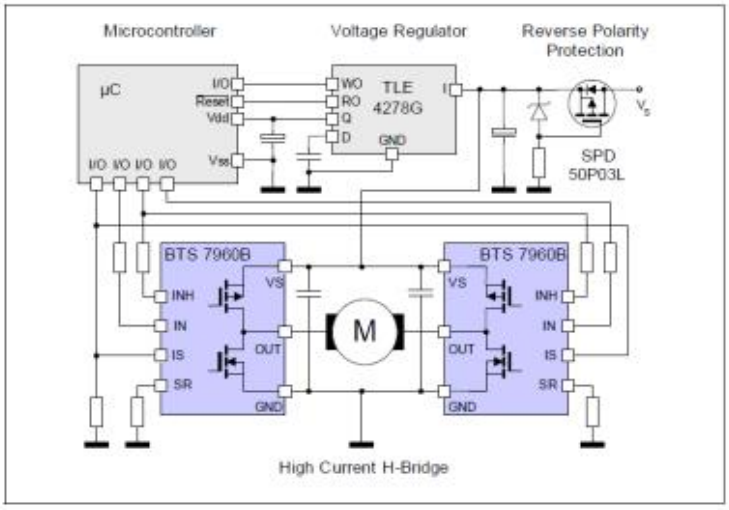
SKU: DRV-1012

Brief Data:

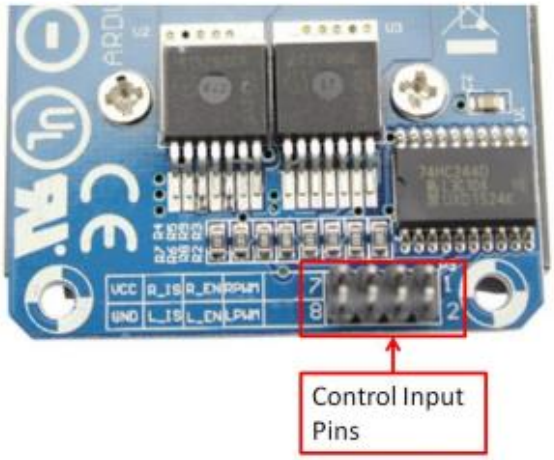
- Input Voltage: 6 ~ 27Vdc.
- Driver: Dual BTS7960 H Bridge Configuration.
- Peak current: 43-Amp.
- PWM capability of up to 25 kHz.
- Control Input Level: 3.3~5V.
- Control Mode: PWM or level
- Working Duty Cycle: 0 ~100%.
- Over-voltage Lock Out.
- Under-voltage Shut Down.
- Board Size (LxWxH): 50mm x 50mm x 43mm.
- Weight: ~66g.

Figure 28. BTS7960 Motor Controller Datasheet Page 1 of 9.

Schematic Diagram:



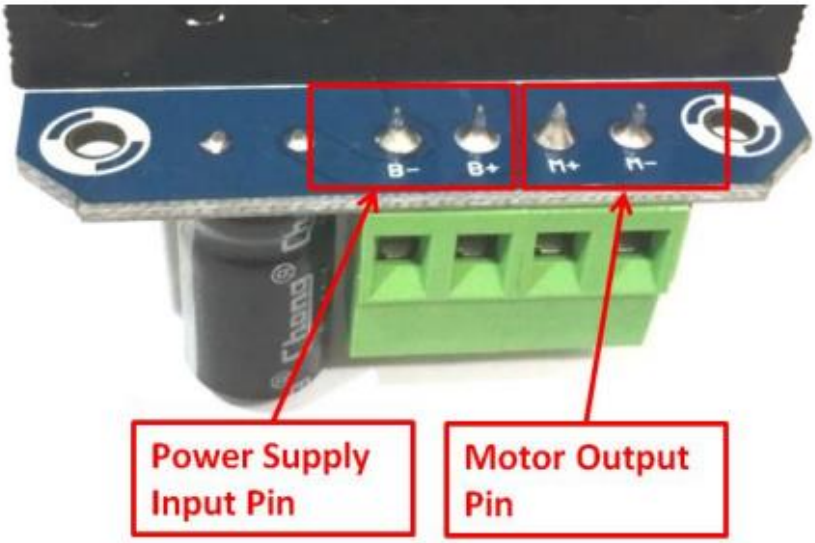
Control Input Pin Function:



Pin No	Function	Description
1	RPWM	Forward Level or PWM signal, Active High
2	LPWM	Reverse Level or PWM signal, Active High
3	R_EN	Forward Drive Enable Input, Active High/ Low Disable
4	L_EN	Reverse Drive Enable Input, Active High/Low Disable
5	R_IS	Forward Drive, Side current alarm output
6	L_IS	Reverse Drive, Side current alarm output
7	Vcc	+5V Power Supply microcontroller
8	Gnd	Ground Power Supply microcontroller

Figure 29. BTS7960 Motor Controller Datasheet Page 3 of 9.

Motor Power Supply & Output Pin Assignment:



Pin No	Function	Description
1	B+	Positive Motor Power Supply. 6 ~ 27VDC
2	B-	Negative Motor Power Supply. Ground
3	M+	Motor Output +
4	M-	Motor Output -

Controlling DC Motor with BTS7960 Using Arduino:

Below is the circuit connection using BTS7960 high power driver to control one DC motor with Arduino board. The potentiometer allows the control of motor speed and rotation direction of the motor.

Figure 30. BTS7960 Motor Controller Datasheet Page 4 of 9.

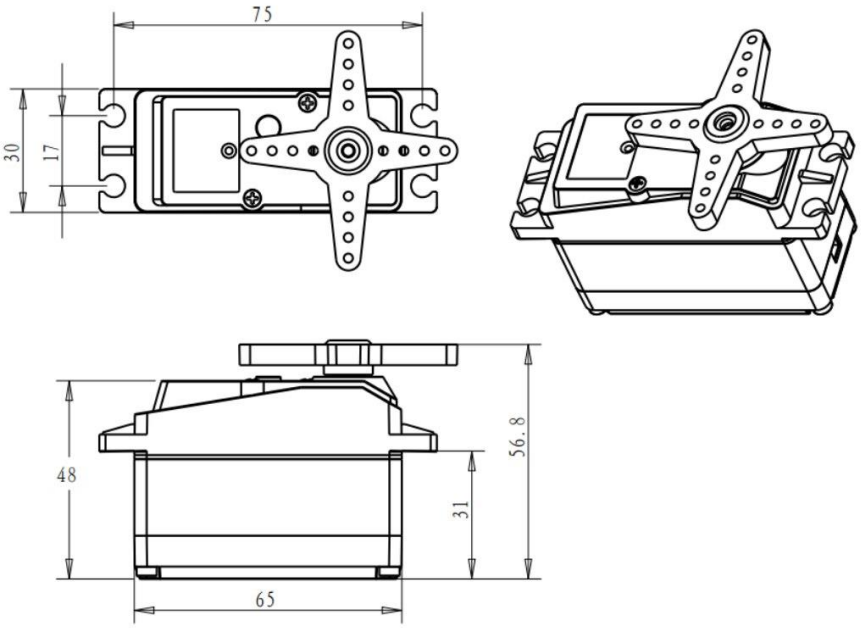
Sketch Listing:

Upload the following sketch to Arduino board. Try to turn the potentiometer clock-wise and anti-clock-wise and observe how the motor turn.

```
/*========================================================================
//  Author      : Handson Technology
//  Project     : BTD7960 Motor Control Board driven by Arduino.
//  Description : Speed and direction controlled by a potentiometer attached
//      to analog input A0. One side pin of the potentiometer (either one) to
//      ground; the other side pin to +5V
//  Source-Code : BTS7960.ino
//  Program: Control DC motors using BTS7960 H Bridge Driver.
//========================================================================
//  Connection to the BTS7960 board:
//  BTS7960 Pin 1 (RPWM) to Arduino pin 5(PWM)
//  BTS7960 Pin 2 (LPWM) to Arduino pin 6(PWM)
//  BTS7960 Pin 3 (R_EN), 4 (L_EN), 7 (VCC) to Arduino 5V pin
//  BTS7960 Pin 8 (GND) to Arduino GND
//  BTS7960 Pin 5 (R IS) and 6 (L IS) not connected
*/

int SENSOR_PIN = 0; // center pin of the potentiometer

int RPWM_Output = 5; // Arduino PWM output pin 5; connect to IBT-2 pin 1 (RPWM)
int LPWM_Output = 6; // Arduino PWM output pin 6; connect to IBT-2 pin 2 (LPWM)

void setup()
{
  pinMode(RPWM_Output, OUTPUT);
  pinMode(LPWM_Output, OUTPUT);
}

void loop()
{
  int sensorValue = analogRead(SENSOR_PIN);

  // sensor value is in the range 0 to 1023
  // the lower half of it we use for reverse rotation; the upper half for forward
rotation
  if (sensorValue < 512)
  {
    // reverse rotation
    int reversePWM = -(sensorValue - 511) / 2;
    analogWrite(LPWM_Output, 0);
    analogWrite(RPWM_Output, reversePWM);
  }
  else
  {
    // forward rotation
    int forwardPWM = (sensorValue - 512) / 2;
    analogWrite(LPWM_Output, forwardPWM);
    analogWrite(RPWM_Output, 0);
  }
}
```


Figure 31. BTS7960 Motor Controller Datasheet Page 6 of 9.

产品型号（Product Name）：DS5180（RED）
产品描述（Product Description）：7.4V 80kg RC Digital Servo
产 品 图（Drawing）



1. 使用环境条件 Apply Environmental Condition

No.	Item	Specification
1-1	存储温度 Storage Temperature Range	-30℃～80℃
1-2	运行温度 Operating Temperature Range	-15℃～70℃
1-3	工作电压范围 Operating Voltage Range	6-8.4V

2. 机械特性 Mechanical Specification

No.	Item	Specification
2-1	尺寸 Size	65*30*48mm
2-2	重量 Weight	158g
2-3	齿轮比 Gear ratio	357
2-4	轴承 Bearing	Double bearing
2-5	舵机线 Connector wire	300±5mm
2-6	马达 Motor	3-pole(s)
2-7	防水性能 Waterproof performance	IP67

Figure 32. DS5180 Servo Datasheet.

LM2596 DC-DC Adjustable PSU Module

LM2596 DC to DC step down regulator, adjustable +1.23 to 35vdc output, 2A. Ideal for battery operated projects requiring a regulated powersupply.

Specifications

Regulator Type:	Step Down (Non Isolated input to Output)
Input Voltage:	+4 to 40vdc
Output Voltage:	+1.23 to 35vdc
Output Current:	2A rated, (3A maximum with heatsink)
Efficiency:	Up to 92% (when output voltage is set high)
Switching Frequency:	150kHz
Dropout Voltage:	2vdc minimum
Protection:	Short circuit current limiting
Load Regulation:	+/- 0.5%
Voltage Regulation:	+/- 2.5%
Temperature:	-40 to +85 deg C (output power less than 10Watts)
Board Size:	43.6mm L x 21mm W x 14mm H
Data Sheet:	National LM2596

Figure 33. LM2596 DC-DC Adjustable PSU Module Datasheet Page 1 of 2.

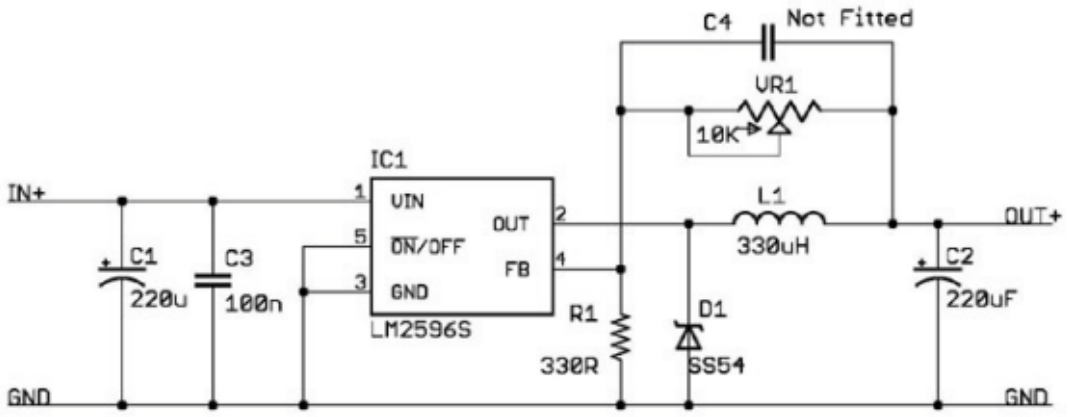


Figure 34. LM2596 DC-DC Adjustable PSU Module Datasheet Page 2 of 2.

SONGLE RELAY

松乐继电器 ® SONGLE RELAY	RELAY ISO9002	SRD

1. MAIN FEATURES

- Switching capacity available by 10A in spite of small size design for highdensity P.C. board mounting technique.
- UL,CUL,TUV recognized.
- Selection of plastic material for high temperature and better chemical solution performance.
- Sealed types available.
- Simple relay magnetic circuit to meet low cost of mass production.

2. APPLICATIONS

- Domestic appliance, office machine, audio, equipment, automobile, etc.
  (Remote control TV receiver, monitor display, audio equipment high rushing current use application.)

3. ORDERING INFORMATION

SRD	XX VDC	S	L	C
Model of relay	Nominal coil voltage	Structure	Coil sensitivity	Contact form
SRD	03、05、06、09、12、24、48VDC	S:Sealed type	L:0.36W	A:1 form A
				B:1 form B
		F:Flux free type	D:0.45W	C:1 form C

4. RATING

CCC	FILE NUMBER:CH0052885-2000	7A/240VDC
CCC	FILE NUMBER:CH0036746-99	10A/250VDC
UL /CUL	FILE NUMBER: E167996	10A/125VAC 28VDC
TUV	FILE NUMBER: R9933789	10A/240VAC 28VDC

5. DIMENSION(unit:mm) DRILLING(unit:mm) WIRING DIAGRAM

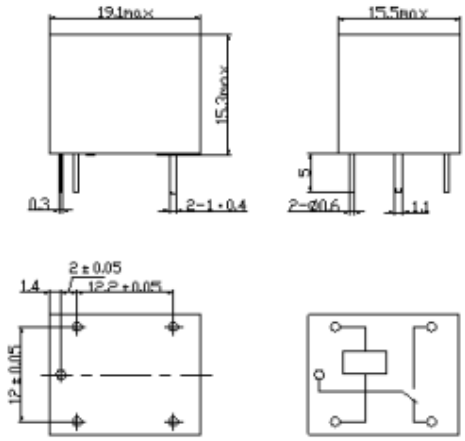


Figure 35. Songle SRD-12VDC-SL-C Relay Datasheet Page 1 of 2.

6. COIL DATA CHART (AT20°C)

Coil Sensitivity	Coil Voltage Code	Nominal Voltage (VDC)	Nominal Current (mA)	Coil Resistance (Ω) ±10%	Power Consumption (W)	Pull-In Voltage (VDC)	Drop-Out Voltage (VDC)	Max-Allowable Voltage (VDC)
SRD (High Sensitivity)	03	03	120	25	abt. 0.36W	75%Max.	10% Min.	120%
	05	05	71.4	70				
	06	06	60	100				
	09	09	40	225				
	12	12	30	400				
	24	24	15	1600				
	48	48	7.5	6400				
SRD (Standard)	03	03	150	20	abt. 0.45W	75% Max.	10% Min.	110%
	05	05	89.3	55				
	06	06	75	80				
	09	09	50	180				
	12	12	37.5	320				
	24	24	18.7	1280				
	48	48	10	4500	abt. 0.51W			

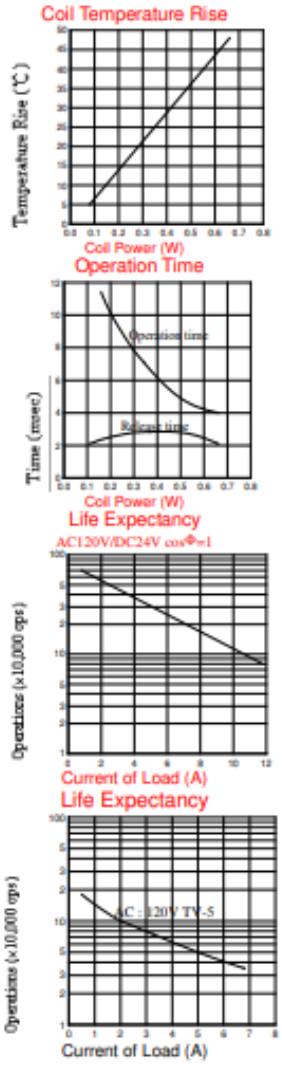
7. CONTACT RATING

Item \ Type	SRD FORM C	SRD FORM A
Contact Capacity Resistive Load (cosΦ=1)	7A 28VDC 10A 125VAC 7A 240VAC	10A 28VDC 10A 240VAC
Inductive Load (cosΦ=0.4 L/R=7msec)	3A 120VAC 3A 28VDC	5A 120VAC 5A 28VDC
Max. Allowable Voltage	250VAC/110VDC	250VAC/110VDC
Max. Allowable Power Force	800VAC/240W	1200VA/300W
Contact Material	AgCdO	AgCdO

8. PERFORMANCE (at initial value)

Item \ Type	SRD
Contact Resistance	100mΩ Max.
Operation Time	10msec Max.
Release Time	5msec Max.
Dielectric Strength Between coil & contact Between contacts	 1500VAC 50/60HZ (1 minute) 1000VAC 50/60HZ (1 minute)
Insulation Resistance	100 MΩ Min. (500VDC)
Max. ON/OFF Switching Mechanically Electrically	 300 operation/min 30 operation/min
Ambient Temperature	-25°C to +70°C
Operating Humidity	45 to 85% RH
Vibration Endurance Error Operation	 10 to 55Hz Double Amplitude 1.5mm 10 to 55Hz Double Amplitude 1.5mm
Shock Endurance Error Operation	 100G Min. 10G Min.
Life Expectancy Mechanically Electrically	 10^7 operations. Min. (no load) 10^5 operations. Min. (at rated coil voltage)
Weight	abt. 10grs.

9.REFERENCE DATA



2

Figure 36. Songle SRD-12VDC-SL-C Relay Datasheet Page 2 of 2.

FAIRCHILD
SEMICONDUCTOR®

October 2005

BSS138

BSS138

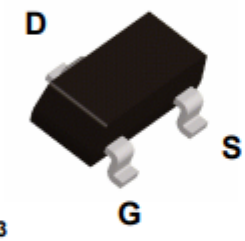
N-Channel Logic Level Enhancement Mode Field Effect Transistor

General Description

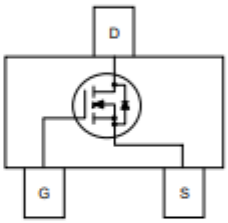
These N-Channel enhancement mode field effect transistors are produced using Fairchild's proprietary, high cell density, DMOS technology. These products have been designed to minimize on-state resistance while provide rugged, reliable, and fast switching performance.These products are particularly suited for low voltage, low current applications such as small servo motor control, power MOSFET gate drivers, and other switching applications.

Features

- 0.22 A, 50 V. $R_{DS(ON)}$ = 3.5Ω @ V_{GS} = 10 V
  $R_{DS(ON)}$ = 6.0Ω @ V_{GS} = 4.5 V
- High density cell design for extremely low $R_{DS(ON)}$
- Rugged and Reliable
- Compact industry standard SOT-23 surface mount package



SOT-23



Absolute Maximum Ratings

T_A=25°C unless otherwise noted

Symbol	Parameter		Ratings	Units
V_{DSS}	Drain-Source Voltage		50	V
V_{GSS}	Gate-Source Voltage		±20	V
I_D	Drain Current – Continuous	(Note 1)	0.22	A
	– Pulsed		0.88	
P_D	Maximum Power Dissipation	(Note 1)	0.36	W
	Derate Above 25°C		2.8	mW/°C
T_J, T_{STG}	Operating and Storage Junction Temperature Range		–55 to +150	°C
T_L	Maximum Lead Temperature for Soldering Purposes, 1/16" from Case for 10 Seconds		300	°C

Thermal Characteristics

$R_{\theta JA}$	Thermal Resistance, Junction-to-Ambient	(Note 1)	350	°C/W

Package Marking and Ordering Information

Device Marking	Device	Reel Size	Tape width	Quantity
SS	BSS138	7"	8mm	3000 units

Figure 37. BSS138 N-Channel Enhancement Datasheet Page 1 of 5.

BSS138

Electrical Characteristics

T_A = 25°C unless otherwise noted

Symbol	Parameter	Test Conditions	Min	Typ	Max	Units
Off Characteristics						
BV_{DSS}	Drain–Source Breakdown Voltage	V_{GS} = 0 V, I_D = 250 µA	50			V
$\frac{\Delta BV_{DSS}}{\Delta T_J}$	Breakdown Voltage Temperature Coefficient	I_D = 250 µA,Referenced to 25°C		72		mV/°C
I_{DSS}	Zero Gate Voltage Drain Current	V_{DS} = 50 V, V_{GS} = 0 V			0.5	µA
		V_{DS} = 50 V, V_{GS} = 0 V T_J = 125°C			5	µA
		V_{DS} = 30 V, V_{GS} = 0 V			100	nA
I_{GSS}	Gate–Body Leakage.	V_{GS} = ±20 V, V_{DS} = 0 V			±100	nA
On Characteristics (Note 2)						
$V_{GS(th)}$	Gate Threshold Voltage	V_{DS} = V_{GS}, I_D = 1 mA	0.8	1.3	1.5	V
$\frac{\Delta V_{GS(th)}}{\Delta T_J}$	Gate Threshold Voltage Temperature Coefficient	I_D = 1 mA,Referenced to 25°C		–2		mV/°C
$R_{DS(on)}$	Static Drain–Source On–Resistance	V_{GS} = 10 V, I_D = 0.22 A		0.7	3.5	Ω
		V_{GS} = 4.5 V, I_D = 0.22 A		1.0	6.0	
		V_{GS} = 10 V, I_D = 0.22 A, T_J = 125°C		1.1	5.8	
$I_{D(on)}$	On–State Drain Current	V_{GS} = 10 V, V_{DS} = 5 V	0.2			A
g_{FS}	Forward Transconductance	V_{DS} = 10V, I_D = 0.22 A	0.12	0.5		S
Dynamic Characteristics						
C_{iss}	Input Capacitance	V_{DS} = 25 V, V_{GS} = 0 V, f = 1.0 MHz		27		pF
C_{oss}	Output Capacitance			13		pF
C_{rss}	Reverse Transfer Capacitance			6		pF
R_G	Gate Resistance	V_{GS} = 15 mV, f = 1.0 MHz		9		Ω
Switching Characteristics (Note 2)						
$t_{d(on)}$	Turn–On Delay Time	V_{DD} = 30 V, I_D = 0.29 A, V_{GS} = 10 V, R_{GEN} = 6 Ω		2.5	5	ns
t_r	Turn–On Rise Time			9	18	ns
$t_{d(off)}$	Turn–Off Delay Time			20	36	ns
t_f	Turn–Off Fall Time			7	14	ns
Q_g	Total Gate Charge	V_{DS} = 25 V, I_D = 0.22 A, V_{GS} = 10 V		1.7	2.4	nC
Q_{gs}	Gate–Source Charge			0.1		nC
Q_{gd}	Gate–Drain Charge			0.4		nC
Drain–Source Diode Characteristics and Maximum Ratings						
I_S	Maximum Continuous Drain–Source Diode Forward Current				0.22	A
V_{SD}	Drain–Source Diode Forward Voltage	V_{GS} = 0 V, I_S = 0.44 A (Note 2)		0.8	1.4	V

Notes:

1. $R_{\theta JA}$ is the sum of the junction-to-case and case-to-ambient thermal resistance where the case thermal reference is defined as the solder mounting surface of the drain pins. $R_{\theta JC}$ is guaranteed by design while $R_{\theta CA}$ is determined by the user's board design.

a) 350°C/W when mounted on a minimum pad..

Scale 1 : 1 on letter size paper

2. Pulse Test: Pulse Width ≤ 300 µs, Duty Cycle ≤ 2.0%

BSS138 Rev C(W)

Figure 38. BSS138 N-Channel Enhancement Datasheet Page 2 of 5.

SSM SUSUMU

Metal thin film chip resistors
(precision)

■RR series

Features

- Precision chip resistors excellent in resistance tolerance, TCR, frequency performance, noise characteristics, and linearity.

* This product is not currently recommended for use in new design systems. Production may be discontinued in the near future.

Applications

- Consumer electronics that requires precision resistors
- All purpose resistors in any area of electronics

Lead free | Halogen free | RoHS Compliance

Thin film surface mount resistors

RR series

◆Part numbering system

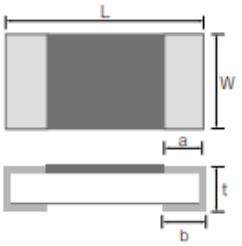
RR 0816 P - 102 - D - (M) - (***)

- Series code: RR
- Size: RR0306, RR0510, RR0816, RR1220,: 0816
- Temperature coefficient of resistance: P
- Nominal Resistance: RR0306, RR0510, RR0816, RR1220 E-24: 3 digit, RR0306, RR0510, RR0816, RR1220 E-96: 4 digit: 102
- Resistance tolerance: D
- Letter M is added for RR1220 E-96 series 4digit codes: (M)
- only given to 3 digit coders for RR0816 E-96 series: (***)

◆Electrical Specification

Type	Power ratings	Temperature coefficient of resistance (ppm/°C)	Resistance range(Ω) Resistance tolerance ±0.1% (B)	±0.5% (D)	±1% (F)	Maximum voltage	Resistance value series	Operating temperature	Packaging quantity
RR0306	1/20W	±25(P)	–	33≤R≤22k		15V	E-24	-55℃ ~ 125℃	5,000pcs
		±100(R)	–	–	10≤R≤30				
RR0510	1/16W	±25(P)	–	100≤R≤100k	–	25V	E-24, E-96		10,000pcs
		±100(R)		10≤R<100					
RR0816	1/16W	±25(P)	–	100≤R≤360k	–	75V			5,000pcs
		±50(Q)		10≤R<100					
RR1220	1/10W	±25(P)	–	100≤R≤1M		100V			
		±50(Q)	–	10≤R<100	–				

◆Dimensions



Type	Size (inch)	L	W	a	b	t
RR0306	0201	0.60±0.05	0.30±0.05	0.12±0.05	0.12±0.05	0.23±0.03
RR0510	0402	1.00±0.05	0.50±0.05	0.20±0.10	0.25±0.05	0.35±0.05
RR0816	0603	1.60±0.20	0.80±0.20	0.30±0.20	0.30±0.20	0.40±0.10
RR1220	0805	2.00±0.20	1.25±0.20	0.40±0.20	0.40±0.20	0.40±0.10

(unit : mm)

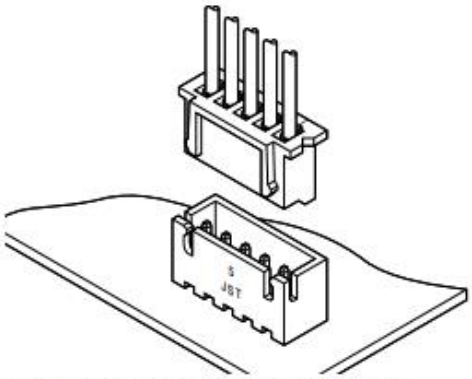
35

Figure 39. Susumu 10K Ohm 0.5% 0805 Resistor Data Sheet.

XH CONNECTOR

Emboss Tape Radial Tape

2.5 mm pitch/Disconnectable Crimp style connectors



The XH connector was developed based on the high reliability and versatility of our NH series connectors. The connector is very small with a mounting height of 9.8 mm.

- **Original folded beam contact**
- **Box-shaped shrouded header**
- **Header with a boss**
- **Interchangeability**
- **Conforming to the HA terminal**

Specifications

- Current rating: 3 A AC/DC (AWG #22)
- Voltage rating: 250 V AC/DC
- Temperature range: -25℃ to +85℃ (including temperature rise in applying electrical current)
- Contact resistance: Initial value/ 10 mΩ max. After environmental tests/ 20 mΩ max.
- Insulation resistance: 1,000 MΩ min.
- Withstanding voltage: 1,000 VAC/minute
- Applicable wire: AWG #30 to #22
- Applicable PC board thickness: 1.6 mm

* In using the products, refer to "Handling Precautions for Terminals and Connectors" described on our website (Technical documents of Product information page).
* RoHS2 compliance
* Dimensional unit: mm
* Contact JST for details.

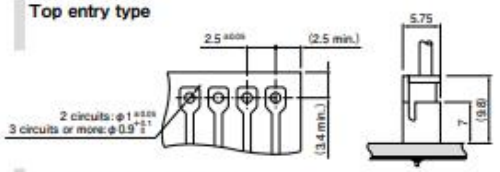
Standards

Recognized E60389
J50014297

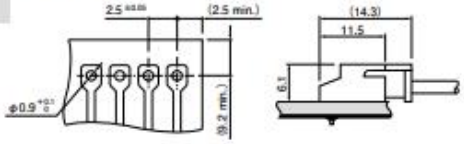
PC board layout and Assembly layout

⟨Through-hole type (viewed from soldening side)⟩

Top entry type



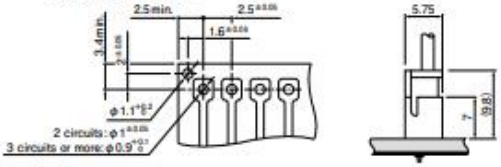
Side entry type



Top entry type with a boss

<Single circuit>

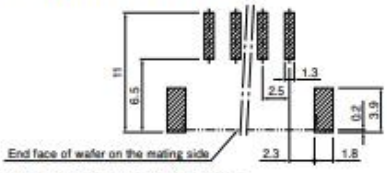
<2 circuits or more>



Note: 1. Tolerances are non-cumulative: ± 0.05 mm for all centers.
2. Hole dimensions differ according to the type of PC board and piercing method. If PC boards made of hard material are used, the hole dimensions should be larger. The dimensions above should serve as a guideline. Contact JST for details.

⟨SMT type (viewed from connector mounting side)⟩

Side entry type



Note: 1. Tolerances are non-cumulative: ± 0.05 mm for all centers.
2. The dimensions above should serve as a guideline. Contact JST for details.

Figure 40. JST Connector Data Sheet Page 1 of 6.

Appendix B: Drawing Files

Figure 41. Back Indicator Drawing Page 1 of 4.

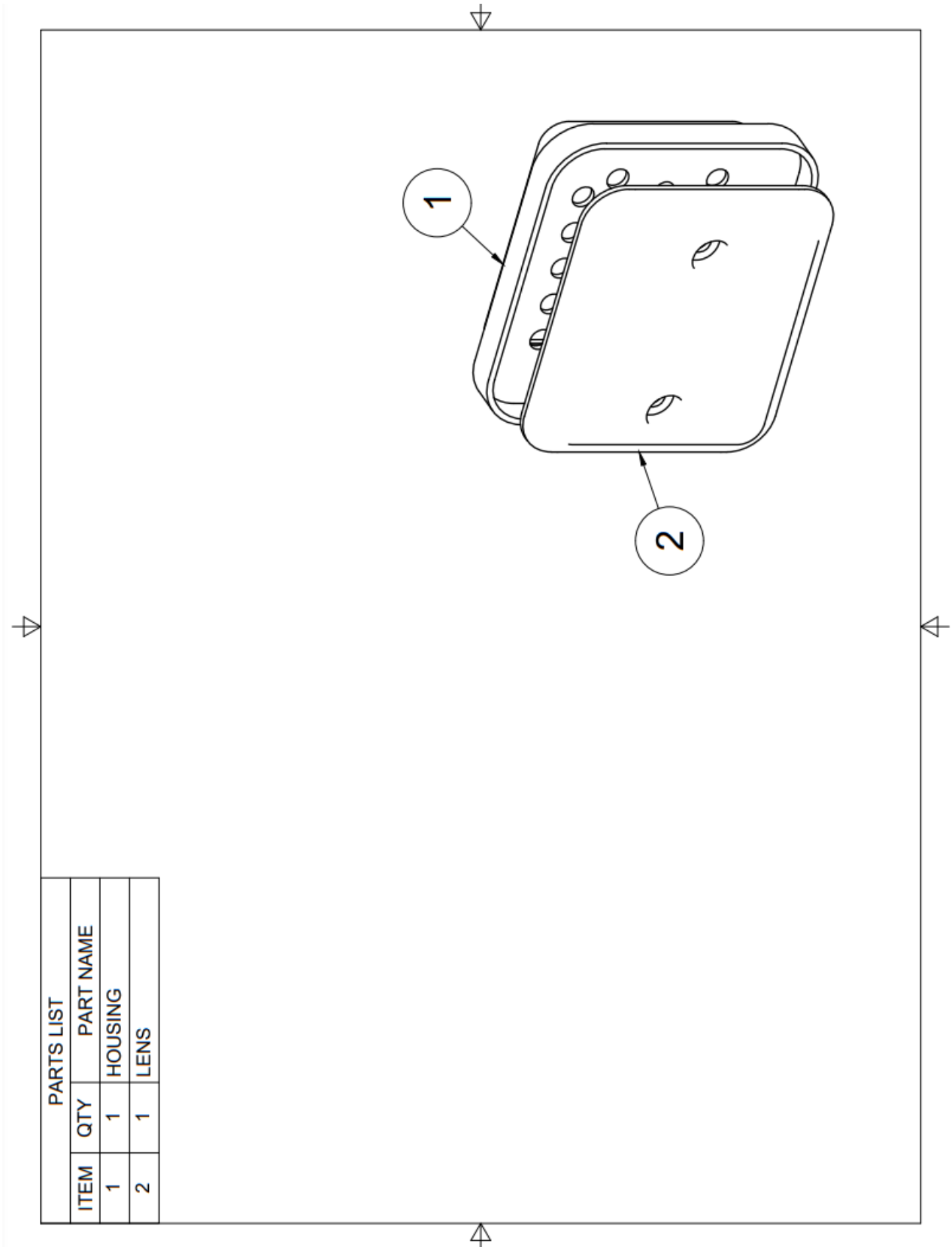


Figure 42. Back Indicator Drawing Page 2 of 4.

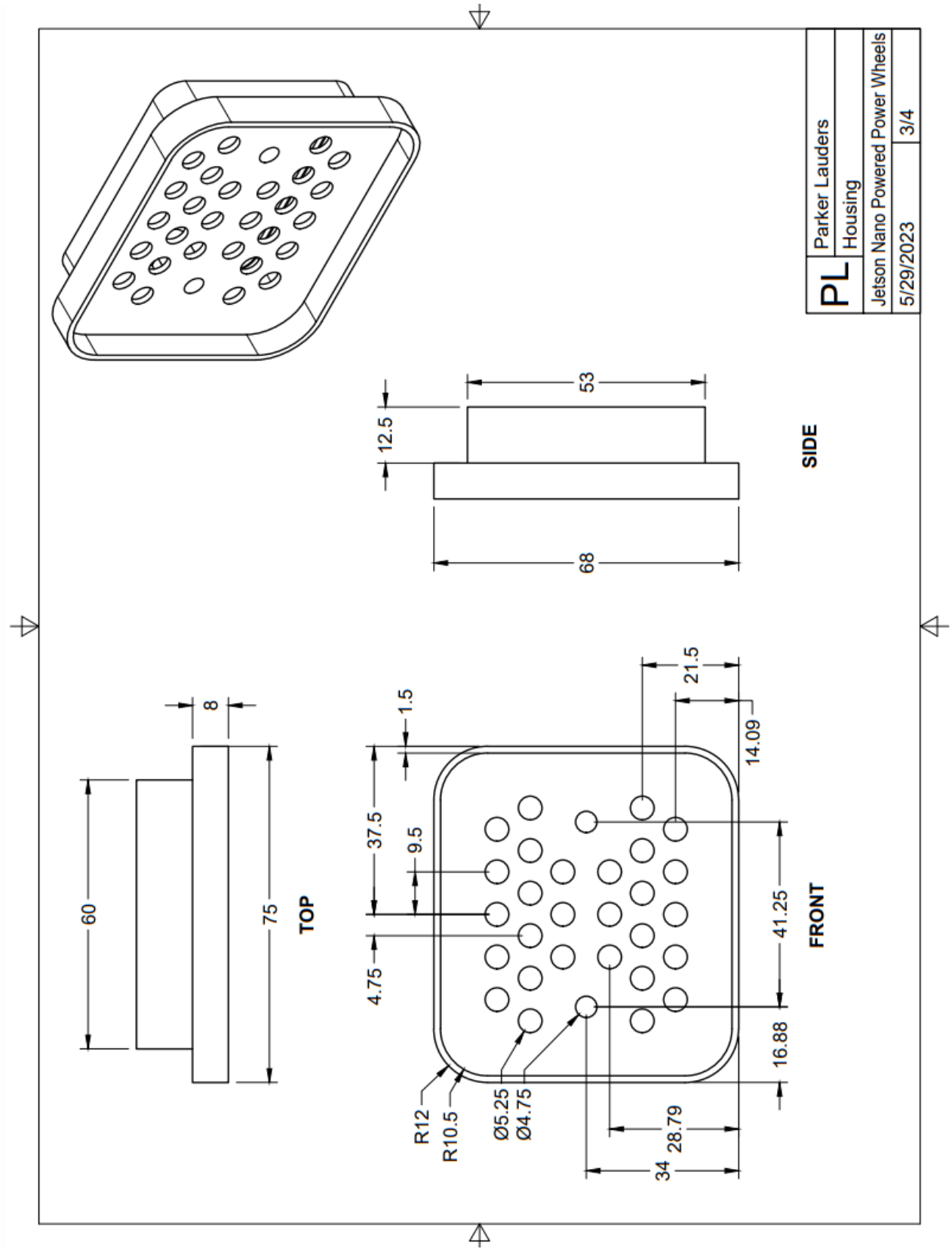


Figure 43. Back Indicator Drawing Page 3 of 4.

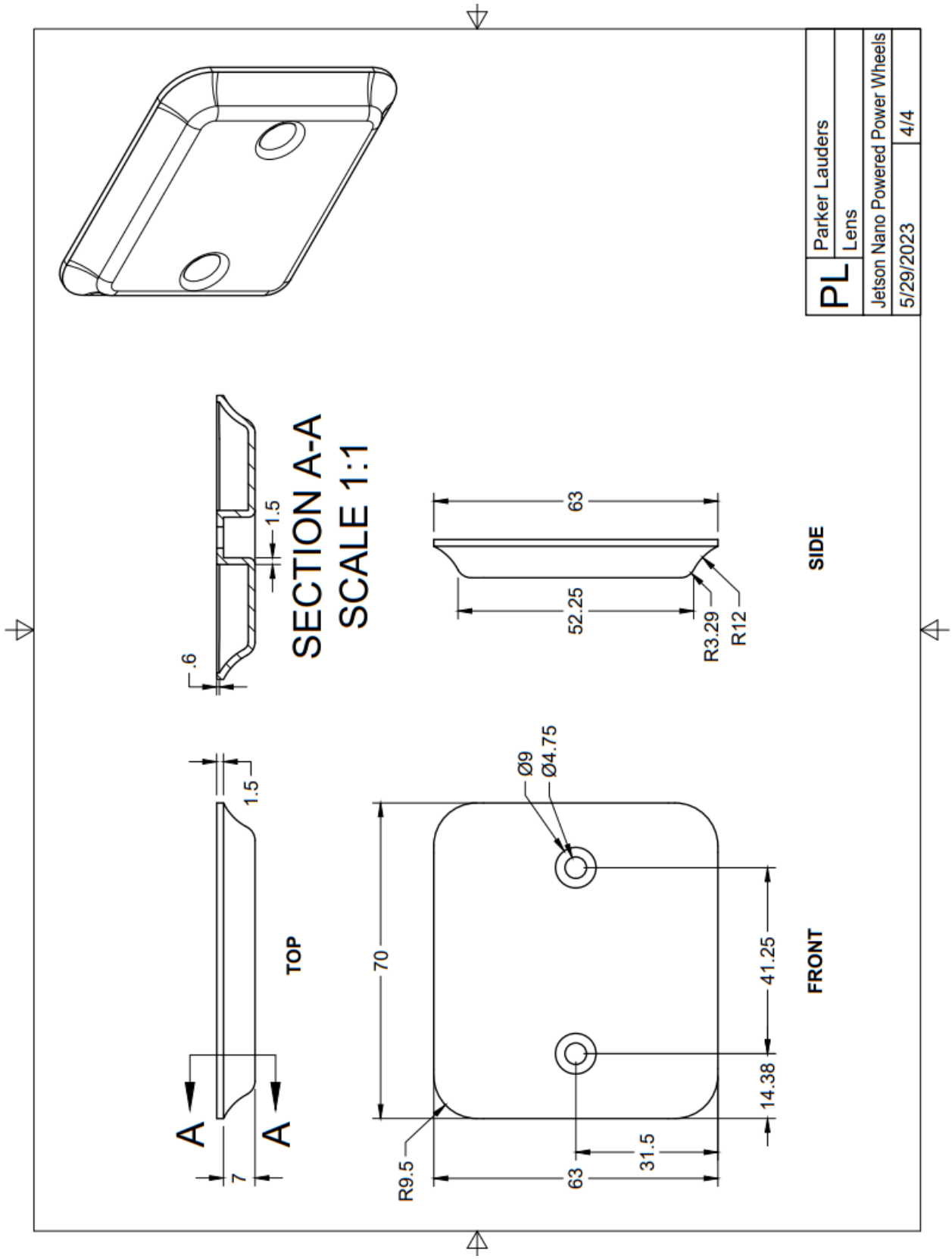


Figure 44. Back Indicator Drawing Page 4 of 4.

Figure 45. Safety Indicator Drawing Page 1 of 3.

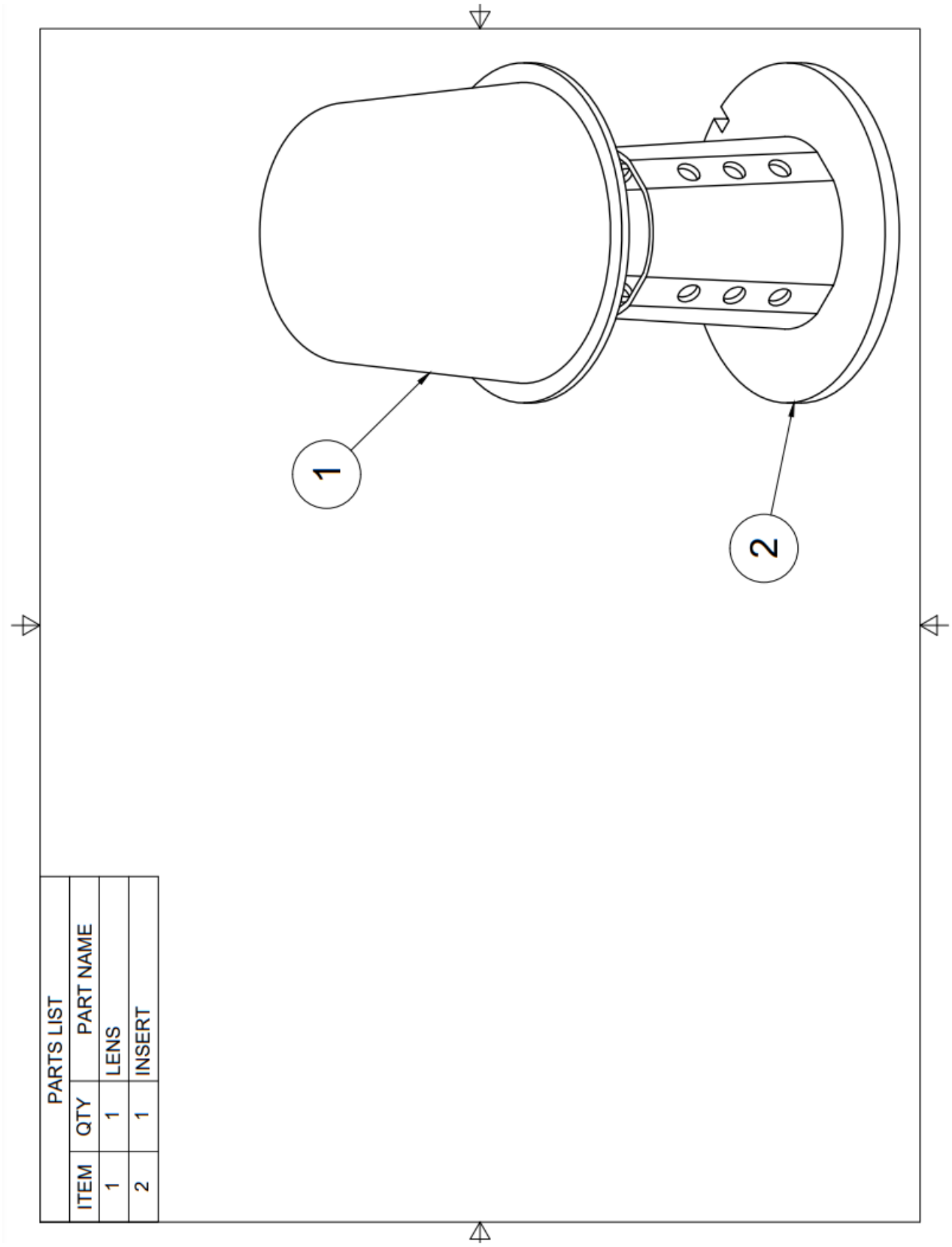


Figure 46. Safety Indicator Drawing Page 2 of 3.

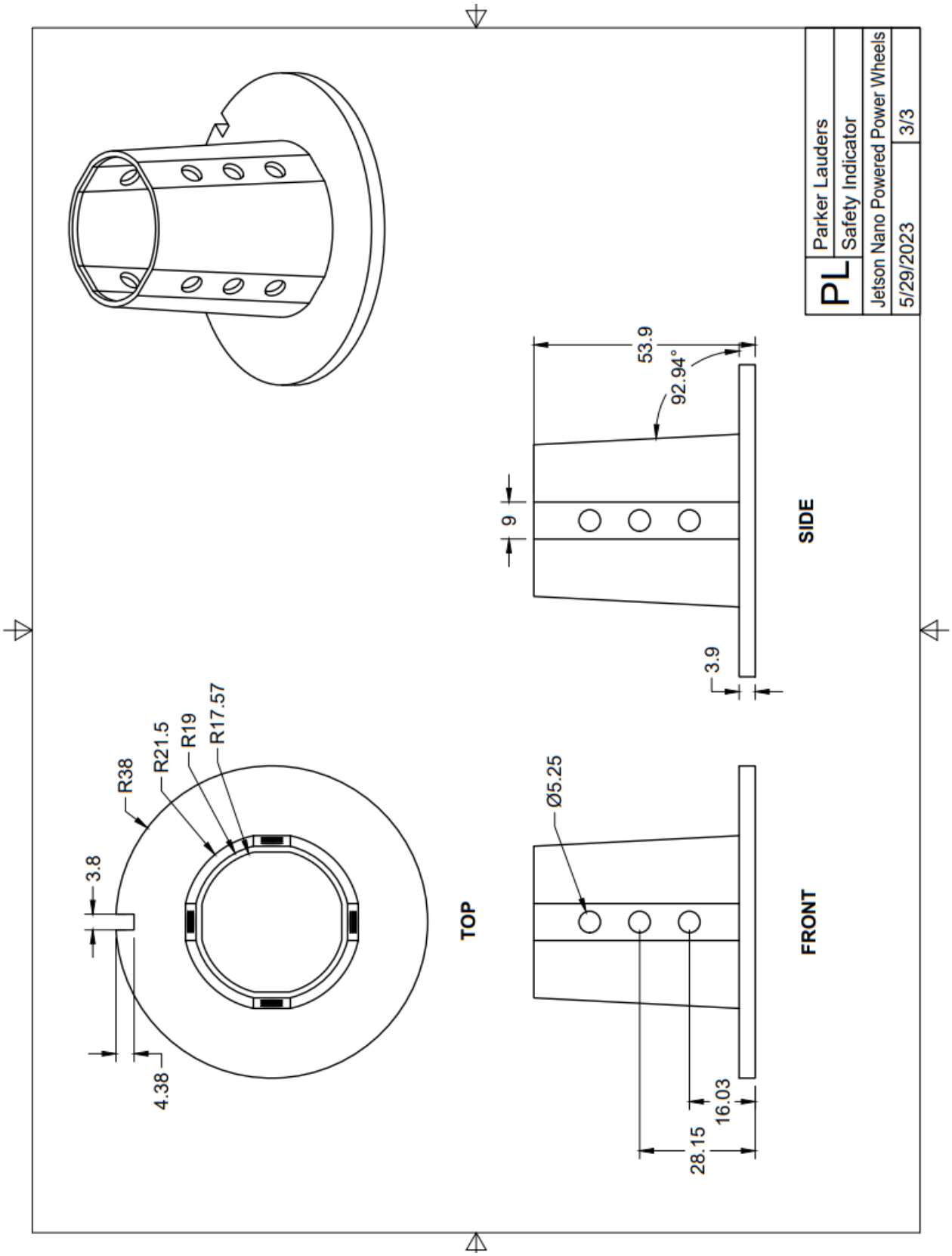


Figure 47. Safety Indicator Drawing Page 3 of 3.

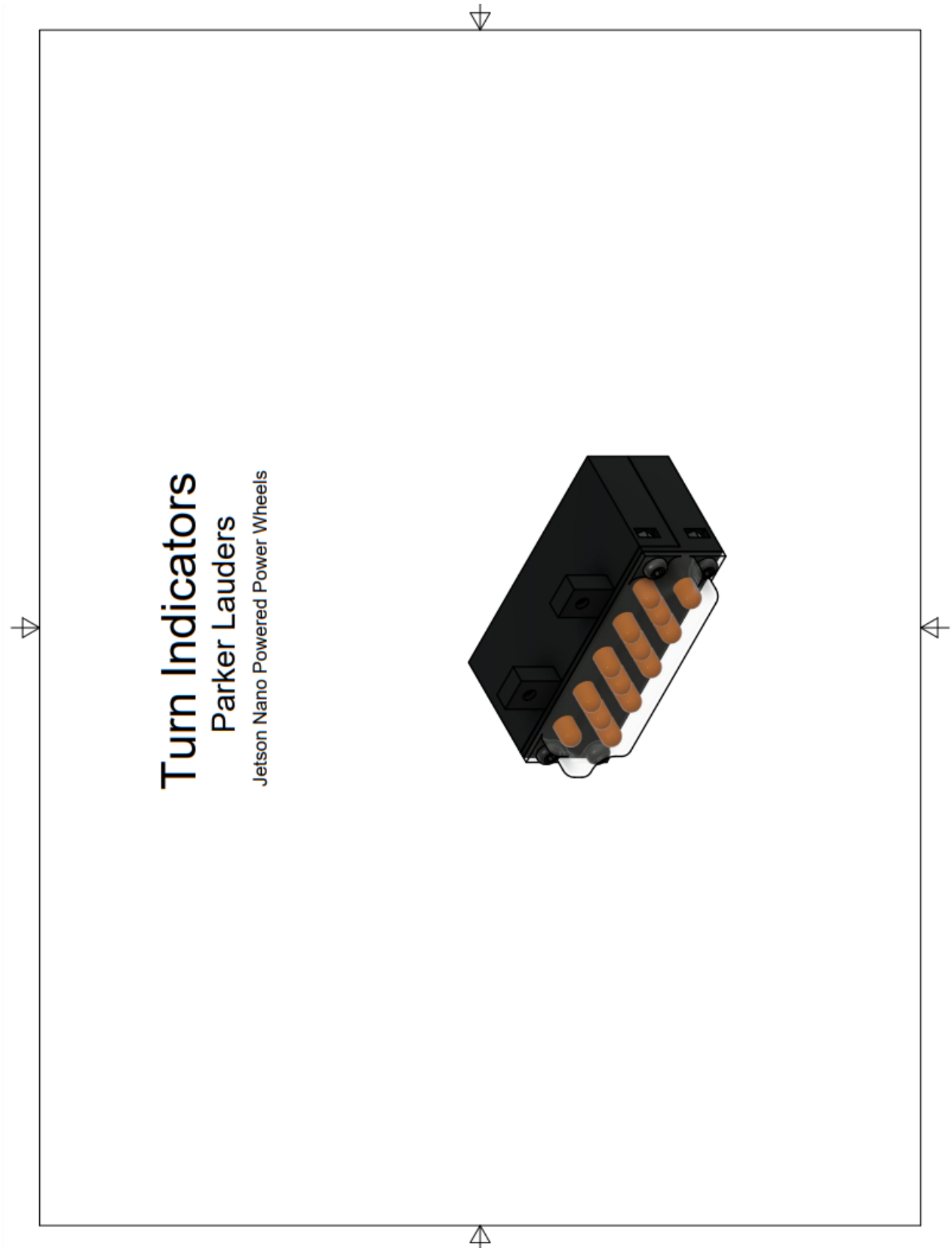


Figure 48. Turn Indicator Drawing Page 1 of 5.

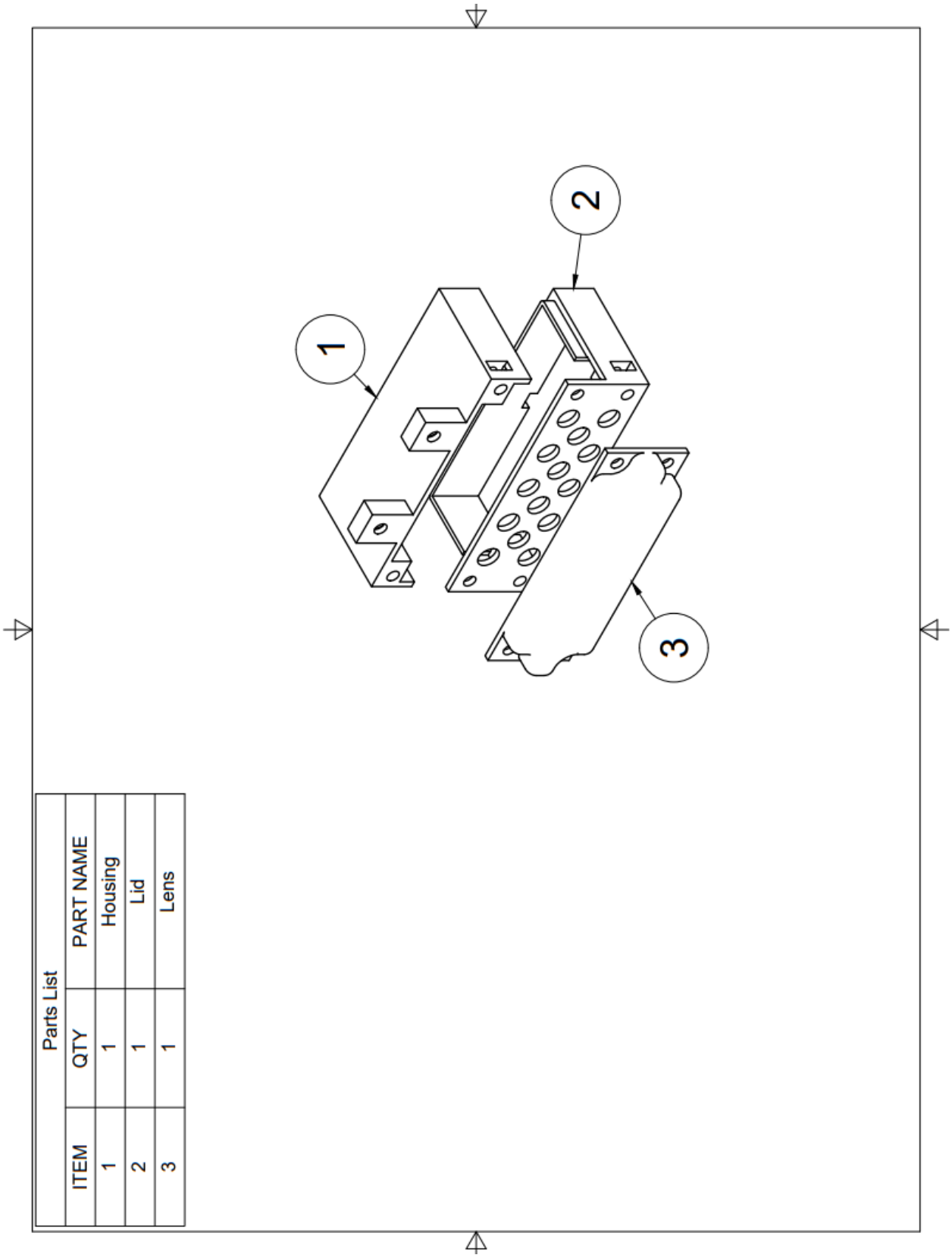


Figure 49. Turn Indicator Drawing Page 2 of 5.

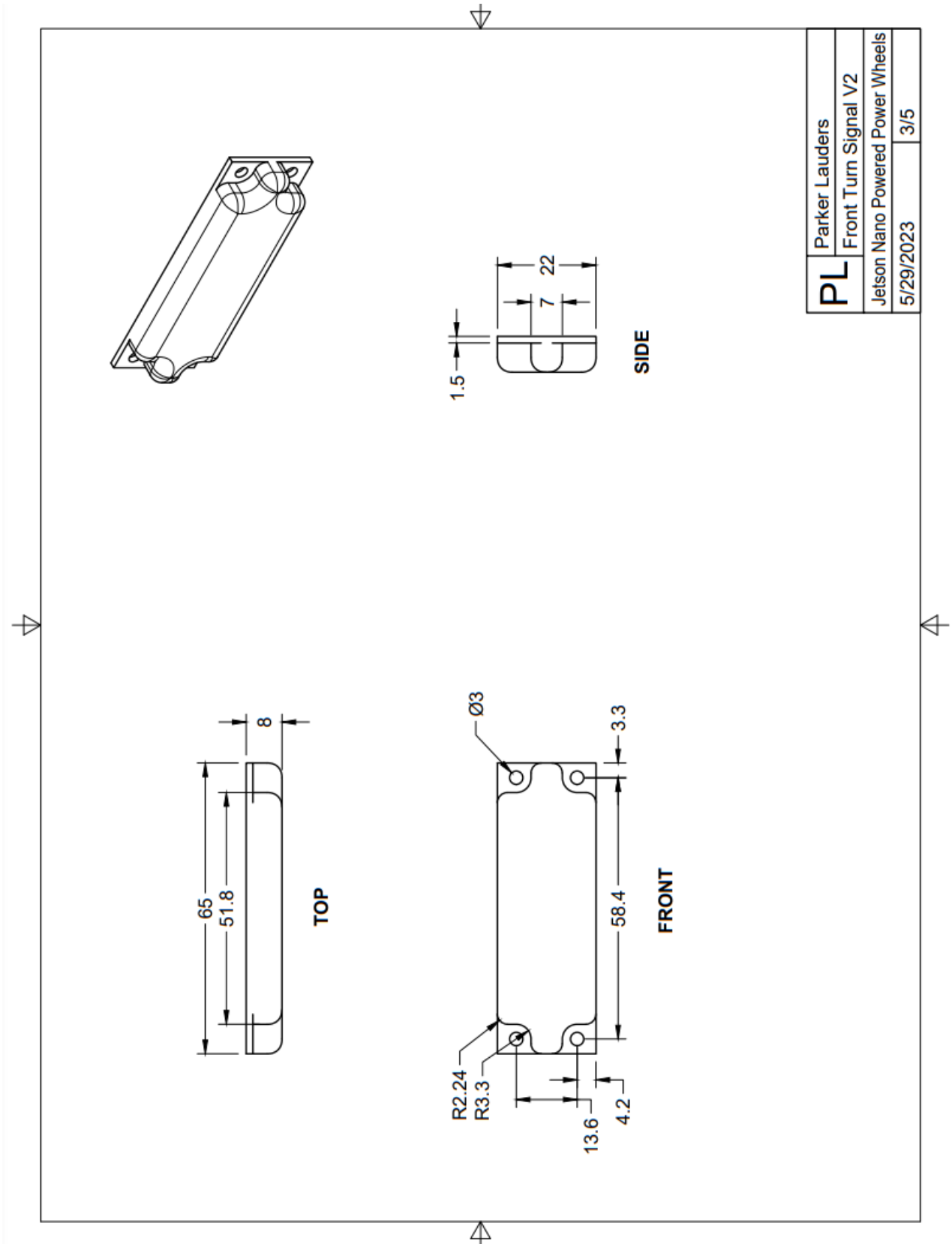


Figure 50. Turn Indicator Drawing Page 3 of 5.

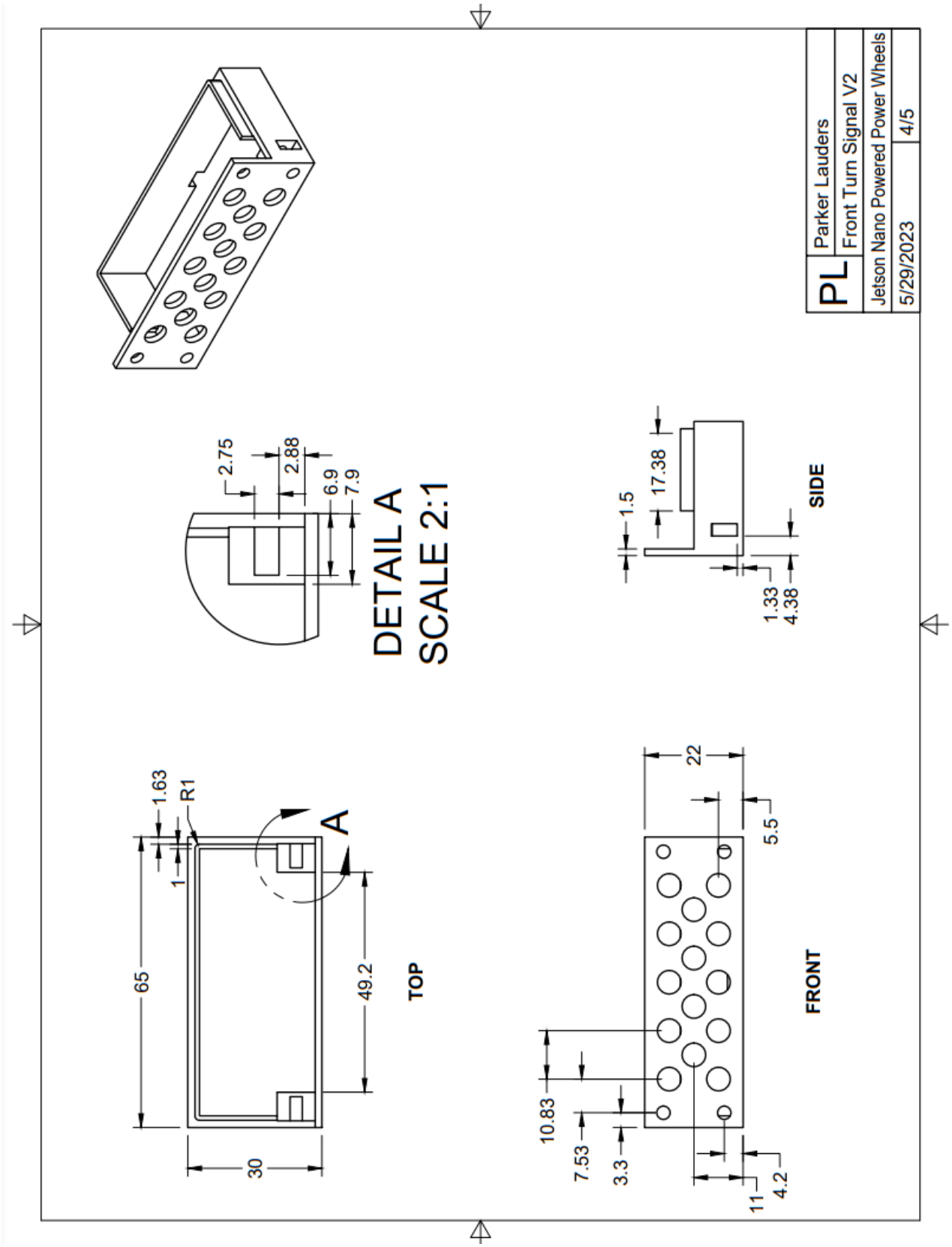


Figure 51. Turn Indicator Drawing Page 4 of 5.

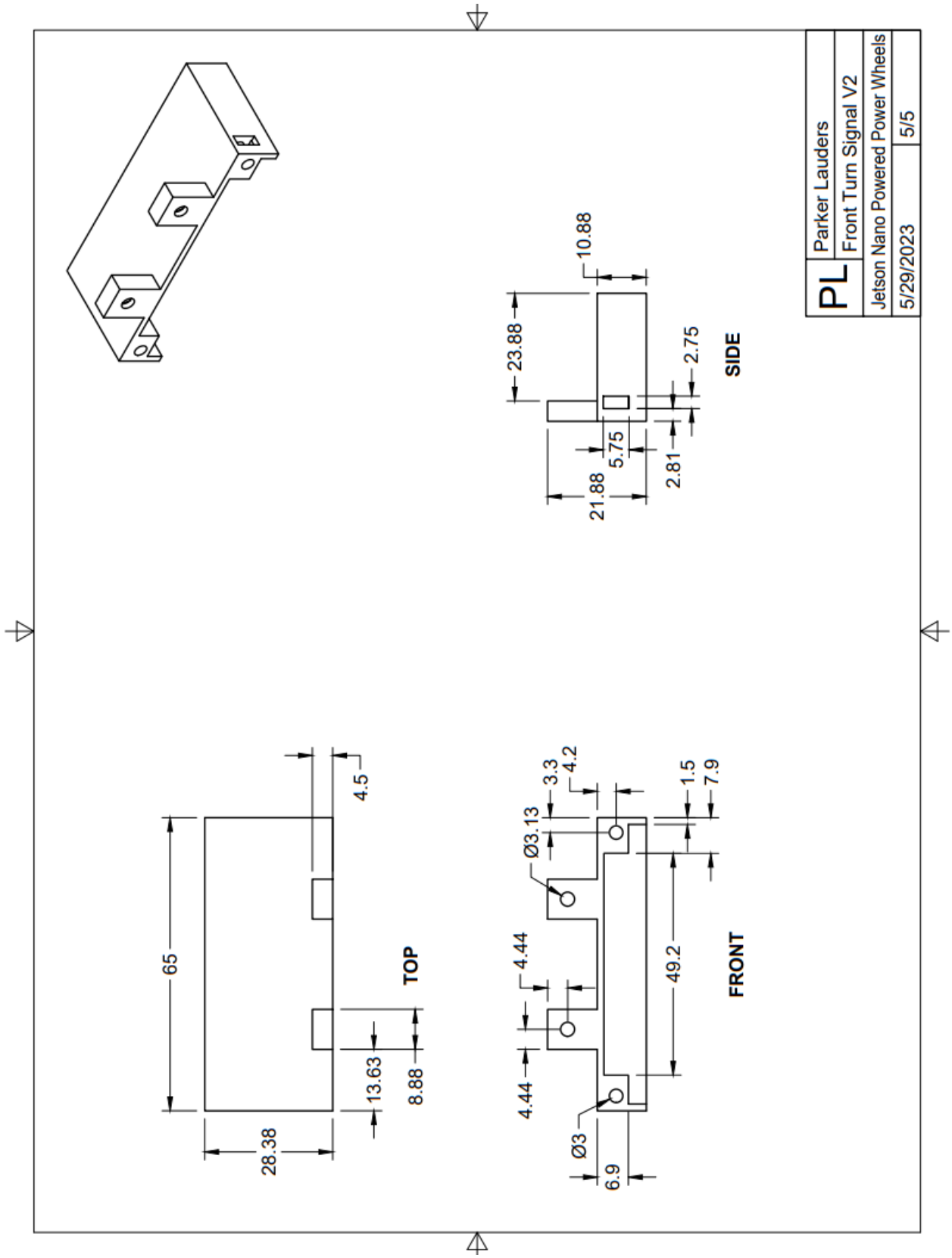


Figure 52. Turn Indicator Drawing Page 5 of 5.

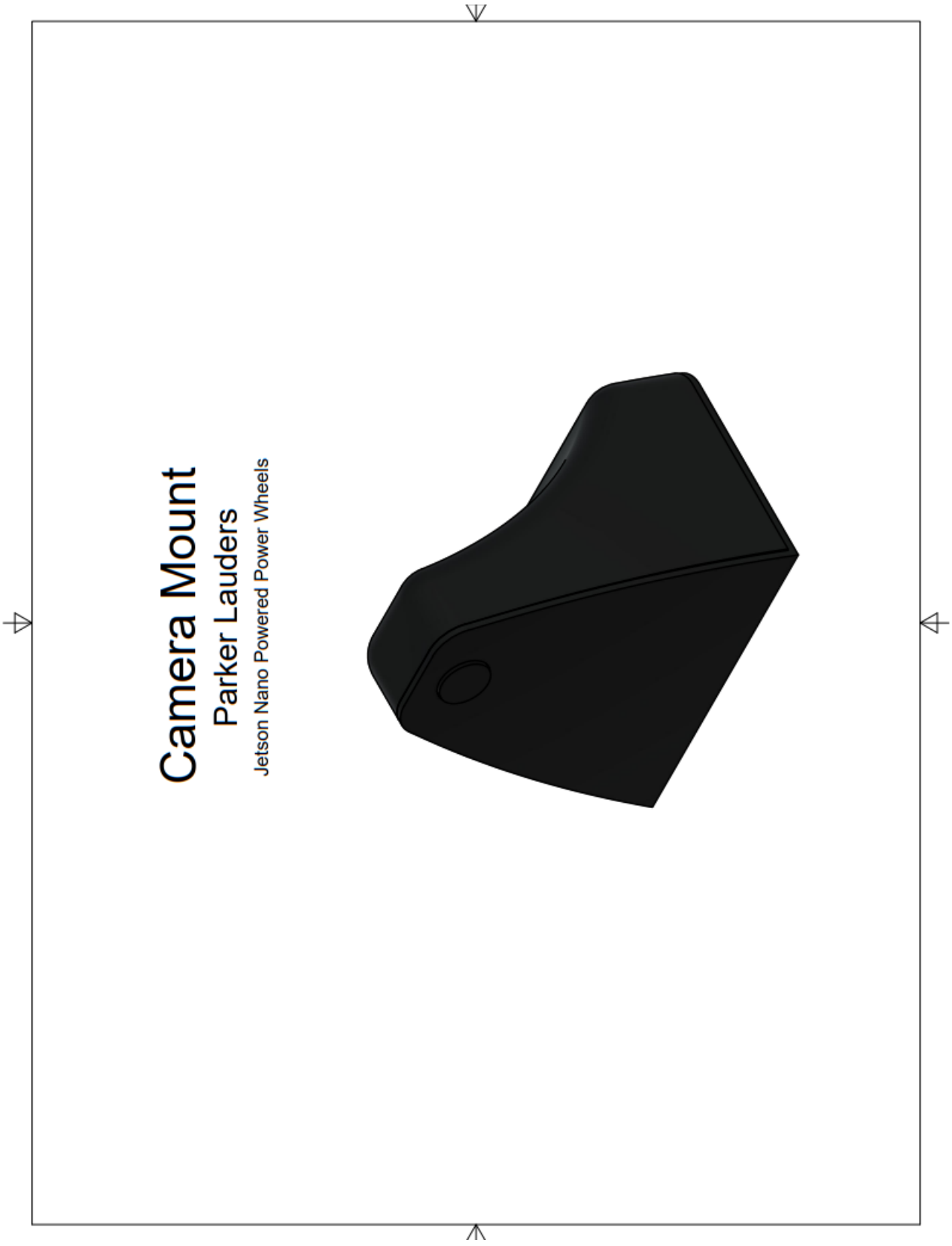


Figure 53. Camera Mount Drawing Page 1 of 4.

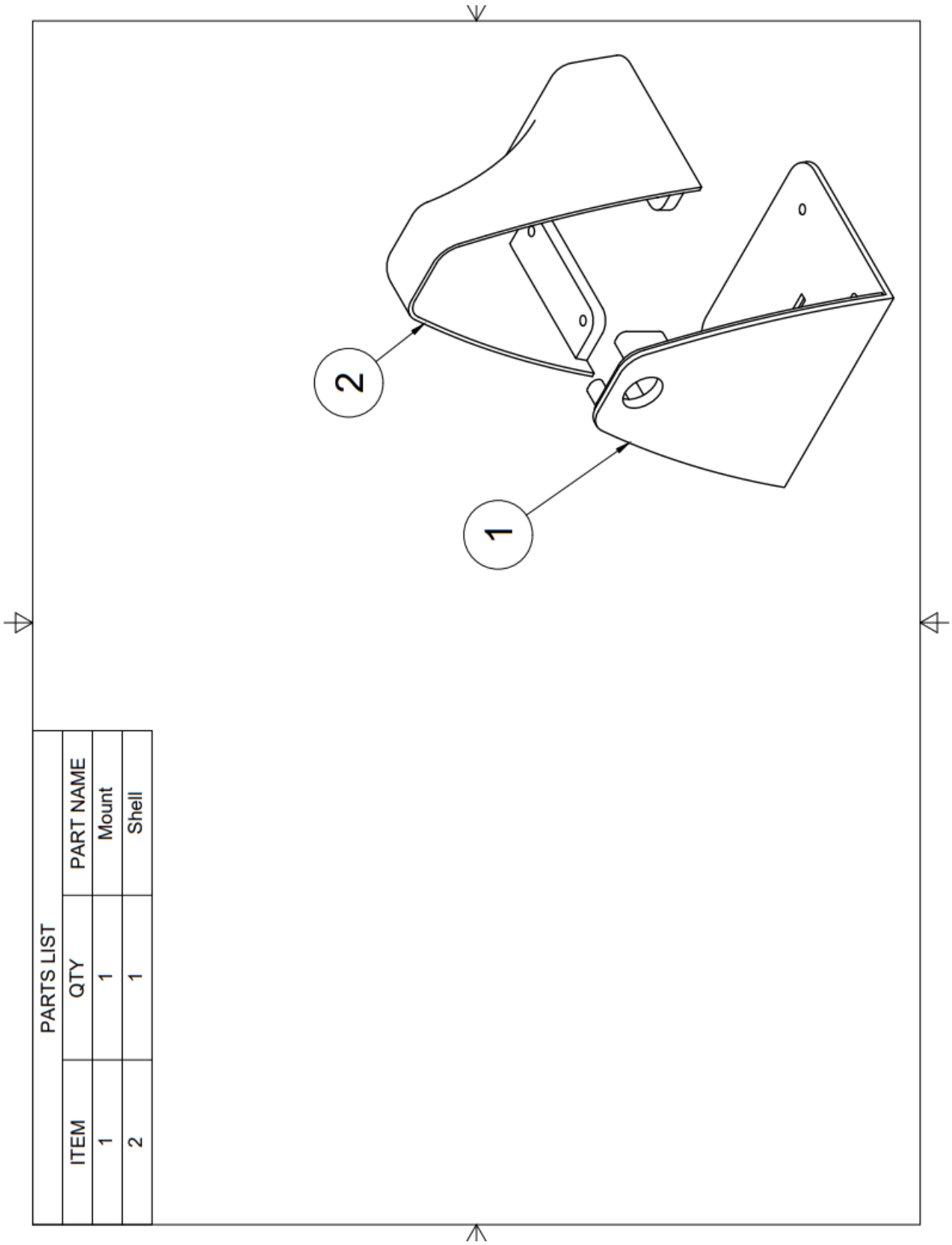


Figure 54. Camera Mount Drawing Page 2 of 4.

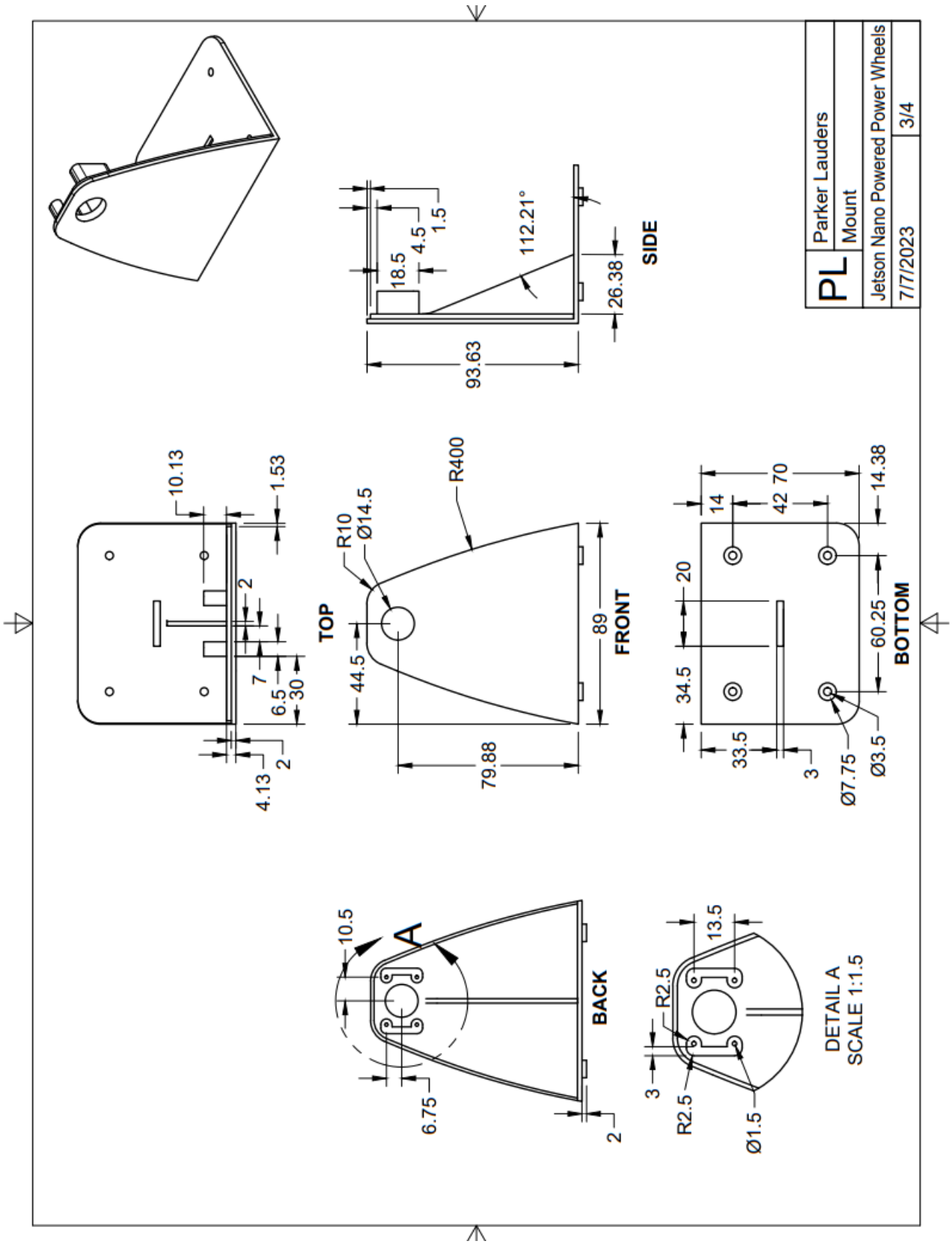


Figure 55. Camera Mount Drawing Page 3 of 4.

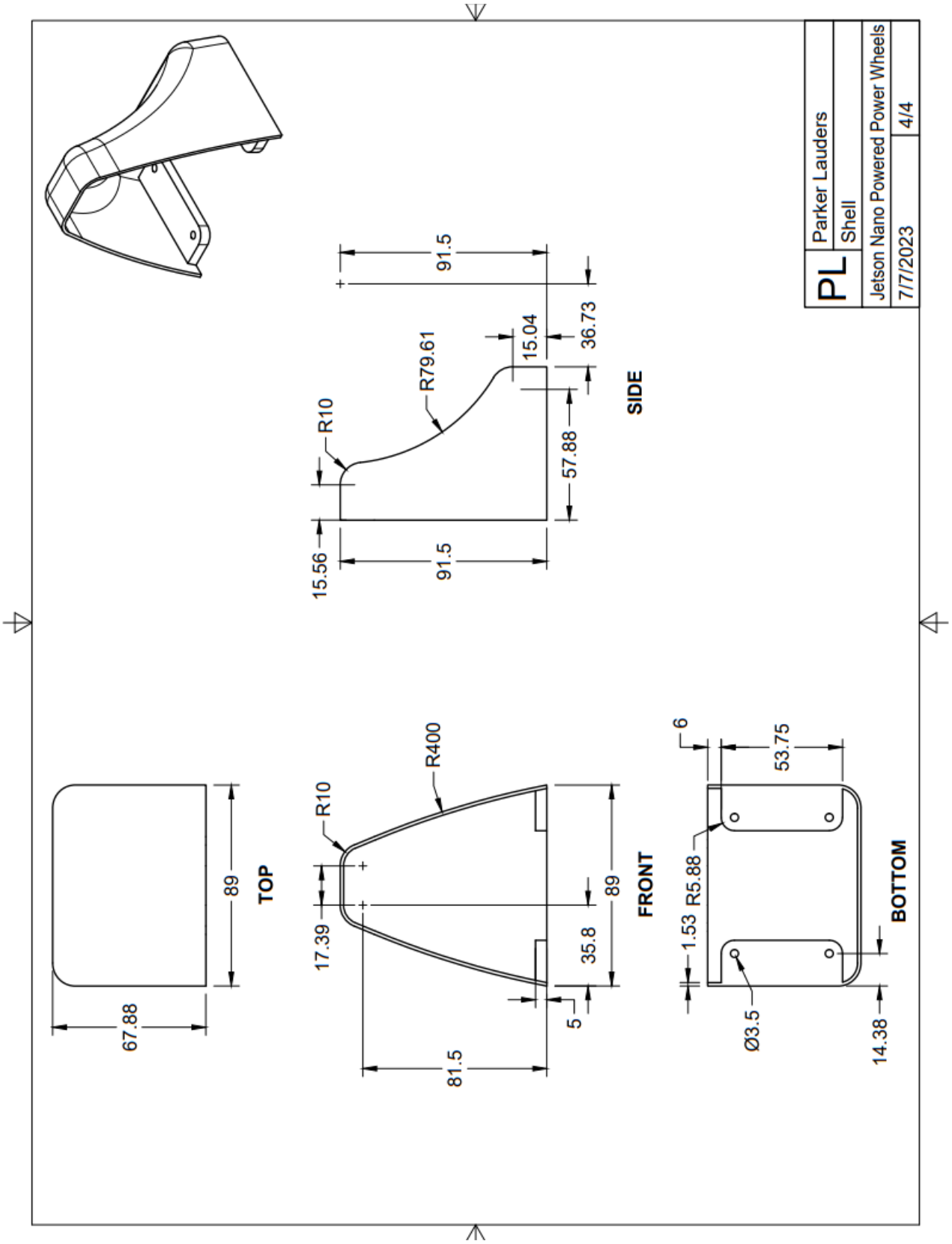


Figure 56. Camera Mount Drawing Page 4 of 4.

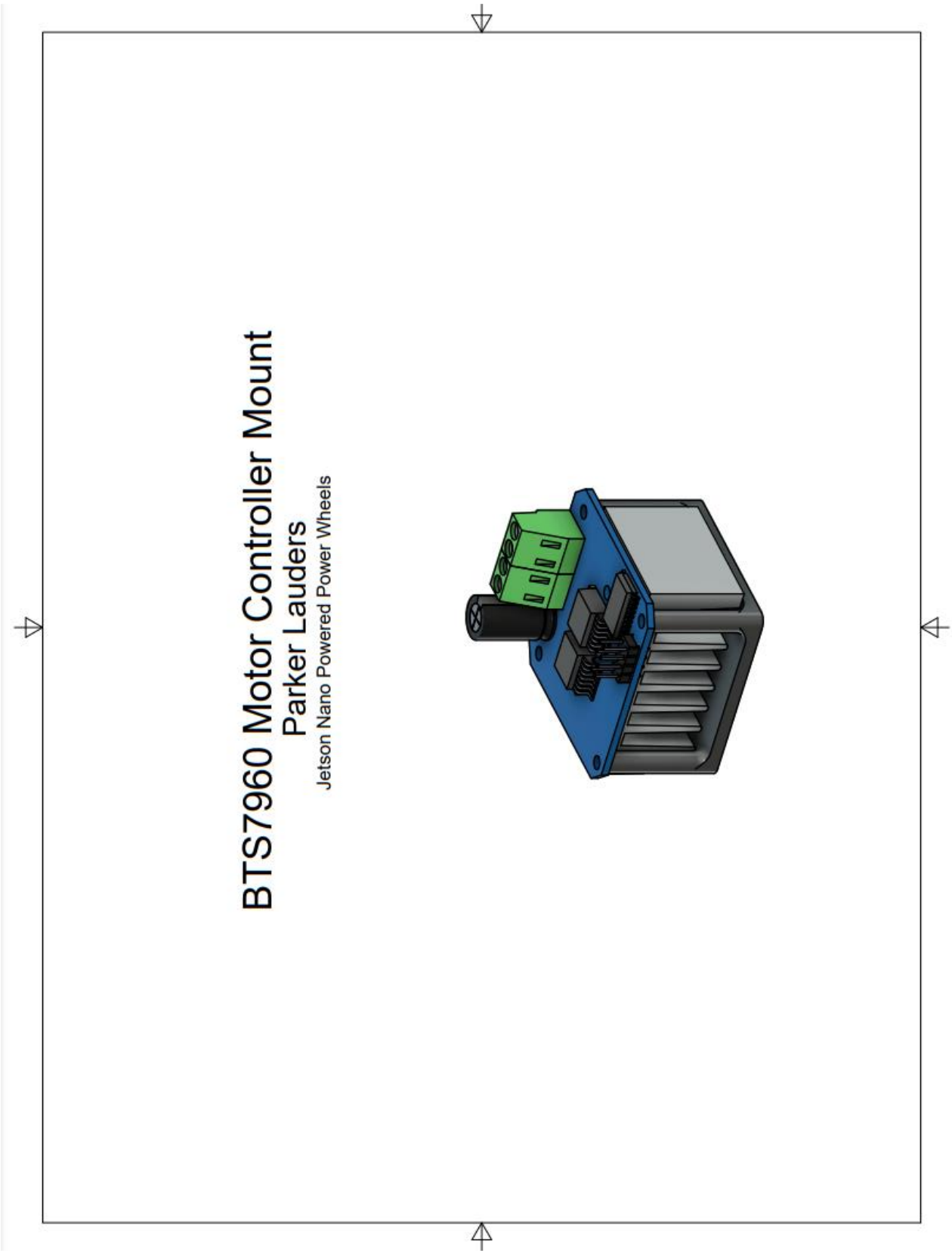


Figure 57. BTS7960 Motor Controller Mount Drawing File Page 1 of 3.

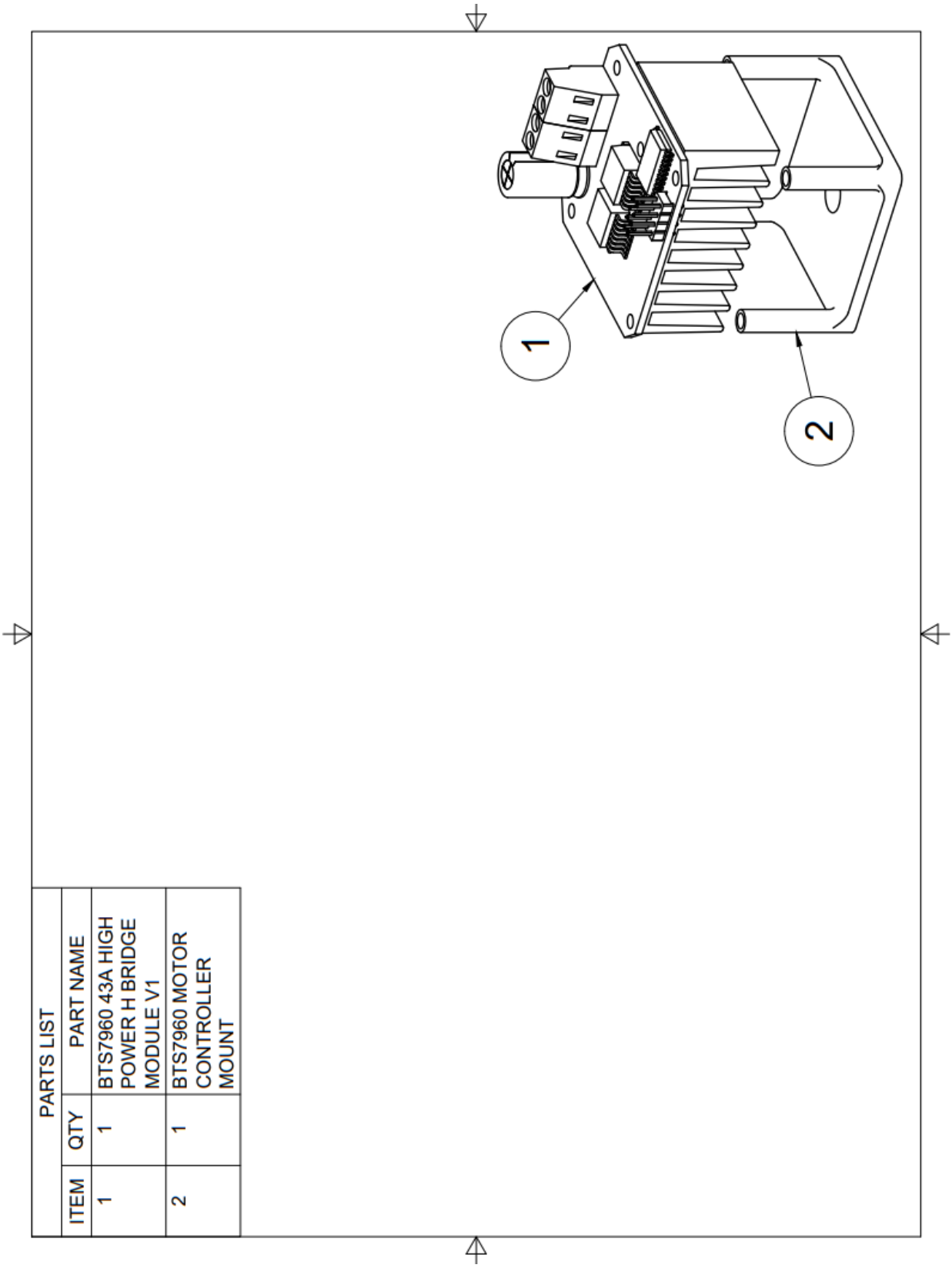


Figure 58. BTS7960 Motor Controller Mount Drawing File Page 2 of 3.

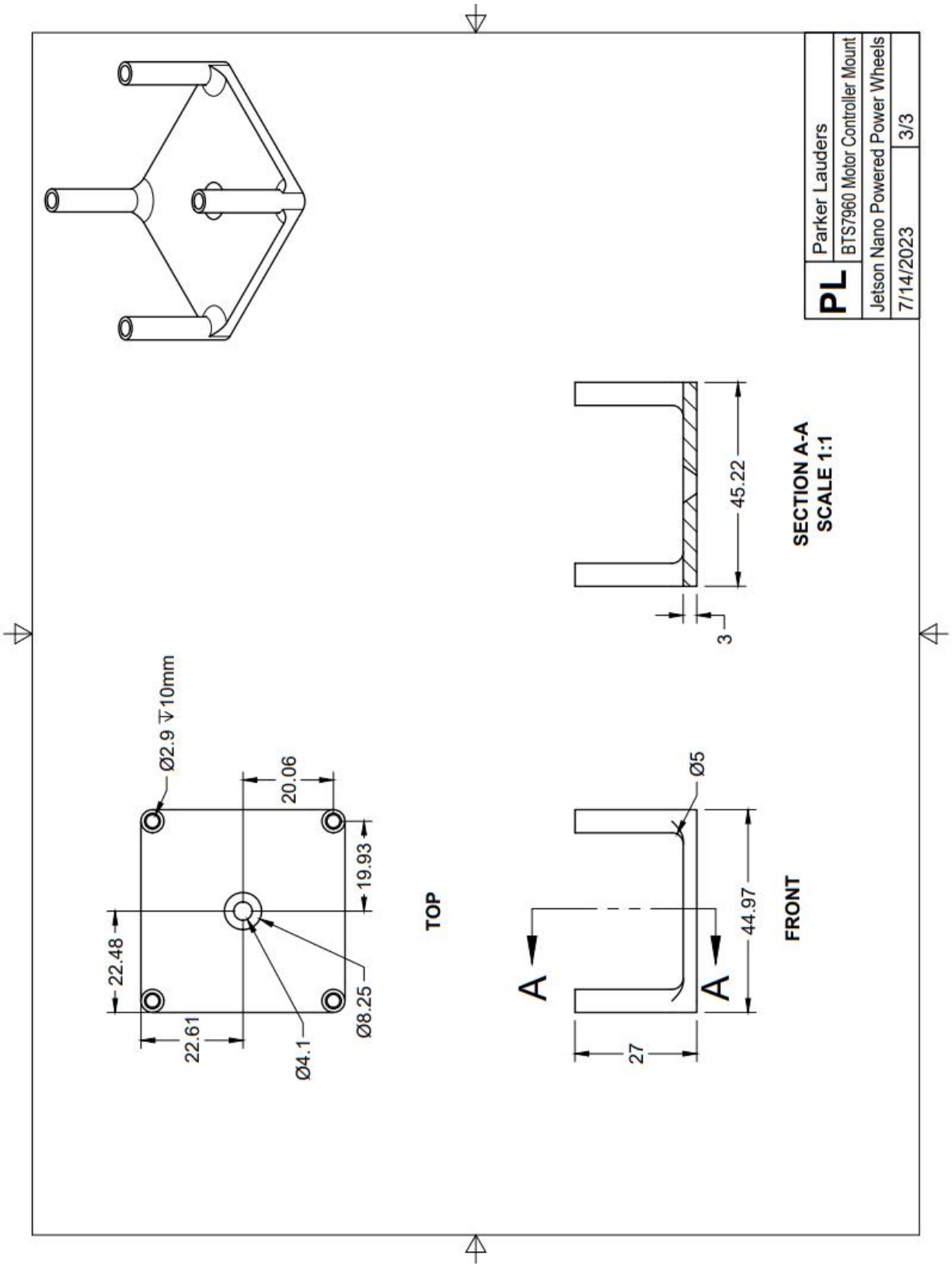


Figure 59. BTS7960 Motor Controller Mount Drawing File Page 3 of 3.

Figure 60. Mode Switch Mount Page 1 of 2.

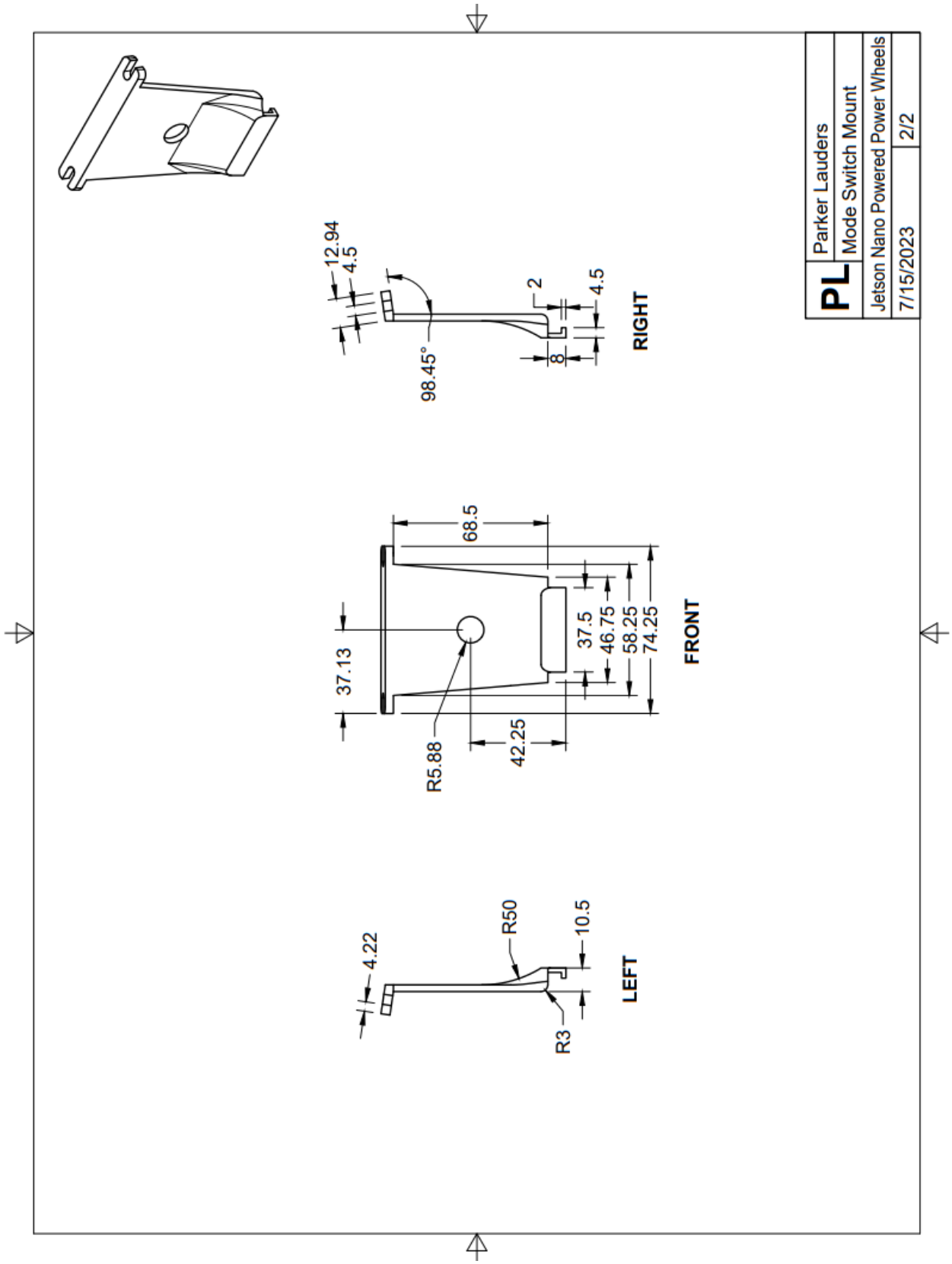


Figure 61. Mode Switch Mount Page 2 of 2.

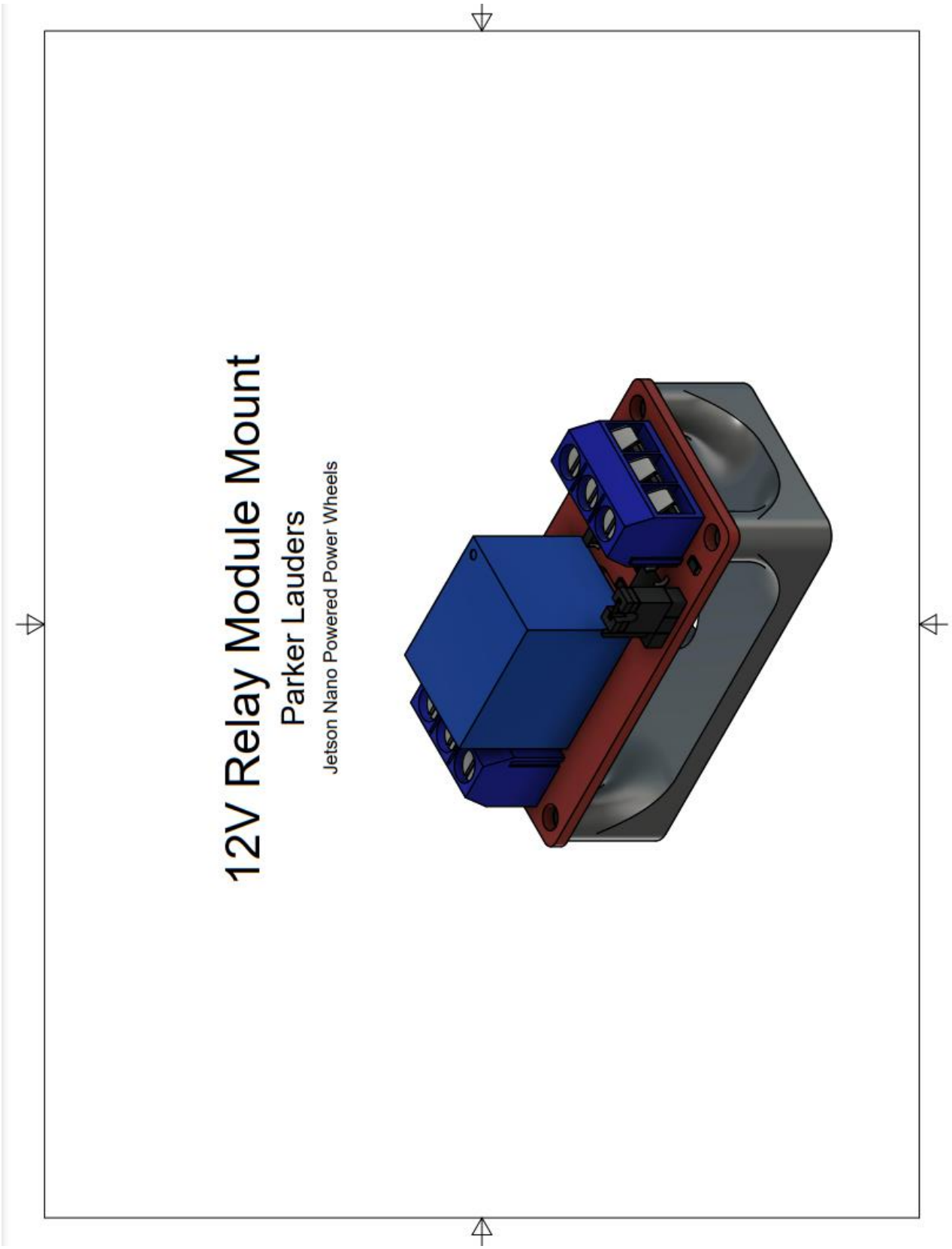


Figure 62. 12V Relay Module Mount Drawing File Page 1 of 3.

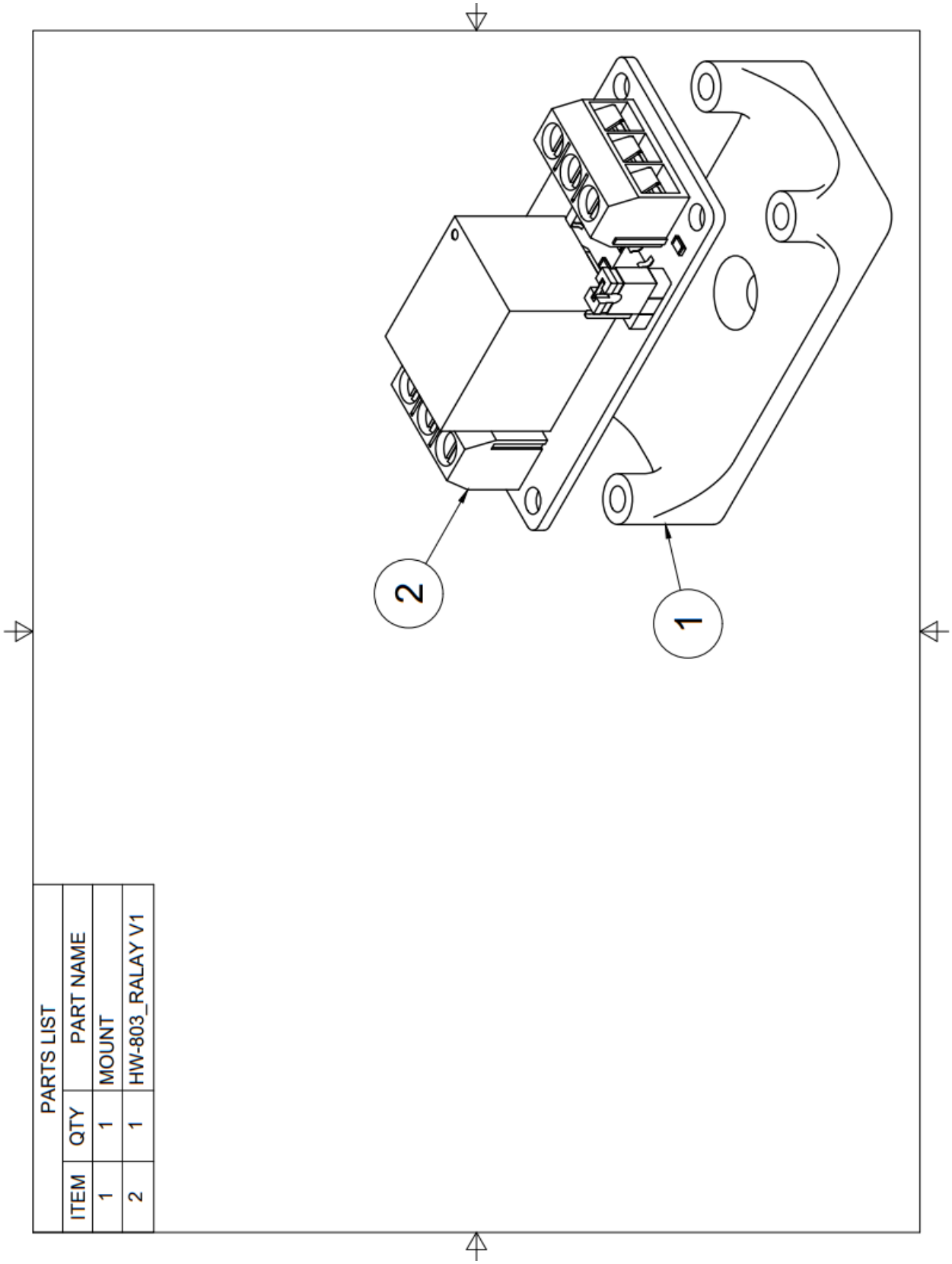


Figure 63. 12V Relay Module Mount Drawing File Page 2 of 3.

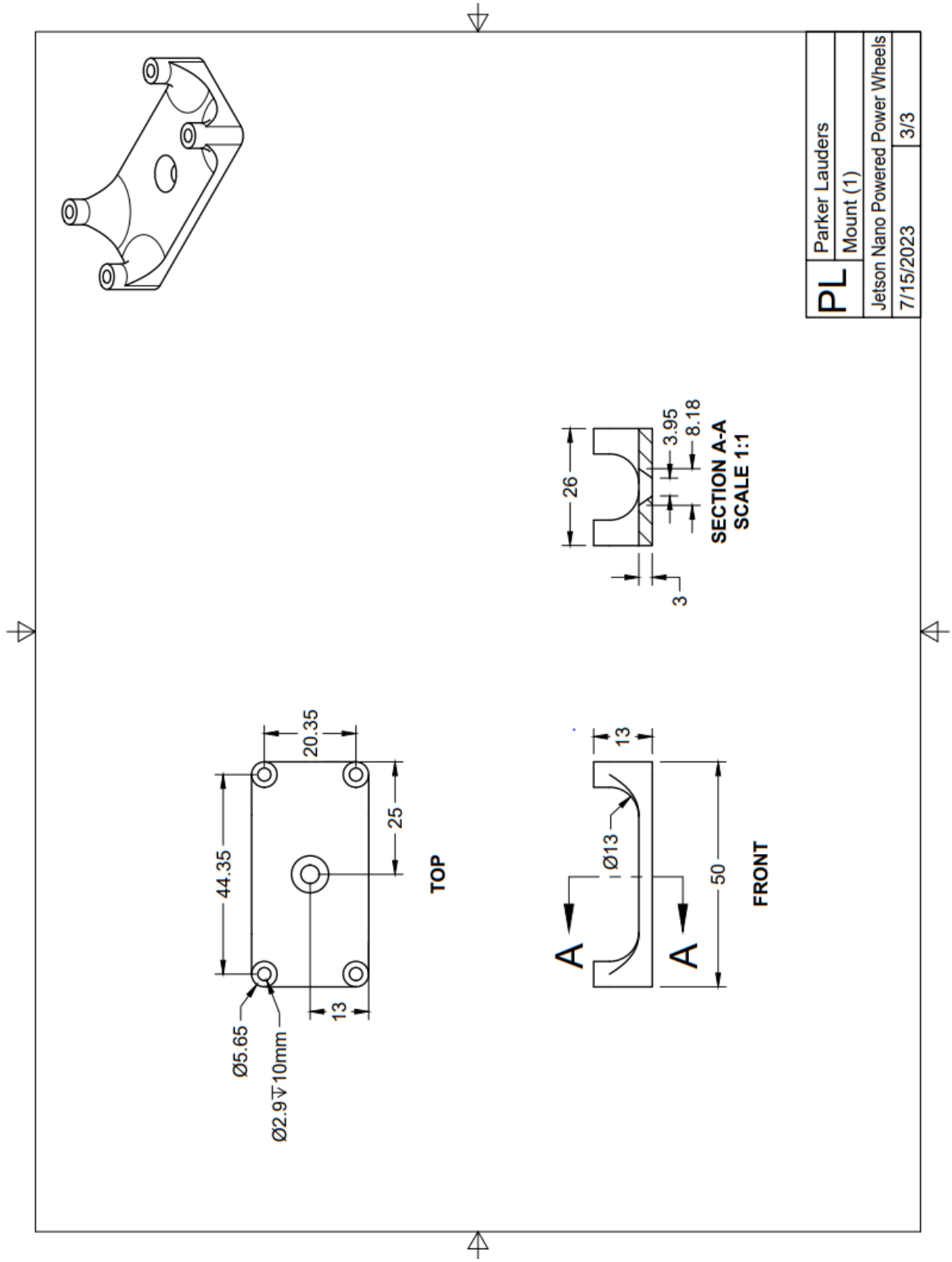


Figure 64. 12V Relay Module Mount Drawing File Page 3 of 3.

Figure 65. Computing System Mount Drawing File Page 1 of 3.

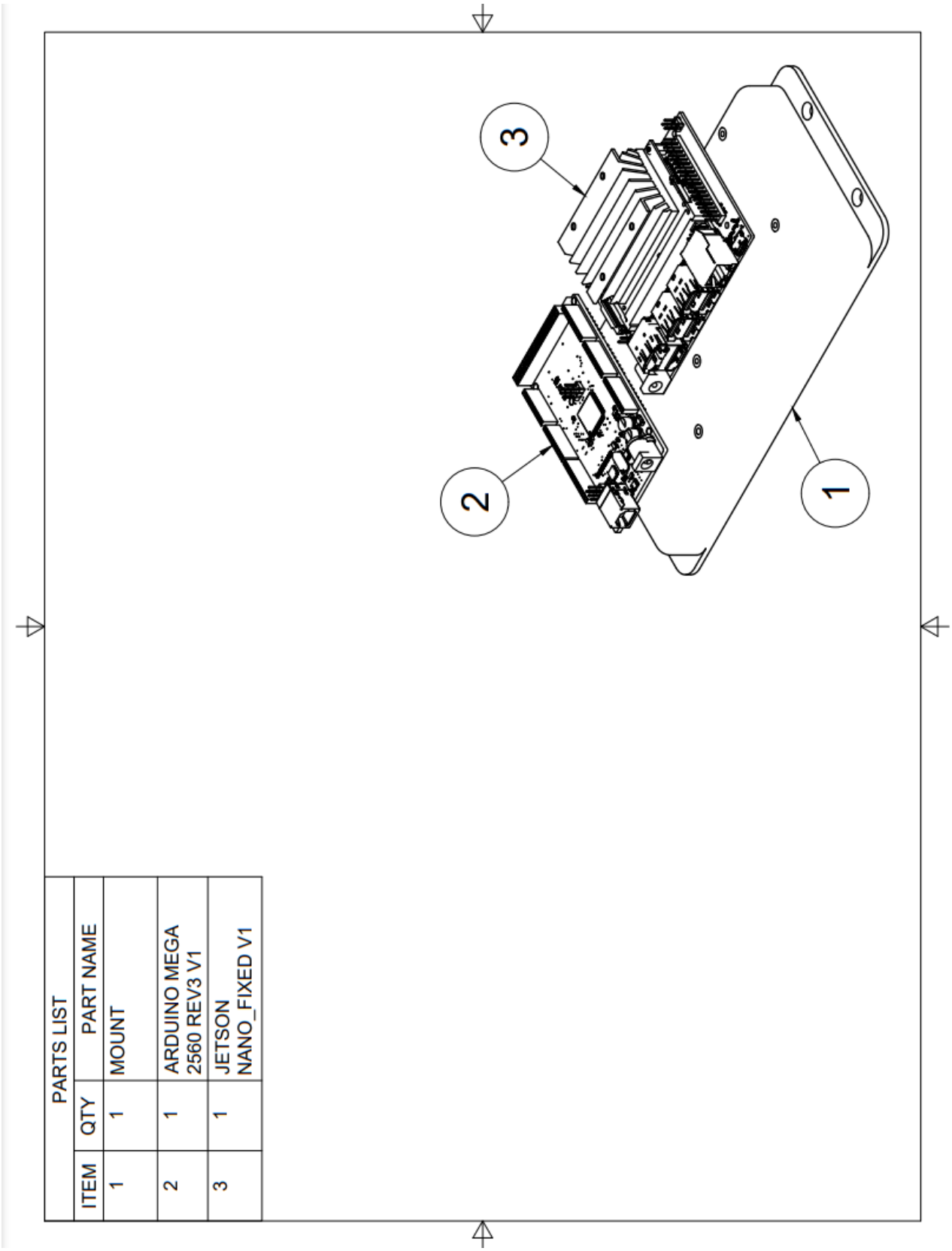


Figure 66. Computing System Mount Drawing File Page 2 of 3.

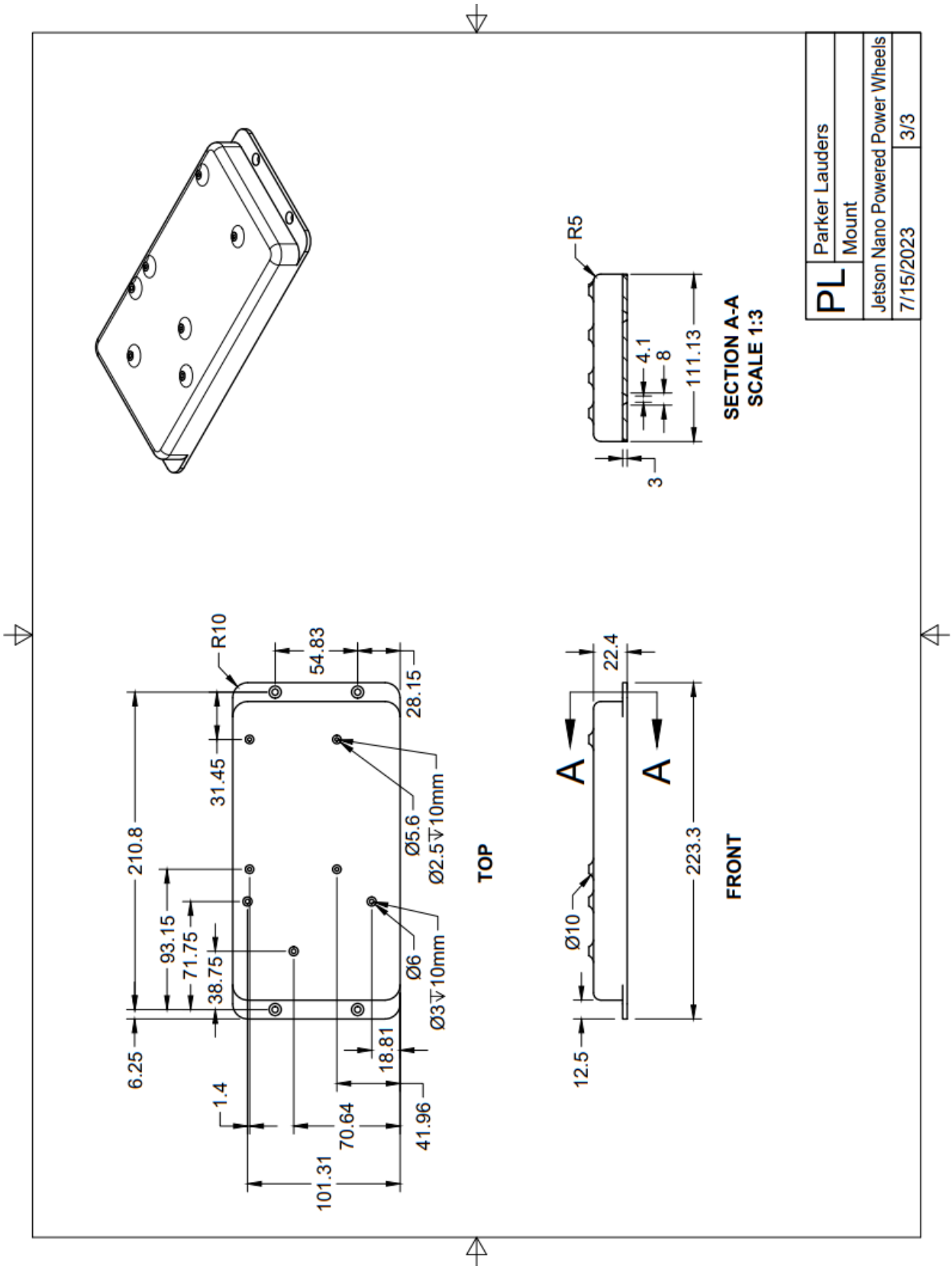


Figure 67. Computing System Mount Drawing File Page 3 of 3.

Figure 68. Battery Holder Drawing File Page 1 of 2.

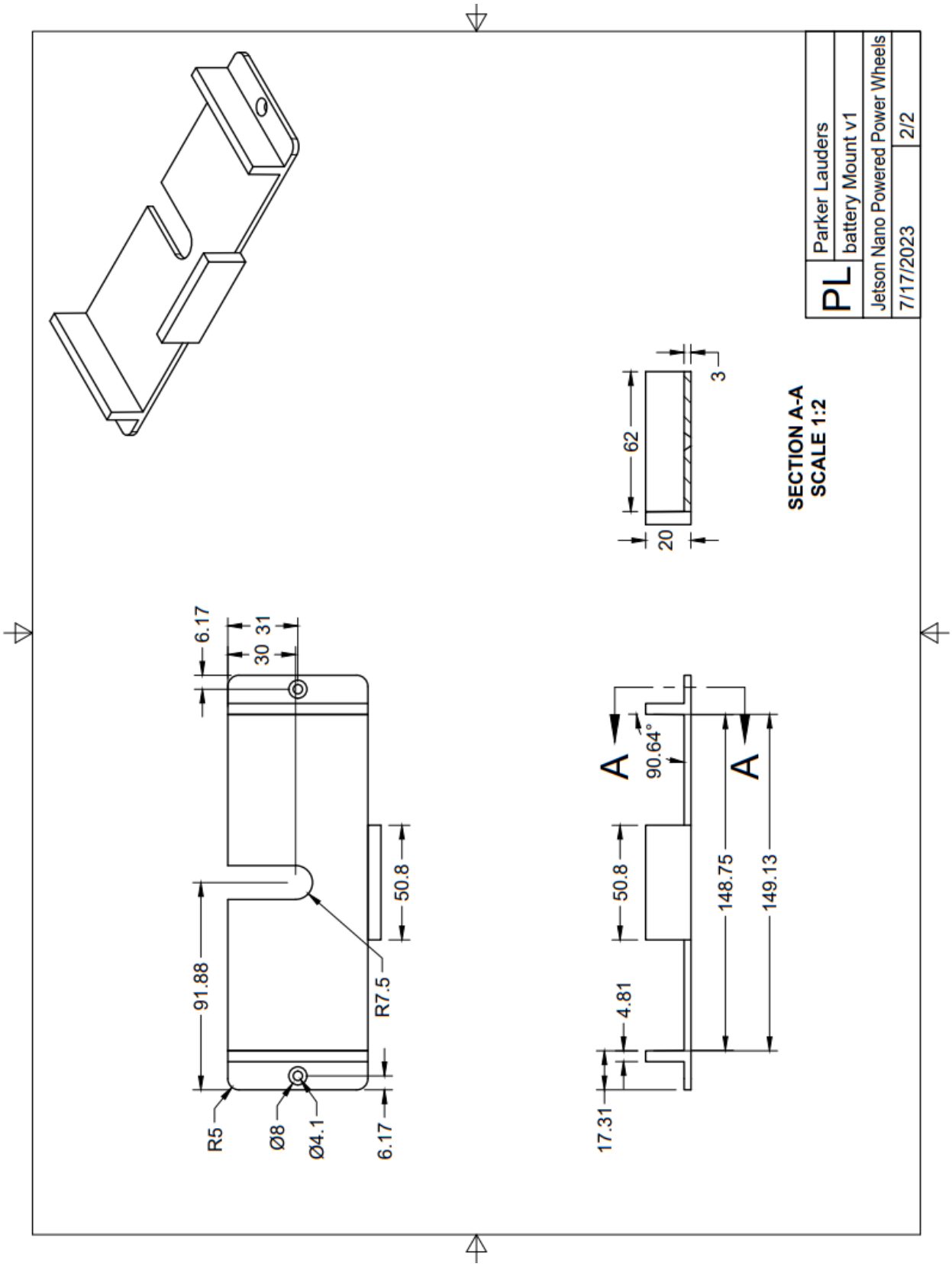


Figure 69. Battery Holder Drawing File Page 2 of 2.

Figure 70. Logitech Camera Mount Drawing File Page 1 of 4.

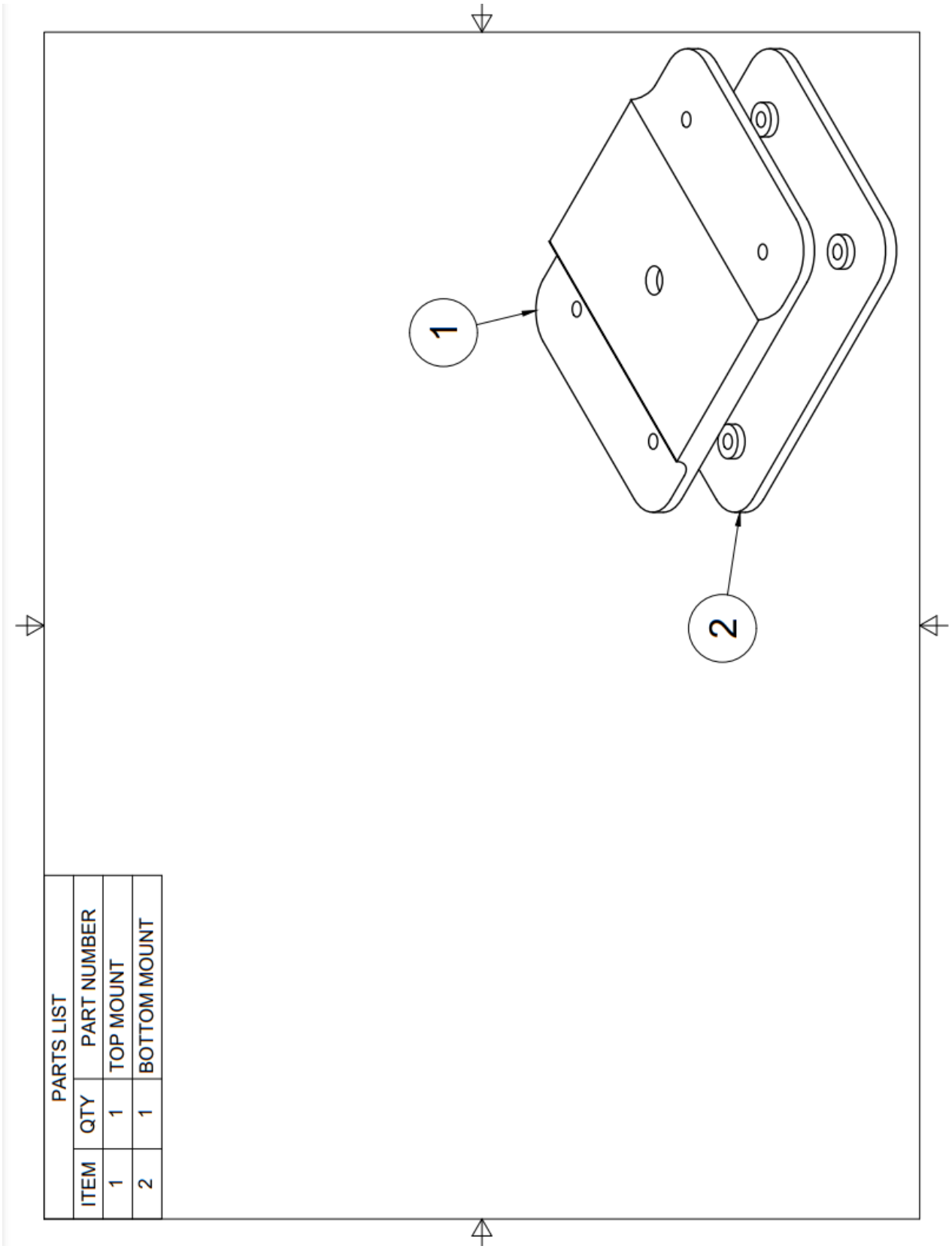


Figure 71. Logitech Camera Mount Drawing File Page 2 of 4.

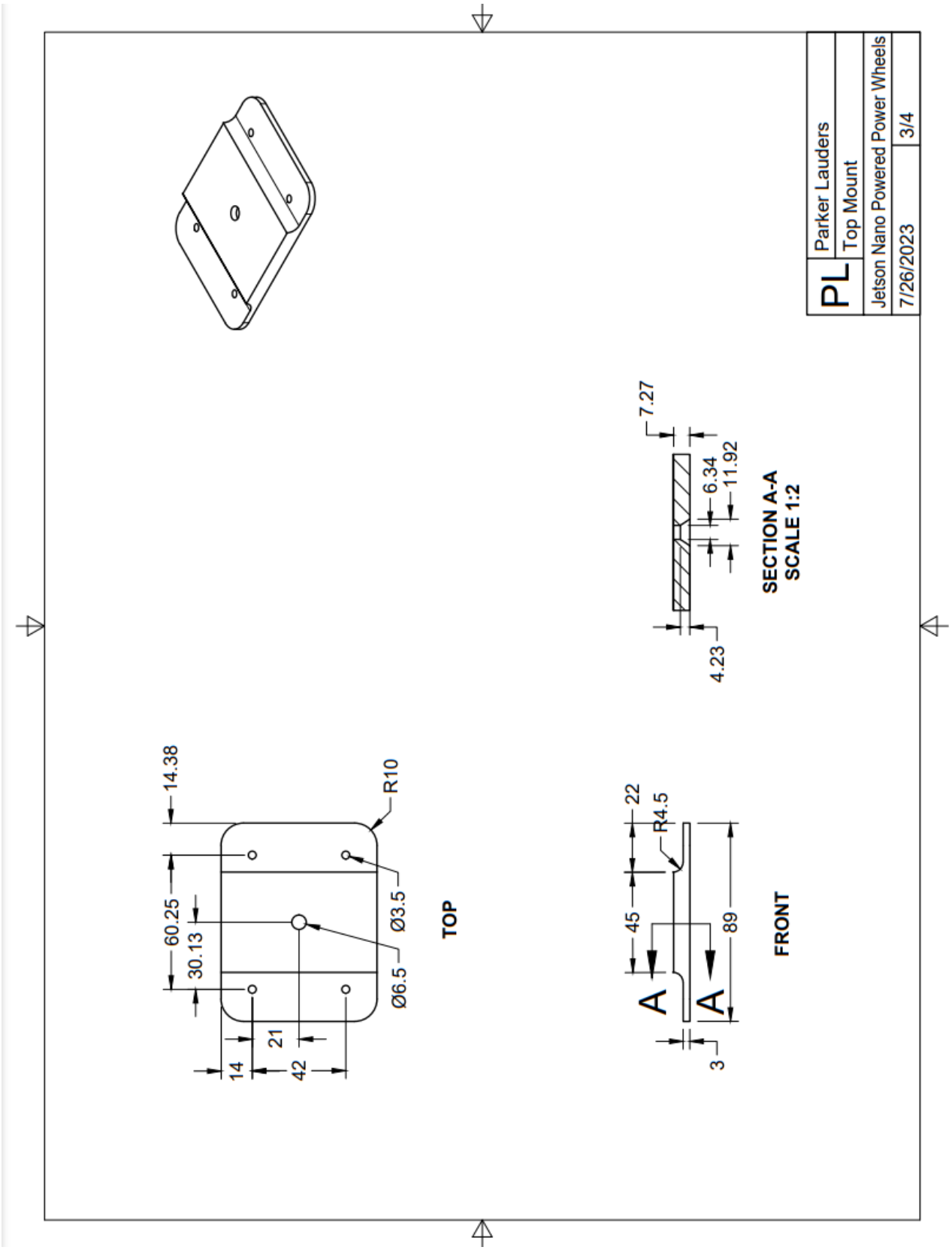


Figure 72. Logitech Camera Mount Drawing File Page 3 of 4.

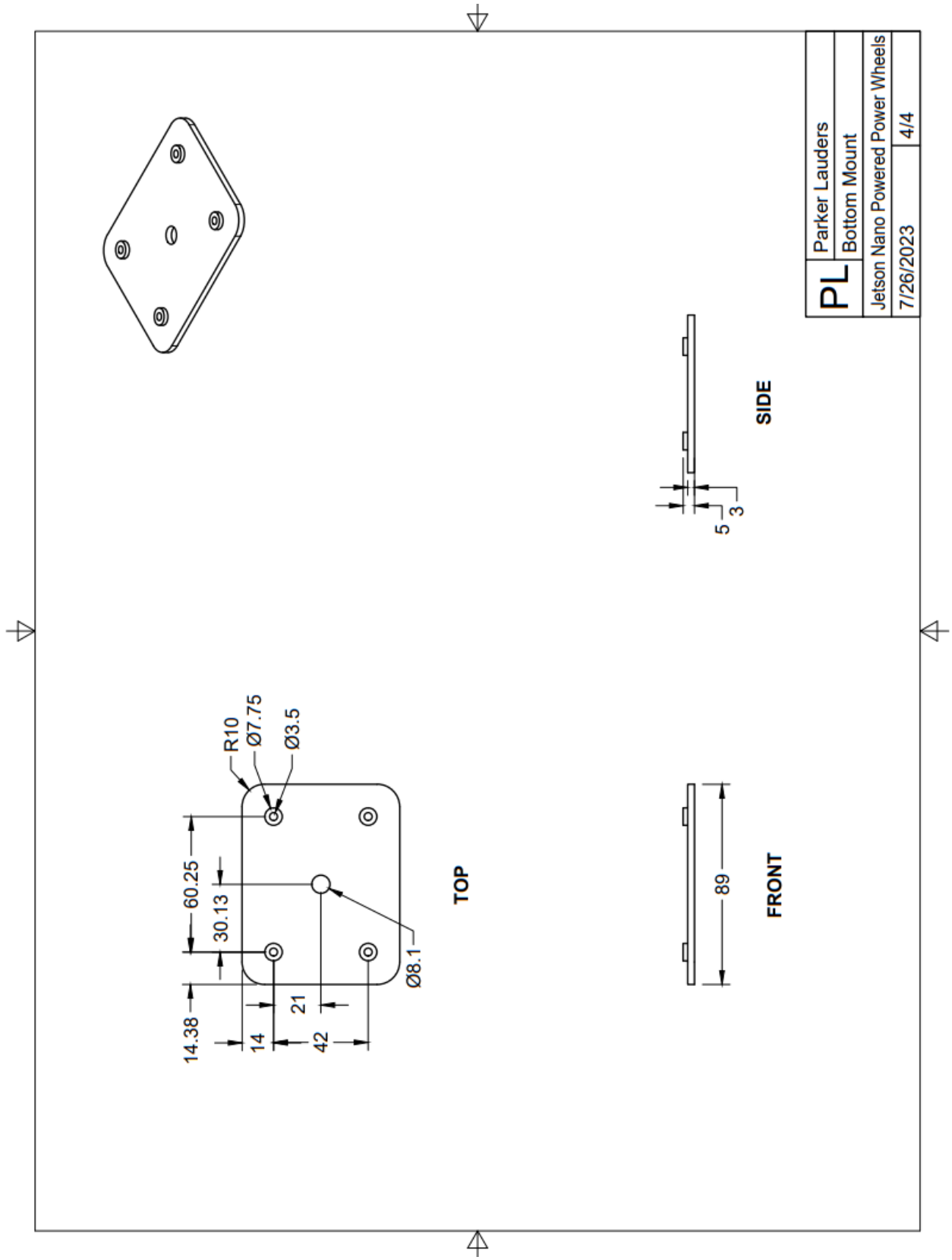


Figure 73. Logitech Camera Mount Drawing File Page 4 of 4.

Figure 74. Back Sensor Housing Drawing File Page 1 of 2.

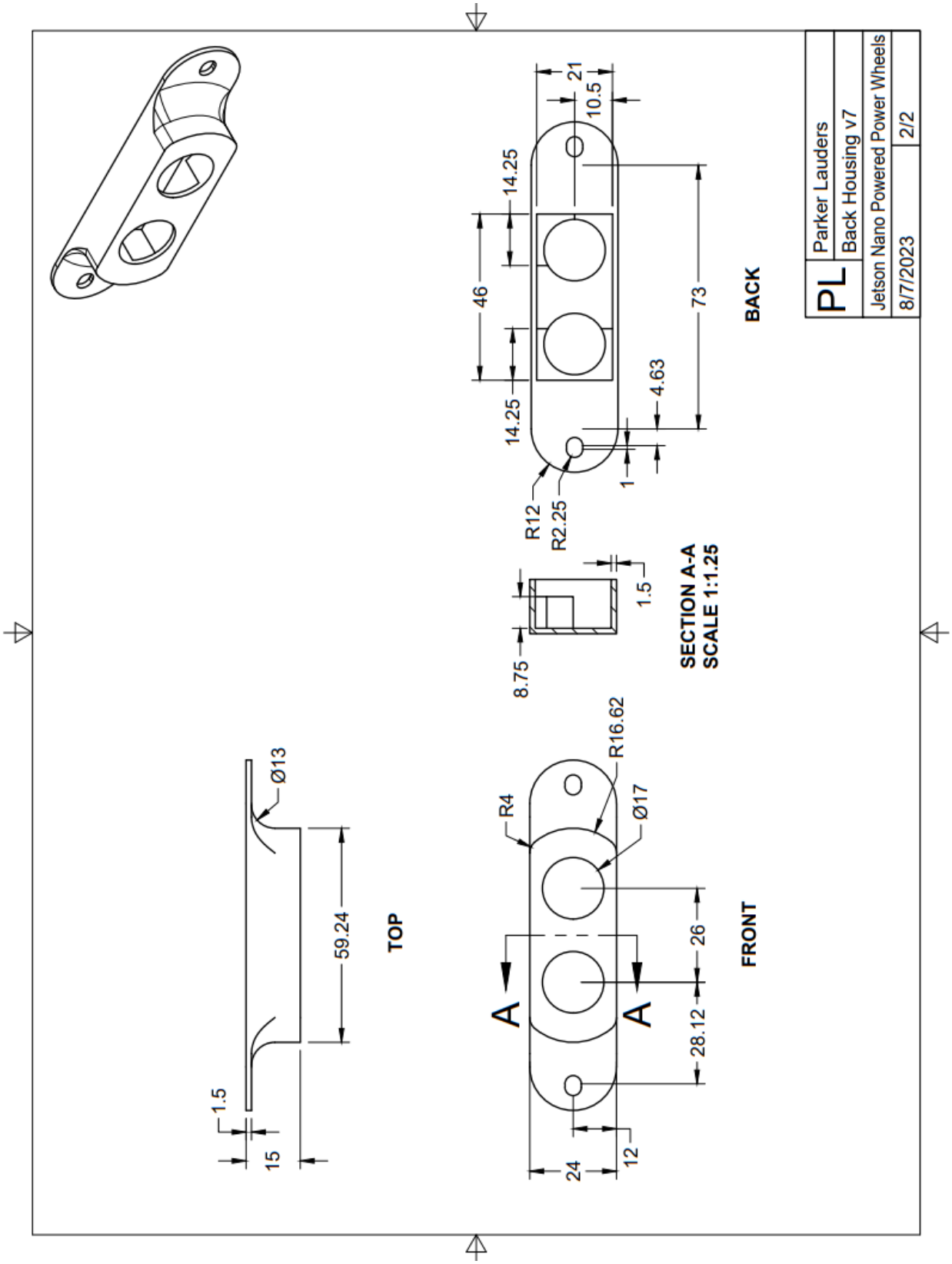


Figure 75. Back Sensor Housing Drawing File Page 2 of 2.

Figure 76. Front Sensor Housing Drawing File Page 1 of 4.

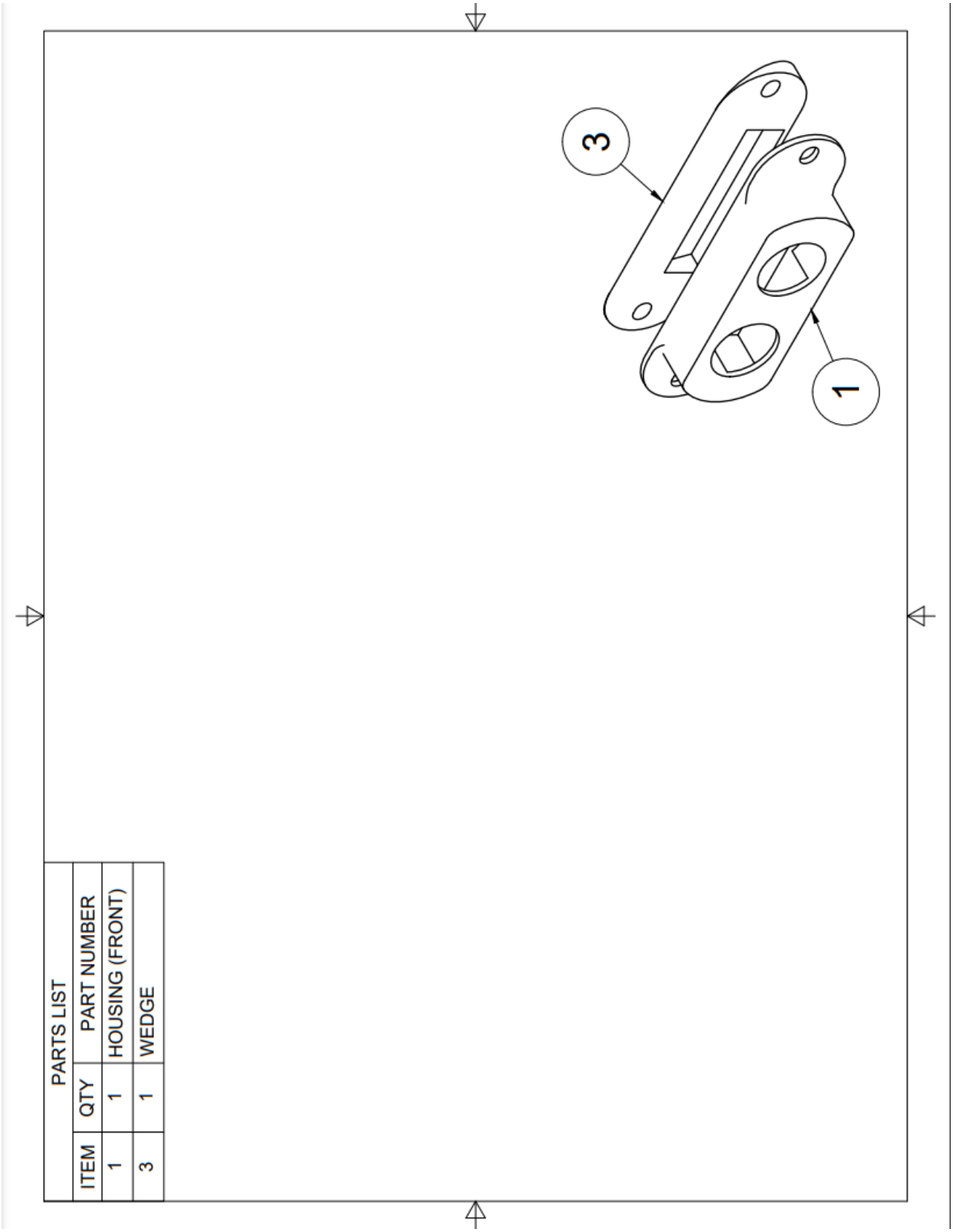


Figure 77. Front Sensor Housing Drawing File Page 2 of 4.

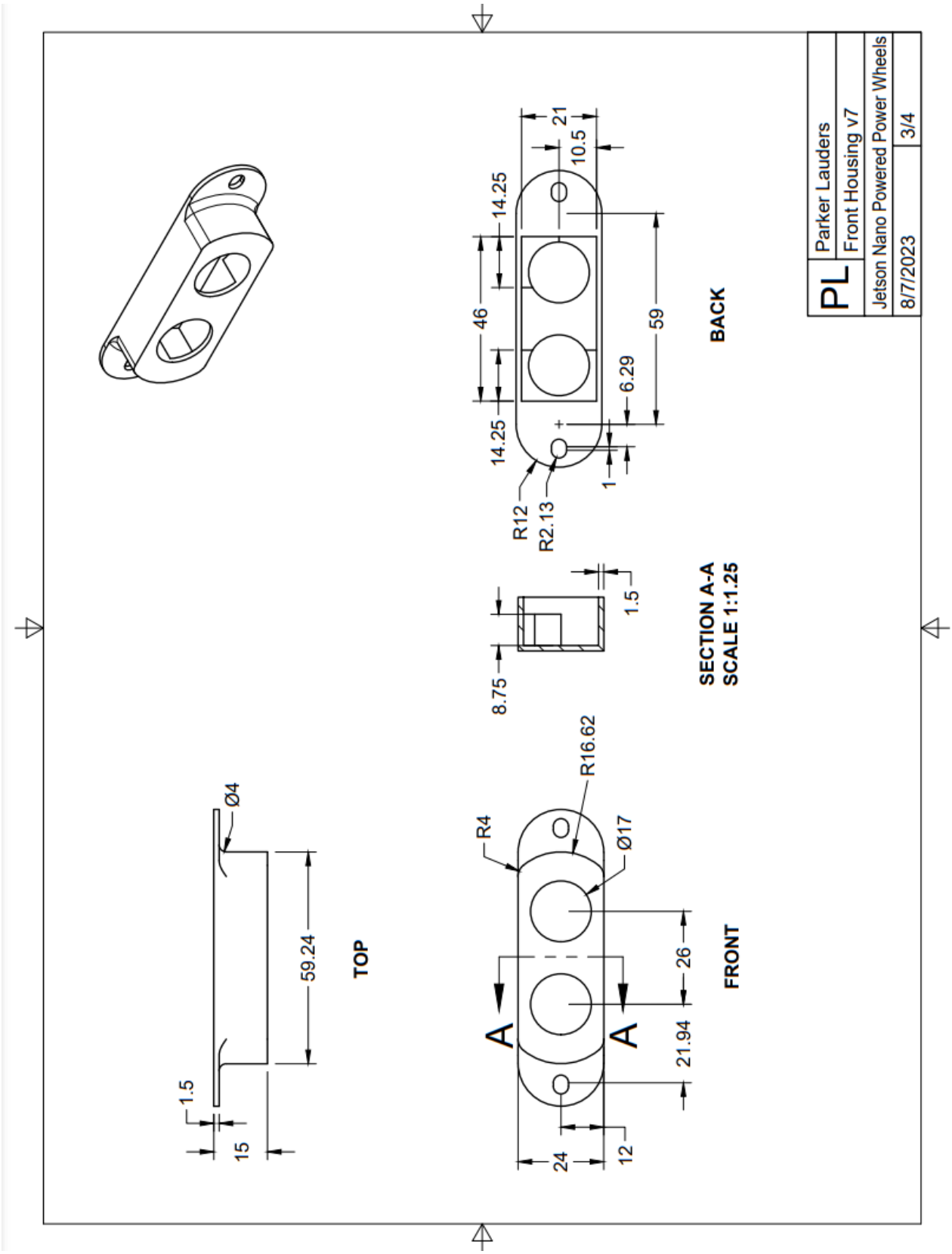


Figure 78. Front Sensor Housing Drawing File Page 3 of 4.

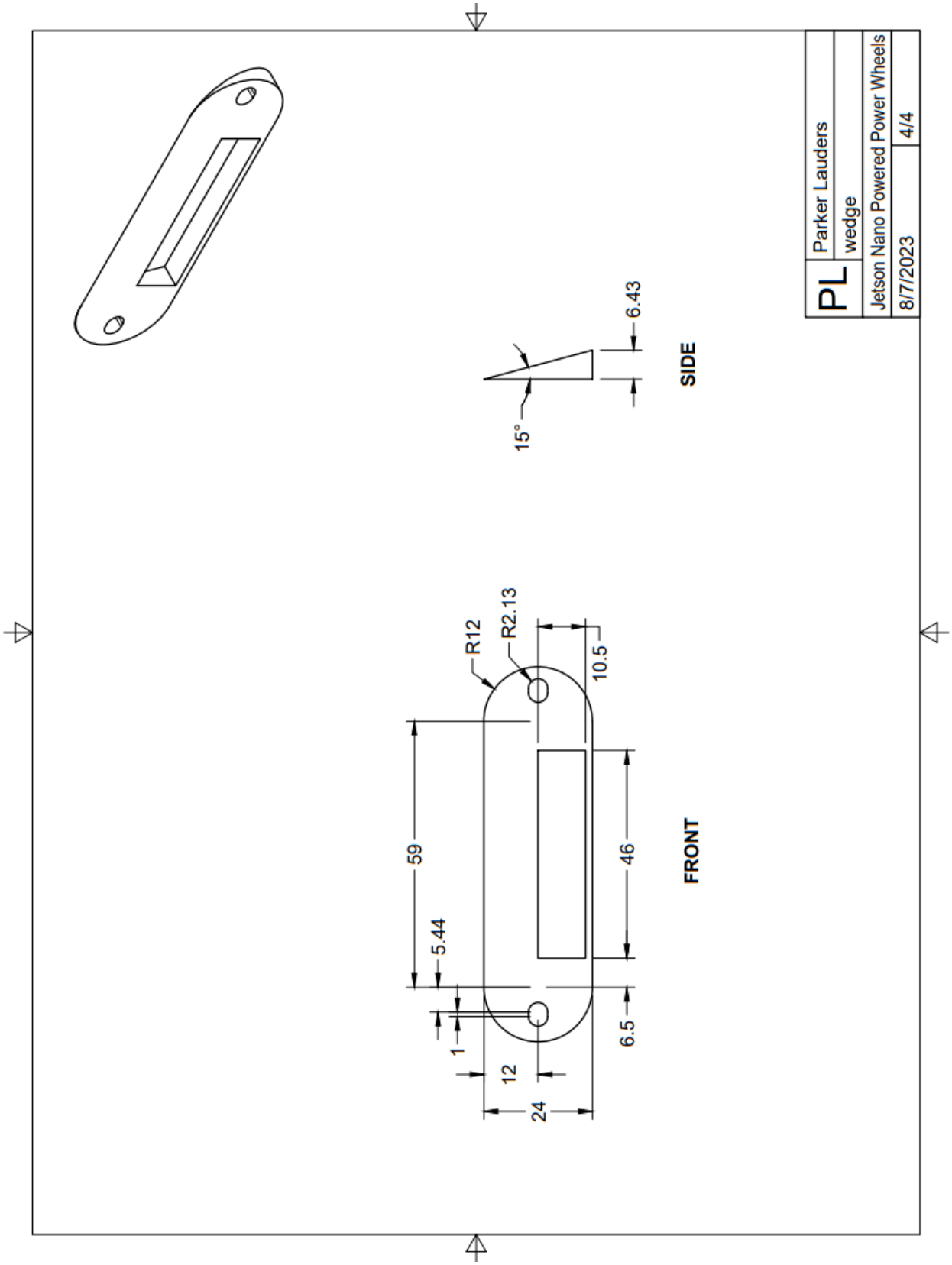


Figure 79. Front Sensor Housing Drawing File Page 4 of 4.

Appendix C: Electronics

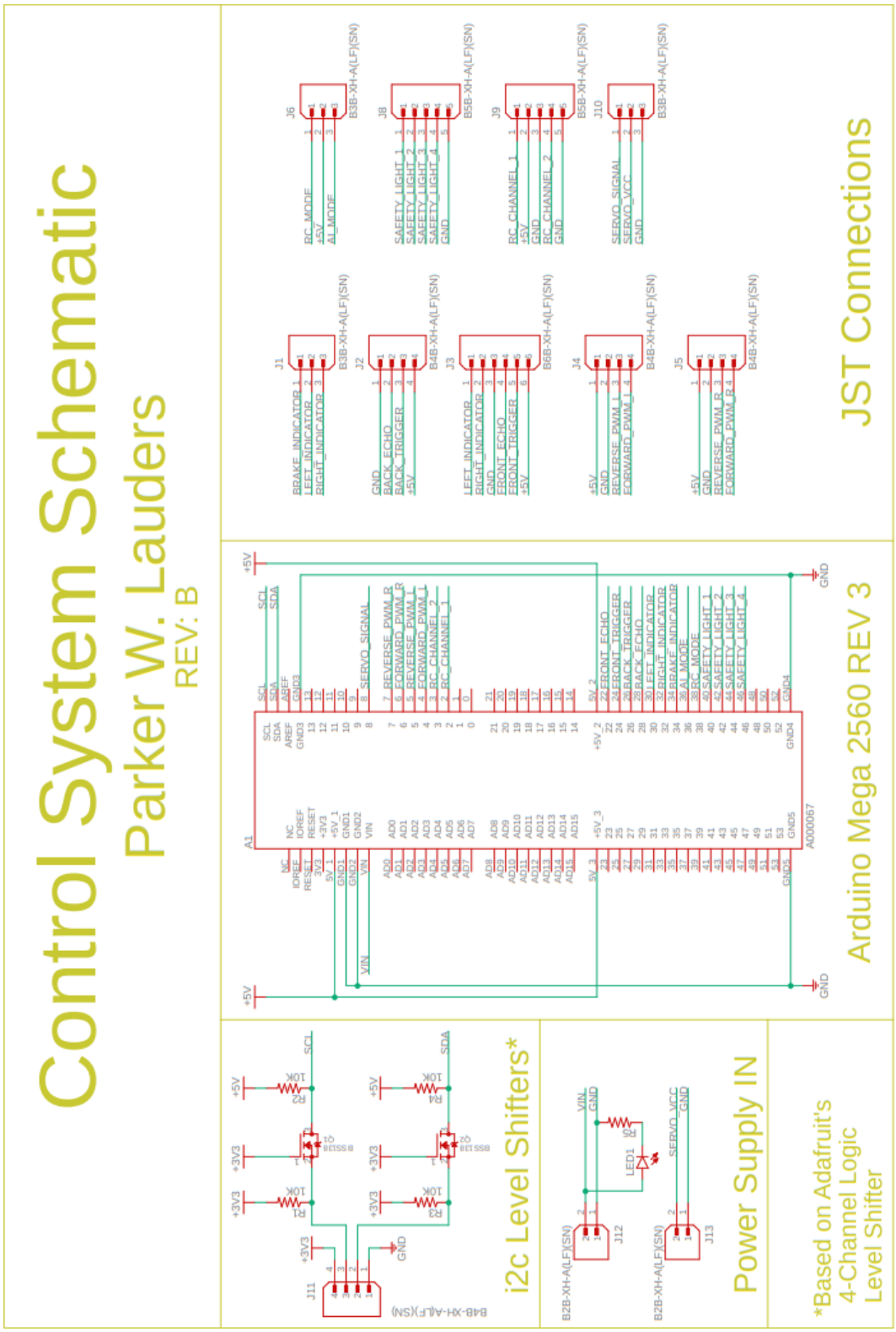


Figure 80. Control System Arduino Mega 2560 REV 3 Shield Schematic.

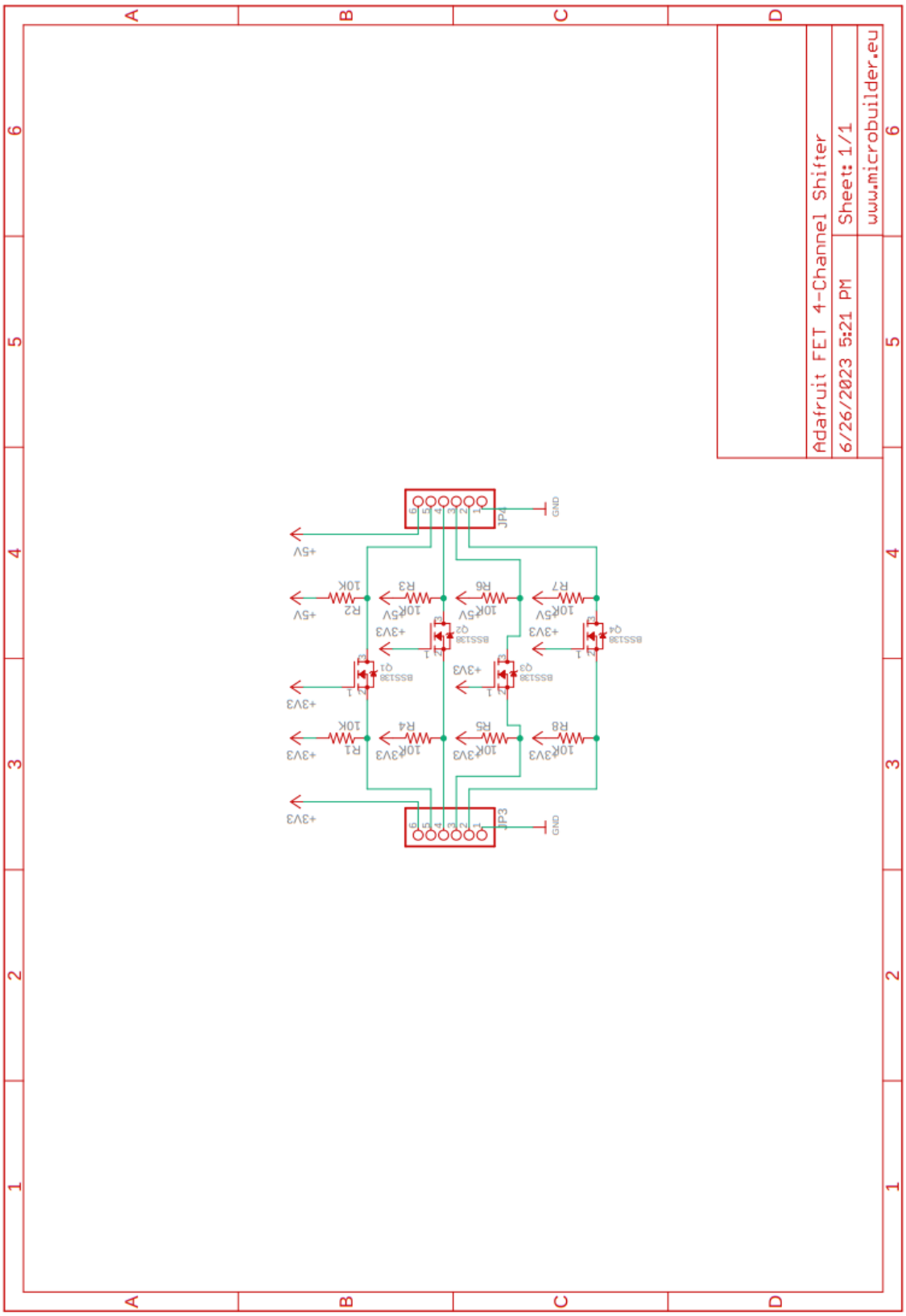


Figure 81. Adafruit 4-Channel Level Shifter Schematic.

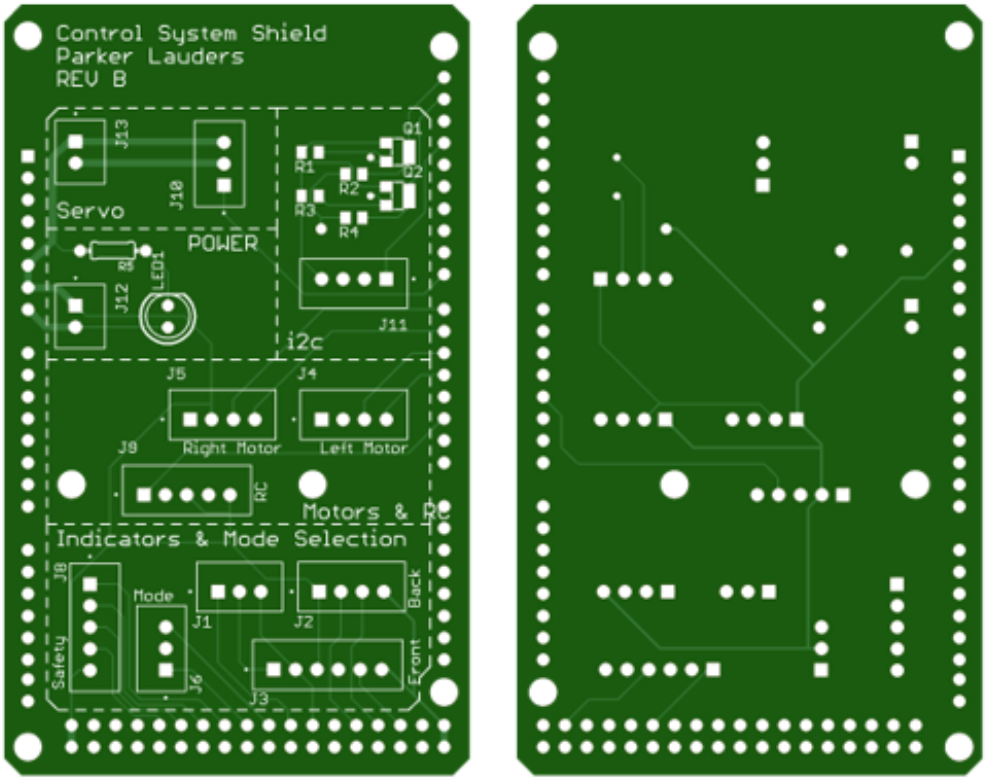


Figure 82. Control System Arduino Mega 2560 REV 3 Shield Board.

$$W\ (\text{Motor PWMs}) = \frac{\left(\frac{161\text{mA}}{0.048*10^{0.44}}\right)^{\frac{1}{0.725}}}{1.378} = 0.985\ \text{mil}$$

$$W\ (\text{RC Signals}) = \frac{\left(\frac{128\text{mA}}{0.048*10^{0.44}}\right)^{\frac{1}{0.725}}}{1.378} = 0.718\ \text{mil}$$

$$W\ (\text{Ultrasonic Sensor Signals}) = \frac{\left(\frac{15\text{mA}}{0.048*10^{0.44}}\right)^{\frac{1}{0.725}}}{1.378} = 0.037\ \text{mil}$$

$$W\ (\text{Turn Signals}) = \frac{\left(\frac{42\text{mA}}{0.048*10^{0.44}}\right)^{\frac{1}{0.725}}}{1.378} = 0.154\ \text{mil}$$

$$W\ (\text{Brake Indicator}) = \frac{\left(\frac{80\text{mA}}{0.048*10^{0.44}}\right)^{\frac{1}{0.725}}}{1.378} = 0.375\ \text{mil}$$

$$W\ (\text{Model Selection Signals}) = \frac{\left(\frac{500\text{mA}}{0.048*10^{0.44}}\right)^{\frac{1}{0.725}}}{1.378} = 4.703\ \text{mil}$$

$$W\ (\text{Safety Indicator Signals}) = \frac{\left(\frac{26\text{mA}}{0.048*10^{0.44}}\right)^{\frac{1}{0.725}}}{1.378} = 0.079\ \text{mil}$$

$$W\ (\text{Arduino \& Servo Power IN}) = \frac{\left(\frac{2}{0.048*10^{0.44}}\right)^{\frac{1}{0.725}}}{1.378} = 31.830\ \text{mil}$$

$$W\ (+5\text{V}) = \frac{\left(\frac{1.5\text{A}}{0.048*10^{0.44}}\right)^{\frac{1}{0.725}}}{1.378} = 21.404\ \text{mil}$$

$$W\ (\text{GND}) = \frac{\left(\frac{1.5\text{A}}{0.048*10^{0.44}}\right)^{\frac{1}{0.725}}}{1.378} = 21.404\ \text{mil}$$

Figure 83. Control System Arduino Mega 3560 REV 3 Shield PCB Trace Calculations.

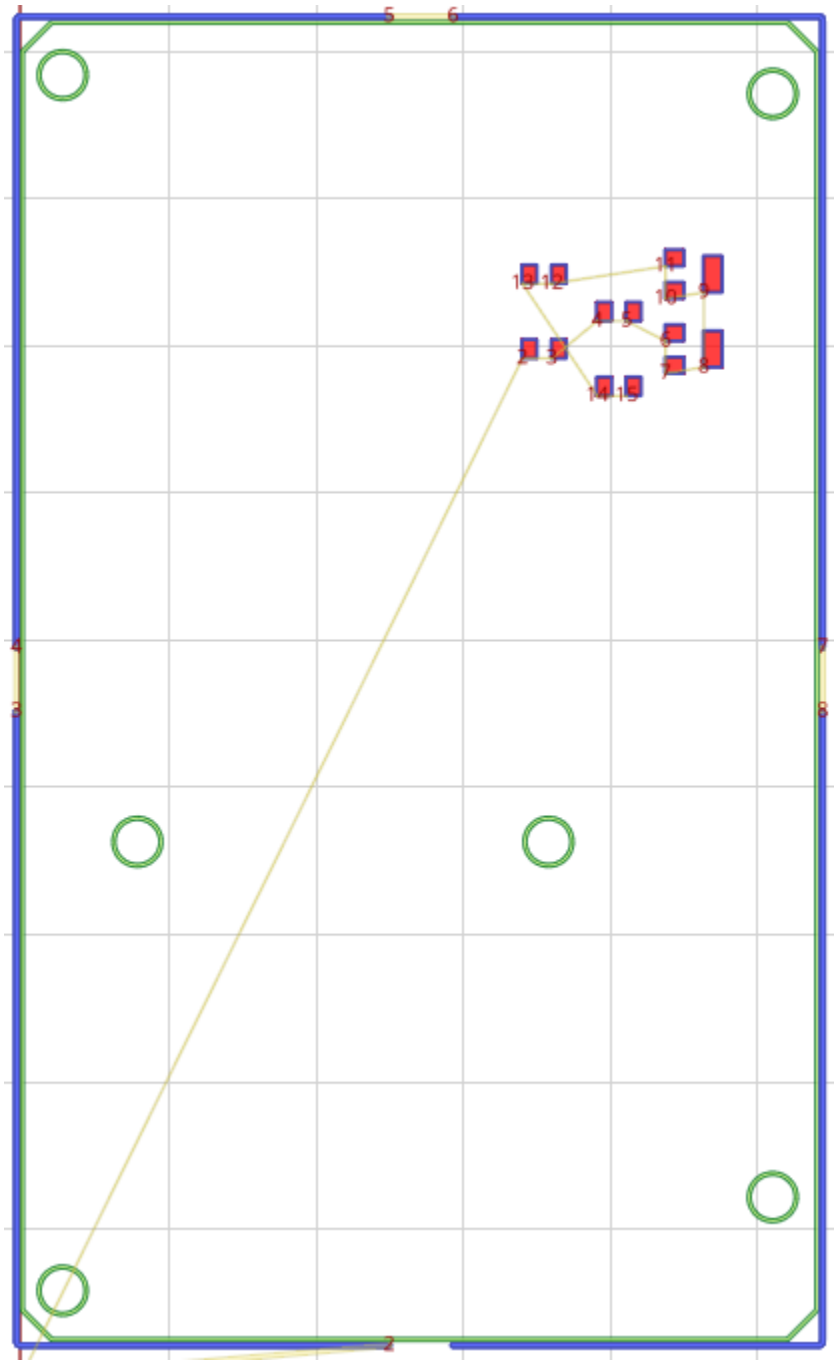


Figure 84. Control System Arduino Mega 3560 REV 3 Shield SMT Stencil CAM File.

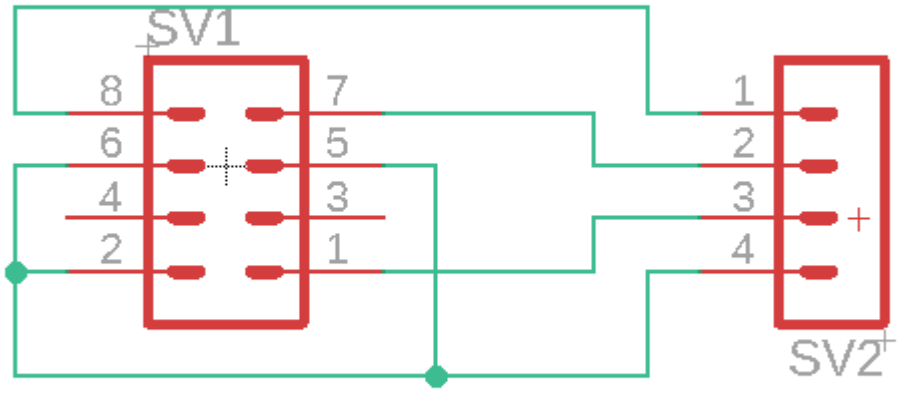


Figure 85. Motor Controller Breakout Board Schematic.

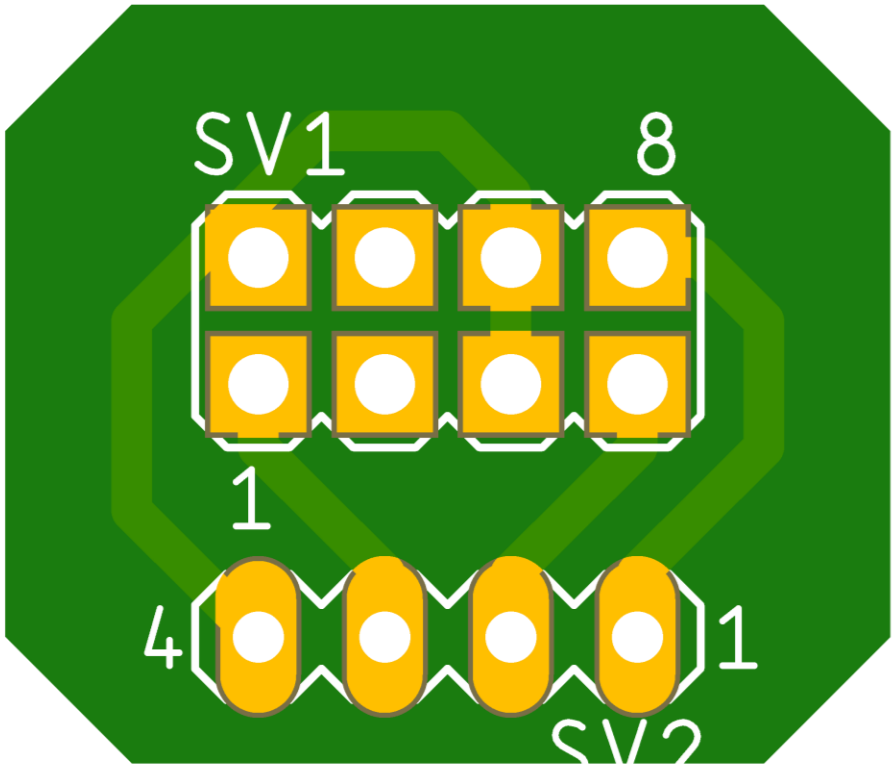


Figure 86. Motor Controller Breakout Board.

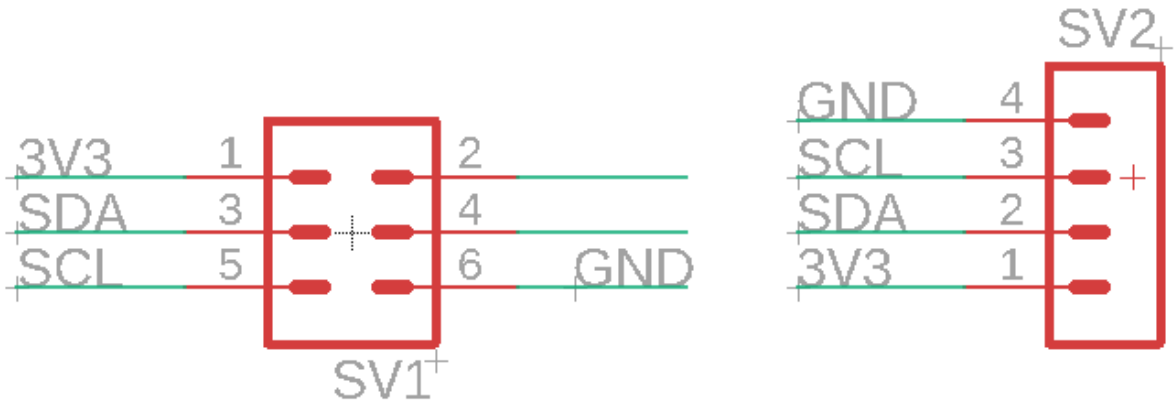


Figure 87. Jetson Nano I2C Breakout Board Schematic.

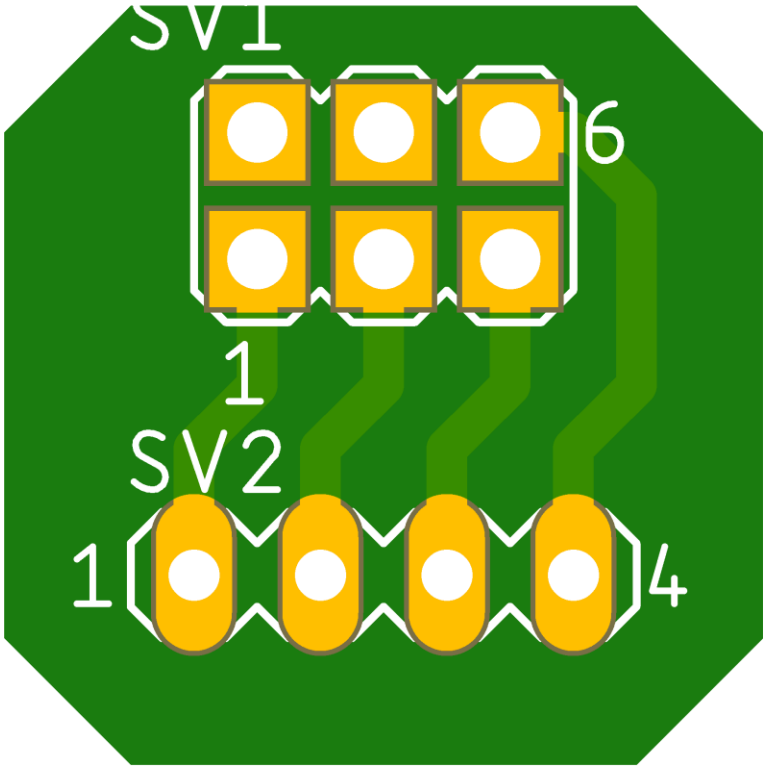


Figure 88. Jetson Nano I2C Breakout Board.

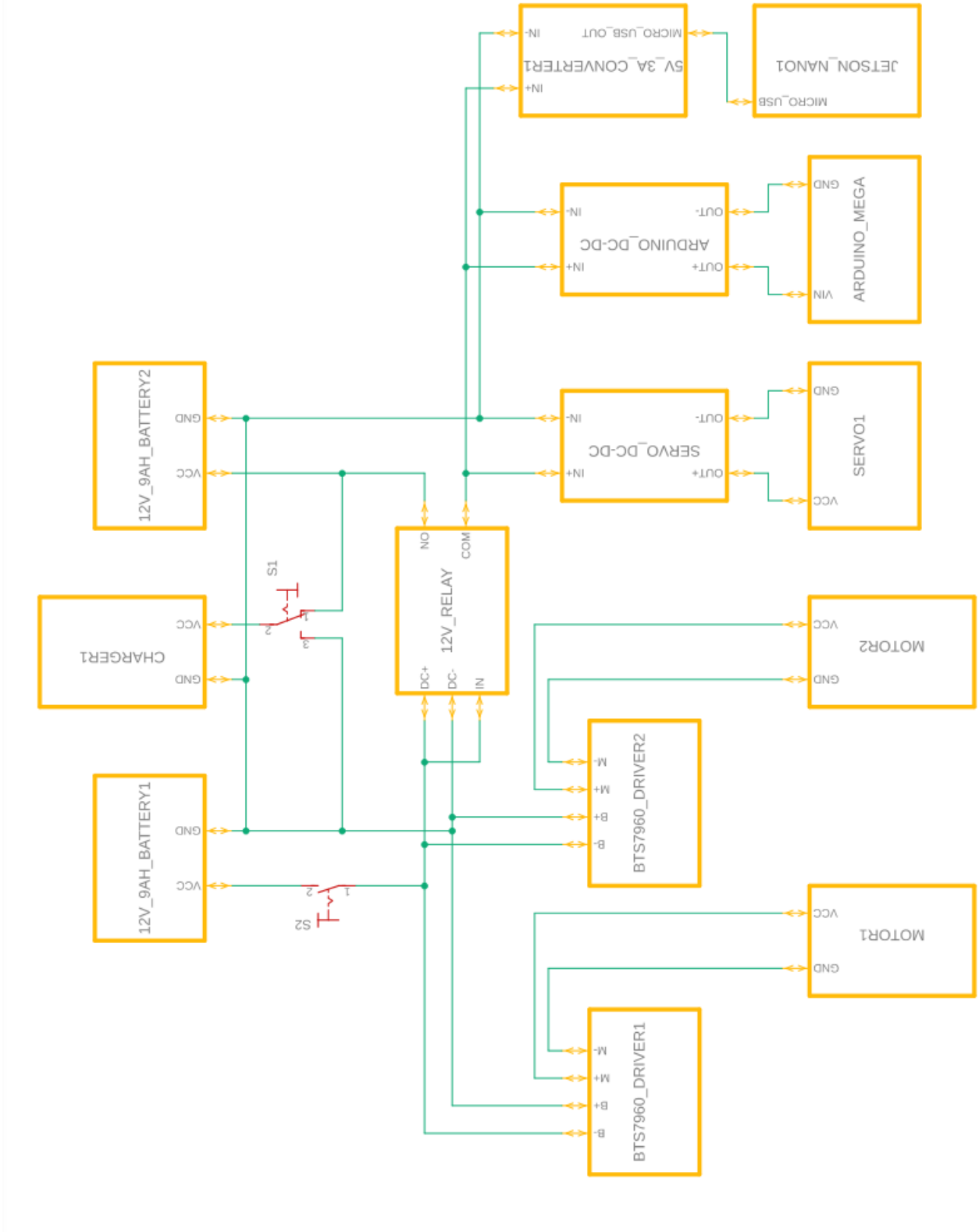


Figure 89. Battery Wiring Diagram.

Appendix D: Code

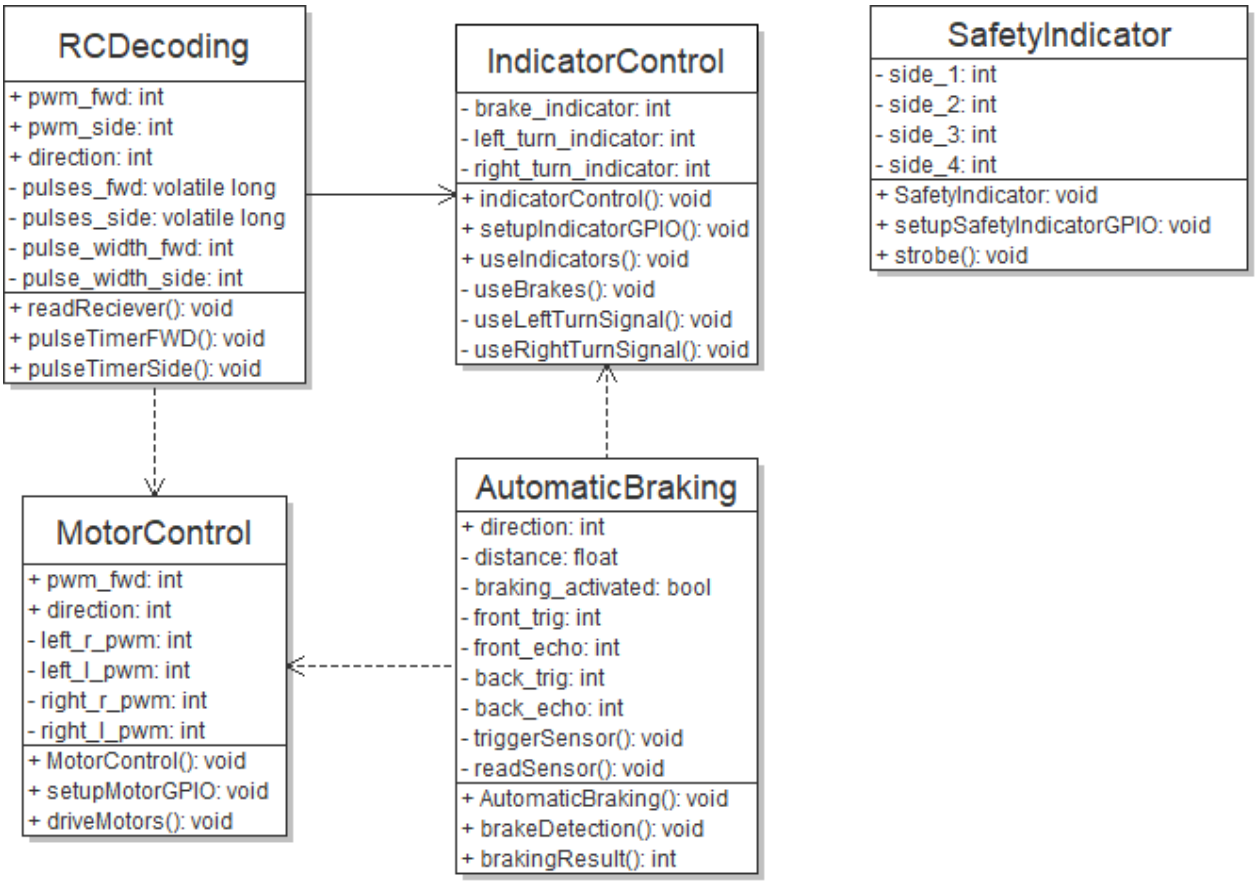


Figure 90. Control System Class Diagram.

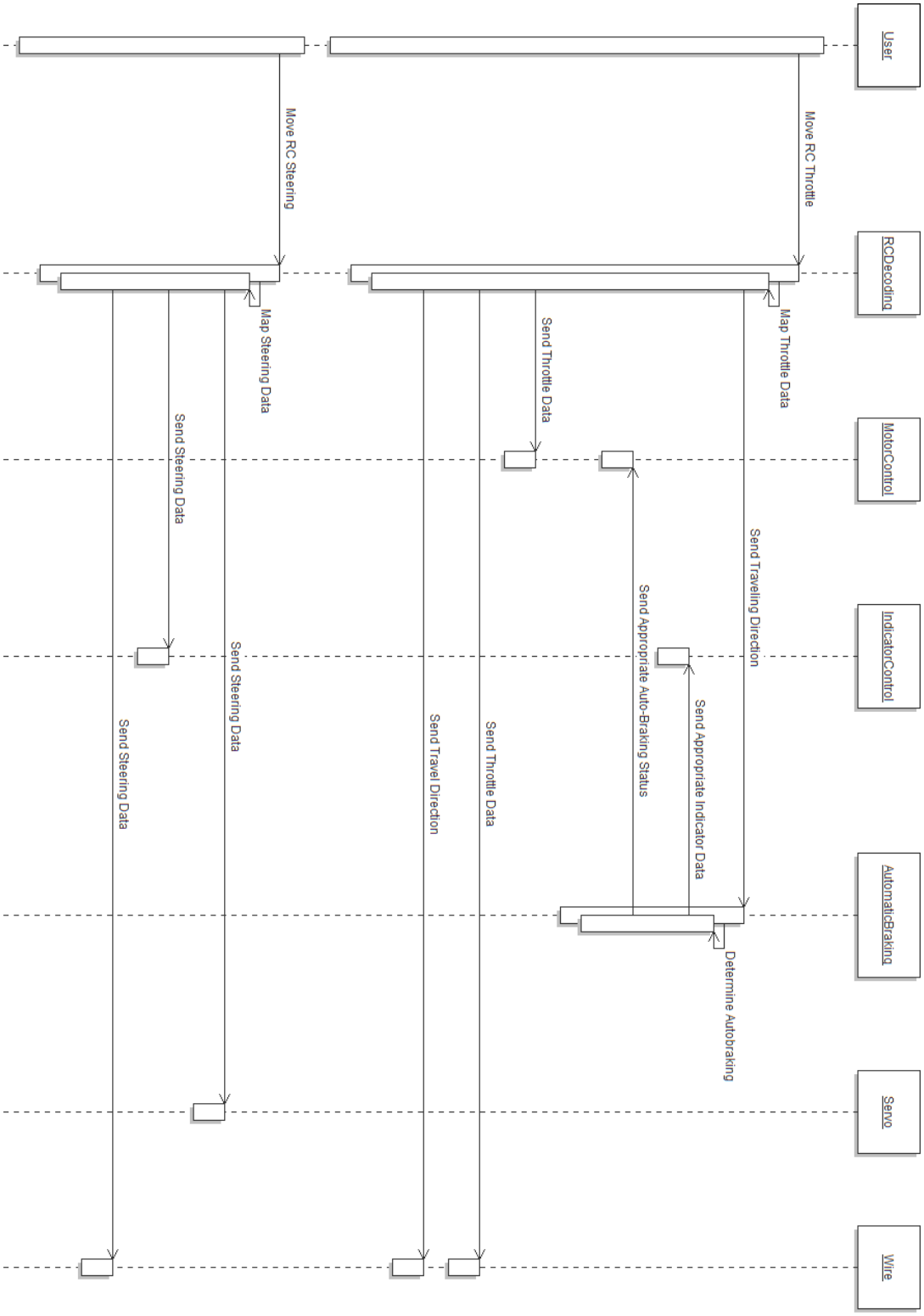


Figure 91. Control System RC Mode Sequence Diagram.

RCDecoding Class

```
#include "RCDecoding.h"
#include <Arduino.h>

// Global Variables
int pwm_fwd = 0;
int pwm_side = 0;
int direction = 0;

// Initialize Class Variables
volatile long RCDecoding::pulses_fwd = 0;
volatile long RCDecoding::pulses_side = 0;
int RCDecoding::pulse_width_fwd = 0;
int RCDecoding::pulse_width_side = 0;

// Map RC Receiver PWM Data
void RCDecoding::readReceiver() {
  // Save Pulse Lengths for FWD Controls Less Than 2000 Microseconds; Max PWM
from Receiver
  if(pulses_fwd < 2000) {
    // Save as 'pulse_width_fwd'; PWM Signal for FWD Controls
    pulse_width_fwd = pulses_fwd;

    // Map RC Receiver PWM to Usable PWM
    if((pulse_width_fwd >= 984) && (pulse_width_fwd <= 1479)) {            //
Forward Control Mapping
      // Map Data to PWM
      pwm_fwd = map(pulse_width_fwd, 984, 1479, 127, 0);

      // Set Direction Signal
      direction = 0;

    } else if((pulse_width_fwd >= 1480) && (pulse_width_fwd <= 1520)){      //
Sticks Centered Mapping; Dead Zone
      // Map Data to PWM
      pwm_fwd = map(pulse_width_fwd, 1480, 1520, 0, 0);

      // Set Direction Signal
      direction = 0;

    } else if((pulse_width_fwd >= 1521) && (pulse_width_fwd <= 1988)) {     //
Reverse Control Mapping
      // Map Data to PWMLeft
      pwm_fwd = map(pulse_width_fwd, 1521, 1988, 0, 127);

      // Set Direction Signal
```

```
      direction = 1;
    }
  }

  // Save Pulse Lengths for Side Controls Less than 2000 Microseconds; Max PWM
from Receiver
  if(pulses_side < 2000) {
    // Save as 'pulse_width_side'; PWM Signal for Steering Controls
    pulse_width_side = pulses_side;

    // Map RC Receiver PWM to Usable PWM
    if((pulse_width_side >= 996) && (pulse_width_side <= 1450)) {
      pwm_side = map(pulse_width_side, 996, 1450, 140, 91);                //
Left Steering Mapping
    } else if((pulse_width_side >= 1450) && (pulse_width_side <= 1520)) {
      pwm_side = map(pulse_width_side, 1450, 1520, 90, 90);                //
Sticks Centered Mapping; Dead Zone
    } else if((pulse_width_side >= 1521) && (pulse_width_side <= 1996)) {
      pwm_side = map(pulse_width_side, 1521, 1996, 89, 75);                //
Right Steering Mapping
    } else {
      pwm_side = 90;
    }
  }
}

// Function for Getting RC Receiver PWM for Throttle (Channel 2)
void RCDecoding::pulseTimerFWD() {
  // Instance Variables
  volatile long current_time = micros();      // Runtime of Arduino Mega 2560
REV 3
  static unsigned long previous_time = 0;     // Previous Time Stamp of Last
Completion

  // Get Pulses
  if(current_time - previous_time) {
     pulses_fwd = current_time - previous_time;
     previous_time = current_time;
  }
}

// Function for Getting RC Receiver PWM for Steering (Channel 1)
void RCDecoding::pulseTimerSide() {
  // Instance Variables
  volatile long current_time = micros();      // Runtime of Arduino Mega 2560
REV 3
```

```
  static unsigned long previous_time = 0;    // Previous Time Stamp of Last
Completion

  // Get Pulses
  if(current_time - previous_time) {
    pulses_side = current_time - previous_time;
    previous_time = current_time;
  }
}
```

MotorControl Class

```
#include "MotorControl.h"
#include <Arduino.h>

// Class Constructor
MotorControl::MotorControl(int  left_pwm_forward,  int  left_pwm_reverse,  int
right_pwm_forward, int right_pwm_reverse) {
  left_r_pwm = left_pwm_forward;
  left_l_pwm = left_pwm_reverse;
  right_r_pwm = right_pwm_forward;
  right_l_pwm = right_pwm_reverse;
}

// Setup Motor
void MotorControl::setupMotorGPIO() {
  // Set GPIO Outputs
  pinMode(left_r_pwm, OUTPUT);
  pinMode(left_l_pwm, OUTPUT);
  pinMode(right_r_pwm, OUTPUT);
  pinMode(right_l_pwm, OUTPUT);
}

// Control Motors Depending on 'pwm_data' and 'direction' Received
void MotorControl::driveMotors(int PWM_data, int direction) {
  // Send 'PWM_data' to Motors
  if(direction == 0) {
    // Clear Reverse Signal
    analogWrite(left_l_pwm, 0);
    analogWrite(right_l_pwm, 0);

    // Send Forward Signal
    analogWrite(left_r_pwm, PWM_data);
    analogWrite(right_r_pwm, PWM_data);
  } else {
    // Clear Forward Signal
```

```
    analogWrite(left_r_pwm, 0);
    analogWrite(right_r_pwm, 0);

    // Send Reverse Signal
    analogWrite(left_l_pwm, PWM_data);
    analogWrite(right_l_pwm, PWM_data);
  }
}
```

IndicatorControl Class

```
#include "IndicatorControl.h"
#include <Arduino.h>

// Class Constructor
IndicatorControl::IndicatorControl(int brake pin, int left_turn_pin, int
right_turn_pin) {
  brake_indicator = brake_pin;
  left_turn_indicator = left_turn_pin;
  right_turn_indicator = right_turn_pin;
}

// Setup and Test Indicators
void IndicatorControl::setupIndicatorGPIO() {
  // Set GPIO Outputs
  pinMode(brake_indicator, OUTPUT);
  pinMode(left_turn_indicator, OUTPUT);
  pinMode(right_turn_indicator, OUTPUT);

  // Test Brake Indicator
  // Delay Will Not Affect the Program Once Setup is Complete
  useBrakes(true);
  delay(1000);
  useBrakes(false);

  // Test Left Turn Indicator
  // Delay Will Not Affect the Program Once Setup is Complete
  for(int i = 0; i < 1000; i++) {
    useLeftTurnSignal(true);
    delay(2.5);
  }

  // Disable Turn Signal
  useLeftTurnSignal(false);

  // Test Right Turn Indicator
```

```
  // Delay Will Not Affect the Program Once Setup is Complete
  for(int i = 0; i < 1000; i++) {
    useRightTurnSignal(true);
    delay(2.5);
  }

  // Disable Turn Signal
  useRightTurnSignal(false);
}

// Turn ON/OFF Brakes Depending on State of 'brake_status'
void IndicatorControl::useBrakes(bool brake_status) {
  // Turn ON/OFF Braking Indicators Based Upon 'brake_status'
    brake_status   ==   true   ?   digitalWrite(brake_indicator,   HIGH)   :
digitalWrite(brake_indicator, LOW);
}

// Turn ON/OFF Left Turn Signal Depending on State of 'left_turn_status'
void IndicatorControl::useLeftTurnSignal(bool left_turn_status) {
  // Instance Variables
  unsigned long current_time = millis();    // Runtime for Arduino 2560
  static unsigned long previous_time = 0;   // Timestamp for Last Preformed
Routine
  const long BLINK_INTERVAL = 400;          // Amount of Time Between Blinks
  static bool blink_state = false;          // Toggles if Indicator is ON/OFF

  // Blink Indicator if 'left_turn_status' is 'true'
  if(left_turn_status == 1) {
    // Check if 'BLINK_INTERVAL' Amount of Time Has Passed
    if(current_time - previous_time >= BLINK_INTERVAL) {
      blink_state = !blink_state;
      digitalWrite(left_turn_indicator, blink_state ? HIGH : LOW);
      previous_time = current_time;
    }
  } else {
    digitalWrite(left_turn_indicator, LOW);
  }
}

// Turn ON/OFF Right Turn Signal Depending on State of 'right_turn_status'
void IndicatorControl::useRightTurnSignal(bool right_turn_status) {
  // Instance Variables
  unsigned long current_time = millis();    // Runtime for Arduino 2560
  static unsigned long previous_time = 0;   // Timestamp for Last Preformed
Routine
  const long BLINK_INTERVAL = 400;          // Amount of Time Between Blinks
```

```
  static bool blink_state = false;          // Toggles if Indicator is ON/OFF

  // Blink Indicator if 'left_turn_status' is 'true'
  if(right_turn_status == 1) {
    // Check if 'BLINK_INTERVAL' Amount of Time Has Passed
    if(current_time - previous_time >= BLINK_INTERVAL) {
      blink_state = !blink_state;
      digitalWrite(right_turn_indicator, blink_state ? HIGH : LOW);
      previous_time = current_time;
    }
  } else {
    digitalWrite(right_turn_indicator, LOW);
  }
}

// Checks for Required Actions to Indicators
void IndicatorControl::useIndicators(int motor_direction, int steering_angle)
{
  // Determine if Brakes are Needed
  motor_direction == 0 ? useBrakes(0) : useBrakes(1);

  // Determine if Left Turn Indicators are Needed
  steering_angle > 128 ? useLeftTurnSignal(true) : useLeftTurnSignal(false);

  // Determine if Right Turn Indicators are Needed
  steering_angle < 79 ? useRightTurnSignal(true) : useRightTurnSignal(false);
}
```

AutomaticBraking Class

```
#include "AutomaticBraking.h"
#include <Arduino.h>

// Class Variables
static float AutomaticBraking::distance = 0;
static bool AutomaticBraking::braking_activated = false;

// Class Constructor
AutomaticBraking::AutomaticBraking(int front_trig_pin, int front_echo_pin, int
back_trig_pin, int back_echo_pin) {
  front_trig = front_trig_pin;
  front_echo = front_echo_pin;
  back_trig = back_trig_pin;
  back_echo = back_echo_pin;
}
```

```
// Triggers HC-SR04 Ultrasonic Sensor
// 'sensor_trig_pin' Determines Which Sensor to Trigger
void AutomaticBraking::triggerSensor(int sensor_trig_pin) {
  pinMode(sensor_trig_pin, OUTPUT);
  digitalWrite(sensor_trig_pin, LOW);
  delayMicroseconds(2);
  digitalWrite(sensor_trig_pin, HIGH);
  delayMicroseconds(10);
  digitalWrite(sensor_trig_pin, LOW);
}

// Obtains Data from HC-SR04 Ultrasonic Sensor
void AutomaticBraking::readSensor(int sensor_trig_pin, int sensor_echo_pin) {
  // Instance Variables
  unsigned long current_time = millis();
  static unsigned long previous_time = 0;
  const long SCAN_INTERVAL = 100;
  float duration = 0;

  // Check if 'SCAN_INTERVAL' of Time Has Passed
  if(current_time - previous_time >= SCAN_INTERVAL) {
    // Trigger the HC-SR04 Sensor
    direction == 0 ? triggerSensor(front_trig) : triggerSensor(back_trig);

    // Read Echo
    duration = pulseIn(sensor_echo_pin, HIGH);
    distance = (duration * 0.0344555) / 2; // '0.0344555' May Give off Distance
Measures Depending on Different HC-SR04 Sensors

    // Save Millis
    previous_time = current_time;
  }
}

// Determine if Braking Needs to be Activated Based on HC-SR04 Ultrasonic Sensor
Data
void AutomaticBraking::brakeDetection(int direction) {
  // Instance Variables
  unsigned long current_time = millis();
  static unsigned long previous_time = 0;
  const long BRAKING_INTERVAL = 5000;
  const float SAFETY_ZONE = 914.4;

  // Read Sensor
  direction == 0 ? readSensor(front_trig, front_echo) : readSensor(back_trig,
back_echo);
```

```
  // Check for Objects in the 'SAFETY_ZONE' range
  if(distance <= SAFETY_ZONE) {
    braking_activated = true;
    previous_time = current_time;
  } else {
    if(current_time - previous_time >= BRAKING_INTERVAL) {
    braking_activated = false;
    previous_time = current_time;
    }
  }
}

// Return the Value of 'braking_activated'
int AutomaticBraking::brakingResult() {
  return braking_activated == true ? 1 : 0;
}
```

SafetyIndicator Class

```
#include "SafetyIndicator.h"
#include <Arduino.h>

// Class Variables
static int currentLed = 0;
unsigned long previous_time = 0;

// Class Constructor
SafetyIndicator::SafetyIndicator(int LED_1, int LED_2, int LED_3, int LED_4) {
  LED1 = LED_1;
  LED2 = LED_2;
  LED3 = LED_3;
  LED4 = LED_4;
}

// Set GPIO Outputs
void SafetyIndicator::setupSafetyIndicatorGPIO() {
  pinMode(LED1, OUTPUT);
  pinMode(LED2, OUTPUT);
  pinMode(LED3, OUTPUT);
  pinMode(LED4, OUTPUT);
}

// Strobe Function
void SafetyIndicator::strobe(bool safety_indicator_status) {
  if(safety_indicator_status == true) {
```

```
    // Get Current Runtime
    current_time = millis();

    // Set Interval of On Period
    const unsigned long interval = 100;

    // Turn Off the Previously Active LED
    digitalWrite(LED1, LOW);
    digitalWrite(LED2, LOW);
    digitalWrite(LED3, LOW);
    digitalWrite(LED4, LOW);

    // Determine the Current LED to Activate
    switch(currentLed) {
      case 0:
        digitalWrite(LED1, HIGH);
        break;
      case 1:
        digitalWrite(LED2, HIGH);
        break;
      case 2:
        digitalWrite(LED3, HIGH);
        break;
      case 3:
        digitalWrite(LED4, HIGH);
        break;
    }

    // Update the Current LED Based on Interval Timing
    if((current_time - previous_time) >= interval) {
      currentLed = (currentLed + 1) % 4;
      previous_time = current_time;
    }
  } else {
    // Turn Off the Previously Active LED
    digitalWrite(LED1, LOW);
    digitalWrite(LED2, LOW);
    digitalWrite(LED3, LOW);
    digitalWrite(LED4, LOW);
  }
}
```

Control System Main Program

```
/*
    Arduino Mega 2560 Rev 3 Code
     Purpose: All-in-one motor controller, automatic braking, and indicator
system

    Author: Parker Lauders
    Date: 5/31/2023

    Version: 4.6.0
    Updated: 8/10/2023 1:40 PM EST

*/

// Include Classes
#include "RCDecoding.h"
#include "IndicatorControl.h"
#include "MotorControl.h"
#include "AutomaticBraking.h"
#include "SafetyIndicator.h"

// Include Libraries
#include <Servo.h>
#include <Wire.h>

// GPIO Pin Declaration
const int RC_CHANNEL_1 = 2;       // Signal from the RC Receiver for Channel 1
const int RC_CHANNEL_2 = 3;       // Signal from the RC Receiver for Channel 2
const int LEFT_R_PWM = 4;         // Left BTS7960 Forward PWM
const int LEFT_L_PWM = 5;         // Left BTS7960 Reverse PWM
const int RIGHT_R_PWM = 6;        // Right BTS9760 Forward PWM
const int RIGHT_L_PWM = 7;        // Right BTS7960 Reverse PWM
const int FRONT_ECHO_PIN = 22;    // Front HC-SR04 Module's Echo Pin
const int FRONT_TRIG_PIN = 24;    // Front HC-SR04 Module's Trigger Pin
const int BACK_TRIG_PIN = 26;     // Back HC-SR04 Module's Trigger Pin
const int BACK_ECHO_PIN = 28;     // Back HC-SR04 Module's Echo Pin
const int LEFT_INDICATOR = 31;    // Left Turn Indicator
const int RIGHT_INDICATOR = 32;   // Right Turn Indicator
const int BRAKE_INDICATOR = 35;   // Brake Indicator
const int AI_MODE = 36;          // Signal for AI Mode; Allows AI Functionality
While Taking Inputs from the Jetson Nano via Communication Bit's
const int RC_MODE = 38;          // Signal for RC Mode; Allows RC Functionality
While Outputting to Jetson Nano Via Communication Bit's
const int SAFETY_LIGHT_1 = 40;    // Signal Light for AI Mode Activation
const int SAFETY_LIGHT_2 = 42;    // Signal Light for AI Mode Activation
const int SAFETY_LIGHT_3 = 44;    // Signal Light for AI Mode Activation
```

```
const int SAFETY_LIGHT_4 = 46;    // Signal Light for AI Mode Activation

// Create Instance for Classes
Servo STEERING_SERVO;
RCDecoding RC;
IndicatorControl Indicator(BRAKE_INDICATOR, LEFT_INDICATOR, RIGHT_INDICATOR);
MotorControl Motor(LEFT_R_PWM, LEFT_L_PWM, RIGHT_R_PWM, RIGHT_L_PWM);
AutomaticBraking  AutoBraking(FRONT_TRIG_PIN,  FRONT_ECHO_PIN,  BACK_TRIG_PIN,
BACK_ECHO_PIN);
SafetyIndicator     si(SAFETY_LIGHT_1,     SAFETY_LIGHT_2,     SAFETY_LIGHT_3,
SAFETY_LIGHT_4);

// Initialize I2C Variables
byte throttle_byte = pwm_fwd;
byte steering_byte = pwm_side;
bool direction_bit = direction;

// Create A.I. Variables
byte AI_throttle = 0;
byte AI_steering = 90;  // Center Value
bool AI_direction = 0;
unsigned long i2c_current = 0;
unsigned long i2c_previous = 0;
bool killswitch = false;

void setup() {
  // Initialize Serial Communication (used for debugging)
  Serial.begin(9600);

  // Initialize i2c Protocol Communication
  Wire.begin(0x04); // Changeable Address, Adjust in Python Programs for Proper
Function!

  // Link Data Sending Function for Data Requests
  Wire.onRequest(sendData);

  // Link Data Reading Function for Data Receiving
  Wire.onReceive(receiveData);

  // Setup Input GPIO Pins (not controlled by a class)
  pinMode(AI_MODE, INPUT);
  pinMode(RC_MODE, INPUT);
  pinMode(RC_CHANNEL_1, INPUT_PULLUP);
  pinMode(RC_CHANNEL_2, INPUT_PULLUP);

  // Setup External Interrupts
```

```
    attachInterrupt(digitalPinToInterrupt(RC_CHANNEL_1),   RC.pulseTimerSide,
CHANGE);
    attachInterrupt(digitalPinToInterrupt(RC_CHANNEL_2),   RC.pulseTimerFWD,
CHANGE);

  // Setup Motors
  Motor.setupMotorGPIO();

  // Setup Indicators
  Indicator.setupIndicatorGPIO();

  // Set Servo to Pin 8
  STEERING_SERVO.attach(8);

  // Center Steering Servo
  STEERING_SERVO.write(90);

  // Set Up AI Safety Indicator
  si.setupSafetyIndicatorGPIO();
}

void loop() {
  // Determine What Functionality to Call Based on the Position of the Mode
Switch
   if(((digitalRead(AI_MODE) == LOW) && (digitalRead(RC_MODE) == LOW)) ||
((digitalRead(AI_MODE) == HIGH) && (digitalRead(RC_MODE) == HIGH))) {
    // Disable Indicators
    Indicator.useIndicators(0, 90);

    // Disable Motors
    Motor.driveMotors(0, 0);

    // Disable Safety Indicator
    si.strobe(false);

  } else if(digitalRead(AI_MODE) == HIGH) {
    // Activate AI Safety Strobe Light
    si.strobe(true);

    // Call the AI Function
    AIFunctionality();
  } else if(digitalRead(RC_MODE) == HIGH) {
    // Disable AI Safety Strobe Light
    si.strobe(false);

    // Call the RC Function
```

```
    RCFunctionality();
  }
}

void AIFunctionality() {
  // Map RC PWM Data for Throttle and Steering
  RC.readReceiver();

  // Fail Safe Check
  if((pwm_fwd > 64) && (pwm_side > 105) && (direction == 1)) {
    killswitch = true;
  } else if((pwm_fwd > 64) && (pwm_side < 83) && (direction == 0)) {
    killswitch = false;
  }

  // Disable Power Wheels if Kill Switch is Active
  if(killswitch) {
    Motor.driveMotors(0, AI_direction);
    Indicator.useIndicators(1, 90);
    STEERING_SERVO.write(90);   // Center
  } else {
    // Check if the I2C Data Stopped
    if(Wire.available() == false) {
      i2c_current = millis();
    }

    // See if I2C has Been Timed Out Longer than 1.5 Seconds
    if(i2c_current - i2c_previous >= 1500) {
      AI_direction = 1;
      AI_throttle = 0;
      AI_steering = 90;
    } else {
      // Check Auto Braking
      AutoBraking.brakeDetection(AI_direction);

      // Save the Result of 'brakingResult'
      bool result = AutoBraking.brakingResult();

      // Control Indicators Based on Value of 'result'
      // Value '1': Activate Brake Indicator
      // Value '0': Activate Indicators Based on Regular Input
           result   ==   1   ?   Indicator.useIndicators(1,   AI_steering)   :
Indicator.useIndicators(AI_direction, AI_steering);

      // Act Upon Returned Value of 'brakeDetection'
```

```
      // Value 1: Send PWM Value of 0 (Stopped) and Set Motor Direction to
Reverse (Activates Brake Lights)
      // Value 0: Send PWM Speed Data and Set Motor Direction
            result   ==   1   ?   Motor.driveMotors(0,   !AI_direction)   :
Motor.driveMotors(AI_throttle, AI_direction);

      // Set the Steering Servo with the Steering Angle
      STEERING_SERVO.write(AI_steering);
    }
  }
}

// Function for Receiving the I2C Data from the Jetson Nano
void receiveEvent(int byte_count) {
  if(byte_count >= 3) {
    throttle_byte = Wire.read();
    steering_byte = Wire.read();
    direction_bit = Wire.read();
  }
}

void RCFunctionality() {
  // Map RC PWM Data for Throttle and Steering
  RC.readReceiver();

  // Create Variable to Hold Status of Auto Braking
  static int result = 0;

  // Prevent Unneeded Auto Braking
  if(pwm_fwd > 0) {
    // Check for Auto Braking
    AutoBraking.brakeDetection(direction);

    // Save the Result of 'brakingResult()'
    result = AutoBraking.brakingResult();
  }

  // Control Indicators Based on Value of 'result'
  // Value 1: Activate Brake Indicator
  // Value 0: Activate Indicators Based on Regular Input
     result    ==    1    ?    Indicator.useIndicators(1,    pwm_side)    :
Indicator.useIndicators(direction, pwm_side);

  // Act Upon Returned Value of 'brakeDetection'
  // Value 1: Send PWM Value of 0 (Stopped) and Set Motor Direction to Reverse
(Activates Brake Lights)
```

```
  // Value 0: Send PWM Speed Data and Set Motor Direction
  result == 1 ? Motor.driveMotors(0, !direction) : Motor.driveMotors(pwm_fwd,
direction);

  // Set the Steering Servo with the Steering Angle
  STEERING_SERVO.write(pwm_side);
}

// Send Data to the Jetson Nano (Master) for Data Collection
void sendData() {
  // Initialize I2C Variables
  throttle_byte = pwm_fwd;
  steering_byte = pwm_side;
  direction_bit = direction;

  // Write Data
  Wire.write(throttle_byte);
  Wire.write(steering_byte);
  Wire.write(direction_bit);
}

// Receive Data from the Jetson Nano (Master) for Implementation
void receiveData(int amount) {
  // Loop Only While Data is Available
  while(Wire.available()) {
    // Read and Save the Incoming Data
    AI_direction = Wire.read();
    AI_steering = Wire.read();
    AI_throttle = Wire.read();

    //AIFunctionality();
    i2c_previous = millis();
  }
}
```

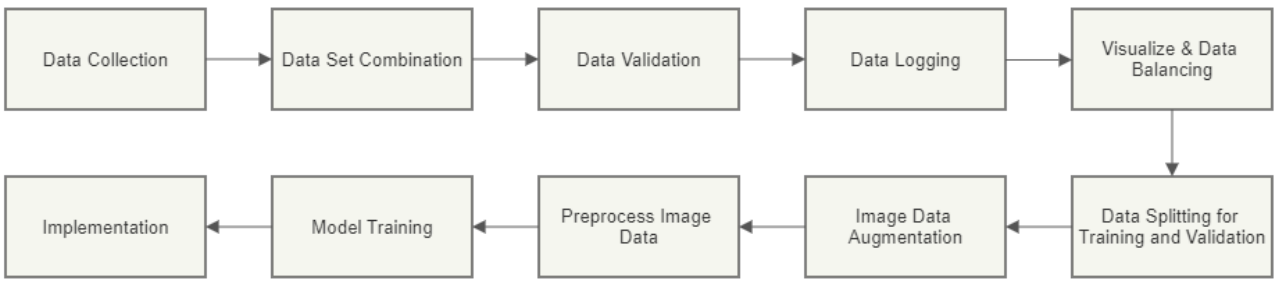


Figure 92. A.I. Training Diagram.

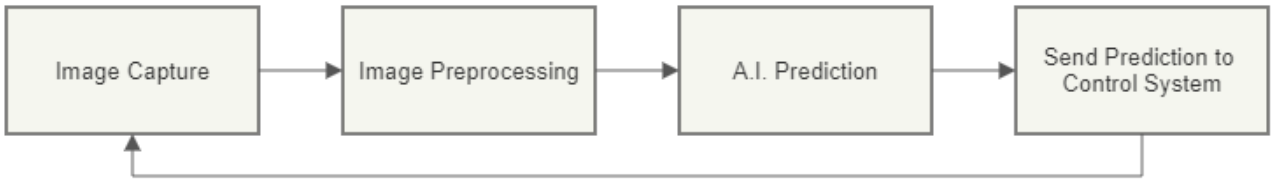


Figure 93. A.I. Real-Time Processing Diagram.

Data Collection

```
import cv2
import smbus
import time

# Function For Getting Camera Image
def get_camera_data(image_count):
    # Create GStreamer Camera Pipeline
    cap = cv2.VideoCapture(1)

    # Check if the Camera Opened Successfully
    if not cap.isOpened():
        print("Failed to Open the CSI Camera!")
        exit()

    # Read the Frame from the Camera
    ret, frame = cap.read()

    # Check if Frame was Read Successfully
    if not ret:
        print("Failed to Read a Frame from the Camera")
        cap.release()  # Release the camera capture
        exit()

    # Create File Name
    filename = "captured_images/image_{}.jpg".format(image_count)

    # Save the Frame as an Image
    cv2.imwrite(filename, frame)

    # Release the camera capture
    cap.release()


# Function for Reading Data from Arduino Mega 2560 REV 3 via I2C Protocol
def read_data_from_arduino(address, num_bytes):
    # Create a I2C Bus Object for the Arduino Mega 2560 REV 3
```

```
    # I2C '0' Bus is SDA on 27, SCL on 28
    # I2C '1' Bus is SDA on 3, SCL on 5
    bus = smbus.SMBus(1) # Changeable Bus

    # Read Data from I2C
    try:
        data = bus.read_i2c_block_data(address, 0, num_bytes)
    except Exception as e:
        print("Error Reading Data: ", e)
        data = [0] * num_bytes
    return data

def convert_bytes_to_values(data):
    # Convert Bytes to Data
    throttle_data = data[0]
    steering_data = data[1]
    direction_data = bool(data[2])

    # Make Throttle Negative if 'direction_data' Signals Reverse
    if direction_data == True:
        throttle_data = -throttle_data

    # Return Data Values
    return throttle_data, steering_data, direction_data

# Function for Saving I2C Data
def save_to_text_file(throttle, steering, data_count):
    with open(f"throttle_data/throttle_{data_count}.txt", 'a') as file:
        file.write(str(throttle))

    with open(f"steering_data/steering_{data_count}.txt", 'a') as file2:
        file2.write(str(steering))

def main():
    # Create Variable for Number of Data Points
    i = 0

    # Create Variable for I2C Address of Slave
    # Slave is Arduino Mega 2560 REV 3
    address = 0x04; # Changeable Address, Must Change on Arduino

    # Create Variable for Number of Bytes to Read from Slave
    num_bytes = 3

    # Create Variable for Number of Data Points a Second
    data_per_seconds = 100
```

```
    while True:
        # Get I2C Data
        data = read_data_from_arduino(address, num_bytes)

        # Convert Bytes to Decimal Values
        throttle, steering, direction = convert_bytes_to_values(data)

        # Print I2C Data
        print(f"Throttle: {throttle} Steering: {steering}")

        if throttle > 0:
            get_camera_data(i)
            save_to_text_file(throttle, steering, i)
            i += 1

        time.sleep(1/data_per_seconds)

if __name__ == "__main__":
    main()
```

Data Set Combination

```
import cv2
import os
import time

# Get Current Directory
current_dir = os.path.dirname(os.path.abspath(__file__))

# Construct Paths to the Data Sets
dataset_1 = os.path.join(current_dir, "CollectionOne", "Data Collection")
dataset_2 = os.path.join(current_dir, "CollectionTwo", "Data Collection")
dataset_3 = os.path.join(current_dir, "CollectionThree", "Data Collection")
dataset_4 = os.path.join(current_dir, "CollectionFour", "Data Collection")

# Construct Path to Output Directorys
image_output_dir     =     os.path.join(current_dir,     "Combined     Data",
"captured_images")
throttle_output_dir    =    os.path.join(current_dir,    "Combined    Data",
"throttle_data")
steering_output_dir    =    os.path.join(current_dir,    "Combined    Data",
"steering_data")

# Function for Clearing Terminal
def clear_terminal():
```

```
    # Clear Terminal Base on Operation Systems
    if os.name == "posix":
        os.system("clear")          # UNIX/Linux/Mac
    elif os.name == "nt":
        os.system("cls")            # Windows

# Function for Combining Throttle Data Sets into One Directory
def combine_throttle_data():
    # Construct Path to the Throttle Data Directory in Data Sets
    throttle_set_1 = os.path.join(dataset_1, "throttle_data")
    throttle_set_2 = os.path.join(dataset_2, "throttle_data")
    throttle_set_3 = os.path.join(dataset_3, "throttle_data")
    throttle_set_4 = os.path.join(dataset_4, "throttle_data")

    # Construct Path to Combined Throttle Data Output Directory
    output_dir = os.path.join(current_dir, "Combined Data", "throttle_data")

    # Clear Output Directory
    for f in os.listdir(output_dir):
        os.remove(os.path.join(output_dir, f))

    # Create a Variable to Hold the Number of the Last Throttle Data Saved
    last_data_saved = 0

    # Combine Data
    last_data_saved += add_data(throttle_set_1, "throttle", last_data_saved)
    last_data_saved += add_data(throttle_set_2, "throttle", last_data_saved)
    last_data_saved += add_data(throttle_set_3, "throttle", last_data_saved)
    last_data_saved += add_data(throttle_set_4, "throttle", last_data_saved)

    # Return Number of Combined Data Points
    return last_data_saved

# Function for Combining Throttle Data Sets into One Directory
def combine_steering_data():
    # Construct Path to the Throttle Data Directory in Data Sets
    steering_set_1 = os.path.join(dataset_1, "steering_data")
    steering_set_2 = os.path.join(dataset_2, "steering_data")
    steering_set_3 = os.path.join(dataset_3, "steering_data")
    steering_set_4 = os.path.join(dataset_4, "steering_data")

    # Construct Path to Combined Throttle Data Output Directory
    output_dir = os.path.join(current_dir, "Combined Data", "steering_data")

    # Clear Output Directory
    for f in os.listdir(output_dir):
```

```
        os.remove(os.path.join(output_dir, f))

    # Create a Variable to Hold the Number of the Last Throttle Data Saved
    last_data_saved = 0

    # Combine Data
    last_data_saved += add_data(steering_set_1, "steering", last_data_saved)
    last_data_saved += add_data(steering_set_2, "steering", last_data_saved)
    last_data_saved += add_data(steering_set_3, "steering", last_data_saved)
    last_data_saved += add_data(steering_set_4, "steering", last_data_saved)

    # Return Number of Combined Data Points
    return last_data_saved

# Function for Adding Data to the Output
def add_data(dataset, data_type, last_number_saved):
    # Get the Number of Data Points
    dataset_length = len(os.listdir(dataset))

    # Create Variable for Desired Data Number
    data_number = 0

    # Create Variable for Saved Data Points
    data_saved = 0

    # Add Data to Output Directory
    while data_saved != dataset_length:
        # Construct Path to Desired Data File
        if data_type.lower() == "throttle":
            data = os.path.join(dataset, f"throttle_{data_number}.txt")
        elif data_type.lower() == "steering":
            data = os.path.join(dataset, f"steering_{data_number}.txt")
        else:
            print("Error: Unknown Data Type!")
            exit()

        # Obtain Data
        try:
            with open(data, 'r') as data_file:
                # Check for Available Data
                if data_file is not None:
                    # Get Data
                    raw_data = data_file.readline()

                    # Construct Path to Desired Data Output
                    if data_type.lower() == "throttle":
```

```
                            data_output = os.path.join(throttle_output_dir,
f"throttle_{data_saved + last_number_saved}.txt")
                    elif data_type.lower() == "steering":
                            data_output = os.path.join(steering_output_dir,
f"steering_{data_saved + last_number_saved}.txt")
                    else:
                        print("Error: Unknown Data Type!")
                        exit()

                    # Save Data to Desired Output Directory
                    with open(data_output, "w") as data:
                        data.write(raw_data)

                    # Increase Number of Saved Data Points
                    data_saved += 1

                    # Increase Desired Data Point Number
                    data_number += 1
        except FileNotFoundError:
            # Increase Desired Data Point Number
            data_number += 1

    # Return Number of Saved Data Points
    return data_saved

# Function for Combining Image Data Sets into One Directory
def combine_images():
    # Construct Path to the Image Directory in Data Sets
    image_set_1 = os.path.join(dataset_1, "captured_images")
    image_set_2 = os.path.join(dataset_2, "captured_images")
    image_set_3 = os.path.join(dataset_3, "captured_images")
    image_set_4 = os.path.join(dataset_4, "captured_images")

    # Construct Path to Combined Image Output Directory
    output_dir = os.path.join(current_dir, "Combined Data", "captured_images")

    # Clear Output Directory
    for f in os.listdir(output_dir):
        os.remove(os.path.join(output_dir, f))

    # Create a Variable to Hold the Number of the Last Image Saved
    last_number_saved = 0

    # Add Data Set Images to Output Directory
        last_number_saved   +=   add_images(image_set_1,   ".jpg",   ".jpg",
last_number_saved)
```

```
        last_number_saved   +=   add_images(image_set_2,   ".png",   ".jpg",
last_number_saved)
        last_number_saved   +=   add_images(image_set_3,   ".png",   ".jpg",
last_number_saved)
        last_number_saved   +=   add_images(image_set_4,   ".png",   ".jpg",
last_number_saved)

    # Return Number of Saved Images
    return last_number_saved

# Function for Adding Images to the Output Directory
def add_images(dataset, image_format, output_format, last_image_number):
    # Get the Number of Images in the Data Set
    dataset_length = len(os.listdir(dataset))

    # Create Variable for Desired Image
    image_number = 0

    # Create Variable for Number of Saved Images
    images_saved = 0

    # Add Images to Output Directory
    while images_saved != dataset_length:
        # Construct Path to Desired Image
        img = os.path.join(dataset, f"image_{image_number}{image_format}")

        # Grab the Image
        img = cv2.imread(img)

        # Act on Image Availability
        if img is None:
            # Clear Terminal of Warnings
            clear_terminal()
            print("Combining...")
        else:
            # Construct Path to Save Image
            save_path = os.path.join(image_output_dir, f"image_{images_saved +
last_image_number}{output_format}")

            # Save if the Image is Available
            cv2.imwrite(save_path, img)

            # Increase the Count for Saved Images
            images_saved += 1

        # Move to Next Image
```

```
        image_number += 1

    # Return Number of Combined Images
    return images_saved

# Main Program Function
if  __name__ == "__main__":
    # Clear Terminal
    clear_terminal()

    # Ask for Input on Desired Action
    user_input = input("\nTo Combine Throttle Data Press 'N', to Skip Press
'S', to Exit Press 'Q': ").upper()

    # Act Accordingly
    if user_input == 'N':
        print("\nStarting Throttle Combination\n")

        # Get Start Time
        start_time = time.time()

        # Combine Throttle Data
        completed = combine_throttle_data()

        # Clear Terminal
        clear_terminal()

        # Output Results
        print("---Throttle-Combination-Completed---")
          print(f"Combined {completed} Data Points in {round((time.time() -
start_time), 2)} Seconds!\n")
    elif user_input == 'S':
        print("\nSkipping...")
    elif user_input == 'Q':
        exit()

    # Ask for Input on Desired Action
    user_input = input("\nTo Combine Steering Data Press 'N', to Skip Press
'S', to Exit Press 'Q': ").upper()

    # Act Accordingly
    if user_input == 'N':
        print("\nStarting Steering Combination\n")

        # Get Start Time
        start_time = time.time()
```

```
        # Combine Throttle Data
        completed = combine_steering_data()

        # Clear Terminal
        clear_terminal()

        # Output Results
        print("---Steering-Combination-Completed---")
          print(f"Combined {completed} Data Points in {round((time.time() -
start_time), 2)} Seconds!\n")
    elif user_input == 'S':
        print("\nSkipping...")
    elif user_input == 'Q':
        exit()

    # Ask for Input on Desired Action
    user_input = input("\nTo Combine Image Data Press 'N', to Exit Press 'Q':
").upper()

    # Act Accordingly
    if user_input == 'N':
        print("\nStarting Steering Combination\n")

        # Get Starting Time
        start_time = time.time()

        # Combine Images
        completed = combine_images()

        # Clear Terminal
        clear_terminal()

        # Output Results
        print("---Image-Combination-Completed---")
            print(f"Combined {completed} Images in {round((time.time() -
start_time), 2)} Seconds!\n")
    elif user_input == 'Q':
        exit()
```

Data Preprocessing

```
import os
import time
import random
import numpy as np
```

```
from skimage import io
from skimage import exposure

# Get Current Directory Path
current_dir = os.path.dirname(os.path.abspath(__file__))

# Function for Clearing Terminal
def clear_terminal():
    # Clear Terminal Base on Operation Systems
    if os.name == "posix":
        os.system("clear")          # UNIX/Linux/Mac
    elif os.name == "nt":
        os.system("cls")            # Windows

# Function for Preprocessing Data
def data_preprocessing(data_type, remix_count):
    # Construct the Path to the "throttle_data" Dataset Directory
       data_dir   =   os.path.join(current_dir,   "..",   "Data   Collection",
f"{data_type}_data")

    # Construct the Path to the Output Directory
             output_dir     =     os.path.join(current_dir,     "Processed",
f"{data_type}_data_output")

    # Clear Output Directory
    for f in os.listdir(output_dir):
        os.remove(os.path.join(output_dir, f))

    # Create Variable for Holding Number of Preprocessing Errors
    data_errors = 0

    # Get Dataset
    dataset = os.listdir(data_dir)

    # Create Variable for Desired Data Point Number
    data_number = 0

    # Create Variable for Total Saved Files
    data_saved = 0

    # Process Data
    while data_number != len(dataset):
        # Create File Path to Files in Order
        file_path = os.path.join(data_dir, f"{data_type}_{data_number}.txt")

        # Open File
```

```
        with open(file_path, 'r') as data:
            # Get Raw Data
            raw_data = data.readline()

        # Set Acceptable Data Range Base Upon 'data_type'
        if data_type == "throttle":
            min = 0
            max = 127
        elif data_type == "steering":
            min = 75
            max = 140

        # Check if Data is Inside Allowable Range
        if int(raw_data) > max or int(raw_data) < min:
            # Get Length of 'raw_data'
            data_length = len(raw_data)

            if data_length == 4:
                if raw_data[0:2] == raw_data[2:4]:
                    processed_data = raw_data[0:2]
            elif data_length == 5:
                processed_data == raw_data[0:2]
            elif data_length == 6:
                if raw_data[0:3] == raw_data[3:6]:
                    processed_data == raw_data[0:3]
            else:
                # Save a List of Unsolved Range Errors
                unsolved_error_path = os.path.join(output_dir, "..", f"Unsolved
Data Errors ({data_type}).txt")
                with open(unsolved_error_path, 'a') as error_file:
                  error_file.write(str(file_path) + ":       " + str(raw_data)
+ '\n')

                # Increase Unsolved Errors by One
                data_errors += 1
        else:
            # Skip Data Correction Save
            processed_data = raw_data

        # Map Data from Allowed Ranges to a Range of 0-1
        mapped_data = (int(processed_data) - min) * (1 - 0) / (max - min)

        for i in range(int(remix_count)):
            # Construct Data Saving Path
                                filename     =     os.path.join(output_dir,
f"processed_steering_{data_saved}.txt")
```

```
            # Save Data to Output Directories
            with open(filename, "w") as processed_file:
                processed_file.write(str(mapped_data))

            # Increase Save Count
            data_saved += 1

        # Increase Desired Data Number
        data_number += 1


    return data_saved, data_errors

# Function for Preprocessing Image Data
def image_preprocessing(remix_count):
    # Construct the Path to the "captured_images" Directory
       dataset_dir  =  os.path.join(current_dir,  "..",  "Data  Collection",
"captured_images")

    # Construct the Path to the Output Directory
    output_dir = os.path.join(current_dir, "Processed", "images_output")

    # Clear Output Directory
    for f in os.listdir(output_dir):
        os.remove(os.path.join(output_dir, f))

    # Create Variable for Holding Number of Preprocessing Errors
    preprocessing_errors = 0

    # Get Dataset
    dataset = os.listdir(dataset_dir)

    # Get Data Set Length
    dataset_len = len(dataset)

    # Create Variable for Desired Image Number
    image_number = 0
    # Create Variable for Total Images Saved
    saved_images = 0

    # Perform Data Augmentation
    for image_name in dataset:
        # Construct Original Image Path
        image_path = os.path.join(dataset_dir, f"image_{image_number}.jpg")
```

```
        # Load Image
        image = io.imread(image_path)

        # Perform Image Brightness Augmentation
        for i in range(int(remix_count)):
            # Get Random Brightness Value
            brightness = random.uniform(0.5, 2)

            # Apply Brightness Change
               bright_image = exposure.adjust_gamma(image, gamma=brightness,
gain=1)

            # Normalize the Image
                    norm_image = (bright_image - np.min(bright_image)) /
(np.max(bright_image) - np.min(bright_image))

            # Construct Image Saving Path
                                filename = os.path.join(output_dir,
f"processed_image_{saved_images}.npy")

            # Save the Preproceseed Image
            np.save(filename, norm_image, allow_pickle=True)

            # Load the Saved Data for Validation
            data_validation = np.load(filename)

            # Compare the Saved Data to the Normalized Image
            valid = np.array_equal(norm_image, data_validation)

            # Check for Invalid Data
            if valid == False:
                # Save Error
                    image_error_path = os.path.join(output_dir, "..", "Image
Preprocessing Errors.txt")
                with open(image_error_path, 'a') as error_file:
                    error_file.write(str(image_name) + '\n')
                    preprocessing_errors += 1

            # Increase Number of Saved Images
            saved_images += 1

        # Status Output
        clear_terminal()
         print(f"({round((saved_images/(dataset_len*remix_count)) * 100, 2)}%)
Image Preprocessing...")
```

```
        # Increase Next Desired Image
        image_number += 1

    return preprocessing_errors

if __name__ == "__main__":
    # Clear Terminal
    clear_terminal()

    # Ask for 'remix_count'
     remixes = input("\nTo Duplicate Data Please Enter a Number Greater then
1,\nTo Keep the Original Amount of Data Enter 1: ")

    # Check that Input is Valid
    if remixes.isnumeric is False:
        print("ERROR: Invalid Input!")
        exit()

    # Ask for Input on Desired Action
    user_input = input("\nTo Validate Data Press 'N', to Skip Data Validation
Press 'S', to Exit Press 'Q': ").upper()

    # Act Accordingly
    if user_input == 'N':
        # Clear Terminal
        clear_terminal()
        print("(0%) Validating Data...")

        # Get Starting Time
        start_time = time.time()

        # Preprocesss Steering Data
        # Returns: Number of Unsolved Errors
         completed_steering, steering_errors = data_preprocessing("steering",
remixes)

        # Clear Terminal
        clear_terminal()
        print("(50%) Validating Data...")

        # Preprocess Throttle Data
        # Returns: Number of Unsolved Errors
         completed_throttle, throttle_errors = data_preprocessing("throttle",
remixes)

        # Calculate Runtime
```

```
        completion_time = round(time.time() - start_time, 2)

        # Clear Terminal
        clear_terminal()

        # Output Results
        print("---Data-Validation-Completed---")
         print(f"Preprocessed {completed_throttle + completed_steering} Files
in {completion_time} Seconds!")
        print(f"Unsolved Steering Data Errors: {steering_errors}")
        print(f"Unsolved Throttle Data Errors: {throttle_errors}")
    elif user_input == 'S':
        print("\nSkipping to Image Preprocessing!")
    elif user_input == 'Q':
        print("Exiting Program...")
        exit()
    else:
        print("Error: Invalid Input, Exiting Program!")
        exit()

    # Ask for Input on Next Action
    user_input = input("\nTo Continue Preprocessing Press 'N', to Exit Press
'Q': ").upper()

    # Act Accordingly
    if user_input == 'N':
        # Clear Terminal
        clear_terminal()
        print("(0%) Image Preprocessing...")

        # Get Start Time
        start_time = time.time()

        # Normalize Captured Images
        preprocessing_errors = image_preprocessing(remixes)

        # Calculate Runtime
        completion_time = round(time.time() - start_time, 2)

        # Clear Terminal
        clear_terminal()

        # Output Results
        print("\n---Image-Preprocess-Completed---")
        print(f"Preprocessed in {completion_time} Seconds!")
        print(f"Preprocessing Errors: {preprocessing_errors}")
```

```
    elif user_input == 'Q':
        print("Exiting Program...")
        exit()
    else:
        print("Error: Invalid Input, Exiting Program!")
        exit()
```

Data Logging

```
import os
import time
import csv

# Function for Making Log File
def make_log(min_steering, centered_steering, max_steering, max_throttle):
    # Get the Current Directory
    current_dir = os.path.dirname(os.path.abspath(__file__ ))

    # Construct Path to Image Directory
    image_dir = os.path.join(current_dir, "Combined Data", "captured_images")

    # Construct Path to Steering Directory
    steering_dir = os.path.join(current_dir, "Combined Data", "steering_data")

    # Construct Path to Throttle Directory
    throttle_dir = os.path.join(current_dir, "Combined Data", "throttle_data")

    # Create a Variable for Number of Images in 'image_dir'
    number_images = len(os.listdir(image_dir))

    # Create a Variable for Number of Steering Data in 'steering_dir"
    number_steering = len(os.listdir(steering_dir))

    number_throttle = len(os.listdir(throttle_dir))

    # Ensure All Data is Present
    if number_images != number_steering or number_images != number_throttle or
number_steering != number_throttle:
        print("ERROR: Number of Data Does Not Match!")
        print(f"       Images: {number_images}")
        print(f"       Steering: {number_steering}")
        print(f"       Throttle: {number_throttle}")
        exit()

    # Create Variable for Desired Data Number
    data_wanted = 0
```

```
    # Loop for Going Through Data Set
    while data_wanted != number_images:
        # Construct Image Path
        image_path = os.path.join(image_dir, f"image_{data_wanted}.jpg")

        # Construct Steering Data Path
                            steering_path     =     os.path.join(steering_dir,
f"steering_{data_wanted}.txt")

        # Construct Throttle Data Path
                            throttle_path     =     os.path.join(throttle_dir,
f"throttle_{data_wanted}.txt")

        # Get Corresponding Steering Data
        with open(steering_path, 'r') as file:
            data_steering = file.readline()

        # Get Corresponding Throttle Data
        with open(throttle_path, 'r') as file:
            data_throttle = file.readline()

        # Map Steering Data
        if int(data_steering) < centered_steering:
             data_steering = (int(data_steering) - centered_steering) * (-1 -
0) / (min_steering  - centered_steering)     # (-1, 0)
        elif int(data_steering) == centered_steering:
            data_steering = 0
        elif int(data_steering) > centered_steering:
            data_steering = (int(data_steering) - centered_steering) * (1 - 0)
/ (max_steering - centered_steering)      # (0, 1)

        # Map Throttle Data
        data_throttle = int(data_throttle) / max_throttle       # (0, 1)

        # Create Path to Save Log At
        output_path = os.path.join(current_dir, "Combined Data", "log_1.csv")

        # Check for Previous Log
        if data_wanted == 0:
            if os.path.exists(output_path):
                # Delete Outdated Log
                os.remove(output_path)

                # Create Blank Log
                with open(output_path, 'w') as blank:
```

```
                    blank = csv.writer(blank)

        # Save the Data to the Log File
        with open(output_path, 'a', newline='') as csvfile:
            writer = csv.writer(csvfile)
            writer.writerow([image_path, data_steering, data_throttle])

        # Increase to Next Data Point
        data_wanted += 1

# Main Loop
if __name__ == "__main__":
    # Get Start Time
    start_time = time.time()

    # Produce the Log
    make_log(75, 90, 140, 127)

    # Print Completion Time
    print(f"Completed the Log in {round(time.time() - start_time, 2)} Seconds!")
```

Model Training (Training)

```
print('Setting UP')
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
from sklearn.model_selection import train_test_split
from utlis import *

#### STEP 1 - INITIALIZE DATA
path = 'DataCollected'
data = importDataInfoV2(path)
print(data.head())

#### STEP 2 - VISUALIZE AND BALANCE DATA
data = balanceDataV2(data,display=True)

#### STEP 3 - PREPARE FOR PROCESSING
imagesPath, steerings, throttles = loadDataV2(path,data)
#       print('No       of       Path       Created       for       Images
',len(imagesPath),len(steerings),len(throttles))
# img = cv2.imread(imagesPath[5])
# cv2.namedWindow('Test Image', cv2.WINDOW_NORMAL)
# cv2.resizeWindow('Test Image', (960, 540))
# cv2.moveWindow('Test Image', 0, 0)
# cv2.imshow('Test Image', img)
```

```
# cv2.waitKey(0)
# exit()

#### STEP 4 - SPLIT FOR TRAINING AND VALIDATION
xTrain, xVal, yTrain, yVal, zTrain, zVal = train_test_split(imagesPath,
steerings, throttles,
                                                test_size=0.2,random_state=10)
print('Total Training Images: ',len(xTrain))
print('Total Validation Images: ',len(xVal))

trainValGraph(yTrain, yVal, "Steering")
trainValGraph(zTrain, zVal, "Throttle")

#### STEP 5 - AUGMENT DATA

#### STEP 6 - PREPROCESS

#### STEP 7 - CREATE MODEL
model = createModel()

#### STEP 8 - TRAINNING
history = model.fit(
    dataGenV2(xTrain, yTrain, zTrain, 15, 1),
    steps_per_epoch=15,
    epochs=6,
    validation_data=dataGenV2(xVal, yVal, zVal, 15, 0),
    validation_steps=15
)

#### STEP 9 - SAVE THE MODEL
model.save('model_V5.h5')
print('Model Saved')

#### STEP 10 - PLOT THE RESULTS
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['Training', 'Validation'])
plt.title('Loss')
plt.xlabel('Epoch')
plt.show()
```

Model Training (Utlis)

```
import os
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
import cv2
import random
import matplotlib.image as mpimg
from sklearn.utils import shuffle

from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Convolution2D,Flatten,Dense, Input,
Concatenate
from tensorflow.keras.optimizers import Adam
from imgaug import augmenters as iaa

#### STEP 1 - INITIALIZE DATA
def getName(filePath):
    myImagePathL = filePath.split('/')[-2:]
    myImagePath = os.path.join(myImagePathL[0],myImagePathL[0])
    print(myImagePath)
    exit()
    return myImagePath

def importDataInfoV2(path):
    columns = ['Center','Steering', 'Throttle']
    #noOfFolders = len(os.listdir(path))//2
    data = pd.DataFrame()

    try:
        x = 1  # Number of logs, can be replaced for a 'for x in range(<min log
number>, <max log number>)'
        current_dir = os.path.dirname(os.path.abspath(__file__))
         dataNew = pd.read_csv(os.path.join(current_dir, path, "log_2.csv"),
names = columns)
        print(f'{x}:{dataNew.shape[0]} ',end='')
        #### REMOVE FILE PATH AND GET ONLY FILE NAME
        #print(getName(data['center'][0]))
        dataNew['Center']=dataNew['Center'].apply(getName)
        data =pd.concat(objs=[data, dataNew], ignore_index=True)
    except FileNotFoundError:
        print("ERROR: No Log File Found")
        exit()

    print(' ')
    print('Total Images Imported',data.shape[0])
    return data

#### STEP 2 - VISUALIZE AND BALANCE DATA
def balanceDataV2(data,display=True):
```

```
    nBin = 31
    samplesPerBin =  71

    # Create Histograms for Data
    hist, bins = np.histogram(data['Steering'], nBin)      # Steering Histogram
    hist2, bins2 = np.histogram(data['Throttle'], nBin)    # Throttle Histogram

    # Display if 'True'
    if display:
        # Steering Histogram
        center = (bins[:-1] + bins[1:]) * 0.5
        plt.bar(center, hist, width=0.03)
              plt.plot((np.min(data['Steering']),  np.max(data['Steering'])),
(samplesPerBin, samplesPerBin))
        plt.title('Data Visualisation')
        plt.xlabel('Steering Angle')
        plt.ylabel('No of Samples')
        plt.show()

        # Throttle Histogram
        center2 = (bins2[:-1] + bins[1:]) * 0.5
        plt.bar(center2, hist2, width=0.03)
              plt.plot((np.min(data['Throttle']),  np.max(data['Throttle'])),
(samplesPerBin, samplesPerBin))
        plt.title('Data Visualisation')
        plt.xlabel('Throttle Value')
        plt.ylabel('No of Samples')
        plt.show()

    # Create Index Array
    removeindexList = []

    for j in range(nBin):
        binDataList = []
        for i in range(len(data['Steering'])):
            if data['Steering'][i] >= bins[j] and data['Steering'][i] <= bins[j
+ 1]:
                binDataList.append(i)
        binDataList = shuffle(binDataList)
        binDataList = binDataList[samplesPerBin:]
        removeindexList.extend(binDataList)

    print('Removed Images:', len(removeindexList))
    data.drop(data.index[removeindexList], inplace=True)
    print('Remaining Images:', len(data))
```

```
    if display:
        #  Balanced Steering
        hist, _ = np.histogram(data['Steering'], (nBin))
        plt.bar(center, hist, width=0.03)
              plt.plot((np.min(data['Steering']),  np.max(data['Steering'])),
(samplesPerBin, samplesPerBin))
        plt.title('Balanced Data')
        plt.xlabel('Steering Angle')
        plt.ylabel('No of Samples')
        plt.show()

        # Balanced Throttle
        hist2, _ = np.histogram(data['Throttle'], (nBin))
        plt.bar(center2, hist2, width=0.03)
              plt.plot((np.min(data['Throttle']),  np.max(data['Throttle'])),
(samplesPerBin, samplesPerBin))
        plt.title('Balanced Data')
        plt.xlabel('Throttle Value')
        plt.ylabel('No of Samples')
        plt.show()

    return data

#### STEP 3 - PREPARE FOR PROCESSING
def loadDataV2(path, data):
  imagesPath = []
  steering = []
  throttle = []

  for i in range(len(data)):
    indexed_data = data.iloc[i]
    imagesPath.append( os.path.join(path,indexed_data[0]))
    steering.append(float(indexed_data[1]))
    throttle.append(float(indexed_data[2]))

  imagesPath = np.asarray(imagesPath)
  steering = np.asarray(steering)
  throttle = np.asarray(throttle)

  return imagesPath, steering, throttle

#### STEP 4 - Training and Validation Graphs
def trainValGraph(train, val, type):
    nBin = 31

    hist, bins = np.histogram(train, (nBin))
```

```
    center = (bins[:-1] + bins[1:]) * 0.5
    plt.bar(center, hist, width = 0.03)
    plt.plot(np.min(train), np.max(train))
    plt.title(f"{type} Training Set")

    if type == "Steering":
        plt.xlabel("Steering Angle")
    elif type == "Throttle":
        plt.xlabel("Throttle Value")
    else:
        print(f"ERROR: Unknown Type: {type}")

    plt.ylabel("No of Samples")
    plt.show()

    hist, bins = np.histogram(val, (nBin))
    center = (bins[:-1] + bins[1:]) * 0.5
    plt.bar(center, hist, width = 0.03)
    plt.plot(np.min(val), np.max(val))
    plt.title(f"{type} Validation Set")

    if type == "Steering":
        plt.xlabel("Steering Angle")
    elif type == "Throttle":
        plt.xlabel("Throttle Value")
    else:
        print(f"ERROR: Unknown Type: {type}")

    plt.ylabel("No of Samples")
    plt.show()

#### STEP 5 - AUGMENT DATA
def augmentImageV2(imgPath,steering,throttle):
    img =  mpimg.imread(imgPath)

    if np.random.rand() < 0.5:
       pan = iaa.Affine(translate_percent={"x": (-0.1, 0.1), "y": (-0.1, 0.1)})
        img = pan.augment_image(img)
    if np.random.rand() < 0.5:
        zoom = iaa.Affine(scale=(1, 1.2))
        img = zoom.augment_image(img)
    if np.random.rand() < 0.5:
        brightness = iaa.Multiply((0.5, 1.2))
        img = brightness.augment_image(img)
    if np.random.rand() < 0.5:
        img = cv2.flip(img, 1)
```

```
        steering = -steering

    return img, steering, throttle

## Uncomment For Testing
# for x in range(0, 10):
#     imgRe,st = augmentImage(f'DataCollected/IMG1/image_{x}.jpg',0)
#     mpimg.imsave('Result.jpg',imgRe)
#     plt.imshow(imgRe)
#     plt.show()
# exit()

#### STEP 6 - PREPROCESS
def preProcess(img):
    img = img[875:1150,:,:]
    img = cv2.resize(img, (200, 66))
    img = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
    img = cv2.GaussianBlur(img,  (3, 3), 0)
    img = cv2.resize(img, (200, 66))
    img = img/255

    return img

## Uncomment For Testing
# for x in range(0, 10):
#     imgRe = preProcess(mpimg.imread(f'DataCollected/IMG1/image_{x}.jpg'))
#     mpimg.imsave('Result.jpg',imgRe)
#     plt.imshow(imgRe)
#     plt.show()
# exit()

#### STEP 7 - CREATE MODEL
def createModel():
    image_input = Input(shape=(66, 200, 3))
    x = Convolution2D(6, (5, 5), (2, 2), activation='elu')(image_input)
    x = Convolution2D(8, (5, 5), (2, 2), activation='elu')(x)
    x = Convolution2D(12, (5, 5), (2, 2), activation='elu')(x)
    x = Convolution2D(16, (3, 3), activation='elu')(x)
    x = Convolution2D(16, (3, 3), activation='elu')(x)

    x = Flatten()(x)
    x = Dense(25, activation='elu')(x)
    x = Dense(12, activation='elu')(x)
    x = Dense(5, activation='elu')(x)

    # Define Output Nodes to the Model
```

```
    steering_out = Dense(1, name='steering')(x)
    throttle_out = Dense(1, name='throttle')(x)

    model = Model(inputs=image_input, outputs=[steering_out, throttle_out])

    model.compile(Adam(lr=0.0001), loss='mse', loss_weights=[1.0, 1.0])

    return model

#### STEP 8 - TRAINING
def dataGenV2(imagesPath, steeringList, throttleList, batchSize, trainFlag):
    while True:
        imgBatch = []
        steeringBatch = []
        throttleBatch = []

        for i in range(batchSize):
            index = random.randint(0, len(imagesPath) - 1)
            if trainFlag:
                  img, steering, throttle = augmentImageV2(imagesPath[index],
steeringList[index], throttleList[index])
            else:
                img = mpimg.imread(imagesPath[index])
                steering = steeringList[index]
                throttle = throttleList[index]
            img = preProcess(img)
            imgBatch.append(img)
            steeringBatch.append(steering)
            throttleBatch.append(throttle)

        # Yield two separate arrays for 'steering' and 'throttle'
                  yield   np.asarray(imgBatch),   [np.asarray(steeringBatch),
np.asarray(throttleBatch)]
```

Model Tester

```
import cv2
import numpy as np
from tensorflow.keras.models import load_model
import os
import pandas as pd
import matplotlib.pyplot as plt

# Obtain the Current Directory
current_dir = os.path.dirname(os.path.abspath(__file__))
```

```
# Construct Path to the Model
model_path = os.path.join(current_dir, "Models", "model_8-6-2023-2-35-pm.h5")

# Load the Model
model = load_model(model_path)

# Construct Path to Data Set Log File
data_set_log = os.path.join(current_dir, "DataCollected", "log_2.csv")
print(data_set_log)

# Function for Getting Image Name
def getName(filePath):
    myImagePath = filePath.split('/')[-2:]
    myImagePath = os.path.join(myImagePath[0], myImagePath[0])

    return myImagePath

# Function for Importing Data Info Saved in Log Files
def importDataInfo():
    columns = ['Image', 'Steering', 'Throttle']
    data = pd.DataFrame()

    try:
        dataNew = pd.read_csv(data_set_log, names = columns)

        dataNew['Image'] = dataNew['Image'].apply(getName)
        data = pd.concat(objs= [data, dataNew], ignore_index= True)

    except FileNotFoundError:
        print("ERROR: No Log File Found!")
        exit()

    print('\nTotal Images Imported', data.shape[0])

    return data

# Function for Loading Data from the Log File
def loadData(path, data):
    imagesPath = []
    steering = []
    throttle = []

    for i in range(len(data)):
        indexed_data = data.iloc[i]
        imagesPath.append(os.path.join(path, indexed_data[0]))
        steering.append(float(indexed_data[1]))
```

```
        throttle.append(float(indexed_data[2]))

    imagesPath = np.asarray(imagesPath)
    steering = np.asarray(steering)
    throttle = np.asarray(throttle)

    return imagesPath, steering, throttle

# Function for Displaying Data
def cycleData(data, images, steerings, throttles):
    # Create Array to Later Find Steering Prediction Average
    prediction_steering = []
    prediction_throttle = []

    # Create Arrays to Later Hold the Real-World Data
    real_steering = []
    real_throttle = []

     # Create Arrays to Later Hold the Difference Between Real World and
Prediction Data
    results_steering = []
    results_throttle = []

    for i in range(len(data)):
        # Open Image
        img = cv2.imread(images[i])

        # Preprocess Image
        img = img[875:1150, :, :]
        img = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
        img = cv2.GaussianBlur(img, (3, 3), 0)
        img = cv2.resize(img, (200, 66))
        img = img / 255

        # Get A.I. Prediction
        input_data = np.expand_dims(img, axis=0)
        prediction = model.predict(input_data)

        # Correct Out of Range Data
        if float(prediction[0]) < -1:
            prediction[0] = -1
        elif float(prediction[0]) > 1:
            prediction[0] = 1

        # Convert Steering Prediction to Usable Data
        if float(prediction[0]) <= 0:
```

```
            prediction[0] = float(prediction[0]) * (90 - 75) + 90
        elif float(prediction[0]) > 0:
            prediction[0] = float(prediction[0]) * (140 - 91) + 90

        # Convert Steering Real World to Usable Data
        if float(steerings[i]) <= 0:
            steerings[i] = float(steerings[i]) * (90 - 75) + 90
        elif float(prediction[0]) > 0:
            steerings[i] = float(steerings[i]) * (140 - 91) + 90

        # Round Steering Data
        steerings[i] = round(steerings[i])

        # Add to Steering Prediction Array
        prediction_steering.append(float(prediction[0]))

        # Convert Throttle Prediction to Usable Data
        prediction[1] = float(prediction[1]) * 127

        # Conver Throttle Real World to Usable Data
        throttles[i] = float(throttles[i]) * 127

        # Round Throttle Data
        throttles[i] = round(throttles[i])

        # Correct Out of Range Data
        if float(prediction[1]) < 0:
            prediction[1] = 0
        elif float(prediction[1] > 127):
            prediction[1] = 127

        # Add to Throttle Prediction Array
        prediction_throttle.append(float(prediction[1]))

        # Display Text
        steering_val = str(round(float(prediction[0])))
        throttle_val = str(round(float(prediction[1])))
        cord = (50, 60)
        font = cv2.FONT_HERSHEY_COMPLEX
        fontScale = 0.5
        color = (255, 0, 255)
        thickness = 1
          image = cv2.putText(img, f"{steering_val}    {throttle_val}", cord,
font, fontScale, color, thickness, cv2.LINE_AA)

        # Display Image with Text
```

```
        cv2.namedWindow(f'Image {i}', cv2.WINDOW_NORMAL)
        cv2.resizeWindow(f'Image {i}', (860, 540))
        cv2.moveWindow(f'Image {i}', 100, 100)
        cv2.imshow(f'Image {i}', image)
        cv2.waitKey(1)

        # Destroy Window from Two Images Ago
        # Helps Keep Operation Speeds High
        if i > 2:
            cv2.destroyWindow(f'Image {i-2}')

        # Append Real World Data to Appropriate Arrays
        real_steering.append(int(steerings[i]))
        real_throttle.append(int(throttles[i]))

        # Append Differences to Appropriate Arrays
        results_steering.append(int(steerings[i]) - int(steering_val))
        results_throttle.append(int(throttles[i]) - int(throttle_val))

    # Get the Average of the 'predictions' Arrays and 'results' Arrays
    average_steering = sum(prediction_steering) / len(prediction_steering)
    average_throttle = sum(prediction_throttle) / len(prediction_throttle)
    average_steering_diff = sum(results_steering) / len(results_steering)
    average_throttle_diff = sum(results_throttle) / len(results_throttle)

    # Print the Averages
    print("Average Steering: ", average_steering)
    print("Average Throttle: ", average_throttle)
    print("Average Steering Diff: ", average_steering_diff)
    print("Average Throttle Diff: ", average_throttle_diff)

    print(min(results_steering), max(results_steering))
    print(min(results_throttle), max(results_throttle))

    # Destroy All OpenCV Windows
    cv2.destroyAllWindows()

    # Generate Graphs
      graphResults(results_steering,  results_throttle,  prediction_steering,
prediction_throttle, real_steering, real_throttle)

# Function for Generating Result Graphs
def     graphResults(steering_diff,     throttle_diff,     steering_predict,
throttle_predict, steering_real, throttle_real):
    # Generate Steering Histogram Graph
    hist, bins = np.histogram(steering_diff, bins=31)
```

```
    center = (bins[:-1] + bins[1:]) * 0.5
    plt.bar(center, hist, width = 1.25)
    plt.plot(np.min(steering_diff), np.max(steering_diff))
    plt.title('Steering Data Differences')
    plt.xlabel('Difference')
    plt.ylabel('Frequency')
    plt.show()

    # Generate Throttle Histogram Graph
    hist, bins = np.histogram(throttle_diff, bins=31)
    center = (bins[:-1] + bins[1:]) * 0.5
    plt.bar(center, hist, width = 1.25)
    plt.plot(np.min(throttle_diff), np.max(throttle_diff))
    plt.title('Throttle Data Differences')
    plt.xlabel('Difference')
    plt.ylabel('Frequency')
    plt.show()

    # Generate Steering Versus Line Graph
    x = np.arange(0, len(steering_real))
    plt.figure(figsize=(10, 5))
    plt.plot(x, steering_real, label='Real Data', color='blue')
    plt.plot(x, steering_predict, label='Predicted Data', color='red')
    plt.xlabel('Index Values')
    plt.ylabel('Values')
    plt.title('Comparison of Real and Predicted Steering Data')
    plt.legend()
    plt.show()

    # Generate Throttle Versus Line Graph
    x = np.arange(0, len(throttle_real))
    plt.figure(figsize=(10, 5))
    plt.plot(x, throttle_real, label='Real Data', color='blue')
    plt.plot(x, throttle_predict, label='Predicted Data', color='red')
    plt.xlabel('Index Values')
    plt.ylabel('Values')
    plt.title('Comparison of Real and Predicted Throttle Data')
    plt.legend()
    plt.show()

# Main Loop
if __name__ == "__main__":
    data = importDataInfo()

    imagesPath, steerings, throttles = loadData('DataCollected', data)
```

```
    cycleData(data, imagesPath, steerings, throttles)
```

Implementation

```
import cv2
import numpy as np
import smbus
import os
from tensorflow.keras.models import load_model

# Get Current Directory
current_dir = os.path.dirname(os.path.abspath(__file__))

# Construct Path to Model
# Must Change to the Correct Name of Trained Model
model_path = os.path.join(current_dir, "model_V1.h5")

# Load the Trained Model
model = load_model(model_path)

# Create an I2C Bus Object
# I2C '0' Bus is SDA on 27, SCL on 28
# I2C '1' Bus is SDA on 3, SCL on 5
bus = smbus.SMBus(1)

# Function for Getting Image from the Webcam
def getImg():
    # Initialize Camera
    cap = cv2.VideoCapture(0)   # Can be Updated to Appropriate Camera

    # Check if the Camera Opened
    if not cap.isOpened():
        print("Failed to Open Camera")
        exit()

    # Get an Image from the Webcam
    ret, img = cap.read()

    # Check if the Frame was Read
    if not ret:
        print("Failed to Read Camera Frame!")
        exit()

    # Release the Webcam Capture
    cap.release()
```

```
    return img

# Function for Preprocessing Image
def preProcess(img):
    img = img[875:1150,:,:]                          # Crop
    img = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)       # Convert RGB to YUV
    img = cv2.GaussianBlur(img, (3, 3), 0)           # Blur
    img = cv2.resize(img, (200,66))                  # Resize
    img = img / 255                                  # Normalize

    return img

# Function for Sending I2C Data to Control System
def sendData(address, direction, steering_data, throttle_data):
    # Create a I2C Bus Object for the Arduino Mega 2560 REV 3
    # I2C '0' Bus is SDA on 27, SCL on 28
    # I2C '1' Bus is SDA on 3, SCL on 5
    bus = smbus.SMBus(1)

          bus.write_i2c_block_data(address,    direction,    [steering_data,
throttle_data])

# Main Loop
if __name__ == "__main__":
    while True:
        # Get Image
        img = getImg()
        img = preProcess(img)

        # Expand Input Data
        input_data = np.expand_dims(img, axis=0)

        # Get Prediction
        prediction = model.predict(input_data)

        # Save Corresponding Array to Variable
        steering = float(prediction[0])
        throttle = float(prediction[1])

        # Correct Out of Range Data
        if steering < -1:
            steering = -1
        elif steering > 1:
            steering = 1

        if throttle < 0:
```

```
    throttle = 0
elif throttle > 1:
    throttle = 1

# Convert Data to Usable Bytes
if steering <= 0:
    steering = (steering + 1) * (90 - 75) + 90
elif steering > 0:
    steering = steering * (140 - 90) + 90

throttle *= 127

# Round to Remove Decimals
steering = round(steering)
throttle = round(throttle)

# Correct Out of Range Data
if steering < 75:
    steering == 75
elif steering > 140:
    steering == 140

if throttle < 0:
    throttle == 0
elif throttle > 127:
    throttle == 127

# Check if Direction Needs to be Changed
if throttle < 0 :
    direction = 1
    throttle *= -1
elif throttle > 0:
    direction = 0

# Send Via I2C to Control System
controlSystemAddress = 0x04
sendData(controlSystemAddress, direction, steering, throttle)
```

Appendix E: Build

Build Components			
Part Name	Price	QTY	Total Price
Power Wheels Fire & Rescue Jeep	$ -	1	$ -
NVIDIA Jetson Nano Developer Kit (4GB)	$ 149.00	1	$ 149.00
Micro Center 64GB Micro SD Card	$ 4.72	1	$ 4.72
5mm Orange LED Diode with Resistor	$ 0.07	56	$ 3.91
5mm Red LED Diode with Resistor	$ 0.07	28	$ 1.96
Intel AC8265 Dual Mode Wireless Module	$ 21.99	1	$ 21.99
Arduino Mega 2560 REV 3	$ -	1	$ -
Mighty Max 12V 9Ah Rechargeable SLA Battery	$ 24.99	2	$ 49.98
Uxcell ON/OFF Keylock Switch	$ 6.49	1	$ 6.49
Waterproof 12V DC Car Cigarette Lighter Socket	$ 4.25	1	$ 4.25
BTS7960 43A High Power Motor Driver Module	$ 10.99	2	$ 21.98
12V-5V 3A Micro-USB Buck Converter Step Down Module	$ 8.76	1	$ 8.76
DC-DC Buck Converter Step-Down Module	$ 1.83	2	$ 3.66
12V DC 1-Channel Relay Module	$ 1.75	1	$ 1.75
RP-SMA Male - RPSMA Female 24" Extension Cable	$ 8.99	1	$ 8.99
SPDT 3 Pin 12V DC Waterproof Switch	$ 5.49	1	$ 5.49
80KG Digital Servo 8.4V Metal Geared	$ 49.99	1	$ 49.99
HC-SR04 Ultrasonic Sensor Modules	$ 5.96	2	$ 11.92
PLA-CF FDM Printed Parts	$ 3.40	1	$ 3.40
SPDT 3 Pin 12V DC Switch	$ 8.23	1	$ 8.23
ABS-Like Resin DLP Printed Parts	$ 3.48	1	$ 3.48
14 AWG Wire Splitters	$ 9.00	1.19	$ 10.71
25' CAT 6 Ethernet Cable	$ 9.79	1	$ 9.79
FlySky FS-i6X 6 Channel RC Transmitter and Reciver	$ 55.99	1	$ 55.99
14 AWG Wire Black and Red Copper Wire	$ 0.25	10	$ 2.46
Logitech C920x HD Pro Webcam	$ 69.99	1	$ 69.99

Figure 94. Bill of Material Build Components.

PCB Components			
Part Name	Price	QTY	Total Price
PCB Shield Board	$ 0.72	1	$ 0.72
JST B2B-XH-A(LF)(SN) Connector	$ 0.15	2	$ 0.30
JST B3B-XH-A(LF)(SN) Connector	$ 0.20	3	$ 0.60
JST B4B-XH-A(LF)(SN) Connector	$ 0.22	4	$ 0.88
JST B5B-XH-A(LF)(SN) Connector	$ 0.27	2	$ 0.54
JST B6B-XH-A(LF)(SN) Connector	$ 0.29	1	$ 0.29
JST XHP-2 Connector	$ 0.10	2	$ 0.20
JST XHP-3 Connector	$ 0.11	3	$ 0.33
JST XHP-4 Connector	$ 0.11	4	$ 0.44
JST XHP-5 Connector	$ 0.13	2	$ 0.26
JST XHP-6 Connector	$ 0.15	1	$ 0.15
5mm Red LED Diode with Resistor	$ 0.07	1	$ 0.07
BSS138 N-Channel MOSFET	$ 0.17	2	$ 0.34
10K 1/10W 0805 Resistor	$ 0.16	4	$ 0.64

Figure 95. Bill of Material PCB Components.

Hardware Components			
Part Name	Price	QTY	Total Price
M2 x 6mm Flat Head Screw	$ 0.03	4	$ 0.12
M3 x 6 Flat Head Screw	$ 0.03	3	$ 0.09
M3 x 10mm Button Head Screw	$ 0.08	12	$ 1.02
M3 x 16mm Button Head Screw	$ 0.69	1	$ 0.69
M3 Nylon Stop Nut	$ 0.65	1	$ 0.65
M3 Washer	$ 0.09	6	$ 0.54
M3 x 12mm Button Head Screw	$ 0.55	4	$ 2.20
M3 Nut	$ 0.23	4	$ 0.92
M4 x 10mm Button Head Screw	$ 0.55	1	$ 0.55
M4 x 14mm Button Head Screw	$ 0.65	2	$ 1.30
M4 Washer	$ 0.15	6	$ 0.90
M4 Lock Washer	$ 0.17	3	$ 0.51
M4 Nylon Stop Nut	$ 0.15	3	$ 0.45
1/4" x 0.5" Bolt	$ 0.27	1	$ 0.27
1/4" Washer	$ 0.11	3	$ 0.33
1/4" Lock Washer	$ 0.15	1	$ 0.15
1/4" Nylon Stop Nut	$ 0.15	1	$ 0.15
1/8" x 1" x 13.25" Aluminum Bar	$ 3.00	1	$ 3.00

Total	$ 538.49
Tax	$ 39.04
Final Total	$ 577.53

Figure 96. Bill of Material Hardware Components and Total.

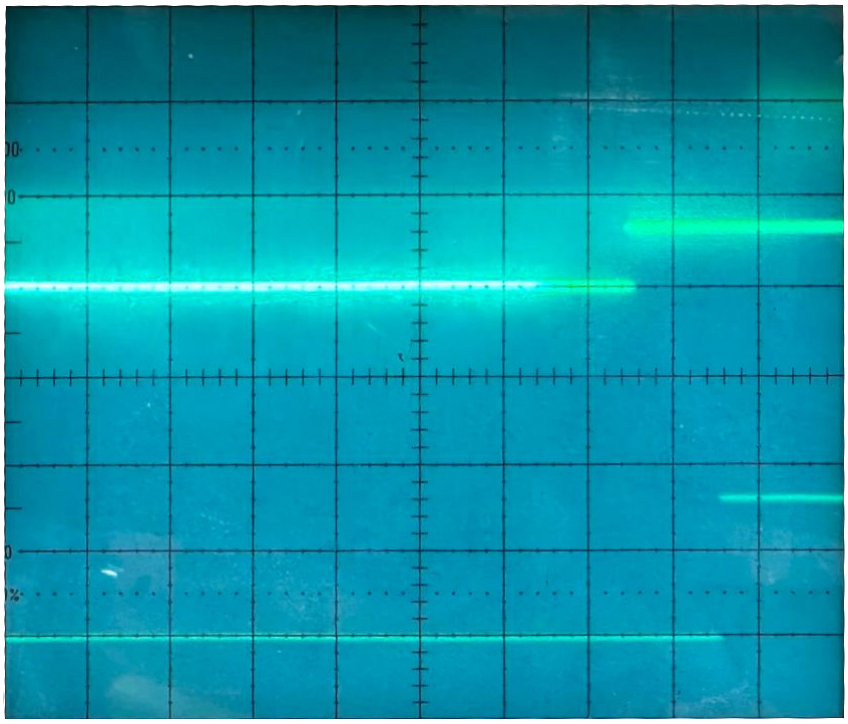


Figure 97. RC PWM Signal for Left/Reverse RC Signals (Bottom Signal).

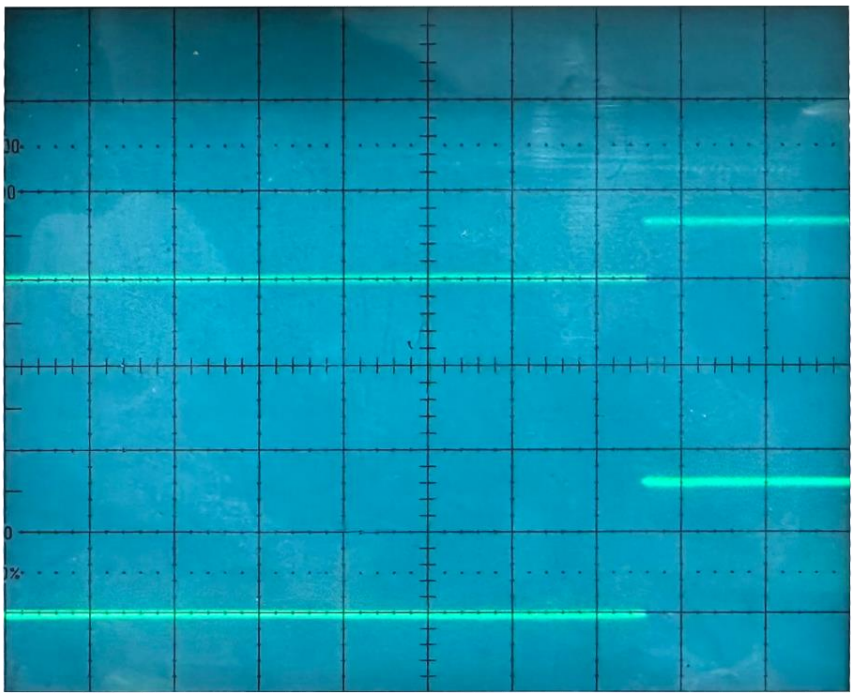


Figure 98. RC PWM Signal for Sticks Centered (Bottom Signal).

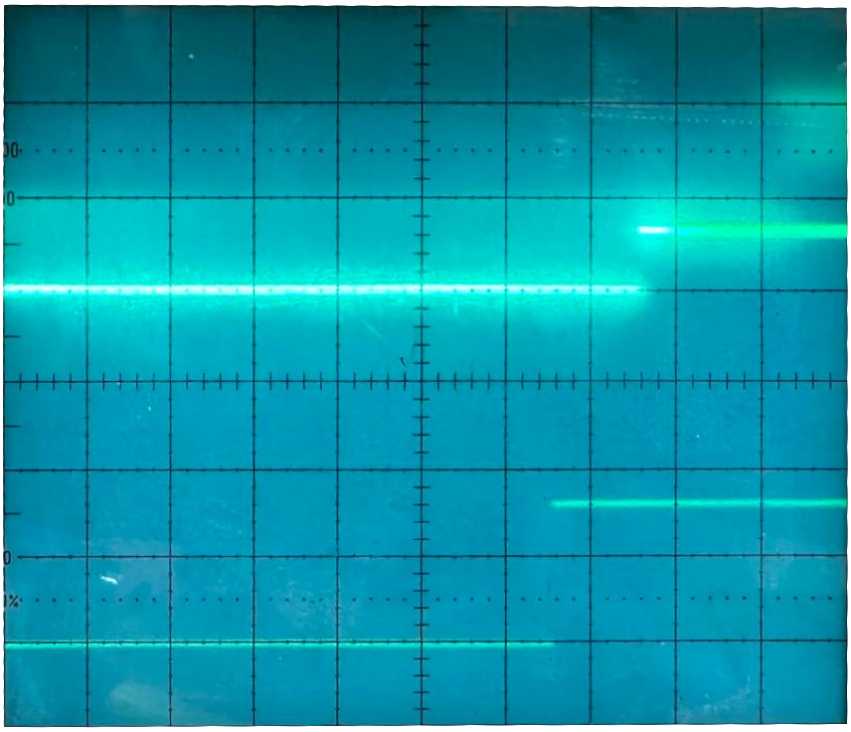


Figure 99. RC PWM Signal for Right/Forward RC Signals (Bottom Signal).

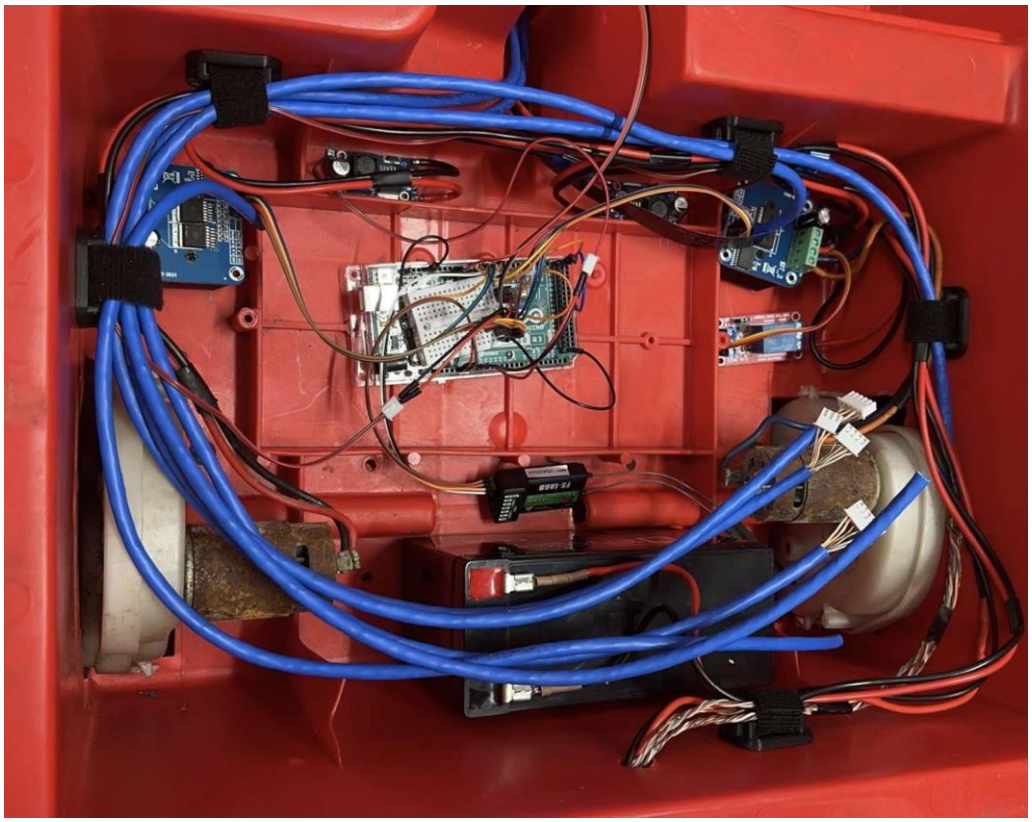


Figure 100. BTS7960 Motor Controller Integration Prototyping.

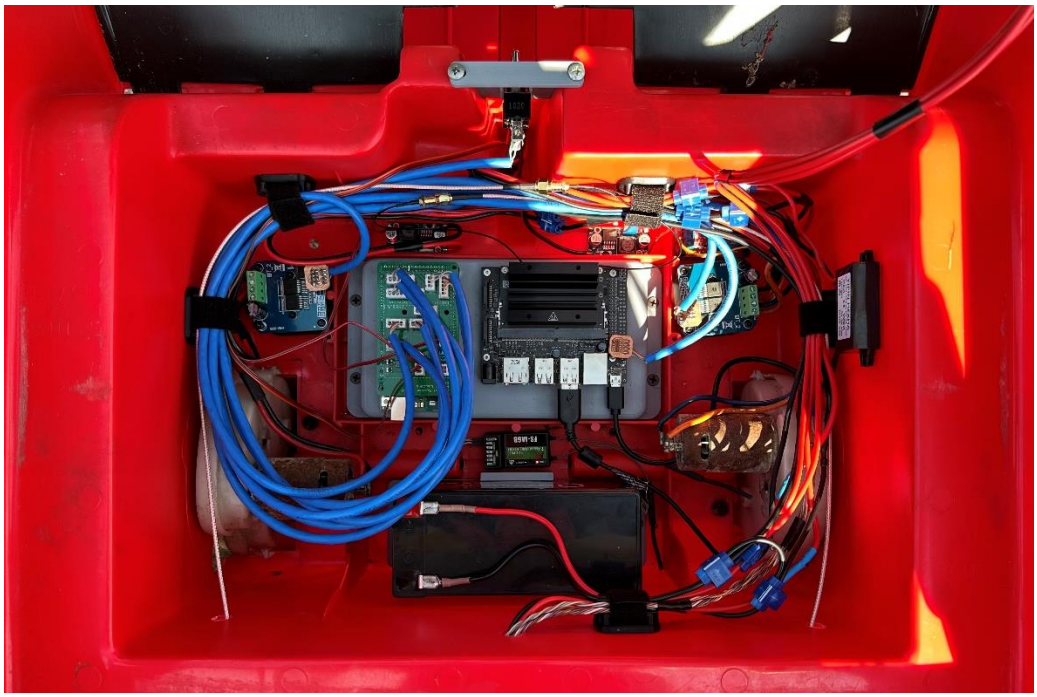


Figure 101. Completed Electronic Bay.

Figure 102. Completed Side View.

Figure 103. Completed Front/Back View.