Solar-Powered Autonomous Beach Buggy

A, D, V
Department of Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract*—In today's world, renewable energy sources are becoming increasingly important in order to combat climate change. In addition, autonomous vehicles are becoming smarter and easier to implement, which translates to less accidents on the road and more time saved for commuters. In order to combine these technologies, we have created a module that connects with and sends drive commands to three separate solar-powered buggies. The buggies were created by teams of mechanical engineers, and our module uses a combination of sensors attached to these buggies and machine learning models to generate and output a steering angle and throttle value to control them.**

***Index Terms*—Robot programming, image classification, supervised learning, artificial neural networks, computer simulation, autonomous systems, intelligent vehicles, robot vision systems, solar panels, solar power generation.**

I. Introduction

All across the world, millions of cars crowd the roads and spew tons of greenhouse gases into the air. This can be very deadly for drivers and their passengers; In the United States, 37,133 deaths occurred as a result of car accidents in 2017 alone. Even safe driving is harmful to the environment, so how can these problems be answered? Duke Energy and UCF are seeking this answer via our project, the interdisciplinary Solar Powered Beach Buggy challenge. We have worked with three teams of mechanical engineers who have designed and built buggies that will be driven by our module, a system of ultrasonic sensors, a depth camera, and a processor. Our project explores two possible solutions to reduce fuel consumption and injury: the use of solar power to reduce emissions and the use of autonomous navigation to prevent the need for a human driver. We would consider it a success if it can in any way encourage or further the development of sustainably powered autonomous vehicles.

II. Design Overview

The current designs for autonomous vehicle systems vary widely, though the kinds of systems that can be realized on our budget are less numerous. We used these models and sensor designs as a basis for our research into how we wanted to build our system, and improved the system from that point. Our system was required to drive off road and in unmarked streets and parking lots, and so does not have the luxury of lines to follow. For this reason, a depth camera was more effective in determining the locations of obstacles, and ultrasonic sensors provided a last line of defense for detecting obstacles while the buggy navigates.

Our approach incorporated a Jetson Nano to process all the sensor data into instructions. Our sensor system will feed information to the Jetson which it will in turn feed into algorithms that recognize obstacles and determine the buggy's surroundings. The Jetson will also help in determining instructions, as the location and orientation of the system will directly affect how the buggy needs to steer. Once the necessary instructions are determined, these will be outputted to the mechanical engineers' motor controller which will turn them into pulses that will instruct the speed at which the motors will turn and the angle that the buggy will steer.

III. Goals, Objectives, and Function

The goal of this challenge was to design and create a system to be used on three beach buggies built by the Black, Gold, and Blue mechanical engineering teams. This buggy had to be capable of driving itself in different environments while avoiding obstacles. Originally the buggy was going to be tested on a beach, but early on in our meetings with the MAE advisors it was determined that we would be testing on paved and unpaved locations near UCF instead.

This buggy had to be able to drive at a brisk pace of three miles per hour. The mechanical engineering students also needed to ensure that their buggy could keep this pace while it drove for at least twenty miles and be powered completely by solar energy. While it drove along, it had to be able to avoid obstacles and create a new path to move around these obstacles. This autonomy would be a result of our module, which needed to be able to attach to each buggy separately. In order to facilitate the process, we created an Interface Control Document early on, per Dr. Kurt Stresau's advice. The mechanical engineering team buggies had varying degrees of success communicating with our module, but our simulation and test vehicle discussed later in the paper were able to communicate and take controls consistently.

IV. Broader Impacts

The autonomous Beach Buggy we helped to build is not just our senior design project; it represents all that we have learned so far and what we will continue to learn as we progress in our field. It will be the last project we create as UCF students and will certainly be the most impactful. We hope that the autonomous Beach Buggy accomplishes a myriad of objectives and creates lasting impacts on societies throughout the world.

If it is successful, the technology and knowledge created could reduce carbon footprints by making solar-powered vehicles more desirable. In 2016, over two billion tons of carbon dioxide was released by transportation in the United States, far surpassing the amount released from the burning of fossil fuels for power generation. If even a small amount of vehicles started using renewable energy such as solar we could make a difference in saving our environment.

An even more important aspect of an intelligent automobile is the number of accidents it could reduce. Computers are faster, calmer, and more focused when in stressful situations than humans. A car will not panic if it is about to hit another car and will hit the brakes quicker, angle the car to reduce impact, and in general, make the best decision without fear. A smart car always will obey the law and drive defensively in the process, minimizing the number of situations that will even lead to an accident. Removing people from the equation will make driving, which is an unavoidable consequence of civilization, a safer activity that is no longer clouded by human error. With the creation of our solar-powered autonomous beach buggy, we hope to pave the future for these ideals. We also hope that the success or even the failure of our project will contribute to the overall outcome of a society with vehicles that do not damage the environment or living things.

V. Budget and Financing

Duke Energy sponsored every mechanical engineering team with 2000 dollars, 15 percent of which went to our computer science team. This meant that we had a total of 900 dollars for purchasing any parts and services that were needed to make our autonomous system a reality. Since we had to provide a brain to three beach buggies, we created a single solution that was used for each buggy and was purchased using our portion of the budget. This way, we only needed to spend money on one set of parts rather than spend it on one for each buggy. We also were able to get a MYNT EYE S that the purchasing department already had, which gave us a lot of breathing room in our budget. We ended up using only $476.47 of our $900 dollar budget, and we spent it on these items:

- Hardware for doing computationally intensive tasks (Computer Vision & Machine Learning):
  - NVIDIA Jetson Nano
    To be used for running object recognition and path planning
- Hardware to be used for the collection of the input data:
  - Elegoo HC-SR04
    Ultrasonic sensors to be used as a failsafe for stopping the buggy
  - MYNT EYE S
    To be used for determining depth of obstacles as well as their location relative to the buggy
- Hardware to be used for testing, connecting sensors, buggy, and Jetson:
  - Exceed RC 1/16 2.4Ghz Magnet EP Electric RTR Off Road Truck
    RC car used to test our autonomy system before the mechanical engineering buggies are finished
  - HiLetgo L293D DC Motor Drive Shield Stepper Motor
  - XT60 Plug Male Female Connector with Sheath
  - SunFounder PCA9685 PWM Servo Driver
  - ELEGOO UNO R3 Board
    Arduino board used for communicating between Jetson and motors
  - L298N Motor Controller
  - DROK Voltage Regulator
  - Ovonic Lipo Battery
    Extra battery for RC test vehicle
  - HTRC Lipo Charger
    Charger for the extra battery

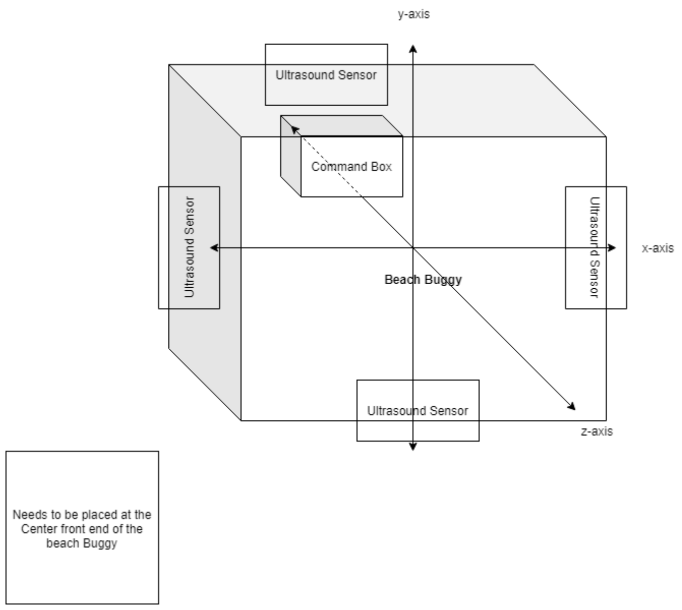

Fig. 1. Module placement per the Interface Control Document.

VI. Interface Control Document

The purpose of this autonomy system is to provide instructions for the buggies to follow in order to get to their destination without manual human control of the vehicle. In order to better accomplish this we, the computer science team, detail in an Interface Control Document (ICD) how the buggies interfaced with our system. Our autonomy system is comprised of a Jetson Nano as our AI computing device, a MYNT EYE S as our depth camera, and an array of ultrasonic sensors for reliable close-range sensing. Values for steering and throttle are sent to each buggy's Arduino via a "drive command" which consists of a float value for steering $\theta \in [-1, 1]$ and a float value for throttle $\alpha \in [-1, 1]$. θ values of -1.0, 0, and 1.0 mean max steer left, straight ahead, and max steer right respectively. α values of -1.0, 0, and 1.0 mean max throttle backwards, no throttle, and max throttle forwards, respectively.

VII. Data Pre-processing

The depth image data that is produced by the depth camera on its own is not as effective for training of our machine learning models as it could be. This is because of how the depth is encoded in the image. Values in the collected depth images ranged from $\sim$250 (very close) to $\sim$35000 (very

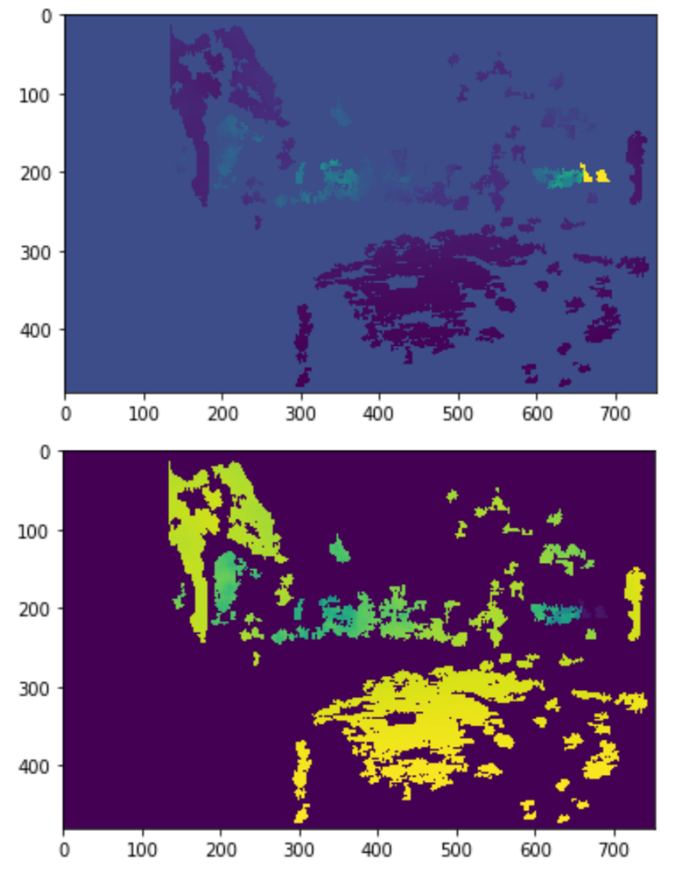

Fig. 2. The top image is what comes straight from the MYNT EYE S depth camera. The bottom image is the same image after having gone through pre-processing.

far). When the camera is not confident enough for the depth calculation for a pixel, it places a value of 10000 for that pixel in the image. This is a problem since this is within the range of values for valid depths and it is not certain that anything actually exists at the distance in front of the camera. To solve this issue, we create a new image D, and use the original depth image I to construct it by using the following equation:

$$D[i][j] = \begin{cases} I[i][j], & 10000 < I[i][j] > 10000 \\ 40000, & I[i][j] = 10000 \end{cases} \tag{1}$$

We place the same values in D that are in I only if they are not equal to 10000. Otherwise, we place a value of 40000 for that pixel in D. This essentially moves some of the uncertain data out of the way and has our machine learning models better understand the valid depths during training.

To help our machine learning models converge, we create a new image N by using the following equation:

$$N[i][j] = \frac{40000 - D[i][j]}{40000}. \tag{2}$$

Here, the max value expected to be in any image (40000) is subtracted by the value in $D[i][j]$. We then divide by 40000 to normalize the image by having all values be between the range [0,1]. The larger values in N represent the smaller depth values in the original depth image I. We do this because the smaller depth values from the original image are most useful for our neural networks.

VIII. Blocked and Direction Models

To generate our drive commands, we must know whether the vehicle is in a blocked or unblocked state. If blocked, we must also figure out whether we need to turn left or right. To make these calls autonomously we are using two neural networks: the Blocked Model and the Direction Model. Both models achieve $\sim$90% accuracy.

A. *Blocked Model*

The Blocked Model is a neural network that decides whether or not the buggy is in a blocked or unblocked state given the depth image from our MYNT EYE S depth camera. The input layer is a maxpool layer with dimensions 480 by 752 (the size of the depth image). The filter dimensions for this layer is 20 by 20. After this, the output from the maxpool layer is flattened and fed into a hidden layer with 16 neurons. This is followed by another hidden layer with 8 neurons that outputs to the output layer that consists of just one neuron with a sigmoid activation function that outputs a value in the range [0,1]. Any output value greater than or equal to 0.5 would be considered blocked. Any output value less than 0.5 would considered unblocked. This model is similar to that of a model provided by NVIDIA for their Jetbot [1], but we require an additional model for turning left or right to unblock the vehicle.

B. *Direction Model*

The Direction Model is a neural network that decides which direction the buggy needs to turn based on the input depth image. The Direction Model has the same structure as the Blocked Model. An output value greater than or equal to 0.5 is considered a need to turn right. An output value below 0.5 is considered a need to turn left.

IX. Autonomy System

The Autonomy System is what generates drive commands autonomously. It does this by taking in sensor data about the buggy's environment and making decisions based on that sensor data. The Autonomy System consists of the following ROS nodes: the Joystick Node, the Autonomous Driver Node, the Filter Node and the Arduino Node.

A. *Joystick Node*

The Joystick Node listens to all messages coming from our Xbox One controller and maps those messages to messages that are meaningful to other nodes in the Autonomy System. Fig. 3 shows what we are mapping each button to. The left joystick and right trigger are used for controlling steering and throttle, respectively. We also toggle the sign of the throttle value by using right bumper button. The Y and B buttons are used for setting the Autonomy System to User Drive Mode and

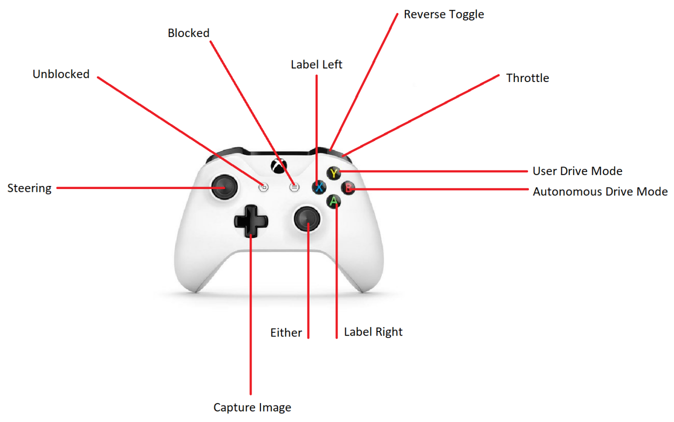

Fig. 3. Xbox One contoller button mapping.

Autonomous Drive Mode, respectively. We also mapped some buttons for aid in the collection of our data set. We first set the label for the image we are about to capture using the back button (Label ”Unblocked”), start button (Label ”Blocked”), X button (Label ”Left”), A button (Label ”Right”) and right joystick press (Label ”Either left or right”), then capture the image using the down button on the D-pad.

B. *Autonomous Driver Node*

The Autonomous Driver Node subscribes to messages published by the Joystick Node. This node also subscribes to messages coming from the MYNT EYE S depth camera and messages that come from the Arduino Node (ultrasonic data). The Autonomous Driver Node has two modes used for passing on drive commands: the User Drive Mode and the Autonomous Drive Mode.

1) User Drive Mode: When User Drive Mode is selected, the node only publishes the drive commands given by the user (drive commands coming from the Joystick Node).

2) Autonomous Drive Mode: When Autonomous Drive Mode is selected, we make use of the Blocked Model, Direction Model and ultrasonic data that is given by the Arduino Node. If we ever receive an ultrasonic distance value of less than 150 (1.5 meters), we set the Autonomy System back to the User Drive Mode. Every time we receive a new depth image from our depth camera, we pre-process that image and have our Blocked Model decide whether the vehicle is in a blocked or unblocked state. If it decides the vehicle are unblocked, the node publishes a drive command with $\alpha = 1$ and $\theta = 0$. If it decides the vehicle is blocked, it passes the same image to the Direction Model to decide whether the vehicle must turn left or right. If it decides a left turn is needed, the node publishes a drive command with $\alpha = 1$ and $\theta = -1$. If it decides a right turn is needed, the node publishes a drive command with $\alpha = 1$ and $\theta = 1$.

C. *Filter Node*

To help ensure a smoother ride, we use this Filter Node to treat the drive commands as a signal and apply a low pass filter to that signal. This reduces the amount of jerk and puts less stress on the buggy’s hardware. We implement this low pass filter by use of a weighted average of the previous drive command and the current drive command. The Filter Node proved to be helpful in simulation for smoothing the movement of the buggy. The Filter Node was not as useful when we integrated with any one of the physical vehicles as friction seemed to be enough of a dampener.

D. *Arduino Node*

The Arduino Node subscribes and executes the drive commands being published from Filter Node. The Arduino Node also publishes an array of distance values (in centimeters) from each of the ultrasonic sensors that are fixed to the vehicle. This Node publishes its gathered ultrasonic data every 300 milliseconds to the Autonomous Driver Node. The logic required to successfully execute the given drive commands was to be implemented by each of the mechanical engineering teams.

X. SIMULATION

One important factor of our project is the separation into teams based on discipline. This means that the engineers who build and design the buggy were not a part of building the module that controls the buggy, and our team was not involved in creating the buggy itself. For this reason, we were not be able to test the module’s ability to process input information from the sensors on the buggy until shortly before the entire module was due. For this reason, in order to test our module’s ability to process the sensor data, we needed to create simulated sensors that “collect” data from a simulated environment. By accomplishing this, we were able to create a model that was able to effectively traverse a simulated environment similar to the one the real buggy had to navigate through. When the real buggy was built we were hoping the module would be able to control it with little to no adjustments. Although the simulation ended up being very different from the real world, it did help us to visualize our depth and ultrasonic sensor data.

A. *ROS*

ROS stands for Robot Operating System, and it serves as an operating system for a robot, which means that it handles control of low level devices and hardware abstraction. Normally these components would exist entirely in isolation until code and drivers were written to handle different hardware inputs and outputs, different protocols, and different means for controlling and reading information from the various parts of a robot. ROS handles this process automatically, so nodes can either request information from another node or publish information continuously to a topic visible by other nodes. This information could be instructions, images, sensor readings, or anything else relevant to the function of a robot.

ROS divides all of the tasks a robot must perform and its different parts into individual processes, or nodes, and coordinates and keeps track of these nodes and their activity.

ROS also makes it much easier to allow these nodes to communicate, which can improve performance of a robot and make its actions more efficient. This is accomplished by storing certain values, such as a temperature, in the ROS master process and allowing all other nodes to access these values. Another way ROS allows nodes to communicate is through direct message passing. This level of cooperation is not necessarily just between sensors and one processor, as ROS permits a fully distributed format whereby multiple computers and sensor systems work to control the robot as a whole. Sharing computation, storing important values and allowing for message passing makes ROS an invaluable tool when designing robots, as orchestrating the cooperation between components would otherwise be a difficult process. Listed below are a couple of ROS packages we used on this project

1) rviz: One package that helps us display important data when simulating is Rviz. Rviz is used to visualize what the buggy's sensors are telling it and how the Nano is processing that data. We use Rviz as a means of seeing through our buggy's camera, as trying to understand what the buggy sees based on numerical values of pixels or groups would be next to impossible. By being able to see what our buggy sees, we can quickly identify issues with the camera or the processing of the camera data. Rviz also helps us visualize depth information from the camera as well as the path the buggy intends to take through the environment.

2) rosserial arduino: One important function our robot has is its ability to communicate with a motor controller that is handled by the mechanical engineering teams. The teams all used an Arduino, and ROS has packages that allow our Nano to speak with an Arduino over its Universal Asynchronous Receiver/Transmitter, or UART. It treats the Arduino as if it were a ROS node, and this means it can publish/subscribe to messages from other systems in the buggy. The Arduino can then use those messages to send certain PWM pulses to control the steering and throttle intensity.

B. Gazebo

Gazebo is considered an excellent environment for simulating robots that implement object avoidance and computer vision, and it is the software that we chose to use. It works in conjunction with ROS. We use ROS for communication between components, such as depth camera to Jetson and Jetson to Arduino. Gazebo can also use ROS services when simulating robots, and this was a very accurate way of modeling how our buggy functions, even using the same signals and message system. This comes at no cost also, as both Gazebo and ROS are open source and free to use. In addition, Gazebo can be set up to be as lightweight as possible, and many of its plug-ins are lightweight as well. Despite their advantages these technologies are hard to figure out and learning how to use them required a large time investment. There were difficulties getting the software set up and working properly, but with some tweaking and some equations learned in physics we were able to get it working. Outlined below are the components that needed to be simulated individually.

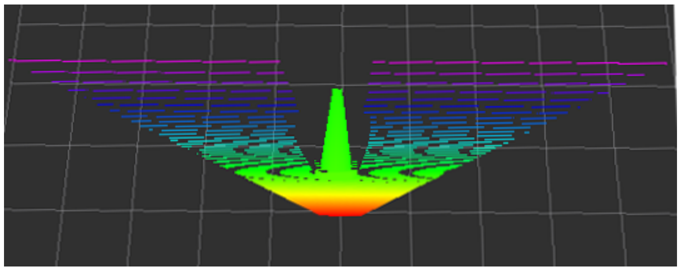
Fig. 4. Simulated depth camera image.

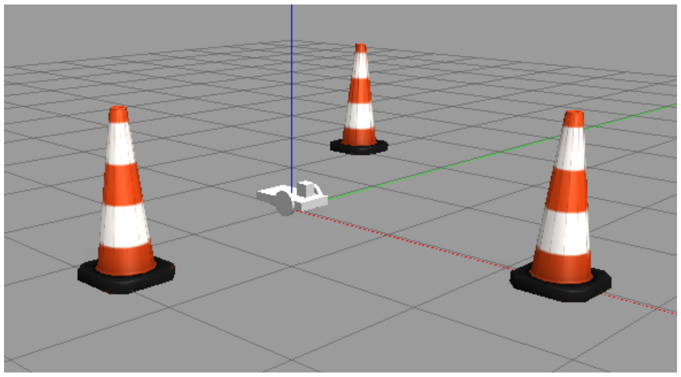
Fig. 5. Simulated environment.

C. Ultrasonic Sensors

In order to implement a simulated ultrasonic sensor, we needed to have some way of simulating sound waves that could be made in Gazebo. This was thankfully easy to implement, as Gazebo's built in library contains an ultrasonic sensor that we were able to use. The buggy was also programmed to stop when the sensor registers a very short time period between when a sound is fired from the sensor and when it returns.

D. Depth Camera

The camera of the car is another component that needed to be simulated. The camera is probably the most important sensor and the one that the buggy uses to "see". This can also be simulated in Gazebo. This built-in camera is affixed to the front of a simulated buggy in the same spot the physical camera is on the real buggy. After "mounting" the simulated camera, it captures the images it sees onto files at a rate based on the frames in the simulation. These files include information about the depth of objects away from the camera. This provides enough data for an algorithm to run object recognition, and then the buggy can drive accordingly.

E. Simulated Environment

If the sensors of the buggy are simulated, then an environment for the buggy to drive in and sense must be simulated as well. This was done quickly and effectively in Gazebo using their built-in physics system. Objects were placed into a "world" for the buggy to sense and navigate within.

XI. Test Vehicle

To facilitate our integration with the MAE teams we created a small scale test vehicle using an RC car that utilizes the same command box we used for the actual beach buggies with a similar electrical set up. This allows us to physically test our code, wiring, and functionality. We used the electronic speed control and servo motor that came with our RC car, an Exceed Magnet, to drive the car. Those two were hooked up to a PCA9685 servo driver controller which was controlled using an Arduino Uno. Just like our actual beach buggies we pass the standardized throttle and steering values from our Jetson Nano to be interpreted by the Arduino.

XII. Hardware Integration

Our actual integration with each mechanical team was very hectic. We had to match up schedules with a different team each week while making sure there was enough sunlight for testing battery usage and charging controller functionality. Each team before we met up would send us their Arduino code for general tweaking and installing of the proper functions and lines of code for ROS node functionality. They each had different configurations for steering and throttle so it made sense to defer the coding of their controls to each team individually. On the actual day of testing we would try to run each car but there was always bugs to grind out either in the code, the wiring, or even the physical components. Things would break and we had to wait while parts were reordered so we could test another day. The day before the showcase was spent visiting each car and making sure everything was working to maximum effect. In the end we got each vehicle to move and somewhat work autonomously with greater effect depending on the robustness of the vehicle architecture.

XIII. Conclusion

Through hours of hard work, we were finally able to achieve our goal for designing and creating a module that could autonomously deliver steering and throttle instructions to an Arduino. This concept was proven by reliably driving our test RC vehicle around obstacles as well as sending commands to avoid obstacles to the ME buggy. These commands were based on depth camera data as well as ultrasonic readings, just as we had planned last spring. We hope our example will show that autonomous and solar powered vehicles are viable projects, even on a tight budget.

References

[1] N. AI, "Jetbot examples." [Online]. Available: https://github.com/NVIDIA-AI-IOT/jetbot/wiki/Examples