

2018-01-01

# Crawler Inspection Robot With Machine Vision (lifelong Endeavour)

Francisco Martinez

*University of Texas at El Paso*, [fj.martinez.engr@gmail.com](mailto:fj.martinez.engr@gmail.com)

Follow this and additional works at: [https://digitalcommons.utep.edu/open\\_etd](https://digitalcommons.utep.edu/open_etd)

---

## Recommended Citation

Martinez, Francisco, "Crawler Inspection Robot With Machine Vision (lifelong Endeavour)" (2018). *Open Access Theses & Dissertations*. 1482.

[https://digitalcommons.utep.edu/open\\_etd/1482](https://digitalcommons.utep.edu/open_etd/1482)

This is brought to you for free and open access by DigitalCommons@UTEP. It has been accepted for inclusion in Open Access Theses & Dissertations by an authorized administrator of DigitalCommons@UTEP. For more information, please contact [lweber@utep.edu](mailto:lweber@utep.edu).

CRAWLER INSPECTION ROBOT WITH MACHINE  
VISION (LIFELONG ENDEVOUR)

FRANCISCO JOAQUIN MARTINEZ  
Master's Program in Systems Engineering

APPROVED:

---

Tzu-Liang (Bill) Tseng, Ph.D., Chair

---

Yirong Lin, Ph.D.

---

Eric Smith, Ph.D.

---

Charles Ambler, Ph.D.  
Dean of the Graduate School

Copyright ©

by

Francisco Joaquin Martinez

2018

## **Dedication**

For Catalina.

CRAWLER INSPECTION ROBOT WITH MACHINE  
VISION (LIFELONG ENDEVOUR)

**by**

FRANCISCO JOAQUIN MARTINEZ, B.S.M.E.

THESIS

Presented to the Faculty of the Graduate School of  
The University of Texas at El Paso  
in Partial Fulfillment  
of the Requirements  
for the Degree of

MASTER OF SCIENCE

Department of Industrial, Manufacturing, and Systems Engineering

THE UNIVERSITY OF TEXAS AT EL PASO

May 2018

## **Acknowledgements**

I would like to acknowledge and thank the following people that have helped me in the journey to complete the MS in Systems Engineering. Dr. Bill Tseng for being there for support, feedback, and guidance. Dr. Yirong Lin for giving encouragement and direction when I needed it whether; or not I knew I needed it. And finally, Orlando Gordillo, Michael Ellis, Martin Morales, and Luis Osegueda. Without their support and friendship this project would not have succeeded.

## **Abstract**

The presence of foreign objects (FO) in an aircraft can mean failure to achieve a mission's objective, loss of aircraft, or a catastrophic failure such as loss of life. Currently, Lockheed Martin inspection methods for foreign objects include a person accessing very tight areas that have the potential for FO to be present. There is a need that has been identified to remove the person from this very tight/potentially hazardous area that is being inspected and placing them behind the control center of a remote inspection system that can identify, objectively, whether or not FO is present or not. The proposed inspection system will integrate a machine vision system capable of automatically identifying and qualifying certain features and attributes with an automatically or remotely controlled vehicle that is capable of maneuvering and accessing hard to reach areas of high importance. The system's status and controls will be shown on a user interface for the person overseeing the inspection as well as a different user interface for personnel observing the inspection. Objects that have not passed the inspection criteria will be identified a record will be generated automatically in a report of the inspection. The video of the inspection will be archived for future reference.

This document will provide an overview of the development of the system that UTEP has produced for Phase 1 of the Crawler Project.

## Table of Contents

Acknowledgements .....	v
Abstract .....	vi
List of Tables .....	ix
List of Figures .....	x
Chapter 1: Literary Review .....	1
Chapter 2: User Interface .....	2
2.1 Prototypes .....	2
2.2 User Interface Future Work .....	5
Chapter 3: Database .....	7
3.1 Database Components .....	7
3.2 Database Future Work .....	9
Chapter 4: Vision System .....	11
4.1 Current Machine Vision and Machine Learning .....	11
4.2 Algorithms and Setup Used .....	12
4.3 Alternatives Evaluated .....	16
4.4 Machine Vision Future Work .....	20
Chapter 5: System Interfaces .....	24
5.1 System Interfaces Design .....	24
Chapter 6: Mechanical Design .....	27
6.1 Design Overview .....	27
6.2 Commercial off the Shelf Components .....	29
6.3 Motion Control .....	30
6.4 Brushless Motors and Electronic Speed Controller Configuration .....	30
6.5 Frame Components .....	31
6.6 Flow Shroud .....	32
6.7 Motor Supports .....	33
6.8 Stereolithography 3D Printed Parts .....	34
6.9 Flow Simulation .....	35



6.10 Prototype/Results .....	42
Chapter 7: Conclusions .....	45
References .....	46
Appendix .....	47
Vita .....	51

## **List of Tables**

Table 4.1.1: Comparison of Object Identification Methods .....	19
Table 6.10.1: Position Performance.....	43
Table 6.10 2: Obstacle Surmounting Performance .....	43

## List of Figures

Table 1 1: Evaluation of Climbing Robots .....	1
Figure 2.1.1: First Prototype .....	3
Figure 2.1.2: Partially Functional Prototype .....	4
Figure 2.1.3: Demoed Prototype.....	5
Figure 3.1 1: Entity Relational Diagram.....	9
Figure 4.2.1: Neural Network Example .....	14
Figure 4.2.2: Cropped and Context Images .....	15
Figure 4.2.3: Detecting Multiple Defects Example .....	15
Figure 4.3.1: KNN Drawing .....	17
Figure 4.3.2: KNN with Fibonacci Algorithm.....	18
Figure 5.1.1: System Component Map .....	24
Figure 6.1 1: Thrust Needed for Vertical Maneuvers .....	28
Figure 6.1 2: Thrust Needed for Overhead Maneuvers .....	28
Figure 6.1.3: Current Crawler design. ....	29
Figure 6.3.4:Arduino/ESC/Motor Schematic .....	31
Figure 6.5.1: Carbon Fiber Frame Components .....	31
Figure 6.6.1: Flow Shroud (Red) .....	33
Figure 6.7.1: Propeller Motor Supports .....	34
Figure 6.8.1: SLA 3D Printed Components.....	35
Figure 6.9.1: Geometry Simulated.....	36
Figure 6.9.2: Geometry with Shell.....	37
Figure 6.9.3: Geometry with Increased Shell Height .....	38

Figure 6.9.4: Geometry with Increased Distance Between Shell and Propeller .....	39
Figure 6.9.5: Shell with a 30 Degree Lead-in .....	40
Figure 6.9.6: Propeller Placement Raised.....	41
Figure 6.9.7: Final Geometry and Propeller Placement Used .....	42
Figure 10.1.1: Working Built Prototype .....	44
A 1: Page 1 of Crawler Setup Reference Document.....	47
A 2:Page 2 of Crawler Setup Reference Document.....	48
A 3:Page 3 of Crawler Setup Reference Document.....	49
A 4:Page 4 of Crawler Setup Reference Document.....	50

## Chapter 1: Literary Review

An exploration of different locomotion types for the vehicle to mount a machine vision system was performed with the benefits and disadvantages identified in the table below.

Table 1 1: Evaluation of Climbing Robots

Name of Publication	Locomotion Type	Disadvantage
The design of tracked mobile robot for non-urban environment	Tracks	Cannot Climb Walls
Semi-autonomous serially connected multi-crawler robot for search and rescue	Tracks/Modules	Cannot Climb Walls
An Innovative Approach to Pipeline Health Monitoring Using Crawler Robot with PVDF Based Probe.	Tracks	Built for the Inside of pipes
Grasping Claws of Bionic Climbing Robot for Rough Wall Surface: Modeling and Analysis	Claws	Limited Degrees of Freedom
Development of a Modular Crawler for Tracked Robots	Tracks	Cannot Climb Walls
VERTICAL CLIMBING LOCOMOTION OF A NEW GEKO ROBOT USING DRY ADHESIVE MATEIRIAL	Tracks/Dry Adhesive	Limited to Glass Surfaces
Series of Multilinked Caterpillar Track-type Climbing Robots*	Tracks/Dry Adhesive/Magnets/Suction Pads	Limited Degrees of Freedom

One thing to note about the main method of locomotion with the robots that were evaluated included tracks. This is the main reasons for selecting tracks for the locomotion for achieving movement.

## **Chapter 2: User Interface**

The User Interface will display the current inspection being performed to the operator and personnel that have credentials to access the stream of the inspection.

The user interface was developed on the Android Studios development platform using a model view controller software development architecture. Android Studio has many preset tools that aid in rapid development; Android Studios is programmed in Java and is currently one of the most popular languages in terms of user base and online documentation. The User Interface was decided to be simple, mimic popular video game controllers, and to have all important information quickly accessible.

During the development process, studies and development methodologies were borrowed from “Designing the User Interface, 6th edition” by Ben Shneiderman and Catherine Pleasant. This was the reference material used in the Fall 2017 Human Interaction Courses at the University of Texas at El Paso.

### **2.1 PROTOTYPES**

The original design of the user interface (see Figure 1.1) was very simple, consisting of a grey box indicating where a live stream would go, an analog stick for manual movements of the crawler, arbitrary buttons pertaining to some form of user control, and information pertaining to any defects found during the course of the inspection. This prototype was made to determine some key features that the crawler interface would have and purposely looks simple to aid in eliciting requirements from Lockheed Martin, this is because studies show more informative feedback is given when presented very bland and basic setups.

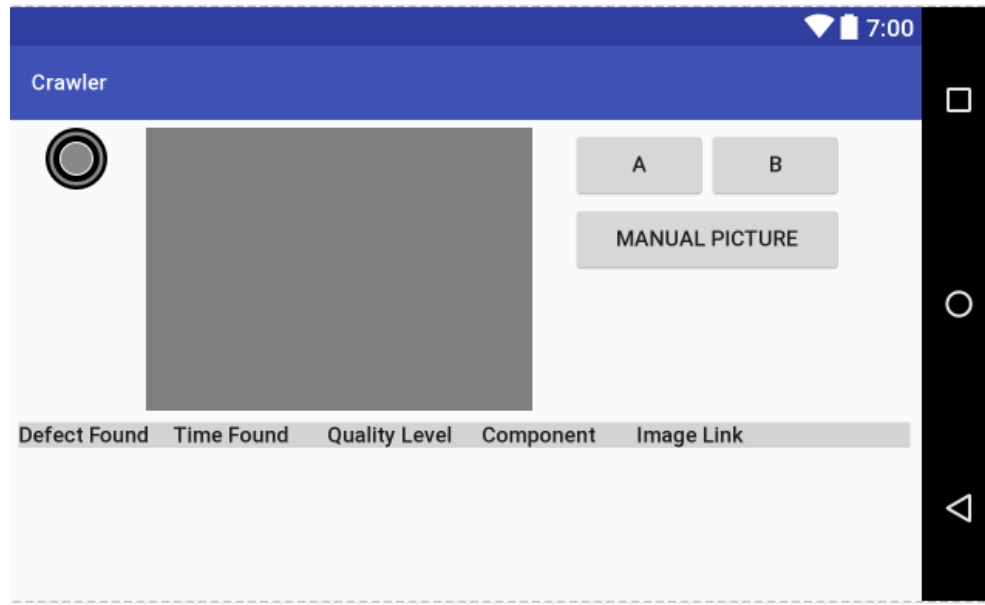


Figure 2.1.1: First Prototype

The next phase of the prototype included basic functionality due to the need for testing, it included a live stream of MJPEG pictures taken directly from the camera and pushed to a website that displays a single MJPEG image at any given time. This method was chosen because it was the fastest streaming available for android devices. This design also included two analog sticks due to the new requirement of adding a future pan and tilt functionality for the crawler's camera. Lastly, the ability to view the pictures of stored defects on the device was added to show the ability to store and retrieve information directly to and from the database. This information was displayed over the live feed and disappeared when clicked.

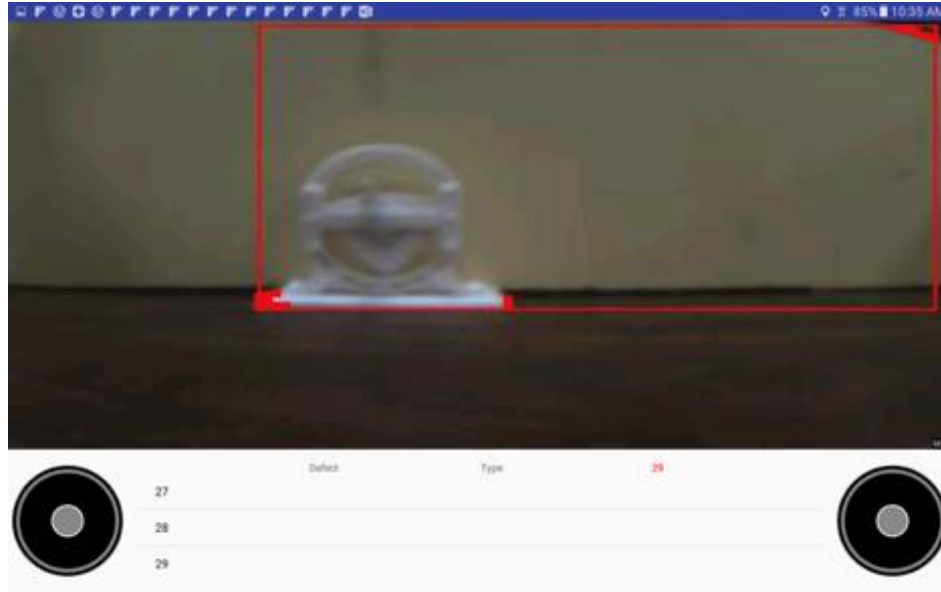


Figure 2.1.2: Partially Functional Prototype

The next phase was a more polished version and was intended to serve as a partially functional prototype with a Lockheed theme. This phase of development offered all the previous functionality with a change in layout. Appearance and usability were now important at this phase. As such a color scheme that was easy on the eyes was chosen using Lockheed Martin's primary colors. The Stream was placed directly in the center with padding to both the left and right. The Right side padding stored information about the current inspection, such as the category of defect and the time it was found, and when clicked would bring up a pop up box to showing a picture of the defect and allowing the inspector to change the defect's corresponding category. The Left Padding became a placeholder for an imbedded texting application that would be used to communicate between the inspector and anyone on the website. The controls were centered towards the bottom of the user interface and dynamically change based off the current crawler state. Notes have been overlaid in red text to provide clarity.





Figure 2.1.3: Demoed Prototype

## 2.2 USER INTERFACE FUTURE WORK

One item of future work would be to introduce some form of augmented reality to aid inspectors in determining problems that may not present themselves as defects, such as bays being inconsistent in the placement of pipes and other structures. The current course of action to implement this would be to develop a simplistic 3d environment similar to the inspection environment. A virtual crawler would also be added into this environment and the inspections would occur simultaneously with a tangible overlay of the virtual environment being displayed on top of the real time environment. The additional development tool required is the 3d Unity gaming engine.

Another action item is fixing the alignment of the text as well as other usability issues that may arise from feedback based on the opinions of inspectors that work for Lockheed Martin. This is currently low priority and development time is spent elsewhere on the system.

The Website needs to be reworked to handle multiple crawler's live streams at any given time and send and receive messages with the inspector. Future work may include running an inspection from just the website.

The user interface can be reprogrammed to have the same look and feel as the current user interface but with Android and IOS compatibility.

## **Chapter 3: Database**

The current database for the crawler system acts as the central hub for communication between different components of the crawler software system. The database is the most highly centralized point and handles most of the communication between the different components of the system, this is because the database is highly unlikely to change whereas other components may need to undergo various changes throughout development. The database has three main tables: Path, Inspection, and Defect.

### **3.1 DATABASE COMPONENTS**

The Path table consists of various paths a crawler can take. Each path has a unique identifier corresponding to a bay and each bay has a series of steps that may be undertaken. For example, bay 1, may have 12 steps labeled 1-12. Each step has an associated number of actions which are read by the Arduino. The Arduino has several basic functions including forward, backward, left, right, propeller up, propeller down, stop and display defects, and take a picture.

Each step consists of a single action and a time frame. Some actions such as taking a picture and controlling the propeller have no need for a time frame and the respective category is stored as NULL. The steps are executed in numerical order by the Arduino and each have a bay that they correspond to. This design was chosen as to support any number of paths without having the Arduino store the paths internally

The Inspection Table Contains information that is shared by each defect of a particular inspection, as well as information that is entered manually by the inspector. The reason for this table is to reduce the amount of repeated information while isolating non-computer vision related

information that is obtained by the inspector. This information is obtained through the iPad and contains information such as the tail number of the aircraft and the bay which is to be inspected.

The Defect Table consists of information populated by the deep learning algorithm along with the location a defect was found. The Location of the defect is determined at run time by consulting the system along with the time. A picture of the defect itself is stored a sequence of letters known as a byte array in which a machine can learn and decode to be the original picture.

Below is a simplified version of these tables. The table below is an ER diagram and shows the interaction between different tables. Rectangles are the tables themselves, circles their attributes, and triangles their relations. The numbers between associations dictate multiplicity, in this case one inspection may have zero to many defects. The Choice behind this layout was to keep things simple, reduce repeated information to ensure as little data is taken up per entry, and to calculate some values rather than storing them explicitly.

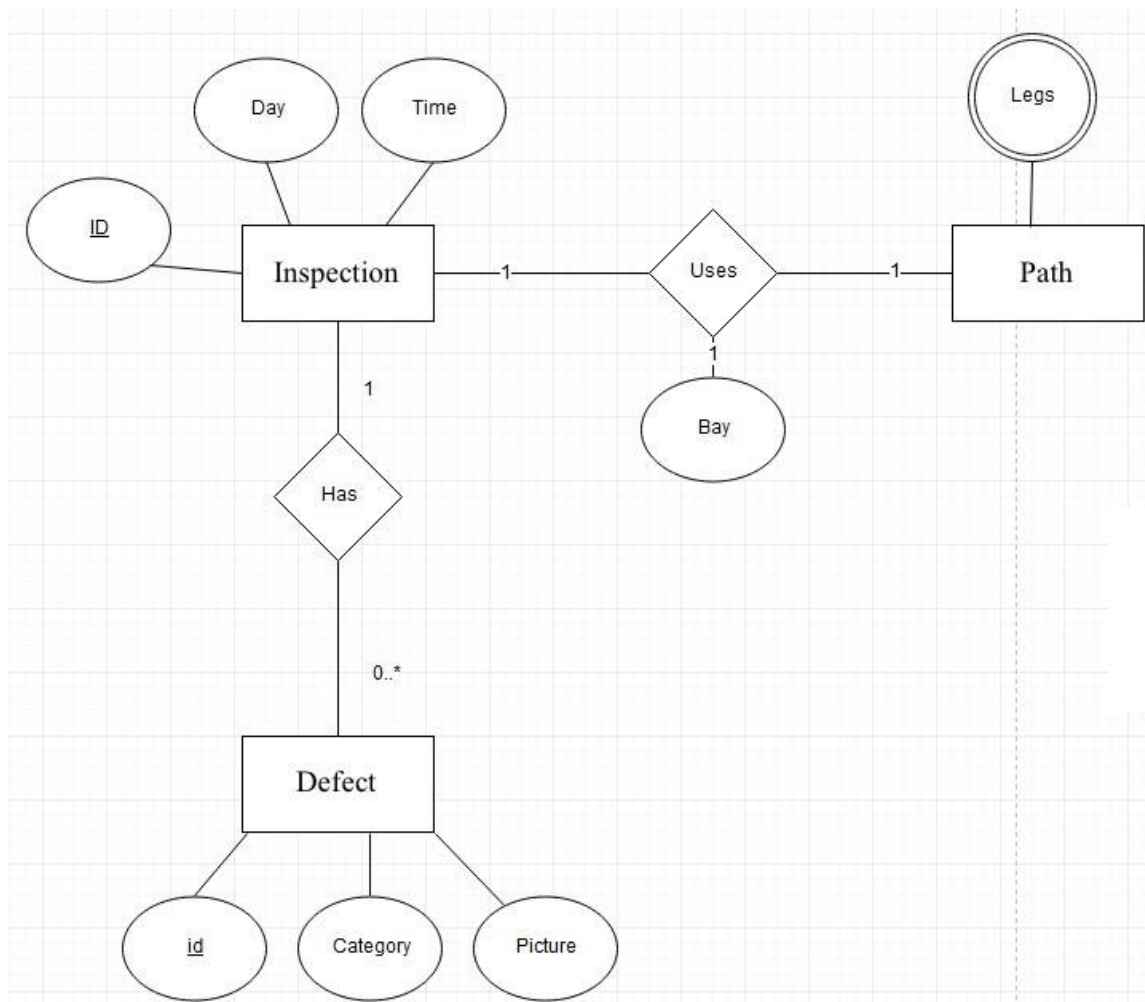


Figure 3.1 1: Entity Relational Diagram

### 3.2 DATABASE FUTURE WORK

The database currently has tables representing arbitrary information that the team found useful for identifying defects. Future tables will have all values that need are needed by Lockheed Martin.

The Database currently only supports one path, the way the path is stored has to be changed to support multiple paths that can be determined by the inspector. Currently the clean inspections

are stored on the computer, in order to have a more dynamic system that can be accessed from anywhere the clean stages of the environment also need to be stored in the database.

The database may be modified to store manual paths temporarily; this is because most communication between components traverses the database. The reason manual paths may need to be stored is because if an automated inspection is interrupted the path must be traversed backwards in order to continue the automated inspection when manual control is relinquished by the inspector.

Currently the database runs on Microsoft SQL, if Lockheed uses a different version of SQL we will have to change our database to that language. This would require changing the queries to be of the same format throughout the system.

Auto QAR reports have been experimented with and appear feasible, given a template. Most components exist to allow for the capability to automatically generate QARs. While this is not a requirement for the crawler, it could potentially be implemented during future phases.

## **Chapter 4: Vision System**

In general, Machine Vision is the field of Computer Science where information is extracted from an image or a collection of images. Object recognition is an example of information that can be extracted. We are using Machine Vision in the LMA Crawler project so that we can detect defects inside of an airplane bay.

Machine Learning is another field of Computer Science that creates systems that can learn and improve from past experiences. Machine Learning is a hot topic at the moment with large technology companies. For the LMA Crawler project, we are using Machine Learning to classify defects. That means to take a collection of interesting pixels and being able to determine that it is a piece of tape, for example.

### **4.1 CURRENT MACHINE VISION AND MACHINE LEARNING**

The first step that a user needs to take for our system to work is to place a crawler in an ideal airplane bay and run a training script. By ideal, we mean an airplane bay that is free of defects. The training script orders the crawler to move around the bay in a predefined path, stop at specific points of interest, and take pictures. These pictures are stored on your local machine; in the future, we would save these in a database. The ideal images are then used in future inspections of airplane bays that potentially have defects.

The second step for the user is to place a crawler in an airplane bay of the same model of aircraft to be inspected and run an inspection script. The crawler follows the same predefined paths, stop at areas of interest, and take pictures of these scenes. The images are then processed to detect and classify any defects found in the scene. These defects are then stored in a database to be used

in inspection reports, order the removal or fixing of defects, and improve the defect classification system.

The user interacts with the system through an Android application and a computer. On the computer, the user needs to start the Crawler Vision System on QT creator and select the training or inspection script. The user then uses the Android application to begin training and inspection scripts, to view the live-feed of the crawler as it moves around, and to see any defects found.

## **4.2 ALGORITHMS AND SETUP USED**

Currently, the pieces of hardware needed for the Machine Learning and Machine Vision aspect of the LMA Crawler project include a Pixy Camera, an Arduino board, and a computer. The software used is Qt 5.2.1, Qt creator 3.0.1, MinGW 4.8 32-bit, Git, CMake 3.7.2, and OpenCV-3.2. You also need to be running a Microsoft SQL database. Details for what tables the database should contain can be found in the <INSERT USER UI/DB LINK> section. For instructions on setting up the development environment, please see the README.md file in the CrawlerVisionSystem Github repository that can be found here (Note: This repository is private, so access needs to be given before you can see this page. The code and development environment setup instructions may also be zipped up and sent to interested parties).

### ***Training Script:***

When the user places a crawler in an airplane bay and starts up the Training Script on the computer and Android application, the computer sends movement commands via serial communication to the Arduino board. The board then starts and stops the appropriate motors to follow the movement commands.



There is a particular movement command that may be sent for areas of interest where the Pixy camera takes a picture; then the Arduino backs up and moves forward before continuing with the next movement commands. The reason that the Arduino backs up and moves forward is to keep it consistent with the inspection script movement. In the inspection script, moving back is necessary before taking a picture to account for sections of the image being lost due to image alignment; this process is explained in the Inspection Script section.

All of the images taken in the training script have an edge-detection algorithm called Sobel applied. This algorithm uses thresholds to determine the edges of all of the objects in the image. This process is used to simplify the image, which is crucial for the accuracy of the subtraction process used in the Inspection Script.

### ***Inspection Script:***

Similar to the Training Script, the user places a crawler in an airplane bay of the same aircraft model and starts the Inspection Script on the computer and Android application. The crawler follows the same path it uses in the Training Script by using the same movement commands from the database. When it takes a picture, it follows five steps: 1. Apply Sobel filter and image alignment algorithm 2. Subtract the current Sobel image with the trained clean Sobel image 3. Detect if there are any defects in the image 4. Classify each defect and save the context and cropped images to the database 5. Pause to show the inspector if any defects were found.

One of the key changes we have made for the system is the ability to classify defects. We used a neural network for classification. A Neural Network is a network of layered nodes that receive some input and give some output. In our case explicitly, after the network is trained, we give an image that may be a washer, tape, or something else, and the output is whether the network classifies it as a washer, tape, or unknown. We train a neural network by passing it a collection of

labeled images. That means for each actual image of an individual defect; we tell the network what kind of defect it is. Below is an image of a sample Neural Network with an arbitrary amount of nodes.

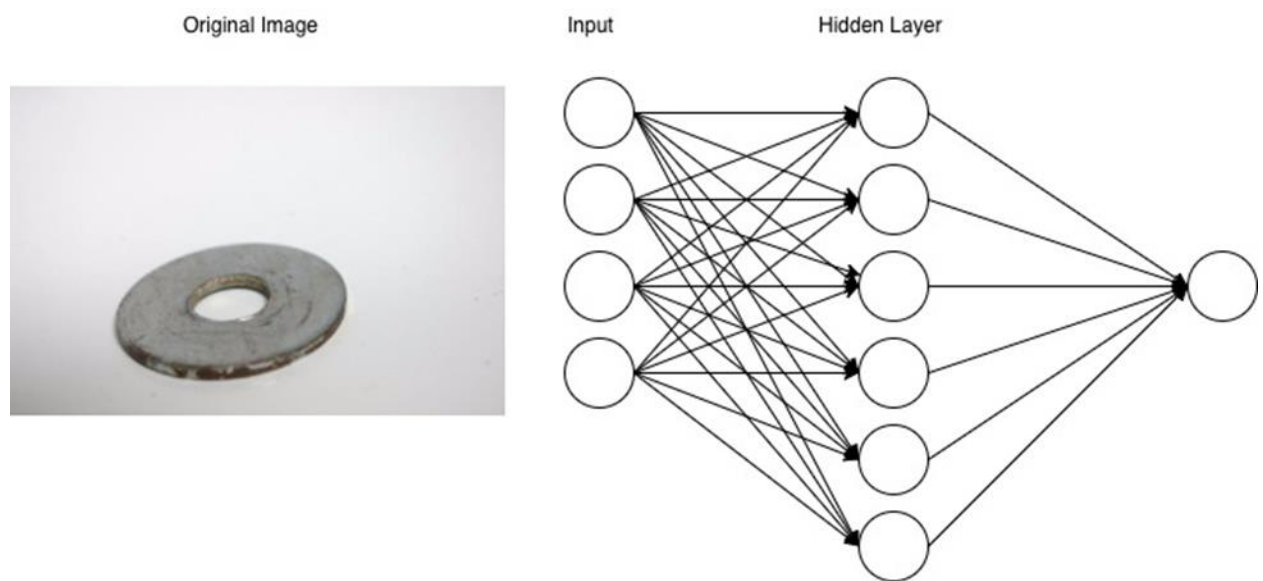


Figure 4.2.1: Neural Network Example

After classifying a detected defect, we then save a context image, cropped image, and the classification to the database. A cropped image is an image of only the defect with intensified lines and a slight padding on the sides. The cropped images are used to train the neural network and classify the object. A context image is an entire image from the live-feed (which may contain other defects), with a box around the specific object in question. Below is an image of how a cropped and context image look.

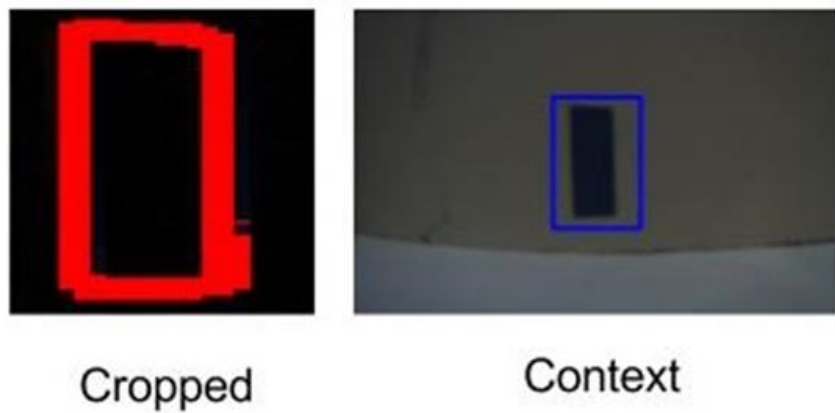


Figure 4.2.2: Cropped and Context Images

Finally, the last step is to pause the crawler for a few seconds to show boxes around all of the defects found in the image. We pause and show the boxes to make the inspector aware that defects were found. Below is an image of multiple defects with boxes around them.

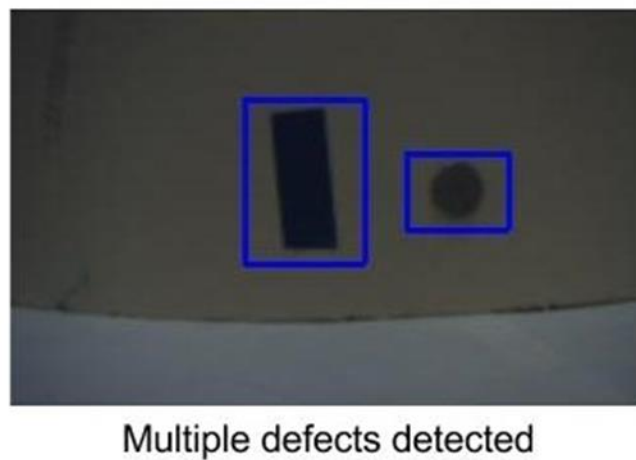


Figure 4.2.3: Detecting Multiple Defects Example

### 4.3 ALTERNATIVES EVALUATED

#### *K's Nearest Neighbors with a Fibonacci Algorithm for calculating resemblance weights:*

Before we settled on using a Neural Network, we attempted to use a naive machine learning algorithm called K-Nearest Neighbors (KNN) to classify defects. This algorithm works by training the system with a collection of labeled images (similar to how it is described for neural networks in the Inspection Script section). Then to classify an image, we pass the image to the system and a k-neighbors value. For each trained image, the system calculates a resemblance value of how different the new image is to the trained image. The system finds whichever classification has at least k-neighbors closest to the new image, and outputs that classification.

The image below is included to demonstrate KNN. The green circle is the image to be classified. The blue squares and red triangles are labeled images used to train the KNN algorithm; their distance from the green circle is how closely they resemble the circle. With a k-value of three, the system looks for the three closest images to the image to be classified. In this case, the algorithm finds three blue squares closer than three red triangles. The algorithm would then classify the image as a blue rectangle in this example. For our purposes, a blue square might be a washer, and a red square might be a tape.

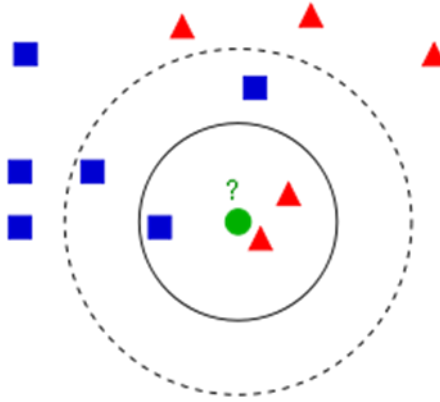


Figure 4.3.1: KNN Drawing

We used this algorithm in conjunction with a homemade Fibonacci algorithm. This algorithm just took the resemblance weights of the closest k-neighbors, weighted them using Fibonacci values that favored closer images, and returned a weighted average of how much the closest k-neighbors resembled the new image. This value was a threshold. If the averaged resemblance did not pass a threshold, that means there were not enough images that closely resembled the image to be classified, therefore it was classified as unknown. As it was implemented, there was a flaw in the Fibonacci algorithm that took the closest k-neighbors and not the closest k-neighbors of the final classification type. In other words, in the image above, if the k-value were three, the Fibonacci algorithm would give an average of the two red triangles and one blue square closest to the green circle, instead of using the resemblance value of the closest three blue squares since it was classified as a blue square. Below is an image to demonstrate how the average weighted resemblance value (labeled as confidence in the image) was calculated using Fibonacci numbers.

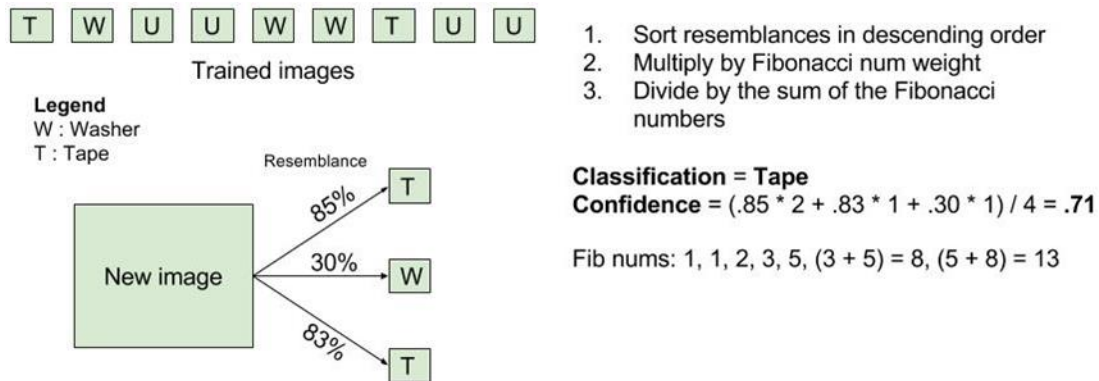


Figure 4.3.2: KNN with Fibonacci Algorithm

We decided to abandon the KNN algorithm in favor of the neural network because, in our experimentation, the KNN algorithm seemed to be less accurate than neural networks at classifying images. Unfortunately, statistics were not taken of the difference in accuracy, but we would ballpark the difference of accuracy as 30%.

### ***Feature Detection:***

We considered using a feature detection algorithm, instead of a combination of Sobel and subtraction algorithms, for detecting defects. The way a feature detection algorithm works are that features from a specific defect (ex. washer) are taken and stored in a feature detector. Then, when we look for defects, we use all of our known feature detectors to search for defects. Below is a table that compares the advantages/disadvantages of each approach.

Table 4.1.1: Comparison of Object Identification Methods

	Sobel/Subtraction	Feature Detection
Pros	<ul style="list-style-type: none"> <li>• Already implemented</li> <li>• Excels at detecting differences in images</li> </ul>	<ul style="list-style-type: none"> <li>• Does not require aligning</li> <li>• Does not need to train areas (works in all environments instantly)</li> <li>• Excels at recognizing objects</li> </ul>
Cons	<ul style="list-style-type: none"> <li>• Need to align image, which causes inaccuracy of subtraction</li> <li>• Needs to train areas</li> </ul>	<ul style="list-style-type: none"> <li>• Each feature detector must be custom made for each defect</li> <li>• Time to implement when we already have Sobel/subtraction</li> <li>• Won't detect untrained defects*</li> </ul>

We abandoned the idea of feature detectors, because there is a chance that a defect that has not had a feature detector created for it (ex. a boot) would appear in the airplane bay, and the system would not detect it. Not being able to detect untrained defects was a deal-breaker because we strive to avoid false-negatives at all costs. In future phases, we hope to combine sobel and subtraction with a second layer of feature detectors to gain the advantages of both.

### ***Subtraction Inaccuracy:***

One of the challenges we faced in the Machine Vision aspect of this project was the inaccuracy of subtraction. In an ideal world, every airplane bay would be created the exact same, so that we may subtract the differences between two airplane bay points of interests with ease. Unfortunately, that is not the case. We try to overcome slight differences (such as textures in the background) by applying Sobel to detect edges in the image. However, if the edges in the background are not precisely the same, they are recognized in the subtraction algorithm as differences in the image. This causes the subtracted differences to be misclassified.

One of the reasons why edges are not exactly the same is because of path inaccuracy. When we tested the off-the-shelf crawler, we noticed it does not move the same distance every time. The slight differences in path accuracy mean that the crawler is in a different position when it comes to points of interest. These differences try to be fixed by aligning the images, but this sometimes leads to skewing of edges which appears as differences after subtraction.

Another limitation to subtraction is a lack of consistency with lighting. We noticed that since washers reflected lighting, the only edges that appeared were the ones that had light reflecting off of them. This means that a washer may only be seen as a half-circle and that would make classifying washers more difficult because not all of the washers would look similar.

#### **4.4 MACHINE VISION FUTURE WORK**

##### ***Deep Learning Neural Networks:***

Deep learning is a type of neural network that we would like to utilize in the LMA Crawler project. It is a neural network that consists of many layers. The many layers would allow the system to detect smaller features of an object that may seem incomprehensible to humans, but they have been shown to perform better than traditional machine learning algorithms when they have more training data. [2] Since we are saving all of the defect images to increase our database, we will have plenty of training data. We would use deep learning neural networks for image alignment and object detection.

##### ***Image Alignment***

Instead of using OpenCV functions for warping images for alignment, we want to experiment using a deep learning neural network to align images better. This will hopefully prevent line skewing that causes inaccuracies in subtraction. [3]



### ***Object Detection:***

In the Inspection Script, we currently use a regular neural network for classification. We want to experiment using a deeper network to see if we have better classification accuracy. All we need to change is the amount of layers and nodes in the layers, which OpenCV allows us to change very easily because we set the layer sizes when we create the network. [4]

### ***Include More Defects:***

For the first phase of the project, we focused on two foreign objects (washer and tape) to prove the concept of object detection and classification. We chose these two foreign objects because they are two of the three most common foreign objects found in airplane bays (as seen in the Figure 19). We decided to exclude shavings (second most common foreign object) because it would be difficult to differentiate between a shaving and a line. For the next phase, we plan to include more defects.

### ***Consistent Lighting:***

As mentioned in the subtraction inaccuracy section, inconsistent lighting causes reflections to appear differently on reflective objects (ex. washers). If the lighting is too low, defects are also not able to be detected. Therefore, we plan to create a solution which will increase the lighting and keep it consistent with the entire image of the camera.

### ***Camera with better resolution:***

One of our limitations is the distance the crawler can be from the specific areas of interest before taking pictures. We would be able to cover more area if we had a camera that allowed us to take higher resolution pictures from farther. The higher resolution would only be needed for taking

pictures of scenes. The rest of the streaming feed can be at whatever resolution is reasonable for streaming reliably.

Using a different camera would also mean that we would not be using the Pixy Camera. The Pixy Camera was initially selected because it can run machine vision and machine learning algorithms directly on the camera. This made the machine vision and machine learning development easier, but we also noticed that it had performance limitations. If our script took too long to complete, the Pixy Camera would disconnect and stop the inspection. Using the Stream Processing Server described in the following section, we would have to adapter the machine vision and machine learning code to run on the server instead of on the camera.

### ***Separating Stream Processing Server and Application Subscribers:***

Right now we use QT Creator to create an application with a GUI that allows us to choose to run the training or inspection script. After they choose the script, then the Android tablet can start the inspection. With this system, we have an unnecessary reliance on the GUI application running on our computer.

We hope to create a new C++ program that would run on a server and handle all machine learning and machine vision with the OpenCV library. This Stream Processing Server application could be set to open whenever the server starts and we could have multiple servers with load balancing. The program would handle everything we do already on QT Creator which is the edge detection, subtraction, and classification. It would also have an output stream that could be subscribed to. This stream would have the live feed from the camera on the pixy and would include overlays when defects are detected. Then, we could have the Play Store or App Store application, and a Web Server that subscribe to the processed feed from the Stream Processing Server.

Finally, we would remove the necessity to directly select the training or inspection scripts from the Stream Processing Server, and instead be able to select it from the Play Store or App Store application. This would also allow the User Interface to have multiple scripts to select for different airplane bays or other scenarios.

### ***Network Card:***

The crawler vision system currently is connected serially to the computer to send movement and pixy camera commands. We plan to attach a network card to the Arduino so that we can communicate via WiFi. Using a wireless network will extend the range of the crawler. Another issue is that the serial communication often disconnects, which shuts down the whole inspection. If we were to use a TCP connection via WiFi, we would be able to ensure the reliability of commands. [5]

## Chapter 5: System Interfaces

The crawler system has several subsystems that communicate with each other in order to complete inspections. The system was designed with the intention of supporting more than just a single crawler. There are many design decisions behind these choices that will be elaborated on briefly.

### 5.1 SYSTEM INTERFACES DESIGN

The subsystem of the crawler are as follows: the pixicam, website, database, Android device, and the Arduino.

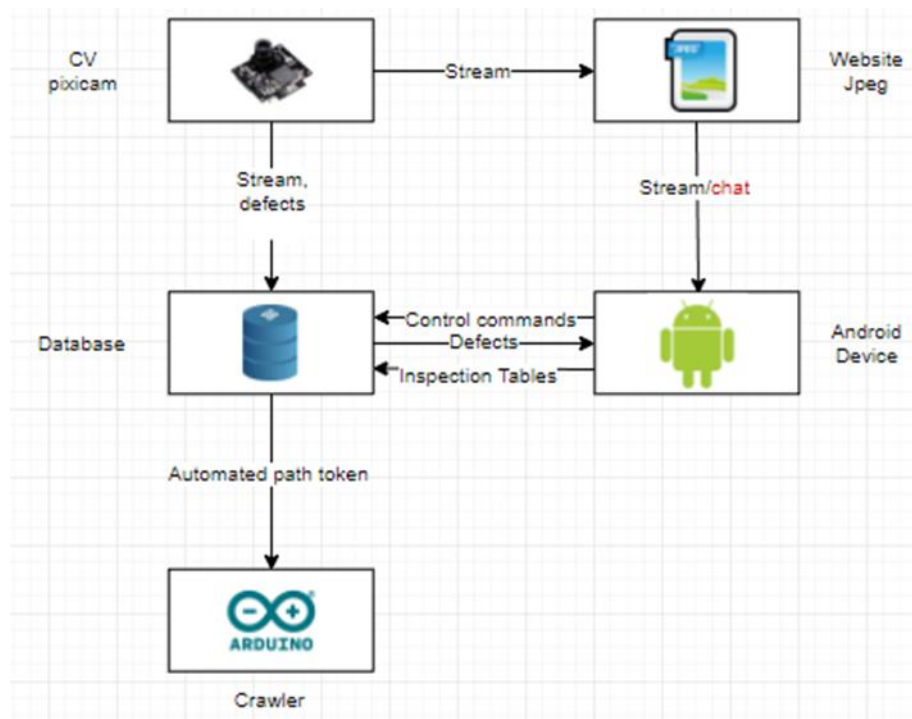


Figure 5.1.1: System Component Map

The pixicam subsystem is responsible for the machine vision and machine learning aspects of the crawler system. This subsystem has the job of capturing images, modifying the images, and

pushing the images to the website subsystem, as well as populating the database with its findings. This is beneficial for a system design as it encapsulates all visual processing into a single subsystem and simply pushes its results to other subsystems.

The Website subsystem receives and broadcasts the current image as a live stream. The intended purpose of this system is to display the feed to multiple clients while remaining open to future development options. This system is currently the most prone to change, the website currently only displays a feed but will likely be a hub of communication to allow interested parties to communicate directly with the inspector. The main advantage of this subsystem is its multiplicity, many crawlers can push a stream to the website and many clients may pull the feed from any crawler.

The Database subsystem is the central hub of communication between various components of the system and has the primary responsibility of storing inspection results. The Database is the component that is least likely to change throughout the course of development (that is in terms of language, not tables), so using it as a form of communication allows us to easily interchange various other subsystems without affecting unrelated subsystems. The database is currently written in Microsoft SQL, this decision was made based off of SQL's popularity and abundance of online resources. This subsystem stores information from inspections, stores all crawler paths, uses tables to communicate current state, and executes instructions (e.g. manual control, start inspection instructions).

The android device is the user interface subsystem and interacts directly with the inspector. An Android device was chosen due to its wide availability, the programming language it is written in (Java), and the tools it offers that aid in rapid development. This subsystem is responsible for displaying information directly to the inspector. Additionally, the Android device allows the

inspector to make quick simple changes to information that may have been stored inaccurately by the vision system.

The Arduino subsystem bridges the gap between the physical crawler and the software portion of the crawler. The Arduino receives different instructions that pertain to different physical components on the crawler. The Arduino subsystem has three main Responsibilities: receive new commands, execute commands, and communicate it's current state.

## **Chapter 6: Mechanical Design**

The goal of the prototype Crawler that will carry the vision system is to maneuver vertically and overhead in order to have maximum access to bays. The summary of major design components and the simulation performed for the flow shell design will be covered in this section.

### **6.1 DESIGN OVERVIEW**

The method of achieving the goals stated above include a track driven vehicle with a propeller mounted in the middle to provide a down force to the surface in which it is currently driving on. The figure below shows the current design configuration. Each track is driven by its own low RPM, high torque motor with an integrated gearbox. The propeller in the middle of the vehicle is sized such that it gives adequate thrust to maintain traction on vertical and overhead surfaces. A flow shell is shown surrounding the diameter of the propeller to protect the blades as well as to optimize flow such that optimal thrust is achieved. Analysis of multiple shapes and configurations of this flow shell have been performed using ANSYS flow simulation software. Section 09 Flow Simulation will provide an overview of the methodology for arriving to the shape of the shell moving forward. Additionally, the figure below shows the analysis of the forces needed for vertical maneuvers. The assumption made here is that the interface between the tracks and the surface that the crawler is currently climbing on is that of a Polystyrene on Polystyrene interface. This interface combination was selected because the value that it yields is .5, which is in the middle of the slippery interface to very grippy interface spectrum.

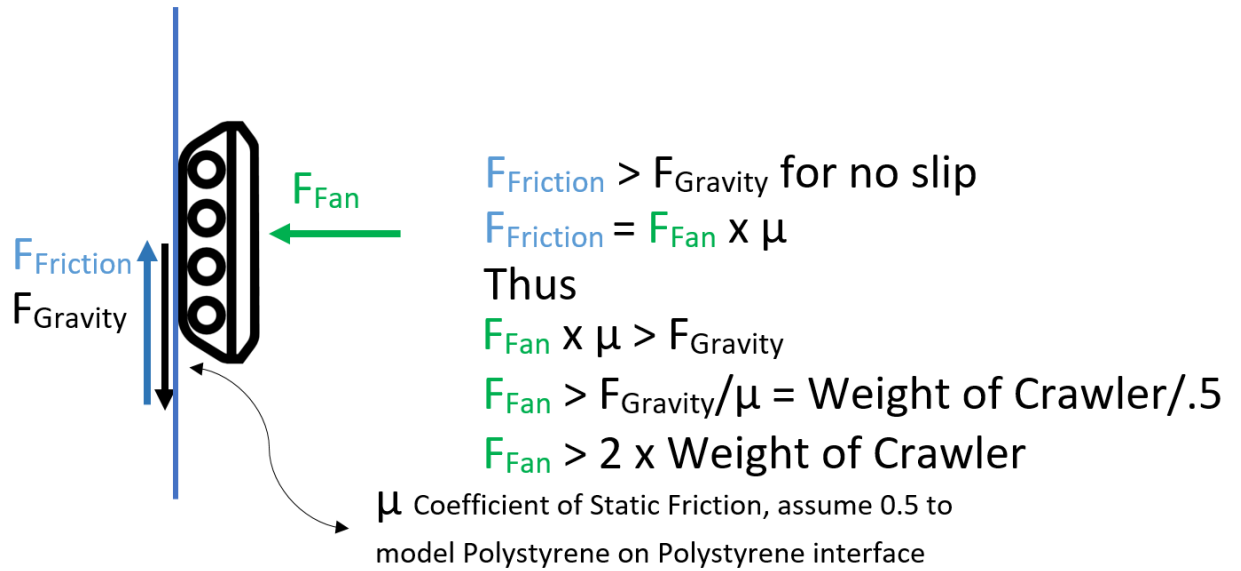


Figure 6.1 1: Thrust Needed for Vertical Maneuvers

The figure below shows similar analysis for overhead maneuvers.

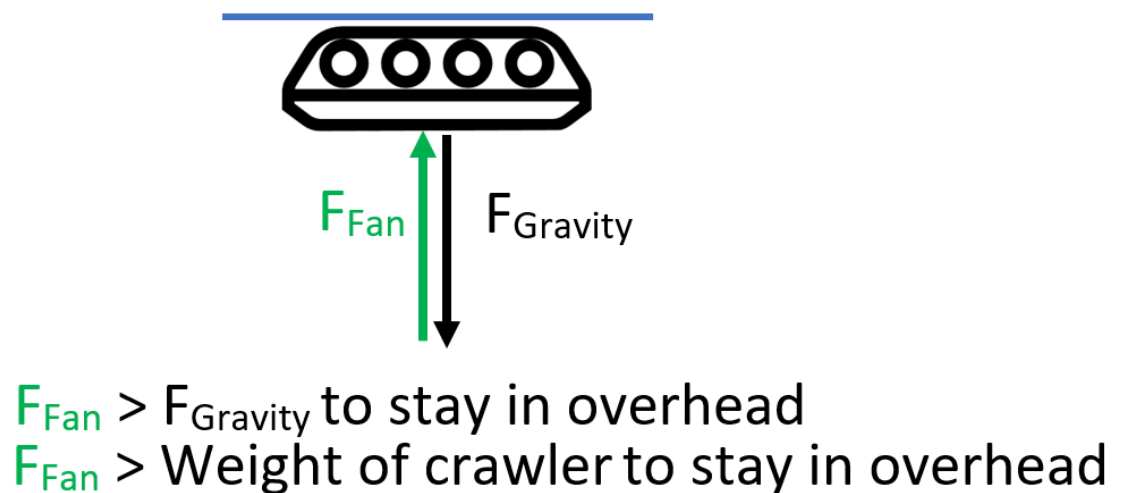


Figure 6.1 2: Thrust Needed for Overhead Maneuvers



Note that the force needed for vertical maneuvers is more than that required for overhead maneuvers thus making this the larger design constraint.

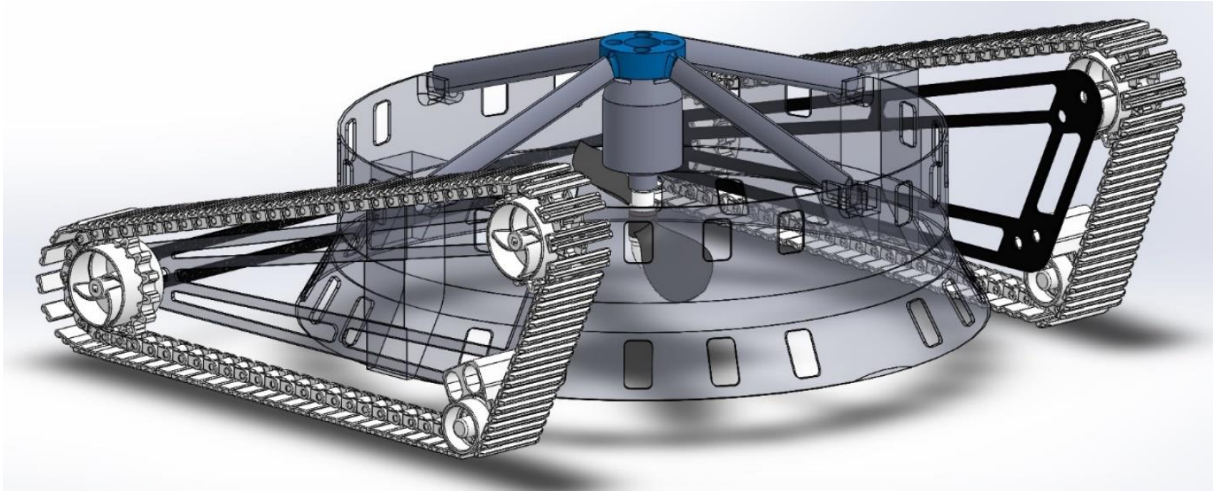


Figure 6.1.3: Current Crawler design.

## 6.2 COMMERCIAL OFF THE SHELF COMPONENTS

Below is a list of all of the components that were purchased for the build of the prototype:

- Arduino Microcontroller from Robot Shop Crawler
- 60A Electronic Speed Controller for Propeller Motor
- Power 25 42Amp Outrunner Motor
- 4 Blade Propeller
- Vex Track Kit
- DC Power Supply for Track Motors
- 12V DC Track Motors
- Airfoil Shape Extruded Aluminum

- Tinker Toy Orange Circle Piece
- Hardware
- 3500 PSI rated Epoxy

### **6.3 MOTION CONTROL**

An Arduino microcontroller is used to interpret commands from the User Interface/control system for executing paths and manual commands. This microcontroller is not onboard. A tether containing motor contactors, power, and communication for the vision system will be in place for the current phase of the project.

### **6.4 BRUSHLESS MOTORS AND ELECTRONIC SPEED CONTROLLER CONFIGURATION**

An electronic speed controller (ESC) with a 60 amp capacity was used because it proved to be adequate to run power 25 outrunner brushes motor. This motor was selected due to its published recommended use for 3.5lb aircraft which means that its net output thrust should be greater than 3.5lbs. This ESC was compatible with Arduino commands which uses existing code to emulate a remote control drone receiver. The figure below shows the configuration of the Arduino micro controller, the ESC, motor, and battery.

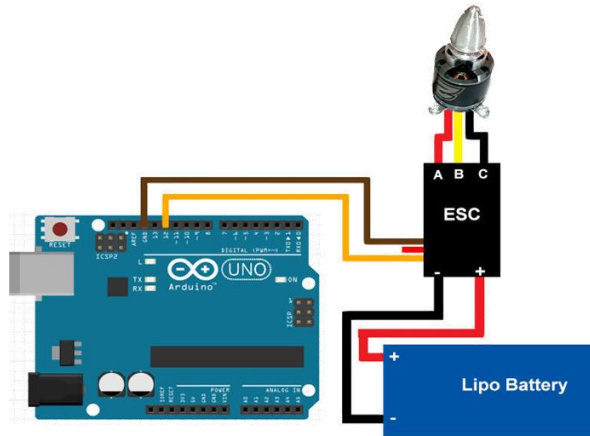


Figure 6.3.4:Arduino/ESC/Motor Schematic

## 6.5 FRAME COMPONENTS

Sections of the frame included profiles that were cut out of carbon fiber to support the tracks and attach to the flow shroud. The figure below shows these frame component's profile and location on the crawler.

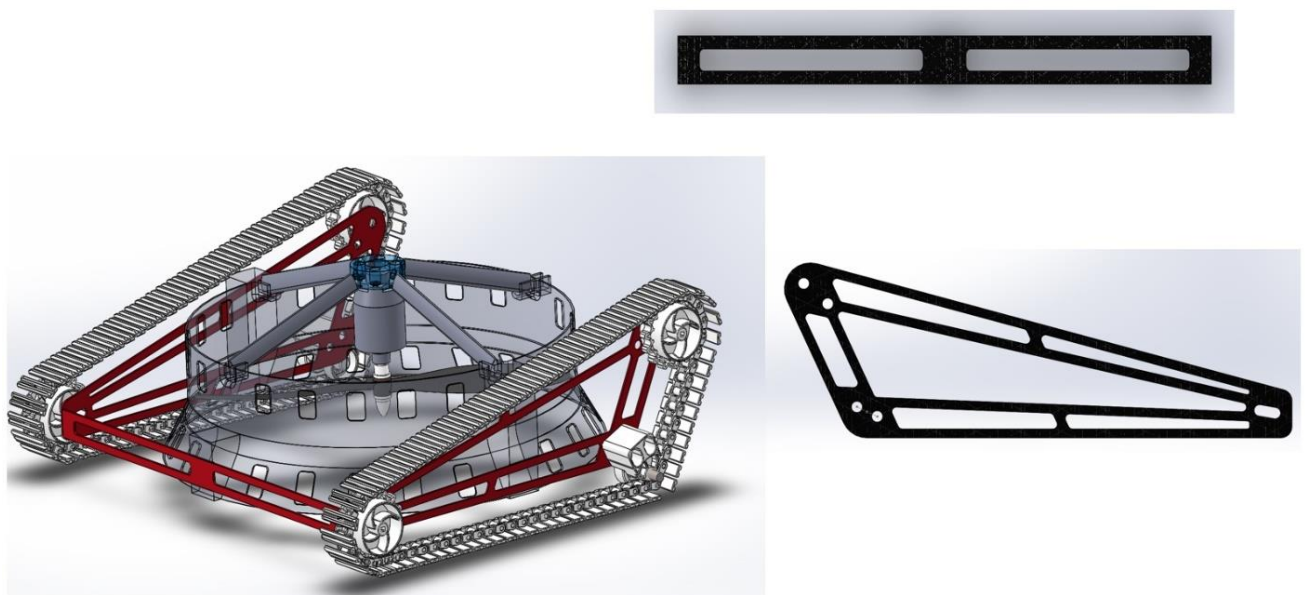


Figure 6.5.1: Carbon Fiber Frame Components

## **6.6 FLOW SHROUD**

A Flow Shroud component was integrated into the design of the crawler in order to increase the efficiency of the propeller and motor being used. The shape of this component was arrived to through using flow simulations as described in Section 6.9.

The shroud was 3D printed using ABS material using a Stratasys Dimension Elite 3D printer in 4 sections due to work envelope constraints. Some features of the shroud include sections for the carbon fiber frame components to be mounted as well as mounts for the propeller motor mount supports. Additionally, the cutouts that appear around the shroud were later covered in tape in the effort to reduce weight. The figure below shows the shroud sections that were printed as well as where it is located (in red) in the crawler.

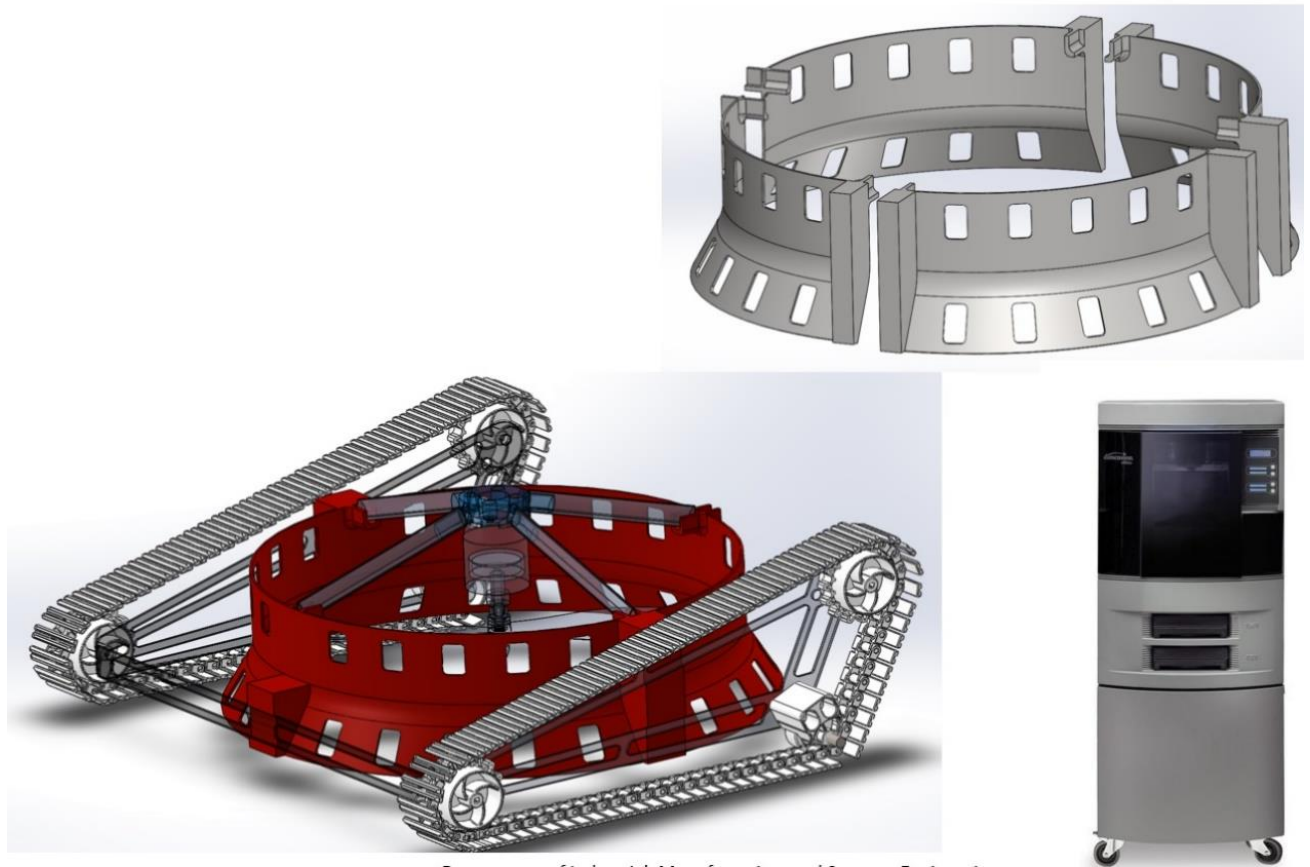


Figure 6.6.1: Flow Shroud (Red)

## 6.7 MOTOR SUPPORTS

The propeller motor is supported by some extruded aluminum in the shape of an airfoil. This shape was selected in the effort to reduce the amount of flow impedance introduced by this structural component. This extruded aluminum shape is commercially available and is shown in the figure below.

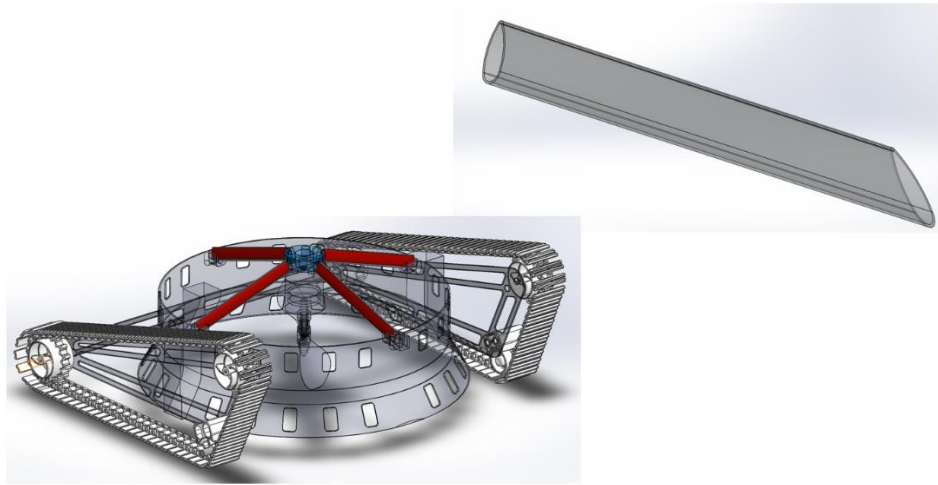


Figure 6.7.1: Propeller Motor Supports

## 6.8 STEREOLITHOGRAPHY 3D PRINTED PARTS

Parts of this prototype were produced using a FormLabs Form1+ stereolithography (SLA) 3d printer. A “tough” resin was selected due to its improved mechanical properties from the standard resin. The figure below shows the different components that were printed using this method and their description.

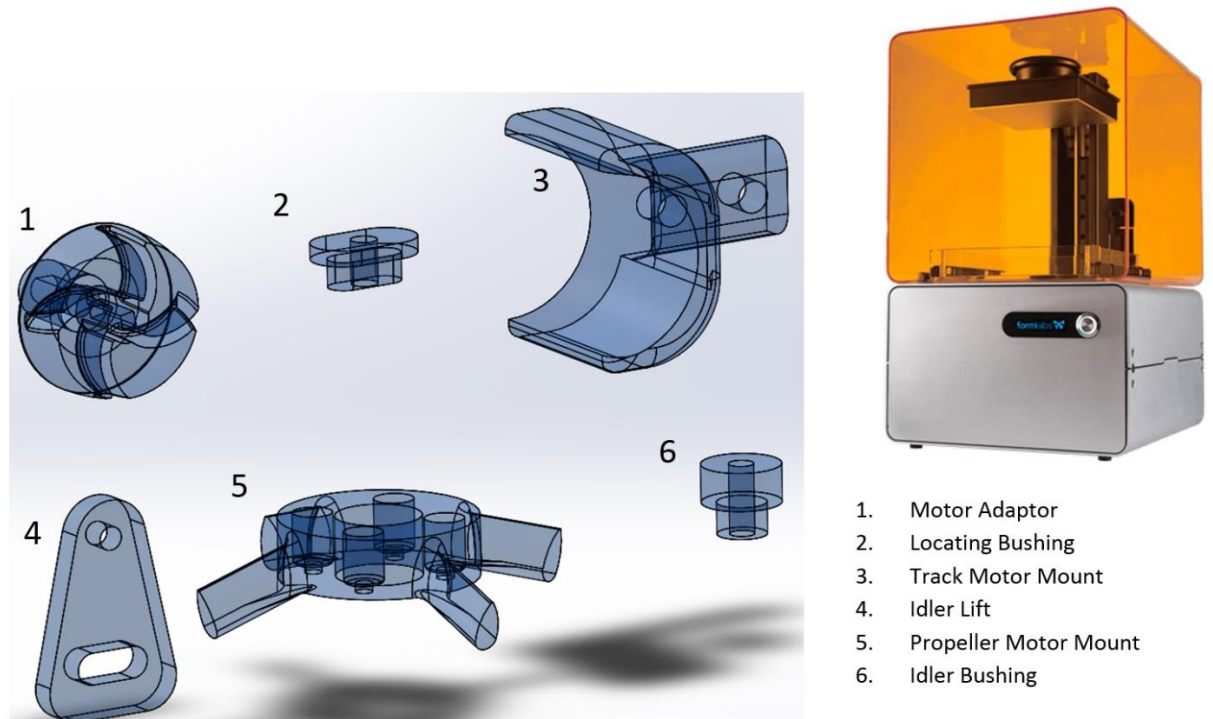


Figure 6.8.1: SLA 3D Printed Components

## 6.9 FLOW SIMULATION

A cylinder of 16 inches in height and 18 inches in diameter was defined as the domain. A rotating domain representing the propeller was set up in the middle of said cylinder. This cylinder was set to rotate at a speed of 100 rad/s. Top surface of cylinder was defined as the inlet, bottom surface was defined as outlet, but the initial velocity at these boundaries was equal to 0 m/s. Remaining surface was defined as a wall. All simulations were run for a total of 0.5 seconds, enough time for the flow to stabilize. Average velocity at outlet surface was calculated. This was used to find the optimum results.

In the first simulation, no shell was simulated, only the domain and the propeller. An average velocity of 0.347 m/s. The figure below shows the geometry simulated.

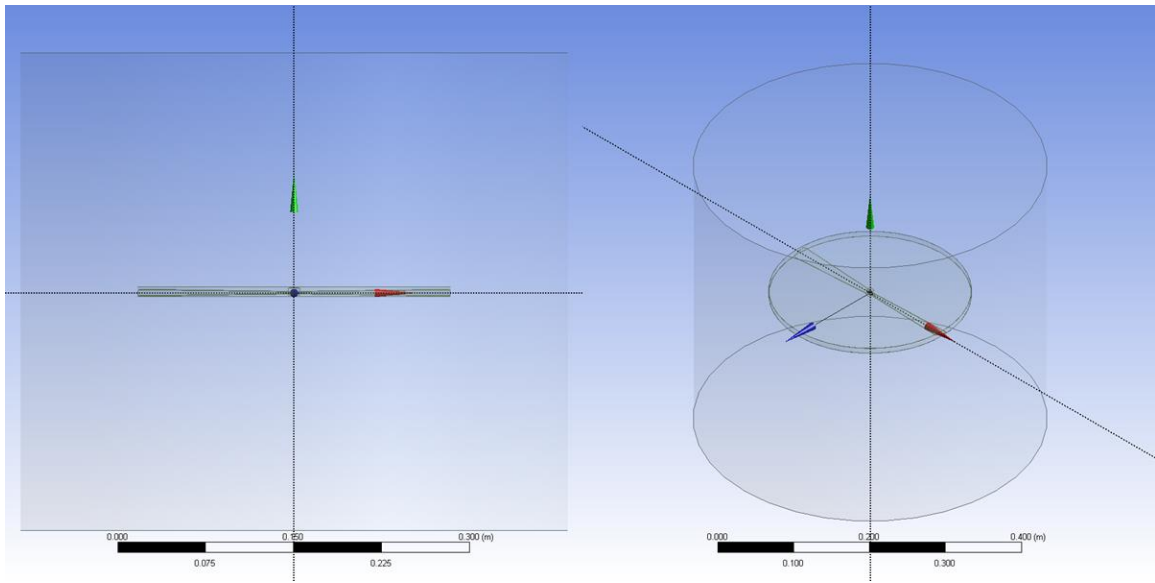


Figure 6.9.1: Geometry Simulated

Next, a shell of the same height of the propeller was introduced to the model. This is shown in the figure below. Average velocity remained the same, although velocity vectors seemed to show more order and a more uniform direction in general.



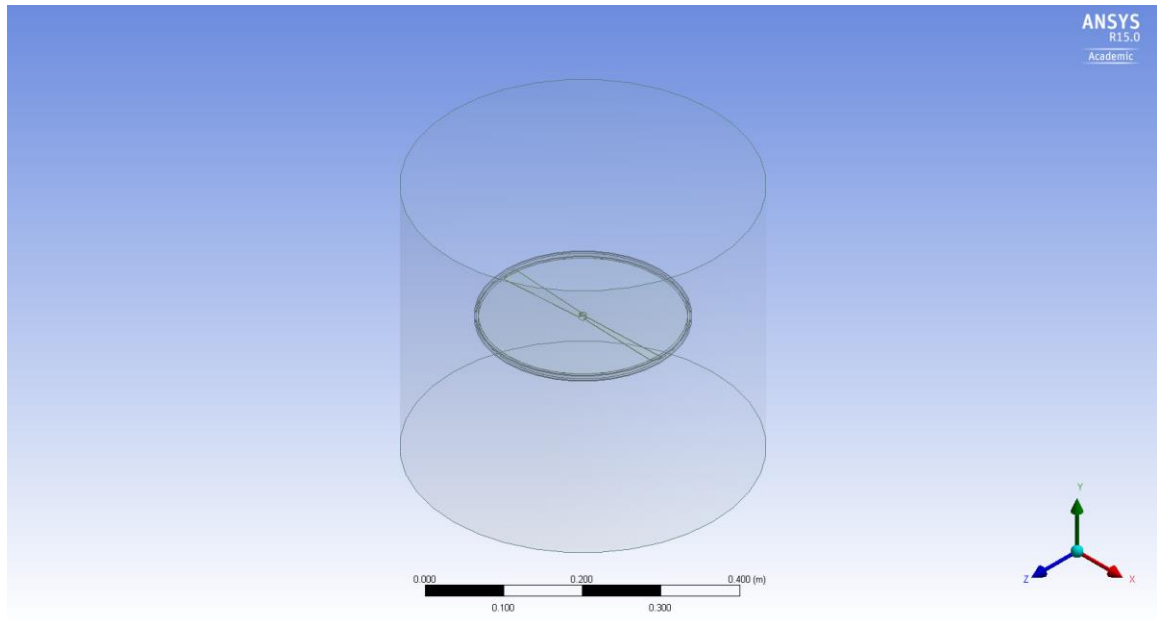


Figure 6.9.2: Geometry with Shell

The size height of the shell was then increased to see the impact of the shell height on the results. Shell height did not seem to have an impact on the velocity average. However, velocity vectors seemed to be more uniform as far as direction. The figure below shows the model after the shell height was increased.

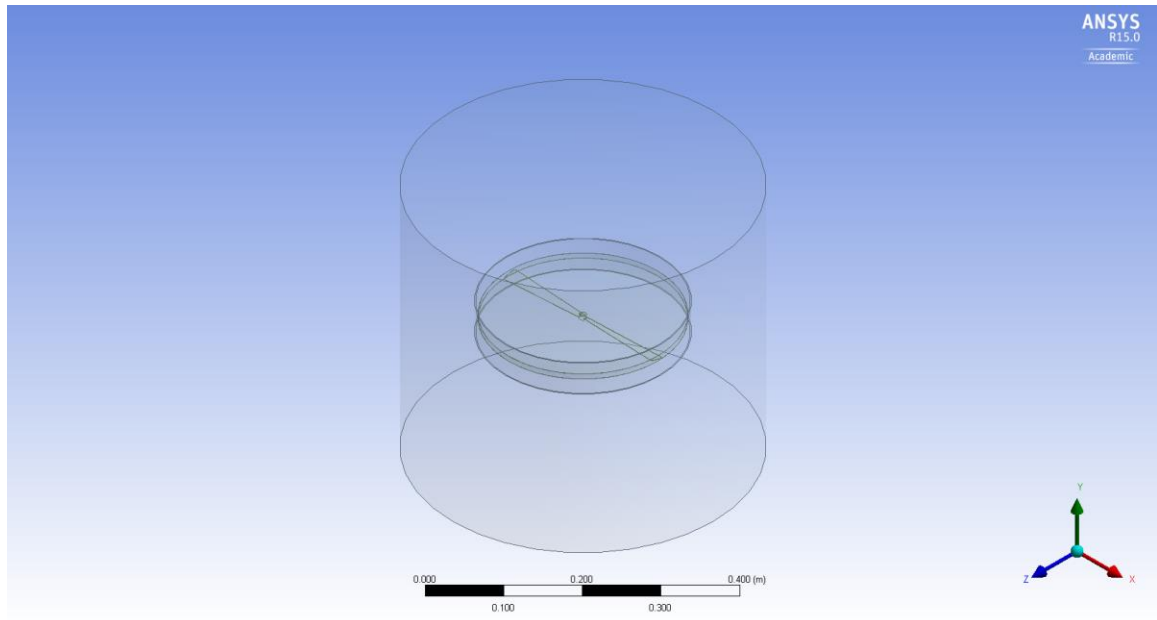


Figure 6.9.3: Geometry with Increased Shell Height

The distance between the shell and the propeller was increased. A comparison between the previous model and the one with longer distance can be seen in the figure below. Although the radius difference is of 5mm, a decrease of 0.02 m/s, or 6% was observed. From this, the first main design parameter was defined. It was observed that the higher the distance between the shell and the propeller, the lower the overall outlet velocity.

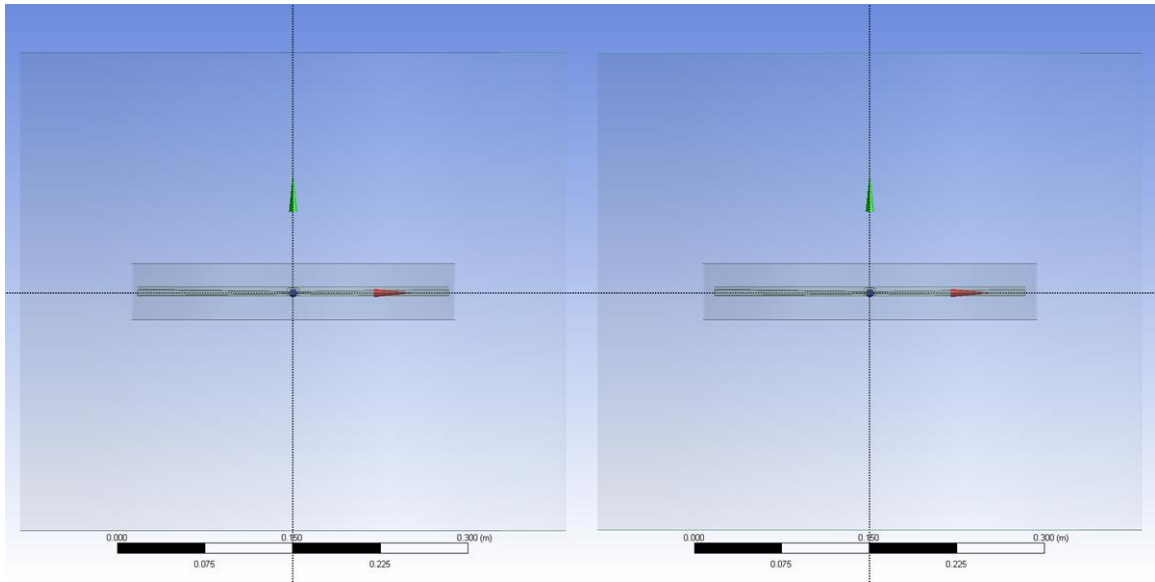


Figure 6.9.4: Geometry with Increased Distance Between Shell and Propeller

After the distance between the shell and the propeller were fixed, the angle of the shell was varied. The figure below shows an example of the shell angle being varied. An increase of 0.026 m/s was observed when the angle was varied to 15°. After that, the angle was increased to 30°. Another increase, now of 0.013m/s was observed. An overall increase in outlet velocity of 12.6% with respect to the control model was found. No significant improvement was found when increasing the angle to 45°. 30° angle model was used to move forward.

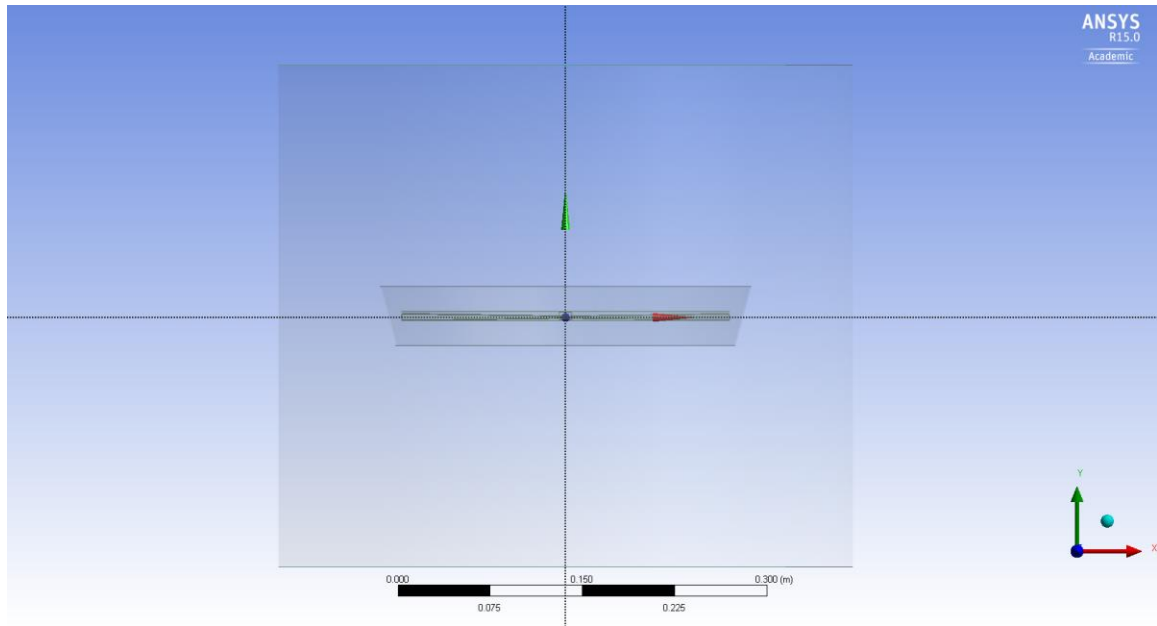


Figure 6.9.5: Shell with a 30 Degree Lead-in

After the optimum angle was found, the height of the shell with respect to the propeller was changed. The figure below shows how the shell was brought upwards. A further increase of 0.02 m/s was observed, yielding an overall improvement of 18.3% with respect to the control model.

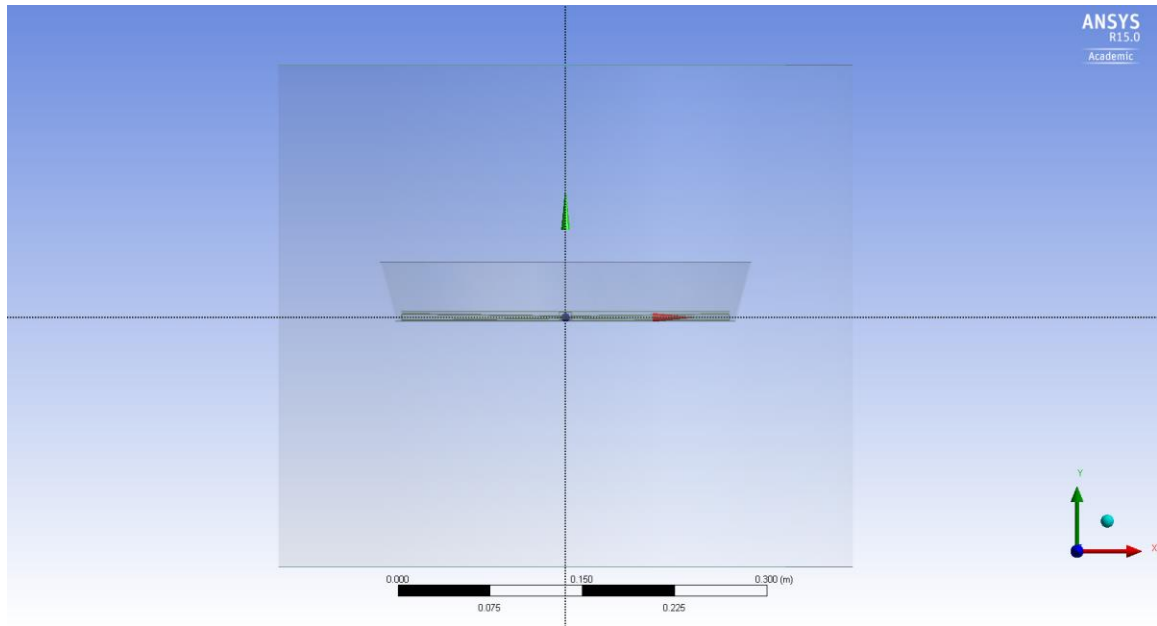


Figure 6.9.6: Propeller Placement Raised

Finally, a composed shell was modelled. This was done by maintaining the upper part of the shell where the previously explained model was, and adding a bottom shell with no angle. This bottom part was added to help all vectors of velocity be more uniformly oriented by not allowing them to go to the sides instead of downwards. The figure below shows the final model of the shell. This shell helped yield an outlet velocity of 0.428m/s. An overall improvement in velocity of 23%.

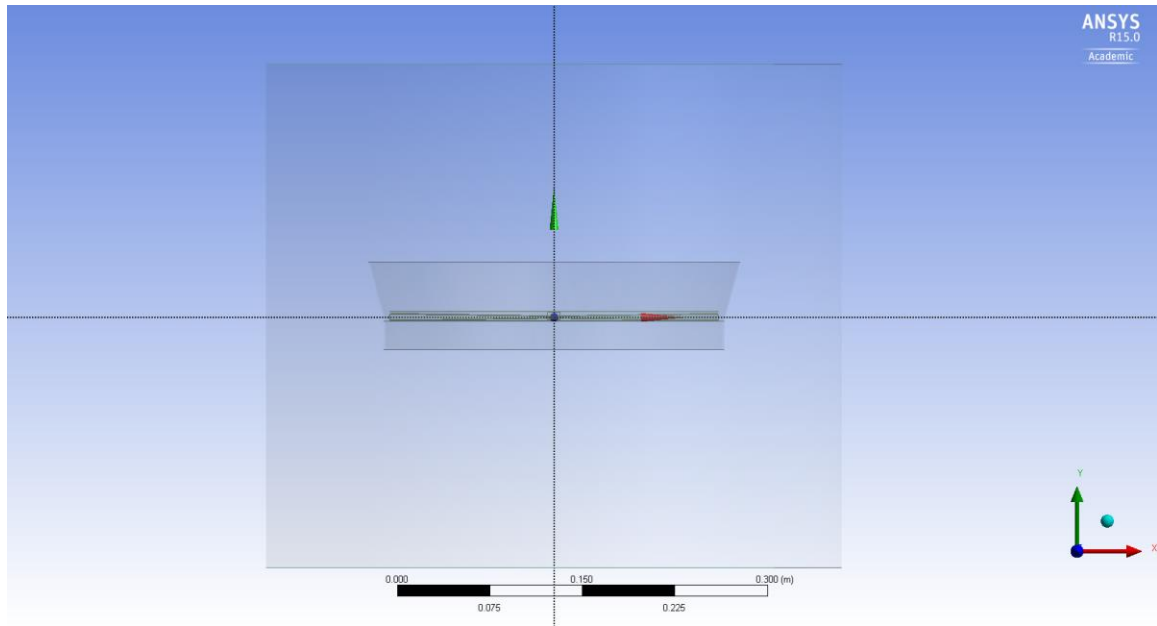


Figure 6.9.7: Final Geometry and Propeller Placement Used

## 6.10 PROTOTYPE/RESULTS

The as built prototype of the crawler mechanism is shown in Figure 6.10.1. The performance of the Crawler was evaluated with the criterion shown in the tables below:

Table 6.10.1: Position Performance

Experiment Positioning Mechanism		
Position	Time	Held position?
Horizontal	30sec	Passed
Horizontal	1min	Passed
Horizontal	3min	Passed
Vertical	30sec	Passed
Vertical	1min	Passed
Vertical	3min	Passed
Ceiling	30sec	Passed
Ceiling	1min	Passed
Ceiling	3min	Passed

Table 6.10 2: Obstacle Surmounting Performance

Experiment Crawler Mechanics		
Position	Time	Passed obstacle
Barrier	Horizontal	Passed
Barrier	Vertical	Passed
Barrier	Ceiling	Passed
Pipe	Horizontal	Passed
Pipe	Vertical	Passed
Pipe	Ceiling	Passed
Harness	Horizontal	Passed
Harness	Vertical	Passed
Harness	Ceiling	Passed

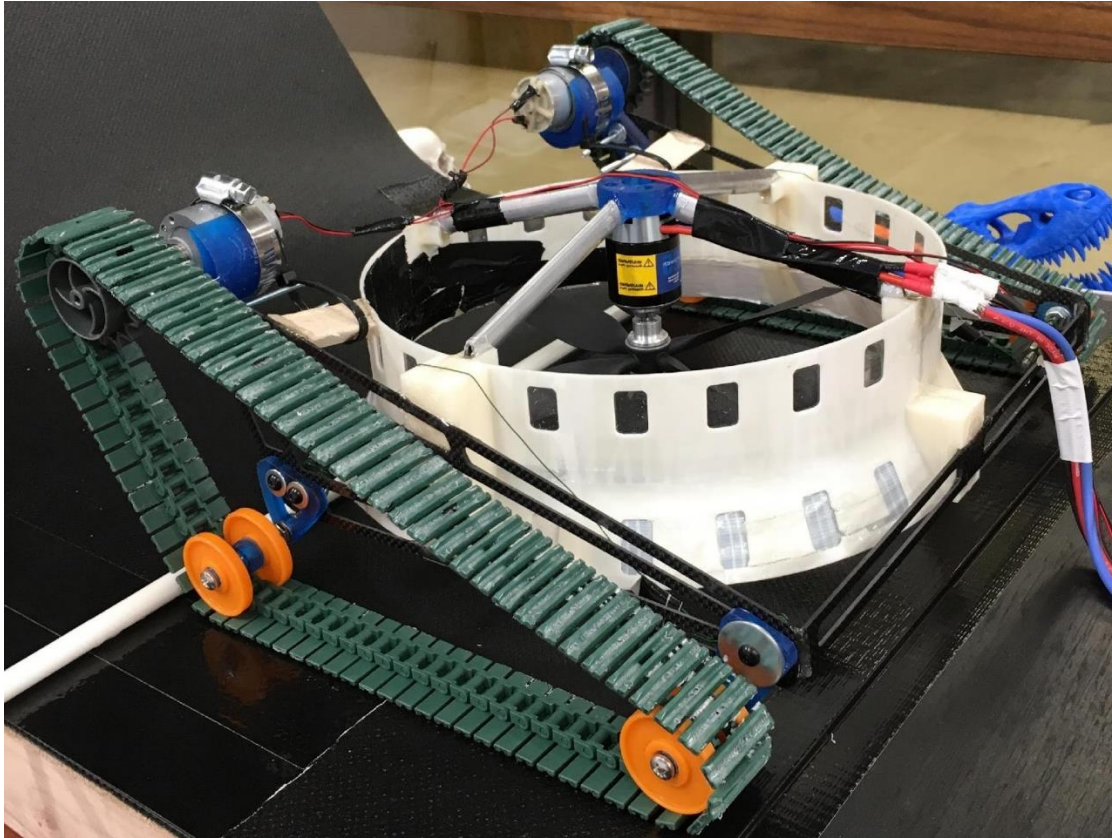


Figure 10.1.1: Working Built Prototype

The reference manual for the setup of the crawler prototype is included in the Appendix section of this document.



## **Chapter 7: Conclusions**

After the completion of Phase 1 of the crawler project, we have learned that we can successfully identify objects in a known environment and a vehicle can successfully maneuver vertically and overhead while overcoming obstacles in those orientations. Future phases can include further refining the machine vision/machine learning portion of this project as well as reducing the size of the crawler while preserving the vertical and overhead maneuverability in addition to wireless operation.

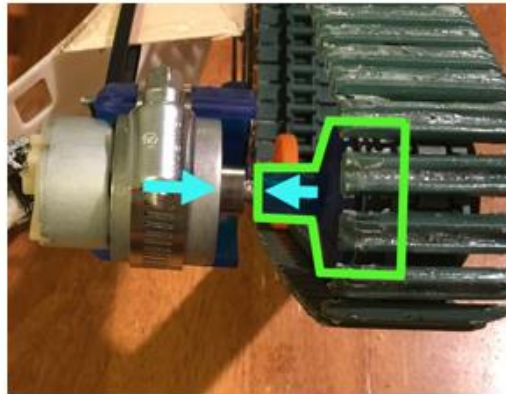
## References

- [1] <http://www.rroij.com/open-access/performance-analysis-of-canny-and-Sobeledgedetectionalgorithms-in-image-mining.php?aid=43752>
- [2] <https://www.analyticsvidhya.com/blog/2017/04/comparison-between-deep-learning-machinelearning/>
- [3] <https://github.com/MarekKowalski/DeepAlignmentNetwork>
- [4] [https://docs.opencv.org/2.4/modules/ml/doc/neural\\_networks.html](https://docs.opencv.org/2.4/modules/ml/doc/neural_networks.html)
- [5] [https://www.diffen.com/difference/TCP\\_vs\\_UDP.](https://www.diffen.com/difference/TCP_vs_UDP.)
- [6] Quan, Qiquan, and Shugen Ma. “Development of a Modular Crawler for Tracked Robots.” *Advanced Robotics*, vol. 25, no. 13-14, 2011, pp. 1839–1849., doi:10.1163/016918611x584712.
- [7] Jiang, Quansheng, and Fengyu Xu. “Grasping Claws of Bionic Climbing Robot for Rough Wall Surface: Modeling and Analysis.” *Applied Sciences*, vol. 8, no. 1, 2017, p. 14., doi:10.3390/app8010014.
- [8] Agarwal, Vimal, et al. “An Innovative Approach to Pipeline Health Monitoring Using Crawler Robot with PVDF Based Probe.” *Biomechanics / 752: Robotics*, 2012, doi:10.2316/p.2012.752-027.
- [9] Ito, Kazuyuki, and Haruo Maruyama. “Semi-Autonomous Serially Connected Multi-Crawler Robot for Search and Rescue.” *Advanced Robotics*, vol. 30, no. 7, 2016, pp. 489–503., doi:10.1080/01691864.2015.1122553.
- [10] Lubica, Miková, et al. “The Design of Tracked Mobile Robot for Non-Urban Environment.” *Applied Mechanics and Materials*, vol. 816, 2015, pp. 288–293., doi:10.4028/www.scientific.net/amm.816.288.
- [11] Yu, Zhiwei, et al. “Vertical Climbing Locomotion Of A New Gecko Robot Using Dry Adhesive Material.” *International Journal of Robotics and Automation*, vol. 32, no. 4, 2017, doi:10.2316/journal.206.2017.4.206-5054.
- [12] Lee, Giuk, et al. “Series of Multilinked Caterpillar Track-Type Climbing Robots.” *Journal of Field Robotics*, vol. 33, no. 6, 2014, pp. 737–750., doi:10.1002/rob.21550.

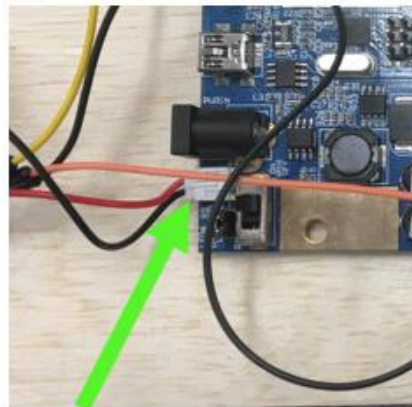
## Appendix

### Crawler Reference Manual

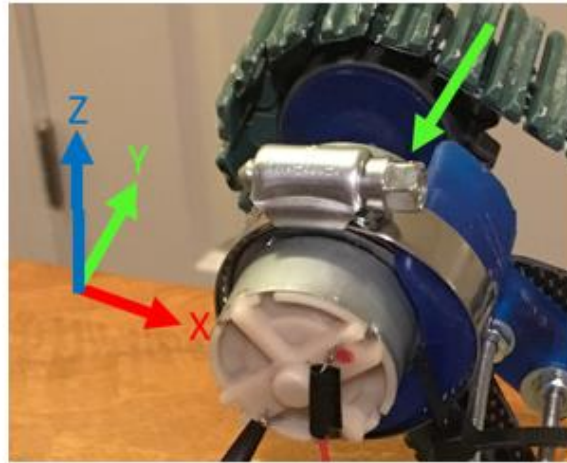
1. Download Arduino software.
2. Plug in cable connecting Arduino to computer. Turn Arduino's power switch to the "On" position.
3. Make sure the Arduino has the latest sketch uploaded to it that enables track and propeller control.
4. Download Putty or any type of program that provides a serial communication interface.
5. Ensure fasteners on idlers are not loose
6. Make sure the motor shaft adapters (outlined in green) are properly seated and secured to the motor shaft (press in the direction of arrows):



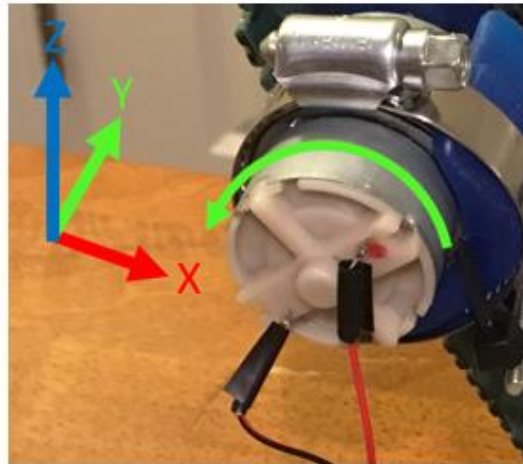
7. Connect power supply to the wires coming from the white Arduino connector at this this location:



8. Check voltage by taking a reading from a multimeter at the track motor terminals while pressing the key for forward track movement. Adjust power supply to achieve 12V at the terminals.
9. Check alignment by lifting crawler so the tracks are not in contact with surface and pressing the key for forward movement (w). Tracks are aligned if the forward motion is smooth and not jerky. If the movement is not smooth, adjustment in the Y direction of the image below can be achieved through adjusting the hose clamp (green arrow) around the motor and shifting it in the Y direction.



10. Once proper alignment is achieved rotate the motor along the +/- Y axis of rotation (picture) to pick up slack in the track tension. Tighten the hose clamp to secure position of motor once alignment and tension is set.



11. Ensure that the propeller is properly fastened to the motor shaft and tighten if necessary.
12. Check all of the joints that are held together with epoxy to ensure that they have their structural integrity.

Maneuver the crawler using the following keys:

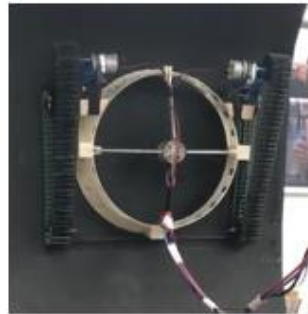
W: Forward

A: Left

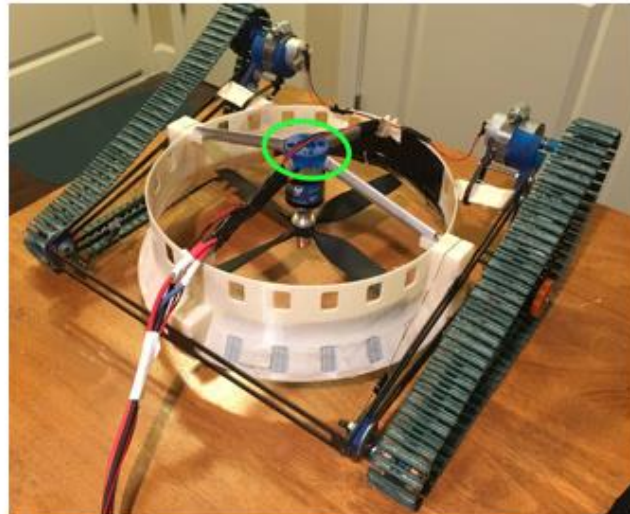
S: Backward

D: Right

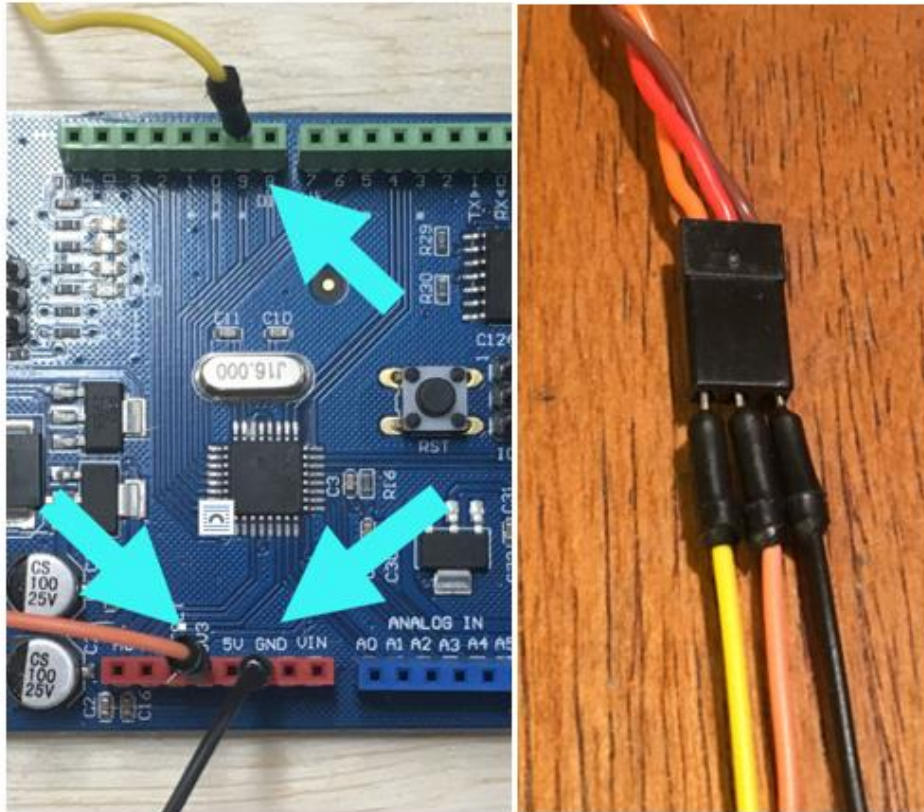
B: motor brake torque to prevent back idling/holding vertical position (image below).



Handle the crawler from the blue motor mount (highlighted below) and avoid handling it from the tracks or other less sturdy components.



13. Check that the wires connecting the ESC and Arduino are at the proper pin locations and in the proper locations at the receiver connector as shown in images below:



14. Make sure battery is fully charged then connect it to the ESC. The propeller motor should give audio confirmation that the Arduino is connected and ready to send signals.

Propeller motion is controlled through the following keys:

H: Propeller initiation and speed increment

L: Speed decrement

M: Propeller motion termination.



## **Vita**

Francisco Joaquin Martinez graduated high school at Antonito Jr./Sr. High School in Antonito, Colorado in 2007. He then earned his B.S. in Mechanical Engineering from Colorado State University in 2013 and then became a Project Engineer for Wolf Robotics where he worked in industrial robotics to automate fabrication processes, mainly heavy welding and surface finishing. He then moved to El Paso, Texas in 2016 in pursuit of his M.S. in Systems Engineering after which he will then begin employment with Lockheed Martin Aero.

Typist email address: [f.j.martinez.engr@gmail.com](mailto:f.j.martinez.engr@gmail.com)