**Progress Report**

**Introduction**

This project brings together the work I have done from year one, the making of a temperature sensor, and year two, the making of a web hosted video camera, some elements of year three and some external content too. Doing this project furthers my understanding of the work that I have done throughout my university career.

When hobbyists or industrial brewers of alcohol brew temperature monitoring, especially for hobby brewers, is done manually via dipping a thermometer into the liquid mixture. While this works, I hope to replace this manual method with something more up to date by using a microcontroller and a single board computer to make this process automated. Also, to help brewers know what to do when I will incorporate a timing system that will deliver email-based alerts so that they know when to take the next appropriate actions.

**Aim and Objectives**

The main aim of this project is to produce a system that can be used to monitor the brewing process remotely from a web page via a temperature sensor and a web enabled camera.

Objectives for this project are:

1. Working temperature acquisition system
2. Raspberry Pi setup for data acquisition from the Arduino system
3. Completed automatically updating web page and integrated web camera
4. Completed automatically updating web page with video feed
5. Completed web page is hosted locally
6. Completed web page with a working web server solution and port forwarding
7. To create a light source for the camera that will automatically switch on in dark environments.
8. To create an email-based update system to notify the user of impending changes that need to be made or problems that need solving time permitting an SMS system could be implemented also.
9. Time permitting a relay and a heating element could be added to be able to fully

**Approach** **&** **methodology**

The main approach to the first steps of this project have been to use work that I have done in previous years that has been glued together throughout the early build stages. I initially rebuilt a basic temperature logger with an Arduino Pro mini and then interfaced it through UART to a Raspberry Pi I then rebuilt the web hosted camera from my previous year of study. There were no real holdups or major problems.

The more recent approach has been based on research and learning making recent progress inherently slower. This new approach to building has left me with less time than initially anticipated however with early quick successes I have been ahead of schedule and this has left me with plenty of time to complete the ahead tasks. There have been some new tasks generated with the research these new tasks are replacing the current web server solution with a new customised solution written in node.js, a new solution for getting the data from the serial port to the web page written in JavaScript and the led lighting system will also be written in JavaScript and all of this will be controlled by the webserver which will run on boot.

**Literature Survey/Theory** (up to 10 pages)

The Project will be constructed using an embedded system and a single board computer. For my choice of programmable microcontroller for the base for my embedded I went for a prebuilt solution on a premade PCB as these are generally similar in price to the chips on their own, furthermore I found that this approach cut large chunks of time off the build process and allowed me to focus on building my project.

Initially looking at the specifications for two major contenders, the STM32F103C8T6 () would be a suitable development platform as it has one of the better clock speeds and better memory capacities. Any of the boards that I chose could be directly soldered too as to reduce the form factor of the probe. Direct soldering means I can determine the length of cable the microcontroller has from the Raspberry Pi. For this project however, I have decided to use the Arduino pro mini a 3.3v () as it is a low power low cost option and it is a platform that I am most familiar with and there is a good set of libraries available on the Arduino playground for reading the input from the DS18B20 and can easily be interfaced with the raspberry pi. I am using the Arduino Pro Mini board over Arduino Uno () as it is physically smaller cheaper and runs on 3.3v whereas the Arduino Uno runs on 5v the bad things about the Arduino Pro Mini are that it has a fixed clock speed of 8MHz however for my prototype uses this is not an issue as I only need it to act as a temperature sensor.

I Won’t be discussing the “just add water kits” such as the beer buddy kit (Young, n.d) as these kits don’t conform to what my project is designed to do. However, while these kits do require some temperature regulation and my project could be used to monitor or regulate the temperature of these brews while they ferment, I won’t be discussing them as they are not the intended end target.

Starter kits come with all the needed parts to start brewing alcohol, such as the full equipment style of kits from the Home Brew Shop (The Brew Home Shop, 2019) but do not come with anything other than a manual way for measuring the temperature and kind of temperature regulation is absent this is where my project will come in and replace this manual method of temperature.

Having done research about the languages that I’m using and I have come to the conclusion that using python for the update code is a bad choice as it overcomplicates the update system and adds an unnecessary language group.

* DS18B20 bought for £2.45 on ebay.co.uk
* Pro Mini 8MHz 3.3v / 5v MEGA328P bought on ebay.co.uk for £1.50
  + Fixed to 8MHz has variable 3.3v / 5v input supply

**Progress Made**

In order to start this project, I needed to connect everything to the Arduino pro mini’s pins this meant soldering a six-pin male header to the pins at the end of the board. After individually twisting the wires and applying a little solder to them the DS18B20 temperature sensor’s power, ground and data wires were soldered to the VCC ground and A3 analogue data pins on the Arduino pro mini. During this process a 4 kilo-Ohm resistor was omitted this initially caused bad results and was fixed by soldering two, two kilo-Ohm resistors in series across the VCC and data wires this was done instead of adding a four kilo-Ohm resistor due to there not being one to hand at the time. Having attempted to write the code for the Arduino several times and initially not realising that I needed to use two libraries the OneWire and DallasTemperature libraries. During early iterations of the design of the code I used the Delay() function to control the timings of the program however, as the design became more sophisticated and more features were added this function became difficult to use and was replaced with a new method of timing control. The reason that the Delay() function was replaced was due to the nature of its working, as the delay function counts out its allotted time it holds up the entire microcontroller, making timing for multiple loops of the same program incredibly difficult or impossible. I then created a state machine to separate the tasks the Arduino had to complete and make timing these tasks easier. I began using the millis() function, the millis() function does not use the whole microcontroller like the Delay() function does, as a basis of an if statement to control the intervals at which individual tasks happen. Once the working code was uploaded to the Arduino, I set about setting up the Raspberry pi. This required using 2 pieces of software that can take some time to run up to five to ten minutes. As the Raspberry Pi 3 runs its operating system from a micro SD card I decided to use a 32GB card this is far more space than will ever be required by my project however, it means that I can avoid worrying about running out of space during any of my build stage. Having chosen an SD card, I then wiped it using a program called SD card formatter (SD Association, 2018) fully wiping the card like this removes any data on the card and sets all the space into one partition so that when we write the Raspbian disk image (Raspberry Pi Foundation, 2018) to the SD card there won’t be any problems. Having written the operating system to the SD card I then setup the Raspberry Pi for first time boot and after booting and logging in with the username pi and the password raspberry I set about setting up the operating system for my project the first commands that should always be run when logging into the Raspberry Pi are the sudo apt-get update and the sudo apt-get dist-upgrade -y commands the first of these commands updates the Raspberry Pi’s package lists to the latest version this allows the Raspberry Pi to find the latest packages and their updates if there are any the second of these commands updates the out of date packages present on the system. I then set about uninstalling unnecessary applications such as the office suite that comes with Raspbian, I did these in batches in order to minimise error and keep the processing time per command low. I ran the following uninstall commands to uninstall the following programs and remove their config files as I won’t be using them and they’re cluttering up the operating system the final command that I ran out of these three was to remove and dependency packages that were now on the system and were redundant. The following three bullet points are the commands that I used in the console.

* sudo apt-get purge minecraft-pi wolfram-engine scratch -y
* sudo apt-get scratch2 libreeoffice\*
* sudo apt-get autoremove

The next step was to set the Raspberry Pi up for my project and there were two pieces of software that I wanted to install Apache2 webserver software (The Apache Software Foundation, 1997) and Motion4.0 (Motion Project, 2018) a security camera streaming software. I also set the Raspberry Pi’s local IP to be static with help from a guide on the internet (ModMyPi LTD, 19 April 2016) as this greatly helps during and port forwarding and any testing that I wanted to do. The following two bullet points are the install commands that I used in the console.

* sudo apt-get install apache2
* sudo apt-get install motion

After installing both of these I performed some first time use tests by opening a browser and using the Raspberry Pi’s local IP address in the address bar, in my case 192.168.1.128, this brought up the Apache2 welcome page that I will be changing out later. With Apache2 working I turned my attention to Motion having set up Motion before I knew there were a couple of things I needed to do the first of which is to tell the Raspberry Pi that to use the camera module that I own so I opened the etc/modules file and added this the camera that I was using at the end of that file I then saved my changes and closed the file. I then changed to editing the config file for motion and then changed the daemon setting from off to on this allows motion to start on boot and run in the background. The next setting, I needed to change was the resolution of the camera that I was using from the default to 768 wide by 1024 high next I changed the framerate from 1 to 60 this is so that the viewer of the stream gets a clear and smooth video feed. One of the major features of motion is its ability to capture images of things when they move, as part of its security side of the package, as I don’t want this to happen due to me not wanting the SD card to run out of space, I switched this setting off in the config file. In order to have the stream hosted within the local network I need to have a it hosted on an open port and have the web control setup on an open port too for the stream I selected port 8081 and for the web control I selected 8082 I then changed the local host setting from off to on saved and exited the config file and restarted motion using sudo systemctl restart motion. The following bullet points show the changes made in to the config files talked about above.

* sudo nano /etc/modules
  + adding “bcm2835-v4l2” to the end of the file
* sudo nano /etc/motion/motion.conf
  + changing daemon off to daemon on
  + changing the height and width properties to match the cameras height 768 width 1024
  + changing the framerate from 1 to 60
  + changing output\_pictures on to output\_pictures off
  + changing stream\_port 0 to stream\_port 8081
  + changing stream\_localhost off to stream\_localhost on
  + changing webcontrol\_port 0 to webcontrol\_port 8082

Now that the web services are setup, I opened Geaney Brush Matthew, Hopf Dominic, Lanitz Frank, Treleaven Nick, Tröger Enrico and Wendling Colomban. (2006), one of the raspberry pi’s IDEs, to write the code for the website in two files the index.html file contains the main HTML code for the webpage detailing the main objects on the page and the files that are linked with it. The second file called MainPage.css this file details the details of the content on the webpage such as the position, size and colours. After coding the HTML and CSS files were moved to the /var/www/html folder where the test index.html was replaced with my webpage’s index.html currently the page and stream are accessible on a local network only with the webpage on port 80 hosting the stream in an iframe. Now that the webpage was as set up as it could be I started on writing the code for interfacing the Arduino and the Raspberry Pi through a UART connection using the Raspberry Pi’s GPIO (general purpose input and output) pins and the Arduino’s pre setup connection initially this took a little trial and error. First I connected the raspberry pi’s 3.3v supply (gpio pin 1) ground (gpio pin 6) TxD (gpio pin 8) and RxD (gpio pin 10) to a breadboard for easy prototyping of circuitry initially and incorrectly I connected Tx and Rx directly to the respective pins on the Arduino however at closer inspection the Tx of the Arduino needed to be connected to the Rx of the raspberry pi and the Rx of the Arduino to the Tx of the raspberry pi. Following this I went into the raspi-config menu with the command sudo raspi-config and went to option 5 interfacing options then to option P6 serial and disabled console over serial option but kept the hardware enabled. After setting up the required hardware options I wrote some code that I adapted from (Instructables, n.d) see appendix code listing 5 and tested it, where it failed, I then ran the command ls –l /dev to see what port I was using and the name of that port in this case I was using serial0 which is /dev/ttyS0 instead of /dev/ttyAMA0 which was on the instructables article the code for getting my temperature readings from the Arduino into the Raspberry Pi now worked. After doing some research into JavaScript, to become more informed upon how to make webpages more interactive, I found that having written the serial communications code in Python I made my project overly complex so I set about researching a better and more streamlined solution and came back with writing the whole thing in JavaScript and running it on the webserver. In order to make a start on this changeover I needed to install node.js (), which is a JavaScript webserver package, having done some work on other areas of my project over the recent days I ran the update commands again and then ran the command to install node.js and ran a verification command to make sure that it was installed correctly. The next two bullet points are the commands I used to install and verify node.js.

* Installing node.js for raspberry pi by running the upgrade commands sudo apt-get update && sudo apt-get dist-upgrade -y then running sudo apt-get install -y nodejs
  + running node -v verifies the version and that the install went according to plan

Finally, I wrote a proof of concept test code in node.js using the serialport module and a tutorial (w3schools, 1999) the code takes data from the serial port and writes it to the console this code worked first time as I used the ideas and concepts that I had learned while writing the python code.

**Work plan for the second semester**

1. Write as I work. Write the Reports and presentations as I go so as not to forget anything
2. Build the new web server and get it to replace the current server (apache)
   1. in order to use the node.js with the pi’s gpio we need to install the onoff module this was done with the npm install onoff command
   2. in order to host a webservice that we can interface our AJAX and serial port javascript to we need to install socket.io this is done with the command nmp install socket.io –save
   3. (follow on with code to create webserver use ajax and node.js to create and updateable webpage that displays the data.)
3. Build the website up so that it includes the JavaScript elements, the updating graphs and incorporate the GPIO code.
4. While keeping motion installed and the primary camera streaming method attempt to make a new camera streaming method.
5. Tidy up the scripts that have been written and get everything to run on boot
6. Run a testing process on the system for local connections
7. Attempt to find a method of getting the internet as current accommodation doesn’t allow port forwarding
8. Find a method of creating an emailing service and test it
9. Create a light source that comes on when it is dark enough and is needed by the camera

**References**

The Apache Software Foundation (1997) *Apache HTTP Server Project* Retrieved from <https://httpd.apache.org/> accessed 20/08/18

Brush Matthew, Hopf Dominic, Lanitz Frank, Treleaven Nick, Tröger Enrico and Wendling Colomban. (2006) *Geany* Retrieved from <https://www.geany.org/> accessed 21/08/18

Burton, Miles. (2018) *Arduino Temperature Control library* retrieved from <https://github.com/milesburton/Arduino-Temperature-Control-Library> accessed 17/08/18

Emmeshop (n.d) *Read and Write from serial port with Raspberry Pi*. Retrieved from <https://www.instructables.com/id/Read-and-write-from-serial-port-with-Raspberry-Pi/> accessed 20/08/18

The Home Brew Shop. (2019). Beer *Starter Kit With Barrel – Equipment Only*. Retrieved from <https://www.the-home-brew-shop.co.uk/acatalog/Beer_Starter_Kit_With_Barrel_Equipment_Only.html> accessed on 17/08/18

Kumar, Shambhoo., & Rajan, Rajeev. (2017, November 16) *How to use DS18B20 Waterproof Temperature Sensor* Retrieved from [https://create.arduino.cc/projecthub/iotboys/how-to-use-ds18b20-water-proof-temperature-sensor-2adecc accessed 15/08/18](https://create.arduino.cc/projecthub/iotboys/how-to-use-ds18b20-water-proof-temperature-sensor-2adecc%20accessed%2015/08/18)

ModMyPi LTD. (2016, 19 April). *How to give your Raspberry Pi a Static IP Address - UPDATE* Retrieved From <https://www.modmypi.com/blog/how-to-give-your-raspberry-pi-a-static-ip-address-update> accessed 18/08/18

Motion Project (2018) *Motion - Project* Retrieved from <https://motion-project.github.io/index.html> accessed 20/08/18

Raspberry Pi Foundation. (n.d). *Downloads* Retrieved from <https://www.raspberrypi.org/downloads/> accessed 17/08/18

Raspberry Pi Foundation. (2018, 30 April). *Setting up and Apache 2 webserver on Raspberry Pi* Retrieved from <https://www.raspberrypi.org/documentation/remote-access/web-server/apache.md> accessed 18/08/18

Raspberry Pi Tutorials. (n.d.). *How to setup a Raspberry Pi Security camera Livestream* Retrieved from <https://tutorials-raspberrypi.com/raspberry-pi-security-camera-livestream-setup/> accessed 18/08/18

SD Association. (2018). *SD Memory Card Formatter* Retrieved from <https://www.sdcard.org/downloads/formatter_4/index.html> accessed 18/08/18

Stoffregen, Paul. (2018) *OneWire* Retrieved from [https://github.com/PaulStoffregen/OneWire accessed 17/08/18](https://github.com/PaulStoffregen/OneWire%20accessed%2017/08/18)

w3schools. (1999) *Node.Js and Raspberry Pi* Retrieved from. [https://www.w3schools.com/nodejs/nodejs\_raspberrypi.asp accessed 10/12/18](https://www.w3schools.com/nodejs/nodejs_raspberrypi.asp%20accessed%2010/12/18)

Youngs. (n.d) *Brew Buddy Beer 40 Pints Home Brew Starter Kit* Retrieved from <https://www.amazon.co.uk/Brew-Buddy-Beer-Pints-Starter/dp/B001C54KQW/ref=sr_1_5?s=kitchen&ie=UTF8&qid=1542641698&sr=1-5&keywords=brew+buddy> accessed on 17/08/18

**Appendix**

**Not all code listings are present at this time as some are still being worked on or are not currently readily accessible**



Figure 1 showing the pinout and wiring of the whole project.

Code listing 1 (First attempt at Arduino code failed attempt)

//setting up the required variables

int tempVal = 0;

const int tempPin = A3;

const int ledPin = 13;

long previousMillis = 0;

long previousMillis2 = 0;

unsigned long currentMillis = millis();

unsigned long currentMillis2 = millis();

void setup() {

//setting up serial for output

//setting the led pin to ouput for blinking

Serial.begin(9600);

pinMode(ledPin, OUTPUT);

}

void loop() {

// put your main code here, to run repeatedly:

//reading the analog pin

//sending data

tempVal = analogRead(tempPin);

sendValue(tempVal);

ShowLife();

}

//custom function for sending the raw data to the RPI

void sendValue(int Value)

{

//mapping the data to a siutable range

//Convert to deg c within the ds18b20's range

//printing the value to the serial connection

//waiting for the adc to settle for next reading

long SendInterval = 2;

if (currentMillis2 - previousMillis2 > SendInterval)

{

Value = map(Value, 0, 1023, -55, 125);

Serial.println(Value);

previousMillis2 = currentMillis2;

}

}

void ShowLife()

{

long blinkInterval = 1000;

int ledState = LOW;

if (currentMillis - previousMillis > blinkInterval)

{

if(ledState == LOW)

{

ledState = HIGH;

}

else

{

ledState = LOW;

}

}

digitalWrite(ledPin, ledState);

previousMillis = currentMillis;

}

Code listing 2 (second attempt at Arduino code failed attempt)

#include<OneWire.h>

#include<DallasTemperature.h>

//setting up constant variables

const int ledPin = 13;

const int tempPin = A3;

const int blinkInterval = 500;

const int readingInterval = 5;

//setting up variables

int ledState = HIGH;

int tempValRaw = 0;

unsigned long currentMillis = 0;

unsigned long previousMillisLED = 0;

unsigned long previousMillisTemp = 0;

void setup() {

// put your setup code here, to run once:

Serial.begin(9600);

pinMode(ledPin, OUTPUT);

}

void loop() {

// put your main code here, to run repeatedly:

//get the latest value of millis

//update the led based on the currentMillis and ledState variables

currentMillis = millis();

tempValRaw = analogRead(tempPin);

LifeLED();

SendData(tempValRaw);

}

void SendData(int tempValRaw)

{

int tempValMapped = tempValRaw;

if ((unsigned long) currentMillis - previousMillisTemp >= readingInterval)

{

//tempValMapped= map(tempValRaw, 0, 1023, -55, 125);

Serial.println(tempValMapped);

previousMillisTemp = readingInterval;

}

}

void LifeLED()

{

//checking to see if the led state is low

//if true we check to see if the blinkInterval has passed

//if true we set the led state to high and write this change to the pin

//we then update the previousMillisLED varaible by our blinkInterval

//this is so that the intervals are equal

//as the line could also be previousMillisLed = currentMills

//however this can be less accurate

if (ledState == LOW)

{

if((unsigned long) currentMillis - previousMillisLED >= blinkInterval)

{

ledState = HIGH;

digitalWrite(ledPin, ledState);

previousMillisLED += blinkInterval;

}

}

//if the led is already high we check to see that the blinkInterval has passed

//if true we set the ledState to low and write this change to the pin

//and update the previousMillisLED by the blinkInterval

//this is done the same way as above for the same reason

else

{

if((unsigned long) currentMillis - previousMillisLED >= blinkInterval)

{

ledState = LOW;

digitalWrite(ledPin,ledState);

previousMillisLED += blinkInterval;

}

}

}

Code listing 3 Arduino code pulled from Kumar and Rajan (2017)

#include <OneWire.h>

#include <DallasTemperature.h>

#define ONE\_WIRE\_BUS A3

OneWire oneWire(ONE\_WIRE\_BUS);

DallasTemperature sensors(&oneWire);

float Celcius=0;

float Fahrenheit=0;

void setup(void)

{

Serial.begin(9600);

sensors.begin();

}

void loop(void)

{

sensors.requestTemperatures();

Celcius=sensors.getTempCByIndex(0);

Fahrenheit=sensors.toFahrenheit(Celcius);

Serial.print(" C ");

Serial.print(Celcius);

Serial.print(" F ");

Serial.println(Fahrenheit);

delay(1000);

}

Code listing 4 Final code for the Arduino labelled MK3

//included libraries

//OneWire.h and DallasTemperature.h for getting the data

//from the ds18b20 temperature sensor

#include <OneWire.h>

#include <DallasTemperature.h>

//defining the bus that the components will be connected to

#define ONE\_WIRE\_BUS A3

//initialising the OneWire and Dallas Temperature classes

OneWire oneWire(ONE\_WIRE\_BUS);

DallasTemperature sensors(&oneWire);

//setting up constant variables

const int ledPin = 13;

const int blinkInterval = 500;

const int readingInterval = 1000;

//setting up variables

int ledState = HIGH;

float Celcius=0;

float Fahrenheit=0;

unsigned long currentMillis = 0;

unsigned long previousMillisLED = 0;

unsigned long previousMillisTemp = 0;

void setup() {

// put your setup code here, to run once:

//setting up the serial interface along with the sensors

//and the needed pin for the onboard LED

Serial.begin(9600);

sensors.begin();

pinMode(ledPin, OUTPUT);

}

void loop() {

// put your main code here, to run repeatedly:

currentMillis = millis();

LifeLED();

SendData();

}

void SendData()

{

if ((unsigned long) currentMillis - previousMillisTemp >= readingInterval)

{

sensors.requestTemperatures();

Celcius = sensors.getTempCByIndex(0);

Fahrenheit = sensors.toFahrenheit(Celcius);

Serial.print(Celcius);

Serial.print(",");

Serial.println(Fahrenheit);

previousMillisTemp += readingInterval;

}

}

void LifeLED()

{

//checking to see if the led state is low

//if true we check to see if the blinkInterval has passed

//if true we set the led state to high and write this change to the pin

//we then update the previousMillisLED varaible by our blinkInterval

//this is so that the intervals are equal

//as the line could also be previousMillisLed = currentMills

//however this can be less accurate

if (ledState == LOW)

{

if((unsigned long) currentMillis - previousMillisLED >= blinkInterval)

{

ledState = HIGH;

digitalWrite(ledPin, ledState);

previousMillisLED += blinkInterval;

}

}

//if the led is already high we check to see that the blinkInterval has passed

//if true we set the ledState to low and write this change to the pin

//and update the previousMillisLED by the blinkInterval

//this is done the same way as above for the same reason

else

{

if((unsigned long) currentMillis - previousMillisLED >= blinkInterval)

{

ledState = LOW;

digitalWrite(ledPin,ledState);

previousMillisLED += blinkInterval;

}

}

}

Code listing 5 Code from Instrucables to be adapted

#!/usr/bin/env python

import time

import serial

ser = serial.Serial(

port='/dev/ttyUSB0',

baudrate = 9600,

parity=serial.PARITY\_NONE,

stopbits=serial.STOPBITS\_ONE,

bytesize=serial.EIGHTBITS,

timeout=1

)

counter=0

while 1:

x=ser.readline()

print x