**N**ottingham **T**rent **U**niversity

**S**chool of **S**cience and **T**echnology

The Development and Testing of a Solar Tracking, Measurement and Logging System

By

Arron Moore

**2016**

Project report submitted in partial fulfilment of the requirements of Nottingham Trent University for the degree of MSc Computer Science

# Abstract

The project looks at the development of a system to enable solar panels to track the sun using modern technology that is readily available in 2016 such as the Raspberry Pi. The tracking is done through both a mathematical approach using equations and a closed loop “scanning” method. Once developed the investigation looks into seeing how these different types of tracking perform in a residential area, and whether this conforms to existing research which paints dual axis trackers to be most optimal type of tracking. It was found that the dual axis tracker is still the most optimal type however modifications may be needed the sun rise or sunset is partially obscured. It was also made clear there is no best solution as it heavily dependent on the environment in which the panels are set up.

# Contents

[Abstract ii](#_Toc461173890)

[1 Introduction 1](#_Toc461173891)

[2 Literature Survey 3](#_Toc461173892)

[3 System Requirements 13](#_Toc461173893)

[3.1 Functional Requirements 13](#_Toc461173894)

[3.1.1 Tracking 13](#_Toc461173895)

[3.1.2 Measurements and Logging 13](#_Toc461173896)

[3.2 Non Functional Requirements 14](#_Toc461173897)

[3.2.1 Tracking 14](#_Toc461173898)

[3.2.2 Measurements and Logging 14](#_Toc461173899)

[4 System Design 15](#_Toc461173900)

[4.1 Introduction 15](#_Toc461173901)

[4.1.1 Modules 15](#_Toc461173902)

[4.1.2 Equipment Needed 17](#_Toc461173903)

[4.1.3 Top Level Diagram 18](#_Toc461173904)

[4.2 Solar Panel Module 19](#_Toc461173905)

[4.2.1 Tracking Systems 19](#_Toc461173906)

[4.2.2 Motor Controller 20](#_Toc461173907)

[4.2.3 Motors 20](#_Toc461173908)

[4.2.4 Solar Panel Class 20](#_Toc461173909)

[4.2.5 Panel Information Server 21](#_Toc461173910)

[4.2.6 Panel Information txt 21](#_Toc461173911)

[4.2.7 MCP3208 22](#_Toc461173912)

[4.2.8 Voltmeter 23](#_Toc461173913)

[4.3 Data Server Module 25](#_Toc461173914)

[4.3.1 Introduction 25](#_Toc461173915)

[4.3.2 Requesting Measurements 25](#_Toc461173916)

[4.3.3 Database Class 25](#_Toc461173917)

[4.3.4 Data Requests 26](#_Toc461173918)

[4.4 Web Server Module 26](#_Toc461173919)

[4.4.1 Introduction 26](#_Toc461173920)

[4.4.2 Getting Data 27](#_Toc461173921)

[4.4.3 Live Data 27](#_Toc461173922)

[4.5 Arduino UNO vs. Raspberry Pi 28](#_Toc461173923)

[4.5.1 Arduino UNO 28](#_Toc461173924)

[4.5.2 Raspberry Pi 29](#_Toc461173925)

[4.5.3 Conclusion 30](#_Toc461173926)

[4.6 Programming Language 31](#_Toc461173927)

[4.6.1 Java 31](#_Toc461173928)

[4.6.2 C++ 32](#_Toc461173929)

[4.6.3 Python 32](#_Toc461173930)

[5 Development 34](#_Toc461173931)

[5.1 Solar Panel Module 34](#_Toc461173932)

[5.1.1 Equations 34](#_Toc461173933)

[5.1.2 Tracking 44](#_Toc461173934)

[5.1.3 Voltmeter 49](#_Toc461173935)

[5.1.4 Panel Information Server 51](#_Toc461173936)

[5.2 Web Server 54](#_Toc461173937)

[5.2.1 Web Server Beginnings 54](#_Toc461173938)

[5.2.2 Home Page Design 54](#_Toc461173939)

[5.2.3 Live data 54](#_Toc461173940)

[5.3 Data Server 56](#_Toc461173941)

[5.3.1 Setup 56](#_Toc461173942)

[5.3.2 Requesting Data 56](#_Toc461173943)

[5.3.3 Responding to Requests for data 57](#_Toc461173944)

[5.3.4 Data Persistence 57](#_Toc461173945)

[6 Experiments and Results 59](#_Toc461173946)

[6.1 Dual Axis Tracking vs. Single Axis Tracking vs. Fixed Attempt One 59](#_Toc461173947)

[6.1.1 The Test 59](#_Toc461173948)

[6.1.2 Problems 59](#_Toc461173949)

[6.1.3 System Evaluation 60](#_Toc461173950)

[6.1.4 Results 60](#_Toc461173951)

[6.1.5 Conclusion 63](#_Toc461173952)

[6.2 Dual Axis Tracking vs. Single Axis Tracking vs. Fixed Attempt Two 64](#_Toc461173953)

[6.2.1 The Test 64](#_Toc461173954)

[6.2.2 Problems 64](#_Toc461173955)

[6.2.3 System Evaluation 65](#_Toc461173956)

[6.2.4 Results 65](#_Toc461173957)

[6.2.5 Conclusion 67](#_Toc461173958)

[6.3 Fixed vs. Single Axis Scanner vs. Dual Axis Scanner 68](#_Toc461173959)

[6.3.1 The Test 68](#_Toc461173960)

[6.3.2 Problems 68](#_Toc461173961)

[6.3.3 System Evaluation 68](#_Toc461173962)

[6.3.4 Results 69](#_Toc461173963)

[6.3.5 Conclusion 72](#_Toc461173964)

[6.4 Fixed vs. Dual Axis Tracker vs. Dual Axis Scanner 73](#_Toc461173965)

[6.4.1 The Test 73](#_Toc461173966)

[6.4.2 Problems 73](#_Toc461173967)

[6.4.3 System Evaluation 73](#_Toc461173968)

[6.4.4 Results 73](#_Toc461173969)

[6.4.5 Conclusion 75](#_Toc461173970)

[7 Conclusion 77](#_Toc461173971)

[8 References 82](#_Toc461173972)

# List of Figures

[Figure 1: Dual Axis Tracker 9](file:///C:\Users\moorey\Desktop\MscPro\Arron_Moore_N0441947.docx#_Toc459997921)

[Figure 2: Mamlook et al, Comparison between fixed and tracked solar system (21) 10](#_Toc459997922)

[Figure 3: Top Level Diagram showing systems and data flow 17](file:///C:\Users\moorey\Desktop\MscPro\Arron_Moore_N0441947.docx#_Toc459997923)

[Figure 4: SG90 Micro Servo 19](file:///C:\Users\moorey\Desktop\MscPro\Arron_Moore_N0441947.docx#_Toc459997924)

[Figure 5: MCP3208 Pin Layout 20](file:///C:\Users\moorey\Desktop\MscPro\Arron_Moore_N0441947.docx#_Toc459997925)

[Figure 8: Servo Circuit 28](file:///C:\Users\moorey\Desktop\MscPro\Arron_Moore_N0441947.docx#_Toc459997926)

[Figure 9: MCP3008 Voltmeter Circuit 29](file:///C:\Users\moorey\Desktop\MscPro\Arron_Moore_N0441947.docx#_Toc459997927)

# Introduction

In 2015 statistics released by the Department of Energy and Climate Change in the United Kingdom tell us that 29.5 per cent the countries energy was generated using gas and 22.6 per cent was generated using coal. 24.7 per cent was generated through renewable sources the most notable being wind and solar energy. (1). Overall this means 52.1 per cent of the energy produced came from non-renewable sources; sources that will eventually no longer exists. Sources that also release carbon dioxide when combusted that will contribute to the greenhouse effect and global warming. Coal and gas together are responsible for more than 90 per cent of the planets carbon dioxide emissions (2). Carbon dioxide being one of the main causes of global warming. It is therefore important that research and investigations are undertaken to develop alternate methods to produce energy and techniques to make those methods more efficient and available to the general public. One alternate method are solar or photovoltaic panels, and one such technique is a solar tracking system, that is the main focus of this project.

The project has two key aims. The first aim is to investigate how different types of solar tracking work and how they compare to one another when used in a residential environment. Almost all research involving tracking systems has been performed in a best case scenario regarding the location, for example high up on buildings or in wide open fields. Whilst this information is useful not everyone has access to such areas for their solar panels. As the technology power behind solar panels becomes cheaper and more accessible to the general public. I am of the opinion that research into looking how tracking systems perform in a residential area will become more relevant.

The second aim is to investigate whether a low cost tracking system can be constructed using modern household technology. Most research was performed ten to fifteen years ago (early 2000s) therefore technology naturally has progressed a lot more, particularly in the areas of low cost small computers capable of producing embedded applications. Furthermore currently most commercial tracking systems can cost thousands of pounds, if tracking systems are to become available to the general public the price of such a system will need to reduce dramatically.

The benefit of developing a logging and measurement system is that it will allow for a very large amount of data to be collected with minimal effort on the users part. Whereas in most research the researchers have taken readings manually every hour or every half hour, a specialised measurement system could take a reading every second. This will give a much larger data set to work with and with so much data any anomalous results can be easily removed.

The success of this project will deemed on the following qualities.

Does it work?

Does it conform to existing research? If not what has been found?

What do the tracking systems add to a solar panel system in a residential area?

Can it be considered low cost? Can it be scaled up?

Has anything of value been discovered?

What improvements could have been made?

# Literature Survey

Solar energy is the heat and light that we receive from the sun every day that is harnessed by plants for photosynthesis and by humans for heating water or generating electricity. The energy we receive is called solar irradiance or insolation; this is a measure of the power per unit of area generated by the sun. Outside of the Earth’s atmosphere the energy density is around 1353 W/m2, this is known as the solar constant (3), though not a true constant as the value varies depending on the time of the year. The actual solar insolation on the Earth’s surface is much lower due to energy being lost by being absorbed and reflected when traveling through the atmosphere. It is lowered even more so as the constant assumes 24 hours of direct light, which does not happen on the surface (3). The solar insolation can be used to calculate the electricity generated by an array of photovoltaic cells.

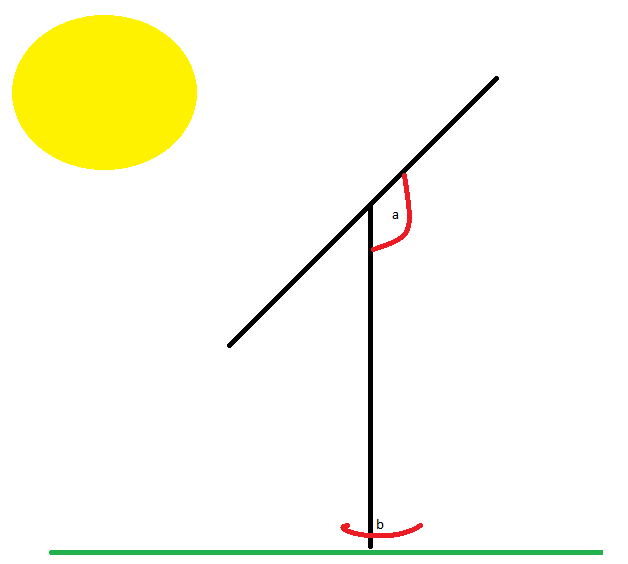
Photovoltaic cells work by being exposed to the thermal radiation emitted by the sun and converting the thermal energy into electricity. (4). It has many advantages, the key advantage being photovoltaic cells provide a clean, renewable source of energy that does not create any pollution whilst in operation. Though it is not without disadvantages as they are expensive to produce and the energy generated is dependent on external environmental factors such as the weather, in particular clouds that may obscure the sun. (4). The first practical photovoltaic cell (also known as a solar cell) was built in 1954 using crystalline silicon which is still to this day the most commonly produced solar cell due to its long life and the ease of mass production (5). There are then two types of crystalline silicon solar cell: monocrystalline and polycrystalline. Monocrystalline silicon solar cells are typically more expensive to purchase but are more efficient whereas polycrystalline silicon cells are cheaper but less efficient (5).

There are many factors that will affect the total energy a photovoltaic cell will generate. One such factor is the total solar insolence at any given time. As discussed earlier the solar insolence can vary depending on the weather conditions, time of year and the detector’s latitude/longitude. The energy generated also depends on the physical properties of the photovoltaic panel, for example its surface area, the direction that the panel is oriented, the tilt angle and the materials it is made of (polycrystalline versus monocrystalline).

In their article “The effect of weather conditions on the efficiency of PV panels in the southeast of UK” Ghazi and Ip look at how the weather affects the power generated by a photovoltaic panel array and in particular the build-up of dust on a panels glass cover. They carried out tests indoors “under controlled test conditions” with a temperature of 25oC and a solar irradiance of 1000W/m2 which they acknowledge will vary “depending on the locality”. They measured the weight of dust deposited and the transmittance(transmittance is the effectiveness of transferring radiant energy) and found the “maximum amount of transmittance reduction is about 5%” with their indoors tests. They also monitored the weathers effect on outdoor monocrystalline photovoltaic panels by selectively cleaning the panels. They found that rain and bird droppings contributed significantly to the dirt deposits on the panels and that weather such as high humidity, rain and wind led to poor efficiency with the panels (6). Their study does not appear to make any reference to how the results were collected, whether they developed a computerised system to log the results at certain intervals or manually read readings. It is also does not mention what they used as a light source for their indoor tests.

To maximise the total solar radiation a photovoltaic panel receive its needs to be angled such that the rays of light arrive perpendicular to the panel. This means that more rays of light are absorbed by the panel rather than reflected. For panels located in the northern hemisphere it is advised that the panels face south and at an angle approximate to their latitude (7). In their article “Determination of the optimal tilt angle and orientation for solar photovoltaic arrays” Mehleri et al suggests that there is an optimal tilt angle and orientation for photovoltaic arrays; where tilt angle is the angle of the panel and the orientation is the direction the panel is facing. The article intends to look at static panels rather than tracked panels as the author deems tracking systems as “expensive and are not always applicable”. Ultimately the paper is looking at how the tilt angle of a photovoltaic panel and its orientation can be modified to maximised the solar irradiance on the panel and the further goes on to look at how the variance the power produced can be minimized across a given time period (7). To do this the authors generated several computerised models using techniques such as neural networks and fuzzy logic to predict the solar irradiance on a tilted surface (7). They found that if they tilt angle was changed twice a year for a winter and summer then photovoltaic power generation increased by 3.5% and not only improves the performance but the “uniformity of the power output” (7).

The alternate to a fixed solar panel as discussed by Mehleri et al is a tracked panel. The first tracked system was developed in 1962 by Finster (8) and was of a purely mechanical design. There panels follow the movement of the sun in the sky so that the they are always pointing towards it. In their article “Determining optimum tilt angles and orientations of photovoltaic panels in Sanliurfa, Turkey” Kacira et al look at the power generated between two 120W single crystalline photovoltaic panels, one of which is at a fixed tilt angle of 14o, facing south and the second is a two axis tracked panel that will follow the sun throughout the day (9). They measure the current, and voltage readings of the panels along with environmental factors such as the air temperature, wind speed, temperature of the panels and the angle of the tracked and fixed panel. The measurements were carried out every hour between 6am and 6pm; the article does not mention whether they did this manually or through a computerised system. The authors found that a two axis tracked photovoltaic panel generated 34.6% more power than its fixed counterpart, though it is noted that this is just the result of one days measurement and the total benefit will vary depending on the time of the year. (9)

In a separate experiment Kacira et al also changed the tilt angle of a fixed panel from 0o to 60o at solar noon (the point at which the sun is at its highest point in the sky). Like Mehleri et al they found that lower tilt angles produced better results in summer and higher angles produced more electricity in winter. Kacira et al also noted that it is a better to alter the tilt angle of the panel per season and that doing it every day or every month is not an optimal option. (9)

As Kacria et al found a tracked photovoltaic panel has the potential to produce more power than its fixed alternative. However there are two types of tracked panel. Kacria et al used a two axis panel, these tracking systems can change both the orientation and tilt angle of the panel. Figure 1 demonstrate a dual axis tracker whereby angle a is modifiable along with angle b. The alternative is a single axis system that just changes the tilt angle, this would mean with regards to figure 1 only angle a can be modified. Li et al in their article “Optical Performance of Vertical Single-axis Tracked Solar Panels” note that dual axis tracked panels perform better though the complicated tracking mechanism means that a single axis system is often “technically and economically more attractive” (10).

Figure 1: Dual Axis Tracker

In their article Li et al looked at comparing the annual collected solar radiation between single axis tracked photovoltaic panels, full two axis tracked panels and fixed panels. Using some complicated mathematical equations to estimate the solar gain of the single axis tracked panels the authors found that single axis tracked panels performed much better than fixed panels, but performed 5-6% worse than dual axis tracked panels (10).

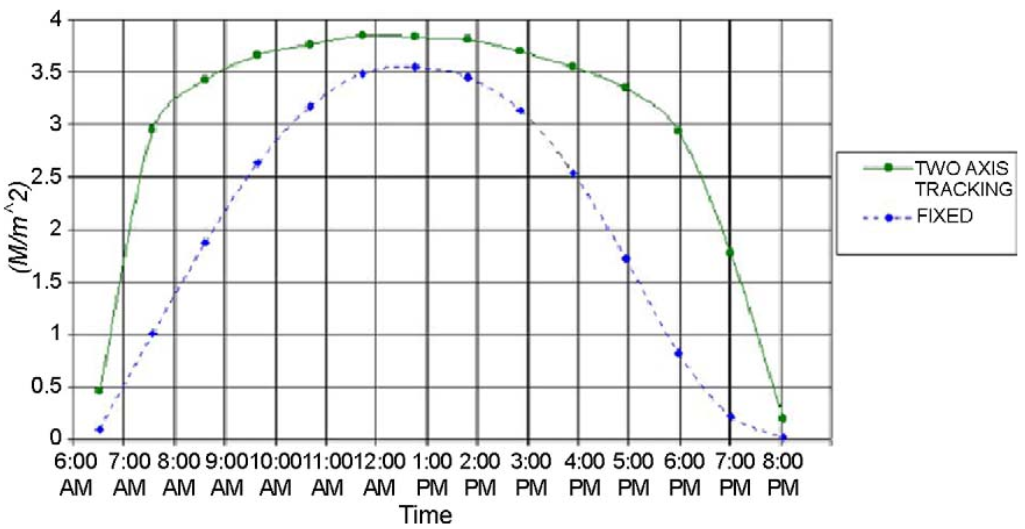
Mamlook et al in their article “A Programmable Logic Controller (PLC) to Control Two Axis Sun Tracking System” looked at designing and created a dual axis open tracked system using programmable logic circuit (11). They used the PLC to periodically to rotate the panel at set intervals in correlation to where the sun would be that was determined using mathematical formulae. This was done as they estimated the motors and control system would use less than 2% of the total collected energy (11). Ultimately they found that their two axis tracking system resulted in a 41.34% increase in daily solar collection compared to a fixed one (11) and they organised the hourly recording into the graph (Figure 2). 

Figure 2Mamlook et al, Comparison between fixed and tracked solar system (21)

Al-Mohammad et al developed a similar system to Mamlook et al, as they used a single axis tracking system that was control by programmable logic circuit(PLC) (12). However it differed in that instead of using mathematics to determine the suns position, Al-Mohammed et al used two photo-resistive sensors and shadows to determine the level of light (12). They found that power generated was greatly increased during the mornings and evenings, with 40 per cent increase in the hours between 0600 and 1000 and between two and four per cent during the day (12).

One manufacturer of single axis tracked photovoltaic systems is a German company called LORENTZ. They were founded in 1993 and produce a wide range of products including solar pumps and solar tracking systems (13). One of their offered solar tracking systems is the “ETATRACK active 2000” a “Single-axis tracking system for PV Modules”. The system supports panels with a surface area of up 20.5m2 and boasts low energy usage and suitability of high wind speeds. LORENTZ also claim that their tracked system offers “up to 40%” more energy than a fixed installation, which given the works of Kacira et al and Li et al seems to be appropriate (14). LORENTZ do not sell their trackers directly but do so through partners. One such partner is an English company called “Wind and Sun”. Excluding VAT the “ETA-2000”; the LORENTZ panel discussed earlier costs £3,214.51 with VAT(20%) this brings the cost to £3857.41 (15). It should be noted that this is the cost for the tracking system and does not include installation for which no quote is given.

In order for a tracked system to be able to point the photovoltaic panel towards the sun, it needs to know where the sun is in the sky. This obviously varies depending on the time of the day, the current season and latitude and longitude of the panel. It also depends on more scientific matters such as the solar declination and the solar hour angle (16).

The current solar declination is a measure of the Earths current tilt. We observe the Earths tilt through seasons but in actuality its measureable and reaches a maximum of 23.45o (16).We know this as the summer solstice, the longest day of the year, alternatively -23.45o is the winter solstice and the shortest day of the year. Thankfully there is already an equation to calculate solar declination as detailed below.

The solar hour angle represent the number of degrees the sun has moved in the sky. The sun moves approximately 15o every hour and a value of 0 indicates the sun is directly above, this is known as solar noon (16). The current solar hour angle can be calculated using the equation below and will be needed later when calculating the solar angle.

This equation then further introduces a new unknown. The solar time, this is usually different from the time we would see on a watch or clock (standard time) but can be calculated with another equation (16). Solar time is different than the time we are familiar with because a day is not actually 24 hours long, it takes the earth 23 hours 56 minutes and 4.1 seconds to complete a rotation on its axis. This is known as a sidereal day (16). In order to convert standard time (clock time) into solar time we use the equation below.

We first need to calculate the longitude time correction though. We need to this because our time is based off of the prime meridian. The prime meridian is the hypothetical line drawn through Greenwich, England and is the basis of Greenwich Mean Time(GMT) though now more formally referred to as Coordinated Universal Time (UTC). Each time zone is then defined every 15o and labelled UTC+1 for one hour ahead or UTC -5 for five hours behind UTC, these are known as standard meridians (16).

The second value we need is the analemma time correction. This takes into account the fact that the Earth does not rotate perfectly on its axis (16) and we cannot guarantee that the sun will be in the same place year after year. The equation to account for this is known as the equation of time(Et). However we first need to calculate a coefficient B that is based on the current day (16).

B is then used in the equation below to give us the analemma time correction.

This finally gives us everything needed to calculate the solar time (16).

We now have everything we need to calculate the solar elevation. This is the angle that the photovoltaic panel will need to be if it is going to be pointing directly at the sun. All we have to do now is enter the results from equations previously discussed into the equation below (16).

The sun has two angles relevant to tracking, the first is the solar elevation which is the angle of the sun relative to the horizon and the second is the solar azimuth. The solar azimuth is measure of how far along the sky the sun is. It is measured horizontally starting with north being 0o. The result of the equation is partially determined by the current hour angle. If the our angle is less than zero, indicating that it is morning then the azimuth angle should be interpreted as a value between zero and 180 degrees. If the hour angle is greater than zero, indicating that it is after noon then the azimuth angle will be between 180 and 360 degrees. The equation for calculating the solar azimuth angle is below. (16)

The equations for each component of this equation have all been explained previously except for the solar zenith angle. The solar zenith angle is the complement to the solar elevation and be calculated by first calculating the solar elevation and subtracting the solar elevation from 180.

If looking to construct a solar tracking system for photovoltaic panels knowing this equation is very important. Although there are websites such as suncalc.net or SunEarthTools.com that will take the various attributes and perform the calculation for you, none of them provide this data in the form of an API so that it is easy to consume from an external source. Doing the calculation allows the tracker to operate without the need for an internet connection and removes the reliance on an external source. It will help to keep the costs of such a system low as if an API is found it will most likely require a paid license, and keeping costs low is very important considering the extremely high cost of existing systems.

Using mathematical equations to calculate the suns position is not the only way to determine where the sun is, and it may not always be the best. Using a closed loop system in which a series of sensors detect light levels in certain circumstances could be a more favourable solution, in particular in the United Kingdom where we tend to have very cloudy days. A closed loop system may be able to detect where there are and are not clouds and point the panel at an area of clear sky rather than a cloudy bit. Roth et al in their article “Design and construction of a system for sun-tracking” looked at exactly this. They constructed a tracking system using closed loop servos that took input from a pyrheliometer, a device that measures solar irradiance (17). They used a computerised system that would detect when the sensor was covered by cloud and could not detect the sun, the system would reposition itself until it could detect the sun again (17). They concluded that after constructing and testing their “low cost sun tracker” that the system worked stably and that the system could be adapted for using with larger photovoltaic panels (17). They performed their study in 2004 which by today’s standards is quite old and though they do not state the technology they used to build they do describe it as cheap. This should mean that constructing a similar system today using 2016 technologies should be even cheaper.

Rubio et al formulate a similar solution in which they take a two stepped approach. In their system they have two defined modes, a normal tracking mode and a search mode (18). In normal mode the system would simply follow solar models in order to predict where the sun would be. However when a set error boundary was met (defined by Rubio et al) in their algorithm the Search mode would begin. In search mode Rubio et al had the system begin a spiral movement until the system could detect the sun (18). Like Roth et al, Rubio et al used a pyrherliometer to detect the solar irradiance. In their results they found that the power generated was 55% than that of a standard open loop system.

The benefits of a closed loop solution are clear and a hybrid solution similar to Rubio et al seems to be optimum however when considering the costs of such a system specifically the reliance on pyrheliometer is problematic as it is specialised and expensive piece of equipment. Another option to detect the light to use another type of electro-optical device and enhance the accuracy through the use of photodiodes mounted on the photovoltaic panel like Heredia et al (19). However Mousazadeh et al characterise these as being complex which makes them expensive and unreliable (20).

To conclude I believe that a single axis tracked system using mathematical formulae to predict the suns location would provide the most cost efficient solution for maximising the solar energy produced however I would like to personally experiment with a closed loop solution. The biggest challenge with a closed loop system being how to detect where the brightest area of sky would be. I think this could be done more simply than measuring the solar irradiance. I think it would be possible to sample areas of the sky with the panel and record the power output, where the power output is highest will most likely be the clearest area of sky. A dual axis system may provide more energy as Li et al found and I agree with their conclusion that the more complicated mechanical aspects of dual axis system would mean constructing and maintaining such a system would offset the slight increase in energy production.

# System Requirements

A system will be developed in order to carry out a series of tests to understand how tracking systems work in a residential area. The system will do both the tracking and take measurements relevant to the current status of the solar panels.

## Functional Requirements

### Tracking

1. The system should feature a mode for tracking the sun using mathematical formulae including the solar elevation and solar azimuth
2. The system should have a tracking mode to scan for the brightest point in the sky.
3. The system should have a mode for setting the panel to a fixed angle.
4. The system should be able to control single and dual axis trackers.
5. The tracking systems should update at set intervals.
6. The tracking should be autonomous requiring no user input and not interrupt the system.
7. The system should be able to control the tracking of multiple solar panels simultaneously.
8. The system should allow the user to specify the type of tracking for a solar panel
9. The system should read user specified information from a text file.

### Measurements and Logging

1. The system should be able to read the voltage being generated by a solar panel
2. The voltage of a solar panel should be saved to a database
3. The current voltage should be displayed to the user.
4. The system should support being able to read multiple voltages from multiple solar panels
5. The measurements should be taken automatically without any user input

## Non Functional Requirements

### Tracking

1. The Scanning mode should be able to scan at least every five minutes
2. The Tracking mode should update at least every five minutes
3. The Tracking program must run on a Raspberry Pi

### Measurements and Logging

1. The system should be able to take a voltage reading at least every second.
2. Measurements should be displayed in a clean and easy to understand way.
3. The database should be saved in an easily accessible place.

# System Design

## Introduction

In this chapter the discussion will focus on the architecture and the main functions of the project and how they will work and why it has been chosen to do that particular way. The design methodology used is Rapid Application Development(RAD). RAD sacrifices planning in favour of developing working prototypes of different parts of the system. These different parts can then be integrated together to complete the system. RAD is an iterative process, when one module of the system is completed it will eventually be revisited or re-done entirely. Whilst this sounds like more work RAD allows for a lot of code re-usability so after the first iteration, code can be copied and pasted and tinkered as necessary, rarely needing an entire re-write. The high modularity of the project makes RAD a good choice. There is a high probability of changing requirements during the project when certain features are too complicated or new features are thought of requiring a change in the design, the code re-usability and modularity inherent to RAD will allow development to continue without much of a disruption.

### Modules

The system will be split into three different modules. The modules will run separately and communicate over the network using TCP. The decision to have the modules separately and networked was chosen because having the modules separately reduces the reliance of each module on one another. This means during development one module can be working without any of the others even being started. The modules also do three inherently different tasks so when thinking about the system it helps to see how the different systems can communicate and prevents any confusing obfuscation in the code. In general however the modularity will help with the portability of the system, each module can be ran anywhere on any computer and they be able to communicate happily without any problems.

The three modules are the Solar Panel Module, the Data Server module and a Web Server module.

The solar panel module(SPM) will read data from varies sensors about the condition of the solar panel, it will also control the tracking of the solar panel. All of the sensor data is stored in a Solar Panel class. The solar panel module will be running on a Raspberry Pi.

Inside the solar panel module is a Panel Information Server, this is a TCP server that will wait for external requests for the data about the panels. This server upon receiving a request will pool all the data from the various solar panels and transmit the data to the Data Server.

The data server is the main storage point for all the data that the SPM generates. It will keep a certain amount of data in memory for quick access but will also be storing each bit of data it receives into an SQLite database to keep permanently.

The web server is a typical web server that will use HTTP to responds to requests and return html. The web server for this system is slightly different however as it will be able to make custom requests to the data server. This will allow for the data being generated by the SPM to be displayed in a web browser on any computer anywhere.

Whenever networking is performed the Transmission Control Protocol(TCP) will be used. The alternative to TCP is User Datagram Protocol(UDP), the reason TCP is being used over UDP is because whilst UDP offers better performance, it isn’t need and the three way handshake performed by TCP will be more useful at preventing errors.

Each module will be developed using the Python programming language, specifically Python3, the reasons for this will be expanded upon later however python was chosen for its cross platform capability and how easy to use the language is.

### Equipment Needed

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Quantity** | **Item** | **Price** | **Purchase Location** | **Extra Info** |
| 1 | Raspberry Pi B+ | £23.70 | eBay  http://www.ebay.co.uk/itm/Raspberry-Pi-Model-B-v1-2-SD-Card-NOT-included-/232044186948?hash=item3606e94d44:g:NRoAAOSw5IJWcsN6 | All ready owned,  Added for approximation |
| 4 | Kitronik 3V Solar Cell | £2.99 Each  £11.96  Total | Maplin  http://www.maplin.co.uk/p/kitronik-3v-solar-cell-100ma-a73rj | One Spare |
| 5 | SG90 Micro Servo | £1.33 Each  £6.95  Total | eBay  <http://www.ebay.co.uk/itm/262548182197?_trksid=p2057872.m2749.l2649&var=561403614875&ssPageName=STRK%3AMEBIDX%3AIT> | One Spare |
| 1 | MCP3208 | £4.65 | eBay  http://www.ebay.co.uk/itm/350851413689?\_trksid=p2057872.m2749.l2649&ssPageName=STRK%3AMEBIDX%3AIT |  |
| 5 | 3xAA Battery Holders | £1.10 Each  £5.49  Total | eBay  http://www.ebay.co.uk/itm/281526522654?\_trksid=p2057872.m2749.l2649&var=580551881056&ssPageName=STRK%3AMEBIDX%3AIT |  |
| 15 | Asda AA Batteries | 23p Each  £5.50 | Asda  https://groceries.asda.com/asda-webstore/landing/home.shtml#/product/910001481958 |  |
|  |  | £58.25 |  |  |

The equipment mentioned will build five single axis trackers or two dual axis trackers and an extra single axis tracker

### C:\Users\moorey\Downloads\Top_Level(3).pngTop Level Diagram

Figure 3: Top Level Diagram showing systems and data flow

## Solar Panel Module

The Solar panel module will control everything about the solar panels, this includes the tracking and the reading of data from the sensors. This module is designed to run on the Raspberry Pi.

### Tracking Systems

The system will feature three types of tracking. The main purpose of the tracking system is keep the solar pointing towards a particular direction.

#### Mathematical Tracking

The first is tracked based on mathematical formula, specifically calculating solar azimuth and the solar elevation . In their experiment Mamlook et al used a technique similar to this, in their report they simply mention “mathematical formulae” not the equation by name, however it can be assumed they’re using the solar azimuth and elevation equations, or a variation/modification of the equations. The system is capable of tracking using a single axis setup whereby just the solar azimuth is used to position the motors and a dual axis mode in which two motors are used, a top one to control the elevation and bottom one to position based on the azimuth.

#### Closed Loop Scanning

The second is a scanning based approach in which the system will actively try to find the brightest light source and point the panel towards it. Rubio et al created hybrid solution using this type of tracking, theirs differed in that they would use formulae up to a certain point and which point a “searching” mode would begin. The design of the system differs from Rubio et al in one key aspect and that is in their report they mention the use of a pyrheliometer, instead of using this specialist and expensive equipment to measure the solar irradiance the system will use the voltage being generated by the solar panel as an indicator to how much light is in a particular spot. The scanning mode also features capability to scan using a single or dual axis set up.

#### Fixed Angle

The final type will keep the panel at a fixed elevation and orientation, the user will specify the angle and when the system starts the position will be set. The user will have to modify the elevation manually, Kacira et al suggested doing this at the change of each season, using a higher angle in the summer (around 60o) and lower in the winter.

These three types of tracking give the system a lot of flexibility and allows for the opportunity to compare and contrast how each type performs and then whether or not that conforms to the results gathered by Mamlook, Rubio and Kacira. The system will also be built to allow the type of tracking to be changed very quickly and easily and allow for multiple tracking systems to initialised and operating at the same time.

### Motor Controller

The motor controller class is an interface to control the motors that position the solar panels. The class will allow the system to specify the GPIO Pin on the Raspberry Pi that it will use to communicate with the motor. This will enable multiple motors to be controller very easily and flexibly. After initialising the class it will be required to enable the pulse width modulation on the pin specified, call a function to set the angle and then stop the pulse width modulation. Pulse width modulation is not an ideal way to control the motors because it is an analogue system. It uses pulses of current to induce duty cycles, if the current is high for a specified amount of time then the motor will move a corresponding amount. The problem with this is that it causes jitter, for the purposes of a prototype system the jitter is acceptable. The solution to this is to use a dedicated servo driver.

### http://www.generalhobby.com/images/products/tower-pro-sg90.jpgMotors

The motors being used are SG90 Micro Servo Motors(Figure 4).The SG90 is rated for 1.8kg/cm of torque at 5V and weigh only 9 grams. The torque produced by these motors is far more than what is required, the solar panels being used only weigh around 20 grams, at most the bottom servo of a dual axis tracker will have to move around 30 grams.

The low cost of the SG90 is also a huge bonus, they can be purchased in packs of 10 for £11.95 from eBay which works out at about £1.20 each.

Figure 4: SG90 Micro Servo

Each SG90 will be powered by three AA battery to produce 4.8V total.

### Solar Panel Class

The solar panel class is designed to encapsulate all the details and control of the solar panels. This means that the solar panel class will control initialise the tracking system and then pass the responsibility of tracking to that system. The class will also be in control of gathering and storing the sensor data produced. A solar panel class will be generated for each solar panel in the system and will be unique to that solar panel. The class will contain the details regarding the channel on the ADC the panel is connected to, the GPIO pins of the motors controlling the panel, the tracking method of the panel and the also provides an interface for this information to retrieved from the class. This class was designed in interest of keeping each panel unique and to allow for the addition or removal of any number of panels from the system.

### Panel Information Server

The panel information server is the main access point to the data gathered. As the name suggests are its core is a TCP server. This TCP server waits for a request from the data server. When a request is received the panel information server will gather data from the various Solar Panels that will be running, then transmit this data back to the data server. The panel information server also acts as start-up sequence, it will read data from Panel\_Information.txt a text file containing the details of the solar panels that will require setting up. It will parse this file and then set up the Solar Panel classes for each record in the file.

### Panel Information txt

This is a simple text file for storing the parameters of a solar panel. It uses a comma separated format, and it use value has a specific meaning. A typical record will be structured like the following,

“Channel, Tracking Mode, GPIO Pin”

There are multiple variation however depending on the tracking mode.

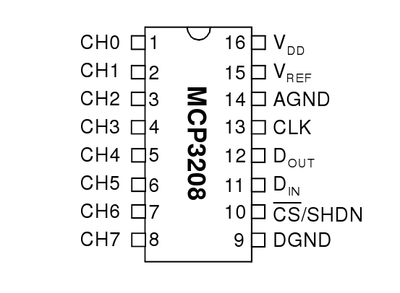
The tracking modes are:-

* SA-T – Single axis tracked
  + channel, tracking\_mode, GPIO\_Pin
  + 0,SA-T,2
* SA-S - Single Axis Scanner
  + channel, tracking mode, gpio pin
  + 0,SA-S,2
* SA-F – Single Axis Fixed
  + channel, tracking mode, gpio pin, angle
  + 1,SA-F,3,45
* DA-T
  + Channel,tracking mode, gpio pin, gpio\_pin
  + 2,DA-T,2,3
* DA-S
  + Channel,tracking\_mode,gpio pin,gpio pin
  + 3,DA-S,2,3

### MCP3208

The MCP3208 is a 12 bit, eight channel analogue to digital converter. Its main purpose will be to convert the analogue voltage generated by the solar panel into a digital signal that the Raspberry Pi will be able to understand.

The MCP3208 was chosen for its resolution, as its 12 bits this means that it will interpret the analogue signal as a value between 0 and 4095, the alternative to the MCP3208 is the MCP3008 which only has a 10 bit resolution giving a value between 0 and 1023. Both ADC’s offer eight channel but the higher resolution of the MCP3208 makes it a better choice. The cost difference between the difference is negligible. The MCP3208 was also partly chosen based on the recommendations of various users on internet forums (21) (22).

The solar panel will connect to any of the “CH” pins

VDD(16) connects to the 5V supply (via raspberry pi connected to a breadboard)

VREF (15) to the 5V supply

AGND (14) to ground on the breadboard

CLK(13) connects to the GPIO pins on the Raspberry Pi

DOUT(12) connects to the GPIO pins on the Raspberry Pi

DIN(11) connects to the GPIO pins on the Raspberry Pi

Figure 5: MCP3208 Pin Layout

CS/SHDN( 10) connects to the GPIO pins on the Raspberry Pi

DGND(9) connects to ground.

The tutorial by Matt at raspberrypi-spy.co.uk helped with the above connections (21).

### Voltmeter

The voltmeter interfaces with the analogue to digital converter(ADC) to measure the voltage being generated by the solar panel. The voltmeter is sent the channel on the ADC to read and contains a function to return the current voltage. To communicate with ADC a python library called spidev will be used. This library provides an interface to Serial Peripheral Interface devices such as the ADC. The ADC being used is the MCP3208

As mentioned the ADC gives a reading between 0 and 4095, this value needs to be converted into a corresponding voltage. The equation for this is outlined below.

The 5 represents 5V, this is what is connected the voltage reference pin on the ADC and the 4095 represents the maximum value the ADC will produce.

**Example 1:**

ADC\_Reading = 4095

Equals 5 Volts.

**Example 2:**

ADC\_Reading = 2645

Equals 3.22 Volts.

## Data Server Module

### Introduction

The data server’s purpose is to request the data from the solar panel module (SPM) and store that data for later viewing. The reason this module exists and is not part of the SPM is because whilst the Raspberry Pi is relatively powerful for its size and price, it’s not suited for applications that have the possibility of using a large amount of memory or storage space. The problem with the Raspberry Pi is that it only has 512 megabytes of RAM and though it is interchangeable, the prototype only has an 8gb micro SD card. Therefore the decision was made to have the data storage on an external machine with better storage capabilities.

### Requesting Measurements

The data server will periodically open a socket connection to the SPM and request a reading. The request does not need to be anything specific as the SPM is only expecting connections from the data server. The response will be formatted like the following

*Time, channel, GPIO pin(s), angle mode, angle, voltage/*

Once the data server has the reading it will store them it into an SQLite database. The code will use the database class described later on to interface with the database and will use the current date as a filename. The date is used so that it is easier to see when each set of readings was taken. SQLite was chosen because of its portability and it is very well supported in Python with its own library. The portability is important makes it will make analysing any data produced easier in the future as the system does not have that functionality built into it.

After the data has been saved into the database it will be saved into memory, whilst saving into the database adds persistence to the data, reading from the database is not as fast as reading from memory so getting real time data will be quicker if its read from main memory rather than the database.

### Database Class

The database class is used as an interface for the SQLite database. It is a very simple class with only the constructor and an insert method. The constructor will read the file name as a parameter and open the database with that filename. If no database exists it will create one. As mentioned before the code will automatically insert the date as the filename.

The insert method takes the reading the data server gets from the SPM as a parameter. This reading is then parsed into its component attributes, the channel, angle and voltage etc and inserted into the database using SQL.

*“CREATE TABLE if not exists readings(time text,channel text, gpio\_pin text, angle\_mode text, angle text, voltage text)”*

Above is the SQL used to create the table the data is stored in and below is the insert sql. The insert sql uses pythons .format function to enter the variables in place of the {}.

*"INSERT INTO readings VALUES ('{}','{}','{}','{}',’{}’,’{}’)".format(time,read\_channel,gpio\_pin,angle\_mode,angle,voltage)”*

### Data Requests

The data server acts as a source for all the data in the system therefore it not only has to request data but responds to request for data. This is the other key part of the data server, a TCP server that responds to requests from the Web Server for data. The data server will wait for a custom request called a GET\_DATA request. This request is formatted in a specific way so that the data server can customise the response for exactly what the end user needs. For example if the last five readings where needed then the web server would send GET\_DATA A 5, the GET\_DATA specifies the request, the A stands for all data (not a particular attribute) and five is how many readings it wants. The response is then formatted into the same string it was received in and returned back to the source connection.

## Web Server Module

### Introduction

The Web Server modules main purpose is to display the data generated by the solar panels to the user. In the interest of time and because a lot of the functionality is not required many aspects of the HTTP standard will not implemented. Instead only the core functionality of the GET request will be implemented. This will allow web pages to be displayed and data to be transferred between web pages. The web server itself will be written in the python programming language, naturally however the web pages will be written in HyperText Mark-up Language(HTML) with some JavaScript. The reason it was decided to create a web server rather than use an existing one such as Apache or Nginx was made because I custom web server allows for more flexibility. Specifically a custom web server allows the developer to use sockets whilst generating the HTML.

### Getting Data

The web server needs to display data contained in the data server. It will do this by opening a socket to the data server and making a GET\_DATA request.

When a user accesses the web page their web browser sends a HTTP GET request to the web server, the web server then responds by sending the appropriate HTTP headers back and the HTML of the web page. The web server module when generating the HTML will send the GET\_DATA request and then format the response into the HTML. When the user receives the HTML, they will be able to review the data in their web browser.

### Live Data

To get live updates from the data server into the users web browser python can no longer be used as the web page is not under the systems control. However JavaScript is as JavaScript code is executed client side, meaning it will be ran on the user’s computer. To get live data a technique within JavaScript will be used known as AJAX. AJAX stands of Asynchronous JavaScript and XML, this will let code be executed asynchronously, ie in the background. AJAX will be used to send HTTP requests to a separate URL hosted by the web server. When this separate URL receives a HTTP request it will open a socket to the data server and request the last reading the data server received and return the response back to the JavaScript. The JavaScript will then parse the response and format the response into a readable manner.

## Arduino UNO vs. Raspberry Pi

Due to the heavy requirement of the system to interact with external sensors and components a device with easily accessible GPIO Pins is necessary. This essentially gives two options, the Arduino UNO or a Raspberry Pi. This section will detail the pros and cons of both and then summarise why the project will be using a Raspberry Pi.

### Arduino UNO

The Arduino UNO is a microcontroller featuring an ATmega328 processor with a CPU speed of 16 MHZ and 32kB of flash memory. It has fourteen digital pins, with six capable of pulse width modulation and 6 analogue inputs (23). Being a microcontroller means that the Arduino is tailored for embedded systems; systems that may have to interface with external sensors and systems that do not need the functionality of an operating system and microprocessor.

Code for the Arduino UNO is written in the Arduino Programming Language which is based on the programming language Wiring which itself uses C/C++ (24). In order to upload code onto the Arduino the Arduino Software IDE is used.

#### Advantages

* Low Power Usage
  + Power usage of device has less impact on the total power generated by the solar panels.
* More than enough GPIO Pins
  + Allows for the addition of more motors, so different configurations can be ran at the same time
* Analogue to Digital Converter Built in
  + Reduces the total cost as a separate ADC does not need to be purchased
* Large User Base and Support Network
  + If any problems occur during development the Arduino has very accessible amount of knowledge to help solve the problem.
* Expansive set of code libraries
  + A large number of libraries speed up development by making it much easier.
* Expandable functionality through addition of shields
  + Certain functionality such as network connectivity can be added by fixing a network shield to the Arduino
* Simple Device
  + At its core the Arduino is very simple device, this greatly reduces the number of possible problems that may occur.

#### Disadvantages

* No native network functionality.
  + The Arduino does not have an Ethernet port. An Ethernet port can be added through a network shield however this increases the total cost of the system.
* Low memory
  + The small amount of memory of the Arduino limits the size of programs to 32256 bytes, this will be a problem with larger programs.
* Low Processing Power
  + The Arduinos low CPU speed on 16MHZ severely limits what it can. It will be able to calculate the mathematical equations fine but it may struggle with more complex networking.

### Raspberry Pi

The Raspberry Pi is a small single board computer featuring an ARM1176 microprocessor with a CPU speed of 700mhz and 512MB of memory. It has forty GPIO pins, four USB ports and an Ethernet port of network connectivity. The Raspberry Pi is capable of running a full Unix based operating system such as Linux and providing a graphical user interface.

#### Advantages

* Powerful
  + The Arduino is a computer capable of everything a computer can do, therefore it will have no problem running the system.
* Operating System
  + The Raspberry Pi is capable of running a full operating system, the most common being Linux based Raspbian, this gives a lot more flexibility on how the program is developed
* More than enough GPIO Pins
  + The same as the Arduino the Pi has a lot of pins.
* Graphical User Interface
  + The Raspberry Pi has an HDMI port, and four USB ports, this means it can be used with a mouse, keyboard and monitor. This will help with development.
* Ethernet Port
  + The Pi has a Ethernet port built in meaning it can be connected to a network. This is vital for the successful operation of the system.

#### Disadvantages

* High Power Usage
  + The Pi uses more power because it is more powerful, the higher power usage will cut into the total power generated by the panels.
* More Complex
  + The Pi is far more complex due to this if something goes wrong in something unrelated to the project, the project or the Pi itself may shutdown.
* No ADC
  + The Pi does not feature an analogue to digital converter, this means one will have to be purchased. This will increase the complexity of the circuit and cost of the project.

### Conclusion

As mentioned the project will be using a Raspberry Pi. The lack of an ADC is a problem but the cost of an ADC is very low, also the possibility of the Pi crashing however Raspbian is a very robust as it is based off the Debian distribution of Linux, Debian has a reputation for being very stable. The higher power usage may be a problem however it should be negligible relative to the total power of a proper full size solar panel.

The biggest reason the Raspberry Pi was chosen is because it allows the flexibility for the project to be developed in almost any of the major programming languages such as C++, Java or Python. With the Arduino, the Arduino Programming Language(APL) has to be used, whilst the APL is a fine language and excellent at what it does, it is not designed for larger projects and with the scale of the project it will become too unwieldy and confusing.

## Programming Language

There are many programming languages available to use to develop the project however it has been narrowed down to the three most suitable.

### Java

Java is programming language developed by Sun Microsystems (now Oracle Corporation), that started development in 1991 and was first released in the form of Netscape Navigator internet browser, in 1995 (25). Java is an open source, object oriented language that runs on millions of devices across the globe. The reason java is so widespread is its biggest selling point, the Java Virtual Machine (JVM).

The JVM allows java code to be executed on any computer, regardless of its architecture or operating system. It does this by converting the Java code into Java byte code, a language similar to assembly (one layer above binary), this is thankfully done automatically and requires no effort by the user. As java is so widespread as a language there is a lot of support for it, in the form of internet forums and text books so if you need help with something it should not be a problem. It also has the bonus that comes with being so popular in that there are many libraries already available, this is particularly useful as the project makes a lot of use of networking, in particular sockets.

The biggest downsides to java are its speed and reasons personal to me. Whilst its speed was considerably slower it has improved over the years, it is still not near a compiled language such as C or C++. My own opinion of the language are the biggest reasons it will not be used, personally I find the syntax too verbose. There is always a fine balance in programming of using function and variable names that are clear and understandable but not too long that they are difficult to type or clutter the code. I believe java takes the verbosity too far and the code becomes cluttered with unnecessary letters making it frustrating to type and difficult to read. The other main personal reason for not wanting to use Java is I have yet to find an Integrated Development Environment(IDE) that I like, I find working on the command line is too time consuming and the likes of Net beans or Eclipse are too slow and cumbersome. These are all of course personal reasons for not like the language even though on paper Java is a great language however when undertaking a project like this it will be important to enjoy developing it and Java will not allow that.

### C++

C++ is the successor to C, the ++ being a joke indicating that C++ is the next increment along from C. The language entered the history books in 1985 when the creator Bjarne Stroustrup published his reference book “The C++ Programming Language” (26). C++ has influenced the creation of many languages such as Java and C#, making it in my opinion one of the most important programming languages. It is a an object oriented (not in the same way as java though) compiled language meaning if you need something performed quickly C++ is a go to language. However here lies the main reason C++ will not be used, as its compiled this means that the executable created is specific to the operating system and architecture that the machine was compiled on. As an example a program compiled one an x86 Windows machine will not run on an ARM powered Linux machine (such as Raspberry Pi). This will make development harder and more time consuming than necessary, the biggest problem being the Raspberry Pi’s weak processor, compiling large amounts of code can take a long time.

### Python

Python is an open source (Python Software Foundation License), cross platform, interpreted programming language created by Guido van Rossum in 1991. Python is also object oriented and has very large standard library including libraries for mathematics, sockets, GPIO and data serialization. The syntax of python is designed around the idea of being readable and it succeeds at this. The readability of python code makes very easy and quick to develop and was programs have been developed they’re just as easy to review and maintain. There is also a great deal of support regarding python and pythons use on a Raspberry Pi. With regards to python itself it hosts a very large website dedicated to the documentation (27) and has the documentation built into the language through the help command. The Raspberry Pi Foundation, the creators of the Raspberry Pi also host their website full of documentation relating to the use of python with a Raspberry Pi (28).

There are a couple of problems with python, for example because it is an interpreted language meaning its executing line by line, it is not as fast as a compiled language such as C++ or even Java. Also because it is interpreted errors are sometimes hidden because the error is only found when the code is executed. Python also does not need the user to specify types for variables, this is great because it saves time however sometimes when a variable is misspelled by accident it will create the new variable and not alert you to your mistake, this leads to some frustrating problems.

Despite its faults python is a fantastic language. Development can be done on a desktop computer running windows, the program can be written and tested in windows then be transferred to a Raspberry Pi running Linux and run perfectly happily. This is along with that python is an enjoyable language to use are the main reasons that it will be the main programming language used to develop this project.

# Development

Development will be done iteratively and in small pieces. When each piece has been created the will be put them together to create a rough version of the main program. If the rough version is still a little too rough it will be refactored until it that it works as intended and is easy maintain.

The reason the project will developed this way is because each piece that is completed gives a sense of progress towards completing the project, even when modules are incomplete the project still feels as if it progressing. This helps the project stay enjoyable and helps stave off any procrastination or stagnation. A mercurial based repository for version control will be used to keep track of changes and to provide a centralised location for the project. The repository is especially useful due to the different modules and the different pieces of those modules being created

This section will not detail all the iterations and prototypes used because there are many and most were scrapped or interacted with systems that are not in the final design. Through the use of Rapid Application Development(RAD) the modularity and code re-use allowed parts of the systems to be scrapped or reworked with very little time loss.

## Solar Panel Module

### Equations

Development began with implementing the equations need to calculate the suns position. For the single axis tracker the solar azimuth is needed and for the dual axis tracker the solar azimuth and solar elevation are needed. The first equation needed was the solar declination

#### Solar Declination

The equation for solar declination is:

Solar declination is one of the easier equations to implement because it only has one unknown variable however calculating the current day of the year is not an easy task. This is the first example of proof that Python was a good choice for the project as Pythons datetime library has a function to do exactly what is needed.

day\_of\_year = datetime.now().timetuple().tm\_yday

The above code returns the current day of the year as an integer. This was written with help from user DzinX on Stack Overflow (27)

The biggest problem encountered was confusion surrounding units, the sin function in python returns an answer in radians but the 23.45 is in degrees, so it was important to remember to convert to radians, this was something that was a constant problem throughout developing the equations. It was made easier as Pythons math library contains a function to convert to degrees to radians and vice versa.

To check the results a website maintained by the National Oceanic and Atmospheric Administration an American Government Agency which calculates the suns position was used to cross reference the results (28).

#### Solar Hour Angle

The equation for the Solar Hour Angle is

This equation has the first unknown value in which another equation is needed, the solar time is explained in a later chapter.

The code is very simple as the solar time can be separated off into its own function, this leaves just two lines.

*solar\_time = self.solar\_time(current\_time,longitude)*

*solar\_hour\_angle = 15 \* (solar\_time - 12)*

The biggest problem encountered was verifying that the angle generated was accurate, the NOAA website used earlier does not provide the solar hour angle for comparison. A new website was found that does provide it however the website appears to be set up by a private individual rather than a government agency and so may not be as accurate. The data this new website does provide however corresponds with some of the NOAA results so it should be accurate enough for the needs of the system (29).

#### Solar Time

The local solar time can be calculated using the equation below.

A simple equation that firsts needs the time correction (TC) and the current time. The current time can be gathered from the datetime library, the only problem being the time needs to be in minutes. Python however lets you use break the time down by hours, minutes, and seconds so the conversion was not hard and can be placed into its own function for easier use.

*current\_time = datetime.now()*

*hour\_to\_minutes = (float(current\_time.hour) \* 60)*

*seconds\_to\_minutes = float(current\_time.second) / 60*

*current\_time\_minutes = hour\_to\_minutes + float(current\_time.minute) + seconds\_to\_minutes*

The time correction is also separated off into its own function so calculate the solar time in code looks like the below

*solar\_time = current\_time\_minutes - self.time\_correction(longitude)*

#### Time Correction

The time correction is a set of two equations that are added together, the first being Longitude time Correction and the second is analemma time correction.

#### Longitude Time Correction

This equation is easier than it appears because in the United Kingdom we sit on the prime median where longitude is close to zero. So our is zero.

Therefore the code is simple

*def longitude\_time\_correction(self,longitude):*

*longitude\_std\_meridian = 0*

*correction = 4 \* (longitude\_std\_meridian - longitude)*

*return correction*

#### Analemma Time Correction

The equation for the analemma time correction is quite daunting at the first glance

It is made worse by the fact another equation (below) is needed to calculate B.

This equation was not difficult to implement because as we have discovered n is easy to calculate in python. The most important aspect of this equation is remembering to convert B into radians at the end.

*day\_of\_year = datetime.now().timetuple().tm\_yday*

*B = (day\_of\_year - 1) \* (360/365) \* (180/pi)*

*B=radians(B)*

Now to calculate

It was found to be much easier to split the equation into parts.

P1 =

P2 =

P3 =

P4 =

Once each part has been calculated individually they can be substituted back into the equation

*p1 = radians(229.2) \* 0.0000075*

*p2 = radians(229.2) \* ((0.0001868 \* cos(B)) - (0.32077 \* sin(B)))*

*p3 = radians(-229.2) \* (0.014615 \* cos(2\*B))*

*p4 = 0.04089 \* sin(2\*B)*

*return(p1 + p2 - p3 + p4)*

As always the main problem here was remembering to convert the degrees into radians.

#### Solar Elevation

The solar elevation is the second most important equation in the system at is calculates how high in the sky the sun is, however the implementation of all the equations previously are the components of this equation.

Each component already has a function to return the result and pythons math library provides all the sin and cos functions so implementing this equation was easy.

First the other functions are called and their results assigned to a variable.

*latitude = radians(lat)*

*declination = self.solar\_declination()*

*hour\_angle = self.solar\_hour\_angle(current\_time,longitude)*

The latitude is passed in as a parameter.

Once again to make calculating it in code easy it was split into two parts.

The first part being and the second part

*p1 = sin(latitude) \* sin(declination)*

*p2 = cos(latitude) \* cos(declination) \* cos(hour\_angle)*

The parts then just need adding together

elevation = p1 + p2

This is not the finished answer yet because the equation gives sin(elevation).

To calculate the actual elevation an inverse sin needs to performed, in python this is the asin function from the math library

*elevation = degrees((asin(altitude)))*

The elevation is also converted to degrees so it can be used later on.

#### Solar Azimuth

The solar azimuth is most important equation of everything discussed as it will tell the motor how far along the sky the sun is.

Like the solar elevation all the equations needed for it have been implemented into their own functions and can simply be formatted into the following equation to calculate the solar azimuth.

The only new part of this equation is the solar zenith angle, this can be easily calculated using the solar elevation as they are complimentary angles.

*elevation = radians(self.get\_solar\_elevation(latitude,longitude,current\_time\_minutes))*

*solar\_zenith = 90 - degrees(elevation)*

*solar\_zenith = radians(solar\_zenith)*

Again to make the equation easier to program it was split in three parts.

The first being the second and third

The first and second parts make up the top of the equation are calculated using the code below.

*azimuth\_top\_1 = sin(solar\_declination) \* cos(latitude)*

*azimuth\_top\_2 = cos(hour\_angle) \* cos(solar\_declination) \* sin(latitude)*

*azimuth\_top = azimuth\_top\_1 -azimuth\_top\_2*

The bottom of the equation.

*azimuth\_bottom = sin(solar\_zenith)*

These are then combined together to complete the equation

*solar\_azimuth = azimuth\_top / azimuth\_bottom*

An inverse cos needs to be performed to get the actual solar azimuth angle, again like the inverse sin pythons math library provides an acos function.

*solar\_azimuth = acos(solar\_azimuth)*

Finally because the equation only gives a value between zero and 180 it needs a slight modification for use in the system.

*if(hour\_angle > 0):*

*solar\_azimuth = radians(180) + (radians(180) - solar\_azimuth)*

if the hour angle is greater than zero, indicating the time is after noon, then 180 minus the solar azimuth is added to 180.

For example here is are solar azimuth angles each hour using the equation without and with the modification

|  |  |  |
| --- | --- | --- |
| Time | Angle Without Modification | Angle With Modification |
| 0.0 | 1.43 | 1.43 |
| 1.0 | 16.17 | 16.17 |
| 2.0 | 32.66 | 32.66 |
| 3.0 | 47.68 | 47.68 |
| 4.0 | 61.27 | 61.27 |
| 5.0 | 73.8 | 73.8 |
| 6.0 | 85.82 | 85.82 |
| 7.0 | 97.87 | 97.87 |
| 8.0 | 110.59 | 110.59 |
| 9.0 | 124.64 | 124.64 |
| 10.0 | 140.64 | 140.64 |
| 11.0 | 158.83 | 158.83 |
| 12.0 | 178.46 | 178.46 |
| 13.0 | 161.5 | 198.5 |
| 14.0 | 143.2 | 216.8 |
| 15.0 | 127.11 | 232.89 |
| 16.0 | 112.98 | 247.02 |
| 17.0 | 100.21 | 259.79 |
| 18.0 | 88.14 | 271.86 |
| 19.0 | 76.14 | 283.86 |
| 20.0 | 63.64 | 296.36 |
| 21.0 | 50.11 | 309.89 |
| 22.0 | 35.16 | 324.84 |
| 23.0 | 18.74 | 341.26 |

The modification allows the solar azimuth to be represented using 360 degrees to allow for a continuous rotation.

### Tracking

#### Servo Control

The first prototype of the tracking was very basic. The goal was to program a servo to move its arm at a specified angle. This was an easy task because of a very helpful online tutorial at razzpisampler.oreilly.com (30). The author provides the code necessary and provides a update function to easily change the angle.

*def update(self, angle):*

*duty = float(angle) / 10.0 + 2.5*

*pwm.ChangeDutyCycle(duty)*

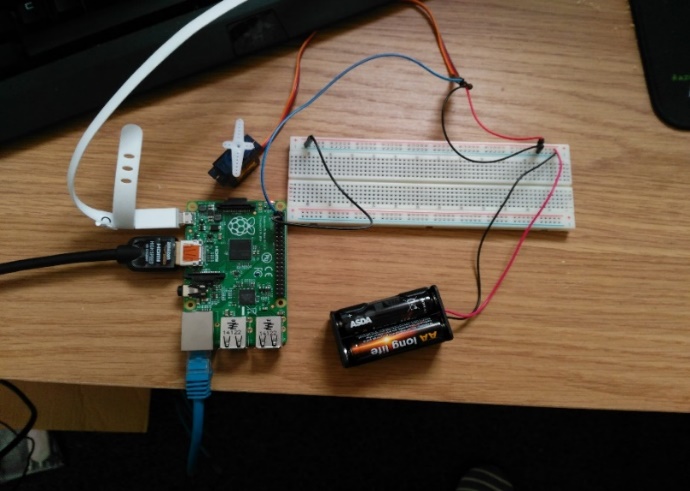
The code had to be modified slightly because my servo motor was connected to a different GPIO pin that the one in the tutorial. The biggest problem that was encountered was not with the development but trying to understand how the duty cycle was calculated.

Figure 6: Prototype one - Servo Circuit

To ensure the angle was accurate it was measured using a protractor which confirmed that the angle to duty cycle conversion worked appropriately.

#### Equation Integration

The key aim of prototype two was to get the result from the solar azimuth equation and use that to position the servo arm. The solar azimuth was already in its own function so it could be called very simply. A crude approach was taken whereby the servo control code was copied and pasted together with the solar equations into one file. However it did work.

*angle = solar\_azimuth()*

*update(angle)*

*sleep(0.5)*

A problem was encountered whereby the arm was not rotating the full amount. The cause of this problem is still unknown however it is theorized it was because the pulse width modulation controlling the servo movement was being stopped before the full rotation could be completed. The solution found was to use sleep from the python time library to make the program wait until the arm has moved fully. The cause may be unknown however the solution works well.

#### Servo Controller Class

The servo controller class was designed because of the need to operate more than one servo.

The class has four methods include the constructor.

*def \_\_init\_\_(self,pin):*

*def update(self,angle):*

*def start(self):*

*def stop(self):*

##### Init

Init is the constructor, when a servo controller object is created the GPIO pin that the controller needs to control is sent to the constructor. The constructor will then perform the necessary set up on the pin

##### Update

Update is the most important method of the class as it is what moves the servo arm. The code has changes somewhat from prototype one but the principles are the same. The function uses the angle parameter to calculate a duty cycle and then uses that to position the servo arm.

##### Start and Stop

The start and stop methods are for starting and stopping the pulse width modulation. As servos have a tendency to twitch if the pulse width modulation is left on, it is optimal that after each movement the PWM is turned off. However if the PWM is switched on and off very quickly the python code crashes. The solution to this is to separate start and stop into their own methods as this allows the PWM to be stopped after a single movement or left on if there needs be continuous movement such as when scanning.

#### Solar Equations class

The solar equations class was developed to keep the equations for calculating the solar elevation and azimuth in one place and separate from everything else. The class has methods for each of the equations described in section 4.1.1.

The main two methods, however are get\_solar\_elevation and get\_solar\_azimuth.

Both methods are implementations of the equations explained earlier, they return their respective angles in degrees. This allows the return of these methods to be directly passed into the Servo Controller class.

*azimuth\_angle = self.equation.get\_solar\_azimuth(self.latitude,self.longitude,current\_time\_minutes)*

#### Tracker Class

The tracker class is the main control class for mathematical tracking in the system. It will store physical properties such as the latitude and longitude and pass these to the solar equations class to get accurate readings for the solar elevation and azimuth.

The tracker also controls the servo through the use of the servo controller class. In this iteration the class can only manipulate single axis tracker, dual axis tracking will be added later.

To control a single axis tracker the SA\_update function is called. This function will create an object of the servo controller class and pass the GPIO pin the servo is connected to. It will then proceed to get the current time in minutes from the solar equations class and then get azimuth angle by calling the get\_solar\_azimuth function. This angle is then sent to the update method in the servo controller class which will move the servo arm to that position.

It was discovered at this point that the solar azimuth being generated was correct, however it would not work with servo.

The first problem was that the solar azimuth is measured as an angle clockwise from north. Where 0o is North, 90o is east, 180o is south and 270o is west. The servos being used are not capable of a full 360o rotation and are limited to 180o

The second problem was that the servo when moving incrementally moves anti-clockwise when facing south.

The solution the first problem is to subtract 90o. This means that East is now 00 and west is 180o, as the sun rises in the east and sets in the west, north east and north west are not important and can be disregarded. The program will interpret anything below zero or greater than 180 as it being night.

The solution to the second problem is to flip the angle so 0 becomes 180, 50 becomes 130 etc. This is done by subtracting the azimuth angle from 180. This will make the servo arm effectively move backwards.

#### Scanner Class

Scanning is where the tracking system will search for the best angle for the solar panel. It does this by rotating the panel through 180 degrees and checking the voltage at each angle.

When the scanner class is initialised the channel of the solar panel and the gpio pin are passed as parameters. The channel is used to create a voltmeter object and GPIO pin is used to create a servo controller object.

The main implementation is within the scan function. The scan function will loop from zero to 180 in increments of ten degrees. Ten degrees was chosen because it allows for a good coverage and can be performed relatively quickly. The speed at which the scanning is done is important because whilst scanning the solar panel will be moved into sub optimal positions ie towards dark spots or into shadows, this less time spent in these position the better.

At each angle the servo controller class is used to position the servo arm and the current voltage being generated is read from the ADC. A check is performed to see whether or not this is the highest voltage so far. If it is the angle and the voltage are recorded. The voltage is saved to be used as a comparison for the next angle.

When the scanning is complete the best angle found is returned and the controller class is used to set the solar panel into that position. The scanning is currently activated every thirty minutes.

#### Dual Axis Tracking

Dual axis tracking is where the solar azimuth is used to position one motor and the solar elevation is used to position another in order to create a more precise way of tracking. Dual axis tracking is implemented into the Tracker class as a separated function called DA\_update. If a solar panel is using dual axis tracking the solar panel class will know and call the DA\_update function automatically.

This was more difficult as it requires to co-ordination of two motors rather than just one. The solution to this already existed in the form of the servo controller class. Two instances of the controller class are created, the first to control the top motor (elevation) and the second to control the bottom motor (azimuth).

Once the motor controllers where initialised the solar elevation and azimuth needed to be calculated. As discussed in section 4.1.1 the implementation of the equations have been completed and placed inside a solar equations class. An instance of this class is created and a call to the get\_solar\_elavation and get\_solar\_azimuth methods are made. The results of these functions are then passed into the servo controllers update method.

*azimuth\_angle = self.equation.get\_solar\_azimuth(self.latitude,self.longitude,time) elevation = self.equation.get\_solar\_altitude(self.latitude,self.longitude, time) bottom\_servo.start() bottom\_servo.update(azimuth\_angle) bottom\_servo.stop() top\_servo.start() top\_servo.update(elevation) top\_servo.stop()*

#### Dual Axis Scanning

Dual axis scanning is implemented in the Scanner class through the DA\_scan function. It uses the same technique as the dual axis tracker for controlling both motors whereby two instances of the controller class are created for the top and bottom motors. Two for loops are used to create each possible combination of angles. The outer for loop will determine the azimuth and the inner loop is the elevation. Instead of using equations to position the panel towards the sun it, at each angle combination it will get a reading from the voltmeter (section 4.1.3) and record the highest, the same as the single axis scanner.

### Voltmeter

#### Prototype One – Getting an ADC Reading

The first prototype of the ADC involved just trying to get a reading from the ADC and displaying it on the raspberry pi.

This was quite a challenge because personally I have never used an ADC before and I am not hugely familiar with this type of programming.

The ADC used initially was MCP3008, the 10 bit version of the MCP3208.

Getting a value from the ADC has two stages, reading the channel on the ADC and converting the value into a voltage.

##### Reading the Channel

The MCP3008 has eight channels numbered zero to seven. To read a channel Serial Peripheral Interface(SPI) is used. SPI is a bus used to communicate with embedded devices. To this in code python has spidev library to help. The code for this is below:

*adc = spi.xfer2([1,(8+channel)<<4,0])*

*data = ((adc[1]&3) << 8) + adc[2]*

##### Converting to Voltage

The reading the MCP3008 gives is between 0 and 1023. This reading can be converted into a voltage.

The equation for this is

And its python counter part

*volts = (data \* 5) / float(4095)*

where data is the ADC reading.

The conversion to voltage is in its own function and the ADC reading is passed as a parameter.

#### Prototype Two

Prototype one provided a good starting point for building a Voltmeter class. The system needs to be able to read the voltage being generated by different panels. To this the voltmeter needs to able to choose which channel on the ADC it is reading. This is done by passing the channel into the constructor of the Voltmeter class.

The voltmeter class has four functions, including the constructor they are read\_channel, convert\_reading and get\_voltage.

The constructor will assign the channel passed into it to member variable.

This variable is then used in the read\_channel function along with SPI code discussed earlier to get a reading.

The voltage read by the voltmeter class was cross referenced using a digital multi-meter to ensure it was working correctly. The values were within 0.01 Volts of one another, which is acceptably close enough.

### Panel Information Server

#### Loading Panel Information

Information about the physical solar panels is stored in a text file called Panel\_Information.txt. When the panel information server is started it will read this text file and parse its contents. The contents will be used to create an instance (or multiple instances) of the solar panel class, each corresponding to a line in Panel\_Information.txt. Whilst the file is being parsed the code will check what kind of tracking should be initialised, from this it will know other parameters that it should find. For example if it’s a dual axis tracker it will know that there should be two GPIO pins listed. Using comma separated values in a text file is not very a technical way of doing this, however it is effective and easy to use. The text file also means that the panels can have their parameters changed very quickly, a single axis tracker can become a single axis scanner or even a fixed panel with the change of one value.

#### Solar Panel Class

The solar panel class was implemented to represent a solar panel, therefore multiple instances of the solar panel class are created for each solar panel in the system. It was important when implementing this class that each solar panel is individual. The class does exceed the scope to include the code for starting the tracking. Whilst the tracking is not a physical part of the solar panel it is directly related and it was deemed important that each solar panel have its own tracker. Integrating the tracking into the class seemed to the simplest way of doing it.

The constructor for the class takes the channel of the solar panel on the ADC, the tracking mode and GPIO pin(s) of the servo, as parameters and assign them to member variables. After this it will then create an instance of the voltmeter class as a member variable that will be used later on.

The class will then wait until its initialise tracking method is called. This is function is used to start the tracking system for the class, it will determine the type of tracking using the member variable set inside the constructor. To start the tracking it will create an instance of the relevant tracking function, this could be either tracked or scanner. Once the tracking class has been created the appropriate start function for the tracking class will be called, for example either single axis or dual axis tracking. The tracking is started in a separate thread so once it is started then it will run in the background and not interfere with the rest of the program.

Finally the class has various “getters”. These are methods use to get information about the solar panel, for example the get\_voltage function will use the voltmeter object created in the constructor to get the current voltage of the solar panel and then return that value. Other getters include, get\_channel, get\_angle\_mode and get\_gpio\_pins. The getters are called to form a reading that will be transmitted to the data server.

#### Information Server

When the data server sends a request for some readings, it will send it to the information server. The information server, will iterate through each of the solar panels that it has created and get a reading from each one of them. When it has a complete set of readings it will transmit the readings back to the data server. The server is multithreaded so it can handle requests from multiple sources. By default the information server will listen for any connections on port 15000.

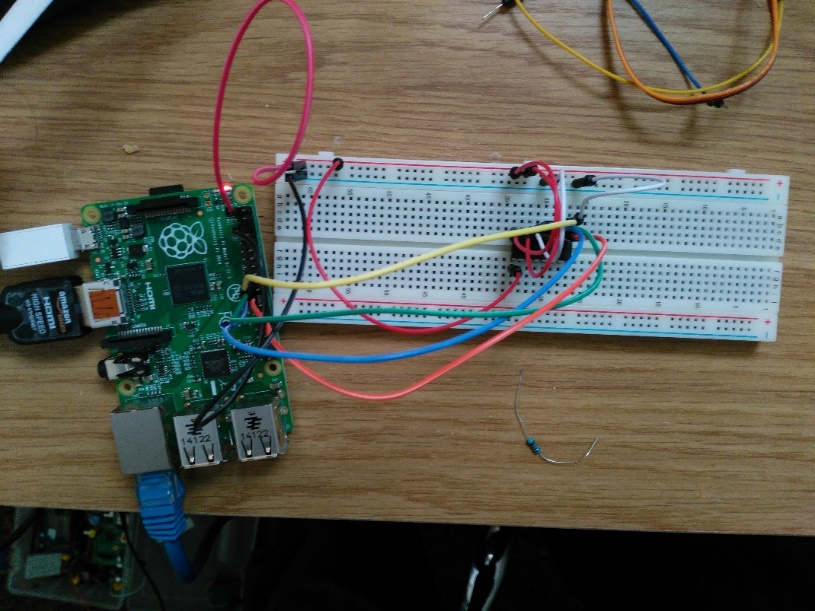


Figure 7: MCP3008 Voltmeter Circuit

## Web Server

### Web Server Beginnings

The web server did not require much development to begin with as in a previous personal project a web server had been developed before. This allowed for the code to mostly be copied over. More work was done removing features from the web server because whilst important aspects of a proper web server, they just added complexity and more areas for something to go wrong.

### Home Page Design

The index page is hard coded into the web Server so that it can directly get data from the data server and into the users web browser. This is done using a socket connection to the data server and using the custom GET request to get the data. This data is then used by being embedded into the HTML of the web page.

The index page is an important aspect of the project as it will be the first real aspect of the system a user will interact with. Therefore it needs to balance looking and nice yet still provide plenty of useful information to the user. An initial version was created of what the web page should look like, this version does not look nice, however the layout is more important at this stage and that particular is good. The focus is on the layout rather than the aesthetics because ultimately this is a working system, having the data laid it in a more concise and readable is more important than the colours matching or complimenting each other.

### Live data

The live readings are an important aspect for web page, it is supposed to show the user in real time how the solar panel is performing. This is implemented in the HTML of the web page served by the HTTP Server Module. It uses the web page designed in 5.14 and custom get explained in section 5.9.

In the HTTP server a module a new URL was configured, on the test machine this URL translates to <http://localhost:9000/data>, this location is where the custom get requests are sent. Upon receiving a custom get request the /data location returns the latest results that were entered into the data module. To return the last result entered the custom get request is simply GET\_DATA A 1.

The biggest problem encountered was here, it was not immediately clear how to get live updates onto a web page without having to refresh the web browser. Typically web pages are mostly static with a few animations or fancy effects. Therefore asynchronous HTTP requests seemed quite difficult.

As a side note JavaScript is a fantastic language, it’s not particular nice, the syntax is a bit verbose, the ambiguous need for semi-colons(or not) and sometimes it can just be a bit funny. However all JavaScript code is open source by nature, as its embedded into the HTML of every webpage it’s very clearly visible. This has led to a huge amount of support available, and almost anything you can think has already been made and someone will have created a tutorial for it. As was the case for asynchronous HTTP requests.

AJAX aptly named Asynchronous JavaScript and XML was the key needed. AJAX is strange, rather being a library or a framework it’s a collection of techniques. It was coined in 2005 by Jesse James Garrett and was described as a new approach to using technologies such as HTML, XML, JavaScript and the most important part XMLHttPRequest objects (31). Using XMLHttPRequest objects it became possible to send a HTTP Request to /data which would then return the last reading recorded by the data module. Getting the data was the hard part, once the web page had it all it took was simple to JavaScript to insert the new data at the top of an existing table and it would live update.

## Data Server

### Setup

The data server is split into two parts. The first is requesting data and second is responding to requests.

Both of these features are separate and both need to run at the same time. Unfortunately for the data requests the server that responds to requests will block it and the requests will never get made.

One solution to this is to have the data server request data every time it receives a request for data. This is not ideal because if no requests are made to the data server, then no data is going to get stored into the data server.

The second solution is multi-thread. Multi-threading allows code to be ran asynchronously in threads, this is what is used in the data server. When the module is started a threading is started to respond to requests and another thread is started to request measurements.

*data\_server = Data\_Server.Data\_Server()start\_thread = Thread(target=data\_server.start\_server)*

*start\_thread.start()request\_thread = Thread(target=data\_server.request\_measurements)*

*request\_thread.start()*

### Requesting Data

To request data a socket is opened to the panel information server. If the socket cannot connect to the server then an error message is displayed asking the user if they want to re-connect. This was added because it’s helpful to the user also it was a disruption during development that every time the panel information server was restarted, the data server would crash.

The socket will then send the string “Hi” to the panel information server. This string does not matter because the panel information does not actually look at it and will return a measurement anyway.

When the measurement has been received it is entered into a database (section 4.3.4) . There is then a delay for one second, afterwards the process will repeat itself.

### Responding to Requests for data

The data servers response server starts on port 15000 when the module is started. The response server is multithreaded so that I can handle multiple connections without blocking.

When a connection is opened to the server, the server will wait for something to be sent, this where the blocking occurs. When data has been received the server will parse the request by splitting it by white spaces.

The first value the parsing looks for is what type of request it is. Currently only the GET\_DATA request is implemented but the code is written in such a way that new requests can be added easily.

If the value is a GET\_DATA request the server then reads the next value. If the request is formatted correctly this value should the type of data the request wants. Once the server has determined the type of data it will move onto to parsing how many readings have been requested.

Once the request has been successfully parsed the data will be retrieved following the criteria of the request and formatted as a comma separated string, if multiple readings have been requested, each reading will be separated by a forward slash(/).

The string is then returned to whomever sent the request.

### Data Persistence

To save the readings the database class was implemented, the class uses an SQLite3 database and the sqlite3 library provided by Python. The data server will create an instance of the database class when it receives a reading, it will get the current date from the datetime library and use the date as the filename.

The database will then be opened if it exists or created if it does not, the same will be done for a readings table inside the database.

Once the database has been opened the insert method is called and the reading is used as a parameter. Inside the insert method the reading will be parsed into its component parts and an SQL insert statement will be constructed. The SQL is then executed and the changes committed to the database to ensure that they are saved.

Overall this class provides an easy interface for interactions with the database and I am pleased with how it works.

Development of this was used with help from the Python SQLite3 documentation (32).

# Experiments and Results

## Dual Axis Tracking vs. Single Axis Tracking vs. Fixed Attempt One

This first test involved comparing the fixed, single axis and dual axis tracking systems. The main aim of this test was to confirm that the system works as expected and to see whether or not the data would confirm what was found by other researchers. The expected conclusion was that the dual axis tracker would perform best, followed by the single axis and finally the tracked panel. It was also expected this test would not go smoothly as it was the first time the system was actually being used.

### The Test

The test was performed on the 29th of August 2016 in Grimsby, Lincolnshire (53.540799, -0.159103 ) in my back garden. The weather was mostly clear all day with some spots of cloud. The panels were set up such that at noon they would be facing south, to ensure this was accurate a compass was used. The test began at 9:04 UTC+1 and finished at 20:01 UTC +1.

The angles of the tracked panels were updated every 30 minutes and readings taken every second for each tracking type.

The fixed solar panel was set up to be oriented south all day. The elevation of the solar panel was set at 45o, a protractor was used to ensure this was accurate.

The single axis tracker is set up to use the solar azimuth to position itself towards the sun. Like with the fixed panel the elevation was set at 45o through the use a protractor.

The dual axis tracker uses both the solar azimuth and solar elevation to position the solar panel therefore did not require any manual position other than to position so it would be south facing at noon.

### Problems

There were two major problems and some several areas identified for improved in future experiments.

The first major problem was that the experiment started too late and therefore missed arguably one of the most important data points; the sun rise. This was down to poor planning more than anything and the solution is to set the experiment up earlier so that it is already when running the sun rises.

The second major problem was that the solar panels became shadowed by some nearby conifers in the late afternoon. As with the sun rise this meant that the panels missed the other most important data point, the sun setting. This was again due to poor planning and for future experiments a better location has already been selected.

Some of the areas of improvement identified are that the tracked panels could be updated more often perhaps increasing the frequency to every 15 minutes, also reducing the frequency readings are taken. Whilst taking readings every second gives a lot of very useful data, it has proven to be difficult to work with when analysing the data.

### System Evaluation

This was the first time the system had been tested in the environment it had been designed for. Aside from the environmental problems the system performed fantastically. No bugs were discovered and each module just worked without any problems.

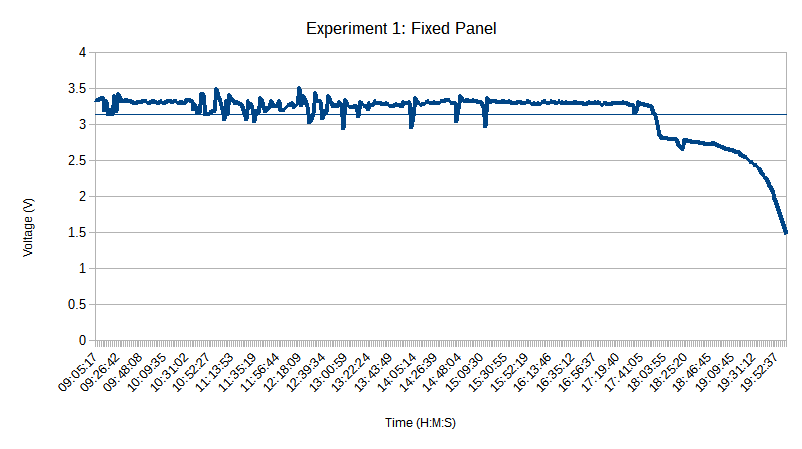
### Results

Despite the problems with shading the results still give a reasonable representation of how each of the panels performed across the day.

#### Fixed Panel Results

The average voltage across the entire experiment for the fixed solar panel was 3.1470 Volts and the median voltage was 3.2882 Volts.

Due to the large number of results (around 36,600) a graph of all the results could not be produced as it would cause Libreoffice Calc to exit. Therefore a python script was written to select every 100th result and a graph was produced using those results.



The area of choppiness between 10:31 and 12:53 corresponds with times of increased cloud coverage.

The graphs shows that shortly after 17:41 is where the solar panels become shaded. The exact time of this was identified as 17:47:15.

Due to the shadowing the average of all results is not really representative of how well the each type of tracking was performing however the average of all the results up to 17:47:15 should be.

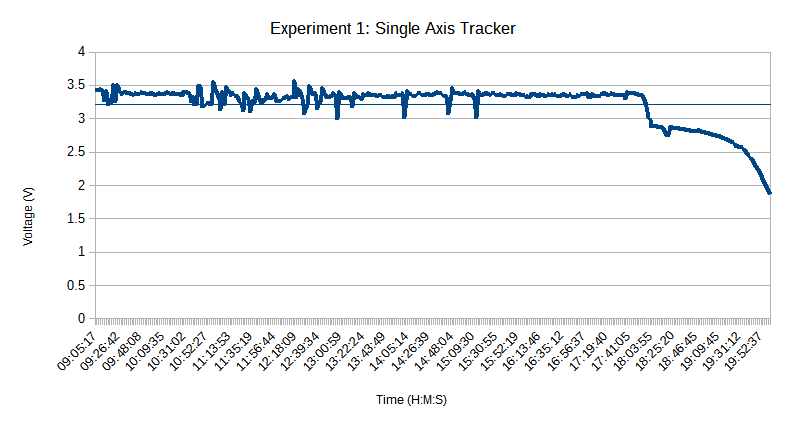
Doing this the average voltage rises to 3.2860 Volts and the median is 3.3004 Volts.

Out of curiosity the average voltage post 17:47:15 was calculated. In the time while the fixed panel was shaded and whilst the sun was setting the average voltage was 2.5962 Volts and the median was 2.6935 Volts.

#### Single Axis Tracked Results

The average voltage across the experiment was 3.2180 Volts and the median was 3.3455 Volts

This shows an increase in average voltage by 2.3% and an increase in the median by 1.8%

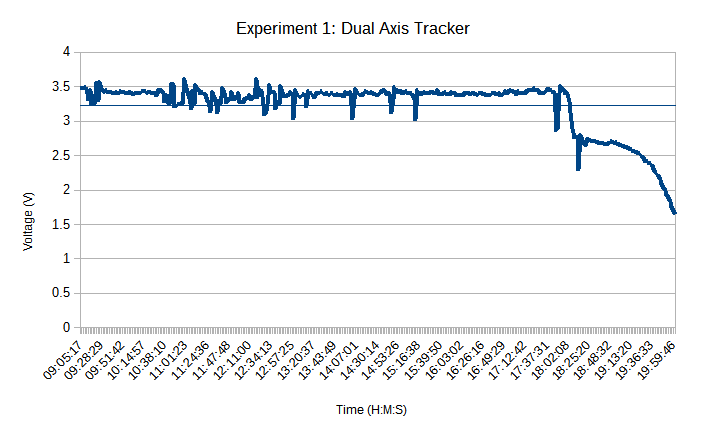


The average of voltage pre 17:47:15 raises to 3.3416 Volts and the median 3.3565 Volts

The average voltage post 17:47:15 was 2.7280 Volts and the median 2.7839 Volts

#### Dual Axis Tracked Results

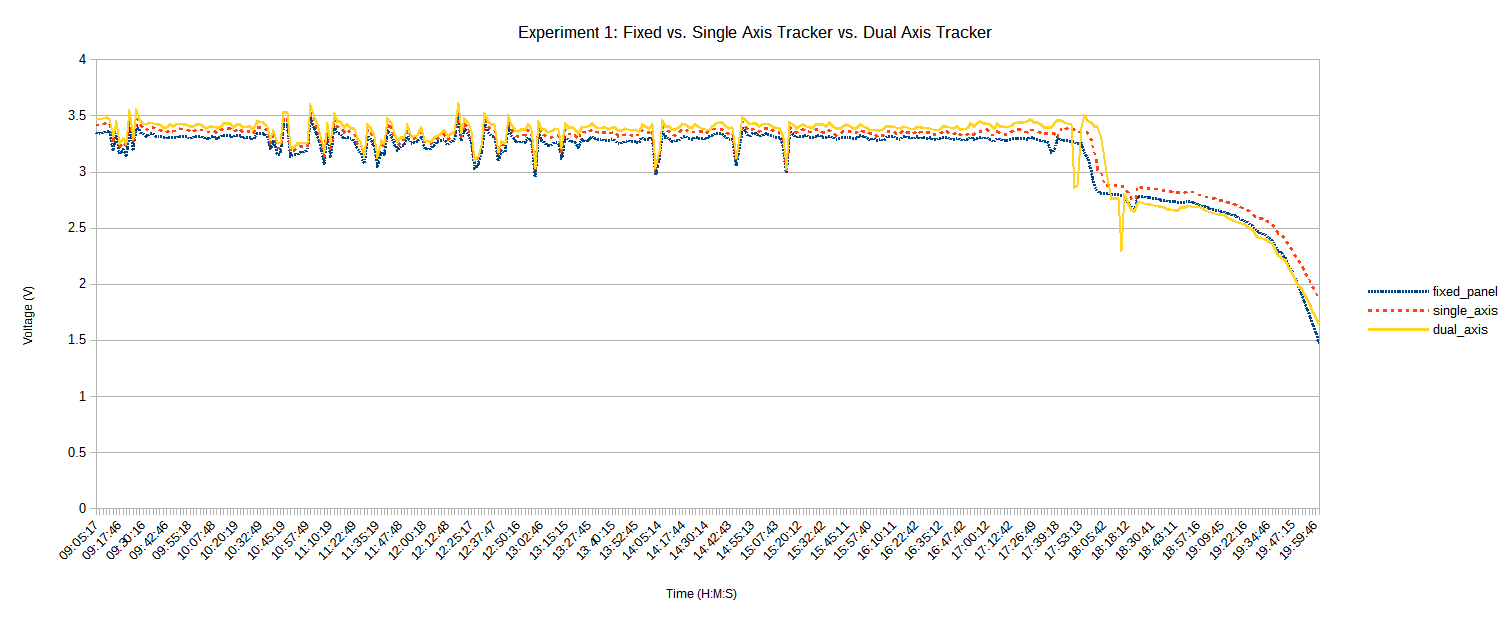
The average voltage across the experiment was 3.2296 Volts and the median was 3.3956 Volts



The average of voltage pre 17:47:15 raises to 3.3886 Volts and the median 3.3455 Volts

The average voltage post 17:47:15 was 2.5992 Volts and the median 2.6606 Volts

### Conclusion



When looking at the results before the shadowing takes place it is clear that the dual axis tracking performs best, followed by the single axis and the static panel performs the worst. This conforms to the results found by Kacira, Mehleri and Li when they performed similar experiments.

Interestingly though, whilst in the shade of the conifers the single axis tracked solar panel performed the best. I believe this is because the elevation angle (45o) of the single axis panel meant that it would capture the rays of light better when facing the conifers, whereas the fixed panel was facing the wrong direction and the dual axis was angled too far down (due to the actual solar elevation) causing it to face the bottom of the conifers.

## Dual Axis Tracking vs. Single Axis Tracking vs. Fixed Attempt Two

In order to try and capture the sun rise and sun set the experiment described in 5.1 was repeated. The sun rise and sun set are arguably the most important times of the day for a solar panel system as it is when a fixed panel is at the biggest disadvantage.

### The Test

This test was performed on 1st September 2016, it began at 6:58 UTC +1 and ended at 20:30 UTC +1. The parameters for the test where kept the same as in section 5.1.1. The day was slightly cloudy in the morning but cleared up later.

The system was placed on top of the garage at my home in an effort to avoid the shading that occurred earlier.

### Problems

Frustratingly the start of the system was delayed due to some technical and set up difficulties. Positioning the panels into the new position proved more difficult than anticipated, the new position was higher to avoid the shade, but the highness made it cumbersome and more time consuming than anticipated. Unfortunately the panels could not have been positioned any sooner due to concerns on how the early morning damp would affect the batteries and other electrical equipment.

The first technical difficulty happened whilst position the panels, whilst trying to manoeuvre the board they are attached to several wires came loose. This was quick to repair however upon starting the system it became evident the equations were not being calculated correctly, after some frantic debugging it was found that some code used for testing purposes the night before had not been removed. Once this system the code was removed the equations started working again however it was too late to capture the sun rise fully.

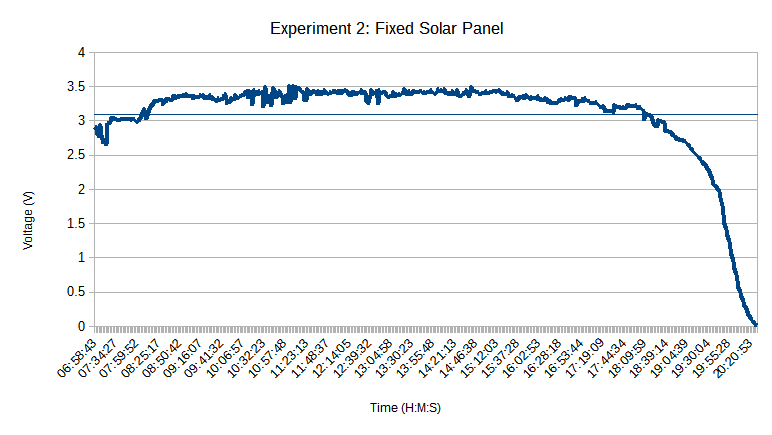
More frustratingly the sunset could also not be captured suitably with the dual axis tracker. The dual axis tracker will position the solar panel based one the solar azimuth and solar elevation. The elevation is a measure of how high the sun is, during the test towards the sunset the sun became obscured behind some trees quite far from the test area. This meant that the dual axis tracker was no longer pointing at the most optimal point and was in fact pointing at a sub-optimal point.

### System Evaluation

Aside from the early issues the system performed fine, once it was up and running there were no problems at all.

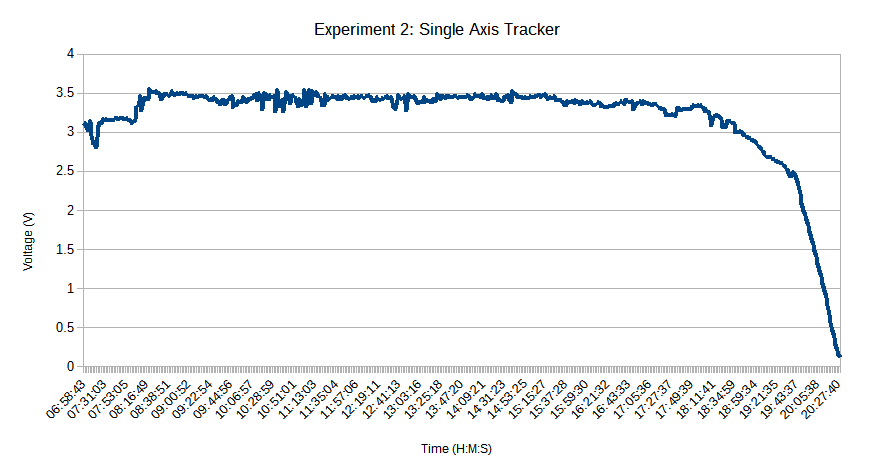
### Results

#### Fixed

The average voltage for the day 3.0837 Volts with a median of 3. 3211 Volts.

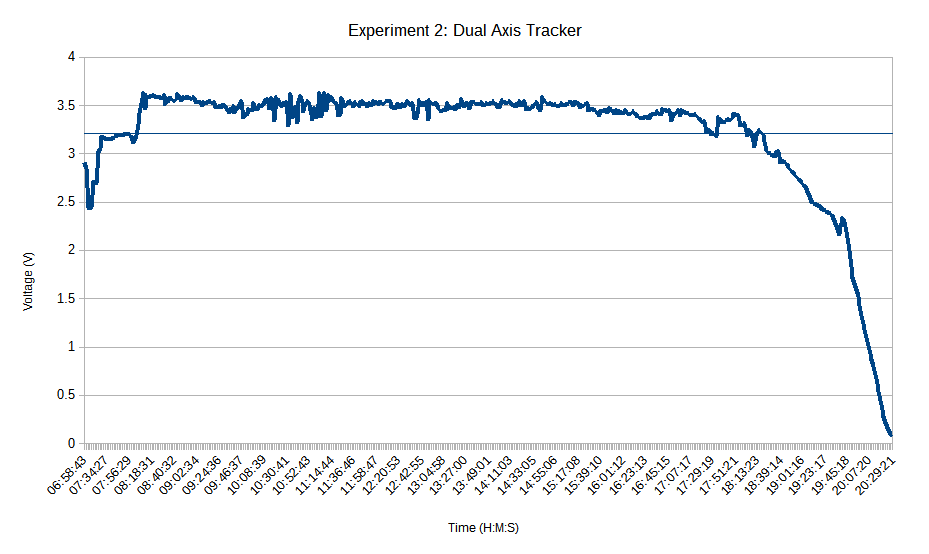
#### Single Axis Tracked

The average voltage for the day 3. 1902 Volts with a median of 3. 3883 Volts.



#### Dual Axis Tracked

The average voltage for the day 3.1861 Volts with a median of 3.4581 Volts.



### Conclusion

The dual axis tracker performed the best during the day as was to be expected however in the early morning and the evening during sun set the dual axis tracker was either matched or beaten in performance by the single axis tracker. I believe the this is because the dual axis tracker is too accurate and at the time where the sun is lower, there is more change of something obscuring the view of the sun. When something does obscure the sun the dual axis tracker will point directly at the object whereas the single axis tracker points more in the general area of the sun. This means that single axis tracker is able to gather more residual rays of light that are reflected and bounced around allowing it to perform better. The fixed solar panel can be seen to performing much worse in the morning and slightly worse during sunset. The large peak at around 8:00 was because the suns peaked over the top of some distance trees.

## Fixed vs. Single Axis Scanner vs. Dual Axis Scanner

### The Test

This test was set up to be a comparison between the type of scanner implemented in the program. The purpose of the scanner is to search for the area with the most sunlight and position the solar panel in that direction.

The test began at 8:48 UTC+1 and finished at 20:30 UTC +1. As in 5.1 the elevation of the fixed and single axis panels was set to 45o and the fixed panel was set to face directly south. The weather was very cloudy all day.

The scanners were set to scan every twenty minutes and a reading was taken every second. The decision to keep the readings at every second was done because when analysing previous data the large set of results meant more anomalous results could be removed without reducing the data size significantly.

The system was again set up on top of the garage.

### Problems

There were a couple of technical difficulties encountered during the test with regards to the circuitry. In particular the ground cable connected to the single axis scanner would fall out. This meant there are periods in the results where the single axis voltage is wrong. When this was noticed the circuit was repaired at the scanner was restarted to make sure the panel is positioned correctly.

The major problem encountered was that whilst the dual axis scanner is scanning, it would cause the voltages of the single axis and fixed panels to drop significantly for a very brief second. This is believed to be because while scanning the motors are drawing more power that interferes with the ADC. It is not known how to fix this however because the system takes so many readings the erroneous ones can be spotted easily and removed.

### System Evaluation

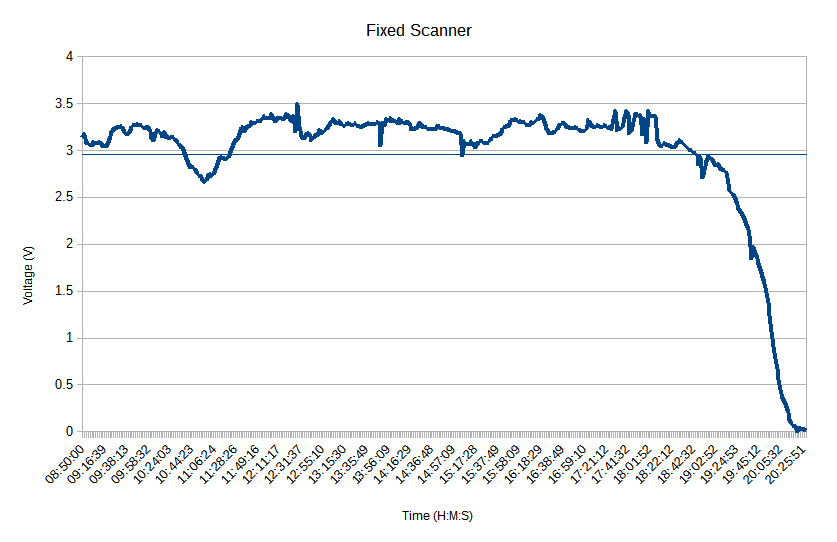
The code again performed well with no issues becoming apparent. The circuit on the other hand could not cope with the increased movement of the scanners which as mentioned meant the ground cable for the single axis panel would fall out and skew the readings. This will be resolved later by securing the cables better, and by more rigorous testing of the motors.

### Results

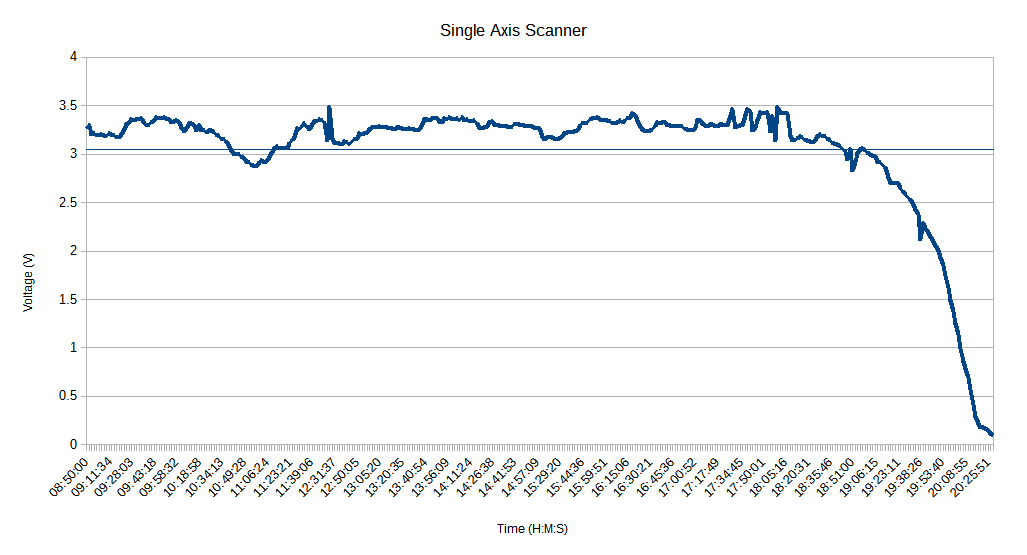
Again because of the large set of data causing the graph software to crash the data for the graphs was reduced to be every 100th result.

Due to the problems discussed with the single axis scanner, the averages were also calculated based off of the reduced results. For any readings where the single axis scanner was not working correctly the readings for the fixed and dual axis were also removed. It was not feasible to go through the full results and remove all the erroneous readings.

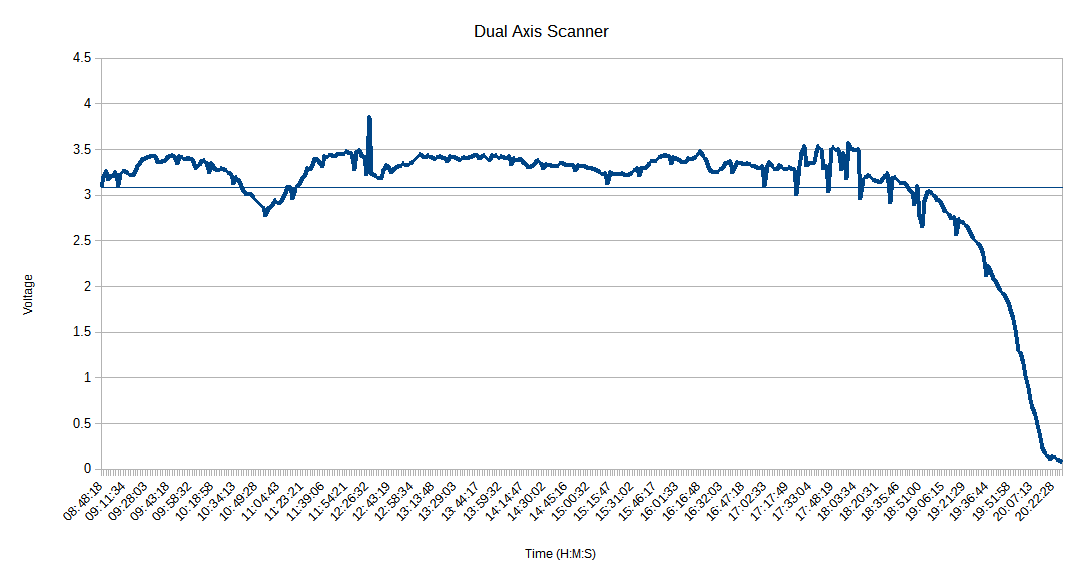
#### Fixed

The average voltage for the day was 2.9459 Volts and the median was 3.1575 Volts

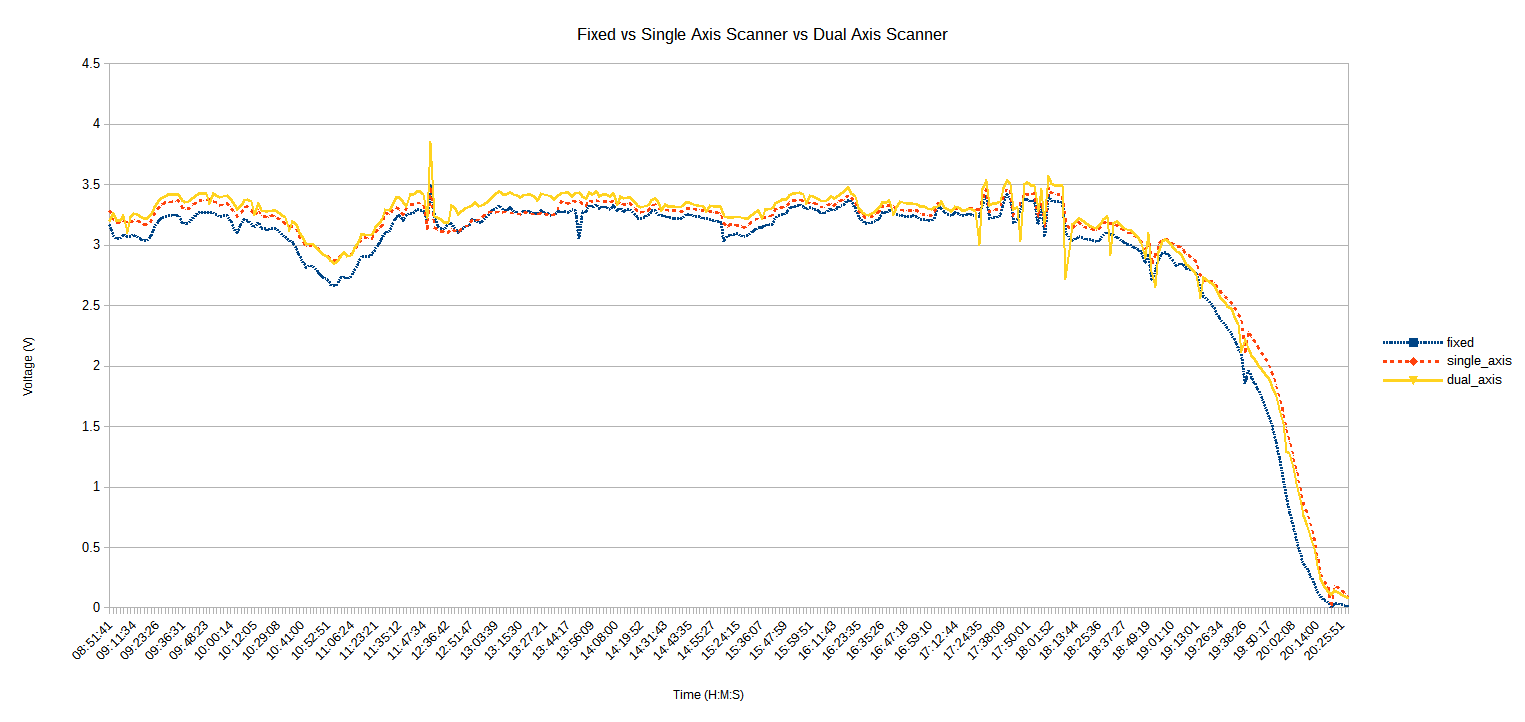
#### Single Axis Scanner

The average voltage for the day was 3.0432 Volts and the median was 3.2540 Volts 

#### Dual Axis Scanner

The average voltage for the day was 3.0824 Volts and the median was 3.3050 Volts 

### Conclusion



The above graph shows that the dual axis scanner performed the best during, this is too be expected because it is able to more accurately determine where the brightest area is. Interestingly at sunset the single axis and dual axis scanner are both about equal. This may be because the light level decreases to the point at which the dual axis accuracy bonus does not help. The fixed panel ultimately performs the worst but can keep up with the single axis tracker at around noon (12:00). This is because the fixed panel is oriented such that it will capture the most sunlight at this time.

## Fixed vs. Dual Axis Tracker vs. Dual Axis Scanner

### The Test

The dual axis tracker and dual axis scanner both performed best in their respective tests. This test is a comparison of them both. A fixed panel will be used as a reference point. This test was performed on the 6th September 2016.

The panels will be set up facing south, the fixed panel will be facing directly south and the panel titled at 45o. As the other two panels are on dual axis systems they do not require any manual modification.

The tracker will update its angle every fifteen minutes and the scanner will scan every 15 minutes. A reading will be taken every second.

### Problems

There were no new problems with this test. The voltage problem explained in section 6.3.2 occurred again, it is still unknown how to fix it.

### System Evaluation

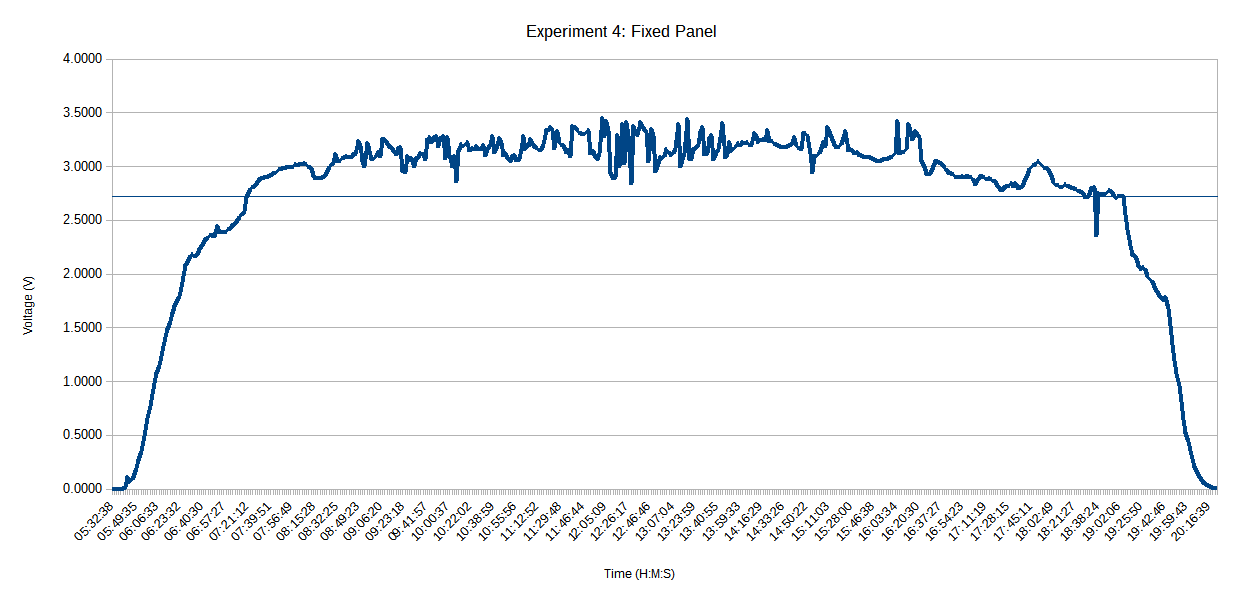
Once again the software ran very well, there were no problems at all.

### Results

The average and median results were calculated using the full data set and the graphs produced using a reduced data set in which every 100th value was taken. For the graphs any results judged to be anomalous are removed.

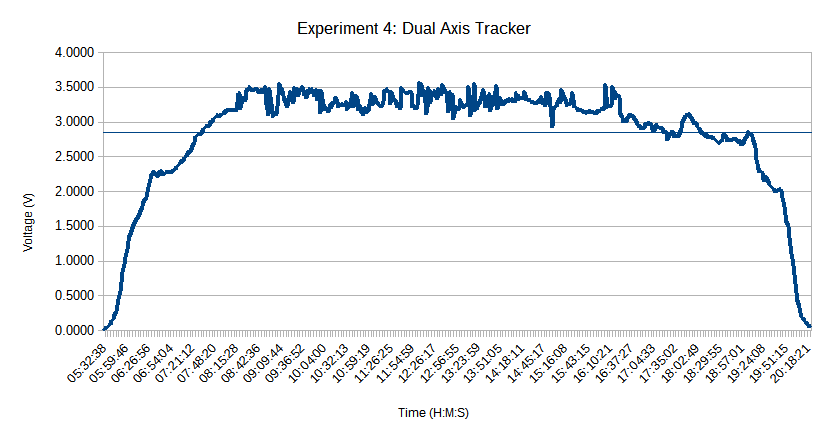
#### Fixed

The average voltage across the day was 2.7106 Volts with a median of 3.0261 Volts



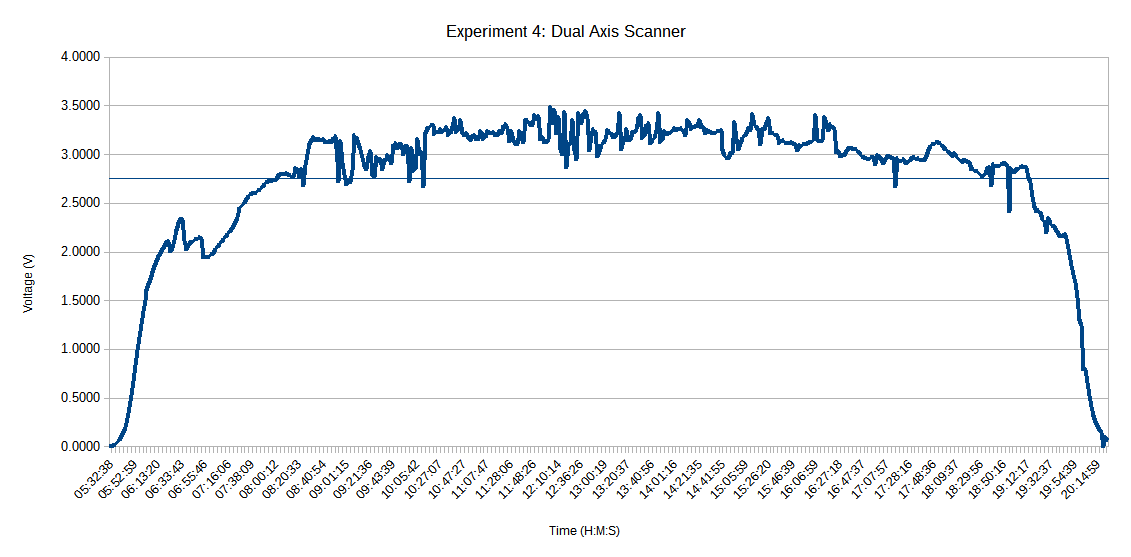
#### Dual Axis Tracker

The average voltage was 2.8345 Volts and the median was 3.1575 Volts



#### Dual Axis Scanner

The average voltage for the day was 2.7400 Volts and the median voltage was 2.9841 Volts



### Conclusion

The results are a bit erratic during the main course of the day, this was because the weather was patchy, being cloudy at times and clear at others.

Interestingly the scanner performed very poorly in the morning, much worse than the fixed but performed either equal to or better than the tracked panel in the late evening. In the morning it started off strong, performing better than the tracked panel but the voltage dropped dramatically and did not recover. It is not clear why this is, it may have been a software error or a problem with how the scanner was set up in the morning or even an environmental problem such as a leaf or dirt covering the panel. It did however correct itself without any human intervention.

As other tests have shown the dual axis tracker performed the best during the main course of the day and performing slightly better in the evening than a fixed panel. This is the first test done that has captured the performance of the dual tracker in the very early morning, the graph shows that it performs slightly better than a fixed panel.

Overall for the day it can safely be said that the dual axis tracker performed the best as it had the highest average and median voltages.

# Conclusion

To understand the general success of this project I will refer back to the questions asked in the introduction.

**Does it work?**

Yes, it does work, it actually worked much better than anticipated, in this regard if someone wanted a tracking and measuring system for small 3V solar panels then they could use it. Though something that was not really considered during the inception of the project but whilst the project in its current form is in no way suitable for a commercial environment, it could be used for educational purposes. Regardless of the size the system shows how a solar panels power output varies across the day, and how tracking system can impact the power output.

**Does it conform to existing research? If not what has been found?**

For the most part yes. Ultimately the results are going to be a little different because when testing the tracking systems in a residential area there are factors and problems that the other researchers did not encounter. For example the neighbour’s garage roof blocking the sunset or trees blocking the sunset however with the tracked system it can be considered that the dual axis tracker works the best.

What I believe I have found to be new is that in these situations whereby the sun is blocked during sunset and sunrise the dual axis is not better and that the single axis tracked system works better because it uses a more general, less direct tilt angle that causes it to receive more residual rays of light. This does not mean a dual axis system should not be used however, with the system that’s been developed the dual axis system can be easily tailored to suit a particularly environment. The system is able to detect when it is morning and when it is evening by using the solar hour angle, from this the system can be told to move to particular angle. If a user knows that the sunset is going to be blocked, the system can be set to move to a particular angle that would be similar to the single axis tracker.

No mention of the scanners has been made yet and that is because while they are very interesting it is difficult reach a verdict about their performance and viability. The scanner proved to be more unreliable in my tests because they are more complex which led to more things going wrong. For example the increase movement would cause insecurely connected wires to come loose which did not happen with a tracked system. Unfortunately it was not recorded but the increased movement of the motors will have meant more power was used to power them, this may have had a noticeable impact on the total power generated, if this is too noticeable then a fixed panel may have offered better overall performance.

Both the single and dual axis scanner performed much better than standard fixed panel but at the same time both the single and dual axis tracker performed much better than the fixed panel. My own tests showed that during the day the dual axis tracker performed better than the scanner, but the scanner performed better during the sunset. It is at this point I believe that is no clear better system but it is highly dependent upon the environment. A scanning system is most beneficial when there is very little natural light or the sun is being obscured, for example a very densely built residential area and a tracked system is best the majority of the day is clear from any obstacles.

Rubio et al suggested that a hybrid tracked-scanning system would be optimal and before my tests I agreed. It is has become clear though it is heavily dependent on the environment in which the panels are located. For example my homes back garden, a dual axis tracked system with modifications for morning and evening would be the most optimal solution, because of the obscured sun rise and sun set.

**What does the tracking systems add to a solar panel system in a residential area?**

In the tests performed there is a definite and undeniable voltage increase during the course of the day when any sort of tracked system over a fixed panel. There is also a definite increase in the voltage generated during sun rise and sun set. This leads to the conclusion that there is absolutely some benefit in the form of increased power production for using a tracked system in a residential area. Though as mentioned it is very dependent on the environment and the benefit of tracked system may not be as prominent it all areas.

**Can it be considered low cost? Can it be scaled up?**

The system built is definitely low cost when compared to a commercial tracking system, including the raspberry pi the system cost is about £30 for one tracker, but one raspberry pi can happily run more than one tracker and the MCP3208 has eight channels for eight solar panels.

The biggest upgrade needed will be the motors, the current motors (SG90 Micro Servo) are not powerful enough to move the larger heavier solar panels. A 250W Mitsubishi Solar Panel weighs 20kg (33) which sounds a lot but servos are fantastically powerful little devices. For example an “industrial grade” servo motor with a torque of 38kg/cm is available for purchase for £54.14 (34) would be sufficient.

The other weak point would be the ADC, it is not clear how much voltage you can feed into one before damaging it, unfortunately the data sheet for the MCP3208 does not supply this information. The Mitsubishi solar panel’s maximum open circuit voltage is 37.6V to measure this with the MCP3208 a voltage divider will most likely be needed.

The software would cost nothing as it can be released for free under GNU General Public License.

Overall I am not sure if it can be considered low cost there are a lot of factors I do not have the knowledge to comment on such as labour costs, actual engineering to make it safe etc. I believe however it will be cheaper than the currently available solutions.

**Has anything of value been discovered?**

I think so.

Closed loop systems are not as good as a tracked system due to being more unreliable and higher power consumption but still offer an increase in power production over fixed systems

A tracked system in a residential area works well and provides a clear benefit over a static, fixed panel.

Dual axis tracking does not work as well a single axis tracked system during sun set and sun rise if the sun is partially blocked during these times, however with modifications to the dual axis system it could work just as well.

**What improvements could have been made?**

It would have been ideal to be able to do more day long tests of the system, comparing different tracking methods. More tests would have made the results and implications of each much clearer and not leave room for any assumptions. It would have also made the results far more reliable, in particular it would have been best to do the test described in section 6.4 again due to the strange morning readings for the dual axis scanner. Unfortunately more tests could not be completed in the time frame of the project due to the long length of the tests and they require constant supervision to ensure nothing goes wrong. As an extra feature to the software, a method for detecting when there was a problem would be implemented so that the system required less supervision.

Another improvement would be to complete more tests in different locations, for example around other homes in different circumstances. This would give a much better understanding of how the system performed in different types of neighbourhoods in entirely different environments. This could not be done because there was no access to other areas. Also the system in its current form is not very portable because of the raspberry pi. The system does not need to be networked so this was not a problem but the raspberry pi does need powering which could only be done through USB connected to a mains socket. It is possible to the power the raspberry pi using batteries however the batteries do not last very long.

**My Opinion**

When looking at the project, looking at what I have created and the work I have put into it. I am very happy with what I have created. This is the biggest project I have undertaken featuring multiple components and technologies that previously I had never used. From this perspective, when looking at just the software and the hardware that I have created the project in my eyes is a great success. When I ran the first test I was amazed that it worked so well, each of the modules worked flawlessly, the motors would move correctly, the information server was getting readings from the ADC and responding to the data server, everything just worked.

When considering the research aspects of the project and whether or not what has been found is useful or reliable, I am not sure. I would like to think what has been is useful, in particular about dual axis trackers performing worse than single axis trackers with obscured sun rises and sun sets. More tests definitely needed to be performed in order to confirm that finding across multiple scenarios and locations. This required resources that I did not have.

# References

1. **Change, Department of Energy and Climate.** Energy Trends: electricity. [Online] 28 April 2016. [Cited: 19 May 2016.] https://www.gov.uk/government/statistics/electricity-section-5-energy-trends.

2. *Climate change: A crack in the natural-gas bridge.* **Davis, Steven J and Shearer, Christine.** 2014, Nature, Vol. 514, pp. 436-437.

3. **Neville, Richard C.** *Solar Energy Conversion: The Solar Cell.* 2nd. Amsterdam, New York : Elsevier, 1995.

4. **Tagare, D. M.** *Electric Power Generation: The Changing Dimensions.* Hoboken, NJ : Wiley, 2011.

5. **Chen, C. Julian.** *Physics of solar energy.* Hoboken, N.J. : John Wiley & Sons, 2011.

6. *The effect of weather conditions on the efficiency of PV panels in the southeast of UK.* **Ghazi, Sanaz and Ip, Kenneth.** 2014, Renewable Energy, Vol. 69, pp. 50-59.

7. *Determination of the optimal tilt angle and orientation for solar photovoltaic arrays.* **Mehleri, E.D., et al.** 11, 2010, Renewable Energy, Vol. 35, pp. 2468-2475 .

8. *El heliostato de la Universidad Santa Maria.* **Finster, C.** s.l. : Scientia, 1962, Vol. 119.

9. *Determining optimum tilt angles and orientations of photovoltaic panels in Sanliurfa, Turkey.* **Kacira, Murat, et al.** 8, 2004, Renewable Energy, Vol. 29, pp. 1265-1275.

10. *Optical performance of vertical single-axis tracked solar panels.* **Li, Zhimin, Liu, Xinyue and Tang, Runsheng.** 2011, Renewable Energy, Vol. 36, pp. 64-68.

11. *A Programmable Logic Controller to Control Two Axis Sun Tracking System.* **Mamlook, Rustom, Nijmeh, Salem and Abdallah, Salah M.** s.l. : Information Technology Journal, 2006, Vol. 5. 1812-5638.

12. *Efficiency improvements of photo-voltaic panels using a sun-.* **A, Al-Mohamad.** s.l. : Applied Energy, 2004, Vol. 79.

13. **LORENTZ.** About us. [Online] 2016. [Cited: 1 May 2016.] https://www.lorentz.de/en/company/about-us.html.

14. —. Solar Tracking Systems ETATRACK active. [Online] 2016. [Cited: 1 May 2016.] https://www.lorentz.de/en/products/solar-tracking-systems/etatrack.html.

15. **Wind & Sun Ltd.** Lorentz Trackers. [Online] 2016. [Cited: 1 May 2016.] http://www.windandsun.co.uk/products/PV-Mounting-Structures/Lorentz-Trackers.

16. **Brownson, Jeffrey R. S.** *Solar Energy Conversion Systems.* Oxford : Oxford Academic Press, 2014.

17. *Design and construction of a system for sun-tracking.* **Roth, P., Georgiev, A. and Boudinov, H.** Valparaiso : Renewable Energy, 2004, Vol. 29.

18. *Application of new control strategy for sun tracking.* **Rubio, F.R., et al.** 7, s.l. : Energy Conversion and Management, 2007, Vol. 48. ISSN: 0196-8904.

19. **Heredia IL, Moreno JM, Magalhaes PH, Cervantes R, Quemere G, Laurent O.** *Heredia IL, Moreno JM, Magalhaes PH, Cervantes R, Quemere G, Laurent O.* s.l. : Springer, 2007.

20. *A review of principle and sun-tracking methods for maximizing solar systems output.* **Mousazadeh, Hossein, et al.** s.l. : Renewable and Sustainable Energy Reviews, 2009, Vol. 13. ISSN: 1364-0321.

21. **Matt.** Analogue Sensors On The Raspberry Pi Using An MCP3008. *raspberrypi-spy.co.uk.* [Online] raspberrypi-spy.co.uk, 20 10 2013. [Cited: 26 August 2016.] http://www.raspberrypi-spy.co.uk/2013/10/analogue-sensors-on-the-raspberry-pi-using-an-mcp3008/.

22. **Stack Exchange.** How can I connect an analog to digital converter (ADC)? *Stack Exchange.* [Online] 26 July 2012. [Cited: 26 August 2016.] http://raspberrypi.stackexchange.com/questions/1293/how-can-i-connect-an-analog-to-digital-converter-adc.

23. **Arduino.** Compare board specs. *Arduino.cc.* [Online] Arduino. [Cited: 27 August 2016.] https://www.arduino.cc/en/Products/Compare.

24. —. What is Arduino? *arduino.cc.* [Online] Arduino. [Cited: 27 August 2016.] https://www.arduino.cc/en/Guide/Introduction.

25. **Oracle Corporation.** The History of Java Technology. *oracle.com.* [Online] Oracle Corporation. [Cited: 07 August 2016.] http://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html.

26. **History of C++. *cplusplus.com.* [Online] cplusplus.com. [Cited: 8 August 2016.] http://www.cplusplus.com/info/history/.**

**27. DzinX. Convert Year/Month/Day to Day of Year in Python. *Stack Overflow.* [Online] 8 March 2009. [Cited: 13 June 2016.] http://stackoverflow.com/questions/620305/convert-year-month-day-to-day-of-year-in-python.**

**28. National Oceanic and Atmospheric Administration. Solar Position Calculator. *noaa.gov.* [Online] noaa.gov, 27 August 2016. [Cited: 13 June 2016.] http://www.esrl.noaa.gov/gmd/grad/solcalc/azel.html.**

**29. Giesen, Jürgen. Approximate Solar Calculations. *jgiesen.* [Online] 4 Octoboer 2007. [Cited: 13 June 2016.] http://www.jgiesen.de/astro/suncalc/.**

**30. Controlling Servo Motors. *oreilly.com.* [Online] oreilly.com. [Cited: 15 June 2016.] http://razzpisampler.oreilly.com.**

**31. Mozilla. Ajax. *mozilla.org.* [Online] Mozilla Foundation, 12 June 2015. [Cited: 15 August 2016.] https://developer.mozilla.org/en/docs/AJAX.**

**32. Python Software Foundation. sqlite3 — DB-API 2.0 interface for SQLite databases. *python.org.* [Online] Python Software Foundation, 24 August 2016. [Cited: 27 August 2016.] https://docs.python.org/3/library/sqlite3.html.**

**33. Mitsubishi Electric. MLU Series Photovoltaic Modules. *mitsubishielectricsolar.com.* [Online] [Cited: 8 August 2016.] http://www.mitsubishielectricsolar.com/images/uploads/documents/specs/MLU\_spec\_sheet\_250W\_255W.pdf.**

**34. Active Robots. High Torque DC Servo Motor 60RPM With UART/12C/PPM Drive . *active-robots.com.* [Online] Active Robots. [Cited: 8 August 2016.] http://www.active-robots.com/high-torque-dc-servo-motor-60rpm-with-uart-12c-ppm-drive.**

**35. Controlling Servo Motors. *oreilly.com.* [Online] 2016. [Cited: 13 June 2016.] http://razzpisampler.oreilly.com/ch05.html.**

**36. PWM. *arduino.cc.* [Online] Arduino. [Cited: 2 August 2016.] https://www.arduino.cc/en/Tutorial/PWM.**

**37. Details about MCP3008 Microchip10 bit ADC converter 16pin DIL 8ch SPI for Pi Arduino PIC . *ebay.co.uk.* [Online] 05 February 2015. [Cited: 14 August 2016.] http://www.ebay.co.uk/itm/MCP3008-Microchip10-bit-ADC-converter-16pin-DIL-8ch-SPI-for-Pi-Arduino-PIC-/111470284252?hash=item19f4258ddc:g:37kAAOSwKIpWCXpR.**

**38. Python object serialization. *python.org.* [Online] Python Software Foundation. [Cited: 08 16 2016.] https://docs.python.org/3/library/pickle.html.**