MAE 322

Final Design Report
Spring 2020

Instructors: Dan Nosenchuck, Glenn Northey, Al Gaillard

12 May 2020

Team Name: The QuaranTeam

SaRR Name: Ceci N'est Pas Un Robot

Contributors: Ainil Norazman, Ava Goldinger, Celine Park, Conor Rachlin, Ekin Gurgen, Lucy Norton, Noah Schochet, Peter Fisher

Team Contributions*

Team Member	Roles	Main Contributions to FDR
Ainil Norazman	Programming	Systems Specifications, Autonomous Control Section
Ava Goldinger	Project Management, drawings	Executive summary, Intro, PM section, Drawings, Conclusion
Celine Park	Gear Ratio/Power Budget	Gear Ratio, Power Budget, FBD analysis, Results
Conor Rachlin	CAD, materials selection	Jointed Robot, Robot Geometry, Arm Simulations, Drawing PDFs
Ekin Gurgen	Electronics	Autonomous control section
Lucy Norton	FBD/Creo Simulations	FBDs, Free-body Analysis Section, Simulations, Drawing PDFs
Noah Schochet	Programming	Brainstorming, Results, Conclusion
Peter Fisher	CAD, materials selection, drawings	Specifications section, FBD calculations, Drawings,

*More about each team member's roles on different sub-teams can be seen in A.5

Executive Summary

This report describes the methods and approach to the design of a Search and Rescue Robot (SaRR) that must successfully autonomously complete an obstacle course in the least amount of time possible. The SaRR must retrieve a 3-pound medical kit, breach a 1-foot-tall wall (that has a six inch step), navigate autonomously through a 15-foot chute, and use light sensors to place the medical kit into a 6 inch high walled basket. The design described in this report, referred to as a 'half-track' design, has two large back wheels that are roughly 12 inches in diameter and tank treads in the front, one on each side. These treads only make contact with the ground at one point each while driving, allowing the robot to have better traction while still having an optimized angle for wall breaching. With a current projected weight of 37 pounds and a total length of 31.6 inches, this SaRR's design has been optimized for every aspect of the course.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

2 Introduction

2.1 The Mission

The course that the Search and Rescue Robot (SaRR) must complete has five main tasks, which are shown in Figure 1. The next task is to drive around a pylon and end at the climbing wall fifty feet away. Next, the SaRR must successfully climb the wall. The wall is one foot tall and features a six inch step as well as a bar three feet above the step. The fourth task, the fifteen foot long chute, is supposed to be driven through without touching either of the walls. The final task,

upon coming out of the chute, is to drive to the light source located on the target receptacle and place the med kit into the receptacle.

As shown in the below figure, while some of the course (the first two tasks) can be completed through teleoperation by a team member, the final three tasks must be done autonomously to be truly successful. One of the team's goals for the project is to minimize this "teleop" control of the robot as much as possible.

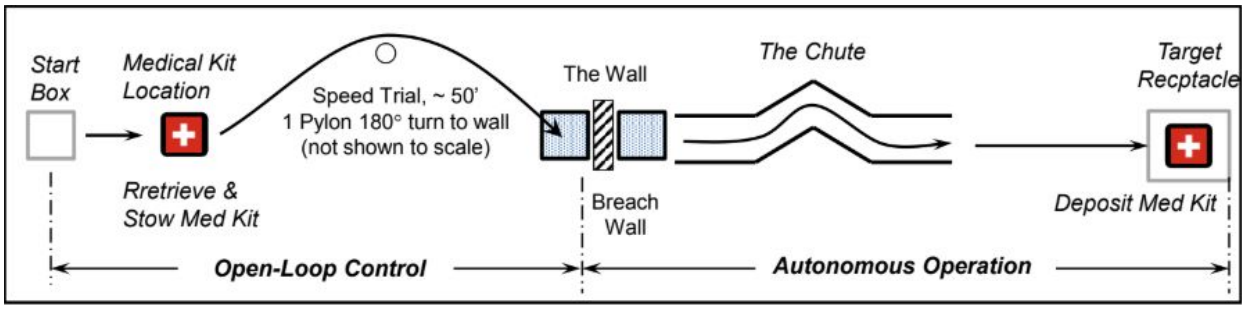


Figure 1: A depiction of the obstacle course that the SaRR needs to traverse

2.2 The Brainstorming Process

Prior to one of the team's early meetings, team members were asked to research existing designs beforehand in order to start off the brainstorming with a set of concepts that have already been proven to work. The three designs whose merits were discussed at length were a centrally-jointed robot with four track segments which could be used like "wheels", a robot with two back wheels and two X shaped paddle wheels in the front, and a 'half-track' design that has wheels in the back and tank treads in the front (These can be seen in A.1, A.2 and Figure 2, respectively). The team used a cost-benefit table, scoring the three designs in five categories: cost, reliability, uniqueness in design, feasibility (manufacturing), and time.

1-3 (3= best)	Cost	Reliability	Unique?	Manufacturing Feasibility	Time	Total Score
Jointed	1	1	3	1	1	7
X Paddles	3	2	1	3	3	12
Half Track	2	2	3	2	2	11

Table 1: The cost-benefit analysis table that the team used to rank how they felt the three potential designs met specific and important design criteria.

2.2.1 The Jointed Robot

The jointed SaRR design featured a four-tracked drivetrain with an actuated central joint, likely operated using a pneumatic cylinder. This variable-geometry feature would allow the robot to operate either in a “tiptoe” orientation, driving on its four outer tread pulleys like wheels while elevating the rest of the body (allowing minimal contact area with the ground to improve traction and control on the low-friction floor) or in a traditional “tank” orientation, with the rear tracks flat on the ground and the front tracks raised at an angle, to be used when scaling the wall.

This SaRR design was rejected for several reasons, mostly centering on the design’s excessive complexity. For one, the actuated hinge would have added the necessity of finding space on the chassis for a pneumatic system (or an extra motor and worm screw mechanism, as was also considered). Additionally, the four independent track segments would have required either four separate motors (adding cost and weight) or an exceedingly complex power transmission system able to provide tractive power to both tracks on a side regardless of the robot’s changing orientation. Ultimately it was decided that the novelty of the idea was outweighed by its high cost (in materials and time) and unnecessary difficulty to design and manufacture.

2.2.2 The X Paddle Robot

The X Paddle SaRR design is a rather standard idea that has been used by many teams in the past to complete the mission. It is relatively simple to design and build, has been proven time and time again that it works, and was overall a very safe design choice for the team to pick. In the table above, it has the highest score, meaning that it would be the most efficient on average, based on the team criteria.

This SaRR has two wheels in the back that are used to drive and steer around the course. The X-shaped paddles, in our design, would have been mounted so that they sat slightly above the ground to reduce friction, the front instead being supported by a ball transfer. The idea behind this was to make the robot move more predictably on a single central low-friction contact point, allowing more accurate control.

2.2.3 The Half Track Robot

The half track design, shown below, can be seen as a compromise between the two aforementioned ideas. It is not entirely track-based like the Jointed SaRR, and it has two back drive wheels like the X-Paddle SaRR for improved traction on slippery surfaces. Raising the

back end of the treads up off of the ground gives the SaRR fewer contact points, which allows the robot to have greater traction on the slippery test surface. Additionally, the wheel and track on each side are driven by the same motor, reducing mechanical complexity.

This is the design that the team ultimately decided to pick. Though it has a slightly lower cost-benefit score than the X-Paddle SaRR, the team felt that the chance to work on something interesting, challenging, and original made up for the one point difference. One of the added selling factors was that in the worst-case scenario, if it came down to the wire and the tracks were proving ineffective, the tracks could be removed and the existing frame geometry retrofitted with belt-driven x-paddles at the front of the robot.

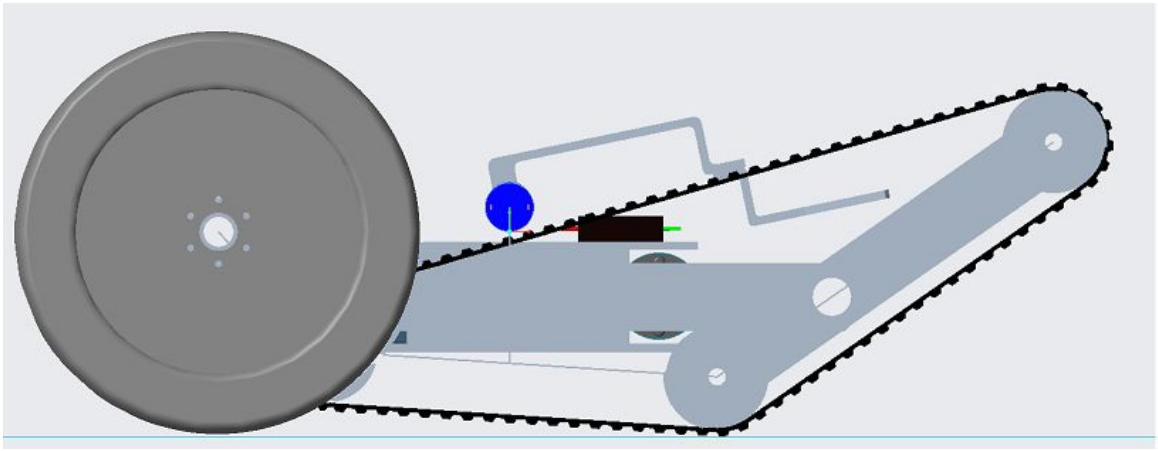


Figure 2: An early version of the half track SaRR design that was chosen by the team.

3 Systems Specifications

3.1 Physical Specifications

The SaRR design is expected to have the following physical specification in order to navigate through the required tasks successfully. Critical overall dimensions are shown in Figures 3 and 4.

Criteria	Dimension	Additional Notes
Bounding Box	18” W x 31.6” L x 12” H 18” W x 31.6” L x 17.3” H	Arm Down position Arm Up position
Wheel base	16” W x 13.7” L	
Front tread angle of Attack	35°	

Total Weight	37 lbf	Includes medkit
Center of Mass Location	0.04” to right of centerline, 1.2” below and 11.8” forward of wheel axle	With med kit positioned forward
	0.04” to right of centerline, 0.76” below and 10.7” forward of wheel axle	With med kit stowed

Table 2: Physical specification based on current design

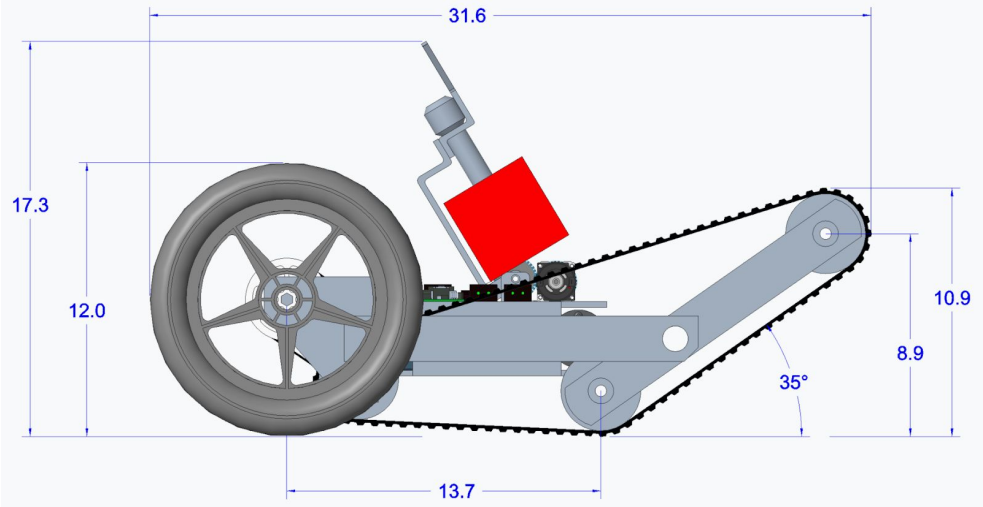


Figure 3: Profile view of the SaRR with critical overall dimensions annotated. Though the design has gained complexity since these figures were made,the critical dimensions shown have remained the same.

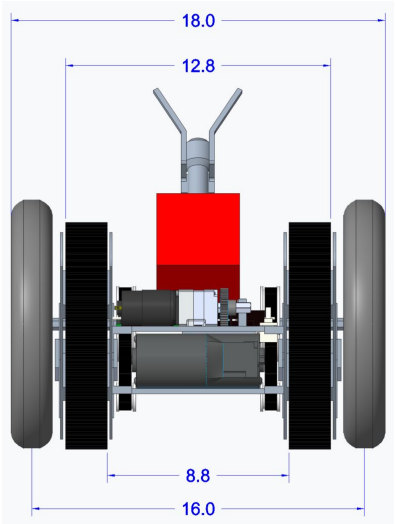


Figure 4: Front view of the SaRR with critical overall dimensions annotated

3.2 Power System Characteristics

Our SaRR will be fully self-powered, using an on-board lithium-ion battery. The following system will deliver power to the SaRR's moving components:

System Component	Specification	Additional Notes
Battery	12 V, 7 Ah capacity, 1 A max current	
Drive train motors (Mini CIM with 16:1 gearbox)	12 VDC, 5840 rpm free speed, 3A free current, 12.4 in-lbs stall torque, 89A stall current	Quantity: 2
Planetary Gear Motor from ServoCity (for arm)	12VDC, 12 rpm free speed, 0.54A free current, 507 in-lbs stall torque, 20A stall current	Supply direct power to arm Quantity: 1

Table 3: Power component specification

3.3 Expected Performance

- Time to lift med kit: approximately 1.74 seconds
- Theoretical top speed: 0.5 m/s
- Theoretical maximum range: 0.7 miles

3.4 Modes of Operation

The performance requirement for the SaRR is a function of multiple successful tasks conducted in five environments. In order to achieve the performance requirement, the SaRR main system is divided into 4 subsystems as described below:

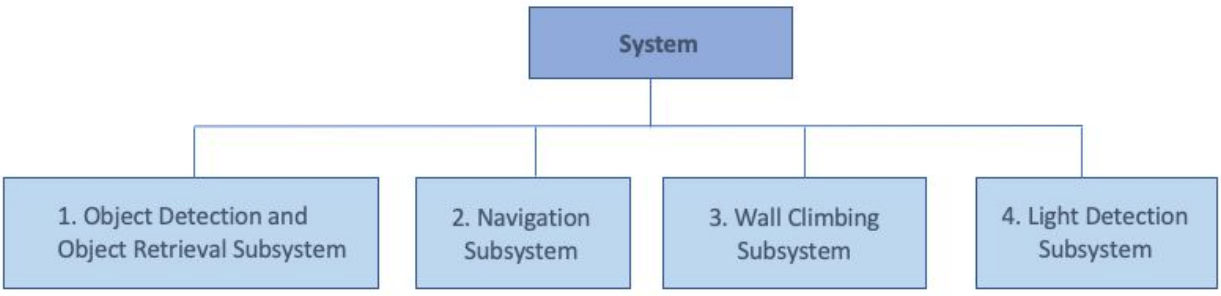


Figure 5: Hierarchy of Robot Subsystems

The robot system may operate in the following modes:

Autonomous Object Detection Mode: This operational mode is used when deploying the subsystem for object detection and object retrieval. The mode is used with the goal of picking up the medkit autonomously using a vision-based object segmentation method.

Autonomous Navigation Mode: This mode is used for all ground navigation. This mode implements a sequence of actions that avoids ground obstacles and automates line following. Autonomous navigation will be implemented for forward driving and driving through chute.

Manual Navigation Mode: This mode is used in the case that the system is unable to continue directed or autonomous driving. Manual navigation is deployed through the use of a wireless remote control operating on transmitted radio signals.

Autonomous Wall Climbing Mode: This mode is deployed when the robot faces a wall obstacle and requires maneuvering over the wall to proceed to the next stage.

Light Detection Mode: This mode is deployed for light detection in the object environment. In this mode, the robot is expected to move towards a light source and place the medkit in designated goal. Robot's dynamics are determined by the robot's photosensor value.

4 Project Management

4.1 Management Responsibilities and Approach

The team leader is in charge of keeping all sub-teams on track with their deadlines, as well as coordinating and running the weekly team meeting. This includes writing the agenda, which is based on the goals and discussions from the previous week. The meeting agenda is sent out ahead of time in case any sub-teams need to add anything they want discussed or troubleshot. When a team member has identified an issue, the team talks through it either in the groupchat (if it needs solved before the team meeting on Monday) or at the Monday team meeting in order to come up with a plan of action that solves it.

Had the manufacturing phase gone into effect, a running document would have been made where those in the machine shop that day summarized what had been started, worked on, and accomplished. This would have been a useful tool to track how on schedule the team actually was, allowing the team leader to make necessary changes or suggestions based on the reality and the goal.

The team leader also keeps track of a Project Management spreadsheet, which has tabs for budget tracking (Figure A.3), the class timeline (Figure A.4), the list of which team members make up each sub-teams and their responsibilities (Figures A.5 and A.6), and both the old and new waterfall schedule charts. As the drawing creation stage of the project went into effect, a

new tab featuring a running list of all parts in the robot was made in order to organize and streamline the drawing process (Figure A.6). All team members have access to the document in order to access both the schedule and the budget, and the team leader checks all items added to the budget with the sub-team that added it to make sure it is correct. The team leader is in charge of making sure the project is in budget, helping sub-teams figure out where to compromise if necessary. They also are in charge of the final Bill of Materials.

4.2 Team Scheduling

The team's original, midterm, and final waterfall schedules (see A.6, A.7, and Figure 15 below) were made in conjunction with the initial schedule sent out at the beginning of the semester. All important due dates are included, and all large project components were scheduled into the semester in order to make the project as stress free as possible. With this base schedule made, any changes to the project from the professor or any delays in the project from the team can be clearly factored in to ensure that the projects continues to run smoothly.

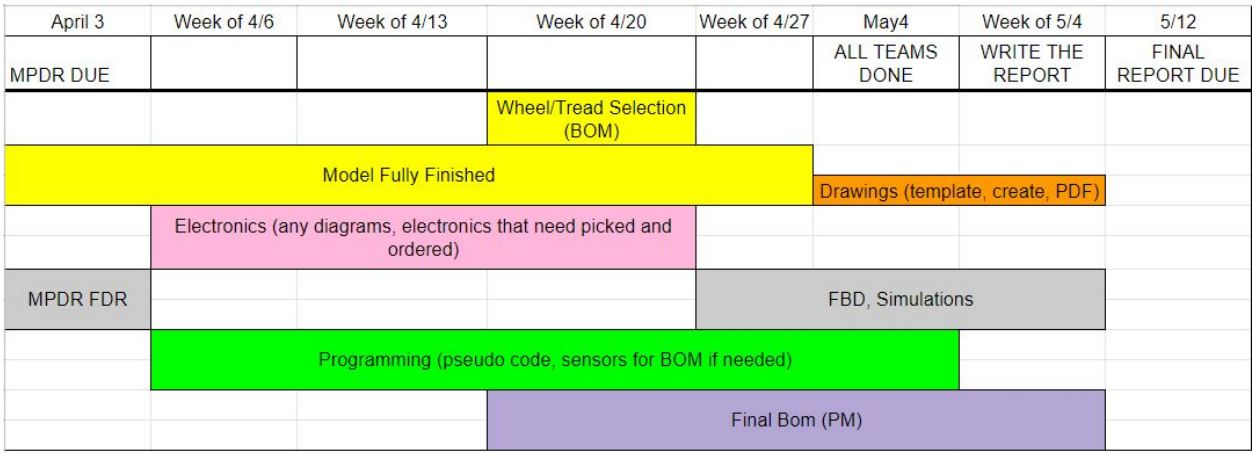


Figure 6: The realistic schedule that was followed the second half of the semester..

4.3 Anticipated Hours

Though this project will no longer be physically built by our team, the anticipated hours for manufacturing, building, and troubleshooting hardware and software have still been included below. So far, most of the focus has been on CAD, light-following programming, and Free Body Diagrams. Now that the team has entered the back half of the semester, focus on all other areas is now beginning to pick up. Almost every sub-team will be working in tandem to ensure that everything is completed in time. This is where point people for sub-teams will become very helpful, so that the team leader can easily collect information on how the tasks are going. The

exact sub-team tasks that fit into the components below can be seen on the sub-team list, Figure A.5.

Task	Anticipated Hours over the Semester	Actual Hours over the Semester
CAD Design	60 hours	120 hours
CAD Simulation	40 hours	40 hours
CAD Drawings	30 hours	40 hours
Material Selection	10 hours	10 hours
Free Body Diagrams	40 hours	50 hours
Team Meetings	120 hours	120 hours
PM Tasks	15 hours	20 hours
Coding Team	15 hours	30 hours
Electronics	10 hours	20 hours
**Manufacturing	25 hours	25 hours

Table 4: This table shows the anticipated and actual number of hours per large task for the successfully completed robot.
** Done pre-midterm, when class was still in person

4.4 Pluses and Deltas

Pluses (+)	**Deltas (Δ)**
- Clear workflow division via subteams, simple task assignments	- Need to implement and enforce standardized naming convention
- Strong team dynamic, fun and motivating culture	- Should create protocol for CAD file organization and version control
- Excellent communication, in person and remote, across 8 members in 4 time zones	- Should transition from weekly to biweekly (2x/week) in weeks leading up to project due date

- Rigorous calculation and calculation checking	- Robot was well meticulously designed, but not the simplest design to meet the given criteria
- Balanced desire to make design innovative with necessity to make SaRR feasible.	
- Exceeded project minimum requirements and effort (i.e. chose to do tracks despite known difficulty, chose to make robot fully autonomous from start to finish)	

Table 5: Pluses and Deltas

5 Design and Analysis

5.1 Robot Geometry

Perhaps the most important factor determining the SaRR's ability to successfully complete the course is its basic geometry, and as such this was the primary design focus early on. Through ongoing free-body diagram analyses, any remaining problems with chassis geometry can be identified and addressed as the project progresses.

The chassis overall was designed to be as narrow and short of wheelbase as possible to improve maneuverability (particularly in the chute), while still having a long enough wheelbase to allow execution of the wall climb and fit all the necessary drivetrain and electrical components. Creo mockups were used to determine chassis geometry suitable for wall traversal, and any problems with the chassis geometry that may be found through free-body analyses will be addressed.

5.1.1 Tread Arm Angle

Using a Creo assembly mockup of the wall-traversal procedure, the best angle for the front tread arms was determined to be 35°. Experimentation with treads on hand in the shop showed that treads engage the wall better at higher angles of attack (although this relation breaks down when approaching vertical), but too high an angle of attack would result in the front treads only being able to engage the wall when initially pulling the robot up the first step, and only spinning in midair thereafter. Additionally, too steep an angle would have resulted in an unfavorable near-vertical angle of attack on the upper step which would be prone to slipping. Too low an angle of attack, however, would have resulted in the forward tread pulleys contacting the flat

side of the upper step while the robot was still trying to climb the lower step, preventing further forward progress. A 35° angle of attack was found to be a good compromise - as high an angle of attack as would still allow good engagement with the wall during all stages of the climb and smooth transitions between different climbing orientations.

5.1.2 Gripper Arm

The decision was made to use a passive gripper to pick up the medkit for reasons of simplicity. A simple bent metal fork with prongs angled outwards allows the grabber to securely retain the medkit using gravity after acquisition and to be self-centering, reducing the need for precision in the pickup process without moving parts. The arm is gear-driven by the arm motor from the sample robot, which was determined adequate for this task. The arm picks up the 3.5 lbf medkit at a distance of roughly 9.5” from its axle, so the requisite torque to lift the medkit at an apparent weight due to acceleration equal to twice its weight is just 66.5 lbf-in, and the arm motor’s stall torque is 506 lbf-in. While this may be overkill, this motor is readily available and therefore doesn’t cost us anything. It is also already geared for low-speed operation (12 RPM), and since the arm’s travel is only about 115° this would be accomplished in 1.6 sec at this no-load speed. Even if the motor were to be bogged down under load to a third its speed, which seems like a large reduction given its high stall torque compared to the torque needed, the lift would still only take 5 seconds. The arm itself is to be made in two pieces: a body machined from flat aluminum stock and a fork bolted to the body which is bent from a flat piece of aluminum cut into a U shape.

5.1.3 The Chassis

The primary design intent behind the robot frame was to keep the parts required as simple as possible to manufacture and assemble, even though the treaded construction required a frame somewhat more complicated than the simple square box of the sample robot. To that end, the frame is entirely symmetric, and the tread frames along each side are themselves symmetric, meaning that almost all the parts of the chassis are duplicate or quadruplicate parts. The side tread frames are made from ⅛” thick, 2” wide aluminum plates, and since the height dimension is noncritical there is no need to machine down the stock from its factory 2” nominal width. These flat plates are connected to each other with bolts and short sections of extruded aluminum U-channel serving as spacers, which are an off-the-shelf component from McMaster.

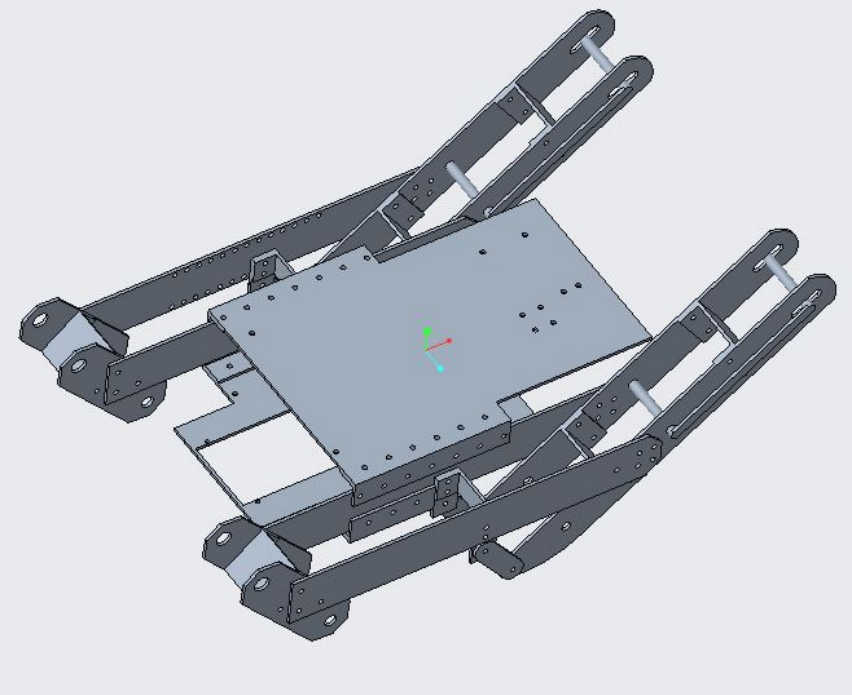


Figure 7. CAD view of the bare chassis

The top and bottom main plates that form the robot's central structure are also made from ⅛" thick plate aluminum, and are each connected to the side tread frames using sections of bolted-on extruded aluminum 90° angle beam - another off-the-shelf component. The cutout visible in the lower plate was made in order to influence the center of mass as well as provide a convenient area to install the master power switch, which is placed at the furthest rear point of the chassis for safety: when approaching the robot from the rear, the power-off switch is the first item in reach and can be reached without putting the hand near large moving parts, and when turning the power on, the operator is behind the robot, well away from moving parts like the treads and arm.

The battery and the two drive motors are positioned between the two chassis plates, while all other electronics and the arm are mounted to the top plate. The underside of the chassis was deliberately left clear in order to eliminate the possibility of anything snagging on the wall steps during the climb. For slight shock absorption and ease of installation, all electronic components mounted on the top chassis plate are to be attached using rubberized double-sided tape. Also in order to deal with shock loading, the bolts used to hold the chassis together ought to be guarded against vibration through the use of nylon lock nuts.

5.1.4 Tread Drivetrain

The tread used is actually a double-sided timing belt sourced from Brecoflex, 50 mm wide with a 20 mm tooth pitch. This is deliberately a rather wide and coarse timing belt: wide in order to better distribute the robot's mass, particularly when focusing the load on very small points

during the wall climb, and coarse in order to better engage the wall. The belt's spine is reinforced with steel cabling for strength. It rides on 6 tread pulleys: 2 driven pulleys and 4 idlers.
The two driven pulleys are located at the rear of the robot in close proximity to the rear drive wheels. This is done because on each side the rear drive wheel and the driven tread pulley are driven by the same motor via a belt drive as shown in the figure below. Because the tread pulley and rear wheel have different diameters but need to operate at equal speed relative to the ground for smooth driving (to avoid skidding of the treads), they are connected to toothed pulleys of different diameter (50:19 ratio) which locks their relative rotation rates. A pair of 3D-printed belt tensioners provide the necessary force to keep the belt both taut and in a shape that does not interfere with the chassis or tread geometry.

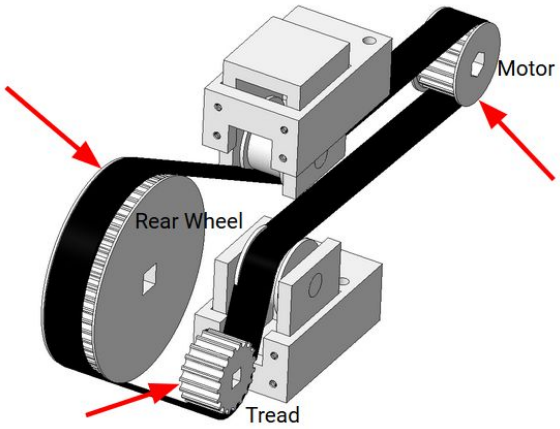


Figure 8. Belt Drive

As is visible in the figure above, the rear wheel shaft and driven pulley shaft engage with the driven wheels and pulleys using a square drive, and are retained to said shafts by cotter pins. As seen in the figure below, all six tread pulleys are 4" in diameter, toothed to engage with the teeth on the inside of the tread, and are to be CNC machined from UHMW plastic. Oilite sleeve bushings are press-fit into the four idler pulleys to provide smooth rotation about the fixed shafts. The shafts on which the driven pulleys are mounted also ride in bushings. Also visible below are the four small tapped holes in the pulley face for the mounting of thin sheets of aluminum to serve as sidewalls, preventing the tread from slipping side to side off the pulleys.

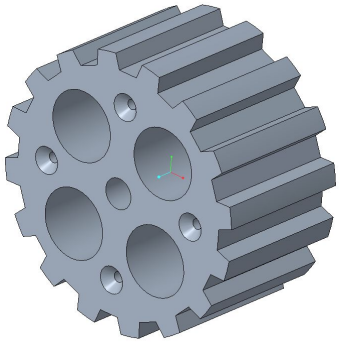


Figure 9. Tread Idler Pulley

Tension in each tread is provided by a pair of turnbuckles mounted to the sides of the tread frame, as seen in the figure below. The upper idler pulleys' shafts are not directly connected to the sides of the tread frame but rather run in a slot in the frame and are connected at each side to a turnbuckle. This arrangement allows tension to be maintained in the driving treads without using a fourth idler pulley to provide tension, which would have added complexity to the frame. Using the turnbuckles creates a tensioning system that fits entirely within the original dimensions of the robot frame, and is notable both for its simplicity and its unobtrusiveness.

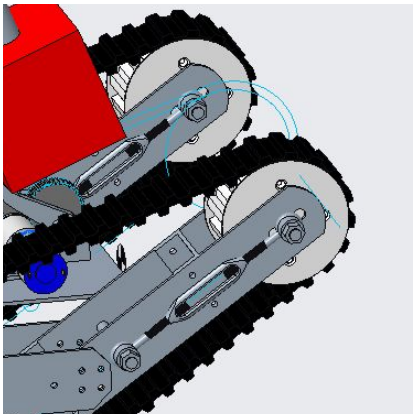


Figure 10. Close-up of turnbuckle tread-tensioning mechanism

5.2 Free-Body Analysis

One of the most valuable tools in the design and analysis process is free-body analysis. This allows for insight into maintaining machine balance, identifying risks of the design, modification of the design through changing parameters, and the realistic capabilities of the design. As such, four free-body diagrams were created at various points throughout the most difficult challenge: the wall-scaling. These diagrams were then analyzed for the forces and moments on the SaRR at its points of contact with the wall and the ground in terms of various parameters. These parameters, as shown in Figures 12, 14, 16, and 18, include various distances in both the SaRR and course coordinate systems, the mass of the SaRR, the accelerations in both the x and y directions, and coefficients of friction for the wheels and the tread. Once equations were determined and solved for, this allows for values for these parameters to be inputted into the equations to determine the forces and moments at these points.

5.2.1 Free-Body Analysis #1

The first analysis completed was that of the SaRR when first reaching the wall. In this diagram, as shown in Figure 11, the middle pulley is just lifted off the ground such that there are no forces at its contact point. It was determined that the conditions at this stage should be the following: the accelerations in the x and y directions should both be positive, as well as the moment about

the point of contact of the wheel with the ground (point A in the diagram). These conditions, as well as the resulting equations that they created, are shown in Figure 12.

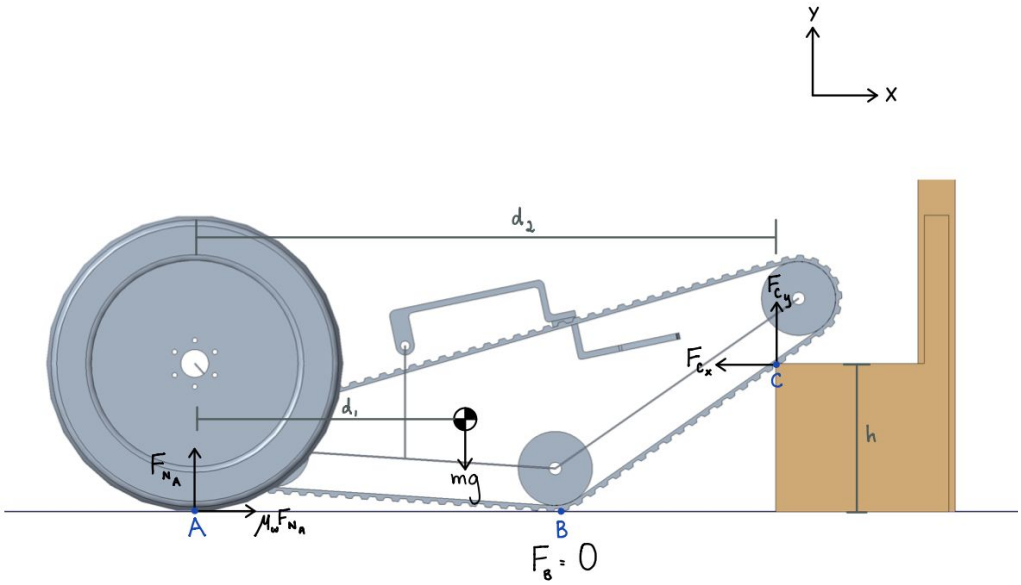


Figure 11: Free-body Diagram #1

$$\Sigma F_x > 0: \quad \mu_w F_{N_A} - F_{C_x} = ma_x,\ a_x > 0$$

$$\Sigma F_y > 0: \quad F_{N_A} + F_{C_y} - mg = ma_y,\ a_y > 0$$

$$\Sigma M_A > 0: \quad F_{C_x} h + F_{C_y} d_2 - mgd_1 = I\alpha,\ \alpha > 0$$

Figure 12: Free-body analysis for Diagram #1

5.2.2 Free-Body Analysis #2

The second analysis completed was that of the SaRR when first reaching the wall. In this diagram, as shown in Figure 13, the middle pulley is just lifted off the ground such that there are no forces at its contact point. The conditions at this stage are the exact same as in the prior analysis: the accelerations in the x and y directions should both be positive, as well as the moment about the point of contact of the wheel with the ground (point A in the diagram). These conditions, as well as the resulting equations that they created, are shown in Figure 14.

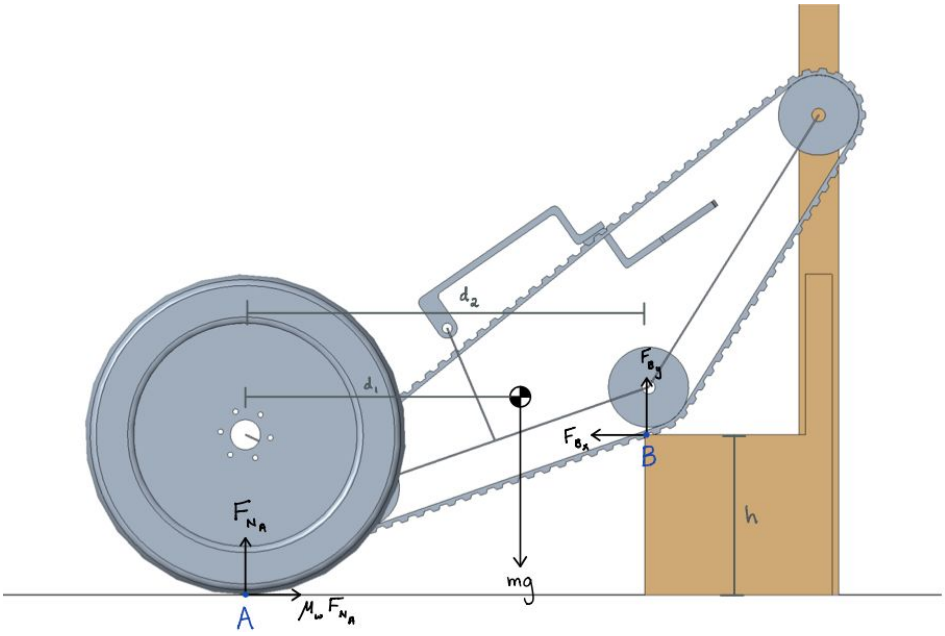


Figure 13: Free-body Diagram #2

$$\Sigma F_x > 0: \quad \mu_w F_{N_A} - F_{B_x} = ma_x,\ a_x > 0$$

$$\Sigma F_y > 0: \quad F_{N_A} + F_{B_y} - mg = ma_y,\ a_y > 0$$

$$\Sigma M_A > 0: \quad F_{B_x} h + F_{B_y} d_2 - mgd_1 = I\alpha,\ \alpha > 0$$

Figure 14: Free-body analysis for Diagram #2

5.2.3 Free-Body Analysis #3

The third analysis completed was that of the SaRR climbing the wall, with the middle pulley having reached the second step. In this diagram, as shown in Figure 15, there is contact between the treads and both steps, and the wheel has just lost contact with the ground. It was determined that the conditions at this stage should be the following: the accelerations in the x and y directions should both be positive, and the moment about the point of contact of the treads with the second step (point C in the diagram) should be negative. These conditions, as well as the resulting equations that they created, are shown in Figure 16.

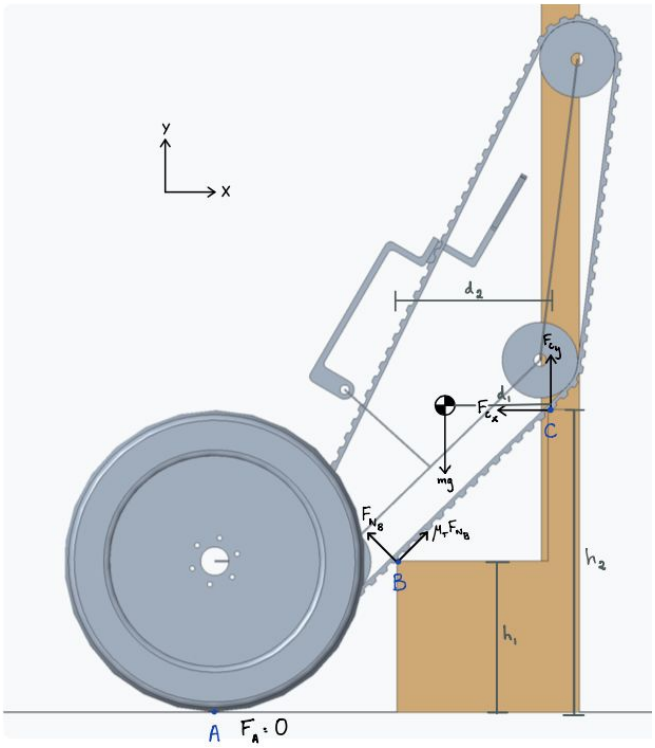


Figure 15: Free-body Diagram #3

$$\Sigma F_x > 0: \quad F_{N_B}(\mu_T \cos\theta - \sin\theta) - F_{C_x} = ma_x,\ a_x > 0$$

$$\Sigma F_y > 0: \quad F_{C_y} + F_{N_B}(\mu_T \sin\theta + \cos\theta) - mg = ma_y,\ a_y > 0$$

$$\Sigma M_A < 0: \quad -F_{N_B}\sqrt{d^2 + (h_2 - h_1)^2} - mgd_1 = I\alpha,\ \alpha < 0$$

Figure 16: Free-body analysis for Diagram #3

5.2.4 Free-Body Analysis #4

The fourth and final analysis completed was that of the stage where the SaRR must topple over the wall. In this diagram, as shown in Figure 17, both pulleys are past the wall and the wheel has just lost contact with the first step. There are two main considerations at this stage: first, that the center of mass of the SaRR must be beyond the wall at this point so that the moment about the point of contact with the second step (point B in the diagram) is in the negative direction. Second, the accelerations in the x and y directions should both continue to be positive. These conditions, as well as the resulting equations that they created, are shown in Figure 18.

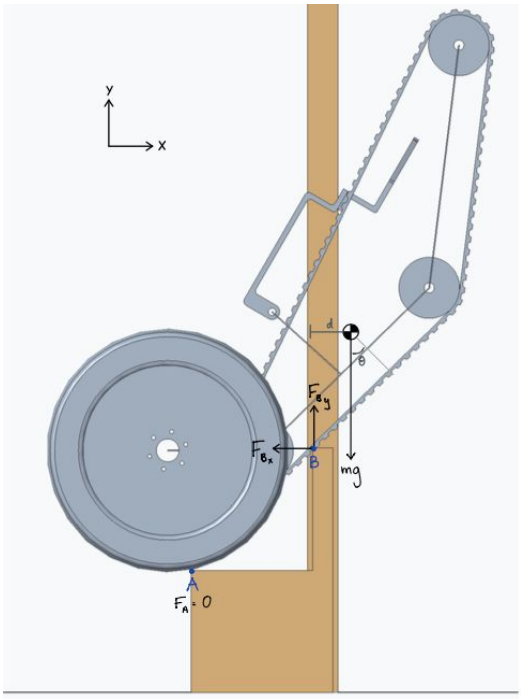


Figure 17: Free-body diagram #4

$$\Sigma F_x > 0: \quad -F_{B_x} = ma_x,\ a_x > 0$$

$$\Sigma F_y > 0: \quad F_{B_y} - mg = ma_y,\ a_y > 0$$

$$\Sigma M_A < 0: \quad -mgd = I\alpha,\ \alpha < 0$$

Figure 18: Free-body analysis for Diagram #4

5.2.5 Tread Deflection

One of the potential issues identified with the use of treads and the accuracy of the FBDs was the potential deflection of the treads in the manner shown in Figure 19 below.

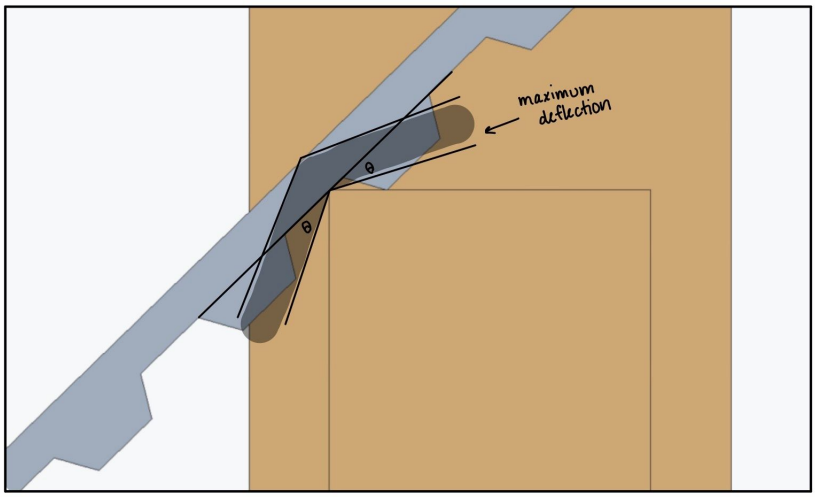


Figure 19: Diagram representing the potential tread deflection during the wall climb with unknown maximum deflection angle (not to scale)

As a result, the maximum tread deflection angle for any point on the treads was calculated by assuming a maximum load of the weight of the robot with the tread parallel to the ground. This allows for the balance of the force of gravity of the robot with the vertical components of the tread tension, resulting in the following equation:

$$mg = 2 * F_T * sin\,(\theta)$$

Given a mass of 37 lbs and the maximum tread tension of 3.5 kN, θ was found to be equal to a maximum of 1.34 degrees. This is small enough to be negligible in the FBD calculation, and as this is the maximum deflection angle it was determined that tread deflection can be neglected for the calculations.

5.3 Simulations

Another valuable tool of analysis is stress analysis simulations. This allows for testing of various high-stress points on the robot during the course to ensure that the parts of the robot under stress will hold up. The maximum stress calculated by the simulation is compared to the yield strength of the material(s) in question to determine this. As a result, three simulations were run, simulating the following: the drop test, the robot toppling over the wall, and picking up the med kit with the arm.

5.3.1 Drop Test

The load applied to the robot for this one-foot drop test was calculated using the concept of impulse and change in momentum. First, kinematic equations were used to figure out the final velocity of the robot right before it hit the ground after starting at rest from one foot above the ground:

$$v_f^2 = v_i^2 + 2a\Delta h \text{ where } v_i = 0\ m/s;\ \ a = -9.8\ m/s^2;\ \ \Delta h = -1\ ft * \frac{0.3048\ m}{1\ ft} = -0.3048\ m$$

$$v_f = 2.44 \text{ m/s}$$

Then using the impulse equation:

$$\Delta p = F\Delta t = m\Delta v,$$

where F here is the impact force felt by the robot.

Assuming the impact was about 0.1 second,

$F = m\frac{\Delta v}{\Delta t} = (18.14\ kg) * (2.44\ m/s) / (0.1\ s) = 442.7\ N$

This force was applied as a load in CREO Simulate to the bottom surface of the frame that would make contact with the ground during the fall. The result is shown in Figure 20 below.

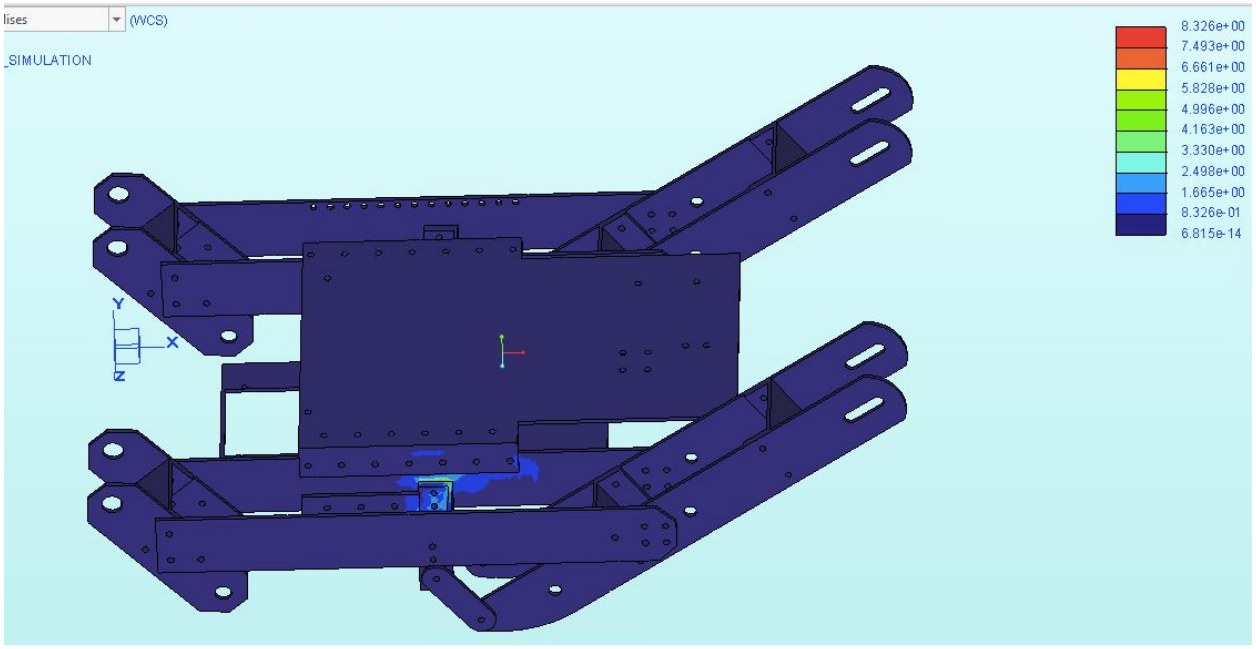


Figure 20: Creo simulation of drop test

As shown on the scale, the maximum stress on the frame for this simulation was 8.3 ksi, well below the yield stress of 40 ksi for Al 6061. A close-up of the place of greatest stress of the simulation is shown in Figure 21.

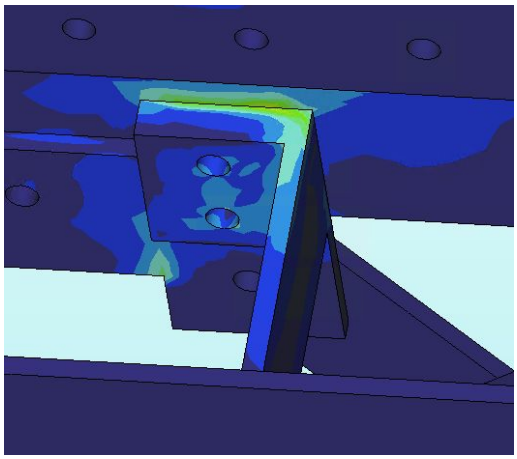


Figure 21: Close-up of Creo drop test simulation at place of greatest stress

5.3.2 Toppling from the Wall

The load conditions applied to the robot for the simulation of the fall after a successful climb of the wall used the same equations as above, just with values changed for the height of the fall. The resulting load, which came out to 768.3 N, was applied to the four points of contact of the front of the tread arms that would make contact with the ground during the fall. The results of the simulation are shown in Figure 22 below.

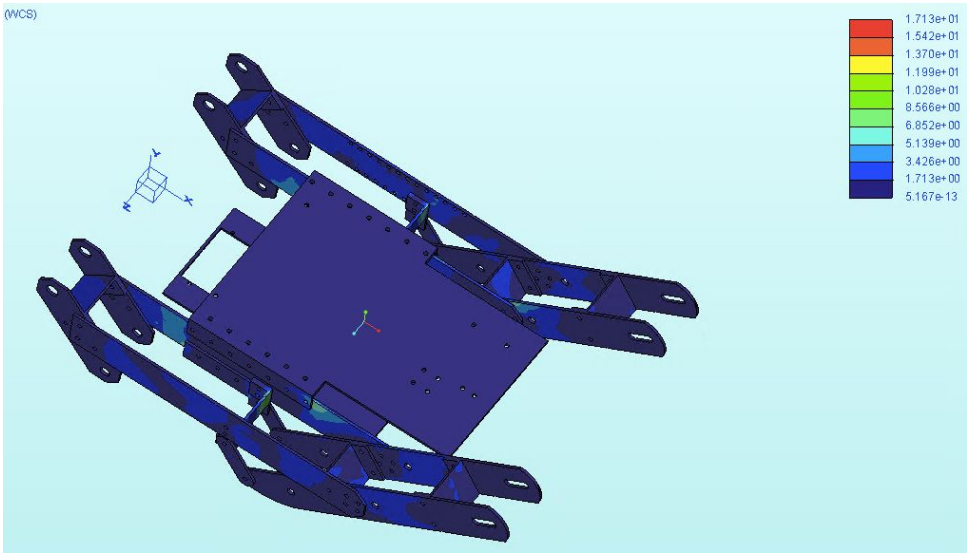


Figure 22: Creo simulation of stress on robot after toppling off the wall

As shown on the scale, the maximum stress on the frame for this simulation was 17 ksi, well below the yield stress of 40 ksi for Al 6061. A close-up of the place of greatest stress of the simulation is shown in Figure 23.

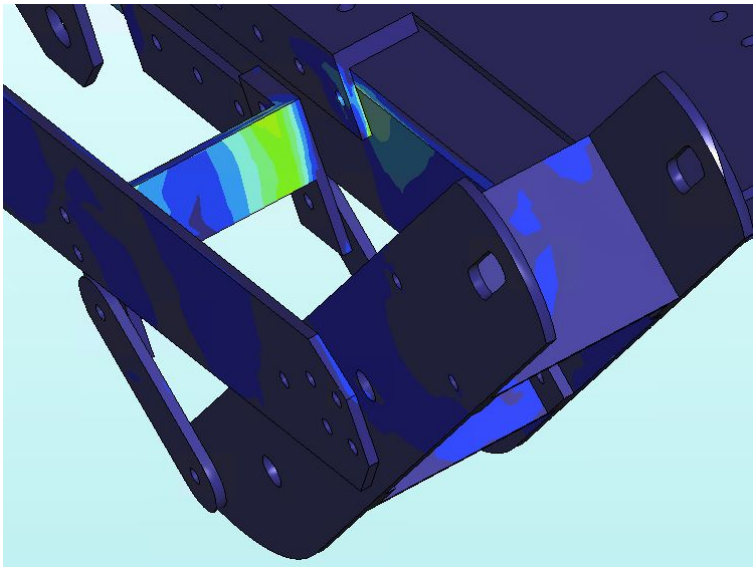


Figure 23: Close-up of Creo simulation of toppling off the wall at place of greatest stress

5.3.3 Lifting the Med Kit

In order to validate the arm design, Creo simulations were run to test its performance under load conditions. In pure lifting mode, the arm proved to be quite sufficiently rigid, reaching only 37% of the material's yield stress even when made to lift ten times the weight of the medkit. Under these extreme conditions the arm experienced a maximum deflection of slightly less than ⅛ of an inch along its length, which although representing an appreciable deformation is well within the elastic regime and once again is reached under extreme conditions: the arm motor is powerful, but not powerful enough to lift the medkit with an acceleration of 10G as was simulated. Under normal use this deflection would be considerably smaller.

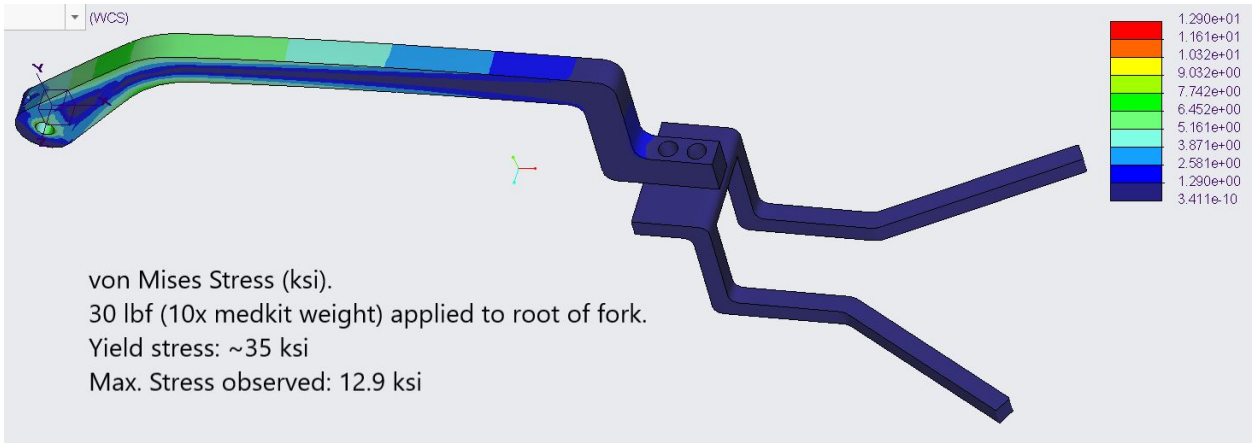


Figure 24: Creo simulation of arm lifting medkit

Another potential mode of failure for the arm is lateral loading, which could be experienced if the hanging medkit were to swing side to side as the robot corners. This too was simulated as shown below, with the medkit applying a force on the fork equal to three times its weight downward and five times its weight to one side. Under these conditions the material reaches 48% of its yield strength, but again these are extreme conditions that are not expected to occur in practice, especially as the robot is not designed for hard cornering. In simulation, the material only got quite close to yielding once subjected to lateral load of 30 lbf: 10x the medkit weight. One might also notice the difference between the two pictures: the fork was strengthened in order to reduce its flexibility and improve its resistance to lateral loading. Under test conditions, which again are exaggerated, the maximum lateral deflection of the arm was about 0.4" at the tip, corresponding to a 2.5° maximum deflection angle along the 9" arm - not huge, within the elastic regime, and experienced under extreme loading conditions, so real deformations would almost certainly be smaller.

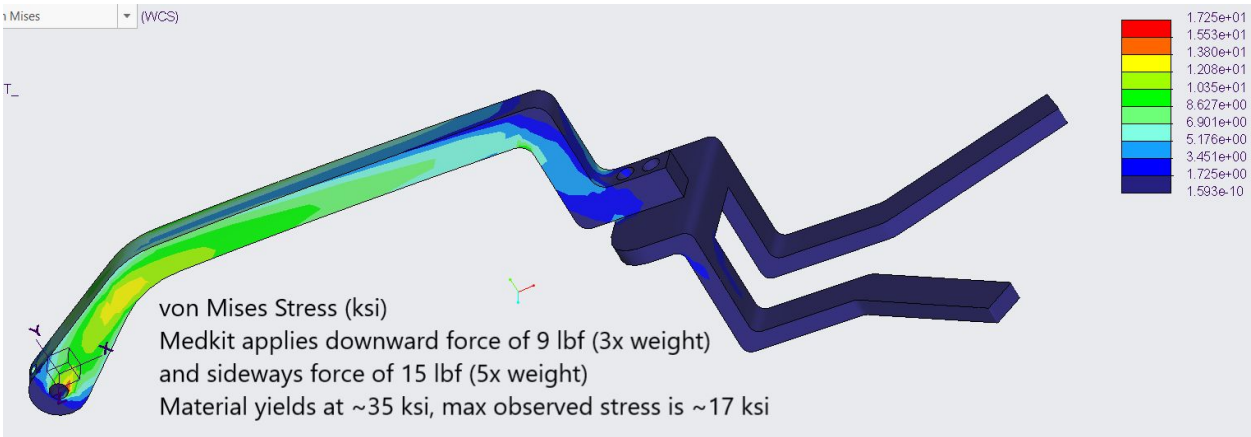


Figure 25: Creo simulation of arm experiencing side load

5.4 Power Budget

5.4.1 Gear Ratio

To optimize for costs, we decided to use the mini CIM motors from the sample robot instead of buying for new ones, which raised the question of what gear ratio to use between the motors and the treads and the wheels. Since there are two treads, weight divided by two is applied to each of them. The shown FBD is the position of the robot right as it loses contact with the ground, which means that the normal force on the wheel is zero. Using this information in a quasi-static analysis, we know that the minimum required torque to lift the robot off the ground and accelerate it forward is $\frac{mg}{2} * sin(\theta)$, which is equal to 56.19 N and much less than the treads'

maximum tension. With the radius of the driving tread pulley that is 0.058m, this corresponds to about 3.3 Nm of required torque. This minimum required torque was too small for us to figure out an appropriate gear ratio that would give us a reasonable speed for the robot. The gear ratio reduction based on working with required minimum torque would result in a speed that is too fast for this task.

The alternative approach we took was using our desired speed to come up with a reasonable gear ratio. We want the robot to be moving at around 0.8 m/s, which equates to 100 rpm. However, this speed would be capped at around 0.5 m/s for the robot due to transmission, thermal and program limits that will be mentioned in the power budget section. Based on the motor curve of the VEX mini CIM motors, a good rpm to get from the motor without drawing too much current and without letting the rpm run free for a capped robot speed of 0.5 m/s seemed to lie around the peak power point at 3000 rpm. This meant a gear ratio of about 50:1, which would give us a robot speed of 0.48 m/s and also torque of 35 Nm, which gives us a safety factor of about 10 compared to the minimum torque required. This gives us a lot more room for transmission and thermal losses while being able to accelerate the robot to our desired speed. This would be achieved by layering a 5:1 gear from the shop with a 10:1 gear set.

For the arm of the robot, the gear motor is connected without further gear reductions. So using its built-in gearbox with ratio 721:1, our robot approximately takes 1.74 seconds to lift the med-kit.

5.4.2 Power Consumption Breakdown

Given a 12V 10Ah battery, we have 120 W-hr of energy from a fully charged battery. This power budget makes sure that the robot will be able to make a complete run on one battery.

In lieu of experimental data, much of power consumption is estimated to be average power based on time estimates of completing tasks. Since we were not able to test out the performance of the robot physically, time estimates observed from previous years' videos and our sample robot's performance prior to the PDR were used.

While the desired speed for the robot was 0.8 m/s, the gear ratio for the mini CIM motors were calculated based on the capped robot speed of 0.5 m/s because of losses due to transmission, thermal and program limits. Most importantly, the theoretical code would limit the speed because it doesn't allow the robot to fully accelerate to its top speed but rather accelerates in bursts, using sensors and conditionals in the code to control the speed.

Based on this cap speed of 0.5 m/s, to take into account the acceleration and deceleration that happens, each portion of the SaRR task was assigned an estimated speed based on how much of the motion involves accelerating/decelerating. For example, during the Speed portion where the robot is driven by a driver on the team, the robot is likely to maintain a higher average speed than when the robot is climbing the wall. Each of these robot speeds were then converted into corresponding power consumption values based on the robot's gear ratio and the information on the motor curve. The radius of the wheel is 6 in. The calculation looked like this:

Robot speed $(m/s) * \frac{60s}{1\ min} * \frac{1\ rotation}{2\pi r} *$ *gear ratio gives us the motor rpm. This motor rpm was then matched with corresponding power value from the motor curve. For example, with the cap speed of 0.3 m/s, this corresponds to the robot's wheels spinning at 37 rpm, which corresponds to the motor running at 1880 rpm. This corresponds to approximately 180 Watts of power consumed by a mini CIM motor.*

The total energy consumed by the mini CIM motors calculated based on this method is shown below. Note that power consumed by sensors and actuators are negligible amounts. The light sensors as well as the proximity sensors draw about 0.025 Watts of power each, which would be about 0.1 Watt and would not affect the 12V battery significantly. The power consumed by the arm motor will be calculated separately below this table.

Task	Task breakdown	Average Speed (m/s)	Corresponding motor speed (rpm)	Drawn current (A)	Power supplied to motor (W)	Time (s)	Energy (W hr)
Med-kit pickup	accelerate to drive to med-kit	0.4	2506.377057	52.237	626.844	15	2.61185
	arm motor to retrieve and store						
Speed segment	accelerate	0.5	3132.971321	43.596	523.152	40	5.8128
	turn corner	0.4	2506.377057	52.237	626.844	10	1.741233333
Breach wall	increase potential energy	0.3	1879.782792	61.741	740.892	15	3.08705
Drive through chute	accelerate through multiple turns	0.25	1566.48566	66.062	792.744	30	6.6062
	wall proximity sensors*						
Locate target; deposit med-kit	accelerate	0.25	1566.48566	66.062	792.744	30	6.6062
	light sensors*						
	arm motor						
						Total	26.46533333

Table 5: The Power Budget for the SaRR

As shown above, each CIM motor will consume about 26.5 Whr, and since we have two of them, they will consume about 53 Whr.

The arm motor retrieves and returns the med-kit once, respectively. The ability of the arm motor from the sample robot has been explained earlier in section 5.1.2 Gripper Arm. Given its specifications, even if its no load speed was lowered by a factor of 3, which would bring it down to 4 rpm, it would be able to lift the med-kit in about 5 seconds. With a gear ratio of 721:1, 4 rpm translates to 2884 rpm motor speed, which would draw about 14 A of current and consume 168 Watts of power. Given this information, the arm motor consumes about 0.23 W-hr each time it retrieves and returns the med-kit. Each happens once, respectively, which means that the arm motor consumes about 0.46 W-hr in total.

The maximum power consumed by the Raspberry Pi board based on its specifications is about 0.53 W-hr.

In total, neglecting the power consumed by the sensors and not considering the independent power consumption from its own battery in Teensy 3.0, the power consumption of the robot comes to 54 W-hr, which is about 45% of the total energy of the battery.

6 Autonomous Control

6.1 Autonomous Tasks and Control Strategies

The table below summarizes the performance requirement for each task according to the given autonomous requirements and elaboration on the team's control strategy.

Tasks	Task Details	Autonomous Requirement	Team Control Strategy
Wall Climbing	Wall dimension is 12” H x 36”	Yes	Control strategy is elaborated in **5.1.1 Wall Climb**
Navigating through long bent chute	Chute dimension is 15” L x 3” W	Yes	Control strategy is elaborated in **5.1.2 Navigation in Chute**
Detecting light source and deploying medkit in basket		Yes	Control strategy is elaborated in **5.1.3 Photosensor Navigation and Medkit Deployment**
Detecting and Picking up Medkit		No	We developed two control strategies for non-autonomous tasks. The strategies are divided into two parts which are manual operation and autonomous operation. Details are elaborated in **5.1.4 Optional autonomous operation**
Navigating around pylon		No	

Table 6: Autonomous Control Tasks and Control Strategies

6.1.1 Wall Climb

The SaRR will slowly approach the wall until the front treads are in contact with the first step. A proximity sensor placed in the front of the robot will be used for measuring the distance. After the initial approach, the robot will drive forward with both its back wheels and its treads moving

at a speed that will later be decided experimentally to enable the SaRR climb over the wall. An accelerometer placed on the medkit arm will be utilized throughout the wall-traversal to measure the SaRR's angle relative to the horizontal. When a negative angle is measured, the treads will stop and the back wheels will continue spinning until the SaRR is back in the horizontal position.

6.1.2 Navigation in the Chute

Proximity sensors that will be placed on the left and right sides of the SaRR will be utilized for autonomous collision avoidance in the narrow chute. A Proportional-Derivative feedback controller will be used and the gains will be manually tuned. The closed-loop control algorithm will continuously read the difference between the two sensor measurements (Eq. 1) and calculate the left and right motor speeds (Eq. 2 and Eq. 3).

$$\Delta_{Prox} = Prox_{Left} - Prox_{Right} \tag{1}$$

$$Motor_{Left} = Const + K_P * \Delta_{Prox} + K_D * \frac{\Delta_{Prox}}{\Delta T} \tag{2}$$

$$Motor_{Right} = Const - K_P * \Delta_{Prox} - K_D * \frac{\Delta_{Prox}}{\Delta T} \tag{3}$$

6.1.3 Photosensor Navigation and Medkit Deployment

Autonomous navigation to the light source will utilize two light sensors that will be placed in the front of the robot with inward angles. An additional proximity sensor that will be placed in the front will be used for stopping the SaRR as it approaches the target. A control method that has been successfully tested on a prototype robot will be modified for the SaRR to complete this task. The following pseudocode describes the control strategy (constant values a, b, c, and d will be determined during sensor calibration) :

```
if proxVal > a:
        Stop
while lPhotoVal > b and rPhotoVal > c:
        Slowly turn left
if rPhotoVal - lPhotoVal > d:
        if rPhotoVal > lPhotoVal:
                Slowly turn left
        else:
                Slowly turn right
else:
        Go forward
```

After the photosensor navigation algorithm terminates, the medkit deployment procedure will begin. An accelerometer will be utilized to bring the medkit arm to the desired angle and the SaRR will move away from the light source.

6.1.4 Optional Autonomous Tasks

In order to complete the performance requirement in full autonomous mode, navigating around the pylon and picking up the medkit tasks can be performed autonomously with object-detection implementation. An RGB camera can be mounted on the robot and vision-based control strategy can be utilized for autonomous control. This SaRR implements on board computer vision -the more robust, yet more challenging method of autonomous navigation. In order to to facilitate the computer vision functionality, the robot will be equipped with a Raspberry Pi computer in addition to the Teensy microcontroller. The Raspberry Pi (AKA RPI) is analogous to the cerebral cortex, responsible for more advanced computation and cognition. In this case, it will run OpenCV using Python. OpenCV is an open source computer vision library that enables key functions such as reading and writing image/video, object and color recognition, and more. The RPI will use OpenCV to intake a variety of data about its surroundings from a simple digital camera module. The RPI then uses this information to make decisions according to programmed instructions. Once the RPI makes a decision, it must issue commands to the Arduino, which then issues commands to the motors and actuators. In theory it is possible to consolidate functionality into just one controller board, but in practice this becomes difficult as neither board is intentionally designed to handle both tasks. The RPI communicates with the Arduino via its GPIO pins or General Purpose Input Output pins. It is limited to only two values, high and low, but this is sufficient for a simple task like steering using only two motors.
To detect the medkit, a color detection method can be used which detects the location of the red medkit and the SaRR can autonomously navigate towards it. Similarly, color detection can be used on the orange pylon to enable the SaRR autonomously navigate towards it. The SaRR can stop at a reasonable distance away from the pylon and perform a hard-coded “going around” motion and continue towards the wall. If color detection methods fail to detect the wall, various other object recognition methods (such as detecting an AprilTag or ArUco sticker on the wall) can be utilized for enabling the SaRR to navigate towards the wall autonomously. Successfully implementing these computer vision methods require an extensive amount of trial and error and manually tuning parameters. An alternative approach is to modify the obstacle course by placing black tape on the white floor that marks a predetermined path for the SaRR and to mount the camera angled down at the floor. The input image from the camera can be fed into a feedback controller that would output motor signals for keeping the black tape in the middle of the camera’s frame. Finally, in order to perform these computer vision tasks, a Raspberry Pi microcomputer can be used. A cheaper but less reliable alternative would be using a wireless FPV camera that communicates with an external computer. The computer can handle the image

processing and communicate with the Teensy via an XBee wireless module. Alternately, line-following can be implemented with a simple infrared LED and IR detector forming a rudimentary radar system.

Algorithm 1 Autonomous Navigation to Wall

```
1: procedure DetectObstacleVision(camera, objectColor, fDistSensor)
2: Initialize Im ← camera.color
3: Apply mask ← cv2.inRange(Im, objectColor)
4: Draw output ← cv2.drawCountours(output, cv2.findContours , (r, g, b))
5: for cnt in find.Contours.... draw
6:     rect ← minimumarea(cnt)
7:     box ← rboxPoint(rect)
8:     if contourArea(cnt) > maxArea, boundBox = box
9: if (boundbox.size > 0.5)
10:    Compute distance ← distanceto(boundBox)
11:    Check checkSensors()
12:    while fDistsensor < 500
13:            forward(distance , 2)
14: else turnleft(30, 2)
15: end procedure
```

Algorithm 2 Med Kit Retrieval

```
1: procedure MedkitRetrieval(camera, robot_position, tag_id, fDistSensor)
2: Initialize Im ← camera.color
3: Search detector ← tagDetector(Im, tag_id)
4: Compute (x,y,z) ← position3D(detector)
5: Navigate robotToKit(robot_position, (x,y,z))
6: if (bumpSwitch())
7:    lowerArmRoutine()
8:    gripOnRoutine()
9:    gripstoreRoutine()
10: end procedure
```

Figure 26: Pseudocode for optional autonomous navigation tasks

6.2 Sensors

Performing the 3 required autonomous tasks (see section 5.1) will utilize 3 proximity sensors, 2 photosensors, and 1 accelerometer.

6.2.1 Proximity Sensors

Proximity sensors consist of an infrared light source and an infrared detector. The source emits pulses of infrared waves that are reflected off an object and captured by the detector. Using the elapsed time between emission and detection of the infrared light, the sensor estimates the distance to the object. Two proximity sensors placed on the sides of the SaRR will be used for navigation in the chute, and an additional sensor placed in the front will be used for measuring the distance from the wall and the light source.

6.2.2 Photosensors

Photosensors consist of photocells that can change resistance depending on the amount of light incident on them. By measuring the change in resistance across the sensor, photosensors can generate an output signal that indicates the intensity of light. In order to prevent the angular ambiguity from a single sensor, the SaRR will use two photosensors that are placed with inward angles. As explained in the pseudocode in section 5.1.3, the differential sensor signal is provided to the control algorithm as the error input.

6.2.3 Accelerometer

A 3-axis accelerometer measures a body's accelerations in all three directions (x, y, z) in its own rest frame. These measurements can be used to calculate the robot's tilt angle in all three planes. A 3-axis accelerometer will be placed on the medkit arm to measure the SaRR's angle as it is landing back on the ground after the wall traversal (see section 5.1.1). Similarly, this accelerometer will provide the necessary angle measurement for lowering and retrieving the arm for picking up and deploying the medkit (see section 5.1.3). Therefore, for our purposes, we are interested in sensing the inclination angle in the xy-plane. Equation 4 below, where Θ denotes the inclination angle and a_i denotes the measured acceleration in the i-direction, will be used to compute the angle:

$$\theta = arctan\left(\frac{a_X}{\sqrt{a_Y^2 + a_Z^2}}\right) \quad (4)$$

6.3 Electronics

The required autonomous tasks will be performed utilizing a Teensy microcontroller. Figure 14 below illustrates the wiring schematic of the SaRR. Components drawn with dashed lines are only required for the optional autonomous tasks.

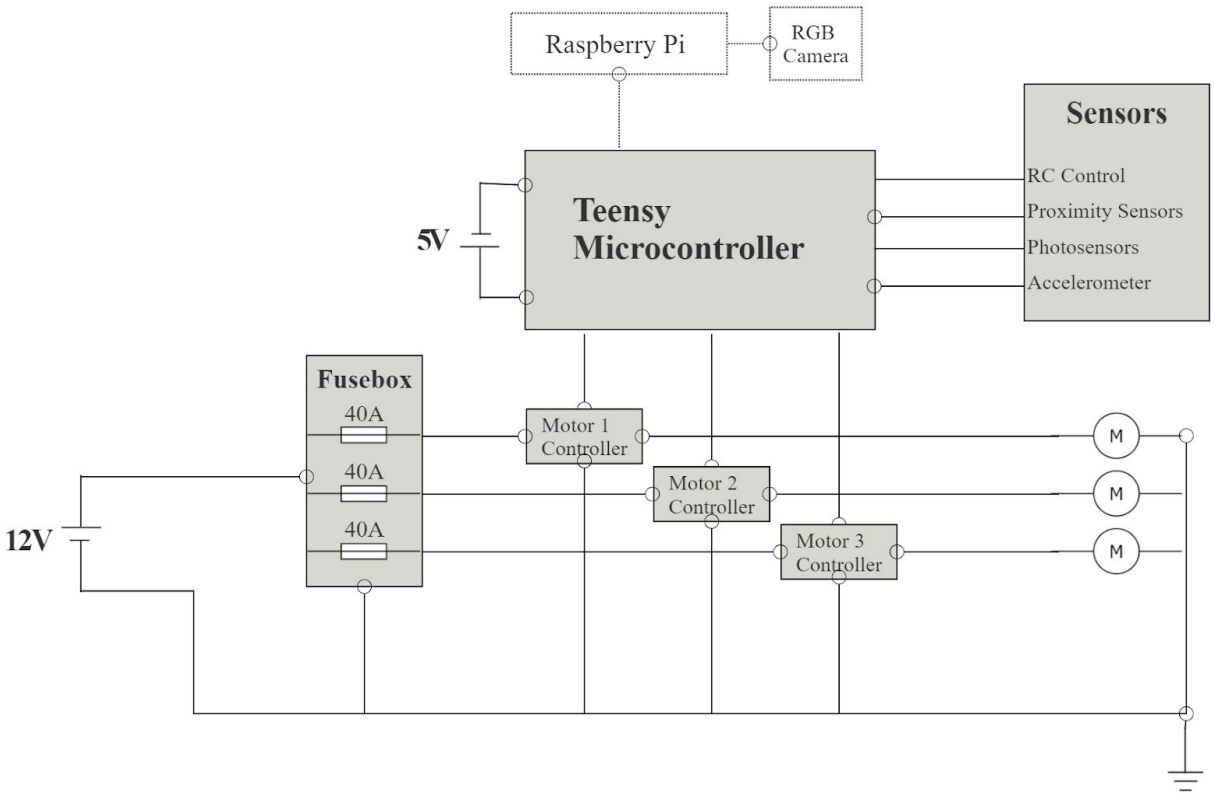


Figure 27: Wiring Schematic of the SaRR

6.4 Programming

All of the required modes of operation (see section 3.4) will be coded in a modular manner to enable collaboration among team members and to increase the reusability of the code. These required autonomous functions will be implemented in C language using the Arduino IDE. As explained earlier in section 5.1.4, a Raspberry Pi microcomputer or an external computer that can communicate with the Teensy wirelessly can be used for performing the vision computations that are needed for the optional autonomous tasks. Python language and its OpenCV library can be utilized for performing these computations.

7. Key Drawings

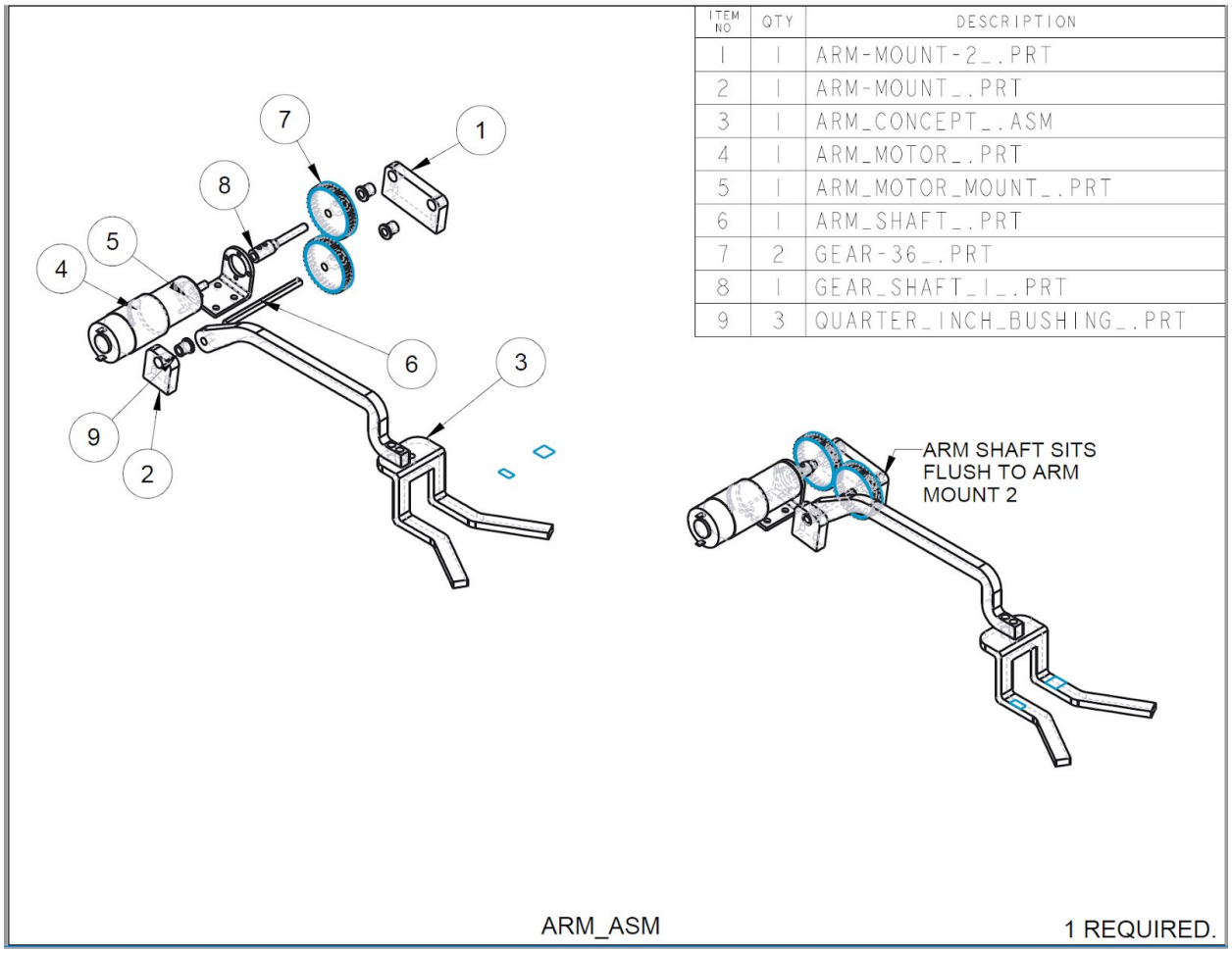


Figure 28: The arm assembly of the SaRR.

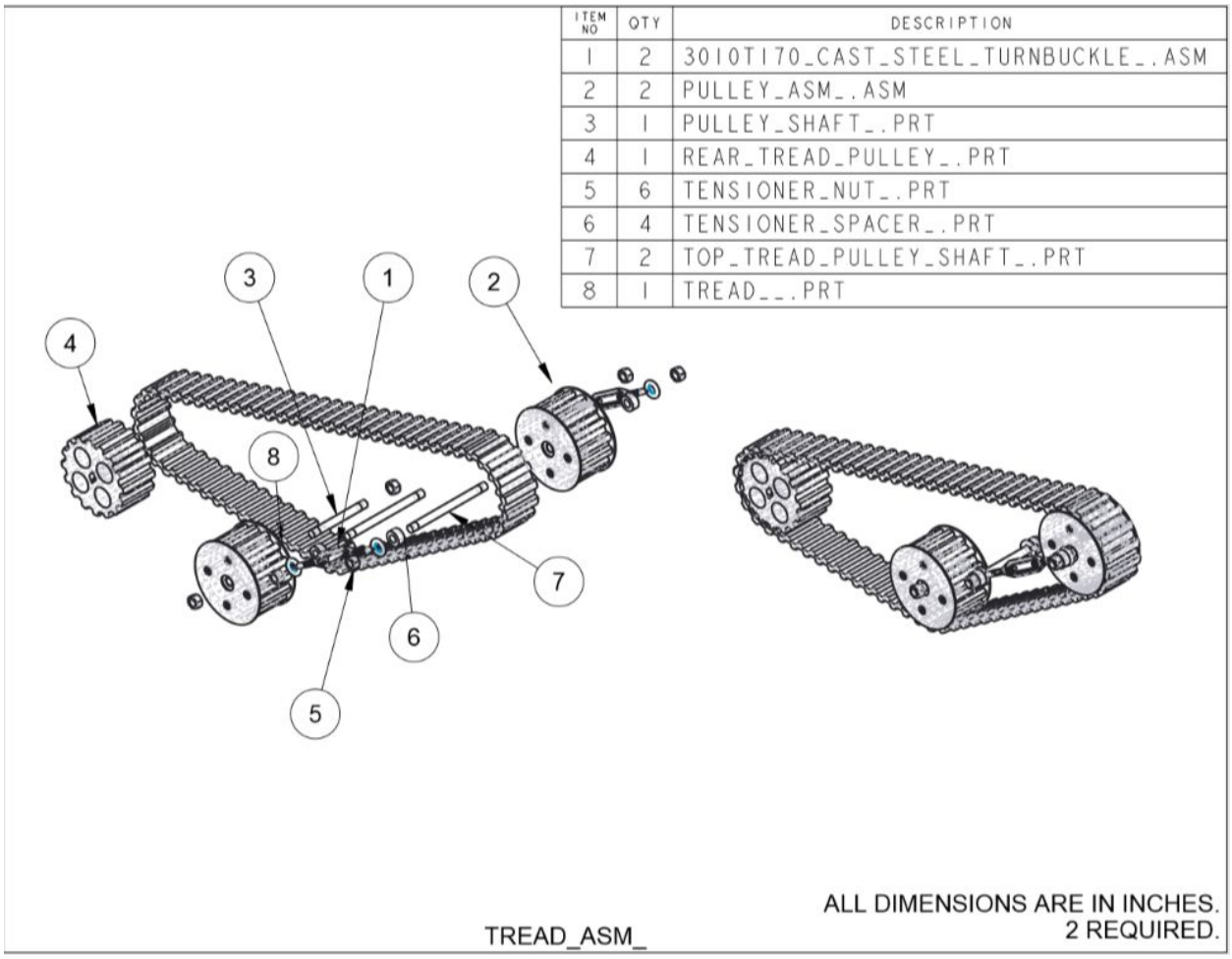


Figure 29: The tread assembly of the SaRR.

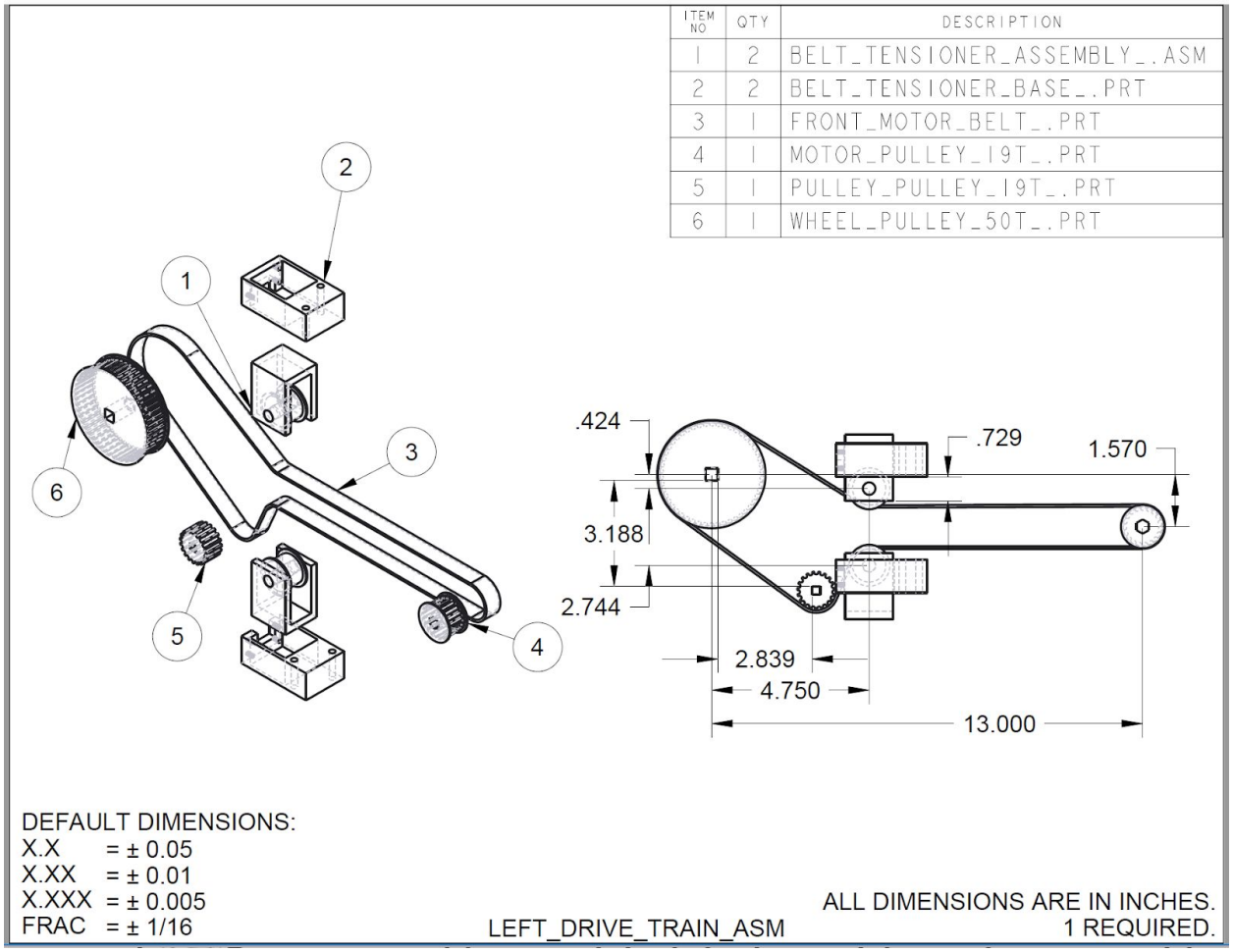


Figure 30: The left drive train of the SaRR.

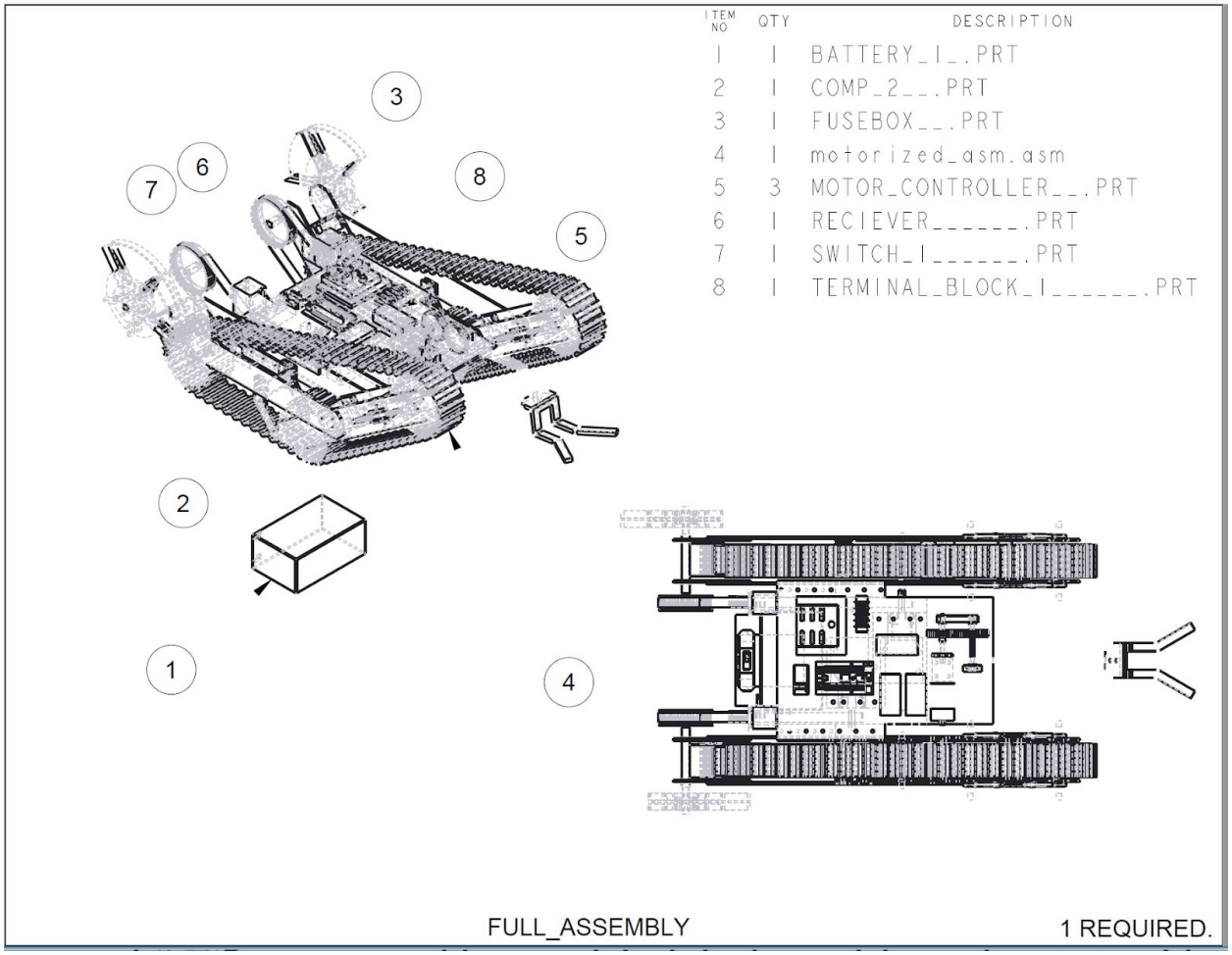


Figure 31: The Full Assembly of the SaRR

8 Results

8.1 Performance Estimates

Given that we were not able to physically obtain the dimensions of the robot course, the distance and the time required to complete the course were based on observations from videos of previous students who had completed the course.

As mentioned in the Power Budget section, the team's desired speed for the robot was 0.8 m/s and accounting for some inevitable energy losses via heat, transmission friction, and programming limits mainly determined the chosen gear ratio. This gear ratio is optimized around the robot having the speed of 0.5 m/s, and with additional deceleration at certain points of the course, like wall-breaching vs. chute navigation, that robot speed was further reduced. This speed takes into account the torque that would be required to drive the robot's treads and wheels over the wall.

The estimated times from observed videos and the estimated robot speeds allow us to calculate the distances that the robot will be running, which is shown below:

Task	Task breakdown	Average Speed (m/s)	Time (s)	Distance (m)
Med-kit pickup	accelerate to drive to med-kit	0.4	15	6
	arm motor to retrieve and store			
Speed segment	accelerate	0.5	40	20
	turn corner	0.4	10	4
Breach wall	increase potential energy	0.3	15	4.5
Drive through chute	accelerate through multiple turns	0.25	30	7.5
	wall proximity sensors*			
Locate target; deposit med-kit	accelerate	0.25	30	7.5
	light sensors*			
	arm motor			
			Total distance traveled	49.5

Table 7: Task Breakdown in Distance and Time

The maximum time estimated to take the robot to complete the course is 150 seconds, based on observed videos with large rooms for error.

Based on all simulations, the robot is physically able to withstand the drop test and its fall after the wall-breaching. The estimated speeds shown above and in the power budget are appropriately estimated to allow enough room of error and slowdown for the robot if necessary during the drive through chute, locating the light source, and depositing the med-kit. Last but not least, the power consumption of the robot is about 45% of the total energy of the battery, which is less than half and makes us confident that the robot will be able to make it through the course with one fully-charged battery.

8.2 Performance Estimates (Programming)

The programming team was unable to implement and debug the code on a fully assembled robot due to aforementioned circumstances. In lieu of that and the countless hours that would’ve been spent debugging, the team opted to plan and analyze the software and electronics for a slightly more challenging fully autonomous build. In order to retrofit the base build, which utilized an

Arduino microcontroller and an array of simple sensors, the programming team added a RaspberryPi with an RGB camera module. The Raspberry Pi is able to perform more complex computations, and runs an open source computer vision library developed by Intel called OpenCV. The Raspberry Pi interfaced with the Arduino through GPIO pins, which allowed it to send high/low signals to execute simple commands such as turning left, right, and raising/lowering the arm.

This method represents the simplest approach to a very difficult problem. As such, it suffers from limitations. For starters, it would require a lot of experimentally derived data to fine tune it's operations. Making the robot fully autonomous increases the complexity of the system and number of failure points, which leads to a lower theoretical reliability in the long run. To combat this, our robot would retain the ability to be controlled manually, so that in the event of a failure or malfunction, the mission could still be completed.Additionally, the code used for finding the light source is unoptimized, and may lead to slow or jerky performance on the last leg of the course. Another limitation this design faces is the low bandwidth of information communicated between the Arduino and the Raspberry Pi. The information is only carried omnidirectionally from the Pi to the Arduino, and the ports can only send a binary high/low signal. The alternative approach would be to create a master/slave relationship between the Pi and the Arduino where the PI runs all scripts and logic, and the arduino is used just as an interface with the motors and sensors. This approach should be implemented in future iterations using a library such as nanpy.

Overall, the comprehensive programming for this project represents a balanced approach. While there may be faster, or more efficient implementations, the approach taken meets or exceeds all requirements while still being feasible given the timeline, resources, and limited software experience of the team.

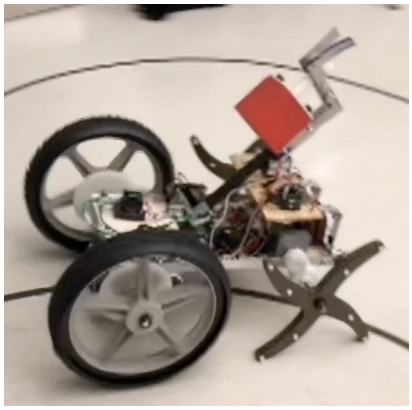
9 Conclusions and Further Work

The robot was humorously named "Ceci N'est Pas Un Robot" , yet this project turned out to be anything but a joke. This team was likely one of the first teams formed and began meeting weekly shortly thereafter. The first task was to brainstorm, evaluate, and choose a general design. The main differentiating feature of the designs was how the robot would scale the wall. Through a cost-benefit analysis of the three main brainstorming suggestions, the design we picked was interesting, original, and still feasible when looking at cost and time efficiency. With an estimated time of around 2:20, our SaRR would make it to the med drop off point well within the time requirements. This team is proud of both the learning done to get here and our SaRR that we can proudly stand behind. Our solid CAD and programming foundation mean that we would be ready to start the manufacturing stage if needed.

Future extensions of this work could include integrating some form of suspension system into the robot to manage shock loading, especially for the drop test and the fall off the wall. This was included in the initial design concept, but ended up being too complex to fit into the final design. In the initial idea, the forward tread arms would have been able to pivot slightly, resisted by a spring, but the required spring force would have been quite high given that the suspension travel would have to be restricted in order to avoid losing tension in the treads.

Appendix

A.1: A robot similar to the jointed SaRR design idea that was considered by the team. Note that this robot is not jointed.



A.2: An example of the x-paddle SaRR, one design idea that was considered by the team.

BILL OF MATERIALS. TOTAL COST $429.79.

Sub Team	Description	Quantity	Source	Part Number	link	Notes	Cost Per Unit	Total Cost***
-	-	-	-	-	-	-	-	$429.79
Mechanical Fabrication	-	-	-	-	-	-	-	**$356.08**
	1/2" Oilite Bronze Bushings, Flanged	4	McMaster Carr	6338K417	https://www.mcmaster.com/6338K417	for wheel shafts	$1.22	$4.88
	3/8" Oilite Bronze Bushings, Flanged	4	McMaster Carr	6338K414	https://www.mcmaster.com/6338K414	for driven tread pulleys	$0.88	$3.52
	3/8" Oilite Bronze Bushings, Unflanged	8	McMaster Carr	6391K876	https://www.mcmaster.com/6391K876	for free tread pulleys	$0.65	$5.20
	1/4" Oilite Bronze Bushings	3	McMaster Carr	6338K412	https://www.mcmaster.com/6338K412	for the arm	$0.82	$2.46
	3/8" Al 6061 Round Shaft, 3 ft	1	McMaster Carr	8974K24	https://www.mcmaster.com/8974K24	Tread Pulley Shafts, 6 x 5"	$5.40	$5.40
	1/2" Al6061 Round Shaft, 1 ft	1	McMaster Carr	8974K28	https://www.mcmaster.com/8974K28	Rear Wheel Shafts, 2 x 6"	$2.50	$2.50
	1/4" Al6061 Round Shaft, 1/2 ft	1	McMaster Carr	8974K22	https://www.mcmaster.com/8974K22	arm shaft	$1.59	$1.59
	Al 6061 1/8"x12"x24" Plate	1	McMaster Carr	89015K28	https://www.mcmaster.com/89015K28	top& bottom chassis plates	$47.11	$47.11
	Al 6061 1/8"x2" Bar, 6.5 ft (6 ft + 6")	2	McMaster Carr	8975K582	https://www.mcmaster.com/8975K582	chassis sides/tread arms/truss connectors	$12.03	$24.06
	Al 6061 2" U-Channel, 1.5 ft (1 ft + 6")	1	McMaster Carr	1630T29	https://www.mcmaster.com/1630T29	tread frame connecting bits	$10.58	$10.58
	Al 6061 1" Angle Beam, 3 ft	1	McMaster Carr	8982K11	https://www.mcmaster.com/8982K11	chassis connecting bits	$10.75	$10.75
	Al 6061 3/8"x2" Bar, 1 ft	1	McMaster Carr	8975K59	https://www.mcmaster.com/8975K59	medkit arm and mounts	$5.87	$5.87
	Al 6061 1/4"x1 3/4" Bar, 6 in	1	McMaster Carr	8975K598	https://www.mcmaster.com/8975K598	arm fork	$2.49	$2.49
	4" UHMW Rod, 1 ft	1	McMaster Carr	8701K59	https://www.mcmaster.com/8701K59	tread pulleys	$35.01	$35.01
	Tumbuckles	4	McMaster Carr	3010T17	https://www.mcmaster.com/3010T17	for tensioning	$2.36	$9.44
	180T 5mm HTD belt, 15mm wide	1	Vex	217-3481	https://www.vexrobotics.com/htdbelts15.html	From vex robotics	$19.99	$19.99
	150T 5mm HTD belt, 15mm wide	1	Vex	217-3484	https://www.vexrobotics.com/htdbelts15.html	From vex robotics	$18.99	$18.99
	12" Pneumatic Wheels	2	Machine Shop	-	-	Available in the shop	$0.00	$0.00
	69T 20mm Double-Sided Belt, 50mm wide	2	Brecoflex		Called for a quote		$71.87	$143.74
	1/16" cotter pins (pack of 100)	1	McMaster Carr	90520A102	https://www.mcmaster.com/90520a102		$2.50	$2.50
	10-24 bolts	AR	Machine Shop	-	-	-	$0.00	$0.00
	3/8-16 nuts	12	Machine Shop	-	-	-	$0.00	$0.00
Electrical Fabrication	-	-	-	-	-	-	-	**$40.94**
	Victor SPX Motor Controllers	3	Sample Robot	-	http://www.ctr-electronics.com/victor-spx.html	Taken from sample robot	-	$0.00
	12V 10AH Lithium Battery	1	Sample Robot	-	https://dakotalithium.com/product/dakota-lithium-12v-10ah-battery/?v=eb021079e5a	Taken from sample robot	-	$0.00
	FlySky FS-i6 6ch Transmitter and Receiver	1	Sample Robot	-	https://www.amazon.com/Flysky-Transmitter-Multirotor-Helicopter-Quadcopter/dp/B07CX191CT/ref=sr_1_2?dchild=1&keywords=FlySky+FS-i6+2.4G+6ch+Transmitter+and+Receiver&qid=1588533423&sr=8-2	Taken from sample robot	-	$0.00
	Radio Shack 276-1657 Photocells 5-pack	1	Sample Robot	-	https://www.radioshack.com/products/rds-photoresistors	Taken from sample robot	-	$0.00
	Proximity sensors	3	Sample Robot	-	https://www.bananarobotics.com/shop/Sharp-GP2Y0A21YK0F-IR-Distance-Sensor	1 taken from sample robot, 2 ordered	$8.99	$17.98

	3 Axis Accelerometer Gyroscope Module	1	Sample Robot	-	https://www.amazon.com/HiLetgo-MPU-6050-Accelerometer-Gyroscope-Converter/dp/B01DK83ZYQ/ref=sr_1_3?crid=MCLWZNW0F1VP&dchild=1&keywords=3%2Baxis%2Baccelerometer%2Bmodule&qid=1588533909&sprefix=3%2Baxis%2Baccele%2Caps%2C279&sr=8-3&th=1	Taken from sample robot	$4.99	$4.99
	Teensy 3.2 USB Development Board	1	Sample Robot	-	https://store.arduino.cc/usa/teensy-3-2-usb-development-board	Taken from sample robot	-	$0.00
	40A Circuit Breaker	3	Sample Robot	-	https://www.vexrobotics.com/circuit-breakers.html	-	$5.99	$17.97
	Wires	AR	Machine Shop	-	-	-	-	$0.00
	Connectors	AR	Machine Shop	-	-	-	-	$0.00
Programming/Sensors	-	-	-	-	-	-	-	**$32.77**
	Raspberry Pi 3A+	1		-	https://www.microcenter.com/product/514076/3_Model_A_Board		$19.99	$19.99
	Raspberry Pi camera module	1			https://www.amazon.com/Raspberry-Camera-Module-Megapixels-Sensor/dp/B07L82XBNM/ref=sr_1_18?dchild=1&keywords=raspberry+pi&qid=1588026679&s=electronics&sr=1-18)		$8.79	$8.79
	Bump switch	1		-	https://www.wiltronics.com.au/product/9030/bump-sensor-switch/		$3.99	$3.99

A3 Final Bill of Materials not inclusive of shipping cost

Group Set Deadline		
Date	Progress	Goal
week of 2/24	class robot done	teleop drive train
week of 3/2	90% success rate	autonomous driving to light
3/27/20	finished	midterm PDR due
4/3/20		speed drive with ramp
week of 4/6		teleop wall climbing
week of 4/13		teleop object retrieval and placement
week of 4/20		autonomous chute and object placement
5/11, 5pm		robot locked away
5/12, 9am		testing!

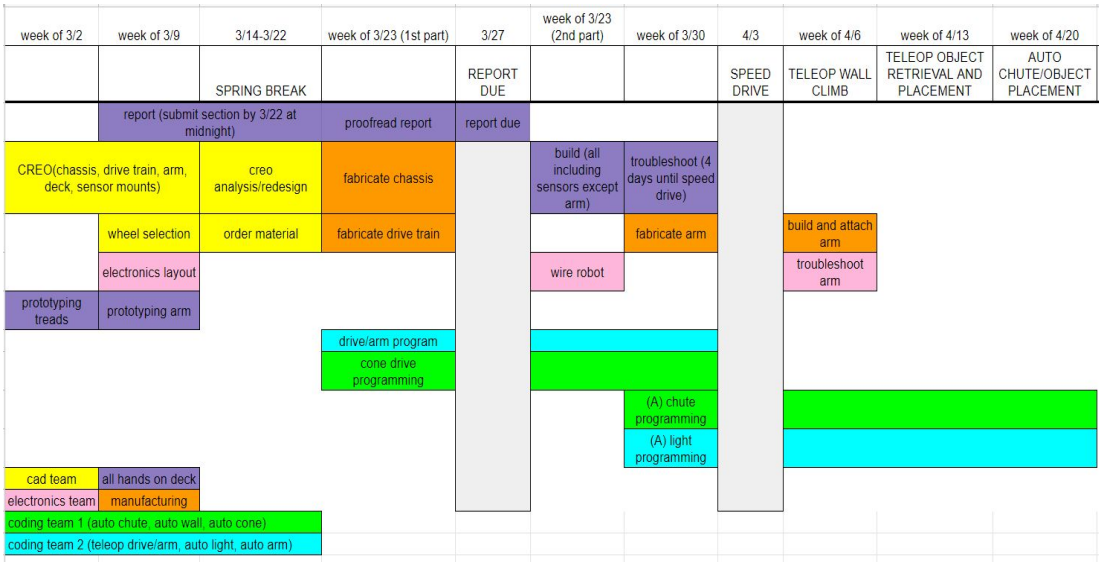
A.4: The (former) class timeline for the SaRR (includes deadlines and deliverables).

CAD	**Conor**	**Peter**	
Drawings	**Ava**	Peter	
Electronics	**Ekin**		
Programming	**Ainil**	Ekin	Noah
PM	**Ava**		
FBD/Simulations	**Lucy**	Celine	Ekin
Gear Ratio/Power Budget	**Celine**		

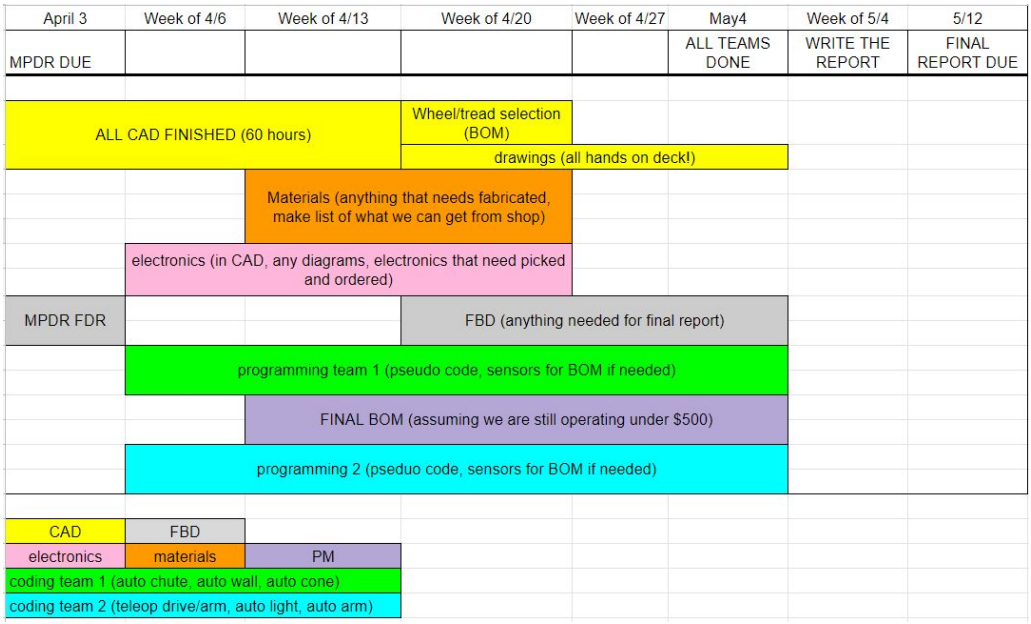
A.5: The sub-teams list, each of which has a point person/people who is bolded, as well as an overview of their assigned roles.

	CAD model, Mechanical Fabrication materials assignment (BOM), wheels/tread (BOM)
	CAD drawings
	electronics in CAD model, wiring diagrams, electronics (BOM)
	code/pseudocode, Sensors (BOM)
	run meetings, make agenda, Final BOM compilation, organization (schedules, lists, etc)
	Free Body Diagrams, CAD Simulations
	Gear Ratio and Power Budget Calculations

A.6: Main subteam responsibilities (paired with A.5)



A.7: The original schedule for the semester.



A.8: The revised schedule for the semester (which was created around midterms).

A.9: Autonomous Navigation Code

```
// MAE 322 QuaranTeam SaRR Code
#include <Servo.h>

//*********************************************************
```

```
//*************** CONSTANT DECLARATIONS ******************
//********************************************************
const int transmitterConstant = 21000;
const int autonomousActivation = 1600;
const int motorMin = 1000;
const int motorMax = 2000;
const int motorZero = 1500;

int Rwheel, Lwheel;

// *** pin numbers are arbitrary ***
int Ch1, Ch2, Ch3, Ch4, Ch5, Ch6;
const int Ch1Pin = 1;
const int Ch2Pin = 2;
const int Ch3Pin = 3;
const int Ch4Pin = 4;
const int Ch5Pin = 5;
const int Ch6Pin = 6;

// *** pin numbers are arbitrary ***
const int rPhotoSensorPin = A0;
const int rPhotoSensorPin = A1;
const int fDistSensorPin = A2;
const int rDistSensorPin = A3;
const int lDistSensorPin = A4;
const int R_ServoPin = A5;
const int L_ServoPin = A6;
const int Medkit_ServoPin = A7;

bool finishedLineFollowing = false;
bool finishedWall = false;
bool finishedChute = false;
bool finishedLightNavigation = false;
bool finishedMedkit = false;
bool faceWall = false;
bool treadsActivated = false;

// Declaring servo as global variables
Servo R_Servo;
Servo L_Servo;
Servo Medkit_Servo;

// Declaring sensors as global variables
int rPhotoSensor, lPhotoSensor, fDistSensor, rDistSensor, lDistSensor, photoSensorDiff, distSensorDiff;

//********************************************************
//********************** setup() *************************
```

```
//*************************************************************
void setup() {
  pinMode(Ch1Pin, INPUT);
  pinMode(Ch2Pin, INPUT);
  pinMode(Ch3Pin, INPUT);
  pinMode(Ch4Pin, INPUT);
  pinMode(Ch5Pin, INPUT);
  pinMode(Ch6Pin, INPUT);

  pinMode(rPhotoSensorPin, INPUT);
  pinMode(lPhotoSensorPin, INPUT);
  pinMode(fDistSensorPin, INPUT);
  pinMode(rDistSensorPin, INPUT);
  pinMode(lDistSensorPin, INPUT);

  R_Servo.attach(R_ServoPin);
  L_Servo.attach(L_ServoPin);
  Medkit_Servo.attach(Medkit_ServoPin);
}

//*************************************************************
//********************** forward() ************************
//*************************************************************
void forward(int delayTime, int power)
{
  int right = motorZero - 100*power;
  int left = motorZero + 100*power;

  constrain(right, motorLow, motorHigh);
  constrain(left, motorLow, motorHigh);

  R_Servo.writeMicroseconds(right);  // sets the servo position
  L_Servo.writeMicroseconds(left);   // sets the servo position

  delay(delayTime);
}

//*************************************************************
//********************** turnLeft() ***********************
//*************************************************************
void turnLeft(int delayTime, int power)
{
  int right = motorZero - 100*power;
  int left = motorZero;

  constrain(right, motorLow, motorHigh);
  constrain(left, motorLow, motorHigh);
```

```
  R_Servo.writeMicroseconds(right);  // sets the servo position
  L_Servo.writeMicroseconds(left);   // sets the servo position

  delay(delayTime);
}

//*************************************************************
//********************** turnRight() **********************
//*************************************************************
void turnRight(int delayTime, int power)
{
  int right = motorZero;
  int left = motorZero + 100*power;

  constrain(right, motorLow, motorHigh);
  constrain(left, motorLow, motorHigh);

  R_Servo.writeMicroseconds(right);  // sets the servo position
  L_Servo.writeMicroseconds(left);   // sets the servo position

  delay(delayTime);
}

//*************************************************************
//********************** backward() ***********************
//*************************************************************
void backward(int delayTime, int power)
{
  int right = motorZero + 100*power;
  int left = motorZero - 100*power;

  constrain(right, motorLow, motorHigh);
  constrain(left, motorLow, motorHigh);

  R_Servo.writeMicroseconds(right);  // sets the servo position
  L_Servo.writeMicroseconds(left);   // sets the servo position

  delay(delayTime);
}

//*************************************************************
//********************** stopRobot() **********************
//*************************************************************
void stopRobot(int delayTime)
{
  R_Servo.writeMicroseconds(motorZero);  // sets the servo position
```

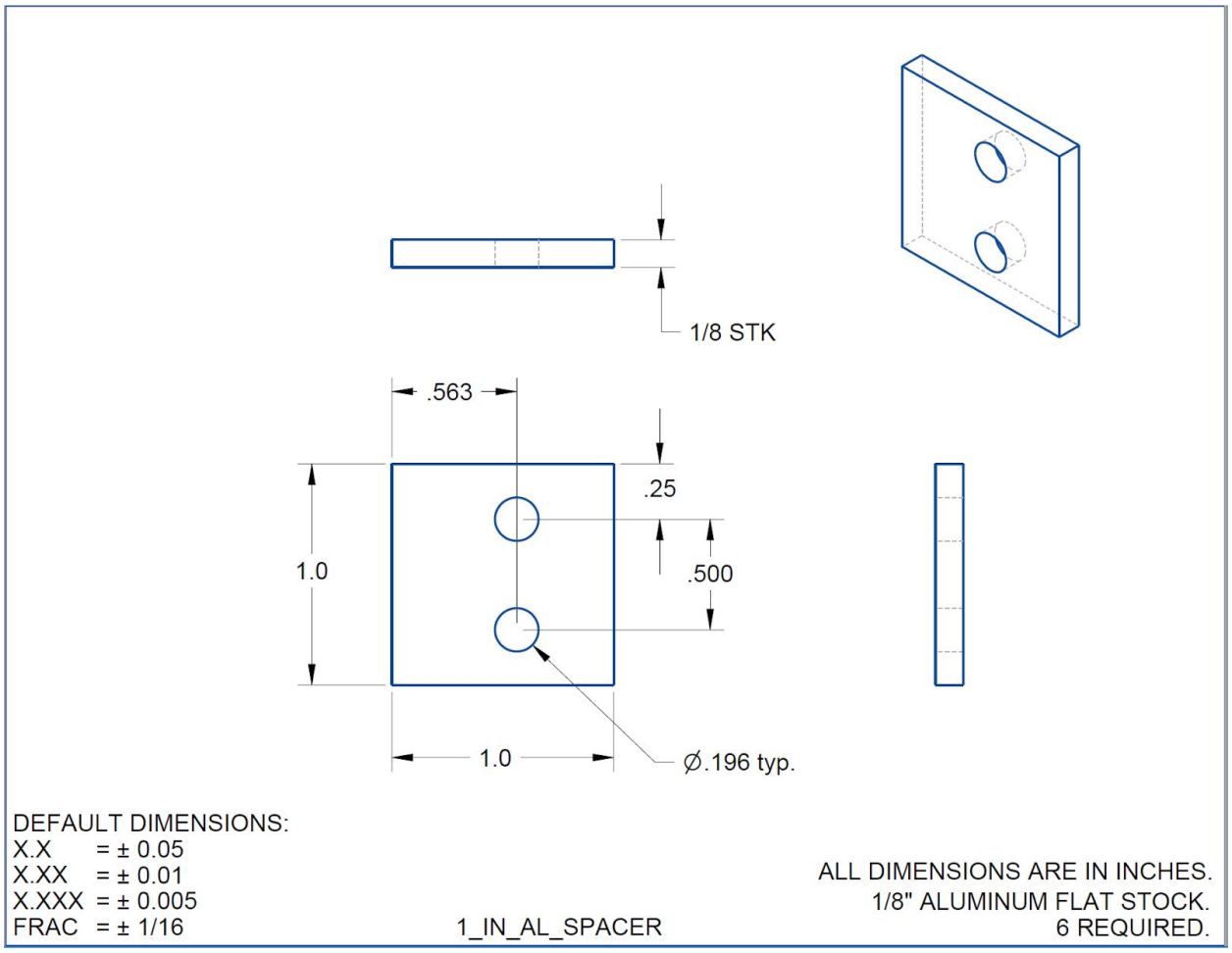
```
  L_Servo.writeMicroseconds(motorZero);   // sets the servo position

  delay(delayTime);
}

//*************************************************************
//******************** checkSensors() ***********************
//*************************************************************
void checkSensors() {
  // Read photo sensors
  rPhotoSensor = analogRead(rPhotoSensorPin);
  lPhotoSensor = analogRead(lPhotoSensorPin);
  photoSensorDiff = abs(lPhotoSensor - rPhotoSensor);

  // Read distance sensors
  for (int i = 0; i < 4; i++) {
    fDistSensor += analogRead(fDistSensorPin);
    rDistSensor += analogRead(rDistSensorPin);
    lDistSensor += analogRead(lDistSensorPin);
  }
  fDistSensor /= 5;
  rDistSensor /= 5;
  lDistSensor /= 5;

  distSensorDiff = rDistSensor - lDistSensor;
}

//*************************************************************
//******************** lineFollowing() ************************
//*************************************************************
void lineFollowing() {
  /* Line following is handled by the Raspberry Pi
  See the pseudocode in the Autonomous Navigation section for this part. */
}

//*************************************************************
//******************** wallTraverse() *************************
//*************************************************************
void wallTraverse() {
  if (fDistSensor < 500 && !faceWall) {
    forward(200, 2);
  }
  else {
    facewall = true;
    treadsActivated = true;
    forward(500, 5);
    stopRobot(200);
```

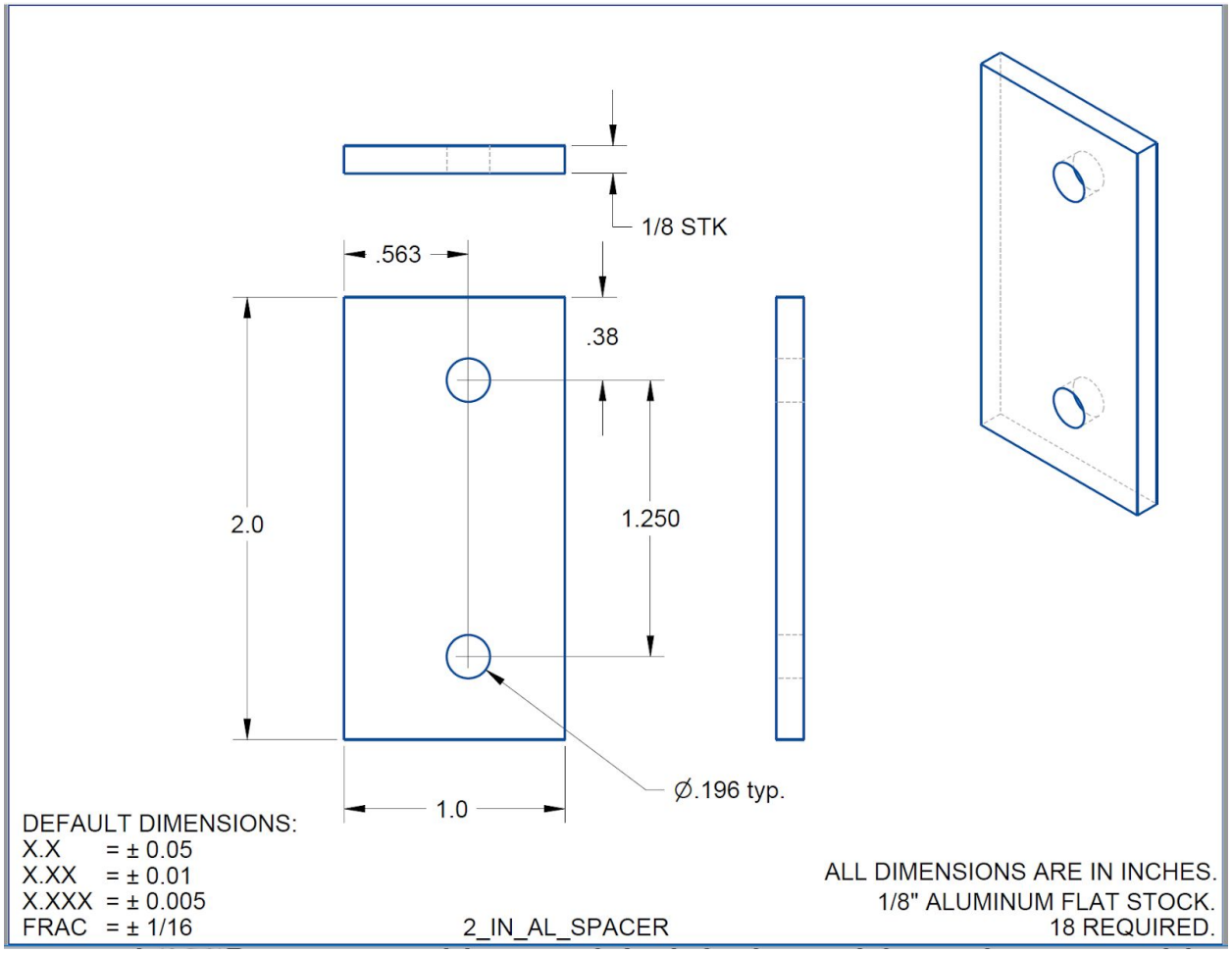
```
    finishedWall = true;
  }
}
//************************************************************
//********************* chuteTraverse() **********************
//************************************************************
void chuteTraverse() {

  if (rDistSensor > 100 && lDistSensor > 100) {
    if(distSensorDiff > 50) {
      turnLeft(10, 2);
    }
    else if(distSensorDiff < -50) {
      turnRight(10, 2);
    }
    else {
      forward(10, 3);
    }
  }
  else {
    stopRobot(100);
    finishedChute = true;
  }
}

//************************************************************
//******************** lightNavigation() *********************
//************************************************************
void lightNavigation() {

  if (lPhotoSensor > 350 && rPhotoSensor > 350) {
    if (lPhotoSensor > rPhotoSensor) {
      turnRight(100, 2);
    }
    else {
      turnLeft(100, 2);
    }
  }
  else if (photoSensorDiff > 70) {
    if (lPhotoSensor < rPhotoSensor) {
      turnLeft(100, 2);
      forward(100, 3);
    }
    else {
      turnRight(100, 2);
      forward(100, 3);
      }
```

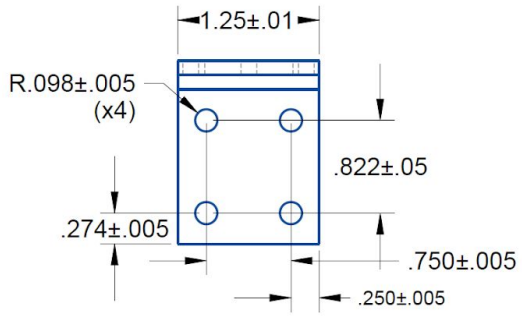
```
  }
  else {
    forward(200, 3);
  }
  finishedLightNavigation = true;
}

//************************************************************
//******************** placeMedkit() ***********************
//************************************************************
void placeMedkit() {

  Medkit_Servo.writeMicroseconds(1600);
  delay(500);
  Medkit_Servo.writeMicroseconds(motorZero);
  backward(100, 3);
  finishedMedkit = true;
}

//************************************************************
//********************** autonomous() ************************
//************************************************************
void autonomous() {
  checkSensors();

  if (!finishedLineFollowing)
    lineFollowing();
  else if (finishedLineFollowing && !finishedWall)
    wallTraverse();
  else if (finishedLineFollowing && finishedWall && !finishedChute)
    chuteTraverse();
  else if (finishedLineFollowing && finishedWall && finishedChute && !finishedLightNavigation)
    lightNavigation();
  else if (finishedLineFollowing && finishedWall && finishedChute && finishedLightNavigation &&
!finishedMedkit)
    placeMedkit();
  else
    stopRobot();
}

//************************************************************
//************************* loop() ***************************
//************************************************************
void loop(){
  Ch1 = pulseIn(Ch1Pin, HIGH, 21000);
  Ch2 = pulseIn(Ch2Pin, HIGH, 21000);
  Ch3 = pulseIn(Ch3Pin, HIGH, 21000);
```

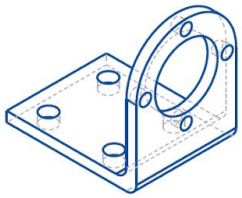
```
  Ch4 = pulseIn(Ch4Pin, HIGH, 21000);
  Ch5 = pulseIn(Ch5Pin, HIGH, 21000);
  Ch6 = pulseIn(Ch6Pin, HIGH, 21000);

  checkSensors();

  if (Ch5 > autonomousActivation)
    autonomous();
  else
    driveServosRC();
}

//*************************************************************
//********************** driveServosRC() **********************
//*************************************************************
void driveServosRC() {
  if (Ch2 <= 1500) {
    Lwheel = Ch1 + Ch2 - 1500;
    Rwheel = Ch1 - Ch2 + 1500;
    SetLimits();
  }
  if (Ch2 > 1500) {
    int Ch1_mod = map(Ch1, 1000, 2000, 1000, 2000); // Invert the Ch1 axis to keep the math similar
    Lwheel = Ch1_mod + Ch2 - 1500;
    Rwheel = Ch1_mod - Ch2 + 1500;
    pulseMotors();
  }
}

//********************   pulseMotors  ***************************
//pulses either mapped or direct signals
//*************************************************************
void pulseMotors() {

  //un-comment the next two to map a control range.
  //*** Take the standard range of 1000 to 2000 and frame it to your own minimum and maximum
  //*** for each wheel.
  Rwheel = map(1700, 1000, 2000, 1000, 2000);
  Lwheel = map(1300, 1000, 2000, 1000, 2000);
  R_Servo.writeMicroseconds(Rwheel);
  L_Servo.writeMicroseconds(Lwheel);

  // un-comment this line do display the value being sent to the motors
  //  PrintWheelCalcs(); //REMEMBER: printing values slows reaction times

}
```

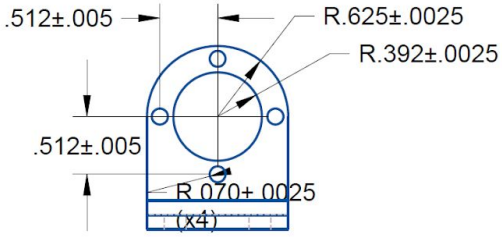
Appendix B: Drawings

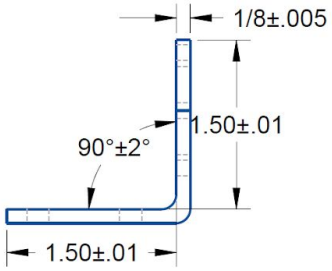



1/8 STK
.563
.38
2.0
1.250
Ø.196 typ.
1.0
DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16
2_IN_AL_SPACER
ALL DIMENSIONS ARE IN INCHES.
1/8" ALUMINUM FLAT STOCK.
18 REQUIRED.





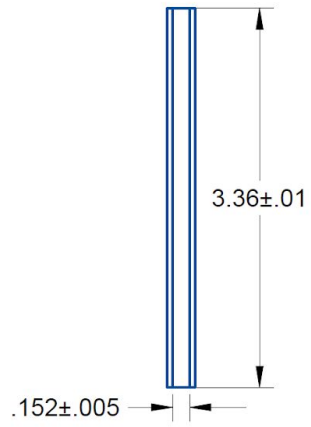


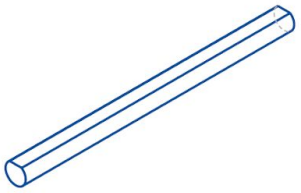


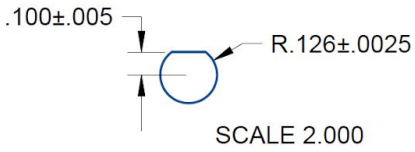
DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16

ARM_MOTOR_MOUNT

ALL DIMENSIONS ARE IN INCHES.
BENT FROM 1/8" FLAT ALUMINUM STOCK.
1 REQUIRED.



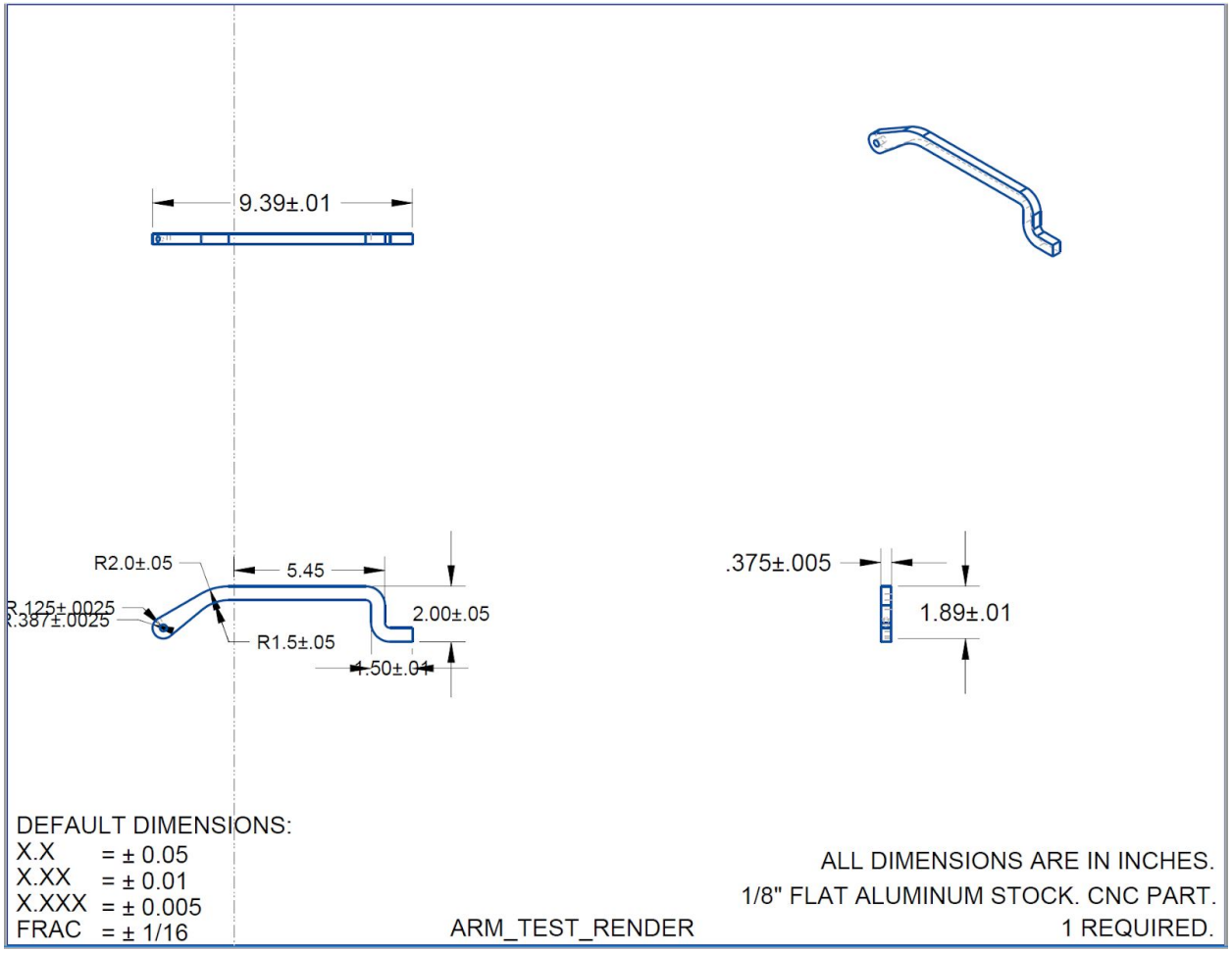


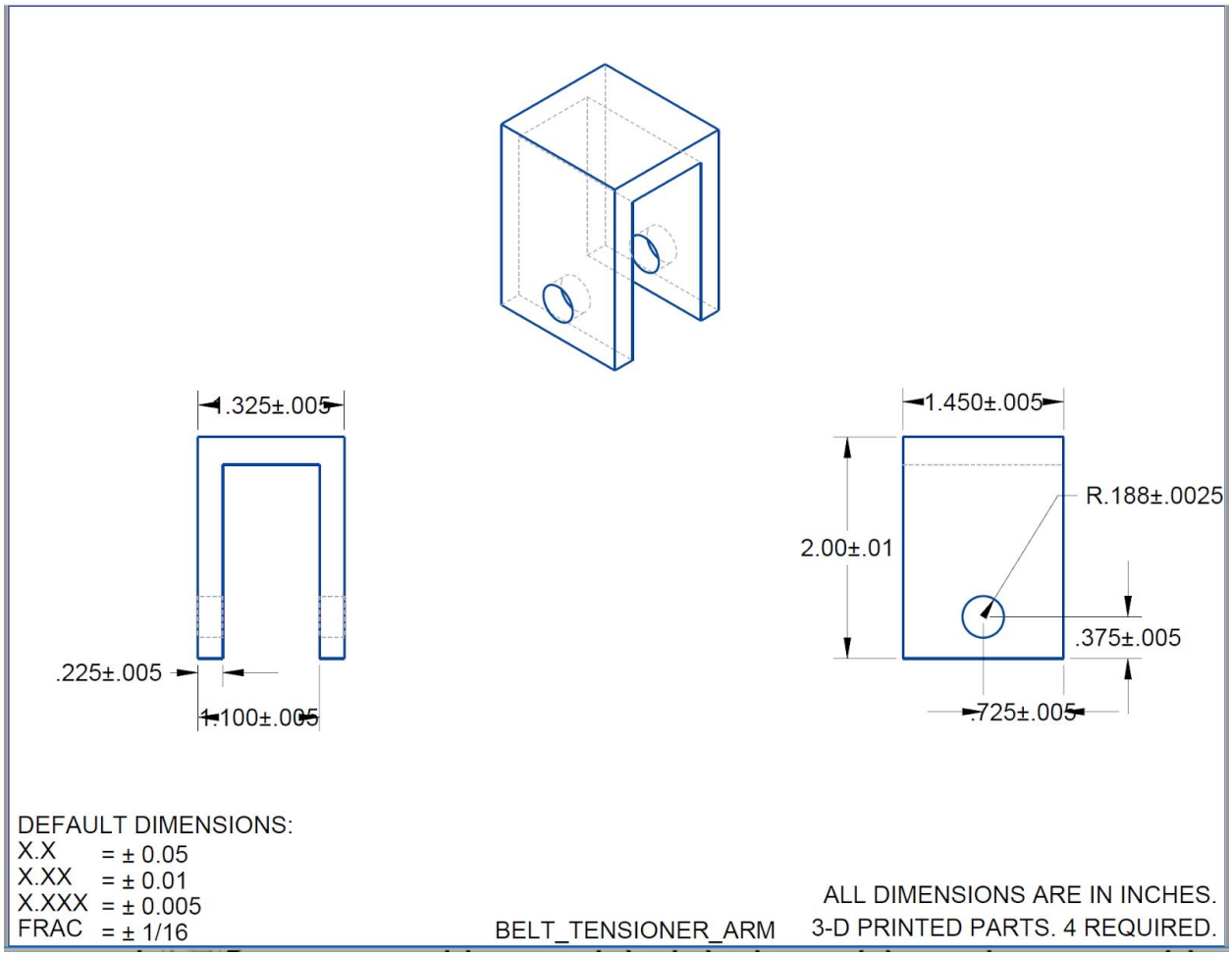


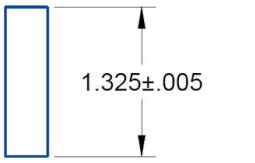
DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16

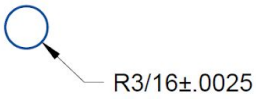
ARM_SHAFT

ALL DIMENSIONS ARE IN INCHES.
1/4" ALUMINUM BAR STOCK.
1 REQUIRED.


9.39±.01
R2.0±.05
5.45
R.125±.0025
.387±.0025
2.00±.05
R1.5±.05
1.50±.04
.375±.005
1.89±.01
DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16
ARM_TEST_RENDER
ALL DIMENSIONS ARE IN INCHES.
1/8" FLAT ALUMINUM STOCK. CNC PART.
1 REQUIRED.


1.325±.005
.225±.005
1.100±.005
1.450±.005
2.00±.01
R.188±.0025
.375±.005
.725±.005
DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16
BELT_TENSIONER_ARM
ALL DIMENSIONS ARE IN INCHES.
3-D PRINTED PARTS. 4 REQUIRED.

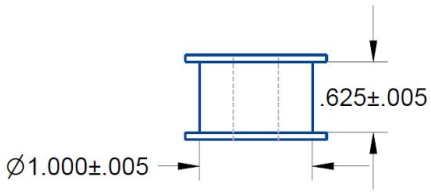


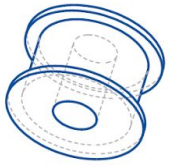


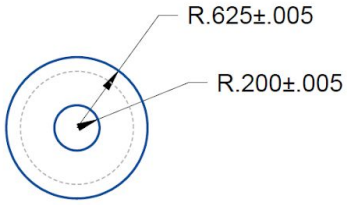
DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC =± 1/16

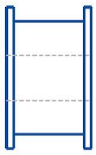
BELT_TENSIONER_AXLE

ALL DIMENSIONS ARE IN INCHES.
3-D PRINTED PART. 4 REQUIRED.





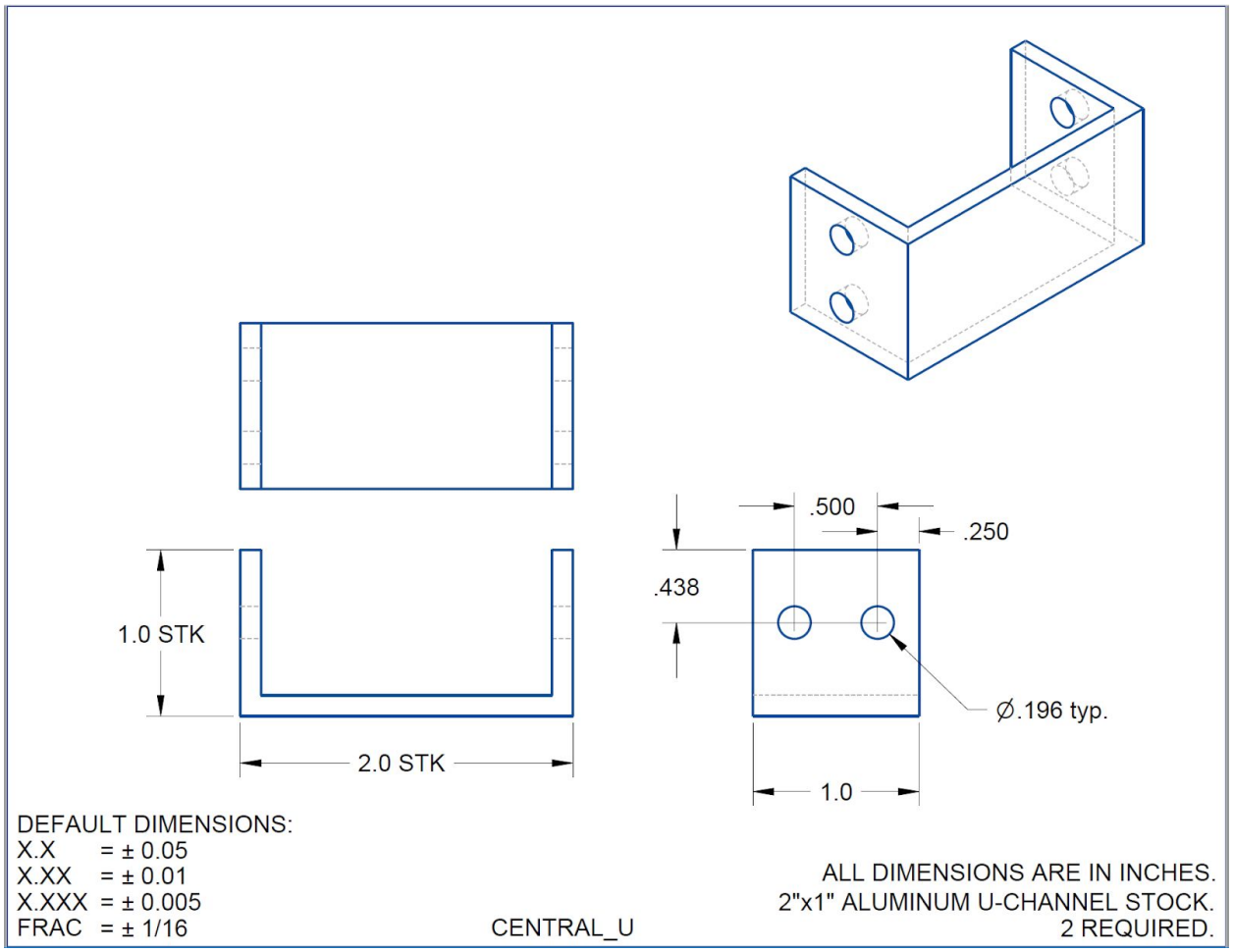




DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± .0.1
X.XXX = ± 0.05
FRAC = ± 1/16

BELT_TENSIONER_PULLEY

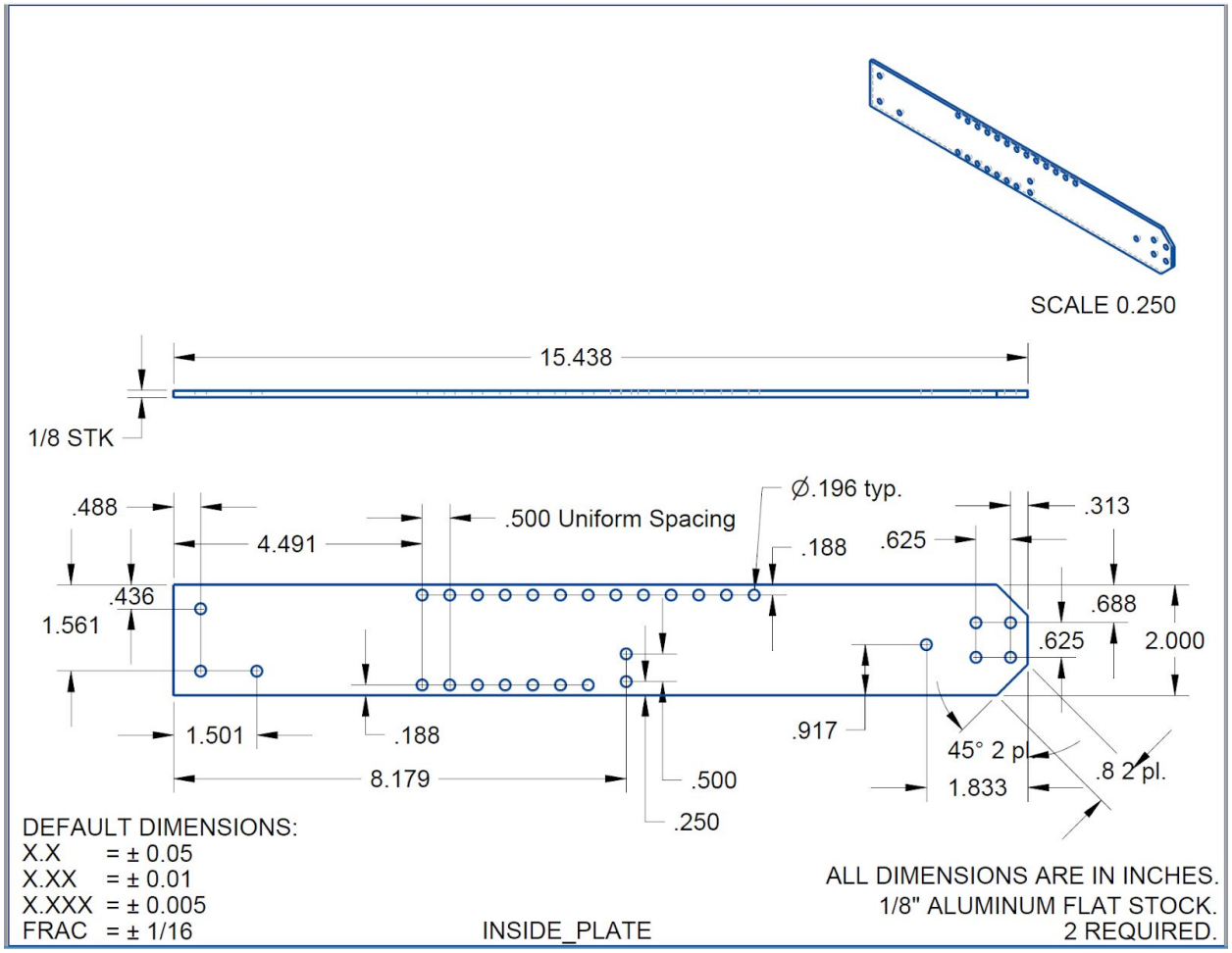
ALL DIMENSIONS ARE IN INCHES.
3-D PRINTED PART. 4 REQUIRED

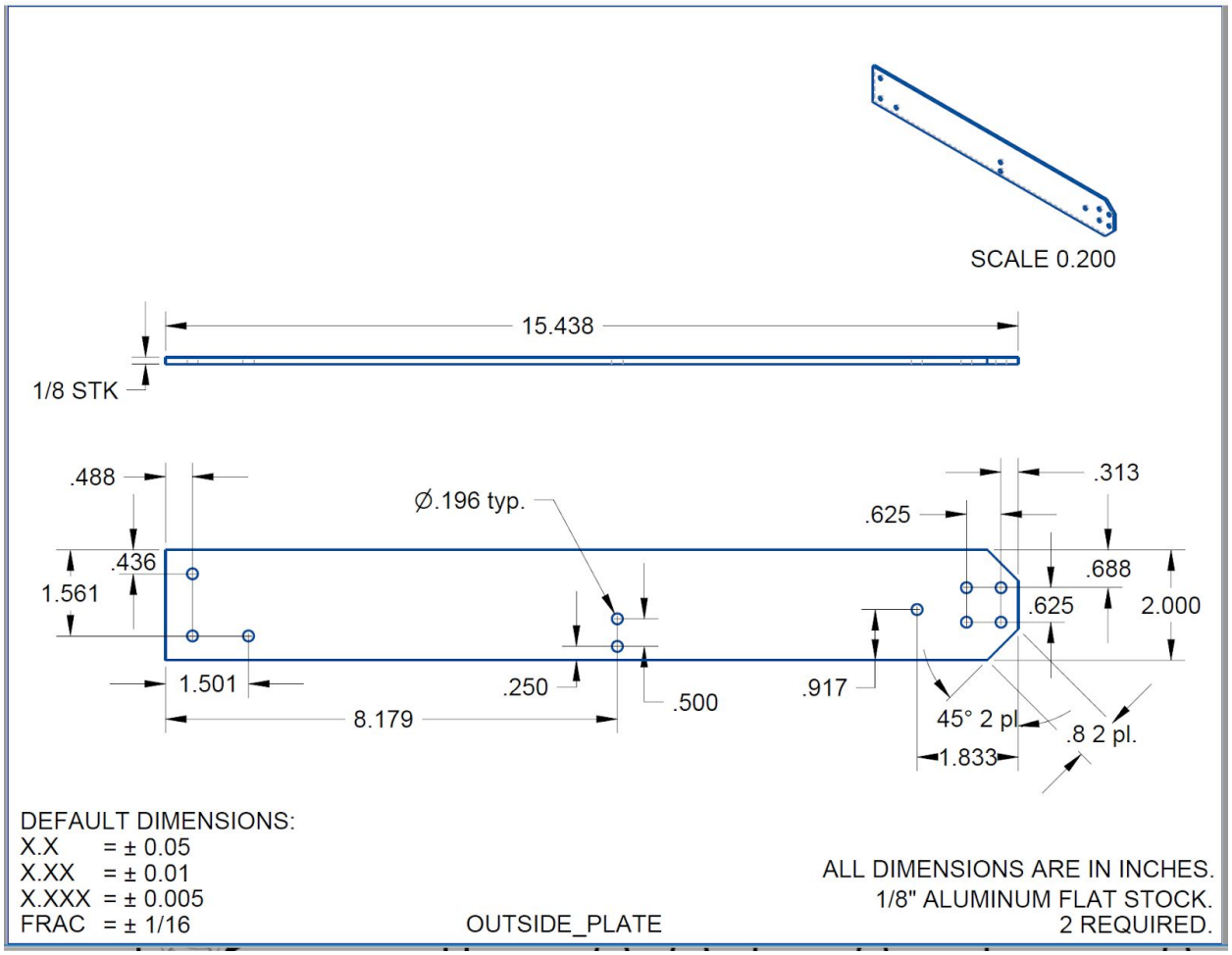


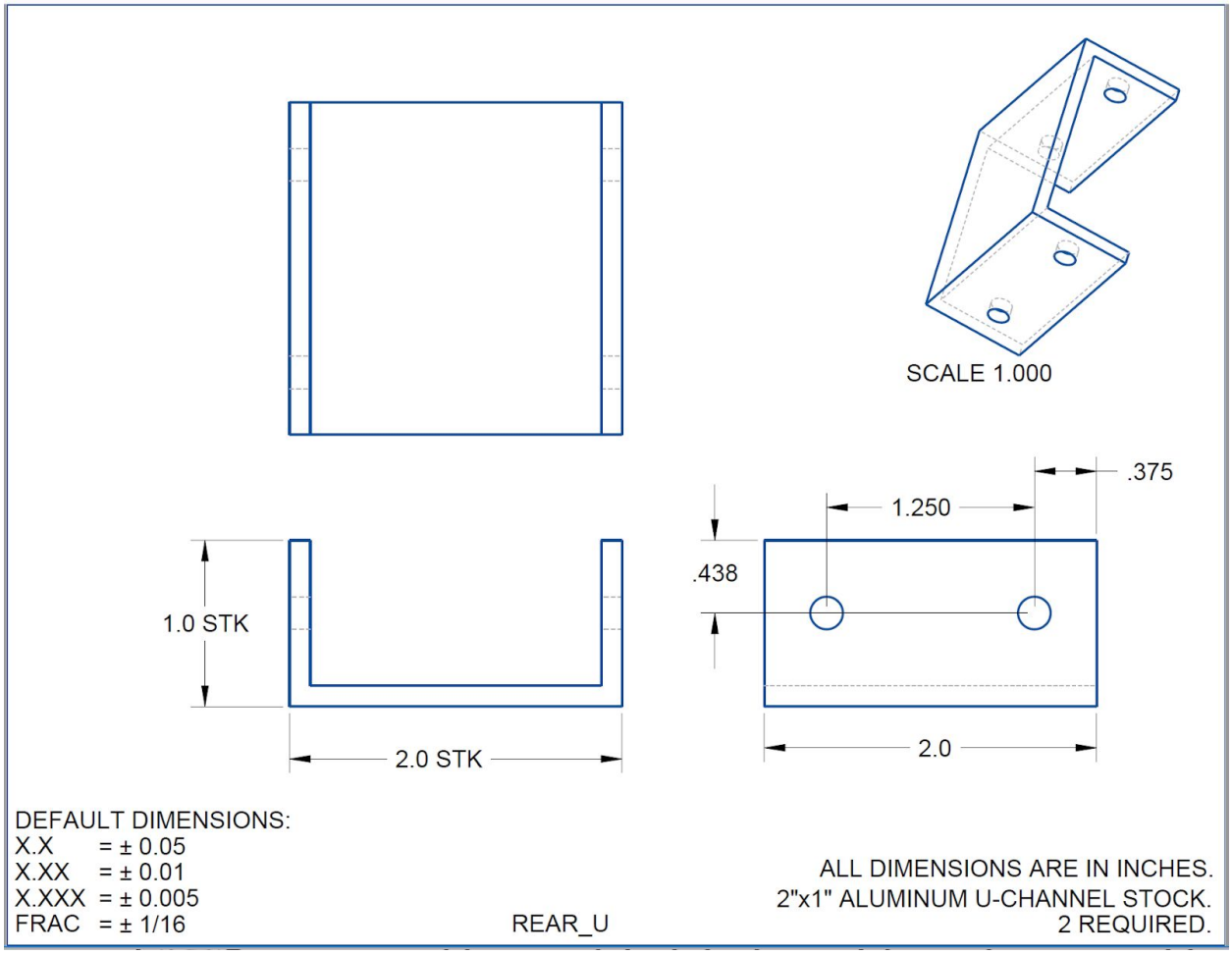
DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16

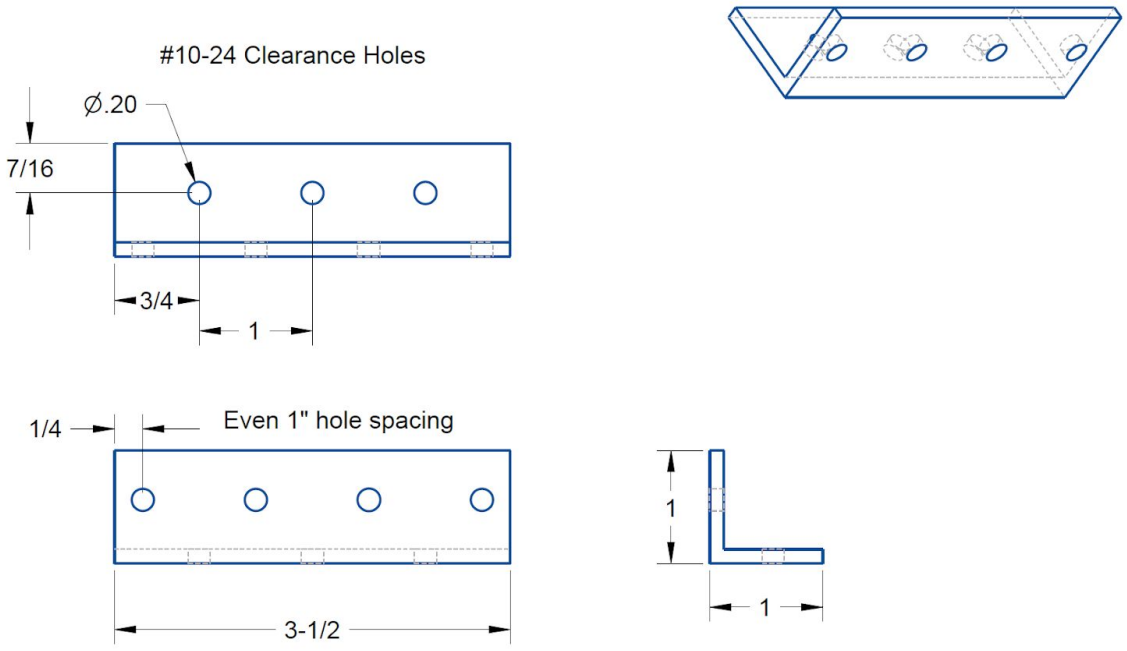
CENTRAL_U

ALL DIMENSIONS ARE IN INCHES.
2"x1" ALUMINUM U-CHANNEL STOCK.
2 REQUIRED.


SCALE 0.250
15.438
1/8 STK
.488
4.491
.500 Uniform Spacing
Ø.196 typ.
.188
.625
.313
.436
1.561
.688
.625
2.000
1.501
.188
.917
45° 2 pl.
.8 2 pl.
8.179
.500
1.833
.250
DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16
INSIDE_PLATE
ALL DIMENSIONS ARE IN INCHES.
1/8" ALUMINUM FLAT STOCK.
2 REQUIRED.


SCALE 0.200
15.438
1/8 STK
.488
Ø.196 typ.
.313
.625
.436
.688
1.561
.625
2.000
1.501
.250
.500
.917
8.179
45° 2 pl.
.8 2 pl.
1.833
DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16
OUTSIDE_PLATE
ALL DIMENSIONS ARE IN INCHES.
1/8" ALUMINUM FLAT STOCK.
2 REQUIRED.

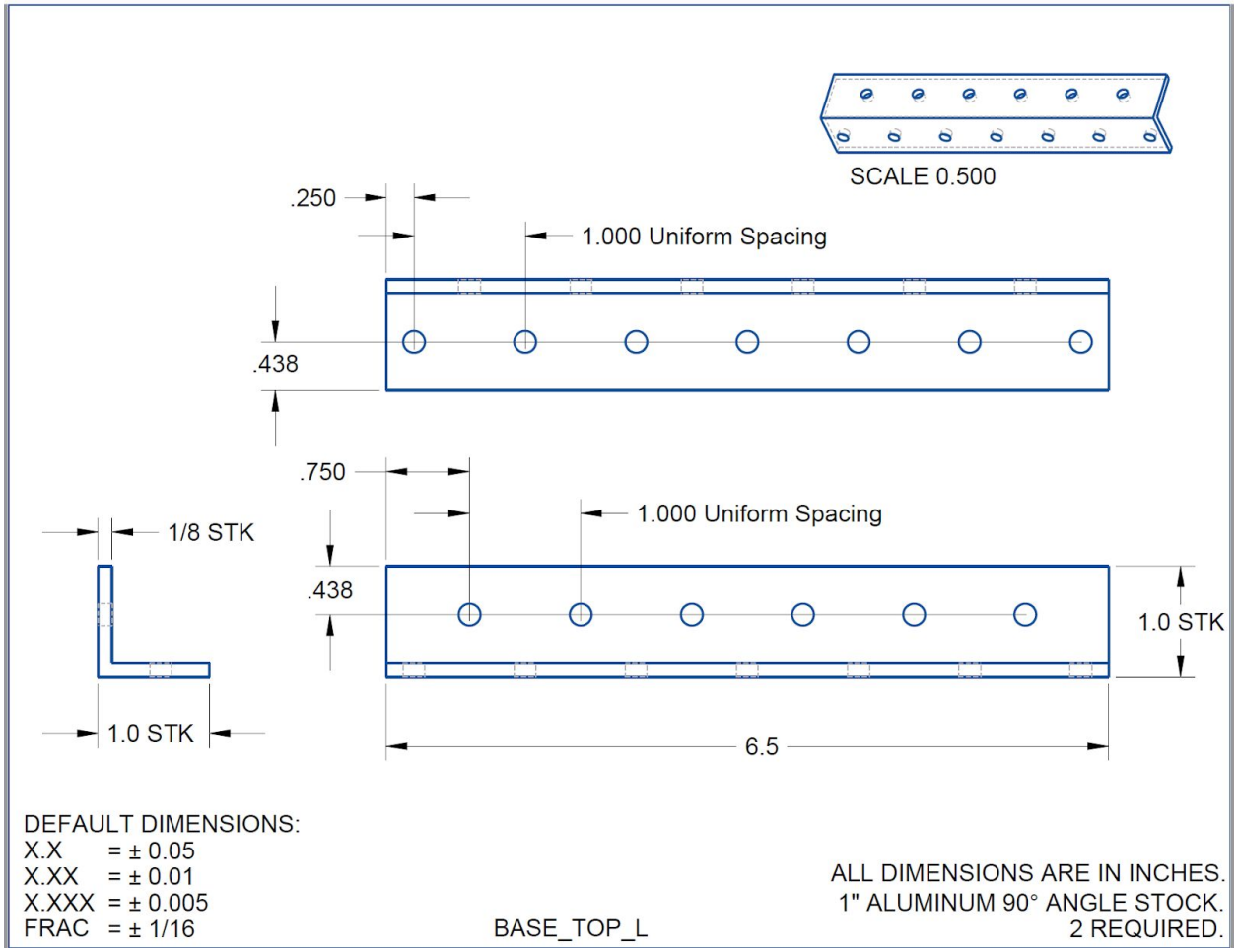

SCALE 1.000
.375
1.250
.438
1.0 STK
2.0 STK
2.0
DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16
REAR_U
ALL DIMENSIONS ARE IN INCHES.
2"x1" ALUMINUM U-CHANNEL STOCK.
2 REQUIRED.



DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16

BASE_BOTTOM_L

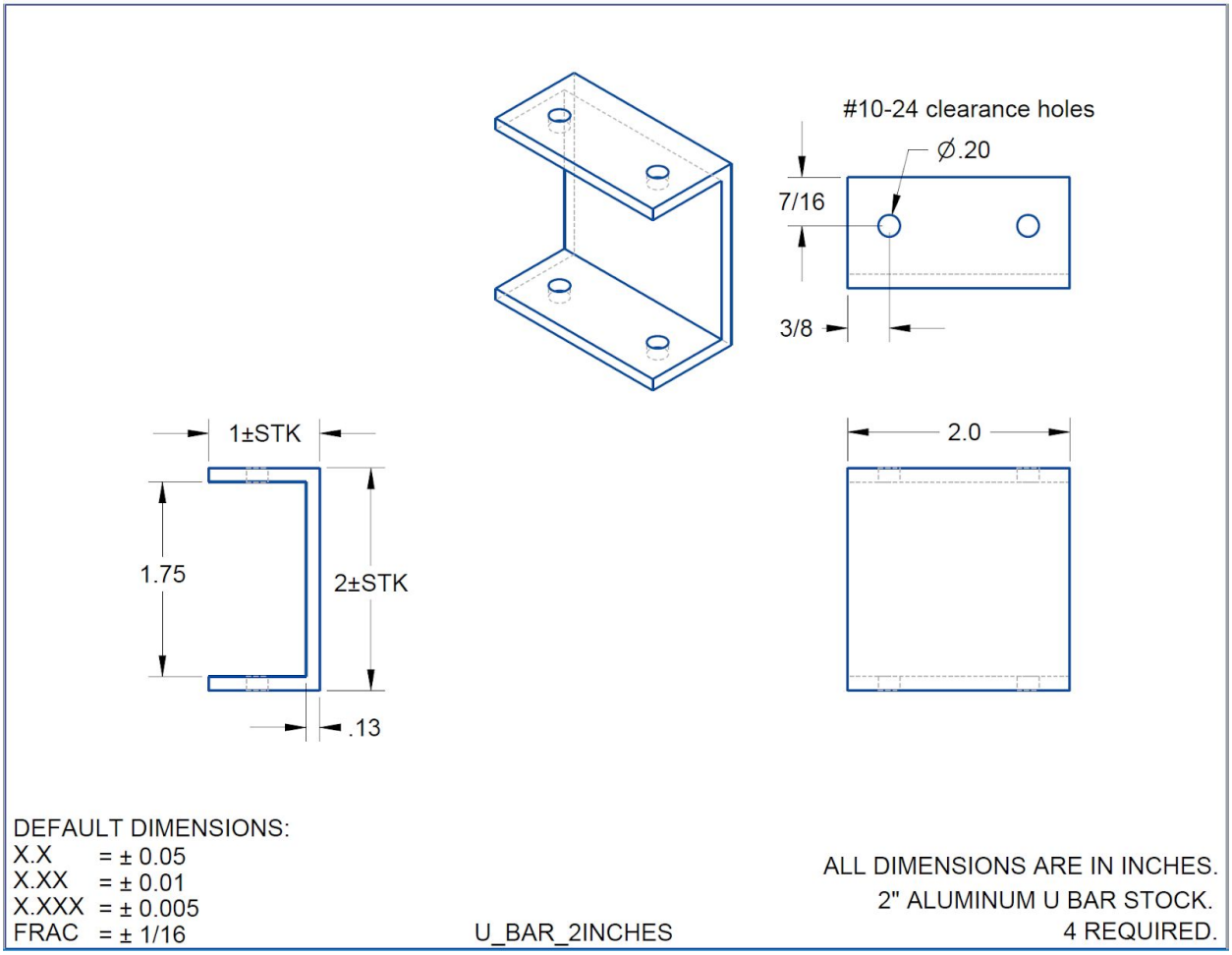
ALL DIMENSIONS ARE IN INCHES.
1" ALUMINUM 90° ANGLE STOCK.
2 REQUIRED.



DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16

BASE_TOP_L

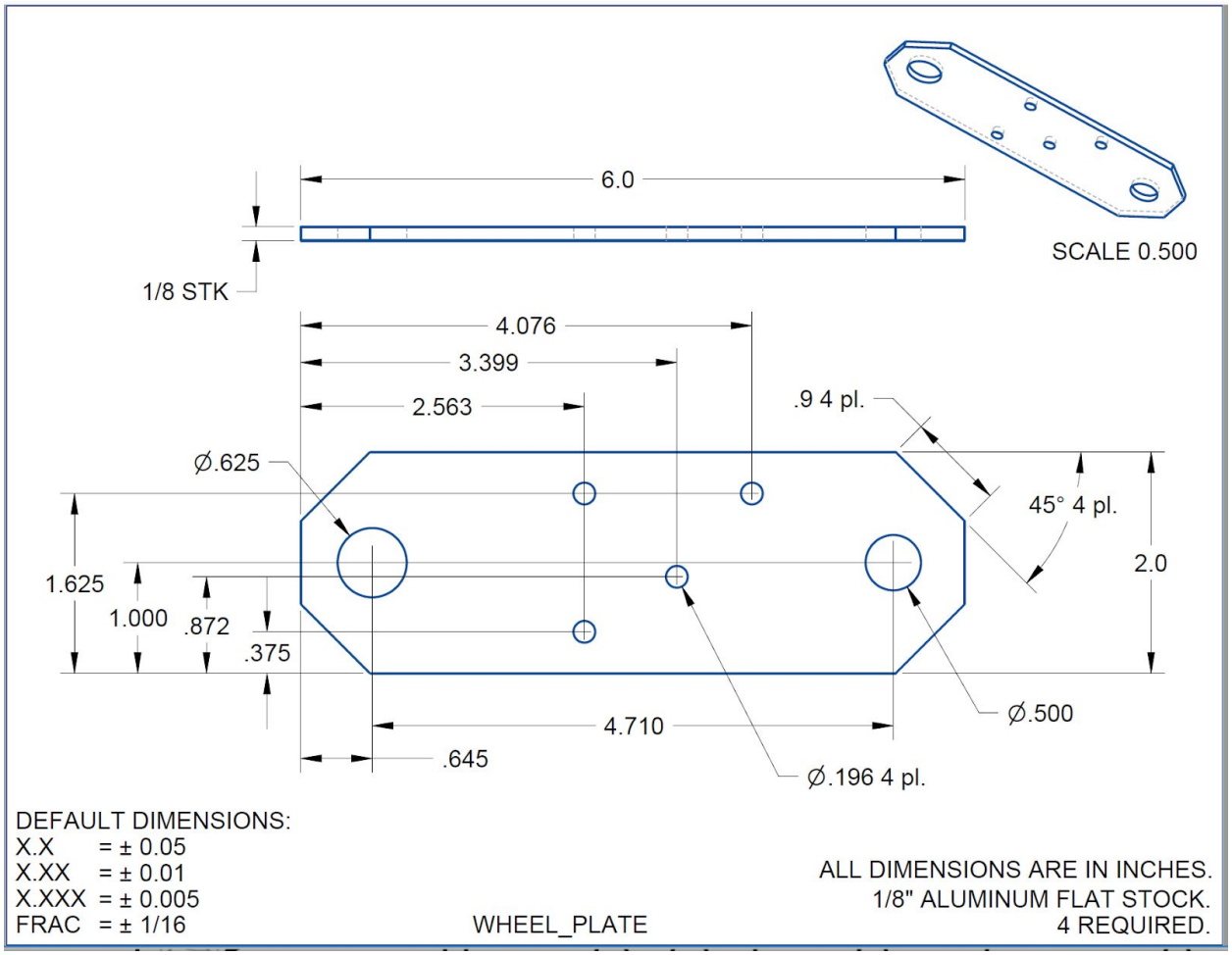
ALL DIMENSIONS ARE IN INCHES.
1" ALUMINUM 90° ANGLE STOCK.
2 REQUIRED.

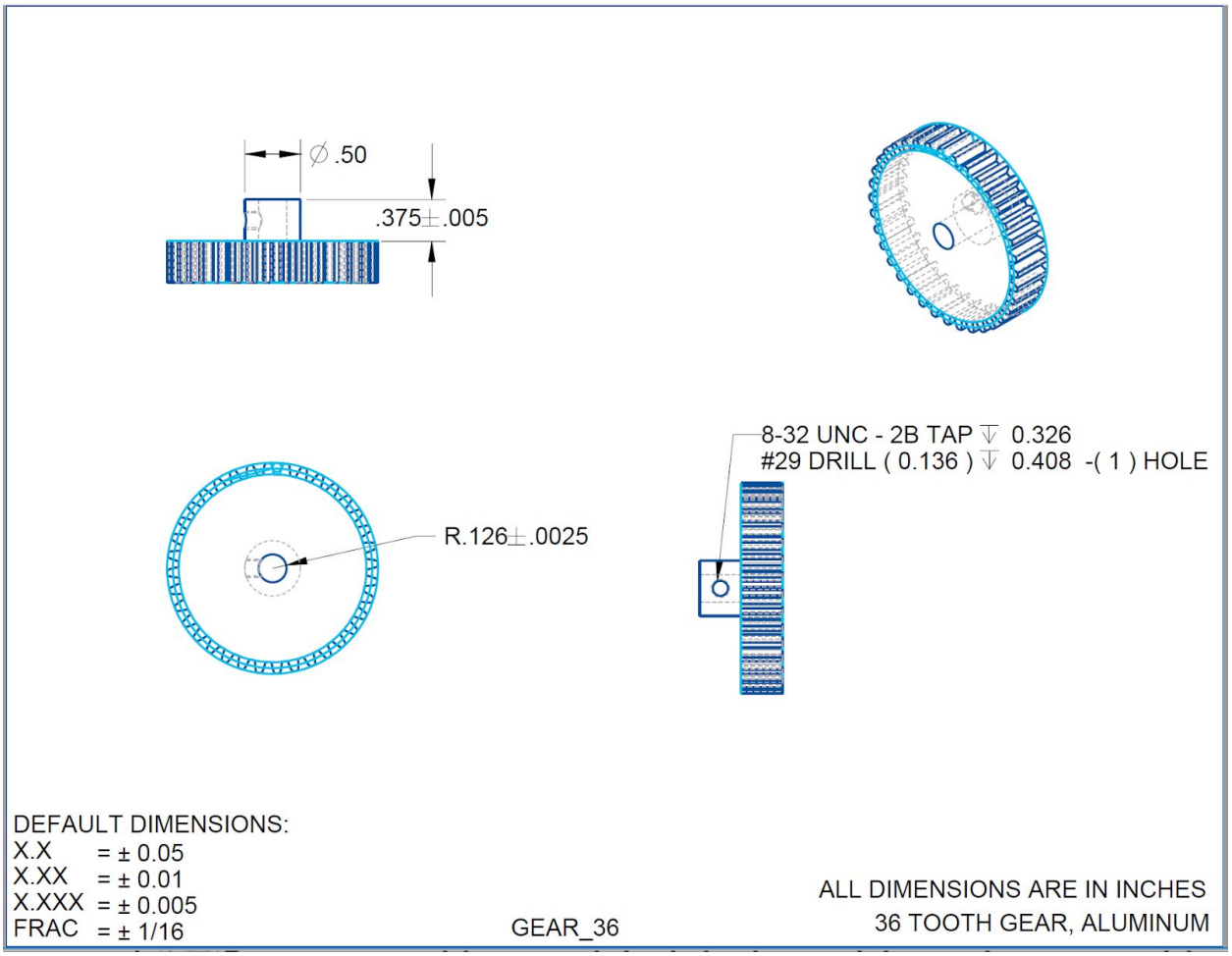


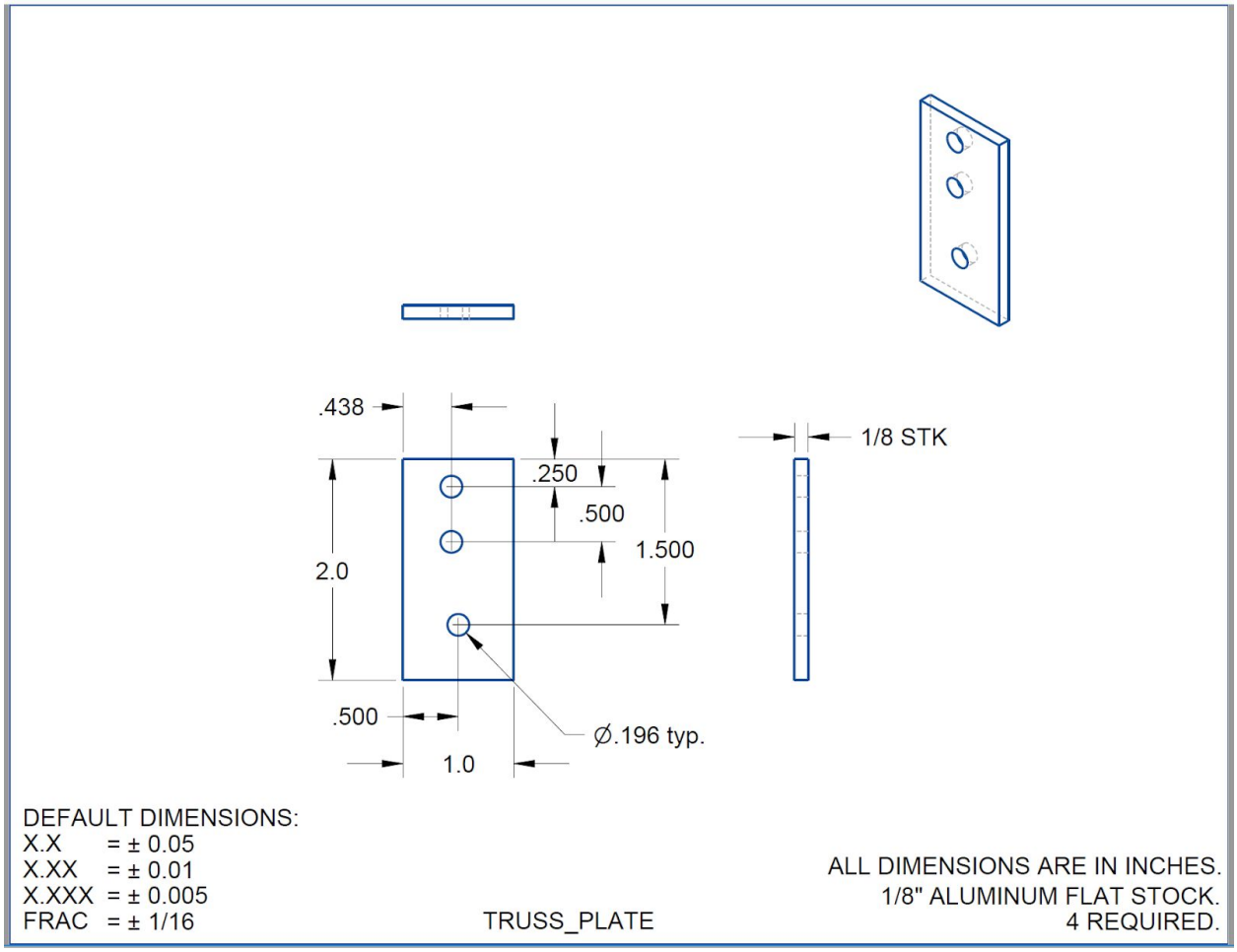
DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16

U_BAR_2INCHES

ALL DIMENSIONS ARE IN INCHES.
2" ALUMINUM U BAR STOCK.
4 REQUIRED.


6.0
1/8 STK
SCALE 0.500
4.076
3.399
2.563
.9 4 pl.
Ø.625
45° 4 pl.
2.0
1.625
1.000
.872
.375
4.710
.645
Ø.500
Ø.196 4 pl.
DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16
WHEEL_PLATE
ALL DIMENSIONS ARE IN INCHES.
1/8" ALUMINUM FLAT STOCK.
4 REQUIRED.

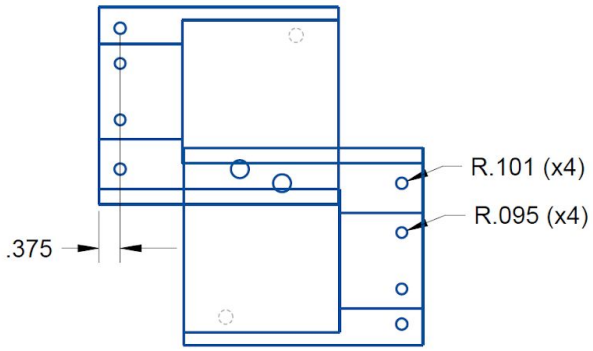

∅ .50
.375±.005
R.126±.0025
8-32 UNC - 2B TAP ↧ 0.326
#29 DRILL (0.136) ↧ 0.408 -(1) HOLE
DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16
GEAR_36
ALL DIMENSIONS ARE IN INCHES
36 TOOTH GEAR, ALUMINUM

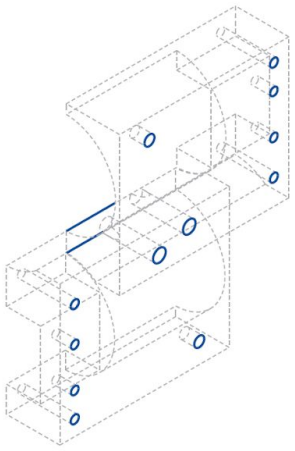


DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16

TRUSS_PLATE

ALL DIMENSIONS ARE IN INCHES.
1/8" ALUMINUM FLAT STOCK.
4 REQUIRED.

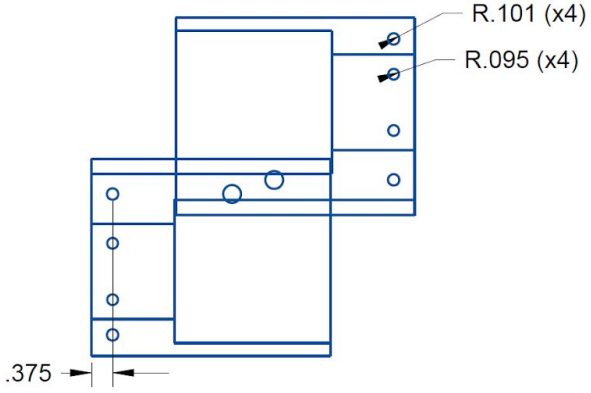


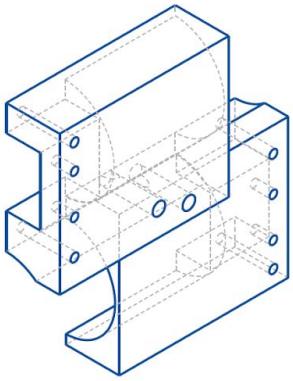


DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16

MOTORMOUNT_LOWER

ALL DIMENSIONS ARE IN INCHES.
3-D PRINTED PART.
1 REQUIRED.

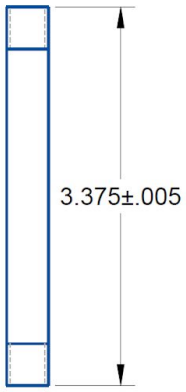


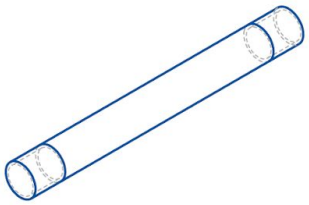


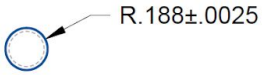
DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16

ALL DIMENSIONS ARE IN INCHES.
3-D PRINTED PART.
1 REQUIRED.

MOTORMOUNT_UPPER



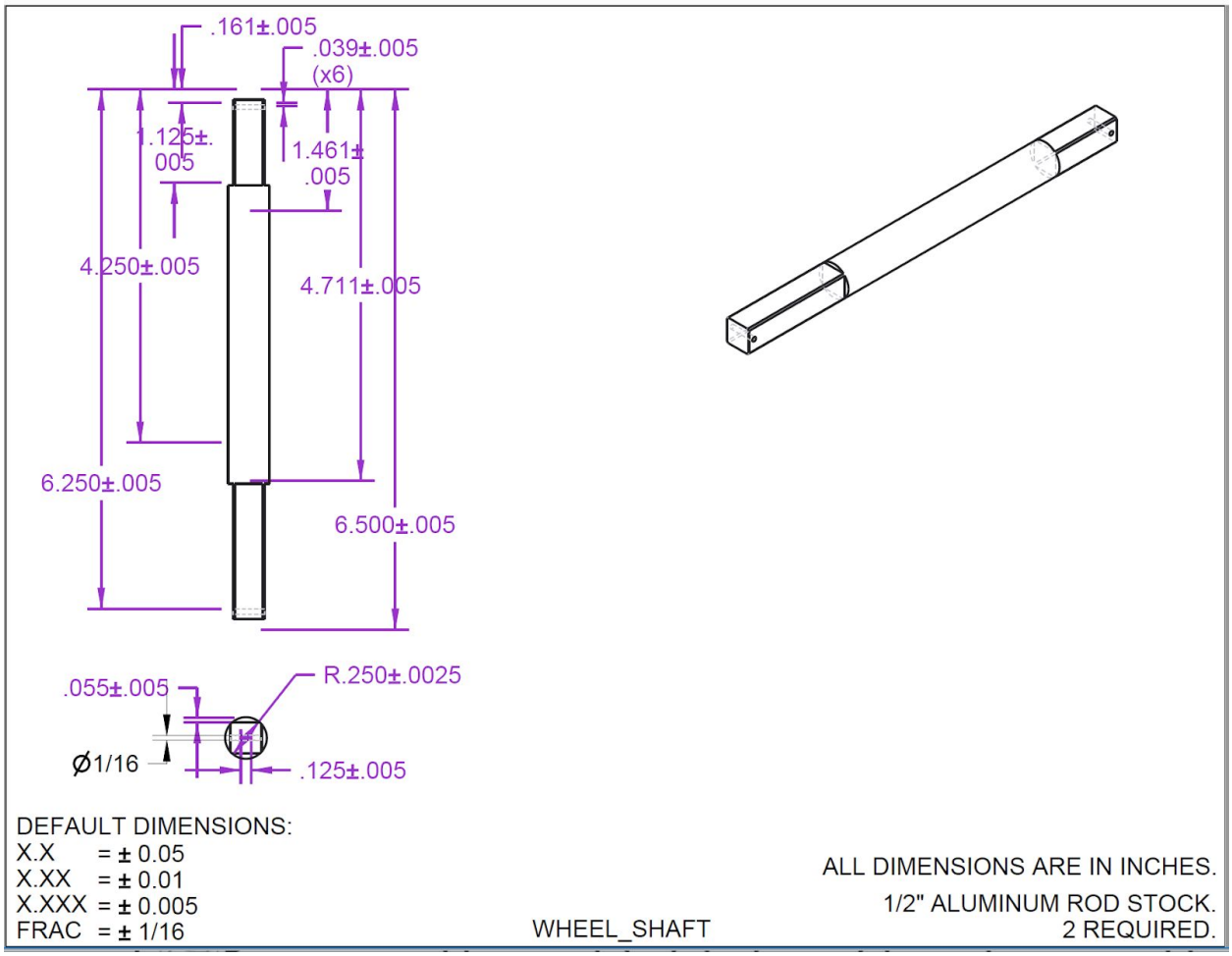


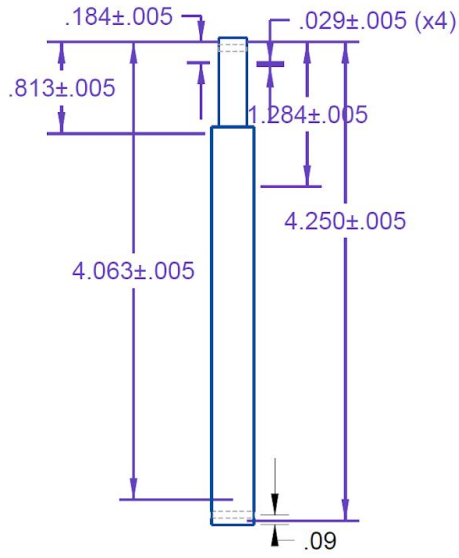


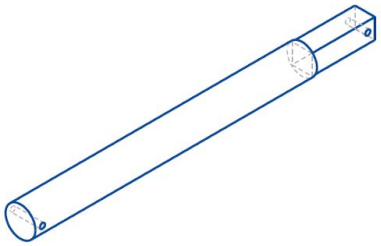
DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16

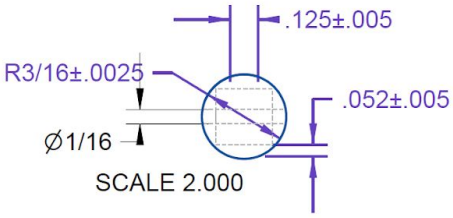
PULLEY_SHAFT

ALL DIMENSIONS ARE IN INCHES.
1/4" ALUMINUM BAR STOCK.
2 REQUIRED.


.161±.005
.039±.005
(x6)
1.125±.
005
1.461±
.005
4.250±.005
4.711±.005
6.250±.005
6.500±.005
.055±.005
R.250±.0025
Ø1/16
.125±.005
DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16
ALL DIMENSIONS ARE IN INCHES.
1/2" ALUMINUM ROD STOCK.
2 REQUIRED.
WHEEL_SHAFT



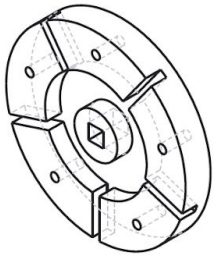


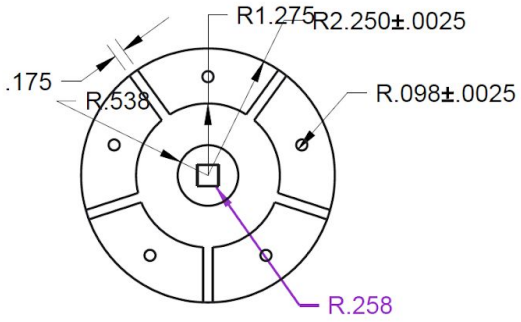


DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16

REAR_PULLEY_SHAFT

ALL DIMENSIONS ARE IN INCHES.
1/2" ALUMINUM ROD STOCK.
2 REQUIRED.

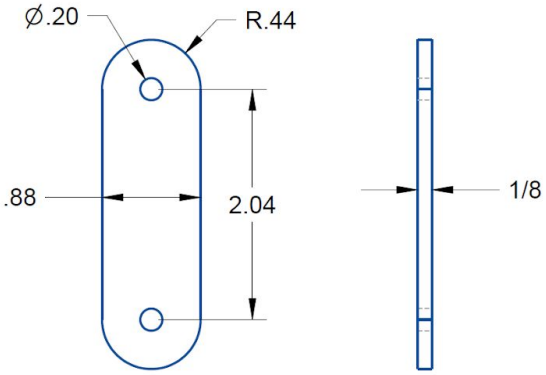


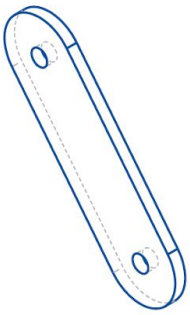


DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16

ALL DIMENSIONS ARE IN INCHES
3-D PRINTED PART.
4 REQUIRED.

WHEEL_BLOCK

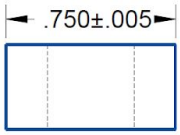


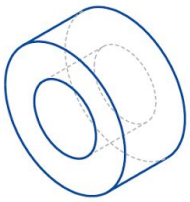


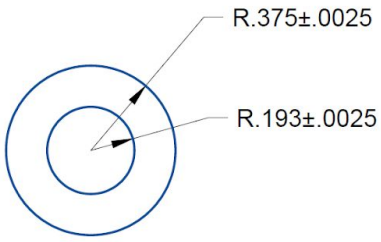
DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16

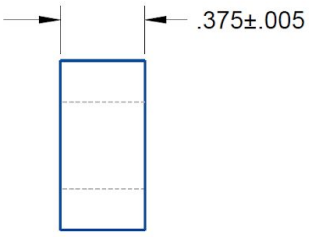
TRUSS

ALL DIMENSIONS ARE IN INCHES.
1/8" ALUMINUM FLAT STOCK.
4 REQUIRED.





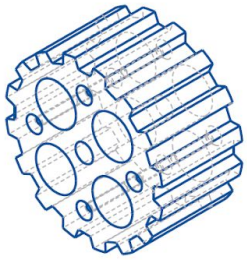


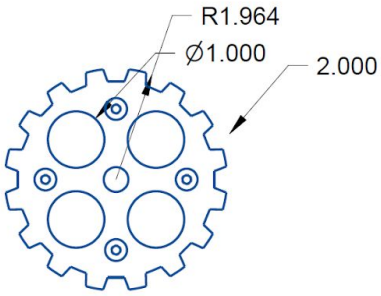


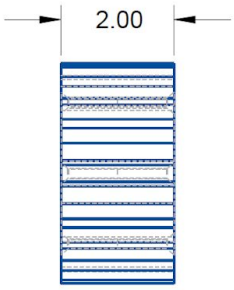
DEFAULT_DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16

TENSIONER_SPACER

ALL DIMENSIONS ARE IN INCHES
3-D PRINTED PART.
8 REQUIRED.



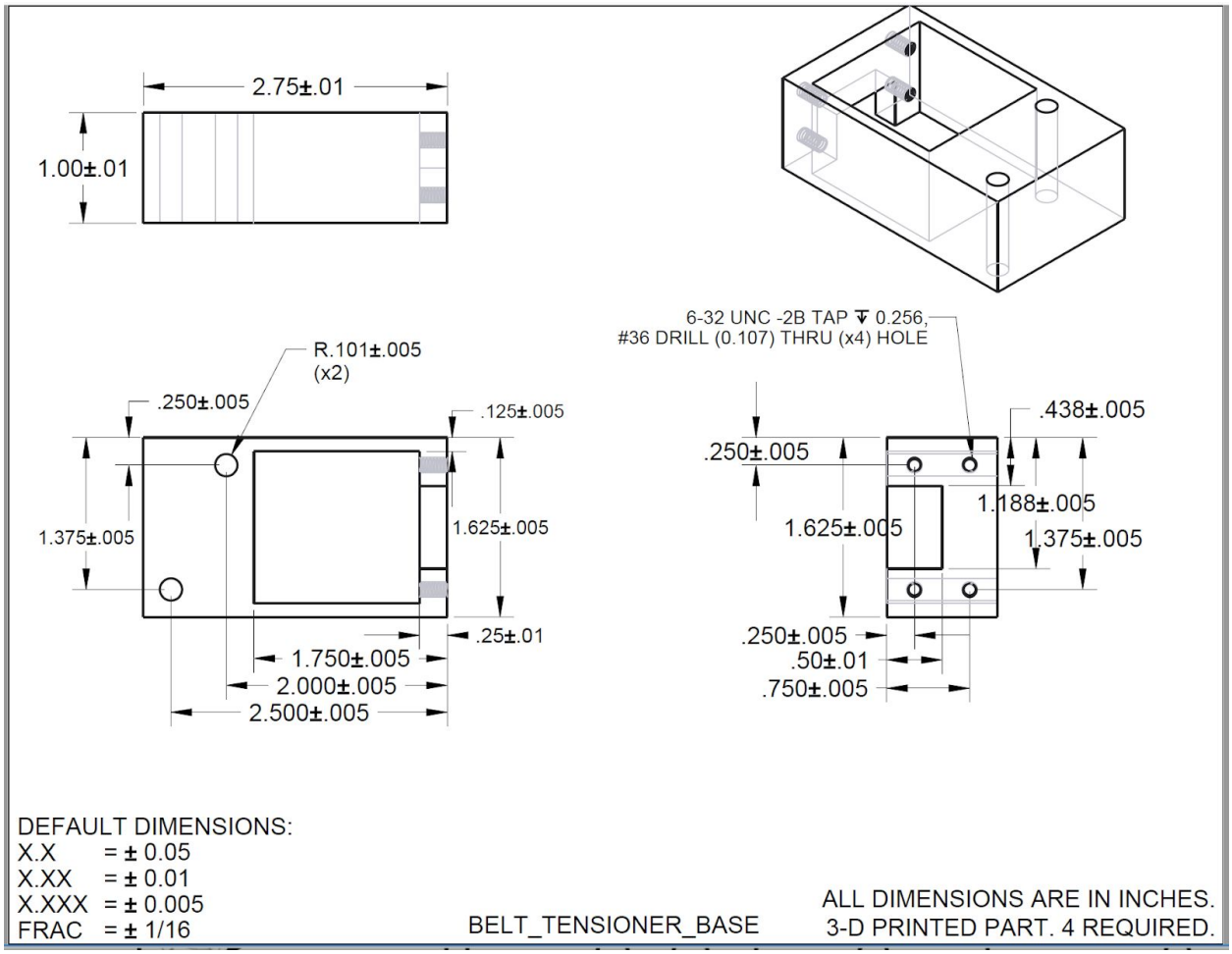




DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16

TREAD_PULLEY

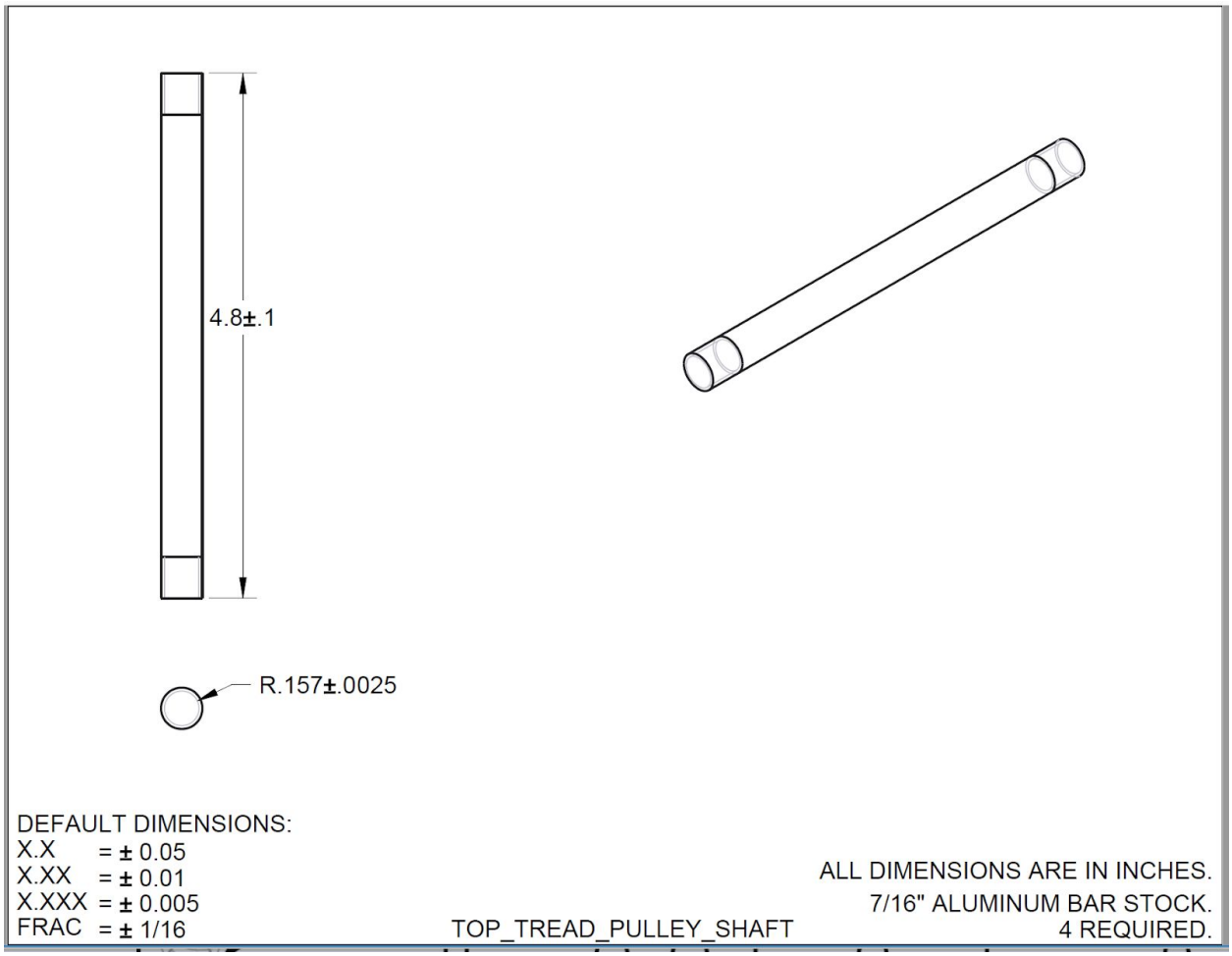
ALL DIMENSIONS ARE IN INCHES.
3D PRINTED PART. 6 REQUIRED.

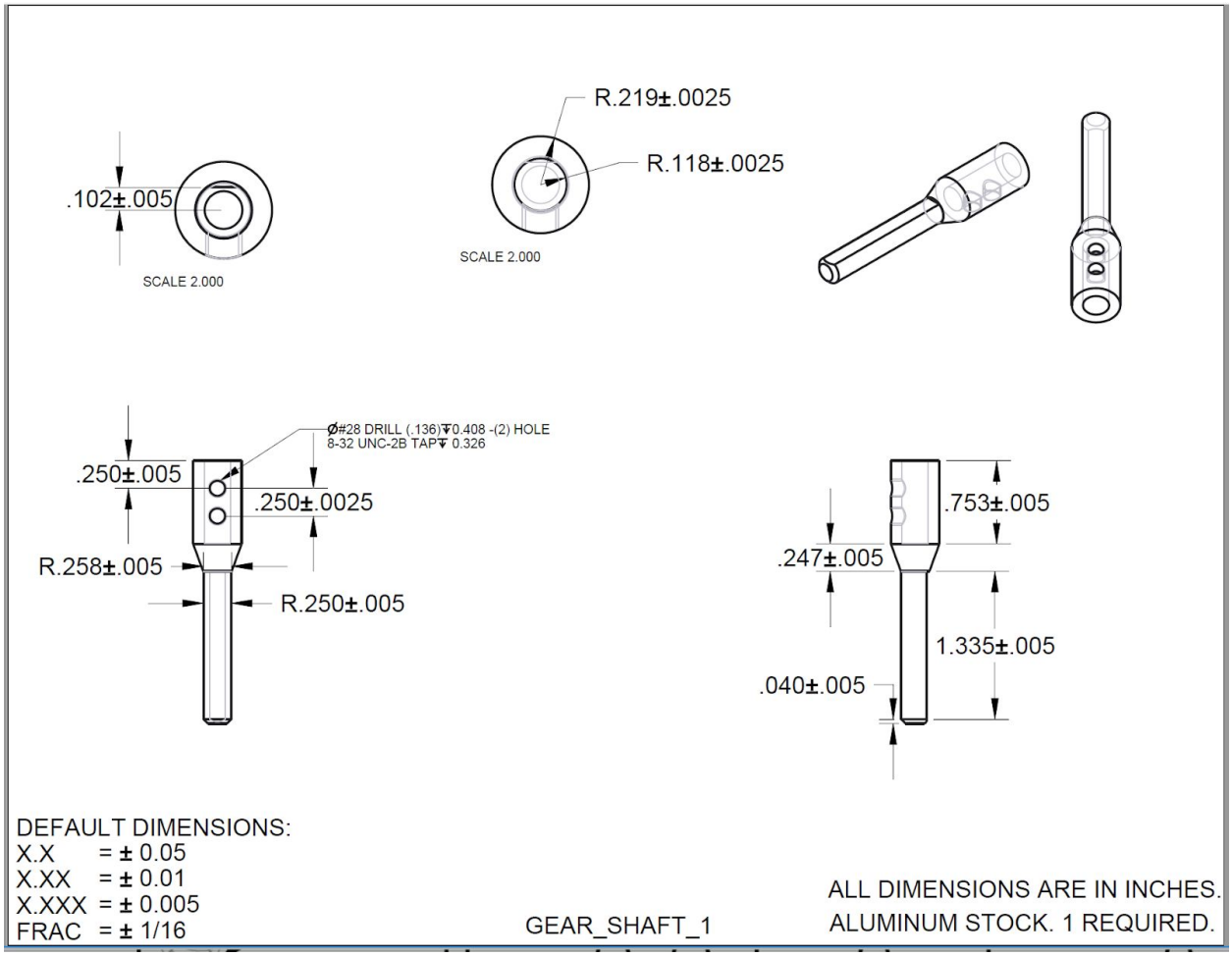


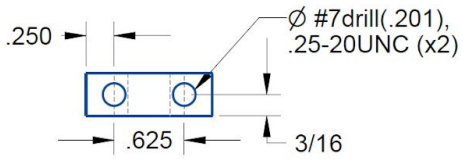
DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16

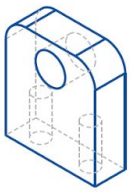
BELT_TENSIONER_BASE

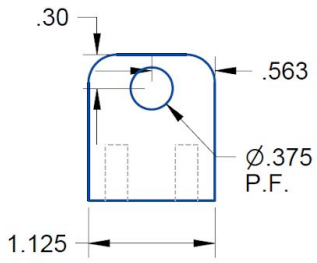
ALL DIMENSIONS ARE IN INCHES.
3-D PRINTED PART. 4 REQUIRED.

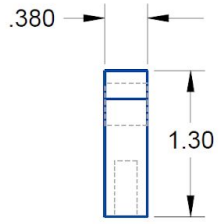

4.8±.1
R.157±.0025
DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16
TOP_TREAD_PULLEY_SHAFT
ALL DIMENSIONS ARE IN INCHES.
7/16" ALUMINUM BAR STOCK.
4 REQUIRED.


.102±.005
SCALE 2.000
R.219±.0025
R.118±.0025
SCALE 2.000
Ø#28 DRILL (.136)↧0.408 -(2) HOLE
8-32 UNC-2B TAP↧ 0.326
.250±.005
.250±.0025
R.258±.005
R.250±.005
.753±.005
.247±.005
1.335±.005
.040±.005
DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16
GEAR_SHAFT_1
ALL DIMENSIONS ARE IN INCHES.
ALUMINUM STOCK. 1 REQUIRED.





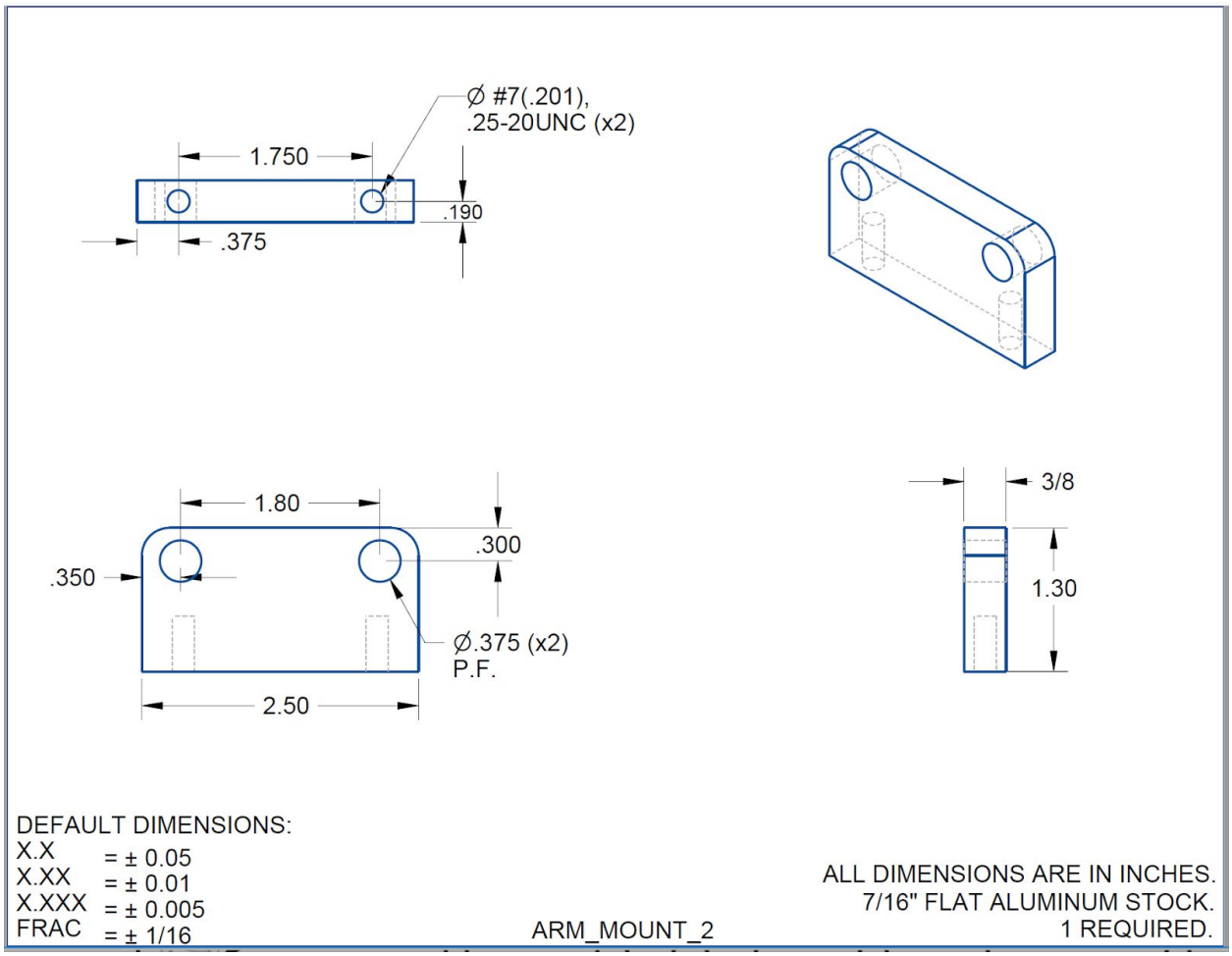




DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16

ARM_MOUNT

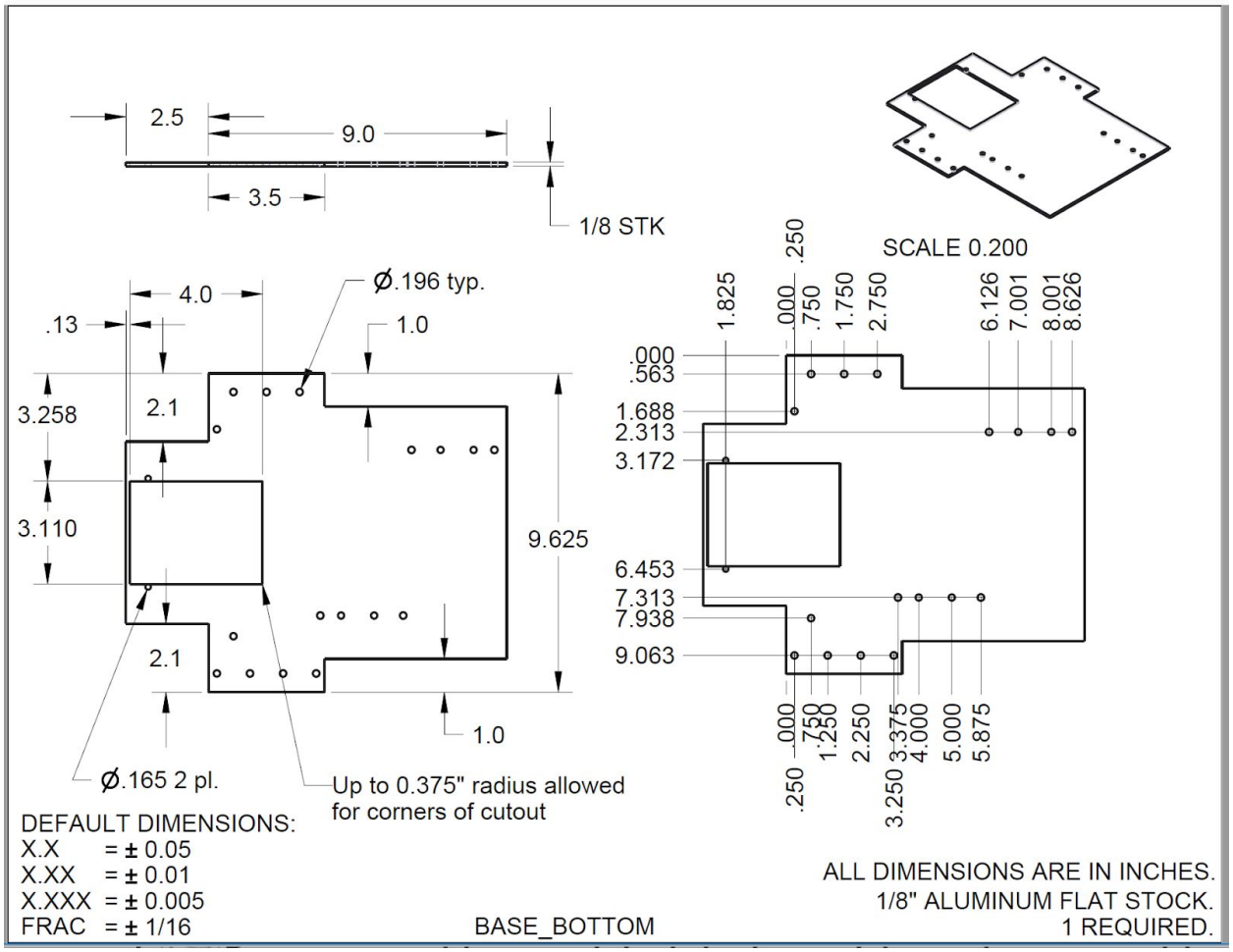
ALL DIMENSIONS ARE IN INCHES.
7/16" ALUMINUM FLAT STOCK.
1 REQUIRED.

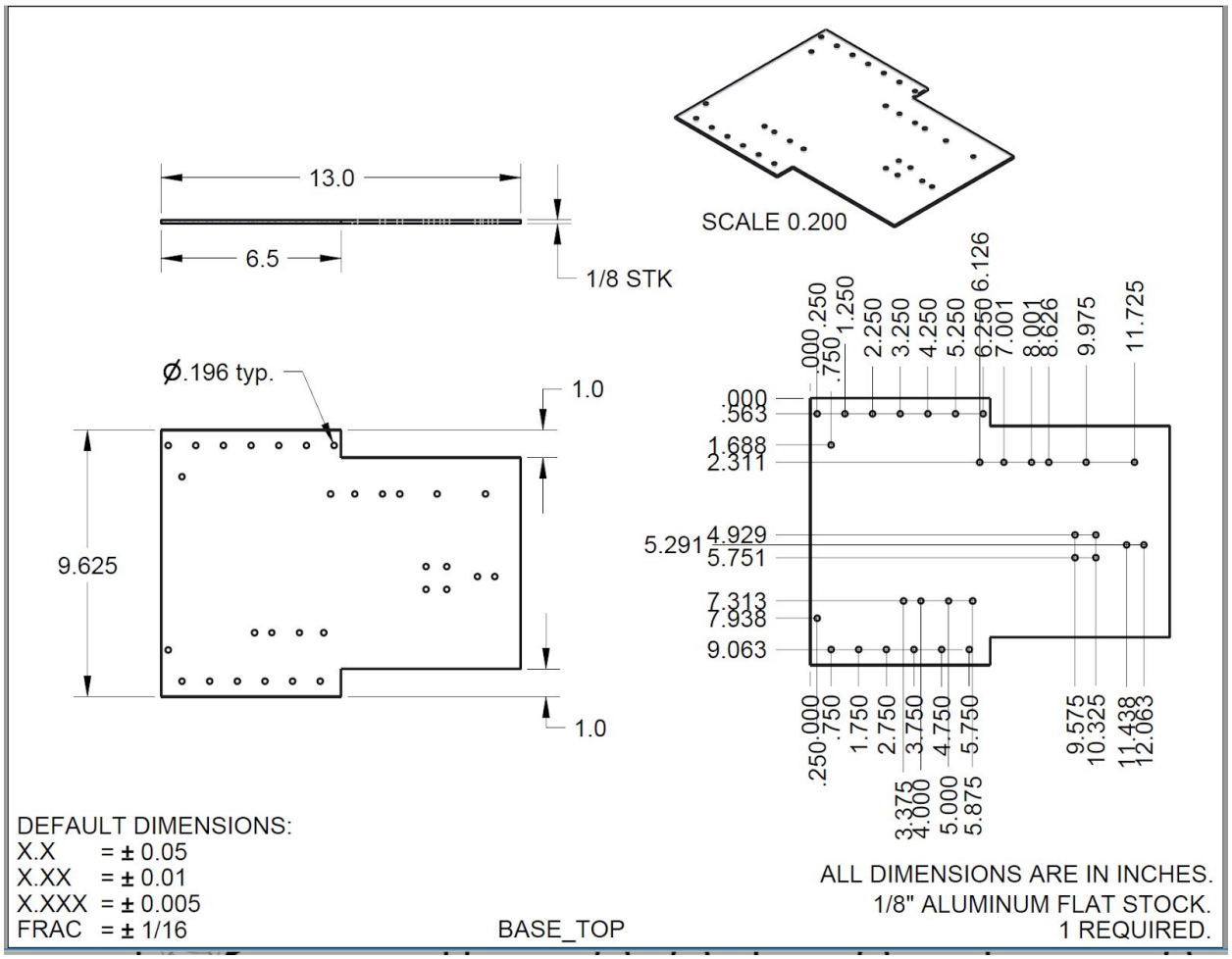


DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16

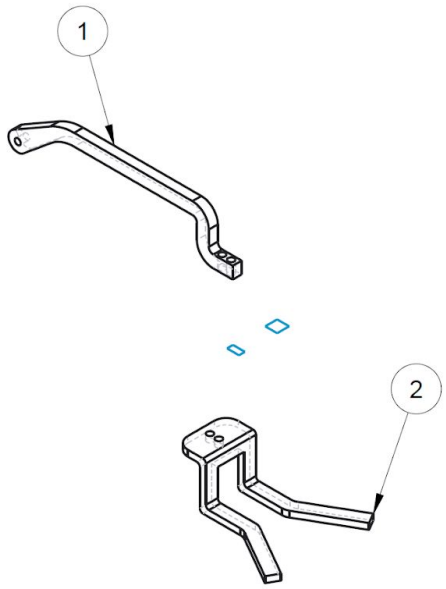
ARM_MOUNT_2

ALL DIMENSIONS ARE IN INCHES.
7/16" FLAT ALUMINUM STOCK.
1 REQUIRED.


2.5
9.0
3.5
1/8 STK
SCALE 0.200
4.0
.13
Ø.196 typ.
1.0
3.258
2.1
3.110
9.625
2.1
1.0
Ø.165 2 pl.
Up to 0.375" radius allowed
for corners of cutout
1.825
.000
.250
.750
1.750
2.750
6.126
7.001
8.001
8.626
.000
.563
1.688
2.313
3.172
6.453
7.313
7.938
9.063
.250
.000
.750
1.250
2.250
3.250
3.375
4.000
5.000
5.875
DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16
BASE_BOTTOM
ALL DIMENSIONS ARE IN INCHES.
1/8" ALUMINUM FLAT STOCK.
1 REQUIRED.


13.0
6.5
1/8 STK
SCALE 0.200
Ø.196 typ.
1.0
9.625
1.0
.000 .250 .750 1.250 2.250 3.250 4.250 5.250 6.126 6.250 7.001 8.001 8.626 9.975 11.725
.000
.563
1.688
2.311
5.291
4.929
5.751
7.313
7.938
9.063
.250 .000 .750 1.750 2.750 3.375 3.750 4.000 4.750 5.000 5.750 5.875 9.575 10.325 11.438 12.063
DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16
BASE_TOP
ALL DIMENSIONS ARE IN INCHES.
1/8" ALUMINUM FLAT STOCK.
1 REQUIRED.

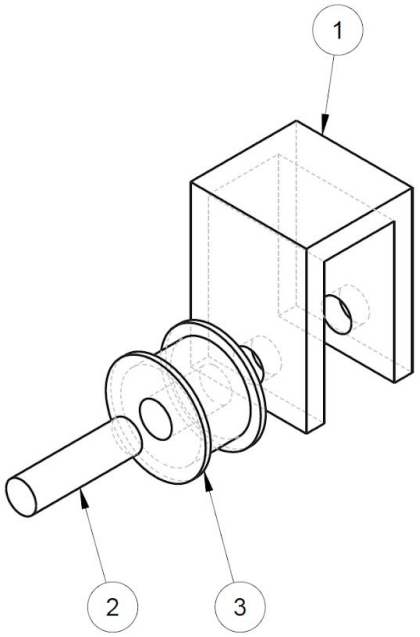
ITEM NO	QTY	DESCRIPTION
1	1	ARM_TEST_RENDER_.PRT
2	1	FORK_CONCEPT_.PRT

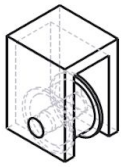


ARM_CONCEPT

1 REQUIRED.

ITEM NO	QTY	DESCRIPTION
1	1	BELT_TENSIONER_ARM_.PRT
2	1	BELT_TENSIONER_AXLE_.PRT
3	1	BELT_TENSIONER_PULLEY_.PRT



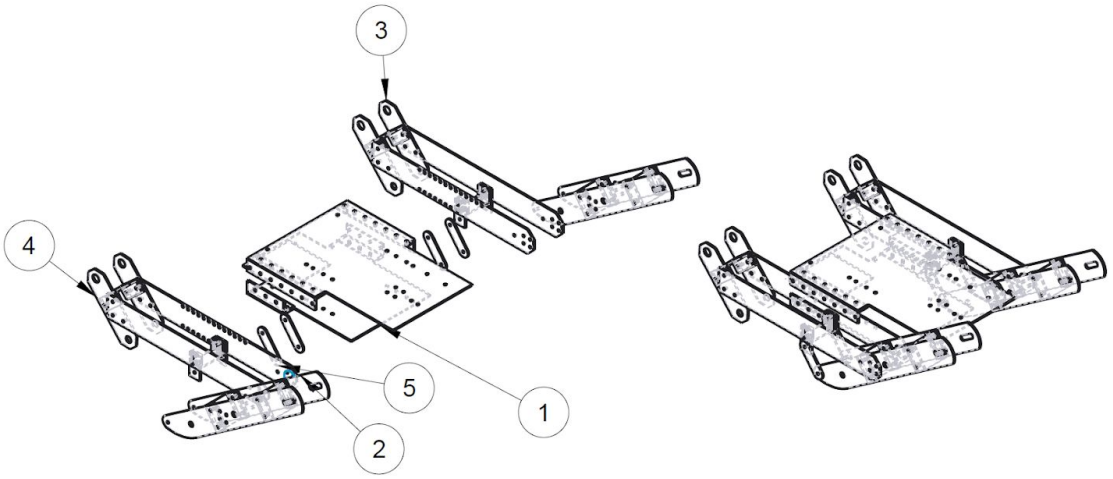


SCALE 0.500

BELT_TENSIONER_ASSEMBLY

4 REQUIRED.

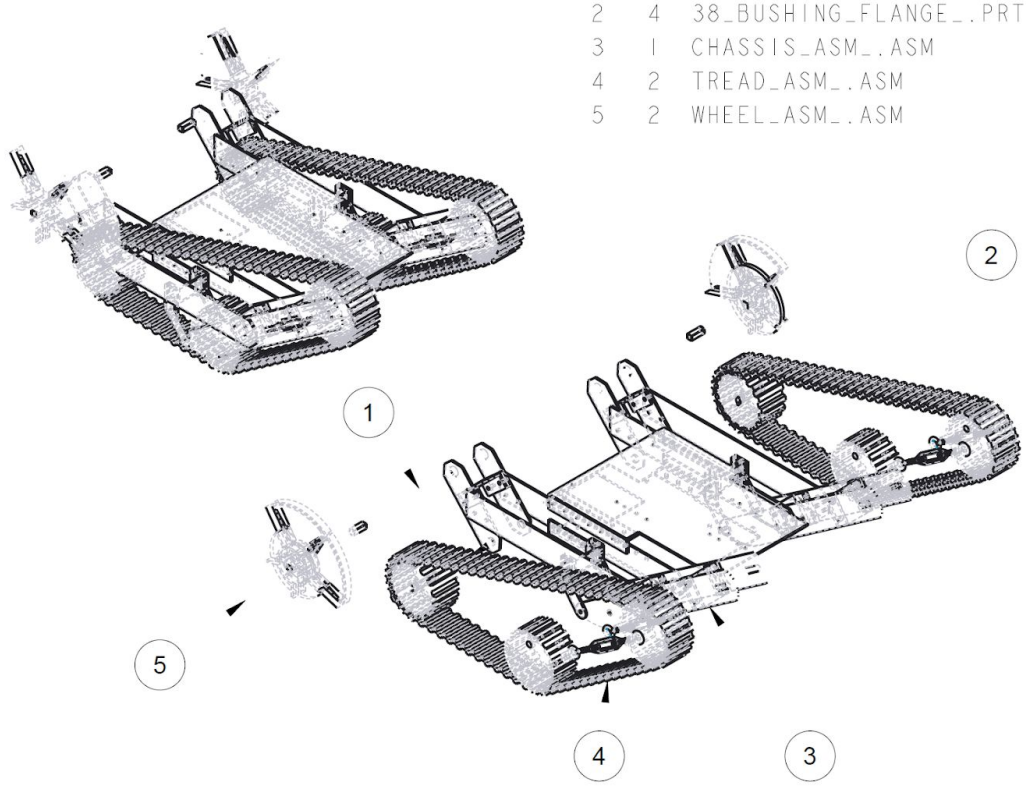
ITEM NO	QTY	DESCRIPTION
1	1	ROBOT_BASE_.ASM
2	2	TREAD_ARM_ASM_.ASM
3	1	TREAD_BASE_MIRROR_ASM_.ASM
4	1	TREAD_BASE_SIDE_ASM_.ASM
5	4	TRUSS_.PRT



CHASSIS_ASM

1 REQUIRED.

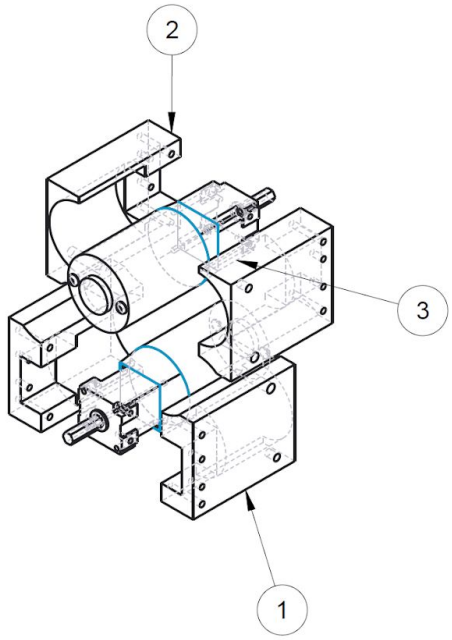
ITEM NO	QTY	DESCRIPTION
1	4	12_BALL_BEARING_.PRT
2	4	38_BUSHING_FLANGE_.PRT
3	1	CHASSIS_ASM_.ASM
4	2	TREAD_ASM_.ASM
5	2	WHEEL_ASM_.ASM



DRIVE_BASE_ASM

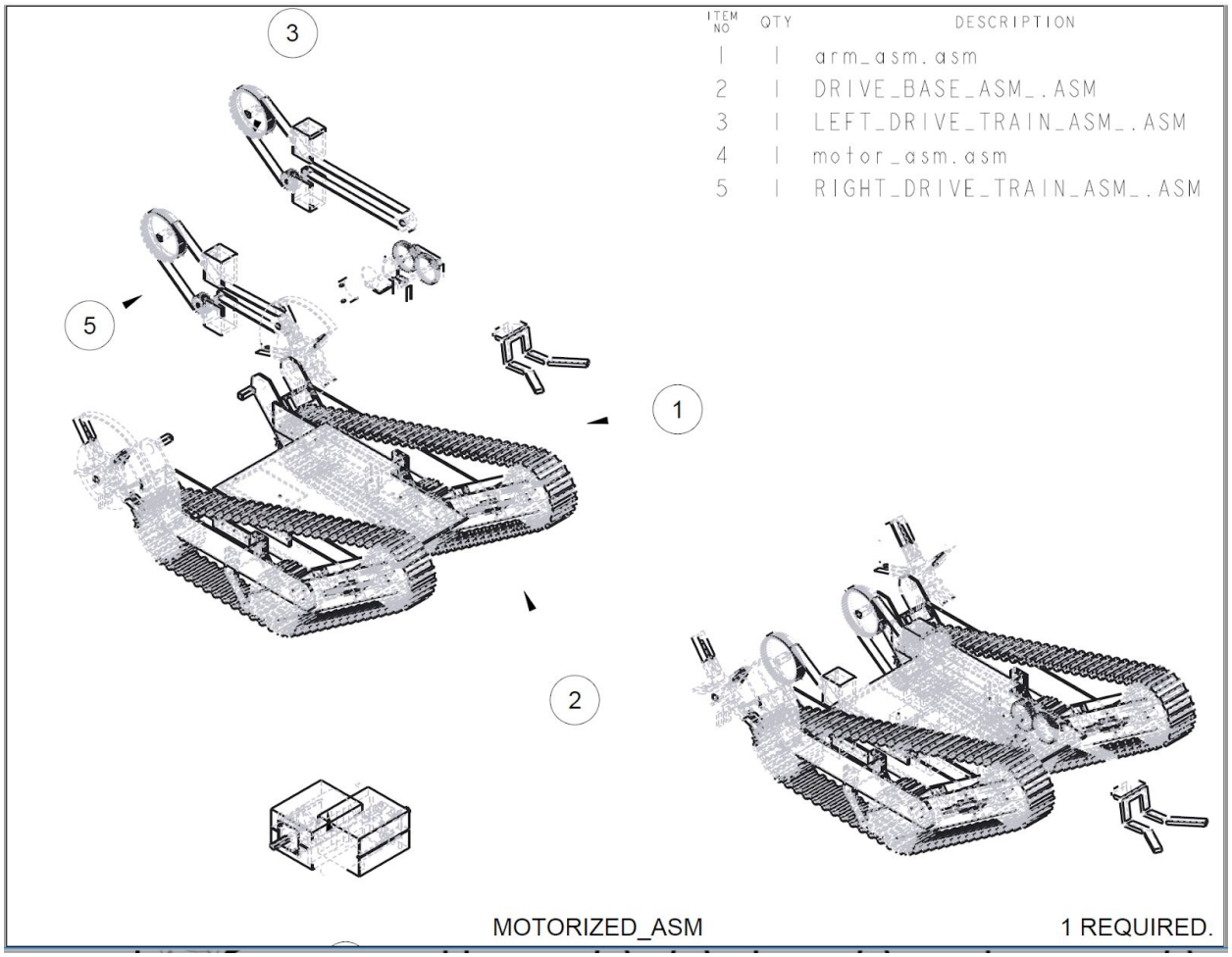
ALL DIMENSIONS ARE IN INCHES.
1 REQUIRED.

ITEM NO	QTY	DESCRIPTION
1	1	MOTORMOUNT_LOWER_.ASM
2	1	MOTORMOUNT_UPPER_.ASM
3	2	VEX_MOTOR__.PRT

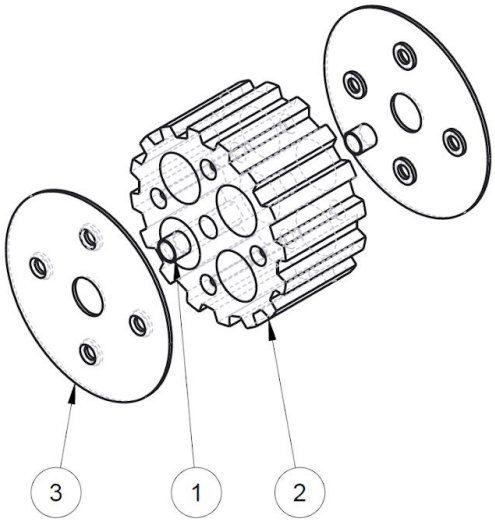


MOTOR_ASM

ALL DIMENSIONS ARE IN INCHES.
1 REQUIRED.


ITEM NO
QTY
DESCRIPTION
1 1 arm_asm.asm
2 1 DRIVE_BASE_ASM_.ASM
3 1 LEFT_DRIVE_TRAIN_ASM_.ASM
4 1 motor_asm.asm
5 1 RIGHT_DRIVE_TRAIN_ASM_.ASM
3
5
1
2
MOTORIZED_ASM
1 REQUIRED.

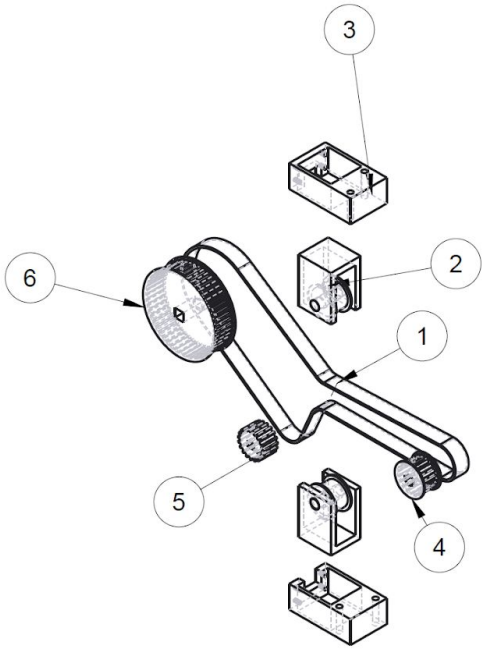
ITEM NO	QTY	DESCRIPTION
1	2	38_BUSHING_.PRT
2	1	TREAD_PULLEY_.PRT
3	2	TREAD_PULLEY_FLANGE_.PRT

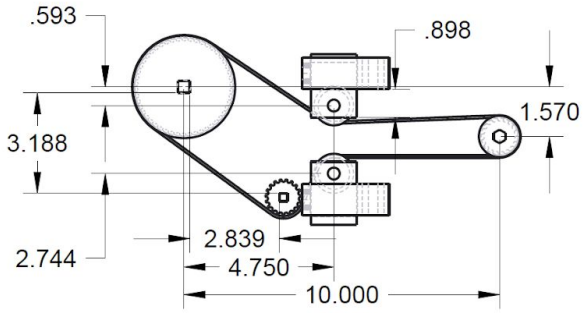


PULLEY_ASM

4 REQUIRED.

ITEM NO	QTY	DESCRIPTION
1	1	BACK_MOTOR_BELT_.PRT
2	2	BELT_TENSIONER_ASSEMBLY_.ASM
3	2	BELT_TENSIONER_BASE_.PRT
4	1	MOTOR_PULLEY_19T_.PRT
5	1	PULLEY_PULLEY_19T_.PRT
6	1	WHEEL_PULLEY_50T_.PRT



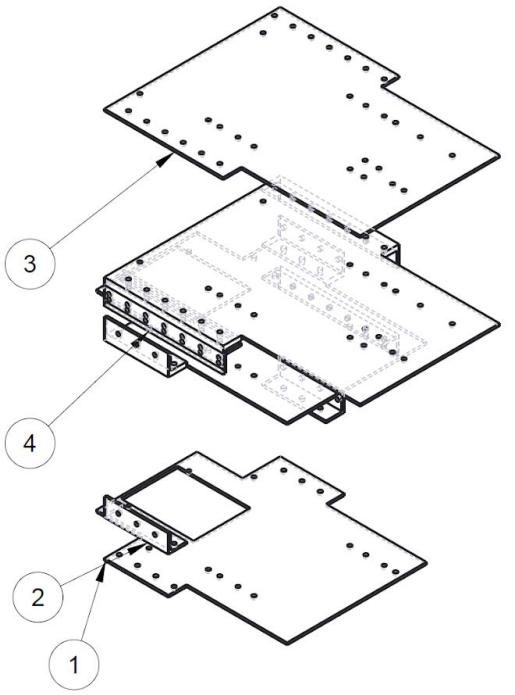


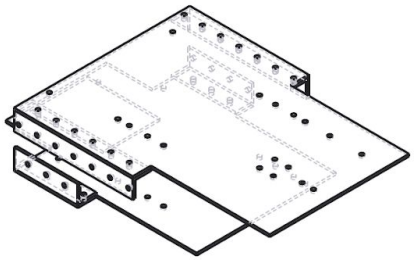
DEFAULT DIMENSIONS:
X.X = ± 0.05
X.XX = ± 0.01
X.XXX = ± 0.005
FRAC = ± 1/16

RIGHT_DRIVE_TRAIN_ASM

ALL DIMENSIONS ARE IN INCHES.
1 REQUIRED.

ITEM NO	QTY	DESCRIPTION
1	1	ROBOT_BASE_BOTTOM_.PRT
2	2	ROBOT_BASE_BOTTOM_LI_.PRT
3	1	ROBOT_BASE_TOP_.PRT
4	2	ROBOT_BASE_TOP_L_.PRT
5	1	ROBOT_BASE__.PRT

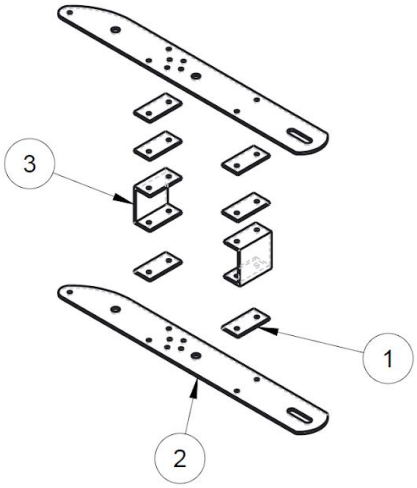


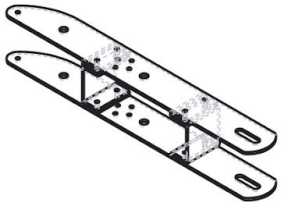


ROBOT_BASE

1 REQUIRED,

ITEM NO	QTY	DESCRIPTION
1	6	2_in_al_spacer.prt
2	2	TREAD_ARM_PLATE_.PRT
3	2	U_BAR_2INCHES_.PRT

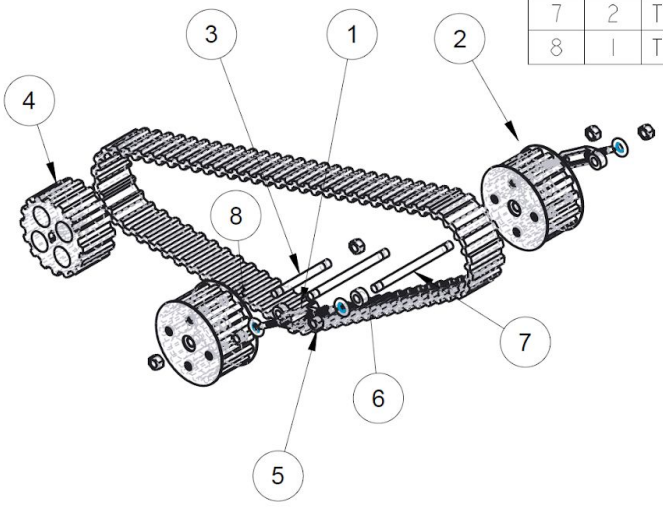


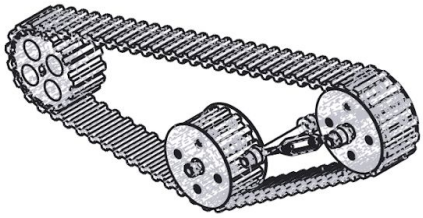


TREAD_ARM_ASM

2 REQUIRED.

ITEM NO	QTY	DESCRIPTION
1	2	3010T170_CAST_STEEL_TURNBUCKLE_.ASM
2	2	PULLEY_ASM_.ASM
3	1	PULLEY_SHAFT_.PRT
4	1	REAR_TREAD_PULLEY_.PRT
5	6	TENSIONER_NUT_.PRT
6	4	TENSIONER_SPACER_.PRT
7	2	TOP_TREAD_PULLEY_SHAFT_.PRT
8	1	TREAD_.PRT

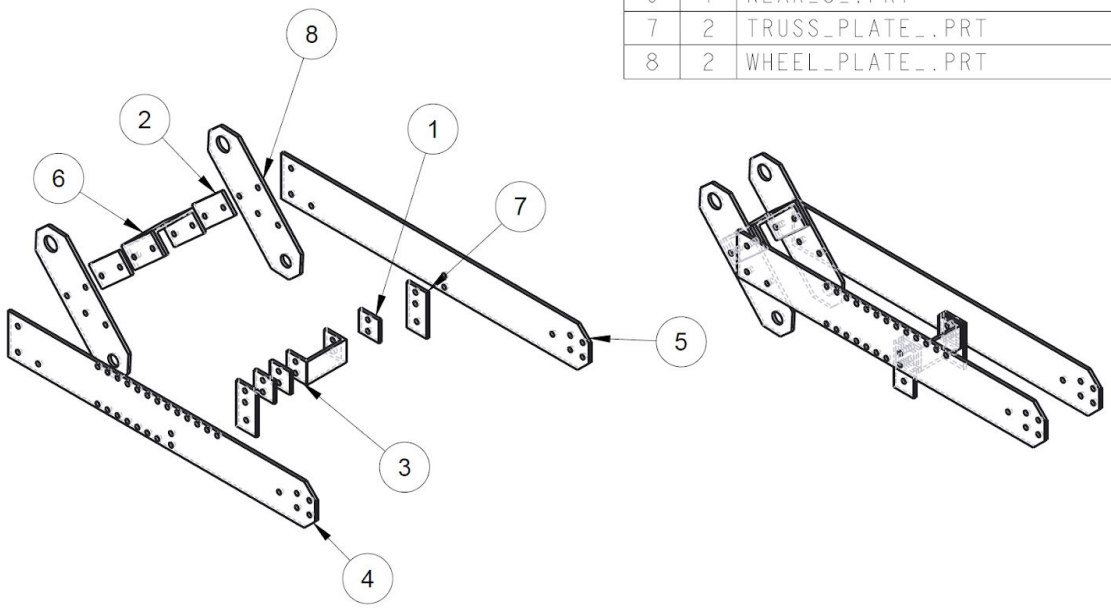




ALL DIMENSIONS ARE IN INCHES.
2 REQUIRED.

TREAD_ASM_

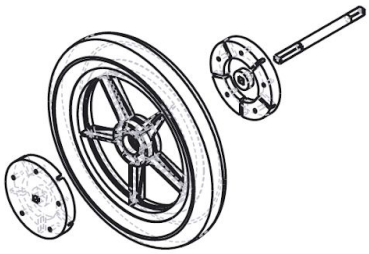
ITEM NO	QTY	DESCRIPTION
1	3	1_in_al_spacer.prt
2	3	2_in_al_spacer.prt
3	1	CENTRAL_U_.PRT
4	1	INSIDE_PLATE_.PRT
5	1	OUTSIDE_PLATE_.PRT
6	1	REAR_U_.PRT
7	2	TRUSS_PLATE_.PRT
8	2	WHEEL_PLATE_.PRT

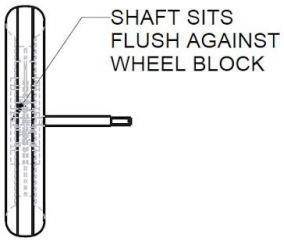


TREAD_BASE_MIRROR_ASM

2 REQUIRED.

ITEM NO	QTY	DESCRIPTION
1	1	AM0970_8IN_WHEEL_.PRT
2	2	WHEEL_BLOCK_.PRT
3	1	WHEEL_SHAFT_.PRT





WHEEL_ASM

2 REQUIRED.

This paper represents our own work in accordance with University regulations.