MAE322

Final Design Report

Team
Trust The Process

SaRR
The Process

Team Members
Justin Chae ‘23
Joshua Coleman ‘22
Karla Soto Cuevas ‘23
Hudson Godfrey ‘23
Mohamed Hamza ‘23
Jonathan Melkun ‘23

Table of Contents

1. Executive Summary

The objective of team “Trust the Process” was to design and fabricate a Search and Rescue Robot (SaRR) that can navigate a three-module course to deliver a medical kit to a stranded victim. The team robot, named ‘The Process’, must pass a one foot drop test, manually navigate an obstacle course, non-autonomously traverse a 1 ft wall, and autonomously search for a med-kit receptacle, dropping off the kit upon arrival.

“The Process” utilizes a four-wheel drive system. Each side of the drive system is powered by a single belt and powered together using a timing belt. Two 7 -inch arms extend from the front of the SaRR, rotating up and down to assist in breaching the wall and folding underneath the SaRR securely upon touchdown on the other side of the wall. Two photon sensors are placed at the front of the robot to detect the light source emanating from the med-kit receptacle, and an algorithm autonomously drives the SaRR to this location. An inclined chute containing the med-kit manually unhinges with the use of a small servo motor to deposit the med-kit.

The dimensions of our SaRR are 18 inches in length, 18 inches in width, and 9 inches in average height with a peak height of 16 inches at the top of the med-kit chute and battery pack. We expect “The Process” to travel at a maximum speed of 4 ft/s with our dual current motor selection, with an average travel speed of 2 ft/s, given that the autonomous feedback portion of the course should be traversed with care. The robot is designed to transport a medical kit over and around obstacles, such as a 12”-high wall and subsequently deliver the kit, weighing 3.1 pounds, to a ‘victim’ with a flashlight in 15 seconds. The SaRR weighs around 30 pounds and costs approximately $350 to manufacture. It is also controlled manually with an RC aircraft controller and autonomously with an Arduino program sent to a Teensy 3.5 microcontroller.

After iterations of designing, prototyping, and testing, a final robot consisting of the dimensions, specifications, and design as described above was constructed. It was determined that the robot excels in manual driver control and is capable of autonomously finding the “victim’s” location. There were some complications that arose in regards to wall traversal and the chassis strength but overall the robot performed as expected. For future iterations of “The Process”, changes would be made to the arm system in order to properly help the robot traverse the wall.

2. Introduction

The objective of this project was to design a Search and Rescue Robot (SaRR) that could successfully navigate an obstacle course to deliver a med-kit to a notional person in need. Inspiration for the project came from the emergency response to 9/11, and the significant progress that has been made with this category of robots since the tragedy. A SaRR would be useful to find and assist victims in the rubble, an environment too dangerous for a human to navigate. Another advantage of the SaRR is speed given its relatively small size and weight, as every second counts when providing medical assistance. The obstacle course simulates this environment with a maze and a wall, and the victim is modeled by a basket with a light beacon.

One of the main design philosophies adopted by "Trust The Process" was to keep the design as simple as possible. The team followed Professor Nosenchuck's advice and agreed that the fewest number of moving parts would lead to the lowest chance of failure. Another philosophy central to the design was modularity. To make it through the obstacle course, the robot must make it through different sections. As such, the design of the robot was split into different modules to tackle different sections of the obstacle course. The first module was the drive system, controlled by an RC controller. The second module was the arm system, designed to assist the robot in making it over the 1ft wall. Finally, the third module was the med-kit drop-off system, which delivered the med-kit to a basket at the end of the course.

For the initial design process, ideas were researched and devised individually, then combined together in an all-hands brainstorming session. Because of the individual research, many different ideas were posed to solve the various portions of the objective. For instance, for the drive system a wheel-free, vibration driven robot idea inspired by a "HexBug" bristlebot toy was considered. For traversal over the wall, inspiration was taken from claw wheels from a monster truck toy popular in the 80's. In an attempt to narrow down and eliminate ideas, the following design was initially decided on: a four wheel drive system powered by timing belts and motors, a front-mounted arm to assist the robot over the wall, and a ramp and door system to deliver the med-kit.

Modeling was accomplished in CREO following the initial design decision. The first iteration of the robot was scheduled to be completed by Thanksgiving break. Throughout this process, the robot followed the same general design that was planned, but small details were changed to be able to fit the robot on the chassis, for better functionality, etc. Also, version 1 of the robot did not include the medkit delivery system or light sensors, as the team focused on breaching the wall first. **Figure 1** shows a drawing of the first iteration of the robot. Important parts to note are the single, 12-inch long aluminum arm, asymmetric arm motor placement, bevel and spur gear train to fit the motor on the chassis, and acrylic chassis.

After testing, it was clear that iteration 1 was not effective in breaching the wall. Using the information gathered from the tests performed on version 1 of "The Process",

version 2 of the SaRR was constructed over Reading Period and included an improved arm mechanism, medkit delivery system, as well as photo and proximity sensors for the final autonomous navigation section. The second and final iteration is shown in **Figure 2**. Note a vertically mounted arm motor for symmetry, the medkit delivery system on the back of the robot, and two 7-inch arms instead of one singular arm.

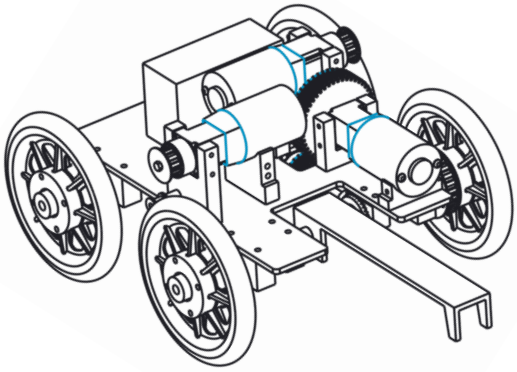


Figure 1: "The Process" Full Assembly Version 1

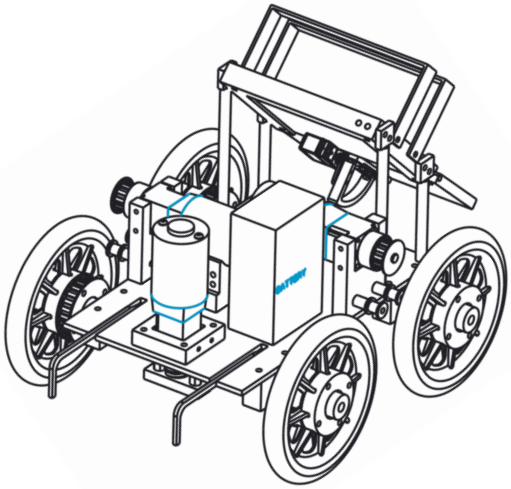


Figure 2: "The Process" Full Assembly Version 2

3. Specifications

The dimensions and specifications for “The Process” was originally determined by a number of factors including overall weight, speed, cost, feasibility, etc. However, one of the greatest factors that impacted the dimensions of the robot was the maze traversal portion of the course. Given that there would be a certain distance between the walls of the maze, it was crucial that the robot fit comfortably within the confines of the walls while still being able to make turns at corners.

Although the maze portion of the course was eventually removed due to time constraints after the version 1 robot was complete, the initial dimensions for the first version of the robot were kept. Although this led to difficulties fitting three motors in the given space, a compact, space efficient design was still able to be assembled.

Entire Robot (Figure 2)

- Dimensions
  - 20 x 17 x 9 inches (max height)
- Motors
  - Quantity: 4
  - Type: drill (x3) & servo motors (x1)
- Overall Weight
  - 29.2 lbs

Drive System (Figure 3)

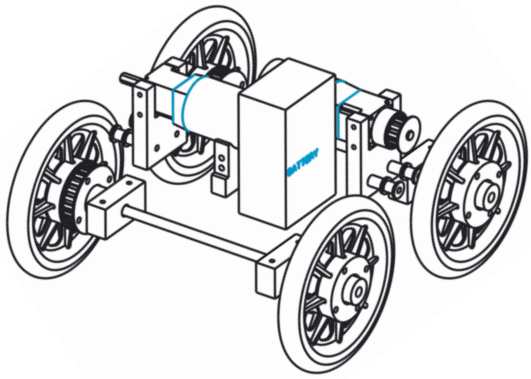


Figure 3: Drive System Full Assembly

- Dimensions
    - 20 x 12 x 6 inches (arm lowered)
- Motors
    - Quantity: 2
    - Type: drill motor
- Wheels
    - Diameter: 6 inches
    - Material: rubber
    - Hub and Gear Material: PLA
- Differential steering
    - Left and right side powered independently
    - Front and back wheel on each side powered using a timing belt connected to the drill motor
    - Max Speed: 4 ft/s
- Estimated Cost:
    - $107

After reviewing the recorded videos of previous SaRR models, our group agreed on having a robot with four wheels. Each side of the robot would be run by a single motor with the front and back wheels powered together by a timing belt connecting to the motor. In regards to the material selection of the wheels, rubber was preferred because of its high friction coefficient and common availability. It was decided that the wheels should be less than a foot in diameter, which is the height of the obstructing wall. These choices were made to mitigate the wheel slippage as well as allow our robot to overcome the wall climbing difficulties of the previous models.

To arrive at the current wheel design, the group discussed several treaded designs (including treads with sloped fronts to help climbing), two wheeled and four wheeled robots, and tire sizing. Although a tread design would allow for a more consistent friction during climbing, it also would require the SaRR to slide the treads while turning. It was noted that this was a reason why the treads fell off on one of the previous designs in the videos. It also would make the robot less efficient than a wheeled design, which has one point of rotation. A two-wheel system caused concerns around potential swinging of the robot during movement and direction changes as well as the inability to consistently turn efficiently. This would be an unstable setup for the wall climbing portion of the project since there would be many inconsistent dynamic variables to the robot as it attempts to climb the wall. On the other hand, a four-wheel system has more stability than a two-wheel design. It also has more power to climb the wall given that all four wheels are working towards pushing the robot upwards and over the obstruction. Having four wheels also reduces the possibility of the robot getting stuck in a position where no wheels are touching the wall while attempting to traverse it. In addition, this

setup allows for more efficient turning which is helpful for both the obstacle course navigation and autonomous sections of the SaRR test.

In search of finding the optimal tire size, it was determined that thick wheels would make the robot heavier and potentially make it harder to traverse the wall. Medium sized wheels with a large number of grooves would help in creating a lightweight robot that would carry itself over the obstruction with the help of the friction from the wheels. In the end, the final SaRR iteration tested utilized 8 inch rubber wheels. These wheels allowed the SaRR to drive and turn smoothly; however, the wheels slipped at the wall obstacle and were not able to drive the robot over even the initial step. Additional 8 inch rubber treaded wheels were ordered in anticipation of this, but they did not arrive in time.

Arm Mechanism (Figure 4)

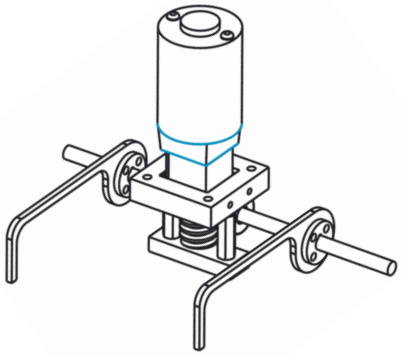


Figure 4: Arm System Assembly

- Dimensions
  - 7 x 2 x 3/16 inches
- Arm
  - Quantity: 2
  - Material: 6061 Aluminum
  - Point of rotation: shared with the wheel shafts
  - Chassis cutouts (4 inch long, 0.3 inch wide) in the front of the robot for the arm to rest at a vertical position before it is needed
- Motor
  - Quantity: 1
  - Type: CIM Motor
  - Max Torque: 12.5 lbf*in

    - Gear Ratio: 200:1 with planetary gears, 18:1 with worm gear for a total of 3600:1 to lower 3000 RPM to ~1 RPM
- Cost:
    - 1 worm: cast iron, ~$40

The initial main mechanism to get the SaRR over the wall was a long arm attached to the front of the robot. The group first decided to have one arm instead of two, as one arm in the center should accomplish the same task as two utilizing the same power supply. Additionally, a single-arm design provided room on the sides to mount the motor. However, concerns arose later in the design process about the balance and symmetry of the robot, so the group decided to use two smaller arms instead.

It was believed that one arm would be a simpler mechanism earlier in the process, but after designing a two arm mechanism, it was found to be much simpler and required less parts. This was because the motor was able to be mounted vertically, shown in **Figure 4**. With this configuration, a simple worm gear assembly was used to translate the rotation of the motor shaft to the arm shaft. The arms were keyed to the shaft and fixed in place with shaft collars, so the arms rotate rigidly with the shaft.

Additionally, the point of rotation of the bar was originally in the center of the body of the robot. But, it was found that moving the point of rotation to the front of the robot reduced the length of arm required to reach the top of the wall. It also allowed the back wheels to still make contact with the ground, assisting the robot by supplying a horizontal force in addition to the vertical force from the arm.

Two thin notches had to be cut out of the chassis to allow the arms to have space to rest in an upright position. The arm, in combination with the wheels, was intended to elevate the front of the robot above the wall, then push the robot forwards over the wall once it made it over the second step. Considerations were made to make sure the arm does not intersect with the ground in its down position, was short enough to not prevent the robot from getting over the first step, and safely tucked away below the chassis to not prevent the robot from driving after it has fulfilled its purpose of traversing the wall.

Med-Kit

Figure 5: Medkit Dispensing Mechanism Assembly

- Dimensions
  - 6.5 x 8.5 x 4 inches
  - Angle: 45 degrees (leaves a 3x safety factor for our current estimate of the coefficient of static friction)
- Material
  - ⅛ in 5086 series aluminum
- Total weight
  - 2.5 lbs of aluminum (assuming width all faces of the chute = 0.125 in)
- Motor
  - Quantity: 1
  - Type: servo motor
  - Torque: 0.075 lb-ft
- Estimated cost: $55

When designing the med-kit drop-off portion of the robot, the team wanted to focus on having a small, lightweight mechanism with a minimal number of controlled moving parts. The concern of wanting a lightweight mechanism was to support another design goal the team decided upon of having a front-loaded robot, or having a forward center of mass. With our armed robot design, the biggest concern is getting the first set of wheels over the wall, whereas the back two should follow. The med-kit dropper will be mounted on the back of the SaRR such that motors, sensors, and wiring take up the majority of the space in the front. Thus, it is key for it to be lightweight, given that the med-kit itself is already 3 pounds.

Our next concern of having a small number of moving parts greatly affected the narrowing down of brainstormed ideas. First, we conceptualized a flat trough in which the med-kit would sit. In this trough, a pushing plate would be held in place by a set of two triangular, spring-loaded teeth, compressing a spring that would serve to launch the med-kit off the edge of the SaRR when released. The benefit of this iteration is that the setup is really straightforward as well as having only one controlled part, but we had difficulty deciding on how to autonomously retract the teeth. Another similar conception we had was to have hinged flaps like a pinball machine to hold back a similar spring-tensioned pushing plate. While the same in essence, this idea had a more obvious solution on how to control our release mechanism, but we ran into difficulties with the interference of the flippers with the med-kit and finding a properly fitted spring. Finally, we took the best of both ideas, incorporating minimal moving parts by letting the med-kit drop itself off with the force of gravity doing all the work.

Once the team settled on this design, the decision was made to clear the zone underneath the incline, replacing the solid material with a simple two supporting poles on the elevated corners. This way, the center of gravity should be lower down in the SaRR, but more importantly, it leaves room for the other groups to design flexibly with other mechanisms, whether it be wiring, motors, etc.

Another concern that drove our design was the conflicting requirements of having a secure hold on the med-kit, while also making it easy to free from the SaRR at the end of the course. The initial idea was to encase the med-kit completely in aluminum; instead, we opted for an aluminum skeletal holding chamber due to our desire for a lightweight robot. Furthermore, we designed the chute with a 0.5 inch tolerance on each side so that there is no chance for the straps and other parts sticking out from the kit preventing the package from sliding out.

Besides the research to find a reasonable static friction value for 100% polyester on aluminum, our planned incline material, the group also looked into the best kind of motor to control the hinged door at the bottom end of the chute. A servo motor was selected for this purpose, for they control rotational motions with ease, are low cost, low profile, and lightweight, having almost no drawbacks besides the torque concerns, which have been analyzed in the sections to come. The servo motor turned a sliding mechanism attached to a door. This mechanism was 3D printed with PLA due to its more complicated design and its low weight.

Control System

The robot control system implemented both open loop control for basic driving and arm movement as well as autonomous control for specific tasks (wall traversal and medkit delivery). Lack of testing as a result of time constraints made the autonomous wall traversal unfavorable when compared to an open-loop approach. Thus, to traverse the first section of the course, the user would send commands via an RC remote. Once the robot had driven around an obstacle and then over the wall, the user would turn on

the autonomous medkit delivery system, and the robot would guide itself to the medkit delivery.

4. Project Management

Justin Chae (*Team Leader*)

Organized agenda for team meetings, responsible for keeping team on track for all build and design checkpoints, and oversaw construction of total robot with all subcomponents combined.

Joshua Coleman (*Programming Lead*)

Oversaw the programming of the SaRR, ensured all parts of the robot were controlled as specified, and contributed to the drive system operation and analysis.

Karla Soto Cuevas (*Material & Finances Lead*)

Oversaw the overall finances for the team, responsible for organizing a Bill of Materials which outlines the material assignment, fabrication or purchasing details, and cost of each part.

Hudson Godfrey (*Analysis & Modeling Lead*)

Oversaw all CREO finite element analysis for essential components, assisted in modeling of individual components and overall SaRR model.

Mohamed Hamza (*Electronics Lead*)

Oversaw the connections and layout/format of the wires and electronic systems of the SaRR, and contributed to the drive system operation and analysis.

Jonathan Melkun (*CREO Lead*)

Specialized in modeling and assembly in CREO, responsible for arm mechanism design and manufacturing, managed CNC operations

Justin Chae was selected as the leader of team Trust the Process, and every other member of the team had the leading role in a specific area over which their skills uniquely lie. The list above states each group member and their lead role and responsibilities. While all members of the team were leads in certains areas for this project, tasks were still shared amongst everyone and the designing/building of the SaRR was a collective effort.

As well as these overarching roles, the team members were separated into three groups of two, each corresponding to a different subcomponent of the SaRR. Justin and Jonathan headed the arm/wall ascent team, Karla and Hudson headed the med-kit drop-off and chassis team, and Joshua and Mohamed headed the wheels/drive team.

In terms of team meetings, three hours every Thursday afternoon allocated to our lab served as time to work together on the SaRR. Furthermore, every team member allocated time on Sundays for a team meeting to catch each other up on what they have completed this week, what they have had difficulty with and need assistance with, as well as setting expectations of what should be done by the next week's team meeting.

In addition, each subcomponent group met individually to discuss, design, and manufacture and assemble parts together as the deadline approached.

Table 1: Moscow Chart

Won't	Could	Should	Must
• Break wall	• Extraneous features (LED, sound, decorations)	• Reasonable weight (<100lbs) • Reasonable speed • Simple and elegant design (less parts) • Safe to handle • Smooth turning • Clean, commented code • Well documented process (meeting notes, drawings) • Cost efficient	• Get over wall • Traverse through maze autonomously without bumping into walls • Hold on to med-kit the entire time • Must be done by Dean's Date • Survive drop test • Be within the $750 budget • Clear communication within the team

Table 2: Expected Project Timeline	
Week 0 (10/17 - 10/23)	● Brainstorm ideas for robot (all subcomponents) individually ● Allocate division of work and timelines for PDR ● Open-loop drive and autonomous navigation to light source completed
Week 1 (10/24 - 10/30)	● Finalize idea for robot design ● Form groups to specialize in subcomponent design ● Finish PDR ● Deliver PDR Presentation ● Tabulate budget ● Allocate individual responsibilities
Week 2 (10/31 - 11/06)	● Review PDR comments ● Begin CREO ○ Design parts ○ Analysis ○ Assembly
Week 3 (11/07 - 11/13)	● Continue CREO ○ Shift towards more analysis & assembly of subcomponents ● 3D print rough prototypes (before lab so we can test/think during lab)
Week 4 (11/14 - 11/20)	● CREO ○ Assembling subcomponents into version 1 of a complete robot
Week 5 (11/21 - 11/27)	-- Thanksgiving Break --
Week 6 (11/28 - 12/04)	● Version 1 of robot complete
Week 7 (12/05 - 12/11)	● Robot testing ○ Drop test ○ Autonomous ○ Light finding ● Finalize revisions to robot
Week 8 (12/12 - 12/14)	● Final report/presentation

Table 3: Actual Project Timeline	
Week 0 (10/17 - 10/23)	● Brainstormed ideas for robot (all subcomponents) individually ● Allocated division of work and timelines for PDR ● Open-loop drive and autonomous navigation to light source completed (sample robot)
Week 1 (10/24 - 10/30)	● Finalized idea for robot design ● Formed groups to specialize in subcomponent design ● Finished PDR ● Delivered PDR Presentation ● Allocated individual responsibilities
Week 2 (10/31 - 11/06)	● Begin CREO ○ Drafted arm (chalkboard sketches in class) ○ Early assembly
Week 3 (11/07 - 11/13)	● Continue CREO ○ Finalized assembly of robot arm and chassis ● Started 3D printing gears and new wheel hubs
Week 4 (11/14 - 11/20)	● Manufacturing ○ Arm milled out from 12 inch long square aluminum tube ○ HDPE blocks bandsawed and milled including wheel mounts, drive and arm motor mounts, spacers, etc. ○ First prototype assembled, not functional
Week 5 (11/21 - 11/27)	-- Thanksgiving Break --
Week 6 (11/28 - 12/04)	● Version 1 of robot complete ● Initial testing conducted-wall breaching ● Collected improvements for second iteration
Week 7 (12/05 - 12/11)	● Version 2 begin ○ Finalized medkit mechanism, 3D printed and machined parts ○ Ordered and CNC machined chassis out of

	polycarbonate ○ Ordered new wheels, did not arrive in time ● Finalized revisions to robot
Week 8 (12/12 - 12/14)	● Version 2 of Robot assembled and completed ● Final Presentation complete ● First test of the second iteration done live on Dean's Date, no time for testing beforehand ● Final Design report complete

Table 4: Hours	
Week 0 (10/17 - 10/23)	● First Meeting ● 2 hours per person, 10 collectively
Week 1 (10/24 - 10/30)	● 9 hours collectively
Week 2 (10/31 - 11/06)	● 9 hours collectively
Week 3 (11/07 - 11/13)	● 12 hours
Week 4 (11/14 - 11/20)	● 9 hours collectively for team meeting ● Estimated 2 hour per day per member to meet deadline ● 40 hours collectively
Week 5 (11/21 - 11/27)	-- Thanksgiving Break --
Week 6 (11/28 - 12/04)	● Estimated 2 hour per day per member to meet extended deadline
Week 7 (12/05 - 12/11)	● Estimated 2 hour per day per member, 60 hours collectively
Week 8 (12/12 - 12/14)	● 8 hours per member on 12/13 (crunch time) ● 3-8 hours per member on 12/12 ● 88 collectively
Lab Time	● 7 weeks of 3 hour lab: 21 individually ● 126 hrs collectively

Lecture Time	• 6 times lecture met in shop, plus coming in early some days • 15 hrs collectively
Misc Out of Class Time Throughout Semester	• 150 hrs
Gross Total	560 hours (extremely approximate)

Overall Management

The team generally didn't have many issues throughout the semester. One aspect of our team was the large number of people compared to the other teams. This was found to be helpful when organizing smaller subteams because it allowed for two members to work on a single subsystem together without it being too overwhelming for a single member. It also allowed our team to work more modularly given that each team could work independently from the others. However, this 'advantage' also served to be a bit of a disadvantage at times. Given that we had so many people in our group, it made it difficult for everyone to work on the robot at once. For instance, if there were changes that had to be made to the total assembly by the drive subteam, there just wasn't enough space for other subteams to be working on the robot. Overall though, I feel as though it served more as an advantage throughout the semester.

In terms of meetings, our group met fairly consistently throughout the beginning portion of our build process. However, as the semester progressed, our team's schedule became busier and busier, thus making it difficult for all of us to meet at the same time. Due to this, we naturally progressed into a system of independently coming into the shop and working on various parts asynchronously. Generally it wasn't completely necessary for all of us to be in the shop at the same time (if anything, as mentioned above, it was almost better to have less people). Therefore, this new system of working independently throughout the week and then giving updates during our lab time ended up working quite well for our team.

As we progressed into the final week of building, our meeting hours increased significantly. However, given that many of us had various review sessions and other responsibilities to attend to during the week, we generally worked asynchronously during this time as well. At the end of each work day, I would go into the shop to see what progress had been accomplished during the day and relay an update to the rest of the team. Then, based on what had been accomplished and what was left to be finished, I wrote out a to-do list for the following day.

Overall there were many management lessons learned from this project. Our entire team can attest that every single part or assembly that we initially created in CREO took considerably longer to manufacture than we initially expected. Therefore, we

learned that we should allocate more time for manufacturing parts, especially given the lack of experience for our team. Additionally, another key lesson was in ordering parts. There were multiple instances where our team was unable to work because we were still waiting for a piece of stock material to come in to be machined. Had we more thoroughly planned ahead of time and had the foresight to order earlier, we could have potentially saved a considerable amount of time. Finally, one of the most important lessons we learned was the importance of taking the due diligence to plan out ahead. There's always a balance between sitting around and planning something versus actually going out and doing "it", whatever "it" may be. In our case, I think we didn't allocate enough time to properly plan and CREO out our designs. Had we given more time to CREO every part and assemble it into a proper assembly, we could have prevented many issues that ended up arising during our assembly process. In hindsight, taking the time to assemble the robot together virtually is well worth the time in the long run.

Overall the team performed exceptionally well. Although there may have been times where certain members contributed more hours than others, everyone still put in time to create our final robot. Fortunately we didn't have any major interpersonal issues and we managed to create a well structured team.

5. Detailed Design and Analysis

Drive System

Although the drive system's main role is to allow the SaRR to traverse in all directions, the drive system's main design parameters were constrained by its role in traversing the maze and the wall. To traverse the maze, the robot needed to be able to turn 360 degrees between walls that are 30 inches apart. This means the robot must fit within a 21 in. x 21 in. square (which has a 30 inch diagonal). The 18 in. x 12 in. dimensions were chosen in conjunction with the arm design team to make sure the robot would meet both team's needs. Although we ended up getting more space through the elimination of the maze task, the team stuck with the sizing which aided us with having a lightweight SaRR.

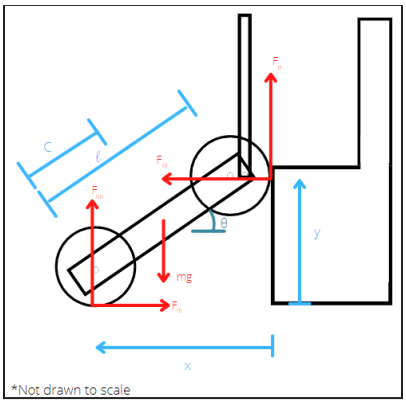


Figure 6:
FBD for wheels contacting walls

In previous years, multiple robots have had issues with wheel slippage when they first begin to accelerate. A relationship between the maximum friction and maximum torque applied was necessary to solve this issue. Starting with the no-slip condition:

$$\begin{aligned} v &= R\omega \\ \frac{dv}{dt} &= R\frac{d\omega}{dt} \\ I(m\frac{dv}{dt}) &= mR(I\frac{d\omega}{dt}) \\ I(\Sigma F_{\hat{v},robot}) &= mR(\Sigma T_{wheel}). \end{aligned}$$

Since this equation is a result of the general no-slip condition, the resulting equation can be applied to the wheels as they traverse up the wall. With an equation for the energy of the SaRR system, as well as a geometric constraint between the x and y positions, a relationship between the torques applied to each wheel can be created.

Using these equations, the maximum torque would be about 3.5 ft-lbs on the back wheels and 8.8 ft-lbs on the front when the SaRR is driving horizontally. At an estimated top speed of about 4 ft/s, both motors would be able to supply a torque of 7.9 ft-lbs each after a 20:1 gearbox. Thus, these values are well within the range of the current motors. See the appendix for more details on the motor sizing and drive load estimates.

When the SaRR is driving up the wall, the normal forces on the front wheels are fully dependent on the frictional forces from the back. This means it is best to try to maximize the back's frictional force. The preliminary estimates found that the current motors should have been able to supply more than enough torque for both wheels, even when the back wheel's torque starts at 8.2 ft-lbs and changes linearly with the x position to 9.2 ft-lbs. These were very rough estimates: the total weight and center of mass were not the same as the final design. Further estimates and tests showed that more torque was needed to climb the wall. The idea to use the arm to increase the upward force during the first section was presented and used during the final test. The final estimates are presented in the appendix.

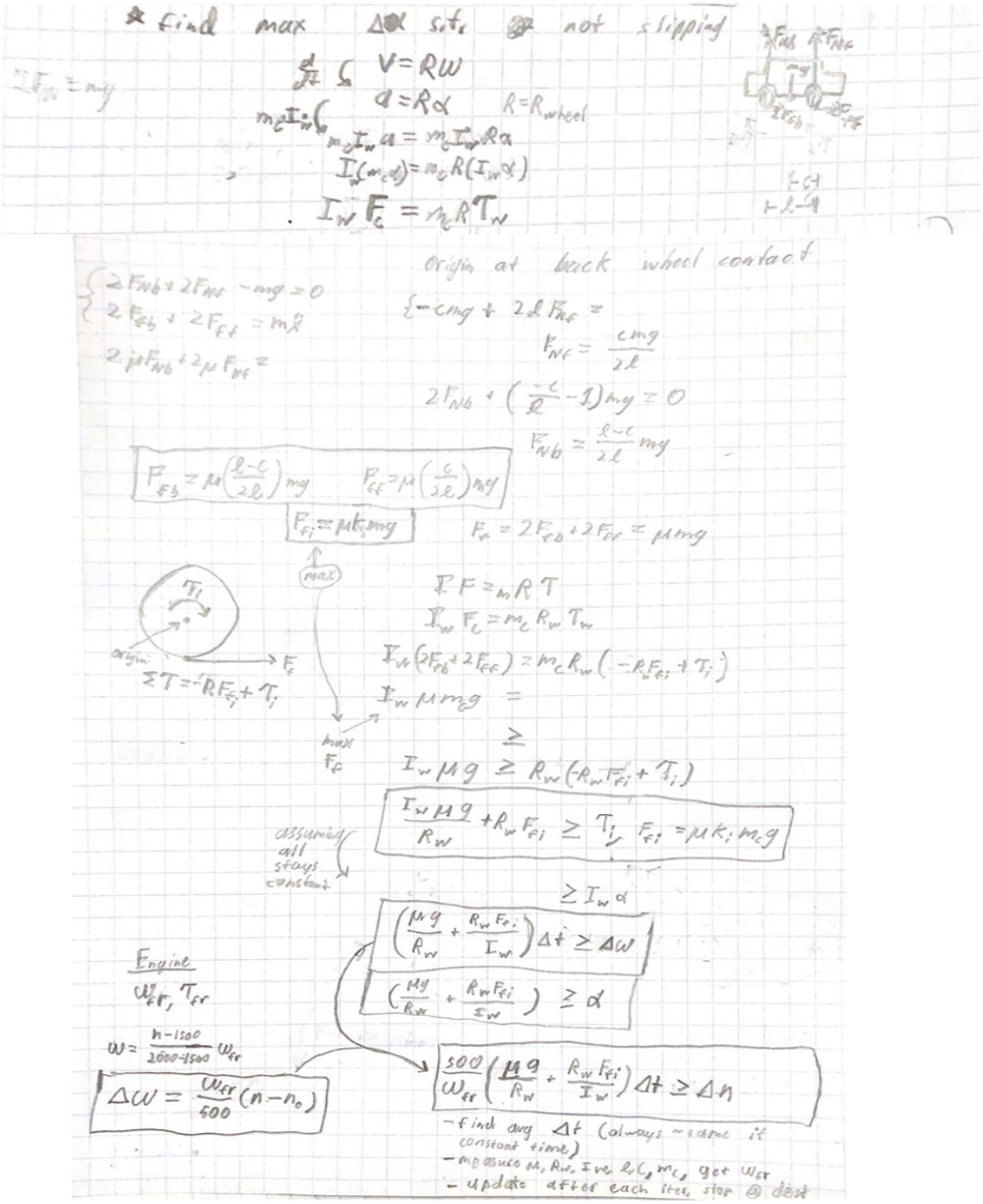


Figure 7: Drive system FBD diagram calculations

Arm

One consideration is if the arm would be able to support the weight of the robot when moving over the wall. The arm and motor would have to undergo significant torque as it is lifting the robot over the last step of the wall. A free-body diagram of the arm is shown in **Figure 8**.

Cross sectional area of arm A is 0.25 in^2

Applied torque M from the weight of the robot is 2in*30lbs= 60 lb*in

$$\text{Moment of inertia of arm I} = \frac{bh^3}{12} = \frac{(1)(0.25)^3}{12} = 0.0013 \text{ in}^4$$

$$\text{Maximum bending stress} = \frac{My}{I} = \frac{(60)(0.125)}{0.0013} = 5760 \text{ psi}$$

$$\text{Two arms, so stress on each arm is } \frac{5760}{2} = 2880 \text{ psi}$$

The yielding stress of aluminum is around 40000 psi, using a factor of safety of 2 this is still well below 20000 psi, so the arm will not be damaged by the weight of the robot.

Also, a gearbox is needed to increase the torque of the drill motor to push the robot over the wall. 60 lb*in is required to get the robot over the wall, calculated previously, so we would need to use a 10:1 gear ratio to account for a safety factor of 2. This can be accomplished with just planetary gears attached to the motor. However, the rotational velocity of the motor shaft at this ratio would be 360 RPM, which is much too fast and can damage the robot if it is pushed past the intended range of motion. As such, an additional set of planetary gears was added to the motor, bringing the gear reduction to 200:1. In combination with the 18:1 gear ratio of the worm gear mechanism, this multiplies to 3600:1, which brings the RPM of the arms to a safe speed. This also increases the torque output of the motor, ensuring that the motors will deliver more than enough force to lift the robot.

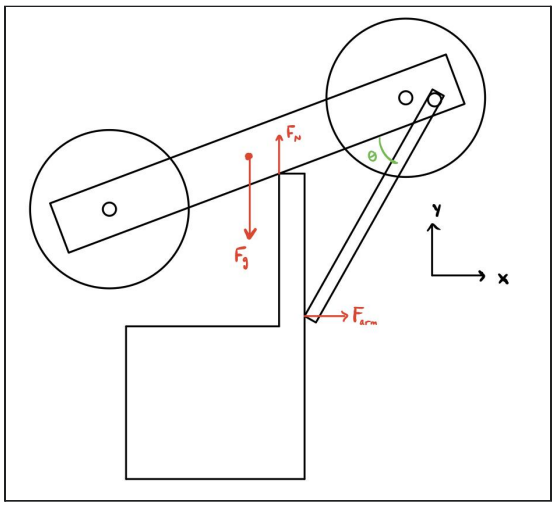


Figure 8: FBD for arm contacting wall

Arm FBD analysis:

$$\Sigma F_x = F_{arm}$$

$$\Sigma F_y = F_N - F_g = 0 \rightarrow F_N = F_g$$

Based on the FBD for the arm mechanism, while the robot is traversing the wall, it is advantageous to place more of the weight further forward on the robot in order to shift the center of gravity forward. By placing the center of gravity further forward, the amount of force required to push the robot forward is less. Given that the minimum force required by the arm is a function of the weight of the robot, by reducing the amount of total weight that is below the point of rotation of the robot's body against the wall, the minimum force required by the arm can be reduced. **Figure 9** shows the placement of the center of gravity of the robot obtained through a Creo analysis, which shows that it does in fact lie in the front half of the robot. It is closer to the center than the front half, but it should still assist in pushing the robot over the wall, as shown in the FBD analysis.

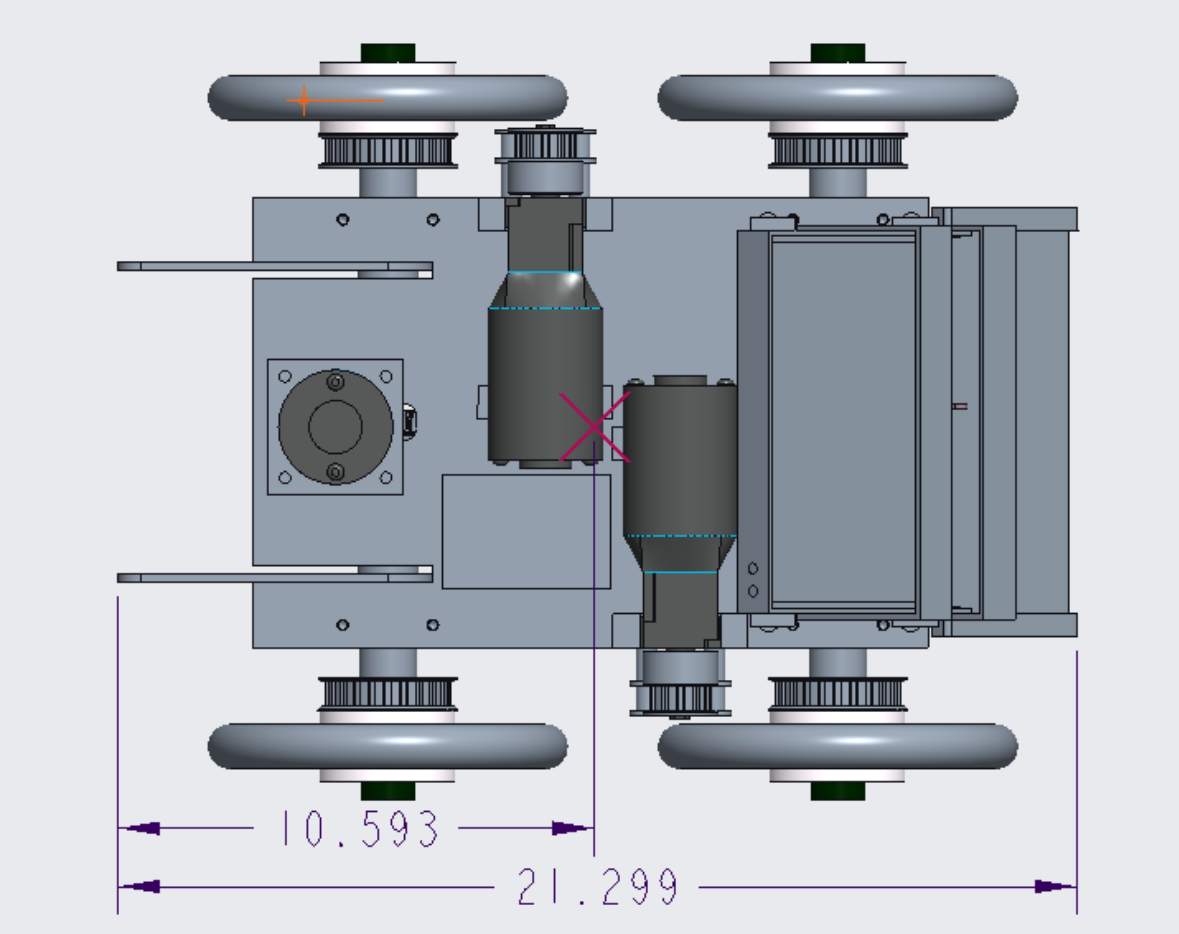


Figure 9: Red X shows the center of gravity, calculated with Creo. Note that this was approximated with the motors being assigned a heavy material (steel), and everything else being a lightweight material (PVC)

Key elements under large stress were the motors, which had to be secured to the chassis with motor mounts. Four screws prevented the motors from rotating within the motor mount blocks, then an additional four screws secured the motor mounts to the

chassis. This, in turn exerts bending stress on the chassis as well. To mitigate this, a strong polycarbonate material was used, as well as cutouts through the body were kept as small as possible to minimize stress concentrators that may lead to cracking. The two slits for the arms were likely the biggest issue with getting the robot over the wall, but they could not be taken out without significantly impeding the range of motion and in turn the functionality of the arms.

Med-Kit Dispensing Mechanism

Because the SaRR had many design constraints with the drive and arm subsystems, the med kit subgroup wanted to design with the concern of "modularity" in mind, seeking to have a flexible med kit design that could ideally fit wherever the team could manage after the designs of other subsystems were complete. In accordance with this idea, the device for this SaRR utilizes an inclined chute to dispense a med-kit into the elevated basket.

The med-kit mechanism is shown in **Figure 5**. An aluminum rectangular chute with dimensions 6.5" x 8.5" x 4" is designed to have a single opening on the bottom side where the med-kit will slide out using the force of its own weight. The med-kit can be inserted manually before the trial starts, and we designed it to be enclosed on all sides so that during the drop test or while the SaRR is inclined on its way over the wall, the med-kit would remain secure. Originally, the door was designed to be hinged upon the top surface of the open edge in order to minimize the interference with the med-kit as it slid out to the dropoff site. In order to power the door, we plan to use a small, lightweight servo motor to rotate the hinge 90°. The door flap, being approximately 3" x 8" x 0.5", will be on the order of 1lb given that aluminum has a density of 0.1lb/in^3, meaning that a motor with a torque of a mere 3 lbf*in would be sufficient.

Online, the servo motors that met these requirements all had plastic gears, which we were worried might strip, not from the weight of the door being pushed, but from the weight of the medkit pushing against the door during the closed position, which could impart shocks in particular moments, such as a wall dismount. For this reason, the group decided to use a high torque rated servo motor available from the shop's stock, rated at 35kg. This would be more than enough to support our door, and although we could not identify the degree of rotation from the serial number, only 90° of rotation were needed, which is a bare minimum for servo motors, and which was also verified experimentally through testing with Arduino.

However, the medkit dispensal device did not remain this way. As further analysis was completed with the version 2 of the SaRR, the team realized that the center of gravity needed to be further up in the robot, and as a result, measures to reduce the weight in the back were executed through the medkit construction. Instead of a fully enclosed design, the group opted for sheet metal strips to secure the medkit, which was an apt solution. Similarly, to reduce weight in the door and actuation devices, the team

turned to 3D printed PLA parts, measured out to match the horns on the servo motor we selected.

Also, in this rehaul of the medkit dispenser, the servo motor was moved to the bottom surface of the slide, rather than the top, to coincide with a hinge from the bottom, as shifting down the center of gravity would work in our favor as well as moving it towards the front of the robot. Unfortunately, the mounting device was not fully designed in Creo, and when it came to assembling the parts on the last day of shop time, unstable solutions such as duct tape had to be utilized in order to complete the SaRR in time for the deadline.

Aside from the servo motor troubles, several iterations of 3D prints were done to make sure that the interference between the slider along the backside of the door and the pinned arc that remained fixed to the motor was smooth. The arc-shaped piece neatly translated the rotational motion of the servo horn into linear motion along the slider, resulting in a well-spaced transition from closed to open positions. Although, this was never executed in practice, primarily due to the misplacement of the servo motor mount mentioned previously.

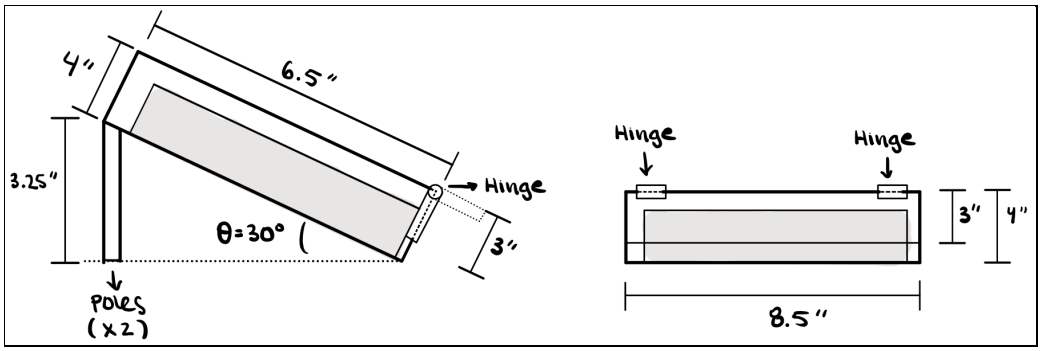


Figure 10:
Drawing of 6.5” x 8.5” x 4” aluminum med-kit dispensing mechanism

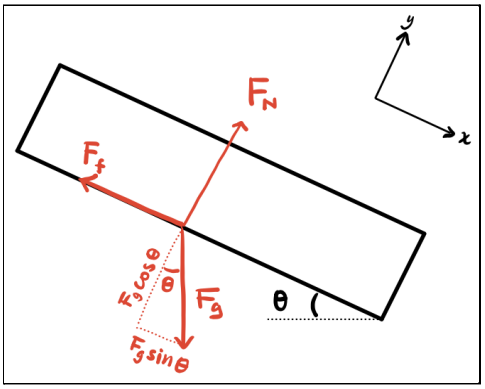


Figure 11:
Free-body diagram of med-kit inside of dispensing chute

The following are the calculations used to identify the minimum angle,θ, required to overcome static friction between the aluminum plate and the 100% polyester med-kit, which we found from the Amazon link to the product. Although we could not find the exact coefficient of static friction between aluminum and the polyester med-kit, we did research to find as close of a match in materials as possible. A close match was aluminum and canvas, which has static friction coefficient $\mu_s = 0.3$.The minimum angle of the chute is subsequently 16.7°. To account for the possibility that the static friction being larger in practice, we decided to set our angle θ= 30° for now, using a safety factor of roughly two in order to account for possible overlooked assumptions, like straps on the med-kits that would result in an experimentally higher friction coefficient. A preliminary test placing the med-kit on a ~30° rectangular sheet of aluminum, confirms that this angle is large enough for the med-kit to overcome static friction and slide.

$$\Sigma F_y = F_N - F_G cos\theta = 0$$
$$F_N = F_G cos\theta$$

$$\Sigma F_x = F_G sin\theta - \mu_s F_N = 0$$
$$F_G sin\theta - \mu_s F_G cos\theta = 0$$
$$F_G sin\theta = \mu_s F_G cos\theta$$
$$\mu_s = tan\theta$$
$$\theta \geq tan^{-1}(\mu_s)$$

The following preliminary designs were considered in the process of deciding to create an elevated chute:

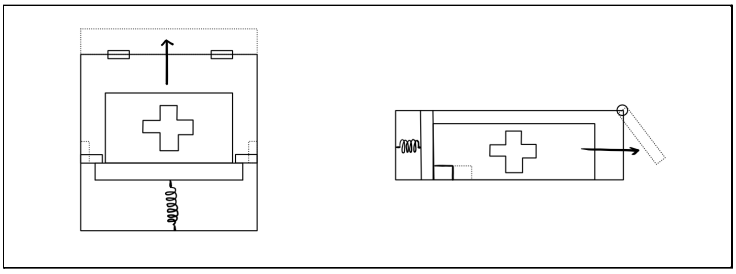


Figure 12:
Preliminary design #1 for the med-kit dispensing mechanism picturing a sliding plate attached to a spring that pushes outwards when triangular teeth recoil out of the chute.

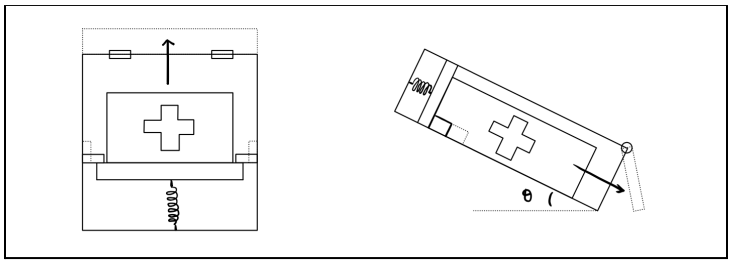


Figure 13:
Preliminary design #2 for the med-kit dispensing mechanism picturing a sliding plate attached to a spring that pushes outwards when rectangular teeth are powered by a motor to lift.

While rehauling the design for the medkit, a primary concern of the team was that parts made out of PLA may yield or even fracture under the loads of either the medkit or the torque from the servo motor. In order to avoid printing parts that would just break immediately, finite element analysis was executed in Creo in order to test these worries. Two tests were performed, one at a rated normal force on the thin slider surface of 5 lbf, and the other at 15 lbf, shown in Fig. 14 and Fig. 15, respectively.

This part would be under stress from the pin of the arc-shaped arm that connects to the servo motor. If the motor moves too fast, then it could break, but if the code sent to the servo was fine-tuned to operate in a slower manner, the part should maintain its structural integrity. The second test from Fig. 15 did result in a stress exceeding that of the yield stress at PLA at the weak corners, and this fact was conveyed to the code team to prepare counter measures.

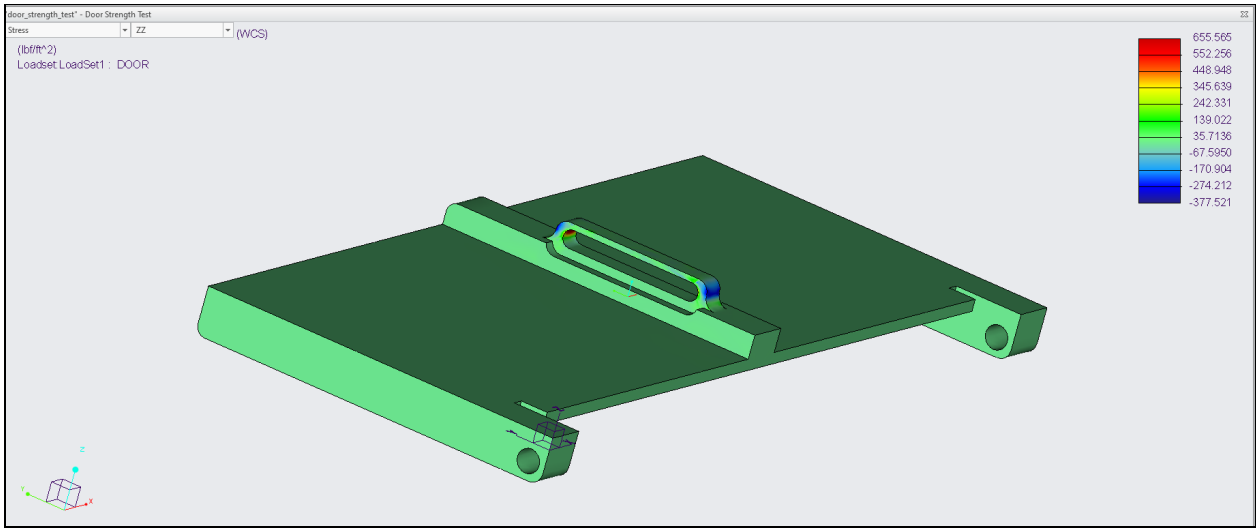


Figure 14:
Stress on the PLA med kit door slider resulting from a 5 lbf normal force along the length of the surface, which did not result in yielding of the material.

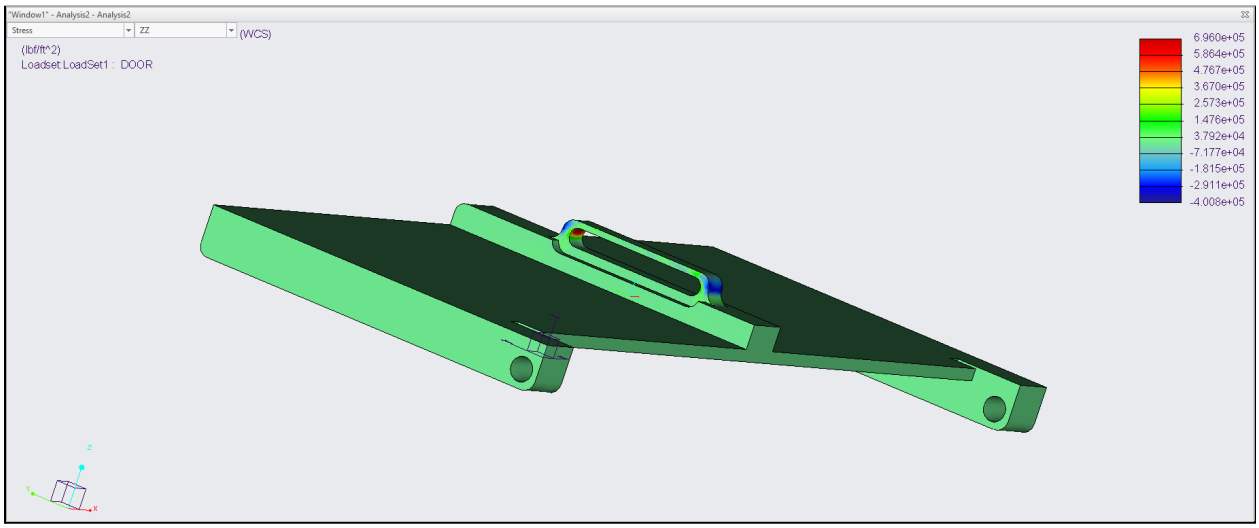


Figure 15:
Stress on the PLA med kit door slider resulting from a 15 lbf normal force along the length of the surface, which resulted in yielding of the material, but not fracture.

Control and Autonomous System

The robot's control system has sections corresponding to the three main tasks that it must accomplish during: open-loop, traversal of an obstacle course, wall breach, and autonomous medical kit delivery. Rather than directly sending signals to the arm, drive, and medkit drop-off servos, each section acts as a "virtual controller" input to the rest of the control system. For example, the autonomous systems save their control

outputs in the variables that would normally hold the values sent from the RC controller. While the data for the medkit door immediately controls the medkit servo, the other data is sent to the arm and drive control sections. These then send their outputs to the SaRR's three drill motors.

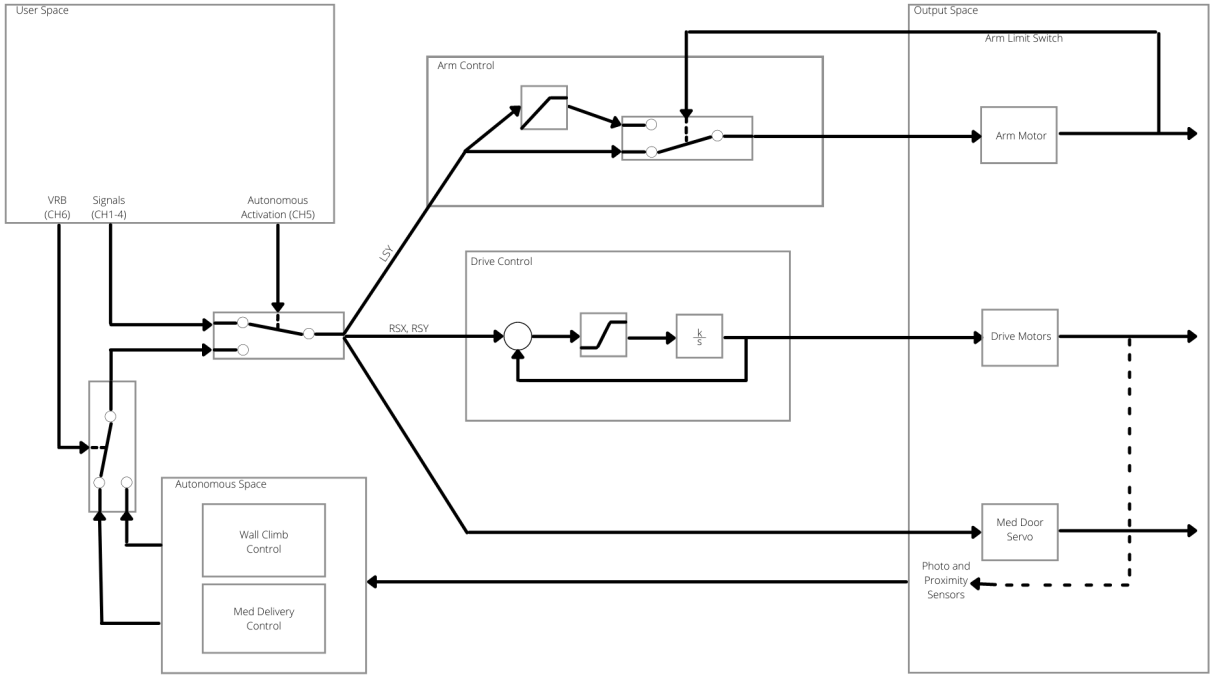


Figure 16:
Block Diagram of SaRR Control System

Both the arm and drive sections employ feedback in some fashion. The arm system allows the arm to move freely until it receives a positive feedback signal from a limit switch that's pressed at the arm's maximum angle. At that point, the arm controller limits it's out signals to below 1500, which allows for downward motion but not upward motion. The drive system controller uses feedback to control the rate at which the drive controller's output signal increases, thereby controlling the acceleration of the SaRR. The feedback signal comes in the form of the previous output value: the error between the current and desired values is constrained to a maximum change in rate and then sent into an integral controller. The integral controller's output is then sent to the drive motor.

Indirect feedback is then used by the autonomous wall climb (which was partially implemented but not tested or used) and medkit delivery sections. The two autonomous operation methods are best exemplified by the medkit delivery system. The first method is to perform an action until the sensors reach certain values. At that point, the next action can occur. The medkit control system uses this method in the beginning: it rotates until both photosensors read a high enough intensity. This ensures that at least

one sensor is pointing at the light. At that point, the medkit system uses the second method, which is to use a usual feedback controller. For this system, the photo sensor values are given to a proportional controller which updates the drive system's target values. It uses the first method alongside the second by stopping and dropping the medkit when a proximity sensor on the back of the robot reads that it is close enough to the drop-off box.

As will be discussed in section 7, the medkit autonomous system did not perform as planned. This was partially due to a lack of testing beforehand (the final SaRR test was the first autonomous test). Due to the fact that the control program was created to be easily adjustable, the issues with this system could be resolved in a very short time and with only a few tests.

6. Drawings (Creo)

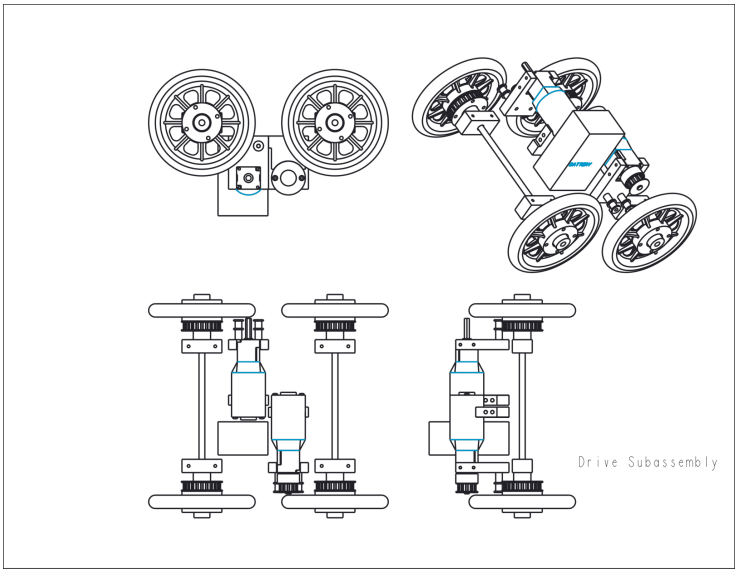


Figure 17: Drive Subassembly, with 8 inch diameter wheels, two CIM motors with motor mounts, belt tensioners, and wheel hubs.

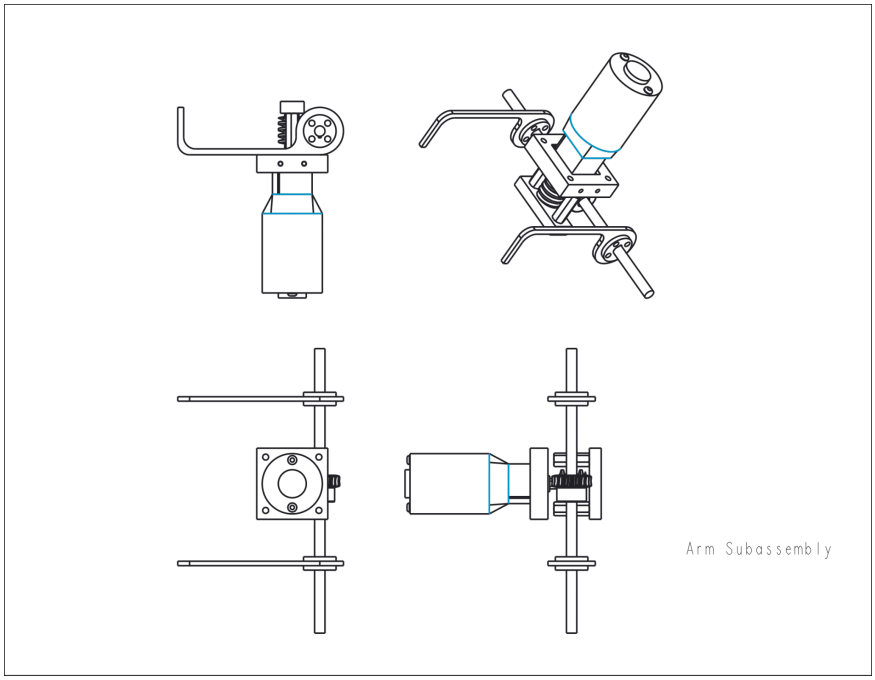


Figure 18: Arm subassembly, with vertically mounted motor, two 7-inch long arms, and a worm gear to transfer the motor torque to the rotating shaft.

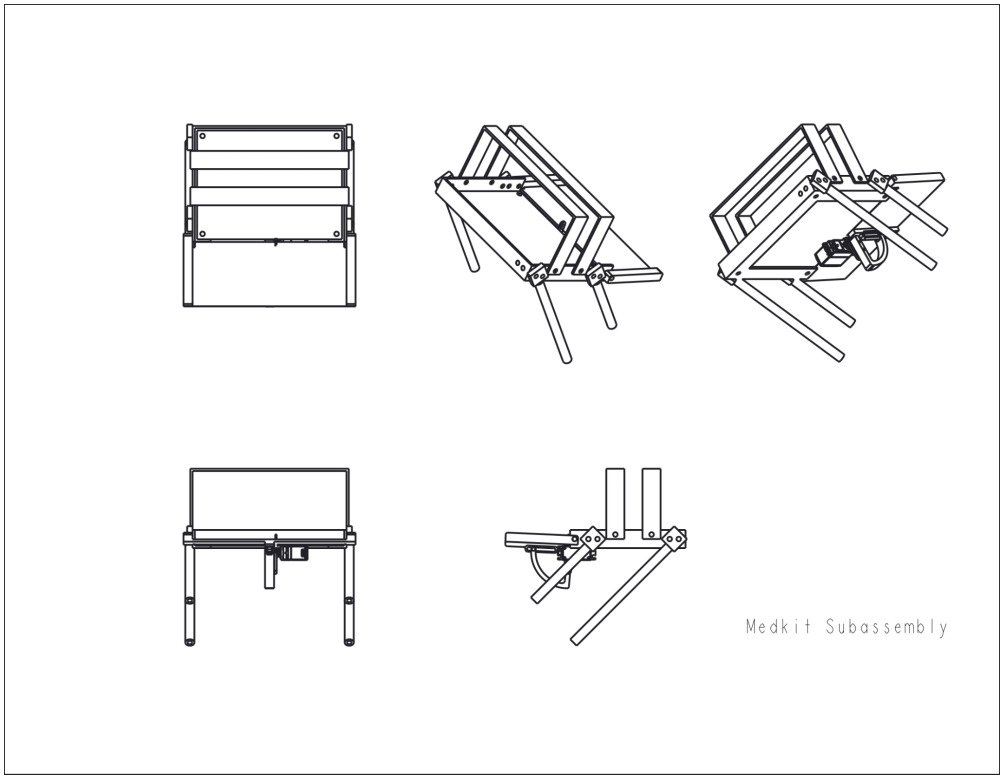


Figure 19: Medkit Delivery System Subassembly, with standoffs, servo motor for the door mechanism, and cage to secure medkit

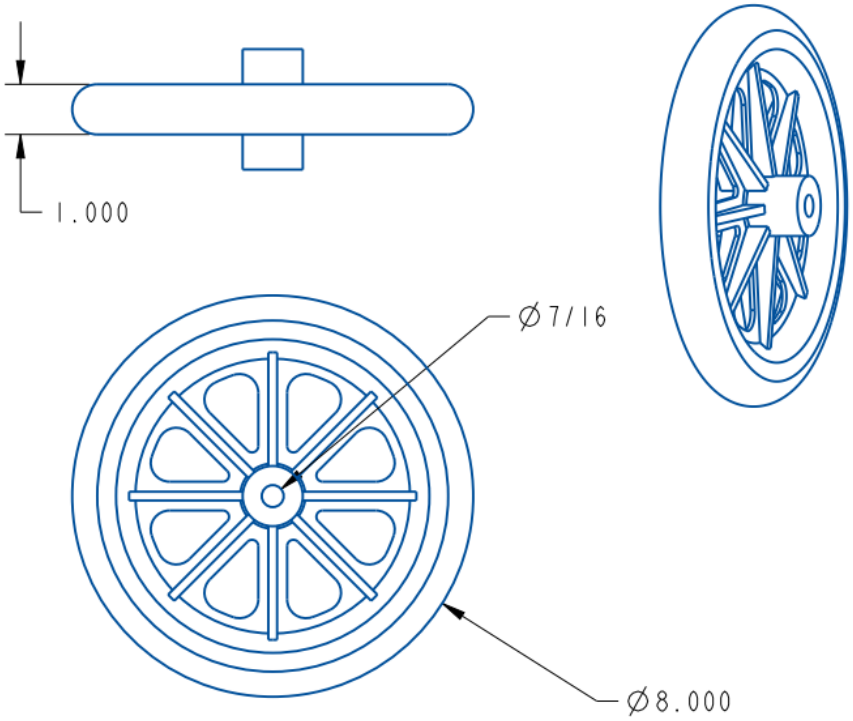


Figure 20: Wheel Drawing, with a bearing in the middle to secure to a 7/16 diameter shaft

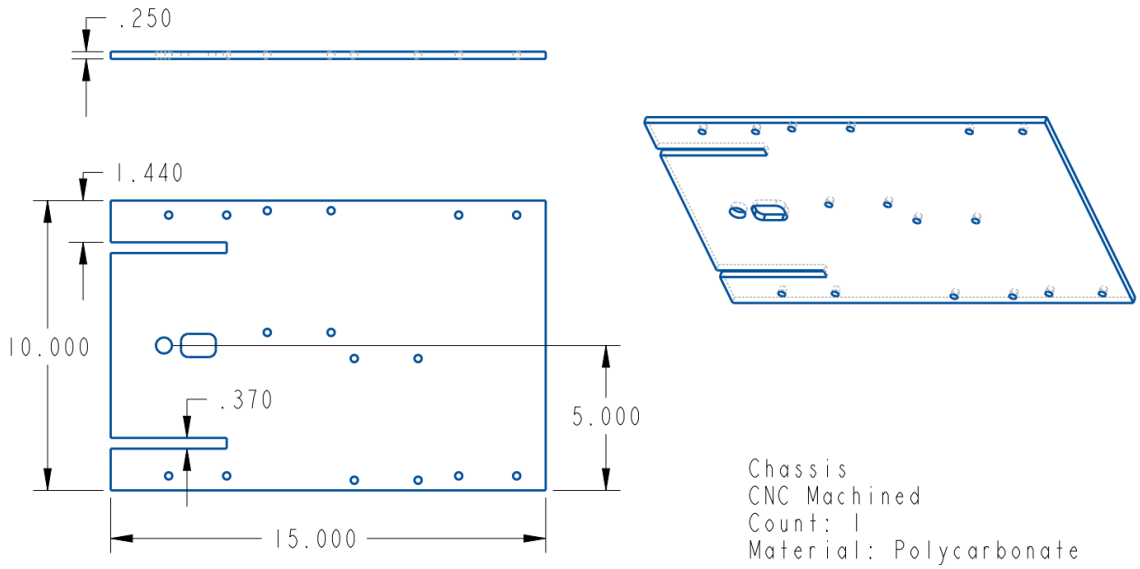


Figure 21: Creo Drawing of Body (part was bandsawed to overall dimensions then machined in CNC)

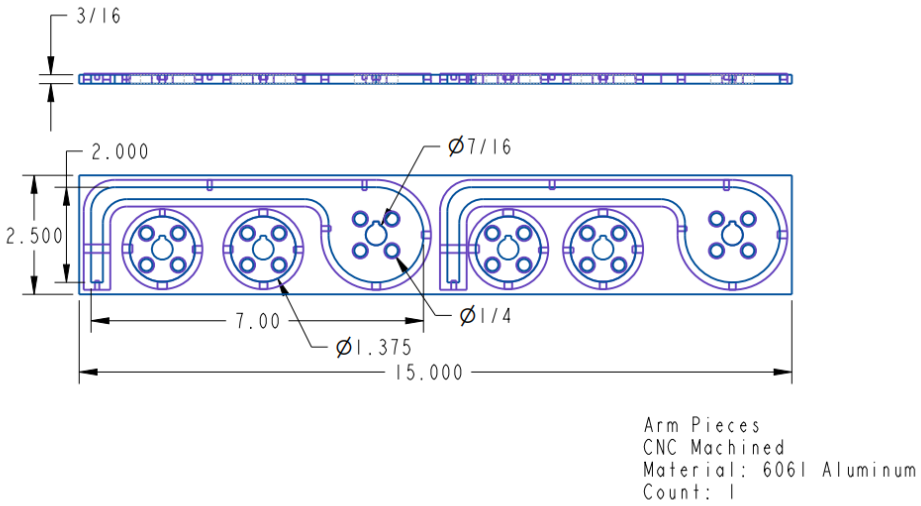


Figure 22: Creo Drawing of Two Arm Parts with Circular Reinforcements (part was bandsawed to overall dimensions then machined in CNC)

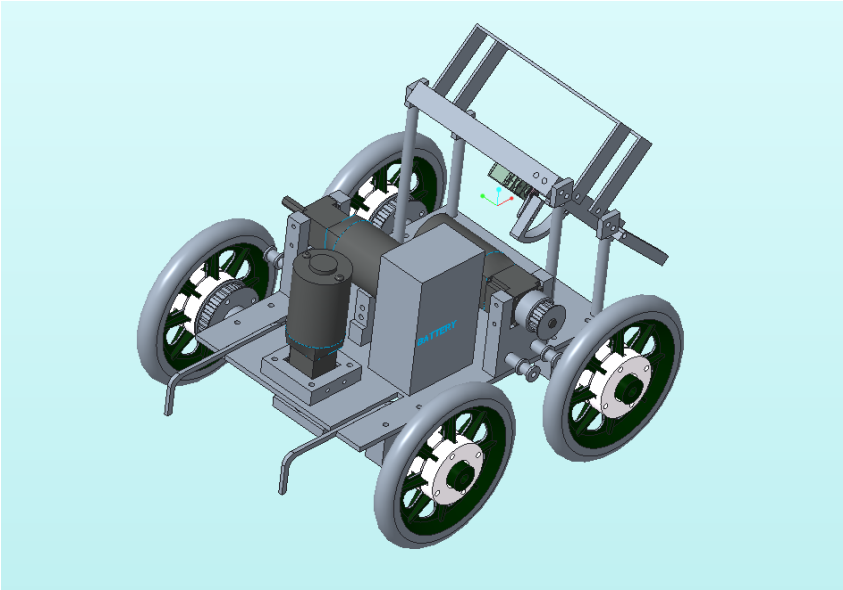


Figure 23: Final Assembly

7. Test Results

Version 1

Material choices were a key issue during the testing of the first robot. For instance, the chassis was made of unreinforced acrylic. As such, it was brittle and broke during the initial drive testing, and would have no chance to resist fracture from the central arm had it still been intact. This was mainly due to a cutout for the gear train for the arm mechanism, and was temporarily fixed with electrical tape. For the next iteration, we used a stronger polycarbonate material for the chassis. Also, when testing the arm mechanism, the gears broke because they were made of weak 3D printed PLA. For the next iteration, we planned to use metal gears for added robustness.

Additionally, the robot could not make it up the first step of the wall. This was due to the wheels not having enough friction and being too small to scale the first step alone. The 12-inch long arm was too long to fit on the first step, so it could not assist the robot in getting up the first step. We focused so much on getting the robot over the second step that we forgot the first step was an issue as well. There were also concerns of one arm not being enough to balance the robot, especially since version 1 did not have a symmetric motor placement. To address this, the one arm was split into two for better balance, and the arm motor was moved to the center of the chassis for better weight distribution.

Also, the underside of the robot was an issue during testing. In particular, a gear for the arm mechanism was hitting the top of the wall, jamming the mechanism and preventing the robot from getting over the last step. To mitigate this, the next iteration aimed to contain all underside components within the profile of the wheels, so it would be protected throughout the entire wall breaching process.

Version 2

The drop test dealt no damage to the SaRR and the robot was able to drive perfectly afterwards. The speed test was a success too. The robot traveled around the post and to the wall in a matter of seconds. This was likely due to the compact and light design, allowing for tight turning and a higher top speed. The drive subsystem was the best functioning part of the robot.

Unfortunately, the other parts of the test did not go as smoothly. Electronics were also a significant issue, as the robot would not respond to the controller. This was fixed by adding extra insulating tape to the connections, as the connectors seemed to be sparking and causing issues with sending power to the motors. Also, the chassis was an issue, much like in the previous iteration. Even though a stronger polycarbonate material was used, the material still bent and flexed, rendering the arms ineffective. Rather than tilt the entire robot up onto the first step, the arms simply bent the chassis. It is likely that the arms were supplying enough torque as calculated, but the force was being absorbed by the bending of the chassis rather than lifting the robot. This seemed to be the main issue preventing the robot from getting over the wall, since with a little

manual push the robot made it over as planned. The geometry of the design worked in theory, but had major shortcomings in practice.

The medkit delivery was unfinished due to time constraints, for as one of the last milestones, it was put off by the team in the face of more serious concerns. First, the robot followed the light source, but it moved incredibly slowly. This could be easily tweaked, as it just consists of changing a few numbers in the code, seen by our successful trials with the sample robot. The slow movement also caused the SaRR to have a huge turning radius as it entered its search routine for the light source, and had to be manually steered in the right direction due to space constraints. The robot made it to the basket eventually, but upon arriving, pushed the basket with the open medkit door. Had the medkit door been connected and running, it would have been a success if the eight inch wheels were mounted on the robot rather than the six inch ones, as tabulated throughout our designs. Alas, the medkit door mechanism was unfinished, so that was the end of the test.

8. Conclusions and Further Work

The objective of team “Trust the Process” was to design and fabricate a Search and Rescue Robot (SaRR) that can pass a one foot drop test, manually navigate an obstacle course, non-autonomously traverse a 1 ft wall, and autonomously search for a med-kit receptacle, dropping off the kit upon arrival. Named “The Process”, the SaRR utilizes a four-wheel drive system powered by belts and motors. Two 7 -inch arms extend from the front of the SaRR, rotating up and down to assist in breaching the wall and folding underneath the SaRR securely upon touchdown on the other side of the wall. Two photon sensors are placed at the front of the robot to detect the light source emanating from the med-kit receptacle, and an algorithm autonomously drives the SaRR to this location. An inclined chute containing the med-kit manually unhinges with the use of a small servo motor to deposit the med-kit. The SaRR is 18 inches in length, 18 inches in width, and 9 inches in average height with a peak height of 16 inches at the top of the med-kit chute and battery pack. It traveled at a maximum speed of 4 ft/s , weighs around 30 pounds and cost approximately $350 to manufacture.

After iterations of designing, prototyping, and testing, a final robot consisting of the dimensions, specifications, and design as described above was assembled. It was determined that the robot excelled in manual driver control and is capable of autonomously finding the “victim’s” location. There were some complications that arose in regards to wall traversal and the chassis strength but overall the robot performed as expected.

Many of the things that went wrong during the assembly process could have been prevented with further planning. Because of time constraints, parts were being manufactured before the full assembly was finalized in Creo, which resulted in parts not fitting properly, having to use washers to line up parts properly, and having to drill new holes for new parts. For future projects, finalizing the assembly before manufacturing would likely save time in the long run since less issues would arise later.

If we had more time, one improvement to the design could be better wheels with more traction to get the robot over the wall. In fact, we had ordered slightly larger wheels that had a higher friction coefficient, but they unfortunately did not come in time before testing. Having wheels with a larger diameter and a higher friction coefficient would definitely allow the SaRR to successfully traverse the wall, preventing the slippage of the wheels against the walls as we saw in our testing. Additionally, a shock absorbing system on the wheels should be added to protect the robot from a fall.

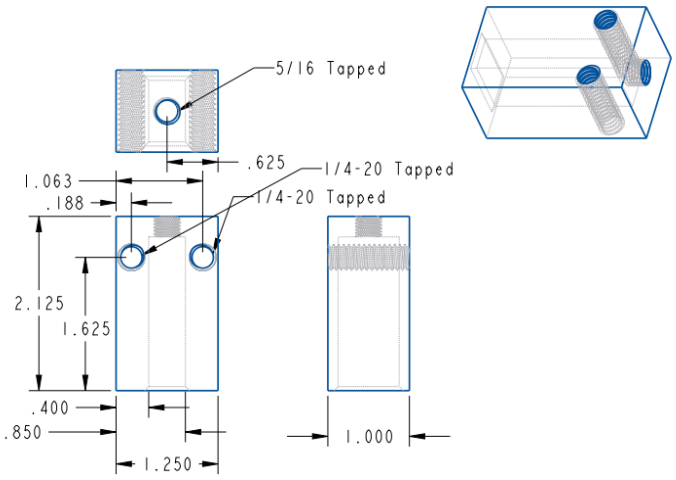
Apart from the wheels, the chassis should be reinforced with aluminum to prevent it from bending, a situation that arose when the arm was in use during the wall climbing phase of the SaRR test.

Lastly, experimenting more with the values in the code that were directly related to the speed and accuracy of the SaRR would have helped with having the robot work at a more reasonable pace during the autonomous section of the test as it was searching and driving towards the victim, represented by a bright light source.

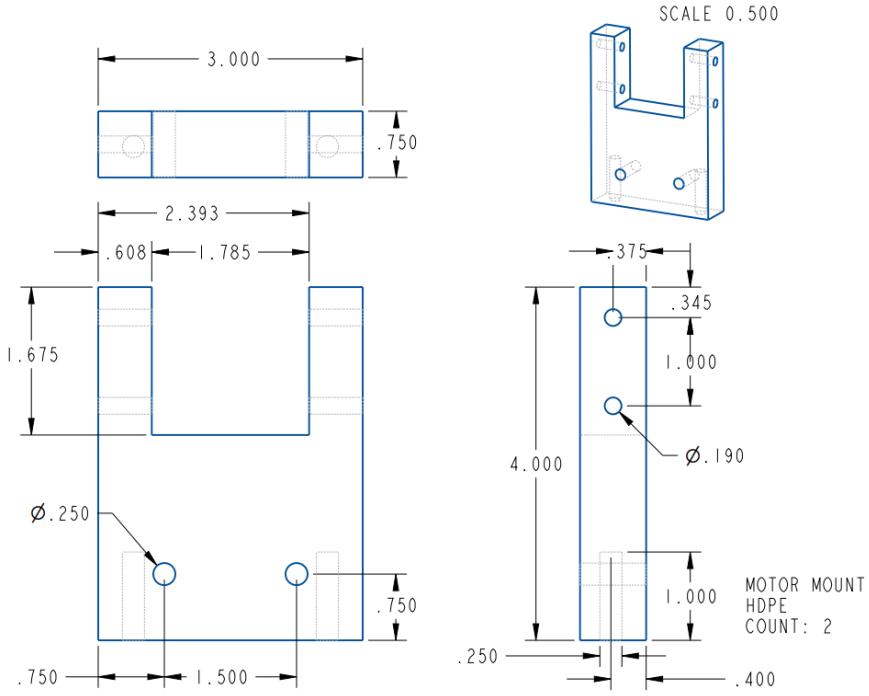
To move further with the SaRR, currently the robot is only equipped to move on flat ground and navigate a specifically shaped wall. To make the robot more versatile, the body of the robot can have flexible or servo-driven joints to attach to the wheels to allow it to traverse rough terrain and be allowed to climb more types of uneven surfaces.

Finally, with regards to whether a robot of this group's design could be mass manufactured and distributed, with the proper improvements introduced above, it would be a possibility in theory. With some "smart chassis" on the market going for around $150, and considering that our robot with its current set of wheels was only $250, plus the number of free parts, it is not out of the question, although the profit margins would be slim or perhaps negative given the man hours put into the machine.

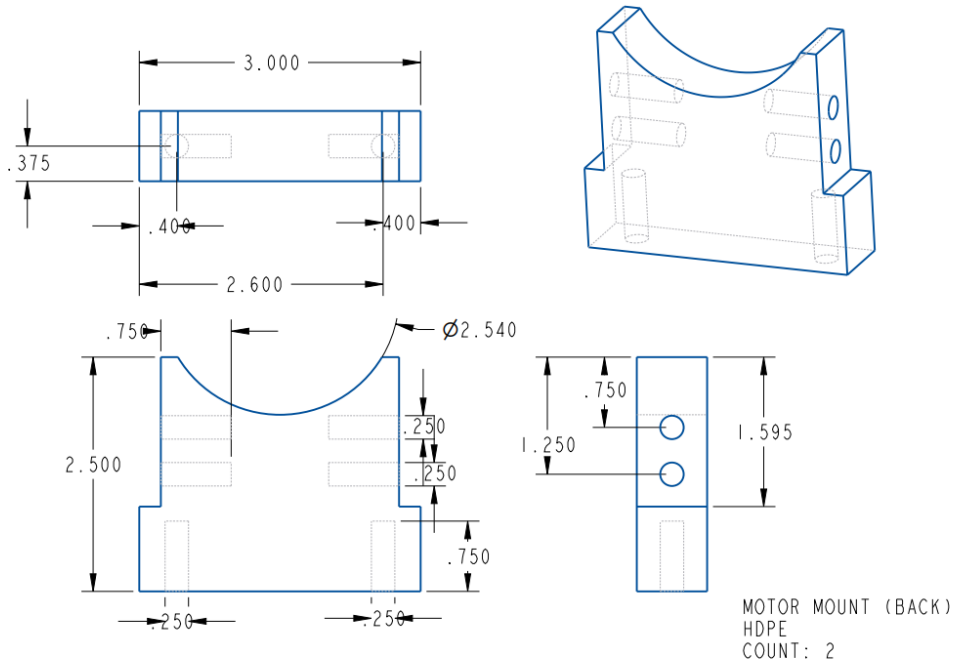
Appendix A: Creo Drawings of Specific Parts



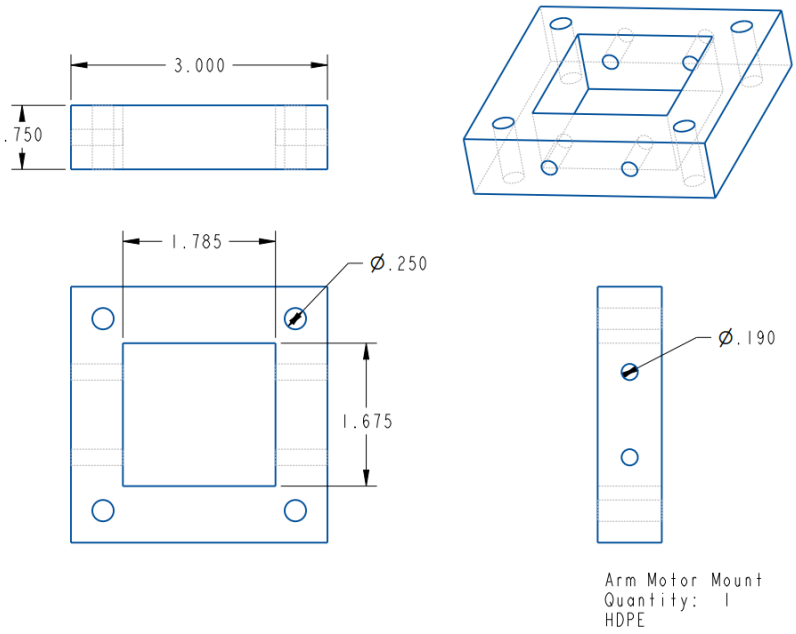
Creo Drawing of Tensioner



(Front) Motor Mount for Drive System



(Back) Motor Mount for Drive System



Arm Motor Mount

Appendix B: SaRR Control Code

```
 * ****************************************************************
 * *Version is marked by milestone reached by code. m0 = driving.
 * Adapted from:
  Complete RC Control with..
  Autonomous guidance with simple if statements and variable speeds
  3/14/17 by Brian Patton
\* ****************************************************************** */
#include <Servo.h>
#include <Time.h>

//*************************************************************
//*************  Variable Declaration/Definition  **************
//*************************************************************
/* NOTE: check right stick pins and channels */
// These macros can be used as "substitutes." For example:
// later on, where ever you see "RServoPin", treat it as a "2"
#define RServoPin 3
#define LServoPin 2
#define ArmPin 4
#define AutoPin 8
#define RSYPin 11
#define RSXPin 12
#define LSYPin 10
#define LSXPin 9
#define Ch6Pin 7
// arm limit switch pins, hook to signal and ground
#define LimitSwitchPin 31
// prox sensor pins (aka "sharp sensor" or "IR Range sensor")
#define BackProxPin 18
#define FrontProxPin 17
// pins for the photosensors
#define LPhotoPin 16
#define RPhotoPin 15
// pin for the medkit servo
#define medPin 14

// more macros. If you see "RSX" it becomes "Ch1"
#define RSX Ch1
#define RSY Ch2
#define LSX Ch4
#define LSY Ch3

// macros for sending signals. He had these at 1350-1650, but the
// normal 1000-2000 works
#define MAX_SIG 2000
#define MIN_SIG 1000

// Create Variables to hold the Receiver signals. Each
// starts out with the pin number that its hooked up to
int RSX; // right stick x axis
int RSY; // right stick y axis
```

```
int LSX; // left stick x axis
int LSY; // left stick y axis
int Ch5; // Auto Control Switch
int Ch6; // "VRB" knob
const int LED = 13;       // Onboard LED pin

// time between wheel speed updates
const double cylcedt = 50e-3; // ms*10^-3 = s, must be < ~140 ms
// max torque for the wheels to not slip (Nm)
const double Tmax = 0.239084 + 0.596242; // n/s^2
// moment of inertia of wheels
const double Iwheel = 0.00101173; // kg*m^2
// maximum estimated angular velocity
const double wmax = 4.6/(4.0/12.0); //4.36512/(4.0/12.0); // ft/s * (in/12) = 1/s
// max angular velocity change per cycle (mapped to signal value);
const double dvmax = map(Tmax/Iwheel*cylcedt, 0, wmax, 1500, 2000); // n/s
// "integrator" k value
const double k = 1; // if its sluggish, increase this

// holds right and left wheel speeds (actual)
int Rwheel = 1500, Lwheel = 1500;
// holds the target speeds for the motors
int Rtarg = 1500, Ltarg = 1500;
// holds signal sent to motors, outside for diagnostic purposes
int rSig = 1500, lSig = 1500;

// light sensor values
int lPhotoVal;            // Variable to store L photoresistor value
int rPhotoVal;            // Variable to store R photoresistor value

// prox sensor values
int backProxVal;
int frontProxVal;

// max distance from medkit drop-off before dropping off
const int dropoffDist = 200;
// tells if medkit door is closed
bool isClosed = true;

// Create Servo Objects as defined in the Servo.h files
Servo L_Motor;  // Servo DC Motor Driver (Designed for RC cars)
Servo R_Motor;  // Servo DC Motor Driver (Designed for RC cars)
Servo Arm;      // Arm servo
Servo medServo; // Medkit door servo

// for auto mode: tells if moved LSY down before trying to go
bool ready = false;
// for auto mode: tells if moved LSY back up
bool go = false;

// for auto wall climb: see function for use
int wallClimbMode = 0;
// for auto wall climb: marks the time the arm has been moving
double armTime;
```

```
// for use if I begin to track the time
double t = 0;
// for printing diagnostic data without interrupting control motion
double diagT = 0;

//*************************************************************
//*************************  Macros  **************************
//*************************************************************
/**
 * diagnostic mode stuff. If don't want it, comment
 * "#define ______" lines out. Its best if you only
 * have one uncommented at a time
 */
#define DELAY 1
// DIAGNOSTIC: print values for RC Mode
#ifndef DIAG_RC
    //#define DIAG_RC
#endif
// DIAGNOSTIC: print how much time each loop takes
#ifndef DIAG_B
    //#define DIAG_B
#endif
// DIAGNOSTIC: print the control parameters
#ifndef DIAG_C
    /**
    #define DIAG_C
    #undef DELAY
    #define DELAY 0.1
    /**/
#endif
// DIAGNOSTIC: print values of sensors
#ifndef DIAG_S
    /**/
    #define DIAG_S
    #undef DELAY
    #define DELAY 0.1
    /**/
#endif


//*************************************************************
//*****************  Setup & Timing  **************************
//*************************************************************

/**
 * gets the current time in seconds.
 */
double time() {
    return ((double)millis()) / 1000.0;
}

/**
 * setup()
```

```
 */
void setup() {
    // change mode of transmitter pins to input mode
    pinMode(RSXPin, INPUT); // Ch1
    pinMode(RSYPin, INPUT); // Ch2
    pinMode(LSXPin, INPUT); // Ch3
    pinMode(LSYPin, INPUT); // Ch4
    pinMode(AutoPin, INPUT); // Ch5
    pinMode(Ch6Pin, INPUT); // Ch6

    pinMode(LED, OUTPUT); // Onboard LED to output for diagnostics
    pinMode(LimitSwitchPin, INPUT_PULLUP); // top arm limit switch

    // Attach Speed controller that acts like a servo to the board
    R_Motor.attach(RServoPin);
    L_Motor.attach(LServoPin);

    // Attach the arm servo
    Arm.attach(ArmPin);

    // Attach medkit servo
    medServo.attach(medPin);

    //Flash the LED on and Off 10x before entering main loop
    for (int i = 0; i < 10; i++) {
        digitalWrite(LED, HIGH);
        delay(200);
        digitalWrite(LED, LOW);
        delay(200);
    }

    // enable USB reading
    Serial.begin(9600);

    // make sure medkit door is closed
    medClose();

    #ifdef DIAG_RC
    Serial.println("LSX,LSY,RSX,RSY,Ch5,Ch6");
    #endif

    // DIAGNOSTIC: for Serial Plotting controller data
    #ifdef DIAG_C
    Serial.println("rw,Rtarg,rSig,lw,Ltarg,lSig");
    #endif

    // DIAGNOSTIC: for Serial Plotting prox data
    #ifdef DIAG_S
    Serial.println("limt,LPhoto,RPhoto,FProx,BProx");
    #endif
}

/**
 * Main Loop
```

```
 */
void loop() {
    // gets the current time (used in several spots later)
    double t2 = time();

    // check if mode swap
    double oCh5 = Ch5; // old value

    // read from photo sensors
    // NOTE: darker is higher
    rPhotoVal = analogRead(RPhotoPin);
    lPhotoVal = analogRead(LPhotoPin);
    // read from prox sensors
    backProxVal = proxDist(BackProxPin);
    frontProxVal = proxDist(FrontProxPin);

    // check if should be in auto mode
    Ch5 = pulseIn(AutoPin, HIGH, 21000);
    Ch6 = pulseIn(Ch6Pin, HIGH, 21000);
    if (Ch5 > 1600) { // auto mode
        // check for mode swap
        if (oCh5 <= 1600) { resetMotion(); }

        // use this to tell it to progress
        LSY = pulseIn(LSYPin, HIGH, 21000);

        // make it so it will only run if you pull LSY down
        if (!ready && LSY < 1700) {
            ready = true;
        } if (ready && LSY > 1300) {
            go = true;
        }

        // check which auto mode it should be in
        if (Ch6 > 1500) {// climb the wall vt
            digitalWrite(LED, HIGH);

            // climb the wall
            if (ready && go) { climbWall(); }
            else { resetMotion(); }
        } else { // deliver the medkit
            // do a "heartbeat blink" to signal the mode
            if ((int)t2 % 2 == 0 && (int)(4*t2) % 2 == 0) {
                digitalWrite(LED, HIGH);
            } else {
                digitalWrite(LED, LOW);
            }

            // deliver the medkit
            if (ready && go) { deliverMedkit(); }
            else { resetMotion(); }
        }
    } else { // RC mode
        // check for mode swap
```

```
    if (oCh5 > 1600) { resetMotion(); }

    // capture from recieve
    RSX = pulseIn(RSXPin, HIGH, 21000);
    RSY = pulseIn(RSYPin, HIGH, 21000);
    LSX = pulseIn(LSXPin, HIGH, 21000);
    LSY = pulseIn(LSYPin, HIGH, 21000);

    /*if (LSX > 1500) {
      medDrop();
    } else {
      medClose();
    }*/

    digitalWrite(LED, LOW);

    // set the motor targets
    setWheelTargets();
}

// only update the motors if enough time has passed
double dt = t2 - t;
if (dt > cylcedt) {
    // update the current time
    t = t2;

    // update the speeds and positions to send to the motors and servos
    updateWheelSpeeds();
}

// cause the motors to drive
drive();

// move the arm
moveArm();


double diagT2 = time();
if (diagT2 - diagT > DELAY){
    diagT = diagT2;

    // DIAGNOSTIC: print how much time each loop takes
    #ifdef DIAG_B
    bench("loop() ", dt);
    #endif

    // DIAGNOSTIC: print values for RC Mode
    #ifdef DIAG_RC
    PrintRC();
    #endif

    // DIAGNOSTIC: print values of sensors
    #ifdef DIAG_S
    printSensors();
```

```
        #endif

        #ifdef DIAG_C
        printCtrl();
        #endif
    }
}

/**
 * Resets the robot
 */
void resetMotion() {
    Rwheel = 0;
    Lwheel = 0;
    LSY = 0;

    ready = false;
    go = false;

    wallClimbMode = 1;
}


//*************************************************************
//**********************  Drive System  ***********************
//*************************************************************
/**
 * Sets the target speed for the wheels by reading data
 * sent from the controller.
 */
void setWheelTargets() {
    int RSY_mod = RSY;
    // fix wierd drive issue
    int RSX_mod = map(RSX, 1000, 2000, 2000, 1000);

    Ltarg = RSY_mod + RSX_mod - 1500;
    Rtarg = RSY_mod - RSX_mod + 1500;

    // make sure targets are between 1000 and 2000
    Ltarg = constrain(Ltarg, 1000, 2000);
    Rtarg = constrain(Rtarg, 1000, 2000);
}

/**
 * Update Rwheel and Lwheel so that they are closer to Rtarg and
 * Ltarg. This should have a delay between calls so the wheels don't
 * slip.
 */
void updateWheelSpeeds() {
    // Right//////////////////////////////////////////
    // find where to cap Rwheel value
    int upper = 2000;
    int lower = 1000;
    if (Rwheel <= Rtarg) { // increasing to Rtarg
```

```
        upper = Rtarg;
    } if (Rwheel >= Rtarg) { // decreasing to Rtarg
        lower = Rtarg;
    }

    // update Rwheel
    int dRW = (Rtarg - Rwheel) * k;
    Rwheel += constrain(dRW, -dvmax, dvmax);
    Rwheel = constrain(Rwheel, lower, upper);

    // Left//////////////////////////////////////////
    // find where to cap Lwheel value
    upper = 2000;
    lower = 1000;
    if (Lwheel <= Ltarg) { // increasing to Rtarg
        upper = Ltarg;
    } if (Lwheel >= Ltarg) { // decreasing to Rtarg
        lower = Ltarg;
    }

    // update Lwheel
    int dLW = (Ltarg - Lwheel) * k;
    Lwheel += constrain(dLW, -dvmax, dvmax);
    Lwheel = constrain(Lwheel, lower, upper);
}


/**
 * Causes the motor to drive.
 *
 * Technically, this updates the current speed of the motor in
 * a way that keeps the robot from sliding.
 */
void drive() {
    // convert the wheel speeds to signals to send to the motors
    lSig = map(Lwheel, 1000, 2000, MAX_SIG, MIN_SIG);
    rSig = map(Rwheel, 1000, 2000, MIN_SIG, MAX_SIG);

    // send the signals to the motors
    R_Motor.writeMicroseconds(rSig);
    L_Motor.writeMicroseconds(lSig);
}


//*************************************************************
//**********************  Arm System  ************************
//*************************************************************
/**
 * Sets the speed of the arm (with LSY > 1500 being up)
 */
void moveArm() {
    // convert the right stick signal value to an arm servo signal
    LSY = constrain(LSY, 1000, 2000);
    int armSig = map(LSY, 1000, 2000, MIN_SIG, MAX_SIG);
```

```
    /* Dead zone *
    if (LSY > 1500) {

    } else {

    }
    /* end Dead Zone*/

    // stop arm if either limit switch is hit
    if (digitalRead(LimitSwitchPin) == LOW && LSY > 1500) {
        armSig = 1500;
    }

    // send the signal to the servo
    Arm.writeMicroseconds(armSig);
}

/**
 * Used to move arm in auto mode, since LSY may be used in other
 * cases.
 */
void moveArmAuto(int signal) {
    int LSY_mod = LSY;
    LSY = signal;
    moveArm();
    LSY = LSY_mod;
}

//*****************************************************************
//************************  Sensors  *************************
//*****************************************************************

/**
 * NOTE: analogRead reads a value from 0 to 1023.
 */

// converts voltage to the corresponding analog read value
// (0-1023)
int adc(double volts) {
    return (int) map(volts, 0.0, 3.3, 0, 1023);
}

/**
 * Prox Sensor: measures 10-80 cm and sends data as an analog
 * value (See Digikey part # 1855-1062-ND)
 */
// get distance from a prox sensor and returns it as an int from
// 0 to 5115 (1023*5). See Data curves for exact values.
int proxDist(int proxPin) {
    int pv = 0;
    for (int i = 0; i < 5; i++) {
        pv += analogRead(proxPin);
    }
```

```
    return pv;
}



//*************************************************************
//************************  Auto Mode  ************************
//*************************************************************
/*** NOTE: need auto mode to only set the target speeds/positions
 * for the servos and motors.*/
/** * NOTE: want it to perform so it stops one set of wheels at 20
 * degrees. */

// WALL CLIMB
// climbs the wall
void climbWall() {
    /*** NOTE: step 5: if the arm folds in before it tips, use time (and hope)*/
    // check current task based on wall climb mode
    switch (wallClimbMode) {
    case 0: // make sure at the wall
        if (frontProxVal >= 1023) {
            wallClimbMode = 1;
        } else {
            // drive forward
            RSY = 2000;
            RSX = 1500;
            LSY = 2000;
            // use arm for small push if need be (just pulse for a bit then go back)
            break;
        }
    case 1: // initial climb up 1st step
        // check if done climbing
        if (backProxVal >= adc(0.9)) { // found distance of ~12"
            // done climbing
            wallClimbMode = 2;
        } else {
            // drive forward
            RSY = 2000;
            RSX = 1500;
            //LSY = 1000;
            // use arm for small push if need be (just pulse for a bit then go back)
            break;
        }
    case 2: // driving to position for arm
        // check if done driving
        if (frontProxVal >= adc(1.65)) { // found distance of ~6"
            // done driving
            wallClimbMode = 3;
            armTime = t;
        } else {
            // drive forward
            RSY = 1700;
            RSX = 1500;
```

```
            //LSY = 2000;
            break;
        }
        // NOTE: if want to check climb fail, do it as first if
    case 3: // arm latch
        if (t - armTime > 3.0) { // at correct angle
            // done driving
            wallClimbMode = 4;
        } else {
            // move arm into position
            moveArmAuto(1300);
            break;
        }
    case 4: // final climb
        // check if cleared the first step
        if (frontProxVal >= 1000) { // cleared
            // done driving
            wallClimbMode = 5;
        } else {
            // move arm and drive forward
            moveArmAuto(1000);
            RSY = 2000;
            RSX = 1500;
            break;
        }
    case 5: // getting over it
        // go until bottom switch hit
        if (backProxVal >= 1000) { // hit
            wallClimbMode = 6;
        } else {
            // move arm and drive forward
            moveArmAuto(1000);
            RSY = 2000;
            RSX = 1500;
            break;
        }
        /** NOTE: watch for flipping **/
    case 6: // land
        break;
    default: // wait
        break;
    }
}

// MEDKIT DELIVERY
// Find the zero direction and then go to the light
void seekLight() {
    // convert them to "distances:" this way, when the sensor
    // is far away (and the intensity is low), the value will be
    // high, and the opposite when the sensor is cloes.
    //int rpv = 1023 - rPhotoVal;
    //int lpv = 1023 - lPhotoVal;
    int rpv = rPhotoVal; // it works opposite of what I expected
    int lpv = lPhotoVal;
```

```
    // Find a good starting point for the light search by
    // finding where the "light distance" is < 700
    if (rpv < 700 && lpv < 700){ // play with 700 value
        // rotate to find when that's not true
        RSY = 1200;
        RSX = 1700;
        setWheelTargets();
    } else {
        // set wheel targets: turn left (decrease Ltarg relative)
        // to Rtarg) if
        Ltarg = map(rpv, 0, 1023, 1500, 1000); // backwards ? 2000 -> 1000
        Rtarg = map(lpv, 0, 1023, 1500, 1000);
        /* "forward bias method:" *
        int bias = map(lpv - rpv, 0, 1023, 200, 0);
        Ltarg = map(rpv - bias, -200, 1023, 1000, 2000); // backwards ? 2000 -> 1000
        Rtarg = map(lpv + bias, -200, 1023, 1000, 2000);
        /* NOTE: Bias' max sets the stopping dist for _Targ:
        // rpv and lpv have to == 200 for no motion, This could
        // be overcome by a prox/light sensor check, but can't
        // be shorter than the distance set by this */

        Ltarg = constrain(Ltarg, 1000, 2000);
        Rtarg = constrain(Rtarg, 1000, 2000);

        // make RSY and RSX follow Ltarg and Rtarg (for monitoring purposes)
        int RSY_mod = (Ltarg + Rtarg)/2;
        int RSX_mod = (Ltarg - Rtarg + 3000)/2;
        RSY = RSY_mod;
        RSX = map(RSX_mod, 1000, 2000, 2000, 1000);
    }
}

// open med doors to drop the medkit
void medDrop() {
    medServo.writeMicroseconds(1000);
    isClosed = false;
}
// close med doors
void medClose() {
    medServo.writeMicroseconds(1700);
    isClosed = true;
}
// returns true if close enough to drop medkit, false otherwise
bool shouldDrop() {
    // NOTE: if isn't far enough down, use light sensors
    double dist = proxDist(BackProxPin);
    return dist < 200;
}
// do medkit process
void deliverMedkit() {
    // check if close enough to drop medkit
    if (shouldDrop()) {
        // wait 3 seconds to make sure stopped
```

```
        delay(3000);

        // drop the medkit
        medDrop();
        return;
    }
    // find the light
    seekLight();
}

//*************************************************************
//**********************  Diagnostics  ************************
//*************************************************************
void printSensors() {
    Serial.printf("%d %d %d %d %d\r\n",
        digitalRead(LimitSwitchPin)*1023,
        lPhotoVal/5, rPhotoVal/5, frontProxVal, backProxVal
    );
}

// print Channel values
void PrintRC() {
    Serial.printf("%d %d %d %d %d %d\r\n", LSX, LSY, RSX, RSY, Ch5, Ch6);
}
// prints the relevant control parameters
void printCtrl(){
  /**/
  int rw = map(Rwheel, 1500, 2000, 1500, 2000);
  int lw = map(Lwheel, 1500, 2000, 1500, 2000);
  Serial.printf("%d %d %d %d %d %d\r\n", rw, Rtarg, rSig, lw, Ltarg, lSig);
  /**/
}
void bench(String name, double t) {
    Serial.printf("%s took %d seconds\n", name.c_str(), t);
}
```