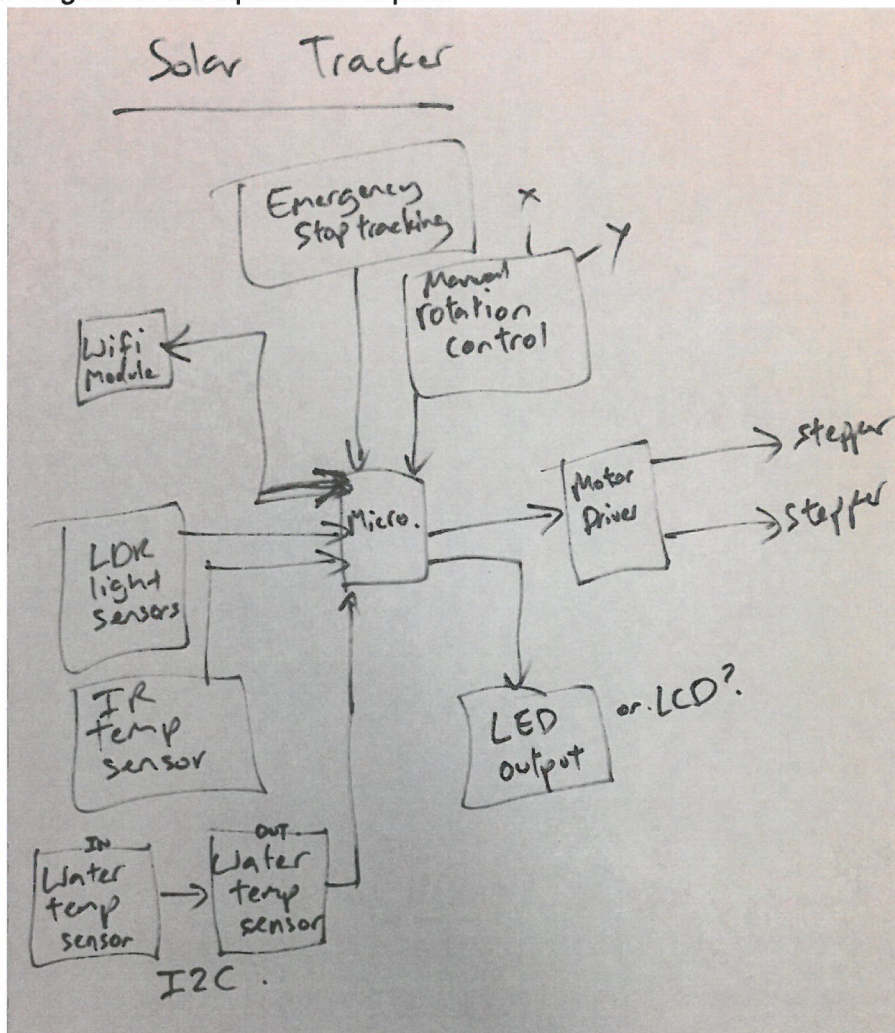




Complex Concepts

My project design is an automatic solar-thermal energy concentrator water heating system. It uses a Fresnel lens (a compact magnifying lens) to focus sunlight onto a water coil, generating heat to heat water to save power on water heating systems. A light sensor (using LDRs) detects the position of the sun and stepper motors drive the mechanism to keep the Fresnel lens perpendicular to the direction of the light coming from the sun.

Here is a diagram of the inputs and outputs:



NZQA Assessed

Complex Software Concepts

Structuring Complex Programs Logically:

Having a complex program structured logically is important because it needs to be easy to understand and alter.

```
16
17 // the setup function runs once when you press reset or power the board
18 void setup() {
19   // initialize digital pin 13 as an output.
20   pinMode(13, OUTPUT);
21 }
22
23 // the loop function runs over and over again forever
24 void loop() {
25   digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)
26   delay(1000);           // wait for a second
27   digitalWrite(13, LOW); // turn the LED off by making the voltage LOW
28   delay(1000);           // wait for a second
29 }
```

Above is an example of a basic program which includes various structuring techniques to help make it easy to follow and understand.

White space – White space helps divide the program into chunks allowing it to be readable and differentiable. Using white space in programs is important because having specific sections of code helps the reader to differentiate different parts of the code. In the example shown, the “setup” function is separated from the “loop” function by a band of white space. This is useful because you can easily see the difference between them whereas it would be more difficult if there wasn’t a gap in it. It would also make it difficult to fault find if there was no gap since there would just be one clump of code to sort through. White space also reduces strain on the reader’s eyes when sorting through the program as it makes it less visually intense.

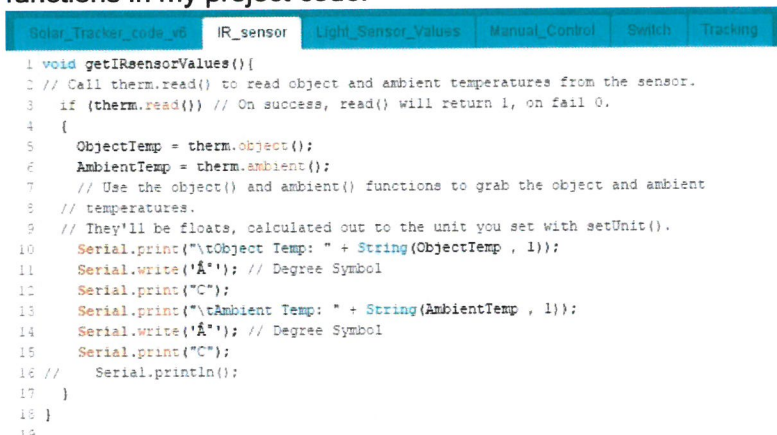
Commenting – Having comments next to the lines of code in the program is useful because having explanations for the programmer can help with remembering what they were writing and can help someone else understand the program. Comments can also be used with white space to section off areas of code which helps keep the program organised easily followed. If there were no comments in the program, the programmer can confuse themselves if they come back to the code or lose their train of thought. It would also be a problem if they were to get someone else to check the program since certain parts may not be straight-forward to understand.

Line Indents – Using line indents is useful to keep the program neat as they help with organising each line in a function. Here is an example of this:

```
41 void loop() { // read the sensor:
42   if (Serial.available() > 0) {
43     int a = Serial.read();
44
45     switch (a) {
46       case 'a':
47         digitalWrite(2, HIGH);
48         break;
49
50       case 'b':
51         digitalWrite(3, HIGH);
52         break;
53
54       case 'c':
55         digitalWrite(4, HIGH);
56         break;
57
58       case 'd':
```

This code shows the use of indents for each line under each certain statement. Having indents sorts the code into a form of sub headings which helps to distinguish which lines of code relate to one another. This is an advantage over having no indents because if there weren't any, the function would look continuous without sections which would make it difficult to understand and to fault find.

Using different tabs – The Arduino program has a useful feature that allows us to add different tabs which allows code to be separated. This can improve the organisation of the program and helps to make it more navigable and therefore easier to fault find. An example of using tabs is for storing functions. This is what I have done for all my functions in my project code:



```
1 void getIRsensorValues(){
2 // Call therm.read() to read object and ambient temperatures from the sensor.
3 if (therm.read()) // On success, read() will return 1, on fail 0.
4 {
5   ObjectTemp = therm.object();
6   AmbientTemp = therm.ambient();
7   // Use the object() and ambient() functions to grab the object and ambient
8   // temperatures.
9   // They'll be floats, calculated out to the unit you set with setUnit().
10  Serial.print("\tObject Temp: " + String(ObjectTemp , 1));
11  Serial.write('°'); // Degree Symbol
12  Serial.print("C");
13  Serial.print("\tAmbient Temp: " + String(AmbientTemp , 1));
14  Serial.write('°'); // Degree Symbol
15  Serial.print("C");
16 //   Serial.println();
17 }
18 }
19
```

Analogue to Digital Conversion:

An analogue signal can be converted to a digital signal using an ADC (Analogue to Digital Converter); it takes the analogue voltage value of a pin and converts it to a digital value. The Arduino Uno microcontroller is capable of this although not all the pins are able to.

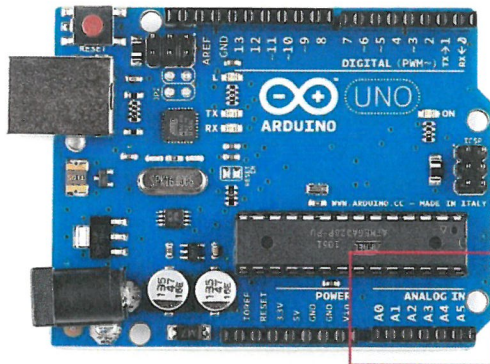


Figure 1: Arduino Uno from <https://learn.sparkfun.com>

These are the 6 'analog in' pins from A0 to A5. In the Arduino program, the analogue values of these pins can be found by using the 'analogRead' function. The ADC in the Arduino Uno is a 10-bit ADC so it can measure 1024 (2^{10}) discrete analogue levels, this is why the 'analogRead' function picks up values between 0 and 1023. ADCs are complex and there are many different methods to achieve analogue to digital conversion but the most common method is used in the Arduino. It uses the analogue voltage to charge up an internal capacitor and

measures the time it takes for it to discharge across an internal resistor. The microcontroller measures time by monitoring how many clock cycles that pass before the capacitor discharges and this number is that value that gets returned once the ADC is complete.

My project uses the Arduino ADC when returning the values of my LDR light sensor circuits to then determine where the solar tracker should be aiming. Here is an example of this:

```
Light_Sensor$
34   for (int thisXReading = 0; thisXReading < XnumReadings; thisXReading++) {
35     Xreadings[thisXReading] = 0;
36   }
37   Serial.begin(9600);
38 }
39
40 void loop() {
41   // subtract the last reading:
42   Xtotal = Xtotal - Xreadings[XreadIndex];
43   // read from the sensor:
44   Xreadings[XreadIndex] = analogRead(lightSensorXpin);
45   // add the reading to the total:
46   Xtotal = Xtotal + Xreadings[XreadIndex];
47   // advance to the next position in the array:
48   XreadIndex = XreadIndex + 1;
49
50   // if we're at the end of the array...
51   if (XreadIndex >= XnumReadings) {
52     // ...wrap around to the beginning:
53     XreadIndex = 0;
54   }
55
56   // calculate the average:
57   Xaverage = Xtotal / XnumReadings;
58   // send it to the computer as ASCII digits
59   Serial.print("X Light value = ");
60   Serial.print(Xaverage);
61   delay(1); // delay in between reads for stability

```

X Light value = 428	Y Light value = 487
X Light value = 430	Y Light value = 488
X Light value = 431	Y Light value = 488
X Light value = 432	Y Light value = 488
X Light value = 432	Y Light value = 488
X Light value = 432	Y Light value = 488
X Light value = 433	Y Light value = 489
X Light value = 433	Y Light value = 489
X Light value = 434	Y Light value = 489
X Light value = 434	Y Light value = 488
X Light value = 434	Y Light value = 489
X Light value = 435	Y Light value = 488
X Light value = 436	Y Light value = 487
X Light value = 436	Y Light value = 487
X Light value = 437	Y Light value = 487
X Light value = 437	Y Light value = 487
X Light value = 438	Y Light value = 487
X Light value = 438	Y Light value = 487
X Light value = 439	Y Light value = 487
X Light value = 440	Y Light value = 487
X Light value = 440	Y Light value = 487
X Light value = 440	Y Light value = 488
X Light value = 440	Y Light value = 488
X Light value = 440	Y Light value = 488
X Light value = 440	Y Light value = 488
X Light value = 440	Y Light value = 489
X Light value = 440	Y Light value = 489
X Light value = 441	Y Light value = 489

NZQA Assessed

This code shows how my program reads the X-axis light sensor values. It uses the 'analogRead' function to read the 'lightSensorXpin' which is the variable set for the

Arduino analog input pin 'A0'. It also features a data smoothing algorithm to help eliminate random data values that may be picked up and to improve the accuracy of the data. On the right is a screenshot of the serial monitor which is what we print on to see how the microcontroller is communicating with the computer.

Serial and Parallel Communication:

Electronic systems are all about linking circuits together such as processors and other integrated circuits. This involves exchanging information between these circuits and to do this they must share a common communication method. Hundreds of methods of communication have been devised but they all fall into two categories; serial or parallel.

Parallel vs. Serial:

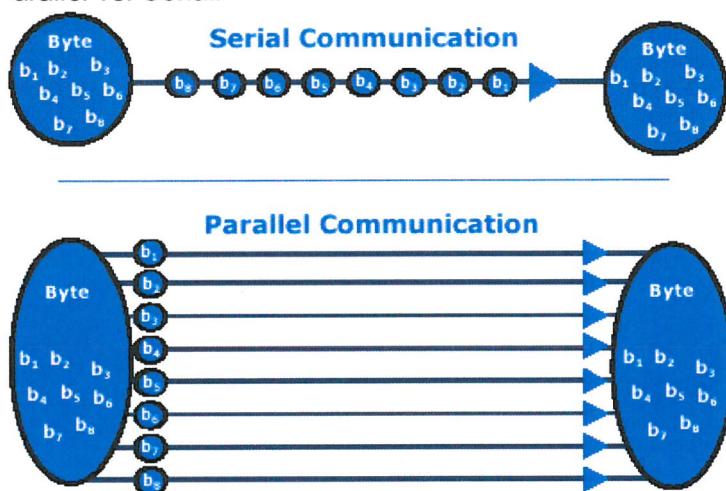


Figure 2: Parallel vs serial communication from <http://maxembedded.com/2013/09/serial-communication-introduction/>

Parallel interfaces send and receive data in multiple bits at a time through 8, 16 or more channels (e.g. wires, printed circuit board tracks, optical fibres) at once. The data is transmitted/received in several streams of data. The main advantage of parallel communication is that it can be much faster than serial since it can send a byte of data in the same time it takes for a serial communication system to send one bit of data. Although the speed is favourable, parallel communication is not often used in long distance applications due to how it needs a wire for every channel which becomes very expensive and impractical. It is also designed for low voltage applications like in computers as the signal wires only need to span over a very small distance and the high speed is needed.

Serial communication interface systems commonly only use two wires although they can use as little as one wire but never more than four. They can also wirelessly transmit

and receive streams of data. The two wires are for the data channel and the clock channel. The serial interfaces stream their data one bit per clock cycle along the data channel. There is a vast spectrum of serial protocols designed to fit the needs of various embedded systems. The two more prevalent types of serial interfaces used in computing are Ethernet and the USB (universal serial bus). Other common serial interfaces include I²C (a multi-master bus which allows multiple chips to be connected to the same bus) and SPI (a serial peripheral interface that enables communication between two devices) but all which fall under two categories which are 'synchronous and asynchronous'.

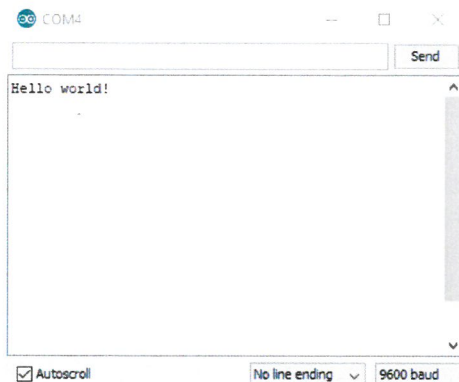
A synchronous serial interface pairs its data channels with a common clock signal. This allows other devices to be hooked into the same clock which ensures a smooth, reliable, and often faster serial transfer. Consequently, at least one extra wire is needed between the communicating devices. Examples of this include I²C and SPI.

An asynchronous serial interface is a form of serial communication which does not use a clock signal. This method is useful for minimising in/out pins and wires, but this means that we need to put extra work into transferring and receiving data reliably.

Arduino boards communicate asynchronously with computers through USB. The speed of the clock signal is set by the user with the 'Serial.begin' function. For example, the Arduino Uno microcontroller communicates at a 9600 baud rate which means a maximum of 9600 bits per second. Here is a bare basic example of serial communication between a computer and an Arduino board:

```
1 // Simple Serial Echo script - Modified by Scott 08/07/2012
2
3 // Use a variable called byteRead to temporarily store
4 // the data coming from the computer
5 byte byteRead;
6
7 void setup() {
8 // Turn the Serial Protocol ON
9 Serial.begin(9600);
10 }
11
12 void loop() {
13 // check if data has been sent from the computer
14 if (Serial.available()) {
15 // read the most recent byte
16 byteRead = Serial.read();
17 // Echo the value that was read, back to the serial port
18 Serial.write(byteRead);
19 }
20 }
```

This code reads what is sent into the serial monitor and then prints it.



When I type “Hello world!” into the box at the top and hit ‘send’, the message is sent to the Arduino board where the programme reads the characters and then writes back into the serial what was typed. This is the basics of how the Arduino board and the computer communicate.

Counters

Counters are variables that increase (or decrease) when certain events happen. They are usually used when the counter has reached a certain number to then trigger an event and then reset the counter.

I will be using counters to count the steps of the stepper motors in my solar tracking project so that I don’t need any feedback position sensors for the movements of the mechanism while tracking the sun. They will be to set limits for how far the mechanism moves, so the system knows where the mechanism is in physical space, and to add in an ‘emergency stop focusing’ position where the mechanism is no longer perpendicular to the sun and therefore the Fresnel lens won’t be able to focus the sunlight into a point. Since this eliminates the need of feedback position sensors, there is now more pins left in the microcontroller for other components.

Complex Hardware Concepts

Microcontroller

A microcontroller is a small computer which is often a single integrated circuit chip containing programmable input and output pins, one or more CPU’s, programmable ROM (Read Only Memory) and RAM (Random Access Memory). Microcontrollers are applicable in power tools, appliances, toys, engine control systems remote controls and other embedded systems. Much like microprocessors (found in computers), microcontrollers execute instructions in an ordered manner with the speed of which depending on the microcontroller’s internal oscillator (clock). These instructions are stored in the micro’s ROM.

A microcontroller is needed in my solar tracking system as I have multiple inputs and outputs which need controlling and to have a 'brain' of the system. The features I need from the microcontroller include the low cost, multiple input and output pins, ADC and DAC for the sensors (LDRs, joystick, thermometers, and infrared thermometer).

There are many different families of microcontrollers such as 8051, AVR, Arduino, ARM, and PIC. Each family and each chip have their own benefits such as cost, availability, number of pins, amount of memory, clock speed, etc. which affect their capabilities and applications. For example, a system with a large number of components will require a large number of pins. In this section I will be covering Arduino microcontrollers. They are cheap, very accessible, and I'm most familiar with them as I have been using them for a few years.

A typical example of an Arduino board is the Arduino Uno. It has 28 pins in total and is controlled by an ATmega328 microcontroller chip. Here is a diagram highlighting the most important features on the Arduino Uno:

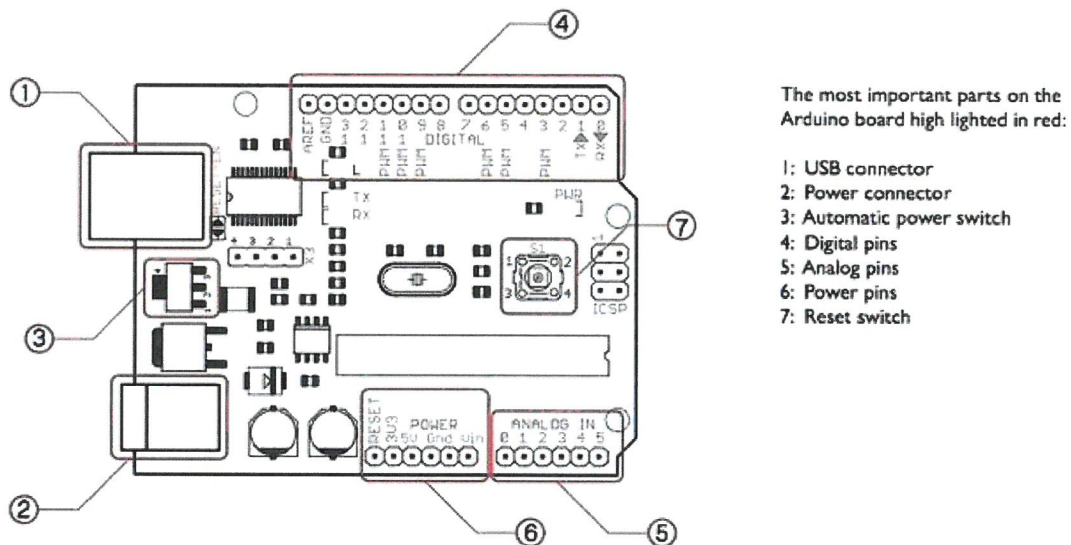


Figure 3: Arduino Uno <http://www.edgefxkits.com/blog/arduino-technology-architecture-and-applications/>

The Uno board consists of 14 digital input/output pins and 6 of which have the capability of pulse width modulation. Among these, some pins have special functions. Pin 0 and 1 can be used as transmitter and receiver channels which are used for serial communication, pins 2 and 3 can be used as external interrupts, pin 13 activates an on-board LED and pins 3,5,6,9,11 are the pulse width modulation output pins. The Uno

also has 6 analogue input pins which connect to a 10-bit ADC. This means it can map these analogue voltages from 0-5V and return a value between 0 and 1023.

The memory of ATmega328 consists of two different memories; program memory and data memory. It stores the code in the flash program memory and the data in the data memory. It runs at a 16MHz clock speed and contains 3 types of memory:

- 32kb of flash memory – This is where the program is stored. It cannot be modified by executing code, to do this it would need to be stored in the SRAM.
- 2kb of SRAM (Static Random-Access Memory – This can be read and written from executing the program and is used to reserve memory for variables, functions, etc. which remain in the memory for the duration that the system is powered.
- 1kb of EPROM (Erasable Programmable Read-Only Memory) – This is another form of memory which can be read and written from executing the program and typically used for the task of secondary storage.

This is a great microcontroller to use for my project as it has an adequate amount of room for my program as it will be running for most of the day at a time.

Infrared Thermometers:

All materials with a temperature above absolute zero have moving molecules, therefore they all emit infrared radiation. Infrared radiation is invisible to the human eye as the range is between the visible and radio sections in the electromagnetic radiation spectrum. As materials get hotter, the radiation emitted from them increases as the molecules become more excited and vibrate more vigorously. This is why metal turns red when it gets to a certain temperature; it begins emitting red light because it's pushing past the infrared range into the visible range of electromagnetic radiation, where red is the colour with the lowest frequency.

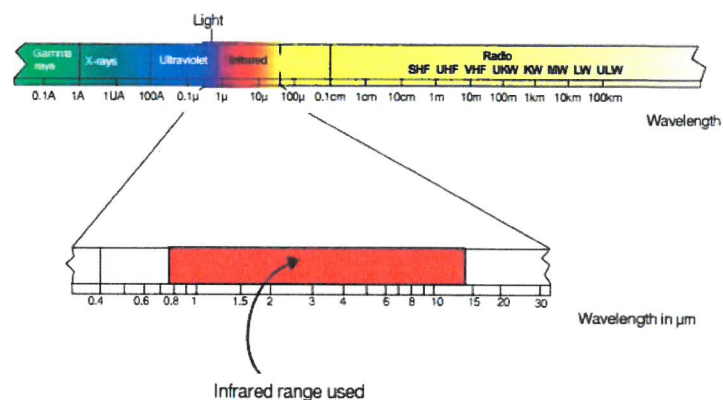


Figure 4: Electromagnetic Spectrum. From <http://www.apiste-global.com>

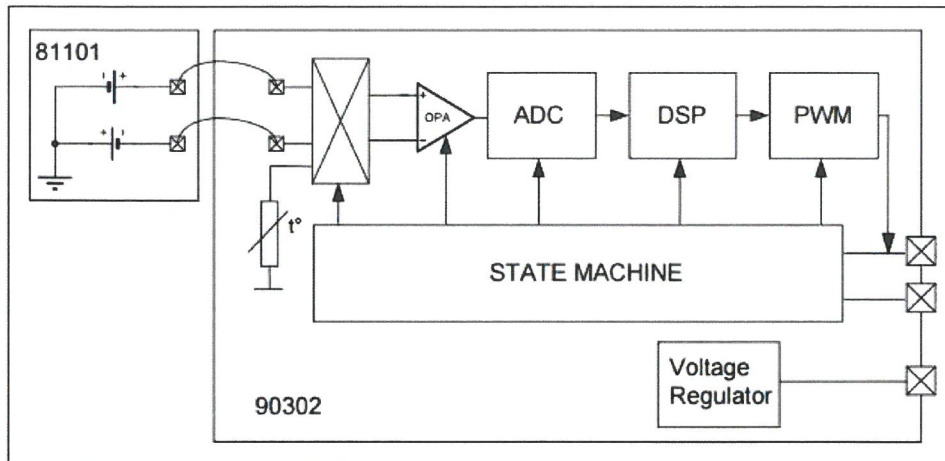


Figure 5: MLX90614 Infrared Thermal Sensor System Diagram. From: <https://www.sparkfun.com>

In my project, I'm going to be using an infrared MLX90614 contactless thermal sensor to measure the temperature of the surface which the Fresnel lens heats.

The operation of this IR sensor is controlled by an internal state machine, which is made up of sensor, an op-amp, an ADC (Analogue to Digital Converter), DSP (Digital Signal Processor) and a PWM (Pulse Width Modulation) state machine.

The output of the IR sensor is amplified and converted through the ADC into a single bit stream. The ADC is called a delta-sigma modulator in this case, which takes the analogue signal and converts it into a digital bit-stream. Here, the signal passes through IIR (Infinite Impulse Response) and FIR (Finite Impulse Response) low pass filters to gain the average signal level out of the bit-stream and to reduce the bandwidth of the input signal so that the desired noise performance and refresh rate can be achieved. It then passes through the DSP for further processing. It is then transmitted by a 10-bit PWM with a programmable range. In this case, the range of the ambient temperature function is -20 to 120 °C with an output resolution of 0.14 °C although both the calculated ambient and object temperature measurements have a resolution of 0.1 °C. These specific resolutions are mathematically found by dividing the value of the range of measurement by the amount of 'bits'. So, for example, the ambient temperature resolution is found by dividing 140°C by 1024 which gives a value of 0.1367°C ... and rounds to 0.14°C although you can decide how many decimal places want in the program.

This infrared sensor will be a useful component in my project as it will be used as a contactless thermometer to measure the temperature of the surface which the Fresnel lens will be heating. The advantage of using a contactless thermometer is that it is

unlikely to be damaged from the heat output of the lens since it can be held away from the object.

H-Bridge

An H-bridge is a complex electronic circuit which allows a voltage to be applied across a component in two different directions. H-bridges are most commonly used in motor controllers and therefore robotics as they allow a sufficient power input to motors and can control the speed and direction of rotation of DC motors. They are constructed either using separate transistors or integrated in a chip.

The most common examples of using an H-bridge on a chip are included in the L293 and L298. Each of these chips have two H-bridges therefore two motors can be controlled from one chip, or one stepper motor. The L298 motor controller chip has the capability to sense current which can be used to detect motor overload.

The H-bridge is used in my solar tracking project context to control the two stepper motors. Stepper motors need two separate pairs of power inputs so that both coils in the motor can be activated in both directions to set up the correct magnetic fields. This means that each stepper motor will ideally use their own dual H-bridge motor driver chip.

L293D

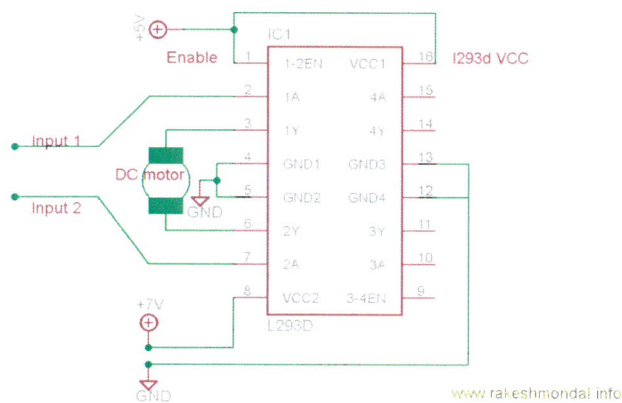


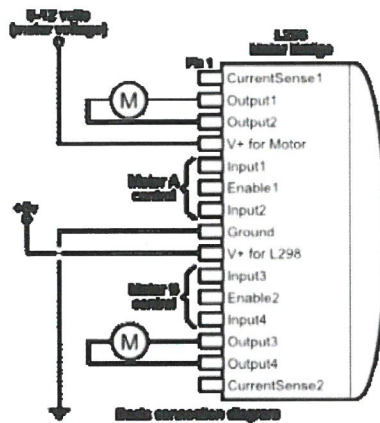
Figure 6: : L2983D schematic from:
<http://www.rakeshmondal.info/L293D-Motor-Driver>

EMF, this is what the 'D' stands for in L293D. I will detail back-EMF and 'flyback' diodes below. This is a useful feature to have in a motor driver since it cuts back on components as the diodes would otherwise need to be externally connected to the

The L293D is a 16 pin IC (Integrated circuit) chip which is commonly used to drive small DC motors. It requires one 5V input (VCC) for the chip to operate and an external supply voltage (VSS) to supply sufficient power to the motor(s). 36V and 600mA are the maximum values the chip can handle although when reaching these, a large heatsink may be required to dissipate the heat. This model of L293 has integrated 'flyback' diodes to deal with back-

circuit. Since the L293D can only supply a maximum of 600mA, it won't be adequate to use for my stepper motors.

L298

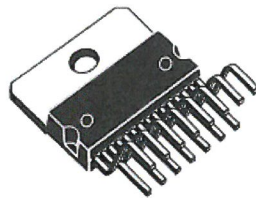


This motor driver chip has 15 pins including two current sensing pins which can be used to detect when a motor is being overloaded. The L298 also requires a 5V for it to operate and an external supply voltage for the motors. It can handle up to 4 amperes and 46V which is ideal for my stepper motors as they draw approximately 3 amperes.

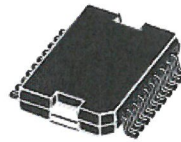
Figure 7: L298 pin diagram from <http://www.robotoid.com/appnotes/circuits-l298-hbridge.html>

When operating at currents of this magnitude, a large amount of heat is dissipated.

The L298 exists in two different packages; the Multiwatt15 and the PowerSO20.



Multiwatt15



PowerSO20

Figure 8: L298 packages
<http://www.electroschematics.com/8869/l298->

The Multiwatt15 is configured vertically with offset pins and a lug which is useful for attaching a heatsink of any size to further dissipate heat. It can dissipate 25W of heat without a heatsink whereas the PowerSO20 is only capable of 5W. The PowerSO20 is configured horizontally and can have a heatsink attached but isn't as effective.

These L298 chips require flyback diodes to be included in the circuit to protect the chip against the back EMF (electromotive force) induced by the coils in the stepper motors. The coils in the stepper motors are inductors and inductors oppose changes in current. When an inductor is fully energized, the flowing current causes it to induce a magnetic field. When the current supply is switched off, the inductor will try to resist this change in current by using its

energy from the magnetic field to form an opposing voltage. This causes an extremely large potential difference; a large negative potential will be created where there was a large positive potential and vice versa. Since this potential difference is so high, it could arc across a

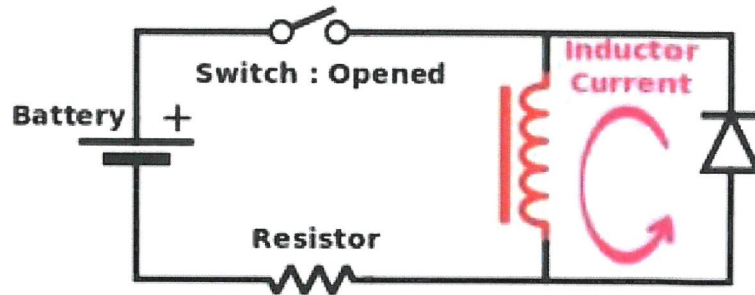


Figure 9 Inductor example from <https://www.westfloridacomponents.com/blog/what-is-back-emf-and-what-does-it-do/>

switch or could cause a power surge through the whole circuit. Flyback diodes can be wired into the circuit across the inductor from the negative terminal to the positive terminal (the opposite direction of which the current was initially flowing) to protect the circuit. This means the diode is allowing the back EMF to continuously cycle back through the inductor allowing it to draw current from itself until the energy is dissipated through losses in the wires. Here is a diagram of how they are wired in:

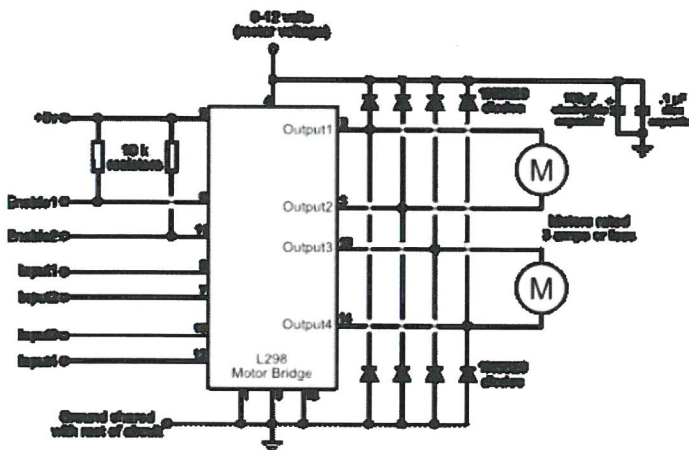


Figure 10: L298 wiring diagram from <http://www.robotoid.com/appnotes/circuits-l298-hbridge.html>

This diagram displays two motors as the outputs whereas when using a stepper motor, these outputs will be the two coils. The current in the coils of the stepper motors are constantly changing so flyback diodes are vital in this situation.

NZQA Assessed

Stepper Motors

Stepper motors are brushless DC motors that move in discrete steps. They are made up of multiple coils which are organised in groups called 'phases'. Stepper motors very precise in their movements due to the small steps, and the accuracy can be further improved by gearing them down. They are commonly used in CNC machinery and robotics for these features. Since stepper motors move step by step, they can be programmed to set themselves at certain positions without the need for position sensors or feedback.

I decided to use stepper motors in my project for the reasons above. The sun moves across the sky at only approximately 15° per hour so my solar tracking mechanism only needs to move accordingly.

Unipolar and Bipolar

There is a vast variety of stepper motors and they fall under two categories; unipolar and bipolar.

The unipolar steppers always power their phases in the same direction; one common lead per phase will always be negative and one common lead per phase will always be positive. This means that for each step, a separate coil needs to be energized and they can be controlled with simple transistor circuitry. This results in unipolar stepper motors generally having more phases than bipolar. The disadvantage of unipolar is that only half of the coils can be energized at a time so they have less available torque than they could have. Here is a diagram of a 4 phase unipolar demonstrating how it steps:

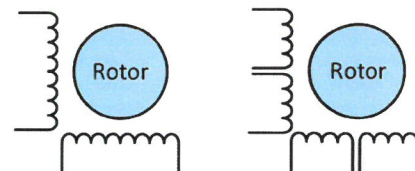


Figure 11: Bipolar and unipolar stepper motors from <https://learn.adafruit.com/all-about-stepper-motors/types-of-steppers>

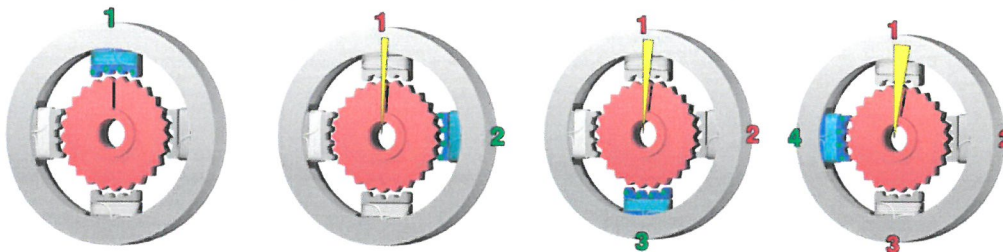


Figure 12: Captured GIF of unipolar stepper from <https://en.wikipedia.org/w/index.php?title=File:StepperMotor.gif>

Bipolar stepper motors commonly use fewer coils as they can reverse their magnetic field directions. The coils have two wires each and have no common leads. The advantage of bipolar stepper motors is that each coil is fully energized when they are activated which increases the output torque. The disadvantage is how the control needs to be more complex using H-bridges, which is only a minor setback as they are so accessible.