**Progress Report**

**Introduction** (1 - 2 pages)

Outline the problem.

**Aim and Objectives** (1/2 page)

State the aim and the major objectives of the project.

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** (1- 2 pages)

In case of any problems appeared, explain how they are addressed. Include the update of a work plan.

Trail and error with theft from stack overflow. (references) find something better

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. The more recent approach has been based on research and learning making recent progress inherently slower.

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

This should include a critical account of the major findings of the literature survey. This is of major importance, literature review could be used as it is (or with small modifications, additions) in the final report.

* Won’t be talking about the “just add water kits” as these kits don’t conform to what my project is designed to do however my project could be used to monitor or regulate the temperature of brew while it ferments.
* Starter kits come with all the needed parts but do not come with anything other than a manual way for measuring the temperature and kind of temperature regulation is absent.
* Having done some research I have come to the conclusion that using python for the update code is a bad choice as it overcomplicates the programming and adds an unnecessary language

For this project 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. However, the STM32F103C8T6 would also be a suitable development platform as it has better clock speeds and better memory capacities.

* Using pro mini dev board over Arduino Uno as it is physically smaller cheaper and runs on 3.3v whereas the Uno runs on 5v cons that it has a fixed clock speed of 8MHz however for my prototype uses this is not an issue
  + Can be directly soldered too as to reduce the form factor of the probe
  + Direct soldering means I can determine the length of cable the mini has from the pi
* 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
  + Tx and Rx used for serial connection to Raspberry Pi

**Progress Made** (up to 5 pages)

Discuss the progress made during the first semester and include preliminary observations/results.

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)

* Set up the raspberry pi:
  + Ran following update commands:
    - sudo apt-get update
    - sudo apt-get dist-upgrade -y
  + Ran the following uninstall commands
    - sudo apt-get purge minecraft-pi wolfram-engine scratch -y
    - sudo apt-get scratch2 libreeoffice\*
    - sudo apt-get autoremove
  + Ran the following install commands:
    - sudo apt-get install apache2
    - sudo apt-get install motion
      * 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 8080
* Now that the web services are setup I opened Geaney, one of the raspberry pi’s IDEs, to write the code for the website in two files and index.html file and a MainPage.css file that after coding 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.
* Connecting the Arduino to the raspberry pi via a UART connection 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 (adapted from instructables) 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 Ardiuno into the pi now worked.
* 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
  + 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
  + 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
  + (follow on with code to create webserver use ajax and node.js to create and updateable webpage that displays the data.)
* Wrote proof of concept code in node.js using the serialport module this code takes the data from the serial port and writes it to the console.

**Work plan for the second semester** (1 page)

Detail how the remainder of the project is to be carried out. Include a work plan.

**References**

Use the recommended format (either Harvard-SHU or APA 6) for more info see <http://libguides.shu.ac.uk/referencing>

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

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)

**Appendix**



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;

}

}

}