

Mobile Solar PV Trainer

Final Report

June 6, 2014

By

Matthew Myers

mmyers05@calpoly.edu

Matthew Clause

mclause@calpoly.edu

Lawrence Smith

ltsmith@calpoly.edu

Sponsor:

Professor Dale Dolan

Statement of Disclaimer

Since this project is a result of a class assignment, it has been graded and accepted as fulfillment of the course requirements. Acceptance does not imply technical accuracy or reliability. Any use of information in this report is done at the risk of the user. These risks may include catastrophic failure of the device or infringement of patent or copyright laws. California Polytechnic State University at San Luis Obispo and its staff cannot be held liable for any use or misuse of the project.

TABLE OF CONTENTS

1. Introduction:	6
2. Background:	7
3. Objectives:	10
4. Ideation:	14
4.1: Methods	14
4.2: Frame Subsystem Part 1 (Azimuth/Elevation paradigm)	14
4.2.1: Single-axis tracker on wheels (matches the existing PV trainers)	15
4.2.2: Single axis tracker mounted on a large thrust bearing (turntable)	16
4.2.3: Single axis tracker mounted on vertical shaft (periscope)	17
4.2.4: Frame-mounted, fixed point of rotation on wheels (the “foot”)	18
4.3: Frame Subsystem Part 2 (Alternate)	19
4.3.1: Dual axis tracker using the ‘telescope’ paradigm	21
4.4 Concept Evaluation: Frame Subsystem	22
4.5: Drive System Ideation	24
4.5.1 End-mounted chain or belt system with a traveling tensioner	25
4.5.2 Axle mounted chain, belt or gear system	26
4.6 Concept Evaluation: Drive Subsystem	26
4.7: Panel Cooling Ideation	27
4.7.1: Forced convection using cooling fans mounted below the panel	27
4.6.2: Evaporative cooling from the front of the panel using either pressurized misters or drips. ...	28
4.6.3: A closed-loop circulating feedwater system (using a pump, heat exchanger and/or water reservoir).....	29
4.6.4: Refrigeration Cycle.....	30
4.6.5: Combinations of systems 1-4.....	30
4.7: Panel Cooling ‘Proof-of-Concept’ Testing	31
4.8 Concept Evaluation: Cooling Subsystem	36
5. Final Design Description:	38
5.1: Design Summary	39
5.2: Frame	41
5.3: Traveler & Counterweight	43
5.4: Student Work Surface	44
5.5: Inverter & Battery Mounts	46
5.6: Shafts & Bearings	47
5.7: Panel Mounting Structure	49
5.8: Drive System	50
5.9: Cooling System	51
5.10: Electrical Box & Wiring	51
5.11: Control Panel & User Interface	53

5.12: User Interface Operating Instructions	55
5.13: Sensors.....	56
5.14: Microcontroller.....	58
6. Design Considerations	60
6.1: Design Analysis:.....	60
6.1.1 Analysis of Steel Drive Shaft Deflection.....	60
6.1.2 Analysis of Stress in Steel Drive Shaft	62
6.1.3 Angular Deflection of Frame	63
6.1.4 Tipping Condition Analysis.....	71
6.2: Safety Considerations:	75
6.3: Maintenance & Repair	77
6.4: Cost Analysis	78
7. Manufacturing.....	79
7.1 Fabrication	79
Resizing the Aluminum T-Slot	79
Caster Wheel Mounting Plates	80
Encoder Brackets	81
Inverter Mount, Battery Mount, and Work Surface	82
Work Surface Folding Legs	83
7.2 Building the Microprocessor Board	84
Building the ME405 Board	84
Setting up the LCD screen	86
Configuring the Keypad.....	87
Connecting the DS18B20 Temperature Sensors.....	87
Connecting the Avago Absolute Encoders	88
8. Design Verification Testing:.....	89
8.1: Positioning System Testing.....	89
8.2: Cooling System Testing	93
8.3: Electrical Subsystem Testing	97
8.3: Testing Summary	97
9. Suggested Improvements	99
9.1 Structure	99
9.2 Control Panel.....	99
9.3 Wiring	100
9.4 Encoder Brackets	100
9.5 T-Slot Assembly	100
10. Conclusion:.....	101
Works Cited:.....	102

Appendix A: QFD Analysis.....	103
Appendix B: Cooling Test Raw Data	104
Appendix C: Coordinate Transformations.....	105
Appendix D: Gantt Chart.....	106
Appendix E: Bill of Materials.....	109
Appendix F: Drawing List and Part Drawings	117
Appendix G: Software Task & State Diagrams	ccxii
Appendix H: Software Documentation.....	ccxvi

1. INTRODUCTION:

The 'Mobile Solar PV Trainer Project' is a capstone design project overseen by the California Polytechnic State University Mechanical Engineering department. The aim of this project is to design, construct and test a versatile and robust laboratory training rig for the Cal Poly Electrical Engineering department. This final project report contains a detailed description of the trainer's desired specifications, our ideation process, and a thorough description of the selected trainer design including part drawings, assembly documentation and a bill of materials. Manufacturing documentation, testing results and operating instructions are also included.

This project was inspired by the desire of one or more California Polytechnic State University electrical engineering professors to add laboratory components to two established upper division electrical engineering courses in photovoltaic panels and solar power. In order to demonstrate basic and advanced principles of photovoltaic panels in a laboratory setting, a custom designed test rig is required which will serve as a primary data collection device. This test rig must be easy for faculty members to configure, engaging for students to use and rugged enough to survive outdoor storage and use. The rig must support a wide and versatile array of sensors and control systems as it will be used for many different (and as of yet undetermined) laboratory experiments. At a minimum, the rig must support a wide variety of commercially available solar panels, be capable of tracking the sun along two different axes, possess a student work surface, and be capable of decreasing the steady state operating temperature of a panel.

The mobile solar PV trainer team consists of three Cal Poly undergraduate mechanical engineering students: Matthew Myers, Lawrence Smith and Matthew Clause. We are 4th and 5th year students with backgrounds in mechanical design, mechatronics, and solar tracking systems. We are excited to help add a new laboratory component to the sustainable energy curriculum here at Cal Poly. Cal Poly electrical engineering professor Dale Dolan has served a dual role as both the project's sponsor and as a technical advisor.

2. BACKGROUND:

Solar Tracking:

Photovoltaic solar panels are the most efficient at producing electricity when they are oriented perpendicular to (i.e. directly facing) the sun. In order to improve efficiency, many solar arrays track the sun as it changes position throughout the day. Solar tracking is accomplished in a variety of ways, with some panels rotating on a single axis, and others, on two axes. One common method of solar tracking is called tilted single-axis tracking (TSAT). A solar panel array employing TSAT has a fixed elevation angle based on its geographical location and rotates slowly along an east-west axis as the day progresses. Solar panels mounted in this manner produce considerably more energy throughout a given day (when compared to fixed-mount designs) since they are consistently oriented towards the sun. Since they are single axis trackers, they achieve this efficiency increase using only one motor and controller. Section 4.2 of this report will examine TSAT tracking in more detail and explain how it influenced our frame design.

Dual-axis tracking, by contrast, adds an additional degree of freedom and allows a panel to directly face the sun despite seasonal changes in solar alignment. Despite this efficiency increase, most solar panels only rotate on one axis because dual-axis energy gains are often relatively low. This is true because solar panels with 5 degrees or less of angular misalignment to the sun will still produce more than 99% of the power of a perfectly aligned panel (see image to the right)¹.

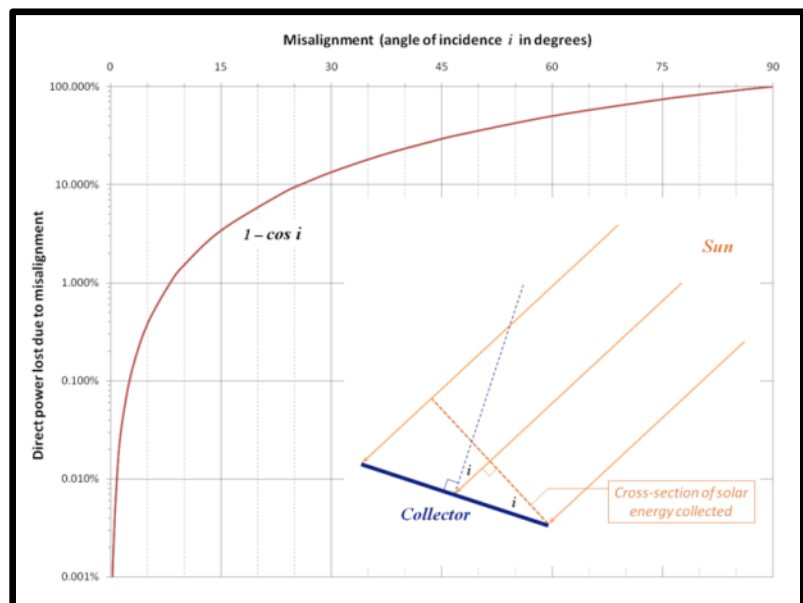


FIGURE 1 - PV SOLAR PANEL POWER OUTPUT AS A FUNCTION OF MISALIGNMENT ANGLE

¹ "Solar Tracker." *Wikipedia*. Wikimedia Foundation, 12 Jan. 2013. Web. 04 Dec. 2013.

That aside, for our rig, we decided to specify a maximum misalignment angle of 1 degree. We want our solar trainer to have the look and feel of a professional piece of equipment, and eliminating as much slop, backlash, and misalignment as possible will give students a better lab experience. Our rig will also feature dual axis tracking capability, along with the ability to control panel orientation manually, powered (motorized) positioning, and automatic motorized tracking. This report will cover the concept generation process that we used to come up with our positioning system, as well as other concepts that we considered.

Lab Equipment Design:

In our research on how to engage students with electronic lab equipment, we found that it is helpful to include visual indications of a measured quantity. As such, we hope to provide analog backups or additional visual cues for measurements like angular position, voltage produced, and irradiance. For example, a dial indicator that turns according to panel efficiency would be aesthetic as well as functional. Additionally, relating experiments to situations a student will see in the workforce is important because it demonstrates a lab's relevance. We want students to make meaningful connections between learning materials and their real life experiences.

Weatherproofing:

Since this test rig will be stored outdoors, we have decided to design it to an ingress protection rating of IP53². This standard specifies that:

- 1) Ingress of dust is not completely prevented, but dust is prevented from entering in a quantity that would impair the functioning of the object.
- 2) Water hitting the object at any angle up to 60 degrees from normal will not have any harmful effect.

Temperature Effects:

A final consideration about solar panel performance that we researched is the effect of elevated temperature. Increased operating temperature makes solar panels less efficient. Our rig will incorporate a variable cooling system so that students can measure panel performance at different steady state temperatures. To our knowledge, this will be the first time that temperature control has been integrated into the solar instrumentation lab at Cal Poly.

² "IP RATINGS EXPLAINED." *IP Ratings*. OKW Enclosures Inc., 31 Jan. 2012. Web. 04 Dec. 2013.

Previous Projects:

We have examined the reports of previous senior projects that dealt with solar tracking, including the projects that generated the existing rigs in the courtyard of the Cal Poly Electrical Engineering building. We have also studied senior projects that produced work that we haven't seen in person, such as the PVC frame Solar Tracker project by A. Hsing in 2010. Studying these previous projects has provided us insight into which design elements have worked well and which could use improvement. Dr. Dolan has identified several shortcomings of even the latest (best) version of the PV trainer, including excessive slop in the panel alignment, difficulty navigating rough ground, inconveniently placed displays, and incomplete weatherproofing. Our goal is to expand on the excellent work done by these previous senior project teams and to create the best PV trainer yet.



FIGURE 2 - PREVIOUS SOLAR PV TRAINER

3. OBJECTIVES:

Since education remains the rig's primary function, it must be engaging to use, safe, and quickly reconfigured. In addition, for the PV trainer to meet basic customer specifications, it must be both mobile and versatile.

In order to insure that the engineering specifications chosen closely matched the sponsor's requirements, a quality function deployment (QFD) analysis was performed. Using a 'house of quality' comparison matrix, individual specifications were sorted in terms of their importance for meeting specific customer requirements; two sets of potential customers were compared (lab instructors and students). Finally, previous designs were benchmarked against these new specifications to insure that the finished product will provide a better lab experience than previous efforts. This QFD matrix can be found in Appendix A.

Since the rig will be both used and stored outdoors, it must survive prolonged exposure to both UV radiation and rain. As such, all components that are not directly shielded from sunlight will possess a UV package that guarantees functionality for up to two years. Similarly, all plugs, motors, and other electronic components will either be able to function reliably after repeated water exposure or will be housed in an IP53 rated (or equivalent) container. We will not have the IP rating of this container professionally verified.

The rig must be capable of physically securing a wide range of photovoltaic panels. One advantage of our rig highlighted by our QFD analysis is its flexibility; this rig will support more panel types than previous rigs. While non-rectangular panels will not be accommodated, the rig will comfortably support panels as large as 2000mm wide x 1000mm tall x 55mm deep and as small as 50mm wide x 500mm tall x 30mm deep. Panels will be mounted in their 'portrait' orientation (longest dimension vertical) to conserve space. It should take two individuals no more than 10 minutes to reconfigure the rig with a new panel.

This PV test platform will need to be easily transported across pavement, concrete, carpet, dirt, and grassy surfaces. Our quality function deployment (QFD) analysis segmented mobility into several categories. The rig will be capable of navigating vertical ledges such as door seals that are up to 2 inches tall and remain stable on slopes up to 20 degrees. It must fit through a standard single door that is at minimum 32" wide and 80" tall and will be capable of navigating ADA accessible wheelchair ramps which are at minimum 36" wide (note: the trainer does *not need* to fit through standard doorways when equipped with 2000mm tall panels; the largest panels that need to be accommodated when moving through doors are 66 inches tall). Handles or grip points will be provided on convenient locations to facilitate safe transportation. In order to navigate hallways or other tight spaces, the rig will be able to pivot roughly within its own footprint.

While completing our QFD analysis, we considered the rig's two-axis positioning capability to be one of the most important specifications for the system. In elevation, the panel

will be able to position to anywhere between 0 (parallel to the ground plane) and 90 degrees (perpendicular to the ground plane) in 1 degree increments to within a tolerance of ± 1.0 degree relative to the frame. The panel must take no more than 2 minutes to cross its entire range in elevation. The rig must also be capable of azimuth adjustment either by moving the panel within the frame or by altering the orientation of the entire rig. This measurement must similarly be adjustable in 1 degree increments to within a tolerance of ± 1.0 degree relative to the ground plane. The panel must take no more than 5 minutes to cross its entire range in azimuth if moving relative to the frame and/or no more than 8 minutes if the frame's orientation must be changed. A 'hand crank' type device will be included to allow users to adjust the panel's position manually³. All mechanical components will be designed to survive 10 years of laboratory use.

The rig will be capable of reducing the operating temperature of the panel by at least 20 degrees Fahrenheit from its steady-state operating temperature (presuming that the operating temperature is above 120 degrees Fahrenheit). It should reach its cooled steady state temperature in 10 minutes or less.

The rig must include a wide variety of sensors to monitor its efficiency and electrical outputs. These sensors will include a measure the panel altitude angle, the panel azimuth angle, the panel temperature, the ambient solar irradiance, the module voltage, the module current, the battery voltage and the charge controller current. All voltage sensors will be accurate to within $\pm 0.5V$ and all current sensors will be accurate to within $\pm 0.25A$. Most to all of the instantaneous readouts will include analog backups and all will be legible under bright light or direct sunlight. Provisions for 'relative reference frame solar tracking' will also be provided in the form sensors capable of identifying the sun's location in the sky.

The system must be 'electrically flexible' to accommodate many possible lab procedures. As such, interchangeable plugs (e.g. banana plugs) will be used in place of permanent electrical connectors wherever possible. All subsidiary electrical systems, such as sensors and motors, will be powered off of the rig's battery. The remaining electrical components such as the inverters will operate off of the panel during normal use.

A simple and friendly user interface will be critical to providing an enjoyable lab experience. Our QFD analysis pointed to deficiencies in previous senior projects in the area of user friendliness and student engagement. Sensor readouts and controls will be located in close proximity to each other and any automatic functionality will be governed by appropriate safety appliances (such as hard stops or limit switches). In addition, the device will include a flat and stable student work surface of 5 square feet about 3 feet off of the ground. In order to reduce wasted lab time, the rig will take two students no more than 15 minutes to appropriately configure.

³ This feature was eventually removed; please consult Section 5.8 for a detailed explanation.

Finally, the rig should cost no more than \$1500 dollars to construct and assemble. The full list of our engineering specifications appears on the follow page.

TABLE 1 - PV SOLAR TRAINER ENGINEERING REQUIREMENTS SUMMARY

Spec. #	Description	Requirement (units)	Tolerance	Risk Level	Compliance*
1	Max Panel Length	2000 mm	Max	Low	A, S
2	Min Panel Length	500 mm	Min	Low	A, S
3	Max Panel Width	1000 mm	Max	Low	A, S
4	Max Panel Depth	55 mm	Max	Low	A, S
5	Min Panel Depth	30 mm	Min	Low	A, S
6	Panel Replacement Time	10 minutes	Max	Medium	T, S
7	Navigate Various Surfaces	Concrete, Grass, Dirt	Min	Low	A, S
8	Navigate Vertical Ledges	2in	Min	Medium	A, T, S
9	Stable on Slopes During Transport	20 degrees	Min	Medium	A, T, S
10	Easy to Maneuver During Transport	Able to Pivot in Own Footprint	Min	Medium	T, I
11	'Collapsed' Width	32 inches	Max	Medium	A, I
12	'Collapsed' Height	80 inches (not including 2000mm panels)	Max	Medium	A, I
13	Panel Elevation Range	0-90 degrees	+/- 1.0 degree	High	A, T, I
15	Elevation Adjustment Time	2 minutes	Max	Low	T
16	Panel Azimuth Range	0-360 degrees	+/- 1.0 degree	High	A, T, I
18	Azimuth Adjustment Time	8 minutes	Max	Medium	T
19	Measure Panel Voltage	Sensor Included	+/- 0.5 volts	Low	A
20	Measure Panel Current	Sensor Included	+/- 0.25 amp	Low	A
21	Irradiance	Measure Ambient Irradiance	Min	Medium	I
22	Sensor Refresh Rate	0.5 sec	Min	Low	T
23	Sensor Readout Visibility	Legible in Direct Sunlight	Min	Medium	I
24	Reduce Panel Operating Temp.	20 °F	Min	High	A, T
25	Panel Temp Reduction Time	10 minutes	Max	High	A, T
26	Panel Temp Sensor	Sensor Included	+/- 0.5 degrees	Medium	A
27	Student Work Surface	5 ft ² ; 4 feet off of ground	Min	Low	A
28	Weatherproofing	IP53	Min	Medium	A, T
29	Backup Control Systems	Manual Controls Included	Min	Low	I
30	Backup Sensing Systems	Backup readouts for all sensors	Min	Low	I
31	Mechanical Component Life	10 years	Min	Medium	A
32	Electrical Flexibility	Plugs used where possible	Min	Low	I
33	UV package/shielding	Parts specified for 2 years	Min	Medium	A
34	Safety/Fail-safes	Limit Switches	Min	Low	A, T, S
35	Construction Cost	\$1500	Max	Medium	I

*compliance represents a method for verifying design requirements; Analysis (A); Testing (T); Similarity to Existing Designs (S); Inspection (I)

4. IDEATION:

4.1: METHODS

After defining the engineering specifications for our project, we began design ideation. For development purposes, we divided the project into three primary subsystems: the frame design, the cooling system, and the drive mechanism. We applied brainstorming techniques and decision-making processes to each of these three subsystems separately. This allowed us to find the best solution for each problem, so that we could then combine them to create a full design that best satisfied engineering requirements. We used Pugh matrices to compare and evaluate our brainstormed ideas. In this section of the report, we will explain different concepts that we generated for each subsystem, compare them to one another, and finally explain why we chose our final selection.

4.2: FRAME SUBSYSTEM PART 1 (AZIMUTH/ELEVATION PARADIGM)

We considered five distinct frame designs, which incorporated two different control/axis paradigms to achieve dual-axis tracking:

The first four of these designs fell into the paradigm that we dubbed “pure” elevation/azimuth tracking. For each of these systems, one axis (called the ‘elevation axis’) allowed the panel to move in pitch, relative to the horizon, while a second axis (called the ‘azimuth axis’) allowed the panel to move in yaw, relative to lines of latitude. This paradigm offered the advantage of providing a relatively intuitive manual control interface: one axis is adjusted relative to the horizon and the other is adjusted relative to a latitude reference (such as magnetic North). Of the two paradigms, the pure elevation and azimuth approach closely matched both Professor Dolan’s original specifications and the existing PV trainer rigs. That said, for this ‘pure azimuth’ system, dual axis tracking of the sun requires continuous adjustment of both axes. Our four frame concepts appear on the following pages.

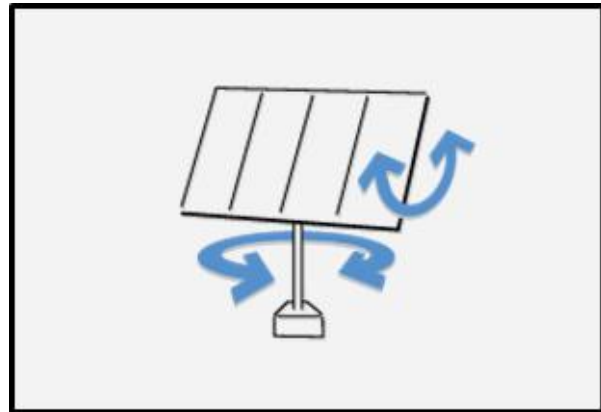


FIGURE 3 - AZIMUTH-ELEVATION SOLAR TRACKER

4.2.1: SINGLE-AXIS TRACKER ON WHEELS (MATCHES THE EXISTING PV TRAINERS)

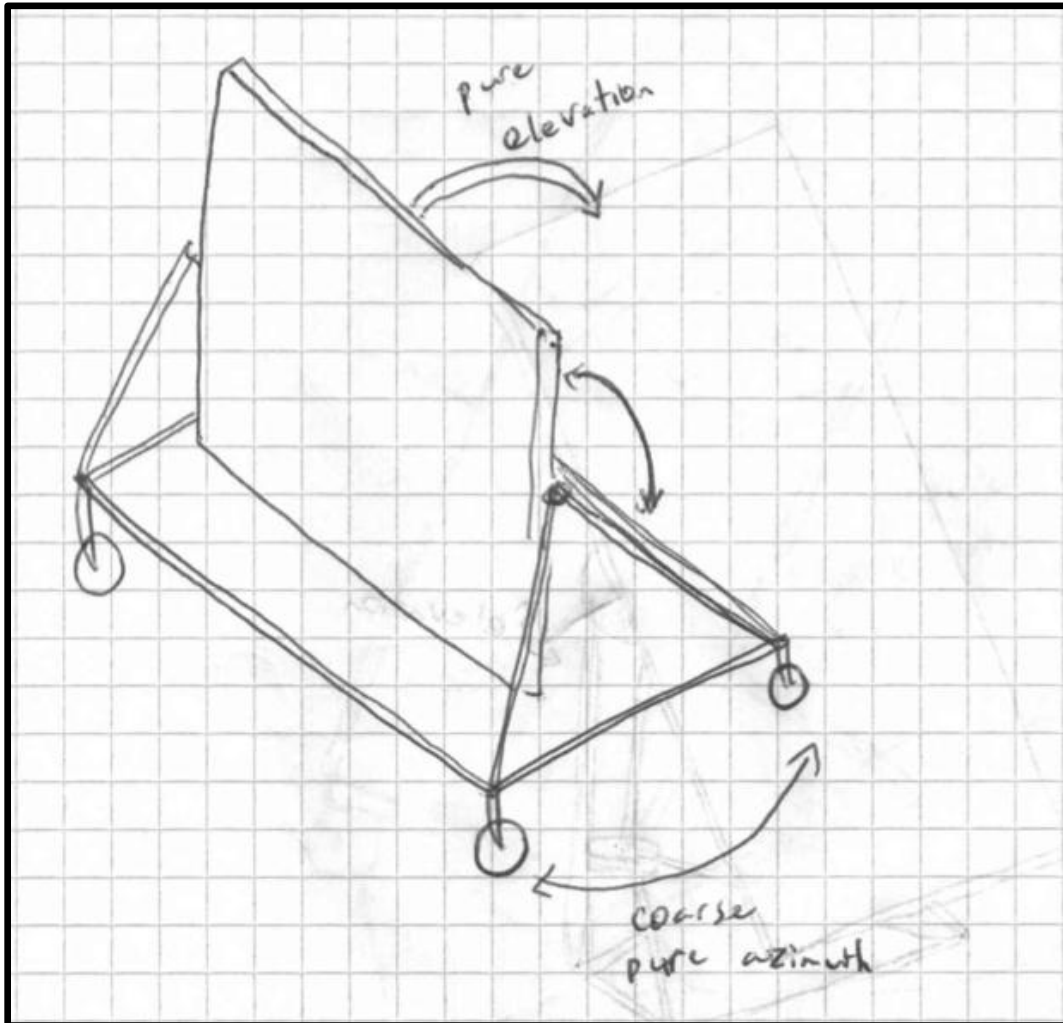


FIGURE 4 - SINGLE AXIS TRACKER SKETCH

Of the frame designs considered, the single axis tracker on wheels was the simplest. Essentially a copy of the existing PV trainer designs, a single 'elevation' axis is driven by a power system and the azimuth axis is controlled by physically turning the frame (on its wheels) between tests. This design is proven, mechanically simple and requires the fewest components. However, it is difficult to adjust the azimuth axis to a repeatable set point, especially if the rig is resting on uneven terrain. In addition, this design does not represent a major upgrade from the existing PV solar-trainer test rigs.

4.2.2: SINGLE AXIS TRACKER MOUNTED ON A LARGE THRUST BEARING (TURNTABLE)

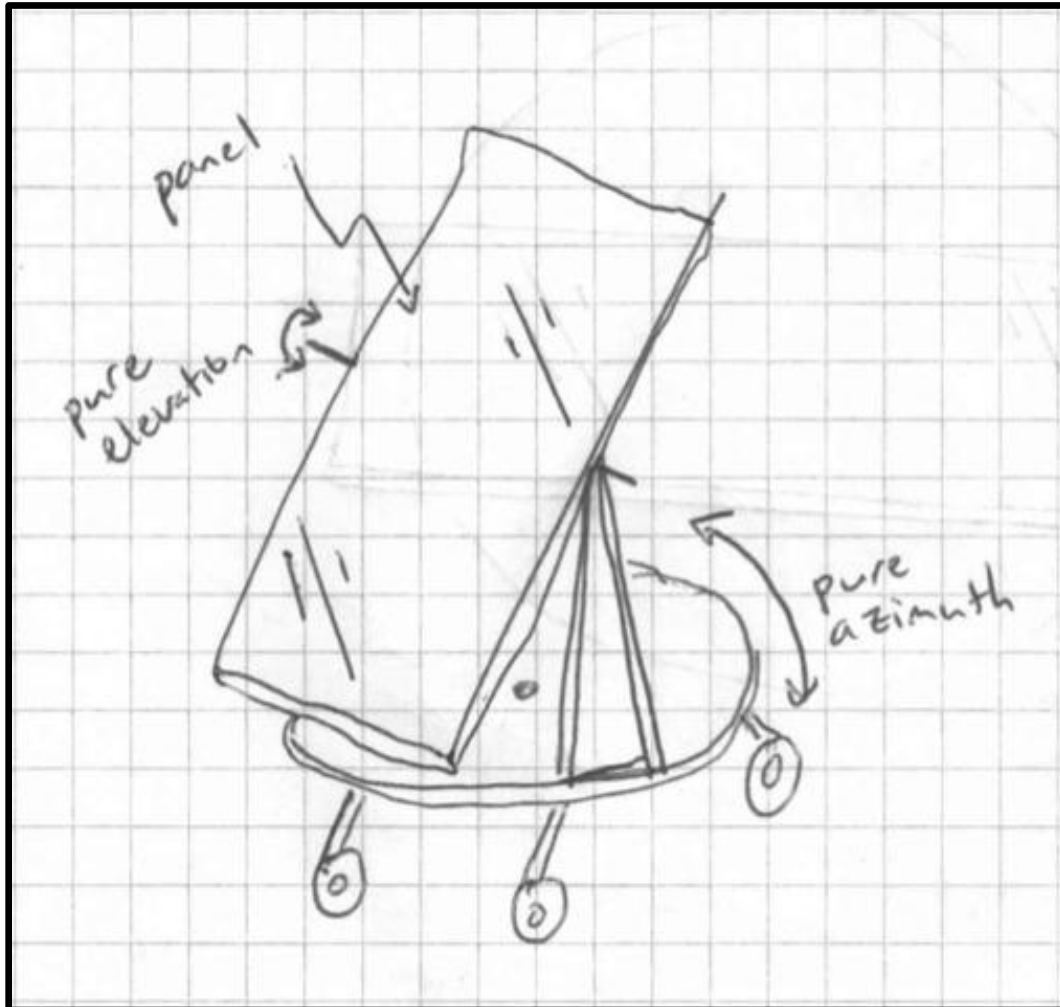


FIGURE 5 - SINGLE AXIS TRACKER ON TURNTABLE SKETCH

While relatively simple, the ‘large thrust bearing’ design seemed to present an unnecessary machining and manufacturing challenge. Inexpensive and weather resistant ‘turntables’ are commercially available; however they require a flat and sturdy support structure (such as concrete floor or large steel plate). This support structure could prove difficult to mill, machine or otherwise fabricate.

4.2.3: SINGLE AXIS TRACKER MOUNTED ON VERTICAL SHAFT (PERISCOPE)

FIGURE 6 - SINGLE AXIS TRACKER ON A VERTICAL SHAFT

The single axis tracker ‘vertical shaft design,’ replaces the large thrust bearing (turntable) with a single vertical shaft. This system eliminates the need for a complicated circular support structure. That said, since the new shaft supports the weight of the entire PV panel structure axially, thrust bearings would be required to transfer force from the shaft to the frame. The central shaft would also be exposed to damaging bending moments if the rig became unbalanced (such as in windy conditions).

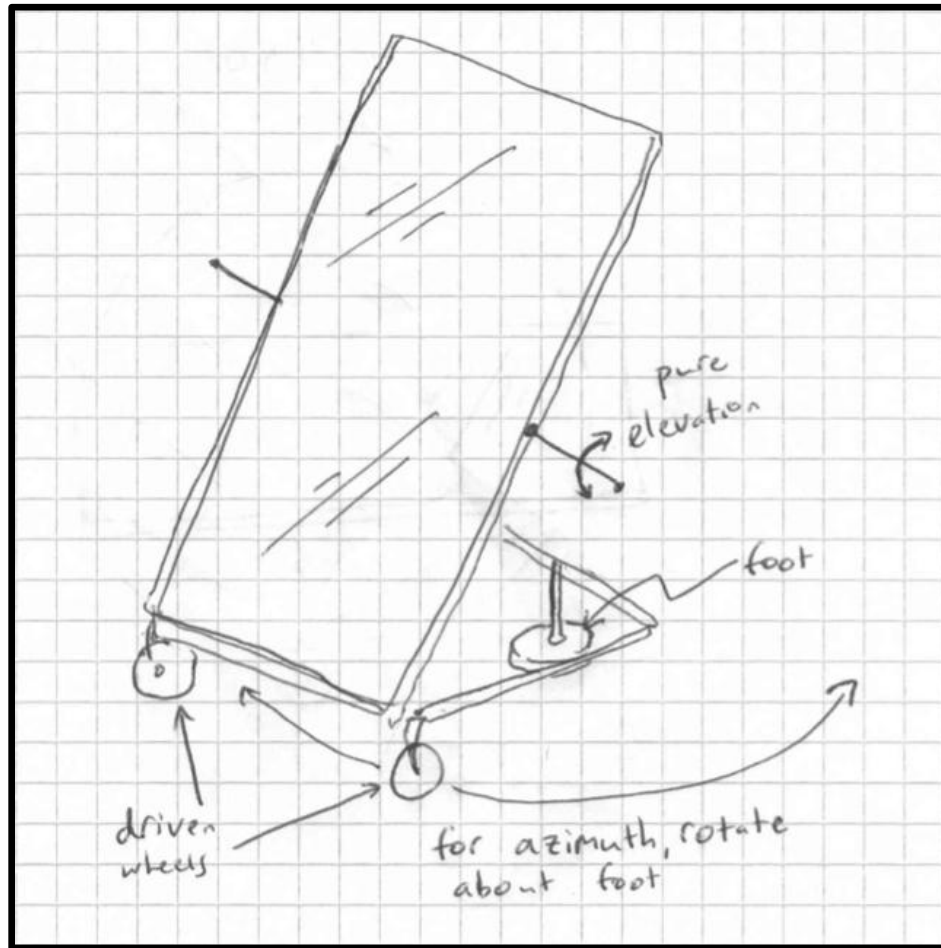
4.2.4: FRAME-MOUNTED, FIXED POINT OF ROTATION ON WHEELS (THE “FOOT”)

FIGURE 7 - PURE AZIMUTH ROTATION ABOUT FIXED FOOT

The 'fixed point of rotation' or 'foot design' represents an attempt to improve the performance of the 'single axis tracker on wheels' concept. Instead of requiring the operator to manually adjust the azimuth axis, more precise measurements can be accomplished by attaching motors and a control system to the wheels. A fixed rotation point (dubbed the 'foot') descends below the rig after it has been positioned, lifting two of the four tires into the air. The other two wheels are powered and swing the frame in an arc about the contact point. A sensor (such as a potentiometer) is incorporated into the 'foot' and precisely measures the frame's angular offset from its starting point.

While this design offered some advantages over the single-axis tracker on wheels (the baseline), it would still likely be defeated by rough terrain (such as gravel or grass). It would also require a relatively large plot of open ground on which to operate.

4.3: FRAME SUBSYSTEM PART 2 (ALTERNATE)

The second tracking paradigm that we considered was inspired by optical and radio telescope mounts. These telescopes do not configure their axes in the aforementioned fashion: one axis (the elevation axis) adjusts pitch, as before, while the second axis rotates *with* the elevation axis, thereby changing its pivot angle relative to the ground.



FIGURE 8 - MOUNT PLEASANT RADIO OBSERVATORY IN AUSTRALIA; NOTE THAT THE MOUNTING STRUCTURE DOES NOT EMPLOY THE PREVIOUS "AZIMUTH/ELEVATION" POSITIONING SYSTEM

This design also allows the dish (in our case PV panel) to face any direction in the sky. That said, it does not maintain the panel in any particular orientation relative to the ground. It also complicates manual positioning (since there is no longer a logical 'azimuth angle' to set).

Concerns aside, if this paradigm was used, one axis (the analog to the elevation axis) could conceivably be interpreted as the 'time of year' axis while the second (the new axis) could be interpreted as the 'time of day' axis. A rig constructed in this way would therefore allow for a greatly simplified tracking algorithm (since tracking the sun during the day would only require the user to adjust one axis at a time).

Such an arrangement would also allow a professor to easily simulate the motion of fixed-tilt, single-axis trackers (TSAT); he would achieve this by facing the panel to the south, setting the main axis to a particular angle (say thirty degrees) and adjusting the secondary axis to follow

the sun. If used in this manner, the rig would behave as an 'adjustable' fixed tilt single axis tracker.

He could similarly use such a rig to simulate a 'polar aligned' single axis tracker (PASAT); this would be done by aligning the main axis towards the North Star, Polaris, and adjusting the secondary axis as before.

While both of these tracking systems could conceivably be replicated using a standard 'azimuth/elevation' rig, each would require a dedicated and complex two-axis tracking algorithm. If a telescope-type design was selected, TSAT and PASAT could both be simulated manually.

Additionally, if the lab instructor desired, he could configure the rig to use the same positioning system that the current PV Trainer does, by locking the secondary axis in the center of its range. He could then rotate the entire frame to achieve coarse azimuth control, and use the primary axis for elevation control.

Finally, it would still remain easy for the professor to use the rig in a more conventional manner. If he wished to simulate a horizontal single axis tracking (HSAT) for example, he would simply align the trainer east-west, lock the secondary (time of year axis) at zero and sweep across the sky using the primary (elevation analog) axis only. In addition, coarse azimuth control could be achieved by rotating the entire rig on its wheels, as the previous PV trainer senior projects did it.

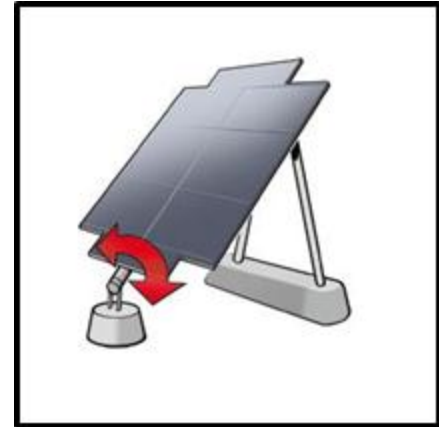


FIGURE 9 - FIXED-TILT, SINGLE AXIS SOLAR TRACKER



FIGURE 10 - EAST-WEST PANNING HORIZONTAL SINGLE AXIS TRACKERS

4.3.1: DUAL AXIS TRACKER USING THE 'TELESCOPE' PARADIGM

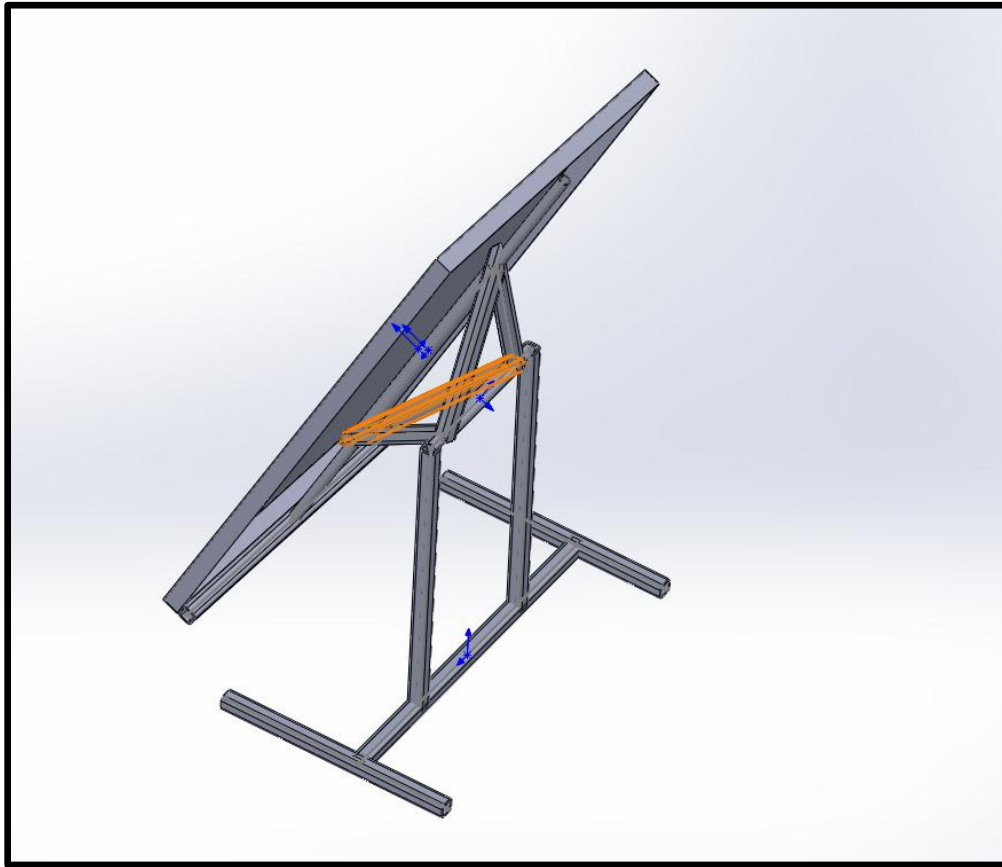


FIGURE 11 - TELESCOPE-INSPIRED DUAL AXIS TRACKER FRAME CONCEPT

A dual axis tracker using a telescope-inspired tracking system offers several distinct advantages. Such a system does not require a complicated base structure or a heavy vertical shaft. In addition, it allows for precise dual axis tracking control within an easily designable range.

That said, without a counterweight to offset the mass of the panel and its mounting structure, this design is inherently imbalanced about the 'main pivot' (elevation analog); this effect is maximized when the panel is oriented vertically in reference to the ground. Reducing this imbalance requires one to reduce the overturning moment arm of the panel and its mount, but unless the panel mounting structure is offset from the main axis pivot upwards of one foot (see figure 12, next page), the secondary axis will not be able to span a full 180 degrees. Implementing this design therefore requires one to find a balance between accommodating a larger range of motion and reducing the system's imbalances. Assuming a compromise in favor of stability, this means that the panel will not be able to face any direction in the sky (as it could with the turntable and periscope designs) without first resetting the base using 'coarse azimuth' adjustments.

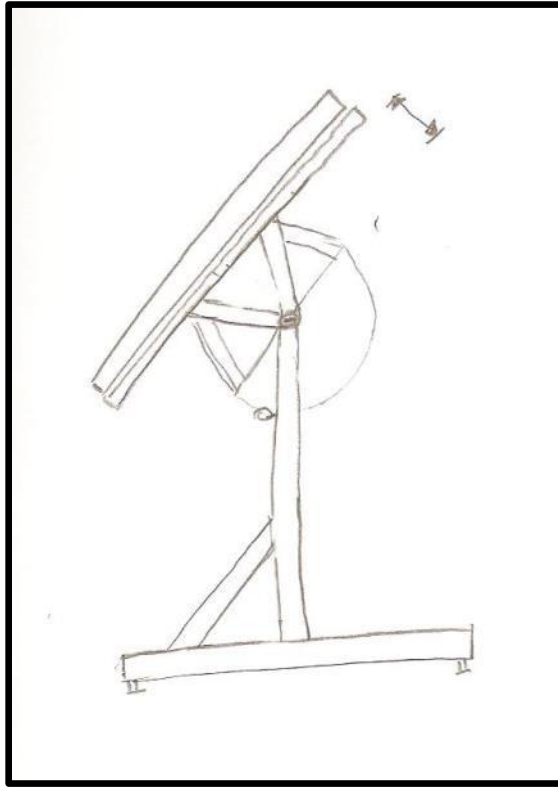


FIGURE 12 - SKETCH SHOWING OFFSET BETWEEN PANEL AND PRIMARY AXIS OF ROTATION

4.4 CONCEPT EVALUATION: FRAME SUBSYSTEM

After generating the five frame subsystem ideas described above, we needed to pick the one best suited to solving our design challenge. The Pugh Matrix on the following page ranks each of the proposed designs to a datum: the existing PV Solar Trainer (as pictured in section 2).

TABLE 2 - FRAME SUBSYSTEM PUGH MATRIX

Criteria	Turntable Azimuth + Pure Elevation	"Telescope" Positioning	Previous Senior Project Design	"Periscope" Azimuth, Pure Elevation	Wheels with Fixed Rotation Point
Cost	+	-	Datum	-	-
Difficulty of Assembly	+	+	Datum	-	+
Additional Axis Adjustment Capability	S	+	Datum	S	S
Number Components/ Subsystems	-	-	Datum	-	S
Mobile	S	S	Datum	+	-
Full Range Azimuth + Elevation Positioning	S	+	Datum	S	+
Accurate positioning to +/- 1.0 degrees	+	+	Datum	+	+
Small footprint	-	S	Datum	+	S
Self-Powered	S	S	Datum	S	-
Sum of +'s	3	4	Datum	3	3
Sum of -'s	2	2	Datum	3	3
Sum of S's	4	3	Datum	3	3

This Pugh matrix reflects well on our ideation phase – all of our proposed ideas scored above the datum. However, this analysis did not produce one “front runner” idea that was far superior to the others. In the end, we made a decision to employ the “telescope” style positioning system described in section 4.3.1 for a few key reasons:

- 1) Telescope positioning satisfies the basic criteria of providing repeatable dual axis position control.
- 2) Telescope positioning is a very close analog to the ubiquitous TSAT mounting system.
- 3) Telescope positioning allows for very accurate position measurement over a realistic range of motion.
- 4) Telescope positioning presents a significant update to all previous Senior Project designs.

4.5: DRIVE SYSTEM IDEATION

Several drive mechanism designs were considered; these included: chains, belts, gears and linear actuators. Chains (specifically bicycle chains) are inexpensive, easy to source, sturdy and high strength. Unfortunately, they tend to rust if exposed to the elements and require regular lubrication. As a relevant example, the chain used on the existing PV Solar Trainer shows clear signs of rust and weathering from prolonged outdoor storage. Chains also hold slack and thereby decrease the accuracy of the positioning system (a major shortcoming of the current test rig). Timing or V belts relieve many of the issues associated with chains but unfortunately come with their own set of drawbacks; while a timing belt will not rust when exposed to moisture, it may degrade after prolonged UV exposure or physically slip if the appropriate level of tension is not maintained. Ball-screw type linear actuators require low torques to adjust, are self-locking, are self-contained and are typically weather resistant. They are often very expensive and difficult to purchase in small numbers however. They may also lack the large range of motion necessary to adjust a panel between 0 and 90 degrees in elevation. Spur gears have tight tolerances, are accurate, common and easy to obtain. That said, they are relatively expensive and may rust if exposed to water.

Regardless of the drive method used, a mechanical advantage lies in making position adjustments as far as possible away from the pivot axis. Increasing the moment arm of the drive system both decreases the magnitude of the force needed to move the structure (meaning less expensive motors and motor drivers) and increases the backlash threshold needed to stay within a tight angular position tolerance. For this reason, if one uses a chain or belt system, one should place the chain/belt attachment points as far away as possible from the pivot. If one mounts the belt wheel or chain sprocket directly on the drive shaft, as large a wheel or sprocket as possible should be used. For the same reason, larger gears are more desirable than smaller ones.

After eliminating linear actuators for cost reasons⁴, we developed two distinct drive mechanism designs:

4.5.1 END-MOUNTED CHAIN OR BELT SYSTEM WITH A TRAVELING TENSIONER

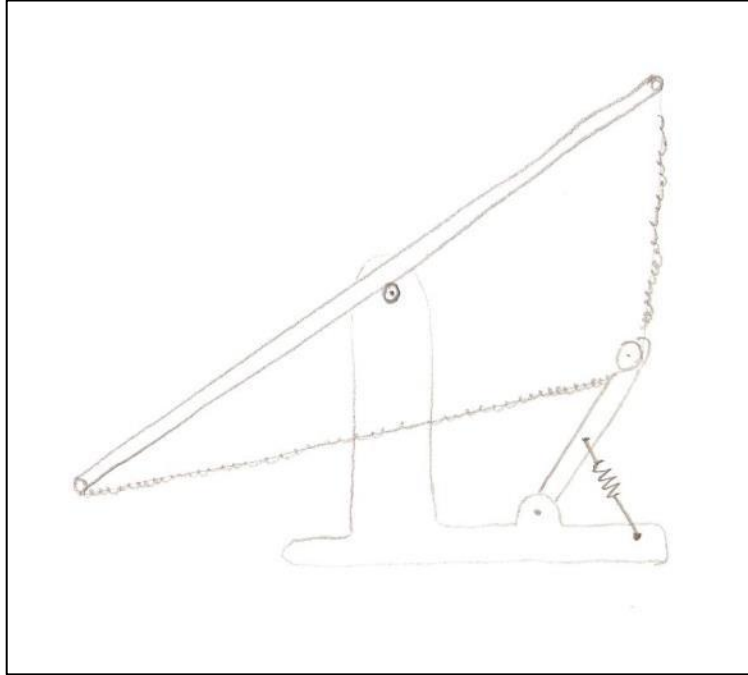


FIGURE 13 - TENSIONED END-MOUNTED CHAIN DRIVE SKETCH

An end-mounted chain or belt driven power system would allow for precise angle adjustments without large custom components. That said, since the effective length of the chain changes as the panel pivots, a tensioner would be required to maintain the panel's position. This tensioner would consist of a lever-mounted sprocket or guide wheel. This lever floats with the chain as it moves, maintaining a downward force on it using either a spring or a weight. A drive motor which floats with the tensioner would spin the guide sprocket to adjust the panel angle.

Such a system would likely require a brake mechanism to hold the panel securely in place during data collection (especially in a windy location). This is because undesired panel angle movements might occur whenever a force is applied to the panel which was sufficiently strong to lift or disturb the tensioner.

⁴ This conclusion ultimately proved unfounded; please consult Section 5.8 for a detailed explanation.

4.5.2 AXLE MOUNTED CHAIN, BELT OR GEAR SYSTEM

An axle mounted drive system would only be limited in its locking stability by the holding torque of the drive motor. The challenge of implementing such a system lies in finding a sufficiently large gear, sprocket or belt pulley that is available at an acceptable cost.

4.6 CONCEPT EVALUATION: DRIVE SUBSYSTEM

Please reference Section 5.8 for a detailed description of which drive system was eventually selected (and why).

4.7: PANEL COOLING IDEATION

Several possible solutions for cooling the solar panel were developed in depth and the preferred among them were empirically tested.

4.7.1: FORCED CONVECTION USING COOLING FANS MOUNTED BELOW THE PANEL

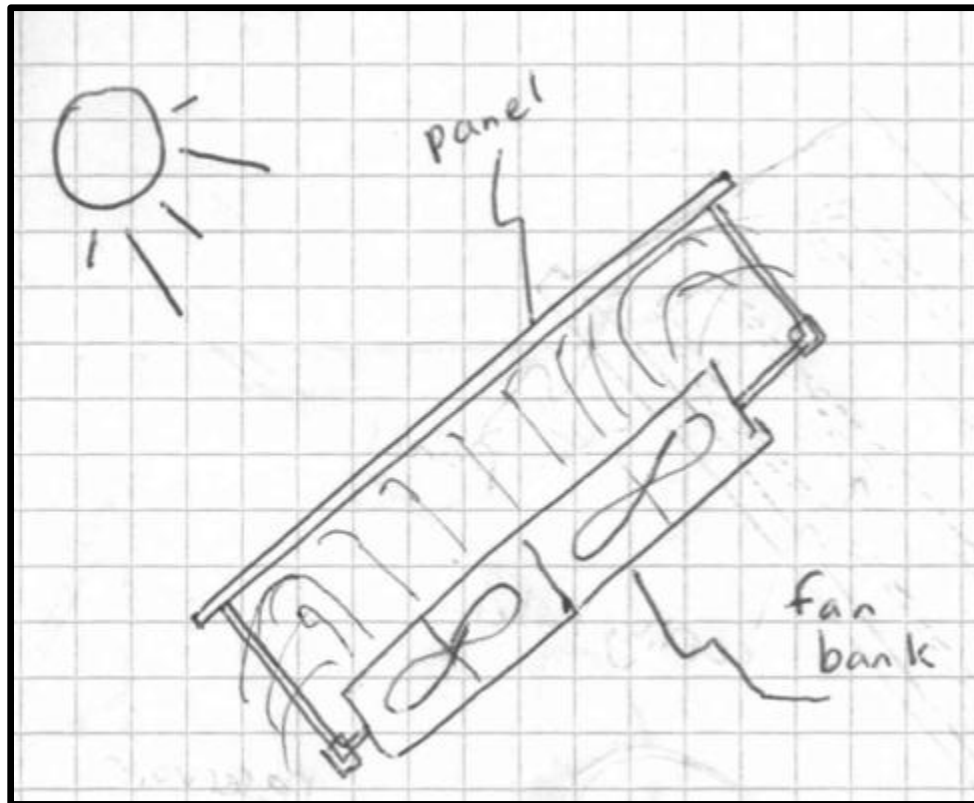


FIGURE 14 - FORCED CONVECTIVE COOLING SETUP

Fan-driven forced convection represents, in our minds, the simplest method for cooling a solar panel. Fans can cool a panel in any orientation and would not need to be physically attached to the panel to function (thereby allowing the panel to be replaced quickly and easily). No 'refills' would be required for such a system and maintenance should be minimal. Steady-state temperature should be easily adjustable by varying the fan speed. That said, fans are relatively inefficient and consume more power than other solutions. There was also concern that fans alone would not provide enough cooling to meet our specifications.

4.6.2: EVAPORATIVE COOLING FROM THE FRONT OF THE PANEL USING EITHER PRESSURIZED MISTERS OR DRIPS.

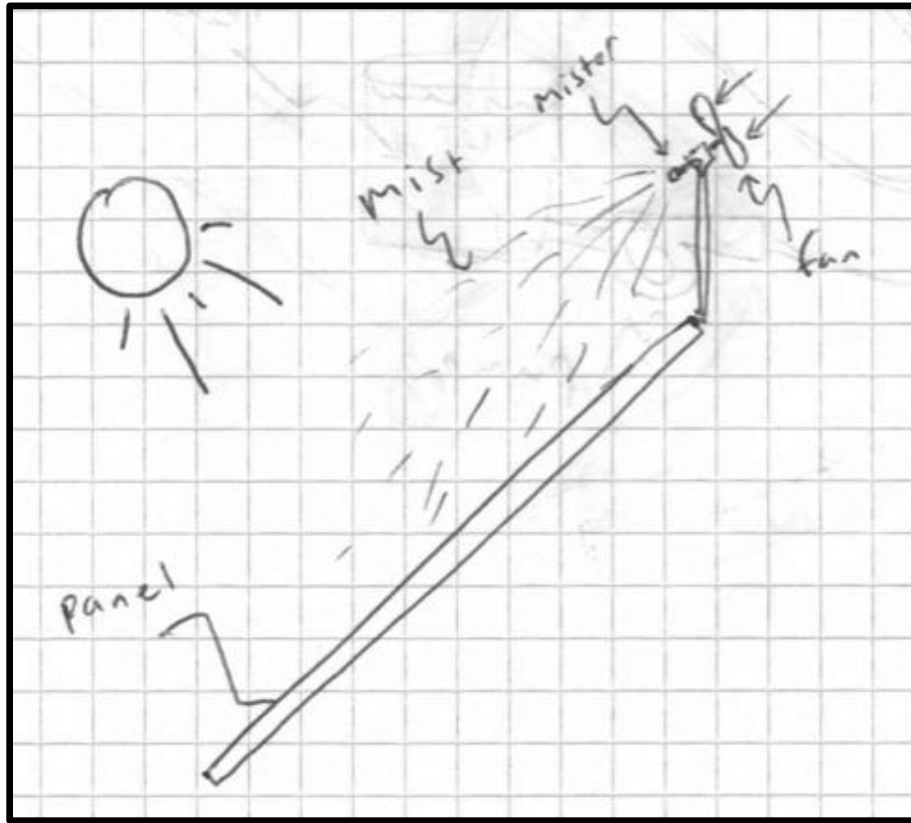


FIGURE 15 - EVAPORATIVE COOLING WITH DIRECTED MISTERS

Our research indicated that evaporative cooling is the most common method (used by consumers) for cooling household solar arrays. It is energy efficient and requires relatively few expensive components.

‘Drip’ and ‘mist’ irrigation components are readily available at local hardware stores and useful for these purposes. That said, drip irrigators require the panel to be mounted at an angle (so that the water stream flows across the panel), which may not always be the case for the PV trainer. Misters solve this problem by directing a mist of water across the panel, but they also require a considerable back-pressure (and therefore a pump).

Any evaporative system would require a refillable water reservoir (which would likely need periodic cleaning) and would expose the rig’s electrical components to moisture. An evaporative system might also present control problems, as the cooling would cease as soon as the water had evaporated.

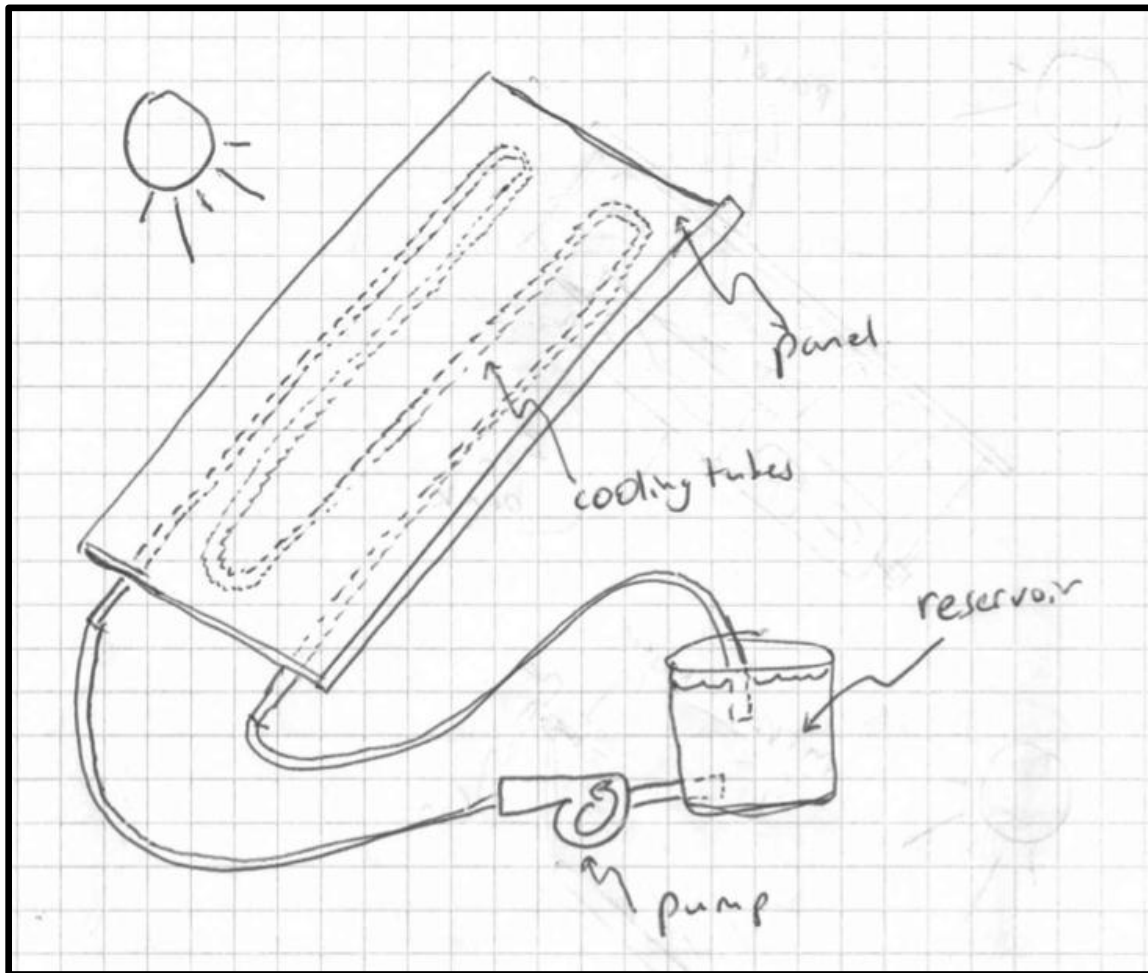
4.6.3: A CLOSED-LOOP CIRCULATING FEEDWATER SYSTEM (USING A PUMP, HEAT EXCHANGER AND/OR WATER RESERVOIR).

FIGURE 16 - RECIRCULATING WATER-COOLING SKETCH

A closed-loop water circulation system would likely produce the fastest, most even and most steady panel cooling of any of the methods investigated. It would be easy to control, energy efficient, and work at any panel angle.

Depending only how it was configured, such a system would be extremely difficult to reconfigure to fit both large and small panels. In addition, it would require either a large coolant reservoir or several separate components (such as an air-cooled heat exchanger). Leaks would also be an ongoing concern.

4.6.4: REFRIGERATION CYCLE

Like a closed-loop feed water circulation system, a heat pump/refrigeration cycle would be effective but require an array of complex (and likely expensive) components.

4.6.5: COMBINATIONS OF SYSTEMS 1-4

Several combinations of these four systems were discussed which would maximize the advantages of each (such as a 'swamp cooler' which cools air by convection through evaporation). That said, these systems are, by their nature, more complicated than their components. As a result, these were considered 'fallback solutions' that could be used if no simpler solutions could be found.

4.7: PANEL COOLING 'PROOF-OF-CONCEPT' TESTING

Of the systems listed above, fan driven forced-convection quickly emerged as the favorite because of its simplicity. Concerns lingered that fans alone would not provide enough air movement to cool a solar panel in the required time (under ten minutes). The team decided to perform proof-of-concept testing on an existing panel in the EE courtyard to determine if fan cooling was feasible; the goal was to decrease the operating temperature of the panel by roughly 20 degrees F in less than 10 minutes.

In preparation for testing, a solar panel from the Electrical Engineering courtyard was reoriented so that it would be pointing towards the sun the following afternoon. We chose a PV panel that measured 1210 x 1008 mm and was mounted at a fixed elevation angle on a wooden support structure (see figure 7, below). Unlike the other panels that were mounted in the courtyard, this panel did not have any external modifications (such as affixed copper tubing or a drip cooling setup). We conducted our tests in the courtyard at around 1:00pm on the afternoon of November 13th.



FIGURE 17 - FIXED TILT PANEL USED FOR COOLING SYSTEM TESTING

First, we determined the ambient and initial panel temperatures. Using T-type thermocouples sourced from the Mechanical Engineering Department's Thermal Sciences Lab, we determined the ambient air temperature to be roughly 75 degrees F. Next we measured the surface temperature of the panel (on both the front and rear faces). The measurements were 127.2 F for the front side and 126.8 F on the back side; this indicated a negligible temperature gradient across the panel (less than 0.4 degrees). This measurement was important because it indicated that air cooling from the back of the panel might be effective (since the panel conducts heat easily from front to back).

The first cooling system test that we conducted involved forced convective cooling using a box fan. In order to gather our test data, we mounted T-type thermocouples onto the back of the panel using tape. These thermocouples were mounted across from each other about 1.5ft from one of the sides and about 1.5ft from the top and bottom respectively. At the beginning of the test, each thermocouple indicated a temperature of roughly 127 degrees F. While this temperature is lower than the expected max temperature, the panel does not have to be at maximum temperature in order for us to evaluate the effectiveness of the cooling system; we are shooting for cooling of at least 20 degrees F, which was specified in the project proposal. We mounted a 22 x 22 inch box fan parallel to the panel, in the lower corner, about one foot away from the rear face (see figure 8, below). We set the fan to its maximum speed and began recording temperatures. The raw data from this test can be found in Appendix B.



FIGURE 18 - BOX FAN POSITIONED BEHIND PANEL, AIMED AT LOWER LEFT REAR CORNER

We recorded the thermocouple readouts at each location once per minute for ten minutes. The results are presented in Figure 19, below.

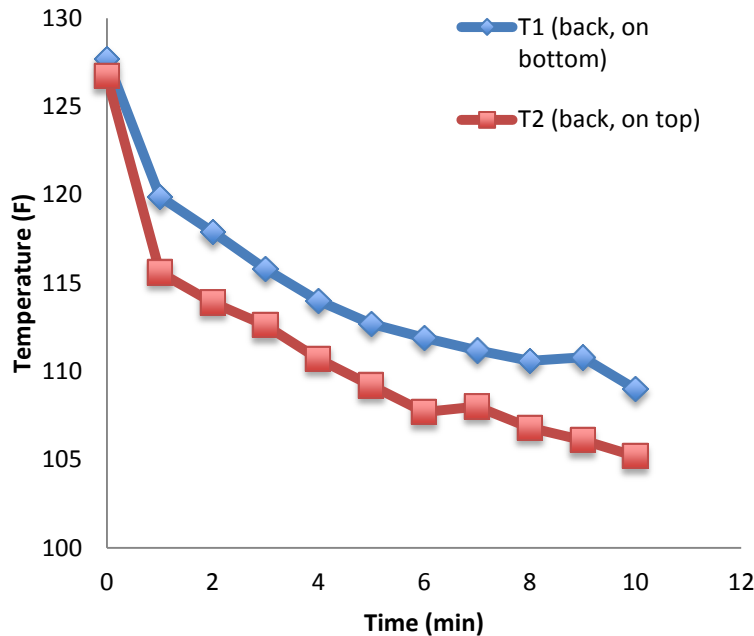


FIGURE 19 - TRANSIENT SOLAR PANEL TEMPERATURE RESPONSE; REAR MOUNTED FAN

We reached three important conclusions from these results:

- 1) The response appears roughly first order, with a large initial drop in temperature that trails off to a flatter temperature profile after a few minutes. In the first minute the temperature drops over 8 degrees, and by the fifth minute the temperature drops by about 1 degree/minute.
- 2) The total drop in panel temperature is 21.5 degrees F (sufficient to reach our target), and the final temperature of the panel after 10 minutes is 105.2 degrees F.
- 3) The temperatures at both the top and bottom of the panel dropped at roughly the same rate, despite the fan being pointed directly at the bottom thermocouple *only*. This indicates effective lateral heat conduction across the panel (which we already knew was effective at conducting heat through the short dimension). Practically, this means that cooling one region of the panel will effectively drop average panel temperatures.

We determined that forced convective cooling (using rear-mounted fans) was the most feasible panel cooling solution. We used only one fan for these tests, whereas the final design would have two or more fans to increase airflow to the back of the panel. This will both speed up

cooling and decrease the steady-state panel temperature. The decision of how many fans the rig uses will be made further out, when we have a better idea of the power consumption of the positioning system.

The second test that we conducted relied on evaporative cooling and a directed misting system. The test apparatus consisted of the same 22 x 22 inch diameter box fan equipped with a loop of $\frac{1}{4}$ " plastic drip irrigation tubing and four "fogger" irrigation attachments (see figure 18). We spliced a T-junction into the loop and ran another $\frac{1}{4}$ " diameter length of tubing to the nearest hose outlet. When we turned the fan on and supplied water to the irrigation tubing, we created a ~20 foot long plume of fine, directed mist. The purpose of the fan in this cooling system was not necessarily to blow air across the panel, but instead to direct the mist in a controlled manner from a distance. The purpose of this "directed mist" method of evaporative cooling over the more conventional 'drip method' was to eliminate the dependence on gravity; with conventional drip cooling, the hardware dripping the water must be higher than the hardware catching the water, which means that the system will not function when the panel is close to horizontal.

With the misters operating, we held the fan in front of the panel (far enough away that it didn't shade any of the panel area). We aimed the mist at the panel from a distance of about 8 feet, turned it on, and began recording temperatures. We recorded temperatures once per minute, for 6 minutes. The data from our second cooling system test are presented in Figure 20, below.

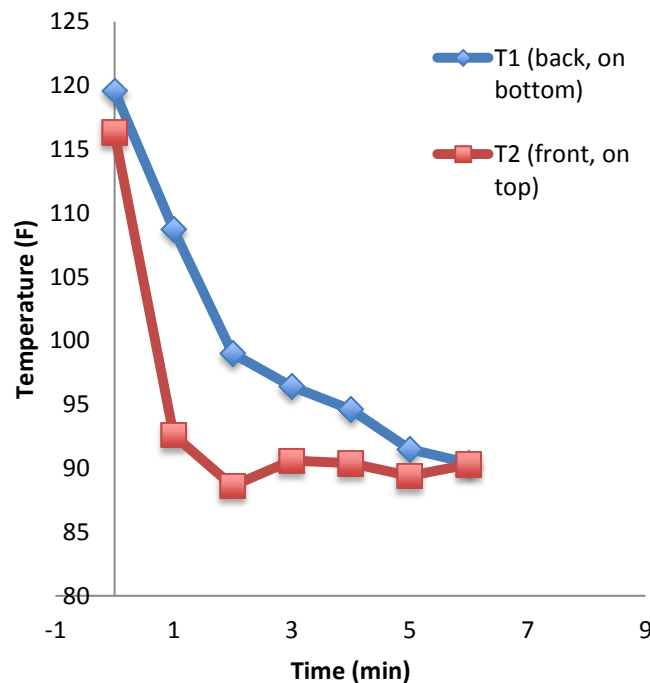


FIGURE 20 – TRANSIENT SOLAR PANEL TEMPERATURE RESPONSE; FRONT MOUNTED FAN (WITH MISTERS)

We made three key observations from the results of our second test:

- 1) Again, the cooling panel demonstrated first-order behavior. In the first minute, the panel dropped by 25 degrees F, and it reached steady state at 6 minutes.
- 2) The final steady state temperature was measured at 90 degrees F, with a 0.1 degree difference in temperature from the back to the front of the panel. This is 37 F cooler than the initial temperature of the panel.
- 3) After 3 minutes of misting, the panel was covered with an even coat of water to the point that it began pooling into rivulets and running down the panel.

After performing the second test, we were impressed by the effectiveness of the mister. It was clear that directing a fine mist at the panel was an effective way to lower the panel temperature. However, as mentioned in the ideation section above, we also saw several disadvantages with evaporative cooling that we didn't see with air-only cooling. Firstly, there was a large amount of mist not hitting the panel, or hitting the panel and bouncing off. If we are going to be working with high voltages, electronic controls, and even wood (for the student work surface), we want to minimize the amount of water sprayed across the rig. Too much water exposure could prematurely age or even short out various components. Secondly, running the misters requires a supply of pressurized water. Using an on-board pump and a water reservoir would make the rig heavy, and refilling the water before each lab would be an inconvenience to instructors and students alike. Alternatively, we could provide a hose so that the rig could be tied to a nearby water outlet, but this solution limits the mobility of the rig greatly.

An evaluation of our cooling system ideas appears on the following page in the form of a Pugh Matrix.

4.8 CONCEPT EVALUATION: COOLING SUBSYSTEM

TABLE 3 - COOLING SUBSYSTEM PUGH MATRIX

Criteria	Rear Mount Fan Bank	Bottom Mount Fan Bank	Swamp Cooler	Heat Pump	Misters (evaporative)	Drippers (evaporative)	Rear-Mounted Cooling Plate	Rear-Mount Fans (with Radiator)
Cost	Datum	S	-	-	+	+	-	-
Complexity (custom parts)	Datum	S	-	-	S	S	-	-
Number of Components	Datum	S	-	-	-	S	-	-
Ease of Temp Adjustment	Datum	S	S	S	-	-	+	S
Weight	Datum	S	S	-	+	+	-	-
Flexibility (many panels)	Datum	S	S	S	S	-	-	S
Ease of Use (refills, etc.)	Datum	S	-	S	-	-	S	-
Uniform Cooling	Datum	-	+	+	S	-	S	+
Response Time	Datum	S	+	S	+	+	S	+
Effective At Many Angles	Datum	S	S	S	S	-	+	S
Energy Use	Datum	S	+	-	S	+	+	S
Sum of +'s	Datum	0	3	1	3	4	3	2
Sum of -'s	Datum	1	4	5	3	5	5	5
Sum of S's	Datum	10	4	5	5	2	3	4

Given that we were able to cool the panel by more than 20° F with forced convection and only one fan, and since we have the space on our rig to accommodate at least 2 fans, we decided to use fans to cool the solar panel on our rig. This will provide a clean, effective solution to the cooling problem, and it has several advantages, mentioned above. For one, because the fans are mounted at an offset from the panel rather than connected directly to the back of it, one could swap out panels easily without worrying about disconnecting cooling tubes, irrigation lines, or anything else. Also, since the flow of air through the fans is determined by the fan speed, one can easily change the amount of convective cooling on the panel by simply turning the fans up or down.

5. FINAL DESIGN DESCRIPTION:



FIGURE 21 - FINISHED MOBILE SOLAR PV TRAINER ('IN USE' CONFIGURATION)

5.1: DESIGN SUMMARY

This section contains a detailed description of the finished Solar PV Trainer design. Full dual axis tracking is accomplished using the ‘radio telescope’ approach (see the Section 4.3 for a more thorough explanation). The rig has a range of motion from 0 to 90 degrees in the ‘main pivot’ axis and -20 to 20 degrees in the ‘secondary’ pivot axis. Panel cooling is accomplished using two rear-mounted box fans; digital thermometers are used to determine the panel temperature. The finished system weighs roughly 200 pounds (not including ancillary electrical components such as the battery, inverters or charge controller). When not in use, the rig collapses into a bounding box measuring 36 inches wide, 78.25 inches tall and 58 inches long. These dimensions give it a large, stable base while still allowing it to fit cleanly through a standard 80 inch by 32 inch doorway. The rig remains stable on inclines up to 25 degrees during transport. The panel rack can accommodate panels up to 1 meter in width and 2 meters in length. Additionally, the rack can accommodate panels with a maximum depth of 55 mm and minimum depth of 30 mm. If a PV panel needs to be replaced, the brackets attached to the panel rack allow a new one to be substituted in less than 10 minutes.

Structurally, the Solar PV Trainer design consists of seven distinct subsystems: the frame, the student work surface, the inverter mount, the traveler assembly, the shafts and bearings, the panel mounting rails and the electrical box.

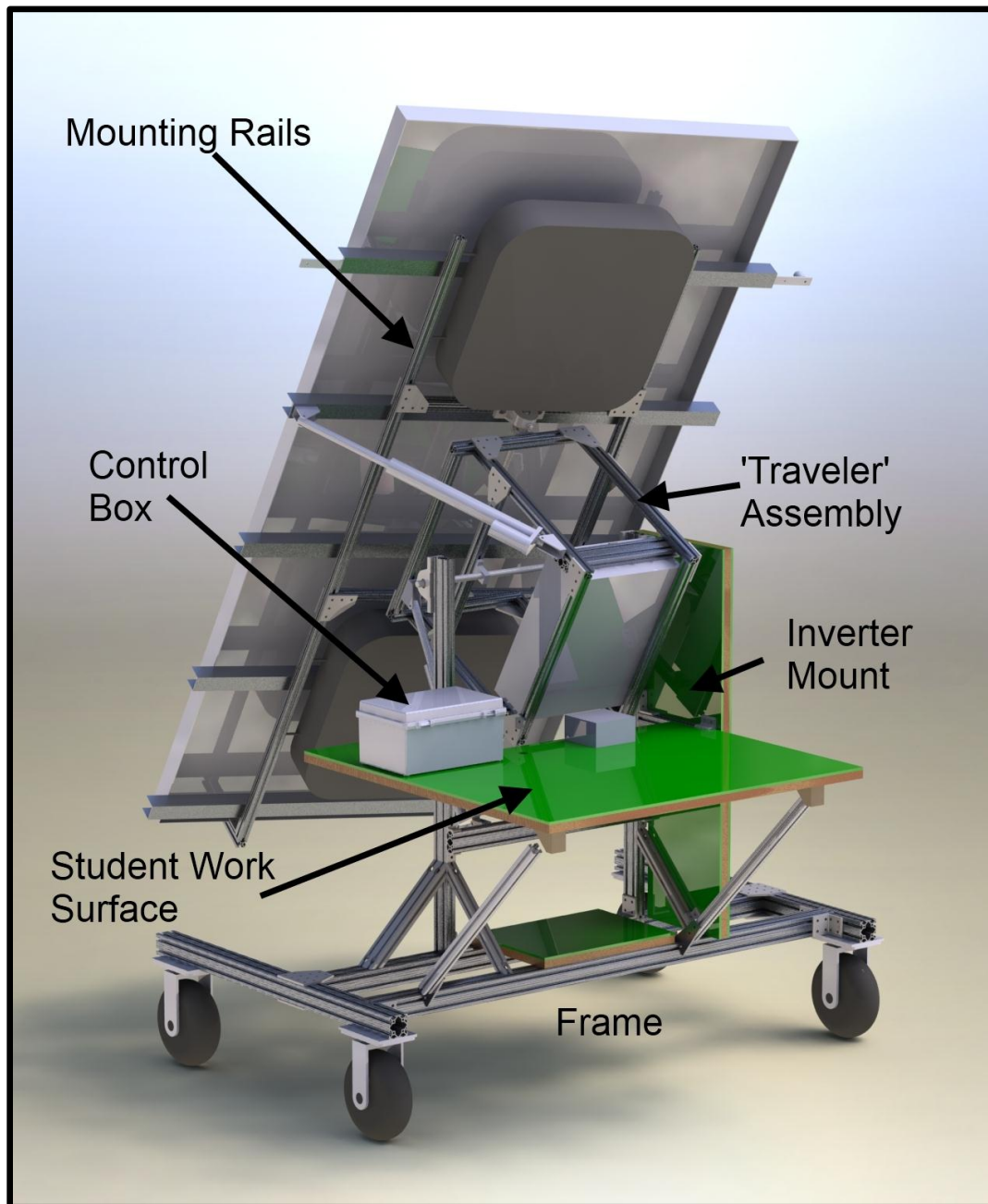


FIGURE 22- MOBILE SOLAR PV TRAINER STRUCTURAL SUBSYSTEMS

Other subsystems include the drive system, the electrical system and the panel cooling system. Each of these subsystems will be discussed in detail in the following sections. Dimensioned drawings, assembly drawings, data sheets and a bill of materials can be found at the end of this report in Appendix D.

5.2: FRAME

The trainer's frame is constructed almost entirely from 10-Series '8020' Aluminum T-slotted extrusion. The extrusion components are attached to one another using purpose-made hex-head fasteners and 8020 gusset plates.

Special care needs to be taken while assembling the rig to ensure that a sufficient number of T-slot nuts are slid into each T-slot channel before the brackets are tightened down. If T-slot nuts (or carriage bolts) are overlooked, portions of the rig must be disassembled (often at inconvenient times) to add the necessary fasteners. Consult the diagrams carefully before assembly.

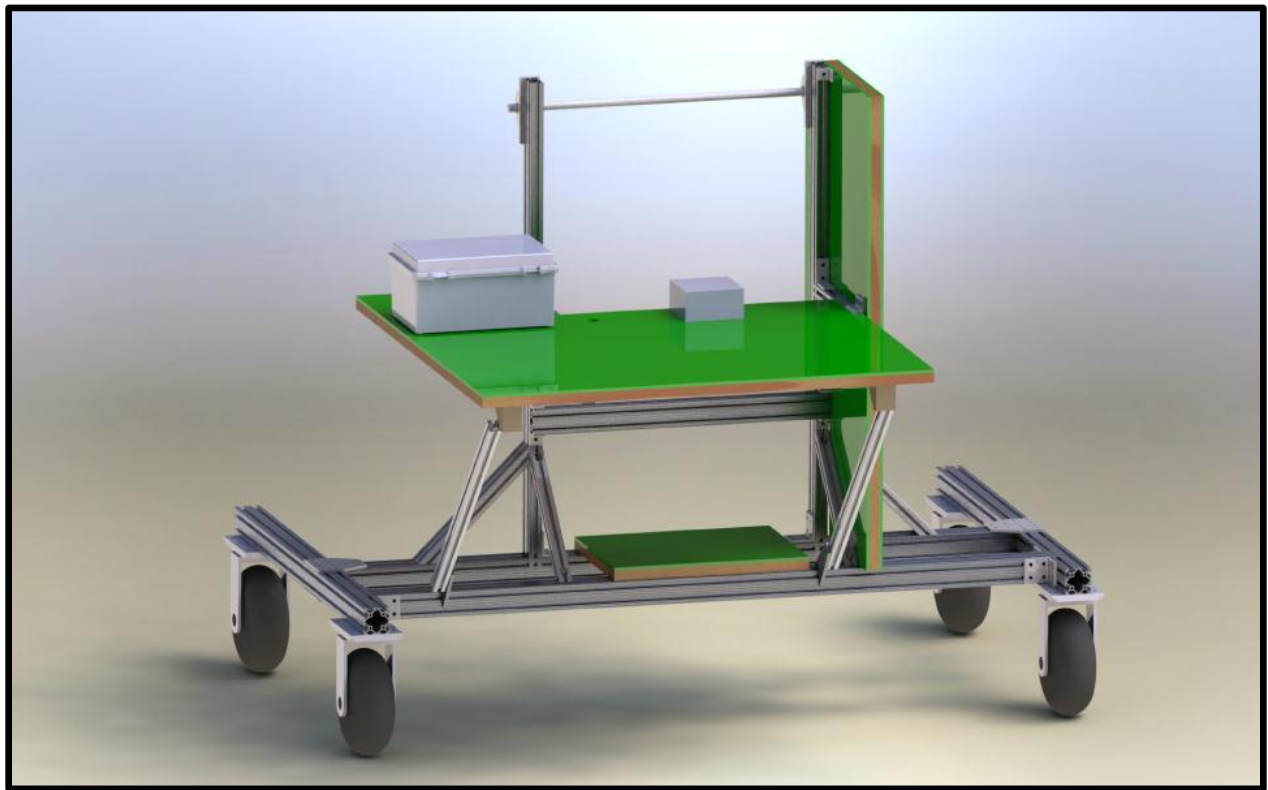


FIGURE 23 - MOBILE SOLAR PV TRAINER FRAME

The four large, 8 inch diameter, spinning rubber casters are attached to the frame using dedicated mounting plates. These plates are attached to the T-slot frame using four low-profile T-slot fasteners and bolted to the wheel mounting plates using three (or four) $\frac{1}{4}$ inch diameter carriage bolts (with nuts and washers). These large wheels allow the rig to navigate rough terrain (e.g. concrete, grass, dirt), as well as vertical ledges up to 2 inches tall. All four wheels swivel to facilitate hallway navigation and lock to secure the rig in place during experiments

Nearly all of the non-standard T-slot components (such as bearing blocks and shaft mounts) are attached to the frame using $\frac{1}{4}$ inch diameter stainless steel carriage bolts. These bolts slide cleanly and tightly into the T-slot channels and hold their position firmly when tightened in place (see Figure 24 to the right). All references to carriage bolts for the remainder of this report will refer to these $\frac{1}{4}$ inch diameter bolts.

We elected to use T-slot aluminum extrusion for several reasons. First, the T-slot system allows frame components to be joined non-permanently; this permits the builder to modify the frame during (or after) assembly, if necessary. In addition, it simplifies manufacturing and fabrication; T-slot brackets guarantee solid and square right angles and no welder or other special tools are required for assembly. Since no welding is required, there is no risk of heat damaging the frame, creating unsightly weld defects or burning through the walls of an extruded part. T-slot also gives the rig a clean, polished look without the need for professional painting or powder coating. Also, aluminum is corrosion-resistant and will not be damaged by prolonged outdoor storage. Finally, aluminum T-slot is extremely easy to work with. Consequently, if the Cal Poly EE department decides to fabricate additional PV Solar Trainers, it should be relatively easy for them to do so.

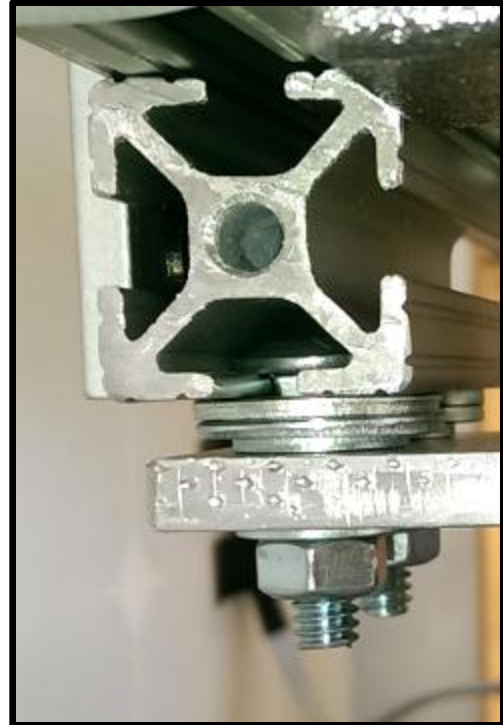


FIGURE 24 - A BRACKET SECURED TO T-SLOT EXTRUSION USING CARRIAGE BOLTS

5.3: TRAVELER & COUNTERWEIGHT

The ‘traveler’ assembly rotates with the secondary axis and serves as a mounting point for the bearings and linear actuators (actuators are discussed in detail in Section 5.8). The traveler also houses a counterweight which offsets the imbalance (about the primary axis) caused by the panel mounting structure.

The counterweight itself weighs roughly 45 pounds and consists of a painted and sealed solid steel block that was donated to the project by the Cal Poly IME department. This block was cleaned with a wire brush, primed, painted, and finally rubber-coated using Plasti-Dip aerosol spray. Four 1/4 inch diameter countersunk clearance holes (two each on both the top and bottom faces of the block) were drilled into the counterweight. These holes mate with four 1/4 inch diameter carriage bolts that are slid into the T-slot supports. The weight is then held in place by the clamping pressure of the two (top and bottom) T-slot supports.

In the event that no steel weights can be obtained from the IME department, a brick counterweight could be substituted (as demonstrated in Figure 25). Cost and weathering concerns drove the selection of brick as a backup counterweight material.

All other ancillary parts (including the bearing blocks, shaft mounts, and actuator brackets) are attached to the frame using 1/4 inch diameter carriage bolts (with the complementary washers and nuts).

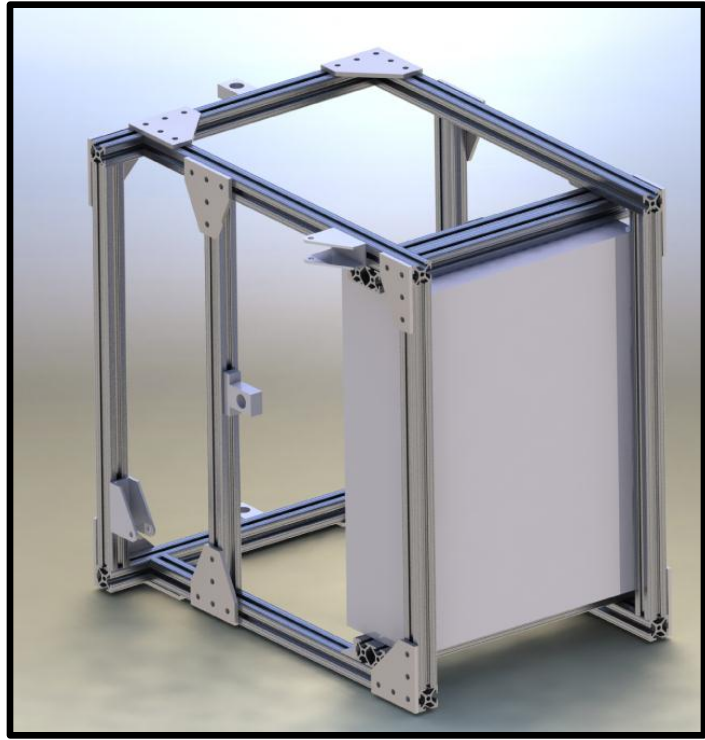


FIGURE 25- TRAVELER & COUNTERWEIGHT ASSEMBLY

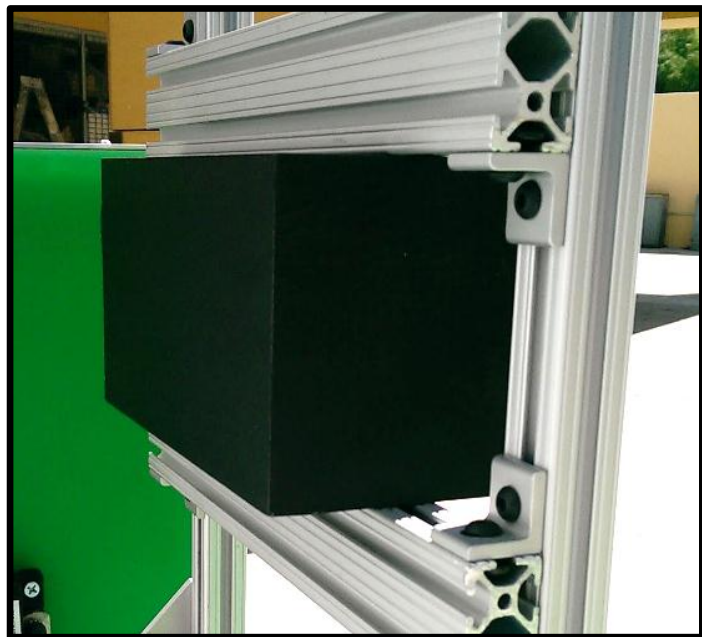


FIGURE 26 - SOLID STEEL COUNTERWEIGHT

5.4: STUDENT WORK SURFACE

A 5 ft² student work surface is provided which sits below the electrical box and reaches around behind the rig. The work surface is made from $\frac{3}{4}$ inch thick plywood topped with a $\frac{1}{4}$ inch thick green acrylic sheet. The wood base is spray-painted black on all sides to help preserve it from the elements.



FIGURE 27 - UNFOLDED STUDENT WORK SURFACE

The green acrylic sheet used for both the work surface and inverter mount was leftover from a previous Mechanical Engineering senior project; it was donated to the PV Solar Trainer team by the Cal Poly ME department. The acrylic is attached to the wood base using lengths of purpose-made double sided ‘acrylic tape’ that runs around the outside edges of each piece. This tape interacts with the translucent acrylic to create an unsightly black shadow on the acrylic surface. Consequently, no tape was placed across the center of the acrylic panels. The wooden base of the work surface was attached to the frame using $\frac{3}{4}$ inch long wood screws and aluminum ‘8020’ support brackets.

All edges of the work surface are surrounded by 1 inch tall extruded aluminum “L” bracket. This “L” bracket was primarily hand cut and is attached to the surface using a combination of double sided acrylic tape and $\frac{1}{2}$ inch long wood screws. These screws run through the extrusion and into the side of the plywood laminate.

In order to reduce the trainer's footprint during transport, a 30 inch long piano hinge is attached between the fixed and folding sections of the work surface. This hinge is attached such that a slight air gap exists between the two parts. This gap allows the joint to hyperextend several degrees beyond horizontal (which in turn allows the folding support legs to swing down easily). When stowed, a small magnetic clip secures the table in place.

The work surface is supported by two folding legs made from 22 inch lengths of T-slot extrusion. These pieces of extrusion are mounted on hinges that attach to the bottom of the work surface via 45 degree angle blocks. These blocks, made from 2 x 4 pine scraps donated to the project by the Cal Poly ME department, allow the legs to fold upwards and alongside the bottom of the table for transport. The ends of the legs are capped by small cut aluminum 'feet' which latch solidly onto one corner of the T-slot frame. Scraps of 1 inch tall aluminum "L" extrusion were used to protect the finish of the large 2 x 2 T-slot frame member.



FIGURE 28 - STUDENT WORKSURFACE SUPPORT LEGS/FEET

An emergency safety switch is mounted on the work surface next to the electrical box. This safety switch cuts power to the motors and illuminates a warning light on the control panel.

5.5: INVERTER & BATTERY MOUNTS

The inverters and other weather-resistant electrical test equipment are mounted on a $\frac{3}{4}$ inch thick plywood sheet that is sandwiched between two layers of $\frac{1}{4}$ inch acrylic; this assembly is screwed to the frame opposite the electrical box. This inverter mounting plate is surrounded by 1.5 inch tall extruded aluminum “L” bracket (which accounts for the added thickness of the double-sided acrylic sandwich). Since this extrusion is thicker than the 1 inch tall sections used elsewhere, it needed to be cut using a horizontal band saw. The extrusion is attached to the inverter mount in exactly the same manner as elsewhere (acrylic tape and $\frac{1}{2}$ inch long wood screws). The electrical components themselves are attached to the inverter mount surface using $\frac{3}{4}$ inch long wood screws.

The battery is mounted on a small plywood/acrylic platform located on the frame below the student work surface. It is housed in a weather resistant plastic enclosure which shelters the contacts and protects the battery from water damage.

Wires from all of these devices are carefully run along the frame to their respective destinations; these wires are secured in place using wire mounting clips.



FIGURE 29 - INVERTER MOUNTING PLATE WITH INVERTERS AND CHARGE CONTROLLERS

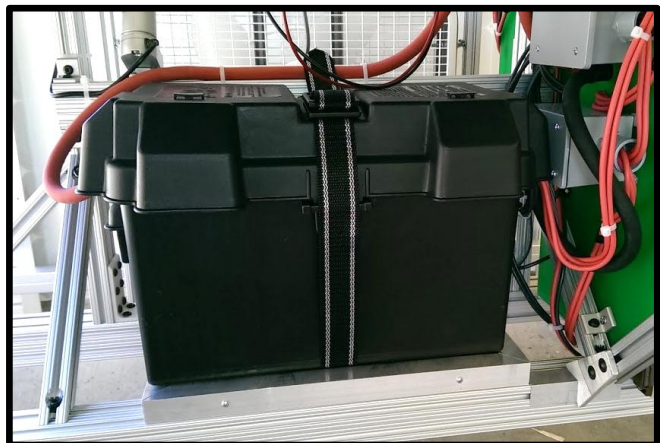


FIGURE 30 - BATTERY SHELF (AND ENCLOSURE)

5.6: SHAFTS & BEARINGS

The shafts and bearings specified for both axes were chosen to reduce cost and to simplify fabrication.

Both shafts are 24 inch long, 5/8 inch diameter nitride-coated steel shafts. These shafts support the structure while minimizing deflection (and therefore panel misalignment). A standard length of 24 inches was chosen so that the shafts would not need to be cut or machined. The nitride coating protects the shaft from weathering; a coated-steel shaft was chosen over other alternatives, such as aluminum or stainless steel shafts, to reduce costs. A 5/8 inch diameter shaft was chosen because it allows us to use 16mm (metric) shaft mounting blocks (5/8 inch = 15.875 mm) if necessary. Please reference Section 9.1 for a new shaft sizing recommendation.



FIGURE 31 - PRIMARY AXIS CLAMPING SHAFT MOUNT (WITH FASTENERS AND SPACERS)

The shafts are mounted to the frame using clamping shaft mounts (as shown in Figure 31 above). Since these shaft mounts are relatively narrow, stainless steel mending plates were used to provide a flat surface wide enough to support the mounts. 1/4 inch carriage bolts were used to clamp both the spacers and mounts to the T-slot frame. Both the shaft mounts and the mending plates needed to have their through holes widened to accommodate the 1/4 inch carriage bolts.

The bearing blocks are nickel coated cast-iron and are attached to the frame using ¼ inch carriage bolts. A double stack of fender washers proved necessary to support the nuts as they were tightened.

The bearings themselves are deep grooved, double-sealed ball bearings which offer a high degree of protection from dirt or water damage. The shaft is secured to the bearings using set screws. These bearings can handle up to thirty degrees of angular misalignment, which is far more than the maximum angular deflection expected from the shafts. These bearings are rated to support upwards of 700 pounds under dynamic loading which is almost ten times what we expect them to encounter; these oversized bearings were chosen because they were the least expensive weather resistant option that could accommodate a shaft of this diameter.

When the panel has been positioned in its 'travel configuration,' the secondary axis shaft will be vertical with respect to the ground. Consequently, the loads on this shaft will be purely axial and the bearings will need to support a considerable thrust load. This load will consist of the weight of the panel itself, the weight of the panel mounting rails and the weight of the two fans comprising the cooling system (a load possibly in excess of 50 pounds). While these particular bearings have not been specifically rated to handle thrust loads, a local bearing supplier assured us that bearings of this type safely support loads up to 50% of their radial load capacity. Consequently, each of the bearings supporting this axis should safely hold at least 350 pounds of axial load. Two solid, oil-impregnated bronze (SAE 841) thrust bushings (and one shaft clamp) were used to brace the bearing against the shaft mount to support these axial loads. These spacers keep the set screws from needing to support any axial loads.



FIGURE 32- SECONDARY AXIS 'THRUST BUSHING' AND SHAFT CLAMP PLACEMENT

5.7: PANEL MOUNTING STRUCTURE

The solar panel ‘rail assembly’ supports the PV solar panel, sensors and cooling fans. The panel itself is clamped to the mounting structure using proprietary aluminum extrusion (which is similar in design to Unistrut). Universal mounting clips clamp the panel in place against these mounting struts. A small T-slot ledge is provided to support the panel while it is being attached.

The box fans are secured in place using four ¼ inch diameter carriage bolts. These carriage bolts feed through holes drilled in the metal fan housings and slide into the T-slot aluminum extrusion. Wing nuts and washers are used to clamp these fan support bolts semi-permanently against the T-slot rails.

The mounting struts themselves are bolted to the rails using ¼ inch carriage bolts. The struts are offset from the rail using two washers and ½ inch tall nylon spacers. These spacers lift the mounting struts sufficiently high above the rails to clear the plastic housings that cover the box fans.

The irradiance sensor is affixed to the topmost piece of mounting strut using an 8 inch long galvanized steel mending plate. This plate is bolted to the strut using ¼ inch diameter bolts; one half of this plate covered with ten layers of Plasti-dip rubberized coating (to reduce the risk of head injury).

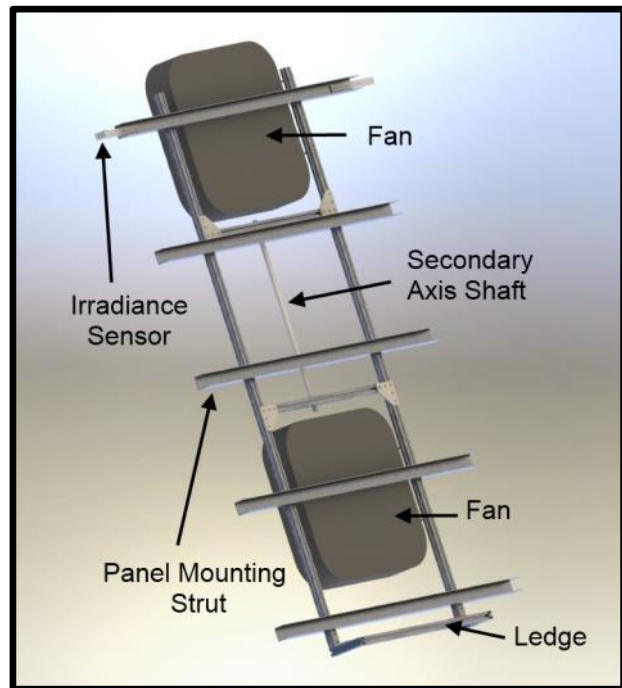


FIGURE 33 - PANEL MOUNTING STRUCTURE



FIGURE 35 - ATTACHED SOLAR PANEL MOUNTING STRUT (WITH NYLON SPACER)



FIGURE 34 - FAN MOUNTING BOLT

5.8: DRIVE SYSTEM

Both the elevation and azimuth axes are driven using one of the same model of 12 volt linear actuators. These actuators have a travel length of roughly 12 inches and move at a speed of 18 inches per minute. As a result, each axis traverses its full range in roughly 40 seconds. Like most linear actuators, these are self-locking and hold their position when no power is being applied. These actuators come equipped with self-contained limit switches which prevent them from being extended beyond their usable range. They are designed for outdoor use and possess an IP65 weather rating.

Linear actuators were chosen over gears or belts to reduce system complexity and cost. Unlike gear or belt systems, these actuators are fully self-contained and will not rust or corrode in the elements. They also reduce the number of hazardous ‘pinch points’ on the trainer and eliminate all exposed spinning mechanical hazards; they therefore improve the system’s safety. In addition, since these actuators are self-locking, no ‘brake system’ needs to be included to hold the panel in a fixed position while measurements are being gathered. Standard mounting brackets can be purchased to interface with these actuators; this eliminates the need for any custom ‘motor mounting brackets.’ Finally, these actuators are also considerably less expensive than a comparable motor and gear or pulley system.

These actuators will *not* allow the system to be adjustable via ‘hand cranks’ or other purely mechanical methods as was originally planned. Consequently, backup controls are included (in the form of manual control switches) so that users may readily adjust the panel position from the control panel.



FIGURE 36 - LINEAR ACTUATOR USED FOR PANEL ARTICULATION

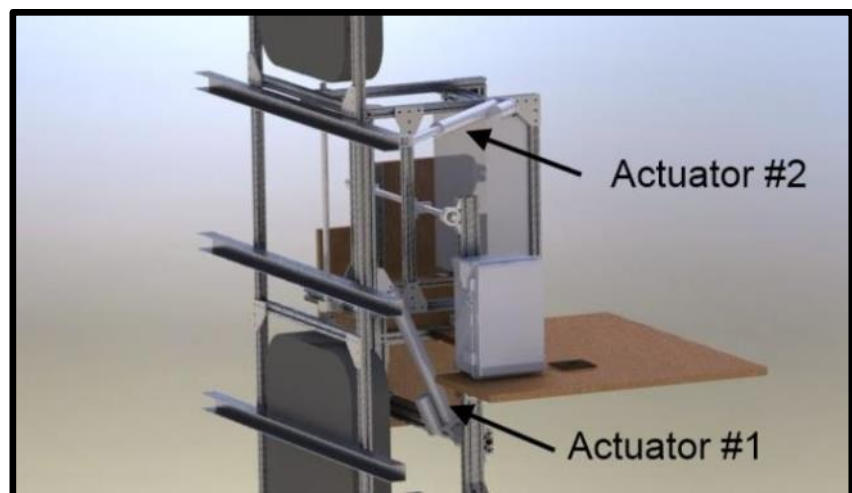


FIGURE 37 - LOCATION OF ACTUATORS

5.9: COOLING SYSTEM

Panel cooling is provided by two large, rear-mounted AC box fans. These fans receive power from the smaller of the trainer's AC inverters (Inverter #1). The panel's temperature is adjusted by manually altering the speeds of these fans. These fans are capable of reducing the temperature of the panel by around 20 °F in less than 10 minutes. For small panels only one fan should be used (to reduce power consumption). The fans are mounted using four ¼ in diameter machine screws which slide into the aluminum T-slot and bolt to the metal fan case (reference Section 5.7 for details). These particular fans were chosen because they are easily available, are sufficient to cool the panel and are extremely inexpensive.



FIGURE 38 - BOX FAN USED FOR PANEL COOLING

5.10: ELECTRICAL BOX & WIRING

Most of the weather-sensitive electronic components including LCD screen, keypad, switches, microcontroller, and motor drivers are contained within an 'electrical box' attached to the end of the student work surface. This 12.0 x 8.0 x 5.5 inch UV-stabilized, gray plastic outdoor junction box features a hinged door and pre-threaded mounting holes (both interior and exterior). It is fully weather proof, corrosion resistant and rated to withstand wash-downs from high powered water nozzles. This particular junction box was chosen because of its simple mounting system (pre-threaded holes) and relatively low cost (when compared to comparable powder-coated steel junction boxes).



FIGURE 39 – TRAINER ELECTRICAL AND JUNCTION BOXES

Two smaller outdoor junction boxes are provided to house the power outlets, fan controls and banana jack receptacles. The first (which houses the fan controls and outlets) is located adjacent to the electrical box. The second (which houses banana jack ports) is attached to the work surface near the center of the rig (below the traveler assembly). Both junction boxes are attached to the work surface using $\frac{3}{4}$ inch long wood screws



FIGURE 40 – BANANA JACK JUNCTION BOX



FIGURE 41 – FAN CONTROL JUNCTION BOX

All power for the rig's control and sensor systems is provided by the trainer's onboard DC battery. This battery is mounted below the rig on a small shelf (see Section 5.5).

All rig wiring is either attached to the work surface using plastic clips, zip-tied strategically to the frame or concealed within the aluminum extrusion using 'T-slot wire covers'. The heavier gauge wiring is run through the inverter mount and affixed to the bottom of the student work surface (wherever possible). All wire connections are contained within junction boxes.

All metallic components (including the larger frame members, metal junction boxes and commercial electrical components) are grounded together to form a chassis ground. Grounding wires are run from the frame to mounted components and copper grounding straps create a conductive path between the frame, traveler and rails.

5.11: CONTROL PANEL & USER INTERFACE

We recognize a huge opportunity to engage students in the lab with an effective design of the control panel: this is where the students will interact with the trainer: read data, control the motorized positioning systems, control panel cooling, take measurements, etc. We aimed to design the control panel to be as elegant and intuitive as possible.



FIGURE 42 - PV SOLAR TRAINER IN USE

The control panel itself consists of an opaque, black 0.236 inch thick acrylic ‘false wall’ façade which bridges the interior of the electrical box roughly one inch behind the front of the door. This façade is secured in place using four 80mm long M5 mounting screws (and nuts) which engage with the four pre-threaded mounting holes included in the back of the electrical box. These four mounting screws need to be loosened in order to access the microcontroller and panel wiring. The front of this panel is laser-engraved with appropriate labels to make the display more intuitive. User interface elements, such as an LCD screen, are attached to the panel using double-sided acrylic tape. A laser cut acrylic control panel was selected because such a panel is relatively inexpensive, easy to manufacture and looks professional. Relatively thick acrylic (0.236 inches) was chosen to ensure that the panel would not flex while in use.

User interface elements on the control panel include:

- 1) *LCD screen*: This displays relevant information, notably: angular positions, temperatures in the digital thermometer array, and solar irradiance. The high contrast screen is legible in normal daylight and updates every 0.5 seconds.
- 2) *Keypad*: The keypad enables students to interact with the microcontroller. For example, the student may cycle through different screens using the function keys.
- 3) *Power switches*: These illuminated power switches allow the student to supply power to the microcontroller and motor driver chips respectively. Two LEDs mounted on the panel also display the current status of the microcontroller (whether or not it is powered) and the current status of the emergency stop switch.
- 4) *Positioning system mode*: This switch has two states. The first, “keypad” routes power to the motor driver chips and microcontroller. In this mode, the position of the actuators is controlled by the microcontroller using a Pulse Width Modulated (PWM) signal. The second mode, “manual,” isolates the motor drivers from the actuators and engages the manual switch array. This switch array routes 12 Volts directly from the battery to the actuators (completely isolating the microcontroller from the system).
- 5) *Actuator select*: This switch determines which actuator is being controlled whenever the manual positioning switches are active.
- 6) *Actuator direction*: This switch has two states. The first extends the selected actuator and the second retracts the selected actuator. This switch will only be used when the positioning system has been set to manual.

5.12: USER INTERFACE OPERATING INSTRUCTIONS

To start the system: flip the two green power switches on the top right of the display. The first switch supplies power to the microcontroller and second supplies power to the motor drivers. If the system is operating correctly, internal LEDs inside both switches should illuminate. As the microcontroller starts, a splash screen for the device will display. After a moment, the splash screen will fade and the LCD will display the current primary and secondary axis angles (as measured by the encoders).

From this screen, the desired set point of the actuators can be controlled using the up, down, left and right arrows on the keypad. Pressing these arrows will increment or decrement the target position of each axis by 1 degree respectively. To enable actuator power and to begin moving to these target angles, press the start button on the keypad. To disable actuator power, press the stop button on the keypad at any time. If you wish to return to this screen at a later time, simply press the F1 button on the keypad.

To control the angles of the solar panel in azimuth-elevation “world coordinates,” press the F2 function key on the keypad. This screen functions in the same manner as the previous one, except that angles can now be specified in terms of azimuth and elevation “world coordinates.” You will likely notice that an adjustment to the azimuth or elevation set point will cause both the primary and secondary axes to adjust accordingly. The microprocessor resolves the desired movement in azimuth or elevation into movements about both axes appropriately. For the complete coordinate transformation equations used to accomplish this, refer to Appendix C.

To display current temperatures from the DS18B20 temperature sensors, press the F3 button on the keypad. Additionally, the solar irradiance is displayed at the bottom of this screen.

To manually input target position values for the encoders (in both coordinate systems), press the F4 button on the keypad. Target position values can be input using the numbers on the keypad. To confirm your selection when a number has been input, press F1. If you wish to return without making changes, press F1 (with no numbers input). To backspace, press the F2 button. To change the sign of your input, press the F3 button. To cycle between which angle or coordinate system to use, press F4. Once a number has been input, you will be returned to the appropriate actuator status screen. Verify your target positions and press the start button to begin moving.

5.13: SENSORS

The trainer is equipped with a variety of sensors to provide data for lab experiments.

Temperature Sensors-

Our proof-of-concept cooling tests indicated that surface temperatures did not vary widely between regions of the panel (less than ± 2.5 degrees) - either front to back or across the face. Therefore, we concluded that several individual 'point' temperature sensors would be sufficient to generate an accurate average panel temperature. Digital thermometers were chosen because they are accurate, inexpensive, and can be powered by the same wire that transmits the temperature data (reducing the complexity of the wiring required). In addition, with the proper circuitry setup, digital thermometers can be easily read by a microcontroller using a single data line, saving valuable IO pins. A bank of 4 digital thermometers was used to measure changes in panel temperature. These sensors are strategically attached to the rear of the PV panel (at easy to reach locations) using tape and thermal grease. Standard phone cable is run from these sensors to the microcontroller and shrink tubing was used to protect the thermometer leads.

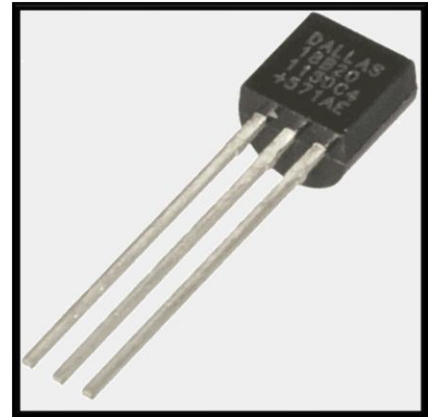


FIGURE 43 – DIGITAL THERMOMETER

The reading from each digital thermometer is averaged with the others and the result is displayed at the control panel. We selected DS18B20 digital thermometer, pictured above, because of its accuracy, response time, and simple communication protocol. The full data sheet for this type of sensor, along with data sheets for the rest of our electronics, can be found in appendix E. These temperature sensors can record temperature accurate to better than ± 1 °F (satisfying our design requirement).

Rotary Position Sensors-

The offset angle of each axis is measured using an AEAT-6010 absolute encoder attached to each shaft. The measurements generated by these encoders will be displayed on the control panel (in degrees). Using these encoders, the system is accurate to better than ± 1.0 degree for both axes. Encoders were chosen over potentiometers because they provide more repeatable and accurate angular displacement measurements. *Absolute* encoders were chosen specifically to ensure that the trainer will not 'forget' its angular position whenever power is disconnected. This eliminates the need for a tedious calibration step each time the panel loses power.



FIGURE 44 – ABSOLUTE ENCODER

Custom designed brackets and interface hardware was developed to attach the encoders to the frame and shafts. Please reference Section 7.1 for a more thorough description of this mounting hardware.

Irradiance Sensor-

Our sponsor, Professor Dolan, has agreed to provide us with an Apogee irradiance sensor. This sensor will be used to measure how much potential energy (irradiance) is available from the sun on a given day. This sensor generates a voltage proportional to the solar irradiance which is read by the analog to digital converter on the microcontroller. The irradiance reading will be displayed in real time on the control panel's LCD screen.



FIGURE 45 – IRRADIANCE SENSOR

5.14: MICROCONTROLLER

Hardware -

We have chosen to use an AVR microcontroller donated by Professor Ridgley from the ME 405 (mechatronics) lab. This board includes imbedded motor drivers and comes with a pre-written software library (courtesy of Professor Ridgely). The microprocessor used is a ATmega 1281 operating at 16 MHz. This clock speed is useful for ensuring that the motor control and LCD all update quickly enough that the user does not notice any update lag. This microprocessor is equipped with an analog to digital converter and has PWM capabilities (for use with motor control). The board can also communicate with external devices using the SPI communication protocol functionally built into the chip.

Software -

We have chosen to program in C++. Higher level languages like Java would be unable to interface with the hardware registers necessary for this project. Other lower level languages such as C and assembly were considered for their efficiency, but the convenience of object oriented programming and the availability of pre-existing code libraries from ME 405 made C++ the most attractive option.

In order to ensure that the rig operates as expected, we have elected to use FreeRTOS-driven multitasking. Some systems (such as the LCD display) demand more processing time than others; if multi-tasking was not employed, processor hungry tasks might impede the operations of others. FreeRTOS works by allowing higher priority tasks to interrupt tasks of lower importance; slow tasks have to wait their turn for processor time.

Further documentation for our final program can be found in Appendix H.

Task Structure -

We have assigned several dedicated task systems in our microcontroller for governing the various subsystems.

Actuator tasks have been created to control each axis of rotation of the panel; these utilize proportional control loops. The actuator tasks interface with the board's motor driver chips to avoid bringing dangerous voltage and current loads near the microcontroller. The actuators are powered using microcontroller-driven pulse width modulation. When not in use, the actuator drivers can be disabled to save power.

A temperature measuring task is used to measure and interpret data from the digital thermometers. To collect the temperature data, a one-wire data transmission protocol was employed. This data line is shared by all of the attached temperature sensors. Once these digital

values are loaded into the microcontroller, the temperature task averages them and sends them to the display task.

The display task handles all communications between the microcontroller and the LCD display. We elected to use an SPI communications protocol to communicate with the LCD display (to avoid using too many I/O pins on the microcontroller board). The SPI shares the same pins as the encoders (since all of these devices use SPI).

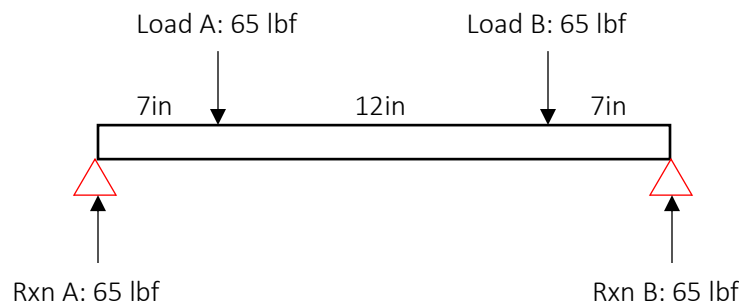
6. DESIGN CONSIDERATIONS

6.1: DESIGN ANALYSIS:

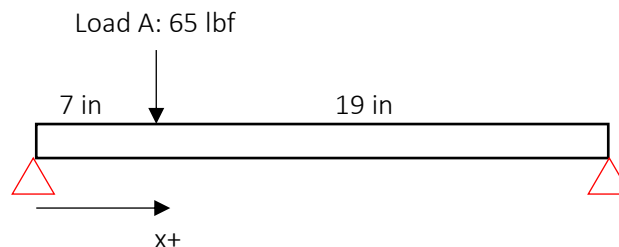
In order to validate the sizing of the load-bearing components of our design, namely the T-Slot aluminum frame, the steel drive shafts, and the caster mounting plates, we turned to hand calculations and analysis. We sized the parts according to rough estimates, and then used the dimensions to calculate the stresses and deflections that a given component might experience. As a general rule, we wanted a safety factor of at least 2 for the worst-case static loading, with panel deflections less than 0.2 inches. We have broken the analysis down by component.

6.1.1 ANALYSIS OF STEEL DRIVE SHAFT DEFLECTION

The ‘elevation axis shaft’ supports the panel weight, counterweight, the supporting drive system, and frame. It is subject to vertical radial loads at the two points where the panel clamps onto the shaft, and where the bearings support it from below. Using the geometry in the solid model (which is symmetric about the midpoint), and imposing global equilibrium, we can easily determine the loads at each bearing and attachment point:



By inspection, we can tell that the point of maximum deflection will be at the center of the axle. However, since we are concerned with the deflection of the panel itself, which is supported by a bracket 7 inches from each end, we need only find the deflection at this point. Given the symmetry of the loading, we can break the beam into two halves and calculate the deflection of one side alone. The first load is shown below:



Next, we can introduce the following beam loading equation for deflection at any point along a beam due to an eccentric point load at location $a=7$ inches. We will use the formula to determine the deflection at the bearing block, $x=7$ inches.

$$\delta(\text{bearing block}) = \frac{Fa(3(a+b)^2 - 4a^2)}{48EI}$$

F – Force applied * Factor of Safety [lbf]

a – distance to load [in]

b – remainder of beam [in]

E – Modulus of Elasticity for Aluminum [psi]

I – area moment of inertia for beam [in^4]

The first three values in the equation above are given, and the modulus of elasticity is easily found in tables to $10\text{E}6$ psi. To determine the area moment of inertia for the beam, we use a known formula for hollow tubes:

$$I = \frac{\pi}{4}r^2$$

Given that we have specified 5/8 inch OD shafts, we can easily calculate the area moments of inertia:

$$I = \frac{\pi}{4}r^2 = \frac{\pi}{4}0.625^2 = 0.306\text{in}^4$$

Inserting this into the beam deflection equation above, we can find the deflection of the shaft

$$\delta(\text{bearing block}) = \frac{(65 \text{ lbf})(7 \text{ in})(3(26 \text{ in})^2 - 4(7 \text{ in})^2)}{48(10\text{E}6)(0.306\text{in}^2)} = 0.0057 \text{ inches}$$

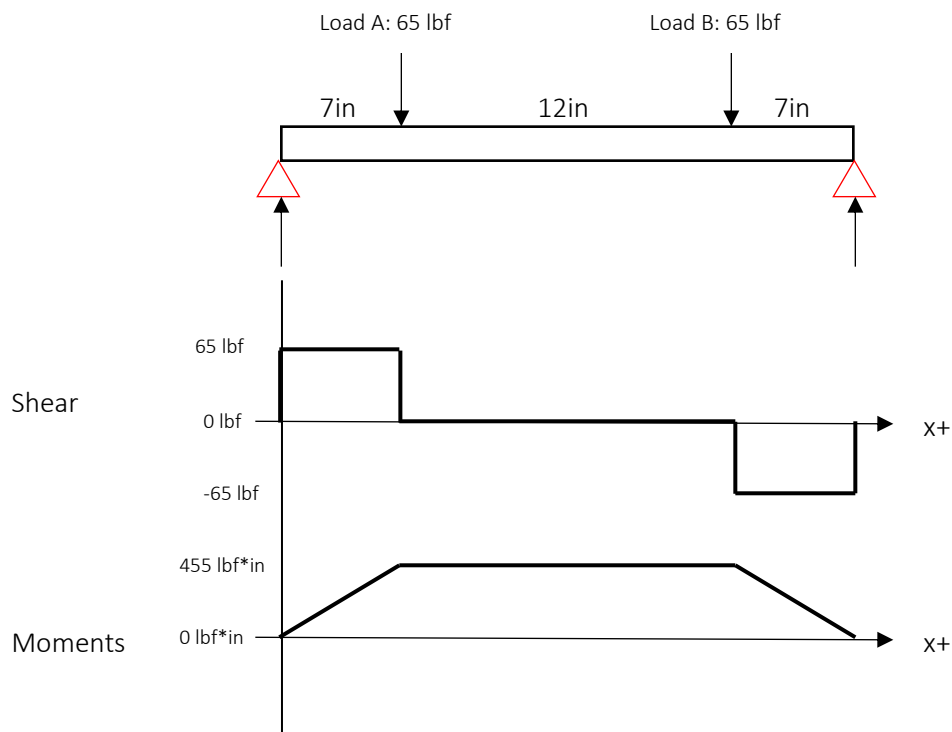
A deflection of 6 thousandths of an inch is about the thickness of a sheet of paper, and this is within the tolerance that we specified of 0.2 inches.

6.1.2 ANALYSIS OF STRESS IN STEEL DRIVE SHAFT

The stress in a beam undergoing shear and moment loads can be described by the equation

$$\sigma = \frac{F}{A} + \frac{My}{I}$$

The maximum bending stress will occur in tension at the outermost fibers of the axle, so we can substitute the outer radius of our tube for y , the distance from the neutral axis to the element in question. We determine the loads on the axle by constructing the shear-moment diagram shown below:



The cross sectional area of the axle is given by the simple formula

$$A = \pi(r^2)$$

Combining the stress equation with the area moment of inertia determined in the calculation above, we can determine the stress to be 519 psi. With a yielding strength of 32,000 psi, we can see that the axle has a safety factor of 61 to yield. This meets our criteria.

6.1.3 ANGULAR DEFLECTION OF FRAME

In this section, we will consider the angular deflection of the panel due to frame deflection only. This will validate the stiffness of our frame, which must be stiff enough resist angular deformation that causes the panel to exceed our angle misalignment specifications, set at 1 degree of misalignment in any direction. We will examine two important loading cases: a vertical load on the student work surface, and a wind load on the panel. These loads will be applied to the frame at their appropriate locations, and we will observe several sources of deflection, including torsion in the crossbeam member of the base and bending in the two rail members.

First, to define the loading cases considered, we need to make assumptions about reasonable upper limits for weight placed on the student work surface, and for wind loading on the panel.

The work surface will be supported by legs that tie it to either the upper crossbars of the frame, or to the rail members of the base. It should be able to support an estimated 50 lbs, which is roughly the weight of a few textbooks and the force of a student leaning on it. Depending on the way this surface is supported, it will either cause a moment about the crossbeam member of the base and bending in the two upright crossbeam members and the two rail members, or it will cause the load to flow directly through the supports into the tires. We will analyze the first scenario since it is the “worst case,” and if it passes we won’t bother with the other.

The panel will be operated outdoors, where the possibility of wind loading is very real. San Luis Obispo is a windy area, and we must take this into account in our frame stiffness design. However, while we can design the rig to withstand some degree of wind loading, we recognize that there is no way to design the rig so that it will never deflect. In general, drag forces on a bluff body increase with wind velocity squared, which means that as wind speed gets higher and higher, the rig could experience very high loads, on the order of hundreds of pounds, distributed across the panel area.

The following data was taken from an online database of yearly weather patterns for cities across America, and it helps define our “acceptable wind load” for these calculations (source: weatherspark.com):

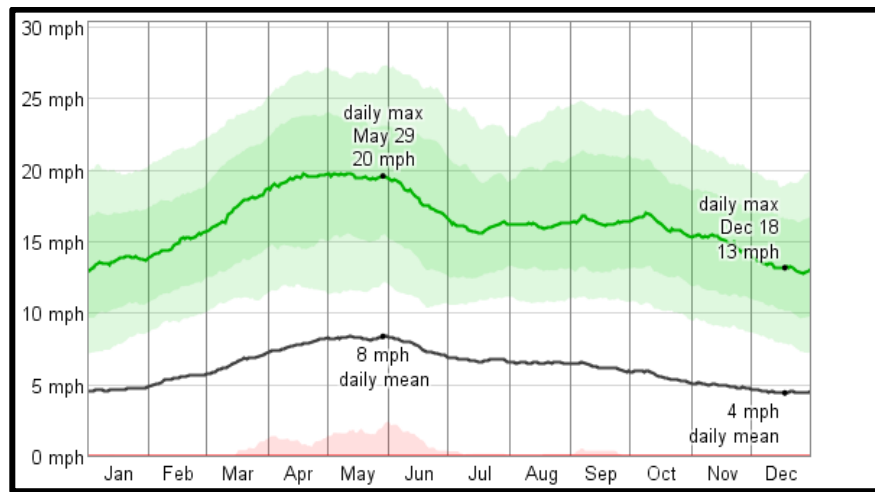


FIGURE 46 - AVERAGE MAXIMUM WINDSPEED THROUGHOUT THE YEAR FOR SAN LUIS OBISPO

Over the course of the year typical wind speeds vary from 0 mph to 20 mph (calm to fresh breeze), rarely exceeding 27 mph (strong breeze).

The HIGHEST average wind speed of 8 mph (gentle breeze) occurs around May 29, at which time the average daily maximum wind speed is 20 mph (fresh breeze).

The LOWEST average wind speed of 4 mph (light breeze) occurs around December 18, at which time the average daily maximum wind speed is 13 mph (moderate breeze).

Taking this data into account, we will design the rig to stay in alignment under average wind loading of 15mph. This represents a rough average for 95% of the average daily maximum wind speed over a given year. The following calculation resolves this wind speed to a force distributed over the panel. This load will be most serious when it is oriented directly toward the panel. We will consider the case where the panel is in operational configuration (elevation angle set to 45 degrees) rather than in storage configuration because for this calculation we are concerned that wind loading will pull the panel out of alignment rather than tip it, and alignment is only relevant during operation.

Drag forces on a bluff (non-streamlined body) can be described by the equation

$$F_{drag} = c_d \rho v^2 A / 2$$

F drag - Force on panel due to wind loading

Cd – drag coefficient [1.2]

ρ.. – density of air [4.61E-5 lbf/in³]

A – Area of panel [3100 in²]

V – velocity of airflow [26.4 in/s]

The appropriate drag coefficient was selected from the diagram below. The drag coefficient is a function of geometry and air conditions. Values for a flat plate perpendicular to a flow vary between 1.2 and 2.0.

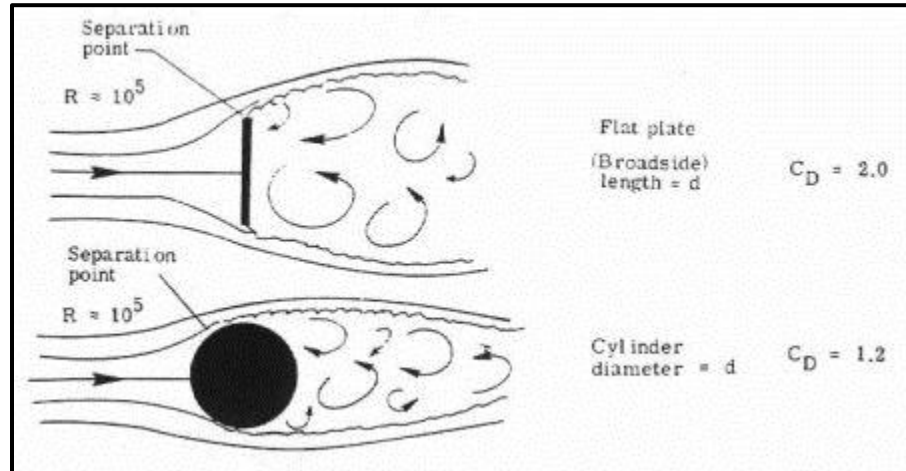


FIGURE 47 - DRAG COEFFICIENTS FOR BLUFF BODIES

Now we calculate the wind loading for our worst case:

$$F_{drag} = c_d \rho v^2 A / 2 = (2.0)(0.0000461)(3100)(26.4)^2 / 2 = 199.2 \text{ lbf}$$

Given a combined loading of 50 lbf on the work surface and roughly 200 lbf on the panel in operational configuration, we will examine three sources of angular deflection: torsional deflection in the crossbeam member of the base (1), bending deflection in the rail members of the base (2), and bending deflection in the upright crossbeam members (3). Refer to the diagram on the following page for a clear depiction of these three deflections.

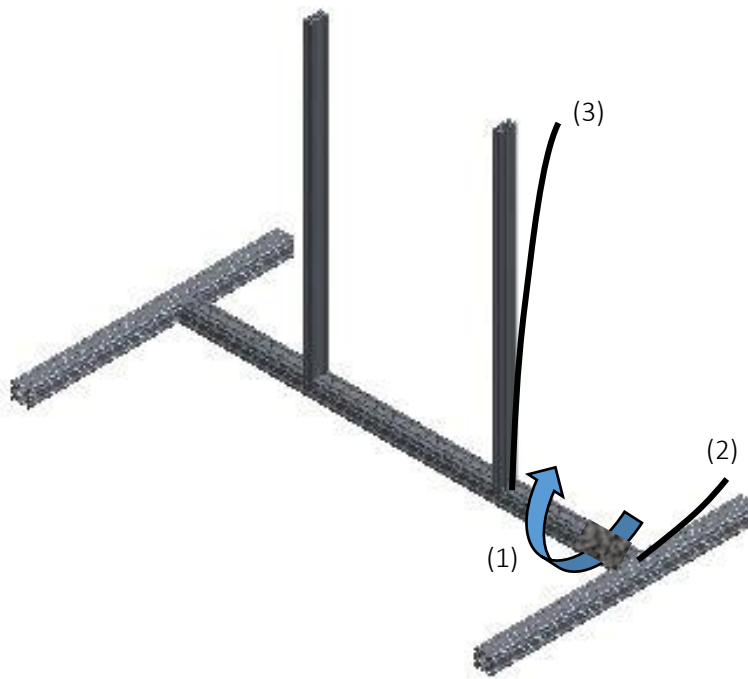


FIGURE 48 - REPRESENTATION OF ANGULAR DEFLECTIONS IN FRAME THAT AFFECT PANEL ANGLE

First, to consider the torsional deflection of the crossbeam member, we resolve the two loads to moments about the crossbeam member. Multiplying the loads themselves (red arrows) by their respective lever arms (green lines, see geometry below), we find this moment:

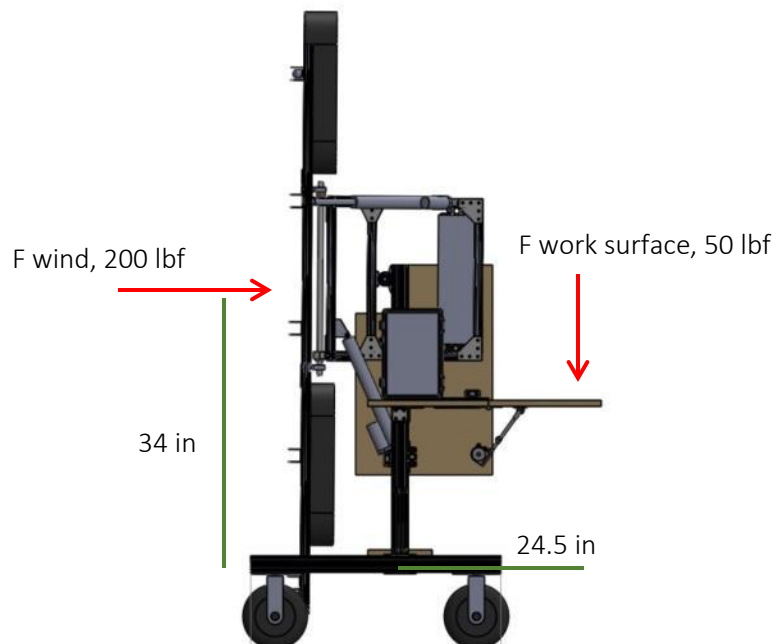


FIGURE 49 - WORST LOADING CASE FOR OPERATIONAL USE

$$M_{crossbeam} = (50lbf)(24.5in) + (200lbf)(34in)$$

$$M_{crossbeam} = 8025 \text{ in} * lbf$$

In general, angular deflection about a shaft in torsion is given by the formula:

$$\theta_{torsion} = \frac{Ml}{JG}$$

with the following variable definitions:

M – Moment about the beam

l – Length of beam

J – Second moment of area for beam geometry

G – Modulus of rigidity for beam material

From the geometry shown in Figure 39, we can see the length of the beam we are interested in to be 15.25 inches. We are only interested in this section of the beam because any deflection between the two upright members will not cause the upright members to rotate. We have already calculated M to be 8025 in*lbf, and the modulus of rigidity for aluminum is tabulated at 3.9E6 psi⁵. This leaves only the second moment of area, which not entirely straightforward to calculate from first principles because it depends on the cross sectional area of the beam in question, which is shown below:

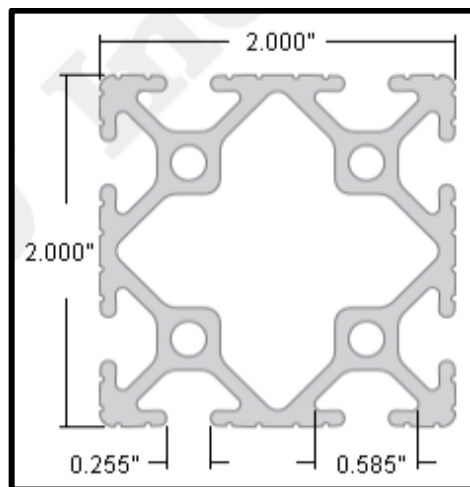


FIGURE 50 - CROSS SECTION OF 2X2 FRACTIONAL T-SLOT EXTRUSION

⁵ The Engineering Toolbox *Modulus of Rigidity for Various Metals*. 2/01/14.

However, the second moment of area is related to the two area moments of inertia used in bending calculations by the following formula:

$$J_z = I_x + I_y$$

Luckily, the area moments of inertia are specified by the manufacturer on the data sheet for this type of T-slot, and because the profile exhibits square symmetry, they are the same value:

$$I_x = I_y = 0.5509 \text{ in}^4$$

And,

$$J_z = 1.102 \text{ in}^4$$

This allows us to calculate the expected angular deflection in the crossbeam due to torsion to be

$$\theta_{\text{torsion}} = \frac{Ml}{JG} = \frac{(8025 \text{ in} \cdot \text{lbf})(15.25 \text{ in})}{(1.102 \text{ in}^4)(3900000 \frac{\text{lbf}}{\text{in}^2})} = 0.0285 \text{ radians} = 1.63 \text{ degrees}$$

Next, we calculate the bending in the upright members to determine their angular deflections at the point of attachment with the panel, namely the elevation angle bearing brackets. The geometry for this type of deflection is shown in Figure 39, and the simplified loading scheme is shown below:

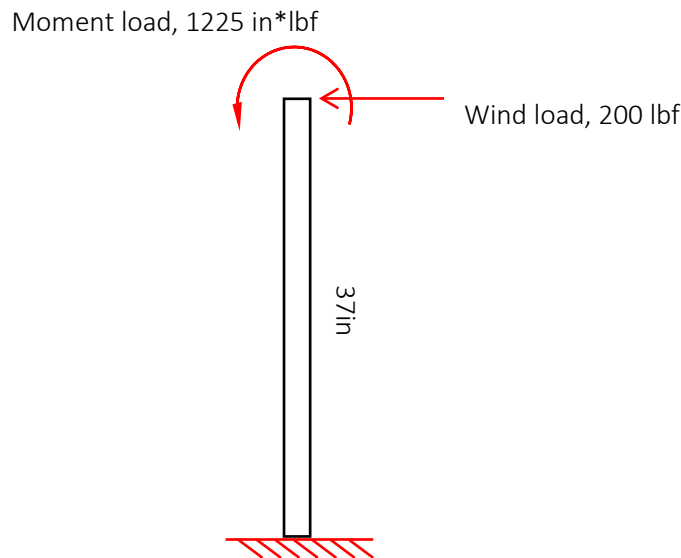


FIGURE 51 -SIMPLIFIED LOADING FOR UPRIGHT MEMBERS

Again, because of symmetry, we will only calculate the deflection on one side, and consider the other side to show the same deflection under the same loading. We will treat the upright members as cantilever beams with a point load and a moment load on the tip. The wind load on the panel resolved to the horizontal will be the point load, and the work surface load

resolved to the bearing bracket will be the moment load (see above for clarification). Both of these loads have already been calculated:

$$M_{work\ surface} = 1225\ in * lbf$$

$$F_{wind} = 200\ lbf$$

We will use simple beam deflection formulas to calculate the deflection and the principle of superposition to combine them, giving the total deflection. The angular deflection at the tip of a cantilever beam due to a moment and point load at the tip is given by:

$$\theta_{upright\ bending} = \frac{Ml}{EI} + \frac{Fl}{2EI}$$

Where:

M – Moment load

l – Length of beam

F – Point load

E – Modulus of elasticity

I – Area moment of inertia

Since M, F, l, and I were all determined in the previous section, and E is tabulated for aluminum at 10.2E6 psi⁶, we can determine the total angular deflection in the upright member to be:

$$\begin{aligned}\theta_{upright\ bending} &= \frac{(1225\ in * lbf)(37\ in)}{\left(10200000\ \frac{lbf}{in^2}\right)(0.5509\ in^4)} + \frac{(200\ lbf)(37\ in)}{2\left(10200000\ \frac{lbf}{in^2}\right)(0.5509\ in^4)} \\ &= 0.0814\ radians = 0.466\ degrees\end{aligned}$$

⁶ The Engineering Toolbox *Modulus of Elasticity for Various Metals*. 2/01/14

Finally, we calculate the angular deflection in the rail members of the base due to the moment in the crossbeam member of the base (rail member highlighted below).

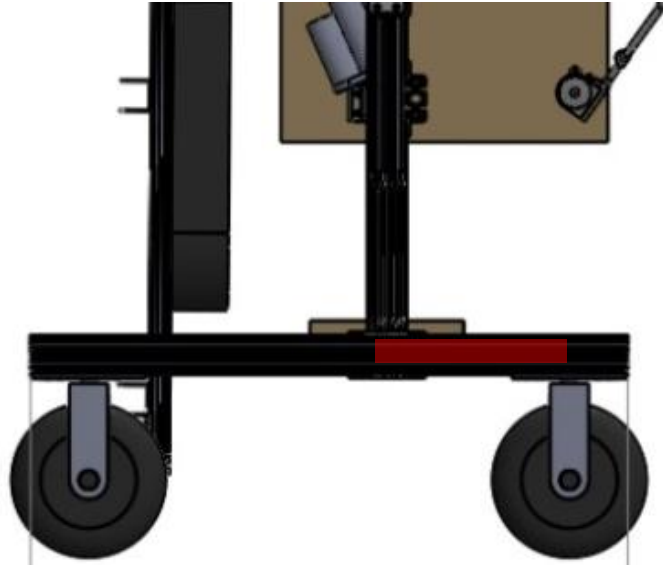


FIGURE 52 - CRITICAL SECTION OF BASE RAIL MEMBER

Again, because of symmetry, we need only calculate the deflection on one rail. We will treat the portion of the rail member from the upright members to the rear wheel as a cantilever beam with a moment load on the end, as shown in the diagram below:

Crossbeam Moment, 8025 in*lb

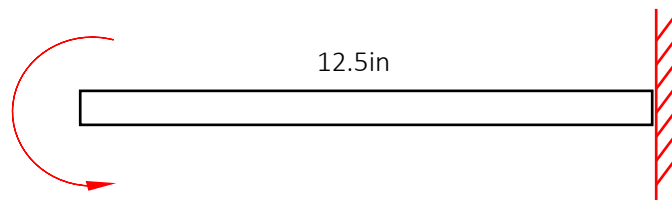


FIGURE 53 - SIMPLIFIED LOADING FOR RAIL MEMBERS

We will use beam deflection formulas to find the angular deflection at the tip:

$$\theta_{rail\ bending} = \frac{M_{crossbeam}l}{EI}$$

where the variables are exactly the same as in the above section, except that the moment load is the moment in the torsional member. This moment load was found in the first section by resolving the work surface load and the wind load down to the cross beam member of the base. Inserting all known values into the above equation, we find the deflection of the rail member:

$$\theta_{rail\ bending} = \frac{(8025\ in * lbf)(12.5\ in)}{(10200000\ \frac{lbf}{in^2})(0.5509\ in^4)} = 0.01785\ radian = 1.023\ degrees$$

In order to find the total panel deflection, we use the principle of superposition to combine the angular deflections found in the previous sections. Thus,

$$\theta_{panel} = \theta_{torsion} + \theta_{upright\ bending} + \theta_{rail\ bending}$$

$$\theta_{panel} = 1.630 + 0.466 + 1.023 = 3.12\ degrees$$

This deflection is outside the range of our panel misalignment specs. Consequently, we added a second crossbeam member to our design in order to improve frame stiffness. This crossbeam member will the same length and cross section as the existing one, and will brace the uprights with angle pieces.

6.1.4 TIPPING CONDITION ANALYSIS

One potential safety hazard discussed in the Safety section (6.2) of this report is the possibility of the rig tipping over, which could damage expensive components and cause injury to students. To avoid tipping, we can widen the wheelbase of our project, and we can lower the center of gravity. However, these both have their limits – the wheelbase cannot be widened so much that it can't fit through a doorway, and certain heavy components linear actuators, counterweight, and our panel cannot be lowered due to design geometry.

In the previous section, we mentioned the average wind conditions in San Luis Obispo, and we picked an average wind load that our rig should be able to withstand without deflecting. However, the panel deflecting a few degrees is a relatively minor failure mode compared with the rig actually tipping over. Consequently, we're going to pick a much larger wind load that our rig should withstand before tipping than the load it takes to deflect the panel. From the plot in Figure 37, we will choose a maximum wind load that our rig can withstand to be 20mph. Additionally, we will consider the worst-case loading scenario, meaning that the wind is acting exactly normal to the panel, and the panel is in its fully vertical (folded) state. Given these parameters, using the same calculation as above, the wind load can be shown to be 350 lbf.

This diagram shows the rig in profile, with the wind load resolved to the center of the panel (in this diagram, the yellow circle denotes the location of the center of gravity):

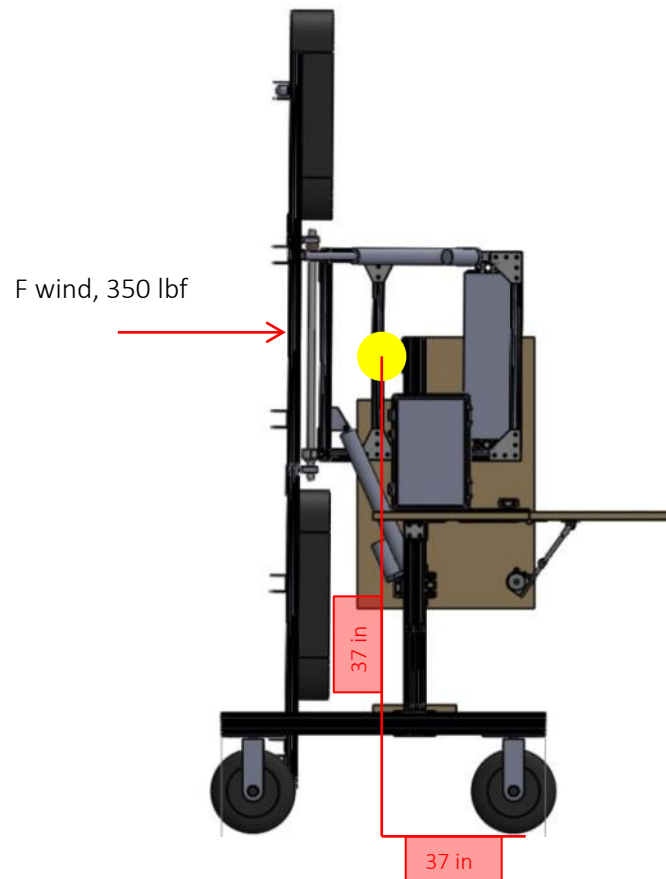
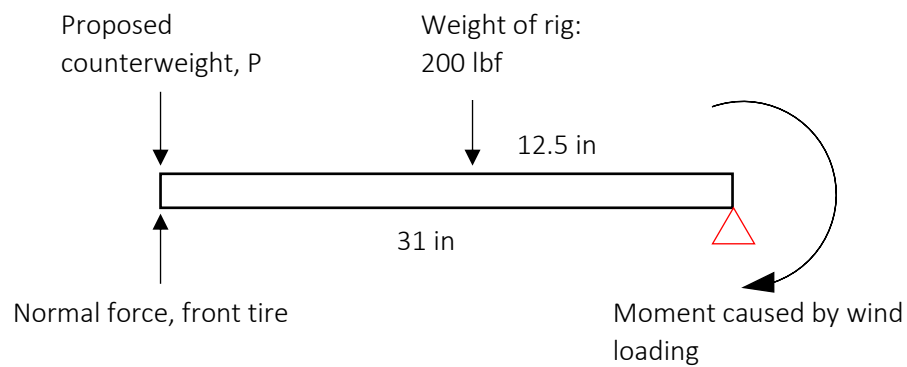


FIGURE 54 - PROFILE VIEW OF RIG ON LEVEL GROUND

A simple way to calculate the tipping force is to consider the contact patch of the back tires as a pivot, and then to sum the moments about that pivot:



Since we are examining the tipping condition, we can set the normal force of the front tire to zero. This leaves the weight of the rig, applied below the center of mass with a magnitude of about 200lb, and the moment about the back wheel caused by the wind load on the panel. We've also introduced a proposed counterweight force that may be needed in order to prevent the rig from tipping. First, we will ignore this proposed counterweight, sum moments about the pivot and solve for the wind load required to tip the panel:

$$\sum M_{pivot} = F_{wind}(37 \text{ in}) - (200 \text{ lbf})(12.5 \text{ in}) = 0$$

Solving this equation gives:

$$F_{wind} = 67.5 \text{ lbf}$$

This is clearly much too low of a wind force that tips our rig than we can tolerate. Next, we'll introduce the wind force we would like to withstand, 350 lbf, as well as the proposed counterweight into the moment equation. This allows us to find the required counterweight force applied at the front wheel that would prevent the rig from tipping under the wind load.

$$\sum M_{pivot} = (350 \text{ lbf})(37 \text{ in}) - (200 \text{ lbf})(12.5 \text{ in}) - P(30 \text{ in}) = 0$$

This gives:

$$P_{required} = 34.8 \text{ lbf}$$

This tells us we need to add some weight to the base in order to resist tipping. One point to consider is that this is the total weight required, and it will be divided evenly between the two rails, keeping the rig symmetrical. This estimate shows that we should include around 35 pounds (or 8 standard bricks) to the base.

One alternate tipping condition is unrelated to wind, but just as important. We need to determine the angle at which the rig will tip over while being rolled up a slope. Again, we will calculate the tipping point by finding when the normal force on the front tires falls to zero. However, since our rig will be transported by rolling it "sideways" (that is, with the narrower base dimension forward), we will look at the instant that the normal force on the tires on the left side of the rig reaches zero. Interestingly, this analysis is completely geometrical and does not require knowing any weights or forces beyond the location of the center of gravity. The diagram below guides our analysis:

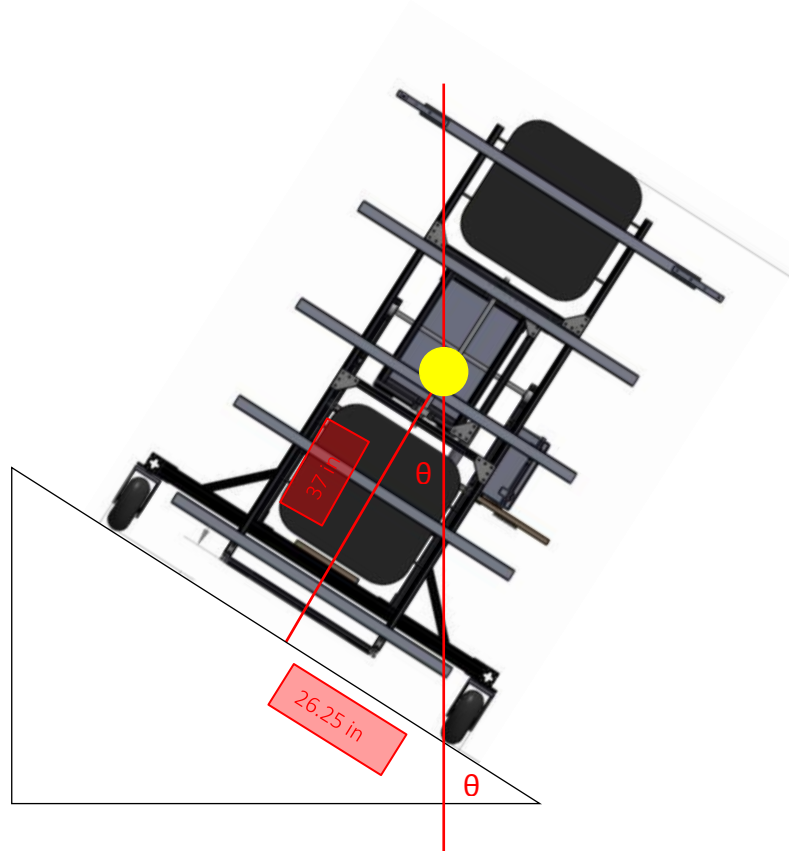


FIGURE 55 - FRONTAL VIEW OF RIG ON SLANTED GROUND

The tipping point will occur when the center of gravity is brought directly over the contact point of the rear wheels. Using the geometry in the figure above, we can calculate the angle required to induce this condition by the following trigonometric relationship:

$$\theta_{to\ tip} = \tan^{-1}\left(\frac{37}{26.25}\right) = 45.6\ degrees$$

This angle far exceeds the angle of any ramp or inclines that the rig will be pushed up.

6.2: SAFETY CONSIDERATIONS:

This section will examine our final solar panel trainer design through a lens of safety. We will discuss potential safety hazards and the measures that we have taken to eliminate or reduce them. Areas of discussion include: mechanical, electrical, user interface, and weatherproofing concerns.

Our solar trainer system uses linear actuators to articulate a panel on two axes and two fans are used to cool the back of the panel. The spinning blades of the fans are contained behind plastic grilles to prevent them from being accidentally contacted by students or instructors. Few pinch points exist, especially because we have stripped the design of gears, belts, pulleys, chains, and sprockets.

The trainer moves a panel which weighs between 20 and 30 pounds. However it moves the panel at angular velocities on the order of 1-3 degrees/second; consequently, the corresponding momentum of the rotating pieces is low. That said the specified actuators are extremely high-torque and have little to no play; in other words, if they begin to contact an obstacle, there is no clutch mechanism in the drivetrain (or flexibility in the frame) to prevent the actuators from continuing to drive forward into the obstacle. In the event that the panel starts to collide with a foreign object, the user is able to immediately stop all panel movement by hitting an emergency stop button located on the student work surface. From here, users can manually move the panel to a position where no interference occurs.

Since the trainer has a relatively small footprint, it is prone to overturning in high winds (especially when a large panel is mounted on it). We have determined via calculations that the trainer will tip over before the frame fails or the panel is torn off, for example. Consequently, we have designed the frame with as wide a wheelbase as possible to increase the trainer's stability. We have also placed heavy masses (such as the battery) low on the frame to lower the trainer's center of gravity; this strengthens the rig's resistance to overturning moments.

We have taken care to ensure the user's safety against electrical hazards. The trainer incorporates a large 12V DC battery and an inverter that generates 120 V/AC. The largest panels also generate a nominal 72 V at peak operation, which is run through a charge controller to the battery. All electrical junctions are therefore contained in sealed and grounded junction boxes. Thick-gauge wire (donated by our sponsor) is used to pass electricity safely through the system. We have also properly grounded the system and ensured that all electrical connections are shielded in plastic (so that no bare wire is exposed).

The rig needs to withstand storage and/or operation in wind, fog, humidity, and heat. There is consequently a potential that weather damage could make the rig unsafe to operate; for example, pins on the microcontroller could be shorted out by moisture thereby leading to unsafe operating conditions. We have therefore taken care to place all electrical components in a sealed, weatherproof electrical box with a NEMA 4X rating. In addition, we have chosen to use aluminum and ceramic-coated frame components that will not weaken or corrode in San Luis Obispo's environment. The student work surface was similarly carefully painted and covered to prevent water damage.

Finally, throughout our entire design and assembly process, we have applied the overarching principle that: every part of the system should be easy to use safely. For example, the actuators include limit switches that prevent panel motion beyond safe ranges (regardless of any user commands). The trainer has locking wheels to prevent a rollaway, and both axes move slowly. Above all, the frame was designed with large safety factors, meaning that it will be sturdy enough to last as long as possible even with frequent use by Cal Poly students.

6.3: MAINTENANCE & REPAIR

We have designed the PV Solar Trainer to require minimum maintenance (even when stored primarily outdoors). During the initial design stages, we made the decision to use aluminum for frame construction because of its corrosion resistance. Our design also encloses all of the electronics in weatherproof boxes so that they will not fail as a result of water or humidity damage. When sourcing the caster wheels for the frame, we selected relatively expensive wheels that are meant to last longer and are rated to support loads of up to 450 lbs each. Our intent was to minimize the likelihood of part failures by overdesigning critical components (within reason).

Furthermore, in the event that a user error damages one of the parts (e.g. the rig is rammed into a wall at high speed and a bracket breaks) we have designed the rig to be as modular as possible. We have used standard fasteners wherever possible and minimized the number of custom components.

In the event that an electronics failure occurs, all electrical parts can be replaced relatively easily. Professor Ridgely can provide replacement ME 405 boards (or components) and all of the board's parts and schematics are thoroughly documented. The sensors and other electronics are all available via DigiKey and can be ordered with relative ease.

If the actuators cease to receive power from the ME 405 board, double check that the fuse mounted on the face of the board is intact. If it has been damaged, replace it (Professor Ridgely should have spares) and re-verify the wiring leading to the motor driver. Ensure that all of the traces are still functional (note: one trace was damaged and bridged during constriction). In the event that the motor drivers burn out, they can be removed and replaced using the tools available at a hot air rework station.

If the ATmega 1281 microprocessor burns out, it can be removed and replaced using a hot air rework station. Once replaced, a USBtinyISP programmer should be used to reset the internal fuses provided on the ATmega chip. Once done, the PV Solar Trainer software suite can be re-uploaded into the new ME 405 board.

If the readings displayed from the encoders appear incorrect, verify that the wires leading between the encoders and the SPI bus are intact. If both encoders both output incorrect values to the screen, it is possible that their chip select wires have been switched on the breadboard. In the event that the encoder is truly broken, a new encoder can be installed in its place. The new encoder may need to be recalibrated in the code (or have its magnet manually rotated until the calibration matches).

If the readings coming from the temperature sensors are the same value (or cease to update), one of the datalines is likely shorting to either power or ground. To determine which wire is causing the program, disconnect each sensor (one at a time) until the temperature readings return to normal. If necessary, a new temperature sensor can be soldered to the end of the phone cable (in place of the broken one).

6.4: COST ANALYSIS

Several major changes occurred to our budget over the course of the project. The original budget provided by Professor Dolan was \$1,000. After our conceptual design review, this budget was expanded to \$1,500. The final projected cost of the system (at the critical design review) was roughly \$1,700. This meant that, as of February 2014, the project was projected to run roughly \$200 over budget.

The largest single expense of the project was projected to be the T-slot aluminum frame, costing about \$900. Roughly half of the cost of the frame resulted from the brackets, plates, and fasteners; all which expenses that could not be avoided. Immediately after the critical design review however, we received word that the Cal Poly ME department had secured a large donation of T-slot components for student projects. As a result of these savings, we were able to order several upgraded components.

The final cost of the components required to build the Solar PV Trainer was \$1,415 (plus the value of the donated T-slot).

A detailed parts list can be found in Appendix E.

7. MANUFACTURING

7.1 FABRICATION

For this project, manufacturing tasks were generally split between fabrication tasks and assembly tasks (since most of our parts were off-the shelf units). The following sections detail each of the fabrication tasks that our team completed in the second two quarters of senior project; fabrication tasks are defined as processes that involve cutting, drilling, joining, or bonding different raw materials.

RESIZING THE ALUMINUM T-SLOT

We were lucky enough to receive a large donation of material from a T-slot manufacturer. Due to a miscommunication that occurred during the ordering process however, the extrusion arrived in 20 foot long sections. As a result, the first step in the fabrication process was to cut the extrusion to the correct lengths (as specified in Appendix F). This was a relatively simple process and was completed using the aluminum chop saw in the Bonderson machine shop. In general, we attempted to cut the lengths beginning with the longest pieces first; in this way, if any pieces were cut incorrectly, the resulting scraps could be repurposed to become one of the rig's shorter frame members. While making cuts (see below), we only performed one cut at a time per end to avoid ending up with pieces that were slightly shorter than we wanted (due to the thickness of the blade). After making each cut, we verified the length of the resulting piece using a tape measure. Wherever necessary, we deburred the inner and outer edges of every cut we made using files. Additionally, we saved all scraps of T-slot (even those that were only a few inches long); these scraps became part of the encoder brackets.



FIGURE 56 – CUTTING T-SLOT ON AN ALUMINUM CHOP SAW

CASTER WHEEL MOUNTING PLATES

In order to properly stabilize the rig, we purchased four large 8-inch diameter locking casters. These casters are very well suited for our purposes; however the provided mounting holes are too far apart to interface with the slots in our T-slot aluminum. Our solution was to machine a small, $\frac{1}{4}$ inch thick aluminum plate for each wheel that was large enough to span the holes in the casters. These plates also included four holes spaced 2 inches apart to accept standard T-slot fasteners.



FIGURE 57 - 8 INCH CASTER WHEEL

Fabrication of these plates took place in the Hangar machine shop. We began with a 12x12 inch aluminum plate. We used a vertical bandsaw set to the appropriate speed (and cooled with plenty of lubricant) to cut the plate into four rectangular pieces. We then used a spring-loaded center punch to mark where we wanted to drill the holes. Next, we used a drill press to drill 8, $\frac{1}{4}$ inch diameter holes in the proper locations. This diameter was chosen to accommodate our $\frac{1}{4}$ inch carriage bolts and T-slot fasteners. These parts were then deburred using files and other deburring implements.



FIGURE 58 - USING THE DRILL PRESS TO DRILL HOLES IN CASTER MOUNTING PLATES

ENCODER BRACKETS

The encoders that we purchased did not come equipped with the brackets that we expected. These encoders were made specifically to fit onto the back of a particular variety of electric motor and therefore have an internal cap with a magnet on it that is meant to fit over a 6mm motor shaft. They also feature an external housing that incorporates the electronics that sense position. This housing includes screw holes which are designed to align with existing holes on the back of a motor. In order to allow the encoders to interface correctly with our 5/8 inch shafts, we designed a custom 3D-printed part that stepped down the shaft diameter from 5/8 inch to 6mm (see image at right).



FIGURE 59 - CUSTOM 3D PRINTED ENCODER ADAPTER, UNPAINTED

Next, we machined a custom encoder bracket from a leftover piece of 1/4 inch thick aluminum plate. This piece consisted of a rectangular bar with two holes for mounting screws, one hole for the 6mm shaft to pass through, and two small tapped holes that accepted M3 mounting screws from the encoders. We used the vertical bandsaw and the drill press in the Bonderson machine shop to cut these brackets, and we used a metric tap set to drill and tap the mounting holes for the encoders by hand.

We spray painted the encoder adapters black to match the rest of the black accents, and then pressed them onto the protruding ends of the shafts. Next, we mounted the aluminum brackets to the frame with enough washers to that they cleared the shoulders of the encoder adapters. Because of this extra spacing, we had to use carriage bolts instead of T-slot bolts. The data sheet called for a 9mm protrusion of the 6mm diameter shaft, so we used a micrometer to measure the distance from the tip of the encoder mount to the face of the encoder bracket, and then a razor saw to cut it down to 9mm. Next, we assembled the encoders in the following order: Screw the base of the encoder housing into the bracket using the tapped holes, place the magnetic cap on the tip of the shortened 6mm shaft, and the clip the main housing over the base to complete the assembly.



FIGURE 61 - PRIMARY AXIS ENCODER FULLY MOUNTED, WITH ENCODER BRACKET VISIBLE

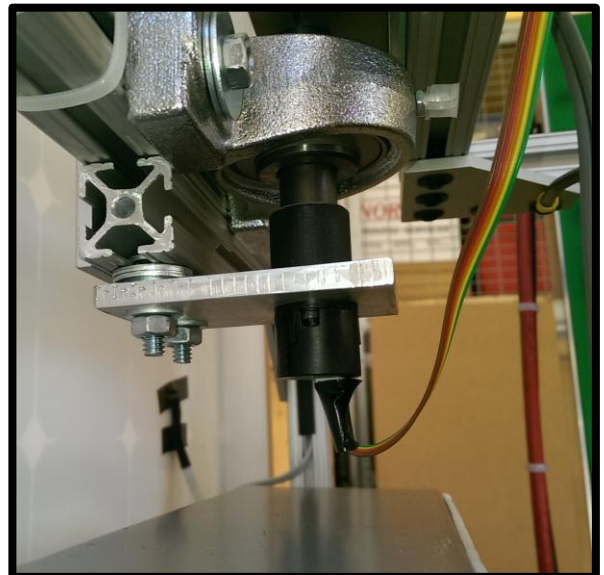


FIGURE 60 - SECONDARY AXIS ENCODER FULLY MOUNTED, WITH SPACING WASHERS, ENCODER BRACKET, AND ENCODER ADAPTER VISIBLE

INVERTER MOUNT, BATTERY MOUNT, AND WORK SURFACE

To make the various mounting and work surfaces, we first used a table saw and jigsaw in the ASI Craft Center to cut a piece of $\frac{3}{4}$ inch plywood to the required dimensions. We sanded the edges of these pieces to remove splinters and imperfections. Next, we sprayed two coats of black paint onto all of the surfaces, allowing proper drying time. This was done to cover any discolorations in the wood and to protect it from moisture. While the paint was drying, we cut the same dimension shapes from green acrylic, taking care to leave the protective plastic on the side that would eventually be facing away from the wood. This protective plastic reduced the likelihood that the acrylic would develop unsightly scratches.

While cutting the acrylic, we found that it was very important to maintain a steady feed rate, or the acrylic would begin to melt and gum up the cutting teeth of the blade. Both a table saw and a jigsaw were used to cut the acrylic.

Once cutting was complete, we clamped the corresponding pieces together and sanded all around the edges. Next, we cleared all surfaces of dust and laid down industrial strength permanent adhesive acrylic tape onto the perimeter of each wood surface, one at a time. We carefully lowered the acrylic pieces onto the wood and applied force.

Once the wood and acrylic were permanently bonded, we cut 'L' aluminum extrusion to the correct lengths and angles so that all edges of acrylic were protected by aluminum. The aluminum extrusions met at the corners in miter joints. These pieces of aluminum were attached to the work surfaces using a combination of industrial adhesive tape and button head screws (the screws were affixed to the edges of the plywood). The thicker extruded aluminum pieces were cut using the horizontal bandsaw in the Bonderson machine shop. A block of wood was used to snug the metal into the jaws of the saw's clamp; this measure prevented the vertical section of the extrusion being cut from vibrating and producing a ragged edge.

WORK SURFACE FOLDING LEGS

We left the design of the folding legs until most of the other manufacturing was finished. We aimed to ensure that this folding action would not interfere with any of the rig's other main components. We cut a short section of a 2x4 piece of pine wood at a 45 degree angle, and spray painted the two resulting pieces black. We mounted them to the underside of the folding table using wood screws, and attached T-hinges with custom drilled holes that held 22 inch lengths of 1x1 inch T-slot extrusion.



FIGURE 62 - UNDERSIDE OF FOLDING WORK SURFACE IN THE "IN USE" CONFIGURATION, WITH THE ANGLED BLOCKS AND PROTOTYPE ACRYLIC FEET VISIBLE



FIGURE 63 - FINISHED VIEW OF THE ALUMINUM FEET, ALONG WITH THE PROTECTIVE ALUMINUM ANGLE EXTRUSION ON THE CROSS MEMBER

Next, we machined several iterations of "feet" from the last of our remaining $\frac{1}{4}$ inch aluminum plate. In order to correctly size the feet, we first made prototypes from scraps of green acrylic (as visible in Figure 62). The feet are essentially tapered extensions of the legs that have a 90 degree fork cut in them so that they sit cleanly on the T-slot cross member.

To make the final version of the feet, we used the vertical bandsaw in the Bonderson machine shop to cut the outline of the foot shape. We found that the blade tended to wander unless the machine was set to the "HIGH" blade speed setting. After the outline of the feet was finished, we drilled two $\frac{1}{4}$ inch holes with the drill press so that the feet could accept two T-slot mounting screws. As a final touch, we placed two short pieces of L-extrusion on the cross member so that the feet wouldn't mar the surface finish of the aluminum T-slot (see Figure 63).

7.2 BUILDING THE MICROPROCESSOR BOARD

BUILDING THE ME405 BOARD

The PCB for the ME 405 board was donated from Dr. Ridgely of the Mechanical Engineering department at Cal Poly. Once all the necessary parts required for the board's operation had been acquired, the board was set up for soldering. First the PCB was held in place using a weighted pair of helping hands to ensure the board could be kept still. Non-corrosive flux was applied to the board areas where soldering was to take place and the SMDs were aligned and soldered into place. These packages were carefully aligned according to the required schematic and then tacked down when aligned to ensure good connection. Once all surface mount ICs were soldered to the board, testing was done using a multimeter to ensure no two pins were bridged by excess solder. Any excess solder found was removed using a solder wick.

Next to be soldered to the board was the larger mini-usb port, reset switch, and fuse holder. The fuse holder and reset switch were relatively simply to solder, but the mini-usb posed a bit of an issue due to the housing making the soldering angle awkward. On future builds, it is recommended to use different mini-usb housing for this reason. After these were soldered to the board all through-hole devices were added to the board as can be seen in Figure 64.

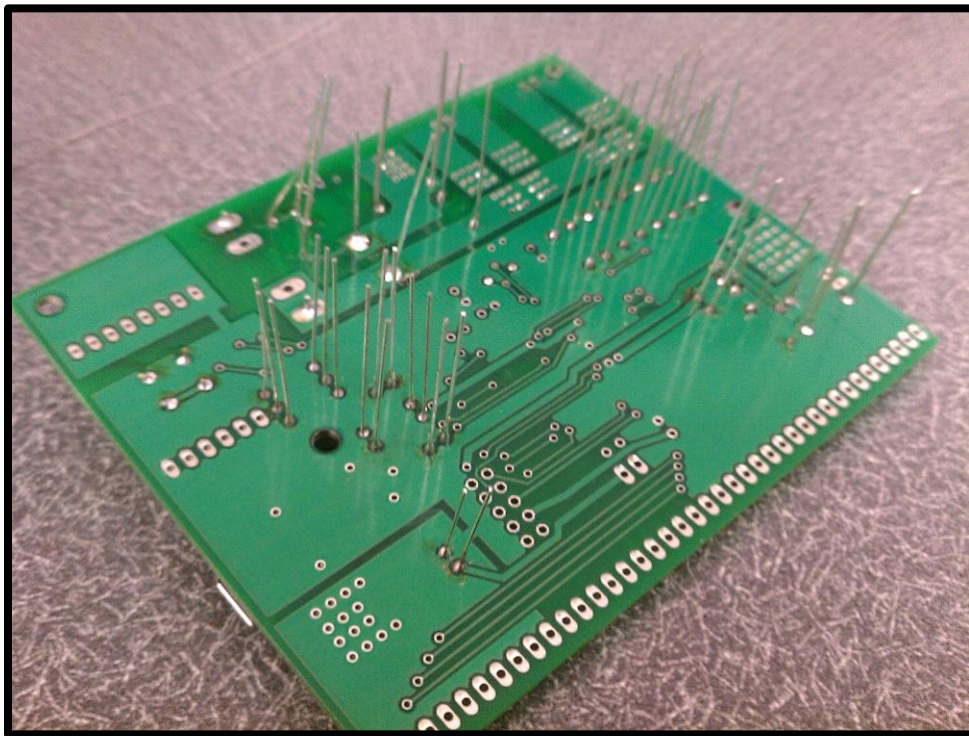


FIGURE 64 - SOLDERING OF THROUGH HOLE COMPONENTS

Lastly, the header pins and terminal blocks were soldered to the board. The ME 405 board usually uses a female header block for its SPI bus, but we substituted a terminal block for ease of prototyping. Once all components were soldered onto the board and continuity checks

had been made in all vital areas, the motor driver fuse was inserted and the board was powered up for testing.

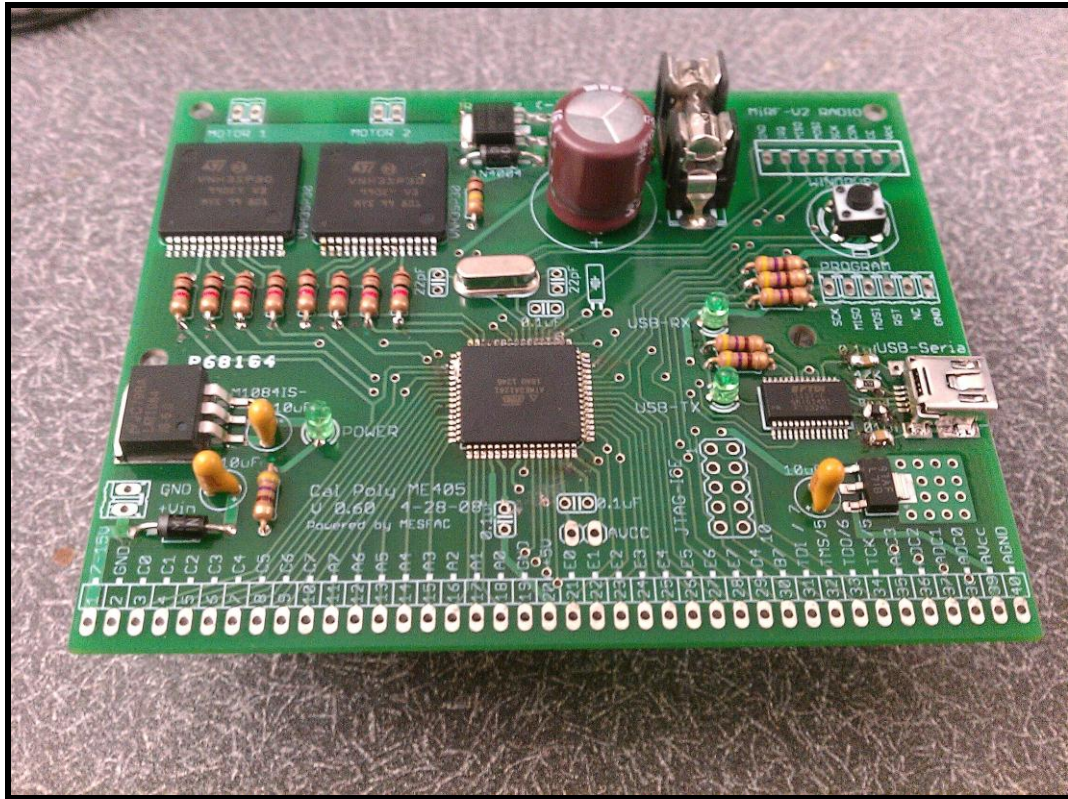


FIGURE 65- ME 405 BOARD WITH ALL COMPONENTS EXCEPT HEADER AND TERMINAL BLOCKS

The final step for making the ME 405 board operational was properly setting the fuses. The board was connected via its an USBtinyISP AVR programmer. Using the makefile provided with our code, the command “make fuses” was issued using the WinAVR software. This setup the Atmel chip installed on the board to operate and communicate properly so that it could then be programmed with our program. It should be noted, the SPI encoders should be disconnected from the SPI bus while programming to avoid problems.

SETTING UP THE LCD SCREEN

The LCD display for this project utilized SPI; to enable this feature a short needed to be placed across the display's second relay. A small piece of wire was soldered across the joint as described in the LCD datasheet to accommodate this. Next, a six position header pin was soldered to the LCD to enable easy plug and play assembly with the LCD. We experienced trouble with our six pin plug however, leading us to solder our SPI wires directly to the header pins. For future builds, if an adequate plug is selected this soldering step may be avoided. The wires soldered to the LCD were then connected to the SPI terminal block on the breadboard to verify the component worked properly. The LCD can be seen in Figure 66 (the pictures were taken before the driver was fully functional).

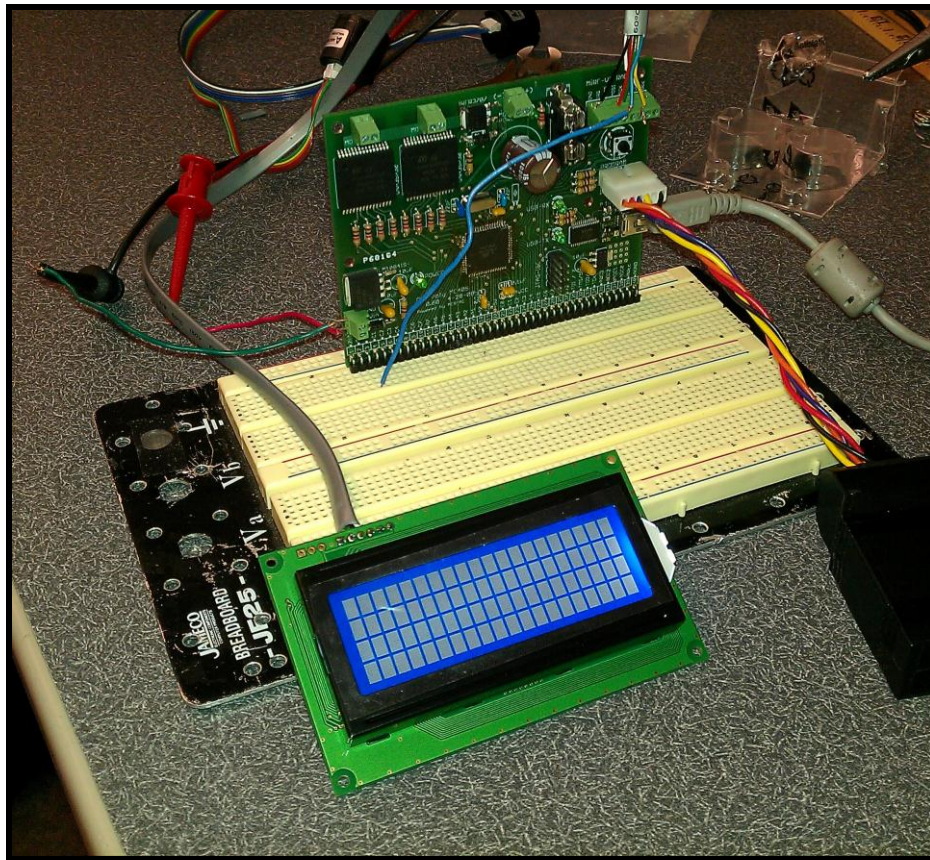


FIGURE 66 - LCD DISPLAY CONNECTED FOR INITIAL PROGRAMMING

CONFIGURING THE KEYPAD

The 16 button keypad used for this project comes with a female header connector (as can be seen in Figure 67). In order to connect to the ME 405 board via a bread board, a set of 6 male header pins were glued into the female header. Once plugged into the breadboard, the pins were connected via short jumper wires to pins E0-E7. On future builds, the jumper wires will not be necessary; they were included simply to speed the prototyping process.

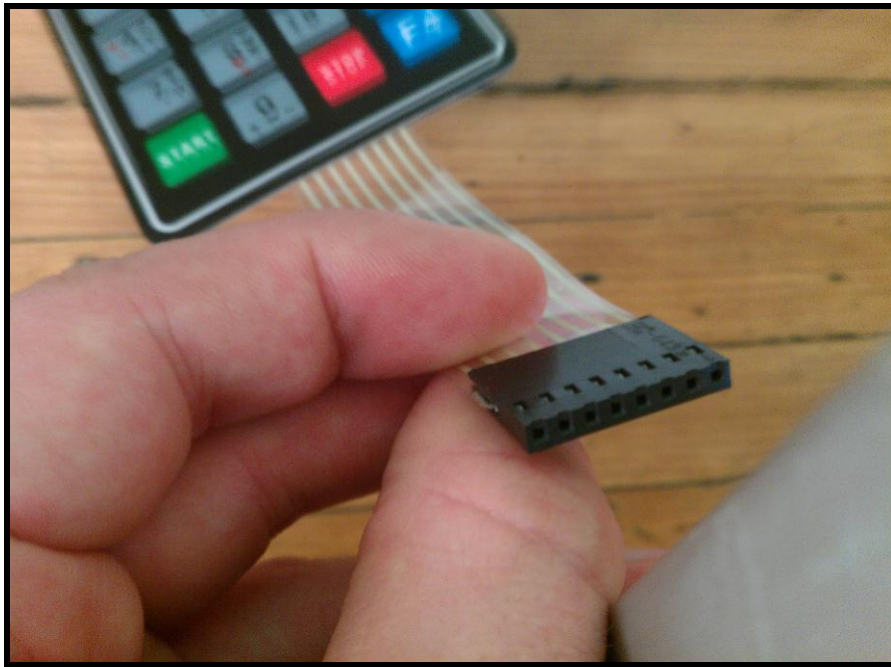


FIGURE 67 - FEMALE HEADER CONNECTION FOR KEYPAD

CONNECTING THE DS18B20 TEMPERATURE SENSORS

The DS18B20 temperature sensors were soldered directly to the wire which was connected to the breadboard. RJ11 phone cable was used due to the availability of the connectors making it easy to splice our network of temperature sensors together. The power and ground pins were connected to the VCC and GND pins via the breadboard on the microcontroller. To connect the dataline to the ME 405 board via pin D4, a 3K ohm pull-up resistor was used to ensure proper data transmission. For future design builds, if new temperature sensors need to be selected it is recommended not to use 1-wire data transmissions as this protocol is very sensitive to problems with the data line.

CONNECTING THE AVAGO ABSOLUTE ENCODERS

The absolute encoders selected for this project have a 5 pin 1.25 mm pitch connector. Unfortunately, our group was unable to find this connector for sale pre-assembled. We attempted to solder together a plug to suit our needs, but this proved to be too difficult with the tools that we had available to us. We circumvented this issue by soldering our SPI wires directly to the pins on the encoders. For future builds, it is recommended to have a premade connector plug available. The final wiring scheme used for all of the electrical components in the electronics box can be seen in Figure 68.



FIGURE 68 - FINAL WIRING SCHEME OF ALL COMPONENTS TO THE ME 405 BOARD

8. DESIGN VERIFICATION TESTING:

8.1: POSITIONING SYSTEM TESTING

The first round of testing that we conducted was intended to verify the performance of the positioning system. In doing so, we quantified two important parameters: the accuracy of the angle measuring system, and the accuracy of the positioning system. This is an important distinction: if the system were only to fulfill one of these requirements, the design would be unsatisfactory. One could imagine, for example, a rig that could measure the angular position of the panel to many decimal places but could not hold a desired angle consistently. On the other extreme, a rig could be built that proved extremely steady at maintaining a position, but could not measure that position with sufficient accuracy. We have designed our rig to satisfy requirements for both parameters (as detailed in previous sections).

As built, the PV Trainer measures each axis angle using an onboard absolute digital encoder. These encoders are coupled to each shaft and are periodically read by the microcontroller.

In order to determine the accuracy of our system (with respect to angle measurement), we positioned the panel at specified set points and then recorded the measured angle displayed on the LCD screen. We then compared this reported angular position to the actual angular position which was calculated geometrically. Because we wished to measure the actual angle very accurately, we used a laser pointer to magnify the angular displacements and thereby reduce measurement uncertainty. The steps below outline the general procedure:

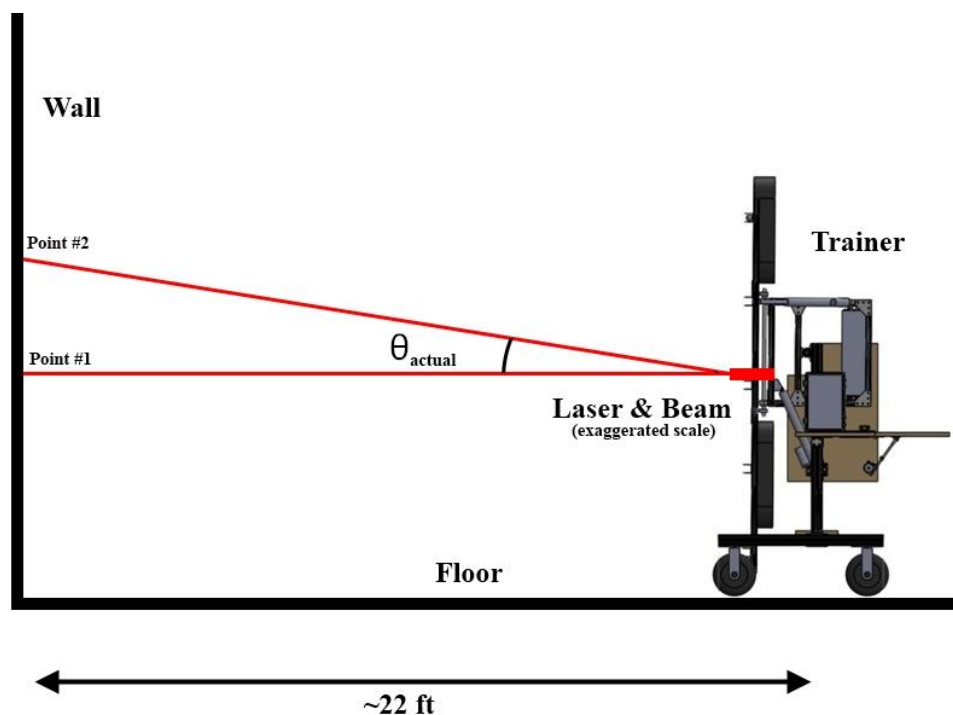


FIGURE 69 – POSITIONING TESTS (EXPERIMENTAL SET UP)

1. Find a vertical wall approximately 15 feet high with level ground in front of it. Place the rig approximately 22 ft from the wall (with the panel directly facing it). Measure and record the distance between the wall and the primary axis shaft.
2. Orient the panel upright (until the primary axis equals zero). Use a bubble level to verify that the front face of the panel is vertical.
3. Securely tape a laser pointer to the side of the panel so that it is horizontal with respect to the ground (normal to the panel) and pointed at the wall.
4. Measure the distance from the laser to the ground and record this length. Next, measure the vertical distance from the ground to the laser point on the wall. Adjust the angle of the laser as necessary until these two measurements are the same. At this point, the laser will be calibrated to the elevation axis of the frame.
5. Slowly increase the elevation angle in increments of 5 degrees. At each interval, measure and record the vertical distance from the ground to the laser, as well as the distance from the floor to the mark on the wall.
6. Once the laser mark becomes too high to measure without a ladder (this will likely occur around 20 degrees), remove the laser pointer and set the primary axis to 90. Re-level the panel, re-attach the laser pointer (this time parallel to the panel), and record the new initial heights.
7. Begin reducing the primary axis angle (again in 5 degree increments) and record the resulting lengths.

The table below lists the results of this testing procedure (for the as-built primary axis):

TABLE 4 – PRIMARY AXIS POSITIONING ACCURACY

System Set Point (degree)	Mark Height (in)	Laser Distance From Wall (in)	Actual Angle (degree)	Misalignment Angle (degree)
0	-	251.0	-	-
5	22.000	247.5	5.08	-0.08
10	43.625	244.5	10.12	-0.12
15	65.750	241.5	15.23	-0.23
18	79.875	240.0	18.41	-0.41
75	24.000	264.0	75.19	-0.19
80	49.750	264.0	80.67	-0.67
85	73.375	264.0	85.53	-0.53
90	94.875	264.0	89.77	0.23

This table demonstrates that at angles over the entire range of motion for the primary axis, the rig can position and measure the angle of the panel to within an average of ± 0.250 degrees; in addition, all data points satisfy the ± 1.0 degree specification.

To verify the accuracy of the secondary axis, repeat this general procedure by setting the primary axis to 0 and panning across the wall (instead of climbing it). The table below lists the results of this testing for the as-built secondary axis:

TABLE 5 – SECONDARY AXIS POSITIONING ACCURACY

System Set Point (degree)	Mark Height (in)	Actual Angle (degree)	Misalignment Angle (degree)
-20	-95.125	-19.82	-0.18
-15	-74.000	-15.66	0.66
-10	-47.875	-10.28	0.28
-5	-24.125	-5.22	0.22
0	-	-	-
5	23.875	5.17	-0.17
10	47.750	10.25	-0.25
15	72.375	15.33	-0.33
20	99.125	20.58	-0.58

As demonstrated in Table 5, the secondary axis also meets specifications; the average misalignment of the panel about the secondary axis is only ± 0.044 degrees. This is well within the ± 1.0 degree target specification.

A brief test was also performed to verify that the majority of the misalignments measured above are bias errors (ie, that rig can return to the same position repeatedly). For this test, the secondary axis was set to zero and the primary axis set to five degrees. The location of the laser point on the wall at 5 degrees was recorded. Next, the primary axis was set to move to 15 degrees, to wait and finally to return to 5 degrees. The new location of the laser point was then recorded. This sequence was repeated for set points of 15, 25 and 35 degrees respectively (with the return position of the laser point marked after each sequence of motions). The result of these tests is shown on the subsequent page:

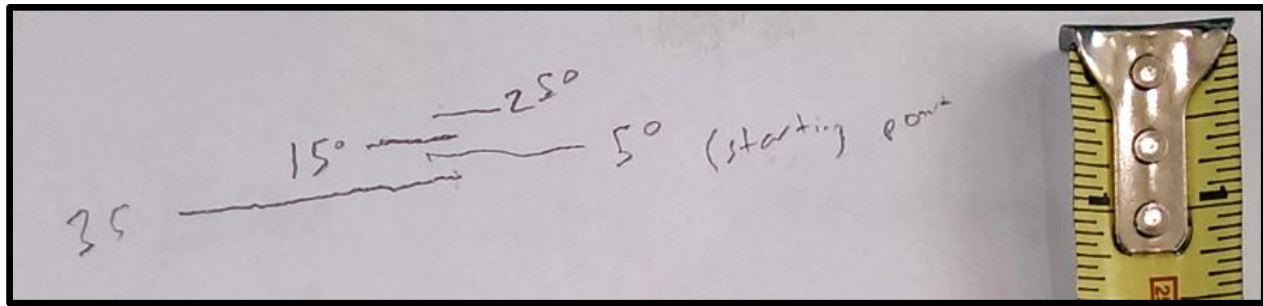


FIGURE 70 – PRIMARY AXIS REPEATABILITY TEST RESULTS

As shown, the rig reliably returned the laser point to within a range of about $\frac{3}{4}$ of an inch. At 22 feet from the pivot, this translates into a spread of roughly 0.16 degrees.

Finally, the response time of each axis was measured. The stated specifications for the rig require that it be able to traverse the entire range of each axis in under 2 minutes. As a result, a series of response tests were conducted to verify that the rig met this requirement.

The test to verify this specification proved simple enough: we positioned each axis at one extreme of its motion, and then sent it to the other extreme. The transit time of the axis was then recorded. Since the primary axis is subject to an imbalance (due to the weight of the panel) that particular test was run two times (once moving in each direction). The table below gives the results of this response time testing:

TABLE 6 – MAXIMUM SYSTEM RESPONSE TIMES FOR EACH AXIS

Test	Total Response Time (s)
Primary Axis Ascending (0 -> 90)	44.3
Primary Axis Descending (90 -> 0)	48.4
Secondary Axis (-20 -> 20)	39.5

These times indicate that both axes are capable of arriving at any set point (from any other) in less than 2 minutes (120 seconds).

8.2: COOLING SYSTEM TESTING

Tests related to the cooling system focused primarily on the rig's overall panel cooling capacity. It was critical that the trainer be capable of reliably and quickly cooling a panel of any size (within specifications) in the allotted time. In order to verify that the system was capable of cooling a 1m x 2m panel in less than 20 minutes, a test was conducted to characterize the thermal response of the panel as it was cooled by the onboard fans.

First, the panel was allowed to reach an uncooled steady state operating temperature. To accomplish this, the panel was left outside in the sun on a hot morning (around 75° F) and oriented to face the sun. The panel's temperature was then monitored using the four onboard digital temperature sensors. When these measured temperatures ceased to climb (around 120°F), the system was deemed 'ready-to-test.'

The even distribution of the temperature values across panel (as measured by the four sensors) verified our proof of concept tests which indicated that a few point temperature readings could accurately characterize the overall panel temperature. That said, we did observe that the baseline operating temperature varied as a function of weather conditions (cloudiness, windiness, air temperature, solar irradiance, etc.) and did not remain constant.

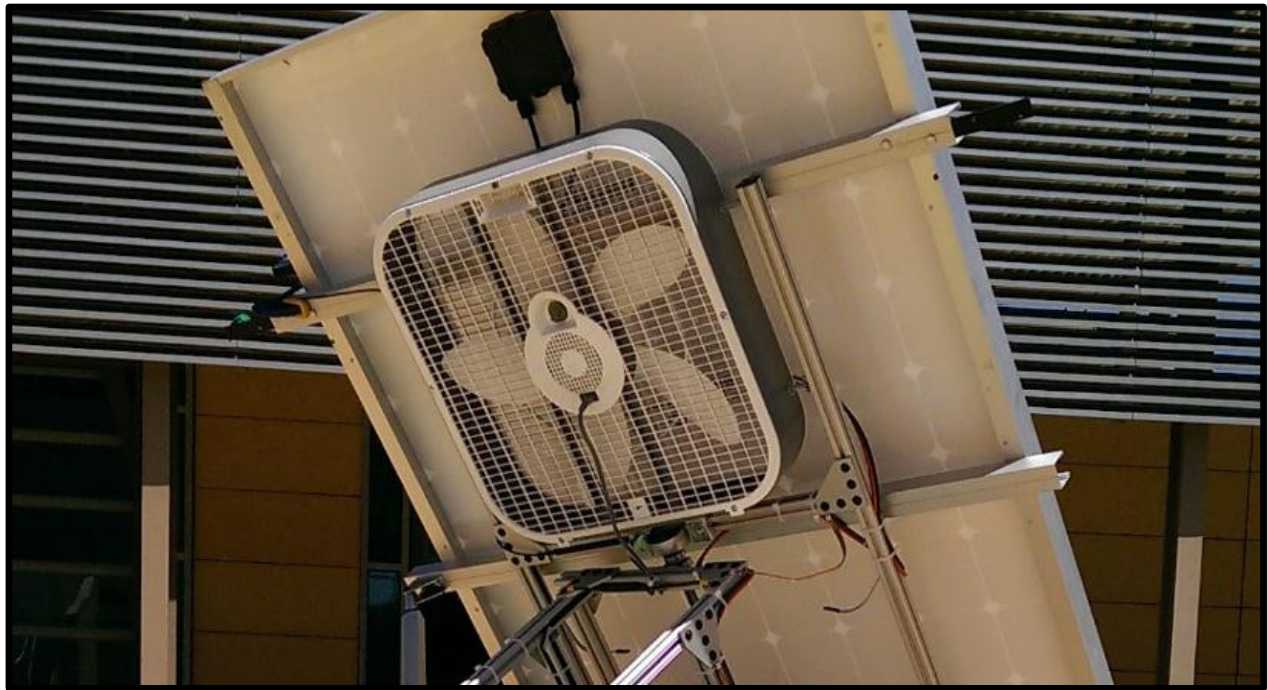


FIGURE 71 – PANEL COOLING FAN

The thermal response of the system was then measured by starting the onboard fans and recording the temperature measured by each sensor every 30 seconds. Since a large 1m x 2m panel was used, both fans were run at full power for the duration of the test. Test results are recorded in Table 8, below:

TABLE 7 - THERMAL SYSTEM RESPONSE TEST RESULTS

Time (min)	T1 (F)	T2 (F)	T3 (F)	T4 (F)	T_avg (F)
0.0	119	119	118	119	118.75
0.5	113	114	113	113	113.25
1.0	110	110	111	108	109.75
1.5	108	110	109	108	108.75
2.0	107	109	108	107	107.75
2.5	107	108	108	107	107.5
3.0	105	108	108	107	107.0
3.5	104	107	107	107	106.25
4.0	104	107	107	108	106.5
4.5	103	106	106	106	105.25
5.0	103	106	107	104	105.0
5.5	103	106	106	103	104.5
6.0	103	107	107	104	105.25
6.5	103	106	106	103	104.5
7.0	102	106	106	101	103.75
7.5	102	106	105	99	103.0
8.0	101	105	105	100	102.75
8.5	101	105	104	99	102.25
9.0	100	105	103	99	101.75
9.5	100	104	103	97	101.0
10.0	99	104	102	96	100.25
11.0	99	104	101	96	100.0
12.0	99	103	102	97	100.25

Over the course of the test, the average panel temperature dropped by about 19°F. This drop is about 1°F short of the desired minimum temperature drop. That said the local panel temperatures proved mostly uniform (within 5 degrees of each other) throughout the test. In addition, the panel reached its new steady state operating point within 10 minutes (roughly twice as fast as was required by the specifications). The following plot presents the system response graphically:

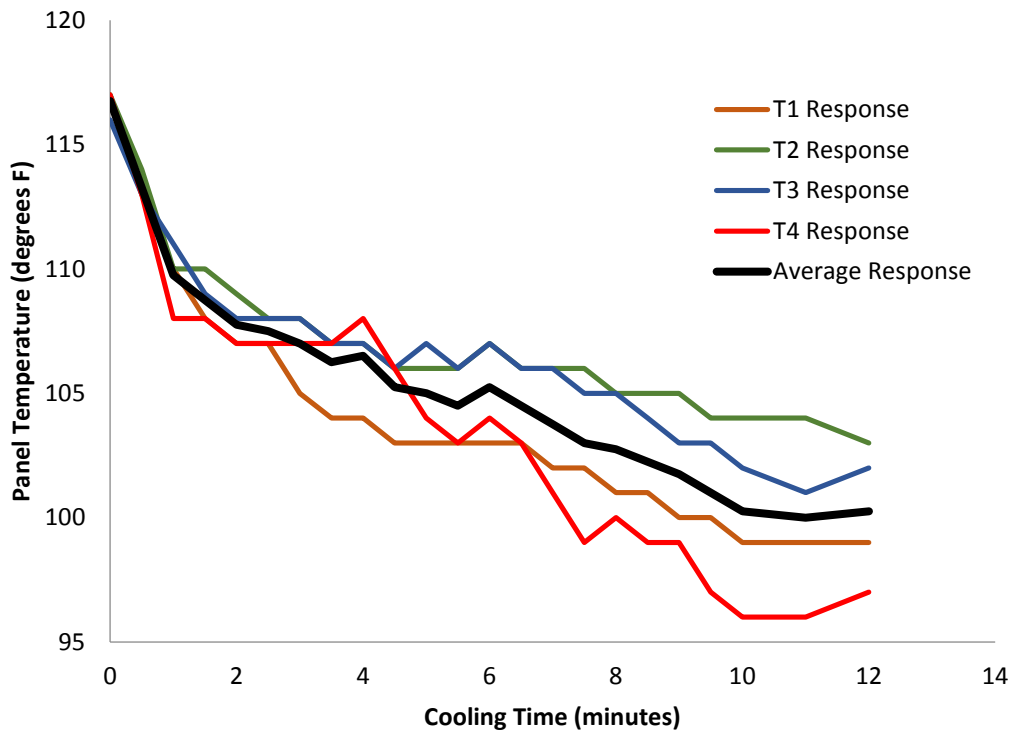


FIGURE 72 - THERMAL SYSTEM RESPONSE TEST RESULTS

This response curve generally matches expectations. First, the fans do indeed cool the panel. Also, the response of the panel temperature appears to be first order in nature (with a large initial drop in temperature and then a slow decrease in temperature afterwards). In addition, the local temperature responses are well distributed about the average (and lie within a limited and acceptable range).

That said the average temperature of the panel drops by only 18.5°F in 10 minutes (even with maximum speed fan cooling applied). This drop does not quite reach our design specification (which requires a 20°F decrease). However, it is worth noting that the rig's relative cooling performance will vary dramatically as a function of the initial baseline temperature. Even more importantly, the baseline operating temperature of the panel (with no cooling applied) may range between about 115°F and 135°F depending on the weather conditions.

The primary mode of heat transfer used to cool the panel is forced convection. Forced convection is a combination of heat transfer between the surface of the panel and the air flowing over it (or *diffusion*), and heat being carried away by the rapidly flowing air as it exits to the surroundings (called *advection*). The governing equation for this type of heat transfer (per unit of surface area) varies solely as a function of the temperature difference between the surrounding air and the panel. Thus, when the panel is much warmer than the surrounding air, there will be large amount of heat transfer (or panel cooling); when the panel is only slightly warmer than the

surrounding air, there will be a much smaller degree of heat transfer. This behavior means that the rate of cooling slows down as the temperature drops, which explains the shape of the first-order response curve.

Given this information, it is clear that the system will not be able to lower the panel from 100°F to 80°F nearly as easily as it can lower the panel temperature from 130°F to 110°F (even though both of these cases constitute a 20°F drop). In effect, the cooling system can only ever bring the panel temperature closer to some equilibrium value, regardless of what temperature it starts from. Given our tests to date, it remains our assessment that our system can cool the panel to around 100-105°F, no matter what the surrounding weather conditions are. If the panel started at 110 degrees, it is unlikely that our system could drop it to below 100, but if it started at 140 degrees, the system could lower the temperature to 105 degrees, a 35-degree temperature drop. To demonstrate this effect, we plotted the results from our proof-of-concept thermal testing in Fall 2013 against the actual thermal system response, including extrapolated data points consistent with a first order response. It should be noted that these tests were performed at different times of year, on different size panels, with different levels of airflow over the panel, and at different times of day. The plot is provided below:

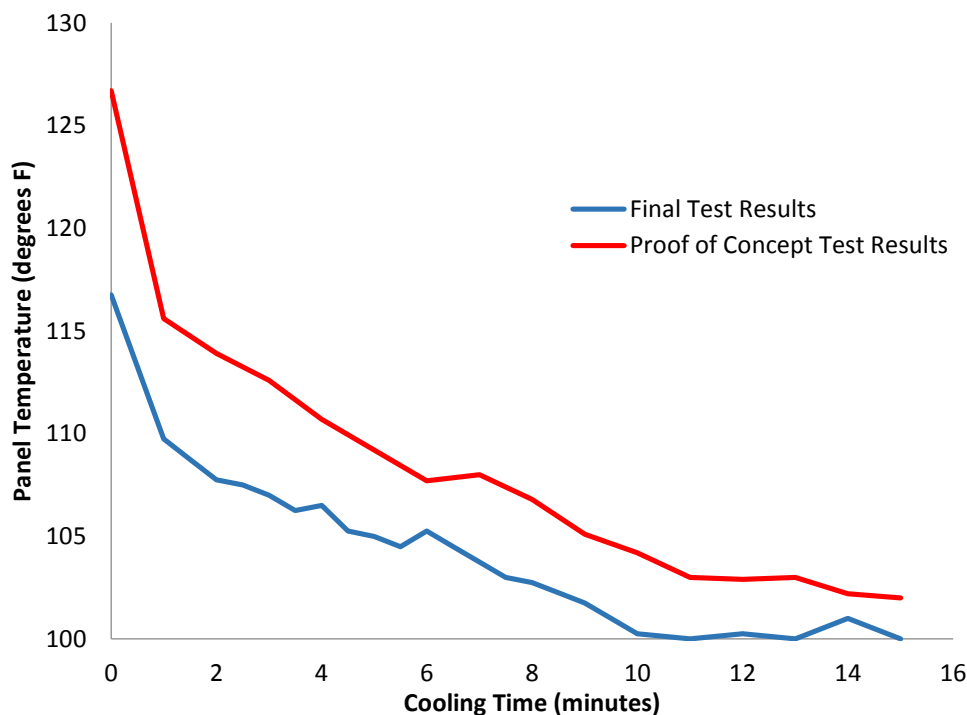


FIGURE 73 - COMPARISON OF PREVIOUS AND CURRENT THERMAL RESPONSE TESTS

Note that while the two panels start at temperatures that differ by 11°F, they end up only 2.5°F apart. Thus, in order to maximize the effect of the cooling system on the photovoltaic efficiency, we recommend that the panel be left in the sun for as long as possible before a laboratory is run so that it warms maximum possible temperature before testing. A higher “baseline operating temperature” will produce a larger temperature drop, which in turn will make the change in panel efficiency more dramatic.

8.3: ELECTRICAL SUBSYSTEM TESTING

Electrical subsystem testing was limited to a series of pass/fail tests that determined the effectiveness of various components in absolute terms. The criteria for these tests are listed below.

- Limit switch at upper extreme of elevation travel stops motion
- Limit switch at upper lower of elevation travel stops motion
- Limit switch at upper extreme of tracking travel stops motion
- Limit switch at upper lower of tracking travel stops motion
- Irradiance sensor responds to difference in light input
- Force sensing in elevation axis halts motion when force limit is reached
- Force sensing in tracking axis halts motion when force limit is reached
- Elevation axis motor driver powers elevation actuator
- Tracking axis motor driver powers tracking actuator
- Digital thermometer 1 responds to changes in temperature
- Digital thermometer 2 responds to changes in temperature
- Digital thermometer 3 responds to changes in temperature
- Digital thermometer 4 responds to changes in temperature
- LCD screen displays information correctly
- Keypad input is registered by microcontroller
- Cooling toggle switch powers cooling system on/off
- Manual/auto positioning mode toggles positioning state
- Power on/off switch toggles power on/off
- Emergency stop button kills power to the system
- Coordinate system transformation toggle switch changes coordinate system display

A summary of these test results may be found in the following section.

8.3: TESTING SUMMARY

A summary of all of our testing and validation efforts are given in a comprehensive table on the following page. This table lists each of our design specifications with units and tolerances (wherever applicable). Comments are also provided.

Spec. #	Description	Requirement (units)	Satisfied Requirement?	Comments
1	Max Panel Length	2000 mm	Yes	-
2	Min Panel Length	500 mm	Yes	-
3	Max Panel Width	1000 mm	Yes	-
4	Max Panel Depth	55 mm	Yes	-
5	Min Panel Depth	30 mm	Yes	-
6	Panel Replacement Time	10 minutes	Yes	-
7	Navigate Various Surfaces	Concrete, Grass, Dirt	Yes	-
8	Navigate Vertical Ledges	2in	Yes	-
9	Stable on Slopes During Transport	20 degrees	Yes	-
10	Easy to Maneuver During Transport	Able to Pivot in Own Footprint	Yes	-
11	'Collapsed' Width	32 inches	No	Frame Widened for Stability; Rig Still Passes Through Doors
12	'Collapsed' Height	80 inches (without 2000mm panel)	Yes	-
13	Panel Elevation Range	0-90 degrees, +/- 1.0 degree	Yes	Actual Range: 0-95 degrees
15	Elevation Adjustment Time	2 minutes	Yes	Actual Time: 42 seconds
16	Panel Azimuth Range	0-360 degrees, +/- 1.0 degree	Yes	Achieved via Drive System and Manual Rotation
18	Azimuth Adjustment Time	8 minutes	Yes	Actual Time: 39 seconds
19	Measure Panel Voltage	Full Range, +/- 0.5 volts	Yes	On Charge Controller
20	Measure Panel Current	Full Range, +/- 0.25 amp	Yes	On Charge Controller
21	Irradiance	Measure Ambient Irradiance	Yes	-
22	Sensor Readout Visibility	Legible in Direct Sunlight	Yes	-
23	Reduce Panel Operating Temp.	20 °F from Operational Baseline	Yes	15-30 °F Reduction (Depending on Weather)
24	Panel Temp Reduction Time	10 minutes	Yes	-
25	Panel Temp Sensor	Full Range, +/- 0.5 degrees	Yes	4 – sensor array
26	Student Work Surface	5 ft ² ; 30 inches off of ground	Yes	Also folds, with locking legs
27	Weatherproofing	IP53	Yes	-
28	Backup Control Systems	Manual Controls Included	Yes	On control Panel
29	Backup Sensing Systems	Backup readouts for all sensors	No	PV Voltage and Current have Analog Backups, (Irradiance and Temperature do not)
30	Mechanical Component Life	10 years	Assumed	Assuming Proper Care / Safe Use
31	Electrical Flexibility	Plugs used where possible	Yes	-
32	UV package/shielding	Parts specified for 2 years	Yes	-
33	Safety/Fail-safes	Limit Switches	Yes	Internal to Linear Actuators
34	Construction Cost	\$1500	Yes	Actual cost of \$1900 was reduced to \$1,300 by donations

9. SUGGESTED IMPROVEMENTS

9.1 STRUCTURE

Concern was expressed both before and during the critical design review that the structure itself might not support a large PV solar panel solidly enough to maintain the desired alignment tolerances (± 1 degree). Preliminary calculations seemed to indicate that bending in the shafts and frame members would not meaningfully contribute to misalignments in the panel. That said, several changes were made to the PV Trainer (post design review) in attempts to increase the rigidity of the structure. Even with those features in place however, the panel structure still flexes a considerable amount under heavy loads. While the positioning system successfully passed both the repeatability and accuracy tests *in calm conditions*, under moderate to heavy wind loads, tight position tolerances would likely not be maintained. This in mind, two relatively minor upgrades have been identified which would help to decrease the motion of the panel dramatically.

First, we recommend that any future PV solar trainers be constructed using a $\frac{3}{4}$ inch diameter primary axis shaft. The existing $\frac{5}{8}$ inch diameter shaft was selected primarily in an attempt to reduce the system's cost. While this shaft is strong enough to support the traveler and panel mount, it can be observed bending visibly under moderate wind loads. While this bending is not indicative of an impending structural failure, it does adversely affect the precise alignment of the panel. A $\frac{3}{4}$ inch diameter nitride steel shaft is not much more expensive than a $\frac{5}{8}$ inch one and would effectively prevent most of this flexing. It is worth mentioning that the secondary axis shaft does not need to be upsized (since it supports much smaller loads).

A second way in which the alignment accuracy of both axes may be improved is use of linear-actuator clevis pins with more precise tolerances. The 'linear actuator mounting brackets' that were specified are actually modified hinges. As a result, the supplied clevis pins have varied outside diameters which do not closely match the inside diameters of the linear actuator mounting holes. Clevis pins with tighter diameter tolerances would reduce the amount of undesired flexing in the structure.

9.2 CONTROL PANEL

The black acrylic specified for the control panel was selected because of its low cost. Unfortunately, this particular acrylic features an extremely glossy finish. As a result, it collects fingerprints easily and begins to appear 'dirty' after even mild use. As a result, we recommend that a less glossy acrylic be used in the future.

9.3 WIRING

The wires that connect the charge controllers to the battery terminals are made up of three 10 gauge wires connected in parallel. This decision was motivated by the fact that we had access to a large supply of both red and black 10 gauge wire. When it came time to wire the rig however, we discovered that we had underestimated the difficulty of routing 20 parallel 10 gauge wires into a single junction box. We would recommend that the power electronics on future rigs be wired together using as few wires as possible (in other words, using larger gauges as opposed to bundles).

9.4 ENCODER BRACKETS

The encoders that we purchased were purpose-built for a specific type of electric motor; as a consequence, we had to develop a complicated mounting structure to make them interface with our rig. We ended up ordering custom 3D printed parts to step down our shaft size, and needed to use many washers to space the encoder brackets properly. Also, since the Bonderson machine shop did not have an M2 tap set (which is what the data sheet called for) we had to use M3 screws to mount the encoders. This required removing some material from the encoder housing with a hand file such that the M3 socket heads could fit inside the housing. This arrangement was not ideal as the encoders could easily have been damaged while this modification was being made. In the future we would recommend investigating other encoder options that might interface more easily the rig's large 5/8 inch diameter shafts.

9.5 T-SLOT ASSEMBLY

Finally, we would like to reiterate our suggestion that the builders of any future Solar PV Trainers use caution when assembling the T-slot. Do take the time to verify that **all** of the necessary T-slot nuts have been placed into each channel before any fasteners/plates are tightened down. This becomes increasingly important the further one progresses in the assembly, because one may have to disassemble a large portion of the rig to access certain T-slot channels.

10. CONCLUSION:

Over the course of this one year long capstone design project, the Solar PV Trainer team has succeeded in meeting its goal of creating a new, highly improved mobile Solar PV Trainer. This completed device meets all of the design requirements that were listed by our sponsor, Professor Dolan, at the outset of the project: it can position a large or small solar panel about two independent axes, it can quickly and efficiently cool a large photovoltaic panel, it can support a wide assortment of inverters and charge controllers, and it features a sturdy and functional student work surface. More generally, we are proud to say that the rig is, as of the completion of this report, completely functional and ready to perform lab experiments whenever called upon to do so.

We hope that this new trainer (now complete) will be a valuable addition to the Cal Poly Electrical Engineering department's collection of laboratory test equipment. Regardless, we would like to say that we have learned an enormous amount while designing and building it.

We would like to thank Professor John Ridgely (for helping extensively with the electronics and microcontroller), Professor Dale Dolan (for being a helpful and responsive sponsor), and the various departments/firms that donated material to support our project.

We would especially like to thank Professor Sarah Harding, our advisor, for guiding us along the way (and for not crippling us with too many 'disappointed mom looks'). Feel better Sarah!

Finally, if Professor Dolan does indeed opt to build more of these trainers (as he has mentioned an interest in doing in the past) then we wish him, and all those helping him, the best of luck: godspeed sir.

WORKS CITED:

"IP RATINGS EXPLAINED." *IP Ratings*. OKW Enclosures Inc., 31 Jan. 2012. Web. 04 Dec. 2013.

Mount Pleasant Radio Telescope. Digital image. *Wikipedia*. Wikimedia Foundation, 8 Oct. 2012. Web. 4 Dec. 2013.

"Solar Tracker." *Wikipedia*. Wikimedia Foundation, 12 Jan. 2013. Web. 04 Dec. 2013.

<https://weatherspark.com/averages/31562/San-Luis-Obispo-California-United-States>

The Engineering Toolbox *Modulus of Rigidity for Various Metals*. 2/01/14.

[illegible]

APPENDIX B: COOLING TEST RAW DATA

TABLE 8 - RAW DATA FOR FORCED CONVECTIVE COOLING TEST

Time (min)	Rear Panel Temperature (back, on bottom), °F	Front Panel Temperature (front, on bottom), °F	Difference in Temperature (T1-T2), °F
0	127.7	126.7	1
1	119.9	115.6	4.3
2	117.9	113.9	4
3	115.8	112.6	3.2
4	114.0	110.7	3.3
5	112.7	109.2	3.5
6	111.9	107.7	4.2
7	111.2	108.0	3.2
8	110.6	106.8	3.8
9	110.8	106.1	4.7
10	109.0	105.2	3.8

TABLE 9 - RAW DATA FOR DIRECTED MIST COOLING TEST

Time (min)	Rear Panel Temperature (back, on bottom), °F	Front Panel Temperature (front, on bottom), °F	Difference in Temperature (T1-T2), °F
0	119.6	116.3	3.3
1	108.7	92.6	16.1
2	99.0	88.6	10.4
3	96.4	90.6	5.8
4	94.6	90.4	4.2
5	91.5	89.4	2.1
6	90.4	90.3	0.1

APPENDIX C: COORDINATE TRANSFORMATIONS

Variables:

$$\theta_0 = \text{Primary Axis Angle}$$

$$\theta_1 = \text{Secondary Axis Angle}$$

$$\theta_{AZIMUTH} = \text{Azimuth Angle}$$

$$\theta_{ELEVATION} = \text{Elevation Angle}$$

To convert from Primary/Secondary angles to Azimuth/Elevation angles:

$$\theta_{AZIMUTH} = \tan^{-1} \left[\frac{-\tan(\theta_1)}{\cos(\theta_0)} \right]$$

$$\theta_{ELEVATION} = \frac{\pi}{2} - \cos^{-1}[\sin(\theta_0) \cos(\theta_1)]$$

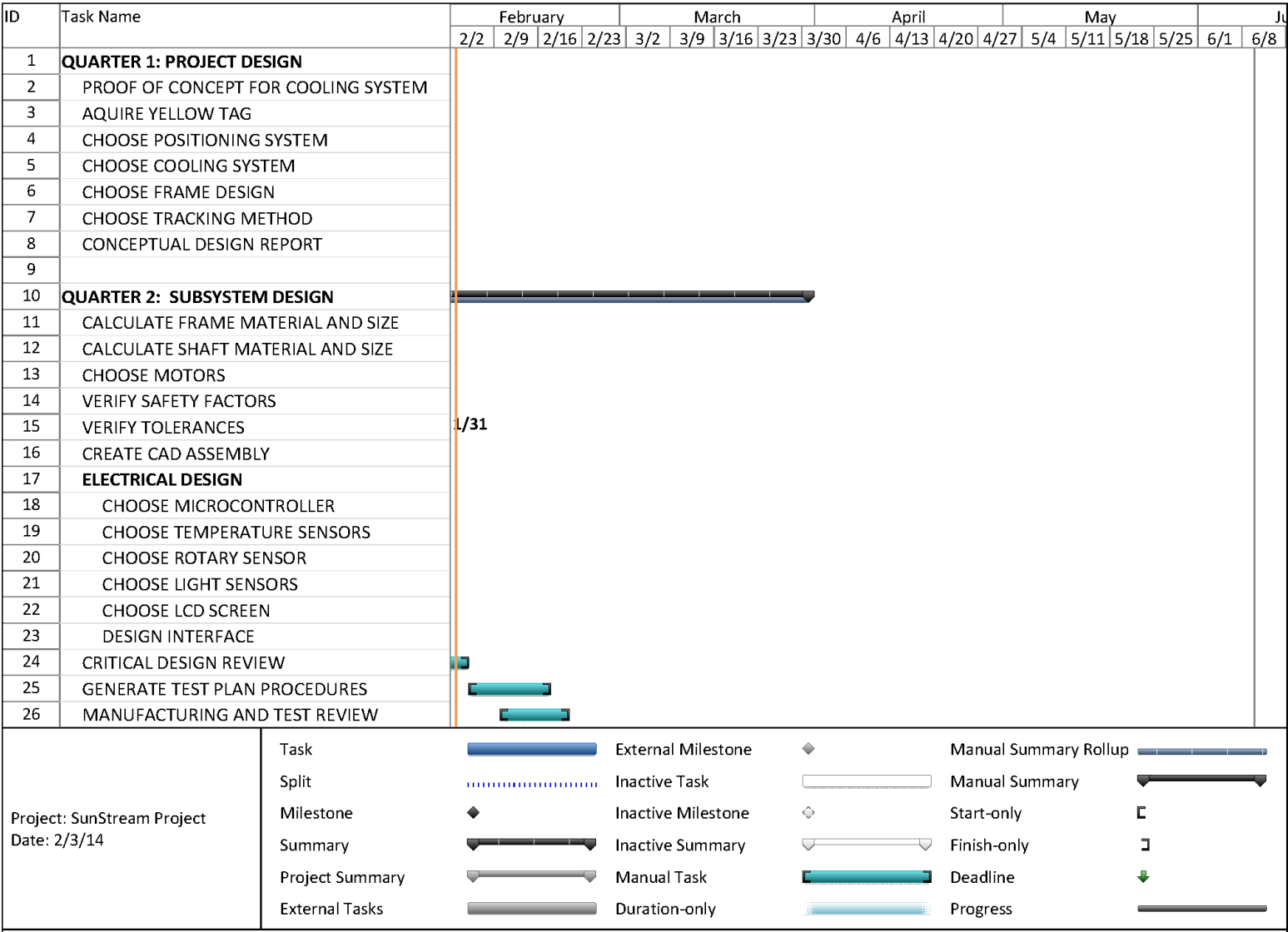
To convert from Azimuth/Elevation angles to Primary/Secondary angles:

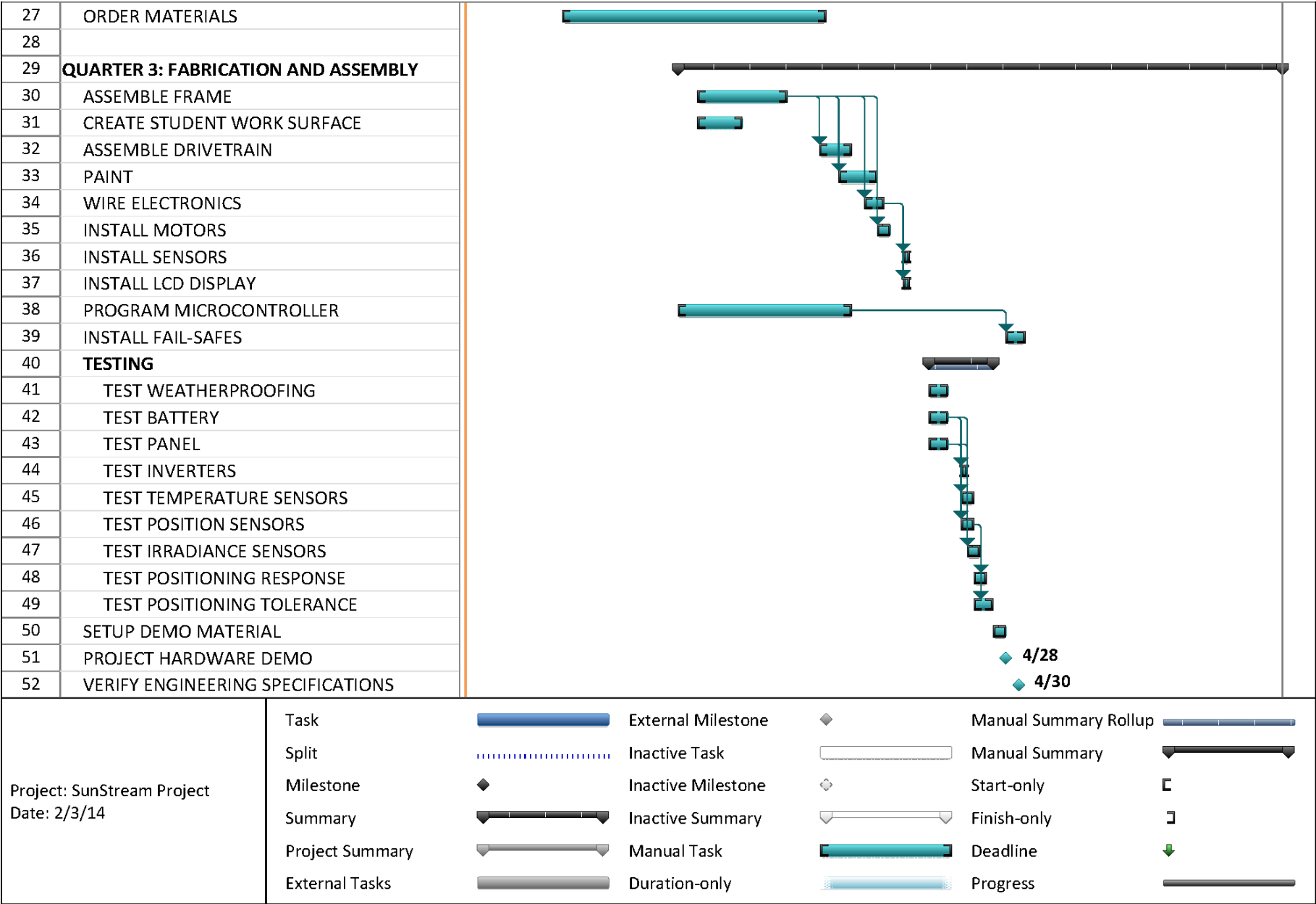
$$\theta_0 = \tan^{-1} \left[\frac{\cos(\pi/2 - \theta_{EL})}{\sin(\pi/2 - \theta_{EL}) \cos(\theta_{AZ})} \right]$$

$$\theta_1 = \sin^{-1}[-\sin(\pi/2 - \theta_{EL}) \sin(\theta_{AZ})]$$

*Note: relations derived using Denavit–Hartenberg coordinate transformations

APPENDIX D: GANTT CHART





ID	Task Name	February					March					April					May					June	
		2/2	2/9	2/16	2/23	3/2	3/9	3/16	3/23	3/30	4/6	4/13	4/20	4/27	5/4	5/11	5/18	5/25	6/1	6/8			
53	SENIOR EXPO																						
54	DESIGN REPORT																						
55	CREATE DESIGN REPORT																						
56	SUBMIT TO SARAH																						
57	SUBMIT TO SPONSOR																						
58	SUBMIT TO LIBRARY																						
Project: SunStream Project Date: 2/3/14		Task		External Milestone		Manual Summary Rollup																	
		Split		Inactive Task		Manual Summary																	
		Milestone		Inactive Milestone		Start-only																	
		Summary		Inactive Summary		Finish-only																	
		Project Summary		Manual Task		Deadline																	
		External Tasks		Duration-only		Progress																	

APPENDIX E: BILL OF MATERIALS

Table 1 – Bill of Materials

CATEGORY	ITEM DESCRIPTION (Part #)	QTY	COST/UNIT	TOTAL COST
Overhead	Shipping etc.	1	100	\$100.00
T -Slot Framing	T-Slot, 10-Series, 2x2, (180 inches required) (214)	2	Donated by T-Slots	
	T-Slot, 10-Series, 1x2, (123 inches required) (213)	2	Donated by T-Slots	
	T-Slot, 10-Series, 1x1, (436 inches required) (212)	4	Donated by T-Slots	
	4 Hole Corner Bracket (222)	8	Donated by T-Slots	
	4 Hole Corner Bracket (223)	4	Donated by T-Slots	
	4 Hole Corner Bracket (223)	2	\$5.58	\$11.16
	8 Hole Corner Bracket (225)	6	Donated by T-Slots	
	2 Hole Corner Bracket (224)	14	Donated by T-Slots	
	2 Hole Corner Bracket (224)	2	\$3.98	\$7.96
	12 Hole Corner Bracket (217)	2	Donated by T-Slots	
	4 Hole Joining Plate (226)	1	Donated by T-Slots	
	4 Hole Joining Plate (226)	2	\$6.20	\$12.40
	8 Hole Joining Plate (227)	1	Donated by T-Slots	
	4 Hole 90 Joining Plate (229)	2	Donated by T-Slots	
	5 Hole 90 Joining Plate (220)	16	Donated by T-Slots	
	5 Hole Tee Joining Plate (219)	6	Donated by T-Slots	
	12 Hole Tee Joining Plate (218)	4	Donated by T-Slots	
	12 In Double Angle Brace (221)	2	\$18.49	\$36.98
	12 In Single Angle Brace (228)	2	\$17.15	\$32.30
	2" x 2" End Cap Black	4	Donated by T-Slots	
	1" x 1" End Cap Black	4	Donated by T-Slots	
	Black T-Slot Cover (188 inches total) (211)	3	Donated by T-Slots	
	Compact Head Fasteners (216)	163	Donated by T-Slots	
	Standard Fasteners (215)	200	Donated by T-Slots	

Shafts & Bearings	Shaft Mounts (311)	4	\$18.00	\$72.00
	Shafts (310)	2	\$16.00	\$32.00
	Bearing Blocks (312)	4	\$37.00	\$148.00
	Bronze Thrust Bearing (315)	2	\$2.00	\$4.00
	Shaft Collar (316)	1	\$5.48	\$5.48
	Shaft Mount Spacer	2	\$4.00	\$8.00
Portability	Casters (213)	4	\$25.00	\$100.00
	Caster Mounting Plates (232)	1	\$30.00	\$30.00
Electronics Housing	Electrical Box (290)	1	\$40.00	\$40.00
	Acrylic for Control Panel (400)	1	\$11.00	\$11.00
	Battery Housing (621)	1	\$14.80	\$14.80
	M5 Mounting Screws	2	\$2.20	\$4.40
	M5 Lock Nuts	2	\$0.41	\$0.82
	M5 Washers	2	\$0.47	\$0.94
Motion Control	Linear Actuator (313)	2	\$84.00	\$168.00
	Linear Actuator Mounts (314)	4	\$4.00	\$16.00
Counterweight	Steel Block (261)	Donated by IME Department		
Work Surface/Inverter Mount	Plywood (4 ft x 8 ft)	1	\$32.00	\$32.00
	Piano Hinge (30 in) (243)	1	\$8.00	\$8.00
	Leg Hinges (package of 2) (246)	1	\$3.50	\$3.50
	Mounting Screws	1	\$5.50	\$5.50
	1/4 in Acrylic (16 ft ²)	Donated by ME Department		
	1 in ‘L’ Extrusion	3	\$9.00	\$27.00
	1.5 in ‘L’ Extrusion	2	\$29.00	\$58.00
	Acrylic Tape	2	\$11.75	\$23.50
	Black Spray Paint	1	\$2.50	\$2.50
Sensor Mounts	Irradiance Sensor Brackets (251)	1	\$2.40	\$2.40
	Plasti-Dip	1	\$6.51	\$6.51
	3D Print Encoder Adaptor (518)	3	\$15.00	\$45.00
	3mm Encoder Screws	2	\$3.00	\$6.00
Cooling	AC Box Fans (252)	2	\$18.00	\$36.00

	1/4" - 20 pan head machine screw, 2" long	4	\$0.73	\$2.92
	18-8 Stainless Steel Type B Flat Washer	8	\$0.62	\$4.96
	Black Luster-Coated Steel Wing Nut	4	\$1.05	\$4.20
	Zinc-plated 1/4" - 20 Nut	8	\$0.12	\$0.96
Electronics (Donated)	Charge Controllers	Donated by Sponsor		
	Solar Inverters	Donated by Sponsor		
	Irradiance Sensor (512)	Donated by Sponsor		
	Battery	Donated by Sponsor		
	Wire - 10 gauge	Donated by Sponsor		
	Wire - 0/2	Donated by Sponsor		
	Wire - 2 gauge	Donated by Sponsor		
	Large Banana Jacks	Donated by Sponsor		
	Spray Paint (for Banana Jacks)	Donated by Lawrence S.		
Electronics (Donated.)	Microcontroller (ME 405) (510)	Donated by ME Department		
	Resistors	Donated by ME Department		
	Capacitors	Donated by ME Department		
	Inductors	Donated by ME Department		
Sensors	DS18B20 (Temp Sensor) (514)	4	\$4.25	\$17.00
	Thermal Grease	1	\$10.33	\$10.33
	Absolute Encoder (511)	2	\$25.68	\$51.36
Control Panel	Small Toggle Switches (413)	3	\$2.00	\$6.00
	Power Switches (410)	2	\$3.00	\$6.00
	E-Stop Switch (414)	1	\$5.40	\$5.40
	LCD Screen (411)	1	\$25.90	\$25.90
	Keypad (415)	1	\$6.22	\$6.22
	Red/Green LED	2	\$0.50	\$1.00
	DPDT Relays (416)	2	\$4.00	\$8.00
	Breadboard (516)	1	\$5.00	\$5.00
	Light Switches	1	\$12.00	\$12.00
	Electrical Outlet (412)	Donated by Matthew M.		
Power Supply	Varistor (515)	1	\$4.52	\$4.52
	Fuse (517)	1	\$0.64	\$2.56

Wiring	Open Junction Box Cover (612)	1	\$6.00	\$6.00
	Flat Junction Box Cover (611)	2	\$4.00	\$8.00
	Outdoor Junction Box (610)	3	\$5.00	\$15.00
	# 10 Ring Connectors (613)	1	\$2.00	\$2.00
	1/4 in Ring Connectors (614)	1	\$2.00	\$2.00
	3/8 in Ring Connectors (615)	1	\$2.00	\$2.00
	1/4 in Wire Clips (616)	2	\$2.00	\$4.00
	3/8 in Wire Clips (617)	2	\$2.00	\$4.00
	Extension/Power Cords (620)	2	\$9.00	\$18.00
	1/2 in Tube Adaptors	5	\$1.00	\$5.00
	3/4 in Tube Adaptors	8	\$2.00	\$2.00
	Phone Cable	Donated by Matthew M.		
	Phone Cable Adaptors (513)	2	\$4.00	\$8.00
	Grounding Strap (619)	2	\$9.02	\$18.04
	Zip Ties	2	\$5.00	\$10.00
Hardware	1/4 in Carriage Bolts	40	0.19	7.60
	1/4 in Bolts	16	0.11	1.76
	1/4 in Washers	65	0.06	3.90
	1/4 in Nuts	50	0.09	4.50
	Long 1/4 in Nylon Spacers	10	0.35	3.50
	1/4 in Nylon Spacers	8	0.55	1.90
			Total Cost	\$1,412.96

Table 2 – Part Sources

Category	Item Description	Part Number	Supplier
T-Slot Framing	T-Slot, 10-Series, 2x2, (112 inches)	B00BMUWIH0	www.amazon.com
	T-Slot, 10-Series, 1x2, (123 inches)	B001F0K4I2	www.amazon.com
	T-Slot, 10-Series, 1x1, (414 inches)	B001F0F112	www.amazon.com
	T-Slot, 10-Series, 1x1, (414 inches)	B001F0I3BC	www.amazon.com
	Flat Right Angle	47065T177	www.mcmaster.com
	Large Single Right Angle	B001IA12Z8	www.amazon.com
	Small Single Right Angle	B001IA4PRA	www.amazon.com
	Small Double Right Angle	B001IA4PSE	www.amazon.com
	Large Double Right Angle	B001IA4PSO	www.amazon.com
	Large T-Plate	B001IA6QIQ	www.amazon.com
	Small T-Plate	B001IA134S	www.amazon.com
	Large Flat Plate	47065T174	www.mcmaster.com
	Small Flat Plate	47065T211	www.mcmaster.com
	Precut Cross Brace	47065T189	www.mcmaster.com
	T-Slot Wire Covers	47065T93	www.mcmaster.com
	Compact Head Fasteners	47065T139	www.mcmaster.com
	Standard Fasteners	47065T142	www.mcmaster.com
	Counter weight	N/A	Donated from IME Dep

Shafts & Bearings	Shaft Mounts	a13051600ux0636	www.amazon.com
	Shafts	5936K83	www.mcmaster.com
	Bearing Blocks	5913K62	www.mcmaster.com
	Bronze thrust bearing/spacer	5906K545	www.mcmaster.com
Portability	Casters	46819	www.harborfreight.com
	Caster Mounting Plates	9246K13	www.mcmaster.com
Electronics Housing	Electrical Box	B005T57OLY	www.amazon.com
	Acrylic for Control Panel	N/A	Spare from ME
	M5 Mounting Screws	203540147	www.homedepot.com
	M5 Lock Nuts	202209675	www.homedepot.com
	M5 Washers	202209686	www.homedepot.com
Motion Control	Linear Actuator	LIN-ACT1-12	www.amazon.com
	Linear Actuator Mounts	2355 Bottom of Form	www.polulu.com
Work Surface/Inverter Mount	Plywood	1502104	www.homedepot.com
	Hinges	202558075	www.homedepot.com
	Mounting Screws	90184A513	www.mcmaster.com
	Plasti Dip	21801	www.homedepot.com
Sensor Mounts	Irradiance/Tracking Sensor Mount	1394A37	www.mcmaster.com

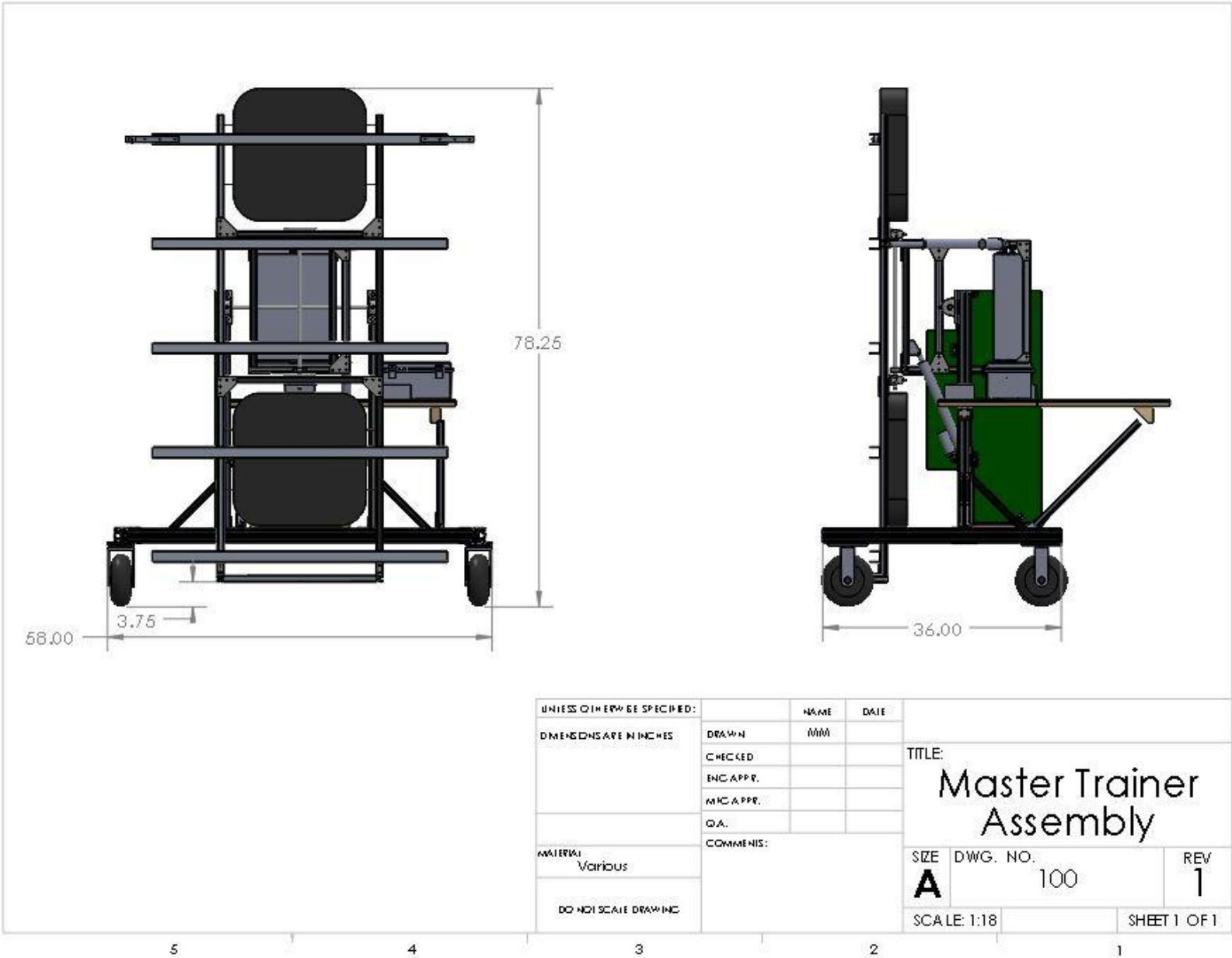
Cooling	AC Box Fans		Target
	1/4" - 20 pan head machine screw, 2" long	91735A548	www.mcmaster.com
	18-8 Stainless Steel Type B Flat Washer	92217A479	www.mcmaster.com
	Black Luster-Coated Steel Wing Nut	98671A210	www.mcmaster.com
	Zinc-plated 1/4" - 20 Nut	202704387	www.homedepot.com
Electronics (Donated)	Charge Controller	N/A	Dale Dolan
	Solar Inverter	N/A	Dale Dolan
	Irradiance Sensor	N/A	Dale Dolan
	Battery	N/A	Dale Dolan
	Wire (Various Gauges)	N/A	Dale Dolan
	Large Banana Jacks	N/A	Dale Dolan
	Spray Paint (for Banana Jacks)	N/A	Lawrence Smith
Electronics (Donated.)	Microcontroller (ME 405 Board)	N/A	John Ridgely
	Resistors	N/A	John Ridgely
	Capacitors	N/A	John Ridgely
	Inductors	N/A	John Ridgely
	LEDs	N/A	John Ridgely
Sensors	DS18B20 (Temperature Sensor)	DS18B20	www.sparkfun.com

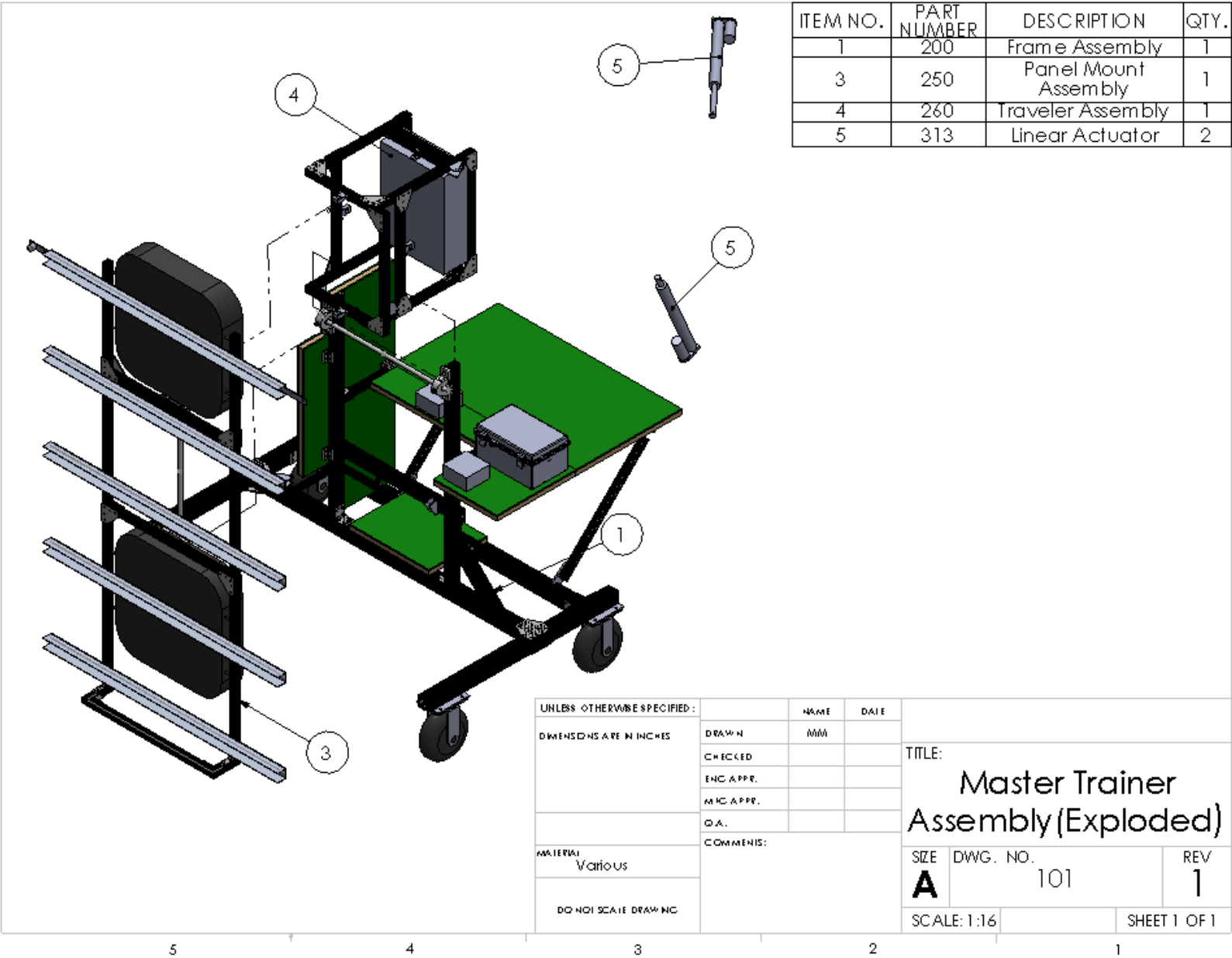
	Thermal Grease	S606N-30	www.digikey.com
	Absolute Encoder	AEAT-6010-A06	www.digikey.com
Control Panel	Toggle Switches	Various	www.radioshack.com
	Kill Switch	B0094GM004	www.amazon.com
	LCD Screen	NHD-0420D3Z-NSW-BBW-V3	www.digikey.com
	Keypad	a12010900ux0122	www.amazon.com
	Outdoor Outlet Box	202284496	www.homedepot.com
	Electrical Outlet	R52-05320-00W	www.homedepot.com
Wire Organization	Zip Ties	CT8NT50S-M	www.digikey.com
	Connectors	OSTTC022162	www.digikey.com
	Wire Conduit	12007-025	www.homedepot.com

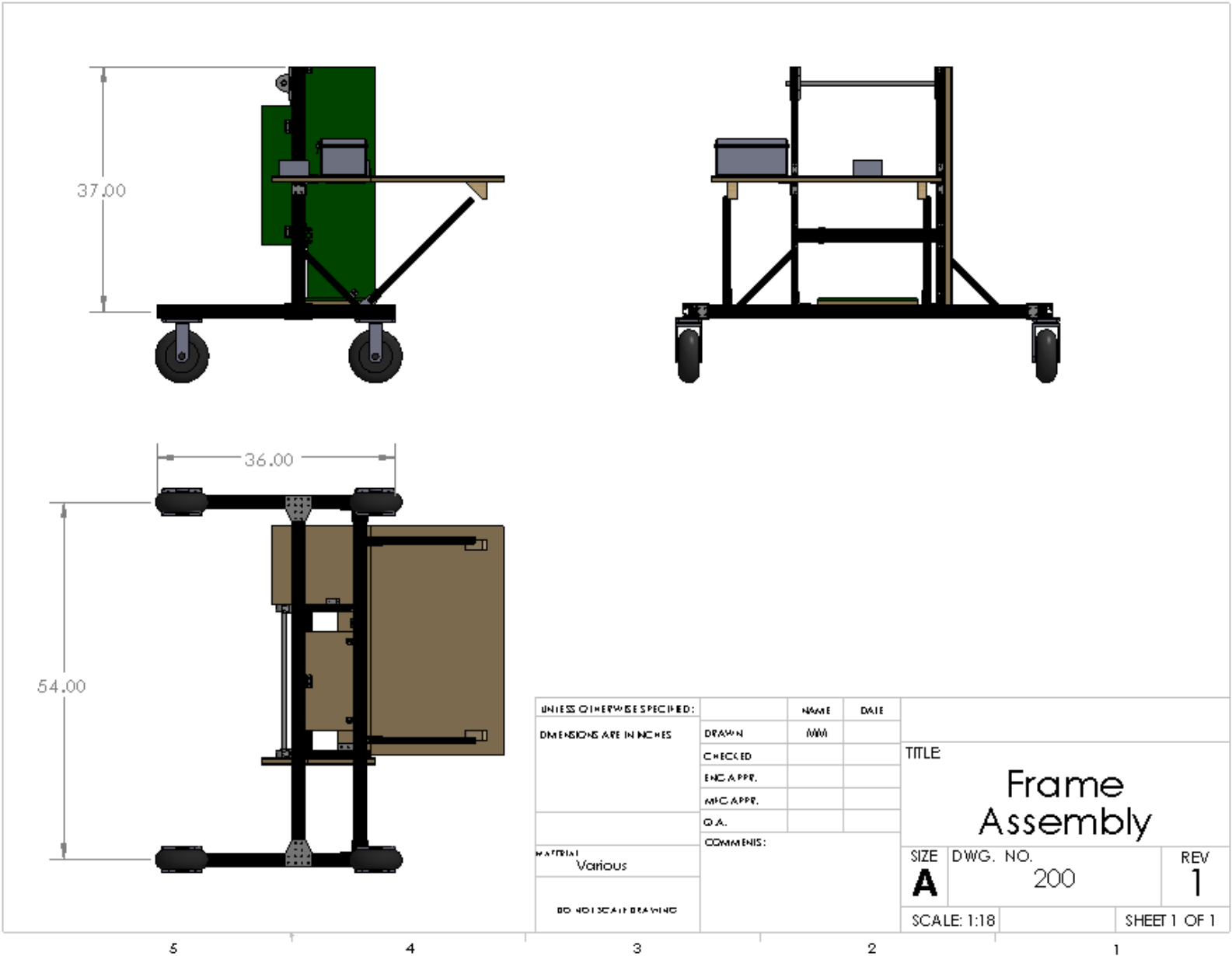
APPENDIX F: DRAWING LIST AND PART DRAWINGS

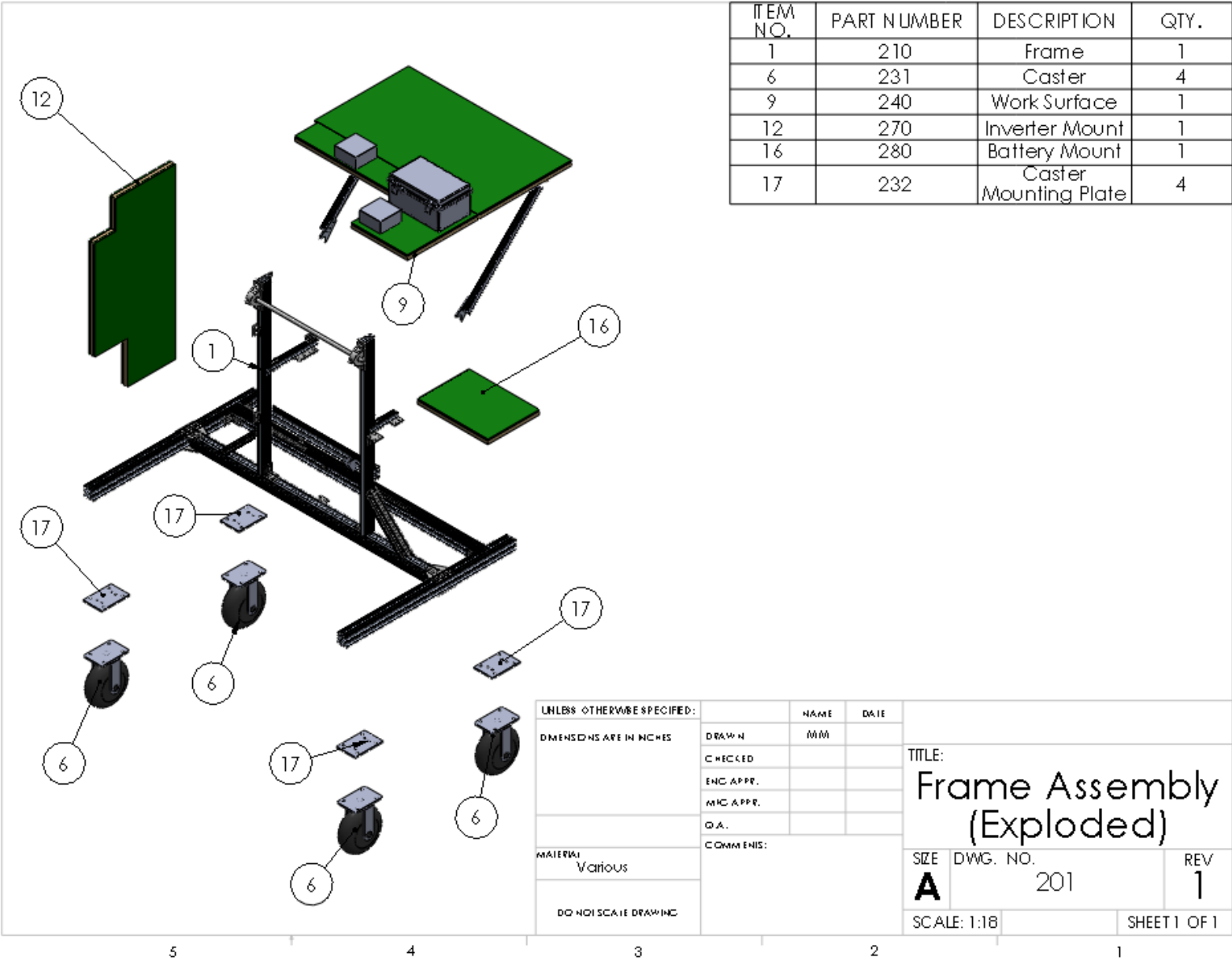
100 - Top Level Assembly**101 - Exploded Top Level Assembly****200 - Frame Assembly****201 - Exploded Frame Assembly****210 - Frame Cut List****210A - Frame (T-Slot) Assembly (1)****210B - Frame (T-Slot) Assembly (2)****210C - Frame (T-Slot) Assembly (3)****211 - T-Slot Wire Cover Data Sheet****212 - 1010 T-Slot 8020 Aluminum Data Sheet****213 - 1020 T-Slot 8020 Aluminum Data Sheet****214 - 2020 T-Slot 8020 Aluminum Data Sheet****215 - T-Slot Standard Hammerhead Bolt Data Sheet****216 - T-Slot Low Profile Hammerhead Bolt Data Sheet****217 - T-Slot 12-Hole 90 Degree Brace Data Sheet****218 - T-Slot 12-Hole T Gusset Plate Data Sheet****219 - T-Slot 5-Hole T Gusset Plate Data Sheet****220 - T-Slot 5-Hole 90 Degree Gusset Plate Data Sheet****221 - T-Slot 12 inch Right Angle Brace Data Sheet (Double)****222 - T-Slot 4-Hole Extended 90 Degree Brace Data Sheet****223 - T-Slot 4-Hole Short 90 Degree Brace Data Sheet****224 - T-Slot 2-Hole 90 Degree Brace Data Sheet****225 - T-Slot 8-Hole 90 Degree Brace Data Sheet****226 - T-Slot 4-Hole Flat Brace Data Sheet****227 - T-Slot 8-Hole Flat Brace Data Sheet****228 - T-Slot 12 inch Right Angle Brace Data Sheet (Single)****229 - T-Slot 4-Hole 90 Degree Gusset Plate Data Sheet****230 - Caster Assembly****231 - Caster Data Sheet****232 - Caster Mounting Plate Drawing****240 - Work Surface Assembly****241 - Hinged Work Surface Drawing****241A - Hinged Work Surface Drawing (Acrylic)****242 - Fixed Work Surface Drawing****242A - Fixed Work Surface Drawing (Acrylic)****243 - Piano Hinge Data Sheet****244 - Leg Brace Drawing****245 - Leg Feet Drawing****246 - Leg Hinge Data Sheet****250 - Rail (T-Slot) Cut List****250A - Panel Mount Assembly (1)****250B - Panel Mount Assembly (2)****251 - Sensor Mount Bracket****252 - Cooling Fan****253 - Panel Mounting Rail Data Sheet****260 - Traveler Cut List****260A - Traveler Assembly (1)**

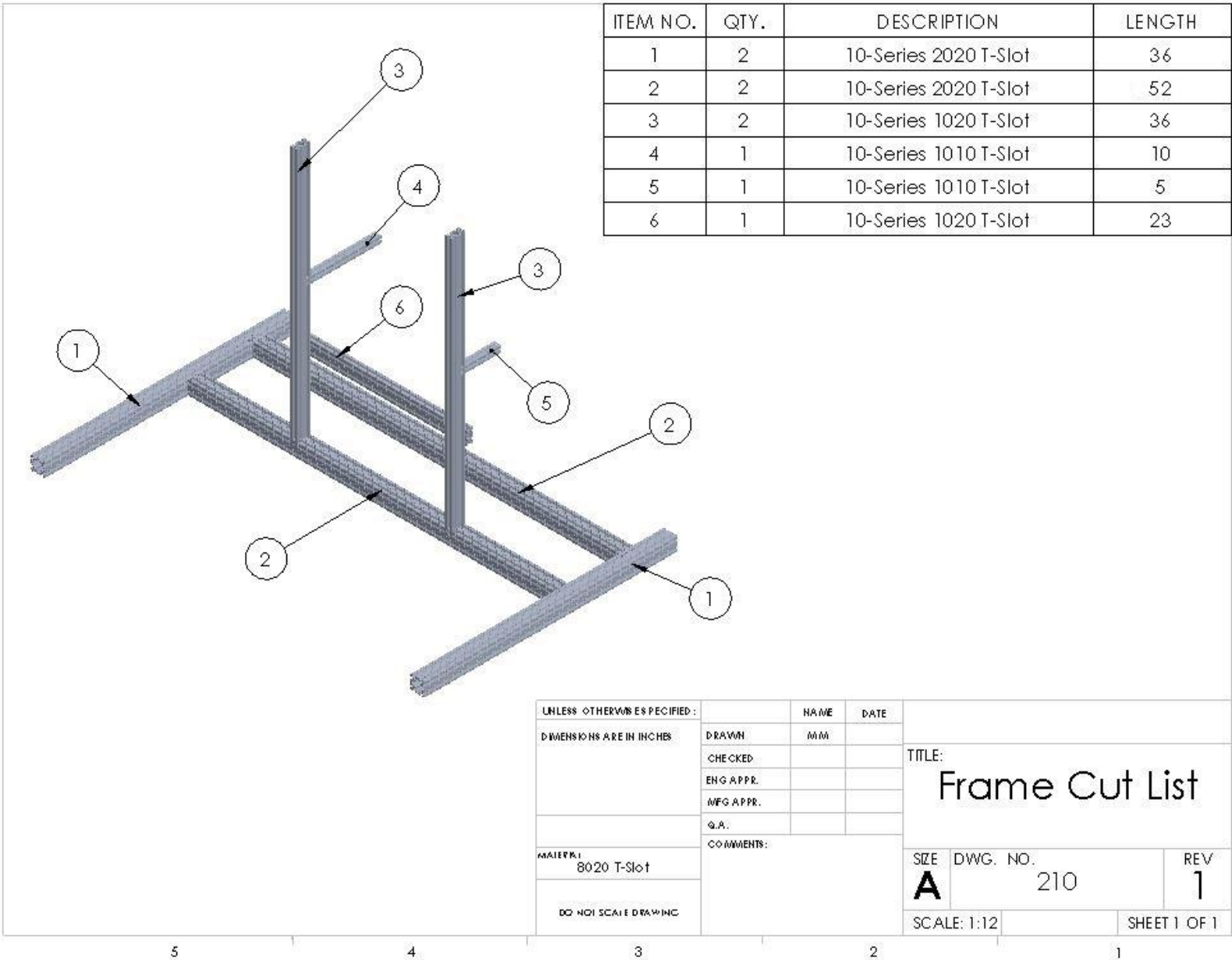
- 260B** - Traveler Assembly (2)
- 260C** - Traveler Assembly (3)
- 261** - Counterweight Drawing
- 270** - Inverter Mount Drawing
- 270A** - Inverter Mount Drawing (Acrylic)
- 280** - Battery Mount Drawing
- 280A** - Battery Mount Drawing (Acrylic)
- 290** - Electronics Box Data Sheet
- 300** - Drive Assembly
 - 310** - Shaft Data Sheet
 - 311** - Mounting Brackets for Drive Shaft
 - 312** - Pillow Bearing Data Sheet
 - 313** - Actuator Data Sheet
 - 314** - Actuator Mount Data Sheet
 - 315** - Thrust Bearing
 - 316** - Shaft Collar
- 400** - Control Panel Drawing
 - 410** - Power Switch Data Sheet
 - 411** - LCD Screen Data Sheet
 - 412** - Wall Socket Data Sheet
 - 413** - Toggle Switch Data Sheet
 - 414** - Emergency Stop Button Data Sheet
 - 415** - Keypad Data Sheet
 - 416** - Relay Data Sheet
- 500** - Microcontroller Wiring Diagram
 - 510** - ME 405 Board Data Sheet
 - 511** - Absolute Encoder Data Sheet
 - 512** - Irradiance Sensor Data Sheet
 - 513** - Phone Cable Connector Data Sheet
 - 514** - DS18B20 Temperature Data Sheet
 - 515** - Varistor Data Sheet
 - 516** - Breadboard Data Sheet
 - 517** - Fuse Data Sheet
 - 518** - 3D-Printed Encoder Adaptor Drawing
 - 519** - Encoder Mounting Bracket Drawing (1)
 - 520** - Encoder Mounting Bracket Drawing (2)
- 600** - Wiring Assembly
 - 610** - Junction Box Data Sheet
 - 611** - Junction Box Cover Data Sheet
 - 612** - Weatherproof Junction Box Cover Data Sheet
 - 613** - Ring Connector Data Sheet (#10)
 - 614** - Ring Connector Data Sheet ($\frac{1}{4}$ Inch)
 - 615** - Ring Connector Data Sheet ($\frac{3}{8}$ Inch)
 - 616** - Cable Holder Data Sheet ($\frac{1}{4}$ Inch)
 - 617** - Cable Holder Data Sheet ($\frac{3}{8}$ Inch)
 - 618** - Grounding Bar Data Sheet
 - 619** - Grounding Strap Data Sheet
 - 620** - Replacement Power Cord Data Sheet
 - 621** - Battery Box Data Sheet

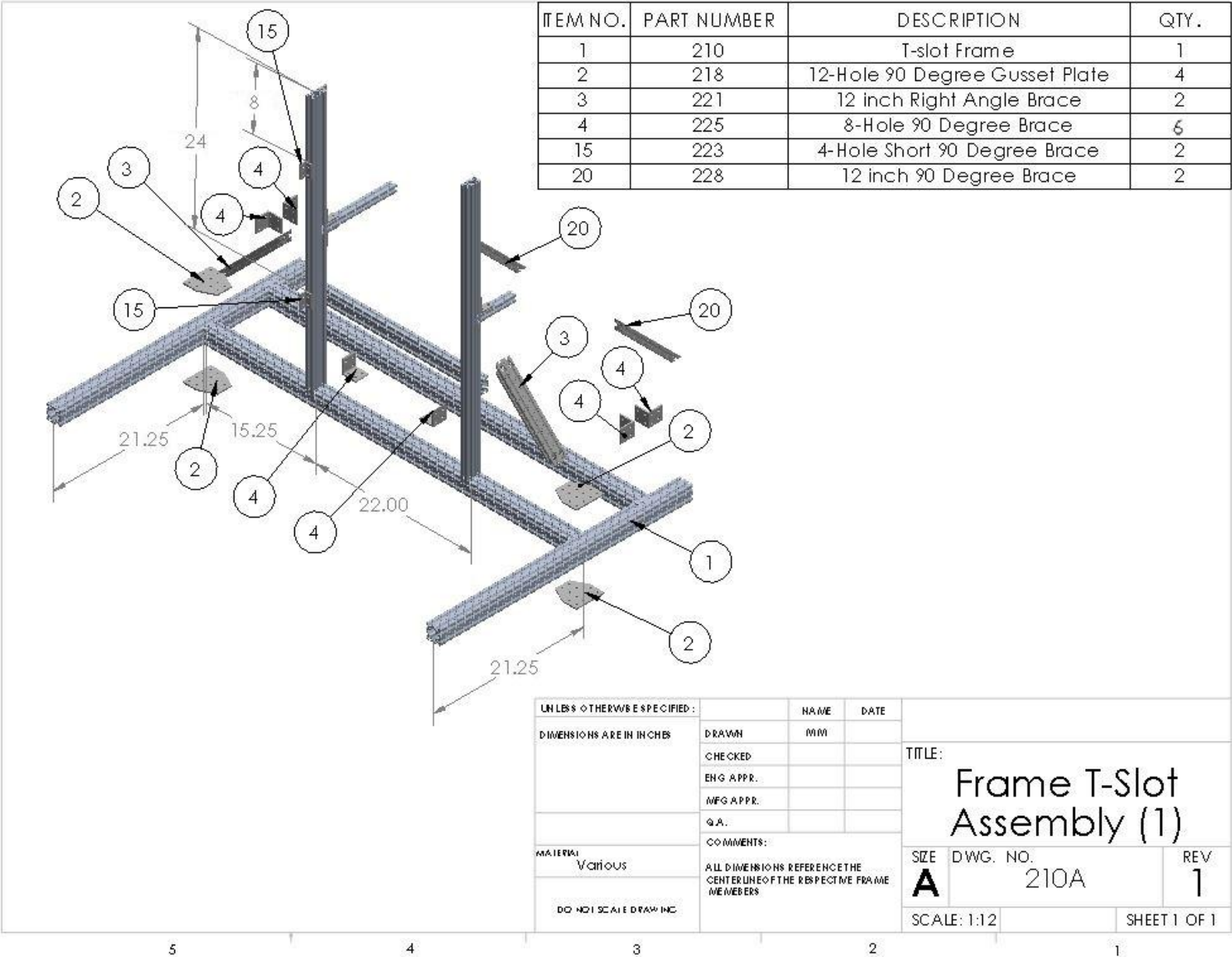


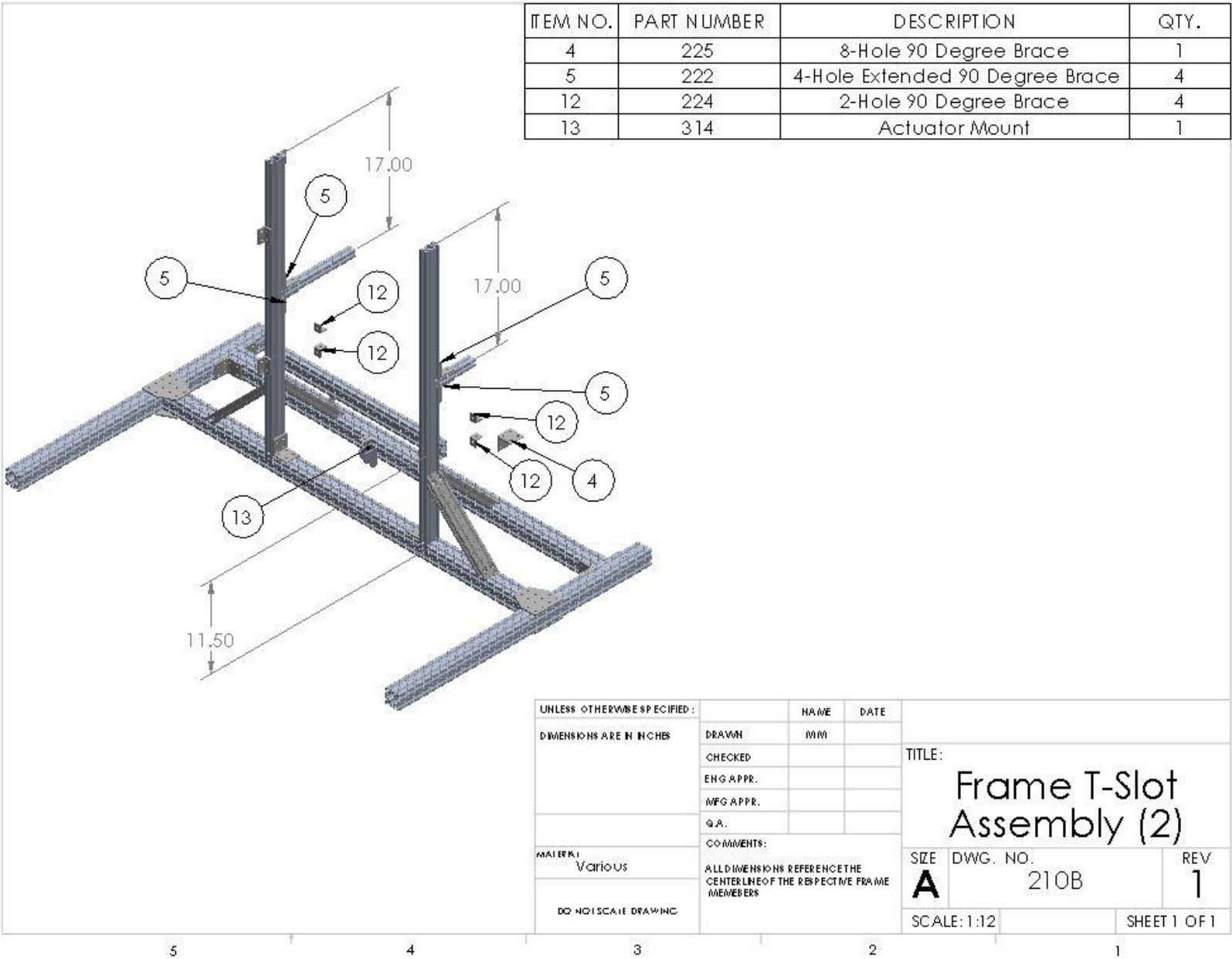


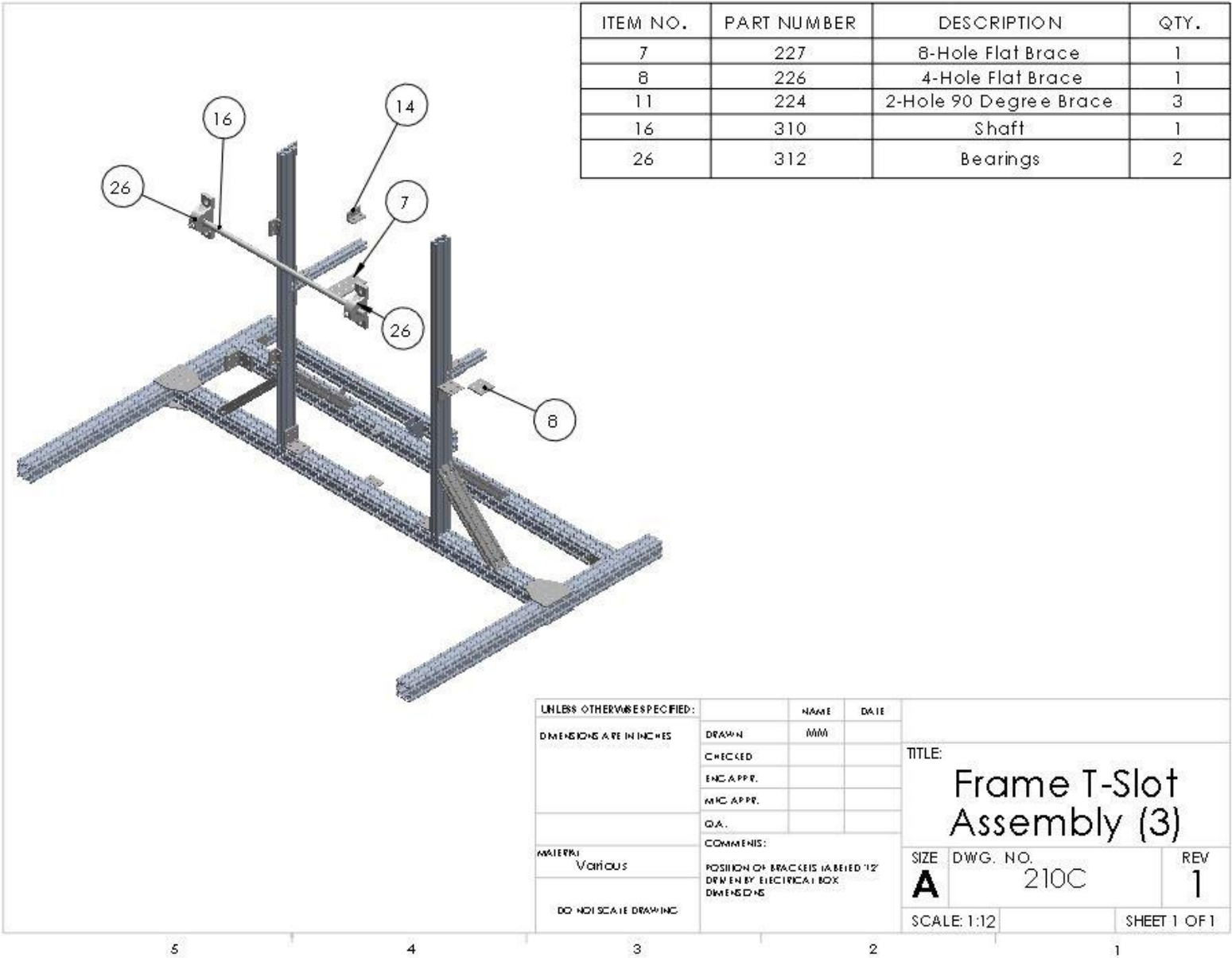


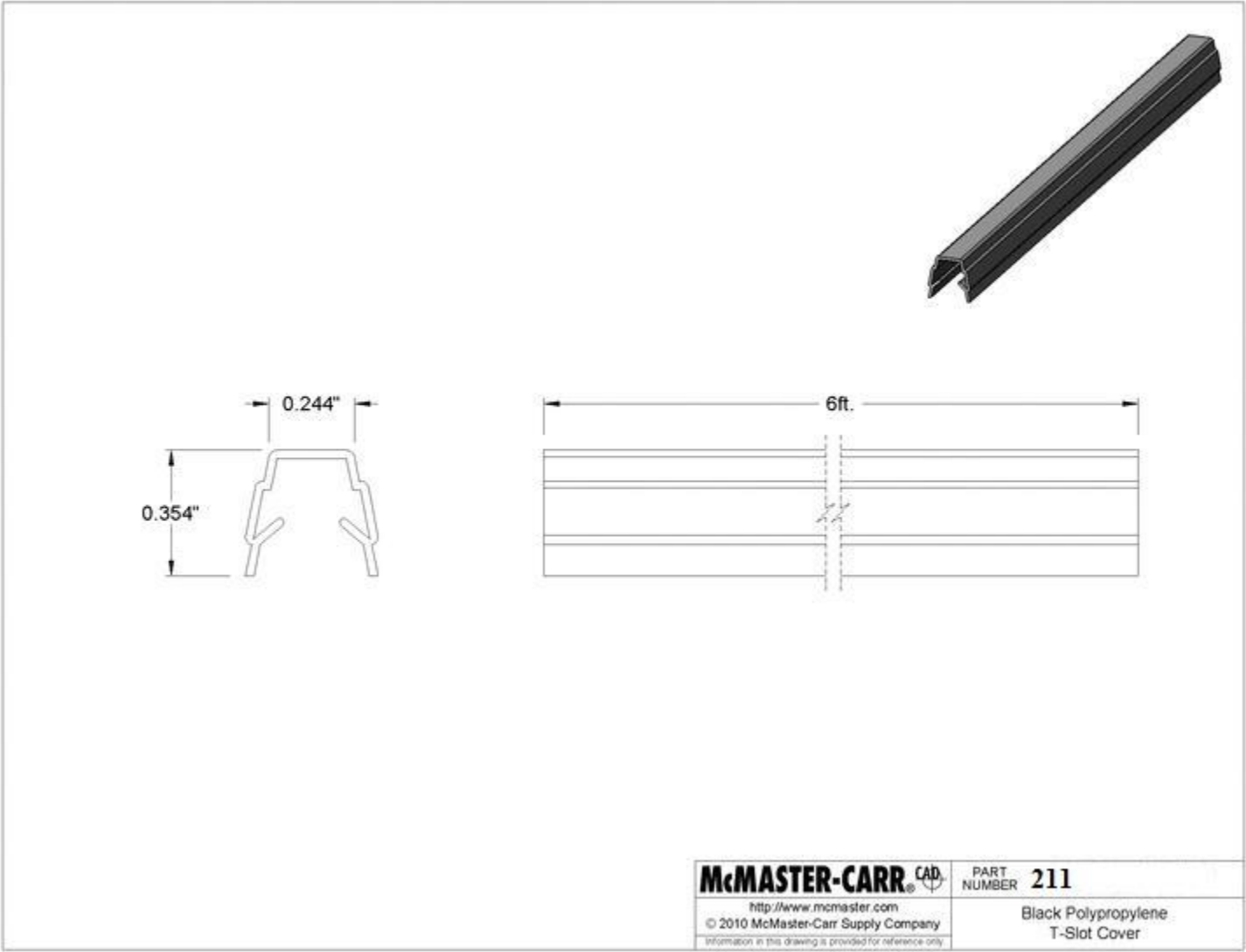


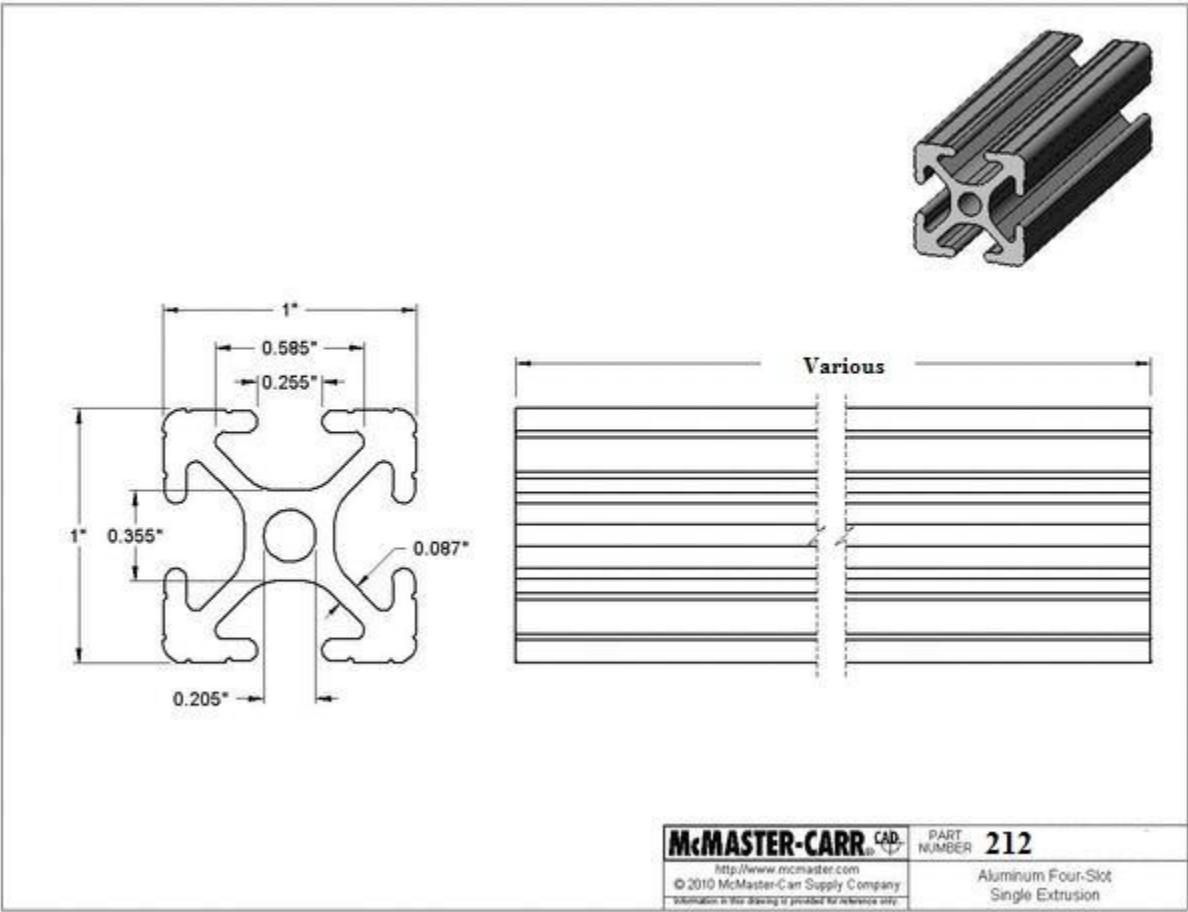


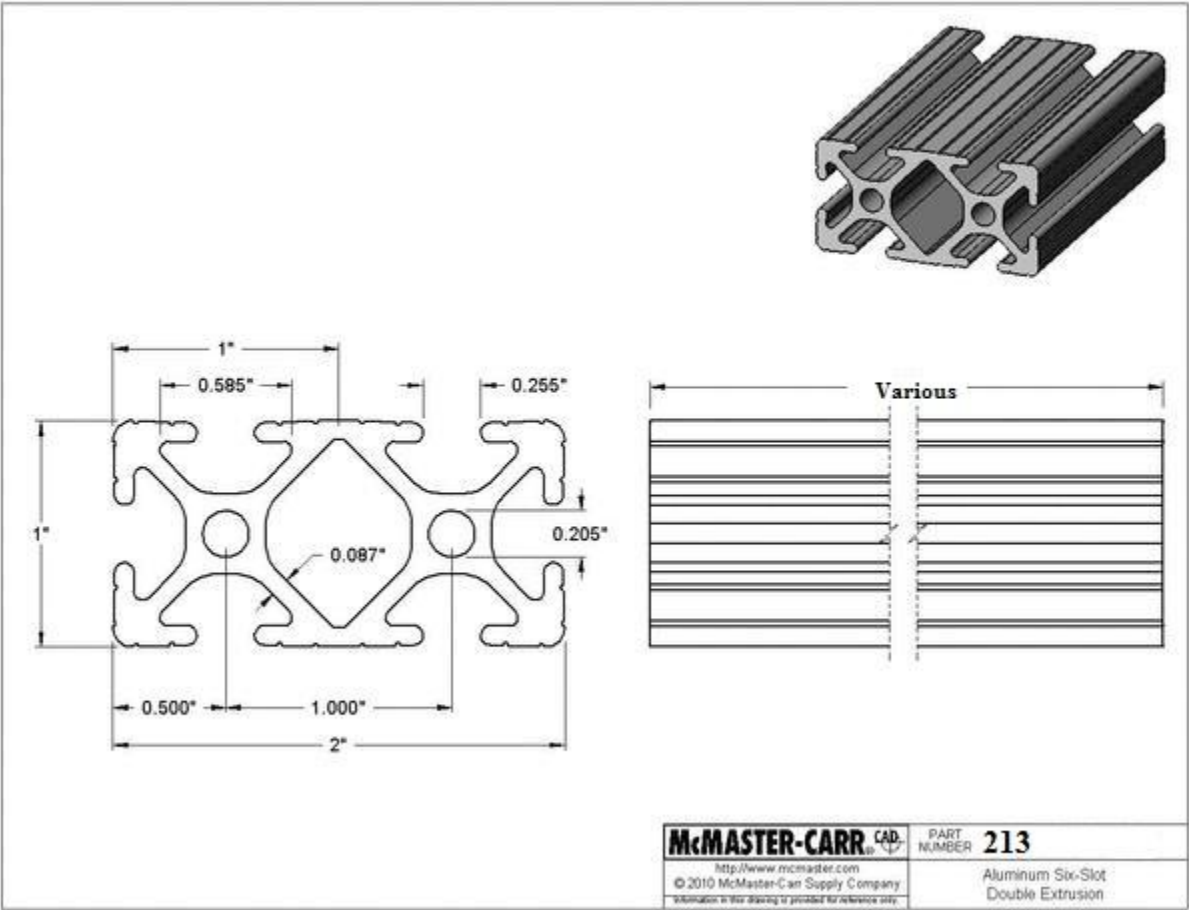












Part No.	2020
Material	6105-T5
Finish	ClearAnodize
Weight Per Foot	1.4060 Lbs.
Stock Length	145" - PN: 2020-145
(+/- .125")	242" - PN: 2020-242
Moment Of Inertia	IX = 0.5509" ^4 IY = 0.5509" ^4
Estimated Area	1.2079 Sq. In.

Cut To Length	Service PN: 7012
Access Hole	Service PN: 7051
Anchor C-Bore	Service PN: 7042
End Tap	Service PN: 7064

PART NUMBER: 214

2.000"

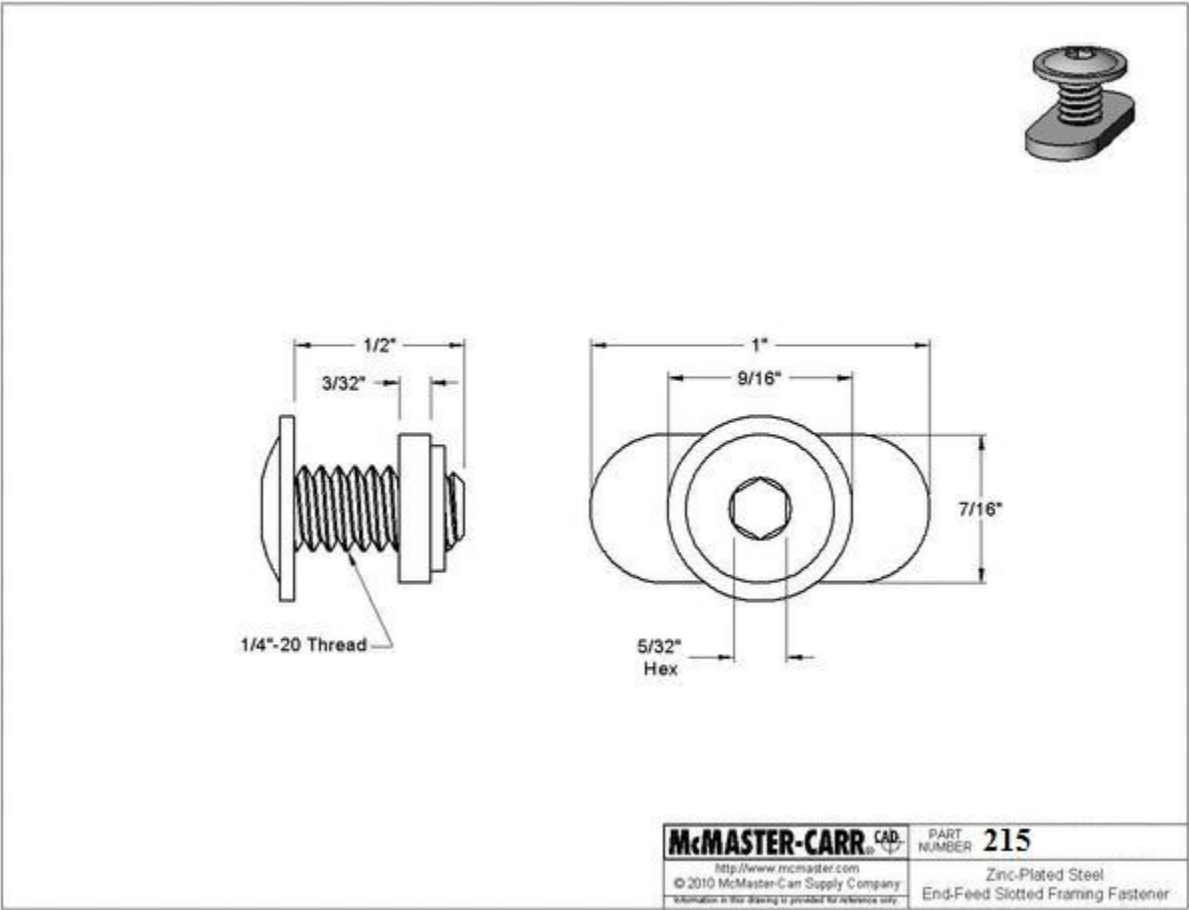
2.000"

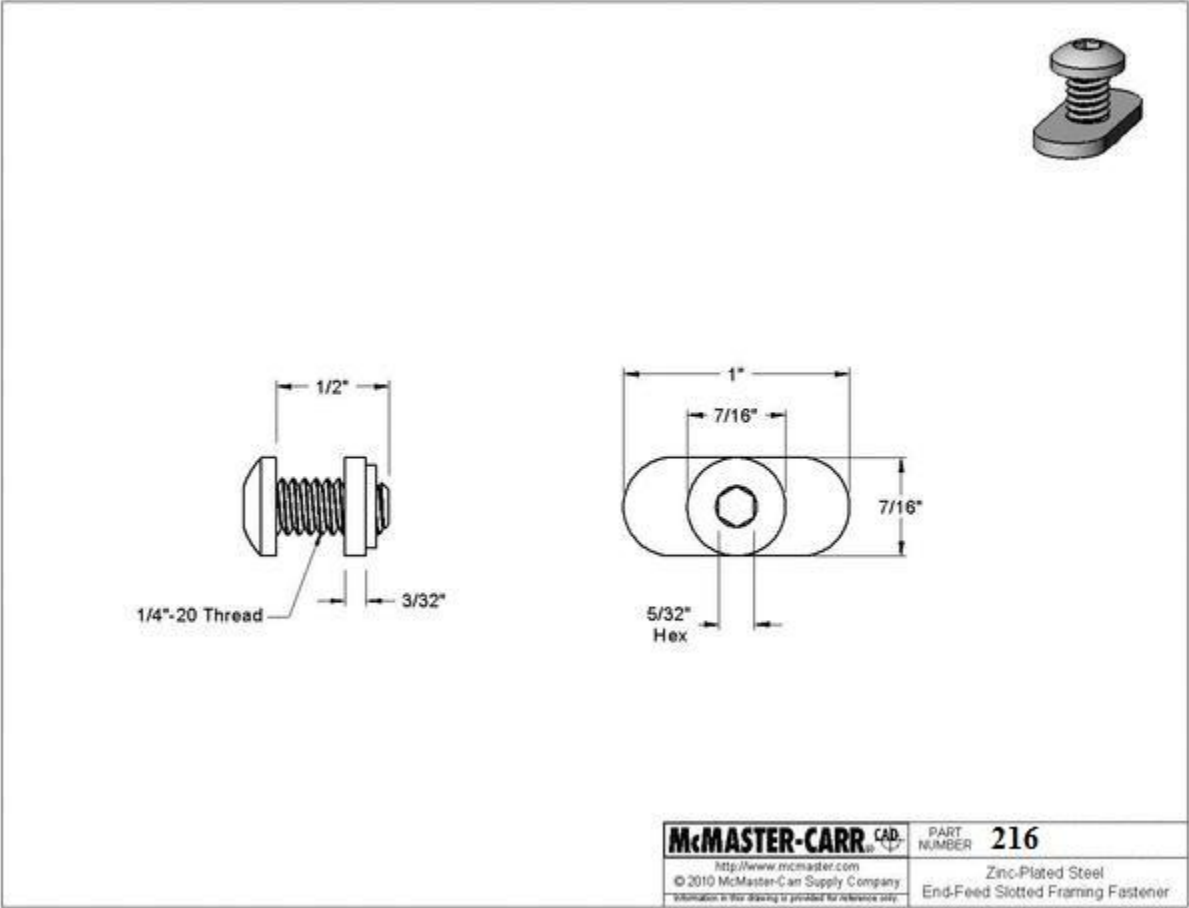
0.255"

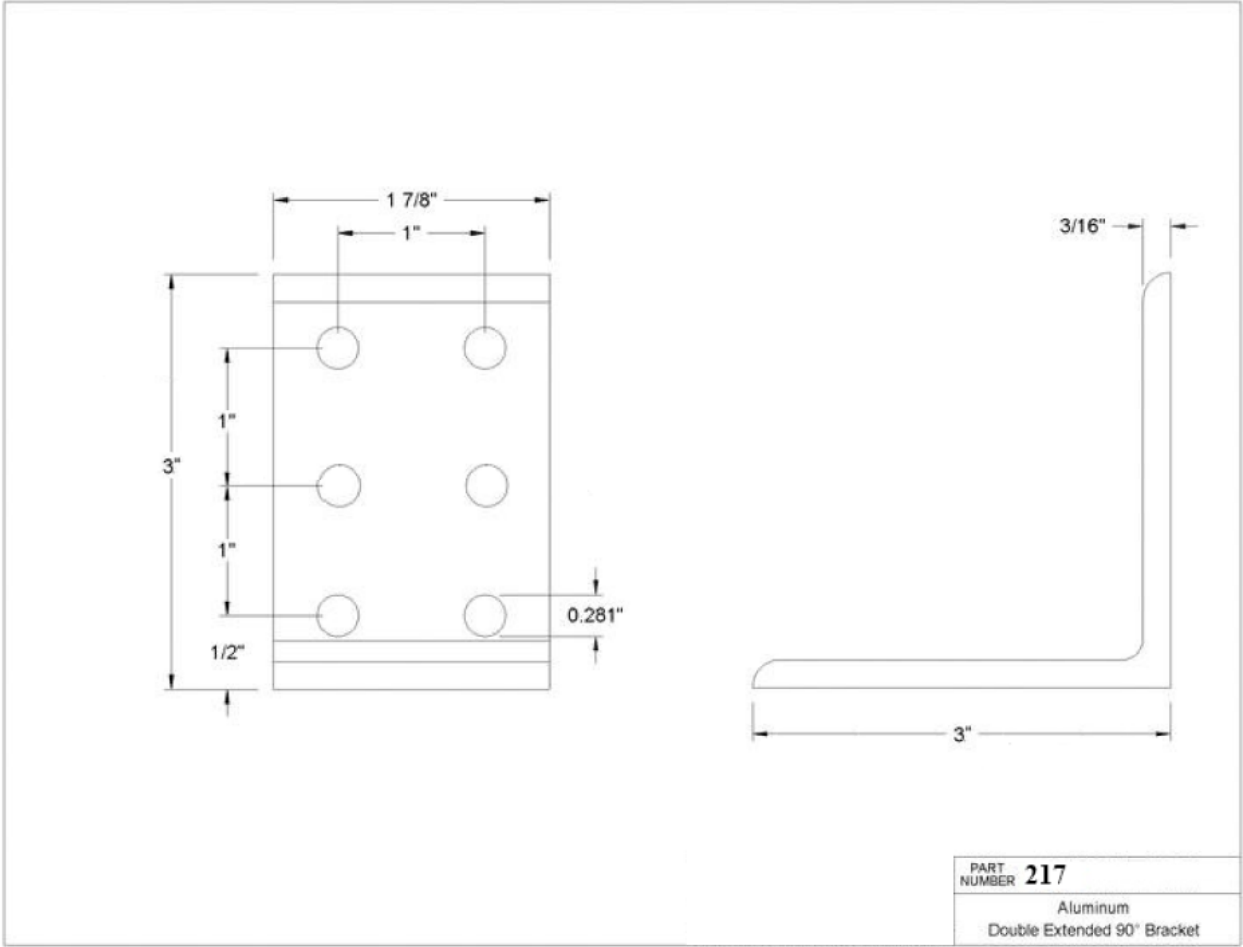
0.585"

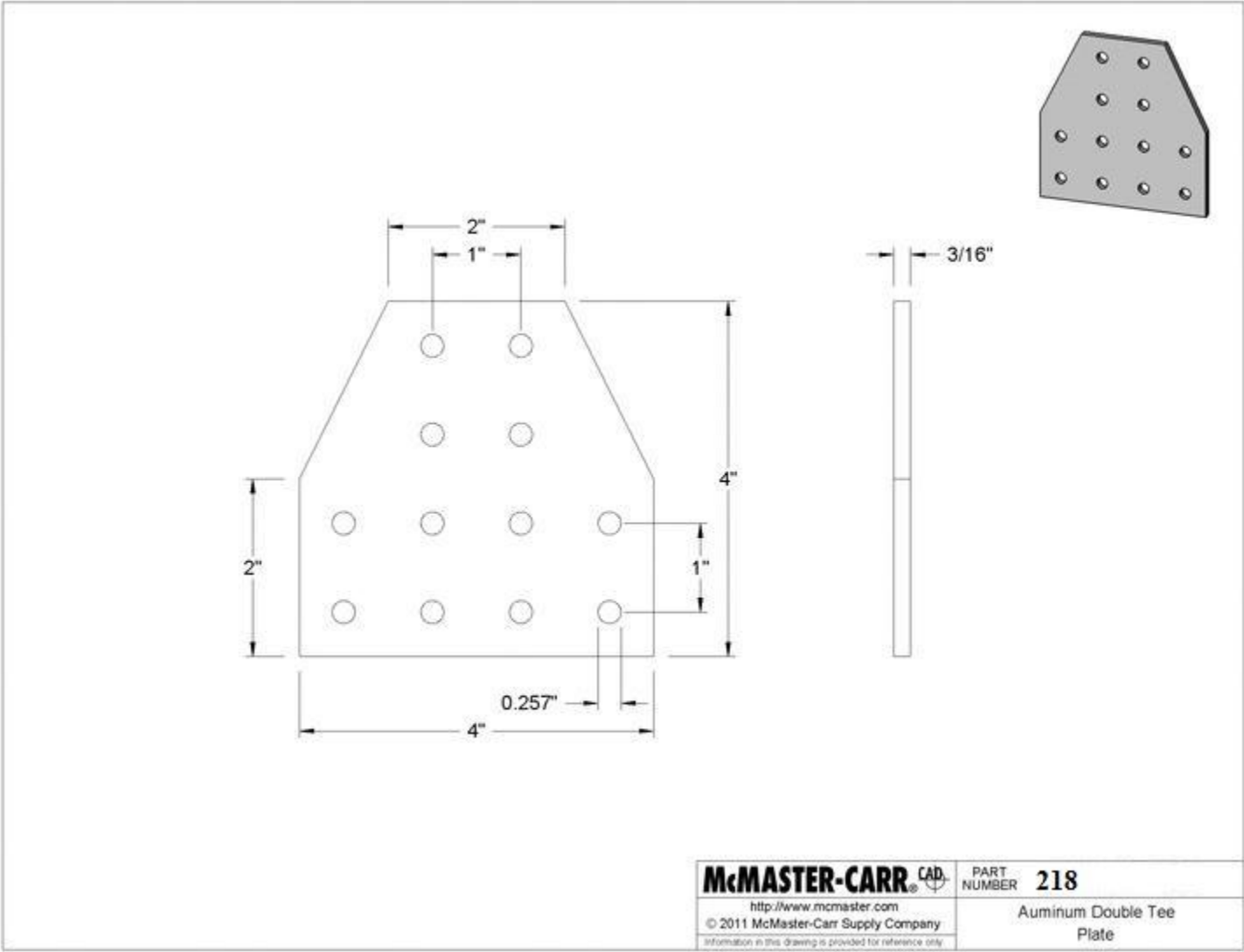
2020

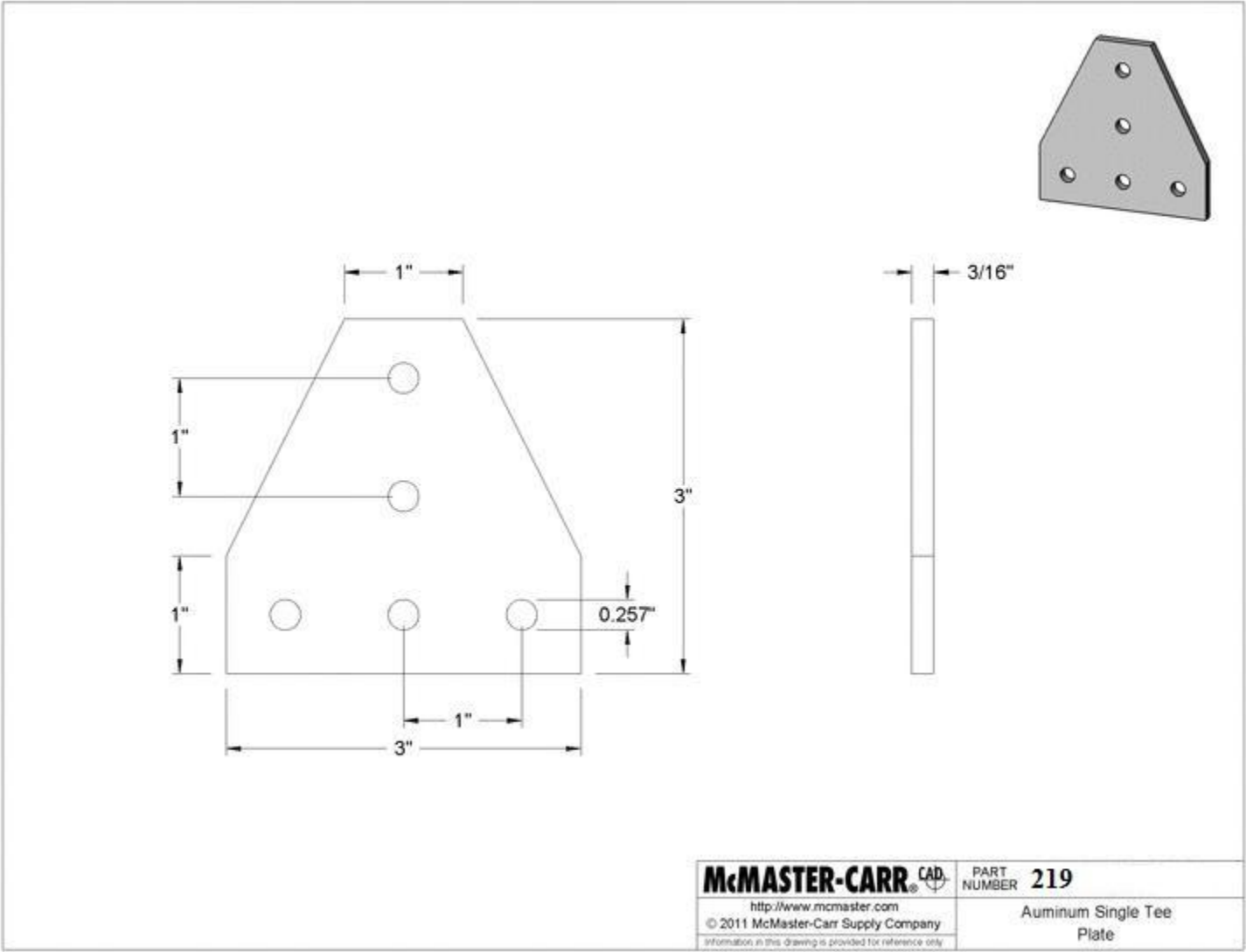
2.00" X 2.00" T-Slotted Profile - Eight Open T-Slots

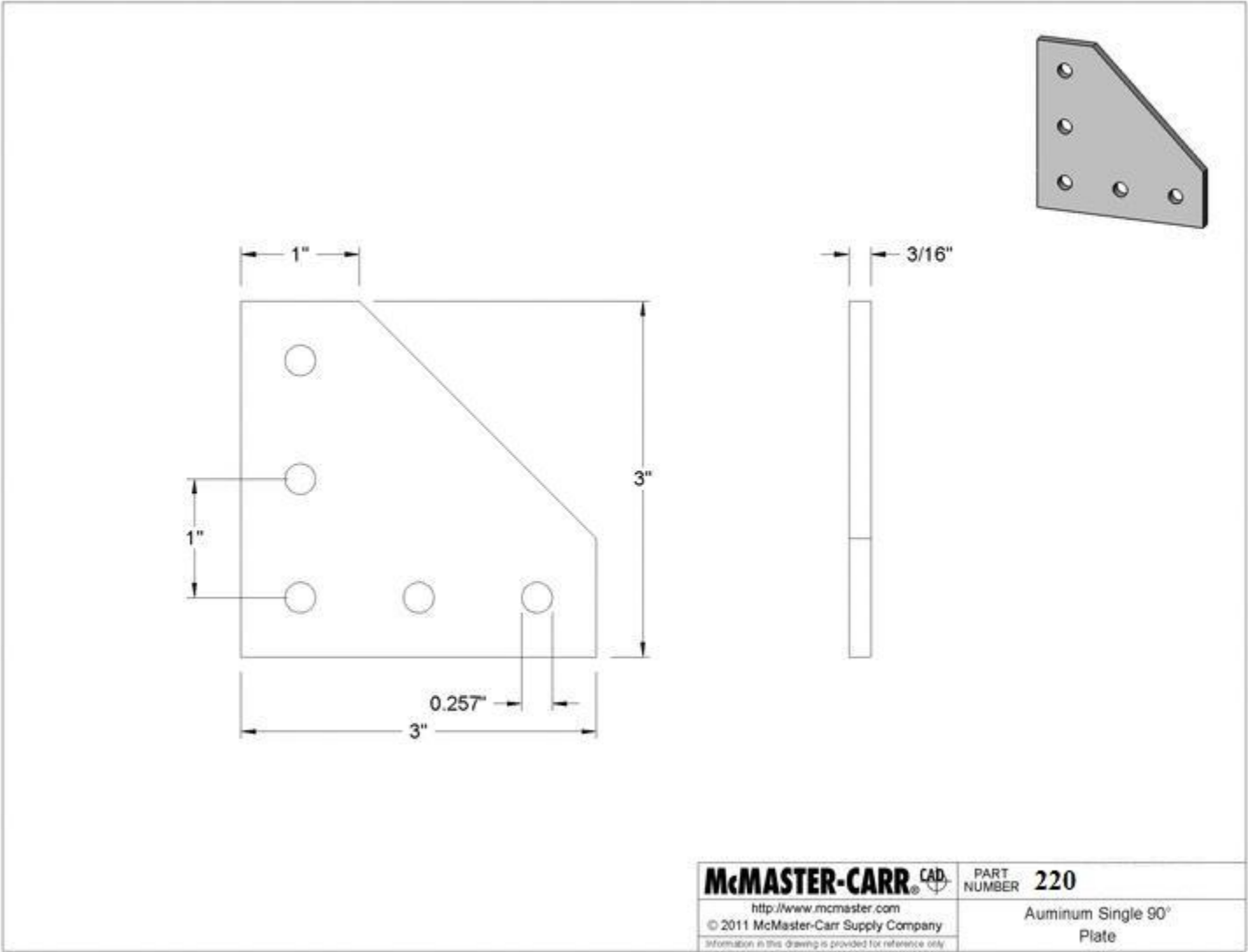


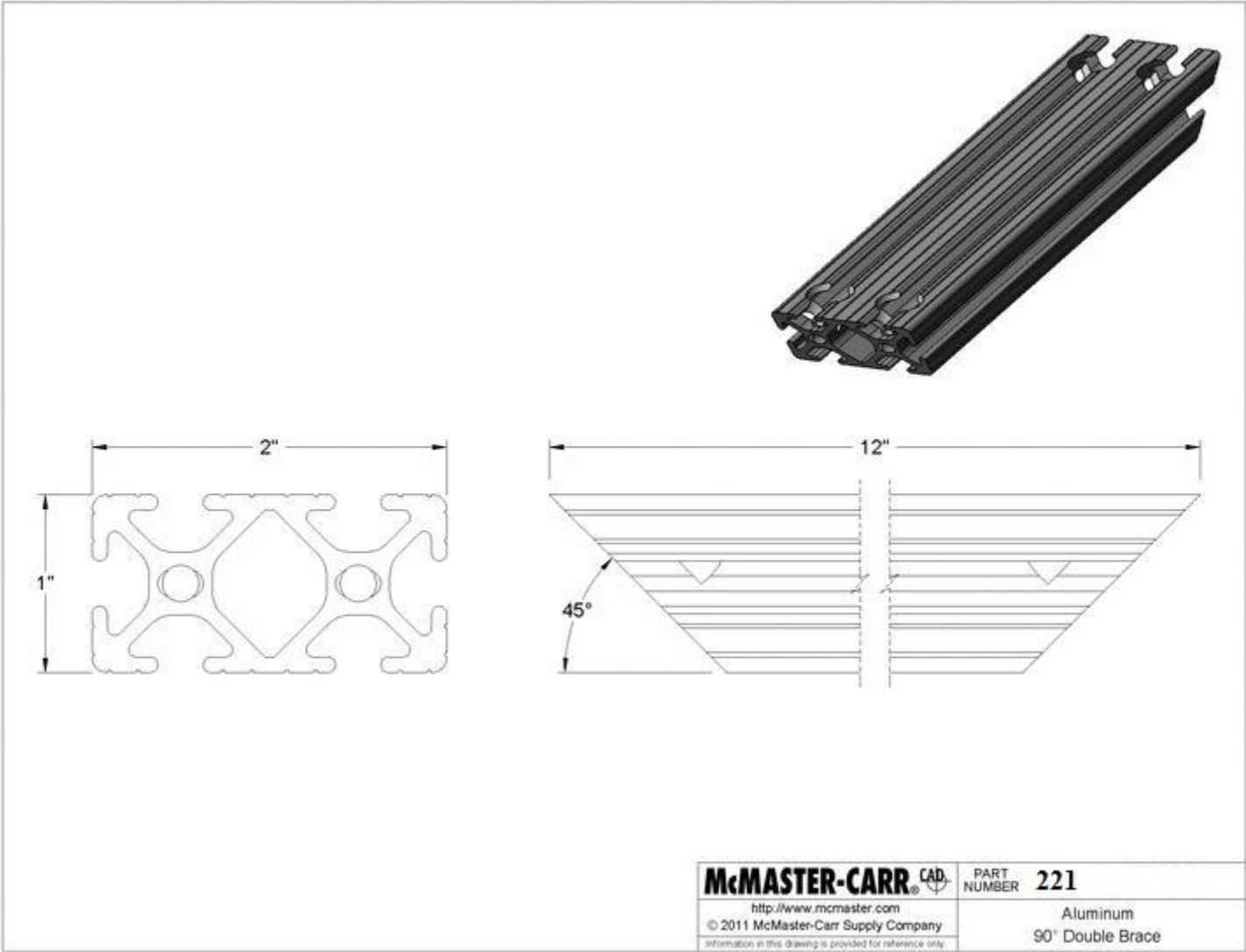


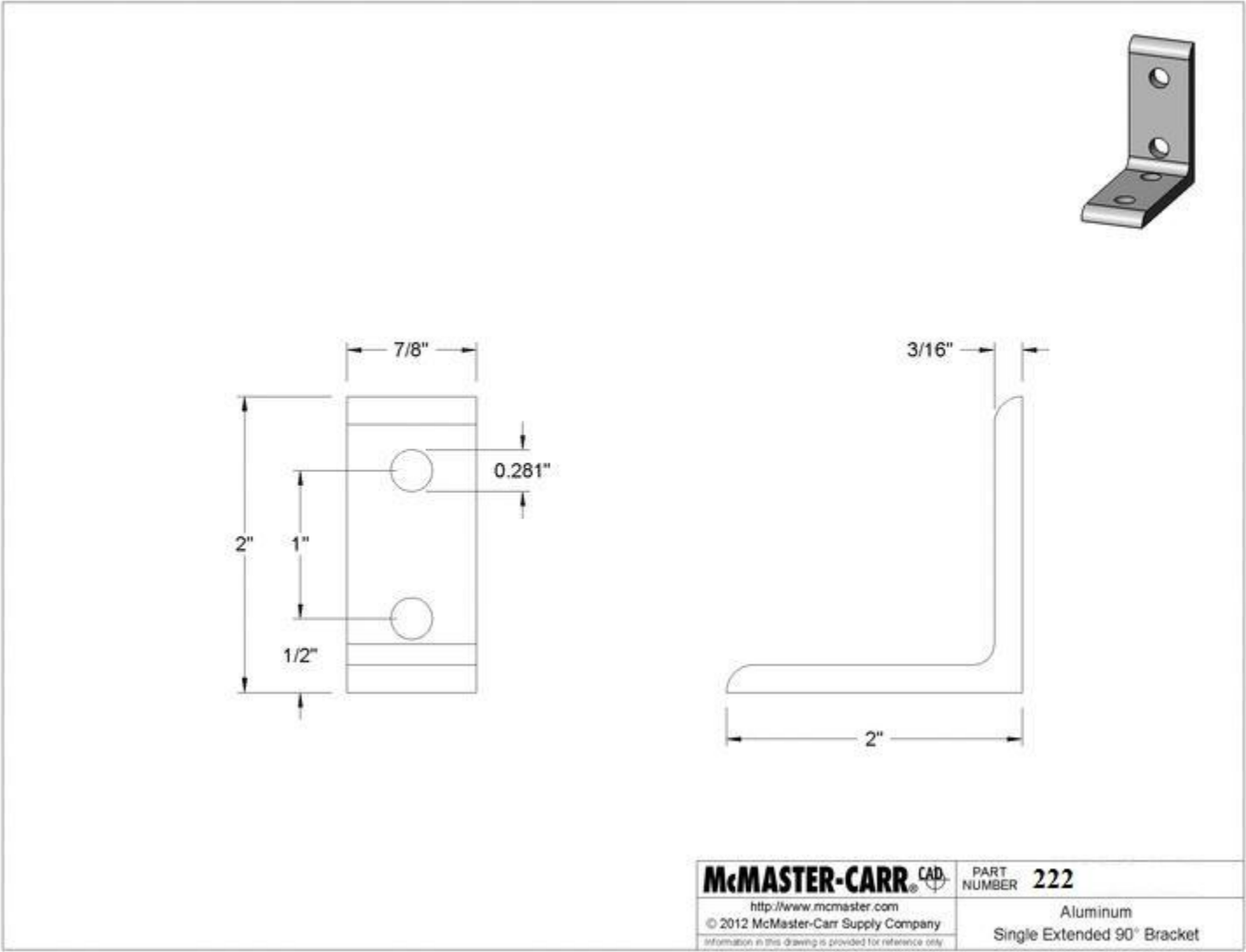


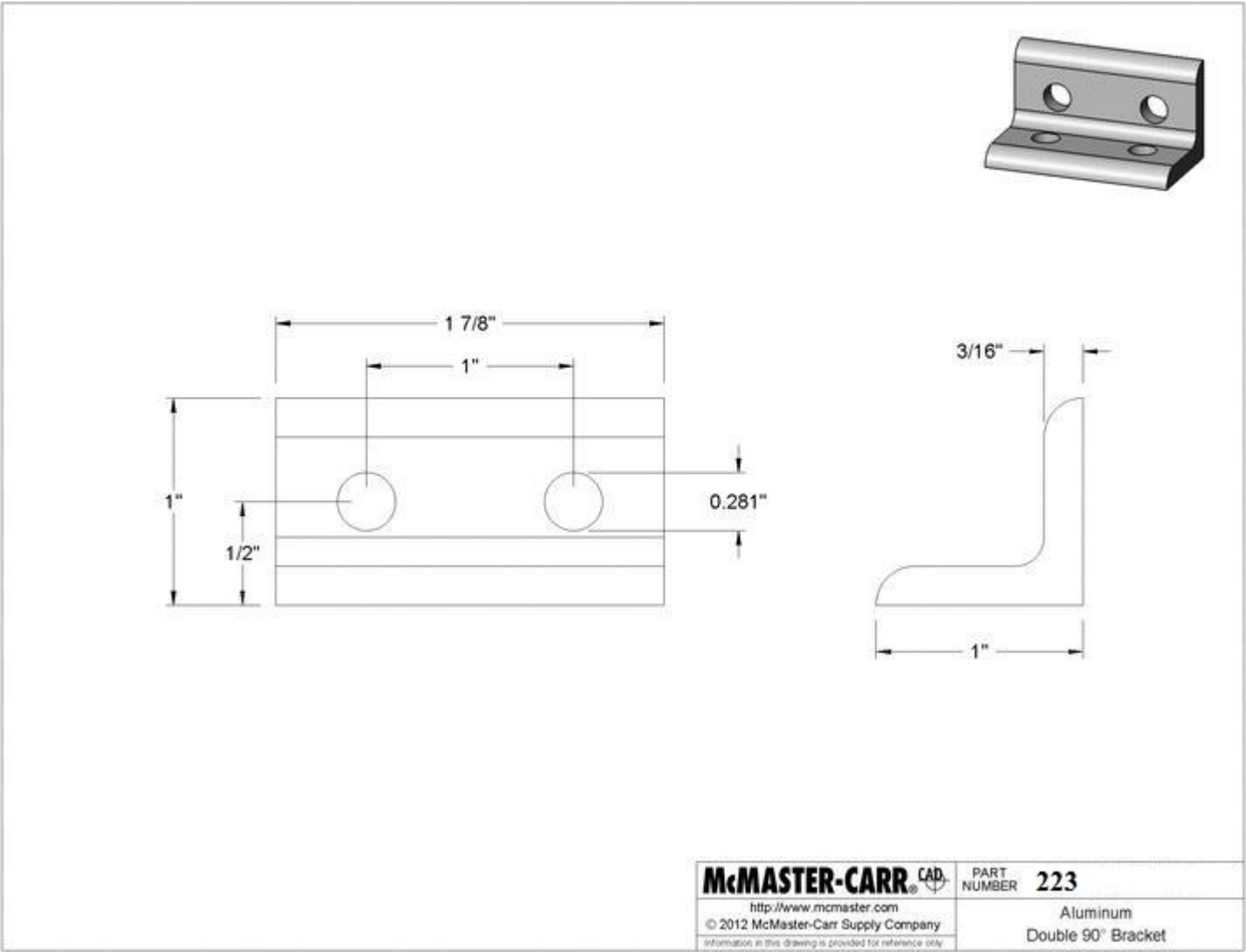


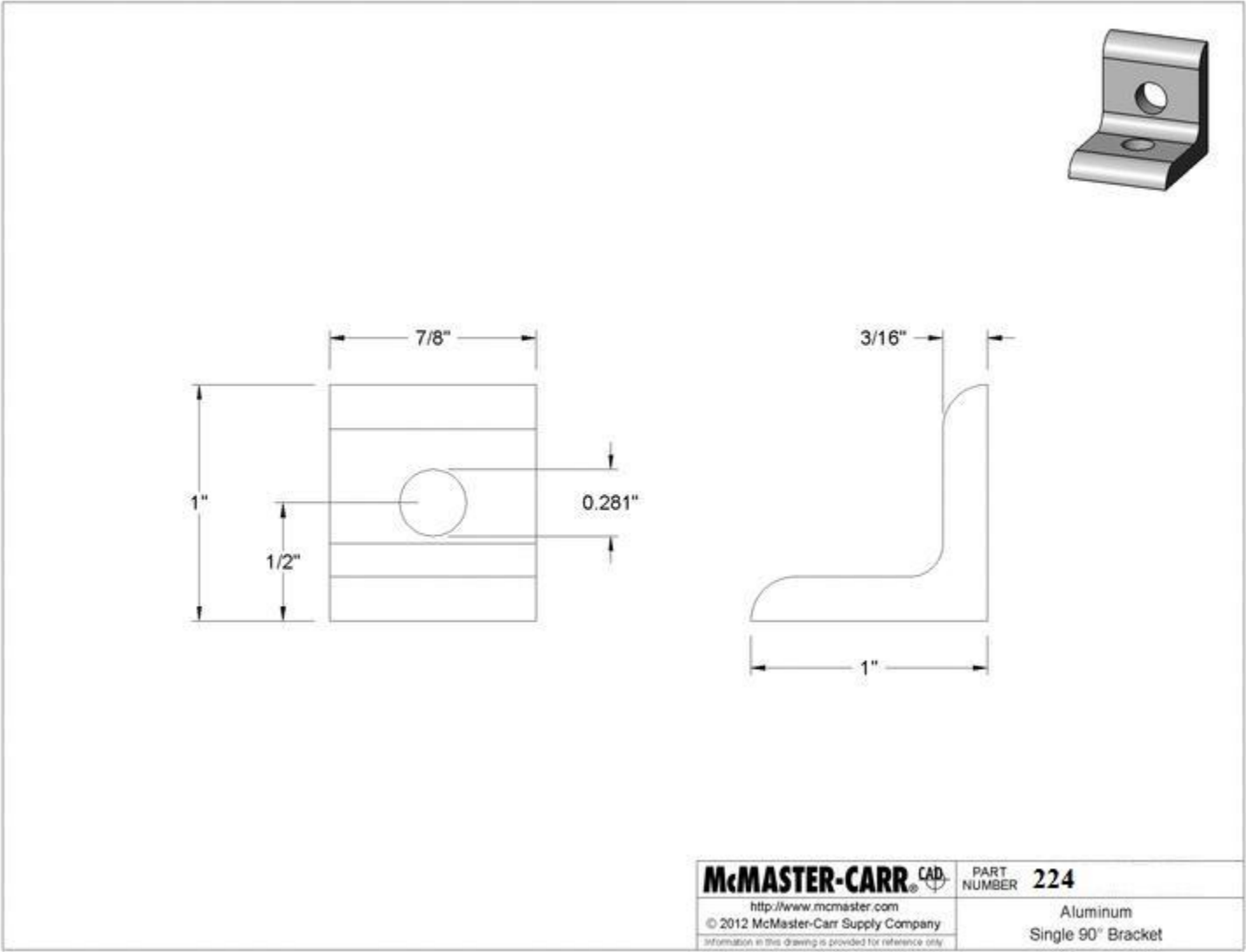


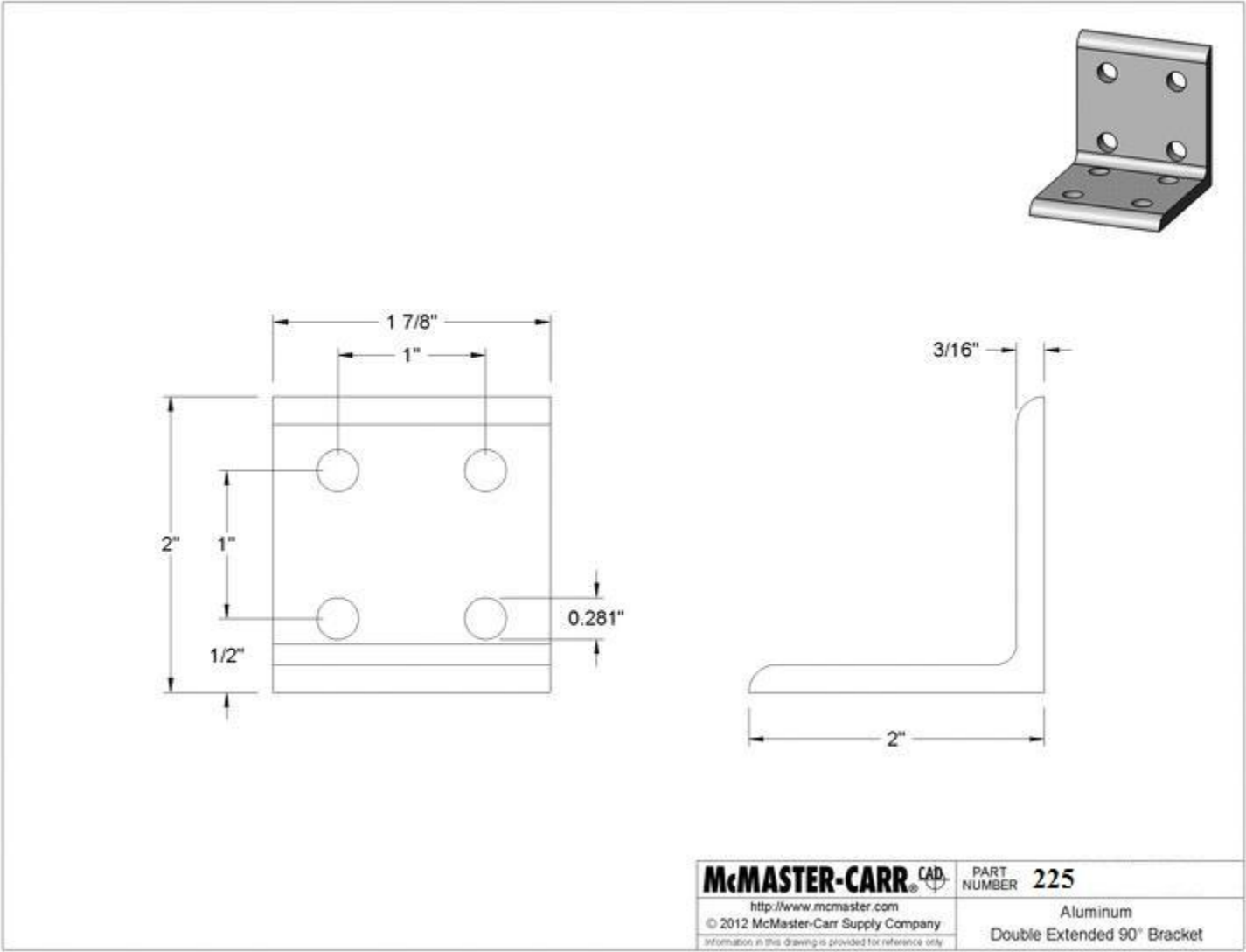


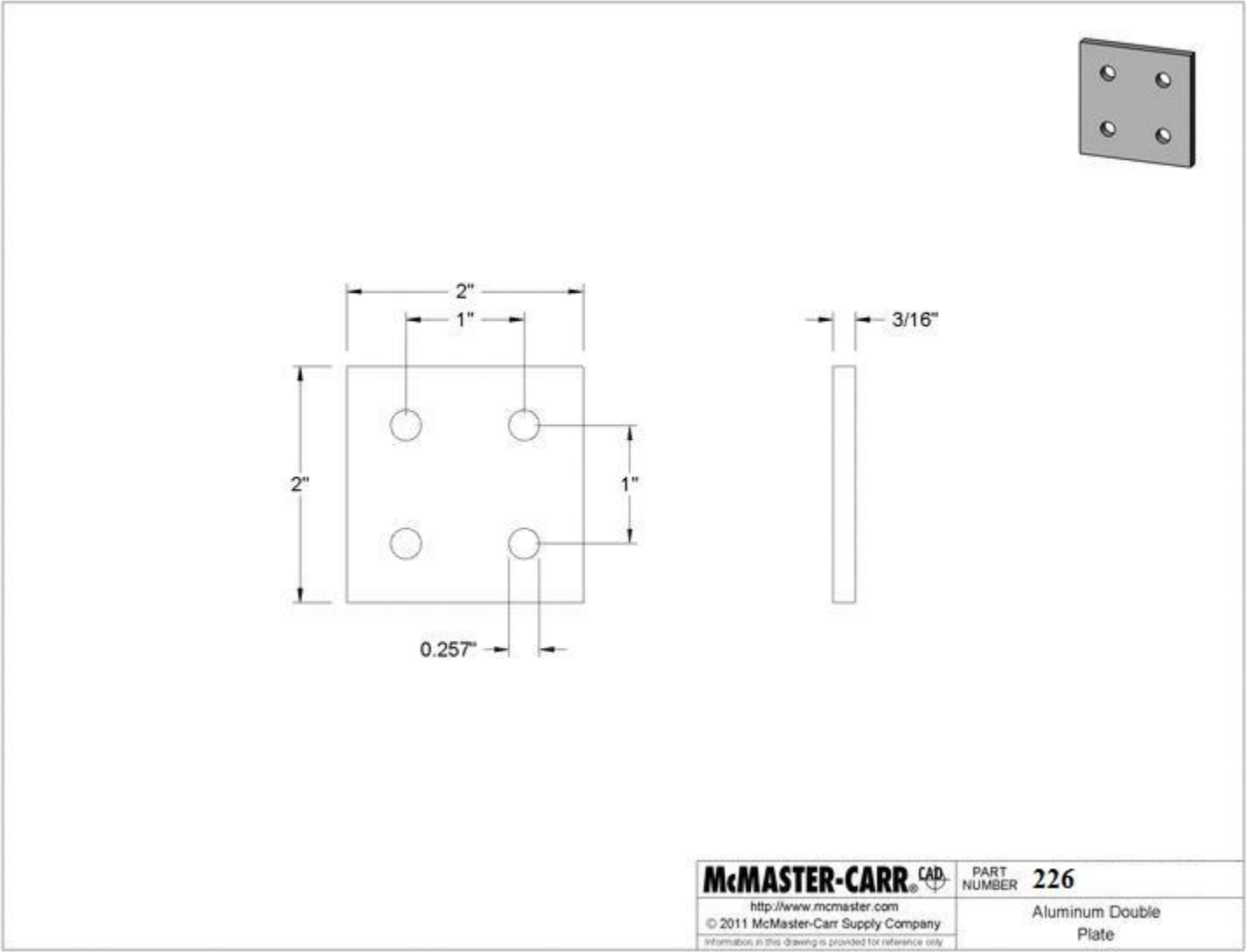


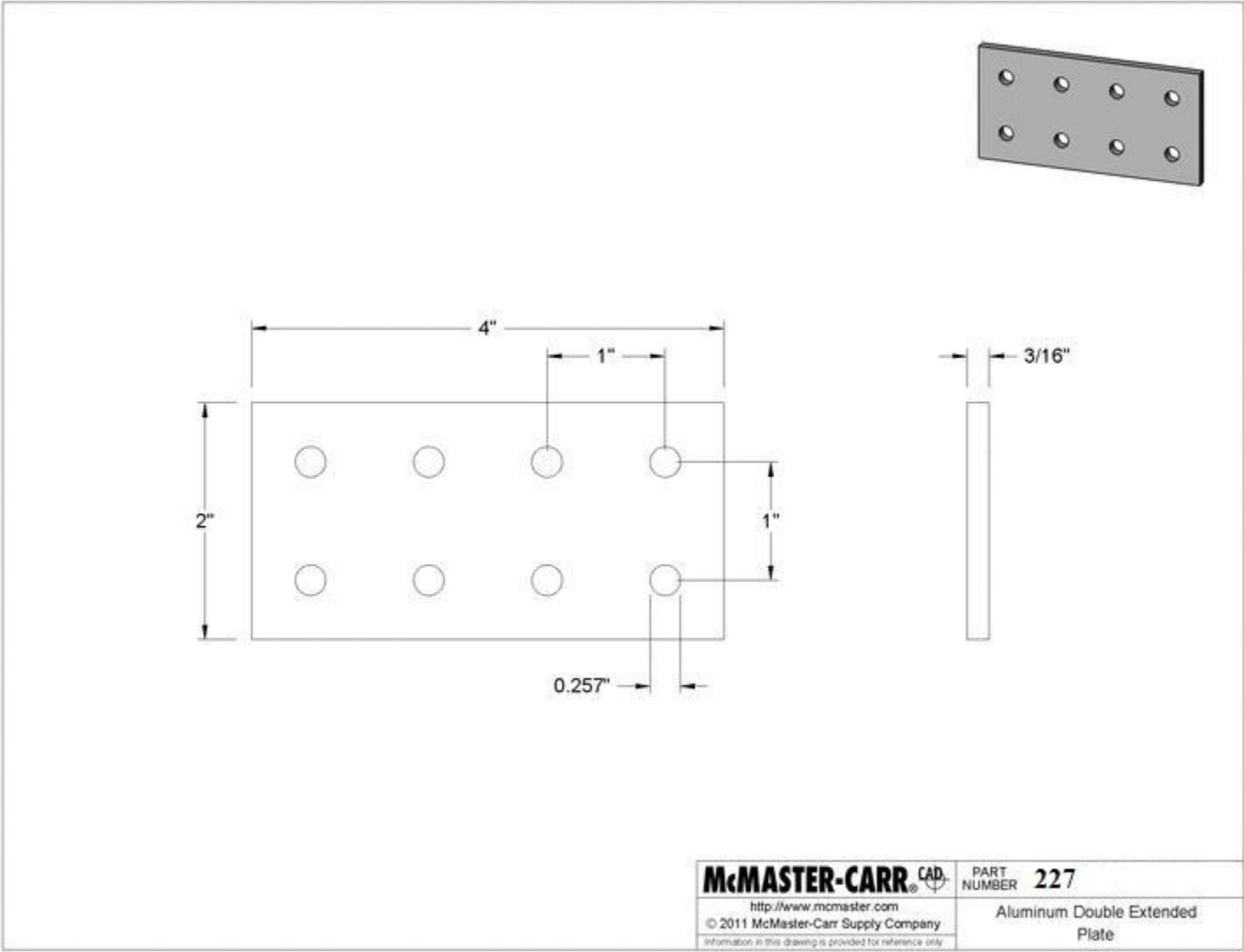


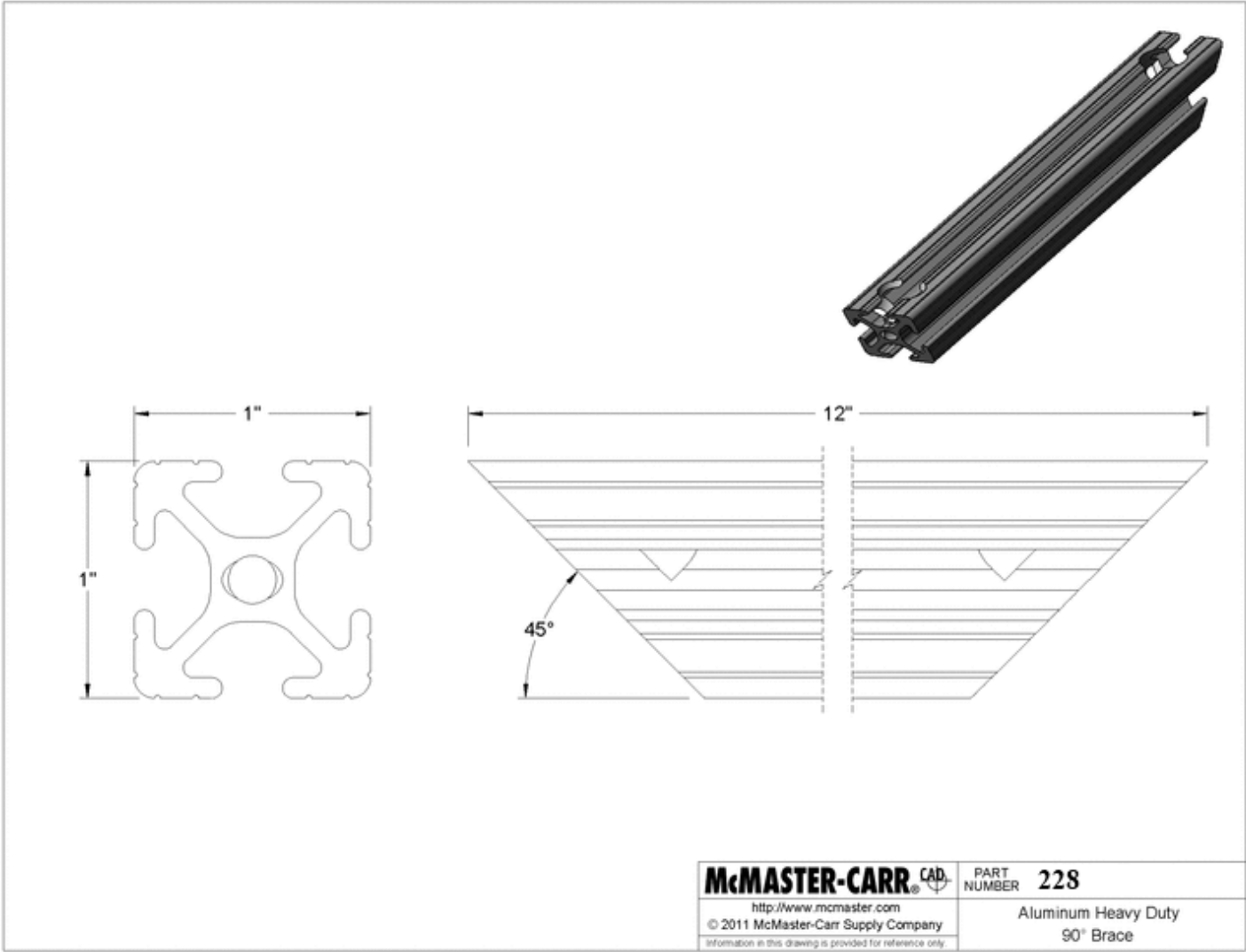


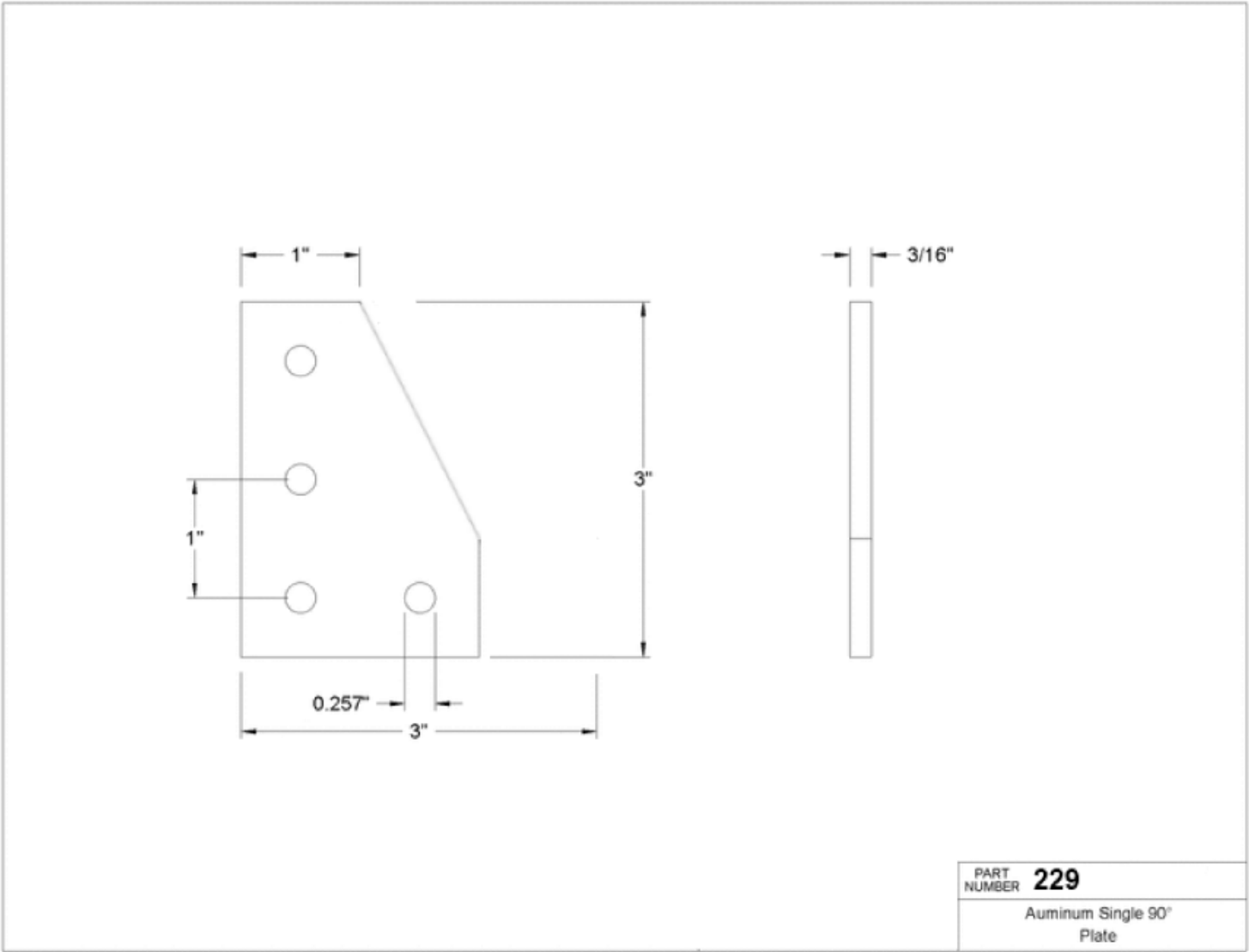


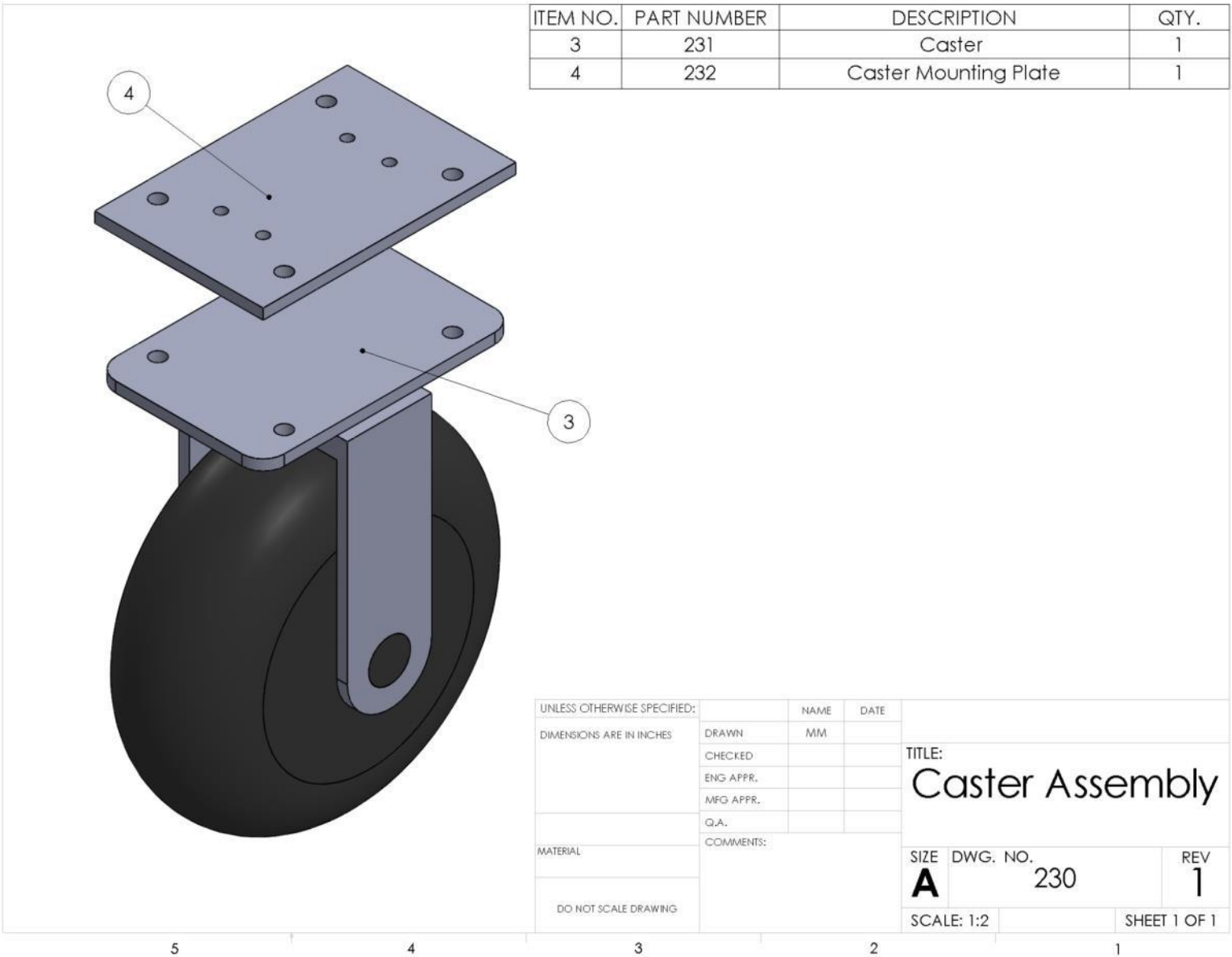












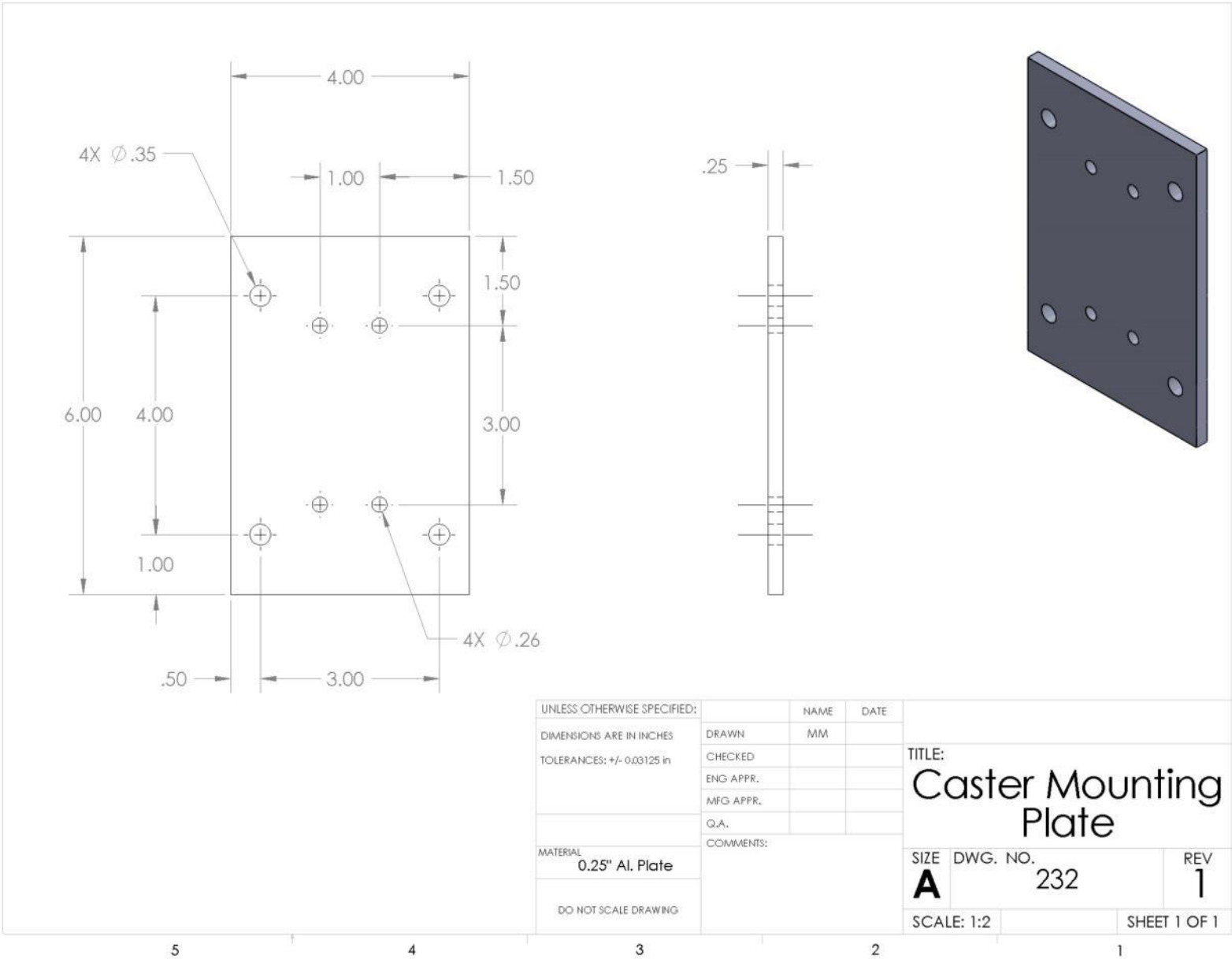
Specifications**PART NUMBER : 231**

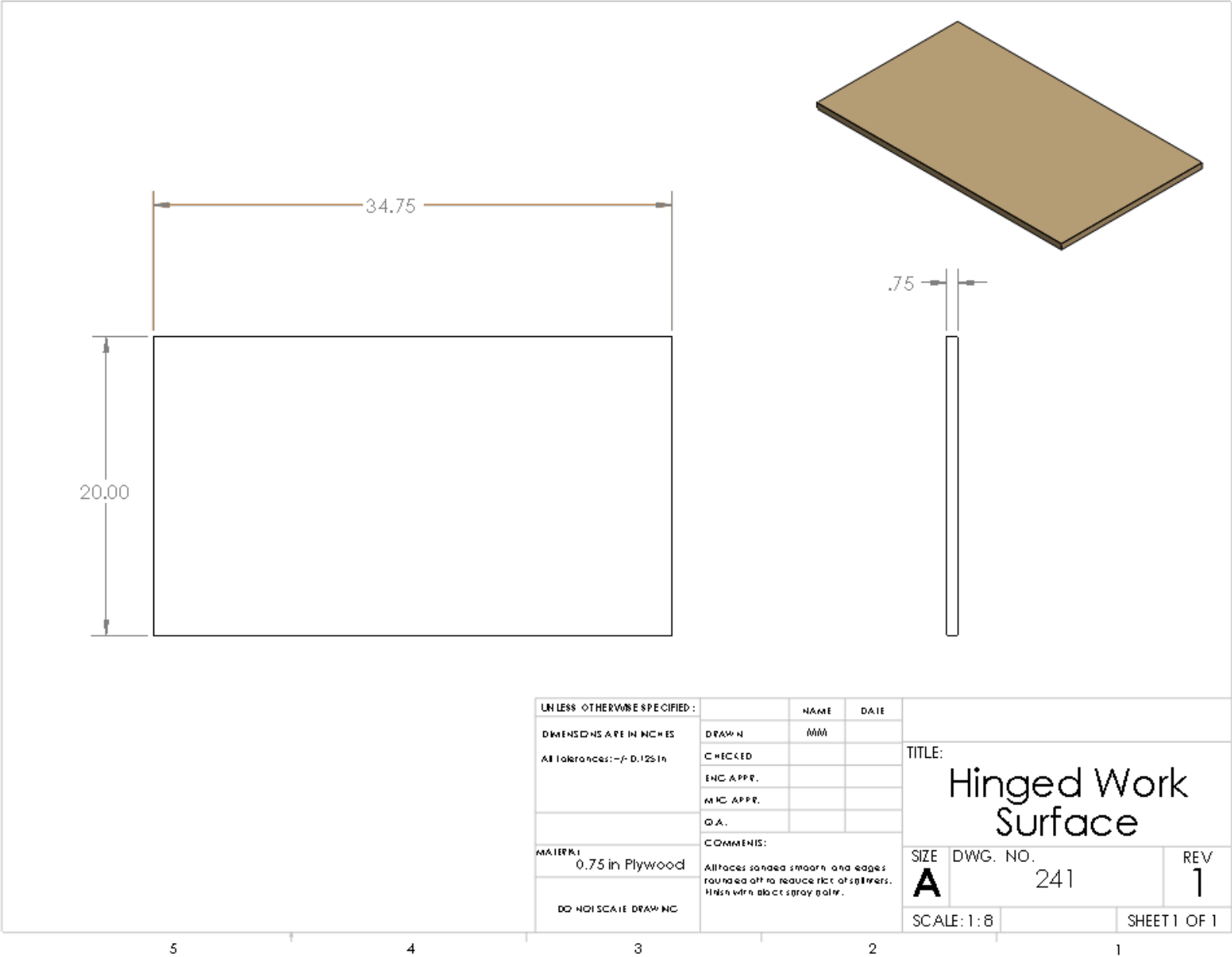
Name	8 in. Cushion Tire Swivel Caster with Brake
SKU	46819
Brake	Yes
Material	Rubber, steel
Maximum Working load (lbs.)	450 lb.
Mounting type	Plate
Quantity	1
Swivel base (deg)	360°
Wheel thickness (in.)	2 in.
Fits tire size (in.)	8 in.
Mounting plate size (in.)	4-1/4 in. x 5-1/8 in.
Product Height	9-1/2 in.
Product Length	8 in.
Product Weight	9 lbs.
Product Width	4-1/4 in.
Shipping Weight	8.95 lb.
Tire size (in.)	8 in.
Warranty	90 Day

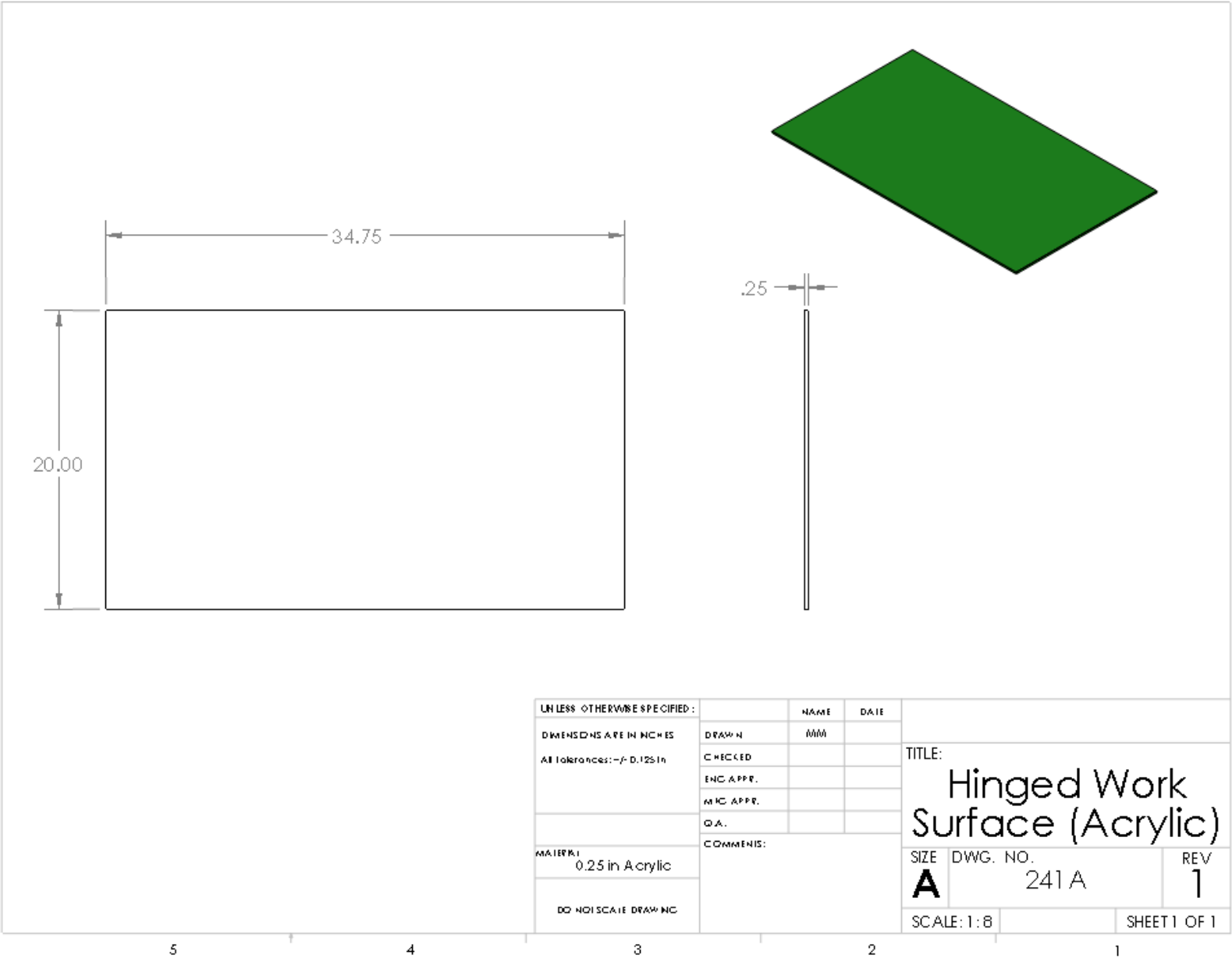
8 In. Cushion Tire Swivel Caster With Brake

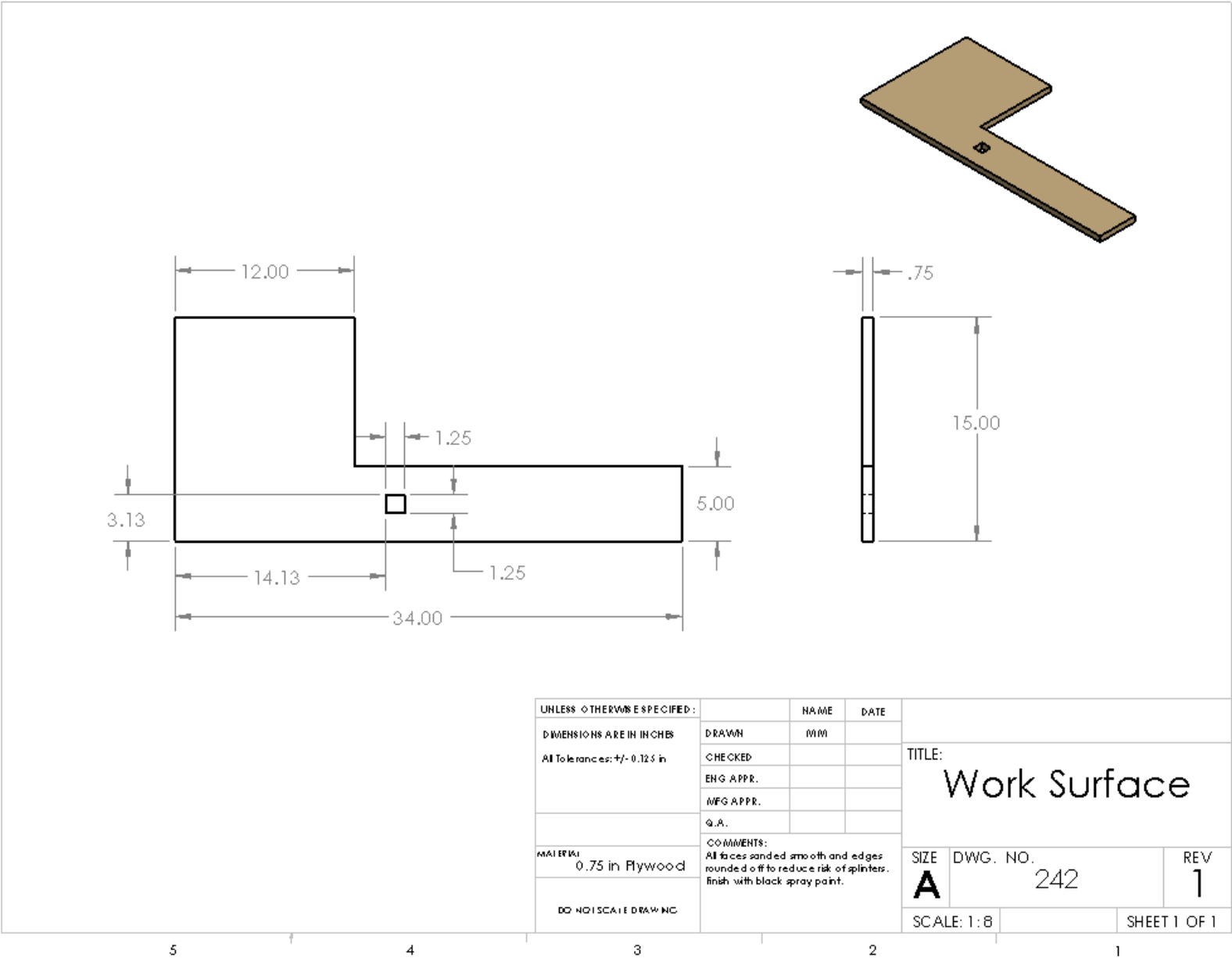
- Item#46819

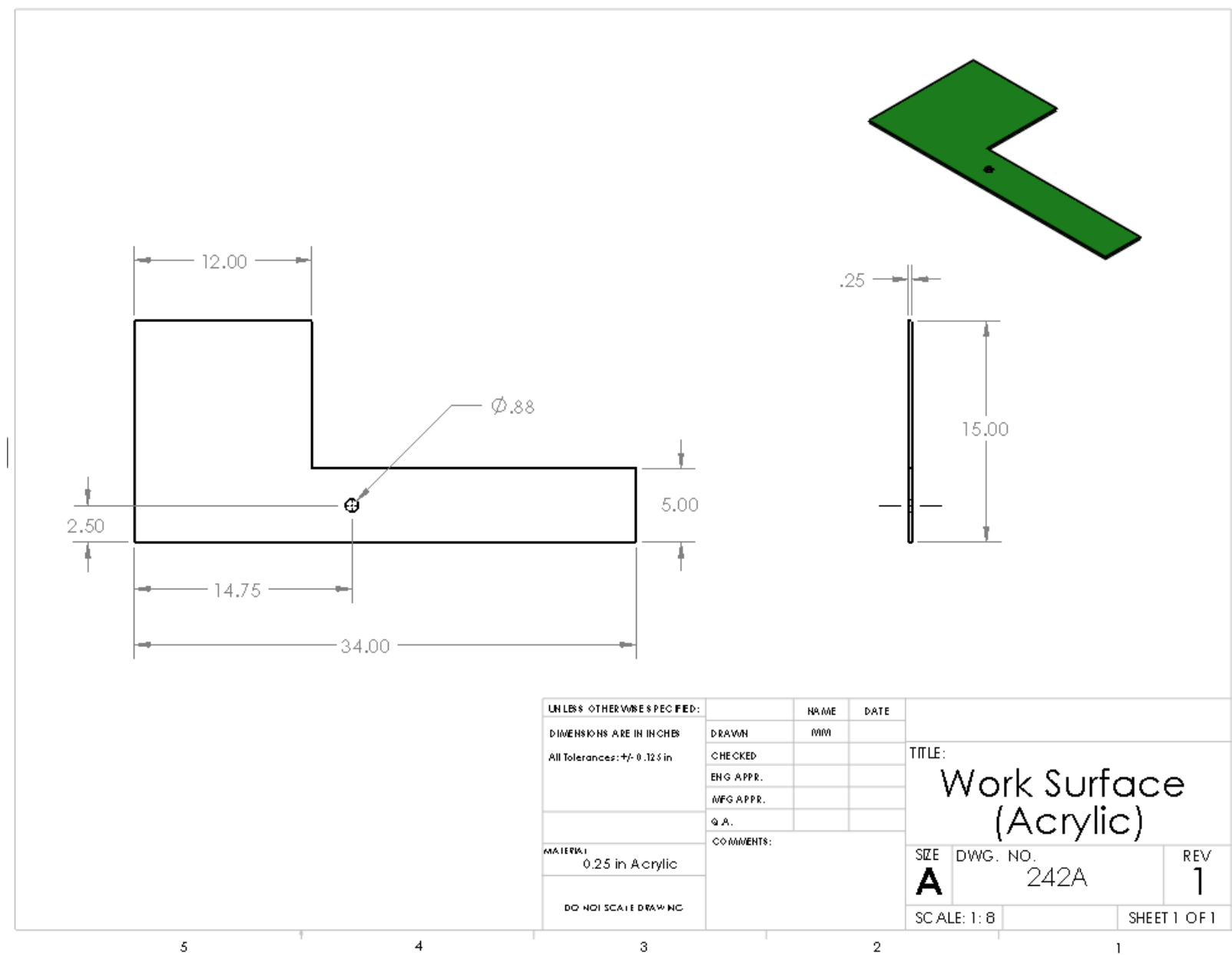










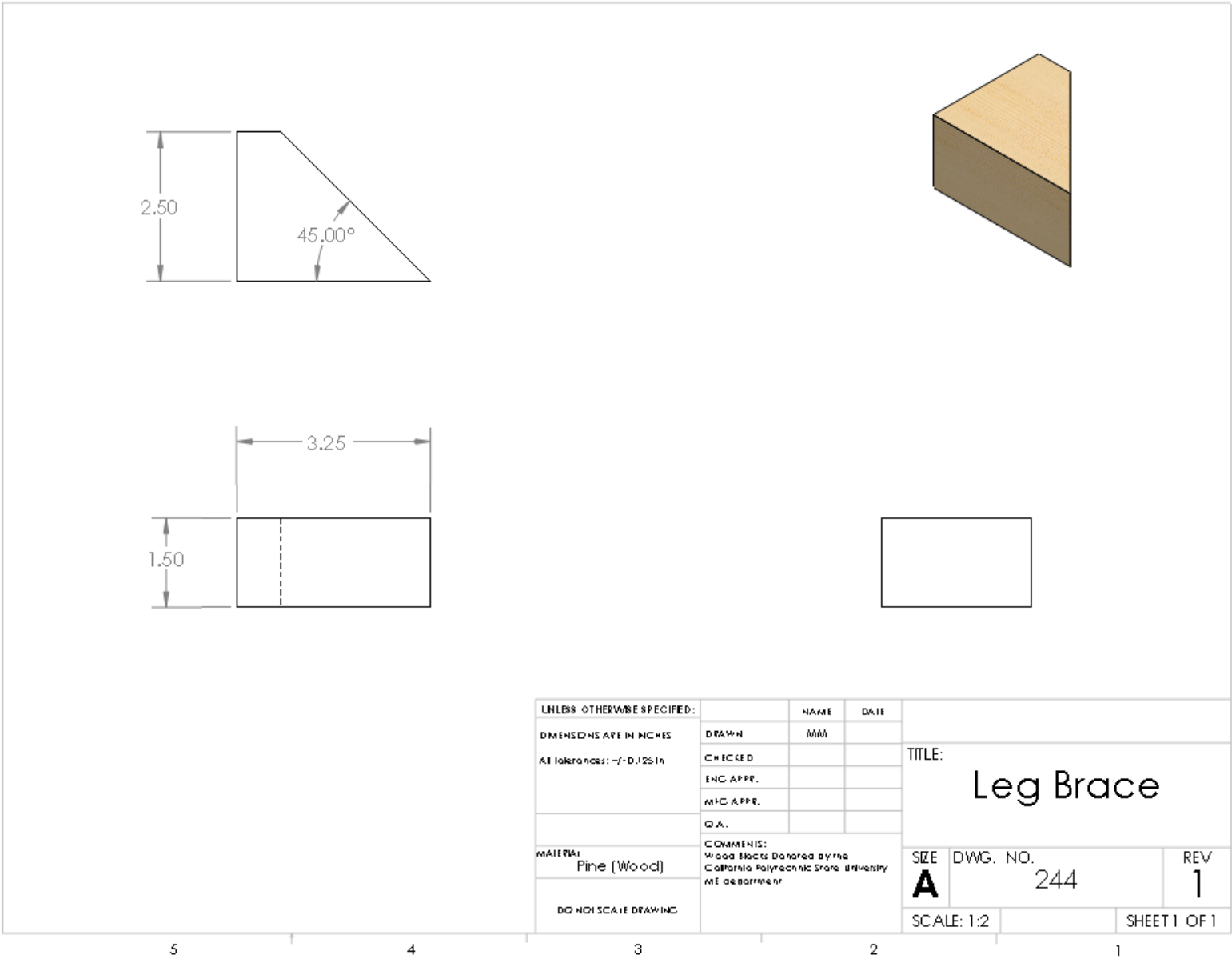


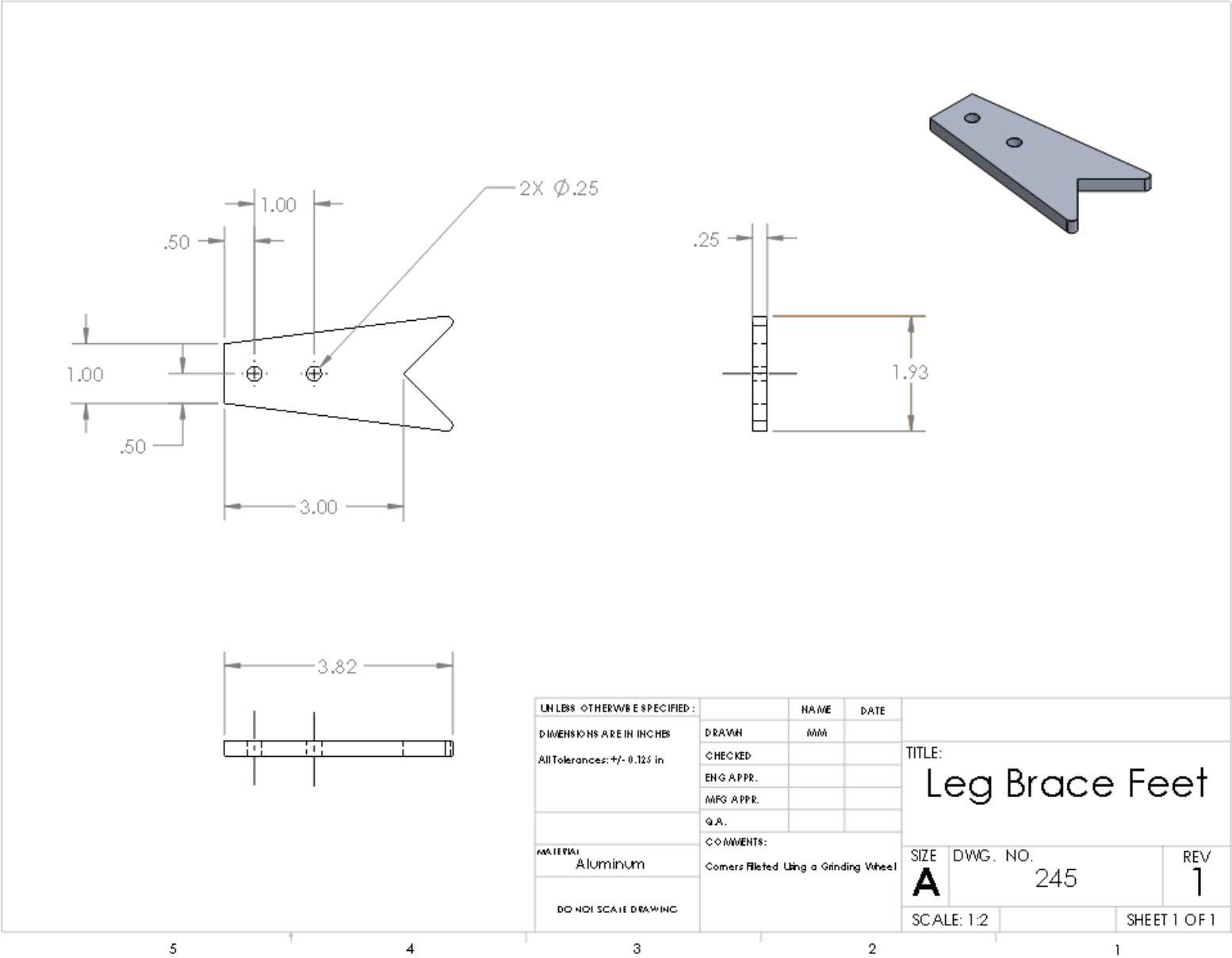
PART NUMBER: 243



SPECIFICATIONS

Assembled Depth (in.)	0.125 in	Assembled Height (in.)	30 in
Assembled Width (in.)	1.5 in	Builders Hardware Product Type	Door Hinge
Commercial / Residential	Residential	Fasteners included	Yes
Finish	Bright Nickel	Finish Family	Nickel
Hinge type	Surface mount	Manufacturer Warranty	None
Material	Steel	Mounting type	Wall
Package Quantity	1	Pin type	Fixed
Radius (in.)	0	Returnable	90-Day





PART NUMBER: 246

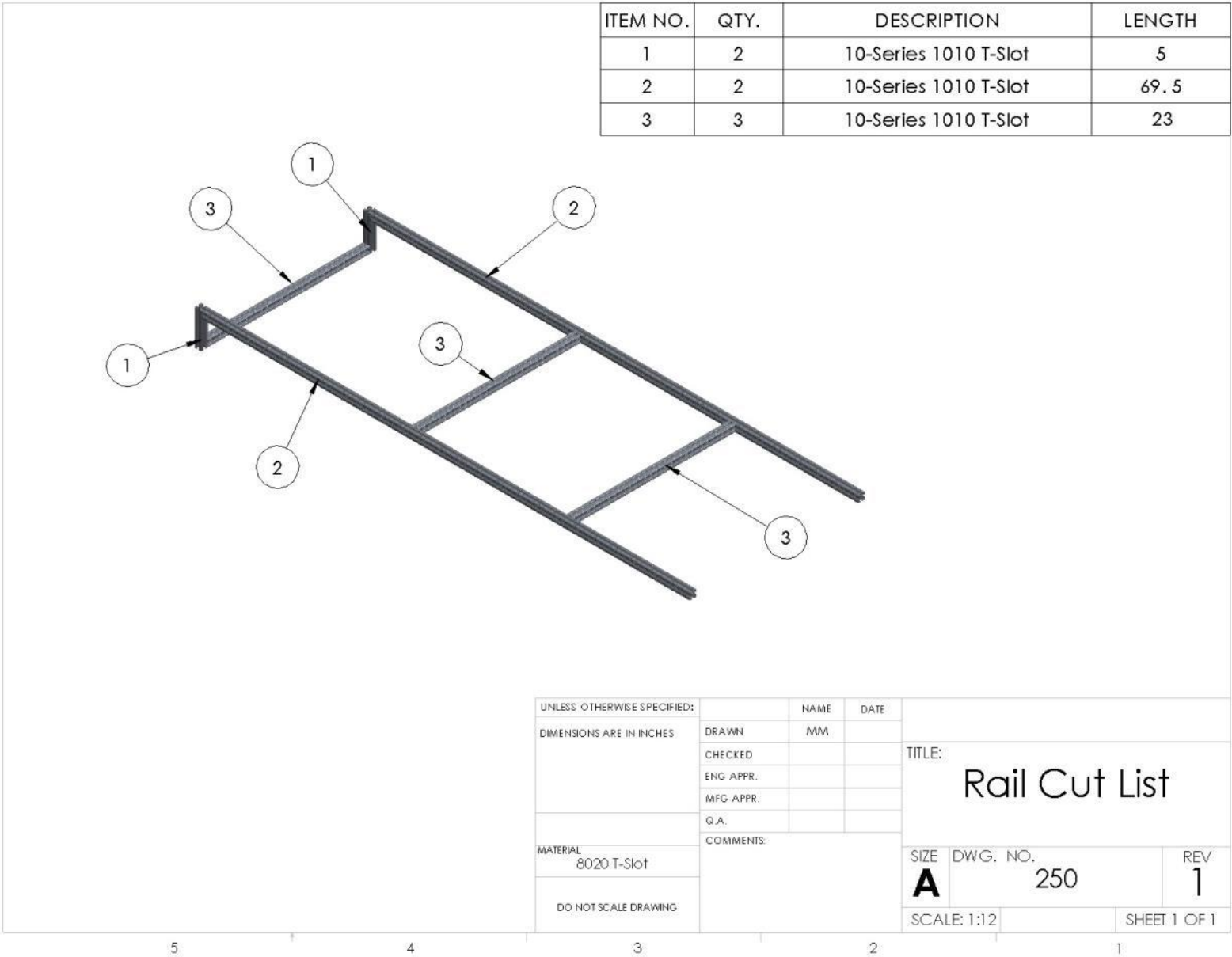
Everbilt | Model # 15291

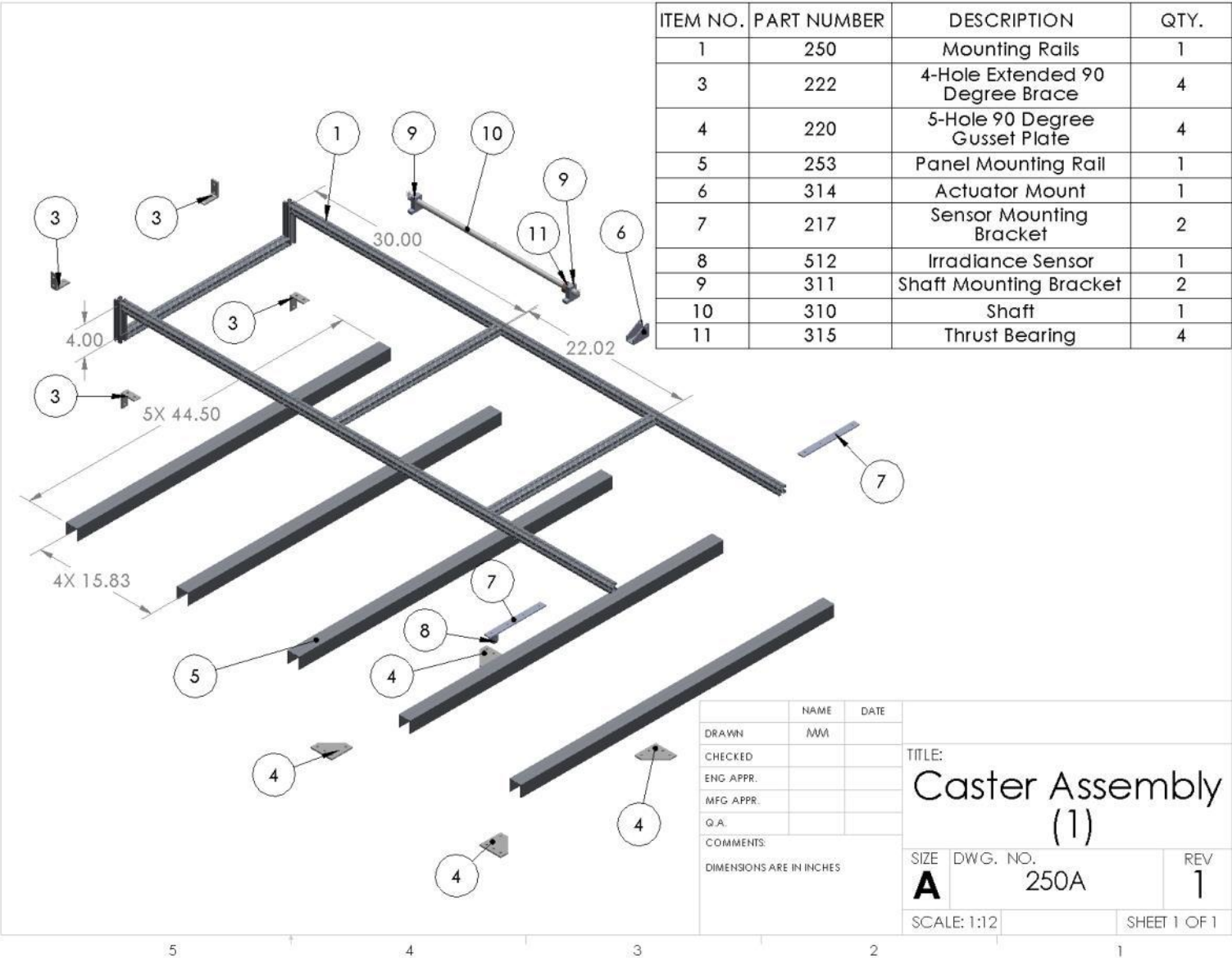
3 in. Zinc Plated Tee Hinges (2-Pack)

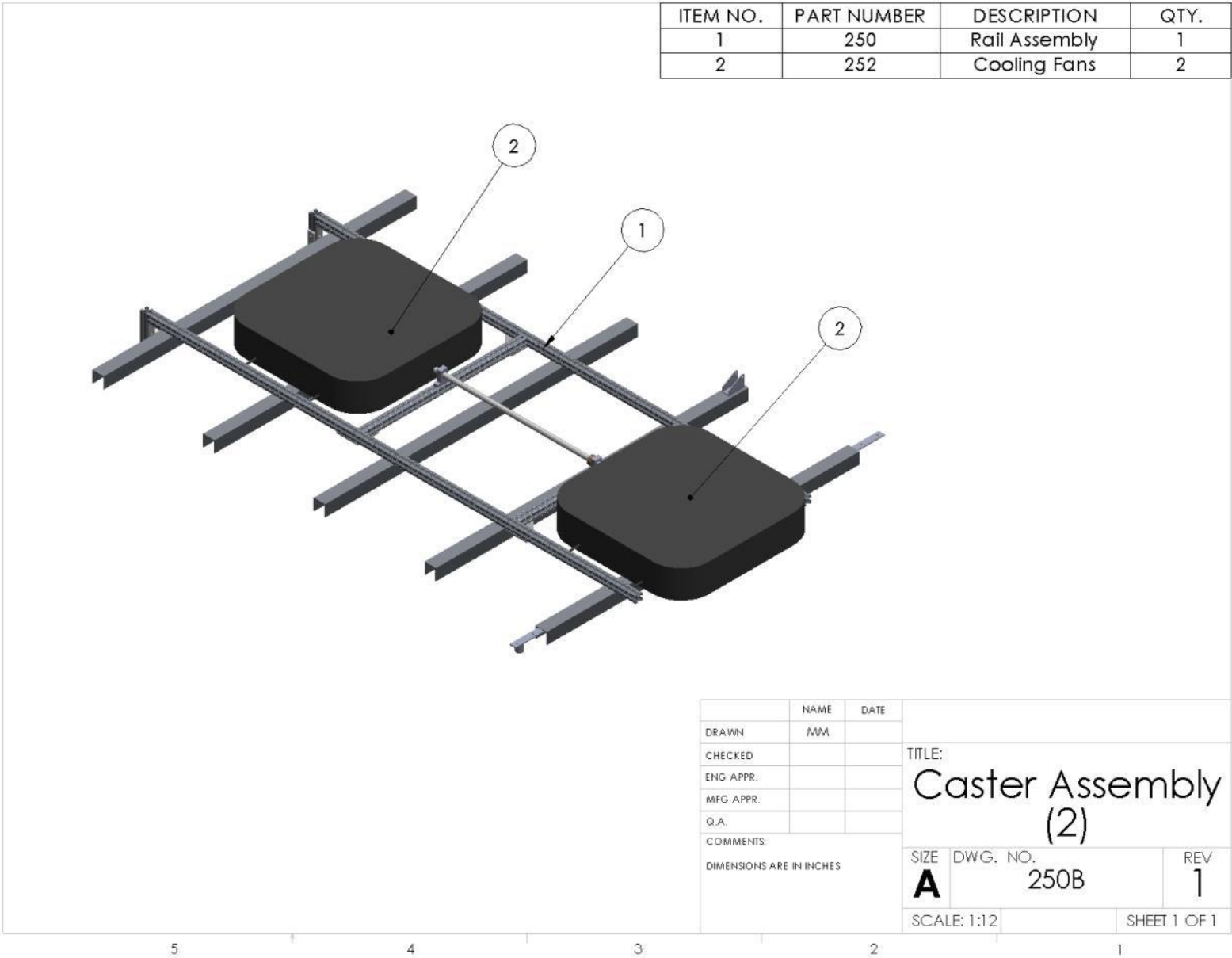


SPECIFICATIONS

Assembled Depth (in.)	0.31 in	Assembled Height (in.)	2.25 in
Assembled Width (in.)	3 in	Builders Hardware Product Type	Gate Hinge
Commercial / Residential	Residential	Commercial Hardware Type	Hinges
Fasteners included	Yes	Finish	Zinc Plated
Finish Family	Silver	Hinge type	Surface mount
Manufacturer Warranty	None	Material	Steel
Mounting type	Gate	Package Quantity	2
Pin type	Fixed	Radius (in.)	0
Returnable	90-Day		

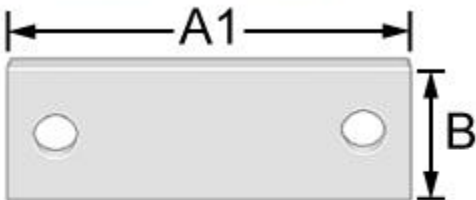






Bracket

Galvanized Steel, 8" Long



☐ Each

ADD TO ORDER

In stock
1-49 Each \$2.40
50 or more \$1.78
1394A37

PART NUMBER: 251

Also known as angle brackets, corner brackets, and mending plates, these brackets support corners and joints. They do not include mounting fasteners, except Style 2 stainless steel brackets.


Note: Prices are approximately 25% lower when you buy 50 or more of the same bracket.

Material	Galvanized Steel
Length (A1), (A2)	8"
Width (B)	7/8"
Thickness	0.14"
Screw/Nail Size	No. 10
Number of Holes	4

Holmes

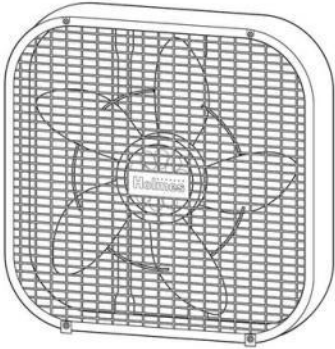
Holmes HBF2010A-WM Box Fan, White

The 20 inch blade diameter of this box fan provides cooling relief during the hot and muggy weather. The sturdy metal frame and multiple speed settings make this ideal for use in medium and large rooms. The integrated carry handle and cord wrap are available for added convenience. Three speed settings allow you to select your fan speed for optimal comfort and airflow. The compact, lightweight design allows you to save space. For added stability, the fan comes with attachable feet. This Made in the USA box fan features a highly efficient motor and blade combination that costs less than 2 cents a day to run. 1 Year Limited Warranty.



FAN TYPE	BEST USE FOR...
Stand	Large blade diameter and motorized oscillation helps to deliver powerful airflow in large living areas where there is ample floor space.
Table	Small footprint allows for use on a table or desk and is ideal for use in bedrooms or other small living areas of the home.
Window	Designed for use in most double hung and slider windows. Can be used to bring in fresh air from outdoors or to exhaust stale indoor air.
Personal	Smaller blade size is designed for personal space cooling and is ideal for use in bedrooms, office cubicles, or dorm rooms.
Tower	Slim space saving tower designs take up less room than stand fans and is ideal for use in living areas where floor space is limited.
Power	Larger blade diameter and shrouded housing is designed to provide cooling relief in large living areas.
High Velocity	Durable construction and powerful air velocity makes this ideal for use in the garage or workshop.

PRODUCT FEATURES & BENEFITS



- 3 Speed Settings**
Set and control your fan speed for optimal air flow and comfort with 3 speed settings
- Energy Efficient**
Running the fan costs less than \$0.02 per hour to operate
- Powerful, Quiet Air Circulation**
Engineered 20 Inch blade design allows for maximum airflow and circulation all while maintaining whisper quiet operation

Durable Metal Construction
The sturdy metal construction means this fan will last long, and the removable front grill allows for easy cleaning.

PART NUMBER: 252

PART NUMBER: 253

Roof Trac™

Patent #6,360,491

The Original "Top-Down" PV Mounting System.

Integrated with Tile Trac® attachments

The patented Roof Trac™ system installed with the Tile Trac® attachment method provides an ideal solution for mounting on a tile roof. Tile Trac® reduces the possibility of broken tiles and leaking roofs, allowing the installer to make structural attachments to the roof rafter. The Roof Trac™ installed with the Tile Trac® allows solar support rails to be adjusted to compensate for uneven roofs.

Roof Trac® & Fast Jack™ protected under Patent #6,360,491 & Other patents pending.
Tile Trac® protected under Patent #5,646,029.

Illustrated above is how the Tile Trac® attachment seamlessly integrates with the Roof Trac™ mounting system.

The Tile Trac® attachment system is the perfect base for surface mounting on composition roofs! It has over 10 sq. inches of base to prevent damaging delicate composition shingles.

The Roof Trac™ support rail was designed to conceal all attachment hardware. All connections are made inside the support rail hidden from view.

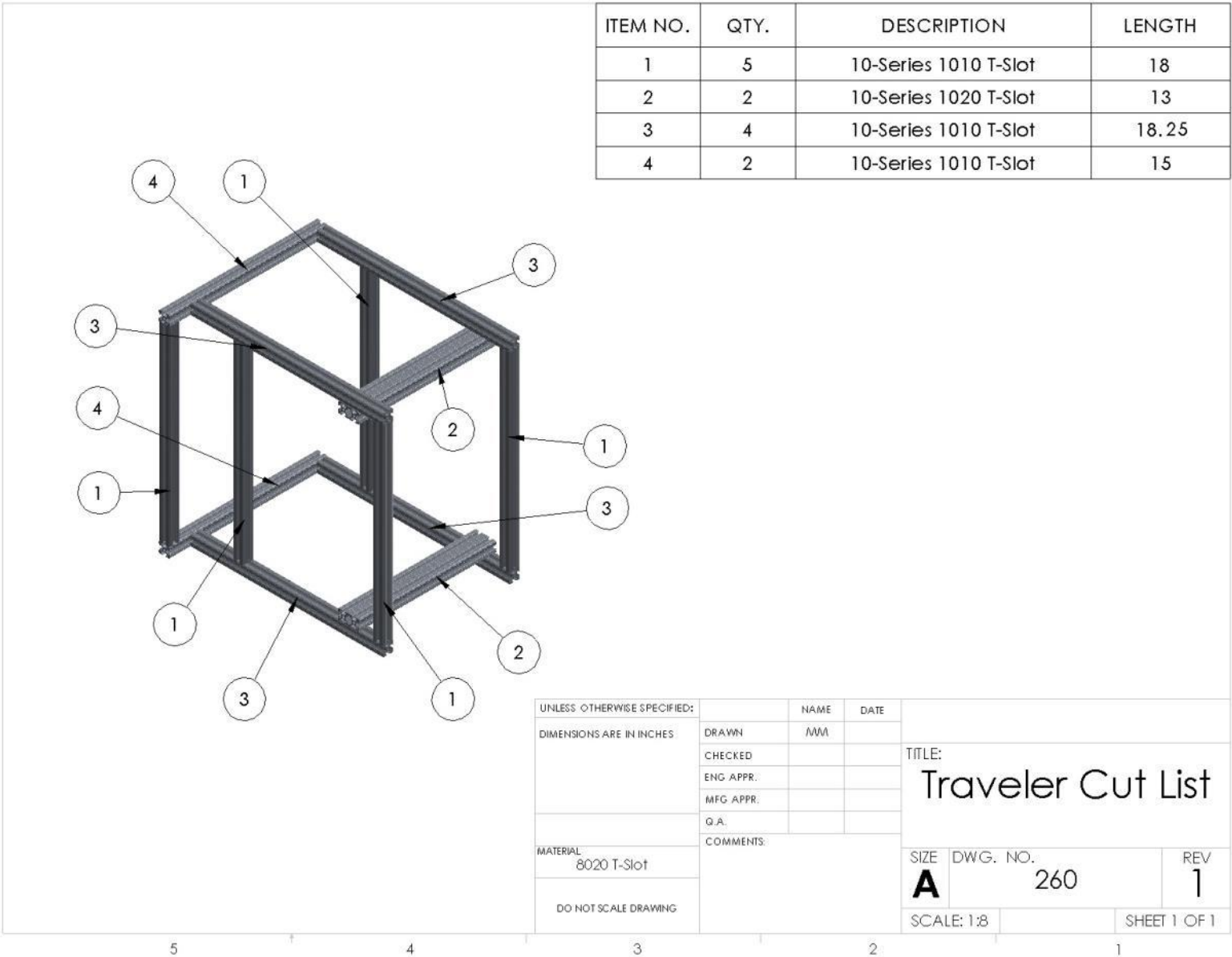
All exposed aluminum is fully anodized to module manufacturer specifications.

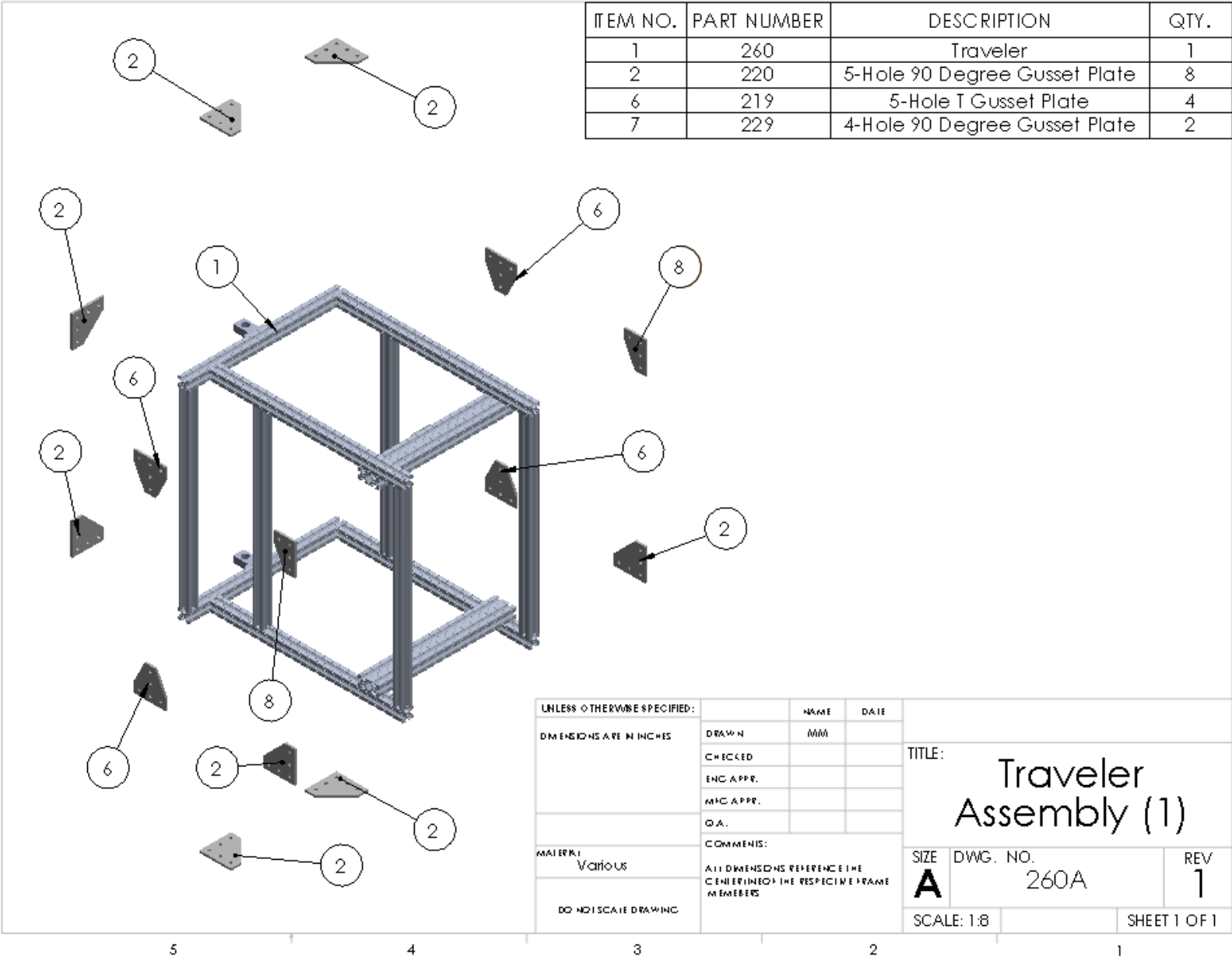
Patented Slide-N-Clamp System

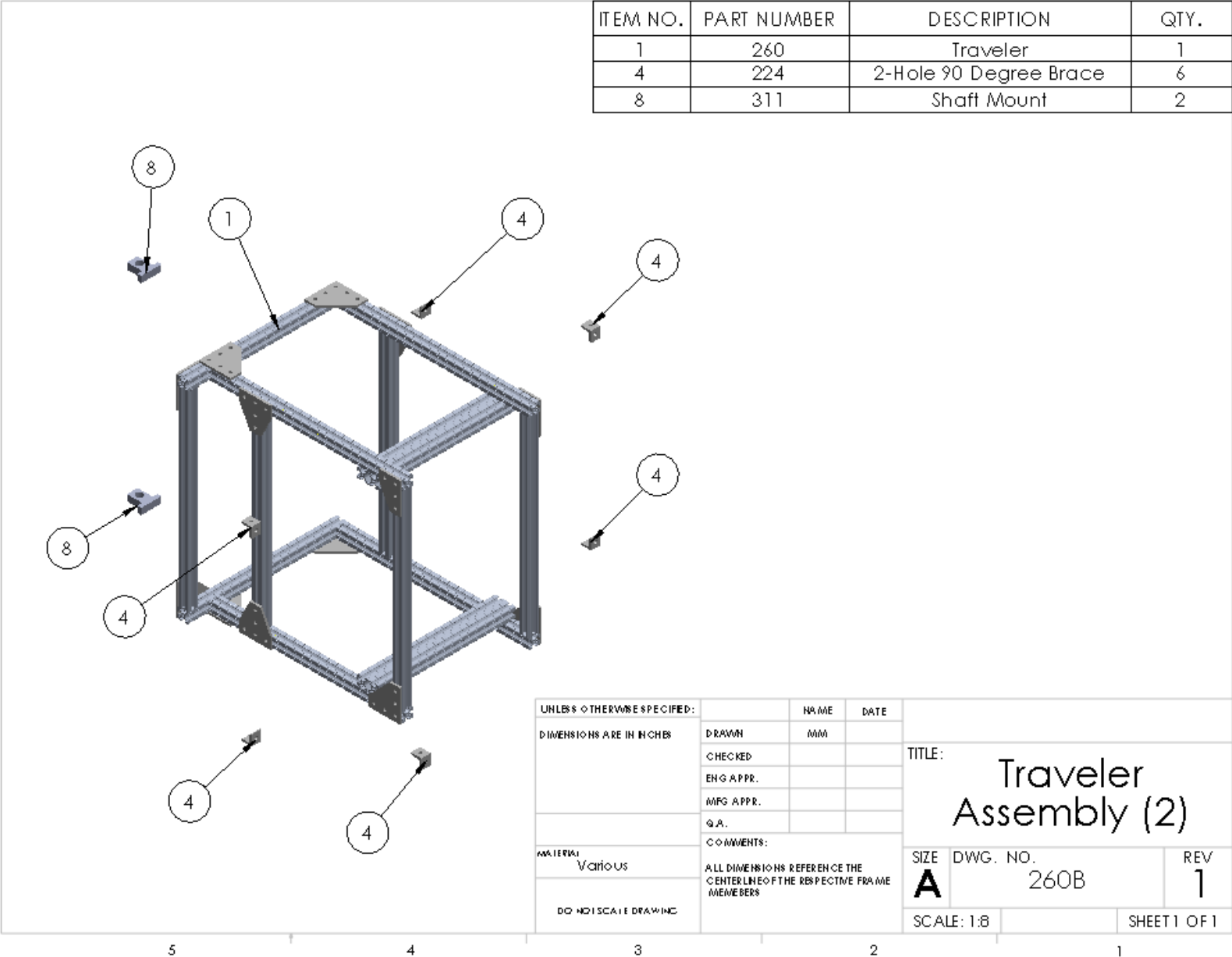
Fully integrated, and patented, clamping system actually changes the structural properties of the aluminum channel making it significantly stronger. This design allows solar modules to be installed at a lower profile to the roof providing a more aesthetically pleasing installation.

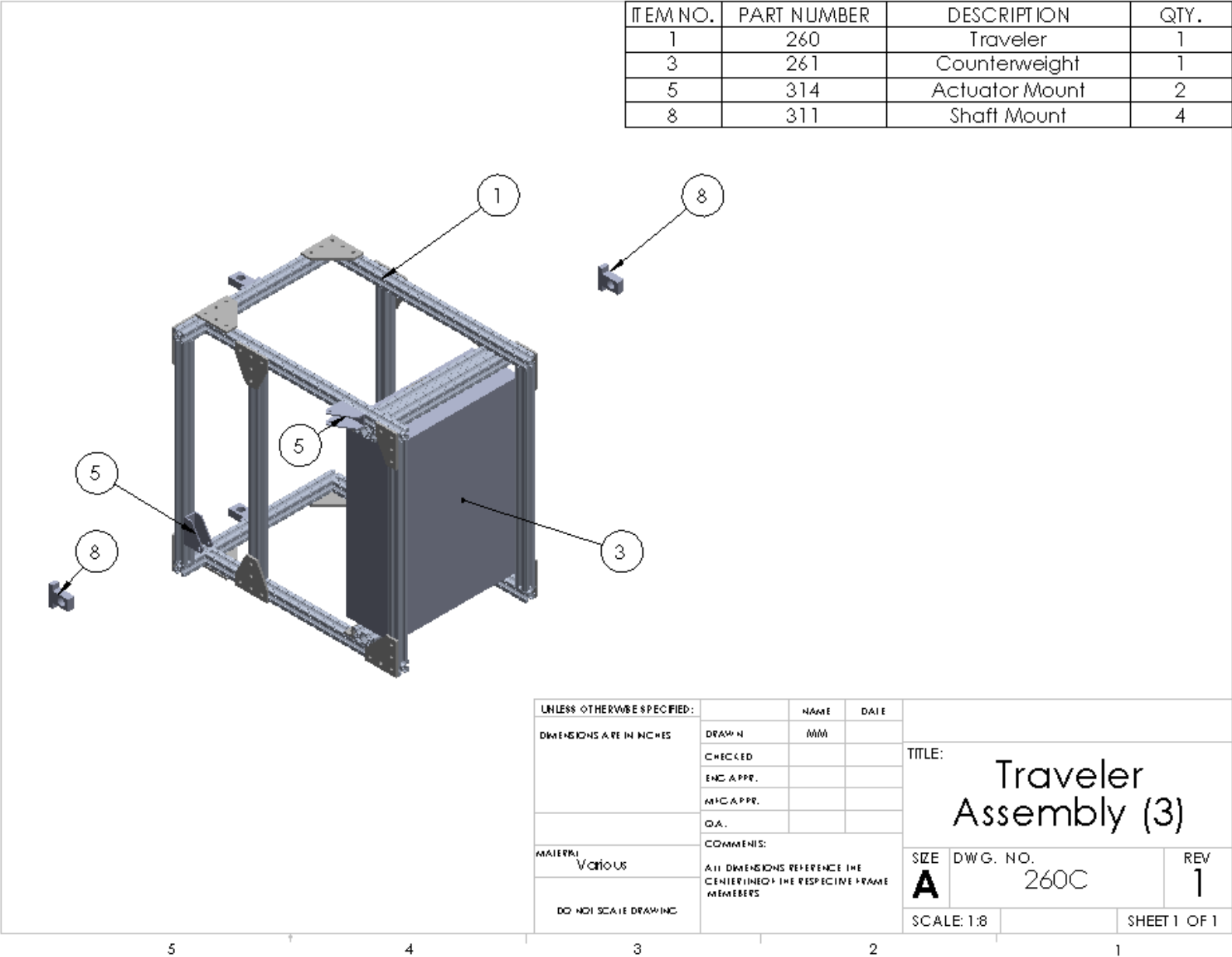
Module clamps are specifically designed, extruded and engineered for each specific module frame. Our innovative clamping system provides inward tension on the module frame securing the laminate in the frame.

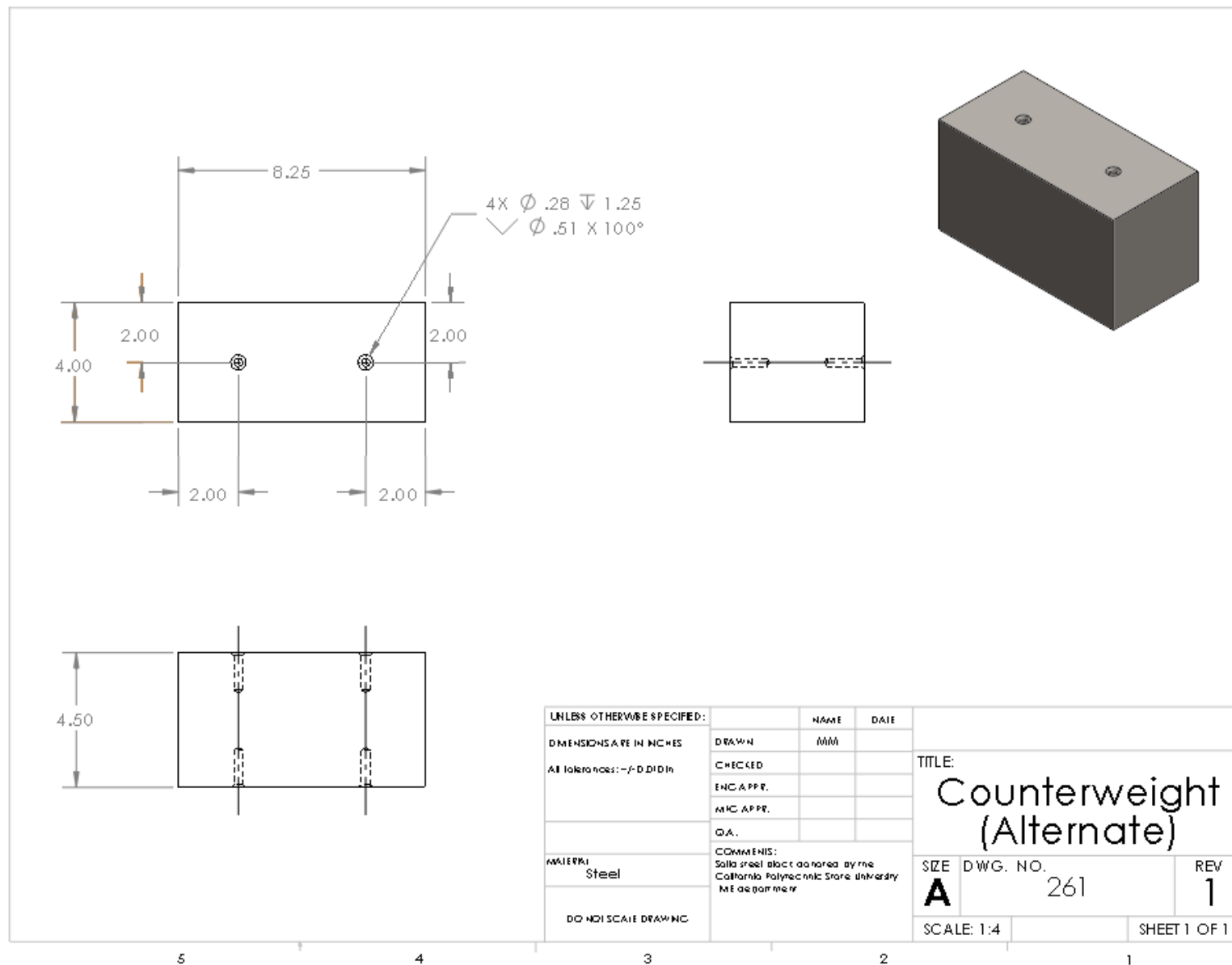
professional SOLAR products

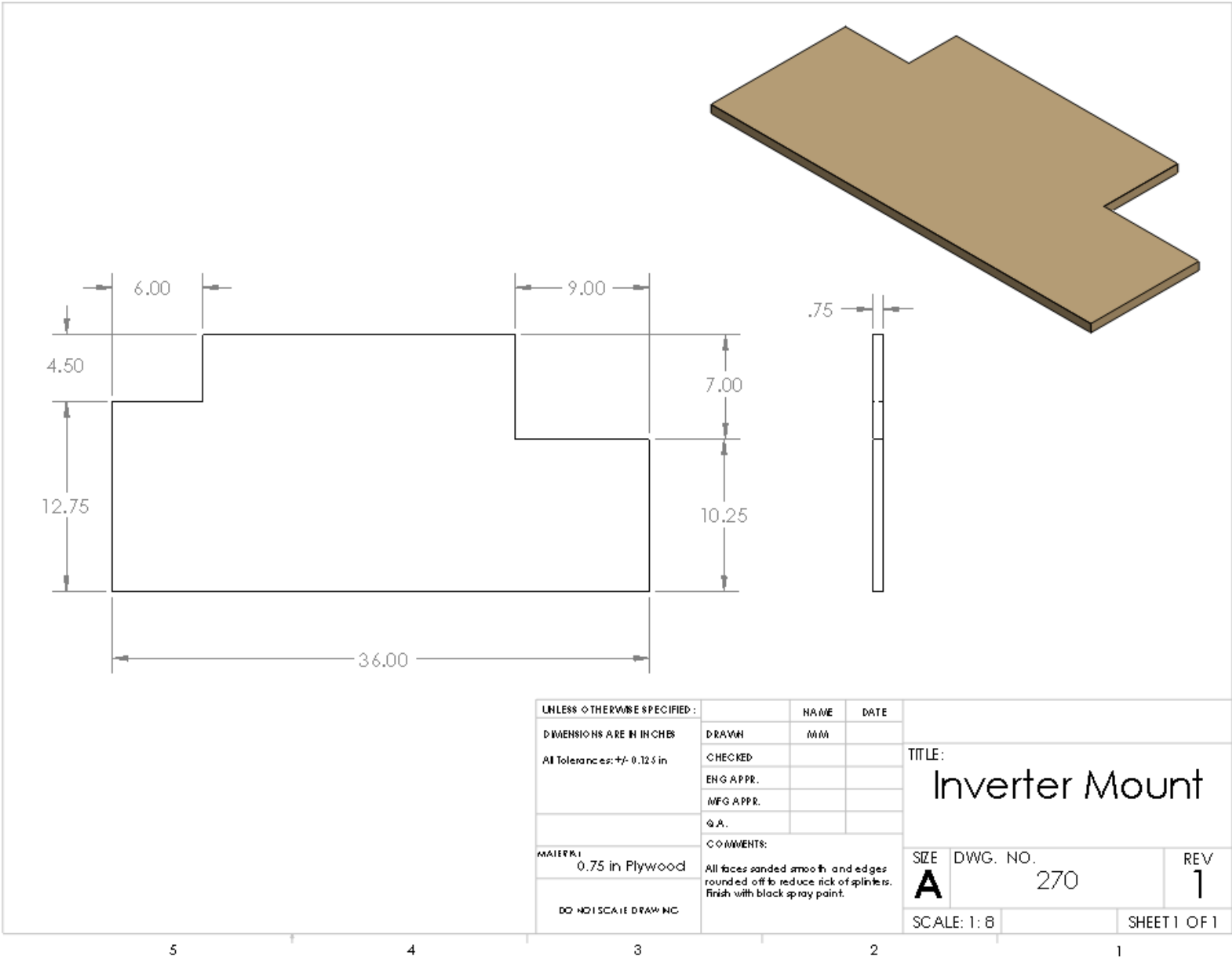




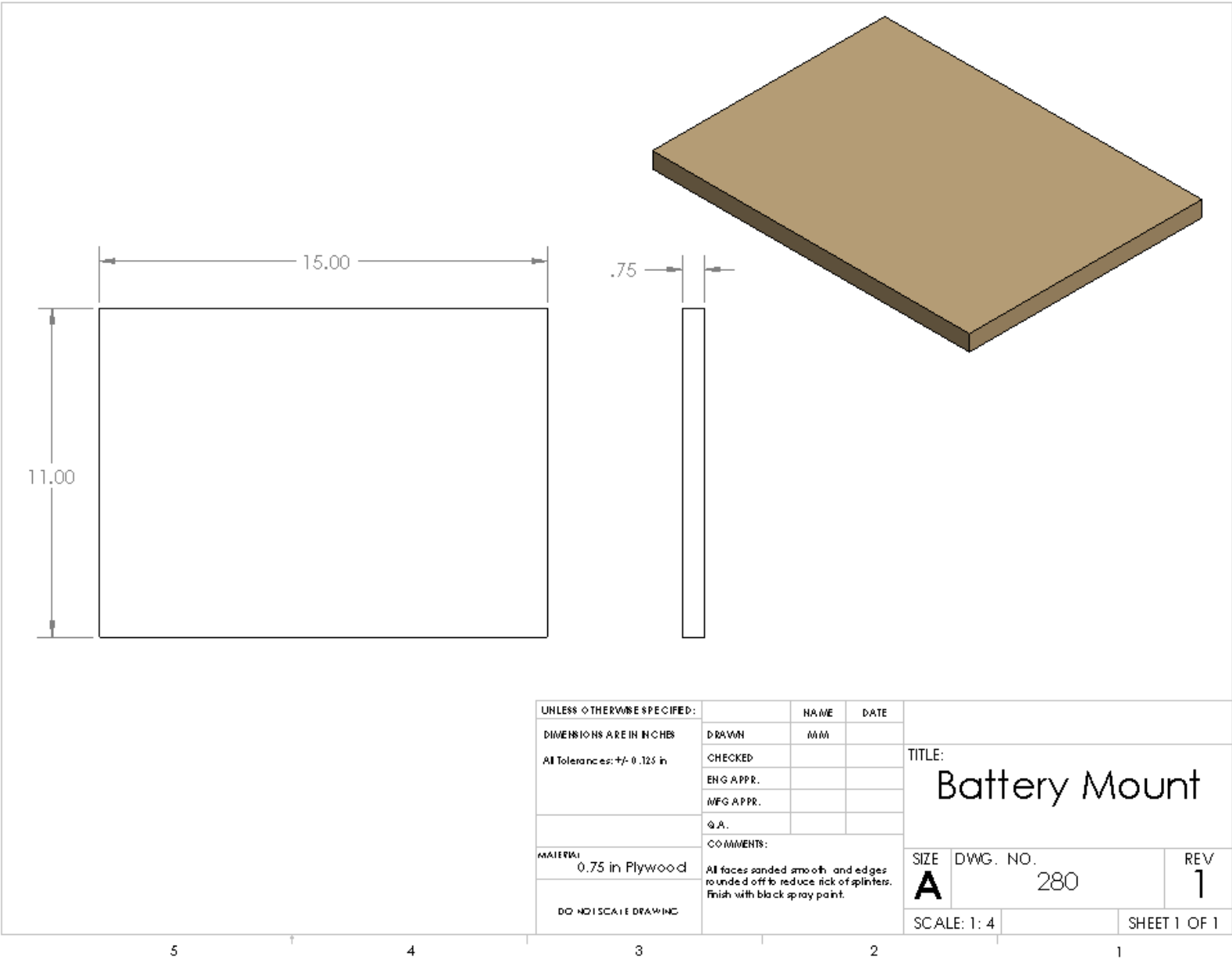


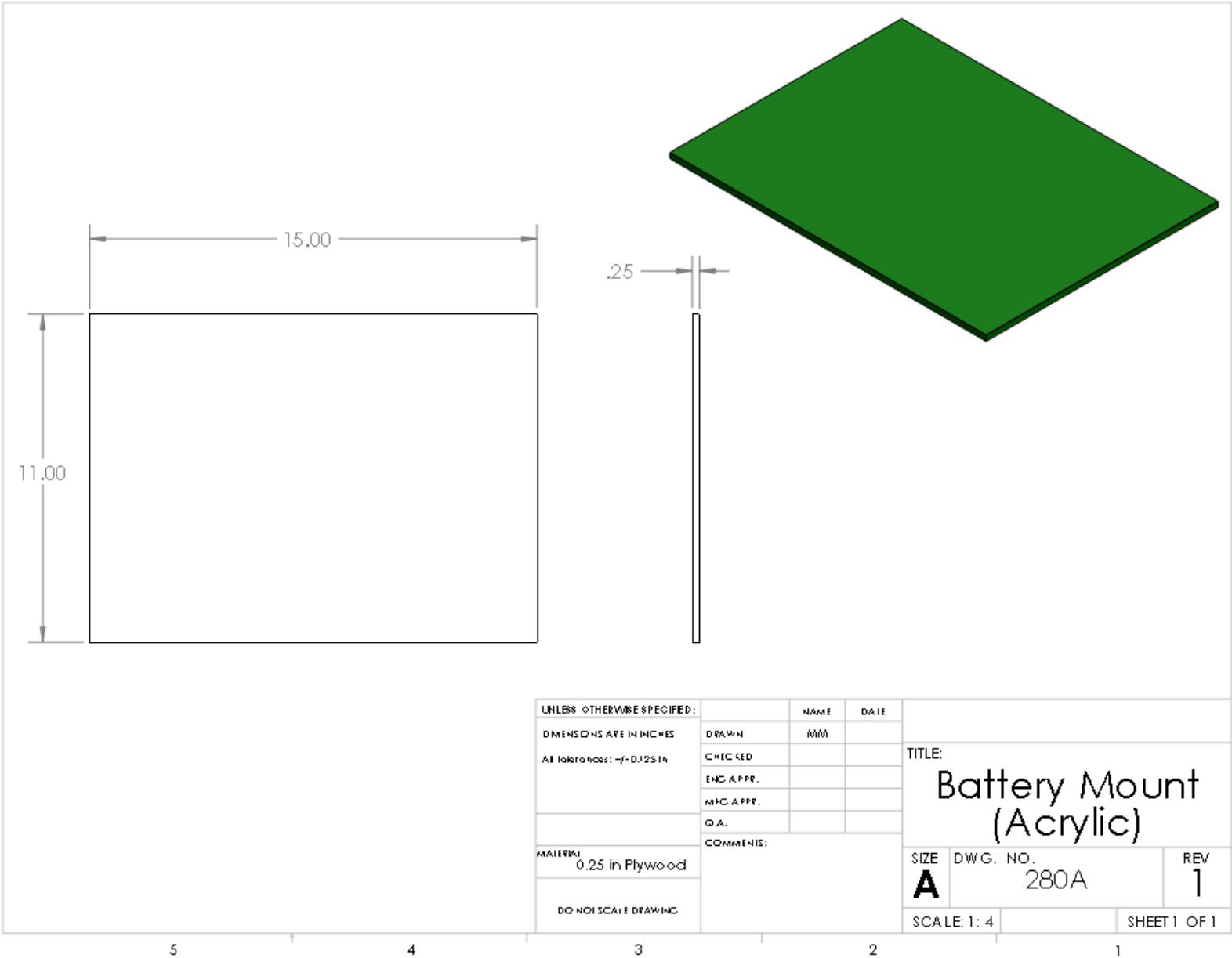


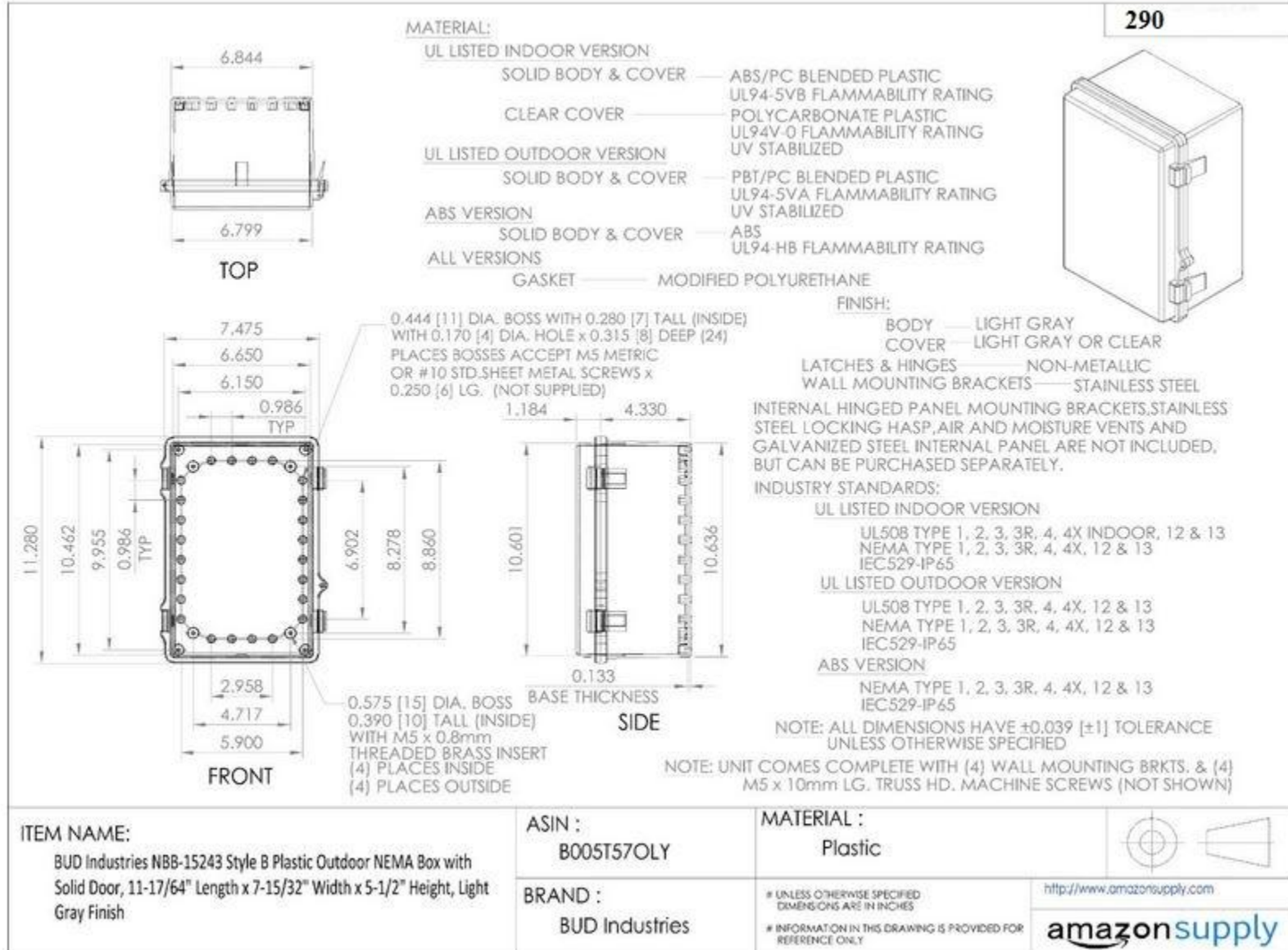


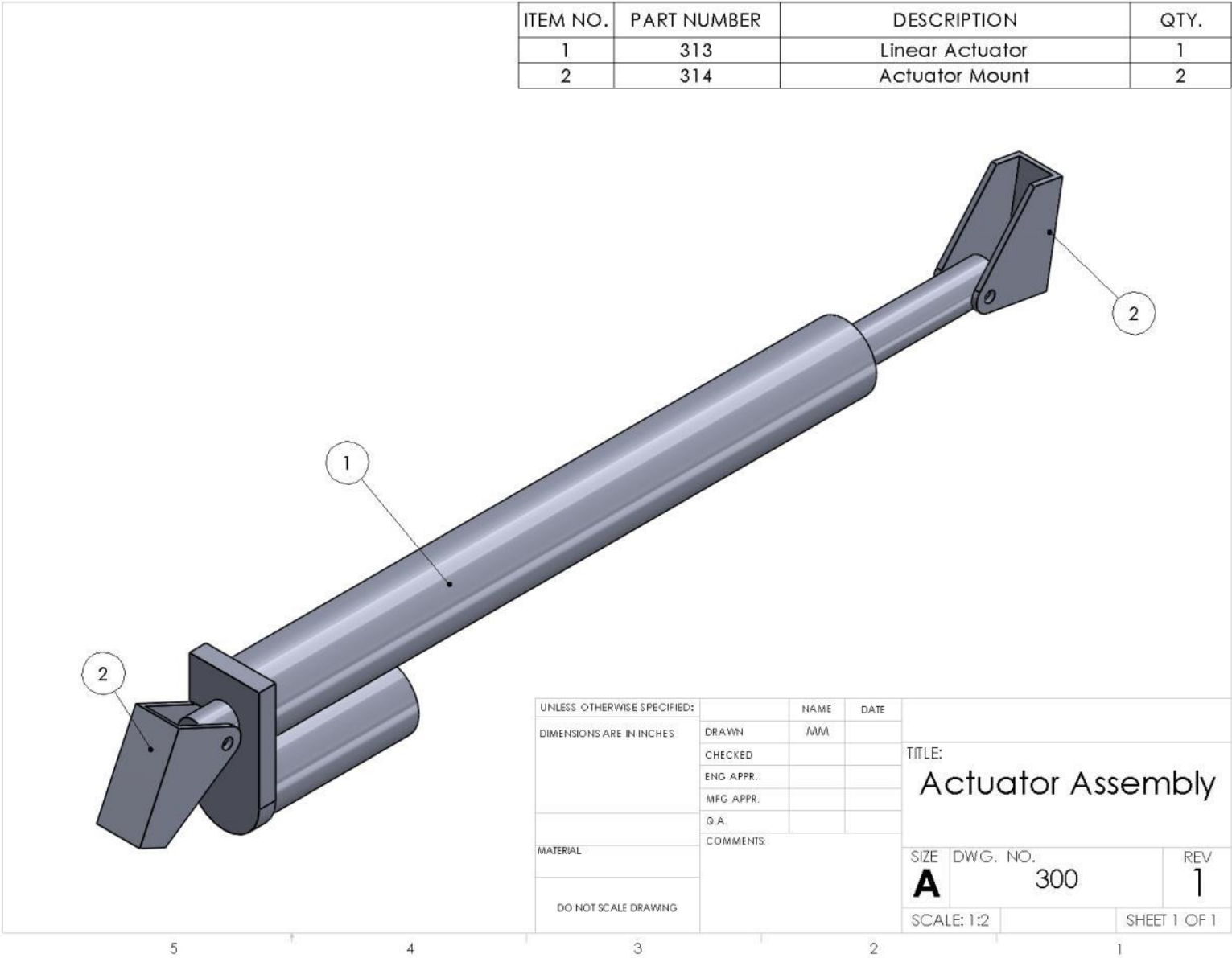


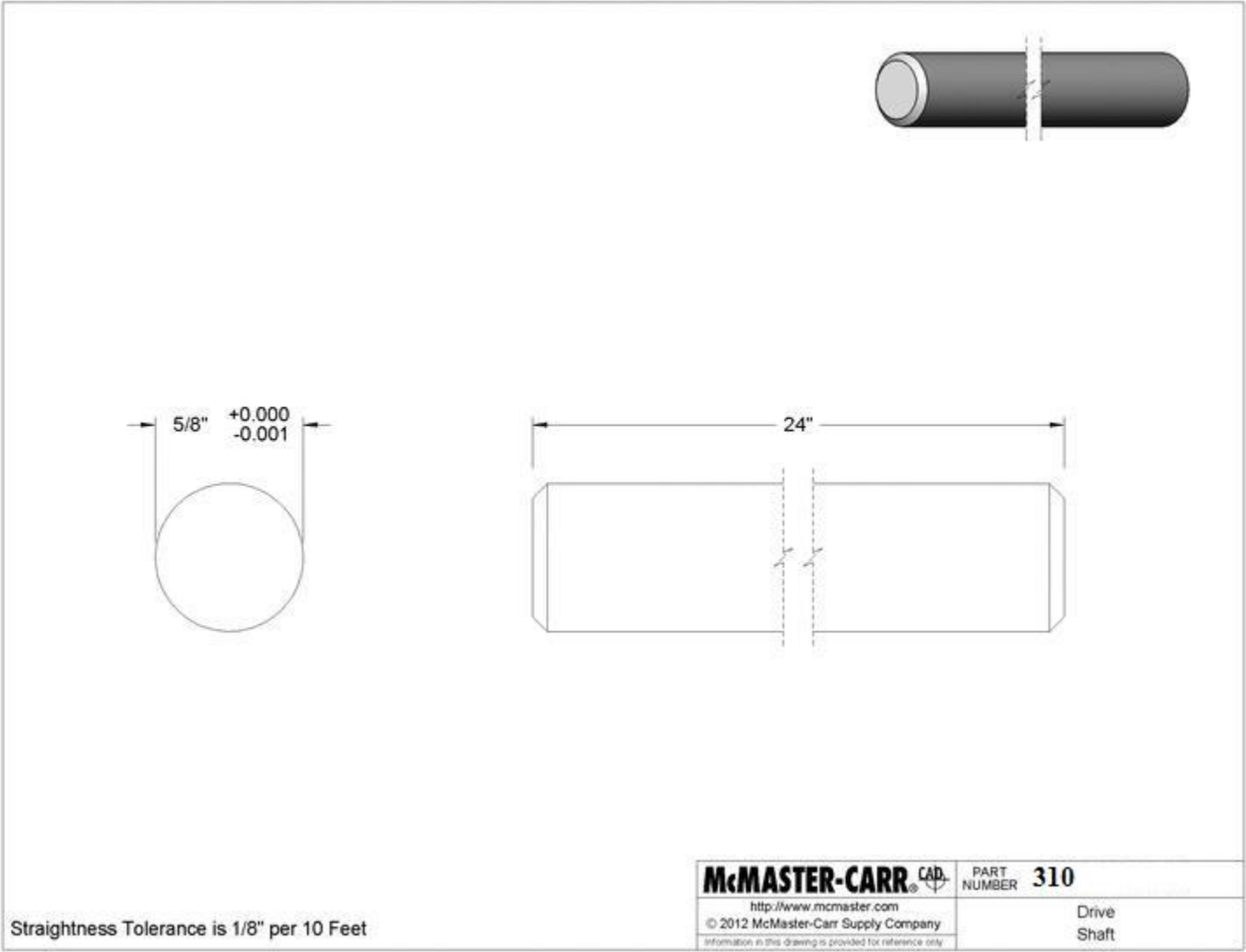




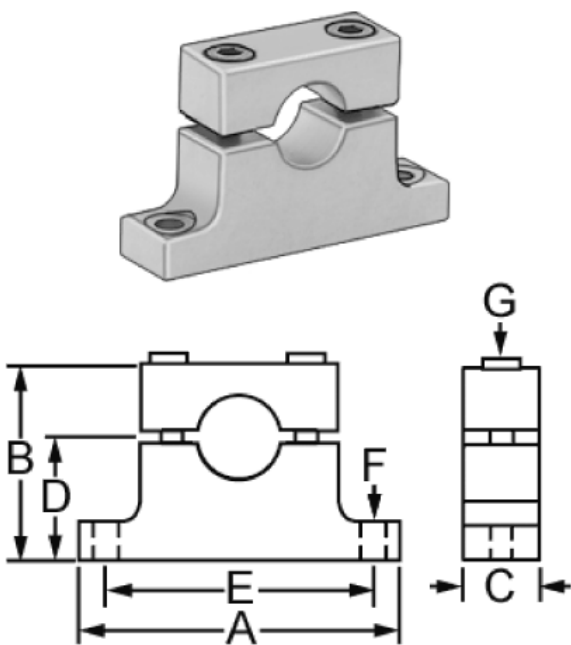








Quick-Access Base Mount Shaft Support
for 5/8" Shaft OD

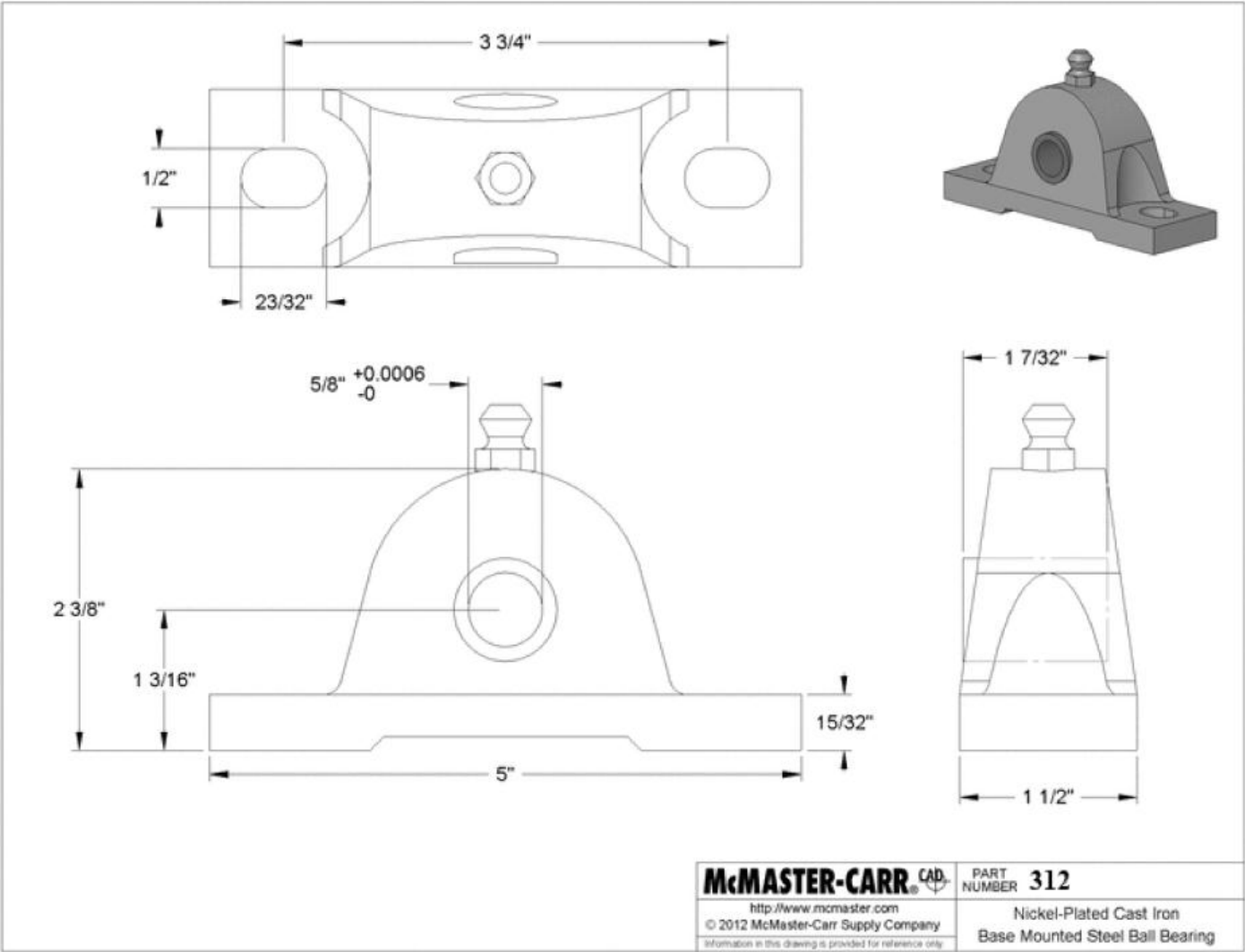


PART NUMBER: 311

For Shaft Diameter	5/8"
Overall	
Length (A)	2 5/8"
Height (B)	1 7/16"
Thickness (C)	1/2"
(D)	15/16"
(E)	2 1/8"
Mounting Hole Diameter (F)	3/16"
Bolt Size (G)	#10
Additional Specifications	Quick-Access Base Mount Aluminum

Brace the ends of your linear shafts when working with light to medium loads where shaft alignment is not critical. For use with closed linear bearings.

Quick-Access Base Mount—Innovative two-piece design lets you quickly remove the top of the support for access to your shaft. Tolerance for (D) dimension is ± 0.003 ".



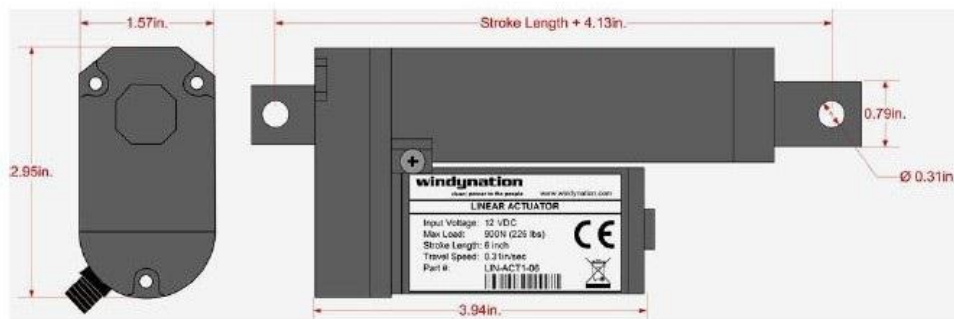
windynation

LIN-ACT1-XX Linear Actuators

PART NUMBER: 313



SPECIFICATIONS	LIN-ACT1-06	LIN-ACT1-12
Stroke Length	6" (152mm)	12" (304mm)
Rated Load	900N (225 lbs)	
Travel Speed (Max)	0.31 in/sec (8 mm/sec)	
Rated Voltage	12VDC	
Limit Switches	Fixed Inner (not adjustable)	
Operation Temperature	-20°C to +65°C	
Protection Class	IP65	
Duty Cycle	25%	
Noise Level	< 50dB	



INSTALLATION

WARNINGS:

1. The load added onto the actuator must be less than or equal to the rated load of actuator.
2. Install the actuator so that the force of the load acts in the center of the extension tube and the rear mounting adapter. Off-centered loads will cause the actuator shaft to rub against the actuator housing which will damage the actuator.
3. Do not exceed the 25% duty cycle of the actuator: If the actuator is used at full load for 2.5 minutes, then it must remain off for 7.5 minutes. Exceeding the duty cycle will cause the actuator motor to overheat.

1. Mount the actuator by securing the top and bottom mounting holes to two fixed positions. The stroke length of the actuator (e.g. 12 inches) and the limitations of the particular application will determine the location of the fixed mounting positions.

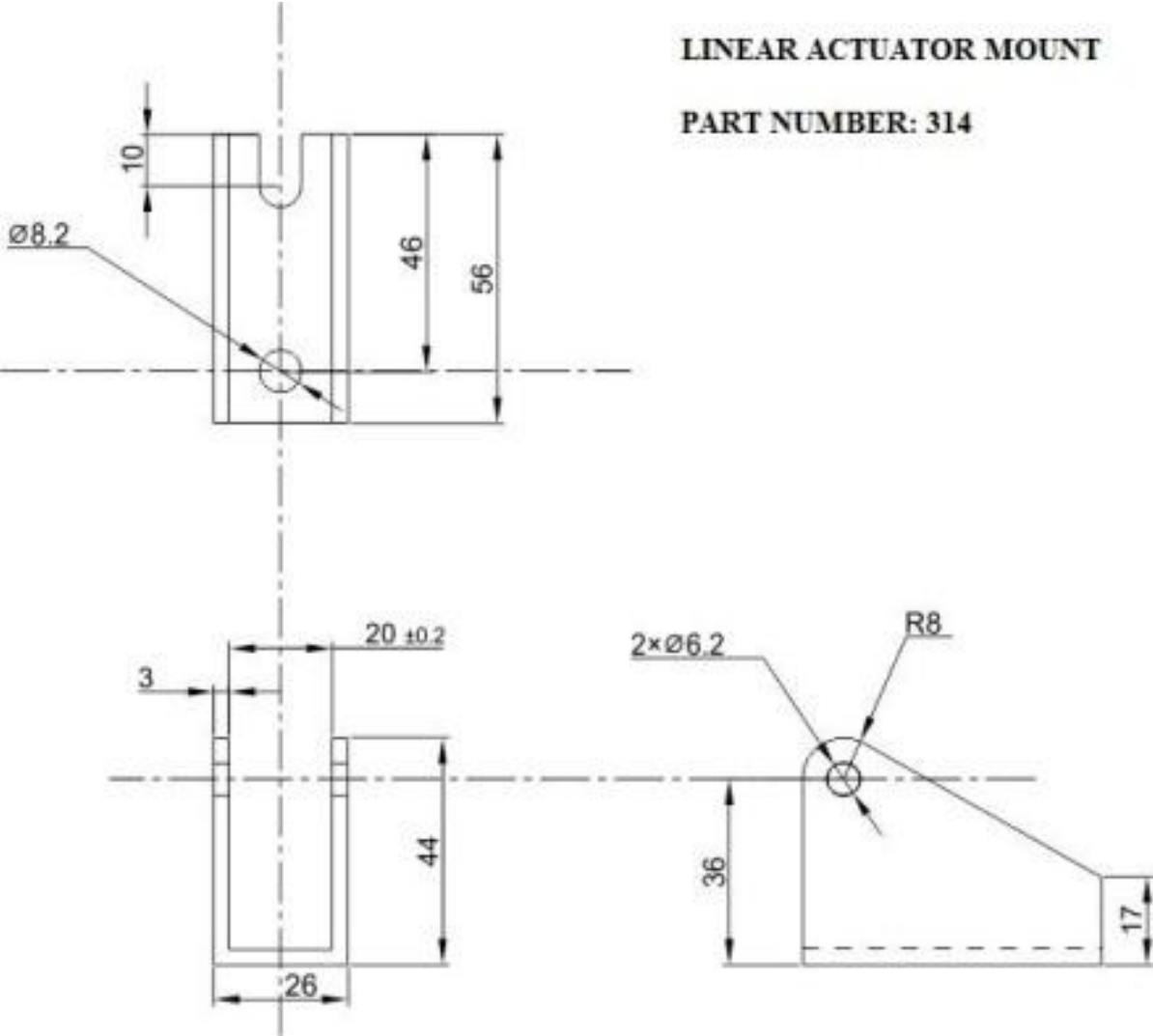
IMPORTANT: Confirm that the up/down movement of the linear actuator is smooth and within the actuator's stroke length after it has been installed.

2. Secure the top and bottom mounting holes of the linear actuator onto the two mounting fixtures using 5/16" diameter bolts.
 3. Connect the red wire to the positive post and the black wire to negative post of the 12 volt DC power supply.
 4. The operation of the linear actuator should be tested manually after the installation is completed.
- Users should use caution to ensure that
- The travel distance of the actuator satisfies the requirement of the structural design.
 - The extended and retracted limit switches operate normally (The limit switches should stop the motor when the extension tube is fully retracted or fully extended)
 - If the motor runs too slow or does not give full force, (1) the power supply is insufficient and needs to be increased or (2) the load being applied to the actuator is too great and needs to be reduced to less than or equal to 225 lbs.

WARRANTY

Windy Nation warrants the item to be free of manufacturing flaws for a period of 90 days.

Windy Nation is not responsible for any injuries and/or damages caused as a result of not complying with the specifications stated.



SAE 841 Solid Bronze Thrust Bearing
for 5/8" Shaft Diameter, 1-3/16" OD, 1/8" Thick

PART NUMBER: 315



Solid

☐ Each

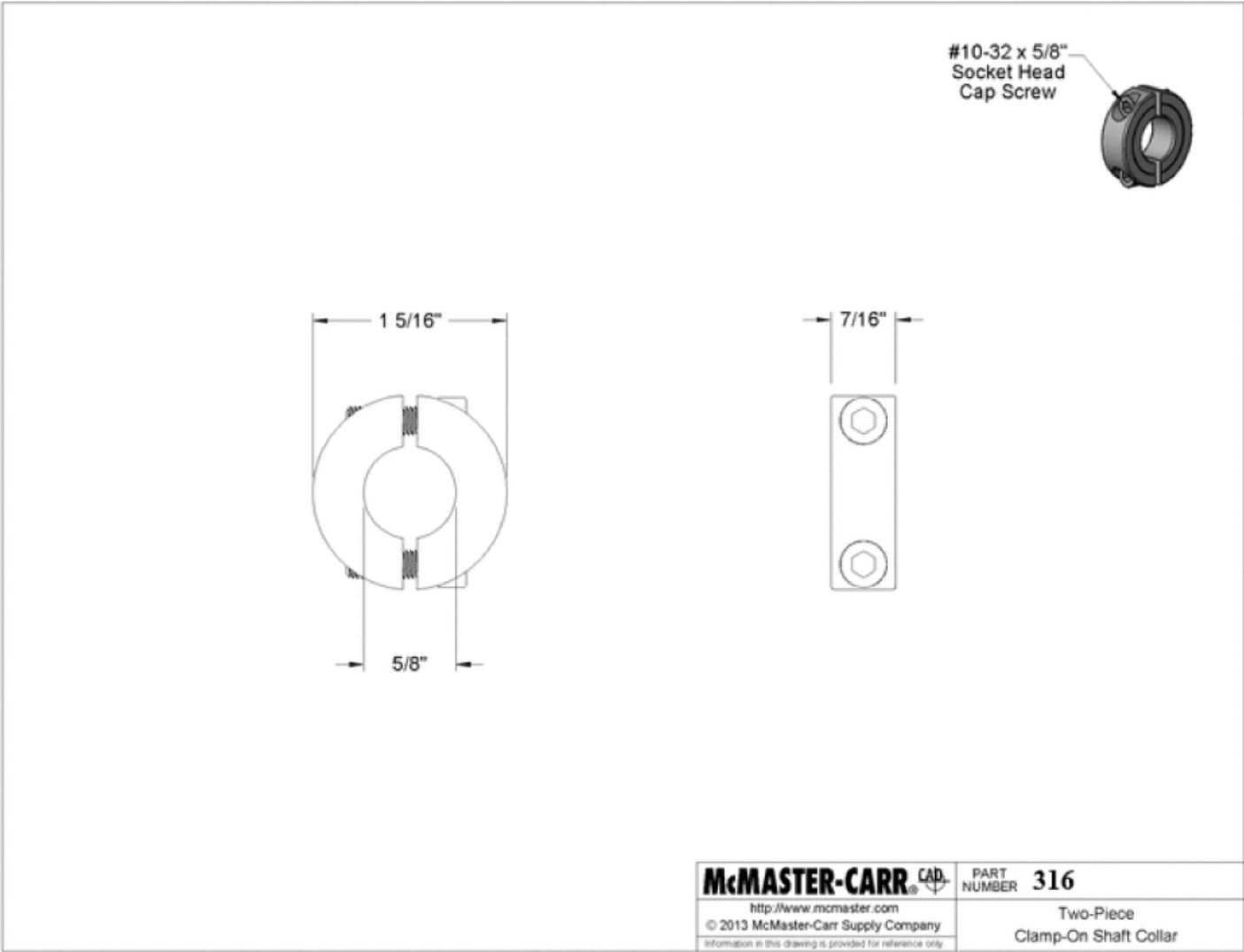
ADD TO ORDER

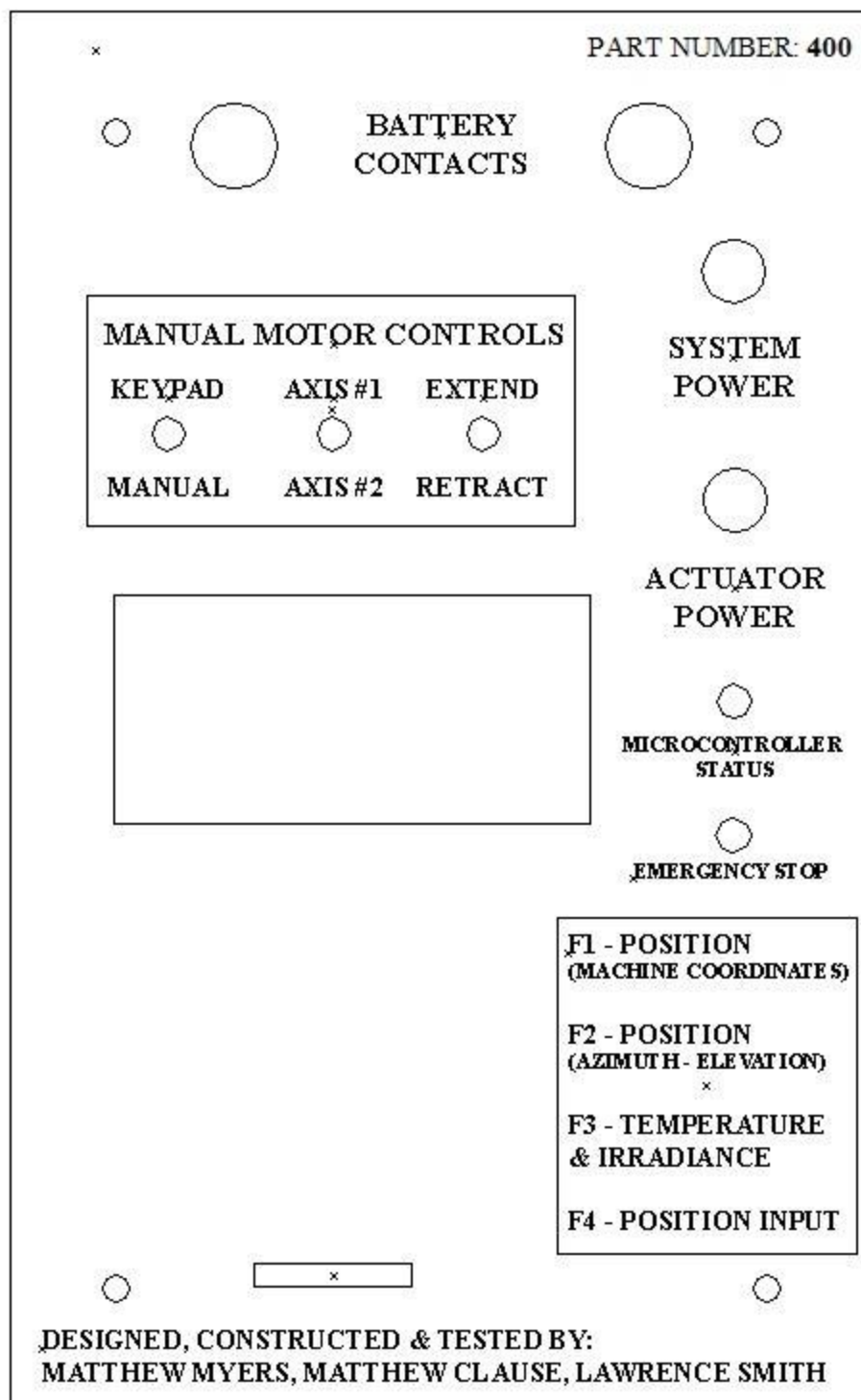
In stock
\$1.97 Each
5906K545

SAE 841—Also called Oilite®, this porous material is impregnated with 19% SAE 30 oil. The oil reduces wear as well as required maintenance.

[View tolerance information for these bearings.](#)

Bearing Material	Solid SAE 841
Temperature Range	10° to 220° F
P Maximum	2,000
V Maximum	1,200
PV Maximum	50,000
Material	SAE 841
For Shaft Diameter	5/8"
OD	1 3/16"
Thickness	1/8"
Additional Specifications	Solid





PART NUMBER: 410



SPST 12VDC/20A Illuminated Toggle Switch with Green LED

Model: 275-019 | Catalog #: 275-019

TECH SPECS

General Features	
Model	275-019
Product Type	SPST
Miscellaneous Features	
Supported Languages	English
#per pack	1
Power Features	
Voltage Required	12VDC



PART NUMBER: 411



NHD-0420D3Z-NSW-BBW-V3

Serial Liquid Crystal Display Module

NHD-	Newhaven Display
0420-	4 lines x 20 characters
D3Z-	Model
N-	Transmissive
SW-	Side White LED Backlight
B-	STN-Blue (-)
B-	6:00 view
W-	Wide Temperature (-20°C~ +70°C)
V3-	Firmware Version 3.00
	RoHS Compliant

Newhaven Display International, Inc.

2511 Technology Drive, Suite 101
Elgin IL, 60124
Ph: 847-844-8795 Fax: 847-844-8796

www.newhavendisplay.com
nhtech@newhavendisplay.com nhsales@newhavendisplay.com

PART NUMBER: 412

Leviton | Model # R51-05320-001

15 Amp Grounding Duplex Outlet - Ivory



SPECIFICATIONS

Assembled Depth (in.)	3.25 in	Assembled Height (in.)	8.5 in
Assembled Width (in.)	7 in	Box Included	Yes
Certifications and Listings	1-UL Listed	Color Family	Gray
Color Family	Gray	Color/Finish	Gray
Electrical Product Type	Outlet	Grounding	Self-Grounding
Hospital Grade	Yes	Indoor/Outdoor	Outdoor
Manufacturer Warranty	1 Year	Material	Metal
Maximum Amperage (amps)	15 A	Number of Outlets	2
Outlet Location	Wall	Outlet Type	Duplex
Package Quantity	4	Pop-Out	No
Product Depth (in.)	3.25	Product Height (in.)	8.5
Product Series	Bell	Product Weight (lb.)	1.225
Product Width (in.)	7	Returnable	90-Day
Rotating	No	Surface Mount	No
Tamper Resistant	Yes	Toggle Switch	No
USB Port	No	Voltage (volts)	120
Wall Plate Included	Yes	Weather Resistant	Yes

TACT
SWITCHESNAVIGATION
SWITCHESPUSHBUTTON
SWITCHES**TOGGLE
SWITCHES**ROCKER
SWITCHESSLIDE
SWITCHESSNAP-ACTION
SWITCHESDIP
SWITCHESKEYLOCK
SWITCHESROTARY
SWITCHESDETECTOR
SWITCHESCAP
OPTIONS

SERIES 100 SWITCHES

TOGGLE SWITCHES - MINIATURE

PART NUMBER : 413



FEATURES & BENEFITS

- ▶ Up to 4 poles available
- ▶ Variety of switching functions
- ▶ Miniature
- ▶ Multiple actuator & bushing options

APPLICATIONS/MARKETS

- ▶ Telecommunications
- ▶ Instrumentation
- ▶ Networking
- ▶ Medical equipment

SPECIFICATIONS

Contact Rating:	See contact material options
Life Expectancy:	40,000 make-and-break cycles
Contact Resistance:	10 mΩ max. typical initial @ 2-4 VDC 100 mA for both silver and gold plated contacts
Insulation Resistance:	1,000 MΩ min.
Dielectric Strength:	1,000 V RMS @ sea level
Operating Temperature:	-30° C to 85° C

MATERIALS

Case:	Diallyl Phthalate (DAP)
Toggle Handle:	Brass, chrome plated
Switch Support:	Brass or steel, tin plated
Bushing:	Brass, nickel plated
Housing:	Stainless steel
Contacts / Terminals:	Silver or gold plated copper alloy



HOW TO ORDER

SERIES	MODEL NO.	ACTUATOR	BUSHING	TERMINATION	CONTACT MATERIAL	SEAL	HARDWARE
100	□ □ □	□ □ □	□ □ □	□ □ □ □	□	E	□
	SP1 SP2* SP3 SP4* SP5* DP1 DP2* DP3 DP4* DP5* DP6 DP7* DP8*	T1† T2 T3 T4 T5 T6 T7 T8 T9 K1 K2	B1 B2 B3 B4 B5 B6 B8 B9 B11 B12 B13 B15 B16 B25** B26**	M1 M2 M3 M5 M6 M7 M71 VS2 VS3 VS5* M64* VS21* VS31*	Q = Silver R = Gold	E = Epoxy Sealed at Base of Terminal	H = Hardware
	3P1 3P2* 3P3 3P4* 3P5* 4P1 4P2* 4P3 4P4* 4P5* 4P6 4P7* 4P8*	† Optional Toggle Cap (T1 Only): T100-1 = White T100-2 = Black					



Example Ordering Number

100-SP1-T4-B2-M1-R-E

Notes: * Not available with the K1 and K2 actuator options.

** Available only with the K1 and K2 actuator options.

Specifications subject to change without notice.

PART NUMBER: 414



Self-locking Type Emergency Stop Red Mushroom Push Switch

by Amico

★★★★★ 3 customer reviews

Price: \$7.02 **Prime**

Note: This item may be available at a lower price from other sellers that are not eligible for Amazon Prime.

In Stock.Sold by **uxcell** and Fulfilled by Amazon. Gift-wrap available.

Want it tomorrow, June 6? Order within **14 hrs 10 mins** and choose **One-Day Shipping** at checkout. [Details](#)

- Type: Push Locking In & Clockwise Direction Move Reset; Contact Type: One Open One Close
- Ith Heating Conventional Current: 10A; Ui Rated Insulation Voltage: AC 660V
- Operation Category: AC15, DC13; IEC: 947; Protective Class: IP57
- Size: 3" x 1.25" x 1.2" (L x W x D); Panel Hole Diameter (Approx.): 2.2cm / 0.86"
- Panel Hole Depth (Approx.): 3.7cm / 1.45"; Mushroom Diameter: 1.5"

› [See more product details](#)



Amico Universal 4x4 16 Key Matrix Membrane Switch Keypad Keyboard 76x69x0.8mm

by Amico

Be the first to review this item

Price: **\$6.22** & **FREE Shipping** on orders over \$35. [Details](#)

Only 16 left in stock.

Sold by uxcell and Fulfilled by Amazon. Gift-wrap available.

Want it Monday, Feb. 10? Order within **38 hrs 40 mins** and choose **One-Day Shipping** at checkout. [Details](#)

- Product Name : Membrane Switch;Key Type : 4x4;Voltage : DC 12V
- Current : 30mA;Contact Resistance : ≥ 100 Ohm;Force : 250-350g
- Contact Bounce : $> 1s$;Pad Size : 76 x 69 x 0.8mm/3" x 2.7" x 0.03"(L*W*T);Cable Length : 72mm/2.8"
- Connector Pitch : 2.54mm/0.1";Material : Plastic;Color : As Picture Shown
- Weight : 9g;Package Content : 1 x Membrane Switch

[See more product details](#)

3 new from **\$3.49**

PART NUMBER: 415

PART NUMBER: 416



TECH SPECS

12VDC/10A DPDT Plug-in Relay

Model: 275-218 | Catalog #: 275-218

General Features

Model	275-218
Product Type	DPDT
Enclosure Color	Black/Silver
Body Material	Multi

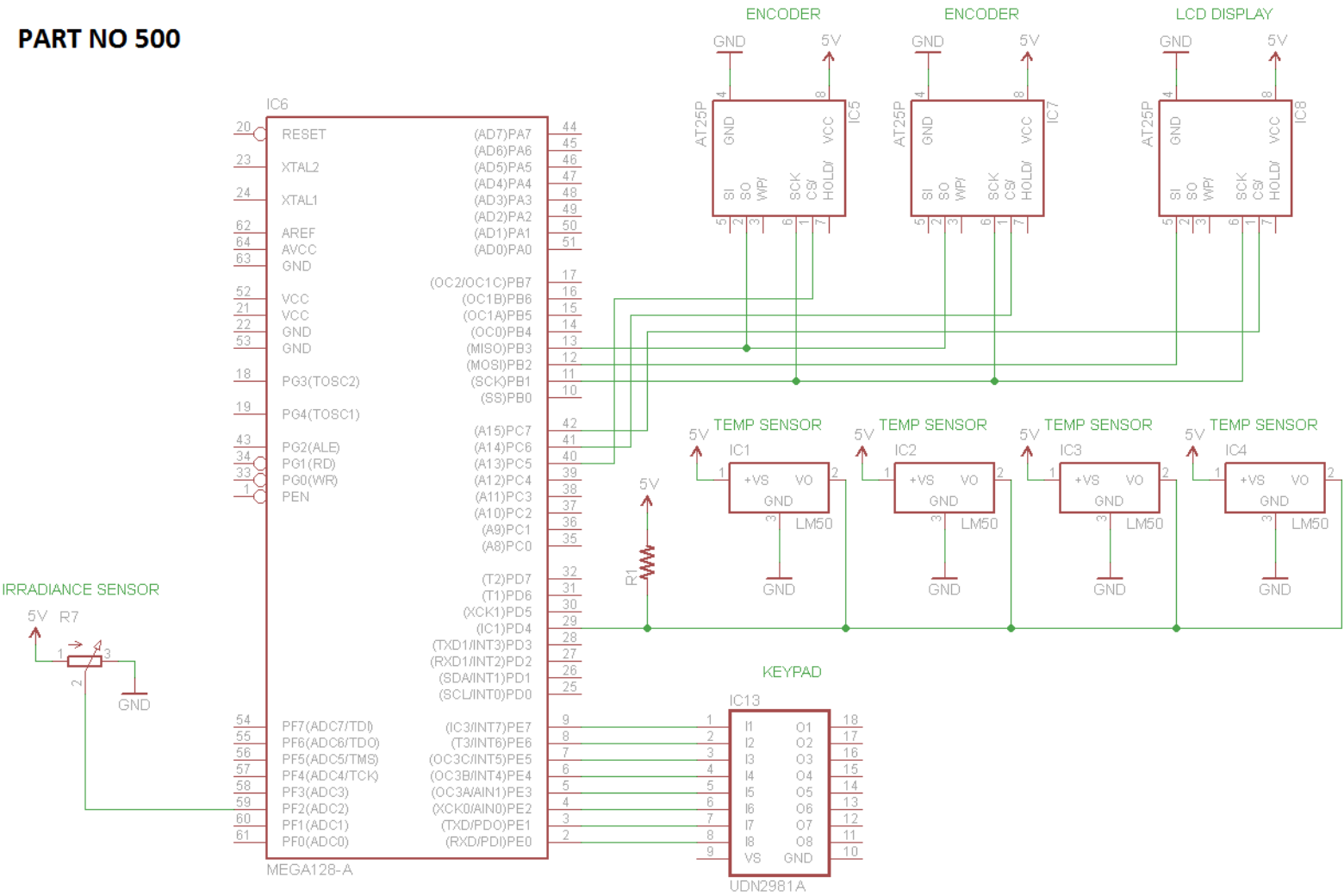
Fits What

Model	275-218
-------	---------

Miscellaneous Features

Supported Languages	English
Mounting Kit	Uses 275-220 Mounting kit.
#per pack	1

PART NO 500



PART NUMBER: 510

Features

- High Performance, Low Power Atmel® AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
 - 135 Powerful Instructions – Most Single Clock Cycle Execution
 - 32 x 8 General Purpose Working Registers
 - Fully Static Operation
 - Up to 16 MIPS Throughput at 16MHz
 - On-Chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
 - 64K/128K/256KBytes of In-System Self-Programmable Flash
 - 4Kbytes EEPROM
 - 8Kbytes Internal SRAM
 - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
 - Data retention: 20 years at 85°C/100 years at 25°C
 - Optional Boot Code Section with Independent Lock Bits
 - In-System Programming by On-chip Boot Program
 - True Read-While-Write Operation
 - Programming Lock for Software Security
 - Endurance: Up to 64Kbytes Optional External Memory Space
- Atmel® QTouch® library support
 - Capacitive touch buttons, sliders and wheels
 - QTouch and QMatrix® acquisition
 - Up to 64 sense channels
- JTAG (IEEE std. 1149.1 compliant) Interface
 - Boundary-scan Capabilities According to the JTAG Standard
 - Extensive On-chip Debug Support
 - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
 - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
 - Four 16-bit Timer/Counter with Separate Prescaler, Compare- and Capture Mode
 - Real Time Counter with Separate Oscillator
 - Four 8-bit PWM Channels
 - Six/Twelve PWM Channels with Programmable Resolution from 2 to 16 Bits (ATmega1281/2561, ATmega640/1280/2560)
 - Output Compare Modulator
 - 8/16-channel, 10-bit ADC (ATmega1281/2561, ATmega640/1280/2560)
 - Two/Four Programmable Serial USART (ATmega1281/2561, ATmega640/1280/2560)
 - Master/Slave SPI Serial Interface
 - Byte Oriented 2-wire Serial Interface
 - Programmable Watchdog Timer with Separate On-chip Oscillator
 - On-chip Analog Comparator
 - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
 - Power-on Reset and Programmable Brown-out Detection
 - Internal Calibrated Oscillator
 - External and Internal Interrupt Sources
 - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
 - 54/86 Programmable I/O Lines (ATmega1281/2561, ATmega640/1280/2560)
 - 64-pad QFN/MLF, 64-lead TQFP (ATmega1281/2561)
 - 100-lead TQFP, 100-ball CBGA (ATmega640/1280/2560)
 - RoHS/Fully Green
- Temperature Range:
 - -40°C to 85°C Industrial
- Ultra-Low Power Consumption
 - Active Mode: 1MHz, 1.8V: 500µA
 - Power-down Mode: 0.1µA at 1.8V
- Speed Grade:
 - ATmega640V/ATmega1280V/ATmega1281V:
 - 0 - 4MHz @ 1.8V - 5.5V, 0 - 8MHz @ 2.7V - 5.5V
 - ATmega2560V/ATmega2561V:
 - 0 - 2MHz @ 1.8V - 5.5V, 0 - 8MHz @ 2.7V - 5.5V
 - ATmega640/ATmega1280/ATmega1281:
 - 0 - 8MHz @ 2.7V - 5.5V, 0 - 16MHz @ 4.5V - 5.5V
 - ATmega2560/ATmega2561:
 - 0 - 16MHz @ 4.5V - 5.5V



**8-bit Atmel
Microcontroller
with
64K/128K/256K
Bytes In-System
Programmable
Flash**

**ATmega640/V
ATmega1280/V
ATmega1281/V
ATmega2560/V
ATmega2561/V**

2549P-AVR-10/2012



AEAT-6010/6012 Magnetic Encoder

10 or 12 bit Angular Detection Device



PART NUMBER: 511

Data Sheet

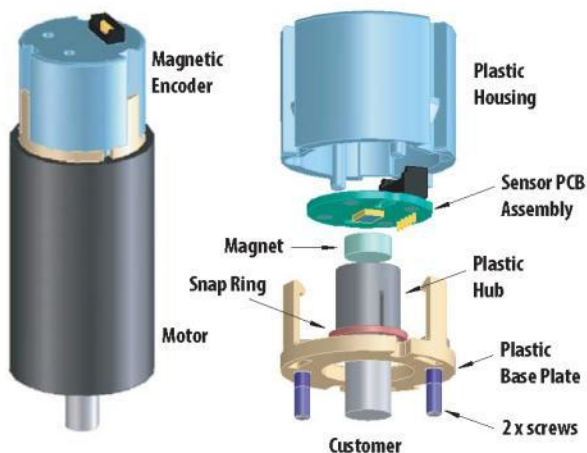
Description

Avago Technologies' AEAT-60xx series of magnetic encoders provides an integrated solution for angular detection. With ease of use in mind, these magnetic encoders are ideal for angular detection within 360°. Based on magnetic technologies, the device is non-contact and ensures reliable operations. It is able to provide absolute angle detection upon power-up, with a resolution of 0.0879°(12 bits version) or 0.35°(10bits version), which is equivalent to 4096 and 1024 positions per revolution respectively. The positional data is provided in serial bit stream. There is no upper speed limit; the only restriction is that there will be fewer samples per revolution as the speed increases.

Features

- 10 or 12 bits resolution
- Contactless sensing technologies
- Wide temperature range from -40° to 125°C
- Absolute angular position detection
- Synchronous serial interface (SSI) output for absolute position data (binary format)
- Code monotony error = ± 1 LSB
- 5V supply
- Easy Assembly, No Signal Adjustment required
- RoHS compliant

Exploded View

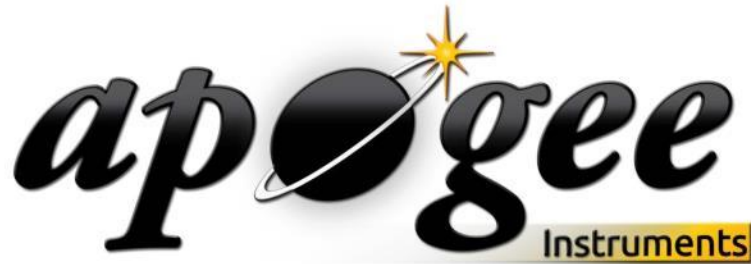


Applications

- Flow meter
- Angular detection
- Knob control
- Rotary encoder

Note: "This product is not specifically designed or manufactured for use in any specific device. Customers are solely responsible for determining the suitability of this product for its intended application and solely liable for all loss, damage, expense or liability in connection with such use."

PART NUMBER : **512**



OWNER'S MANUAL

PYRANOMETER

Model SP-212 and SP-215



APOGEE INSTRUMENTS, INC. 721 WEST 1800 NORTH, LOGAN, UTAH 84321, USA
TEL: (435) 792-4700 FAX: (435) 787-8268 WEB: APOGEEINSTRUMENTS.COM

Copyright © 2013 Apogee Instruments, Inc.

GE | Model # 76572

25 ft. Dual Jack Line Cord - White

PART NUMBER: 513



GE | Model # 76572

25 ft. Dual Jack Line Cord - White

SPECIFICATIONS

Assembled Depth (in.)	.5 in	Assembled Height (in.)	300 in
Assembled Width (in.)	.375 in	Cable Type	Telephone Wire
Certifications and Listings	No Certifications or Listings	Color Family	Stainless Look
Cord Length (ft.)	25	Electrical Product Type	Telephone Cord
Electronics Features	No Additional Features	Manufacturer Warranty	Limited Lifetime Warranty
Product Length (ft.)	25	Returnable	90-Day



PART NUMBER: 514

DS18B20 Programmable Resolution 1-Wire Digital Thermometer

DESCRIPTION

The DS18B20 digital thermometer provides 9-bit to 12-bit Celsius temperature measurements and has an alarm function with nonvolatile user-programmable upper and lower trigger points. The DS18B20 communicates over a 1-Wire bus that by definition requires only one data line (and ground) for communication with a central microprocessor. It has an operating temperature range of -55°C to $+125^{\circ}\text{C}$ and is accurate to $\pm 0.5^{\circ}\text{C}$ over the range of -10°C to $+85^{\circ}\text{C}$. In addition, the DS18B20 can derive power directly from the data line ("parasite power"), eliminating the need for an external power supply.

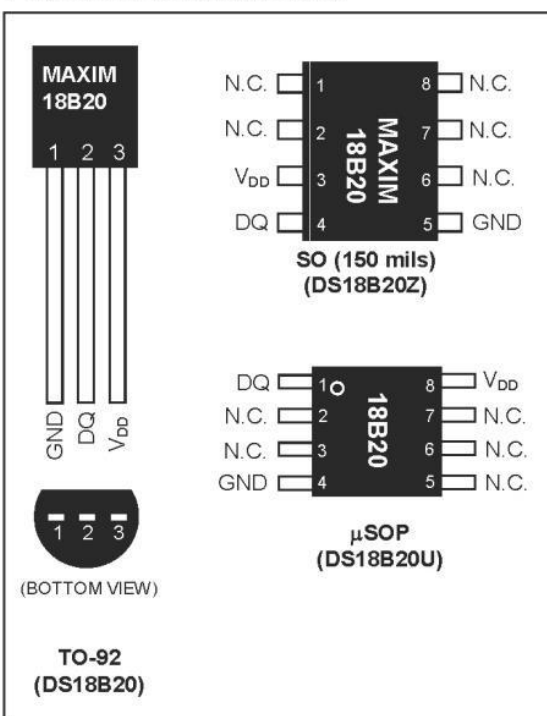
Each DS18B20 has a unique 64-bit serial code, which allows multiple DS18B20s to function on the same 1-Wire bus. Thus, it is simple to use one microprocessor to control many DS18B20s distributed over a large area. Applications that can benefit from this feature include HVAC environmental controls, temperature monitoring systems inside buildings, equipment, or machinery, and process monitoring and control systems.

FEATURES

- Unique 1-Wire® Interface Requires Only One Port Pin for Communication
- Each Device has a Unique 64-Bit Serial Code Stored in an On-Board ROM
- Multidrop Capability Simplifies Distributed Temperature-Sensing Applications
- Requires No External Components
- Can Be Powered from Data Line; Power Supply Range is 3.0V to 5.5V
- Measures Temperatures from -55°C to $+125^{\circ}\text{C}$ (-67°F to $+257^{\circ}\text{F}$)
- $\pm 0.5^{\circ}\text{C}$ Accuracy from -10°C to $+85^{\circ}\text{C}$
- Thermometer Resolution is User Selectable from 9 to 12 Bits
- Converts Temperature to 12-Bit Digital Word in 750ms (Max)

- User-Definable Nonvolatile (NV) Alarm Settings
- Alarm Search Command Identifies and Addresses Devices Whose Temperature is Outside Programmed Limits (Temperature Alarm Condition)
- Available in 8-Pin SO (150 mils), 8-Pin μSOP , and 3-Pin TO-92 Packages
- Software Compatible with the DS1822
- Applications Include Thermostatic Controls, Industrial Systems, Consumer Products, Thermometers, or Any Thermally Sensitive System

PIN CONFIGURATIONS



1-Wire is a registered trademark of Maxim Integrated Products, Inc.

For pricing, delivery, and ordering information, please contact Maxim Direct at 1-888-629-4642, or visit Maxim's website at www.maximintegrated.com.

REV: 042208

Varistor Products

Radial Lead Varistors > ZA Series

PART NUMBER: 515



ZA Varistor Series



Agency Approvals

Agency	Agency File Number
	E135010, (+ E320116 except parts V8ZAxxP and V12ZAxxP)
	116895
	42201-006

Description

The ZA Series of transient voltage surge suppressors are radial leaded varistors (MOVs) designed for use in the protection of low and medium-voltage circuits and systems. Typical applications include motor control, telecom, automotive systems, solenoid, and power supply circuits to protect circuit board components and maintain data integrity.

These devices are available in five model sizes: 5mm, 7mm, 10mm, 14mm and 20mm, and feature a wide V_{DC} voltage range of 5.5V to 615V.

See ZA Series Device Ratings and Specifications Table for part number and brand information.

Features

- Lead-free, Halogen-Free and RoHS compliant
- Wide operating voltage range V_{MACRMS} 4V to 460V
- DC voltage ratings 5.5V to 615V
- No derating up to 85°C ambient
- 5 model sizes available: 5, 7, 10, 14, and 20mm
- Radial lead package for hard-wired or printed circuit board designs
- Available in tape and reel or bulk pack
- Standard lead form options

Absolute Maximum Ratings

• For ratings of individual members of a series, see Device Ratings and Specifications chart.

Continuous	ZA Series	Units
Steady State Applied Voltage:		
AC Voltage Range (V_{MACRMS})	4 to 460	V
DC Voltage Range (V_{DC})	5.5 to 615	V
Transients:		
Peak Pulse Current (I_{PP})		
For 8/20 μ s Current Wave (See Figure 2)	50 to 6500	A
Single Pulse Energy Range (Note 1)		
For 10/1000 μ s Current Wave (W_{TM})	0.1 to 52	J
Operating Ambient Temperature Range (T_A)	-55 to +85	°C
Storage Temperature Range (T_{STG})	-55 to +125	°C
Temperature Coefficient (%) of Clamping Voltage (V_C) at Specified Test Current	<0.01	%/°C
Hi-Pot Encapsulation (COATING Isolation Voltage Capability) (Dielectric must withstand indicated DC voltage for one minute per MIL-STD-202, Method 301)	2500	V
COATING Insulation Resistance	1000	M Ω

CAUTION: Stresses above those listed in "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress only rating and operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

PART NUMBER: 516



TECH SPECS

RadioShack® 6" Modular IC Breadboard Socket

Model: 276WBU202 | Catalog #: 276-002

Dimensions

Product Length	6.54 inches
Product Height	0.08 inches
Product Width	2.03 inches
Product Weight	4 ounces

General Features

Model	276WBU202
Product Type	Breadboards

Miscellaneous Features

Supported Languages	English
---------------------	---------

Axial Lead & Cartridge Fuses

PICO® II > Very Fast-Acting > 251/253 Series

PART NUMBER: 517



251/253 Series, PICO® II, Very Fast-Acting Fuse



Description

The PICO® II Very Fast-Acting Fuse is designed to meet an extensive array of performance characteristics in a space-saving subminiature package.

Features

- Very fast-acting
- Small size
- Wide current rating range (62mA- 15A)
- Halogen-free available
- Wide operating temperature range
- Low temperature derating

Agency Approvals

Agency	Agency File Number	Ampere Range
	E10480	62mA - 15A
	LR 29862	62mA - 15A
	JET1896-31007-1004	1A - 5A
	J50158379	500mA - 10A
	FM10	62mA - 15A
	2009010207386577 - 500mA to 5A	500mA, 1A, 2A, 2.5A, 3A, 4A, 5A

Applications

Secondary protection for space constrained applications

- Flat-panel display TV
- LCD monitor
- LCD backlight inverter
- Office machines
- Power supply
- Audio/Video system
- Lighting system
- Medical equipment

Electrical Characteristics for Series

% of Ampere Rating	Ampere Rating	Opening Time
100%	62mA - 15A	4 Hours, Min.
200%	62mA - 7A	1 Second, Max.
	10A	3 Seconds, Max.
	12 - 15A	10 Seconds, Max.
275%	500mA, 1A, 2A, 2.5A, 3A, 4A, 5A, 7A, 10A	300 msecs., Max.
400%	500mA, 1A, 2A, 2.5A, 3A, 4A, 5A, 7A, 10A	30 msecs., Max.
1000%	500mA, 1A, 2A, 2.5A, 3A, 4A, 5A, 7A, 10A	4 msecs., Max.

Additional Information



Datasheet
251 Series



Datasheet
253 Series



Resources
251 Series



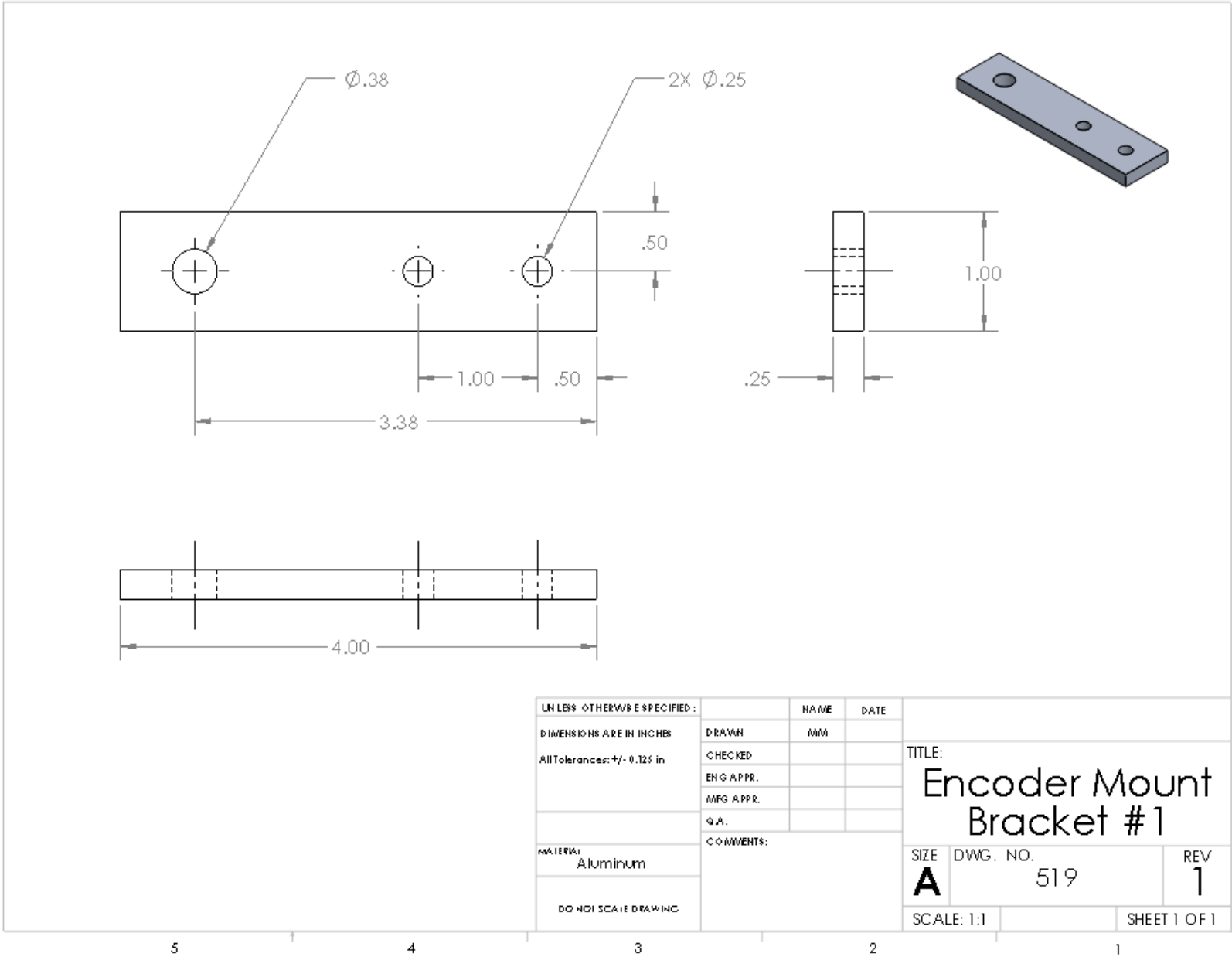
Resources
253 Series

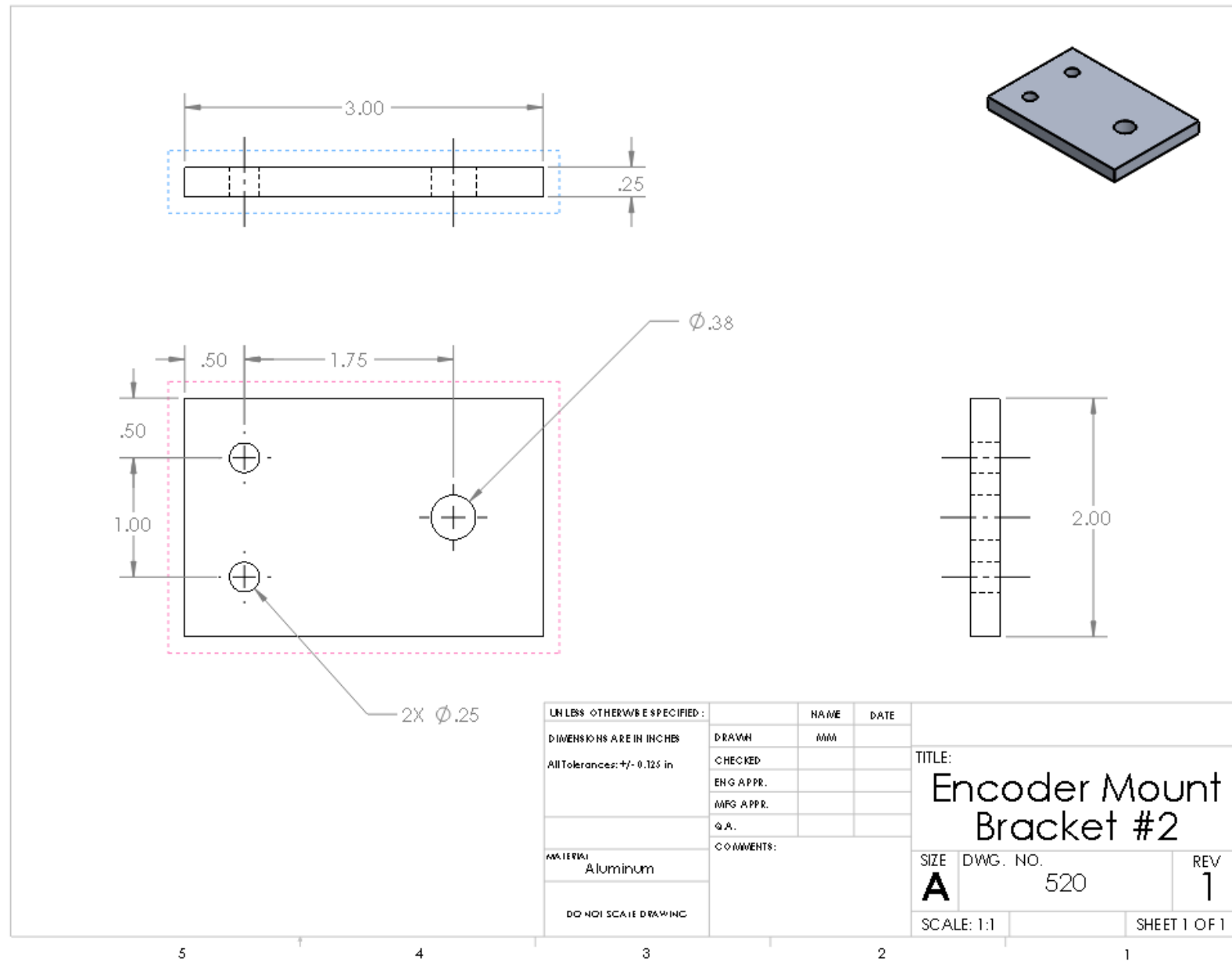


Samples
251 Series



Samples
253 Series





PART NUMBER: 610

Greenfield | Model # B372SPS

2 Gang Weatherproof Electric Outlet Box



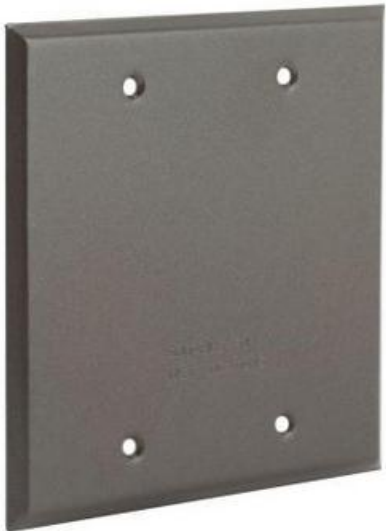
SPECIFICATIONS

Assembled Depth (in.)	2.25 in	Assembled Height (in.)	4.5 in
Assembled Width (in.)	4.5 in	Certifications and Listings	1-UL Listed
Color Family	Gray	Color/Finish	Gray
Electrical Product Type	Weatherproof Box/Cover	Manufacturer Warranty	Limited Warranty
Material	Aluminum	Package Quantity	1
Product Depth (in.)	2.25	Product Height (in.)	4.5
Product Weight (lb.)	0.9392 lb	Product Width (in.)	4.5
Returnable	90-Day	Trade Size (in.)	3/4

PART NUMBER : 611

Bell | Model # 5175-2

2-Gang Blank Weatherproof Cover



SPECIFICATIONS

Assembled Depth (in.)	.125 in	Assembled Height (in.)	4.5 in
Assembled Width (in.)	4.5 in	Certifications and Listings	1-UL Listed
Color Family	Metallic	Color/Finish	Bronze
Electrical Product Type	Weatherproof Box/Cover	Manufacturer Warranty	1 Year
Material	Aluminum	Number of Holes	0.0
Package Quantity	1	Product Depth (in.)	0.125
Product Height (in.)	4.5	Product Weight (lb.)	0.128 lb
Product Width (in.)	4.5	Returnable	90-Day
Trade Size (in.)	Other		

PART NUMBER : 612

Bell | Model # MM2420C

2-Gang Non-Metallic Weatherproof In-Use Cover



SPECIFICATIONS

Assembled Depth (in.)	3.250 in	Assembled Height (in.)	5.75 in
Assembled Width (in.)	5.5 in	Certifications and Listings	1-UL Listed
Color Family	Black	Electrical Product Type	Cover
Manufacturer Warranty	1 Year	Material	Plastic
Number of Gangs	2	Package Quantity	6
Paintable/Stainable	No	Product Depth (in.)	3.1
Product Height (in.)	5.39	Product Weight (lb.)	.6 lb
Product Width (in.)	5.8	Returnable	90-Day
Trade Size (in.)	Other		

PART NUMBER : 613

Gardner Bender | Model # 15-106

12 - 10 AWG, #8 - 10 Stud Size Yellow Vinyl-Insulated Ring Terminals



SPECIFICATIONS

Assembled Depth (in.)	1.25 in	Assembled Height (in.)	4.3 in
Assembled Width (in.)	2 in	Certifications and Listings	1-UL Listed,CSA Listed
Color Family	Yellow	Electrical Product Type	Wire Connector
Manufacturer Warranty	None	Maximum Wire Capacity	12
Minimum Wire Capacity	10	Product Depth (in.)	1.25
Product Height (in.)	4.3	Product Width (in.)	2
Voltage (volts)	600	Wire Type	Bare Copper

PART NUMBER : 614

Gardner Bender | Model # 15-107

12 - 10 AWG, #12 - 1/4 Stud Size Yellow Vinyl-Insulated Ring Terminals



SPECIFICATIONS

Assembled Depth (in.)	1.25 in	Assembled Height (in.)	4.3 in
Assembled Width (in.)	2 in	Certifications and Listings	1-UL Listed,CSA Listed
Color Family	Yellow	Electrical Product Type	Wire Connector
Manufacturer Warranty	None	Maximum Wire Capacity	12
Minimum Wire Capacity	10	Product Depth (in.)	1.25
Product Height (in.)	4.3	Product Width (in.)	2
Voltage (volts)	600	Wire Type	Bare Copper

PART NUMBER : 615



Gardner Bender | Model # 15-108

12 - 10 AWG, 5/16 - 3/8 Stud Size Yellow Vinyl-Insulated Ring Terminals



SPECIFICATIONS

Assembled Depth (in.)	1.25 in	Assembled Height (in.)	4.3 in
Assembled Width (in.)	2 in	Certifications and Listings	1-UL Listed,CSA Listed
Color Family	Yellow	Electrical Product Type	Wire Connector
Manufacturer Warranty	None	Maximum Wire Capacity	12
Minimum Wire Capacity	10	Product Depth (in.)	1.25
Product Height (in.)	4.3	Product Width (in.)	2
Voltage (volts)	600	Wire Type	Bare Copper

PART NUMBER : 616

Model # PPC-1525UVB

1/4 in. Plastic Cable Clamps - Black (18-Pack)



SPECIFICATIONS

Assembled Depth (in.)	1.13 in	Assembled Height (in.)	5.5 in
Assembled Width (in.)	3.75 in	Electrical Product Type	Electrical Tool Accessory
Manufacturer Warranty	None	Product Depth (in.)	1.13
Product Height (in.)	5.5	Product Weight (lb.)	0.0375 lb
Product Width (in.)	3.75		

PART NUMBER : 617



| Model # PPC-1538UVB

3/8 in. Plastic 1-Hole Cable Clamps (15-Pack)



SPECIFICATIONS

Assembled Depth (in.)	1.13 in	Assembled Height (in.)	5.5 in
Assembled Width (in.)	3.75 in	Color Family	Blacks
Electrical Product Type	Electrical Tool Accessory	Fastener Type	Specialty Fastener
Manufacturer Warranty	None	Package Quantity	15
Product Depth (in.)	1.13	Product Height (in.)	5.5
Product Weight (lb.)	0.05 lb	Product Width (in.)	3.75

PART NUMBER : 618

Eaton | Model # GBK10CS

10-Terminal Ground Bar Kit



SPECIFICATIONS

Assembled Depth (in.)	1 in	Assembled Height (in.)	7.188 in
Assembled Width (in.)	2 in	Certifications and Listings	1-UL Listed,ANSI Certified
Electrical Product Type	Grounding Bar	Manufacturer Warranty	1 Year
Product Depth (in.)	.5	Product Height (in.)	6
Product Series	EATON	Product Weight (lb.)	.131 lb
Product Width (in.)	.5	Returnable	90-Day

Brand	Allstar
Item Weight	1.1 ounces
Product Dimensions	7.6 x 15.7 x 0.6 inches
Item model number	ALL76332
Manufacturer Part Number	ALL76332

PART NUMBER : 619



Allstar Performance ALL76332 Copper Ground Strap, 12"

Technical Details

Brand	Allstar
Item Weight	1.1 ounces
Product Dimensions	7.6 x 15.7 x 0.6 inches
Item model number	ALL76332
Manufacturer Part Number	ALL76332

PART NUMBER : 620

Husky | Model # AW62633

8 ft. 16/2 Power Tool Replacement Cord

SPECIFICATIONS

Application	Medium Duty	Assembled Depth (in.)	9.5 in
Assembled Height (in.)	1 in	Assembled Width (in.)	2.5 in
Certifications and Listings	1-UL Listed	Cord Length (ft.)	8
Cord Thickness (In.)	0.03	Electrical Product Type	Electrical Cords & Cord Management
Indoor/Outdoor	Indoor	Manufacturer Warranty	3 years
Maximum Amperage (amps)	13 A	Number of Conductors	2
Number of Outlets	0	Returnable	90-Day
Voltage (volts)	125		

PART NUMBER: 621



Click to open expanded view

NOCO HM318BKS Group 24-31 Snap-Top Battery Box for Automotive, Marine, and RV Batteries

by NOCO

★★★★★ ▾ 27 customer reviews | 8 answered questions

List Price: ~~\$46.79~~Price: **\$14.80** & **FREE Shipping** on orders over \$35. [Details](#)You Save: **\$1.99 (12%)****In Stock.**Ships from and sold by Amazon.com in [easy-to-open packaging](#). Gift-wrap available.**Want it Monday, June 9?** Order within **17 hrs 50 mins** and choose **One-Day Shipping** at checkout. [Details](#)

- Heavy-duty grade battery box for Automotive, Marine, RV and more; Designed for a single group 24-31 battery
- Unique patented design; Features locking tabs to securely fasten the lid to the base, reinforced handles to prevent cracking during relocation, and vent caps to limit water entry
- Designed for rugged environments; Impact resistant down to minus 20 degrees Fahrenheit, withstands acid, gas, oil and other contaminants, and immune to UV exposure
- Keeps your battery safe; Limits water entry, effectively collects battery acid, allows adequate ventilation, and protects against accidental contact of battery terminals
- Ships in Certified Frustration-Free Packaging
- Certified and tested; Meets United States Coast Guard (USCG) Code of Federal Regulations 183.420 and American Boat and Yacht Council (ABYC) E-10.7 specifications

› [See more product details](#)

APPENDIX G: SOFTWARE TASK & STATE DIAGRAMS

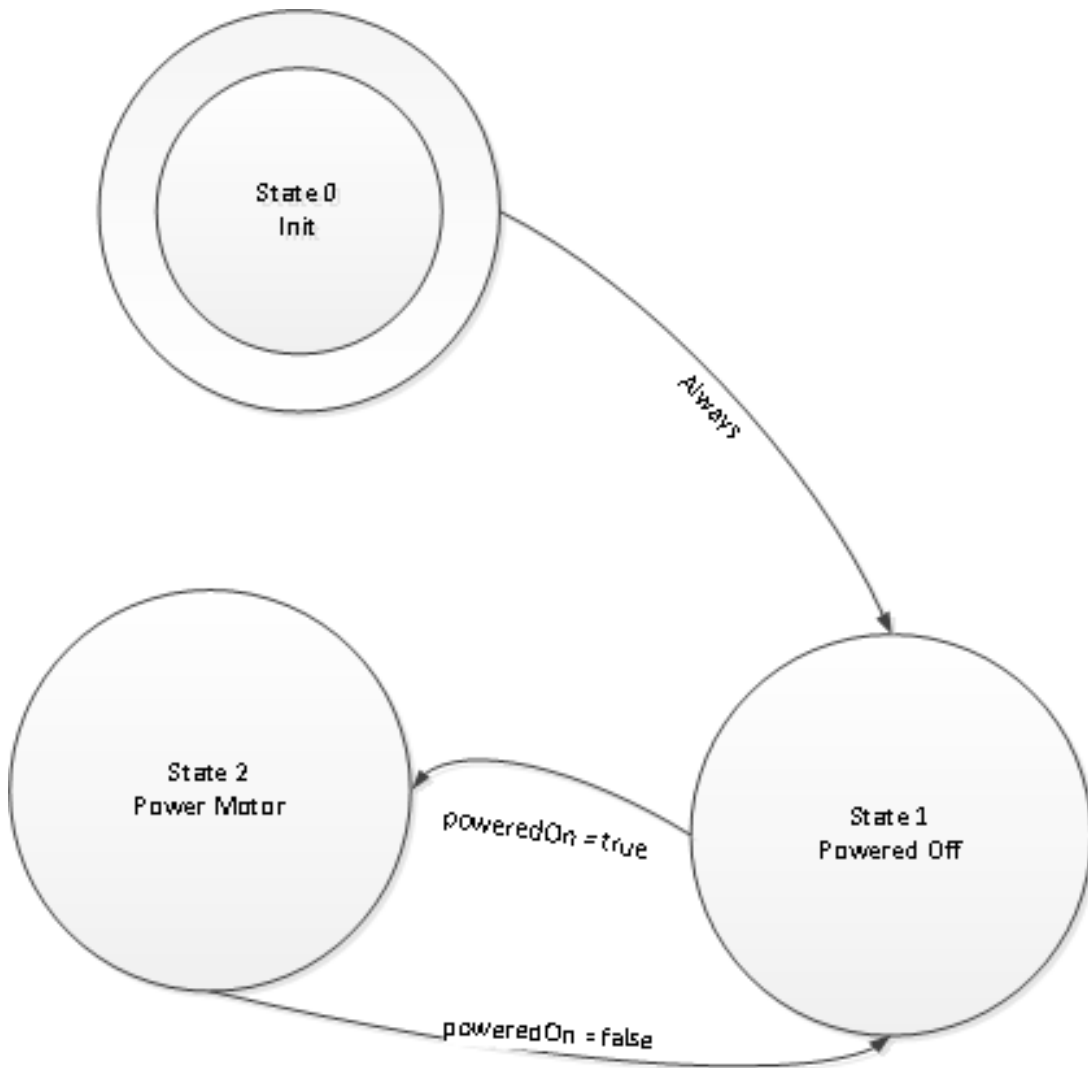


FIGURE 74 - ACTUATOR CONTROLLER STATE DIAGRAM

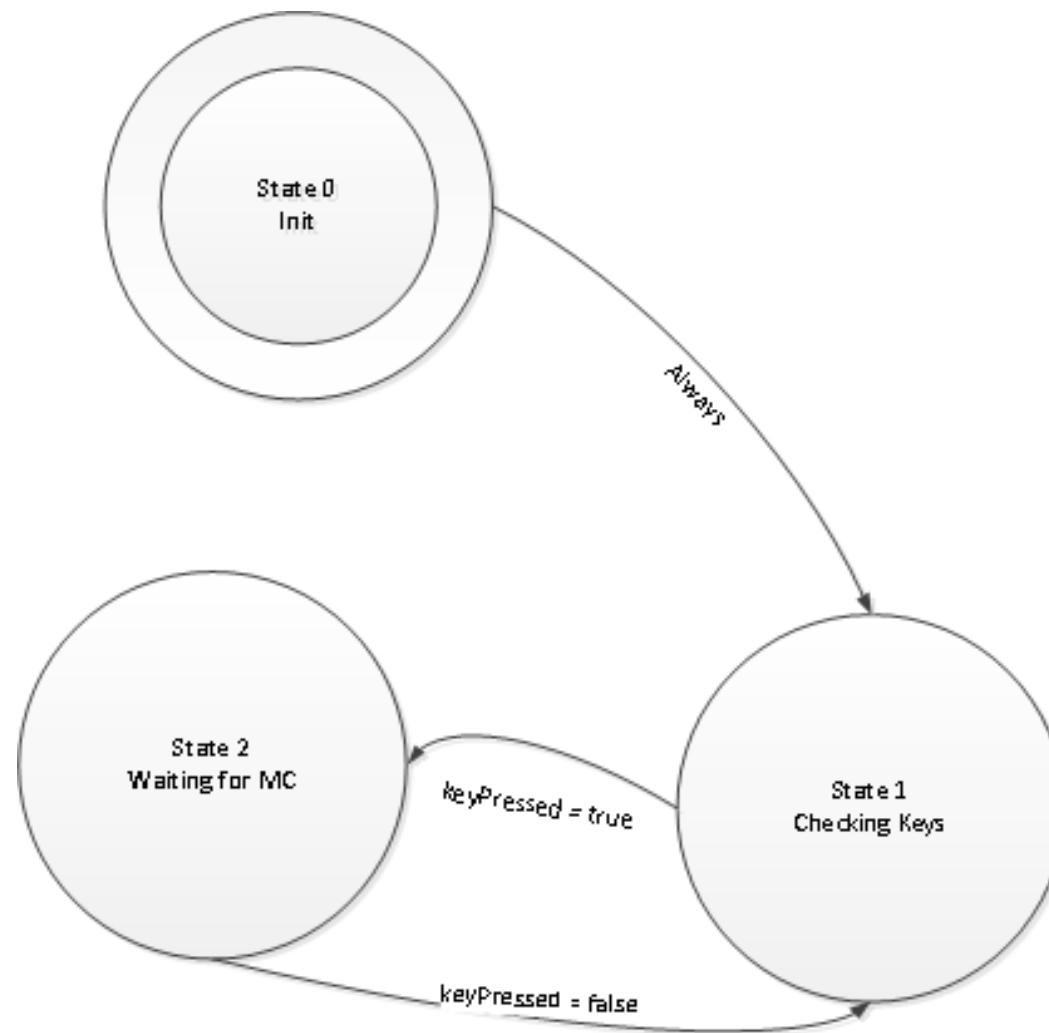


FIGURE 75 - KEY PAD STATE DIAGRAM

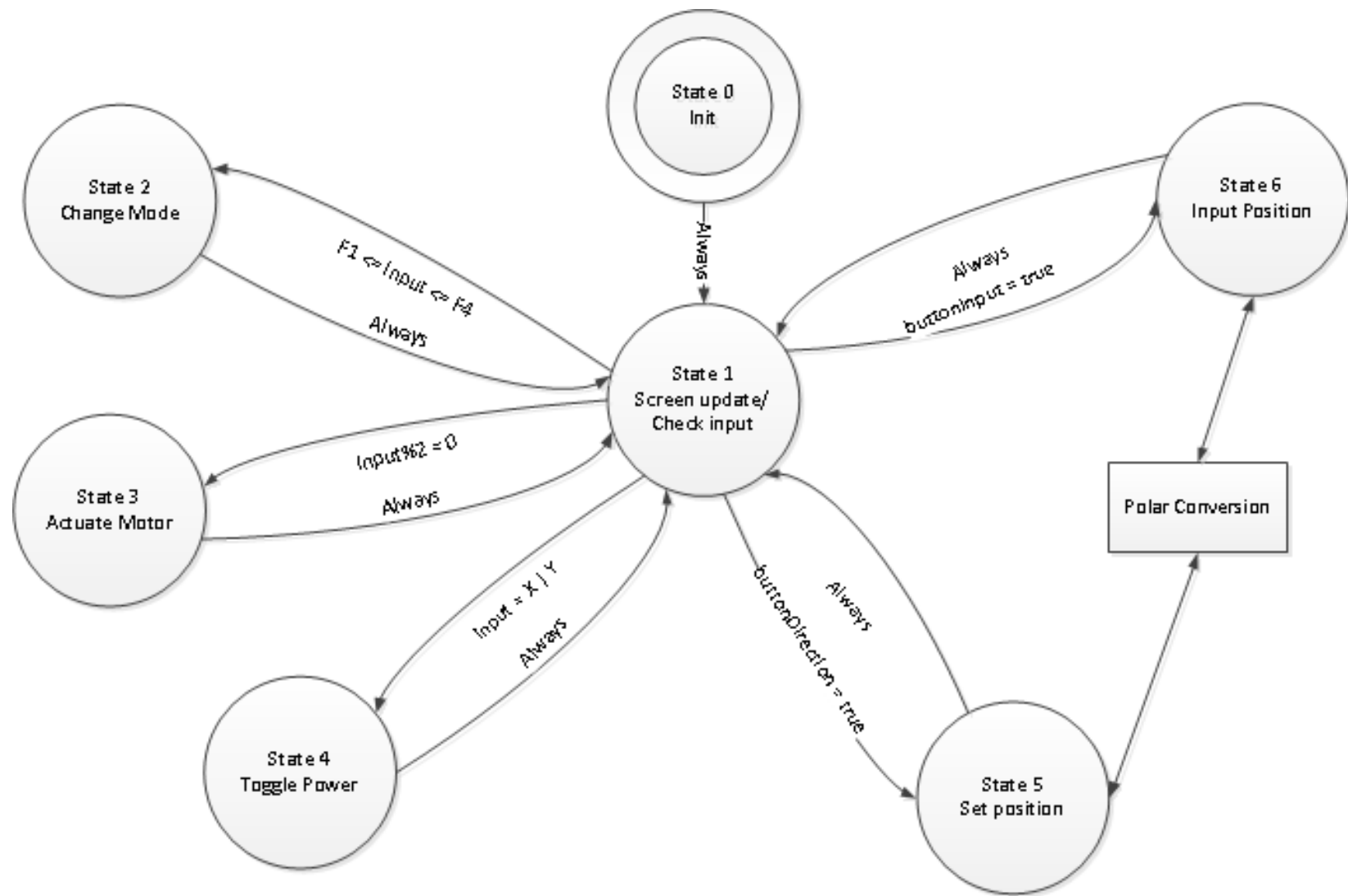


FIGURE 76- UI CONTROLLER STATE DIAGRAM

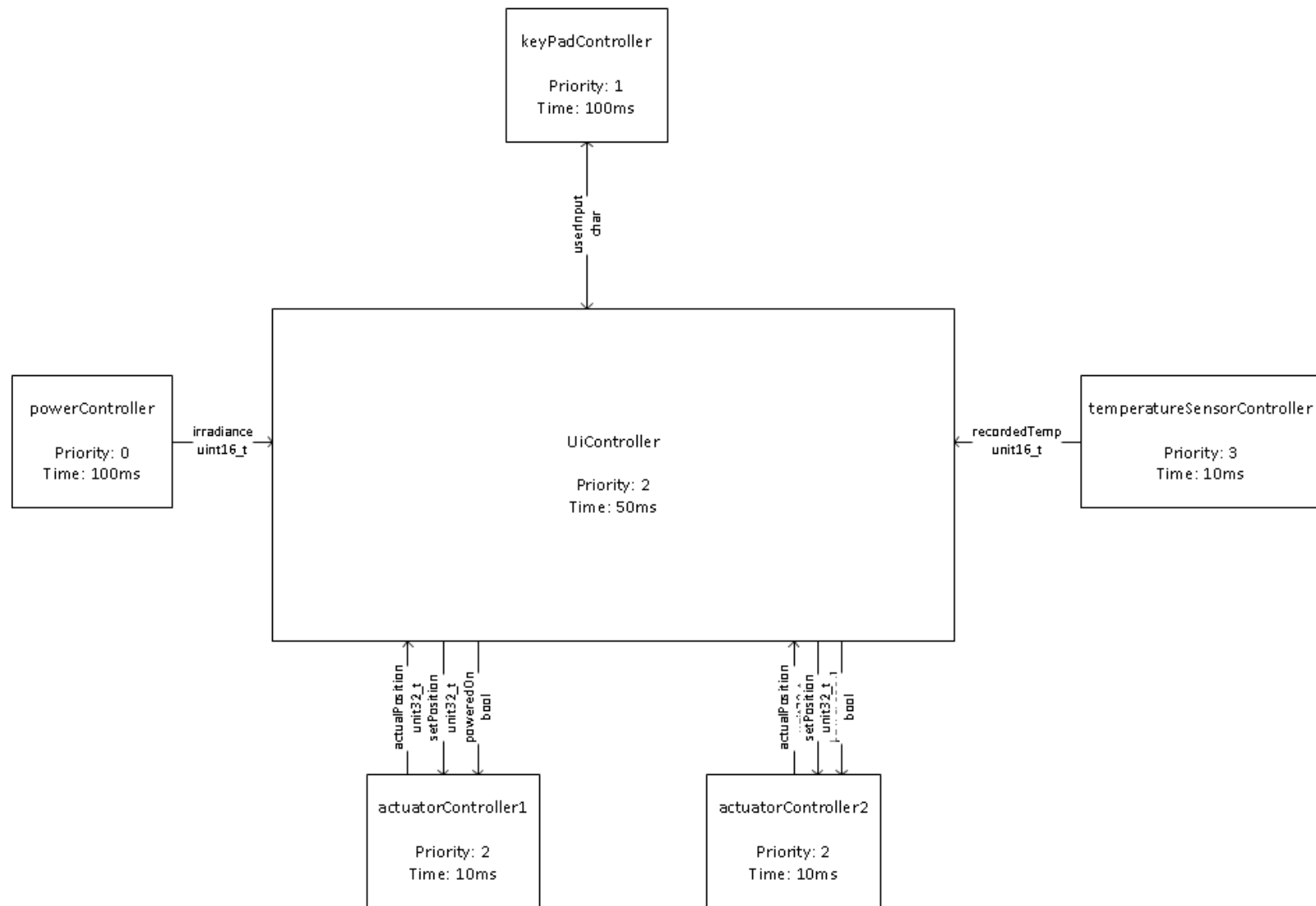


FIGURE 77 - TASK DIAGRAM

APPENDIX H: SOFTWARE DOCUMENTATION

PV SOLAR TRAINER MAIN PROGRAM

TEAM SUNSTREAM

VERSION 1.0

6/5/2014 2:36:00 AM

TABLE OF CONTENTS

Hierarchical Index.....	2
Class Index	3
File Index.....	4
Class Documentation.....	5
AbsoluteEncoderDriver	5
ActuatorController	7
adc.....	10
avr_1wire.....	13
avr_ds182x	20
Usage.....	20
CurrentSensorDriver.....	24
FanDriver	26
IrradianceDriver	28
KeypadController	30
KeypadDriver	32
LcdDriver	34
MotorDriver.....	39
PowerController.....	42
SpiMaster	44
task_user	47
TempController.....	49
UiController	52
VoltageSensorDriver	60
File Documentation	62
AbsoluteEncoderDriver.cpp	62
AbsoluteEncoderDriver.h.....	63
ActuatorController.h	64
adc.cpp	65
adc.h.....	67
avr_1wire.cpp	69
avr_1wire.h.....	70
avr_ds182x.cpp	73
avr_ds182x.h.....	75
ConvertCoord.cpp	77
ConvertCoord.h.....	79
CurrentSensorDriver.cpp	82
CurrentSensorDriver.h.....	83
FanDriver.cpp.....	84
FanDriver.h	85

IrradianceDriver.cpp.....	86
IrradianceDriver.h	87
KeypadController.cpp.....	88
KeypadController.h	89
KeypadDriver.cpp.....	90
KeypadDriver.h	91
LcdDriver.cpp	93
LcdDriver.h	94
MotorDriver.cpp	96
MotorDriver.h	97
PowerController.cpp	99
PowerController.h	100
PvTrainerMain.cpp.....	101
shares.h	105
SpiMaster.cpp	107
SpiMaster.h	109
task_user.h	111
TempController.cpp	112
TempController.h	113
UiController.cpp.....	114
UiController.h	115
VoltageSensorDriver.cpp	116
VoltageSensorDriver.h.....	117
Index.....	118

HIERARCHICAL INDEX

CLASS HIERARCHY

This inheritance list is sorted roughly, but not completely, alphabetically:

AbsoluteEncoderDriver	5
adc	10
avr_1wire	13
avr_ds182x	20
CurrentSensorDriver	24
FanDriver	26
frt_task	
ActuatorController	7
KeypadController	30
PowerController	42
task_user	47
TempController	49
UiController	52
IrradianceDriver	28
KeypadDriver	32
LcdDriver	34
MotorDriver	39
SpiMaster	44
VoltageSensorDriver	60

CLASS INDEX

CLASS LIST

Here are the classes, structs, unions and interfaces with brief descriptions:

AbsoluteEncoderDriver (This driver contains the methods necessary to initialize and control an encoder screen)	5
ActuatorController (This task controls the motor driver to make the actuators work)	7
adc (This class should run the A/D converter on an AVR processor)	10
avr_1wire (This class implements a bit-banged One-Wire Interface (OWI) port)	13
avr_ds182x (This class implements a driver for an AVR processor to a DS182X type "one-wire" temperature sensor)	20
CurrentSensorDriver (This file contains the methods necessary to control a current sensor)	24
FanDriver (This file contains the methods necessary to control a fan relay)	26
IrradianceDriver (This file contains the methods necessary to control an irradiance sensor)	28
KeypadController (This task controls the power sensors and irradiance drivers)	30
KeypadDriver	32
LcdDriver	34
MotorDriver (This class should run a motor driver on an AVR processor)	39
PowerController (This task controls the power sensors and irradiance drivers)	42
SpiMaster (This contains the methods necessary communicate with SPI supported hardware)	44
task_user	47
TempController (This task controls the sensor and fan driver)	49
UiController (This task controls the LCD display)	52
VoltageSensorDriver (This file contains the methods necessary to read panel voltage)	60

FILE INDEX

FILE LIST

Here is a list of all documented files with brief descriptions:

AbsoluteEncoderDriver.cpp	62
AbsoluteEncoderDriver.h	63
ActuatorController.cpp	Error! Bookmark not defined.
ActuatorController.h	64
adc.cpp	65
adc.h	67
avr_1wire.cpp	69
avr_1wire.h	70
avr_ds182x.cpp	73
avr_ds182x.h	75
ConvertCoord.cpp	77
ConvertCoord.h	79
CurrentSensorDriver.cpp	82
CurrentSensorDriver.h	83
FanDriver.cpp	84
FanDriver.h	85
IrradianceDriver.cpp	86
IrradianceDriver.h	87
KeypadController.cpp	88
KeypadController.h	89
KeypadDriver.cpp	90
KeypadDriver.h	91
LcdDriver.cpp	93
LcdDriver.h	94
MotorDriver.cpp	96
MotorDriver.h	97
PowerController.cpp	99
PowerController.h	100
PvTrainerMain.cpp	101
shares.h	105
SpiMaster.cpp	107
SpiMaster.h	109
task_user.cpp	Error! Bookmark not defined.
task_user.h	111
TempController.cpp	112
TempController.h	113
UiController.cpp	114
UiController.h	115
VoltageSensorDriver.cpp	116
VoltageSensorDriver.h	117

CLASS DOCUMENTATION

ABSOLUTEENCODERDRIVER CLASS REFERENCE

This driver contains the methods necessary to initialize and control an encoder screen.

```
#include <AbsoluteEncoderDriver.h>
```

PUBLIC MEMBER FUNCTIONS

- **AbsoluteEncoderDriver** (emstream *ptrSerial, volatile uint8_t *inputPortRegister, volatile uint8_t *inputDdrRegister, uint8_t inputSsPin, uint8_t inputCalibration, **SpiMaster** *inputPtrSpiMaster)
This constructor sets up the encoder driver.
- uint16_t **read** (void)
This method retrieves the position from the encoder.

PROTECTED ATTRIBUTES

- emstream * **p_serial**
Pointer to serial.
- volatile uint8_t * **portRegister**
SPI chip select port.
- uint8_t **ssPin**
SPI chip select pin.
- uint8_t **calibration**
Calibration angle.
- **SpiMaster** * **ptrSpiMaster**
SPI protocol used to receive data from encoder.

DETAILED DESCRIPTION

This driver contains the methods necessary to initialize and control an encoder screen.

The encoder driver uses **SpiMaster.h** to define out it communicates with hardware. The maximum clock rate for communication is 100KHz.

Definition at line 44 of file AbsoluteEncoderDriver.h.

CONSTRUCTOR & DESTRUCTOR DOCUMENTATION

```
ABSOLUTEENCODERDRIVER::ABSOLUTEENCODERDRIVER (EMSTREAM * PTRSERIAL, VOLATILE
    UINT8_T * INPUTPORTREGISTER, VOLATILE UINT8_T * INPUTDDRREGISTER, UINT8_T
    INPUTSSPIN, UINT8_T INPUTCALIBRATION, SPIMASTER * INPUTPTRSPIMASTER)
```

This constructor sets up the encoder driver.

The encoder driver is setup to communicate over an SPI communication protocol.

PARAMETERS:

<i>ptrSerial</i>	A pointer to the serial port which writes debugging info
<i>inputPortRegister</i>	A pointer to the control register for setting pins high or low
<i>inputDdrRegister</i>	A pointer to the register settings pins for input or output
<i>inputSsPin</i>	Pin for controlling SS chip select
<i>inputSpiMaster</i>	Pointer to SPI protocol driver

Definition at line 36 of file AbsoluteEncoderDriver.cpp.

References calibration, p_serial, portRegister, ptrSpiMaster, and ssPin.

MEMBER FUNCTION DOCUMENTATION

UINT16_T ABSOLUTEENCODERDRIVER::READ (VOID)

This method retrieves the position from the encoder.

The latest position is retrieved from the encoder and stored into a shared variable.

Definition at line 65 of file AbsoluteEncoderDriver.cpp.

References calibration, portRegister, ptrSpiMaster, SpiMaster::recieve(), and ssPin.

Referenced by ActuatorController::ActuatorController(), and ActuatorController::run().

THE DOCUMENTATION FOR THIS CLASS WAS GENERATED FROM THE FOLLOWING FILES:

- **AbsoluteEncoderDriver.h**
- **AbsoluteEncoderDriver.cpp**

ACTUATORCONTROLLER CLASS REFERENCE

This task controls the motor driver to make the actuators work.

```
#include <ActuatorController.h>
```

Inheritance diagram for ActuatorController:



PUBLIC MEMBER FUNCTIONS

- **ActuatorController** (const char *, unsigned portBASE_TYPE, size_t, emstream *, **MotorDriver** *, shared_data< int16_t > *, shared_data< int16_t > *, shared_data< bool > *, **AbsoluteEncoderDriver** *)
This constructor creates a generic task of which many copies can be made.
- void **run** (void)
This method is called by the RTOS once to run the task loop for ever and ever.

PROTECTED ATTRIBUTES

- **MotorDriver** * **ptrMotorDriver**
Pointer to motor driver associated with this task.
- **AbsoluteEncoderDriver** * **ptrAbsoluteEncoderDriver**
Pointer to absolute encoder driver.
- int32_t **motorPower**
Power value passed to motor.
- int16_t **setAngle**
Angle to move actuator towards.
- int16_t **angle**
Encoder Angle.
- bool **state**
Dictates running state of motor, ON is true.
- uint16_t **timeChange**
Default to update.
- uint16_t **kp**
Gain for proportional control.
- uint16_t **ki**
Gain for intergral control.
- uint16_t **kd**
Gain for derivative control.
- int32_t **error**
Current error for proportional control.

- `int32_t errorSum`
Accumulated error for integral control.
- `int32_t lastError`
Last recorded error for derivative control.
- `shared_data< int16_t > * ptrSharedAngle`
Pointer to shaft angle.
- `shared_data< int16_t > * ptrSharedEncoder`
Pointer to encoder position.
- `shared_data< bool > * ptrSharedState`
Pointer to motor state.

DETAILED DESCRIPTION

This task controls the motor driver to make the actuators work.

The motor driver is run using files **MotorDriver.h** and **MotorDriver.cpp**. Code in this task sets up a timer/counter in Fast PWM mode and controls the motor's speed and direction.

Definition at line 51 of file ActuatorController.h.

CONSTRUCTOR & DESTRUCTOR DOCUMENTATION

```
ACTUATORCONTROLLER::ACTUATORCONTROLLER (CONST CHAR * A_NAME, UNSIGNED
PORTBASE_TYPE A_PRIORITY, SIZE_T A_STACK_SIZE, EMSTREAM * P_SER_DEV, MOTORDRIVER *
INPUTPTRMOTORDRIVER, SHARED_DATA< INT16_T > * INPUTPTRSHAREDANGLE,
SHARED_DATA< INT16_T > * INPUTPTRSHAREDENCODER, SHARED_DATA< BOOL > *
INPUTPTRSHAREDSTATE, ABSOLUTEENCODERDRIVER * INPUTPTRABSOLUTEENCODERDRIVER)
```

This constructor creates a generic task of which many copies can be made.

This constructor sets up the actuator controller.

The actuator controller constructor takes the appropriate data for its `frt_task` parameters, loads motor driver pointer and number, and sets default state to OFF.

PARAMETERS:

<code>a_name</code>	A character string which will be the name of this task
<code>a_priority</code>	The priority at which this task will initially run (default: 0)
<code>a_stack_size</code>	The size of this task's stack in bytes (default: configMINIMAL_STACK_SIZE)
<code>p_ser_dev</code>	Pointer to a serial device (port, radio, SD card, etc.) which can be used by this task to communicate (default: NULL)

<i>inputPtrMotorDriver</i>	A pointer to associated motor driver to control
<i>inputMotorNumber</i>	A number assigned which dictates which shared vars to use

Definition at line 43 of file ActuatorController.cpp.

References angle, error, errorSum, kd, ki, kp, lastError, motorPower, ptrAbsoluteEncoderDriver, ptrMotorDriver, ptrSharedAngle, ptrSharedEncoder, ptrSharedState, AbsoluteEncoderDriver::read(), setAngle, state, and timeChange.

MEMBER FUNCTION DOCUMENTATION

VOID ACTUATORCONTROLLER::RUN (VOID)

This method is called by the RTOS once to run the task loop for ever and ever.

This method sends the appropriate commands to the motor driver.

If the motor state is set false, this method will send the brake command to the motor. Current architecture scales input percent to function with motor driver class.

Definition at line 82 of file ActuatorController.cpp.

References angle, MotorDriver::brake(), error, errorSum, kd, ki, kp, lastError, motorPower, ptrAbsoluteEncoderDriver, ptrMotorDriver, ptrSharedAngle, ptrSharedEncoder, ptrSharedState, AbsoluteEncoderDriver::read(), MotorDriver::set_power(), setAngle, state, and timeChange.

THE DOCUMENTATION FOR THIS CLASS WAS GENERATED FROM THE FOLLOWING FILES:

- **ActuatorController.h**
- ActuatorController.cpp

ADC CLASS REFERENCE

This class should run the A/D converter on an AVR processor.

```
#include <adc.h>
```

PUBLIC MEMBER FUNCTIONS

- **adc** (emstream *=NULL)
This constructor sets up an A/D converter.
- **uint16_t read_once** (uint8_t)
This method takes one A/D reading from the given channel and returns it.
- **uint16_t read_oversampled** (uint8_t, uint8_t)
This method reads the voltage from a pin several times and returns the average.

PUBLIC ATTRIBUTES

- union {
- uint8_t **byte** [2]
- uint16_t **word**
- } **currentValue**
- float **returnValue**
Last converted voltage scaled appropriately with decimals.

PROTECTED ATTRIBUTES

- emstream * **ptr_to_serial**
The ADC class uses this pointer to the serial port to say hello.
-

DETAILED DESCRIPTION

This class should run the A/D converter on an AVR processor.

This class is setup to initialize the A/D converter on the microcontroller and read the voltage from a requested pin. The << operator is also overloaded for convenience.

Definition at line 50 of file adc.h.

CONSTRUCTOR & DESTRUCTOR DOCUMENTATION

```
ADC::ADC (EMSTREAM * P_SERIAL_PORT = NULL)
```

This constructor sets up an A/D converter.

The A/D is made ready so that when a method such as **read_once()** is called, correct A/D conversions can be performed. The **ADCSRA** and **ADMUX** are set so that the ADC is enabled, the reference pin is selected as **AVCC** with an external capacitor at **AREF**, and sets the clock prescaler to 32.

PARAMETERS:

<i>p_serial_port</i>	A pointer to the serial port which writes debugging info.
----------------------	---

Definition at line 48 of file adc.cpp.

References ptr_to_serial, and returnValue.

MEMBER FUNCTION DOCUMENTATION

UINT16_T ADC::READ_ONCE (UINT8_T CH)

This method takes one A/D reading from the given channel and returns it.

This code selects the pin to be used and returns the measured result as a 16-bit word. The task will run until the **ADSC** pin is set low from the microcontroller, signalling a finished conversion. This method currently multiplies the collected value by **AVCC** (5 Volts), and then divides by 1024.

PARAMETERS:

<i>ch</i>	The A/D channel which is being read must be from 0 to 7
-----------	---

RETURNS:

The result of the A/D conversion

Definition at line 78 of file adc.cpp.

Referenced by VoltageSensorDriver::read(), IrradianceDriver::read(), and read_oversampled().

UINT16_T ADC::READ_OVERSAMPLED (UINT8_T CHANNEL, UINT8_T SAMPLES)

This method reads the voltage from a pin several times and returns the average.

This method calls the **read_once()** function for as many times specified from **samples**. These accumulated values are then averaged and returned as a 16-bit word.

PARAMETERS:

<i>channel</i>	The A/D channel which is being read must be from 0 to 7
<i>samples</i>	The number of times for which the A/D should sample

RETURNS:

The averaged value of several samples

Definition at line 113 of file adc.cpp.

References `read_once()`.

Referenced by `CurrentSensorDriver::read()`.

THE DOCUMENTATION FOR THIS CLASS WAS GENERATED FROM THE FOLLOWING FILES:

- **`adc.h`**
 - **`adc.cpp`**
-

AVR_1WIRE CLASS REFERENCE

This class implements a bit-banged One-Wire Interface (OWI) port.

```
#include <avr_1wire.h>
```

PUBLIC MEMBER FUNCTIONS

- **avr_1wire** (volatile uint8_t &, volatile uint8_t &, volatile uint8_t &, uint8_t, emstream * = NULL)
- bool **reset** (void)
- bool **auto_timing** (void)
- void **search** (void)
- void **read_ID** (void)
- bool **read_bit** (void)
- bool **read_byte** (uint8_t *)
- bool **write_byte** (uint8_t)
- bool **write_byte_rev** (uint8_t)
- void **match_ROM** (uint8_t)
- uint64_t **get_ID** (uint8_t)
- uint8_t **find_by_ID** (uint64_t)
- uint8_t **find_by_type** (uint8_t)
- bool **get_ID_bit** (uint8_t, uint8_t)
- void **set_ID_bit** (uint8_t, uint8_t, bool)
- void **show_devices** (emstream *, uint8_t = AOWI_NUM_IDS)

PROTECTED MEMBER FUNCTIONS

- void **write_0** (void)
- void **write_1** (void)

PROTECTED ATTRIBUTES

- volatile uint8_t & **data_inport**
Pointer to the input port to which the one-wire data bit is connected.
- volatile uint8_t & **data_outport**
Pointer to the output port to which the one-wire data bit is connected.
- volatile uint8_t & **data_ddr**
Pointer to the data direction register for the data port.
- uint8_t **data_mask**
This is a bitmask for the pin used for the 1-wire data line.
- uint8_t **errors**
This is a counter for 1-wire bus errors; it's used for debugging.
- AOWI_device_ID **identifiers** [AOWI_NUM_IDS]
This array holds a set of 64-bit device identification numbers.
- uint16_t **reset_pulse_dur**
- emstream * **p_serial**
This pointer allows the use of a serial device for debugging messages.

DETAILED DESCRIPTION

This class implements a bit-banged One-Wire Interface (OWI) port.

This port uses one generic I/O port pin, manipulating it directly, thus the name "bit-banged." One uses this class to set up a one-wire port, then creates one or more driver objects which use this port to communicate with one-wire devices such as DS1820 or DS1822 temperature sensors. See the documentation for specific driver classes for examples of how to write code using this one-wire driver class.

Definition at line 135 of file `avr_1wire.h`.

CONSTRUCTOR & DESTRUCTOR DOCUMENTATION

```
AVR_1WIRE::AVR_1WIRE (VOLATILE UINT8_T & DATA_IN_PORT, VOLATILE UINT8_T &
DATA_OUT_PORT, VOLATILE UINT8_T & DATA_DIR_REG, UINT8_T DATA_BIT, EMSTREAM *
P_SER_DEV = NULL)
```

This constructor creates a bit-banged one-wire port object. The I/O input and output ports as well as the bitmask for the one-wire pin must be given.

PARAMETERS:

<i>data_in_port</i>	The input port of the data pin, such as PIND
<i>data_out_port</i>	The output port of the data pin, such as PORTD
<i>data_dir_reg</i>	The data direction register for the I/O port, such as DDRD
<i>data_bit</i>	The bit number for the data pin, such as 4 for pin 4
<i>p_ser_dev</i>	A pointer to a serial device for debugging messages (default: NULL, which means no messages will be sent out)

Definition at line 63 of file `avr_1wire.cpp`.

References `AOWI_90us_D`, `AOWI_DELAY`, `AOWI_RESET_D`, `data_ddr`, `data_mask`, `data_outport`, `errors`, `p_serial`, and `reset_pulse_dur`.

MEMBER FUNCTION DOCUMENTATION

```
BOOL AVR_1WIRE::AUTO_TIMING (VOID )
```

This method tries to accomplish a successful reset of the 1-wire device by varying the timing of 1-wire signals.

RETURNS:

True if a timing was found which works, false if not

Definition at line 568 of file avr_1wire.cpp.

References AOWI_DELAY, AOWI_PRESET_D, AOWI_PRESET_END, AOWI_RESET_D, data_ddr, data_inport, data_mask, data_outport, p_serial, and reset_pulse_dur.

UNSIGNED CHAR AVR_1WIRE::FIND_BY_ID (UINT64_T AN_ID)

This method finds the index in the device identifier table of the device whose identifier number matches the given number. If no such device is in the table, 0xFF is returned.

PARAMETERS:

<i>an_ID</i>	The identifier which is to be matched within the table
--------------	--

RETURNS:

The index of the device ID which we want, or 0xFF if it's not there

Definition at line 358 of file avr_1wire.cpp.

References AOWI_NUM_IDS, and identifiers.

Referenced by avr_ds182x::find_by_ID().

UINT8_T AVR_1WIRE::FIND_BY_TYPE (UINT8_T TYPE_ID)

This method finds the index in the device identifier table of the device whose device type number matches the given number. The device type number is the lowest byte in the 64-bit identifier for the device. If no such device is in the table, 0xFF is returned. If two or more devices with a given type are on the same bus, the ID of just one of the devices is returned, and the other is ignored.

PARAMETERS:

<i>type_ID</i>	The device identifier code to be matched within the table
----------------	---

RETURNS:

The index of the device ID which we want, or 0xFF if it's not there

Definition at line 386 of file avr_1wire.cpp.

References AOWI_NUM_IDS, and identifiers.

Referenced by avr_ds182x::find_by_type().

UINT64_T AVR_1WIRE::GET_ID (UINT8_T WHICH_ONE)

This method returns one unique 64-bit device identifier from the table of identifiers kept by this object. If the table index is out of bounds, zero is returned instead.

PARAMETERS:

<i>which_one</i>	The index number in the table
------------------	-------------------------------

RETURNS:

The unique 64-bit identifier of the given device

Definition at line 339 of file avr_1wire.cpp.

References AOWI_NUM_IDS, and identifiers.

Referenced by main(), and show_devices().

BOOL AVR_1WIRE::GET_ID_BIT (UINT8_T WHICH_ID, UINT8_T WHICH_BIT)

This method returns the bit in a One-Wire identifier at a given bit position from 0 to 63. Indices are not checked, but this method is not for general use anyway.

PARAMETERS:

<i>which_ID</i>	The index of the identifier in the table of identifiers
<i>which_bit</i>	The position in the number, from 0 to 63

RETURNS:

The bit which was found, true (1) or false (0)

Definition at line 412 of file avr_1wire.cpp.

References identifiers.

Referenced by match_ROM().

VOID AVR_1WIRE::MATCH_ROM (UINT8_T INDEX)

This method issues a Match ROM command, seeking a match with the device whose identifier is at the given location in the identifier table. This method should be called right after **reset()**.

PARAMETERS:

<i>index</i>	The number of the device's identifier in the table
--------------	--

Definition at line 262 of file avr_1wire.cpp.

References get_ID_bit(), write_0(), write_1(), and write_byte().

Referenced by avr_ds182x::configure(), and avr_ds182x::temperature().

BOOL AVR_1WIRE::READ_BIT (VOID)

This method reads a bit of data from a device on the one-wire bus. Other methods have caused the device to initiate a data transmission. The bit is read by sending a short pulse of about a microsecond, then floating the bus high and sampling it after about 15 microseconds. NOTE: The read delay was changed from AOWI_45us_D to AOWI_15us_D, 15-Dec-2011, and an extra 45 microsecond delay was added to keep subsequent actions in synch

RETURNS:

True if the received bit was a 1, false if it was a 0

Definition at line 183 of file avr_1wire.cpp.

References AOWI_15us_D, AOWI_1us_D, AOWI_45us_D, AOWI_DELAY, data_ddr, data_inport, data_mask, and data_outport.

Referenced by search(), and avr_ds182x::temperature().

BOOL AVR_1WIRE::READ_BYTE (UINT8_T * CH_IN)

This method reads one byte from a device on the One-Wire bus and stores the result in the given character. If something goes wrong, it returns true; if no problem is detected, it returns false.

PARAMETERS:

<i>ch_in</i>	Pointer to the character where the data is stored
--------------	---

RETURNS:

True if there is a problem reading the data, false otherwise. There is currently no test for correct data, so this method always returns false

Definition at line 290 of file avr_1wire.cpp.

References AOWI_15us_D, AOWI_1us_D, AOWI_45us_D, AOWI_DELAY, data_ddr, data_inport, data_mask, and data_outport.

Referenced by read_ID(), and avr_ds182x::temperature().

VOID AVR_1WIRE::READ_ID (VOID)

This method reads the 64-bit identifier code from a single device on the bus and stores that code in the data field 'identifier.' It only works if there is only one device on the bus; if there are more, we must go through the whole search ROM procedure to identify all the devices on the bus.

Definition at line 321 of file avr_1wire.cpp.

References identifiers, read_byte(), and write_byte().

BOOL AVR_1WIRE::RESET (VOID)

This method sends a reset sequence to the one-wire interface. This is accomplished by sending a long (>480 us) pulse, then making sure a presence pulse was sent back by at least one device on the bus.

RETURNS:

True if a presence pulse was detected, false if not

Definition at line 88 of file avr_1wire.cpp.

References AOWI_90us_D, AOWI_DELAY, AOWI_PRES_D, AOWI_PRES_END, AOWI_RESET_D, data_ddr, data_inport, data_mask, data_outport, and p_serial.

Referenced by avr_ds182x::configure(), search(), and avr_ds182x::temperature().

VOID AVR_1WIRE::SEARCH (VOID)

This method searches the One-Wire bus for all connected devices. It issues the "Search ROM" command, then makes guesses about the bits in the devices's ID numbers, one bit at a time; the devices each respond with "right" or "wrong" indications which are wired-OR'ed with each other, allowing the master to know if its guesses were correct or not. The whole process is a complicated mess described in <http://www.maxim-ic.com/products/ibutton/ibuttons/standard.pdf>.

Definition at line 458 of file avr_1wire.cpp.

References AOWI_NUM_IDS, data_mask, identifiers, p_serial, read_bit(), reset(), set_ID_bit(), write_0(), write_1(), and write_byte().

Referenced by main().

VOID AVR_1WIRE::SET_ID_BIT (UINT8_T WHICH_ID, UINT8_T WHICH_BIT, BOOL NEW_BIT)

This method sets the bit in a One-Wire identifier at a given bit position from 0 to 63 to a given value. Indices are not checked, but this method is not for general use anyway.

PARAMETERS:

<i>which_ID</i>	The index of the identifier in the table of identifiers
<i>which_bit</i>	The position in the number, from 0 to 63
<i>new_bit</i>	The new value for the bit, true (1) or false (0)

Definition at line 434 of file avr_1wire.cpp.

References identifiers.

Referenced by search().

```
VOID AVR_1WIRE::SHOW_DEVICES (EMSTREAM * DEBUG_PORT, UINT8_T HOW_MANY =
                              AOWI_NUM_IDS)
```

This method displays a list of devices which have been found on the 1-wire bus. They must have previously been found by **search()**.

PARAMETERS:

<i>debug_port</i>	A pointer to a serial object on which to display the results
<i>how_many</i>	The number of devices to show. If there aren't as many devices as asked for, a bunch of zeros will be displayed for the absent devices. (Default: AOWI_NUM_IDS, the number of elements in the device ID array)

Definition at line 539 of file avr_1wire.cpp.

References AOWI_NUM_IDS, and get_ID().

Referenced by main().

```
VOID AVR_1WIRE::WRITE_0 (VOID ) [ PROTECTED ]
```

This method writes a logic 0 to the one-wire bus. It does so by pulling the data line low for about 90 microseconds, then letting it go high for about 1 us.

Definition at line 129 of file avr_1wire.cpp.

References AOWI_1us_D, AOWI_90us_D, AOWI_DELAY, data_ddr, data_mask, and data_outport.

Referenced by match_ROM(), search(), write_byte(), and write_byte_rev().

```
VOID AVR_1WIRE::WRITE_1 (VOID ) [ PROTECTED ]
```

This method writes a logic 1 to the one-wire bus. It does so by pulling the data line low for about a microsecond, then letting it go high for about 90 us.

Definition at line 153 of file avr_1wire.cpp.

References AOWI_1us_D, AOWI_90us_D, AOWI_DELAY, data_ddr, data_mask, and data_outport.

Referenced by match_ROM(), search(), write_byte(), and write_byte_rev().

```
BOOL AVR_1WIRE::WRITE_BYTE (UINT8_T THE_BYTE)
```

This method writes a byte to the one-wire bus, least significant bit first. It writes each of the 8 bits in turn.

PARAMETERS:

<i>the_byte</i>	The byte (usually a command) which is to be written to the bus
-----------------	--

RETURNS:

True always because there's no timeout

Definition at line 214 of file `avr_1wire.cpp`.

References `write_0()`, and `write_1()`.

Referenced by `avr_ds182x::configure()`, `match_ROM()`, `read_ID()`, `search()`, and `avr_ds182x::temperature()`.

`BOOL AVR_1WIRE::WRITE_BYTE_REV (UINT8_T THE_BYTE)`

This method writes a byte to the one-wire bus, most significant bit first. It just writes each of the 8 bits in the given byte in turn.

PARAMETERS:

<i>the_byte</i>	The byte (usually a command) which is to be written to the bus
-----------------	--

RETURNS:

True if the acknowledgement bit occurred, false if we timed out instead

Definition at line 238 of file `avr_1wire.cpp`.

References `write_0()`, and `write_1()`.

MEMBER DATA DOCUMENTATION

`UINT16_T AVR_1WIRE::RESET_PULSE_DUR [PROTECTED]`

This is a counter to control the duration of the reset pulse; it can be tuned by the **`auto_timing()`** method to compensate for CPU clock speed and variations in 1-wire devices' timing.

Definition at line 159 of file `avr_1wire.h`.

Referenced by `auto_timing()`, and `avr_1wire()`.

THE DOCUMENTATION FOR THIS CLASS WAS GENERATED FROM THE FOLLOWING FILES:

- **`avr_1wire.h`**
- **`avr_1wire.cpp`**

AVR_DS182X CLASS REFERENCE

This class implements a driver for an AVR processor to a DS182X type "one-wire" temperature sensor.

```
#include <avr_ds182x.h>
```

PUBLIC MEMBER FUNCTIONS

- **avr_ds182x** (**avr_1wire** *, uint8_t)
- bool **configure** (uint8_t, uint8_t=0x70, uint8_t=0xE0)
- void **find_by_ID** (uint64_t)
- void **find_by_type** (void)
- int16_t **temperature** (void)
- int16_t **fahrenheit** (void)
- int16_t **celsius** (void)

PROTECTED ATTRIBUTES

- **avr_1wire * the_bus**
Pointer to the 1-wire bus object.
- uint8_t **ID_index**
Index of the sensor in bus's array.
- uint8_t **type_ID**
The sensor's type ID byte.

DETAILED DESCRIPTION

This class implements a driver for an AVR processor to a DS182X type "one-wire" temperature sensor.

The sensor should actually be connected with three wires (power, ground, and data). This driver is not intended for two-wire connection (power/data and ground), which is the way one-wire devices are connected using the minimum number of wires. There is no way to actually connect a one-wire device with just one wire.

USAGE

One or more one-wire devices are connected with their data pins all attached to one digital I/O pin on the microcontroller. It is usually necessary to attach an external pullup resistor of about 4.7K to the pin; the internal pullup resistors in an AVR are generally not strong enough to work properly as 1-wire pullups.

This class uses the **avr_1wire** class to operate the one-wire interface. To use this driver, one must first set up a one-wire interface. The **avr_1wire** constructor takes as parameters the following:

- The parallel input port to which the one wire that goes to the DS182X's data pin is connected
- The parallel output port to which that same wire is connected
- The data direction register which controls the I/O pin for that wire
- The number (from 0 to 7) of the pin

- (Optional) A serial device which will be used to show debugging messages

```
my_1wire = new avr_1wire (PIND, PORTD, DDRD, 7, p_serial);
my_1wire->search ();
my_1wire->show_devices (p_serial);
```

Calling the `search()` method causes the one-wire driver to scan the wire and make a list of all the one-wire devices attached to it. The **avr_1wire** driver must be configured to hold at least as many device ID's as there are devices on the wire; see the define `AOWI_NUM_IDS` in file **avr_1wire.h**. To display the list of the ID's of devices found, call `show_devices()`. After the one-wire driver has been created, the **avr_ds182x** driver is created:

```
my_ds182x = new avr_ds182x (my_1wire);
my_ds182x->find_by_type (0x22);
```

The `0x22` is the type ID of a DS1822; the type ID of a DS1820 is `0x10`. One can only use **find_by_type()** when there is only a single device of a given type attached to the wire; this is usually the case. After the device has been set up, one can read the temperature in desired units with a call to **celsius()** or **fahrenheit()**, which return temperature as signed 16-bit integers in units of tenths of degrees, such as 234 for 23.4 degrees. When using a serial device such as a USB serial port, an SD card, or even a serial RTOS queue, one can also use the overloaded `<<` operator to display the measured temperature:

```
*p_serial << "The temperature is: " << *my_ds182x << endl;
```

Definition at line 110 of file `avr_ds182x.h`.

CONSTRUCTOR & DESTRUCTOR DOCUMENTATION

AVR_DS182X::AVR_DS182X (AVR_1WIRE * A_BUS, UINT8_T TYPE_ID_BYTE)

This constructor creates a DS1822 interface object. A One-Wire interface to which the sensor is attached must already have been created.

PARAMETERS:

<i>a_bus</i>	A pointer to a one-wire bus object which connects to this sensor
<i>type_ID_byte</i>	The type identifier for a given model of sensor, for example <code>0x10</code> for a DS1820 or <code>0x22</code> for a DS1822

Definition at line 48 of file `avr_ds182x.cpp`.

References `ID_index`, `the_bus`, and `type_ID`.

MEMBER FUNCTION DOCUMENTATION

INT16_T AVR_DS182X::CELSIUS (VOID)

This method reads the temperature sensor, then converts the results into an integer which contains the temperature in degrees Celsius times ten.

RETURNS:

The temperature in tenths of a degree Celsius

Definition at line 228 of file avr_ds182x.cpp.

References temperature(), and type_ID.

Referenced by operator<<().

```
BOOL AVR_DS182X::CONFIGURE (UINT8_T RESOLUTION, UINT8_T HIGH_ALARM = 0x70,
                             UINT8_T LOW_ALARM = 0xE0)
```

This method writes a byte to the configuration register of the DS182X. While doing so, it also writes to the alarm configuration registers, which also include the alarm registers. Writing 0x70 to the upper temperature alarm and 0xE0 to the lower temperature alarm should keep them from going off too often, although since one doesn't usually check the alarms, it doesn't matter very much. The resolution byte can be 0x1F for 9 bits, 0x3F for 10 bits, 0x5F for 11 bits and 0x7F for 12 bits resolution. Higher resolution makes the DS182X work more slowly - about 3/4 of a second at 12 bits. The data sent here is stored in the DS182X's EEPROM, so this command doesn't need to be run every time a sensor is powered up.

PARAMETERS:

<i>resolution</i>	A byte which sets the A/D resolution for the sensor
<i>high_alarm</i>	Value for the high alarm register (default: 0x70)
<i>low_alarm</i>	Value for the low alarm register (default: 0xE0)

Definition at line 73 of file avr_ds182x.cpp.

References ID_index, avr_1wire::match_ROM(), avr_1wire::reset(), the_bus, and avr_1wire::write_byte().

Referenced by main().

```
INT16_T AVR_DS182X::FAHRENHEIT (VOID )
```

This method gets a temperature reading in Celsius, then converts it to an old old fashioned Fahrenheit reading.

RETURNS:

The temperature in tenths of a degree Fahrenheit

Definition at line 255 of file avr_ds182x.cpp.

References temperature(), and type_ID.

```
VOID AVR_DS182X::FIND_BY_ID (UINT64_T ID_TO_MATCH)
```

This method finds the index of the given sensor in the One-Wire bus's device ID table by looking for a matching 64-bit device identifier. The result is saved in ID_index. If no match is found, the ID is set to 0xFF.

PARAMETERS:

<i>ID_to_match</i>	The 64-bit ID number for the device we seek
--------------------	---

Definition at line 111 of file avr_ds182x.cpp.

References avr_1wire::find_by_ID(), ID_index, and the_bus.

Referenced by main().

VOID AVR_DS182X::FIND_BY_TYPE (VOID)

This method finds the index of the given sensor in the One-Wire bus's device ID table by looking for a device whose device type matches that of this sensor. If there are more than one such sensors on the bus, this method just finds the first one and returns it. The result is saved in ID_index. If no match is found, the ID is set to 0xFF.

RETURNS:

The index into the device table of this device's 64-bit ID number

Definition at line 126 of file avr_ds182x.cpp.

References avr_1wire::find_by_type(), ID_index, the_bus, and type_ID.

INT16_T AVR_DS182X::TEMPERATURE (VOID)

This method initiates a temperature conversion and reads the converted temperature as a raw 16-bit binary number from the "scratchpad" memory in the DS182X.

RETURNS:

A 16-bit number corresponding to the temperature, or 0 if something's amiss

Definition at line 138 of file avr_ds182x.cpp.

References DS182X_MAX_TOUT, DS182X_RETRIES, ID_index, avr_1wire::match_ROM(), avr_1wire::read_bit(), avr_1wire::read_byte(), avr_1wire::reset(), the_bus, and avr_1wire::write_byte().

Referenced by celsius(), fahrenheit(), and TempController::run().

THE DOCUMENTATION FOR THIS CLASS WAS GENERATED FROM THE FOLLOWING FILES:

- **avr_ds182x.h**
 - **avr_ds182x.cpp**
-

CURRENTSENSORDRIVER CLASS REFERENCE

This file contains the methods necessary to control a current sensor.

```
#include <CurrentSensorDriver.h>
```

PUBLIC MEMBER FUNCTIONS

- **CurrentSensorDriver** (emstream *ptrSerial, shared_data< int16_t > *inputPtrSharedCurrent, **adc** *inputPtrAdc, uint8_t inputVoltagePin)
This constructor sets up the current driver.
- void **read** (void)
This methods reads the output voltage and returns current.

PROTECTED ATTRIBUTES

- emstream * **p_serial**
Pointer to serial.
 - shared_data< int16_t > * **ptrSharedCurrent**
Pointer to shared current value.
 - **adc** * **ptrAdc**
Pointer to ADC.
 - uint8_t **voltagePin**
Pin from which to read voltage.
-

DETAILED DESCRIPTION

This file contains the methods necessary to control a current sensor.

This driver is used to initialize and read from a current sensor.

Definition at line 42 of file CurrentSensorDriver.h.

CONSTRUCTOR & DESTRUCTOR DOCUMENTATION

```
CURRENTSENSORDRIVER::CURRENTSENSORDRIVER (EMSTREAM * PTRSERIAL, SHARED_DATA<
INT16_T > * INPUTPTRSHAREDCURRENT, ADC * INPUTPTRADC, UINT8_T INPUTVOLTAGEPIN)
```

This constructor sets up the current driver.

The current sensor driver is setup to read voltage from the ADC to calculate current.

PARAMETERS:

<i>ptrSerial</i>	A pointer to the serial port which writes debugging info.
<i>inputPtrSharedCurrent</i>	Pointer to shared variable for current.
<i>inputPtrAdc</i>	A pointer to the ADC used to read voltage.
<i>inputVoltagePin</i>	Pin number to read current from.

Definition at line 35 of file CurrentSensorDriver.cpp.

References p_serial, ptrAdc, ptrSharedCurrent, and voltagePin.

MEMBER FUNCTION DOCUMENTATION

VOID CURRENTSENSORDRIVER::READ (VOID)

This methods reads the output voltage and returns current.

The pin voltage is read and is then converted from Volts to Amps.

Definition at line 49 of file CurrentSensorDriver.cpp.

References p_serial, ptrAdc, ptrSharedCurrent, adc::read_oversampled(), and voltagePin.

THE DOCUMENTATION FOR THIS CLASS WAS GENERATED FROM THE FOLLOWING FILES:

- **CurrentSensorDriver.h**
 - **CurrentSensorDriver.cpp**
-

FANDRIVER CLASS REFERENCE

This file contains the methods necessary to control a fan relay.

```
#include <FanDriver.h>
```

PUBLIC MEMBER FUNCTIONS

- **FanDriver** (emstream *ptrSerial, volatile uint8_t *inputPortRegister, volatile uint8_t *inputDdrRegister, uint8_t inputRelayPin)
This constructor sets up the fan driver.
- void **on** (void)
This methods turns the relay on.
- void **off** (void)
This methods turns the relay of.
- void **toggle** (void)
This methods toggles the relay.

PROTECTED ATTRIBUTES

- emstream * **p_serial**
Pointer to serial.
 - volatile uint8_t * **portRegister**
Pin register to change pin status.
 - uint8_t **relayPin**
Pin which controls fan relay.
-

DETAILED DESCRIPTION

This file contains the methods necessary to control a fan relay.

The fan driver is used to initialize the output pin necessary to control the fan relay.

Definition at line 40 of file FanDriver.h.

CONSTRUCTOR & DESTRUCTOR DOCUMENTATION

```
FANDRIVER::FANDRIVER (EMSTREAM * PTRSERIAL, VOLATILE UINT8_T * INPUTPORTREGISTER,  
                      VOLATILE UINT8_T * INPUTDDRREGISTER, UINT8_T INPUTRELAYPIN)
```

This constructor sets up the fan driver.

The fan driver is setup a pin as an output to control a mosfet controlling fans.

PARAMETERS:

<i>ptrSerial</i>	A pointer to the serial port which writes debugging info.
<i>inputPortRegister</i>	A pointer to the control register for setting pins high or low.
<i>inputDdrRegister</i>	A pointer to the register settings pins for input or output.
<i>inputRelayPin</i>	Pin number to control fan relay.

Definition at line 34 of file FanDriver.cpp.

References p_serial, portRegister, and relayPin.

MEMBER FUNCTION DOCUMENTATION

VOID FANDRIVER::OFF (VOID)

This methods turns the relay of.

The pin is set accordingly in the port register and sets the pin connecting to the mosfet low.

Definition at line 64 of file FanDriver.cpp.

References portRegister, and relayPin.

VOID FANDRIVER::ON (VOID)

This methods turns the relay on.

The pin is set accordingly in the port register and sets the pin connecting to the mosfet high.

Definition at line 54 of file FanDriver.cpp.

References portRegister, and relayPin.

VOID FANDRIVER::TOGGLE (VOID)

This methods toggles the relay.

The pin is set accordingly in the port register and sets the pin connecting to the mosfet opposite to its previous value.

Definition at line 74 of file FanDriver.cpp.

References portRegister, and relayPin.

THE DOCUMENTATION FOR THIS CLASS WAS GENERATED FROM THE FOLLOWING FILES:

- **FanDriver.h**
- **FanDriver.cpp**

IRRADIANCEDRIVER CLASS REFERENCE

This file contains the methods necessary to control an irradiance sensor.

```
#include <IrradianceDriver.h>
```

PUBLIC MEMBER FUNCTIONS

- **IrradianceDriver** (emstream *ptrSerial, shared_data< int16_t > *inputPtrSharedIrradiance, **adc** *inputPtrAdc, uint8_t inputVoltagePin)
This constructor sets up the irradiance driver.
- void **read** (void)
This methods reads the output voltage and returns irradiance.

PROTECTED ATTRIBUTES

- emstream * **p_serial**
Pointer to serial.
- shared_data< int16_t > * **ptrSharedIrradiance**
Pointer to shared irradiance value.
- **adc** * **ptrAdc**
Pointer to ADC.
- uint8_t **voltagePin**
Pin from which to read voltage.

DETAILED DESCRIPTION

This file contains the methods necessary to control an irradiance sensor.

The irradiance driver is used to initialize and read from an irradiance sensor.

Definition at line 42 of file IrradianceDriver.h.

CONSTRUCTOR & DESTRUCTOR DOCUMENTATION

```
IRRADIANCEDRIVER::IRRADIANCEDRIVER (EMSTREAM * PTRSERIAL, SHARED_DATA< INT16_T > *  
    INPUTPTRSHAREDIRRADIANCE, ADC * INPUTPTRADC, UINT8_T INPUTVOLTAGEPIN)
```

This constructor sets up the irradiance driver.

The irradiance driver is setup to read voltage from the ADC to calculate irradiance.

PARAMETERS:

<i>ptrSerial</i>	A pointer to the serial port which writes debugging info.
<i>inputPtrSharedIrradiance</i>	A pointer to the shared variable for irradiance.
<i>inputPtrAdc</i>	A pointer to the ADC used to read irradiance.
<i>inputVoltagePin</i>	Pin number to read irradiance from.

Definition at line 35 of file IrradianceDriver.cpp.

References p_serial, ptrAdc, ptrSharedIrradiance, and voltagePin.

MEMBER FUNCTION DOCUMENTATION

VOID IRRADIANCEDRIVER::READ (VOID)

This methods reads the output voltage and returns irradiance.

The pin voltage is read and is then converted from Volts to irradiance.

Definition at line 49 of file IrradianceDriver.cpp.

References ptrAdc, ptrSharedIrradiance, adc::read_once(), and voltagePin.

Referenced by PowerController::run().

THE DOCUMENTATION FOR THIS CLASS WAS GENERATED FROM THE FOLLOWING FILES:

- **IrradianceDriver.h**
- **IrradianceDriver.cpp**

KEYPADCONTROLLER CLASS REFERENCE

This task controls the power sensors and irradiance drivers.

```
#include <KeypadController.h>
```

Inheritance diagram for KeypadController:



PUBLIC MEMBER FUNCTIONS

- **KeypadController** (const char *, unsigned portBASE_TYPE, size_t, emstream *, **KeypadDriver** *, shared_data< char > *inputPtrSharedInput)
This constructor sets up the keypad controller.
- void **run** (void)
This method updates the status of the input to the associated shared variable.

PROTECTED ATTRIBUTES

- **KeypadDriver** * **ptrKeypadDriver**
Pointer to the keypad driver.
- shared_data< char > * **ptrSharedInput**
Pointer to shared keypad input to program.
- char **lastChar**
Last character input from keypad.

DETAILED DESCRIPTION

This task controls the power sensors and irradiance drivers.

The keypad drivers are run using files **KeypadDriver.h** and **KeypadDriver.cpp** . Code in this task communicates with pvTrainerMasterControl via shared variables.

Definition at line 46 of file KeypadController.h.

CONSTRUCTOR & DESTRUCTOR DOCUMENTATION

```
KEYPADCONTROLLER::KEYPADCONTROLLER (CONST CHAR * A_NAME, UNSIGNED
PORTBASE_TYPE A_PRIORITY, SIZE_T A_STACK_SIZE, EMSTREAM * PTRSERIAL, KEYPADDRIVER *
INPUTPTRKEYPADDRIVER, SHARED_DATA< CHAR > * INPUTPTRSHAREDINPUT)
```

This constructor sets up the keypad controller.

The keypad controller constructor takes the appropriate data for its `frt_task` parameters.

PARAMETERS:

<i>a_name</i>	A character string which will be the name of this task
<i>a_priority</i>	The priority at which this task will initially run (default: 0)
<i>a_stack_size</i>	The size of this task's stack in bytes (default: configMINIMAL_STACK_SIZE)
<i>ptrSerial</i>	Pointer to a serial device (port, radio, SD card, etc.) which can be used by this task to communicate (default: NULL)
<i>inputPtrKeypadDriver</i>	A pointer to a keypad driver to be controlled

Definition at line 39 of file KeypadController.cpp.

References `lastChar`, `ptrKeypadDriver`, and `ptrSharedInput`.

THE DOCUMENTATION FOR THIS CLASS WAS GENERATED FROM THE FOLLOWING FILES:

- **KeypadController.h**
 - **KeypadController.cpp**
-

KEYPADDRIVER CLASS REFERENCE

PUBLIC MEMBER FUNCTIONS

- **KeypadDriver** (emstream *ptrSerial, volatile uint8_t *inputPinRegister, volatile uint8_t *inputPortRegister, volatile uint8_t *inputDdrRegister)
This constructor sets up the keypad driver.
- char **checkInput** (void)
This methods checks for user input.

PROTECTED ATTRIBUTES

- emstream * **p_serial**
Pointer to serial.
 - volatile uint8_t * **pinRegister**
Pin register to check pin status.
 - volatile uint8_t * **portRegister**
Pin register to change pin status.
-

DETAILED DESCRIPTION

Definition at line 58 of file KeypadDriver.h.

CONSTRUCTOR & DESTRUCTOR DOCUMENTATION

KEYPADDRIVER::KEYPADDRIVER (EMSTREAM * PTRSERIAL, VOLATILE UINT8_T *
INPUTPINREGISTER, VOLATILE UINT8_T * INPUTPORTREGISTER, VOLATILE UINT8_T *
INPUTDDRREGISTER)

This constructor sets up the keypad driver.

The keypad driver is setup to load its associated ports and pins used for checking user input. This driver assumes an 8-pinhole bus is being used by the keypad.

PARAMETERS:

<i>ptrSerial</i>	A pointer to the serial port which writes debugging info.
<i>inputPinRegister</i>	A pointer to the pin register for reading pin values.
<i>inputPortRegister</i>	A pointer to the control register for setting pins high or low.
<i>inputDdrRegister</i>	A pointer to the register settings pins for input or output.

Definition at line 35 of file KeypadDriver.cpp.

References p_serial, pinRegister, and portRegister.

MEMBER FUNCTION DOCUMENTATION

CHAR KEYPADDRIVER::CHECKINPUT (VOID)

This methods checks for user input.

All pin combinations are checked for continuity to determine input row and column.

RETURNS:

Selected button character

Definition at line 59 of file KeypadDriver.cpp.

References keypadButtons, pinRegister, and portRegister.

Referenced by KeypadController::run().

THE DOCUMENTATION FOR THIS CLASS WAS GENERATED FROM THE FOLLOWING FILES:

- **KeypadDriver.h**
 - **KeypadDriver.cpp**
-

LCDDRIVER CLASS REFERENCE

PUBLIC MEMBER FUNCTIONS

- **LcdDriver** (emstream *ptrSerial, volatile uint8_t *inputPortRegister, volatile uint8_t *inputDdrRegister, uint8_t inputSsPin, **SpiMaster** *inputPtrSpiMaster)
This constructor sets up the LCD driver.
- void **displayOn** (void)
This method turns on the LCD display.
- void **displayOff** (void)
This method turns off the LCD display.
- void **printChar** (unsigned char inputChar)
This methods prints a character to the LCD.
- void **printString** (const char inputString[])
This methods prints a C-style string to the LCD.
- void **moveCursor** (uint8_t inputRow, uint8_t inputColumn)
This methods moves the cursor.
- void **clearScreen** (void)
This method clears the LCD.
- void **backspace** (void)
This method performs a backspace on the LCD.
- void **blinkOn** (void)
This method turns on the cursor.
- void **blinkOff** (void)
This method turns off the cursor.
- void **moveLeft** (uint8_t n)
This method moves the cursor left.
- void **moveRight** (uint8_t n)
This method moves the cursor right.

PROTECTED ATTRIBUTES

- emstream * **p_serial**
Pointer to serial.
- uint8_t **position**
Position of the cursor on screen.
- volatile uint8_t * **portRegister**
SPI chip select port.
- uint8_t **ssPin**
SPI chip select pin.
- **SpiMaster** * **ptrSpiMaster**
SPI protocol used to send commands to LCD.

DETAILED DESCRIPTION

Definition at line 68 of file LcdDriver.h.

CONSTRUCTOR & DESTRUCTOR DOCUMENTATION

LCDDRIVER::LCDDRIVER (EMSTREAM * PTRSERIAL, VOLATILE UINT8_T * INPUTPORTREGISTER, VOLATILE UINT8_T * INPUTDDRREGISTER, UINT8_T INPUTSSPIN, SPIMASTER * INPUTPTRSPIMASTER)

This constructor sets up the LCD driver.

The LCD driver is setup to communicate over an SPI communication protocol.

PARAMETERS:

<i>ptrSerial</i>	A pointer to the serial port which writes debugging info
<i>inputPortRegister</i>	A pointer to the control register for setting pins high or low
<i>inputDdrRegister</i>	A pointer to the register settings pins for input or output
<i>inputSsPin</i>	Pin for controlling SS chip select
<i>inputSpiMaster</i>	Pointer to SPI protocol driver

Definition at line 36 of file LcdDriver.cpp.

References blinkOff(), displayOn(), p_serial, portRegister, ptrSpiMaster, and ssPin.

MEMBER FUNCTION DOCUMENTATION

VOID LCDDRIVER::BACKSPACE (VOID)

This method performs a backspace on the LCD.

The backspace command is sent to the LCD

Definition at line 180 of file LcdDriver.cpp.

References delay_us(), portRegister, PREFIX, ptrSpiMaster, SpiMaster::send(), and ssPin.

Referenced by UiController::f4Input(), and UiController::f4Update().

VOID LCDDRIVER::BLINKOFF (VOID)

This method turns off the cursor.

The disable cursor command is sent to LCD.

Definition at line 200 of file LcdDriver.cpp.

References delay_us(), portRegister, PREFIX, ptrSpiMaster, SpiMaster::send(), and ssPin.

Referenced by UiController::f4Input(), and LcdDriver().

VOID LCDDRIVER::BLINKON (VOID)

This method turns on the cursor.

The enable cursor command is sent to LCD.

Definition at line 190 of file LcdDriver.cpp.

References delay_us(), portRegister, PREFIX, ptrSpiMaster, SpiMaster::send(), and ssPin.

Referenced by UiController::f4Input().

VOID LCDDRIVER::CLEARSCREEN (VOID)

This method clears the LCD.

All content is removed from LCD screen.

Definition at line 170 of file LcdDriver.cpp.

References delay_us(), portRegister, PREFIX, ptrSpiMaster, SpiMaster::send(), and ssPin.

Referenced by UiController::f1Display(), UiController::f2Display(), UiController::f3Display(), UiController::f4Display(), and UiController::initDisplay().

VOID LCDDRIVER::DISPLAYOFF (VOID)

This method turns off the LCD display.

The power off command is sent to the LCD display

Definition at line 86 of file LcdDriver.cpp.

References delay_us(), portRegister, PREFIX, ptrSpiMaster, SpiMaster::send(), and ssPin.

VOID LCDDRIVER::DISPLAYON (VOID)

This method turns on the LCD display.

The initialization command is sent to the LCD display

Definition at line 76 of file LcdDriver.cpp.

References delay_us(), portRegister, PREFIX, ptrSpiMaster, SpiMaster::send(), and ssPin.

Referenced by LcdDriver().

VOID LCDDRIVER::MOVECURSOR (UINT8_T INPUTROW, UINT8_T INPUTCOLUMN)

This methods moves the cursor.

Cursor move command is sent to the LCD.

PARAMETERS:

<i>inputRow</i>	Row to place cursor at
<i>inputColumn</i>	Column to place cursor at

Definition at line 125 of file LcdDriver.cpp.

References delay_us(), p_serial, portRegister, PREFIX, ptrSpiMaster, SpiMaster::send(), and ssPin.

Referenced by UiController::f1Display(), UiController::f1Update(), UiController::f2Display(), UiController::f2Update(), UiController::f3Display(), UiController::f3Update(), UiController::f4Display(), UiController::f4Update(), and UiController::initDisplay().

VOID LCDDRIVER::MOVELEFT (UINT8_T N)

This method moves the cursor left.

The cursor is moved left for the given input.

PARAMETERS:

<i>n</i>	Number of spaces to move
----------	--------------------------

Definition at line 211 of file LcdDriver.cpp.

References delay_us(), portRegister, PREFIX, ptrSpiMaster, SpiMaster::send(), and ssPin.

Referenced by UiController::f4Input().

VOID LCDDRIVER::MOVERIGHT (UINT8_T *N*)

This method moves the cursor right.

The cursor is moved right for the given input.

PARAMETERS:

<i>n</i>	Number of spaces to move
----------	--------------------------

Definition at line 225 of file LcdDriver.cpp.

References delay_us(), portRegister, PREFIX, ptrSpiMaster, SpiMaster::send(), and ssPin.

Referenced by UiController::f4Input(), and UiController::f4Update().

VOID LCDDRIVER::PRINTCHAR (UNSIGNED CHAR *INPUTCHAR*)

This methods prints a character to the LCD.

Prints a character to the LCD at cursor location.

PARAMETERS:

<i>inputChar</i>	Character to output to the screen
------------------	-----------------------------------

Definition at line 97 of file LcdDriver.cpp.

References delay_us(), p_serial, portRegister, PREFIX, ptrSpiMaster, SpiMaster::send(), and ssPin.

Referenced by UiController::f1Update(), UiController::f2Update(), UiController::f3Update(), UiController::f4Input(), UiController::f4Update(), and printString().

VOID LCDDRIVER::PRINTSTRING (CONST CHAR *INPUTSTRING*[])

This methods prints a C-style string to the LCD.

Prints characters until it reaches a null value in the given C string. String length is limited to 80 characters as a safety precaution.

PARAMETERS:

<i>inputString</i>	String to output to the screen
--------------------	--------------------------------

Definition at line 113 of file LcdDriver.cpp.

References printChar().

Referenced by UiController::f1Display(), UiController::f1Update(), UiController::f2Display(), UiController::f2Update(), UiController::f3Display(), UiController::f4Display(), and UiController::initDisplay().

THE DOCUMENTATION FOR THIS CLASS WAS GENERATED FROM THE FOLLOWING FILES:

- **LcdDriver.h**
 - **LcdDriver.cpp**
-

MOTORDRIVER CLASS REFERENCE

This class should run a motor driver on an AVR processor.

```
#include <MotorDriver.h>
```

PUBLIC MEMBER FUNCTIONS

- **MotorDriver** (emstream *, volatile uint8_t *inputStateRegister, volatile uint8_t *inputStateDDR, volatile uint8_t *inputPWMregister, volatile uint8_t *inputPWMDDR, uint8_t inputINA, uint8_t inputINB, uint8_t inputENDIAG, uint8_t inputPWM, volatile uint16_t *inputOCR1n)
This constructor sets up the motor driver.
- void **set_power** (int32_t)
This method detects and sets the direction and magnitude of an input power.
- void **brake** (void)
This method brakes the motor.
- int16_t **get_power** (void)
Returns current motor power variable.

PROTECTED ATTRIBUTES

- emstream * **ptr_to_serial**
The motor class uses this pointer to the serial port to say hello.
 - volatile uint8_t * **STATE_register**
Register for INA, INB, and ENDIAG pins.
 - volatile uint8_t * **STATE_DDR**
DDR for INA, INB, and ENDIAG pins.
 - volatile uint8_t * **PWM_register**
Register for PWM pin.
 - volatile uint8_t * **PWM_DDR**
DDR for PWM pin.
 - volatile uint16_t * **OCR1n**
Used for settings fast PWM in output compare.
 - uint8_t **INA_pin**
Pin number connected to INA on motor controller.
 - uint8_t **INB_pin**
Pin number connected to INB on motor controller.
 - uint8_t **EN_DAIG_pin**
Pin number connected to ENDIAG on motor controller.
 - uint8_t **PWM_pin**
Pin number connected to PWM on motor controller.
 - int32_t **power**
Current assigned power value for motor.
-

DETAILED DESCRIPTION

This class should run a motor driver on an AVR processor.

This class is setup to initialize the motor on the microcontroller and output power using PWM. The << operator is also overloaded for convenience.

Definition at line 45 of file MotorDriver.h.

CONSTRUCTOR & DESTRUCTOR DOCUMENTATION

```
MOTORDRIVER::MOTORDRIVER (EMSTREAM * P_SERIAL_PORT, VOLATILE UINT8_T *
    INPUTSTATeregister, VOLATILE UINT8_T * INPUTSTATEDDR, VOLATILE UINT8_T *
    INPUTPWMregister, VOLATILE UINT8_T * INPUTPWMDDDR, UINT8_T INPUTINA, UINT8_T
    INPUTINB, UINT8_T INPUTENDIAG, UINT8_T INPUTPWM, VOLATILE UINT16_T * INPUTOCR1N)
```

This constructor sets up the motor driver.

The motor driver is setup to load its associated ports and pins used for operation and to save its **OCRn** for adjusting duty cycle. Appropriate values are written to **TCCR1A** and **TCCR1B** to enter **fast PWM mode**.

PARAMETERS:

<i>p_serial_port</i>	A pointer to the serial port which writes debugging info.
<i>inputStateRegister</i>	A pointer to the register for INA, INB, and ENDIAG.
<i>inputStateDDR</i>	A pointer to the DDR for INA, INB, and ENDIAG.
<i>inputPWMregister</i>	A pointer to the register for PWM.
<i>inputPWMDDDR</i>	A pointer to the DDR for PWM.
<i>inputINA</i>	Pin number for INA.
<i>inputINB</i>	Pin number for INB.
<i>inputENDIAG</i>	Pin number for ENDIAG.
<i>inputPWM</i>	Pin number for PWM.
<i>inputOCR1n</i>	A pointer 16-bit OCR1n register, used for fast PWM.

Definition at line 52 of file MotorDriver.cpp.

References EN_DAIG_pin, INA_pin, INB_pin, OCR1n, power, ptr_to_serial, PWM_DDR, PWM_pin, PWM_register, STATE_DDR, and STATE_register.

MEMBER FUNCTION DOCUMENTATION

VOID MOTORDRIVER::BRAKE (VOID)

This method brakes the motor.

This method sets **INA** and **INB** to ground the motor leads, applying brake to the motor.

Definition at line 154 of file MotorDriver.cpp.

References INA_pin, INB_pin, and STATE_register.

Referenced by ActuatorController::run().

VOID MOTORDRIVER::SET_POWER (INT32_T INPUTPOWER)

This method detects and sets the direction and magnitude of an input power.

This code detects the sign of the **inputPower** and changes **INA** and **INB_pin** to spin motor appropriate direction, and then sets **OCR1n** register to new power value.

PARAMETERS:

<i>inputPower</i>	16-bit signed power value to apply to motor
-------------------	---

Definition at line 119 of file MotorDriver.cpp.

References INA_pin, INB_pin, OCR1n, power, and STATE_register.

Referenced by ActuatorController::run().

THE DOCUMENTATION FOR THIS CLASS WAS GENERATED FROM THE FOLLOWING FILES:

- **MotorDriver.h**
 - **MotorDriver.cpp**
-

POWERCONTROLLER CLASS REFERENCE

This task controls the power sensors and irradiance drivers.

```
#include <PowerController.h>
```

Inheritance diagram for PowerController:



PUBLIC MEMBER FUNCTIONS

- **PowerController** (const char *, unsigned portBASE_TYPE, size_t, emstream *, **IrradianceDriver** *, **VoltageSensorDriver** *, **CurrentSensorDriver** *)
This constructor sets up the power controller.
- void **run** (void)
This method updates the status of the input to the associated shared variable.

PROTECTED ATTRIBUTES

- **IrradianceDriver** * **ptrIrradianceDriver**
Pointer to the irradiance driver.
- **VoltageSensorDriver** * **ptrVoltageSensorDriver**
Pointer to a voltage driver.
- **CurrentSensorDriver** * **ptrCurrentSensorDriver**
Pointer to a current driver.

DETAILED DESCRIPTION

This task controls the power sensors and irradiance drivers.

The power drivers are run using files **IrradianceDriver.h** , **VoltageSensorDriver.h** , and **CurrentSensorDriver.h** . Code in this task communicates with pvTrainerMasterControl via shared variables.

Definition at line 49 of file PowerController.h.

CONSTRUCTOR & DESTRUCTOR DOCUMENTATION

```
POWERCONTROLLER::POWERCONTROLLER (CONST CHAR * A_NAME, UNSIGNED
PORTBASE_TYPE A_PRIORITY, SIZE_T A_STACK_SIZE, EMSTREAM * PTRSERIAL,
```

```

IRRADIANCEDRIVER * INPUTPTRIRRADIANCEDRIVER, VOLTAGESENSORDRIVER *
    INPUTPTRVOLTAGESENSORDRIVER, CURRENTSENSORDRIVER *
    INPUTPTRCURRENTSENSORDRIVER)

```

This constructor sets up the power controller.

The power controller constructor takes the appropriate data for its frt_task parameters.

PARAMETERS:

<i>a_name</i>	A character string which will be the name of this task
<i>a_priority</i>	The priority at which this task will initially run (default: 0)
<i>a_stack_size</i>	The size of this task's stack in bytes (default: configMINIMAL_STACK_SIZE)
<i>ptrSerial</i>	Pointer to a serial device (port, radio, SD card, etc.) which can be used by this task to communicate (default: NULL)
<i>inputPtrIrradianceDriver</i>	A pointer to an irradiance driver to be controlled
<i>inputPtrVoltageSensorDriver</i>	A pointer to a voltage driver to be controlled
<i>inputPtrCurrentSensorDriver</i>	A pointer to a current driver to be controlled

Definition at line 42 of file PowerController.cpp.

References ptrCurrentSensorDriver, ptrIrradianceDriver, and ptrVoltageSensorDriver.

THE DOCUMENTATION FOR THIS CLASS WAS GENERATED FROM THE FOLLOWING FILES:

- **PowerController.h**
 - **PowerController.cpp**
-

SPIMASTER CLASS REFERENCE

This contains the methods necessary communicate with SPI supported hardware.

```
#include <SpiMaster.h>
```

PUBLIC MEMBER FUNCTIONS

- **SpiMaster** (emstream *ptrSerial, volatile uint8_t *inputPinRegister, volatile uint8_t *inputPortRegister, volatile uint8_t *inputDdrRegister, uint8_t inputSsPin, uint8_t inputSclockPin, uint8_t inputMosiPin, uint8_t inputMisoPin)
This constructor sets up the keypad driver.
- void **send** (volatile uint8_t *inputPortRegister, uint8_t inputSsPin, uint8_t outByte)
This methods sends data over SPI.
- uint64_t **recieve** (volatile uint8_t *inputPortRegister, uint8_t inputSsPin, uint8_t inBits)
This methods recieves data over SPI.

PROTECTED ATTRIBUTES

- emstream * **p_serial**
Pointer to serial.
- volatile uint8_t * **pinRegister**
Pin register to read pin status.
- volatile uint8_t * **portRegister**
Pin register to change pin status.
- uint8_t **sclockPin**
SCLOCK pin.
- uint8_t **mosiPin**
MOSI pin.
- uint8_t **misoPin**
MISO pin.
- uint8_t **ssPin**
SS chip select pin.
- bool **transmitInProgress**
Indicates if data transmission is in progress.
- uint32_t **baudRate**
Baud Rate in Hz.
- uint16_t **timeDelay**
Time to delay during send/recieve.

DETAILED DESCRIPTION

This contains the methods necessary communicate with SPI supported hardware.

This class sets up the necessary pins and protocols to allow objects to communicate with external hardware.

Definition at line 44 of file SpiMaster.h.

CONSTRUCTOR & DESTRUCTOR DOCUMENTATION

```
SPIMASTER::SPIMASTER (EMSTREAM * PTRSERIAL, VOLATILE UINT8_T * INPUTPINREGISTER,
VOLATILE UINT8_T * INPUTPORTREGISTER, VOLATILE UINT8_T * INPUTDDRREGISTER, UINT8_T
INPUTSSPIN, UINT8_T INPUTSCLOCKPIN, UINT8_T INPUTMOSIPIN, UINT8_T INPUTMISOPIN)
```

This constructor sets up the keypad driver.

The keypad driver is setup to load its associated ports and pins used for checking user input.

PARAMETERS:

<i>ptrSerial</i>	A pointer to the serial port which writes debugging info.
<i>inputPinRegister</i>	A pointer to read if pins are high or low.
<i>inputPortRegister</i>	A pointer to the control register for setting pins high or low.
<i>inputDdrRegister</i>	A pointer to the register settings pins for input or output.
<i>inputSclockPin</i>	Pin for controlling SCLOCK.
<i>inputMosiPin</i>	Pin for controlling MOSI.
<i>inputMisoPin</i>	Pin for controlling MISO.
<i>inputMaxClockRate</i>	Maximum clock rate to talk to hardware with.

Definition at line 39 of file SpiMaster.cpp.

References baudRate, misoPin, mosiPin, p_serial, pinRegister, portRegister, sclockPin, ssPin, timeDelay, and transmitInProgress.

MEMBER FUNCTION DOCUMENTATION

```
UINT64_T SPIMASTER::RECIEVE (VOLATILE UINT8_T * INPUTPORTREGISTER, UINT8_T
INPUTSSPIN, UINT8_T INBITS)
```

This methods recieves data over SPI.

This method raises chip select and recieves data over the MISO line to the hardware.

PARAMETERS:

<i>inputPortRegister</i>	A pointer to the control register for setting pins high or low.
<i>inputSsPin</i>	Pin for controlling SS chip select.
<i>inBits</i>	Number of bits to receive in binary over SPI

RETURNS:

Received data

Definition at line 145 of file SpiMaster.cpp.

References `delay_us()`, `misoPin`, `pinRegister`, `portRegister`, `sclockPin`, `timeDelay`, and `transmitInProgress`.

Referenced by `AbsoluteEncoderDriver::read()`.

```
VOID SPIMASTER::SEND (VOLATILE UINT8_T * INPUTPORTREGISTER, UINT8_T INPUTSSPIN,
                     UINT8_T OUTBYTE)
```

This method sends data over SPI.

This method raises chip select and sends data over the MOSI line to the hardware.

PARAMETERS:

<i>inputPortRegister</i>	A pointer to the control register for setting pins high or low.
<i>inputSsPin</i>	Pin for controlling SS chip select.
<i>outByte</i>	Data to send in binary over SPI

Definition at line 89 of file SpiMaster.cpp.

References `delay_us()`, `mosiPin`, `portRegister`, `sclockPin`, `timeDelay`, and `transmitInProgress`.

Referenced by `LcdDriver::backspace()`, `LcdDriver::blinkOff()`, `LcdDriver::blinkOn()`, `LcdDriver::clearScreen()`, `LcdDriver::displayOff()`, `LcdDriver::displayOn()`, `LcdDriver::moveCursor()`, `LcdDriver::moveLeft()`, `LcdDriver::moveRight()`, and `LcdDriver::printChar()`.

THE DOCUMENTATION FOR THIS CLASS WAS GENERATED FROM THE FOLLOWING FILES:

- **SpiMaster.h**
 - **SpiMaster.cpp**
-

TASK_USER CLASS REFERENCE

```
#include <task_user.h>
```

Inheritance diagram for task_user:



PUBLIC MEMBER FUNCTIONS

- **task_user** (const char *, unsigned portBASE_TYPE, size_t, emstream *)
- void **run** (void)

PROTECTED MEMBER FUNCTIONS

- void **print_help_message** (void)
- void **show_status** (void)

DETAILED DESCRIPTION

This task interacts with the user for force him/her to do what he/she is told. What a rude task this is. Then again, computers tend to be that way; if they're polite with you, they're probably spying on you.

Definition at line 59 of file task_user.h.

CONSTRUCTOR & DESTRUCTOR DOCUMENTATION

TASK_USER::TASK_USER (CONST CHAR * A_NAME, UNSIGNED PORTBASE_TYPE A_PRIORITY, SIZE_T A_STACK_SIZE, EMSTREAM * P_SER_DEV)

This constructor creates a new data acquisition task. Its main job is to call the parent class's constructor which does most of the work.

PARAMETERS:

<i>a_name</i>	A character string which will be the name of this task
<i>a_priority</i>	The priority at which this task will initially run (default: 0)
<i>a_stack_size</i>	The size of this task's stack in bytes (default: configMINIMAL_STACK_SIZE)

<i>p_ser_dev</i>	Pointer to a serial device (port, radio, SD card, etc.) which can be used by this task to communicate (default: NULL)
------------------	---

Definition at line 53 of file task_user.cpp.

MEMBER FUNCTION DOCUMENTATION

VOID TASK_USER::PRINT_HELP_MESSAGE (VOID) [PROTECTED]

This method prints a simple help message.

Definition at line 204 of file task_user.cpp.

References PROGRAM_VERSION.

Referenced by run().

VOID TASK_USER::RUN (VOID)

This method is called by the RTOS once to run the task loop for ever and ever.

This task interacts with the user for force him/her to do what he/she is told. It is just following the modern government model of "This is the land of the free...free to do exactly what you're told."

Definition at line 69 of file task_user.cpp.

References print_help_message(), print_ser_queue, and show_status().

VOID TASK_USER::SHOW_STATUS (VOID) [PROTECTED]

This method displays information about the status of the system, including the following:

- The name and version of the program
- The name, status, priority, and free stack space of each task
- Processor cycles used by each task
- Amount of heap space free and setting of RTOS tick timer

Definition at line 227 of file task_user.cpp.

References PROGRAM_VERSION.

Referenced by run().

THE DOCUMENTATION FOR THIS CLASS WAS GENERATED FROM THE FOLLOWING FILES:

- **task_user.h**
 - task_user.cpp
-

TEMPCONTROLLER CLASS REFERENCE

This task controls the sensor and fan driver.

```
#include <TempController.h>
```

Inheritance diagram for TempController:



PUBLIC MEMBER FUNCTIONS

- **TempController** (const char *, unsigned portBASE_TYPE, size_t, emstream *, **avr_ds182x** *, **avr_ds182x** *, **avr_ds182x** *, **avr_ds182x** *, **FanDriver** *, shared_data< int16_t > *, shared_data< int16_t > *, shared_data< int16_t > *, shared_data< int16_t > *, shared_data< int16_t > *, shared_data< bool > *)
This constructor sets up the temperature controller.
- void **run** (void)
This method updates the status of the input to the associated shared variable.

PROTECTED ATTRIBUTES

- **avr_ds182x** * **ptrTempSensor1**
Pointer to temperature sensor 1.
- **avr_ds182x** * **ptrTempSensor2**
Pointer to temperature sensor 2.
- **avr_ds182x** * **ptrTempSensor3**
Pointer to temperature sensor 3.
- **avr_ds182x** * **ptrTempSensor4**
Pointer to temperature sensor 4.
- uint16_t **averageTemp**
Averaged temperature readings.
- **FanDriver** * **ptrFanDriver**
Pointer to the fan driver.
- shared_data< int16_t > * **ptrSharedTemp1**
Pointer to shared temperature reading 1.
- shared_data< int16_t > * **ptrSharedTemp2**
Pointer to shared temperature reading 2.
- shared_data< int16_t > * **ptrSharedTemp3**
Pointer to shared temperature reading 3.
- shared_data< int16_t > * **ptrSharedTemp4**
Pointer to shared temperature reading 4.
- shared_data< int16_t > * **ptrSharedTempAverage**

Pointer to shared temperature reading averaged.

- `shared_data< bool > * ptrSharedFanState`

Pointer to fan control state.

- `int16_t lastTemp1`

Last temperature recorded.

- `int16_t lastTemp2`

Last temperature recorded.

- `int16_t lastTemp3`

Last temperature recorded.

- `int16_t lastTemp4`

Last temperature recorded.

- `int16_t tempTemp`

Temp temperature for error reduction.

DETAILED DESCRIPTION

This task controls the sensor and fan driver.

The temperature driver is run using files `avr_ds182x.h`, `avr_ds182x.cpp`, `FanDriver.h`, and `FanDriver.cpp`. Code in this task communicates with `pvTrainerMasterControl` via shared variables.

Definition at line 49 of file `TempController.h`.

CONSTRUCTOR & DESTRUCTOR DOCUMENTATION

```
TEMPCONTROLLER::TEMPCONTROLLER (CONST CHAR * A_NAME, UNSIGNED PORTBASE_TYPE
    A_PRIORITY, SIZE_T A_STACK_SIZE, EMSTREAM * PTRSERIAL, AVR_DS182X *
    INPUTPTRTEMPSENSOR1, AVR_DS182X * INPUTPTRTEMPSENSOR2, AVR_DS182X *
    INPUTPTRTEMPSENSOR3, AVR_DS182X * INPUTPTRTEMPSENSOR4, FANDRIVER *
    INPUTPTRFANDRIVER, SHARED_DATA< INT16_T > * INPUTPTRSHAREDTEMP1, SHARED_DATA<
    INT16_T > * INPUTPTRSHAREDTEMP2, SHARED_DATA< INT16_T > * INPUTPTRSHAREDTEMP3,
    SHARED_DATA< INT16_T > * INPUTPTRSHAREDTEMP4, SHARED_DATA< INT16_T > *
    INPUTPTRSHAREDTEMPAVERAGE, SHARED_DATA< BOOL > * INPUTPTRSHAREDFANSTATE)
```

This constructor sets up the temperature controller.

The temperature controller constructor takes the appropriate data for its `frt_task` parameters.

PARAMETERS:

<code>a_name</code>	A character string which will be the name of this task
<code>a_priority</code>	The priority at which this task will initially run (default: 0)
<code>a_stack_size</code>	The size of this task's stack in bytes (default: <code>configMINIMAL_STACK_SIZE</code>)

<i>ptrSerial</i>	Pointer to a serial device (port, radio, SD card, etc.) which can be used by this task to communicate (default: NULL)
<i>inputPtrTempSensor1</i>	A pointer to a temp sensor driver to be controlled
<i>inputPtrTempSensor2</i>	A pointer to a temp sensor driver to be controlled
<i>inputPtrTempSensor3</i>	A pointer to a temp sensor driver to be controlled
<i>inputPtrTempSensor4</i>	A pointer to a temp sensor driver to be controlled
<i>inputPtrFanDriver</i>	A pointer to a fan driver to be controlled
<i>inputPtrSharedTemp1</i>	A pointer to a shared temperature value
<i>inputPtrSharedTemp2</i>	A pointer to a shared temperature value
<i>inputPtrSharedTemp3</i>	A pointer to a shared temperature value
<i>inputPtrSharedTemp4</i>	A pointer to a shared temperature value
<i>inputPtrSharedTempAverage</i>	A pointer to the averaged shared temperature value

Definition at line 48 of file TempController.cpp.

References averageTemp, lastTemp1, lastTemp2, lastTemp3, lastTemp4, ptrFanDriver, ptrSharedFanState, ptrSharedTemp1, ptrSharedTemp2, ptrSharedTemp3, ptrSharedTemp4, ptrSharedTempAverage, ptrTempSensor1, ptrTempSensor2, ptrTempSensor3, ptrTempSensor4, and tempTemp.

THE DOCUMENTATION FOR THIS CLASS WAS GENERATED FROM THE FOLLOWING FILES:

- **TempController.h**
 - **TempController.cpp**
-

UICONTROLLER CLASS REFERENCE

This task controls the LCD display.

```
#include <UiController.h>
```

Inheritance diagram for UiController:



PUBLIC MEMBER FUNCTIONS

- **UiController** (const char *, unsigned portBASE_TYPE, size_t, emstream *, shared_data< char > *, **LcdDriver** *)
This constructor sets up the keypad controller.
- void **run** (void)
This method updates the status of the input to the associated shared variable.
- void **initDisplay** (void)
This method displays init screen.
- void **f1Display** (void)
This method sets up the display state for positions.
- void **f1Update** (void)
This method updates position values.
- void **f2Display** (void)
This method sets up the display state for positions.
- void **f2Update** (void)
This method updates position values.
- void **f3Display** (void)
This method sets up the display state for temperatures.
- void **f3Update** (void)
This method updates temperatures values.
- void **f4Display** (void)
This method sets up the display state for receiving positions.
- void **f4Update** (void)
This method handles switching input mods.
- void **f4Input** (char input)
This method handles receiving positions.
- bool **checkAngle0** (int16_t newAngle)
This method checks if valid for angle0.
- bool **checkAngle1** (int16_t newAngle)
This method checks if valid for angle1.

- **bool checkAngleAzimuth** (int16_t newAngle)
This method checks if valid for angleAzimuth.
- **bool checkAngleElevation** (int16_t newAngle)
This method checks if valid for angleElevation.
- **void setAngle0** (int16_t newAngle)
This method updates angle0.
- **void setAngle1** (int16_t newAngle)
This method updates angle1.
- **void setAngleAzimuth** (int16_t newAngle)
This method updates angleAzimuth.
- **void setAngleElevation** (int16_t newAngle)
This method updates angleElevation.

PROTECTED ATTRIBUTES

- **uint8_t screenState**
LCD display state.
- **char keypadInput**
Input collected from keypad.
- **shared_data< char > * ptrSharedInput**
Pointer to the keypad share.
- **LcdDriver * ptrLcdDriver**
Pointer to the LCD driver.
- **int16_t outputValue**
Data to be sent as output to LCD.
- **bool sphereMode**
Current coordinate mode.
- **int16_t angle0**
Main angle 0.
- **int16_t angle1**
Main angle 1.
- **int16_t angleAzimuth**
Azimuth angle value.
- **int16_t angleElevation**
Elevation angle value.
- **uint8_t selectedAngle**
Angle to set on screen.
- **uint8_t charCount**
Number of characters printed to screen.
- **uint8_t inputChars [3]**
Captured input from keypad.
- **bool negative**
Sign of user input.

DETAILED DESCRIPTION

This task controls the LCD display.

The LCD driver is run using files **LCDDriver.h** and **LCDDriver.cpp**. Code in this task communicates with pvTrainerMasterControl via shared variables.

Definition at line 47 of file UiController.h.

CONSTRUCTOR & DESTRUCTOR DOCUMENTATION

```
UICONTROLLER::UICONTROLLER (CONST CHAR * A_NAME, UNSIGNED PORTBASE_TYPE
A_PRIORITY, SIZE_T A_STACK_SIZE, EMSTREAM * PTRSERIAL, SHARED_DATA< CHAR > *
INPUTPTRSHAREDINPUT, LCDDRIVER * INPUTPTRLCDDRIVER)
```

This constructor sets up the keypad controller.

The keypad controller constructor takes the appropriate data for its frt_task parameters.

PARAMETERS:

<i>a_name</i>	A character string which will be the name of this task
<i>a_priority</i>	The priority at which this task will initially run (default: 0)
<i>a_stack_size</i>	The size of this task's stack in bytes (default: configMINIMAL_STACK_SIZE)
<i>ptrSerial</i>	Pointer to a serial device (port, radio, SD card, etc.) which can be used by this task to communicate (default: NULL)
<i>inputPtrKeypadDriver</i>	A pointer to the keypad to be monitored
<i>inputPtrLcdDriver</i>	A pointer to the LCD to be controlled

Definition at line 40 of file UiController.cpp.

References angle0, angle1, angleAzimuth, angleElevation, charCount, inputChars, keypadInput, negative, outputValue, ptrLcdDriver, ptrSharedInput, screenState, selectedAngle, and sphereMode.

MEMBER FUNCTION DOCUMENTATION

```
BOOL UICONTROLLER::CHECKANGLE0 (INT16_T NEWANGLE)
```

This method checks if valid for angle0.

Angle 0 is checked with respect to physical bounds.

PARAMETERS:

<i>newAngle</i>	New value for angle0
-----------------	----------------------

Definition at line 613 of file UiController.cpp.

Referenced by setAngleAzimuth(), and setAngleElevation().

BOOL UICONTROLLER::CHECKANGLE1 (INT16_T NEWANGLE)

This method checks if valid for angle1.

angle1 is checked with respect to physical bounds.

PARAMETERS:

<i>newAngle</i>	New value for angle1
-----------------	----------------------

Definition at line 628 of file UiController.cpp.

Referenced by setAngleAzimuth(), and setAngleElevation().

BOOL UICONTROLLER::CHECKANGLEAZIMUTH (INT16_T NEWANGLE)

This method checks if valid for angleAzimuth.

angleAzimuth is checked with respect to physical bounds.

PARAMETERS:

<i>newAngle</i>	New value for angleAzimuth
-----------------	----------------------------

Definition at line 643 of file UiController.cpp.

BOOL UICONTROLLER::CHECKANGLEELEVATION (INT16_T NEWANGLE)

This method checks if valid for angleElevation.

angleElevation is checked with respect to physical bounds.

PARAMETERS:

<i>newAngle</i>	New value for angleElevation
-----------------	------------------------------

Definition at line 658 of file UiController.cpp.

VOID UICONTROLLER::F1DISPLAY (VOID)

This method sets up the display state for positions.

The screen is cleared and the proper prompts for position are printed.

Definition at line 199 of file UiController.cpp.

References LcdDriver::clearScreen(), f1Update(), LcdDriver::moveCursor(), LcdDriver::printString(), ptrLcdDriver, and screenState.

Referenced by f4Input(), and run().

VOID UICONTROLLER::F1UPDATE (VOID)

This method updates position values.

Updated position values are printed to the LCD.

Definition at line 220 of file UiController.cpp.

References angle0, angle1, LcdDriver::moveCursor(), outputValue, LcdDriver::printChar(), LcdDriver::printString(), ptrLcdDriver, sharedEncoder1, sharedEncoder2, and sharedPowerState.

Referenced by f1Display(), and run().

VOID UICONTROLLER::F2DISPLAY (VOID)

This method sets up the display state for positions.

The screen is cleared and the proper prompts for position are printed.

Definition at line 279 of file UiController.cpp.

References LcdDriver::clearScreen(), f2Update(), LcdDriver::moveCursor(), LcdDriver::printString(), ptrLcdDriver, and screenState.

Referenced by f4Input(), and run().

VOID UICONTROLLER::F2UPDATE (VOID)

This method updates position values.

Updated position values are printed to the LCD.

Definition at line 300 of file UiController.cpp.

References angleAzimuth, angleElevation, azimuthTransform(), elevationTransform(), LcdDriver::moveCursor(), outputValue, LcdDriver::printChar(), LcdDriver::printString(), ptrLcdDriver, sharedEncoder1, sharedEncoder2, and sharedPowerState.

Referenced by f2Display(), and run().

VOID UICONTROLLER::F3DISPLAY (VOID)

This method sets up the display state for temperatures.

The screen is cleared and the proper prompts for temperature are printed.

Definition at line 359 of file UiController.cpp.

References LcdDriver::clearScreen(), f3Update(), LcdDriver::moveCursor(), LcdDriver::printString(), ptrLcdDriver, and screenState.

Referenced by run().

VOID UICONTROLLER::F3UPDATE (VOID)

This method updates temperatures values.

Updated temperatures values are printed to the LCD.

Definition at line 379 of file UiController.cpp.

References LcdDriver::moveCursor(), outputValue, LcdDriver::printChar(), ptrLcdDriver, sharedIrradiance, sharedTemp1, sharedTemp2, sharedTemp3, sharedTemp4, and sharedTempAverage.

Referenced by f3Display(), and run().

VOID UICONTROLLER::F4DISPLAY (VOID)

This method sets up the display state for receiving positions.

The screen is cleared and the user is prompted for input positions.

Definition at line 422 of file UiController.cpp.

References LcdDriver::clearScreen(), f4Update(), LcdDriver::moveCursor(), LcdDriver::printString(), ptrLcdDriver, screenState, and selectedAngle.

Referenced by run().

VOID UICONTROLLER::F4INPUT (CHAR *INPUT*)

This method handles receiving positions.

The user input is adjusted on screen.

PARAMETERS:

<i>input</i>	Character input received from keypad
--------------	--------------------------------------

Definition at line 533 of file UiController.cpp.

References LcdDriver::backspace(), LcdDriver::blinkOff(), LcdDriver::blinkOn(), charCount, f1Display(), f2Display(), f4Update(), inputChars, LcdDriver::moveLeft(), LcdDriver::moveRight(), negative, LcdDriver::printChar(), ptrLcdDriver, selectedAngle, setAngle0(), setAngle1(), setAngleAzimuth(), and setAngleElevation().

Referenced by run().

VOID UICONTROLLER::F4UPDATE (VOID)

This method handles switching input mods.

The user input is selection is updated

Definition at line 443 of file UiController.cpp.

References angle0, angle1, angleAzimuth, angleElevation, LcdDriver::backspace(), charCount, inputChars, LcdDriver::moveCursor(), LcdDriver::moveRight(), negative, outputValue, LcdDriver::printChar(), ptrLcdDriver, and selectedAngle.

Referenced by f4Display(), and f4Input().

VOID UICONTROLLER::INITDISPLAY (VOID)

This method displays init screen.

The screen is cleared and the proper prompts for position are printed.

Definition at line 181 of file UiController.cpp.

References LcdDriver::clearScreen(), delay_ms(), LcdDriver::moveCursor(), LcdDriver::printString(), and ptrLcdDriver.

Referenced by run().

VOID UICONTROLLER::SETANGLE0 (INT16_T *NEWANGLE*)

This method updates angle0.

Angle 0 is updated with respect to physical bounds.

PARAMETERS:

<i>newAngle</i>	New value for angle0
-----------------	----------------------

Definition at line 673 of file UiController.cpp.

References angle0, angle1, angleAzimuth, angleElevation, azimuthTransform(), and elevationTransform().

Referenced by f4Input(), run(), setAngleAzimuth(), and setAngleElevation().

VOID UICONTROLLER::SETANGLE1 (INT16_T NEWANGLE)

This method updates angle1.

angle1 is updated with respect to physical bounds.

PARAMETERS:

<i>newAngle</i>	New value for angle1
-----------------	----------------------

Definition at line 690 of file UiController.cpp.

References angle0, angle1, angleAzimuth, angleElevation, azimuthTransform(), and elevationTransform().

Referenced by f4Input(), run(), setAngleAzimuth(), and setAngleElevation().

VOID UICONTROLLER::SETANGLEAZIMUTH (INT16_T NEWANGLE)

This method updates angleAzimuth.

angleAzimuth is updated with respect to physical bounds.

PARAMETERS:

<i>newAngle</i>	New value for angleAzimuth
-----------------	----------------------------

Definition at line 707 of file UiController.cpp.

References angle0, angle0Transform(), angle1, angle1Transform(), angleAzimuth, angleElevation, checkAngle0(), checkAngle1(), setAngle0(), and setAngle1().

Referenced by f4Input(), and run().

VOID UICONTROLLER::SETANGLEELEVATION (INT16_T NEWANGLE)

This method updates angleElevation.

angleElevation is updated with respect to physical bounds.

PARAMETERS:

<i>newAngle</i>	New value for angleElevation
-----------------	------------------------------

Definition at line 732 of file UiController.cpp.

References angle0, angle0Transform(), angle1, angle1Transform(), angleAzimuth, angleElevation, checkAngle0(), checkAngle1(), setAngle0(), and setAngle1().

Referenced by f4Input(), and run().

THE DOCUMENTATION FOR THIS CLASS WAS GENERATED FROM THE FOLLOWING FILES:

- **UiController.h**
- **UiController.cpp**

VOLTAGESENSORDRIVER CLASS REFERENCE

This file contains the methods necessary to read panel voltage.

```
#include <VoltageSensorDriver.h>
```

PUBLIC MEMBER FUNCTIONS

- **VoltageSensorDriver** (emstream *ptrSerial, shared_data< int16_t > *inputPtrSharedVoltage, **adc** *inputPtrAdc, uint8_t inputVoltagePin)
This constructor sets up the voltage driver.
- void **read** (void)
This methods reads the output voltage and returns panel voltage.

PROTECTED ATTRIBUTES

- emstream * **p_serial**
Pointer to serial.
 - shared_data< int16_t > * **ptrSharedVoltage**
Pointer to shared current value.
 - **adc** * **ptrAdc**
Pointer to ADC.
 - uint8_t **voltagePin**
Pin from which to read voltage.
-

DETAILED DESCRIPTION

This file contains the methods necessary to read panel voltage.

This driver is used to initialize and read from a the panel voltage.

Definition at line 41 of file VoltageSensorDriver.h.

CONSTRUCTOR & DESTRUCTOR DOCUMENTATION

```
VOLTAGESENSORDRIVER::VOLTAGESENSORDRIVER (EMSTREAM * PTRSERIAL, SHARED_DATA<
INT16_T > * INPUTPTRSHAREDVOLTAGE, ADC * INPUTPTRADC, UINT8_T INPUTVOLTAGEPIN)
```

This constructor sets up the voltage driver.

The voltage sensor driver is setup to read from a voltage divider connected to the solar panel output.

PARAMETERS:

<i>ptrSerial</i>	A pointer to the serial port which writes debugging info.
<i>inputPtrSharedVoltage</i>	Pointer to shared variable for voltage.
<i>inputPtrAdc</i>	A pointer to the ADC used to read voltage.
<i>inputVoltagePin</i>	Pin number to read current from.

Definition at line 35 of file VoltageSensorDriver.cpp.

References p_serial, ptrAdc, ptrSharedVoltage, and voltagePin.

MEMBER FUNCTION DOCUMENTATION

VOID VOLTAGESENSORDRIVER::READ (VOID)

This methods reads the output voltage and returns panel voltage.

The pin voltage is read and is then scaled from voltage divider.

Definition at line 49 of file VoltageSensorDriver.cpp.

References p_serial, ptrAdc, ptrSharedVoltage, adc::read_once(), and voltagePin.

THE DOCUMENTATION FOR THIS CLASS WAS GENERATED FROM THE FOLLOWING FILES:

- **VoltageSensorDriver.h**
- **VoltageSensorDriver.cpp**

FILE DOCUMENTATION

ABSOLUTEENCODERDRIVER.CPP FILE REFERENCE

```
#include "AbsoluteEncoderDriver.h"
```

DETAILED DESCRIPTION

This file contains the methods necessary to initialize and control an absolute encoder using SPI communication protocol.

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **AbsoluteEncoderDriver.cpp**.

ABSOLUTEENCODERDRIVER.H FILE REFERENCE

```
#include "emstream.h"
#include "frt_text_queue.h"
#include "frt_queue.h"
#include "frt_shared_data.h"
#include "shares.h"
#include "queue.h"
#include "semphr.h"
#include "SpiMaster.h"
```

CLASSES

- class **AbsoluteEncoderDriver**

This driver contains the methods necessary to initialize and control an encoder screen.

DETAILED DESCRIPTION

This file contains the methods necessary to initialize and control an absolute encoder using SPI communication protocol.

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **AbsoluteEncoderDriver.h**.

ACTUATORCONTROLLER.H FILE REFERENCE

```
#include "emstream.h"
#include "FreeRTOS.h"
#include "task.h"
#include "frt_task.h"
#include "frt_text_queue.h"
#include "frt_queue.h"
#include "frt_shared_data.h"
#include "shares.h"
#include "queue.h"
#include "semphr.h"
#include "rs232int.h"
#include "AbsoluteEncoderDriver.h"
#include "MotorDriver.h"
```

CLASSES

- class **ActuatorController**

This task controls the motor driver to make the actuators work.

DETAILED DESCRIPTION

This file contains the header for a task class that controls the functionality of a motor using an input from the user. This controller will be able to control several motors by selecting the speed, direction, and state of each motor.

Revisions:

- 04-18-2013 Actuator task setup from motor task structure

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **ActuatorController.h**.

ADC.CPP FILE REFERENCE

```
#include <stdlib.h>
#include <avr/io.h>
#include "rs232int.h"
#include "adc.h"
```

FUNCTIONS

- `emstream & operator<< (emstream &serpt, adc &a2d)`
This overloaded operator "prints the A/D converter."
-

DETAILED DESCRIPTION

This file contains a very simple A/D converter driver. The driver is hopefully thread safe in FreeRTOS due to the use of a mutex to prevent its use by multiple tasks at the same time. There is no protection from priority inversion, however, except for the priority elevation in the mutex. This program uses the appropriate names from ATmega documentation to help prevent obfuscation in the A/D converter class object.

Revisions:

- 01-15-2008 JRR Original (somewhat useful) file
- 10-11-2012 JRR Less original, more useful file with FreeRTOS mutex added
- 10-12-2012 JRR There was a bug in the mutex code, and it has been fixed
- 04-10-2013 WPS Program made functional and readable for Lab 1.

License: This file is copyright 2012 by JR Ridgely and released under the Lesser GNU Public License, version 2. It intended for educational use only, but its use is not limited thereto.

Definition in file **adc.cpp**.

FUNCTION DOCUMENTATION

EMSTREAM& OPERATOR<< (EMSTREAM & SERPT, ADC & A2D)

This overloaded operator "prints the A/D converter."

The << operator returns the pin values of the **ADMUX** register and the **ADCSRA** registers along with the last converted voltage value.

PARAMETERS:

<i>serpt</i>	Reference to a serial port to which the printout will be printed
<i>a2d</i>	Reference to the A/D driver which is being printed

RETURNS:

A reference to the same serial device on which we write information. This is used to string together things to write with "<<" operators

Definition at line 140 of file adc.cpp.

References adc::returnValue.

ADC.H FILE REFERENCE

```
#include "emstream.h"
#include "frt_text_queue.h"
#include "frt_queue.h"
#include "frt_shared_data.h"
#include "shares.h"
#include "queue.h"
#include "semphr.h"
```

CLASSES

- class **adc**

THIS CLASS SHOULD RUN THE A/D CONVERTER ON AN AVR PROCESSOR. FUNCTIONS

- emstream & **operator**<< (emstream &, **adc** &)
This overloaded operator "prints the A/D converter."

DETAILED DESCRIPTION

This file contains a very simple A/D converter driver. The driver is hopefully thread safe in FreeRTOS due to the use of a mutex to prevent its use by multiple tasks at the same time. There is no protection from priority inversion, however, except for the priority elevation in the mutex.

Revisions:

- 01-15-2008 JRR Original (somewhat useful) file
- 10-11-2012 JRR Less original, more useful file with FreeRTOS mutex added
- 10-12-2012 JRR There was a bug in the mutex code, and it has been fixed
- 04-10-2013 WPS Driver modified to include custom data type for A/D read

License: This file is copyright 2012 by JR Ridgely and released under the Lesser GNU Public License, version 2. It intended for educational use only, but its use is not limited thereto.

Definition in file **adc.h**.

FUNCTION DOCUMENTATION

EMSTREAM& OPERATOR<< (EMSTREAM & *SERPT*, ADC & A2D)

This overloaded operator "prints the A/D converter."

The << operator returns the pin values of the **ADMUX** register and the **ADCSRA** registers along with the last converted voltage value.

PARAMETERS:

<i>serpt</i>	Reference to a serial port to which the printout will be printed
--------------	--

<i>a2d</i>	Reference to the A/D driver which is being printed
------------	--

RETURNS:

A reference to the same serial device on which we write information. This is used to string together things to write with "<<" operators

Definition at line 140 of file adc.cpp.

References adc::returnValue.

AVR_1WIRE.CPP FILE REFERENCE

```
#include <stdlib.h>
#include <avr/io.h>
#include "FreeRTOS.h"
#include "task.h"
#include "avr_1wire.h"
```

DETAILED DESCRIPTION

This file contains a class which interfaces an AVR processor with devices on a bit-banged One-Wire Interface. The one-wire interface is used by a number of chips from Dallas Semiconductor (which has merged with Maxim IC). The term "bit-banged" means that this class uses any old I/O port pins for the data line. There can be several 1-wire interfaces in a program, in case the user wants to talk to several devices on different buses instead of connecting all the one-wire devices on the same bus. This can be useful to save power, as the buses can be powered individually; also, some 1-wire devices like different timing than others.

Revisions:

- 07-21-2007 JRR Created this file for Two-Wire Interfaces
- 12-17-2007 JRR Modified to work with the One-Wire Interface
- 07-11-2008 JRR Changed to use port pointers so several I/O ports can be used
- 07-12-2008 JRR Added debugging port which works with STL_DEBUG in stl_task.h
- 03-21-2009 JRR Changed debugging to the global debugging system
- 11-11-2011 JRR Cleaned up formatting, changed to .cpp
- 12-15-2011 JRR Changed read delay from AOWI_45us_D to AOWI_15us_D and fixed a timing bug in **avr_1wire::read_bit()**
- 12-01-2012 JRR Changed to work in new FreeRTOS based ME405 environment, made timing fully F_CPU dependent

License: This file copyright 2007-2012 by JR Ridgely. It is released under the Lesser GNU public license, version 2. It is intended for educational use only, but it may be used for any purpose permitted under the LGPL. The author has no control over the uses of this software and cannot be responsible for any consequences of such use.

Definition in file **avr_1wire.cpp**.

AVR_1WIRE.H FILE REFERENCE

```
#include "emstream.h"
```

CLASSES

- class **avr_1wire**

THIS CLASS IMPLEMENTS A BIT-BANGED ONE-WIRE INTERFACE (OWI) PORT.

MACROS

- `#define AOWI_DELAY(x)` for (volatile uint16_t _awi_dl = 0; _awi_dl < (x); _awi_dl++)
This macro implements a rather dumb delay loop.

VARIABLES

- const uint16_t **AOWI_RETRIES** = 10000
*This define prevents this file from being included more than once in a *.cc file.*
 - const uint8_t **AOWI_NUM_IDS** = 4
This is the size of a table which can hold 64-bit device identifiers.
 - const uint16_t **AOWI_RESET_D** = (uint16_t)(F_CPU / 30000L)
 - const uint16_t **AOWI_PRES_D** = (uint16_t)(F_CPU / 250000L)
 - const uint16_t **AOWI_PRES_END** = (uint16_t)(F_CPU / 100L)
 - const uint16_t **AOWI_1us_D**
 - const uint16_t **AOWI_15us_D** = (uint16_t)(F_CPU / 1800000L)
 - const uint16_t **AOWI_45us_D** = (**AOWI_15us_D** * 3)
 - const uint16_t **AOWI_90us_D** = (**AOWI_15us_D** * 6)
-

DETAILED DESCRIPTION

This file contains a class which interfaces an AVR processor with devices on a bit-banded One-Wire Interface. The one-wire interface is used by a number of chips from Dallas Semiconductor (which has merged with Maxim IC). The term "bit-banded" means that this class uses any old I/O port pins for the data line. There can be several 1-wire interfaces in a program, in case the user wants to talk to several devices on different buses instead of connecting all the one-wire devices on the same bus. This can be useful to save power, as the buses can be powered individually; also, some 1-wire devices like different timing than others.

Revisions:

- 07-21-2007 JRR Created this file for Two-Wire Interfaces
- 12-17-2007 JRR Modified to work with the One-Wire Interface
- 07-11-2008 JRR Changed to use port pointers so several I/O ports can be used
- 07-12-2008 JRR Added debugging port which works with STL_DEBUG in stl_task.h
- 03-21-2009 JRR Changed debugging to the global debugging system
- 11-11-2011 JRR Cleaned up formatting, changed to .cpp
- 12-15-2011 JRR Changed read delay from AOWI_45us_D to AOWI_15us_D and fixed a timing bug in **avr_1wire::read_bit()**
- 12-01-2012 JRR Changed to work in new FreeRTOS based ME405 environment, made timing fully F_CPU dependent

License: This file copyright 2007-2012 by JR Ridgely. It is released under the Lesser GNU public license, version 2. It is intended for educational use only, but it may be used for any purpose permitted under the LGPL. The author has no control over the uses of this software and cannot be responsible for any consequences of such use.

Definition in file **avr_1wire.h**.

VARIABLE DOCUMENTATION

CONST UINT16_T AOWI_15US_D = (UINT16_T)(F_CPU / 1800000L)

This is a delay counter for producing an approximately 15 microsecond delay. (F_CPU / 1800000) has been known to work.

Definition at line 96 of file avr_1wire.h.

Referenced by avr_1wire::read_bit(), and avr_1wire::read_byte().

CONST UINT16_T AOWI_1US_D

Initial value:=

```
(uint16_t)((F_CPU / 6000000L) > 0 ? (F_CPU / 6000000L) : 1)
```

This is a delay counter for producing an approximately 1 microsecond delay. (F_CPU / 6000000) has worked in the past, but (F_CPU / 4000000L) is used because it doesn't cause a zero when F_CPU is 4 MHz.

Definition at line 90 of file avr_1wire.h.

Referenced by avr_1wire::read_bit(), avr_1wire::read_byte(), avr_1wire::write_0(), and avr_1wire::write_1().

CONST UINT16_T AOWI_45US_D = (AOWI_15US_D * 3)

This is a delay counter for producing an approximately 45 microsecond delay. (F_CPU / 500000) has worked in the past.

Definition at line 101 of file avr_1wire.h.

Referenced by avr_1wire::read_bit(), and avr_1wire::read_byte().

CONST UINT16_T AOWI_90US_D = (AOWI_15US_D * 6)

This is a delay counter for producing an approximately 90 microsecond delay. (F_CPU / 250000) has been known to work.

Definition at line 106 of file avr_1wire.h.

Referenced by avr_1wire::avr_1wire(), avr_1wire::reset(), avr_1wire::write_0(), and avr_1wire::write_1().

CONST UINT16_T AOWI_PRES_D = (UINT16_T)(F_CPU / 250000L)

This is the length of time after the end of a reset pulse to wait for a presence pulse. It should usually be around 70 microseconds. (F_CPU / 250000L) has been seen to work.

Definition at line 79 of file avr_1wire.h.

Referenced by avr_1wire::auto_timing(), and avr_1wire::reset().

CONST UINT16_T AOWI_PRES_END = (UINT16_T)(F_CPU / 100L)

This is the number of retries to wait for the presence pulse to end. F_CPU / 100L has been known to work.

Definition at line 84 of file avr_1wire.h.

Referenced by avr_1wire::auto_timing(), and avr_1wire::reset().

CONST UINT16_T AOWI_RESET_D = (UINT16_T)(F_CPU / 30000L)

This is the delay counter for creating a reset pulse. It needs to generate a pulse about 500 microseconds in duration. $F_{CPU} / 30000L$ has been seen to work.

Definition at line 73 of file avr_1wire.h.

Referenced by avr_1wire::auto_timing(), avr_1wire::avr_1wire(), and avr_1wire::reset().

`CONST UINT16_T AOWI_RETRIES = 10000`

This define prevents this file from being included more than once in a *.cc file.

This is the number of retries we'll wait for an acknowledgement from a sensor.

Definition at line 51 of file avr_1wire.h.

AVR_DS182X.CPP FILE REFERENCE

```
#include "FreeRTOS.h"
#include "task.h"
#include "avr_ds182x.h"
```

FUNCTIONS

- `emstream & operator<< (emstream &serial, avr_ds182x &sensor)`
-

DETAILED DESCRIPTION

This file contains a class which interfaces an AVR processor to each of several types of Dallas Semiconductor One-Wire temperature sensors. It has been written to work on the DS18B20 and DS1822. The sensors are connected to a single digital I/O pin on the microcontroller. Many sensors can be connected to the same pin. Each sensor should be powered with Vcc and ground, not bus parasite powered.

Revisions:

- 12-18-2007 JRR Created file
- 12-30-2007 JRR Corrected spelling of Fahrenheit
- 03-20-2009 JRR Updated for newer boards
- 12-01-2012 JRR Changed to work in new FreeRTOS based ME405 environment, made "<<" operator work properly, made generic `_ds182x` class

License: This file copyright 2007-2012 by JR Ridgely. It is released under the Lesser GNU public license, version 2. It is intended for educational use only, but it may be used for any purpose permitted under the LGPL. The author has no control over the uses of this software and cannot be responsible for any consequences of such use.

Definition in file **avr_ds182x.cpp**.

FUNCTION DOCUMENTATION

EMSTREAM& OPERATOR<< (EMSTREAM & SERIAL, AVR_DS182X & SENSOR)

This overloaded operator allows a temperature reading to be printed on a serial device such as a regular serial port or radio module in text mode. This allows a display in the style of 'cout.' The temperature is printed in Fahrenheit to a precision of 0.1 degrees by default. Note that the sensor isn't so accurate, but we might as well not waste what accuracy we have through quantization error.

PARAMETERS:

<i>serial</i>	A reference to the serial-type object to which to print
<i>sensor</i>	A reference to the DS182X object to be displayed

Definition at line 287 of file `avr_ds182x.cpp`.

References `avr_ds182x::celsius()`.

AVR_DS182X.H FILE REFERENCE

```
#include "emstream.h"
#include "avr_1wire.h"
```

CLASSES

- class **avr_ds182x**

THIS CLASS IMPLEMENTS A DRIVER FOR AN AVR PROCESSOR TO A DS182X TYPE "ONE-WIRE" TEMPERATURE SENSOR. FUNCTIONS

- emstream & **operator**<< (emstream &, **avr_ds182x** &)

VARIABLES

- const uint16_t **DS182X_RETRIES** = 40000
This is the number of retries we will wait for a response from the 1-wire chip.
- const uint16_t **DS182X_ST_DEL** = 1000
This is the duration of the start of a delay loop to wait for a conversion.
- const uint8_t **DS1820_TYPE_ID** = 0x10
This is the chip type ID (lowest byte of ID number) for the DS1820.
- const uint8_t **DS1822_TYPE_ID** = 0x22
This is the chip type ID (lowest byte of ID number) for the DS1822.
- const uint8_t **DS182X_MAX_TOUT** = 3
This is the maximum number of retries if the temperature reading isn't reasonable.

DETAILED DESCRIPTION

This file contains a class which interfaces an AVR processor to each of several types of Dallas Semiconductor One-Wire temperature sensors. It has been written to work on the DS18B20 and DS182X. The sensors are connected to a single digital I/O pin on the microcontroller. Many sensors can be connected to the same pin. Each sensor should be powered with Vcc and ground, not bus parasite powered.

Revisions:

- 12-18-2007 JRR Created file
- 12-30-2007 JRR Corrected spelling of Fahrenheit
- 03-20-2009 JRR Updated for newer boards
- 12-01-2012 JRR Changed to work in new FreeRTOS based ME405 environment, made "<<" operator work properly, made generic _ds182x class

License: This file copyright 2007-2012 by JR Ridgely. It is released under the Lesser GNU public license, version 2. It is intended for educational use only, but it may be used for any purpose permitted under the LGPL. The author has no control over the uses of this software and cannot be responsible for any consequences of such use.

Definition in file **avr_ds182x.h**.

FUNCTION DOCUMENTATION

EMSTREAM& OPERATOR<< (EMSTREAM & SERIAL, AVR_DS182X & SENSOR)

This overloaded operator allows a temperature reading to be printed on a serial device such as a regular serial port or radio module in text mode. This allows a display in the style of 'cout.' The temperature is printed in Fahrenheit to a precision of 0.1 degrees by default. Note that the sensor isn't so accurate, but we might as well not waste what accuracy we have through quantization error.

PARAMETERS:

<i>serial</i>	A reference to the serial-type object to which to print
<i>sensor</i>	A reference to the DS182X object to be displayed

Definition at line 287 of file avr_ds182x.cpp.

References avr_ds182x::celsius().

CONVERTCOORD.CPP FILE REFERENCE

```
#include "ConvertCoord.h"
```

FUNCTIONS

- **int16_t azimuthTransform** (int16_t angle0, int16_t angle1)
This method converts from main to spherical coordinates.
 - **int16_t elevationTransform** (int16_t angle0, int16_t angle1)
This method converts from main to spherical coordinates.
 - **int16_t angle0Transform** (int16_t azimuth, int16_t elevation)
This method converts from spherical to main coordinates.
 - **int16_t angle1Transform** (int16_t azimuth, int16_t elevation)
This method converts from spherical to main coordinates.
-

DETAILED DESCRIPTION

This file contains the methods necessary convert between coordinate systems.

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **ConvertCoord.cpp**.

FUNCTION DOCUMENTATION

INT16_T ANGLE0TRANSFORM (INT16_T AZIMUTH, INT16_T ELEVATION)

This method converts from spherical to main coordinates.

This methods performs a coordinate transformation using the given angles.

PARAMETERS:

<i>inputPosition1</i>	First position used in the coordinate transformation.
<i>inputPosition2</i>	Second position used in the coordinate transformation.

Definition at line 66 of file ConvertCoord.cpp.

Referenced by UiController::setAngleAzimuth(), and UiController::setAngleElevation().

INT16_T ANGLE1TRANSFORM (INT16_T AZIMUTH, INT16_T ELEVATION)

This method converts from spherical to main coordinates.

This methods performs a coordinate transformation using the given angles.

PARAMETERS:

<i>inputPosition1</i>	First position used in the coordinate transformation.
<i>inputPosition2</i>	Second position used in the coordinate transformation.

Definition at line 83 of file ConvertCoord.cpp.

Referenced by UiController::setAngleAzimuth(), and UiController::setAngleElevation().

INT16_T AZIMUTHTRANSFORM (INT16_T ANGLE0, INT16_T ANGLE1)

This method converts from main to spherical coordinates.

This file contains the methods necessary to convert between coordinate systems.

This methods performs a coordinate transformation using the given angles.

PARAMETERS:

<i>inputPosition1</i>	First position used in the coordinate transformation.
<i>inputPosition2</i>	Second position used in the coordinate transformation.

Definition at line 32 of file ConvertCoord.cpp.

Referenced by UiController::f2Update(), UiController::setAngle0(), and UiController::setAngle1().

INT16_T ELEVATIONTRANSFORM (INT16_T ANGLE0, INT16_T ANGLE1)

This method converts from main to spherical coordinates.

This methods performs a coordinate transformation using the given angles.

PARAMETERS:

<i>inputPosition1</i>	First position used in the coordinate transformation.
<i>inputPosition2</i>	Second position used in the coordinate transformation.

Definition at line 49 of file ConvertCoord.cpp.

Referenced by UiController::f2Update(), UiController::setAngle0(), and UiController::setAngle1().

CONVERTCOORD.H FILE REFERENCE

```
#include "emstream.h"
#include "frt_text_queue.h"
#include "frt_queue.h"
#include "frt_shared_data.h"
#include "shares.h"
#include "queue.h"
#include "semphr.h"
#include "math.h"
```

MACROS

- `#define PI 3.14159265`

FUNCTIONS

- `int16_t azimuthTransform (int16_t angle0, int16_t angle1)`
This file contains the methods necessary to convert between coordinate systems.
 - `int16_t elevationTransform (int16_t angle0, int16_t angle1)`
This method converts from main to spherical coordinates.
 - `int16_t angle0Transform (int16_t azimuth, int16_t elevation)`
This method converts from spherical to main coordinates.
 - `int16_t angle1Transform (int16_t azimuth, int16_t elevation)`
This method converts from spherical to main coordinates.
-

DETAILED DESCRIPTION

This file contains the methods necessary convert between coordinate systems.

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **ConvertCoord.h**.

FUNCTION DOCUMENTATION

INT16_T ANGLE0TRANSFORM (INT16_T AZIMUTH, INT16_T ELEVATION)

This method converts from spherical to main coordinates.

This methods performs a coordinate transformation using the given angles.

PARAMETERS:

<i>inputPosition1</i>	First position used in the coordinate transformation.
<i>inputPosition2</i>	Second position used in the coordinate transformation.

Definition at line 66 of file ConvertCoord.cpp.

Referenced by UiController::setAngleAzimuth(), and UiController::setAngleElevation().

INT16_T ANGLE1TRANSFORM (INT16_T AZIMUTH, INT16_T ELEVATION)

This method converts from spherical to main coordinates.

This methods performs a coordinate transformation using the given angles.

PARAMETERS:

<i>inputPosition1</i>	First position used in the coordinate transformation.
<i>inputPosition2</i>	Second position used in the coordinate transformation.

Definition at line 83 of file ConvertCoord.cpp.

Referenced by UiController::setAngleAzimuth(), and UiController::setAngleElevation().

INT16_T AZIMUTHTRANSFORM (INT16_T ANGLE0, INT16_T ANGLE1)

This file contains the methods necessary to convert between coordinate systems.

Thransformation equations are applied to the input positions to convert between coordinate systems.

This file contains the methods necessary to convert between coordinate systems.

This methods performs a coordinate transformation using the given angles.

PARAMETERS:

<i>inputPosition1</i>	First position used in the coordinate transformation.
<i>inputPosition2</i>	Second position used in the coordinate transformation.

Definition at line 32 of file ConvertCoord.cpp.

Referenced by UiController::f2Update(), UiController::setAngle0(), and UiController::setAngle1().

INT16_T ELEVATIONTRANSFORM (INT16_T ANGLE0, INT16_T ANGLE1)

This method converts from main to spherical coordinates.

This methods performs a coordinate transformation using the given angles.

PARAMETERS:

<i>inputPosition1</i>	First position used in the coordinate transformation.
<i>inputPosition2</i>	Second position used in the coordinate transformation.

Definition at line 49 of file ConvertCoord.cpp.

Referenced by UiController::f2Update(), UiController::setAngle0(), and UiController::setAngle1().

CURRENTSENSORDRIVER.CPP FILE REFERENCE

```
#include "CurrentSensorDriver.h"
```

DETAILED DESCRIPTION

This file contains the methods necessary to initialize and read a current sensor. Supports for HMS 5..20-P current transducer.

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **CurrentSensorDriver.cpp**.

CURRENTSENSORDRIVER.H FILE REFERENCE

```
#include "emstream.h"
#include "frt_text_queue.h"
#include "frt_queue.h"
#include "frt_shared_data.h"
#include "shares.h"
#include "queue.h"
#include "semphr.h"
#include "adc.h"
```

CLASSES

- class **CurrentSensorDriver**

This file contains the methods necessary to control a current sensor.

DETAILED DESCRIPTION

This file contains the methods necessary to initialize and read a current sensor. Supports for HMS 5..20-P current transducer.

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **CurrentSensorDriver.h**.

FANDRIVER.CPP FILE REFERENCE

```
#include "FanDriver.h"
```

DETAILED DESCRIPTION

This file contains the methods necessary to initialize and control a fan relay.

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **FanDriver.cpp**.

FANDRIVER.H FILE REFERENCE

```
#include "emstream.h"
#include "frt_text_queue.h"
#include "frt_queue.h"
#include "frt_shared_data.h"
#include "shares.h"
#include "queue.h"
#include "semphr.h"
```

CLASSES

- class **FanDriver**

This file contains the methods necessary to control a fan relay.

DETAILED DESCRIPTION

This file contains the methods necessary to initialize and control a fan relay.

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **FanDriver.h**.

IRRADIANCEDRIVER.CPP FILE REFERENCE

```
#include "IrradianceDriver.h"
```

DETAILED DESCRIPTION

This file contains the methods necessary to initialize and read an irradiance sensor. This driver is compatible with SP-212 and SP-215 irradiance sensors.

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **IrradianceDriver.cpp**.

IRRADIANCEDRIVER.H FILE REFERENCE

```
#include "emstream.h"
#include "frt_text_queue.h"
#include "frt_queue.h"
#include "frt_shared_data.h"
#include "shares.h"
#include "queue.h"
#include "semphr.h"
#include "adc.h"
```

CLASSES

- class **IrradianceDriver**

This file contains the methods necessary to control an irradiance sensor.

DETAILED DESCRIPTION

This file contains the methods necessary to initialize and read an irradiance sensor. This driver is compatible with SP-212 and SP-215 irradiance sensors.

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **IrradianceDriver.h**.

KEYPADCONTROLLER.CPP FILE REFERENCE

```
#include "KeypadController.h"
```

DETAILED DESCRIPTION

This file contains task necessary to update input from a keypad.

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **KeypadController.cpp**.

KEYPADCONTROLLER.H FILE REFERENCE

```
#include "emstream.h"
#include "FreeRTOS.h"
#include "task.h"
#include "frt_task.h"
#include "frt_text_queue.h"
#include "frt_queue.h"
#include "frt_shared_data.h"
#include "shares.h"
#include "queue.h"
#include "semphr.h"
#include "KeypadDriver.h"
```

CLASSES

- class **KeypadController**

This task controls the power sensors and irradiance drivers.

DETAILED DESCRIPTION

This file contains task necessary to update input from a keypad.

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **KeypadController.h**.

KEYPADDRIVER.CPP FILE REFERENCE

```
#include "KeypadDriver.h"
```

DETAILED DESCRIPTION

This file contains the methods necessary to initialize and control a keypad.

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **KeypadDriver.cpp**.

KEYPADDRIVER.H FILE REFERENCE

```
#include "emstream.h"
#include "frt_text_queue.h"
#include "frt_queue.h"
#include "frt_shared_data.h"
#include "shares.h"
#include "queue.h"
#include "semphr.h"
#include "rs232int.h"
#include <util/delay.h>
```

CLASSES

- class **KeypadDriver**

VARIABLES

- const uint8_t **keypadRows** = 4
This file contains the methods necessary to initialize and control a keypad.
 - const uint8_t **keypadColumns** = 4
Number of columns on keypad.
 - const char **keypadButtons** [**keypadRows**][**keypadColumns**]
Mapping of keypad buttons.
-

DETAILED DESCRIPTION

This file contains the methods necessary to initialize and control a keypad.

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **KeypadDriver.h**.

VARIABLE DOCUMENTATION

CONST CHAR KEYPADBUTTONS[KEYPADROWS][KEYPADCOLUMNS]

```
Initial value:= {
    { '1', '2', '3', 'A' },
    { '4', '5', '6', 'B' },
    { '7', '8', '9', 'C' },
    { 'X', '0', 'Y', 'D' }
}
```

Mapping of keypad buttons.

Definition at line 50 of file KeypadDriver.h.

Referenced by KeypadDriver::checkInput().

```
CONST UINT8_T KEYPADROWS = 4
```

This file contains the methods necessary to initialize and control a keypad.

The keypad driver is used to initialize the keypad bus and contains the method to check whether input has been recieved from a user. Number of rows on keypad

Definition at line 45 of file KeypadDriver.h.

LCDDRIVER.CPP FILE REFERENCE

```
#include "LcdDriver.h"
```

DETAILED DESCRIPTION

This file contains the methods necessary to initialize and control an LCD screen using SPI communication protocol.

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **LcdDriver.cpp**.

LCDDRIVER.H FILE REFERENCE

```
#include "emstream.h"
#include "frt_text_queue.h"
#include "frt_queue.h"
#include "frt_shared_data.h"
#include "shares.h"
#include "queue.h"
#include "semphr.h"
#include "SpiMaster.h"
```

CLASSES

- class **LcdDriver**

MACROS

- #define **PREFIX** 0xFE
This driver contains the methods necessary to initialize and control an LCD screen.
 - #define **DISPLAY_ON** 0x41
 - #define **DISPLAY_OFF** 0x42
 - #define **SET_CURSOR** 0x45
 - #define **CURSOR_HOME** 0x46
 - #define **UNDERLINE_ON** 0x47
 - #define **UNDERLINE_OFF** 0x48
 - #define **MOVE_LEFT** 0x49
 - #define **MOVE_RIGHT** 0x4A
 - #define **BLINK_ON** 0x4B
 - #define **BLINK_OFF** 0x4C
 - #define **BACKSPACE** 0x4E
 - #define **CLEAR** 0x51
 - #define **SET_CONTRAST** 0x52
 - #define **SET_BACKLIGHT** 0x53
 - #define **LOAD_CSTM_CHAR** 0x54
 - #define **MOVE_DISP_LEFT** 0x55
 - #define **MOVE_DISP_RIGHT** 0x56
 - #define **CHANGE_BAUD** 0x61
 - #define **CHANGE_I2C_ADDR** 0x62
 - #define **DISP_FIRMWARE** 0x70
 - #define **DISP_BAUD** 0x71
 - #define **DISP_I2C_ADDR** 0x72
-

DETAILED DESCRIPTION

This file contains the methods necessary to initialize and control an LCD screen using SPI communication protocol.

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **LcdDriver.h**.

MACRO DEFINITION DOCUMENTATION

#DEFINE PREFIX OXFE

This driver contains the methods necessary to initialize and control an LCD screen.

The LCD driver uses **SpiMaster.h** to define out it communicates with hardware. The maximum clock rate for communication is 100KHz.

Definition at line 44 of file LcdDriver.h.

Referenced by LcdDriver::backspace(), LcdDriver::blinkOff(), LcdDriver::blinkOn(), LcdDriver::clearScreen(), LcdDriver::displayOff(), LcdDriver::displayOn(), LcdDriver::moveCursor(), LcdDriver::moveLeft(), LcdDriver::moveRight(), and LcdDriver::printChar().

MOTORDRIVER.CPP FILE REFERENCE

```
#include <stdlib.h>
#include <avr/io.h>
#include "rs232int.h"
#include "MotorDriver.h"
```

FUNCTIONS

- `emstream & operator<<` (`emstream &serpt`, **MotorDriver** &motorDriver)
This overloaded operator "prints the motor driver."
-

DETAILED DESCRIPTION

This file contains a very simple motor driver. The driver is hopefully thread safe in FreeRTOS due to the use of a mutex to prevent its use by multiple tasks at the same time. There is no protection from priority inversion, however, except for the priority elevation in the mutex.

Revisions:

- 04-18-2013 Motor driver setup from ADC structure

License: This file is copyright 2012 by Matthew Clause and released under the Lesser GNU Public License, version 2. It intended for educational use only, but its use is not limited thereto.

Definition in file **MotorDriver.cpp**.

FUNCTION DOCUMENTATION

EMSTREAM& OPERATOR<< (EMSTREAM & SERPT, MOTORDRIVER & MOTORDRIVER)

This overloaded operator "prints the motor driver."

The << operator returns the value of current motor **power** .

PARAMETERS:

<i>serpt</i>	Reference to a serial port to which the printout will be printed
<i>motorDriver</i>	Reference to the motor driver which is being printed

RETURNS:

A reference to the same serial device on which we write information. This is used to string together things to write with "<<" operators

Definition at line 173 of file MotorDriver.cpp.

References MotorDriver::get_power().

MOTORDRIVER.H FILE REFERENCE

```
#include "emstream.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"
```

CLASSES

- class **MotorDriver**

THIS CLASS SHOULD RUN A MOTOR DRIVER ON AN AVR PROCESSOR. FUNCTIONS

- emstream & **operator**<< (emstream &, **MotorDriver** &)
This overloaded operator "prints the motor driver."
-

DETAILED DESCRIPTION

This file contains a very simple motor driver. The driver is hopefully thread safe in FreeRTOS due to the use of a mutex to prevent its use by multiple tasks at the same time. There is no protection from priority inversion, however, except for the priority elevation in the mutex.

Revisions:

- 04-18-2013 Motor driver setup from ADC structure

License: This file is copyright 2012 by Matthew Clause and released under the Lesser GNU Public License, version 2. It intended for educational use only, but its use is not limited thereto.

Definition in file **MotorDriver.h**.

FUNCTION DOCUMENTATION

*EMSTREAM& OPERATOR<< (EMSTREAM & *serpt*, MOTORDRIVER & *MOTORDRIVER*)*

This overloaded operator "prints the motor driver."

The << operator returns the value of current motor **power** .

PARAMETERS:

<i>serpt</i>	Reference to a serial port to which the printout will be printed
<i>motorDriver</i>	Reference to the motor driver which is being printed

RETURNS:

A reference to the same serial device on which we write information. This is used to string together things to write with "<<" operators

Definition at line 173 of file MotorDriver.cpp.

References MotorDriver::get_power().

POWERCONTROLLER.CPP FILE REFERENCE

```
#include "PowerController.h"  
#include <util/delay.h>
```

DETAILED DESCRIPTION

This file contains task necessary to temperature sensors and fans.

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **PowerController.cpp**.

POWERCONTROLLER.H FILE REFERENCE

```
#include "emstream.h"
#include "FreeRTOS.h"
#include "task.h"
#include "frt_task.h"
#include "frt_text_queue.h"
#include "frt_queue.h"
#include "frt_shared_data.h"
#include "shares.h"
#include "queue.h"
#include "semphr.h"
#include "IrradianceDriver.h"
#include "VoltageSensorDriver.h"
#include "CurrentSensorDriver.h"
```

CLASSES

- class **PowerController**

This task controls the power sensors and irradiance drivers.

DETAILED DESCRIPTION

This file contains task necessary read voltage, current, and irradiance.

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **PowerController.h**.

PVTRAINERMAIN.CPP FILE REFERENCE

```
#include <stdlib.h>
#include <avr/io.h>
#include <avr/wdt.h>
#include <string.h>
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "croutine.h"
#include "emstream.h"
#include "rs232int.h"
#include "time_stamp.h"
#include "frt_task.h"
#include "frt_text_queue.h"
#include "frt_queue.h"
#include "frt_shared_data.h"
#include "shares.h"
#include "task_user.h"
#include "KeypadDriver.h"
#include "KeypadController.h"
#include "SpiMaster.h"
#include "LcdDriver.h"
#include "UiController.h"
#include "AbsoluteEncoderDriver.h"
#include "MotorDriver.h"
#include "ActuatorController.h"
#include "avr_1wire.h"
#include "avr_ds182x.h"
#include "FanDriver.h"
#include "TempController.h"
#include "adc.h"
#include "IrradianceDriver.h"
#include "VoltageSensorDriver.h"
#include "CurrentSensorDriver.h"
#include "PowerController.h"
```

FUNCTIONS

- int **main** (void)

VARIABLES

- frt_text_queue * **print_ser_queue**
This queue allows tasks to send characters to the user interface task for display.
- shared_data< int16_t > **sharedAngle1**
Shared shaft angle 1.
- shared_data< int16_t > **sharedAngle2**
Shared shaft angle 2.
- shared_data< int16_t > **sharedEncoder1**

Shared encoder angle 1.

- `shared_data< int16_t > sharedEncoder2`

Shared encoder angle 2.

- `shared_data< bool > sharedPowerState`

Shared power state.

- `shared_data< bool > sharedFanState`

Shared fan state.

- `shared_data< int16_t > sharedTemp1`

Shared panel temperature 1.

- `shared_data< int16_t > sharedTemp2`

Shared panel temperature 2.

- `shared_data< int16_t > sharedTemp3`

Shared panel temperature 3.

- `shared_data< int16_t > sharedTemp4`

Shared panel temperature 4.

- `shared_data< int16_t > sharedTempAverage`

Shared panel average temperature.

- `shared_data< int16_t > sharedIrradiance`

Shared irradiance.

- `shared_data< int16_t > sharedVoltage`

Shared panel voltage.

- `shared_data< int16_t > sharedCurrent`

Shared panel current.

- `shared_data< char > sharedInput`

Shared keypad input.

DETAILED DESCRIPTION

This file contains the **main()** code for a program which runs the ME405 board for the PV Solar Trainer Senior Project.

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **PvTrainerMain.cpp**.

FUNCTION DOCUMENTATION

INT MAIN (VOID)

The main function sets up the RTOS. Some test tasks are created. Then the scheduler is started up; the scheduler runs until power is turned off or there's a reset.

RETURNS:

This is a real-time microcontroller program which doesn't return. Ever.

Definition at line 117 of file PvTrainerMain.cpp.

References `avr_ds182x::configure()`, `avr_ds182x::find_by_ID()`, `avr_1wire::get_ID()`, `print_ser_queue`, `avr_1wire::search()`, `sharedAngle1`, `sharedAngle2`, `sharedCurrent`, `sharedEncoder1`, `sharedEncoder2`, `sharedFanState`, `sharedInput`, `sharedIrradiance`, `sharedPowerState`, `sharedTemp1`, `sharedTemp2`, `sharedTemp3`, `sharedTemp4`, `sharedTempAverage`, `sharedVoltage`, and `avr_1wire::show_devices()`.

VARIABLE DOCUMENTATION

FRT_TEXT_QUEUE* PRINT_SER_QUEUE

This queue allows tasks to send characters to the user interface task for display.

This is a print queue, descended from `emstream` so that things can be printed into the queue using the "<<" operator and they'll come out the other end as a stream of characters. It's used by tasks that send things to the user interface task to be printed.

Definition at line 77 of file PvTrainerMain.cpp.

Referenced by `main()`, and `task_user::run()`.

SHARED_DATA<INT16_T> SHAREDTEMP1

Shared panel temperature 1.

Shared panel temperature.

Definition at line 92 of file PvTrainerMain.cpp.

Referenced by `UiController::f3Update()`, and `main()`.

SHARED_DATA<INT16_T> SHAREDTEMP2

Shared panel temperature 2.

Shared panel temperature.

Definition at line 94 of file PvTrainerMain.cpp.

Referenced by `UiController::f3Update()`, and `main()`.

SHARED_DATA<INT16_T> SHAREDTEMP3

Shared panel temperature 3.

Shared panel temperature.

Definition at line 96 of file PvTrainerMain.cpp.

Referenced by `UiController::f3Update()`, and `main()`.

SHARED_DATA<INT16_T> SHAREDTEMP4

Shared panel temperature 4.

Shared panel temperature.

Definition at line 98 of file PvTrainerMain.cpp.

Referenced by `UiController::f3Update()`, and `main()`.

SHARED_DATA<INT16_T> SHAREDTEMPAVERAGE

Shared panel average temperature.

Shared panel temperature.

Definition at line 100 of file PvTrainerMain.cpp.

Referenced by UiController::f3Update(), and main().

SHARES.H FILE REFERENCE

VARIABLES

- `frt_text_queue * print_ser_queue`
This queue allows tasks to send characters to the user interface task for display.
- `shared_data< int16_t > sharedAngle1`
Shared shaft angle 1.
- `shared_data< int16_t > sharedAngle2`
Shared shaft angle 2.
- `shared_data< int16_t > sharedEncoder1`
Shared encoder angle 1.
- `shared_data< int16_t > sharedEncoder2`
Shared encoder angle 2.
- `shared_data< bool > sharedPowerState`
Shared power state.
- `shared_data< bool > sharedFanState`
Shared fan state.
- `shared_data< int16_t > sharedTemp1`
Shared panel temperature.
- `shared_data< int16_t > sharedTemp2`
Shared panel temperature.
- `shared_data< int16_t > sharedTemp3`
Shared panel temperature.
- `shared_data< int16_t > sharedTemp4`
Shared panel temperature.
- `shared_data< int16_t > sharedTempAverage`
Shared panel temperature.
- `shared_data< int16_t > sharedIrradiance`
Shared irradiance.
- `shared_data< int16_t > sharedVoltage`
Shared panel voltage.
- `shared_data< int16_t > sharedCurrent`
Shared panel current.
- `shared_data< char > sharedInput`
Shared keypad input.

DETAILED DESCRIPTION

This file contains extern declarations for queues and other inter-task data communication objects used the PV Solar Trainer program.

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **shares.h**.

VARIABLE DOCUMENTATION

FRT_TEXT_QUEUE* PRINT_SER_QUEUE

This queue allows tasks to send characters to the user interface task for display.

This is a print queue, descended from `emstream` so that things can be printed into the queue using the "<<" operator and they'll come out the other end as a stream of characters. It's used by tasks that send things to the user interface task to be printed.

Definition at line 77 of file `PvTrainerMain.cpp`.

Referenced by `main()`, and `task_user::run()`.

SPIMASTER.CPP FILE REFERENCE

```
#include "SpiMaster.h"
```

FUNCTIONS

- void **delay_ms** (uint16_t count)
This methods performs a variable delay in ms.
 - void **delay_us** (uint16_t count)
This methods performs a variable delay in us.
-

DETAILED DESCRIPTION

This file contains the methods to communicate using SPI protocol

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **SpiMaster.cpp**.

FUNCTION DOCUMENTATION

VOID DELAY_MS (UINT16_T COUNT)

This methods performs a variable delay in ms.

This method uses `util/delay.h` to perform a variable delay on demand.

PARAMETERS:

<i>count</i>	Number of ms to delay
--------------	-----------------------

Definition at line 205 of file SpiMaster.cpp.

Referenced by UiController::initDisplay().

VOID DELAY_US (UINT16_T COUNT)

This methods performs a variable delay in us.

This method uses `util/delay.h` to perform a variable delay on demand.

PARAMETERS:

<i>count</i>	Number of us to delay
--------------	-----------------------

Definition at line 216 of file SpiMaster.cpp.

Referenced by LcdDriver::backspace(), LcdDriver::blinkOff(), LcdDriver::blinkOn(), LcdDriver::clearScreen(), LcdDriver::displayOff(), LcdDriver::displayOn(), LcdDriver::moveCursor(), LcdDriver::moveLeft(), LcdDriver::moveRight(), LcdDriver::printChar(), SpiMaster::recieve(), and SpiMaster::send().

SPIMASTER.H FILE REFERENCE

```
#include "emstream.h"
#include "frt_text_queue.h"
#include "frt_queue.h"
#include "frt_shared_data.h"
#include "shares.h"
#include "queue.h"
#include "semphr.h"
#include <avr/io.h>
#include <util/delay.h>
```

CLASSES

- class **SpiMaster**

THIS CONTAINS THE METHODS NECESSARY COMMUNICATE WITH SPI SUPPORTED HARDWARE. FUNCTIONS

- void **delay_ms** (uint16_t count)
This methods performs a variable delay in ms.
- void **delay_us** (uint16_t count)
This methods performs a variable delay in us.

DETAILED DESCRIPTION

This file contains the methods to communicate using SPI protocol

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **SpiMaster.h**.

FUNCTION DOCUMENTATION

VOID DELAY_MS (UINT16_T COUNT)

This methods performs a variable delay in ms.

This method uses `util/delay.h` to perform a variable delay on demand.

PARAMETERS:

<i>count</i>	Number of ms to delay
--------------	-----------------------

Definition at line 205 of file SpiMaster.cpp.

Referenced by UiController::initDisplay().

`VOID DELAY_US (UINT16_T COUNT)`

This methods performs a variable delay in us.

This method uses `util/delay.h` to perform a variable delay on demand.

PARAMETERS:

<i>count</i>	Number of us to delay
--------------	-----------------------

Definition at line 216 of file SpiMaster.cpp.

Referenced by LcdDriver::backspace(), LcdDriver::blinkOff(), LcdDriver::blinkOn(), LcdDriver::clearScreen(), LcdDriver::displayOff(), LcdDriver::displayOn(), LcdDriver::moveCursor(), LcdDriver::moveLeft(), LcdDriver::moveRight(), LcdDriver::printChar(), SpiMaster::recieve(), and SpiMaster::send().

TASK_USER.H FILE REFERENCE

```
#include <stdlib.h>
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "rs232int.h"
#include "adc.h"
#include "time_stamp.h"
#include "frt_task.h"
#include "frt_queue.h"
#include "frt_text_queue.h"
#include "frt_shared_data.h"
#include "shares.h"
```

CLASSES

- class **task_user**

MACROS

- #define **PROGRAM_VERSION** PMS ("PV Trainer Program V 0.1")
This macro defines a string that identifies the name and version of this program.

DETAILED DESCRIPTION

This file contains header stuff for a user interface task for a ME507/FreeRTOS test suite.

Revisions:

- 09-30-2012 JRR Original file was a one-file demonstration with two tasks
- 10-05-2012 JRR Split into multiple files, one for each task
- 10-25-2012 JRR Changed to a more fully C++ version with class **task_user**
- 11-04-2012 JRR Modified from the data acquisition example to the test suite

License: This file is copyright 2012 by JR Ridgely and released under the Lesser GNU Public License, version 2. It intended for educational use only, but its use is not limited thereto.

Definition in file **task_user.h**.

TEMPCONTROLLER.CPP FILE REFERENCE

```
#include "TempController.h"
```

DETAILED DESCRIPTION

This file contains task necessary to temperature sensors and fans.

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **TempController.cpp**.

TEMPCONTROLLER.H FILE REFERENCE

```
#include "emstream.h"
#include "FreeRTOS.h"
#include "task.h"
#include "frt_task.h"
#include "frt_text_queue.h"
#include "frt_queue.h"
#include "frt_shared_data.h"
#include "shares.h"
#include "queue.h"
#include "semphr.h"
#include "avr_ds182x.h"
#include "FanDriver.h"
#include <util/delay.h>
```

CLASSES

- class **TempController**

This task controls the sensor and fan driver.

DETAILED DESCRIPTION

This file contains task necessary to DS18B20 temperature sensors and fans.

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **TempController.h**.

UICONTROLLER.CPP FILE REFERENCE

```
#include "UiController.h"
```

DETAILED DESCRIPTION

This file contains task necessary to control an LCD and keypad.

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **UiController.cpp**.

UICONTROLLER.H FILE REFERENCE

```
#include "emstream.h"
#include "FreeRTOS.h"
#include "task.h"
#include "frt_task.h"
#include "frt_text_queue.h"
#include "frt_queue.h"
#include "frt_shared_data.h"
#include "shares.h"
#include "queue.h"
#include "semphr.h"
#include "LcdDriver.h"
#include "ConvertCoord.h"
```

CLASSES

- class **UiController**

This task controls the LCD display.

DETAILED DESCRIPTION

This file contains task necessary to control an LCD.

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **UiController.h**.

VOLTAGESENSORDRIVER.CPP FILE REFERENCE

```
#include "VoltageSensorDriver.h"
```

DETAILED DESCRIPTION

This file contains the methods necessary to initialize and read panel voltage.

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **VoltageSensorDriver.cpp**.

VOLTAGESENSORDRIVER.H FILE REFERENCE

```
#include "emstream.h"
#include "frt_text_queue.h"
#include "frt_queue.h"
#include "frt_shared_data.h"
#include "shares.h"
#include "queue.h"
#include "semphr.h"
#include "adc.h"
```

CLASSES

- class **VoltageSensorDriver**

This file contains the methods necessary to read panel voltage.

DETAILED DESCRIPTION

This file contains the methods necessary to initialize and read panel voltage.

Revisions:

- 04-19-2014 MEC Program was created

License: This file is copyright 2014 by team SunStream and released under the GNU Lesser Public License, version 3. It intended for educational use only, but its use is not limited thereto.

Definition in file **VoltageSensorDriver.h**.

INDEX

- AbsoluteEncoderDriver, 5
 - AbsoluteEncoderDriver, 5
 - read, 6
- AbsoluteEncoderDriver.cpp, 58
- AbsoluteEncoderDriver.h, 59
- ActuatorController, 7
 - ActuatorController, 8
 - run, 8
- ActuatorController.h, 60
- adc, 10
 - adc, 10
 - read_once, 11
 - read_oversampled, 11
- adc.cpp, 61
 - operator<<, 61
- adc.h, 62
 - operator<<, 62
- angle0Transform
 - ConvertCoord.cpp, 71
 - ConvertCoord.h, 73
- angle1Transform
 - ConvertCoord.cpp, 71
 - ConvertCoord.h, 74
- AOWI_15us_D
 - avr_1wire.h, 66
- AOWI_1us_D
 - avr_1wire.h, 66
- AOWI_45us_D
 - avr_1wire.h, 66
- AOWI_90us_D
 - avr_1wire.h, 66
- AOWI_PRES_D
 - avr_1wire.h, 66
- AOWI_PRES_END
 - avr_1wire.h, 66
- AOWI_RESET_D
 - avr_1wire.h, 66
- AOWI_RETRIES
 - avr_1wire.h, 67
- auto_timing
 - avr_1wire, 13
- avr_1wire, 12
 - auto_timing, 13
 - avr_1wire, 13
 - find_by_ID, 13
 - find_by_type, 14
 - get_ID, 14
 - get_ID_bit, 14
 - match_ROM, 14
 - read_bit, 15
 - read_byte, 15
 - read_ID, 15
 - reset, 15
 - reset_pulse_dur, 17
 - search, 15
 - set_ID_bit, 16
 - show_devices, 16
 - write_0, 16
 - write_1, 16
 - write_byte, 16
 - write_byte_rev, 17
- avr_1wire.cpp, 64
- avr_1wire.h, 65
 - AOWI_15us_D, 66
 - AOWI_1us_D, 66
 - AOWI_45us_D, 66
 - AOWI_90us_D, 66
 - AOWI_PRES_D, 66
 - AOWI_PRES_END, 66
 - AOWI_RESET_D, 66
 - AOWI_RETRIES, 67
- avr_ds182x, 18
 - avr_ds182x, 19
 - celsius, 19
 - configure, 19
 - fahrenheit, 20
 - find_by_ID, 20
 - find_by_type, 20
 - temperature, 20
- avr_ds182x.cpp, 68
 - operator<<, 68
- avr_ds182x.h, 69
 - operator<<, 70
- azimuthTransform
 - ConvertCoord.cpp, 72
 - ConvertCoord.h, 74
- backspace
 - LcdDriver, 33
- blinkOff
 - LcdDriver, 33
- blinkOn
 - LcdDriver, 33
- brake
 - MotorDriver, 37
- celsius
 - avr_ds182x, 19
- checkAngle0
 - UiController, 51
- checkAngle1
 - UiController, 51

checkAngleAzimuth
 UiController, 51
 checkAngleElevation
 UiController, 52
 checkInput
 KeypadDriver, 30
 clearScreen
 LcdDriver, 33
 configure
 avr_ds182x, 19
 ConvertCoord.cpp, 71
 angle0Transform, 71
 angle1Transform, 71
 azimuthTransform, 72
 elevationTransform, 72
 ConvertCoord.h, 73
 angle0Transform, 73
 angle1Transform, 74
 azimuthTransform, 74
 elevationTransform, 74
 CurrentSensorDriver, 22
 CurrentSensorDriver, 22
 read, 23
 CurrentSensorDriver.cpp, 75
 CurrentSensorDriver.h, 76
 delay_ms
 SpiMaster.cpp, 99
 SpiMaster.h, 100
 delay_us
 SpiMaster.cpp, 99
 SpiMaster.h, 100
 displayOff
 LcdDriver, 34
 displayOn
 LcdDriver, 34
 elevationTransform
 ConvertCoord.cpp, 72
 ConvertCoord.h, 74
 f1Display
 UiController, 52
 f1Update
 UiController, 52
 f2Display
 UiController, 52
 f2Update
 UiController, 52
 f3Display
 UiController, 53
 f3Update
 UiController, 53
 f4Display
 UiController, 53
 f4Input
 UiController, 53
 f4Update
 UiController, 53
 fahrenheit
 avr_ds182x, 20
 FanDriver, 24
 FanDriver, 24
 off, 25
 on, 25
 toggle, 25
 FanDriver.cpp, 77
 FanDriver.h, 78
 find_by_ID
 avr_1wire, 13
 avr_ds182x, 20
 find_by_type
 avr_1wire, 14
 avr_ds182x, 20
 get_ID
 avr_1wire, 14
 get_ID_bit
 avr_1wire, 14
 initDisplay
 UiController, 54
 IrradianceDriver, 26
 IrradianceDriver, 26
 read, 27
 IrradianceDriver.cpp, 79
 IrradianceDriver.h, 80
 keypadButtons
 KeypadDriver.h, 84
 KeypadController, 28
 KeypadController, 28
 KeypadController.cpp, 81
 KeypadController.h, 82
 KeypadDriver, 30
 checkInput, 30
 KeypadDriver, 30
 KeypadDriver.cpp, 83
 KeypadDriver.h, 84
 keypadButtons, 84
 keypadRows, 84
 keypadRows
 KeypadDriver.h, 84
 LcdDriver, 32
 backspace, 33
 blinkOff, 33
 blinkOn, 33
 clearScreen, 33
 displayOff, 34
 displayOn, 34
 LcdDriver, 33
 moveCursor, 34
 moveLeft, 34
 moveRight, 34
 printChar, 35
 printString, 35
 LcdDriver.cpp, 86
 LcdDriver.h, 87

PREFIX, 88
 main
 PvTrainerMain.cpp, 94
 match_ROM
 avr_1wire, 14
 MotorDriver, 36
 brake, 37
 MotorDriver, 37
 set_power, 37
 MotorDriver.cpp, 89
 operator<<, 89
 MotorDriver.h, 90
 operator<<, 90
 moveCursor
 LcdDriver, 34
 moveLeft
 LcdDriver, 34
 moveRight
 LcdDriver, 34
 off
 FanDriver, 25
 on
 FanDriver, 25
 operator<<
 adc.cpp, 61
 adc.h, 62
 avr_ds182x.cpp, 68
 avr_ds182x.h, 70
 MotorDriver.cpp, 89
 MotorDriver.h, 90
 PowerController, 39
 PowerController, 39
 PowerController.cpp, 91
 PowerController.h, 92
 PREFIX
 LcdDriver.h, 88
 print_help_message
 task_user, 44
 print_ser_queue
 PvTrainerMain.cpp, 95
 shares.h, 98
 printChar
 LcdDriver, 35
 printString
 LcdDriver, 35
 PvTrainerMain.cpp, 93
 main, 94
 print_ser_queue, 95
 sharedTemp1, 95
 sharedTemp2, 95
 sharedTemp3, 95
 sharedTemp4, 95
 sharedTempAverage, 95
 read
 AbsoluteEncoderDriver, 6
 CurrentSensorDriver, 23
 IrradianceDriver, 27
 VoltageSensorDriver, 57
 read_bit
 avr_1wire, 15
 read_byte
 avr_1wire, 15
 read_ID
 avr_1wire, 15
 read_once
 adc, 11
 read_oversampled
 adc, 11
 recieve
 SpiMaster, 42
 reset
 avr_1wire, 15
 reset_pulse_dur
 avr_1wire, 17
 run
 ActuatorController, 8
 task_user, 45
 search
 avr_1wire, 15
 send
 SpiMaster, 42
 set_ID_bit
 avr_1wire, 16
 set_power
 MotorDriver, 37
 setAngle0
 UiController, 54
 setAngle1
 UiController, 54
 setAngleAzimuth
 UiController, 54
 setAngleElevation
 UiController, 54
 sharedTemp1
 PvTrainerMain.cpp, 95
 sharedTemp2
 PvTrainerMain.cpp, 95
 sharedTemp3
 PvTrainerMain.cpp, 95
 sharedTemp4
 PvTrainerMain.cpp, 95
 sharedTempAverage
 PvTrainerMain.cpp, 95
 shares.h, 97
 print_ser_queue, 98
 show_devices
 avr_1wire, 16
 show_status
 task_user, 45
 SpiMaster, 41
 recieve, 42
 send, 42

- SpiMaster, 42
- SpiMaster.cpp, 99
 - delay_ms, 99
 - delay_us, 99
- SpiMaster.h, 100
 - delay_ms, 100
 - delay_us, 100
- task_user, 44
 - print_help_message, 44
 - run, 45
 - show_status, 45
 - task_user, 44
- task_user.h, 102
- TempController, 46
 - TempController, 47
- TempController.cpp, 103
- TempController.h, 104
- temperature
 - avr_ds182x, 20
- toggle
 - FanDriver, 25
- UiController, 49
 - checkAngle0, 51
 - checkAngle1, 51
 - checkAngleAzimuth, 51
 - checkAngleElevation, 52
 - f1Display, 52
 - f1Update, 52
 - f2Display, 52
 - f2Update, 52
 - f3Display, 53
 - f3Update, 53
 - f4Display, 53
 - f4Input, 53
 - f4Update, 53
 - initDisplay, 54
 - setAngle0, 54
 - setAngle1, 54
 - setAngleAzimuth, 54
 - setAngleElevation, 54
 - UiController, 51
- UiController.cpp, 105
- UiController.h, 106
- VoltageSensorDriver, 56
 - read, 57
 - VoltageSensorDriver, 56
- VoltageSensorDriver.cpp, 107
- VoltageSensorDriver.h, 108
- write_0
 - avr_1wire, 16
- write_1
 - avr_1wire, 16
- write_byte
 - avr_1wire, 16
- write_byte_rev
 - avr_1wire, 17

