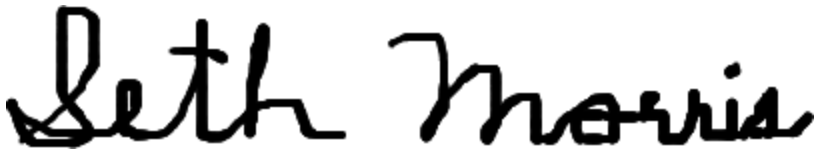
Autonomous Car: CpE & Mech

Seth Morris

12/15/17

Capstone Design ECE 4440

Signatures



Statement of work:

Seth Morris: Wrote the computer vision software to identify and track lanes, cars, and pedestrians. Setup the embedded computer that the software ran on. Will continue to work with the rest of the team next semester to integrate other sensors and develop a path routing system.

Grant Gibson: Sensors and modeling.

Edgar Hoover: Mechanical CAD work and chassis.

David Horta: Mechanical CAD work and chassis.

Frank Laubach: Motors and actuation.

Dillan Mason: Motors and actuation.

Spiros Skenderis: Sensors and modeling.

Table of Contents

Contents

Table of Figures

Abstract

Our group consists of six mechanical engineering majors and myself*. We are building a 1/5th scale autonomous car for our respective capstones. The car has three wheels, a driven wheel on the back and no non-driven wheels on the front. The two non-driven wheels are used to turn, and actuated with a servo-controlled rack and pinion system on the front. Several different types of sensors will be mounted on the body including: a GPS sensor, ultrasonic sensors, a 360 degree field of view LIDAR module, and two cameras for stereoscopic vision. For my capstone, I have primarily worked with the camera feed and an embedded computer to make an embedded computer vision system. The computer vision system has lane detection and following capabilities, as well as pedestrian and car detection/tracking capabilities. This system runs on a ODROID XU4 embedded computer. Next semester I will work with the mechanical engineering majors to combine the other sensor input with my computer vision software to dynamically route paths that the car will follow.

*As the only computer engineering major on the team, I am the sole author of this report.

Background

Making an autonomous car as my computer engineering capstone project was not something I had planned far in advance. Over the summer, I was making plans for the two engineering clubs I lead/led at the university: Rocketry and Hyperloop. Ever since founding the Rocketry club and taking lead of the Hyperloop team at the beginning of my 3rd year I recognized that I was half-intentionally mirroring Elon Musk. Musk is the CTO of SpaceX, a commercial rocketry company. He is also the single person most commonly associated with the Hyperloop concept, and his company SpaceX fund a competition which I fielded a team for. Musk is also the CEO of Tesla Motors, an electric car company that is at the forefront of bringing autonomous cars to market. I figured since I was already doing a Rocketry and Hyperloop club, I would complete the Musk triad and start an autonomous car club. Jokes aside, it is fairly obvious that autonomous cars are the future. Many of the major automakers have implemented partial autonomy, and nearly all of the rest have plans to [1]. There is also movement towards going electric. The automotive industry has a valuation of ~$4 trillion [2], so any revolution within it is going to creates lots of jobs for engineers. The global market for autonomous driving hardware components alone is expected to grow from ~$400,000,000 as of 2015 to $40,000,000,000 in 2030 [3]. I figured building an autonomous car would be a fitting capstone project for a large number of Mechanical, Electrical, and Computer engineering majors, including myself. It contained all the elements of a good capstone project: it was challenging, it was fun, it would be relevant experience for graduates going into the job market, and it was multidisciplinary. Multidisciplinary was a key aspect; in my previous club projects I had managed multidisciplinary teams to design projects of similar scope and ambition to that of building an autonomous car. I found that having a multidisciplinary team is near necessary for anyone project worth doing. Rockets have a *mechanical* frame with an *aero*dynamic shape made with advanced *materials* propelled by a *chemical* motor with an embedded *computer* that activates

electric matches needed for multi-staging and recovery. Autonomous cars are typically also *electric*. They need an embedded *computer*. They have a *mechanical* chassis. Almost all modern projects, at least the ones I consider most value-added, are multidisciplinary. As someone who already had experience leading large multidisciplinary projects, I figured it was my privilege and maybe even my duty to assemble and lead a multi-major capstone group based on a substantive idea that would develop skills directly applicable for graduates entering the job market.

In the first session of the capstone class, I got interest and signatures from ~20 electrical and computer engineering majors to pursue a full-sized and full-featured autonomous car. I had also set up a time to present to a mechanical engineering capstone class of 20 with the objective of getting all of them to join the project. But it was not to be. In a very sharp and dramatic turn, support for this rather ambitious project collapsed. Many of the other ECE students expressed skepticism that the project could be done in one semester, citing concerns about the mechs' one year schedule mismatch, and that the project could turn into a mess with so many people. Ironically, this lead to the number of interested ECEs fizzling away until I was the only one left. About 14 people in the mechanical capstone class still wanted to make the car, but the supervising professor set a restriction that only ⅓ of the class could work on any single project. In the end, there were 6 mechs and myself left. A group larger than most, but very far from my original aim. I include this story both to give note and context of my original ambitions for the project, and because three of the most important lessons I learnt came from the failure to assemble this grand team that I had envisioned. The lessons learnt were: 1: Different people have different motivators and different tolerance for risk. In hindsight, I shouldn't have I expected them to take on this much harder and riskier project without more motivators and guarantees. With Hyperloop and Rocketry, I had the relative luxury that every team member was a volunteer that really wanted to be there. 2: Crowd behavior is a powerful and swift force. When the first person notified the interest group he was out, it catastrophically collapsed within the ensuing hours. 3: That in the end, standing alone for something you believe in is better than becoming part of the crowd. It wasn't a good feeling presenting alone in front of all those people I had won and lost interest from, but it was a transitory feeling. After completing the ECE portion on my own, I have no regrets. I learned cutting edge software frameworks and hardware accelerants for AI, and have an end product I'm proud of and will continue working on in the future.

The original plan was to build a full-sized electric car. It seemed within the realm of possibility given that the majority of the Mechanical Engineering majors were members of the Baja race car team, where they build a full-sized car extracurricularly each year. We then got an offer from Perrone Robotics, a local company in the autonomous car space, to intern with them such that we would meet our capstone requirements through the internships. The general idea was to make a bolt-on autonomous kit having access to their resources and mentorship. For the full details of the offer, see Appendix A. They eventually retracted this offer after a month of negotiating. The reasons cited were that they had deadlines coming up and that it would take up too much of their paid staff's time to work with our group. I wanted to continue with the bolt-on

kit idea and apply it to my own truck, but the mechanical capstone professor said the university would neither support, fund, or recognize such a capstone project due to its risk. We defaulted to making our own 1/5th scale car. This would end up being our differentiator, having to make an autonomous system on a budget. The new plan was for me to do the embedded computer selection and write the computer vision software this semester (the deadline for computer engineering capstone projects), and the mechanical engineering majors would finish the 1/5th scale car early next semester. I would then work with them in the Spring to integrate the ultrasonic, GPS, and LIDAR sensors tand develop a path routing system. The entire project would be done by the end of the academic year (the deadline for mechanical engineering capstone projects).

The ECE courses that I drew upon during this project include:

- CS 2150 - Program Representation: Using bash and program compilation.
- ECE 3430- Intro to Embedded Computer Systems: Communication protocols to and from sensors.
- ECE 4435 - Computer Architecture & Design: RAM, swap memory, virtualization, and different instruction set architectures.
- ECE 4457 - Computer Networks: Wireless protocols. UDP and TCP messages.
- ECE 4501 - Advanced Embedded Systems: Multi-threading and using git.

Courses I could've potentially drawn on had I taken them prior include:

- CS 4414 - Operating Systems: Understanding the file system architecture.
- CS 4710- Artificial Intelligence: Understanding where and when to apply different AI and machine learning techniques.
- CS 4740 - Cloud Computing: Testing my program on computers with different specs to see which hardware accelerated it the best. Training neural networks.
- Special topics courses specific to computer vision.

I also drew heavily from my mechanical engineering courses and extracurricular project experiences in the areas of machine elements, motors, motor drivers, getting funding, and machining. Areas I feel I lacked relevant background knowledge in include making formal deals with a corporation, and forming and maintaining a team composed of people with different interests.

Constraints

Design Constraints

Qualifying arduinos and raspberry pis as "hobbyist" embedded computers (that wouldn't meet the capstone class requirements) imposed a significant constraint. I was already familiar with arduinos and raspberry pis. I had used arduinos in the avionics of rockets and raspberry pis in hyperloop and various entrepreneurial projects. The defense contractor I interned with last summer used a raspberry pi as their embedded computer, so I'm not sure it's fair to classify them

as “hobbyist”. The selection criteria on a new embedded computer was also particularly high for my project since it involved computationally intensive techniques, such as image recognition and neural nets. Having to choose a new embedded computer, combined with shipping, meant that I only got it a few days before the capstone fair. That being said, making Arduinos and Pis off limits forced me to look into new embedded computers such as the Parallela and ODROID XU4. I was particularly impressed with the ODROID and will likely continue to use it for other projects.

The ODROID did come with the unanticipated issue/feature of having an ARM architecture. Some of the software frameworks I used, such as tensorflow, did not have official binaries for ARM. This necessitated me compiling from source and finding custom binaries, which took days worth of time.

There was also another major limitation in using Python as my primary programming language. Early on, I had decided to write the framework in Python due to it being better supported by AI and computer vision relevant libraries. However, the fact that Python is an interpreted and not a compiled language did not cross my mind. Since it is interpreted, it normally only runs one line at a time, which is not good for something as parallel as computer vision and neural nets. I ended up using special libraries, frameworks, novel implementations of algorithms, and an external VPU to speed up the execution so that it could keep up with the camera feed.

Economic and Cost Constraints

Given that we are trying to build a car, cost constraints were significant. We ended up scaling back from a full-sized to 1/5th scale car in large part due to lacking funds. The reasons for this downgrade were a combination of lack of funds, lack of time, and lack of morale. Perrone going back on their offer was a major drain on all three. I had applied for the U.VA. Parent’s Fund grant early in the year (which we ended up not getting), and had also planned to apply for the Experiential Learning Fund grant as well as incorporating the capstone project as a CIO in order to get money from the student council. I let the deadlines for the last two lapse since Perrone’s offer of letting us work with their equipment and cars sounded near guaranteed. We wouldn’t have needed the extra funds in that case, but that case didn’t happen. We ended up spending over a month negotiating back and forth with them and passed by other opportunities. Lesson learnt: always have at least one strong back-up plan. Now we were left with only about $1,000 of available funding, the combined standard budgets of the CpE and ME capstones, forcing us to scale down.

External Standards

No particular standards were consciously adhered to other than good coding practice. If we had built a car that a person would’ve ridden in, there would most certainly have been many standards to adhere to such as the Federal Motor Vehicle Safety Standards [4]. The only

standards I know that were adhered to passively were the IEEE 802 standards for wireless ethernet [5].

Tools Employed

My development environment was rather simple. I ran an Ubuntu 16.04 operating system on both the embedded computer and on my laptop, which I used for development and testing. I did file manipulation through bash and editing with the nano and gedit text editors built into Ubuntu. I accessed the embedded computer using ssh or by attaching a keyboard to type directly into it, and transferred files between the two using GitHub. I did not use any IDE or analysis software.

Software frameworks I used include OpenCV, OpenGL ES, OpenCL, and Tensorflow. OpenCV is a computer vision library that I was familiar with prior to starting this project. I used it for lane detection and object tracking. OpenGL ES is a graphics library that can utilize a computer's GPU. Using the GPU can dramatically speed up the execution of computer vision and neural net algorithms. It came preinstalled on the operating system image I got from HardKernel, the company that manufactures and sells the ODROID. OpenCL is a framework for parallel processing on multiple processors. I needed it to keep the software up to speed with the camera feed. The OpenCV libraries utilized it automatically, so I did not have to learn or interact with it on a low level. Tensorflow, developed by Google, was behind the neural nets I used for vehicle and pedestrian detection; I had never used it prior and learned from open sources examples. I also used various libraries such as numpy, which I consulted the documentation of to learn how to use. Using the Android operating system, Caffe deep learning framework, and Robot Operating System libraries were explored, but ultimately not attempted because of time constraints and incompatibility with other tools I was already using. Using Android would have necessitated rewriting my program in Java or C++, Caffe did not support ARM architectures, and ROS forced an earlier version of OpenCV that didn't have the libraries needed for object tracking.

Ethical, Social, and Economic Concerns

Like the transition from horse and buggy to the original car, the transition to autonomous cars is expected to have a large social, cultural, and economic impact. Many of these impacts are beneficial, and derivatives of a presumed increase in safety. I say presumed because it has not yet been conclusively proven that autonomous cars are safer under all driving conditions. As of now, they have a much lower rate of accidents on highways, but that might be offset by as yet unexplored performance in urban areas and under extreme weather conditions. Cars like the Tesla Roadster are not fully autonomous either; they have partial autonomy that acts under the driver's supervision, so it is not yet fair to assign numbers to the record for fully autonomous cars. That being said, the results of partial autonomy do look promising. To date, the fatality rate when using a partially autonomous car is lower at 1 fatality per 130 million miles driven versus the nationwide average of 1 fatality per 90 million miles driven [6]. The technology is also expected to improve. Other impacts of autonomous car are derived from their use in combination with ride sharing technology. Services like Uber or Lyft would no longer require a paid driver,

making the use of both technologies significantly cheaper and more accessible to people of lower incomes and disabilities.

Environmental Impact

There is a strong correlation between a new model of a car having autonomous features and being electric, but one does not necessarily require the other. Trying to gauge the environmental impact from this angle is not as simple as saying electric cars are better for the environment than gasoline. Electric cars primarily draw power from the grid, and the majority of the grid's power comes from fossil fuels anyway. One must also consider the chemical byproducts and energy needed to make solar panels and lithium batteries.

Instead, the more clear and direct environmental impact is derived from autonomous cars presumably being safer. Car chassis's have been historically made purposefully heavy because of a Nash equilibrium between automakers. They don't want their car to get in a crash with a heavier one and be torn to pieces. It's bad PR. If cars are orders of magnitudes safer, automakers may consider lightening their frames. This would reduce the fuel/energy consumption of both gasoline and electric cars, because they would be using less gas/energy to push the weight of the car around.

Sustainability

Like environmental impact, sustainability is affected by autonomous cars most directly in a non-obvious way. Most people don't use their cars to the fullest extent possible; the vast majority may only drive them an average of 1 - 2 hours a day. The rest of the time, they rust and weather in a parking lot, garage, or driveway. Decreasing costs of ridesharing and mass transit through its combination with autonomy would likely increase the efficient use of vehicles. If stacked with employer policies of flexible hours, individuals may no longer need their own cars as ridesharing would be cheaper and more accessible. This would result in less cars per capita needing to be produced. Less cars produced means less energy consumed in manufacturing, as well as processing the needed raw materials. If the cars of the future are electric, it means less lithium mining and byproducts from the manufacture of solar panels.

If driverless vehicles can park themselves, this also makes it possible to have parking lots and garages outside of walking distance from the end destination. Space in the heart of cities taken up by parking could be used for public parks or the existing parking garage infrastructure could even be converted to vertical farms. This would also have a positive impact on local air quality.

Health and Safety

Assuming autonomous cars do have a lower rate of accidents than human drivers under the same conditions, health and safety will obviously improve. There will be less injuries and deaths resulting from car accidents. Another health impact could come from the emergent ability of people being able to get extra sleep in their car during their morning commute or long trips while

their car drives. Even if they don't sleep, there should be a reduction in stress and increase in productivity as they could do something else during that time.

Manufacturability

The addition of sensors is not expected to dramatically affect the manufacturability of cars, although it may increase the production time and cost. Manufacturing cars is inherently a rich man's (or rather big company's) game as the initial costs are so high. This impedes innovation and the proliferation of autonomy. One way to sidestep this would be to manufacture kits to turn non-autonomous cars autonomous. Perrone Robotics makes kits already, but uses them as a transitory state to test their autonomous software with the end intention of installing the sensors directly within the cars. Another company, Comma, makes plugin kits that only work with cars who already the necessary sensors. There is huge market opportunity in making a bolt-on sensor suite with an embedded computer running autonomy software. There would also be huge liability associated with designing, making, selling, and installing these kits.

Ethical Issues

There are several ethical questions surrounding who has liability in the case of an accident involving an autonomous car. Is the rider liable? Should he or she be expected to supervise the car? Is it the manufacturer? Should any damages be taken on them to incentivize making a better product? I would support the latter, provided that the manufacturer states the car is meant to be fully autonomous and doesn't qualify it as needing supervision. I don't think government regulation would be necessary in this case as free market forces would push towards safer cars. The manufacturer would be incentivized to make a safer car as that would mean less damages taken on by them, more marketability, and lower end costs/barriers to purchase on the consumer's side assuming that insurance premiums take the safety record of the car into account. No government regulation would mean much more innovation and accessibility of the market to new and smaller businesses.

Intellectual Property Issues

As stated previously, many companies are making autonomous cars, but all of them have the sensors embedded in the car. There is market opportunity in a bolt-on sensor suite that can turn non-autonomous cars autonomous. There are still existing patents that would have to maneuvered around though, such as Toyota's very broad patent "Intelligent navigation system" [7]. Claim 1 is "An intelligent navigation system for automatically navigating and maneuvering a motor vehicle" comprising a input device to receive destination information, a sensor to detect at least one maneuverability condition, internal memory, and a processor. I figured there was leeway with requiring a maneuverability condition, which I took to mean weather conditions or condition of the road. However, Claim 4 expands on Claim 1 by stating maneuverability conditions can include "fuel level of the motor vehicle, a speed of the motor vehicle, a horsepower of the motor vehicle, or a brake condition of the motor vehicle, a relative distance between the motor vehicle and a surrounding object, a relative speed between the motor vehicle and the surrounding object, a traffic light output state, or a lane boundary." That is very, very

broad and I don't see how any other company attempting at autonomous cars is not infringing on this patent.

Another market opportunity is applying autonomy to delivery services. Amazon and other distribution companies often have human drivers either directly employed or contracted by them. Amazon already has a patent on this topic: "Transporting network utilizing autonomous vehicles for transporting items" [8]. There is some wiggle room in that their claims reference transportation between two "network" locations. My interpretation is that a network location is a location within Amazon's distribution network, which doesn't include the end destination (i.e. the customer's residence). The claims also state the path taken will include "at least two path segments with a different autonomous vehicle utilized to transport the item along each path segment". So if the transportation is handled by only one autonomous vehicle, it would not infringe.

I have already mentioned the potential value of the combination of ride-sharing with autonomous cars. There is already a patent pertaining to this: "Driverless vehicle commerce network and community" [9]. It has thorough coverage of the idea, and I could not see any workarounds. I noticed it seems to be attributable to a single individual using a patent holding company, so he may possibly be interested in selling the rights to the patent or negotiating royalties.

I did not see any prior work that led me to believe my kit idea is not patentable, with the exception of Toyota's very broad patent. I will consider consulting a patent lawyer through the university regarding both the kit idea and automated delivery and logistics services.

Detailed Technical Description of Project

Six mechanical engineering majors and I have been and are continuing to work on a scaled autonomous car. The main components of the scaled car are the chassis, wheels, motor, servo, embedded computer, and sensors. The sensors used include a 360 degree Field of View LIDAR module, ultrasonic distance sensors, and a USB camera taking 720p video at 30 frames per second. An electric motor drives one wheel towards the back of the car for propulsion and braking. A servo actuates a rack and pinion connected to two wheels in the front to steer. This configuration changed from the beginning of the semester mainly due to budgeting.

Figure 1: CAD model of the scaled car. The laptop will be replaced by the ODROID-XU4.

Figure 2: Old wiring diagram made prior to Perrone offer.

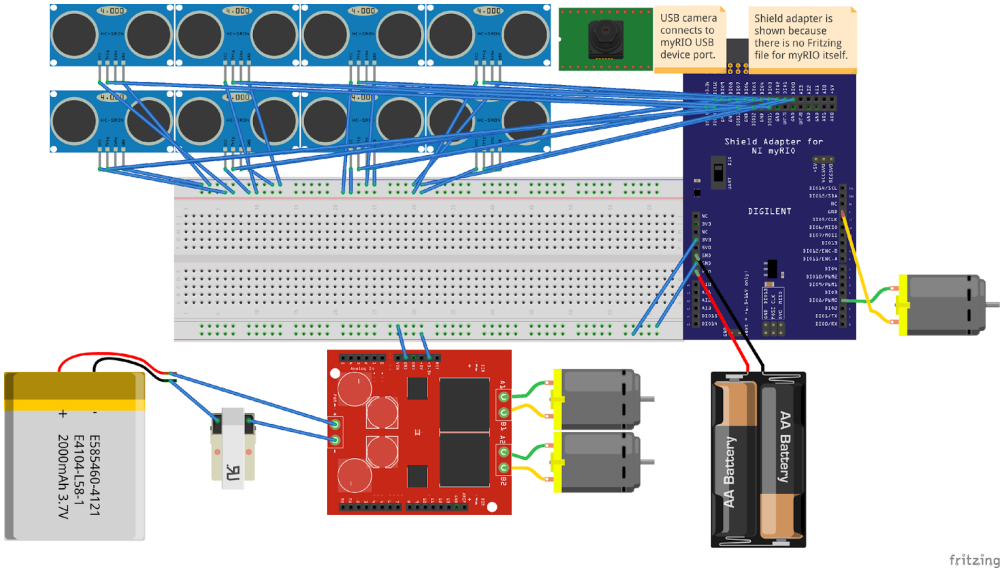
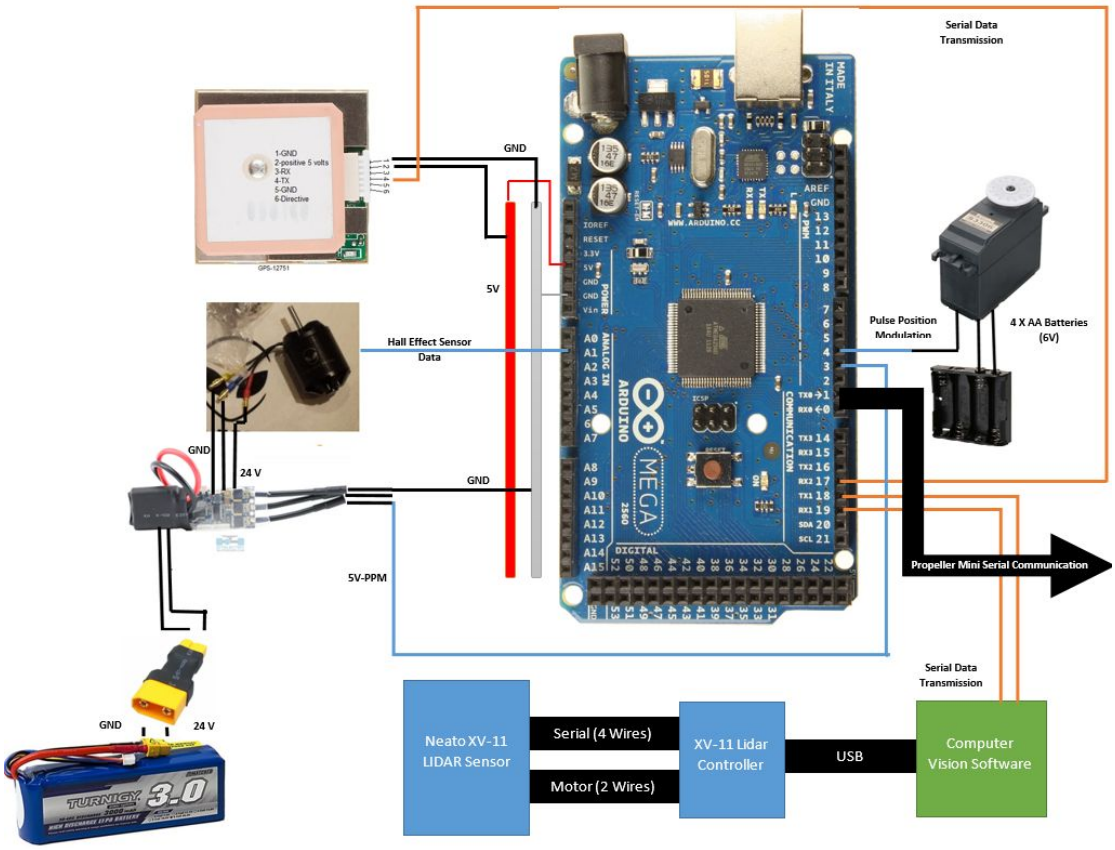


Figure 3: New wiring diagram. My portion is abstracted into the green box.


Serial Data Transmission
GND
5V
Hall Effect Sensor Data
Pulse Position Modulation
4 X AA Batteries (6V)
GND
24 V
GND
5V-PPM
Propeller Mini Serial Communication
Serial Data Transmission
GND
24 V
Neato XV-11 LIDAR Sensor
Serial (4 Wires)
Motor (2 Wires)
XV-11 Lidar Controller
USB
Computer Vision Software

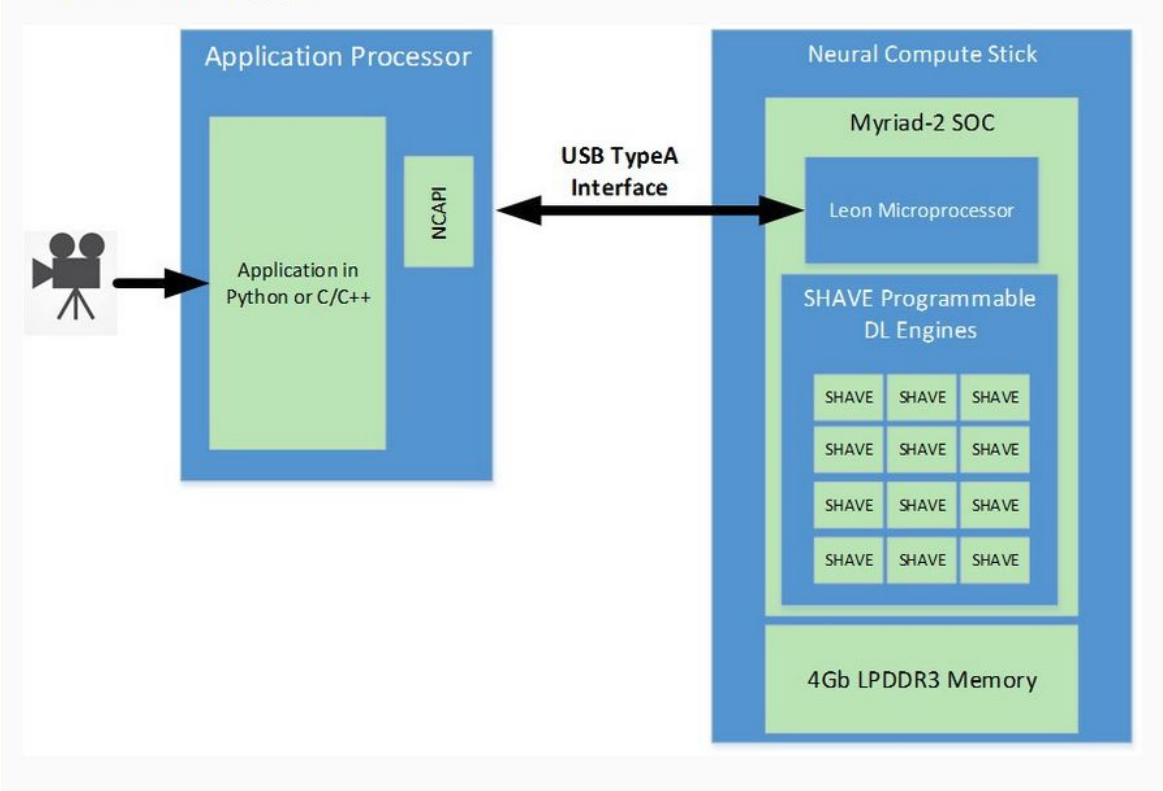
My portion of the project consisted of selecting an embedded computer, writing computer vision software, and optimizing it to run as fast as possible on the embedded computer. The intended use of this software is to have it provide, in combination with other sensors, the necessary data needed for a scaled car to autonomously and dynamically route a path and drive towards a given destination. For the embedded computer, I chose the ODROID-XU4. It has the following specifications [10]:

- Samsung Exynos5422 Cortex™-A15 2 GHz and Cortex™-A7 Octa core CPU
- Mali-T628 MP6(OpenGL ES 3.1/2.0/1.1 and OpenCL 1.2 Full profile)
- 2Gbyte LPDDR3 RAM PoP stacked
- eMMC5.0 HS400 Flash Storage
- 2 x USB 3.0 Host, 1 x USB 2.0 Host
- Gigabit Ethernet port
- HDMI 1.4a for display
- Size : 83 x 58 x 20 mm approx.(excluding cooler)
- Power: 5V/4A input
- Linux Kernel 4.14 LTS

Figure 4: A picture of the ODROID XU-4.

I chose the ODROID XU-4 because it is a relatively cheap (~$70) embedded computer that has the specs, in particular the parallel processing ability, that I needed for my computer vision software. Computer vision software, in particular software that can detect complex objects such as people and vehicles within a video, is notoriously computationally intensive. The computer has to do this complex processing within the time between the frames from the camera feed, which comes in at 30 frames per second. Images contain a lot of data, but the operations done on each piece are relatively simple. In this case, the processor speed, instruction set complexity, and precision of each piece of data doesn't matter as much as the sheer ability to do many very simple operations simultaneously. Therefore, multiple processors are good, but even better is a good Graphics Processing Unit (GPU). GPUs are made specifically to handle the numerous and simple operations common to image processing, but to effectively utilize the GPU in my programs I needed OpenGL and OpenCL. OpenGL is an API for handling graphics and for interacting with GPUs. OpenCL is a framework for writing programs that effectively utilizes different types of processors, including extra CPUs and the GPU. Luckily, both are supported by the ODROID. Later in the project, I would also attempt to use the new Movidius Neural Compute stick [11], an external Vision Processing Unit (VPU), as a hardware accelerator for an artificial neural net.

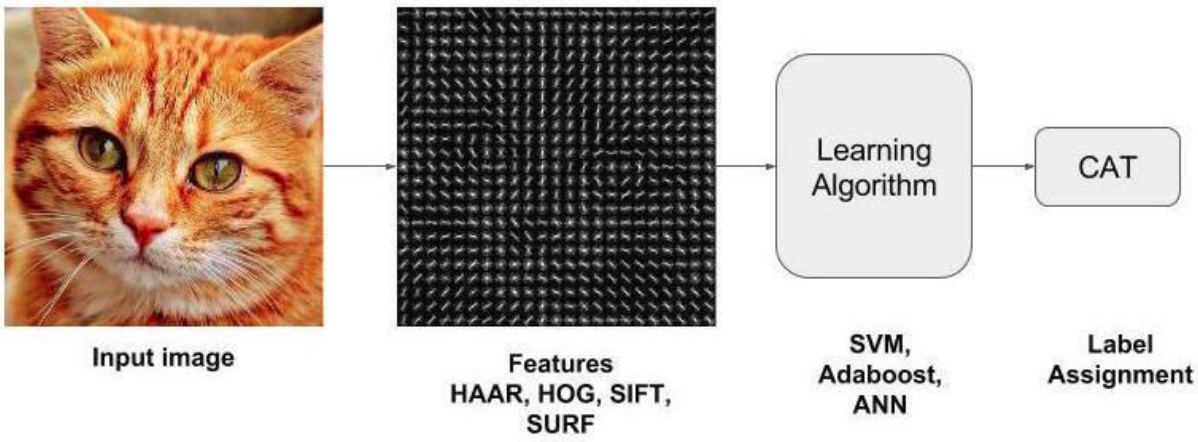
Figure 5: Architecture of the Movidius neural compute stick.



I first started with implementing the pedestrian detection feature because I figured at the time it was most important for an autonomous car. My opinion on the relative importance of this

feature would later change, as well as my implementation. My initial implementation used a Histogram of Oriented Gradients (HOG) and a Linear Support Vector Machine (Linear SVM) to detect people in an image and draw a bounding box around them. A Linear SVM is a supervised learning algorithm used for binary classification. It's trained by being given a set of examples marked as either belonging to the category it's meant to detect or not. In this case, the training data were HOGs. A HOG is a 3D (horizontal, vertical, direction of the gradient) array of values that encode the magnitude of change from one area of the image to another.

Figure 6: Visual representation of a Histogram of Oriented Gradients [12].



To detect an object, the HOG of a frame is computed and then sent to the Linear SVM to classify. But, there is another problem. The matching HOG could be anywhere in the picture at any scale. The cat could fill the picture, or be just a small part in the background. To do address this, multiple HOGs of the same frame are computed at different scales. It would be too computationally intensive to compute and compare HOGs at every possible scale though, so a discrete scaling factor is used. The scaling factor can be thought of as the number the size of an image is divided by to get a new image to compute the HOG of. A larger scaling factor means the image is cut more dramatically each time. This results in a smaller number of images to compute the HOG of and faster execution of the program. As the scaling factor approaches 1, the number of images created and HOGs computed increases and slows down the execution of the program.

Figure 7: Image pyramid of a picture using a scaling factor of 2 [12].

I experimented with multiple scaling factors until getting the best results with 1.1. The best results were still very poor. It would consistently box the brightest object in the room, but inconsistently box people. It didn't like me in particular, although it had an odd tendency to recognize my ear or hand turned sideways as a person. It also had obvious latency issues in the output feed, updating about once a second. Despite this, I added on non-maxima suppression, an algorithm to reduce redundant detections. The execution became even slower. In order to speed it up, I figured out that I didn't need to re-detect the objects every frame. I could track them, and re-detect after a period of time. Tracking is much less computationally intensive than re-detecting. There were many different tracking algorithms to choose from [13]; I used Kernelized Correlation Filters (KCF). I still don't quite understand how it works, but it was certainly fast on its own. The KCF algorithm could be executed on every frame with no perceivable slow-down, and it tracked objects well. It was still limited by the speed of re-detecting objects, despite that being done only every 30 frames. I would eventually end up throwing the HOG, Linear SVM, and non-maxima suppression based detection out and switch to a new implementation using artificial neural nets.

Before making that switch, I decided to implement lane detection. The principles behind this were relatively simple. A frame was first converted to both a grayscale and Hue Saturation Value (HSV) copy of the original. Everything that wasn't the color of lane lines (white and yellow) was then deleted from the respective images. The thresholds for these colors were manually found by testing stock footage over and over again until the end result had acceptable accuracy. The remaining white and yellow images are then recombined and passed to the Canny Edge detection algorithm. It detects edges, if that wasn't obvious. The remaining white and yellow edges then go through further selection criteria to see if they are within the region of interest (in front of the car), more or less straight, and with minimal gaps in between them. These remaining lines are then taken as the lane lines. Testing lane detection on its own went smoothly. My program now had two distinct and independent features: lane detection and object detection. However, simply running one after the other in the frame processing loop resulted in object detection preventing lane detection from executing as fast as it could. I then realized the reason why this was happening.

Early on I had decided to write the whole program in Python. The main reason I chose to do this was the fact that Python has many computer vision, AI, and quantitative finance and data analytics libraries (unrelated to this project). Most of the cutting edge and really interesting job solicitations I've seen want Python programming experience, which I didn't have a lot of prior to this project. I had also read that since the core OpenCV library is written in C with Python headers, the performance difference would be at maximum 4%. This turned out to be untrue, at least for what I was doing. A C++ and Python program might have similar execution times if executing one thread serially, but when trying to do multiple threads in parallel Python has some inherent problems. The main problem is the fact that Python is an interpreted, not a compiled, language with a Global Interpreter Lock, meaning every line is executed only after the preceding has been completely read and executed. That was exactly what I wanted to avoid with vision processing. Luckily, the OpenCL built into the python libraries recognized this and spread some of the computation over the other processors. However, it would still hit a big bump that slowed the whole program down when encountering a particularly intensive computation (i.e. object detection) within the camera frame processing loop. To get around this, I attempted to use Python's 'multiprocessing' library. I was able make multiple threads that executed concurrently, but I had trouble passing data between them using the library's Pipe() objects. It would give me vague error messages when trying to pass an image between two threads, and after spending a lot of time trying to solve the issue I gave up and tried another idea. The other idea was creating my own implementation of a periodic thread by having a function recursively call itself after the desired period. This failed in two different ways. It initially had a problem where it would create so many threads it would freeze the computer. After that was resolved, I found it had a similar problem with exchanging information between threads. I canned that idea and tried using global variables to share data between threads (using the multiprocessing library's implementation), but that didn't work either. Seeing that I was getting nowhere fast with multithreading, I switched back to improving object detection in hopes of reducing its execution time.

Figure 8: My planned multithreading implementation.

CPU 1	CPU 2	CPU 3	CPU 4
Get camera frame	Get LIDAR data		
		Detect pedestrians	
Get camera frame	Get LIDAR data		
Track pedestrians		Detect vehicles	
Get camera frame	Get LIDAR data		
Track pedestrians	Track vehicles	Detect signs	
Get camera frame	Get LIDAR data		
Track pedestrians	Track vehicles	Detect lanes	
Get camera frame	Get LIDAR data		Get ultrasonic
Track pedestrians	Track vehicles	Detect pedestrians	Route path
Get camera frame	Get LIDAR data		Get ultrasonic
Track pedestrians	Track vehicles	Detect vehicles	Route path
Get camera frame	Get LIDAR data		Get ultrasonic

The HOG and Linear SVM implementation was slow and not at all accurate, so I decided to attempt using an artificial neural net. I got it working with relative ease, and although it still slowed down the rest of the program, it had much higher accuracy in detecting pedestrians than the Linear SVM and could also detect vehicles. An artificial neural net works by taking in taking in chunks of data into its first layer of "neurons". Each of these neurons makes a connection that influences the state of all the neurons in the next layer, but to different degrees. The neurons of the second layer influence the third layer, and so on. A neural net is trained to make accurate predictions by adjusting the relative impact that a neuron has on the neurons in the next layer until it correctly classifies (to a desired accuracy) the data piped through it. There is usually a tradeoff between accuracy and execution time though. The particular neural net I chose to use for object detection was the ssd_mobilenet_v1_coco, a type of MobileNet. This type is meant for mobile devices and differentiated by its relative speed of execution.

Figure 9: A diagram showing how a neural net for detection/classification works [14].

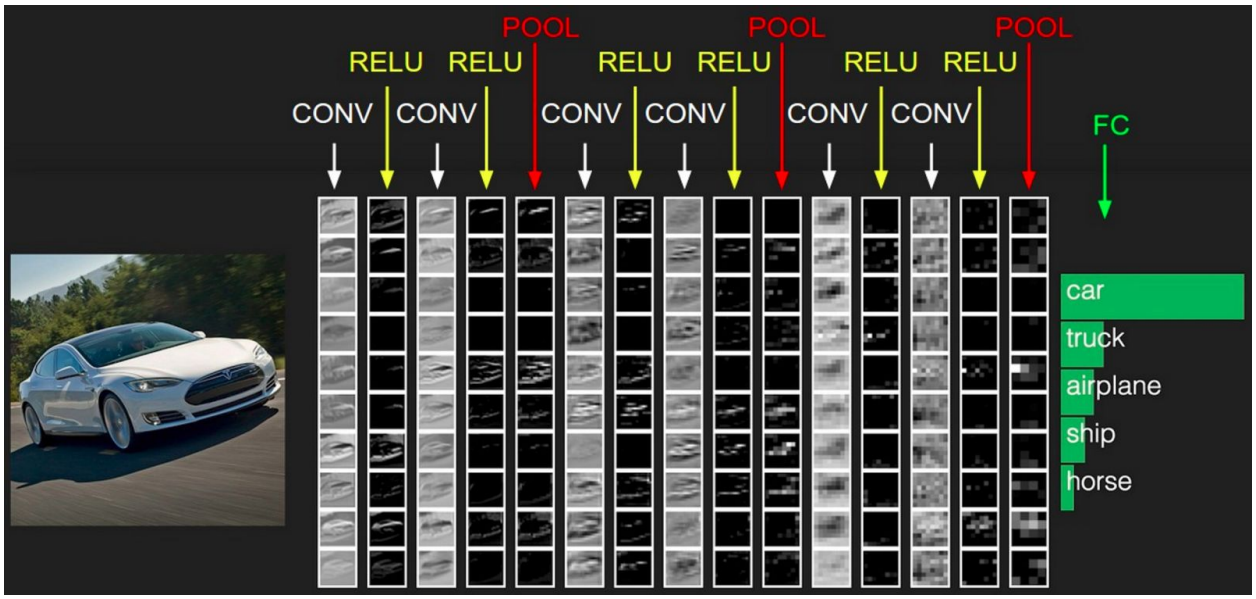
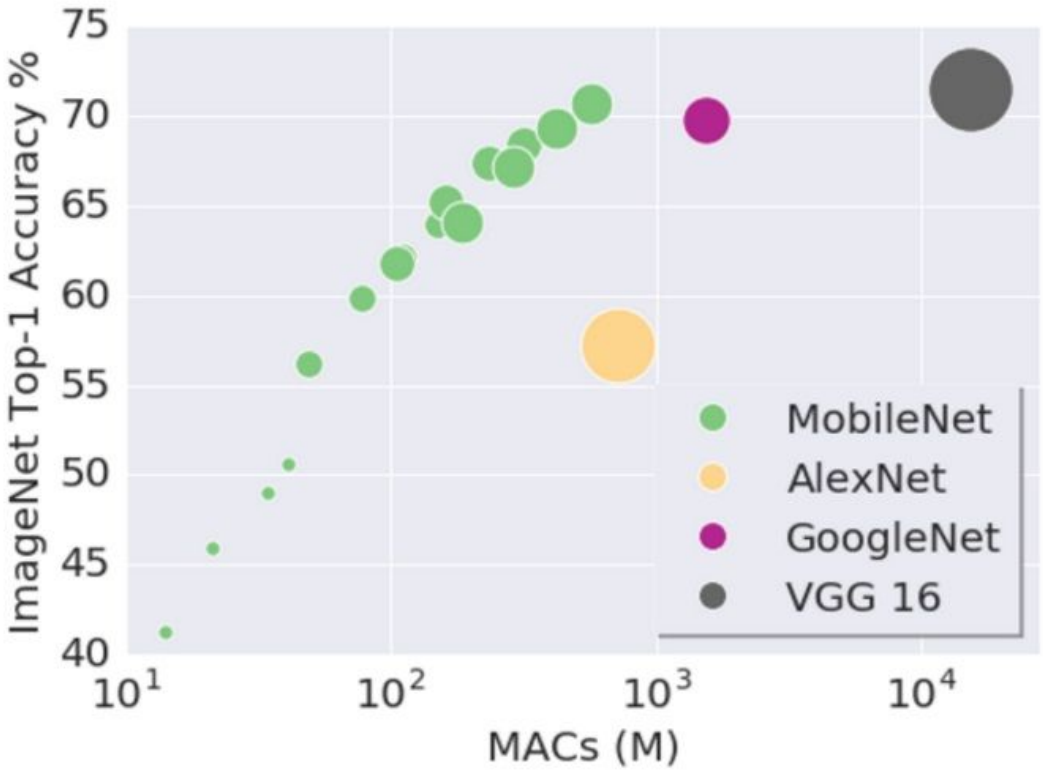


Figure 10: Graph of the accuracy of some pre-trained neural nets versus execution time [15].



Utilizing a neural net in the program meant I would have to setup tensorflow on the ODROID. No problem, I thought. I can just do 'sudo pip3 install tensorflow'...except for the fact that the ODROID's processors have an armhf architecture and not x64 like my laptop. Tensorflow officially supports only x64 architectures. So, I started on the long and difficult journey to compile tensorflow from source. To do that, I had to use a special compilation tool called bazel. 'sudo apt-get install bazel'? Nope. Bazel didn't support armhf architectures either, so I tried compiling it from source with openjdk. That gave an error that the java virtual machine would run out of memory during compilation, so I found and used the option to increase the maximum amount of memory available to the virtual machine. It still gave the "out of memory" error message, so I created an on-disk 4 GB swap file and formatted an 8 GB USB stick as an external swap partition. I also set the swappiness value of the operating system to the maximum of 100. I tried compiling again while watching the system resource utilization to see how much of the RAM and swap it was using. The memory utilization went up a teeny bit, then went back down as it failed. I double checked my java compilation options. The virtual machine had its memory utilization limit set at 3 GB, its max, yet it did not seem to make a difference. After spending over a day on the issue, it suddenly occurred to me that someone else probably had a pre-compiled binary publically available on the internet. So I Googled it, and immediately found and installed it with no issues...nearly.

Meanwhile on my laptop, my primary development environment, I was attempting to reduce object detection execution time with the Movidius neural compute stick. The library to utilize the neural compute stick had installed there without issue. It only depended on tensorflow, so it should have been installable now on the ODROID. However, upon running the installation script I was greeted with an error message "Incorrect version of tensorflow". I didn't have time to go down another rabbithole, so I quickly rewrote my software to not depend on or use the Movidius libraries. After solving some issues with camera drivers, I was able to get the program working on the ODROID.

Figure 11: The software architecture.

File name	autoSteer.py	laneDetection.py	HOG_ObjectDetection.py (obsolete)	objectTracking.py	NN_ObjectDetection.py
Description	Initializes camera and other objects. Contains the frame reading loop.	Detects traffic lanes.	Was used for detecting pedestrians. Replaced by NN_ObjectDetection.py	Used for tracking objects detected by the neural net.	Used for detecting pedestrians and vehicles.
Objects	camera			TrackedObject	detection_graph

Methods	__main__() 	detectLanes() region_of_interest() draw_lines()	detectObject() 	updateTraces() getTraces()	findObject() setupMovidius() filter_boxes() to_image_coords() draw_boxes() load_graph()
Calls on	setupMovidius() detectLanes() findObject() updateTraces() getTraces()	region_of_interest() draw_lines()		findObject()	load_graph() filter_boxes() to_image_coords() draw_boxes()

Figure 12: Lane and vehicle detection.

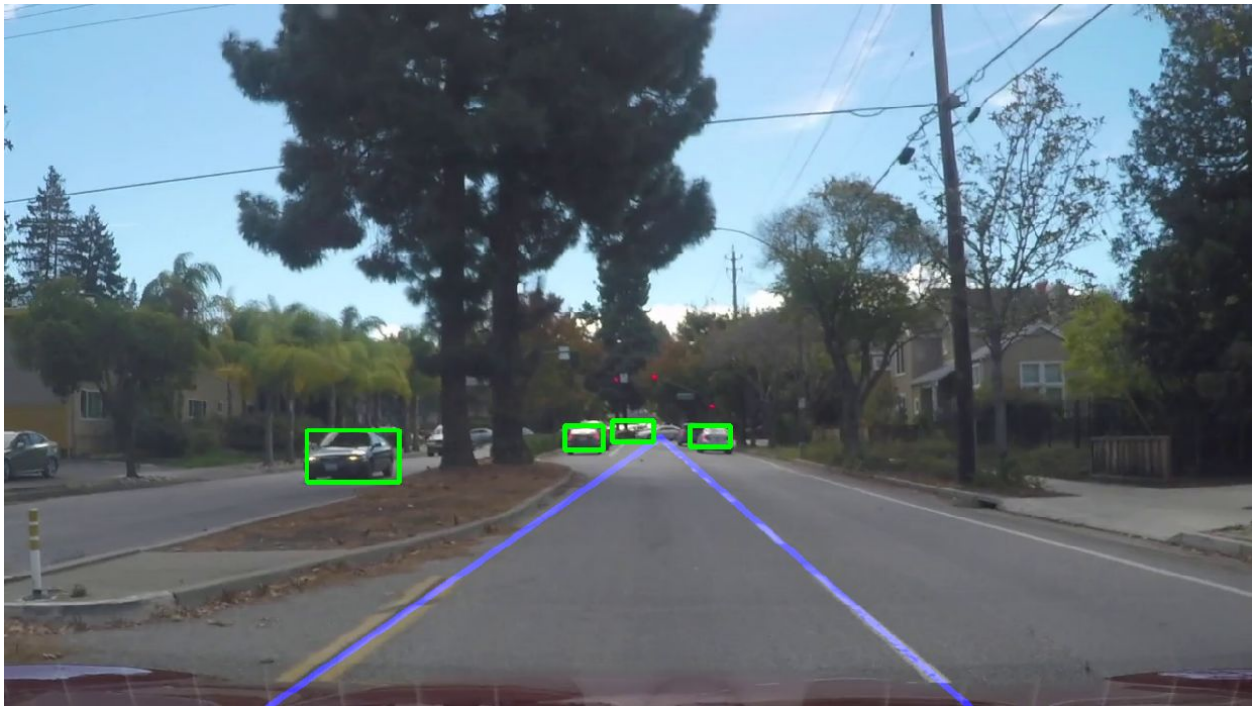
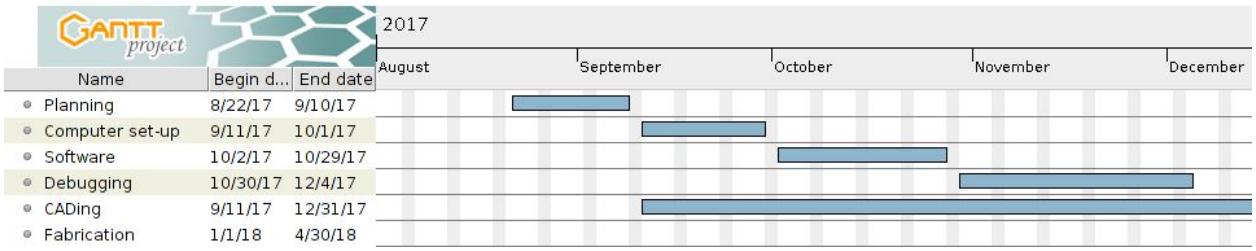


Figure 13: Pedestrian detection. Guess who?

Project Timeline

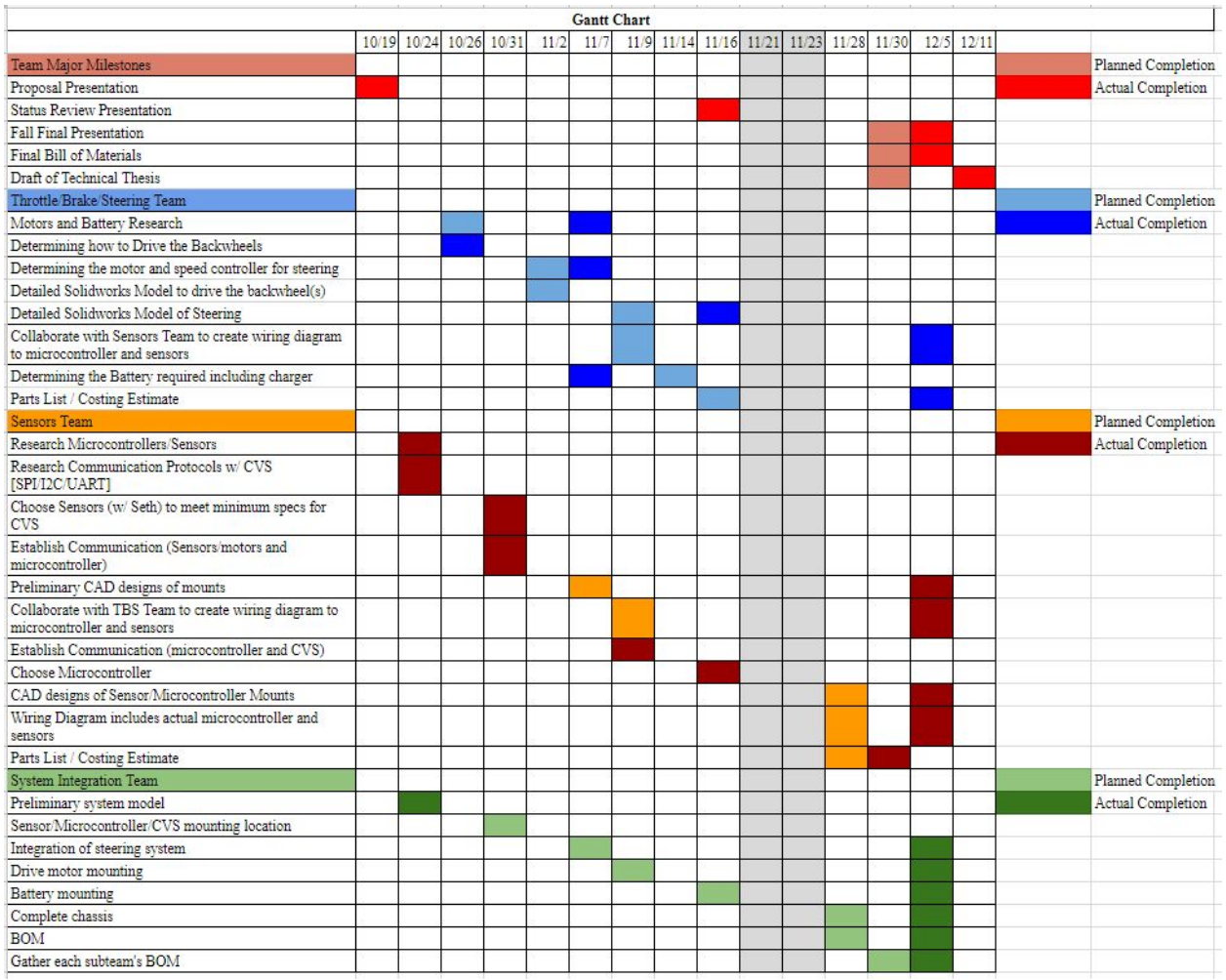
The timeline for the project changed dramatically throughout the semester as the scope of our project changed. In particular, the dead end negotiation with Perrone left a giant month-long hole from September 21st to October 16th where we did not work on our own design.

Figure 14: Gantt chart prior to Perrone offer (from proposal).



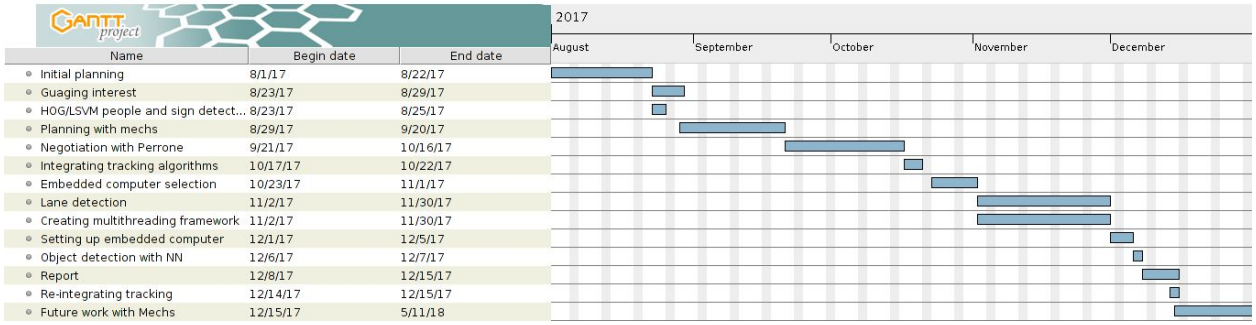
When we settled on the 1/5th scale, we established clear divisions of labor. I would do the embedded computer selection and work on the computer vision and path routing systems. The mechs would do the rest, divided into 3 subteams with two people each. There was no primary or secondary within the subteams.

Figure 15: Gantt chart the Mechs created for the rest of the Fall semester.



Early on in the semester, I had led and managed the mech team. After the Perrone incident, I mostly left them to their own devices while I worked independently on my part. Accordingly, I was dependent on no one else's schedule and did all the work serially when it was personally convenient or when the necessary resources became available.

Figure 16: The actual timeline.



Name	Begin date	End date
Initial planning	8/1/17	8/22/17
Guaging interest	8/23/17	8/29/17
HOG/LSVM people and sign detect...	8/23/17	8/25/17
Planning with mechs	8/29/17	9/20/17
Negotiation with Perrone	9/21/17	10/16/17
Integrating tracking algorithms	10/17/17	10/22/17
Embedded computer selection	10/23/17	11/1/17
Lane detection	11/2/17	11/30/17
Creating multithreading framework	11/2/17	11/30/17
Setting up embedded computer	12/1/17	12/5/17
Object detection with NN	12/6/17	12/7/17
Report	12/8/17	12/15/17
Re-integrating tracking	12/14/17	12/15/17
Future work with Mechs	12/15/17	5/11/18

There were no distinct software writing and debugging periods like in the original gantt. I debugged features as I added them. I also did the computer set-up much later than intended because I changed which computer I was going to use, and ordered the final one late. Accordingly, the timeline to the end is more condensed.

Test Plan

Like our schedule, our test plan changed as the scope of our project changed. The first test plan was made when we thought we would have access to a golf cart and Lincoln Navigator provided by Perrone Robotics. This plan became obsolete after Perrone went back on their offer, as we no longer had a golf cart / car available to test with this semester. A modified version of this will be applicable once the Mechs build their car.

Figure 17: The first test plan.

Do the computer vision libraries and included demos work?		Does GPS module give accurate position while stationary?		Do the ultrasonic and radar sensors give accurate distances when stationary?		Do the brakes actuate the right amount and at the right speed?		Is the "gas" actuated properly to accelerate and maintain speed?		Does the mechanical actuation adjust the steering wheel to the right angle?		Can the map software interface with ours?	
Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No
Does my detection code work? Is it accurate and fast enough?		Does it give accurate position moving with decent latency?		Do they give accurate distances when moving?		Do they stop the golf cart at the right time and speed?		Does it work in other conditions (i.e. slopes, slick roads, gravel)?		Does it do it at the right rate in order to follow the given path?		Does it give accurate and complete data?	
Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No
Does the golf car correctly route a path around obstacles and correctly steer, brake, and apply gas to follow that route? Does it keep on the road?													
Yes							No						
Apply to Lincoln Navigator													

Figure 18: The test plan I actually followed.

Test example programs written by others for algorithms/libraries relevant to autonomous cars. Can I get them to work on the embedded computer?	
Yes	No
Learn the underlying principles and add them to my program.	Find an alternative algorithm/library.
Does my implementation of the algorithm/library run fast enough?	
Yes	No
Keep it in my program, barring incompatibility with a more important feature or finding an even better implementation.	Look for hardware and software accelerants. If none are available, find an alternative algorithm/library.

This test plan above was applied to each feature. Much of the testing had to be done serially since certain features depended on others. Object tracking depended on object detection. Object detection, being the most computationally intensive part, was most affected by my different attempts to implement multiprocessing. Lane detection was separate from the other features, but still affected by my multiprocessing implementations. All the features were dependent on the software libraries I could put on the embedded computer. I ended up replacing my Histogram of Oriented Gradients and Linear SVM implementation of object detection with an implementation based on neural nets, because the previous was not accurate enough. This broke my implementation of object tracking, which was dependent on the output of my previous implementation of object detection. I wasn't able to fix in time for the capstone fair (hence the periodic lag), but have got it working now.

Final Results

The success criteria defined in the original proposal were:

D-level deliverables:

- Autonomy software working on a non-embedded computer that can sometimes detect pedestrians.

C-level deliverables:

- Partially working pedestrian detection software running on an embedded computer.

B-level deliverables:

- Fully working pedestrian detection software running on an embedded computer.

A-level deliverables:

- Fully working pedestrian detection software running on an embedded computer that can control a car and avoid obstacles.

I would put the final results somewhere between the B and A level. I have fully working pedestrian detection software running on an embedded computer as per the B-level deliverables, but it cannot yet control a car. However, I did implement several other features besides pedestrian detection, such as car and lane detection. I will also continue to work with the Mechs next semester to meet the final criteria of controlling a car, and by the time that is done it will have additional features surpassing my original expectations.

Costs

The "car" the Mechs are building is mainly an academic project. It would have little commercial value unless sold as a platform to learn about autonomous cars. Therefore I went ahead and estimated the costs of building bolt-on autonomy kits for non-autonomous cars. Many of the parts did not have accessible bulk pricing information. The ones that did had bulk prices for quantities of 1,000. The cost to make 1 kit came out at around $500. The scaled cost to make a kit is roughly $400. See Appendix B for a more detailed breakdown of component costs. Including development costs, marketing, marketing, distribution, and profit margins I would expect it to cost <$1,000 to the consumer. This is a marketable price for car owners that do not want to trade-in or buy a new car with built-in sensors. Of note is that over half the component costs come from the eMMC memory and LIDAR sensor. These are higher end options which could potentially be substituted by flash memory and radar respectively, but I would rather use higher end components than cheaper for this application.

Future Work

I will continue to work with the six mechanical majors next semester, integrating their sensor data into my program and using it to route paths. The skills I've developed during this project are also directly applicable to jobs I'm interested in. I've already applied to computer vision related engineering positions at Tesla, and plan to apply to similar positions at other companies as well. I was also surprised by the lack of patent coverage for autonomous car kits and robotic delivery vehicles. I'm considering pursuing one or both as entrepreneurial ventures given I've already done a fair portion of the work required by writing this autonomy software.

This work could be expanded on by future students by doing what I originally wanted to: making a full-sized electric car and applying the autonomy software to it. My advice would be this: don't settle for something else simply because it's easier and safer. In the future you won't regret trying and failing, you'll regret never taking the chance.

References

[1] "44 Corporations Working On Autonomous Vehicles", CB Insights Research, 2017. [Online]. Available: https://www.cbinsights.com/research/autonomous-driverless-vehicles-corporations-list/. [Accessed: 13- Dec- 2017].

[2] B. Schmitt and E. Niedermeyer, "Auto Industry 101. Today: How big?", DailyKanban, 2015. [Online]. Available: https://dailykanban.com/2015/03/auto-industry-101-today-big/. [Accessed: 13- Dec- 2017].

[3] "Statistics & Facts on the Global Automotive Industry", www.statista.com, 2017. [Online]. Available: https://www.statista.com/topics/1487/automotive-industry/. [Accessed: 13- Dec- 2017].

[4] "Federal Motor Vehicle Safety Standards and Regulations", *Icsw.nhtsa.gov*. [Online]. Available: https://icsw.nhtsa.gov/cars/rules/import/FMVSS/. [Accessed: 13- Dec- 2017].

[5] "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", *Standards.ieee.org*, 2012. [Online]. Available: http://standards.ieee.org/getieee802/download/802.11-2012.pdf. [Accessed: 13- Dec- 2017].

[6] "How Safe Are Self-Driving Cars?", *Safer America*, 2016. [Online]. Available: http://safer-america.com/safe-self-driving-cars/. [Accessed: 13- Dec- 2017].

[7] Toyota Motor Corp, "Intelligent Navigation System", US8855847B2, 2017.

[8] Amazon Technologies Inc, "Transportation network utilizing autonomous vehicles for transporting items", US9786187B1, 2015.

[9] R. Abhyanker, "Driverless vehicle commerce network and community", US9459622B2, 2016.

[10] “ODROID XU-4 User’s Manual”, *HardKernel*, 2017. [Online]. Available: https://magazine.odroid.com/wp-content/uploads/odroid-xu4-user-manual.pdf [Accessed: 14- Dec- 2017].

[11] “Movidius Neural Compute SDK”, 2017. [Online]. Available: https://movidius.github.io/ncsdk/. [Accessed: 14- Dec- 2017].

[12] S. Mallick, “Image Recognition and Object Detection: Part 1”, *Learn OpenCV*, 2016. [Online]. Available: https://www.learnopencv.com/image-recognition-and-object-detection-part1/. [Accessed: 14- Dec- 2017].

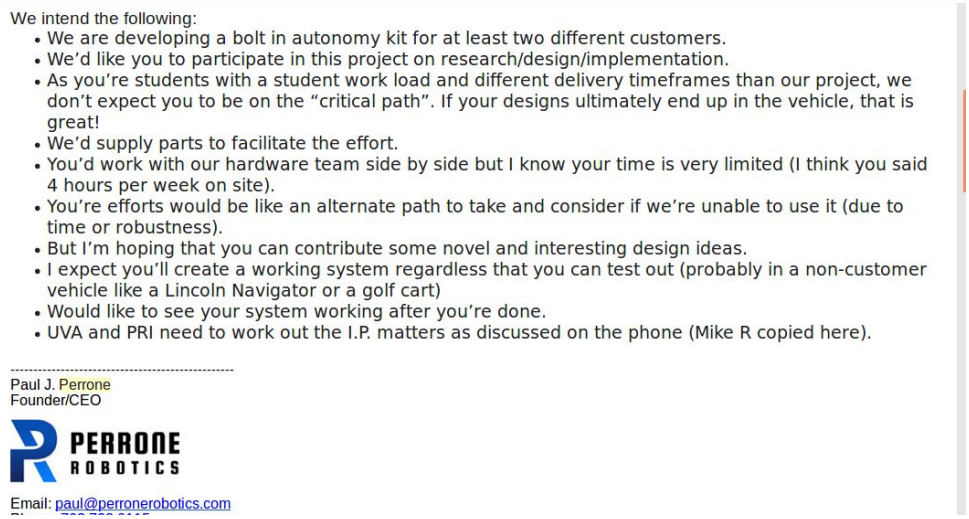
[13] S. Mallick, “Object Tracking using OpenCV (C++/Python)”, *Learn OpenCV*, 2017. [Online]. Available: https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/. [Accessed: 14- Dec- 2017].

[14] P. Kontschieder, “Mapillary Now Able to Recognize and Label Objects in the Wild”, *The Mapillary Blog*, 2017. [Online]. Available: https://blog.mapillary.com/update/2016/09/27/semantic-segmentation-object-recognition.html. [Accessed: 15- Dec- 2017].

[15] R. Meertens, “Google Released MobileNets: Efficient Pre-Trained Tensorflow Computer Vision Models”, *InfoQ,* 2017. [Online]. Available: https://www.infoq.com/news/2017/06/google-mobilenets-tensorflow. [Accessed: 15- Dec- 2017].

Appendix

Appendix A: Perrone's offer.



We intend the following:

- We are developing a bolt in autonomy kit for at least two different customers.
- We'd like you to participate in this project on research/design/implementation.
- As you're students with a student work load and different delivery timeframes than our project, we don't expect you to be on the "critical path". If your designs ultimately end up in the vehicle, that is great!
- We'd supply parts to facilitate the effort.
- You'd work with our hardware team side by side but I know your time is very limited (I think you said 4 hours per week on site).
- You're efforts would be like an alternate path to take and consider if we're unable to use it (due to time or robustness).
- But I'm hoping that you can contribute some novel and interesting design ideas.
- I expect you'll create a working system regardless that you can test out (probably in a non-customer vehicle like a Lincoln Navigator or a golf cart)
- Would like to see your system working after you're done.
- UVA and PRI need to work out the I.P. matters as discussed on the phone (Mike R copied here).

Paul J. Perrone
Founder/CEO

PERRONE ROBOTICS

Email: paul@perronerobotics.com

Appendix B: Detailed cost estimate of building a bolt-on kit.

Part Name	Description	Price per: 1 ($)	Scaling factor	Price per: 1000 ($)	Price of 1000 ($)
ODROID XU4	Embedded computer	76.95	1000	61.95	61950
5V/4A US type	Power supply	6.95	1000	6.95	6950
64 GB eMMC + adapter	Memory	66.45	1000	66.45	66450
WiFi module 3	Internet access	9.95	1000	6.95	6950
USB GPS module	GPS	26.95	1000	21.95	21950
USB-Cam 720p	Camera	19.95	2000	19.95	39900
HC-SR04	Short range distance	2.5	6000	2.26	13560
RPLidar A1M8	360 deg long range distance	199	1000	179	179000
Bolt-on frame	Attaches sensors to car	?	1000	?	?
					396710
R&D cost	without frame	**441.15**	Cost at scale	without frame	**396.71**