Nottingham Trent University

School of Science and Technology

A Solar Tracking, Measurement and Logging System

By

Arron Moore

# Abstract

Contents

[Abstract 3](#_Toc459118653)

[2 Introduction 6](#_Toc459118654)

[3 Literature Survey 7](#_Toc459118655)

[4 Design 14](#_Toc459118656)

[4.1 Modules 14](#_Toc459118657)

[4.1.1 The Tracking Module 17](#_Toc459118658)

[4.1.2 The Measurement Module 18](#_Toc459118659)

[4.1.3 Data Server Module 19](#_Toc459118660)

[4.1.4 HTTP Server Module 19](#_Toc459118661)

[4.2 Arduino UNO vs. Raspberry Pi 20](#_Toc459118662)

[4.3 Programming Language 21](#_Toc459118663)

[4.3.1 Java 21](#_Toc459118664)

[4.3.2 C++ 21](#_Toc459118665)

[4.3.3 Python 22](#_Toc459118666)

[5 Chapter 4: Development 23](#_Toc459118667)

[5.1 Equations (TM) 23](#_Toc459118668)

[5.2 Raspberry Pi and Servo Motor (TM) 24](#_Toc459118669)

[5.3 Combining Equations with Servo Control (TM) 24](#_Toc459118670)

[5.4 ADC Voltmeter (MM) 25](#_Toc459118671)

[5.5 HTTP Server Beginnings (HS) 25](#_Toc459118672)

[5.6 Data Server Beginnings (DM) 25](#_Toc459118673)

[5.7 Voltmeter Server(MM) 25](#_Toc459118674)

[5.8 Data Server Multithread and GET Query (DM) 26](#_Toc459118675)

[5.9 HTTP Server Custom GET (HS) 26](#_Toc459118676)

[5.10 Refactoring Voltmeter and Measurement Module (MM) 26](#_Toc459118677)

[5.11 Placeholder Ammeter (MM) 26](#_Toc459118678)

[5.12 Angle Server (TM) 27](#_Toc459118679)

[5.13 Formalising data (MM) 27](#_Toc459118680)

[5.14 HTML Index Page (HS) 27](#_Toc459118681)

[5.15 Live Readings (HS) 27](#_Toc459118682)

[6 References 30](#_Toc459118683)

[Figure 1: Dual Axis Tracker 8](file:///C:\Users\moorey\Desktop\MscPro\Arron_Moore_N0441947.docx#_Toc459118684)

[Figure 2Mamlook et al, Comparison between fixed and tracked solar system (21) 9](#_Toc459118685)

[Figure 3: SG90 Micro Servo 17](file:///C:\Users\moorey\Desktop\MscPro\Arron_Moore_N0441947.docx#_Toc459118686)

[Figure 4: MCP3008 Pin Layout 18](file:///C:\Users\moorey\Desktop\MscPro\Arron_Moore_N0441947.docx#_Toc459118687)

[Figure 5: Servo Circuit 24](file:///C:\Users\moorey\Desktop\MscPro\Arron_Moore_N0441947.docx#_Toc459118688)

[Figure 6: MCP3008 Voltmeter Circuit 25](file:///C:\Users\moorey\Desktop\MscPro\Arron_Moore_N0441947.docx#_Toc459118689)

# Introduction

# 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 (1), 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 (1). 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. (2). 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. (2). 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 (3). 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 (3).

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 (4). 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 (5). 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 (5). 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 (5). 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” (5).

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 (6) 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 (7). 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. (7)

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. (7)

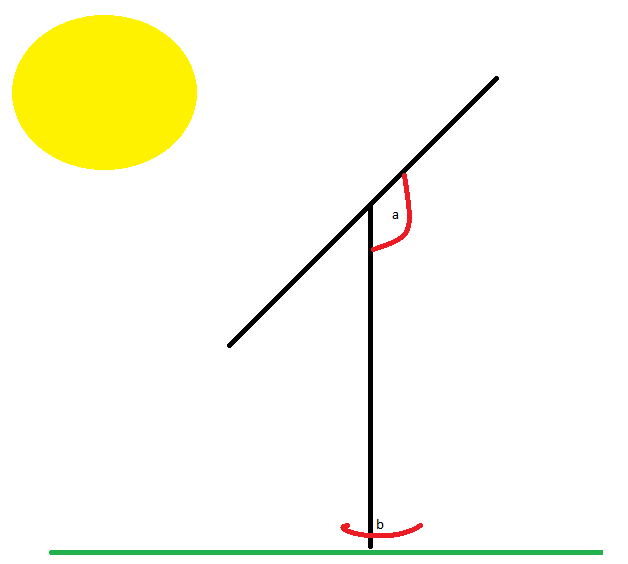
 As Kacria et al found a tracked photovoltaic panel has the potential to produce more power than its fixed alternative. However there are two types of tracked panel. Kacria et al used a two axis panel, these tracking systems can change both the orientation and tilt angle of the panel. Figure 1 demonstrate 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” (8).

Figure : 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 (8).

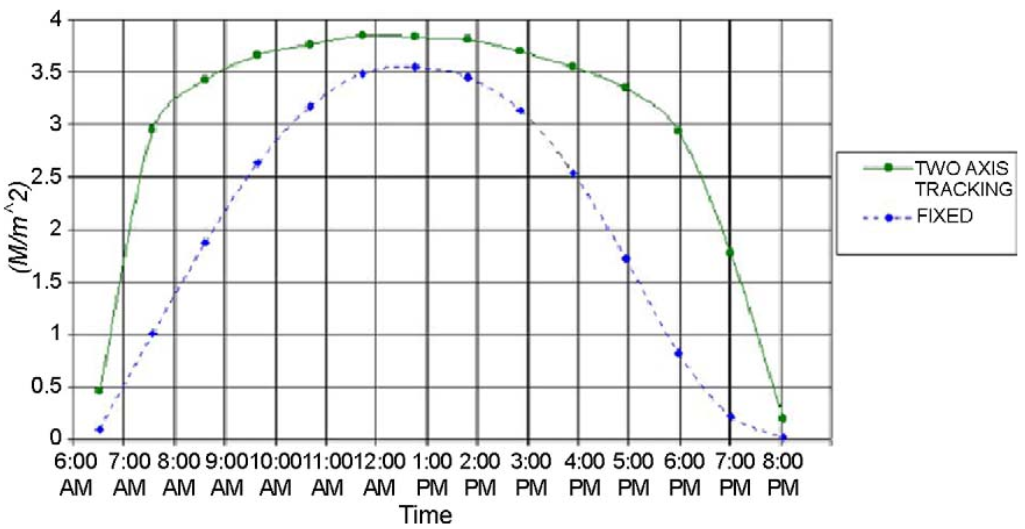
Mamlook et al in their article “A Programmable Logic Controller (PLC) to Control Two Axis Sun Tracking System” looked at designing and created a dual axis open tracked system using programmable logic circuit (9). They used the PLC to periodically to rotate the panel at set intervals in correlation to 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 (9). Ultimately they found that their two axis tracking system resulted in a 41.34% increase in daily solar collection compared to a fixed one (9) and they organised the hourly recording into the graph (Figure 2). 

Figure Mamlook 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) (10). 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 (10). 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 (10).

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 (11). 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 (12). 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 (13). 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 (14).

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 (14).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 (14). 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 (14). 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 (14). 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 (14).

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 (14) 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 (14).

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 (14).

We now have everything we need to calculate the solar altitude. 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 (14).

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 ourselves also 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 the solar altitude equation to calculate the suns position is not the only way we can determine the suns position, 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 (15). 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 (15). 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 (15). 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 (16). 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 (16). 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 (17). However Mousazadeh et al characterise these as being complex which makes them expensive and unreliable (18).

To conclude I believe that a single axis tracked system using the solar altitude 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 simply 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 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.

# Design

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 or developing working prototypes of different part 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 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 project will be split into four key systems that will be defined as modules. The reason for this simple and comes down to one property; modularity.

Being modular means the project is much easier to develop, each module can be developed independently and it is not necessary for one module to be complete for another one to be tested. Modularity also means that the code will be more understandable, as each module is developed separately in its own environment, spaghetti code will not be a problem.

Each module will work on its own and perform its dedicated task, in order to exchange data between modules TCP(Transmission Control Protocol) sockets will be used and a simple custom protocol on top of TCP. TCP has been chosen over UDP(User Datagram Protocol) because whilst UDP offers better performance due to the lack of verification, the performance increase will be negligible and the three way handshaking performed by TCP will be more useful.

The four modules will be for tracking, measuring, holding data, and displaying the data. They are described below. Each module also has a set of core features that are required and a set of secondary features that are less important.

Tracking Module

* Calculates the suns position based on user entered data and solar altitude equation
* Positions the motor arm is such a way that the solar panel is continuously facing the sun
* Will run on a Raspberry Pi
* Will be programmed in Python.

|  |  |
| --- | --- |
| Core Features | Secondary Features |
| Calculate Sun position using solar Altitude Equation | Search for brightest light source |
| Move motor arm in the direction of the sun | Experiment replacing Servo Motor with stepper motor |
| Provide the current angle to other modules |  |
| Provide sufficient error messages to the user when something is not working correctly. |  |

Measuring Server Module

* Gathers data from external sensors such as the voltage, current etc
* Passes gathers data to the data server
* Does not store any data itself
* Will run on a Raspberry Pi
* Will be programmed in Python.

|  |  |
| --- | --- |
| **Core Features** | **Secondary Features** |
| Measure the potential difference generated by solar panel | Use third party API’s to gather extra information such as the weather |
| Measure the current generated by solar panel | More configuration options such as how often to send data. |
| Transmit readings to Data Module |  |
| Request data from the tracking module |  |
| Provide sufficient error measures when something goes wrong |  |

Data Server Module

* Holds data regarding the performance of the solar panel
* Retrieves data from the measuring server periodically.
* Passes data to the HTTP Server on demand
* Will run on any computer
* Will be programmed in Python.

|  |  |
| --- | --- |
| **Core Features** | **Secondary Features** |
| Appropriately handle readings sent by measurement module | More configuration options, such as port number, where to save data etc. |
| Store data in memory using a thread safe data structure |  |
| Store data to disk for persistence |  |
| Provide data to HTTP Server |  |
| Provide sufficient error message to end user |  |
|  |  |
|  |  |

HTTP Server Module

* Displays the data generated by the measuring server in an easy to read well formatted way.
* Uses a very basic implementation of HTTP to display the data in a web browser
* Queries the data server to get data
* Will run on any computer
* Will be programmed in Python.

|  |  |
| --- | --- |
| **Core Features** | **Secondary Features** |
| Request data from data module | Implement more HTTP standards |
| Display data from data module in web browser in tables | Be aesthetically pleasing |
| Correctly parse HTTP GET requests | Display data as graphs. |
| Display live feed of data being recorded |  |
| Be aesthetically acceptable |  |

C:\Users\moorey\Desktop\MscPro\Top_Level.png

Figure : Top Down view of the System.

### The Tracking Module

As mentioned this module’s main purpose is to keep the solar panel facing the sun.

It will do this by calculating the solar altitude using the solar altitude equation as defined by Brownson et al (14). Mamlook et al used a similar technique in that they used “mathematical formulae” however they do not detail specifically what as used.

The module will require the user to enter their latitude and longitude for the equation to work. The equation has been chosen to calculate the suns because it is simple, requires very little computational power and is reliable. The main downside is that it is difficult to understand how accurate the equation is however this makes relatively little difference because even if it is a couple of degrees out it should not affect the performance of the solar panel.

If time allows a closed loop system will be investigated whereby the system actively searches for what it believes the best position to generate the power will be, as this is more complicated it will be a secondary feature implemented was the core features are implemented.

Once the module has the solar altitude it will use pulse width modulation to angle the panel towards the sun. Pulse width modulation(PWM) is a technique for simulating analogue signals. It does this by sending pulses as a square wave. When the wave is high this represents 5v’s (on) and when its low this represents 0v (off). The length the wave is high dictates how far the servo should move. Whilst PWM is effective it does have one major downside in that it causes the servo arm to twitch, this can be solved by using a dedicated servo controller board (19). The solar altitude will be the angle that code uses to position the arm of the servo motor. Currently a SG90 micro servo will be used. The SG90 operates at 4.8v so will be powered by 4 AA batteries and is capable of 1.8kg/cm of torque. This servo has been chosen for the prototype because it has sufficient torque to hold the solar panel and does not use much power itself. I also have experience using this particular servo motor which made it the logical choice to keep reduce development time. Another option could have been the SG92R, a servo with relatively similar specifications however a newer model but the upgrade is not necessary.

Figure : SG90 Micro Servo

The primary motor will be the servo motor detailed above for the reasons described. However there is an alternate, a stepper motor could be used instead. Stepper motors are similar to servos however they move in fixed increments (steps) and use a more open loop approach. As servos are operated in a closed loop way meaning they only have current supplied when they need to move, stepper motors are open loop meaning that current is always supplied therefore they will have a higher power draw and create more excess heat. Stepper motors also have less torque, this means a higher powered stepper motor would be required to do the same task as a lower powered servo alternative. The biggest reason for using a stepper motor instead is that they do not suffer the jittering that servo motors experience. If time allows a stepper motor in place of the servo motor will be investigated.

The module will contain a small TCP server running constantly but quietly in the background. This server will simply respond to requests from the measurement module for the current angle of the servos arm. It will send the angle and close the connection, nothing else. This will be running independently from the main program and as such if it is not working or if the measurement module is not working it should not affect the tracking modules ability to move the solar panel. This server is necessary to be able to get the data automatically from the tracking module. The current angle is an important bit of information and without you would not get the full picture regarding the performance of the solar panel. Again TCP was chosen over UDP because it does not need the extra speed UDP offers and the error protection TCP provides is more useful.

### The Measurement Module

This module handles all the measurements, as the name would suggests. It pools data from various sensors and the tracking module and transmits the data to the data module. This module does not store any of the data and instead is used to synchronise the bits of data from various sensors to ensure that the data readings are accurate for a particular given time. The code values it will measure are the current, voltage and the angle of the solar however this may be expanded on to include extra data such as the weather however it is not a priority. Using the current and voltage it is possible to calculate the power in Watts currently being generated by the solar panel.

To measure the voltage an MCP3008 analogue to digital converter will be used. It has eight input channels and a 10bit resolution. Eight channels is more than enough as only two will be used, one for the voltmeter and another for the ammeter. The specifications are partly why it has been chosen, the other reason is because researched showed that the MCP3008 is favoured by many developers, this leads to implication it is easy to use and that will be a lot of support available if necessary. Another factor was the price of device, currently £2.49 on eBay UK (20).

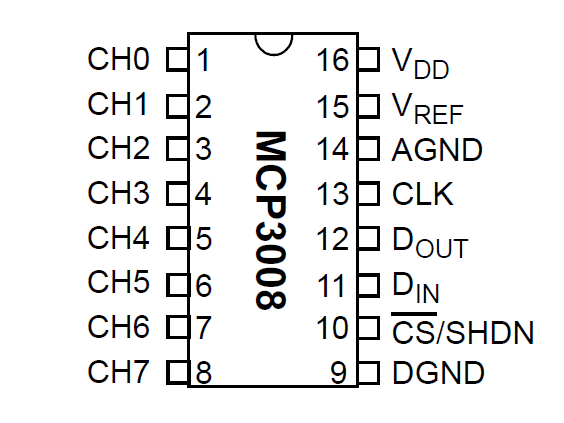


Figure : MCP3008 Pin Layout

The solar panel will be connect to CH0 (1)

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

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

DGND(9) connects to ground.

The code will be able to query the ADC using Serial Peripheral Interface (SPI). SPI is a synchronous interface bus that is primarily used in embedded systems to communicate with microcontrollers. Which is exactly what the project does, fortunately there is a library for python available on the Raspberry Pi called spidev to this exact task. The spidev library will be used to communicate with the ADC and to retrieve a value that represents the current voltage. As the ADC is 10 bits this means the maximum value will be 1023, if the ADC reads a value of 1023 this should be equal to 5V, the voltage reference connected to the ADC. Therefore to calculate the voltage the following equation is used.

The measurement module will open a socket to the data module and keep that socket open whilst continuously transferring the data as comma separated values. For example it will send the time, current angle, voltage, and current. The time will be a UTC timestamp in seconds, the angle is requested from the tracking module and the voltage/current are read from the ADC. If the connection to the data server is lost the measuring module will stop and wait for the user to fix the problem and restart the measuring module by entering a yes or no.

### Data Server Module

This module acts as a central location for all the data the system produces. The data module stores all the data generated by the measurement module. The main part of the module is TCP server at its heart that responds to requests for data and also responds to requests to store data. The HTTP server is main source of requests for data whilst the measurement module is constantly sending requests to store data.

A request from the HTTP server may look something like the follow GET V 10. The GET is telling the server it wants to get something, the V stands for voltage and the ten is how many it wants. So this particular request means get the last ten voltage records. The data server will then get the ten last voltage readings it received, convert them into JavaScript Object Notation(JSON) and transmit them back over the network the HTTP server. The reason to do it like this is because the request is simple and requires very little parsing, it is also quick and does not use much bandwidth. JSON has been chosen for the transmission to the HTTP server because JSON is widely supported and it will most likely be parsed by JavaScript in the user web browser. A possible alternative to JSON is XML, XML stands for extensible mark-up language and is useful for transmitting data over a network, however in this insistence because the data will be sent to JavaScript it makes more sense to use JSON.

The data server will be multithreaded so it can handle multiple incoming connections without blocking. It will also have a background thread for saving the data it receives to disk to keep more permanently. For saving the data to disk pythons pickle library (21) will be used. Pickling in python is a method of data serialization, the pickle function will convert a python object hierarchy into a byte stream, this means entire data structure such as lists that containing hundreds of classes can be written to disk. Naturally the pickle library also has a un-pickle function that will do the opposite and serialize data back into the python object hierarchy. Using this library will save a great deal of time writing complex methods of saving data and then parsing the saved data to load it again.

### HTTP Server Module

The HTTP Server will be a very customized and stripped down version of a standard web server. It will most likely not support most of the HTTP protocol other than the essentials such as GET and POST.

The HTTP servers main role is to display the data contained within the data module. It will do this by requesting data from the data module and displaying the data it receives as HTML in the users web browser, like a normal website. When the user tries to access the web page the web browser will send a HTTP GET request and the HTTP server should respond appropriately such that the web page loads normally and will be displaying the data. The biggest reason for displaying the data like this because almost, if not all computers have a web browser, therefore this eliminates a lot of problems surrounding an interface not working on one particular operating system. HTML and CSS are also very customizable so data can be presented in many different forms such as tables or just as plain text. Instead of doing the simple thing and using an existing HTTP such as Apache or Nginx a custom HTTP server will be developed because a custom server will for more control and more specific functionality. In particular the custom HTTP server will allow for the integration of python code whilst the web page is built, what this allows for is the embedding of sockets, sockets that will be used to open a connection to the data server so that the data can be requested.

## Module Design Revisions

The growing complexity of the system during development and the addition of new features not originally specified as led to some design modifications that will be detailed below.

The biggest change is the tracking module and measurement module have been merged together. This module is now known as the Solar Panel module. This was decided because both the tracking and measurement relate to the solar panel and in its current form the system could not deal with multiple solar panels sufficiently. This new module will allow of individual solar panels to be defined, the measurements can then be gotten and remain specific to that one solar panel. This is done through the implementation of a solar panel class. The solar panel class can choose from three different types of tracking, scanning, tracked or a fixed angle. It will load the details about each panel from csv file stored on disk.

In addition to this, a new method of transmitting the data to the data server was developed. Instead of the measurements being continually sent to the data server, the new panel information server will wait for a request from the data server and then gather the measurements through a solar panel class then transmit the readings back to the data server.

C:\Users\moorey\Downloads\top_level_revised.png

Figure : Revised Top Level Diagram

## Arduino UNO vs. Raspberry Pi

The tracking module and measuring module both interact with external components. The tracking module needs to manipulate the motor to move the solar panel and the measuring module needs to read data from various sensors.

This can be done with multiple devices but the main two are either an Arduino or Raspberry Pi. For the purpose of this project the Raspberry Pi has been chosen.

The Arduino is a microcontroller and does not feature a microprocessor like the Raspberry Pi. This means that Arduino is a more focused device, particularly focused more towards embedded devices which would be ideal for my the project. The Arduino also features fourteen GPIO pins including six capable of Pulse Width Modulation(PWM) for controlling servo motor, this is more than enough for the project. The Arduino is quite limiting in how code is written for it, to begin it has its own language, a variation on C, there is also not much choice in IDE as it has its own custom one. The Arduino is very bare bones so it doesn’t do a great deal, for example the standard Arduino does not have any network capability. This is not a real downside however because the Arduino has a great deal of expandability and if network access is needed an Ethernet shield can be purchased and fitted to the Arduino. Finally the potentially biggest problem is that because the Arduino is microcontroller it only has a limited amount of memory. The memory is so small that it’s possible that if the code is particularly long, it will not fit on the Arduino.

The Raspberry Pi does have a microprocessor, specifically an ARM1176, this gives the Pi greater computational power however at the expense of higher power usage. The Pi also has four USB ports, and an Ethernet port for network connectivity. It also has forty GPIO ports capable of interacting with motors and sensors. The main reason for using the Raspberry Pi is so the project can be developed using the Python programming language.

Programming the Arduino involves using a modified version of the C programming language and using a specific IDE to develop for the Arduino. There is nothing technically wrong with the Arduino’s programming language and it is very expansive with many libraries and with good support.

Whereas the microprocessor in the Pi allows you to use almost every common programming language including C++, Java and Python. As mentioned earlier the project will be developed using Python, this is the main selling point of the Raspberry Pi. Python is an open source, very widely used programming language and a very “easy” language to use. Therefore when problems inevitably occur, there will be many resources such as text books and internet forum to use for guidance. There is also a personal element to my decision in that I like using Python and have much more experience using it than the Arduinos language.

## 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 (22). 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” (23). 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 are a couple of problems with python, for example because its interpreted it is slow. Also because it is interpreted error are sometimes hidden because the error is only found when the code is executed. Python also 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.

# Chapter 4: Development

Development will be done in 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 things are still rough 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 is structured chronologically from what was developed first to last. Each heading will be tagged with the module that that section corresponds to, the tracking module with be TM, the measurement module will be MM, the data server module will be DM and the HTTP server will be HS.

## Equations (TM)

Development of the tracking module began by simply creating program that will calculate the various equations needed to produce the solar altitude. Specifically it began with calculating the solar declination, this was easier than expected. It was programmed it in python and happily it was discovered python has a lot of built in libraries to help, the most useful being the time library which provides a function to give the current day of the year as an integer (rather than a conventional date). Python also provides a lot of the mathematical functions such as sine, cosine and pi in its Math library.

Once each equation(See Chapter 2) had been implemented individually they were formed into functions that returned the appropriate variable. This meant that when putting each respective quantity into the solar altitude equation it was only necessary to call the specific function, this helped a great deal in keep the code clean and easy to follow.

The only major problem encountered after completing the solar altitude equation was it was quite a big problem as it was not getting the expected answer. This was different problem than the ones usually encountered in programming. When programming if something does not do you what you expect then there is usually a logic error somewhere in the code, something being somewhere it should not be, or in python a misspelled variable giving a null value. However it was in fact a mathematical error as the units where becoming mixed up. Some of the equations where being calculate in degrees and others in radians. Once this was correct the equations worked as intended.

## Raspberry Pi and Servo Motor (TM)

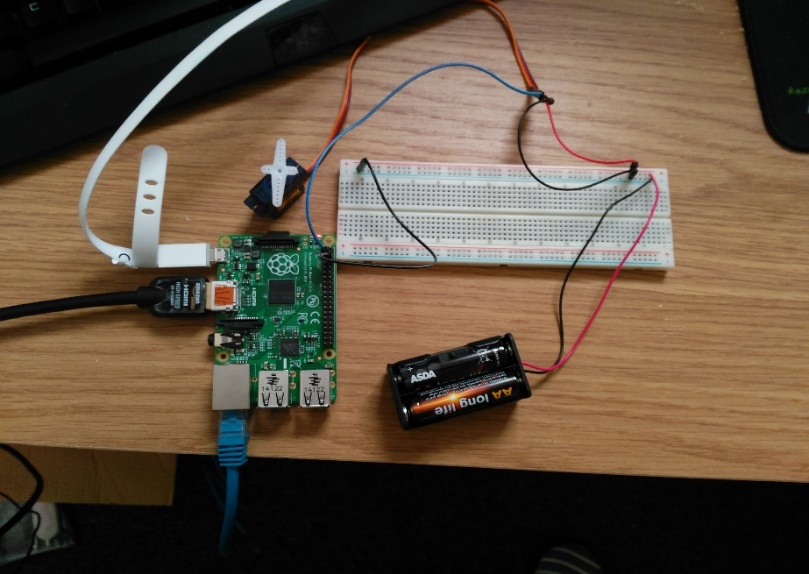
After completing the work with the equations for now, the focus moved onto looking at how to control a servo motor using the Raspberry Pi. Having done similar work with servos using and Arduino I already had some experience using servos. This was quite similar to the Arduino so did not take much time, it involved setting up the GPIO pins and then calculating the duty angle (how far the servo would move). To move the arm involved passing the duty cycle a function from pythons GPIO library.

Figure : Servo Circuit

To make moving the arm simpler a function to update the servo was created. The will take an angle and move the servo arm to that angle. The hardest part of this was trying to understand how the duty cycle corresponds with a particular angle, there are thankfully many internet sources available.

## Combining Equations with Servo Control (TM)

Now there are two independent programs, one to calculate the solar altitude (an angle) and one to position a servo arm at a particular angle. The amalgamation of both was therefore simple, the equations where already structured in functions so to position the servo arm at the current solar altitude all that need to be done was to pass the return value from the solar altitude function to the update servo arm function. This mostly worked however it was found that the servo arm was not moving the full angle, it was decided the problem was because the pulse width modulation was being stopped before the servo arm could fully rotate, I’m not sure if this is actually what was happening. The solution found for this was to add a delay of a couple of millisecond after moving the servo arm, whilst in most programs a delay would be detrimental, the servo control runs in a separate thread so a small delay is not going to be blocking any other code.

## ADC Voltmeter (MM)

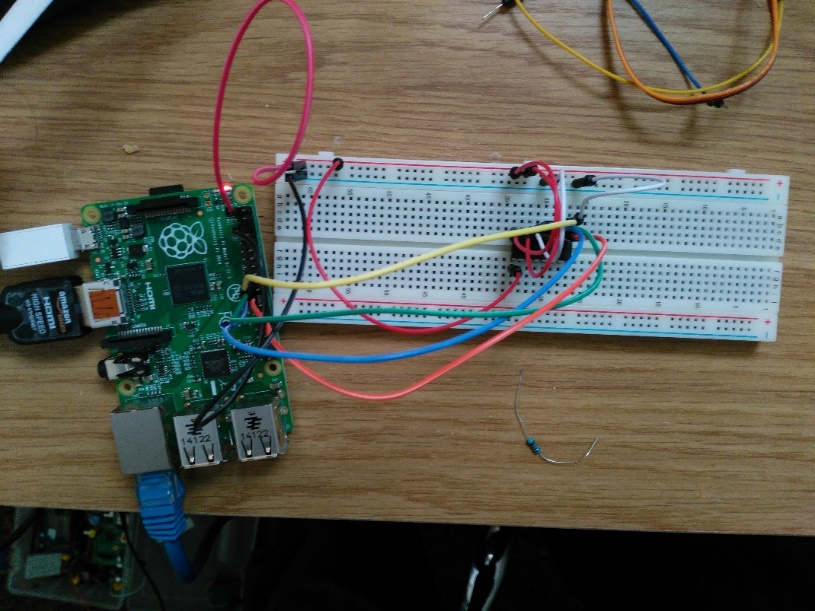
The base of the tracking module was complete so development moved onto the measurement module. The first area looked at was creating a voltmeter that could measure the potential difference being generated by the solar panel. To begin with however a solar panel was not available so I used the 3.3v pin on the Raspberry Pi and measured that. The spidev library provided by python was used to read a specific ADC channel, once the value was extracted from the ADC it had to be converted into an actual voltage. As discussed earlier the ADC is 10 bits and therefore will give an output between 0 and 1023, where 0 is 0v and 1023 in this case would be 3.3v (the maximum voltage provided by the 3.3v pin). An MCP3008 analogue to digital converter was used, it was mentioned that one of the reasons this ADC was chosen was because it seemed very common. This was very helpful because when development halted it was very easy to find solution through a multitude of online forms and tutorials.

Figure : MCP3008 Voltmeter Circuit

## HTTP Server Beginnings (HS)

The HTTP Server was easy to get started with as in a previous project a HTTP server had already been developed. The most important thing to do now was to actually remove certain function from the already developed HTTP server. A lot of the proper functions and error codes of a real HTTP simply aren’t necessary and if left in the program may introduce error and complications that can just be eliminated at the beginning. A very bland html index page was created to show it was working.

## Data Server Beginnings (DM)

The data server to begin with like the HTTP server didn’t require much work to begin with. At its basic the data server is simply a TCP server. A socket from pythons socket library was created. It was set it listen on port 15000 and then an infinite loop was set up to wait for connections. When the server gets a connection it simply stores whatever is sent to it in a list and sends a string “Received” back. A very basic server at this point but its core functionality (to store data) works.

## Voltmeter Server(MM)

The voltmeter at this point is simply reading the voltages from the ADC and then disregarding them, not very useful. So a way of transmitting the voltages to the data server for storage is needed. This is done using sockets and a custom “PUT” command. A socket connects to the data server, initially the data server was hosted at 192.168.0.250 on the local network on port 15000. The voltmeter running on the Raspberry Pi would connect to 192.168.0.250 through the socket and then send “put voltage” to the data server.

## Data Server Multithread and GET Query (DM)

To deal with the now constant stream of data, the data server was modified to spawn a new thread for each request it got. Now when the data server receives a put command from the measurement module, it will parse the put command and store the data into a python list. Python lists are natively thread safe so they were the optimal choice.

A get query was also developed. This GET query will be used by the HTTP server to request data from the data module. The data server parses this requests and identifies what data it should be returning and how much data. Once it’s worked this out it will pick the data of the list and send it back over the incoming socket (created on receiving the request).

## HTTP Server Custom GET (HS)

This is a not a conventional HTTP GET but something new and the main reason for creating a custom HTTP server. When you access the index page through the web server it will open a socket to the data server and request the latest live voltages, it does this using the custom GET request. Specifically a GET request could look like the following “GET A 10”, this means get the last ten values. The A stands for all meaning it wants every bit of data stored (not just the voltage or current). To begin with data server returns the data formatted like a python array, not very useful in practice but will be modified later. When it was first tested it was pleasantly surprising to see that it worked and that the values being recorded by the Raspberry Pi and transmitted to the data server, where being displayed in the web browser.

## Refactoring Voltmeter and Measurement Module (MM)

The project had progressed significantly and a rough version was beginning to shape up. Everything was working fine however the code was not in a great state and required tidying up. This is the side effects of doing things in pieces it sometimes gets messy. The solution to this was to take some extra time and go over the code moving re-usable bits into functions or classes and renaming variables. The biggest suspect for this was the voltmeter. To begin with it was not in a class and instead was a loose set of functions, this was remedied by creating Voltmeter.py and creating a class for the voltmeter code. A getVoltage() method was created to properly encapsulate the class, this improves the code immensely because it hides all of the spidev functions that handle getting values from the ADC and leaves just a very simple interface to the voltmeter class.

Doing this meant modification had to made to the rest of the measurement module. It was changed to use the Voltmeter class and to call getVoltage(). A try catch (try except in Python) was set up to stop the server from hanging when it couldn’t connect to the data server. Instead of just throwing errors and occasionally crashing when it can not connect to the data server, it now outputs a message indicating to the user the connection failed and gives them the option to attempt a reconnection. This feature was mostly added to help development at this stage, whenever the data server was shut down to restart it the measurement module would react badly, this was annoying from a development perspective and would be for the end user as well.

## Placeholder Ammeter (MM)

An ammeter was proving difficult to create, all the necessary equipment such as a current sensor was available however it was not clear how to place it in the circuit without distorting the voltage readings. It was decided too much time was being spent on it at the moment with little to no progress so a placeholder class was created in its place. This placeholder Ammeter class would use a random number generator to generated a current. This was place in a getCurrent() function so for all intents and purposes it could have been working perfectly fine like the Voltmeter. This placeholder class at the time was the right decision to make so that the project could progress further.

\*Special note from the author – I actually worked out why I was having difficulty with placing the current sensor in the circuit whilst I was re-wording this section.

## Angle Server (TM)

There was still no way of getting the current angle of the servo arm from the tracking module, after researching and contemplating multiple solutions it became clear the best way to do this was to set up a TCP server within the tracking module, to send the angle back to whomever requested . A Tracker class was created to handle the tracking and a getAngle() function that would return the current angle of arm, like with the Voltmeter class the getAngle function nicely hides away all the mathematics behind the class. When the tracking module is started a server now starts on port 12345 and then waits for connections, when a connection is received it returns the current angle from getAngle() and ignores what was sent to it.

## Formalising data (MM)

Up until now the measurement module has just been sending the voltage it reads from the ADC to the data module. One of the problems encountered is try solve how would to keep the data synchronised so that the user would know for sure each reading was done at the same time. This was mostly focused on the angle because it was coming from a separate module. The solution to this was to pool all the data in the measurement module and form a simple space separated string and send that string to the data module. This way it was possible to be sure each of readings would be relevant to one another and not at mismatched times. The transmission string is simply “time angle voltage current” the data server can work out the power itself using the voltage and current.

## HTML Index Page (HS)

The index page is hard coded into the HTTP Server so that it can directly get data from the data module and into the users web browser. This is done using a socket connection to the data module 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 Readings (HS)

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 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 (24). 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. It is far from working perfectly as the it will continue to get new readings without removing any of the old ones, this slows down the web page quite considerable and clutters up the page.

## Ammeter

It became whilst working on this report that wiring the current sensor into the circuit was simpler than first thought. It therefore did not take long to get the ACS712 current sensor integrated into the circuit, it was place between the solar panel and the ADC. To test that it was plugged in correctly the system was checked to make sure that the ADC was still reading accurate voltages, it was.

The code to get the data from the current sensor is similar to the voltmeter, the spidev library is used to read a channel on the ADC, the current sensor is wired to channel one (voltmeter is channel two). The current sensor sends a voltage to the ADC that corresponds to a particular amperage. The equation for this is below.

The 50 refers to 50A the maximum possible value, this is extrapolated from data given by the ACS712 current sensors datasheet whereby it gives 100mV equal to 1 Amp. The 5V comes from the power being supplied to the ACS712.

There is a very big problem that arises from this, the ACS712 is not capable of reading currents as low as what is needed. The solar panels being used generates 3V and a maximum of 100mA, the lowest value the ADC can read from the current sensor is 0.0488A (48.8mA) and the second lowest is 0.977A (97.7mA) this means there are only two values to represent the entire range of the solar panels power output.

This problem is specific to the prototype however as a larger more powerful solar panel will no doubt create higher currents that the sensor will be able to detect more accurately. However in the interest of wanting a working prototype a different current sensor will be requisitioned specifically a different version of ACS712 used with lower currents. An upgrade to the MCP3008 the current ADC being used has been purchased, the new ADC the MCP3208 features a 12 bit resolution. The 12 bit resolution will give between 0 and 4096 values that will hopefully increase the accuracy.

## Scrapping Ammeter

It was decided that the ammeter was more trouble that it was worth and the current reading wasn’t particularly useful without more complicated circuitry. The voltage reading would be sufficient to determine the power production capabilities of the solar panel at any given time. The ACS712 was removed the circuit however replacing with the MCP3008 with the MCP3208 would still be worth the upgrade and will still be done, though not a priority.

## Readings From Multiple ADC Channels

The ADC has eight channels to read analogue signals, realistically the system doesn’t need all eight however it definitely needs to use more than one. As a rudimentary implantation of this pythons argparse library was used to add command line arguments to the measurement module. When the python script is started the –np flag can be used to specify the number of panels used, this is quite restrictive because it assumes that the solar panels are not connected to arbitrary channels. If two solar panels are specified then the system will assume they are connected to channels 0 and 1. The biggest problem with this solution is that there is no way to specify whether a panel is being tracked and which channel the tracked panel is connected to, the only solution at the time for this is to designate channel 0 for a tracked panel, not optimal and will be refactored at a later time.

The data server was also modified to now accommodate the change so that it will store the multiple channels, this was simple because the extra readings can be concatenated onto the original, this was interesting as the code could easily adapt for the extra readings which was a pleasant surprise.

# Experiments

## Tracked vs Fixed Solar Panels

This kind of experiment has been done extensively for example as discussed earlier the work of Li and Kacira. They found unsurprisingly that a tracked solar panel performs better than its static counterpart. Though the result should be predictable it’s an experiment still worth doing, the result will show a certain degree of success to the project and an insight into how readily available cheap technology performs compared to the more specialised equipment used in many experiments at the turn of the century.

The experiment will be simple, two 3V solar panels will be used, one will be at a fixed angle of 60o and the other will use the solar altitude equation to calculate the suns position. Readings will be automatically taken every ten seconds and the tracker will update every thirty minutes. Ten seconds was chosen because it will give a lot of data to work and is not very taxing on the system. Thirty minutes was chosen for the tracker because the angle difference any less than that is very little and the servo motor controlling the position of the panel is not accurate enough to allow for such small increments.

The data will be viewed in real time and also saved into a database for further analysis later on.

It is important for this experiment that both panels are tested at the same times on the same day. Varying weather effects and cloud coverage will affect the readings and care will be taken to ensure that neither panel are covered by shadows.

Equipment used:-

2x Kitronik 3.0V 100mA Polycrystalline Solar Cell

2x SG90 Micro Servo

1x MCP3208 Analogue to Digital Converter

1x Raspberry Pi B+

## Tracked vs Fixed vs Scanning

Al-Mohammad et al performed a similar experiment to this in which they used photo-resistors to determine the light level, they found there was around a 40% in power generation during the early morning and evening. I believe it will be interesting to see how my relatively low tech and cheap closed loop tracking solution will compare and whether it will see the same 40% increase Al-Mohammad found almost twelve years ago in 2004.

In the experiment I will be performing the light level will be determined by the voltage being generated by the solar panel. The panel will be rotated through 100o every twenty minutes in 10o increments. Twenty minutes appears to be a good interval as performing the rotation too often will increase the power usage of the motor (decreasing overall power generation) and the solar panel may spend more time outside of what is optimum. 10o was chosen because its realistically the most stable movement the motor can perform quickly. The tracked panel will be set up to follow the sun using mathematical equations, the fixed panel will be kept at 60o for the duration of the experiment. The readings will be recorded every ten seconds for the same reasons as before and the results will be viewable in real time and stored in a database.

3x Kitronik 3.0V 100mA Polycrystalline Solar Cell

3x SG90 Micro Servo

1x MCP3208 Analogue to Digital Converter

1x Raspberry Pi B+

# References

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

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

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

4. *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.

5. *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 .

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

7. *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.

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

9. *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.

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

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

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

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

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

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

16. *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.

17. **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.

18. *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.

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

**20. 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.**

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

**22. 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.**

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

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

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