Solar Tracking System for Photovoltaic Arrays

by ECE 4416 Group #11 -

Noah Janzen

Oscar Sanchez

Edwin Tojo

Greg Solis-Reyes

(*Faculty Advisor*: Dr. Firouz Badrkhani Ajaei)

ECE4416 Electrical/Computer Engineering Project

Final Project Report

Department of Electrical and Computer Engineering

Western University

London, Ontario, Canada

Friday, April 3, 2020

Submitted in partial fulfillment

of the requirements for the

degree of Bachelor of Engineering Science

Abstract

The purpose of our project was to design, build, and test a solar tracking system for photovoltaic arrays. We built a scaled-down prototype system which tracks a light source on a single axis of rotation. The system consists of an input sensor system, a microcontroller, and an output actuator system. The input sensor system finds the angle of the light source relative to the solar array. The output actuator system moves a large designated power-producing solar panel to the optimal angle for maximum power production, to match the angle found by the input sensor system. The microcontroller serves as the software and control hub with multiple hardware connections to the input and output subsystems. Our prototype system managed to achieve a power production increase over conventional fixed solar array systems of approximately 26%.

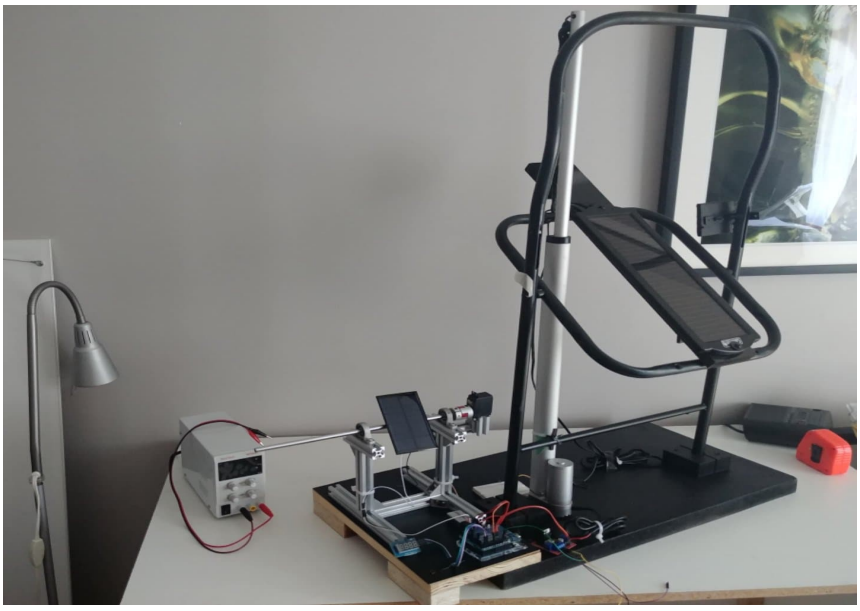


Figure A-1: Final Prototype System

Important Note

We would like to bring to your attention several working demonstration videos of our prototype system. The videos can be found on our team YouTube page at the link below.

There are two playlists: one with videos for the final prototype, and one with videos of working (non-final) prototypes. We would ask that these videos be watched to gain a full understanding of our system. Further description of our videos can be found in the report below, and within the videos themselves.

https://www.youtube.com/channel/UC9q9LPnj8SVVNERrfFckrug/playlists

Contribution of the Team Members

The team member contribution has changed slightly ever since the midterm report submission. The components of the project have been split in such a way that each connects with one another and to ensure a collaborative effort. Over the course of the entire year, the responsibilities of each report and components of the project have been divided between the team members to equally contribute to the final design.

Noah Janzen:

- Led the design of the sensor circuits and amplifiers to collect input data and relay it to the microcontroller.
- Ordered parts for the output solar farms such as wood, fixtures, pillow bearings, aluminum shaft, etc.
- Helped in build the 1st generation of the prototype
- Designed the project's Gantt chart, ensured team stayed on track of deadlines and submissions

Oscar Sanchez:

- Led the design of testing parameters and testing system implementation
- Managed the prototype lab tests and iterative process (power consumption, efficiencies, etc.)
- Assembled electrical for both the 1st and 2nd generation of the prototype

Greg Solis-Reyes:

- Led in the programming and integration of all the hardware in the design project
- Ordered parts for the 1st generation input sensor system (such as the Arduino kit, wiring, stepper motors, etc.) and built the framing for it
- Ordered parts for the 2nd generation output solar system and built the framing for it

- Maintained communication between the team and our advisor and instructor with regular emails

Edwin Tojo:

- CAD drawings for some mechanical parts in the design project such as the shaft coupling and the shaft mount for the solar panels
- Helped build the 1st generation prototype's output system
- Ordered parts for the 2nd generation prototype input system and helped build the framing/fixtures

Acknowledgments

Our Team would like to graciously acknowledge and thank all persons and groups who have directly and indirectly contributed to the successful completion of our project.

Firstly, we would like to thank our Faculty Supervisor Dr. Firouz Ajaei for taking our team on and providing us with guidance throughout the development of our project. Dr. Ajaei took time out of his already busy schedule to give suggestions, help resolve issues, grade deliverable submissions, and provide high-level leadership which allowed our team to track towards success. Dr. Ajaei also graciously donated several hardware components which were used in the construction of our final prototype, without which the completion of our project would not have been possible.

We would also like to thank Dr. Ajaei's TA's and research group members for always giving us access to their lab facility to work on our project. We would especially like to thank Kouroush Saffar who was always at the lab early on Friday mornings to open the door for us. Kouroush also helped us navigate the available equipment in the lab and answered questions for us throughout our working sessions.

Next we would like to thank the Western Electronics Shop staff for their support and insight. We acquired several key hardware components from the shop, including a servo motor which was graciously donated to our project.

We also thank Dr. Rao, the Capstone course coordinator, and his TA for their efforts throughout the term. Dr. Rao did an excellent job providing instruction and making expectations clear to all students throughout the term. Dr. Rao's efforts are especially commendable given that this is his first time coordinating the Capstone course.

Finally, all group members would like to thank our families and roommates for their support and tolerance of our efforts throughout the term. At times there were tight and late deadlines to be met, and construction to be done, but those we live with allowed us the space and support we needed to be successful.

Table of Contents

List of Figures & Tables

Introduction

Problem Statement

The goal of our project is to produce a system which will track the sun throughout the day to increase the power output of solar panels in solar arrays ("farms") that are typically mounted in a fixed position. Our control system should adjust the solar panel's angle to maximize the sunlight incident on the panel and thus, increase the quantity of power generated. Our system should improve existing efficiencies of solar panels as a power-generation method and therefore, increase the viability of solar panels as a power source on a large scale.

Detailed Literature Review

To begin the preliminary design and investigation for the construction of our system, it was necessary to gather background information and investigate current methods used in the solar power-production industry and mechanical/electrical control systems industry. To do this, we examined several articles from various sources. The literature review detailed below is essentially the same as that detailed in our prior "Midterm Progress Report", as the review was used as a starting point for our project. Further investigation related to subsequent design and construction phases of the project were carried out without a deep, structured review of technical literature.

Examination of Solar Industry and Project Motivation

The first article reviewed was '*World estimates of PV optimal tilt angles and ratios of sunlight incident upon tilted and tracked PV panels relative to horizontal panels*' by Mark Z. Jacobson and Vijaysinh Jadhav from the Department of Civil and Environmental Engineering at Stanford University [1]. This study looks at between one to four sites in various countries around the world to find the estimated gains by adding

various types of solar tracking. This also contains estimates of the optimal angle to mount the solar panel when in a fixed position. This study uses a combined weather model that uses 30 years of historical data to predict weather trends and another weather model that accounts for expected deviation from historical trends. This also explains the effects of diffusion via cloud cover and why various locations at the same latitude can have different optimal angles. This information is useful because it helps provide background as to why solar trackers are used and the expected gains based on location. Here, it shows where solar trackers are the most useful and where they are less useful. Furthermore, latitude is not the only factor that explains that the diffusion of cloud cover can alter the optimal angle. This source was used because it is informative in regards to the expected angles and also gives background to the viability of solar trackers. This source is weaker since it does not consider the energy used by the solar tracker.

Review of Microcontrollers

The next item we reviewed was the article "*A review of Arduino boards, Lilypads, & Arduino shields*" by Anad Nayyar and Vikram Puri, published in the 2016 International Conference on Computing for Sustainable Global Development [2]. In the article, the authors outline various advantages of using the Arduino platform over other microprocessor units in projects requiring a microprocessor. Some of the advantages include: the open source nature of the underlying microcontroller, the size and activity of the online community of Arduino users, and the variety of Arduino products available for use. This article provided valuable information when making selections as to which controller package to use for the system's processing and control needs. The article went on to give a general overview of the functions and properties of a typical Arduino module. Some of these functions include: the use of C to program the Arduino device, the availability of an Arduino-created IDE for programming, and various input and output capabilities for each Arduino board. The information outlined in the article provided a general idea of the functionality and capabilities of an Arduino

microcontroller. This information was applied to decide if an Arduino controller would be appropriate for this design project. The article then described some general aspects of the hardware for different Arduino boards, Lilypads, and shields. From this section of the article, the various different baseboards and add-ons Arduino offered was learned. In addition, the article provided information about the different target uses for each of the various boards. This section of the article helped to decide which, if any, of the Arduino microcontroller boards could be a good fit for this project.

System Testing Investigation

To further enrich our detailed literature review and also guarantee the success of this project, we have investigated methods and proper ways that can be used to test the system as a whole. "*Developing a test platform for PV Panels*" by Tilda Akiki, Laura Eid, Serena Abboud, Bechara Nehme is an article published at the 2017 International Conference on Microelectronics [3]. This article focuses on the development and testing of an indoor test platform for PV panels and the implementation of a reflector on which hang LED strips that will provide a light, comparable to the sun rays that will then be captured by a PV panel to provide electricity. Furthermore, the authors discuss their approach to the design of the various components of the project by using different software and hardware. Softwares such as Comsol Multiphysics, used in the design of the reflector mentioned above, and Matlab in the development of a Graphical User interface where the user can save multiple data obtained during PV panel testing were of extreme importance. Arduino board is the interface between the hardware and the code developed in MATLAB. The Arduino board is responsible for converting analog signals to digital signals. It stands as the ADC between the panel and the computer. Lastly, it was time to choose the lighting fixtures that would be emulating the solar energy source. This information is important because to implement it in our project, we would only need to know the required irradiance, and also the area of the PV panel.

Project Objectives

The main objective and scope of this project is to successfully design, build, and test a complete scaled-down prototype system which will allow photovoltaic panel(s) to track a light source as the source moves (specifically sunlight as the sun moves throughout the day). The tracking will occur on a single 180º axis of motion. The prototype system should be able to be scaled-up (with appropriate modifications) to achieve control for commercial-sized solar panel farms. The scaled-up system should be able to be retrofitted onto typical existing solar panel farms. The methods for scaling-up the system are described herein our final report. The system should increase the overall output level of the solar panels by at least 7% over a year when at least 5 commercial-sized panels are controlled by the system. This means that the output energy of the panels minus the input energy required for the control system (energy for: sensor(s), processor, movement of panels) should exceed the original output energy from the panels (when static) by at least 7%, over a year. The 7% figure was determined by realistically rounding down estimates of 33% [4] and 38% [5] power increases from previous analyses of ideal dual/ single axis solar tracking systems (to account for losses, implementation inefficiencies).

Design Approach

Input Sensor Array Design

Concept Generation

The sensor array had two concepts generated as possibilities. The first was a series of attuned solar panels. This would see a number of solar panels mounted at different angles and measure which panel had the largest voltage output. For this method only 6 panels would have been used as that is the maximum number of analog inputs.

A sketch of this preliminary design can be seen below. The sketch was prepared using AutoCad, an Engineering Drafting software commonly used in many industries for the presentation of Engineering Designs.

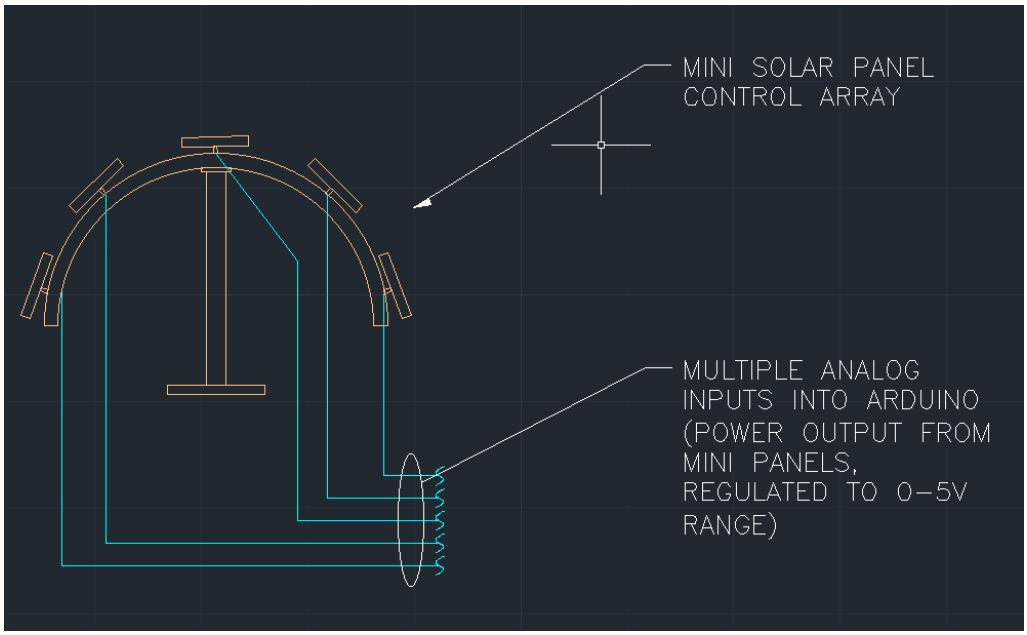


Figure CG-1: Preliminary Input Sensor Design Concept

The second method was a single panel attached to a stepper motor which samples light output at a range of angles, with the maximum output selected as corresponding to the desired angle. This method would be more accurate but also consume more power for the motor and moving parts.

Concept Design and Evaluation

The first concept was designed to have panels spaced 5 degrees apart with the middle being the estimated optimal angle. This was ultimately discarded due to the high degree of error as the steps between measurements are extremely high. This method would likely work better on a large scale as a controller could be designed with as many inputs as needed and the elimination of moving parts would reduce power consumption and the sample rate would no longer be an issue.

The second concept was ultimately decided on as the stepper motor can be geared for as much accuracy as required; it needs minimal analog inputs. The weaknesses of this model is that the stepper motor consumes electricity. The other problem with this method is that if the solar panel fails the circuit becomes completely dependent on the backup. The backup system is a photoresistor that is not as accurate as a solar panel because the photoresistor reacts to different waveforms of light than the panel.

In our final system prototype we replaced the stepper motor with a DC servo motor (model Futaba s-148, see Figure PD-1 in Product Data Sheets below). This was done since the servo motor was able to move faster when running through the sampling angle range, and provided a more accurate control of its angular position. The stepper motor had to be geared, while the servo motor did not, simplifying the design and construction of our prototype system.

Control Unit Design

Concept Generation

The first step in the design of our control unit was selecting the control unit we would use for our project. We knew that we would need a control unit to input feedback signals from sensors, and output control signals to actuators. We knew that there would be various sensors and actuators to be controlled in our system.

From these principles, we decided that we would need to use some sort of programmable controller with hardware input and output capabilities. We decided that we would use either a PLC (programmable logic controller), or a microcontroller unit for our control unit. We considered, but ruled out using an FPGA (field programmable logic gate array) as an FPGA would be too difficult to program and not flexible enough for our project.

Concept Design and Evaluation

Once we decided that our control unit would be either a PLC or a microcontroller, we needed to decide between one of the two.

Microcontrollers had the advantage that they were easy to use, were cheap, often came complete with electronic accessories (i.e, small-project ready).

PLC's had the advantage that they were more robust (often used in industrial applications), and were easy to program (with ladder-logic diagrams).

Ultimately, the deciding factor was the fact that our group members had experience through previous coursework working with microcontrollers, whereas nobody had experience working with a PLC. Cost was also an important factor, as we knew we would be able to get a good microcontroller unit for under $100 while PLC's could cost several hundred dollars.

The final aspect of our selection of a control unit was to decide on the specific microcontroller we would use. We decided between using an 8051 controller (made by Intel), an Arduino Nano unit, or an Arduino Uno unit.

We ultimately decided to use the Arduino Uno microcontroller for our project. The Arduino Uno has a low overall cost, an excellent online support community (more so than the Arduino Nano), many input and output pins available (enough for our project's needs), and comes complete with a hardware interface board that can be readily connected to electronics for prototyping [6]. The 8051 unit has the big disadvantage that it does not come complete with a hardware interface board [7]. This means that if we selected the 8051, we would have had to purchase hardware

interface components separately and assemble a test board ourselves. We decided that this would be an unnecessary time expenditure as the Arduino Uno was ready to use out-of-the-box.

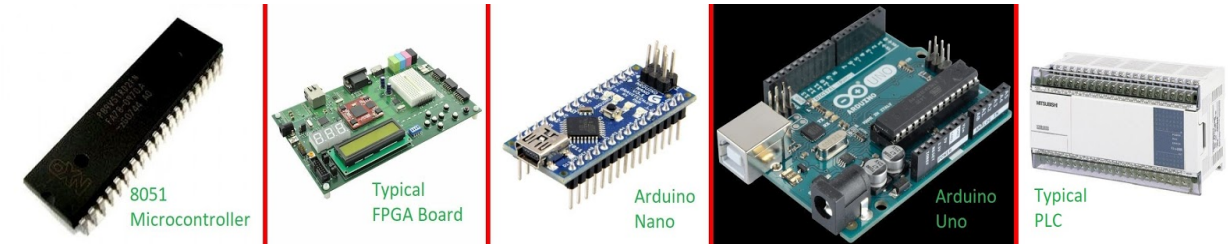


Figure CG-2: Considered controller options

Output Actuator and Array Design

Concept Generation

There were several concepts and ideas generated involving the motion of the solar tracker. The solar tracker can be designed to move a solar panel in two ways; single axis and dual axis. The single axis solar tracker would simply rotate the beam on which the solar panel is to sit using a mechanism. This strategy uses less electronics and would cost less than a dual axis solar tracker. After conducting some research, the efficiency/accuracy was put into perspective and it was concluded that the dual axis tracker seemed to provide more by providing motion in two axes. Moving on, the type of motor and then electronics used to control this motor was evaluated. To comply with the Arduino Uno, the motor should be able to rotate the solar panel at various angles to test the panel's ultimate position at which it generates the most electricity. We discussed that the motor will be responsible for rotating at these angles (during the measuring phase) and then repositioning itself to the identified angle. Some of the specifications that were considered when generating these design concepts included the rated voltage at which the motor requires to operate, its torque, ease to program/integrate and its expense.

A sketch of this preliminary design possibility is shown below:

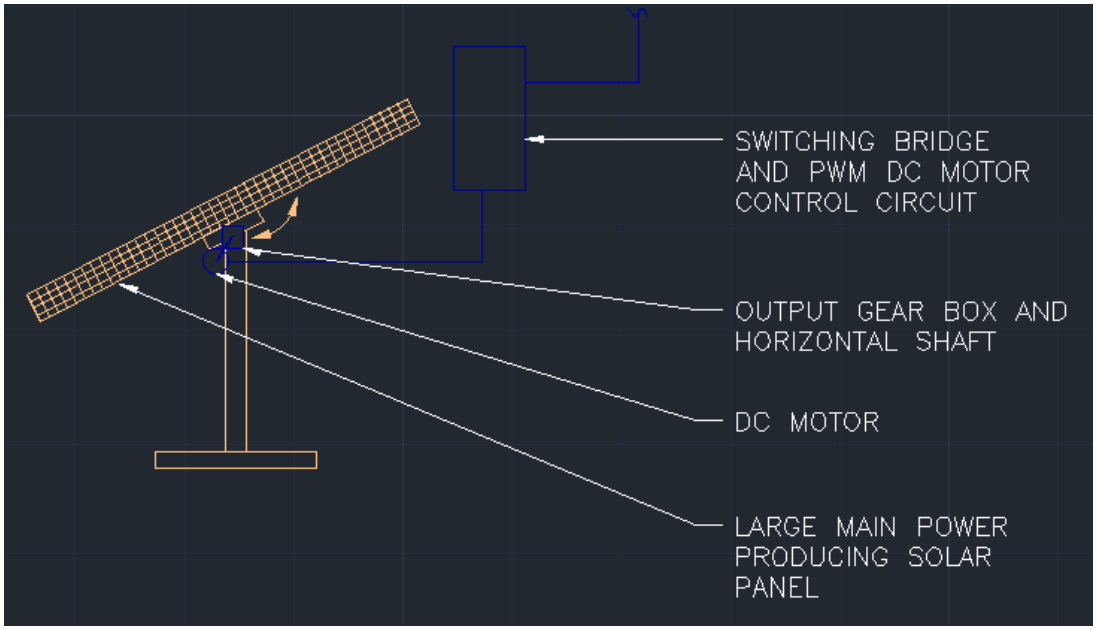


Figure CG-3: Preliminary Output Actuator Design Concept

Concept Evaluation and Selection

For this project, it was decided to proceed with a single axis tracker as it requires less components, less power consumption and, ultimately, it will lower the cost of the project. In our project proposal, we proposed the idea of using a DC geared motor in conjunction with a motor controller and potentiometer. However, we explored other alternatives and investigated stepper motors and servo motors. Servo motors are unique in that it's motor shaft can be moved to a precise angle. Although most servos can only be rotated 180 degrees, there are some modified servos that can provide the full 360 rotation. The servo motor is aware of it's position unlike the stepper motor and can be moved to a specific angle even if it is interfered with by any external force. The stepper motor can be misaligned if their shaft is moved by an external force.

Despite all these benefits, we were unable to find a servo motor that would have an adequate torque to support and turn a large or even medium-sized solar panel. Due to this limitation, we were forced to revise our design to implement an output actuator

system that would use a DC linear actuator to turn a panel mounted on a separate frame. A sketch of this final design concept can be seen below:

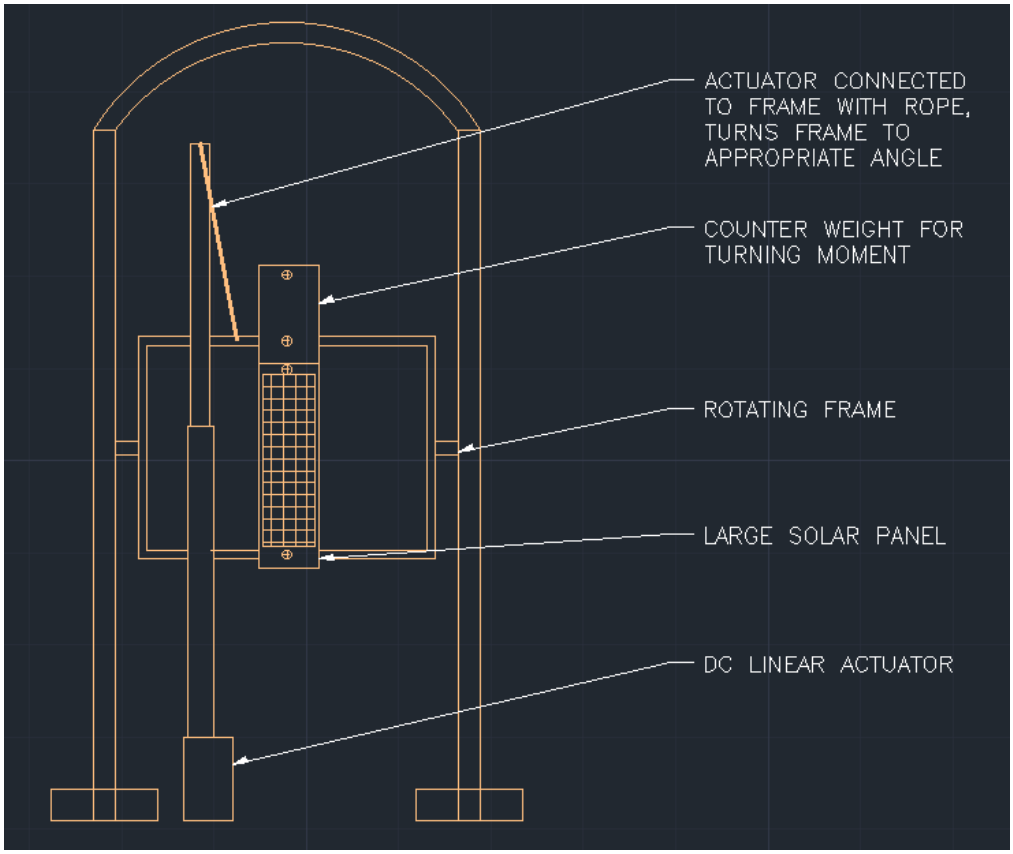


Figure CG-4: Final Output Actuator Design Concept

Testing system design

Concept Generation

The goal of the system will be to track the movement of the sun throughout the day to maximize light flux incident on the panel surface thereby also maximizing the panel's power output. We will test and analyze results from the system to check for its overall efficiency and see if it can be increased, and by how much.

Concept design and evaluation

We will select an appropriate microcontroller unit (Arduino Uno) to accept and process the input from the sensor. The microcontroller will then send a control signal to

the actuators mounted to the system to adjust the arrays' angle towards the sun for maximum power output. We may be able to use a simulator (such as MATLAB) to simulate the controller output and actuator response before interfacing physically with the real actuator. MATLAB software will also be selected to create a graphical user interface where the user can save multiple data obtained during PV panel testing. Voltage, current and temperature of the PV panel as well as the ambient temperature are read by the Arduino board and sent to an Excel file where the user can generate different plots. Additionally, we have found out that working on MATLAB with Arduino requires the MATLAB Support Package for Arduino to ensure the communication between the program and the controller.

Although we had originally planned to implement the automatic testing procedure described above, it resulted to be infeasible for a variety of factors including time, cost, and incompatibility of voltages for measuring equipment. For this reason, we revised our testing procedure to do all final testing manually, as can be seen in our Final Results and Validation section below.

Design Analysis

Engineering Techniques/Software tools

A variety of engineering tools and techniques were used throughout the project for the successful development, analysis, and testing of our prototype system.

The first, but still very important tool used in all our design procedures, were hand-drawn schematics and sketches. Through these sketches and hand-drawn schematics were were able to layout basic designs which could then be transcribed to more in-depth engineering software and drafting software. For example, for the construction of our aluminum frame input prototype, a hand-drawn dimensioned isometric sketch was used to plan out the necessary cuts, lengths, and tools needed to build the frame. See the sketch below.

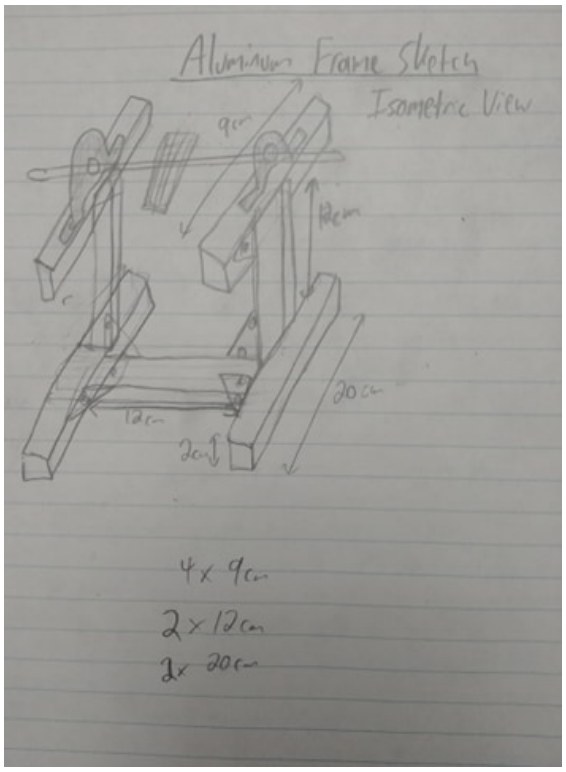


Figure ET-1: Aluminum Frame Isometric View Sketch

After sketches/designs were produced by hand, we used engineering drafting software and tools to make our designs clear, legible, and professional. One of the main software tools we used to achieve this was AutoCad, a common industry tool used for engineering drafting. The AutoCad drawings were then either plotted or directly copied into appropriate areas of our various reports to clearly illustrate our system. A screenshot of an AutoCad design used in our project can be seen below:

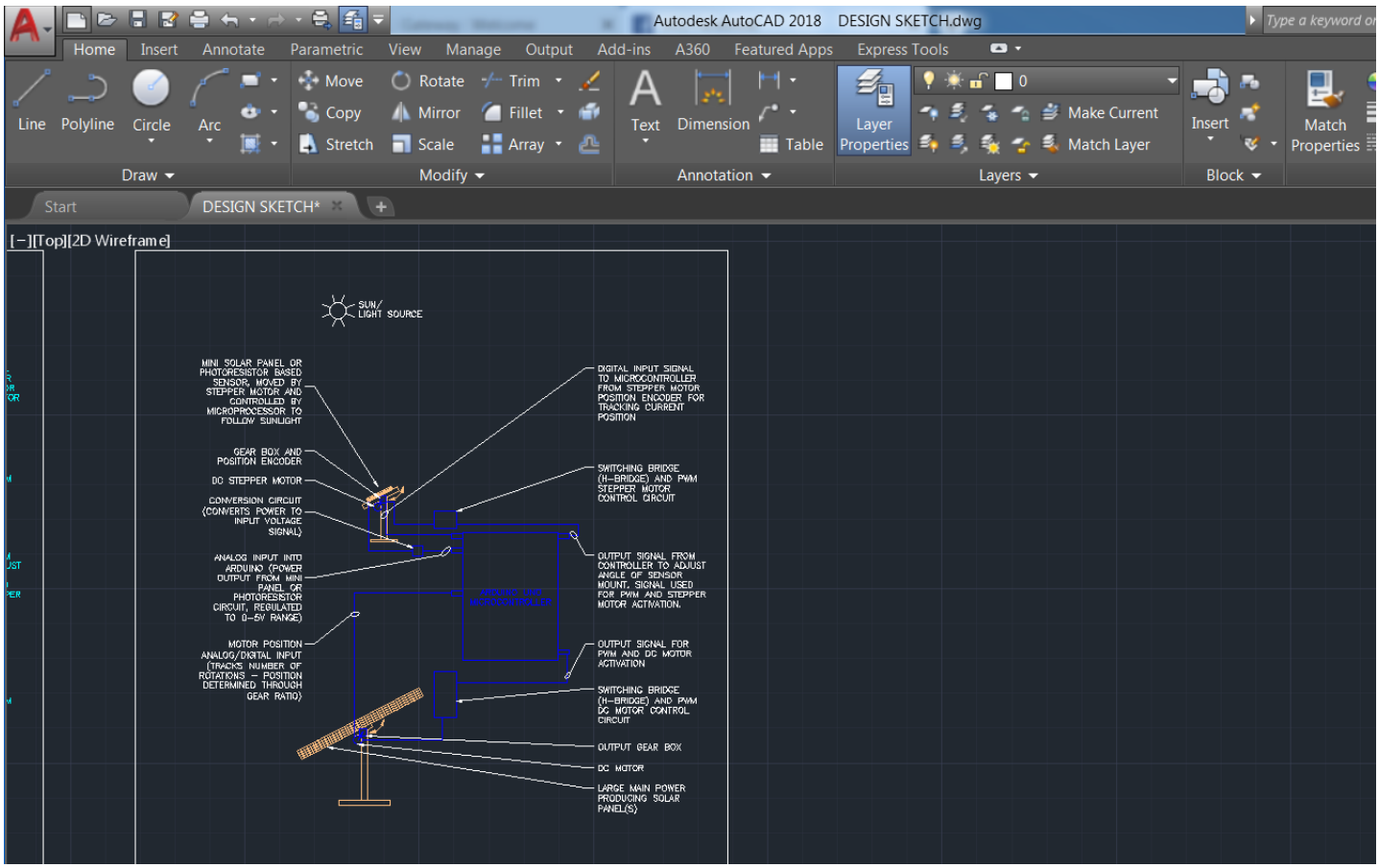


Figure ET-2: AutoCad Project Design File

We also used common electrical engineering tools to simulate some of our preliminary design circuits, including the SPICE circuit simulator Micro-Cap. Micro-Cap was used to simulate our simple input circuitry which was used in our final design to

calibrate proper amperage for our inputs into our Arduino Uno unit. An example of this can be seen below.

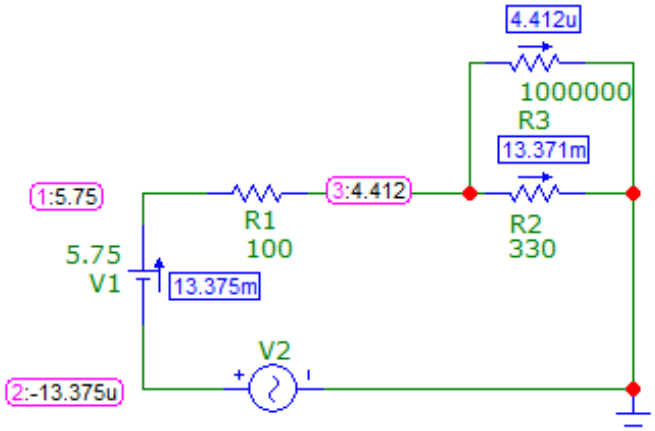


Figure ET-3: Micro-Cap Light Detection Circuit Simulation

Another tool we used to create simulations and plots for analysis throughout the design of our project was the simulation software MATLAB. Several MATLAB scripts and plots can be seen in the Calculations sections below, as they were used in the design and analysis related to our output actuator control system. MATLAB was especially helpful as it allowed us to see real simulations of our manual calculations, and helped to serve as a first line of validation for our various designs.

Finally, the software we used the most, and the most important piece of Engineering software for this project was the Arduino Uno IDE (Integrated [software] Development Environment). The IDE is a programming environment for the creation of control code to be uploaded to Arduino hardware devices.

Through the use of this software, we were able to apply standard software and computer engineering practices to the design of our control code. The code used in the Arduino Uno IDE is a close approximation of C code, which is itself a close approximation of C++ code, the standard programming language taught to

undergraduate engineering students. All of the software control code used in the various iterations of our control system is also copied to the Detailed Calculation section of our Appendix for detailed reference. Some details of the Arduino Uno IDE environment can be seen in the screen capture below. The screen capture below shows some of our annotated preliminary code, used for an early iteration of our prototype, and is only shown for the purpose of illustrating the Arduino IDE.

Figure ET-4: The Arduino IDE Software

Complete Analysis/Calculations

The main component of the project design was the creation of the control system, including hardware and software control mechanisms. In order to create the control software, it was necessary to integrate knowledge of electrical engineering principles related to electrical hardware, and principles related to information such as control theory and algorithm design.

The first iteration of the control software included configuring the control of the motor to be used for the input actuator system. This code included a trial for position and speed control of a stepper motor (see Figure PD-2) for stepper motor specification. The control code for this development step can be found in the Appendix Figure DC-1.

After we learned how to control the position and speed of the stepper motor, we created a control algorithm to control the position of the stepper motor (and hence the sensor) based on the number of steps taken by the motor. The stepper motor is intrinsically geared, and it takes a certain number of “steps” to complete a revolution. Using this information from the above mentioned specification sheet, we created an algorithm to scan and determine the maximum light angle. The code and more details within the code for this iteration of our design can be found in the Appendix Figure DC-2.

After designing the input control algorithm, we designed a microcontroller script to integrate the input and output systems together to work together automatically. This script sent the angle found by the input system to the output actuation system. The code for this iteration can be found in Appendix DC-3.

The next step in the development of our control software included starting to implement more sophisticated methods into our code. This included creating functions which would carry out the important tasks in our code. This is a standard procedure in

software engineering, and is seen as good system design for any project involving computer code.

We created two functions in this iteration, a scan function and a findMax function. The scan function contained the code needed for the input system to scan, and the findMax function contained our logical algorithm which found the maximum stored value. The code for this iteration can be found in Appendix DC-4.

The next iteration of our software control code involved changing the input scanner motor from the stepper motor described above, to the servo motor. This essentially involved rewriting the scan function to accommodate control of the servo motor. The code for this iteration can be found in Appendix DC-5.

After reprogramming our system for the servo motor as the input actuator, control code had to be created to control the position of our linear actuator. The inclusion and use of a linear actuator was a late addition to our project,and the control of the position of the output system using the linear actuator proved to be one of the most challenging parts of the project. There was a lot of analysis needed to create a control algorithm, due to the higher-voltage nature of the power used by the linear actuator. Also, unlike the Servo Motor and stepper motor used previously, the linear actuator did not have any build-in control methods which could be implemented through the Arduino IDE. Therefore we had to create our whole own control method.

The first step in designing the control for the linear actuator (to achieve a precise angular position of our output panel) was to examine the geometry. The output actuator system is shown in a side (profile) view sketch below where the various relevant angles can be seen.

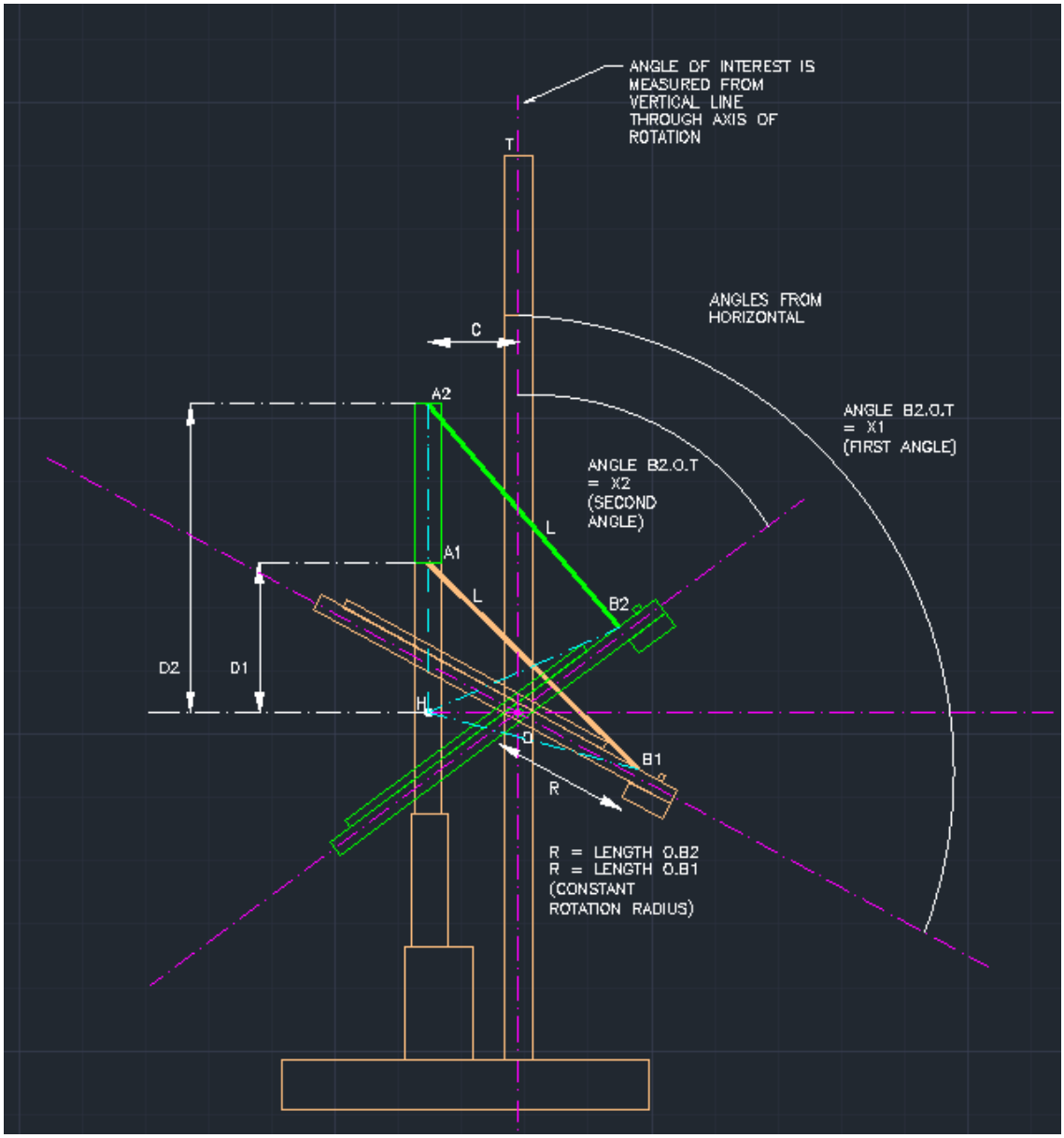


Figure CA-1: Geometric Sketch of Actuator Arm

From the above sketch, we can see the angles that must be calculated to determine the angle from the vertical (angle B2,O,T and B1,O,T). The sketch shows the output actuator system at two distinct positions to aid in the derivation of the output angle as a function of the linear actuator arm position. In order to finalize and quantify our trigonometric derivation based on the geometric sketch above, we measured the actual parameters of our constructed system. We then used those parameters and our

analytical mathematical expression for the vertical angle as a function of the actuator arm position to create a MATLAB model of our system.

The purpose of creating this model was to determine how the linear actuator arm should be moved to achieve a corresponding angular position for the output frame and output solar panel. In other words, we were mapping a linear system to a radial (angular) coordinate system. The MATLAB script for this analysis can be seen below in Figure CA-2. The analytical expression we derived for our function can be seen in closed form within the MATLAB script.

calculateDistances.m | outputActuatorModelsingleLine.m

```
d1 = 17.4; %smallest shaft distance from axis of rotation - 17.4
dfinal=48.5;%largest shaft distance from axis of rotation - 48.5
y = 0:0.1:(dfinal-d1);%vector for travel distances
theta = zeros (1,length(y));%rotation angle vector
ropeLength = 35.5;%constant rope length
supportLength = 18;%constant support length
horizontalLength = 4;%constant distance between shaft and rotation plane

%calculates angle as a function of shaft travel based on manual derivation
for i=1:length(theta)
theta(i) = acos((supportLength^2+horizontalLength^2+(d1+y(i))^2-ropeLength^2)/(2*supportLength*sqrt(horizontalLength^2+(d1+y(i))^2)));
theta(i) = theta(i)-atan(horizontalLength/(d1+y(i)));
end

%plot the angle as a function of shaft travel distance
figure (1);
plot (y,180*theta/pi,'Linewidth',1.5); title('Vertical Distance vs Vertical Panel Travel Angle');
xlabel('Vertical Distance');ylabel('Panel Travel Degrees');

%create linear approximation
line=polyfit(y,180*theta/pi,1);
hold on; plot (y,line(1)*y+line(2));
legend ('actual','linear approximation');

%output the largest, then smallest angles for the specified limits
thetaFirst=180/pi*theta(1)
thetaLast=180/pi*theta(length(y))
```

Command Window

New to MATLAB? See resources for Getting Started.

```
>> outputActuatorModelsingleLine

thetaFirst =

  150.9407


thetaLast =

   30.4614
```

Figure CA-2: MATLAB Simulation Script for Actuator Position vs Panel Angle

If we examine the above MATLAB script, we can see that it has outputted a first possible angle of 30.46 degrees and a last possible angle of 150.94 degrees. These

calculated values agree very closely with our real-world measured positions of 27 and 154 degrees for the range of motion of our output arm construction.

The starting and ending angles mentioned above are intrinsic to our system due to the positioning of the linear actuator relative to the frame, and the stoppers included in our system, which can be seen in the system photos. It is a good indication that our simulation script is reflecting our real system when the angular limits of the simulation script agree closely with our real world parameters.

From the above script, we were able to create a plot of the vertical travel distance (from the top position) vs the resulting output frame angle. From this plot (shown below as Figure CA-3), we can see that the geometric reality of the situation results in a slightly nonlinear relation between the arm position and actuator frame angle. The thicker blue line in the plot is the actual analytical angular position of the output frame.

We can also note from the plot below that our system has been calibrated in the matter described above. Namely, that when the actuator arm is at its highest highest possible position (corresponding to “0” distance), the output angle is approximately 150 degrees. When the actuator arm is at its lowest possible position (corresponding to the bottom of the actuator shaft, approximately 33 cm from the top), the output angle is approximately 30 degrees.

The thing we noted from the plot, however, was that as mentioned the relationship was not perfectly linear. It was close though, so we decided to use a linear approximation to see how closely it would follow the actual system control curve. The linear approximation can be seen superimposed on the plot as the thin red line. From the comparison plot, we decided that the linear approximation was close and would yield a satisfactory result if used in our system. Therefore any error that may result from our control of the output actuator may be due to the use of this linear approximation.

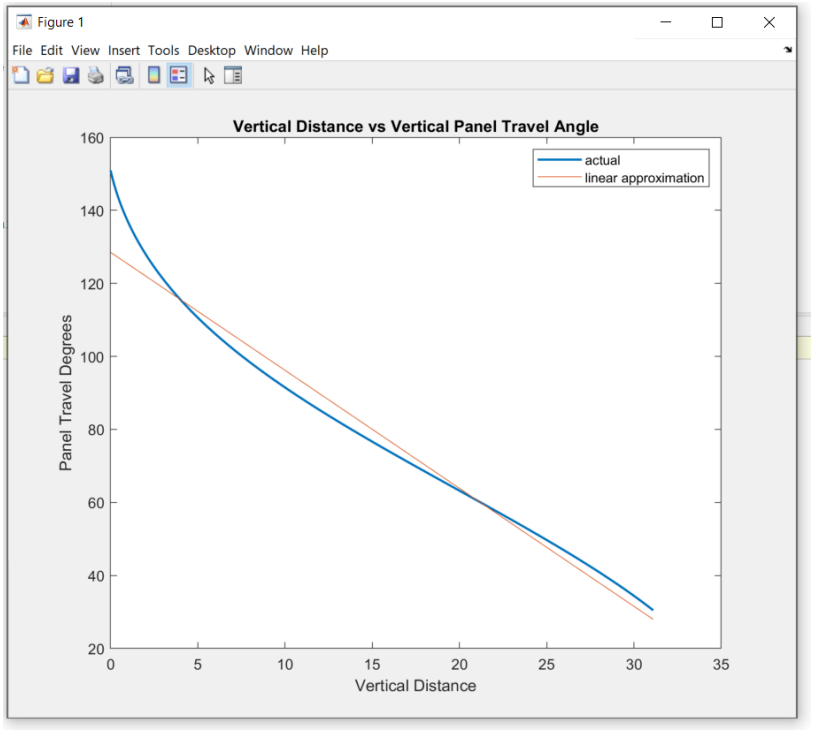


Figure CA-3: MATLAB Simulation Script Plot for Actuator Position vs Panel Angle

Using the linear approximation described above, we developed code for the control of the linear actuator using activation timing to control its position. That is to say, we would calculate the amount of time the actuator should be on (moving up or down) to go to the new position as determined by the input. We attempted to implement this algorithm directly in the control code, as can be seen in our iteration code in the Appendix Figure DC-6. However, it resulted that the Arduino C code did not support the variable types we were using in the way we were using them . Therefore it would not be possible to make the timing calculations directly in the control code.

This created the necessity to create an array of timing differences which the system would use to implement the time delay for the actuator to travel between different angles. The creation of this array of timing differences was done through the use of another MATLAB script, shown below in Figure CA-4. The script uses the linear interpolation methodology we described above to create an array of timing differences for all possible angle differences within our 126-degree range of motion (1 per degree).

Since our input system samples every 7 degrees, the array of values with a timing difference for every degree was more than enough accuracy.

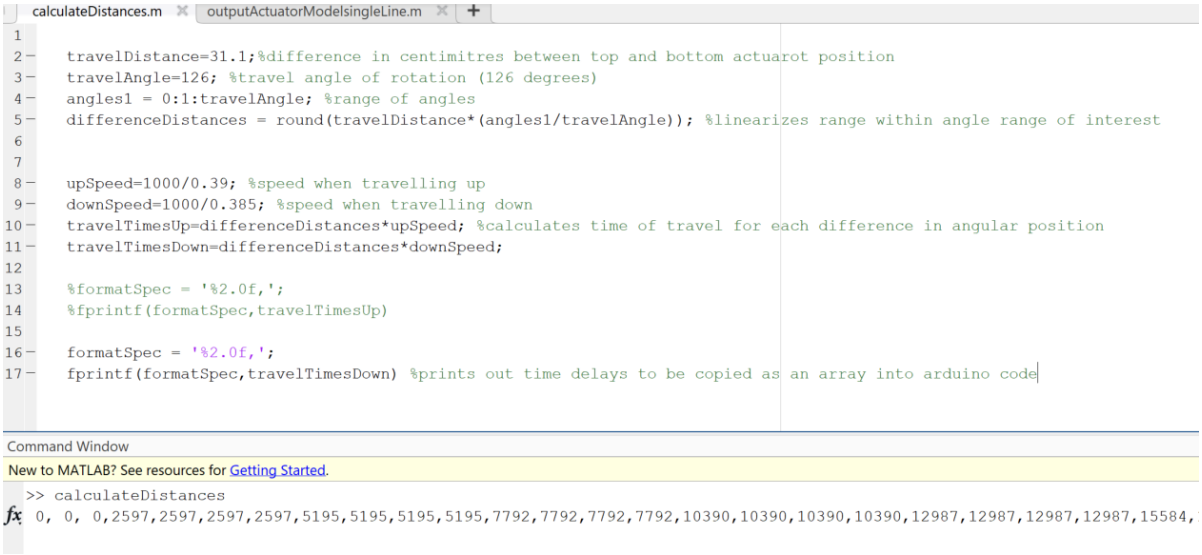


Figure CA-4: MATLAB Script Plot for Creation of Time Delay Array

We implemented the timing difference array calculated by the MATLAB script above in our first full-system prototype control code. The code (in-depth details found in the code comments) can be seen in the Appendix Figure DC-7.

After we completed the control for our whole system, we decided to add an LED display for showing the input voltages tracked by our sensor system. In order to do this, we purchased a digital display module (see specification sheet in Appendix Figure PD-3). In order to familiarize ourselves with the use of this display, a preliminary control test code was used, seen in Appendix Figure DC-8.

Finally, after creating an appropriate function and making necessary adjustments for the inclusion of the LED display, our final control software was complete. It is shown below in its entirety, including all procedural code, the loop code, and the functions. The details for each part of the control software can be found in the comments. Please refer to Figures CA-5a and CA-5b below.

full_servo_plus_actuator_display

```
#include <Servo.h> //includes built-in servo library
#include <Arduino.h>
#include <TM1637Display.h>

// Module connection pins (Digital Pins for display)
#define CLK 2
#define DIO 3

TM1637Display display(CLK, DIO);//create display for input voltages

Servo servo1; //create servo for input sensor
int pos1 = 0; //set starting position

const int in4 = A0; //input from sensor to analog pin 1
const int scanAngle = 7; //degrees for each scan position
const int rangeOfMotion = 126;
int inputVoltage; //variable for displaying input sensor readings

const int in1 = 10; //first motor direction pin
const int in2 = 9; //second motor direction pin


const int numberOfScans = 19; //number of scans for the range and scan angle
int sensorValues[numberOfScans]; //stores the input from the sensor in an array of values for comparison to find max
int newPosition = 0;//tracks new position of system
int currentPosition = 0;//assume start at top position - 27 degrees = 0 degrees on our scale

//uses mathematical model to map each angle difference 0-126 to a corresponding upwards travel time
int travelTimesUp[127] = {0, 0, 0,2564,2564,2564,2564,5128,5128,5128,5128,7692,7692,7692,7692,10256,10256,10256,10256,12821,
12821,12821,12821,15385,15385,15385,15385,17949,17949,17949,17949,20513,20513,20513,20513,23077,23077,23077,23077,25641,25641,
25641,25641,28205,28205,28205,28205,30769,30769,30769,30769,33333,33333,33333,33333,35897,35897,35897,35897,38462,38462,38462,
38462,41026,41026,41026,41026,43590,43590,43590,43590,46154,46154,46154,46154,48718,48718,48718,48718,48718,51282,51282,51282,
51282,53846,53846,53846,53846,56410,56410,56410,56410,58974,58974,58974,58974,61538,61538,61538,61538,64103,64103,64103,64103,
66667,66667,66667,66667,69231,69231,69231,69231,71795,71795,71795,71795,74359,74359,74359,74359,76923,76923,76923,76923,79487,
79487,79487};

//uses mathematical model to map each angle difference 0-126 to a corresponding downwards travel time
int travelTimesDown[127] = {0, 0, 0,2597,2597,2597,2597,5195,5195,5195,5195,7792,7792,7792,7792,10390,10390,10390,10390,12987,12987,
12987,12987,15584,15584,15584,15584,18182,18182,18182,18182,20779,20779,20779,20779,23377,23377,23377,23377,25974,25974,25974,25974,
28571,28571,28571,28571,31169,31169,31169,31169,33766,33766,33766,33766,36364,36364,36364,36364,38961,38961,38961,38961,41558,41558,
41558,41558,44156,44156,44156,44156,46753,46753,46753,46753,49351,49351,49351,49351,51948,51948,51948,51948,54545,54545,54545,
54545,57143,57143,57143,57143,59740,59740,59740,59740,62338,62338,62338,62338,64935,64935,64935,64935,67532,67532,67532,67532,70130,
70130,70130,70130,72727,72727,72727,72727,75325,75325,75325,75325,77922,77922,77922,77922,80519,80519,80519};



void setup() {

servo1.attach(8); // attached motor to digital pin 8
servo1.write(0); // sets up the motor to angle of 0 degrees

pinMode (in1,OUTPUT); //set h-bridge direction pin to output type
pinMode (in2,OUTPUT); //set h-bridge direction pin to output type
digitalWrite(in1,LOW);digitalWrite(in2,LOW);
Serial.begin(9600); //starts serial

display.clear();//clear output display
display.setBrightness(7);//initialize output diaplay

}

void loop() {//this is the main function which the controller will run continuously

delay (15000); //intial delay to aid with setup

//this block will scan the input, find the maximum angle, and send the input sensor to that angle
scan();
newPosition = findMax();
servo1.write(newPosition);

//move the output to the curresponding angle
moveOutput(currentPosition,newPosition);

delay (10000); //delay until next sampling time (in real life 30 mins)
currentPosition = newPosition;//set the current position to the newly moved position

}
```

Figure CA-5a: Final Control Code Full part 1

full_servo_plus_actuator_display

```
#include <Servo.h> //includes built-in servo library
#include <Arduino.h>
#include <TM1637Display.h>

// Module connection pins (Digital Pins for display)
#define CLK 2
#define DIO 3

TM1637Display display(CLK, DIO);//create display for input voltages

Servo servo1; //create servo for input sensor
int pos1 = 0; //set starting position

const int in4 = A0; //input from sensor to analog pin 1
const int scanAngle = 7; //degrees for each scan position
const int rangeOfMotion = 126;
int inputVoltage; //variable for displaying input sensor readings

const int in1 = 10; //first motor direction pin
const int in2 = 9; //second motor direction pin


const int numberOfScans = 19; //number of scans for the range and scan angle
int sensorValues[numberOfScans]; //stores the input from the sensor in an array of values for comparison to find max
int newPosition = 0;//tracks new position of system
int currentPosition = 0;//assume start at top position - 27 degrees = 0 degrees on our scale

//uses mathematical model to map each angle difference 0-126 to a corresponding upwards travel time
int travelTimesUp[127] = {0, 0, 0,2564,2564,2564,2564,5128,5128,5128,5128,7692,7692,7692,7692,10256,10256,10256,10256,12821,
12821,12821,12821,15385,15385,15385,15385,17949,17949,17949,17949,20513,20513,20513,20513,23077,23077,23077,23077,25641,25641,
25641,25641,28205,28205,28205,28205,30769,30769,30769,30769,33333,33333,33333,33333,35897,35897,35897,35897,38462,38462,38462,
38462,41026,41026,41026,41026,43590,43590,43590,43590,46154,46154,46154,46154,48718,48718,48718,48718,48718,51282,51282,51282,
51282,53846,53846,53846,53846,56410,56410,56410,56410,58974,58974,58974,58974,61538,61538,61538,61538,64103,64103,64103,64103,
66667,66667,66667,66667,69231,69231,69231,69231,71795,71795,71795,71795,74359,74359,74359,74359,76923,76923,76923,76923,79487,
79487,79487};

//uses mathematical model to map each angle difference 0-126 to a corresponding downwards travel time
int travelTimesDown[127] = {0, 0, 0,2597,2597,2597,2597,5195,5195,5195,5195,7792,7792,7792,7792,10390,10390,10390,10390,12987,12987,
12987,12987,15584,15584,15584,15584,18182,18182,18182,18182,20779,20779,20779,20779,23377,23377,23377,23377,25974,25974,25974,25974,
28571,28571,28571,28571,31169,31169,31169,31169,33766,33766,33766,33766,36364,36364,36364,36364,38961,38961,38961,38961,41558,41558,
41558,41558,44156,44156,44156,44156,46753,46753,46753,46753,49351,49351,49351,49351,51948,51948,51948,51948,54545,54545,54545,
54545,57143,57143,57143,57143,59740,59740,59740,59740,62338,62338,62338,62338,64935,64935,64935,64935,67532,67532,67532,67532,70130,
70130,70130,70130,72727,72727,72727,72727,75325,75325,75325,75325,77922,77922,77922,77922,80519,80519,80519};



void setup() {

servo1.attach(8); // attached motor to digital pin 8
servo1.write(0); // sets up the motor to angle of 0 degrees

pinMode (in1,OUTPUT); //set h-bridge direction pin to output type
pinMode (in2,OUTPUT); //set h-bridge direction pin to output type
digitalWrite(in1,LOW);digitalWrite(in2,LOW);
Serial.begin(9600); //starts serial

display.clear();//clear output display
display.setBrightness(7);//initialize output diaplay

}

void loop() {//this is the main function which the controller will run continuously

delay (15000); //intial delay to aid with setup

//this block will scan the input, find the maximum angle, and send the input sensor to that angle
scan();
newPosition = findMax();
servo1.write(newPosition);

//move the output to the curresponding angle
moveOutput(currentPosition,newPosition);

delay (10000); //delay until next sampling time (in real life 30 mins)
currentPosition = newPosition;//set the current position to the newly moved position

}
```

Figure CA-5b: Final Control Code Full part 2

Qualitative Analysis of Prototypes

See the prototype descriptions in the subsequent section “Results and Validation” for an explanation of the different prototype iterations used in our project.

The initial prototypes had a number of weaknesses. The input array had 2 major problems and the output array a problem. There was also a collective problem. The first problem in the input array was the materials that were used. The initial prototype was made of plastic building kits with the solar panel taped on. This caused major issues with the stability of the input. The second issue was the stress that the gearing system put on the motor. This puts stress on the motor to hold the panel in the right place. The stress burned out the motor and as a result it needed to be replaced. The issue with the output array was that the solar panel attached is too small to simulate a larger scale array. The collective problem was that both prototypes were built of wood and plastic kits. Neither of these prototypes were representative of the intention to utilize the prototype outside. Both also looked like a preliminary prototype.

The new prototype was built out of metal with a wooden base. Unlike the initial prototype the final was constructed as a single structure. The metal solved the problem with the flimsy plastic buckling under the weight and the wood being unsuitable for outdoors. The new prototype also used a servo motor for the input and an actuator for the output. This solved the problem of the motors burning out as the servo motor was designed for such loads. The input was built using a metal rod and pillow bearings while the output used a rotating stand. Both of these were stable which solved the problem brought on by the initial plastic being flimsy. The metal rotating stand was also large enough to support a larger panel to get better results and better simulate a larger scale construction.

Results and Validation

Prototype

Our prototype as planned in the midterm report was originally designed as per the general block diagram seen below:

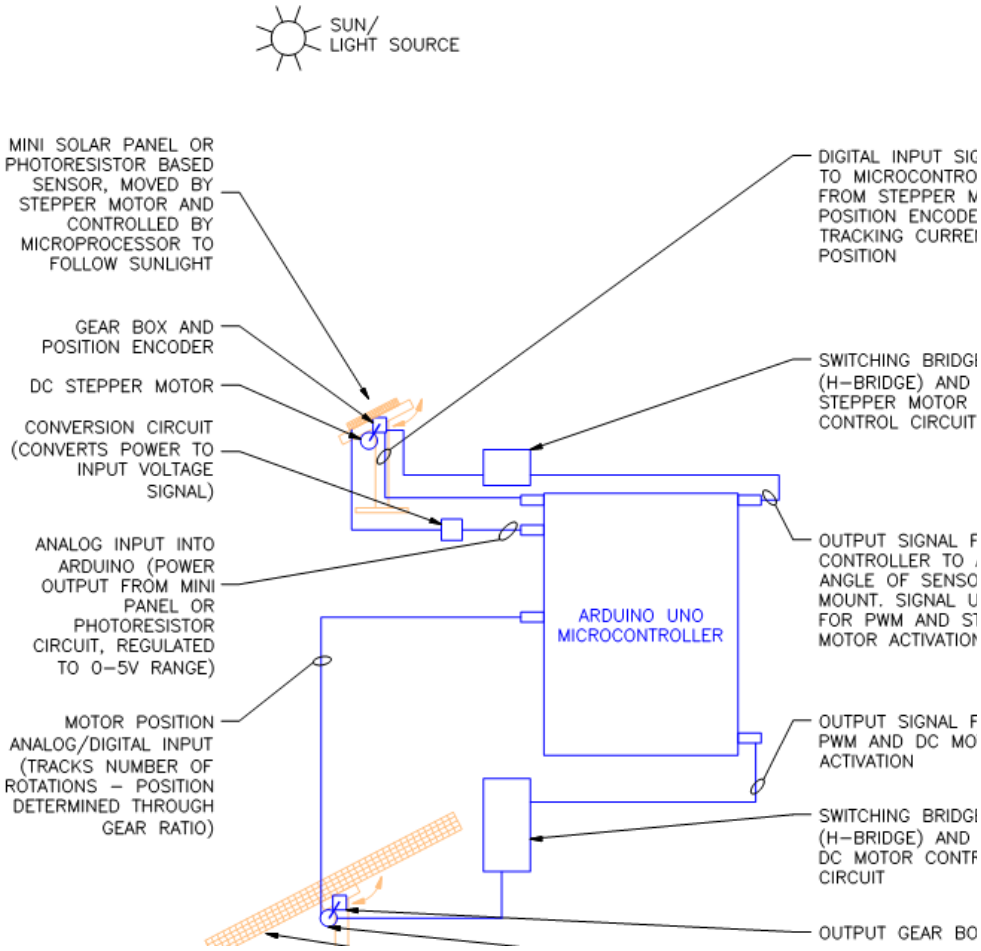


Figure P-1: Old Block Diagram of System - No Longer In use

The construction of the prototype was based on assembling and combining various sub-systems depicted in the block diagram. The team had built each subsystem separately in order to prototype and test the design and functionality, this was done to

make sure that it was constructed correctly. These subsystems were then combined into the final prototype for further testing and iteration process.

The first generation, as identified in the remainder of the report, was constructed accordingly as planned. The input tracking system was built out of a frame which was purchased from Amazon. The small 5V solar panel was attached to the frame using fixtures and was mounted onto a shaft. The shaft, which was connected to a stepper motor, was used to control the angle at which the “tracking solar panel” was turned to.

Close-up images of what the first generation input prototype looked like can be seen in Figure P-x below. We also produced a demonstration video showing the operation of our first generation input sensor system, which can be seen at the following YouTube video link:

https://www.youtube.com/watch?v=8RAkLJ_ANTU

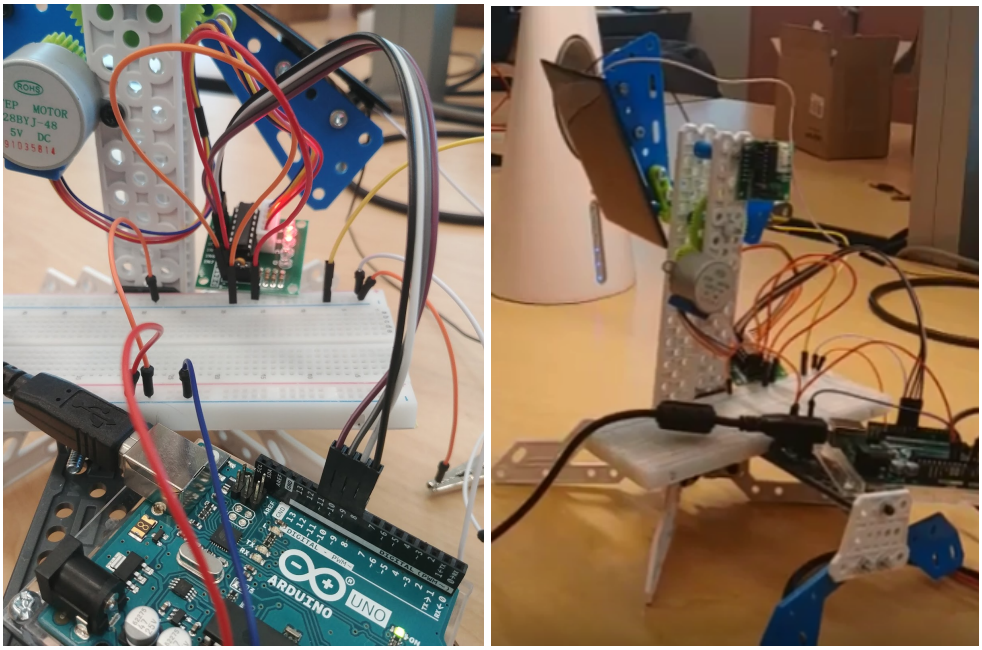


Figure P-2: Original First Iteration Input System Prototype

After conducting some preliminary testing with the tracking subsystem, the stepper motor seemed to have failed and stopped working. After some troubleshooting, it was decided to use a servo motor instead.

The output subsystem of the 1st generation of the prototype was constructed of wood, pillow bearings, an aluminum shaft, 5V solar panel (representing the solar arrays), fixtures and shaft coupling as shown in the image in Figure P-3 below. After some simple wiring and programming, the servo motor had successfully rotated the output solar panel from 0 to 180 degrees on-demand.

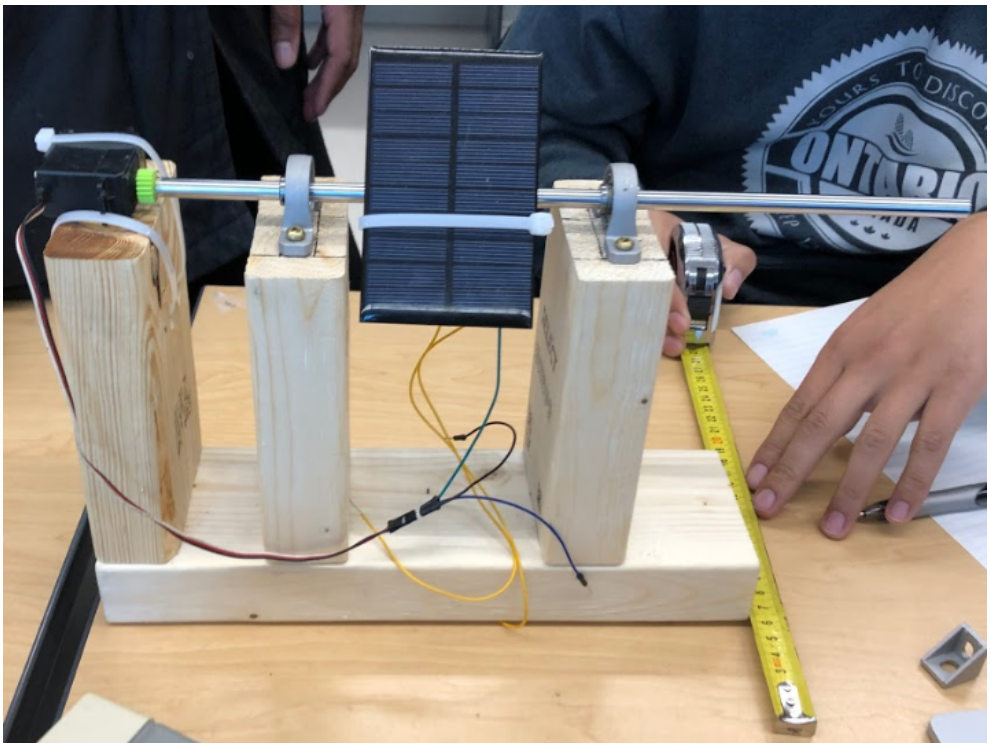


Figure P-3: Original First Iteration Output System Prototype

The first iteration input system and first iteration output system were combined to create a full first-generation prototype working system. A video demonstration of this first generation working prototype can be found at the following YouTube video link:

https://www.youtube.com/watch?v=bD2FWck4ea4

During the team's iterative process on the output subsystem, it was decided that the wood framing be replaced with aluminum extruded metal for more sturdiness and durability. Some images of the construction work, and the final new input aluminum prototype frame can be seen in the images below.

Figure P-4: Metal Shop Construction of New Aluminum Frame

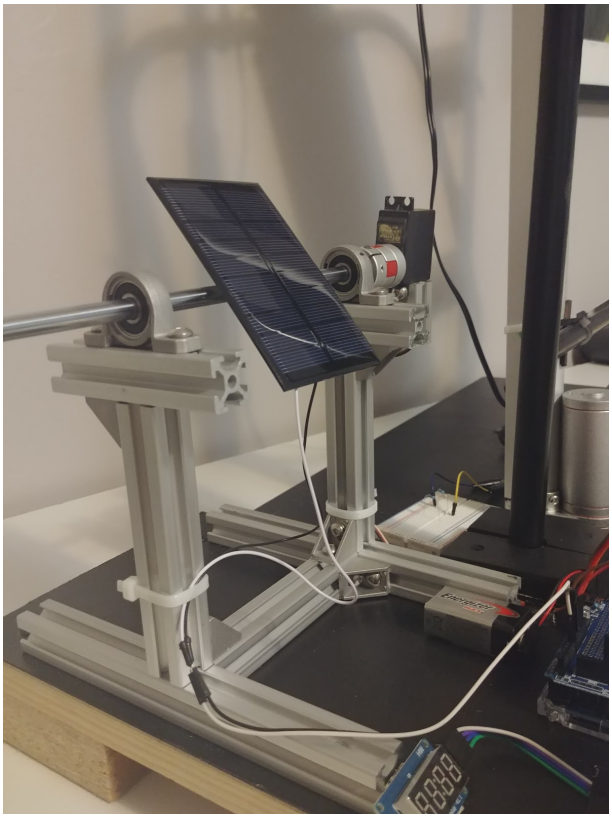


Figure P-5: Final Aluminum Frame Construction

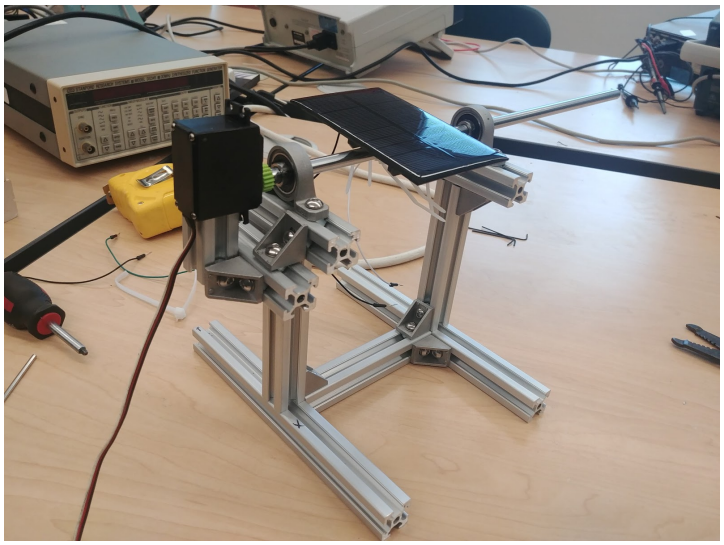


Figure P-6: Aluminum Frame Working Construction

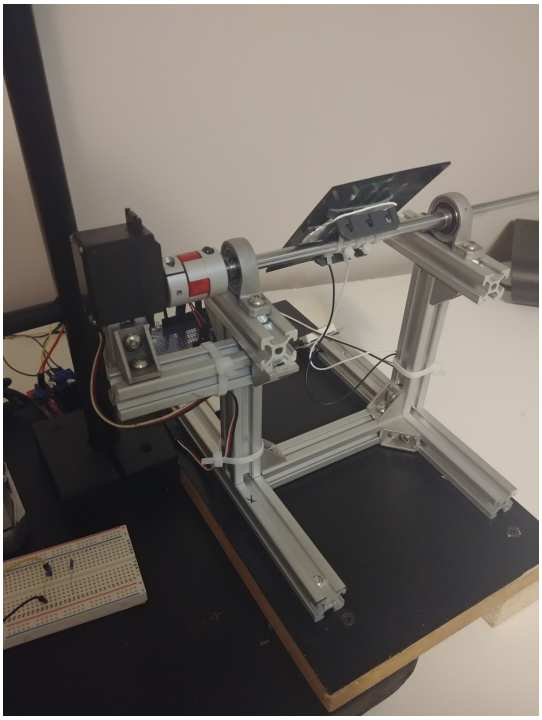


Figure P-7: Aluminum Input System Rear View

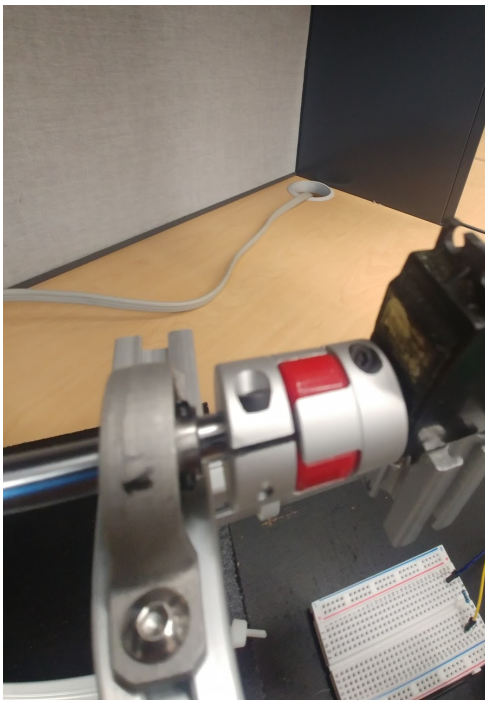


Figure P-8: Input System Shaft Coupling Close-Up View

It was also decided that for demonstrational purposes, all subsystems, including the microcontroller and peripheral electronics be mounted on the same board. Several photos of different views of the different subsystems and the full system of the final prototype can be seen below in Figures P-9 to Fiture P-13 below. The budget of all the expenses towards the prototype are listed in the Appendix - Budget Table.

Figure P-9: Final Complete Prototype System Isometric View

Figure P-10: Final Complete Prototype System Right Side View

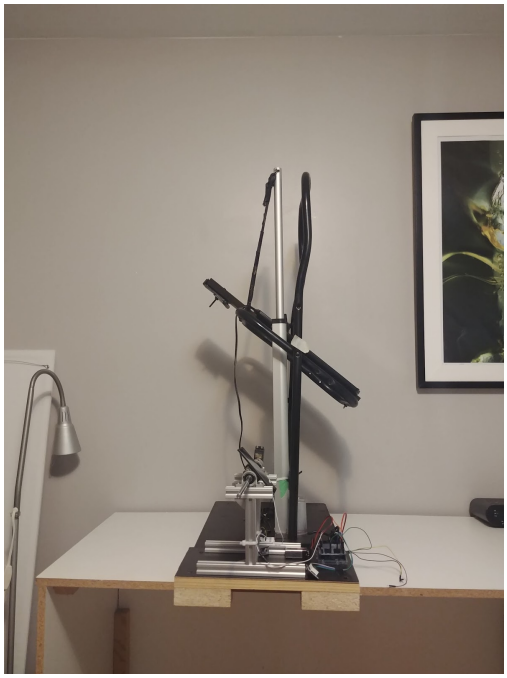


Figure P-11: Final Complete Prototype System Left Side View

Figure P-12: Final Complete Prototype System Rear View

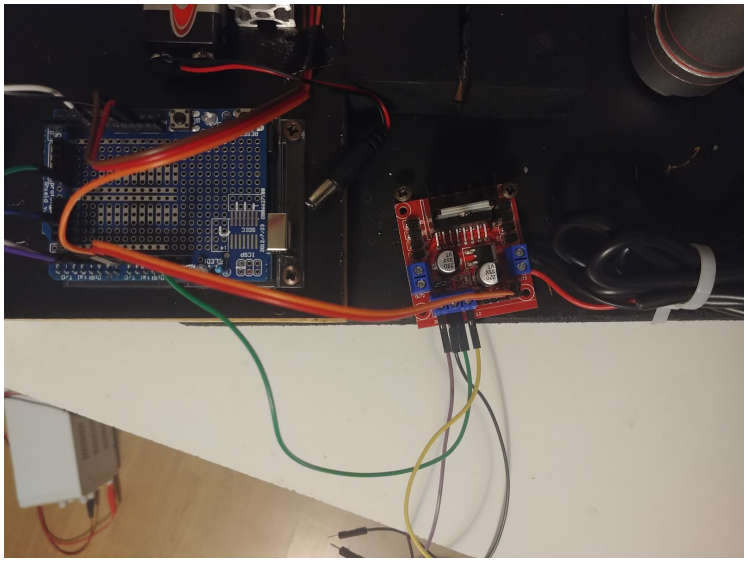


Figure P-13: Final Complete Prototype System Close-Up Control Connections

As described in the “Engineering Analysis” section of the report, several software tools were used in order to construct the two generations of the prototype. SolidWorks was introduced to design a better shaft coupling for the servo motor to mount to the aluminum shaft. Originally, a salvaged piece of plastic was melted to both shafts’ ends which was not ideal for smooth rotation. The final layout of the designed shaft coupling to be laser cut is found in Figure P-14 below. Arduino IDE software was used to program the inputs and outputs of all the hardware in the prototype. The hardware includes the servo motor, linear actuator, Arduino Uno, digital display, etc. Furthermore, MicroCap 11 simulation software was used to model the preliminary design of the sensor system.

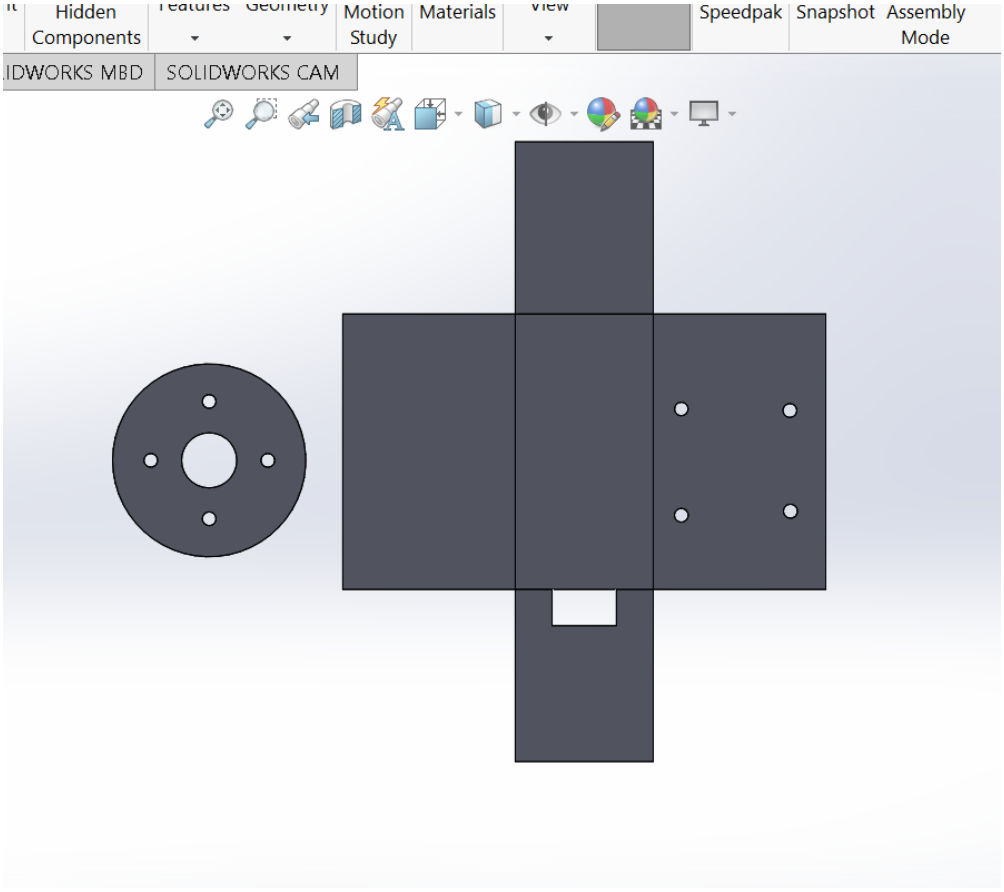


Figure P-14: SolidWorks file of Shaft Coupling and Servo Motor Holder

Testing Strategy/ Validation Protocols

The main objective of our project was to build a system capable of tracking a source of light (sunlight, for scaled-up systems) as the source moves on a single 180º axis of motion. However, to know whether our system works various tests needed to be conducted. These tests consisted of finding out whether our system was capable of successfully tracking any source of light as well as finding out how efficient it is.

The tracking feature of our final prototype was tested by simulating the sun and its trajectory during daytime. In order to simulate sunlight, we decided to use a floor lamp with a bendable frame that allowed us to change angles and direct the light to the panels at any given point in time. Only indoor testing was performed. Therefore, in order to achieve the best and most accurate results, we conducted the tests in a room with no outside lighting coming into the room, or any other light turned on, besides the lamp used for testing. When the light was pointed at the input sensor, it would then identify the light and track it by finding its angle. Once this happens, the output actuator system moves until it matches the angle of the input. After performing a series of tests it was possible to conclude that the system does track the light source successfully.

Now that tracking had been proved to work with no issues, it was time to test for efficiency which is an extremely important aspect of our project. It was crucial to observe that tracking systems would be more efficient when compared with fixed solar systems. We tested how efficient our prototype is, by comparing data with respect to when the output panel was generating power in a fixed position of 45-47 degrees and when tracking was used to find the optimum angle in a range of 0-180 degrees. A series of tests were conducted and measurements of voltage and current were taken in both circumstances. Two multimeters were connected to the output panel and by measuring voltage and current, power generated could be easily calculated. We concluded that using the tracking technique was much more efficient than using fixed

systems. The power increase of our tracking system was 26% higher than that of a fixed solar panel.

Final Results and Validation

We completed construction of our final prototype, after iterations to our design and methodology as described above. The final prototype system consists of an aluminum-frame input sensor system, and a hollow-tube steel output actuator system. The input, output actuation system, microcontroller, and load bank test bed are all mounted to a common lumber (wood) baseboard for stability and completeness. The baseboard is painted black, and the steel output actuator construction was also painted black. The aluminum input sensor frame and output linear actuator were left unpainted. The final system can be seen from a front profile view below.

Figure RV-1: Front View of Final Prototype System Construction

A comprehensive walkthrough demonstration of our hardware prototype can be found at the following YouTube video link:

(WAIKTHROUGH): https://www.youtube.com/watch?v=JttYNmQNMus

A brief written description of the hardware prototype is as follows (although it is recommended to watch the walkthrough video for the full details of the system). On the left, the aluminum frame is the input system. The input system consists of a mini solar panel mounted to the aluminum frame through a steel shaft, which is connected to steel bearings. The shaft is coupled through a 8mm to 12mm shaft coupler to the shaft of a DC servo motor. There are several connections from the input system to the

microcontroller, which can be seen described in more detail in Figure RV-xx and Table RV-xx below.

The input system also includes a small LED display which displays the input millivolt signal coming from the mini solar sensor panel into the Arduino Uno microcontroller.

On the right is the large black hollow-steel frame output actuation system. The output system consists of a frame and a linear actuator which turns the frame via a connection with a rope. The linear actuator is bolted firmly to the common black wood baseboard.

The output linear actuator connects to an H-Bridge relay switching circuit which provides for the control of the actuator by the microcontroller. The H-Bridge circuit can be seen in approximately the middle of the baseboard. Again, a full set of details can be found in Figure RV-2 and Table RV-1 below.

A functional description of the prototype system operation is as follows. The input system periodically scans (motion driven by the servo motor), tracks a light source and saves the position of the light source. The LED display shows the millivolt output in real time as the system is scanning. A video demonstration of just the input system scanning to find the optimal angle can be found at the following YouTube video link:

(INPUT DEMO): https://www.youtube.com/watch?v=qrp9faL0yCg

The position of the light source (the optimum position for the output panel) is then sent to the microcontroller which relays the information through a control algorithm to the H-Bridge circuit which then controls the activation of the linear actuator, allowing the output panel to move to an appropriate position. The whole final prototype system can be seen working together automatically in the video clip at the following YouTube link:

(FULL DEMO): https://www.youtube.com/watch?v=WqGh_Fozv4c

We also prepared a brief walkthrough and demonstration of our software control code, found at the following YouTube video link:

(SOFTWARE DEMO): https://www.youtube.com/watch?v=L51wtnjnL7Y

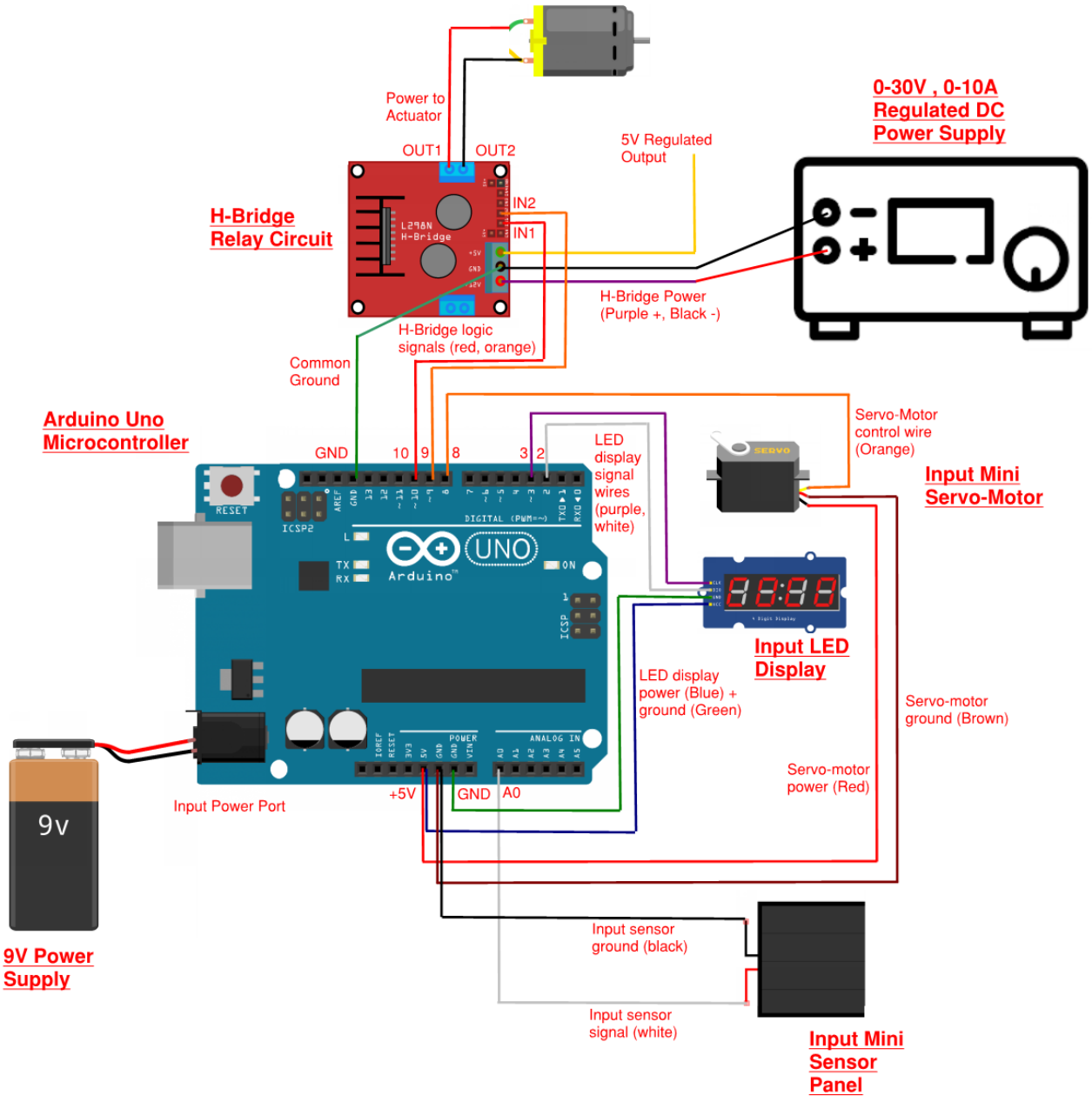


Figure RV-2: Prototype System Electronic and Power Connections

Prototype System Connections						
Connection Description	Wire Color	Type	Analog/Digital	In Terminal	Out Terminal	Voltage (DC Volts)
9V Power Supply	RED/BLACK	POWER	N/A	Arduino - Power port	9V Battery Terminals	9
Input Sensor Ground	BLACK	GND	ANALOG	Arduino - GND	Sensor Panel - (-)	0
Input Sensor Signal	WHITE	LOGICAL INPUT	ANALOG	Arduino - A0	Sensor Panel - (+)	0-5
Servo-Motor Power	RED	GND	N/A	Servo-Motor Power	Arduino - +5V	0
Servo-Motor Ground	BROWN	POWER	N/A	Servo-Motor Ground	Arduino - GND	5
Servo-Motor Control	ORANGE	LOGICAL OUTPUT	DIGITAL	Servo-Motor Control	Arduino - 8	0-5
LED Display Power	BLUE	POWER	N/A	LED - VCC	Arduino - +5V	5
LED Display Ground	GREEN	GND	N/A	LED - GND	Arduino - GND	0
LED Display DIO	PURPLE	LOGICAL OUTPUT	DIGITAL	LED - DIO	Arduino - 3	0-5
LED Display CLK	WHITE	LOGICAL OUTPUT	DIGITAL	LED - CLK	Arduino - 2	0-5
H-Bridge Power	PURPLE	POWER	N/A	H-Bridge - +12V	DC Power Supply - (+)	12
H-Bridge Ground for Power	BLACK	GND	N/A	H-Bridge - GND	DC Power Supply - (-)	0
H-Bridge Common Ground	GREEN	GND	N/A	Arduino - GND	H-Bridge - GND	0
H-Bridge Control 1	RED	LOGICAL OUTPUT	DIGITAL	H-Bridge - IN1	Arduino - 10	0-5
H-Bridge Control 2	ORANGE	LOGICAL OUTPUT	DIGITAL	H-Bridge - IN2	Arduino - 9	0-5
H-Bridge Regulated Output	YELLOW	POWER/LOGICAL	DIGITAL	N/A (open wire)	H-Bridge - +5V	5
Actuator Output Power	RED	POWER	N/A	Linear Actuator - Power	H-Bridge - OUT1	12
Actuator Output Ground	BLACK	GND	N/A	Linear Actuator - GND	H-Bridge - OUT2	0

Table RV-1: List of Prototype System Connections

Efficiency and Power Output Testing

Our group performed a dedicated testing session to determine the efficiency of our tracking system in comparison to a typical static solar array system. This was done to determine the gains that are present when tracking is used when compared to when tracking is not used. This part was determined over a series of tests that compared the power generated when the angle was at 45 degrees and when the panel was tilted to the optimum.

Our test setup can be seen in the photos shown below. As can be seen from the photos, we tested in a mostly dark room to allow for the optimal tracking of the lamp light source. Our test setup methodology was as follows: we connected two multimeters through a load resistor in a resistive load bank. The load bank was

connected to the power output terminals from our output solar panel (see image below for load bank). The first multimeter (top) measured the current through the resistor, and the bottom multimeter measured the voltage across the load resistor. We calculated the power and recorded the values in a table (see Table RV-2 below).

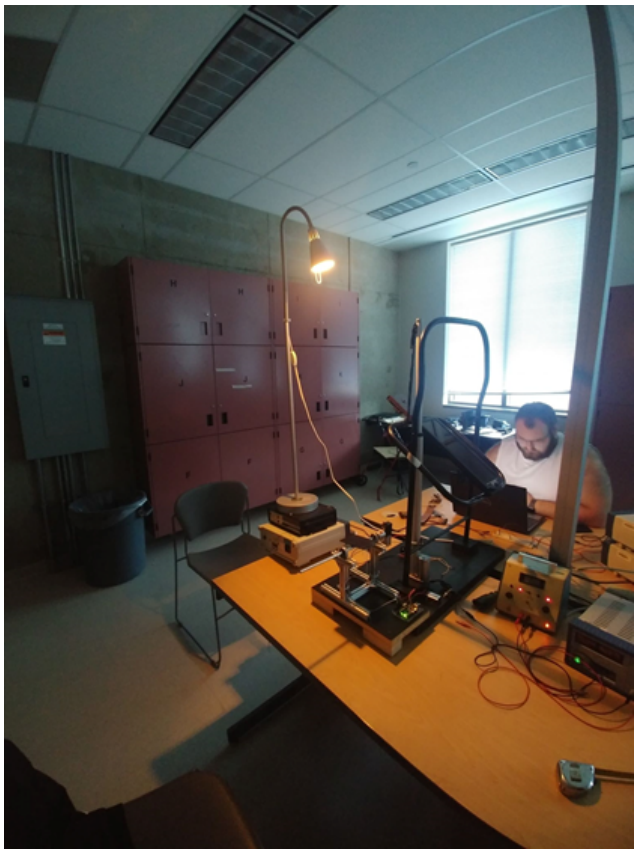


Figure RV-3: Efficiency and Power Testing Setup

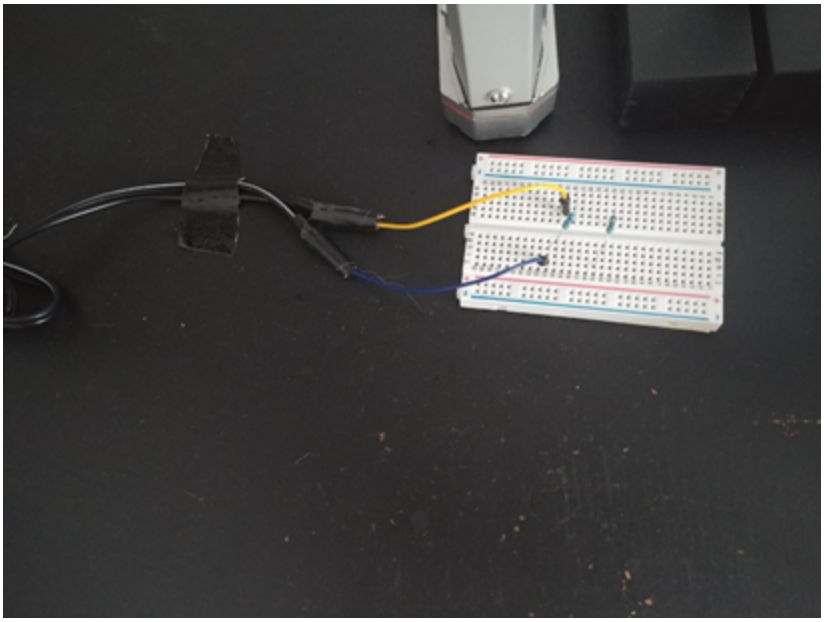


Figure RV-4: Load Bank with Load Resistor and Panel Output Terminals

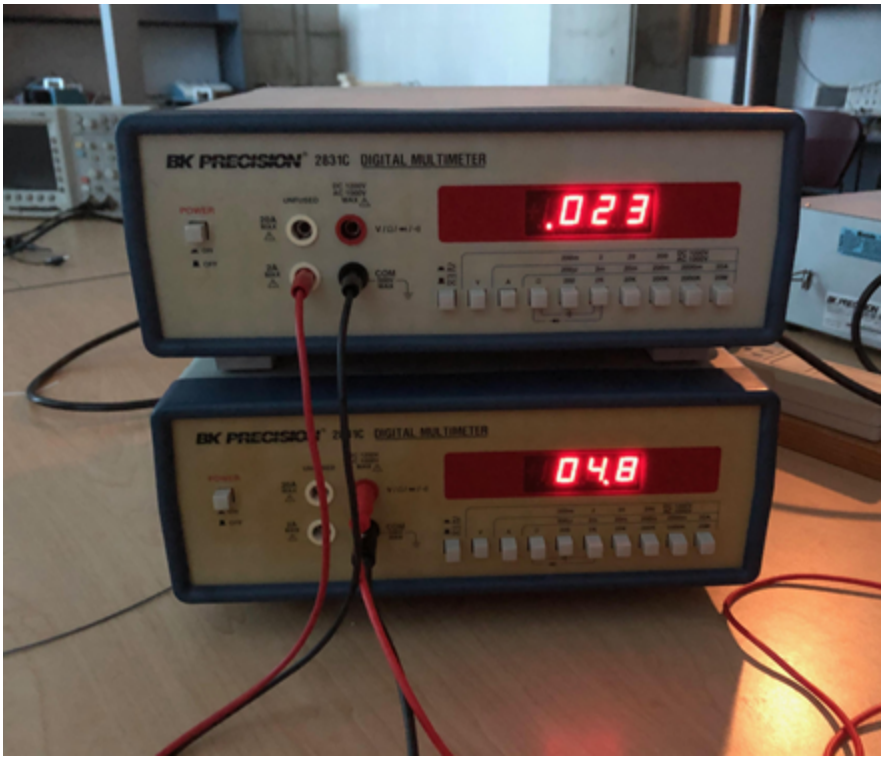


Figure RV-5: Multimeter Test Bed Setup

When we began to test our system, we noticed that our panel was producing an extremely low output current (and therefore output power). In order to determine what

was causing this, we decided to determine the Thevenin equivalent circuit for the solar panel source.

In order to do this we needed to determine the Thevenin voltage and the Thevenin resistance. To determine the Thevenin voltage, we connected multimeter (voltmeter mode) leads across the solar panel terminals in an open circuit. This open circuit voltage is the Thevenin equivalent voltage. To determine the Thevenin resistance, we connected the multimeter (ammeter mode) leads across the solar panel terminals to create a short circuit and measure the short circuit current. This short circuit current was very small, indicating a high input resistance. The Thevenin equivalent resistance was then determined as the open circuit voltage divided by the short circuit current. The Thevenin equivalent circuit we found can be seen below.

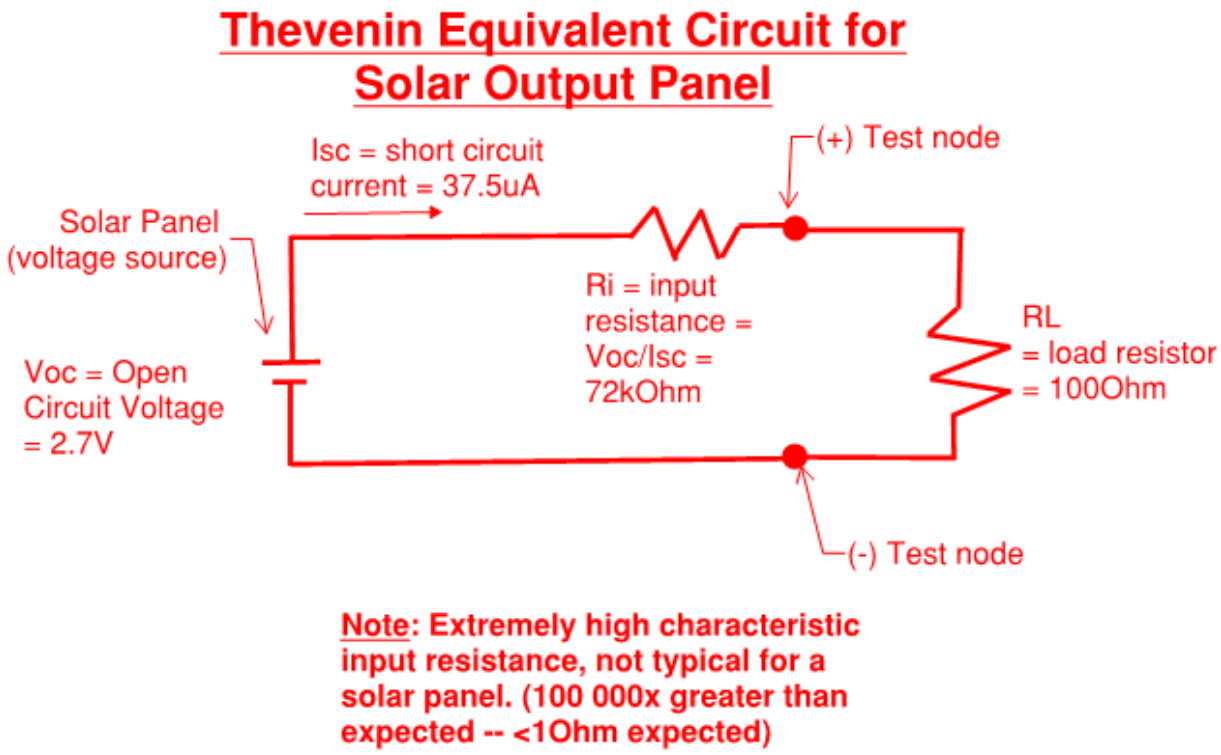


Figure RV-6: Thevenin Equivalent Circuit for Output Solar Panel

We note that the input equivalent resistance of 72kOhm we found is considered extremely high for a solar panel. It is our opinion that the solar panel we used has some

sort of internal malfunction causing this resistance. Nevertheless, for the purposes of comparison, we still used this solar panel despite its abnormally low power production level.

We also verified the abnormal high input resistance by connecting a 100kOhm resistor as the load resistor. When we measured the load current we found a value of 20.1uA, indicating that the current was split approximately in half, which is what we would expect given our calculated 72kOhm input resistance.

The data we collected for our testing was divided into several groups: a group for the panel at the fixed angle (45 degrees from the vertical) and a group for the panel using our solar tracking system. We collected data for two (2) simulated half-days (sunrise to noon), of 9 data points each. This approximately corresponds to 6 hours of daylight (6am to noon). We tested using half days since we were unable to lift our lamp test source very high above our solar panel. therefore when the simulated position extended past the flat position (noon), the fixed panel at 45 degrees had very poor performance since the physical prototype was blocking out the light source.

The data we collected was compared in three (3) histograms (see Figures RV-xx to RV-xx below). The first two histograms compare the frequency of higher power outputs for the fixed panel and the tracking panel. The second histogram tracks the frequency of percentage power increases.These show 2 distinct patterns. The first is that the distribution of power in the tilted histogram is clustered around a higher value and more heavily showing that when the tracker is used the panel tends to collect more power. The second pattern is that the percentage gains are above 40% when using the median value. This leads to a conclusive end of the first part of validation, showing that the tracking does increase power outputted by the panel.

We also plotted the power output of the fixed panel vs the tracking panel at the different positions across the half-day (see Figure RV-xx below). As would be expected, the tracking panel had significantly better performance than the fixed panel when the light source was at positions far from the fixed panel's angle. The two panels

produced similar power output when the light source was near the fixed panel's angle (as would be expected).

We were able to tabulate the data and determined that the approximate average efficiency increase over a simulated half-day was approximately 26%. Therefore our tracking system had a higher performance than a conventional fixed system, and exceeded our stated goal of a 7% power increase (see above in Problem Statement).

It is important to note, however, several caveats with our testing.The testing showed the additional power that could be generated but neglected to mention the power draw of the motors and control systems. This still validates that the design works but it means that the true gains cannot be properly quantized. For our small system, we thought it appropriate to neglect the gain of these systems since our output panel was producing such a small power output, that to consider the external power demand would render our results meaningless. Also, the microcontroller and input sensor system would have an extremely small (negligible) power draw when compared to the power output of a real-world solar farm. Our true efficiency gain may therefore be smaller than what we found, especially when we factor in the power drawn by the actuator motor, which was also not considered.

	Test No.	Fixed Panel Voltage	Fixed Panel Amperege	Fixed Panel Power	Tracked Panel Voltage	Tracked Panel Amperage	Tracked Panel Power	%Increase
Half-Day 1	1	3.2	16	51.2	4.6	23	105.8	106.64%
	2	4.5	22	99	5.6	27	151.2	52.73%
	3	5.1	25	127.5	5.7	28	159.6	25.18%
	4	4.8	24	115.2	5.2	25	130	12.85%
	5	7.1	37	262.7	7.1	37	262.7	0.00%
	6	7.9	40	316	8.1	40	324	2.53%
	7	7.1	33	234.3	7.3	34	248.2	5.93%
	8	4.3	21	90.3	5	27	135	49.50%
	9	4.9	26	127.4	7.3	36	262.8	106.28%
Half-Day 2	10	3	14	42	4.9	23	112.7	168.33%
	11	3.5	17	59.5	4.7	23	108.1	81.68%
	12	4.9	24	117.6	5.5	27	148.5	26.28%
	13	4.3	21	90.3	4.6	22	101.2	12.07%
	14	6.9	34	234.6	6.9	34	234.6	0.00%
	15	7.8	38	296.4	8	39	312	5.26%
	16	7.2	35	252	7.4	36	266.4	5.71%
	17	4.3	20	86	5.1	25	127.5	48.26%
	18	5.1	25	127.5	7.1	35	248.5	94.90%
	Units	mV	uA	pW				
					Average Power Increase:			**25.99%**

Table RV-2: Efficiency Testing Data Over Two Half-Days

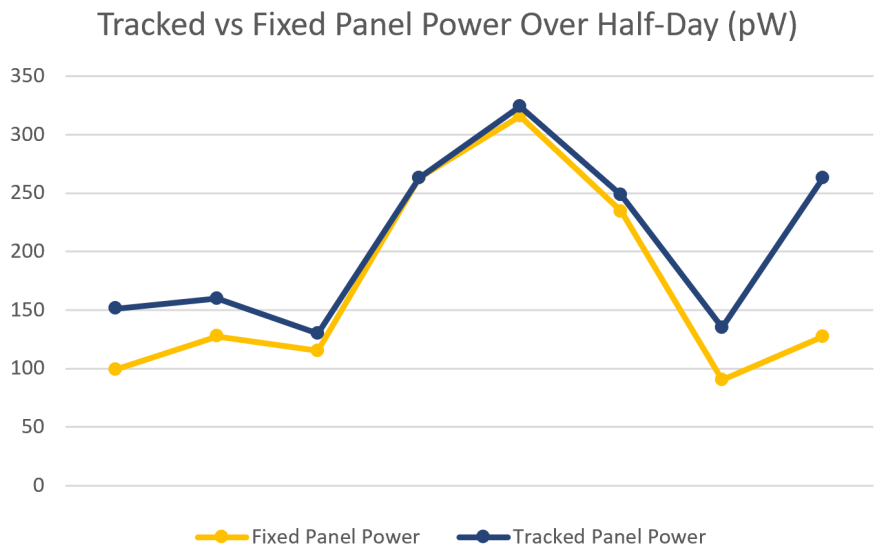


Figure RV-7: Fixed Panel vs System Solar Tracking Panel Power Output

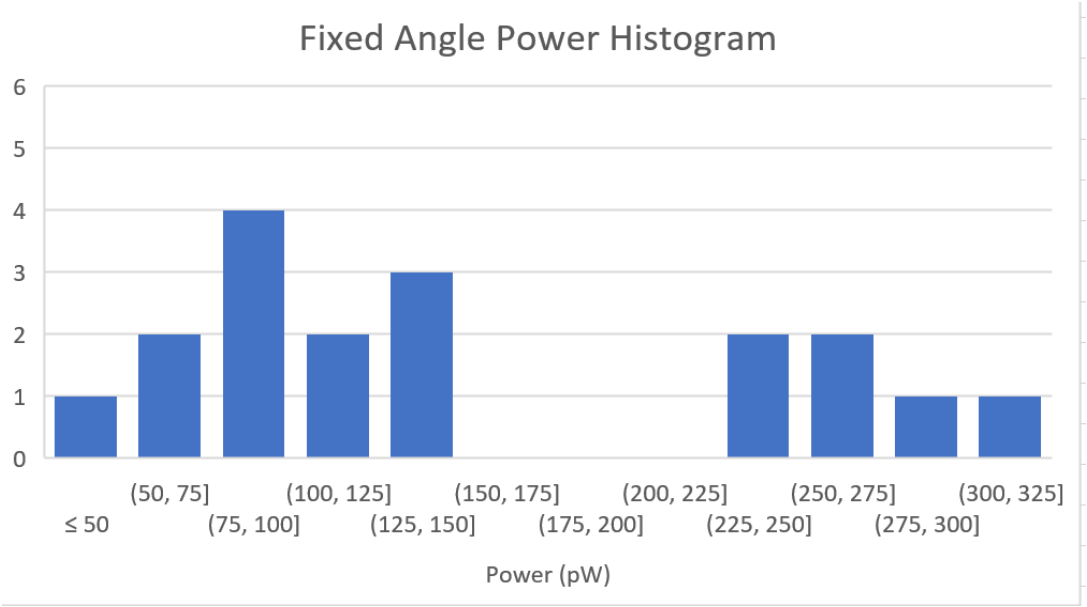


Figure RV-8: Fixed (at 45 degrees) Panel Power Production Histogram

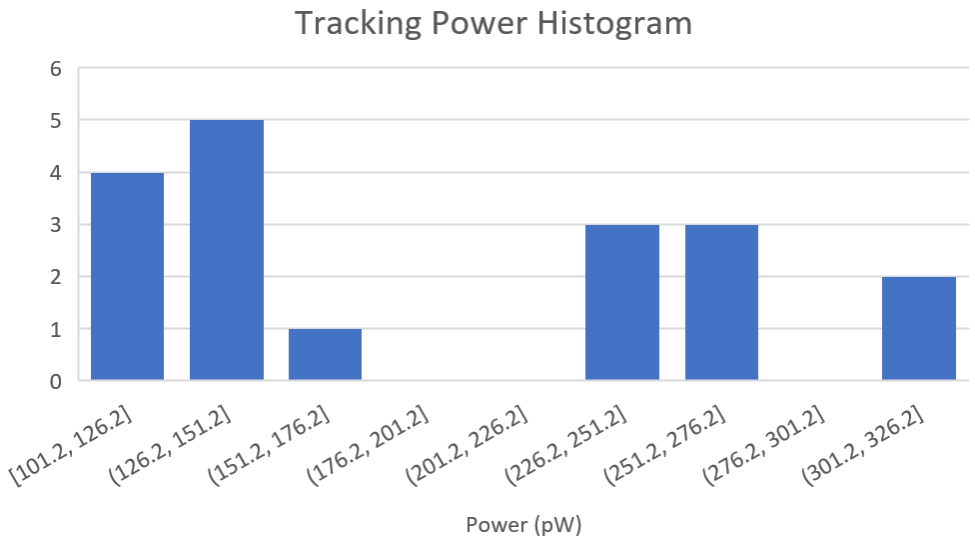


Figure RV-9: Solar Tracking Panel Power Production Histogram

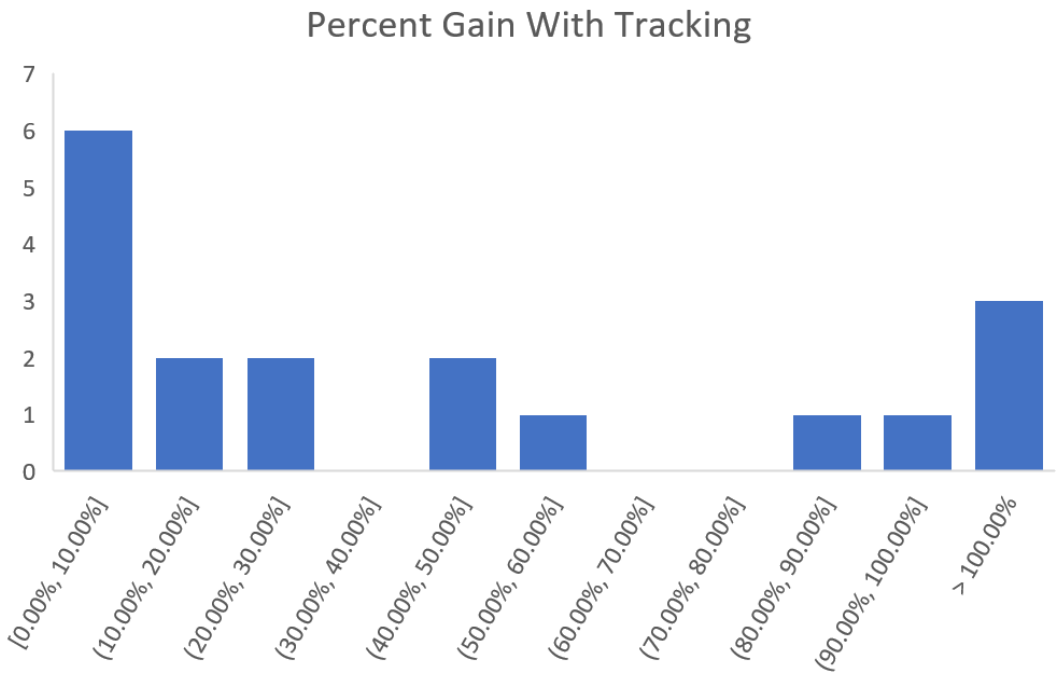


Figure RV-10: Gain Histogram for Tracked Panel vs Static Panel

Conclusions, Future Work and Recommendations

Conclusions

This report has discussed the development of a scaled down solar tracking system which tracks a light source on a single 180 degrees axis of rotation. The objectives of the project were to design, build, and test a solar tracking system which tracks the sun during the day in order to produce higher power generation when compared to solar arrays that are normally installed in fixed positions. All objectives in this project were successfully met. The system performed as expected and even better than expected in aspects such as testing. It was expected an increase of 7% over common solar panels at a fixed position; however, an increase of 26% of power generation was achieved.

Recommendations

After having completed the prototyping, testing the design of our project and analyzing the results, the recommendation of the design of this project can be inferred. After performing the analysis on the results of the prototype's testing, it was calculated that the efficiency of the solar panels can be increased by approximately 26%. Therefore, it is proven that the tracking system that was designed in this project boosts the performance of a conventionally fixed system. The sun's trajectory is influenced by the time of day, geographical location and the season. The goal of this project was to follow the sun's or any light source's trajectory and ensure that the solar arrays are positioned accordingly to produce maximum power.

From preliminary research conducted on the background of this project, it was projected that between 25-45% of increased power will be produced. As this project achieved this goal, this is the biggest benefit and would be recommended for use around the world. Obviously, in large-scale PV farms, there will be maintenance or

repairs required to keep the performance of the tracking systems running consistently. Although it may take some time, generally a couple years, to begin saving money on expenses, it is still unique technology that is worth investing into. Another outlook this project had was to increase power-saving of the tracking system itself so that it doesn't consume much power, thus netting more power produced and will be mentioned in the "Future Work". Thus further research is recommended to develop this option to optimize the efficiency of the design as much as possible. During the design selection phase, other alternatives were also evaluated, especially the dual-axis tracker, which is not ideal for large-scale solar farms and thus, would not be recommended. Furthermore, testing for this project was conducted using an artificial light source to emulate the trajectory of the sun and may have affected the results of the testing. Thus, it is recommended that the testing should have been done outside using the natural trajectory and intensity of the sun to measure the total "extra" power produced.

Future Work

In the future there are two (2) major improvements that would need to be made in a final version. There would also be an option that would need to be explored. The first improvement would be a casing to protect the electronic components. The current design has all of these components accessible to ensure that the parts can be modified and replaced relatively easily if there are any issues. In the future these components would need to be increased to protect them from the elements. The next improvement would be to use more matched components. Currently the actuator is rated for far more weight than it is moving and this can result in efficiency as it consumes more power than a smaller actuator. By matching the panel weight to the actuator capacity more easily then the actuator would operate with greater efficiency in regards to the amount of additional power generated. The options to be explored would first be the concept that was discarded in the initial planning stages. This involved using multiple panels at different fixed angles for an input array rather than the design that was used that had a single panel with a motor. In a larger scale design this would be better as it has fewer

moving parts and does not need a sample period. This concept was discarded because of the need for multiple panels and a larger number of inputs then were available on the microcontroller. On a larger scale the microcontroller could be designed with the application in mind and more panels could be purchased. Finally, another option that was brainstormed during the preliminary prototyping stages was power-saving in the electronics. This could be designed by programming an external timer to turn off the Arduino after periodically completing a sweep of measuring the voltages. This can be accomplished using a MOSFET, a display, two high resistance resistors and some additional wiring.

References

[1] M. Z. Jacobson and V. Jadhav, “World estimates of PV optimal tilt angles and ratios of sunlight incident upon tilted and tracked PV panels relative to horizontal panels,” *Solar Energy*, vol. 169, pp. 55–66, Apr. 2018.

[2] A. Nayyar and V. Puri, "A review of Arduino boards, Lilypads & Arduino shields," in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, New Delhi, 2016, pp. 1485-1492.

[3] T. Akiki, L. Eid, S. Abboud, B. Nehme, "Developing a test platform for PV Panels", *International Conference on Microelectronics*, 2017.

[4] S. V. Mitrofanov, D. K. Baykasenov and M. A. Suleev, "Simulation Model of Autonomous Solar Power Plant with Dual-Axis Solar Tracker," in *2018 International Ural Conference on Green Energy (UralCon)*, Chelyabinsk, 2018, pp. 90-96.

[5] J. G. Elerath, "Solar tracker effectiveness: It's all about availability," in *2017 IEEE International Telecommunications Energy Conference (INTELEC)*, Broadbeach, QLD, 2017, pp. 156-162.

[6] M. Banzi and M.Shiloh, *Getting Started With Arduino, Third Edition*, pp 46. Maker Media, 2015

[7] L. Ali, L. Rahman and S. Akhter, "Module-based Edukit for teaching and learning 8051 microcontroller programming," *2017 IEEE International Conference on Telecommunications and Photonics (ICTP)*, Dhaka, 2017, pp. 57-61.

[8] Arduino Staff, “Arduino Software (IDE)” July 2019
https://www.arduino.cc/en/Guide/Environment

Appendices

Detailed Calculations/ Design Iteration Details

stepper_motor_control_test

```
#include <Stepper.h>
const int stepsPerRevolution = 64; // steps per revolution for our motor

// initialize the stepper library on pins 8 through 11:
Stepper myStepper(stepsPerRevolution, 2, 3, 4, 5);
int stepCount = 0; // number of steps the motor has taken

void setup() {
// nothing inside setup yet
}

void loop() {
// read the sensor value:

int sensorReading = analogRead(A0);
// map it to a range from 0 to 100:

int motorSpeed = map(sensorReading, 0, 1023, 0, 100);
// set the motor speed:

if (motorSpeed > 0) {
myStepper.setSpeed(motorSpeed);
// step 1/64 of a revolution:

myStepper.step(stepsPerRevolution / 64);

}

}
```

Figure DC-1: First Test of Stepper Motor Control Code

stepper_motor_control_test_new__

```
#include <Stepper.h>
const int stepsPerRevolution = 64; // steps per revolution for our motor

// initialize the stepper library on pins 8 through 11:
Stepper myStepper(stepsPerRevolution, 8, 10, 9, 11);

const int in1 = A0;
const int stepAngle;

int maxCheckPosition = 0;
int maxCheck = 0;

int stepCount = 0;
int sensorValues[20];
int stepCountInteger;
int stepsBack;


void setup() {
// nothing inside setup yet
myStepper.setSpeed(120);


}

void loop() {

stepCountInteger=0;
maxCheck=0;
maxCheckPosition = 0;

while (stepCountInteger<20){
delay (200);
sensorValues[stepCountInteger] = analogRead(in1);

myStepper.step(180);
stepCountInteger = stepCountInteger+1;

}

for (int i=0;i<20;i++) {
  if (sensorValues[i] > maxCheck){
    maxCheck = sensorValues[i];
    maxCheckPosition = i;
  }
}

stepsBack=-180*(20-maxCheckPosition);
myStepper.step(stepsBack);

delay (10000);

stepsBack = -180*maxCheckPosition;
myStepper.step(stepsBack);

delay (20000);

}
```

Figure DC-2: First Input Prototype Scanning Code

full_control_test_commented

```
#include <Stepper.h> //includes built-in stepper motor library
#include <Servo.h> //includes built-in servo library

const int stepsPerRevolution = 64; // steps per revolution for our motor

Servo servo1; //create servo
int pos1 = 0; //set starting position

// initialize the stepper library on pins 8 through 11:
Stepper myStepper(stepsPerRevolution, 8, 10, 9, 11);

const int in1 = A0; //input from sensor to analog pin 1
const int scanAngle = 9; //degrees for each scan position
int numberOfScans = round(180/scanAngle); //number of scans for a 180 degree range
const int stepsPerScan = 20*scanAngle; //for our gear ratio, the stepper takes 20 steps per degree

int maxCheckPosition = 0; //variable to compare position with a maximum value
int maxCheck = 0; //variable to check and store the maximum analog input

int sensorValues[numberOfScans]; //stores the input from the sensor in an array of values for comparison to find max
int stepCountInteger; //counts the number of steps taken
int stepsBack; //variable for stepping the sensor back to position


void setup() {
myStepper.setSpeed(240); //sets stepper motor speed (0-255)

servo1.attach(6); // attached motor to digital pin 6
servo1.write(0); //- sets up the motor to angle of 0 degrees

}

void loop() {

delay (5000); //intial delay to aid with setup

stepCountInteger=0; //resets the number of samples
maxCheck=0; //resets the maximum value
maxCheckPosition = 0; //resets the maximum position

while (stepCountInteger<numberOfScans){
delay (150); //small delay to let voltage stabilize at each position
sensorValues[stepCountInteger] = analogRead(in1); //reads in the analog input at each position, stores to array

myStepper.step(stepsPerAngle); //the motor steps to the next sampling position;
stepCountInteger = stepCountInteger+1; //increments the number of samples taken for loop condition
}

//this for loop will find the maximum stored input value
//from the input sensor, and find the position
for (int i=0;i<numberOfScans;i++) { //scans through for the length of the stored array
  if (sensorValues[i] > maxCheck){
    maxCheck = sensorValues[i]; //sets the new maximum as the largest value
    maxCheckPosition = i; //stores the position of the maximum value
  }
}
//steps backwards to position with maximum stored input
stepsBack=-stepsPerScan*(numberOfScans-maxCheckPosition);
myStepper.step(stepsBack);

servo1.write(9*maxCheckPosition);//writes maximum position to output servo

delay (1000); //delay to show maximum position

//steps backward to position 0
stepsBack = -stepsPerScan*maxCheckPosition;
myStepper.step(stepsBack);

delay (30000); //delay until next sampling time (in real life 30 mins)

}
```

Figure DC-3: First Full Automatic System Prototype Code

full_control_test_new

```
#include <Servo.h> //includes built-in servo library

Servo servo1; //create servo for input sensor
int pos1 = 0; //set starting position

const int in1 = A0; //input from sensor to analog pin 1
const int scanAngle = 7; //degrees for each scan position
int rangeOfMotion = 126;
//float scanNumber = round(rangeOfMotion/scanAngle); //number of scans for a 126 degree range
const int numberOfScans = 19; //number of scans for the range and scan angle
int sensorValues[numberOfScans]; //stores the input from the sensor in an array of values for comparison to find max

void setup() {

servo1.attach(8); // attached motor to digital pin 8
servo1.write(0); // sets up the motor to angle of 0 degrees
}

void loop() {

delay (7000); //intial delay to aid with setup

scan();

servo1.write(findMax());

delay (10000); //delay until next sampling time (in real life 30 mins)
}

//the scan() function scans through the range of motion and stores the input sensor values
int scan (){
  int sampleCount = 0; //reset counting number of samples
  //int samples[numberOfScans];//create array to store scanned values
  servo1.write(0); //reset servo position

  while (sampleCount<numberOfScans){
    delay (500); //small delay to let voltage stabilise at each position
    sensorValues[sampleCount] = analogRead(in1); //reads in the analog input at each position, stores to array
    sampleCount = sampleCount+1; //increments the number of samples taken for loop condition
    servo1.write(sampleCount*scanAngle); //write the new sampling position
  }

}

//the findMax() function finds the angle with the maximum light flux input from the sensor
int findMax (){

  int maxCheck = 0; //comparison variable to find maximum
  int maxCheckPosition = 0; //stores maximum position

  //this for loop will find the maximum stored input value from the input sensor, and find the position
  for (int i=0;i<numberOfScans;i++) { //scans through for the length of the stored array
    if (sensorValues[i] > maxCheck){
      maxCheck = sensorValues[i]; //sets the new maximum as the largest value
      maxCheckPosition = i; //stores the position of the maximum value
    }
  }

  return scanAngle*maxCheckPosition; //returns the maximum position in degrees
}
```

Figure DC-4: First Full Automatic System Prototype Code with Functions

servo_as_input

```
#include <Servo.h> //includes built-in servo library

Servo servo1; //create servo for input sensor
int pos1 = 0; //set starting position

const int in1 = A0; //input from sensor to analog pin 1
const int scanAngle = 7; //degrees for each scan position
int rangeOfMotion = 126;
//float scanNumber = round(rangeOfMotion/scanAngle); //number of scans for a 126 degree range
const int numberOfScans = 19; //number of scans for the range and scan angle
int sensorValues[numberOfScans]; //stores the input from the sensor in an array of values for comparison to find max
int newPosition = 0;//tracks new position of system
int currentPosition = 0;//tracks current position of system

void setup() {

servo1.attach(8); // attached motor to digital pin 8
servo1.write(0); // sets up the motor to angle of 0 degrees
}

void loop() {

delay (15000); //intial delay to aid with setup

scan();

newPosition = findMax();
servo1.write(newPosition);


delay (150000); //delay until next sampling time (in real life 30 mins)
currentPosition = newPosition;
}

//the scan() function scans through the range of motion and stores the input sensor values
int scan (){
  int sampleCount = 0; //reset counting number of samples
  //int samples[numberOfScans];//create array to store scanned values
  servo1.write(0); //reset servo position

  while (sampleCount<numberOfScans){
    delay (400); //small delay to let voltage stabilize at each position
    sensorValues[sampleCount] = analogRead(in1); //reads in the analog input at each position, stores to array
    sampleCount = sampleCount+1; //increments the number of samples taken for loop condition
    servo1.write(sampleCount*scanAngle); //write the new sampling position
  }

}

//the findMax() function finds the angle with the maximum light flux input from the sensor
int findMax (){

  int maxCheck = 0; //comparison variable to find maximum
  int maxCheckPosition = 0; //stores maximum position

  //this for loop will find the maximum stored input value from the input sensor, and find the position
  for (int i=0;i<numberOfScans;i++) { //scans through for the length of the stored array
    if (sensorValues[i] > maxCheck){
      maxCheck = sensorValues[i]; //sets the new maximum as the largest value
      maxCheckPosition = i; //stores the position of the maximum value
    }
  }

  return scanAngle*maxCheckPosition; //returns the maximum position in degrees
}
```

Figure DC-5: Reprogramming Servo Motor as Input Scanner Code

linear_actuator_control_new

```
  int in1 = 10; //first motor direction pin
  int in2 = 9; //second motor direction pin

  int currentPosition = 1; //assume start at top position - 27 degrees = 0 degrees on our scale
  int newPosition = 0;

  //uses mathematical model to map each angle from 0-126 to a corresponding vertical shaft distance
  int positionMap[127] = {31.2,30.9,30.6,30.3,29.9,29.6,29.3,29.0,28.7,28.4,28.1,27.8,27.4,27.1,26.8,26.5,26.2,25.9,25.6,
  25.3,24.9,24.6,24.3,24.0,23.7,23.4,23.1,22.8,22.4,22.1,21.8,21.5,21.2,20.9,20.6,20.3,19.9,19.6,19.3,19.0,18.7,18.4,18.1,
  17.8,17.4,17.1,16.8,16.5,16.2,15.9,15.6,15.3,14.9,14.6,14.3,14.0,13.7,13.4,13.1,12.8,12.4,12.1,11.8,11.5,11.2,10.9,10.6,
  10.3,9.9,9.6,9.3,9.0,8.7,8.4,8.1,7.8,7.4,7.1,6.8,6.5,6.2,5.9,5.6,5.3,5.0,4.8,4.7,4.6,4.5,4.4,4.3,4.1,4.0,3.9,3.8,3.7,3.5,
  3.4,3.3,3.2,3.1,3.0,2.8,2.7,2.6,2.5,2.4,2.2,2.1,2.0,1.9,1.8,1.7,1.5,1.4,1.3,1.2,1.1,0.9,0.8,0.7,0.6,0.5,0.3,0.2,0.1,0.0};
  float fastFactor = 0.38;

  float upSpeed = 2564.1; // equal to 1/upwards speed x 1000ms = 1/0.392*1000
  float downSpeed = 2697.4; // equal to 1/downwards speed x 1000ms = 1/0.385*1000

  const int scanAngle = 7; //degrees for each scan position
  int rangeOfMotion = 126;
  const int numberOfScans = 19; //number of scans for the range and scan angle

void setup() {
  // put your setup code here, to run once:

  pinMode (in1,OUTPUT); //set h-bridge direction pin to output type
  pinMode (in2,OUTPUT); //set h-bridge direction pin to output type
  digitalWrite(in1,LOW);digitalWrite(in2,LOW);
  Serial.begin(9600); //starts serial

}

void loop() {

  //digitalWrite(in1,HIGH);digitalWrite(in2,LOW);
  delay (15000);
  newPosition = 25;
  moveOutput (currentPosition,newPosition);
  currentPosition = newPosition;
  delay (20000);

  newPosition = 80;
  moveOutput (currentPosition,newPosition);
  currentPosition = newPosition;
  delay (25000);

  newPosition = 125;
  moveOutput (currentPosition,newPosition);
  currentPosition = newPosition;
  delay (20000);
}

int moveOutput (int currentPosition, int newPosition){

  float currentDistance = positionMap[currentPosition];
  float newDistance = positionMap[newPosition];
  float travelTime;

      if (newDistance<=5){ //if the new distance is within the sped up zone

              if (currentDistance>5){ //this case is when the first distance is outside the sped zone, will always move down into fast zone

                      //this block will move down to 5 at regular speed
                      travelTime = round((currentDistance-5)*downSpeed);
                      digitalWrite(in1,LOW);digitalWrite(in2,HIGH); // moves shaft motor counterclockwise (in - red out1, black out2)
                      delay (travelTime);
                      digitalWrite(in1,LOW);digitalWrite(in2,LOW);//stop movement

                      //this block will move down the rest of the way for less time since the angular velocity is increased
                      travelTime = round(fastFactor*(5-newDistance)*downSpeed); //slow down movement by travelling less time
                      digitalWrite(in1,LOW);digitalWrite(in2,HIGH); // moves shaft motor counterclockwise (in - red out1, black out2)
                      delay (travelTime);
                      digitalWrite(in1,LOW);digitalWrite(in2,LOW);//stop movement

              }

              else {//in this case both the old and new positions are within the fast zone

                        if (newDistance<=currentDistance){//moves down if the old position is above the new
                                  travelTime = round(fastFactor*(currentDistance-newDistance)*downSpeed); //slow down movement by travelling less time
                                  digitalWrite(in1,LOW);digitalWrite(in2,HIGH); // moves shaft motor counterclockwise (in - red out1, black out2)
                                  delay (travelTime);
                                  digitalWrite(in1,LOW);digitalWrite(in2,LOW);//stop movement
                        }
                        else{//moves up if the new position is above the old
                                  travelTime = round(fastFactor*(newDistance-currentDistance)*upSpeed); //slow down movement
                                  digitalWrite(in1,HIGH);digitalWrite(in2,LOW); // moves shaft motor clockwise (out - red out1, black out2)
                                  delay (travelTime);
                                  digitalWrite(in1,LOW);digitalWrite(in2,LOW); //stop movement
                        }

              }

      }

      if (currentDistance<5) { //in this case the current position is inside the fast zone, and the new position is not, so it will be above, move up


              travelTime = round(fastFactor*(5-currentDistance)*upSpeed); //slow down movement by travelling less time
              digitalWrite(in1,HIGH);digitalWrite(in2,LOW); // moves shaft motor clockwise (out - red out1, black out2)
              delay (travelTime);
              digitalWrite(in1,LOW);digitalWrite(in2,LOW); //stop movement

              travelTime = round((newDistance-5)*upSpeed);
              digitalWrite(in1,HIGH);digitalWrite(in2,LOW); // moves shaft motor clockwise (out - red out1, black out2)
              delay (travelTime);
              digitalWrite(in1,LOW);digitalWrite(in2,LOW); //stop movement


      }

      else { //in this case the new position and current position are outside the fast zone

              if (newDistance<=currentDistance){//in the case of the new position below the old position, move down
                        travelTime = round((currentDistance-newDistance)*downSpeed);
                        digitalWrite(in1,LOW);digitalWrite(in2,HIGH); // moves shaft motor counterclockwise (in - red out1, black out2)
                        delay (travelTime);
                        digitalWrite(in1,LOW);digitalWrite(in2,LOW);//stop movement
              }
              else{
                        travelTime = round((newDistance-currentDistance)*upSpeed); //in the case of the old position below the new position, move up
                        digitalWrite(in1,HIGH);digitalWrite(in2,LOW); // moves shaft motor clockwise (out - red out1, black out2)
                        delay (travelTime);
                        digitalWrite(in1,LOW);digitalWrite(in2,LOW); //stop movement
              }

      }
```

Figure DC-6: First Linear Actuator Control Test Code

full_servo_plus_actuator

```
#include <Servo.h> //includes built-in servo library

Servo servo1; //create servo for input sensor
int pos1 = 0; //set starting position

const int in4 = A0; //input from sensor to analog pin 1
const int scanAngle = 7; //degrees for each scan position
const int rangeOfMotion = 126;

const int in1 = 10; //first motor direction pin
const int in2 = 9; //second motor direction pin


const int numberOfScans = 19; //number of scans for the range and scan angle
int sensorValues[numberOfScans]; //stores the input from the sensor in an array of values for comparison to find max
int newPosition = 0;//tracks new position of system
int currentPosition = 0;//assume start at top position - 27 degrees = 0 degrees on our scale

//uses mathematical model to map each angle difference 0-126 to a corresponding upwards travel time
int travelTimesUp[127] = {0, 0, 0,2564,2564,2564,2564,5128,5128,5128,5128,7692,7692,7692,7692,10256,10256,10256,10256,12821,
12821,12821,12821,15385,15385,15385,15385,17949,17949,17949,17949,20513,20513,20513,20513,23077,23077,23077,23077,25641,25641,
25641,25641,28205,28205,28205,28205,30769,30769,30769,30769,33333,33333,33333,33333,35897,35897,35897,35897,38462,38462,38462,
38462,41026,41026,41026,41026,43590,43590,43590,43590,46154,46154,46154,46154,48718,48718,48718,48718,51282,51282,51282,
51282,53846,53846,53846,53846,56410,56410,56410,56410,58974,58974,58974,58974,61538,61538,61538,61538,64103,64103,64103,64103,
66667,66667,66667,66667,69231,69231,69231,69231,71795,71795,71795,71795,74359,74359,74359,74359,76923,76923,76923,76923,79487,
79487,79487};

//uses mathematical model to map each angle difference 0-126 to a corresponding downwards travel time
int travelTimesDown[127] = {0, 0, 0,2597,2597,2597,2597,5195,5195,5195,5195,7792,7792,7792,7792,10390,10390,10390,10390,12987,12987,
12987,12987,15584,15584,15584,15584,18182,18182,18182,18182,20779,20779,20779,20779,23377,23377,23377,23377,25974,25974,25974,25974,
28571,28571,28571,28571,31169,31169,31169,31169,33766,33766,33766,33766,36364,36364,36364,36364,38961,38961,38961,38961,41558,41558,
41558,41558,44156,44156,44156,44156,46752,46753,46753,46753,49351,49351,49351,49351,51948,51948,51948,51948,54545,54545,54545,
54545,57143,57143,57143,57143,59740,59740,59740,59740,62338,62338,62338,62338,64935,64935,64935,64935,67532,67532,67532,67532,70130,
70130,70130,70130,72727,72727,72727,72727,75325,75325,75325,75325,77922,77922,77922,77922,80519,80519,80519};


void setup() {

servo1.attach(8); // attached motor to digital pin 8
servo1.write(0); // sets up the motor to angle of 0 degrees

pinMode (in1,OUTPUT); //set h-bridge direction pin to output type
pinMode (in2,OUTPUT); //set h-bridge direction pin to output type
digitalWrite(in1,LOW);digitalWrite(in2,LOW);
Serial.begin(9600); //starts serial

}

void loop() {//this is the main function which the controller will run continuously

delay (15000); //intial delay to aid with setup

//this block will scan the input, find the maximum angle, and send the input sensor to that angle
scan();
newPosition = findMax();
servo1.write(newPosition);

//move the output to the corresponding angle
moveOutput(currentPosition,newPosition);

delay (10000); //delay until next sampling time (in real life 30 mins)
currentPosition = newPosition;//set the current position to the newly moved position

}


//the scan() function scans through the range of motion and stores the input sensor values
int scan (){
  int sampleCount = 0; //reset counting number of samples
  //int samples[numberOfScans];//create array to store scanned values
  servo1.write(0); //reset servo position

  while (sampleCount<numberOfScans){
    delay (400); //small delay to let voltage stabilise at each position
    sensorValues[sampleCount] = analogRead(in4); //reads in the analog input at each position, stores to array
    sampleCount = sampleCount+1; //increments the number of samples taken for loop condition
    servo1.write(sampleCount*scanAngle); //write the new sampling position
  }

}

//the findMax() function finds the angle with the maximum light flux input from the sensor
int findMax (){

  int maxCheck = 0; //comparison variable to find maximum
  int maxCheckPosition = 0; //stores maximum position in array

  //this for loop will find the maximum stored input value from the input sensor, and find the position
  for (int i=0;i<numberOfScans;i++) { //scans through for the length of the stored array
    if (sensorValues[i] > maxCheck){
      maxCheck = sensorValues[i]; //sets the new maximum as the largest value
      maxCheckPosition = i; //stores the position of the maximum value
    }
  }

  return scanAngle*maxCheckPosition; //returns the maximum position in degrees
}


int moveOutput (int currentPos, int newPos){

    int travelTime;

    if (newPos>currentPos){//in the case of the new position below the old position (greater angle), move down
          travelTime = travelTimesDown[newPos-currentPos];
          digitalWrite(in1,LOW);digitalWrite(in2,HIGH); // moves shaft motor counterclockwise (in - red out1, black out2)
          delay (travelTime);
          digitalWrite(in1,HIGH);digitalWrite(in2,LOW);
          delay (50);
          digitalWrite(in1,LOW);digitalWrite(in2,LOW);//stop movement
    }
    else {
          travelTime = travelTimesUp[currentPos-newPos]; //in the case of the old position below the new position, move up
          digitalWrite(in1,HIGH);digitalWrite(in2,LOW); // moves shaft motor clockwise (out - red out1, black out2)
          delay (travelTime);
          digitalWrite(in1,LOW);digitalWrite(in2,HIGH);
          delay (50);
          digitalWrite(in1,LOW);digitalWrite(in2,LOW); //stop movement
    }
    digitalWrite(in1,LOW);digitalWrite(in2,LOW); //stop movement

}
```

Figure DC-7: First Full Final System Test Code

TM1637Test

```
#include <Arduino.h>
#include <TM1637Display.h>

// Module connection pins (Digital Pins)
#define CLK 2
#define DIO 3

// The amount of time (in milliseconds) between tests
#define TEST_DELAY   2000

TM1637Display display(CLK, DIO);

int sensorReading=205;
int inputVoltage = map(sensorReading, 0, 1023, 0, 5000);
uint8_t data[] = { 0xff, 0xff, 0xff, 0xff };

void setup()
{
    display.setSegments(data);

}


void loop()
{

  display.showNumberDec(inputVoltage, false); // Expect: _301
  delay(TEST_DELAY);
  while(1);

}
```

Figure DC-8: LED Display Test Code

Gantt Charts

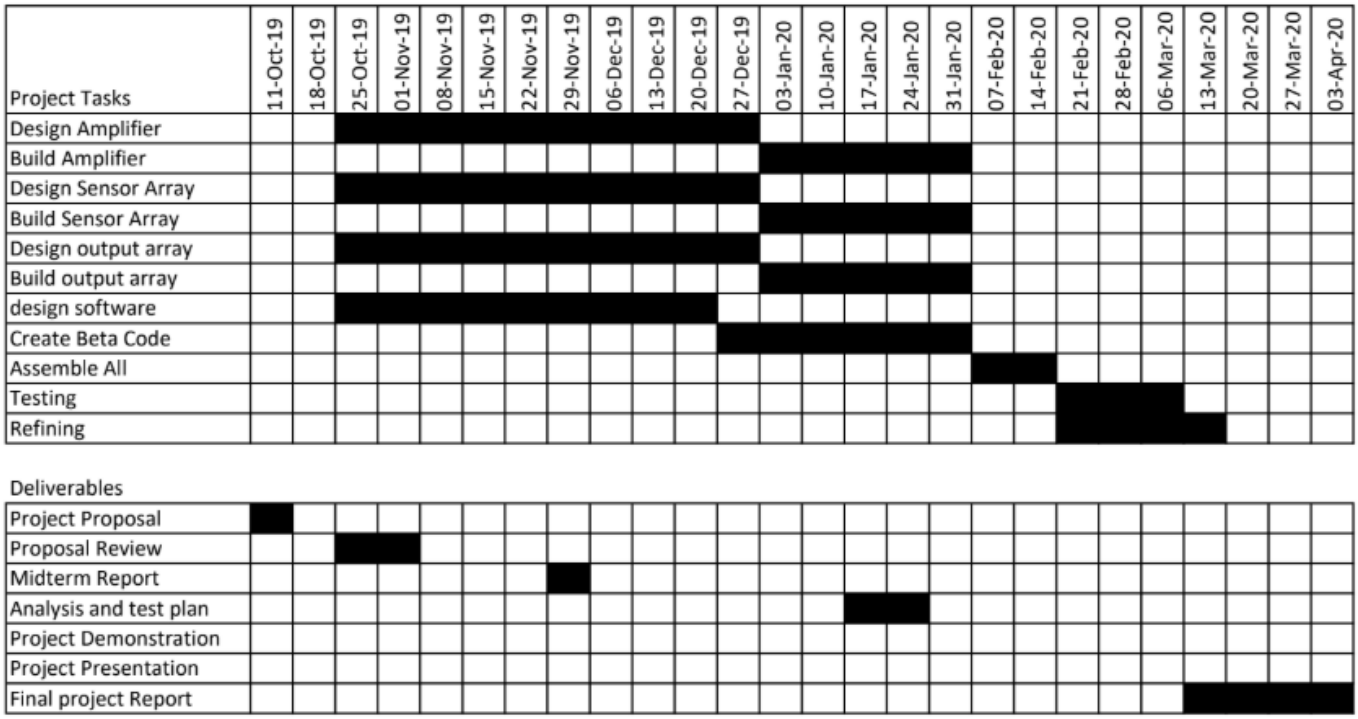


Figure G-1: Final March 2020 Gantt Chart

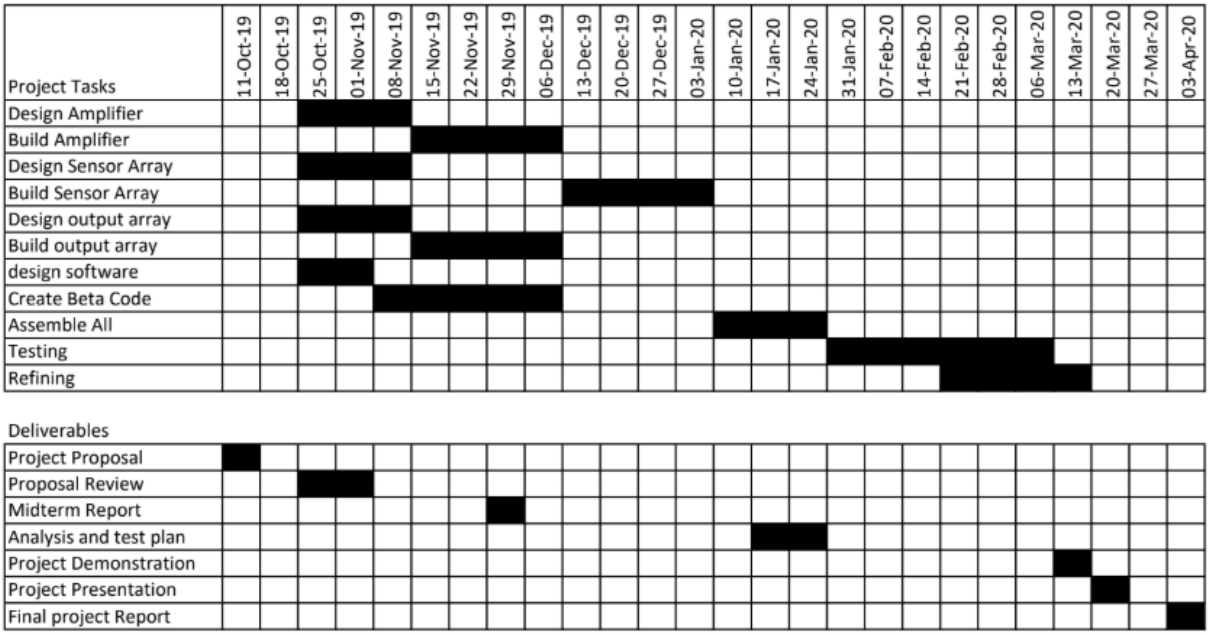


Figure G-2: Old February 2020 Gantt Chart

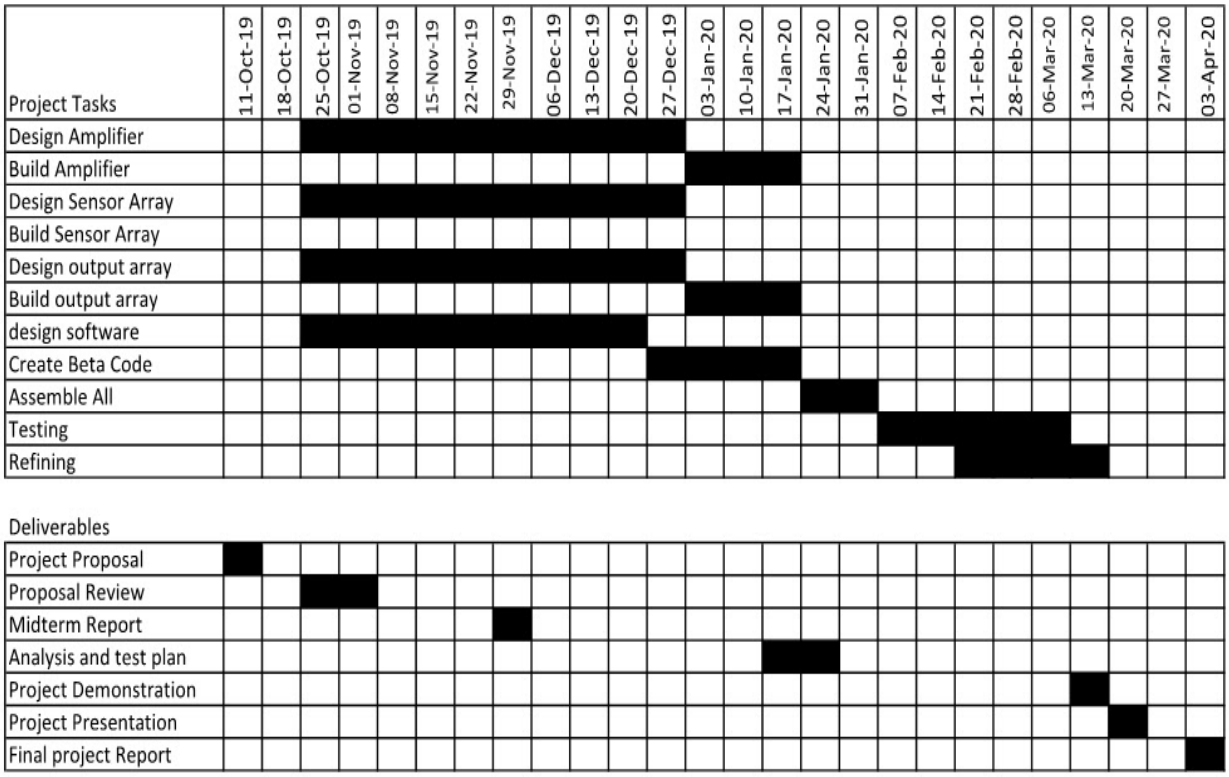


Figure G-3: Old Midterm Progress Report (December 2019) Gantt Chart

Product Data Sheets

Futaba S148 - Servo Standard Precision

Specifications

Modulation:	Analog
Torque:	**4.8V:** 33.30 oz-in (2.40 kg-cm) **6.0V:** 41.70 oz-in (3.00 kg-cm)
Speed:	**4.8V:** 0.28 sec/60° **6.0V:** 0.22 sec/60°
Weight:	1.57 oz (44.4 g)
Dimensions:	**Length:** 1.57 in (39.9 mm) **Width:** 0.79 in (20.1 mm) **Height:** 1.42 in (36.1 mm)
Motor Type:	3-pole
Gear Type:	Plastic
Rotation/Support:	Bushing
Rotational Range:	180°
Pulse Cycle:	(add)
Pulse Width:	550-2330 µs
Connector Type:	J

Brand:	Futaba
Product Number:	FUTM0710
Typical Price:	14.99 USD
Compare:	add+

Figure PD-1: Futaba FP-148 Servo Motor (Input Sensor System Motor)

HuaNing MOTOR

24BYJ48
28BYJ48 type stepping motor
30BYJ46

24BYJ48/28BYJ48/30BYJ46

Applicatim:
mainly used
for air-conditioners...

24BYJ48 Motor Specification

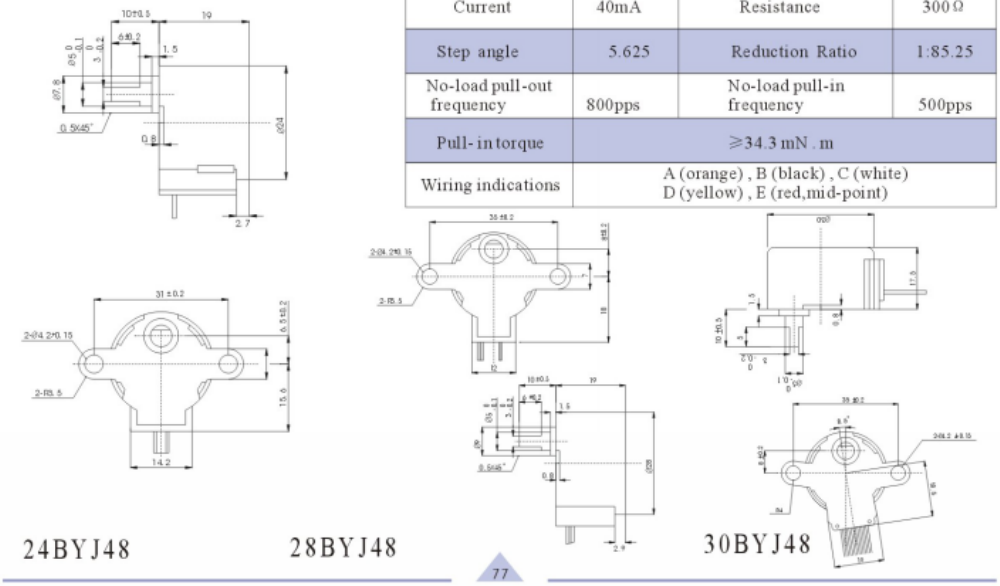
No.of phases	4	Voltage	12VDC
Current	40mA	Resistance	300Ω
Step angle	5.625	Reduction Ratio	1:64
No-load pull-out frequency	1000pps	No-load pull-in frequency	500pps
Pull-in torque	⩾29.4 mN . m		
Wiring indications	A (orange) , B (yellow) , C (blue) D (grey) , E (red,min-point)		

28BYJ48 Motor Specification

No.f phases	4	Voltage	12VDC
Current	40mA	Resistance	300Ω
Step angle	5.625	Reduction Ratio	1:64
No-load pull-out frequency	1000pps	No-load pull-in frequency	500pps
Pull-in torque	⩾29.4 mN . m		
Wiring indications	A (orange) , B (yellow) , C (blue) D (grey) , E (red,min-point)		

30BYJ46 Motor Specification

No.f phases	4	Voltage	12VDC
Current	40mA	Resistance	300Ω
Step angle	5.625	Reduction Ratio	1:85.25
No-load pull-out frequency	800pps	No-load pull-in frequency	500pps
Pull-in torque	⩾34.3 mN . m		
Wiring indications	A (orange) , B (black) , C (white) D (yellow) , E (red,mid-point)		

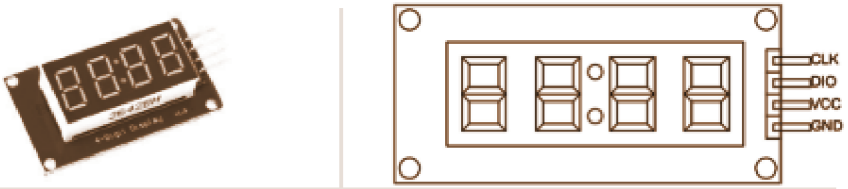


24BYJ48 28BYJ48 30BYJ48

77

Figure PD-2: HuaNing 24BYJ48 Stepper Motor (Preliminary Input Sensor Motor)

TM1637- Grove 4 Digit Display Module



TM1637- Grove 4 Digit Display Module | TM1637 Display Module Pinout

[Click the image to enlarge it]

TM1637 DISPLAY module is used for displaying numbers. The module consists of four 7- segment displays working together. The module working is based on '**TM1637' IC** present internally and hence the name '**TM1637 display**'.

TM1637 Pin Configuration

TM1637 DISPLAY is a four terminal device. We will describe function of each pin below.

Pin Name	Description
VCC	Connected to power source.
GND	Connected to ground.
DIO	Data Input/output pin
CLK	Clock pin

TM1637 Display Features and Specifications

- Two wire interface
- Eight adjustable luminance levels
- Grove compatible interface (3.3V/5V)
- Four alpha-numeric digits
- Potable size
- Operating voltage: 3.3V – 5.5V
- Operating current consumption: 80mA
- Operating temperature: -10ºC to +80ºC

Figure PD-3: Titan Micro-Electronics TM1637 4-Digit LED Display Module

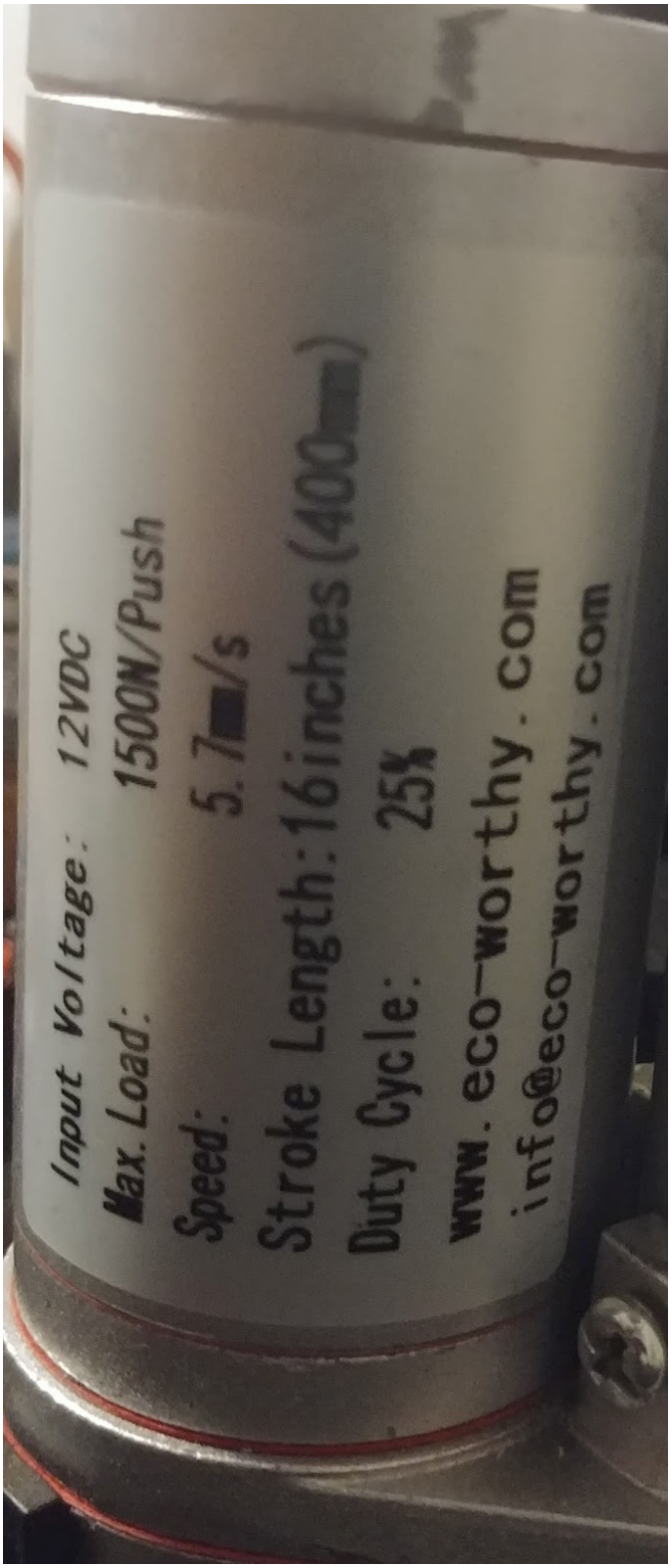


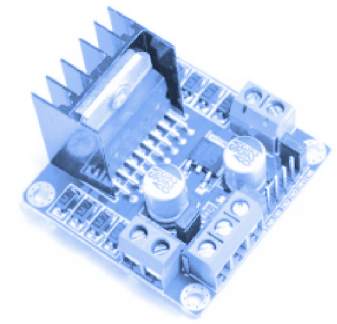
Figure PD-4: Linear Actuator Nameplate Data

Handson Technology

User Guide

L298N Dual H-Bridge Motor Driver

This dual bidirectional motor driver, is based on the very popular L298 Dual H-Bridge Motor Driver Integrated Circuit. The circuit will allow you to easily and independently control two motors of up to 2A each in both directions.It is ideal for robotic applications and well suited for connection to a microcontroller requiring just a couple of control lines per motor. It can also be interfaced with simple manual switches, TTL logic gates, relays, etc. This board equipped with power LED indicators, on-board +5V regulator and protection diodes.



SKU: MDU-1049

Brief Data:

- Input Voltage: 3.2V~40Vdc.
- Driver: L298N Dual H Bridge DC Motor Driver
- Power Supply: DC 5 V - 35 V
- Peak current: 2 Amp
- Operating current range: 0 ~ 36mA
- Control signal input voltage range :
- Low: -0.3V ≤ Vin ≤ 1.5V.
- High: 2.3V ≤ Vin ≤ Vss.
- Enable signal input voltage range :
  - Low: -0.3 ≤ Vin ≤ 1.5V (control signal is invalid).
  - High: 2.3V ≤ Vin ≤ Vss (control signal active).
- Maximum power consumption: 20W (when the temperature T = 75 ℃).
- Storage temperature: -25 ℃ ~ +130 ℃.
- On-board +5V regulated Output supply (supply to controller board i.e. Arduino).
- Size: 3.4cm x 4.3cm x 2.7cm

Figure PD-5a: L298N H-Bridge Relay Circuit Specification Sheet part 1

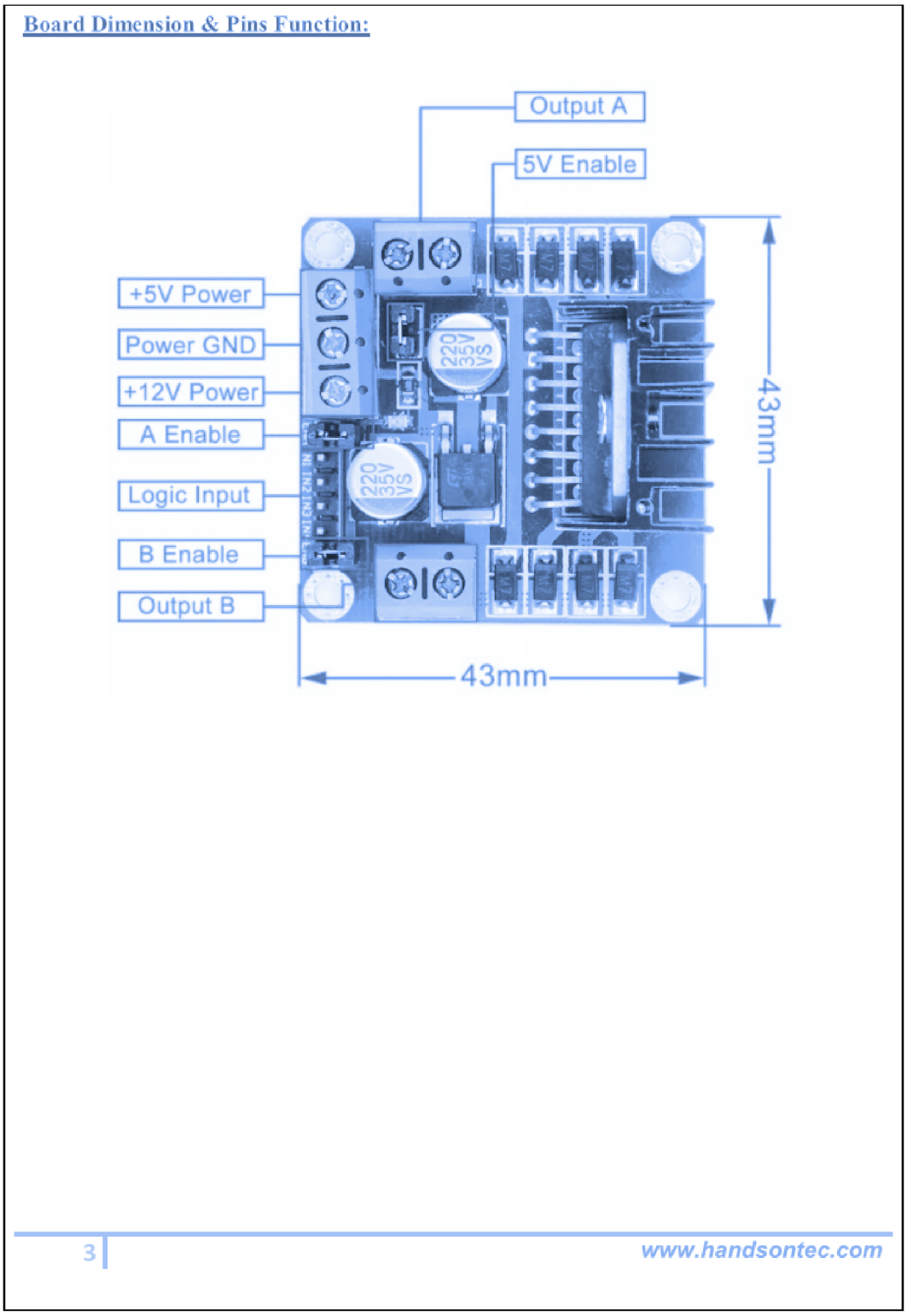


Figure PD-5b: L298N H-Bridge Relay Circuit Specification Sheet part 2

Figure PD-6: Mini Input Solar Panel Nameplate Data

Technical Parameters

Weight	1.8-4.8lbs
Stroke Length	2-18 inches (50-450mm)
Rated Load	330lbs(1500N)
Travel Speed (Max)	0.22in/s(5.7mm/s)
Rated Voltage	12VDC
Starting Current (No-Load)	0.8A
Rated Current	3Amps
Material	Aluminum Alloy
Limit Switches	Fixed Inner (not Adjustable)
Gears	Lubricated Metal Gears
Operation Temperature	-20℃ to +65℃
Protection Class	IP65
Duty Cycle	25%
Noise Level	<=50dB

Figure PD-7a: Linear Actuator Technical Data

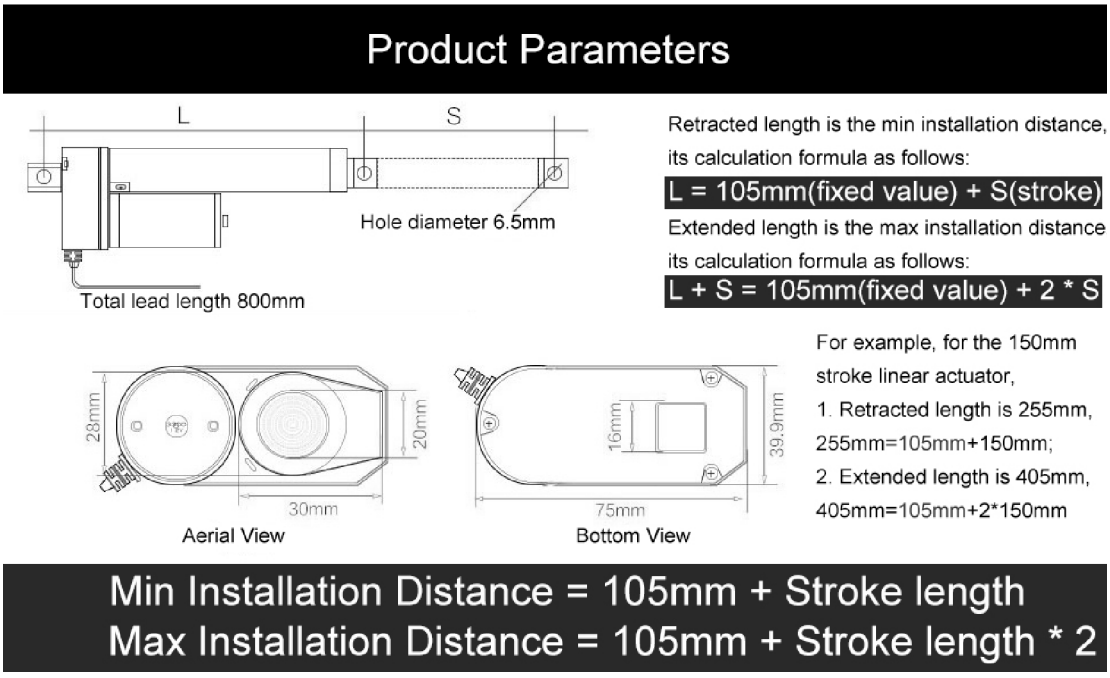


Figure PD-7b: Linear Actuator Construction Specifications

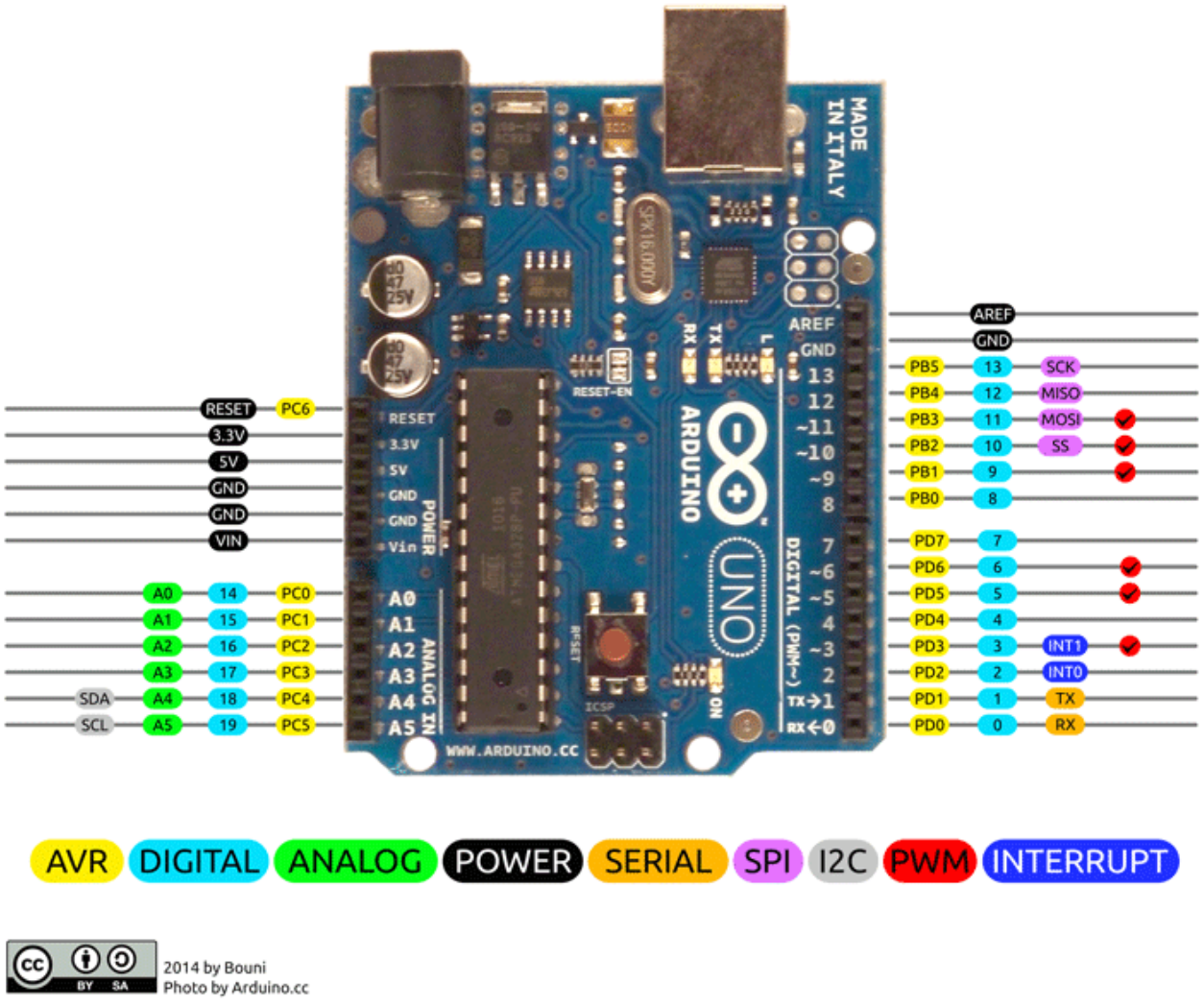
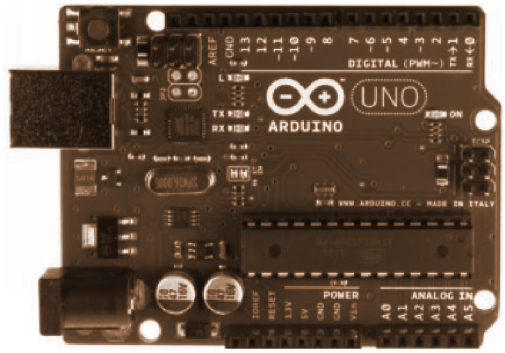
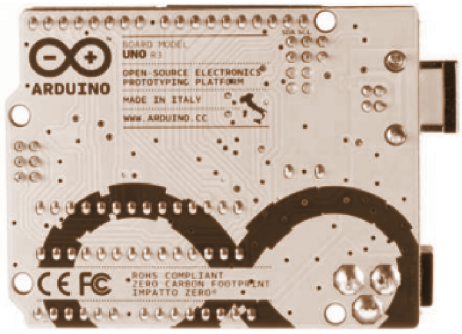


Figure PD-8a: Arduino Uno Pin Connection Diagram

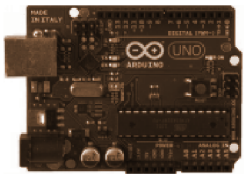
Arduino Uno



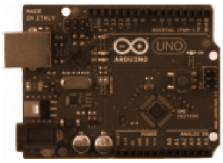
Arduino Uno R3 Front



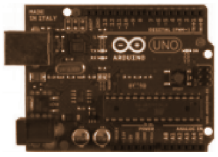
Arduino Uno R3 Back



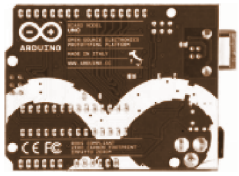
Arduino Uno R2 Front



Arduino Uno SMD



Arduino Uno Front



Arduino Uno Back

Overview

The Arduino Uno is a microcontroller board based on the ATmega328 (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.
The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converter.
Revision 2 of the Uno board has a resistor pulling the 8U2 HWB line to ground, making it easier to put into DFU mode.
Revision 3 of the board has the following new features:

- 1.0 pinout: added SDA and SCL pins that are near to the AREF pin and two other new pins placed near to the RESET pin, the IOREF that allow the shields to adapt to the voltage provided from the board. In future, shields will be compatible both with the board that use the AVR, which operate with 5V and with the Arduino Due that operate with 3.3V. The second one is a not connected pin, that is reserved for future purposes.
- Stronger RESET circuit.
- Atmega 16U2 replace the 8U2.

"Uno" means one in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform; for a comparison with previous versions, see the index of Arduino boards.

Summary

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V

Figure PD-8b: Arduino Uno Specification Sheet Part 1

Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

Schematic & Reference Design

EAGLE files: arduino-uno-Rev3-reference-design.zip (NOTE: works with Eagle 6.0 and newer)
Schematic: arduino-uno-Rev3-schematic.pdf
Note: The Arduino reference design can use an Atmega8, 168, or 328, Current models use an ATmega328, but an Atmega8 is shown in the schematic for reference. The pin configuration is identical on all three processors.

Power

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically.
External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.
The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.
The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.**This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

Memory

The ATmega328 has 32 KB (with 0.5 KB used for the bootloader). It also has 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output

Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- **External Interrupts: 2 and 3.** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 3, 5, 6, 9, 10, and 11.** Provide 8-bit PWM output with the analogWrite() function.

Figure PD-8c: Arduino Uno Specification Sheet Part 2

- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** These pins support SPI communication using the SPI library.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

The Uno has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function. Additionally, some pins have specialized functionality:

- **TWI: A4 or SDA pin and A5 or SCL pin.** Support TWI communication using the Wire library.

There are a couple of other pins on the board:

- **AREF.** Reference voltage for the analog inputs. Used with analogReference().
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

See also the mapping between Arduino pins and ATmega328 ports. The mapping for the Atmega8, 168, and 328 is identical.

Communication

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega16U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '16U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, on Windows, a .inf file is required. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1).
A SoftwareSerial library allows for serial communication on any of the Uno's digital pins.
The ATmega328 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation for details. For SPI communication, use the SPI library.

Programming

The Arduino Uno can be programmed with the Arduino software (download). Select "Arduino Uno from the **Tools > Board** menu (according to the microcontroller on your board). For details, see the reference and tutorials.
The ATmega328 on the Arduino Uno comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).
You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details.
The ATmega16U2 (or 8U2 in the rev1 and rev2 boards) firmware source code is available . The ATmega16U2/8U2 is loaded with a DFU bootloader, which can be activated by:

- On Rev1 boards: connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the 8U2.
- On Rev2 or later boards: there is a resistor that pulling the 8U2/16U2 HWB line to ground, making it easier to put into DFU mode.

You can then use Atmel's FLIP software (Windows) or the DFU programmer (Mac OS X and Linux) to load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU bootloader). See this user-contributed tutorial for more information.

Figure PD-8d: Arduino Uno Specification Sheet Part 3

Budget Spreadsheet

Prototype Component	Item Description	Quantity	Est. Price per Item	Total Item Cost		Notes
Input Sensor Array						
	Small tracking solar panel	1	$ -	$ -		Provided by Supervisor
	Stepper motor	1	$ -	$ -		Included in Arduino Uno Kit
	Resistors	5	$ -	$ -		Included in kit, free from Electronics shop
	Connection Wires	20	$ -	$ -		Included in kit, free from Electronics shop
	Power Supply for Motor	1	$ 10.00	$ 10.00		
	Tracker stand	1	$ 10.00	$ 10.00		
	Tracker solar panel mount	1	$ 5.00	$ 5.00		
	Screws, nails, bolts	10	$ 0.50	$ 5.00		
	Glue	1	$ 2.00	$ 2.00		
	PCB printing	1	$ 10.00	$ 10.00		From Electronics shop
	PCB solder	1	$ 5.00	$ 5.00		
	Shaft bearings	5	$ 1.00	$ 5.00		
	Photoresistors	5	$ -	$ -		Included in kit, free from Electronics shop
	Testing Breadboard	1	$ 6.00	$ 6.00		
	Transistors	5	$ -	$ -		Included in kit, free from Electronics shop

	2020 Aluminum Extrusion	1	$ -	$ -		Provided by a design team
	2020 Aluminum Extrusion Fittings Kit	1	$ 33.00	$ 33.00		
	Shaft Coupling	1	$ 6.00	$ 6.00		
	LED Display	1	$ 10.00	$ 10.00		
Total					$ 107.00	
Control Unit						
	Arduino Uno	1	$ 40.00	$ 40.00		
	Arduino Uno Kit	1	$ 40.00	$ 40.00		
	Connection Wires	10	$ -	$ -		Included in Arduino Uno Kit
	Testing Breadboard	1	$ -	$ -		Included in Arduino Uno Kit
			$ -	$ -		
			$ -	$ -		
Total					$ 80.00	
Output Actuator System						
	Solar panel mount	1	$ 15.00	$ 15.00		
	H-Bridge Motor controller	1	$ 8.00	$ 8.00		
	Solar panel stand material	1	$ 15.00	$ 15.00		
	Stepper M	1	$ 15.00	$ 15.00		
	Power Supply for motor	1	$ 10.00	$ 10.00		

	Transistors	5	$ -	$ -		Included in Arduino kit, free from Electronics shop
	Testing Breadboard	1	$ 6.00	$ 6.00		
	Shaft bearings	5	$ 1.00	$ 5.00		
	Screws, nails, bolts	10	$ 0.50	$ 5.00		
	Solar panel mount material	1	$ 5.00	$ 5.00		
				$ -		
	Linear Actuator	1	$ 70.00	$ 70.00		
	Chair Framing	1	$ -	$ -		salvaged
				$ -		
				$ -		
Total					$ 154.00	
Testing System						
	Connection Wires	15	$ -	$ -		Included in Arduino kit, free from Electronics shop
	Wattmeter	1	$ -	$ -		Already owned and supplied by group member
	Testing Breadboard	1	$ 6.00	$ 6.00		
Total					$ 6.00	
Project Total					$ 347.00	
Budget				$ 300.00		
On Budget?				**NO**		