G54MRT Coursework 2 - The Ergonomotron

Jack Ellis
psyje5@nottingham.ac.uk
4262333

Contents

0.1 Summary 2
0.2 Background and Motivation 2
0.3 Related Work 2
0.4 Design 2
0.5 Implementation 4
0.5.1 Setup 4
0.5.2 Calibration 5
0.5.3 Real-Time Sensing 6
0.5.4 Processing 6
0.6 Testing 7
0.7 Critical Reflection 7

0.1 Summary

The broad idea of this project is to ensure a user is sitting properly and comfortably at their computer. The project will use a combination of light sensors, and accelerometers to detect first whether or not the user is looking at their screen, and second whether or not they are craning their neck to do so in such a way as would lead to long-term damage.

0.2 Background and Motivation

Prolonged viewing of a computer screen can lead to many problems with regard to eye health, including dry eye syndrome[1], a condition in which the eyes do not produce enough tears, leading to irritation. "Back and neck pain, headaches, and shoulder and arm pain are common computer-related injuries."[2], and are related to poor posture when a user is at their workstation. This project aims to alleviate these issues using a combination of light sensors and a 3-axis accelerometer. The intended setting is, ultimately, in the workplace. Many office jobs require employees to be sat at computers for large amounts of time, with no proper enforcement of regular breaks to ensure employee health.

0.3 Related Work

There is a lot of work currently centered around making computers more ergonomic and healthier to use, including standing desks, the adoption of blue light filters in software, and various monitor and laptop stands designed to raise the device screen such that the user is not craning their neck down risking damage.

0.4 Design

The system is designed around 3 sensors: 2 light sensors, and a 3-axis gyroscope/accelerometer. The light sensors will be arranged vertically at the front of the headgear, with a divider in between them to give an "above/beneath" light reading. The gyroscope will be mounted wherever is convenient on the headgear. Figure 1 shows the connections that will be made.

Figure 1: A diagram showing the connections

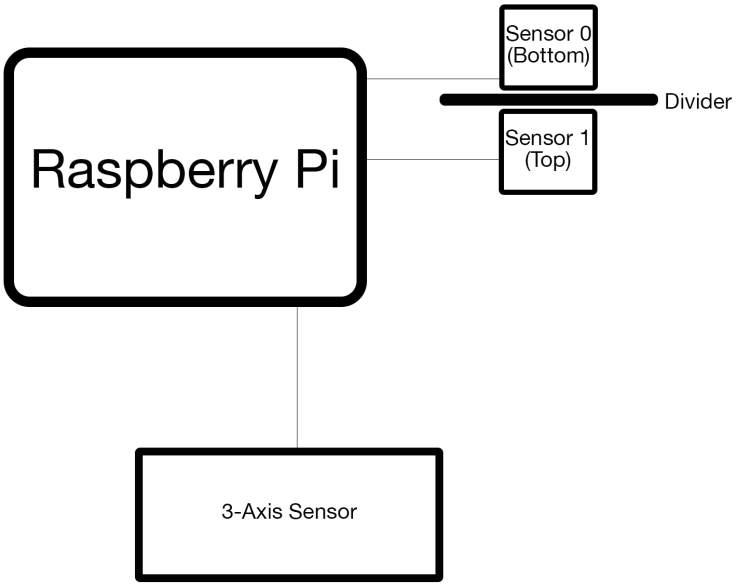
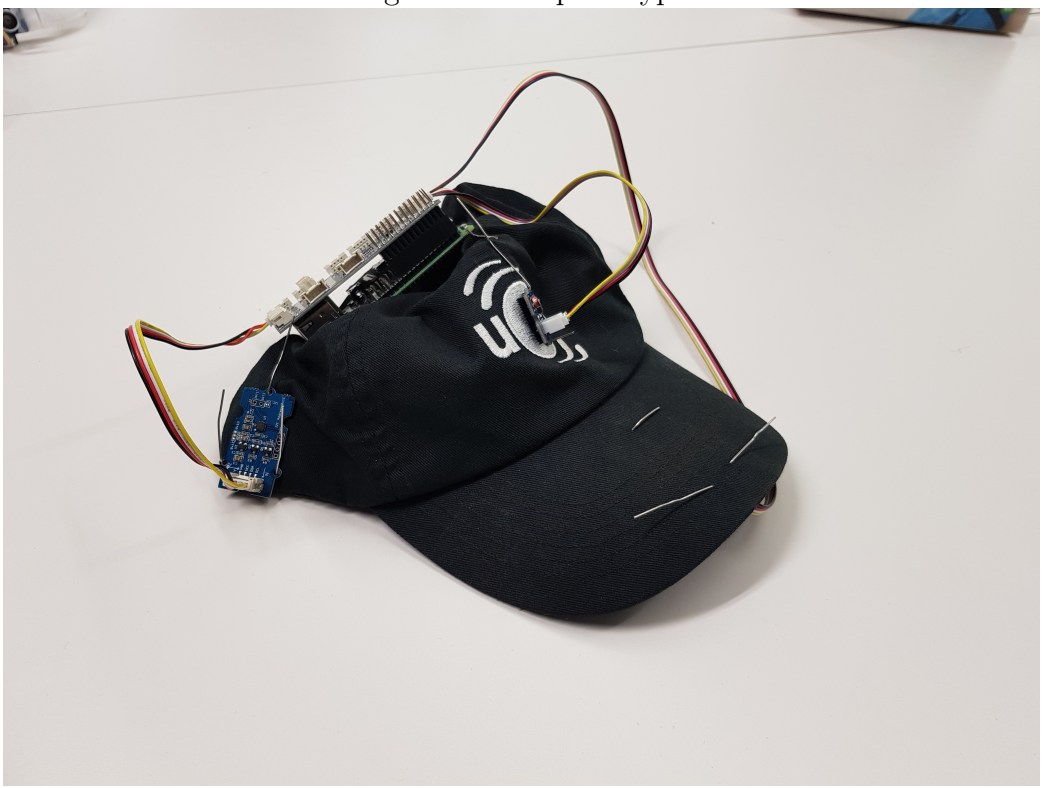


Figure 2: The prototype



0.5 Implementation

The prototype is based around a baseball cap, with the peak acting as the divider. The sensors are attached using bent paperclips, and the Raspberry Pi is mounted to the top of the hat. One light sensor is mounted above the peak, and the other hangs below it, while the gyroscope is mounted to the side of the hat, as shown in the photo in Figure 2.

The code is split into three main sections: the calibration phase and the actual real-time sensing phase.

0.5.1 Setup

This takes place outside of the main `while` loop, and will declare all used variables and functions within the code.

First `time` and `math` (as well as the grovepi modules) are imported. This is to allow for tracking of the duration of certain events and to allow the value of `pi` to be used in angle calculations.

Next a number of global variables are declared. The main timer, `t`, is set to 0, and the initial values for the low-pass filters on both the difference in light levels and the angle of neck lean are set, to 0 and $\pi/2$ respectively. Finally the variables for the calibration phase are initialised - `calibrationPassesInit` is set to 100 (there will be 100 passes of calibration, this can be adjusted as deemed necessary), and `calibrationPasses` is set to the value of `calibrationPassesInit`. The accumulator values that will be used to calibrate both light sensors are set to 0.

Listing 1: Setting up variables

```
t = 0                  # Initialise time
aDiffLo = 0            # Initialise the low-passed value for the difference in light levels
leanLo = math.pi/2     # Initialise the low-passed value for the lean

problemTime = 20          # The number of seconds before there's a problem
problemAngle = math.pi/8 # The angle at which there is a problem

calibrationPassesInit = 100 # Number of calibration passes
calibrationPasses = calibrationPassesInit # Calibration pass counter

a0Cal = 0 # Accumulator for the values recorded by a0
a1Cal = 0 # Accumulator for the values recorded by a1
```

0.5.2 Calibration

Initially there are two variables, `calibrationPassesInit` and `calibrationPasses`. These are both set to 100, which will be the number of times the sensors take data with which to calibrate themselves to the light levels in the rest of the room. This corresponds to roughly 10 seconds of reading.

The light levels are stored in variables, with the values being accumulated over the number of passes, and once the counter (as defined by `calibrationPasses`) reaches 0, that value is divided by the initial number of passes (defined by `calibrationPassesInit`), to give an average value for the light level in the room both above and below the divider.

Listing 2: Calibration Phases

```
  a0 = grovepi.analogRead(0)


  # Read from an analog sensor on input 1 (this should be the bottom one)
  a1 = grovepi.analogRead(1)

  if calibrationPasses > 0:
    a0Cal += a0 #/ calibrationPassesInit
    a1Cal += a1 #/ calibrationPassesInit
    print("Calibrating, ", calibrationPasses, "a0Cal: ", a0Cal, " a1Cal: ", a1Cal)
    calibrationPasses -= 1
  elif calibrationPasses == 0:
    a0Cal = a0Cal / calibrationPassesInit
    a1Cal = a1Cal / calibrationPassesInit
    print("Calibrating done, a0Cal: ", a0Cal, " a1Cal: ", a1Cal)
    calibrationPasses -= 1
  else:
```

0.5.3 Real-Time Sensing

Once the calibration phase is completed, the real time sensing of light levels and neck angle begins. The light values from the sensors have the values calculated by the calibration phase subtracted from them, and the relative difference between the fixed values of the top and bottom sensor calculated. Finally for this section, a low-pass filter is applied to this difference such that the user cannot quickly move their head and reset the timer (the timer will be discussed later).

Listing 3: Calculating the difference between sensor values

```
a1Fix = a1 - a1Cal

# Calculate the absolute difference between a0 and a1
aDiff = a1Fix / a0Fix

# Apply a low-pass filter to this difference
aDiffLo = loPass(aDiff, aDiffLo)
```

The neck angle is then taken and a low-pass filter applied similar to the process for the light sensors.

Listing 4: Calculating the neck angle

```
# Apply a low-pass filter to this
leanLo = loPass(lean, leanLo)
```

0.5.4 Processing

In this phase it is first determined whether or not the difference between top and bottom sensors is more than a factor of 1.2 (i.e. whether or not the bottom sensor value is more than 1.2 times the top sensor value). If it is, the user is deemed to be looking at a screen, and the timer has 0.1 added to it (this roughly corresponds to the number of seconds between each sensor reading). If the timer reaches the value of `problemTime`, defined before the main loop begins, it prints a message instructing the user to look away from the screen. While this goes on, the value of the user's neck angle is monitored, and if it exceeds the value of `problemAngle`, again defined outside of the main loop, it will print a different error message instructing the user to lean their head back. If the user is not deemed to be looking at a screen the timer resets.

Listing 5: Processing and printing the data

```
t += 0.1            # Increase the time by 0.1
if t > problemTime: # If that time is more than 200 (should be ~20s)
  print("You have been looking at the screen for too long, please look away now")
else:               # Else we have a look at how much the user's leaning
  if abs(leanLo) < problemAngle:  # If the LP'd lean value is within bounds...
    print("You are looking at a screen, and leaning fine")
  else:                           # Otherwise...
    print("You are looking at a screen, and leaning too much!")
```

```
else:          # If the difference is small
  t = 0        # Reset the timer, print the values
  print("You are not looking at a screen")
```

0.6 Testing

Testing the system consisted of attempting to use the system and determining whether or not looking at a screen was detected at all. The testing was also an opportunity to fine-tune the threshold values for said determination, as well as the method by which the difference was calculated.

Initially the difference was calculated as an absolute value, i.e. the value of one sensor minus that of the other, however in testing in rooms of varying ambient light intensities it was found that this did not work - a brighter room would result in false negatives. Adjusting the code to use a relative difference, i.e. the value of one sensor *over* that of the other, made the code work considerably more effectively.

Once this "calibration" testing was done I set about testing the system in a variety of scenarios: light room, dark room, and with different types of screens (phone, laptop, desktop). In all cases it successfully determined when I was looking at a screen, and whether or not my neck was at an excessive angle. In some cases however the neck tilt was such that both light sensors were aimed at the screen in use, consequently the system failed to register the excessive angle. In a future revision I would invert the order of the if statements - detect the neck tilt regardless of the light level and alert the user of excessive neck angle whether or not they are looking at a screen.

0.7 Critical Reflection

Overall I am happy with the system, as stated in the previous section I would change the order of "event priority", neck tilt taking precedence over looking at a screen. The system works well in my view however the initial prototype is very clunky, not to mention quite heavy, which will exacerbate any neck-tilt issues as the user must contend with the weight of the device. Having the user tied to a desk via the Pi's USB lead or having to carry around a battery pack is also not ideal, but this is only a prototype and consequently the finished system would be somewhat more refined.

There is scope for further enhancement; additional sensors to detect the distance from the user to the screen would be the next logical step, with the "problem time" value becoming a factor of this distance.

Broadly speaking however I feel that this project has done what it set out to do very well.

Bibliography

[1] *Office screen work linked to dry eye syndrome*, 2014-06-18
www.nhs.uk/news/lifestyle-and-exercise/office-screen-work-linked-to-dry-eye-syndrome
Accessed 2018-03-08

[2] *Computer-related injuries*
www.betterhealth.vic.gov.au/health/healthyliving/computer-related-injuries
Accessed 2018-03-08

Appendix A - Instructions for setup/running

This will assume you are using the prototype hat seen in Figure 2.

1. Connect an LED screen to the Raspberry Pi - this can be disconnected later however it is needed so as to ascertain the IP address of the Pi.
2. Connect the Raspberry Pi to power.
3. From an SSH client connect to the Pi using the IP address on the LED screen, logging in as user `g54mrt`.
4. Disconnect the LED screen.
5. Copy `sensors.py` to the root folder of the user `g54mrt`.
6. Run the command `python sensors.py`.
7. The program will run. To terminate the program press control-C.