

6-2011

# Design and Testing of an Automated Semi- Trailer Control System

Conor H. Dodd

*Union College - Schenectady, NY*


Follow this and additional works at: <https://digitalworks.union.edu/theses>



Part of the [Mechanical Engineering Commons](#), and the [Transportation Engineering Commons](#)

---

## Recommended Citation

Dodd, Conor H., "Design and Testing of an Automated Semi- Trailer Control System" (2011). *Honors Theses*. 967.  
<https://digitalworks.union.edu/theses/967>

This Open Access is brought to you for free and open access by the Student Work at Union | Digital Works. It has been accepted for inclusion in Honors Theses by an authorized administrator of Union | Digital Works. For more information, please contact [digitalworks@union.edu](mailto:digitalworks@union.edu).

# MER-498

## Design and Testing of an Automated Semi-Trailer Control System



Conor Dodd

Mechanical Engineering, Advisor: Prof. Keat

MER-498

## Contents

1. Introduction.....	2
2. The Proposed System.....	4
2.1 Requirements.....	4
2.2 Binding.....	7
2.3 Project Schedule.....	11
3. Detailed Design.....	12
3.1: The Tractor.....	12
3.2: The Bogie.....	13
3.3: The Trailer.....	16
3.4 The Electrical System.....	19
3.5 Programming.....	27
4. Summary.....	31

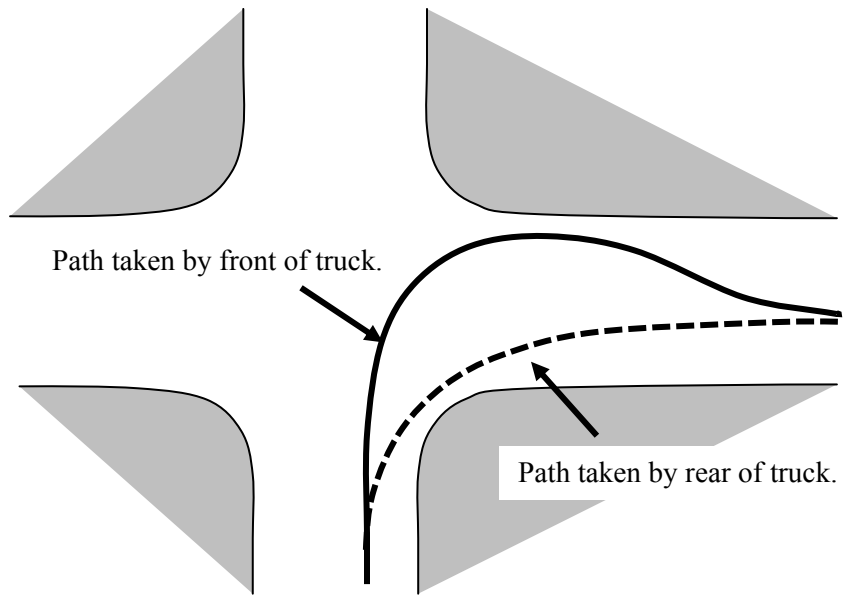
## Appendices

A: References and Acknowledgments.....	32
B: Budget and Sourcing.....	33
C: MATLAB and C++ Code.....	34
C: Detailed Drawings.....	35

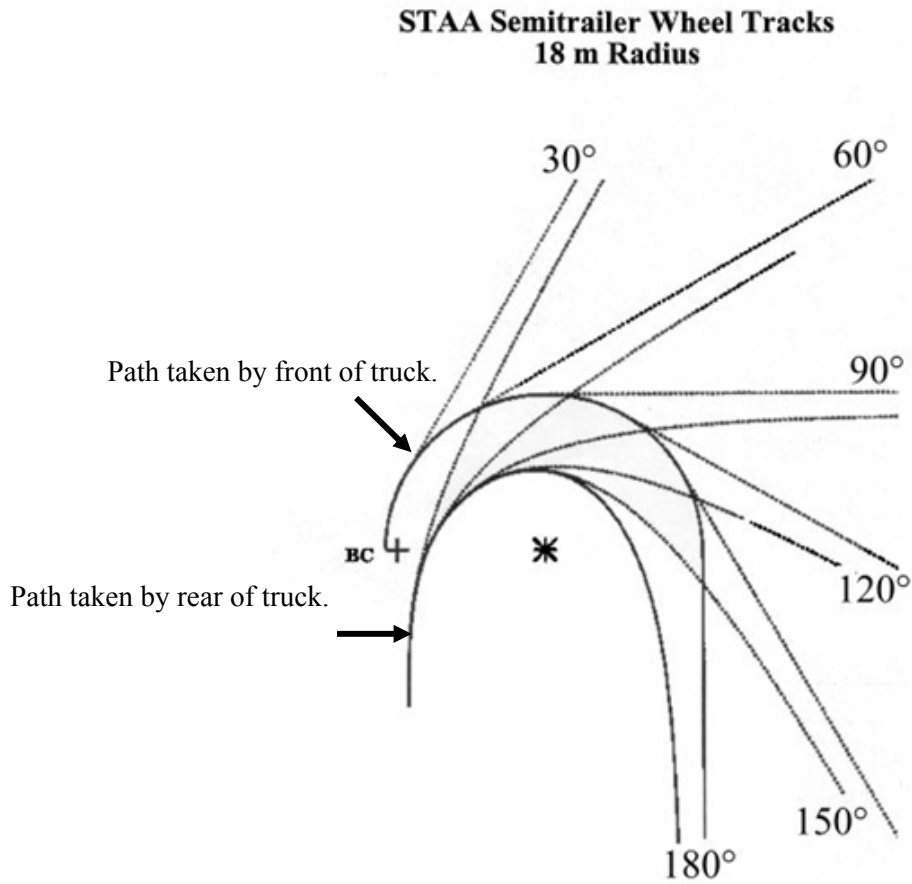
# **1. Introduction: Addressing a Problem in the Trucking Industry**

Growing up in Maine, the trucking industry is an ever-present part of life. With a very sparse, lightly used rail system, almost all of the raw materials and goods that enter and leave the state do so by truck. Truckers in Maine are constantly challenged by Maine's roads, which can often be difficult or even treacherous to navigate. The rough logging roads near Jackman or Millinocket, or the sharp ninety degree turns near the Portland oil docks, can ruin a trucker's day, or even worse, cause an accident. For example, there is a tiny four-way intersection on Broadway in South Portland, where full-length oil semi-trucks have to turn ninety degrees as they leave the massive oil storage tanks at the docks. Broadway is a very busy road; as soon as the trucks get the signal to turn, there are lines of cars stopped at the light. The trucks have to turn wide, right into the stopped line of cars. This means the cars then have to coordinate and back up to allow the truck to turn. After much confusion and after the light has turned green, then red, then green again about ten times with the trailer completely cutting off the intersection, the truck can move on. The same thing happens along the backroads of Millinocket. When trucks come off the logging roads they usually have to turn ninety degrees onto the main road. This is tricky to do at best, and dangerous at worst. The roads usually aren't wide enough for the truck to turn onto them without going over the other side. If the logging companies don't build pulloffs on the side of the road for the trucks to run through, then eventually a trucker is going to try to turn on to the main road and end up in the ditch on the other side, with the trailer completely blocking the way until the truck can be moved.

The cause of all of these problems is the fact that the truck must turn wide: the cab must make a turn much wider than necessary in order to keep the trailer from going over the curb. The reason for this lies in the wheels of the semi-trailer. The assemblage of wheels is often called a "bogie". The term comes from the rail industry, as almost all train cars and engines roll on bogies. The problem with a truck is that the rear bogie on the trailer is fixed, the wheels always point toward the truck. So when the truck makes a turn, the bogie on the trailer follows a different path. A standard turn of a semi truck is shown in Figure 1.1. These turning curves are also often published for trailers of a certain length. Figure 1.2 shows the turning curves for a 42' trailer.



**Figure 1.1: Path Diagram for a 90 Degree Turn.**



**Figure 1.2: Path Diagram for Multiple Turns.**

My project is to design a system to eliminate the need for a semi-truck to make wide turns. I will do this by implementing a system of self-steering trailer bogies. The bogies will be able to steer themselves automatically and will follow the path the cab takes, so the driver can drive as if there is nothing behind him. There are also many other capabilities a self-steering system could bring to a truck company or driver. I will design my system so that the rear bogies think independently, with no central computer system. That means a trucker can hook as many bogies, and thus trailers, up as is legal and they will all steer themselves. So a truck with three trailers will still turn as sharply as a cab with nothing behind it at all. Other possible benefits could include having the bogies steer themselves to avoid accidents, correct for skidding and jackknifing, allow for easy reversing, and reduce braking distance.

To get some feedback on the design, I posted the basics of my idea on “The Trucker’s Report”, a forum for truckers and trucking companies. Responses to the idea have been quickly filling up the pages. The general opinion is that the idea, especially if I could make it work with multiple trailers, could easily be a very potent force in the industry if I designed it for at least a ten-year life span. The increase in maneuverability could open up a significant number of new routes that could save time and fuel. And the ability to haul multiple trailers could vastly increase the amount of material the truck could haul for the fuel. Already in some stated road trains, trucks with multiple trailers, have been having a positive impact. These trains, however need to turn even wider to make turns, and so are even further limited as to where they can drive safely. The combination of increased maneuverability with increased capacity could really make a difference if it works.

For the senior project I will build a scale model robot of a steerable bogie system. The robot will be designed specifically to test the validity of the steering concept, to see whether or not it could work when implemented on a full-scale truck. The model will be remote controlled. An operator will control the tractor, and the bogies will attempt to steer on a “virtual track” just like what the full scale system would do.

As the ultimate goal of this project is to prove or disprove the validity of the whole self-steering bogie concept, both the physics and the programming of the machine have to be tested. The programming is the easy part; if I just wanted to test that I could simply simulate it. But for the case of this project, it is not just the programming that will determine whether or not it can work. There are many physical unknowns that need to be tested that, if they do not work, then the project cannot succeed no matter how good the programming is.

## **2: The Proposed System**

### **2.1: Requirements**

From the suggestions posted online I developed three critical design requirements for a system like this to be successful in the real world.

- Must significantly reduce the required width for a truck to maneuver. (Over 50% reduction for all common turning radii).
- Must be easily adaptable to any trailer configuration, and can be retrofitted to existing trailers. Must be programmable and adjustable to different road and driving conditions.
- Must be cost-effective over a 15 year service life.

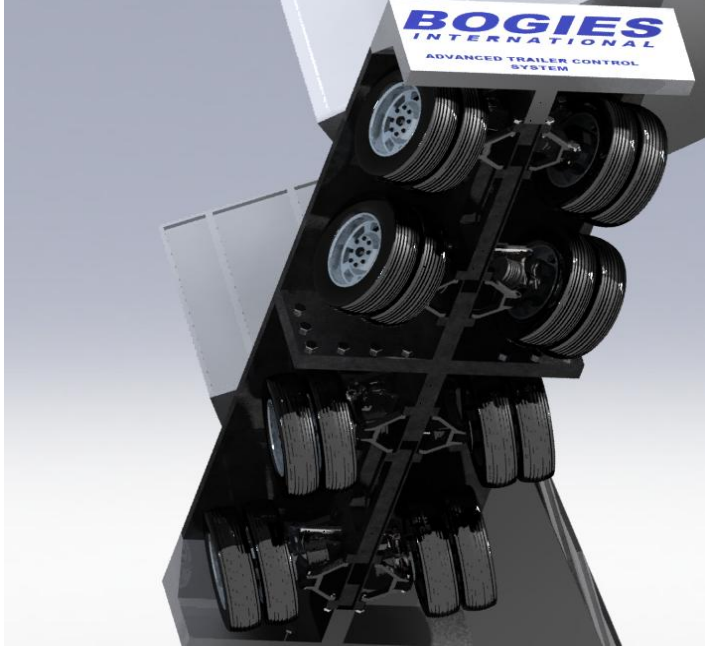
I have designed a theoretical system to achieve this goal. The bogies are modular units, they steer and compute their position independently and automatically. They just need to be plugged into the truck's alternator. The trailers in this case can be anything, and any length, and the bogies will function. They are also modular. They can be bolted back to back to create a double bogey, connecting two trailers but still spreading the weight over the required number of wheels. I have attached some pictures of my SolidWorks design. It is this design that will form the basis of what will be my project.



**Figure 2.1: The conceptual model of the future system.**



**Figure 2.2:** A view of a single bogie. The bogie is attached to the trailer with the same 5<sup>th</sup> wheel coupling found on the tractor. This allows the bogie to pivot. To attach this, the trailer owner can simply cut off the existing bogie wheels and weld on the new coupling.



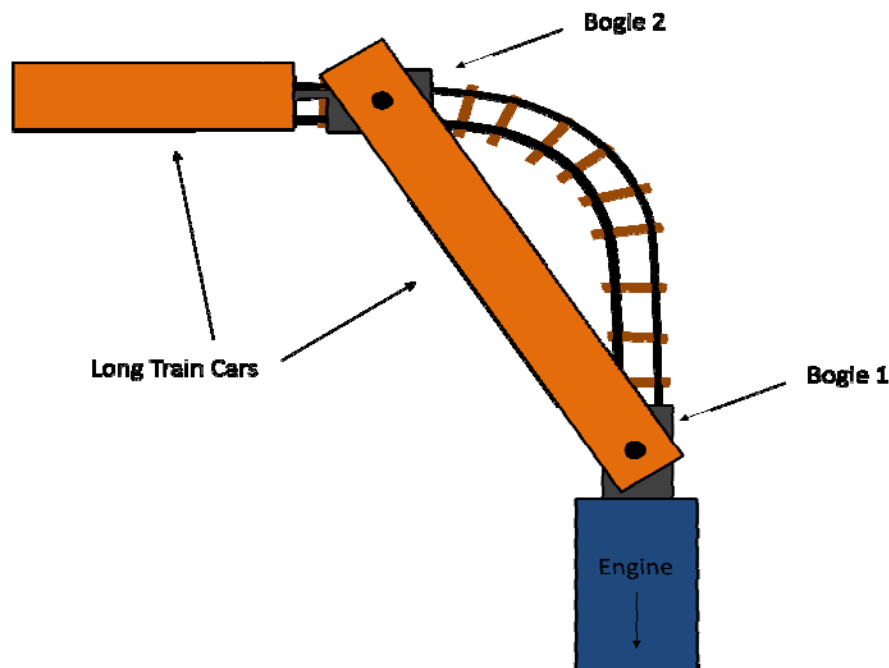
**Figure 2.3:** Two views showing the undercarriage arrangement for both a single bogie and a pair of two bogies bolted back-to-back to create a double. Each axle is designed to steer independently, so the steering behavior can easily be flipped between single-and double configurations. The bogies do not need special tires or wheels, and can be used with standard brakes, though this configuration provides a god opportunity to implement smarter braking systems. These bogies are also designed with AirRide independent air suspension bags for an improved ride over conventional bogies.



While I am confident that a system like this can be put together mechanically, that does not necessarily mean that it will work. There are a lot of different elements that need to work successfully in order for this system to succeed. The control system needs to be able to work with an erratic human driver, and it may be that throwing the weight of the rear trailer around makes the truck more difficult to drive. Maybe it will work under some conditions, but there are limits which need to be observed. In order to see if this steering system is a viable idea, some testing needs to take place. As I can't afford a real semi-truck, the testing will have to take place with a scale model that is as close as possible in construction to the real thing as possible.

## **2.2: Binding**

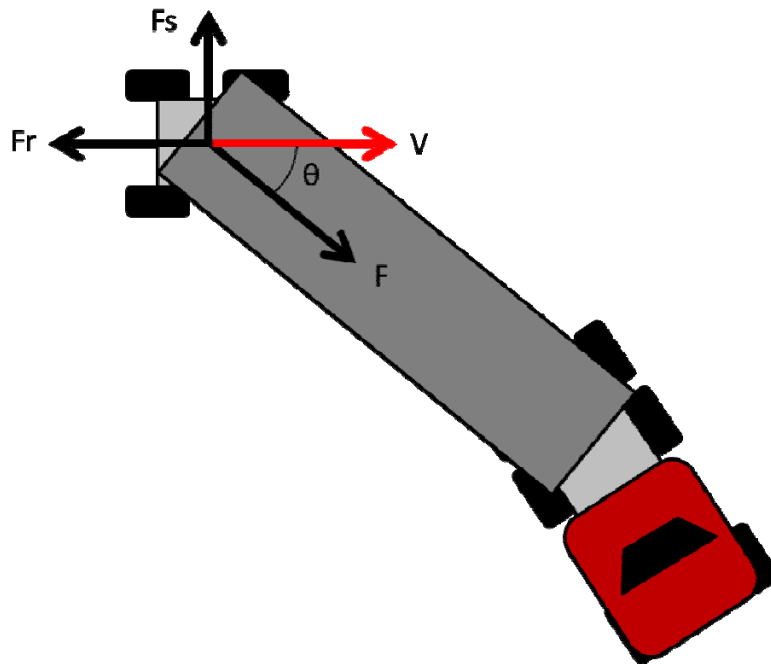
By far the most major physical obstacle to the success of this system is a phenomenon known to the rail industry as “binding”. “Binding” is a phenomenon where a train car with rotating bogies at each end attempts to roll across a very sharp turn, as illustrated in Figure 2.4.



**Figure 2.4: A train experiencing “binding”.**

When an engine pulls a train, all the force it exerts must go through the cars, like links in a chain. With a train, however, the direction the train can move is defined by the track, not the direction of the force from the engine. This can lead to a situation where the force of the engine is not in the direction the train can move. If the engine is somehow oriented 90 degrees to the track, the train won't move no matter how hard it pulls. In the situation in Figure 2.4, because the bogie #2 is travelling in such a different direction than the car is pointing, only a small amount of the force the engine exerts acts in the direction the bogie can travel. If that component of the force becomes too small to pull the weight of the train behind bogie #2, then the train will stop, and will not move no matter how hard the engine pulls. This is “binding”, the train car gets stuck and the train cannot move. The only way to get the train “unbound” was to get another engine to push bogie #2 through the curve. In the early days of rail, when there were few track standards and even fewer spare engines to extricate “bound” trains, sometimes the only solution was to cut the rear bogies off and drag the car through the turn, then attempt to reattach the bogie.

The self-steered truck experiences the same “binding” phenomenon, but with a slightly different effect.



**Figure 2.5: “Binding” behavior for a truck with my system.**

In the situation in Figure 2.5, binding will occur if either of the following two equations are true.

$$F \cos \theta \leq \sum \mu_r F_n \quad \text{Equation (1)}$$

$$F \sin \theta \geq \mu_s F_n \quad \text{Equation (2)}$$

Equation 1 determines whether the truck will be able to pull the trailer. If the force of the truck is not enough to pull the bogie along its direction of travel, it will not move. This is what happened to those early trains. But there is a second case unique to this trucking system. In this case, governed by Equation 2 when the truck “binds up”, instead of getting stuck, the front of the truck will simply drag the rear bogie sideways, since it isn’t on a fixed track like a train bogie. If the truck force is greater than the force required to pull the bogie sideways it will do so. In this effect lies the physical problem to be tested. There will always be a situation in which the truck “binds up”, but exactly under what conditions that will happen is unknown. It could be that the truck will only bind up in turns no sane truck driver would imagine making, or it could be the case that at even the slightest turn will result in the rear bogies skidding. If the latter is true, that the entire concept is essentially useless, no matter how good the programming is. So setting up this proof of concept test in such a way so that the binding behavior of the test rig matches that of a full-scale truck is absolutely crucial to this project. The reason I mention all of this is that it will be the crucial factor in selecting my design.

There are three main factors that affect the conditions under which the truck will bind, the friction coefficient of the tires on the ground, the weight on those tires, and the length of the trailers. So to achieve a valid test, the size and weight of the rig must be proportional to the actual truck it is modeled after, but also the friction of the tires needs to be as close as possible to what actual truck tires are like. This is the major factor that determines the scale. If I were to build this rig on a very small scale the friction coefficients of the small tires would not even approximate those of full scale tires, even if they were made of rubber and inflated. The test would be essentially useless because the small scale tires would be much more likely to slide than larger scale ones. I decided that tires on the scale of lawn mower tires, or even small trailer tires, would be a close enough approximation for the test to retain a reasonable validity.

## **2.3 Project Schedule**

As this project was being conceived it became clear quickly that a schedule was needed to keep the project moving. In early spring of 2010 a project schedule was created that would govern the entire process of developing the prototype.

Winter '09 / '10	<b>Concept developed and researched.</b>
Spring '10	<b>Truck and initial robot design completed.</b>
Summer '10	<b>Tractor vehicle chassis rebuilt in Maine and electronics restored to working condition. Programming 50% completed.</b>
Fall '10	<b>Entire robot mechanically complete, Remaining programming methodology developed, electrical systems designed.</b>
Winter '10 / '11	<b>Electrical system including sensors and controllers installed, beta program completed, begin initial testing.</b>
Spring '11	<b>Debugging and final testing, expansion and addition of features if time allows.</b>

Figure 2.6: Project schedule (written spring 2010).

It is the completion of these milestones upon which the progress of the project will be judged.

### **Section 3: Detailed Design**

The robot is roughly broken up into three components: the tractor, the bogie, and the trailer. Construction started over the summer and was completed on Nov.7. The robot design adheres as closely as possible to the concept outlined in Chapter 2, but there were limitations due to the available materials and funding. The braking and suspension systems were scrapped from the robot design because they are really not crucial to testing whether or not this will work. While it may not be an exact model of a real system, it is accurate in all of the most crucial areas needed to get the best test possible.

The overall machine is a roughly 1/3 scale model of a truck with a 42' trailer. It is roughly 14 feet long and about 2.5 feet wide at it's widest. The three components are separable at the joints, allowing the robot to be quickly broken up and assembled for easier storage. The electrical controls are separable as well, so when assembling the robot all one needs to do is plug the three components together. This eliminates any need to do rewiring during breakdown and assembly.

### **3.1: The Tractor**

The first section of the robot to be built was the tractor, (shown in Figure 3.1), in fact, it had been built five years previous. For a senior project a few years back a robot was built to run around campus guided by GPS. The robot had fallen into disrepair, but it is still more than powerful enough to pull trailers, and even better, it is built to the scale I'm aiming for. I decided to use this robot as the basis upon which to scale the rest of the design. Based on the tractor robot's width the overall machine will be a roughly 1/3 scale representation of a real truck. Over the summer I refurbished the existing robot into a reliable tractor unit that will be controlled by remote control. The repairs included new batteries, a new flip top to accommodate a trailer mount on top, and a refurbished undercarriage.

The tractor unit frame is constructed out of 80/20 aluminum extrusion. Steel plates were mounted between the lower frame rails in order to support two marine deep cycle batteries mounted at either end. The batteries are actually substantially larger than the original batteries so the original guards on the front and back were removed and replaced with sheet metal. The sheet metal holds the batteries in place, but otherwise they are not bolted or mounted to anything to allow for easy removal. In the center of the tractor are the two 24V DC motors. These are mounted to steel plates on the subframe and also to a square steel connecting bar. Each motor is mounted un-gear to a sprocket and chain. The wheels on each side are all connected by sprocket and chain, so all three wheels on each side turn together. Each side's chain system is connected with the sprocket and chain coming from one of the 24V motors. Each motor drives all three wheels on one side. In this sense, all of the wheels are drive wheels. This configuration is called skid steer, like a tank. By having each motor turn a different amount, the wheels on each side of the tractor turn different amounts, and the tractor turns. Even though the tractor is skid steer, it can be driven close enough to the way a real truck drives that I don't anticipate it being a problem. One future project being discussed is turning the robot into a more "truck-like" machine if time and money allows.

The wheels themselves are 11.5" pneumatic dolly tires. The tires are mounted to three-piece steel axles with two linked CV (constant velocity) joints. Inside the frame of the tractor each axle connects to the chain drive. The CV joints allow the axle to move up and down, which is important because each wheel is supported by its own independent suspension. The suspension linkage is a steel fourbar with four pivot points on the frame and the wheel plate. Attached to each of the suspension arms is a spring-shock assembly from a mountain bike.

On the top of the tractor the original electronics platform has been replaced with a folding lid. The lid is constructed out of 80/20 aluminum with a sheet metal top. The lid was built with two main

purposes in mind. First, When closed it protects the sensitive electronics under the lid from being hit. Secondly, it allows for the mounting of the trailer hitch on the top without any obstructions. The lid is locked shut while in use by two 1/4"-20 bolts.



**Figure 3.1: SolidWorks tractor model (above), and real tractor (below).**

### **3.2: The Bogie**

The second major component is the bogie, shown in detail in Figures 3.2 and 3.3. The primary material the bogie is constructed of is 80mm x 40mm aluminum extrusion (commonly called “profile”) donated by my place of internship, Lanco Assembly Systems. It was all metal that had been removed from automated assembly lines that were being refurbished. I designed a H-frame out of the profile that would be more than strong enough to handle any load I would put on it. The steering assemblies were then designed as two fourbar linkages. There are four steering knuckles cut from solid aluminum by the waterjet which pivot on steel bars set through the profile. Originally there were going to be bearings included, but these were eventually left out as the aluminum-to-aluminum contact had a very low friction coefficient already and bearings are expensive. The knuckles are connected by 1” steel square stock bars and pinned by M8 bolts to the knuckles. On each knuckle is a 1’ long 5/16” threaded steel rod secured with hex nuts. The threaded rod is sheathed with a 3/4” steel tube on which the wheel is set. The wheels themselves are the same type as was used on the tractor robot, but with bearings built-in. Mounting them on the tubes allows the wheels to be slid along the entire length of the tube, and allows me to test the bogie in an oversize load configuration. In position the wheels are secured by shaft collars.

With the steering system built I borrowed a few force gauges from Stan Gorski. I used the gauges to pull on the coupler bars and noted the maximum force indicated by the gauge. When doing this I tried to pull the steering through its full range of motion in about three to four seconds with the assembly sitting still on the shop floor. The gauges indicated about 15lbs of force was needed to actuate the steering assembly. On this basis I selected a 12V, 25lb linear servo which extends 4” in 2 seconds. There are very few servos with both a 25lb force and 2in/sec speed available that cost less than \$500, this \$130 model was a very good choice. The servo is mounted horizontally on the lower crossbar of the frame and its position can be easily adjusted to correct for bias. The servo is connected to the steering bars via two 15” aluminum links salvaged from a previous senior project. The long links run almost parallel to the servo and bars so the force is transmitted as close as possible in the direction of travel, but allows for the full range of motion of the bars.

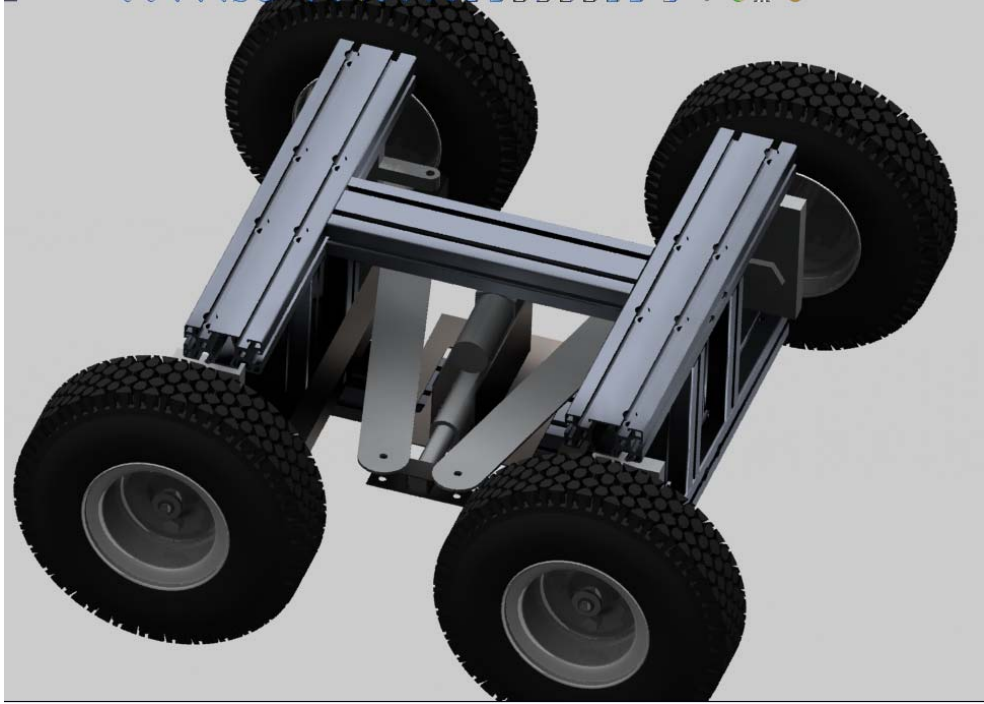


Figure 3.2: SolidWorks models of the bogie.

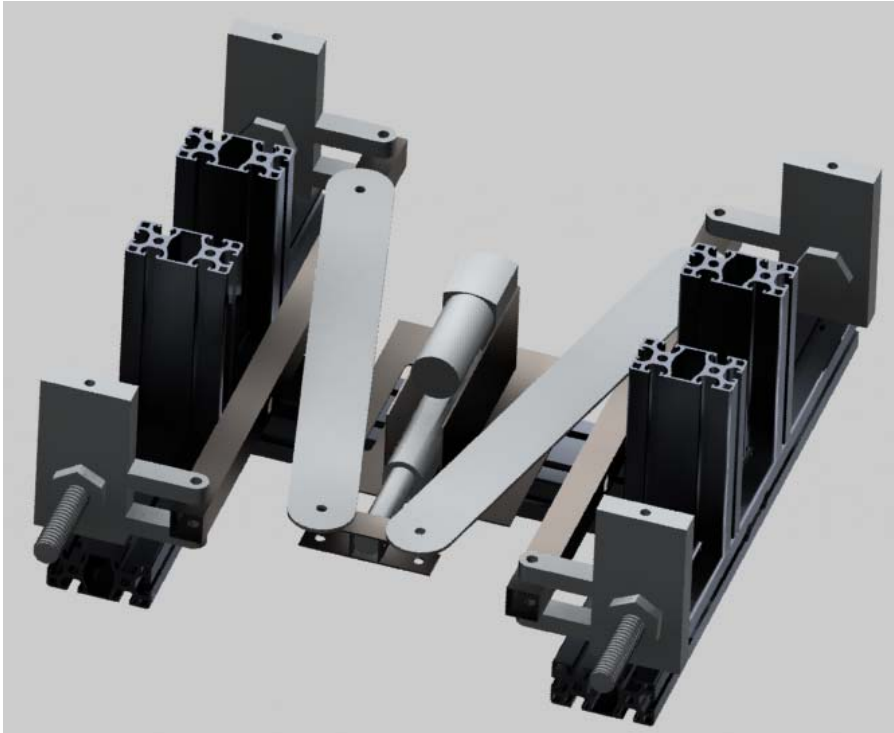
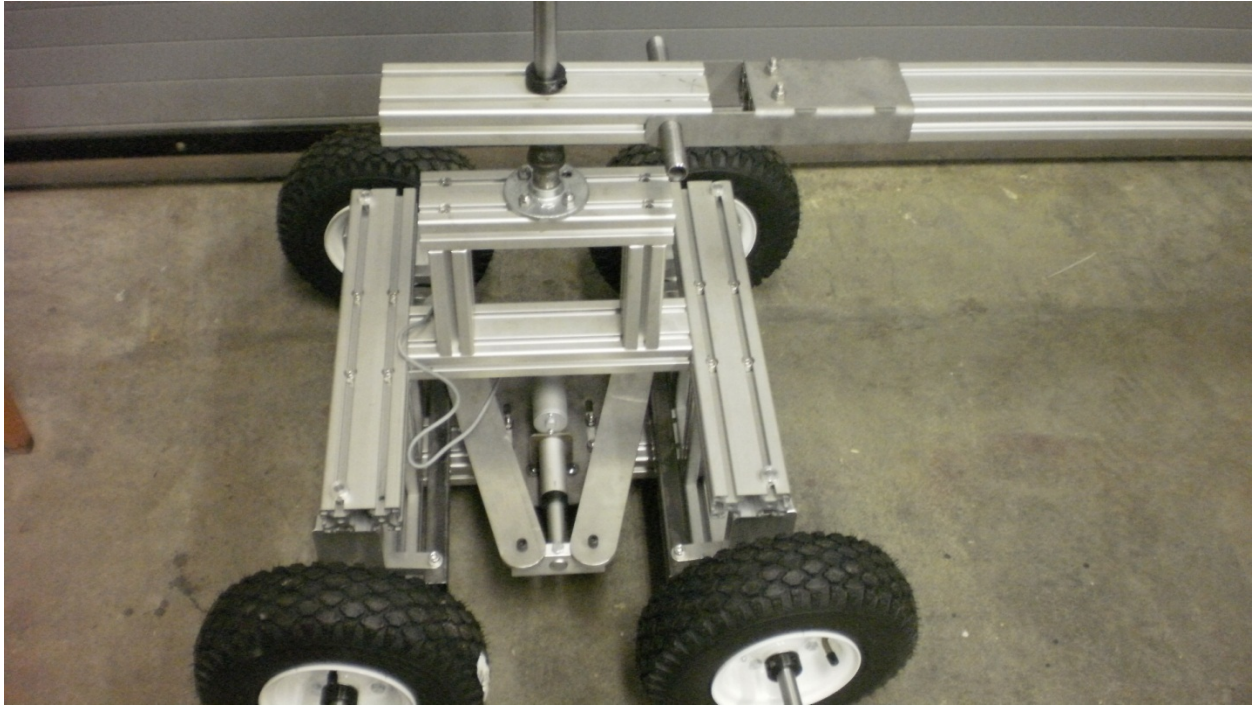


Figure 3.3: The model with the steering system isolated.



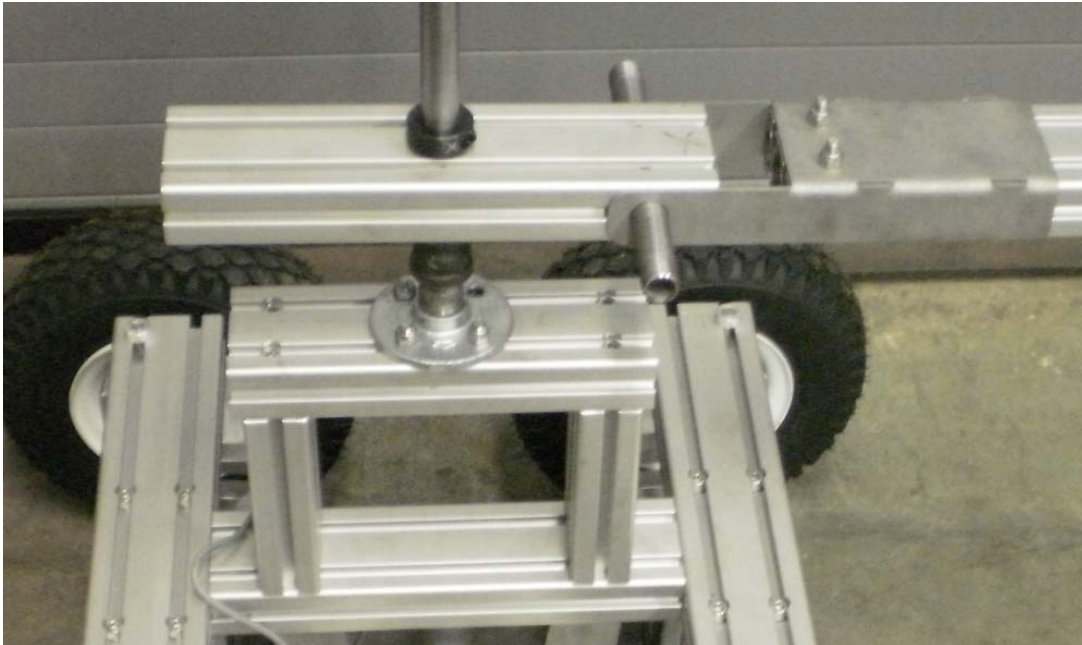


**Figure 3.4: Final bogie assembly with trailer attached.**

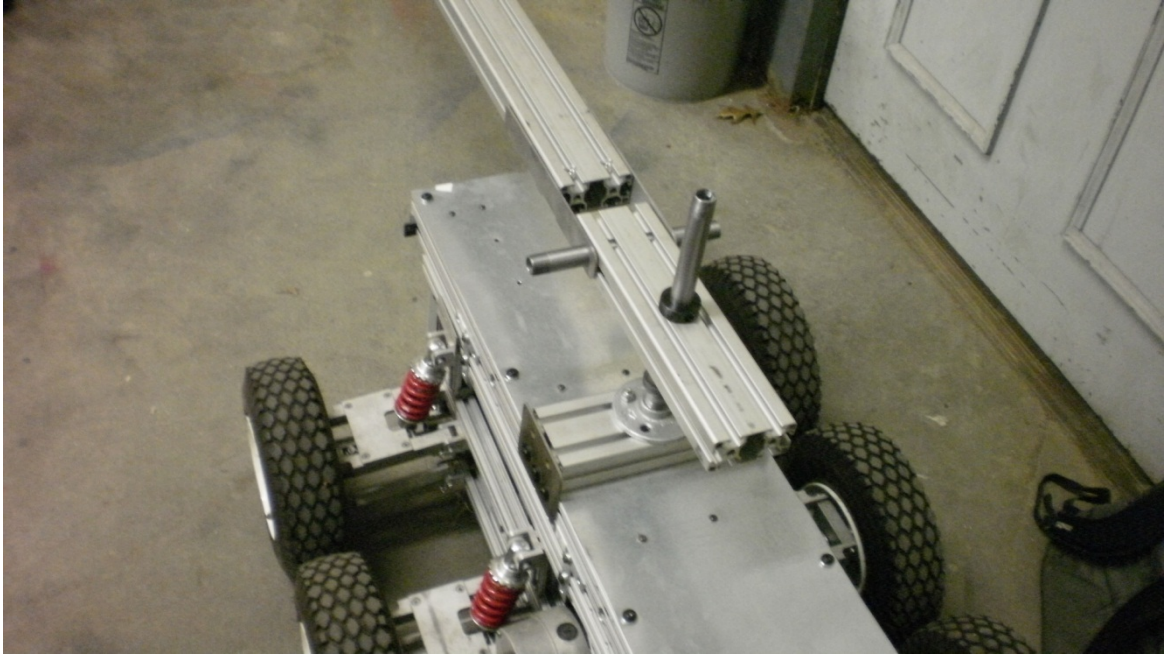
### **3.3: The Trailer**

The “trailer” assembly was the last to be designed, and is really not a trailer at all. It is more of a connecting bar connecting the tractor and bogie as seen in Figure 3.5-3.8. The trailer is 13’ from pivot to pivot, 45’ scaled. This makes the robot a scale representation of an average medium-long length trailer. The trailer itself is constructed from two lengths of profile joined at the middle by a 1/8” steel sheath bolted between the ends. This holds the two pieces securely together as one and can hold all the weight I would anticipate this bar would ever hold. At each end of the trailer are two stainless steel hinges, constructed the same way as the connector joint for a strong joint. A 3/4” pipe section runs through the steel holes and through a piece of profile, and secured with shaft collars, acts as a hinge pin. The hinge allows the robot to go up and over bumps without stressing the trailer. The trailer ends pivot about 3/4in pipe sections which are on the tractor and trailer mounts. This allows both the bogie and tractor to rotate independently relative to the trailer, which is crucial for the concept to work. The trailer had to be designed to fit both the construction of the tractor and bogie, so each mounting fixture was slightly different. Both use the 3/4” fittings for pivoting mounted with a steel flange to a length of profile. On the tractor, the profile is mounted to the opening lid with two steel plates. The plates had to be custom cut in

order to adapt the geometry of the metric profile to the 80/20 aluminum the tractor was built from. This mounting method allows the lid to be opened with the trailer off, but still support and pull the trailer with the lid closed. On the bogie the pivot pipe had to be raised 9" off the original frame in order to sit at the same height as the pipe on the tractor, as the tractor is much higher off the ground. This extra height was achieved by building a small frame out of the profile used the original bogie frame.



**Figure 3.5: Picture of the trailer mount on the bogie.**



**Figure 3.6: Picture of the trailer mount on the tractor.**



**Figure 3.7: View of the entire robot in SolidWorks.**



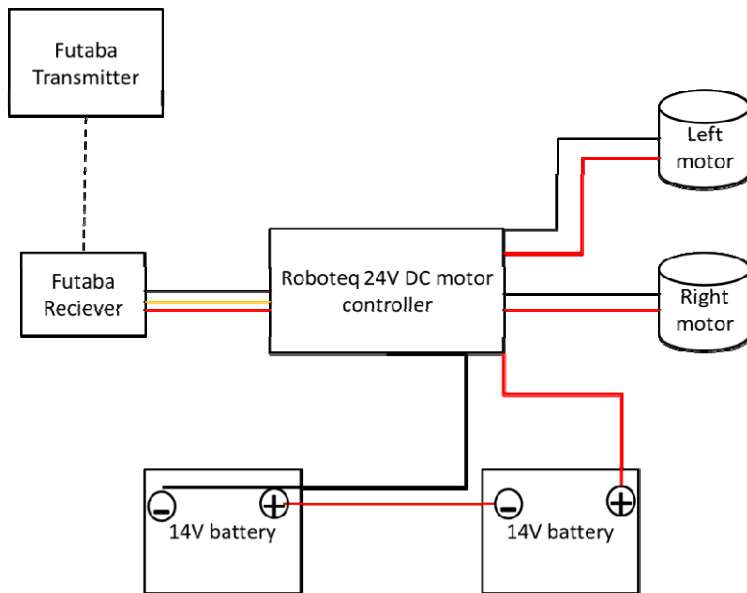
**Figure 3.8: View of the entire robot in the shop.**

### **3.4 The Electrical System**

The electrical system was both the primary focus of the project work over the second term, and one of the main pieces of work completed over the summer. The electrical system can be cleanly broken up into two parts, the drive system and the control system. Both systems were built at different times and operate almost independently of each other.

The drive system is located on the tractor and is an extensive modification of the electronics that were previously on the robot. The original system provided an extensive system of relays, a DC-AC charger for a laptop, and a whole other array of control circuits because the robot was originally intended to operate autonomously. Over time this electrical system had degraded, many of the circuits had burnt out, and there was no wiring diagram available to try to determine how it worked. So it was decided to start from scratch. All the tractor needed to do for my project was to drive by remote control, and behave just like the tractor of a real truck. The entire electrical system including the batteries and motors was

removed over the summer because at the time the entire frame was being rebuilt and refurbished. As the frame came back together a new system was built using as many of the original components as possible. The two original batteries were replaced with 14.4V marine deep cycle batteries in order to improve the available running time of the tractor. The original Roboteq AX2550 24V dc motor controller was mounted under the new flip top for protection. Also the Futaba receiver unit was mounted under the top and the antenna rerouted around the side. The new drive system is wired according to Figure 3.9.



**Figure 3.9: The drive system diagram.**



**Figure 3.10: The drive system installed.**

The basic operation principle of the drive system is as follows. The marine batteries provide power to operate the motor controller and the motors, and a small battery powers the Futaba 2-channel receiver. An operator sends signals to the receiver from the remote. The receiver then sends a series of PWM signals to the Roboteq controller. The Roboteq controller translates these signals into net voltages across each of the motors. This essentially allows the tractor to be driven like an R/C car. Actually it is more precisely an R/C tank, which was a small source of concern. The tractor robot has no steering linkage like a real truck. It uses a method called “skid-steer”, the same as a tank, crawler crane, or bobcat. It steers by moving the wheels on one side at a different rate than on the other. This allows the tractor to literally spin in place. It is possible to actually program the tractor using a microcontroller to have “truck-like” steering behavior, but after some early tests, this was abandoned. It is actually quite easy to drive the tractor like a truck with a little practice.

The control system was the primary focus of the winter term project. The focus and setup is completely different from the drive system, and it operates completely independently. The objective of the control system is to retrieve essential data from sensors set up around the robot, send that data to a laptop running the control algorithm, and retrieve from the laptop a command telling the steering servo on the bogie what to do. It must also power and control that steering servo based on the laptop’s command. Unlike the drive system, the control system is completely autonomous.

Before the start of winter term the program had been mostly written, and the measurements needed to run it had been identified:

- The absolute X-Y position of the center of the tractor.
- The relative angle of the trailer to the tractor
- The relative angle of the bogie to the tractor
- The angle of the bogie’s steering system.

The final three readings had been thought out ahead of time and were very simple to make. Simple 10k $\Omega$  potentiometers were installed at the points where the bogie and tractor pivot relative to the trailer.

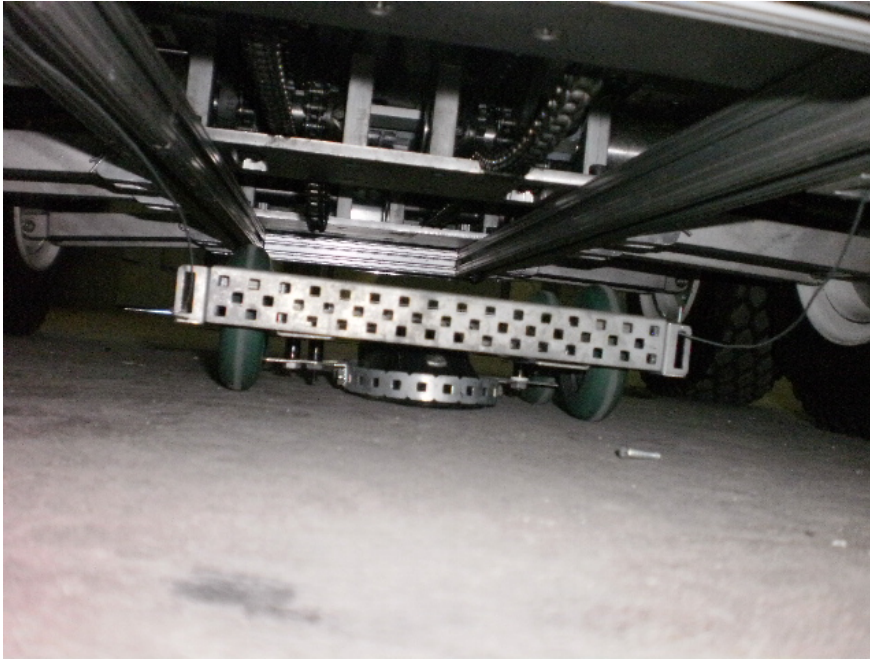
Another 10k $\Omega$  potentiometer was pre-installed on the linear actuator to measure it’s extension length.

These potentiometers output a voltage drop that changes as the resistance in the potentiometer changes. A simple calibration function is all that is needed to translate this voltage reading into an angle measurement.

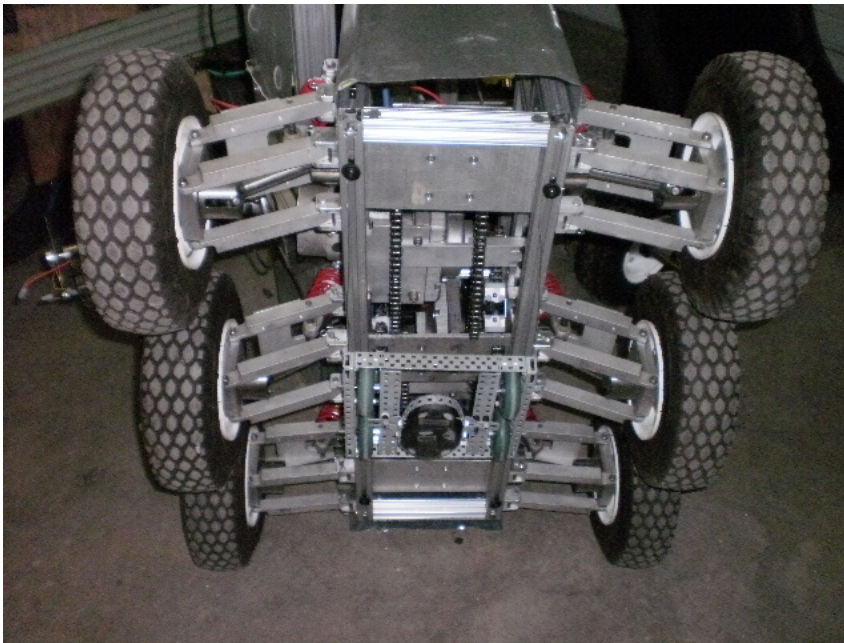
What was much more difficult to determine were the absolute measurements of the tractor position. Over four weeks many ideas were tossed around with the generous help of Prof. Hedrick. The first concept to be explored was the idea of putting rotation encoders on the wheels of the tractor. By measuring how each

side of the skid-steer drive rotates, the position and orientation of the tractor relative to its original position could be calculated. The problem is that this will not be accurate if the wheels of the tractor slip at all, which is nearly guaranteed to happen with a skid-steer device. The next option explored was the GPS option. In a full-scale truck a GPS resolution of a couple feet is not a problem, that is good enough for the system to work. But in this scale model GPS is simply not accurate enough for the system to work. So GPS was abandoned. What was then tried was an inertial navigation system. A Logitech MX Air mouse was purchased for testing. The MX Air mouse is a mouse which operates entirely based on gyros inside the unit. It is intended for presentations, where the presenter can hold the mouse in the air and use it to control the cursor on a projector. It is quite accurate, and can essentially function as a “poor-man’s” inertial navigation system. The problem is that the mouse has the wrong set of axes. That is, the cursor moves up on the screen if one moves the mouse up, not forward-back like a conventional mouse. Many fixes were tried, but the mouse was designed well and the axes would not re-orient despite many trials. That made it useless for getting X-Y coordinates, so it was sent back.

After the failure of the air mouse there were very few options left, but the air mouse itself had inspired an idea. A conventional mouse works with either an optical camera or laser aimed at the ground. The mouse fires the laser or takes a picture, and records how the ground has moved under it since the previous picture or laser firing. It is an absolute position recording. It was decided to investigate a conventional mouse as a method of recording the X-Y position of the tractor. A Microsoft Laser mouse was used. The key to making it work was getting the mouse close enough to the ground so that it would get a signal, because most laser mice are tuned to turn off if lifted a small distance above the ground, to allow the user to reposition the mouse more easily. It was understood that using the mouse would limit the robot to smooth, flat floors, but that was deemed acceptable for this demonstrator. After many unsuccessful trials a cart was built that cradled the mouse 1.5mm off of the floor. This cart is pictured in Figure 3.11.



**Figure 3.11: Cart with mouse on the ground.**



**Figure 3.12: Cart with mouse in the air to illustrate its positioning.**



The cart is attached to the tractor only by four wires, which ensure the cart is always moving in the direction of the tractor but do not fix its height. The cart sits on its own set of wheels, which allows it to move up and down with the slight contours of any floor. This keeps the mouse at the proper distance. The mouse itself is wireless and has its own onboard batteries. As the tractor moves forward or back, the cursor moves up or down on the screen. A program using the Windows input manipulator AutoHotkey was written which resets the cursor to the bottom of the screen if it reaches the top, and vice versa. This allows the mouse to move forward or back continuously. The laptop running MATLAB reads the position of the cursor on the screen, and via a conversion factor converts it into a distance. The system works well on both concrete and tile floors, getting 2cm or better resolution at low speeds, and 10cm resolution at high speeds. The problem with the mouse alone is that it only provides a scalar displacement measure. The mouse is constrained to the same direction as the tractor, so even if the tractor turns the mouse will only measure the scalar amount of forward/back travel. To get an X-Y position the heading of the tractor must be known along with its scalar displacement. This was thankfully much easier to obtain. A Devantech CMPS03 digital compass was purchased. The compass outputs a PWM signal which corresponds to the current heading. Formulas are readily available to convert this PWM signal into a degree reading. True north is 0 degrees, south 180. The compass is accurate to roughly half a degree, which allows an accurate heading of the tractor to be obtained. This provides all of the necessary information to calculate the X-Y position for the tractor. It should also be noted that the compass is actually mounted on the trailer, but as there is a potentiometer giving the angle of the tractor to the trailer, the heading of the tractor is still easily computed.

The setup of the control system is shown in system 3.12. .

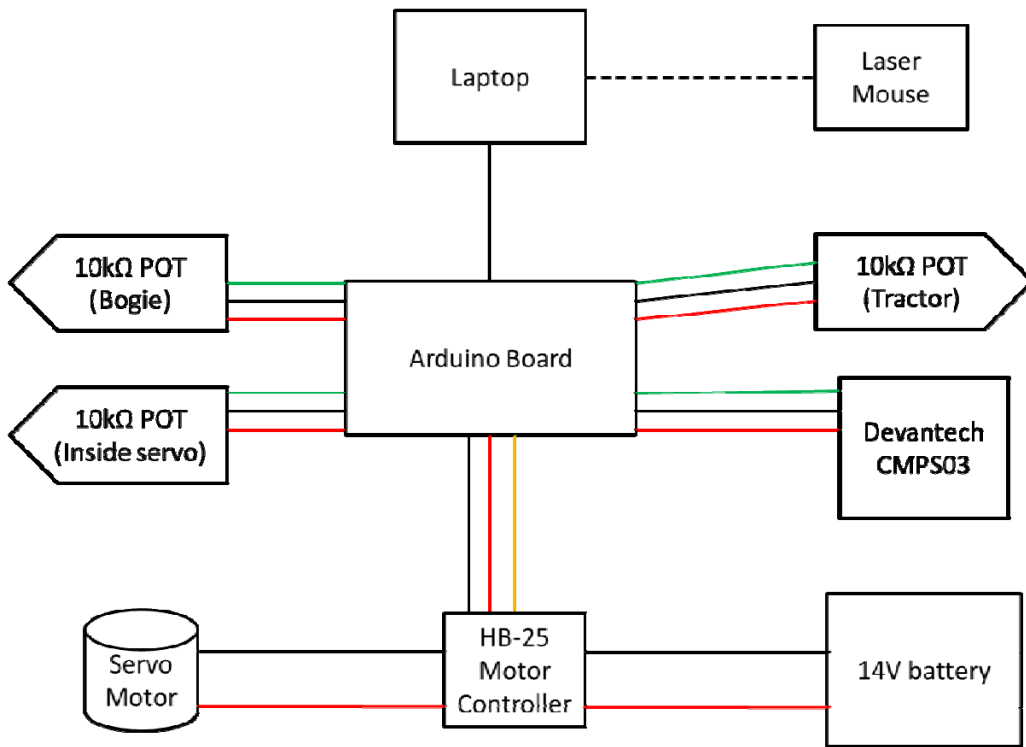


Figure 3.12: System diagram

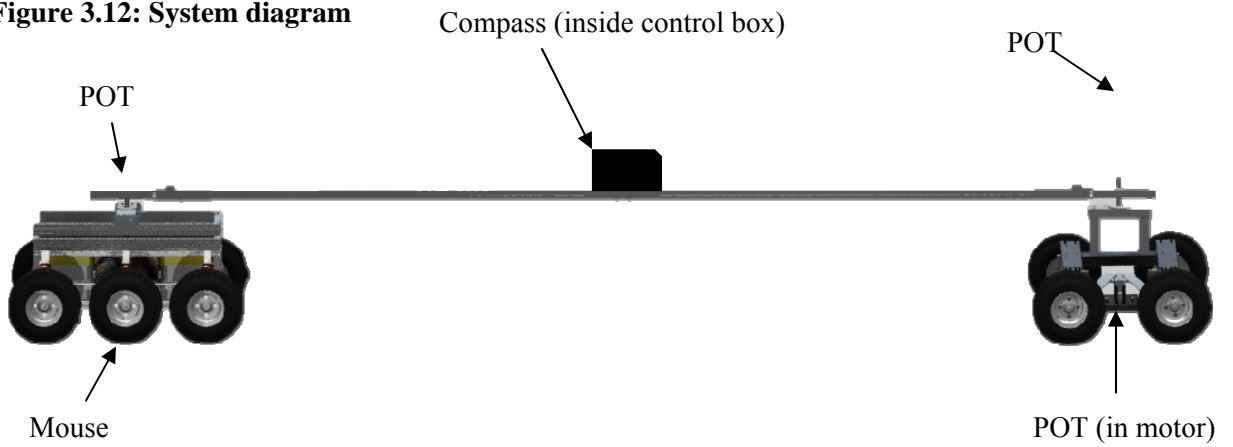
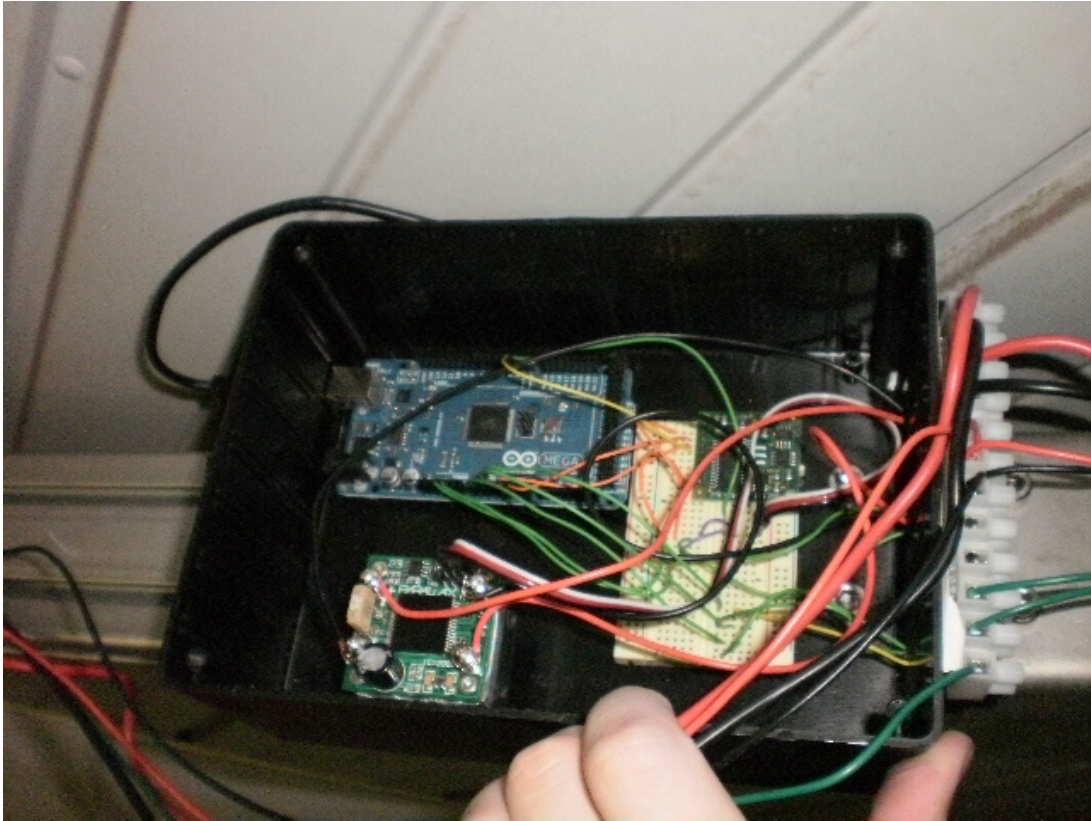


Figure 3.13: Sensor Layout



**Figure 3.14: Picture of the Control Box**

There are three main control components which enable all of the sensors and motors in the control system to work with each other. All of these components are mounted in a box which in turn is mounted on the center of the trailer. The main controller is the laptop, which runs the top-level MATLAB code. Currently I have been using my personal laptop but once the code is complete a dedicated Acer Eee Pc will be used. The laptop runs on its own batteries and connects wirelessly to the mouse via USB port. Connected to the other USB port of the laptop is the second major control component, the Arduino Mega 2560 microcontroller. The Arduino board is powered by the laptop, and performs almost all I/O functions. It powers all of the sensors via a 5V signal and sets motor power via PWM. The Arduino runs its own program, but that program is slaved to the MATLAB code via the serial port. All the Arduino does is take the sensor inputs and sends them to the MATLAB code. While the Arduino board itself could potentially control the system without the laptop, the idea to use the board came after most of the code had been written in MATLAB. The third control component is a Parallax HB-25 motor controller. The HB-25 is connected to one of the marine batteries on the tractor unit, to the linear actuator, and to the Arduino board. Upon receiving a command from MATLAB the Arduino board sends a PWM signal to the HB-25, which then translates it into a voltage seen by the linear motor. The voltage is supplied by the marine

battery and is completely reversible. So the motor can be run at full forward, full backward, and anything in between.

This and all of the electronics have at present been fully installed and tested, and are all working correctly.

### **3.5: Programming**

The programming of this system was almost entirely done in MATLAB though there is a small secondary component which should be mentioned first. The Arduino board is not programmed in MATLAB, it is programmed in C++. The way it communicates with MATLAB is through a class called ArduinIO written initially by TheMathWorks. ArduinIO is a MATLAB class and Arduino script which allow the two to communicate effectively over the serial port. In this way it actually slaves the Arduino to MATLAB. The way it works is that the Arduino board is set up with a server script. It waits for commands to be sent from the MATLAB code, it executes those commands, and returns the result back to MATLAB. In the case of ArduinIO, those commands are generally to read and write to the boards pins. The MATLAB class is a set of functions which, when called, send commands to the Arduino server, and waits for it to send results back. These functions allow the MATLAB code to read and write pins on the Arduino directly. The source ArduinIO code was downloaded from TheMathWorks, and both the C++ server and the MATLAB class were extensively modified for use in this project.

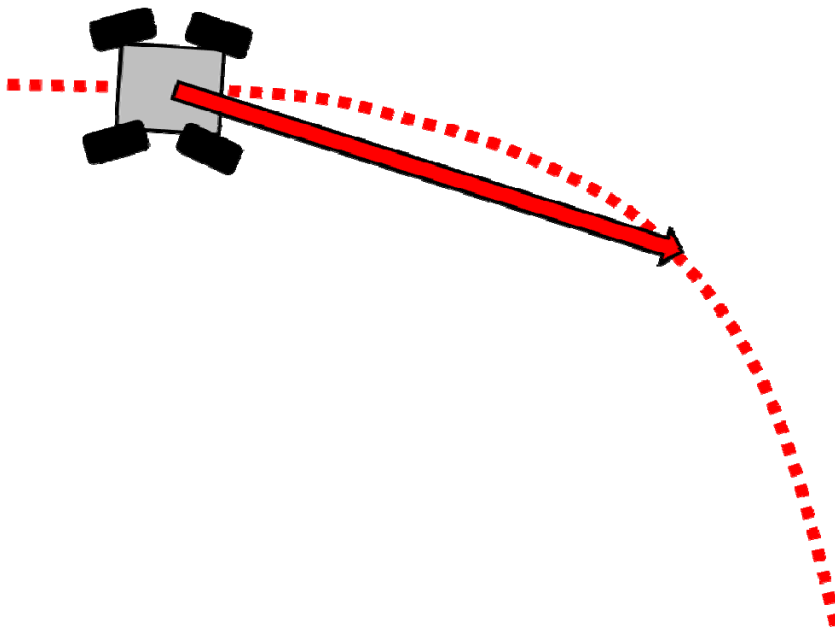
The control algorithm for this system runs entirely in MATLAB and has been designed using the following goals:

- To control the bogie enough so that it follows the path of the cab as closely as possible.
- To do so in an entirely general way independent of driver behavior or turn geometry.  
(Within physical limits, of course.)
- To do so in a way that is user-friendly, reliable, and adaptable.

There are a number of different possibilities for control options, a proportional controller which corrects based on how far the bogie is from the intended path, and integral controller which corrects based on how far off it is and how long it has been off for, and a derivative controller which corrects based on whether it is getting closer to the tractor's path or not. Each of these alone was deemed insufficient to solve this problem, for a good reason. A simple reactionary control system will not work in this case because of the output, the bogie steering. This control system can only really affect one thing, the power of the linear servo on the bogie steering. This means that all it can really control is the rate of change of

the bogie's steering angle, or the rate of change of the rate of change of the bogie's heading. The rest is all determined by physics. Because the control system is so limited in its effect it cannot simply be reactionary. It cannot simply slide the bogie to the right if it drifts left, it must turn it, which takes forward motion to make happen. In other words, this system must be a proactive in its control methods. That is, the system must send the signal to start turning the wheels before it actually reaches the turn in order to have the linear actuator moving in the right direction when the turn arrives. The need for this functionality necessitated an original control algorithm.

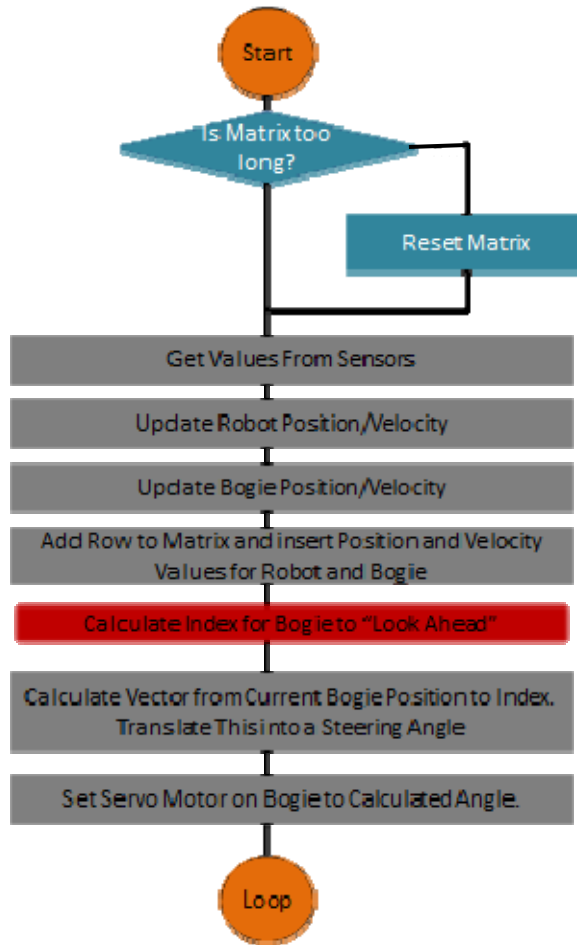
The basic premise of the control algorithm is illustrated in Figure 3.15.



**Figure 3.15: The Idea Behind the Control Code.**

The control program iterates over and over, recording the position of the tractor and the bogie at each instant in time. The path of the tractor is the red dots above. For each iteration, the program takes the position of the bogie, and “looks ahead” a certain number of points along the tractor's path. It then “aims” for that point. It calculates the vector from the bogie's position to the point, if that vector is to the right of where the bogie is currently heading, it sets the motor to steer more right, and vice-versa. Remember, the program can only set the rate at which the steer angle changes.

Figure 3.16 is a flowchart outlining the logic behind the control algorithm.



**Figure 3.16: Logic Flowchart.**

This flowchart outlines the basic program setup, the actual files are in the appendix. The first key function is the first, blue function. “Matrix” refers to an array in which all pertinent data for each iteration is stored, like the tractor’s XY position and heading, it’s velocity, and the same for the bogie. It is where the path of the tractor is stored. The blue function actually performs a number of tasks. First it resets the Matrix if it gets too long. This just saves memory. When it does so it keeps the last few rows so that the part of the path that the bogie will still reference is preserved. It also does not add values to Matrix if the robot is not moving. The reason this is done is because of the method of picking a point.

What this program does when it picks an “aim” point is really to pick an index of Matrix. First it finds the point on the tractor’s path that is closest to the current position of the bogie. It then takes that point, and, based on some formula, looks a certain number of rows down in Matrix. It then takes the tractor’s position data for that row and uses it as the aim point. The weakness of this design is that as the tractor accelerates and decelerates, the points at which a position measurement is taken become closer or

more spread out. This creates a potential problem because the algorithm calculates the number of rows to look ahead, not the distance to look ahead. If the tractor is moving slowly for a long time and then takes off, the position point will become very densely populated and large numbers of rows of Matrix may read almost the same tractor position. This means the bogie will look a certain number of indices ahead and it will spend much of its time looking at the points where the tractor moves slowly, even though distance-wise it should be looking farther ahead. This problem becomes most acute if the tractor is not moving for an extended period. Matrix will fill up with a position of (0,0) for the tractor, and when the cab starts moving and the bogie “looks ahead”, all it will aim for is the (0,0) point until the Matrix has filled way beyond those initial points.

Highlighted in red in the flowchart is the function that is really the heart of the control algorithm, calculating how far to “look ahead”. The rest of the functions are really just doing vector math and translating sensor readings into variables; it is the function in red that really controls things. At present the exact nature of this function is being investigated; all that is currently known is that it is a velocity-dependent function. That is, the bogie looks farther ahead for its aim point if it is traveling faster. Otherwise, this function is currently unknown.

The programming is at present the only part of this project that is still incomplete. The functions in grey on the flow chart were all written and tested either over the summer or during the second term of the project. The entire program is complete except for the red function. It was decided to delay the running of the first beta programs in the robot, (originally scheduled for week 8 of winter term), and investigate the red function further. Initial beta testing has now been moved to the first few weeks of spring term. The reason for the delay was to allow for time to write and test simulation functions. These functions simulate the movement of a tractor and bogie and allow the control system to be tested, debugged, and properly simulated before being run in the robot. Currently these simulation functions are being debugged, and hopefully the control algorithm can start being tested by final exam week.

## Section 4: Summary

With the completion of the robot on Nov.7, 2010 I had successfully completed the major goals for term #1:

- Have the robot mechanically completed.
- Have a basic program architecture completed.
- Made initial determinations as to what would be an optimal sensor layout.

And as of March 8, 2011 I had completed all but one of the major goals for the second term.

- Design and test electrical system
- Calibrate all sensors
- Complete program architecture
- **Run first beta program (incomplete)**

This puts the project about one or two weeks behind schedule for the spring term. With the robot finished mechanically, and the electrical system and sensor setup finished, next term will be more devoted to debugging and testing the control systems, with the ultimate goal of completing a polished demonstrator by Steinmetz. At present here is the schedule for the next term.

- By 3<sup>rd</sup> week: Complete beta program.
- Testing, debugging.
- Steinmetz day: Completed demonstrator

I believe that achieving these goals is very feasible barring any more major problems with the programming. It should be noted that when I started this project last spring my goal was a working prototype by Steinmetz, and I am on track to achieve that.



## **A: References and Acknowledgments**

The truckers at Benoit Trucking and TheTruckersReport.com

Prof. Keat

Lanco Assembly Systems

Paul Tompkins, James Howard at the machine shop

Professor Hedrick

Stan Gorski

NSF CT Scholars funding

Union IEF

## **B: Budget and Sourcing**

### Funding

Union IEF: \$500

NSF CT Scholars \$1000

### **Major Expenses**

- Wheels- Northern Tool Inc.: \$150
- Servo- ServoCity: \$130
- Square Stock, Locknuts, 5/16 Threaded Rod, 3/4 Tubing, Flanges, Pipe Connectors, 1/8 and 3/16 Steel Rod, Misc. Hardware: Lowe's: \$100
- M8 Stainless Steel Bolts 45mm,55mm,80mm, Locknuts, Lockwashers, Delrin Bushings, 3/4" Shaft Collars: McMaster-Carr: \$80.
- 35ft of aluminum 80mmx40mm profile: Scrap donated by Lanco Assembly Systems.
- 1/8" steel plate for trailer and servo mounts: Scrap metal from Machine Shop.
- 1/8" aluminum links: Scrap from previous senior project.
- Arduino Mega 2560: \$125
- Parallax HB-25 Motor Controller: \$50
- Devantech CMPS03 Compass: \$75
- Potentiometers: \$10
- Electrical box, Terminals, Wire, 3- Wire connectors, Misc. components: \$30

## **C: Code**

MousemoveX.ahk

```
a=0
w=1600
h=900
CoordMode, Mouse, Screen
Loop {
MouseGetPos, xpos, ypos
xnew=xpos

if xpos >= 1599
{
MouseMove, 2, ypos
xnew=2
}

if xpos <= 1
{
MouseMove, 1598, ypos
xnew=1598
}

}
```

```
#include <Wire.h>

#define address 0x60//defines address of compass

void setup(){

  Serial.begin(9600);
}

void loop(){
  int pot=analogRead(2);
  int bearing =pulseIn(3, HIGH)/100;
  if (bearing < 0){
    bearing = 327+(bearing+327);
  }
  else {
  }
  Serial.println(pot);
  delay(1000);
}
```

```
/* Analog and Digital Input and Output Server for MATLAB */
```

```
void setup() {
```

```
    int i;
```

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

```
        pinMode(i,INPUT);
```

```
        digitalWrite(i,0);
```

```
    }
```

```
    /* initialize serial */
```

```
    Serial.begin(9600);
```

```
}
```

```
void loop() {
```

```
    /* variables declaration and initialization */
```

```
    static int    s    = -1; /* state */
```

```
    static int    pin  = 13; /* generic pin number */
```

```
    int          val   = 0; /* generic value read from serial */
```

```
    int          agv   = 0; /* generic analog value */
```

```
    int          dgv   = 0; /* generic digital value */
```

```
    if (Serial.available() >0) {
```

```
        val =Serial.read();
```

```
        switch (s) {
```

```
            /* s=-1 means NOTHING RECEIVED YET ***** */
```

```
            case    -1:
```

```
                /* calculate next state */
```

```
                if (val>47 && val<58) {
```

```
                    /* the first received value indicates the mode
```

```
                       49 is ascii for 1, ... 90 is ascii for Z
```

```
                    s=0 is change-pin mode
```

```
                    s=10 is DI; s=20 is DO; s=30 is AI; s=40 is AO;
```

```
                    s=50 is servo status; s=60 is aervo attach/detach;
```

```
                    s=70 is servo read; s=80 is servo write
```

```
                    s=90 is query script type (1 basic, 2 motor)
```

```

                                                                    */
    s=10*(val-48);
}
break; /* s=-1 (initial state) taken care of */

/* s=0 or 1 means CHANGE PIN MODE */

case 0:
/* the second received value indicates the pin
   from abs('c')=99, pin 2, to abs('t')=116, pin 19 */
if (val>98 && val <117) {
    pin=val-97; /* calculate pin */
    s=1; /* next we will need to get 0 or 1 from serial */
}
else {
    s=-1; /* if value is not a pin then return to -1 */
}
break; /* s=0 taken care of */

case 1:
/* the third received value indicates the value 0 or 1 */
if (val>47 && val <50) {
    /* set pin mode */
    if (val==48) {
        pinMode(pin,INPUT);
    }
    else {
        pinMode(pin,OUTPUT);
    }
}
s=-1; /* we are done with CHANGE PIN so go to -1 */
break; /* s=1 taken care of */

/* s=10 means DIGITAL INPUT ***** */

case 10:
/* the second received value indicates the pin
   from abs('c')=99, pin 2, to abs('t')=116, pin 19 */
if (val>98 && val <117) {
    pin=val-97; /* calculate pin */
    dgV=pulseIn(pin, HIGH)/100; /* perform Digital Input */
    if (dgV < 0){

```

```

    dgv = 327+(dgv+327);
}
else {
    }/* perform Digital Input */
    Serial.println(dgv);
    delay(1000);    /* send value via serial */
}
    s=-1; /* we are done with DI so next state is -1 */
break; /* s=10 taken care of */

/* s=20 or 21 mean DIGITAL OUTPUT ***** */

case 20:
/* the second received value indicates the pin
    from abs('c')=99, pin 2, to abs('t')=116, pin 19 */
if (val>98 && val <117) {
    pin=val-97; /* calculate pin */
    s=21; /* next we will need to get 0 or 1 from serial */
}
else {
    s=-1; /* if value is not a pin then return to -1 */
}
break; /* s=20 taken care of */

case 21:
/* the third received value indicates the value 0 or 1 */
if (val>47 && val <50) {
    dgv=val-48; /* calculate value */
    digitalWrite(pin,dgv);    /* perform Digital Output */
}
    s=-1; /* we are done with DO so next state is -1 */
break; /* s=21 taken care of */

/* s=30 means ANALOG INPUT ***** */

case 30:
/* the second received value indicates the pin
    from abs('a')=97, pin 0, to abs('f')=102, pin 6,
    note that these are the digital pins from 14 to 19
    located in the lower right part of the board */
if (val>96 && val <103) {
    pin=val-97; /* calculate pin */
    agv=analogRead(pin);    /* perform Analog Input */
}

```



```

        Serial.println(agv);          /* send value via serial */
    }
    s=-1; /* we are done with AI so next state is -1 */
    break; /* s=30 taken care of */

/* s=40 or 41 mean ANALOG OUTPUT ***** */

case 40:
/* the second received value indicates the pin
   from abs('c')=99, pin 2, to abs('t')=116, pin 19 */
if (val>98 && val <117) {
        pin=val-97; /* calculate pin */
        s=41; /* next we will need to get value from serial */
    }
else {
        s=-1; /* if value is not a pin then return to -1 */
    }
    break; /* s=40 taken care of */

case 41:
/* the third received value indicates the analog value */
analogWrite(pin, val);          /* perform Analog Output */
    s=-1; /* we are done with AO so next state is -1 */
    break; /* s=41 taken care of */

/* s=90 means Query Script Type (1 basic, 2 motor) */
case 90:
if (val==57) {
        /* if string sent is 99 send script type via serial */
        Serial.println(1);
    }
    s=-1; /* we are done with this so next state is -1 */
    break; /* s=90 taken care of */

} /* end switch on state s */

} /* end if serial available */

} /* end loop statement */

```

```

classdef arduino < handle

    % This class defines an "arduino" object

    properties (SetAccess=private,GetAccess=private)
        aser    % Serial Connection
        pins    % Pin Status Vector
        srvs    % Servo Status Vector
        mspd    % Motor Speed Status
        sspd    % Servo Speed Status
        mots    % Motor Server Running on the Arduino Board
    end

    methods

        % constructor, connects to the board and creates an arduino object
        function a=arduino(comPort)

            % Add target directories and save the updated path
            addpath(fullfile(pwd));
            savepath

            % check nargin
            if nargin<1,
                comPort='DEMO';
                disp('Note: a DEMO connection will be created');
                disp('Use a the com port, e.g. ''COM5'' as input argument to connect to
the real board');
            end

            % check port
            if ~ischar(comPort),
                error('The input argument must be a string, e.g. ''COM8'' ');
            end

            % check if we are already connected
            if isa(a.aser,'serial') && isvalid(a.aser) && strcmpi(get(a.
aser,'Status'),'open'),
                disp(['It looks like Arduino is already connected to port ' comPort ]);
                disp('Delete the object to force disconnection');
                disp('before attempting a connection to a different port.');
```

```
        disp('to delete the port before attempting another connection');
        error(['Port ' comPort ' already used by MATLAB']);
    end

    % define serial object
    a.aser=serial(comPort);

    % connection
    if strcmpi(get(a.aser, 'Port'), 'DEMO'),
        % handle demo mode

        fprintf(1, 'Demo mode connection ..');
        for i=1:4,
            fprintf(1, '.');
            pause(1);
        end
        fprintf(1, '\n');
        pause(1);

        % chk is 1 or 2 depending on the script running on the board
        chk=round(1+rand);

    else
        % actual connection

        % open port
        try
            fopen(a.aser);
        catch ME,
            disp(ME.message)
            delete(a);
            error(['Could not open port: ' comPort]);
        end

        % it takes several seconds before any operation could be attempted

        fprintf(1, 'Attempting connection ..');
        for i=1:4,
            fprintf(1, '.');
            pause(1);
        end
        fprintf(1, '\n');

        % query script type
        fwrite(a.aser, [57 57], 'uchar');
        chk=fscanf(a.aser, '%d');

        % exit if there was no answer
        if isempty(chk)
            delete(a);
            error('Connection unsuccessful, please make sure that the Arduino is ✓
```

powered on, running either adiosrv.pde or mororsrv.pde, and that the board is connected to the indicated serial port. You might also try to unplug and re-plug the USB cable before attempting a reconnection.');

```

    end

end

% check returned value
if chk==1,
    disp('Basic I/O Script detected !');
elseif chk==2,
    disp('Motor Shield Script detected !');
else
    delete(a);
    error('Unknown Script. Please make sure that either adiosrv.pde or
motorsrv.pde are running on the Arduino');
end

% sets a.mots flag
a.mots=chk-1;

% set a.aser tag
a.aser.Tag='ok';

% initialize pin vector (-1 is unassigned, 0 is input, 1 is output)
a.pins=-1*ones(1,19);

% initialize servo vector (-1 is unknown, 0 is detached, 1 is attached)
a.srvs=0*ones(1,2);

% initialize motor vector (0 to 255 is the speed)
a.mspd=0*ones(1,4);

% initialize stepper vector (0 to 255 is the speed)
a.sspd=0*ones(1,2);

% notify successful installation
disp('Arduino successfully connected !');

end % arduino

% destructor, deletes the object
function delete(a)

% if it is a serial, valid and open then close it
if isa(a.aser,'serial') && isvalid(a.aser) && strcmpi(get(a.
aser,'Status'),'open'),
    if ~isempty(a.aser.Tag),
        try
            % trying to leave it in a known unarmful state
            for i=2:19,
```

```

        a.pinMode(i, 'output');
        a.digitalWrite(i, 0);
        a.pinMode(i, 'input');
    end
catch ME
    % disp but proceed anyway
    disp(ME.message);
    disp('Proceeding to deletion anyway');
end

end
fclose(a.aser);
end

% if it's an object delete it
if isobject(a.aser),
    delete(a.aser);
end

end % delete

% disp, displays the object
function disp(a) % display
    if isvalid(a),
        if isa(a.aser, 'serial') && isvalid(a.aser),
            disp(['<a href="matlab:help arduino">arduino</a> object connected to
' a.aser.port ' port']);
            if a.mots==1,
                disp('Motor Shield Server running on the arduino board');
                disp(' ');
                a.servoStatus
                a.motorSpeed
                a.stepperSpeed
                disp(' ');
                disp('Servo Methods: <a href="matlab:help servoStatus">
>servoStatus</a> <a href="matlab:help servoAttach">servoAttach</a> <a href="matlab:help
servoDetach">servoDetach</a> <a href="matlab:help servoRead">servoRead</a> <a href="
matlab:help servoWrite">servoWrite</a>');
                disp('DC Motors and Stepper Methods: <a href="matlab:help
motorSpeed">motorSpeed</a> <a href="matlab:help motorRun">motorRun</a> <a href="matlab:
help stepperSpeed">stepperSpeed</a> <a href="matlab:help stepperStep">stepperStep</a>');
            else
                disp('IO Server running on the arduino board');
                disp(' ');
                a.pinMode
                disp(' ');
                disp('Pin IO Methods: <a href="matlab:help pinMode">pinMode</a>
<a href="matlab:help digitalRead">digitalRead</a> <a href="matlab:help digitalWrite">
>digitalWrite</a> <a href="matlab:help analogRead">analogRead</a> <a href="matlab:help
analogWrite">analogWrite</a>');
            end
        end
    end
end

```

```

        end
        disp(' ');
    else
        disp('<a href="matlab:help arduino">arduino</a> object connected to
an invalid serial port');
        disp('Please delete the arduino object');
        disp(' ');
    end
else
    disp('Invalid <a href="matlab:help arduino">arduino</a> object');
    disp('Please clear the object and instantiate another one');
    disp(' ');
end
end

% pin mode, changes pin mode
function pinMode(a,pin,str)

% a.pinMode(pin,str); specifies the pin mode of a digital pins.
% The first argument before the function name, a, is the arduino object.
% The first argument, pin, is the number of the digital pin (2 to 19).
% The second argument, str, is a string that can be 'input' or 'output',
% Called with one argument, as a.pin(pin) it returns the mode of
% the digital pin, called without arguments, prints the mode of all the
% digital pins. Note that the digital pins from 0 to 13 are located on
% the upper right part of the board, while the digital pins from 14 to 19
% are better known as "analog input" pins and are located in the lower
% right corner of the board.
%
% Examples:
% a.pinMode(11,'output') % sets digital pin #11 as output
% a.pinMode(10,'input') % sets digital pin #10 as input
% val=a.pinMode(10);    % returns the status of digital pin #10
% a.pinMode(5);        % prints the status of digital pin #5
% a.pinMode;          % prints the status of all pins
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ARGUMENT CHECKING %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% check nargin
if nargin>3,
    error('This function cannot have more than 3 arguments, object, pin and
str');
end

% first argument must be the arduino variable
if ~isa(a,'arduino'), error('The first argument must be an arduino
variable'); end

% if pin argument is there check it

```

```
if nargin>1,
    errstr=arduino.checknum(pin,'pin number',2:19);
    if ~isempty(errstr), error(errstr); end
end

% if str argument is there check it
if nargin>2,
    errstr=arduino.checkstr(str,'pin mode',{'input','output'});
    if ~isempty(errstr), error(errstr); end
end

% perform the requested action
if nargin==3,

    % check a.aser for validity
    errstr=arduino.checkser(a.aser,'valid');
    if ~isempty(errstr), error(errstr); end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% CHANGE PIN MODE ✓
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % assign value
    if lower(str(1))=='o', val=1; else val=0; end

    if strcmpi(get(a.aser,'Port'),'DEMO'),
        % handle demo mode here

        % average digital output delay
        pause(0.0087);

    else
        % do the actual action here

        % check a.aser for openness
        errstr=arduino.checkser(a.aser,'open');
        if ~isempty(errstr), error(errstr); end

        % send mode, pin and value
        fwrite(a.aser,[48 97+pin 48+val],'uchar');

    end

    % detach servo 1 or 2 if pins 10 or 9 are used
    if pin==10 || pin==9, a.servoDetach(11-pin); end

    % store 0 for input and 1 for output
    a.pins(pin)=val;

elseif nargin==2,
    % print pin mode for the requested pin
```

```

mode={'UNASSIGNED','set as INPUT','set as OUTPUT'};
disp(['Digital Pin ' num2str(pin) ' is currently ' mode{2+a.pins(pin)}]);

else
    % print pin mode for each pin

mode={'UNASSIGNED','set as INPUT','set as OUTPUT'};
for i=2:19;
    disp(['Digital Pin ' num2str(i,'%02d') ' is currently ' mode{2+a.pins
(i)}]);
end

end

end % pinmode

% digital read
function val=digitalRead(a,pin)

% val=a.digitalRead(pin); performs digital input on a given arduino pin.
% The first argument before the function name, a, is the arduino object.
% The argument pin, is the number of the digital pin (2 to 19)
% where the digital input needs to be performed. Note that the digital pins
% from 0 to 13 are located on the upper right part of the board, while the
% digital pins from 14 to 19 are better known as "analog input" pins and
% are located in the lower right corner of the board.
%
% Example:
% val=a.digitalRead(4); % reads pin #4
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ARGUMENT CHECKING %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% check nargin
if nargin~=2,
    error('Function must have the "pin" argument');
end

% first argument must be the arduino variable
if ~isa(a,'arduino'), error('The first argument must be an arduino
variable'); end

% check pin
errstr=arduino.checknum(pin,'pin number',2:19);
if ~isempty(errstr), error(errstr); end

% check a.aser for validity
errstr=arduino.checkser(a.aser,'valid');
if ~isempty(errstr), error(errstr); end

```



\*\*\*\*\* PERFORM DIGITAL INPUT \*\*\*\*\*

```
if strcmpi(get(a.aser, 'Port'), 'DEMO'),
    % handle demo mode
```

```
    % average digital input delay
    pause(0.0247);
```

```
    % output 0 or 1 randomly
    val=round(rand);
```

```
else
```

```
    % check a.aser for openness
    errstr=arduino.checkser(a.aser, 'open');
    if ~isempty(errstr), error(errstr); end
```

```
    % send mode and pin
    fwrite(a.aser, [49 97+pin], 'uchar');
```

```
    % get value
    val=fscanf(a.aser, '%d');
```

```
end
```

```
end % digitalread
```

```
% digital write
```

```
function digitalWrite(a, pin, val)
```

```
% a.digitalWrite(pin, val); performs digital output on a given pin.
% The first argument before the function name, a, is the arduino object.
% The second argument, pin, is the number of the digital pin (2 to 19)
% where the digital output needs to be performed.
% The third argument, val, is the value (either 0 or 1) for the output
% Note that the digital pins from 0 to 13 are located on the upper right part
% of the board, while the digital pins from 14 to 19 are better known as
% "analog input" pins and are located in the lower right corner of the board.
```

```
%
```

```
% Examples:
```

```
% a.digitalWrite(13,1); % sets pin #13 high
```

```
% a.digitalWrite(13,0); % sets pin #13 low
```

```
%
```

\*\*\*\*\* ARGUMENT CHECKING \*\*\*\*\*

```
% check nargin
```

```
if nargin~=3,
```

```
        error('Function must have the "pin" and "val" arguments');
    end

    % first argument must be the arduino variable
    if ~isa(a,'arduino'), error('The first argument must be an arduino
variable'); end

    % check pin
    errstr=arduino.checknum(pin,'pin number',2:19);
    if ~isempty(errstr), error(errstr); end

    % check val
    errstr=arduino.checknum(val,'value',0:1);
    if ~isempty(errstr), error(errstr); end

    % pin should be configured as output
    if a.pins(pin)~=1,
        warning('MATLAB:Arduino:digitalWrite',['If digital pin ' num2str(pin) '
is set as input, digital output takes place only after using a.pinMode(' num2str(pin)
','output')']; ']);
    end

    % check a.aser for validity
    errstr=arduino.checkser(a.aser,'valid');
    if ~isempty(errstr), error(errstr); end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% PERFORM DIGITAL OUTPUT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    if strcmpi(get(a.aser,'Port'),'DEMO'),
        % handle demo mode

        % average digital output delay
        pause(0.0087);

    else

        % check a.aser for openness
        errstr=arduino.checkser(a.aser,'open');
        if ~isempty(errstr), error(errstr); end

        % send mode, pin and value
        fwrite(a.aser,[50 97+pin 48+val],'uchar');

    end

end % digitalwrite

% analog read
function val=analogRead(a,pin)
```

```

% val=a.analogRead(pin); Performs analog input on a given arduino pin.
% The first argument before the function name, a, is the arduino object.
% The second argument, pin, is the number of the analog input pin (0 to 5)
% where the analog input needs to be performed. The returned value, val,
% ranges from 0 to 1023, with 0 corresponding to an input voltage of 0 volts,
% and 1023 to a value of 5 volts. Therefore the resolution is .0049 volts
% (4.9 mV) per unit.
% Note that the analog input pins 0 to 5 are also known as digital pins
% from 14 to 19, and are located on the lower right corner of the board.
% Specifically, analog input pin 0 corresponds to digital pin 14, and analog
% input pin 5 corresponds to digital pin 19. Performing analog input does
% not affect the digital state (high, low, digital input) of the pin.
%
% Example:
% val=a.analogRead(0); % reads analog input pin # 0
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ARGUMENT CHECKING %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% check nargin
if nargin~=2,
    error('Function must have the "pin" argument');
end

% first argument must be the arduino variable
if ~isa(a,'arduino'), error('The first argument must be an arduino
variable'); end

% check pin
errstr=arduino.checknum(pin,'analog input pin number',0:5);
if ~isempty(errstr), error(errstr); end

% check a.aser for validity
errstr=arduino.checkser(a.aser,'valid');
if ~isempty(errstr), error(errstr); end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% PERFORM ANALOG INPUT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if strcmpi(get(a.aser,'Port'),'DEMO'),
    % handle demo mode

    % average analog input delay
    pause(0.0267);

    % output a random value between 0 and 1023
    val=round(1023*rand);

else

    % check a.aser for openness

```

```

    errstr=arduino.checkser(a.aser,'open');
    if ~isempty(errstr), error(errstr); end

    % send mode and pin
    fwrite(a.aser,[51 97+pin],'uchar');

    % get value
    val=fscanf(a.aser,'%d');

end

end % analogread

% function analog write
function analogWrite(a,pin,val)

% a.analogWrite(pin,val); Performs analog output on a given arduino pin.
% The first argument before the function name, a, is the arduino object.
% The first argument, pin, is the number of the DIGITAL pin where the analog
% (PWM) output needs to be performed. Allowed pins for AO are 3,5,6,9,10,11
% The second argument, val, is the value from 0 to 255 for the level of
% analog output. Note that the digital pins from 0 to 13 are located on the
% upper right part of the board.
%
% Examples:
% a.analogWrite(11,90); % sets pin #11 to 90/255
% a.analogWrite(3,10); % sets pin #3 to 10/255
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ARGUMENT CHECKING %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% check nargin
if nargin~=3,
    error('Function must have the "pin" and "val" arguments');
end

% first argument must be the arduino variable
if ~isa(a,'arduino'), error('The first argument must be an arduino
variable'); end

% check pin
errstr=arduino.checknum(pin,'pwm pin number',[3 5 6 9 10 11]);
if ~isempty(errstr), error(errstr); end

% check val
errstr=arduino.checknum(val,'analog output level',0:255);
if ~isempty(errstr), error(errstr); end

% pin should be configured as output
if a.pins(pin)~=1,
    warning('MATLAB:Arduino:analogWrite',['If digital pin ' num2str(pin) ' is

```

```
set as input, pwm output takes place only after using a.pinMode(' num2str(pin)
,'output'); ']);
end

% check a.aser for validity
errstr=arduino.checkser(a.aser,'valid');
if ~isempty(errstr), error(errstr); end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% PERFORM ANALOG OUTPUT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if strcmpi(get(a.aser,'Port'),'DEMO'),
    % handle demo mode

    % average analog output delay
    pause(0.0088);

else

    % check a.aser for openness
    errstr=arduino.checkser(a.aser,'open');
    if ~isempty(errstr), error(errstr); end

    % send mode, pin and value
    fwrite(a.aser,[52 97+pin val],'uchar');

end

end % analogwrite

% servo attach
function servoAttach(a,num)

% a.servoAttach(num); attaches a servo to the corresponding pwm pin.
% The first argument before the function name, a, is the arduino object.
% The second argument, num, is the number of the servo, which can be either 1
% (top servo, uses digital pin 10 for pwm), or 2 (bottom servo, uses digital
% pin 9 for pwm). Returns Random results if motor shield is not connected.
%
% Example:
% a.servoAttach(1); % attach servo #1
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ARGUMENT CHECKING %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% check nargin
if nargin~=2,
    error('Function must have the "num" argument');
end
```

```

    % first argument must be the arduino variable
    if ~isa(a,'arduino'), error('The first argument must be an arduino
variable'); end

    % check servo number
    errstr=arduino.checknum(num,'servo number',[1 2]);
    if ~isempty(errstr), error(errstr); end

    % check a.aser for validity
    errstr=arduino.checkser(a.aser,'valid');
    if ~isempty(errstr), error(errstr); end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ATTACH SERVO %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    if strcmpi(get(a.aser,'Port'),'DEMO') || a.mots==0,
        % handle demo mode

        % average digital output delay
        pause(0.0087);

    else

        % check a.aser for openness
        errstr=arduino.checkser(a.aser,'open');
        if ~isempty(errstr), error(errstr); end

        % send mode, num and value (1 for attach)
        fwrite(a.aser,[54 96+num 48+1],'uchar');

    end

    % store the servo status
    a.srvs(num)=1;

    % update pin status to unassigned
    a.pins(11-num)=-1;

end % servoattach

% servo detach
function servoDetach(a,num)

    % a.servoDetach(num); detaches a servo from its corresponding pwm pin.
    % The first argument before the function name, a, is the arduino object.
    % The second argument, num, is the number of the servo, which can be either 1
    % (top servo, uses digital pin 10 for pwm), or 2 (bottom servo, uses digital
    % pin 9 for pwm). Returns random results if motor shield is not connected.
    %
    % Examples:

```

```

% a.servoDetach(1); % detach servo #1
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ARGUMENT CHECKING %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% check nargin
if nargin~=2,
    error('Function must have the "num" argument');
end

% first argument must be the arduino variable
if ~isa(a,'arduino'), error('The first argument must be an arduino
variable'); end

% check servo number
errstr=arduino.checknum(num,'servo number',[1 2]);
if ~isempty(errstr), error(errstr); end

% check a.aser for validity
errstr=arduino.checkser(a.aser,'valid');
if ~isempty(errstr), error(errstr); end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DETACH SERVO %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if strcmpi(get(a.aser,'Port'),'DEMO') || a.mots==0,
    % handle demo mode

    % average digital output delay
    pause(0.0087);

else

    % check a.aser for openness
    errstr=arduino.checkser(a.aser,'open');
    if ~isempty(errstr), error(errstr); end

    % send mode, num and value (0 for detach)
    fwrite(a.aser,[54 96+num 48+0],'uchar');

end

a.srvs(num)=0;

end % servodetach

% servo status
function val=servoStatus(a,num)

% a.servoStatus(num); Reads the status of a servo (attached/detached)
% The first argument before the function name, a, is the arduino object.

```







```
% check nargin
if nargin~=2,
    error('Function must have the servo number argument');
end

% first argument must be the arduino variable
if ~isa(a,'arduino'), error('The first argument must be an arduino
variable'); end

% check servo number
errstr=arduino.checknum(num,'servo number',[1 2]);
if ~isempty(errstr), error(errstr); end

% check status
if a.srvs(num)~=1,
    error(['Servo ' num2str(num) ' is not attached, please use a.servoAttach
(' num2str(num) ') to attach it']);
end

% check a.aser for validity
errstr=arduino.checkser(a.aser,'valid');
if ~isempty(errstr), error(errstr); end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% READ SERVO ANGLE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if strcmpi(get(a.aser,'Port'),'DEMO') || a.mots==0,
    % handle demo mode

    % average analog input delay
    pause(0.0267);

    % output a random value between 0 and 180
    val=round(180*rand);

else

    % check a.aser for openness
    errstr=arduino.checkser(a.aser,'open');
    if ~isempty(errstr), error(errstr); end

    % send mode and num
    fwrite(a.aser,[55 96+num],'uchar');

    % get value
    val=fscanf(a.aser,'%d');

end

end % servoread
```

```

% servo write
function servoWrite(a,num,val)

% a.servoWrite(num,val); writes an angle on a given servo.
% The first argument before the function name, a, is the arduino object.
% The second argument, num, is the number of the servo, which can be
% either 1 (top servo, uses digital pin 10 for pwm), or 2 (bottom servo,
% uses digital pin 9 for pwm). The third argument is the angle in degrees,
% typically from 0 to 180. Returns Random results if motor shield is not
% connected.
%
% Example:
% a.servoWrite(1,45); % rotates servo #1 of 45 degrees
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ARGUMENT CHECKING %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% check nargin
if nargin~=3,
    error('Function must have the servo number and angle arguments');
end

% first argument must be the arduino variable
if ~isa(a,'arduino'), error('The first argument must be an arduino
variable'); end

% check servo number
errstr=arduino.checknum(num,'servo number',[1 2]);
if ~isempty(errstr), error(errstr); end

% check angle value
errstr=arduino.checknum(val,'angle',0:180);
if ~isempty(errstr), error(errstr); end

% check status
if a.srvs(num)~=1,
    error(['Servo ' num2str(num) ' is not attached, please use a.servoAttach
(' num2str(num) ') to attach it']);
end

% check a.aser for validity
errstr=arduino.checkser(a.aser,'valid');
if ~isempty(errstr), error(errstr); end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% WRITE ANGLE TO SERVO %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if strcmpi(get(a.aser,'Port'),'DEMO') || a.mots==0,
    % handle demo mode

```

```

    % average analog output delay
    pause(0.0088);

else

    % check a.aser for openness
    errstr=arduino.checkser(a.aser,'open');
    if ~isempty(errstr), error(errstr); end

    % send mode, num and value
    fwrite(a.aser,[56 96+num val],'uchar');

end

end % servowrite

% motor speed
function val=motorSpeed(a,num,val)

% val=a.motorSpeed(num,val); sets the speed of a DC motor.
% The first argument before the function name, a, is the arduino object.
% The second argument, num, is the number of the motor, which can go
% from 1 to 4 (the motor ports are numbered on the motor shield).
% The third argument is the speed from 0 (stopped) to 255 (maximum), note
% that depending on the motor speeds of at least 60 might be necessary
% to actually run it. Called with one argument, as a.motorSpeed(num),
% it returns the speed at which the given motor is set to run. If there
% is no output argument it prints the speed of the motor.
% Called without arguments, itprints the speed of each motor.
% Note that you must use the command a.motorRun to actually run
% the motor at the given speed, either forward or backwards.
% Returns Random results if motor shield is not connected.
%
% Examples:
% a.motorSpeed(4,200)      % sets speed of motor 4 as 200/255
% val=a.motorSpeed(1);    % returns the speed of motor 1
% a.motorSpeed(3);        % prints the speed of motor 3
% a.motorSpeed;           % prints the speed of all motors
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ARGUMENT CHECKING %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% check nargin
if nargin>3,
    error('This function cannot have more than 3 arguments, arduino object,
motor number and speed');
end

% first argument must be the arduino variable
if ~isa(a,'arduino'), error('The first argument must be an arduino
variable'); end

```

```
% if motor number is there check it
if nargin>1,
    errstr=arduino.checknum(num,'motor number',1:4);
    if ~isempty(errstr), error(errstr); end
end

% if speed argument is there check it
if nargin>2,
    errstr=arduino.checknum(val,'speed',0:255);
    if ~isempty(errstr), error(errstr); end
end

% perform the requested action
if nargin==3,

    % check a.aser for validity
    errstr=arduino.checkser(a.aser,'valid');
    if ~isempty(errstr), error(errstr); end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SET MOTOR SPEED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    if strcmpi(get(a.aser,'Port'),'DEMO') || a.mots==0,
        % handle demo mode

        % average analog output delay
        pause(0.0088);

    else

        % check a.aser for openess
        errstr=arduino.checkser(a.aser,'open');
        if ~isempty(errstr), error(errstr); end

        % send mode, num and value
        fwrite(a.aser,[65 48+num val],'uchar');

    end

    % store speed value in case it needs to be retrieved
    a.mspd(num)=val;

    % clear val if is not needed as output
    if nargout==0,
        clear val;
    end

elseif nargin==2,

    if nargout==0,
        % print speed value
```

```

        disp(['The speed of motor number ' num2str(num) ' is set to: '✓
num2str(a.mspd(num)) ' over 255']);
    else
        % return speed value
        val=a.mspd(num);
    end

else

    if nargin==0,
        % print speed value for each motor
        for num=1:4,
            disp(['The speed of motor number ' num2str(num) ' is set to: '✓
num2str(a.mspd(num)) ' over 255']);
        end
    else
        % return speed values
        val=a.mspd;
    end

end

end % motorspeed

% motor run
function motorRun(a,num,str)

% a.motorRun(num,str); runs a given DC motor.
% The first argument before the function name, a, is the arduino object.
% The second argument, num, is the number of the motor, which can go
% from 1 to 4 (the motor ports are numbered on the motor shield).
% The third argument, str, is a string that can be 'forward' (runs the
% motor forward) 'backward' (runs the motor backward) or 'release',
% (stops the motor). Returns Random results if motor shield is not
% connected.
%
% Examples:
% a.motorRun(1,'forward');      % runs motor 1 forward
% a.motorRun(3,'backward');    % runs motor 3 backward
% a.motorRun(1,'release');     % release motor 1
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ARGUMENT CHECKING %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% check nargin
if nargin~=3,
    error('Function must have 3 arguments, object, motor number and✓
direction');
end

% first argument must be the arduino variable

```

```

% check motor number
errstr=arduino.checknum(num,'motor number',1:4);
if ~isempty(errstr), error(errstr); end

% check direction
errstr=arduino.checkstr(str,'direction',{'forward','backward','release'});
if ~isempty(errstr), error(errstr); end

% check a.aser for validity
errstr=arduino.checkser(a.aser,'valid');
if ~isempty(errstr), error(errstr); end

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% RUN THE MOTOR %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

if strcmpi(get(a.aser,'Port'),'DEMO') || a.mots ==0,
    % handle demo mode

    % average analog output delay
    pause(0.0088);

```

```

else

```

```

    % check a.aser for openness
    errstr=arduino.checkser(a.aser,'open');
    if ~isempty(errstr), error(errstr); end

    % send mode, num and value
    fwrite(a.aser,[66 48+num abs(str(1))],'uchar');

```

```

end

```

```

end % motorrn

```

```

% stepper speed
function val=stepperSpeed(a,num,val)

```

```

% val=a.stepperSpeed(num,val); sets the speed of a given stepper motor
% The first argument before the function name, a, is the arduino object.
% The second argument, num, is the number of the stepper motor,
% which can go from 1 to 4 (the motor ports are numbered on the motor
shield).
% The third argument is the RPM speed from 1 (minimum) to 255 (maximum).
% Called with one argument, as a.stepperSpeed(num), it returns the
% speed at which the given motor is set to run. If there is no output
% argument it prints the speed of the stepper motor.
% Called without arguments, itprints the speed of each stepper motor.
% Note that you must use the command a.stepperStep to actually run

```

```
if ~isa(a,'arduino'). error('The first argument must be an arduino')✓
```



```

% the motor at the given speed, either forward or backwards (or release
% it). Returns Random results if motor shield is not connected.
%
% Examples:
% a stepperSpeed(2,50) % sets speed of stepper 2 as 50 rpm
% val=a stepperSpeed(1); % returns the speed of stepper 1
% a stepperSpeed(2); % prints the speed of stepper 2
% a stepperSpeed; % prints the speed of both steppers
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ARGUMENT CHECKING %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% check nargin
if nargin>3,
    error('This function cannot have more than 3 arguments, object, stepper
number and speed');
end

% first argument must be the arduino variable
if ~isa(a,'arduino'), error('The first argument must be an arduino
variable'); end

% if stepper number is there check it
if nargin>1,
    errstr=arduino.checknum(num,'stepper number',1:2);
    if ~isempty(errstr), error(errstr); end
end

% if speed argument is there check it
if nargin>2,
    errstr=arduino.checknum(val,'speed',0:255);
    if ~isempty(errstr), error(errstr); end
end

% perform the requested action
if nargin==3,

    % check a.aser for validity
    errstr=arduino.checkser(a.aser,'valid');
    if ~isempty(errstr), error(errstr); end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% PERFORM ANALOG OUTPUT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    if strcmpi(get(a.aser,'Port'),'DEMO') || a.mots==0,
        % handle demo mode

        % average analog output delay
        pause(0.0088);

    else

```

```
        % check a.aser for openness
        errstr=arduino.checkser(a.aser,'open');
        if ~isempty(errstr), error(errstr); end

        % send mode, num and value
        fwrite(a.aser,[67 48+num val],'uchar');

    end

    % store speed value in case it needs to be retrieved
    a.sspd(num)=val;

    % clear val if is not needed as output
    if nargout==0,
        clear val;
    end

elseif nargin==2,

    if nargout==0,
        % print speed value
        disp(['The speed of stepper number ' num2str(num) ' is set to: '✓
num2str(a.sspd(num)) ' over 255']);
    else
        % return speed value
        val=a.sspd(num);
    end

else

    if nargout==0,
        % print speed value for each stepper
        for num=1:2,
            disp(['The speed of stepper number ' num2str(num) ' is set to: '✓
num2str(a.sspd(num)) ' over 255']);
        end
    else
        % return speed values
        val=a.sspd;
    end

end

end % stepperspeed

% stepper step
function stepperStep(a,num,dir,sty,steps)

% a.stepperStep(num,dir,sty,steps); rotates a given stepper motor
% The first argument before the function name, a, is the arduino object.
```

```

% The second argument, num, is the number of the stepper motor, which is
% either 1 or 2. The third argument, the direction, is a string that can
% be 'forward' (runs the motor forward) 'backward' (runs the motor backward)
% or 'release', (stops and releases the motor). Unless the direction is
% 'release', then two more argument are needed: the fourth one is the style,
% which is a string specifying the style of the motion, and can be 'single'
% (only one coil activated at a time), 'double' (2 coils activated, gives
% an higher torque and power consumption) 'inteleave', (alternates between
% single and double to get twice the resolution and half the speed), and
% 'microstep' (the coils are driven in PWM for a smoother motion).
% The final argument is the number of steps that the motor has
% to complete.
% Returns Random results if motor shield is not connected.
%
% Examples:
% % rotates stepper 1 forward of 100 steps in interleave mode
% a stepperStep(1, 'forward', 'double', 100);
% % rotates stepper 2 forward of 50 steps in double mode
% a stepperStep(1, 'forward', 'double', 50);
% % rotates stepper 2 backward of 50 steps in single mode
% a stepperStep(2, 'backward', 'single', 50);
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ARGUMENT CHECKING %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% check nargin
if nargin>5 || nargin <3,
    error('Function must have at least 3 and no more than 5 arguments');
end

% first argument must be the arduino variable
if ~isa(a, 'arduino'), error('The first argument must be an arduino
variable'); end

% check stepper number
errstr=arduino.checknum(num, 'stepper number', 1:2);
if ~isempty(errstr), error(errstr); end

% check direction
errstr=arduino.checkstr(dir, 'direction', {'forward', 'backward', 'release'});
if ~isempty(errstr), error(errstr); end

% if it is not released must have all arguments
if ~strcmpi(dir, 'release') && nargin~=5,
    error('Either the motion style or the number of steps are missing');
end

% can't move forward or backward if speed is set to zero
if ~strcmpi(dir, 'release') && a stepperSpeed(num)<1,
    error('The stepper speed has to be greater than zero for the stepper to
move');

```

```

end

% check motion style
if nargin>3,
    % check direction
    errstr=arduino.checkstr(sty,'motion style',
{'single','double','interleave','microstep'});
    if ~isempty(errstr), error(errstr); end
else
    sty='single';
end

% check number of steps
if nargin==5,
    errstr=arduino.checknum(steps,'number of steps',0:255);
    if ~isempty(errstr), error(errstr); end
else
    steps=0;
end

% check a.aser for validity
errstr=arduino.checkser(a.aser,'valid');
if ~isempty(errstr), error(errstr); end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ROTATE THE STEPPER %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if strcmpi(get(a.aser,'Port'),'DEMO') || a.mots==0,
    % handle demo mode

    % average analog output delay
    pause(0.0088);

else

    % check a.aser for openness
    errstr=arduino.checkser(a.aser,'open');
    if ~isempty(errstr), error(errstr); end

    % send mode, num and value
    fwrite(a.aser,[68 48+num abs(dir(1)) abs(sty(1)) steps],'uchar');

end

end % stepperstep

end % methods

methods (Static) % static methods

function errstr=checknum(num,description,allowed)

```

```
% errstr=arduino.checknum(num,description,allowed); Checks numeric argument.
% This function checks the first argument, num, described in the string
% given as a second argument, to make sure that it is real, scalar,
% and that it is equal to one of the entries of the vector of allowed
% values given as a third argument. If the check is successful then the
% returned argument is empty, otherwise it is a string specifying
% the type of error.

% preliminary: check nargin
if nargin~=3,
    error('checknum needs 3 arguments, please read the help');
end

% preliminary: check description
if isempty(description) || ~ischar(description)
    error('checknum second argument must be a string');
end

% preliminary: check allowed
if isempty(allowed) || ~isnumeric(allowed)
    error('checknum third argument must be a numeric vector');
end

% initialize error string
errstr=[];

% check num for type
if ~isnumeric(num),
    errstr=['The ' description ' must be numeric'];
    return
end

% check num for size
if numel(num)~=1,
    errstr=['The ' description ' must be a scalar'];
    return
end

% check num for realness
if ~isreal(num),
    errstr=['The ' description ' must be a real value'];
    return
end

% check num against allowed values
if ~any(allowed==num),

    % form right error string
    if numel(allowed)==1,
        errstr=['Unallowed value for ' description ', the value must be ✓
```

```

xactly ' num2str(allowed(1))];
    elseif numel(allowed)==2,
        errstr=['Unallowed value for ' description ', the value must be
either ' num2str(allowed(1)) ' or ' num2str(allowed(2))];
    elseif max(diff(allowed))==1,
        errstr=['Unallowed value for ' description ', the value must be an
integer going from ' num2str(allowed(1)) ' to ' num2str(allowed(end))];
    else
        errstr=['Unallowed value for ' description ', the value must be one
of the following: ' mat2str(allowed)];
    end

```

```
end
```

```
end % checksum
```

```
function errstr=checkstr(str,description,allowed)
```

```

% errstr=arduino.checkstr(str,description,allowed); Checks string argument.
% This function checks the first argument, str, described in the string
% given as a second argument, to make sure that it is a string, and that
% its first character is equal to one of the entries in the cell of
% allowed characters given as a third argument. If the check is successful
% then the returned argument is empty, otherwise it is a string specifying
% the type of error.

```

```
% preliminary: check nargin
```

```

if nargin~=3,
    error('checkstr needs 3 arguments, please read the help');
end

```

```
% preliminary: check description
```

```

if isempty(description) || ~ischar(description)
    error('checknum second argument must be a string');
end

```

```
% preliminary: check allowed
```

```

if ~iscell(allowed) || numel(allowed)<2,
    error('checknum third argument must be a cell with at least 2 entries');
end

```

```
% initialize error string
```

```
errstr=[];
```

```
% check string for type
```

```

if ~ischar(str),
    errstr=['The ' description ' argument must be a string'];
    return
end

```

```
% check string for size
```

```

if numel(str)<1,
    errstr=['The ' description ' argument cannot be empty'];
    return
end

% check str against allowed values
if ~any(strcmpi(str,allowed)),

    % make sure this is a horizontal vector
    allowed=allowed(:)';

    % add a comma at the end of each value
    for i=1:length(allowed)-1,
        allowed{i}=['' allowed{i} '', '];
    end

    % form error string
    errstr=['Unallowed value for ' description ', the value must be either: '
allowed{1:end-1} 'or '' allowed{end} '''];
    return
end

end % checkstr

function errstr=checkser(ser,chk)

% errstr=arduino.checkser(ser,chk); Checks serial connection argument.
% This function checks the first argument, ser, to make sure that either:
% 1) it is a valid serial connection (if the second argument is 'valid')
% 3) it is open (if the second argument is 'open')
% If the check is successful then the returned argument is empty,
% otherwise it is a string specifying the type of error.

% preliminary: check nargin
if nargin~=2,
    error('checkser needs two arguments, please read the help');
end

% initialize error string
errstr=[];

% check serial connection
switch lower(chk),

    case 'valid',

        % make sure is a serial port
        if ~isa(ser,'serial'),
            disp('Arduino is not connected, please re-create the object
before using this function. ');
            errstr='Arduino not connected';

```

```
        return
    end

    % make sure is valid
    if ~isvalid(ser),
        disp('Serial connection invalid, please recreate the object to
reconnect to a serial port. ');
        errstr='Serial connection invalid';
        return
    end

    case 'open',

        % check openess
        if ~strcmpi(get(ser,'Status'),'open'),
            disp('Serial connection not opened, please recreate the object to
reconnect to a serial port. ');
            errstr='Serial connection not opened';
            return
        end

    otherwise

        % complain
        error('second argument must be either ''valid'' or ''open''');

    end

end % chackser

end % static methods

end % class def
```



```

function [x,y]=cabpath()
%
%GENERATE THE CAB PATH
%
%General path starts with a straightway that heads due north (+y dir), then
%turns left or right along a circular arc of specified angle (turnang), and
%finally continues along another straightway (equal in length to the
%first straightaway)
%
%Other Notes:
% 1. Points are always equally spaced along the arc
% 2. Spacing along straightaways can increase in proportion to distance
%    from ends of arc
% 3. The code probably will not work well for turn angles close to
%    zero (corresponding to a straight path)
% 4. The input variable "turnang" should always be given a positive
%    value.
% 5. I have not yet loaded the path coordinates and headings into a
%    matrix. These values are currently stored in the vectors: x,y,theta
%
%
%inputs
%
dir=1;           %flag to indicate right(dir=1) or left (dir=2) turn
rho=30.;        %radius of curvature of turn
turnang=pi/2.; %turn angle (e.g. =pi/2 for 90 deg turn; =pi for U-turn)
narc=200;       %no of equally spaced intervals along the arc
nstr=10;        %no of intervals along each straightaway
rate=1.1;       %rate at which spacing increases on straightaways at either end of arc)
%
%calculate length of each interval along arc
%
arcang=turnang;
arclength=arcang*rho;
ds=arclength/narc;
%
%generate positional coordinates along first straight away
%(Note: origin is at the end of this first straightaway)
%
theta(nstr+1)=0.;
x(nstr+1)=0.;
y(nstr+1)=0.;
dy=ds;
for i=nstr:-1:1
    dy=1.1*dy;
    theta(i)=0.;
    x(i)=0.;
    y(i)=y(i+1)-dy;
end
%
```

```
%generate positional coordinates along arc
%
i=nstr+1;
delang=turnang/narc;
for j=1:narc
    i=i+1;
    theta(i)=j*delang;
    x(i)=rho-rho*cos(theta(i));
    y(i)=rho*sin(theta(i));
end
%
%generate positional coordinates along second straight away
%
angsav=theta(i);
dx=ds*sin(angsav);
dy=ds*cos(angsav);
for j=1:nstr
    dx=1.1*dx;
    dy=1.1*dy;
    i=i+1;
    theta(i)=angsav;
    x(i)=x(i-1)+dx;
    y(i)=y(i-1)+dy;
end
%
%modify signs for left turn
%
imax=i;
if dir==2
    for j=1:imax
        x(j)=-x(j);
        theta(j)=-theta(j);
    end
end
%
%plot path
%
xmin=min(x);
xmax=max(x);
ymin=min(y);
ymax=max(y);
```



```
function [bval] = calcIndex(dbPos,Matrix,bPosX,bPosY,index);
%index=Msearch(bPosX,bPosY,Matrix); %The bogie has a position, but it needs
%to referene a position point of the robot. The way my algorith works is
%it needs co sart at a point on the robot's path, then look a calculated
%number of indicies forward. That is the aiming point. But the bogie is
%notnecessarily on the robot's path. This function searhes the Matrix for
%the closest point on the robot's path to the bogie's position. It then
%gives the index number for that point. From now on the bogie is considered
%to be where the robot was at that index.
%ff=ceil(10/(dbPos+.00001)); % How many indicies to look ahead "fudge factor"
ff=0;
bval=index-ff;%bval is the index to look for.
if bval < 1
    bval = 1;
else
end
```

```
function [mouse,hdg,ra,ba,drPos]= GetSensorVals(mouse0,a)
timerObj = timer('TimerFcn',@timerCallback,'Period',1);
start(timerObj);
mouse=timerCallback(); %the preceding code retrieves the mouse position on the screen.
dmouse=(mouse-mouse0);
if dmouse < -800 %the screen is 900 pixels tall, if the mouse reaches the top of the screen it jumps to the bottom
code calculates the distance traveled. %if the mouse jumps then this
    drPos=(900-mouse0)+mouse;
if dmouse > 800 % this is if it jumps in reverse.
    drPos=mouse0-(900-mouse);
else %normal operation, if no jump occurs.
    drPos=mouse-mouse0;
drPos=drPos/16400; %pixels to meters conversion
ra=a.analogRead(0); % arduino class functions to retrieve potentiometers.
ra=(ra-3.4025)*58; %voltage to degrees conversion,
ba=a.analogRead(1);
ba=(ba-1.5)*100;
sa=a.analogRead(1);
sa=(sa-2.5)*7.2;
hdg=a.digitalRead(3) %reads compass heading in degrees
```

```
function [Mat,ind,rep] = matSetup(Mat,ind,rep,drPos)

    if ind == rep %matrix reaches full length
        Mat(0:99,:)= Mat((rep-100):rep,:); %Moves last 100 indicies to the top.
        Mat(100:rep,:) = 0; %Deletes the rest.
        ind=100; %starts at first empty index.

    else

        if drPos == 0 % if robot is not moving,do not fill up matrix

            else

                ind=ind+1; %advances an index. if robot is moving

            end

        end

    end

end
```

```
function [i1] = Msearch(bPosX,bPosY,Matrix)
x = Matrix(:,1);
y = Matrix(:,2);
dt = DelaunayTri(x,y);
qrypts = [bPosX bPosY];
pid = nearestNeighbor(dt, qrypts);
```

```
function [rPosX,rPosY,rHDG,drPos]= robotSim(index,X,Y,rPosX0,rPosY0)
rPosX=X(index); % sets cab positions from the points that came out of "cabpath"
rPosY=Y(index);
drPosX=rPosX-rPosX0;
drPosY=rPosY-rPosY0;
rHDG=atand(drPosX/drPosY);
drPos=sqrt(drPosX^2+drPosY^2);
```



```
function setBsteering(bVals,sa)
```

```
% This code is very simple, if the steering is at a different angle than  
% required, it goes full one way until the steering is at the right  
% position. Just full one way-full other way-stop operation.
```

```
if bVals > sa
```

```
{  
    a.analogWrite(2,0); %arduino functions, sets motor power.  
}
```

```
if bVals < sa
```

```
{  
    a.analogWrite(2,255);  
}
```

```
else
```

```
{  
    a.analogWrite(2,127);  
}
```

```
function sa=steeringSim(bVals,sa,tstep)

% This code is very simple, if the steering is at a different angle than
% required, it goes full one way until the steering is at the right
% position. Just full one way-full other way-stop operation.
if bVals > sa

    dir=(18/2)*tstep; % rate of steering angle change. (degrees/second)

elseif bVals < sa

    dir=-(18/2)*tstep;

else

    dir=0;

end
saMAX=10; % sa is the steering angle of bogie. The if statement below just keeps it from
turning beyond the physical limits.
if sa <= -saMAX
    sa = -saMAX;
elseif sa >= saMAX
    sa = saMAX;
else
    sa=sa+dir;% dttime is the time since the last iteration. dir is the rate of change of
the steering angle. This takes the original steering angle and gets a new one.
end
```

```
function sa=steeringSim(bVals,sa,tstep)
```

```
% This code is very simple, if the steering is at a different angle than  
% required, it goes full one way until the steering is at the right  
% position. Just full one way-full other way-stop operation.  
if bVals > sa
```

```
    dir=(18/2)*tstep; % rate of steering angle change. (degrees/second)
```

```
elseif bVals < sa
```

```
    dir=- (18/2)*tstep;
```

```
else
```

```
    dir=0;
```

```
end  
saMAX=10; % sa is the steering angle of bogie. The if statement below just keeps it from  
turning beyond the physical limits.
```

```
if sa <= -saMAX
```

```
    sa = -saMAX;
```

```
elseif sa >= saMAX
```

```
    sa = saMAX;
```

```
else
```

```
    sa=sa+dir;% dtime is the time since the last iteration. dir is the rate of change of  
the steering angle. This takes the original steering angle and gets a new one.
```

```
end
```

```
function [bPosX,bPosY,dbPos,bHDG]= bogieUpdate(rPosX,rPosY,hdg,ba,bPosX0,bPosY0)

    bPosX=rPosX+(5*sin(hdg)); %Calculating bogie position,
    bPosY=rPosY-(5*cos(hdg));
    bHDG=hdg+ba; %Calculating trailer angle. Same formula used for cab heading.
    dbPosX=bPosX-bPosX0;
    dbPosY=bPosY-bPosY0;
    dbPos=sqrt((dbPosX^2)+(dbPosY^2)); %Calculating the absolute velocity of the bogie.
```

```
function [rPosX,rPosY,rHDG]= robotUpdate(drPos,hdg,rPosX0,rPosY0,rHDG0,ra);
```

```
:HDG=hdg+ra; %the compass is located on the trailer, not the cab, so the cab's  
%heading is the compass heading plus the angle of the potentiometer at the pivot point.  
rHDGAvg=(rHDG-rHDG0)/2; %Average heading of the robot  
rPosX=rPosX0+(drPos*sind(rHDGAvg)); %calculating x-y movements.  
rPosY=rPosY0+(drPos*cosd(rHDGAvg));
```

```
function [mouse,hdg,ra,ba,drPos]= GetSensorVals(mouse0,a)
timerObj = timer('TimerFcn',@timerCallback,'Period',1);
start(timerObj);
mouse=timerCallback(); %the preceding code retrieves the mouse position on the screen.
dmouse=(mouse-mouse0);
if dmouse < -800 %the screen is 900 pixels tall, if the mouse reaches the top of the screen it jumps to the bottom
code calculates the distance traveled. %if the mouse jumps then this
    drPos=(900-mouse0)+mouse;
if dmouse > 800 % this is if it jumps in reverse.
    drPos=mouse0-(900-mouse);
else %normal operation, if no jump occurs.
    drPos=mouse-mouse0;
drPos=drPos/16400; %pixels to meters conversion
ra=a.analogRead(0); % arduino class functions to retrieve potentiometers.
ra=(ra-3.4025)*58; %voltage to degrees conversion,
ba=a.analogRead(1);
ba=(ba-1.5)*100;
sa=a.analogRead(1);
sa=(sa-2.5)*7.2;
hdg=a.digitalRead(3) %reads compass heading in degrees
```

```
function y= timerCallback()
% this function is executed every time the timer object triggers

% read the coordinates
coords = get(0,'PointerLocation');
% print the coordinates to screen
fprintf('x: %4i y: %4i\n',coords)
y=coords(2);

end
```

```

function [bPosX,bPosY,bHDG,dbPos,sa,O2X,O2Y]= bogieSim(rPosX,rPosY,rHDG,drPos,sa,rPosX0,
rPosY0,rHDG0,bPosX0,bPosY0,bHDG0)

% the first section of the simulation calculates the bogie's center of
% curvature, point O2.

rPos=[rPosX,rPosY];% rPos is the cab's current position
rVec=[rPosX-rPosX0,rPosY-rPosY0]; % rVec is the vector from the cab's initial position to
the new position.
rVecU=rVec/norm(rVec); %unit vector
rVecM=norm(rVec); %magnitude.
R3=5; %R3 is the trailer length in meters.
hdg=atand((rPosX0-bPosX0)/(rPosY0-bPosY0)); % this is the trailer heading.
if sa > 0 % a steering anlge of zero is straight ahead. negative is one way and positive
is the other.
    R2=.23/sind(abs(sa)); % this calculateds the perpendicuar distance from the bogie to
it's center of curvature. i.e., the length of the "phantom" link.
    O2X=bPosX0-R2*sind(90-bHDG0);
    O2Y=bPosY0+R2*cosd(90-bHDG0);
    O2=[O2X,O2Y]; % O2 is the pin that the bogie turns around, that is, the center of
it's radius of curvature.
elseif sa < 0 %positive and negative steering angles put O2 on different sides of the
bogey.
    R2=.23/sind(abs(sa));
    O2X=bPosX0+R2*sind(90-bHDG0);
    O2Y=bPosY0-R2*cosd(90-bHDG0);
    O2=[O2X,O2Y];
else
    R2=1000; % this is a bit of a hand wave. At a steering anlge of zero the radius of
curvature is infinite. I just say it is really big.
    O2X=bPosX0-R2*sind(90-bHDG0);
    O2Y=bPosY0+R2*cosd(90-bHDG0);
    O2=[O2X,O2Y];
end

% now that point O2 is found, the cab is moved to it's new location and
% the new bogie position is calculated.

% it does this by creating a circle of length R3 and center at the cab's new position,
and a circle of length R2 with O2 the center point.
% it then solves for the two points where those circles intersect. those
% are the two possible bogie positions.

dVec=[rPosX-O2X,rPosY-O2Y]; % Vector from O2 to the new robot position.
D=norm(dVec);
a=(R2^2-R3^2+D^2)/(2*D); %this is the distance along dVec between O2 and the point where
the perpendicular bisector, h intersects dVec.
h=sqrt(abs(R2^2-a^2)); % h is the normal distance from dVec to the bogie's position, i.e.

```



```
, where R3 and R2 intersect.
% it is this value that is getting fouled up as "a" occasionally becomes
greater than R2 for unknown reasons.
P2=O2+(a.*(rPos-O2))./D; % P2 is the point where dVec and h intersect.
bPosX1=P2(1)+h*(rPosY-O2Y)/D; % there are two places there these circles cross, one on ✓
either side of the trailer. this calculates the position of both points.
bPosY1=P2(2)-h*(rPosX-O2X)/D;
bPosX2=P2(1)-h*(rPosY-O2Y)/D;
bPosY2=P2(2)+h*(rPosX-O2X)/D;
if sa > 0 % there are two points the could be the bogie position, but only one real ✓
position. This if statement chooses which one is correct based on the steer angle.
    bPosX=bPosX2;
    bPosY=bPosY2;
elseif sa < 0
    bPosX=bPosX1;
    bPosY=bPosY1;
else
    bPosX=bPosX2;
    bPosY=bPosY2;
end
dbPosX=bPosX-bPosX0; %caculates the scalar changes in position.
dbPosY=bPosY-bPosY0;
dbPos=((dbPosX)^2+(dbPosY)^2)^.5;
bMHDG=atand(dbPosX/dbPosY);
ang=bHDG0-bMHDG;
bHDG=bMHDG-ang;
```

```
% commented out functions are the 'real' things. because we are just running
% a performance simulation, some functions are not needed or relevant. But
% in the final test they will be.
```

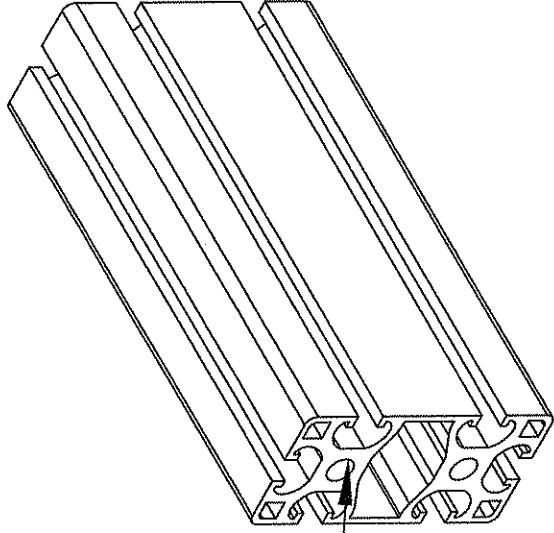
```
clear
clc
%import arduino
%a=arduino('COM3')
screenH=1600;
ra=0;
ba=0;
sa=0;
index=0;
[X,Y]=cabpath(); % runs your cabpath function, which I modified a bit.
X=X';
Y=Y';
sa=0;
dir=0;
pause on
index=2;
rPosX0=X(1);
rPosY0=Y(1);
rPosX=X(2); % sets cab positions from the points that came out of "cabpath"
rPosY=Y(2);
rHDG0=0;
drPos=0;
tstep=.08;
repeat=max(size(X))+1;
bPosX0=0;
bPosY0=rPosY-5;
bPosX=0;
bPosY=rPosY-5;
bHDG=0;
bHDG0=0;
dbPos=0;
O2X=0;
O2Y=bPosY;
while index<200 % stops the simulation when it's gone through all of "cabpath"'s points.
%mouse0=mouse;
%hdg0=hdg;
[rPosX,rPosY,rHDG,drPos]= robotSim(index,X,Y,rPosX0,rPosY0); % simulates cab position.

[bPosX,bPosY,bHDG,dbPos,sa,O2X,O2Y]= bogieSim(rPosX,rPosY,rHDG,drPos,sa,rPosX0,rPosY0,
rHDG0,bPosX0,bPosY0,bHDG0);

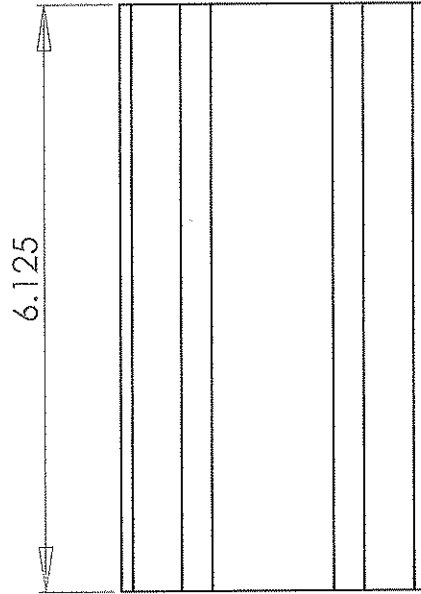
%[mouse,hdg,ra,ba,sa,drPos] = GetSensorVals(mouse0,a); %retrieve sensor values.
%[rPosX,rPosY,rHDG]= robotUpdate(drPos,hdg,rPosX0,rPosY0,rHDG0,ra); %Update robot
position
%[bPosX,bPosY,dbPos,bHDG]= bogieUpdate(rPosX,rPosY,hdg,ba,bPosX0,bPosY0); %Update bogie
position
Matrix(index,:)= [rPosX,rPosY,rHDG,drPos,bPosX,bPosY,bHDG,dbPos,O2X,O2Y]; %Store these in
```

```
Matrix
bIndex=calcIndex(dbPos,Matrix,bPosX,bPosY,index); %calculate index
bVals=steering(Matrix(bIndex,1),Matrix(bIndex,2),Matrix(index,5),Matrix(index,6),bHDG); %
Calculate vector to "aim" point and generate a steering angle.
%setBsteering(bVals,sa); %Set servo position
sa=steeringSim(bVals,sa,tstep); % simulates the steering function.
[Matrix,index]= matSetup(Matrix,index,repeat,drPos); %resets matrix if too long,
advances index if not.
rPosX0=rPosX; %Saves previous values for quick reference
bPosX0=bPosX;
rPosY0=rPosY;
bPosY0=bPosY;
rHDG0=rHDG;
bHDG0=bHDG;
%pause (tstep)%creates a realistic delay.
end
fprintf('done\n')
plot(X,Y,Matrix(:,5),Matrix(:,6),'o'),xlabel('x'),ylabel('y'),axis equal
```

## **D: Detailed Drawings**



2x M8 TAP BOTH ENDS



English (in)

TITLE:

Conor Dodd  
Sr. Project

SIZE DWG. NO.

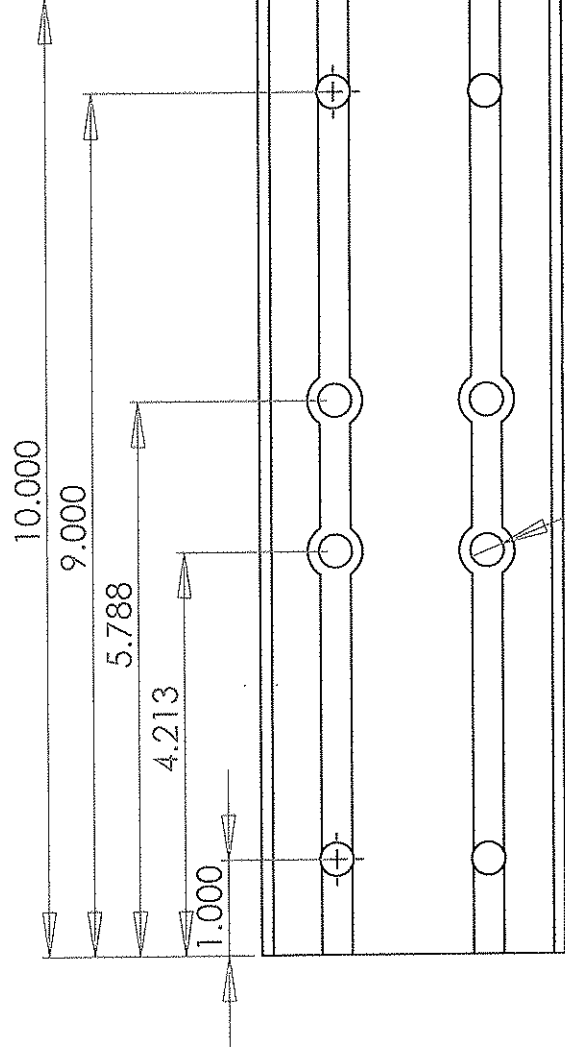
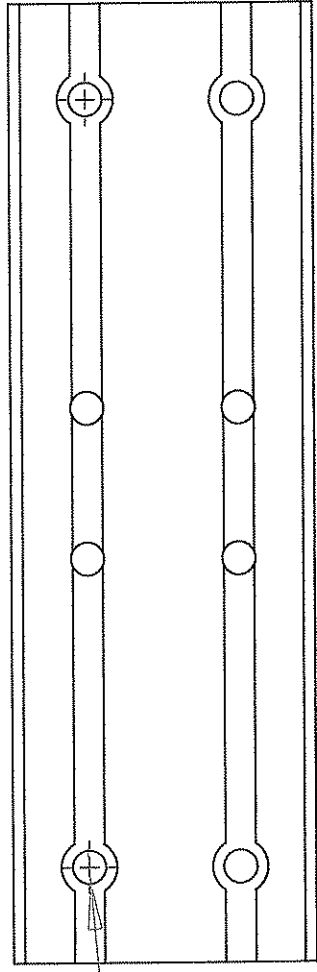
**A** Trailer

REV

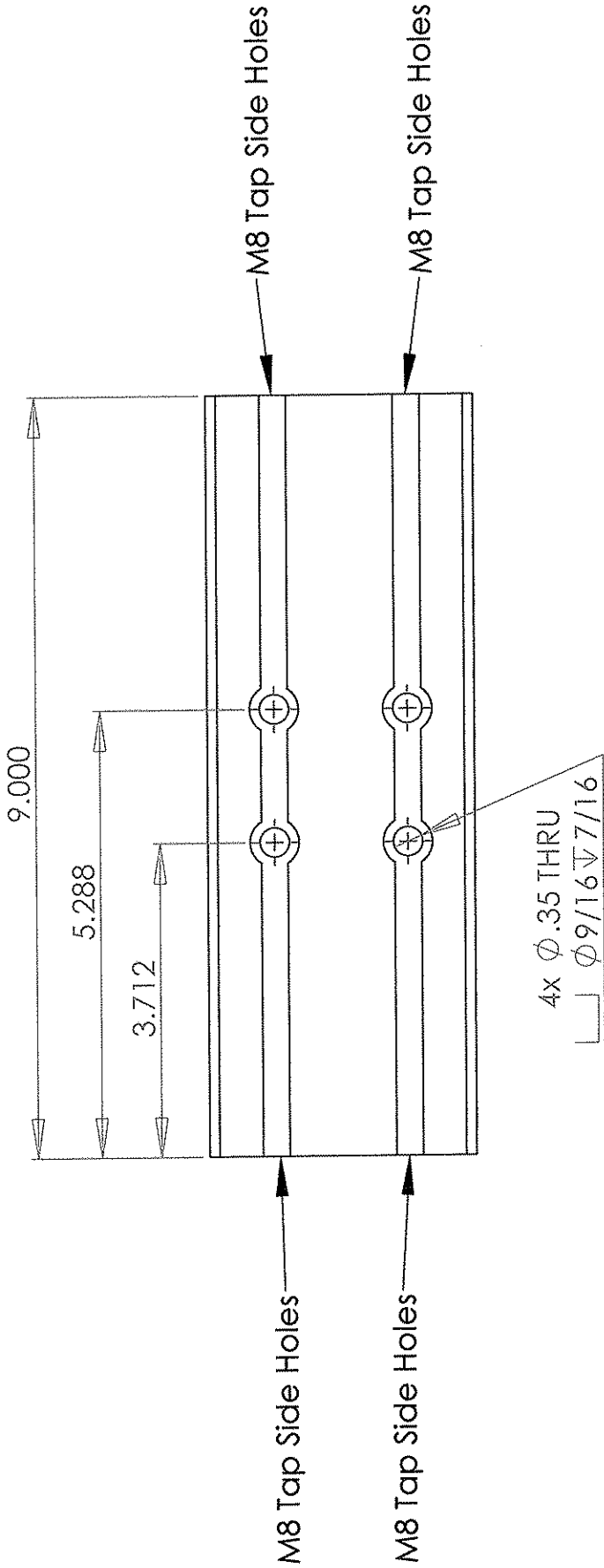
SCALE: 1:2 WEIGHT:

SHEET 1 OF 1

4x  $\phi$ .35 THRU  
L  $\phi$ 9/16  $\nabla$ 7/16



TITLE:	English (in)	REV	
	Conor Dodd		
	Sr. Project		
SIZE	DWG. NO.	REV	
<b>A</b>			
SCALE: 1:5	WEIGHT:		SHEET 1 OF 1



English (in)

TITLE:

Conor Dodd  
Sr. Project

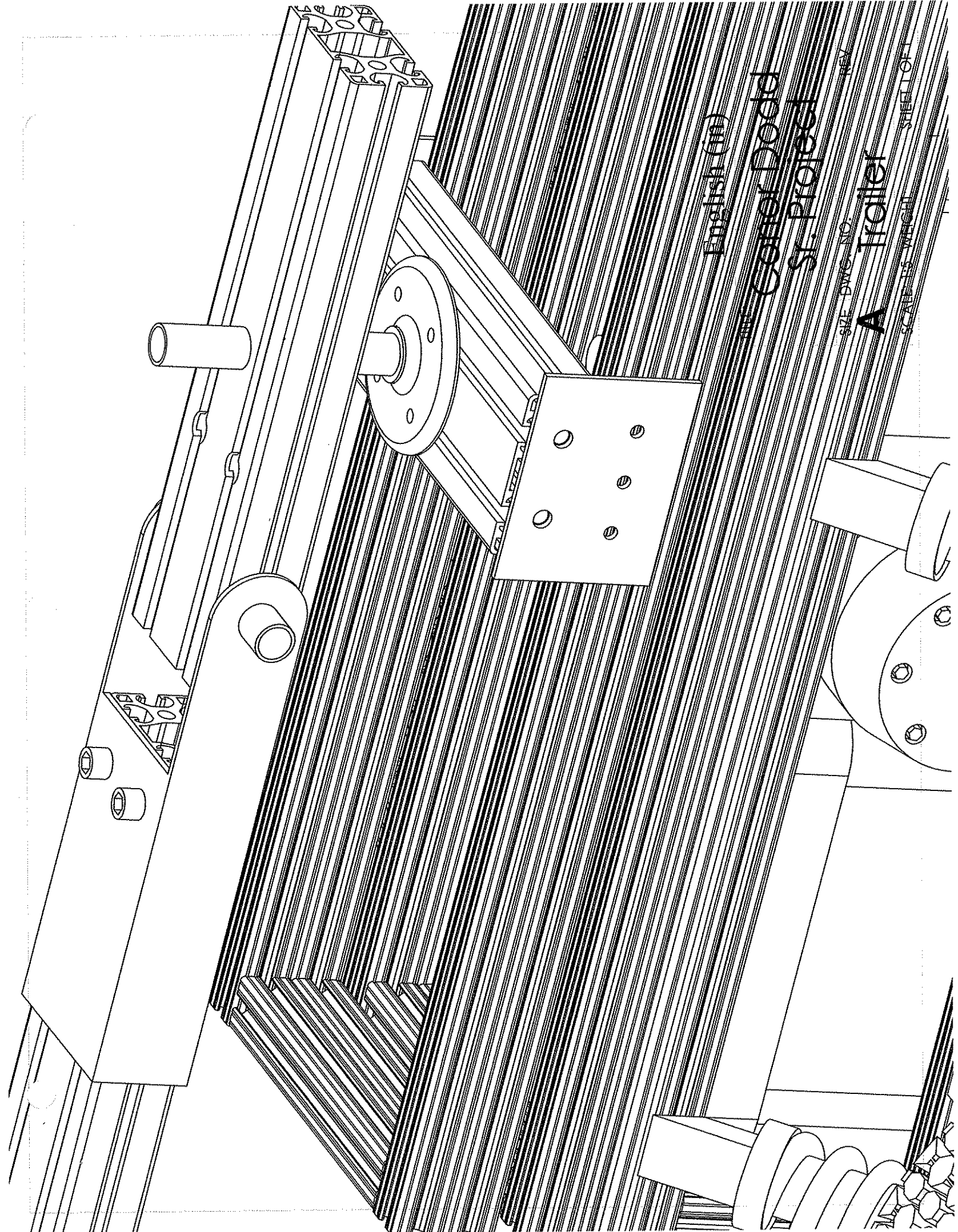
REV

SIZE DWG. NO.

**A** Trailer

SCALE: 1:5 WEIGHT:

SHEET 1 OF 1



English (UK)

Genet Dodd  
St. Project

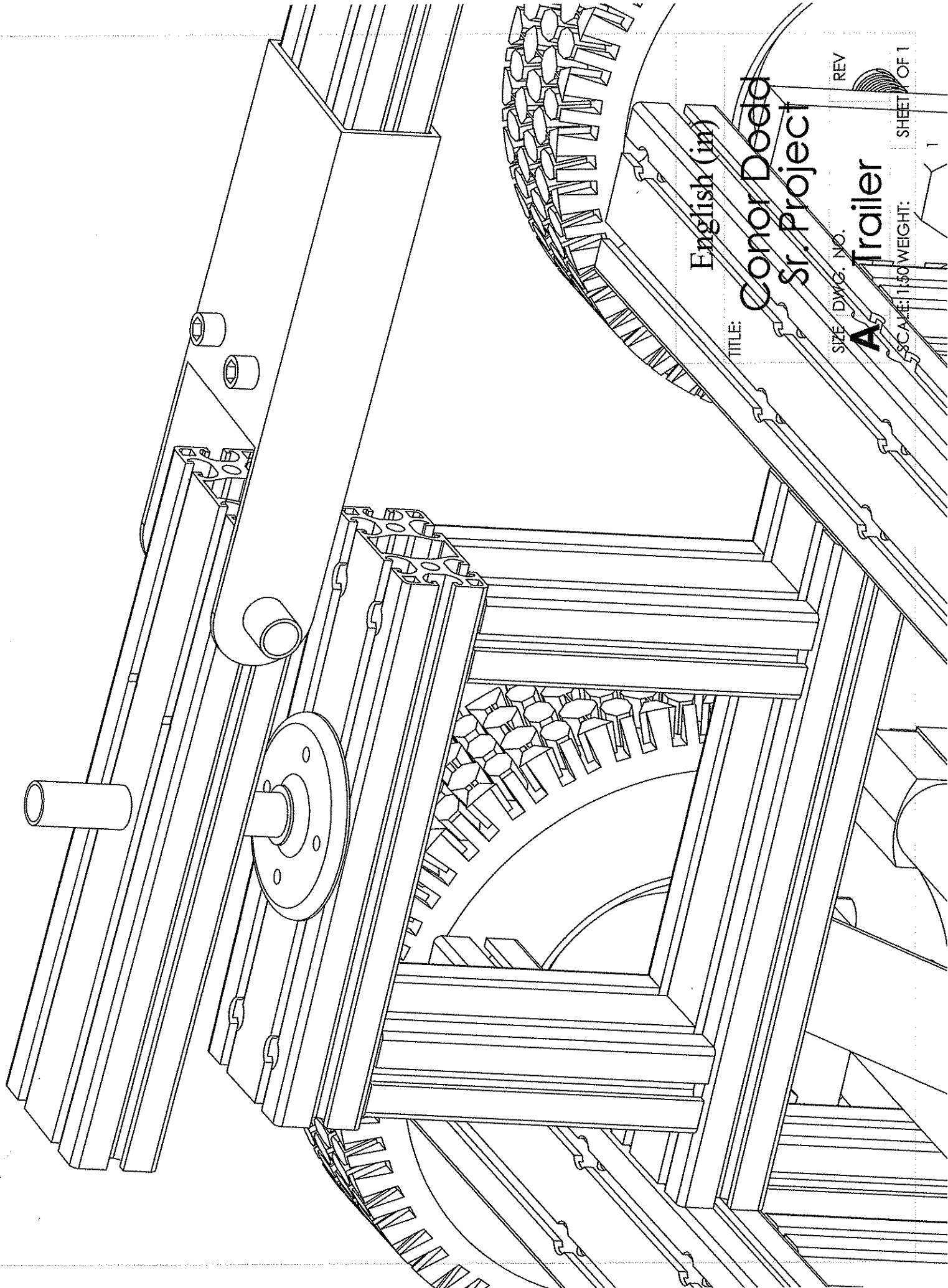
SIZE DWG NO  
REV

**A** Trailer

SCALE 1:5 WEIGHT

SHEET 1 OF 1





English (in)

TITLE:

Conor Doedd  
Sr. Project

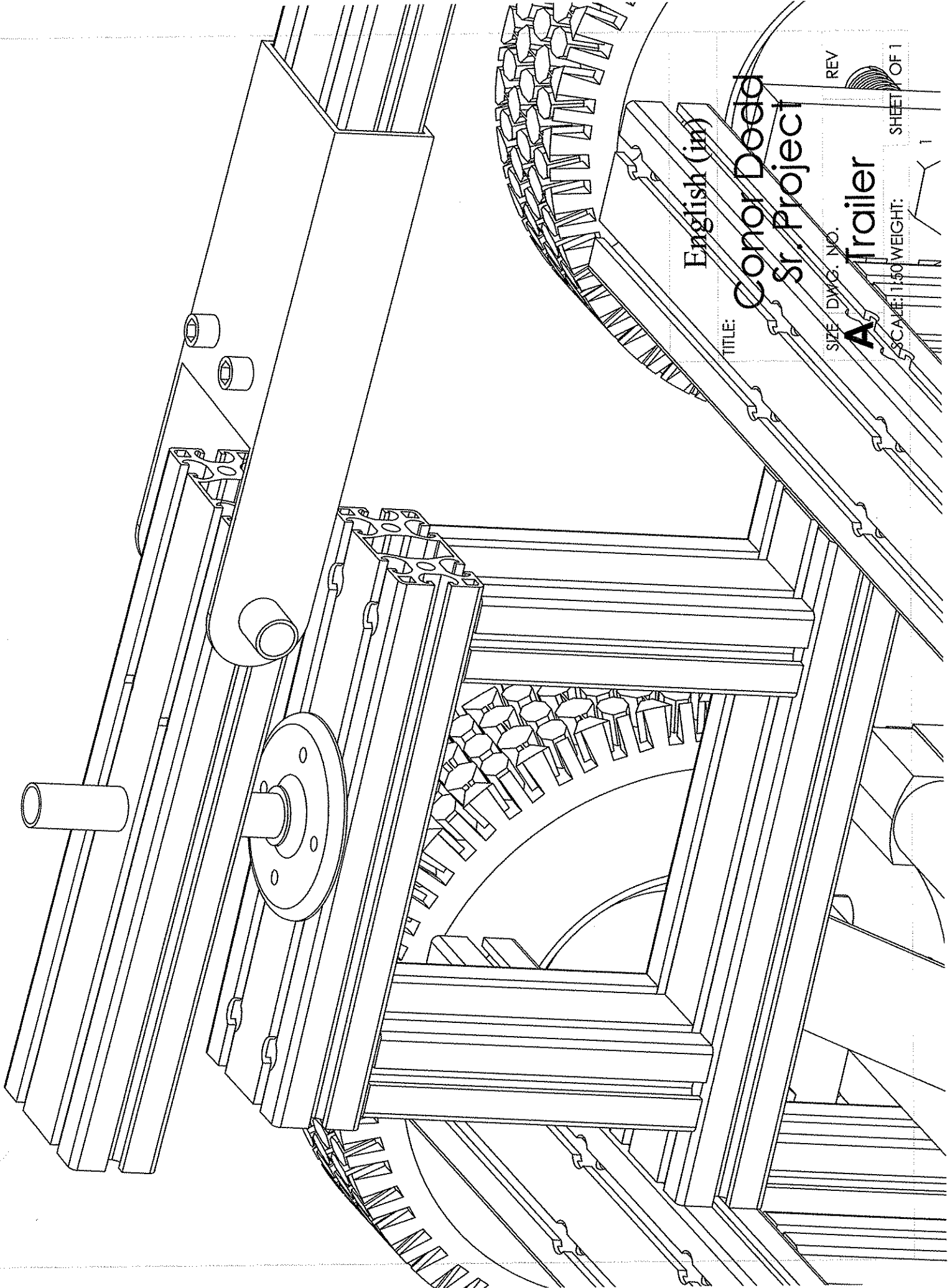
SIZE DWG. NO.

**A** trailer

SCALE: 1/30 WEIGHT:

REV

SHEET OF 1



English (in)

TITLE:

Conor Doedd  
Sr. Project

SIZE: DWG. NO.

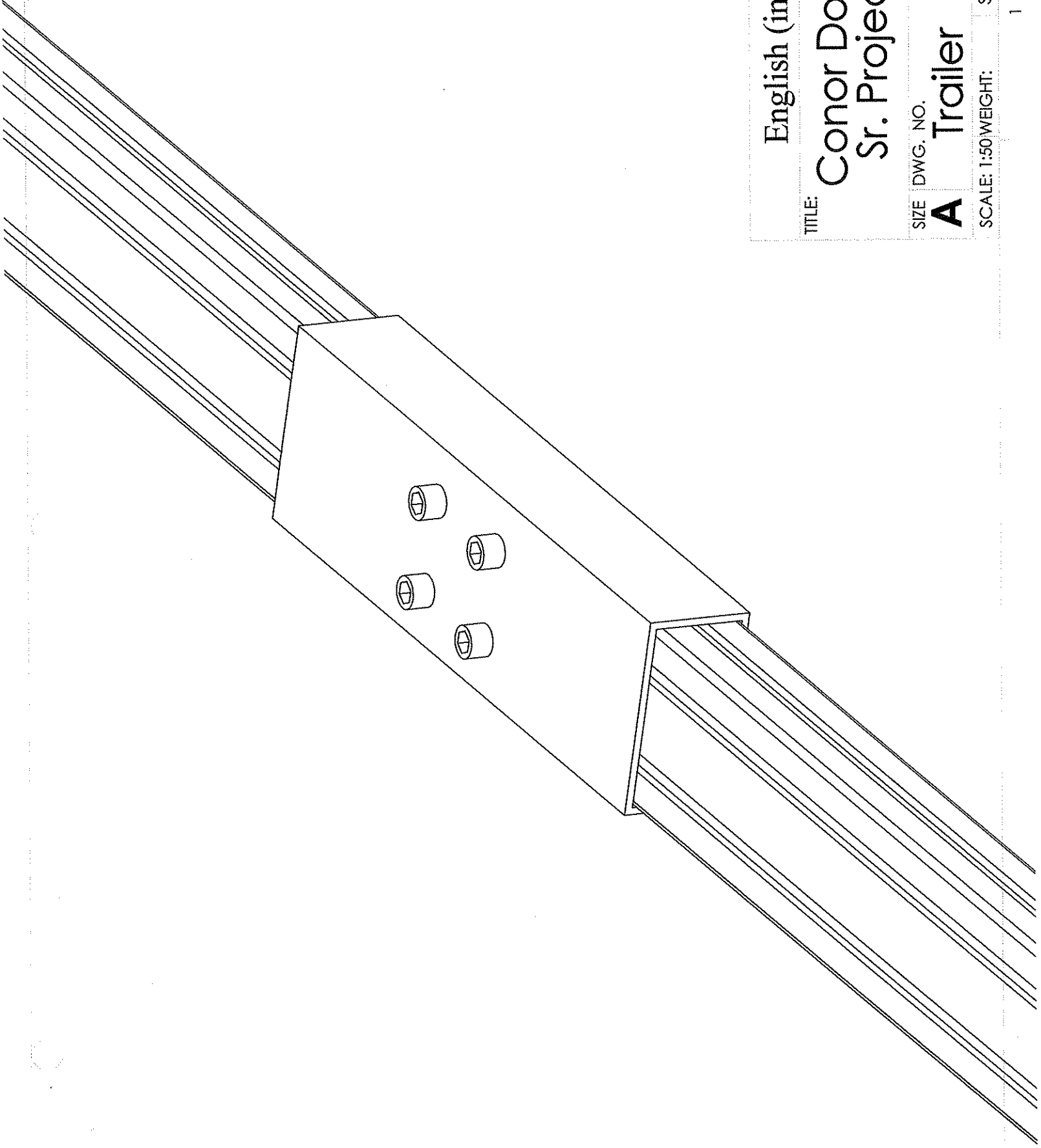
**A** Trailer

SCALE: 1/150 WEIGHT:

REV

SHEET OF 1

1



English (in)

TITLE:

Conor Dodd  
Sr. Project

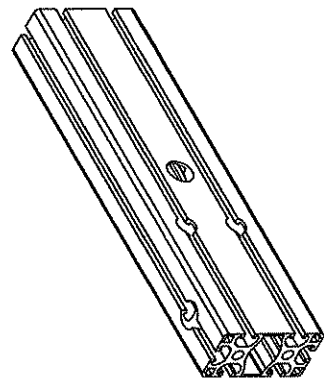
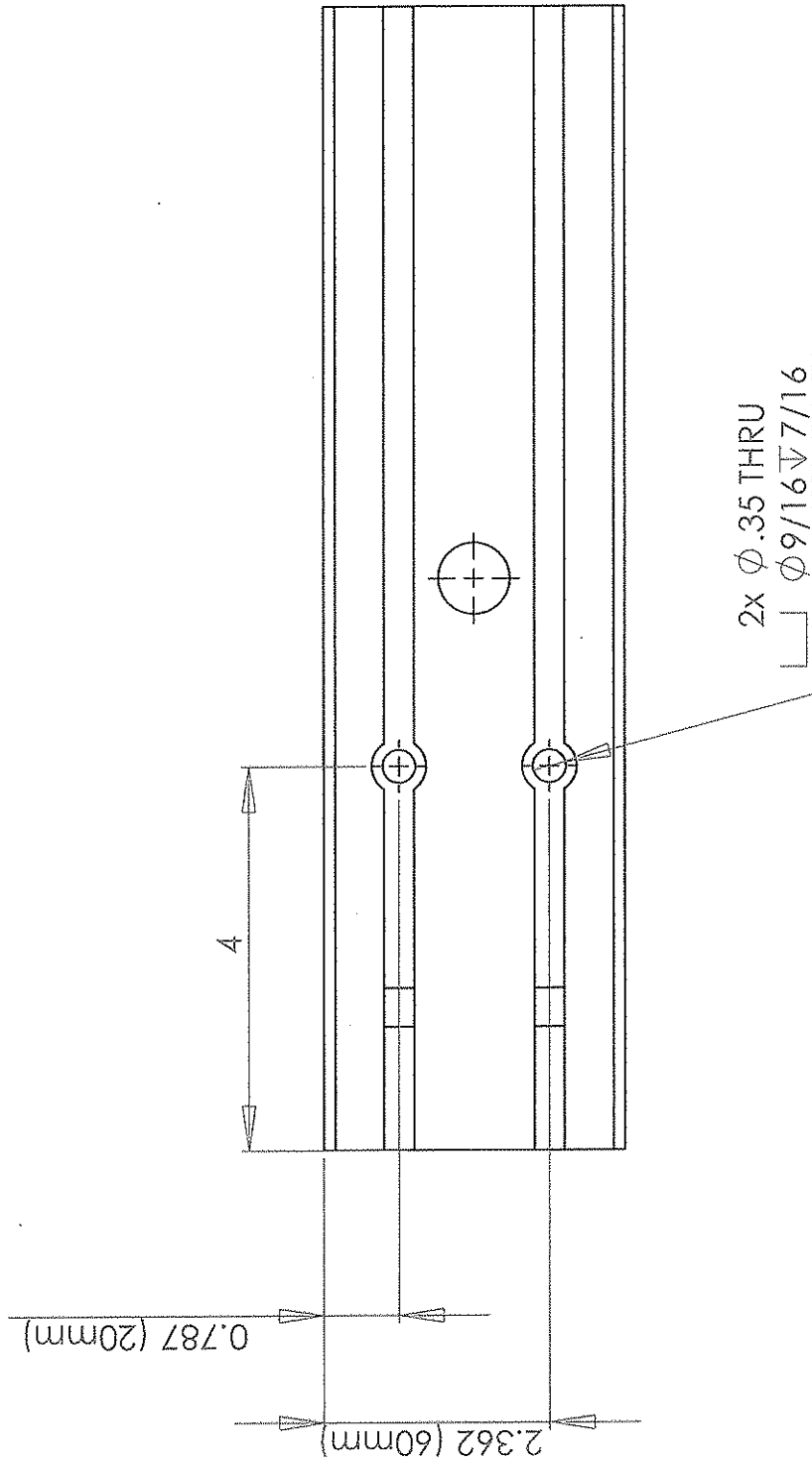
SIZE DWG. NO.

**A** Trailer

REV

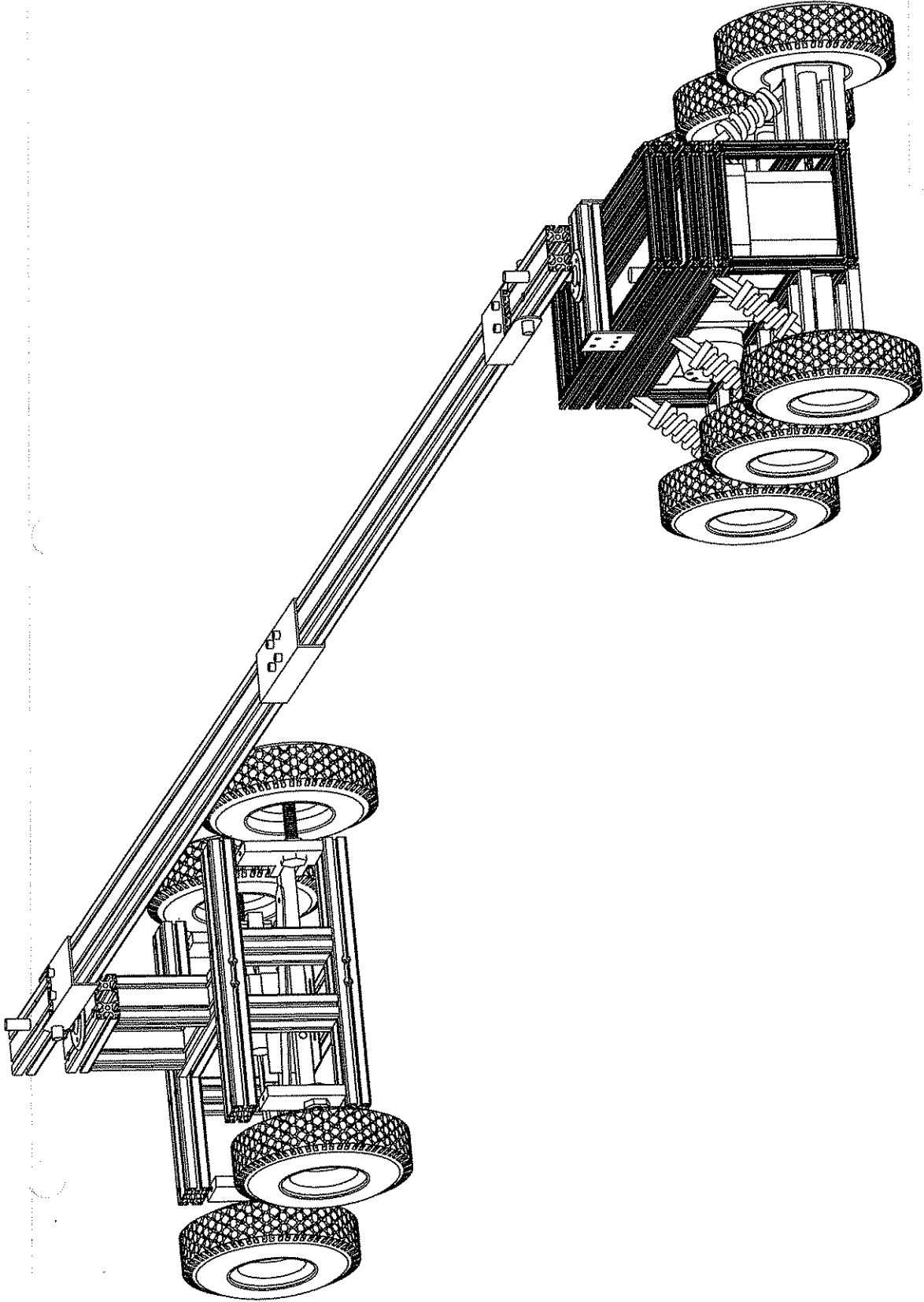
SCALE: 1:50 WEIGHT:

SHEET 1 OF 1



X2

English (in)	REV
TITLE: Connor Dodd Sr. Project	
SIZE DWG. NO. A	
Trailer	
SCALE: 1:5	WEIGHT: SHEET 1 OF 1



English (in)

TITLE:

Conor Dodd  
Sr. Project

SIZE DWG. NO.

REV

**A** Trailer

SCALE: 1:50 WEIGHT:

SHEET 1 OF 1

3.150

1.575

Dimensions to INSIDE edges

4x  $\varnothing$  0.350 THRU

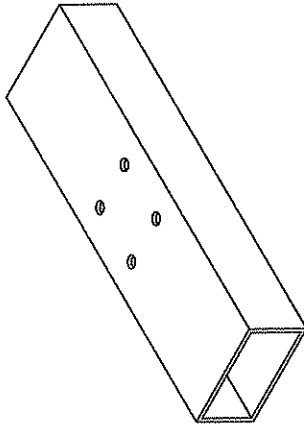
1.575

(.787 from inside edge)

5.000

7.000

12.000



English (in)

TITLE:

Conor Dodd  
Sr. Project

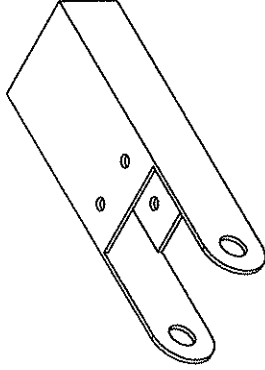
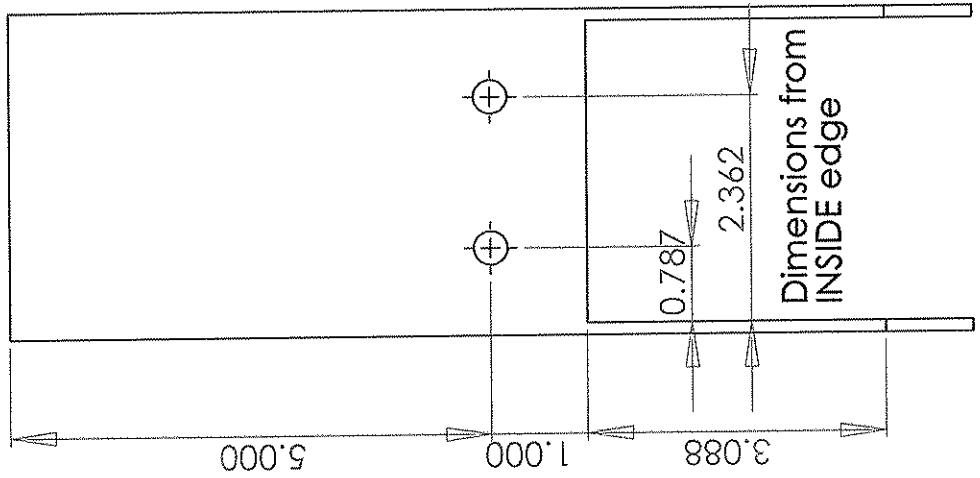
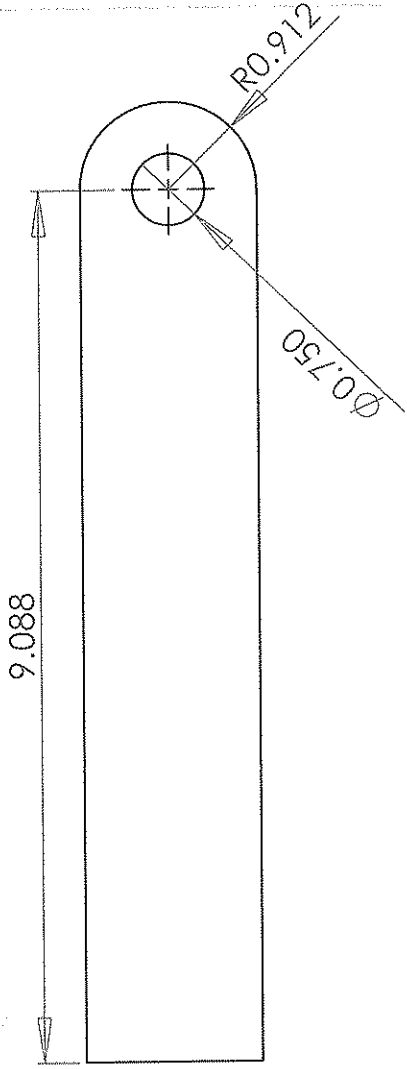
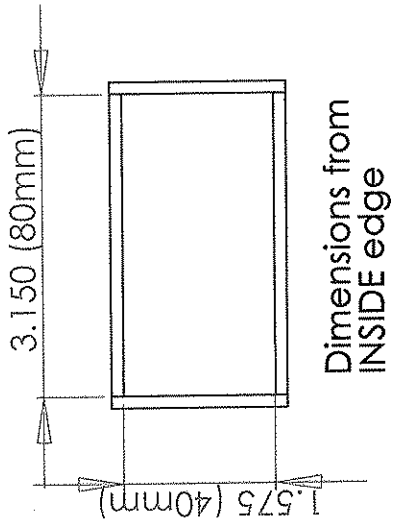
REV

SIZE DWG. NO.

**A** Trailer

SCALE: 1:5 WEIGHT:

SHEET 1 OF 1



English (in)

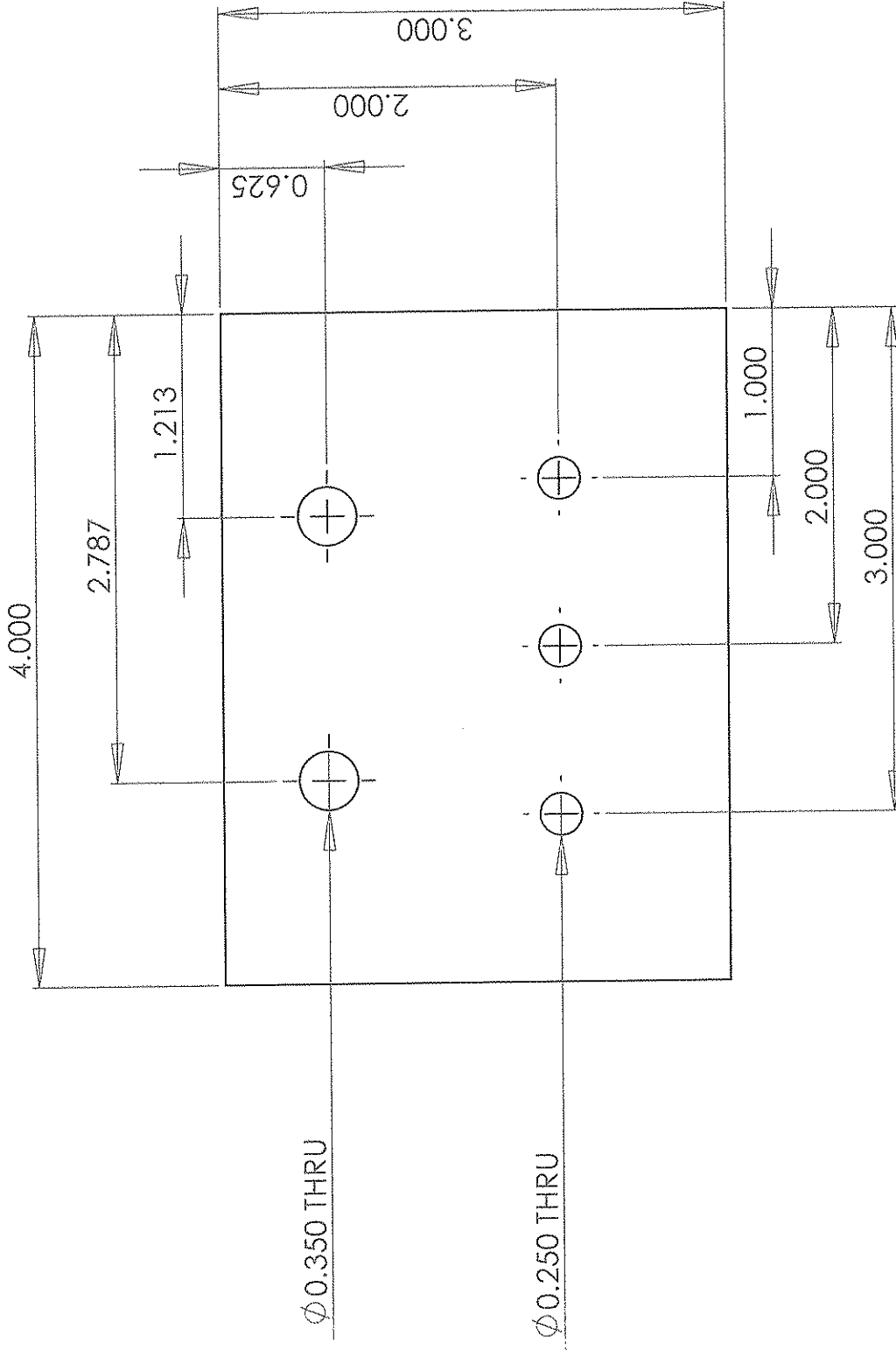
TITLE:

Conor Dodd  
Sr. Project

SIZE DWG. NO. REV

**A** Trailer

SCALE: 1:2 WEIGHT: SHEET 1 OF 1



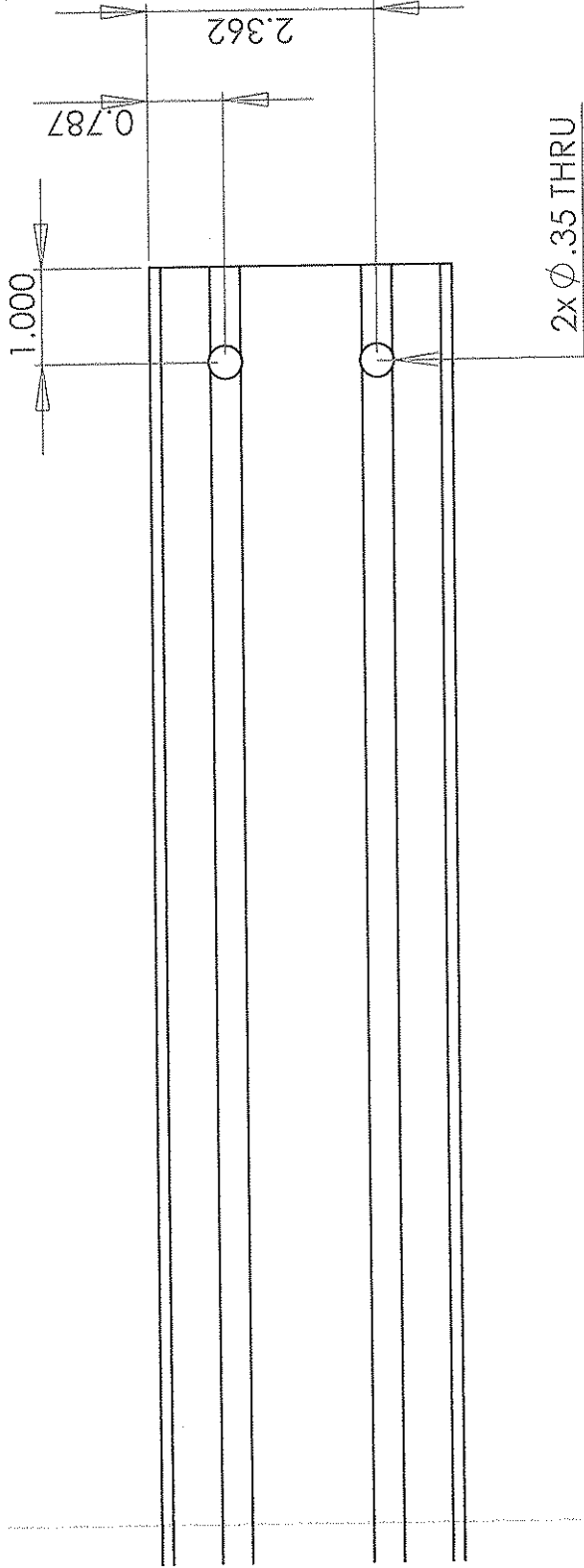
**Assumed 1/8 Steel, not Critical**

English (in)

TITLE: **Conor Dodd  
Sr. Project**

SIZE	DWG. NO.	REV
<b>A</b>	<b>Trailer</b>	
SCALE: 1:1	WEIGHT:	SHEET 1 OF 1





**Don't cut peice to any length, the current lengths are fine.**

**Cut these holes at both ends.**

English (in)

TITLE:

Conor Dodd  
Sr. Project

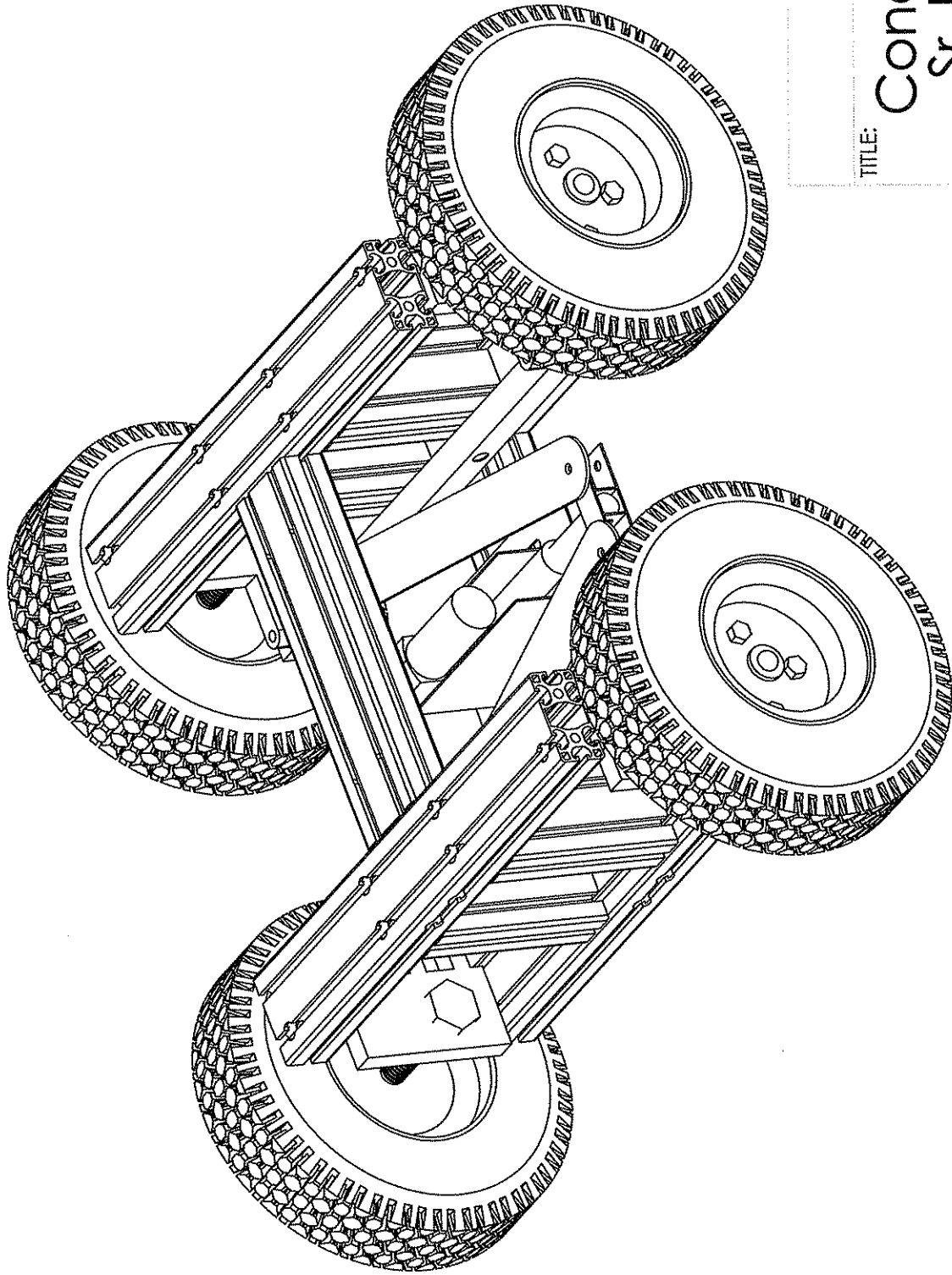
SIZE DWG. NO.

**A** Trailer

REV

SCALE: 1:20 WEIGHT:

SHEET 1 OF 1



TITLE:

Conor Dodd  
Sr. Project

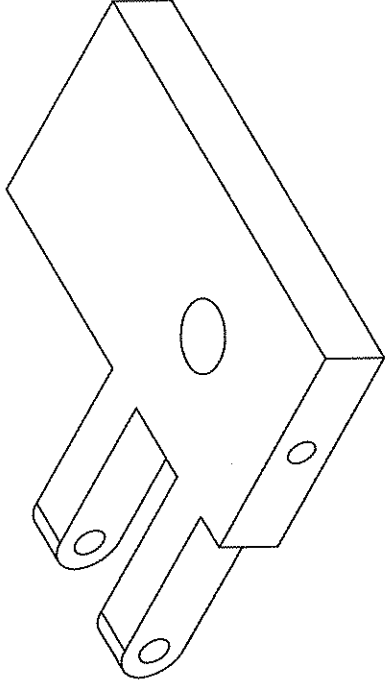
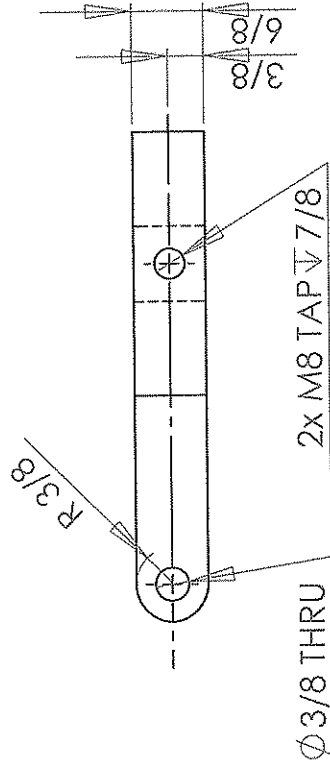
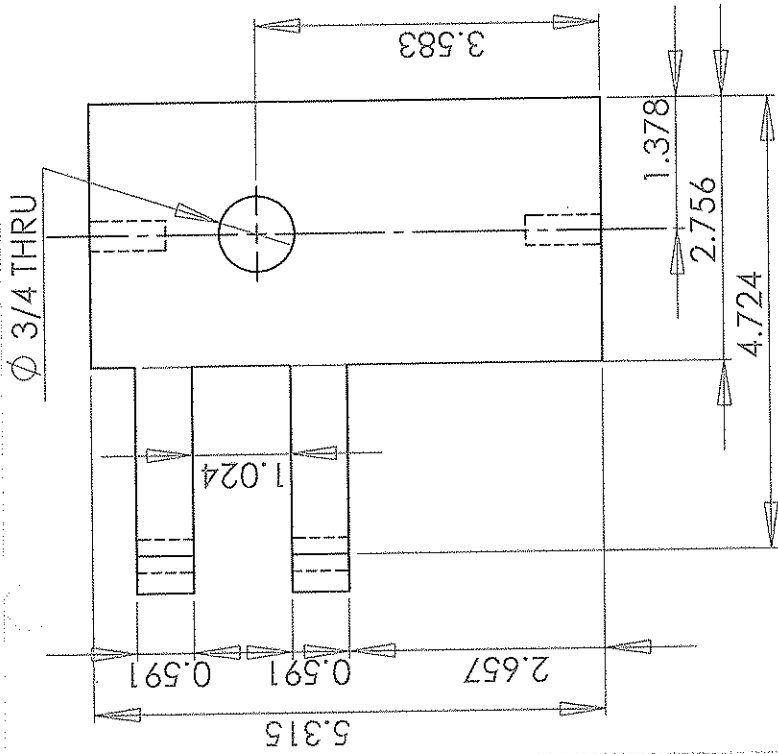
SIZE DWG. NO.

REV

**A** Frame

SCALE: 1:10 WEIGHT:

SHEET 1 OF 1



ENGLISH (in)

TITLE:

Conor Dodd  
Sr. Project

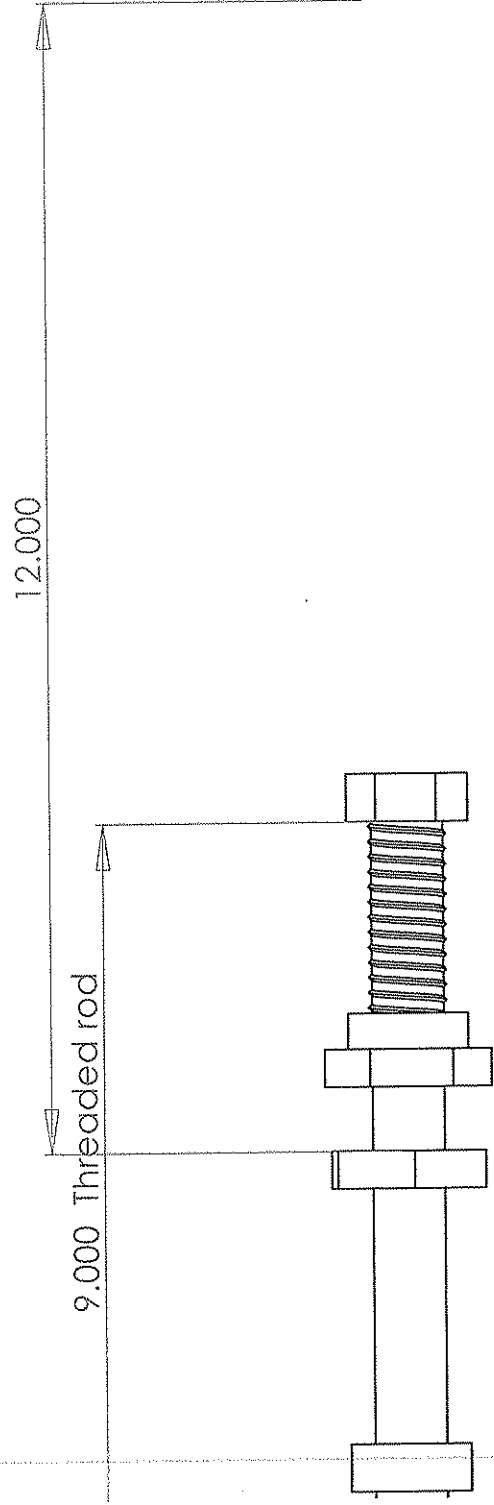
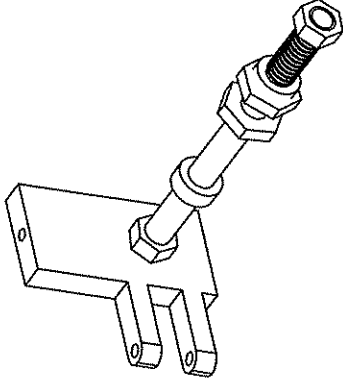
SIZE DWG. NO.

REV

**A** Knuckle

SCALE: 1:2 WEIGHT:

SHEET 1 OF 1



ENGLISH (in)

TITLE:

Conor Dodd  
Sr. Project

SIZE DWG. NO.

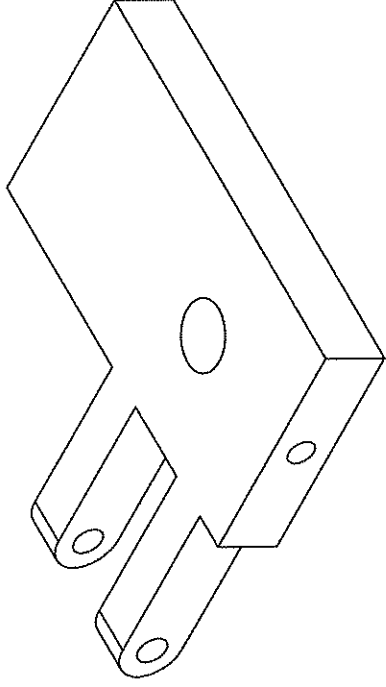
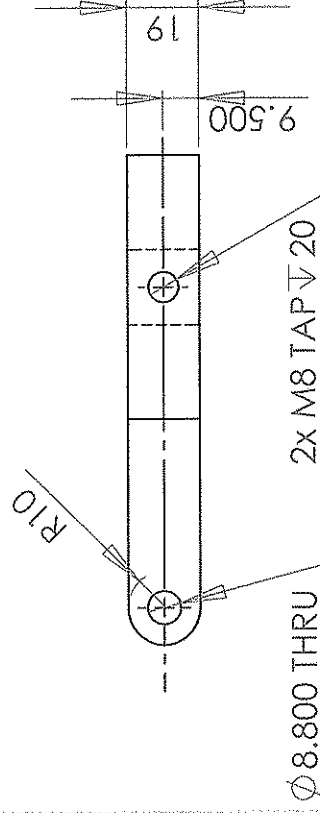
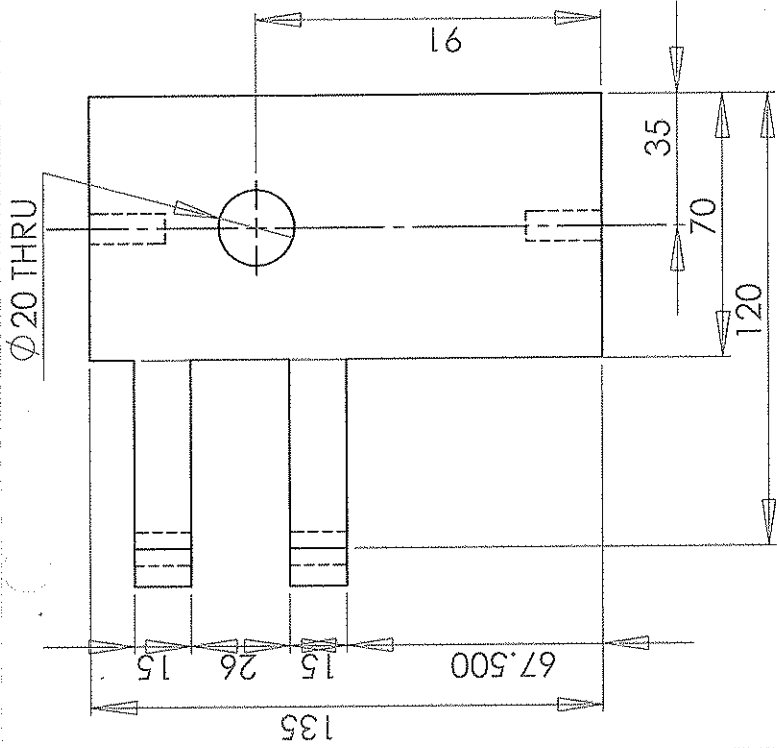
**A** Knuckle

REV

SCALE: 1:5 WEIGHT:

SHEET 1 OF 1

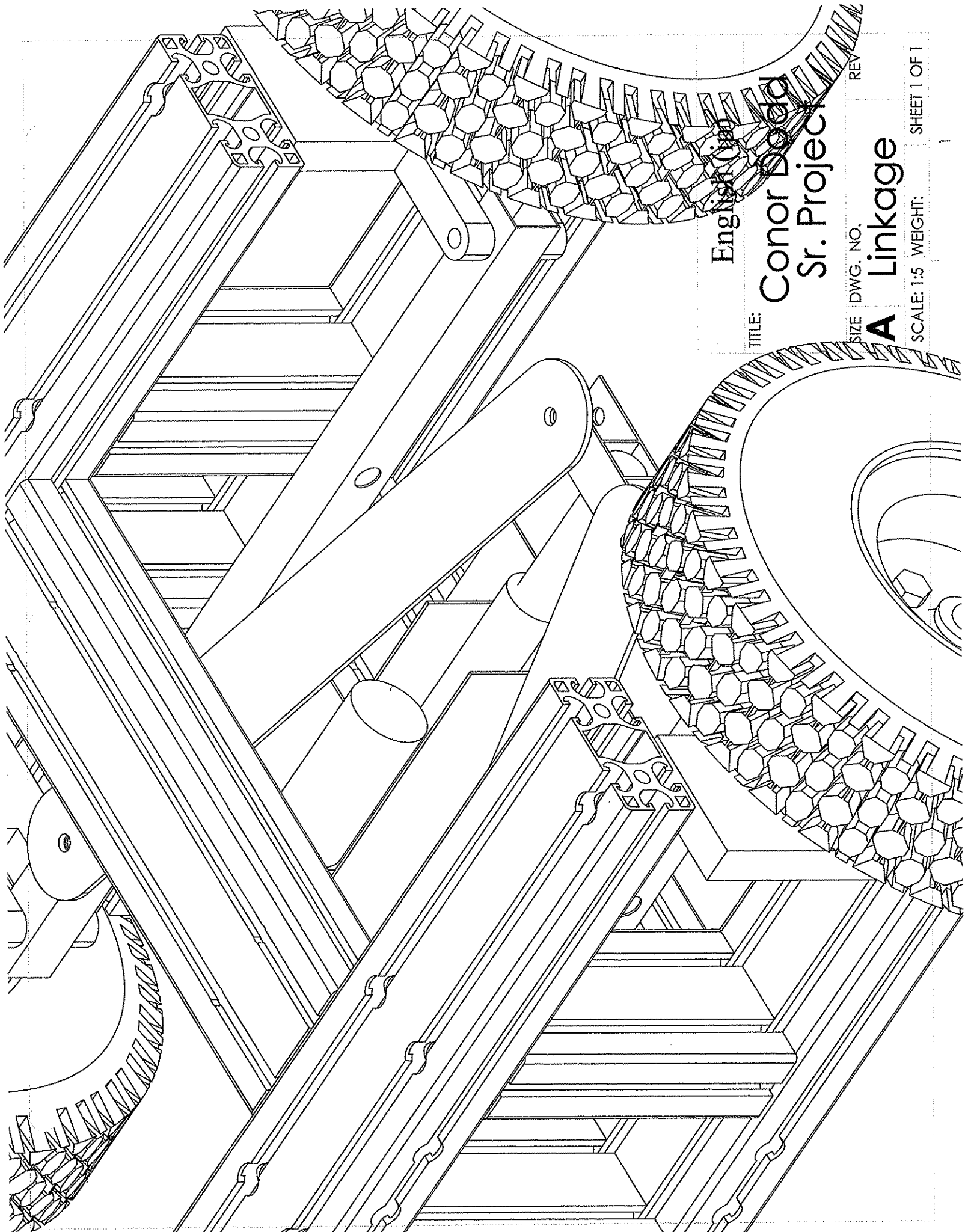
**SolidWorks Educational License  
Instructional Use Only**



TITLE: **Conor Dodd  
Sr. Project**

SIZE DWG. NO. **A** Knuckle

SCALE: 1:2 WEIGHT: SHEET 1 OF 1



English (in)

TITLE:

Conor Dodd  
Sr. Project

SIZE DWG. NO.

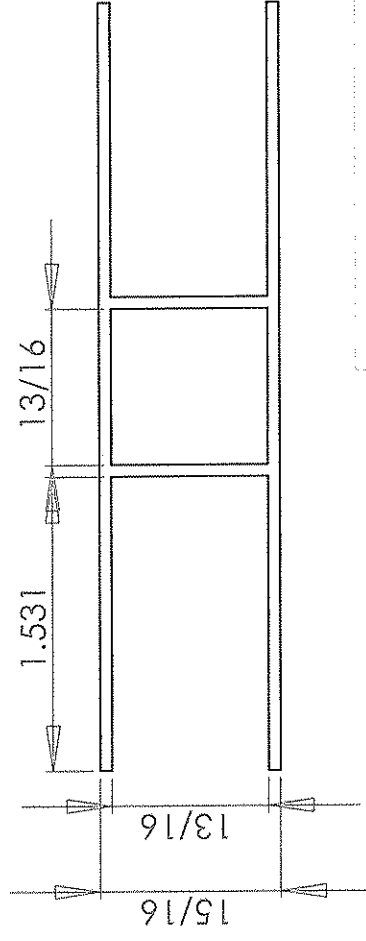
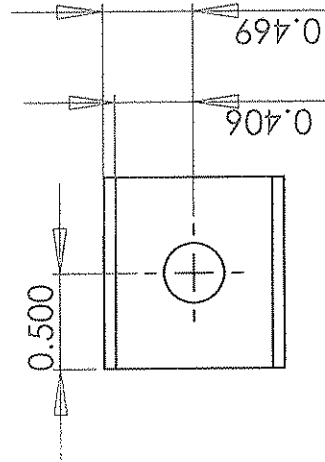
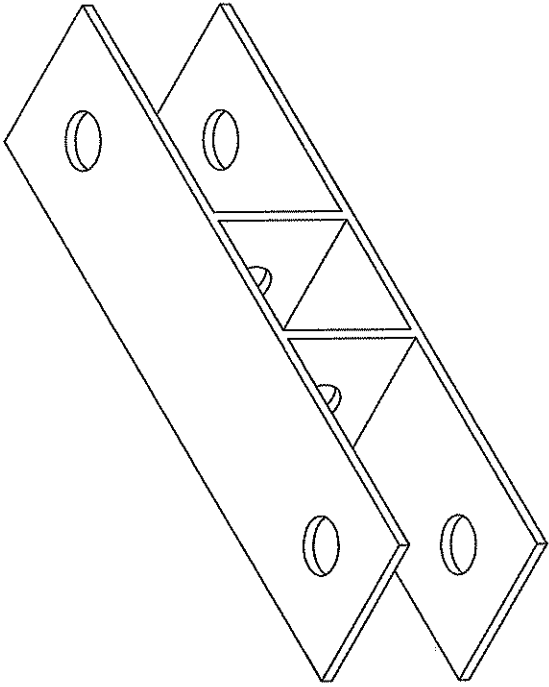
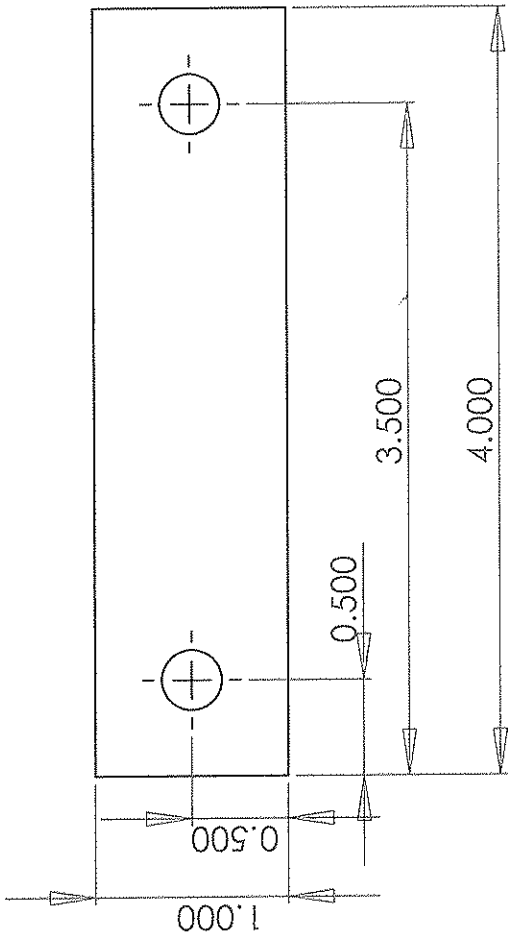
**A** Linkage

REV

SCALE: 1:5 WEIGHT:

SHEET 1 OF 1

1



English (in)

TITLE:

Conor Dodd  
Sr. Project

SIZE DWG. NO.

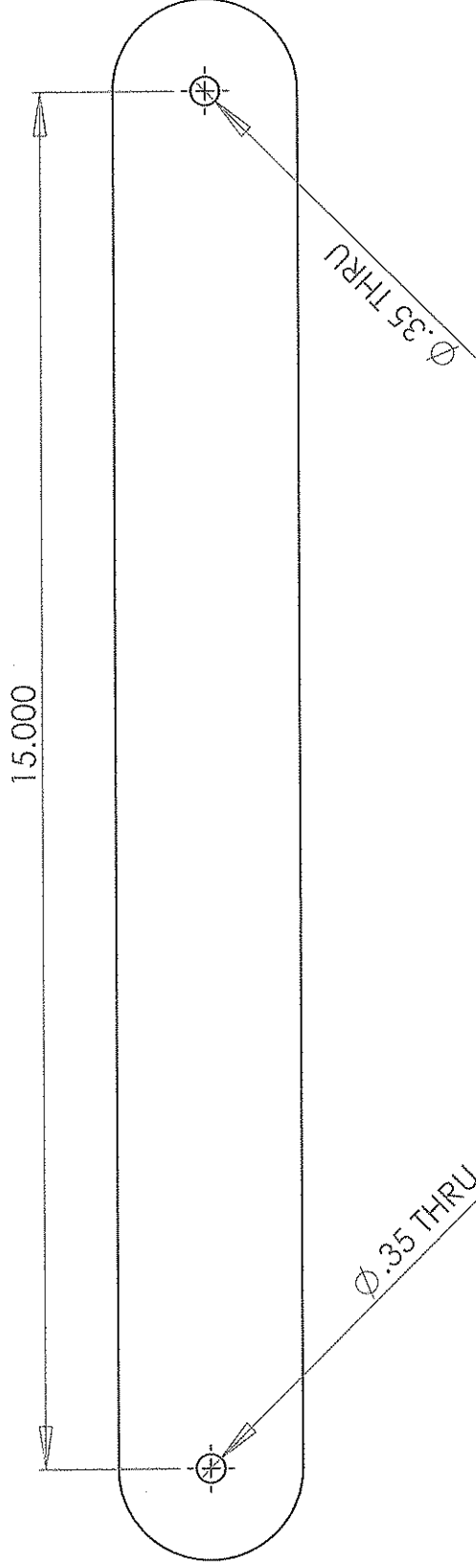
**A** Linkage

REV

SCALE: 1:5 WEIGHT:

SHEET 1 OF 1

NOTE: All steel plates 1/16in thick.



English (in)

TITLE: **Conor Dodd**  
**Sr. Project**

REV

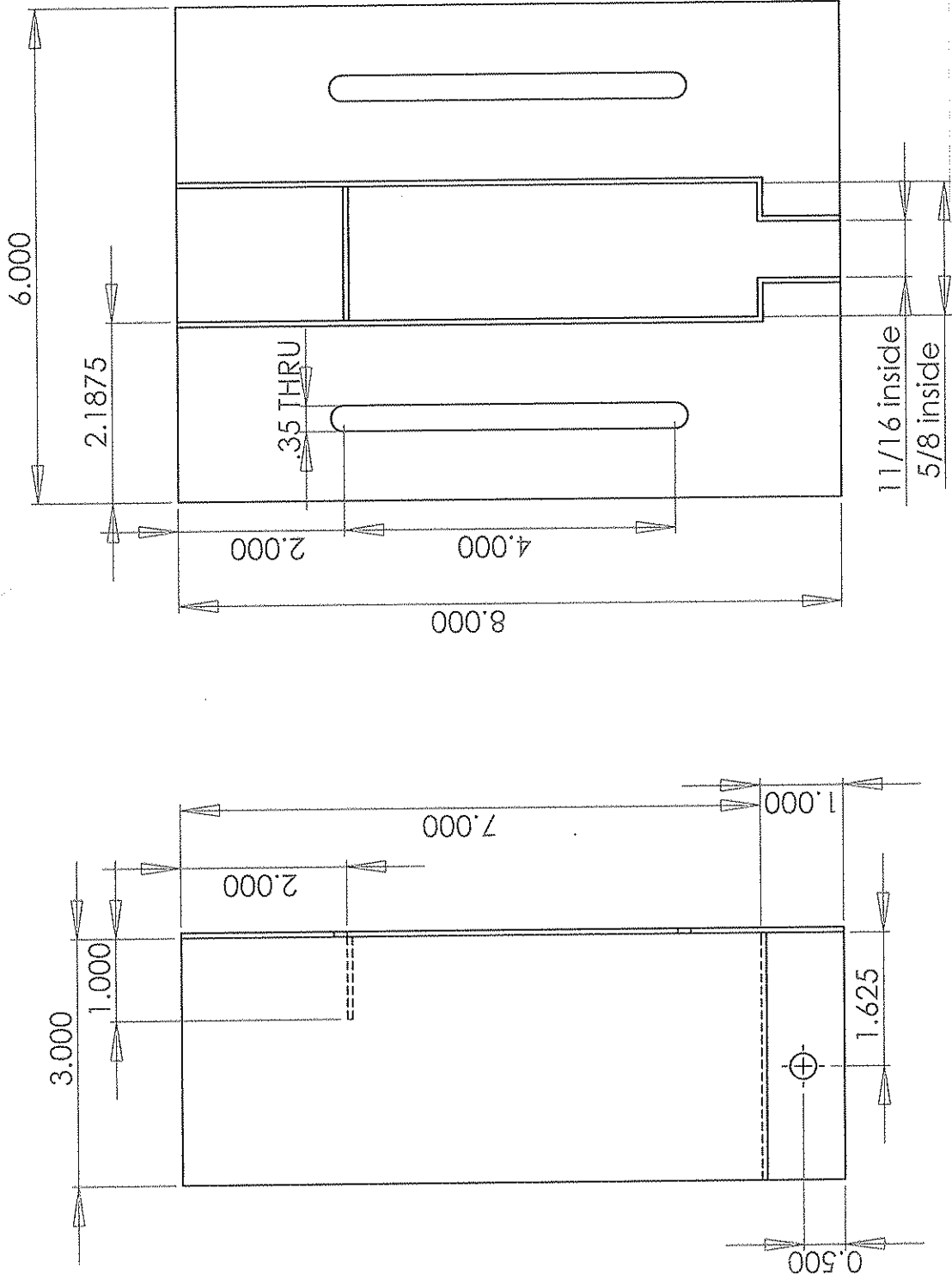
SIZE DWG. NO.

**A** Linkage

SHEET 1 OF 1

SCALE: 1:5 WEIGHT:





English (in)

TITLE:

Conor Dodd  
Sr. Project

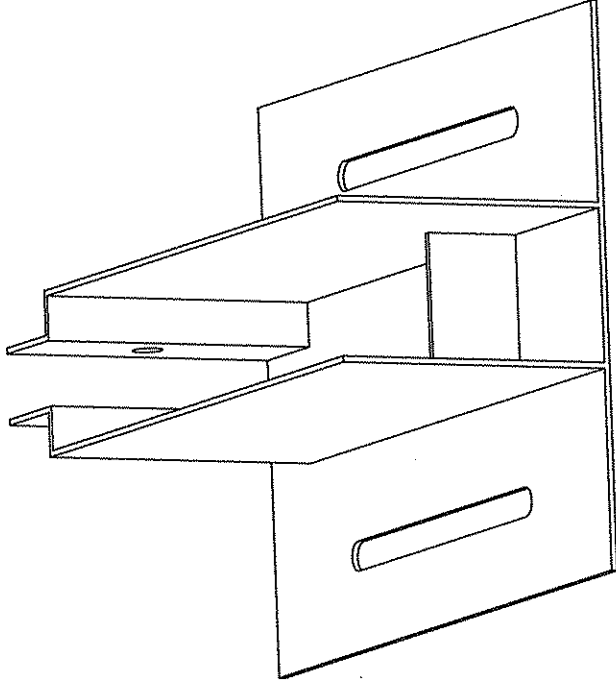
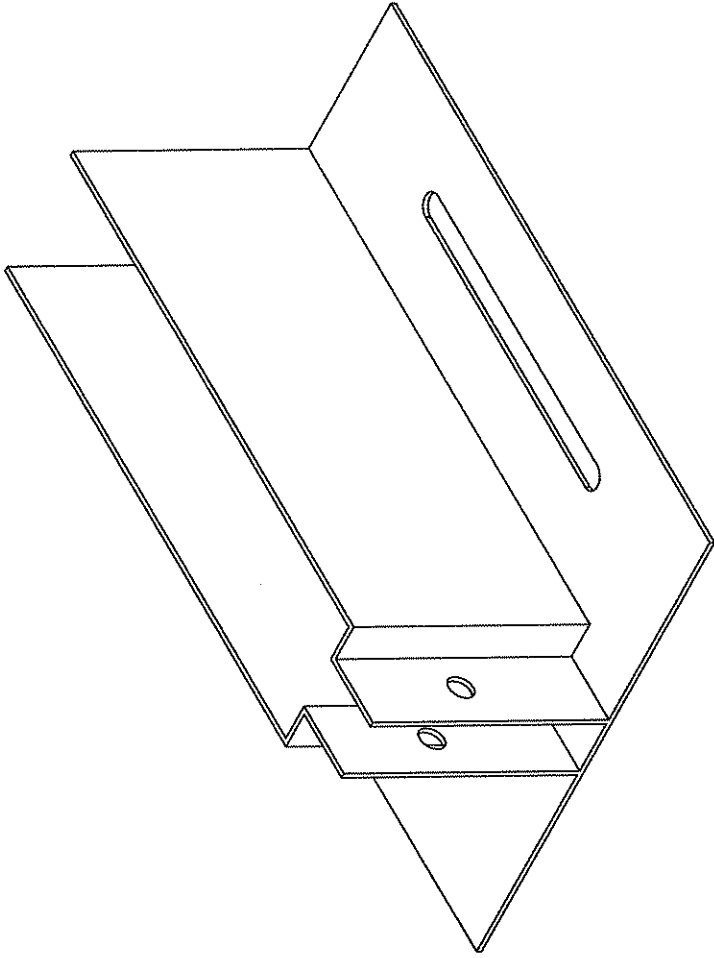
SIZE DWG. NO. REV

**A** Linkage

SCALE: 1:5 WEIGHT:

SHEET 1 OF 1

NOTE: All steel plates 1/16in thick.



English (in)

TITLE:

Conor Dodd  
Sr. Project

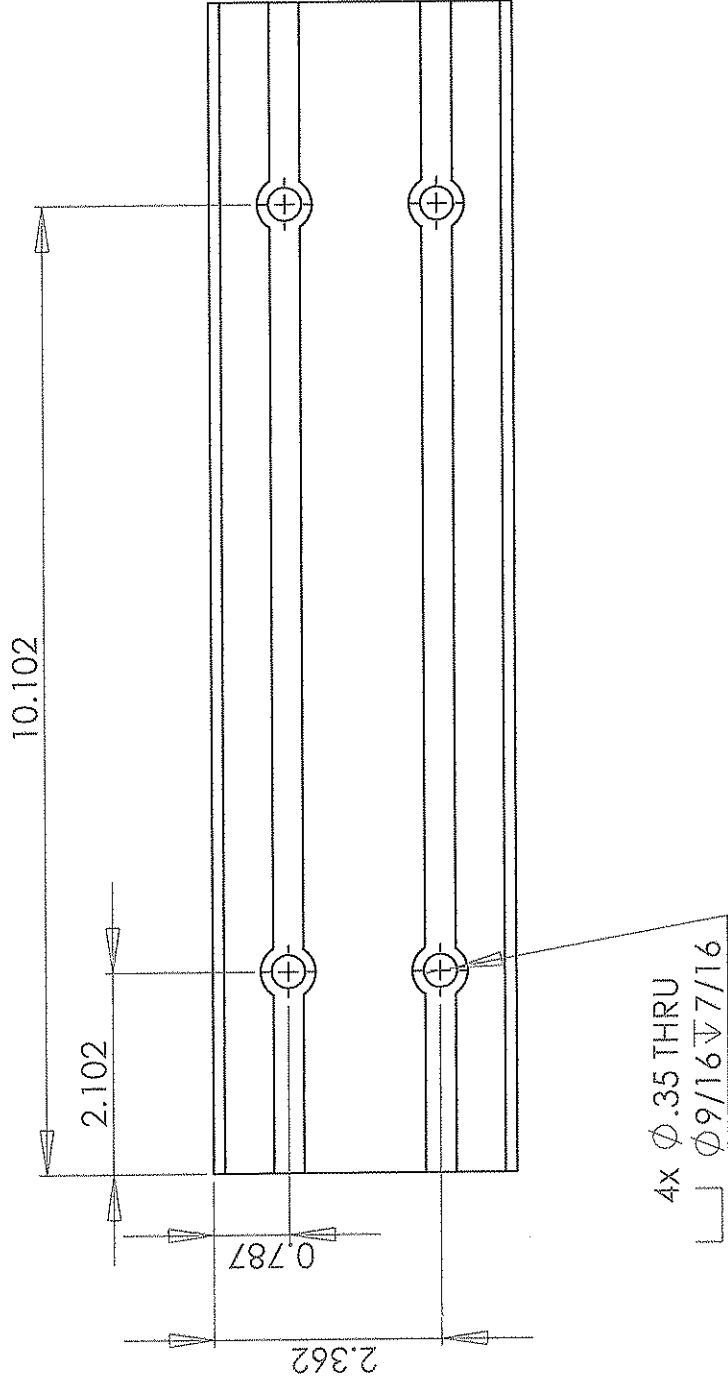
SIZE DWG. NO.

REV

**A** Linkage

SCALE: 1:5 WEIGHT:

SHEET 1 OF 1



English (in)

TITLE:

Conor Dodd  
 Sr. Project

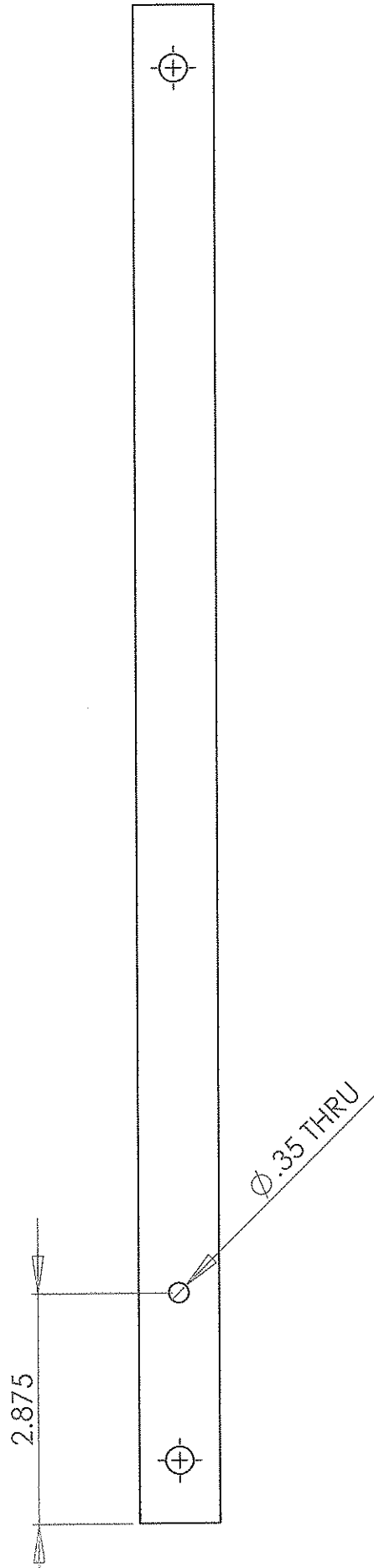
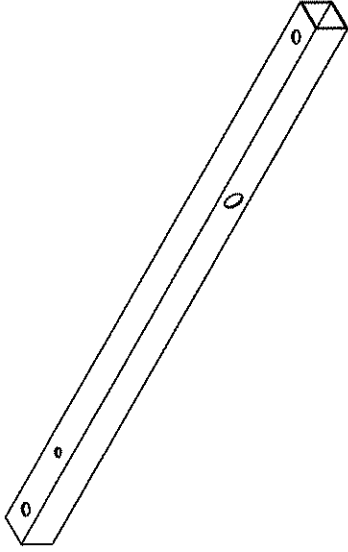
SIZE DWG. NO.

REV

**A** Linkage

SCALE: 1:5 WEIGHT:

SHEET 1 OF 1



English (in)

TITLE:

Conor Dodd  
Sr. Project

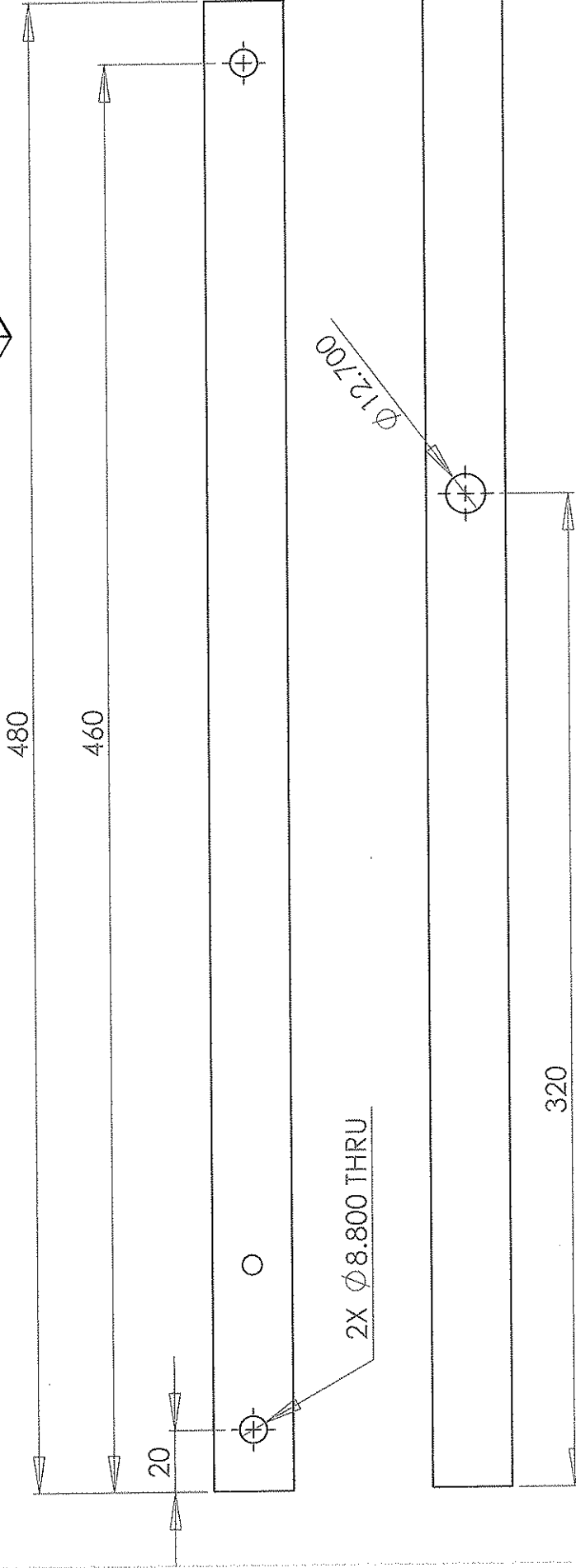
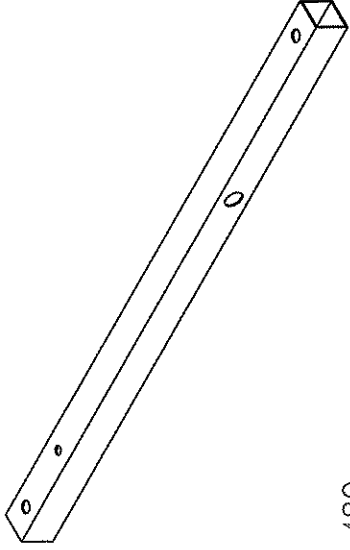
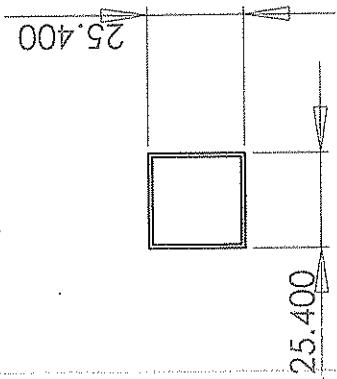
SIZE DWG. NO.

REV

**A** Linkage

SCALE: 1:5 WEIGHT:

SHEET 1 OF 1

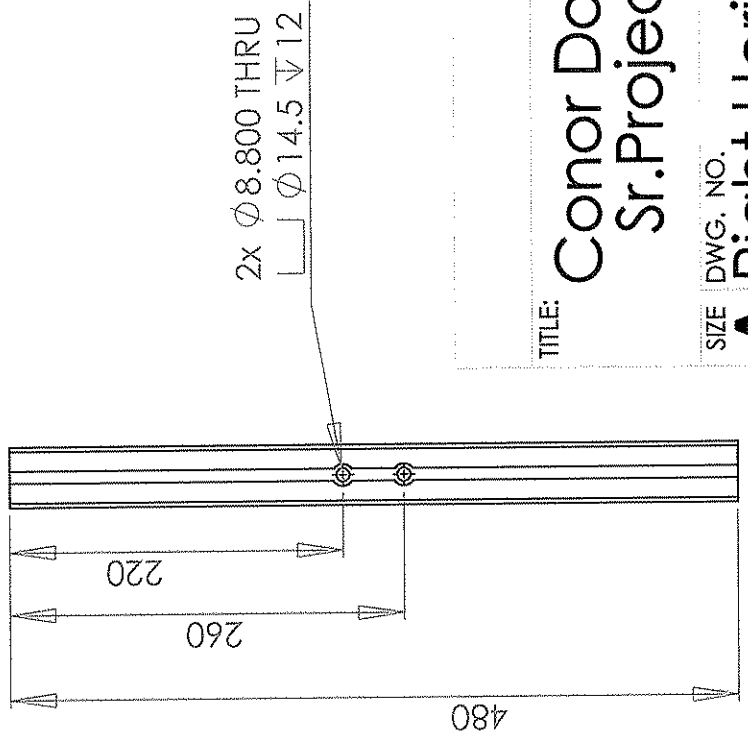
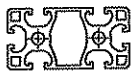
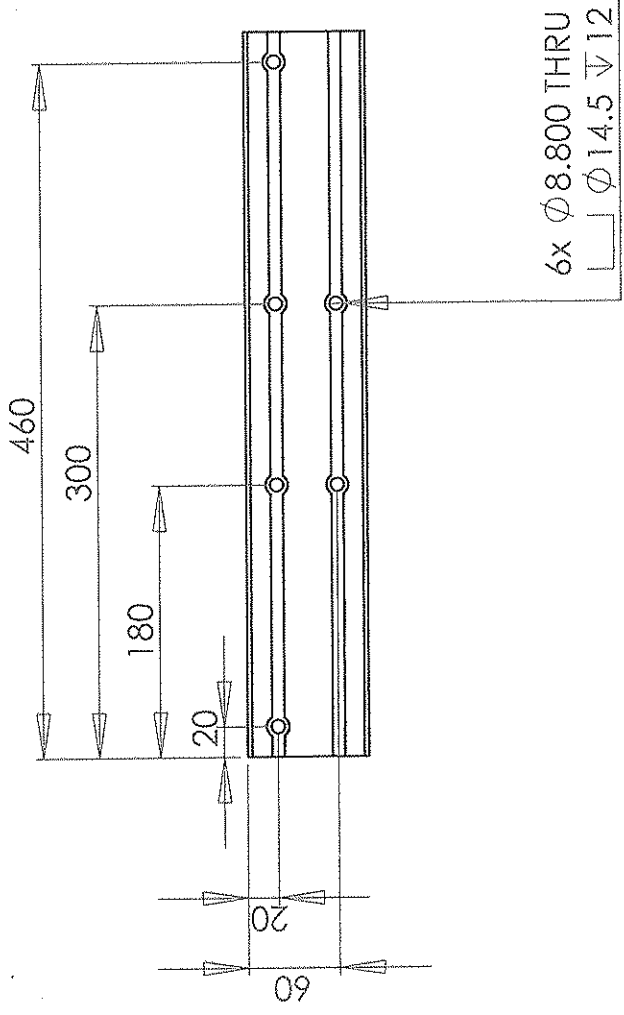


METRIC (mm)

TITLE: **Conor Dodd**  
**Sr. Project**

SIZE DWG. NO. **A** Linkage REV

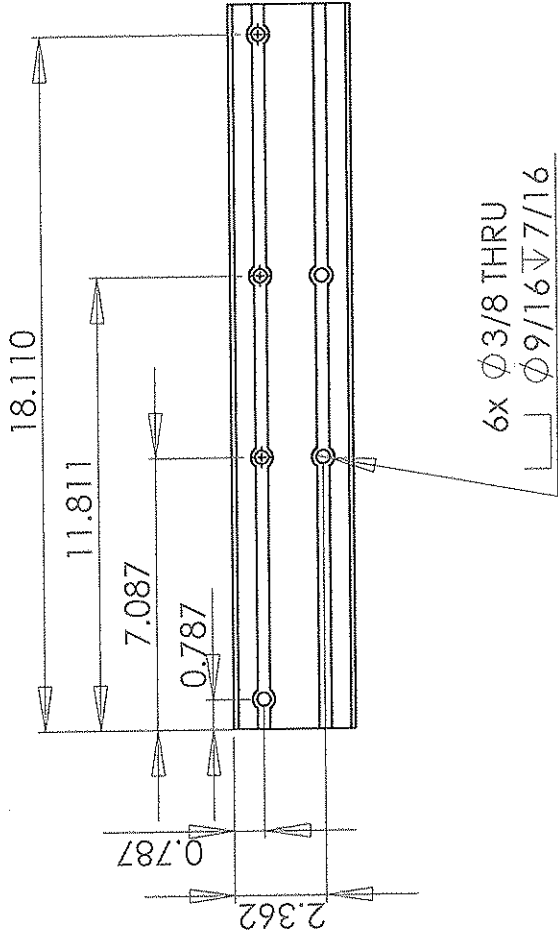
SCALE: 1:5 WEIGHT: SHEET 1 OF 1



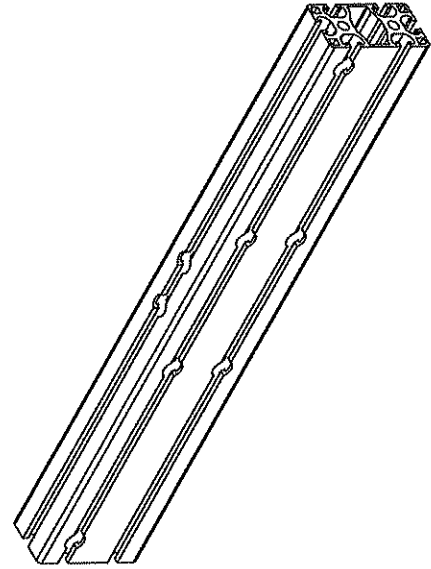
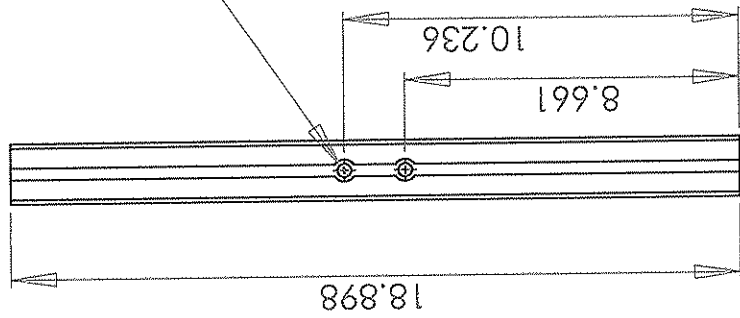
TITLE: **Conor Dodd**  
**Sr.Project**

SIZE DWG. NO. **A** REV  
**Right Horiz.**

SCALE: 1:5 WEIGHT: SHEET 1 OF 1



2x  $\Phi 3/8$  THRU  
 $\Phi 9/16 \nabla 7/16$



ENGLISH (in)

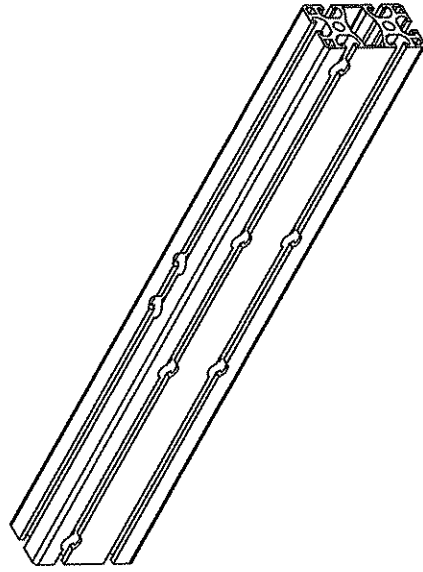
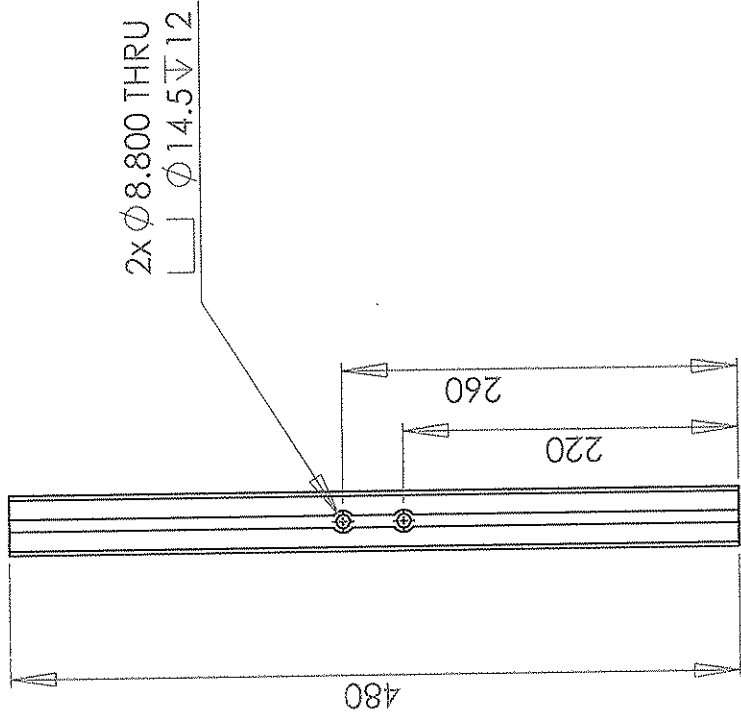
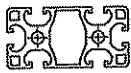
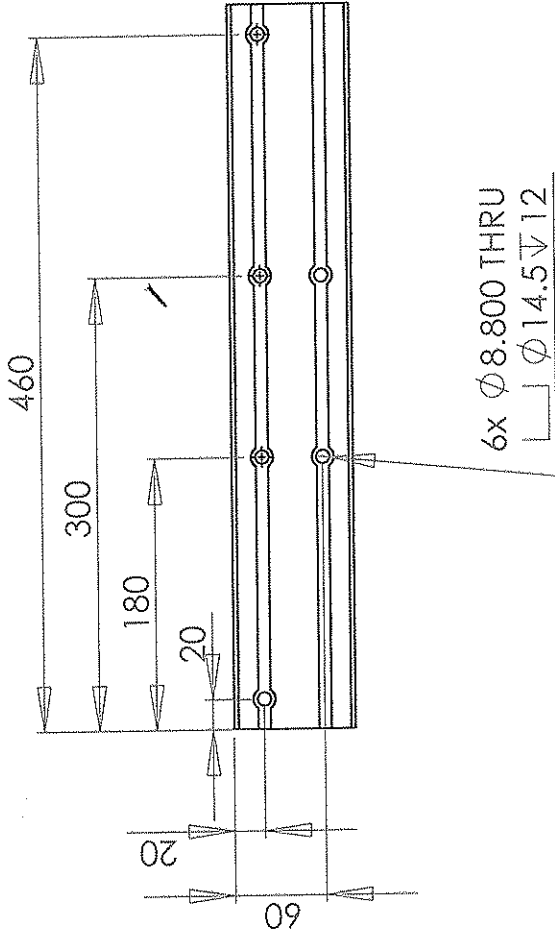
TITLE:  
 Connor Dodd  
 Sr.Project

SIZE DWG. NO. REV

**A** Horizontal

SCALE: 1:10 WEIGHT:

SHEET 1 OF 1



METRIC (mm)

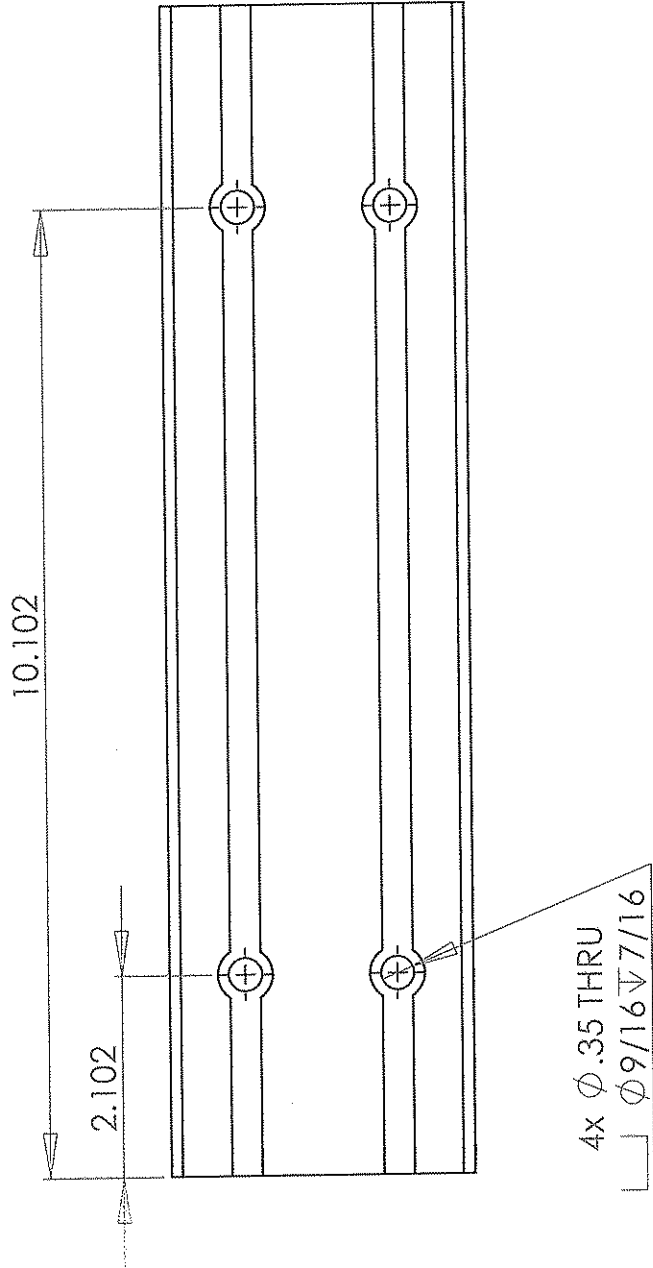
TITLE: **Conor Dodd**  
**Sr.Project**

SIZE DWG. NO. REV

**A** Horizontal

SCALE: 1:10 WEIGHT: SHEET 1 OF 1



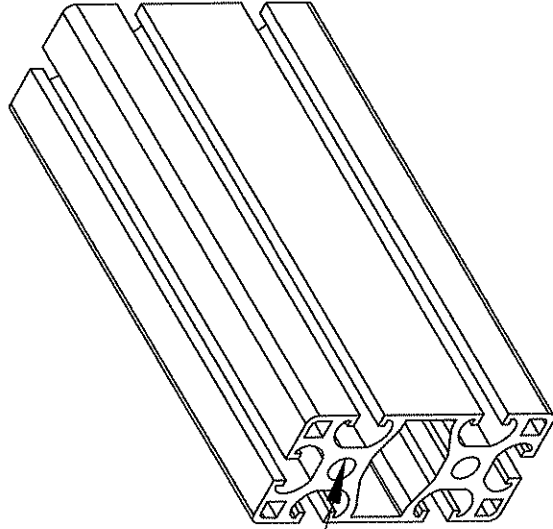
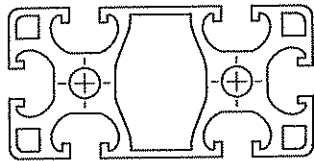
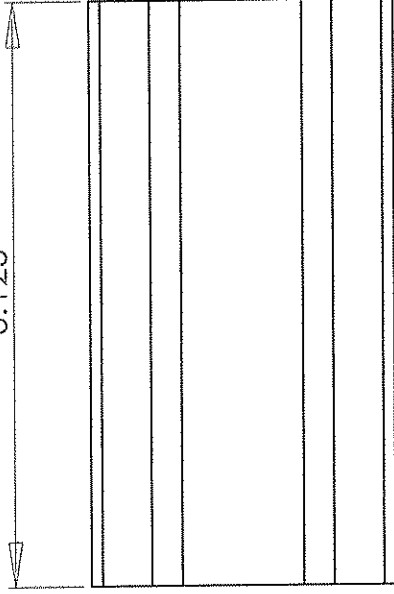


# USE PRE-CUT BAR

ENGLISH (in)  
 TITLE: **Conor Dodd  
 Sr. Project**

SIZE	DWG. NO.	REV
<b>A</b>		
SCALE: 1:5		WEIGHT: SHEET 1 OF 1

6.125



4 x M8 TAP

X2

English (in)

TITLE:

Conor Dodd  
Sr. Project

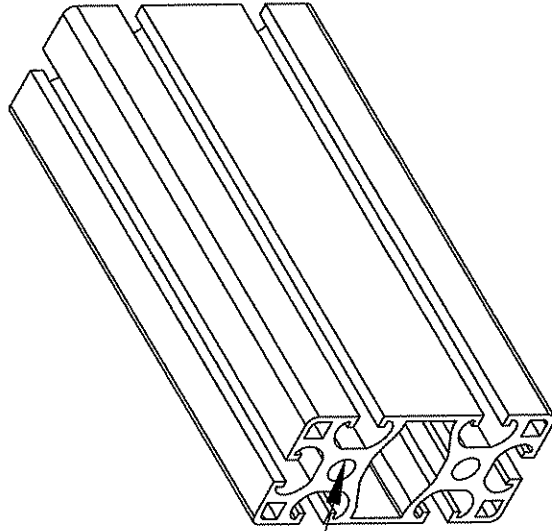
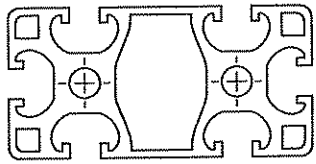
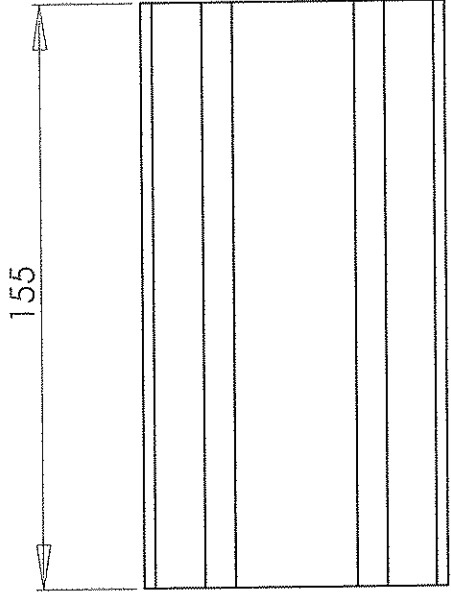
REV

SIZE DWG. NO.

**A** 6.125in Bar

SCALE: 1:2 WEIGHT:

SHEET 1 OF 1



4 x M8 TAP

METRIC (mm)

TITLE:

Conor Dodd  
Sr. Project

REV

SIZE DWG. NO.

**A** 155mm Bar

SCALE: 1:2 WEIGHT:

SHEET 1 OF 1