TEAM 18 – PENETROMETER

Final Report

Carren Brown[1], Deneuve Brutus[2], Peter Hettmann[3], Sean Kane[4], Natalie Marini[5], Mitchell Robinson[6], Maritza Whittaker[7]

Mechanical and Electrical Engineering Departments
FAMU-FSU College of Engineering
2525 Pottsdamer Street
Tallahassee, Florida United States 32310-6046

[1]ME, carren1.brown@famu.edu

[2]CpE, db10c@my.fsu.edu

[3] ME, pmh10@my.fsu.edu

[4]EE, srk10e@my.fsu.edu

[5] ME,nrm10d@my.fsu.edu

[6]EE, mwr10c@my.fsu.edu

[7] ME, mnw11b@my.fsu.edu

Due Friday, April 10th, 2015

Mr. Mike Russo and the National Park Service: Project Sponsor
Dr. Chiang Shih: Project Mentor/Advisor
Dr. Nikhil Gupta: ME Project Co-Mentor
Dr. Michael Frank: EE Project Coordinator/Instructor
Dr. Linda DeBrunner: Project Mentor/Advisor

TABLE OF CONTENTS

TABLE OF FIGURES

Table of Tables

Group Member Information

Carren Brown is the Team Ambassador. She is an ME student at FAMU, expecting to graduate with a specialty in Mechanics and Materials. She has completed two summer internships at Florida Power & Light and Colgate-Palmolive. She plans to earn a Master's degree in Engineering Management, and then begin a career in manufacturing.

Deneuve is a CpE student at FSU, expecting to graduate in May of 2015. He is currently a .NET intern at Marquis Software, Inc. in Tallahassee, FL. He plans on working as a Software Engineer, while pursuing a Master's of Science in Engineering Management. He is an active Reservist in the United States Navy.

Peter Hettmann is the team’s Treasurer. He is an ME student at FSU, expecting to graduate with a specialty in Dynamics and Mechatronic Design. He has completed two summer internships at Siemens Energy and at Senninger Irrigation. After graduation, he plans to pursue higher education in Computer Science/Engineering with a career in robotics.

Sean Kane is the team’s Lead EE. He is an EE student at FSU. He currently holds a part time internship with RCC Consultants, Inc., who is contracted with the Florida Department of Transportation (FDOT) to work on the public safety communication system. He plans to stay in Tallahassee, FL after he graduates to continue work with the FDOT.

Natalie Marini is the Team Leader. She is an ME student at FSU, expecting to graduate with a specialty in Thermal Fluid Sciences. She has previously completed three summer internships at Siemens Energy, and plans on continuing work in the power industry when she graduates.

Mitchell is an EE student at FSU, expecting to graduate in May of 2015. He plans on using his degree to work in the power industry and is specializing in power generation. Mitchell intends on being part of the collaborative effort in design the smart grid in order to meet higher energy efficiency.

Maritza Whittaker is the Team Secretary and Webmaster. She is an ME student at FSU. She has previously completed two summer internships at exp, Inc. and at Oceaneering Entertainment Systems in Orlando, FL. She hopes to pursue a career in the entertainment industry after graduation.

Acknowledgement

The 2014-2015 Penetrometer group would like to thank the following people for contributing their knowledge, expertise and support to the project.

To Dr. Frank for sharing his knowledge and expertise with the electrical aspect of the design. To Dr. Shih for frequently providing important assessment of the overall design of the project. To Dr. Gupta for providing expertise to ensure the compatibility between the electrical and mechanical aspects of the design. To Dr. DeBrunner for assisting in electrical questions. Finally to Dr. Russo for providing support and feedback to the group to ensure the design of the project meets the needs of the National Park Service.

Abstract

The team was given the task to design and prototype a penetrometer for the National Park Services (NPS). The penetrometer will be used to assist archaeologist in identifying different soil types and in locating midden at their dig sites. Midden is soil that contains domestic waste and artefacts of past human occupation. The penetrometer must be easy to use, portable, weigh less than 50 pounds, and be reliable. The penetrometer will also have the ability to wirelessly transmit data to a handheld Android device. Taking into account the design from last year's team, the requirements and wants from the sponsor, and the research conducted by the team members, the team has come up with a final design for the penetrometer prototype. This prototype will utilize a drop weight similar to last year's design, two load cells in the shaft to obtain the friction coefficient, and a personalized app and DAQ system to obtain the experimental data. To keep the team on schedule, a Gantt chart was developed, as shown in Appendix A. Constant communication as also been kept between the team and the advisor, instructors, and sponsor, in order to seek guidance and have transparency on the project.

1. Introduction

The purpose of this project is to design and construct a penetrometer that can properly differentiate soil types and identify any midden that is present. Current handheld penetrometers are solely used to determine the compaction of the soil. This method is not an exact science, and requires a person with much experience to determine the results. Penetrometers that are able to detect different soil types and midden are far too large to be used efficiently in the field. The goal of Team 18 is to combine these two ideas and develop a handheld penetrometer that has the ability to identify midden by determining the soil's friction coefficient. The team has created a prototype that is lightweight and easy to use in the field. The device is also portable and has the ability to transfer the force from the bottom portion of the shaft to the top housing where the load cells are located. The readings from the load cells will be transmitted wirelessly to a handheld tablet, using the data acquisition module. These readings will allow the team to calculate the friction coefficient of the soil, and therefore determine the type of soil present and if there is any midden in the area.

A penetrometer is a basic force instrument in design and simple in use. However, it cannot be effectively used by a novice for precise results. Originally, a penetrometer was used by agricultural personnel for penetration of the ground soil on several acres of land to determine the soil compaction and how viable the soil will be for crop production. Before a standardized penetrometer, results could vary from farm to farm and with different surveying teams. Depending on the varying level of experience by the surveying team, these results can either be interpreted as good or bad soil results.

As an extension of the 2013-2014 senior design project, it is the objective of Team 18 to redesign a penetrometer which will detect midden levels in the soil present at the Southeast Archaeological Center & National Park Services' field testing site. This penetrometer will have portable and wireless capabilities in order to properly distinguish the type of soil present below the ground. It has been established that the sponsor is looking for a more reliable and easier-to-use system than the prototype designed by the previous senior design project.

The goal statement of Team 18 is as follows: "Design an instrument that can identify midden and differentiate soil types at various depths."

The following objectives were provided to the Team from discussion with the National Park Services and Dr. Russo, the Team's sponsor. It must be able to identify midden levels in remote locations and weigh less than 50 lbs. The penetrometer should wirelessly display results to a handheld device and be very portable to use out in the field.

There have been several of constraints placed on the design. These constraints are as follows: the prototype design must be easy to use by only one person in the field, without assistance, the diameter of the prototype must be small enough for the device to penetrate the ground easily, the material of the prototype must be strong enough for the device to penetrate the ground without fracturing, the prototype design must wirelessly relay reliable data out in the field, making it be portable for the user, and finally the total cost must not exceed $2,000. However, the sponsor is able to expand the budget if it is deemed necessary by the team and the advisor.

2. Project Scope

The following project definition will explain the Team's background research of the project, need statement, goal statements, objectives and constraints of the senior design project.

2.1. Problem Statement

Current handheld penetrometers are used to determine the compaction of the soil being tested. Penetrometers that are able to distinguish between different types of soil are of a significant size and usually brought in on a larger vehicle. The sponsor desires a device that can combine these two concepts: a friction cone penetrometer that is portable, wireless, and easy to use that can differentiate between soil types and detect midden. Midden is soil that contains organic matter and artefacts from past groups of human who occupied that land. The penetrometer will determine the type of soil based on its friction coefficient, which is calculated using forces measured by the load cells.

2.2 Design Requirements

The goal of Team 18 is to redesign the penetrometer from the 2013-2014 senior design team who started on this endeavour. The penetrometer needs to be lightweight, portable, and easy to use. To lighten the weight, the penetrometer will be smaller in diameter, which will also allow for easier penetration into the ground. The penetrometer itself will be transported in only two parts. There will be no exposed wires, due to the use of load cells and technology that has Bluetooth capabilities. The load cells will be housed at the top to lessen the amount of debris and impact they are exposed to. All the electrical components will be in a separate housing that is also lightweight and easy to carry. The penetrometer device is simple to use, and can be operated by 1-2 people. The data will be sent to a handheld device using the Bluetooth capabilities of the data acquisition module (DAQ). The app created on the handheld device has been made user-friendly.

2.3 Objectives

As previously stated, Team 18 needs to design an instrument that can differentiate between soil types and identify midden at various depths. This device also has to be lightweight, portable, and easy to use. The portability requirement also includes the device being wireless. The device should take no more than two people to use, but should be able to be used by one with no issues. The app on the handheld tablet should be user-friendly. The material used for the device must also be strong in compression. Below is a short list of the major objectives for the project.

- Must be able to identify midden levels in remote locations.
- Must weigh less than 50 pounds.
- Should wirelessly display results to a handheld device.
- Device should be very portable.
- Weight should be minimized.

2.4 Background Research

A penetrometer is a basic force instrument in design and simple in use. However, it cannot be effectively used by a novice for precise results. Originally, a penetrometer was used by agricultural personnel for penetration of the ground soil on several acres of land to determine the soil compaction and how viable the soil will be for crop production. Before a standardized penetrometer, results could vary from farm to farm and with different surveying teams. Depending on the varying level of experience by the surveying team, these results can either be interpreted as good or bad soil results. To account for this inexperience during surveying of the ground, calculations will be used to be unbiased in the testing of the soil composition and compaction before any ground comparisons need to be done via a computer.

The standard design of a penetrometer was adopted by the American Society of Agricultural Engineers in 1999 and with this standard design the comparison of data across a wide range of locations could be compared and used for soil compaction. This design calls for a 30 degree cone angle and the use of a 1/2 inch or 3/4 inch base cone as seen in Figure 1. These dimensions more closely resemble a root growing and penetrating the ground as it grows and with certain ground compaction can yield higher or lower crop turn out.[1]

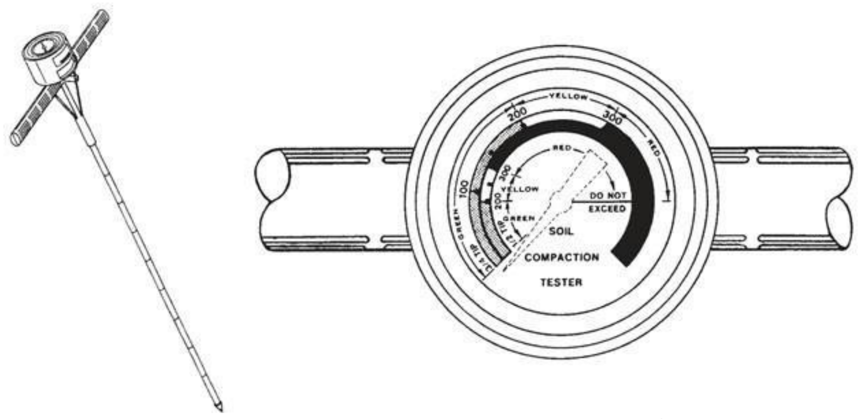


Figure 1. Standardized Penetrometer Design[2]

In the field of archaeology, soil compaction and composition can save a lot of time and money from large excavation digging to uncover important soil types shallow or deep underneath the top soil. A penetrometer is being used to detect the location of midden, which is archaeological soil type produced from decomposed artefacts that were tossed into the environment during the time of population in that certain location. The used method to determine the midden is a basic T-bar penetrometer that has several extendable rods that can allow for several meters of distance to map the location and depth of midden. When used by an experienced surveying team, the midden can be located based on the “feel” of the midden soil type as the compaction and compression is different than the surrounding soil types. This feel can be misinterpreted by an inexperienced surveyor and the data collected could be wrong. To account for this inexperience, load cells can be used along with a computer program to determine the depth and soil types.

One method closely related to our approach on the penetrometer is the cone penetrometer test (CPT) which incorporates an electronic friction cone and piezocone penetrometer as seen in Figure 2. When used to test the soil composition and compaction, a computer logs the values from the cone and friction sleeve and uses the ratio to determine if the soil is suitable for use. Using this same concept of separating load cells to determine the friction ratio, archaeological dirt can be determined several meters under the topsoil without digging several holes. The surveying team using the device with not need a high level of experience as the data collected will be based on calculated values to determine the actual soil that is being penetrated.[3]

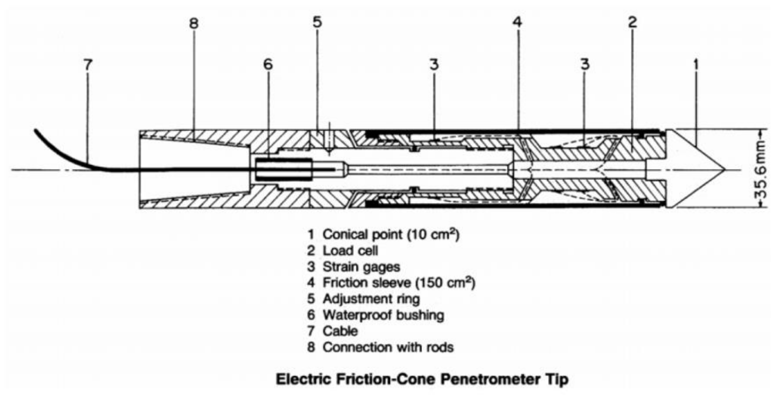


Figure 2. Electrical Components of the Penetrometer Tip[3]

3. Evolution of Project

This section of the report discusses the evolution of design of the friction cone penetrometer as well as how decisions were made choosing the final design.

Over the last two semesters, Team 18 has made drastic changes to the overall design of the friction cone penetrometer. At the beginning of the Fall 2014 semester, the team was fortunate enough to use last year's prototype for testing purposes alongside Dr. Russo and his NPS team. While out in the field, the team had the opportunity of learning, in great detail, the purpose behind the project and why it will be a necessity to the NPS team. It was discussed that last year's prototype failed and would need to be redesigned in order to be a success. Testing last year's prototype allowed the team to learn what was necessary for the project's design, and more importantly, what not to do with their prototype. After testing, it was noted that it took far too many people to man the penetrometer out in the field. It took at least four people to set the device up, a table to place the electrical components as well as a twenty-pound generator. Thus, one of the main objectives of the project was to narrow down the man power of using this device in half and make it lightweight, preferably less than fifty pounds. Another problem the team realized they would need to improve would be the seals used at the end of the penetrometer to protect itself from debris when used out in the densely forested testing sites. Last year, there was only a layer of metal epoxy to protect the penetrometer out in the field. It not only was ineffective, it also failed when testing the prototype. Another requirement that the team took into consideration from testing with the NPS team was that overall diameter of the penetrometer would need to be reduced. Last year's prototype was rather large, at one-and-a-half-inch diameter, this made probing the penetrometer into the ground difficult. The smaller the shaft size of the penetrometer, the easier it will go through the ground, making it less tedious of a process for the user. This experience allowed the team to deduce the following that the redesigned penetrometer must: be lightweight, where a maximum of two people are needed to man the penetrometer; be reliable and not fail when out in the field; have seals that will protect it from the elements; have a smaller diameter to more easily probe the ground.

After gathering the crucial information from both the sponsor and the experience of testing out in the field, Team 18 began the designing process of the friction cone penetrometer. It was noted that a lot of alterations to last year's design was going to be necessary in order to make this prototype a success. In the Fall 2014 semester, Team 18 produced four design possibilities for the friction cone penetrometer. In mechanical design A, as seen below, a drop weight at the top of the penetrometer would be used to force the device consistently into the ground. Two load cells are placed at the bottom of the penetrometer just above the friction cone tip to obtain the voltage readings. The load cells are placed inside of a friction sleeve; this allows a fictional force to be read, which is used to calculate the friction coefficient of the soil. To improve last year's design, changes need to be made mainly to the shaft, the load cell design, and the portability. The shaft of the penetrometer needs to be strong under repetitive compressive forces, but last year's design fractured multiple times in the field while in use. The compressive strength of the shaft needs to be increased, which can be done by choosing a stronger material, such as titanium, or by adding ceramic fibers. Ceramics are stronger under compressive loads than most metals, therefore in ceramic fibers were added into the metal shaft, the overall yield strength would increase. The load cells used for the prototype last year were large in size, forcing the shaft diameter to increase. Using smaller load cells would allow for a thinner penetrometer, which would permit easier entry into the ground. The wiring of the load cells was not housed, exposing it to any surrounding elements. The wires should be housed inside the penetrometer shaft or in a secure box at the top of the penetrometer.

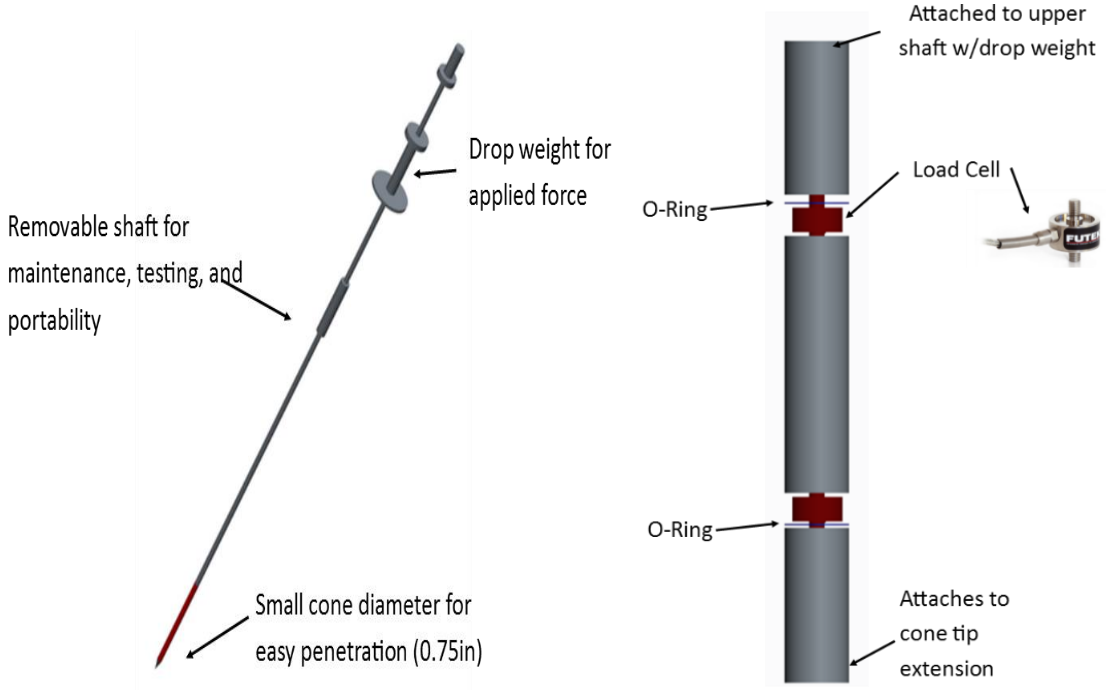

Removable shaft for maintenance, testing, and portability
Drop weight for applied force
Small cone diameter for easy penetration (0.75in)
Attached to upper shaft w/drop weight
O-Ring
Load Cell
O-Ring
Attaches to cone tip extension

Figure 3. Fall 2014 Mechanical Design A

A second mechanical design was produced that was very much similar to mechanical design A, as seen in Figure 3. However, the main and very important difference between these two models is the actual location of the load cells. In design A, the cells are at the bottom of the shaft and have a direct impact with the soil. In design B, the load cells are at the top of the shaft. This makes it much easier to keep the load cells weather resistant and it enables a larger sized load cell without having a large shaft diameter. Testing will have to be done with material choices to ensure the load from the bottom of the shaft can accurately be transferred to the load cells at the top of the shaft. Another modification to this design is the housing shown in blue. Since the model is to be wireless and battery operated, it would make sense to have a separate housing from the actual shaft itself.

A third and fourth design, seen in Figure 4, were proposed that seemed to be very much similar to one another, yet it differed from design A and B because it utilized strain gauges, instead of load cells, for the data it was receive from the applied standard load of the drop weight. The top compartment will receive the load applied from the drop weight and as the force is transferred through the rod the secondary load cell will be placed directly above the penetration cone allowing for less forces to be lost from the transfer of the force from the ground. The difference between both designs lies in the actual placement of the strain gauges and the housing of the strain gauges that will be receiving the impact force. For design C, the strain gauges will be set up in a vertical orientation along a material specimen that will experience a deformation in the horizontal direction and for design D, the strain gauges will be set up in a horizontal orientation and the load applied will create a deflection of the material specimen in the vertical direction. Both of these concepts will be explored more deeply for sensitivity levels and accurate transfer of the applied load.

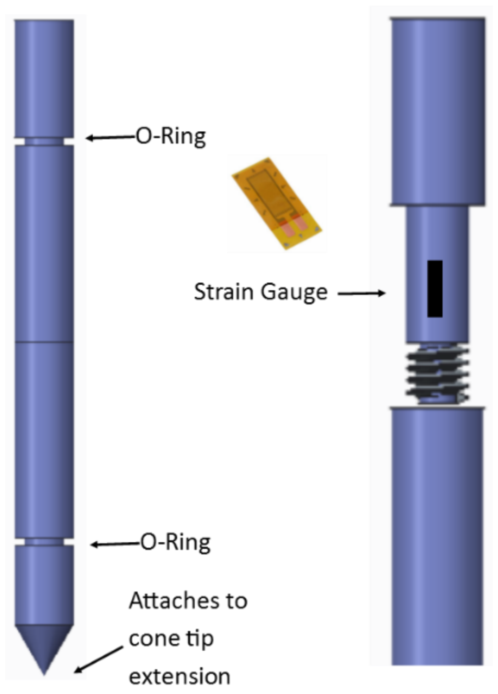


Figure 4. Mechanical Designs C and D

At the end of the Fall 2014 semester, a decision matrix was implemented in order to choose the overall design of the friction cone penetrometer, which can be seen in Table 1. The mechanical design criteria for the selection of a final design consists of six main categories based on the project objectives and goals developed earlier. The six categories, in order of descending weight, are: portability, ease of use, weight, measurability, durability, and cost.

- **Portability**: Portability is the top priority when designing the penetrometer. The device will be used continuously for 8-9 hours, and the user will be moving across the work site to test multiple areas. If the device cannot be transported easily, it is of no use. It should not take more than two people to transport the device, and the device should not have to be transported as many separate parts.
- **Ease of Use**: The device must be able to be operated by 1-2 people while in the field. The setup, use, and breakdown of the penetrometer must be simple and quick to allow for more time to test holes at the work site with little to no complications.
- **Weight**: The weight of the mechanism must be light enough to be carried to and from the work site, and transported across the work site continuously. The goal is to construct a device that weighs no more than 50 pounds.
- **Measurability**: The purpose of using this device over the current method is to remove any bias that may come from the user of the penetrometer. Therefore, the device must deliver reliable data and results.
- **Durability**: The mechanism must be extremely durable because the user will not be able to make any major repairs in the field. The shaft, friction cone tip, and handle should not crack or fracture at any time during use.
- **Cost**: The cost is of the lowest weight because our sponsor has made clear that the top priority is to construct a feasible prototype. While we are taken our given budget into heavy consideration, our sponsor has informed us that if we do need more funding to purchase materials of a higher quality, he will be willing to consider increasing the budget.

The Electrical design criteria for the selection of a final design consists of five main categories based on the project objectives and goals developed earlier. The five categories, in order of descending weight, are: ease of use, portability/wireless, durability, and cost.

- **Ease of Use**: The application developed to display real-time results on an android device must be able to display results without any configuration by the user.
- **Portability/Wireless**: The connection between the android device and the data acquisition must not impede efficient work in the field because the users need to be able to move from hole to hole with ease during an 8 hour period. It is imperative that the user can carry all of the equipment with very few wires so that the user does not have to spend time or energy untangling wires.
- **Durability**: The android device and the data acquisition must be able to withstand typical weather conditions and possible contact to dirt. The user should not have to worry about the data acquisition or android device failing because of typical weather conditions in Florida.
- **Cost**: The price of the data acquisition system and the android device should not exceed the amount of money that the sponsor is willing to spend.

Using the design matrix with the chosen criteria, the best design concept is design D, which utilizes strain gauges mounted vertically on the penetration shaft. This design had the highest score, or tied for the highest score, in five out of the six categories. It scored low in the measurability section, but we will look into ways to improve the reliability of the data gathered when using this design. Design D tied with design C, which is the alternative strain gauge design, on five out of the six categories because there were only a few minor differences between the two designs. The major difference was the alignment of the strain gauges within the penetration shaft; the alignment is the cause of the drastic difference between the scores of the two designs in the durability section. When the strain gauges are loaded vertically on the shaft, they are able to withstand a greater load. When all the criteria are combined, design D had the highest score, making it the best choice to consider for our final mechanical design concept.

Table 1. Decision Matrix for Fall 2014 Designs

		Portability		**Ease of Use**		**Weight**		**Measurability**		**Durability**		**Cost**		**Total**
	Weight (%)	0.30		0.25		0.15		0.15		0.10		0.05		1.00
Designs	A	Score	Total	Score	Total	Score	Total	Score	Total	Score	Total	Score	Total	4.95
		4	1.2	6	1.5	2	0.3	8	1.2	5	0.5	5	0.25	
	B	Score	Total	Score	Total	Score	Total	Score	Total	Score	Total	Score	Total	5.7
		5	1.5	6	1.5	7	1.05	5	0.75	6	0.6	6	0.3	
	C	Score	Total	Score	Total	Score	Total	Score	Total	Score	Total	Score	Total	6.25
		5	1.5	8	2	8	1.2	6	0.9	3	0.3	7	0.35	
	D	Score	Total	Score	Total	Score	Total	Score	Total	Score	Total	Score	Total	6.65
		5	1.5	8	2	8	1.2	6	0.9	7	0.7	7	0.35	

As the team approached the new Spring 2015 semester, a lot of changes were going to be necessary in order for the penetrometer to be successful. The team was able to meet with their advisor and sponsor, and it was found that none of the previously mentioned designs would be used in order for the prototype to be effective. Therefore, the team met with advisor, Dr. Shih, who advised that the strain gages would no longer be of use in the penetrometer. New designs were pitched, until the final design was approved by Dr. Shih. Changes were constantly being made throughout the design process, the main one being the switch from strain gauges to load cells, as well as their placement on the penetrometer. It was chosen to bring the load cells to the top of the penetrometer, where they sit inside of a housing component that protect it from both the natural elements as well as from being destroyed by the constant force it feels from the drop weight as it digs the penetrometer deeper into the ground. Another change that was made in the final design of the penetrometer was the shaft diameter. In the fall semester, it was designed to be too thick, it was asked by sponsor, Dr. Russo, to bring the shaft down as small as possible in order to easily penetrate within the ground. The shaft was re-dimensioned to be a slightly less than one inch, with a standard cone tip size attached to the end. After speaking with Dr. Russo, it was found that a friction sleeve two-inches in length would be long enough to detect the midden levels as they are usually in one and a half inch layers within the soil. The housing that keeps the load cells safe went through a lot of redesign; this was due to because of the inability to be machined at the school's machine shop. It was redesigned to use a series of disks that stack on one another in order to keep the load cells safe. The team chose stainless steel as their material for the design as it was cheap and lightweight yet sturdy enough to take the fifteen pound load that will be constantly dropped on the device. Further details into the final design can be seen in section four of this report.

4. Final Design

The following section will give further details on the final design chosen for the friction cone penetrometer and its components.

4.1 Design Choice

The final design chosen by team 18 consists of three separate rods extending the length of the penetrometer that will distinguish the sliding friction force and the impact force felt from the soil during impact. The final design is composed of three sections, each described in more detail below: the impact or driving force component, the housing components and the shaft design.

The impact or driving force used for the final design will be to implement a drop weight impact force design. Figure 5 shows that the drop weight will be located at the top most part of the penetrometer and the weight will either be a 10 lb or 25 lb cylindrical designed weight that will be securely fastened into the housing. Once secured, the weight will be lifted along the guiding shaft and when ready the weight will be released and impact the top of the housing of the penetrometer causing the shafts to penetrate the ground with a constant force. The reason for using the drop weight shaft design is due to the consistency of the drop weight itself. The weight is never changing, either 10 lb or 25 lb, and will be lifted and dropped from the same distance ever occurrence with only a minimal change.

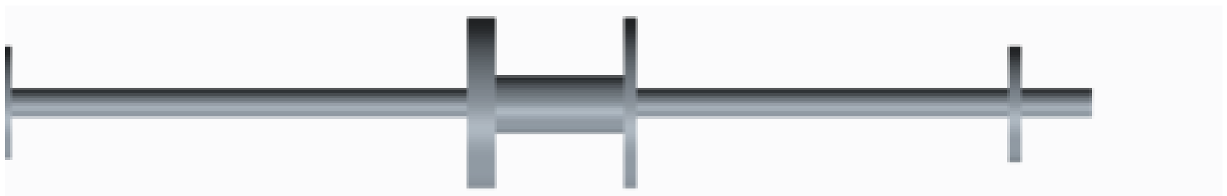


Figure 5. Drop Weight Model Design

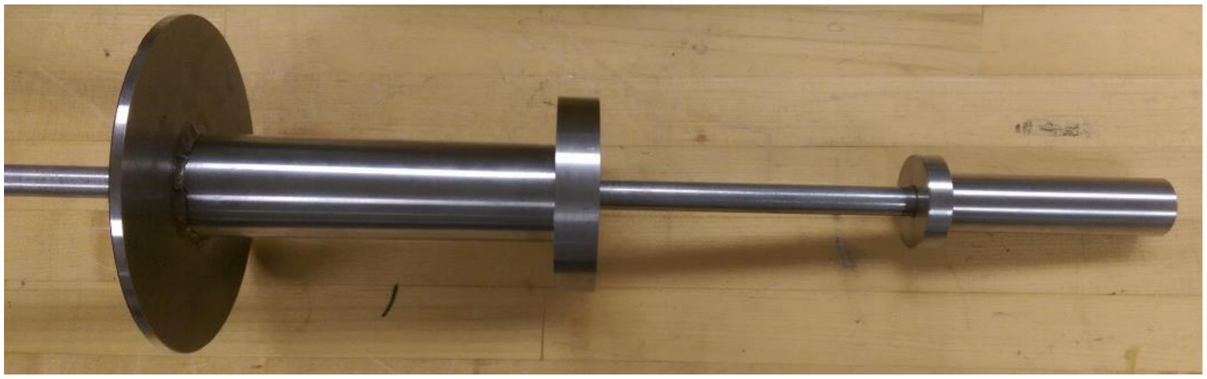


Figure 6. Drop Weight Component

The housing components is the middle section of the penetrometer. The housing will securely hold both the donut and button load cells and protect these parts from the harshness fo the environment while testing. The housing is comprised of eight circular discs that are usd as place holders, guides for the rods and securing discs for the load cells. These disc are stacked vertically and tightly fitted into the shell that goes around all of the discs. Figure 7 shows the model of the enclosed housing and the exploded view will be talked about later.

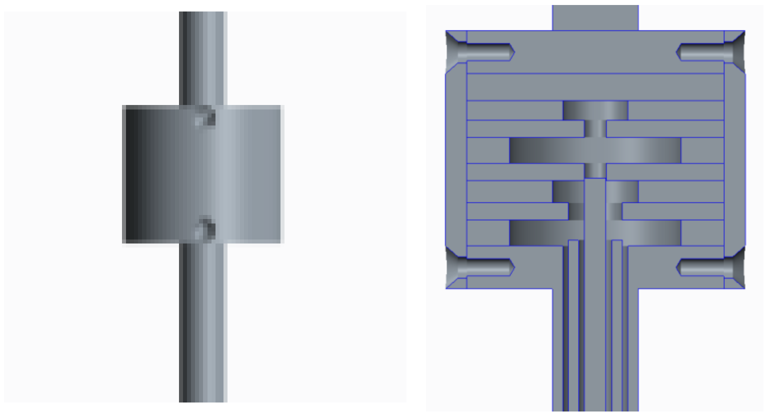


Figure 7. Housing Model

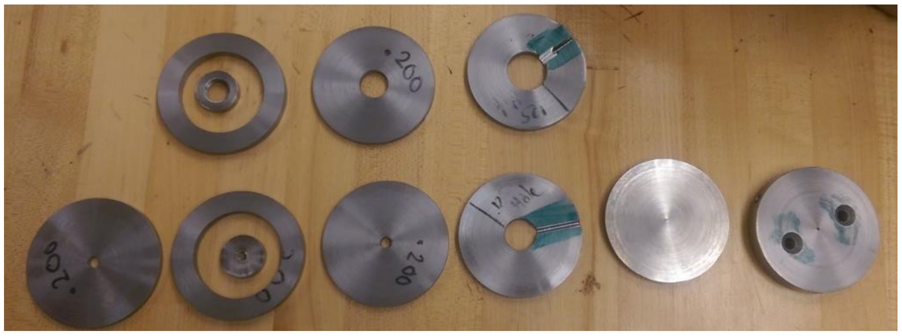


Figure 8. Housing Discs

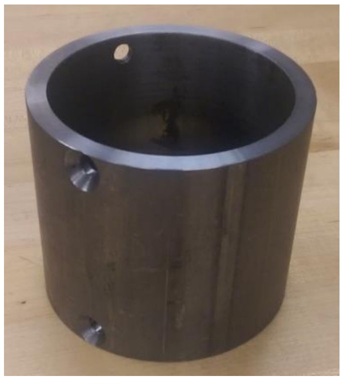


Figure 9. Housing Shell

The third and final section of the design is the shaft components. The shaft is comprised of three different parts: the outer rod, the friction sleeve rod and the cone tip rod. The cone tip is a standard dimension used for a wide variety of penetrometers that will feel the large majority of the force while it is forced through the ground by the drop weight above, this cone tip is connected to the center most shaft, the cone tip shaft, and the force felt by the cone tip will be measured by the top most button load cell. The next component of the shaft is the friction sleeve. The friction sleeve is a 2.5 inch long outer sleeve that will feel the force of the soil as the penetrometer is forced through the ground. The friction sleeve will receive a very small force compared to that of the cone tip due to the only force acting on the sleeve will be the friction coefficient of the soil from rubbing on the sides. The friction rod is the second layer of rods that run the length of the penetrometer and this force will be measured by the donut load cell which is located below the button load cell.

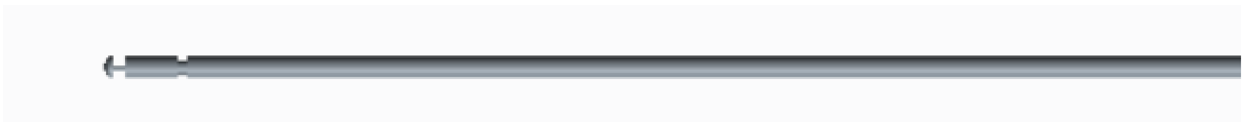


Figure 10. Shaft Design Model

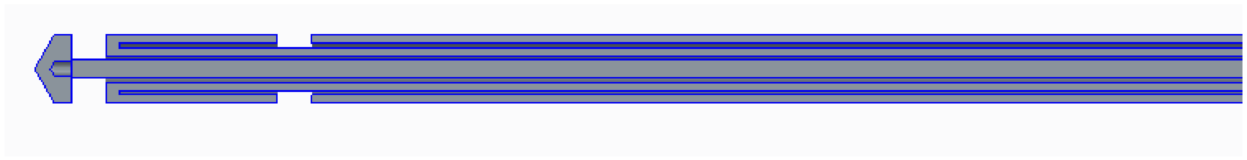


Figure 11. Shaft Design Model Section View

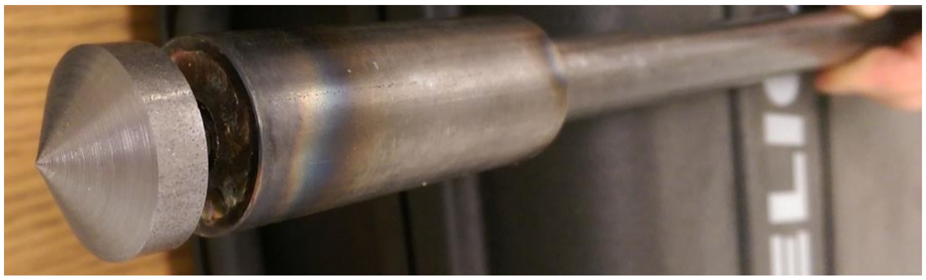


Figure 12. Shaft Design Component

The load cells chosen were a donut load cell, Futek FSH00297, and the button load cell, Futek FSH01050. The donut load cell has a rated max force reading of 100 lbs and the button load cell has a max force reading of 250 lbs. This difference in the max reading is due to the geometry of the surface that each force will be coming from. The donut load cell will only be measuring the sliding frictional force from the soil on the friction sleeve while the button load cell will be reading the main impact on the cone tip penetrating the soil. Each load cell will be securely placed in the housing in contact with the specific rod from each force.

The block diagram in Figure 13 is the electrical design. The load cells will be powered by a 22.2 V rechargeable battery that can be replaced in the field if the battery dies. A 15 V voltage regulator is connected to the battery to ensure that the specified 15 V is supplied to the load cells. An amplifier manufactured by the load cell company, Futek, will amplify the load cell analog output to a +/- 5 V range, so that it can be within the resolution of the wireless DAQ. The wireless DAQ is powered by two rechargeable AA batteries and will directly record the output voltage of the load cells from the amplifier. The battery, amplifier, and wireless DAQ will be placed in an electrical housing to protect from the elements of nature such as water, dirt, etc. The wireless DAQ will then send the data via Bluetooth to an Android tablet running an application, or app, to be developed by the team. The app will display real time results and store the data for further analysis. The laser range finder also runs on two AA batteries, and will record the depth that the penetrometer travels into the soil. This data is sent through Bluetooth to the same app. Once the Android device is paired with the laser range finder and DAQ, it will notify the user that it is ready to start recording data. The data is displayed on the app and generates a file to be saved for further analysis.

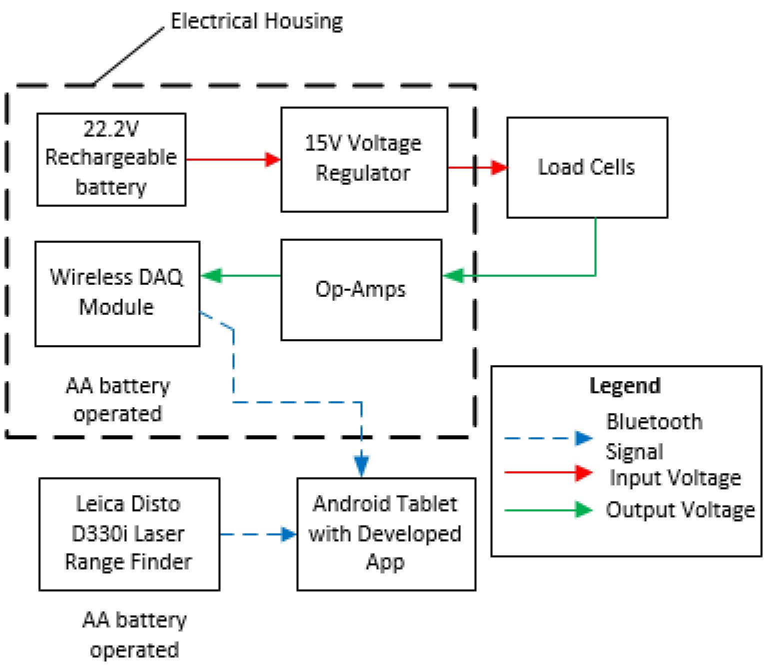


Figure 13. Electrical Block Diagram

4.2 Important Components

The most important component dimensions necessary in designing the penetrometer were the friction sleeve length and the cone tip size. The friction sleeve is of the utmost importance in relation with the reliability of the data found from penetrometer testing. If the sleeve is too long, it will measure the friction of several different layers of soil which could skew the readings too much to be able to identify the correct soil type. If it is too short in length, then the friction reading will be too small to analyse. Seen in Figure 14 is the close up friction sleeve with dimensions. The length was chosen as 2.5 inches. After consulting with the archaeologists it was found that the midden depths can greatly vary based on how many years there were deposits. If the level is measured to be 3 cm then the midden is not important enough to document so the only midden being probed for is midden with depths greater than an inch. Two and a half inches was chosen in order to guarantee a friction measurement could be made and it is not too long to be skewed by the layers previously measured about it.

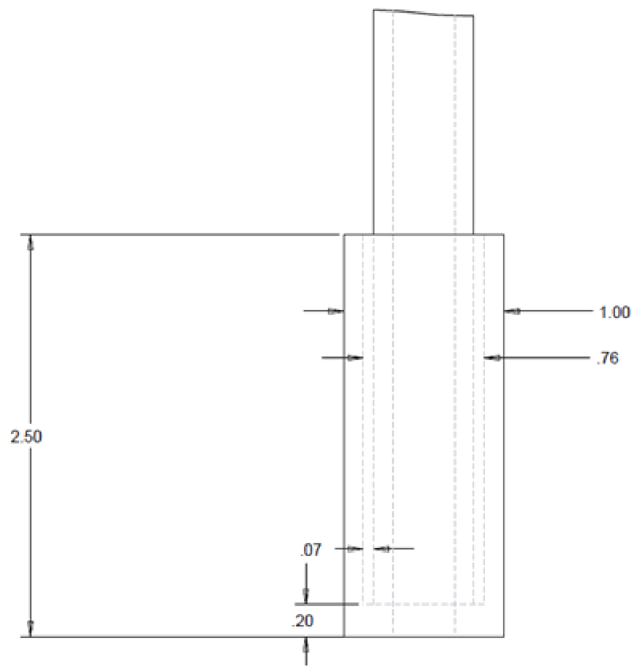


Figure 14. Friction Sleeve Model

The cone tip design is based off of penetrometer standards found in the Geology National Engineering Handbook by NRCS[4].This caused for a 30 degree angle cone tip combined with our diameter of 1 inch. The cone has a depth of 0.289 inches. This can be seen in Figure 15.

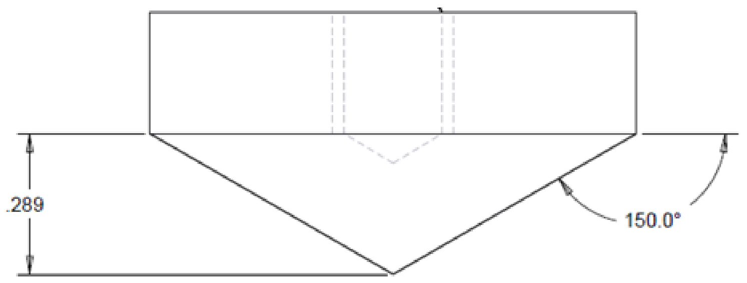


Figure 15. Cone Tip Model

Other than the dimension tolerances that were important, the load cells sensitivity was very important to take into consideration due to the low force read from the friction sleeve and the high force read from the cone tip. The design allowed us to choose more on the capacity of the load cells instead of the size of each load cell. From each load cell, the voltage measurement will be transferred through an amplifier that will allow for a strong signal that will be easily readable by the data acquisition module that was purchased. The housing discs that were machined for each load cell must be created with a tight enough tolerance as for the load cells not to shift while receiving the impact force from the respective rods.

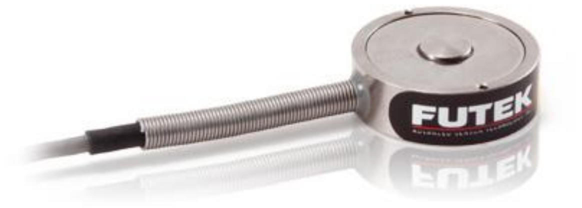


Figure 16. Futek Button Load Cell

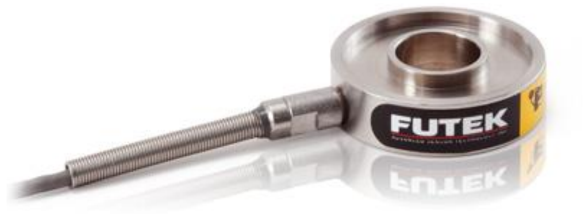


Figure 17. Futek Donut Load Cell

BTH-1208LS Wireless Multifunctional Data Acquisition (DAQ)

The DAQ shown in Figure 18 acquires data over Bluetooth or USB connection. The device will record the output voltage from the load cells and relay the data, through Bluetooth, to an Android device running an application. The specifications can be seen in Figure 19.

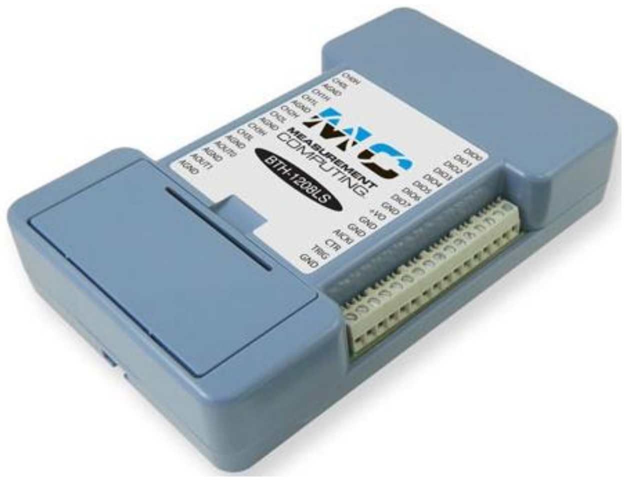
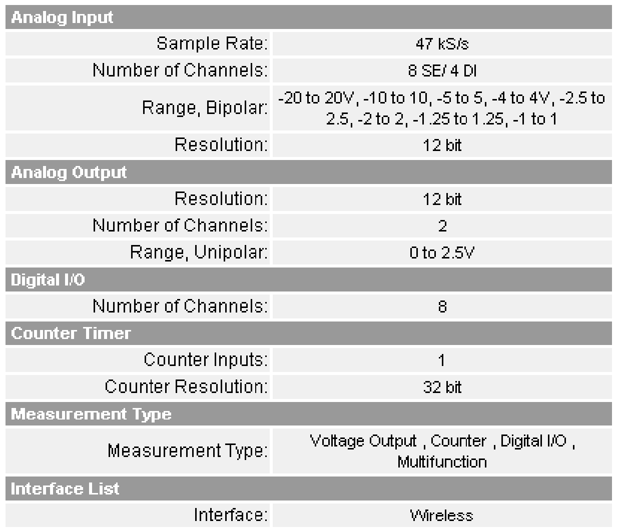


Figure 18. DAQ BTH-1208LS



Analog Input	
Sample Rate:	47 kS/s
Number of Channels:	8 SE/ 4 DI
Range, Bipolar:	-20 to 20V, -10 to 10, -5 to 5, -4 to 4V, -2.5 to 2.5, -2 to 2, -1.25 to 1.25, -1 to 1
Resolution:	12 bit
Analog Output	
Resolution:	12 bit
Number of Channels:	2
Range, Unipolar:	0 to 2.5V
Digital I/O	
Number of Channels:	8
Counter Timer	
Counter Inputs:	1
Counter Resolution:	32 bit
Measurement Type	
Measurement Type:	Voltage Output , Counter , Digital I/O , Multifunction
Interface List	
Interface:	Wireless

Figure 19. DAQ Specifications

Laser Range Finder

The laser range finder shown in Figure 20 is a device from last year’s design. This device uses a laser and a reference point on the penetrometer. As the penetrometer travels into the soil, the reference point will move closer to the laser and measure the displacement. This displacement is the distance the tip of the penetrometer has travelled. This device measures and records the depth and sends the information to an app developed by the team.

Texas Instruments UA7810 15V Voltage Regulator

A Texas Instruments 15V voltage regulator shown in Figure 21 will be used to ensure that a constant 15V is provided to the load cells. The voltage regulator has a maximum input voltage of 30V and a minimum input voltage of 17.5V. A 22.2V rechargeable battery will supply the input voltage for this project.

Futek CSG110 Amplifier

The same manufacturer of the load cells makes amplifiers as well. This amplifier, pictured in Figure 22, is used so that the analog output of the load cells will be in the +/- 5V range. The Futek amplifier will filter noise much more effectively than an amplifier designed by the team because Futek has the resources and technology to get the best out of their products. Designing an amplifier on our own will create too much noise, and the signal may be lost. The Futek amplifier has adjustable gain DIP switches to achieve your specified output.

Figure 20. Laser Range Finder

Figure 21. Voltage Regulator

Figure 22. Futek Amplifier

4.3 Exploded View and Assembly

As shown in the following figures, the completed penetrometer is composed of three sections (Figure 23): a housing design (Figure 24a), a shaft design (Figure 24b), and a drop weight design (Figure 24c). Each section is composed of several pieces that can be found in the Appendix D of this report. The drop weight design is composed of four separate pieces which were ordered pre-modelling of the penetrometer and do not have to be machined for the completion of the penetrometer. The section that connects the drop weight design and the housing design is a connector piece that will have to be machined and welded in place that will be taking a large majority of the impact from the 25 lb weight that will be dropped to apply the load through the penetrometer.

The second piece of the penetrometer is the housing design, the housing design has the most amount of pieces incorporated into the design and must have the highest level of precision when machining. This precision is needed due to the housing having to securely place both the button and donut load cells that will be receiving the force from the friction sleeve rod and the cone tip rod. The load cells that are secured into the housing must not move from the repetitive force from each respective rod and must not shift and off center the location of the force on each load cell. The other discs located in the housing each have their respective duties as supporting each rod to not fall through the penetrometer, and guiding each rod to their respective load cells. The housing will also be sealed by two plates at the bottom and top of the housing shell and this will allow for easy extraction of the discs for maintenance and repair when necessary.

The final section of the penetrometer is the shaft section which is comprised of the outer shell, friction sleeve rod, cone tip rod, and cone tip. The outer shell rod is used as protective layer as dirt, moisture and damage cannot be done to the friction sleeve and inner most cone tip rod. The friction sleeve is the second most layer of the rod design and is connected to the friction sleeve itself which will “feel” the force of the different soil as it slides through the soil. The inner most layer is the cone tip rod, this rod is connected directly to a detachable cone tip located at the bottom of the penetrometer and used as the striking point as the penetrometer enters the ground. The cone tip rod will transmit the force applied to the cone tip through the center of the penetrometer to its respective load cell located in the housing.

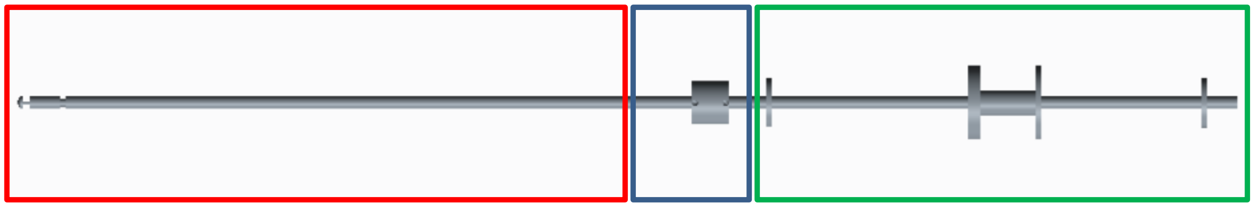
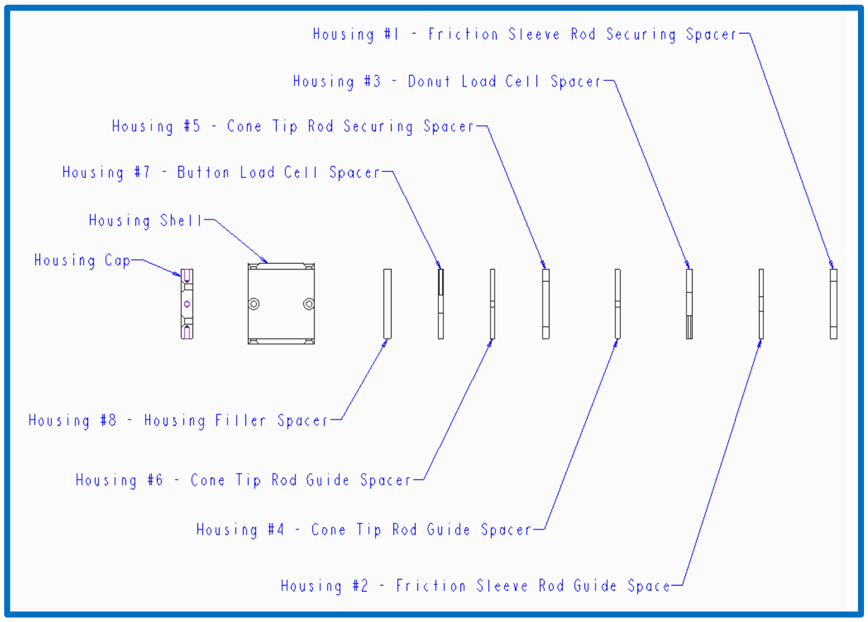


Figure 23. 3D Model of Penetrometer


Housing #1 - Friction Sleeve Rod Securing Spacer
Housing #3 - Donut Load Cell Spacer
Housing #5 - Cone Tip Rod Securing Spacer
Housing #7 - Button Load Cell Spacer
Housing Shell
Housing Cap
Housing #8 - Housing Filler Spacer
Housing #6 - Cone Tip Rod Guide Spacer
Housing #4 - Cone Tip Rod Guide Spacer
Housing #2 - Friction Sleeve Rod Guide Space
Figure 24a. Exploded View of Housing

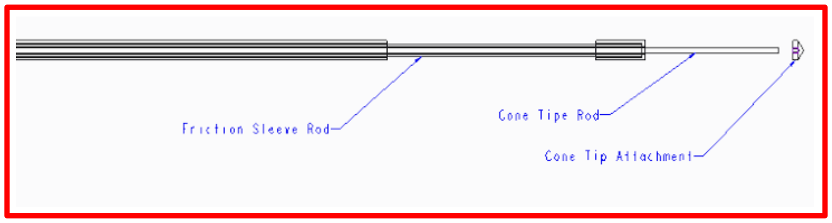


Figure 24b. Exploded View of Rod

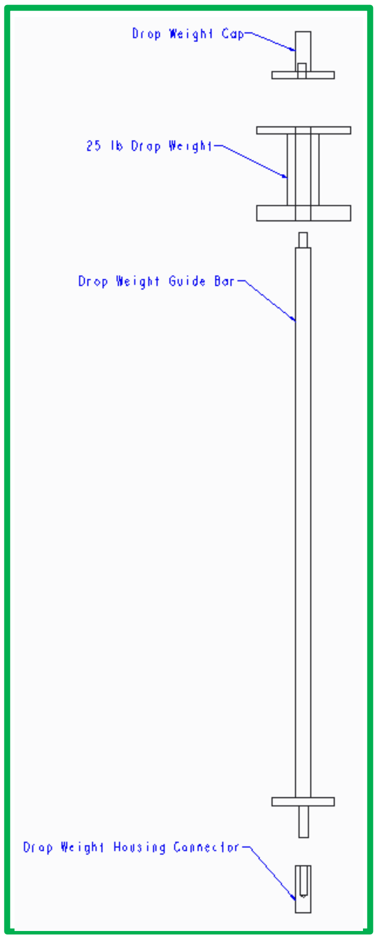


Figure 24c. Exploded View of Drop Weight

4.4 Major Analysis

The failure modes and effects analysis table seen in Appendix C (Table 2) shows the potential failure modes that could occur with the penetrometer operation along with what effects these failures will cause in relation to the reliability and further use of the penetrometer. The RPN is calculated by multiplying the severity ranking by the occurrence and the ability to detect if the problem is going to happen. The higher the number, the more of a problem the failure mode is. Looking at the table it is seen that the two worst failure modes are if the seals that hold the friction sleeve buckle and break, and if the alignment of the rods is off it can also pose a potential problem. To prevent this, extra sealant material was purchased to have on hand at all times for a quick fix, and the seals are being tested repeatedly in the lab. By design, the alignment problems were minimized, but testing is done to ensure calibration in case there is slight bending. The material choice of stainless steel also ensures that there will not be any fracture in the penetrometer rods themselves.

4.5 Programming

The programming aspect of the application is very straightforward. The following is a basic flowchart for the software (Figure 25) and a brief description of the code. The full code can be found in Appendix A and B.

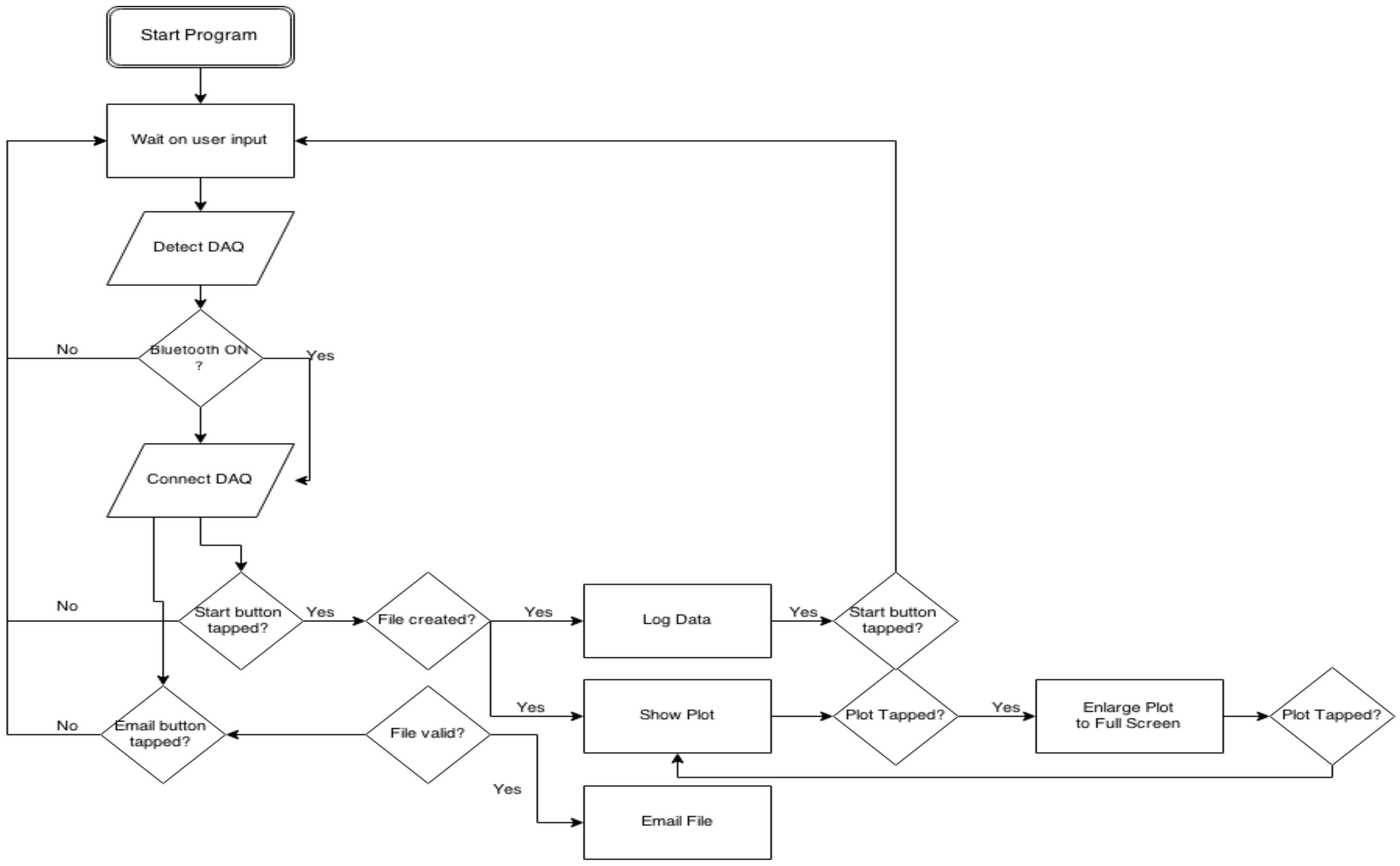


Figure 25. Flowchart for Software

When the program starts, it invokes the OnCreate() Method. This initializes all the necessary variables and most listeners. The OnClickListener() and its objects are called in order to constantly ”listen” for user input. Then the user has the option to DetectDaq().

The OnDaqSelectedListiner() works hand in hand with the DetectDAQ() button to listen for user input when searching for a DAQ that is in Bluetooth mode. Without a connected DAQ, the application will not go any further. It will not start, and all the user can do is either view an older file by tapping the view file button or email the file.

When the start button is chosen, the application calls forth the display scan and data method. This invokes a plot chart that passes its plot point in a two-dimensional array. If the “Email File” button is chosen, this will implement the “send email” feature, and the file will be loaded automatically. An event will popup asking which email provider the user wants to submit the file with, and it will auto-populate the fields based on who is logged in. The file will also be auto-populated into the attachments.

If the “View File” button is chosen, the same flow as the email button being chosen will happen. Both the view and email file method interacts with the user, in order to make the data more mobile. Implementing the full screen view will be done within a few lines of code. Panorama view must be invoked, and the screen locks once the start button has been pressed. The programmer can also choose to programmatically rotate the chart, through its x and y values. Beware the chart will need its own activity.

The data log will be implemented in the displayAndScanData() method. This will pass in the mPlotData double array, along with necessary variables into the LogFileManager class. Once it is there, it will implement the custom user library for Microsoft excel (which can be found at sourceforge.net). This is useful because we need to manipulate the data, as the sponsor wants a graph of the channels Ch0 and Ch1, which will show the two different load cell values.

5. Manufacturing Report and Operation Manual

This section of the report gives a small portion from the Manufacturing, Reliability, and Economics report and the Operation Manual. These two documents, in their entirety, can be attached to this report.

5.1. Mechanical Assembly Process

1. Screw the cone tip on to the smallest rod, the cone tip rod.

Figure 26. Cone Tip Rod

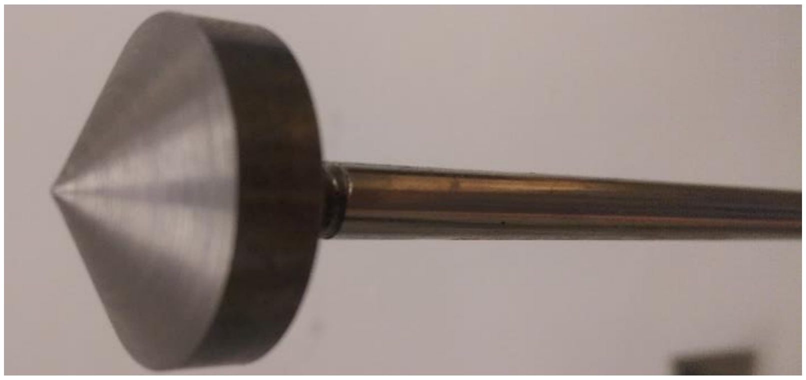


Figure 27. Cone Tip Rod with Cone Tip

2. The second layer is the friction sleeve rod which the cone tip assembly will be slide through with the cone tip at the bottom of the rod near the friction sleeve itself.

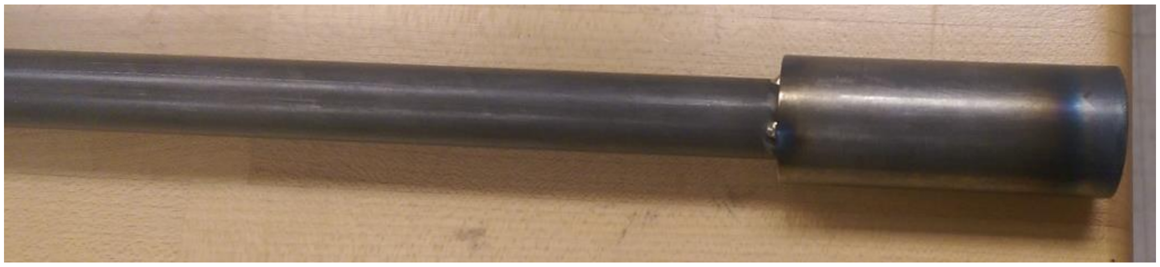


Figure 28. Friction Sleeve Rod

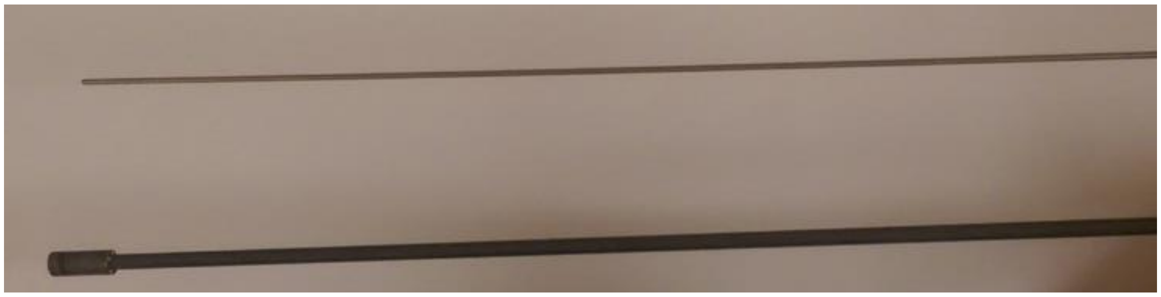


Figure 29. Friction Sleeve Rod and Cone Tip Rod

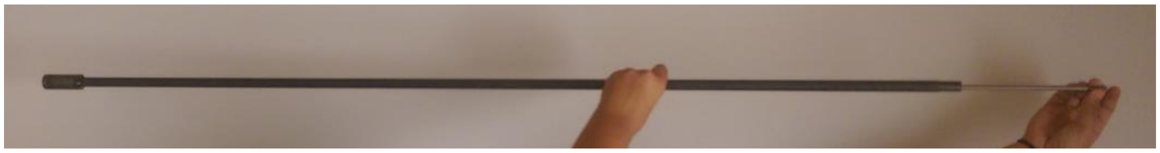


Figure 30. Friction Sleeve and Cone Tip Rod Assembly

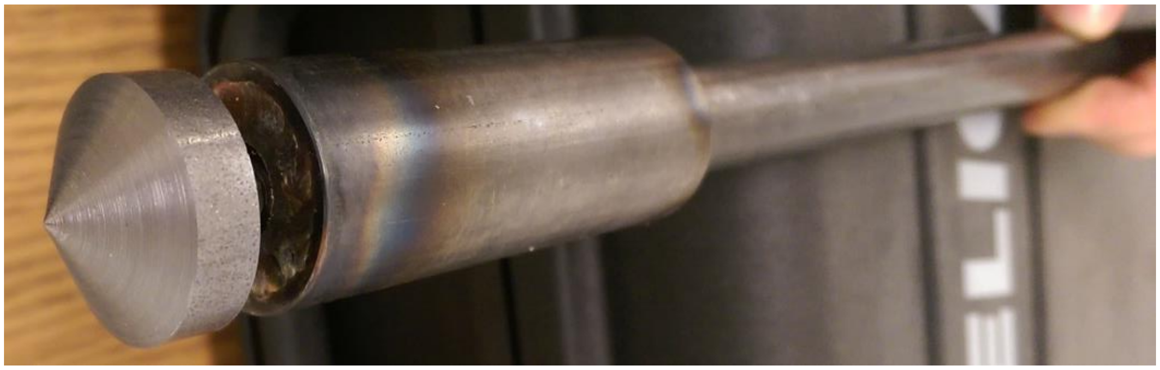


Figure 31. Shaft Assembly

3. The outer shell shaft will be slid over the combined friction sleeve and cone tip rods all the way to the friction sleeve at the bottom of the rod. The two rods, friction sleeve rod and cone tip rod, ends will be protruding from the housing base connected to the out shell shaft.

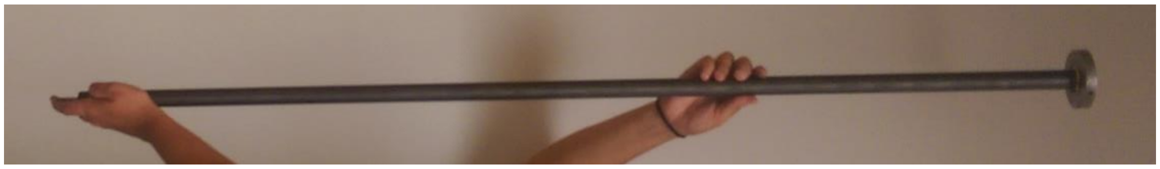


Figure 32. Outer Shell Shaft

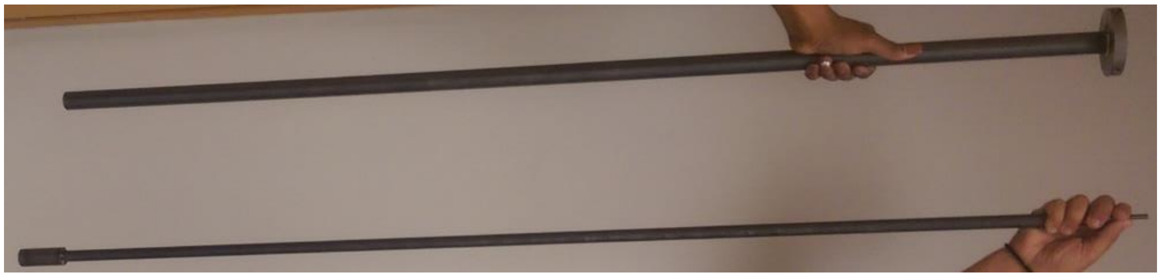


Figure 33. Outer Shell Shaft and Shaft Assembly

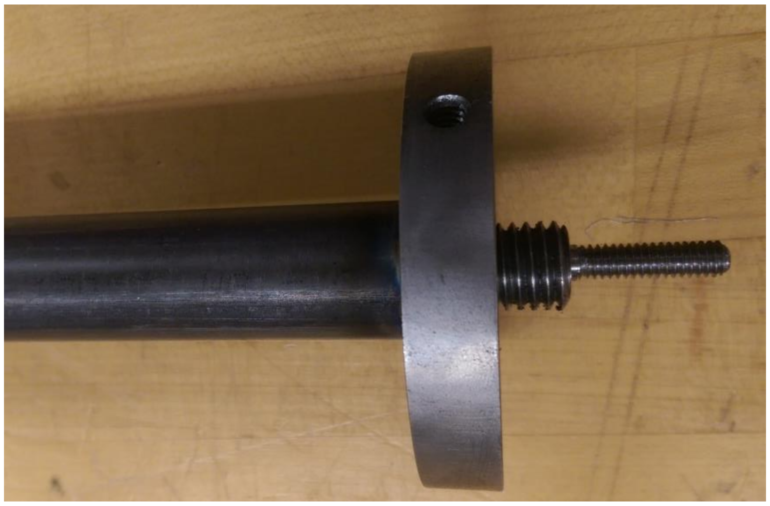


Figure 34. Housing Base Assembly

4. The friction sleeve rod and cone tip will have a corresponding restraining disc matched with the rod. The restraining disc of the friction sleeve will be screwed on only and the cone tip restraining disc will be screwed on later in the assembly.

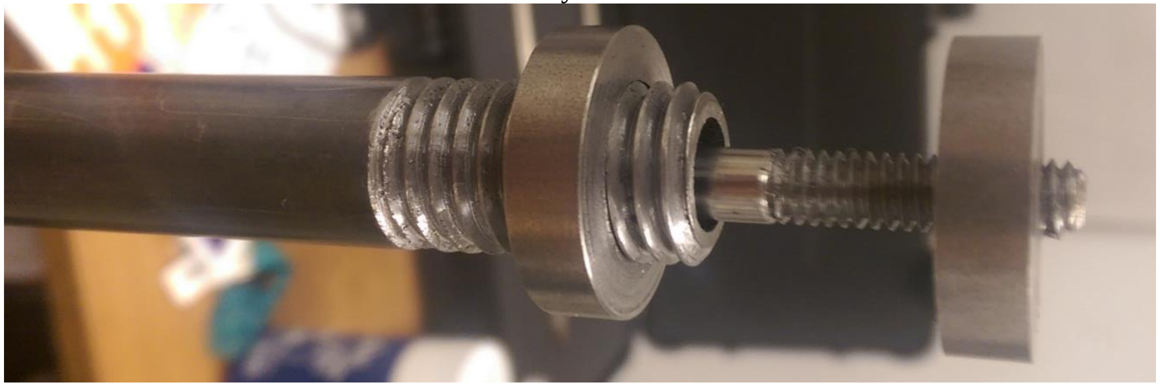


Figure 35. Restraining Discs

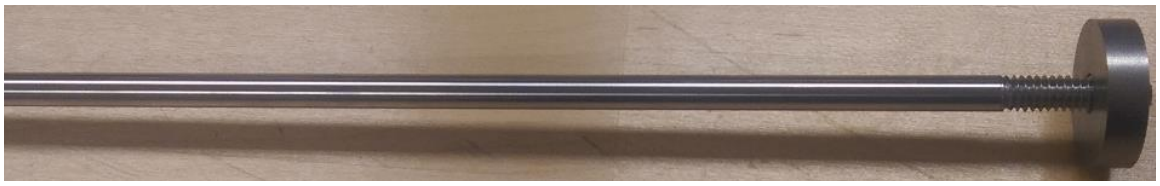


Figure 36. Cone Tip Rod Restraining Disc

5. The housing is comprised of a base, shell, eight inner discs and a top disc for sealing. The discs will have a specific geometry and placement within the housing. Refer to the appendix for numbering and geometry of each disc.

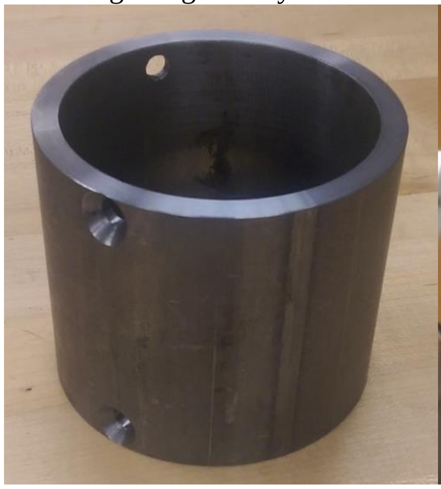


Figure 37. Housing Shell

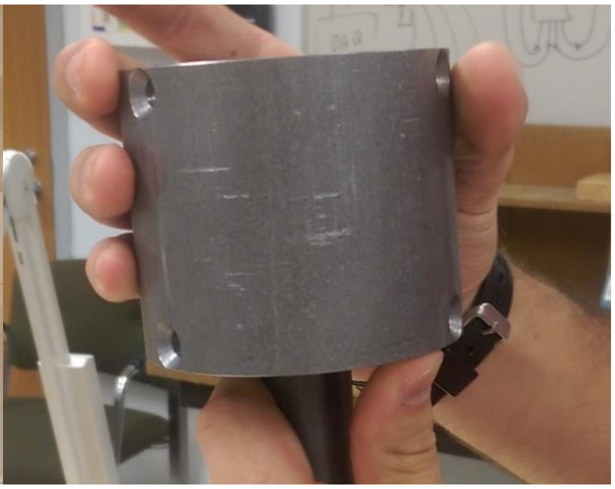


Figure 38. Housing and Base Assembly

The ordering for the discs within the housing are as follows: disc 1, disc 2, disc 3, disc 4, disc 1, disc 4, disc 7 and disc 8. Disc 1 and disc 4 have multiple discs of the same geometry that will be used in the overall housing. Please refer to Appendix D for specific discs.

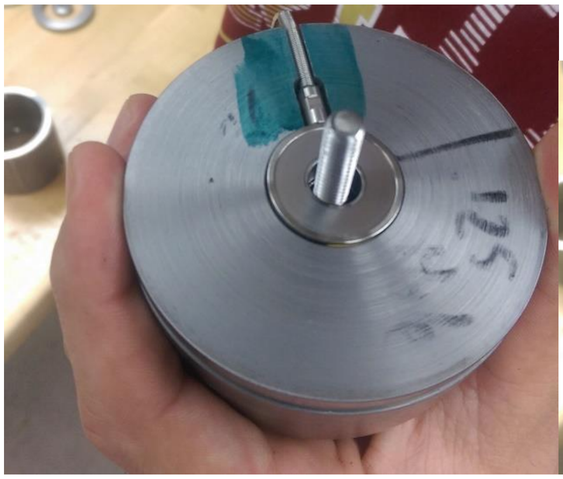


Figure 39. Donut Load Cell Assembly

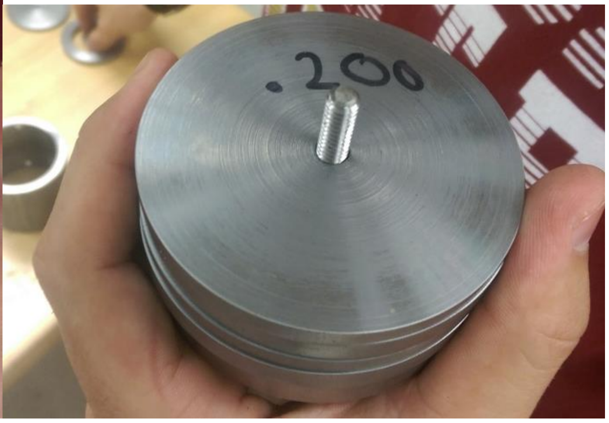


Figure 40. Disc Assembly 3

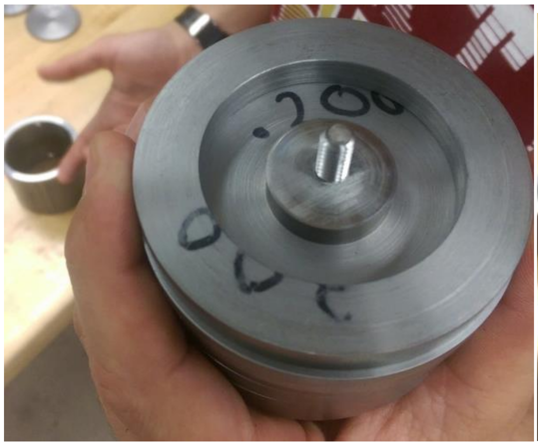


Figure 41. Disc Assembly 4

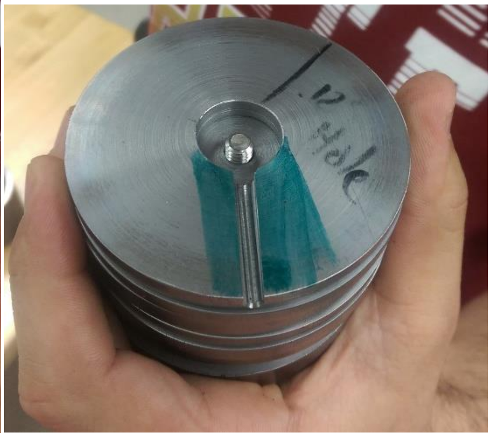


Figure 42. Button Load Cell Disc

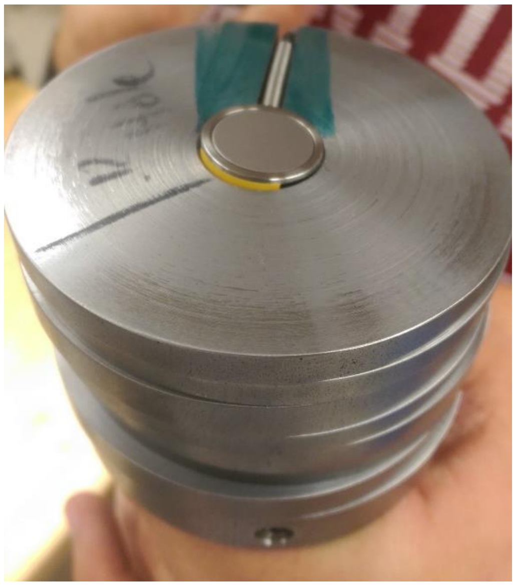


Figure 43. Button Load Cell Assembly

6. After all of the disks are installed, the top disk can be secured into place with screws. Then the drop weight rod can be screwed onto the housing followed by the drop weight and the top lock cylinder as well. This completes the construction of the mechanical side of construction.

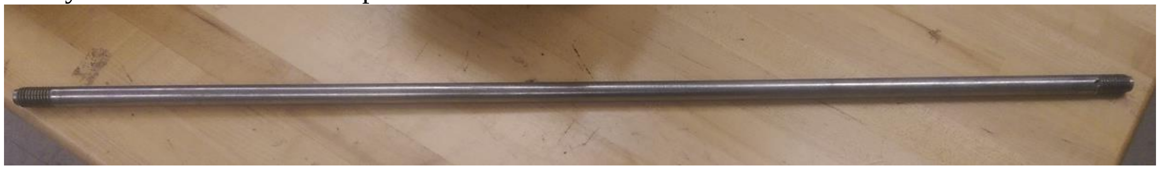


Figure 44. Drop Weight Guide Bar

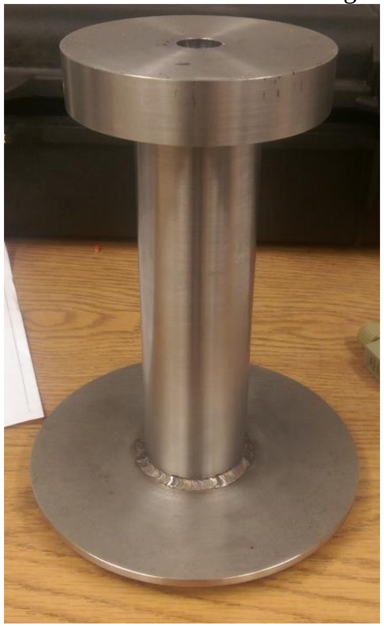


Figure 45. Drop Weight

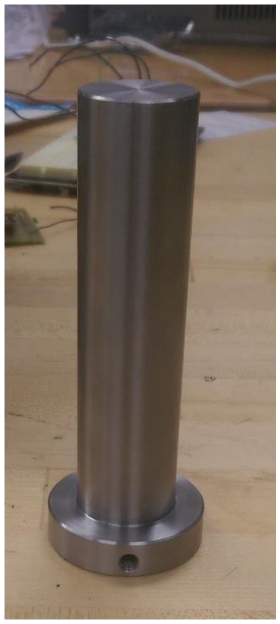


Figure 46. Securing Drop Weight

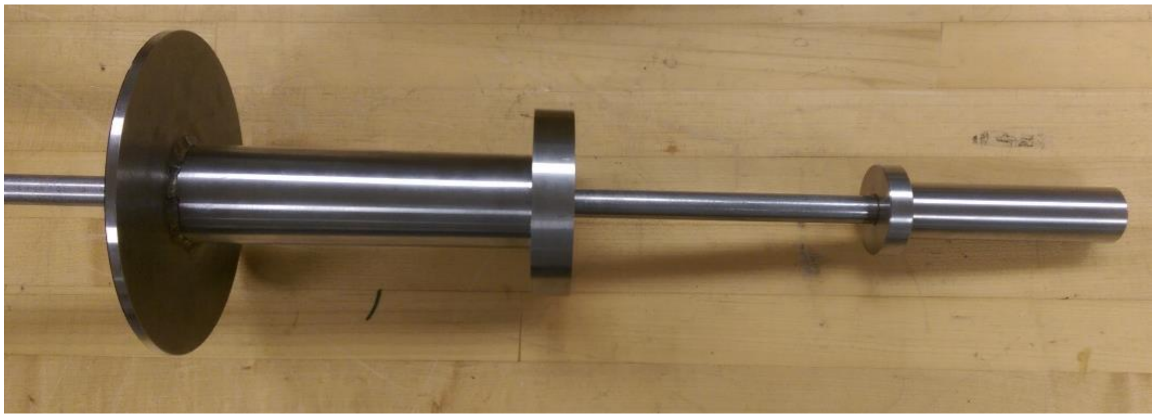


Figure 47. Drop Weight Assembly

5.2. Electrical Assembly Process

The first step for assembly is to apply the acetyl to the center shaft and friction transmission rod to add support and minimize friction. The acetyl is machined in shorter pieces and needs to be stacked all the way up the rods for full support. After this is finished the rods can be combined. Starting with the center load shaft, the cone tip can be screwed onto the shaft that already contains the acetyl. Then, the friction transmission shaft can be slid down over the center load shaft. The friction shaft must be aligned correctly and measured out to have a gap below the shaft and the cone tip to allow for a seal. At this point, the bottom seal should be applied by use of a heat gun and steady hands. After the seal cools and is tested to be sturdy, the outside shaft can be slid on over the friction transmission shaft with another gap left in between for a seal. At this point, the second seal is applied by means of the heat gun, and then it is cooled. Once the entire lower section of the shaft is assembled, the load cells and upper housing will need to be constructed. The outer shaft is to be screwed onto the bottom housing disk. Then, the walls of the housing will be screwed on. At this point, the load cells can be installed. Taking the two sides of the housing insulation, each load cell should be placed in their respective location. Both housing shelves are then inserted at the same time into the housing cylinder. The top housing disk is then screwed on.

The bulk of construction should now be complete with the exception of the bar used for the penetrometer's drop weight. Another cushion disk should be screwed onto the upper housing disk. The drop weight rod is then attached to this disk. The drop weight should at this point slide onto this rod. A top disk is screwed onto the drop weight rod, and the mechanical assembly should be finished.

After the load cells have been installed in the penetrometer the electrical system is ready for installation. Now the voltage regulator must be connected to the electrical system. First the voltage regulator must be connected to a PC board like the one seen in Figure 48. Notice that the PC board pictured has two vertical strips in the middle. Taking the left strip as the positive node and the right strip as the negative node, place the input pin, as seen in Figure 49, into the positive strip and the common pin into the negative strip. Then place the output pin into the nearest node to the left or right of the center strips (whichever is easier). Now solder the pins to their respective nodes. The battery is now ready to be connected to the voltage regulator. Take the positive terminal of the battery (red wire) and connect it to the positive strip of the PC board. Now place the negative terminal of the battery (black wire) into the common node of the PC board. Solder the two terminal of the battery to the PC board. The output node

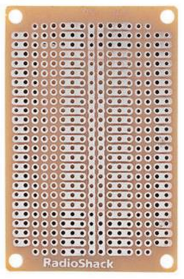


Figure 48. PC Board

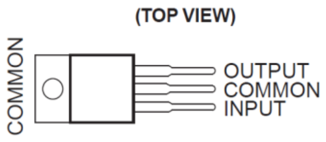


Figure 49. Voltage Regulator with Pins

Figure 50. Voltage Regulator

and common node will now serve as the "new" power nodes with fixed voltage (15V).

Take the casing off both op amps by inserting a flat edge between the upper and bottom cover casings and prying them loose from each other. In order to set the desired gain use the switches denoted by "SW3" of the op amp and place pins 6 and 8 in the up (on) position (all others in down/off position). Now flip SW1 to up so that the excitation is set to 5V. To set the polarity to straight (for compression) set SW2 to the down position. To set the bandwidth of the op amp to 10kHz put SW4 in the up position. After all switches are correctly placed, put the casing back on both op amps.

Now take one power side DB9 cable (pictured in Figure 51) and place its red wire into the output node of the PC board. Then place the black wire of the power DB9 cable and place it into the common node of the PC board. Now solder both the red and black wires to the PC board. The remaining connections of the DB9 power cable are the blue, orange, white and green wires. The white and blue wires are for current output and will not be used in this application. The green and orange wires will later be connected to the DAQ. Now the female DB9 of the power side DB9 cable is ready to be plugged into the male DB9 of the

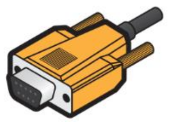


Figure 51. Power
Side DB9 Cable

op amp. For the next power side DB9 cable repeat the same process that was just described. At this point the PC board is ready to be housed for protection. Before placing the housing around the PC board make sure all connections to the board are secure. Place the housing around the PC board.

Now it is time to connect each load cell to an op amp. First take apart the casing of the DB9 connector by removing the nuts and screws that are pictured in Figure 52. Now the female side of the DB9 connector is free for the load cell wires to be soldered to. Before continuing it is imperative to connect the wires of each load cell to the female DB9 connector as described ahead and not to the female DB9 that is attached to the op amp. Using the pin configuration of the female DB9, seen in Figure 53, solder the red wire of the LLB 300 load cell into pin location 1. Next, take the green wire from the LLB 300 load cell and solder to pin 2 of the female DB9. Take the white wire from the LLB 300 and solder it to pin 3 of the female DB9 connector. Then take the black wire of the LLB 300 and solder it to pin 4 of the female DB9 connector. Next place the casing around the DB9 connector and connect the male end into the female end of one of the op amps. For the LTH 300 load cell follow the same procedure and same color code. Now both load cells are connected to an op amp.

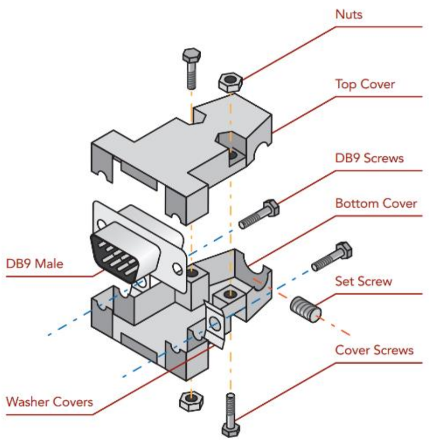


Figure 52. DB9 Connector

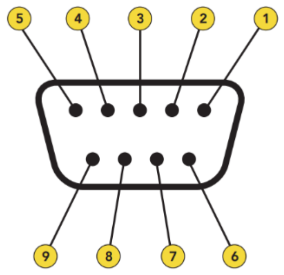


Figure 53. Female Connection of the DB9 Connector

Next connect one op amp to the DAQ by connecting the green wire of the power side DB9 of the selected op amp to the analog input 0. Then take the orange cable of the selected power DB9 cable and connect it to analog input 1. Repeat the same process for the other op amp however connect it to analog input 2 and 3, respectively.

The Android device is now ready to be connected to the DAQ. First make sure the android device is configured to host a Bluetooth device. See the respective android device’s manual to turn on its Bluetooth pairing capability. Next press and hold the red button that is located at position 4 of Figure 54 for at least 5 seconds in order to power on the device and put it into pairing mode. Once the power and statues LEDs are flashing alternately the device is in pairing mode and is ready to be paired with the host device. These LEDs are located at location 5 of Figure 54.

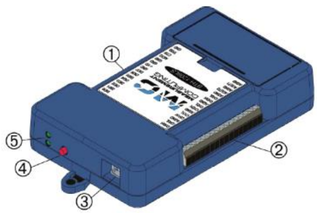
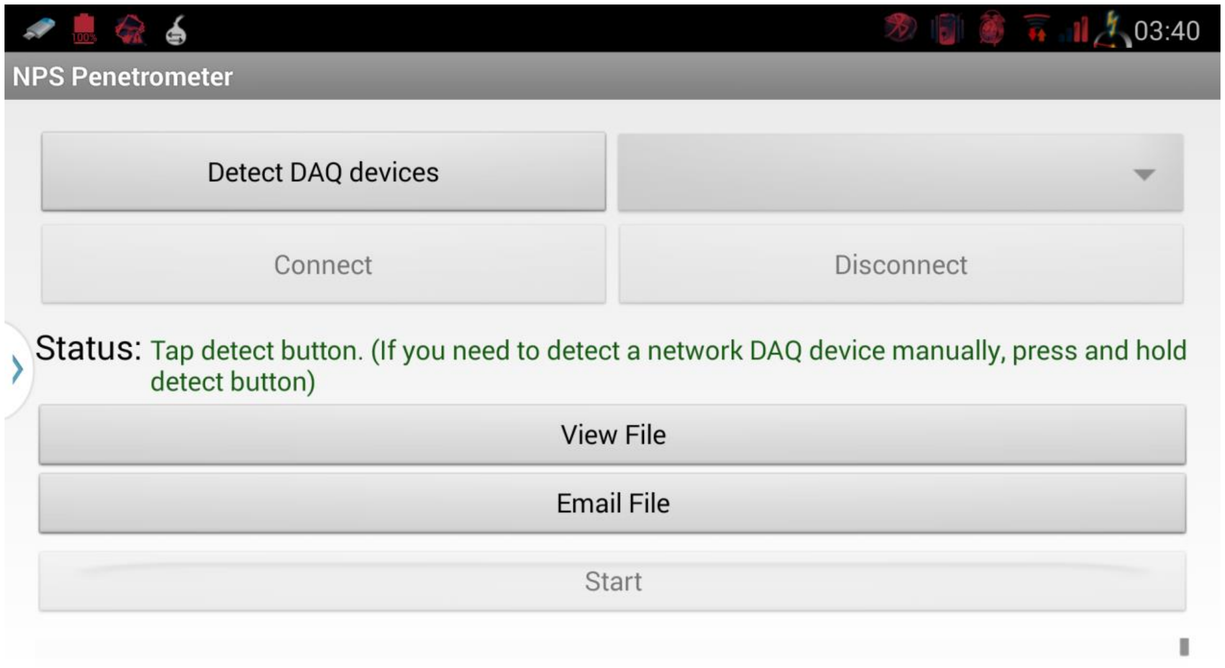


Figure 54. DAQ Components

Now open up the custom NPS app of the android device. Once opened the home screen of the NPS app should be pictured as seen in Figure 55. Press the button that says “detect DAQ devices”. Once the DAQ is detected, press the button that says “Connect to DAQ”. The entire electrical system of the NPS penetrometer is now ready for use.



Status: Tap detect button. (If you need to detect a network DAQ device manually, press and hold detect button)

Figure 55. Home Screen of NPS App

6. Design of Experiment

This section of the report covers how the penetrometer was tested for accuracy and performance. Testing methods and analysis are discussed.

6.1 Tests Performed on Components and Prototype

After finalizing the build of the penetrometer several tests were done in order to test accuracy and how well the penetrometer actually worked. Each load cell needed to be calibrated in order to have useable data and after they were calibrated they needed to be tested in relation with each other. The top load cell measures the friction sleeves force which is a smaller load than read by the bottom load cell that reads the cone tip impact force. Using a comparison of these two values the penetrometer should output what type of soil is being felt. In order to tell which soil the values correlate to, bucket tests were to be done. This involves having large buckets of different soil compositions tested individually to calibrate the penetrometer to each type. For example, the penetrometer is placed in the sand bucket and then the drop weight mechanism is used all the way down the bucket to get the data back from the load cells. The information received is then documented as that particular soil type. That way the next time those range of values come up it is known that it is sand that is being penetrated.

To do these bucket tests, soil samples were collected from Dr. Russo of the National Park Services. Each bucket was to be tested to compare the load cell data and find the differences in each soil. If the data correlates repeatedly with each soil as a single type, then the penetrometer can prove its use in further field testing.

6.2 Results

In figure 56 the graph for the cone tip impact force is shown over an increasing depth in a bucket test. Each peak seen on the graph is when the drop weight was dropped and the penetrometer was forced further into the ground. This graph in particular is what was being read by the button load cell in the upper housing. The peaks seen are pretty consistent throughout the entire test done.

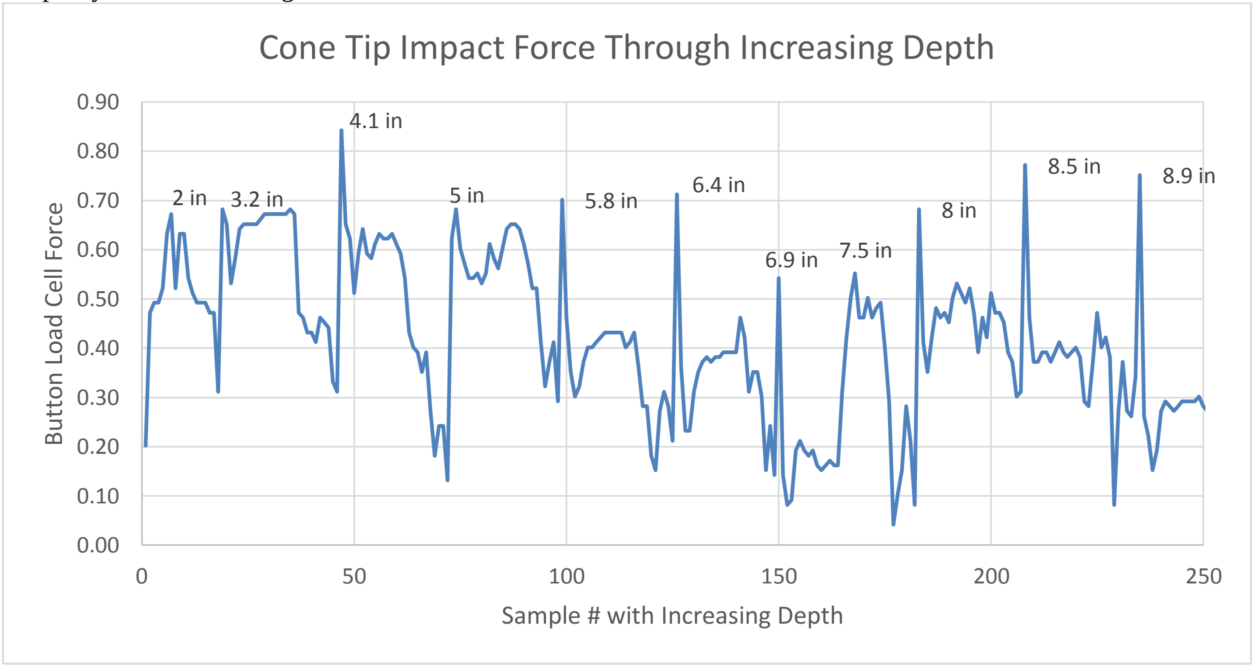


Figure 56. Cone Tip Impact Force through Increasing Depth

In figure 57, the graph for the friction sleeve force is shown over an increasing depth in a bucket test. This is what was read by the thru-hole load cell in the lower part of the housing. The friction sleeve is connected to a transmission shaft so whenever the shaft is moved by traveling further into the soil, the transmission shaft activates the load cell and you get the peaks seen.

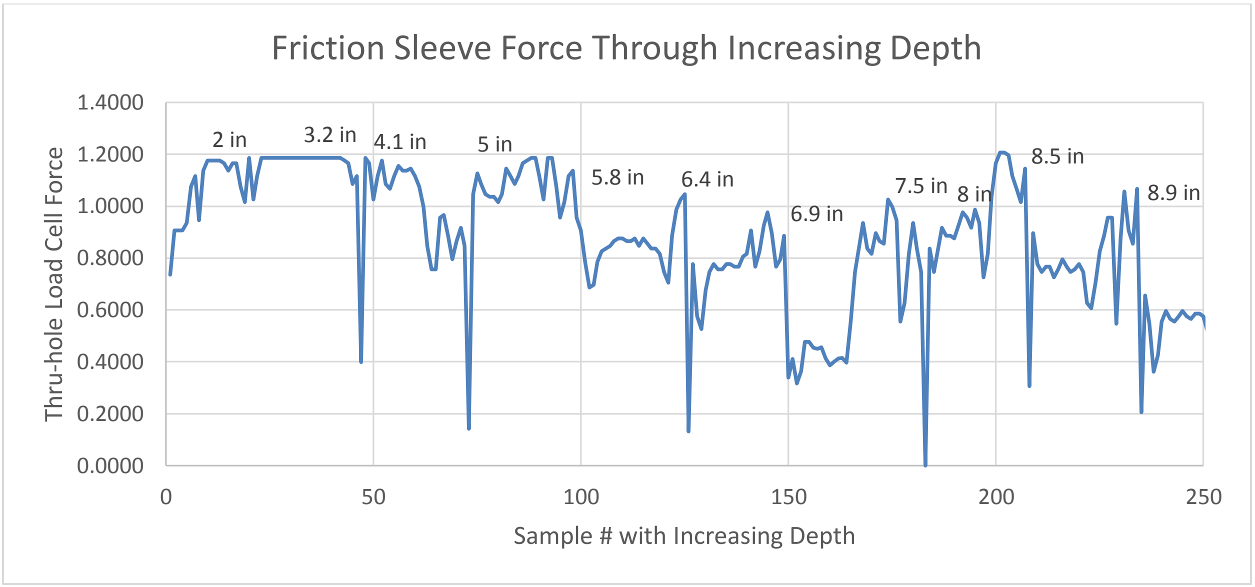


Figure 57. Friction Sleeve Force through Increasing Depth

Using these two figures, a friction coefficient can be found to identify the soil. Each peak has a force value given by the calibrated load cells. Using the average value for the peaks in the cone tip impact force diagram and the average value for the peaks in the friction sleeve force diagram the following equation can be used to calculate the friction coefficient.

$$Friction\ Coefficient = \frac{Top\ Load\ Cell - Bottom\ Load\ Cell}{Bottom\ Load\ Cell}$$

For this test, it was found that the friction coefficient was 0.3966. Using a given table for materials and their friction coefficients, it is found that the "Clean fine sand, silty or clayey fine to medium sand" friction factor is in the range of 0.35 to 0.45. This agrees with our data's finding with our value directly in the middle of this range confirming the test was accurate. When tested again a friction coefficient of 0.39906 was found. This is a difference of 0.00246. Both tests were very close in their results found further proving the accuracy of the probe.

7. Considerations for Environment, Safety, and Ethics

The penetrometer has a low environmental impact because it allows archaeologists to test each site for midden before they dig up the entire field. This probing process allows the archaeologist to be more selective in where he or she chooses to excavate. Because the penetrometer is only one inch in diameter and 4 feet in length, the holes it creates in the field will not have much impact on wildlife. The penetrometer is designed to be transported in very few parts, allowing the user to make only one trip to the test site and lessening the chance of the user dropping anything. This also simplifies the process to pack up at the end of the day, and minimizes the chance of a part being left in the field, which could cause harm to the environment.

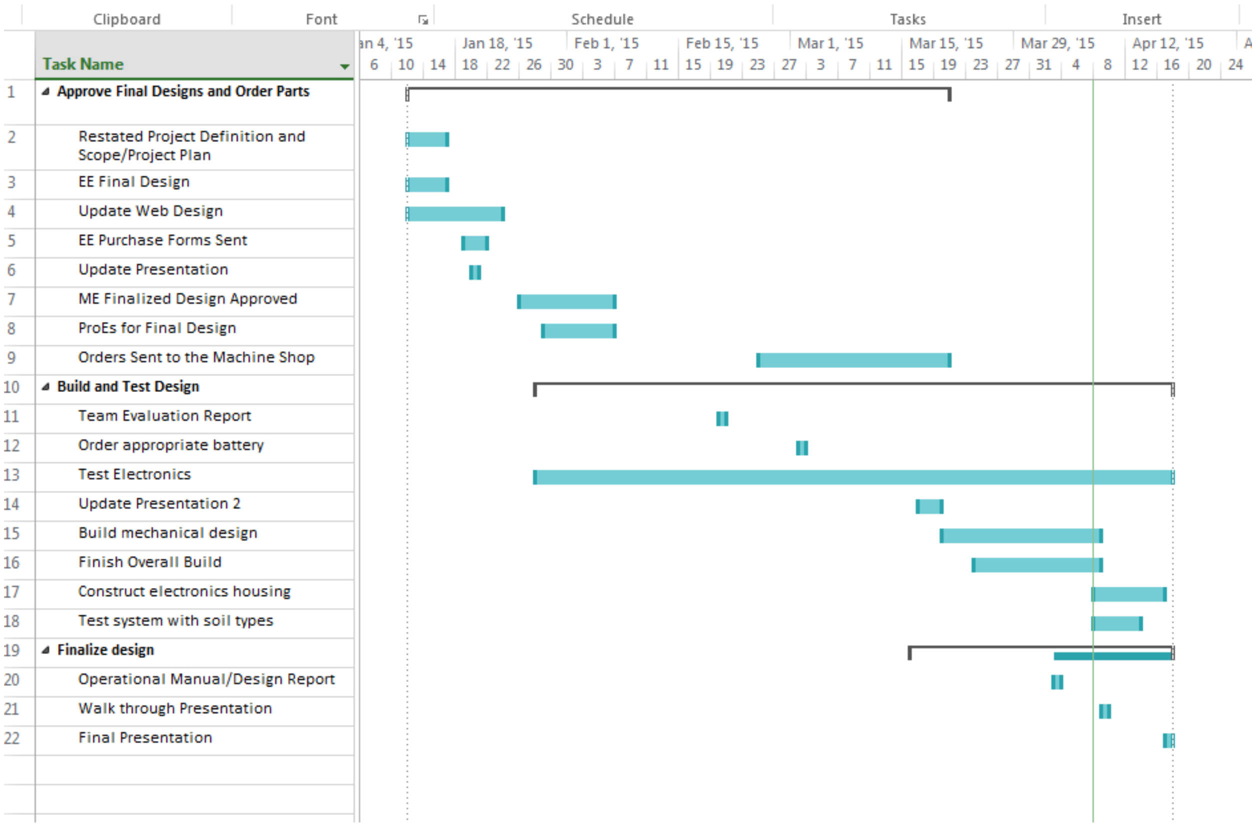
8. Project Management

The following section will discuss the schedule that was laid out, the resources used, the procurement of materials, usage of the allotted budget, and team dynamics.

8.1 Schedule and Resources

Team 18 used many resources throughout the length of the project. Dr. Michael Russo continuously gave us feedback on how he wanted the penetrometer designed and Dr. Shih assisted our design as it progressed throughout the fall semester. The machine shop at the College of Engineering manufactured our parts and Dr. Chuy allowed us to use some of his resources to construct the product.

When referring to our original schedule it seems the team was too optimistic. A large portion of the project was delayed due to unforseen problems. The team originally thought a final design would be completed at the end of November, but due to some advisor and sponsor doubts a complete redesign was done pushing the final design completion to the end of January. The ProE models were completed shortly thereafter, and all of the purchase orders were sent in. Another delay was due to how long the purchase orders took. The procurement office waits until someone else orders from the same company in order to minimize shipping cost. This delayed the arrival of some of our parts for another week or so. After all of the parts came in, the machine shop received all of our drawings and materials the Wednesday before Spring break, March 4th. It took five weeks to receive our parts back. They spent a total of two days machining all of our parts, but they still delayed our project five weeks which was unforseen. This in turn delayed our testing. Our penetrometer was completely built by the 8th of April and testing began immediately.



	Task Name
1	**Approve Final Designs and Order Parts**
2	Restated Project Definition and Scope/Project Plan
3	EE Final Design
4	Update Web Design
5	EE Purchase Forms Sent
6	Update Presentation
7	ME Finalized Design Approved
8	ProEs for Final Design
9	Orders Sent to the Machine Shop
10	**Build and Test Design**
11	Team Evaluation Report
12	Order appropriate battery
13	Test Electronics
14	Update Presentation 2
15	Build mechanical design
16	Finish Overall Build
17	Construct electronics housing
18	Test system with soil types
19	**Finalize design**
20	Operational Manual/Design Report
21	Walk through Presentation
22	Final Presentation

Figure 58. Spring Gantt Chart

8.2 Procurement

Our procurement is broken down into three sections of products: electrical components, mechanical components and electrical mechanical components. The mechanical components, mostly steel, was purchased due to convenience and pricing because of the large amount of different sizes that are necessary for our project. The batteries that were bought comprised a large majority of the budget and these batteries were calculated to be within the needed voltage and amp per hour for all day use with the electronics of the project. There were additional batteries bought for quick exchanges in the field because there will not be a power supply in the field of work with the penetrometer. The data acquisition module is a more unique purchase as it was researched and compared to several other modules such as microprocessors, like the Arduino. The built-in Bluetooth feature of the module and the sampling data rate made the data acquisition module a more suitable choice. The electrical mechanical components were bought together from Futek. The load cells did not have to be special ordered but instead were in stock and standard makes from Futek. This availability for a mid-range force reading in such a small size made these load cells optimal for purchasing. Futek also had their own model of a 5V and 10V amplifiers that could be purchased alongside these load cells to minimize the amount of noise and zero offset from the signal and allowed the team to move forward without the necessity to build an op amp not standard to the load cells.

Below is the mechanical and electrical breakdown of the budget where the team was allotted a $2,000.00 budget. Team 18 has exceeded this amount; however, there has been a $20,000.00 budget from the National Park Services and they were able to purchase the entirety of the project with this new budget. A total of $2,711.40 has been used.

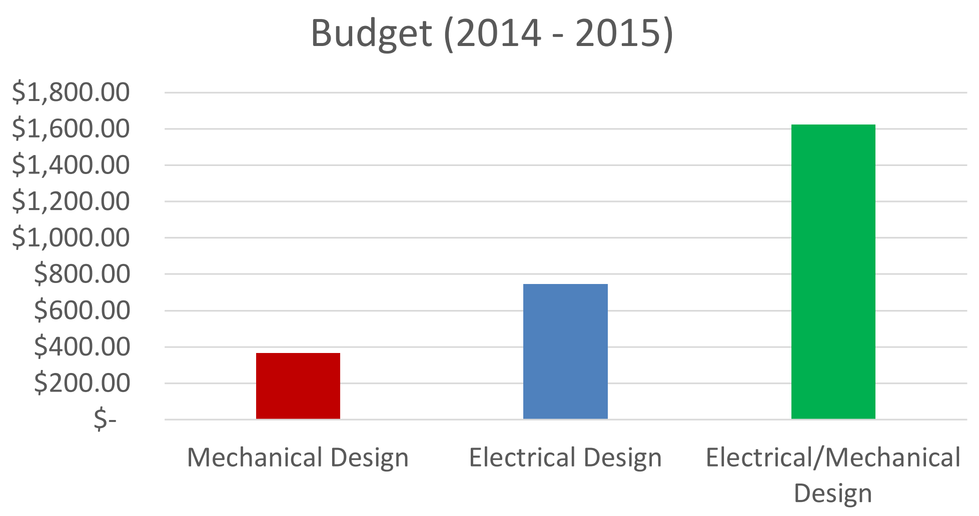


Figure 59. Overall Budget Analysis

For the mechanical breakdown of the budget, the bulk of the money was spent on both the donut and button load cells: $500.00 was spent on the button load cell and $425.00 was spent on the donut load cell. Both load cells were purchased from Futek. The remaining breakdown comes from the materials necessary to complete the project, purchased from McMaster-Carr, totalling in $340.21. Its breakdown can be seen in Figure 60.

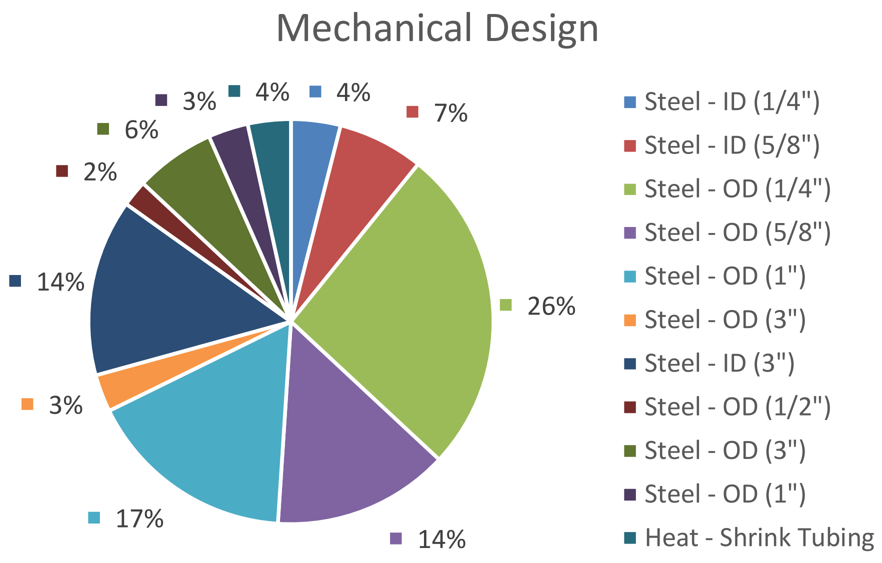


Figure 60. Mechanical Design Budget Analysis

For the electrical breakdown of the budget, the bulk of the money was spent on the batteries, totalling to 73% of the electrical budget at a cost of $700.00, which was also purchased directly from Futek. $542.36 was spent on four batteries and $199.99 was spent on the DAQ. A total of only $3.85 was spent on the voltage regulator purchased from Texas Instruments. This total breakdown can be seen in Figure 61.

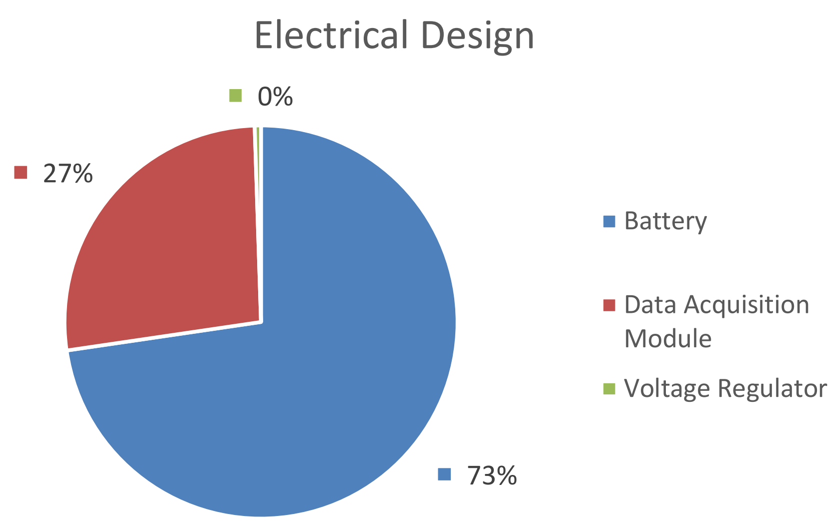


Figure 61. Electrical Design Budget Analysis

For the electrical and mechanical breakdown of the budget, as seen in Figure 62, the bulk of the money was spent on the two load cells, totalling to 57% of the budget at a cost of $925.00 and 43% of the budget was spent on the load cell amplifier totalling to $700.00.

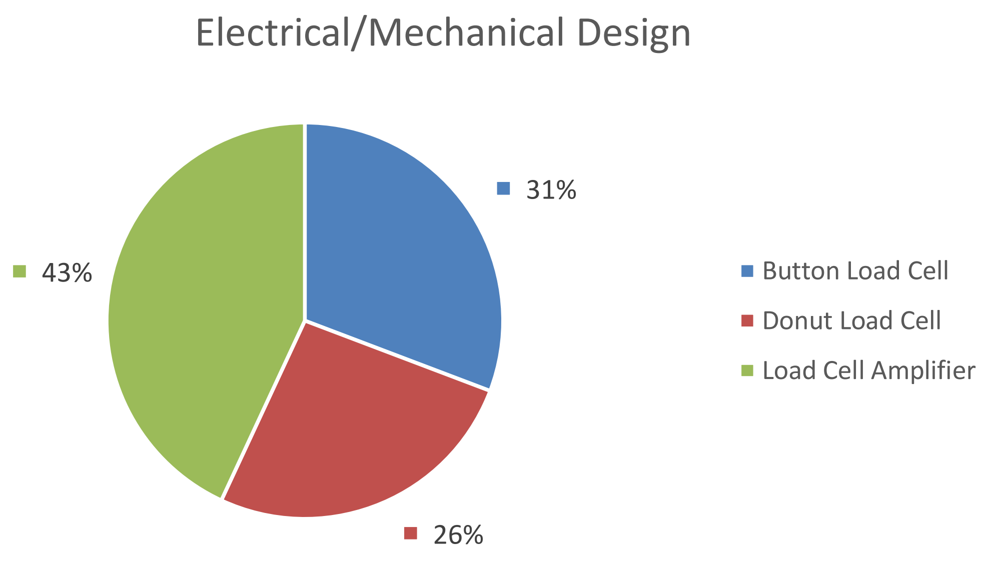


Figure 62. Electrical/Mechanical Design Budget Analysis

8.3 Communications

Communications for Team 18 have not been extremely troublesome this school year. The group has facilitated constant communication using emails, GroupMe messages, text messages, and phone calls. The group also met regularly 1-2 times a week to discuss any issues, work on reports and presentations, and construct and test the prototype. The team members got along quite well, which also contributed to easy communication. The team was able to communicate with the sponsor, Dr. Russo, mainly through emails to keep him updated. He also attended the majority of the Team's presentations, and the Team would discuss with him at that time any issues that were of concern. Biweekly staff meetings were held during both the Fall and Spring semesters. During these meetings, the team would converse with the instructor, Dr. Gupta, and the advisors, Dr. Shih and Dr. Frank, about the progress of the project, any issues that had come up, and potential solutions to these issues. Between these meetings, the team would setup any necessary additional meetings with Dr. Shih or Dr. Frank to seek guidance or gain approval. At times, it was a bit difficult to find time slots when both the team and the advisors were available to meet, and occasionally emails got lost in the mix of others, but the team was able to work through this by persistently reaching out to them.

9. Conclusion and Future Recommendations

This section includes recommendations and suggestions for next year's team, if the sponsor wishes to continue with the project.

9.1 Mechanical Future Recommendations

In the future, it may be of interest to the designer to plan ahead of time and begin this project as quickly as possible. The mechanical side of this project ran into a time crunch at the end of the semester as the machine shop grew busier over the semester. As far as the prototype itself, some future recommendations may include changing the shaft diameter down smaller than an inch and making the overall length of the penetrometer much shorter. At the beginning of the Fall 2014 semester, the sponsor had mentioned wanting to making extension rods to the penetrometer to reach lower depths than the previous prototype, however, this was out of the scope and budget for this year's design team.

9.2 Electrical Future Recommendations

Moving forward with the current prototype, it is suggested to look into some signal processing of the data obtained from the load cells to gather more meaningful data. The current data is somewhat noisy, but is tolerable. The next recommendation is to integrate the laser range finder into the same app with the DAQ. There was not enough time to integrate both Bluetooth devices into one app, and the results from the load cells were of our highest priority.

The objective of this project was to design and build a prototype of a functioning penetrometer that has the ability to differentiate between different types of soil and locate midden based upon the calculated friction coefficient. The penetrometer is portable and weighs less than 50 pounds. It is also user friendly, meaning the operation is not cumbersome. The penetrometer will be forced into the ground, perpendicularly, by means of a drop weight mechanism. As it penetrates the ground, forces will be read by two load cells, a doughnut and button load cell housed above the shafts that will provide readings to determine the friction coefficient of the soil present. The team has assembled their design and will be testing in the near future. After discussion with sponsor, Dr. Russo and the NPS, it has been found that they both are very pleased with the progress of the team and are excited to test and analyse the data alongside Team 18.

10. References

[1]Fee, Rich. "Soil Penetrometers." Probing for Compaction (2005). Successful Farming. Web. 25 Sept. 2014. <http://www.specmeters.com/assets/1/7/soil_penetrometers.pdf>.

[2] McCauley, Amy, and Clain Jones. "Water and Solute Transport in Soils." Soil and Water Management. Montana State University, 1 Jan. 2005. Web. 26 Sept. 2014. <http://landresources.montana.edu/SWM/PDF/final_SW4_proof_11_18_05.pdf>.

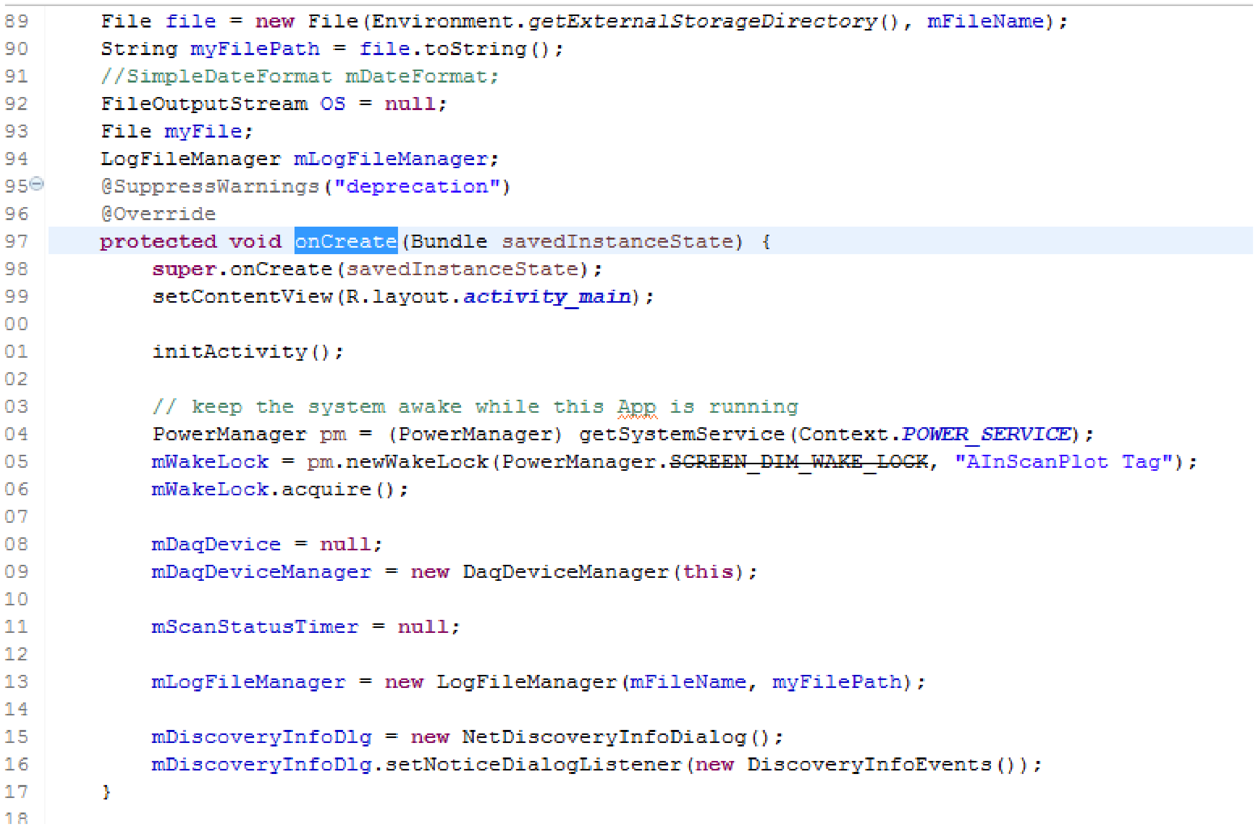
[3] "NOTES on the CONE PENETROMETER TEST." Web.mst.edu. Advanced Engineering Geology & Geotechnics, 1 Jan. 2004. Web. 25 Sept. 2014. <http://web.mst.edu/~rogersda/umrcourses/ge441/Cone Penetrometer Test.pdf>.

11. Appendix A

Code for Application

A – Code for Application

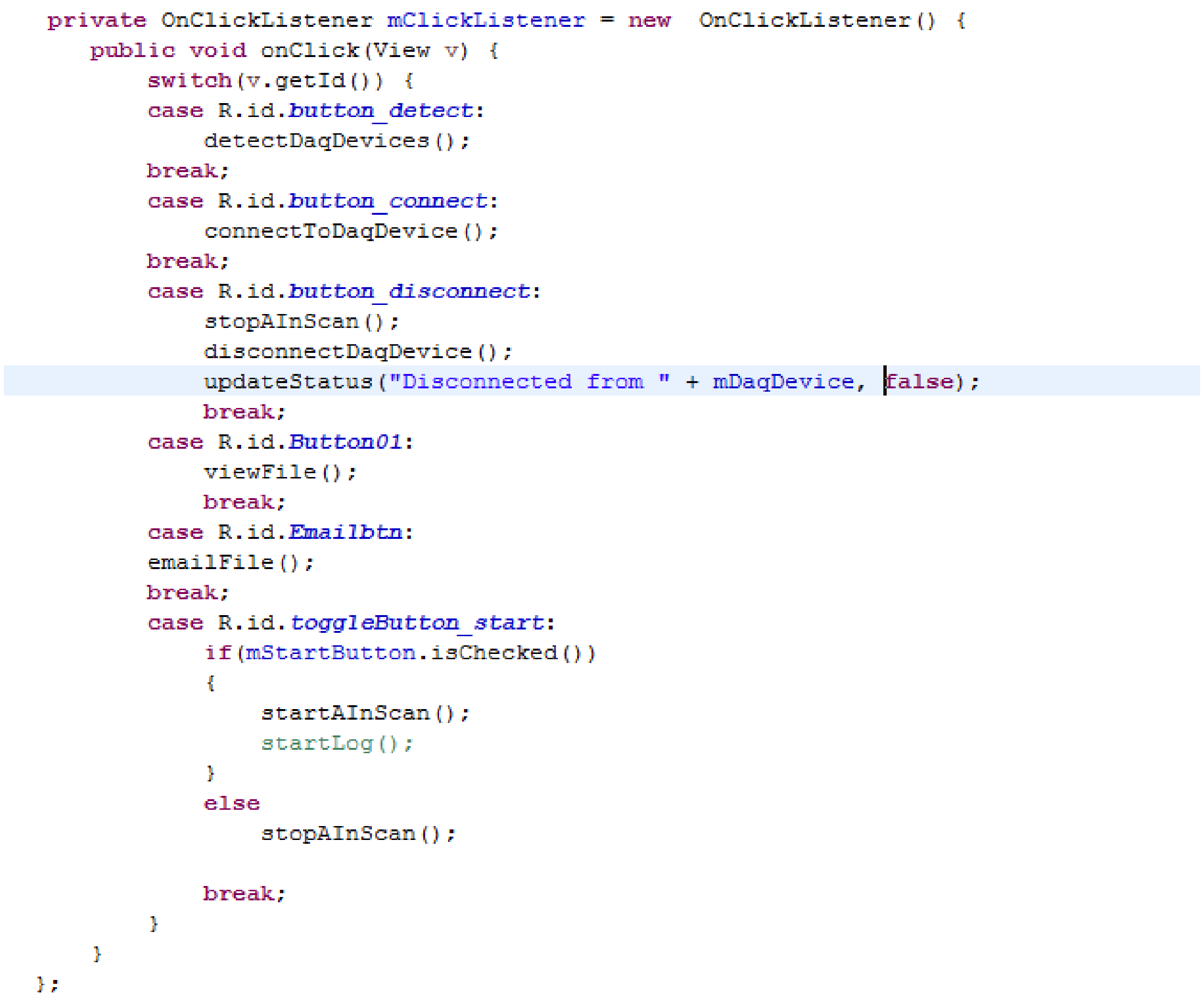
1. OnCreate() – *Program startup*



```
    File file = new File(Environment.getExternalStorageDirectory(), mFileName);
    String myFilePath = file.toString();
    //SimpleDateFormat mDateFormat;
    FileOutputStream OS = null;
    File myFile;
    LogFileManager mLogFileManager;
    @SuppressWarnings("deprecation")
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        initActivity();

        // keep the system awake while this App is running
        PowerManager pm = (PowerManager) getSystemService(Context.POWER_SERVICE);
        mWakeLock = pm.newWakeLock(PowerManager.~~SCREEN_DIM_WAKE_LOCK~~, "AInScanPlot Tag");
        mWakeLock.acquire();

        mDaqDevice = null;
        mDaqDeviceManager = new DaqDeviceManager(this);

        mScanStatusTimer = null;

        mLogFileManager = new LogFileManager(mFileName, myFilePath);

        mDiscoveryInfoDlg = new NetDiscoveryInfoDialog();
        mDiscoveryInfoDlg.setNoticeDialogListener(new DiscoveryInfoEvents());
    }
```

A – Code for Application

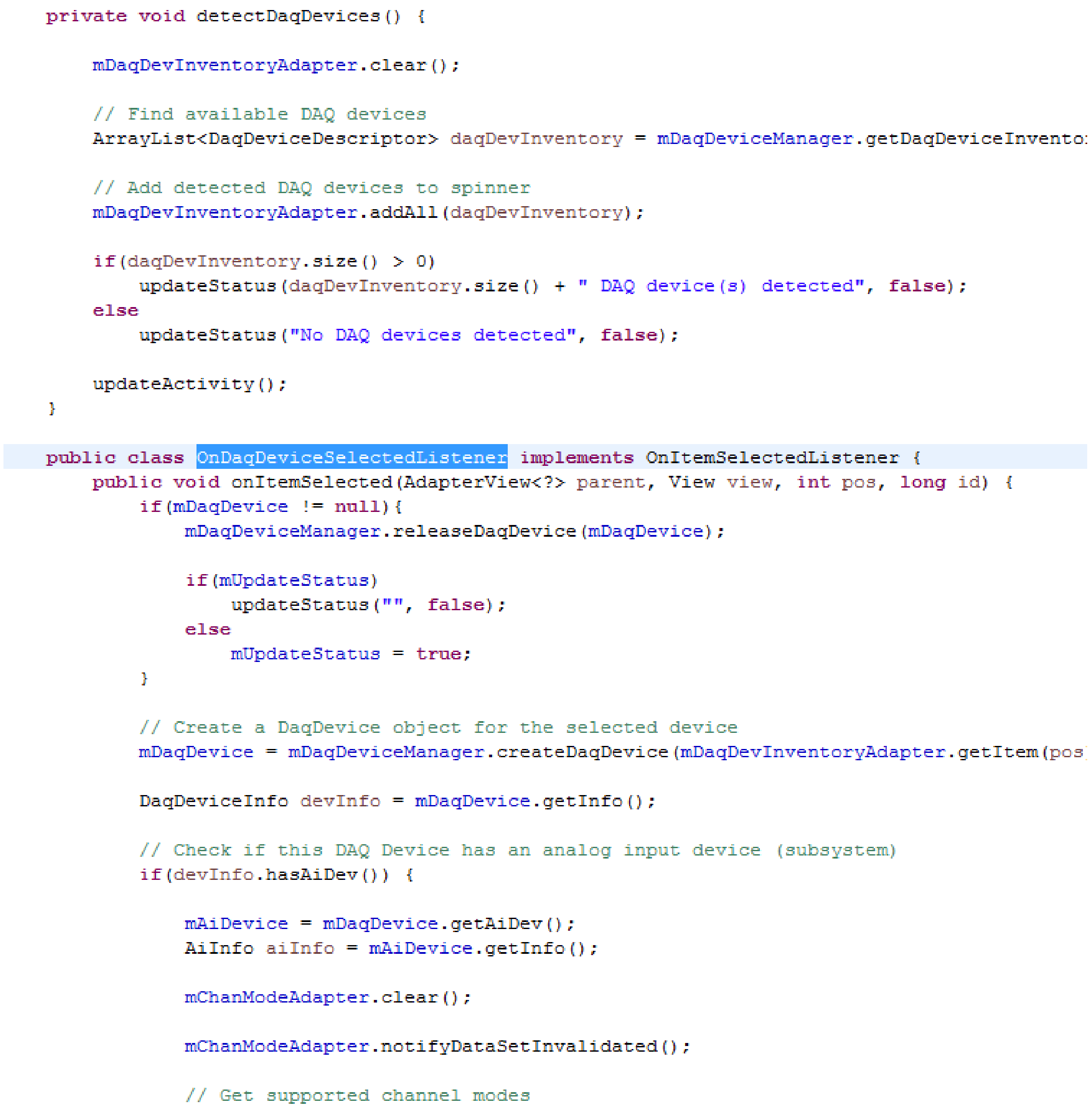
2. OnClickListener – *“Listens” for user input*



```
private OnClickListener mClickListener = new  OnClickListener() {
   public void onClick(View v) {
       switch(v.getId()) {
       case R.id.button_detect:
           detectDaqDevices();
       break;
       case R.id.button_connect:
           connectToDaqDevice();
       break;
       case R.id.button_disconnect:
           stopAInScan();
           disconnectDaqDevice();
           updateStatus("Disconnected from " + mDaqDevice, false);
           break;
       case R.id.Button01:
           viewFile();
           break;
       case R.id.Emailbtn:
       emailFile();
       break;
       case R.id.toggleButton_start:
           if(mStartButton.isChecked())
           {
               startAInScan();
               startLog();
           }
           else
               stopAInScan();

           break;
       }
   }
};
```

A – Code for Application

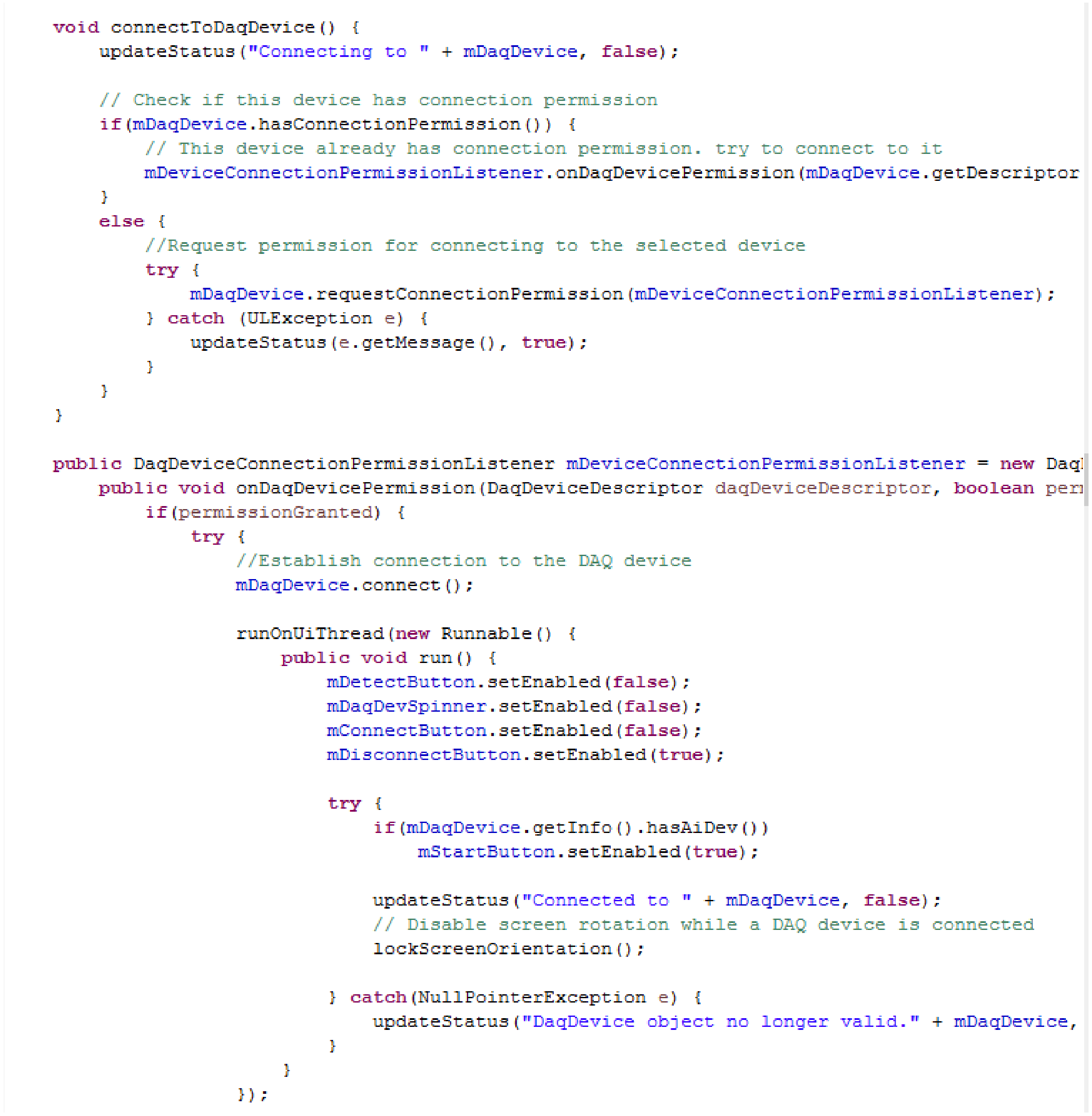
3. DetectDaq and OnDaqSelectedListener – *Searches for DAQ that is in Bluetooth mode*



```
private void detectDaqDevices() {

    mDaqDevInventoryAdapter.clear();

    // Find available DAQ devices
    ArrayList<DaqDeviceDescriptor> daqDevInventory = mDaqDeviceManager.getDaqDeviceInvento

    // Add detected DAQ devices to spinner
    mDaqDevInventoryAdapter.addAll(daqDevInventory);

    if(daqDevInventory.size() > 0)
        updateStatus(daqDevInventory.size() + " DAQ device(s) detected", false);
    else
        updateStatus("No DAQ devices detected", false);

    updateActivity();
}

public class OnDaqDeviceSelectedListener implements OnItemSelectedListener {
    public void onItemSelected(AdapterView<?> parent, View view, int pos, long id) {
        if(mDaqDevice != null){
            mDaqDeviceManager.releaseDaqDevice(mDaqDevice);

            if(mUpdateStatus)
                updateStatus("", false);
            else
                mUpdateStatus = true;
        }

        // Create a DaqDevice object for the selected device
        mDaqDevice = mDaqDeviceManager.createDaqDevice(mDaqDevInventoryAdapter.getItem(pos

        DaqDeviceInfo devInfo = mDaqDevice.getInfo();

        // Check if this DAQ Device has an analog input device (subsystem)
        if(devInfo.hasAiDev()) {

            mAiDevice = mDaqDevice.getAiDev();
            AiInfo aiInfo = mAiDevice.getInfo();

            mChanModeAdapter.clear();

            mChanModeAdapter.notifyDataSetInvalidated();

            // Get supported channel modes
```

A – Code for Application

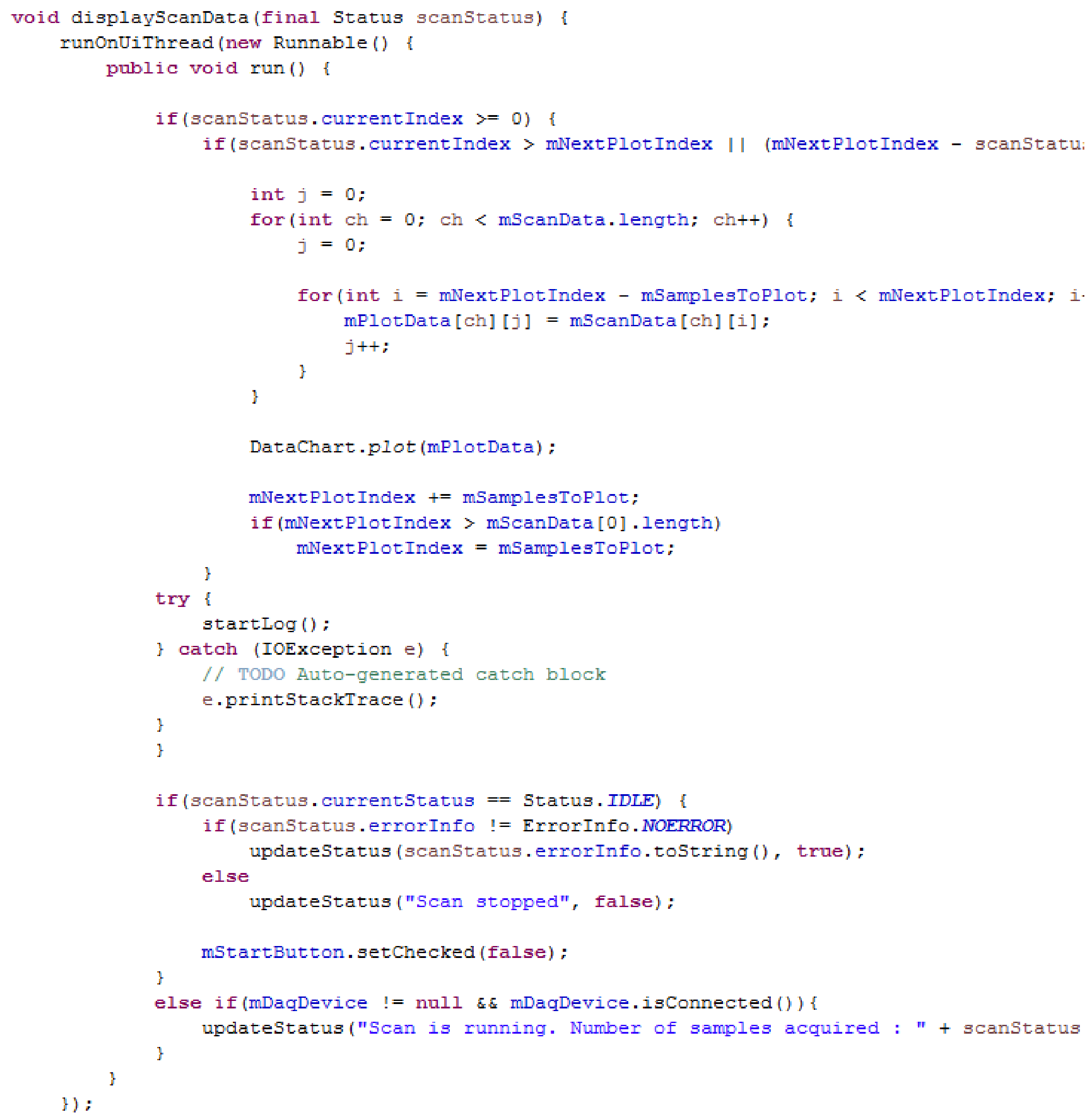
4. connectToDaqDevice()



```
void connectToDaqDevice() {
    updateStatus("Connecting to " + mDaqDevice, false);

    // Check if this device has connection permission
    if(mDaqDevice.hasConnectionPermission()) {
        // This device already has connection permission. try to connect to it
        mDeviceConnectionPermissionListener.onDaqDevicePermission(mDaqDevice.getDescriptor
    }
    else {
        //Request permission for connecting to the selected device
        try {
            mDaqDevice.requestConnectionPermission(mDeviceConnectionPermissionListener);
        } catch (ULException e) {
            updateStatus(e.getMessage(), true);
        }
    }
}

public DaqDeviceConnectionPermissionListener mDeviceConnectionPermissionListener = new Daq
    public void onDaqDevicePermission(DaqDeviceDescriptor daqDeviceDescriptor, boolean per
        if(permissionGranted) {
            try {
                //Establish connection to the DAQ device
                mDaqDevice.connect();

                runOnUiThread(new Runnable() {
                    public void run() {
                        mDetectButton.setEnabled(false);
                        mDaqDevSpinner.setEnabled(false);
                        mConnectButton.setEnabled(false);
                        mDisconnectButton.setEnabled(true);

                        try {
                            if(mDaqDevice.getInfo().hasAiDev())
                                mStartButton.setEnabled(true);

                            updateStatus("Connected to " + mDaqDevice, false);
                            // Disable screen rotation while a DAQ device is connected
                            lockScreenOrientation();

                        } catch(NullPointerException e) {
                            updateStatus("DaqDevice object no longer valid." + mDaqDevice,
                        }
                    }
                });
```

A – Code for Application

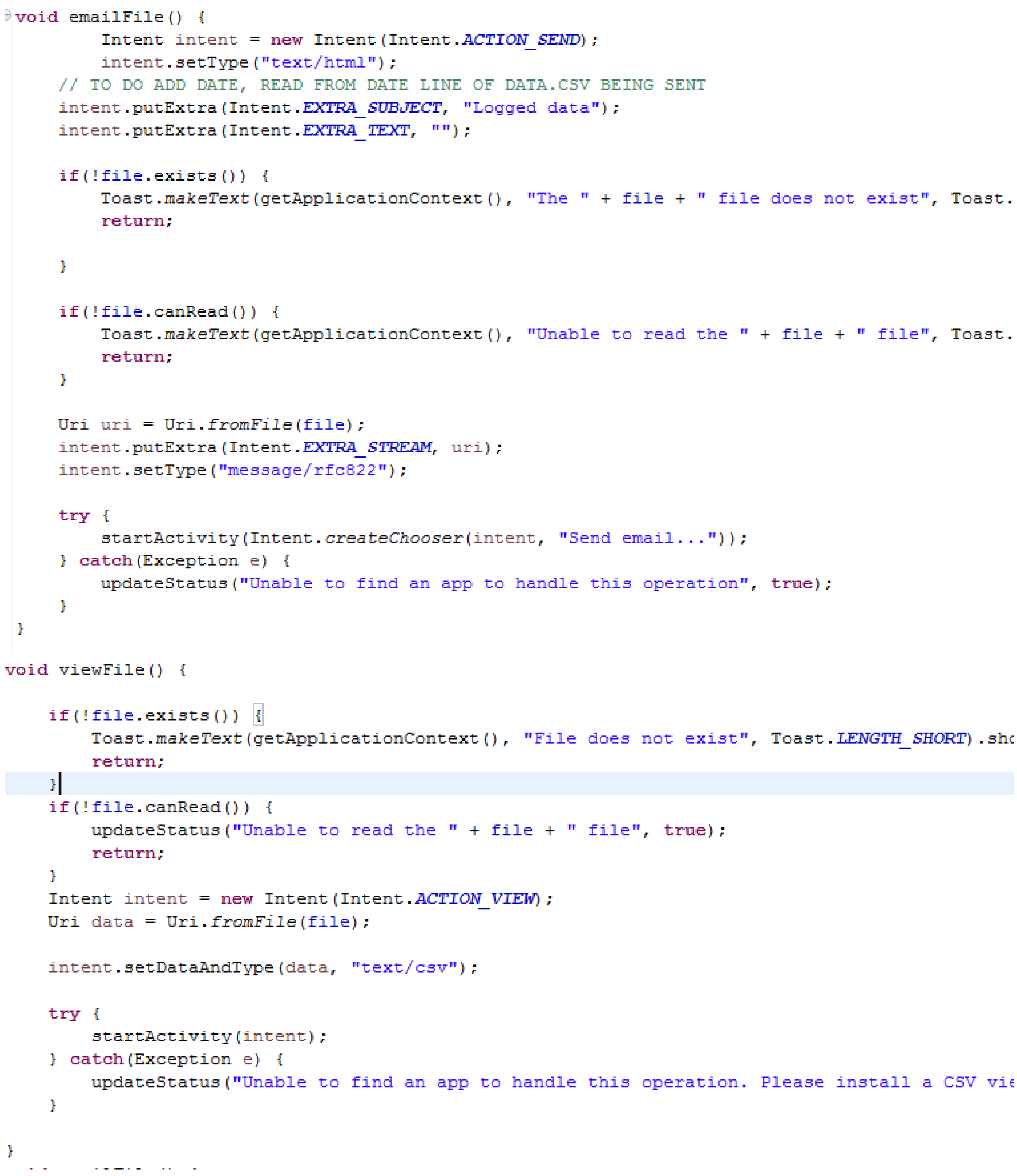
5. DisplayScanData – *Displays Scan Data*



```
void displayScanData(final Status scanStatus) {
    runOnUiThread(new Runnable() {
        public void run() {

            if(scanStatus.currentIndex >= 0) {
                if(scanStatus.currentIndex > mNextPlotIndex || (mNextPlotIndex - scanStatu

                    int j = 0;
                    for(int ch = 0; ch < mScanData.length; ch++) {
                        j = 0;

                        for(int i = mNextPlotIndex - mSamplesToPlot; i < mNextPlotIndex; i
                            mPlotData[ch][j] = mScanData[ch][i];
                            j++;
                        }
                    }

                    DataChart.plot(mPlotData);

                    mNextPlotIndex += mSamplesToPlot;
                    if(mNextPlotIndex > mScanData[0].length)
                        mNextPlotIndex = mSamplesToPlot;
                }
            try {
                startLog();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            }

            if(scanStatus.currentStatus == Status.IDLE) {
                if(scanStatus.errorInfo != ErrorInfo.NOERROR)
                    updateStatus(scanStatus.errorInfo.toString(), true);
                else
                    updateStatus("Scan stopped", false);

                mStartButton.setChecked(false);
            }
            else if(mDaqDevice != null && mDaqDevice.isConnected()){
                updateStatus("Scan is running. Number of samples acquired : " + scanStatus
            }
        }
    });
```

A – Code for Application

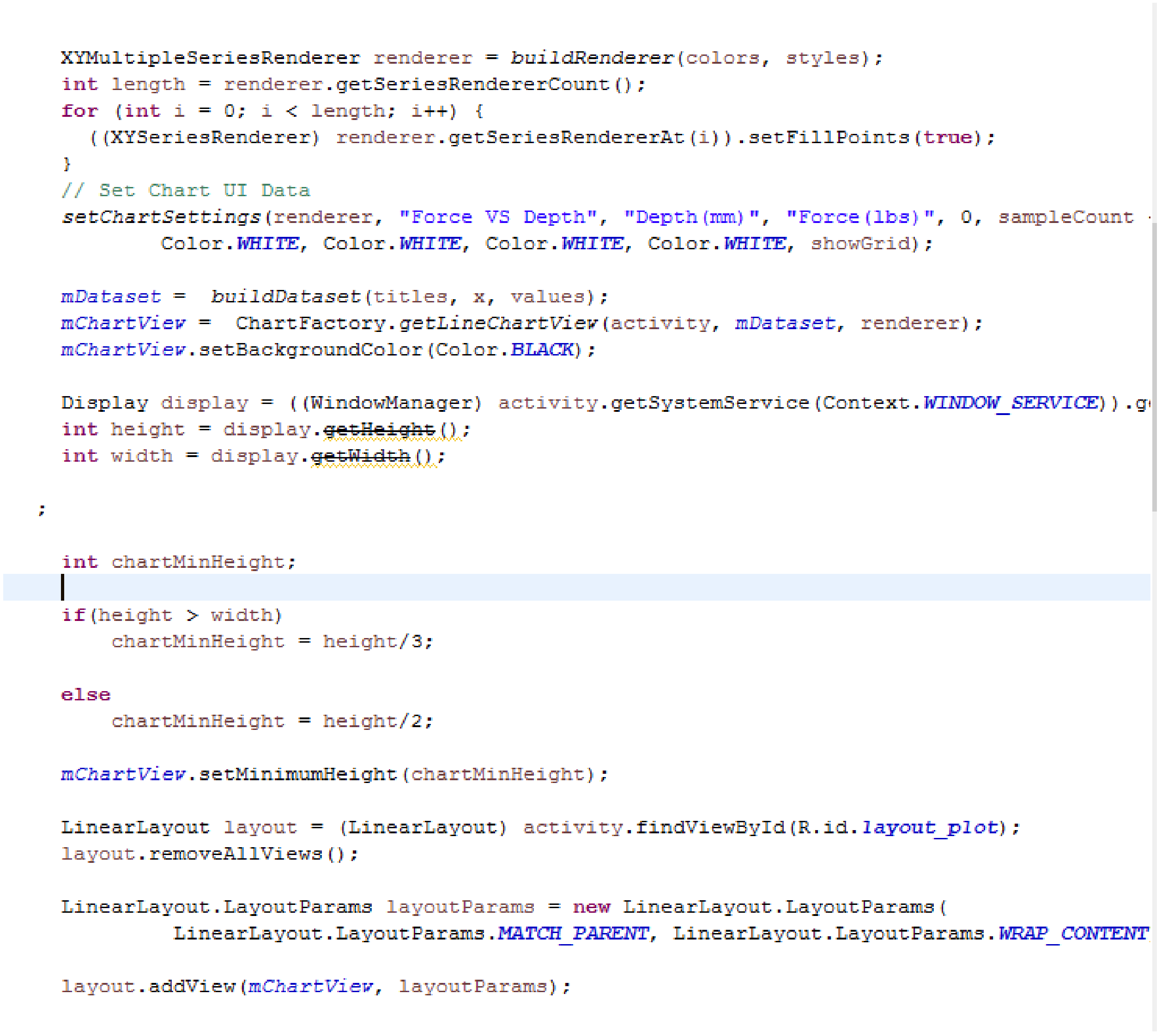
6. EmailFile() – *Emails File*



```
void emailFile() {
        Intent intent = new Intent(Intent.ACTION_SEND);
        intent.setType("text/html");
    // TO DO ADD DATE, READ FROM DATE LINE OF DATA.CSV BEING SENT
    intent.putExtra(Intent.EXTRA_SUBJECT, "Logged data");
    intent.putExtra(Intent.EXTRA_TEXT, "");

    if(!file.exists()) {
        Toast.makeText(getApplicationContext(), "The " + file + " file does not exist", Toast.
        return;

    }

    if(!file.canRead()) {
        Toast.makeText(getApplicationContext(), "Unable to read the " + file + " file", Toast.
        return;
    }

    Uri uri = Uri.fromFile(file);
    intent.putExtra(Intent.EXTRA_STREAM, uri);
    intent.setType("message/rfc822");

    try {
        startActivity(Intent.createChooser(intent, "Send email..."));
    } catch(Exception e) {
        updateStatus("Unable to find an app to handle this operation", true);
    }
}

void viewFile() {

    if(!file.exists()) {
        Toast.makeText(getApplicationContext(), "File does not exist", Toast.LENGTH_SHORT).shc
        return;
    }
    if(!file.canRead()) {
        updateStatus("Unable to read the " + file + " file", true);
        return;
    }
    Intent intent = new Intent(Intent.ACTION_VIEW);
    Uri data = Uri.fromFile(file);

    intent.setDataAndType(data, "text/csv");

    try {
        startActivity(intent);
    } catch(Exception e) {
        updateStatus("Unable to find an app to handle this operation. Please install a CSV vie
    }

}
```

A – Code for Application

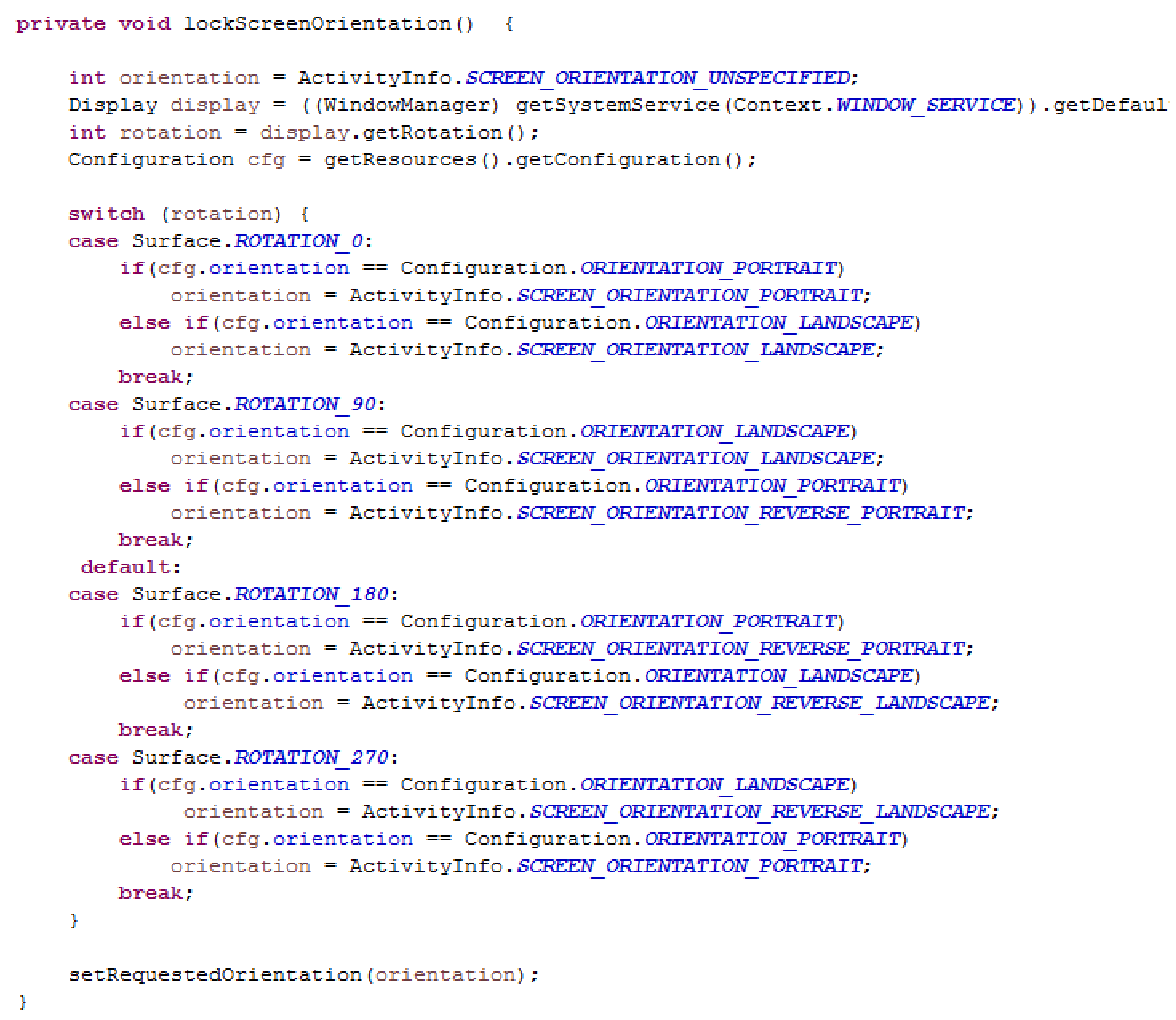
7. FullScreenView - *(Needs Implementing)*



```
    XYMultipleSeriesRenderer renderer = buildRenderer(colors, styles);
    int length = renderer.getSeriesRendererCount();
    for (int i = 0; i < length; i++) {
      ((XYSeriesRenderer) renderer.getSeriesRendererAt(i)).setFillPoints(true);
    }
    // Set Chart UI Data
    setChartSettings(renderer, "Force VS Depth", "Depth(mm)", "Force(lbs)", 0, sampleCount
            Color.WHITE, Color.WHITE, Color.WHITE, Color.WHITE, showGrid);

    mDataset =  buildDataset(titles, x, values);
    mChartView =  ChartFactory.getLineChartView(activity, mDataset, renderer);
    mChartView.setBackgroundColor(Color.BLACK);

    Display display = ((WindowManager) activity.getSystemService(Context.WINDOW_SERVICE)).g
    int height = display.~~getHeight~~();
    int width = display.~~getWidth~~();

  ;

    int chartMinHeight;
    |
    if(height > width)
        chartMinHeight = height/3;

    else
        chartMinHeight = height/2;

    mChartView.setMinimumHeight(chartMinHeight);

    LinearLayout layout = (LinearLayout) activity.findViewById(R.id.layout_plot);
    layout.removeAllViews();

    LinearLayout.LayoutParams layoutParams = new LinearLayout.LayoutParams(
             LinearLayout.LayoutParams.MATCH_PARENT, LinearLayout.LayoutParams.WRAP_CONTENT

    layout.addView(mChartView, layoutParams);
```

A – Code for Application

8. LockScreenOrientation() – *Locks Screen after DaqConnect*



```
private void lockScreenOrientation()  {

    int orientation = ActivityInfo.SCREEN_ORIENTATION_UNSPECIFIED;
    Display display = ((WindowManager) getSystemService(Context.WINDOW_SERVICE)).getDefaul
    int rotation = display.getRotation();
    Configuration cfg = getResources().getConfiguration();

    switch (rotation) {
    case Surface.ROTATION_0:
        if(cfg.orientation == Configuration.ORIENTATION_PORTRAIT)
            orientation = ActivityInfo.SCREEN_ORIENTATION_PORTRAIT;
        else if(cfg.orientation == Configuration.ORIENTATION_LANDSCAPE)
            orientation = ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE;
        break;
    case Surface.ROTATION_90:
        if(cfg.orientation == Configuration.ORIENTATION_LANDSCAPE)
            orientation = ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE;
        else if(cfg.orientation == Configuration.ORIENTATION_PORTRAIT)
            orientation = ActivityInfo.SCREEN_ORIENTATION_REVERSE_PORTRAIT;
        break;
     default:
    case Surface.ROTATION_180:
        if(cfg.orientation == Configuration.ORIENTATION_PORTRAIT)
            orientation = ActivityInfo.SCREEN_ORIENTATION_REVERSE_PORTRAIT;
        else if(cfg.orientation == Configuration.ORIENTATION_LANDSCAPE)
             orientation = ActivityInfo.SCREEN_ORIENTATION_REVERSE_LANDSCAPE;
        break;
    case Surface.ROTATION_270:
        if(cfg.orientation == Configuration.ORIENTATION_LANDSCAPE)
             orientation = ActivityInfo.SCREEN_ORIENTATION_REVERSE_LANDSCAPE;
        else if(cfg.orientation == Configuration.ORIENTATION_PORTRAIT)
            orientation = ActivityInfo.SCREEN_ORIENTATION_PORTRAIT;
        break;
    }

    setRequestedOrientation(orientation);
}
```

12. Appendix B
Complete Application Code

AInScanPlotActivity

```
package com.mcc.ul.example.ainscan.plot;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.EnumSet;
import java.util.Locale;
import java.util.Timer;
import java.util.TimerTask;

import com.mcc.ul.AiDevice;
import com.mcc.ul.AiInfo;
import com.mcc.ul.AiScanOption;
import com.mcc.ul.AiUnit;
import com.mcc.ul.AiChanMode;
import com.mcc.ul.DaqDevice;
import com.mcc.ul.DaqDeviceConnectionPermissionListener;
import com.mcc.ul.DaqDeviceDescriptor;
import com.mcc.ul.DaqDeviceInfo;
import com.mcc.ul.DaqDeviceManager;
import com.mcc.ul.ErrorInfo;
import com.mcc.ul.Range;
import com.mcc.ul.Status;
import com.mcc.ul.ULException;
import com.mcc.ul.example.ainscan.plot.NetDiscoveryInfoDialog;
import com.mcc.ul.example.ainscan.plot.NetDiscoveryInfoDialog.NoticeDialogListener;
import com.mcc.ul.example.ainscan.plot.R;
import com.mcc.ul.example.ainscan.plot.LogFileManager;

import android.net.Uri;
import android.os.Bundle;
import android.os.Environment;
import android.os.PowerManager;
import android.app.Activity;
import android.app.AlertDialog;
import android.app.DialogFragment;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.pm.ActivityInfo;
import android.content.res.Configuration;
import android.graphics.Color;
import android.view.Display;
import android.view.Surface;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.View.OnLongClickListener;
import android.view.WindowManager;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.ScrollView;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;
import android.widget.ToggleButton;

/*
 * Library Call Demonstrated:    AiDevice.aInScan(), continuous background mode

   Purpose:                   Scans a range of A/D Input Channels continuously
                                   in the background and plots the latest acquired data.

   Demonstration:               Continuously collects data on user-specified channels.

   Other Library Calls:         AiDevice.getStatus()
                           AiDevice.StopBackground()

   Special Requirements:        Selected device must have an A/D converter.


   Steps:
   1. Create a DaqDeviceManager object
   2. Call DaqDeviceManager.getDaqDeviceInventory() to find available DAQ devices.
   3. Call DaqDeviceManager.createDevice() to create a DaqDevice object for the desired device
   4. Call DaqDevice.requestConnectionPermission() to request permission for connecting to the device
   5. If Permission granted call DaqDevice.connect() to connect to the device
   6. Call DaqDevice.getAiDev() to retrieve the analog input device object
   7. Call AiDevice.aInScan() to start the scan operation
   8. Call AiDevice.getStatus() to check the status of the background operation
   9. Call AiDevice.stopBackground() when scan is completed

           Note: Declare the following permissions in your application manifest file (AndroidManifest.xml)
           <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />

    In order to create large data arrays add android:largeHeap="true" to AndroidManifest.xml under <application>
tag
*/
/*********
 * Deneuve Brutus
 * 3/1/2015
 * Modified the log button code to create Log file data.csv in root folder
 *
 * *
 * **********/
public class AInScanPlotActivity extends Activity {
```

```
final static int TIMER_PERIOD = 10; //ms
final static int GREEN = Color.parseColor("#165B12");
final static double DEFAULT_RATE = 500.0;
final static int DEFAULT_SAMPLE_PER_CHAN = 10000;
final static int MAX_SAMPLES_TO_PLOT = 1000;
final static int MAX_SAMPLES_TO_PLOT_HIGH_RATE = 100;
final static int HIGH_RATE = 10000;
final static String LOG_FOLDER = "NPS";
final static String DEFAULT_LOG_FILE_NAME = "data.csv";

private DaqDeviceManager mDaqDeviceManager;
private DaqDevice mDaqDevice;
private AiDevice mAiDevice;



private Timer mScanStatusTimer;
private PowerManager.WakeLock mWakeLock = null;

boolean mUpdateStatus;
NetDiscoveryInfoDialog mDiscoveryInfoDlg;

// UI objects
Button Logbtn;
Button mViewFileButton;
Button mDetectButton;
Button mConnectButton;
Button mDisconnectButton;
Spinner mDaqDevSpinner;
Spinner mLowChanSpinner;
Spinner mHighChanSpinner;
EditText mSamplesToPlotEditText;
EditText mRateEditText;
Spinner mChanModeSpinner;
Spinner mRangeSpinner;
Spinner mUnitSpinner;
ToggleButton mStartButton;
TextView mScanDataTextView[];
TextView mScanChansTextView[];
TextView mDataTextView[];
TextView mFileNameEditText;
ArrayAdapter<DaqDeviceDescriptor> mDaqDevInventoryAdapter;
ArrayAdapter<Integer> mLowChanAdapter;
ArrayAdapter<Integer> mHighChanAdapter;
ArrayAdapter<AiChanMode> mChanModeAdapter;
ArrayAdapter<Range> mRangeAdapter;
ArrayAdapter<AiUnit> mUnitAdapter;
TextView mStatusTextView;


double [] mAInData;
double[][] mScanData;
double[][] mPlotData;

AiUnit mUnit;
```

```
            //Set units to VOLTS for ALL
            AiUnit units = mUnit.VOLTS;
            //Set All channel mode to Differential
            AiChanMode mode = AiChanMode.DIFFERENTIAL;
            AiChanMode mChanMode = mode;
            //Set Range for all to +-5V
            Range range = Range.BIP5VOLTS;
            Range mRange = range;
            int mChanCount;
            int mSamplesToPlot;
            int mNextPlotIndex;
            int mSamplesPerChan;
            int mSamplesRead;
            long mTimerPeriod;
            int mLowChan;
            int mHighChan;

            SimpleDateFormat mDateFormat;
            SimpleDateFormat mTimeFormat;
            FileWriter mFileWriter;

            LinearLayout mDataLayout;
            LinearLayout mChansLayout;
            TextView mChansTextView[];

            //FileWriter mFileWriter;
            String mFileName = "data.csv";
            File file = new File(Environment.getExternalStorageDirectory(), mFileName);
            String myFilePath = file.toString();
            //SimpleDateFormat mDateFormat;
            FileOutputStream OS = null;
            File myFile;
            LogFileManager mLogFileManager;
    @SuppressWarnings("deprecation")
            @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        initActivity();

        // keep the system awake while this App is running
        PowerManager pm = (PowerManager) getSystemService(Context.POWER_SERVICE);
                        mWakeLock = pm.newWakeLock(PowerManager.SCREEN_DIM_WAKE_LOCK, "AInScanPlot
Tag");
                        mWakeLock.acquire();

        mDaqDevice = null;
        mDaqDeviceManager = new DaqDeviceManager(this);

        mScanStatusTimer = null;

        mLogFileManager = new LogFileManager(mFileName, myFilePath);

        mDiscoveryInfoDlg = new NetDiscoveryInfoDialog();
        mDiscoveryInfoDlg.setNoticeDialogListener(new DiscoveryInfoEvents());
    }

     private OnClickListener mClickListener = new  OnClickListener() {
            public void onClick(View v) {
                        switch(v.getId()) {
                        case R.id.button_detect:
                                detectDaqDevices();
                        break;
                        case R.id.button_connect:
                                connectToDaqDevice();
                        break;
                        case R.id.button_disconnect:
                                stopAInScan();
                                disconnectDaqDevice();
                                updateStatus("Disconnected from " + mDaqDevice, false);
                                break;
                        case R.id.Button01:
                                viewFile();
                                break;
                        case R.id.Emailbtn:
                        emailFile();
                        break;
                        case R.id.toggleButton_start:
                                if(mStartButton.isChecked())
                                {
                                        startAInScan();
//                                  startLog();
                                }
                                else
                                        stopAInScan();

                                break;
                        }
            }
    };
    private void createHeader(DaqDevice daqDev, int lowChan, int highChan, long period)
    {
            daqDev = mDaqDevice;
            lowChan =mLowChan;
            highChan = mHighChan;
            period = mTimerPeriod;

            final Calendar c = Calendar.getInstance();
     //   c.add(Calendar.DATE, "");  // number of days to add
        int newYear =   (c.get(Calendar.YEAR));
       int newMonth = (c.get((Calendar.MONTH)));
     int  newDay = (c.get(Calendar.DATE));
        String newDate = String.valueOf(newMonth) + "/" +String.valueOf(newDay) +"/"+ String.valueOf(newYear);
            try {
                                mFileWriter = new FileWriter(file);
                                mFileWriter.append("Date: ");
                                mFileWriter.append(newDate);
                                mFileWriter.append('\n');
                                mFileWriter.append("Device: ");
                                mFileWriter.append(daqDev.getDescriptor().productName);
                                mFileWriter.append('\n');
```

```
                                        mFileWriter.append("Serial Number: ");
                                        mFileWriter.append(daqDev.getConfig().getSerialNumber());
                                        mFileWriter.append('\n');
                                        mFileWriter.append("Timer Period(ms): ");
                                        mFileWriter.append(Long.toString(period));
                                        mFileWriter.append('\n');
                                        mFileWriter.append('\n');
                                        mFileWriter.append("Time: ");
                                        for(int ch = lowChan; ch <= highChan; ch++) {
                                                mFileWriter.append("channel: " + ch);
                                                if(ch != highChan)
                                                        mFileWriter.append(",");
                                        }
                                        mFileWriter.append('\n');
                                        mFileWriter.flush();
                                        mFileWriter.close();
                                } catch (IOException e) {
                                        e.printStackTrace();
                                }

   }

   private void startLog() throws IOException
   {
       //Called Constructor to create folder and file
            // By clicking the Log button users can create a log file
 myFile = new File(Environment.getExternalStorageDirectory(), mFileName);
                                  try {
                                OS = new FileOutputStream(myFile);
                                        }
                                    catch (FileNotFoundException e)
                                        {
                                                                                        e.printStackTrace();
                                                                                }
                                                                if(myFile.exists())
                                                                        {
                                                                Toast.makeText(getApplicationContext(), "File created at
"+file+" directory", Toast.LENGTH_SHORT).show();
                                                                createHeader(mDaqDevice, mLowChan, mHighChan,
mTimerPeriod);
                                                                writeData(mAInData, mUnit);
                                                                //startAInTimer();
                                                                        }
                                                                else
                                                {
                                                Toast.makeText(getApplicationContext(), "File was not created successfuly",
Toast.LENGTH_SHORT).show();
                                                }



        }
public void writeData(double [] data, AiUnit unit)
{

        String dataStr;
```

```
            // Overwrites existing file
            //mFileWriter = new FileWriter(file);

            for(int i = 0; i < data.length; i++) {
                        if(unit == AiUnit.COUNTS)
                                    dataStr = String.format(Locale.US, "%.0f", data[i]);
                        else
                                    dataStr = String.format(Locale.US, "%.6f", data[i]);

                        try {

                                    if (i == 0) {
                              String time = mTimeFormat.format(Calendar.getInstance().getTime());
                              mFileWriter.append(time);
                              mFileWriter.append(',');
                                    }

                                    mFileWriter.append(dataStr);
                                    if( i != (data.length - 1))
                                                mFileWriter.append(',');
                                    else {
                                                mFileWriter.append('\n');
                                                mFileWriter.flush();
                                                }
                        }
                        catch (Exception e)
                        {
                        //TODO
                        }
            }

}
void viewFile() {

            if(!file.exists()) {
                        Toast.makeText(getApplicationContext(), "File does not exist", Toast.LENGTH_SHORT).show();
                        return;
            }
            if(!file.canRead()) {
                        updateStatus("Unable to read the " + file + " file", true);
                        return;
            }
   Intent intent = new Intent(Intent.ACTION_VIEW);
   Uri data = Uri.fromFile(file);

   intent.setDataAndType(data, "text/csv");

   try {
            startActivity(intent);
   } catch(Exception e) {
            updateStatus("Unable to find an app to handle this operation. Please install a CSV viewer app.", true);
   }

}
void emailFile() {
                        Intent intent = new Intent(Intent.ACTION_SEND);
```

```
                    intent.setType("text/html");
            // TO DO ADD DATE, READ FROM DATE LINE OF DATA.CSV BEING SENT
            intent.putExtra(Intent.EXTRA_SUBJECT, "Logged data");
            intent.putExtra(Intent.EXTRA_TEXT, "");

            if(!file.exists()) {
                    Toast.makeText(getApplicationContext(), "The " + file + " file does not exist",
Toast.LENGTH_SHORT).show();
                    return;

            }

            if(!file.canRead()) {
                    Toast.makeText(getApplicationContext(), "Unable to read the " + file + " file",
Toast.LENGTH_SHORT).show();
                    return;
            }

            Uri uri = Uri.fromFile(file);
            intent.putExtra(Intent.EXTRA_STREAM, uri);
            intent.setType("message/rfc822");

            try {
                    startActivity(Intent.createChooser(intent, "Send email..."));
            } catch(Exception e) {
            updateStatus("Unable to find an app to handle this operation", true);
    }
}

    private void detectDaqDevices() {

            mDaqDevInventoryAdapter.clear();

            // Find available DAQ devices
            ArrayList<DaqDeviceDescriptor> daqDevInventory = mDaqDeviceManager.getDaqDeviceInventory();

            // Add detected DAQ devices to spinner
            mDaqDevInventoryAdapter.addAll(daqDevInventory);

                    if(daqDevInventory.size() > 0)
                            updateStatus(daqDevInventory.size() + " DAQ device(s) detected", false);
                    else
                            updateStatus("No DAQ devices detected", false);

                    updateActivity();
    }

    public class OnDaqDeviceSelectedListener implements OnItemSelectedListener {
            public void onItemSelected(AdapterView<?> parent, View view, int pos, long id) {
                    if(mDaqDevice != null){
                            mDaqDeviceManager.releaseDaqDevice(mDaqDevice);

                            if(mUpdateStatus)
                                    updateStatus("", false);
                            else
                                    mUpdateStatus = true;
```

```
                        }

                        // Create a DaqDevice object for the selected device
                        mDaqDevice =
mDaqDeviceManager.createDaqDevice(mDaqDevInventoryAdapter.getItem(pos));

                        DaqDeviceInfo devInfo = mDaqDevice.getInfo();

                        // Check if this DAQ Device has an analog input device (subsystem)
            if(devInfo.hasAiDev()) {

                        mAiDevice = mDaqDevice.getAiDev();
                        AiInfo aiInfo = mAiDevice.getInfo();

                        mChanModeAdapter.clear();

                        mChanModeAdapter.notifyDataSetInvalidated();

                        // Get supported channel modes
                        EnumSet<AiChanMode> chanModes = aiInfo.getChanModes();
                        mChanModeAdapter.addAll(chanModes);
                        //Tell users that unit has been set to VOLTS
                        Toast.makeText(getApplicationContext(), "Unit Has been set to " + units + " :",
Toast.LENGTH_SHORT).show();

                        mUnitAdapter.clear();

                        // Get supported units
            //          EnumSet<AiUnit> units = mUnit.VOLTS;

                        //AiUnit units = mUnit.VOLTS;
                        mUnitAdapter.addAll(units);


                        // Set Voltage Range to +/- Volts
                        Range selectedRange = Range.BIP5VOLTS;
                        mRangeAdapter.clear();
                        // Get the maximum supported scan rate
                        double maxRate = aiInfo.getMaxScanRate();

                        double rate = Double.parseDouble(mRateEditText.getText().toString());

                        if(rate > maxRate)
                                    mRateEditText.setText(String.valueOf(maxRate));
            }
            else
                        updateStatus("Selected device does not support analog input", true);

            }

            public void onNothingSelected(AdapterView<?> parent){// Do nothing.
            }
    }
    //Call Different Activity Class

    void connectToDaqDevice() {
            updateStatus("Connecting to " + mDaqDevice, false);

                        // Check if this device has connection permission
                        if(mDaqDevice.hasConnectionPermission()) {
                                    // This device already has connection permission. try to connect to it

            mDeviceConnectionPermissionListener.onDaqDevicePermission(mDaqDevice.getDescriptor(), true);
                        }
                        else {
                                    //Request permission for connecting to the selected device
                                    try {

            mDaqDevice.requestConnectionPermission(mDeviceConnectionPermissionListener);
                                    } catch (ULException e) {
                                                updateStatus(e.getMessage(), true);
                                    }
                        }
    }

    public DaqDeviceConnectionPermissionListener mDeviceConnectionPermissionListener = new
DaqDeviceConnectionPermissionListener() {
                public void onDaqDevicePermission(DaqDeviceDescriptor daqDeviceDescriptor, boolean
permissionGranted) {
                        if(permissionGranted) {
                                    try {
                                                //Establish connection to the DAQ device
                                                            mDaqDevice.connect();

                                                            runOnUiThread(new Runnable() {
                                                                        public void run() {
                                                                                    mDetectButton.setEnabled(false);
                                                                        mDaqDevSpinner.setEnabled(false);
                                                                        mConnectButton.setEnabled(false);
                                                                        mDisconnectButton.setEnabled(true);

                                                                        try {
                                                                                    if(mDaqDevice.getInfo().hasAiDev())
                                                                                                mStartButton.setEnabled(true);

                                                                                    updateStatus("Connected to " + mDaqDevice, false);
                                                                                    // Disable screen rotation while a DAQ device is
connected
                                                                                    lockScreenOrientation();

                                                                        } catch(NullPointerException e) {
                                                                                    updateStatus("DaqDevice object no longer valid." +
mDaqDevice, true);
                                                                        }
                                                                 }
                                                            });


                                                } catch (Exception e) {
                                                            updateStatus("Unable to connect to " + mDaqDevice + ". " +
e.getMessage(), true);
                                                }
```

```
                    }
                    else {
                                        updateStatus("Permission denied to connect to " + mDaqDevice, true);
                    }
            }
    };
    // When you click start button all this happens *********START BUTTON***************
    void startAInScan() {
            int lowChan = (Integer) mLowChanSpinner.getSelectedItem();
            int highChan = (Integer) mHighChanSpinner.getSelectedItem();

     //AiChanMode mode = (AiChanMode) mChanModeSpinner.getSelectedItem();
    // AiChanMode mode = AiChanMode.DIFFERENTIAL;

            // Tell users Channel mode has been set to differential
            Toast.makeText(getApplicationContext(), "Channel mode has been to " +mode, 2000).show();

            //Range range = Range.BIP5VOLTS;
            mSamplesToPlot = Integer.parseInt(mSamplesToPlotEditText.getText().toString());
            int samplesPerChan = DEFAULT_SAMPLE_PER_CHAN > mSamplesToPlot * 4 ?
DEFAULT_SAMPLE_PER_CHAN : mSamplesToPlot * 4;
            double rate = Double.parseDouble(mRateEditText.getText().toString());
            EnumSet<AiScanOption> options = EnumSet.of(AiScanOption.DEFAULTIO,
AiScanOption.CONTINUOUS);
            mUnit = (AiUnit) mUnitSpinner.getSelectedItem();

            mChanCount = highChan >= lowChan ? highChan - lowChan + 1 : 1;
            mScanData = new double[mChanCount][samplesPerChan];
            mAInData = new double[mChanCount];
            int resolution = mDaqDevice.getAiDev().getInfo().getResolution();

            // Call to start adding to text
            //addScanDataTextViews(chanCount);

            if(mSamplesToPlot > MAX_SAMPLES_TO_PLOT) {
                    updateStatus("Number of samples to plot is too high. Please set it to " +
MAX_SAMPLES_TO_PLOT + " or less", true);
                    mStartButton.setChecked(false);
                    return;
            }
            if(rate > HIGH_RATE && mSamplesToPlot > MAX_SAMPLES_TO_PLOT_HIGH_RATE) {
                    updateStatus("Number of samples to plot is too high for the specified rate. Please set it to " +
MAX_SAMPLES_TO_PLOT_HIGH_RATE + " or less", true);
                    mStartButton.setChecked(false);
                    return;
            }

            try {
                            @SuppressWarnings("unused")

                            //Collect the values by calling the aInScan function
                            double actualScanRate = mAiDevice.aInScan(lowChan, highChan, mode, range,
samplesPerChan, rate, options, mUnit , mScanData);

                            initPlot(lowChan, highChan, range, resolution);
```

```
                        startScanStatusTimer();

            }
            catch (final ULException e) {
                        updateStatus(e.getMessage(), true);
                        mStartButton.setChecked(false);
                  }
            //startLog();
                  }


    private void scanStatusTimer(){
            try {
                  synchronized(this) {
                        if(mDaqDevice != null) {
                              // Check if the background operation has finished. If it has, then the background
operation must be explicitly stopped

                                    final Status scanStatus = mAiDevice.getStatus();

                                    if(scanStatus.currentStatus != Status.RUNNING) {
                                          // always call stopBackground upon completion...
                                          stopAInScan();

                                          stopScanStatusTimer();

                                          if(scanStatus.errorInfo == ErrorInfo.DEADDEV) {
                                                disconnectDaqDevice();
                                          }
                                    }

                                    displayScanData(scanStatus);
                        }
                  }

                  } catch (final ULException e) {
                        stopScanStatusTimer();

                        updateStatus(e.getMessage(), true);
                        e.printStackTrace();
                  }
    }
private void startAInTimer() {


            long timerPeriod = 100;

            if(timerPeriod >= 10) {

              Timer mAInTimer = new Timer();
                        mAInTimer.schedule(new TimerTask() {
                     @Override
                     public void run() {
                        startAInTimer();
                     }
```

```
                        }, 0, timerPeriod);
            } else {
                        mStartButton.setChecked(false);

                        updateStatus("Timer period must be greater than or equal " + 10, true);

            }
    }
    void displayScanData(final Status scanStatus) {
                        runOnUiThread(new Runnable() {
                                    public void run() {

                                                if(scanStatus.currentIndex >= 0) {
                                                            if(scanStatus.currentIndex > mNextPlotIndex || (mNextPlotIndex -
scanStatus.currentIndex) > mSamplesToPlot) {

                                                                        int j = 0;
                                                            for(int ch = 0; ch < mScanData.length; ch++) {
                                                                        j = 0;

                                                                        for(int i = mNextPlotIndex - mSamplesToPlot; i <
mNextPlotIndex; i++) {
                                                                                    mPlotData[ch][j] = mScanData[ch][i];
                                                                                    j++;
                                                                        }
                                                            }

                                                            DataChart.plot(mPlotData);

                                                            mNextPlotIndex += mSamplesToPlot;
                                                            if(mNextPlotIndex > mScanData[0].length)
                                                                        mNextPlotIndex = mSamplesToPlot;
                                                            }
                                                try {
                                                            startLog();
                                                } catch (IOException e) {
                                                            // TODO Auto-generated catch block
                                                            e.printStackTrace();
                                                }
                                                }

                                                if(scanStatus.currentStatus == Status.IDLE) {
                                                            if(scanStatus.errorInfo != ErrorInfo.NOERROR)
                                                                        updateStatus(scanStatus.errorInfo.toString(), true);
                                                            else
                                                                        updateStatus("Scan stopped", false);

                                                            mStartButton.setChecked(false);
                                                }
                                                else if(mDaqDevice != null && mDaqDevice.isConnected()){
                                                            updateStatus("Scan is running. Number of samples acquired : " +
scanStatus.currentCount, false);
                                                }
                                    }
                        });
    }

    void initPlot(int lowChan, int highChan, Range range, int resolution) {

            mNextPlotIndex = mSamplesToPlot;
                        mPlotData = new double[mChanCount][mSamplesToPlot];
                        DataChart.init(this, lowChan, highChan, mSamplesToPlot, range, mUnit , resolution);

                        // Tell users range is restricted to +-5V
                        Toast.makeText(getApplicationContext(), "Voltage Range has been restricted to " + range,
3000).show();

                        // scroll to the bottom of the view to show the results
            final ScrollView scrollview=((ScrollView) findViewById(R.id.scrollView1));
            scrollview.post(new Runnable() {

          @Override
          public void run() {
            mRateEditText.setFocusableInTouchMode(false);
            mSamplesToPlotEditText.setFocusableInTouchMode(false);
            scrollview.fullScroll(ScrollView.FOCUS_DOWN);
            mRateEditText.setFocusableInTouchMode(true);
            mSamplesToPlotEditText.setFocusableInTouchMode(true);
          // Make Screen Larger when Tapped on Screen

          }
        });

    }


    void stopAInScan() {
            try {
                        mAiDevice.stopBackground();
                        } catch (ULException e) {
                                    e.printStackTrace();
                        }
    }


    private void disconnectDaqDevice() {
            mDaqDevice.disconnect();

            runOnUiThread(new Runnable() {
                                    public void run() {

                        mDetectButton.setEnabled(true);
            mDaqDevSpinner.setEnabled(true);
            mConnectButton.setEnabled(true);
            mDisconnectButton.setEnabled(false);
            mStartButton.setChecked(false);
            mStartButton.setEnabled(false);

            setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_UNSPECIFIED);
                                    }
            });
```

```
}

public class OnChanModeSelectedListener implements OnItemSelectedListener {
	public void onItemSelected(AdapterView<?> parent, View view, int pos, long id) {

			AiInfo aiInfo = mDaqDevice.getAiDev().getInfo();

			mLowChanAdapter.clear();
			mHighChanAdapter.clear();

			// Get number of analog input channels for the selected channel mode
			int numChannels = aiInfo.getNumChans(mChanModeAdapter.getItem(pos));

			for(int chan = 0; chan < numChannels; chan++ )
			{
				mLowChanAdapter.add(chan);
				mHighChanAdapter.add(chan);
			}
			//Hard coded range to +- 5Volts
			Range selectedRange = Range.BIP5VOLTS;
			//Range selectedRange = (Range) mRangeSpinner.getSelectedItem();
			mRangeAdapter.clear();

			// Get supported ranges for the specified channel mode
			EnumSet<Range> ranges = aiInfo.getRanges(mChanModeAdapter.getItem(pos));

			mRangeAdapter.addAll(ranges);

			// set the range to current selected range if the new mode supports it

			int rangePos = mRangeAdapter.getPosition(selectedRange);
			if(rangePos != -1)
				mRangeSpinner.setSelection(rangePos);
			else
				mRangeSpinner.setSelection(0);

	}

	public void onNothingSelected(AdapterView<?> parent){// Do nothing.
	}
}

private void initActivity() {
	mDetectButton = (Button)findViewById(R.id.button_detect);
	mConnectButton = (Button)findViewById(R.id.button_connect);
	mDisconnectButton = (Button)findViewById(R.id.button_disconnect);

	mDetectButton.setOnClickListener(mClickListener);
	mDetectButton.setOnLongClickListener((OnLongClickListener) mLongClickListener);

	mConnectButton.setOnClickListener(mClickListener);
	mConnectButton.setEnabled(false);

	mDisconnectButton.setOnClickListener(mClickListener);
	mDisconnectButton.setEnabled(false);
```

```
        mDaqDevSpinner = (Spinner) findViewById(R.id.spinner_daqDev);
        mDaqDevInventoryAdapter =  new ArrayAdapter <DaqDeviceDescriptor> (this,
android.R.layout.simple_spinner_item );

        mDaqDevInventoryAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_ite
m);
        mDaqDevSpinner.setAdapter(mDaqDevInventoryAdapter);
        mDaqDevSpinner.setOnItemSelectedListener(new OnDaqDeviceSelectedListener());
        mDaqDevSpinner.setEnabled(false);

        mLowChanSpinner = (Spinner) findViewById(R.id.spinner_lowChan);
        mLowChanAdapter =  new ArrayAdapter <Integer> (this, android.R.layout.simple_spinner_item );
        mLowChanAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        mLowChanSpinner.setAdapter(mLowChanAdapter);
        mLowChanSpinner.setEnabled(false);

        mHighChanSpinner = (Spinner) findViewById(R.id.spinner_highChan);
        mHighChanAdapter =  new ArrayAdapter <Integer> (this, android.R.layout.simple_spinner_item );
        mHighChanAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        mHighChanSpinner.setAdapter(mHighChanAdapter);
        mHighChanSpinner.setEnabled(false);

        mChanModeSpinner = (Spinner) findViewById(R.id.spinner_chanMode);
        mChanModeAdapter =  new ArrayAdapter <AiChanMode> (this, android.R.layout.simple_spinner_item );
        mChanModeAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        mChanModeSpinner.setAdapter(mChanModeAdapter);
        mChanModeSpinner.setOnItemSelectedListener(new OnChanModeSelectedListener());
        mChanModeSpinner.setEnabled(false);

        mSamplesToPlotEditText =  (EditText)findViewById(R.id.editText_samplestoPlot);
        mSamplesToPlotEditText.setEnabled(false);

        mRateEditText =  (EditText)findViewById(R.id.editText_rate);
                mRateEditText.setText(String.valueOf(DEFAULT_RATE));
        mRateEditText.setEnabled(false);

        mRangeSpinner = (Spinner) findViewById(R.id.spinner_range);
        mRangeAdapter =  new ArrayAdapter <Range> (this, android.R.layout.simple_spinner_item );
        mRangeAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        mRangeSpinner.setAdapter(mRangeAdapter);
        mRangeSpinner.setEnabled(false);

        mUnitSpinner = (Spinner) findViewById(R.id.spinner_unit);
        mUnitAdapter =  new ArrayAdapter <AiUnit> (this, android.R.layout.simple_spinner_item );
        mUnitAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        mUnitSpinner.setAdapter(mUnitAdapter);
        mUnitSpinner.setEnabled(false);

        mStatusTextView = (TextView) findViewById(R.id.textView_errInfo);

        mStartButton = (ToggleButton) findViewById(R.id.toggleButton_start);
        mStartButton.setOnClickListener(mClickListener);
        mStartButton.setEnabled(false);


        Button Button01 = (Button)findViewById(R.id.Button01);
```

```
            Button01.setOnClickListener(mClickListener);

            Button emailFileButton = (Button) findViewById(R.id.Emailbtn);
            emailFileButton.setOnClickListener(mClickListener);


            this.getWindow().setSoftInputMode(WindowManager.LayoutParams.SOFT_INPUT_STATE_ALWAYS_
HIDDEN);

            updateStatus("Tap detect button. (If you need to detect a network DAQ device manually, press and hold
detect button)", false);

            findViewById(R.id.mainLayout).setFocusableInTouchMode(true);
            findViewById(R.id.mainLayout).requestFocus();
   }

   private void startScanStatusTimer() {
            updateStatus("Scan is running", false);

            mScanStatusTimer = new Timer();
                      mScanStatusTimer.schedule(new TimerTask() {
                 @Override
                 public void run() {
                      scanStatusTimer();
                 }

              }, 0, TIMER_PERIOD);
   }

   private void stopScanStatusTimer() {
            if(mScanStatusTimer != null)
                      mScanStatusTimer.cancel();
   }

   private void lockScreenOrientation()  {

              int orientation = ActivityInfo.SCREEN_ORIENTATION_UNSPECIFIED;
              Display display = ((WindowManager)
getSystemService(Context.WINDOW_SERVICE)).getDefaultDisplay();
              int rotation = display.getRotation();
              Configuration cfg = getResources().getConfiguration();

              switch (rotation) {
              case Surface.ROTATION_0:
                      if(cfg.orientation == Configuration.ORIENTATION_PORTRAIT)
                                orientation = ActivityInfo.SCREEN_ORIENTATION_PORTRAIT;
                      else if(cfg.orientation == Configuration.ORIENTATION_LANDSCAPE)
                                orientation = ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE;
                 break;
              case Surface.ROTATION_90:
                      if(cfg.orientation == Configuration.ORIENTATION_LANDSCAPE)
                                orientation = ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE;
                      else if(cfg.orientation == Configuration.ORIENTATION_PORTRAIT)
                                orientation = ActivityInfo.SCREEN_ORIENTATION_REVERSE_PORTRAIT;
                 break;
               default:
```

```
            case Surface.ROTATION_180:
                  if(cfg.orientation == Configuration.ORIENTATION_PORTRAIT)
                           orientation = ActivityInfo.SCREEN_ORIENTATION_REVERSE_PORTRAIT;
                  else if(cfg.orientation == Configuration.ORIENTATION_LANDSCAPE)
                            orientation = ActivityInfo.SCREEN_ORIENTATION_REVERSE_LANDSCAPE;
              break;
            case Surface.ROTATION_270:
                  if(cfg.orientation == Configuration.ORIENTATION_LANDSCAPE)
                            orientation = ActivityInfo.SCREEN_ORIENTATION_REVERSE_LANDSCAPE;
                  else if(cfg.orientation == Configuration.ORIENTATION_PORTRAIT)
                           orientation = ActivityInfo.SCREEN_ORIENTATION_PORTRAIT;
              break;
            }

            setRequestedOrientation(orientation);
}

void updateStatus(final String message, final boolean error) {
      runOnUiThread(new Runnable() {
                           public void run() {
                                    int textColor = GREEN;
                           if(error)
                                    textColor = Color.RED;

                           mStatusTextView.setTextColor(textColor);
                                    mStatusTextView.setText(message);
                           }
      });
}

private void updateActivity() {

      if(!mDaqDevInventoryAdapter.isEmpty()) {
                  mDaqDevSpinner.setEnabled(true);
                  mConnectButton.setEnabled(true);
      mLowChanSpinner.setEnabled(true);
      mHighChanSpinner.setEnabled(true);
      mChanModeSpinner.setEnabled(true);
      mRangeSpinner.setEnabled(true);
      mUnitSpinner.setEnabled(true);
      mSamplesToPlotEditText.setEnabled(true);
      mRateEditText.setEnabled(true);

      mUpdateStatus = false;

      if(mDaqDevSpinner.getSelectedItemPosition() != 0)
                  mDaqDevSpinner.setSelection(0);
      else
                  mDaqDevSpinner.getOnItemSelectedListener().onItemSelected(null, null, 0, 0);
      }
      else {
                           mConnectButton.setEnabled(false);
                           mDisconnectButton.setEnabled(false);
                           mDaqDevSpinner.setEnabled(false);
                  }
}
```

```
   private void detectNetDaqDeviceManually(String host, int port) {

           mDaqDevInventoryAdapter.clear();

           DaqDeviceDescriptor netDevDescriptor = null;

           int discoveryTimeout = 10000; // ms

                      try {
                                 netDevDescriptor = mDaqDeviceManager.getNetDaqDeviceDescriptor(null, host, port,
discoveryTimeout);

                      if(netDevDescriptor != null) {
                                 mDaqDevInventoryAdapter.add(netDevDescriptor);
                                 updateStatus(netDevDescriptor.productName + " device detected", false);
                      }
                      else
                                 updateStatus("No network DAQ devices detected", false);

                                 updateActivity();

                      } catch (ULException e) {
                                 updateStatus(e.getMessage(), true);
                      }
   }

   class DiscoveryInfoEvents implements NoticeDialogListener{

                      @Override
                      public void onDialogPositiveClick(DialogFragment dialog) {

                                 String host = mDiscoveryInfoDlg.getHost();
                                 int port = mDiscoveryInfoDlg.getPort();
                                 detectNetDaqDeviceManually(host, port);
                      }

                      @Override
                      public void onDialogNegativeClick(DialogFragment dialog) {
                      }

   }

   private OnLongClickListener mLongClickListener = new OnLongClickListener() {

                      @Override
                      public boolean onLongClick(View v) {
                                 AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(v.getContext());

                                 alertDialogBuilder.setMessage("Would you like to detect a network DAQ device
manually?");
                                 alertDialogBuilder.setCancelable(false);
                                 alertDialogBuilder.setPositiveButton("Yes",new DialogInterface.OnClickListener() {
                                            public void onClick(DialogInterface dialog,int id) {

                                                       mDiscoveryInfoDlg.show(getFragmentManager(), "InfoTag");
```

```
                                }
                        });

                        alertDialogBuilder.setNegativeButton("No",new DialogInterface.OnClickListener() {
                                public void onClick(DialogInterface dialog,int id) {
                                        dialog.cancel();
                                }
                        });

                        AlertDialog alertDialog = alertDialogBuilder.create();
                        alertDialog.show();

                        return true;
                }
    };


        @Override
    protected void onDestroy() {
        stopScanStatusTimer();

        synchronized(this) {
                if(mDaqDevice != null){
                                mDaqDeviceManager.releaseDaqDevice(mDaqDevice);
                }

                mDaqDevice = null;
        }

        // Wake locks should be released in onPause, however in order to keep the system awake while
        // scan is running and the application is minimized, the release method is called here
        if(mWakeLock != null) {
                mWakeLock.release();
                mWakeLock = null;
                }


       super.onDestroy();
    }
         void displayAndLogData() {

                        runOnUiThread(new Runnable() {
                                public void run() {

                                        int chanCount = mAInData.length;

                                        // display the scan data from each channel
                                for(int i = 0; i < chanCount; i++) {

                                        if( mUnit== AiUnit.COUNTS)
                                                mDataTextView[i].setText(String.format("%.0f",
mAInData[i]));
                                        else
                                                mDataTextView[i].setText(String.format("%.6f",
mAInData[i]));
                                }
```

```
                                        if(mSamplesRead >= mSamplesPerChan) {
                                                    stopAInTimer();
                                        }

                                                    updateStatus("Number of samples acquired : " + mSamplesRead,
false);

                                        }
                              });

                }
                private void stopAInTimer() {
                        if(mScanStatusTimer != null)
                                  mScanStatusTimer.cancel();

                        mStartButton.setChecked(false);
                        mFileNameEditText.setEnabled(true);
                }
}
```

DataChart.Java

```
package com.mcc.ul.example.ainscan.plot;
import com.mcc.ul.example.ainscan.plot.AInScanPlotActivity;
import java.util.ArrayList;
import java.util.List;

import android.os.Bundle;
import android.view.Display;
import android.view.Window;
import android.view.WindowManager;

import org.achartengine.ChartFactory;
import org.achartengine.GraphicalView;
import org.achartengine.chart.PointStyle;
import org.achartengine.model.XYMultipleSeriesDataset;
import org.achartengine.model.XYSeries;
import org.achartengine.renderer.XYMultipleSeriesRenderer;
import org.achartengine.renderer.XYSeriesRenderer;

import com.mcc.ul.AiUnit;
import com.mcc.ul.Range;

import android.app.Activity;
import android.content.Context;
import android.graphics.Color;
import android.graphics.Paint.Align;
import android.widget.LinearLayout;

public class DataChart
{

            static GraphicalView mChartView;
            static XYMultipleSeriesDataset mDataset;
            public static void init(Activity activity, int lowChan, int highChan, int sampleCount, Range range, AiUnit
unit, int resolution) //double[][] dataValues)
```

```
      {

                  int chanCount = highChan - lowChan + 1;
                  String[] titles = new String[chanCount];
                  int [] colors = new int[chanCount];
    PointStyle[] styles = new PointStyle[chanCount];

    boolean showGrid = true;
    // Setting Range to +/-10 Volts to Have space implement Log
    range = Range.BIP5VOLTS;
     //double minValue = (unit == AiUnit.COUNTS) ? 0 : range.getMinValue();
       //double maxValue = (unit == AiUnit.COUNTS) ? Math.pow(2, resolution) - 1 : range.getMaxValue();
    double minValue = -5.00;
    double maxValue = 5.00;


                  for(int i = 0; i < chanCount; i++) {
                        titles[i] = "Chan " + (i + lowChan);
                        colors[i] = getColor(i);
                        styles[i] = PointStyle.POINT;
                  }

    List<double[]> x = new ArrayList<double[]>();
    List<double[]> values = new ArrayList<double[]>();

    for(int ch = 0; ch < chanCount; ch++)
       x.add(new double[sampleCount]);

    for (int i = 0; i < sampleCount; i++)
       for(int ch = 0; ch < chanCount; ch++)
                  x.get(ch)[i] = i;

    for(int i = 0; i < chanCount; i++) {
       double dataValue[] = new double[sampleCount];

       for (int j = 0; j < sampleCount; j++)
                  dataValue[j] = minValue;

       values.add(dataValue);


    }



    XYMultipleSeriesRenderer renderer = buildRenderer(colors, styles);
    int length = renderer.getSeriesRendererCount();
    for (int i = 0; i < length; i++) {
      ((XYSeriesRenderer) renderer.getSeriesRendererAt(i)).setFillPoints(true);
    }
    // Set Chart UI Data
    setChartSettings(renderer, "Force VS Depth", "Depth(mm)", "Force(lbs)", 0, sampleCount - 1, minValue,
maxValue,
                  Color.WHITE, Color.WHITE, Color.WHITE, Color.WHITE, showGrid);

    mDataset =  buildDataset(titles, x, values);
    mChartView =  ChartFactory.getLineChartView(activity, mDataset, renderer);
```

```
        mChartView.setBackgroundColor(Color.BLACK);

        Display display = ((WindowManager)
activity.getSystemService(Context.WINDOW_SERVICE)).getDefaultDisplay();
        int height = display.getHeight();
        int width = display.getWidth();

      ;

        int chartMinHeight;

        if(height > width)
              chartMinHeight = height/3;

        else
              chartMinHeight = height/2;

        mChartView.setMinimumHeight(chartMinHeight);

        LinearLayout layout = (LinearLayout) activity.findViewById(R.id.layout_plot);
        layout.removeAllViews();

        LinearLayout.LayoutParams layoutParams = new LinearLayout.LayoutParams(
                                        LinearLayout.LayoutParams.MATCH_PARENT,
LinearLayout.LayoutParams.WRAP_CONTENT);

        layout.addView(mChartView, layoutParams);

              }
              static XYMultipleSeriesDataset buildDataset(String[] titles, List<double[]> xValues,
                    List<double[]> yValues)
              {
                  XYMultipleSeriesDataset dataset = new XYMultipleSeriesDataset();
                  addXYSeries(dataset, titles, xValues, yValues, 0);
                  return dataset;
               }

  static void addXYSeries(XYMultipleSeriesDataset dataset, String[] titles, List<double[]> xValues,
                    List<double[]> yValues, int scale)
  {
                  int length = titles.length;
                  for (int i = 0; i < length; i++) {
                    XYSeries series = new XYSeries(titles[i], scale);
                    double[] xV = xValues.get(i);
                    double[] yV = yValues.get(i);
                    int seriesLength = xV.length;
                    for (int k = 0; k < seriesLength; k++) {
                      series.add(xV[k], yV[k]);
                    }
                    dataset.addSeries(series);
                  }
  }

  static void plot(double[][] DataValues)
  {
                for(int i = 0; i < mDataset.getSeries().length; i++)
```

```
                    mDataset.getSeriesAt(i).clear();

            for(int i = 0; i < mDataset.getSeries().length; i++)
                    for (int j = 0; j < DataValues[0].length; j++)
                            mDataset.getSeriesAt(i).add(j, DataValues[i][j]);

    mChartView.repaint();
}

static XYMultipleSeriesRenderer buildRenderer(int[] colors, PointStyle[] styles)
{
    XYMultipleSeriesRenderer renderer = new XYMultipleSeriesRenderer();

    setRenderer(renderer, colors, styles);
    return renderer;
  }

static void setRenderer(XYMultipleSeriesRenderer renderer, int[] colors, PointStyle[] styles)
{
    renderer.setAxisTitleTextSize(16);
    renderer.setChartTitleTextSize(20);
    renderer.setLabelsTextSize(15);
    renderer.setLegendTextSize(15);
    renderer.setPointSize(5f);
    renderer.setMargins(new int[] { 20, 30, 15, 20 });
    int length = colors.length;
    for (int i = 0; i < length; i++) {
      XYSeriesRenderer r = new XYSeriesRenderer();
      r.setColor(colors[i]);
      r.setPointStyle(styles[i]);
      renderer.addSeriesRenderer(r);
    }
  }

static void setChartSettings(XYMultipleSeriesRenderer renderer, String title, String xTitle,
            String yTitle, double xMin, double xMax, double yMin, double yMax, int axesColor,
            int labelsColor, int xLablesColor, int yLablesColor, boolean showGrid)
{
            renderer.setChartTitle(title);
            renderer.setXTitle(xTitle);
            renderer.setYTitle(yTitle);
            renderer.setXAxisMin(xMin);
            renderer.setXAxisMax(xMax);
            renderer.setYAxisMin(yMin);
            renderer.setYAxisMax(yMax);
            renderer.setAxesColor(axesColor);
            renderer.setLabelsColor(labelsColor);

            renderer.setYLabelsAlign(Align.RIGHT);

            int[] margins = {40,70,10,50};
            renderer.setMargins(margins);
            renderer.setMarginsColor(Color.LTGRAY);

            renderer.setBackgroundColor(Color.BLACK);
```

```
                renderer.setShowGrid(showGrid);
                renderer.setXLabelsColor(xLablesColor);
                renderer.setYLabelsColor(0, yLablesColor);
                renderer.setXLabels(11);
                renderer.setYLabels(11);
  }

  static int getColor(int index) {
            //int color = Color.CYAN;
              int color = Color.WHITE;
            switch(index % 5) {
            case 0:
                        //color = Color.CYAN;
                        color = Color.WHITE;
                        break;
            case 1:
                        color = Color.YELLOW;
                        break;
            case 2:
                        color = Color.GREEN;
                        break;
            case 3:
                        //color = Color.WHITE;
                        color = Color.BLUE;
                        break;
            case 4:
                        color = Color.RED;
                        break;
            }
            return color;
  }

}
```

LogFileManager

```
package com.mcc.ul.example.ainscan.plot;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Locale;


import com.mcc.ul.AiUnit;
import com.mcc.ul.DaqDevice;
import android.annotation.SuppressLint;
import android.os.Environment;

@SuppressLint("DefaultLocale")
public class LogFileManager {
	String fileName="data.csv";

	String mFileName;
	File mFolder;
	File mFile;
	String mFilePath;
	SimpleDateFormat mDateFormat;
	SimpleDateFormat mTimeFormat;
	FileWriter mFileWriter;
	LogFileManager(String folder, String fileName) {
		//mFolder = new File(Environment.getExternalStoragePublicDirectory(fileName),"");
		mFolder = new File (Environment.getExternalStorageDirectory(), File.separator + folder);
		mFileName = fileName;
		mFilePath = mFolder + File.separator + fileName;
				//mFile = new File(mFilePath);
		mFile = new File(mFilePath);

		mDateFormat = new SimpleDateFormat("dd-MMM-yyyy", Locale.US);
		mTimeFormat = new SimpleDateFormat("hh:mm:ss.SSS aaa", Locale.US);
	}

	public void setFileName(String fileName) {
		mFileName = fileName;
		mFilePath = mFolder + File.separator + fileName;
		mFile = new File(mFilePath);
	}

	public String getFileName() {
		return mFileName;
	}

	 public boolean fileExists() {
		if(mFile.exists())
			return true;
		else
			return false;
	 }
```

```
         public boolean createHeader(DaqDevice daqDev, int lowChan, int highChan, long period) {

                    boolean created = true;
        if (!mFolder.exists())
                    mFolder.mkdir();

    String date = mDateFormat.format(Calendar.getInstance().getTime());
    String time = mTimeFormat.format(Calendar.getInstance().getTime());

        try {
                              mFileWriter = new FileWriter(mFilePath);
                              mFileWriter.append("Date: ");
                              mFileWriter.append(date);
                              mFileWriter.append('\n');
                              mFileWriter.append("Time: ");
                              mFileWriter.append(time);
                              mFileWriter.append('\n');
                              mFileWriter.append("Device: ");
                              mFileWriter.append(daqDev.getDescriptor().productName);
                              mFileWriter.append('\n');
                              mFileWriter.append("Serial Number: ");
                              mFileWriter.append(daqDev.getConfig().getSerialNumber());
                              mFileWriter.append('\n');
                              mFileWriter.append("Timer Period(ms): ");
                              mFileWriter.append(Long.toString(period));
                              mFileWriter.append('\n');
                              mFileWriter.append('\n');
                              mFileWriter.append("Time: ");
                              for(int ch = lowChan; ch <= highChan; ch++) {
                                        mFileWriter.append("channel " + ch);
                                        if(ch != highChan)
                                                  mFileWriter.append(",");
                              }

                              mFileWriter.append('\n');
                              mFileWriter.flush();

                    } catch (IOException e) {
                              created = false;
                              e.printStackTrace();
                    }

        return created;
}


        boolean writeData(double[] data, AiUnit unit) {
                    boolean written = false;

                    if(data == null || mFileWriter == null || !mFile.exists())
                              return written;

                    String dataStr;

                    for(int i = 0; i < data.length; i++) {
```

```
                    if(unit == AiUnit.COUNTS)
                            dataStr = String.format(Locale.US, "%.0f", data[i]);
                    else
                            dataStr = String.format(Locale.US, "%.6f", data[i]);

                    try {

                            if (i == 0) {
                                String time = mTimeFormat.format(Calendar.getInstance().getTime());
                                mFileWriter.append(time);
                                mFileWriter.append(',');
                            }

                                    mFileWriter.append(dataStr);
                                    if( i != (data.length - 1))
                                            mFileWriter.append(',');
                                    else {
                                            mFileWriter.append('\n');
                                            mFileWriter.flush();
                                            written = true;
                                    }

                            } catch (IOException e) {
                                    written = false;
                            }
            }

                    return written;
            }

            public static boolean isValidCsvFile(String fileName) {
            boolean valid = false;
            String filenameArray[] = fileName.split("\\.");
        String extension = filenameArray[filenameArray.length-1];

        if(extension.equalsIgnoreCase("csv"))
            valid = true;

        return valid;
    }

}
```

NetDiscoveryInfoDialog

```
package com.mcc.ul.example.ainscan.plot;

import android.app.AlertDialog;
import android.app.Dialog;
import android.app.DialogFragment;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.ContextThemeWrapper;
import android.view.LayoutInflater;
import android.view.View;
import android.widget.EditText;
```

```
public class NetDiscoveryInfoDialog extends DialogFragment {

	NoticeDialogListener mListener = null;
	String mHost = "192.168.0.1";
	int mPort = 54211;

	EditText mEditText_host;
	EditText mEditText_port;

	@Override
	public Dialog onCreateDialog(Bundle savedInstanceState) {

		AlertDialog.Builder builder = new AlertDialog.Builder(new ContextThemeWrapper(getActivity(), 
android.R.style.Theme_Holo_Light_Dialog_MinWidth));

		LayoutInflater inflater = getActivity().getLayoutInflater();

		View view = inflater.inflate(R.layout.discovery_info, null);
		builder.setView(view);

		mEditText_host = (EditText) view.findViewById(R.id.editText_host);
		mEditText_host.setText(mHost);

		mEditText_port = (EditText) view.findViewById(R.id.editText_port);
		mEditText_port.setText(Integer.toString(mPort));

		builder.setPositiveButton(R.string.net_detect_text, new DialogInterface.OnClickListener() {
			public void onClick(DialogInterface dialog, int id) {

			if(mListener != null) {
				mHost = mEditText_host.getText().toString();
				mPort = Integer.parseInt(mEditText_port.getText().toString());
				mListener.onDialogPositiveClick(NetDiscoveryInfoDialog.this);
				}
			}
		});

		builder.setNegativeButton(R.string.cancel_text, new DialogInterface.OnClickListener() {
			public void onClick(DialogInterface dialog, int id) {

				if(mListener != null)
					mListener.onDialogNegativeClick(NetDiscoveryInfoDialog.this);
			}
		});
		return builder.create();
	}

	public void setNoticeDialogListener(NoticeDialogListener noticeDialogListener) {
		mListener = noticeDialogListener;
	}


	public String getHost() {
		return mHost;
	}
	public int getPort() {
		return mPort;
	}

	public interface NoticeDialogListener {
		public void onDialogPositiveClick(DialogFragment dialog);
		public void onDialogNegativeClick(DialogFragment dialog);
	}
}
```

13. Appendix C

Failure Modes and Effects Analysis

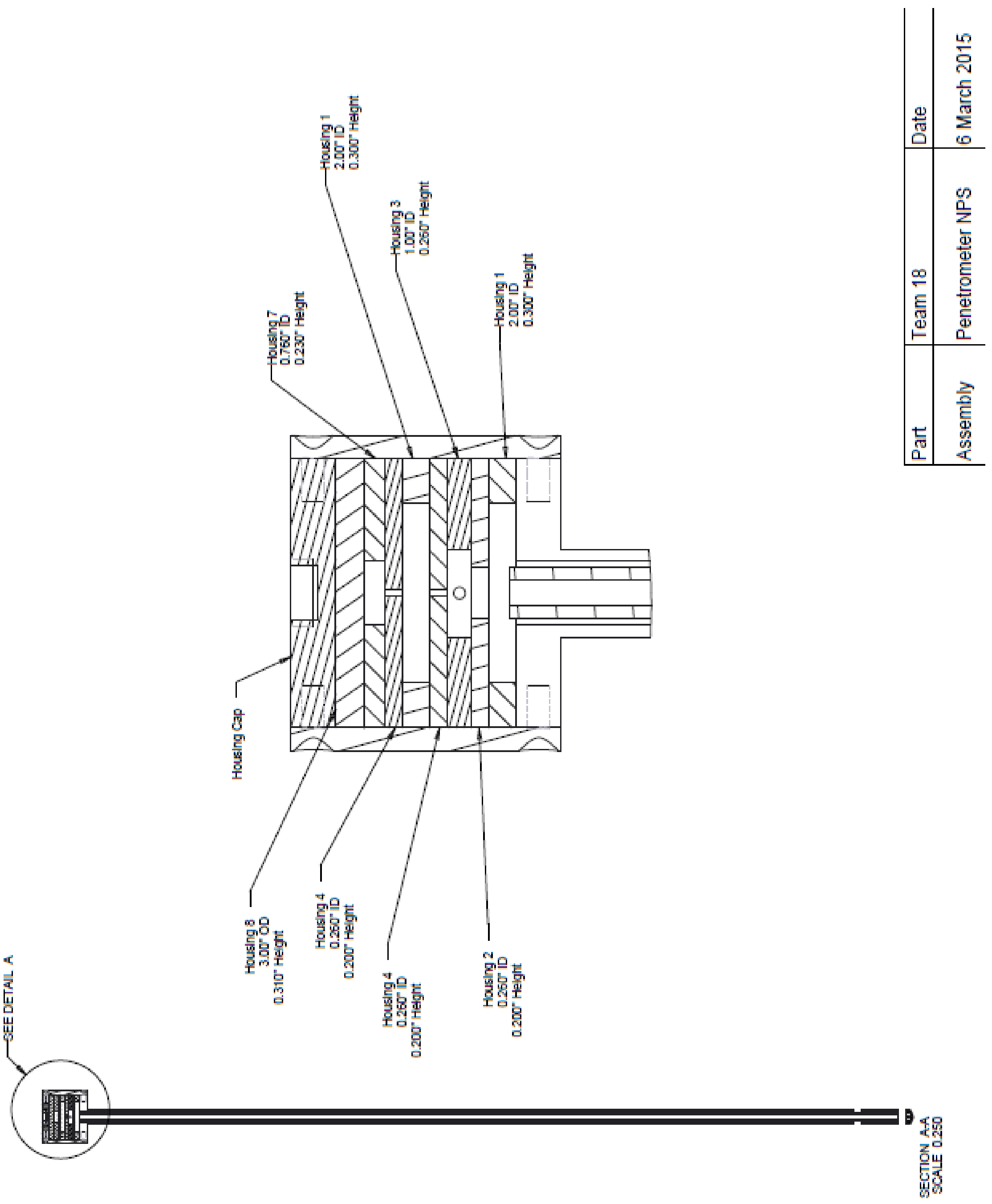
Table 2. Failure Modes and Effects Analysis

Process Function	Potential Failure Mode	Potential Failure Effects	SEV	Potential Causes	OCC	Current Controls	DET	RPN	Recommended Action	Action Taken
Cone tip impacts with soil	Cone tip breaks	False readings, debris into the system, seals brake	9	Assembled incorrectly, weak material, shreaded screw	1	Use strong material and ensure proper installation	2	18	Have an extra cone tip in case of damage	Visually check cone tip before each test
Penetrometer impacts with soil	Seals buckle and break	False readings, debris in the system, linear alignment is bent throughout the rods	7	Assembled incorrectly, not enough sealant used, material not strong enough	4	Use a strong sealant and give it proper time to set when installed	3	84	Test seals in the lab before ground testing them, have extra sealing material around to ensure the correct amount is used	Visually check seals before each test
Friction sleeve slides through soil	Seals are not elastic enough to move with friction	No friction readings from the upper load cell	7	Poor choice in sealing material	2	Test seals in the lab to ensure they are elastic enough	1	14	Test seals in the lab for elasticity	Check readings in lab before field testing
Penetrometer impacts with soil	Alignment of rods is bent	False readings, no readings, too much friction between shafts	6	Not enough support between rods, seals aren't straight, housing is unaligned	4	Very carefully construct device, tight clearnaces	3	72	Check at each step of assembly for alignment and test results for accuracy	Careful installation
Penetrometer impacts with soil	Housing unaligned	False readings, no readings	6	Housing disks don't have tight enough clearances, installed wrong	1	Machine properly, check clearances during assembly	1	6	Machine properly, check clearances during assembly	Machine properly, check clearances during assembly
App setup	DAQ is not in Bluetooth pairing mode	No readings	1	DAQ not in Bluetooth pairing mode	2	App indicates if there is a connection with the DAQ or not	1	2	Hold the button on the DAQ for 3 seconds until green flashing lights are seen	Hold the button on the DAQ for 3 seconds until green flashing lights are seen
DAQ setup	Low DAQ battery	No readings	3	AA batteries in the DAQ are low, not charged	3	Charge the bateries when not in use	1	9	Charge the batteries when not in use, carry spare AA batteries	Replace the AA batteries with the spare ones, and charge the dead batteries
Power source	22.2V battery is low	False readings, no readings	3	Battery pack is low, not charged	4	Charge the bateries when not in use	1	12	Charge the batteries when not in use, carry spare battery pack	Replace the battery pack with the spare one, and charge the dead battery

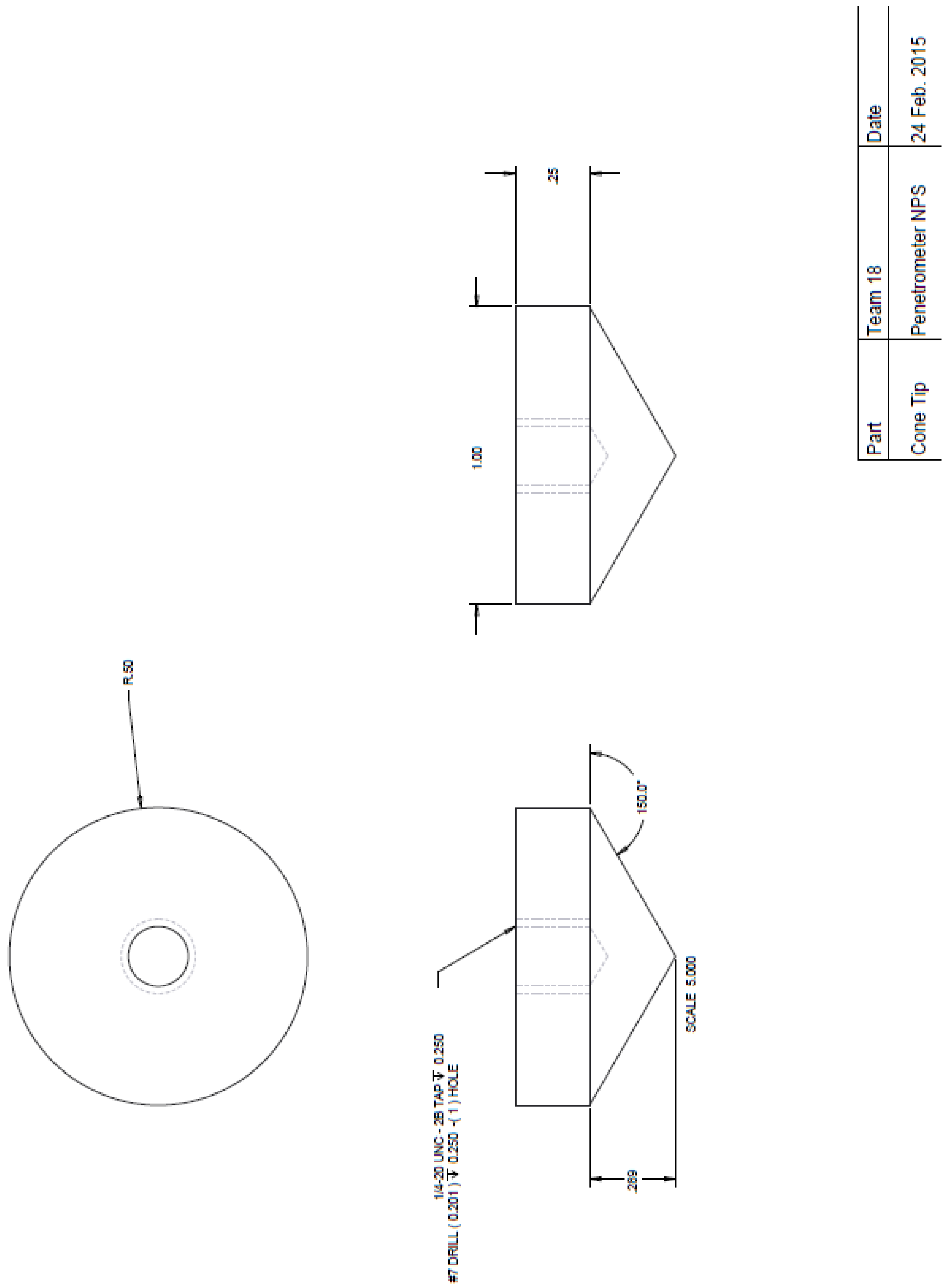
14. Appendix D

ProE Drawings

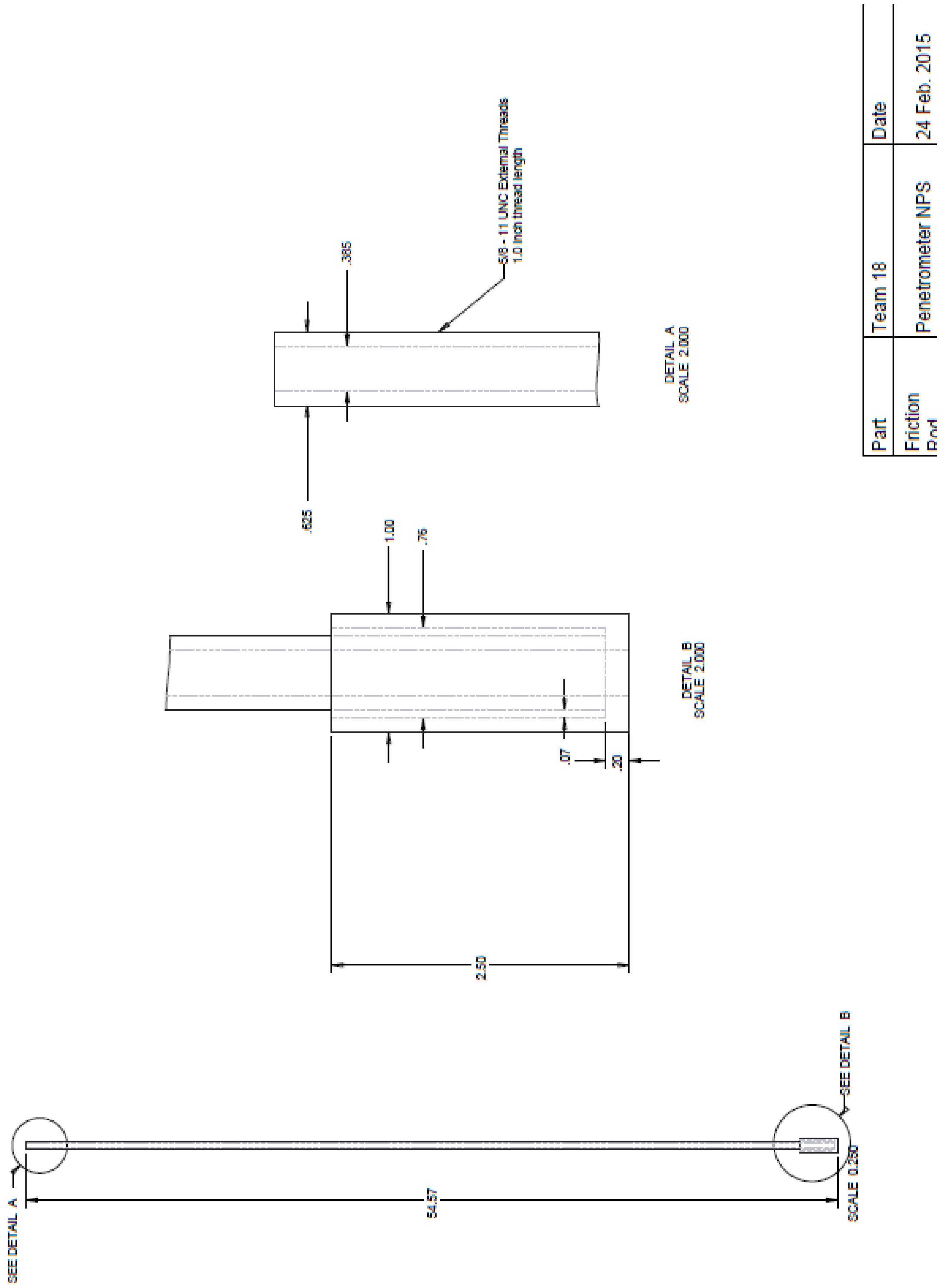
D – Pro E Drawings


Part
Assembly
Team 18
Penetrometer NPS
Date
6 March 2015
Housing 1 2.00" ID 0.300" Height
Housing 3 1.00" ID 0.250" Height
Housing 7 0.750" ID 0.230" Height
Housing 1 2.00" ID 0.300" Height
Housing Cap
Housing 8 3.00" OD 0.310" Height
Housing 4 0.260" ID 0.200" Height
Housing 4 0.260" ID 0.200" Height
Housing 2 0.260" ID 0.200" Height
SEE DETAIL A
SECTION A-A SCALE 0.250

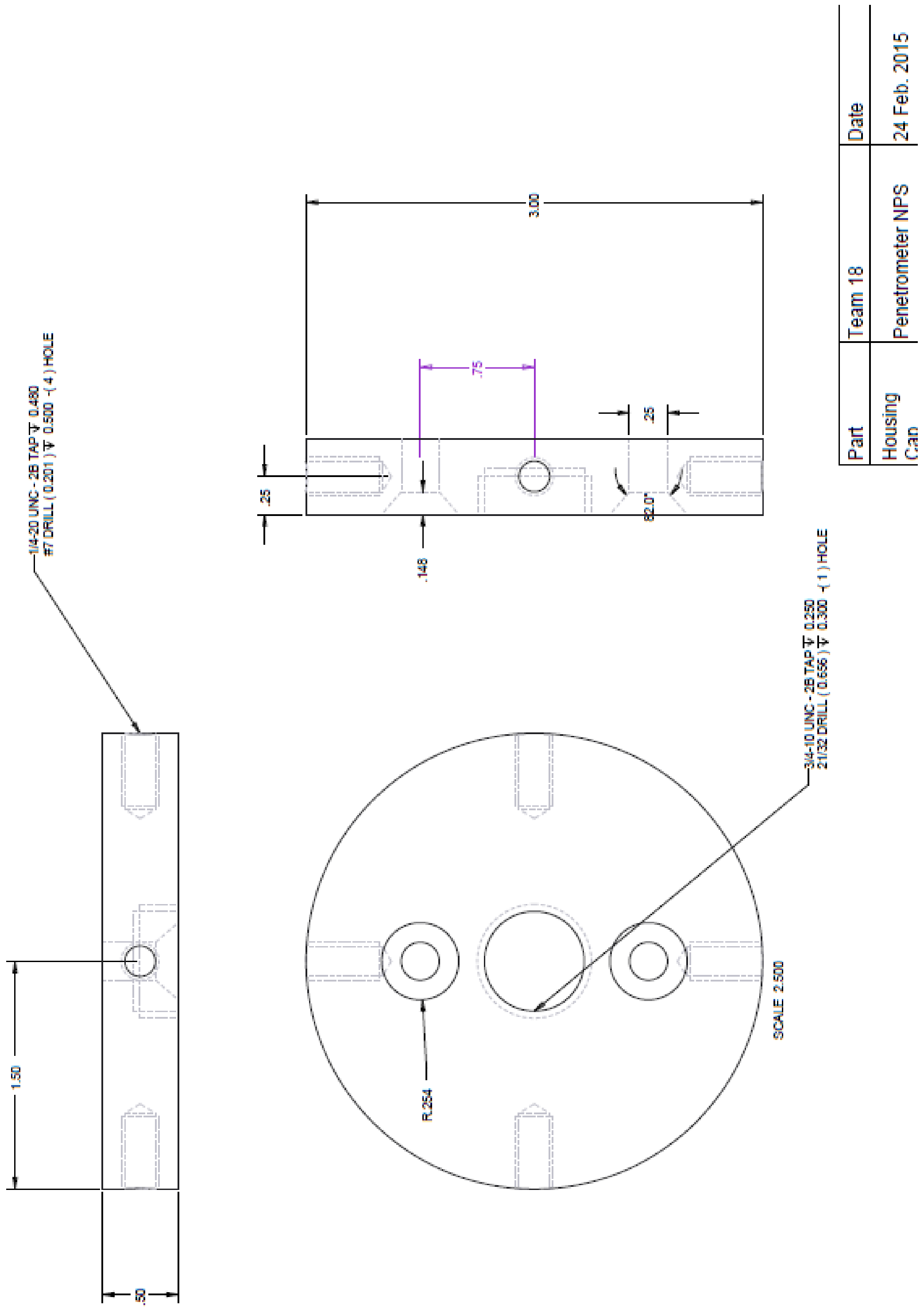
D – Pro E Drawings



D – Pro E Drawings

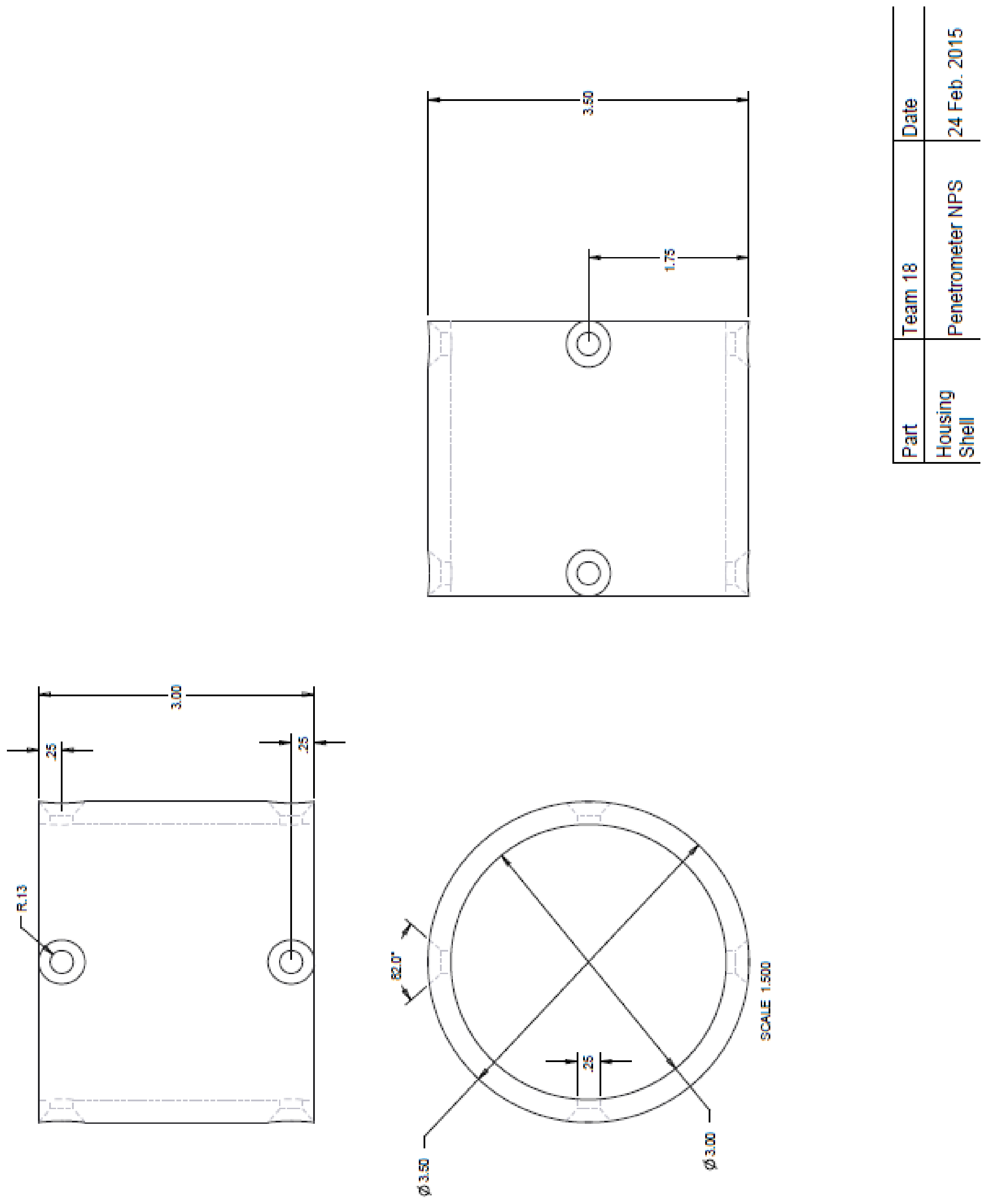


D – Pro E Drawings



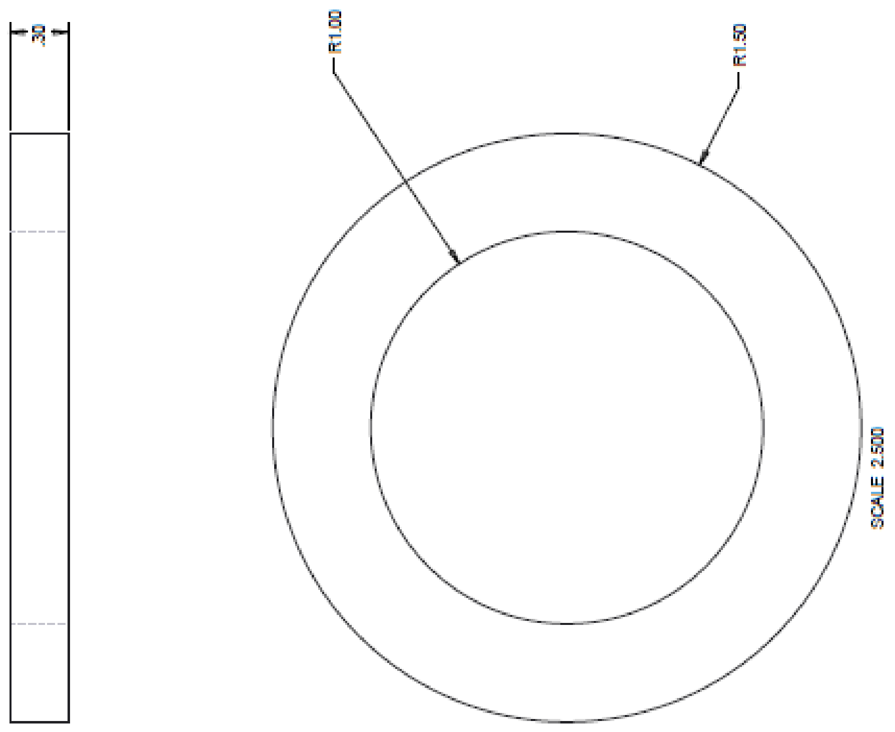
1/4-20 UNC - 2B TAP ↧ 0.480
#7 DRILL (0.201) ↧ 0.500 - (4) HOLE
3.00
.75
.25
.25
.148
82.0°
3/4-10 UNC - 2B TAP ↧ 0.250
21/32 DRILL (0.656) ↧ 0.300 - (1) HOLE
1.50
.50
R.254
SCALE 2.500
Part
Housing Cap
Team 18
Penetrometer NPS
Date
24 Feb. 2015

D – Pro E Drawings

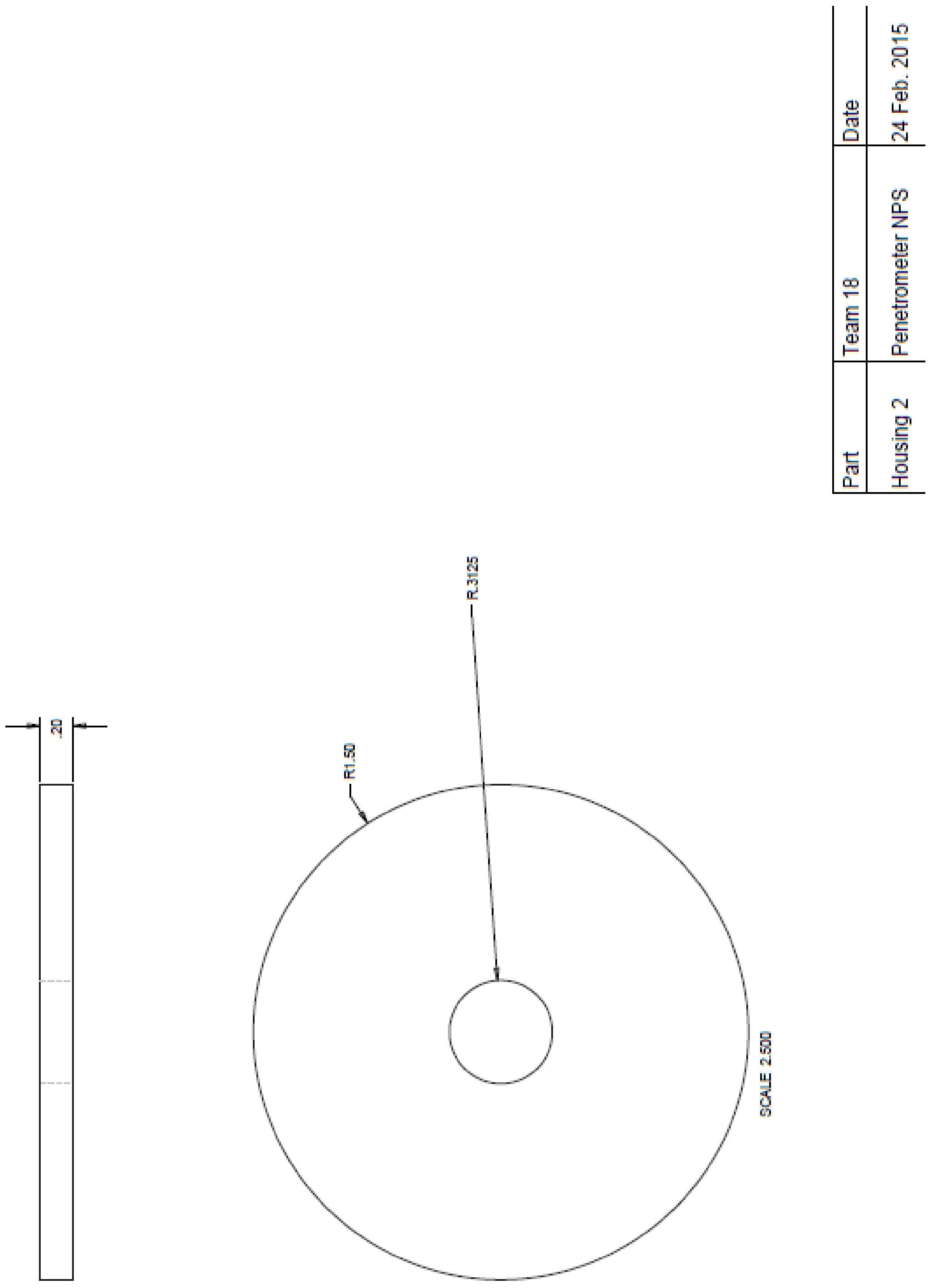


D – Pro E Drawings

Part	Team 18	Date
Housing 1	Penetrometer NPS	24 Feb. 2015

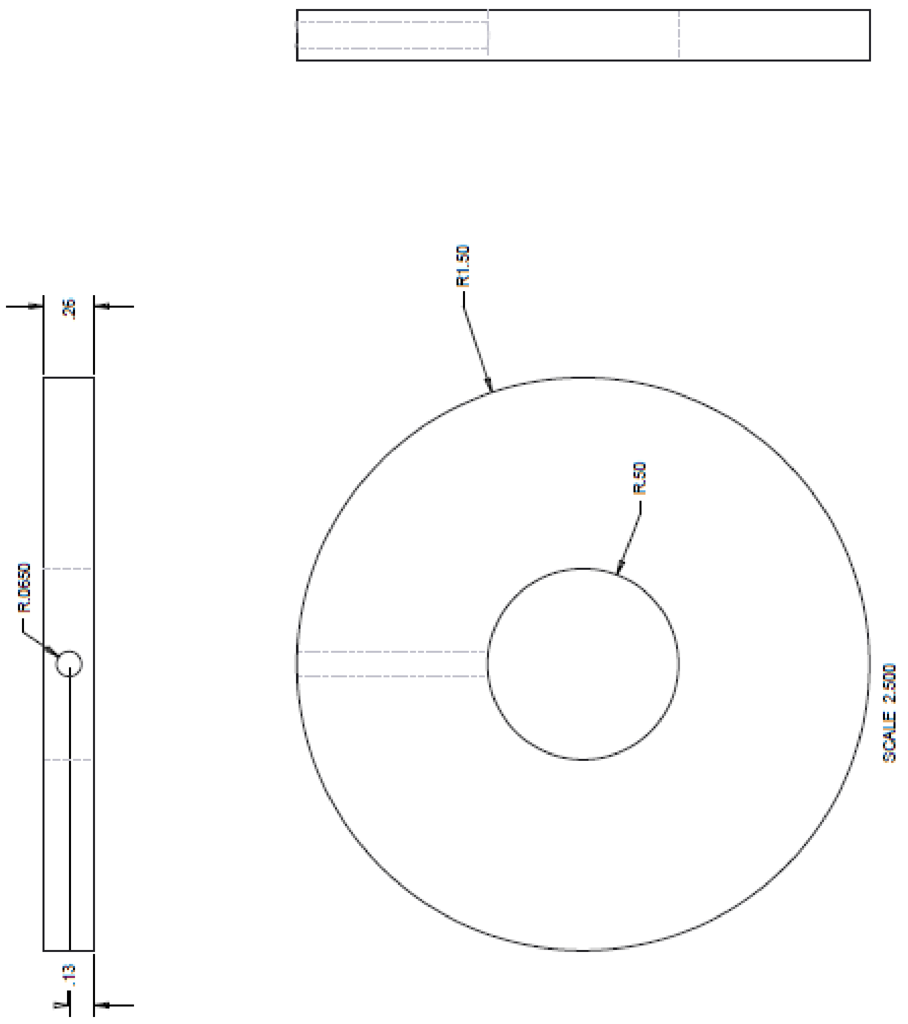


D – Pro E Drawings

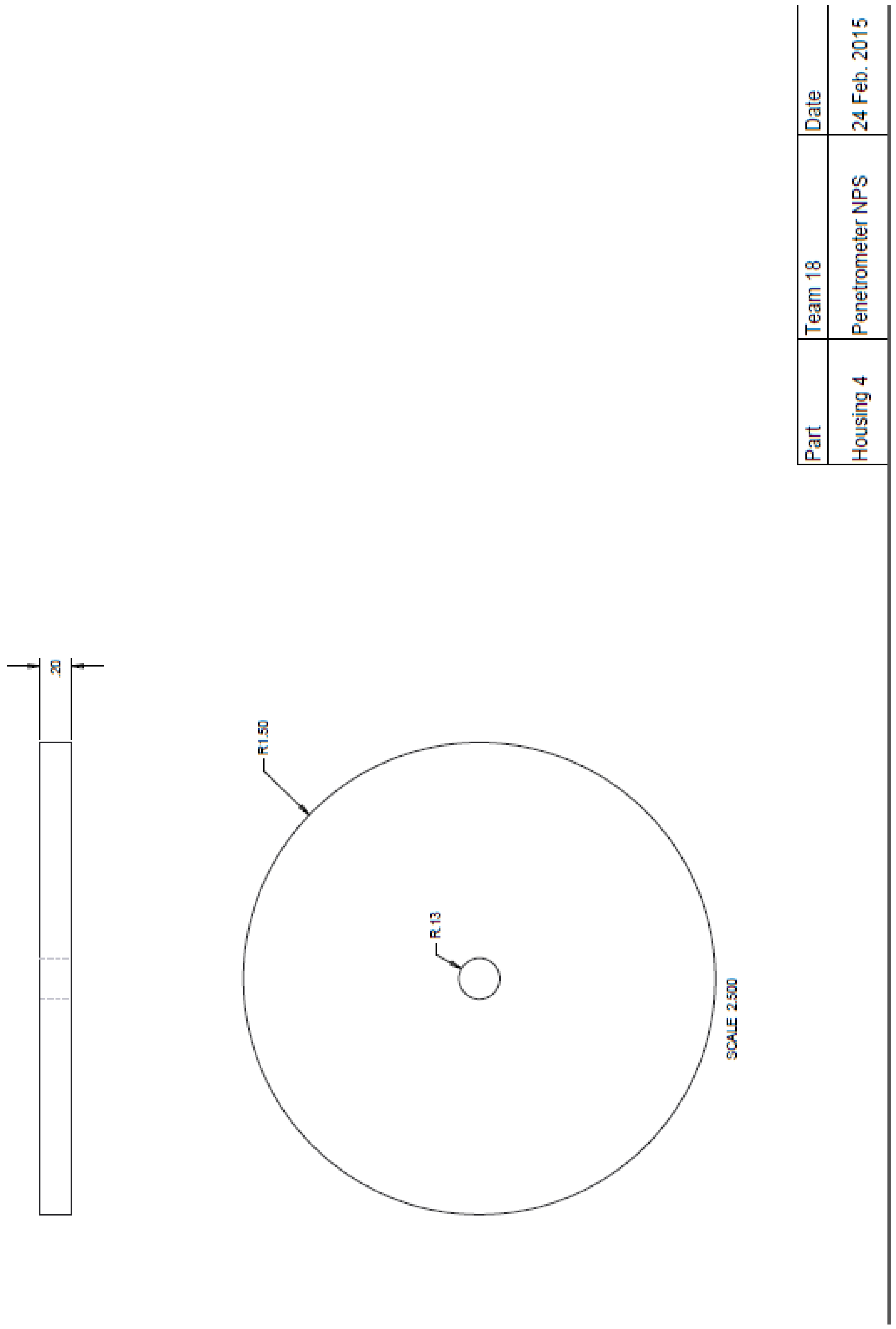


D – Pro E Drawings

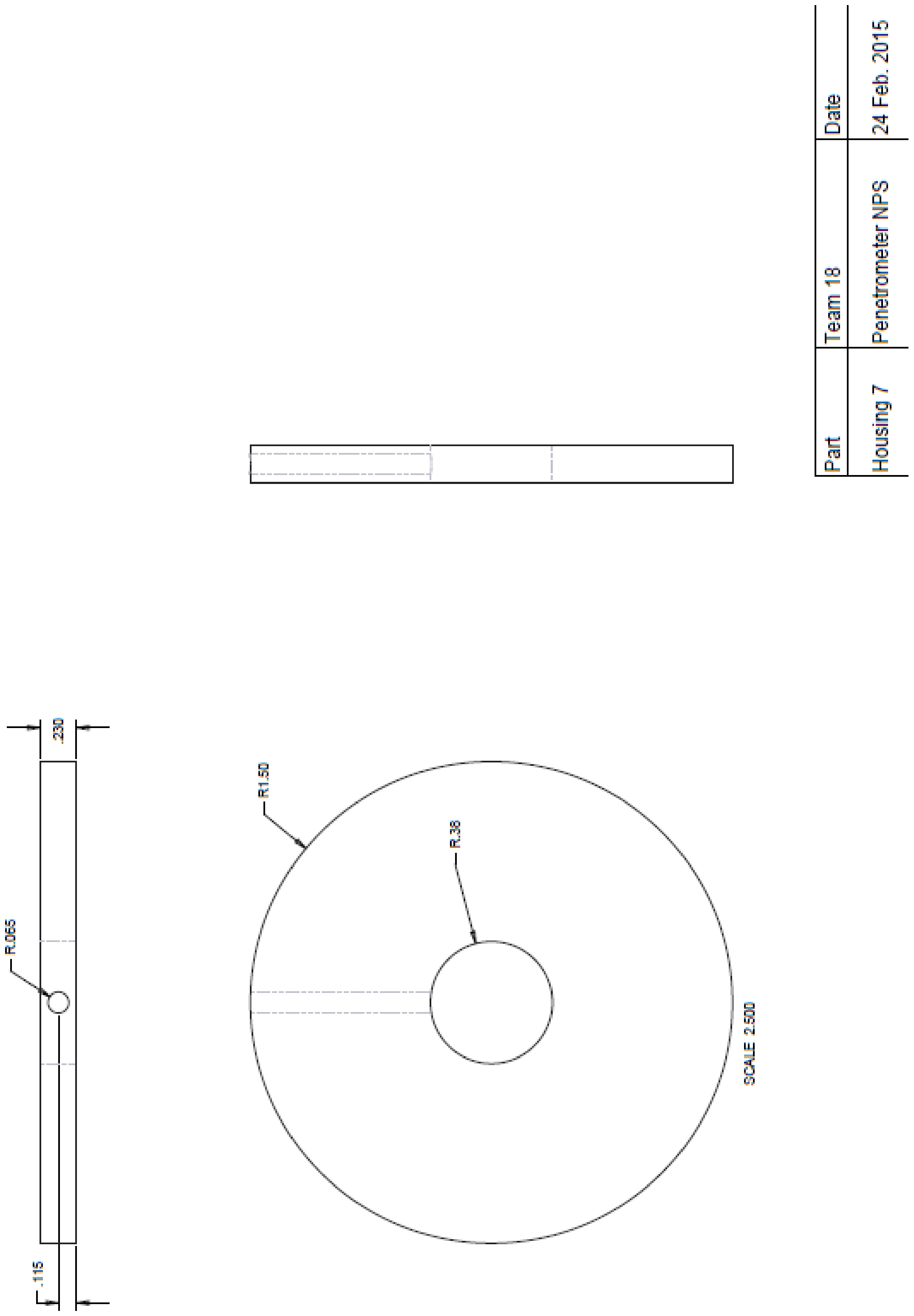
Part	Team 18	Date
Housing 3	Penetrometer NPS	24 Feb. 2015



D – Pro E Drawings

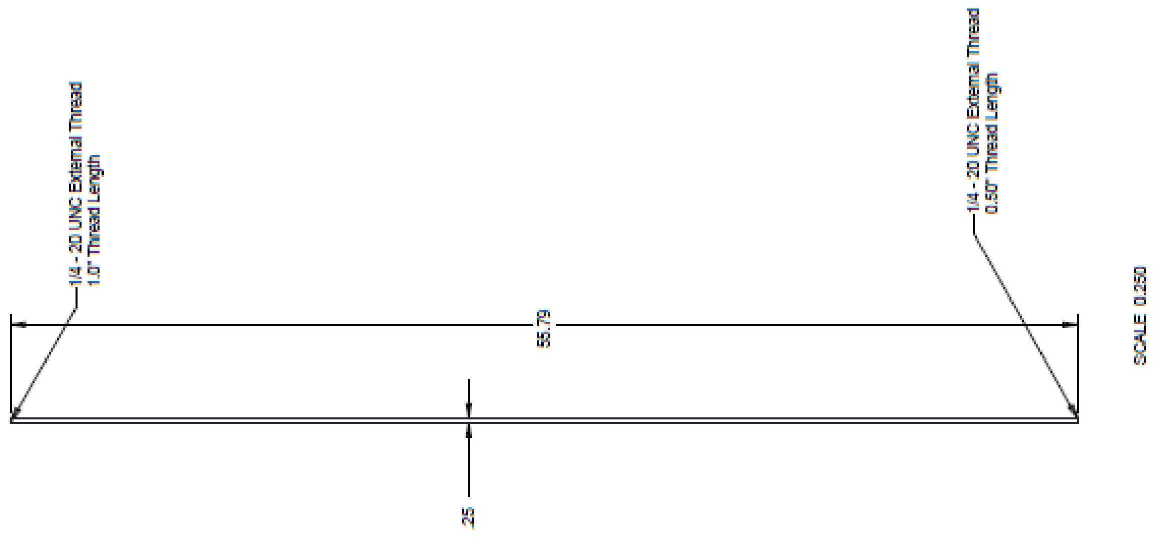


D – Pro E Drawings

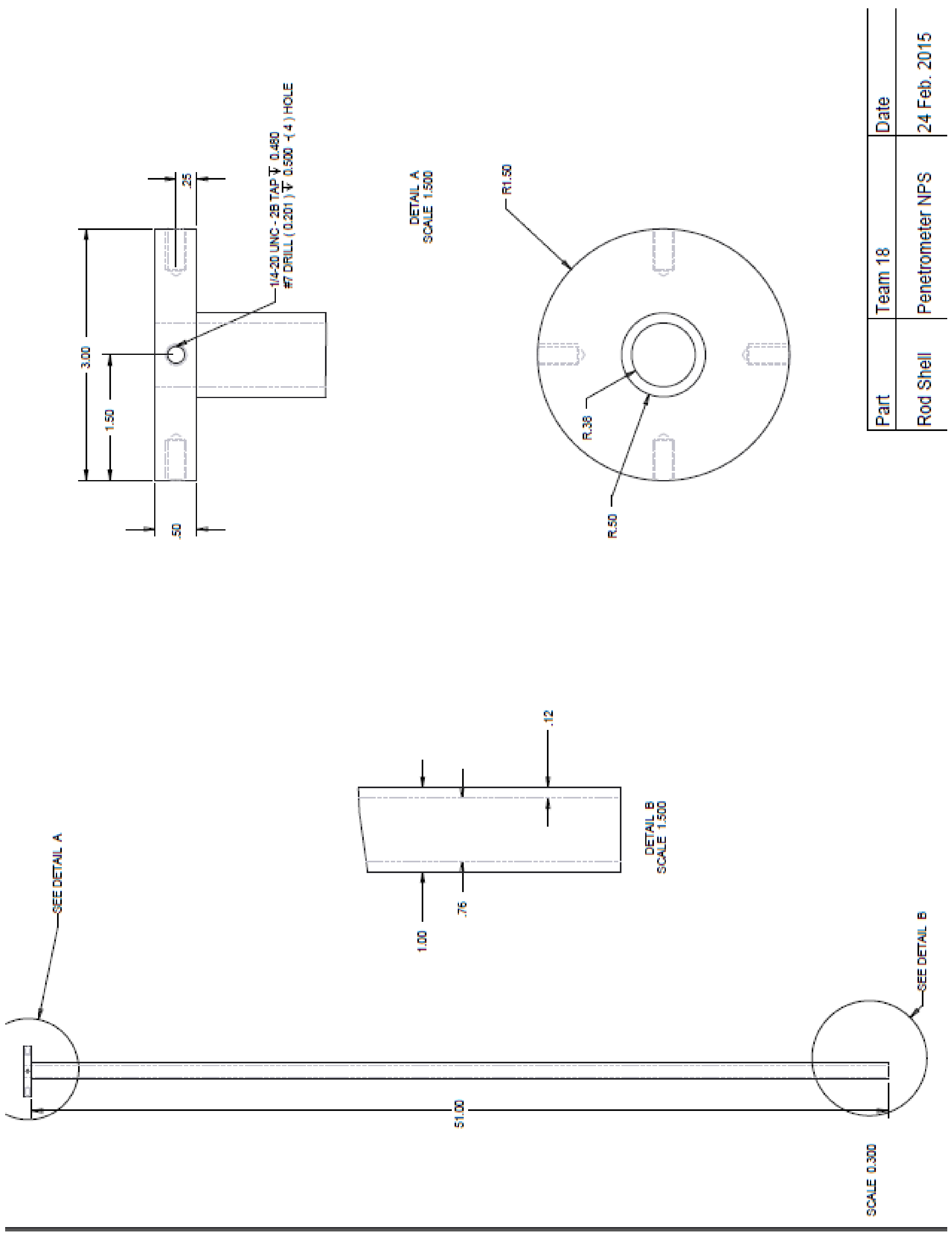


D – Pro E Drawings

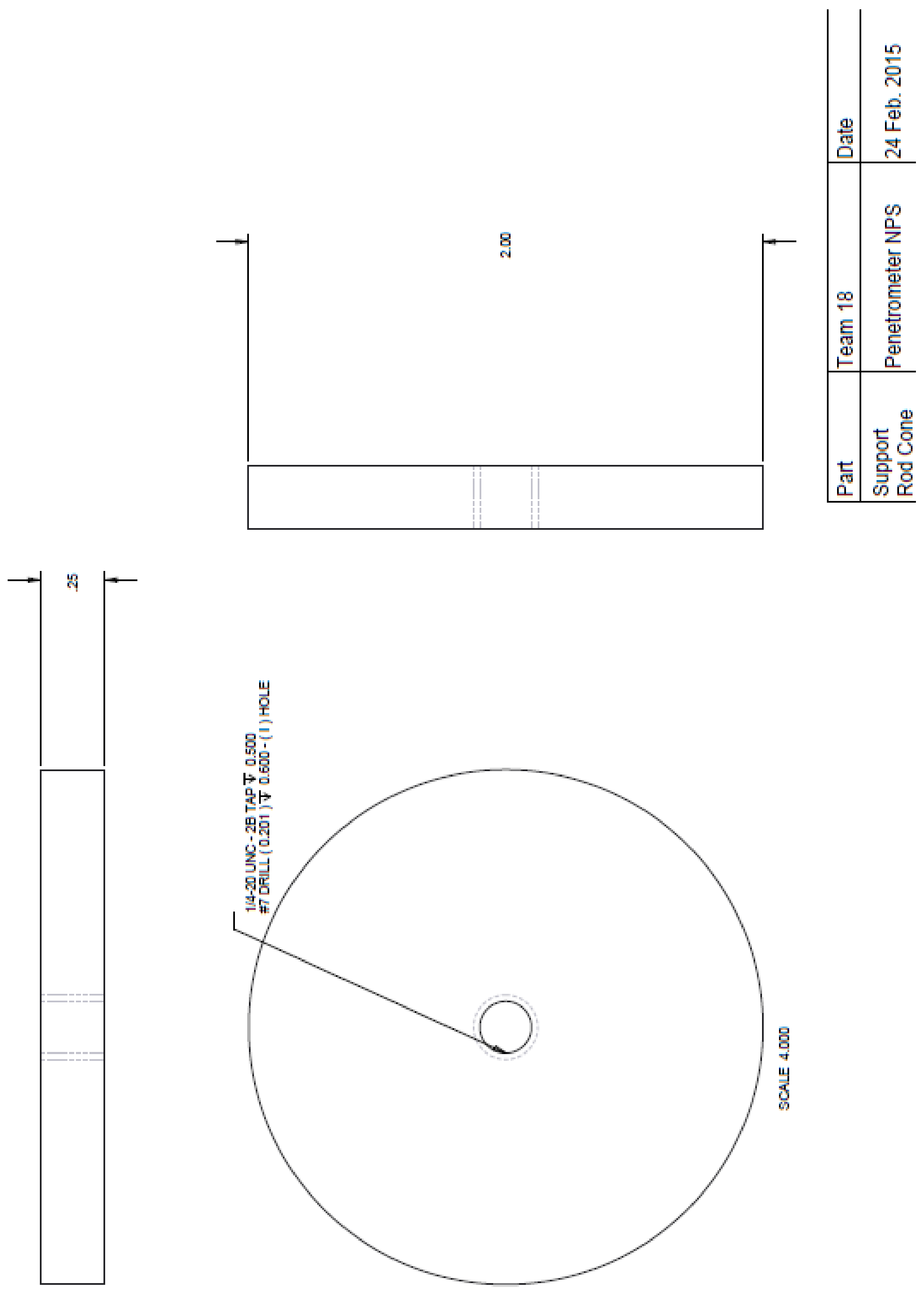
Part	Team 18	Date
Cone Rod	Penetrometer NPS	24 Feb. 2015



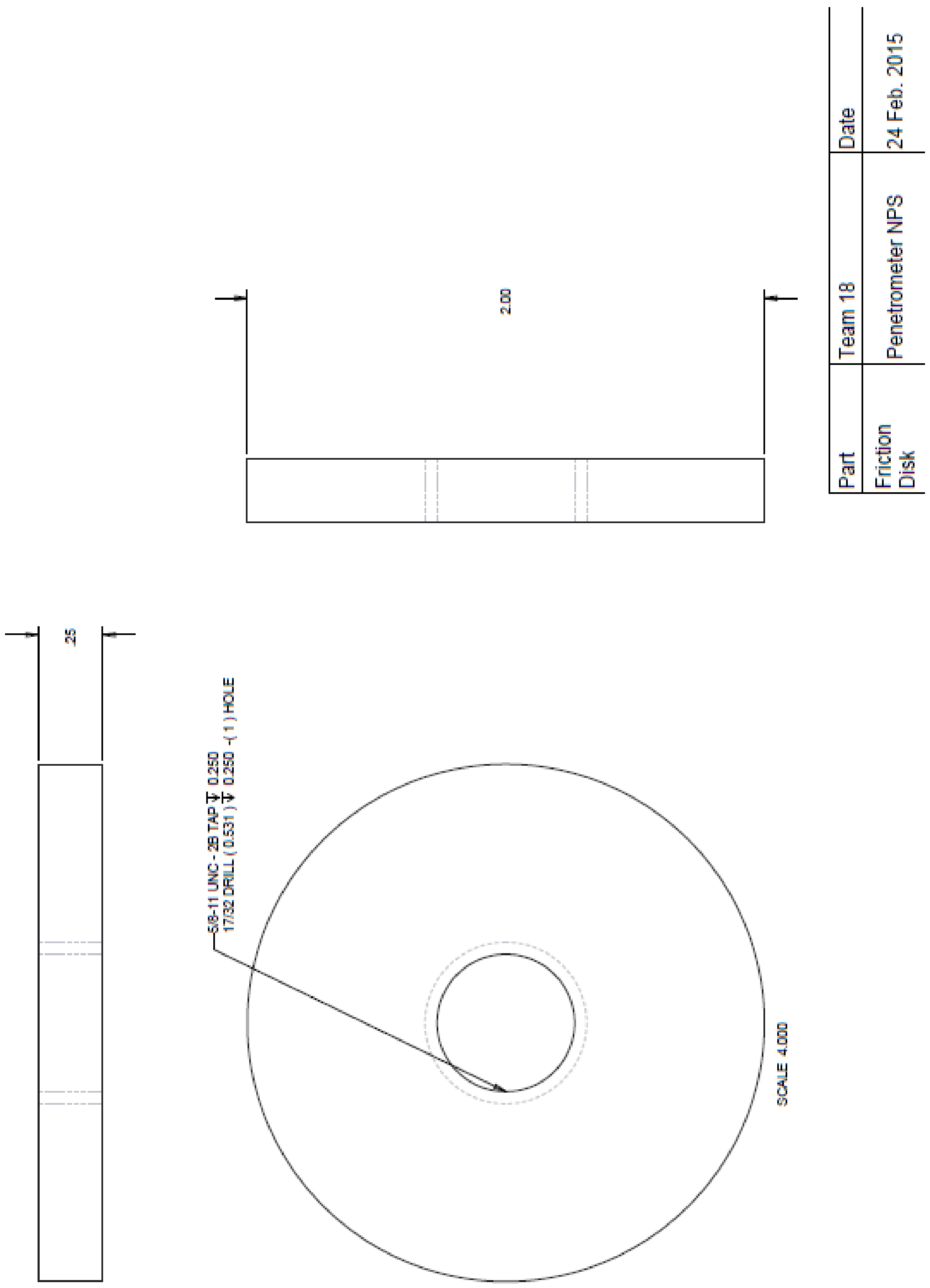
D – Pro E Drawings



D – Pro E Drawings



D – Pro E Drawings



Part
Friction Disk
Team 18
Penetrometer NPS
Date
24 Feb. 2015
2.00
.25
5/8-11 UNC - 2B TAP ↧ 0.250
17/32 DRILL (0.531) ↧ 0.250 -(1) HOLE
SCALE 4.000