# Design and Manufacturing of Control System for Monochromator

James Adams

Physics 492R Capstone Project Report

August 2020

Dr. R Steven Turley, Advisor

Department of Physics and Astronomy

Brigham Young University

# Abstract

Our team is focused on testing mirrors and other phenomena in the Extreme Ultraviolet (XUV). For this, we need a way to select wavelengths used for testing. A grazing incidence monochromator is one way to achieve this goal, and so one is installed in our system. The focus of this project was to automate the positioning of the monochromator and thus automate wavelength selection.

Over the course of this project, a system was designed to attach a motor to the micrometer knob controlling the position of the monochromator. Once designed, the system was constructed and installed in the overall system. The system was then calibrated to correlate motor position to wavelength selected. The motor accurately positions the micrometer knob to within 0.001mm, which translates to within 0.008nm of the desired wavelength, and the support system meets all design goals.

# Introduction

Our team is currently working on a project to test various mirrors designed for use in the extreme ultra-violet (XUV) part of the electromagnetic spectrum. We have an XUV hollow cathode source that produces photons of various wavelengths in the XUV, and we would like to be able to select which wavelengths to admit into the test chamber. To do this, we are using a grazing-incidence monochromator.[1]

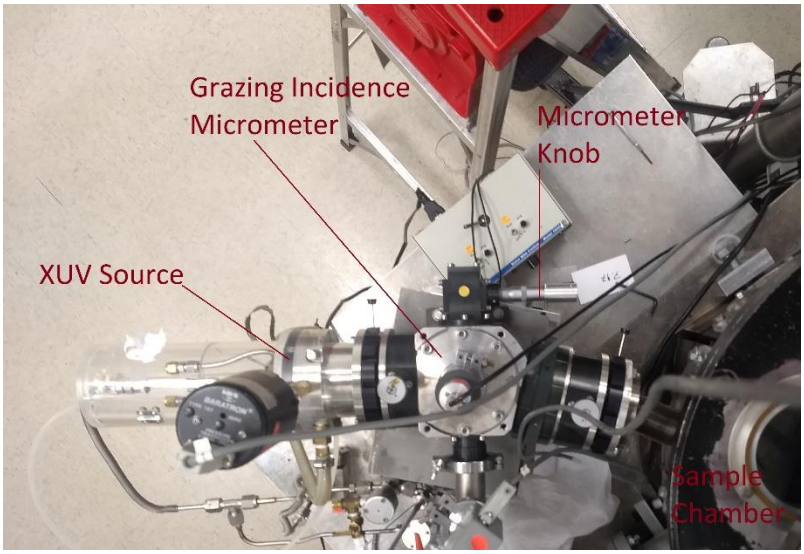



*Figure 1 Pictured here is the XUV light source, the grazing-incidence monochromator, and the micrometer knob this project is designed to turn.*

At the start of my project, the only way to change the wavelength was via a manual knob on the side of the monochromator. This process was slow and didn't allow automatic computer acquisition of intensity as a function of wavelength. Our goal in this project was to attach a high-resolution stepper motor to this knob in order to obtain the automation we desired.

There are many reasons why being able to scan wavelengths is a useful addition to our system. The first is that we would like to test different plasmas in our system. Currently, we are using helium gas to produce the plasma for our source. As we know the relative intensities of the lines of helium, we can easily identify which wavelength we have selected by looking at the counts on our detector and comparing the intensity to the lines of helium. As we do not know the relative

[1] See Mitchell and Turley for construction of the monochromator.

intensities of the lines of other gases, it would be much harder to determine the exact wavelength selected. Calibrating our system to calculate wavelength based on micrometer position will allow us to utilize any line we might come across.

The second reason we'd like to scan wavelengths is related to the first, and that is that other gases like hydrogen and argon produce a lot of background in the spectrum we're studying. Future work in the XUV group will focus on the background as a function of wavelength.

# Methods

## Preexisting parts

At the start of this project, there were a few parts of the system that had been selected previously, namely the stepper motor and the motor controller. This specific stepper motor was selected due to its high torque and resolution (Phidget 3328_0 - 42STH38 NEMA-17 Bipolar Stepper with 51:1 Gearbox), as this project has been attempted before but failed due to a low torque stepper motor. The motor controller (Phidget 4A Stepper Phidget STC1003_0) was also selected as a result of previous experience; this controller could be connected to a Phidget hub (Phidget Wireless VINT Hub HUB5000_0) that could be connected to the main computer wirelessly, sidestepping many issues the group has had in the past with USB motor controllers.

## Designing the system

When the design process was started, a few factors needed to be taken into consideration. First, when the micrometer knob is turned, the knob itself moves in and out, as shown in Figure 2. Any design would have to take this movement into consideration. Second, the finished product should have as little play as possible to improve wavelength accuracy. Finally, the finished product would need to be adjustable so the

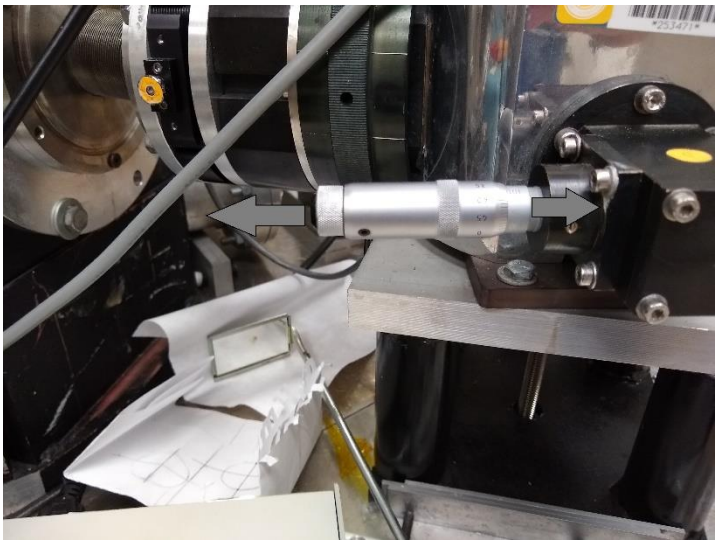


*Figure 2 As the knob is rotated upward, it extends out to the left according to the right-hand rule. The opposite occurs when the knob is rotated down.*

connection between the motor and knob can be as accurate as possible; any pressure on the knob could cause a leak, resulting in loss of vacuum.

Dr. Turley, Chante Clinchers and I considered many designs for the system. While initially disregarded, we decided to move forward with the idea of connecting the motor to the knob directly and supporting the motor on a rail. This design was selected after consultation with Jeremy Peterson, the head machine shop supervisor, since he suggested this design would be the easiest and most straightforward to construct.

The next question was where in the system the sliding would occur. We considered two options: sliding occurring between the base and the table or sliding between the base and the motor. As there would be less mass to move, causing less stress to the remainder of the vacuum system, we settled on the second option.

The next question was how to provide the translational motion between the base and the motor. Again, we debated between two options: rods or rails. Rods with bushings seemed to provide a solid option, as two parallel rods could provide stability and ease of movement while still being relatively inexpensive. Rails, however, had the added advantage of requiring only a single rail to prevent motor rotation relative to the base. This proved to be a significant advantage as it is difficult to machine parts to be exactly parallel.

A search on Amazon.com produced a number of 3D printer rails that would be effective for this project. The rail we decided on (Iverntech AW013) proved to be inexpensive and stable enough for our application. It was, however, two long. This was remedied by having one of the TAs in the machine shop cut the rail in half.

Another problem presented was how we would attach the stepper motor to the rail. Another search on Amazon.com produced a stepper motor bracket that would suit our needs (STEPPERONLINE Mounting Bracket for Nema 17 Stepper Motor). One issue with this bracket and all others like it is that the holes in the bracket would not match the holes in the rail carriage. This issue was remedied by drilling holes in the proper locations on the bracket.

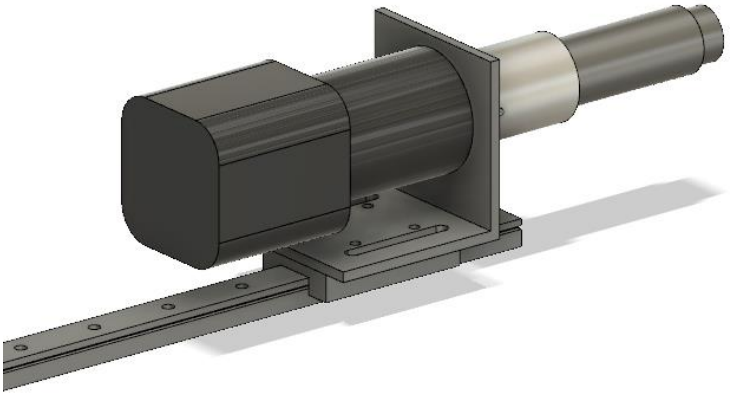


*Figure 3: The initial design of the attachment system. The micrometer knob (pictured in grey on the right) is attached to the stepper motor (pictured in black on the left) via the mating piece (in silver). The motor is attached to the rail via a bracket.*

I then decided to model the entire system so we could confirm which designs would work and which wouldn't. The initial design is included in Figure 3. I conducted another consultation with Jeremy Peterson to consider design options for the base. As a result of this meeting, I selected an A-frame as the basic design for the base as it would be quite stable while being easy to manufacture. A figure of the completed design can be found in Figure 4.

Work began on constructing the mating piece between the motor and the micrometer knob almost immediately. As I was not certified to work in the machine shop at first, I underwent training to receive that certification. Training complete, I machined the part from a piece of 1" scrap aluminum rod from the shop.

*Figure 4 The final design. The rail is bolted to the A-frame. The frame is supported by screws which allow for the height to be adjusted for perfect connection between the motor and the micrometer knob.*

During my meeting with Jeremy Peterson to discuss different base designs, we also discussed the material from which the base would be made. One suggestion he made was to make a 3D printed prototype of the base first, then use a CNC mill to cut the final design from aluminum. This would allow for any issues in the design to be realized before a more time-intensive version of the product was made. Following the arrival of the rail, I requested a 3D print of the base. The design of the base proved to be effective overall as it fit all spatial constraints of the system.

Upon assembly and instalment of the 3D printed base, it was discovered that the knob is not level and slopes downward toward the motor. Fortunately, the design was able to handle this unseen challenge. The screws in the back could be let out to lower the back end of the rail, causing the rail to perfectly match the slope of the micrometer knob. This is one reason why making a 3D printed prototype turned out to be wise. If the design was immediately machined from aluminum and then proved unable to account for this unforeseen constraint, much time would have been lost in constructing a new prototype.

Despite the overall success of the design, several improvements were made. One major change was the screws we would use for the aluminum base. The original design had most of the screw holes designed for metric screws. This became an issue since the machine shop of the physics

department on campus has a limited number of metric screws. The base was therefore redesigned to use imperial screws.

**Programming the motor**

Throughout the design and building process, Chante Clinchers and I worked on developing C# code to control the motor. This was done in an effort to write code that could then be incorporated into the main program controlling the entire system. We were able to develop a small Windows application that allowed us to choose a position for the motor to move to, as well as modify the maximum acceleration and velocity limit.

When the code in the main program was examined to check for compatibility with our code, it was determined that the code we wrote was largely incompatible with the existing code, on account of the existing code having a very specific class structure that made our custom code unnecessary. Acacia Ricks was given charge over adding the control of our motor to the overarching program.

**Calibration**

Calibration turned out to be difficult for a few reasons, most of which were beyond our control. For one, the photon detectors we were using in the system were not working properly. Once we were able to get a helium plasma going in the source, the detector picked up very few counts, despite proper alignment of the system. Once this problem was identified, new parts for the detector were ordered to replace the failing ones. A second issue identified after the new parts

were received and the detectors returned to working order is that we were unable to obtain a consistent plasma in the source. It would only stay lit for around a minute or two at a time, despite high currents trying to keep it alive. The main culprit for this is thought to be leaks in our vacuum system preventing us from keeping the pressures we needed. If given more time to solve this issue, this problem would also likely be resolved.

We were, however, able to confirm the location of a few lines previously identified. These line positions allowed us to create a fit to determine what position on the micrometer is desired for any given wavelength. We were specifically able to confirm the position of three lines, the 30.4nm line, the 25.63nm line, and the 24.3nm line. These lines were identified at micrometer readings of 4.33mm, 4.91mm, and 5.1mm, respectively.

# Results and Discussion

## Mechanical System

Here I've included pictures of the 3D printed and aluminum versions of the mechanism in place in the overarching system (figures 5 and 6, respectively). I ran a few tests with the motor on the 3D printed stand, and its accuracy is incredible. A complete 10 revolution slew was only 0.01 rev off, and this turned out to be because I didn't have the proper number of steps per revolution in my code. Calculating the proper number of steps per complete revolution combined with the new aluminum stand, which is even sturdier, should produce even greater accuracy.

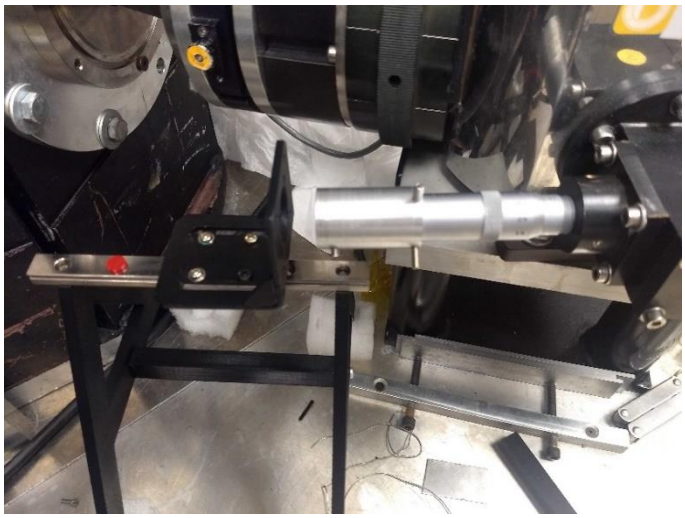


*Figure 5 A picture of the 3D printed base and rails (motor not included), along with the mating piece attached to the micrometer knob.*

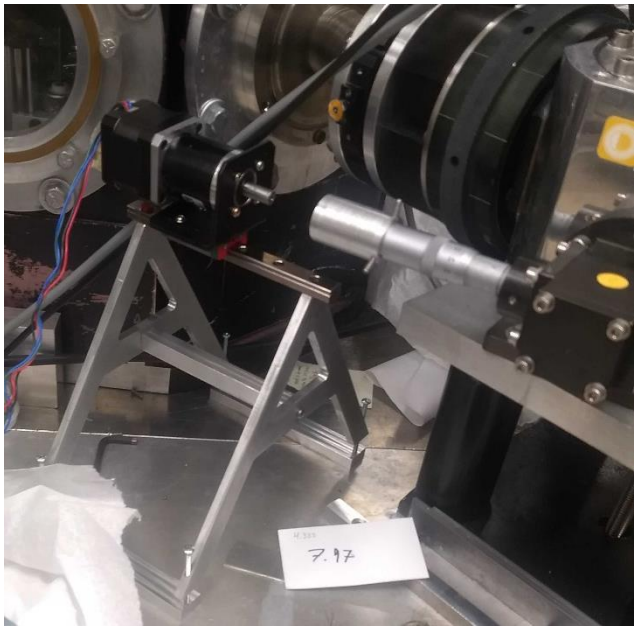


*Figure 6 A picture of the final aluminum product, with motor detached from the mating piece.*

One should note that despite the motor being slightly off after a 10-revolution slew, when the motor was instructed to return to its starting position, the difference in position between start and finish was indistinguishable to the human eye. I'd place an upper bound on the return accuracy to be 0.002 revolutions, which translates to 0.001mm on the micrometer. Given

the correct number of steps per revolution (which we have since calculated) and the sturdier frame, I feel that we can generalize this accuracy to the entire range of the micrometer.

**Programming**

As noted above, Chante Clinchers and I developed basic code to control the stepper motor position. This code has been included in Appendix A. The work of integrating the control of the stepper motor into the main control program has been handed off to our dedicated software engineer, Acacia Ricks.

**Calibration**

According to work done by Dr. Turley for fitting the wavelength dispersion equation,[2] there are two different fits to the wavelength equation, a linear and a non-linear fit. The nonlinear fit is based on solving the grating equation for the specific constants of our grazing incident monochromator. As the fit appears to be linear to the human eye, a second approach is to simply fit a line to the points we have confirmed, and this in fact produces a very close approximation.

My goal is to simply show that the error in the motor positioning is smaller than the potential error from the equation fits. From the source lines we confirmed from before, we conclude the linear equation has the form $\lambda = ax + b$ where $a = -7.726 \pm 0.048$ nm/mm and $b = 64.72 \pm 0.28$ nm. Plugging our upper limit in positioning error of 0.001mm into this equation gives a maximum of 0.008 nm error from motor positioning alone. This is much smaller than the error

[2] See Shevelko for work done on the dispersion equations.

from any other portion of the equation. It is therefore safe to assume that the error from motor positioning will not significantly contribute to error in wavelength selection.

# Conclusion

At the beginning of this project, we set out to achieve three goals. The first goal was to design a system whereby we could control the position of micrometer knob. The second goal was to develop software whereby we could control the motor. The final goal was to calibrate the system so that we could choose a wavelength and have the system take the micrometer to the necessary position.

We have demonstrated that the mechanisms to connect the motor to the knob has been designed, manufactured, and installed into the overarching system. We have also developed our own code for controlling the motor and have handed this responsibility off to our dedicated software engineer. Finally, we have made significant progress in calibrating the system to provide accurate wavelength selection.

# References

Sarah B. Mitchell, R. Steven Turley, “Installation of a Grazing Incidence Monochromator for use in the Extreme Ultraviolet,” Journal of the Utah Academy of Sciences, Arts, and Letters, 84, 154-160 (2008).

Alexander P. Shevelko, Larry V. Knight, R. Steven Turley, Oleg F. Yakushev, “Intense XUV Source of Radiation Within the 4-to 45-nm Spectral Range Based on Capillary Discharge Plasmas,” Proc. SPIE, Vol. 4504, p. 143-150 (2001).

The Phidget Library was also used extensively in the coding of this project:

https://www.phidgets.com/docs/Main_Page

# Acknowledgements

I would like to acknowledge my advisor and mentor for this project, Dr. R. Steven Turley, for his guidance and support of this project. I would also like to acknowledge my fellow students Chante Clinchers, Acacia Ricks, and Vinny Clements for their contributions and assistance in making this project a reality. I would also like to acknowledge Jeremy Peterson and the other TAs in the Physics Department Machine Shop for their assistance in the design and manufacturing of the system. Finally, I'd like to acknowledge the College of Physical and Mathematical Sciences for their financial support of this capstone project, without which none of this would be possible.

# Appendix

## Appendix A – Code for Motor

This code was designed to test the motor accuracy and hence accepts a position in revolutions. The code is separated into three parts, following the Model View Presenter pattern. PhidgetControlWindow is the View and handles all the UI elements, MicrometerMotorPresenter is the Presenter and handles updating the view and invoking methods on the Model, and StepperMotor is the Model, responsible for connecting to the stepper motor. Program.cs ties all the pieces together on program startup.

PhidgetControlWindow.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Phidget_Theory
{
    //The window is defined to be a view for the presenter. This allows the presenter to
    //call necessary functions on the window.
    public partial class PhidgetControlWindow : Form, MicrometerMotorPresenter.View
    {
        MicrometerMotorPresenter presenter;
        public PhidgetControlWindow()
        {
            InitializeComponent();
        }

        //The window defines a method to set the presenter.
        public void setPresenter(MicrometerMotorPresenter presenter)
        {
            this.presenter = presenter;
        }

        //A method to define how the window should handle a change in position.
        //This method outputs the new position to a text box.
        public void onChangePosition(double newPos)
        {
```

```
            Invoke(new Action(() =>
            {
                string strPos = newPos.ToString();

                actualPositionBox.Text = strPos;

            }));
        }

        //A method to define what happens when the Go To button is pressed.
        //It invokes a method on the presenter to change the motor position.
        private void goButton_Click(object sender, EventArgs e)
        {
            double targetAng = Convert.ToDouble(positionInput.Text);
            presenter.setTargetPosition(targetAng);
        }

        private void positionInput_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Enter)
            {
                goButton.PerformClick();
            }
        }

        //A method to set the current position as 0.
        private void zeroButton_Click(object sender, EventArgs e)
        {
            presenter.zeroPosition();
            positionInput.Text = "0";
            actualPositionBox.Text = "0";
        }
    }
}
```

MicrometerMotorPresenter.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Phidget_Theory
{
    // Define the presenter to be a listener for the stepper motor. Allows
    // the motor class to alert the presenter of updates in position.
    public class MicrometerMotorPresenter : StepperMotor.StepperListener
    {
        // An estimate of the number of steps per revolution
        // of the stepper motor.
        public static double STEPS_PER_REVOLUTION = 162874;
        private StepperMotor motor;
        private View view;
```

```
        public MicrometerMotorPresenter(StepperMotor motor, View view)
        {
            this.motor = motor;
            motor.setListener(this);
            this.view = view;
        }

        // An interface describing methods any view needs to implement
        public interface View
        {
            void onChangePosition(double position);
        }

        // A method to convert revolutions to stepper motor position.
        public void setTargetPosition(double rev)
        {
            //convert revolution to stepper motor position via formula
            int stepNum = (int)(rev * STEPS_PER_REVOLUTION);
            motor.setTargetPosition(stepNum);
        }

        // A method invoked when the stepper motor changes position.
        // Converts the motor position in steps to revolutions before
        // informing the view the position has changed.
        public void onPositionChanged(int step)
        {
            double rev = step / (double)STEPS_PER_REVOLUTION;
            view.onChangePosition(rev);
        }

        //A method to zero the position of the stepper motor.
        public void zeroPosition()
        {
            motor.zeroPosition();
        }
    }
}
```

StepperMotor.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Phidget22;

namespace Phidget_Theory
{
    public class StepperMotor
    {
        // This constant defines how many steps to overshoot a
        // target position when decreasing the target position.
```

```
        private static int OVERSHOOT_STEPS = 10000;
        Stepper stepper;
        StepperListener presenter;
        bool awaitingReversal = false;

        public StepperMotor()
        {
            stepper = new Stepper();

            // This is all the details of our stepper motor
            // so the library can connect us to it.
            stepper.HubPort = 0;
            stepper.DeviceSerialNumber = 596775;
            stepper.IsRemote = true;

            // Try to connect to the stepper motor, time out
            // after 20000 ms
            stepper.Open(20000);

            // Set event handlers for position changing and
            // the stepper motor stopping.
            stepper.PositionChange += Ch_PositionChange;
            stepper.Stopped += Ch_Stopped;

            // Set the velocity limit. This number is 3x greater
            // than the default.
            stepper.VelocityLimit = 30000;

            // Send current to the motor.
            stepper.Engaged = true;
        }

        // Event handler for the position changing. Alert the presenter
        // that the position changed.
        private void Ch_PositionChange(object sender,
Phidget22.Events.StepperPositionChangeEventArgs e)
        {
            presenter.onPositionChanged((int)e.Position);
        }

        // Define an interface for any listener, the presenter in this case
        public interface StepperListener
        {
            void onPositionChanged(int position);
        }

        // A method for setting the listener
        public void setListener(StepperListener listener)
        {
            this.presenter = listener;
        }

        // A method to change the target position. If the motor
        // must turn backward to reach the new target, we want
        // the motor to overshoot the target, then come back up.
        // This ensures we always approach a position from the
        // same direction, limiting the effects of play on position.
        public void setTargetPosition(int stepNum)
```

```
        {
            // If the new position is less than the current position,
            // overshoot the target and set a flag for the code to
            // come back to the original target when the motor stops.
            if (stepNum < stepper.Position)
            {
                stepNum -= OVERSHOOT_STEPS;
                awaitingReversal = true;
                stepper.TargetPosition = stepNum;
            }
            else
            {
                awaitingReversal = false;
                stepper.TargetPosition = stepNum;
            }

        }

        //Event handler for when the motor stops.
        private void Ch_Stopped(object sender, Phidget22.Events.StepperStoppedEventArgs
e)
        {
            //If the target was overshot, come back.
            if (awaitingReversal)
            {
                stepper.TargetPosition = OVERSHOOT_STEPS + stepper.Position;

                awaitingReversal = false;
            }
        }

        //Method to zero the position of the stepper motor.
        public void zeroPosition()
        {
            double position = stepper.Position;
            stepper.AddPositionOffset(-position);
        }
    }
}
```

Program.cs

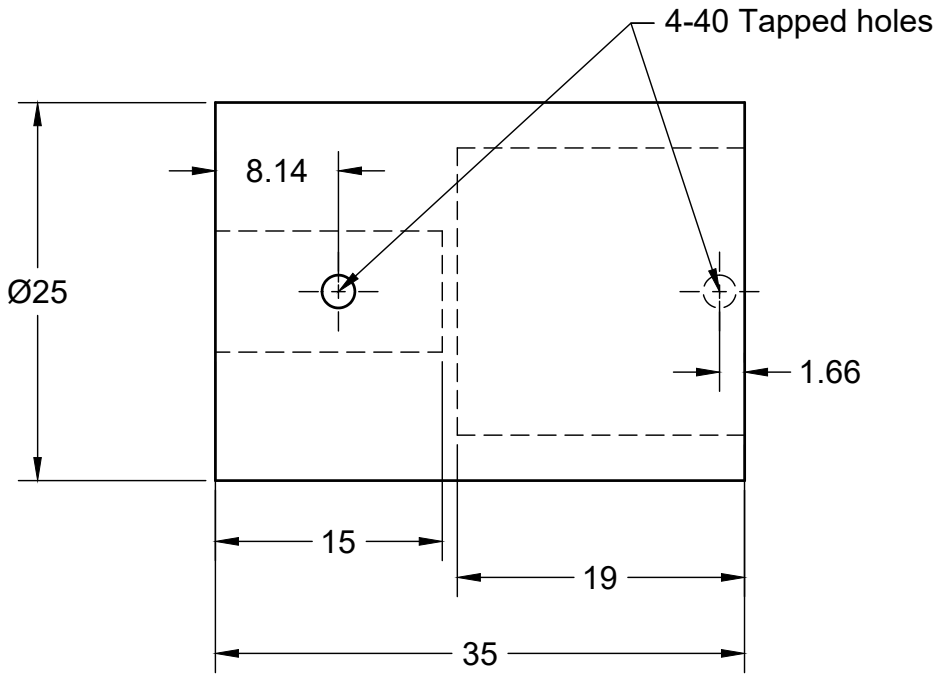
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;
using Phidget22;

namespace Phidget_Theory
{
    static class Program
    {
        /// <summary>
```

```
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            // Setup the Phidget library.
            Net.AddServer("serverName", "192.168.100.1", 5661, "", 0);

            // Setup the windows
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);

            // Create the view, model and presenter and link them all up.
            PhidgetControlWindow window = new PhidgetControlWindow();
            StepperMotor motor = new StepperMotor();
            MicrometerMotorPresenter presenter = new MicrometerMotorPresenter(motor,
window);
            window.setPresenter(presenter);

            // Run the program.
            Application.Run(window);
        }
    }
}
```

## Appendix B – Part Drawings

Drawings for each of the parts of the base are included in the following pages.

Side View



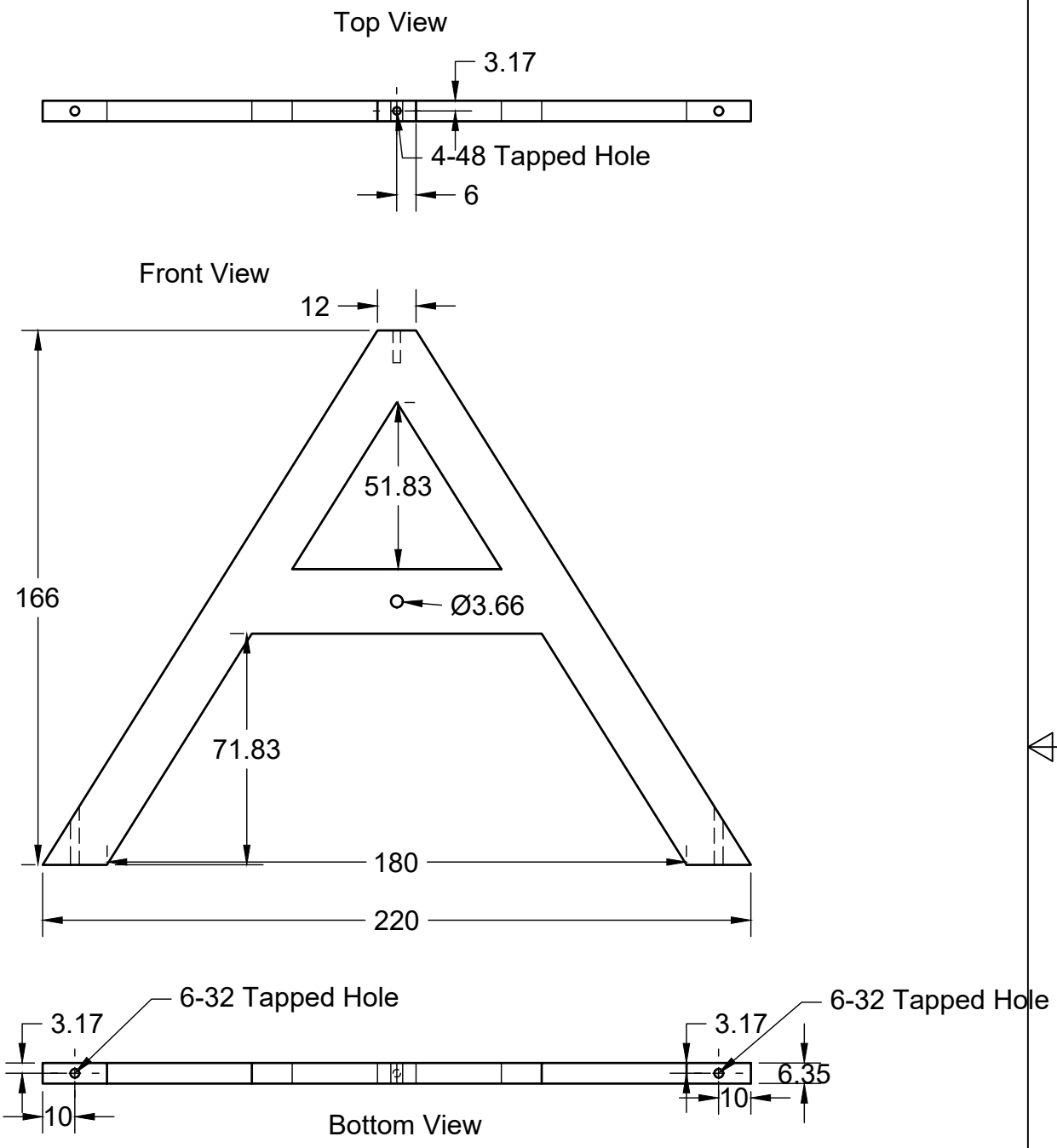

PROJECT

# Capstone Project

TITLE

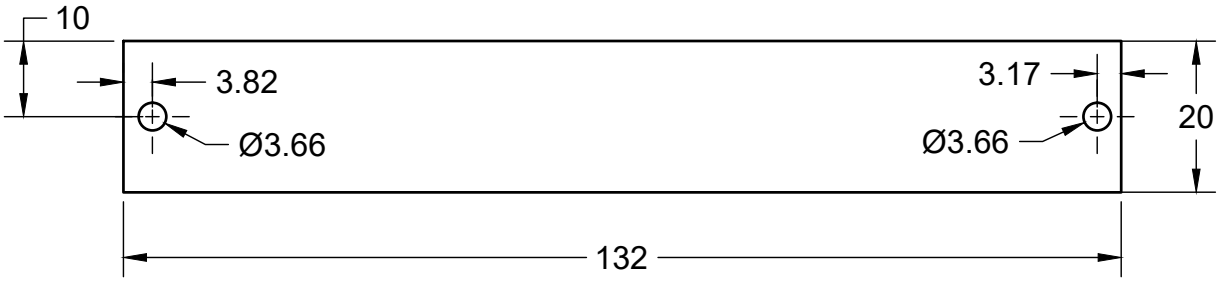
# Mating Piece

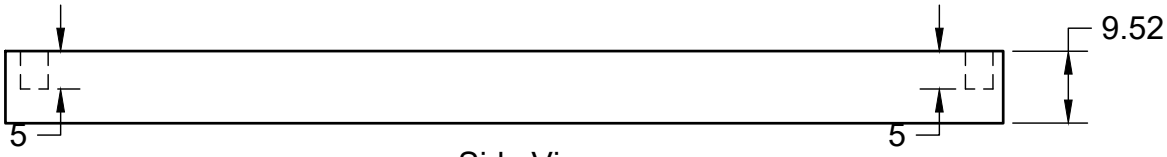
| | | | | |
|---|---|---|---|---|
| APPROVED | SIZE | CODE | DWG NO | REV |
| CHECKED | A | | | 1.0 |
| DRAWN James Adams 8/13/2020 | SCALE 2:1 | WEIGHT | | SHEET 1/1 |




| | | | | | |
|---|---|---|---|---|---|
| | PROJECT<br>Capstone Project | | | | |
| | TITLE<br>A-Frame Support | | | | |
| APPROVED | SIZE | CODE | DWG NO | | REV |
| CHECKED | A | | | | 1.0 |
| DRAWN James Adams 8/13/2020 | SCALE 1:2 | WEIGHT | | SHEET 1/1 | |

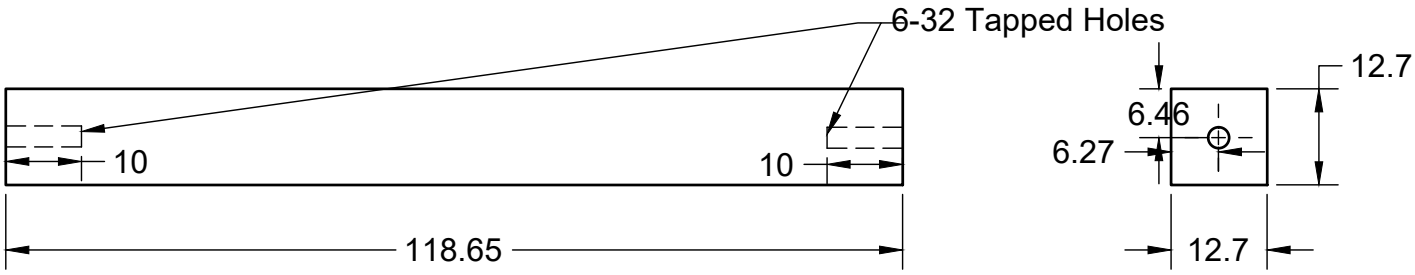
Top View




Side View

| | | | | | |
|---|---|---|---|---|---|
| | PROJECT<br>Capstone Project | | | | |
| | TITLE<br>Base Plate | | | | |
| APPROVED | SIZE<br>A | CODE | DWG NO | | REV<br>1.0 |
| CHECKED | | | | | |
| DRAWN James Adams 8/13/2020 | SCALE 1:1 | | WEIGHT | | SHEET 1/1 |




PROJECT

# Capstone Project

TITLE

# Connector Piece

| | | | | |
|---|---|---|---|---|
| APPROVED | SIZE | CODE | DWG NO | REV |
| CHECKED | A | | | 1.0 |
| DRAWN James Adams 8/13/2020 | SCALE 1:2 | WEIGHT | | SHEET 1/1 |