PRODUCT DEVELOPMENT FILE

Group 11

Caroline Aitken, C. Tyler Booth, Usman Dauda, Claire Lizotte

Table of Contents

[Table of Figures 4](#_Toc479325680)

[Problem Definition, Specifications and Planning 5](#_Toc479325681)

[Problem Definition 5](#_Toc479325682)

[Design Requirements 5](#_Toc479325683)

[Planning 7](#_Toc479325684)

[Conceptual Design Phase 10](#_Toc479325685)

[Conceptual Design Development 10](#_Toc479325686)

[Drive System 10](#_Toc479325687)

[Cube Location 12](#_Toc479325688)

[Cube Manipulation 13](#_Toc479325689)

[Pyramid Location 14](#_Toc479325690)

[Pyramid Manipulation 14](#_Toc479325691)

[Cube Storage 15](#_Toc479325692)

[Sensors/Collision Avoidance 16](#_Toc479325693)

[Concept Evaluation 17](#_Toc479325694)

[Drive System: 17](#_Toc479325695)

[Cube Location: 17](#_Toc479325696)

[Cube Manipulation: 18](#_Toc479325697)

[Pyramid Manipulation: 18](#_Toc479325698)

[Pyramid Location: 19](#_Toc479325699)

[Cube Storage: 19](#_Toc479325700)

[Sensors/Collision Avoidance: 19](#_Toc479325701)

[Product Design Phase 20](#_Toc479325702)

[Refining of Selected Concepts 20](#_Toc479325703)

[Drive System: 20](#_Toc479325704)

[Cube Location: 21](#_Toc479325705)

[Cube Manipulation: 21](#_Toc479325706)

[Pyramid Location: 23](#_Toc479325707)

[Pyramid Manipulation: 24](#_Toc479325708)

[Cube Storage: 26](#_Toc479325709)

[Sensors/Collision Avoidance: 26](#_Toc479325710)

[Design Meeting Review 26](#_Toc479325711)

[Engineering Analysis 27](#_Toc479325712)

[Driving Motors 27](#_Toc479325713)

[Wheels 30](#_Toc479325714)

[Fly Wheel 31](#_Toc479325715)

[Servo Motor 31](#_Toc479325716)

[Detailed Documentation 34](#_Toc479325717)

[Drawings 34](#_Toc479325718)

[Justification of Parts 60](#_Toc479325719)

[PPN7 DC Motor 60](#_Toc479325720)

[RS550 Motor 60](#_Toc479325721)

[Adafruit 1142 High Torque Servo 61](#_Toc479325722)

[40t and 15t Bevel Gears 62](#_Toc479325723)

[VersaPlanetary Gearbox 62](#_Toc479325724)

[Cytron 13A Motor Controller 63](#_Toc479325725)

[Tread Options 63](#_Toc479325726)

[Hall Effect Sensor 63](#_Toc479325727)

[The IR sensor 64](#_Toc479325728)

[IR Sensor Encasing 64](#_Toc479325729)

[IR Sensor Lid 65](#_Toc479325730)

[IR Sensor Assembly 65](#_Toc479325731)

[IR Distance Sensor 65](#_Toc479325732)

[Cube Holder 66](#_Toc479325733)

[Chute 66](#_Toc479325734)

[Bill of Materials 67](#_Toc479325735)

[Robot Code 69](#_Toc479325736)

[Pinouts 69](#_Toc479325737)

[Master Code 70](#_Toc479325738)

[Slave code 92](#_Toc479325739)

[Product Evaluation 99](#_Toc479325740)

[Objectives That Were Met 100](#_Toc479325741)

[Improvements 102](#_Toc479325742)

[Works Cited 103](#_Toc479325743)

[Information 103](#_Toc479325744)

[Photos 103](#_Toc479325745)

[Progress Report #1 105](#_Toc479325746)

[Progress Report #2 106](#_Toc479325747)

# **Table of Figures**

[Figure 1 - Critical Path Chart 8](#_Toc479326635)

[Figure 2 - Sweeper for cube manipulation 14](#_Toc479326636)

[Figure 19- RS550 motor18Figure 20- Different components of VersaPlanetary gearbox. 27](#_Toc479326637)

[Figure 19- RS550 motor18 27](#_Toc479326638)

[Figure 19- RS550 motor18 27](#_Toc479326639)

[Figure 25- Torque curves of different length (x-axis in m) and arm (y-axis in kg) combinations for all arms and for servo motorMass of arm 32](#_Toc479326640)

[Figure 25- Torque curves of different length (x-axis in m) and arm (y-axis in kg) combinations for all arms and for servo motor 34](#_Toc479326641)

[Figure 25- Torque curves of different length (x-axis in m) and arm (y-axis in kg) combinations for all arms and for servo motor 34](#_Toc479326642)

# Problem Definition, Specifications and Planning

## Problem Definition

*Need Statement:* The Power Pyramid needs to have a Tesseract underneath it.

*Goal Statement:* Design an autonomous mechatronic device that can obtain a tesseract power pack and place it under a pyramid.

Design Requirements

A list of customer requirements was created, and from these, a list of engineering specifications was generated. The list of specifications is summarized in the Quality Function Deployment (QFD) as the foldout attached to the end of the document. The chart has been broken into sections so that it is legible.

It is provided in the goal statement that the main requirement is to consistently get the cube under the pyramid and ensure that the solution created does not require human intervention to complete the task. To simplify the problem, the tasks were broken up into subtasks. The sub-tasks and their specific requirements are:

1. Ability to move around the course unimpeded by the Power Conduits.
   * Move Forward
   * Get over the Power Conduits
   * Turn
   * Not slip
   * Minimize size and mass
2. Locating and obtaining the tesseract.
   * Consistently locate the tesseract
   * Can manipulate the tesseract
   * Does not drop the tesseract
   * Accurately locates the tesseract
3. Locating and obtaining the pyramid.
   * Consistently locate the pyramid
   * Can manipulate the pyramid
   * Does not drop the pyramid
   * Accurately locates the pyramid
   * Can expose the bottom of the pyramid
4. Manipulating the pyramid and/or the cube to get the pyramid on top of the cube.
   * Consistently successful

Once the goal statement, QFD, and requirements were outlined, concepts of a solution could be generated that satisfies the need while remaining within the design requirements, so that the solution generated would be aligned with the customer requirements.

## Planning

*Program Plan*: To obtain substantial results with the assigned task within the tight schedule, ambitious goals were set out, with weekly deadlines. This would aid in enforcing a strong work ethic on the project, since time for completing the project while ensuring each step of the design process was properly followed was limited.

Table - Project schedule for design project

|  |  |  |
| --- | --- | --- |
| Week in Schedule | Dates | Tasks |
| 1 | Feb 12 – Feb 18 | Initial Group Meeting and Setup |
| 2 | Feb 19 – Feb 25 | Independent Brainstorming of Concepts |
| 3 | Feb 26 – Mar 4 | Problem Definition, QFD, Progress Report 1 |
| 4 | Mar 5 – Mar 11 | Concept Selection and Design |
| 5 | Mar 12 – Mar 18 | Build and Code, Progress Report 2 |
| 6 | Mar 19 – Mar 25 | Testing of the Design |
| 7 | Mar 26 – Apr 1 | Iteration, Debugging, Testing, Final Reports |
| 8 | Apr 2 – Apr 7 | Finish Up final tasks, Showcase |

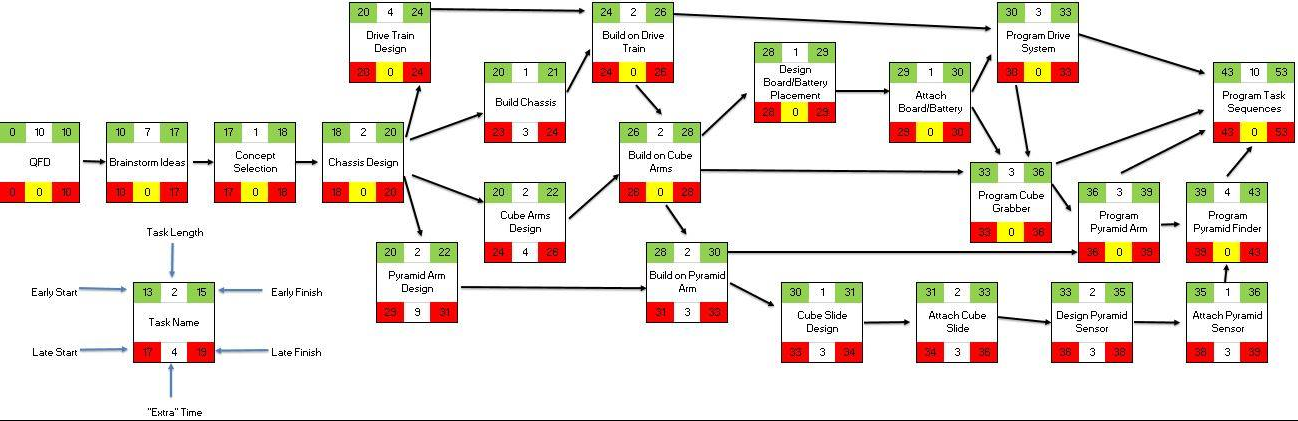
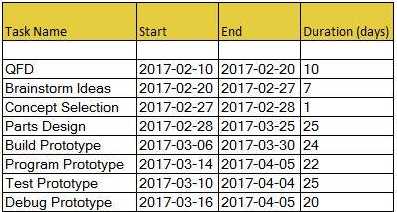


Figure 1 - Critical Path Chart

Table - Gantt chart tasks



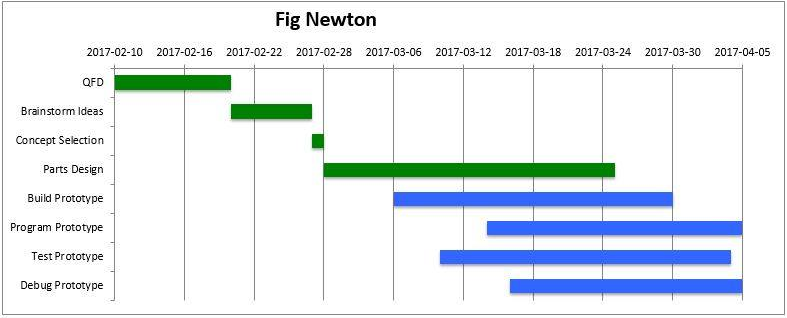


Figure 3 - Gantt chart

# Conceptual Design Phase

## Conceptual Design Development

For concept generation, the team initially brainstormed ideas individually, then came together to share and develop the ideas. The 4 sub-systems from the specification phase were further broken down. The various systems and the potential solutions for them are (Sketches of each concept are shown after it is stated):

### *Drive System*

1. 6 wheels, rocker system to keep chassis relatively level when going over bumps. This idea was discarded because it was thought that it would be too difficult to use motors and steer.



Figure 4 - 6 wheel rocker drive system

1. 3 Wheels, Triangular base, with the two wheels in line and powered and the third can rotate. The wheels in line are at the front to facilitate steering. This idea was discarded because it was the third wheel might catch on the power conduits and drag.

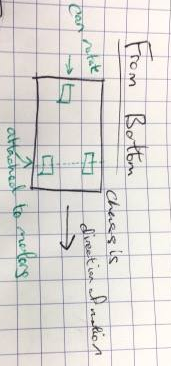


Figure 5 - 3 wheels, triangular base drive system

1. Tank drive to easily get over the Power Conduits without impedance.

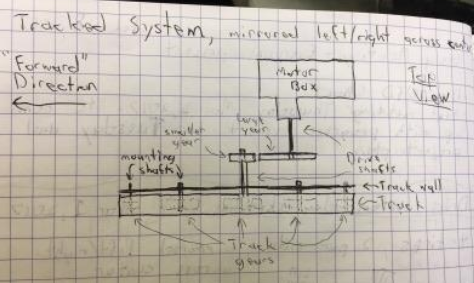


Figure 6 - Top view tank drive system

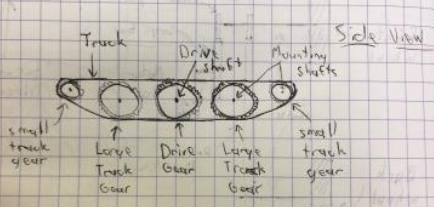


Figure 7 - Side view tank drive system

1. 4 Wheels like a car. Can skid steer. Large tires to overcome the Power Conduits.

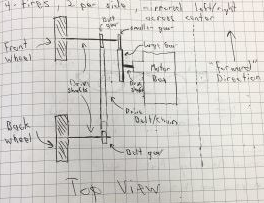


Figure 8 - 4 wheels drive system

1. 6 Wheels, all level
2. 8 Wheels, 4 like a car, then 2 smaller ones at the front and back to catch the chassis if it starts to tip going over the Power Conduits

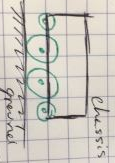


Figure 9 - 8 Wheel Drive System

### *Cube Location*

1. Hall sensor
2. Button that the cube bumps into

### *Cube Manipulation*

1. Grasper – linear actuator

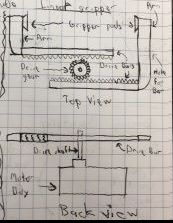


Figure 10 - Linearly actuated grasper for cube manipulation

1. Grasper – Rotational Actuator

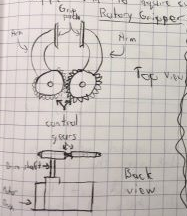


Figure 11- Rotationally actuated grasper for cube manipulation

1. Electromagnet
2. Pusher (comes from outside and pushed into cube holder)
3. The Sweeper. Hits the cube from the far side into a holding dish.

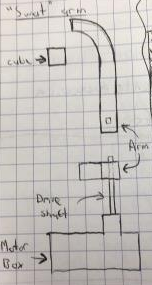


Figure - Sweeper for cube manipulation

### *Pyramid Location*

1. IR sensor
2. Methodically search the course

### *Pyramid Manipulation*

1. Rubber Graspers, angled to accommodate the slope of the pyramid, to pick the pyramid up.

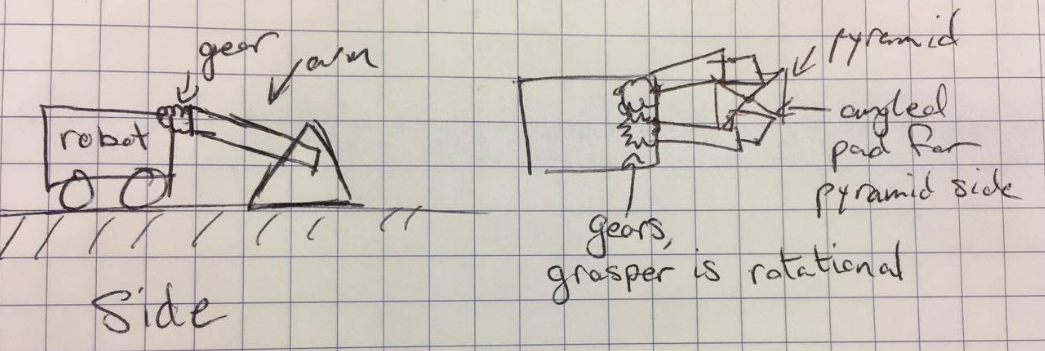


Figure 3 - Grasper to manipulate pyramid

1. Push pyramid until it is against the wall of the course, then have an arm to push the pyramid against the wall to tip it.

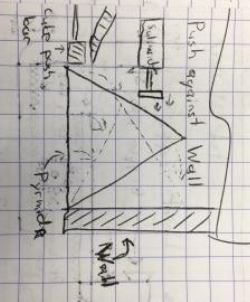


Figure 4 - Push pyramid against the course wall and use a solenoid to tip it

1. Make a wall that can slide out and the pyramid can be pushed against that by an arm
2. Fork lift the pyramid to expose the bottom

### *Cube Storage*

1. Keep it on the arm it was picked up by
2. Deposit it into a storage spot on the robot, and then pick it up again with the arm later
3. Deposit it into a storage spot with a trap door on the bottom that can be activated when required to drop the cube in a desired location.

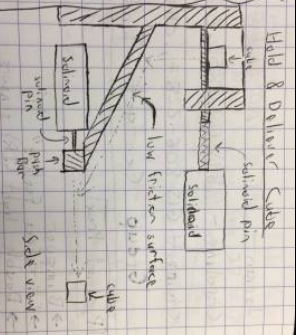


Figure 5 - Holding cube in spot with a trapdoor with push bar to get cube to desired location.

### *Sensors/Collision Avoidance*

1. Infrared Distance Sensors
2. Light Sensors
3. Ultrasonic

## Concept Evaluation

After discussing why select concepts were not viable solutions (stated when the concept was listed), decision matrices were made for individual sub-systems to objectively choose which concepts were the best. If there were only a few concepts for a sub-system, each concept was discussed and evaluated amongst the group to decide which to use.

### *Drive System:*

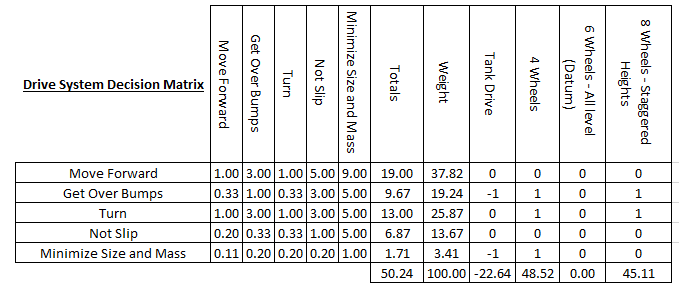


Figure 6 - Drive system decision matrix

The result of the decision matrix was to use the 4 wheels with skid steer. Skid steer is the easiest to implement and 4 wheels provides stability without taking up the amount of space more wheels or tracks would.

### *Cube Location:*

The Hall-effect sensor was selected to locate the cube since it was given, making finding and purchasing a button unnecessary.

### *Cube Manipulation:*

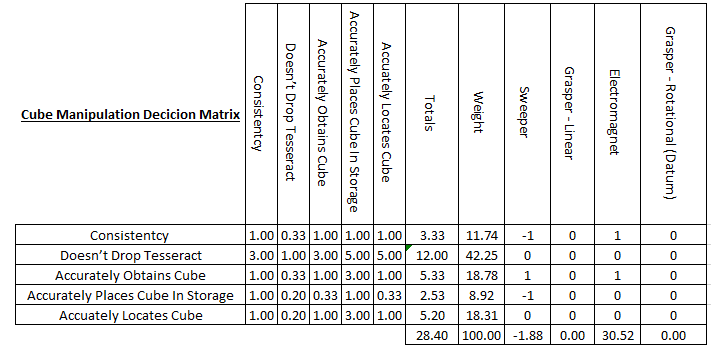


Figure 7 - Cube manipulation decision matrix

It was decided to use the electromagnet, as it could more consistently pick up the cube. The cube would not have to be in the exact position a grasper could require, as the electromagnetic force would draw the cube to it. Also, the electromagnet would work more consistent than hitting the cube with a gripper, which has a risk of knocking the cube into an unwanted position.

### Pyramid Manipulation:

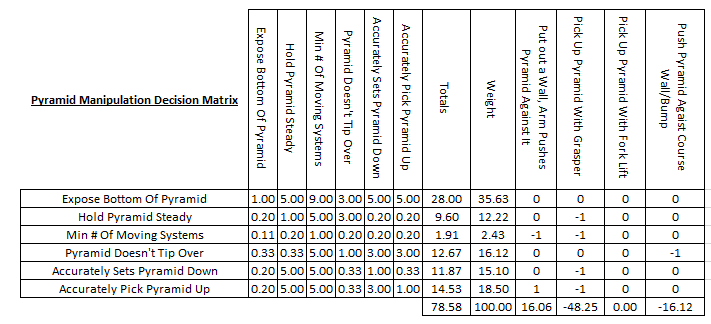


Figure 8 - Pyramid manipulation decision matrix

The result of the decision matrix was to have a wall which the pyramid can be pushed against to tip it. This was considered to have less moving parts than trying to lift the pyramid. The wall also has a greater chance of success than trying to push the pyramid against a course wall because there may be power conduits in the way that prevent the pyramid from getting to a wall.

### *Pyramid Location:*

It was decided by the group to use the IR distance sensor to locate the pyramid since it was given.

### *Cube Storage:*

The decision made on cube storage was to store the cube on the arm because it was thought that once the cube can be securely held, it eliminated the process of putting the cube down and having to pick it up again. It was also thought that a trap door, while ideal so the cube will not be dropped in transition, would be difficult to implement.

### *Sensors/Collision Avoidance:*

IR distance sensors were selected because they do not require the same time delay and are more consistent than ultrasonic sensors. Light sensors would be affected by the environment, and since the system would have to work in different lighting conditions, light sensors were not used.

# 

# Product Design Phase

## Refining of Selected Concepts

### *Drive System:*

Initially, the wheels were located outside the chassis directly attached to motors inside the chassis. To conserve space and to gear the wheels to produce more torque, the wheels were relocated to inside the chassis and the motors were placed in the middle of the chassis. The motors and back wheels were attached using gears and a chain. See figures 19 and 20 below.

The system is a rear wheel drive that turns using skid steer. 6-inch diameter wheels were chosen, since they can easily drive over the power conduits.

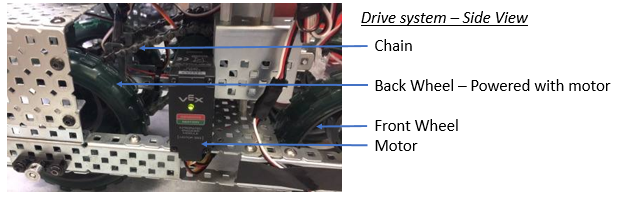


Figure 9 - Drive system side view



Figure 10 - Drive system bottom view

### *Cube Location:*

The cube is to be located by attaching a Hall-effect sensor on the end of the arm that obtains the cube. See figure 21 below.

### *Cube Manipulation:*

The arm of the robot was actuated by a servo motor that can rotate in the vertical plane (any plane perpendicular to the ground). It starts the course in a retracted position, then is put into an extended position. See figures 21 and 22 below. The end of the arm has the Hall-effect sensor on the far side of the cube, preventing the cube from being knocked off out of the course. A plate was attached to the arm, allowing it to hook on to the wall, so the robot cannot drift away from the wall without the cube. The arm would fall behind the cube and push it along if it is not grabbed right away. See figure 23 below.

The original concept was to have the electromagnet activate when the arm is by the cube. However, testing proved that the electromagnet was not strong enough to pick up the cube consistently, as it depended on the location of the magnet inside the cube. The design was changed to have another servo motor at the end of the arm. The servo motor has a support that rests parallel to the arm and is actuated to be perpendicular to the arm to grab the cube, effectively pinching the cube between the support and the hall sensor.

To increase the range in which the cube can be grabbed, another arm was added at the back of the robot to push the cube forward if it is behind the arm with the grasper. After driving forward about a length of the robot, the front arm lifts and the robot backs up, so that the cube is now in front of the robot. The arm comes down and continues to sweep the wall. The only exception to this is if the cube is in the very back corner, where the two walls meet. It was not coded into the prototype due to time constraints, but if the robot reached the far wall without getting the cube, indicating that the cube is in the very back corner or beyond the front-ward reach of the robot, the robot could turn around and the arms could come down on the right side, allowing the far end of the wall to be searched for the cube in the same manner described above, but in the opposite direction. This would allow the cube to be obtained everywhere except the two corners where both walls meet.

After the cube is obtained, the arm holding the cube retracts so it is at approximately a 130’ angle with where it got the cube. The cube remains held by the servo motor throughout the location of the pyramid phase. Retracting the arm to an obtuse angle allows the cube to rest against the arm so there is less of a chance of it being dropped.

Once the pyramid’s base is exposed, the arm will be brought up so that the cube is above the chute, then the servo will release the cube, allowing it to fall in the chute. See figure 24 below. The chute consistently deposits the cube, allowing the robot to back up and place the pyramid over the cube.

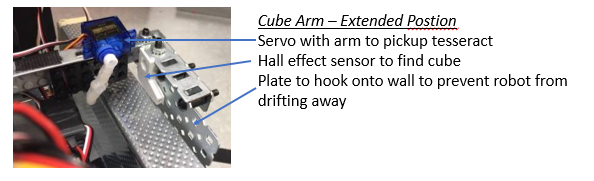


Figure 11 - Cube arm in extended position



Figure 12 - Cube arm in retracted position

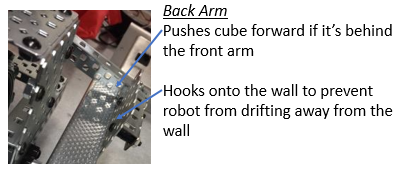


Figure 13 - Back arm

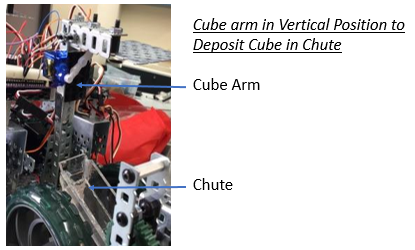


Figure 14- Cube arm in vertical position

### *Pyramid Location:*

Testing the IR sensor showed that the sensor picked up the signal even when not pointing at the pyramid. To narrow the focus of the sensor, a Lego case was built that consisted of a pinhole, a chamber, another pinhole, and then the IR sensor at the back of another chamber. This case was wrapped in aluminum foil, as it reflects IR waves, and electrical tape to hold the aluminum foil in place.

The pyramid IR sensor would be able to determine the direction the pyramid was in, and then a limit switch that is hit by the pyramid when found. There is another IR distance sensor that is looking forwards on the robot to verify that the pyramid is truly found, and not the signal reflecting off the walls. This works because the three sensors are staggered, so that when the limit switch is hit by the pyramid, the horizontal IR distance sensor cannot see the pyramid. So, if the limit switch is hit (which is in line with the pyramid IR sensor), and the horizontal IR distance sensor does not see an object, the pyramid is found. If both sensors detect an object, then it is the wall detected.

Testing the pyramid IR sensor, there was a narrow range in which the signal was detected, and the signal could be detected at a range of 14ft.

The original placement of the pyramid IR sensor was underneath the robot, with it pointing between the chute and the wheel. This location was chosen because it was directly in line with the arm to manipulate the pyramid. This arrangement often could not pick up the IR signal unless the pyramid was within 2 ft. Therefore, the location of the IR sensor was changed to the side of the robot with the pinhole in line with the front of the robot. The sensor would then pick up the signal at 14ft. However, the sensor is no longer in line with the pyramid. To account for this, the robot would turn right before manipulating the pyramid.

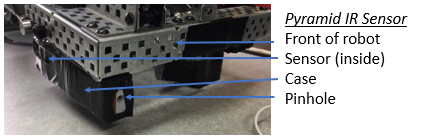


Figure 15 - Pyramid IR sensor

### *Pyramid Manipulation:*

The original concept was to have a wall slide out from the side of the robot, and an arm push the pyramid to tip it from the other side. However, it was quickly noticed that it was easier to construct a wall that moved in the vertical plane. Upon construction, it was realized that if the pyramid was tipped away from the robot, there was a much greater opening to get the cube under the pyramid. So, instead of having the wall on the side of the pyramid, the wall would be placed behind the pyramid after it has been located.

This was implemented by attaching an arm to a motor, and having the wall perpendicular to the arm at the end of it. An attempt using a servo motor was made, however it would not produce enough torque to lift the wall, even after the servo was geared up. So a drive motor was used, ensuring the positioning of the arm would still be consistent by always starting with the arm on the ground and then raising it a set number of encoder counts.

After the pyramid is against the wall, it still needs to be tipped. As it was no longer pushed from the side, and the arm that obtains the cube was constrained to move in the vertical plane, an additional arm was required. Adding another arm was difficult due to the size constraints. It was suggested by a group member to tip the pyramid with a wheel, as the tire provided the required grip to lift the pyramid and it did not have to hit the pyramid in a specific location/angle to work. This was implemented by adding a wheel to the arm with the wall with a motor to spin the wheel once the pyramid is in place.

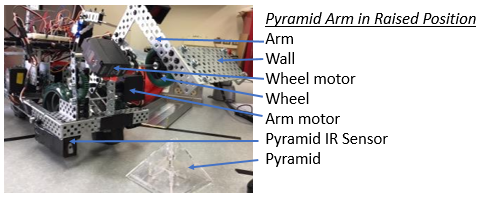


Figure 16 - Pyramid arm in raised position

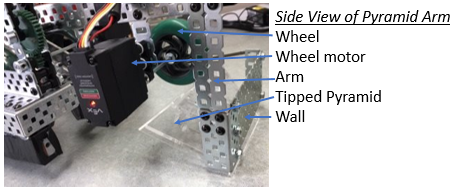


Figure 17 – Side view of pyramid arm

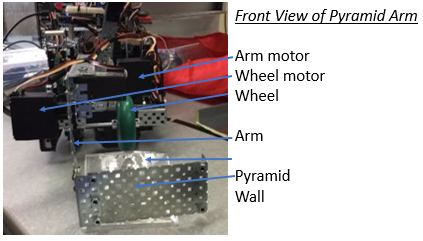


Figure 18 - Front view of pyramid arm

### *Cube Storage:*

Once the cube is obtained, it remains held at the end of the arm by one of the servo motors.

### *Sensors/Collision Avoidance:*

To remain next to the wall while the cube is being obtained, there are hooks on the ends of the arms to prevent the robot from drifting away from the wall.

The IR distance sensor on the left side of the robot prevents collisions with the wall.

### Design Meeting Review

Progress reports are attached at the end of the document.

Table - Design Review Meeting #2

|  |  |
| --- | --- |
| Potential Problem | Solution |
| The robot might drift away from the wall when it's trying to find the cube | Added plates to the ends of the arms to hook on the wall to prevent drifting away. |
| The IR sensor will pick up the pyramid signal even when the sensor is not directly facing the pyramid | Narrowed the sensors range by building a casing for it with a pinhole. The casing is wrapped in aluminum foil to prevent the signal from leaking. |
| There are large spaces of the wall where the robot cannot find the cube | Increased the range the robot can get the cube by adding in the back arm. |

## Engineering Analysis

### Driving Motors

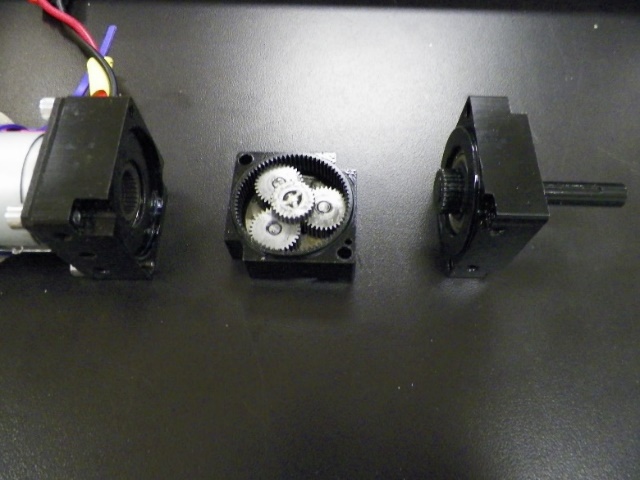
For the driving motors, a high torque configuration was required to minimize the strain of the robot. A requirement of the parts of the robot was to be long lasting, and the best way to protect the drive train motors was to reduce the strain of providing torque for the wheels of the entire robot. The motors used for the robot was the RS550 motor by Banebots. This motor offered a compact, high torque application that would draw small amounts of current when used on the small scale of the robot. This factor would impact other components of the robot such as the choice of motor controller, and supply voltage of the robot. At a maximum efficiency of 74.2W, the motors would draw 10.75A and 8.9V, a range where motor controllers could be found. For the gears in the design, a planetary gearbox was used, with an output gear meshed with a bevel gear, to change the direction of the motor’s output in the direction of the wheels. The gearbox chosen was the VexPro Versaplanetary gearbox because it offered a compact way to create the same gear ratio as the chain drive that was the original design plan. In addition, the Versaplanetary gearbox offered multiple stages of gear reduction, in order to increase the torque to the desired amount by changing the output gear ratio. To meet design needs, only a single stage was needed to drive the motors. To calculate whether this system would be enough to power the motors, the number of teeth required for the gears was chosen (15 and 40 teeth) and the reduction of the gearbox (10:1), and it was assumed the robot could be successfully driven if it could overcome a kinetic (sliding) friction coefficient (µk) of 1, while staying within the stall torque of the robot. The calculations are as follows:

Figure 20- Different components of VersaPlanetary gearbox.

From http://4.bp.blogspot.com/-7zEhOJrUYyU/UPxBn9SHvsI/AAAAAAAAAvQ/S96Tj0sQO8I/s1600/100\_0616.JPG

Figure 19- RS550 motor18Figure 20- Different components of VersaPlanetary gearbox.

From http://4.bp.blogspot.com/-7zEhOJrUYyU/UPxBn9SHvsI/AAAAAAAAAvQ/S96Tj0sQO8I/s1600/100\_0616.JPG

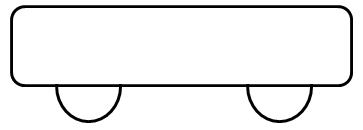
Figure 19- RS550 motor18

Figure 19- RS550 motor18

Assume:

* Even weight distribution on 4 wheels of robot
* µk=1 (for highest possible coefficient of friction)

FBD:



**Weight**

**Normal Forces**

**Friction forces**

**Weight**

**Normal Forces**

**Friction forces**

0.0762m

Figure 21- Cytron single DC motor controller0.0762m

From even weight assumption:

From µs=1 assumption:

Total moment about wheels:

Gear ratio of drive train=

Required torque at motor=

According to the motor specifications sheet, 0.3312 N·m is the stall torque, and the required output of the motor is below the stall torque value. Therefore, the motor will function properly for the robot. To decide the motor controller that would be used to regulate the motors, the torque constant Kt,measured in N·m/A, was used to determine how much current will be drawn by the motors when they are driven.

For current draw of RS550 motor:

Total current draw=

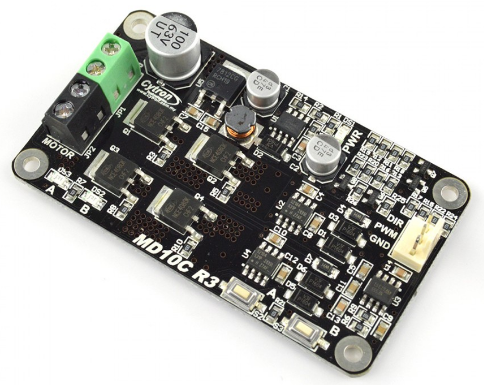
To ensure the motor controllers stay intact, motor controllers were used that could handle up to 13A. The chosen motor controller was the Cytron single DC motor controller. This motor controller was chosen for the ability to handle 13A continuously, and spikes of up to 30A (for 10s).

Figure 21- Cytron single DC motor controller

Figure 22- 6" performance wheel with 1/2 inch boreFigure 21- Cytron single DC motor controller

### Wheels

The wheels chosen were 6-inch performance wheels by the supplier Andymark. The reasoning behind this choice was largely due to the adaptability of the wheel. With a half-inch bore, and mounting holes near the center of the wheel, hubs with different bore sizes can be attached to the wheel, making the wheel reusable to different sizes in the event changes needed to be made to other components. Also, the wheel can easily change the tread between different materials, which was determined to be a very excellent feature of the robot; letting the user pick out what tread would best suit the environment the robot was going to be used in. This would make the robot more efficient for the user because it would minimize slipping in the wheels of the robot as it maneuvers over conduits or uneven ground. Three of the tread options are different kinds of rubber (natural, carbox nitrile, and styrene butadiene), which would help the robot traverse over multiple kinds of smooth surfaces. The last tread option is polyester, which would create low friction if the user wished. The ability to easily change to different materials would be a big step in the robot being able to meet the custom needs of the user.

Figure 22- 6" performance wheel with 1/2 inch bore

Figure 23- Coefficient of friction of polyurethane against different materialsFigure 22- 6" performance wheel with 1/2 inch bore

### Fly Wheel

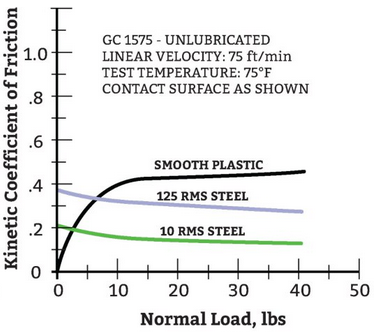
For picking up the pyramid, a material with a high coefficient of friction against the glass surface of the pyramid was desired. Through the design process, a polyurethane flywheel was settled on largely due to the increasing coefficient of friction it had with increasing loads on smooth plastic. In the scope of the design project the smooth plastic would be the pyramid that the robot has to raise in order to place the cube. To spin the wheel system, a PPN7 DC motor was selected. This motor was chosen for its low weight(10.0), and high rotation speed (11605RPM). Using the flywheel and motor, a high friction contact surface could be created that would create enough constant frictional force to raise and lower the pyramid.

Figure 23- Coefficient of friction of polyurethane against different materials

Figure 24- Adafruit LLC 1142 motorFigure 23- Coefficient of friction of polyurethane against different materials

### Servo Motor

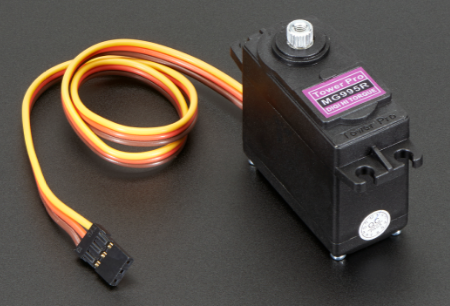
When choosing a servo motor, a motor was needed that would have enough torque to lift all the arms of the motor. Instead of choosing specific servo motors based on the torque created by each arm, it was decided to choose a motor that would be able to lift all three arms. Using the mass properties of the Solidworks model, an upper limit of the torque required for each arm was created, so that a proper servo motor could be selected. The chosen motor was the Adafruit LLC 1142 servo motor, with a torque rating of 988.62mN·m, so that it could control all the arms of the robot to specific positions. The motor was also very compact and lightweight, which would help in meeting the requirement of keeping the robot under 3kg. The calculations for why the specific servo motor was selected are as follows:

Figure 24- Adafruit LLC 1142 motor

Figure 24- Adafruit LLC 1142 motor

Assume: Entire mass of arm is a point mass a distance away equal to the length of the arm. This is to create an upper bound estimate of the torque.

Axis of Rotation

Axis of Rotation

Length of arm

Length of arm

Mass of arm

Figure 25- Torque curves of different length (x-axis in m) and arm (y-axis in kg) combinations for all arms and for servo motorMass of arm

Table 4- Length and mass of different robot arms

|  |  |  |
| --- | --- | --- |
| Arm | Mass (kg) | Length (m) |
| Rear | 0.064 | 0.2000 |
| Middle | 0.1024 | 0.16279 |
| Front | 0.1327 | 0.15132 |

Using the equation for finding moments:

Table - Values of moments due to static friction

|  |  |
| --- | --- |
| Arm | Moment about servo rotor (mN·m) |
| Rear | 125.57 |
| Middle | 163.53 |
| Front | 196.99 |

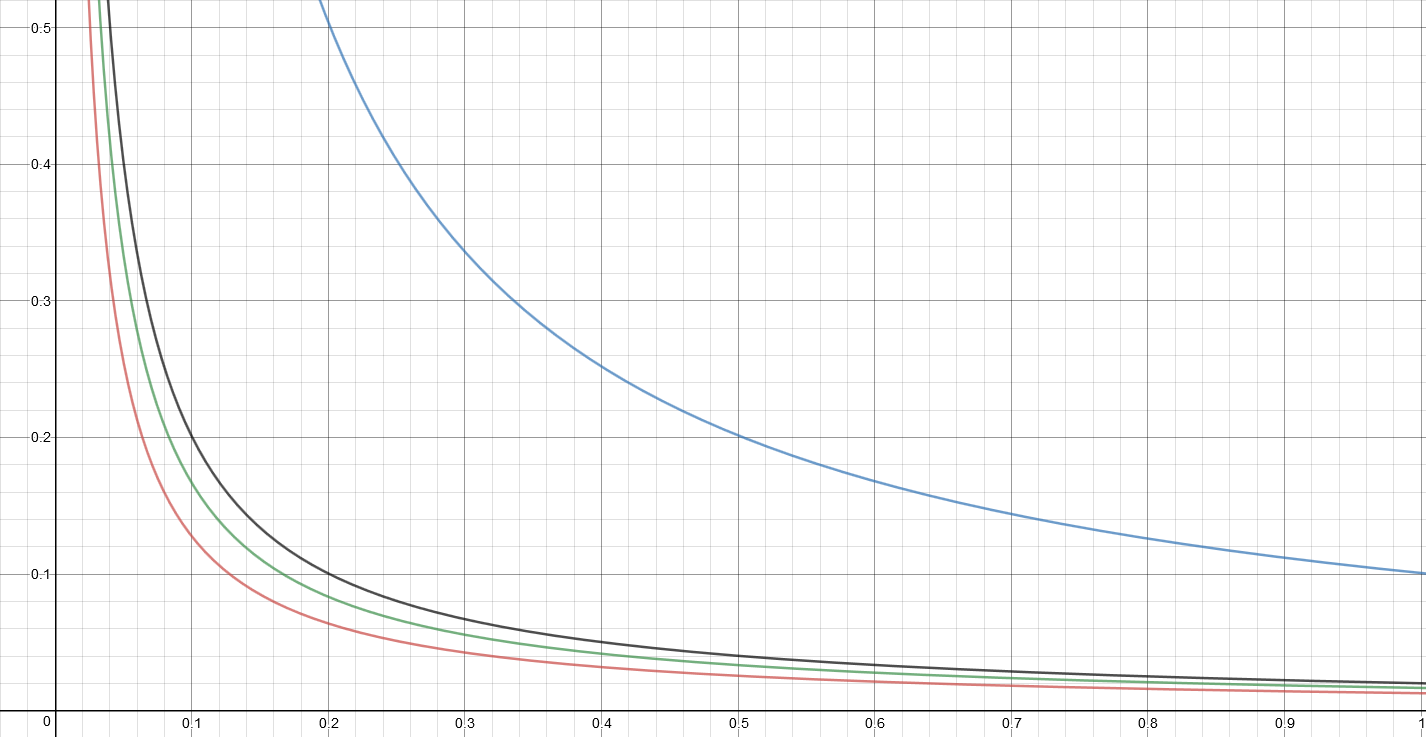
To represent all the information graphically, functions were plotted that display the relationship between arm length and weight for specific generated moments by the arms. Different values of moments for each arm are plotted, and compared to the stall torque of the serve motor. Graphically, as long as the curves of the moment stays below the stall torque curve, it would be possible to use the servo motor to power the arm.

Figure 25- Torque curves of different length (x-axis in m) and arm (y-axis in kg) combinations for all arms and for servo motor

Figure 25- Torque curves of different length (x-axis in m) and arm (y-axis in kg) combinations for all arms and for servo motor

**LLC 1142 motor**

**Rear arm**

**Middle arm**

**Front arm**

Figure 26 - Cube arm**LLC 1142 motor**

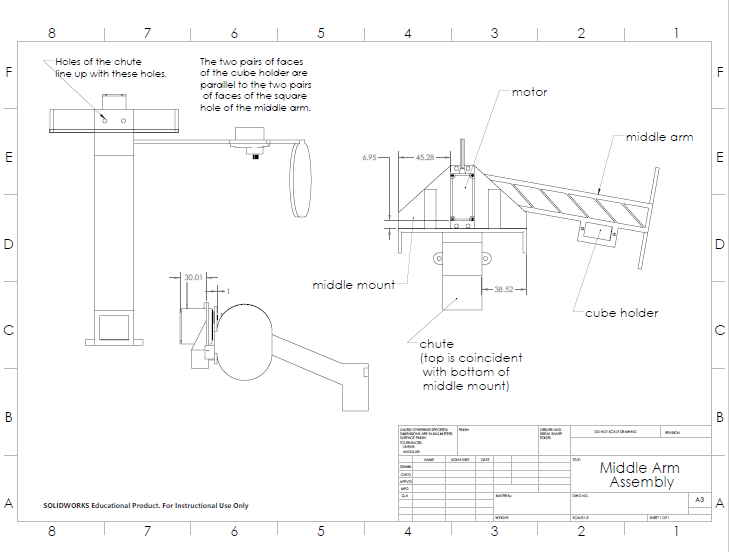
**Rear arm**

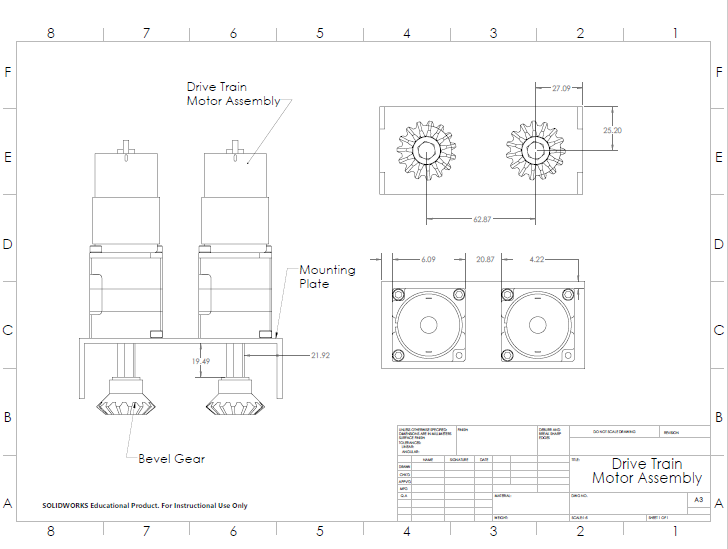
**Middle arm**

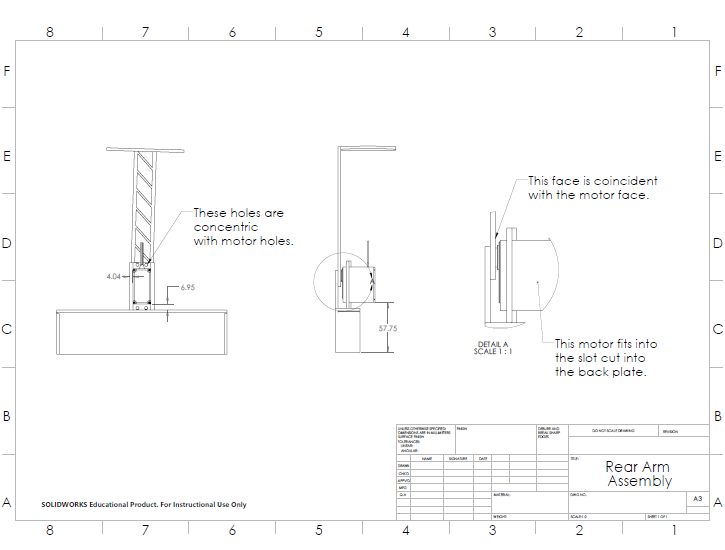
**Front arm**

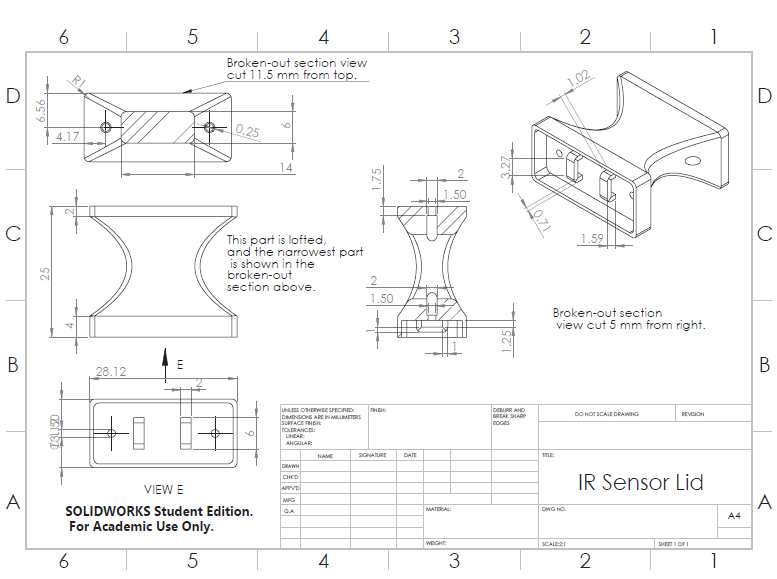
# Detailed Documentation

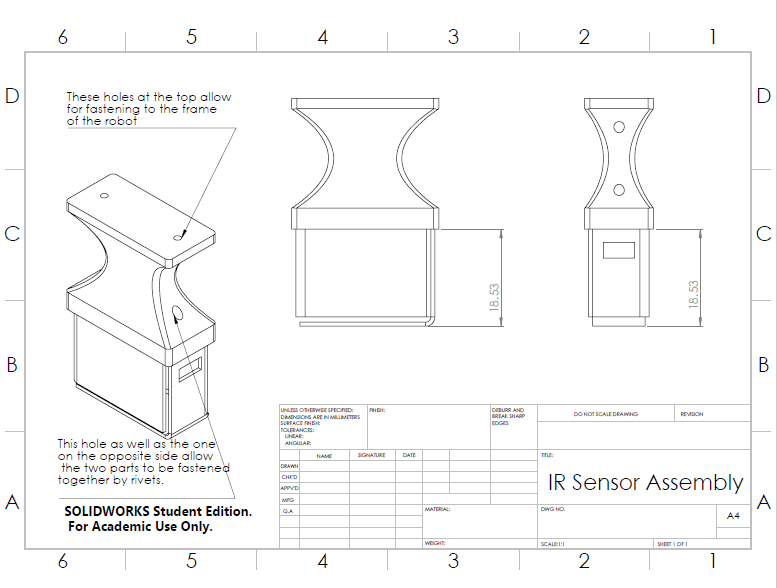
## Drawings

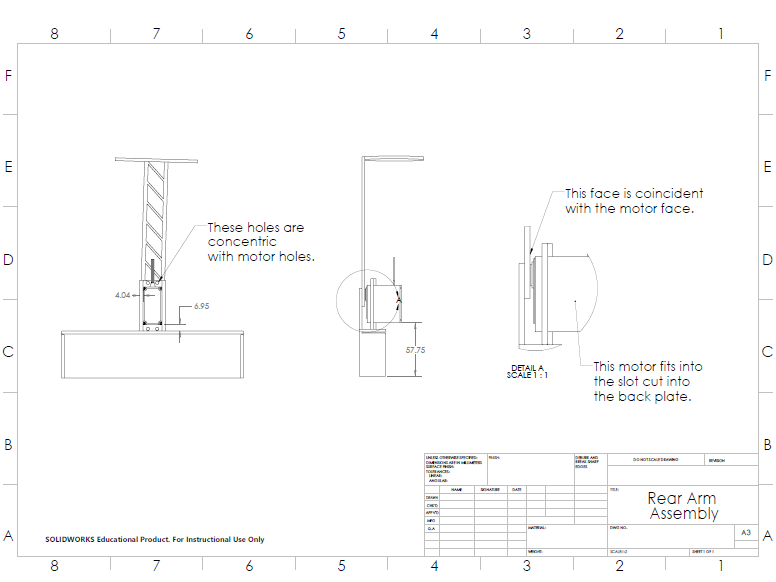


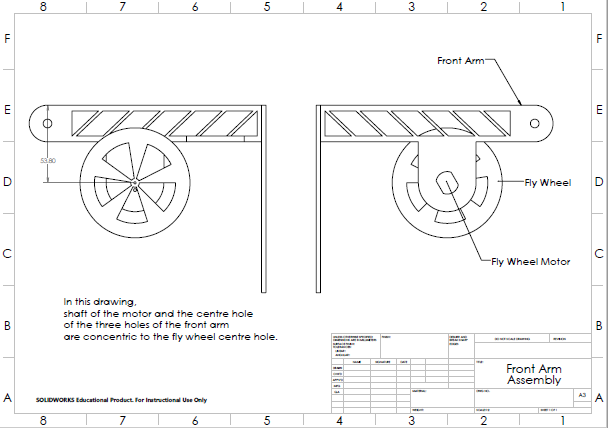


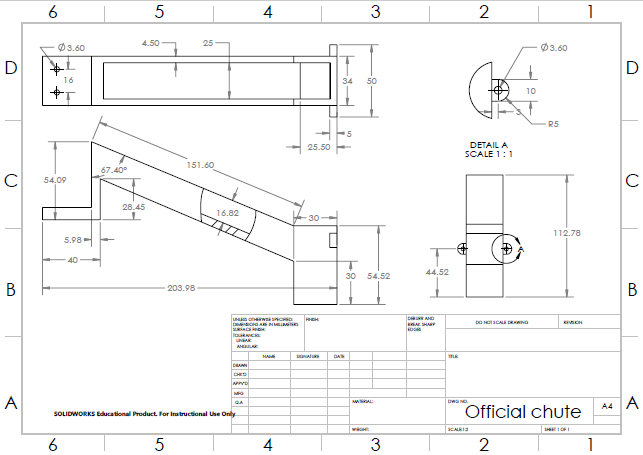












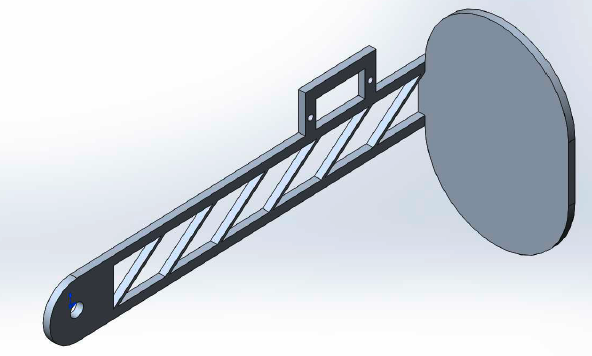


Figure 26 - Cube arm

Figure 27 - Front plateFigure 26 - Cube arm

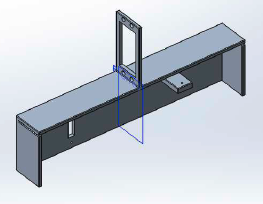


Figure 27 - Front plate

Figure 28- Fly wheelFigure 27 - Front plate

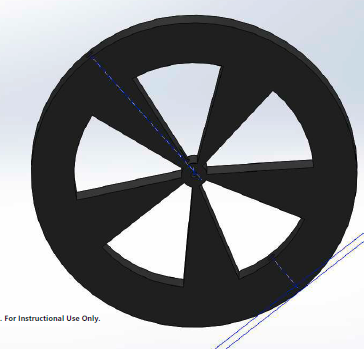


Figure 28- Fly wheel

Figure 29 - Back plateFigure 28- Fly wheel

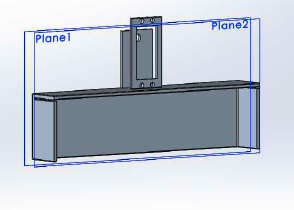


Figure 29 - Back plate

Figure 30 - Back armFigure 29 - Back plate

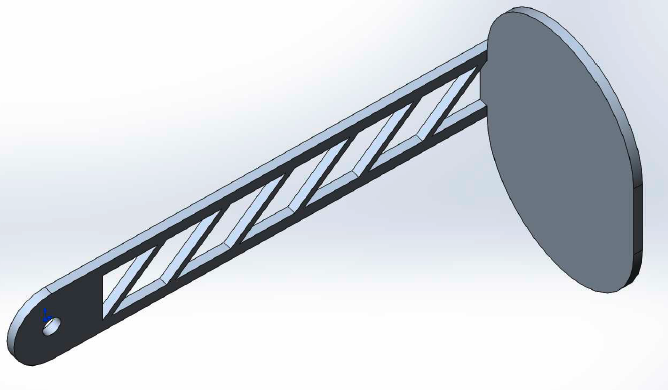


Figure 30 - Back arm

Figure 31 - Middle mountFigure 30 - Back arm

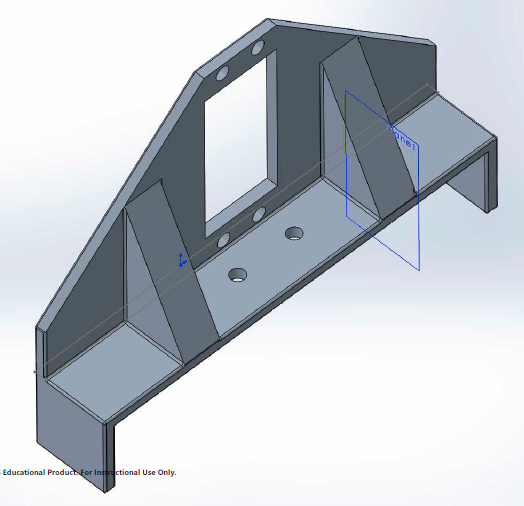


Figure 31 - Middle mount

Figure 32 - Pyramid armFigure 31 - Middle mount

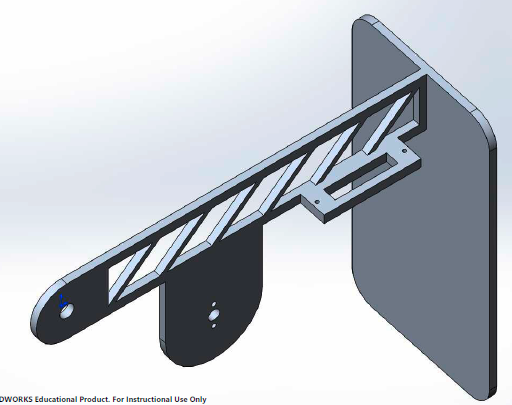


Figure 32 - Pyramid arm

Figure 33 - Planetary mountFigure 32 - Pyramid arm

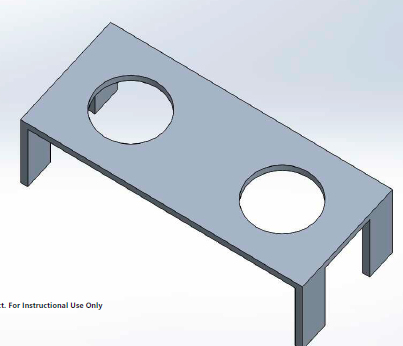


Figure 33 - Planetary mount

Figure 35 - Pyramid IR sensor assemblyFigure 33 - Planetary mount

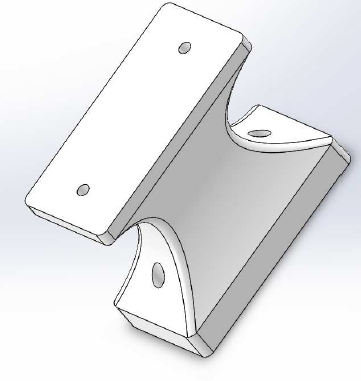
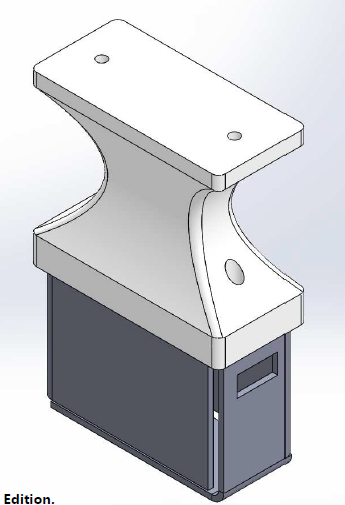


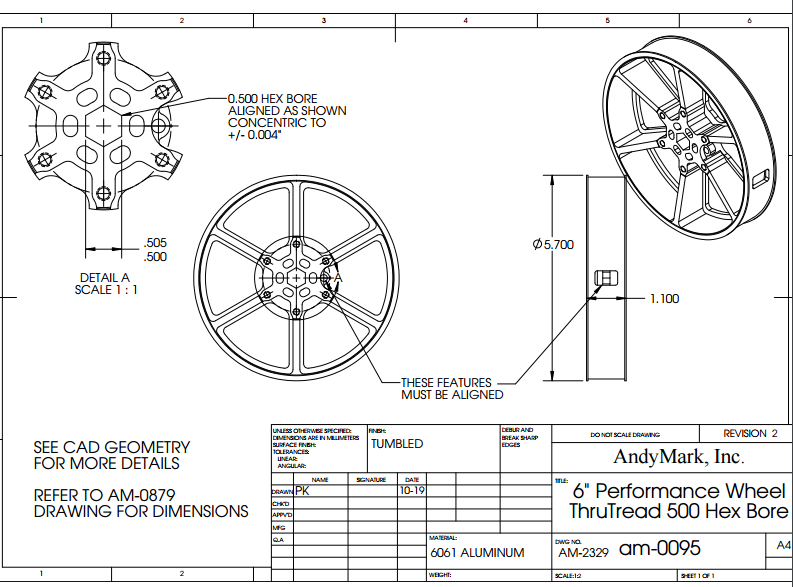
Figure 35 - Pyramid IR sensor assembly

Figure 34 - Pyramid IR sensor lidFigure 35 - Pyramid IR sensor assembly

Figure 34 - Pyramid IR sensor lid

*Figure 36- 6" performance wheels.*

From: http://files.andymark.com/am-0095%206%20Perf%20ThruTread%20500%20Hex.pdfFigure 34 - Pyramid IR sensor lid



*Figure 36- 6" performance wheels.*

*From: http://files.andymark.com/am-0095%206%20Perf%20ThruTread%20500%20Hex.pdf*

Figure 37- 40t Bevel gear

From:http://files.andymark.com/LayoutPrints/am-2620+40t+1-25mod+bevel+gear.pdf*Figure 36- 6" performance wheels.*

*From: http://files.andymark.com/am-0095%206%20Perf%20ThruTread%20500%20Hex.pdf*

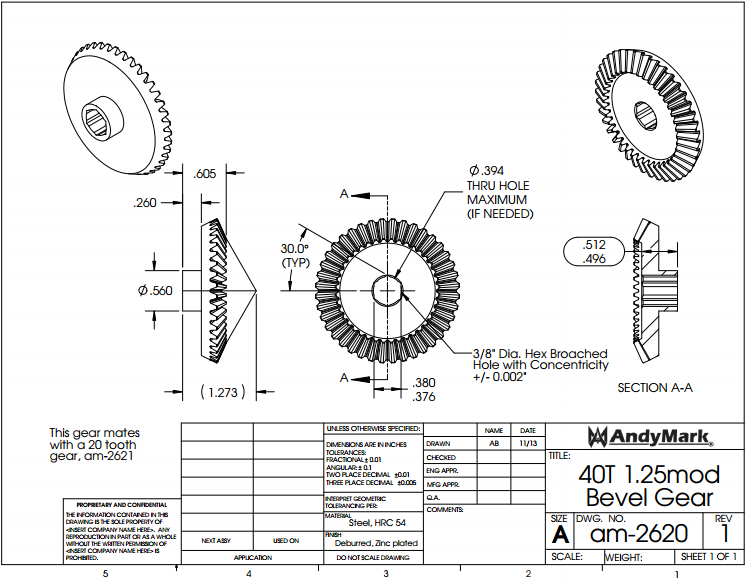


Figure 37- 40t Bevel gear

From:http://files.andymark.com/LayoutPrints/am-2620+40t+1-25mod+bevel+gear.pdf

Figure 38 - 14t gear.

From: http://files.andymark.com/LayoutPrints/am-0151+Toughbox+Small+Cluster+Gear.PDFFigure 37- 40t Bevel gear

From:http://files.andymark.com/LayoutPrints/am-2620+40t+1-25mod+bevel+gear.pdf

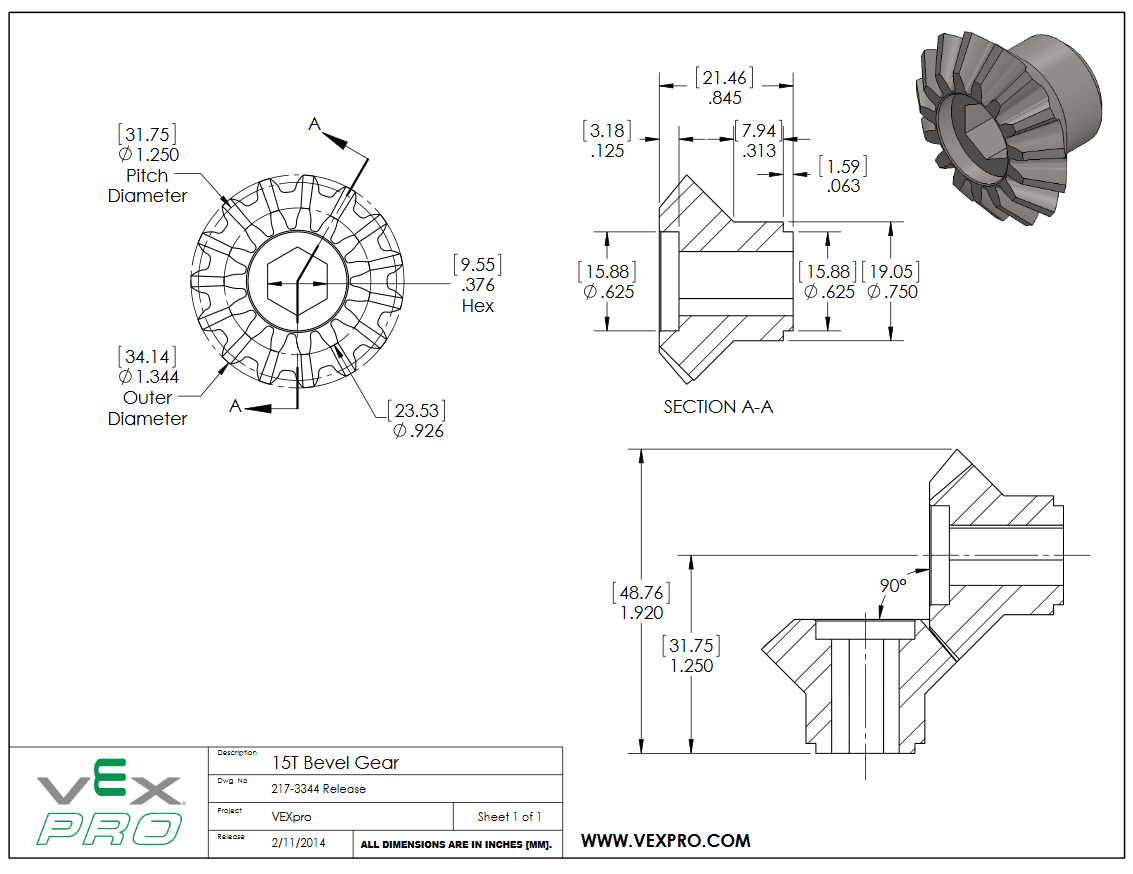


Figure 38 - 14t gear.

From: http://files.andymark.com/LayoutPrints/am-0151+Toughbox+Small+Cluster+Gear.PDF

Figure 39- Single stage Versaplanetary with RS550 input into 3/8” hex output.

From:https://content.vexrobotics.com/vexpro/pdf/VersaPlanetary-Layout-Drawing-20140115.PDFFigure 38 - 14t gear.

From: http://files.andymark.com/LayoutPrints/am-0151+Toughbox+Small+Cluster+Gear.PDF

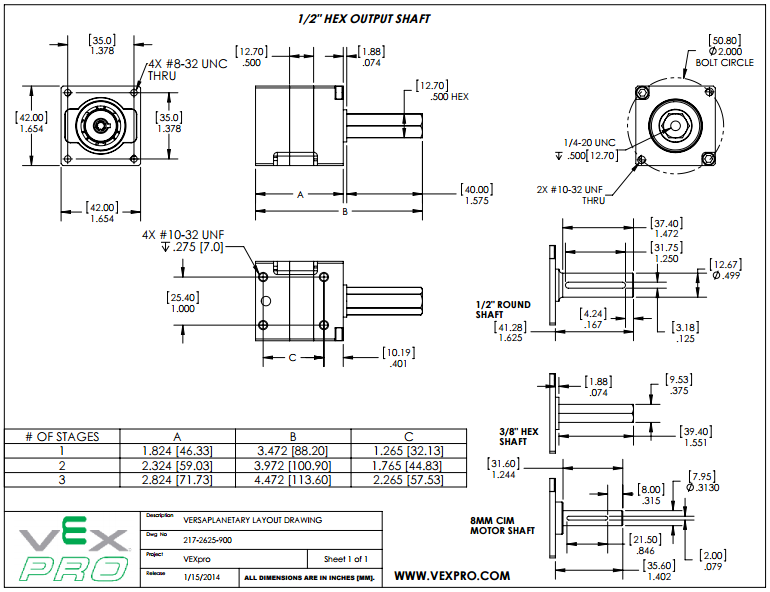


Figure 39- Single stage Versaplanetary with RS550 input into 3/8” hex output.

From:https://content.vexrobotics.com/vexpro/pdf/VersaPlanetary-Layout-Drawing-20140115.PDF

Figure 40- RS550 mounting plate.

From:https://content.vexrobotics.com/vexpro/pdf/217-3564-Drawing-20140212.PDFFigure 39- Single stage Versaplanetary with RS550 input into 3/8” hex output.

From:https://content.vexrobotics.com/vexpro/pdf/VersaPlanetary-Layout-Drawing-20140115.PDF

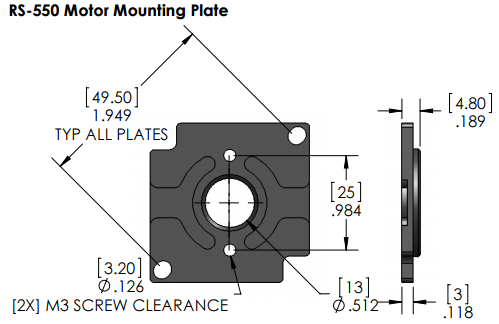


Figure 40- RS550 mounting plate.

From:https://content.vexrobotics.com/vexpro/pdf/217-3564-Drawing-20140212.PDF

Figure 41-PPN7PA12C brushed DC motor.

From: http://www.nmbtc.com/pdf/motors/PPN7.pdfFigure 40- RS550 mounting plate.

From:https://content.vexrobotics.com/vexpro/pdf/217-3564-Drawing-20140212.PDF

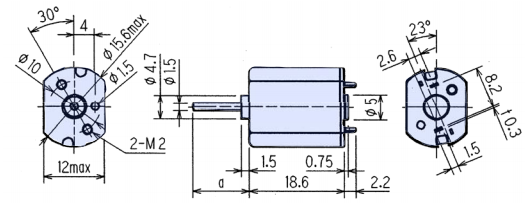


Figure 41-PPN7PA12C brushed DC motor.

From: http://www.nmbtc.com/pdf/motors/PPN7.pdf

Figure 42- Hall-effect sensor.

From: http://www.littelfuse.com/~/media/electronics/datasheets/hall\_effect\_sensors/littelfuse\_hall\_effect\_sensors\_55140\_datasheet.pdf.pdfFigure 41-PPN7PA12C brushed DC motor.

From: http://www.nmbtc.com/pdf/motors/PPN7.pdf

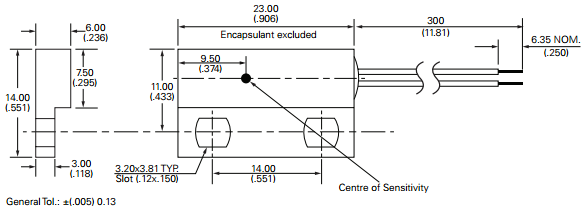
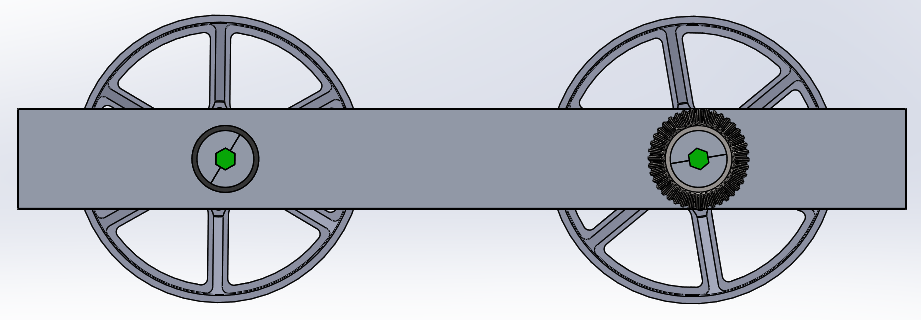
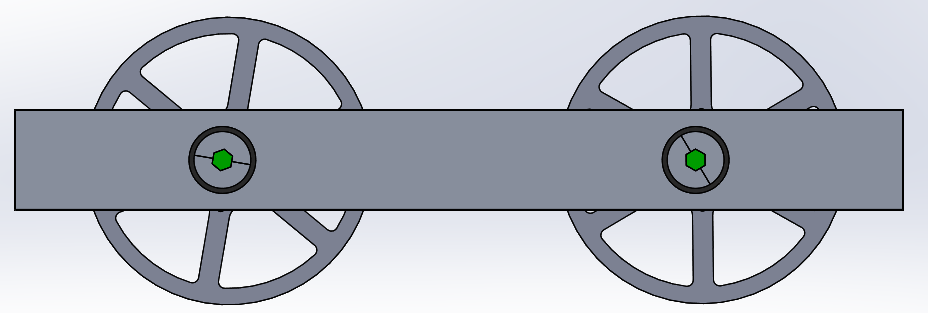


Figure 42- Hall-effect sensor.

From: http://www.littelfuse.com/~/media/electronics/datasheets/hall\_effect\_sensors/littelfuse\_hall\_effect\_sensors\_55140\_datasheet.pdf.pdf

Figure 43 - Pictures of wheel assemblyFigure 42- Hall-effect sensor.

From: http://www.littelfuse.com/~/media/electronics/datasheets/hall\_effect\_sensors/littelfuse\_hall\_effect\_sensors\_55140\_datasheet.pdf.pdf



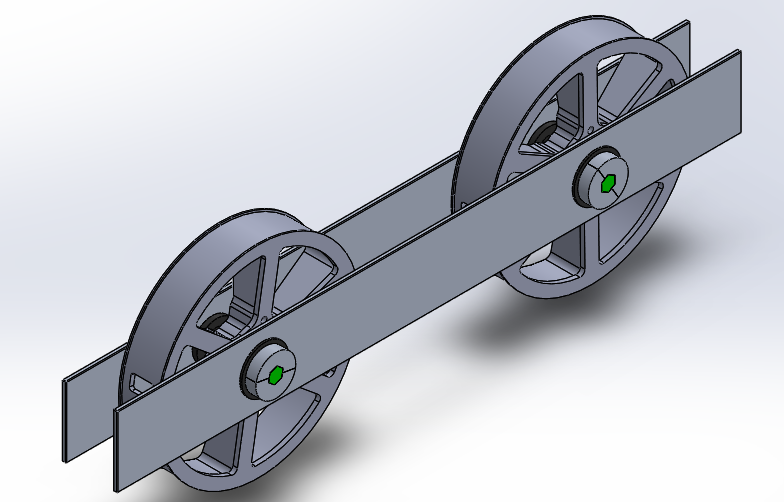


Figure 43 - Pictures of wheel assembly

Figure 44 – Pictures of pyramid arm assemblyFigure 43 - Pictures of wheel assembly

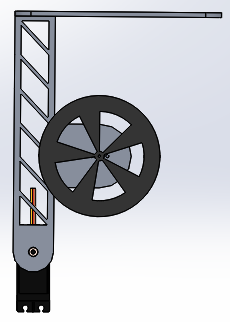
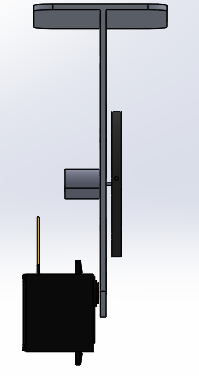


Figure 44 – Pictures of pyramid arm assembly

Figure 45 - Pictures of cube arm assemblyFigure 44 – Pictures of pyramid arm assembly

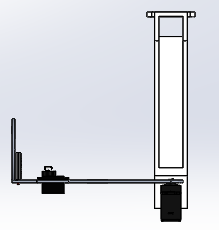
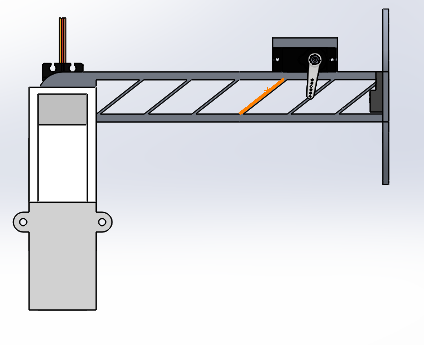
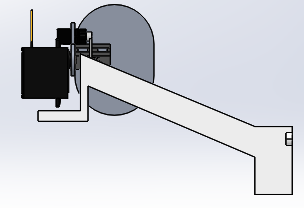
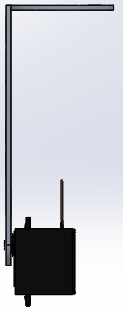
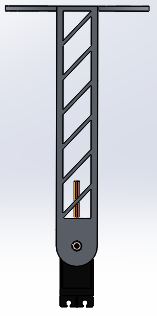


Figure 45 - Pictures of cube arm assembly

Figure 46 - Pictures of back arm assemblyFigure 45 - Pictures of cube arm assembly

Figure 46 - Pictures of back arm assembly

Figure 47 - Pictures of motor drive assemblyFigure 46 - Pictures of back arm assembly



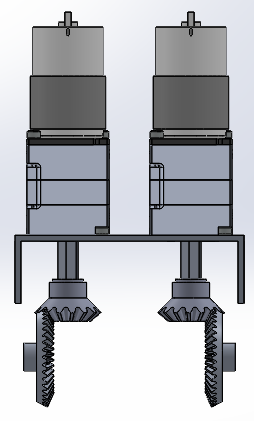
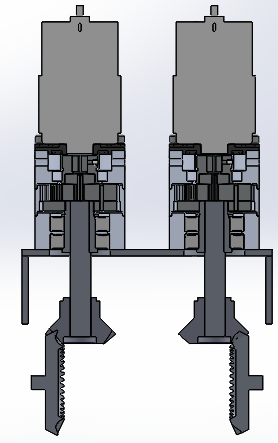
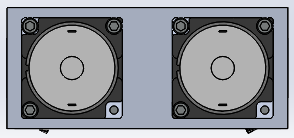


Figure 47 - Pictures of motor drive assembly

Figure 47 - Pictures of motor drive assembly

## Justification of Parts

### PPN7 DC Motor

Figure 48- PPN7 motor.

From: http://media.digikey.com/Photos/NMB%20Tech%20Photos/PPN7A12C1.jpg

Figure 48- PPN7 motor.

From: http://media.digikey.com/Photos/NMB%20Tech%20Photos/PPN7A12C1.jpg

This motor was chosen because of the high output speed it offered for its low weight. The front arm would be lifted by a single servo motor with no gearing, meaning that light parts are required to stay within the stall torque value of the motor. The motor weighs in at about 10g, which made it an excellent choice for powering a flywheel.

Figure 49- RS550 motor.

From: http://www.robotshop.com/media/catalog/product/cache/1/image/900x900/9df78eab33525d08d6e5fb8d27136e95/r/s/rs-550-motor-19300rpm-12v-6249oz-in.jpg

Figure 49- RS550 motor.

From: http://www.robotshop.com/media/catalog/product/cache/1/image/900x900/9df78eab33525d08d6e5fb8d27136e95/r/s/rs-550-motor-19300rpm-12v-6249oz-in.jpg

### RS550 Motor

The RS550 motor was chosen because it was a compact, cheap, low current motor for powering the drive train. The entire design of the robot had to fit within a small volume, which meant space was a necessity. Motor that could output the same torque such as CIM motors took up a lot more space, and would therefore have no place on the robot because it would make the job of fitting the frame parameters difficult.

Tower Pro Micro Servo

Figure 50- Tower pro micro servo.

From http://www.rcshopbd.com/wp-content/uploads/2015/04/2013-07-18-06083533883.jpg

Figure 50- Tower pro micro servo.

From http://www.rcshopbd.com/wp-content/uploads/2015/04/2013-07-18-06083533883.jpg

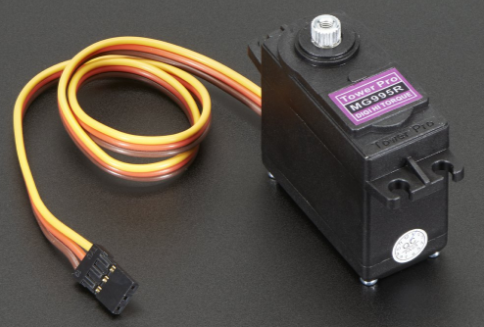
 The micro servo motor was used in the design because an extremely light servo motor was wanted to pick up the cube at the end of the middle arm of the robot. By minimizing the weight placed at the end of the arm, the torque required to lift the arm would also be reduced. The micro servo and armature was very light (19g) and cheap ($3.28), which made it the best option in controlling the armature to pick up the cube.

Figure 51- High torque servo motor.

From https://cdn-shop.adafruit.com/1200x900/1142-06.jpg

Figure 51- High torque servo motor.

From https://cdn-shop.adafruit.com/1200x900/1142-06.jpg

### Adafruit 1142 High Torque Servo

This servo motor was chosen because of the high output torque it offered. When designing the arms of the robot, there were different masses for each arm, and each would therefore require different servo motors to power them. To keep the parts consistent and easy to replace, it was decided to not follow the idea of choosing different motors, and instead pick one servo motor with enough strength to lift each of the arms. This would make the parts of the product easy to replace, and easy to use, since they all operate the same way.

### 40t and 15t Bevel Gears

Figure 52 - 15t and 40t bevel gears.

From: [https://www.andymark.com/product-p/am-2620.htm] and [http://www.wcproducts.net/217-3344]

Figure 52 - 15t and 40t bevel gears.

From: [https://www.andymark.com/product-p/am-2620.htm] and [http://www.wcproducts.net/217-3344]

These gears were used in the design because from the orientation of the motors, a method was required to change the direction of motion of the planetary gearbox output in the direction of the wheels. The bevel gears are a well-known method in changing the direction of motion of a mechanical system, and by varying the sizes of the driving and driven gears (15 and 40 teeth respectively), the gear ratio of the drive motors could be increased for more torque.

Figure 53- Versaplanetary gearbox with hex output.

From: https://www.vexrobotics.com/versaplanetary.html

Figure 53- Versaplanetary gearbox with hex output.

From: https://www.vexrobotics.com/versaplanetary.html

### VersaPlanetary Gearbox

For the gear box, the versa planetary was chosen because it was a compact way to increase the torque of the motor in a single stage. The planetary gearbox also ensures the torque created by the gearbox is entirely in the direction of the output shaft, which would give the robot an advantage in navigating bumps and uneven terrain. The design of the robot also allowed for the addition of more stages, where the user can increase the torque of the motor from the initial 1:10 to a maximum of 1:100 with the addition of a second stage.

### Cytron 13A Motor Controller

Figure 55- Different tread options for robot.

From: https://www.andymark.com/Search-s/545.htm?Search=tread&Submit=

Figure 55- Different tread options for robot.

From: https://www.andymark.com/Search-s/545.htm?Search=tread&Submit=

Figure 54- Cytron 13A motor controller.

From: http://www.robotshop.com/ca/en/13a-dc-motor-driver-grove-compatible.html

Figure 54- Cytron 13A motor controller.

From: http://www.robotshop.com/ca/en/13a-dc-motor-driver-grove-compatible.html

This component as chosen for the final design based on calculations made on the torque required to overcome static friction. When this was calculated for the robot, the torque constant of the motor for current was used to determine that the motor controller would be able to handle 13A of current to overcome static friction. The motor controller can handle a continuous current of 13A and spikes of 30A, so it would be well within operating range.



### Tread Options

The reason there are tread options for the robot was to help suit the needs of the user. Having different tread options allows the robot to perform better on different surfaces, and when coupled with the performance wheel, it is very easy to change the tread of the robot.

### Hall Effect Sensor

A Hall-effect sensor has the ability to detect a magnetic field if one is present in its near vicinity. According to electronics tutorials, this is because "magnetic sensors convert magnetic or magnetically encoded information into electrical signals for processing by electronic circuits (Electromagnetism). The Hall-Effect sensor actually outputs the density of the magnetic field it is by. It ultimately changes voltage (and therefore output values) in the presence of a field over a certain preset value in which it is coded to change the output value. Therefore, as the sensor gets closer to the source of the magnetic field, the values output display a detectable change. Of course, depending on the ultimate field strength of the source, these values change for the distance of the source.

In the application of the assignment, it is useful in detecting the presence of the tesseract in range of the robot. By attaching the Hall-effect sensor at the end of middle arm, it can detect the presence of the tesseract based on previously found values of the magnetic field that the tesseract emits. When the Hall-Effect Sensor outputs a value that signifies the cube is within reach of the robot, the tesseract (cube) holder is activated.

### The IR sensor

The IR sensor is only composed of an infrared light receiver, and is specific to a particular wavelength of light. When the sensor is facing the direction of the source of IR light (in this case the power pyramid), it decreases the positive integer value that it outputs. However, some types of material are opaque to IR light, and can therefore bounce it off into different directions. If the sensor is not protected, it can receive signals from directions other than the power pyramid itself, giving false values to the robot. Therefore, it must surrounded by a material opaque to IR light, and only allow the front of the sensor to be exposed to the outside world.

### IR Sensor Encasing

Four fairly common materials that are impenetrable to IR light are gold, silver, aluminum, and Plexiglass (Reference.com). Gold and silver will not be used for this assignment, as the price for the materials and procedures to have them applied to other materials in a foil manner are too expensive. There is nothing according to this reference that indicates any material is better than the other at reflecting IR light. When it comes to modelling small pieces, aluminum is the simpler material to use as it can be bent and easily welded into shape, whereas Plexiglass must be cut in separate segments, and then glued into place.

Aluminum can also be sold in sheets thick enough to protect the sensor held on the inside from becoming damaged, yet thin enough to allow for easy manipulation of smaller parts. The drawings for the aluminum casing display how the shape will be cut out of the sheet metal as a flat body. Dimensions are included in the presentation of the model in its folded, 3-D format. However, how the parts are folded to get into this shape is displayed on the flattened sheet metal diagram. After the sheet metal is folded, there will be gaps between the faces, and an open face. In order to prevent the IR receiver from getting signals from objects not facing it because they enter through these cracks, the shape will be welded together at the corners, with welding product that contains aluminum so it still has the ability to be opaque to IR light at those edges. The top is open for access to the possibility of having to replace the part. At each of the opposite short faces, a tab will be added to the end to allow for rivet holes to be put in.

### IR Sensor Lid

The sensor lid, based on the geometric properties and small, fairly complex shapes, must be injection molded. To do this, a fairly cheap yet reliable plastic will be used: melamine. This plastic is commonly used in children's toys. It is a durable, thermosetting plastic, meaning it will not bend or alter its shape unless it is heated to a fairly high temperature. Looking at the model for the IR Sensor Lid in the pdf, one can see that the middle of the piece is lofted. When the piece is put through the injection mold in this way, it will cool fairly evenly, as well as use less material overall. Both the top and bottom of the lid are flat to allow for it to be fastened easily to the IR Sensor encasing, and the chassis of the robot. The bottom of the lid is the shelled part with two holes drilled through it. There are two different sized holes so the rivets can sit on the lid and squeeze the lid and the encasing together in the assembly. The holes at the top allow it to be fastened to the chassis of the robot with screws, and nuts on the opposite side of the chassis, holding the assembly in place.

### IR Sensor Assembly

The IR sensor lid will go over top of the open face of the IR Sensor encasing, and its holes on the bottom face are concentric to the holes on the two tabs of the IR Sensor encasing. The holes of the IR Sensor remain on the inside, the spherical face lining up with the hole, the prongs passing through the two swept shapes on the bottom of the lid. Since the lid is made of plastic, no metal touches the IR Sensor. Attached to the prongs is a cable, which passes through the square hole of the IR Sensor encasing.

### IR Distance Sensor

An IR Distance Sensor is similar to a simple IR Sensor in that it has an IR receiver. However, it also includes an IR emitter, which emits infrared light at a particular frequency that matches the frequency that the receiver can read. This wavelength (unlike the wavelength emitted by the power pyramid) is opaque to most objects, and can therefore be bounced off an object and return to the IR distance sensor. In this way, the sensor is capable of outputting a value that corresponds to the distance from the sensor itself to the closest object in front of it. The sensor is required for the design in order to be able to detect where the front of the robot is relative to the nearest wall in front of it. It is also used in order to detect a change in height that correlates to the pyramid, and therefore allow the robot to autonomously locate the location of the pyramid.

### Cube Holder

The group decided that the best approach to pick up the cube was with a small servo motor. A servo motor can be programmed to turn to specific angles. This micro servo motor also has the ability to apply enough force to the tesseract to hold it tight against another surface, without actually damaging it. Weighing less than 10 g itself, it adds very little weight to the middle arm of the robot, therefore not really affecting the torque that the motor for the middle arm needs to exert.

### Chute

This component contains some fairly small dimensions, and should therefore be injection molded. Using melamine resin, the part will remain light, yet hard, and will be able to be injection molded into the tight crevices that the part requires. When the cube is dropped to the ground, it needs a delivery system. By using a chute, the cube can be delivered to the ground to a specific and consistent location every time, allowing for predictability. This allows for overall less code to be written, and easier maneuvering for the robot to perform. The chute itself has small components that extend in either direction in order to allow for fastening to the chassis of the robot.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Bill of Materials Table 6- Bill of materials  Table 6- Bill of materials | **Quantity** | **Unit Mass (g)** | **Total Mass** | **Unit Cost** | **Total Cost (CAD)** | **Supplier** | **Part(s) Number** |
|  |  |  |  |  |  |  |  |
| ***MOTORS*** |  |  |  |  |  |  |  |
| PPN7 DC motor | 1 | 9.98 | 9.98 | 4.93 | 4.93 | DigiKey | P14355-ND |
| RS 550 Motor | 2 | 218g | 436 | 6.65 | 13.3 | RobotShop | RB-Wtc-03 |
| Tower Pro Micro Servo | 1 | 19 | 38 | 3.38 | 3.38 | Gearbest | 1528-1076-ND |
| Adafruit 1142 high torque servo motor | 3 | 55g | 165 | 28.69 | 86.07 | DigiKey | 1528-1083-ND |
|  |  |  |  |  |  |  |  |
| ***GEARS*** |  |  |  |  |  |  |  |
| 3/8 bore 40t bevel gear | 2 | 86.2 | 172.4 | 42 | 84 | AndyMark | am-2620 |
| 15T 3/8" bore bevel gear | 2 | 53.5 | 107 | 14.99 | 29.98 | WestCoast Products | 217-3344 |
| Single stage Versaplanetary with 0.375" hex output | 2 | 254 | 508 | 65.98 | 131.96 | VexPro | (217-4947) & (217-2820) |
|  |  |  |  |  |  |  |  |
| ***HARDWARE*** |  |  |  |  |  |  |  |
| 3/8 inch hex shaft | 1 | 68 | 68 | 5 | 5 | Andymark | am-2290a |
| 3/8 inch ball bearing (flanged) | 8 | 25.4 | 203 | 5 | 40 | AndyMark | am-0692 |
| 3/8 inch hex shaft collar | 8 | 9 | 72 | 5 | 40 | AndyMark | am-2300 |
| 3/8 inch width hex spacer (10 pack) | 1 | 45.4 | 45.4 | 6.99 | 6.99 | VexPro | 217-3262 |
| 6" performance wheels | 4 | 272.2 | 1088.8 | 47 | 188 | AndyMark | am-0879 |
|  |  |  |  |  |  |  |  |
| ***SENSOR AND ELECTRONICS*** |  |  |  |  |  |  |  |
| Hall effect Sensor | 1 | 8.5 | 8.5 | 14.84 | 14.84 | DigiKey | 55140-3H-02-A-ND |
| IR Sensor | 1 | 1 | 1 | 1.91 | 1.91 | DigiKey | 751-1384-5-ND |
| IR Distance Sensor | 2 | 2.5 | 5 | 16.39 | 32.78 | DigiKey | 425-2855-ND |
| Cytron 13A Motor Controller | 3 | 40 | 120 | 13.99 | 41.97 | RobotShop | RB-Cyt-212 |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| ***TREAD OPTIONS*** |  |  |  |  |  |  |  |
| Polyester Tread | 1 | 133.9 | 133.9 | 47 | 47 | AndyMark | am-2611 |
| Carbox Nitrile Rubber Tread | 1 | 313.5 | 313.5 | 35 | 35 | AndyMark | am-3309 |
| SBR Rubber Tread | 1 | 313.5 | 313.5 | 32 | 32 | AndyMark | am-3310 |
| Natural Rubber Tread | 1 | 399 | 399 | 24 | 24 | AndyMark | am-0522 |
|  |  |  |  |  |  |  |  |
| ***FABRICATED PARTS*** |  |  |  |  |  |  |  |
| Back arm | 1 | 55.5 | 55.5 | 0.15 | 0.15 | MetalsDepot | S318T6 |
| Middle arm | 1 | 58 | 58 | 0.16 | 0.16 | MetalsDepot | S318T6 |
| Front arm | 1 | 107.31 | 107.31 | 0.29 | 0.29 | MetalsDepot | S318T6 |
| Rear Plate | 1 | 155.7 | 155.7 | 0.42 | 0.42 | MetalsDepot | S318T6 |
| Front Plate | 1 | 149.58 | 149.58 | 0.4 | 0.4 | MetalsDepot | S318T6 |
| Middle Mount | 1 | 66.34 | 66.34 | 0.18 | 0.18 | MetalsDepot | S318T6 |
| Planetary Mount | 1 | 50.46 | 50.46 | 0.13 | 0.13 | MetalsDepot | S318T6 |
| Chute | 1 | 136.9 | 136.9 | 0.16 | 0.16 | ChemiPlastica | Melochem |
| Fly wheel | 1 | 18.87 | 18.87 | 0.35 | 0.35 | ResinsOnline | SKA06080 |
| IR Sensor Lid | 1 | 6.35 | 6.35 | 0.01 | 0.01 | ChemiPlastica | Melochem |
| IR Sensor Encasing | 1 | 1.23 | 1.23 | 0.003 | 0.003 | MetalsDepot | S318T6 |
| ***TOTAL (Upper cost)*** |  |  | **4132.09** |  | **774.363** |  |  |
| ***TOTAL (Lower cost)*** |  |  | **3988.22** |  | **751.363** |  |  |

## Robot Code

For the code, the provided code for the line following robot was taken the irrelevant sections were removed. The ability to calibrate the motor was desired, use the encoders and have a mode zero where the robot waits to begin the course. This was effective because it resulted in a controlled start to the course and the robot being able to drive straight.

### ***Pinouts***

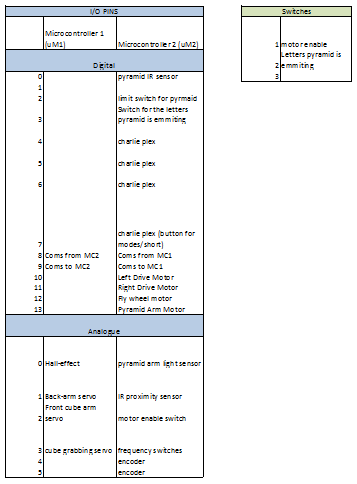


Figure 56-Pinout diagram from robot

Figure 56-Pinout diagram from robot

### Master Code

//libraries

#include <Servo.h>

#include <EEPROM.h>

#include <uSTimer2.h>

#include <CharliePlexM.h>

#include <Wire.h>

#include <I2CEncoder.h>

#include <SoftwareSerial.h>

//function to set motor speeds

void setMotorSpeeds(int left, int right);

void getFacingPyramid();

//serial communication between boards (read,write)

SoftwareSerial boardComs(8,9);

//motors

Servo servo\_RightMotor;

Servo servo\_LeftMotor;

Servo servo\_PyramidArmMotor;

Servo servo\_FlyWheelMotor;

I2CEncoder encoder\_RightMotor;

I2CEncoder encoder\_LeftMotor;

boolean bt\_Motors\_Enabled = true;

//port pin constants

//digital (pins 1 and 0 are for communication to the other Microcontroller)

const int ci\_button = 2;

const int ci\_Letters\_Switch = 3;

const int ci\_Charlieplex\_LED1 = 4;

const int ci\_Charlieplex\_LED2 = 5;

const int ci\_Charlieplex\_LED3 = 6;

const int ci\_Charlieplex\_LED4 = 7;

const int ci\_Mode\_Button = 7;

const int ci\_Left\_Motor = 10;

const int ci\_Right\_Motor = 11;

const int ci\_Fly\_Wheel\_Motor = 12;

const int ci\_Pyramid\_Arm\_Motor = 13;

//analog

const int ci\_Light\_Sensor = A0;

const int ci\_IR\_Proximity\_Sensor = A1;

const int ci\_Motor\_Enable\_Switch = A2;

const int ci\_I2C\_SDA = A4; // I2C data = white

const int ci\_I2C\_SCL = A5; // I2C clock = yellow

// Charlieplexing LED assignments

const int ci\_Heartbeat\_LED = 1;

const int ci\_Indicator\_LED = 4;

//constants

// EEPROM addresses

const int ci\_Left\_Motor\_Offset\_Address\_L = 12;

const int ci\_Left\_Motor\_Offset\_Address\_H = 13;

const int ci\_Right\_Motor\_Offset\_Address\_L = 14;

const int ci\_Right\_Motor\_Offset\_Address\_H = 15;

const int ci\_Left\_Motor\_Stop = 1500; // 200 for brake mode; 1500 for stop

const int ci\_Right\_Motor\_Stop = 1500;

const int ci\_Display\_Time = 500;

// May need to adjust this

const int ci\_Motor\_Calibration\_Cycles = 3;

const int ci\_Motor\_Calibration\_Time = 5000;

//variables

byte b\_LowByte;

byte b\_HighByte;

unsigned int ui\_Motors\_Speed = 1900; // Default run speed

unsigned int ui\_Left\_Motor\_Speed;

unsigned int ui\_Right\_Motor\_Speed;

long l\_Left\_Motor\_Position;

long l\_Right\_Motor\_Position;

unsigned long ul\_3\_Second\_timer = 0;

unsigned long ul\_Display\_Time;

unsigned long ul\_Calibration\_Time;

unsigned long ui\_Left\_Motor\_Offset;

unsigned long ui\_Right\_Motor\_Offset;

unsigned int ui\_Cal\_Count;

unsigned int ui\_Cal\_Cycle;

unsigned int ui\_Robot\_State\_Index = 0;

//0123456789ABCDEF

unsigned int ui\_Mode\_Indicator[6] = {

0x00, //B0000000000000000, //Stop

0x00FF, //B0000000011111111, //Run

0x0F0F, //B0000111100001111, //Calibrate line tracker light level

0x3333, //B0011001100110011, //Calibrate line tracker dark level

0xAAAA, //B1010101010101010, //Calibrate motors

0xFFFF //B1111111111111111 //Unused

};

unsigned int ui\_Mode\_Indicator\_Index = 0;

//display Bits 0,1,2,3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

int iArray[16] = {1,2,4,8,16,32,64,128,256,512,1024,2048,4096,8192,16384,65536 };

int iArrayIndex = 0;

boolean bt\_Heartbeat = true;

boolean bt\_3\_S\_Time\_Up = false;

boolean bt\_Do\_Once = false;

boolean bt\_Cal\_Initialized = false;

//GENERAL VARIABLES

int courseStageMaster = 0;

int courseStageSlave = 0;

int slowForward = 1650;

int slowReverse = 1300;

int fullForward = 2100;

int fullReverse = 900;

int stopMotors = 1500;

int IRProxValue;

int IRProxWhenWall20cm = 0;

int IRProxWhenWall10cm = 500;

int IRProxWhenPyramidInPlace = 0;

int lightSensorValue;

int lightSensorWithoutPyramid = 0;

//PICK UP CUBE VARIABLES

bool cubeObtained = false;

long int leftMotorInitialPosition;

long int leftMotorCurrentPosition;

long int encoderCountFor2ft = 100;

//LOCATE PYRAMID VARIABLES

bool letters; //true when looking for A&E, false for I&O

int facingPyramid = 0; //1 when facing pyramid, 2 when its not

int pyramidValue;

bool haveFacedPyramid = false;

bool leftTurnDone = false;

int buttonValue;

long int newcount;

long int oldcount;

// the setup function runs once when you press reset or power the board

void setup() {

Wire.begin(); // Wire library required for I2CEncoder library

Serial.begin(9600);

boardComs.begin(19200);

CharliePlexM::setBtn(ci\_Charlieplex\_LED1, ci\_Charlieplex\_LED2,ci\_Charlieplex\_LED3, ci\_Charlieplex\_LED4, ci\_Mode\_Button);

// set up drive motors

pinMode(ci\_Right\_Motor, OUTPUT);

servo\_RightMotor.attach(ci\_Right\_Motor);

pinMode(ci\_Left\_Motor, OUTPUT);

servo\_LeftMotor.attach(ci\_Left\_Motor);

// set up arm motors

pinMode(ci\_Fly\_Wheel\_Motor, OUTPUT);

servo\_FlyWheelMotor.attach(ci\_Fly\_Wheel\_Motor);

pinMode(ci\_Pyramid\_Arm\_Motor, OUTPUT);

servo\_PyramidArmMotor.attach(ci\_Pyramid\_Arm\_Motor);

//set up switches

pinMode(ci\_Motor\_Enable\_Switch, INPUT);

pinMode(ci\_Letters\_Switch,INPUT);

pinMode(ci\_button,INPUT); //limit switch

// set up encoders. Must be initialized in order that they are chained together,

// starting with the encoder directly connected to the Arduino. See I2CEncoder docs

// for more information

encoder\_LeftMotor.init(1.0 / 3.0\*MOTOR\_393\_SPEED\_ROTATIONS, MOTOR\_393\_TIME\_DELTA);

encoder\_LeftMotor.setReversed(false); // adjust for positive count when moving forward

encoder\_RightMotor.init(1.0 / 3.0\*MOTOR\_393\_SPEED\_ROTATIONS, MOTOR\_393\_TIME\_DELTA);

encoder\_RightMotor.setReversed(true); // adjust for positive count when moving forward

// read saved values from EEPROM

b\_LowByte = EEPROM.read(ci\_Left\_Motor\_Offset\_Address\_L);

b\_HighByte = EEPROM.read(ci\_Left\_Motor\_Offset\_Address\_H);

ui\_Left\_Motor\_Offset = word(b\_HighByte, b\_LowByte);

b\_LowByte = EEPROM.read(ci\_Right\_Motor\_Offset\_Address\_L);

b\_HighByte = EEPROM.read(ci\_Right\_Motor\_Offset\_Address\_H);

ui\_Right\_Motor\_Offset = word(b\_HighByte, b\_LowByte);

//set up sensors

pinMode(ci\_Light\_Sensor, INPUT);

pinMode(ci\_IR\_Proximity\_Sensor,INPUT);

}

// the loop function runs over and over again until power down or reset

void loop() {

if ((millis() - ul\_3\_Second\_timer) > 3000)

{

bt\_3\_S\_Time\_Up = true;

}

// button-based mode selection

if (CharliePlexM::ui\_Btn)

{

if (bt\_Do\_Once == false)

{

bt\_Do\_Once = true;

ui\_Robot\_State\_Index++;

ui\_Robot\_State\_Index = ui\_Robot\_State\_Index & 7;

ul\_3\_Second\_timer = millis();

bt\_3\_S\_Time\_Up = false;

bt\_Cal\_Initialized = false;

ui\_Cal\_Cycle = 0;

}

}

else

{

bt\_Do\_Once = LOW;

}

// check if drive motors should be powered

bt\_Motors\_Enabled = digitalRead(ci\_Motor\_Enable\_Switch);

// modes

// 0 = default after power up/reset

// 1 = Press mode button once to enter. Run robot.

// 2 = Press mode button four times to enter. Calibrate motor speeds to drive straight.

switch (ui\_Robot\_State\_Index)

{

case 0: //Robot stopped

{

servo\_LeftMotor.writeMicroseconds(stopMotors);

servo\_RightMotor.writeMicroseconds(stopMotors);

encoder\_LeftMotor.zero();

encoder\_RightMotor.zero();

ui\_Mode\_Indicator\_Index = 0;

courseStageMaster = 0;

courseStageSlave = 0;

//check which set of letters we're looking for

letters = digitalRead(ci\_Letters\_Switch);

break;

}

case 1: //Robot Run after 3 seconds

{

if (bt\_3\_S\_Time\_Up)

{

// set motor speeds

ui\_Left\_Motor\_Speed = constrain(ui\_Motors\_Speed + ui\_Left\_Motor\_Offset, 1600, 2100);

ui\_Right\_Motor\_Speed = constrain(ui\_Motors\_Speed + ui\_Right\_Motor\_Offset, 1600, 2100);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* START OF COURSE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* START OF COURSE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* START OF COURSE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Serial.println(courseStageMaster);

switch (courseStageMaster)

{

//STAGE 0-7 - LOCATING AND OBTAINING CUBE

case 0: //wait to start, do anything that needs to be done

{

CharliePlexM::Write(12,HIGH);

servo\_FlyWheelMotor.detach();

leftMotorInitialPosition = encoder\_LeftMotor.getRawPosition();

//Serial.println("initial position: ");

//Serial.println(leftMotorInitialPosition);

//set up pyramid arm to approx 60', starting on the ground

//servo\_PyramidArmMotor.attach(ci\_Pyramid\_Arm\_Motor);

servo\_PyramidArmMotor.write(1200);

delay(925);

servo\_PyramidArmMotor.write(stopMotors);

delay(2000);

courseStageMaster=1;

CharliePlexM::Write(12,LOW);

break;

}

case 1: //wait, do not drive forward, while MC1 lowers arms

{

CharliePlexM::Write(9,HIGH);

setMotorSpeeds(stopMotors,stopMotors);

servo\_PyramidArmMotor.write(1430);

CharliePlexM::Write(9,LOW);

break;

}

case 2: //drive forward slowly,

{

CharliePlexM::Write(6,HIGH);

servo\_PyramidArmMotor.write(1440);

//delay(500);

setMotorSpeeds(slowForward,slowForward);

//check encoder counts to know when we've driven forward about a robot length

leftMotorCurrentPosition = encoder\_LeftMotor.getRawPosition();

if((leftMotorCurrentPosition - leftMotorInitialPosition)<-3000)

{courseStageMaster=3;}

CharliePlexM::Write(6,LOW);

break;

}

case 3: //wait while MC1 raises cube arm

{

CharliePlexM::Write(3,HIGH);

setMotorSpeeds(stopMotors,stopMotors);

servo\_PyramidArmMotor.write(1440);

//reset encoder position

leftMotorInitialPosition = encoder\_LeftMotor.getRawPosition();

CharliePlexM::Write(3,LOW);

break;

}

case 4://reverse two feet

{

delay(200);

CharliePlexM::Write(11,HIGH);

servo\_PyramidArmMotor.write(1440);

setMotorSpeeds(slowReverse,slowReverse);

leftMotorCurrentPosition = encoder\_LeftMotor.getRawPosition();

if((leftMotorCurrentPosition - leftMotorInitialPosition)>2900)

{courseStageMaster=5;}

CharliePlexM::Write(11,LOW);

break;

}

case 5://wait while MC1 lowers cube arm

{

CharliePlexM::Write(8,HIGH);

servo\_PyramidArmMotor.write(1440);

setMotorSpeeds(stopMotors,stopMotors);

CharliePlexM::Write(8,LOW);

break;

}

case 6: //drive forward slowly, increase stage if proximity sensor is above/below XXX, ie we hit a wall

{

CharliePlexM::Write(5,HIGH);

servo\_PyramidArmMotor.write(1440);

setMotorSpeeds(slowForward,slowForward);

//make sure don't crash into wall.

IRProxValue = analogRead(ci\_IR\_Proximity\_Sensor);

// Serial.println(IRProxValue);

if(IRProxValue > 520)

{courseStageMaster=7;}

CharliePlexM::Write(2,LOW);

break;

}

case 7: //wait while SlaveMC gets cube

{

CharliePlexM::Write(10,HIGH);

servo\_PyramidArmMotor.write(1440);

setMotorSpeeds(stopMotors,stopMotors);

CharliePlexM::Write(10,LOW);

break;

}

case 8: //set up for finding pyramid by getting away from the wall

{

//setMotorSpeeds(slowReverse,slowReverse);

servo\_PyramidArmMotor.write(1440);

//reverse

setMotorSpeeds(1200,1200);

delay(3000);

setMotorSpeeds(stopMotors,stopMotors);

//get away from wall

for(int i = 0; i<5; i++)

{

//back up

setMotorSpeeds(fullReverse,fullReverse);

delay(300);

setMotorSpeeds(stopMotors,stopMotors);

//turn left

setMotorSpeeds(fullReverse,fullForward);

delay(900);

setMotorSpeeds(stopMotors,stopMotors);

}

oldcount = millis();

courseStageMaster = 9;

break;

}

//STAGE 9 - LOCATE PYRAMID

case 9:

{

CharliePlexM::Write(12,HIGH);

CharliePlexM::Write(9,HIGH);

servo\_PyramidArmMotor.write(1400);

getFacingPyramid();

switch(facingPyramid)

{

case 2: //not facing pyramid

{

if(haveFacedPyramid && !leftTurnDone) //turn left set amount

{

setMotorSpeeds(1000,2000);

delay(500);

setMotorSpeeds(stopMotors,stopMotors);

leftTurnDone = true;

}

else //if pyramid has not been found, or it has been found and left turn is complete

{

//turn right

setMotorSpeeds(1950,1050);

delay(110);

setMotorSpeeds(1600,1600);

delay(100);

setMotorSpeeds(stopMotors,stopMotors);

}

break;

}

case 1: //currently in line with the pyramid, drive straight

//if facing the pyramid, slowly drive forward while pinging the light sensor and the IRProx

{

haveFacedPyramid = true;

//drive forward

setMotorSpeeds(slowForward,slowForward);

//ping light sensor and IRProx

IRProxValue = analogRead(ci\_IR\_Proximity\_Sensor);

//lightSensorValue = analogRead(ci\_Light\_Sensor);

buttonValue = digitalRead(ci\_button);

if(buttonValue==1 && IRProxValue < IRProxWhenWall10cm) //in front of pyramid

{

setMotorSpeeds(stopMotors,stopMotors);

courseStageMaster = 10;

break;

}

if (IRProxValue > IRProxWhenWall10cm) //wall too close, need to back up

{

setMotorSpeeds(slowReverse,slowReverse);

delay(1000);

setMotorSpeeds(stopMotors,stopMotors);

leftTurnDone = false;

break;

}

if(buttonValue ==1 &&IRProxValue>IRProxWhenWall10cm)

//hit a wall, not pyramid, restart looking

{

haveFacedPyramid = false;

setMotorSpeeds(1300,1000);

delay(1000);

setMotorSpeeds(stopMotors,stopMotors);

}

}

}//end of pyramid switch

//if not facing pyrmid, check IR prox, if too close to a wall, back up.

//else, turn and break to ping pyramid IR again

if (facingPyramid == 0)

{

IRProxValue = analogRead(ci\_IR\_Proximity\_Sensor);

if (IRProxValue > IRProxWhenWall10cm) //wall too close, need to back up

{

setMotorSpeeds(slowReverse,slowReverse);

delay(1000);

setMotorSpeeds(stopMotors,stopMotors);

}

//turn right

setMotorSpeeds(fullForward,fullReverse);

delay(2000);

setMotorSpeeds(stopMotors,stopMotors);

break;

}

//move forward every 5 seconds in case stuck on a bump

if((newcount - oldcount)>5000)

{

oldcount = millis();

setMotorSpeeds(1800,1800);

delay(1000);

setMotorSpeeds(stopMotors,stopMotors);

}

else

{

newcount = millis();

}

buttonValue = digitalRead(ci\_button);

CharliePlexM::Write(12,LOW);

CharliePlexM::Write(9,LOW);

break;

}//end of case 9

// DEPOSIT CUBE UNDER PYRAMID - STAGE 10 - 13

case 10: //put arm down slowly and use flywheel to raise the pyramid

{

CharliePlexM::Write(12,HIGH);

CharliePlexM::Write(6,HIGH);

//turn right.

setMotorSpeeds(fullForward,fullReverse);

delay(1100);

setMotorSpeeds(stopMotors,stopMotors);

//lower arm

servo\_FlyWheelMotor.attach(ci\_Fly\_Wheel\_Motor);

servo\_PyramidArmMotor.write(1600); //slowly lower arm

delay(1500);

servo\_PyramidArmMotor.write(stopMotors);

//tip pyramid

servo\_FlyWheelMotor.write(1650);

delay(1500);

courseStageMaster = 11;

CharliePlexM::Write(12,LOW);

CharliePlexM::Write(6,LOW);

break;

}

case 11: //wait for MC1 to deposit cube. Get current encoder position

{

CharliePlexM::Write(12,HIGH);

CharliePlexM::Write(3,HIGH);

setMotorSpeeds(stopMotors,stopMotors);

servo\_FlyWheelMotor.write(1700);

leftMotorInitialPosition = encoder\_LeftMotor.getRawPosition();

CharliePlexM::Write(12,LOW);

CharliePlexM::Write(3,LOW);

break;

}

case 12: //back up slowly to get cube under pyramid. XXX encoder counts

{

CharliePlexM::Write(12,HIGH);

CharliePlexM::Write(11,HIGH);

servo\_FlyWheelMotor.write(1700);

setMotorSpeeds(1300,1300);

leftMotorCurrentPosition = encoder\_LeftMotor.getRawPosition();

if((leftMotorCurrentPosition - leftMotorInitialPosition)>500)

{courseStageMaster=13;}

CharliePlexM::Write(12,LOW);

CharliePlexM::Write(11,LOW);

break;

}

case 13: //flywheel lowers pyramid, with cube inside

{

CharliePlexM::Write(12,HIGH);

CharliePlexM::Write(8,HIGH);

servo\_FlyWheelMotor.write(1400);

delay (1000);

courseStageMaster=14;

servo\_FlyWheelMotor.write(stopMotors);

CharliePlexM::Write(12,LOW);

CharliePlexM::Write(8,LOW);

break;

}

}//end of switch statement

//get course stage

boardComs.write(courseStageMaster);

if (boardComs.available())

{courseStageSlave = boardComs.read(); }

//CHECK STAGE

if(courseStageMaster != 0)

{

if(courseStageMaster!=courseStageSlave)

{

if(courseStageSlave>courseStageMaster)

{

courseStageMaster = courseStageSlave;

}

}

}//end of checking stage

ui\_Mode\_Indicator\_Index = 1;

}

break;

}

case 2: //Calibrate motor straightness after 3 seconds.

{

if (bt\_3\_S\_Time\_Up)

{

if (!bt\_Cal\_Initialized)

{

bt\_Cal\_Initialized = true;

encoder\_LeftMotor.zero();

encoder\_RightMotor.zero();

ul\_Calibration\_Time = millis();

servo\_LeftMotor.writeMicroseconds(ui\_Motors\_Speed);

servo\_RightMotor.writeMicroseconds(ui\_Motors\_Speed);

}

else if ((millis() - ul\_Calibration\_Time) > ci\_Motor\_Calibration\_Time)

{

servo\_LeftMotor.writeMicroseconds(ci\_Left\_Motor\_Stop);

servo\_RightMotor.writeMicroseconds(ci\_Right\_Motor\_Stop);

l\_Left\_Motor\_Position = encoder\_LeftMotor.getRawPosition();

l\_Right\_Motor\_Position = encoder\_RightMotor.getRawPosition();

if (l\_Left\_Motor\_Position > l\_Right\_Motor\_Position)

{

// May have to update this if different calibration time is used

ui\_Right\_Motor\_Offset = 0;

ui\_Left\_Motor\_Offset = (l\_Left\_Motor\_Position - l\_Right\_Motor\_Position) / 4;

}

else

{

// May have to update this if different calibration time is used

ui\_Right\_Motor\_Offset = (l\_Right\_Motor\_Position - l\_Left\_Motor\_Position) / 4;

ui\_Left\_Motor\_Offset = 0;

}

EEPROM.write(ci\_Right\_Motor\_Offset\_Address\_L, lowByte(ui\_Right\_Motor\_Offset));

EEPROM.write(ci\_Right\_Motor\_Offset\_Address\_H, highByte(ui\_Right\_Motor\_Offset));

EEPROM.write(ci\_Left\_Motor\_Offset\_Address\_L, lowByte(ui\_Left\_Motor\_Offset));

EEPROM.write(ci\_Left\_Motor\_Offset\_Address\_H, highByte(ui\_Left\_Motor\_Offset));

ui\_Robot\_State\_Index = 0; // go back to Mode 0

}

ui\_Mode\_Indicator\_Index = 4;

}

break;

}

}

if ((millis() - ul\_Display\_Time) > ci\_Display\_Time)

{

ul\_Display\_Time = millis();

bt\_Heartbeat = !bt\_Heartbeat;

CharliePlexM::Write(ci\_Heartbeat\_LED, bt\_Heartbeat);

digitalWrite(13, bt\_Heartbeat);

Indicator();

}

}

// set mode indicator LED state

void Indicator()

{

//display routine, if true turn on led

CharliePlexM::Write(ci\_Indicator\_LED,!(ui\_Mode\_Indicator[ui\_Mode\_Indicator\_Index]&(iArray[iArrayIndex])));

iArrayIndex++;

iArrayIndex = iArrayIndex & 15;

}

void setMotorSpeeds (int left, int right)

{

servo\_RightMotor.writeMicroseconds(right);

delay(200);

servo\_LeftMotor.writeMicroseconds(left);

}

void getFacingPyramid()

{

pyramidValue = Serial.read();

if (letters == true)//looking for 'A' and 'E'

{

if(pyramidValue == 65||pyramidValue == 69)

{

facingPyramid = 1;

}

else

{

facingPyramid = 2;

}

}

else //looking for 'I' and 'O'

{

if(pyramidValue == 73 || pyramidValue == 79)

{

facingPyramid = 1;

}

else

{

facingPyramid = 2;

}

}

return;

}

### Slave code

//libraries

#include <Servo.h>

#include <EEPROM.h>

#include <uSTimer2.h>

#include <CharliePlexM.h>

#include <Wire.h>

#include <I2CEncoder.h>

#include <SoftwareSerial.h>

void retractCubeArm();

//software serial

SoftwareSerial boardComs(8,9);

//servos

Servo servo\_BackArm;

Servo servo\_CubeGrabArm;

Servo servo\_CubeGrasper;

//port pin consts

//analog

const int ci\_Hall\_Sensor = A0;

const int ci\_Back\_Arm\_Servo = A1;

const int ci\_Cube\_Grab\_Arm = A2;

const int ci\_Cube\_Grasper = A3;

unsigned int ui\_Cal\_Count;

unsigned int ui\_Cal\_Cycle;

//GENERAL VARIABLES

int courseStageMaster = 0;

int courseStageSlave = 0;

//PICK UP CUBE VARIABLES

int hallSensorValue;

int backArmServoPosition;

int frontArmExtended = 50;

int frontArmRetracted = 160;

int backArmExtended = 185;

int backArmRetracted = 90;

int armVertical = 135;

int grasperOpen = 0;

int grasperClosed = 185;

int cubeGrabbingServoPosition;

bool cubeObtained = false;

int hallByCube = 495;

// the setup function runs once when you press reset or power the board

void setup() {

Serial.begin(9600);

boardComs.begin(19200);

//set up servos

pinMode(ci\_Back\_Arm\_Servo, OUTPUT);

servo\_BackArm.attach(ci\_Back\_Arm\_Servo);

pinMode(ci\_Cube\_Grab\_Arm, OUTPUT);

servo\_CubeGrabArm.attach(ci\_Cube\_Grab\_Arm);

pinMode(ci\_Cube\_Grasper, OUTPUT);

servo\_CubeGrasper.attach(ci\_Cube\_Grasper);

//set up hall sensor

pinMode(ci\_Hall\_Sensor, INPUT);

courseStageSlave=0;

courseStageMaster=0;

}

// the loop function runs over and over again until power down or reset

void loop() {

switch (courseStageSlave)

{

case 0: //wait to start, ensure starting positions

{

servo\_BackArm.write(backArmRetracted);

servo\_CubeGrabArm.write(frontArmRetracted);

servo\_CubeGrasper.write(grasperOpen);

break;

}

case 1: //lower arms, increase stage upon completion

{

servo\_BackArm.write(backArmExtended);

servo\_CubeGrabArm.write(frontArmExtended);

servo\_CubeGrasper.write(grasperOpen);

delay(1000);

courseStageSlave=2;

break;

}

case 2: //ping hall sensor, stage increase to 7 when magnet is found

{

hallSensorValue = analogRead(ci\_Hall\_Sensor);

servo\_BackArm.detach();

Serial.println(hallSensorValue);

if(hallSensorValue<hallByCube)

{delay(500); courseStageSlave = 7;}

break;

}

case 3: //raise front arm only

{

servo\_CubeGrabArm.write(frontArmRetracted);

delay(1500);

courseStageSlave=4;

break;

}

case 4: //wait

{

break;

}

case 5: //lower front arms

{

servo\_CubeGrabArm.write(frontArmExtended);

delay(1000);

courseStageSlave=6;

break;

}

case 6: //ping hall sensor, increase stage when found

{

hallSensorValue = analogRead(ci\_Hall\_Sensor);

Serial.println(hallSensorValue);

if(hallSensorValue<hallByCube)

{delay(500); courseStageSlave=7;}

break;

}

case 7: //activate grasper, get cube, raise arms

{

servo\_BackArm.attach(ci\_Back\_Arm\_Servo);

//activate grasper

servo\_CubeGrasper.write(grasperClosed);

delay(500);

retractCubeArm();

delay(200);

servo\_BackArm.write(backArmRetracted);

delay(2000);

servo\_BackArm.detach();

servo\_CubeGrabArm.detach();

courseStageSlave=8;

break;

}

case 11: //deposit cube in the chute

{

//raise cube arm to above chute

servo\_CubeGrabArm.attach(ci\_Cube\_Grab\_Arm);

servo\_CubeGrabArm.write(armVertical);

delay(1500);

//drop cube

servo\_CubeGrasper.write(grasperOpen);

delay(2000);

courseStageSlave = 12;

break;

}

case 12: //attach the back arm

{

servo\_BackArm.attach(ci\_Back\_Arm\_Servo);

break;

}

case 14: //wag tail because course is completed

{

servo\_BackArm.write(60);

delay(500);

servo\_BackArm.write(120);

delay(500);

break;

}

}//end of switch statement

//get course stage

if(courseStageMaster != 9) //provided not in stage 9

{ boardComs.write(courseStageSlave);}

if(boardComs.available())

{ courseStageMaster = boardComs.read();}

if(courseStageMaster ==0) {courseStageSlave = 0;}

//compare stages to determin if stage had changed

if(courseStageSlave!=courseStageMaster)

{

if(courseStageMaster>courseStageSlave)

{courseStageSlave=courseStageMaster;}

}

}//end of loop

void retractCubeArm()

{

for(int i=frontArmExtended; i<frontArmRetracted;i+=5)

{

servo\_CubeGrabArm.write(i);

delay(45);

}

return;

}

# Product Evaluation

The product meets several of the design objectives, but does not unfortunately meet all of them. The main objective that is not met is that the prototype is not capable of locating the pyramid. The prototype can locate the cube on the wall. It hooks onto the wall using the plates on the cube and back arms. It drives forward about two feet to bring the cube forward if it starts behind the cube arm. The servo motor at the end of the cube arm obtains the cube, then the robot maneuvers away from the wall. The prototype is also able to drive over the bumps on the floor and has the potential to locate the pyramid. Unfortunately, the prototype cannot find the pyramid due to a communication delay caused by the serial communication channels.

Assuming the pyramid is located, the robot would then drive in a straight line towards the pyramid. The prototype drives straight, but if it hits a bump and gets knocked off course, it would begin scanning again. To detect when the robot reached the pyramid, an IR proximity sensor would be used, but due to a shortage of IR sensors, the prototype has a limit switch. The limit switch is not ideal, as it often triggers prematurely.

Since the pyramid IR sensor and the chute don’t line up, after the limit switch is triggered, the prototype turns right slightly so they are in line. Then the pyramid arm comes down, tips the pyramid up, the cube arm drops the cube down the chute, and the prototype backs up so the cube is under the pyramid.

The prototype is consistent in picking up the cube and if the limit switch does not trigger prematurely, it is consistent in getting the cube under the pyramid.

Through spending time fixing coding bugs will the robot be able to locate the pyramid.

## Objectives That Were Met

* The number of moving systems is under 15 (there are 7): By designing versatile components for the robot, multiple subsystems were created that could perform more than one of the small tasks required of the robot. An example of this would be the flywheel system designed to pick up the pyramid. By using round wheels that spun at a common axle, it was avoided having to use multiple parts to align and tip the pyramid.
* The cost of the prototype was under budget: the robot made use of available components in the lab, and only a few components were custom made or ordered for design purposes.
* The number of user set up functions is under 5: the only user function to be performed is motor calibration.
* Number of successful runs to pick up the cube is over 80%: By creating an arm that also served as a rail when it came down, the robot could effectively “latch” to the wall to avoid the robot drifting away from the wall. This not only removed the need for a sensor to monitor the distance from the wall, it enabled us to pick up the tesseract in the same location on our robot each time.
* The product fits in a locker: To fit the robot inside the locker, the components were designed to arm to fold and start in an upwards position. This made use of the vertical space of the locker, so all components of the robot could fit within the locker space.
* It is capable of driving on various surfaces. The implementation of performance wheels that have variable tread lets us change the material of the tread to best suit the different surfaces the robot would have to drive over.
* It is capable of driving over the power conduits: The 6” wheels of the robot provide enough clearance for the robot to drive over the conduits, and the gear ratio of the drive train allows for enough torque to power the robot over the conduits.
* The mass of the product is under 5kg: From complete bill of materials of the components used in the Solidworks model, the robot has a total weight of just over 4kg.
* The accuracy of locating the cube on the first pass is greater than 80%: When the arms latch on to the side wall, the robot will almost always follow the wall, and will be able to detect the cube along the wall.
* Can distinguish between the 4 different IR frequencies the pyramid emits: The robot has different modes for picking up both “A/E” and “I/O” frequencies.
* Weight of objects that can be lifted/tipped is greater than 500g: The flywheel has a relatively high coefficient of kinetic friction (about 0.4) and when forced against the wall at the end of the arm, will be able to generate enough consistent frictional force to lift an object.

Table - Objective That Were Not Met

|  |  |  |
| --- | --- | --- |
| Objective That Was Not Met | Why It Was Not Met | Steps That Can Be Taken to Meet It |
| The operational runtime less than 5 minutes | The robot is not capable of locating the pyramid. | Perform more tests and iterations on the sections of code that focus on locating the pyramid. |
| Delays During Operations are less than 3s | There is a bug in the code that causes a delay when the two microcontrollers are communicating. | Spend more time debugging the code to find and remove what is causing the delay. |
| Turning Radius is less than 10 cm | The robot is long which results in sliding while turning | Iterate the physical design to have the robot be shorter. |
| Accuracy of locating the pyramid on the first pass | The robot can occasionally locate the pyramid. There is a delay with the communication. | Debug the code. |
| Obtain the cube if it is in the very corners | The arms that manipulate the cube can only move in the vertical plane. | Redesign the arms to be able to rotate, or have the back arm on angle so it can pull the cube forward if it is in the corners. |

## *Improvements*

One major problem faced by our robot is the delay caused by the communication between microcontrollers. This has been caused by some aspect of the additional serial communication channels. This could be fixed by having a microcontroller with more input ports to eliminate the need for a second microcontroller.

To increase the range that the cube could be grabbed, more IR proximity sensors could be added to the sides and back of the robot so the robot would be capable of turning around and searching the other end of the wall.

Another improvement to increase the range that the cube can be found would be to put the arms on angles so that they can reach into the corners. To implement this the chassis would have to be redesigned slightly to allow the servo motors to be on angles.

The drive system could be redesigned to allow 4 wheel drive to decrease the turning radius, or the chassis could be redesigned to allow the wheels to be closer together. Decreasing the turning radius would make searching for the pyramid easier and allow the robot to move away from the wall quicker.

Finally, the housing for the IR sensor could be designed to be smaller to allow it to be placed more towards the center of the robot so that the pyramid lines up with the chute.

# Works Cited

## **Information**

(August 2014). *Hall Effect Sensor*. Retrieved from <http://www.electronicstutorials.ws/

electromagnetism/hall-effect.html>

*What materials reflect infrared light?*Retrieved from*<*https://www.reference.com/science/

materials-reflect-infrared-light-80087ca386b2d887>

## **Photos**

AndyMark. *Cytron 13A, 5-25V Single DC Motor Controller.*Retrieved from

<http://www.robotshop.com/ca/en/cytron-13a-5-30v-single-dc-motor-controller.html>

AndyMark. *6" Performance Wheel w/ ½" Bearing (am-0132)*. Retrieved from

<http://www.andymark.com/Performance-p/am-0132.htm>

Adafruit. *Standard Size – High Torque – Metal Gear Servo.* Retrieved from

<https://www.adafruit.com/product/1142>

AndyMark. *40T 1.25 Module 0.375" Hex Bore, Steel Bevel Gear (am-2620).* Retrieved from

<https://www.andymark.com/product-p/am-2620.htm>

AndyMark. *Blue Nitrile Roughtop Tread, 1" Wide (am-3309).*Retrieved from

<https://www.andymark.com/product-p/am-3309.htm>

AndyMark.*Green Grippy Tread (am-2611).*Retrieved from <https://www.andymark.com/

green-grippy-tread-10-foot-roll-p/am-2611.htm>

AndyMark. *Long Roughtop Tread, Nitrile rubber (am-0566).*Retrieved from

<https://www.andymark.com/product-p/am-0566.htm>

RobotShop. *RS-550 Motor 19300rpm 12V 62.49oz-in.*Retrieved from

<http://www.robotshop.com/en/rs-550-motor-19300rpm-12v-6249oz-in.html>

*Tower Pro Micro Servo SG90*. Retrieved from

<http://www.rcshopbd.com/wp-content/uploads/2015/04/2013-07-18-06083533883.jpg>

*Unknown name and author.* <http://media.digikey.com/Photos/NMB

%20Tech%20Photos/PPN7A12C1.jpg>

*Unknown name and author. <*http://4.bp.blogspot.com/-7zEhOJrUYyU/UPxBn9SHvsI/

AAAAAAAAAvQ/S96Tj0sQO8I/s1600/100\_0616.JPG>

VexPro. *VersaPlanetary Gearbox*. Retrieved from

<https://www.vexrobotics.com/versaplanetary.html>

Westcoast Products. *15T Bevel Gear with 3/8" Hex Bore. Retrieved from*

<http://www.wcproducts.net/217-3344>

## Progress Report #1

**What we've done**

In our initial team meeting, we began with the problem definition and recorded initial concept ideas. Over reading week, we each generated additional ideas and concept sketches. During the first week back, we further defined the problem by identifying customer requirements and engineering specifications. We completed a Quality function deployment chart (QFD).

Additionally, a high level project outline has been created detailing the plan for each week.

**What we're doing**

We are currently brainstorming various additional subsystems. During our next meeting, will begin concept selection and feasibility of combinations for the final design.

We have divided the project into various subsystems which include; the drive system, object location, object collection, and object storage during transfer.

**Next Steps**

Our group will use decision matrices of our possible solutions in tandem with our requirements to pick out the best sub-systems four our group, based on our problem definition. Also, we will begin modeling parts and designing the entire system. We have decided to complete a SolidWorks model before the building of the different sub-systems, which will hep us not only with visualizing the robot, but also with identifying any complications with mating parts that might arise.

We are also planning a teambuilding exercise.

**Backup Plan**

If none of the combinations of subsystems appear to be feasible, we will return to developing additional concept generations, and recreate the decision matrix. If a final concept cannot be determined before the end of the week, the schedule we have will be reevaluated.

## Progress Report #2

**What We’ve Done**

We completed concept selection with decision matrices, and finalized the designs of the subsystems. We researched into various motors, wheels and sensors and chose components for our model, and began modeling our design in SolidWorks. Through the process of building the prototype, various subsystems were modified due to space limitations and component availability. We have begun testing the various sensors we are implementing, such as the hall effect and light sensor. All necessary electronic components for the prototype have been ordered.

**What We’re Doing**

We are in the process of testing the various subsystems, as well as coding. Once all ordered parts arrive, we will finish building the prototype. The SolidWorks modeling is almost complete.

\* Continue to code: Using the data obtained from when the sensors were tested, we can implement them for proper orientation to begin detection of different components of the course. This includes the tesseract (hall effect sensor) and pyramid/walls (ultrasonic sensor or light sensor, IR sensor).

\* Test subsystems: Run multiple tests on each subsystem to identify possible issues and potential for failure. It is important to identify any flaws and go through an iteration process for each one because one subsystem failing could halt our sequence for achieving the overall task.

\* SolidWorks modelling: In addition to working on a Cad model of our prototype using the Vex library, we are concurrently working on researching parts to use for our final product and creating CAD components of the subsystems. When the subsystems are finalized, assemblies can be merged into a larger assembly of the final product.

\* Finish building prototype: The most important task, proper testing cannot be done unless a prototype is completed. This goal is paramount to what will be achieved for this week in the project plan (finish building, and starting to code the robot).

**Next Steps**

* Code
* Test
* Debug
* Repeat (x∞ until due date)
* Finish designing and order (laser cut) the cube slide
* Design mount for electro-magnet
* Come up with name and logo

**Back Up Plan**

* -gear ratios for speed/torque
* -if the electromagnet doesn’t work, swap to a mechanical grasper