OVERHEAD TRAFFIC DETECTOR MOUNTING SYSTEM (Phase 2)

FINAL TECHNICAL REPORT

Submitted to

CALIFORNIA DEPARTMENT OF TRANSPORTATION (CALTRANS)

In fulfillment of

RESEARCH TECHNICAL AGREEMENT NO. 65A0166

by

DEPARTMENT OF MECHANICAL AND AERONAUTICAL ENGINEERING
UNIVERSITY OF CALIFORNIA
ONE SHIELDS AVENUE
DAVIS, CA 95616

Principal Investigators: **Fidelis O. Eke and Harry H. Cheng**

Research Assistants: **Ian Strimaitis and Mathew Campbell**

Contract Manager: **Joe Palen**

Period: **June 10, 2003 to April 30, 2007**

OVERHEAD TRAFFIC DETECTOR MOUNTING SYSTEM

PROJECT SUMMARY

Caltrans has funded the development of a new family of out-of-pavement electronic sensing devices for the purpose of monitoring certain characteristics of highway traffic. One promising example is a laser based overhead detector recently developed at UC Davis, and jointly patented by Caltrans and UC Davis. In order to deploy this new generation of detectors, there is a need to develop the capability to safely and efficiently mount these devices above highway traffic lanes, using existing overhead bridges and sign structures as support.

During the fiscal years 2000/2001 and 2001/2002, Caltrans funded a project whose main objective was to carry out a comprehensive engineering study of such a support platform. The project resulted in the complete design and construction of a working prototype of a trolley system that will carry the detection and monitoring devices that are under development or currently available.

The project described in this report constitutes a second phase of the overhead traffic detector mounting project. One important objective of this phase is to design and build a portable collapsible truss that can be easily deployed for off-site testing of laser detectors or other detection devices. Another goal is to develop, construct, and test a reliable mechanism for mounting and dismounting the trolley system onto and down from the tracks on which it rides. Occasional mount and dismount of the platform system is necessary for instrument swapping and for instrument/platform maintenance. The third and final objective is to build an improved trolley system with an upgraded motion control system. The goal is to equip the motion control system with a sensor that can relay the platform's position information to Caltrans personnel, and the ability to move the platform precisely to a commanded position above the roadway, without the need for visual feedback from an operator.

ACKNOWLEDGMENT

The material presented in this report is based on the research effort of several individuals. Major contributors to this work include Mr. Ian Strimaitis, whose M.S. thesis work consisted of the design and construction of both the collapsible truss and the lifting mechanism. Mr. Mathew Campbell is the only graduate student that participated in, and made contributions to every phase of the project. He also led the group of students that worked on the redesign of the trolley system. We also wish to acknowledge the assistance of post-doctoral researchers Zhaoqing Wang, Bo Chen, Ping Feng, visiting scholar Hong Duan, graduate students Stephen S. Nestinger, Yucheng Chou, Kabileshkumar Cheetancheri, and undergraduate students Jonathan Rogado, Saleh Tyebiee, Arman Rowhani, and Mark Bruening. Each of these individuals made important contributions to the final outcome of this research project.

We take this opportunity to express our immense gratitude to Mr. Joe Palen, Senior Research Engineer with ATMIS Development at CALTRANS, who served as Contract Manager for this project. His frequent advice and suggestions, based on experience and deep engineering insight were invaluable throughout the duration of the project.

Finally, we are most grateful to CALTRANS for its generous support of our research efforts through this technical research agreement No. 65A0166.

Fidelis O. Eke and Harry H. Chang
September 2007

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Background

The California Department of Transportation (Caltrans) is interested in monitoring traffic flow to improve transportation efficiency of California highways. Loop detectors are the current primary method to determine the presence of an automobile and the approximate speed. Because loop detectors are buried inside the road's pavement, installing, repairing, and replacing them requires lane closure and places maintenance workers near dangerous, high-speed traffic. Furthermore, loop detectors are known to have low reliably; only 20-40% of the more than 400 loop detectors installed in California's roadways produce useful data most of the time. Speed estimation methodology, using single loop detectors, is presented in Sun et al. (1999).

A vehicle's travel time from one point to another is one of the most important quantities that can be used to measure the effectiveness and efficiency of a transportation system. To determine the travel time, one identifies a specific vehicle as it passes a point A and then re-identifies the same vehicle at another point B, at a known distance away. Though loop detectors can sense the presence of a vehicle, they cannot actually identify a vehicle. Caltrans currently has no system in California capable of continuously measuring and reporting travel time for specific vehicles on specific highways.

In an effort to improve transportation intelligence, Caltrans is encouraging and sponsoring the development of alternatives to loop detectors. Many of the developing technologies use Video Image Processing System (VIPS), a computer system that interprets a video image. More information about VIPS and the need for traffic monitoring is presented in Malika et al. (1997) and Palen (1997). UC Davis has had technical research agreements with Caltrans to develop a sophisticated Array Based Detection System (ABDS) capable of counting automobiles more reliably, determining the speed with accuracy, and recognizing the profile of automobiles for re-identification. The working principle of the electro-mechano-optical ABDS is explained in references (Cheng, et al., 2001, Lin et al., 2001; Wang et al, 2004). Information about the mechanical design and an error analysis of the ABDS system are available in Wang, et al. (2003).

1.2 Problem Overview

Practically all of the new generation vehicle detection systems (including the ABDS) that are being developed are designed to monitor automobiles from above traffic. This means that a suitable method for supporting these devices above traffic must be devised. The current plan is to use existing sign trusses as mounting platforms for the detection devices. This eliminates the need to construct entirely new support structures solely for these detectors.

Deploying automobile detection units above highway lanes is an improvement from loop detectors because out of ground detectors are capable of obtaining more valuable information about traffic and do not require cutting into the road’s pavement. However, overhead detectors have several potential technical problems of their own. It will often be necessary to deploy and/or retrieve overhead detectors during the development stages. Even after these detectors are standardized, there will be the need to make changes to a detector, take a detector out of operation for maintenance, or replace one type of detector with another. All of these will normally require that one or more traffic lanes be closed, and, the use of one or more bucket trucks may become necessary. Requesting qualified personnel and a bucket truck to lift the detector up to the sign or bridge is a burden because it is time consuming, expensive, and involves the coordination of different Caltrans departments. The deployment system should also be vandal resistant and only operated and accessed by Caltrans personnel. The desire is to develop a deployment system that can position overhead detectors above highway lanes without lane closure, elaborate equipment, and with minimum disruption to traffic flow.

1.3 Previous Work

In 2001, Jacob Duane (Duane 2001, Duane, et al., 2004) proposed a deployment system that utilizes existing bridge and sign structures, and that is capable of positioning any overhead detection unit above highway lanes without lane closure, the need for a bucket truck, or any additional technical assistance. The design begins with a pair of rail tracks that are mounted to the underside of existing overhead sign trusses or overpasses. A motorized trolley hangs from and can run on these tracks. The trolley carries a universal platform that can connect to any detection unit. Once the detector is attached to the trolley, the trolley can be remotely commanded to position itself over any lane of traffic and monitor the automobiles passing directly below. Multiple trolley/detector combinations can be deployed by the same deployment system so several lanes of traffic can be monitored at the same location. A prototype trolley was designed and built to transport UC Davis' ABDS above lanes of traffic.

To retrieve the detector-trolley package for repairs or any other purpose, the design calls for the trolley to be driven along its tracks to one end of its support structure; here, it is above the shoulder of the road and no longer above traffic. At this location, an authorized Caltrans employee would be able to safely access the detector without disrupting traffic. A traditional way to access a detector at this location would be with a bucket truck. However, as indicated previously, this method requires coordination between different Caltrans departments. A second way to access the detector is simply with a ladder or stairway that leads up to the end of the trolley tracks. Lifting equipment up to the top of a ladder is unsafe for Caltrans employees and a stairway creates easy access for potential vandals. An automatic lifting and lowering mechanism that can lower the trolley-detector package from the end of the trolley tracks down to the shoulder of the road, and lift it back to the tracks was a part of Duane's proposed system in 2001. If Duane's system works as intended, then gaining access to the detector will not require any special equipment or personnel beyond the individual who needs access to the detector. Duane's concept for the lifting and lowering of the trolley was refined (Duane et al., 2004) to improve the alignment of the trolley's wheels with the track across the highway, but the concept has yet to be implemented and tested.

Figure 1.1 is a schematic view of how the overhead detector system, as well as the lifting and lowering mechanism, are supposed to be integrated with existing highway support truss. The trolley tracks are attached to the underside of the truss, and the lifting mechanism (LM) is attached to the end of the trolley tracks above the shoulder of the highway. The LM is shown lifting a trolley-detector package; a second trolley-detector package is shown already deployed to the middle of the truss span.

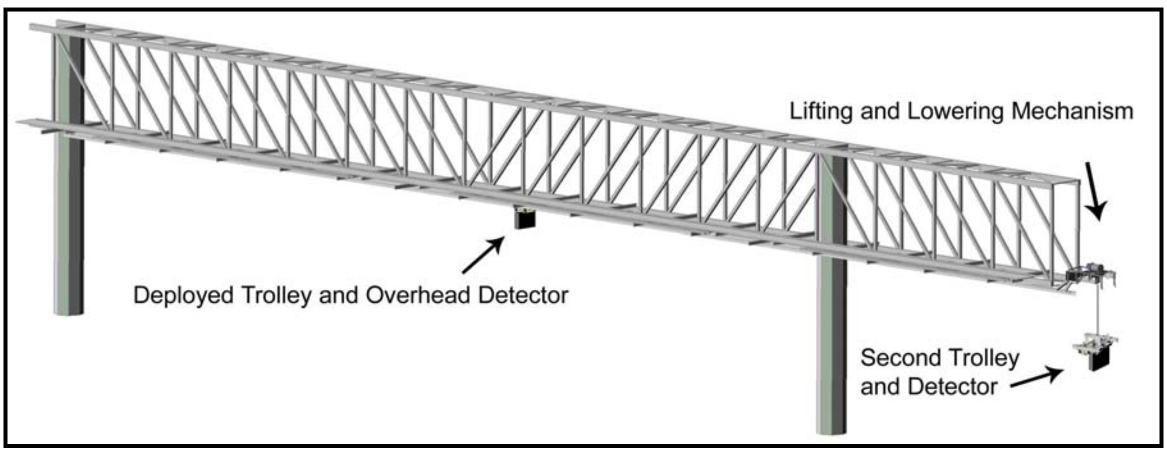


Figure 1.1, Trolley-Detector Package Deployment

1.4 Current Work

There are three main tasks that are to be accomplished in the current study. The first is to examine the most recent design concept proposed by Duane (Duane, 2004) for the trolley's lifting and lowering mechanism. The goal is to turn the concept into a working prototype that can be tested on a real freeway truss. With a working trolley and a functional lifting/lowering mechanism in hand, it is possible to carry out comprehensive tests of the overhead detector and its complete support system.

Rigorously testing the LMs, trolleys, and overhead detectors is essential for successful prototype development and integration of the entire system. A majority of the day-to-day testing of the individual components can be performed in a laboratory setting. Caltrans has approved a location on an Interstate-80 overpass for occasional on-site testing to assess the detector's performance in a true operating environment. Though it is essential to test over the highway with actual traffic, preparation for such tests takes significant time and coordination with Caltrans. Additionally, presentations to the project sponsors and other investigators will be necessary from time to time. Such an overpass with traffic on it and below it is not a practical location for presenting the trolleys, LMs, and new detectors. It is thus desirable to create conditions that will permit investigators and developers to extensively test detection devices and their support systems in an environment that closely simulates the true operating environment. This leads to the second task that is planned for this study. This task is to design and build a portable and collapsible truss, which, when fully deployed, will have the same height as a typical highway sign truss. The span is to extend at least one highway traffic lane in length so a car can be safely driven under it at freeway speeds.

Another item that needs some refinement is the control system for remotely controlling the motion of the trolley. Though the system is quite sophisticated as currently designed, it only operates in an "open-loop" mode. That is, the system has no sensor that can determine where the platform is located relative to its tracks. Thus, if an operator wishes to move a platform and its monitoring device to a specific location above a traffic lane, he/she would have to actually go to the vicinity of the platform, use the hand held remote controller to move the platform while visually watching to determine when the platform is at the desired location above the freeway. The operator will then remotely send a command to stop the trolley and apply the brakes. The need for visual monitoring of the platform's position clearly erodes the degree of accuracy that can be attained in the positioning of the platform. It would also be desirable for the control system to have a means of communicating its exact location (and possibly other platform related data) to Caltrans personnel. Hence, a third objective of this project is to develop an upgrade to the existing trolley design.

In summary, this project seeks to extend Duane's work by building a viable lifting/lowering mechanism for vehicle detectors and their supporting trolley. The project will also design and build a structure that can simulate the type of support trusses found above traffic lanes on California highways. The project will be concluded with the design and construction of an improved trolley system. Chapter 2 will focus on the design and construction of the collapsible truss. Chapters 3 through 8 will discuss the design of the lifting mechanism, and Chapters 9 will present the design and construction of an improved trolley.

CHAPTER 2

THE COLLAPSIBLE TRUSS

Caltrans has approved an I-80 overpass for testing overhead automobile detectors. This location, however, also includes the dangers of the general public in uncontrolled automobiles. A collapsible structure that mimics the I-80 overpass, and that can be set up anywhere at anytime, would allow for more frequent testing and better presentation environment, without the dangers of uncontrolled automobiles and highway road noise. Time consuming coordination to set up I-80 test date with Caltrans would also be avoided for a majority of the testing requiring automobiles.

A location on the UC Davis campus where a collapsible structure can be set up quickly and easily to simulate a highway overpass was secured. Permission has been granted to the UC Davis research team by the grounds division to use a ¼-mile stretch of a dead-end road on campus for testing and presentations on any day for any amount of time.

2.1 Specifications

A collapsible truss and supporting structure, simulating a highway overhead structure, needed to be designed to perform testing and for giving presentations of overhead detectors and deployment systems. The span should be modular in its design and have the ability to be set up in different widths depending on the test location topography or desired type of testing for the particular day. If the width of the road and shoulders is limited, the span should be setup with minimal length. If interference from cars in adjacent highway lanes needs to be simulated for a detector, a wider span should be set up for simulation of two or three highway lanes below.

The vertical supporting structure needs to be able to position the ABDS at a variety of different heights. Sometimes the detector needs to be at the height of a typical freeway overhead sign structure, which is at about 15'. At other times the detector or trolley may need to be constantly accessed by researchers and the span should then be positioned only five or six feet above the ground. There is also the necessity of positioning the detector at the same height above the road as it will be when mounted for testing at the approved I-80 overpass test site, which is 28' high.

Ease of transportability of the truss components is essential if it is to be assembled at different locations. The entire span and supporting structure should be transportable in the back of a pickup or small moving truck. It should be collapsible down to manageable pieces in weight and size so students, Caltrans workers, or anyone else who might need to use it can transport it and set it up. If the testing only lasts one hour it would be unfortunate if the setup and teardown times were two or three hours each. The goal for setting up or tearing down the mobile truss was between ½ and 1 hour. The trolley tracks need to be energized to supply the trolley with 12 volts and attach to the mobile truss in a manner so they are electrically isolated. The collapsible truss criteria are summarized in Table 2.1.

TABLE 2.1, MOBILE TRUSS CRITERIA

1	Design quickly as it is not the primary design project.
2	Reasonable material costs.
3	Support trolley track, trolley, and ABDS.
4	Tracks electrically isolated and capable of supplying power to trolley.
5	Test at same height as I-80 overpass near Truxel Blvd.
6	Test at variable heights below I-80 overpass height.
7	Modular.
8	Mobility and easy handling to testing locations.
9	Assemble in less than 1 hour.
10	Flexibility to allow for setting up at a variety of location topography.

In short, the mobile truss should be light, easily transportable, of reasonable cost, and should be designed and built in the shortest amount of time possible.

2.2 Design Concept

A major design philosophy for the collapsible truss was to use as much off-the-shelf products as possible.

Genie-brand material lifts were researched and purchased because they would give infinite vertical adjustment of the span from the ground to 25' in the air. Because the Genie Lifts are designed to only raise 25' in the air, the collapsible truss needed to be of a design that can hold the ABDS over 28' in the air, which will match the height of the detector when it is tested on the I-80 overpass.

To create a collapsible span, the idea of using aluminum extension ladders as connecting sections was conceived. The thought behind this idea was to use the ladder rungs as attachment points between each section. At first, a $^{1}/_{4}$-scale prototype was built to test the concept. The concept seemed viable and an analysis of a full-scale collapsible truss was then performed. It took two weeks to develop this idea and obtain all the parts. It took two days to build the entire structure, and two more days to add the track, the additional height option, and other accessories. The entire design from conception to deployment took less than a month and about $2500 in materials. Figure 2.1 shows the prototype set up on the sidewalk in front of Bainer Hall on the UCD campus.

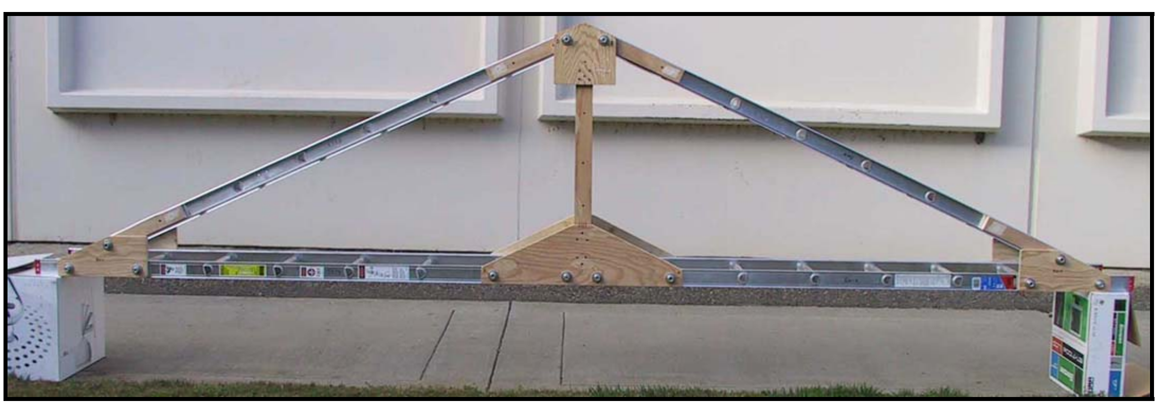


Figure 2.1, Quarter-Scale Span Prototype

2.3 Geometry Implemented

A few different geometries of a full scale span were evaluated for loading. All ladder sections were analyzed as two force members; the outcome for all the designs were very similar, and the final design was chosen for its transportability, ease of assembly, esthetic appearance, and optimization of members used. A compromise was made between the length of the ladder pieces used and the final weight of the truss. If long spans of ladder are used, fewer joints would be required, and less hardware would be needed.

The geometry implemented in the mobile truss is optimized for weight, mobility, and ease of transportation. Transportation is made simple because no single piece of the span is longer than 10' and weighs more than 15_{lbf}; the full-length truss weighs under 400_{lbf}. The truss can be set up in different lengths depending on the space available at a testing location. More sections can also be added to the current design if a longer span is ever desired. The full-length setup has been tested with about 1500_{lbf} of weight distributed unevenly along its bottom ladder sections. The design does not allow the LM to be cantilevered off to the side of the setup; this wasn’t a concern because it is only for testing purposes. The cantilevered mount is only essential in LM’s final implementation on the highway. The full length truss is shown in frame A of Figure 2.2, and a $^{3}/_{4}$-length setup is shown in frame B.

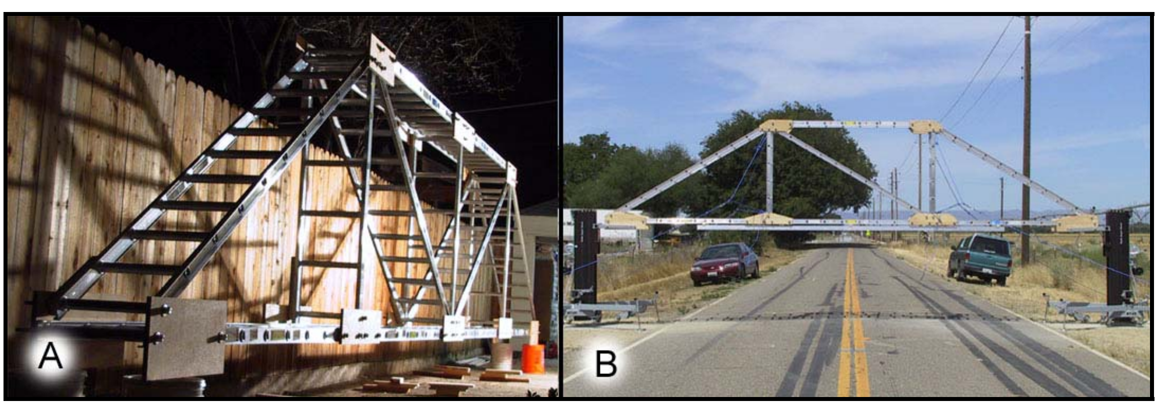


Figure 2.1, Completed Truss

2.4 Setup Configurations

Four different collapsible truss configurations can be used, depending on the level of testing needed. The first configuration is the most practical for equipment testing without a vehicle. Connecting each Genie Lift to one side of a single piece of ladder would allow for testing of the ABDS or LM with the smallest footprint. Figure 2.3 shows this setup.

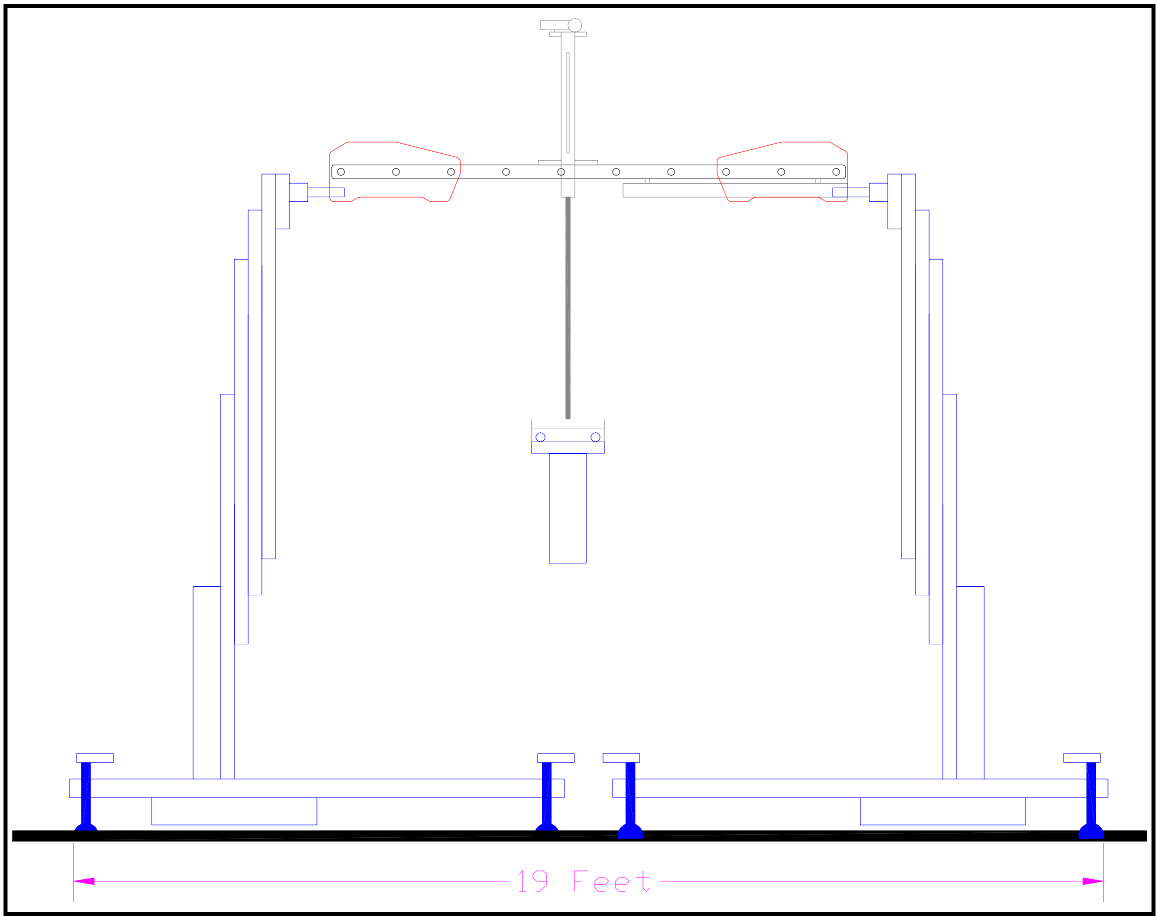


Figure 2.2, Single Section Truss Setup

The next smallest footprint is with the two outside sections of the collapsible truss put together. For reference, a standard highway lane is 12' wide and the opening between the Genie Lifts for this setup is only 10'. Driving a vehicle through this setup isn’t recommended unless at a very slow speed. Figure 2.4 shows the two-section setup.

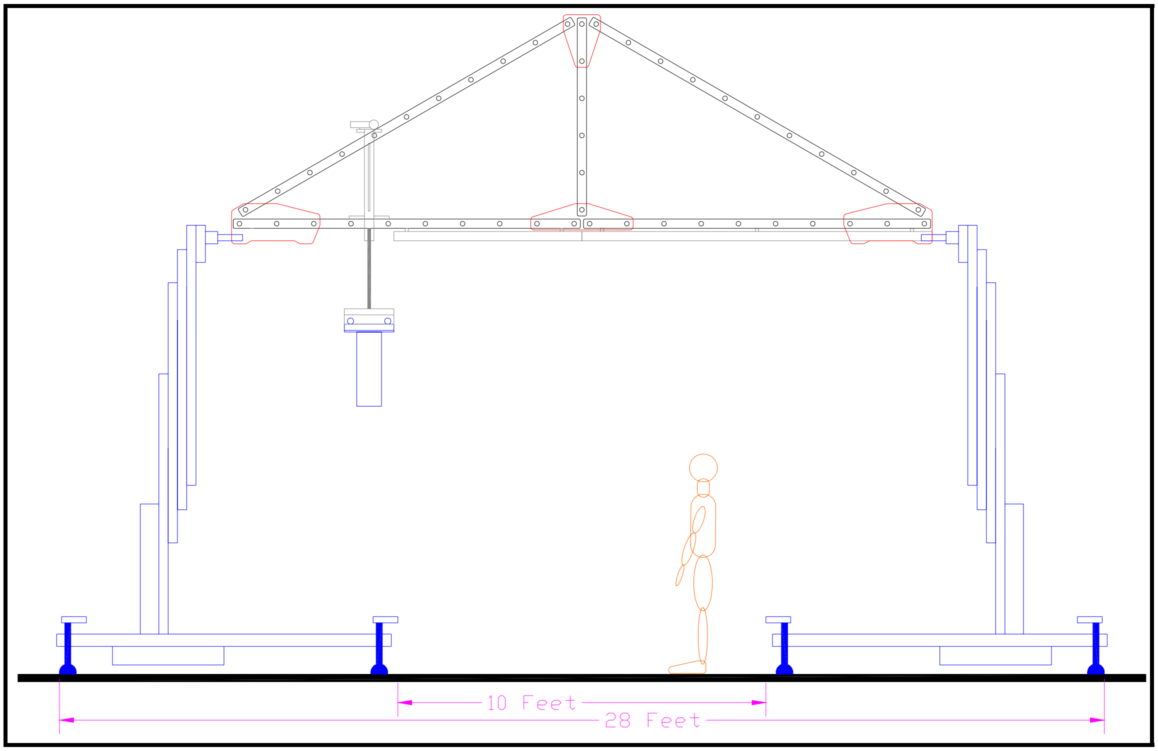


Figure 2.3, Two-Section Mobile Truss Setup

2.5 I-80 Testing Height

Extension legs can be added onto any mobile truss configuration to add eight additional feet to the total height of the span. With the legs added to the mobile truss, the ABDS optics can reach the same height as when testing on the I-80 overpass near Truxel Blvd. Figure 2.5 shows the mobile truss in this configuration.

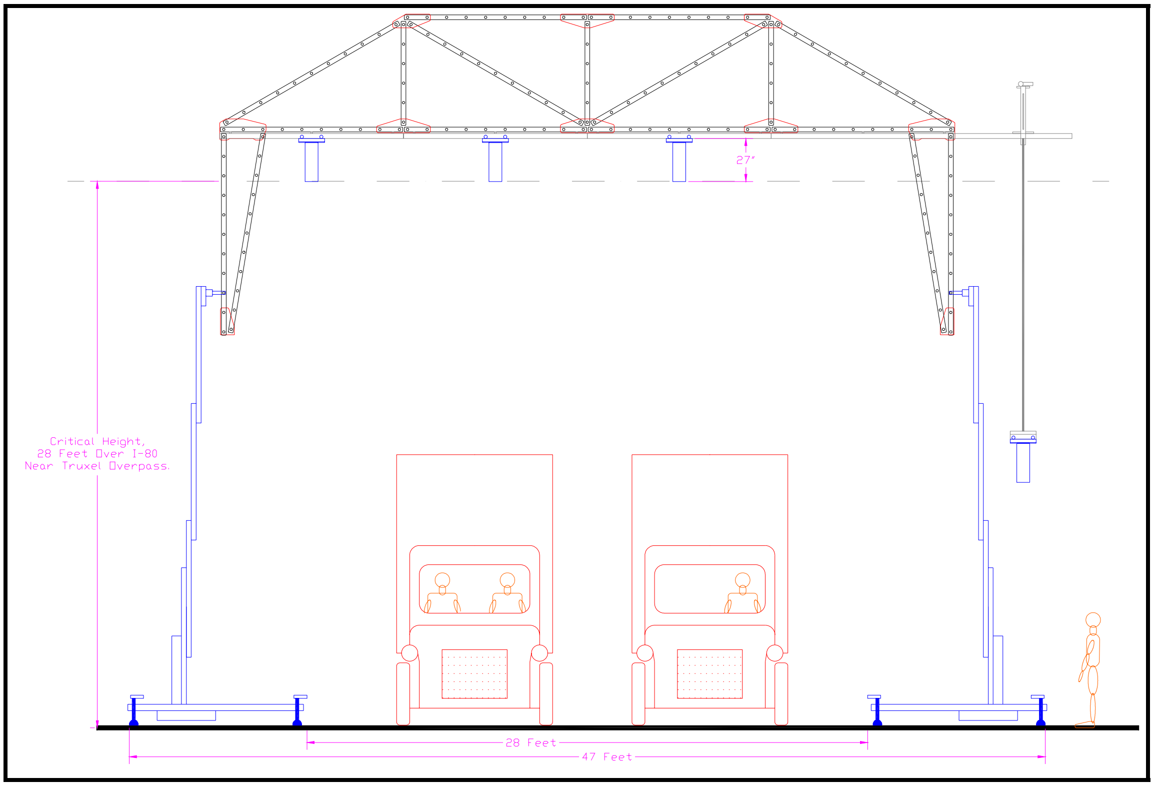


Figure 2.4, Mobile Truss at I-80 Testing Height

2.6 Old Hutchison Road Testing

Permission was granted by the University to test on Old Hutchison Road. The three-section setup is used for testing over Old Hutchison Road. The road is 22' wide with 12,000 volt power lines traveling above the north shoulder of the road. Government regulations require that a 10' minimum clearance be kept around power transmission lines with a voltage between 300 and 50,000 volts. Thus, the maximum height the ABDSs' detector can be raised at this testing location is 16'-4". Figure 2.6 shows the schematic of the three-section mobile truss on Old Hutchison Road. A UCD high-voltage electrician visited out test site to inspect and approve the mobile truss setup aver Old Hutchison Road. A picture was taken at Old Hutchison Road during a presentation given to Caltrans and is shown in Figure 2.7.

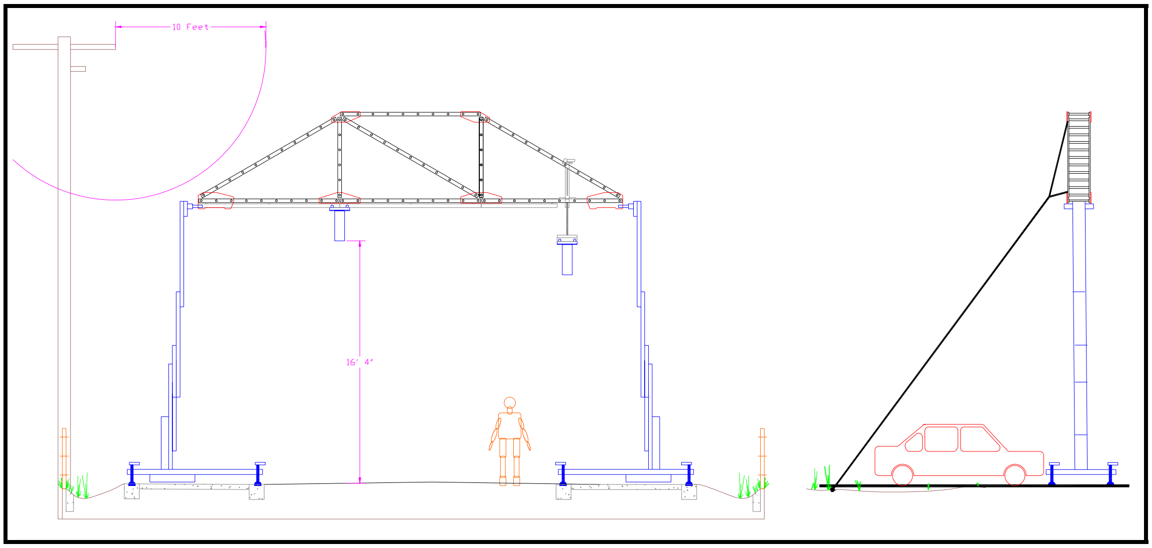


Figure 2.5, Three-Section Mobile Truss Setup

Figure 2.6, Presentation Setup

The front (left side) view of Figure 2.6 shows concrete footings on each side of the road for the Genie Lifts. The footings were poured specifically for the $^{3}/_{4}$-length truss setup at this location. Base rock was packed around each footing to create undulation type topography if a vehicle were to drive on the shoulder. Frame A of Figure 2.8 depicts the concrete pad on the north side of Old Hutchison Road. Frame B shows the pad after poring and before adding base rock. Frames C and D show the sunken hooks used to anchor the stabilizing ropes to the ground. Two hooks are anchored in concrete on each side of the road near the barbed wire fence.

Figure 2.7, Concrete Pads

2.7 Components of the Collapsible Truss

2.7.1 Ladders

Frame A of Figure 2.9 shows a display of ladders in the store. Werner manufactures five different grades of ladders specified by load capacity. The 200_{lbf} ladder grade was chosen for the quarter-scale truss prototype because it was the lightest weight and the least expensive. When this ladder was placed on the ground (frame B) at Home Depot, supported at each end, and load tested, it felt unstable. The 225_{lbf} and 250_{lbf} was also tested in the same manner. The 225_{lbf} was more stable, and the 250_{lbf} was even more stable. Home Depot didn't stock 300_{lbf} or 375_{lbf} extension ladders, but they were much more expensive and would have made the truss too expensive. The 225_{lbf} and 250_{lbf} rated extension ladders were evaluated more thoroughly.

Figure 2.8, Ladders at Home Depot

The 225_{lbf} and 250_{lbf} rated ladders have the exact same rung structure, but the 250_{lbf} rated ladder has slightly heavier duty webbing. This extra webbing rigidity kept the 250_{lbf} rated ladder from sagging and shimmying as much when load tested. The 250_{lbf} rated ladder also added about 10% to the total cost of the ladders and the total truss weight increased by about 15%. No undesired weight was added with the rungs because they were the same on both ladder weight grades. The Werner website was used as a resource to better understand ladder basics and duty ratings. The Type I 250_{lbf} ladder grade was chosen for the mobile truss span. The ladders purchased are shown in Figure 2.10 with the duty ratings shown.

Figure 2.9, Ladders Ready for Cutting

2.7.2 Fish Plating

Marine grade $^{3}/_{4}$" plywood was used as fish plating to connect the mobile truss together at the joints. All plies of marine grade plywood are of the high density Douglas Fir wood, typically only used on the outside layer of common construction plywood. This makes the wood much stronger in sheer and will not compress nearly as easily when the nuts and bolts are tightened during assembly. The cost for a 4' x 8' sheet of $^{3}/_{4}$" marine grade plywood is about $80, while similarly looking construction plywood is $20 to $30. Four different patterns were created for the entire truss; only three are required to assemble the truss without the additional height legs. Frame A of Figure 2.11 is of pieces of plywood connecting the span together for the first time. After the fit of all pieces was checked, the corners were rounded and sanded for safer handling and more stylish appearance. The finished fish plating in use is shown in Frames B and C. The CAD drawings for each fish plate design are in Appendix III.

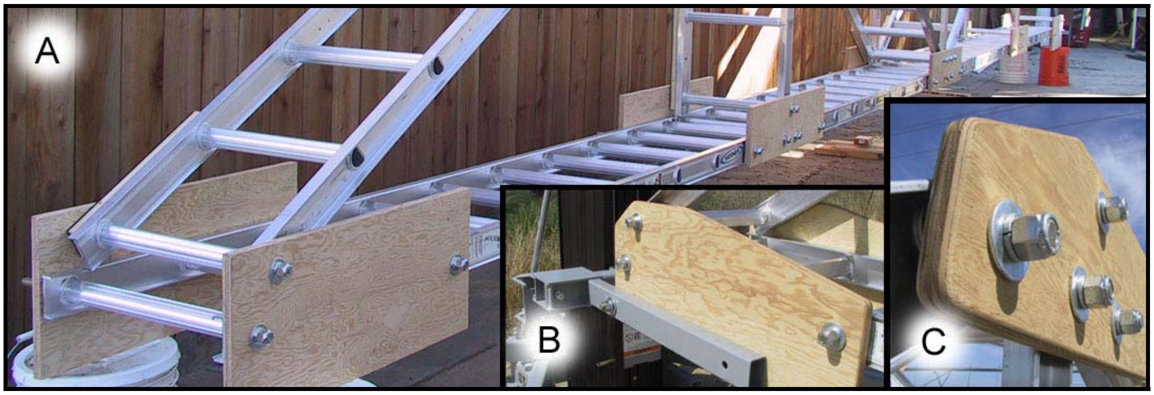


Figure 0.10, Fish Plating

2.7.3 PVC Inserts

Determining the hardware to connect the sections of ladder required a novel solution. PVC garden sprinkler piping and fittings were found, and were almost the perfect size to slide inside the rungs of the ladder sections. Schedule 40, $^{1}/_{2}$" inside diameter PVC sprinkler pipe was used for the piece that extends through the entire ladder rung. The O.D. of this pipe is over $^{1}/_{2}$" smaller than the smallest inside dimension of the ladder rung. Sprinkler pipe couplings were added to each end of the pipe. The couplings were just the right size to fit through the ladder rungs with a clearance fit. This needed to become an interference fit so the inserts could be hammered into the ladder rungs and not fall out. (Notice the rubber mallet in the bottom right hand corner of frame B in Figure 2.12)

Figure 2.11, PVC Inserts

To make this an interference fit, with the asymmetrical shape of the ladder rung, the PVC couplings were sanded down on one side and epoxied or glued to an ABS shim. Frame A of Figure 2.12 is the box of couplings and shims epoxied together. The assembled inserts are in frame B. Figure 2.13 illustrates how the insert fits into the ladder rung with a span ladder piece from the $^{1}/_{4}$-scale prototype. Each coupling was sanded down slightly in one spot (flatted) before pounded into the rung.

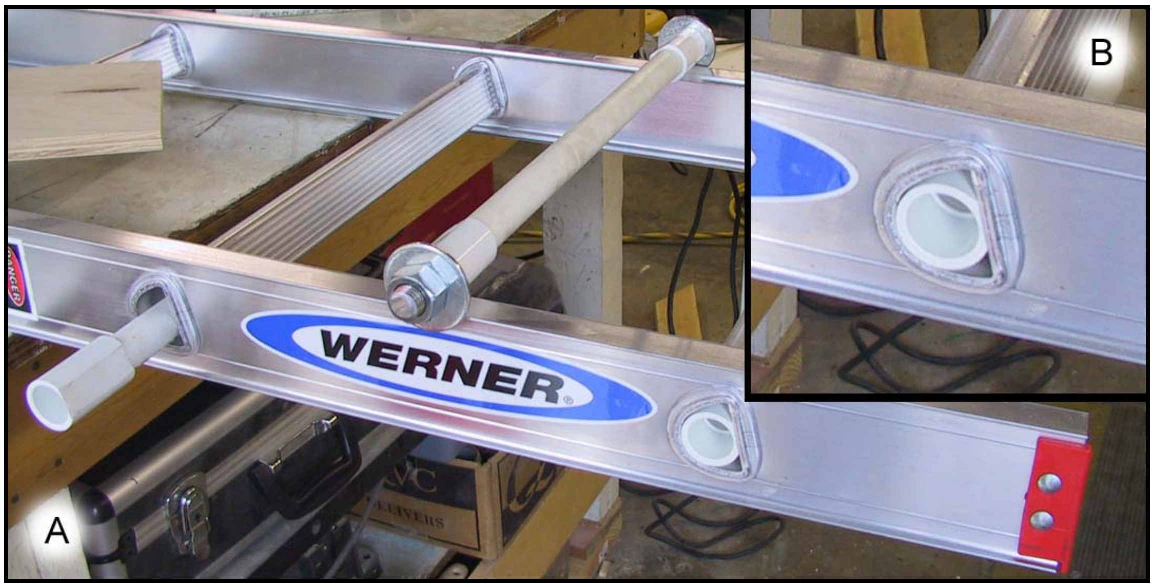


Figure 2.12, Ladder Rung Detail with PVC Insert

2.7.4 All-Thread Rod

All-thread rod was chosen for three reasons. The first is that $^{5}/_{8}$" diameter rod slides nicely through a drilled-out piece of $^{1}/_{2}$" PVC garden sprinkler pipe. The second is that the $^{5}/_{8}$" rod was available for $2 per 24" length at Blue Collar Supply. The third reason is that the $^{5}/_{8}$" rod will be able to withstand much greater shear forces than will ever be applied in the mobile truss application. The 24" lengths were cut down to about 20", just long enough to pass through the ladder, two pieces of plywood, and the nuts and washers. Frame A of Figure 2.14 shows the all-thread rod. After the rod was cut down to length, frames B and C show how it sticks out just far enough to attach the nuts on either side.

Frame D shows a single connection with a piece of all-thread before it was cut down to size.

Figure 2.13, All-Thread Rod

2.7.5 Nuts and Washers

Grade 2, low carbon steel nuts, nylon lock nuts, flat washers, and spring washers were used to connect the mobile truss sections. As illustrated in frame C of Figure 2.14, a nylon lock nut was tightened onto one end of the all-thread rod. A spring washer and flat washer were placed onto the rod before going through the wood and ladder piece. On the other end of the installed rod, illustrated in frame B, a flat washer was put on to the all-thread rod before the last nut, to tighten the connection together, was threaded on.

2.7.6 Genie Lifts

Two ST-25 Genie Lifts were purchased to support the structure on either side. Each lift's connection arm can be raised from about 16" to about 24' above the ground. Each lift has a maximum load of 650_{lbf}. Additionally, Genie Lifts have a very small stowed footprint (shown in frame A of Figure 2.15), which is less than four square feet at the base. They have casters for easy maneuvering and tailgate wheels (shown in frame B) for easy loading and unloading. The Genie Lifts can become very dangerous pieces of machinery if used improperly; proper use of the Genie Lifts and mobile truss is essential for everyone's safety. Read the Standard Operating Procedures for the Mobile Truss

(Appendix X) and the Genie Lift Operator's Manual (Genie, 2001) before engaging in any setup or operation. Frame C shows a Genie Lift being unloaded from a truck. Frames D and E show the Genie Lift hooked up to the mobile truss.

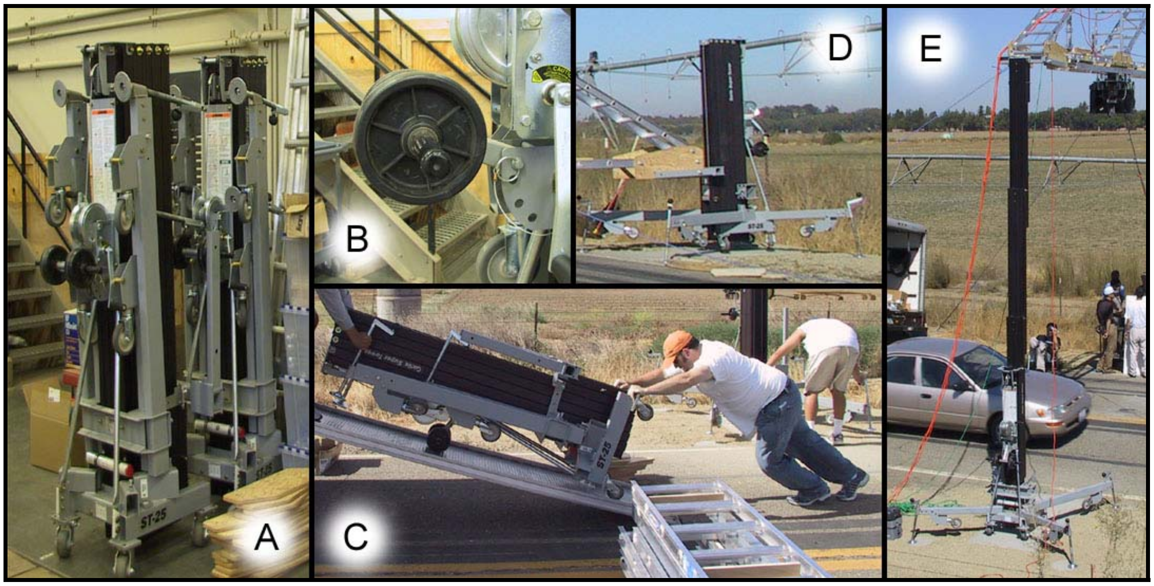


Figure 2.14, Genie Lifts

2.7.7 Stabilization Ropes

Though the Genie Lifts are designed to lift materials up to 24' in the air, stabilization ropes are used while lifting the span. The ropes also tie off the span to the ground after the span has been raised to testing height. The top of frame A in Figure 2.16 shows the two ropes connected to one side of the mobile truss. Frame B is a close-up view of the connection. Frame C shows one rope synched down with a tie-down to the eye-hook buried into the ground with concrete. Frame D is a close-up view of the connection to the ground. Notice the person in frame A holding the two stabilization ropes for that side of the truss while it is being raised. This person can dampen out swaying of the truss until it is securely fastened to the embedded eye-hooks mounted in the concrete pads.

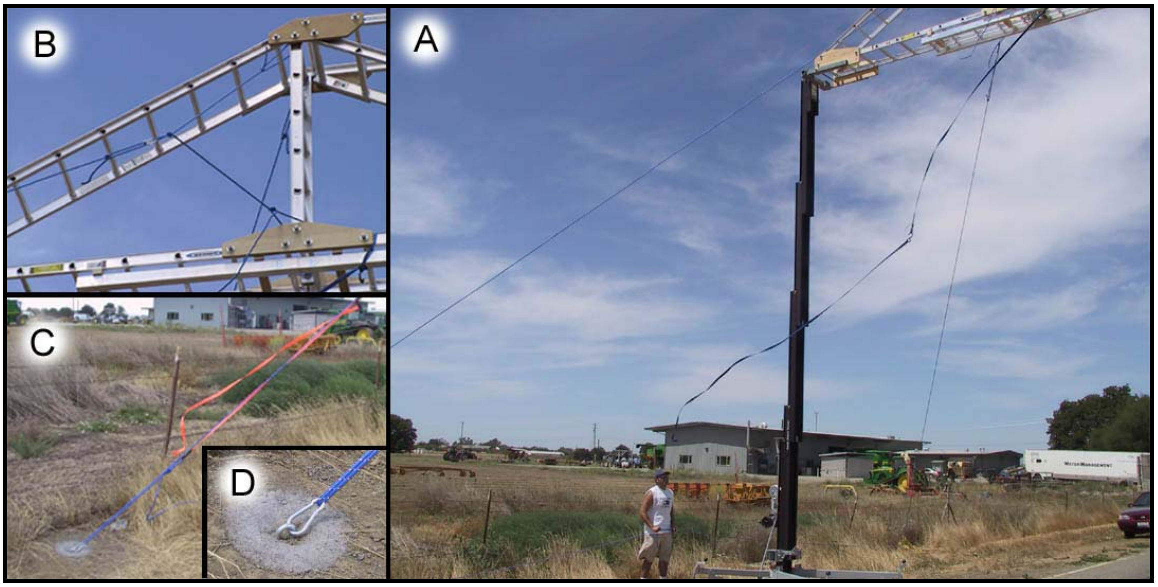


Figure 2.15, Mobile Truss Stabilization Ropes

CHAPTER 3
BASIC DESIGN OF THE MOUNTING SYSTEM

Duane (2001, 2004) came up with a concept of how to transport a detector between the shoulder of the road to a monitoring position above highway. He also built the first prototype trolley, which can shuttle a detector along the underside of an overhead truss. The next step is to build the first prototype lifting mechanism to transport a detector between the ground and the underside of an overhead truss. The following sections describe the concepts proposed by Duane for the mobile platform (or trolley system) as shown in Figure 3.1, and the lifting mechanism.

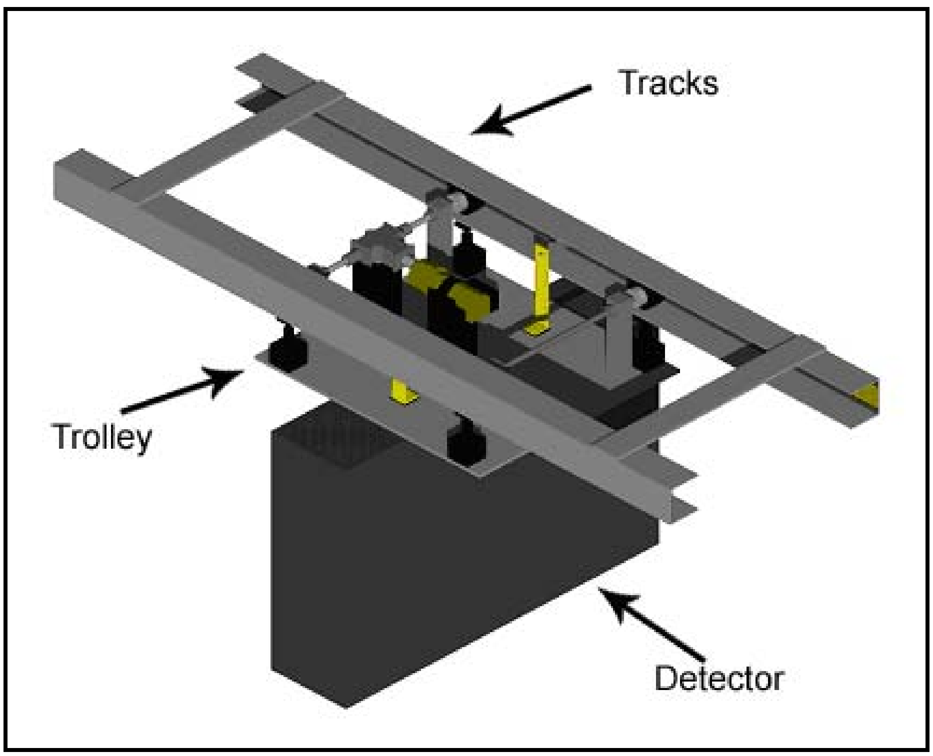


Figure 3.1, Trolley Conceptual Model

3.1 Trolley

The overhead automobile detector deployment system is to retrofit to the underside of existing structures eliminating the need for a special truss to be built. The main components of the system are a trolley that rides on a pair of tracks attached to the underside of the structure, and a lifting mechanism to raise the trolley from the ground up to the tracks running above the highway. The detector will be attached to the belly of the trolley on the ground and will remain on the trolley at all times. The plan for the tracks is to use two aluminum C-channels running parallel to each other. The tracks are held a fixed distance apart and attached to the underside of the structure approximately every 6'. These tracks are connected to the bottom of the truss walkway I-beams using clamps. The trolley is capable of carrying any overhead detector such as the ABDS. The trolley is equipped with four electric motor driven wheels which allow the system to ride along the C-channel tracks so that the whole system can be positioned at any desired location over traffic.

The mobile platform is able to accommodate many different monitoring devices. This is possible because of the use of a mounting plate that allows various shapes and sizes of monitoring devices to be attached to the trolley. Different monitoring devices may need unique mounting plates; however, this should not pose a major problem since these plates are inexpensive and can be easily fabricated. The main advantage of the mounting plate idea is that the trolley itself will not have to be modified to accommodate a device. Supplying power to the trolley system required a novel solution. The method developed by Duane utilizes a thin copper strip attached to each piece of C-channel track to supply power to the trolley. The physical power connection between the strips and the trolley is made using metal brushes that are attached to the trolley and that continuously rub on the strips. This eliminates the myriad problems associated with the use of cables to conduct power directly to the trolley. One of the copper strips is to be connected to the positive terminal of a power supply and the other strip to the negative terminal. The strips are electrically isolated from the tracks via a thin plastic backing. This method of supplying power to the trolley allows the trolley to be a freely moving system that is uncoupled from other trolleys on the same pair of tracks. The trolleys can thus be repositioned quickly and easily, and without cables dangling over highway traffic.

3.2 Lifting Mechanism

Duane also proposed a novel idea for a lifting mechanism to mount/dismount the trolley with the tracks above the highway. The mechanism is shown in Figure 3.2. It consists of two major parts – a mobile plate (detector mounting platform), and its support assembly. The mobile plate has two short pieces of trolley track spaced the same distance apart as the trolley tracks extending across the freeway. These tracks are held in place by the mobile plate. Connected to the top of the mobile plate is a cone called the plug with a pin on its side.

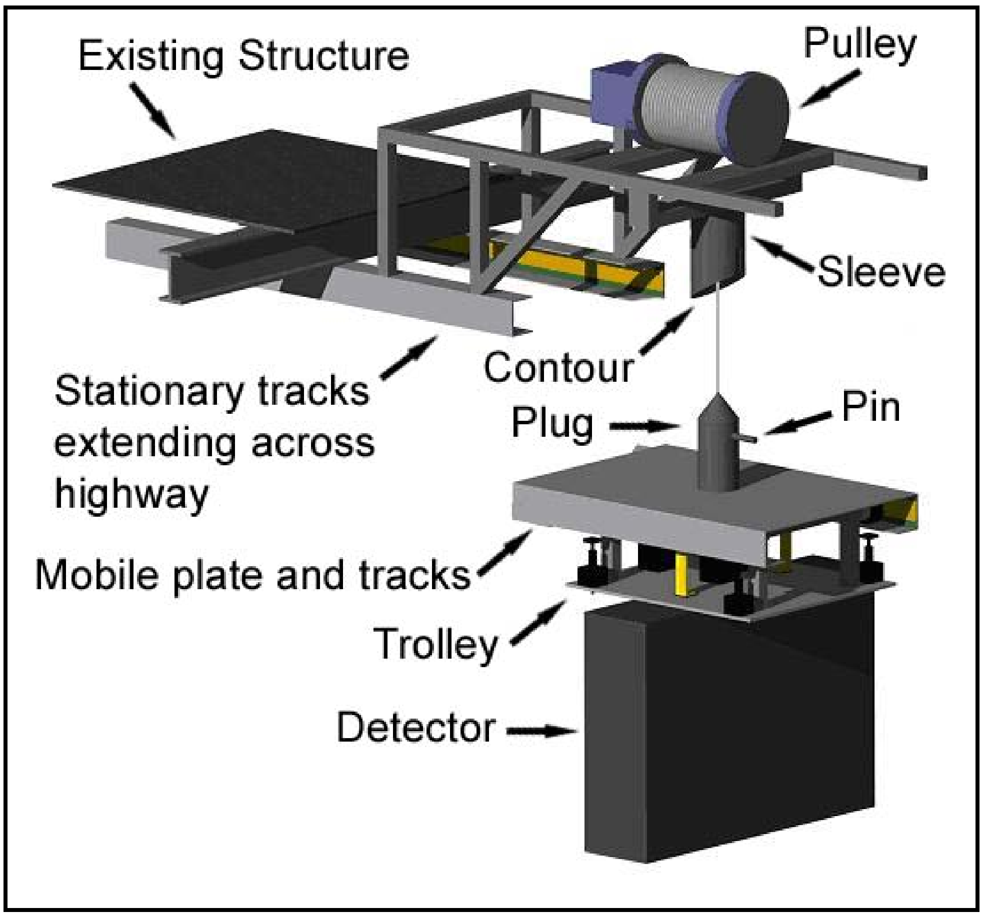


Figure 3.2, Lifting Mechanism Conceptual Model

The second part of the lifting mechanism is a winch system that is rigidly and permanently attached to the support structure where the trolley tracks end. The winch system includes a small motor that is used to wind or unwind cable from a spool. One end of the cable is attached to the plug on the mobile plate, and thus carries the platform at all times. The same cable is passed through a sleeve, and connected to the pulley drum. During normal operation of the trolley system, the mobile plate is held at one end of the trolley tracks in such a way that its short section of track is properly aligned with the track running above the highway. The mobile plate is held up by the cable and winch assembly. When access to the overhead detector is necessary, the trolley is remotely commanded to move to the end of the tracks where the lifting mechanism is located. The trolley can then continue and drive onto the tracks of the lifting mechanism. The trolley's static constraint system is remotely commanded to engage, so that the trolley is securely clamped onto the platform. The motor for the winch system is then commanded to turn the cable pulley to lower the mobile plate. When the detector reaches the operator standing on the ground, the lifting mechanism can be commanded to turn off the winch. The trolley can then be dismounted from the mobile plate and serviced.

To return the trolley and ABDS to above the highway traffic for monitoring, a reverse procedure is used. The winch is activated to raise the mobile plate, with the trolley system firmly held onto the platform. As the mobile plate is raised the trolley system will typically be spinning from the wind. However, as soon as the alignment pin of the plug hits the contour of the sleeve the pin pushes off of the contour and the mobile plate's rotation is controlled. The mobile plate is rotated to where the short section of track on the LM are aligned with the track extending across the freeway. The trolley can then be commanded to release its constraint system and drive onto the tracks extending across the highway and position itself over highway traffic.

The deployment and retrieval described above eliminates the need to hoist someone up to the supporting structure. Direct access to the detector is achieved at a location off highway, and even off the shoulder of the road. This is a safe way to access an overhead detector, and saves both time and financial resources by not involving personnel from several Caltrans departments.

3.3 Construction of the Lifting Mechanism

The LM design proposed by Duane is only a concept thus far. Along with building a support structure to mimic an overhead highway structure, a fully functional and deployable LM prototype must also be built. The prototype will follow Duane’s alignment concept and also interface with the trolley and overhead truss structure as he envisioned.

3.3.1 Specifications

An intermediate task that needed to be completed before design and construction could begin, was to determine the specifications necessary for the LM design. The most important criterion was safety. If the design were to ever become unsafe and have potential to injure any person, its evolution needed to be redirected back to a safe design. The people it could possibly injure would include and are not limited to the operator, on-lookers, automobile drivers, transients, or even the designers operating the LM during development and testing.

The second most important criterion for the design was that it be reliable; reliability was significant for a number of reasons. A lot or resources are being devoted to the design of a deployment system that will simplify the deployment and retrieval of automobile detectors from above lanes of freeway. For the outcome of this deployment design project to be viable, the LM must work perfectly every time. It must not become a problem in itself that requires resources to repair or maintain. One of the reasons the deployment system is being developed is because it will be used to routinely service equipment that is under development. It simply does not make sense to use unreliable equipment to retrieve a detector that you know will need retrieving. To make sure the LM stays reliable it needs to be protected from the weather. This includes rain, sun, dust, and even nesting birds that may disrupt the mechanics.

After safety and reliability are assured, the next most critical criterion is to be able to perform the intended operation of deploying and retrieving the trolley, with its detector, between the end of the tracks and the person on the ground. Different detectors may be mounted to the trolley with its universal platform and this loading may be asymmetric, especially during the products development. If the trolley is ever loaded asymmetrically, the mobile plate or entire LM will need to accommodate it accordingly, in particular during the alignment phase. The current trolley design weighs about 25_{lbf} and the current detector being developed at UC Davis is about 50_{lbf}. This total weight of 75_{lbf} is believed to be a typical, if not maximum weight for any detector-trolley package under development. Though weights of detectors and trolleys are expected to decrease, it was determined that the LM should be able to lift this 75_{lbf} at a minimum.

Once the LM has lifted the trolley all the way up, and the mobile plate is secured, the trolley is to drive onto the tracks extending across the freeway. Because the trolley gets its power from the track it drives on, the short section of track connected to the mobile plate must become energized so the trolley can harness power and drive off of them. After the basic mechanical operation is under control, the next consideration would be to eliminate the possibility of vandalism of the LM and its controls. At the same time, the operator must be at a location on the ground near the LM so he/she can stop the lowering of the detector-trolley package when it reaches the ground.

From an operator's point of view, the quicker the LM can lift and lower the trolley, the more efficient its design. Any deployment system, such as the one proposed, will save an immense amount of time because of the technical resources and support it eliminates. To keep this design novel, it would only make sense to have the retrieving and deploying time between the tracks and ground reasonably short. While 30 minutes may be very reasonable to wait for the LM to operate (the previous alternative was two Caltrans departments, 3 employees, and half of a day), a descent or ascent time of 1 to 5 minutes would be ideal. The last criterion to remember is that Duane has come up with an incredibly simple idea in concept. Sticking closely to his idea will render a very simple and reliable design. The preliminary criteria compiled for the design are shown in Table 3.1.

TABLE 3.1, LIFTING MECHANISM CRITERIA

1	Safety
2	Reliability
3	Weather proof
4	Deploy and retrieve a 70_{lbf}, asymmetrically loaded, trolley-detector package
5	Vandal deterrent mechanical design and control system
6	Quick lifting and lowering
7	User controls the LM from the ground near its mounted operating location
8	Follow Duane's Concept because it is an exceptionally simple and novel idea

3.3.2 First Prototype

A group of undergraduate students made a first attempt to build a partially working prototype of Duane's design as part of their senior design project. The purpose of this first prototype was to begin to understand the heart of Duane's design - how the plug would slide into the sleeve, and how the pin would perform the rotational alignment. The prototype they came up with is illustrated in Figure 3.3. This prototype's plug was made from a piece of solid aluminum round-stock, and the sleeve was made out of an aluminum cylinder. The plug was mounted to a piece of $^1/_4$" aluminum plate. A piece of $^1/_2$"-diameter nylon rope, and a DC gear motor were mounted to a metal support frame and used to pull the plug into the sleeve just as Duane's design called for. Frame A of Figure 3.3 shows the plug and sleeve side by side, with the pin sticking out of the side of the plug. Frame B of Figure 3.3 shows the sleeve over the plug with the bottom of the sleeve's contour resting on the pin. Notice that the pin is resting on somewhat of a flat spot on this sleeve and does not naturally twist in either direction to perform the rotational alignment. Frame C of Figure 3.3 shows the profile of the contour from another perspective.

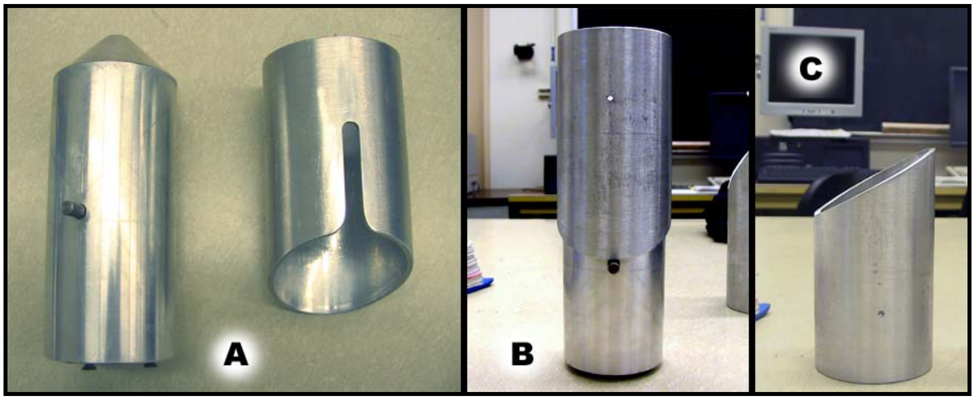


Figure 3.3, First prototype Components

Three new criteria were acquired from the problems encountered with the function of this first LM prototype. First, the contour the pin slides on needed to be redesigned so it does not have any flat spots. Second, there was too much friction between the pin and the sleeve's contour when it did slide along the contour. Third, because the plug and sleeve are both made of aluminum, there is too much friction between the inner wall of the sleeve and the plug when the pin tries to rotate the plug.

3.3.3 Construction Strategy

After studying the functioning of the first prototype, it was realized that constructing a fully functional, reliable, economically affordable, safe, and esthetically appealing LM would take many steps to develop. The strategy adopted was to build a series of prototypes until every detail of a reliable LM was understood. A total of 5 prototypes were built in a prototype development shop using as many off-the-shelf parts and products as possible.

After all the necessary mechanics were determined and completely understood for a reliable LM, the next step was to build the first, deployable prototype. This prototype would be very robust, look professional, and be ready for use in the environment it will ultimately be deployed in. Attention must be paid to every detail in the construction of this prototype because it must work perfectly every time.

3.3.4 The Final Off-the-Shelf Parts Prototype

The final off-the-shelf parts prototype perfected all of the design concepts of the previous prototypes. Frame A of Figure 3.4 shows this prototype mounted on a piece of the collapsible truss. Frame B shows the lifting mechanism's mobile plate hanging below the stationary plate. Frame D introduces a new component called the slider, which is made of clear acrylic for this prototype. The slider sits at the bottom of the sleeve, extending just below the sleeves contour, and holds the cable in the center of the sleeve to ensure the plug slides into the sleeve when it comes back up and begins the rotational alignment. When the mobile plate is docked, it is essential that it be secure and precisely positioned so the trolley can drive off the mobile tracks and onto the tracks extending across the highway. A new component called the final alignment cone was introduced to make sure the mobile plate is precisely aligned and secure when docked. Frame E is a view looking up at the stationary plate and shows a set of four final alignment cones. There is also a set of four final alignment cones on the mobile plate to mate with these cones, which can be seen in frame B of Figure 3.4. The final alignment cones on this prototype were made from white PVC pipe and white Delrin rods. The pin on the side of the plug has also been replaced by a ball bearing to significantly decrease the friction that inhibited the alignment function in the first prototype. The sleeve was made of ABS plastic and the plug was made of acrylic to reduce the friction when the ball bearing performs the rotational alignment by rolling along the sleeve's contour.

Because the trolley gets its electrical power from the tracks it drives on, the mobile tracks of the LM must become energized if the trolley is to drive off of them. Two black power transmitting modules made of garden pop-up sprinkler heads, which are shown in frame E, supply power to the mobile trolley tracks when the mobile plate is docked. The mating power transmitting modules on the mobile plate are also made of black plastic components from garden sprinklers, and can be seen in frame B of Figure 3.4.

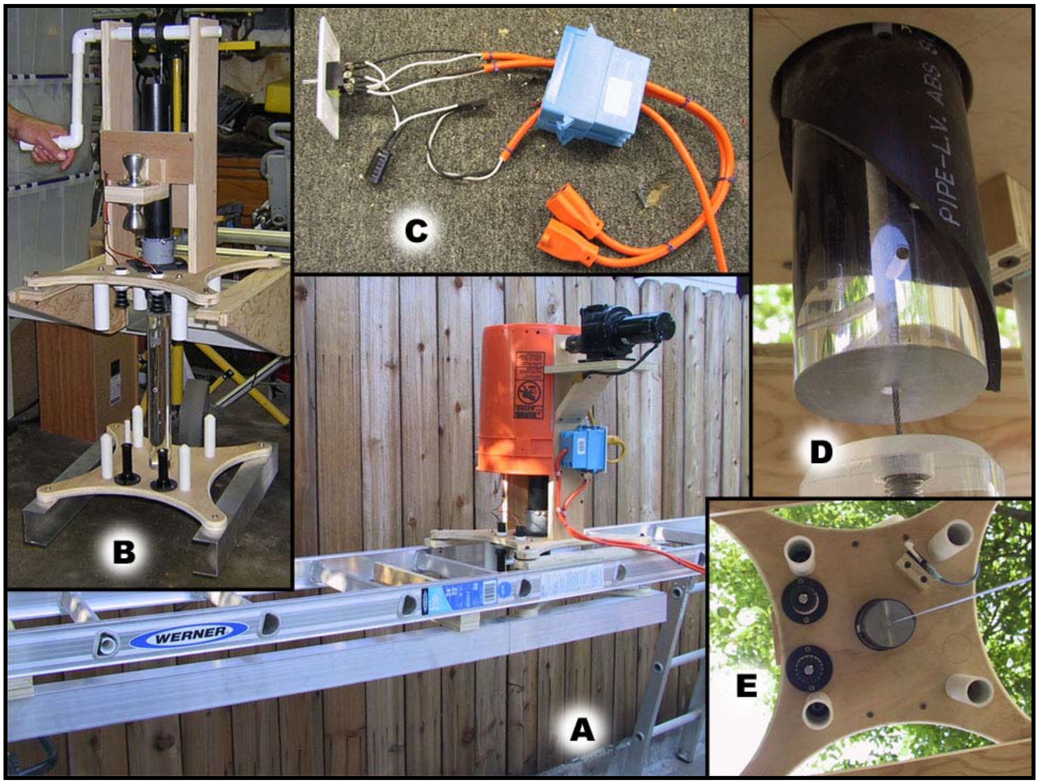


Figure 3.4, Fianal Off-the-Shelf Parts prototype

This prototype was eventually modified to include a gear motor. The friction and inertia of the gear motor must be able to hold the mobile plate with trolley-ABDS package in midair and not unwind when the motor turns off. No specifications are given for most motors that specify how much applied torque to the shaft is needed to turn a gear motor armature. Through experimentation, it was determined that a 1/20-HP motor with 190 in-pounds of output torque through a gear ratio of 130:1 in a parallel shaft gearbox is not enough to hold up 70 pounds of force ($_{lbf}$) in midair and as a result, the motor unwinds. The motor on a deployable LM should have a worm-gear drive, and possibly a higher gear ratio.

Because the motor was incorporated for the first time in this prototype, the first version of the electrical control system was also created. A two pole double throw switch (with neutral position) shown in frame C gave the motor up and down direction. A capacitor and a diode bridge were also incorporated to convert alternating current (AC) to direct current (DC). The limit switch shown connected to the underside of the stationary plate in frame E opened the circuit when the mobile plate docked.

Almost all of the mechanics and components were incorporated into this LM. There were no more unknowns in the design and the prototype shop work was complete. The next step was to make a prototype to be deployed and that would be capable of lifting the trolley and a detector. There was, however, still some shortcoming of this prototype that would need to be improved in the deployable prototype. One of the improvements that needed to be made for a more robust overall design that can lift and align over 70_{lbf}, is a stronger sleeve and supporting structure. Another improvement is to be the addition of another new component called the toggle bone, which protects the apex of the sleeve's contour from the ball bearing on the plug. It will also be necessary to use a motor that won't unwind by the weight of trolley and detector when the motor stops.

3.3.5 Deployable Trolley-ABDS Lifting Mechanism

Finally, the capstone LM in the series of seven LMs has been completed. This robust LM is rated to lift 90_{lbf} and is the first prototype cable of lifting and controlling the entire weight of the trolley and ABDS reliably. Its motor can easily control the ABDS weight and the sleeve can easily control the inertia as the alignment phase takes place. The entire prototype is made of aluminum and Delrin with ABS brackets and steel fasteners. Weather shielding has been manufactured and can be mounted onto the LM to make it rain proof. This LM's motor can actually lift over 200_{lbf}.

Figure 3.5 shows the Deployable Trolley-ABDS LM. Figure 3.6 shows six consecutive images of the rotational alignment phase for the LM.

Figure 3.5, Deployable Trolley-ABDS Lifting Mechanism

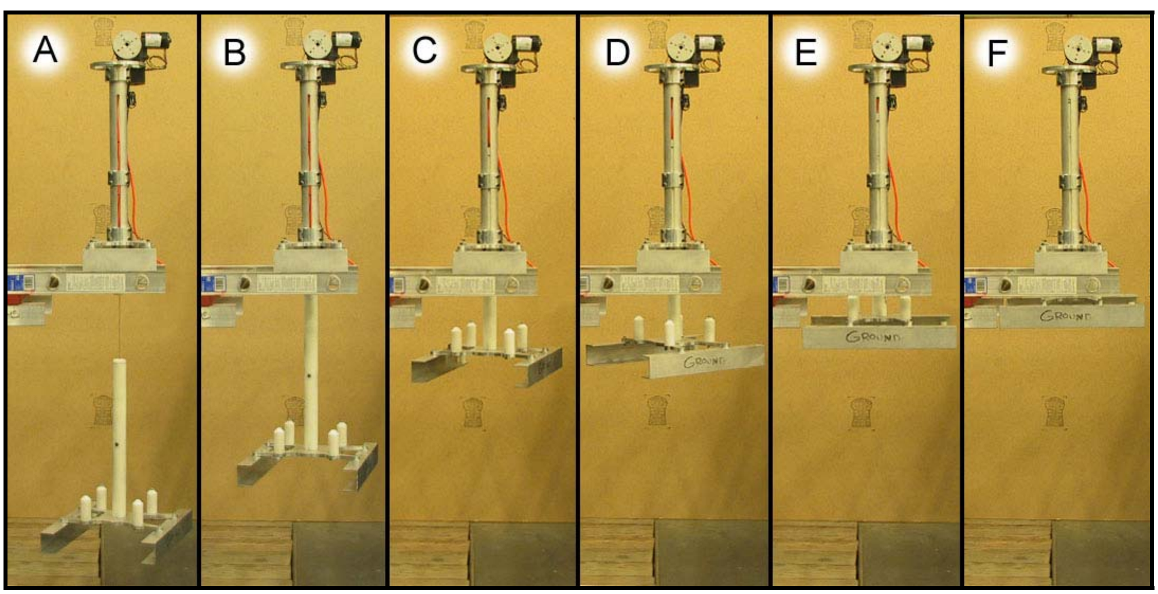


Figure 3.6, Alignement Action

CHAPTER 4

MECHANICAL COMPONENTS OF THE LIFTING MECHANISM

This section explains in detail the geometry, material choices, and interaction of the major mechanical components for the Deployable ABDS-Trolley LM. The unique features and novel solutions to problems with Duane’s concept will also be explained Figure 4.1 should be used as a tool and referenced for part names and locations while reading this chapter. The deployable LM is shown in two instances in this figure. The image on the right shows the LM sitting on the ground and the image on the left shows the LM mounted to a cantilevered piece of the collapsible truss. Additional pictures throughout this document will illustrate individual component details.

4.1 Plug

Frame A of Figure 4.2 is a good illustration showing the plug connected to the MPA. Frame B is a closer view of the top of the plug; it is pushing the slider into the sleeve. The amount of weight the LM must rotationally align determines how long the plug must be; the heavier the weight being lifted, the longer the plug. All the previous prototypes had plugs between 2" and 12" long and were only able to rotationally align between 5 $_{lbf}$ and 25$_{lbf}$, respectively. The plug for the deployable LM was made from a 23" long piece of 2" diameter Delrin round-stock. On the most recent of the previous prototypes the nominal 2" diameter dimension was chosen because it is a common size of round stock materials and easy to obtain. The 2"-dimension continued into the deployable prototypes for additional reasons - all of which are outlined in Table 4.1.

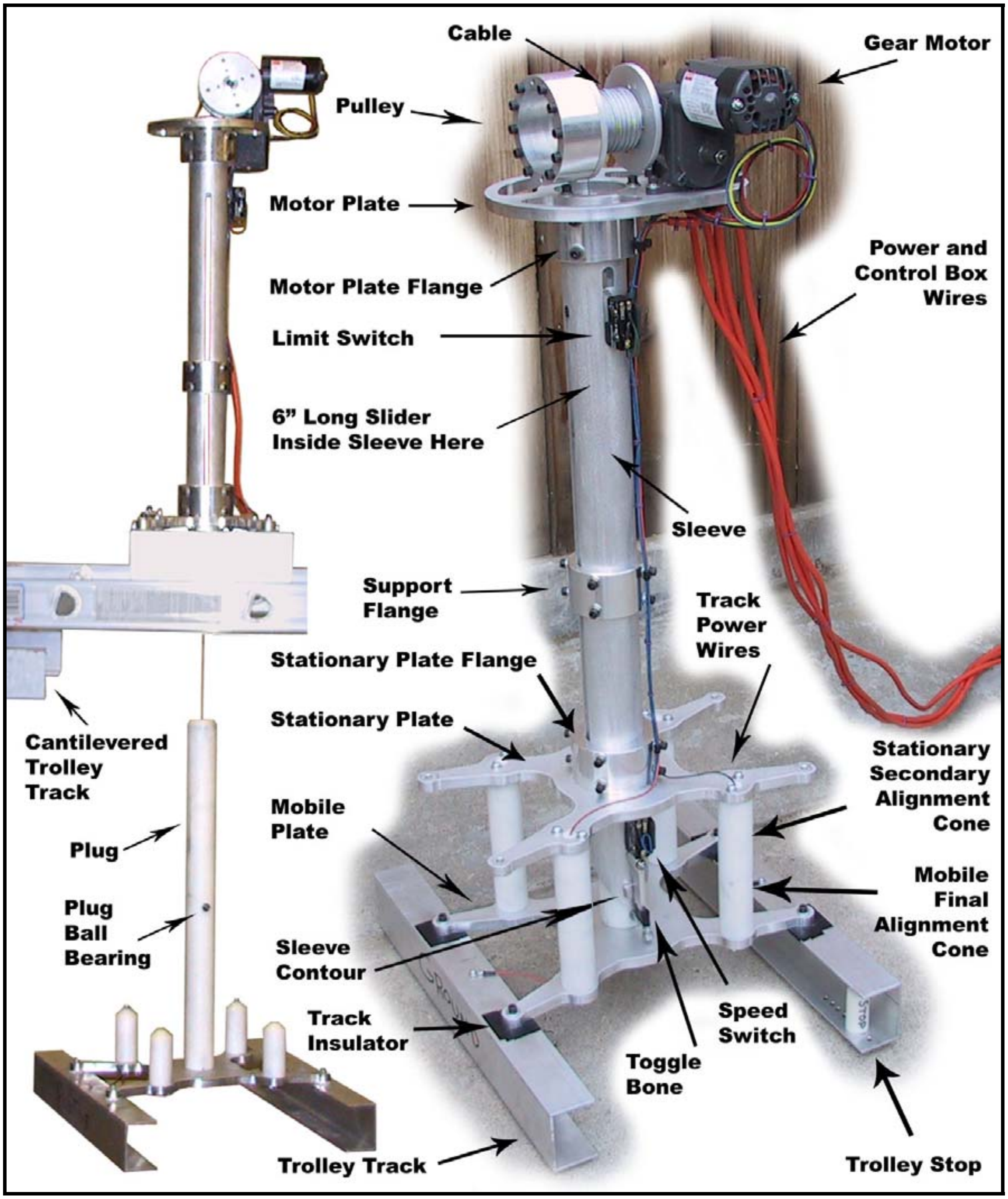


Figure 4.1, Lifting Mechanism Components Labeled

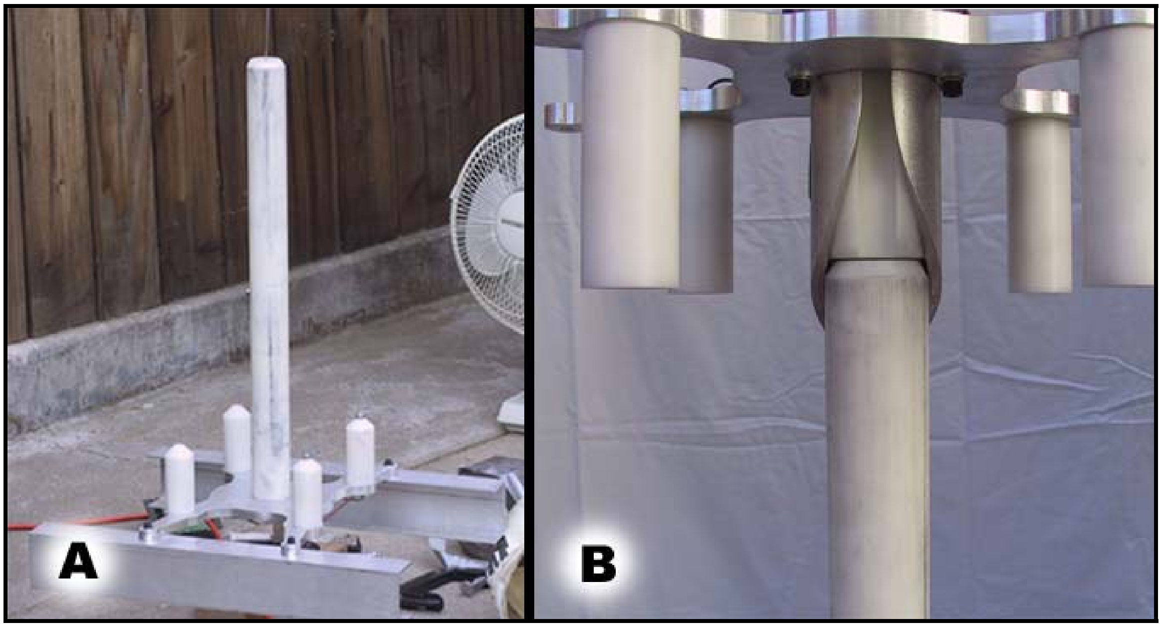


Figure 4.2, Plug

TABLE 4.1, 2" NOMINAL DIMENSION DIAMETER CRITERIA

1	Common size and easy to source.
2	Large enough to create a well defined contour on the corresponding 2" ID sleeve.
3	The 1" diameter stabilization spring can fit inside the plug.
4	¼-20 threads can be tapped into the bottom of the plug, around the stabilization spring hole, to attach the mobile plate.
5	A sleeve with this diameter is very rigid and can act as a back-bone for the entire LM.

A total of five holes are drilled in the bottom end of the plug. Four are located around the perimeter of the plug and taped with $^1/_4$-20 threads; these are used to connect the plug to the mobile plate. The fifth hole, which is a 1" diameter counter-bore 14" deep houses the stabilization spring. At the bottom of the counter-bore is a $^1/_8$" diameter thru-hole that extends the rest of the way through the plug. On the side of the plug is a 1" deep taped hole to attach the free-spinning ball bearing.

At the top end of the plug is a $^1/_4$" X $^3/_8$" chamfer, which accommodates a slight misalignment between the plug and the sleeve before the plug slides into the sleeve’s contour. The misalignment that must be accommodated for comes from three clearance fits that were necessary in the design of the LM. The first clearance fit is the 0.020" diametrical clearance between the slider's outside diameter and sleeve's inside diameter. The second and third clearance fits are the $^{1}/_{8}$" holes the cable passes through at the top of the plug and also at the bottom of the slider. The three clearance fits add up to a 0.145" possible misalignment, the $^{1}/_{4}$"-dimension is more than double the possible axial misalignment and will ensure the plug slides into the sleeve's contour and doesn't get jammed on the toggle bone.

The trolley LM has the longest plug of all LMs designed thus far. The length is determined from three primary criteria: the mass being lifted, the height of the sleeve's contour, and the height of the trolley track. Describing how the total plug length was determined will be broken into two parts; the length above and the length below the plug's free-spinning ball bearing.

The determination of the length above the free-spinning ball bearing will be described first (see Figure 4.3). Before any rotational alignment can begin the plug needs to be up inside the sleeve, above the contour and transition curve by a certain distance. This distance is to be long enough to give the plug torsional bearing against the sleeve's wall when the plug's free-spinning ball bearing begins to turn the plug. If there is not enough of the plug above the contour rubbing against the sleeve's inside wall, the plug's free-spinning ball bearing pushing against the contour can cause the plug's axis to skew with the sleeve's axis. This skewing is a result from the necessary diametrical clearance fit between the plug and sleeve. As shown by previous prototypes, the skewing can cause the plug to jam when the sleeve's contour digs into the side of the plug. To align 70_{lbf}, the minimum length, which was determined by experiment, the plug must be engaged with the sleeve above the contour and transition curve before the plug's free-spinning ball-bearing makes any contact with the sleeve or toggle bone is 4". As the mass increases or the faster the plug is moving upwards, the distance the plug needs to be inside the sleeve before any rotational alignment begins increases.

In addition to the 4" of plug length above the contour and transition curve, there is also the plug length to extend past the transition curve, contour, and toggle bone. Also, adding in other lengths such as the $^3/_8$" of chamfer height, and half the diameter of the plug's free-spinning ball bearing, the hole for the free-spinning ball bearing needs to be drilled 11-$^5/_8$" from the top of the plug. The values of Figure 4.3 are summarized in the top half of Table 4.2.

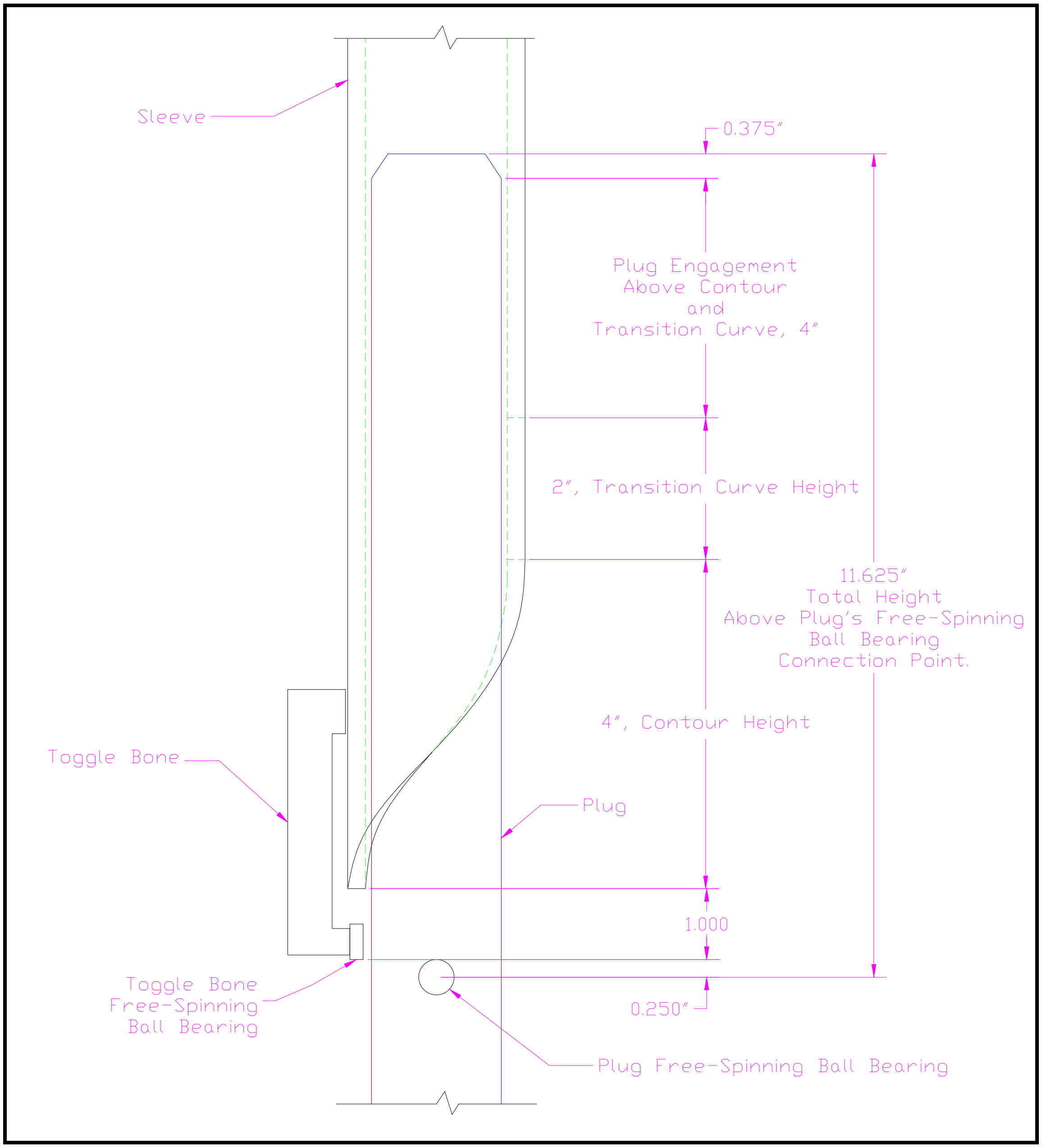


Figure 4.3, Plug Engagement Before Rotation

When the plug's free-spinning ball bearing is just below the toggle bone's free-spinning ball bearing (as shown in Figure 4.3), the height the plug's free-spinning ball bearing needs to travel up to complete the alignment will determine the rest of the plug's total length. The bottom half of the total plug length will be determined next. To rotationally align the LM's tracks with the tracks across the road, the plug's free-spinning ball bearing must travel up 1" to the apex height, 4" to the top of the contour, 2" to the top of the transition curve, and up the 4" tall slot. The bottom half of the plug's free-spinning ball bearing, which is $^{1}/_{4}$", also contributes to the height below the plug's ball bearing. Notice that the plug's free-spinning ball bearing does not quite reach the top of the slot in Figure 4.5 as it is not designed to be the component that stops the mobile plate from moving up vertically; this job is for the final alignment cones. An additional $^{1}/_{8}$" was added to the slot length so the plug's free-spinning ball bearing would not bottom out on the slot. The farthest the plug's free-spinning ball bearing can travel up into its slot on the sleeve is shown in Frame C of Figure 4.5. The lengths to create the plug length below the free-spinning ball bearing is summarized in the bottom half of Table 4.2. Thus far, the plug has a minimum length above the plug's free-spinning ball bearing of 11-$^{3}/_{4}$", and below is 11-$^{1}/_{4}$", which totals a 23"-long plug.

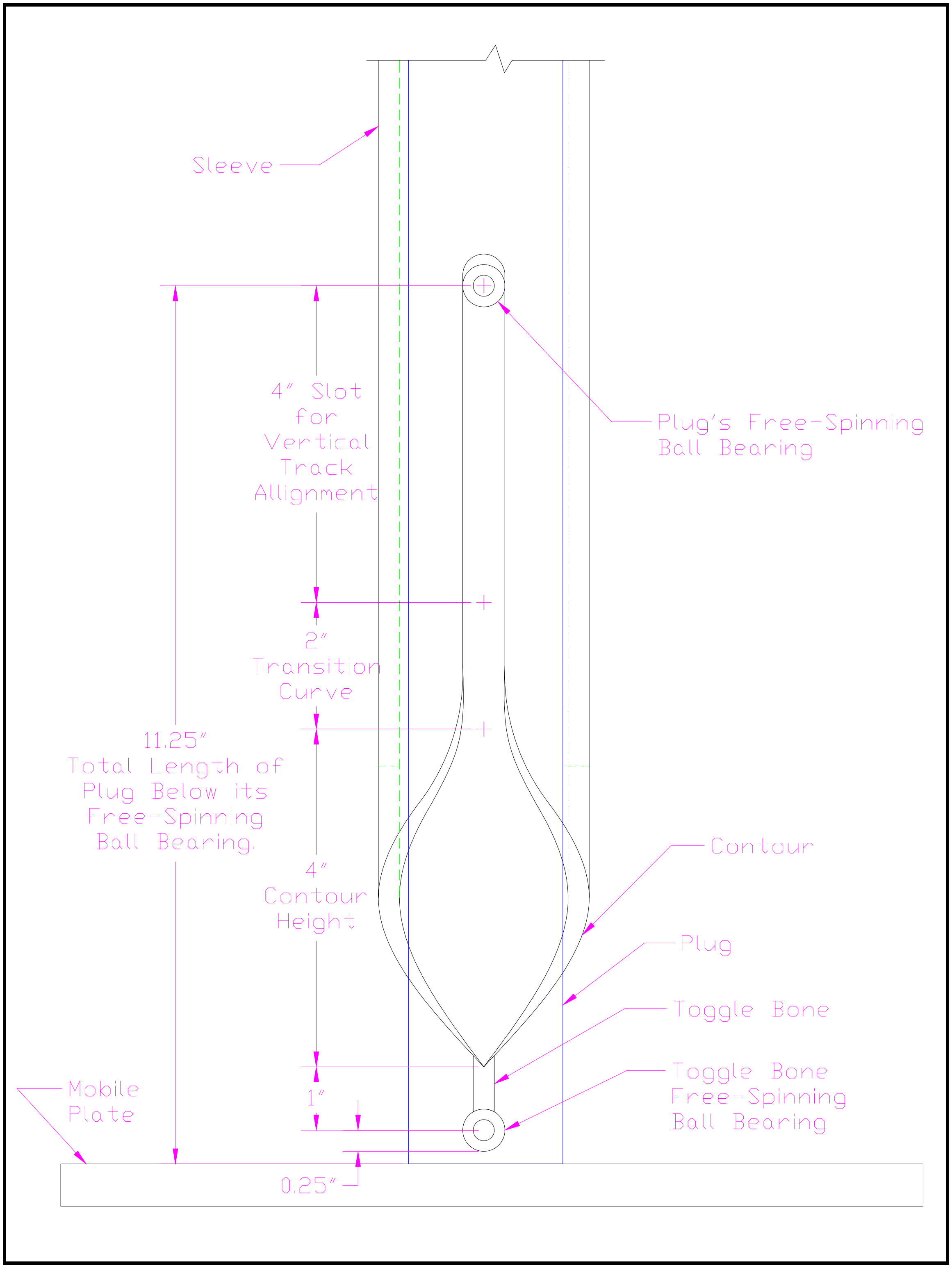


Figure 4.4, PLug Length Below the Plug's Free-Spinning Ball BEaring

TABLE 4.2, PLUG LENGTH CRITERIA

Plug Length Above Free-Spinning Ball Bearing		
1	$^3/_8$"	$^1/_4$" x $^3/_8$" chamfer around top of plug.
2	4"	Minimum plug engagement into sleeve above contour and transition
3	2"	Transition curve height between contour and sleeve.
4	4"	Contour height.
5	1"	Distance toggle bone's free-spinning ball bearing hangs below contour's
6	$^1/_4$"	Half of the plug's $^1/_2$" diameter free-spinning ball bearing.
7	$^1/_8$"	Plug bearing can not bottom out at the top of the sleeve's slot.
11-$^3/_4$"		Total length above Free-Spinning Ball Bearing
Plug Length Below Free-Spinning Ball Bearing		
1	1"	Distance toggle bone's free-spinning ball bearing hangs below contour's
2	4"	Contour height.
3	2"	Transition curve height between contour and sleeve.
4	4"	Length of 0.502" wide slot.
5	$^1/_4$"	Half of the plug's $^1/_2$" diameter free-spinning ball bearing.
11-$^1/_4$"		Total length below Free-Spinning Ball Bearing

4.2 Free-Spinning Ball Bearing on the Plug

The plug's free-spinning ball bearing replaces the pin described in Duane's proposal for the LM. This ball bearing performs the rotational alignment by rolling along the sleeve's contour as the plug is pulled into the sleeve. Previous prototypes showed that it is essential to have a ball bearing used in place of the steel pin to reduce the friction as the rotational alignment occurs. Frame A of Figure 4.5 shows two close-up views of the free-spinning ball bearing. Notice how the ball bearing is offset away from the plug with a 0.025" thick shim-washer. This washer has been placed between the ball bearing and plug to keep the ball bearing's outer race from rubbing against the plug.

Frame B shows the plug's free-spinning ball bearing rolling off the toggle bone's free-spinning ball bearing. Frame D shows the free-spinning ball bearing rolling along the transition curve just before it enters into the 4" long slot. Frame C shows the plug's free-spinning ball bearing at the top of the slot when the mobile plate is docked. The ball bearing is attached with a flat head machine screw with a head diameter of 0.003" larger than the ball bearing's inner diameter. This results in a low profile grip as the machine screw holds the ball bearing onto the plug.

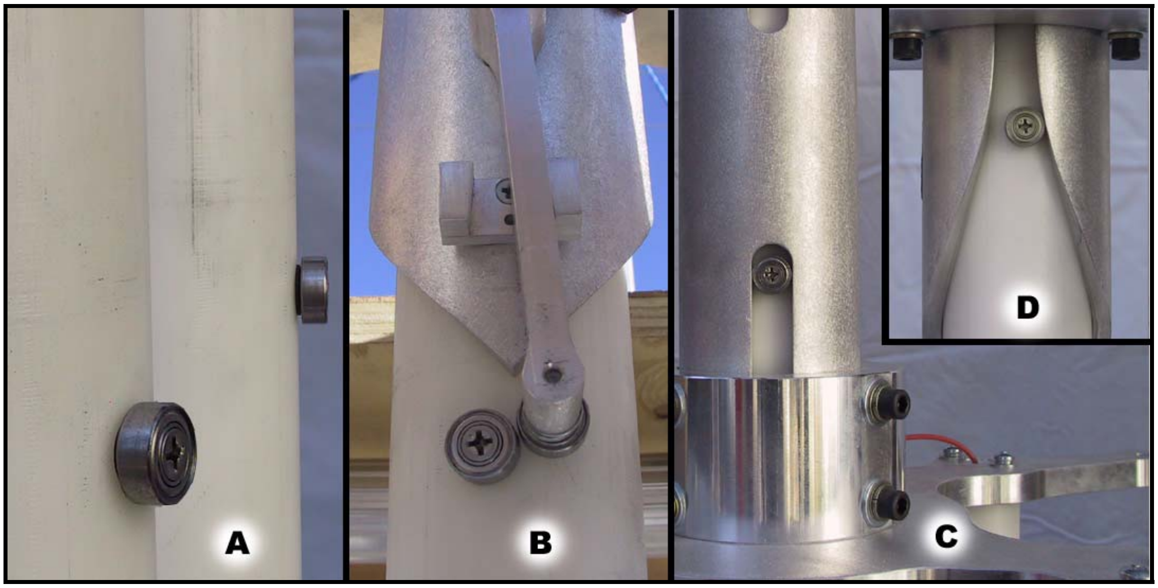


Figure 4.5, Free-Spinning Ball Bearing on the Plug

4.3 Sleeve

The sleeve performs two primary functions for the LM; it acts as the backbone for the entire apparatus, and its contour performs the rotational alignment of the MPA with the plug's free-spinning ball bearing. As the backbone for the entire LM, the sleeve must support the weight of the mass being lifted and be accommodating for attachment of other components such the flanges and control switches. The sleeve has a 2" inner-diameter to fit around the plug. The sleeve is shown in Figure 4.6.

Figure 4.6, Sleeve

The $^{1}/_{4}$" wall thickness was chosen for four reasons. First, $^{1}/_{4}$" was enough thickness to tap four (4) full threads for a $^{1}/_{4}$-20 SAE machine screw and attach the flanges. Second, The load path of the trolley and detector being lifted goes from the pulley through the sleeve to the static plate where the LM is mounted to its supporting structure. With all the slots in the sleeve it loses its closed cross-section along a majority of its length, and the sleeve is not as rigid as it was before. The $^{1}/_{4}$" wall thickness makes the sleeve rigid enough that it doesn't collapse on the plug and slider under loading. Third, when the plug's free-spinning ball bearing runs into the sleeve's contour and the mass being lifted starts to rotate, a lot of torque is applied to the sleeve's contour. Previous prototypes failed to work properly when 20_{lbf} was rotationally aligned because the inertia of the mass deformed the sleeve enough that the secondary alignment components completely missed and did not even have a chance to engage and finalize the rotational alignment. The $^{1}/_{4}$" wall thickness has made this prototype very robust and able to control over 200_{lbf} without the sleeve deforming. Lastly, the $^{1}/_{4}$"-thick walled tubing is easily available and reasonably priced.

The 31-$^{1}/_{2}$" sleeve length was determined from the following four criteria. The sleeve needs to be long enough to fit the slider and the plug inside. Above the slider, there needs to be room for a slot so the limit switch lever can extend through the wall of the sleeve to make contact with the slider to trigger the motor to stop. At the top of the sleeve, just above the limit switch slot, 2" is needed to connect the motor plate flange. These criteria are summarized in Table 4.3.

TABLE 4.3, SLEEVE LENGTH CRITERIA

1	22"	Plug length. (1" is not inside the sleeve)
2	6"	Slider length
3	1-$^{1}/_{2}$"	Limit switch slot.
4	2"	Connection for motor flange.

The sleeve's contour is by far the most unique looking feature of the LM. The plug's free-spinning ball bearing rolls along this contour to perform the rotational alignment of the plug and mobile plate. The single hole with a counter-bore near the contour's apex is for connecting the toggle bone, which will assist in a guaranteed rotational alignment. There are also three sets of hole patterns on the sleeve for connecting the static, motor, and stabilization flanges. These holes are tapped with $^{1}/_{4}$-20 threads. The sleeve also has a total of six slots cut in it. One slot is an extension of the contour. Two of the slots are for the limit and speed control switches. Two more slots that run almost the entire length of the sleeve limit the slider's travel. The sixth slot was for an idea that never needed to be implemented.

The contour at the bottom of the sleeve performs the primary alignment of the plug and mobile plate. As described earlier, the First Prototype had problems because its sleeve was created by cutting a tube in half at an angle, which left two flat spots on the contour as seen from the pin's (now free-spinning ball bearing) reference frame. A contour with a constant slope as seen from the reference frame of the plug's free-spinning ball bearing was the solution to eliminating the flat spots like the ones on the First Prototype's sleeve. This contour geometry, which was used on the deployable LM, was created in a manner similar to how one would begin folding a paper airplane. Start by folding an 8-$^{1}/_{2}$" x 11" piece of paper in half, then open it back up and fold in two corners to the fold line as shown in frame A of Figure 4.7. Wrap the paper around a cylinder and trace the contour.

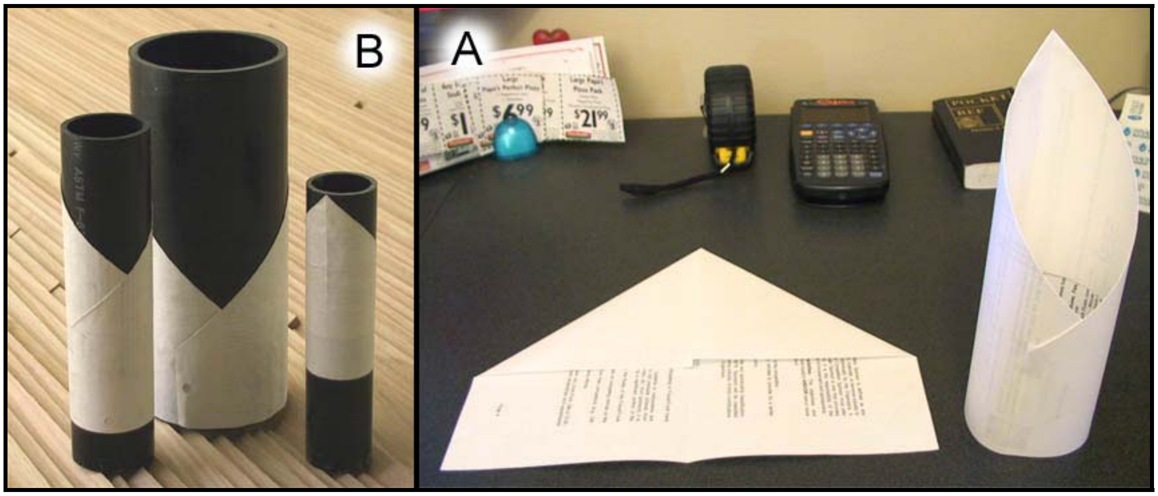


Figure 4.7, Contour Geometry

A few folded papers were wrapped around different sized tubes as shown if frame B. The vertical length the contour occupies along the tube axis was recorded and plotted against the outside diameter of the tube. Figure 4.8 is the plot, which shows that a 2-$^{1}/_{2}$" OD tube with a contour angle of 45° has a height of 4".

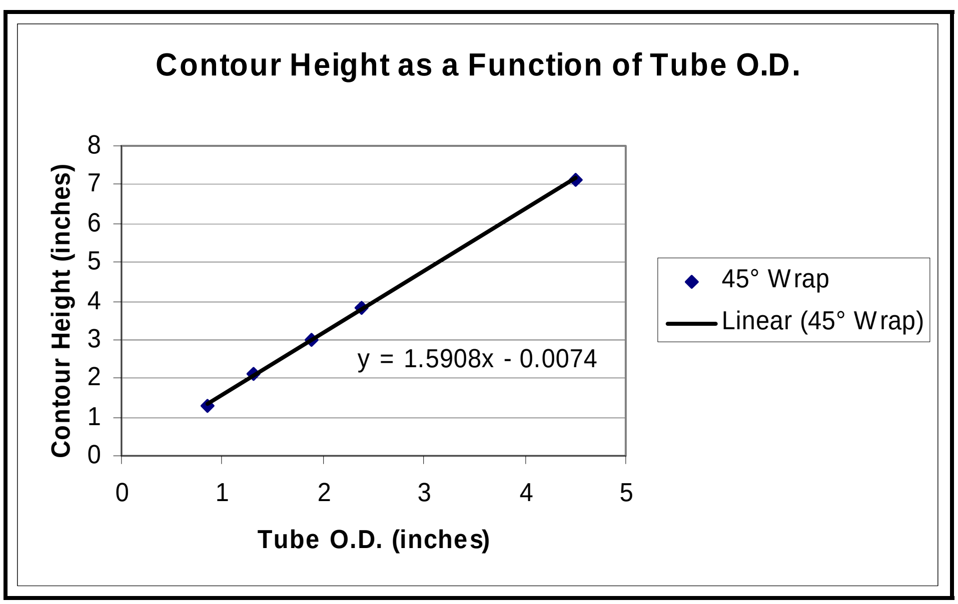


Figure 4.8, Contour Wrap Equation

The first attempt to make the sleeve's alignment contour angle with this new method was at 45° and worked just fine. However, if the slope were steeper the LM would be taller and the alignment phase could happen faster. If the contour was shorter the LM would have to lift the trolley slower during the alignment phase. The 45° is the middle angle between 0° and 90° and seemed like a good starting point for the first deployable prototype. So, 45° it was because optimization was not in the scope of this project.

After the plug's free-spinning ball bearing rolls along the contour, it rolls along a 4" long slot, which keeps the MPA from rotating while the tracks become coincident with the tracks across the highway. The length of 4" was chosen because the tracks are 3" tall. The additional 1" was added for the plug's free-spinning ball bearing to stabilize in the slot before the top plane of the MPA tracks rise above the bottom plane of the stationary tracks. The slot was oversized 0.002" greater than the diameter of the plug's free-spinning ball bearing so the bearing would travel freely up and down the slot. This clearance fit was kept tight because the more the plug is allowed to rotate inside the sleeve, the more the mobile trolley tracks can rotate, and this can cause the mobile tracks to get caught up with the stationary tracks.

There is also a transition region between the contour and the slot for the plug's free-spinning ball bearing. The location of this transition is specified in Figure 4.3. Although it may not be obvious, this transition curve can be seen in Figure 4.6. Notice that as the plugs free-spinning ball bearing rolls along either side of the contour, it will roll smoothly into the 0.502"-wide, 4"-long slot. Previous prototypes, which did not have a transition curve, jerked the trolley and LM around a lot as the plug's free-spinning ball bearing was channeled into the slot abruptly.

The sleeve was the most challenging component of the LM to make because of its size and unique contour feature. There was barely enough travel on the mill to cut the long slots for the slider without needing to redesign the LM for length. As shown in Figure 4.9, two indexing chucks were used to hold and rotate the sleeve to the correct angle for cutting or drilling the corresponding holes or slots. The vise in the middle of the table minimized the amount of vibration from the cutter by stabilizing the sleeve at mid-span.

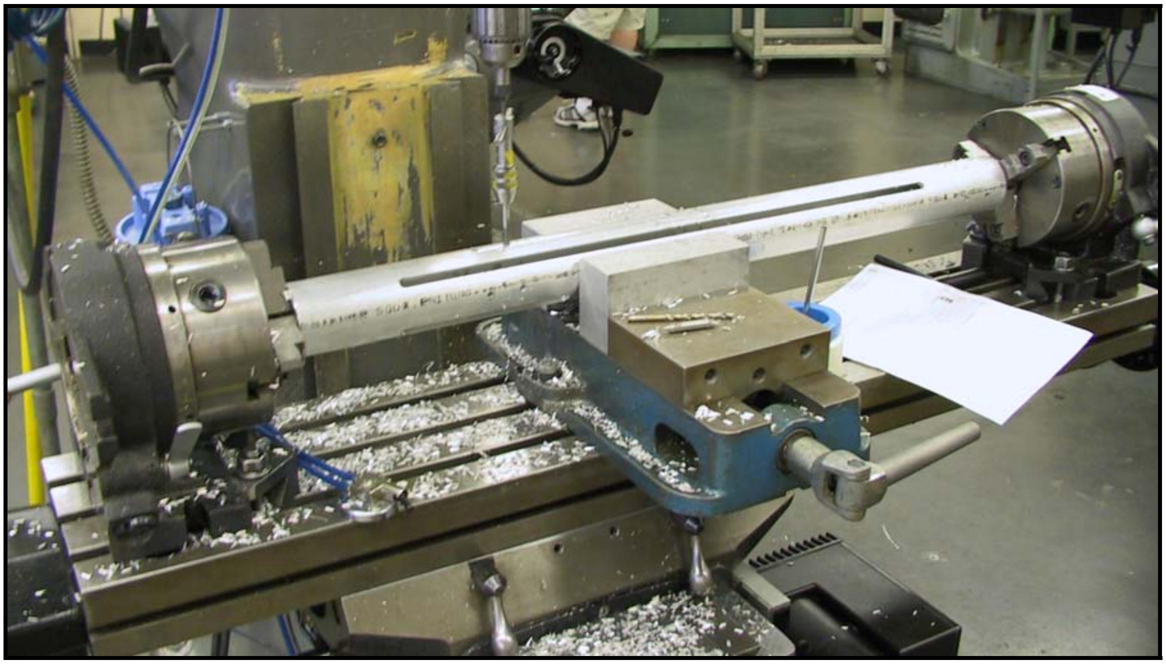


Figure 4.9, Manufacturing the Sleeve

The sleeve CAD drawing is drawn in a unique manner because it is shown in a flat plane with length dimensions in inches and the dimensions around the cylinder (sleeve) are in degrees. By dimensioning the sleeve in this manner, the entire sleeve can be drawn, dimensioned, manufactured, and understood from the single 2-dimensional drawing. By dimensioning the slots and holes around the sleeve in degrees, the drawing can be applied to any combination of diameter and wall-thickness sleeve. If the drawing were printed out to scale it could be wrapped around the corresponding cylinder and the drawing would represent the components of the sleeve.

4.4 Slider

During the development of the LM, it was realized that the cable almost never hangs directly down the center of the sleeve. Instead, the mobile plate usually sways back and forth from the wind, or has some other undesirable movement or natural frequency, which results in the cable moving around inside the sleeve. When the MPA is going up and the plug is about to go into the sleeve, the probability of the plug sliding into the sleeve without guidance was not very good with previous prototypes. A component called the slider was designed and incorporated to hold the cable concentric with the center axis of the sleeve and guarantees the plug will begin sliding into the sleeve. Figure 4.10 is a CAD drawing showing the possible misalignment if the slider is not used, and the guaranteed alignment when the slider is used. Figure 4.11 is a close-up picture of the LM with and without the slider incorporated, showing the same possible miss-alignment as Figure 4.10.

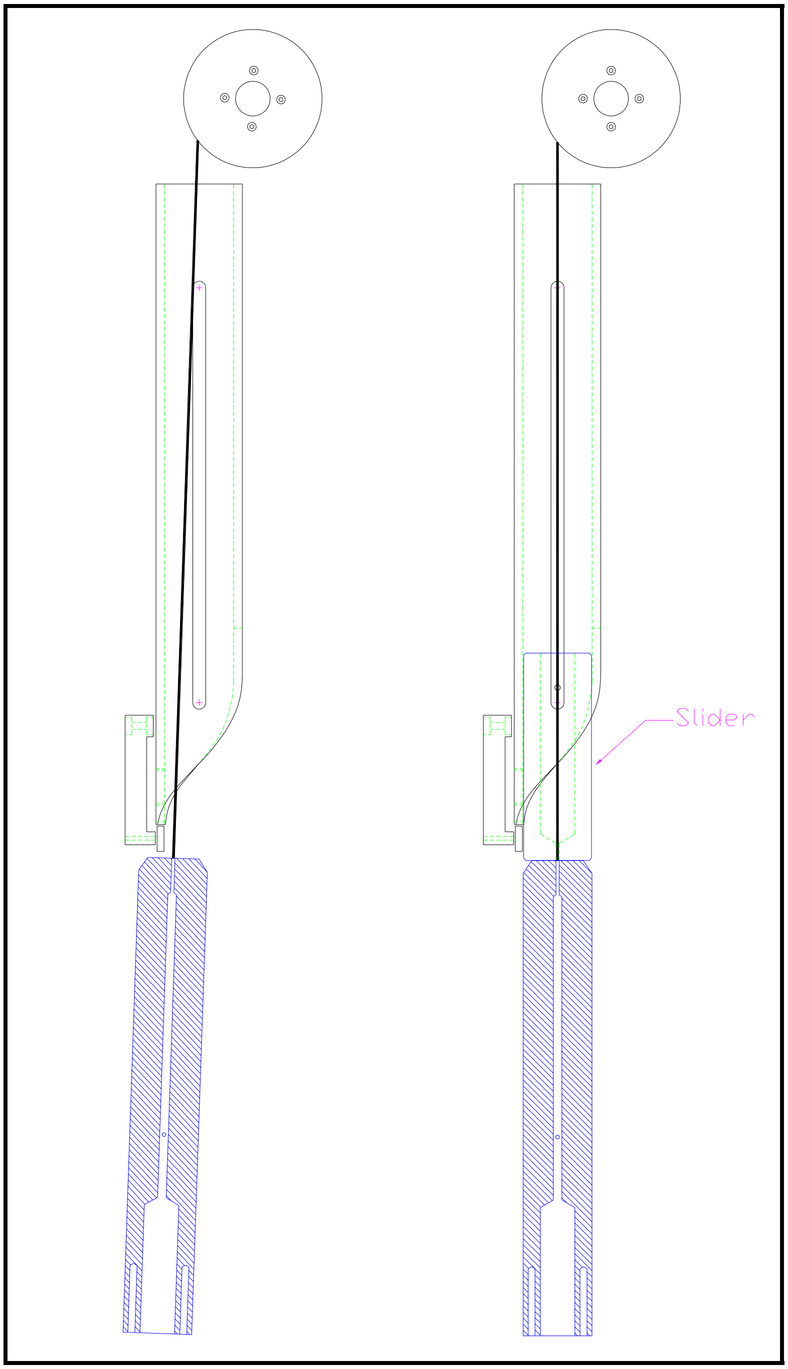


Figure 4.10, Slider Necessity Schematic

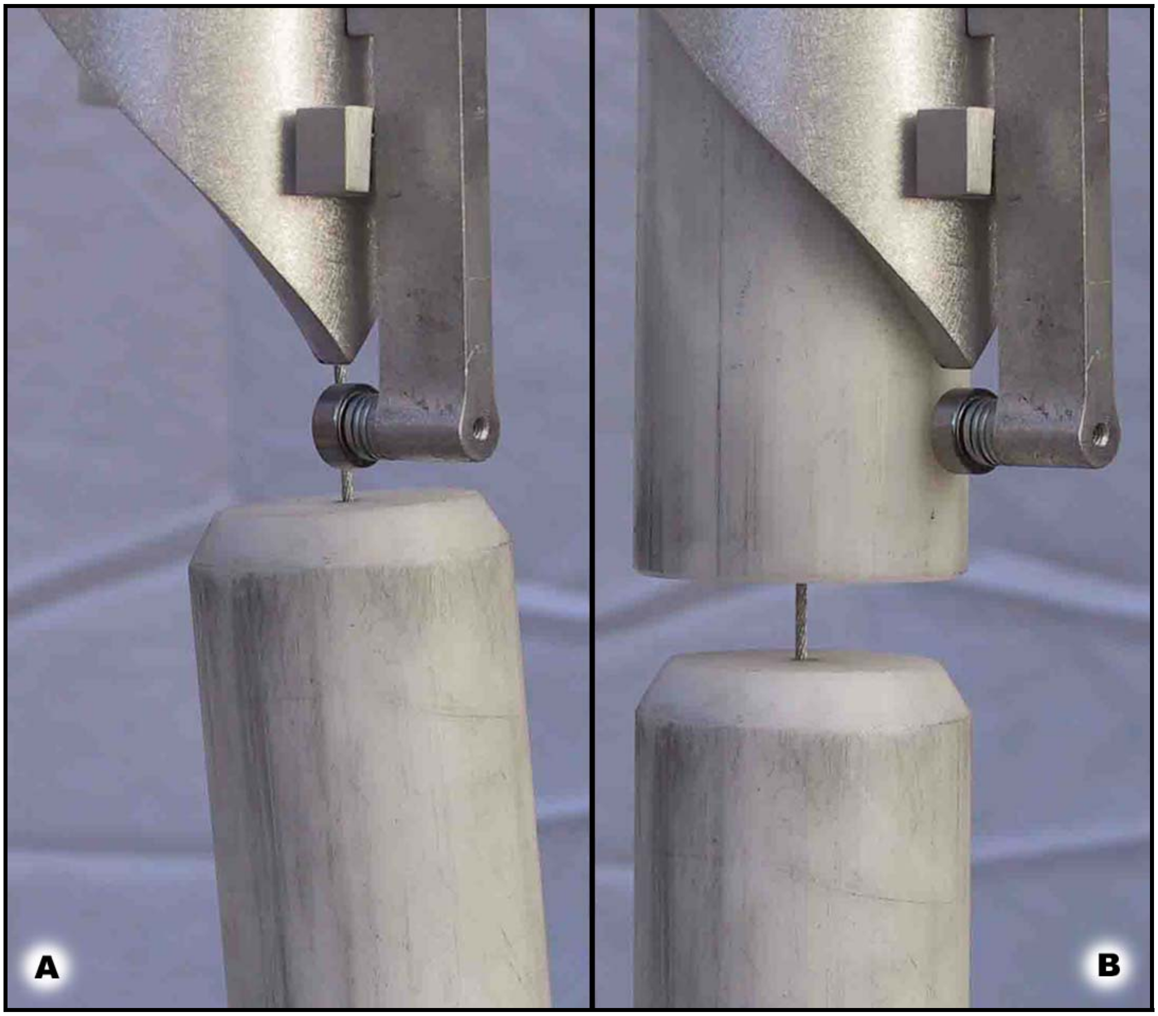


Figure 4.11, Slider Necessity

The slider by itself is shown in Figure 4.12. When the plug is inside of the sleeve, the slider rests atop of the plug and moves up and down with it. When the plug completely exits the sleeve, lowering the detector to the ground, the slider waits at the bottom of the sleeve for the plug to return. The slider's vertical travel is limited by the two pins sticking out the side of the slider. These pins slide along two slots, which almost go the total length of the sleeve. Figure 4.6 shows the sleeve: the long slot visible is for one of the slider's pins, the other slots are for the control switch apparatus, which will be discussed in later sections. The slots stop at the bottom of the sleeve in a location so the bottom of slider hangs just below the toggle bone. The slots continue high enough on the sleeve that they do not impose on the limit switch-slider interaction to turn off the raising motion. It is essential that the slider moves freely inside the sleeve with no possibility of jamming. The slider must also have a low friction, tight clearance fit with the inner wall of the sleeve, and also where the cable runs through the center of it. A 0.020" diametric clearance fit was used between the slider and sleeve and between the slider and the cable. The chamfer at the top of the plug (see Figure 4.2 or Figure 4.11) accommodates these diametric clearance fits between the slider and the cable and sleeve by guiding the plug into the sleeve even if it is still axially misaligned by about $^{1}/_{4}$".

In order for the slider to work properly, and not wobble or become jammed inside the sleeve's contour when the MPA swings in the wind, it must always have part of its body up inside the sleeve, above the contour. This length was experimentally determined to be 1" minimum. At the same time the slider must extend below contour's apex and toggle bone, to hold the cable's fulcrum point stationary for the plug. If the contour is 4" tall and the toggle bone hangs about 1" below the contour's apex, the slider must be a minimum of 6" long.

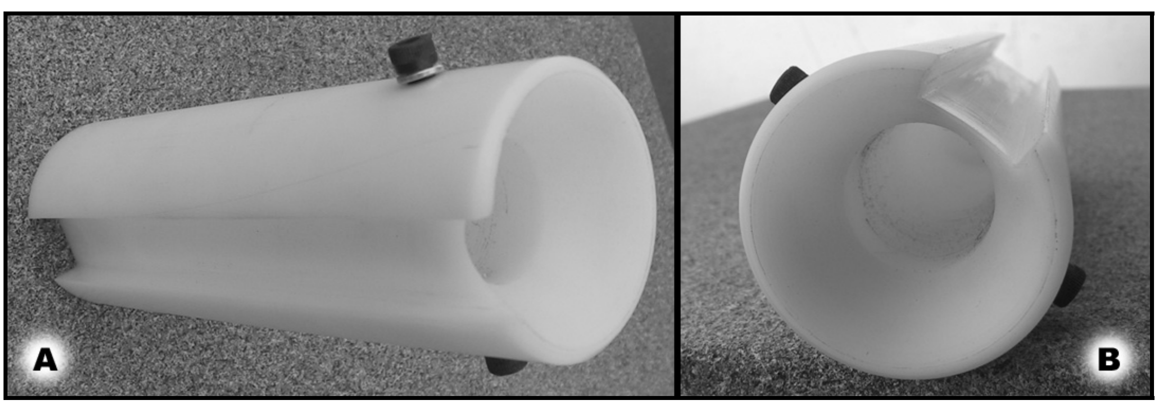


Figure 4.12, Slider

When the slider is near the top of the sleeve it comes within a few inches of the pulley. A 1" counter bore, 5" deep, was drilled into the top of the slider to minimize the cable and motor strain created by the misalignment that will often occur as a result of the cable rolling around the entire width of the pulley. Figure 4.13 shows how skewing between the cables position on the pulley and the center axis of the slider is compensated for inside the 1" counter bore of the slider. An alternative to drilling out the top of the slider could have been extending the length of the sleeve a few inches but would have increased the total height of the LM, and resulted in the entire apparatus being heavier. The large dado along the side of the slider was a machined feature to accommodate a potential concern that never arose.

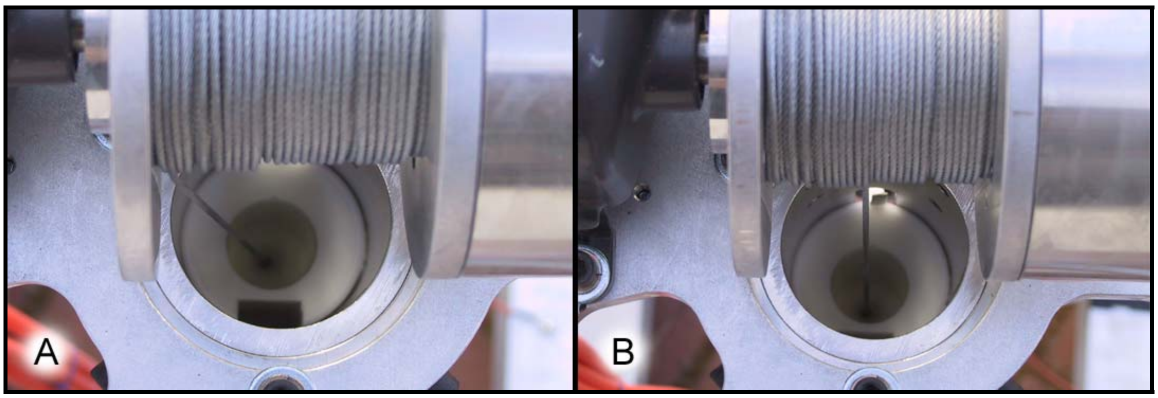


Figure 4.13, Slider Counterbore

In summary, with the slider incorporated, the fulcrum point of the cable is lowered to the bottom of the LM. Even if the plug and mobile plate are swaying from the wind, the slider will assure that the top of the plug is aligned to go into the sleeve. When the plug slides back into the sleeve, it simply begins pushing the slider up into the sleeve.

4.5 Support Flange

The support flange keeps the sleeve walls from deflecting under loading. On previous LMs the slots along the sleeve for the slider's pins were not as long as on this LM and did not affect the function of the sleeve. If the support flange is not used on this LM, the sleeve walls will deflect and squeeze the slider and not allow it to move freely inside the sleeve. It is essential for the sleeve walls to stay rigid because the LM will fail during the lifting process if the plug gets jammed up inside the sleeve and is not at the bottom of the sleeve holding the cable in place when the plug is ready to slide inside the sleeve. Figure 4.14 shows the support flange.

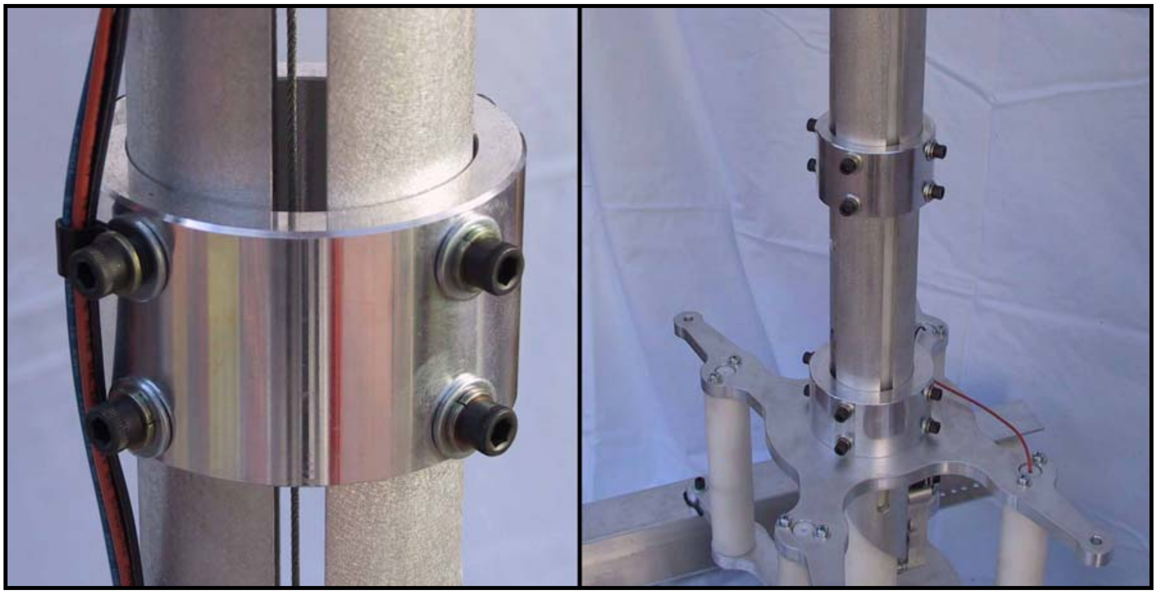


Figure 4.14, Support Flange

4.6 Plate Flanges

The plate flanges connect the stationary and motor plates to the sleeve. The flanges are identical. Each flange thickness is $^5/_8$" to accommodate the three $^1/_4$" tapped holes, which connecting it to the motor and stationary plates. Figure 4.15 shows a flange connecting the motor plate to the sleeve, and Figure 4.16 shows a flange connecting the stationary plate to the sleeve. A four bolt-hole pattern (2 holes every 90°) was used to connect the flange because the sleeve also needed to have slots every 90° for the slider and control switches. The bolt-hole pattern on the sleeve, to connect each flange was rotated 45° from the slot pattern.

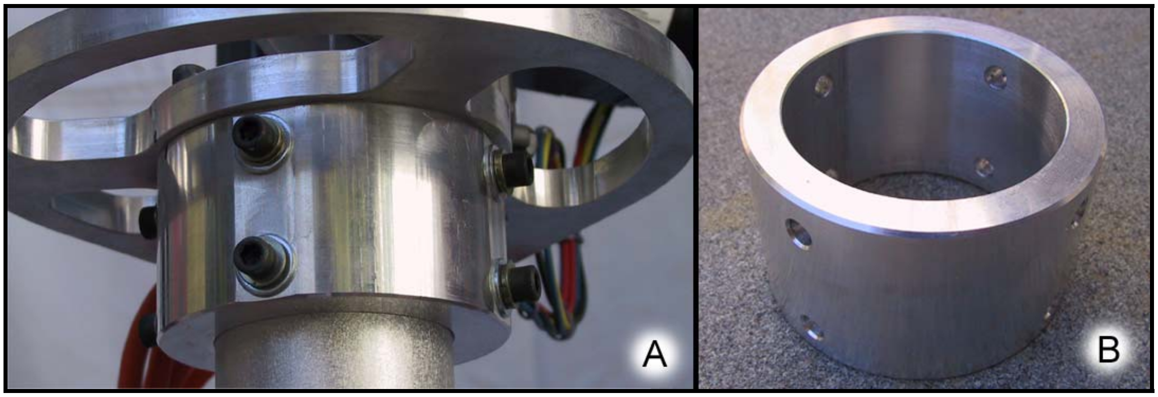


Figure 4.15, Flange Connecting Motor Plate to Sleeve

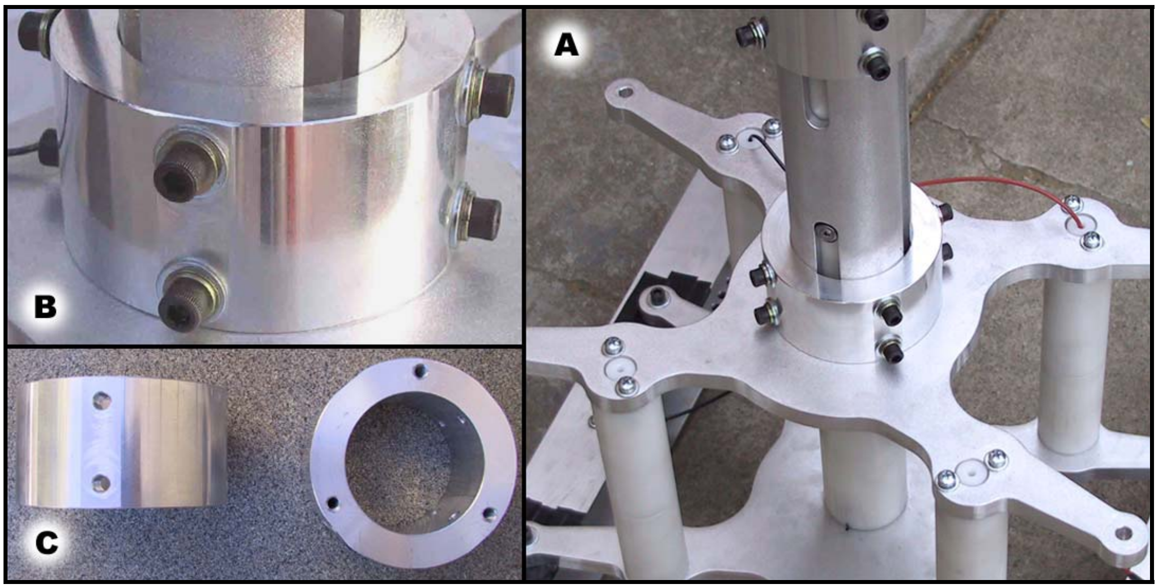


Figure 4.16, Flange Connecting Stationary Plate to Sleeve

4.7 Plates

The $^{1}/_{2}$" thick aluminum plate was used so the thickness resists the bending as a result of the plate being in the load path between the mass being lifted and the LM's support structure. The pockets were cut out to minimize weigh. The stylistic contours are considered industrial design and not a direct input to functionality. They do, however, minimize the weight of the component while providing unique connection points for other components such as the final alignment cones.

4.7.1 Motor Plate

The motor plate is used to mount the motor to the LM, and also functions as a place to attach some of the rain shielding, which will be discussed later. The motor plate's asymmetrical geometry is designed to position the edge of the pulley's drum, so the cable rolls off, and is somewhat aligned with the center axis of the sleeve.

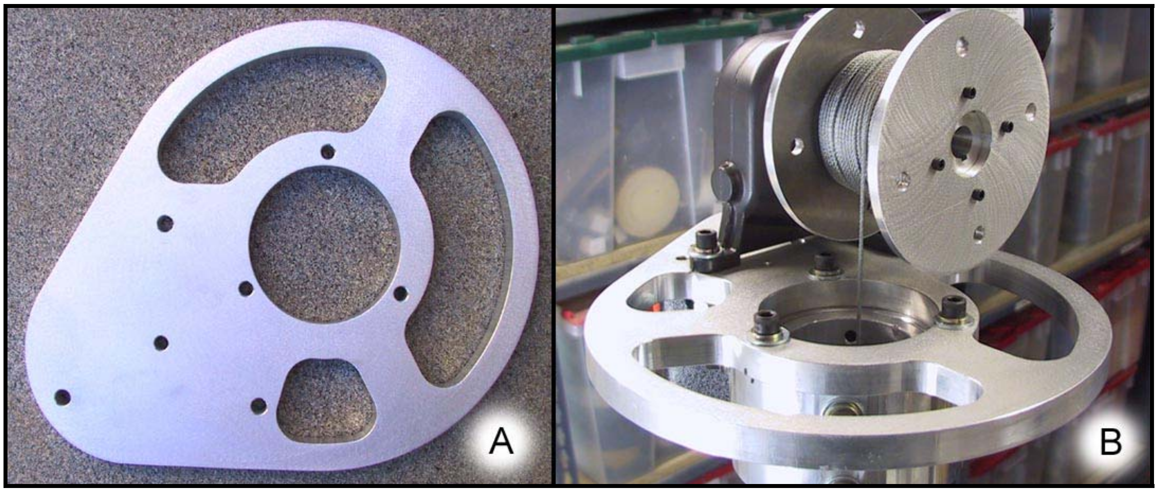


Figure 4.17, Motor Plate

4.7.2 Stationary Plate

The stationary plate has a single hole at the end of each arm to connect the LM to the mobile truss or any other structure. A final alignment cone is connected to the midpoint of each stationary plate arm. The hole-pattern in the center of the stationary plate allows it to fit around the sleeve and connect to a flange.

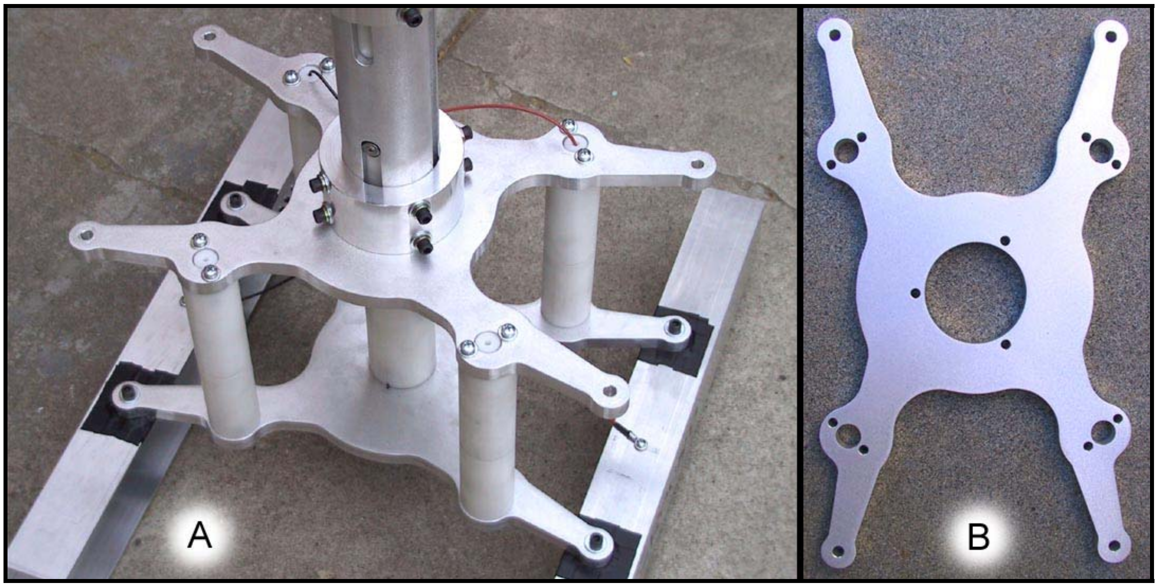


Figure 4.18, Stationary Plate

4.7.3 Mobile Plate

The mobile plate is almost identical to the stationary plate. The only difference between the two plates is the hole-pattern in the middle. The mobile plate uses its center hole-pattern to connect to the plug. The mobile plate connects to its final alignment cones in the same way as the stationary plate. The holes at the end of the arms of the mobile plate are used to connect the tracks.

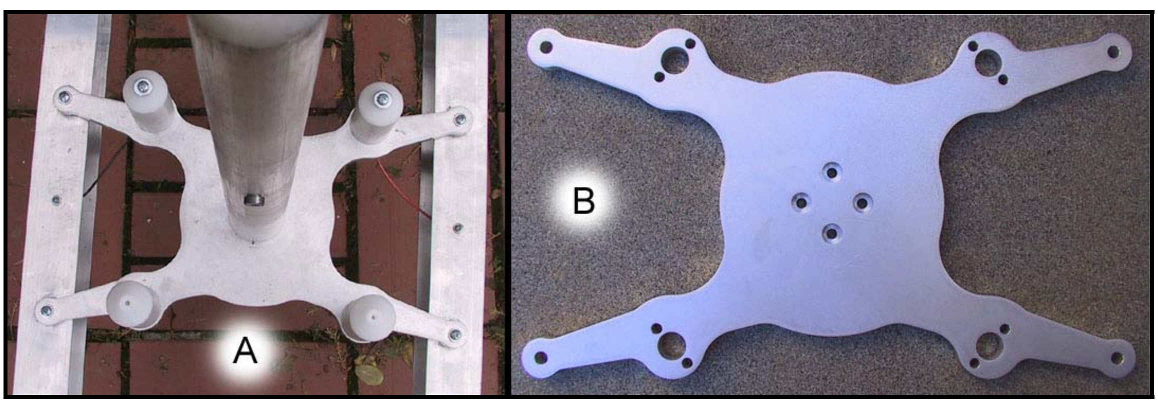


Figure 4.19, Mobile Plate

4.8 Toggle Bone

The toggle bone was the capstone component to the LM's design. It was manufactured and implemented last, and is what finally gave the LM its degree of 100% reliability. When the plug is sliding into the sleeve there is a possibility of the plug's free-spinning ball bearing running into the apex of the sleeve's contour. When this happened in previous prototypes, the plug's center axis skewed with the sleeve's center axis and jammed. Sometimes it would pop to one side or the other of the apex, but other times the motor would just come to a stop. The more the DC motor slowed down the more it pulled on the plug and the more jammed it became.

Attempts were made on previous prototypes to divert the plug's free-spinning ball bearing around the apex of the sleeve's contour. One attempt was to simply make the apex a very sharp point, which surprisingly didn't really improve things. A second attempt was to rigidly fix a ball bearing in front of the apex. This fixed ball bearing diverted the plug's free-spinning ball bearing more often than the sharp apex but still wasn't 100% reliable. The friction and clearance between the plug and sleeve was still a stronger influence than the two ball bearings "stacked" atop each other. The final attempt was to make the fixed ball bearing toggle back and forth a little bit. The concept was to have a design similar to three balls stacked on top of each other. The pivot of the toggle bone would be the first ball. The free-spinning ball bearing on the end of the toggle bone would represent the second ball. And the free-spinning ball bearing on the plug would represent the third ball.

With the toggle bone in place, the combination of the plug's free-spinning ball bearing, toggle bone's free-spinning ball bearing, and the two ball bearings at the toggle bone's pivot, the apex of the sleeve's contour is no longer a problem. Figure 4.20 shows five frames in the motion of an effective toggle bone.

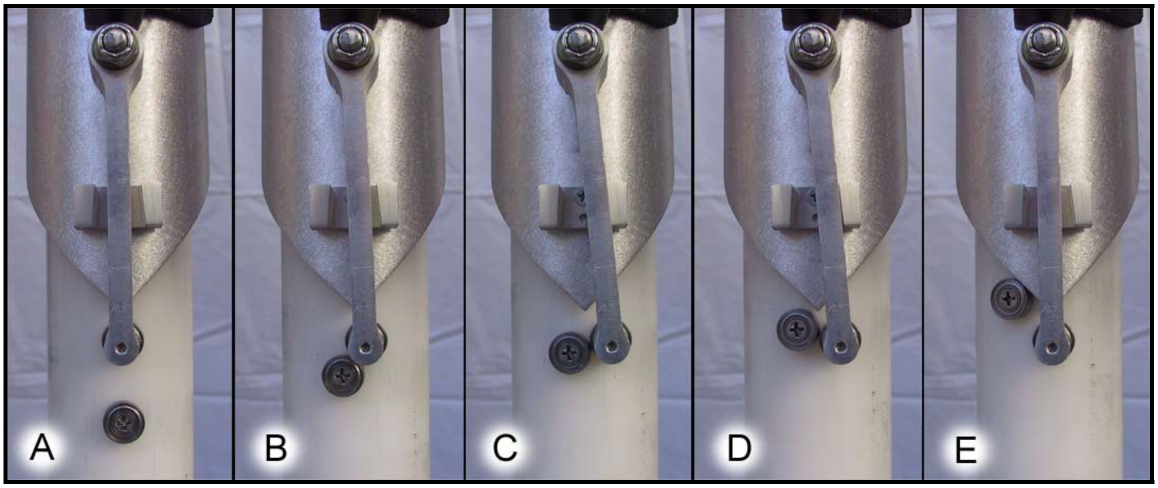


Figure 4.20, Toggle Bone Action

Frame A of Figure 4.20 shows the toggle bone hanging vertically like a pendulum at rest and protecting the contour's apex. The plug's free-spinning ball bearing is moving upward and headed directly at the toggle bone's free-spinning ball bearing. In frame B the plug's free-spinning ball bearing has just run into the toggle bones free-spinning ball bearing; the toggle bone "pops" out of the way and allows the plug's free-spinning ball bearing to continue traveling upward. In frame C the toggle bone has hit the right side of the limit stop and can't move any farther out of the way. The limited movement of the toggle bone is what keeps the toggle bone's free-spinning ball bearing from ever exposing the apex. In this case the plug's free-spinning ball bearing was headed directly at the apex, and pushed off against the toggle bone's free-spinning ball bearing. Frame D shows the plug's free-spinning ball bearing after it pushed off against the toggle bone's free-spinning ball bearing. The plug's free-spinning ball bearing will run into the contour just above the apex. Frame E shows the plug's free-spinning ball bearing rolling along the contour away from the toggle bone and the toggle bone has fallen back to its vertical position of protecting the apex.

For the toggle bone to function properly, the swinging motion must be constrained. Once the toggle bone toggles to one side or the other, it must stop and not completely expose the apex to the plug's free-spinning ball bearing. Figure 4.20 shows the path of the plug's free-spinning ball bearing to begin the primary alignment. Figure 4.21 shows three frames of the limit stop; frame A is of the toggle bone hanging between the stop, frame B is a side view of the stop, and frame C shows the stop's fastener to the sleeve revealed when the toggle bone is toggled to one side.

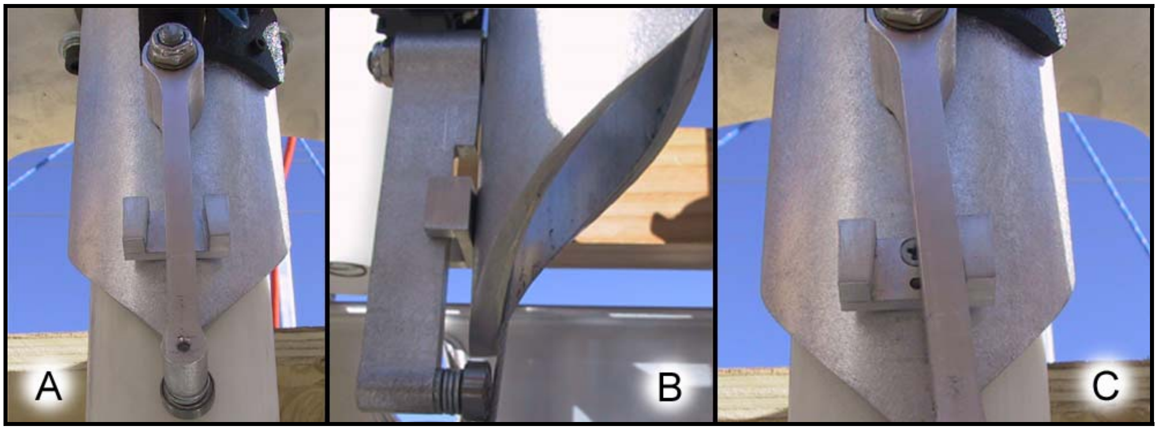


Figure 4.21, Limit Stop

4.9 Final Alignment Cones

The clearance fits between the plug and sleeve, and also between the plug's free-spinning ball bearing and the slot in the sleeve that it rides along, is essential for a smooth rotational alignment but also allows the MPA a few degrees of rotation after it is aligned. This rotation will result in a slight jog between the MPA tracks and the tracks extending across the freeway.

The final alignment cones are used to rotationally and vertically secure the MPA so it doesn't wobble when the trolley drives on and off of it. The ends of the mobile and stationary final alignment cones are both chamfered at 45° and big enough in diameter so they can engage to perform the last little bit of rotational alignment.

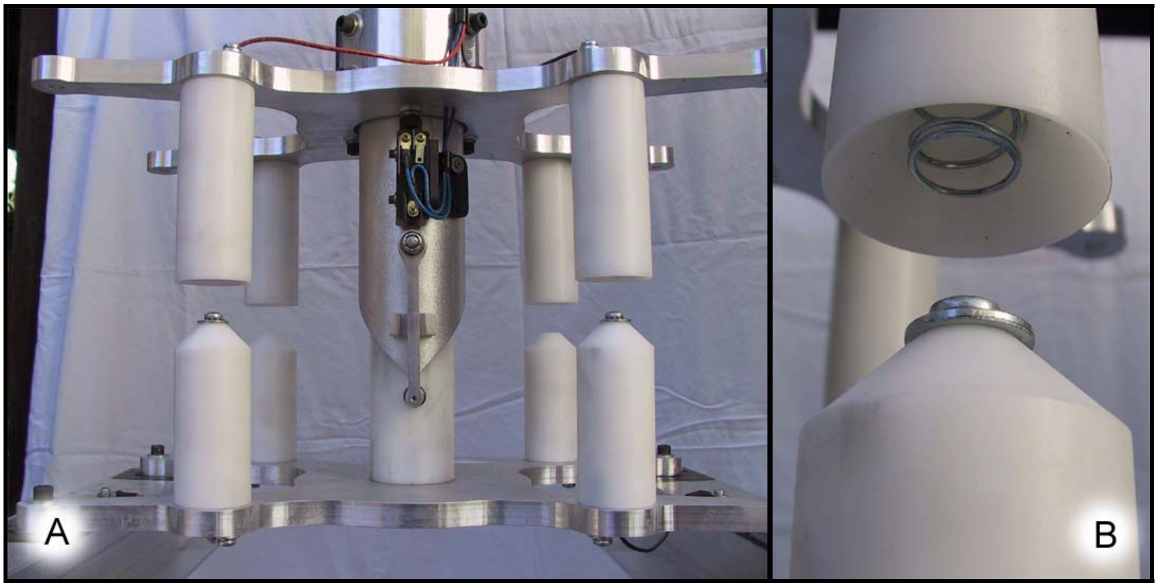


Figure 4.22, Final Alignment Cones

The cones are made of Delrin for the purpose of electrically isolating the power transmission components housed inside of them. The spring and washer of the power transmission assembly shown in frame B of Figure 4.22 , which transmits the 12 volts to the MPA to power the trolley, are discussed in section 5.3

4.10 Stabilization Spring

Another type of misalignment can happen when the trolley drives on and off the MPA as it is illustrated doing in Figure 4.23. As the trolley wheels near the edge of the MPA tracks, its center of mass (plus the weight of the detector) shifts from directly beneath the plug, and the MPA tracks drop down (over ½" with a 70_{lbf} load). This offset creates an obstacle the trolley must drive up in order to get onto the tracks extending across the freeway. The trolley was having trouble getting up this gap with its 2" diameter tires so a solution needed to be found.

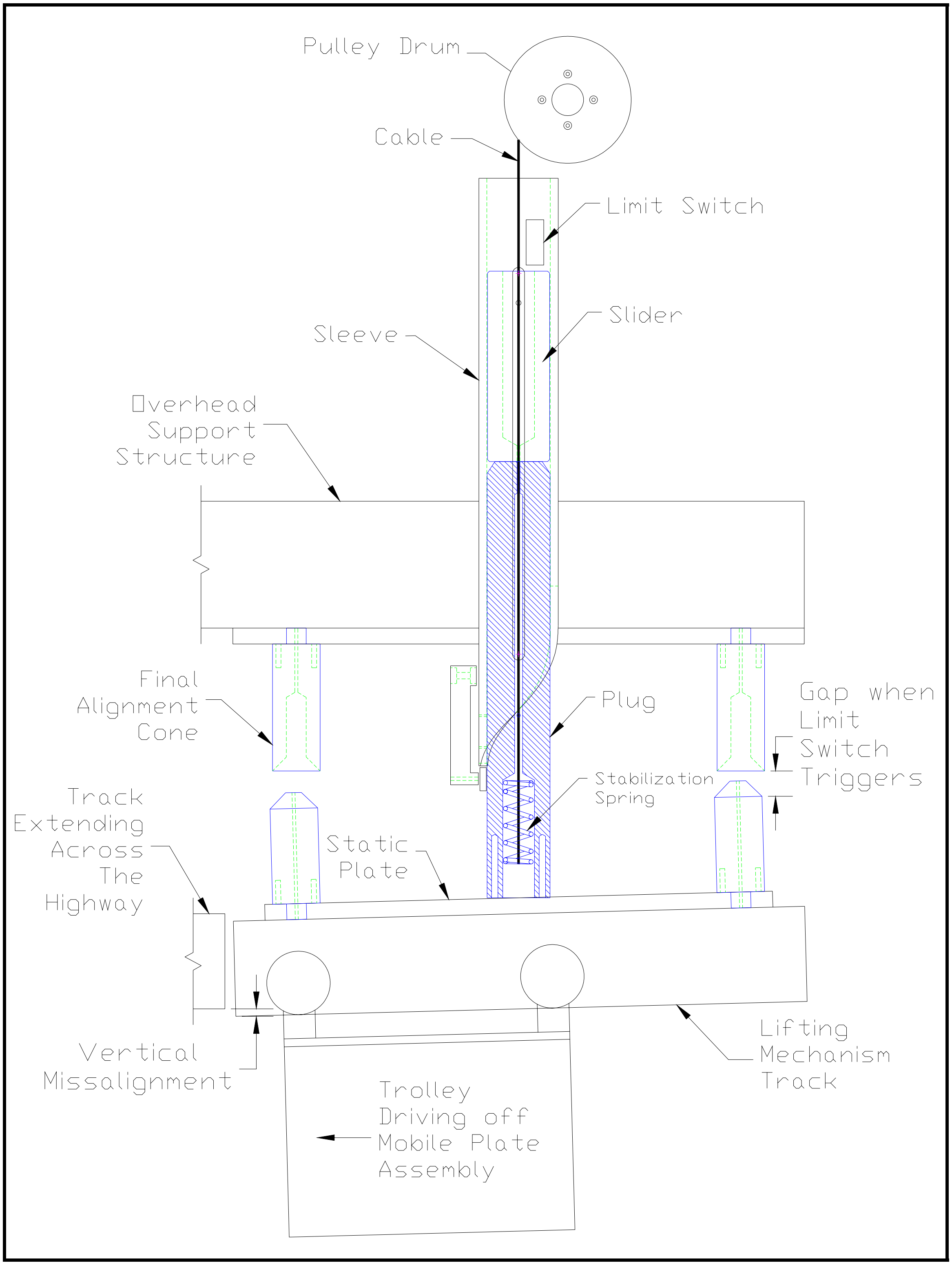


Figure 4.23, Stabilization Spring Necessity

This misalignment occurred for three reasons: the necessary, but small clearance fit between the plug and sleeve, the moment applied to the connection between the mobile plate and plug, and the lack of vertical stability from the final alignment cones. Because the manufacturing process of turning down the outside of the plug with more precision than was done would require special machine shop tooling, making the gap between the plug and sleeve tighter was not feasible. Adding additional triangulation to support the connection of the plug and mobile plate was undesirable because the ideas brainstormed increased the LM's height and number of parts significantly. The stabilization spring was introduced as a solution to this problem, and has since become an essential component of the LM.

The spring has been placed in the load path between the hanging mass (ABDS, trolley, and MPA) and the cable. In conjunction with the final alignment cylinders, the stabilization spring is used to keep the LM tracks stable when the trolley drives on and off them. After the limit switch (electrical components are discussed in section 5) is triggered, an auxiliary power circuit keeps the motor on for an additional 1 second, and pulls the final alignment cones together tightly by compressing the stabilization spring. Now, when the trolley drives on and off the MPA, the MPA tracks will not rock because a pre-load has been placed between the stationary and mobile final alignment cones.
A section view of the stabilization spring inside the plug is shown in Figure 4.24, and a picture of the stabilization spring sticking partially out of the plug is Figure 4.25.

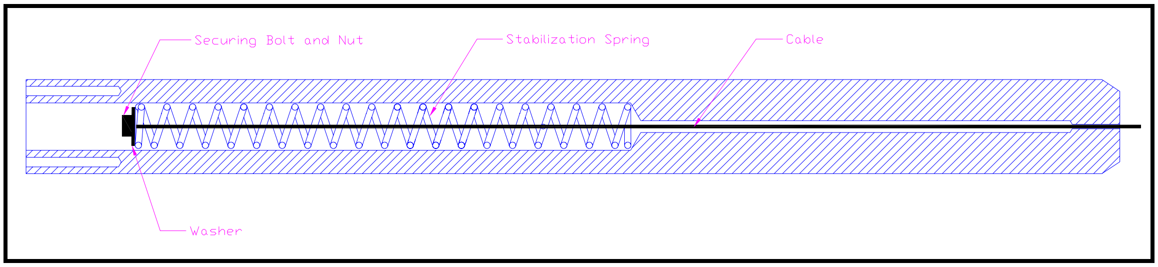


Figure 4.24, Stabilization Spring Section View

Figure 4.25, Stabilization Spring With Plug and Cable

In addition to the stabilization spring's ability to stabilize the MPA, it also absorbs the impact when the plug's free-spinning ball bearing runs into the sleeve's contour. When the plug begins sliding into the sleeve, any rotational momentum of the ABDS-trolley package disappears within a few inches of insertion. The only motion left until the plug's free-spinning ball bearing runs into the sleeve's contour, will be purely vertical. When the plug's free-spinning ball bearing does run into the sleeve's contour, it takes time for the 70_{lbf} of trolley and ABDS to change angular momentum. During this moment, the motor puts a lot of tension in the cable because it still wants to pull the plug directly upward, but the plug has to begin to rotate as its free-spinning ball bearing begins to track along the sleeve's contour. The stabilization spring has been specified so it will compress left in the stabilization spring temporarily postpones the motor's relentless task of pulling the plug upwards at a constant rate. As the motor pulls the cable up at the same speed, the spring compresses, and creates a window of time for the MPA to rotate. As soon as the MPA begins to rotate, the spring will then uncompress to its length before and be ready for the next collision.

In summary, the LM will work without the stabilization spring but not nearly as gracefully or precisely. The LM works much smoother and will have prolonged life with the stabilization spring incorporated because lighter loading will be transmitted to the motor, and throughout the entire apparatus, when the plug's free-spinning ball bearing pushes off against the sleeve's contour. With the spring in place the delay circuit can more precisely control the final tension in the cable after the MPA docks and the final alignment cylinders compress together.

4.11 Trolley Stop

When the plug slides into the sleeve it is important for it to be as vertical as possible. If the plug is skewed from the vertical axis, a lot of additional friction between the inside walls of the sleeve and outside wall of the plug will make for a rough rotational alignment. Even worse, the top of the plug may jam on the sleeve's contour, which would result in a catastrophic failure of the LM's operation.

A part called the trolley stop was created to make sure the mass of the trolley and ABDS is centered underneath the center axis of the plug. A number of holes have been drilled in the mobile track so the trolley stop can be adjusted according to the loading of the trolley and detector. Figure 4.26 shows the trolley stop and the holes for adjustment along the track. When the trolley is on the MPA, and pushed against the pre-adjusted trolley stop, its center of mass (trolley and detector) should be located directly under the plug, which will result in a vertical plug. The trolley stop also keeps the trolley from running right off the track and falling to the ground when it drives onto the MPA.

Figure 4.26, Trolley Stop.

4.12 C-Channel Insulating Bushings

The C-channel trolley track must be electrically isolated because it is energized with 12 volts, which is the trolley's power source. To isolate the short section of trolley track on the LM, white Delrin isolators were placed at each connection point of the track and the mobile plate. Each isolator consists of two custom-made Delrin parts; a bossed washer, and a mating flat washer as shown in frame A of Figure 4.27.

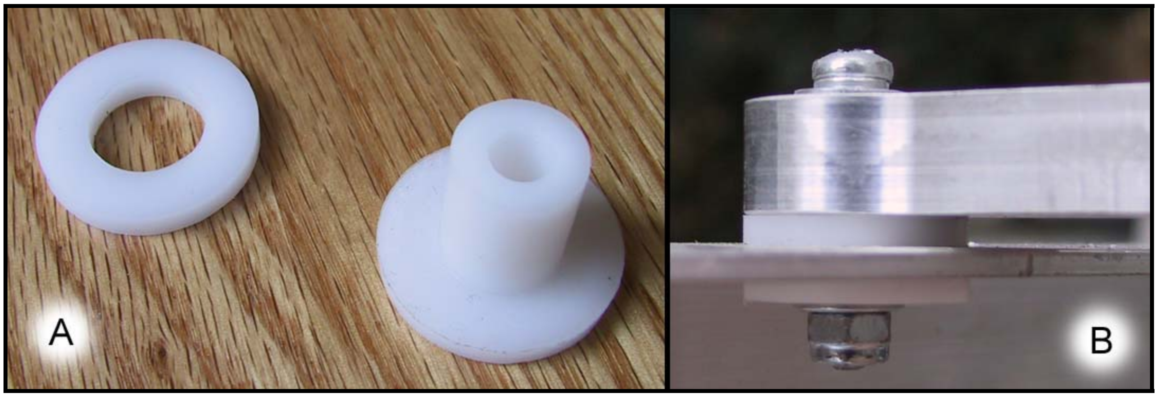


Figure 4.27, Trolley Track Insulating Bushings

The Delrin washer sits between the trolley track and mobile plate. The bossed Delrin washer then slides through the hole in the trolley track, and also through the Delrin washer to completely isolate the trolley track. When the mounting screw is used to attach the C-channel to the mobile plate there is no way for the screw to short out the isolated track pieces because of the way the two Delrin washers isolate the C-channel.

When the LM is mounted to the overhead structure for operating, its short section of track must be adjusted for perfect alignment with the tracks extending across the highway so the trolley can drive on and off with a smooth transition. Frame B shows a 10-32 SAE machine screw inserted through the Delrin isolator and the trolley track connecting it to the mobile plate. The holes Delrin bossed washers are oversized compared to the size of the machine screw to enable the fine-tuning of the alignment between the mobile and stationary track.

4.13 Pulley-Drum Cover

When IEL staff other than the designer used the LM, they did not know tension needed to be kept on the cable to keep it from unwinding and becoming tangled around the pulley and motor shaft. Unlike a nylon rope, which is very flexible and will stay wrapped around a pulley drum without being held in position, a steel stranded cable does not naturally stay in a bent position around a pulley drum; it will tend to try and spring back into a straight position resulting in a tangled rat's nest of cable. When IEL staff were loading and unloading the trolley, they would sometimes hold onto the MPA or placed it on a cart, which supported its weight instead of letting it continue to dangle, which kept the tension in the cable. A cover was heat-formed from ABS plastic and wrapped around the cable and pulley drum. Frame B of Figure 4.28 shows the cover off the LM. Frame A shows the cover installed. When the cable is slacked from below, the cover will hold it in place around the drum and the possibility of it becoming loose and tangled was eliminated.

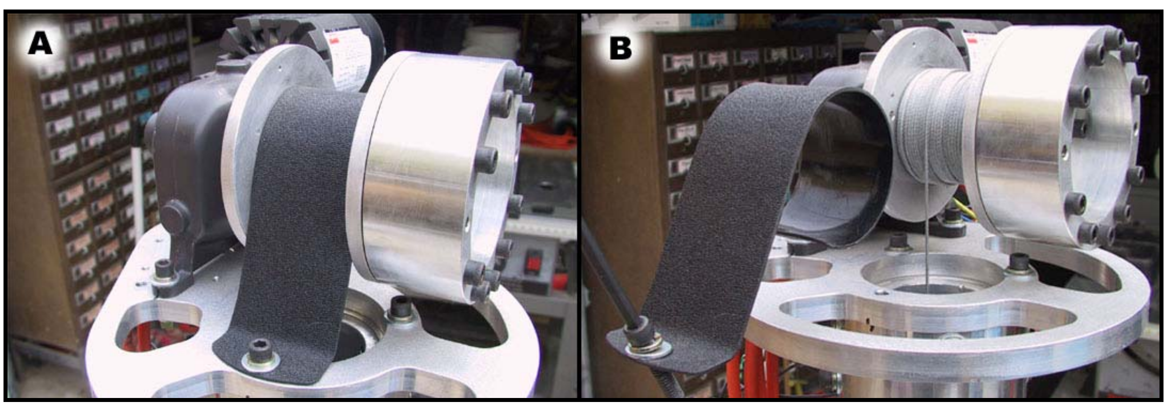


Figure 4.28, Pulley-Drum Cover

CHAPTER 5

ELECTRICAL COMPONENTS OF THE LIFTING MECHANISM

5.1 Speed Control Switch

The speed switch changes the speed at which the motor raises the mobile plate. After the plug runs into the bottom of the slider it begins to push the slider up, into the inside of the sleeve. The slider moves up about $^{1}/_{2}$" and triggers the speed switch. The speed switch triggers a relay, which, in turn, switches the motor to a lower voltage power source. This secondary voltage needs to be available because the speed during rotational alignment typically needs to be much slower, especially when lifting the ABDS-trolley package. The slower the ABDS-trolley package is pulled up during the rotational alignment, the gentler the LM turns the ABDS-trolley package. The speed switch is located just below the stationary plate and is shown in Figure 5.1.

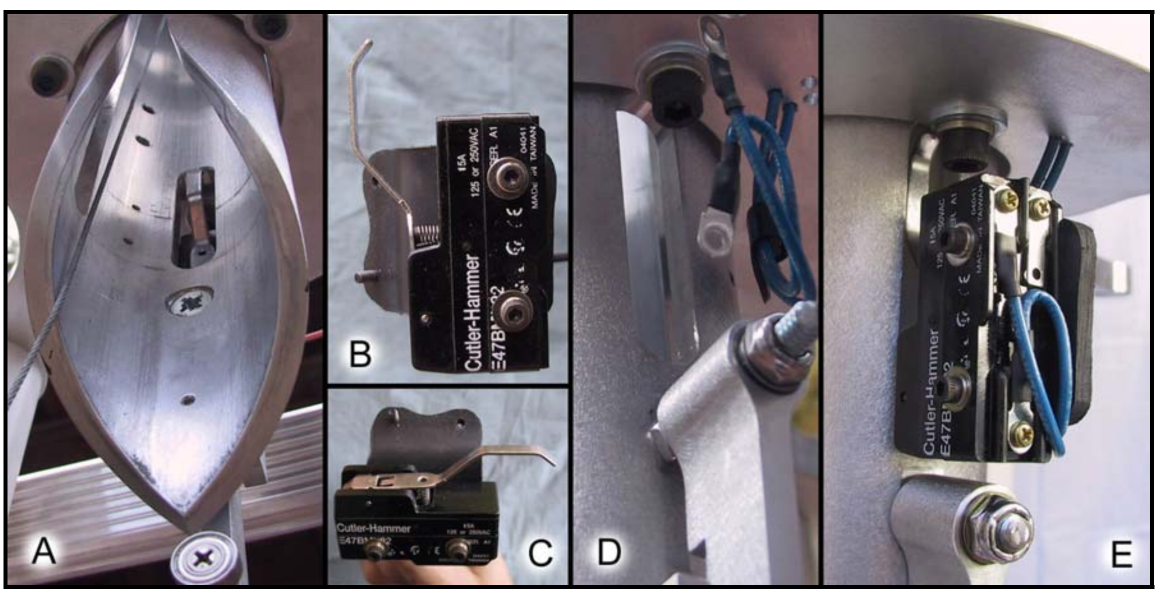


Figure 5.1, Speed Control Switch

Frame A of Figure 5.1 shows the inside view of the slot and the switch lever extending through the slot. Frame B and C are two views of the switch and ABS bracket dismounted from the sleeve. Frame D shows the outside view of the slot in frame A. Frame E illustrates the switch installed with wire connected to it. The geometry of the lever arm is essential for proper operation. The arm was custom bent to have a 1" flat spot, which the slider and plug make contact with. The flat spot enables the switch to stay triggered when the adjacent chamfers at the bottom of the slider and top of the plug pass by. If the contact point was a roller or just the end of the lever the motor would speed up momentarily during the alignment phase.

Limit Switch

When the mobile final alignment cones are $^{1}/_{8}$" from mating to the stationary final alignment cones, the slider triggers the limit switch. The limit switch tells the delay circuit to turn off the relay after a preset amount of time, which opens the circuit to the motor. Figure 5.2 shows three frames of the limit switch. Frame A is of the switch removed from the sleeve, frames B and C are of the switch installed.

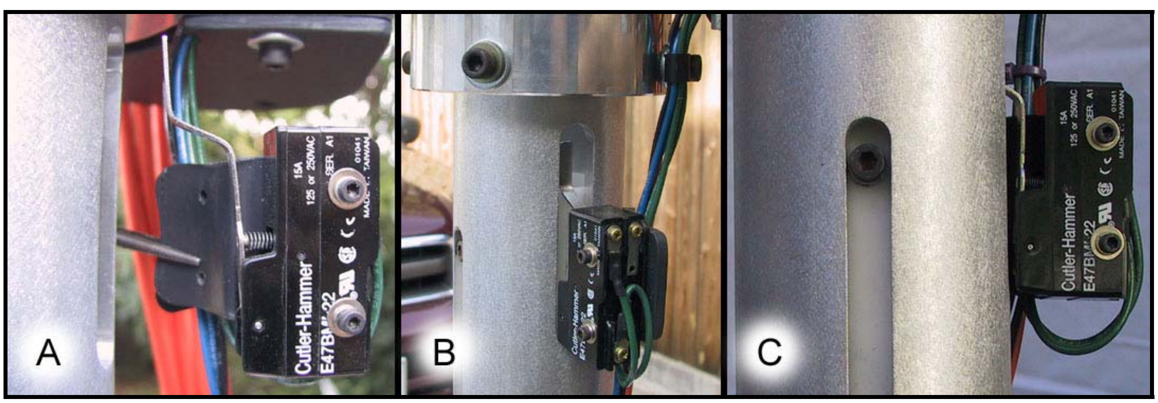


Figure 5.2, Limit Switch

The limit switch is mounted to the top of the sleeve with an ABS bracket just below the motor plate flange. Its actuation lever is positioned to be triggered by the slider about $^{1}/_{8}$" before the final alignment cones mate and the mobile plate stops moving. Positioning the limit switch to trigger $^{1}/_{8}$" before the mobile plate docks ensures the trigger will occur before all movement stops. In other words, there is no reason to “cut it close” because the auxiliary power circuit will keep the motor on until the final alignment cones contact and compress the stabilization spring $^1/_2$" after the mobile plate docks.

5.3 Power Transmission Spring Assembly

When the mobile plate is docked power must be transmitted from the stationary plate to the mobile plate because the trolley requires power from the aluminum c-channel tracks to move. The power transmission and final alignment components were combined in the deployable LMs to create a final product with fewer parts and simplify the LM esthetically. Figure 5.3 shows the power transmission components and the final alignment cones they are mounted to.

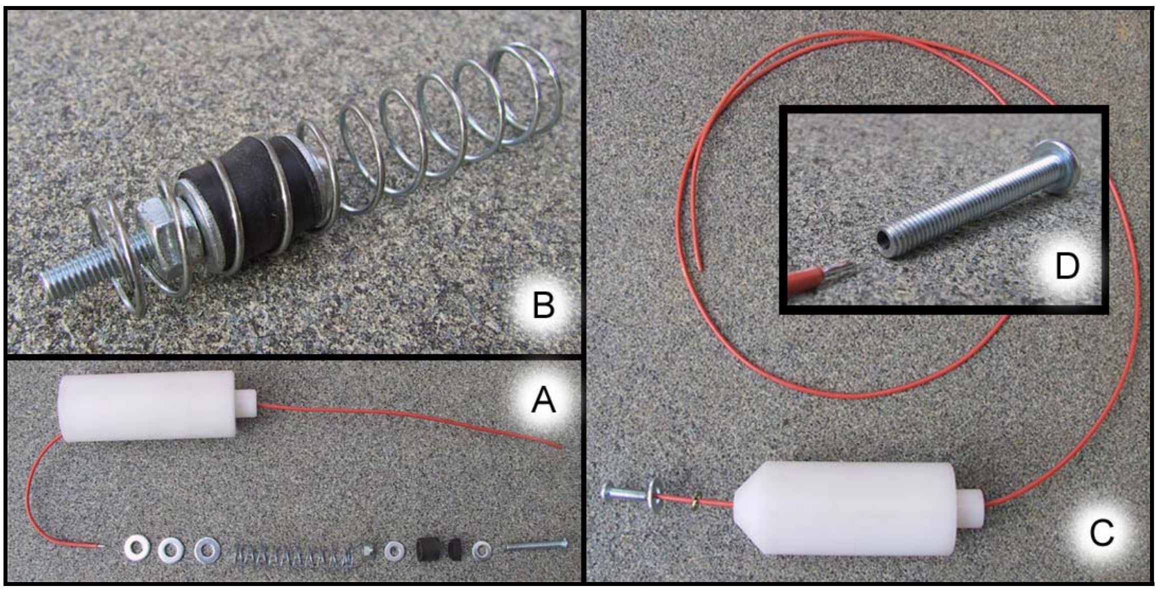


Figure 5.3, Power Transmission Spring Assembly

Frame A displays a stationary alignment cone and the spring assembly. The machine screw on the right side or frame A passes through all the components in the order they are shown and connects to the wire. The wire is pushed into the hole drilled in the end of the machine screw and the two components are soldered together, as shown in frame D. The machine screw, with the wire soldered onto its end, is then threaded into the stationary final alignment cone. The final assembly is shown in frame B, minus the wire.

The mobile power transmission components are of a similar design and shown in frame C of Figure 5.3. Only a single flat and compression washer are needed for this assembly. The two finished assemblies are shown as they are about to mate in frame B of Figure 4.22.

The length of the final alignment cones (at least 7") was chosen for two reasons. First, the distance from the sleeve's apex to above the entire contour, where the static plate's flange can be mounted to the sleeve is about 7". Second, there needs to be room in between the stationary plate and the mobile plate for the toggle bone and speed switch.

The final alignment cones also house the springs for power transmission. The mobile final alignment cones have the corresponding metallic contacts exposed; they make contact with the springs. There are four sets of alignment cones, only two are currently used for power transmission (+12V and ground). The remaining two could also be used, and the LM would have redundant power transmission. Frame A of Figure 4.22 shows the four sets of final alignment cones and Frame B displays a close up of the exposed power transmission components. The power transmission components were incorporated into the final alignment cones to simplify the design by minimizing the number of parts.

5.4 Delay Circuit

The need for a delay circuit originated because the weight lifted by the ABDS-trolley LM is 70_{lbf}. When less weight was lifted and the motor turned off, the armature and gears had enough momentum to wind up the cable another $^1/_4$", and put additional tension in the cable. This method still works well for the camera LM because it lifts less than 20_{lbf}. When the ABDS-trolley package rolls on or off the LMs tracks and no delay is used, the moving weight shifts the mobile tracks alignment up to $^1/_2$". Even though the trolley never got stuck rolling off the LMs tracks, the $^1/_2$" vertical shift was concerning and warranted the inclusion of a delay circuit. The delay circuit is shown in Figure 5.4 and the schematic is contained in Appendix IX.

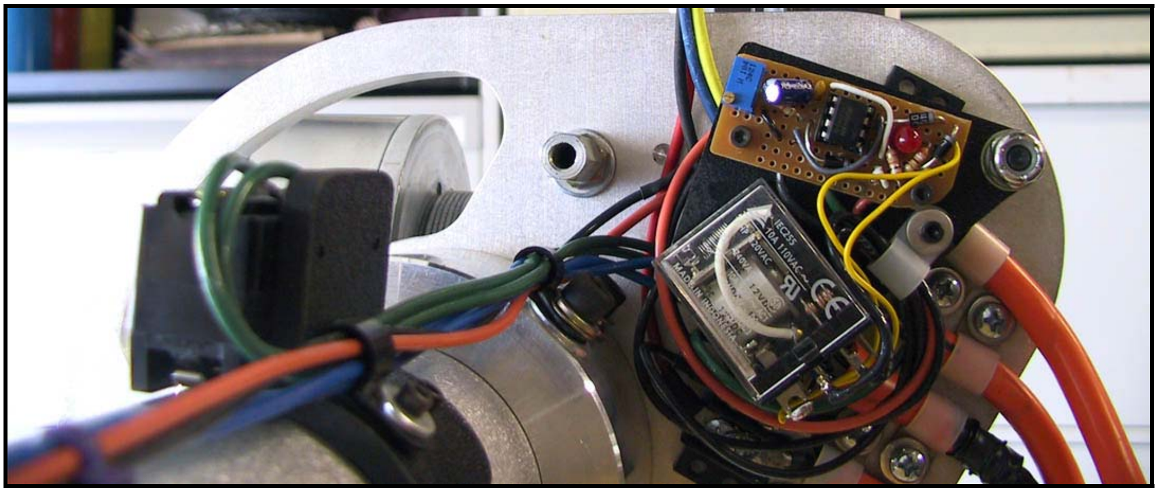


Figure 5.4, Delay Circuit and Relay

5.5 Motor

Five criteria were considered when choosing the LM's motor. They are the torque capability, overhung load rating, output shaft speed, gearbox design, power source, and the brand and distributor. The motor sourced was a compromise of all these criteria. Ideal speed and torque were compromised for weight and price. Two views of the motor are in Figure 5.5. The camera LM and ABDS-trolley LM motors may be indistinguishable from the outside; their difference is the gear ratio inside the gear box.

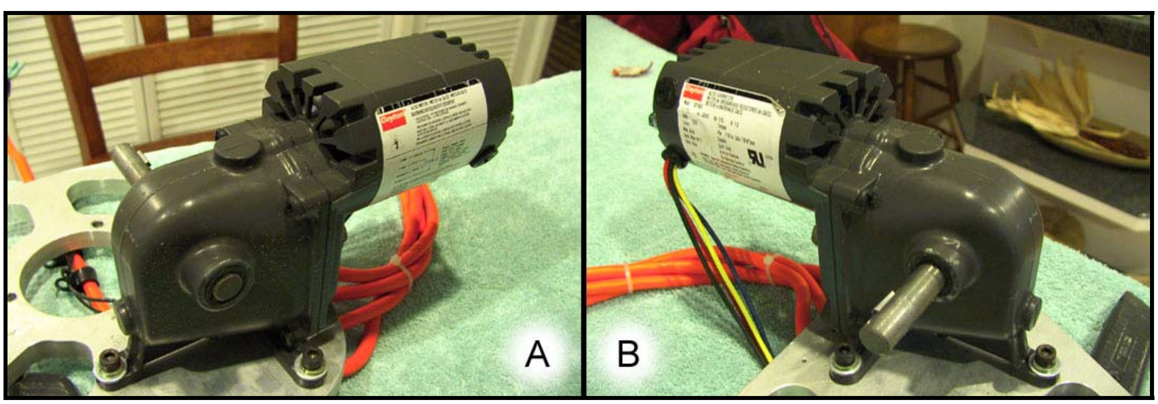


Figure 5.5, AC-DC Electric Gear Motor

5.5.1 Torque Capability

The LM is designed to lift the trolley and ABDS along with its own mobile plate, tracks, and plug. The current trolley weight is 25_{lbf}, the current ABDS weight is 40_{lbf}, and the mobile plate with plug, track, and alignment cones weighs 10_{lbf}. To lift this total weight of 75_{lbf} on a 2" diameter pulley the motor needs to have at least $75_{in\text{-}lbf}$ of torque. When the motor is slowed down for the angular alignment process, the motor still needs to have enough torque to lift the trolley and ABDS. The largest amount of torque is needed after the mobile plate docks and the stabilization spring is compressed. The stabilization spring has a spring rate is 85 $^{lbf}/_{in}$ and is compressed about $^{1}/_{2}$" before the time delay circuit turns off; this adds an additional 45_{lbf} of load to the motor in addition to the 75_{lbf} being lifted for a total of 120_{lbf}.

5.5.2 Gearbox Design

When the motor stops with a full load attached to the mobile plate, whether the mobile plate is docked, dangling in the air, or near the ground being unloaded or loaded, it is undesirable for the motor to unwind. The 1787:1 gear ratio, gear inertia, and worm drive design allows the motor to move the mobile plate and ABDS-trolley package, but the weight of all these hanging components can not move the motor. If the combination of the total overhung load and radius to the point the cable rolls off the pulley barrel has enough torque to overcome the gear-motor, the lifted load will return to the ground when the motor turns off.

The worm-drive gearbox was chosen to ensure that the lifted weight can not unwind the pulley cable when the motor stops. An important consideration is the amount of torque that can be applied to the output shaft before the motor's armature moves and the pulley unwinds. The motor was tested beyond its overhung rated values and typical loading by hanging 200_{lbf} on the mobile plate. The motor was able to pull up the 200_{lbf} and the pulley cable did not unwind when the motor was turned off.

5.5.3 Output Shaft Speed

The speed the ABDS-trolley package is lifted when the initial alignment takes place needs to be as slow as possible. Previous LM prototypes showed the slower the mobile plate is lifted the less jolting applied to the ABDS-trolley package. A gear motor in the IEL was tested on the Second ABS prototype. This motor had a $^1/_{20}$ HP, 13 RPM output shaft speed, which was too fast for the final alignment. If the voltage was turned down, the motor did not have enough torque and control to align the fully loaded mobile plate. Therefore, the output shaft speed of the deployable LM motor is $^1/_3$ of the motor used on the Second ABS prototype.

5.5.4 Overhung Load Rating

The motor's rated overhung load is 100_{lbf}. This means that the motor is designed to have up to 100_{lbf} pushing on the side of the output shaft (Grainger, 1999). For this design, the overhung load would be the weight hanging on the cable. Attaching the pulley directly to the motor's shaft simplifies this prototype from previous designs because it eliminates the need for additional bearings, bearing blocks, a shaft, and shaft coupling between the motor shaft and pulley shaft.

5.5.5 Brand and Distributor

The Dayton brand motor was chosen because it is a popular, well-known brand, which can easily be sourced through Grainger. Replacement parts are also easily sourced through the Dayton motor parts toll-free telephone number. See the company list for the motor and motor parts replacement information. Grainger's motor selection guide was used for understanding motor terminology and sourcing the correct motor (Grainger, 1999).

5.5.6 Power Source

An AC-DC motor was chosen for two reasons. The first being that the prototype was in the early design stages and the power source used during testing or on the highway is subject to change. The second reason is that the control components are also in the development stage and may be subject to change.

5.6 Wiring

The LM has a box mounted to the underside of the motor plate. The cover shown in frame B of Figure 5.6 can be removed by unscrewing two machine screws to expose the delay circuit and motor relay. The delay circuit and motor relay pivot out of the way to expose the junction strip for all wires connecting to the LM and run to the switches and motor. The circuit and junction strip are shown in frame C. The control switch for moving up and down is mounted to the front panel of the control box shown in frame D. Frame A shows the +12V power lead bolted to the LM’s +12V trolley track. The trolley gets its power directly from the energized track; the track is insulated at its mounting locations.

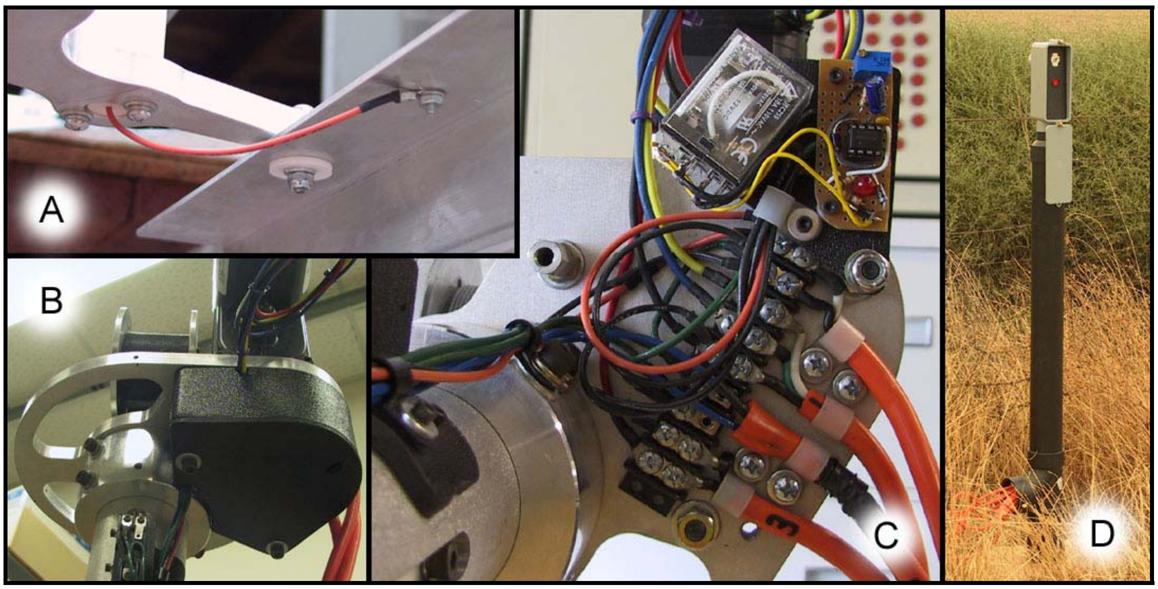


Figure 5.6, Electrical Circuit and Control Box

The control box shown in frame D of Figure 5.6 has two power plugs that can be used for two power sources. One plug has 120V AC written on it. This is the power used to move the mobile plate at full speed. Another plug has “VAR AC” written on it to indicate it should be plugged into a variable AC power supply. The speed switch toggles a relay to supply the motor with the variable power source when the alignment is taking place. The schematic for the wiring of the LM and control box is in Appendix IX.

CHAPTER 6

ENVIRONMENTAL FACTORS

The deployable trolley LM is suitable for initial outside use by Caltrans because it is made of materials that have performed reliably in direct sun, rain, and both cold and warm temperatures. The trolley LM has been subjected to endurance testing during Sacramento's summer and winter seasons.

6.1 Summer Endurance Testing

In the summer, the LM was cycled continuously, over 100 cycles in direct sun and no wind, which could have allowed for forced convection cooling of the motor. The East Sacramento testing location ambient air temperature in the shade on September 6, 2004 was over 100 °F. The LM successfully completed every cycle impeccably in the four hour test; each cycle took almost 2-$^{1}/_{2}$ minutes. Figure 6.1 illustrates the setup used; the fan spun the mobile plate before it rose back up and docked every cycle. The weight lifted during this testing was a collection of steel and aluminum billets and plates weighing 105_{lbf}.

Figure 6.1, Summer Endurance Testing Setup

6.2 Winter Endurance Testing

The LM has been set up in a similar manner to the setup shown in Figure 6.1 and exposed to winter weather for over two consecutive months. It was setup in a damp, shaded testing location for the months of October, November, and part of December 2004. The LM was subject to fog, rain, and freezing temperature conditions. Approximately 10 different dates in that time period the LM was cycled a few times to check for proper operation; it performed flawlessly. Rain proof shielding was on the LM the entire time it was sitting outside during the winter months.

ABS shielding can be installed or removed in about two minutes. Frame A of Figure 6.2 shows the shielding removed from the LM. The sleeve of the LM is covered with two pieces of 4" diameter ABS pipe. Each piece of pipe has been cut along its length to a little more than half; the two pieces pushed together look like a figure 8 from above. Each pipe piece is anchored with a machine screw to the motor plate and stationary plate flanges. The motor and pulley are covered with $^{1}/_{8}$" ABS sheet. One piece of ABS sheet was heat formed to match the motor plate's edge contour. Then another piece was glued to the top as a lid. This cover is attached to the motor plate's edge at three points. The speed control switch has a piece of 3" ABS pipe cut as an enclosure. It is attached with one machine screw through the stationary plate and can be seen in frame C. Frame B shows the LM covered up but still dry, frame D shows the rain shielding being rigorously tested with the horizontal spray from a hose.

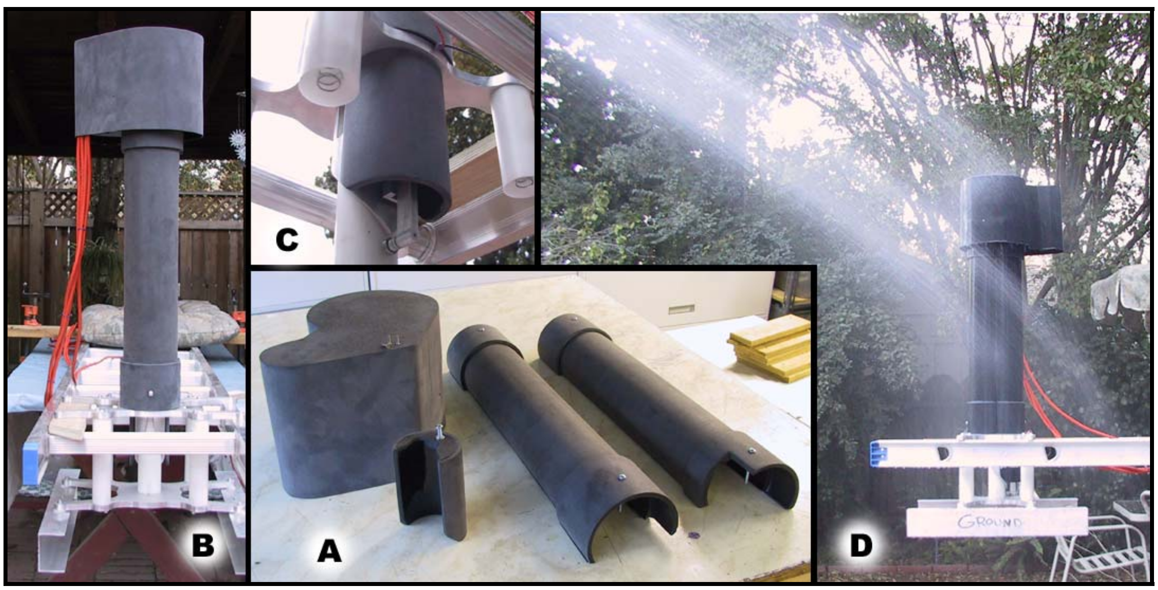


Figure 6.2, Rain Shielding

In addition to the ABS shielding, the speed control and limit switches were rain proofed. This was done by removing the lever arm pin, lever arm, and plunger of each switch. By coating the plungers sides and flange with grease and then putting it back into its slot, the only access rain water has to the switch inner components has been eliminated.

CHAPTER 7

CAMERA LIFTING MECHANISM

The camera LM was created as a bonus to the research technical agreement with Caltrans. This LM has all the features and even uses many identical parts to the ABDS-trolley LM. Some parts were made smaller because the mass being lifted is considerably less. The stationary and mobile plate geometries were changed for mounting a specific camera box specified by Caltrans. The camera box is asymmetrically loaded with the camera components inside so a custom mobile plate was made to position the box's center-of-mass directly under the plug.

It was possible to make the plug and sleeve lengths shorter because less mass needed to be aligned and there are no trolley tracks to align vertically with. The mass being lifted weighs less than 20_{lbf}, compared to the ABDS-trolley LM requirement of 75_{lbf}. This decrease in inertia means smaller forces are applied to the sleeve and plug when the rotational alignment takes place; the distance the plug needs to be into the sleeve before rotational alignment begins does not need to be as long. The plug length was also made shorter. The slider length did not change because it depends on the contour height and toggle bone, which did not change.

The same Dayton motor is used, but the gear ratio of the gear box was much less. Because the camera mass is significantly smaller, it can be lifted faster for the same power input, and has less inertial forces on the LM during rotational alignment. The time delay circuit was omitted from the camera LM design. When the limit switch cuts the power to the motor (about $^{1}/_{8}$" before the final alignment cones bottom out) the rotational inertia of the motor armature and gears in the gear box continue to pull the camera up about $^{1}/_{2}$" as they slow down. The final alignment cones bottom out and the stabilization spring compresses slightly by the time the armature is out of rotational energy and has stopped spinning.

Frame A of Figure 7.1 shows the camera LM. Frame B is a close-up view of the custom mobile plate to mount the asymmetrically loaded camera box. CAD drawings for the camera LM are in Appendix II. The stabilization spring is significantly smaller for the camera LM because it is lifting less than 20_{lbf}. The spring is shown along side the Deployable ABDS-Trolley LM spring in Figure 7.2.

Table 7.1, Camera Lifting Mechanism Criteria

1	Lift 16-pound, asymmetrically weighted aluminum camera box.
2	The docked camera will need power transmitted to it through the LM.
3	User controls the LM from the ground near its mounted operating location.
4	Heat sinks will be located on the top and on one side of the camera box.
5	The camera should not vibrate or move around when docked.
6	From a user's point of view, quick lifting and lowering is best.
7	The aluminum camera box can bounce off the light pole during vertical travel.

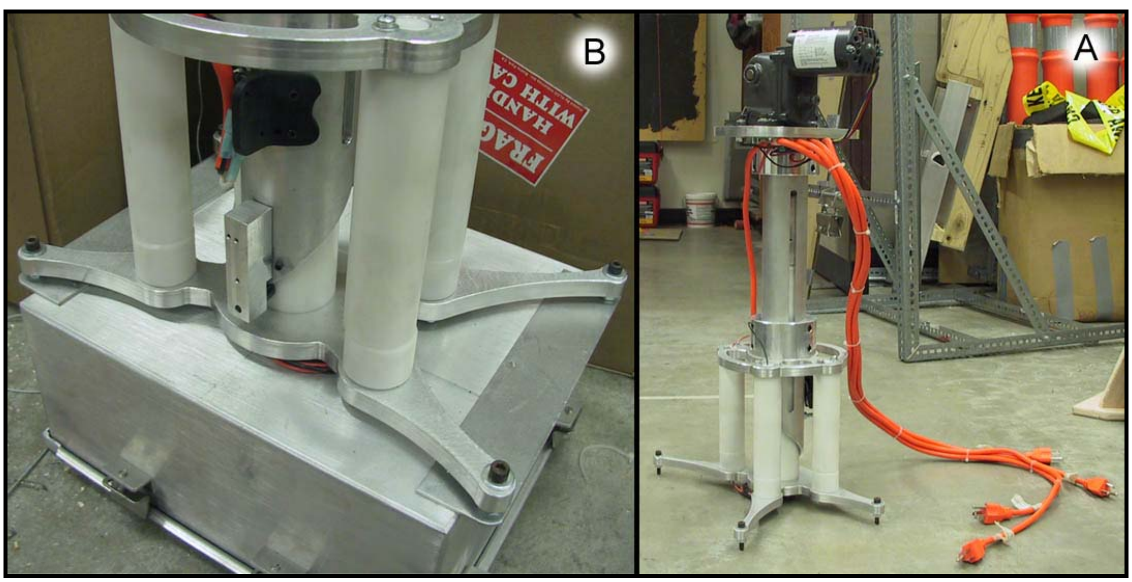


Figure 7.1, Camera Lifting Mechanism

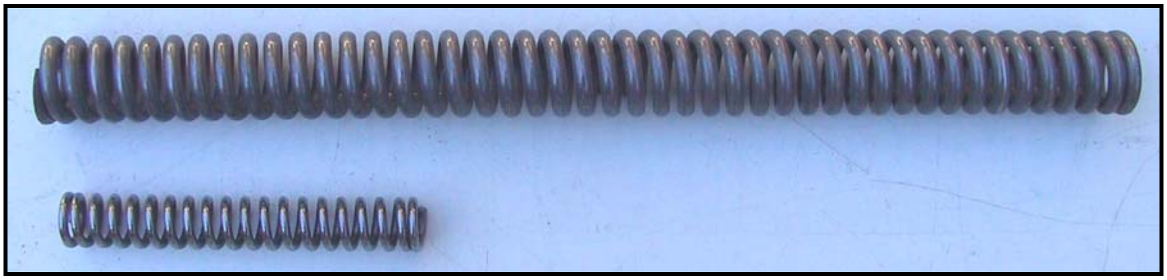


Figure 7.2, Stabilization Spring Comparison

CHAPTER 8

SUGGESTIONS FOR FUTURE WORK

The deployable ABDS-trolley LM has shown a degree of 100% reliability for more than two hundred cycles. Different paths could be followed to further develop the LM. A suggestion would be to optimize for cost, weight, and size. Reliability from a statistical and mathematical perspective could also be addressed. Another prototype could then be built or maybe a short run (5 or 10) if Caltrans wants to deploy ABDS-trolley packages at different locations. Since functionality is already achieved, the amount of time spent improving the LM may not be economical unless it's for marketability within Caltrans or mass manufacturing. Refinements can always be made to a design. Here are a few items worth mentioning if this apparatus is reproduced in larger quantities or if future changes are made to the design.

8.1 Clutch

One of the biggest concerns with the LM is making sure the motor shuts off at the proper time. The Second ABS Prototype was the first prototype to incorporate a motor and automatic shut-off using a micro-switch. The latest prototype also uses a micro-switch, but the micro-switch is connected to an electronic delay circuit with a relay that interrupts the power to the motor after a short period of time. The delay is adjustable between fractions of up to a couple seconds.

The clutch was designed early on in the creation of the deployable ABDS-trolley LM as a safety feature if the micro-switch doesn't trigger. Its purpose was to slip if the plug becomes jammed or stops moving before the limit switch is triggered. The clutch's design was conceptually flawed for the hanging mass LM application. If the clutch slips one "notch" it rotates 30° before re-engaging. The mobile plate and other components attached to it fall about $^1/_2$" before the clutch reengages. This is enough distance for the ABDS and trolley to gain the momentum to overcome the clutch's ability to reengage and hold up the hanging mass. The package will fall to the ground. Figure 8.1 shows five frames of the clutch assembly.

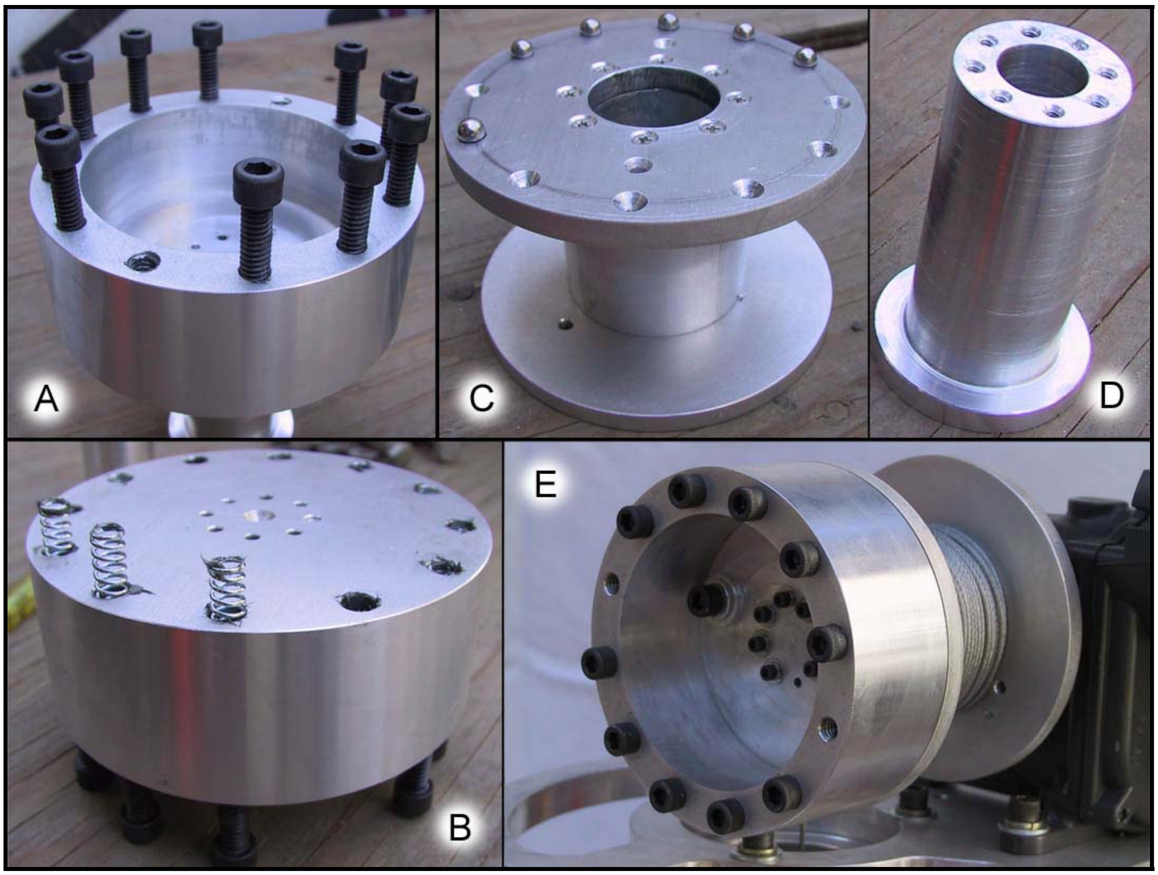


Figure 8.1, Clutch Assembly

The entire clutch assembly is shown in frame D of Figure 8.1. The spring loaded clutching component is shown in frames A and B; each counter-bore holds a spring and over half the bearing's length. This component is rigidly connected to the motor's shaft through the component in frame D via the eight 4-40 SAE holes on the 1-$^1/_2$" bolt-hole circle. The pulley drum and walls, which are rigidly connected together, are shown in frame C. The pulley is connected directly to the cable holding up the ABDS-trolley package. The pulley is also connected indirectly to the motor shaft through the compressive forces from the springs pushing the steel bearings into the recesses. This is where the slipping occurs, between the flat, circular surface shown in frame B and the side of the pulley wall in frame C. Each bearing is spring loaded and each spring’s force on the ball bearing is variable by adjusting the corresponding $^{1}/_{4}$-20 SAE socket head cap screw to that counter-bore. Two counter-bores in the clutch were left empty and without set screws because only 10 springs came in a package and only one package was purchased.

8.2 Limit Switch

The reliability of the limit switch could be statistically evaluated. Joe Palen suggested replacing the switch with an optical sensor. This theory could also be applied to the speed switch. Fairchild Electronics makes an optical sensor with part number QRB1134.

8.3 Sleeve Friction

The amount of friction the sleeve puts on the plug can be increased if desired. Experience with previous LM prototypes warranted a concern when the plug’s free-spinning ball bearing chatters back and forth in the contour’s transition curve, right before it reached the 0.502" wide slot. It was a concern that the LM would see very high stresses compared to the rest of the alignment process. Because the ABDS-trolley package is moving so slowly when the plug aligns, and because the transition contour between the contour and slot is more gradual than past LM sleeves, the bearing doesn’t chatter and no dampening friction was needed. Frame A of Figure 8.2 shows the slot cut into the slider. Frame B shows the slot cut into the sleeve behind the supporting flange. This slot cut out of the sleeve is for access to the plug from a frictional dampening device. The device was never created because the need never arose on this particular LM. The slot cut out of the plug is so the friction device doesn’t affect the slider. If the plug ever experiences angular momentum at this point, additional friction can be added in this manner.

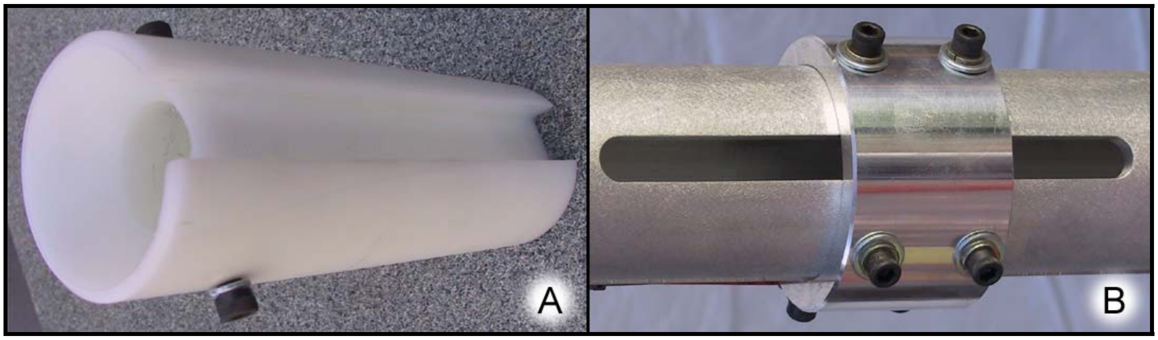


Figure 8.2, Additional Friction on PLug from Sleeve

8.4 Power Transmission Through Final Alignment Cones

The first attempt at power transmission through the final alignment cones worked perfectly, but the system has many parts and is complicated. A simpler way to hold the spring inside the cone and transmitting power through it could be found. The power transmission components should also be tested to failure with electricity and given and Ampere rating.

8.5 Test Lifting Mechanism to Failure

Some endurance testing has been done with the current deployable LMs, but more should be done before any sort of a next prototype is built. Contaminating the sleeve, plug, and slider with dirt to simulate years of dust and wind is recommended. Adding more weight until the LM can’t lift it any more, either because the cable breaks or motor is stresses is also essential. Disconnect the LM speed switch and let it run too fast would be a good test until failure occurs. Disconnecting the limit switch and let the motor and cable connection points self destruct would also be a good test. Testing in more extreme weather would also be a good idea. The LM is at the stage right now to begin this type of destructive testing.

8.6 Mobile Truss Components

The mobile truss is more of a tool and not the final product. However, a couple things could be done to improve its abilities. If all the wood were replaced with a material that is rain proof, the truss could be used more diversely. This would include the fishplates and the isolating plywood between the C-channel and the ladder sections. Be careful, however, wood can be surprisingly strong and is actually quite suitable for this application. Anther material, such as aluminum may be more work than it is worth to make a replacement. Testing over Old Hutchison Road is limited in height by the high-voltage power lines over the road's north shoulder. Finding another approved location and preparing it, if necessary, for testing would allow for more diverse testing of the ABDS.

CHAPTER 9

THE TROLLEY

Project Scope

The universal mounting platform, herein referred to as the trolley, was developed as a flexible and mobile device for a variety of traffic detection systems that are to be placed directly over the lanes of a freeway. Encompassed in this project are the trolley, a set of tracks that are to be mounted to existing trusses that span freeways, and a lifting mechanism to hoist the trolley and detector to the required height. This system enables operators on the ground at the roadside to have easy access to the detectors and the ability to position them over a specific part of the road. The original design was created to accommodate the Laser-Based Detection System (LBDS) designed and constructed at UC Davis. In order to allow field testing of the trolley, lift, and LBDS, a mobile truss was built that allows the entire system to be positioned at a working height. The mobile truss is large enough to span a two lane road. Field tests included driving vehicles underneath the truss so real world data could be collected for the detector.

The primary focus of this chapter will be on the trolley and tracks. The lifting mechanism and mobile truss were detailed exhaustively in previous chapters. The trolley has six major subsystems: a platform; drive system; alignment wheels; brakes; power delivery; and control system. The trolley platform consists of a flat plate that all the trolley's components connect to and a mounting plate that acts as an adapter between the detector and the body. The drive system has two axles, a motor, gearbox, and wheels. The motor drives the rear axle, while the front axle is left idle. Alignment wheels keep the trolley moving straight and prevent rubbing against the tracks. Brakes are located at the front of the trolley and are meant to prevent accidental movement due to wind. Power consists of a DC current fed through conductive strips fixed to the tracks with battery backups. The control system uses onboard electronics to run the motors, read any sensors, and communicate to a controlling program run by the operator.

9.2 Original prototype

The original prototype was single plate with wheels, a drive system, and brakes. The brakes were a pair of linear actuators placed under the wheels that squeezed the tracks between itself and the wheels. A control system was later added to allow the trolley to be operated remotely. This was comprised mostly of custom built circuits driven by a PIC microcontroller and sent signal with a small RF transmitter. The system worked as required, but many problems existed. The electronics were only hand etched boards, didn’t use many commercial chips, and were much larger than necessary. In addition the brakes were actuated with a separate control board requiring a lot of extra space. The remote used had too limited a range. The brakes were underpowered and often got stuck in the locked position requiring a person to pry them loose. Because of the space required for the brakes, the trolley’s main platform rested over 4 inches below the bottom of the tracks. This meant that any detector mounted on the bottom of the trolley was closer than necessary to the traffic below and reduced the safety clearance at any given location. The height of the trolley also created problems in the drive system which required the use of flexible shaft couplings. Initially small plastic hemispheres were attached to the end of each axle to reduce friction with the tracks when the trolley veered to one side. With the additional weight of the detector, the friction increased enough to stall the drive motor. As seen in the following sections, many of these issues were fixed with the design the current trolley prototype.

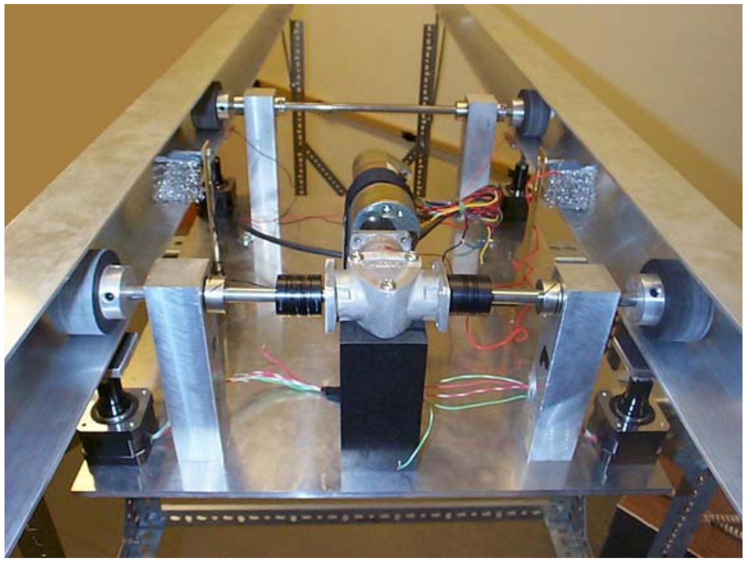


Figure 9.1: Original prototype

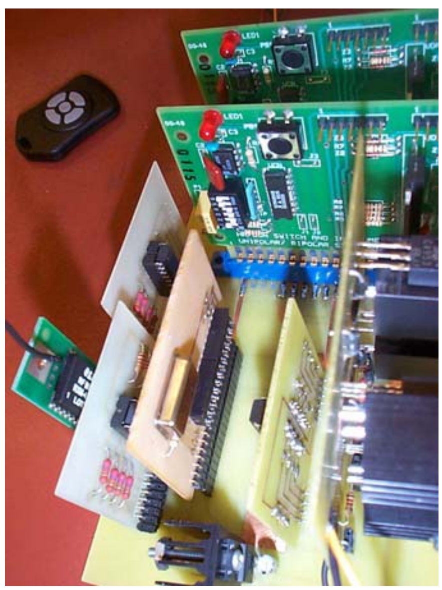


Figure 9.2: Early control system

9.3 Trolley Platform

At its heart, the trolley is simply a large metal plate with wheels, running inside two pieces of C-channel. The material used for the majority of the pieces is aluminum because it is lightweight and corrosion resistant. The body is a rectangular plate of 0.25 inch aluminum drilled with all the necessary mounting holes. The detector, instead of mounting directly to the body, is attached to an individually made mounting plate which is also made out of 0.25 inch aluminum plate and bolted underneath the trolley. This accomplished two goals. First the mounting points on the trolley body are four fixed bolt holes making it a universal platform. Any special mounting requirements for a detector can be taken care of with the mounting plate. With each detector, a custom mounting plate becomes its adaptor. Secondly, the location of the center of mass of the combined trolley, detector, and lift platform is important during lift operation. If the lift platform is tilted too severely when it engages the upper part of the lift, this can create a failure mode by putting extra strain on the lift cable. The combination of the fixed holes on the trolley body and the mounting plate allows the centers of mass to be correctly aligned without repeated measurement.

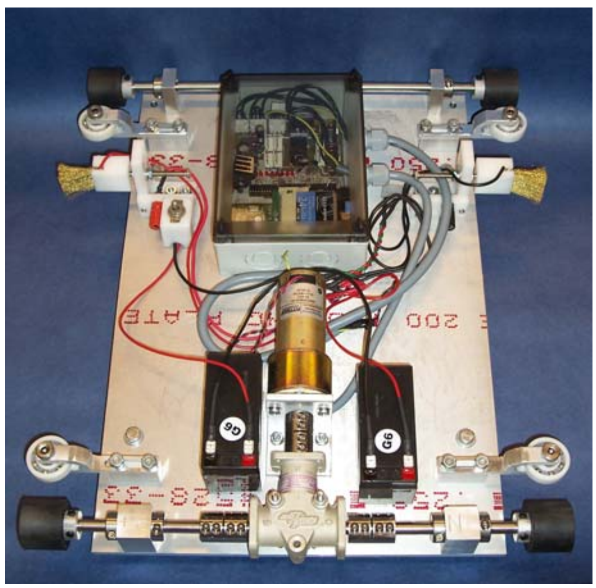


Figure 9.3: Current trolley

The entire device rolls inside the tracks, which are comprised of 3 x 2 x 0.125 inch aluminum c-channel. This provides both an easy shape for mounting onto trusses and encloses the trolley. The only way that the trolley can fall is if the tracks break or the trolley rolls off the end. One end of the tracks is covered by the lift which has an integrated stop and the other end has bolts that pass through each track.

One major improvement to the trolley was the ability to lower the profile to 3 inches so that it fit within the track height. This reduces the distance that any attached detector would encroach into the safety clearance between the truss and vehicles to its minimum. By not sticking down into traffic any further than necessary, more locations for deployment are possible or more space is available for the detector in the vertical direction. Many changes were made that enable the trolley to attain a low profile: The drive system was lowered; The alignment wheels were redesigned; The brake design was completely changed; The brushes and brush arms were reconfigured; The electronics were reduced in size. The changes are detailed in subsequent sections.

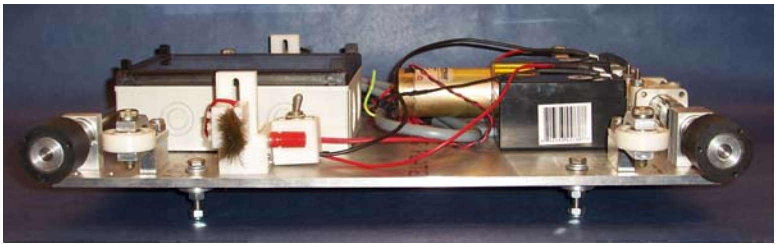


Figure 9.4: Low trolley profile

9.4 Drive System

The drive system for the trolley consists of a geared DC motor, a 1:1 right angle gearbox, and neoprene wheels with aluminum hubs. The motor is capable of 480 in-oz continuously and no further gearing seems necessary even with the full weight of the detector and trolley. The axles for the trolley are a hardened 440C stainless steel to keep some corrosion resistance but add strength to this critical component. The neoprene wheel are pliable enough to provide the necessary traction and have good weather resistance. The area where traction is needed most is at the junction of the lift tracks to the main tracks. Even when fully docked there is some give in the lift platform and the trolley needs enough traction of climb a vertical transition of approximately 0.25 inches.

Part of reducing the height of the trolley was mounting the drive system closer to the body making all the supports much shorter. While creating these pieces, the alignment was improved enough to allow the use of rigid, stainless steel shaft couplers. Since one previous flexible couplers had failed during testing, this improvement to the drive system eliminated a previous problem.

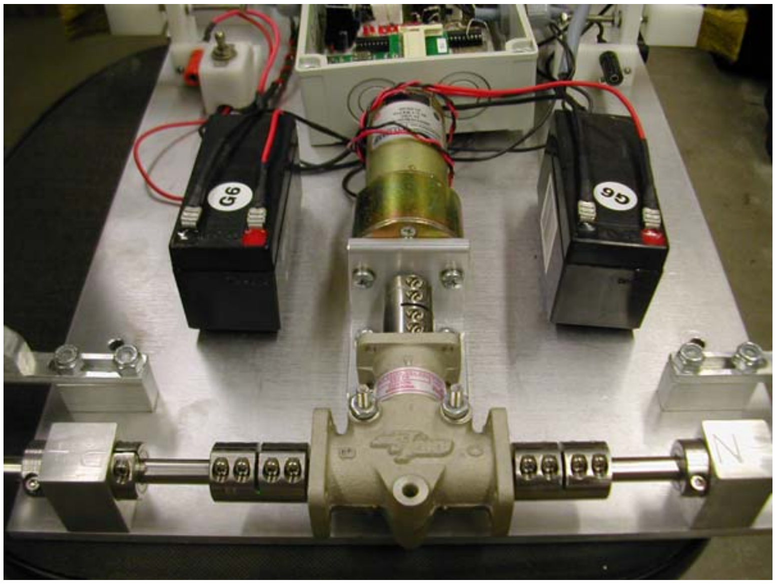


Figure 9.5: Drive system

9.5 Alignment Wheels

The first iteration of the trolley used hemispherical delrin caps at the end of each axle to reduce the friction if the trolley didn't roll straight and the wheels started to rub on the inside face of the tracks. When the trolley was tested on its own, this worked as designed allowing the trolley to continue to run even if it was not going straight. Once the detector was attached to the bottom of the trolley, the resulting friction from the wheels and caps due to the increased weight was enough to stall the trolley. The solution was a set of alignment wheels similar to those used on a roller coaster. A roller bearing was positioned near each wheel, spaced so that if the trolley drifted to one side of the tracks these alignment wheels would make contact first and roll along the inside wall. These worked well, but there were a couple issues with the original design. The bearings were metal requiring a plastic sleeve inside to prevent a short, and they were mounted to the same supports as the axles which became too short to accommodate them. The current design of the alignment wheel moves them to their own supports and utilizes delrin bearings so that the electricity in the tracks is not an issue. The alignment wheels extend past the main wheels by approximately 0.125 inch on each side.

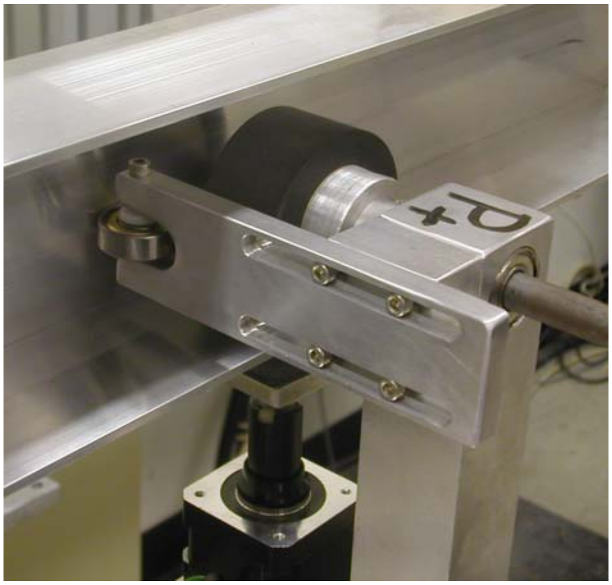


Figure 9.6: First alignment wheels

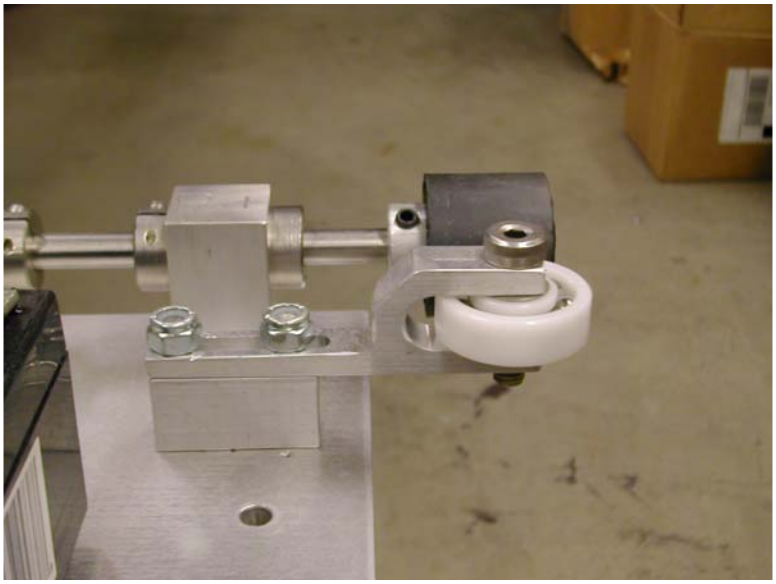


Figure 9.7: Current alignment wheels

9.6 Brakes

The most difficult system to design on the trolley was the brakes. The trolley doesn't need any sort of dynamic brakes because it moves quite slowly. The brakes serve as a parking brake, keeping the trolley from moving inadvertently while on the lift or above the roadway. Originally the brakes were linear actuators driven by a stock circuit board. They provided a decent amount of braking force, but had some problems. Aside from the required circuit board being too large, the actuators would get stuck when set (i.e. the brakes are on), necessitated the added height to the trolley that was undesirable, and relied on the current running through the bottom face of the tracks. Getting stuck was an especially troubling problem because it would require someone to climb up to the unit on the truss. Using the electricity in the tracks needed to change because the power delivery system was also being modified.

Figure 9.8: Original brakes

Through the experiences with the original brakes, goals were specified during the design phase of the current trolley. First was to reduce the vertical space requirement. This would allow the main part of the trolley to fit within the 3 inch track height even if parts of the brakes hung below the tracks. The idea is to get the detector as close to the tracks as possible. Another very important goal was increased reliability (i.e. don't get stuck). Being stuck requires a person to climb over the roadway, negating one of the main safety advantages of the trolley system. Next was to eliminate stress on the wheels, axles, and bearings. Because the trolley hangs above a road, failures can have severe consequences. The first brake design pushed against the bottom of the track and used the wheel to squeeze the track in-between. The best case would be an independent brake that only stressed itself when squeezing the tracks and didn't add any stress to the weight bearing parts of the trolley. To keep everything on the lift platform in the event that it swung severely while going up or down, the brakes would need to provide enough force to support the full trolley and detector weight. This would prove to be difficult, but necessary in the absence of additional safety features. The next goal was to get rid of dedicated controller board for the brake motors and incorporate the electronics into a new main board. This would come with the electronics redesign and by choosing actuators that could be driven with standard circuits. The last goal, but still important, was to provide clearance for vertical track misalignment and for horizontal movement of trolley between the tracks. Even if the installed track sections are perfectly aligned, we know the trolley experiences approximately a 0.25 inch vertical misalignment while leaving the lift due to its shifting weight, and it does have some extra space horizontally.

With these goals in mind, several concepts were considered. In short, several designs that expanded like a scissor jack pushing on the inside of the tracks were dismissed due to the chance that the track could be failed. Solenoid based design required constant power to stay on or off and generally lacked the force we were looking for. Several other concepts were reviewed and evaluated. The idea we pursued came from a bicycle linear-pull brake. Using the pads from a bike brake due to their adjustability and brake arms similar to that of a linear-pull brake, we were able to attain enough motion to both clamp the track and create the desired clearance. These types of brakes are typically actuated by a cable. Since brakes on the trolley are to be left on for long periods of time, a cable would inevitably stretch creating the need for periodic adjustments. Instead, an acme thread was used because it eliminated this issue and because it has a self-locking property to insure that the brake stays set. This was coupled with a worm gear driven by a small DC motor to increase the mechanical advantage. The finished brake prototype created the necessary force at the pads to hold the trolley on the lift. Although slow to actuate due to the high gear ratio, it did not get stuck in our tests. It allowed the trolley body to be located in line with the bottom of the tracks and was controlled with a simple h-bridge. The problems with this design was its complexity and alignment. It required too many custom pieces and was time consuming and costly to construct.

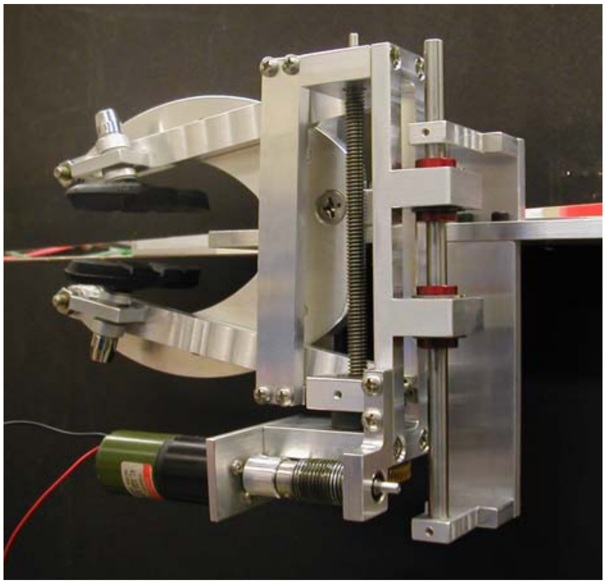


Figure 9.9: Linear-pull style brakes

Instead of trying to simplify the linear-pull style brakes, other changes were made. By adding the locking device on the lift described in the next section, the need for the brakes to keep the trolley on the lift could be ignored. This reduced the force requirement by at least 50%. Less force also meant that the brake could use the wheels as one side of the locking mechanism without too much added stress. A geared DC motor was found that supplied 50 inch-pounds of force. Putting a short brake arm on the axle created a simple way to apply force underneath the wheels. The brake arm in the final version has a rubber coating to both improve friction and electrically isolate the brakes since the test setup still powered the entire track. The coating is *Plasti Dip* obtained from the hardware store and applied in three coats for durability.

The problem with this gearmotor is that it's not self-locking so that any back force against the brake loosened it. To avoid this, a torsion spring was connected to the motor on one side and the brake arm on the other. By allowing the brake arm to pivot on a pair of small bearings all the force is transmitted through the spring and pushing against the brake torques the spring instead of turning the motor. Initially attempts were made to bend the springs outward to fit into a piece attached the motor shaft and the brake arm. The method proved to be very inexact and additional pieces were made that eliminated the need to bend the spring at all. After testing this brake setup with a 40 inch-pound spring, it was determined that a spring with less force needed to be used. Once the motor was shut off with the brakes locked, the springs would push the motor back reducing the applied force. A spring rated at 20 inch-pounds worked well.

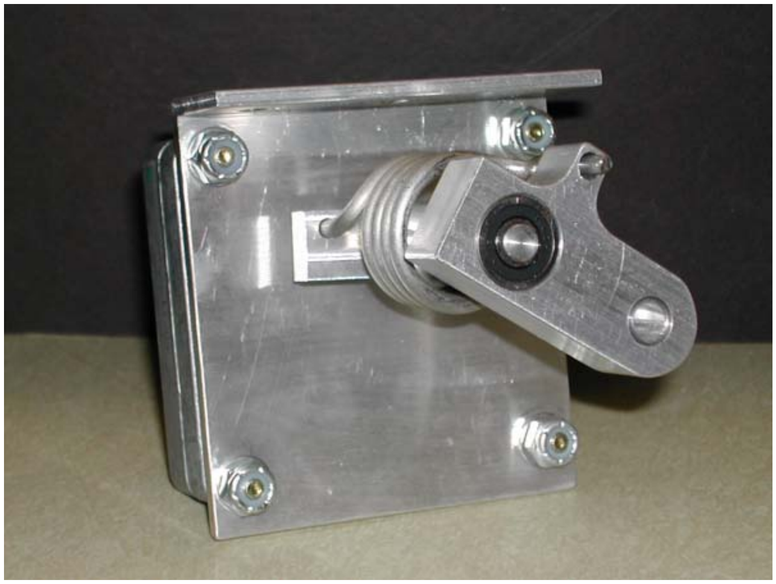


Figure 9.10: Brake with bent spring

The remaining issue was that of control. The spring design aided in this regard. Momentary switches are tripped by the piece fixed to the motor shaft so that it only needs to rotate 180 degrees to set and unset the brake. The brake arm engages the bottom of the track before the shaft has turned 90 degrees with the remaining part of the rotation working to tension the spring. With repeated test it was found that this system worked reliably and was easy to control. The resulting brakes are directional since the arm is rotated in the tracks and remains at a slight angle when set. To solve this issue, we simple made the two brake counter-rotate so their braking forces went in opposite directions.

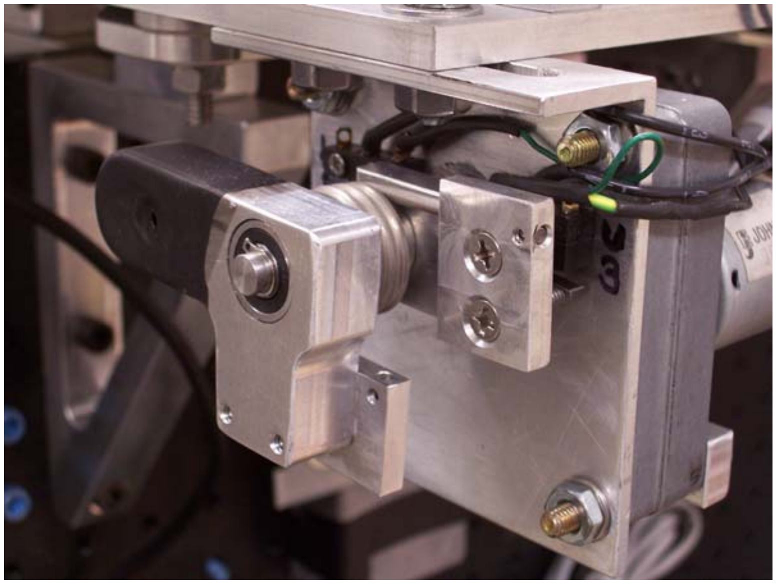


Figure 9.11: Current brakes

One problem was found with the support brackets. Due to the tight spacing a thinner piece of aluminum was used which the brakes would flex inward when set. Attempts to reinforce the corner of the bracket were unsuccessful due to the large bending moment applied. Instead a simple set of braces were constructed that make the brakes rigid.

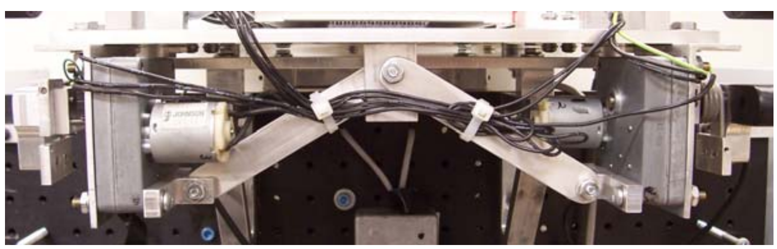


Figure 9.12: Brake braces

As a completed brake system, it does not meet all the goals we had originally set out for ourselves. It does extend below the trolley body so the space requirement goal is not met. However, it is located a one end of the trolley and allows the body to be located at the lower edge of the tracks. This does achieve the overall design goal of fitting the trolley within the track height to mount the detector as high as possible. The goal to provide enough force to support the combined trolley and detector weight was changed due to the addition of the lift gates. The added stress on the wheels, axles, and bearings was not eliminated, but was reduced through the use of less powerful brakes. The reliability requirement of the brakes was definitely achieved with a simple on/off control and a design that doesn't jam any pieces into place. The simple design also doesn't need a dedicated control board as the previous version did. Finally the vertical clearance and horizontal motion requirements are met so the trolley doesn't get hung up anywhere.

9.7 Lift Gates

As mentioned previously, one desired attribute of the brakes was locking the trolley on the lift platform in the case the lift swayed while in motion and the trolley tried to roll off. If the brakes could accomplish this reliably, then a second system to secure the trolley on the lift would not be necessary. After comparing several options of the brakes to options for gates on the lift, it was determined that the simplest system overall would consist of less powerful brakes and lift gates. Solenoid based gates were considered, but the need for constant power in one direction and additional control circuits were avoided with a passive gate system. Angled pieces of delrin are attached to the front of the lift and spring loaded with a small torsion spring around the pivot point. When the lift moves into its fully seated position, the gates are pushed out of the way by the tracks enabling the trolley to leave the lift. After initial tests show the system would work reliably, the delrin was reinforced with aluminum pieces. The only difficulty is getting the spring tension correct so that the lift gates stay in place firmly, but don't require too much extra force on the part of the lift motor to push out of the way. This was largely a matter or trial and error. Small adjustment to the angle of the edge of the gates, the smoothness of the angled face, the contact surface of the tracks, the pivot point position, and tolerances of the pivot holes and bolts all have an effect on the gate motion.

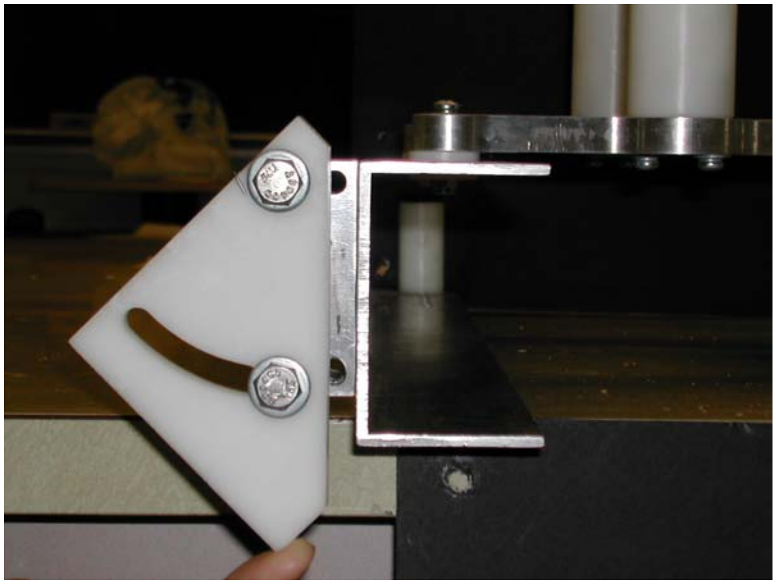


Figure 9.13: Lift gates

9.8 Power

Power delivery was originally specified as 12 volts DC to be delivered to the trolley through the aluminum tracks directly via a pair of bronze brushes. This power drove the trolley itself and was also used to supply the attached detector. After discussions with Caltrans personnel and with the engineers designing the LBDS, we decided to transition the system to 24 volts. The two main reasons behind this decision were that 24 volts is the standard supply available at the roadside where installation of the system was to happen in the field, and that the laser system on the LBDS would be benefit by being supplied with 24 volts. Any subsequent detectors mounted to the trolley can use the supplied 24 volts DC. This was a minor overall change, but is important in that any other required voltage levels need to be converted from this source.

In testing the prototype trolley, it has always been possible to isolate the electrified tracks from any supporting structure using wood. When deployed, however, the tracks will be bolted or clamped directly to the supporting metal truss. In short, the power would be connected directly to ground. A small overhead wire system was considered to isolate the power from the track, but was dismissed due to the added complexity. In order to keep using the metal brushes as in the past, a combination of plastic and foil tapes were explored to allow the power to run along the tracks as before and still be isolated. In the end, samples of a polyolefin film tape and an aluminum tape were used effectively in lab tests. The biggest concern was that at 5 mils, the aluminum would easily tear under pressure from the brushes. It held up very well showing no signs of wear during preliminary tests. At the current time, only a single section of track was outfitted with the conductive strips. The other track sections will need to have the strips added and interconnections between them still need to be designed.

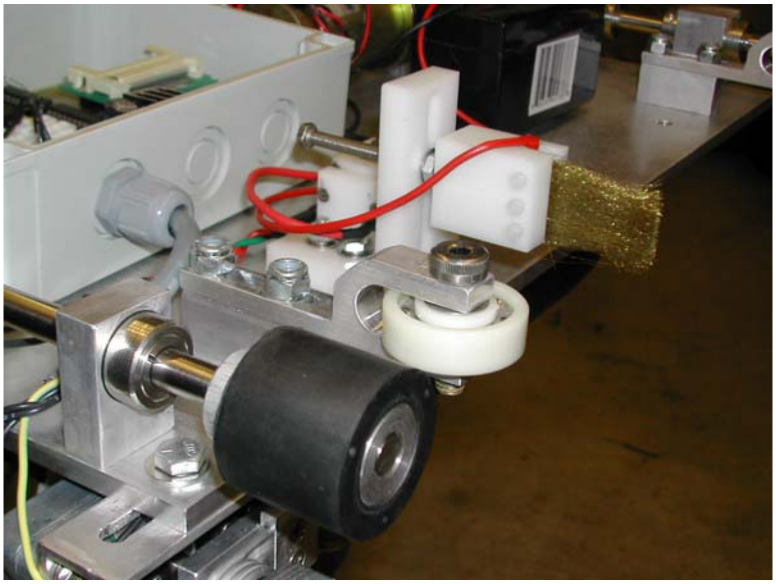


Figure 9.14: Brushes for power

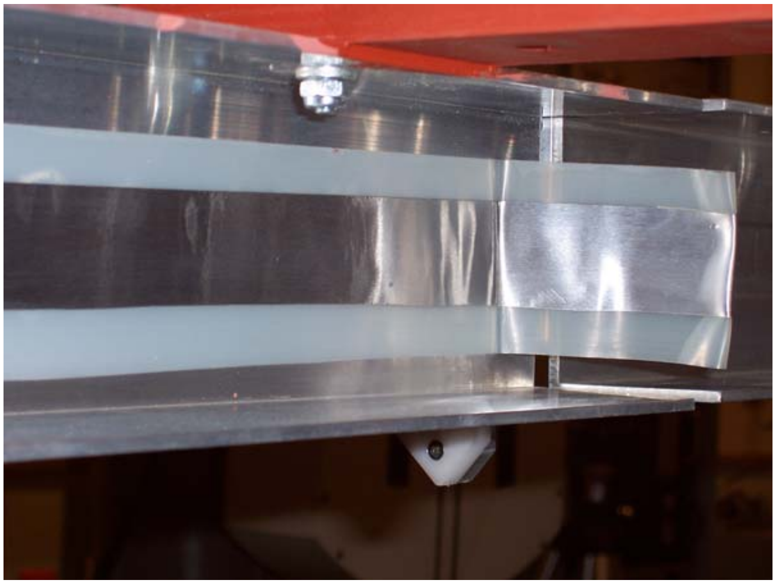


Figure 9.15: Conductive strip

Under normal conditions the trolley should always be connected to the power supply inside the tracks. However, the transition from the lift platform to the tracks and between sections of tracks could create momentary interruptions in power due to gaps in the tracks. To allow the trolley to move past these, batteries are mounted on the trolley body to supply temporary power to the drive system and brakes only. The worst case would be a problem with the power supply after the trolley was over the roadway. The batteries were chosen to allow the trolley to be driven all the way across the truss to the lift before running out of power so the unit could be lowered to the ground and checked for problems. The batteries are wired to not provide any power to the attached detector so that the detector doesn't drain the batteries if the supply is interrupted.

9.9 Electronics and Control System

Aside from the brake system and the reduced height, that largest redesign to the trolley from the initial prototype was the new electronics. As stated previously, the control system was mostly large, handmade circuits driven with a PIC microcontroller and small RF transmitter. Aside from the limited range, the transmitter could only provide eight usable commands and didn't allow two way communication. Wireless control was a goal from the beginning and due to the fact that deployment would most likely include a roadside Wi-Fi base station, standard Wi-Fi was chosen to provide communication. This coupled with the desire for more flexible programming options led to the use of a Rabbit microprocessor with Wi-Fi capabilities at the center of the new control system. The remaining electronics consist mostly of optical isolators, commercial h-bridges for bi-directional control of all the motors, power supply circuits, connectors, and a relay to control detector power. The relay can be turned on or off by the operator remotely. This was all designed to reside on a single circuit board housed within a weather proof box smaller than our 3 inch height envelope. The backup batteries are external to the electrical box and will power the system in the case of an power interruption.

The microprocessor was programmed in C with all the commands being carried out within their own functions. The functions include: system initialization; communication setup; forward and backward motion; setting the speed; setting and checking the brakes; turning power to the detector on and off; ramping the speed up and down; and other internal functions for timing and communication. The drive motor uses a PWM signal for speed control. The ramping of the drive speed smooths the motion of the system to avoid jerking the detector when starting and stopping. Before motion is initiated, the system checks the brake status to make sure they are not set, and it unsets them if necessary. These are the basic commands and could be expanded as more functionality is desired. The biggest advantage of the new control system is bidirectional communication, and with it the ability to send messages back to the user. Communication is handled with two hexadecimal digits that represent a series of predetermined commands and messages referred to as opcodes.

The program written to control the trolley creates a server on the host computer to initiate and maintain a TCP connection with the trolley. The program and the trolley are both set up to connect to a managed Wi-Fi network as apposed to an ad-hoc network. This means that the trolley will always connect to the same network and multiple trolleys can use the same network if deployed at the same location. Once the connection is established, the server program can send and receive opcodes and display messages corresponding to the meaning of an incoming code. A list of the opcodes for commands, messages, and errors is defined in the *control.h* file of the trolley's control program and can be seen in the appendix. A GUI interface is used so that an operator at the roadside can easily position the trolley without needing to type or remember commands. The program is written on a Linux system and uses the GTK libraries for its GUI. It has also been compiled on a Mac system with minor modifications. Other platforms that support the GTK libraries should be able to run the program if all the other necessary libraries are loaded. Testing with the trolley might need to be done to make sure the opcodes are sent correctly. Differences between versions of the same libraries could lead to errors such as opcodes not being sent in the correct order.

Figure 9.16: Control system

9.10 Future Work and Improvements

There are still many improvement that can be made to the trolley and its systems. As it's developed further, some of the parts might be changed completely. These suggestions are for the next step and assume that most components will remain the same.

First a system to check for power delivery from the tracks to the trolley should be implemented. When the system is installed, the entire track or one section might have a problem with electrical connections. Some part of the trolley electronics could be designed to detect an interruption on the incoming power and trigger a message to the operator. With the battery backup in place, the trolley could then be driven along the entire length of the tracks and test if the power supply is working everywhere.

Along the same lines, the aluminum tape that was added to the inside of tracks to provide power should be subjected to some extended tests to determine wear characteristics. Even though the trolley was not really aimed at constant motion, continued use might wear through the 5 mil tape. The bronze brushes might also dig into the plastic insulation layer beneath the conductive tape causing a short. Since repair work would entail climbing over the road, knowing these characteristic is important.

Changes to the power delivery brushes should be considered to help prevent a short. Over time the bristles of brushes tend to spread and get bent. Unless the insulation is extended to all inside surfaces of the track, there's a possibility of the brushes grounding the supply. Placing a band around the bristles to hold them in place or making the brush much skinnier than the insulation width would both help prevent this.

The brake system would benefit from an adjustment to the brake arm geometry. The current brake arm has a rounded end so that regardless of the exact distance from the tracks, the contact area would be the same. This could present a problem in that the arm can get jammed into the tracks like a cam if the trolley is somehow pushed. To avoid this, the brake arm could be machined to have a flat that would rest against the bottom of the tracks.

At this point the weight of the trolley has not been optimized. This was not important for the prototype. The final version would have increased safety margins if the weight of the trolley could be reduced. Primarily the body and mounting plate could be optimized as there are undoubtedly areas that don't carry much of the stress.

Before the trolley is fully deployed, several components need to be weatherproofed. Moisture will probably be the biggest issue. The circuit board, batteries, and bearings are all sealed. The motors will need to be waterproofed, but because they are used intermittently it might be possible to seal them a silicone caulk.

To add more intelligence to the system, some additional sensor could be added. An encoder on the drive motor or on the idle shaft would allow positioning to be done with commands of distance instead of visually. Given the roughness of the positioning necessary, the encoder on the motor would be overkill making the drive shaft or the idle shaft the more appropriate location. Another helpful sensor would be a tilt sensor on the trolley body. It is important that the lift platform remain level during the alignment procedure when being raised. A tilt sensor would make sure that detectors were connected properly so that the system is correctly balanced on the lift.

Outside the trolley, but perhaps the most important improvement needed is a safety backup on the lifting mechanism. The conditions have always been controlled during our test. Even so, there was always a backup rope that runs over the truss and is tied to the lift platform in case the lift cable broke. We did break the lift cable once during testing due to a large imbalance of the trolley and detector on the lift. In a real installation, a safeguard needs to be in place to prevent a catastrophic failure. Early attempts at this included a clutch that would slip before the breaking point of the cable and a pin designed to break before the cable. Neither were carried into the current lift design, but some type of mechanism needs to be considered.

CHAPTER 10

CONCLUSION

Two robust and reliable Lifting Mechanisms have been developed and built from a novel alignment idea. The trolley-detector LM has been equipped with all-weather shielding, and is ready for deployment on the signage truss being built for field testing over I-80 in Sacramento. The trolley-detector LM is designed to lift 75_{lbf} and has shown its ability to lift over 200_{lbf}. The trolley-detector LM has completed an endurance test in +100°F direct sun conditions. The LM lifted 105_{lbf} over 100 times in a four hour time period. It also has lifted the trolley-detector package flawlessly for multiple mobile truss test days. The camera LM has also proved a degree of reliability many times during testing and presentations. The two LMs combined have established a degree of reliability for the design by lifting an estimated 300 consecutive, successful cycles.

A mobile truss has been designed and built for testing detectors, trolleys, and LMs. It has proven its ability to hold over 1500_{lbf} unevenly distributed across its span; its typical loading during field testing is under 200_{lbf} evenly distributed. The mobile truss was used for a large and successful Caltrans presentation on October 14, 2004. This presentation was held on Old Hutchison Road just west of Hwy 113 on the UC Davis campus.

The current iteration of the universal mounting platform (the trolley) is ready for further refinements and extended field test. The major project goals have been reached. As designed the system will provide increased safety when dealing with many types of detection systems that need to be installed overhead. Existing structures can be used to deploy the trolley and detectors. Only certain locations have a truss that extends the entire width of the freeway. However, many of these locations are the same high traffic areas where new detection systems are needed to provide better data for monitoring and planning purposes. To make the trolley a universal platform, the only requirement is an individually designed mounting plate for each type of detector. With this, any detector system that needs to be positioned directly over traffic can be accommodated. Finally, communication and power delivery is done wirelessly. With the extension of the network that is used for communication and additional feedback to the operator, offsite operation could easily be achieved.

Figure 10.1: Entire system on lift

References

Cheng, H.H., Shaw, B., Palen, J., Larson, J.E., Hu, X.D. and Van Katwyk, K., A Real-Tile Laser-Based Detection System for Measurement of Delineations of Moving Vehicles", IEEE/ASME Transactions on Mechatronics, Vol. 6, No. 2, June 2001.

Duane, J., "Mobile Platform for Overhead Detector of Road Vehicles," M.S. Thesis, University of California, Davis, 2001.

Duane, J., Palen, J., Eke, F., Cheng, H., "Design of a Mobile Platform For Overhead Detectors for Vehicles on the Highway", CD-ROM Proc. of the ASME 28th Mechanisms and Robotics Conference, paper # DETC2004-57557, Salt Lake City, Utah, September 28 - October 2, 2004.

Glover, Thomas J., Pocket Reference, 3rd Edition: 7th Printing, Sequoia Publishing Inc., August 2003.

Integrated Engineering Laboratory Home Page, http://iel.ucdavis.edu, 2005.

Lin, B., Cheng, H., Shaw, B., Chen, B., Palen, J., "Mechanical and Optical Design of a Laser-Based Non-Intrusive Vehicle Delineation Detection System", ASME, 2001.

Malika, J., Russell, S., Measuring Traffic Parameters Using Video Image Processing", Intellimotion, Vol. 6 No. 1, pp. 6-7, 12-13, http://path.berkeley.edu, 1997.

McMaster-Carr, Plastic Material Comparisons, http://www.mcmaster.com/param/html/C109AboutPlastics. July, 2004.

Palen, J., "The Need for Surveillance in Intelligent Transportation Systems", Intellimotion, Vol. 6 No. 1, pp. 1-3, http://path.berkeley.edu, 1997.

Sun, C. and Ritchie, S.G., "Individual Vehicle Speed Estimation Using Single Loop Inductive Wave Forms", Journal of Transportation Engineering, November/December 1999.

Precision Electronic Opto-Mechanical System for Vehicle Delineation Detection on Highway", ASME Journal of Mechanical Design, Vol. 125, pp. 802-808, December 2003.

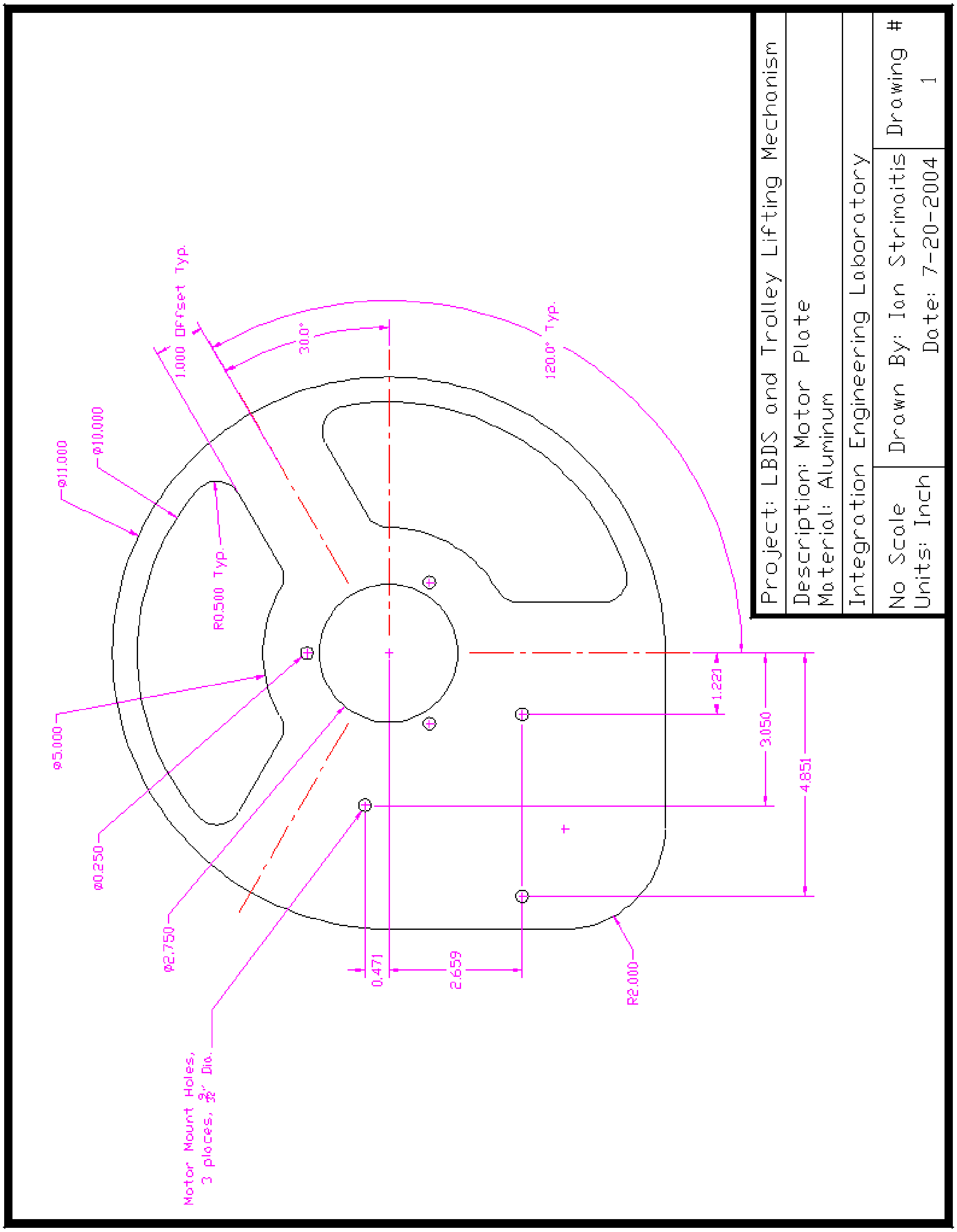
Wang, Z., Nestinger, S., Cheng, H., Shaw, B., "Real-Time Architecture for an Electro-Mech-Optical System for Detection of Vehicles on Highway", paper # DETC2004-57750, Salt Lake City, Utah, September 28-October 2, 2004.

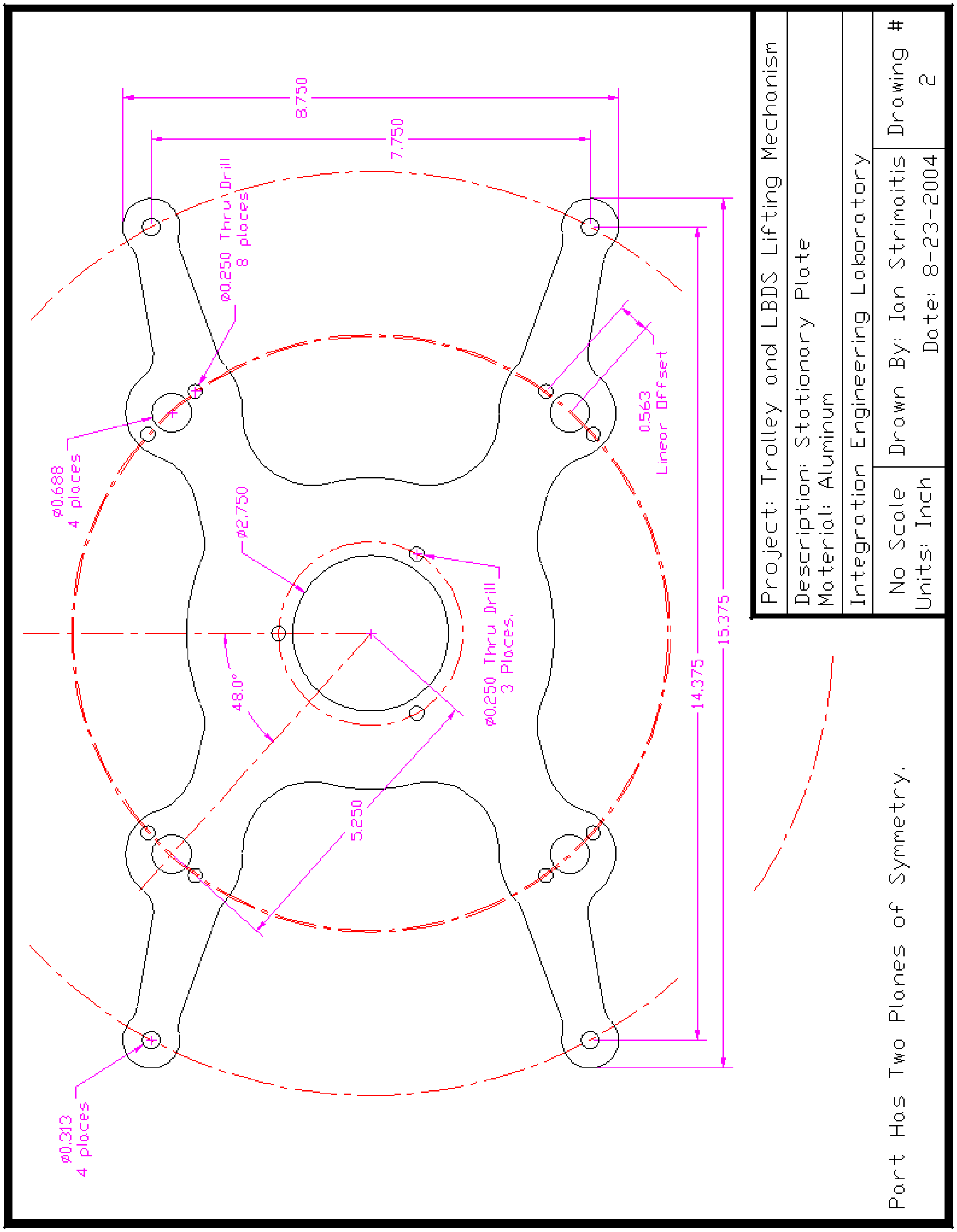
Werner Co., How to choose and use a Werner Ladder, http://www.wernerladder.com, 2005.

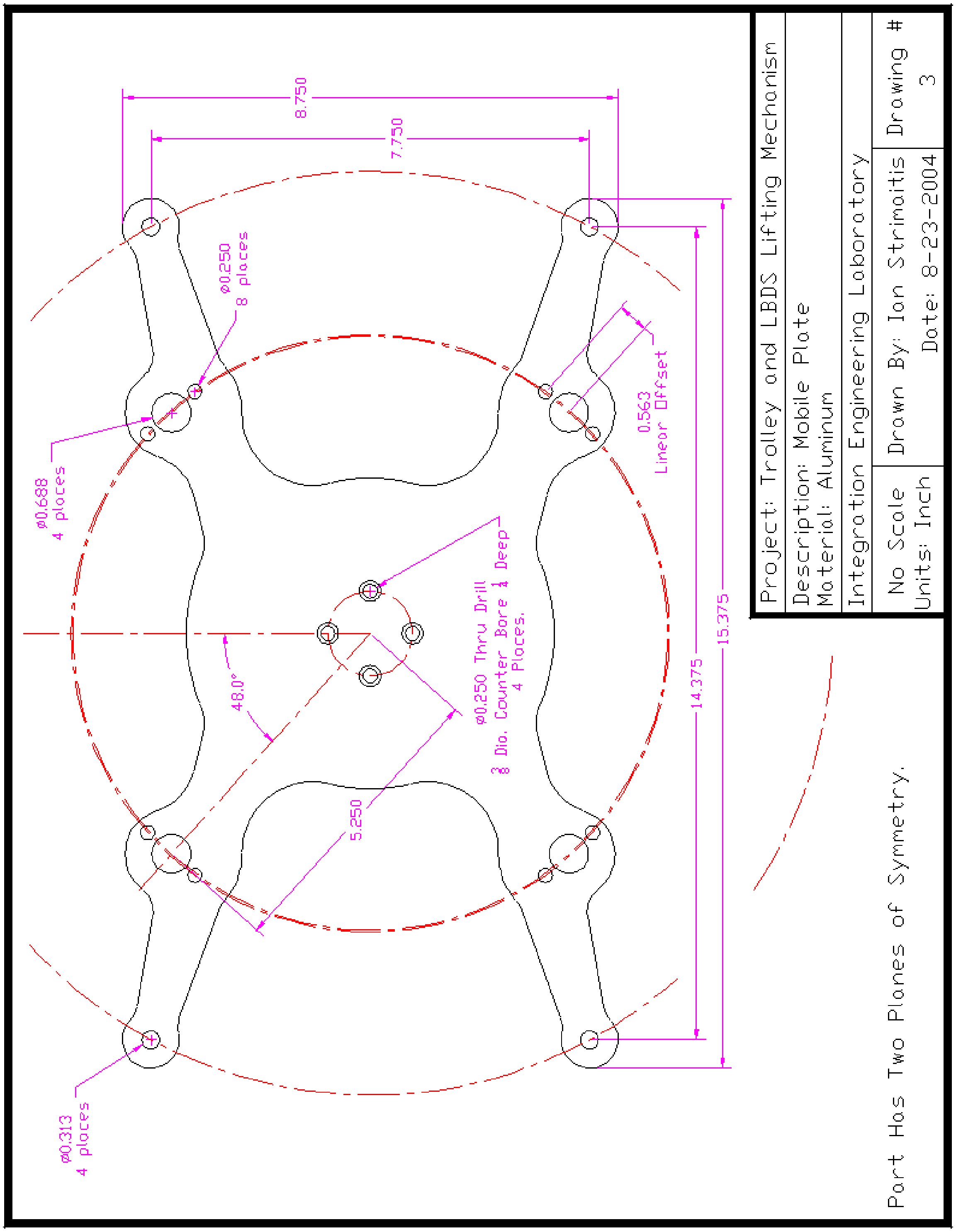
W.W. Grainger, Inc., Grainger Industrial Supply Catalog and Gear Motor Selection Guide, 1999.

Yu, Q., Eke, F., Cheng, H., Duane, J., Palen, J., "Control of a Mobile Support Platform for Vehicle Detectors on Highway", CD-ROM Proc. of the ASME 24th Computers in Engineering Conference, paper # DETC2004-57706, Salt Lake City, Utah, September 28 - October 2, 2004.

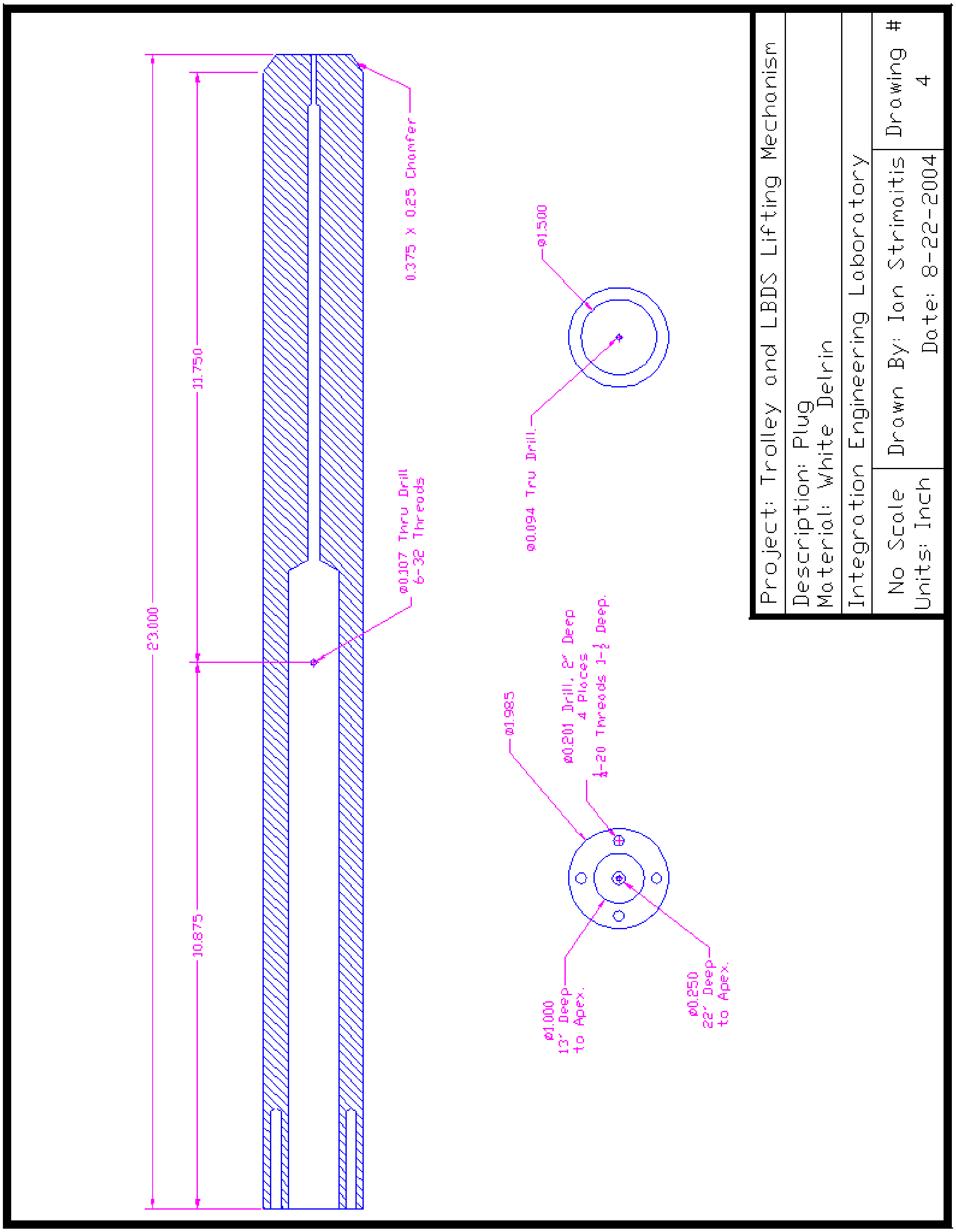
APPENDIX I, ABDS-TROLLEY LM CONSTRUCTION DRAWINGS



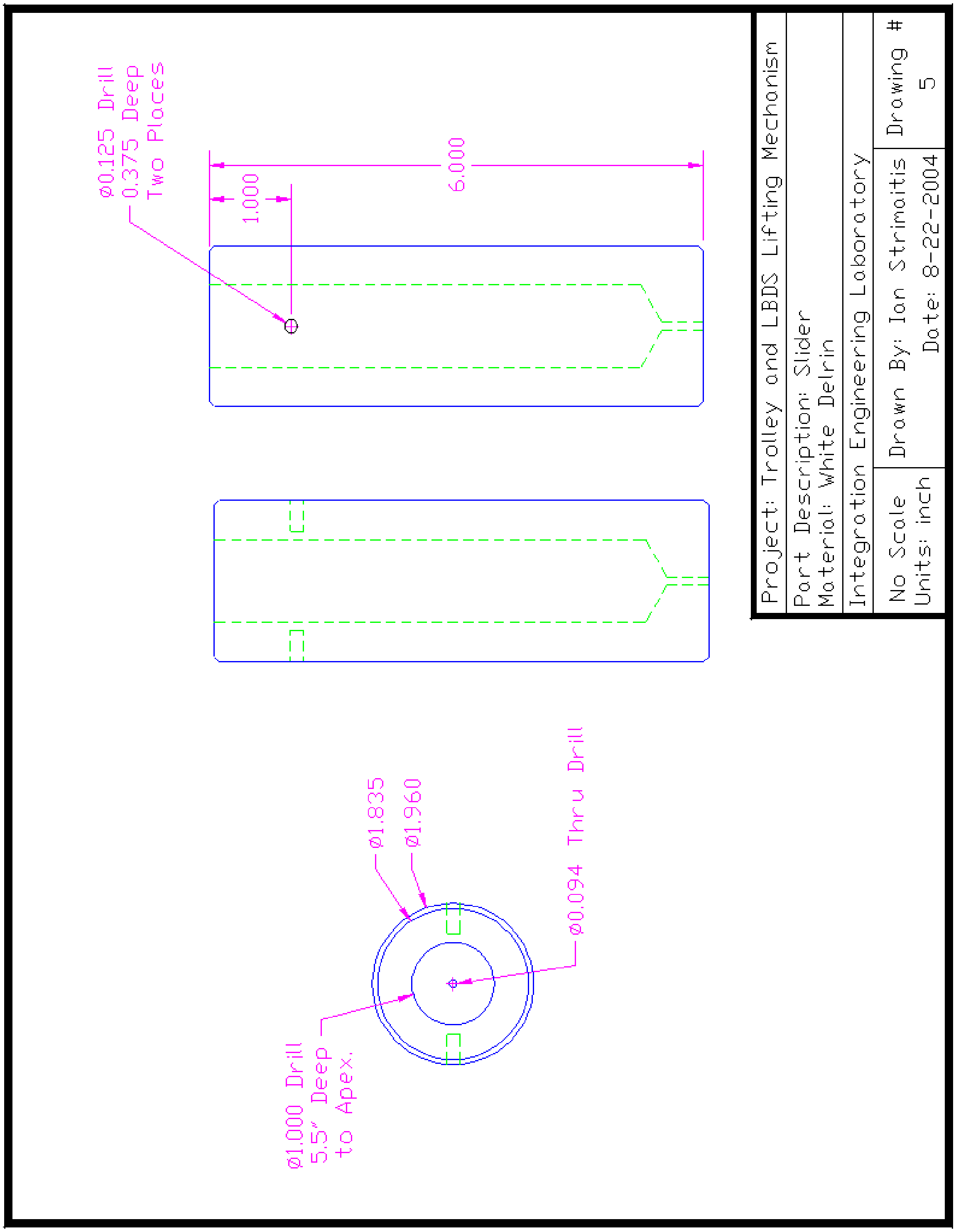

Project: Trolley and LBDS Lifting Mechanism
Description: Stationary Plate
Material: Aluminum
Integration Engineering Laboratory
No Scale
Units: Inch
Drawn By: Ian Strimaitis
Date: 8-23-2004
Drawing #
2
Part Has Two Planes of Symmetry.
8.750
7.750
15.375
14.375
5.250
48.0°
0.563
Linear Offset
Ø0.250 Thru Drill
8 places
Ø0.688
4 places
Ø0.313
4 places
Ø2.750
Ø0.250 Thru Drill
3 Places.

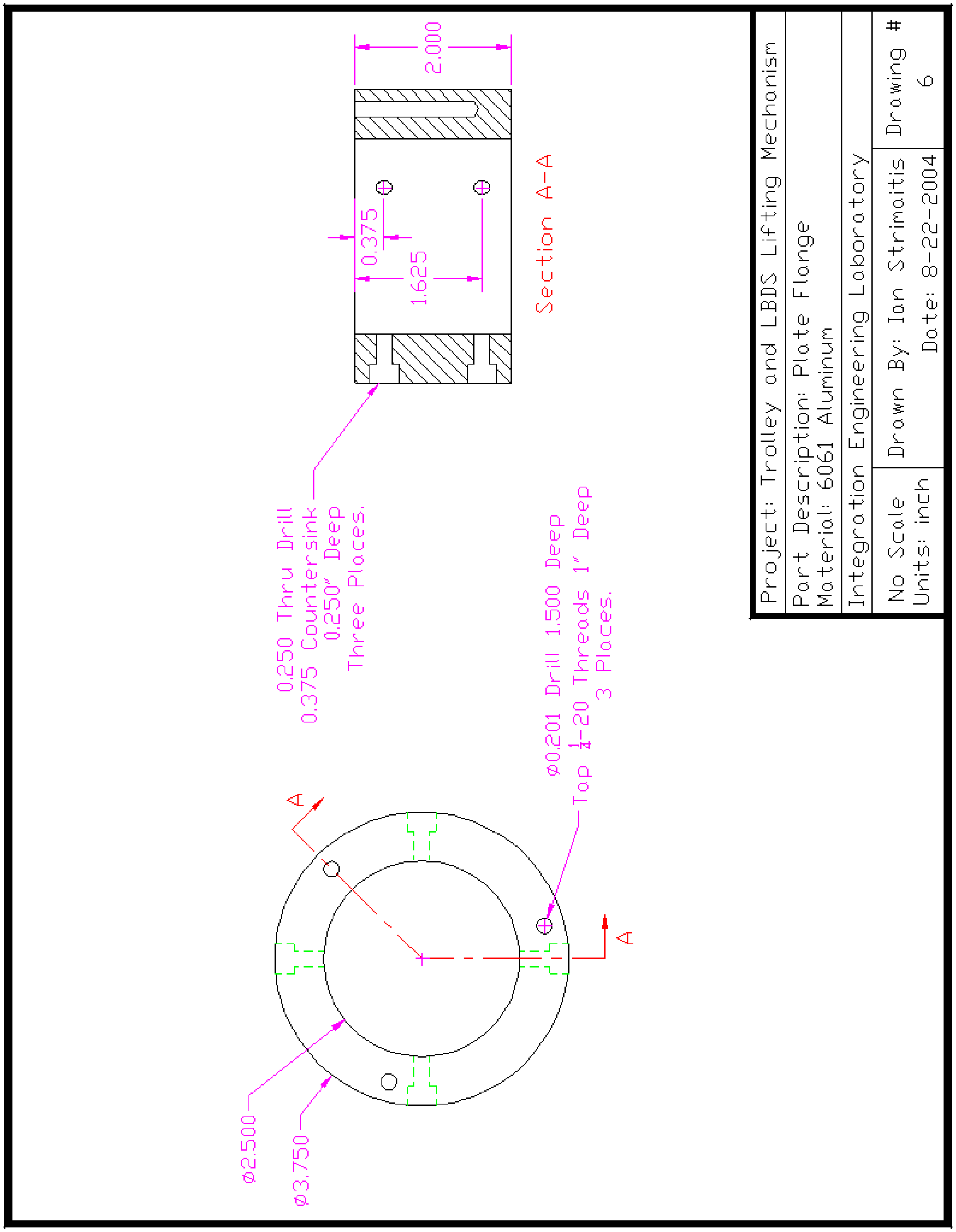


Project: Trolley and LBDS Lifting Mechanism
Description: Mobile Plate
Material: Aluminum
Integration Engineering Laboratory
No Scale
Units: Inch
Drawn By: Ian Strimaitis
Date: 8-23-2004
Drawing #
3
Part Has Two Planes of Symmetry.
8.750
7.750
ø0.250
8 places
ø0.688
4 places
0.563
Linear Offset
ø0.250 Thru Drill
3/8 Dia. Counter Bore 1/4 Deep
4 Places.
48.0°
5.250
14.375
15.375
ø0.313
4 places

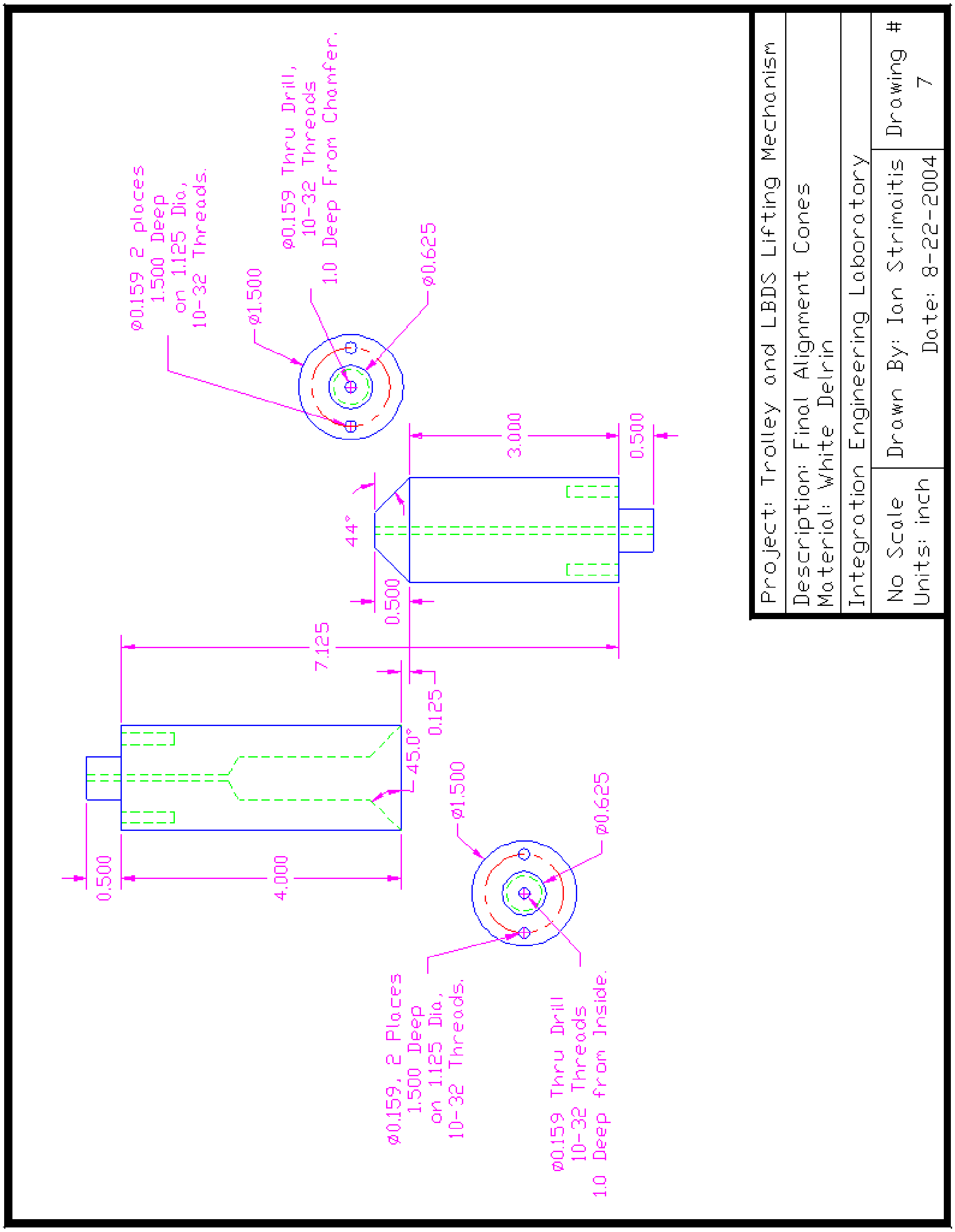


23.000
11.750
10.875
0.375 x 0.25 Chamfer
Ø0.107 Thru Drill
6-32 Threads
Ø1.500
Ø0.094 Tru Drill
Ø1.985
Ø0.201 Drill, 2" Deep
4 Places
1/4-20 Threads 1-1/2 Deep.
Ø1.000
13' Deep
to Apex.
Ø0.250
22' Deep
to Apex.
Project: Trolley and LBDS Lifting Mechanism
Description: Plug
Material: White Delrin
Integration Engineering Laboratory
No Scale
Units: Inch
Drawn By: Ion Strimaitis
Date: 8-22-2004
Drawing #
4

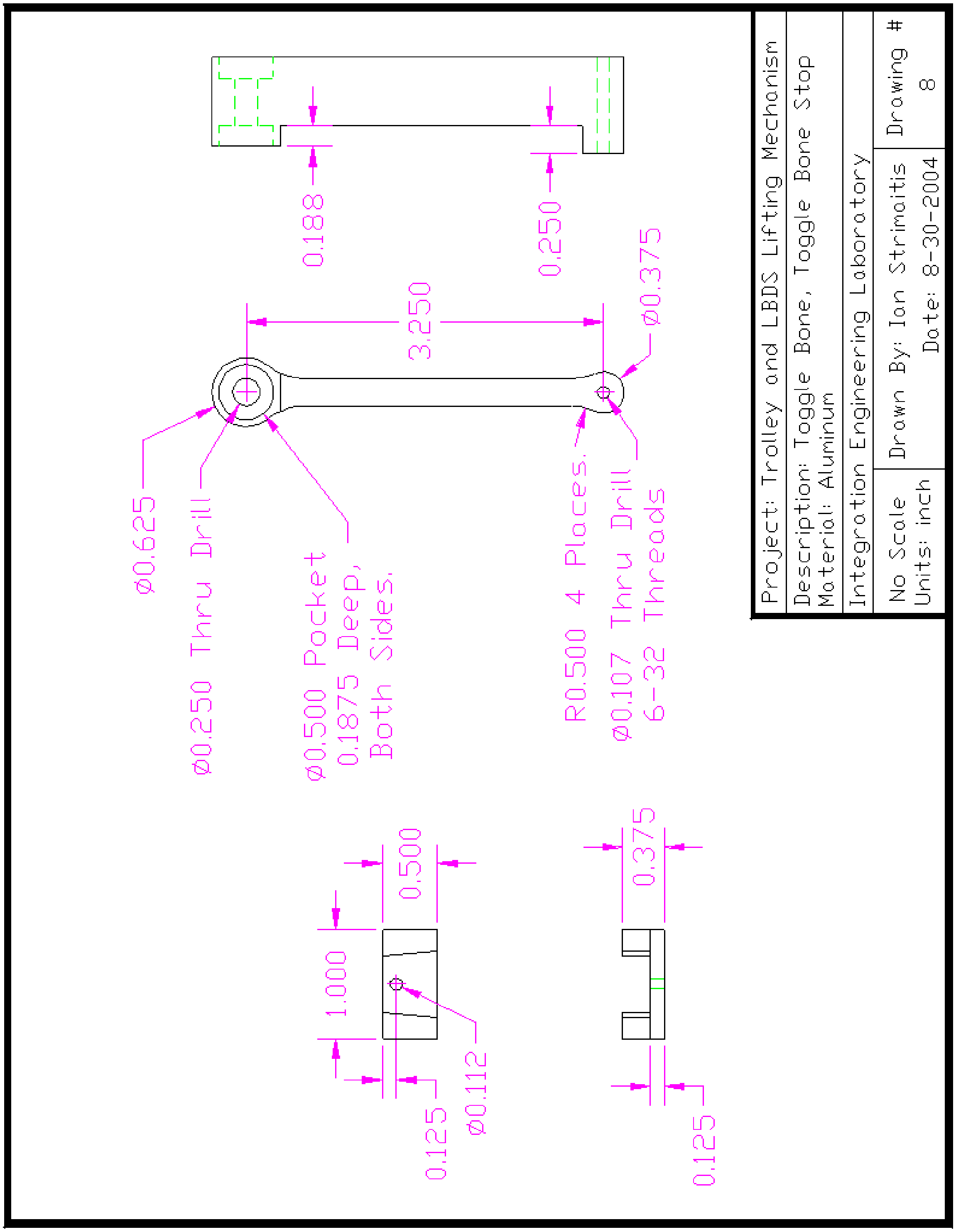

Ø0.125 Drill
0.375 Deep
Two Places
1.000
6.000
Ø1.835
Ø1.960
Ø0.094 Thru Drill
Ø1.000 Drill
5.5" Deep
to Apex.
Project: Trolley and LBDS Lifting Mechanism
Part Description: Slider
Material: White Delrin
Integration Engineering Laboratory
No Scale
Units: inch
Drawn By: Ian Strimaitis
Date: 8-22-2004
Drawing #
5

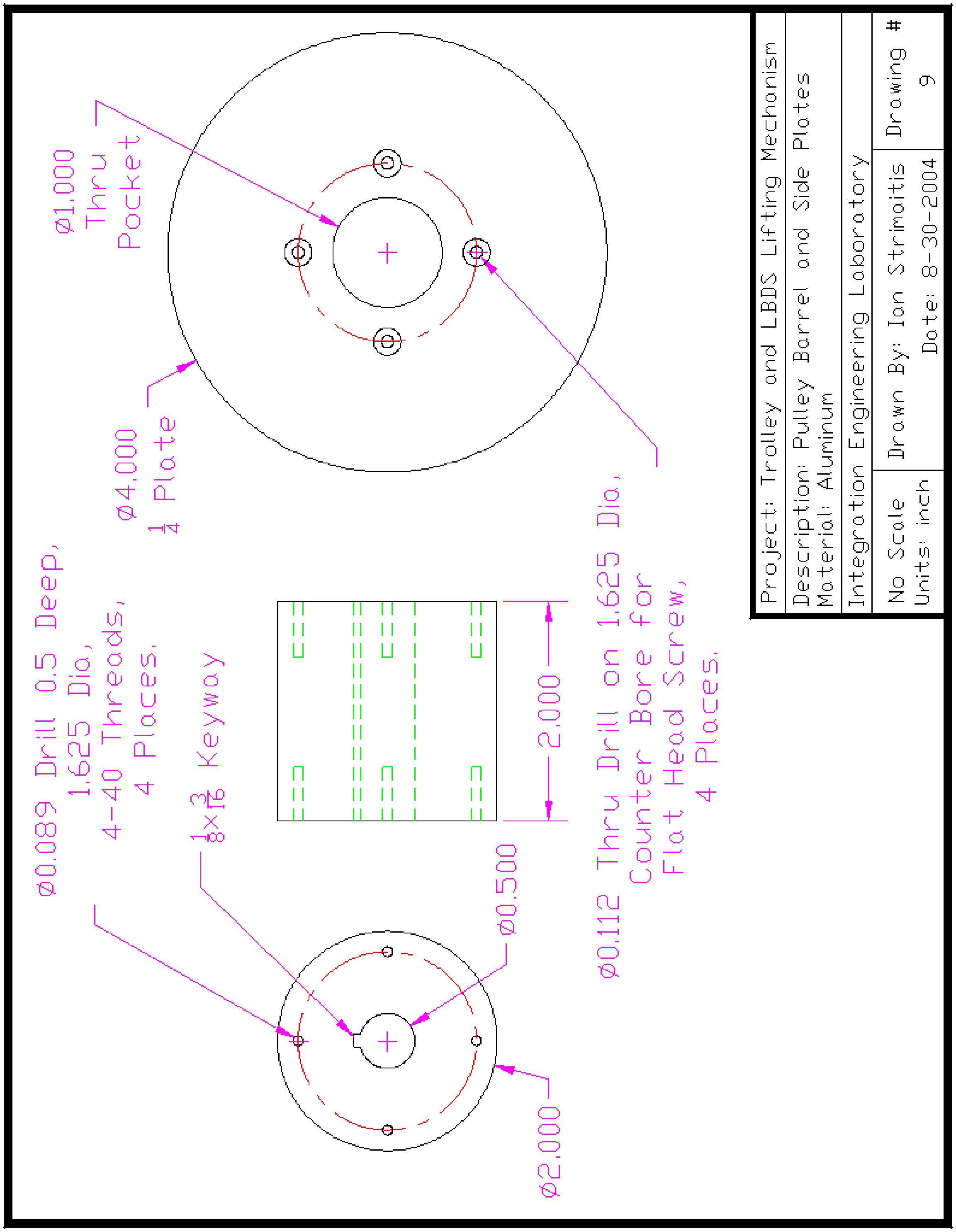


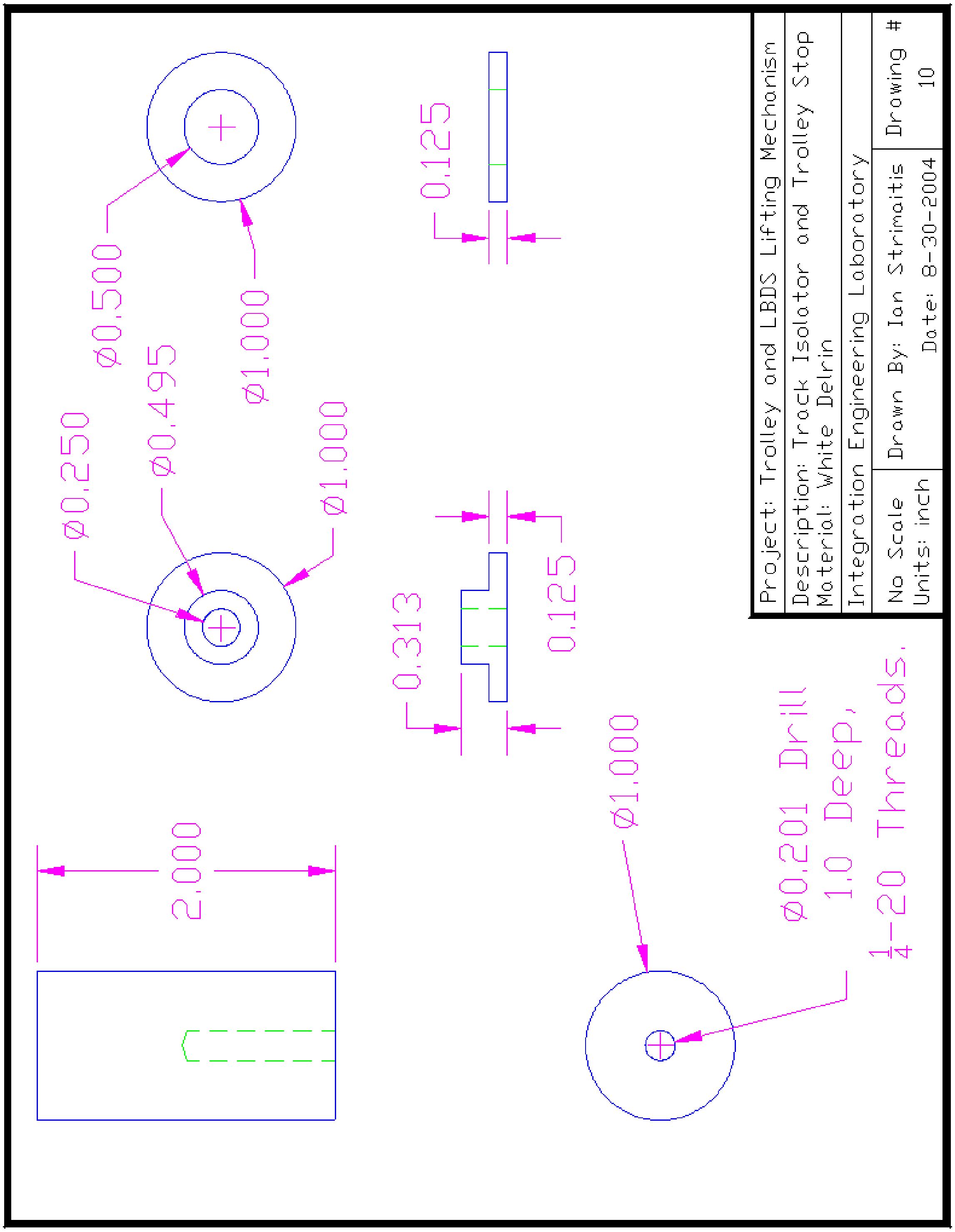
Project: Trolley and LBDS Lifting Mechanism
Part Description: Plate Flange
Material: 6061 Aluminum
Integration Engineering Laboratory
No Scale
Units: inch
Drawn By: Ian Strimaitis
Date: 8-22-2004
Drawing #
6
2.000
0.375
1.625
Section A-A
0.250 Thru Drill
0.375 Countersink
0.250" Deep
Three Places.
Ø0.201 Drill 1.500 Deep
Top ¼-20 Threads 1" Deep
3 Places.
A
A
Ø2.500
Ø3.750

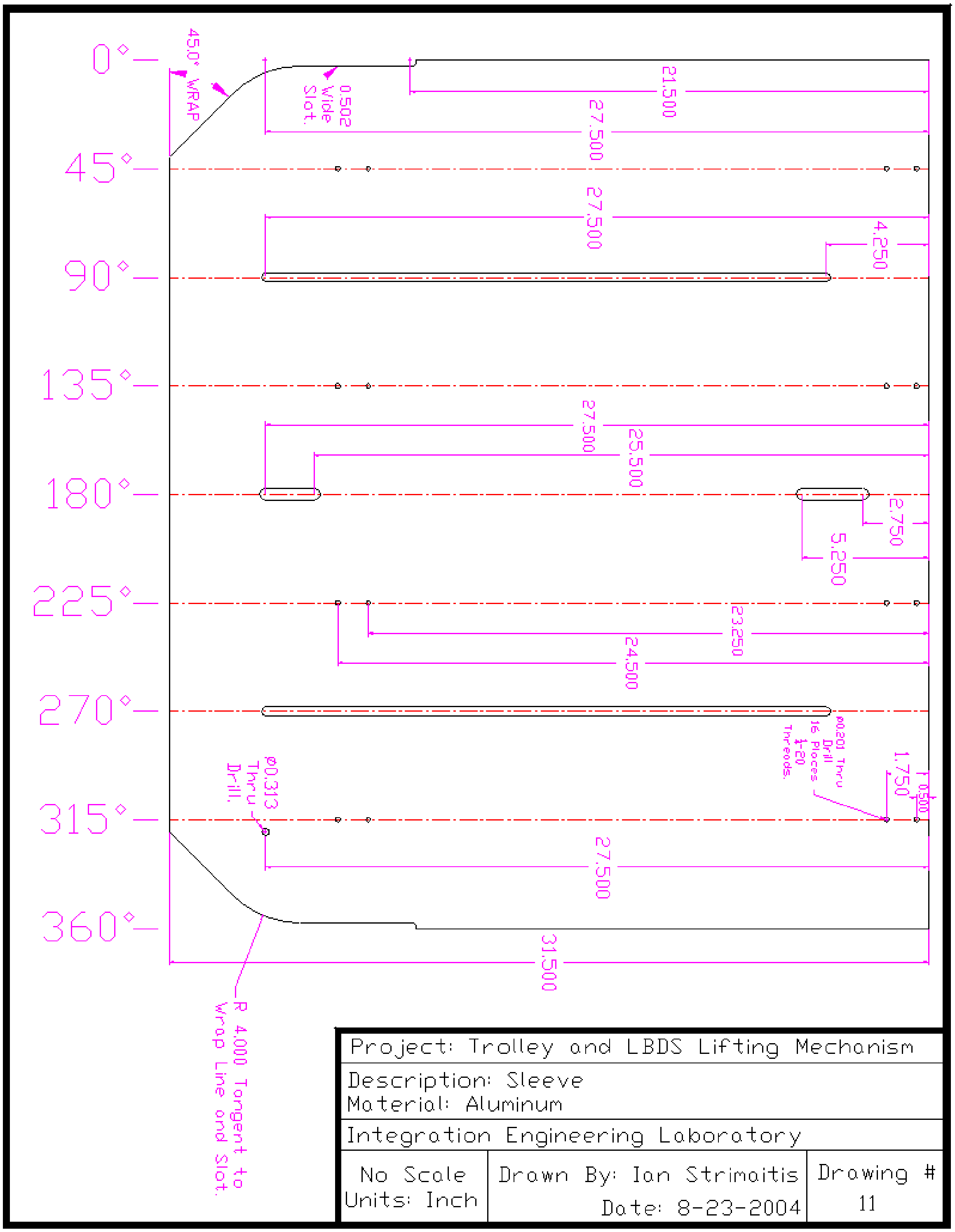


Ø0.159 2 places
1.500 Deep
on 1.125 Dia,
10-32 Threads.
Ø1.500
Ø0.159 Thru Drill,
10-32 Threads
1.0 Deep From Chamfer.
Ø0.625
3.000
0.500
44°
0.500
7.125
0.125
45.0°
Ø1.500
Ø0.625
0.500
4.000
Ø0.159, 2 Places
1.500 Deep
on 1.125 Dia,
10-32 Threads.
Ø0.159 Thru Drill
10-32 Threads
1.0 Deep from Inside.
Project: Trolley and LBDS Lifting Mechanism
Description: Final Alignment Cones
Material: White Delrin
Integration Engineering Laboratory
No Scale
Units: inch
Drawn By: Ian Strimaitis
Date: 8-22-2004
Drawing #
7


0.188
0.250
3.250
Ø0.375
Ø0.625
Ø0.250 Thru Drill
Ø0.500 Pocket
0.1875 Deep,
Both Sides.
R0.500 4 Places.
Ø0.107 Thru Drill
6-32 Threads
1.000
0.500
0.125
Ø0.112
0.375
0.125
Project: Trolley and LBDS Lifting Mechanism
Description: Toggle Bone, Toggle Bone Stop
Material: Aluminum
Integration Engineering Laboratory
No Scale
Units: inch
Drawn By: Ian Strimaitis
Date: 8-30-2004
Drawing #
8

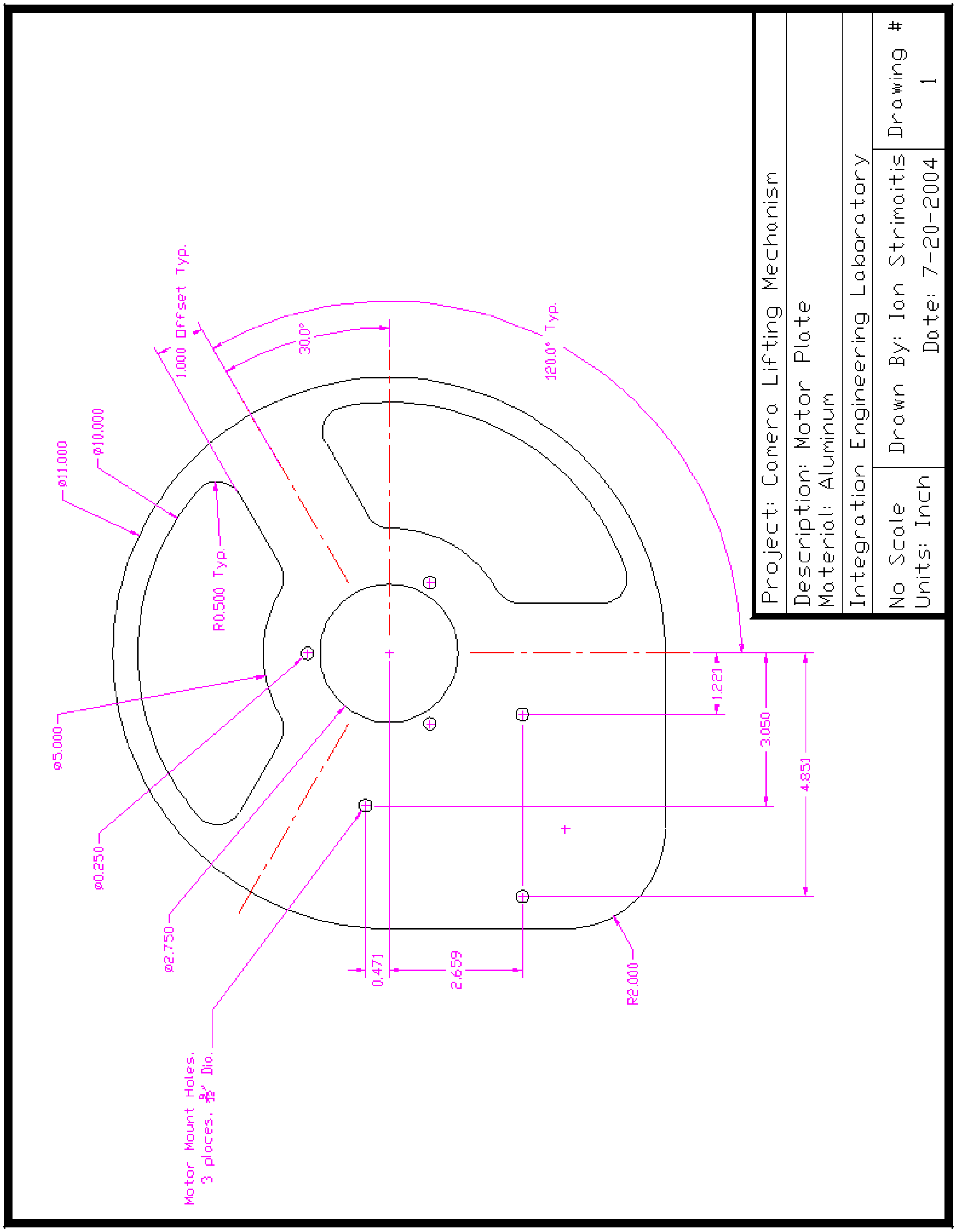

Ø1.000
Thru
Pocket
Ø4.000
1/4 Plate
Ø0.089 Drill 0.5 Deep,
1.625 Dia,
4-40 Threads,
4 Places.
1/8×3/16 Keyway
2.000
Ø0.500
Ø2.000
Ø0.112 Thru Drill on 1.625 Dia,
Counter Bore for
Flat Head Screw,
4 Places.
Project: Trolley and LBDS Lifting Mechanism
Description: Pulley Barrel and Side Plates
Material: Aluminum
Integration Engineering Laboratory
No Scale
Units: inch
Drawn By: Ian Strimaitis
Date: 8-30-2004
Drawing #
9

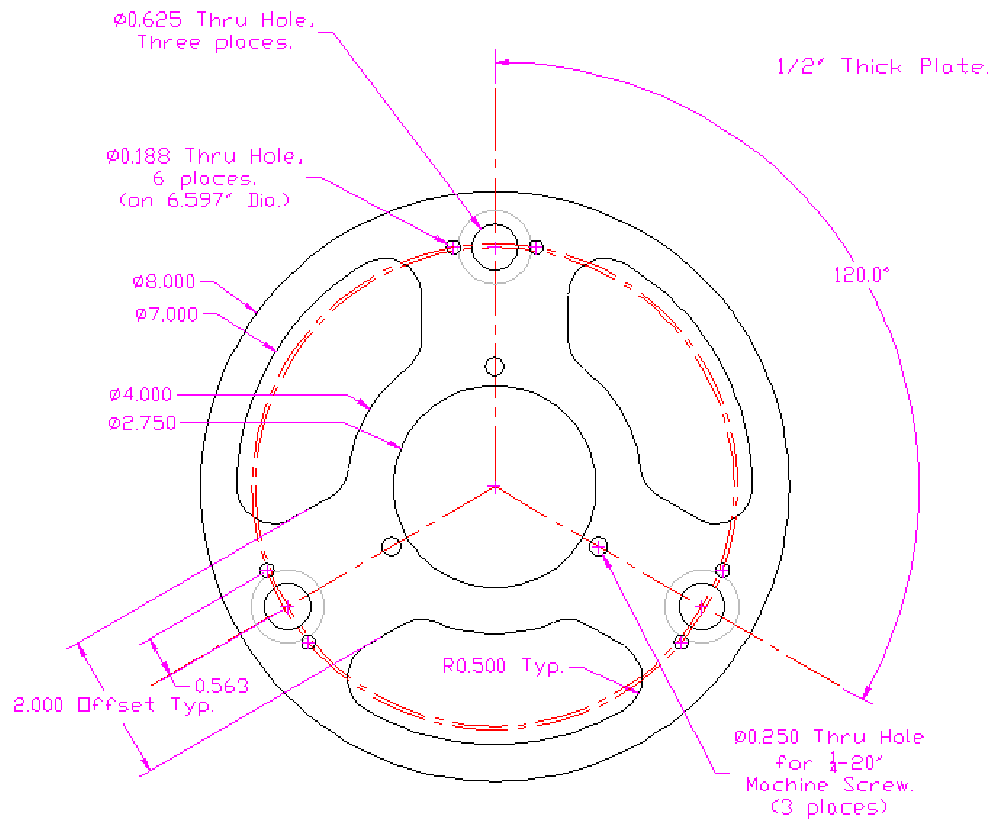

Ø0.500
Ø1.000
0.125
Ø0.250
Ø0.495
Ø1.000
0.313
0.125
2.000
Ø1.000
Ø0.201 Drill
1.0 Deep,
1/4-20 Threads.
Project: Trolley and LBDS Lifting Mechanism
Description: Track Isolator and Trolley Stop
Material: White Delrin
Integration Engineering Laboratory
No Scale
Units: inch
Drawn By: Ian Strimaitis
Date: 8-30-2004
Drawing #
10



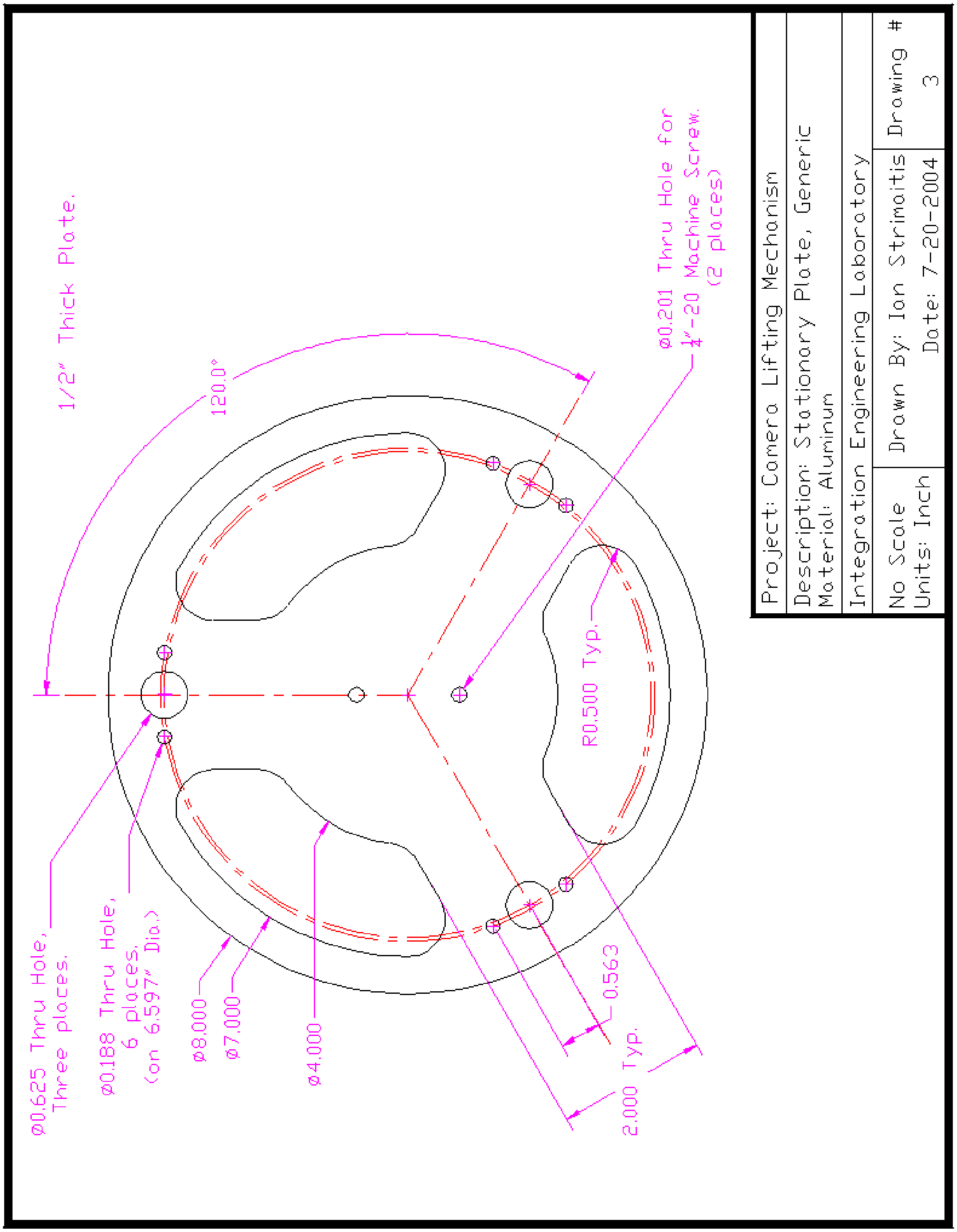
0°
45°
90°
135°
180°
225°
270°
315°
360°
45.0° WRAP
0.502 Wide Slot.
21.500
27.500
4.250
25.500
2.750
5.250
23.250
24.500
Ø0.201 Thru Drill 16 Places 1/4-20 Threads.
1.750
0.500
Ø0.313 Thru Drill.
31.500
R 4.000 Tangent to Wrap Line and Slot.
Project: Trolley and LBDS Lifting Mechanism
Description: Sleeve
Material: Aluminum
Integration Engineering Laboratory
No Scale
Units: Inch
Drawn By: Ian Strimaitis
Date: 8-23-2004
Drawing #
11

APPENDIX II, CAMERA LM CONSTRUCTION DRAWINGS

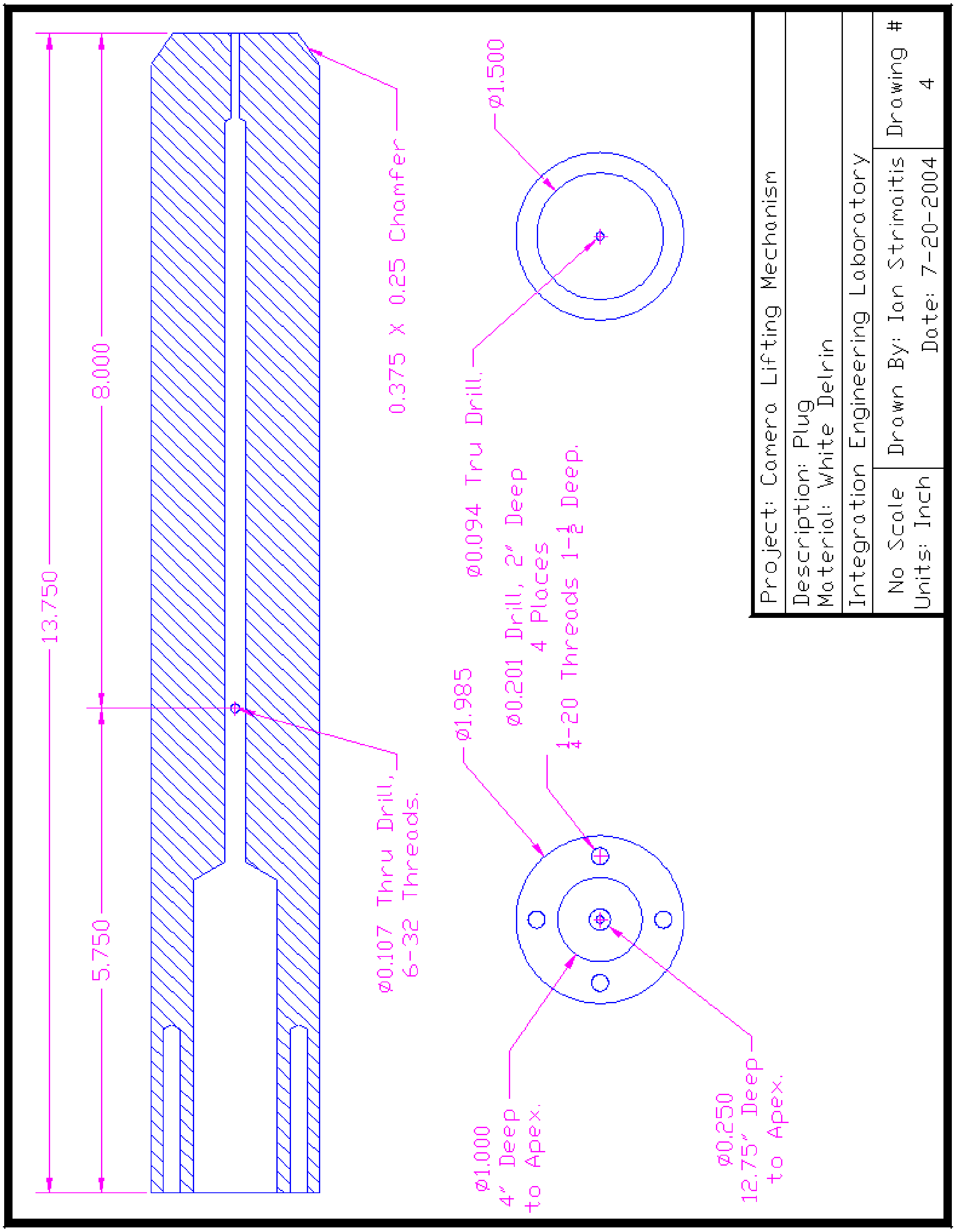




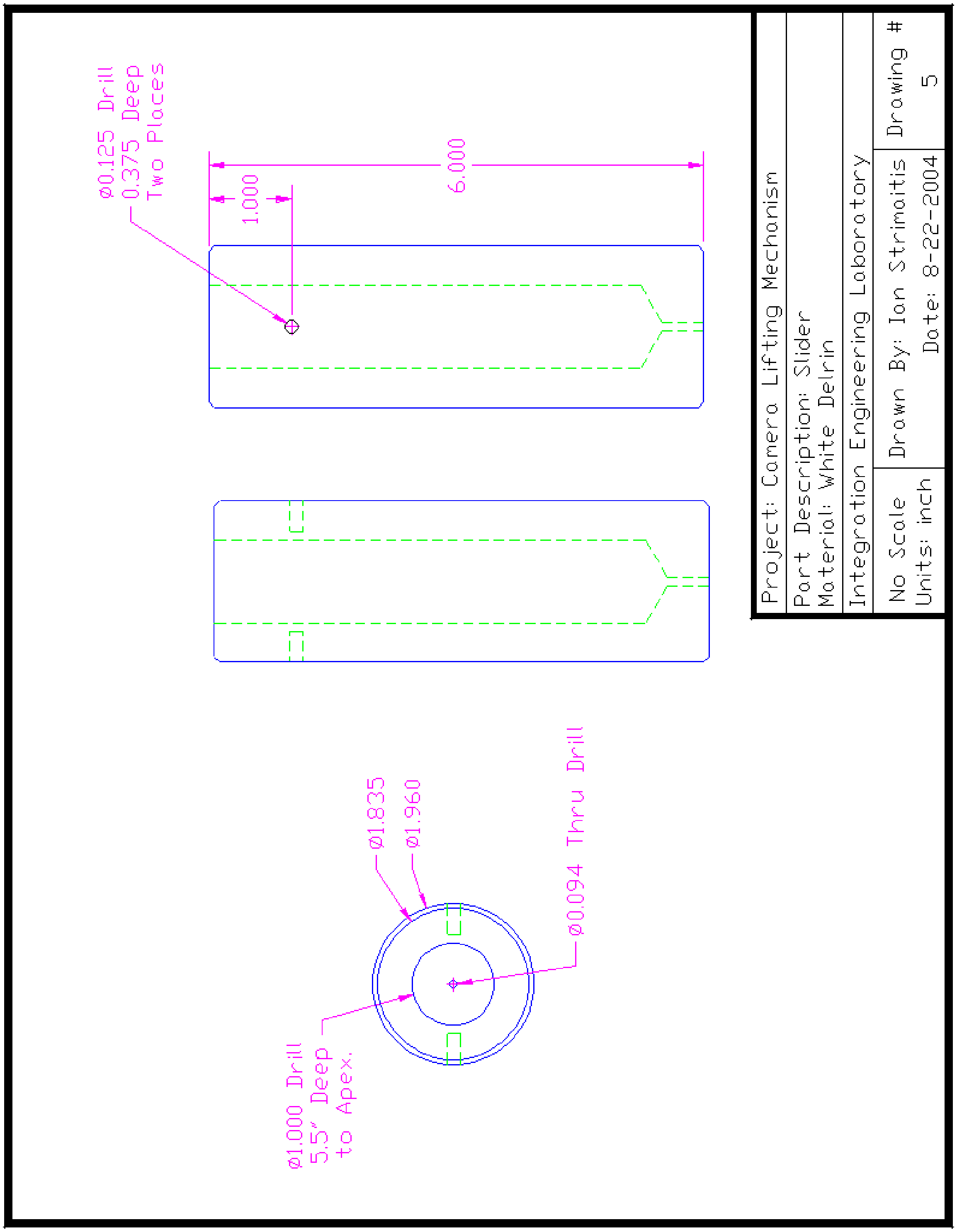
Project: Camera Lifting Mechanism		
Description: Stationary Plate, Generic Material: Aluminum		
Integration Engineering Laboratory		
No Scale Units: Inch	Drawn By: Ian Strimaitis Date: 7-20-2004	Drawing # 2



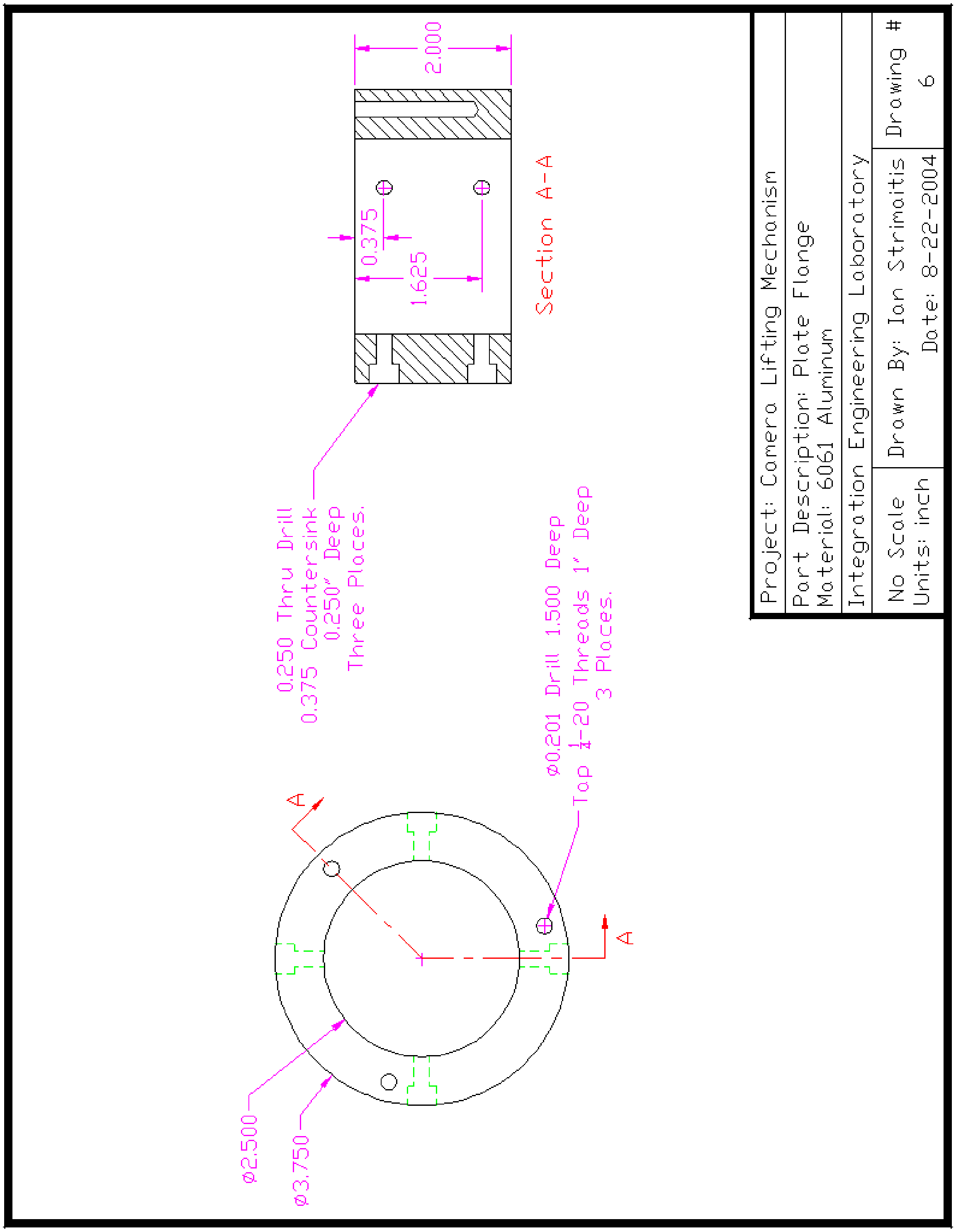
1/2" Thick Plate.
120.0°
Ø0.201 Thru Hole for
1/4"-20 Machine Screw.
(2 places)
R0.500 Typ.
Ø0.625 Thru Hole,
Three places.
Ø0.188 Thru Hole,
6 places.
(on 6.597" Dia.)
Ø8.000
Ø7.000
Ø4.000
0.563
Typ.
2.000
Project: Camera Lifting Mechanism
Description: Stationary Plate, Generic
Material: Aluminum
Integration Engineering Laboratory
No Scale
Units: Inch
Drawn By: Ian Strimaitis
Date: 7-20-2004
Drawing #
3

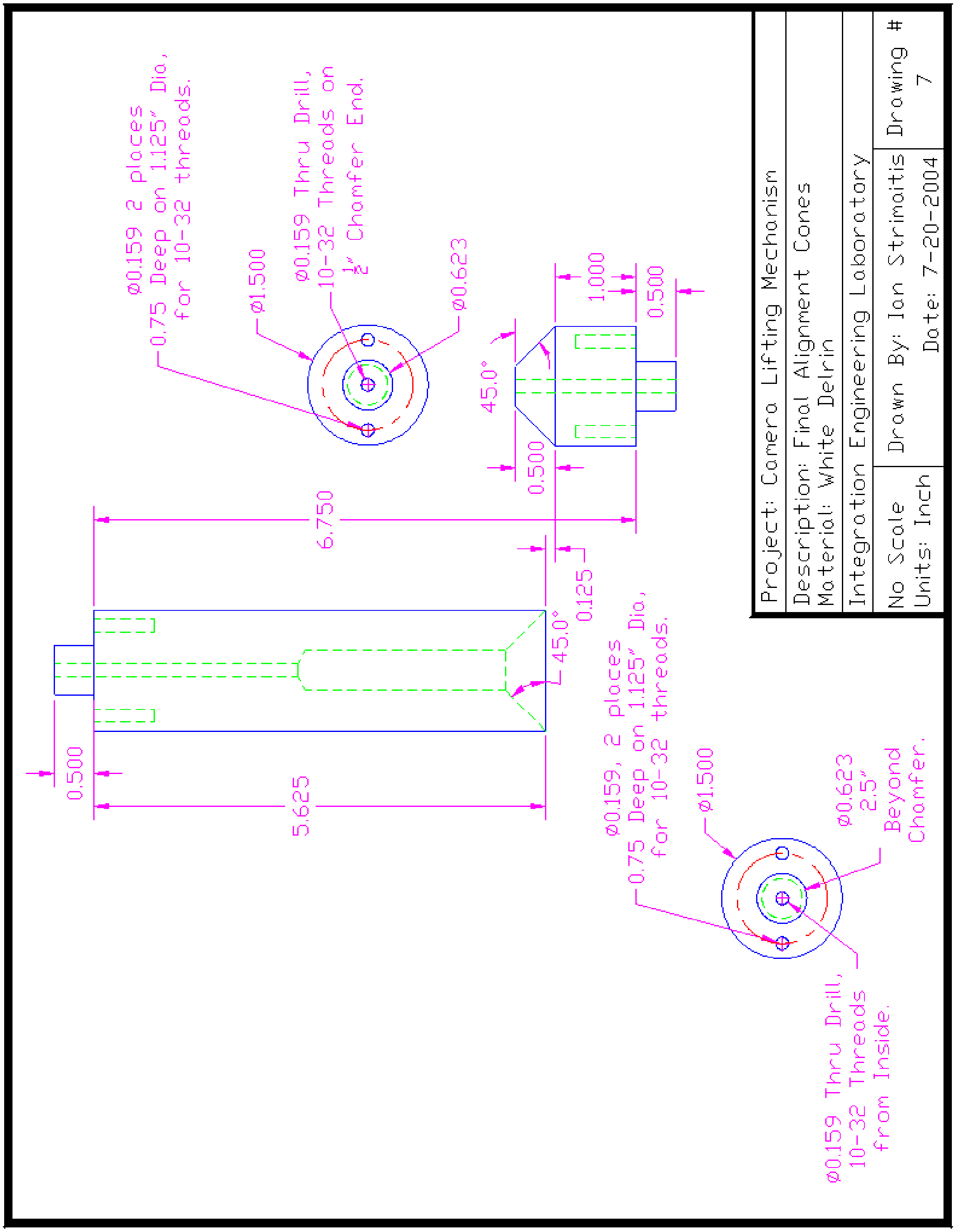


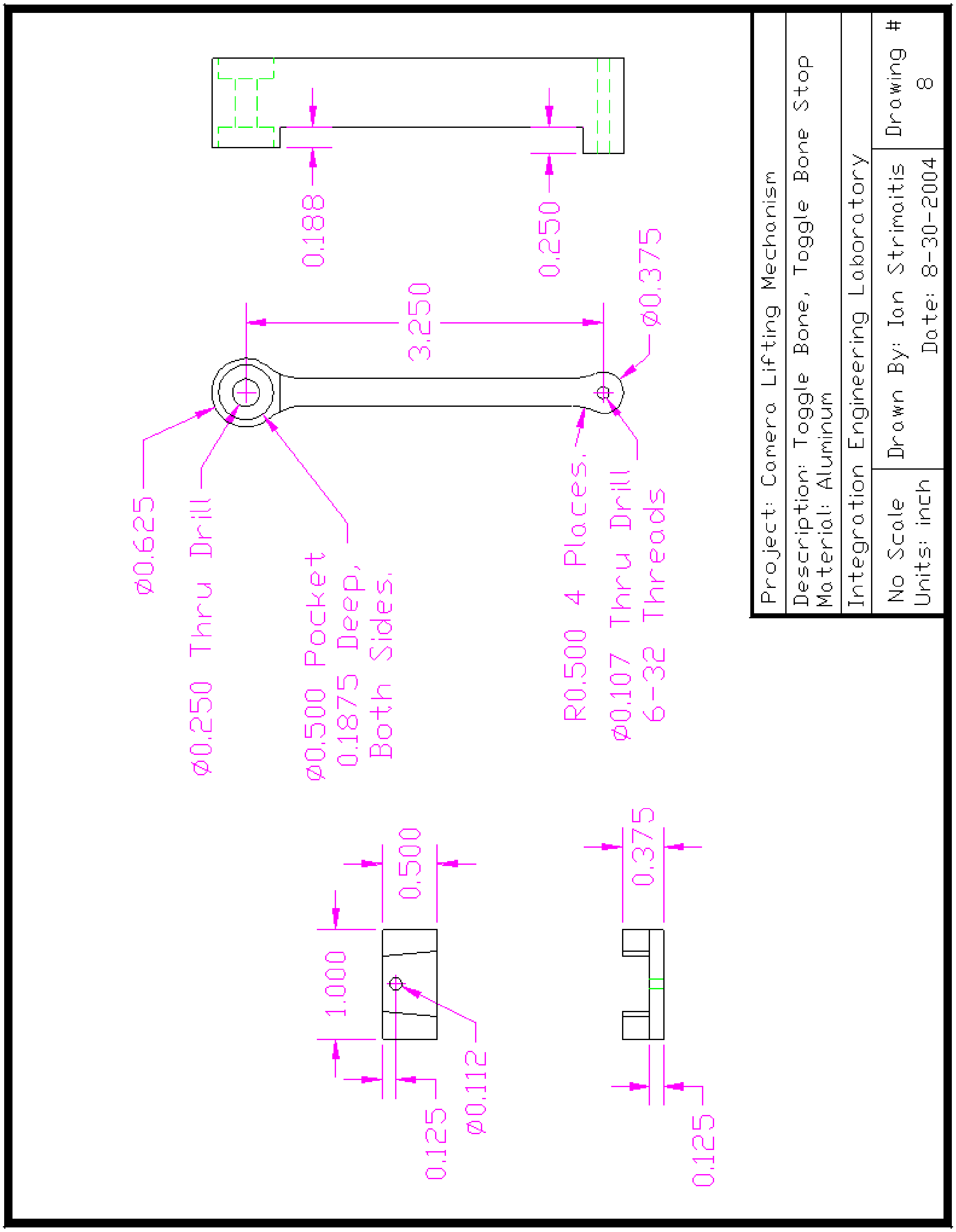
13.750
8.000
5.750
0.375 X 0.25 Chamfer
Ø1.500
Ø0.094 Tru Drill.
Ø0.201 Drill, 2" Deep
4 Places
1/4-20 Threads 1-1/2 Deep.
Ø1.985
Ø0.107 Thru Drill,
6-32 Threads.
Ø1.000
4" Deep
to Apex.
Ø0.250
12.75" Deep
to Apex.
Project: Camera Lifting Mechanism
Description: Plug
Material: White Delrin
Integration Engineering Laboratory
No Scale
Units: Inch
Drawn By: Ian Strimaitis
Date: 7-20-2004
Drawing #
4

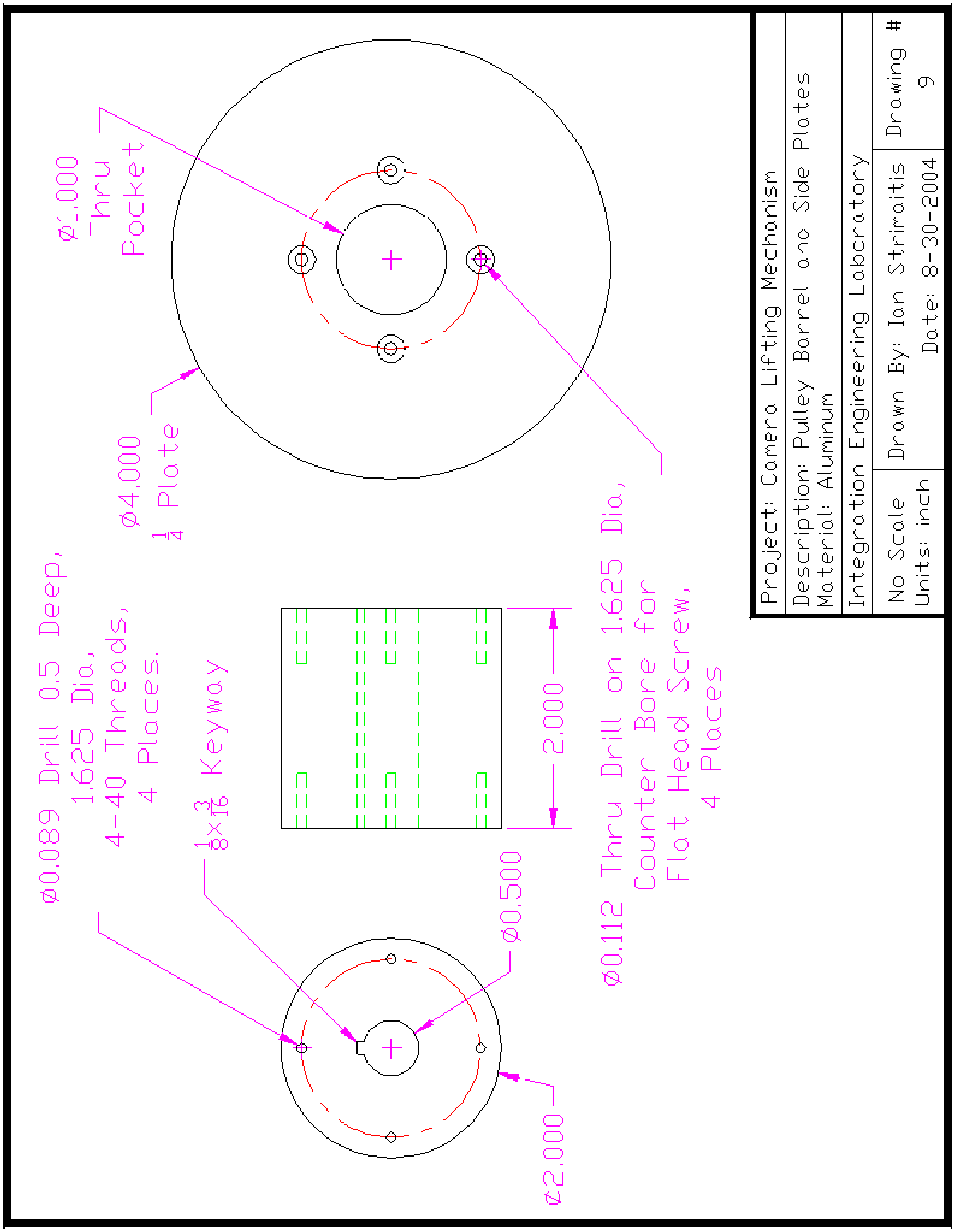


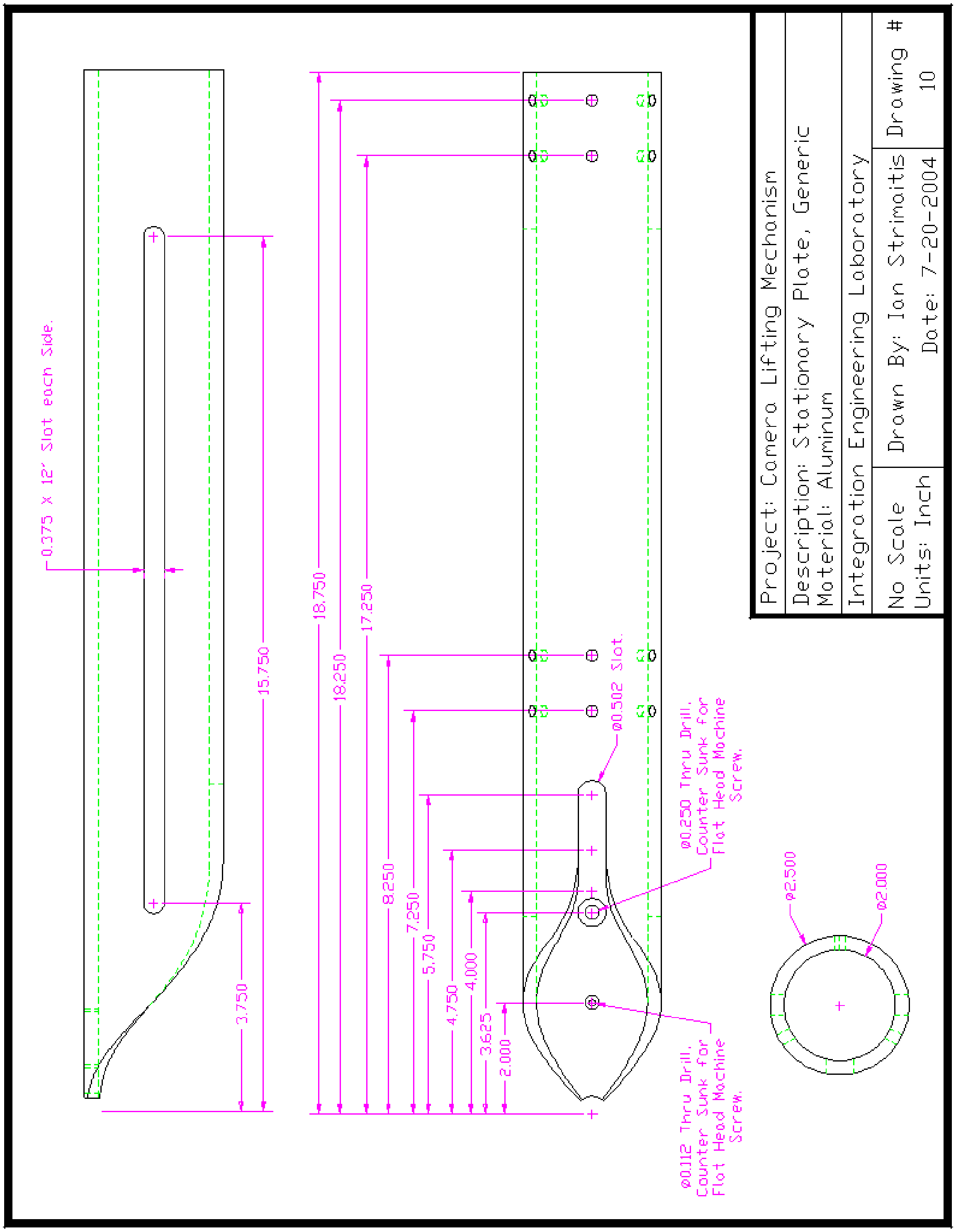
Ø0.125 Drill
0.375 Deep
Two Places
1.000
6.000
Ø1.835
Ø1.960
Ø0.094 Thru Drill
Ø1.000 Drill
5.5" Deep
to Apex.
Project: Camera Lifting Mechanism
Part Description: Slider
Material: White Delrin
Integration Engineering Laboratory
No Scale
Units: inch
Drawn By: Ion Strimaitis
Date: 8-22-2004
Drawing #
5


2.000
0.375
1.625
Section A-A
0.250 Thru Drill
0.375 Countersink
0.250" Deep
Three Places.
Ø0.201 Drill 1.500 Deep
Top 1/4-20 Threads 1" Deep
3 Places.
A
A
Ø2.500
Ø3.750
Project: Camera Lifting Mechanism
Part Description: Plate Flange
Material: 6061 Aluminum
Integration Engineering Laboratory
No Scale
Units: inch
Drawn By: Ian Strimaitis
Date: 8-22-2004
Drawing #
6


Ø0.159 2 places
0.75 Deep on 1.125" Dia,
for 10-32 threads.
Ø1.500
Ø0.159 Thru Drill,
10-32 Threads on
½" Chamfer End.
Ø0.623
45.0°
1.000
0.500
0.500
6.750
0.125
45.0°
0.500
5.625
Ø0.159, 2 places
0.75 Deep on 1.125" Dia,
for 10-32 threads.
Ø1.500
Ø0.623
2.5"
Beyond
Chamfer.
Ø0.159 Thru Drill,
10-32 Threads
from Inside.
Project: Camera Lifting Mechanism
Description: Final Alignment Cones
Material: White Delrin
Integration Engineering Laboratory
No Scale
Units: Inch
Drawn By: Ian Strimaitis
Date: 7-20-2004
Drawing #
7


Project: Camera Lifting Mechanism
Description: Toggle Bone, Toggle Bone Stop
Material: Aluminum
Integration Engineering Laboratory
No Scale
Units: inch
Drawn By: Ian Strimaitis
Date: 8-30-2004
Drawing #
8
Ø0.625
Ø0.250 Thru Drill
Ø0.500 Pocket
0.1875 Deep,
Both Sides,
R0.500 4 Places,
Ø0.107 Thru Drill
6-32 Threads
3.250
Ø0.375
0.188
0.250
1.000
0.500
0.125
Ø0.112
0.375
0.125


Ø1.000
Thru
Pocket
Ø4.000
1/4 Plate
Ø0.112 Thru Drill on 1.625 Dia,
Counter Bore for
Flat Head Screw,
4 Places.
Ø0.089 Drill 0.5 Deep,
1.625 Dia,
4-40 Threads,
4 Places.
1/8x3/16 Keyway
Ø0.500
2.000
Ø2.000
Project: Camera Lifting Mechanism
Description: Pulley Barrel and Side Plates
Material: Aluminum
Integration Engineering Laboratory
No Scale
Units: inch
Drawn By: Ian Strimaitis
Date: 8-30-2004
Drawing #
9


Project: Camera Lifting Mechanism
Description: Stationary Plate, Generic
Material: Aluminum
Integration Engineering Laboratory
No Scale
Units: Inch
Drawn By: Ian Strimaitis
Date: 7-20-2004
Drawing #
10
0.375 X 12" Slot each Side.
15.750
3.750
18.750
18.250
17.250
8.250
7.250
5.750
4.750
4.000
3.625
2.000
Ø0.502 Slot.
Ø0.250 Thru Drill, Counter Sunk for Flat Head Machine Screw.
Ø0.112 Thru Drill, Counter Sunk for Flat Head Machine Screw.
Ø2.500
Ø2.000

APPENDIX III, TRUSS CONSTRUCTION DRAWINGS

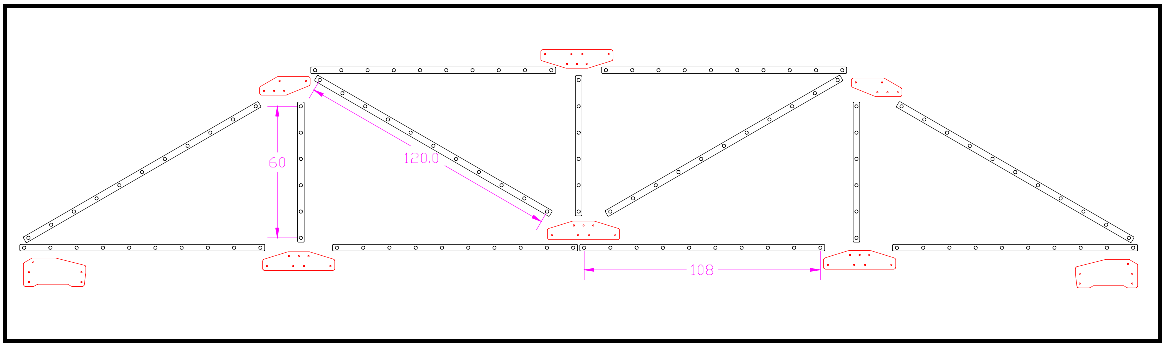


FIGURE 3-A3, DIMENSIONED TRUSS

Units: inches

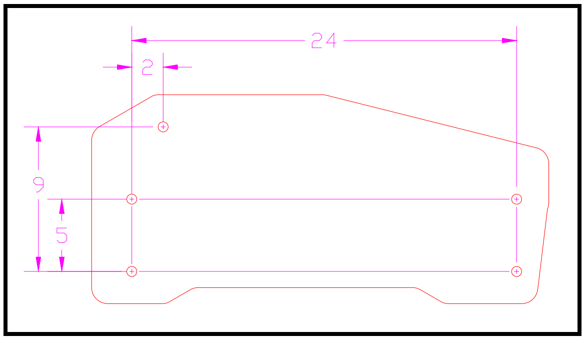


FIGURE 4-A3, BOTTOM END FISH PLATE

Units: inches

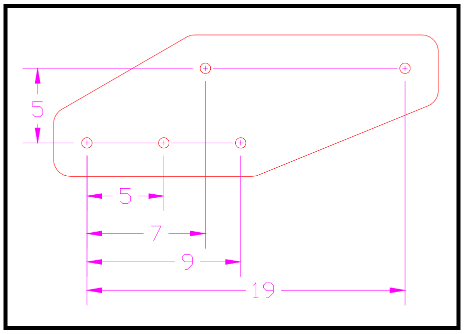


FIGURE 5-A3, TOP END FISH PLATE

Units: inches

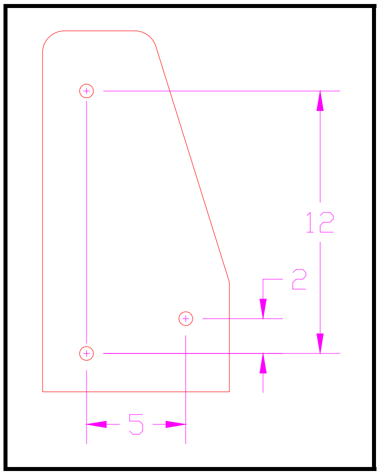


FIGURE 6-A3, EXTENSION LEGS FISH PLATE

Units: inches

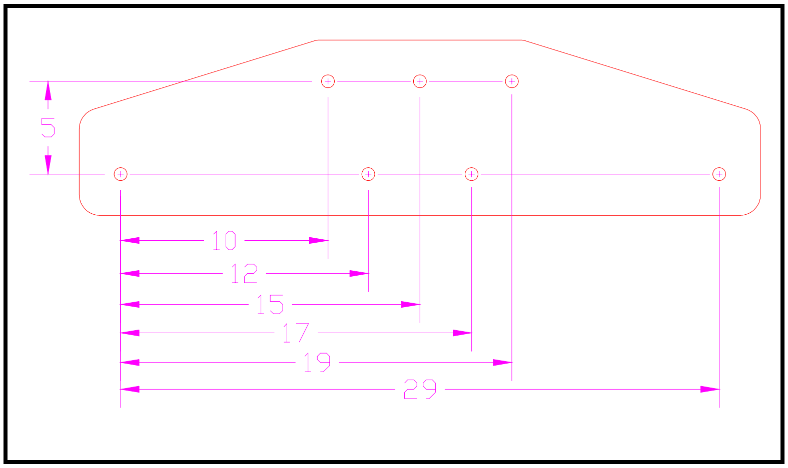


FIGURE 7-A3, STANDARD FISH PLATE

Units: inches

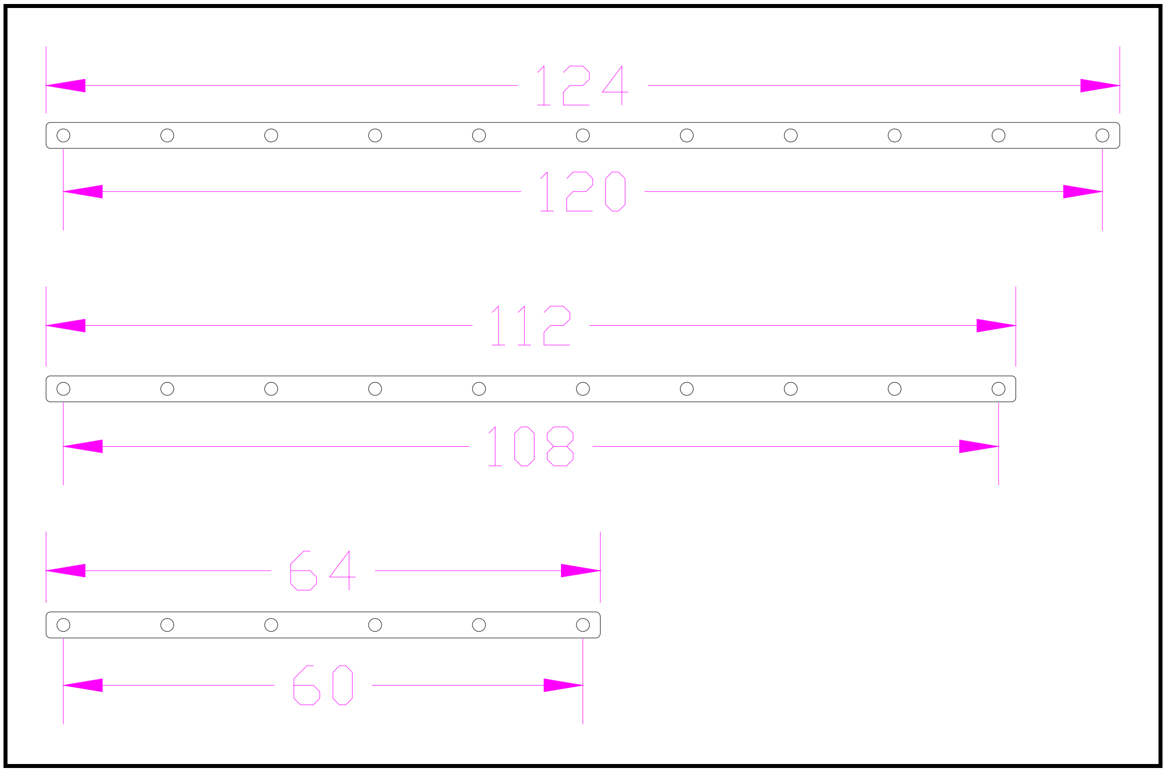


FIGURE 8-A3, LADDER LENGTHS USED FOR MOBILE TRUSS

Units: inches

Appendix IV, Trolley Lifting Mechanism Bill of Materials

Trolley Lifting Mechanism Bill of Materials									
Listing		Part Description	Function	Material	Source	Catalog #	Quantity	Length/Size	Cost
6061 T-8 Aluminum Stock									
AL	1	Sleeve	LM Back Bone / Primary Alignment Fixture/trolley Connection	Round Tubing, 2"ID, 0.25" Wall	ABC Supply	-	1	31.5"	$10.00
AL	2	Mobile Plate	Platform	1/2" Plate	Blue Collar Supply	-	1	8-3/4" X 15-3/8	$14.00
AL	3	Stationary plate	Mounting Plate for Lifting Mechanism	1/2" Plate	Blue Collar Supply	-	1	8-3/4" X 15-3/8	$14.00
AL	4	Motor Plate	Mount for Motor, Connects to Sleeve	1/2" Plate	Blue Collar Supply	-	1	8" X 8"	$30.00
AL	5	Plate Flange	Connect Plates to Sleeve	Round Tubing, 2-1/2" ID, 5/8" Wall	ABC Supply	-	2	4"	$60.00
AL	6	Support Flange	Stabilizes Sleeve Walls Around Slits	Round Tubing Stock	ABC Supply	-	1	2"	$1.00
AL	7	Toggle Bone	Protects Apex of Sleeve Contour	Rectangular Stock	Blue Collar Supply	-	1	4" X 7/8 X 5/8"	$1.00
AL	8	Limit Stop	Limits Toggle Bone Rotation	Rectangular Stock	Blue Collar Supply	-	1	1-1/8" X 1/2" X 3/8"	$1.00
AL	9	Pulley Drum	Connects to Motor, Cable Wraps Around it	2" Round Stock	Blue Collar Supply	-	1	2"	$2.00
AL	10	Pulley Side	Contains Cable on Pulley Drum	1/4" Plate	Blue Collar Supply	-	2	4" Dia.	$1.00
Delrin, White									
D	1	Slider	Lowers Cable Fulcrum	2" Round Stock	McMaster-Carr	8572K29	1	6"	$6.00
D	2	Plug	Performs Initial Alignment with Sleeve	2" Round Stock	McMaster-Carr	8572K29	1	23"	$26.00
D	3	Final Alignment Cone (Stationary)	Performs Final Alignment and Houses	1-1/2" Round Stock	McMaster-Carr	8572K25	4	1/1/2" X 5"	$15.00
D	4	Final Alignment Cone (Mobile)	Performs Final Alignment with Sleeve	1-1/2" Round Stock	McMaster-Carr	8572K25	4	1-1/2" X 4"	$12.00
D	5	Trolley Stop	Stops Trolley on Lifting Mechanism Tracks	1" Round Stock	McMaster-Carr	8572K61	1	1" X 2"	$3.00
D	6	C-Channel Insulating Bushing	Insulates Tracks from Mobile Plate	1" Round Stock	McMaster-Carr	8572K61	4	1" X 1/4"	$1.00
D	7	C-Channel Insulating Flanged Insert	Insulates Tracks from Mobile Plate	1" Round Stock	McMaster-Carr	8572K61	4	1" X 1/8"	$1.00
Black ABS Plastic, Schedule 40 Pipe and Sheet Stock									
ABS	1	Micro Switch Bracket	Connects Micro Switch to Sleeve	3/16" Sheet	Tap Plastics	-	2	3/16" X 1-1/2" X 2-1/4"	$1.00
ABS	2	Pulley Spool Cover	Keeps Cable from Tangling when not Taught	1/16" Sheet	Tap Plastics	-	1	1/16" X 2" X 9"	$1.00
ABS	3	Electronics Side Cover	Covers Time Delay Circuit and Terminal Strip	3/16" Sheet	Tap Plastics	-	1	3/16" X 1-7/8" X 14"	$1.00
ABS	4	Electronics Bottom Cover	Covers Time Delay Circuit and Terminal Strip	3/16" Sheet	Tap Plastics	-	1	3/16 " X 5" X 5"	$1.00
ABS	5	Electronics Mounting Plate	Mounting for Time Delay Circuit and Relay	3/16" Sheet	Tap Plastics	-	1	3/16" X 2-1/4" X 3"	$1.00

ABS	6	Control Panel Support Pipe	Holds Control Panel Off Ground	4" Pipe	Home Depot	-	1	4' X 4" Dia.	$3.00
ABS	7	Control Panel Support Pipe Street Tee	Outlet for Wires	4" Coupling	Home Depot	-	1	4' X 4" Dia.	$2.00
ABS	8	Control Panel Base Connection	Connects Support Pipe to Base	4" Threaded Plug	Home Depot	-	1	4" Dia.	$2.00
ABS	9	Control Panel Coupling	Connects Support Pipe to Base	4" Threaded, Slip Fit, Coupling	Home Depot	-	1	4" Dia.	$2.00
ABS	10	Control Box	Contains Control Switches and Wires	2" Opening Access Panel (Grey PVC)	Home Depot	-	1	2" X 6"	$8.00
Hardware									
Toggle Bone and Plug									
H	1	Flat Head, Phillips Drive, Machine Screw	Pivot for Toggle Bone	Zinc Plated Steel	Orchard Supply	-	1	SAE 1-1/2" 1/4-20	$1.00
H	2	Nylon Lock Nut	Secure Toggle Bone	Zinc Plated Steel	Blue Collar Supply	-	1	SAE 1/4-20	$1.00
H	3	Flat Washer/Shim	Spacer Between Toggle Bearings and Sleeve	Zinc Plated Steel	McMaster-Carr	99040A516	19	SAE 1/4"	$8.20
H	4	Flat Head, Phillips Drive, Machine Screw	Secure Free-Spinning Bearing to Toggle Bone	Stainless Steel	McMaster-Carr	91500A153	1	SAE 1" 10-32	$1.00
H	5	Flat Head, Phillips Drive, Machine Screw	Secure Free-Spinning Bearing to Plug	Stainless Steel	McMaster-Carr	91500A153	1	SAE 1" 10-33	$1.00
H	6	Flat Head, Phillips Drive, Machine Screw	Fasten Limit Stop to Sleeve	Zinc Plated Steel	McMaster-Carr	90273A107	2	SAE 5/16" 4-40	$1.00
H	7	Miniature Precision Stainless Steel Ball Bearing	Initial Alignment on Toggle Bone and Plug	Stainless Steel	RC Country	-	4	1/4" X 1/2" X 3/16"	$16.00
Slider									
H	8	Socket Head Cap Screw	Guide Pin for Slider	Anodized Steel	Orchard Supply	-	2	SAE 1/2" 10-32	$1.00
H	9	Compression Washer	Bearing for Guide Pin	Zinc Plated Steel	Blue Collar Supply	-	2	SAE 10-32	$1.00
Alignment Cones									
H	10	Pan Head, Phillips Drive, Machine Screw	Fasten Final Alignment Cone	Zinc Plated Steel	Blue Collar Supply	-	16	SAE 1-1/2" 10-32	$1.00
H	11	Flat Washer	Fasten Final Alignment Cone	Zinc Plated Steel	Blue Collar Supply	-	16	SAE 10-32	$1.00
H	12	Compression Washer	Fasten Final Alignment Cone	Zinc Plated Steel	Blue Collar Supply	-	16	SAE 10-32	$1.00
Flanges and Plates									
H	13	Socket Head Cap Machine Screw	Fasten Static Plate Flange to Sleeve	Anodized Steel	Blue Collar Supply	-	16	SAE 1" 1/4-20	$1.00
H	14	Socket Head Cap Machine Screw	Fasten Support Flange to Sleeve	Anodized Steel	Blue Collar Supply	-	8	SAE 5/8" 1/4-20	$1.00
H	15	Socket Heat Cap Machine Screw	Fasten Micro Switch to ABS Bracket	Anodized Steel	McMaster-Carr	91251A114	4	SAE 7/8" 4-40	$1.00
H	16	Socket Head Cap Machine Screw	Fasten ABS Bracket to Sleeve	Anodized Steel	McMaster-Carr	91251A108	4	SAE 3/8" 4-40	$1.00
H	17	Socket Head Cap Screw	Fasten Stationary Plates to Flanges	Anodized Steel	Blue Collar Supply	-	6	SAE 1-1/4" 1/4-20	$1.00
H	18	Flat Washer	Bearing Under 1/4-20 Socket Head Cap Screws	Zinc Plated Steel	Blue Collar Supply	-	30	SAE 1/4-20	$1.00
H	19	Compression Washer	Bearing Under 1/4-20 Socket Head Cap Screws	Zinc Plated Steel	Blue Collar Supply	-	30	SAE 1/4-20	$1.00
H	#	Socket Head Cap Screw	Fasten Plug and Mobile Plate	Anodized Steel	Blue Collar Supply	-	4	SAE 1-1/2" 1/4-20	$1.00
C-Channel									

H	21	Pan Head, Phillips Drive, Machine Screw	Fasten C-Channel to Mobile Plate	Zinc Plated Steel	Blue Collar Supply	-	4	SAE 1-1/4" 10-32	$1.00
H	#	Flat Washer	Fasten C-Channel to Mobile Plate	Zinc Plated Steel	Blue Collar Supply	-	4	SAE 10-32	$1.00
H	#	Compression Washer	Fasten C-Channel to Mobile Plate	Zinc Plated Steel	Blue Collar Supply	-	4	SAE 10-32	$1.00
H	#	Lock Nut	Fasten C-Channel to Mobile Plate	Steel, Nylon	Blue Collar Supply	-	4	SAE 10-32	$1.00
H	#	Pan Head, Phillips Drive, Machine Screw	Fasten Power Lug to C-Channel	Zinc Plated Steel	Blue Collar Supply	-	2	SAE 3/8" 4-40	$1.00
H	#	Flat Washer	Fasten Power Lug to C-Channel	Zinc Plated Steel	Blue Collar Supply	-	4	SAE 4-40	$1.00
H	#	Compression Washer	Fasten Power Lug to C-Channel	Zinc Plated Steel	Blue Collar Supply	-	2	SAE 4-40	$1.00
H	#	Nylon Lock Nut	Fasten Power Lug to C-Channel	Zinc Plated Steel	Blue Collar Supply	-	2	SAE 4-40	$1.00
Cable Assembly									
H	#	Docking Stabilization Spring	Secures Docked Mobile Plate	Spring Steel, 175 lbf/in, 230lbf total	Jones Spring	C32-187-384	1	0.970" X 11"	$7.00
H	#	Flat Washer	Bearing on Each Spring End	Zinc Plated Steel	Orchard Supply	-	2	1" OD X 1/4" ID	$1.00
H	31	1/16" Steel Cable	Fasten Mobile Plate to Lifting Mechanism	Stainless Steel	Home Depot	754670	1	1/16" Dia. X 40'	$6.40
H	#	Cable Synch-Down Bolt	Squeezes Cable Inside of Plug	Zinc Plated Steel	Orchard Supply	-	1	SAE 5/8" 3/8-16	$1.00
H	#	Cable Synch-Down Nut	Squeezes Cable Inside of Plug	Zinc Plated Steel	Orchard Supply	-	1	SAE 3/8-16	$1.00
H	#	Cable Synch-Down Washer	Squeezes Cable Inside of Plug	Zinc Plated Steel	Orchard Supply	-	5	SAE 3/8"	$1.00
H	#	Cable Anchor Bolt	Anchor Cable inside of Plug	Zinc Plated Steel	Orchard Supply	-	1	SAE 1/2" 10-32	$1.00
H	#	Cable Anchor Nut	Anchor Cable inside of Plug	Zinc Plated Steel	Orchard Supply	-	1	SAE 10-32	$1.00
H	#	Cable Anchor Washer	Anchor Cable inside of Plug	Zinc Plated Steel	Orchard Supply	-	2	SAE 10-32	$1.00
Pulley									
H	#	Flat Head, Phillips Drive, Machine Screw	Fasten Pulley Sides to Pulley Drum	Zinc Plated Steel	McMaster-Carr	91099A167	8	SAE 1/2" 4-40	$1.00
H	#	Set Screw	Fasten Pulley to Motor Shaft	Zinc Plated Steel	Orchard Supply	-	2	SAE 1/2" 10-32	$1.00
Micro Switches									
H	#	Socket Head Cap Machine Screw	Fasten Micro Switches to ABS Bracket	Anodized Steel	McMaster-Carr	91251A106	4	SAE 1/4" 4-40	$1.00
H	41	Flat Washer	Bearing for Micro Switch Fasteners	Zinc Plated Steel	Blue Collar Supply	-	4	SAE 4-40	$1.00
H	#	Compression Washer	Locking for Micro Switch fasteners	Zinc Plated Steel	Blue Collar Supply	-	4	SAE 4-40	$1.00
Motor and Electrical Cover									
H	#	Socket Head Cap Machine Screw	Fasten Motor to Motor Plate	Anodized Steel	Blue Collar Supply	-	2	SAE 1-1/4" 1/4-20	$1.00
H	#	Flat Washer	Fasten Motor to Motor Plate	Zinc Plated Steel	Blue Collar Supply	-	4	SAE 1/4	$1.00
H	#	Nylon Lock Nut	Fasten Motor to Motor Plate	Steel, Nylon	Blue Collar Supply	-	7	SAE 1/4-20	$1.00
H	#	Long Nut	Fasten Circuit to Motor Plate	Zinc Plated Steel	Orchard Supply	-	1	SAE 1" X 1/4-20	$1.00
H	#	Socket Head Cap Machine Screw	Fasten Motor Cover to Motor Plate	Anodized Steel	Blue Collar Supply	-	1	SAE 1-1/4" 1/4-20	$1.00
H	#	Socket Head Cap Machine Screw	Fasten Motor Cover to Motor Plate	Anodized Steel	Blue Collar Supply	-	1	SAE 2-1/2" 1/4-20	$1.00
H	#	Flat Washer	Fasten Motor Cover to Motor Plate	Zinc Plated Steel	Blue Collar Supply	-	2	SAE 1/4-20	$1.00
H	#	Lock Nut	Fasten Motor Cover to Motor	Steel, Nylon	Blue Collar Supply	-	2	SAE 1/4-20	$1.00

			Plate						
Power Transmission Spring Assembly									
H	51	Pan Head, Phillips Drive, Machine Screw	Squeezes Assembly Together	Zinc Plated Steel	Blue Collar Supply	-	4	SAE 2-1/2 10-32	$1.00
H	#	Flat Washer	Bearing for Rubber Components	Zinc Plated Steel	Blue Collar Supply	-	8	SAE 10-32	$1.00
H	#	Rubber Grommet	Pushes Rubber Spacer Outward	Rubber	Orchard Supply	-	4	5/8" OD, 1/8" ID, 1/4"	$1.00
H	#	Rubber Spacer	Pushes Spring Against Cone Inside Wall	Rubber	Orchard Supply	-	4	5/8" OD, 1/2" ID, 1/2"	$1.00
H	55	Nylon Lock Nut	Determines when Pan head Screw Bottoms Out	Zinc Plated Steel, Nylon	Blue Collar Supply	-	4	SAE 10-32	$1.00
H	#	Power Transmission Spring	Transmits Power to Mobile Plate	Spring Steel	RC Country	-	4	5/8" Dia. X 2-1/2" Lg.	$4.00
H	57	Flat Washer	Spacer at Deepest Point Inside Cone	Zinc Plated Steel	Blue Collar Supply	-	12	SAE 1/4-20	$1.00
Electrical									
E	1	Speed Control Micro Switch, 250V, 15A	Switches Motor to/from slow speed	Steel Lever Arm	McMaster-Carr	7783K12	1	1.94" X 0.69" X 0.95"	$7.66
E	2	Limit Micro Switch, 250V, 15A	Begins motor shut-off process	Steel Lever Arm	McMaster-Carr	7783K12	1	1.94" X 0.69" X 0.95"	$7.66
E	3	Terminal Block	Junction for All Wires on Lifting Mechanism	Nylon, Steel	McMaster-Carr	7527K49	1	7/8" X 4" X 5/16"	$2.00
E	4	Wire Lugs	Connect Wires to Terminal Block	Steel	McMaster-Carr	7113K11	25	#6 Lug, #18 Crimp	$3.00
E	5	Delay Circuit	Compress Stabilization Spring	Ask Mark	Mark Bruening	-	1	1" X 3/4" X 2"	$10.00
E	6	Wires	Connect Components	Copper Stranded	Home Depot	-	1	20'	$4.00
E	7	Power and Control Cords	Power and signal Leads/Plugs	Extension Cord	Home Depot	-	4	6'	$16.00
E	8	DPDT Toggle Switch, Neutral Center	Motor Control Switch, Up, Down, Off	Steel, Glass Filled Nylon	Radio Shack	750653	1	1" X 3/4" X 1-1/2"	$3.00
E	9	Motor, 120V AC/DC, 4.0 RPM, 1/15 HP, 250 in-lb	Lower and Raise Mobile Plate	Steel Housing	Grainger	1L486	1	4-1/4" X 5" X 9"	$116.00
E	10	Relay, 12 Volt Coil, 120 Volt Contact	Turns On and Off Motor	Plastic, Steel, Copper	HSC	-	1	1" " 3/4" X 1-1/2"	$3.00
E	11	Tie Straps	Secure Wires on Lifting Mechanism	Plastic	Home Depot	-	29	4"	$2.00
							414	Total	**$488**

Appendix V, Camera Lifting Mechanism Bill of Materials

Listing		Part Description	Function	Material	Source	Catalog #	Quantity	Length/Size	Cost
Camera Lifting Mechanism Bill of Materials									
6061 T-8 Aluminum Stock									
AL	1	Sleeve	LM Back Bone / Primary Alignment	Round Tubing, 2"ID, 0.25" Wall	ABC Supply	-	1	18-3/8"	$10.00
AL	2	Mobile Plate	Fixture/trolley Connection Platform	1/2" Plate	Blue Collar Supply	-	1	10" X 15"	$14.00
AL	3	Stationary plate	Mounting Plate for Lifting Mechanism	1/2" Plate	Blue Collar Supply	-	1	10" X 15"	$14.00
AL	4	Motor Plate	Mount for Motor, Connects to Sleeve	1/2" Plate	Blue Collar Supply	-	1	8" X 8"	$10.00
AL	5	Plate Flange	Connect Plates to Sleeve	Round Tubing, 2-1/2" ID, 5/8" Wall	ABC Supply	-	2	4"	$60.00
AL	6	Toggle Bone	Protects Apex of Sleeve Contour	Rectangular Stock	Blue Collar Supply	-	1	3" X 7/8 X 5/8"	$1.00
AL	7	Limit Stop	Limits Toggle Bone Rotation	Rectangular Stock	Blue Collar Supply	-	1	1-1/8" X 1/2" X 3/8"	$1.00
AL	8	Pulley Drum	Connects to Motor, Cable Wraps Around it	2" Round Stock	Blue Collar Supply	-	1	2"	$2.00
AL	9	Pulley Side	Contains Cable on Pulley Drum	1/4" Plate	Blue Collar Supply	-	2	4" Dia.	$1.00
Delrin, White									
D	1	Slider	Lowers Cable Fulcrum	2" Round Stock	McMaster-Carr	8572K29	1	5-1/4"	$6.00
D	2	Plug	Performs Initial Alignment with Sleeve	2" Round Stock	McMaster-Carr	8572K29	1	12"	$13.00
D	3	Final Alignment Cone (Stationary)	Performs Final Alignment and Houses	1-1/2" Round Stock	McMaster-Carr	8572K25	4	1/1/2" X 7-5/8"	$30.00
D	4	Final Alignment Cone (Mobile)	Performs Final Alignment with Sleeve	1-1/2" Round Stock	McMaster-Carr	8572K25	4	1-1/2" X 2"	$12.00
D	5	Trolley Stop	Stops Trolley on Lifting Mechanism Tracks	1" Round Stock	McMaster-Carr	8572K61	1	1" X 2"	$3.00
Black ABS Plastic, Schedule 40 Pipe and Sheet Stock									
ABS	1	Micro Switch Bracket	Connects Micro Switch to Sleeve	3/16" Sheet	Tap Plastics	-	2	3/16" X 1-1/2" X 2-1/4"	$1.00
ABS	2	Pulley Spool Cover	Keeps Cable from Tangling when not Taught	1/16" Sheet	Tap Plastics	-	1	1/16" X 2" X 9"	$1.00
ABS	3	Electronics Side Cover	Covers Time Delay Circuit and Terminal Strip	3/16" Sheet	Tap Plastics	-	1	3/16" X 1-7/8" X 14"	$1.00
ABS	4	Electronics Bottom Cover	Covers Time Delay Circuit and Terminal Strip	3/16" Sheet	Tap Plastics	-	1	3/16 " X 5" X 5"	$1.00
ABS	5	Electronics Mounting Plate	Mounting for Time Delay Circuit and Relay	3/16" Sheet	Tap Plastics	-	1	3/16" X 2-1/4" X 3"	$1.00
ABS	6	Control Panel Support Pipe	Holds Control Panel Off Ground	4" Pipe	Home Depot	-	1	4' X 4" Dia.	$3.00
ABS	7	Control Panel Support Pipe Street Tee	Outlet for Wires	4" Coupling	Home Depot	-	1	4" Dia.	$2.00
ABS	8	Control Panel Base Connection	Connects Support Pipe to Base	4" Threaded Plug	Home Depot	-	1	4" Dia.	$2.00

ABS	9	Control Panel Coupling	Connects Support Pipe to Base	4" Threaded, Slip Fit, Coupling	Home Depot	-	1	4" Dia.	$2.00
ABS	10	Control Box	Contains Control Switches and Wires	2" Opening Access Panel (Grey PVC)	Home Depot	-	1	2" X 6"	$8.00
Hardware									
Toggle Bone and Plug									
H	1	Flat Head, Phillips Drive, Machine Screw	Pivot for Toggle Bone	Zinc Plated Steel	Orchard Supply	-	1	SAE 1-1/2" 1/4-20	$1.00
H	2	Nylon Lock Nut	Secure Toggle Bone	Zinc Plated Steel	Blue Collar Supply	-	1	SAE 1/4-20	$1.00
H	3	Flat Washer/Shim	Spacer Between Toggle Bearings and Sleeve	Zinc Plated Steel	McMaster-Carr	99040A516	19	SAE 1/4"	$8.20
H	4	Flat Head, Phillips Drive, Machine Screw	Secure Free-Spinning Bearing to Toggle Bone	Stainless Steel	McMaster-Carr	91500A153	1	SAE 1" 10-32	$1.00
H	5	Flat Head, Phillips Drive, Machine Screw	Secure Free-Spinning Bearing to Plug	Stainless Steel	McMaster-Carr	91500A153	1	SAE 1" 10-33	$1.00
H	6	Flat Head, Phillips Drive, Machine Screw	Fasten Limit Stop to Sleeve	Zinc Plated Steel	McMaster-Carr	90273A107	2	SAE 5/16" 4-40	$1.00
H	7	Miniature Precision Stainless Steel Ball Bearing	Initial Alignment on Toggle Bone and Plug	Stainless Steel	RC Country	-	4	1/4" X 1/2" X 3/16"	$16.00
Slider									
H	8	Socket Head Cap Screw	Guide Pin for Slider	Anodized Steel	Orchard Supply	-	2	SAE 1/2" 10-32	$1.00
H	9	Compression Washer	Bearing for Guide Pin	Steel	Blue Collar Supply	-	2	SAE 10-32	$1.00
Alignment Cones									
H	10	Pan Head, Phillips Drive, Machine Screw	Fasten Final Alignment Cone	Steel	Blue Collar Supply	-	16	SAE 1-1/2" 10-32	$1.00
H	11	Flat Washer	Fasten Final Alignment Cone	Steel	Blue Collar Supply	-	16	SAE 10-32	$1.00
H	12	Compression Washer	Fasten Final Alignment Cone	Steel	Blue Collar Supply	-	16	SAE 10-32	$1.00
Flanges and Plates									
H	13	Socket Head Cap Machine Screw	Fasten Static Plate Flange to Sleeve	Anodized Steel	Blue Collar Supply	-	16	SAE 1" 1/4-20	$1.00
H	14	Socket Head Cap Machine Screw	Fasten Support Flange to Sleeve	Anodized Steel	Blue Collar Supply	-	8	SAE 5/8" 1/4-20	$1.00
H	15	Socket Heat Cap Machine Screw	Fasten Micro Switch to ABS Bracket	Anodized Steel	McMaster-Carr	91251A114	4	SAE 7/8" 4-40	$1.00
H	16	Socket Head Cap Machine Screw	Fasten ABS Bracket to Sleeve	Anodized Steel	McMaster-Carr	91251A108	4	SAE 3/8" 4-40	$1.00
H	17	Socket Head Cap Screw	Fasten Stationary Plates to Flanges	Anodized Steel	Blue Collar Supply	-	6	SAE 1-1/4" 1/4-20	$1.00
H	18	Flat Washer	Bearing Under 1/4-20 Socket Head Cap Screws	Steel	Blue Collar Supply	-	30	SAE 1/4-20	$1.00
H	19	Compression Washer	Bearing Under 1/4-20 Socket Head Cap Screws	Steel	Blue Collar Supply	-	30	SAE 1/4-20	$1.00
H	20	Socket Head Cap Screw	Fasten Plug and Mobile Plate	Anodized Steel	Blue Collar Supply	-	4	SAE 1-1/2" 1/4-20	$1.00
Cable Assembly									
H	21	Docking Stabilization Spring	Secures Docked Mobile Plate	Spring Steel, 20.5 lbs/in, 41.2 lbs Total Load	McMaster-Carr	9657K254	1	3-1/16" Lg. X 7/8" Dia.	$13.49
H	22	Flat Washer	Bearing on Each Spring End	Steel	Orchard Supply	-	2	1" OD X 1/4" ID	$1.00

H	23	1/16" Steel Cable	Fasten Mobile Plate to Lifting Mechanism	Stainless Steel	Home Depot	754670	1	1/16" Dia. X 40'	$6.40
H	24	Cable Synch-Down Bolt	Squeezes Cable Inside of Plug	Steel	Orchard Supply	-	1	SAE 5/8" 3/8-16	$1.00
H	25	Cable Synch-Down Nut	Squeezes Cable Inside of Plug	Steel	Orchard Supply	-	1	SAE 3/8-16	$1.00
H	26	Cable Synch-Down Washer	Squeezes Cable Inside of Plug	Steel	Orchard Supply	-	5	SAE 3/8"	$1.00
H	27	Cable Anchor Bolt	Anchor Cable inside of Plug	Steel	Orchard Supply	-	1	SAE 1/2" 10-32	$1.00
H	28	Cable Anchor Nut	Anchor Cable inside of Plug	Steel	Orchard Supply	-	1	SAE 10-32	$1.00
H	29	Cable Anchor Washer	Anchor Cable inside of Plug	Steel	Orchard Supply	-	2	SAE 10-32	$1.00
Pulley									
H	30	Flat Head, Phillips Drive, Machine Screw	Fasten Pulley Sides to Pulley Drum	Steel	McMaster-Carr	91099A167	8	SAE 1/2" 4-40	$1.00
H	31	Set Screw	Fasten Pulley to Motor Shaft	Steel	Orchard Supply	-	2	SAE 1/2" 10-32	$1.00
Micro Switches									
H	32	Socket Head Cap Machine Screw	Fasten Micro Switches to ABS Bracket	Anodized Steel	McMaster-Carr	91251A106	4	SAE 1/4" 4-40	$1.00
H	33	Flat Washer	Bearing for Micro Switch Fasteners	Steel	Blue Collar Supply	-	4	SAE 4-40	$1.00
H	34	Compression Washer	Locking for Micro Switch fasteners	Steel	Blue Collar Supply	-	4	SAE 4-40	$1.00
Motor and Electrical Cover									
H	35	Socket Head Cap Machine Screw	Fasten Motor to Motor Plate	Anodized Steel	Blue Collar Supply	-	2	SAE 1-1/4" 1/4-20	$1.00
H	36	Flat Washer	Fasten Motor to Motor Plate	Steel	Blue Collar Supply	-	4	SAE 1/4	$1.00
H	37	Nylon Lock Nut	Fasten Motor to Motor Plate	Steel, Nylon	Blue Collar Supply	-	7	SAE 1/4-20	$1.00
H	38	Long Nut	Fasten Circuit to Motor Plate	Steel	Orchard Supply	-	1	SAE 1" X 1/4-20	$1.00
H	39	Socket Head Cap Machine Screw	Fasten Motor Cover to Motor Plate	Anodized Steel	Blue Collar Supply	-	1	SAE 1-1/4" 1/4-20	$1.00
H	40	Socket Head Cap Machine Screw	Fasten Motor Cover to Motor Plate	Anodized Steel	Blue Collar Supply	-	1	SAE 2-1/2" 1/4-20	$1.00
H	41	Flat Washer	Fasten Motor Cover to Motor Plate	Steel	Blue Collar Supply	-	2	SAE 1/4-20	$1.00
H	42	Lock Nut	Fasten Motor Cover to Motor Plate	Steel, Nylon	Blue Collar Supply	-	2	SAE 1/4-20	$1.00
Power Transmission Spring Assembly									
H	43	Pan Head, Phillips Drive, Machine Screw	Squeezes Assembly Together	Zinc Plated Steel	Blue Collar Supply	-	4	SAE 2-1/2 10-32	$1.00
H	44	Flat Washer	Bearing for Rubber Components	Zinc Plated Steel	Blue Collar Supply	-	8	SAE 10-32	$1.00
H	45	Rubber Grommet	Pushes Rubber Spacer Outward	Rubber	Orchard Supply	-	4	5/8" OD, 1/8" ID, 1/4"	$1.00
H	46	Rubber Spacer	Pushes Spring Against Cone Inside Wall	Rubber	Orchard Supply	-	4	5/8" OD, 1/2" ID, 1/2"	$1.00
H	47	Nylon Lock Nut	Determines when Pan head Screw Bottoms Out	Zinc Plated Steel, Nylon	Blue Collar Supply	-	4	SAE 10-32	$1.00
H	48	Power Transmission Spring	Transmits Power to Mobile Plate	Spring Steel	RC Country	-	4	5/8" Dia. X 2-1/2" Lg.	$4.00
H	49	Flat Washer	Spacer at Deepest Point Inside Cone	Zinc Plated Steel	Blue Collar Supply	-	12	SAE 1/4-20	$1.00
				Electrical					
E	1	Speed Control Micro Switch, 250V, 15A	Switches Motor to/from slow speed	Steel Lever Arm	McMaster-Carr	7783K12	1	1.94" X 0.69" X 0.95"	$7.66

E	2	Limit Micro Switch, 250V, 15A	Begins motor shut-off process	Steel Lever Arm	McMaster-Carr	7783K12	1	1.94" X 0.69" X 0.95"	$7.66
E	3	Terminal Block	Junction for All Wires on Lifting Mechanism	Nylon, Steel	McMaster-Carr	7527K49	1	7/8" X 4" X 5/16"	$2.00
E	4	Wire Lugs	Connect Wires to Terminal Block	Steel	McMaster-Carr	7113K11	25	#6 Lug, #18 Crimp	$3.00
E	5	Wires	Connect Components	Copper Stranded	Home Depot	-	1	20'	$4.00
E	6	Power and Control Cords	Power and signal Leads/Plugs	Extension Cord	Home Depot	-	4	6'	$16.00
E	7	DPDT Toggle Switch, Neutral Center	Motor Control Switch, Up, Down, Off	Steel, Glass Filled Nylon	Radio Shack	750653	1	1" X 3/4" X 1-1/2"	$3.00
E	8	Motor, 120V AC/DC, 12.8 RPM, 1/15 HP, 110 in-lb	Lower and Raise Mobile Plate	Steel Housing	Grainger	1L484	1	4-1/4" X 5" X 9"	$116.00
E	9	Tie Straps	Secure Wires on Lifting Mechanism	Plastic	Home Depot	-	29	4"	$2.00

APPENDIX VI, TRUSS BILL OF MATERIALS

Mobile Truss Bill of Materials									
Listing		Part Description	Function	Material	Source	Catalog #	Quantity	Length/Size	Cost
Werner Aluminum Extension Ladders and C-Channel									
AL	1	124" Angled Span Piece	Span Triangulation	24' 250LB, Type 1 Duty	Home Depot	51751012170	4	124"	$318.00
AL	2	112" Bottom Span Piece	Span Tension, Support Aluminum C-Channel	24' 250LB, Type 1 Duty	Home Depot	51751012170	4	112"	$318.00
AL	3	112" Horizontal Top Span Piece	Span Compression, Top Pieces	24' 250LB, Type 1 Duty	Home Depot	51751012170	2	112"	$159.00
AL	4	64" Vertical Span Piece	Support at Each Section Edge	24' 250LB, Type 1 Duty	Home Depot	51751012170	3	64"	$159.00
AL	5	124" Vertical Leg	Height Extension Legs	24' 250LB, Type 1 Duty	Home Depot	51751012187	2	124"	$119.00
AL	6	124" Angled Leg	Height Extension Triangulation	24' 250LB, Type 1 Duty	Home Depot	51751012187	2	124"	$119.00
AL	7	113" C-Channel	Trolley Track	Aluminum C-Channel	ABC Supply, Inc.	51751012095	4	3" X 2" X 1/8" X 113"	$228.12
AL	8	49" C-Channel	Trolley Track	Aluminum C-Channel	ABC Supply, Inc.	51751012095	4	3" X 2" X 1/8" X 49"	$99.18
3/4" Plywood									
W	1	5-Point End-Span Fish Plate (2 Triangulation Points)	Hold Ladders Together	3/4" Marine Grade	Hughes Hardwoods	-	4	3/4" X 13" X 28.5"	$71.95
W	2	5-Point Fish Plate (No Triangulation Points)	Hold Ladders Together	3/4" Marine Grade	Hughes Hardwoods	-	6	3/4" X 8" X 33"	-
W	3	5-Point Quarter-Span Fish Plate (2 Triangulation Points)	Hold Ladders Together	3/4" Marine Grade	Hughes Hardwoods	-	4	3/4" X 8 X 23-1/8"	-
W	4	7-Point Mid-Span Fish Plate (2 Triangulation Points)	Hold Ladders Together	3/4" Marine Grade	Hughes Hardwoods	-	2	3/4" X 8" X 33"	-
W	5	3-Point Fish Plate (1 Triangulation Point)	Hold Ladders Together	3/4" Marine Grade	Hughes Hardwoods	-	4	3/4" X 8" X 16-1/2"	-
W	6	Dielectric Track Support	Connect Tracks to Ladders	3/4" Agathis	Hughes Hardwoods	-	7	3/4" X 2" X 17-1/2"	-
W	7	Dielectric Track Support Rigidity Strips	Stiffen Up Track Supports	3/4" Agathis	Hughes Hardwoods	-	14	3/4" X 2 X 14-1/2"	$26.95
W	8	3/4" Wood Ladder Width Shims	Make ladder Widths Uniform	3/4" Agathis	Hughes Hardwoods	-	14	3/4" X 2-1/2" X 15"	-
Black ABS Plastic									
ABS	1	1/4" Plastic Ladder Width Shims	Make ladder Widths Uniform	1/4" ABS Sheet	Tap Plastics	-	12	1/4" X 2-1/2" X 15"	$5.10
ABS	2	3/16" Insert Shims	Shim Inserts to Snugly Fit Inside Ladder Rungs	1/4" ABS Sheet	Tap Plastics	-	96	3/16" X 1-7/8' X 5/8"	$22.97
Schedule 40 White PVC Plastic									
PVC	1	1/2" Couplers	Adapter Between PVC Pipe and Ladder Rung	1/2" Pipe Coupling	Home Depot	49081137472	100	1/2"	$5.00
PVC	2	1/2" Sleeve	Sleeve Between Couplers for All-Thread to Slide	1/2" Pipe	Home Depot	24599050512	50	1/2" X 15-1/2"	$10.00

			in						
Hardware									
H	1	5/8" All-Thread Rod	Connect Ladders via Fish Plating	Steel	Blue Collar Supply	-	48	20-3/4" SAE 5/8"-13	$104.00
H	2	5/8" All-Thread Rod	Connect Truss to Genie Lifts	Steel	Blue Collar Supply	-	2	24" SAE 5/8"-13	$4.00
H	3	5/8" Washers	Truss Assembly	Steel	Blue Collar Supply	-	100	SAE 5/8"	$15.00
H	4	5/8" Nylon Locking Nuts	Truss Assembly	Steel	Blue Collar Supply	-	50	SAE 5/8"	$15.00
H	5	5/8" Nuts	Truss Assembly	Steel	Blue Collar Supply	-	50	SAE 5/8"	$15.00
H	6	5/8" Compression Washers	Truss Assembly	Steel	Blue Collar Supply	-	50	SAE 5/8"	$5.00
H	7	1" Coarse Threaded Drywall Screws	Connect Ladder Width Shims	Steel	Blue Collar Supply	-	12	1" #6	$1.00
H	8	2" Coarse Thread Drywall Screws	Connect Track Supports to Rigidity Strips	Steel	Blue Collar Supply	-	21	2" #6	$1.00
H	9	1-1/4" Coarse Thread Drywall Screws	Connect Dielectric Track Supports Together	Steel	Blue Collar Supply	-	56	1-1/4" #6	$2.00
H	10	1" Phillips Head Machine Screw	Connect Dielectric Track Support	Steel	Blue Collar Supply	-	28	1" SAE 10-32	$2.00
H	11	10-32 Nylon Lock Nut	Connect Dielectric Track Support	Steel	Blue Collar Supply	-	28	SAE 10-32	$2.00
H	12	10-32 Washer	Connect Dielectric Track Support	Steel	Blue Collar Supply	-	28	SAE 10-32	$2.00
H	13	Tensioning Rope	Keep Truss Stable when Raised	Nylon	Home Depot	-	4	1/2" X 100'	$40.00
H	14	Carabineer	Connects Rope Pieces	Steel	Home Depot	-	8	4"	$24.00
H	15	1-1/4" Coarse Thread Drywall Screws	Secure Cable Loops	Steel	Blue Collar Supply	-	28	1-1/4" #6	$2.00
H	16	Cable Loops	Secure Electrical Plugs to Truss	Nylon	McMaster		14	5/16"	$4.00
H	17	Tie Straps	Secure Electrical Plugs to Truss	Plastic	Home Depot	-	14	4"	$2.00
Electrical									
E	1	Male and Female Extension Cord Ends	Connect C-Channel Electrically at Truss Joints	25' Orange Extension Cord	Home Depot	-	4	24"	$20.00
							883	Totals	**$1,921**

APPENDIX VII, COMPANY CONTACT INFORMATION

Company Contact Information				
Company Name	Postal Address	Internet Address	Telephone Number	Account #
Hughes Hardwoods	11441 Sunrise Gold Circle, Rancho Cordova, CA 95742	hugheshardwoods.com	916 638 8658	-
Home Depot	8000 Folsom Blvd. Sacramento, CA 95826	homedepot.com	916 381 3181	-
Tap Plastics, Store #27	4506 Florin Road, Sacramento, CA 95841	tapplastics.com	916 429 9551	-
ABC Supply, Inc.	2710 R Street, Sacramento, CA 95816	-	916 452 7000	-
Radio Shack, Store #3903	5650 Folsom Blvd. Sacramento, CA 95819	radioshack.com	916 452 4632	-
Jones Spring	140 South Street, Wilder KY 41071	springsfast.com	859 581 7600	-
Grainger	2261 Ringwood Ave. San Jose, CA 95131	grainger.com	408 286 5373	80-928-202-3
McMaster-Carr	9630 Norwalk Blvd. Santa Fe Springs, CA 90670	mcmaster.com	562 692 5911	10515500
HSC Electronics Supply	4837 Amber Ln. Sacramento, CA 95841	-	916 338 2545	-
Bark Bruening	-	-	916 524 6633	-
RC Country	6011 Folsom Blvd. Sacramento, CA 95819	-	916 731 5868	-

APPENDIX VIII, INDUSTRIAL SCAFFOLDING PICTURES

The vertical components of an industrial scaffolding system were considered for the horizontal span of the mobile truss. These components were evaluated at the Sun Rental facility in West Sacramento.

FIGURE 9-A8, MATT STANDING AT CENTER OF SCAFFOLDING SPAN

FIGURE 10-A8, DEFLECTION FROM LOAD AT CENTER OF SCAFFOLDING SPAN

FIGURE 11-A8, INDIVIDUAL SCAFFOLDING PIECE

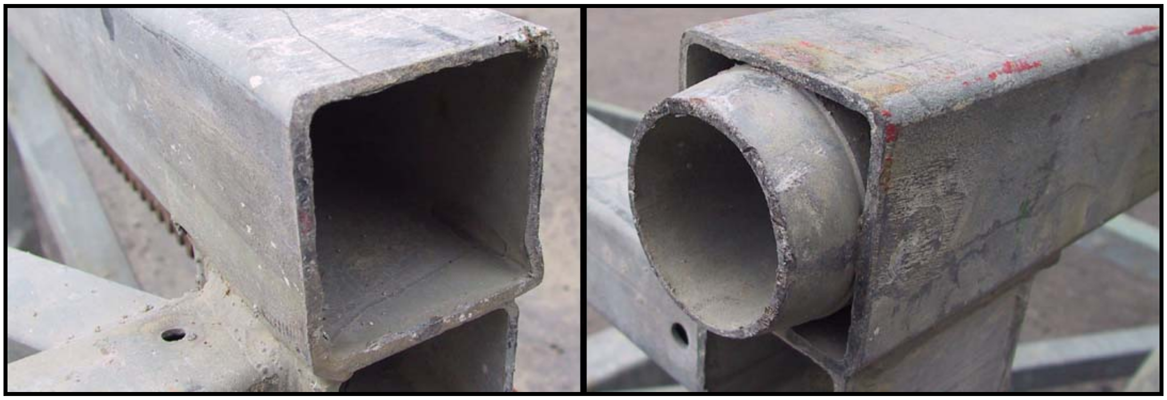


FIGURE 12-A8, SCAFFOLDING CONNECTION PRE-ASSEMBLED CORNER DETAIL

FIGURE 13-A8, TWO SCAFOLDING PIECES BOLTED TOGETHER

APPENDIX IX, ELECTRICAL SCHEMATICS

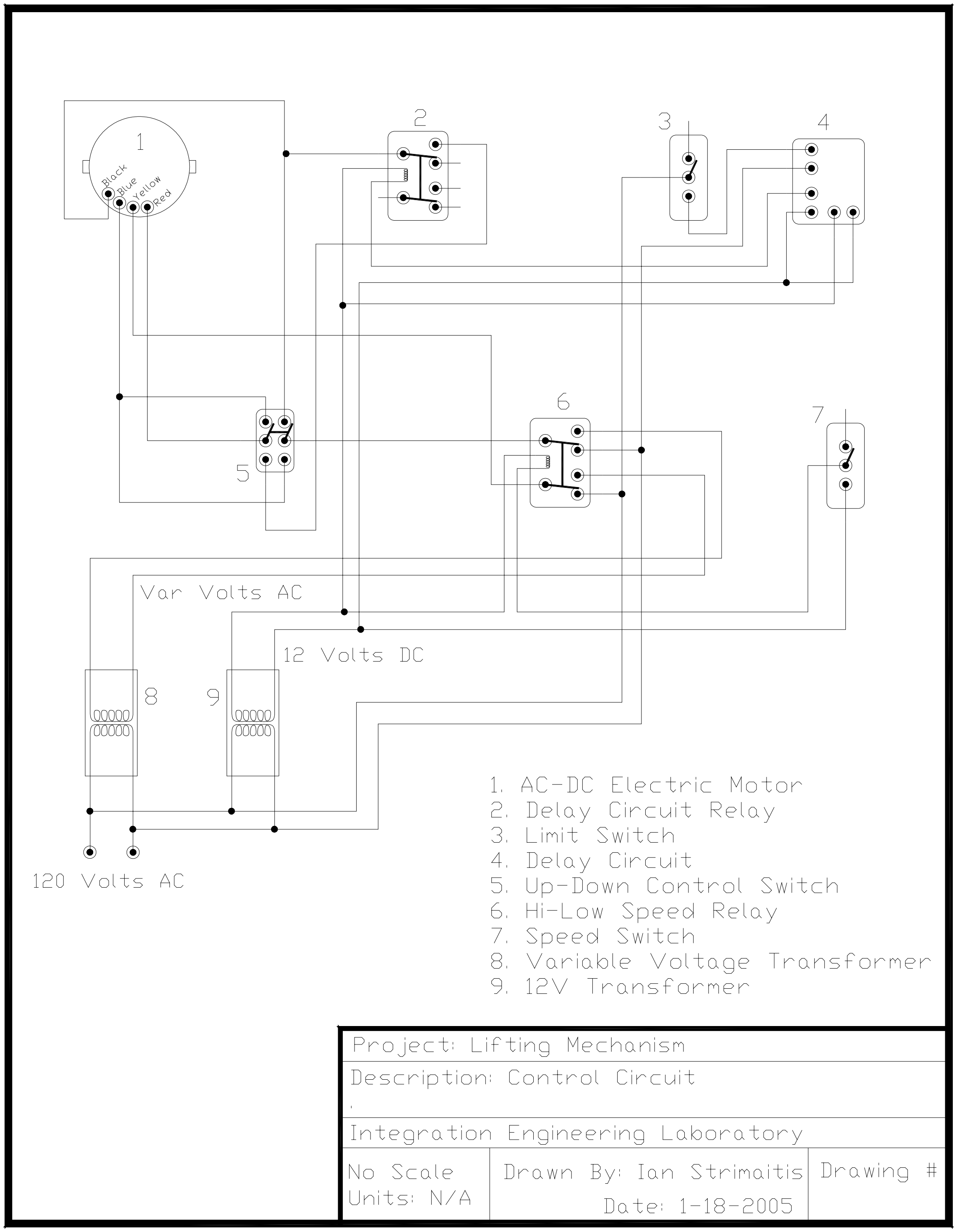


FIGURE 14-A9, LIFTING MECHANISM CONTROL ELECTRICAL CIRCUIT

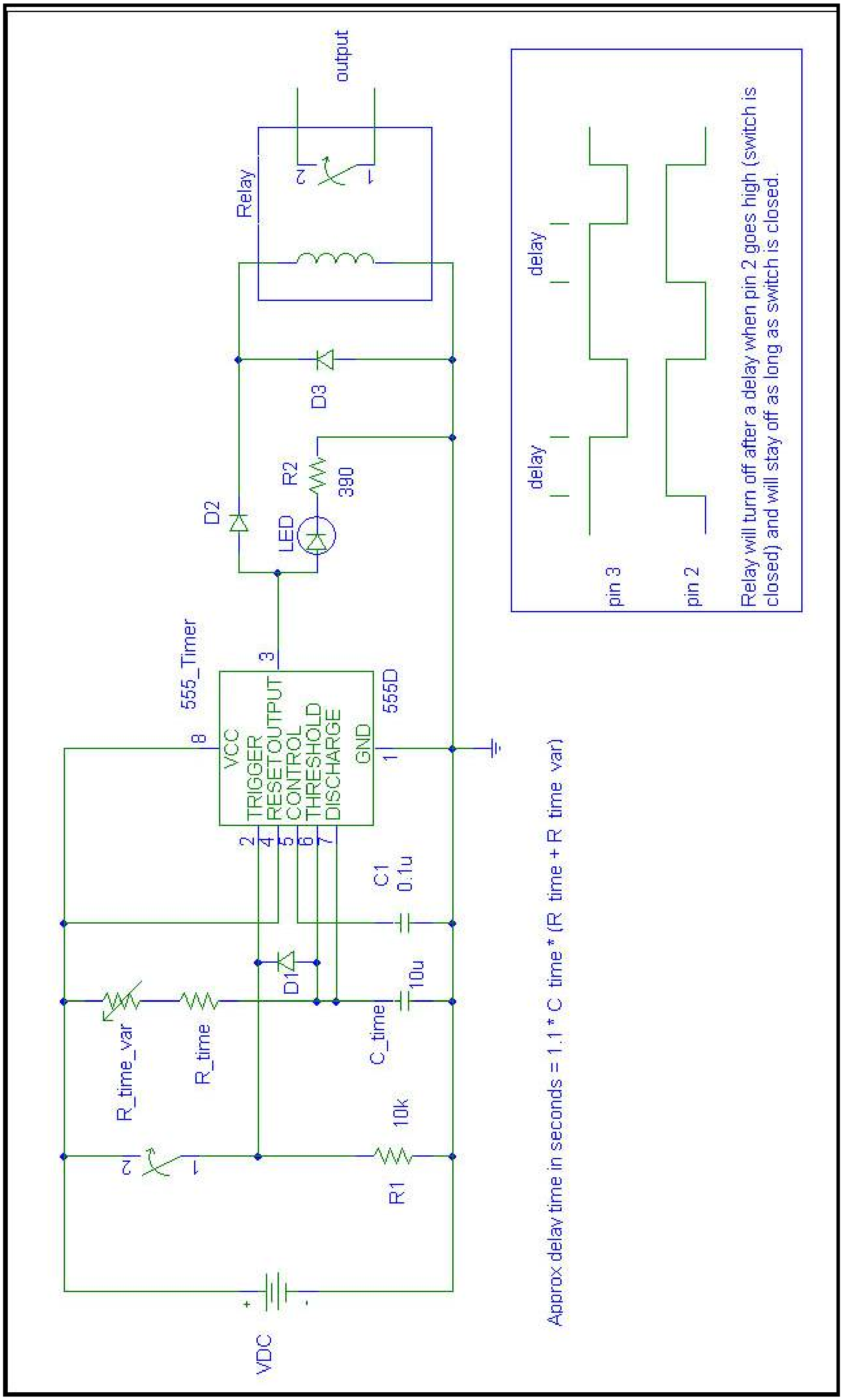
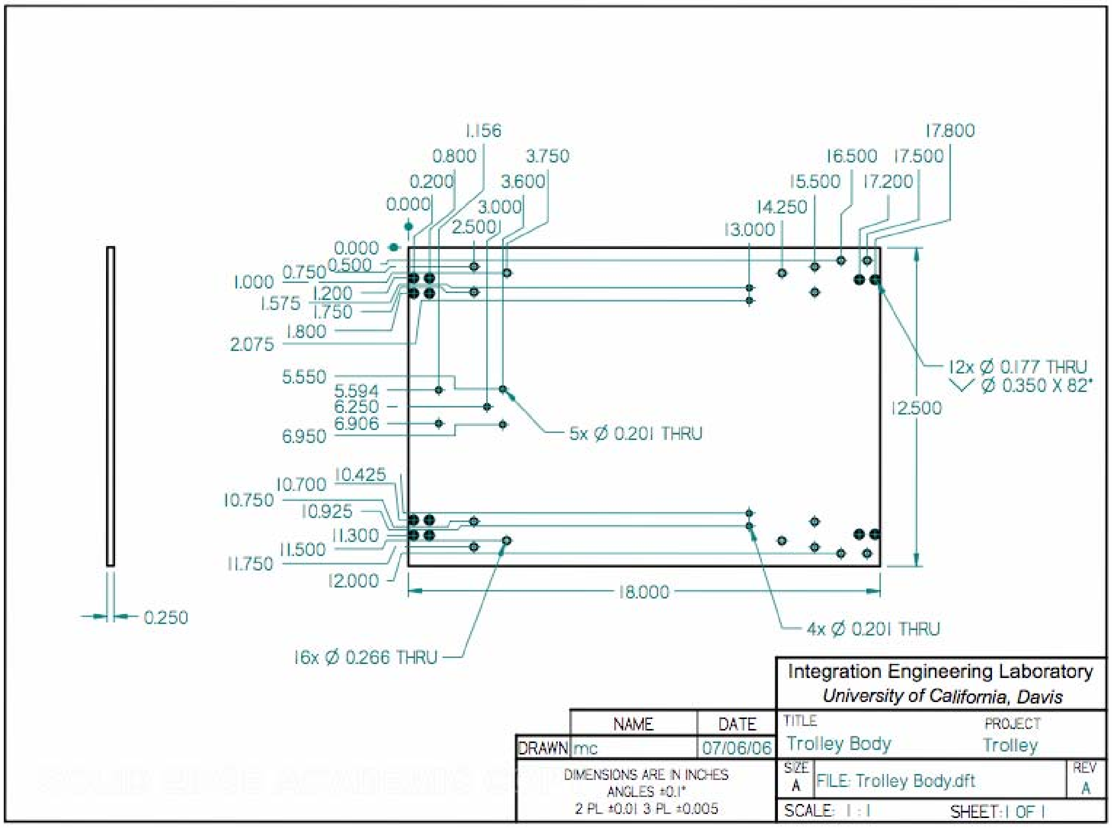
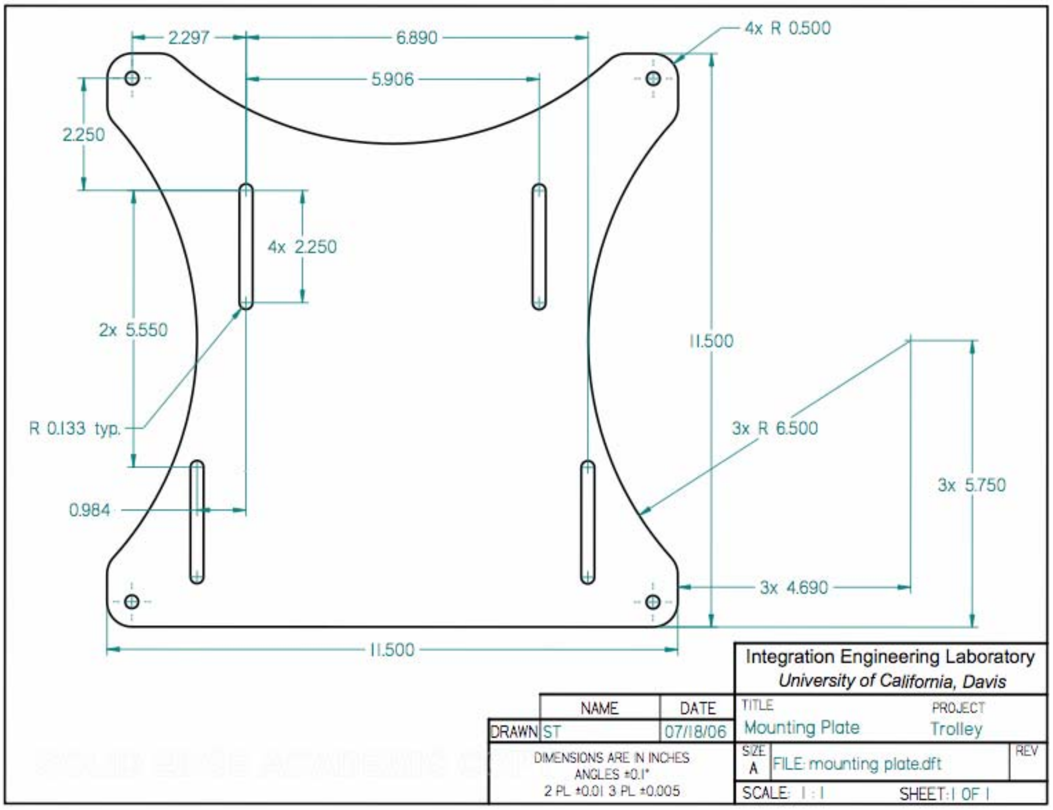
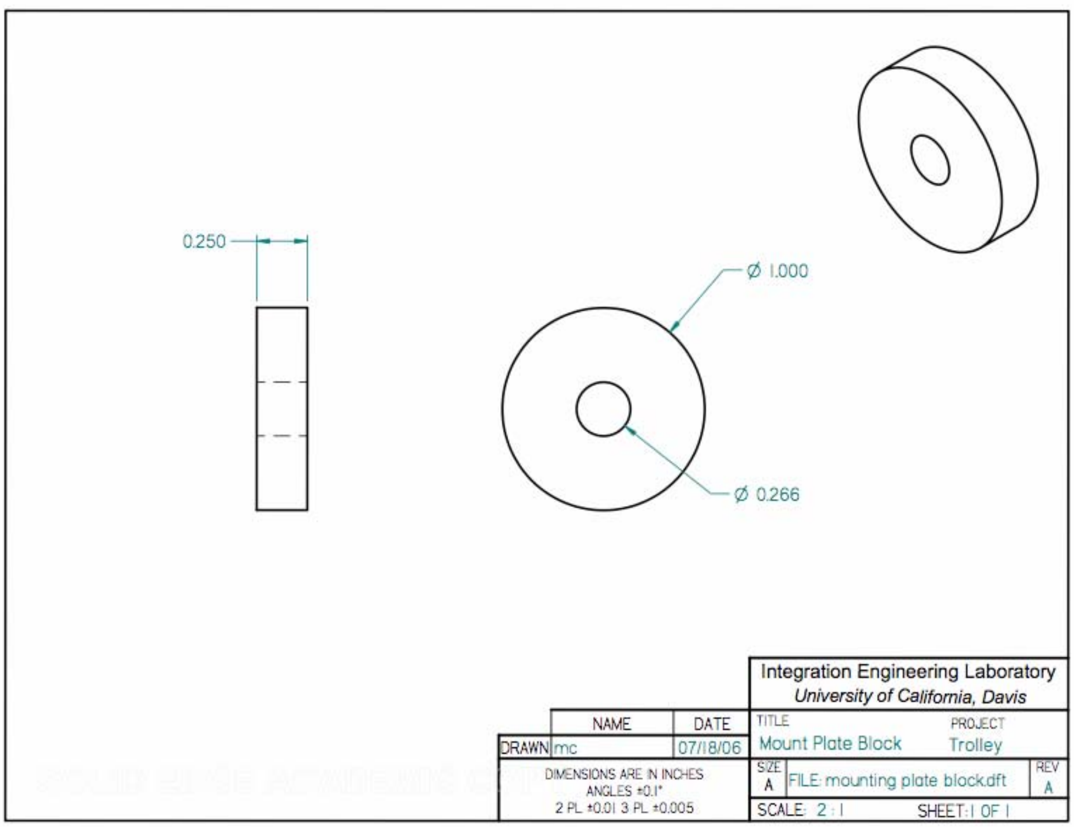


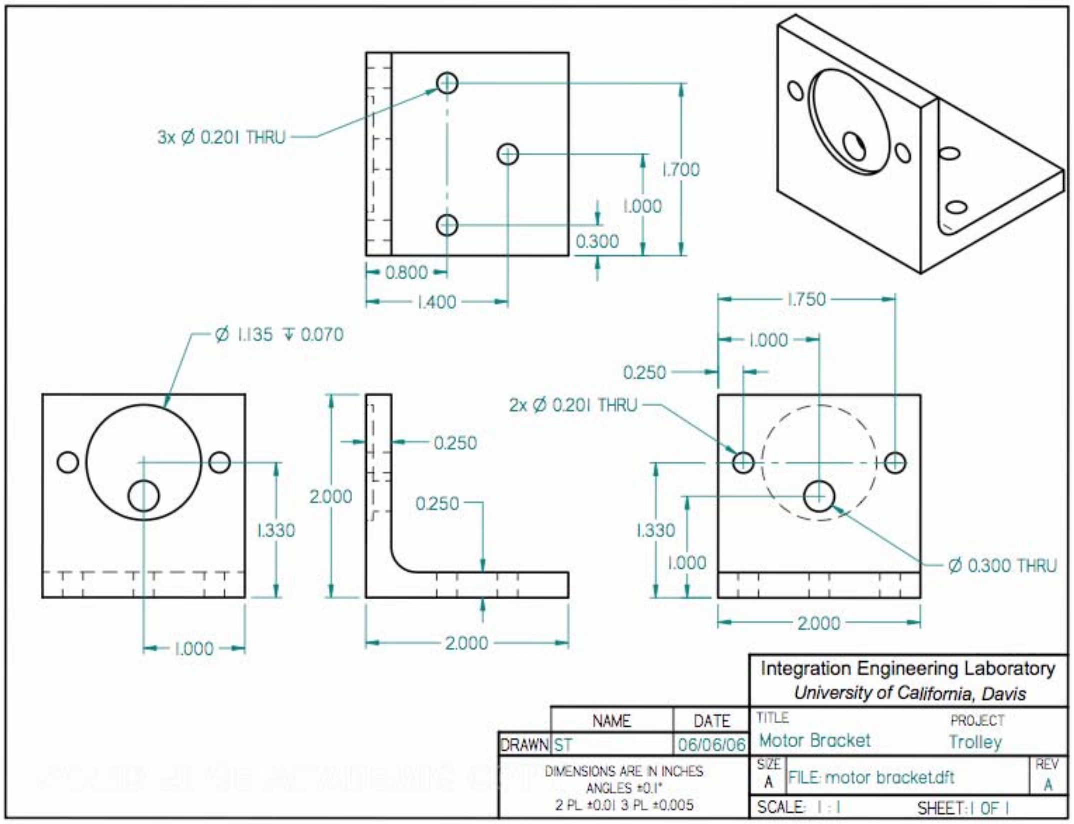
FIGURE 15-A9, ELECTRONIC TIMER DELAY CIRCUIT

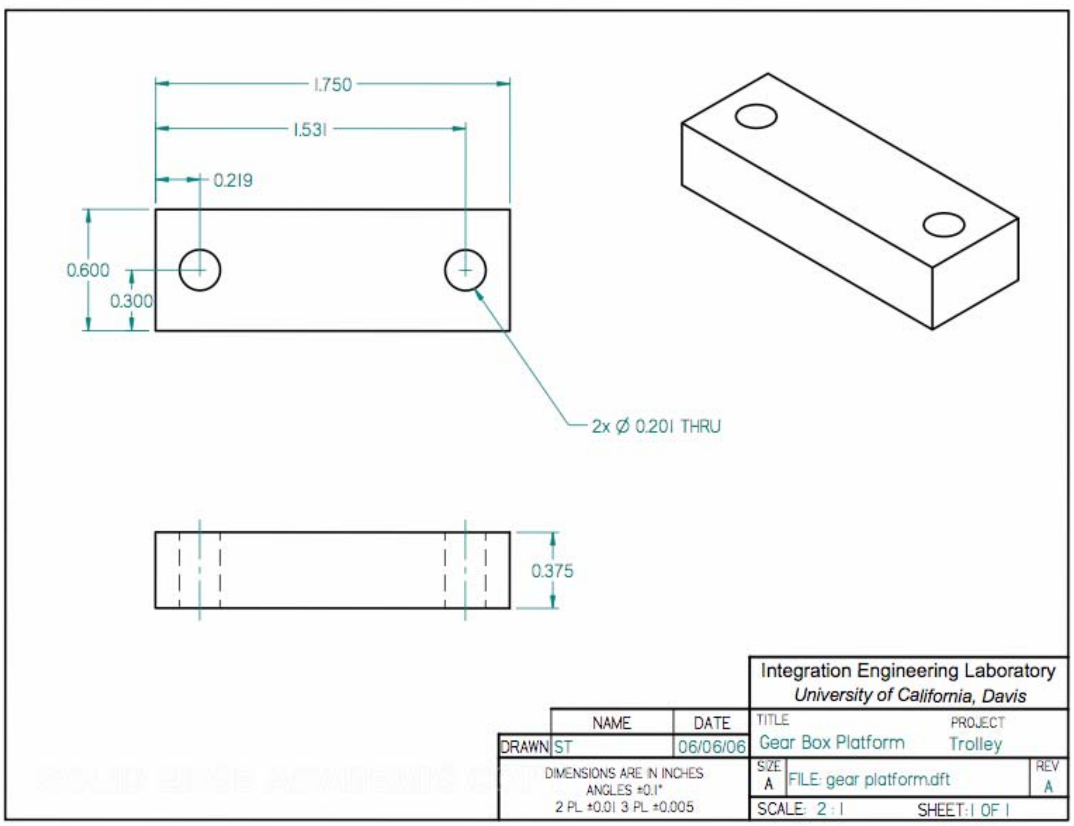
Appendix A: Mechanical Drawings

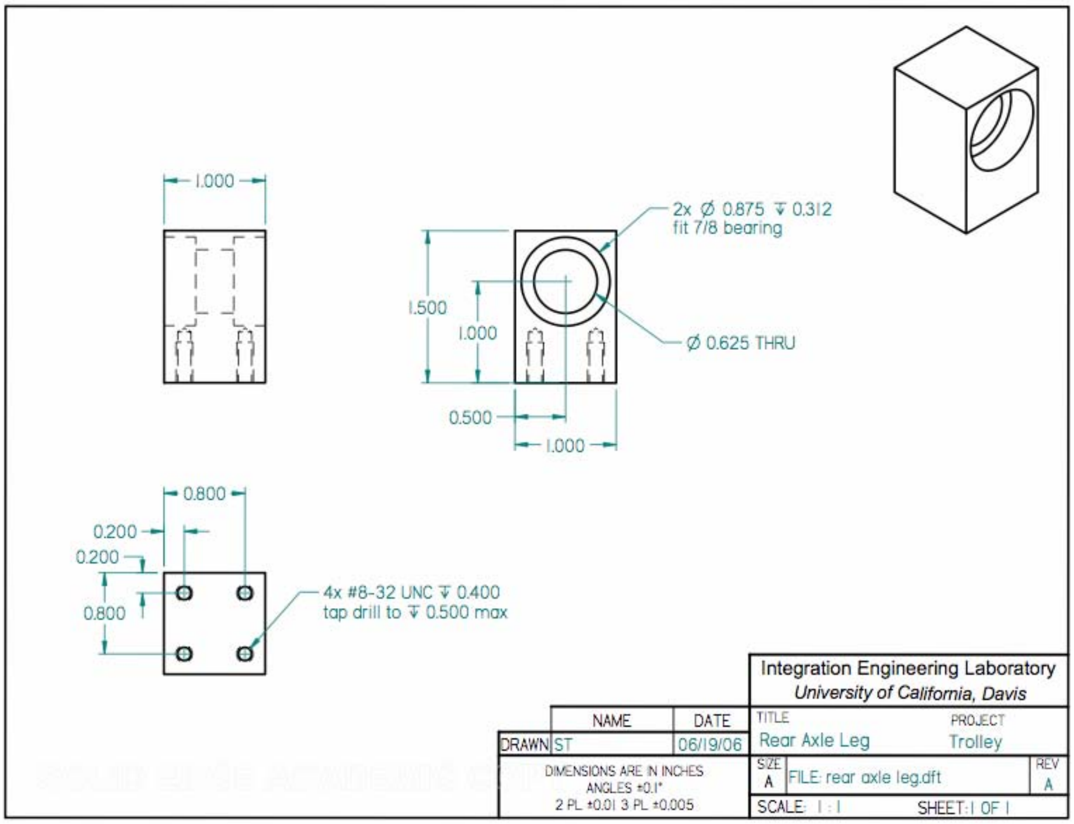


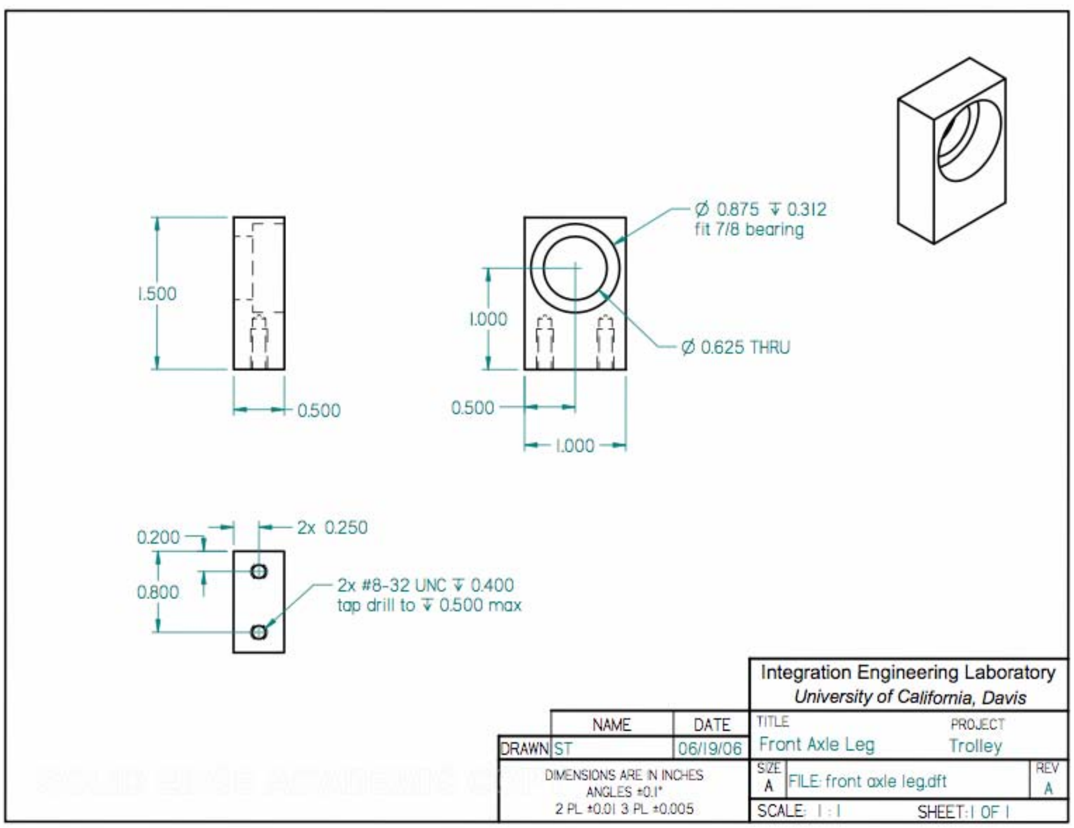

2.297
6.890
4x R 0.500
5.906
2.250
4x 2.250
2x 5.550
11.500
R 0.133 typ.
3x R 6.500
3x 5.750
0.984
3x 4.690
11.500
Integration Engineering Laboratory
University of California, Davis
NAME
DATE
DRAWN ST
07/18/06
TITLE
Mounting Plate
PROJECT
Trolley
DIMENSIONS ARE IN INCHES
ANGLES ±0.1°
2 PL ±0.01 3 PL ±0.005
SIZE A
FILE: mounting plate.dft
REV
SCALE: 1 : 1
SHEET: 1 OF 1

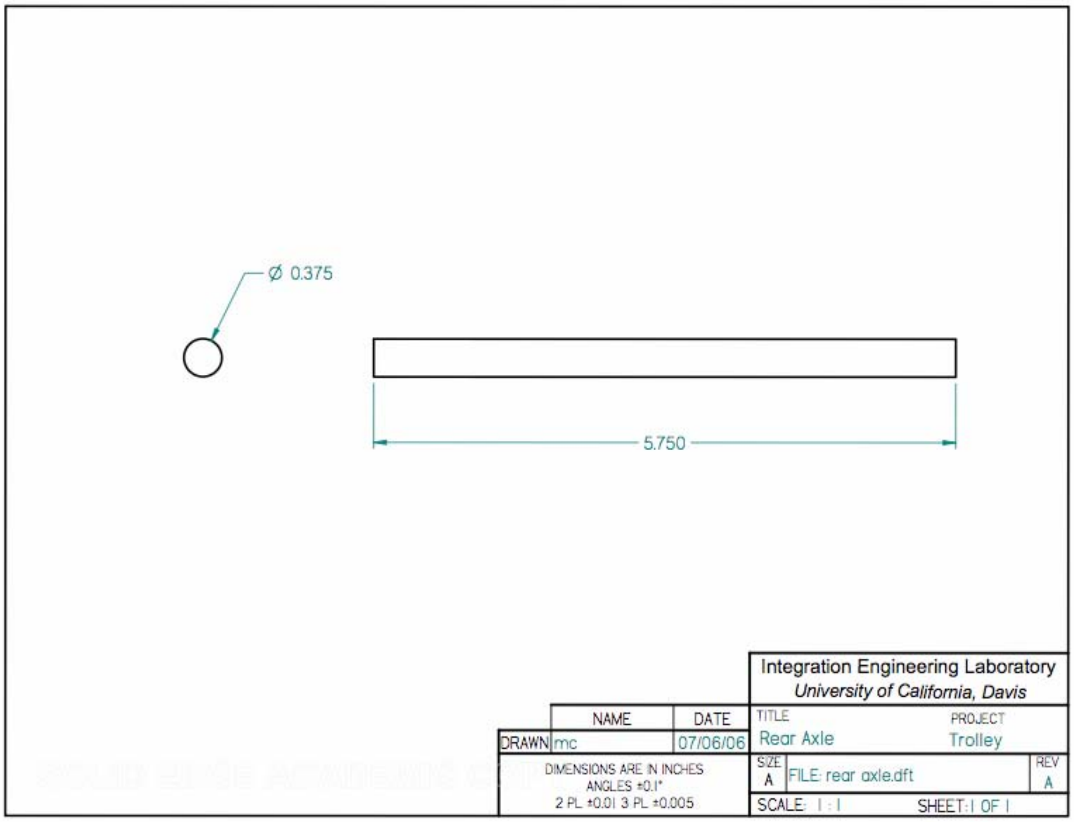

0.250
Ø 1.000
Ø 0.266
Integration Engineering Laboratory
University of California, Davis
NAME
DATE
DRAWN mc
07/18/06
TITLE
Mount Plate Block
PROJECT
Trolley
DIMENSIONS ARE IN INCHES
ANGLES ±0.1°
2 PL ±0.01 3 PL ±0.005
SIZE A
FILE: mounting plate block.dft
REV A
SCALE: 2 : 1
SHEET: 1 OF 1

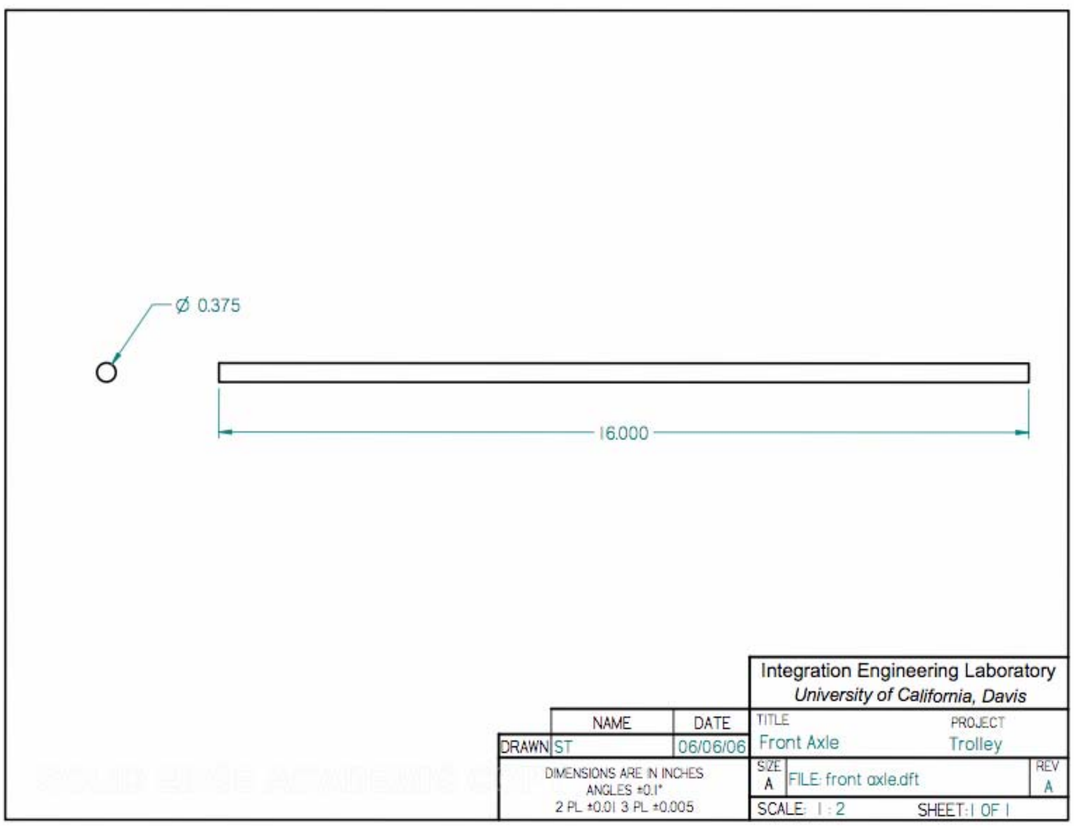

3x Ø 0.201 THRU
1.700
1.000
0.300
0.800
1.400
1.750
1.000
0.250
Ø 1.135 ⊽ 0.070
2x Ø 0.201 THRU
0.250
2.000
0.250
1.330
1.330
1.000
Ø 0.300 THRU
2.000
1.000
2.000
Integration Engineering Laboratory
University of California, Davis
NAME
DATE
DRAWN ST
06/06/06
TITLE
Motor Bracket
PROJECT
Trolley
DIMENSIONS ARE IN INCHES
ANGLES ±0.1°
2 PL ±0.01 3 PL ±0.005
SIZE A
FILE: motor bracket.dft
REV A
SCALE: 1 : 1
SHEET: 1 OF 1

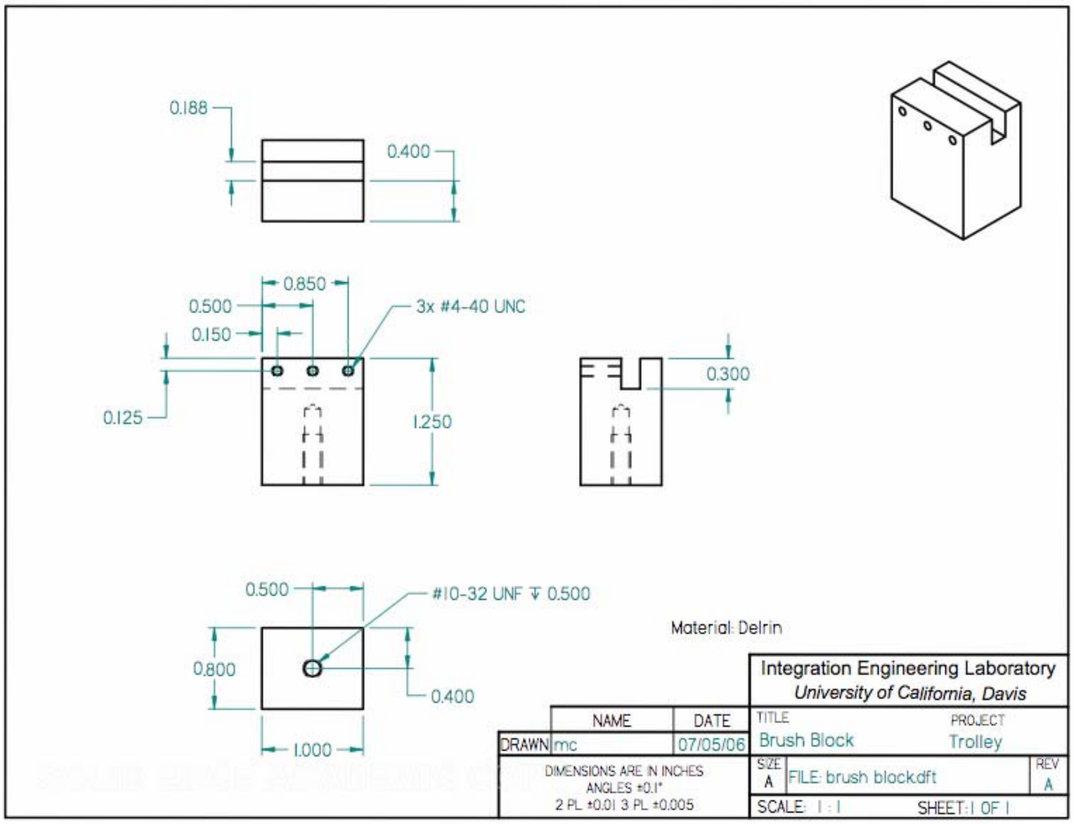

1.750
1.531
0.219
0.600
0.300
2x Ø 0.201 THRU
0.375
Integration Engineering Laboratory
University of California, Davis
NAME
DATE
DRAWN ST
06/06/06
TITLE
Gear Box Platform
PROJECT
Trolley
DIMENSIONS ARE IN INCHES
ANGLES ±0.1°
2 PL ±0.01 3 PL ±0.005
SIZE A
FILE: gear platform.dft
REV A
SCALE: 2 : 1
SHEET: 1 OF 1

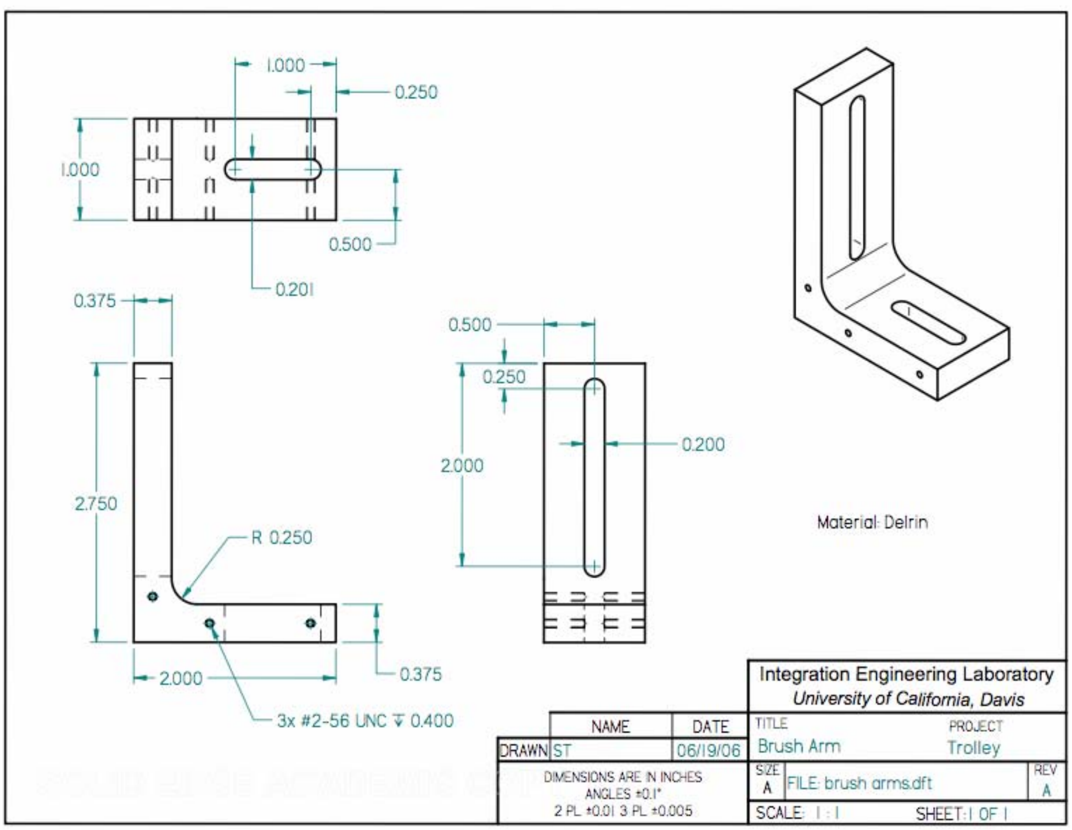

1.000
2x Ø 0.875 ↧ 0.312
fit 7/8 bearing
1.500
1.000
Ø 0.625 THRU
0.500
1.000
0.800
0.200
0.200
0.800
4x #8-32 UNC ↧ 0.400
tap drill to ↧ 0.500 max
Integration Engineering Laboratory
University of California, Davis
NAME
DATE
DRAWN ST
06/19/06
TITLE
Rear Axle Leg
PROJECT
Trolley
DIMENSIONS ARE IN INCHES
ANGLES ±0.1°
2 PL ±0.01 3 PL ±0.005
SIZE A
FILE: rear axle leg.dft
REV A
SCALE: 1 : 1
SHEET: 1 OF 1

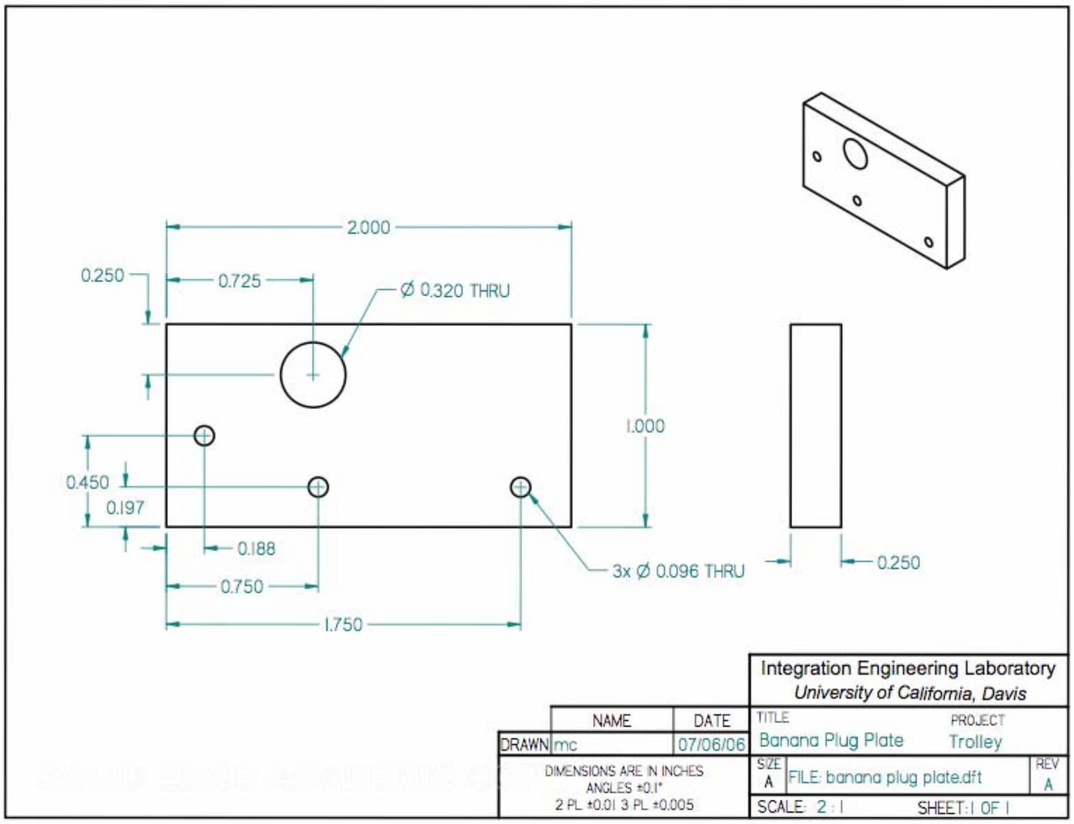

Ø 0.875 ↧ 0.312
fit 7/8 bearing
1.500
1.000
Ø 0.625 THRU
0.500
0.500
1.000
0.200
2x 0.250
0.800
2x #8-32 UNC ↧ 0.400
tap drill to ↧ 0.500 max
Integration Engineering Laboratory
University of California, Davis
NAME
DATE
DRAWN ST
06/19/06
TITLE
Front Axle Leg
PROJECT
Trolley
DIMENSIONS ARE IN INCHES
ANGLES ±0.1°
2 PL ±0.01 3 PL ±0.005
SIZE A
FILE: front axle leg.dft
REV A
SCALE: 1 : 1
SHEET: 1 OF 1

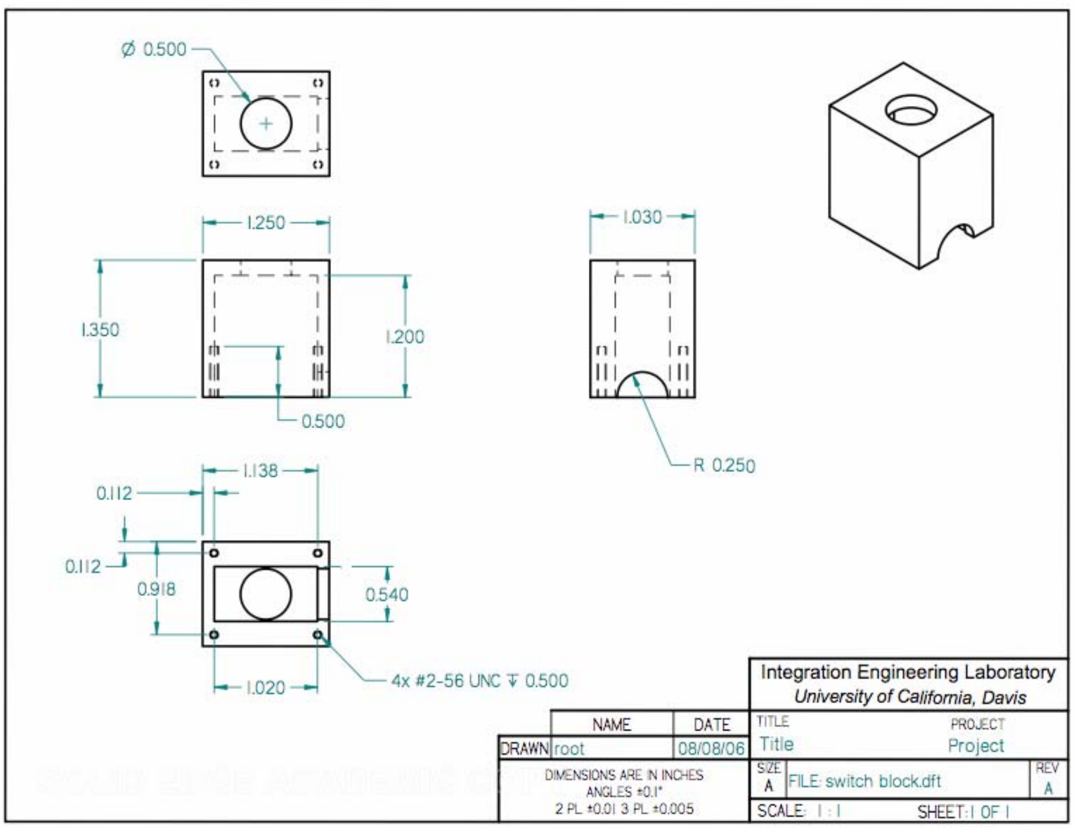

Ø 0.375
5.750
Integration Engineering Laboratory
University of California, Davis
NAME
DATE
DRAWN mc
07/06/06
TITLE
Rear Axle
PROJECT
Trolley
DIMENSIONS ARE IN INCHES
ANGLES ±0.1°
2 PL ±0.01 3 PL ±0.005
SIZE A
FILE: rear axle.dft
REV A
SCALE: 1 : 1
SHEET: 1 OF 1

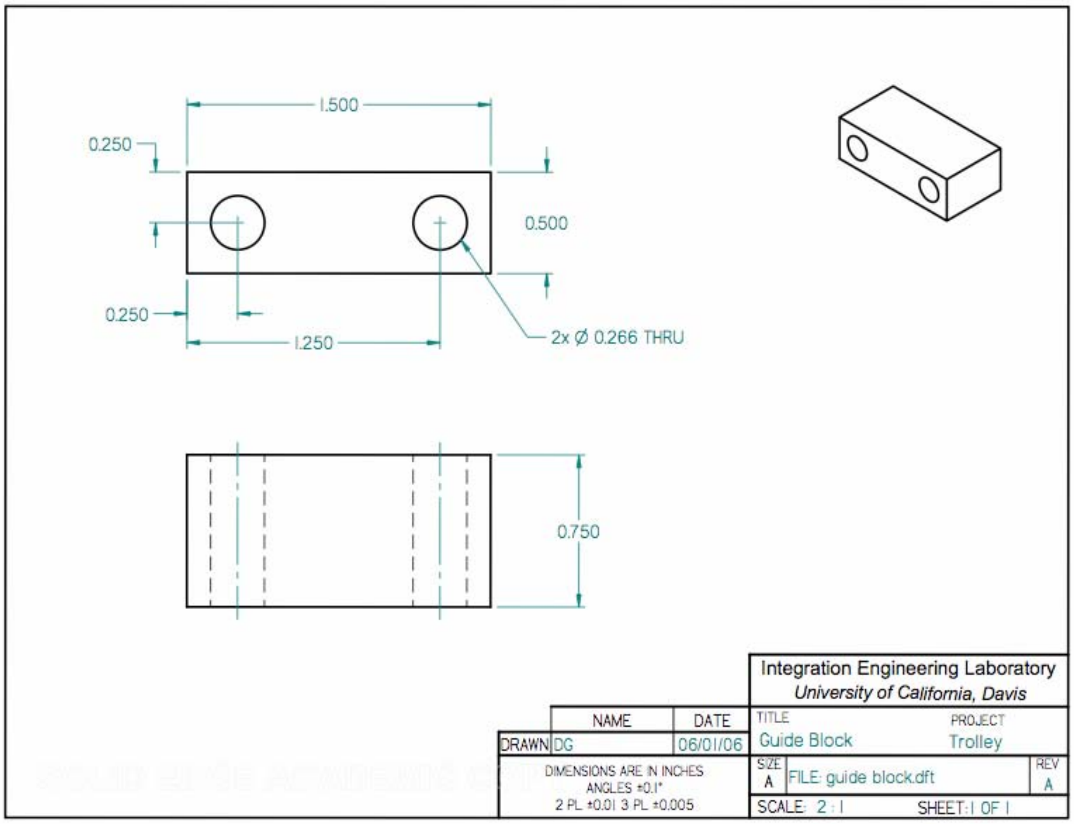

Ø 0.375
16.000
Integration Engineering Laboratory
University of California, Davis
NAME
DATE
DRAWN ST
06/06/06
TITLE
Front Axle
PROJECT
Trolley
DIMENSIONS ARE IN INCHES
ANGLES ±0.1°
2 PL ±0.01 3 PL ±0.005
SIZE A
FILE: front axle.dft
REV A
SCALE: 1 : 2
SHEET: 1 OF 1

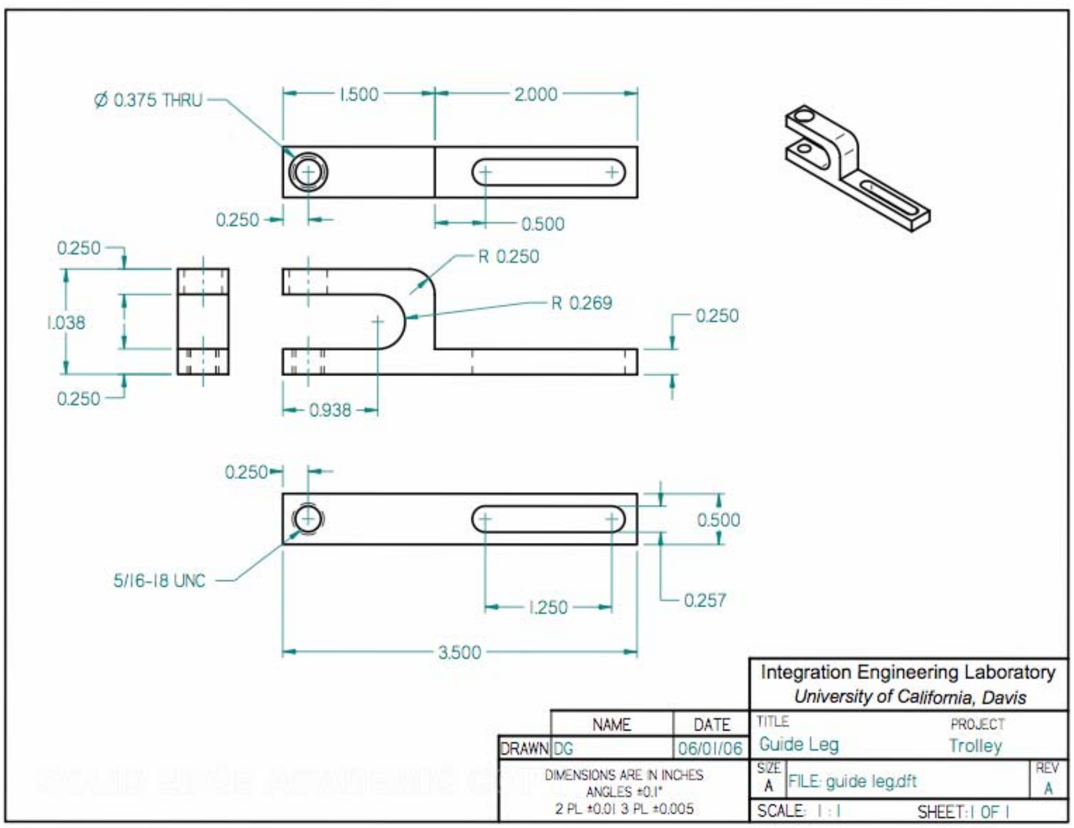

0.188
0.400
0.850
0.500
0.150
3x #4-40 UNC
0.125
1.250
0.300
0.500
#10-32 UNF ⌴ 0.500
0.800
0.400
1.000
Material: Delrin
Integration Engineering Laboratory
University of California, Davis
NAME
DATE
DRAWN mc
07/05/06
TITLE
Brush Block
PROJECT
Trolley
DIMENSIONS ARE IN INCHES
ANGLES ±0.1°
2 PL ±0.01 3 PL ±0.005
SIZE A
FILE: brush block.dft
REV A
SCALE: 1 : 1
SHEET: 1 OF 1

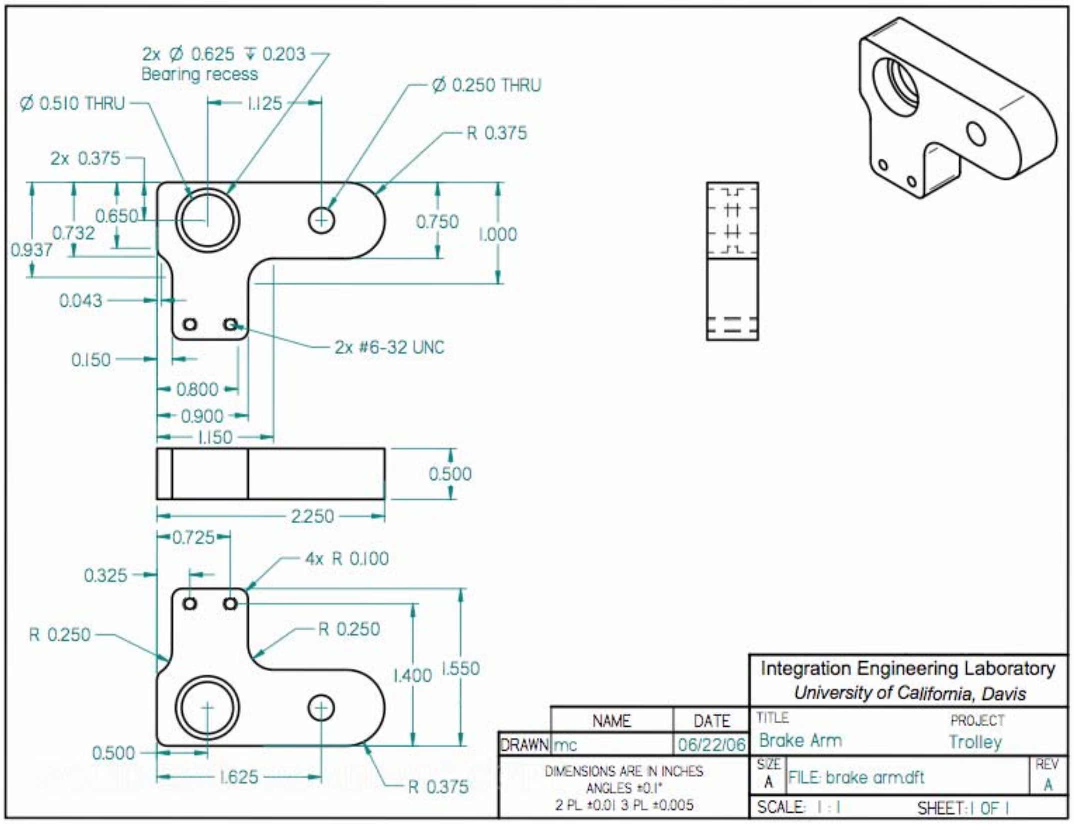

1.000
0.250
1.000
0.500
0.201
0.375
0.500
0.250
0.200
2.000
2.750
R 0.250
2.000
0.375
3x #2-56 UNC ⌴ 0.400
Material: Delrin
Integration Engineering Laboratory
University of California, Davis
NAME
DATE
DRAWN ST
06/19/06
TITLE
Brush Arm
PROJECT
Trolley
DIMENSIONS ARE IN INCHES
ANGLES ±0.1°
2 PL ±0.01 3 PL ±0.005
SIZE A
FILE: brush arms.dft
REV A
SCALE: 1 : 1
SHEET: 1 OF 1

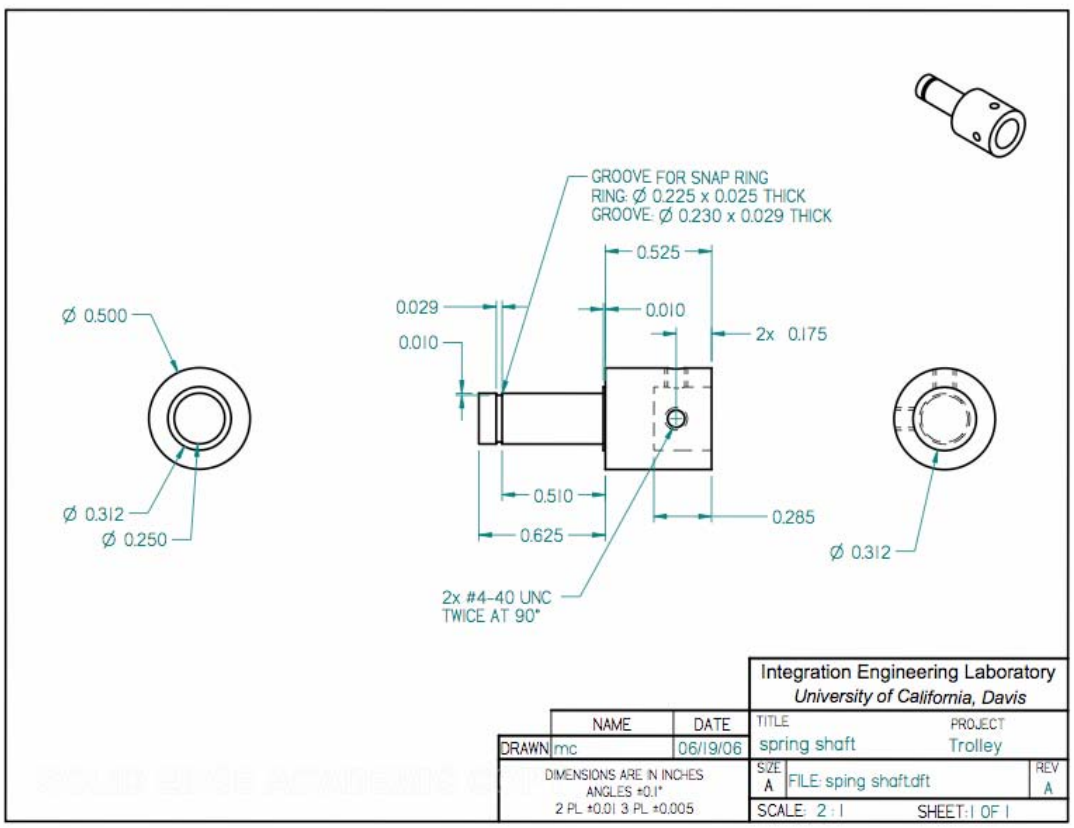

2.000
0.250
0.725
Ø 0.320 THRU
1.000
0.450
0.197
0.188
0.750
1.750
3x Ø 0.096 THRU
0.250
Integration Engineering Laboratory
University of California, Davis
NAME
DATE
DRAWN mc
07/06/06
TITLE
Banana Plug Plate
PROJECT
Trolley
DIMENSIONS ARE IN INCHES
ANGLES ±0.1°
2 PL ±0.01 3 PL ±0.005
SIZE A
FILE: banana plug plate.dft
REV A
SCALE: 2 : 1
SHEET: 1 OF 1

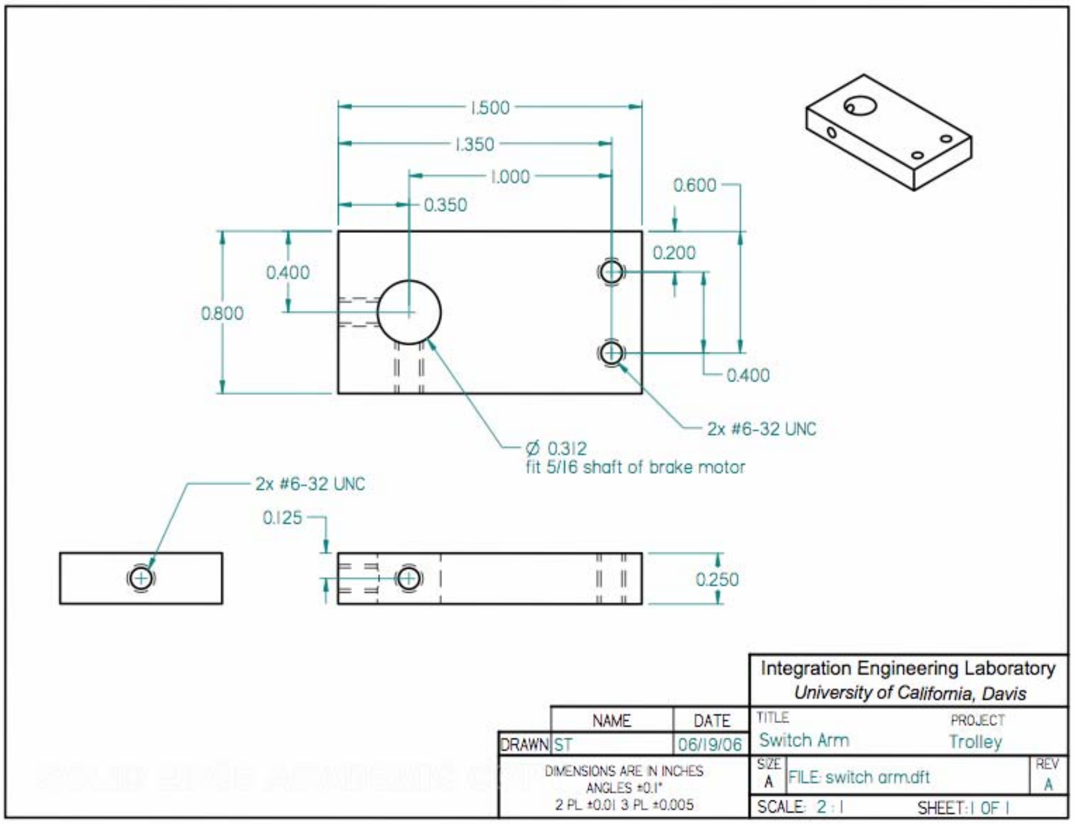

Ø 0.500
1.250
1.350
1.200
0.500
1.030
R 0.250
1.138
0.112
0.112
0.918
0.540
1.020
4x #2-56 UNC ↧ 0.500
Integration Engineering Laboratory
University of California, Davis
NAME
DATE
DRAWN root
08/08/06
TITLE
Title
PROJECT
Project
DIMENSIONS ARE IN INCHES
ANGLES ±0.1°
2 PL ±0.01 3 PL ±0.005
SIZE A
FILE: switch block.dft
REV A
SCALE: 1 : 1
SHEET: 1 OF 1

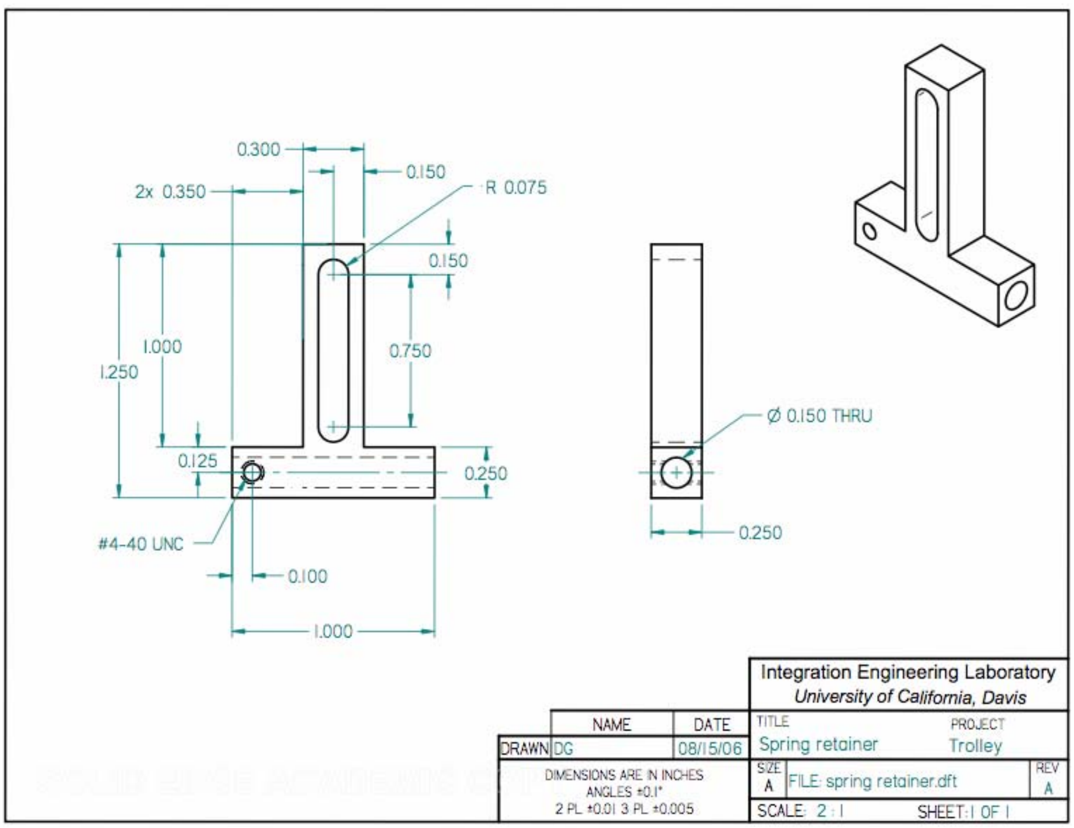

1.500
0.250
0.500
0.250
1.250
2x Ø 0.266 THRU
0.750
Integration Engineering Laboratory
University of California, Davis
NAME
DATE
DRAWN DG
06/01/06
TITLE
Guide Block
PROJECT
Trolley
DIMENSIONS ARE IN INCHES
ANGLES ±0.1°
2 PL ±0.01 3 PL ±0.005
SIZE A
FILE: guide block.dft
REV A
SCALE: 2 : 1
SHEET: 1 OF 1

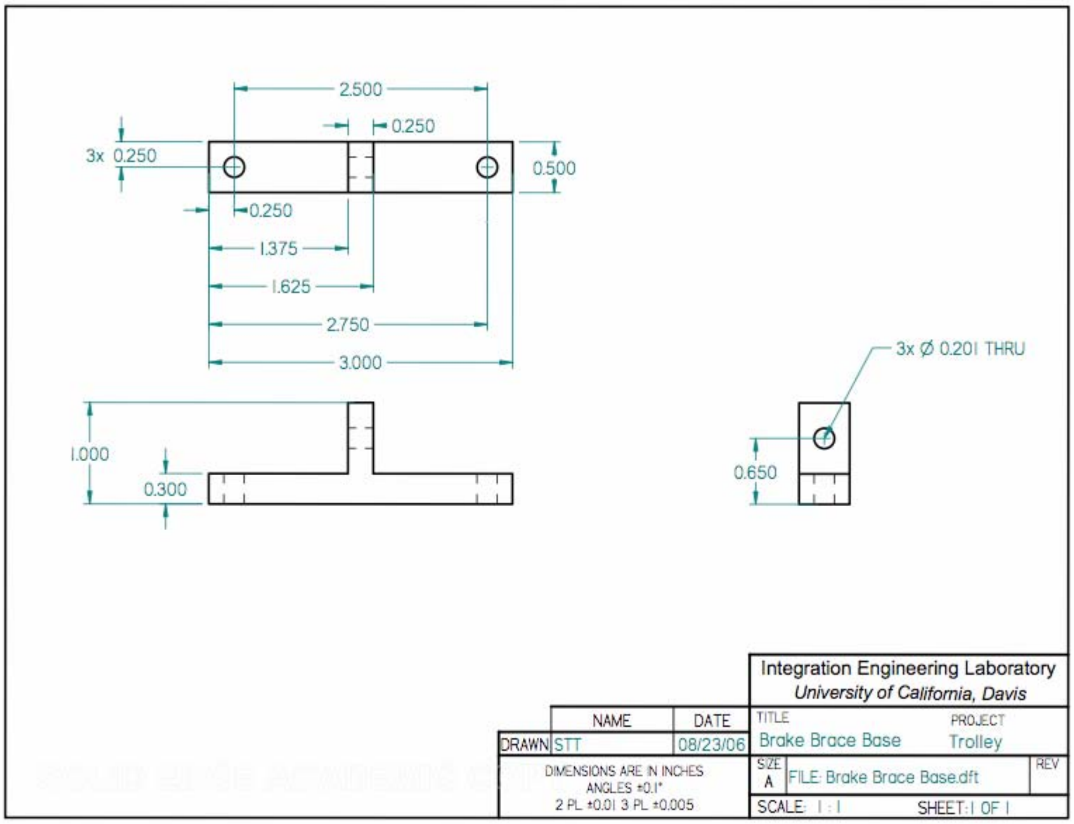

Ø 0.375 THRU
1.500
2.000
0.250
0.500
0.250
R 0.250
1.038
R 0.269
0.250
0.250
0.938
0.250
0.500
5/16-18 UNC
1.250
0.257
3.500
Integration Engineering Laboratory
University of California, Davis
NAME
DATE
DRAWN DG
06/01/06
TITLE
Guide Leg
PROJECT
Trolley
DIMENSIONS ARE IN INCHES
ANGLES ±0.1°
2 PL ±0.01 3 PL ±0.005
SIZE A
FILE: guide leg.dft
REV A
SCALE: 1 : 1
SHEET: 1 OF 1


2x Ø 0.625 ⌴ 0.203
Bearing recess
Ø 0.510 THRU
1.125
Ø 0.250 THRU
R 0.375
2x 0.375
0.650
0.732
0.937
0.750
1.000
0.043
2x #6-32 UNC
0.150
0.800
0.900
1.150
0.500
2.250
0.725
4x R 0.100
0.325
R 0.250
R 0.250
1.400
1.550
0.500
1.625
R 0.375
Integration Engineering Laboratory
University of California, Davis
NAME
DATE
DRAWN mc
06/22/06
TITLE
Brake Arm
PROJECT
Trolley
DIMENSIONS ARE IN INCHES
ANGLES ±0.1°
2 PL ±0.01 3 PL ±0.005
SIZE A
FILE: brake arm.dft
REV A
SCALE: 1 : 1
SHEET: 1 OF 1

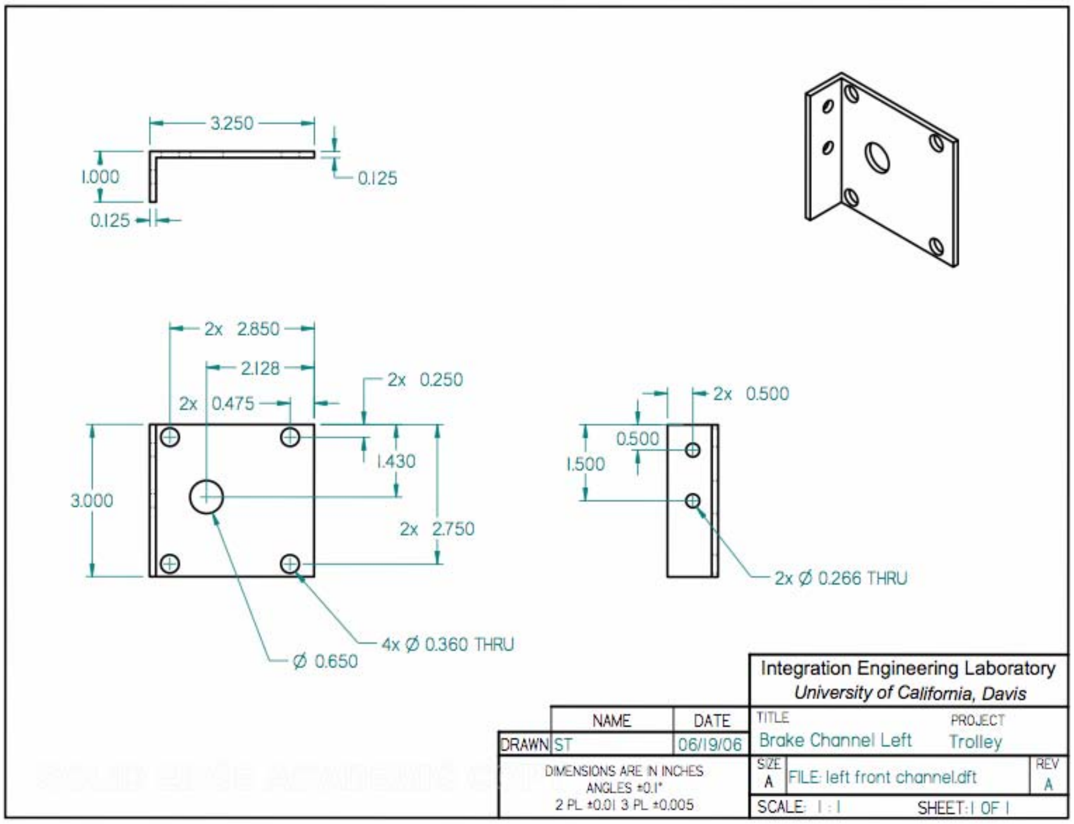

GROOVE FOR SNAP RING
RING: Ø 0.225 x 0.025 THICK
GROOVE: Ø 0.230 x 0.029 THICK
0.525
0.029
0.010
0.010
2x 0.175
Ø 0.500
Ø 0.312
Ø 0.250
0.510
0.625
0.285
Ø 0.312
2x #4-40 UNC
TWICE AT 90°
Integration Engineering Laboratory
University of California, Davis
NAME
DATE
DRAWN mc
06/19/06
TITLE
spring shaft
PROJECT
Trolley
DIMENSIONS ARE IN INCHES
ANGLES ±0.1°
2 PL ±0.01 3 PL ±0.005
SIZE A
FILE: sping shaft.dft
REV A
SCALE: 2 : 1
SHEET: 1 OF 1

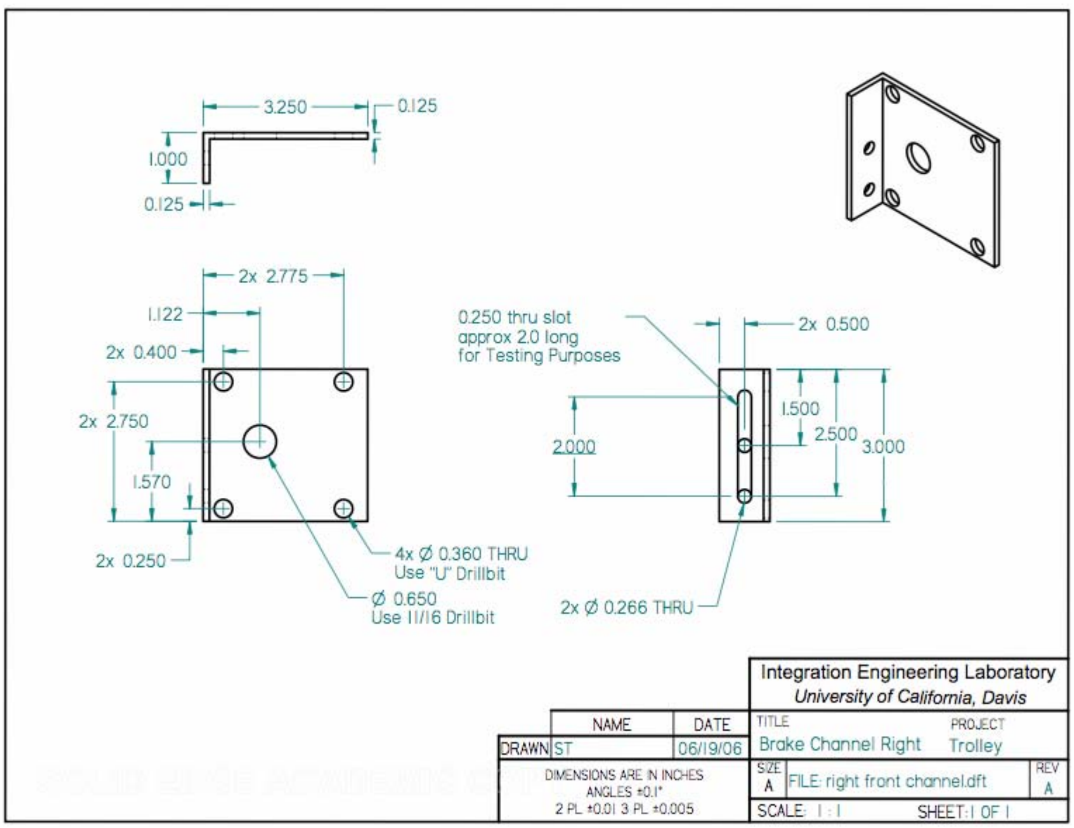

1.500
1.350
1.000
0.350
0.600
0.200
0.400
0.800
0.400
2x #6-32 UNC
Ø 0.312
fit 5/16 shaft of brake motor
2x #6-32 UNC
0.125
0.250
Integration Engineering Laboratory
University of California, Davis
NAME
DATE
DRAWN ST
06/19/06
TITLE
Switch Arm
PROJECT
Trolley
DIMENSIONS ARE IN INCHES
ANGLES ±0.1°
2 PL ±0.01 3 PL ±0.005
SIZE A
FILE: switch arm.dft
REV A
SCALE: 2 : 1
SHEET: 1 OF 1


0.300
0.150
2x 0.350
R 0.075
0.150
1.000
1.250
0.750
0.125
0.250
#4-40 UNC
0.100
1.000
Ø 0.150 THRU
0.250
Integration Engineering Laboratory
University of California, Davis
NAME
DATE
DRAWN DG
08/15/06
TITLE
Spring retainer
PROJECT
Trolley
DIMENSIONS ARE IN INCHES
ANGLES ±0.1°
2 PL ±0.01 3 PL ±0.005
SIZE A
FILE: spring retainer.dft
REV A
SCALE: 2 : 1
SHEET: 1 OF 1



2.500
0.250
3x 0.250
0.500
0.250
1.375
1.625
2.750
3.000
3x Ø 0.201 THRU
1.000
0.300
0.650
Integration Engineering Laboratory
University of California, Davis
NAME
DATE
DRAWN STT 08/23/06
DIMENSIONS ARE IN INCHES
ANGLES ±0.1°
2 PL ±0.01 3 PL ±0.005
TITLE
Brake Brace Base
PROJECT
Trolley
SIZE A
FILE: Brake Brace Base.dft
REV
SCALE: 1 : 1
SHEET: 1 OF 1


3.250
1.000
0.125
0.125
2x 2.850
2.128
2x 0.250
2x 0.475
1.430
3.000
2x 2.750
Ø 0.650
4x Ø 0.360 THRU
2x 0.500
0.500
1.500
2x Ø 0.266 THRU
Integration Engineering Laboratory
University of California, Davis
NAME DATE
DRAWN ST 06/19/06
TITLE Brake Channel Left
PROJECT Trolley
DIMENSIONS ARE IN INCHES
ANGLES ±0.1°
2 PL ±0.01 3 PL ±0.005
SIZE A
FILE: left front channel.dft
REV A
SCALE: 1 : 1
SHEET: 1 OF 1


3.250
0.125
1.000
0.125
2x 2.775
1.122
2x 0.400
2x 2.750
1.570
2x 0.250
4x Ø 0.360 THRU
Use "U" Drillbit
Ø 0.650
Use 11/16 Drillbit
0.250 thru slot
approx 2.0 long
for Testing Purposes
2x 0.500
1.500
2.500
3.000
2.000
2x Ø 0.266 THRU
Integration Engineering Laboratory
University of California, Davis
NAME DATE
DRAWN ST 06/19/06
TITLE Brake Channel Right
PROJECT Trolley
DIMENSIONS ARE IN INCHES
ANGLES ±0.1°
2 PL ±0.01 3 PL ±0.005
SIZE A
FILE: right front channel.dft
REV A
SCALE: 1 : 1
SHEET: 1 OF 1

Appendix B: Parts List

Purchased Parts

Part	Description	Supplier	Part #	Qty	$	Total
Drive Motor	Pittman Lo-Cog Gearmotor, 24volt, 65.5:1	Pittman	GM9236S026	1	$117.12	$117.12
Gear Box	dual output, right angle, 1:1	McMaster	6456K23	1	$169.78	$169.78
Gear Box	dual output, right angle, 1:1 alternate part	Boston Gear	RA631	0	$256.90	$0.00
Axles	Hardened Stainless Steel Shaft, od 0.375", length 18"	McMaster	6253K33	2	$26.74	$53.48
Axle Bearings	ABEC-1, double sealed, id 0.25", od 0.625", bearing no. R6	McMaster	60355K35	6	$4.80	$28.80
Axle Collars	One-piece Clamp-on Shaft Collar, 303 Stainless Steel, 3/8" bore, 7/8" OD	McMaster	6435K33	6	$4.95	$29.70
Brake Bearing	ABEC-1, double sealed, id 0.25", od 0.625", bearing no. R4	McMaster	60355K33	4	$4.77	$19.08
Alignment Bearings	Delrin bearing, stainless steel balls, id 0.375", od 1.375"	McMaster	6455K29	4	$5.28	$21.12
Wheels	Neoprene drive roller, durometer 80A, od 1.25", width 1.25, bore 0.375"	McMaster	2474K31	4	$26.62	$106.48
Drive Shaft Coupler	Stainless Steel clamp-on shaft coupler, bore 0.375"	McMaster	61005K42	2	$31.12	$62.24
Motor Shaft Coupler	Stainless Steel clamp-on shaft coupler, bore 0.375" x 0.25"	McMaster	61005K63	1	$32.41	$32.41
Brake Gear Motor	sub-fractional hp DC gearmotor, 24 volt, 50 in-lb, 4 rpm	McMaster	6409K22	2	$42.03	$84.06
Brake Spring	Stainless Steel torsion spring, 90 deg, CCW/CW, 20 in-lb	McMaster	9287K103	2	$8.60	$17.20

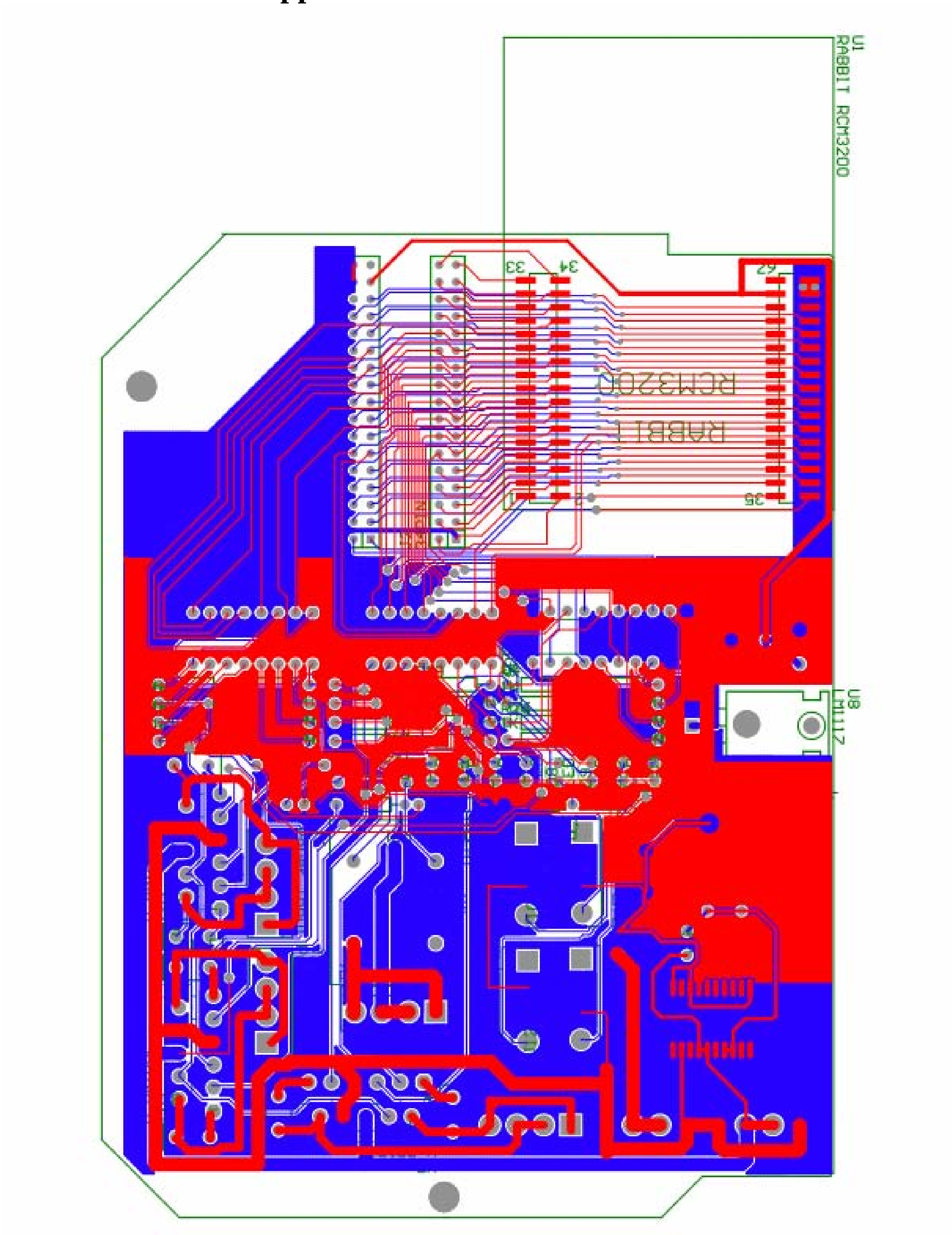
Brake Arm Snap Rings	Stainless Steel External Retaining Ring for 1/4" shaft dia.	McMaster	91590A113	1	$7.00	$7.00
Brake switch	Washdown Subminiature Snap-Acting Switch Spdt, Rigid Lever	McMaster	8085T13	4	$5.43	$21.72
Coating for brake arm	"Plasti Dip" plastic coating	Ace Hardware	N/A	1	$7.00	$7.00
Alignment Shoulder Bolt	self-locking stainless steel shoulder bolt, dia 0.375", length 0.75"	McMaster	91327A156	4	$5.11	$20.44
Brushes	Bronze strip brush, Galvanized channel	Gordon Brush	sample	2	$0.00	$0.00
Batteries - 12v sealed type A512/1.2s	12v sealed type A512/1.2s or equivalent	Sonnenschein	A512/1.2S	2	$15.00	$30.00
Electronic Box	Sealed electrical box with metric knockouts 7.1" x 5.1" x 2.4"	Fibox	PCM150/60T	1	$30.00	$30.00
Electrical Wire Grips	Cord Grip, M-16 Size, Core Dia Range 0.18"-0.39"	McMaster	7310K32	3	$3.68	$11.04
Plastic Tape sample	Adhesive backed plastic strip to isolate power supply	Saint-Gobain	CHR 2302	1	$0.00	$0.00
Aluminum Tape sample	Adhesive backed aluminum strip to conduct power	Saint-Gobain	CHR 26020	1	$0.00	$0.00
Total						***$868.67***

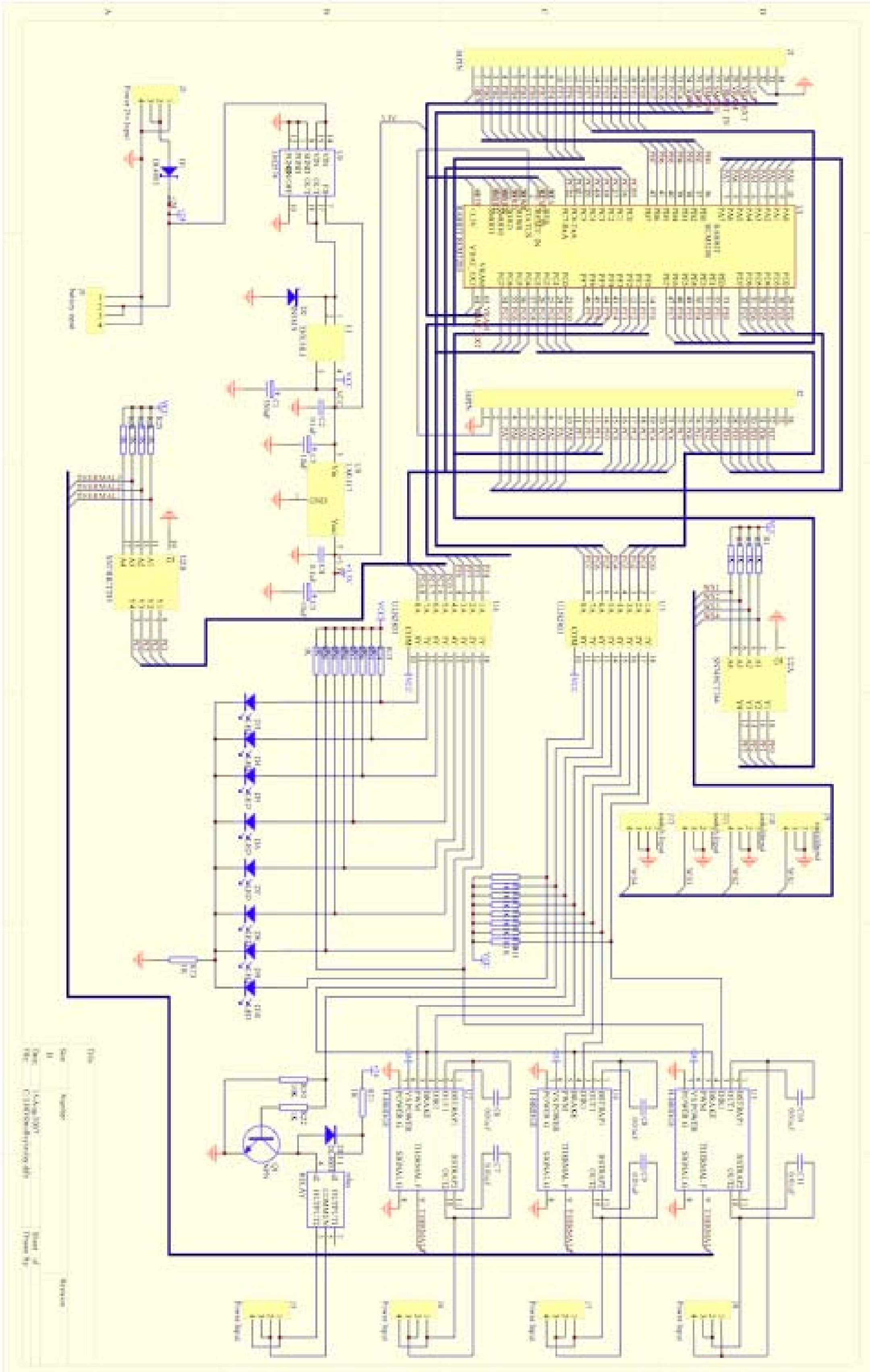
Fabricated Parts

Part	Description	Drawing	Qty	Material
Body	Main trolley body	Trolley Body.dft	1	Al plate 0.25"
Mounting Plate	Detector mounting plate	mounting plate.dft	1	Al plate 0.25"
Mounting Plate Block	Spacer between Body and Mounting Plate	mounting plate block.dft	4	Al bar 1" x 0.25"
Motor Bracket	Drive motor bracket	motor bracket.dft	1	Al L-bracket 0.25"
Gear Box Platform	Block under gear box	gear platform.dft	1	Al block
Front Axle Leg	Front axle support, free spin	front axle leg.dft	2	Al bar 1" x 0.5"
Rear Axle Leg	Rear axle support, driven wheels	rear axle leg.dft	2	Al bar 1" x 1"
Front Axle	Axle length	front axle.dft	1	SS bar 3/8" dia
Rear Axle	Axle length	rear axle.dft	2	SS bar 3/8" dia
Brush Block	Delrin brush holder	brush block.dft	2	Delrin bar 1" x 1"
Brush Arm	Delrin mount for brush assembly	brush arm.dft	2	Delrin bar 1" x 3"
Banana Plug Plate	Delrin piece for aux banana plug	banana plug plate.dft	2	Delrin plate 0.25"
Switch Block	Delrin block holding battery switch	switch block.dft	1	Delrin bar 1.5" x 1.5"
Guide Block	attachment block for alignment wheels	guide block.dft	4	Al bar 1" x 0.5"
Guide Leg	forked leg holding alignment wheels	guide leg.dft	4	Al bar 1.5" x 1"
Brake Arm	Brake arm with rubber suface	brake arm.dft	2	Al bar 1.5" x 1"
Spring Shaft	Shaft between switch arm and brake arm	spring shaft.dft	2	Al rod 0.75"
Switch Arm	Inner arm on brakes to actuate the switches	switch arm.dft	2	Al bar 0.5" x 0.25"
Spring Retainers	Piece hold brake spring in place	spring retainer.dft	4	Al plate 0.25"
Brake Brace Base	Brace mounted on back of brake	brake brace base.dft	2	Al bar 1" x 0.5"

	motor			
Brake Brace Arm	Link connecting brake braces together	built to fit	2	Al bar 0.5" x 0.25"
Brake Brace Connector	Connecting piece to fix brace arms to body	built to fit	1	Al bar 1" x 0.25"
Brake Channel Left	bracket supporting the left brake	left front channel.dft	1	Al Channel 4" x 1"
Brake Channel Right	bracket supporting the right brake	right front channel.dft	1	Al Channel 4" x 1"

Appendix C: Electrical Schematic





Appendix D: Trolley Opcodes

```
// System Types
 #define TYPE_LBDS                    0xA0  // LBDS System
 #define TYPE_TRLY                    0xA1  // Trolley System

 // Trolley Messages to return to user
 // Trolley Data Messages -> data/message to follow
 #define TRM_SPEED                    0xB0  // Sending drive motor duty cycle
 #define TRM_BRAKES_ON                0xB1  // Brakes are fully actuated
 #define TRM_BRAKES_OFF               0xB2  // Brakes are fully open
 #define TRM_DET_ENABLE               0xB3  // Power to detector enabled
 #define TRM_DET_DISABLE              0xB4  // Power to detector enabled
 #define TRM_CONN_CLOSING             0xB5  // Closing TCP connection

 // Trolley control commands -> Trolley is Rear Wheel Drive
 #define TRC_STEP_FORWARD             0xC0  // Step the trolley forward
 #define TRC_STEP_BACKWARD            0xC1  // Step the trolley backward
 #define TRC_CONT_FORWARD             0xC2  // Continuously move Forward
 #define TRC_CONT_BACKWARD            0xC3  // Continuously move Backward
 #define TRC_BRAKE                    0xC4  // Engage trolley brakes
 #define TRC_UNBRAKE                  0xC5  // Disengage trolley brakes
 #define TRC_SPEED_UP                 0xC6  // Speed up the trolley
 #define TRC_SPEED_DOWN               0xC7  // Slow down the trolley
 #define TRC_STOP                     0xC8  // Stop trolley motion
 #define TRC_DETECTOR_POWER_ON        0xC9  // Turn on power to detector
 #define TRC_DETECTOR_POWER_OFF       0xCA  // Turn off power to detector
 #define TRC_CLOSE_CONN               0xCB  // Close the TCP/IP Conn

 // Trolley errors
 #define TRE_NET_OK                   0xE0  // Communications established
 #define TRE_NET_FAIL                 0xE1  // Communication failure
 #define TRE_NET_DHCP_FB              0xE2  // DHCP address with fallbacks
 #define TRE_NET_DHCP_NFB             0xE3  // DHCP address without fallbacks
 #define TRE_BRAKE_FAIL               0xE4  // Brakes not working properly
 #define TRE_MOTOR_FAIL               0xE5  // Drive motor not responding
```

Appendix E: Trolley Operator Control Program

Readme

```
Authors: David Ko & Stephen S. Nestinger & Matt Campbell

Created: Sept. 24, 2004
Updated: September 18, 2006

GTK based Trolley Control Software v. 1.1.0
```

Header files

callbacks.h

```
#ifndef _CALLBACKS_H_
#define _CALLBACKS_H_

#include <gtk/gtk.h>        // Required for GTK
#include <stdio.h>          // For printf

#ifdef DEBUG                // Defined during compilation
#define DEBUG1(x) printf(x) // Used for debugging purposes
#else
#define DEBUG1 //
#endif


void trolleyControl_Quit(void);

void
send_message                        (short int  message);

void
key_press_event                      (GtkWidget       *widget,
                                        GdkEventKey   *event,
                              gpointer    data);
void
about_button_activate                   (GtkMenuItem     *menuitem,
                                        gpointer         user_data);

void
start_server_activate                     (GtkMenuItem     *menuitem,
                                        gpointer         user_data);

void
stop_server_activate                    (GtkMenuItem     *menuitem,
                                        gpointer         user_data);

void
on_continuous_forward_button_pressed      (GtkButton       *button,
                                        gpointer         user_data);

void
on_continuous_forward_button_released     (GtkButton       *button,
                                        gpointer         user_data);
void
on_continuous_backward_button_pressed      (GtkButton       *button,
                                         gpointer         user_data);

void
on_continuous_backward_button_released     (GtkButton       *button,
                                         gpointer         user_data);

void
on_single_step_backward_button_pressed      (GtkButton       *button,
                                          gpointer         user_data);

void
on_single_step_backward_button_released     (GtkButton       *button,
                                          gpointer         user_data);

void
on_single_step_forward_button_pressed      (GtkButton       *button,
```

```
                                         gpointer         user_data);

void
on_single_step_forward_button_released     (GtkButton       *button,
                                         gpointer         user_data);

void
on_stop_button_pressed              (GtkButton       *button,
                                    gpointer         user_data);

void
on_increase_speed_button_pressed      (GtkButton       *button,
                                       gpointer         user_data);

void
on_increase_speed_button_released     (GtkButton       *button,
                                       gpointer         user_data);

void
on_decrease_speed_button_pressed      (GtkButton       *button,
                                       gpointer         user_data);

void
on_decrease_speed_button_released     (GtkButton       *button,
                                       gpointer         user_data);

void
on_brake_on_button_pressed      (GtkButton       *button,
                                 gpointer         user_data);

void
on_brake_on_button_released     (GtkButton       *button,
                                 gpointer         user_data);

void
on_brake_off_button_pressed      (GtkButton       *button,
                                  gpointer         user_data);

void
on_brake_off_button_released     (GtkButton       *button,
                                  gpointer         user_data);

void
on_detecpw_button_toggle      (GtkButton       *button,
                                  gpointer         user_data);

#endif
```

control.h

```
#ifndef _CONTROL_H_
#define _CONTROL_H_
// included to translate Big to Little Endian
// Didn't work as expected
//#include </usr/include/sys/_endian.h>

  // System Types
  #define TYPE_LBDS              0xA0   // LBDS System
  #define TYPE_TRLY              0xA1   // Trolley System
  #define TYPE_SERV              0xA2   // Trolley System

  // Trolley Data Messages -> data/message to follow
```

```
  #define TRM_SPEED              0xB0   // Sending drive motor duty
cycle
  // Trolley Messages to return to user
  #define TRM_BRAKES_ON          0xB1   // Brakes are fully actuated
  #define TRM_BRAKES_OFF         0xB2   // Brakes are fully open
  #define TRM_DET_ENABLE         0xB3   // Power to detector enabled
  #define TRM_DET_DISABLE        0xB4   // Power to detector disable

  // Trolley control commands -> Trolley is Rear Wheel Drive
  #define TRC_STEP_FORWARD       0xC0   // Step the trolley forward
  #define TRC_STEP_BACKWARD      0xC1   // Step the trolley backward
  #define TRC_CONT_FORWARD       0xC2   // Continously move the trolley
Forward
  #define TRC_CONT_BACKWARD      0xC3   // Continously move the trolley
Backward
  #define TRC_BRAKE              0xC4   // Engage trolley brakes
  #define TRC_UNBRAKE            0xC5   // Disengage trolley brakes
  #define TRC_SPEED_UP           0xC6   // Speed up the trolley
  #define TRC_SPEED_DOWN         0xC7   // Slow down the trolley
  #define TRC_STOP               0xC8   // Stop trolley motion
  #define TRC_DETECTOR_POWER_ON  0xC9   // Turn on power to detector
  #define TRC_DETECTOR_POWER_OFF 0xCA   // Turn off power to detector
  #define TRC_CLOSE_CONN         0xCB


  // Trolley errors
  #define TRE_NET_OK             0xE0   // Communications established
  #define TRE_NET_FAIL           0xE1   // Communication failure
  #define TRE_NET_DHCP_FB        0xE2   // DHCP address with fallbacks
  #define TRE_NET_DHCP_NFB       0xE3   // DHCP address without
fallbacks
  #define TRE_BRAKE_FAIL         0xE4   // Brakes not working properly
  #define TRE_MOTOR_FAIL         0xE5   // Drive motor not responding


  extern char *MSG_A[];
  extern char *MSG_B[];
  extern char *MSG_C[];
  extern char *MSG_E[];

#endif
```

icons.h

```
/* XPM */
static char * CF_xpm[] = {
"21 22 68 1",
"     c None",
".    c #FFFFFF",
"+    c #88CEFF",
"@    c #3A5CFF",
"#    c #92E1FF",
"$    c #7CC9FF",
"%    c #3753FF",
"&    c #E5EEFF",
"*    c #7CBFFF",
"=    c #1B2EFF",
"-    c #2C46FF",
";    c #1727FF",
">    c #2D45FF",
",    c #2A45FF",
"'    c #2239FF",
")    c #2133FF",
```

```
"!    c #2032FF",
"~    c #101BF0",
"{    c #080D79",
"]    c #1D2EFF",
"^    c #1622FF",
"/    c #121DEE",
"(    c #030636",
"_    c #1726FF",
":    c #1421FF",
"<    c #101CF7",
"[    c #101BED",
"}    c #091086",
"|    c #020428",
"1    c #111DFF",
"2    c #0F1AE1",
"3    c #0F19DF",
"4    c #010427",
"5    c #0E19D8",
"6    c #74B6FF",
"7    c #83D0FF",
"8    c #77B9FF",
"9    c #0E18D6",
"0    c #0E18D7",
"a    c #253DFF",
"b    c #253AFF",
"c    c #1828FF",
"d    c #1624FF",
"e    c #1321FF",
"f    c #131EFF",
"g    c #101CF5",
"h    c #101BF1",
"i    c #020324",
"j    c #070B61",
"k    c #0B139D",
"l    c #0D16BE",
"m    c #0E19D1",
"n    c #0F19DD",
"o    c #0F1AE5",
"p    c #0F1AE6",
"q    c #0B13A3",
"r    c #02021C",
"s    c #070B5F",
"t    c #0F19DA",
"u    c #010426",
"v    c #0D15B5",
"w    c #010324",
"x    c #080D78",
"y    c #030531",
"z    c #010111",
"A    c #070B5C",
"B    c #030635",
"C    c #00010F",
"                     ",
"                     ",
"          .          ",
"         .+@         ",
"        .#$%&&.....  ",
"       .#*==-;-;-;>  ",
"      .#*,'=)!)!)~{  ",
"     .#*,'];^;^/~/(  ",
"    .#*,']_:;<~~[}|  ",
"   .#*,']_:1<<~23}4  ",
"  .#*,']_:1<<~~25}4  ",
" 678,']_:1<~<~290}4  ",
```

```
"  abcdefgh[~~~290}4  ",
"   ijklmn2op~299q}4  ",
"    rsklmn2o299qq}4  ",
"     rsklmn299qq}}4  ",
"      rsklmtqqq}}}u  ",
"       rsklvq}}}}}w  ",
"        rskx(y(y(yz  ",
"         rAB         ",
"          C          ",
"                     "};
/* XPM */
static char * CB_xpm[] = {
"21 22 68 1",
"     c None",
".    c #FFFFFF",
"+    c #3A5CFF",
"@    c #88CEFF",
"#    c #E5EEFF",
"$    c #3753FF",
"%    c #7CC9FF",
"&    c #92E1FF",
"*    c #2D45FF",
"=    c #1727FF",
"-    c #2C46FF",
";    c #1B2EFF",
">    c #7CBFFF",
",    c #080D79",
"'    c #101BF0",
")    c #2133FF",
"!    c #2032FF",
"~    c #2239FF",
"{    c #2A45FF",
"]    c #030636",
"^    c #121DEE",
"/    c #1622FF",
"(    c #1D2EFF",
"_    c #020428",
":    c #091086",
"<    c #101BED",
"[    c #101CF7",
"}    c #1421FF",
"|    c #1726FF",
"1    c #010427",
"2    c #0F19DF",
"3    c #0F1AE1",
"4    c #111DFF",
"5    c #0E19D8",
"6    c #0E18D7",
"7    c #0E18D6",
"8    c #77B9FF",
"9    c #83D0FF",
"0    c #74B6FF",
"a    c #101BF1",
"b    c #101CF5",
"c    c #131EFF",
"d    c #1321FF",
"e    c #1624FF",
"f    c #1828FF",
"g    c #253AFF",
"h    c #253DFF",
"i    c #0B13A3",
"j    c #0F1AE6",
"k    c #0F1AE5",
"l    c #0F19DD",
```

```
"m    c #0E19D1",
"n    c #0D16BE",
"o    c #0B139D",
"p    c #070B61",
"q    c #020324",
"r    c #070B5F",
"s    c #02021C",
"t    c #010426",
"u    c #0F19DA",
"v    c #010324",
"w    c #0D15B5",
"x    c #010111",
"y    c #030531",
"z    c #080D78",
"A    c #030635",
"B    c #070B5C",
"C    c #00010F",
"                     ",
"                     ",
"          .          ",
"         +@.         ",
"  .....##$%&.        ",
"  *=-=-=-;;>&.       ",
"  ,')!)!);~{>&.      ",
"  ]^'^/=/=(~{>&.     ",
"  _:<''[=}|(~{>&.    ",
"  1:23'[[4}|(~{>&.   ",
"  1:53''[[4}|(~{>&.  ",
"  1:673'['[4}|(~{890 ",
"  1:673''<abcdefgh   ",
"  1:i773'jk3lmnopq   ",
"  1:ii773k3lmnors    ",
"  1::ii773lmnors     ",
"  t:::iiiumnors      ",
"  v:::::iwnors       ",
"  xy]y]y]zors        ",
"         ABs         ",
"          C          ",
"                     "};
/* XPM */
static char * DN_xpm[] = {
"24 24 142 2",
"     c None",
".    c #030635",
"+    c #040846",
"@    c #090E82",
"#    c #0F18BB",
"$    c #2239FF",
"%    c #74B4FF",
"&    c #8EDBFF",
"*    c #E4F8FF",
"=    c #080D70",
"-    c #0B14A2",
";    c #0D16C0",
">    c #0E19D1",
",    c #1A2CFB",
"'    c #324DFF",
")    c #395AFF",
"!    c #BDC9FF",
"~    c #03052E",
"{    c #060A59",
"]    c #0A1192",
"^    c #0E17BD",
"/    c #1829E8",
```

```
"(	c #4A75FD",
"_	c #7EC2FF",
":	c #B8E8FF",
"<	c #E0F4FE",
"[	c #080D71",
"}	c #0A108B",
"|	c #0C15B2",
"1	c #0E19CA",
"2	c #1B2EFD",
"3	c #4F7EFF",
"4	c #5F93FF",
"5	c #CADCFF",
"6	c #020427",
"7	c #060A54",
"8	c #0A118D",
"9	c #0C15B4",
"0	c #0E18CA",
"a	c #192BEB",
"b	c #507DFC",
"c	c #BDE9FF",
"d	c #E3F6FF",
"e	c #0A1088",
"f	c #0C15B0",
"g	c #0F19CA",
"h	c #1727E5",
"i	c #253EFF",
"j	c #3D61FF",
"k	c #79C0FF",
"l	c #A9DDFF",
"m	c #DEF1FF",
"n	c #03042B",
"o	c #0A1191",
"p	c #0C15B6",
"q	c #1A2DED",
"r	c #5382FC",
"s	c #80C5FF",
"t	c #BFEBFF",
"u	c #7682B8",
"v	c #091086",
"w	c #0C15AF",
"x	c #0F19C9",
"y	c #1626E2",
"z	c #263FFF",
"A	c #3E64FF",
"B	c #7CBFFF",
"C	c #A6E3FF",
"D	c #E6F8FF",
"E	c #030327",
"F	c #070B5C",
"G	c #0A1295",
"H	c #0D15B7",
"I	c #0E18CB",
"J	c #1B2EEF",
"K	c #5686FD",
"L	c #83C9FF",
"M	c #4D78D0",
"N	c #0C15AE",
"O	c #0E19C8",
"P	c #1524E1",
"Q	c #3A5EFF",
"R	c #7FC3FF",
"S	c #9DE2FF",
"T	c #EDFAFF",
"U	c #020322",
```

```
"V    c #070B5F",
"W    c #0B1298",
"X    c #0D16B8",
"Y    c #0E18CC",
"Z    c #1B2EF0",
"`    c #5687FF",
" .   c #446ADE",
"..   c #0D17C6",
"+.   c #1524E0",
"@.   c #243CFF",
"#.   c #3859FF",
"$.   c #92E1FF",
"%.   c #F6FCFF",
"&.   c #030324",
"*.   c #070B61",
"=.   c #0B1297",
"-.   c #0D16B9",
";.   c #101BD6",
">.   c #2237F3",
",.   c #1E31E7",
"'.   c #233AFB",
").   c #3B5FFF",
"!.   c #77B7FF",
"~.   c #97DFFF",
"{.   c #F1FBFF",
"].   c #03042C",
"^.   c #070C64",
"/.   c #0D17C4",
"(.   c #111ED9",
"_.   c #1B2EE8",
":.   c #253DF8",
"<.   c #3D62FF",
"[.   c #71AEFF",
"}.   c #9BDEFF",
"|.   c #040533",
"1.   c #070C67",
"2.   c #0B139E",
"3.   c #0D17C3",
"4.   c #1C2FE8",
"5.   c #3F65FF",
"6.   c #6CA8FF",
"7.   c #9DDCFF",
"8.   c #D9F5FF",
"9.   c #04063D",
"0.   c #080C6B",
"a.   c #0B13A0",
"b.   c #304CDB",
"c.   c #69A2FF",
"d.   c #9DDBFF",
"e.   c #D0F2FF",
"f.   c #060952",
"g.   c #080E77",
"h.   c #476EC3",
"i.   c #9BD9FF",
"j.   c #CBF1FF",
"k.   c #5A7AB0",
"                                                ",
"                                                ",
"                                                ",
"                                                ",
"                                                ",
"                                                ",
"                                                ",
"    . + @ # $ % & *           = - ; > , ' ) !   ",
```

```
"    ~ { ] ^ / ( _ : <       [ } | 1 / 2 3 4 5   ",
"    6 7 8 9 0 a b _ c d   = e f g h i j k l m   ",
"      n { o p 0 q r s t u v w x y z A B C D     ",
"        E F G H I J K L M N O P z Q R S T       ",
"          U V W X Y Z `  ...+.@.#._ $.%.        ",
"            &.*.=.-.;.>.,.+.'.).!.~.{.          ",
"              ].^.W /.(._.:.<.[.}.*             ",
"                |.1.2.3.4.5.6.7.8.              ",
"                  9.0.a.b.c.d.e.                ",
"                    f.g.h.i.j.                  ",
"                      *.k.                      ",
"                                                ",
"                                                ",
"                                                ",
"                                                ",
"                                                "};

/* XPM */
static char * DPW_xpm[] = {
"24 24 2 1",
"     c None",
".    c #000000",
"                        ",
"           ...          ",
"           ...          ",
"           ...          ",
"        .  ...  .       ",
"       ..  ...  ..      ",
"      ...  ...  ...     ",
"     ....  ...  ....    ",
"     ...   ...   ...    ",
"    ...    ...    ...   ",
"    ..     ...     ..   ",
"   ...     ...     ...  ",
"   ...     ...     ...  ",
"   ...     ...     ...  ",
"   ...             ...  ",
"    ...           ....  ",
"    ...           ...   ",
"    ....         ....   ",
"     ....       ....    ",
"      .............     ",
"       ...........      ",
"         .......        ",
"                        ",
"                        "};
/* XPM */
static char * OFF_xpm[] = {
"24 24 56 1",
"     c None",
".    c #990505",
"+    c #FF7A7A",
"@    c #BD0606",
"#    c #8C0404",
"$    c #7E0404",
"%    c #800404",
"&    c #320101",
"*    c #CC0707",
"=    c #E20707",
"-    c #FF0E0E",
";    c #FF0D0D",
">    c #FF0B0B",
",    c #520202",
"'    c #A60505",
```

```
")    c #F80909",
"!    c #240101",
"~    c #BA0606",
"{    c #180000",
"]    c #970505",
"^    c #F20707",
"/    c #F40909",
"(    c #D80707",
"_    c #150000",
":    c #EA0707",
"<    c #CE0707",
"[    c #9F0505",
"}    c #140000",
"|    c #A10404",
"1    c #6C0303",
"2    c #500202",
"3    c #860303",
"4    c #D70606",
"5    c #E80707",
"6    c #360101",
"7    c #000000",
"8    c #1B0000",
"9    c #BC0505",
"0    c #F00707",
"a    c #EC0707",
"b    c #E50707",
"c    c #E00707",
"d    c #C90707",
"e    c #C10707",
"f    c #BF0606",
"g    c #B80606",
"h    c #9A0505",
"i    c #FF0909",
"j    c #D30707",
"k    c #A10505",
"l    c #880404",
"m    c #230101",
"n    c #750404",
"o    c #0A0000",
"p    c #040000",
"q    c #030000",
".+@#$$$$$$$$$$$$$$$$$$%&",
"+++++++++++++++++++++++*",
"=+-;;;;;;;;;;;;;;;;;;;>,",
"'+>>>>>>>>>>>>>>>>>>>>)!",
".+>)))))))))))))))))))~{",
"]+>)^^^^^^^^^^^^^^^^/('_",
"]+>)^^^^^^^^^^^^^^^^:<[}",
"]+>)^^^^^^|12234^^^^5<[}",
"]+>)^^^^467|^488|^^^5<[}",
"]+>)^^^^171^^^4784^^5<[}",
"]+>)^^^^779^^^^873^^5<[}",
"]+>)^^^|77^^^^^272^^5<[}",
"]+>)^^^|77^^^^^272^^5<[}",
"]+>)^^^|77^^^^^272^^5<[}",
"]+>)^^^^779^^^^673^^5<[}",
"]+>)^^^^173^^^4784^^5<[}",
"]+>)^^^^467|^468|^^^5<[}",
"]+>)^^^^^^|22234^^^^5<[}",
"]+>)^^^^^^^^^^^^^^^^5<[}",
"]+>)^^0^^^^^^^^^^^^05*[}",
"]+>)^abbbbbbbbbbbbbbcd[}",
"]+>)=deffffffffffffffgh}",
"]+ijk#lllllllllllllll#l}",
```

```
"mn!opqqqqqqqqqqqqqqqqqp7"};
/* XPM */
static char * ON_xpm[] = {
"24 24 84 1",
" 	c None",
".	c #23A32C",
"+	c #2ED63A",
"@	c #27B531",
"#	c #229C2A",
"$	c #209428",
"%	c #209528",
"&	c #145D19",
"*	c #35F342",
"=	c #3EFF4D",
"-	c #3CFF4B",
";	c #3AFF48",
">	c #39FF48",
",	c #39FF47",
"'	c #29BC33",
")	c #2BC636",
"!	c #37FE45",
"~	c #38FF46",
"{	c #38FF45",
"]	c #1A7720",
"^	c #25AA2E",
"/	c #33EC40",
"(	c #35F643",
"_	c #35F542",
":	c #34F241",
"<	c #34EF41",
"[	c #32E73F",
"}	c #2DCF38",
"|	c #114F15",
"1	c #32E93F",
"2	c #33ED40",
"3	c #33EA3F",
"4	c #31E43E",
"5	c #31E13D",
"6	c #30DF3C",
"7	c #30DC3B",
"8	c #27B431",
"9	c #0E4111",
"0	c #23A22C",
"a	c #32E53E",
"b	c #2ED439",
"c	c #2DD239",
"d	c #2DCE38",
"e	c #2AC234",
"f	c #0D3D10",
"g	c #33EB3F",
"h	c #2FDA3B",
"i	c #2DD138",
"j	c #2CCD37",
"k	c #2CCA36",
"l	c #29BD33",
"m	c #24A62D",
"n	c #0D3C10",
"o	c #2BC936",
"p	c #18721E",
"q	c #0E4412",
"r	c #229F2A",
"s	c #000000",
"t	c #27B630",
"u	c #1D8824",
```

```
"v	c #2FD93B",
"w	c #2CCC37",
"x	c #31E23D",
"y	c #2ED53A",
"z	c #2CCB37",
"A	c #2BC736",
"B	c #2BC535",
"C	c #28BB32",
"D	c #28B732",
"E	c #27B631",
"F	c #27B330",
"G	c #23A42C",
"H	c #2FDB3B",
"I	c #29BF34",
"J	c #24A72D",
"K	c #219A29",
"L	c #0C3B10",
"M	c #114E15",
"N	c #1F8E26",
"O	c #092B0B",
"P	c #061C07",
"Q	c #051A07",
"R	c #051906",
"S	c #010902",
".+@#$$$$$$$$$$$$$$$$$$%&",
"*=-;>>>>>>>>>>>>>>>>>>,'",
")!;,~{{{{{{{{{{{{{{{{!*]",
"^/(_*:::::::::::::::<[}|",
".123456666666666666667}89",
"01/a7bccccccccccccccde^f",
"01g4hijjjjjjjjjjjjjjklmn",
"01g4h}jjjjjjjjjjjjjjolmn",
"01g4h}jjjjpqqqrjjjjjolmn",
"01g4h}jjjjjpsstjjjjjolmn",
"01g4h}jjjjjussjjjjjjolmn",
"01g4h}jjjjjussjjjjjjolmn",
"01g4h}jjjjjussjjjjjjolmn",
"01g4h}jjjjjussjjjjjjolmn",
"01g4h}jjjjjussjjjjjjolmn",
"01g4h}jjjjjussjjjjjjolmn",
"01g4h}jjjjjussjjjjjjolmn",
"01g4h}jjjjjpsstjjjjjolmn",
"01g4h}jjjjpqqqrjjjjjolmn",
"01g4v}wjjjjjjjjjjjjwo'mn",
"01gxyzAAAAAAAAAAAAAABCmn",
"011h)CDEEEEEEEEEEEEEEFGn",
"0aHIJ#KKKKKKKKKKKKKKK#KL",
"MN|OPQRRRRRRRRRRRRRRRQPS"};
/* XPM */
static char * SL_xpm[] = {
"21 22 72 1",
" 	c None",
".	c #FFFFFF",
"+	c #88CEFF",
"@	c #3A5CFF",
"#	c #92E1FF",
"$	c #7CC9FF",
"%	c #3753FF",
"&	c #E5EEFF",
"*	c #7CBFFF",
"=	c #1B2EFF",
"-	c #2C46FF",
";	c #4B75FF",
">	c #2D45FF",
```

```
",     c #2A45FF",
"'     c #2239FF",
")     c #2133FF",
"!     c #2032FF",
"~     c #080D79",
"{     c #1D2EFF",
"]     c #1727FF",
"^     c #1622FF",
"/     c #121DEE",
"(     c #030636",
"_     c #1726FF",
":     c #1421FF",
"<     c #101BED",
"[     c #0E15B6",
"}     c #020428",
"|     c #111DFF",
"1     c #0F19DF",
"2     c #0C14A9",
"3     c #010427",
"4     c #101CF7",
"5     c #0E19D8",
"6     c #0C14A5",
"7     c #74B6FF",
"8     c #83D0FF",
"9     c #77B9FF",
"0     c #101BF0",
"a     c #0E18D7",
"b     c #253DFF",
"c     c #253AFF",
"d     c #1828FF",
"e     c #1624FF",
"f     c #1321FF",
"g     c #131EFF",
"h     c #101CF5",
"i     c #101BF1",
"j     c #020324",
"k     c #070B61",
"l     c #0B139D",
"m     c #0D16BE",
"n     c #0E19D1",
"o     c #0F19DD",
"p     c #0F1AE1",
"q     c #0F1AE5",
"r     c #0F1AE6",
"s     c #02021C",
"t     c #070B5F",
"u     c #0E18D6",
"v     c #0F19DA",
"w     c #0B13A3",
"x     c #010426",
"y     c #0D15B5",
"z     c #091086",
"A     c #010324",
"B     c #080D78",
"C     c #030531",
"D     c #010111",
"E     c #070B5C",
"F     c #030635",
"G     c #00010F",
"                      ",
"                      ",
"           .          ",
"          .+@         ",
"         .#$% &.. &.. ",
```

```
"       .#*== -;> -;> ",
"      .#*,'= )!~ )!~ ",
"     .#*,'{] ^/( ^/( ",
"    .#*,'{_: <[} <[} ",
"   .#*,'{_:| 123 123 ",
"  .#*,'{_:|4 563 563 ",
" 789,'{_:|40 a63 a63 ",
"  bcdefghi<0 a63 a63 ",
"   jklmnopqr a63 a63 ",
"    stlmnopq u63 u63 ",
"     stlmnop u63 u63 ",
"      stlmnv wwx wwx ",
"       stlmy wzA wzA ",
"        stlB (CD (CD ",
"         sEF         ",
"          G          ",
"                     "};
/* XPM */
static char * SR_xpm[] = {
"21 22 72 1",
"     c None",
".    c #FFFFFF",
"+    c #3A5CFF",
"@    c #88CEFF",
"#    c #E5EEFF",
"$    c #3753FF",
"%    c #7CC9FF",
"&    c #92E1FF",
"*    c #2D45FF",
"=    c #4B75FF",
"-    c #2C46FF",
";    c #1B2EFF",
">    c #7CBFFF",
",    c #080D79",
"'    c #2032FF",
")    c #2133FF",
"!    c #2239FF",
"~    c #2A45FF",
"{    c #030636",
"]    c #121DEE",
"^    c #1622FF",
"/    c #1727FF",
"(    c #1D2EFF",
"_    c #020428",
":    c #0E15B6",
"<    c #101BED",
"[    c #1421FF",
"}    c #1726FF",
"|    c #010427",
"1    c #0C14A9",
"2    c #0F19DF",
"3    c #111DFF",
"4    c #0C14A5",
"5    c #0E19D8",
"6    c #101CF7",
"7    c #0E18D7",
"8    c #101BF0",
"9    c #77B9FF",
"0    c #83D0FF",
"a    c #74B6FF",
"b    c #101BF1",
"c    c #101CF5",
"d    c #131EFF",
"e    c #1321FF",
```

```
"f    c #1624FF",
"g    c #1828FF",
"h    c #253AFF",
"i    c #253DFF",
"j    c #0F1AE6",
"k    c #0F1AE5",
"l    c #0F1AE1",
"m    c #0F19DD",
"n    c #0E19D1",
"o    c #0D16BE",
"p    c #0B139D",
"q    c #070B61",
"r    c #020324",
"s    c #0E18D6",
"t    c #070B5F",
"u    c #02021C",
"v    c #010426",
"w    c #0B13A3",
"x    c #0F19DA",
"y    c #010324",
"z    c #091086",
"A    c #0D15B5",
"B    c #010111",
"C    c #030531",
"D    c #080D78",
"E    c #030635",
"F    c #070B5C",
"G    c #00010F",
"                     ",
"                     ",
"          .          ",
"         +@.         ",
" ..# ..# $%&.        ",
" *=- *=- ;;>&.       ",
" ,') ,') ;!~>&.      ",
" {]^ {]^ /(!~>&.     ",
" _:< _:< []}(!~>&.   ",
" |12 |12 3[]}(!~>&.  ",
" |45 |45 63[]}(!~>&. ",
" |47 |47 863[]}(!~90a",
" |47 |47 8<bcdefghi  ",
" |47 |47 jklmnopqr   ",
" |4s |4s klmnoptu    ",
" |4s |4s lmnoptu     ",
" vww vww xnoptu      ",
" yzw yzw Aoptu       ",
" BC{ BC{ Dptu        ",
"         EFu         ",
"          G          ",
"                     "};
/* XPM */
static char * STOP_xpm[] = {
"21 22 96 2",
"     c None",
".    c #FFFFFF",
"+    c #456FFF",
"@    c #5284FF",
"#    c #4F7EFF",
"$    c #4974FF",
"%    c #4873FF",
"&    c #4771FF",
"*    c #1626FF",
"=    c #426AFF",
"-    c #4874FF",
```

```
";     c #446DFF",
">     c #334EFF",
",     c #1524FF",
"'     c #0E19E8",
")     c #395BFF",
"!     c #3E63FF",
"~     c #2E47FF",
"{     c #2C46FF",
"]     c #2C45FF",
"^     c #2B44FF",
"/     c #1424FF",
"(     c #1421FF",
"_     c #101BF4",
":     c #070E7F",
"<     c #3759FF",
"[     c #395CFF",
"}     c #2A40FF",
"|     c #273EFF",
"1     c #273CFF",
"2     c #263BFF",
"3     c #121FFF",
"4     c #0A12A5",
"5     c #060A62",
"6     c #2A42FF",
"7     c #283EFF",
"8     c #2539FF",
"9     c #2335FF",
"0     c #101BF8",
"a     c #101BF0",
"b     c #0E17D2",
"c     c #091093",
"d     c #030638",
"e     c #2439FF",
"f     c #101BF6",
"g     c #0E19EE",
"h     c #0D18D7",
"i     c #0B15BC",
"j     c #08108C",
"k     c #0D18D5",
"l     c #0B15B6",
"m     c #0D18DF",
"n     c #0C16CB",
"o     c #0B14B5",
"p     c #000110",
"q     c #101DFF",
"r     c #0C16D4",
"s     c #050957",
"t     c #3657FF",
"u     c #090F8A",
"v     c #2940FF",
"w     c #1220FF",
"x     c #0E19EC",
"y     c #0C16D2",
"z     c #09119B",
"A     c #090F88",
"B     c #1422FF",
"C     c #0C16D0",
"D     c #0C16C8",
"E     c #091199",
"F     c #091195",
"G     c #070C6E",
"H     c #0D17CB",
"I     c #0D17D0",
"J     c #0C16C6",
```

```
"K    c #0A13B0",
"L    c #0A12AA",
"M    c #0A12A7",
"N    c #060C68",
"O    c #060B65",
"P    c #050955",
"Q    c #00010F",
"R    c #080F97",
"S    c #0D17C9",
"T    c #0B15B8",
"U    c #090F8C",
"V    c #050A58",
"W    c #04094C",
"X    c #04084A",
"Y    c #00000E",
"Z    c #03063F",
"`    c #010427",
" .   c #000117",
"..   c #000111",
"+.   c #00010E",
"@.   c #000004",
"                                                ",
"                                                ",
"                                                ",
"      . . . . . . . . . . . . . . . +           ",
"      . @ # $ % % % % % % % % % % & *           ",
"      . = - + ; > > > > > > > > = , '           ",
"      . ) ! ~ { ] ] ] ] ] ^ , / ( _ :           ",
"      . < [ } | 1 2 3 3 3 3 3 ( _ 4 5           ",
"      . < 6 7 8 9 0 0 0 0 0 0 a b c d           ",
"      . < 6 | e f g g g g g g h i j d           ",
"      . < 6 | e _ g g g g g g k l j d           ",
"      . < 6 | e _ g g g g g m n o j p           ",
"      . < 6 | q _ g g g g r r n o s p           ",
"      t < 6 | q _ g g g r r r n u s p           ",
"      t v 6 w q _ x r r r r y z A s p           ",
"      q v B 3 q C D D D D D E F G s p           ",
"      H I ( q J K L M M M : : N O P Q           ",
"      R S T U V W X X X X X X X W X Y           ",
"      Z d `  ...Q Q Q Q Q +.Q +.p Q @.          ",
"                                                ",
"                                                ",
"                                                "};
/* XPM */
static char * UP_xpm[] = {
"24 24 142 2",
"     c None",
".    c #070B61",
"+    c #5A7AB0",
"@    c #060952",
"#    c #080E77",
"$    c #476EC3",
"%    c #9BD9FF",
"&    c #CBF1FF",
"*    c #04063D",
"=    c #080C6B",
"-    c #0B13A0",
";    c #304CDB",
">    c #69A2FF",
",    c #9DDBFF",
"'    c #D0F2FF",
")    c #040533",
"!    c #070C67",
"~    c #0B139E",
```

```
"{    c #0D17C3",
"]    c #1C2FE8",
"^    c #3F65FF",
"/    c #6CA8FF",
"(    c #9DDCFF",
"_    c #D9F5FF",
":    c #03042C",
"<    c #070C64",
"[    c #0B1298",
"}    c #0D17C4",
"|    c #111ED9",
"1    c #1B2EE8",
"2    c #253DF8",
"3    c #3D62FF",
"4    c #71AEFF",
"5    c #9BDEFF",
"6    c #E4F8FF",
"7    c #030324",
"8    c #0B1297",
"9    c #0D16B9",
"0    c #101BD6",
"a    c #2237F3",
"b    c #1E31E7",
"c    c #1524E0",
"d    c #233AFB",
"e    c #3B5FFF",
"f    c #77B7FF",
"g    c #97DFFF",
"h    c #F1FBFF",
"i    c #020322",
"j    c #070B5F",
"k    c #0D16B8",
"l    c #0E18CC",
"m    c #1B2EF0",
"n    c #5687FF",
"o    c #446ADE",
"p    c #0D17C6",
"q    c #243CFF",
"r    c #3859FF",
"s    c #7EC2FF",
"t    c #92E1FF",
"u    c #F6FCFF",
"v    c #030327",
"w    c #070B5C",
"x    c #0A1295",
"y    c #0D15B7",
"z    c #0E18CB",
"A    c #1B2EEF",
"B    c #5686FD",
"C    c #83C9FF",
"D    c #4D78D0",
"E    c #0C15AE",
"F    c #0E19C8",
"G    c #1524E1",
"H    c #263FFF",
"I    c #3A5EFF",
"J    c #7FC3FF",
"K    c #9DE2FF",
"L    c #EDFAFF",
"M    c #03042B",
"N    c #060A59",
"O    c #0A1191",
"P    c #0C15B6",
"Q    c #0E18CA",
```

```
"R    c #1A2DED",
"S    c #5382FC",
"T    c #80C5FF",
"U    c #BFEBFF",
"V    c #7682B8",
"W    c #091086",
"X    c #0C15AF",
"Y    c #0F19C9",
"Z    c #1626E2",
"`    c #3E64FF",
" .   c #7CBFFF",
"..   c #A6E3FF",
"+.   c #E6F8FF",
"@.   c #020427",
"#.   c #060A54",
"$.   c #0A118D",
"%.   c #0C15B4",
"&.   c #192BEB",
"*.   c #507DFC",
"=.   c #BDE9FF",
"-.   c #E3F6FF",
";.   c #080D70",
">.   c #0A1088",
",.   c #0C15B0",
"'.   c #0F19CA",
").   c #1727E5",
"!.   c #253EFF",
"~.   c #3D61FF",
"{.   c #79C0FF",
"].   c #A9DDFF",
"^.   c #DEF1FF",
"/.   c #03052E",
"(.   c #0A1192",
"_.   c #0E17BD",
":.   c #1829E8",
"<.   c #4A75FD",
"[.   c #B8E8FF",
"}.   c #E0F4FE",
"|.   c #080D71",
"1.   c #0A108B",
"2.   c #0C15B2",
"3.   c #0E19CA",
"4.   c #1B2EFD",
"5.   c #4F7EFF",
"6.   c #5F93FF",
"7.   c #CADCFF",
"8.   c #030635",
"9.   c #040846",
"0.   c #090E82",
"a.   c #0F18BB",
"b.   c #2239FF",
"c.   c #74B4FF",
"d.   c #8EDBFF",
"e.   c #0B14A2",
"f.   c #0D16C0",
"g.   c #0E19D1",
"h.   c #1A2CFB",
"i.   c #324DFF",
"j.   c #395AFF",
"k.   c #BDC9FF",
"                                                  ",
"                                                  ",
"                                                  ",
"                                                  ",
```

```
"                                                ",
"                      . +                       ",
"                    @ # $ % &                   ",
"                  * = - ; > , '                 ",
"                ) ! ~ { ] ^ / ( _               ",
"              : < [ } | 1 2 3 4 5 6             ",
"            7 . 8 9 0 a b c d e f g h           ",
"          i j [ k l m n o p c q r s t u         ",
"        v w x y z A B C D E F G H I J K L       ",
"      M N O P Q R S T U V W X Y Z H `  ...+.    ",
"    @.#.$.%.Q &.*.s =.-.  ;.>.,.'.).!.~.{.].^.  ",
"    /.N (._.:.<.s [.}.      |.1.2.3.:.4.5.6.7.  ",
"    8.9.0.a.b.c.d.6           ;.e.f.g.h.i.j.k.  ",
"                                                ",
"                                                ",
"                                                ",
"                                                ",
"                                                ",
"                                                ",
"                                                "};
```

interface.h

```
#ifndef _INTERFACE_H_
#define _INTERFACE_H_

#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <string.h>

#include <gdk/gdkkeysyms.h>
#include <gtk/gtk.h>

enum
{
   MSG00,
   MSG01,
   MSG02,
   MSG03,
   MSG04,
   MSG05,
   MSG06,
   MSG07,
   MSG08,
   MSG09,
   MSG10,
   MSG11,
   MSGFF
};

#define NUM_OF_BUTTONS 10

#define BUTTONS_ON  TRUE
#define BUTTONS_OFF FALSE

GtkWidget* create_mainwin (void);
GtkWidget* create_aboutwin (void);
GtkWidget* xpm_label_box ( GtkWidget*     parent,
                          char*           xpm_filename[]);
void buttonControl (char cmd);
void messageWrite (short msg);
```

```
#endif
```

server.h

```
#ifndef _SERVER_H_
#define _SERVER_H_

#include <sys/socket.h>       /*  socket definitions        */
#include <sys/types.h>        /*  socket types              */
#include <sys/stat.h>
#include <arpa/inet.h>        /*  inet (3) funtions         */
#include <netinet/in.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <pthread.h>
#include <time.h>

#define PORT_NUM (2003)
#define LISTENQ  (0)

/* Server API */
int tc_serverStart(void);
int tc_serverStop(void);
int tc_serverSend(short int command);

#endif
```

callbacks.c

```
#include "callbacks.h"
#include "interface.h"
#include "server.h"
#include "control.h"
// included to make work on PowerPC Mac
// These are Big Endian systems, while Intel are Little Endians systems
// Didn't work as expected
//#include </usr/include/sys/_endian.h>

int error = 0;
unsigned int contButtStatus = 0;
pthread_t contThread;
void* contThread_func(void* arg);

void
trolleyControl_Quit(void)
{
   tc_serverStop();
   gtk_main_quit();
}

void
send_message(short int message)
{

   /* Added to work on PPC based Mac which is Big Endian */
   #ifdef PPC
   short int temp;
   char *from, *to;
   from = (char *)&message;
   to = (char *)&temp;
   memcpy(to+1, from,    1);
   memcpy(to,   from+1,  1);
   //printf("message = %x, %d; temp = %x, %d\n", message, message, temp,
temp);
   message = temp;
   //printf("message = %x\n", message);
   #endif

   error = tc_serverSend(message);
}

void
about_button_activate                    (GtkMenuItem     *menuitem,
                                          gpointer         user_data)
{
}

void
start_server_activate                    (GtkMenuItem     *menuitem,
                                        gpointer         user_data)
{
    DEBUG1 ("Start Sever Activated\n");
    error = tc_serverStart();

    if(error == -100)
      messageWrite(MSG04);
    else if(error)
      messageWrite(MSG05);
```

```
    else
      messageWrite(MSG06);
}

void
stop_server_activate                    (GtkMenuItem     *menuitem,
                                        gpointer         user_data)
{
    DEBUG1 ("Stop Sever Activated\n");
    send_message(TRC_CLOSE_CONN);
    if(tc_serverStop() == -100)
      messageWrite(MSG07);
    else
      messageWrite(MSG08);
}


void
on_continuous_forward_button_pressed      (GtkButton       *button,
                                        gpointer         user_data)
{
   DEBUG1 ("\nContinous Forward Pressed\n");
   contButtStatus = TRC_CONT_FORWARD;
   pthread_create(&contThread, NULL, contThread_func, NULL);
}


void
on_continuous_forward_button_released     (GtkButton       *button,
                                        gpointer         user_data)
{
   DEBUG1 ("\nContinous Forward Released\n");
   contButtStatus = 0;
   on_stop_button_pressed(button, user_data);
}

void
on_continuous_backward_button_pressed      (GtkButton       *button,
                                         gpointer         user_data)
{
   DEBUG1 ("\nContinous Backward Pressed\n");
   contButtStatus = TRC_CONT_BACKWARD;
   pthread_create(&contThread, NULL, contThread_func, NULL);
}

void
on_continuous_backward_button_released     (GtkButton       *button,
                                         gpointer         user_data)
{
   DEBUG1 ("\nContinous Backward Released\n");
   contButtStatus = 0;
   on_stop_button_pressed(button, user_data);
}

void
on_single_step_backward_button_pressed      (GtkButton       *button,
                                          gpointer         user_data)
{
   DEBUG1 ("\nSingle Step Backward Pressed\n");
   send_message(TRC_STEP_BACKWARD);
}

void
on_single_step_backward_button_released     (GtkButton       *button,
```

```
                                          gpointer         user_data)
{
   DEBUG1 ("\nSingle Step Backward Released\n");
   on_stop_button_pressed(button, user_data);
}

void
on_single_step_forward_button_pressed      (GtkButton       *button,
                                          gpointer         user_data)
{
   DEBUG1 ("\nSingle Step Forward Pressed\n");
   send_message(TRC_STEP_FORWARD);
}

void
on_single_step_forward_button_released     (GtkButton       *button,
                                          gpointer         user_data)
{
   DEBUG1 ("\nSingle Step Forward Released\n");
   on_stop_button_pressed(button, user_data);
}

void
on_stop_button_pressed      (GtkButton       *button,
                             gpointer         user_data)
{
   DEBUG1 ("\nStop Pressed\n");
   send_message(TRC_STOP);
}

void
on_increase_speed_button_pressed      (GtkButton       *button,
                                       gpointer         user_data)
{
   DEBUG1 ("\nSpeed Increase Pressed\n");
   send_message(TRC_SPEED_UP);
}

void
on_increase_speed_button_released     (GtkButton       *button,
                                       gpointer         user_data)
{
   DEBUG1 ("\nSpeed Increase Released\n");
   on_stop_button_pressed(button, user_data);
}

void
on_decrease_speed_button_pressed      (GtkButton       *button,
                                       gpointer         user_data)
{
   DEBUG1 ("\nSpeed Decrease Pressed\n");
   send_message(TRC_SPEED_DOWN);
}


void
on_decrease_speed_button_released     (GtkButton       *button,
                                       gpointer         user_data)
{
   DEBUG1 ("\nSpeed Decrease Released\n");
   on_stop_button_pressed(button, user_data);
}

void
```

```
on_brake_on_button_pressed      (GtkButton       *button,
                                 gpointer         user_data)
{
   DEBUG1 ("\nBrake On Pressed\n");
   send_message(TRC_BRAKE);
}

void
on_brake_on_button_released     (GtkButton       *button,
                                 gpointer         user_data)
{
   DEBUG1 ("\nBrake On Released\n");
   on_stop_button_pressed(button, user_data);
}

void
on_brake_off_button_pressed      (GtkButton       *button,
                                  gpointer         user_data)
{
   DEBUG1 ("\nBrake Off Pressed\n");
   send_message(TRC_UNBRAKE);
}

void
on_brake_off_button_released     (GtkButton       *button,
                                  gpointer         user_data)
{
   DEBUG1 ("\nBrake Off Released\n");
   on_stop_button_pressed(button, user_data);
}


void
on_detecpw_button_toggle      (GtkButton       *button,
                                  gpointer         user_data)
{
   static char toggle = 0;

   if(toggle)
   {
      DEBUG1 ("\nDetector Power Untoggled\n");
      send_message(TRC_DETECTOR_POWER_OFF);
      toggle = 0;
   }
   else
   {
      DEBUG1 ("\nDetector Power Toggled\n");
      send_message(TRC_DETECTOR_POWER_ON);
      toggle = 1;
   }
}

void* contThread_func(void* arg)
{
    while(contButtStatus != 0)
    {
      send_message(contButtStatus);
      //printf("hello\n");
      usleep((unsigned int)250000);
    }
      send_message(TRC_STOP);
}
```

client.c

```
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdio.h>

#include "control.h"

#define IPADDRESS   "127.0.0.1"
#define PORT          (2003)

int main(void)
{
    struct sockaddr_in servAddr;
    short int command;
    int sock;
    int i;
    int cc;

    if((sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
          printf("Error openning socket.\n"), exit(0);

    memset(&servAddr, 0, sizeof(servAddr));
    servAddr.sin_family      = AF_INET;
    servAddr.sin_addr.s_addr = inet_addr(IPADDRESS);
    servAddr.sin_port        = htons(PORT);

    if(connect(sock, (struct sockaddr *)&servAddr, sizeof(servAddr)) <
0)
          printf("Error connecting to socket\n"), exit(0);


    command = TRC_STEP_FORWARD;
    if(send(sock, &command, sizeof(command), 0) < sizeof(command))
       printf("Error sending full packet size\n");

    while(cc = recv(sock, &command, sizeof(command), 0))
    {
          printf("command = %X\n", command);
            if(command == TRC_CLOSE_CONN)
               break;
    }

    close(sock);
    return 0;
}
```

control.c

```
char *MSG_A[] = {
   "$ LDBS Type\n",
   "$ Trolley Type\n"
};

char *MSG_B[] = {
   "$ Current duty cycle\n"
};

char *MSG_C[] = {
   "$ Step Forward\n",
   "$ Step Backward\n",
   "$ Continous Forward\n",
```

```
   "$ Continous Backward\n",
   "$ Engage Brakes\n",
   "$ Disengage Brakes\n",
   "$ Speed Up\n",
   "$ Slow Down\n",
   "$ Trolley Stop\n",
   "$ Enable Detector Power\n",
   "$ Brakes are on\n",
   "$ Brakes are off\n",
   "$ Detector Power Enabled\n",
};

char *MSG_E[] = {
   "$ Network OK\n",
   "$ Network Failed\n",
   "$ DHCP FB\n",
   "$ DHCP NFB\n",
   "$ Brake Failure\n",
   "$ Motor Failure\n"
};
```

interface.c

```
#include "interface.h"
#include "control.h"
#include "callbacks.h"
#include "icons.h"

char* msgArray[] =
{
"$ Incorrect Command.\n",
"$ Initiate TCP/IP Server.\n",
"$ Server Not Running.\n",
"$ Send TCP/IP Data Failed.\n",
"$ Server Already Running.\n",
"$ Server Start Failed.\n",
"$ Server Started.\n",
"$ Server Already Stopped.\n",
"$ Server Stopped.\n",
"$ Accepted Connection.\n",
"$ TCP/IP Disconnected.\n",
"$ Unable to shutdown list_s.\n"
};


GtkWidget *all_buttons[NUM_OF_BUTTONS];
//GtkWidget *message_window;

GtkWidget*
create_mainwin (void)
{
  GtkWidget *mainwin;
  GtkWidget *aboutwin;
  GtkWidget *main_vert_boxes;
  GtkWidget *menubar;
  guint tmp_key;
  GtkWidget *file_menu_button;
  GtkWidget *file_menu_button_menu;
  GtkAccelGroup *file_menu_button_menu_accels;
  GtkWidget *quit1;
  GtkWidget *server_menu_button;
  GtkWidget *server_menu_button_menu;
  GtkAccelGroup *server_menu_button_menu_accels;
```

```
  GtkWidget *start_server_button;
  GtkWidget *stop_server_button;
  GtkWidget *help1;
  GtkWidget *help1_menu;
  GtkAccelGroup *help1_menu_accels;
  GtkWidget *about_button;
  GtkWidget *lable_horz_boxes;
  GtkWidget *lable_spacer1;
  GtkWidget *speed_label;
  GtkWidget *lable_spacer3;
  GtkWidget *brake_label;
  GtkWidget *lable_spacer5;
  GtkWidget *speed_brake_horz_boxes;
  GtkWidget *button_spacer1;
  GtkWidget *speed_vert_boxes;
  GtkWidget *increase_speed_button;
  GtkWidget *increase_speed_box;
  GtkWidget *decrease_speed_button;
  GtkWidget *decrease_speed_box;
  GtkWidget *button_spacer3;
  GtkWidget *brake_vert_boxes;
  GtkWidget *brake_on_button;
  GtkWidget *brake_on_box;
  GtkWidget *brake_off_button;
  GtkWidget *brake_off_box;
  GtkWidget *button_spacer5;
  GtkWidget *horz_separator;
  GtkWidget *trolley_controls_label;
  GtkWidget *control_horz_boxes;
  GtkWidget *continuous_forward_button;
  GtkWidget *continuous_forward_box;
  GtkWidget *single_step_forward_button;
  GtkWidget *single_step_forward_box;
  GtkWidget *stop_button;
  GtkWidget *stop_box;
  GtkWidget *single_step_backward_button;
  GtkWidget *single_step_backward_box;
  GtkWidget *continuous_backward_button;
  GtkWidget *continuous_backward_box;
/*
  GtkWidget *message_window_horz_boxes;
  GtkWidget *message_window_label;
  GtkWidget *message_window_vscrollbar;
*/
  GtkWidget *detecpw_horz_boxes;
  GtkWidget *detecpw_label;
  GtkWidget *detecpw_button;
  GtkWidget *detecpw_box;
  GtkAccelGroup *accel_group;
  GtkTooltips *tooltips;
  int i;


  tooltips = gtk_tooltips_new ();

  accel_group = gtk_accel_group_new ();

  mainwin = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  aboutwin = create_aboutwin();

  gtk_object_set_data (GTK_OBJECT (mainwin), "mainwin", mainwin);
  gtk_window_set_title (GTK_WINDOW (mainwin), "Trolley Control");
  gtk_window_set_policy (GTK_WINDOW (mainwin), FALSE, FALSE, FALSE);
```

```
  main_vert_boxes = gtk_vbox_new (FALSE, 0);
  gtk_widget_ref (main_vert_boxes);
  gtk_object_set_data_full (GTK_OBJECT (mainwin),
                            "main_vert_boxes",
                            main_vert_boxes,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (main_vert_boxes);
  gtk_container_add (GTK_CONTAINER (mainwin), main_vert_boxes);

  menubar = gtk_menu_bar_new ();
  gtk_widget_ref (menubar);
  gtk_object_set_data_full (GTK_OBJECT (mainwin), "menubar", menubar,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (menubar);
  gtk_box_pack_start (GTK_BOX (main_vert_boxes), menubar, FALSE, FALSE,
0);
  gtk_menu_bar_set_shadow_type (GTK_MENU_BAR (menubar),
GTK_SHADOW_ETCHED_IN);

  file_menu_button = gtk_menu_item_new_with_label ("");
  tmp_key = gtk_label_parse_uline (GTK_LABEL (GTK_BIN
(file_menu_button)->child),
                                   "_File");
  gtk_widget_add_accelerator (file_menu_button, "activate_item",
accel_group,
                              tmp_key, GDK_MOD1_MASK, (GtkAccelFlags)
0);
  gtk_widget_ref (file_menu_button);
  gtk_object_set_data_full (GTK_OBJECT (mainwin), "file_menu_button",
file_menu_button,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (file_menu_button);
  gtk_container_add (GTK_CONTAINER (menubar), file_menu_button);

  file_menu_button_menu = gtk_menu_new ();
  gtk_widget_ref (file_menu_button_menu);
  gtk_object_set_data_full (GTK_OBJECT (mainwin),
"file_menu_button_menu", file_menu_button_menu,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_menu_item_set_submenu (GTK_MENU_ITEM (file_menu_button),
file_menu_button_menu);
  file_menu_button_menu_accels = gtk_menu_ensure_uline_accel_group
(GTK_MENU (file_menu_button_menu));

  quit1 = gtk_menu_item_new_with_label ("");
  tmp_key = gtk_label_parse_uline (GTK_LABEL (GTK_BIN (quit1)->child),
                                   "_Quit");
  gtk_widget_add_accelerator (quit1, "activate_item",
file_menu_button_menu_accels,
                              tmp_key, 0, 0);
  gtk_widget_ref (quit1);
  gtk_object_set_data_full (GTK_OBJECT (mainwin), "quit1", quit1,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (quit1);
  gtk_container_add (GTK_CONTAINER (file_menu_button_menu), quit1);
  gtk_widget_add_accelerator (quit1, "activate", accel_group,
                              GDK_q, 0,
                              GTK_ACCEL_VISIBLE);

  server_menu_button = gtk_menu_item_new_with_label ("");
  tmp_key = gtk_label_parse_uline (GTK_LABEL (GTK_BIN
(server_menu_button)->child),
                                   "_Server");
```

```
  gtk_widget_add_accelerator (server_menu_button, "activate_item",
accel_group,
                              tmp_key, GDK_MOD1_MASK, (GtkAccelFlags)
0);
  gtk_widget_ref (server_menu_button);
  gtk_object_set_data_full (GTK_OBJECT (mainwin), "server_menu_button",
server_menu_button,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (server_menu_button);
  gtk_container_add (GTK_CONTAINER (menubar), server_menu_button);

  server_menu_button_menu = gtk_menu_new ();
  gtk_widget_ref (server_menu_button_menu);
  gtk_object_set_data_full (GTK_OBJECT (mainwin),
"server_menu_button_menu", server_menu_button_menu,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_menu_item_set_submenu (GTK_MENU_ITEM (server_menu_button),
server_menu_button_menu);
  server_menu_button_menu_accels = gtk_menu_ensure_uline_accel_group
(GTK_MENU (server_menu_button_menu));

  start_server_button = gtk_menu_item_new_with_label ("");
  tmp_key = gtk_label_parse_uline (GTK_LABEL (GTK_BIN
(start_server_button)->child),
                                   "Star_t");
  gtk_widget_add_accelerator (start_server_button, "activate_item",
server_menu_button_menu_accels,
                              tmp_key, 0, 0);
  gtk_widget_ref (start_server_button);
  gtk_object_set_data_full (GTK_OBJECT (mainwin), "start_server_button",
start_server_button,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (start_server_button);
  gtk_container_add (GTK_CONTAINER (server_menu_button_menu),
start_server_button);

  stop_server_button = gtk_menu_item_new_with_label ("");
  tmp_key = gtk_label_parse_uline (GTK_LABEL (GTK_BIN
(stop_server_button)->child),
                                   "Sto_p");
  gtk_widget_add_accelerator (stop_server_button, "activate_item",
server_menu_button_menu_accels,
                              tmp_key, 0, 0);
  gtk_widget_ref (stop_server_button);
  gtk_object_set_data_full (GTK_OBJECT (mainwin), "stop_server_button",
stop_server_button,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (stop_server_button);
  gtk_container_add (GTK_CONTAINER (server_menu_button_menu),
stop_server_button);

  help1 = gtk_menu_item_new_with_label ("");
  tmp_key = gtk_label_parse_uline (GTK_LABEL (GTK_BIN (help1)->child),
                                   "_Help");
  gtk_widget_add_accelerator (help1, "activate_item", accel_group,
                              tmp_key, GDK_MOD1_MASK, (GtkAccelFlags)
0);
  gtk_widget_ref (help1);
  gtk_object_set_data_full (GTK_OBJECT (mainwin), "help1", help1,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (help1);
  gtk_container_add (GTK_CONTAINER (menubar), help1);

  help1_menu = gtk_menu_new ();
```

```
  gtk_widget_ref (help1_menu);
  gtk_object_set_data_full (GTK_OBJECT (mainwin), "help1_menu",
help1_menu,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_menu_item_set_submenu (GTK_MENU_ITEM (help1), help1_menu);
  help1_menu_accels = gtk_menu_ensure_uline_accel_group (GTK_MENU
(help1_menu));

  about_button = gtk_menu_item_new_with_label ("");
  tmp_key = gtk_label_parse_uline (GTK_LABEL (GTK_BIN (about_button)-
>child),
                                   "_About");
  gtk_widget_add_accelerator (about_button, "activate_item",
help1_menu_accels,
                              tmp_key, 0, 0);
  gtk_widget_ref (about_button);
  gtk_object_set_data_full (GTK_OBJECT (mainwin), "about_button",
about_button,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (about_button);
  gtk_container_add (GTK_CONTAINER (help1_menu), about_button);

  lable_horz_boxes = gtk_hbox_new (FALSE, 0);
  gtk_widget_ref (lable_horz_boxes);
  gtk_object_set_data_full (GTK_OBJECT (mainwin), "lable_horz_boxes",
lable_horz_boxes,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (lable_horz_boxes);
  gtk_box_pack_start (GTK_BOX (main_vert_boxes), lable_horz_boxes,
FALSE, FALSE, 0);
  gtk_widget_set_usize (lable_horz_boxes, -2, 20);

  lable_spacer1 = gtk_label_new ("");
  gtk_widget_ref (lable_spacer1);
  gtk_object_set_data_full (GTK_OBJECT (mainwin), "lable_spacer1",
lable_spacer1,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (lable_spacer1);
  gtk_box_pack_start (GTK_BOX (lable_horz_boxes), lable_spacer1, TRUE,
TRUE, 0);
  gtk_widget_set_usize (lable_spacer1, 35, -2);

  speed_label = gtk_label_new ("SPEED");
  gtk_widget_ref (speed_label);
  gtk_object_set_data_full (GTK_OBJECT (mainwin), "speed_label",
speed_label,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (speed_label);
  gtk_box_pack_start (GTK_BOX (lable_horz_boxes), speed_label, FALSE,
FALSE, 0);
  gtk_widget_set_usize (speed_label, 50, 20);

  lable_spacer3 = gtk_label_new ("");
  gtk_widget_ref (lable_spacer3);
  gtk_object_set_data_full (GTK_OBJECT (mainwin), "lable_spacer3",
lable_spacer3,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (lable_spacer3);
  gtk_box_pack_start (GTK_BOX (lable_horz_boxes), lable_spacer3, TRUE,
TRUE, 0);
  gtk_widget_set_usize (lable_spacer3, 20, -2);

  brake_label = gtk_label_new ("BRAKE");
  gtk_widget_ref (brake_label);
```

```
  gtk_object_set_data_full (GTK_OBJECT (mainwin), "brake_label",
brake_label,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (brake_label);
  gtk_box_pack_start (GTK_BOX (lable_horz_boxes), brake_label, FALSE,
FALSE, 0);
  gtk_widget_set_usize (brake_label, 50, 20);

  lable_spacer5 = gtk_label_new ("");
  gtk_widget_ref (lable_spacer5);
  gtk_object_set_data_full (GTK_OBJECT (mainwin), "lable_spacer5",
lable_spacer5,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (lable_spacer5);
  gtk_box_pack_start (GTK_BOX (lable_horz_boxes), lable_spacer5, TRUE,
TRUE, 0);
  gtk_widget_set_usize (lable_spacer5, 35, -2);

  speed_brake_horz_boxes = gtk_hbox_new (FALSE, 0);
  gtk_widget_ref (speed_brake_horz_boxes);
  gtk_object_set_data_full (GTK_OBJECT (mainwin),
"speed_brake_horz_boxes", speed_brake_horz_boxes,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (speed_brake_horz_boxes);
  gtk_box_pack_start (GTK_BOX (main_vert_boxes), speed_brake_horz_boxes,
TRUE, TRUE, 1);
  gtk_widget_set_usize (speed_brake_horz_boxes, 190, 68);

  button_spacer1 = gtk_label_new ("");
  gtk_widget_ref (button_spacer1);
  gtk_object_set_data_full (GTK_OBJECT (mainwin), "button_spacer1",
button_spacer1,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (button_spacer1);
  gtk_box_pack_start (GTK_BOX (speed_brake_horz_boxes), button_spacer1,
TRUE, TRUE, 0);
  gtk_widget_set_usize (button_spacer1, 35, 68);

  speed_vert_boxes = gtk_vbox_new (FALSE, 0);
  gtk_widget_ref (speed_vert_boxes);
  gtk_object_set_data_full (GTK_OBJECT (mainwin), "speed_vert_boxes",
speed_vert_boxes,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (speed_vert_boxes);
  gtk_box_pack_start (GTK_BOX (speed_brake_horz_boxes),
speed_vert_boxes, FALSE, FALSE, 10);
  gtk_widget_set_usize (speed_vert_boxes, 30, -2);
  gtk_widget_realize(mainwin);

// ------------------- Increase Speed Button ---------------------------

  increase_speed_button = gtk_button_new();
  gtk_widget_ref (increase_speed_button);
  gtk_object_set_data_full (GTK_OBJECT (mainwin),
"increase_speed_button", increase_speed_button,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_box_pack_start (GTK_BOX (speed_vert_boxes), increase_speed_button,
FALSE, FALSE, 2);
  gtk_widget_set_usize (increase_speed_button, 30, 30);
  gtk_tooltips_set_tip (tooltips, increase_speed_button, "Increase
Speed", NULL);
  increase_speed_box = xpm_label_box(mainwin,
                                      UP_xpm);
                                      //"pixmaps/UP.xpm");
```

```
  gtk_container_add(GTK_CONTAINER(increase_speed_button),
                    increase_speed_box);
  gtk_widget_show (increase_speed_box);
  gtk_widget_show (increase_speed_button);

// ------------------- Decrease Speed Button --------------------------

  decrease_speed_button = gtk_button_new();
  gtk_widget_ref (decrease_speed_button);
  gtk_object_set_data_full (GTK_OBJECT (mainwin),
"decrease_speed_button", decrease_speed_button,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (decrease_speed_button);
  gtk_box_pack_start (GTK_BOX (speed_vert_boxes), decrease_speed_button,
FALSE, FALSE, 2);
  gtk_widget_set_usize (decrease_speed_button, 30, 30);
  gtk_tooltips_set_tip (tooltips, decrease_speed_button, "Decrease
Speed", NULL);
  decrease_speed_box = xpm_label_box(mainwin,
                                      DN_xpm);
  gtk_container_add(GTK_CONTAINER(decrease_speed_button),
                    decrease_speed_box);
  gtk_widget_show (decrease_speed_box);
  gtk_widget_show (decrease_speed_button);

// -------------------  Spacers  --------------------------

  button_spacer3 = gtk_label_new ("");
  gtk_widget_ref (button_spacer3);
  gtk_object_set_data_full (GTK_OBJECT (mainwin), "button_spacer3",
button_spacer3,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (button_spacer3);
  gtk_box_pack_start (GTK_BOX (speed_brake_horz_boxes), button_spacer3,
TRUE, TRUE, 0);
  gtk_widget_set_usize (button_spacer3, 20, 68);

  brake_vert_boxes = gtk_vbox_new (FALSE, 0);
  gtk_widget_ref (brake_vert_boxes);
  gtk_object_set_data_full (GTK_OBJECT (mainwin), "brake_vert_boxes",
brake_vert_boxes,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (brake_vert_boxes);
  gtk_box_pack_start (GTK_BOX (speed_brake_horz_boxes),
brake_vert_boxes, FALSE, FALSE, 10);
  gtk_widget_set_usize (brake_vert_boxes, 30, -2);

// -------------------- Brake Up Button ----------------------------

  brake_on_button = gtk_button_new();

  gtk_widget_ref (brake_on_button);

  gtk_object_set_data_full (GTK_OBJECT (mainwin),
                            "brake_on_button",
                      brake_on_button,
                            (GtkDestroyNotify) gtk_widget_unref);

  gtk_box_pack_start (GTK_BOX (brake_vert_boxes),
                  brake_on_button,
                  FALSE,
                  FALSE,
                  2);
```

```
  gtk_widget_set_usize (brake_on_button, 30, 30);

  gtk_tooltips_set_tip (tooltips, brake_on_button, "Turn Brakes On",
NULL);

  brake_on_box = xpm_label_box(mainwin, ON_xpm);

  gtk_container_add(GTK_CONTAINER(brake_on_button), brake_on_box);

  gtk_widget_show (brake_on_box);

  gtk_widget_show (brake_on_button);

// ------------------- End Brake Up Button ---------------------------

// -------------------- Brake Down Button ----------------------------

  brake_off_button = gtk_button_new();

  gtk_widget_ref (brake_off_button);

  gtk_object_set_data_full (GTK_OBJECT (mainwin),
                      "brake_off_button",
                      brake_off_button,
                            (GtkDestroyNotify) gtk_widget_unref);

  gtk_box_pack_start (GTK_BOX (brake_vert_boxes),
                  brake_off_button,
                  FALSE,
                  FALSE,
                  2);

  gtk_widget_set_usize (brake_off_button, 30, 30);

  gtk_tooltips_set_tip (tooltips, brake_off_button, "Turn Brakes Off",
NULL);

  brake_off_box = xpm_label_box(mainwin, OFF_xpm);

  gtk_container_add(GTK_CONTAINER(brake_off_button), brake_off_box);

  gtk_widget_show (brake_off_box);

  gtk_widget_show (brake_off_button);

// ------------------- End Brake Down Button --------------------------
-

  button_spacer5 = gtk_label_new ("");
  gtk_widget_ref (button_spacer5);
  gtk_object_set_data_full (GTK_OBJECT (mainwin), "button_spacer5",
button_spacer5,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (button_spacer5);
  gtk_box_pack_start (GTK_BOX (speed_brake_horz_boxes), button_spacer5,
TRUE, TRUE, 0);
  gtk_widget_set_usize (button_spacer5, 35, 68);

  horz_separator = gtk_hseparator_new ();
  gtk_widget_ref (horz_separator);
  gtk_object_set_data_full (GTK_OBJECT (mainwin), "horz_separator",
horz_separator,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (horz_separator);
```

```
  gtk_box_pack_start (GTK_BOX (main_vert_boxes), horz_separator, TRUE,
TRUE, 0);

  trolley_controls_label = gtk_label_new ("Trolley Movement");
  gtk_widget_ref (trolley_controls_label);
  gtk_object_set_data_full (GTK_OBJECT (mainwin),
"trolley_controls_label", trolley_controls_label,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (trolley_controls_label);
  gtk_box_pack_start (GTK_BOX (main_vert_boxes), trolley_controls_label,
FALSE, FALSE, 0);
  gtk_widget_set_usize (trolley_controls_label, -2, 18);

  control_horz_boxes = gtk_hbox_new (TRUE, 0);
  gtk_widget_ref (control_horz_boxes);
  gtk_object_set_data_full (GTK_OBJECT (mainwin), "control_horz_boxes",
control_horz_boxes,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (control_horz_boxes);
  gtk_box_pack_start (GTK_BOX (main_vert_boxes), control_horz_boxes,
FALSE, FALSE, 4);
  gtk_widget_set_usize (control_horz_boxes, 190, 30);


// -------------------- Continuous Forward Button ----------------------
-------

  continuous_forward_button = gtk_button_new();

  gtk_widget_ref (continuous_forward_button);

  gtk_object_set_data_full (GTK_OBJECT (mainwin),
                            "continuous_forward_button",
                            continuous_forward_button,
                            (GtkDestroyNotify) gtk_widget_unref);

  gtk_box_pack_start (GTK_BOX (control_horz_boxes),
                      continuous_forward_button,
                      FALSE,
                      FALSE,
                      0);

  gtk_widget_set_usize (continuous_forward_button, 30, 30);

  gtk_container_set_border_width (GTK_CONTAINER
(continuous_forward_button), 1);

  gtk_tooltips_set_tip (tooltips,
                        continuous_forward_button,
                        "Continuous Forward",
                        NULL);

  continuous_forward_box = xpm_label_box(mainwin, CF_xpm);

  gtk_container_add(GTK_CONTAINER(continuous_forward_button),
continuous_forward_box);

  gtk_widget_show (continuous_forward_box);

  gtk_widget_show (continuous_forward_button);

// ------------------- Single Step Forward Button ---------------------
-----
```

```
  single_step_forward_button = gtk_button_new();

  gtk_widget_ref (single_step_forward_button);

  gtk_object_set_data_full (GTK_OBJECT (mainwin),
                            "single_step_forward_button",
                             single_step_forward_button,
                            (GtkDestroyNotify) gtk_widget_unref);

  gtk_box_pack_start (GTK_BOX (control_horz_boxes),
                      single_step_forward_button,
                      FALSE,
                      FALSE,
                      0);

  gtk_widget_set_usize (single_step_forward_button, 30, 30);

  gtk_container_set_border_width (GTK_CONTAINER
(single_step_forward_button), 1);

  gtk_tooltips_set_tip (tooltips,
                        single_step_forward_button,
                        "Step Forward",
                        NULL);

  single_step_forward_box = xpm_label_box(mainwin, SL_xpm);

  gtk_container_add(GTK_CONTAINER(single_step_forward_button),
                    single_step_forward_box);

  gtk_widget_show (single_step_forward_box);

  gtk_widget_show (single_step_forward_button);

// ------------------- Stop Button --------------------------

  stop_button = gtk_button_new();

  gtk_widget_ref (stop_button);

  gtk_object_set_data_full (GTK_OBJECT (mainwin), "stop_button",
stop_button,
                            (GtkDestroyNotify) gtk_widget_unref);

  gtk_box_pack_start (GTK_BOX (control_horz_boxes),
                      stop_button,
                      FALSE,
                      FALSE,
                      0);

  gtk_widget_set_usize (stop_button, 30, 30);

  gtk_container_set_border_width (GTK_CONTAINER (stop_button), 1);

  gtk_tooltips_set_tip (tooltips, stop_button, "Stop", NULL);

  stop_box = xpm_label_box(mainwin, STOP_xpm);

  gtk_container_add(GTK_CONTAINER(stop_button), stop_box);

  gtk_widget_show (stop_box);

  gtk_widget_show (stop_button);
```

```
// ------------------- Single Step Backward Button -------------------
---------

  single_step_backward_button = gtk_button_new();

  gtk_widget_ref (single_step_backward_button);

  gtk_object_set_data_full (GTK_OBJECT (mainwin),
                      "single_step_backward_button",
                      single_step_backward_button,
                            (GtkDestroyNotify) gtk_widget_unref);

  gtk_box_pack_start (GTK_BOX (control_horz_boxes),
                      single_step_backward_button,
                      FALSE,
                      FALSE,
                      0);

  gtk_widget_set_usize (single_step_backward_button, 30, 30);

  gtk_container_set_border_width (GTK_CONTAINER
(single_step_backward_button), 1);

  gtk_tooltips_set_tip (tooltips, single_step_backward_button, "Step
Backward", NULL);

  single_step_backward_box = xpm_label_box(mainwin, SR_xpm);

  gtk_container_add(GTK_CONTAINER(single_step_backward_button),
                    single_step_backward_box);

  gtk_widget_show (single_step_backward_box);

  gtk_widget_show (single_step_backward_button);

// ------------------- Continuous Forward Button ---------------------
-------

  continuous_backward_button = gtk_button_new();

  gtk_widget_ref (continuous_backward_button);

  gtk_object_set_data_full (GTK_OBJECT (mainwin),
                            "continuous_backward_button",
                            continuous_backward_button,
                            (GtkDestroyNotify) gtk_widget_unref);

  gtk_box_pack_start (GTK_BOX (control_horz_boxes),
                      continuous_backward_button,
                      FALSE,
                      FALSE,
                      0);

  gtk_widget_set_usize (continuous_backward_button, 30, 30);

  gtk_container_set_border_width (GTK_CONTAINER
(continuous_backward_button), 1);

  gtk_tooltips_set_tip (tooltips,
                        continuous_backward_button,
                        "Continous Backward",
                        NULL);

  continuous_backward_box = xpm_label_box(mainwin, CB_xpm);
```

```
  gtk_container_add(GTK_CONTAINER(continuous_backward_button),
                    continuous_backward_box);

  gtk_widget_show (continuous_backward_button);

  gtk_widget_show (continuous_backward_box);

// ----------------- Detector Power Enable Button ------------------

  horz_separator = gtk_hseparator_new ();
  gtk_widget_ref (horz_separator);
  gtk_object_set_data_full (GTK_OBJECT (mainwin), "horz_separator",
     horz_separator, (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (horz_separator);
  gtk_box_pack_start (GTK_BOX (main_vert_boxes), horz_separator, TRUE,
TRUE, 0);


  detecpw_label = gtk_label_new ("Detector Power Button");
  gtk_widget_ref (detecpw_label);
  gtk_widget_show (detecpw_label);
  gtk_box_pack_start (GTK_BOX (main_vert_boxes),
                  detecpw_label, FALSE, FALSE, 0);
  gtk_widget_set_usize (detecpw_label, -2, 18);


  detecpw_horz_boxes = gtk_hbox_new (FALSE, 0);
  gtk_widget_show (detecpw_horz_boxes);
  gtk_box_pack_start (GTK_BOX (main_vert_boxes),
            detecpw_horz_boxes, FALSE, FALSE, 4);
  gtk_widget_set_usize (detecpw_horz_boxes, -2, 30);


  detecpw_button = gtk_toggle_button_new();
  gtk_widget_ref (detecpw_button);
  gtk_object_set_data_full (GTK_OBJECT (mainwin),
                          "detecpw_button",
                          detecpw_button,
                          (GtkDestroyNotify) gtk_widget_unref);
  gtk_box_pack_start (GTK_BOX (detecpw_horz_boxes),
                    detecpw_button,
                    TRUE,
                    FALSE,
                    0);
  gtk_widget_set_usize (detecpw_button, 30, 30);
  gtk_container_set_border_width (GTK_CONTAINER (detecpw_button), 1);
  gtk_tooltips_set_tip (tooltips,
                      detecpw_button ,
                      "Detector Power",
                      NULL);

  detecpw_box = xpm_label_box(mainwin, DPW_xpm);
  gtk_container_add(GTK_CONTAINER(detecpw_button), detecpw_box);
  gtk_widget_show (detecpw_button);
  gtk_widget_show (detecpw_box);

// -------------------- Setup All Buttons Array --------------------


  all_buttons[0] = increase_speed_button;
  all_buttons[1] = decrease_speed_button;
  all_buttons[2] = brake_on_button;
  all_buttons[3] = brake_off_button;
```

```
  all_buttons[4] = continuous_forward_button;
  all_buttons[5] = single_step_forward_button;
  all_buttons[6] = stop_button;
  all_buttons[7] = single_step_backward_button;
  all_buttons[8] = continuous_backward_button;
  all_buttons[9] = detecpw_button;

// -------------------- Disable All Buttons -----------------------

  buttonControl(BUTTONS_OFF);

// -------------------- Message Window ----------------------------

/*
  horz_separator = gtk_hseparator_new ();
  gtk_widget_ref (horz_separator);
  gtk_object_set_data_full (GTK_OBJECT (mainwin), "horz_separator",
     horz_separator, (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (horz_separator);
  gtk_box_pack_start (GTK_BOX (main_vert_boxes), horz_separator, TRUE,
TRUE, 0);


  message_window_label = gtk_label_new ("Message Window");
  gtk_widget_ref (message_window_label);
  gtk_widget_show (message_window_label);
  gtk_box_pack_start (GTK_BOX (main_vert_boxes),
                message_window_label, FALSE, FALSE, 0);
  gtk_widget_set_usize (message_window_label, -2, 18);


  message_window_horz_boxes = gtk_hbox_new (FALSE, 0);
  gtk_widget_show (message_window_horz_boxes);
  gtk_box_pack_start (GTK_BOX (main_vert_boxes),
            message_window_horz_boxes, FALSE, FALSE, 4);
  gtk_widget_set_usize (message_window_horz_boxes, -2, 100);


  message_window = gtk_text_new (NULL, NULL);
  gtk_text_set_editable (GTK_TEXT (message_window), FALSE);
  gtk_box_pack_start (GTK_BOX (message_window_horz_boxes),
                    message_window,
                    TRUE,
                    TRUE,
                    0);
  gtk_widget_show (message_window);


  message_window_vscrollbar =
        gtk_vscrollbar_new (GTK_TEXT (message_window)->vadj);
  gtk_box_pack_start (GTK_BOX (message_window_horz_boxes),
                message_window_vscrollbar, FALSE, FALSE, 0);
  gtk_widget_show (message_window_vscrollbar);

*/
  messageWrite(MSG01);

// -------------------- Callback Signals ----------------------------

// Quit Menu Button
  gtk_signal_connect (GTK_OBJECT (quit1), "activate",
                    GTK_SIGNAL_FUNC (trolleyControl_Quit),
                    NULL);
```

```
// Open Server Menu Button
  gtk_signal_connect (GTK_OBJECT (start_server_button), "activate",
                      GTK_SIGNAL_FUNC (start_server_activate),
                      NULL);

// Close Fifo Menu Button
  gtk_signal_connect (GTK_OBJECT (stop_server_button), "activate",
                      GTK_SIGNAL_FUNC (stop_server_activate),
                      NULL);

// About Menu Button
  gtk_signal_connect_object (GTK_OBJECT (about_button), "activate",
                             GTK_SIGNAL_FUNC (gtk_widget_show),
                             GTK_OBJECT (aboutwin));
// Continous Forward Button
  gtk_signal_connect (GTK_OBJECT (continuous_forward_button), "pressed",
                      GTK_SIGNAL_FUNC
(on_continuous_forward_button_pressed),
                      NULL);

  gtk_signal_connect (GTK_OBJECT (continuous_forward_button),
"released",
                      GTK_SIGNAL_FUNC
(on_continuous_forward_button_released),
                      NULL);

// Continous Backward Button
  gtk_signal_connect (GTK_OBJECT (continuous_backward_button),
"pressed",
                      GTK_SIGNAL_FUNC
(on_continuous_backward_button_pressed),
                      NULL);

  gtk_signal_connect (GTK_OBJECT (continuous_backward_button),
"released",
                      GTK_SIGNAL_FUNC
(on_continuous_backward_button_released),
                      NULL);

// Single Step Forward Button
  gtk_signal_connect (GTK_OBJECT (single_step_forward_button),
"pressed",
                      GTK_SIGNAL_FUNC
(on_single_step_forward_button_pressed),
                      NULL);

  gtk_signal_connect (GTK_OBJECT (single_step_forward_button),
"released",
                      GTK_SIGNAL_FUNC
(on_single_step_forward_button_released),
                      NULL);

// Single Step Backward Button
  gtk_signal_connect (GTK_OBJECT (single_step_backward_button),
"pressed",
                      GTK_SIGNAL_FUNC
(on_single_step_backward_button_pressed),
                      NULL);

  gtk_signal_connect (GTK_OBJECT (single_step_backward_button),
"released",
                      GTK_SIGNAL_FUNC
(on_single_step_backward_button_released),
                      NULL);
```

```
// Stop Button
  gtk_signal_connect (GTK_OBJECT (stop_button), "pressed",
                      GTK_SIGNAL_FUNC (on_stop_button_pressed),
                      NULL);

// Speed Increase Button
  gtk_signal_connect (GTK_OBJECT (increase_speed_button), "pressed",
                      GTK_SIGNAL_FUNC
(on_increase_speed_button_pressed),
                      NULL);

  gtk_signal_connect (GTK_OBJECT (increase_speed_button), "released",
                      GTK_SIGNAL_FUNC
(on_increase_speed_button_released),
                      NULL);

// Speed Decrease Button
  gtk_signal_connect (GTK_OBJECT (decrease_speed_button), "pressed",
                      GTK_SIGNAL_FUNC
(on_decrease_speed_button_pressed),
                      NULL);

  gtk_signal_connect (GTK_OBJECT (decrease_speed_button), "released",
                      GTK_SIGNAL_FUNC
(on_decrease_speed_button_released),
                      NULL);

// Brake On Button
  gtk_signal_connect (GTK_OBJECT (brake_on_button), "pressed",
                      GTK_SIGNAL_FUNC (on_brake_on_button_pressed),
                      NULL);

  gtk_signal_connect (GTK_OBJECT (brake_on_button), "released",
                      GTK_SIGNAL_FUNC (on_brake_on_button_released),
                      NULL);

// Brake On Button
  gtk_signal_connect (GTK_OBJECT (brake_off_button), "pressed",
                      GTK_SIGNAL_FUNC (on_brake_off_button_pressed),
                      NULL);

  gtk_signal_connect (GTK_OBJECT (brake_off_button), "released",
                      GTK_SIGNAL_FUNC (on_brake_off_button_released),
                      NULL);

// Detector Power Button
  gtk_signal_connect (GTK_OBJECT (detecpw_button), "toggled",
                      GTK_SIGNAL_FUNC (on_detecpw_button_toggle),
                      NULL);


  gtk_widget_grab_focus (mainwin);
  gtk_object_set_data (GTK_OBJECT (mainwin), "tooltips", tooltips);

  gtk_window_add_accel_group (GTK_WINDOW (mainwin), accel_group);


  return mainwin;
}

GtkWidget*
create_aboutwin (void)
{
```

```
  GtkWidget *aboutwin;
  GtkWidget *vbox1;
  GtkWidget *label2;
  GtkWidget *hseparator1;
  GtkWidget *label1;
  GtkWidget *hseparator2;
  GtkWidget *hbox1;
  GtkWidget *label3;
  GtkWidget *about_close_button;
  GtkWidget *label4;

  aboutwin = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  gtk_object_set_data (GTK_OBJECT (aboutwin), "aboutwin", aboutwin);
  gtk_widget_set_usize (aboutwin, 210, 230);
  gtk_window_set_title (GTK_WINDOW (aboutwin), "About");
  gtk_window_set_policy (GTK_WINDOW (aboutwin), FALSE, FALSE, FALSE);

  vbox1 = gtk_vbox_new (FALSE, 0);
  gtk_widget_ref (vbox1);
  gtk_object_set_data_full (GTK_OBJECT (aboutwin), "vbox1", vbox1,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (vbox1);
  gtk_container_add (GTK_CONTAINER (aboutwin), vbox1);

  label2 = gtk_label_new ("Trolley Control Program");
  gtk_widget_ref (label2);
  gtk_object_set_data_full (GTK_OBJECT (aboutwin), "label2", label2,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (label2);
  gtk_box_pack_start (GTK_BOX (vbox1), label2, FALSE, FALSE, 0);
  gtk_widget_set_usize (label2, -2, 30);

  hseparator1 = gtk_hseparator_new ();
  gtk_widget_ref (hseparator1);
  gtk_object_set_data_full (GTK_OBJECT (aboutwin), "hseparator1",
hseparator1,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (hseparator1);
  gtk_box_pack_start (GTK_BOX (vbox1), hseparator1, FALSE, FALSE, 0);
  gtk_widget_set_usize (hseparator1, -2, 2);

  label1 = gtk_label_new ("Integration Engineering Laboratory\n\nCreated
By:\nStephen Nestinger\n\nSeptember 24th, 2004\n\nUpdated:\nJune 6th,
2006");
  gtk_widget_ref (label1);
  gtk_object_set_data_full (GTK_OBJECT (aboutwin), "label1", label1,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (label1);
  gtk_box_pack_start (GTK_BOX (vbox1), label1, FALSE, FALSE, 0);
  gtk_widget_set_usize (label1, -2, 166);

  hbox1 = gtk_hbox_new (FALSE, 0);
  gtk_widget_ref (hbox1);
  gtk_object_set_data_full (GTK_OBJECT (aboutwin), "hbox1", hbox1,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (hbox1);
  gtk_box_pack_start (GTK_BOX (vbox1), hbox1, FALSE, FALSE, 0);
  gtk_widget_set_usize (hbox1, -2, 30);

  label3 = gtk_label_new ("");
  gtk_widget_ref (label3);
  gtk_object_set_data_full (GTK_OBJECT (aboutwin), "label3", label3,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_box_pack_start (GTK_BOX (hbox1), label3, FALSE, FALSE, 0);
```

```
  gtk_widget_set_usize (label3, 80, -2);
  gtk_widget_show (label3);

  about_close_button = gtk_button_new_with_label ("Close");
  gtk_widget_ref (about_close_button);
  gtk_object_set_data_full (GTK_OBJECT (aboutwin), "about_close_button",
about_close_button,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_widget_show (about_close_button);
  gtk_box_pack_start (GTK_BOX (hbox1), about_close_button, FALSE, FALSE,
0);
  gtk_widget_set_usize (about_close_button, 50, -2);
  GTK_WIDGET_UNSET_FLAGS (about_close_button, GTK_CAN_FOCUS);

  label4 = gtk_label_new ("");
  gtk_widget_ref (label4);
  gtk_object_set_data_full (GTK_OBJECT (aboutwin), "label4", label4,
                            (GtkDestroyNotify) gtk_widget_unref);
  gtk_box_pack_start (GTK_BOX (hbox1), label4, FALSE, FALSE, 0);
  gtk_widget_set_usize (label4, 80, -2);
  gtk_widget_show (label4);

  gtk_signal_connect_object (GTK_OBJECT (about_close_button),
"released",
                             GTK_SIGNAL_FUNC (gtk_widget_hide),
                             GTK_OBJECT (aboutwin));
  return aboutwin;
}


GtkWidget *xpm_label_box( GtkWidget *parent,
                          char     *xpm_filename[])
{
    GtkWidget *box1;
    GtkWidget *pixmapwid;
    GdkPixmap *pixmap;
    GdkBitmap *mask;
    GtkStyle *style;

    /* Create box for xpm and label */
    box1 = gtk_hbox_new (FALSE, 0);
    gtk_container_set_border_width (GTK_CONTAINER (box1), 0);

    /* Get the style of the button to get the
     * background color. */
    style = gtk_widget_get_style(parent);

    /* Now on to the xpm stuff */
    pixmap = gdk_pixmap_create_from_xpm_d (parent->window, &mask,
                                         &style->bg[GTK_STATE_NORMAL],
                                         xpm_filename);

    pixmapwid = gtk_pixmap_new (pixmap, mask);

    /* Create a label for the button */
    //label = gtk_label_new (label_text);

    /* Pack the pixmap and label into the box */
    gtk_box_pack_start (GTK_BOX (box1),
                        pixmapwid, TRUE, TRUE, 0);

    //gtk_box_pack_start (GTK_BOX (box1), label, FALSE, FALSE, 3);

    gtk_widget_show(pixmapwid);
```

```
    //gtk_widget_show(label);

    return(box1);
}

void
buttonControl(char cmd)
{
   int i;
/*
   if(cmd)
   {
     for(i=0; i<NUM_OF_BUTTONS; i++)
        gtk_widget_set_sensitive(all_buttons[i], TRUE);
   }
   else
   {
     for(i=0; i<NUM_OF_BUTTONS; i++)
        gtk_widget_set_sensitive(all_buttons[i], FALSE);
   }
*/

   return;
}

void
messageWrite(short msg)
{

/*
  gtk_widget_realize (message_window);
  gtk_text_freeze (GTK_TEXT(message_window));

  if(msg < MSGFF)
     gtk_text_insert (GTK_TEXT(message_window), NULL,
              &message_window->style->black, NULL,
              msgArray[msg], -1);
  else if(msg >= 0xA0 && msg <= 0xAF)
     gtk_text_insert (GTK_TEXT(message_window), NULL,
              &message_window->style->black, NULL,
              MSG_A[msg & 0x0F], -1);
  else if(msg >= 0xC0 && msg <= 0xCF)
     gtk_text_insert (GTK_TEXT(message_window), NULL,
              &message_window->style->black, NULL,
              MSG_C[msg & 0x0F], -1);
  else if(msg >= 0xE0 && msg <= 0xEF)
     gtk_text_insert (GTK_TEXT(message_window), NULL,
              &message_window->style->black, NULL,
              MSG_E[msg & 0x0F], -1);
  else
     gtk_text_insert (GTK_TEXT(message_window), NULL,
              &message_window->style->black, NULL,
              msgArray[1], -1);


  gtk_text_thaw (GTK_TEXT(message_window));
*/

  if(msg < MSGFF)
     printf(msgArray[msg]);
  else if(msg >= 0xA0 && msg <= 0xAF)
     printf(MSG_A[msg & 0x0F]);
  else if(msg >= 0xC0 && msg <= 0xCF)
     printf(MSG_C[msg & 0x0F]);
```

```
  else if(msg >= 0xE0 && msg <= 0xEF)
     printf(MSG_E[msg & 0x0F]);
  else
     printf(msgArray[1]);

  return;
}
```

main.c

```
/************************************************************
 * main.c
 *
 * Author: Stephen Nestinger
 * Date:   September 24th, 2004
 *
 ***********************************************************/

#ifdef HAVE_CONFIG_H
#  include <config.h>
#endif

#include <gtk/gtk.h>

//#include "callbacks.h"
#include "interface.h"

#pragma import "callbacks.c"
#pragma import "interface.c"
int
main (int argc, char *argv[])
{
  GtkWidget *mainwin;
  gtk_set_locale ();
  gtk_init (&argc, &argv);

  mainwin = create_mainwin ();
  gtk_widget_show (mainwin);

  gtk_main ();
  return 0;
}
```

server.c

```
#include "server.h"
#include "interface.h"
#include "control.h"

// Global Variables
char serverStatus = 0;
char connStatus = 0;
int conn_s = 0;
int list_s;
char threadStatus = 0;
pthread_t serverThread;
short int command;
char stopThread = 0;

// Private function prototypes
void* tc_server(void *arg);
void threadCleanup (void * arg);
```

```
#ifdef UDP
   struct sockaddr *trly_addr;
   socklen_t *trly_addr_len;
#endif

int tc_serverStart(void)
{
   // Check the serverStatus, if high, return -100
   if(serverStatus)
      return -100;

   stopThread = 0;

   // Start the server thread tc_server
   // pthread_create() should return 0 on success and EAGAIN, >0, on
error
   return pthread_create(&serverThread, NULL, tc_server, NULL);
}

int tc_serverStop(void)
{
   // Check the serverStatus, if high, return -100
   if(!serverStatus)
      return -100;

   shutdown(list_s, SHUT_RDWR);
   close(list_s);
   stopThread = 1;

   return 0;
}

int tc_serverSend(short int cmd)
{
   command = cmd;
   // Check the serverStatus, if low, no connection so return -100
   if(!serverStatus)
      return -100;

#ifdef UDP
   return sendto(list_s, &command, sizeof(command), 0,
                      trly_addr, *trly_addr_len);
#else
   return write(conn_s, &command, sizeof(command));
#endif
}

void* tc_server(void *arg)
{
   short int port;
   struct sockaddr_in servaddr;
   struct sockaddr_in remoteaddr;
   short int command;
   int cc;
   int addr_len;
   int bytecount;
   char message[100];
   struct timeval tv_sock;
   unsigned int yes = 1;

#ifdef UDP
   trly_addr = (struct sockaddr*)malloc(sizeof(struct sockaddr));
   trly_addr_len = (socklen_t*)malloc(sizeof(socklen_t));
```

```
#endif

   // If the server is ready, change the server status
   serverStatus = 1;
   threadStatus = 1;

   servaddr.sin_family = PF_INET;
   servaddr.sin_port = htons(PORT_NUM);
   servaddr.sin_addr.s_addr = INADDR_ANY;

   tv_sock.tv_sec  = 60;
   tv_sock.tv_usec = 0;

   addr_len = sizeof(struct sockaddr_in);

#ifndef UDP
   list_s = socket(PF_INET, SOCK_STREAM, 0);
#else
   list_s = socket(PF_INET, SOCK_DGRAM, 0);
#endif

   if (setsockopt(list_s, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(yes)) <
0)
   {
      perror("Reusing ADDR failed");
      exit(1);
   }


   if ( bind(list_s, (struct sockaddr*)&servaddr, sizeof(servaddr) ) <
0)
   {
       fprintf(stderr,"Could not bind!\n");
       threadStatus = 0;
       serverStatus = 0;
       printf("Server Thread Stopped\n");
       return NULL;
   }

#ifndef UDP
   listen(list_s, 1);

   while( (conn_s = accept(list_s, NULL, NULL))>0)
   {
      setsockopt(conn_s, SOL_SOCKET, SO_KEEPALIVE, &yes,
sizeof(yes));
      setsockopt(conn_s, SOL_SOCKET, SO_SNDTIMEO,  &tv_sock,
sizeof(tv_sock));
      setsockopt(conn_s, SOL_SOCKET, SO_RCVTIMEO,  &tv_sock,
sizeof(tv_sock));

      //messageWrite(MSG09);
      printf("conn accepted\n");
      buttonControl(BUTTONS_ON);

      do
      {
         bytecount = recv(conn_s, &command, sizeof(command), 0);

         /* Added to work on PPC based Mac which is Big Endian */
         #ifdef PPC
         short int temp;
         char *from, *to;
         from = (char *)&command;
```

```
         to = (char *)&temp;
         memcpy(to+1, from,    1);
         memcpy(to,   from+1,  1);
         command = temp;
         //printf("message from trolley = %x\n", command);
         #endif

         switch(command)
         {
            case TRM_SPEED:
               bytecount = recv(conn_s, &command, sizeof(command), 0);

               /* Added to work on PPC based Mac which is Big Endian */
               #ifdef PPC
               memcpy(to+1, from,    1);
               memcpy(to,   from+1,  1);
               command = temp;
               #endif

               printf("Speed now set to %d\n", command);
               break;
            case TRM_BRAKES_ON:
               printf("Breaks are fully actuated\n");
               break;
            case TRM_BRAKES_OFF:
               printf("Breaks are fully open\n");
               break;
            case TRM_DET_ENABLE:
               printf("Power to the detector enabled\n");
               break;
            case TRM_DET_DISABLE:
               printf("Power to the detector disabled\n");
               break;
         }

      } while(bytecount > 0);
      //messageWrite(MSG10);
      printf("disconnected\n");
      close(conn_s);

      // Disable all of the buttons
      buttonControl(BUTTONS_OFF);
   }
#else
   while(!stopThread)
   {
      if( (bytecount = recvfrom(list_s, &command, sizeof(command), 0,
                  trly_addr, trly_addr_len) > 0))
      {
         switch(command)
         {
            case TRM_SPEED:
               bytecount = recvfrom(list_s, &command, sizeof(command),
0,
                  trly_addr, trly_addr_len);
               printf("Speed now set to %d\n", command);
               break;
            case TRM_BRAKES_ON:
               printf("Breaks are fully actuated\n");
               break;
            case TRM_BRAKES_OFF:
               printf("Breaks are fully open\n");
               break;
            case TRM_DET_ENABLE:
```

```
               printf("Power to the detector enabled\n");
               break;
            case TRM_DET_DISABLE:
               printf("Power to the detector disabled\n");
               break;
            case TYPE_TRLY:
               printf("Trolley found\n");
               command = TYPE_SERV;
               sendto(list_s, &command, sizeof(command), 0,
                      trly_addr, *trly_addr_len);
               break;
         }
      }
   }
#endif

   // Before we exit, change the server and thread status
   threadStatus = 0;
   serverStatus = 0;
   printf("Server Thread Stopped\n");
   return 0;
}
```

Makefile

```
#-----------------------------------------------------------------
# Makefile for the trolley control program
# ----------------------------------------------------------------

# imported variables from the command line
# use make V=1 if you wish to have verbose output
ifndef V
  quiet = @
endif

ifdef UDP
  DEF = -DUDP
endif

ifdef TEST
  TARGET2 = testClient
  OBJS2 = src/client.$(OBJ)
endif

# Macros
CC     = $(quiet) gcc
RM     = $(quiet) -rm
CFLAGS = -g -DPPC
IFLAGS = -I include `gtk-config --cflags`
LIBS   = `gtk-config --libs` -lpthread
OBJ    = o
OBJS   = src/main.$(OBJ)      \
       src/interface.$(OBJ) \
       src/callbacks.$(OBJ) \
       src/server.$(OBJ) \
       src/control.$(OBJ)
       src/client.$(OBJ)
TARGET = trolleyControl

all:: $(TARGET) $(TARGET2)

# Explicit rules
$(TARGET):: $(OBJS)
      $(if $(quiet), @ echo Building $@)
      $(CC) $(CFLAGS) -o $@ $(OBJS) $(LIBS)

$(TARGET2):: $(OBJS2)
      $(if $(quiet), @ echo Building $@)
      $(CC) $(CFLAGS) -o $@ $(OBJS2) $(LIBS)

%.$(OBJ) :: %.c
      $(if $(quiet), @ echo Compiling $@)
      $(CC) $(CFLAGS) -c $*.c -o $@ $(IFLAGS) $(DEF)

.PHONY: clean
clean::
      $(if $(quiet), @ echo Removing $(OBJS))
      $(RM) -f $(OBJS) $(OBJS2)

# ----------------------------------------------------------------
```

Appendix F: Trolley Microprocessor Program

```
/**********************************************************************
 *  Trolley Wireless Rabbit Controller
 *  created & modified by:
 *     Stephen Nestinger& Matt Campbell
 *  last change: 9-8-2006
 **********************************************************************/

//----------------------------------------------------------
// Defines
//----------------------------------------------------------
  // uncomment for additional tcp stack debugging info
  //#define DCRTCP_VERBOSE

  // uncomment the following for additional CF Wifi debugging info
  //#define CF_VERBOSE
  //#define CF_DEBUG

  // uncomment when programming cable connected
  //#define PROGRAM

  // What is this for?
  #memmap xmem

  // Setup default network parameters
  // Type 3 has Ethernet and DHCP
  #define TCPCONFIG   100
  #define REMOTE_IP   "10.0.0.11"  // IP address of server
  #define REMOTE_PORT 2003         // Port for server connection
  #define CMD_LENGTH  2

  // Set the default wifi paramenters before sock_init()
  //   MODE     = connection mode: BSS = Managed, IBSS = Ad-Hoc
  //   SSID     = ESSID of access point you wish to connect to
  //   WEP_FLAG = enable or disable WEB: 0 = disable
  #define WIFI_INIT     \
    pd_ioctl(0,WIFI_MODE,       "BSS"          ,0); \
    pd_ioctl(0,WIFI_SSID,       "TROLLEYNET"   ,0); \
    pd_ioctl(0,WIFI_WEP_FLAG,   "0"            ,0);
    // Use only when WEP security is enabled
    // pd_ioctl(0,WIFI_WEP_USEKEY, "1"            ,0); \
    // pd_ioctl(0,WIFI_WEP_KEY0,   "ababababab"   ,0);

  // Bring in the cfprim drivers
  //   Linsys uses the prism chipset
  #define PKTDRV cfprism.lib

  // Bring in TCP stack
  //   Strictly for networking
  #use dcrtcp.lib


  // System Definitions
  #define BLINK_TIMES         5      // Number of blinks on start-up
  #define LED_ON              0      // bit setting for LED on
  #define LED_OFF             1      // bit setting for LED off
  #define SOCK_TRIES          5      // Number of tries to open socket
  #define DRIVE_FORWARD       0      // Drive motor direction settings
  #define DRIVE_BACKWARD      1      // Trolley is Rear Wheel Drive!
  #define BRAKES_LOCK         0      // Brake motor direction setting
  #define BRAKES_UNLOCK       1      // 0 -> Lock; 0 -> Unlock
  #define BRAKES_ON           0      // Brake motor enable
  #define BRAKES_OFF          1      // Brake motor disable
```

```
  #define DETECTOR_ON         0      // turn on detector power relay
  #define DETECTOR_OFF        1      // turn off detector power relay
  #define MOVING              1      // for "movingStatus" variable in
  #define NOT_MOVING          0      // continuous move = 1, stopped = 0
  #define MOTOR_BRAKE_ON      0      // H-Bridge motor brake control
  #define MOTOR_BRAKE_OFF     1      // 0 -> On; 1 -> Off
  #define UP                  1      // direction control for ramp
function
  #define DOWN                0      //    "         "     "    "     "
  #define STEP_LENGTH         10     // sleep time for a single small
step
  #define BIG_STEP_LENGTH     50     // sleep time for a single big step

  // Port Bit Defines
  // for Brakes
  #define PORT_LEFT_BRAKE_SWITCH_LOCK    0 //port bit for left brake
lock switch
  #define PORT_LEFT_BRAKE_SWITCH_UNLOCK  1 //  "   "   "   "     "
unlock  "
  #define PORT_RIGHT_BRAKE_SWITCH_LOCK   2 //  "   "   "  right  "
lock   "
  #define PORT_RIGHT_BRAKE_SWITCH_UNLOCK 3 //  "   "   "   "     "
unlock  "
  #define PORT_LEFT_BRAKE_DIRECTION      1 //port bit for left brake
direction
  #define PORT_LEFT_BRAKE_CONTROL        2 //port bit for left brake
control
  #define PORT_RIGHT_BRAKE_DIRECTION     3 //port bit for right brake
direction
  #define PORT_RIGHT_BRAKE_CONTROL       4 //port bit for right brake
control
  // for Drive Motor
  #define PORT_DRIVE_MOTOR_CONTROL       4 //port bit for drive motor
control
  #define PORT_DRIVE_MOTOR_DIRECTION     0 //port bit for drive motor
direction
  // for Motors
  #define PORT_MOTOR_BRAKE               6 // port for motor brake
  // for Detector Power Relay
  #define PORT_DETECTOR_POWER            5 // port for detector power
  // for LEDs
  #define PORT_LED_1  7 // port bit for LED 1; port G bit 7
  #define PORT_LED_2  5 // port bit for LED 2; port F bit 5
  #define PORT_LED_3  6 // port bit for LED 3; port F bit 6
  #define PORT_LED_4  0 // port bit for LED 4; port C bit 0
  #define PORT_LED_5  2 // port bit for LED 5; port C bit 2
  #define PORT_LED_6  4 // port bit for LED 6; port C bit 4
  #define PORT_LED_7  6 // port bit for LED 7; port C bit 6

  // PWM Definitions
  #define PWM_FREQ            1000   // Set clock freq on pwm for drive
motor
  #define PWM_CHANNEL         0      // Set pwm channel to 0 -> Port F4
  #define PWM_OPTION          0      // Set pwm option to single block

  // System Types
  #define TYPE_LBDS              0xA0   // LBDS System
  #define TYPE_TRLY              0xA1   // Trolley System

  // Trolley Messages to return to user
  #define TRM_SPEED              0xB0   // Sending drive motor duty
cycle
  #define TRM_BRAKES_ON          0xB1   // Brakes are fully actuated
  #define TRM_BRAKES_OFF         0xB2   // Brakes are fully open
```

```
  #define TRM_DET_ENABLE         0xB3   // Power to detector enabled
  #define TRM_DET_DISABLE        0xB4   // Power to detector enabled
  #define TRM_CONN_CLOSING       0xB5   // Closing TCP connection

  // Trolley control commands -> Trolley is Rear Wheel Drive
  #define TRC_STEP_FORWARD       0xC0   // Step the trolley forward
  #define TRC_STEP_BACKWARD      0xC1   // Step the trolley backward
  #define TRC_CONT_FORWARD       0xC2   // Continously move the trolley
Forward
  #define TRC_CONT_BACKWARD      0xC3   // Continously move the trolley
Backward
  #define TRC_BRAKE              0xC4   // Engage trolley brakes
  #define TRC_UNBRAKE            0xC5   // Disengage trolley brakes
  #define TRC_SPEED_UP           0xC6   // Speed up the trolley
  #define TRC_SPEED_DOWN         0xC7   // Slow down the trolley
  #define TRC_STOP               0xC8   // Stop trolley motion
  #define TRC_DETECTOR_POWER_ON  0xC9   // Turn on power to detector
  #define TRC_DETECTOR_POWER_OFF 0xCA   // Turn off power to detector
  #define TRC_CLOSE_CONN         0xCB   // Close the TCP/IP Conn

  // Trolley errors
  #define TRE_NET_OK             0xE0   // Communications established
  #define TRE_NET_FAIL           0xE1   // Communication failure
  #define TRE_NET_DHCP_FB        0xE2   // DHCP address with fallbacks
  #define TRE_NET_DHCP_NFB       0xE3   // DHCP address without
fallbacks
  #define TRE_BRAKE_FAIL         0xE4   // Brakes not working properly
  #define TRE_MOTOR_FAIL         0xE5   // Drive motor not responding

  // This sytem specific defines
  #define MY_TYPE TYPE_TRLY     // What type am I

//---------------------------------------------------------
//  End Defines
//---------------------------------------------------------

//---------------------------------------------------------
// Function Prototypes
//---------------------------------------------------------
  int baseSysInit(void);      // initialize system
  int portInit(void);         //     "      Rabbit ports
  int sockInit(void);         //     "      communications
  int pwmInit(void);          //     "      PWM signal
  int openTCP(void);          // open tcp socket
  int closeTCP(void);         // close tcp socket
  int stepForward(void);      // step trolley Forward once
  int stepBackward(void);     // step trolley Backward once
  int contForward(void);      // move trolley continuously Froward
  int contBackward(void);     // move trolley continuously Backward
  int stop(void);             // stop continuous movement for trolley
  int brake(void);            // engage brakes
  int unbrake(void);          // disengage brakes
  int speedUp(void);          // increase drive speed -> increase PWM
duty cycle
  int speedDown(void);        // decrease drive speed -> decrease PWM
duty cycle
  int checkBrakes(void);      // check brake status -> locked = 0 &
unlocked = 1
  int enableDetector(void);   // turn on power to detector
  int disableDetector(void);  // turn off power to detector
  // ramps drive motor up and down for smooth motion
  void ramp(unsigned int time, unsigned int interval, char dir);
  // delay function for multiples of 100 ms
  void trSleep(unsigned int hundMS);
```

```
  // delay function for multiple of 1 ms
  void trMSleep(unsigned int ms);
  // check for continuous command to move while executing continuous
move
  void continuousMoveCheck(void);
  // clear buffered commands received while executing a function
  void clearRecBuffer(void);
//----------------------------------------------------------
// End Function Prototypes
//----------------------------------------------------------

//----------------------------------------------------------
// Global Variables
//----------------------------------------------------------
  static tcp_Socket tcpsock;   // socket for tcp communication
  int command;                 // incoming command storage
  unsigned int duty_cycle;     // pwm duty cycle for drive motor as seen
by user
  char movingStatus;           // determine if in continuous move or not
  int left_brake_status;       // status of left brake
  int right_brake_status;      // status of right brake
  longword ping_who;           // resolve ip address of server
//  int step_complete;           // set to make only one step happen per
command
//  int drive_dir;               // controls drive direction
//  int brake_dir;               // controls brake direction
//  unsigned int rec_buf_size;   // size of receive buffer on tcp socket
//----------------------------------------------------------
// End Global Variables
//----------------------------------------------------------

//----------------------------------------------------------
// Main
//----------------------------------------------------------
  void main()
  {

    struct _wifi_status wstatus;   // for wifi
    int error;                     // errors returned from functions
    movingStatus = NOT_MOVING;     // set moving status to 0, not moving
    ping_who = resolve(REMOTE_IP); // get ip address for ping connection
check

    // Initialization Base System:
    //   Intialize the ports
    //   Flash LEDs at start up
    //   Check status of the system
    //   Output LED status
    //   Not sure want to engage brakes on startup
      //   Make sure brakes are engaged, if not, engage them
      //   Blink LED while engaging brakes
      //   Update status of LED when braking complete
    baseSysInit();  // Initialize required ports

    // Intialize Communications:
    //   Initialize packet driver system
    //   Connect to access point
    error = sockInit();  // Initialize packet driver system

    // Open TCP/IP sockect for communication
    // Try to connect to server
    openTCP();

    // get size of receive buffer for when need to read in
```

```
    // unwanted buffered commands
    //rec_buf_size = sock_rbsize(&tcpsock);
    //printf("rec_buf_size = %d\n", rec_buf_size);

    while(1)
    {

       tcp_tick(&tcpsock);   // forces ethernet backgroup processes to
go

       // Check the status of the TCP/IP socket
       // if connected, wait for commands ; if not, open connection
again
       // if port closed, stop everything and enable brakes
       // try to re-establish communication
       pd_ioctl( 0,WIFI_STATUS, (char *)&wstatus, sizeof(wstatus) );
       if(wstatus.status == 5 || !sock_established(&tcpsock))
       {
          stop();
          openTCP();
       }
       //trSleep(5);
       // printf("Socket Established = %d\n",
sock_established(&tcpsock));

       // process commands
       // deal with controls
       command = 0; // reset command input

       // Check for unwanted buffered commands
       // clear input if more than one is waiting
       if ( sock_bytesready(&tcpsock) > 2 )
          clearRecBuffer();

       // Read message from trolley control program
       if ( ( sock_aread ( &tcpsock, (char *)&command, CMD_LENGTH )) ==
2 )
       {
          #ifdef PROGRAM
          printf("Command received, %d: %X\n", command, command);
          //printf("sizeof(command) = %d  --- sizeof(short) = %d\n",
          //   sizeof(command), sizeof(short));
          #endif// PROGRAM
       }


       // Trolley Commands -> Trolley is Rear Wheel Drive
       switch ( command )
       {
         case TRC_STEP_FORWARD:       // Step the trolley forward
           stepForward();
           break;
         case TRC_STEP_BACKWARD:      // Step the trolley backward
           stepBackward();
           break;
         case TRC_CONT_FORWARD:       // Continuously move trolley
forward
           //contForward();
           bigStepForward(); // Continuous not working, use big step
instead
           break;
         case TRC_CONT_BACKWARD:      // Continuously move trolley
backward
           //contBackward();
```

```
          bigStepBackward(); // Continuous not working, use big step
instead
          break;
        case TRC_STOP:                // Stop the trolley
          stop();
          break;
        case TRC_BRAKE:               // Engage trolley brakes
          brake();
          break;
        case TRC_UNBRAKE:             // Disengage trolley brakes
         unbrake();
          break;
        case TRC_SPEED_UP:            // Speed up the trolley
          speedUp();
          break;
        case TRC_SPEED_DOWN:          // Slow down the trolley
          speedDown();
          break;
        case TRC_DETECTOR_POWER_ON:  // turn on power to detector
          enableDetector();
          break;
        case TRC_DETECTOR_POWER_OFF: // turn on power to detector
          disableDetector();
          break;
        case TRC_CLOSE_CONN:          // close tcp connection
          stop();
          closeTCP();
          break;
      } // end switch for trolley commands

      // blink in main loop
      BitWrPortI(PGDR, &PGDRShadow, LED_ON, PORT_LED_1);
      trMSleep(25);
      BitWrPortI(PGDR, &PGDRShadow, LED_OFF, PORT_LED_1);
      trMSleep(25);

    } // end while(1)

  } // end main()

//----------------------------------------------------------
// baseSysInit() : Initialization Base System:
//      Intialize the ports
//      Flash LEDs at start up
//      Check status of the system
//      Output LED status
//      Make sure brakes are engaged, if not, engage them
//      Blink LED while engaging brakes
//      Update status of LED when braking complete
//----------------------------------------------------------
  int baseSysInit(void)
  {
    int i;
    i = 0;

    portInit(); // initialize the ports
    pwmInit();  // initialize pwm for drive motor

    // Turn all LEDs off
    BitWrPortI(PGDR, &PGDRShadow, LED_OFF, PORT_LED_1);
    BitWrPortI(PFDR, &PFDRShadow, LED_OFF, PORT_LED_2);
    BitWrPortI(PFDR, &PFDRShadow, LED_OFF, PORT_LED_3);
    #ifndef PROGRAM
    BitWrPortI(PCDR, &PCDRShadow, LED_OFF, PORT_LED_4);
```

```
BitWrPortI(PCDR, &PCDRShadow, LED_OFF, PORT_LED_5);
BitWrPortI(PCDR, &PCDRShadow, LED_OFF, PORT_LED_6);
BitWrPortI(PCDR, &PCDRShadow, LED_OFF, PORT_LED_7);
#endif// NOT PROGRAM

while(1)// Blink LEDs on startup
{
   if ( i >= BLINK_TIMES )
     break;

   #ifndef PROGRAM
   BitWrPortI(PCDR, &PCDRShadow, LED_OFF, PORT_LED_7);
   #endif// NOT PROGRAM
   #ifdef PROGRAM
   BitWrPortI(PFDR, &PFDRShadow, LED_OFF, PORT_LED_3);
   #endif// PROGRAM
   BitWrPortI(PGDR, &PGDRShadow, LED_ON, PORT_LED_1);
   trMSleep(25);
   BitWrPortI(PGDR, &PGDRShadow, LED_OFF, PORT_LED_1);
   BitWrPortI(PFDR, &PFDRShadow, LED_ON, PORT_LED_2);
   trMSleep(25);
   BitWrPortI(PFDR, &PFDRShadow, LED_OFF, PORT_LED_2);
   BitWrPortI(PFDR, &PFDRShadow, LED_ON, PORT_LED_3);
   trMSleep(25);
   #ifndef PROGRAM
   BitWrPortI(PFDR, &PFDRShadow, LED_OFF, PORT_LED_3);
   BitWrPortI(PCDR, &PCDRShadow, LED_ON, PORT_LED_4);
   trMSleep(25);
   BitWrPortI(PCDR, &PCDRShadow, LED_OFF, PORT_LED_4);
   BitWrPortI(PCDR, &PCDRShadow, LED_ON, PORT_LED_5);
   trMSleep(25);
   BitWrPortI(PCDR, &PCDRShadow, LED_OFF, PORT_LED_5);
   BitWrPortI(PCDR, &PCDRShadow, LED_ON, PORT_LED_6);
   trMSleep(25);
   BitWrPortI(PCDR, &PCDRShadow, LED_OFF, PORT_LED_6);
   BitWrPortI(PCDR, &PCDRShadow, LED_ON, PORT_LED_7);
   trMSleep(25);
   #endif// NOT PROGRAM

   i++;
} // end while(1)

// Turn all LEDs off
BitWrPortI(PGDR, &PGDRShadow, LED_OFF, PORT_LED_1);
BitWrPortI(PFDR, &PFDRShadow, LED_OFF, PORT_LED_2);
BitWrPortI(PFDR, &PFDRShadow, LED_OFF, PORT_LED_3);
#ifndef PROGRAM
BitWrPortI(PCDR, &PCDRShadow, LED_OFF, PORT_LED_4);
BitWrPortI(PCDR, &PCDRShadow, LED_OFF, PORT_LED_5);
BitWrPortI(PCDR, &PCDRShadow, LED_OFF, PORT_LED_6);
BitWrPortI(PCDR, &PCDRShadow, LED_OFF, PORT_LED_7);
#endif// NOT PROGRAM

// turn off motor brake to allow motors to spin
BitWrPortI(PGDR, &PGDRShadow, MOTOR_BRAKE_OFF, PORT_MOTOR_BRAKE);

// Check brake status and set status variables
left_brake_status = checkLeftBrake();
right_brake_status = checkRightBrake();

// Not sure about engaging brakes at startup
// engage brakes if not already
// TEST THIS LOGIC BEFORE RUNNING TROLLEY WITH IT!
// if ( left_brake_status != 1 || right_brake_status != 1 )
```

```
    //   brake();
    // TEST THIS LOGIC BEFORE RUNNING TROLLEY WITH IT!

    return 0;
  } // end baseSysInit()

//----------------------------------------------------------
// pwmInit(): sets pwm clock freq and set duty cycle
//----------------------------------------------------------
  int pwmInit(void)
  {
    unsigned long set_freq;

    set_freq = pwm_init(PWM_FREQ);  // set pwm clock freq
    #ifdef PROGRAM
    printf("Clock freq set for drive motor pwm = %d\n", set_freq);
    #endif// PROGRAM

    // set duty cycle to 0% -> really @ 100% cause logic reversed
    // returns 0 = okay
    if ( pwm_set ( PWM_CHANNEL, 1024, PWM_OPTION ) )
    {
      #ifdef PROGRAM
      printf("ERROR - problems initializing pwm channel\n");
      #endif// PROGRAM
    }

    // set initial duty cycle to 100% for first moves
    // possible pwm setting 0 - 1024 -> means ticks on = duty cycle
    // because of logic reverse:
    // setting pwm to 0 => 100% motor on
    // setting pwm to 1024 => 0% motor on
    // when using duty_cycle, use ((100 - duty_cycle) * 10.24)
    // cast to (unsigned int)
    duty_cycle = 100;

    return 0;
  } // end pwmInit()

//----------------------------------------------------------
// sockInit():  initializes the packet driver system
//              prints out status
//   return value: 0 = ethernet initialized
//                -1 = ethernet not working for some reason
//----------------------------------------------------------
  int sockInit(void)
  {
    int status;            // system status
    status = sock_init();  // initialize the packet driver system
    switch(status)
    {
       case 0:
           #ifdef PROGRAM
           printf("Network was successfully initialized\n");
           #endif// PROGRAM
           break;
       case 1:
           #ifdef PROGRAM
           printf("Ethernet packet driver initialization failed\n");
           #endif// PROGRAM
           return -1;
       case 2:
           #ifdef PROGRAM
           printf("DHCP failed, using fallback definition\n");
```

```
            #endif// PROGRAM
            return -1;
        case 3:
            #ifdef PROGRAM
            printf("DHCP failed, no fallbacks defined\n");
            #endif// PROGRAM
            return -1;
        default:
     }

     // Possible implimentation in future for security
     // Check to see if we are connected to IEL
     //   if not keep trying

     return 0;
   } // end sockInit()

//---------------------------------------------------------
// openTCP():  opens TCP socket
//---------------------------------------------------------
  int openTCP(void)
  {
     short cmd;
     int i;
     int tries;

     #ifdef PROGRAM
     printf("Entering openTCP...\n");
     #endif// PROGRAM
     sock_abort(&tcpsock);

     while(1)
     {
        #ifdef PROGRAM
        printf("\nCall TCP_OPEN()\n");
        #endif// PROGRAM
        tcp_tick(NULL);

        if(tcp_open ( &tcpsock, 1050, resolve(REMOTE_IP), REMOTE_PORT,
NULL ))
        {
           tries = 0;
           while ( !sock_established ( &tcpsock ) &&
                    sock_bytesready(&tcpsock) == -1 &&
                    tries++ < SOCK_TRIES)
           {
              tcp_tick(&tcpsock);

              #ifdef PROGRAM
              printf("\tWaiting to establish connection\n");
              printf("Est = %d - tick = %d\n\n",
                sock_established(&tcpsock), tcp_tick(&tcpsock));
              #endif// PROGRAM
              // blinking LEDs for establishing connection
              BitWrPortI(PFDR, &PFDRShadow, LED_ON, PORT_LED_2);
              trSleep(2);
              BitWrPortI(PFDR, &PFDRShadow, LED_OFF, PORT_LED_2);
              trSleep(2);
          } // end while ( !socket established etc... )
        }

        // if sock is successfully established, break from loop
        if(sock_established ( &tcpsock ))
           break;
```

```
        #ifdef PROGRAM
        printf("TCP_OPEN() Unsuccessfull\n\n");
        #endif// PROGRAM

        // quick blink, tried to estblish socket SOCK_TRIES times
        // and not successful, will retry next time thru while(1) loop
        for(i=0; i<BLINK_TIMES; i++)
        {
          BitWrPortI(PFDR, &PFDRShadow, LED_ON, PORT_LED_2);
          trMSleep(50);
          BitWrPortI(PFDR, &PFDRShadow, LED_OFF, PORT_LED_2);
          trMSleep(50);
        }
     } // end while(1)

     cmd = TRE_NET_OK;
     sock_awrite(&tcpsock, (char *)&cmd, CMD_LENGTH);
     printf("TCP esablished successfully!\n\n");

     #ifdef PROGRAM
     printf("Leaving openTCP...\n");
     #endif// PROGRAM
   } // end openTCP()

 //----------------------------------------------------------
 // closeTCP():  opens TCP socket
 //----------------------------------------------------------
   int closeTCP(void)
   {
      sock_awrite(&tcpsock, (char *)TRM_CONN_CLOSING, CMD_LENGTH);
      #ifdef PROGRAM
      printf("Closing conn!\n");
      #endif// PROGRAM
      tcp_close(&tcpsock);
   } // end closeTCP()

 //----------------------------------------------------------
 // stop(): stop continuous trolley movement
 //----------------------------------------------------------
   int stop(void)
   {
     #ifdef PROGRAM
     printf("running stop\n");
     #endif// PROGRAM

     if(movingStatus)
        ramp(1,1,DOWN);
     else
     {
       pwm_set( PWM_CHANNEL, 1024, PWM_OPTION );
     }
     // turn off brakes motors
     BitWrPortI(PGDR, &PGDRShadow, BRAKES_OFF, PORT_LEFT_BRAKE_CONTROL);
     BitWrPortI(PGDR, &PGDRShadow, BRAKES_OFF, PORT_RIGHT_BRAKE_CONTROL);

     // Turn all LEDs off
     BitWrPortI(PGDR, &PGDRShadow, LED_OFF, PORT_LED_1);
     BitWrPortI(PFDR, &PFDRShadow, LED_OFF, PORT_LED_2);
     BitWrPortI(PFDR, &PFDRShadow, LED_OFF, PORT_LED_3);
     #ifndef PROGRAM
     BitWrPortI(PCDR, &PCDRShadow, LED_OFF, PORT_LED_4);
     BitWrPortI(PCDR, &PCDRShadow, LED_OFF, PORT_LED_5);
     BitWrPortI(PCDR, &PCDRShadow, LED_OFF, PORT_LED_6);
```

```
    BitWrPortI(PCDR, &PCDRShadow, LED_OFF, PORT_LED_7);
    #endif// NOT PROGRAM

    // read Detector Power status and reset LED - only persistent LED
    if ( (BitRdPortI(PGDR, PORT_DETECTOR_POWER)) == DETECTOR_ON )
    {
      BitWrPortI(PFDR, &PFDRShadow, LED_ON, PORT_LED_3);
    }

    movingStatus = NOT_MOVING;
  } // end stop()

//----------------------------------------------------------
// stepForward(): step trolley Forward once
//----------------------------------------------------------
  int stepForward(void)
  {
    #ifdef PROGRAM
    printf("running stepForward\n");
    #endif// PROGRAM

    // set drive direction
    BitWrPortI(PGDR, &PGDRShadow, DRIVE_FORWARD,
PORT_DRIVE_MOTOR_DIRECTION);

    // check if brakes are on with _brake_status variable
    // if true -> run unbrake()
    left_brake_status = checkLeftBrake();
    right_brake_status = checkRightBrake();
    if ( left_brake_status != -1 || right_brake_status != -1 )
      unbrake();

    // ramp up to speed, move set step length, ramp down
    ramp(1,1,UP);
    trSleep(STEP_LENGTH);
    ramp(1,1,DOWN);

    return 0;
  } // end stepForward()

//----------------------------------------------------------
// stepBackward(): step trolley Backward once
//----------------------------------------------------------
  int stepBackward(void)
  {
    #ifdef PROGRAM
    printf("running stepBackward\n");
    #endif// PROGRAM

    // set drive direction
    BitWrPortI(PGDR, &PGDRShadow, DRIVE_BACKWARD,
PORT_DRIVE_MOTOR_DIRECTION);

    // check if brakes are on with _brake_status variable
    // if true -> run unbrake()
    left_brake_status = checkLeftBrake();
    right_brake_status = checkRightBrake();
    if ( left_brake_status != -1 || right_brake_status != -1 )
      unbrake();

    // ramp up to speed, move set step length, ramp down
    ramp(1,1,UP);
    trSleep(STEP_LENGTH);
    ramp(1,1,DOWN);
```

```
    return 0;
  } // end stepBackward()

//----------------------------------------------------------
// bigStepForward(): take a big step Forward once
//----------------------------------------------------------
  int bigStepForward(void)
  {
    #ifdef PROGRAM
    printf("running bigStepForward\n");
    #endif// PROGRAM

    // set drive direction
    BitWrPortI(PGDR, &PGDRShadow, DRIVE_FORWARD,
PORT_DRIVE_MOTOR_DIRECTION);

    // check if brakes are on with _brake_status variable
    // if true -> run unbrake()
    left_brake_status = checkLeftBrake();
    right_brake_status = checkRightBrake();
    if ( left_brake_status != -1 || right_brake_status != -1 )
       unbrake();

    // ramp up to speed, move set step length, ramp down
    ramp(1,1,UP);
    trSleep(BIG_STEP_LENGTH);
    ramp(1,1,DOWN);

    return 0;
  } // end bigStepForward()

//----------------------------------------------------------
// bigStepBackward(): take a big step Backward once
//----------------------------------------------------------
  int bigStepBackward(void)
  {
    #ifdef PROGRAM
    printf("running bigStepBackward\n");
    #endif// PROGRAM

    // set drive direction
    BitWrPortI(PGDR, &PGDRShadow, DRIVE_BACKWARD,
PORT_DRIVE_MOTOR_DIRECTION);

    // check if brakes are on with _brake_status variable
    // if true -> run unbrake()
    left_brake_status = checkLeftBrake();
    right_brake_status = checkRightBrake();
    if ( left_brake_status != -1 || right_brake_status != -1 )
       unbrake();

    // ramp up to speed, move set step length, ramp down
    ramp(1,1,UP);
    trSleep(BIG_STEP_LENGTH);
    ramp(1,1,DOWN);

    return 0;
  } // end bigStepBackward()

//----------------------------------------------------------
// contForward(): move trolley continuously Froward
// Not used cause buggy
//----------------------------------------------------------
```

```
  int contForward(void)
  {
    #ifdef PROGRAM
    printf("running contForward\n");
    #endif// PROGRAM

    // check to see if brakes are fully open
//    if (  left_brake_status != -1 && right_brake_status != -1 )
//       unbrake();

    BitWrPortI(PGDR, &PGDRShadow, DRIVE_FORWARD, 0); // set drive
direction
    // turn on drive motor @ duty_cycle
    // pwm_set( PWM_CHANNEL, (int)(((100-duty_cycle)*0.01)*1024),
PWM_OPTION );
    // movingStatus = MOVING; // set moving status variable
    // ramp(1,1,UP); // ramp drive motor up to duty_cycle
    // continuousMoveCheck();
    stop();
  } // end contForward()

//----------------------------------------------------------
// contBackward(): move trolley continuously Backward
// Not used cause buggy
//----------------------------------------------------------
  int contBackward(void)
  {
    #ifdef PROGRAM
    printf("running contBackward\n");
    #endif// PROGRAM

    //
//    if (  left_brake_status != -1 && right_brake_status != -1 )
//       unbrake();

    BitWrPortI(PGDR, &PGDRShadow, DRIVE_BACKWARD, 0); // set drive
direction
    // turn on drive motor @ duty_cycle
    // pwm_set( PWM_CHANNEL, (int)(((100-duty_cycle)*0.01)*1024),
PWM_OPTION );
    // movingStatus = MOVING; // set moving status variable
    // ramp(1,1,UP); // ramp motor up to duty_cycle
    // continuousMoveCheck();
    stop();
  } // end contBackward()

//----------------------------------------------------------
// continuousMoveCheck(): check continually for command input
//  while executing a continuous move & monitor for stop command
//  NOT WORKING WELL - because of delay
//----------------------------------------------------------
  void continuousMoveCheck(void)
  {
     int receive;
     int stop2;
     receive = 1;
     stop2 = 0;
     command = 0;

     while ( receive ) // while receiving move command go forward.
     {
       tcp_tick(&tcpsock);
       receive = 0;
       command = 0;
```

```
      #ifdef PROGRAM
      printf("Waiting to recieve commands\n");
      #endif// PROGRAM
      trSleep(3);
      tcp_tick(&tcpsock);
      trSleep(3);
      #ifdef PROGRAM
      printf("Data bytes %d %d\n",
             sock_dataready(&tcpsock), sock_bytesready(&tcpsock));
      #endif// PROGRAM
      // read all commands looking for a stop
      while (!stop2 && tcp_tick(&tcpsock))
      {
         if( sock_aread (&tcpsock, (char *)&command, CMD_LENGTH) < 1 )
         {
              #ifdef PROGRAM
              printf("Stopping because sock_aread() <= 0\n");
              #endif// PROGRAM
              receive = 0;
              stop();
              break;
         }
         else
         {
         #ifdef PROGRAM
         printf("Command received = %d\n", command);
         #endif// PROGRAM
         switch ( command )
         {
            case TRC_CONT_FORWARD:
            case TRC_CONT_BACKWARD:
              #ifdef PROGRAM
              printf("Move\n");
              #endif// PROGRAM
              receive = 1;
              break;
            default:
              #ifdef PROGRAM
              printf("Stop\n");
              #endif// PROGRAM
              receive = 0;
              stop2 = 1;
              stop();
              break;
         }
         }
      } // end while ( bytes in socket && not stop )
    } // end while(i)

    if ( stop2 )
    {
      #ifdef PROGRAM
      printf("Received Stop Command.\n");
      #endif// PROGRAM
    }
    else
    {
      #ifdef PROGRAM
      printf("Lost Connection with Server.\n");
      #endif// PROGRAM
    }
  } // end continuousMoveCheck()

//----------------------------------------------------------
```

```
// brake(): engage brakes
//---------------------------------------------------
  int brake(void)
  {
    short cmd;
    #ifdef PROGRAM
    printf("running brake\n");
    #endif// PROGRAM

    left_brake_status = checkLeftBrake();
    right_brake_status = checkRightBrake();
    #ifdef PROGRAM
    printf("starting left brake  = %d\n", left_brake_status);
    printf("starting right brake = %d\n", right_brake_status);
    #endif// PROGRAM

    // set brake motor direction
    BitWrPortI(PGDR, &PGDRShadow, BRAKES_LOCK,
PORT_LEFT_BRAKE_DIRECTION);
    BitWrPortI(PGDR, &PGDRShadow, BRAKES_LOCK,
PORT_RIGHT_BRAKE_DIRECTION);

    while ( !( (left_brake_status == 1) && (right_brake_status == 1) ) )
    {
      tcp_tick(&tcpsock);
      if ( left_brake_status == 1 ) // if lock switch pressed turn off
brake
      {
        BitWrPortI(PGDR, &PGDRShadow, BRAKES_OFF,
PORT_LEFT_BRAKE_CONTROL);
        #ifndef PROGRAM
        BitWrPortI(PCDR, &PCDRShadow, LED_ON, PORT_LED_4);
        #endif// NOT PROGRAM
      }
      else // if brakes not locked turn on
      {
        BitWrPortI(PGDR, &PGDRShadow, BRAKES_ON,
PORT_LEFT_BRAKE_CONTROL);
        #ifndef PROGRAM
        BitWrPortI(PCDR, &PCDRShadow, LED_OFF, PORT_LED_4);
        #endif// NOT PROGRAM
      }

      if ( right_brake_status == 1 ) // if lock switch press turn off
brake
      {
        BitWrPortI(PGDR, &PGDRShadow, BRAKES_OFF,
PORT_RIGHT_BRAKE_CONTROL);
        #ifndef PROGRAM
        BitWrPortI(PCDR, &PCDRShadow, LED_ON, PORT_LED_6);
        #endif// NOT PROGRAM
      }
      else // if brake not locked turn on
      {
        BitWrPortI(PGDR, &PGDRShadow, BRAKES_ON,
PORT_RIGHT_BRAKE_CONTROL);
        #ifndef PROGRAM
        BitWrPortI(PCDR, &PCDRShadow, LED_OFF, PORT_LED_6);
        #endif// NOT PROGRAM
      }
      left_brake_status = checkLeftBrake();
      right_brake_status = checkRightBrake();
    } // end while ( braking )
```

```
    stop(); // needed because problem with dumping commands
    // Send message "Brake On"
    cmd = TRM_BRAKES_ON;
    sock_awrite(&tcpsock, (char *)&cmd, CMD_LENGTH);
    #ifdef PROGRAM
    printf("ending left brake  = %d\n", left_brake_status);
    printf("ending right brake = %d\n", right_brake_status);
    #endif// PROGRAM
  } // end brake()

//-------------------------------------------------------
// unbrake(): disengage brakes
//-------------------------------------------------------
  int unbrake(void)
  {
    short cmd;
    #ifdef PROGRAM
    printf("running unbrake\n");
    #endif// PROGRAM

    left_brake_status = checkLeftBrake();
    right_brake_status = checkRightBrake();
    #ifdef PROGRAM
    printf("starting left brake  = %d\n", left_brake_status);
    printf("starting right brake = %d\n", right_brake_status);
    #endif// PROGRAM

    // set brake motor direction
    BitWrPortI(PGDR, &PGDRShadow, BRAKES_UNLOCK,
PORT_LEFT_BRAKE_DIRECTION);
    BitWrPortI(PGDR, &PGDRShadow, BRAKES_UNLOCK,
PORT_RIGHT_BRAKE_DIRECTION);

    while ( !( (left_brake_status == -1) && (right_brake_status == -1) )
)
    {
      tcp_tick(&tcpsock);
      if ( left_brake_status == -1 ) // if unlock switch pressed turn
off brake
      {
        BitWrPortI(PGDR, &PGDRShadow, BRAKES_OFF,
PORT_LEFT_BRAKE_CONTROL);
        #ifndef PROGRAM
        BitWrPortI(PCDR, &PCDRShadow, LED_ON, PORT_LED_5);
        #endif// NOT PROGRAM
      }
      else // when not unlocked turn on
      {
        BitWrPortI(PGDR, &PGDRShadow, BRAKES_ON,
PORT_LEFT_BRAKE_CONTROL);
        #ifndef PROGRAM
        BitWrPortI(PCDR, &PCDRShadow, LED_OFF, PORT_LED_5);
        #endif// NOT PROGRAM
      }
      if ( right_brake_status == -1 ) // if unlock switch pressed turn
off brake
      {
        BitWrPortI(PGDR, &PGDRShadow, BRAKES_OFF,
PORT_RIGHT_BRAKE_CONTROL);
        #ifndef PROGRAM
        BitWrPortI(PCDR, &PCDRShadow, LED_ON, PORT_LED_7);
        #endif// NOT PROGRAM
      }
      else // when not unlocked turn on
```

```
      {
        BitWrPortI(PGDR, &PGDRShadow, BRAKES_ON,
PORT_RIGHT_BRAKE_CONTROL);
        #ifndef PROGRAM
        BitWrPortI(PCDR, &PCDRShadow, LED_OFF, PORT_LED_7);
        #endif// NOT PROGRAM
      }
      left_brake_status = checkLeftBrake();
      right_brake_status = checkRightBrake();
    } // end while ( braking )

    stop(); // needed because problem with dumping commands
    // Send message "Brake Off"
    cmd = TRM_BRAKES_OFF;
    sock_awrite(&tcpsock, (char *)&cmd, CMD_LENGTH);
    #ifdef PROGRAM
    printf("ending left brake  = %d\n", left_brake_status);
    printf("ending right brake = %d\n", right_brake_status);
    #endif// PROGRAM
  } // end unbrake()

//---------------------------------------------------------
// checkLeftBrake(): check left brake status
//   return value: -> locked = 1, unlocked = -1, & unknow = 0
//---------------------------------------------------------
  int checkLeftBrake(void)
  {
    // read switch 1 left brake -> locked
    if ( !BitRdPortI(PFDR, PORT_LEFT_BRAKE_SWITCH_LOCK))
    {
      #ifdef PROGRAM
      printf("Switch 0 depressed.\n");
      #endif// PROGRAM
      return 1;
    }
    // read switch 2 left brake -> unlocked
    else if ( !BitRdPortI(PFDR, PORT_LEFT_BRAKE_SWITCH_UNLOCK))
    {
      #ifdef PROGRAM
      printf("Switch 1 depressed.\n");
      #endif// PROGRAM
      return -1;
    }
    else // neither switch pressed
      return 0;
  } // end checkLeftBrake()

//---------------------------------------------------------
// checkRightBrake(): check Right brake status
//   return value: -> locked = 1, unlocked = -1, & unknow = 0
//---------------------------------------------------------
  int checkRightBrake(void)
  {
    // read switch 3 right brake -> locked
    if ( !BitRdPortI(PFDR, PORT_RIGHT_BRAKE_SWITCH_LOCK))
    {
      #ifdef PROGRAM
      printf("Switch 2 depressed.\n");
      #endif// PROGRAM
      return 1;
    }
    // read switch 4 right brake -> unlocked
    else if ( !BitRdPortI(PFDR, PORT_RIGHT_BRAKE_SWITCH_UNLOCK))
    {
```

```
      #ifdef PROGRAM
      printf("Switch 3 depressed.\n");
      #endif// PROGRAM
      return -1;
    }
    else // neither switch pressed
      return 0;
  } // end checkRightBrake()

//----------------------------------------------------------
// checkBrakes(): check brake status
//                only for testing
//----------------------------------------------------------
  int checkBrakes(void)
  {
    if ( !BitRdPortI(PFDR, 0)) // read switch 1
      printf("Switch 1 depressed.\n");
    if ( !BitRdPortI(PFDR, 1)) // read switch 2
      printf("Switch 2 depressed.\n");
    if ( !BitRdPortI(PFDR, 2)) // read switch 3
      printf("Switch 3 depressed.\n");
    if ( !BitRdPortI(PFDR, 3)) // read switch 4
      printf("Switch 4 depressed.\n");
  } // end checkBrakes()

//----------------------------------------------------------
// speedUp(): increase drive speed -> increase PWM duty cycle
//----------------------------------------------------------
  int speedUp(void)
  {
    short cmd;
    #ifdef PROGRAM
    printf("running speedUp\n");
    #endif// PROGRAM
    if ( duty_cycle < 100 )
    {
      duty_cycle += 10;
      #ifdef PROGRAM
      printf("Current drive speed = %d%%\n", duty_cycle);
      printf("Calc'ed duty cycle = %d\n", (int)(((100-
duty_cycle)*0.01)*1024));
      #endif// PROGRAM
    }
    else if ( duty_cycle == 100 )
    {
      #ifdef PROGRAM
      printf("Speed at maximum -> %d%%\n", duty_cycle);
      printf("Calc'ed duty cycle = %d\n", (int)(((100-
duty_cycle)*0.01)*1024));
      #endif// PROGRAM
    }

    cmd = TRM_SPEED;
    sock_awrite(&tcpsock, (char *)&cmd, CMD_LENGTH);
    cmd = (short)duty_cycle;
    sock_awrite(&tcpsock, (char *)&cmd, CMD_LENGTH);
  } // end speedUp()

//----------------------------------------------------------
// speedDown(): decrease drive speed -> decrease PWM duty cycle
//----------------------------------------------------------
  int speedDown(void)
  {
    short cmd;
```

```
    #ifdef PROGRAM
    printf("running speedDown\n");
    #endif// PROGRAM
    if ( duty_cycle > 0 )
    {
      duty_cycle -= 10;
      #ifdef PROGRAM
      printf("Current drive speed = %d%%\n", duty_cycle);
      printf("Calc'ed duty cycle = %d\n", (int)((100-
duty_cycle)*10.24));
      #endif// PROGRAM
    }
    else if ( duty_cycle == 0 )
    {
      #ifdef PROGRAM
      printf("Speed at minimum -> %d%%\n", duty_cycle);
      printf("Calc'ed duty cycle = %d\n", (int)((100-
duty_cycle)*10.24));
      #endif// PROGRAM
    }

    cmd = TRM_SPEED;
    sock_awrite(&tcpsock, (char *)&cmd, CMD_LENGTH);
    cmd = (short)duty_cycle;
    sock_awrite(&tcpsock, (char *)&cmd, CMD_LENGTH);
  } // end speedDown()

//----------------------------------------------------------
// enableDetector(): turn on power to detector
//----------------------------------------------------------
  int enableDetector(void)
  {
    short cmd;
    #ifdef PROGRAM
    printf("running enableDetector\n");
    #endif// PROGRAM
    // turn on detector power relay
    BitWrPortI(PGDR, &PGDRShadow, DETECTOR_ON, PORT_DETECTOR_POWER);
    BitWrPortI(PFDR, &PFDRShadow, LED_ON, PORT_LED_3);
    cmd = TRM_DET_ENABLE;
    sock_awrite(&tcpsock, (char *)&cmd, CMD_LENGTH);
  } // end enableDetector()

//----------------------------------------------------------
// disableDetector(): turn off power to detector
//----------------------------------------------------------
  int disableDetector(void)
  {
    short cmd;
    #ifdef PROGRAM
    printf("running disableDetector\n");
    #endif// PROGRAM
    // turn off detector power relay
    BitWrPortI(PGDR, &PGDRShadow, DETECTOR_OFF, PORT_DETECTOR_POWER);
    BitWrPortI(PFDR, &PFDRShadow, LED_OFF, PORT_LED_3);
    cmd = TRM_DET_DISABLE;
    sock_awrite(&tcpsock, (char *)&cmd, CMD_LENGTH);
  } // end disableDetector()

//----------------------------------------------------------
// trSleep():
//   Function will take up 100*hundMS milliseconds.
//----------------------------------------------------------
  void trSleep(unsigned int hundMS)
```

```
  {
     unsigned int i;
     unsigned int j;
     unsigned int times;

     times = 35500;

     for(j=0;j<hundMS;j++)
     {
       for(i=0;i<times;i++);
     }

     return;
  } // end trSleep()

//----------------------------------------------------------
// trMSleep():
//   Function will take up ms milliseconds.
//----------------------------------------------------------
  void trMSleep(unsigned int ms)
  {
     unsigned int i;
     unsigned int j;

     for(j=0;j<ms;j++)
     {
       for(i=0;i<500;i++);
     }

     return;
  } // end trMSleep()

//----------------------------------------------------------
// ramp():     used to start/stop drive motor slowly
//   time:     delay time between ramp steps in ms
//   interval: amount to jump pwm each step, from 0 - 1024
//   dir:      sets ramp direction to ramp up or down
//             1 -> ramp speed up; 0 -> ramp speed down
//----------------------------------------------------------
  void ramp(unsigned int time, unsigned int interval, char dir)
  {
    int i;
    unsigned int maxspeed;

    maxspeed = (unsigned int)((100 - duty_cycle)*10.24);

    if(dir)
    {
      for(i=1024; i>maxspeed; i-=interval)
      {
        trMSleep(time);
        pwm_set(PWM_CHANNEL, i, PWM_OPTION); // ramp drive up
      }
    }
    else
    {
      for(i=maxspeed; i<1024; i+=interval)
      {
        trMSleep(time);
        pwm_set(PWM_CHANNEL, i, PWM_OPTION); // ramp drive down
      }
    }

    return;
```

```
  } //end ramp()

//--------------------------------------------------------
// clearRecBuffer(): reads commands waiting on input buffer
//  used to get rid of commands set by server while the
//  trolley is in the middle of another command
//  otherwise all commands will be stored and excuted
//--------------------------------------------------------
  void clearRecBuffer(void)
  {
    while ( sock_dataready(&tcpsock) != 0 )
    {
       tcp_tick(&tcpsock);
       if ( (sock_aread (&tcpsock, NULL, CMD_LENGTH)) == 2 )
       {
          #ifdef PROGRAM
          printf("\nDumped one command\n");
          #endif// PROGRAM
       }

       // blink when dumping commands
       BitWrPortI(PFDR, &PFDRShadow, LED_ON, PORT_LED_3); // LED on
       trMSleep(25);
       BitWrPortI(PFDR, &PFDRShadow, LED_OFF, PORT_LED_3); // LED off
       trMSleep(25);

       // read Detector Power status and reset LED
       if ( (BitRdPortI(PGDR, PORT_DETECTOR_POWER)) == DETECTOR_ON )
       {
          BitWrPortI(PFDR, &PFDRShadow, LED_ON, PORT_LED_3);
       }

    } // end while ( bytes in socket )
  } // end clearRecBuffer()

//--------------------------------------------------------
// portInit() : initializes ports C, F, & G
//--------------------------------------------------------
  int portInit(void)
  {
    // Initialize C Port: LEDs -> C6 & C7 used for serial debug
    //   Bit  I/O  Usage
    //   ---  ---  -------------------------
    //     0   o   LED 1
    //     1   i   input not used
    //     2   o   LED 2
    //     3   i   input not used
    //     4   o   LED 3
    //     5   i   input not used
    //     6   o   LED 4
    //     7   i   input not used
    // LEDs use same port as programming cable
    // so disable them when running with cable to allow feedback
    #ifndef PROGRAM
    WrPortI(PCFR,  &PCFRShadow,  0x00);    //set all bits = normal
    WrPortI(PCDR,  &PCDRShadow,  0x00);    //set all bits low
    #endif// NOT PROGRAM

    // Initialize F Port: Brake Switches and Drive Motor PWM
    //   Bit  I/O  Usage
    //   ---  ---  -------------------------
    //     0   i   Brake switch LeftFront  locked   switch 0
    //     1   i   Brake switch LeftFront  unlocked    "   1
    //     2   i   Brake switch RightFront locked      "   2
```

```
    //     3   i   Brake switch RightFront unlocked    "   3
    //     4   o   Drive Motor Control - PWM
    //     5   o   Reserved
    //     6   o   Reserved
    //     7   o   -> Doesn't work - fried at the chip level
    WrPortI(PFCR,  &PFCRShadow,  0x00);    //clear all bits for pclk/2
    WrPortI(PFFR,  &PFFRShadow,  0x10);    //set bit 4 -> PWM, others =
normal
    WrPortI(PFDCR, &PFDCRShadow, 0x00);    //set all bits to drive high-
low
    WrPortI(PFDR,  &PFDRShadow,  0x10);    //set all bits low, except 4
    WrPortI(PFDDR, &PFDDRShadow, 0xF0);    //set bits 0-3 = input, 4-7 =
output

    // Initialize G Port: Motor Control/Direction/Brake and Detector
Power
    //   Bit  I/O  Usage
    //   ---  ---  -------------------------
    //     0   o   Drive Motor Direction
    //     1   o   LeftFront Brake Direction
    //     2   o   LeftFront Brake Control
    //     3   o   RightFront Brake Direction
    //     4   o   RightFront Brake Control
    //     5   o   Detector Power Enable
    //     6   o   Motor Brake
    //     7   o   Reserved
    WrPortI(PGCR,  &PGCRShadow,  0x00);    //clear all bits for pclk/2
    WrPortI(PGFR,  &PGFRShadow,  0x00);    //clear all bits for normal
function
    WrPortI(PGDCR, &PGDCRShadow, 0x00);    //set all bits to drive high-
low
    WrPortI(PGDR,  &PGDRShadow,  0x34);    //set all bits low, except 2,
4, & 5
    WrPortI(PGDDR, &PGDDRShadow, 0xFF);    //set all bits to output

    return 0;
  } // end portInit()
//-------------------------------------------------------
```