Design, Analysis, and Manufacturing of an Autonomous Luggage Transportation System for the Orlando-Melbourne International Airport

An Interim Report
Presented to
The Advisory Committee and Project Sponsors
Senior Design Course of Fall-2015 & Spring 2016

Florida Institute of Technology

In Partial Fulfillment
of the Requirements for the Course
MAE 4194: Mechanical Engineering Design

by
Autonomous Luggage Transportation System
ALTS 2016

Alexander Eierle, Jeffrey Hagans, Bradley Jones, Paul Kepinski, Haoyan Kuang, Max Le, Mo Mohamed, Eric Voigtlander

April 8th 2016

Accepted by:
Dr. Beshoy Morkos, Advisory Committee Chair
Dr. Mary Helen McCay, Advisor
Elisabeth Kames, Graduate Student Advisor

ABSTRACT

The team has been tasked with the design and development of an Autonomous Luggage Transportation System (ALTS) for the use at the Orlando-Melbourne International Airport (MLB) and their luggage transportation to and from airplanes. The ALTS will be implemented in place of the existing luggage transportation system at MLB. The existing system of the airport is a fuel powered vehicle that is fully operated by an airport employee. The ALTS will be completely powered by renewable/sustainable energy sources, and it also will operate and function fully autonomously.

The purpose behind this project is to implement the ALTS into an airport setting to make it possible for the airport to have a luggage transportation system that is more efficient, less costly, and more eco-friendly. These qualities have a very high potential to save the airport a large sum of money over time.

The ALTS has a fairly large list of requirements to meet in order to operate properly and safely. These requirements come from an array of sources: there are rules and requirements given by the Orlando-Melbourne International Airport regarding the tarmac and their facility that must be met and abided by; specific airline companies have rules and regulations regarding vehicles operating around aircraft; there also exist many regulations given by OSHA and FAA that must be met for the ALTS to be legal and safe.

The ALTS vehicle base is a Club Car Carryall 6, long bed utility golf cart. This model cart was purchased for $3,850. This cart has an electrical motor and is battery powered, meeting the renewable energy requirement. This cart has gone through several modifications in order to become the ALTS; these include modifications to the steering, braking, and throttle systems, as

well as general modifications to the bed of the cart. These modifications together, along with an electronic control system, will make the ALTS a success.

The modifications for braking, steering, and throttle will be drive by wire modifications. Drive by wire is a method of autonomy that consists of electrical signals commanding the mechanical control system, therefore driving the vehicle autonomously.

The brain of the ALTS is a computer board called the Jetson TK1, and will be the computer that tells the mechanical control system what to do and how to drive through this autonomously. To help the Jetson TK1 do its job, multiple sub-devices are required, such as: ultrasonic sensors, a stereo camera, and a differential GPS. These are all plugged in directly to the Jetson TK1 and serve as our source of navigation for the ALTS.

Since the ALTS must be powered by renewable/sustainable energy, research was completed with regards to these type of energy sources. The team looked for sources of renewable energy that could serve as a supplementary energy source. Although the ALTS vehicle body is electrically powered and is strong enough to function without any other energy, supplemental energy can be implemented to increase time between charges and help the battery life last longer. After several energy sources were explored, solar energy is going to be the best option for the ALTS.

The team has completed most research required to develop the ALTS and a large majority of the design process is complete.

ACKNOWLEDGMENTS

The team would like to extend thanks and appreciation to the following people for their help and contributions to the project. Without their help, this project would not be possible.

The 2016 ALTS Advisory Committee

Dr. Beshoy Morkos

Dr. Mary Helen McCay

Elisabeth Kames

Sponsors

Orlando - Melbourne International Airport

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

Chapter 1: INTRODUCTION

1.1 Problem Statement

Our objective is the design and development of a renewable/sustainable unmanned baggage buggy system for use within the Orlando-Melbourne International Airport. This buggy system will be capable of supporting baggage acquisition, travel, and drop off to its necessary location. Moreover, the challenge in developing this system will be the sustainable energy aspect. This buggy will perform in a "green" manner, through the use of renewable energy.

1.2 Motivation

The motivation behind the development of the ALTS is the benefits it can offer to not only to the Orlando-Melbourne International Airport, but to the environment as well. The ALTS if implemented will make the luggage transportation process at the airport much more time efficient. The ALTS will maximize its travel time on the airport tarmac because it will not need an operator to continue its job. This will decrease chances of lost luggage, and shorten the total loading time leading to increased satisfaction of customers, while saving money on operation costs. The money that the airport can save on fuel and maintenance of luggage transportation vehicles is very large because of the renewable energy of the ALTS compared to the currently operating vehicles utilizing diesel fuel. While the renewable energy of the ALTS will help save money, it also helps keep the earth clean because of decreased vehicle emissions.

1.3 Requirements

The ALTS has to comply with all requirements for vehicles within the airport terminal, these requirements were obtained from the Federal Aviation Administration (FAA), Occupational Safety and Hazard Administration (OSHA), and Orlando-Melbourne International

Airport rules and requirements. A full list of our requirements will be included in Appendix M. Some of the major requirements are elaborated on below.

- The ALTS must be fully autonomous.
- The ALTS must function and perform its intended task fully with minimal human input. In order to achieve this, it will have various proximity sensors, such as ultrasonic sensors and a stereo camera, in so it can sense the surroundings and determine the ideal path. A GPS will also be included so the ALTS can determine its current and intended position. All of these systems will synergies together and allow the ALTS to be autonomous.
- The ALTS must be powered using sustainable or "green" energy.
- The ALTS must also be powered only using sustainable or "green" energy such as solar or electrical energy. It will be mainly powered by electrical energy from rechargeable batteries, but it will also include solar panels to supplement the battery power and reduce the vehicle's load on the batteries.
- The ALTS must be able to carry ten bags at once.
- The ALTS must be able to transport at least 10 bags at one time. Ten bags would weigh no more than 600 lbs. The vehicle that was chosen as the base of the ALTS has a maximum bed capacity of 1000 lbs, which is more than enough to hold 10 bags.
- The ALTS must not interfere with the current frequencies at the airport.
- The ALTS must not interfere with any frequencies that the airport uses. Possible frequencies that could interfere with the airport's frequencies have been researched. None of our devices one the ALTS will conflict with the airport's frequencies. This research is outlined more in depth in section 2.1.4.

1.4 Global, Social, and Contemporary Impact

The development and implementation of the ALTS at the Orlando-Melbourne International Airport, and even other airports, in time, will have large impacts on the earth, and society. The largest global impact that the ALTS would be responsible for is its large amounts of fuel savings because of its strict renewable energy usage. The ALTS can help preserve the environment and Earth's natural state. The main social impact would be the satisfaction of the airport customers. The ALTS will reduce wait times and lost luggage to better benefit the customers. The other social impact of the ALTS is the ease of operation for the airport luggage-handling employees. This is also important in the efficiency of the luggage transportation process about the airport. The contemporary impact of the ALTS is that it is fully autonomous and renewable energy powered. This is a large step in the world of new technology and automation. No airport luggage transporter that has ever been built has ever been completely autonomous, making the ALTS a very contemporary piece of airport equipment. This could possibly lead to more and more systems similar to it.

Chapter 2: BACKGROUND

2.1 Literature Review

2.1.1 FAA

Most of the rules from the Federal Aviation Administration (FAA) pertain to aircraft and personnel, but there are some rules that are for vehicles on the airport tarmac. The FAA requires that all vehicles on the tarmac have a band of high gloss paint or white reflective tape around the perimeter of the vehicle; it must also include a flashing yellow light on the uppermost frame of the vehicle. The exact phrasing of these rules will be in Appendix K.

2.1.2 OSHA

The ALTS will be interacting with ground personnel throughout the airport so it is important to know what kind of rules the system must obey. The Occupational Safety and Health Administration (OSHA) has its own set of rules that must be followed in order for the ALTS to interact with people. Some of the most important rules that apply to the ALTS are that it must have a working audible warning device, at least two working head and tail lights along with brake lights, and it must have rubber fenders. All of these rules are important because they impact the design of the ALTS. The rest of the OSHA rules are listed in the appendix.

2.1.3 Orlando-Melbourne International Airport

The Orlando-Melbourne International Airport also has its own rules that the ALTS would have to follow. These rules and regulations mostly specify how the ALTS should maneuver about the airport. A few important rules are that taxiing aircraft and emergency vehicles always have the right of way, no vehicle can pass between an aircraft and its parking stand, no vehicle

can exceed a speed of 15 mph within the terminal ramp, and vehicles at night must have adequate headlights and taillights. A complete list of the airports rules and regulations is included in Appendix L.

2.1.4 Federal Communications Commissions (FCC) Regulation

According to the FCC, the signals within the airport for general and specific sensors cannot surpass the following maximum values.

Table 1: FCC Airport Maximum Allowable Frequency

Signal	Operating Frequency
Air traffic control	118-121.95 MHz
Distress/Military	121.5/ 243 MHz
Air to air	122 MHz

Table 2: General Sensor Frequency

Sensor	Operating Frequency
GPS (satellite)	L1: 1575.42 MHz L2: 1227.60 MHz
Ultrasonic sensors (acoustic waves)	40 kHz
Radars	300 MHz to 15 GHz

Table 1 and Table 2 show the allowable frequency ranges for different sensors. These maximum frequencies are extremely important because interferences can occur. It can be seen that the GPS and radars utilize very high frequency due to need of triangulation. The ultrasonic sensor's frequency range is significant lower than the other two because it uses acoustic waves to navigate.

2.1.5 NASA Study of GPS

In a study from NASA in October 2014, the effects of GPS on aircraft are investigated. Specifically, the study focuses on mobile phone interference with aircraft GPS navigation systems. See Appendix I for more information regarding this study.

Figure 19 shows a loss of information about 5dB attenuation. However, the maximum emission was only 5 dB above the composite maximum emission from eleven non-intentional transmitters previously reported. Therefore, according to NASA, this is considered acceptable in an airport environment.

2.2 Current State of the Art

2.2.1 Google Car Case Studies

During the process of researching about existing autonomous vehicles, the Google Car comes up as an ideal model to study. There are many different sensors on the Google Car; however, only the ones responsible for autonomous navigation are researched.

Figure 1: Google Car[9]

In terms of sensors, the Google car is equipped with ultrasonic sensors on the rear wheels to avoid obstacles. Meanwhile, the GPS and a form of high definition visual sensor are

responsible for navigating. One of the main features on the Google car is the LIDAR sensor, which can create 3D images for the car to bypass obstacles. In terms of software capabilities, the Google car relies on hard-coded data (such as stopping at red lights) and learned data (base on previous experiences).

In terms of sensors, the Google car is equipped with ultrasonic sensors on the rear wheels to avoid obstacles. Meanwhile, the GPS and a form of high definition visual sensor are responsible for navigating. One of the main features on the Google car is the LIDAR sensor, which can create 3D images for the car to bypass obstacles. In terms of software capabilities, the Google car relies on hard-coded data (such as stopping at red lights) and learned data (base on previous experiences).

Airport Background Research

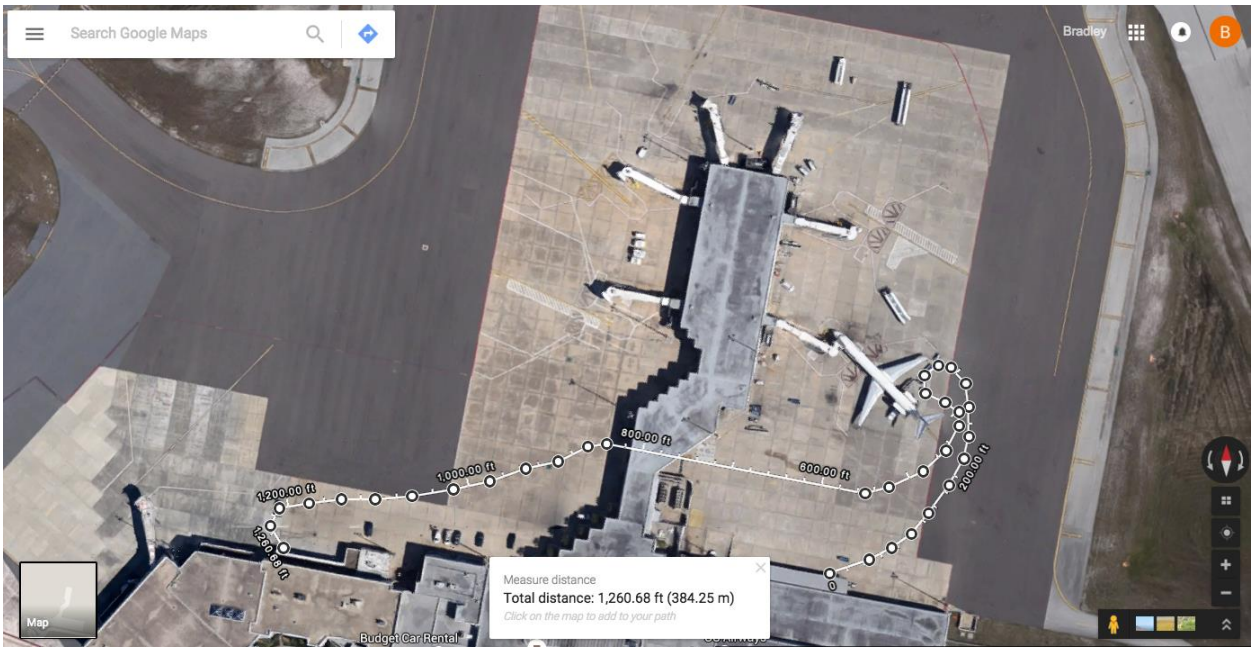


Figure 2: MLB Airport Proposed Path

2.2.2 Bag Estimate for Delta

The number of bags is one of the main requirements in this project. If the number of checked bags is known, then a better estimation of the number of bags that the vehicle must be able to carry in order to improve efficiency can be achieved. Therefore, a good estimation is needed to push the project forward. Based on Orlando-Melbourne International Airport Flight Schedule, the most common flight is operated by Delta Airline. Specifically, flight Delta 793 from Melbourne to Atlanta is the most frequent one. This particular model is a McDonnel Douglass 90 with a capacity of 160 passengers. Below is an existing record of Delta Airline's revenue as well as their baggage policy.

Table 3: Delta Airline Bag Policy

Between	Delta One™, First, Business	Main Cabin	
		First Checked Bag	Second Checked Bag
Travel within the **United States and Canada** (If traveling **to/from Key West**, see note below)	Free	25 USD/CAD[1]*	35 USD/CAD[1]*

Table 4: Estimation of Baggage Fees[10]

Rank	Airline	2014 Baggage Fees
1	Delta	$862,909,000
2	United	$651,857,000
3	American	$574,430,000
4	US Airways	$511,281,000
5	Spirit	$241,867,000

For this particular MD-90 plane, there are 31 premier seats and 129 normal seats. In order to make the calculations simple, it is considered that premier class will check 2 bags per passenger while economy class will check 1 bag per passenger.

As of March 2015, the following information is gathered regarding….. [10]:

- 5400 flights per day at all airports. This includes both departure and arrival flight.
- Delta makes revenue of around $862 million on checked bags alone.

Based on a September schedule of Orlando-Melbourne International Airport, which is shown in Appendix H, there are around 212 flights per month at Melbourne. This includes both departure and arrival flight. Denoting the parameters as follow:

$$F_{ALL} = 5400\ \frac{flights}{day}$$

Equation 1: Number of Delta Flights per Day at all Airports

$$F_{MLB} = 212\ \frac{flights}{month} \approx 7\ \frac{flights}{day}$$

Equation 2: Number of Delta flight per Day at MLB

$$Cost_{ALL} = \$862{,}909{,}000\ /year = \ \$2{,}364{,}134\ /day$$

Equation 3: Checked bag fees at all Airport per year from Delta

Assume a linear relationship between the number of flights and the cost of checked bags, the following ratio can be considered:

$$r = \frac{F_{MLB}}{F_{ALL}} = \frac{Cost_{MLB}}{Cost_{ALL}}$$

Equation 4: Ratio Between Number of Flights and Cost of Checked Bags

Equation 4 gives for all Delta flights leaving and arriving at Orlando-Melbourne International Airport per day. As indicated above, MLB has around 7 Delta flights leaving and arriving on a daily basis.

$$Cost_{Delta\ per\ day} = \frac{\$3064}{7\ flights}\ per\ day$$

Equation 5: Cost of Checked Bags for Delta Airline per day at MLB

Therefore, the cost of checked bags per Delta flight at Orlando-Melbourne International Airport is calculated to be around $437 per flight per day. Recall from the Delta bag policy, this $437 in revenue per flight per day is only paid by the economy classes. As a result, this implies around 18 checked bags per flight on a daily basis. For the worst case scenario, if each of the 31 premier passengers checks in 2 bags, then the total of checked bags per Delta flight per day is 80 checked bags. This method is realistic because it is based upon official information from Delta Airlines.

Chapter 3: RECOMMENDED SOLUTION

3.1 Solution Principles

The overall solution is to take an existing golf cart vehicle base and make it fully autonomous by implementing a main computer control board with a visual camera sensor, analog distance sensor equipment, and a GPS module for navigation. The vehicle abides by all FAA and OSHA regulations and be programmed with all safety algorithms in check. The proposed solution will cost a total of $16,887 using a new golf cart base. The ALTS will begin saving the airport money after 23 months.

3.2 Analysis

3.2.1 Return on Investment

One of the main requirements for this project is the return on investment for the sponsor, Orlando-Melbourne International Airport. Based on a job listing from MLB, the hourly wage for ground worker varies between $10.00 and $16.00. The airport has two baggage areas and considering the following scenario:

2 workers to load + 2 workers to unload + 1 driver = 5 workers in total per baggage area.

Assuming the airport does not spend a lot on workers; in other words, each worker is paid $10/hour. Additionally, a typical day is assumed to go from 6 A.M. until 12 A.M., resulting in 18 hours of work. Then, the following can be calculated:

$$Wage\ for\ all\ workers\ at\ 1\ baggage\ area\ per\ week = \left(\frac{\$10}{1\ hr}\right) * 5\ workers * 126\ working\ hours = \$6300$$

Equation 6: Workers' wage at 1 baggage area per week

$$Wage\ for\ all\ workers\ at\ 2\ baggage\ areas\ per\ week = 2 * \$6300 = \$12{,}600$$

Equation 7: Workers' wage at 2 baggage areas per week

$$ALTS's\ price\ in\ the\ 1st\ week\ at\ 2\ baggage\ areas = \$20{,}000 + (4 * \$10 * 126\ working\ hours * 2)$$
$$= \$30{,}800$$

Equation 8: ALTS's price at 2 baggage areas per week

Initially, it looks as if the ALTS is more expensive than the existing system at the airport. However, it should be noted that the workers' wage are paid every hour of the working day; while the ALTS only needs the initial investment of a little over $8,800 one time.

This can be further visualized by assuming that the costs to maintain the ALTS are as follow:

- Cost for maintenance (tire/battery replacement): $0.50/day
- Average charge cost: $1.83/day

This gives a total cost per day of around $2.33 for the ALTS. Unlike the worker's wage, the ALTS only need this as its maintenance fee every day. The airport, based on research, spends $20.00 per day on fuel cost and maintenance cost of $5.48/day. This gives a total cost to keep the current system around $25.48.

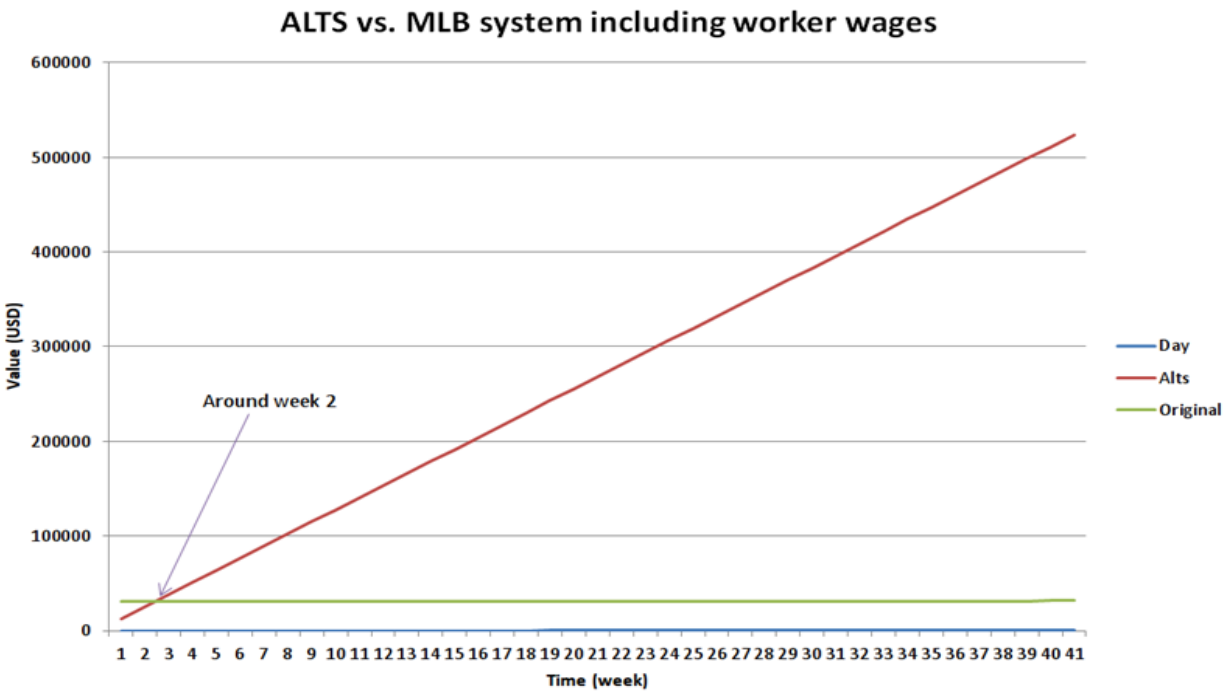


Figure 3: Return on investment between ALTS and MLB worker including worker wage

Figure 3 shows how the airport can achieve a reliable return on investment in the long run. Although the ALTS starts at a higher price (at around $9,037), after about two weeks, the airport will get back their initial investment.

If the wages for workers are not considered, then the relationship is shown in Figure 4:

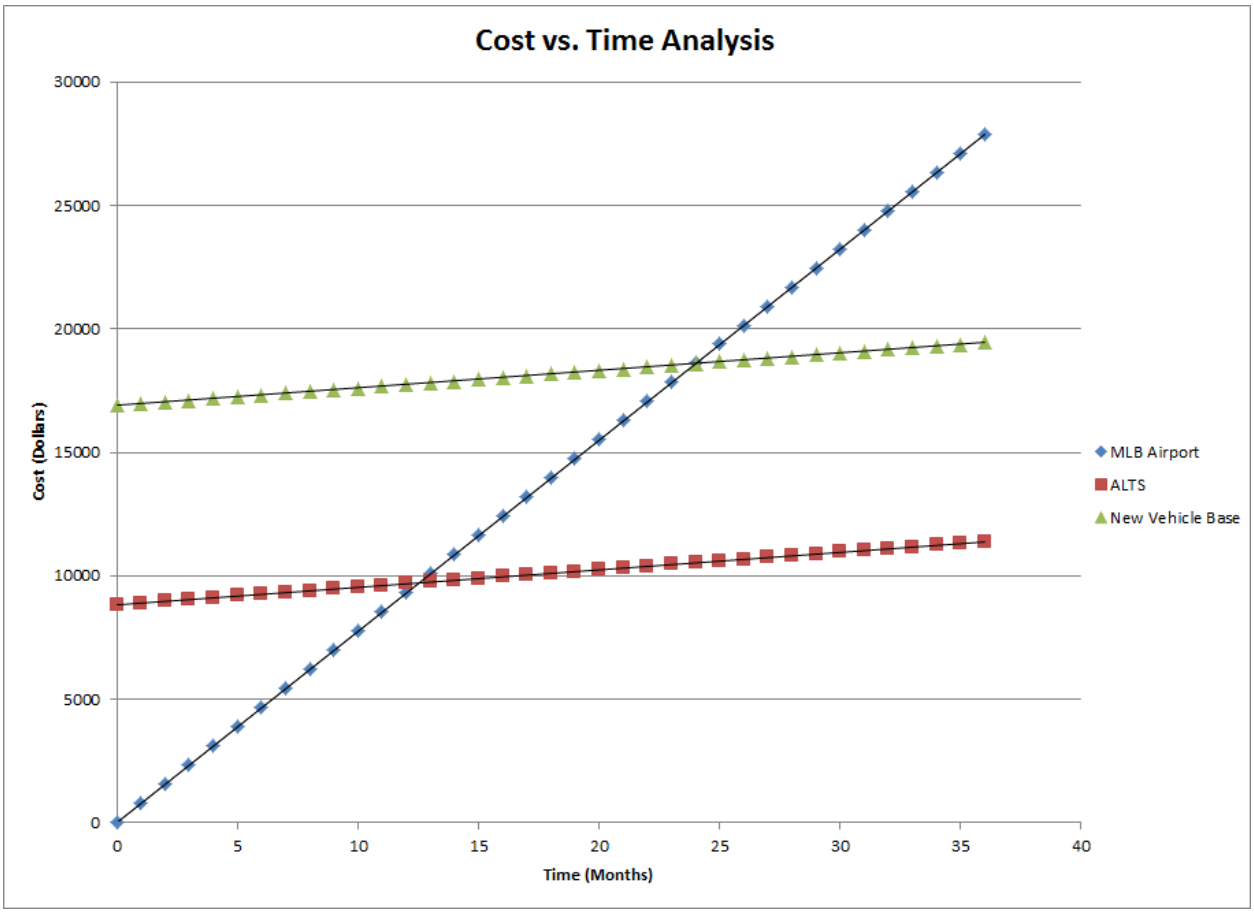


Figure 4: Return on investment between ALTS and MLB worker not including worker wage

On a monthly basis, this shows that the airport can get back their initial investment after 12.5 months which is a little over a year. This graph also shows when the airport would get back their initial investment if they used a new vehicle base. The blue line represents the current MLB airport system, the red line indicates the ALTS, and the green line mirrors what would happen if the airport superimposed the ALTS system onto a new electric golf cart.

3.2.2 Maximum Torque on Steering Column

For steer by wire, the decision on the steering design depends on the steering torque of the steering column. A study of the steering torque at parking speeds was done by Imperial College of Science, Technology and Medicine. Appendix G shows a graph based on their

research. Reading from the graph, the maximum steering torque at the steering column is estimated to be 25 N*m. However, the weight of the vehicle, the loads at the front end, and the gear ratio of the vehicle are all factors which can change the steering torque.

The experimental steering torque measurement on the ALTS's vehicle base was taken, as shown in Table 5. In order to measure this torque, a weight scale was used on the hand wheel. By pulling the scale, the force to turn the hand wheel can be measured. The steering torque is the force multiplied by the radius of the hand wheel. This value was found to be 11.5 N*m with no additional load. A second case which has a single person sitting on the front driver seat results in a steering torque of 20.3 N*m. The final case is two persons sitting at the front driver seat, resulting in a steering torque of 27.1 N*m.

The expected load case for the ALTS is one in which there is no load at the front and 10 luggage bags at the back. If a person is sitting at the front and overriding the vehicle, the steer by wire system will not be needed. Therefore, the steering torque should never be as high as the case including two people. A more accurate case which has 10 luggage bags sitting at the bed will be demonstrated. The maximum steering torque is estimated to be 25 N*m.

Table 5: Steering Torque Measurement

Load	Torque at steering column (N*m)
No additional load (0 N)	11.5
One person at the driver seat (854.5 N)	20.3
Two persons at the driver seat (1708.9 N)	27.1

3.3 Decision Matrix

3.3.1 Sensor Fitting Options

In order to determine the path that the project would travel there were three sensor outfitting cases that were considered: Case 1- Map Guided System (A), Case 2- Map Guided System (B), and Case 3- Line Guided System. The differences between Case 1 and 2, were the methods of long range detection. Case 1 used a 360º/3D LIDAR, which created a laser mapping of the surroundings, and a simple video camera, used for basic vision for a user. Case 2, however, only used a stereo camera, which created basic imaging as well as obstacle distance mapping. Case 1 was eliminated due to the extensive cost of the sensors, as shown in Table 6. Case 3 demonstrated an entirely different method of navigation, where the map guided system used solely a differential GPS for navigation, the line guided system used pre-painted lines (a set path) and line sensing technology to travel along those lines. Case 3 would use photoelectric sensors, to detect the lines, in series with a standard GPS, to determine direction along the lines and a normal camera to view the path. Case 3 was determined to be the cheapest option, as shown in Table 6, however, Case 3 was also the most difficult to implement at the airport due to the large number of changes that need to occur to the tarmac, so it was concluded that Case 2 would be the best option due to the cost as well as the ability to easily integrate into the current airport system without any changes being made.

Table 6: Sensor Pricing Chart

Type of Sensor	Sensors	Price ($)	Quantity		
			Map Guided System A	Map Guided System B	Line Guided System
GPS	Differential GPS (RTK Kit 915 MHz)	1000	1	1	0
	Standard GPS	40	0	0	1

Microcontroller	Jetson TK1	192	1	1	1
Camera/Light	Color serial JPEG w/ infrared	50	2	0	2
	ZED Stereo camera	450	0	2	0
	Photoelectric Sensor	40	0	0	4
Ultrasonic	Ultrasonic (5m)	40	4	4	4
Laser	3D LIDAR 360	8000	1	0	0
		Total Price	**$9,452.00**	**$2,252.00**	**$652.00**

3.3.2 Steer by Wire Options

For steer by wire, three options were considered. In Option A, a linear actuator is connected to a front wheel. When the linear actuator extends, it will move the front wheels accordingly. A linear actuator often comes with linear position feedback. The ALTS can use that information to calculate what angle the wheels are at. Option B uses a servomotor to turn the steering column. The servomotors provide relatively high maximum torque. They can be applied to a gear on the steering column directly. A DC motor provides relatively high speed. However, the maximum torque which a DC motor can provide is very low. Therefore, in order to use DC motor in Option C, a gear box must be added. The objective of the gearbox is to increase the maximum torque. A potentiometer or an encoder must be added to provide position feedback.

In addition to the weight of the vehicle base, the ALTS will also carry at least 10 luggage bags. The total load on the front wheels is very high. Therefore, the linear actuator in Option A has to provide a very high force in order to push the front wheels. There exist linear actuators that can provide such high force. However, these linear actuators require an AC voltage or a very

high DC voltage. The ALTS can only provide maximum 48 DC voltage. It is unrealistic to consider Option A.

As shown in Section 3.2.2, the maximum steering torque at the steering column was estimated to be 25 N*m. In Option B, a servomotor with a maximum torque 11.3 N*m was found. It costs $289.99. It can provide a relatively high torque. However, it still does not meet the estimated maximum torque. A servomotor that provides enough torque costs a lot more than $400 and require AC or much higher DC voltage. Furthermore, a servomotor usually has limited rotatory angle. In most cases, the maximum rotatory angle is 270 degrees. The steering column turns 540 degrees in both clockwise and counterclockwise. The servomotor cannot allow the steering column to work properly. Therefore, Option B will not be used.

Option C will modify the DC motor to meet the requirements. It will provide enough torque to turn the steering column even when the vehicle is not in motion. It will allow the steering column to turn 180 degrees in either direction. Adding a potentiometer or an encoder will provide position feedback of the steering column. Table 7 shows the decision matrix of the Steer by Wire options. Overall, Option C fits the ALTS the most.

Table 7: Steer by Wire Decision Matrix

Objective	Option A	Option B	Option C
Meet force/ torque requirements	√	×	√
Move the wheels to desired angle	√	×	√
Operation voltage under DC 48 V	×	√	√
Provide position feedback	√	√	√
Cost lower than $400	×	√	√

3.4 QFD

As viewable in Appendix N, the House of Quality shows the relationship between the requirements (what) and the requirement verification methods (how). From the QFD Chart, it

was shown that the ability for the ALTS to not impact the day-to-day operations at the airport and also to follow all FAA rules & regulations for vehicle operation was the most important overall requirement. A few other notable requirements were the ability for the ALTS to operate autonomously and in an airport setting, carry a minimum weight of at least 10 bags per trip, travel a maximum speed of 15 mph, and use a sustainable energy source. For a complete list of the requirements and their respective verification methods view the Appendix M, the numbered value assigned in the requirements list directly relate to the numbers assigned in the QFD Chart.

3.5 Bill of Material

	Item	Retail Cost	Quantity	Total Cost	Spent to Date
MECHANICAL	Golf Cart	$3,850.00	1	$3,850.00	$3,850.00
	Aluminum	$317.49	1	$317.49	$317.49
	Hardware & Paint	$313.19	1	$313.19	$313.19
	Steel Sprocket, 0.5" bore	$8.00	2	$16.00	$16.00
	Steering Bearing Assembly	$4.82	4	$19.28	$19.28
	Linear Actuator Braking	$129.00	1	$129.00	$129.00
	Thrifty Throttle	$29.00	1	$29.00	$29.00
	#35 Chain	$12.00	1	$12.00	$12.00
	#35 Chain link	$0.75	5	$3.75	$3.75
	1/8x1/8x0400 Machine Key, steel (am-1018)	$0.50	2	$1.00	$1.00
ELECTRONICS	NVIDIA Jetson TK1	$199.00	2	$398.00	$398.00
	Puget System Acrylic Development Enclosure for NVIDIA Jetson TK1	$35.90	1	$35.90	$35.90
	Sabrent 4-Port USB 3.0 Hub with LED	$12.99	1	$12.99	$12.99
	Arduino Leonardo	$35.86	1	$35.86	$35.86
	Swiftnav RTK	$995.00	1	$995.00	$995.00
	ZED Camera	$449.00	1	$449.00	$449.00
	MB1003 HRLV-MaxSonar-EZ0, 5 pack	$169.79	1	$169.79	$169.79
	2.5” CIM Motor am-0255	$28.00	1	$28.00	$28.00

	Sonic Shifter	$279.00	1	$279.00	$279.00
	Digital Potentiometer	$0.98	3	$2.94	$2.94
	Small Electrical Components	$485.06	1	$485.06	$485.06
	Conv DC/DC 0.5A 17-48VIN 12VOUT	$14.93	2	$29.86	$29.86
	CONV DC/DC 12V 16.7A 200W	$219.40	1	$219.40	$219.40
	SWITCH ROLLER SPDT 15A QC 125V	$6.28	6	$37.68	$37.68
	SWITCH ROCKET DPDT 20A 125V	$4.91	6	$29.46	$29.46
	SWITCH 20A LONG LEVER 125V	$15.51	2	$31.02	$31.02
	Victor SP Speed Controller	$60.00	1	$60.00	$60.00
	12V, 12000 mAh DC Battery	$80.00	1	$80.00	$80.00
	120 Amp Breaker	$30.00	1	$30.00	$30.00
Power	Tektrum 200W 48V Solar Panel	$889.90	1	$889.90	$889.90
SHIPPING	**Shipping fee**			$47.57	$47.57
Total:				**$9,037.14**	**$9,037.14**

3.6 Assembly

3.6.1 Bed Modifications

Since the ALTS vehicle base has a long bed, this is the primary location for the luggage to be stored during transportation around the airport. The first option for luggage storage on the ALTS is to build a frame to put on the back to hold the luggage while driving. With this option, the lift/dump bed on the frame of the ALTS vehicle would be utilized and aid in the dropping off of luggage at necessary locations. However, with this option, the luggage will be dropped off from the bed. This may damage the fragile items in the luggage. Moreover, it is hard to access the luggage in the inner section of the bed. Therefore, the final decision on the storage frame is a frame with a roof, front, and back while the sides are open for easy access to the luggage to be loaded and unloaded from either side of the ALTS.

The bed of the ALTS was first modified by removing the large food storage compartment, originally installed by 6 bolts. This allowed the team to measure the appropriate dimensions of the frame that would be welded onto the cart. The frame that would be attached would also act as a mount for the solar panels as well as an enclosure to protect the baggage from weather phenomena. The dimensions of the frame were 108" by 48" by 48". The frame was welded by cutting 14 pieces of square aluminum rods to form the base.

After the base was constructed, we then riveted thin $1/16^{th}$ inch aluminum sheets to the frame. The sheet aluminum was bent to ensure no leaks would develop across the frame as another barrier of the rain. The full constructed frame was then welded on to the base of the frame along four points to ensure a stable base.

Once the frame was completed the ALTS was then primed and painted with 3 layers. The ALTS was then ready to attach to solar panels. The roof accommodated a single hole to run the electrical wires down the frame which ran across the bottom of the ALTS into the respected positive and negative terminal of the batteries. The frame was then equipped with 8 push buttons on each side that were drilled into the frame. The other side of the button was then attached onto the PVC matt cover. Finally, the cart was sprayed with bed liner to ensure a rough layer to hold the baggage in place, so that it will not slide around turns and during acceleration and deceleration.

Figure 5: Bed Frame after Modification

3.6.2 Mechanical Additions

To successfully accomplish full autonomy for the chosen vehicle base, the current mechanical system has to be modified or added to in order to allow for autonomous control using the added on board computer.

3.6.2.1 Steer by Wire

Three options for steer by wire were discussed at Section 3.3.2. Option C, modifying a DC motor, is the better option for the ALTS.

As Section 3.2.2 mentioned, the maximum steering torque at the steering column is estimated to be 25 N*m. The torque which a DC motor can withstand is much lower than 25 N*m. For instance, the CIM motor from AndyMark can provide a maximum torque of 1.21 N*m at the angular speed 2655 rpm. It is only one-twentieth of the required torque. Therefore, a gearbox is added to increase the torque.

Using the Sonic Shifter gearbox from AndyMark which has a low gear ratio of 30.1:1, the maximum torque at the output shaft of the gearbox is increased to 36.42 N*m which is much higher than the estimated required torque. The speed of the output shaft is 1.47 turns per second. One sprocket is installed on the output shaft of the gearbox and one sprocket is welded on the original steering column of the vehicle base. A chain is added to connect the two sprockets, so that the steering column can be controlled by the motor. An encoder is in line with the output shaft in order to provide the position feedback to the Arduino Leonardo.

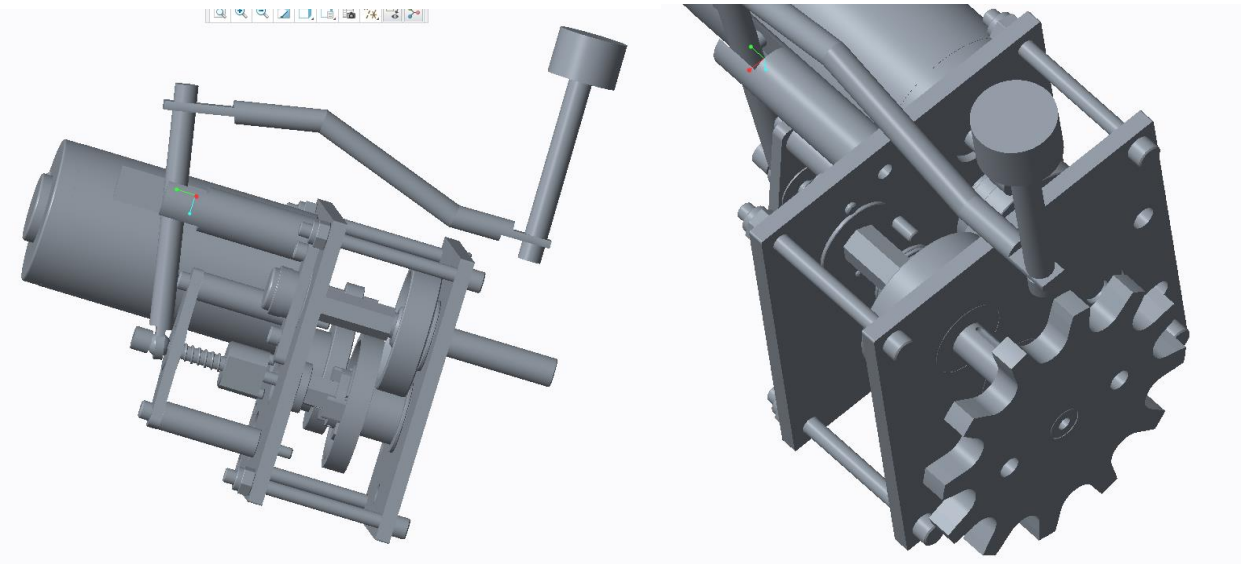


Figure 6: CAD Model of the Steer by Wire System

The ALTS also features manual override under special condition. A shift is designed to switch between the high gear and the low gear sets in the Sonic Shifter gearbox. For the implementation of the ALTS, the high gear set is removed. The gearbox now can be switched between the low-gear slot and an open slot.

During autonomous operation, the switch is in the low-gear slot. If the ALTS needs to be overridden manually for any emergency, there is a lever with a clutch release mechanism

attached on the dash of the ALTS. When the clutch release is activated, the gearbox is disengaged from the DC motor, making the ALTS steering wheel completely manual. Behind the gearbox is a switch. When the lever is pulled and locked into position, it will shut off all three autonomous systems so that the ALTS will return to its fully manual setting. When autonomy needs to be reactivated the lever is simply put back in autonomy position. The gears will be disengaged from the output shaft.

3.6.2.2 Brake by Wire

The ALTS already has a current braking system due to the vehicle base, but for it to be completely autonomous there must be some adjustments made. An actuator is needed to engage the brake pedal for the ALTS to stop; but there also needs to be a way of implementing a manual override in case of emergencies.

The main challenge in the design of the braking system was designing a system that can run autonomously and still be able to be controlled manually by a driver. After a few design iterations we settled on the design shown in Figure 7.

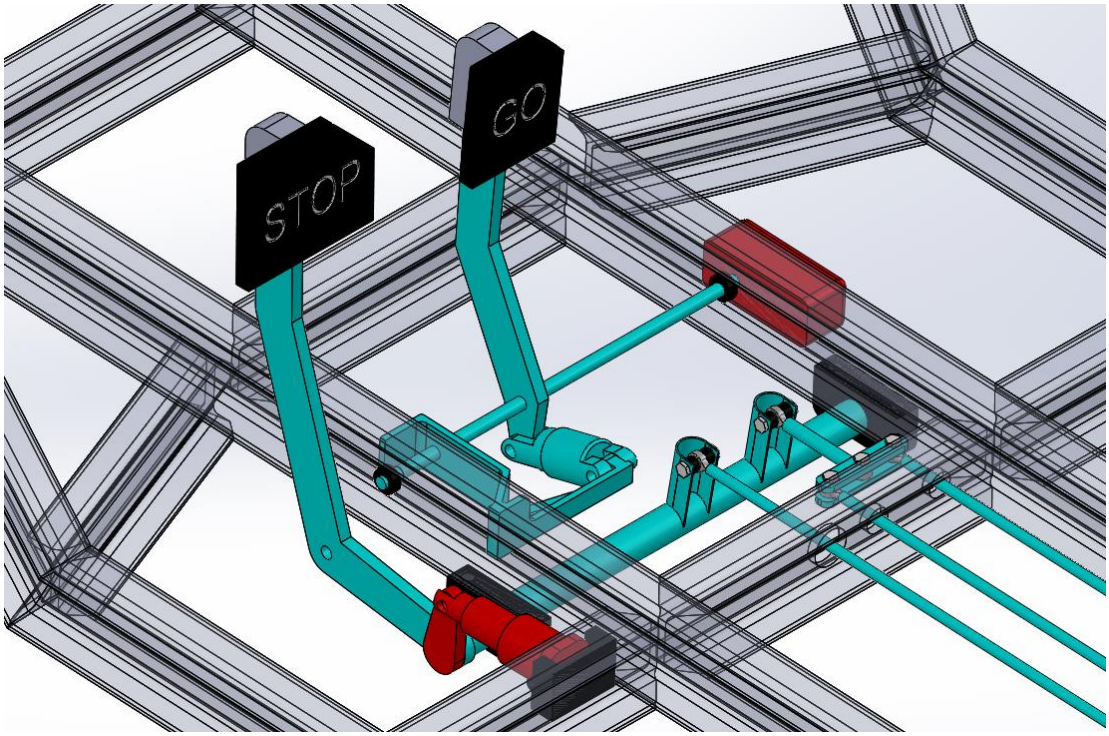


Figure 7: Additions to braking and throttle (marked in red)

The golf cart's original braking system is shown in blue and our modifications are shown in red. Our modifications include a linear actuator that is attached to the braking shaft. As the linear actuator extends, the brake pedal is pressed which engages the brakes on the tires. In order for the brakes to also be used manually, a slip for the actuator was designed and built that allows the brake to be pressed even when the actuator doesn't move. A design for the slip is shown in Figure 8.

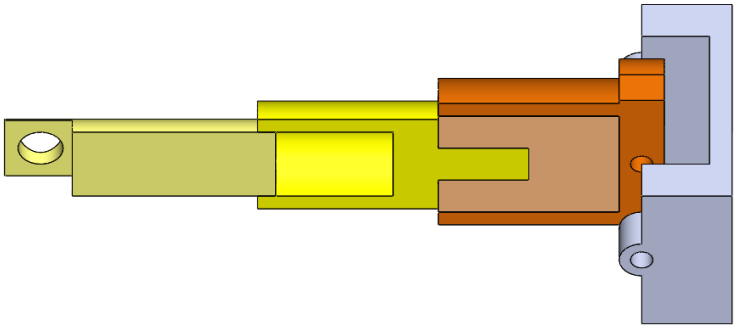


Figure 8: Actuator with slip for braking

This slip allows the brakes to be pressed further than the actuator is extended; allowing manual override in an emergency situation.

3.6.2.3 Throttle by Wire

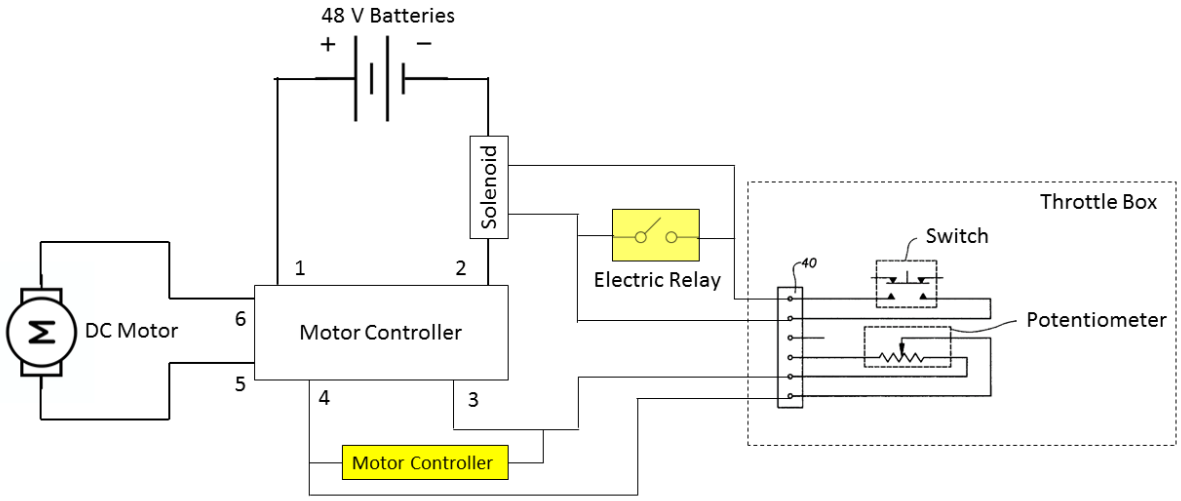


Figure 9: Throttle Schematic

In order to allow the ALTS to operate autonomously the throttle must be able to be controlled using electrical signals instead of using the pedal. In order to bypass the throttle box and allow for electrical control, an electrical controller and a smaller motor controller was added. Figure 9, above, is a circuit diagram depicting the major components in the ALTS's throttle subsystem. The additions to the system are highlighted in yellow.

The original system consists of two main components: a mechanical switch and potentiometer. When the pedal is initially pressed, the mechanical switch is closed. This allows current to flow to the solenoid, closing it. The solenoid acts as a switch for the high voltage

circuit including the batteries and the motor, so the motor will not function unless the solenoid is closed.

As the pedal is pressed down, the mechanical potentiometer in the throttle box increases linearly from 0 to 5000 ohms. This change in resistance reduces the voltage being inputted to the motor controller at terminal 3, increasing the voltage difference between terminals 3 and 4. As the voltage difference between the two terminals increases, more power produced by the motor and the speed of the ALTS increases.

In order to control the ALTS using electric signals we bypassed the switch and potentiometer in the throttle box with an electric relay and a smaller motor controller. The electric relay is hooked up in parallel with the original mechanical switch. This allows for the solenoid to be closed if either the mechanical switch is closed or the electrical relay is closed. A smaller motor controller was also added in parallel to the potentiometer in the throttle box. This motor controller allows the voltage at terminal 3 to be altered using signals from an Arduino. These two modifications to the ALTS throttle subsystem allow it to be operated autonomously while keeping the option of manual control in case of an emergency.

3.6.2.4 Renewable Energy

For the ALTS to meet the renewable/sustainable energy requirement it will be beneficial to add supplemental energy to the vehicle batteries through another energy source. This additional power will increase the battery life throughout the normal operating day for the ALTS, reducing downtime and saving the airport money.

Several renewable and sustainable energy sources were researched including wind energy, Triboelectric, kinetic energy and solar energy. Wind energy was taken into consideration because with Melbourne, Florida being a coastal city on the Atlantic Ocean, strong winds are common on

a daily basis. This gave the idea that utilizing the wind energy could be a very good opportunity for adding additional energy to the ALTS. Wind energy was further researched and it was found that the average daily constant wind speed in Melbourne, Florida is 7 mph. The team then looked into possible wind turbines that would be applicable and able to install on the ALTS. After finding the best wind turbine option to implement, the most energy able to be collected based off of calculations was about 5 watts. With the energy production this low, wind energy would not be worth the investment. When doing research on the kinetic energy and the triboelectric energy, even the systems with the lowest cost were found to be well out of the budget for this project.

The ALTS is equipped with solar panels for its supplemental energy source. Solar power is the conversion of sunlight into electricity. This can either occur directly using photovoltaic (PV) cells or indirectly using concentrated solar power (CSP). Because the power produced by a solar panel is directly related to the ultraviolet (UV) index, it would be beneficial to know the UV index for Melbourne, Florida. It was calculated that the average UV index in Melbourne per year is 7.75, which is significantly higher than the U.S. average. Due to the high UV index, photovoltaic cells are the best option. Tektrum Universal 200w, 48V monocrystalline panels are installed on the ALTS.

The Tektrum solar panels have an efficiency of 22% while the other quality brand panels have 19% - 20%. The solar panels are also tempered for high temperatures. The efficiency will decrease as temperatures approach the upper 65-85 degrees Celsius. (refer to solar chart). During a day with high UV-index, we can expect to generate 200W of power and 98W of power on an average day. This will not allow the system to be powered purely off solar power, but it is estimated to increase travel distance by 22%.

The Tektrum solar panels were able to supplement our energy requirements, charging our batteries in a total of 7 hours and 46 minutes. Data was taken during March 2, 2016 and over the time span of 6 hours' data was collected every 15 minutes with the voltage produced. Over the course of 6 hours we were able to generate 130.84 Watts and recorded a total of 3.271 kW. This is about 27.69% of the total energy needed to charge our 9.06 kWh batteries. The day we experimented with the solar panels was partly cloudy with a 73 high and 65 low. Overall the solar panels were able to ensure us 3 ½ hours of extra drivable time.

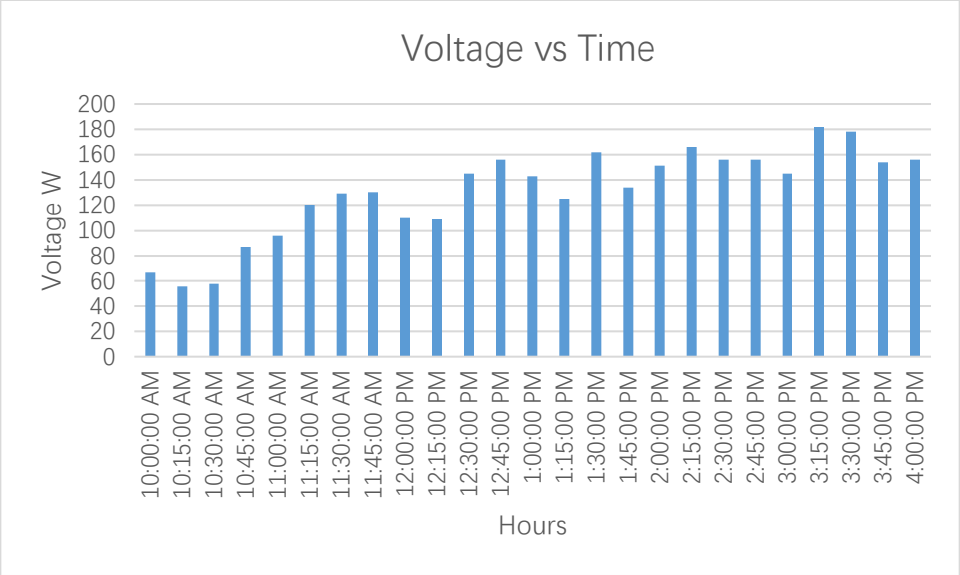


Figure 10: Voltage Vs Time March 2,2016

Figure 11: Solar Panel Configuration

3.6.2.5 Batteries

Although the existing batteries in the golf cart base are sufficient, the team researched possible battery options in the event that the existing batteries require replacement. The current batteries powering the vehicle base are Crown Deep-Cycle batteries; there are six 8V batteries that are a year old. When the voltages were checked on the batteries, only two of the batteries registered a reading below 8V. This proves that the installed batteries are still like new and are sufficient for the ALTS. Refer to Figure 11 for a picture of the installed batteries. If the batteries need to be replaced, the team chose Trojan T-875 batteries. The Trojan batteries offer a superior capacity with regards to the amp-hours with a comparable recharge rate. The deep-cycle flooded

batteries use a technology which enables it to stay cool during charging and discharging phases. The reason this occurs is because batteries expand when exposed to continuous use and high temperatures.

Figure 12: Current Batteries in Vehicle

3.6.3 Electronic Control System Additions

To connect the steer by wire, brake by wire, and throttle by wire and control them, a proper control system was designed that has a high graphics powered computer which can take a GPS signal and a stereo camera with sonar sensors in order to control the mechanical systems. Proper autonomous algorithms will be written to abide by the paths that need to be taken and the regulations that have to be met. A system wiring diagram was developed for the proposed system and can be seen in Appendix F.

3.6.3.1 NVIDIA Jetson TK1

The main computer that was chosen to handle all of the sensors and autonomous algorithms was the NVIDIA Jetson TK1 (Figure 13). The major decision that led to this choice was the high graphics power required by high frame rate video distance rendering. The board is built using the

same technology as the NVIDIA Tegra K1 SoC which uses cores from the NVIDIA Kepler graphics processing unit. There are 192 kepler CUDA cores which, coped with the ARM Cortex A15 central processing unit, will be able to handle high frame rates with distance rendering. The board is built around being a developer board so all of its ports and classes are accessible to the user. The Jetson features a USB 3.0 version 3 port which allows for high capacity flow thru of data which will be more than enough to handle a stereo camera setup along with a variety of GPS modules and control units. The operating system on the board is Linux Ubuntu 14.0.4 which allows for root kernel access to control all components within the board with ease. The board includes the CUDA library and OpenGL 4.4 which will be used with a C++ compiler to write the autonomous algorithm. The Jetson also allows for programming using Python, C#, and Java which allows flexibility to use previously created control algorithm and integrate them with the main C++ method. See Appendix A for the Jetson TK1 Specifications sheet and Appendix P for installation.

Figure 13: NVIDIA Jetson TK1[4]

3.6.3.2 Arduino Leonardo

Since the Jetson TK1 is primarily set up as a video processing board, the Arduino Leonardo (Figure 14) was selected as the main sensor and drive control unit. The Arduino will be

programmed only to take sensor inputs and send them to the Jetson and take Jetson commands and send them to the drive; in essence acting as a bridge between the Jetson and the rest of the electrical system. The ports on the Jetson TK1 were researched and it was determined that they could not be used to directly control the drive system. After researching and validating, the Arduino Leonardo was commonly paired with the Jetson for robotic controls. This is because the Arduino is equipped with 16 bit ARM core processor, which will allow for the proper processing speed to be achieved in order to have a millisecond response time between the sensors and the drivetrain being controlled. The Arduino Leonardo provides the system response time necessary to be able to process commands from the Jetson TK1 with minimal delay. See Appendix B for the Arduino Leonardo specifications sheet and Appendix P for installation.

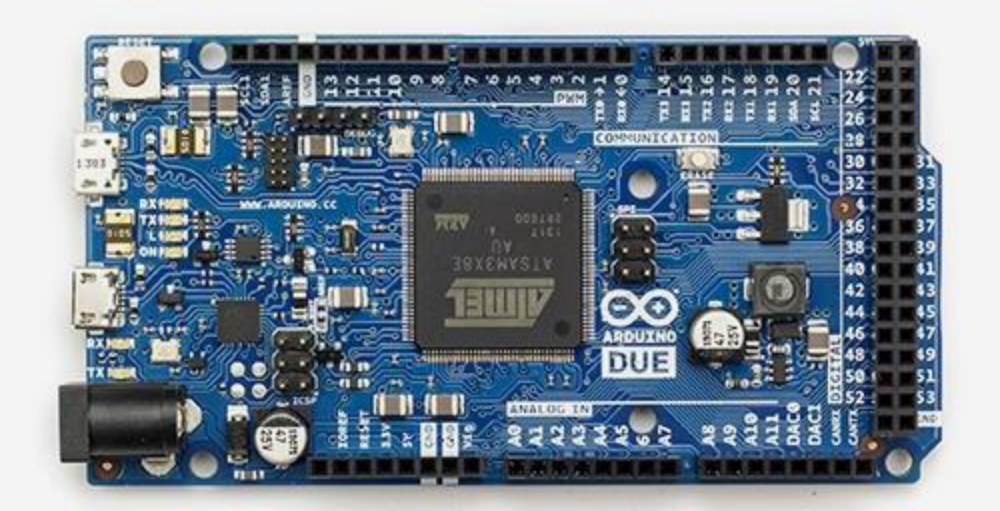


Figure 14: Arduino Leonardo 16 Bit[8]

3.6.3.3 *Coding*

The majority of the autonomous control was coded using the on board Arduino with an additional Relay Shield. To begin the code the pins were initialized using the following layout:

```
int Relay1 = 7;
int Relay2 = 6;
int Relay3 = 5;
int Relay4 = 4;
int PWM9 = 9;
int PWM10 = 10;
int PWM11 = 11;
int softstop = 1;
int brakepot = A1;
int temp = A2;

#include <Encoder.h>
Encoder myEnc(2, 3);
```

```
void setup() {
  //Serial.begin(9600);
  pinMode(Relay1, OUTPUT);
  pinMode(Relay2, OUTPUT);
  pinMode(Relay3, OUTPUT);
  pinMode(Relay4, OUTPUT);
  pinMode(PWM9, OUTPUT);
  pinMode(PWM10, OUTPUT);
  pinMode(PWM11, OUTPUT);
  pinMode(softstop, INPUT);

  pinMode(brakepot, INPUT);
  pinMode(temp, OUTPUT);
}
```

The relay shield uses pins, 4, 5, 6 and 7 to activate the throttle relay, brake on relay, brake off relay, and horn relay. 3 PWM signals were used, one for the steering DC motor and one for the throttle speed control box. The third was a spare PWM port. The soft stop switch was set to port 1. The brake potentiometer was set to read into Analog Port 1. The speed encoder was using ports 2 and 3 to read the encoder points. The encoder class was imported from the Arduino library and initialized. Following the naming, all pins were enabled using their proper inputs or outputs.

The class that controls the zed camera stop and start is written below:

```
void checkonoff(){
  while(digitalRead(softstop) == HIGH){
    delay(1000);
    Serial.println("WAIT.....................");
    analogWrite(PWM10, 185);
    analogWrite(PWM9, 185);
     digitalWrite(Relay1, LOW);
      digitalWrite(Relay2, LOW);
       digitalWrite(Relay3, LOW);
        digitalWrite(Relay4, LOW);
  }

}
```

The Jetson TK1 was installed into port 1 in the Arduino. It was utilizing the read portion only of the UART connection of the TK1. If the stop signal was triggered, the system will turn off all speed controllers and all relays until the signal is given to continue from the TK1.

The code to handle the right and left turning is written below. It is written using the P portion of PID control, utilizing a basic feedback of position in the encoder attached to the gear box.

```
void turn(char x, int t){
  int temp = t;
  if(x=='r'){ Serial.println("Turning Right");
  while(temp > 0){
    checkonoff();
    encupdate();
    if(encpos > 1200 && encpos < 1300){
      checkonoff();
      encupdate();
      analogWrite(PWM10, 215);
      Serial.print("MID");
    }
    else if(encpos < 1300){
      checkonoff();
      encupdate();
      analogWrite(PWM10, 220);
      Serial.print("HIGH");
    }
    else{
    checkonoff();
    analogWrite(PWM10, 185);
    Serial.print("OFF");
    }
  temp = temp-1;
  Serial.print("     ");
  Serial.print(temp);
  Serial.print("     ");
  Serial.println(encpos);
  delay(1);
  }
    analogWrite(PWM10, 185);
  }
if(x=='l'){ Serial.println("Turning Right");
while(temp > 0){
  checkonoff();
  encupdate();
  if(encpos < -1200 && encpos > -1300){
    checkonoff();
    encupdate();
    analogWrite(PWM10, 165);
    Serial.print("MID");
  }
  else if(encpos > -1300){
    checkonoff();
    encupdate();
    analogWrite(PWM10, 155);
    Serial.print("HIGH");
  }
  else{
  checkonoff();
  analogWrite(PWM10, 185);
  Serial.print("OFF");
  }
temp = temp-1;
Serial.print("     ");
Serial.print(temp);
Serial.print("     ");
Serial.println(encpos);
delay(1);
}
  analogWrite(PWM10, 185);
}
```

For both the left and the right turn, there are 1300 points from the center that the encoder reads to pull a full turn. The motor in the gearbox is ramped up to full speed until the encoder position is 1200 points off center. Once this point is reached the motor speed is turned down to

60% to finely reach the 1300 point marker exactly. The loop keeps running as long as the turn needs to be held. To account for the play in the rack and pinion system, the code keeps running so that if the wheels deviate from 1300 it runs at low or high speed depending on how far and how quick the wheels turn away from point. After the turn is made the code to straighten out the wheel is run and is shown below:

```
if(x=='s'){ Serial.println("Straightening Out");

  while(temp > 0){
    checkonoff();
    if(encpos > 300){
      checkonoff();
     encupdate();
     analogWrite(PWM10, 155);
     Serial.print("HIGH L");
    }
    else if(encpos < -300){
      checkonoff();
      encupdate();
      analogWrite(PWM10, 220);
      Serial.print("HIGH R");
    }
    else if(encpos > 80 && encpos < 299){
      checkonoff();
     encupdate();
     analogWrite(PWM10, 165);
     Serial.print("MID L");
    }
    else if(encpos < -80 && encpos > -299){
      checkonoff();
      encupdate();
      analogWrite(PWM10, 215);
      Serial.print("MID R");
    }
    else{
      checkonoff();
      encupdate();
      analogWrite(PWM10, 185);
      Serial.print("OFF");
    }
    encupdate();
    temp = temp-1;
    Serial.print("     ");
    Serial.print(temp);
    Serial.print("     ");
    Serial.println(encpos);
    delay(1);
  }
}
```

The principle behind the straightening out code is the same as the left and right turn code. The only difference is that it re-aligns the wheel to the center with a 0 margin being +/- 80 of the

true 0 on the encoder. If the wheels veer off center the PWM is engaged to run the motor in forward or reversed based on which direction the wheels need to be turned to maintain a straight path.

The throttle code is shown below:

```
void throttleon(){ Serial.println("Throttle on");
checkonoff();
  digitalWrite(Relay3, HIGH);
  analogWrite(PWM9, 235);
}

void throttleoff(){ Serial.println("Throttle off");
  analogWrite(PWM9, 185);
  digitalWrite(Relay3, LOW);
}
```

The throttle code to turn on the throttle first turns on the motor relay, Relay 3, then turns on the speed controller to a high setting. When the throttle off request is used, the speed controller is turned off followed by turning the motor relay off so that no power is drained by the idling of the motor and that the cart is not electronically braked by the resistance in the enabled motor.

The brake code is below:

```
void brakeon(){ Serial.println("Brake Actuated");
checkonoff();
  while(analogRead(A1) < 390){
    checkonoff();
digitalWrite(Relay2, HIGH);
}
digitalWrite(Relay2, LOW);
}

void brakeoff(){ Serial.println("Brake Deactuated");
checkonoff();
  while(analogRead(A1) > 150){
    checkonoff();
  digitalWrite(Relay1, HIGH);
}
digitalWrite(Relay1, LOW);
}
```

When the brake on command is given, the brake will actuate using relay 2 till the analog reading from the potentiometer is 390 mV. When the brake off command is given, the brake will actuate in reverse until the potentiometer reaches 150 mV.

The main method in the Arduino tracks the signals from the GPS for left and right turns and straight-aways, and the start and stop signals from the TK1. With the code listed above, the Arduino was fully programmed.

3.6.3.4 Stereo Labs ZED Camera

The stereo camera of choice is the Stereo Labs ZED Developer Camera (Figure 15). This camera was chosen due to the fact that it was designed around the NVIDIA Jetson TK1. The camera is fully open source and features sample codes for distance recognition and a full SDK with all needed codes and classes. The camera runs on the Jetson TK1 at 60 FPS frame rate with full 4000x2000 resolution. The camera includes a class that can be called to output the distancing in millimeters to a point cloud matrix, in essence the camera shows how far the object of each pixel is on the image. The camera SDK simplified the coding to where no coding has to be done

to make a driver algorithm for it as it has already been done by Stereo Labs. The camera is connected using the Jetsons USB 3.0 port to allow for the high processing rate of each frame. This camera is setup for autonomous robot developers. See Appendix C for the Stereo Labs ZED Camera specifications sheet.

Figure 15: Stereo Labs ZED Camera[5]

Initially, the code from StereoLab allows the ZED SDK to output a distance matrix of an image it takes. This distance matrix gives the depth of every single pixel on that image. The algorithm that we developed is then used to find out the difference in distance between consecutive pixels. After that, the locations of objects are then determined using distance differences that were previously calculated. The final result is a matrix of the same size, but only 2 values are given: 0 if there is no object at that pixel and 1 if there is an object. These numerical values are then translated into 2 signals: STOP and GO. These signals correspond to the voltages for the Arduino: 0V for STOP, and 1.8V for GO. The Arduino will then control the brakes on the ALTS depending on the voltage it receives. See Appendix P for installation.

3.6.3.5 ZED Camera Code

```
//standard includes
#include <iostream>
#include <fstream>

//OpenCV includes
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>

//ZED includes
#include <zed/Camera.hpp>
```

For any C/C++ programs, the use of includes are needed. These includes are used so that the program can be compiled correctly. To read inputs/outputs, iostream and fstream are used. Additionally, opencv2 and zed/Camera.hpp are necessary to run the computations via the ZED SDK. Lastly, the "zed.h" is a custom header that contains many functions required for the algorithm.

Due to simplicity, the code “main.cpp” only contains 1 function, which is int main(). Parts of the main function are explained below:

```
   sl::zed::Camera* zed = new sl::zed::Camera(sl::zed::VGA,15);

sl::zed::ERRCODE err = zed->init(sl::zed::MODE::PERFORMANCE, 0, true, false, false);


    //These are settings for the ZED camera, changing these values will change the qualit

    zed->setCameraSettingsValue(sl::zed::ZED_BRIGHTNESS, 4, true);
    zed->setCameraSettingsValue(sl::zed::ZED_CONTRAST, 4, true);
    zed->setCameraSettingsValue(sl::zed::ZED_HUE, 0, true);
    zed->setCameraSettingsValue(sl::zed::ZED_SATURATION, 6, true);
    zed->setCameraSettingsValue(sl::zed::ZED_GAIN, 4, true);
    zed->setCameraSettingsValue(sl::zed::ZED_WHITEBALANCE, 4600, true);


    zed-> setCameraSettingsValue(sl::zed::MODE::PERFORMANCE, 0, true);

   // Quit if an error occurred
   if (err != sl::zed::SUCCESS) {
      std::cout << "Error: while ZED initialization: " << errcode2str(err) << std::endl;
      delete zed;
      return 1;
   }

   // Initialize color image and depth

   int width = zed->getImageSize().width;
   int height = zed->getImageSize().height;
   cv::Mat depth(height, width, CV_8UC4,1);
   cv::Mat image(height, width, CV_8UC4,1);
```

This first part tells the computer to call the important parameters from the ZED SDK. These parameters are: fps and quality, the camera settings, the mode, and the dimensions of the image size.

For fps and quality, the following line shows 2 parameters that can be changed:

sl::zed::Camera* zed = new sl::zed::Camera(sl::zed::VGA,15);

The two parameters are VGA and 15. VGA is an argument for the video size, while 15 is an argument for the number of frame per second. Ideally, the ZED camera can have the following video size: VGA, HD720 and 2K. Meanwhile, the number of frame per second can go as high as 60. Under testing conditions, these numbers are reduced in order to prevent stresses on the computations.

For camera settings, by changing different values correspond to brightness, contrast or white balance, the user can improve the quality of the image taken:

```
zed->setCameraSettingsValue(sl::zed::ZED_BRIGHTNESS, 4, true);
zed->setCameraSettingsValue(sl::zed::ZED_CONTRAST, 4, true);
zed->setCameraSettingsValue(sl::zed::ZED_HUE, 0, true);
zed->setCameraSettingsValue(sl::zed::ZED_SATURATION, 6, true);
zed->setCameraSettingsValue(sl::zed::ZED_GAIN, 4, true);
zed->setCameraSettingsValue(sl::zed::ZED_WHITEBALANCE, 4600, true);
```

The numerical arguments for these can be found through the ZED Depth Viewer app, each setting has a different range for numerical arguments. Additionally, it is also possible to set the mode of the camera using the following:

```
zed-> setCameraSettingsValue(sl::zed::MODE::PERFORMANCE, 0, true);
```

Due to data being taken at real time, the mode is set at "PERFORMANCE". The next lines are important because they setup the required image size to be taken:

```
int width = zed->getImageSize().width;
int height = zed->getImageSize().height;
cv::Mat depth(height, width, CV_8UC4,1);
cv::Mat image(height, width, CV_8UC4,1);
```

The first two lines tell the computer to get the width and height of the image by going through the ZED SDK. The function "getImageSize()" allows it to do so. Because the nature of the computation requires matrices and OpenCV, the next two lines append the height and width to arbitrary "Mat depth" and "Mat height". These are then used as primary dimensions when creating matrices.

```
for(int i = 0; i < 20; i++){

        // Get frames and launch the computation
        if (!zed->grab(sl::zed::SENSING_MODE::FULL))
        {
        // get image from ZED camera
        sl::zed::Mat left = zed->retrieveImage(sl::zed::SIDE::LEFT);
        memcpy(image.data,left.data,width*height*4*sizeof(uchar));

         sl::zed::Mat depthmap = zed->normalizeMeasure(sl::zed::MEASURE::DEPTH);
        memcpy(depth.data,depthmap.data,width*height*4*sizeof(uchar));
        save(image, zed->retrieveMeasure(sl::zed::MEASURE::DEPTH));

    }
    objectLocation();
    examineObject();
    //matrixToFile(image, zed->retrieveMeasure(sl::zed::MEASURE::DEPTH));
}
```

The code above is a typical user defined loop. Essentially, this forces the camera to take 20 pictures whenever it starts up. Then, the user defined functions from the zed.h file are called. These functions belong to the algorithm.

The first function called in the main function is GPIOinit(). This function can be seen below:

```
void GPIOinit(){
    //pin 40 on 75 pin
    system("echo 160 > /sys/class/gpio/export");
    system("echo out > /sys/class/gpio/gpio160/direction");
    system("echo 1 > /sys/class/gpio/gpio160/value");

}
```

The main purpose of this code is to initialize the general purpose input and output (gpio) pins. This particular code initialized pin number 40 on the 75 pin bus on the Jetson TK1. It sets it as an output pin and sets the initial value to 1 which corresponds to it outputting 1.8 volts.

The next function called is the save() function. This function takes the distance inputs from the camera and compiles them into one matrix. Parameters such as depth.data and depth.step are recalled from the main function. The save function is shown below:

```
//SAVE FUNCTION
void save(cv::Mat& image, sl::zed::Mat depth)
{
    //SAVING DEPTH MAPS

    float* ptr_d;

    for (int i = 0; i< depth.height; ++i)
    {
        ptr_d = (float*)(depth.data + i * depth.step);

        for (int j = 0; j < depth.width * depth.channels; ++j)

        {
            if (ptr_d[j] < 0 || ptr_d[j] > sensingDistance ){
                ptr_d[j] = sensingDistance;
            }

            distanceMatrix[i][j] = ptr_d[j];

        }


    }
    //std::cout << "DONE" << std::endl;

}
```

The next function called is the objectLocation() function. The main purpose of this function is to take the distance matrix that the save() function outputs and determine if there is an object at each pixel. The code for this function is shown below:

```
void objectLocation(){

    for(int i = 0; i < height; i++){

        inBoundary = false;

        for(int j = 0; j < width; j++){

            if(distanceMatrix[i][j] > sensingDistance && j < width){
                distanceMatrix[i][j] = sensingDistance;
            }

            if(j > 0){

                dDistance[i][j-1] = distanceMatrix[i][j] - distanceMatrix[i][j-1];

                if(dDistance[i][j-1] < -1*error && dDistance[i][j] >= -1*error && dDistance[i][j] <= error){ //- to 0
                    object[i][j] = 1;
                    inBoundary = true;
                }
                else if(dDistance[i][j-1] > error && dDistance[i][j] <= error && dDistance[i][j] >= -1*error){//+ to 0
                    object[i][j-1] = 1;
                    inBoundary = false;
                }
                else if(inBoundary == true && dDistance[i][j-1] >= -1*error && dDistance[i][j-1] <= error
&& dDistance[i][j] >= -1*error && dDistance[i][j] <= error){// 0 to 0
                    object[i][j-1] = 1;
                }
                else if(dDistance[i][j-1] > error && dDistance[i][j] > error){ //+ to +
                    object[i][j-1]= 1;
                }
                else if(dDistance[i][j-1] < -1*error && dDistance[i][j] < -1*error){//- to -
                    object[i][j-1]= 1;
                }

                else if(dDistance[i][j-1] < -1*error && dDistance[i][j] > error){//- to +
                    object[i][j-1]= 1;
                    inBoundary = false;
                }
                else if(dDistance[i][j-1] >error && dDistance[i][j] < -1*error){//+ to -
                    object[i][j-1]= 1;
                }

                else{
                    object[i][j-1] = 0;
                }

                if(j == width-1){
                    dDistance[i][j] = 0;
                    object[i][j] = 0;
                }
            }
        }
    }
}
```

This function takes the difference in distance between two consecutive points and determines whether or not an object is present at that pixel using the differences between the distances. There are four main cases that were accounted for in this code. The first case is that the difference in the distances goes from a negative number to zero. This is the case that represents the entrance boundary of an object. Consecutive pixels after this boundary will be the object until the exit boundary is reached. If this is the case, the program will set the object matrix to 1 at that point, representing that there is an object at that pixel, and it will set the boolean variable

inBoundary to true. This variable keeps track of whether or not the program is scanning inside an object. Another case is that the difference goes from positive to zero. This is the opposite of the previous case. This represent the exit boundary of the object.

The next two cases involve the differences going from zero to zero. This would mean that the distances for those consecutive points are basically the same value. This case can happen inside the boundaries of an object or outside the boundaries. If it happens inside the boundaries then we would want to place a 1 in the object matrix at that point to represent that there is an object at that point. If it is not inside the boundaries of an object then we would want there to be a 0 in the object matrix. To keep track of whether or not the program is inside the boundary of an object the boolean variable inBoundary was used. If the differences are both zero and inBoundary equals true then the program will output a 1 to the object matrix; if inBoundary is false and the differences are both zero then it will output a 0 to the object matrix.

These are the four main cases in the objectLocation function. This function will output a matrix that is the same size as the distance matrix but is made up of only zeroes and ones. The ones represent that an object is present at that pixel.

```
void examineObject()
{

    checkheight = floor((iEnd-iStart)/step);
    checkwidth = floor((jEnd-jStart)/step);

    int sumMatrix[480][640];
    int objectCheckSum = 0;

    //TURNING

    switch(turn){
        case -1: {jStart = 9; jEnd = 409; iStart = 0; iEnd = 240;}
        break;
        case 0: {jStart = 209; jEnd = 429; iStart = 240; iEnd = 480;}
        break;
        case 1: {jStart = 209; jEnd = 629; iStart = 240; iEnd = 480;}
        break;
        case 2: {jStart = 9; jEnd = 629; iStart = 0; iEnd = 480;}
        break;
        default: "Something went Wrong.\n";
    }
    for(int i = iStart; i < iEnd; i = i + step)
    {

        //std::cout <<"Row : " <<i <<"  ";

        for(int j = jStart; j < jEnd; j = j+step)
        {

        sum =0;
        //startx = 0;
        //starty = 0;

    //  for(int u = 0; u <= x <= 20;

            for(int u = i; u <= i + step - 1; ++u)
            {
                for(int v = j; v <= j + step- 1; ++v)
                {
                    sum = sum + object[u][v];
                }
            }

            int icheck = (int)((i-step/2)/step);
            int jcheck = (int)((j-jStart)/step);
```

```
            double maxsum = percentObject*(step-1)*(step-1);
            //std::cout << maxsum <<"\n";

            if ((double)sum >= maxsum)
            {
                objectcheck[icheck][jcheck] = 1;
                objectCheckSum++;

            }
            else
            {
                objectcheck[icheck][jcheck] = 0;
            }


        }
    }

    if(objectCheckSum > 0){
        cout<< "Stop\n";
        cout << objectCheckSum << endl;
        stop = true;
        system("echo 0 > /sys/class/gpio/gpio160/value");
    }
    else{
        cout<< "Go\n";
        cout << objectCheckSum << endl;
        stop = false;
        system("echo 1 > /sys/class/gpio/gpio160/value");
    }


}
```

The next function is called examineObject(). For arbitrary variables such as checkheight and checkwidth, the algorithm outputs a different matrix called objectCheck. This new matrix is significantly smaller than the original matrix, its main goal is to determine the areas where objects are located as well as filter out any random errors that could occur in the code. To do this the code starts at a point and counts the number of ones in an area around that point. The area scanned will be a square with the length of one side as ‘step’, which was 20 in our program. If the amount of ones is greater than 80 percent of the area scanned then the program will output a 1 to the objectCheck matrix at that point. It will then move ‘step’ number of pixels over and repeat the process.

The switch statement is to alter what areas of the object matrix are scanned to save on processing power. For example, if the ALTS is turning left then the left side of the image needs to be scanned, the right side isn't important because the ALTS will not travel there.

Next, the if and else statements at the bottom send out appropriate voltages for the pin. If there is an object, or objectCheckSum is greater than 0, then the code will send out a 0 voltage on the pin so that the throttle is off.

In other words, the code sends a command to the Arduino, telling the ALTS to stop because there is an object in front. On the other hand, if there are no objects, then the code will send out a 1, which is equivalent to 1.8V on the pin, so that the Arduino tells the ALTS to continue forward. On the terminal, these newly appended values will be displayed as either "Stop" or "Go".

The function printMatricesToFile() is used for testing purpose. This function prints out user defined matrices such as distanceBase, objectCheck, and object. The main purpose of this function is to fine tune the algorithms used for determining object: objectLocation() and examineObject().

3.6.3.6 Swift NAV GPS Module

For accurate positioning of the vehicle on the airport tarmac, a GPS module had to be incorporated into the proposed design. After research on multiple modules, the Swift Nav Piksi module (Figure 16) was chosen. This module is a local GPS system which will be setup relative to a point (the Command/Ground Station) in the airport. The integration of local GPS allows for accuracy up to centimeters. The board is open source so it allows for custom algorithm

programming using the Jetson TK1. The frequency of the module is 50 Hz so it does not interfere with any of the current frequencies at the airport, satisfying that requirement. The board also offers a gyroscopic sensor and accelerometer feedback. This will be used in parallel with the GPS positioning to ensure accuracy of the module itself. In the event that the GPS signal is lost through the transport tunnel for very short intervals, the accelerometer and gyroscope can be used as a secondary short range feedback control. See specifications sheet in Appendix D

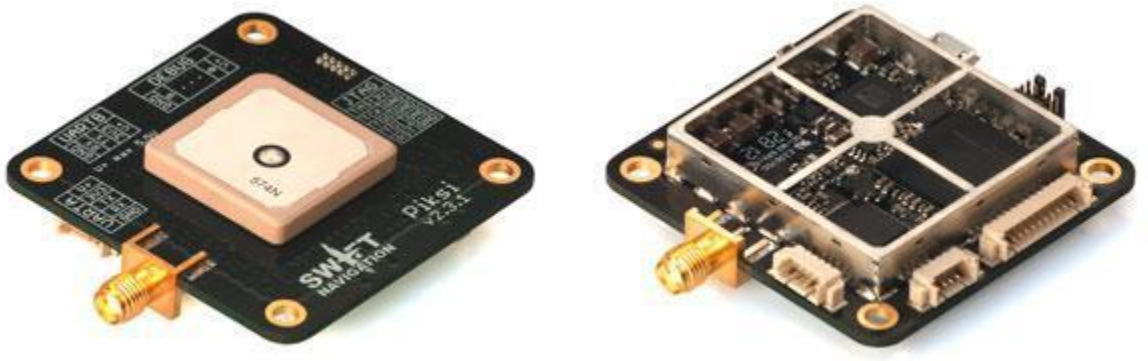


Figure 16: SwiftNAV GPS Module[6]

In addition to the Piksi module, a secondary GPS system, the Pixhawk, will be operating in parallel. The system connection diagram can be found in Appendix G. The Pixhawk system uses a general GPS and compass, as well as the Mission Planner software, in order to send signals to the control system of any navigational corrections. In this configuration the Piksi will still be providing the Pixhawk system with the 10 cm accuracy for improved navigation. The command/ground station will then process both the Piksi's live feed location, relative to the station's position, and the Mission Planner's plotted course. The Pixhawk control module is relayed all course adjustments over the 433MHz telemetry units and then sends all navigational

corrections to the Arduino for interfacing with mechanical systems. View the figure below for a sample path in the Mission Planner software.

Figure 17: Mission Planner Software

3.6.3.7 Encoder

An encoder must be included for feedback from the motor to the Jetson TK1. When a voltage is applied to the motor controller for the drive by wire, there must be feedback so that the Jetson knows what the RPM of the drive shaft is. As an example, if 12V is applied to the motor controller on day A, the motor may spin at 1000 RPM, if the same 12V is applied to the motor controller on day B where the humidity is higher the motor may spin at 900 RPM. To alleviate this the Jetson will instead be programmed with a proportional integral derivative algorithm (PID control algorithm) so that it has output of a specific RPM can adjust the voltage to match the required RPM. To ensure high accuracy in encoder readings, a minimum of a 360 point per revolution encoder will be used.

3.6.3.8 Sonar Sensors

Sonar sensors were chosen for short range object detection. Infrared sensors were researched but they do not meet FAA regulations for allowed frequencies on the tarmac. The HRLV-MaxSonar-EZ series of sonar sensors was looked at. The MB1003 was chosen from that series due to its wider beam pattern.

Figure 18: MaxSonar MB1003

The sonar sensors are wired using the digital PWM ports on the Arduino Leonardo for maximum accuracy in the readings. The sensor is accurate up to 600 centimeters. One of the key features in the sensor is its triggered operation. This allows the sensor to be in an idle state, taking minimal power and only triggering when values are called for it. The sensor has a small form factor, allowing it to be positioned anywhere on the vehicle. The optimal sensor placement requires using 4 sensors, two on the left and two on the right side, where on each side there is a sensor on the front and rear positioning. See specification sheet in Appendix E.

To test the accuracy of the sensors, experiments were completed to test whether or not the weather conditions affected the sensor readings. Based on the results of the experiment, sensor readings change dramatically in conditions of heavy rain. The sound waves emitted detect the

rain rather than an obstacle. Therefore, in rainy days the sonar sensors will be turned off and the ZED camera will be the only sensor detecting obstacles.

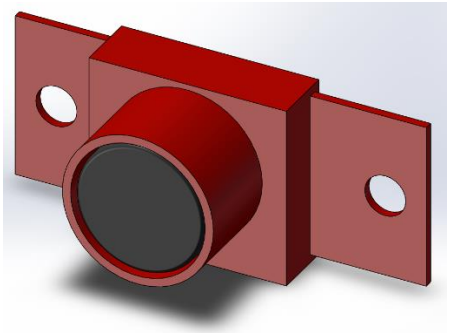


Figure 17: CAD Model of the Waterproof Case

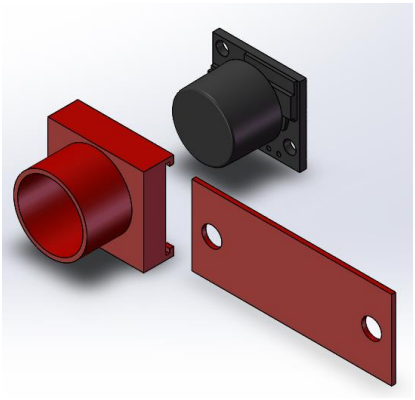


Figure 2018: CAD Model of the Waterproof Case (Exploded View)

Five sonar sensors are installed on the ALTS. One sensor is installed at the front of the ALTS; two sensors are installed on the right side and the other two on the left side. In order to protect the sensors from the water, a waterproof case is designed and 3D printed for each sensor. The 3D model is shown in Figure 17 and Figure 2018. In the figures, the black item represents the sonar sensor and red items are the case.

Implemented Electronics

Below is a figure of all the electronics implemented in the first version of the ALTS. The power rail is on the upper right hand side. The main battery rack charger is on the right hand

center side. The GPS sensors are located in the midle right above the hours of run time for the cart. The Arduino Leonardo is stacked above the Jetson TK1 to allow for proper ventilation on the far left of the figure. Then sensors are all being fed through the terminals on the upper left hand side with the motor speed controller right under those.

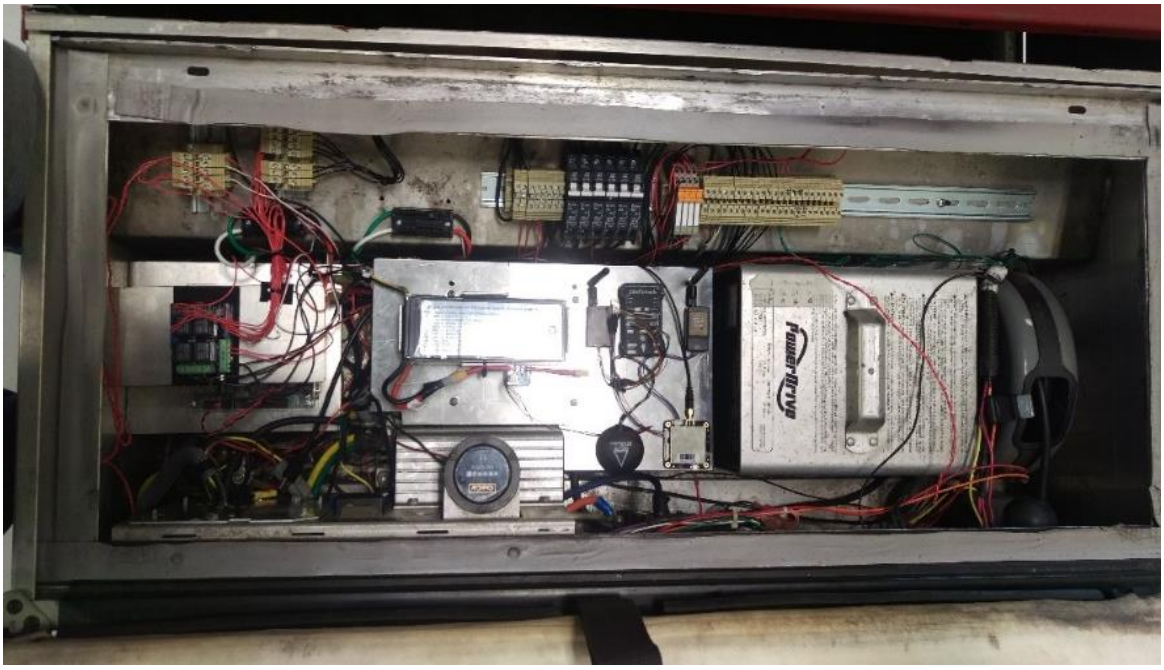


Figure 191: Overall Electronics Implementation

The figure below is the power rail for the cart. The left terminals are for the incoming power from the 12V, 12000 mAh battery. The terminals house the positive lead of the battery and the negative leads are going through the 4, 5 Amp breakers, and the 2, 20 Amp breakers. A gap is left between the 4 relays and the breakers to allow for wires to be fed. The left two relays are for the brake actuator and are wired in H-type.

Figure 202: Electronics Power Rail

The H-type wiring for the brake actuators is below:

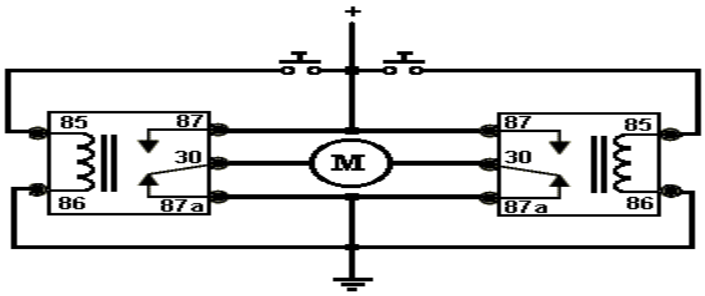


Figure 213: H-type wiring setup for double relay

Below is a figure of the GPS telemetry package equipped in the alts. The left hand size is the custom battery required for the pixhawk system. The telemetry antennas are included for the Piski GPS and the Pixhawk GPS unit. The PixHawk and GPS units are located in the centerline of the ALTS so that the proper GPS values are recorded in orientation of the front center of the vechicle.

Figure 224: Electronics GPS System

The Arduino Leonardo is hooked up with a 4 port relay. The wiring from the sensor area is going into this board as well as the outputs to the industrial relays and motor speed controllers. The Jetson TK1 is under the Arduino with approximately 5 inches of clearance above the fan. The boards are mounted on sturdy ¼ inch brackets so that the vibrations of the cart do not shake any of the wires loose.

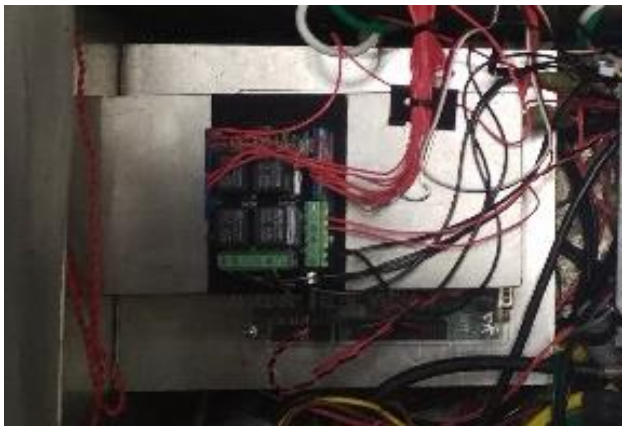


Figure 235: Arduino Leonardo & Jetson TK1 Installation

The figure below shows the terminals which all of the sensors of the ALTS come to where the go from to the Arduino and Jetson TK1.

Figure 246: Sensor Connection Rail

Chapter 4: CONCLUSIONS AND FUTURE WORK

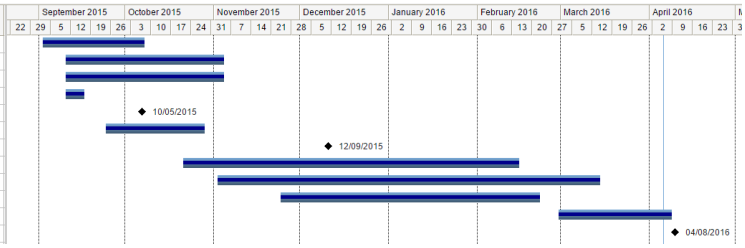
4.1 Conclusion

In general, the ALTS design process is successful. The modifications of the vehicle base and assembly of the mechanical control systems, specifically the brake, throttle, and steering are completed. Additionally, the computer board and electronic systems are properly coded and programmed to receive signals from all of its inputs. These signals are then sent as commands to the mechanical controls so that they work in unison. This allows the ALTS to drive autonomously to its necessary locations. Overall, if the ALTS were to be implemented into the Orlando-Melbourne International Airport, it would increase the time efficiency, and cost effectiveness of the airport as well as help save our environment. The ALTS could positively change the entire luggage system in airports across the world. More specific work to be done to the ALTS to increase its effectiveness, efficiency, and safety in the future are described in section 5.3.

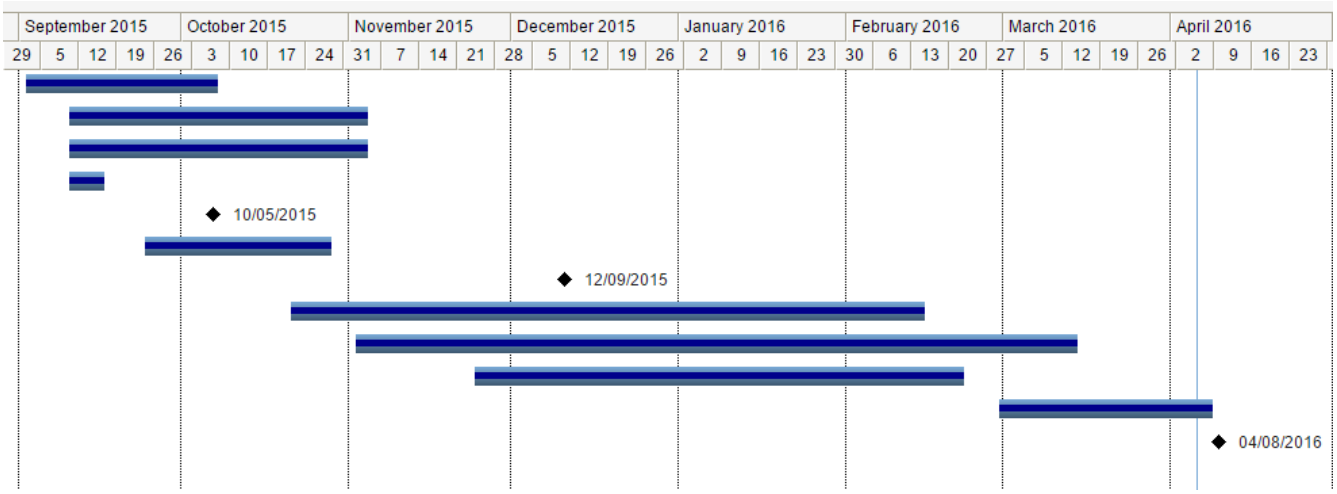
4.2 Timeline

Below is the overall timeline of the project.

	Name	Duration	Start	Finish
1	Background Research on MLB Airport	26d	09/02/2015	10/07/2015
2	Background Research on Autonomous Systems and Sensors	40d	09/10/2015	11/04/2015
3	Background Research on Renewable Energy Sources	40d	09/10/2015	11/04/2015
4	Create Requirements List	5d	09/10/2015	09/16/2015
5	MLB Airport Presentation	1d	10/05/2015	10/05/2015
6	Vehicle Power/Energy and Return on Investment Calculations	25d	09/24/2015	10/28/2015
7	Mid-Year PDR and Final Presentation	1d	12/09/2015	12/09/2015
8	Order Parts and Purchase Vehicle Base	84d	10/21/2015	02/15/2016
9	Programming and Sensor Testing	96d	11/02/2015	03/14/2016
10	Fabrication and Modifications to the Vehicle	65d	11/24/2015	02/22/2016
11	Testing and Bug Fixing	30d	02/29/2016	04/08/2016
12	Senior Design Showcase and Final Report Due	1d	04/08/2016	04/08/2016



Name	Duration	Start	Finish
Background Research on MLB Airport	26d	09/02/2015	10/07/2015
Background Research on Autonomous Systems and Sensors	40d	09/10/2015	11/04/2015
Background Research on Renewable Energy Sources	40d	09/10/2015	11/04/2015
Create Requirements List	5d	09/10/2015	09/16/2015
MLB Airport Presentation	1d	10/05/2015	10/05/2015
Vehicle Power/Energy and Return on Investment Calculations	25d	09/24/2015	10/28/2015
Order Parts and Purchase Vehicle Base	68d	10/21/2015	01/22/2016
Programming and Sensor Testing	91d	11/02/2015	03/07/2016
Fabrication and Modifications to the Vehicle	60d	11/16/2015	02/05/2016
Mid-Year PDR and Final Presentation	1d	12/09/2015	12/09/2015
Testing and Bug Fixing	30d	02/29/2016	04/08/2016
Senior Design Showcase and Final Report Due	1d	04/15/2016	04/15/2016



4.3 Future Work

If the ALTS is adopted by the Orlando-Melbourne International Airport, there are several suggestions that the ALTS team has to offer, assuming a budget greater than $10,000 and more time to invest in development in this project. The ALTS team suggests that there be multiple, or a fleet, of ALTS for the airport to have the overall most effective and efficient transportation system for MLB. Another suggestion is to make the ALTS more time efficient by attaching

additional luggage carts for the ALTS to pull behind it to increase the amount of luggage carried in each trip.

Regarding the physical systems, if the ALTS is invested in by MLB, the ALTS team suggests instead of modifying used golf carts, that there be a custom manufactured vehicle base specifically for the ALTS. To supplement this suggestion, specifically a higher quality rack and pinion mechanism should be installed along with a more sensitive straight line adjustment. These two suggestions will assist in making the ALTS have a more accurate straight line reading for steering adjustments, as well as narrow the range of its error in steering (decreasing "play" in the steering column.)

Taking into consideration the electrical components, the ALTS should also have a custom electrical braking actuator for quicker and more sensitive braking. This improvement will not only help the ALTS become more effective but this will also improve the overall safety of the entire ALTS system. Also implementing in a more efficient control module than the Jetson TK1. Although the Jetson TK1 is the best computer board available as of now, a custom computer should be made specifically for the ALTS and simplify its communications between subsystems of the ALTS.

For additional safety, the ALTS team suggests that with more time and budget, obstacle avoidance and maneuvering should be researched and installed. As of now the ZED camera signal allows the ALTS to come to a stop when detection of an object occurs; however, if the ALTS could redirect its route and avoid the obstacle without having to come to a stop, this would be more time efficient for the MLBs overall luggage transportation system, as well as making the ALTS even more safe.

Appendix A: Nvidia Jetson TK1 Specifications Sheet

The **NVIDIA Jetson TK1 developer kit** gives you everything you need to unlock the power of the GPU for embedded systems applications.

It's built around the revolutionary NVIDIA Tegra® K1 SoC and uses the same NVIDIA Kepler™ computing core designed into supercomputers around the world. This gives you a fully functional NVIDIA CUDA® platform for quickly developing and deploying compute-intensive systems for computer vision, robotics, medicine, and more.

NVIDIA delivers the entire BSP and software stack, including CUDA, OpenGL 4.4, and Tegra-accelerated OpenCV. With a complete suite of development and profiling tools, plus out-of-the-box support for cameras and other peripherals, NVIDIA gives you the ideal solution for helping shape the future of embedded on our Jetson Embedded Portal.

BUY NOW

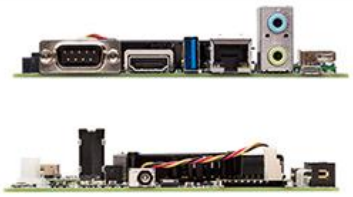
Related Links:
Jetson Embedded Portal for Developers
NVIDIA Embedded Forum
Ecosystem Partners

KIT CONTENTS

- Tegra K1 SOC
  - NVIDIA Kepler GPU with 192 CUDA Cores
  - NVIDIA 4-Plus-1™ Quad-Core ARM® Cortex™-A15 CPU
- 2 GB x16 Memory with 64-bit Width
- 16 GB 4.51 eMMC Memory
- 1 Half Mini-PCIE Slot
- 1 Full-Size SD/MMC Connector
- 1 Full-Size HDMI Port
- 1 USB 2.0 Port, Micro AB
- 1 USB 3.0 Port, A
- 1 RS232 Serial Port
- 1 ALC5639 Realtek Audio Codec with Mic In and Line Out
- 1 RTL8111GS Realtek GigE LAN
- 1 SATA Data Port
- SPI 4 MByte Boot Flash

The following signals are available through an expansion port:

- DP/LVDS
- Touch SPI 1x4 + 1x1 CSI-2
- GPIOs
- UART
- HSIC
- i2c



Appendix B: Arduino Due Specifications Sheet

Technical specs

AVR Arduino microcontroller

Microcontroller	AT91SAM3X8E
Operating Voltage	3.3V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-16V
Digital I/O Pins	54 (of which 12 provide PWM output)
Analog Input Pins	12
Analog Output Pins	2 (DAC)
Total DC Output Current on all I/O lines	130 mA
DC Current for 3.3V Pin	800 mA
DC Current for 5V Pin	800 mA
Flash Memory	512 KB all available for the user applications
SRAM	96 KB (two banks: 64KB and 32KB)
Clock Speed	84 MHz
Length	101.52 mm
Width	53.3 mm
Weight	36 g

Appendix C: Zed Camera Specifications Sheet

Description

The ZED Stereo Camera is a lightweight depth sensor based on passive stereovision. It outputs a high resolution side-by-side video on USB 3.0 that contains two synchronized left and right video streams. Using the ZED SDK, the graphics processing unit (GPU) from a host machine computes the depth map from the side-by-side video in real-time.

Video

Video Mode	Frames per second	Output Resolution (side by side)
2.2K	15	4416x1242
1080p	30	3840x1080
720p	60	2560x720
VGA	120	1280x480

Depth

Depth Resolution
Same as selected video resolution (with ZED SDK)

Depth Format
32-bits
OpenCV compatible

Depth Range
1.5 - 20 m (3.5 to 68 ft)

Camera Baseline
120 mm (4.7")

Lens

- Wide-angle all-glass lens with reduced distortion
- Field of View: 110° (D) max.
- *f*/2.0 aperture

Sensors

Sensor Resolution
4M pixels per sensor

Sensor Format
Native 16:9 Format
for a greater horizontal field of view

Sensor Size
1/2.7" backside illumination sensors

Shutter Sync
Electronic Synchronized Rolling Shutter

Camera Controls
Adjust Resolution, Frame Rate, Brightness, Contrast, Saturation, Gamma, Sharpness and White Balance

ISP Sync
Synchronized Auto Exposure, Auto White Balance

Connector

- USB 3.0 port
- Integrated 2 m cable

Mounting

- Mount the camera by using the attached mini tripod or use the 1/4"-20 UNC thread mount

Power

- Power via USB
- 5V / 380mA

Size and Weight

Dimensions
175 x 30 x 33 mm (6.89 x 1.18 x 1.3")

Weight
159 g (0.35 lb)

Compatible OS

- Windows 7, Windows 8

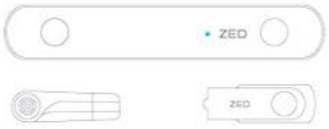
- Linux

SDK System Requirements

- Dual-core 2,3GHz or faster processor
- 4 GB RAM or more
- Nvidia GPU with Compute Capability > 2.0
- CUDA 6.5
- USB 3.0 port

In The Box

- ZED Stereo camera
- Mini Tripod stand
- USB Drive with Drivers and SDK
- Documentation



Appendix D: Swift NAV Piski GPS Specifications Sheet

Piksi Datasheet

Flexible, high-performance GPS receiver platform running open-source software

Features

- Centimeter accurate relative positioning (Carrier phase RTK)
- 50 Hz position/velocity/time solutions
- Open-source software and board design
- Low power consumption - 500mW typical
- Small form factor - 53x53mm
- USB and dual UART connectivity
- Integrated patch antenna and external antenna input
- Full-rate raw sample pass-through over USB
- 3-bit, 16.368 MS/s L1 front-end supports GPS, GLONASS, Galileo and SBAS signals

Applications

- Autonomous vehicle guidance, e.g. UAV formation flight and autonomous landing
- GPS/GNSS research
- Surveying systems

Overview

Piksi is a low-cost, high-performance GPS receiver with Real Time Kinematics (RTK) functionality for centimeter level relative positioning accuracy.

Its small form factor, fast position solution update rate, and low power consumption make Piksi ideal for integration into autonomous vehicles and portable surveying equipment.

Piksi's open source firmware allows it to be easily customized to the particular demands of end users' applications, easing system integration and reducing host system overhead.

In addition, Piksi's use of the same open source GNSS libraries as Peregrine, Swift Navigation's GNSS post-processing software, make the combination of the two a powerful toolset for GNSS research, experimentation, and prototyping at every level from raw samples to position solutions.

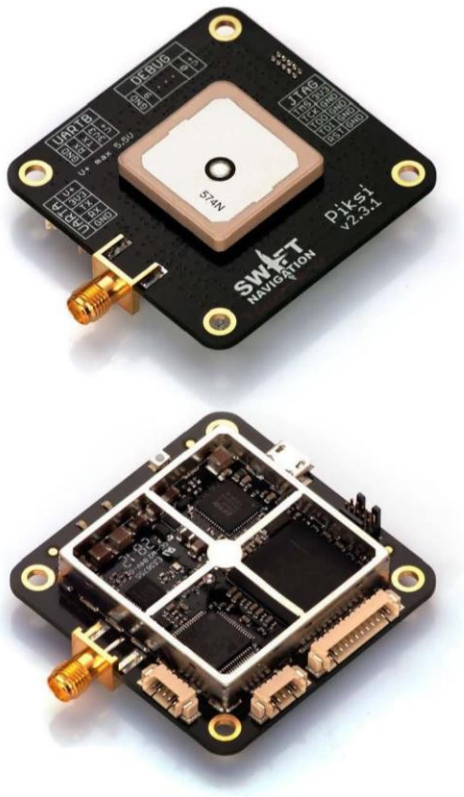


Figure 1: Piksi front and back view

With these tools developers can quickly move from prototyping software on a desktop to running it standalone on the Piksi hardware.

A high-performance DSP on-board and our flexible Swift-NAP correlation accelerator provide Piksi with ample computing resources with which advanced receiver techniques, such as multipath mitigation, spoofing detection, and carrier phase tracking can be implemented.

System Architecture

The Piksi receiver architecture consists of three main components. The RF front-end downconverts and digitizes the radio frequency signal from the antenna. The digitized signal is passed into the SwiftNAP which performs basic filtering and correlation operations on the signal stream. The SwiftNAP is controlled by a microcontroller which programs the correlation operations, collects the results, and processes them all the way to position/velocity/time (PVT) solutions.

Front-end

The RF front-end consists of a Maxim MAX2769 integrated down-converter and 3-bit analog-to-digital converter operating at up to 32.736 MS/s (16.368 MS/s typical). This front-end is capable of covering the L1 GPS, GLONASS, Galileo and SBAS signal bands.

The input to the front-end can be switched between an on-board patch antenna or an external active antenna.

SwiftNAP

The SwiftNAP consists of a Xilinx Spartan-6 FPGA that comes pre-programmed with Swift Navigation's SwiftNAP firmware. The SwiftNAP contains correlators specialized for satellite signal tracking and acquisition, as well as programmable digital notch filters for CW noise nulling. The correlators are flexible and fully programmable via a high-speed SPI register interface and are used as simple building blocks for implementing tracking loops and acquisition algorithms on the microcontroller.

Whilst the SwiftNAP HDL is not open-source at this time, the Piksi has no restrictions against loading one's own firmware onto the on-board Spartan-6 FPGA. More detail on the SwiftNAP is available from the SwiftNAP datasheet[1].

Microcontroller

The on-board microcontroller is an STM32F4 with an ARM Cortex-M4 DSP core running at up to 168 MHz. This powerful processor performs all functions above the correlator level including tracking loop filters, acquisition management and navigation processing and is able to calculate PVT solutions at over 50 Hz in our default software configuration. All software running on the microcontroller is supplied open-source.

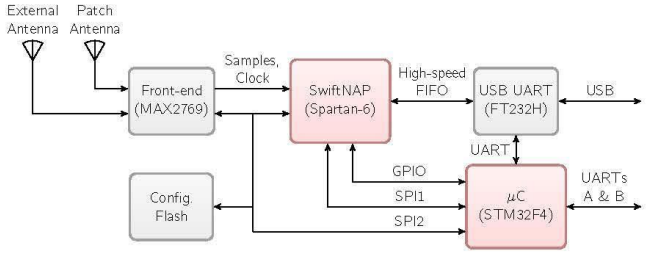


Figure 2: Piksi Block Diagram

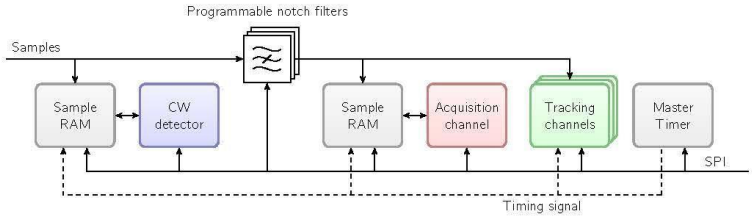
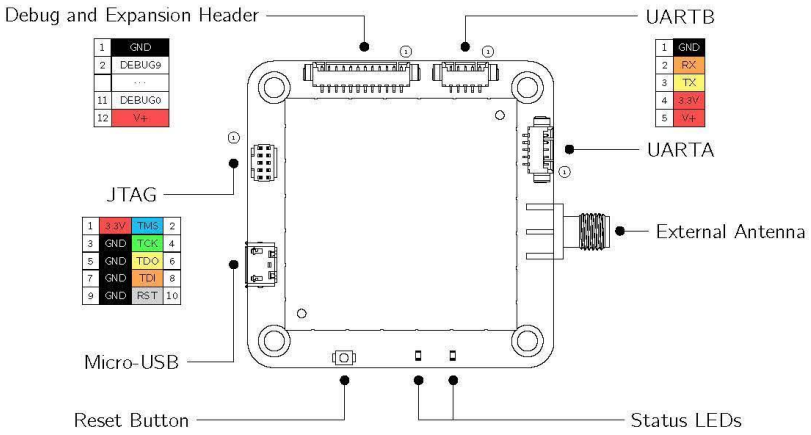


Figure 3: SwiftNAP Block Diagram

[1] Link to SwiftNAP datasheet

Connections



USB

A Micro-USB socket provides USB connectivity to the host. This can be configured as a USB-Serial bridge to the microcontroller (the default) or as a high-speed FIFO interface to the SwiftNAP for streaming full-rate raw IF data samples to or from the host.

This allows capture of raw IF data for processing on the host or running the Piksi from pre-recorded data or simulator output for hardware-in-the-loop testing.

UARTs A & B

Two UARTs provide high-speed 3.3V LVTTL level asynchronous serial interfaces which can be configured to transmit NMEA-0183 messages or binary navigation solution data, system status and debugging information and receive commands or differential corrections from the host or another Piksi board.

When configured in USB-Serial bridge mode, the USB interface functions identically to the two dedicated UARTs.

External Antenna

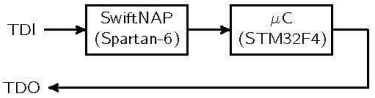
In addition to the on-board patch antenna, an external active antenna input is provided on an SMA connector and features a software switchable 3.3V bias.

Selection between the on-board patch and the external antenna is made in software or can be set to automatically detect the presence of an external active antenna and switch accordingly.

JTAG

No JTAG adapter is required to develop for the Piksi as the board is supplied with a built-in bootloader.

For advanced debugging, a 0.05" pitch micro JTAG header compatible with the ARM Cortex-M standard pinout[2] is provided on the board. This allows access to the Spartan-6 FPGA and STM32F4 microcontroller JTAG interfaces.



Debug and Expansion Header

Access is provided to debugging signals from the SwiftNAP and I/O for future expansion boards and accessories. Assignment of these signals varies depending on the SwiftNAP firmware configuration.

Power

Power may be supplied to the board either over USB or through the V+ pins on the UART connectors. A 3.3V output from the on-board switching regulator is provided to power any external peripherals.

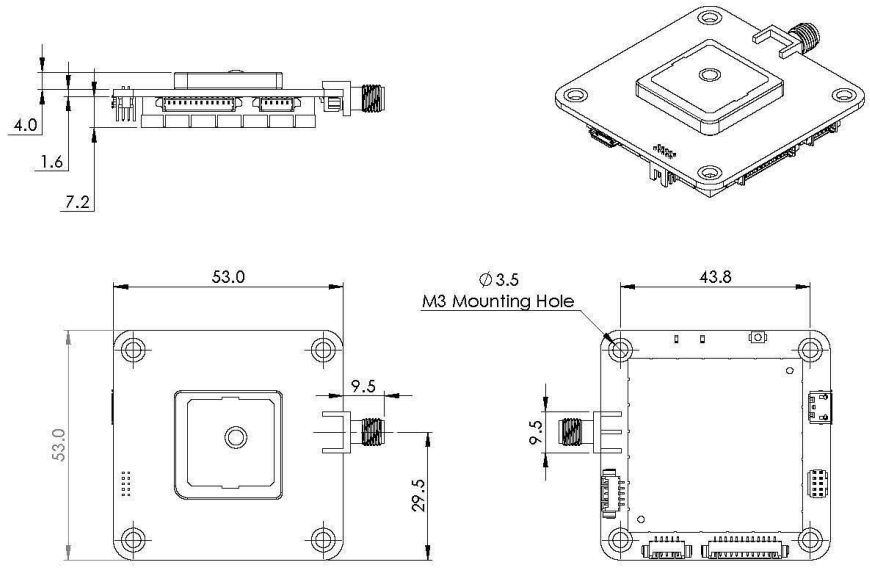
[2] ARM Cortex-M Debug Connector specification, http://bit.ly/ICb6W6

Electrical Specifications

Supply voltage	3.5 – 5.5 V	Active antenna input impedance	50 Ω
Power consumption	500 mW[1]	Active antenna bias voltage	3.3 V[2]
Max. 3.3V output current draw	500 mA	Max. antenna bias current draw	57 mA

[1] Typical, dependant on firmware configuration.
[2] Switchable in software

Mechanical Drawing



All dimensions are in millimeters. Drawing not to scale.

Notes

1. Mass 32g.
2. M3 mounting holes are plated through and connected internally to ground.
3. 3D CAD models are available from our website, http://www.swift-nav.com.
4. If the on-board patch antenna is used, care should be taken to mount the Piksi such that the antenna has as clear and unobstructed view of the sky as possible. No conductive objects should be placed close to the antenna.

Appendix E: LV-MaxSonar-EZ MB1000 Specifications Sheet

LV-MaxSonar®-EZ™ Series

High Performance Sonar Range Finder

MB1000, MB1010, MB1020, MB1030, MB1040

With 2.5V - 5.5V power the LV-MaxSonar-EZ provides very short to long-range detection and ranging in a very small package. The LV-MaxSonar-EZ detects objects from 0-inches to 254-inches (6.45-meters) and provides sonar range information from 6-inches out to 254-inches with 1-inch resolution. Objects from 0-inches to 6-inches typically range as 6-inches[1]. The interface output formats included are pulse width output, analog voltage output, and RS232 serial output. Factory calibration and testing is completed with a flat object. ***[1]See Close Range Operation***

Features

- Continuously variable gain for control and side lobe suppression
- Object detection to zero range objects
- 2.5V to 5.5V supply with 2mA typical current draw
- Readings can occur up to every 50mS, (20-Hz rate)
- Free run operation can continually measure and output range information
- Triggered operation provides the range reading as desired
- Interfaces are active simultaneously
- Serial, 0 to Vcc, 9600 Baud, 81N
- Analog, (Vcc/512) / inch
- Pulse width, (147uS/inch)
- Learns ringdown pattern when commanded to start ranging
- Designed for protected indoor environments
- Sensor operates at 42KHz
- High output square wave sensor drive (double Vcc)

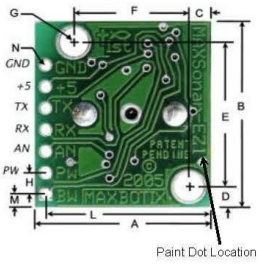
Benefits

- Very low cost ultrasonic rangefinder
- Reliable and stable range data
- Quality beam characteristics
- Mounting holes provided on the circuit board
- Very low power ranger, excellent for multiple sensor or battery-based systems
- Fast measurement cycles
- Sensor reports the range reading directly and frees up user processor
- Choose one of three sensor outputs
- Triggered externally or internally

Applications and Uses

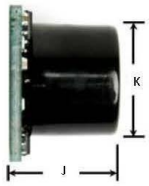
- UAV blimps, micro planes and some helicopters
- Bin level measurement
- Proximity zone detection
- People detection
- Robot ranging sensor
- Autonomous navigation
- Multi-sensor arrays
- Distance measuring
- Long range object detection
- Wide beam sensitivity

LV-MaxSonar-EZ Mechanical Dimensions



A	0.785"	19.9 mm	H	0.100"	2.54 mm
B	0.870"	22.1 mm	J	0.610"	15.5 mm
C	0.100"	2.54 mm	K	0.645"	16.4 mm
D	0.100"	2.54 mm	L	0.735"	18.7 mm
E	0.670"	17.0 mm	M	0.065"	1.7 mm
F	0.510"	12.6 mm	N	0.038" dia.	1.0 mm dia.
G	0.124" dia.	3.1 mm dia.	weight, 4.3 grams		

Part Number	MB1000	MB1010	MB1020	MB1030	MB1040
Paint Dot Color	Black	Brown	Red	Orange	Yellow



Close Range Operation

Applications requiring 100% reading-to-reading reliability should not use MaxSonar sensors at a distance closer than 6 inches. Although most users find MaxSonar sensors to work reliably from 0 to 6 inches for detecting objects in many applications, MaxBotix® Inc. does not guarantee operational reliability for objects closer than the minimum reported distance. Because of ultrasonic physics, these sensors are unable to achieve 100% reliability at close distances.

Warning: Personal Safety Applications

We do not recommend or endorse this product be used as a component in any personal safety applications. This product is not designed, intended or authorized for such use. These sensors and controls do not include the self-checking redundant circuitry needed for such use. Such unauthorized use may create a failure of the MaxBotix® Inc. product which may result in personal injury or death. MaxBotix® Inc. will not be held liable for unauthorized use of this component.

About Ultrasonic Sensors

Our ultrasonic sensors are in air, non-contact object detection and ranging sensors that detect objects within an area. These sensors are not affected by the color or other visual characteristics of the detected object. Ultrasonic sensors use high frequency sound to detect and localize objects in a variety of environments. Ultrasonic sensors measure the time of flight for sound that has been transmitted to and reflected back from nearby objects. Based upon the time of flight, the sensor then outputs a range reading.

Pin Out Description

Pin 1-BW-*Leave open or hold low for serial output on the TX output. When BW pin is held high the TX output sends a pulse (instead of serial data), suitable for low noise chaining.

Pin 2-PW- This pin outputs a pulse width representation of range. The distance can be calculated using the scale factor of 147uS per inch.

Pin 3-AN- Outputs analog voltage with a scaling factor of (Vcc/512) per inch. A supply of 5V yields ~9.8mV/in. and 3.3V yields ~6.4mV/in. The output is buffered and corresponds to the most recent range data.

Pin 4-RX– This pin is internally pulled high. The LV-MaxSonar-EZ will continually measure range and output if RX data is left unconnected or held high. If held low the sensor will stop ranging. Bring high for 20uS or more to command a range reading.

Pin 5-TX- When the *BW is open or held low, the TX output delivers asynchronous serial with an RS232 format, except voltages are 0-Vcc. The output is an ASCII capital "R", followed by three ASCII character digits representing the range in inches up to a maximum of 255, followed by a carriage return (ASCII 13). The baud rate is 9600, 8 bits, no parity, with one stop bit. Although the voltage of 0-Vcc is outside the RS232 standard, most RS232 devices have sufficient margin to read 0-Vcc serial data. If standard voltage level RS232 is desired, invert, and connect an RS232 converter such as a MAX232. When BW pin is held high the TX output sends a single pulse, suitable for low noise chaining. (no serial data)

Pin 6-+5V- Vcc – Operates on 2.5V - 5.5V. Recommended current capability of 3mA for 5V, and 2mA for 3V.

Pin 7-GND- Return for the DC power supply. GND (& Vcc) must be ripple and noise free for best operation.

Range "0" Location

The LV-MaxSonar-EZ reports the range to distant targets starting from the front of the sensor as shown in the diagram below.

Range Zero

The range is measured from the front of the transducer.

In general, the LV-MaxSonar-EZ will report the range to the leading edge of the closest detectable object. Target detection has been characterized in the sensor beam patterns.

Sensor Minimum Distance

The sensor minimum reported distance is 6-inches (15.2 cm). However, the LV-MaxSonar-EZ will range and report targets to the front sensor face. Large targets closer than 6-inches will typically range as 6-inches.

Sensor Operation from 6-inches to 20-inches

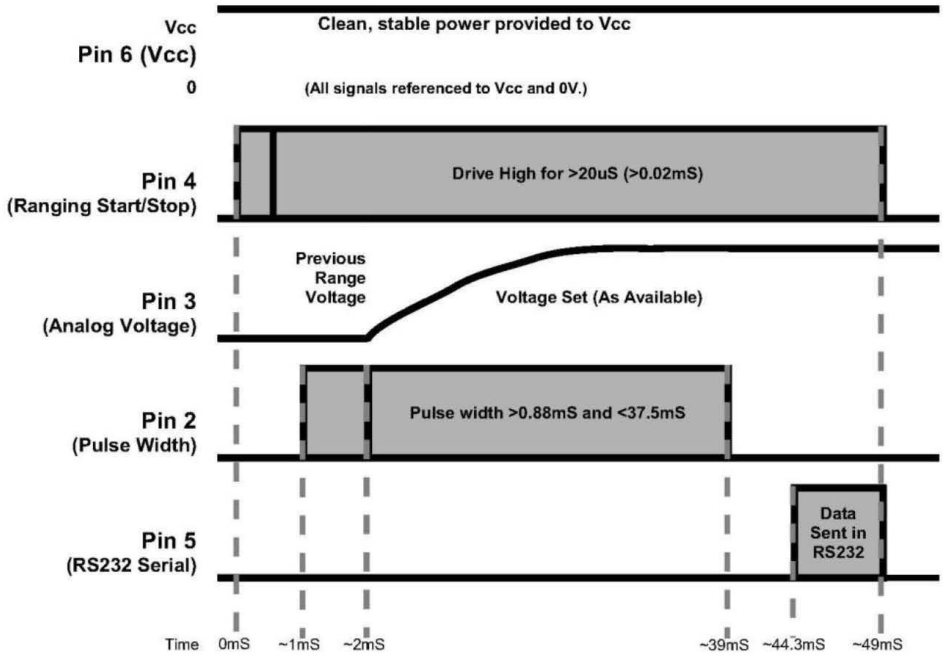
Because of acoustic phase effects in the near field, objects between 6-inches and 20-inches may experience acoustic phase cancellation of the returning waveform resulting in inaccuracies of up to 2-inches. These effects become less prevalent as the target distance increases, and has not been observed past 20-inches.

General Power-Up Instruction

Each time the LV-MaxSonar-EZ is powered up, it will calibrate during its first read cycle. The sensor uses this stored information to range a close object. It is important that objects not be close to the sensor during this calibration cycle. The best sensitivity is obtained when the detection area is clear for fourteen inches, but good results are common when clear for at least seven inches. If an object is too close during the calibration cycle, the sensor may ignore objects at that distance.

The LV-MaxSonar-EZ does not use the calibration data to temperature compensate for range, but instead to compensate for the sensor ringdown pattern. If the temperature, humidity, or applied voltage changes during operation, the sensor may require recalibration to reacquire the ringdown pattern. Unless recalibrated, if the temperature increases, the sensor is more likely to have false close readings. If the temperature decreases, the sensor is more likely to have reduced up close sensitivity. To recalibrate the LV-MaxSonar-EZ, cycle power, then command a read cycle.

Timing Diagram



Timing Description

250mS after power-up, the LV-MaxSonar-EZ is ready to accept the RX command. If the RX pin is left open or held high, the sensor will first run a calibration cycle (49mS), and then it will take a range reading (49mS). After the power up delay, the first reading will take an additional ~100mS. Subsequent readings will take 49mS. The LV-MaxSonar-EZ checks the RX pin at the end of every cycle. Range data can be acquired once every 49mS.

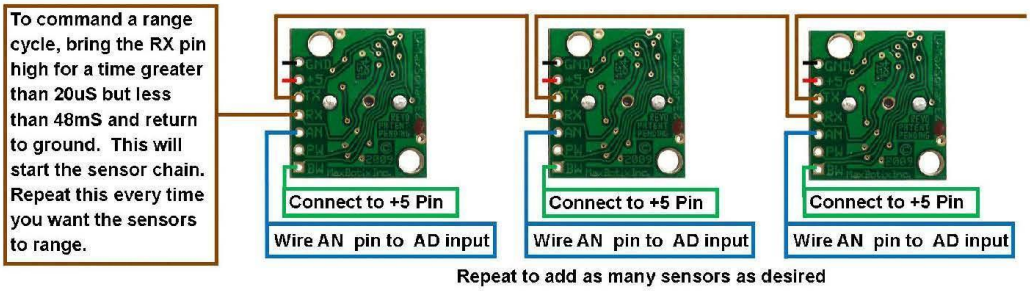
Each 49mS period starts by the RX being high or open, after which the LV-MaxSonar-EZ sends the transmit burst, after which the pulse width pin (PW) is set high. When a target is detected the PW pin is pulled low. The PW pin is high for up to 37.5mS if no target is detected. The remainder of the 49mS time (less 4.7mS) is spent adjusting the analog voltage to the correct level. When a long distance is measured immediately after a short distance reading, the analog voltage may not reach the exact level within one read cycle. During the last 4.7mS, the serial data is sent.

The LV-MaxSonar-EZ timing is factory calibrated to one percent at five volts, and in use is better than two percent. In addition, operation at 3.3V typically causes the objects range, to be reported, one to two percent further than actual.

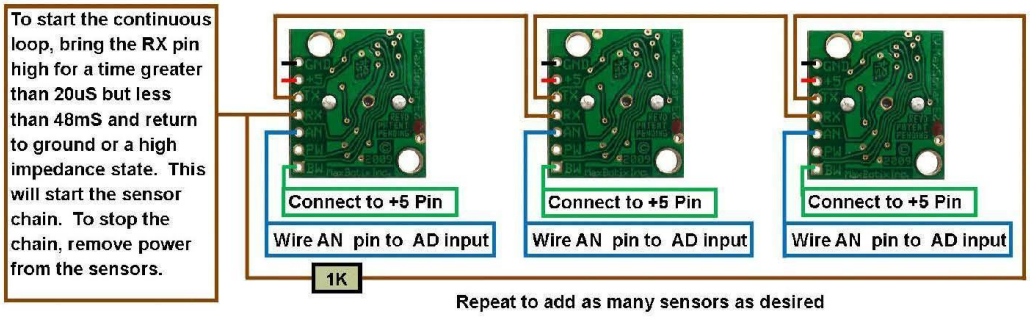
Using Multiple Sensors in a single system

When using multiple ultrasonic sensors in a single system, there can be interference (cross-talk) from the other sensors. MaxBotix Inc., has engineered and supplied a solution to this problem for the LV-MaxSonar-EZ sensors. The solution is referred to as chaining. We have 3 methods of chaining that work well to avoid the issue of cross-talk.

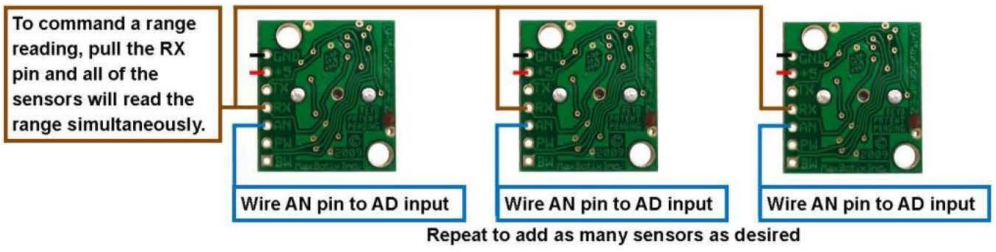
The first method is AN Output Commanded Loop. The first sensor will range, then trigger the next sensor to range and so on for all the sensor in the array. Once the last sensor has ranged, the array stops until the first sensor is triggered to range again. Below is a diagram on how to set this up.



The next method is AN Output Constantly Looping. The first sensor will range, then trigger the next sensor to range and so on for all the sensor in the array. Once the last sensor has ranged, it will trigger the first sensor in the array to range again and will continue this loop indefinitely. Below is a diagram on how to set this up.



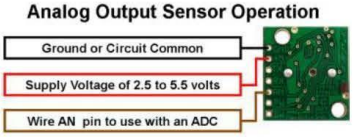
The final method is AN Output Simultaneous Operation. This method does not work in all applications and is sensitive to how the other sensors in the array are positioned in comparison to each other. Testing is recommend to verify this method will work for your application. All the sensors RX pins are conned together and triggered at the same time causing all the sensor to take a range reading at the same time. Once the range reading is complete, the sensors stop ranging until triggered next time. Below is a diagram on how to set this up.



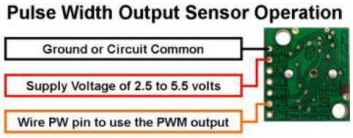
Independent Sensor Operation

The LV-MaxSonar-EZ sensors have the capability to operate independently when the user desires. When using the LV-MaxSonar-EZ sensors in single or independent sensor operation, it is easiest to allow the sensor to free-run. Free-run is the default mode of operation for all of the MaxBotix Inc., sensors. The LV-MaxSonar-EZ sensors have three separate outputs that update the range data simultaneously: Analog Voltage, Pulse Width, and RS232 Serial. Below are diagrams on how to connect the sensor for each of the three outputs when operating in a single or independent sensor operating environment.

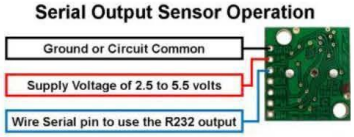
Analog Output Sensor Operation



Pulse Width Output Sensor Operation



Serial Output Sensor Operation



Selecting an LV-MaxSonar-EZ

Different applications require different sensors. The LV-MaxSonar-EZ product line offers varied sensitivity to allow you to select the best sensor to meet your needs.

The LV-MaxSonar-EZ Sensors At a Glance

People Detection Wide Beam High Sensitivity		Best Balance		Large Targets Narrow Beam Noise Tolerance
MB1000	MB1010	MB1020	MB1030	MB1040

The diagram above shows how each product balances sensitivity and noise tolerance. This does not effect the maximum range, pin outputs, or other operations of the sensor. To view how each sensor will function to different sized targets reference the LV-MaxSonar-EZ Beam Patterns.

Background Information Regarding our Beam Patterns

Each LV-MaxSonar-EZ sensor has a calibrated beam pattern. Each sensor is matched to provide the approximate detection pattern shown in this datasheet. This allows end users to select the part number that matches their given sensing application. Each part number has a consistent field of detection so additional units of the same part number will have similar beam patterns. The beam plots are provided to help identify an estimated detection zone for an application based on the acoustic properties of a target versus the plotted beam patterns.

People Sensing: For users that desire to detect people, the detection area to the 1-inch diameter dowel, in general, represents the area that the sensor will reliably detect people.

Each beam pattern is a 2D representation of the detection area of the sensor. The beam pattern is actually shaped like a 3D cone (having the same detection pattern both vertically and horizontally). Detection patterns for dowels are used to show the beam pattern of each sensor. Dowels are long cylindered targets of a given diameter. The dowels provide consistent target detection characteristics for a given size target which allows easy comparison of one MaxSonar sensor to another MaxSonar sensor.

For each part number, the four patterns (A, B, C, and D) represent the detection zone for a given target size. Each beam pattern shown is determined by the sensor's part number and target size.

The actual beam angle changes over the full range. Use the beam pattern for a specific target at any given distance to calculate the beam angle for that target at the specific distance. Generally, smaller targets are detected over a narrower beam angle and a shorter distance. Larger targets are detected over a wider beam angle and a longer range.

MB1000 LV-MaxSonar-EZ0

The LV-MaxSonar-EZ0 is the highest sensitivity and widest beam sensor of the LV-MaxSonar-EZ sensor series. The wide beam makes this sensor ideal for a variety of applications including people detection, autonomous navigation, and wide beam applications.

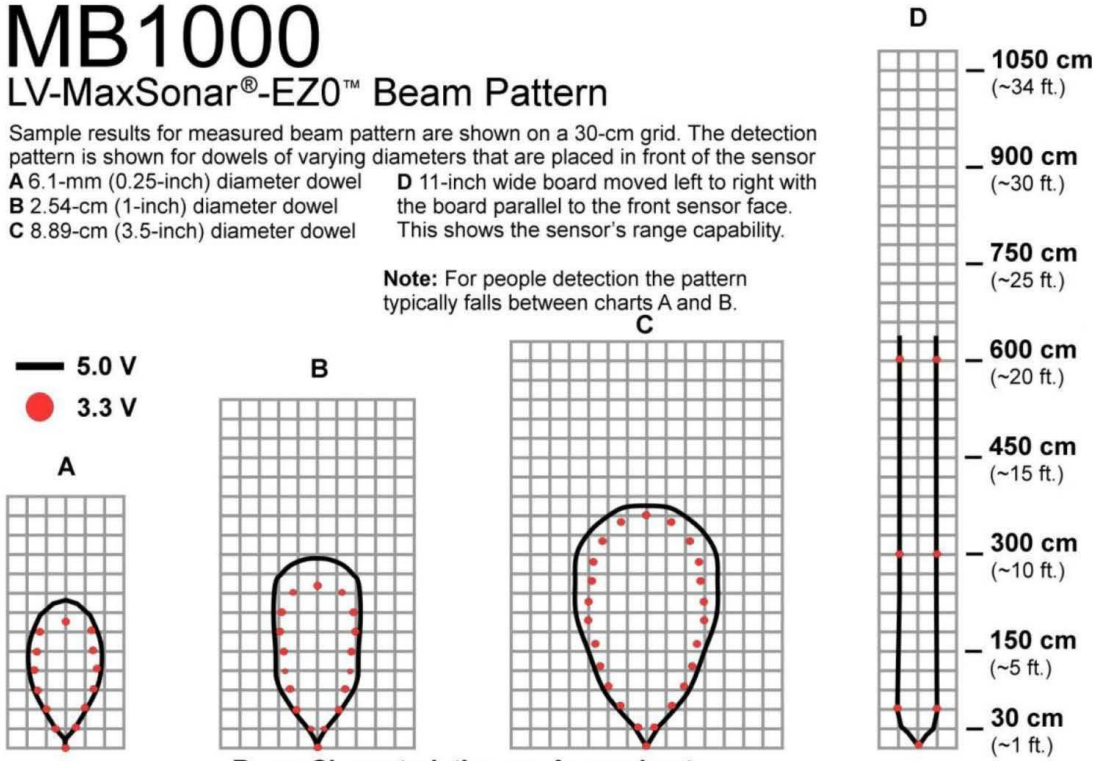
MB1000

LV-MaxSonar®-EZ0™ Beam Pattern

Sample results for measured beam pattern are shown on a 30-cm grid. The detection pattern is shown for dowels of varying diameters that are placed in front of the sensor

A 6.1-mm (0.25-inch) diameter dowel
B 2.54-cm (1-inch) diameter dowel
C 8.89-cm (3.5-inch) diameter dowel
D 11-inch wide board moved left to right with the board parallel to the front sensor face. This shows the sensor's range capability.

Note: For people detection the pattern typically falls between charts A and B.



Beam Characteristics are Approximate

Beam Pattern drawn to a 1:95 scale for easy comparison to our other products.

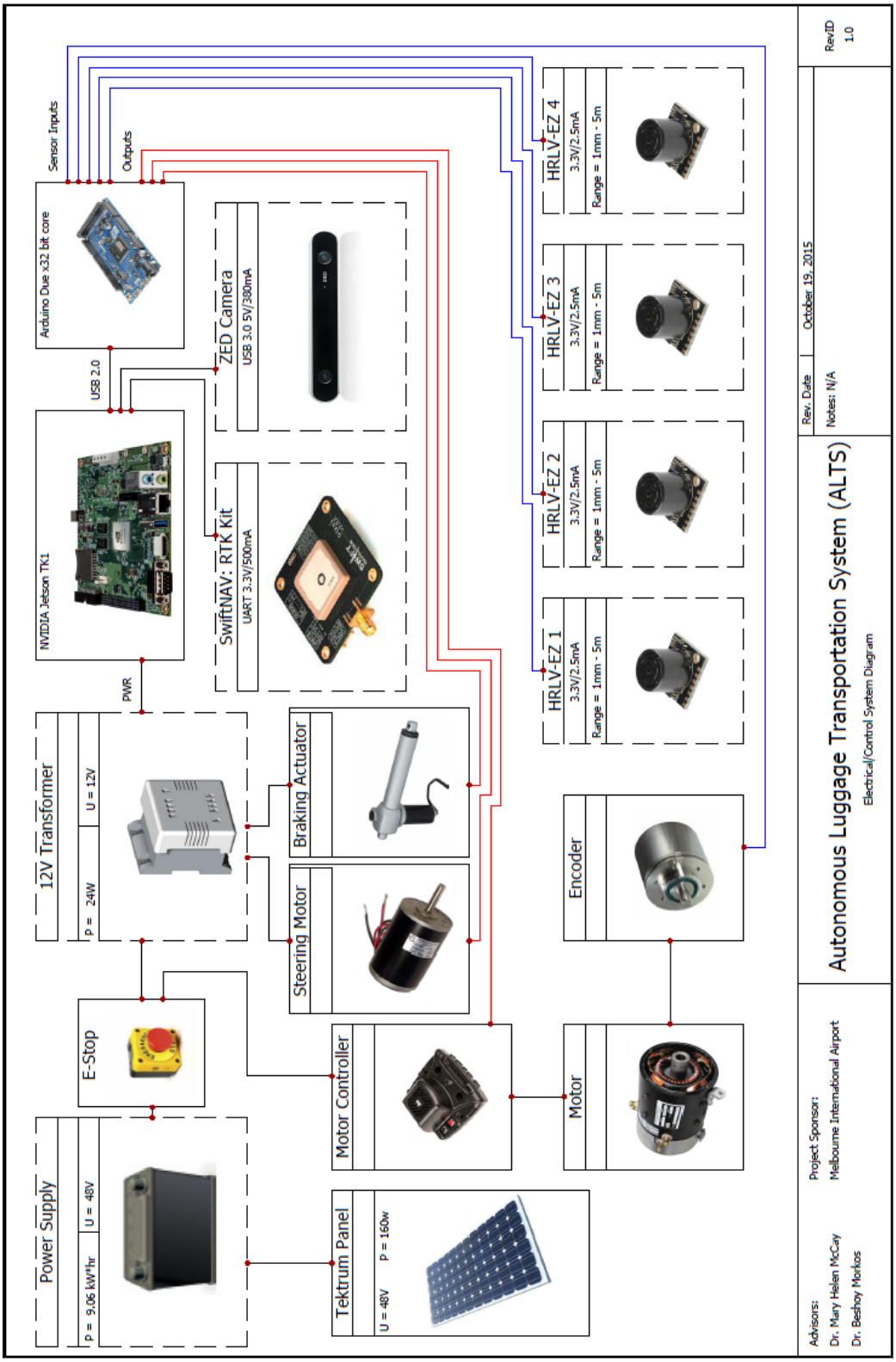
MB1000 Features and Benefits

- Widest and most sensitive beam pattern in LV-MaxSonar-EZ line
- Low power consumption
- Easy to use interface
- Will pick up the most noise clutter when compared to other sensors in the LV-MaxSonar-EZ line
- Detects smaller objects
- Best sensor to detect soft object in LV-MaxSonar-EZ line
- Requires use of less sensors to cover a given area
- Can be powered by many different types of power sources
- Can detect people up to approximately 10 feet

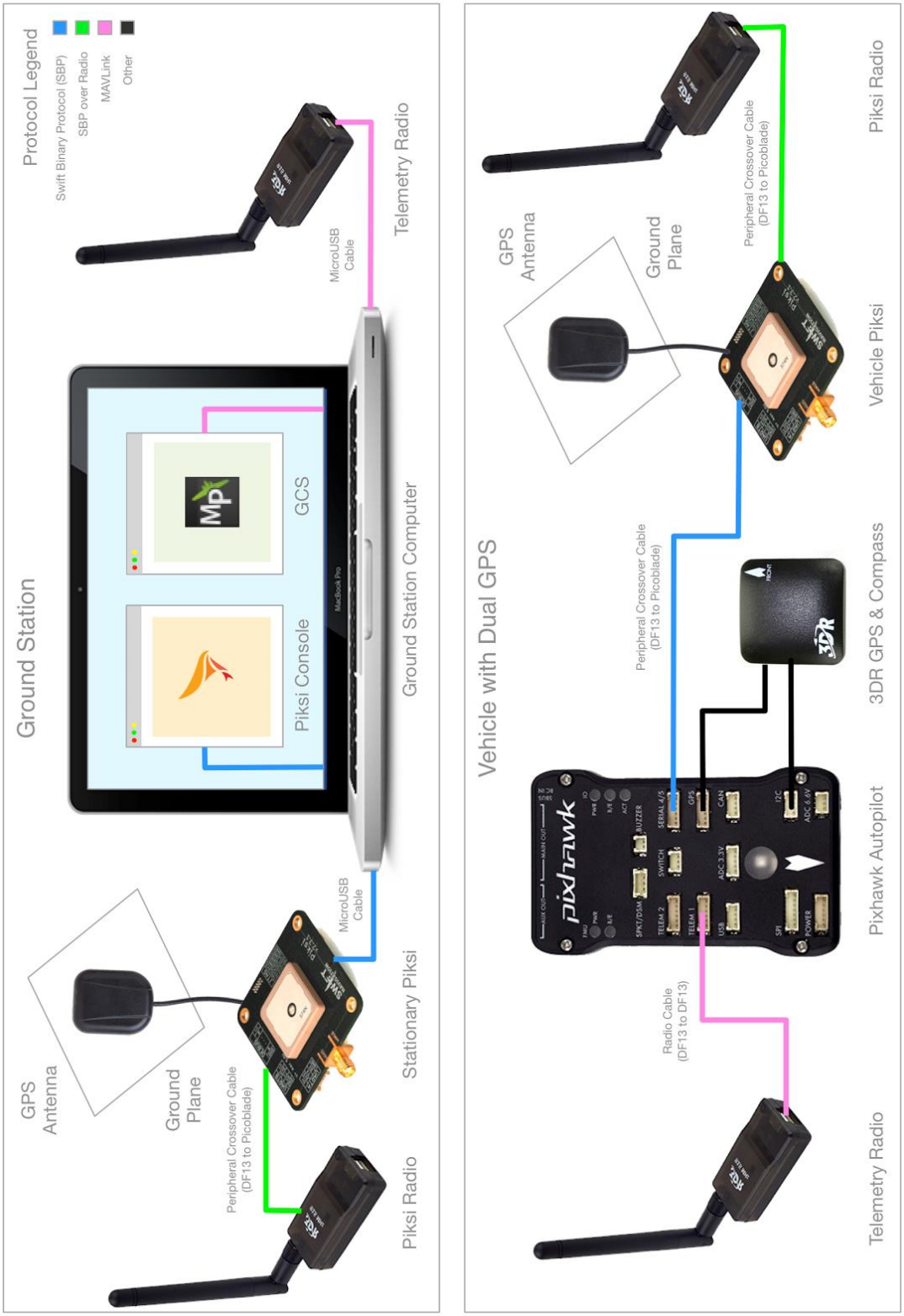
MB1000 Applications and Uses

- Great for people detection
- Security
- Motion detection
- Used with battery power
- Autonomous navigation
- Educational and hobby robotics
- Collision avoidance

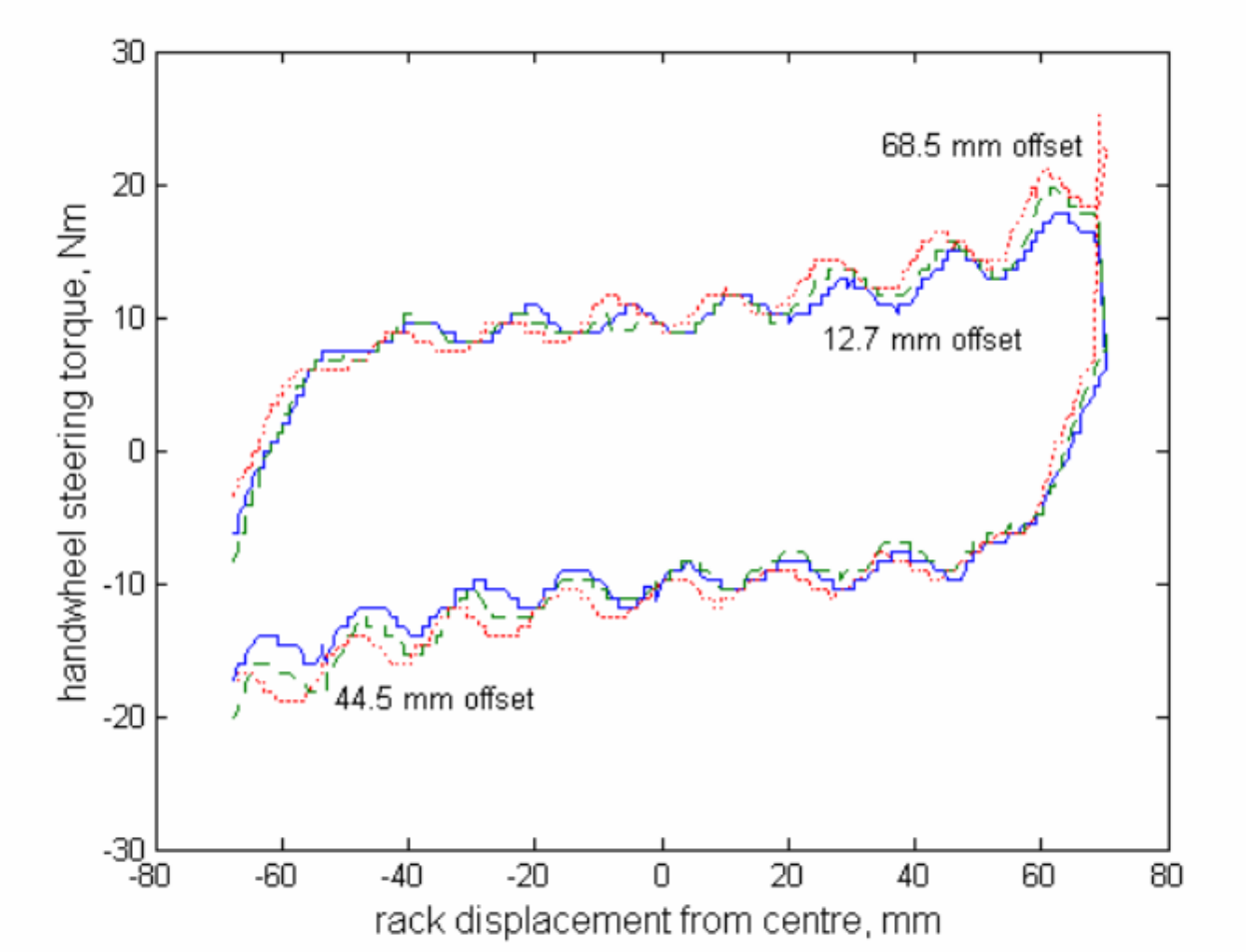
Appendix F: Electrical System Diagram



Appendix G: GPS System Diagram



Appendix G: Steering Torque vs. Rack Displacement[3]



Appendix H: MLB Airport Schedules

Melbourne International Airport

SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
		1 10:34am Delta 1856 1:12pm AA 5132 1:30pm Delta 2213 5:07pm Delta 742 7:23pm AA 5068 10:47pm Delta 1756 11:52pm AA 5141	2 10:34am Delta 1856 1:12pm AA 5132 1:30pm Delta 2213 5:07pm Delta 742 7:23pm AA 5068 10:47pm Delta 1756 11:52pm AA 5141	3 10:34am Delta 1856 1:12pm AA 5132 1:30pm Delta 2213 5:07pm Delta 742 7:23pm AA 5068 10:47pm Delta 1756 11:52pm AA 5141	4 10:34am Delta 1856 1:12pm AA 5132 1:30pm Delta 2213 5:07pm Delta 742 7:23pm AA 5068 10:47pm Delta 1756 11:52pm AA 5141	5 1:28pm Delta 2213 2:37pm AA 5132 08:19pm AA 5141 10:45pm Delta 1756
6 1:12pm AA 5132 1:28pm Delta 2213 5:05pm Delta 742 7:23pm AA 5068 10:47pm Delta 1756 11:50pm Elite Airways 11:52pm AA 5141	LABOR DAY 7 10:34am Delta 1856 1:12pm AA 5132 1:30pm Delta 2213 5:07pm Delta 742 7:23pm AA 5068 10:47pm Delta 1756 11:52pm AA 5141	8 10:34am Delta 1856 1:12pm AA 5132 5:07pm Delta 742 7:23pm AA 5068 10:47pm Delta 1756 11:52pm AA 5141	9 1:16pm AA 5132 1:30pm Delta 2213 7:24pm AA 5068 10:47pm Delta 1756 11:55pm AA 5141	10 1:16pm AA 5132 1:30pm Delta 2213 5:07pm Delta 742 7:24pm AA 5068 10:47pm Delta 1756 11:55pm AA 5141	11 1:16pm AA 5132 1:3pm Delta 2213 5:07pm Delta 742 7:24pm AA 5068 10:47pm Delta 1756 11:55pm AA 5141	12 1:05pm AA 5132 1:28pm Delta 2213 7:49pm AA 5068 10:45pm Delta 1756
13 10:34am Delta 1856 1:16pm AA 5132 1:30pm Delta 2213 5:07pm Delta 742 7:24pm AA 5068 10:47pm Delta 1756 11:55pm AA 5141	14 10:34am Delta 1856 1:16pm AA 5132 1:30pm Delta 2213 5:07pm Delta 742 7:24pm AA 5068 10:47pm Delta 1756 11:55pm AA 5141	15 10:34am Delta 1856 1:16pm AA 5132 1:30pm Delta 2213 5:07pm Delta 742 7:24pm AA 5068 10:47pm Delta 1756	16 10:34am Delta 1856 1:16pm AA 5132 1:30pm Delta 2213 5:07pm Delta 742 7:24pm AA 5068 10:47pm Delta 1756 11:55pm AA 5141	17 10:34am Delta 1856 1:16pm AA 5132 1:30pm Delta 2213 5:07pm Delta 742 7:24pm AA 5068 10:47pm Delta 1756 11:55pm AA 5141	18 10:34am Delta 1856 1:16pm AA 5132 1:30pm Delta 2213 5:07pm Delta 742 7:24pm AA 5068 10:47pm Delta 1756 11:55pm AA 5141	19 10:32am Delta 1856 2:41pm AA 5132 1:28pm Delta 2213 07:24pm AA 5068 10:45pm Delta 1756
20 10:34am Delta 1856 1:16pm AA 5132 1:30pm Delta 2213 5:07pm Delta 742 7:24pm AA 5068 10:47pm Delta 1756 11:55pm AA 5141	21 10:34am Delta 1856 1:16pm AA 5132 1:30pm Delta 2213 5:07pm Delta 742 7:24pm AA 5068 10:47pm Delta 1756 11:55pm AA 5141	22 10:34am Delta 1856 1:16pm AA 5132 1:30pm Delta 2213 5:07pm Delta 742 7:24pm AA 5068 10:47pm Delta 1756	23 10:34am Delta 1856 1:16pm AA 5132 1:30pm Delta 2213 5:07pm Delta 742 7:24pm AA 5068 10:47pm Delta 1756 11:55pm AA 5141	24 10:34am Delta 1856 1:16pm AA 5132 1:30pm Delta 2213 5:07pm Delta 742 7:24pm AA 5068 10:47pm Delta 1756 11:55pm AA 5141	25 10:34am Delta 1856 1:16pm AA 5132 1:30pm Delta 2213 5:07pm Delta 742 7:24pm AA 5068 10:47pm Delta 1756 11:55pm AA 5141	26 10:32am Delta 1856 2:41pm AA 5132 1:28pm Delta 2213 7:24pm AA 5068 10:45pm Delta 1756
27 10:34am Delta 1856 1:16pm AA 5132 1:30pm Delta 2213 5:07pm Delta 742 7:24pm AA 5068 10:47pm Delta 1756 11:55pm AA 5141	28 10:34am Delta 1856 1:16pm AA 5132 1:30pm Delta 2213 5:07pm Delta 742 7:24pm AA 5068 10:47pm Delta 1756 11:55pm AA 5141	29 10:34am Delta 1856 1:16pm AA 5132 1:30pm Delta 2213 5:07pm Delta 742 7:24pm AA 5068 10:47pm Delta 1756	30 10:34am Delta 1856 1:16pm AA 5132 1:30pm Delta 2213 5:07pm Delta 742 7:24pm AA 5068 10:47pm Delta 1756 11:55pm AA 5141	September 2015 ARRIVALS		

Figure 257: MLB September 2015 schedule for arriving flights

Melbourne International Airport

SUNDAY	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY
		1 06:20am Delta 793 08:15am AA 5175 11:15am Delta 1856 1:45pm AA 5132 2:12pm Delta 2213 5:50pm Delta 742 7:50pm AA 5068	2 06:25am Delta 793 08:15am AA 5175 11:15am Delta 1856 1:45pm AA 5132 2:12pm Delta 2213 5:50pm Delta 742 7:50pm AA 5068	3 06:25am Delta 793 08:15am AA 5175 11:15am Delta 1856 1:45pm AA 5132 2:12pm Delta 2213 5:50pm Delta 742 7:50pm AA 5068	4 06:25am Delta 793 08:15am AA 5175 09:00am Elite Airways 11:15am Delta 1856 1:45pm AA 5132 2:12pm Delta 2213 5:50pm Delta 742 7:50pm AA 5068	5 06:25am Delta 793 08:15am AA 5364 2:12pm Delta 2213 3:05pm AA 5132
6 10:15am AA 5175 11:15am Delta 2213 1:45pm AA 5104 2:12pm Delta 2213 5:50pm Delta 742 7:50pm AA 5068	LABOR DAY 7 06:25am Delta 793 08:15am AA 5175 11:15am Delta 1856 1:45pm AA 5132 2:12pm Delta 2213 5:50pm Delta 742 7:50pm AA 5068	8 06:25am Delta 793 08:15am AA 5175 11:15am Delta 1856 1:45pm AA 5132 2:12pm Delta 2213 5:50pm Delta 742 7:50pm AA 5068	9 06:25am Delta 793 08:10am AA 5175 1:45pm AA 5132 2:12pm Delta 2213 7:50pm AA 5068	10 06:25am Delta 793 08:10am AA 5175 1:45pm AA 5132 2:12pm Delta 2213 5:50pm Delta 742 7:50pm AA 5068	11 06:25am Delta 793 08:10am AA 5175 1:45pm AA 5132 2:12pm Delta 2213 5:50pm Delta 742 7:50pm AA 5068	12 06:25am Delta 793 08:10am AA 5364 2:12pm Delta 2213 3:10pm AA 5132
13 06:25am Delta 793 10:15am AA 5175 11:15am Delta 1856 1:45pm AA 5104 2:12pm Delta 2213 5:50pm Delta 742 7:50pm AA 5068	14 06:25am Delta 793 08:10am AA 5175 11:15am Delta 1856 1:45pm AA 5132 2:12pm Delta 2213 5:50pm Delta 742 7:50pm AA 5068	15 06:25am Delta 793 08:10am AA 5175 11:15am Delta 1856 1:45pm AA 5132 2:12pm Delta 2213 5:50pm Delta 742	16 06:25am Delta 793 08:10am AA 5175 11:15am Delta 1856 1:45pm AA 5132 2:12pm Delta 2213 5:50pm Delta 742 7:50pm AA 5068	17 06:25am Delta 193 08:10am AA 5175 11:15am Delta 1856 1:45pm AA 5132 2:12pm Delta 2213 5:50pm Delta 742 7:50pm AA 5068	18 06:25am Delta 193 08:15am AA 5175 11:15am Delta 1856 1:45pm AA 5132 2:12pm Delta 2213 5:50pm Delta 742 7:55pm AA 5068	19 06:25am Delta 793 08:10am AA 5364 11:15am Delta 1856 3:10pm AA 5132 2:12pm Delta 2213
20 06:25am Delta 793 10:15am AA 5175 11:15am Delta 1856 1:45pm AA 5132 2:12pm Delta 2213 5:50pm Delta 742 7:50pm AA 5068	21 06:25am Delta 793 08:10am AA 5175 11:15am Delta 1856 1:45pm AA 5132 2:12pm Delta 2213 5:50pm Delta 742 7:50pm AA 5068	22 06:25am Delta 793 08:10am AA 5175 11:15am Delta 1856 1:45pm AA 5132 2:12pm Delta 2213 5:50pm Delta 742	23 06:25am Delta 793 08:10am AA 5175 11:15am Delta 1856 1:45pm AA 5132 2:12pm Delta 2213 5:50pm Delta 742 7:50pm AA 5068	24 06:25am Delta 193 08:10am AA 5175 11:15am Delta 1856 1:45pm AA 5132 2:12pm Delta 2213 5:50pm Delta 742 7:50pm AA 5068	25 06:25am Delta 193 08:10am AA 5175 11:15am Delta 1856 1:45pm AA 5132 2:12pm Delta 2213 5:50pm Delta 742 7:50pm AA 5068	26 06:25am Delta 793 08:10am AA 5364 11:15am Delta 1856 3:10pm AA 5132 2:12pm Delta 1412
27 06:25am Delta 793 10:15am AA 5175 11:15am Delta 1856 1:45pm AA 5132 2:12pm Delta 2213 5:50pm Delta 742 7:50pm AA 5068	28 06:25am Delta 793 08:10am AA 5175 11:15am Delta 1856 1:45pm AA 5132 2:12pm Delta 2213 5:50pm Delta 742 7:50pm AA 5068	29 06:25am Delta 793 08:10am AA 5175 11:15am Delta 1856 1:45pm AA 5132 2:12pm Delta 2213 5:50pm Delta 742	30 06:25am Delta 793 08:10am AA 5175 11:15am Delta 1856 1:45pm AA 5132 2:12pm Delta 2213 5:50pm Delta 742 7:50pm AA 5068	September 2015 DEPARTURES		

Figure 268: MLB September 2015 schedule for departing flights

Appendix I: Evaluation of GPS on aircraft by NASA

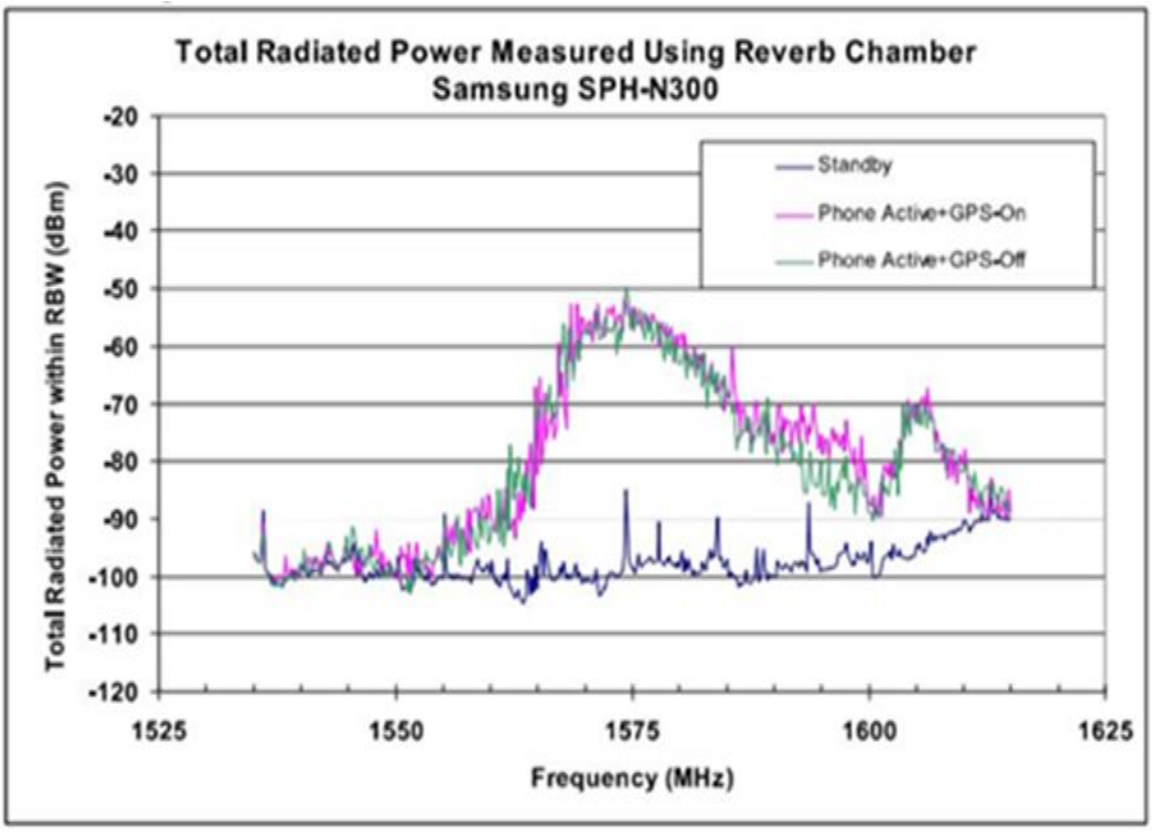


Figure 279: Evaluation of GPS on aircraft by NASA

Appendix J: OSHA Laws and Regulations

Occupational Safety and Health Administration Laws and Regulations

- All vehicles shall have a service brake system, an emergency brake system, and a parking brake system. These systems may use common components, and shall be maintained in operable condition.
- Whenever visibility conditions warrant additional light, all vehicles, or combinations of vehicles, in use shall be equipped with at least two headlights and two taillights in operable condition.
- All vehicles, or combination of vehicles, shall have brake lights in operable condition regardless of light conditions.
- All vehicles shall be equipped with an adequate audible warning device at the operator's station and in an operable condition.
- No employer shall use any motor vehicle equipment having an obstructed view to the rear unless:
- The vehicle has a reverse signal alarm audible above the surrounding noise level or:
- The vehicle is backed up only when an observer signals that it is safe to do so.
- All vehicles with cabs shall be equipped with windshields and powered wipers. Cracked and broken glass shall be replaced. Vehicles operating in areas or under conditions that cause fogging or frosting of the windshields shall be equipped with operable defogging or defrosting devices.
- All haulage vehicles, whose pay load is loaded by means of cranes, power shovels, loaders, or similar equipment, shall have a cab shield and/or canopy adequate to protect the operator from shifting or falling materials.
- Tools and material shall be secured to prevent movement when transported in the same compartment with employees.
- Vehicles used to transport employees shall have seats firmly secured and adequate for the number of employees to be carried.
- Seat belts and anchorages meeting the requirements of 49 CFR Part 571 (Department of Transportation, Federal Motor Vehicle Safety Standards) shall be installed in all motor vehicles.

- Trucks with dump bodies shall be equipped with positive means of support, permanently attached, and capable of being locked in position to prevent accidental lowering of the body while maintenance or inspection work is being done.
- Operating levers controlling hoisting or dumping devices on haulage bodies shall be equipped with a latch or other device which will prevent accidental starting or tripping of the mechanism.
- Trip handles for tailgates of dump trucks shall be so arranged that, in dumping, the operator will be in the clear.
- All rubber-tired motor vehicle equipment manufactured on or after May 1, 1972, shall be equipped with fenders. All rubber-tired motor vehicle equipment manufactured before May 1, 1972, shall be equipped with fenders not later than May 1, 1973.
- Mud flaps may be used in lieu of fenders whenever motor vehicle equipment is not designed for fenders.
- All vehicles in use shall be checked at the beginning of each shift to assure that the following parts, equipment, and accessories are in safe operating condition and free of apparent damage that could cause failure while in use: service brakes, including trailer brake connections; parking system (hand brake); emergency stopping system (brakes); tires; horn; steering mechanism; coupling devices; seat belts; operating controls; and safety devices. All defects shall be corrected before the vehicle is placed in service. These requirements also apply to equipment such as lights, reflectors, windshield wipers, defrosters, fire extinguishers, etc., where such equipment is necessary

Appendix K: FAA Regulations

Federal Aviation Administration Rules and Regulations

- A minimum 8 inch wide horizontal band of high gloss white paint or white reflective tape must be used around the vehicle's surface to improve night time recognition.
- The standard for identification lighting is a yellow flashing light that is mounted on the uppermost part of the vehicle structure. A steady yellow light designates vehicles limited to non-movement areas.
    - The light must be visible from any direction, day and night, including from the air.
    - Lights must have peak intensity within the range of 40 to 400 candelas (effective) from 0° (horizontal) up to 10° above the horizontal and for 360° horizontally. The upper limit of 400 candelas (effective) is necessary to avoid damage to night vision.
    - From 10° to 15° above the horizontal plane, the light output must be 1/10th of peak intensity or between 4 and 40 candelas (effective).
    - Lights must flash at 75 ± 15 flashes per minute.

Appendix L: MLB Rules and Regulations

Orlando-Melbourne International Airport Rules and Regulations:

- Taxiing aircraft and aircraft under tow have the right-of-way over all ground vehicles.
- Ground vehicles must yield to aircraft during engine startups.
- Vehicles responding to an emergency (e.g. ARFF, MLB Operations, and MAPD vehicles) have the right-of-way.
- Vehicles cannot pass between an aircraft and its intended parking stand.
- No person shall drive a vehicle at MLB at a speed greater than is reasonable and prudent under the existing conditions, while having regard for the existing actual and potential hazards. The Terminal ramp speed limit is 15 miles per hour and Service Road speed is 25 miles per hour.
- Vehicles should not be parked under passenger loading bridges or under an aircraft (unless the vehicle is servicing the loading bridge).
- Vehicles should not be parked in a manner that interferes with access to fire hydrants or fire extinguishers.
- Vehicles should not be parked under passenger loading bridges or under an aircraft (unless the vehicle is servicing the loading bridge).
- No vehicle shall be permitted on MLB unless it is in sound mechanical order. Vehicles used at night must have adequate headlights and taillights.
- Maintenance of vehicles and equipment at Terminal gate positions or adjacent apron/ramp areas is prohibited unless authorized by the Director.
- All motor vehicles on the AOA shall yield the right-of-way to aircraft in motion under all conditions.

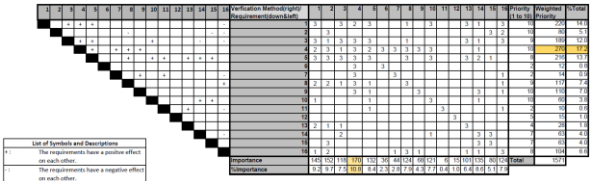
Appendix M: Requirements List

Florida Institute of Technology
High Tech with a Human Touch

Project Name [1]	Autonomous Luggage Transportation System (ALTS)
Date	9/15/2015

Req. No. [2]	Requirement [3]	Constraint/ Criteria [4]	Source [5]	Justification [6]	Date of elicitation [7]	Verification method [8]	Update (if any) [9]	Associated ECN(s)
1	Able to fuction autonomously.	Criteria	MLB Airport	Project requirement from MLB Airport.	9/15/2015	By implementation of an autonomous system with multiple sensors for detection and avoidance.		
2	Able to carry 10 bags at one time.	Criteria	MLB Airport	Project requirement from MLB Airport.	9/15/2015	The ALTS must be able to support the weight of 10 bags (estimated maxium supported weight is 750lbs)		
3	Able to move to/from various locations, also can reverse.	Criteria	MLB Airport	Project requirement from MLB Airport.	9/15/2015	The ALTS is mobile and travel destinations can be adjusted. It can also safely move in reverse.		
4	Will not impact day to day operations at the airport and also follows FAA Rules & Regulation for vehicle operation.	Criteria	MLB Airport and FAA Rules & Regulations	Project requirement from MLB Airport and requirement by FAA.	9/15/2015	Follows all airport ground traffic rules and FAA ground traffic rules.		
5	Able to function in an airport simulated type setting.	Criteria	MLB Airport	Project requirement from MLB Airport.	9/15/2015	Will be able to manuver around an airport without incedent.		
6	A minimum 8 inch wide horizontal band of high gloss white paint or white reflective tape must be used around the vehicle's surface.	Criteria	FAA Rules & Regulations	Required by FAA.	9/15/2015	Includes the reflective band on the surface of the vehicle.		
7	A steady yellow flashing light is required to be mounted on the vehicle's uppermost structure.	Criteria	FAA Rules & Regulations	Required by FAA.	9/15/2015	The ALTS includes a steady flashing light, which follows FAA rules.		
8	Vehicle must not exceed 15 mph on the terminal ramp.	Criteria	FAA Rules & Regulations	Required by FAA.	9/15/2014	Limit the max speed of the cart to 15 mph.		
9	Vehicles at night must have adequate headlights and taillights. (At least two head and tail lights for visibility with operable brake lights.)	Criteria	FAA Rules & Regulations and OSHA Laws & Regulations	Required by FAA and OSHA.	9/15/2015	The ALTS includes headlights and taillights that meet regulations.		
10	Normal and parking brake system.	Criteria	OSHA Laws & Regulations	Required by OSHA	9/15/2015	The ALTS has a braking system and a parking brake.		
11	Working audible warning device for reversing and other operations.	Criteria	OSHA Laws & Regulations	Required by OSHA	9/15/2015	The ALTS inculdes audible warning device.		
12	Vehicles with tires must have rubber fenders.	Criteria	OSHA Laws & Regulations	Required by OSHA	9/15/2015	The ALTS has a rubber fender.		
13	Cannot be operated by anyone unless there is a clear view of the rear.	Criteria	OSHA Laws & Regulations	Required by OSHA	9/15/2015	Includes a senor on the rear of the vehicle.		
14	Must be able to function in varying weather conditions as seen in Florida.	Constraint	ALTS Team	Florida has a variety of changing weather and temperature conditions, which may effect performance.	9/15/2015	The ALTS can opperate in rain/fog/lightning and other accliment weather experienced in Florida, while Airport is active. (A severe weather shutdown does not require activity.)		
15	Must be able to transport bags safely and without damage.	Constraint	ALTS Team	Limit the liability of MLB Airport.	9/15/2015	Bags do not fall out of cart and are protected from the weather.		
16	Powered using sustainable or "green" energy	Criteria	MLB Airport	Project requirement from MLB Airport.	9/15/2015	The ALTS is powered using a rechargable battery and supplimentary solar panel.		

Appendix N: QFD – House of Quality

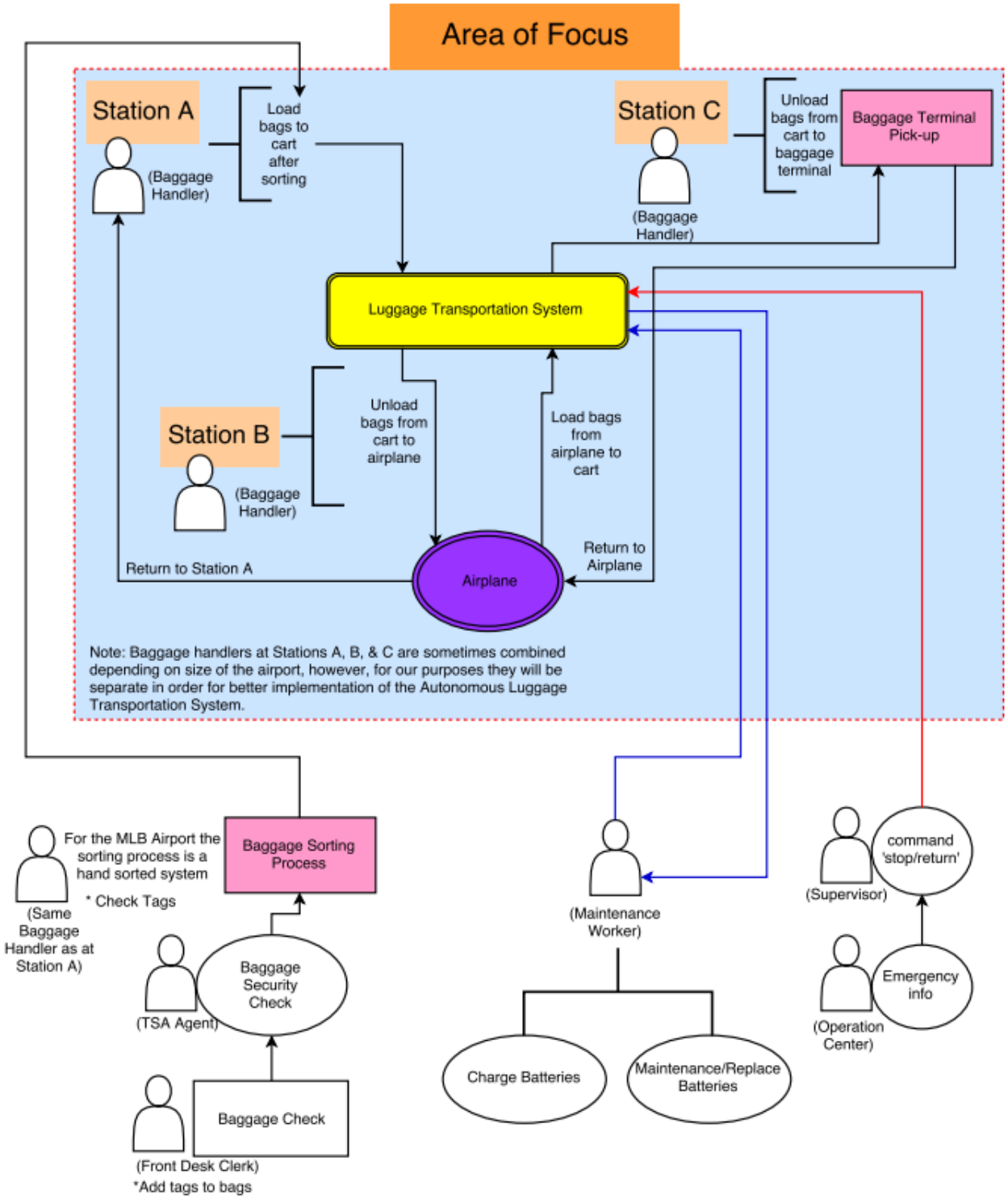


1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	Verfication Method(right)/ Requirement(down&left)	1	2	3	4	5	6	7
		+	+	+											-	1	3		3	2	3		
							-							-	-	2		3					
			+	+					+				-		-	3	3	1	3	3	3		
				+		+	+	+								4	2	3	1	3	2	3	3
							+		+	+		+	+	+		5	3	3	3	3	3		
																6				3		3	
								+		+					-	7				3			3
															+	8	2	2	1	3	1		
																9				3	1		
													+	+		10	1				1		
												+			-	11					1		
																12							
																13	2	1	1				
															-	14			2				
																15		3					
																16	1	2					1
																Importance	145	152	118	170	132	36	44
																%Importance	9.2	9.7	7.5	10.8	8.4	2.3	2.8

List of Symbols and Descriptions	
+ :	The requirements have a positve effect on each other.
- :	The requirements have a negative effect on each other.
1-3 :	The ranking scale of the requirement's relationship to the verification method.

8	9	10	11	12	13	14	15	16	Priority (1 to 10)	Weighted Priority	%Total
1		3			3	1		3	10	220	14.0
							3	2	10	80	5.1
1					3	1		3	9	189	12.0
3	3	3				1			10	270	17.2
3		3			3	2	1		8	216	13.7
									2	12	0.8
								1	2	14	0.9
3								1	9	117	7.4
	3					3		1	10	110	7.0
		3				1			10	60	3.8
			3					1	2	10	0.6
				3					5	15	1.0
					3				4	28	1.8
		1				3	3		7	63	4.0
						3	3		7	63	4.0
3	1				1	1		3	8	104	6.6
124	68	121	6	15	101	135	80	124	Total	1571	
7.9	4.3	7.7	0.4	1.0	6.4	8.6	5.1	7.9			

Appendix O: Use Case Diagram



Appendix P: Jetson Installation

Troubleshooting Softwares

Setting up the Jetson TK1

In order to setup the Jetson TK1, it is important to have a separate computer running Ubuntu 14.04 as a host. The following tasks are executed on the host computer:

1. Downloading the Jetpack installation file. This is required to install Ubuntu 14.04 on the Jetson TK1. This can be done using the following website: https://developer.nvidia.com/embedded/jetpack
2. To run the installation file, permissions must be changed. This can be achieved by running the following command: sudo chmod +x {filename.run}, where "filename.run" is the name of the downloaded Jetpack file.
3. Next, the setup file can be executed using: ./{filename.run}. It is important to note that in b) and c), the commands will only work if the user is in the correct directory. Otherwise, they will not work. In order to get to the correct directory, use the "cd" command: cd {path to file}.

After the installation file is started, follow the instruction on the NVIDIA website (step 1 to 7): http://docs.nvidia.com/jetpack-tk1/1_1/content/developertools/mobile/jetpack_install.htm

Next, connect the host computer and the Jetson TK1 with the provided micro-USB to USB cable. Step 8 in the document above will guide user through the process of flashing Ubuntu 14.04 onto the Jetson TK1. After rebooting, user will be presented with the graphical interface of Ubuntu 14.04.

Setting up NVIDIA CUDA on the Jetson TK1.

The following commands must be executed chronologically via terminal to get NVIDIA CUDA working on the Jetson TK1.

4. Update Ubuntu
    a. sudo apt-get update
    b. sudo apt-get upgrade
5. Installing OpenCV
    a. sudo apt-get install libopencv-dev
    b. sudo apt-get install build-essential checkinstall cmake pkg-config yasm
    c. sudo apt-get install libtiff4-dev libjpeg-dev libjasper-dev
    d. sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libdc1394-22-dev libxine-dev libgstreamer0.10-dev
    e. sudo apt-get install libgstreamer-plugins-base0.10-dev libv4l-dev
    f. sudo apt-get install python-dev python-numpy
    g. sudo apt-get install libtbb-dev
    h. sudo apt-get install libqt4-dev libgtk2.0-dev
    i. sudo ldconfig
6. Installing NVIDIA CUDA
    a. Install the CUDA toolkit: sudo apt-get install nvidia-cuda-toolkit
    b. Install CUDA 6.5 using the following link: https://developer.nvidia.com/linux-tegra-rel-21

Note: It is important to follow the above link and not any other links. There are multiple versions of CUDA, but only the above link allows downloading CUDA for the Jetson

TK1. The downloaded file should be a ".deb" file and can be run easily by preinstalled Software Center in Ubuntu 14.04.

7. Post-installation process for CUDA
    a. Follow this document to do the post installation process: http://developer.download.nvidia.com/compute/cuda/6_5/rel/docs/CUDA_Getting_Started_Linux.pdf#page=24&zoom=auto,108,720
    b. The above link should take user directly to chapter 6 to setup the environment. It is advised that section 6.1 must be followed carefully. If the commands in this section are typed incorrectly, then there will be errors later on.
8. Installing gcc for C/C++ programming
    a. This should be preinstalled on Ubuntu 14.04. However, the following command should be run to make sure: sudo apt-get install gcc

Setting up environment for the ZED Camera

In a new terminal, type: opencv –version to check if opencv, gcc, nvcc are installed.

In order to build and run a C++ program via the ZED SDK, the following files must be available:

1. ".cpp" file, this will be created by the user.
2. "CMakeLists.txt" file. This file tells basic facts about the compiled code. Follow this tutorial to get the file: https://www.stereolabs.com/blog/index.php/2015/06/28/zed-with-opencv/

    In general, this file will not change regardless of the code.

Add "ENABLE_LANGUAGE(C)" at the top of CMakeLists.txt file. This is important because it allows C/C++ codes to be executed.

Fix problem with different version of Cuda

The ZED compiler always looks for the CUDA 6.5 library files; so if you have 7.5, it will fail. To fix this problem, link the 7.5 libraries with the “given” 6.5 libraries using the following command:

sudo ln -s /usr/local/cuda-7.5/lib64/name of required 7.5 libraries /usr/local/cuda-7.5/lib64/name of required 6.5 libraries.

Build and Compile a C++ program via ZED SDK

Firstly, download the ZED SDK program from the following link:

https://www.stereolabs.com/developers/

Make sure to download the correct version for the Jetson TK1.

First create a project folder. It is advised to put this in the /usr/local/zed/ directory. Call it “project”. Then inside “project”, create another folder called “src”. Write and save your code in a file called “main.cpp”. Follow the steps below carefully:

1. Put the main.cpp file inside the src folder.
2. Put the CMakeLists.txt file outside src (inside project).
3. Make a new folder called “build”.
4. In a terminal, change directory into this “build” folder.
5. Type: "cmake ..". If all goes well, terminal should return a checklist of everything.
6. Type: make. Again, if all goes well, it should say “100% Built target…..”

7. Finally, type: ./your target name, to run the file. As default, this is “./ZED_PROJECT”

Automatizing software on Jetson TK1

Automatizing the software is important because it allows our code to run without human interactions. In order to do so, the code must be compiled so that an executable file is created as a result.

For example, the task is to get a code called test.cpp so that it runs every time the system starts. The procedures below are then followed carefully:

1. Compile the test.cpp file using the command below:

    g++ test.cpp -o {name of outputfile, say "test"}

2. Create a blank text file. Inside the file, type the following commands as exactly as they are:

    #!/bin/bash

    cd /path to location of file, in this case Desktop/

    sudo -u {username} /path to location of file

    exit 0

3. Save the text file with extension “.sh” . For example, test.sh and save it on the Desktop.

4. In a new terminal, type the following commands:

    cd /etc/init.d

    sudo nano rc.local

5. This should open up a text file in the terminal.

6. Use the arrow keys on keyboard to navigate because trackpad and mouse don't work.

7. Go down to the last line, where you can see the word "esac"

8. Enter a new line, type:

sh /path to your test.sh file &

exit 0

Note: Do not forget the "&" at the end.

9. Save the text file in terminal. This can be achieved by pressing "Control + O", which will writes out. Press enter to agree to changes and then press "Control + X" to exit.

10. Reboot, everything should work now.

Appendix Q: ZED Camera Main code

```
//FilesInclude
#include"zed.h"

//standard includes
#include <iostream>
#include <fstream>

//OpenCV includes
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>

//ZED includes
#include <zed/Camera.hpp>


 int main(int argc, char** argv)
{

        GPIOinit();

 // Initialize ZED color stream in HD and depth in Performance mode

  sl::zed::Camera* zed = new sl::zed::Camera(sl::zed::VGA,15);
  sl::zed::ERRCODE err = zed->init(sl::zed::MODE::PERFORMANCE, 0, false);



  // Quit if an error occurred
  if (err != sl::zed::SUCCESS) {
    std::cout << "Error: while ZED initialization: " << errcode2str(err) << std::endl;
    delete zed;
    return 1;
  }

  // Initialize color image and depth

        do{
          int width = zed->getImageSize().width;
          int height = zed->getImageSize().height;
          cv::Mat depth(height, width, CV_8UC4,1);
          cv::Mat image(height, width, CV_8UC4,1);
```

```
                // Get frames and launch the computation
                if (!zed->grab(sl::zed::SENSING_MODE::FULL))
                {
                        // get image from ZED camera
                        sl::zed::Mat left = zed->retrieveImage(sl::zed::SIDE::LEFT);
                        memcpy(image.data,left.data,width*height*4*sizeof(uchar));

                         sl::zed::Mat depthmap = zed-
>normalizeMeasure(sl::zed::MEASURE::DEPTH);
                        memcpy(depth.data,depthmap.data,width*height*4*sizeof(uchar));

                        save(image, zed->retrieveMeasure(sl::zed::MEASURE::DEPTH));
                }

                objectLocation();
                examineObject();
        } while(eStop == false);

        printMatricesToFile();

        GPIOclose();


  delete zed;

}
```

Appendix R: zed.h Code

```
//standard includes
#include <iostream>
#include <fstream>
#include<stdio.h>
using namespace std;

//OpenCV includes
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>

//ZED includes
#include <zed/Camera.hpp>

//GPIO
#include <unistd.h>
//#include "SimpleGPIOReal.h"
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <fcntl.h>
#include <poll.h>



int height = 480;
int width = 640;
float distanceMatrix[480][640];
int object[480][640];
float dDistance[480][640];
float difference[480][640];
int objectcheck[480][640];

int jStart;// = 100;
int jEnd;// = 200;
int iStart;
int iEnd;
int sum =0;
int turn = 2; // Left:-1; Straight:0; Right:1; Full Screen:2

const int step = 20; //examineObjects()
const int error = 100; //findObjects()
```

```
const float percentObject = 0.80;
const float sensingDistance = 5000;
const int maxHeight = 360;

bool stop = true;
bool throttle = false;
bool eStop = false;

unsigned int brakePin = 40;
unsigned int throttlePin = 43;
unsigned int eStopPin = 49;

int checkheight;
int checkwidth;


//SAVE FUNCTION
void save(cv::Mat& image, sl::zed::Mat depth)
{
	//SAVING DEPTH MAPS

	float* ptr_d;

	for (int i = 0; i< depth.height; ++i)
	{
		ptr_d = (float*)(depth.data + i * depth.step);

		for (int j = 0; j < depth.width * depth.channels; ++j)

		{
			if (ptr_d[j] < 0 || ptr_d[j] > sensingDistance ){
				ptr_d[j] = sensingDistance;
			}

			distanceMatrix[i][j] = ptr_d[j];

		}


	}
	//std::cout << "DONE" << std::endl;

}

void GPIOinit(){
```

```
	//pin 40 on 75 pin
	system("echo 160 > /sys/class/gpio/export");
	system("echo out > /sys/class/gpio/gpio160/direction");
	system("echo 1 > /sys/class/gpio/gpio160/value");

}

void GPIOclose(){
	system("echo 160 > /sys/class/gpio/unexport");
}

//OBJECT LOCATION FUNCTION
void objectLocation(){

	for(int i = 0; i < height; i++){

		inBoundary = false;

		for(int j = 0; j < width; j++){

			if(distanceMatrix[i][j] > sensingDistance && j < width){
				distanceMatrix[i][j] = sensingDistance;
			}

			if(j > 0){

				dDistance[i][j-1] = distanceMatrix[i][j] - distanceMatrix[i][j-1];

				if(dDistance[i][j-1] < -1*error && dDistance[i][j] >= -1*error &&
dDistance[i][j] <= error){ //- to 0
					object[i][j] = 1;
					inBoundary = true;
				}
				else if(dDistance[i][j-1] > error && dDistance[i][j] <= error &&
dDistance[i][j] >= -1*error){//+ to 0
					object[i][j-1] = 1;
					inBoundary = false;
				}
				else if(inBoundary == true && dDistance[i][j-1] >= -1*error &&
dDistance[i][j-1] <= error
&& dDistance[i][j] >= -1*error && dDistance[i][j] <= error){// 0 to 0
					object[i][j-1] = 1;
				}
				else if(dDistance[i][j-1] > error && dDistance[i][j] > error){ //+ to
+
```

```
                                        object[i][j-1]= 1;
                                }
                                else if(dDistance[i][j-1] < -1*error && dDistance[i][j] < -
1*error){//- to -
                                        object[i][j-1]= 1;
                                }

                                else if(dDistance[i][j-1] < -1*error && dDistance[i][j] > error){//-
to +
                                        object[                         i][j-1]= 1;     inBoundary =
false;
                                }
                                else if(dDistance[i][j-1] >error && dDistance[i][j] < -1*error){//+
to -
                                        object[i][j-1]= 1;
                                }

                                else{
                                        object[i][j-1] = 0;
                                }

                                if(j == width-1){
                                        dDistance[i][j] = 0;
                                        object[i][j] = 0;
                                }
                        }
                }
        }
}

//EXAMINE OBJECT FUNCTION: examine small areas, using "OBJECT" in LOCATION
function above.
void examineObject()
{

        checkheight = floor((iEnd-iStart)/step);
        checkwidth = floor((jEnd-jStart)/step);

        int sumMatrix[480][640];
        int objectCheckSum = 0;

        //TURNING

        switch(turn){
                case -1: {jStart = 9; jEnd = 409; iStart = 0; iEnd = 240;}
```

```
        break;
        case 0: {jStart = 209; jEnd = 429; iStart = 240; iEnd = 480;}
        break;
        case 1: {jStart = 209; jEnd = 629; iStart = 240; iEnd = 480;}
        break;
        case 2: {jStart = 9; jEnd = 629; iStart = 0; iEnd = 480;}
        break;
        default: "Something went Wrong.\n";
}


for(int i = iStart; i < iEnd; i = i + step)
{

        //std::cout <<"Row : " <<i <<"          ";

        for(int j = jStart; j < jEnd; j = j+step)
        {

        sum =0;

//      for(int u = 0; u <= x <= 20;

                for(int u = i; u <= i + step - 1; ++u)
                {
                        for(int v = j; v <= j + step- 1; ++v)
                        {
                                sum = sum + object[u][v];

                        }
                }

                int icheck = (int)((i-step/2)/step);
                int jcheck = (int)((j-jStart)/step);

                //sumMatrix[icheck][jcheck] = sum;

                //      std::cout << sum <<"  ";

                double maxsum = percentObject*(step-1)*(step-1);
                //std::cout << maxsum <<"\n";

                if ((double)sum >= maxsum)
                {
                        objectcheck[icheck][jcheck] = 1;
```

```
                        objectCheckSum++;

                }
                else
                {
                        objectcheck[icheck][jcheck] = 0;
                }


        }
    }

    if(objectCheckSum > 0){
        cout<< "Stop\n";
        cout << objectCheckSum << endl;
        stop = true;
        system("echo 0 > /sys/class/gpio/gpio160/value");
    }
    else{
        cout<< "Go\n";
        cout << objectCheckSum << endl;
        stop = false;
        system("echo 1 > /sys/class/gpio/gpio160/value");
    }



}



//TAKE PREMADE MATRIX AND OUTPUT TO A FILE

void printMatricesToFile(){

//PRINT distanceBase

/*
    FILE* outfileNew;
    outfileNew = fopen("baseMatrixNew.dat", "w");
    for(int i = 0; i < height; i = i + 1){

        fprintf(outfileNew, "%lf", i);
        fprintf(outfileNew, "\t");
```

```
                for(int j = 0; j < width; j = j + 1){
                        fprintf(outfileNew, "%lf\t",distanceBase[i][j]);
                }

                fprintf(outfileNew,"\n");
        }
        fclose(outfilenew);
*/


//PRINT OBJECT CHECK MATRIX

/*
        for(int i = 0; i < checkheight; i = i + 1){

                std     ::cout <<"Row: " <<i <<"       ";

                for(int j = 0; j < checkwidth; j = j + 1){
                        cout << objectcheck[i][j];
                        cout << "\t";
                }

                std::cout<<"\n";
        }
*/

//PRINT OBJECT CHECK?  CHECK Please..


/*
        FILE* outfileObject;
        outfileObject = fopen("objectcheck.dat", "w");
        for(int i = 0; i < checkheight; i++){
                for(int j = 0; j < checkwidth; j++){

                        fprintf(outfileObject, "%d\t",objectcheck[i][j]);
                }

                fprintf(outfileObject,"\n");
        }
        fclose(outfileObject);

 */
```

```
//PRINT OBJECT MATRIX

/*
	FILE* outfile;
	outfile = fopen("object.dat", "w");
	for(int i = 0; i < height; i++){
		for(int j = 0; j < width; j++){

			fprintf(outfile, "%d\t",object[i][j]);
		}

		fprintf(outfile,"\n");
	}
	fclose(outfile);

*/

//PRINT dDistance MATRIX

/*
	FILE* outfileDifference;
	outfileDifference = fopen("dDistance.dat", "w");
	for(int i = 0; i < height; i++){
		for(int j = 0; j < width; j++){

			fprintf(outfileDifference, "%lf\t",dDistance[i][j]);
		}

		fprintf(outfileDifference,"\n");
	}
	fclose(outfileDifference);

*/


//PRINT distanceMatrix


/*
	FILE* outfileDistance;
	outfileDistance = fopen("distanceMatrix.dat", "w");
	for(int i = 0; i < height; i++){
		for(int j = 0; j < width; j++){
```

```
                        fprintf(outfileDistance, "%lf\t",distanceMatrix[i][j]);
                }

                fprintf(outfileDistance,"\n");
        }
        fclose(outfileDistance);
*/


}
```

References

[1] Federal Aviation Administration, 'Federal Aviation Administration Advisory Circular', 2015. [Online]. Available: http://www.faa.gov/documentLibrary/media/Advisory_Circular/150_5210_5d.pdf. [Accessed: 09- Sep- 2015].

[2] Melbourne International Airport, 'Melbourne International Airport Rules and Regulations', 2015. [Online]. Available: http://www.mlbair.com/Portals/0/docs/Airport%20Manuals/MAA%20Rules%20and%20Regulations.pdf. [Accessed: 09- Sep- 2015].

[3]*On car steering torques at parking speeds*, 1st ed. London, 2003, p. 7.

[4] NVIDIA Jetson TK1 main page

http://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html

[5] Stereo Labs ZED Camera main page

http://zedstore.stereolabs.com/products/zed

[6] Swift NAV GPS main page

http://www.swiftnav.com/piksi.html

[7] MaxSonar Sensors main page

http://www.maxbotix.com/Ultrasonic_Sensors/Rangefinders.htm

[8] Arduino Due main page

https://www.arduino.cc/en/Main/ArduinoBoardDue

[9] Google Car Photo

http://www.businessinsider.com/the-google-car-is-a-huge-threat-to-the-auto-industry-2015-1

[10] Statisticsbrain.com