README file cs380 final

Rhys Jordan and Braylee Cumro

**Raspberry Pi**

Python (raspberry pi) code

Libraries:

- picamera2
- gpiozero
- time
- smbus
- datetime
- picamera2.encoders
- numpy
- PIL

We added any libraries missing with the format “sudo apt install python3-LibraryName" where library name is the library we were adding. Power cycle the device to see the changes.

ClearSans-Thin.ttf file

Usually, a default font for a raspberry pi

Interfaces:

- SSH
- VNC
- I2C

These are the interfaces we had turned on in the final configuration. There are two ways these can be added.

1) In the rpi navigator there is a drop-down option called preferences and in preferences there should be an option raspberry pi configuration. In the navigator that pops up there should be a tab labeled interface. You can toggle on these interfaces here and then power cycle the device to save these changes.

2) At the command line you can enter “sudo raspi-config". From the screen that pops up go to the interface options and turn on the needed interfaces. To save these changes, power cycle the device.

Cron Job:

If you want to test how we set up the program to run automatically when the raspberry pi was plugged in or rebooted, you need to set up a Cron Job. To add a Cron Job type “crontab -e” at the command line. This opens a file that you can add jobs to. At the bottom of this file add the line “@reboot python /home/class380/Desktop/cs380_finalProject.py &” with the full direct path to the program file. If you want to check the contents of the file, you modified you can use “crontab -l”

Versions:

The python version we were using was 3.9.2

The raspberry pi operating system version was a Raspbian GNU/Linux 11 (bullseye) operating system. The full information on this operating system is below.

PRETTY_NAME="Raspbian GNU/Linux 11 (bullseye)"

NAME="Raspbian GNU/Linux"

VERSION_ID="11"

VERSION="11 (bullseye)"

VERSION_CODENAME=bullseye

ID=raspbian

ID_LIKE=debian

HOME_URL="http://www.raspbian.org/"

SUPPORT_URL="http://www.raspbian.org/RaspbianForums"

BUG_REPORT_URL="http://www.raspbian.org/RaspbianBugs"

Environment setup:

The raspberry pi needs an operating system flashed onto a micro-SD card to work correctly. To add the operating system to the micro-SD card I used the Raspberry Pi Imager. With this imager choose the device as a raspberry pi 4 device. As for the operating system, choose the option “Raspberry Pi OS (other)” to get the older versions of the operating system. This should pull up a list with some of the Bullseye operating systems. I believe I choose the Raspberry Pi OS

(Legacy) Full option. The Bullseye ones should match as long as the description includes the desktop environment and the recommended applications.

Our code assumes you have a directory named /home/class380/Desktop/dataFinal/. To get this setup the username of the raspberry pi should be class380. This is set in the initial boot with the operating system along with a password. If this directory does not exist because of the username of the machine lines 17, 41,46, and 53 use this file path.

If you want to hook up a keyboard or mouse this is done with the USB ports. To get display there is a micro-HDMI port labeled HDMI10. Hook up any display cord with this end to a monitor to get the desktop view of the raspberry pi. This allows you to access the Thonny application and command line for the raspberry pi.

The PIR sensor should be hooked up to the SDA and SCL specific pins on the raspberry pinout as well as one of the ground pins. The temperature sensor should be on GPIO pin 17. The camera can be put in the camera port by pulling up on the clasps and facing the blue on the ribbon towards the USB ports. Once situated you can clasp the ribbon in place.

Software tools:

The software we used was a Thonny environment. The operating system should come with Thonny automatically. This application can be accessed by going to the raspberry pi icon (the navigator like the windows icon) and choosing the programming option. In this option, there should be a Thonny environment we used to build this code.

Install

The program can be uploaded into the Thonny environment and ran that way if you do not set it up as a Cron Job. There should not be any other installations once you have the operating system and the libraries downloaded on the raspberry pi.

Startup

Before running make sure all of the sensors and the camera are hooked up correctly. You can test the temp sensor on the I2C with the command “sudo i2cdetect –y 1” at the command line. If it is hooked up correctly it should show the temperature sensor at address 0x48 of the I2C interface. The camera connection can be tested with the command “libcamera-hello” which will tell you if it can connect to a camera.

To run the code, you can hit the play button on the Thonny environment or restart the device to run Cron Job. If the code is run through Thonny, there is a built-in keyboard interrupt to cleanly end the code if need be. This keyboard interrupt is CTRL + C. You can also stop the code with the stop button if needed.

The raspberry pi should be connected to the internet to get an accurate timestamp. Otherwise, the timestamps shown will match the time on the raspberry pi rather than the local time.

Test

Once you run the code, the first thing it should do is send a message to the text file named systemStatus. This message has three parts separated by commas: the timestamp, the temperature in Fahrenheit, and the state of the machine. It should start in the state CAMERA_MOTION.

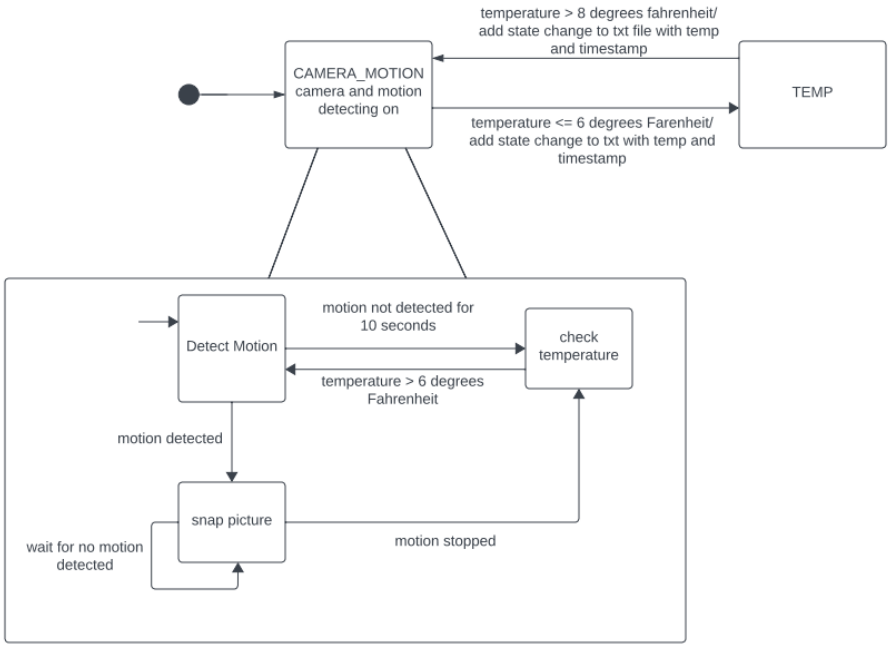
The device when working correctly will then wait ten seconds for motion in front of the camera. If no motion is detected before the timeout, it checks if it is too cold for the motion sensor (<= 6 degrees Fahrenheit). If the temperature is too cold, then it will switch to the state TEMP. Upon switching states, it should add another line in the systemState file with the timestamp, temperature, and the state. If the temperature is > 6 degrees Fahrenheit it restarts the loop of waiting for motion for ten seconds.

If motion is detected, it should take a picture. The picture should be named with the timestamp followed by an underscore followed by the temperature. All of them will be JPG images. This image is then given a caption on the top right. This caption has the timestamp followed by an underscore followed by the temperature. Finally, it should be saved in the same directory as the systemStatus text file.

There is one other piece after the motion is detected that can be tested. After motion is detected, it takes a picture; however, it does not move on to check the temperature and if a state change is needed until it stops sensing motion. Therefore, if there is continuous motion in front of the camera you will hopefully only get one image for that motion sequence if the motion sensor is detecting you correctly. In the finite state machine this is the arrow labeled “motion stopped”. Until the motion sensor is no longer detecting movement it should wait before moving to the next instruction in the code.

When I was testing this, to test the temperature I set it to the room temperature and used my hands to warm up the sensor enough to switch between states while maintaining the 2-degree gap between the switch cases.

If the Cron Job is working correctly, you should see a systemStatus message be added once the startup is complete. Note that the ampersand at the end of the Cron Job definition means it waits for the raspberry pi to start any of its applications and handles the infinite loop so this message may not be instantaneous.




**Base Station**

Required Libraries:

- Flask
- Flask_sqlalchemy
- Nodejs

Added the first two libaries using “pip install *library_name*” and the last on using “sudo apt install nodejs”

Versions:

- Python 3.12.0
- Node Js 20.10.0

Software tools:

Used Thonny to run the python code to host the server for the web app.

Startup:

Running the scp.js script:

To get data off the raspberry pi we use the scp.js script located in the TrailCameraWebApp/static/ directory. To run this script, you must have node.js installed and be in the TrailCameraWebApp/static/ directory. Then you can run “node scp.js” in the command line. If the script runs successfully then pictures and text files in the /home/class380/Desktop/dataFinal directory on the raspiberry pi will be copied to the TrailCameraWebApp/static/images directory on your local device. This script must be run in the TrailCameraWebApp/static/ directory. Beware that this script DELETES THE CONTENTS OF THE DIRECTORY located on the raspberry pi.

Running the web app:

You must first make sure you have all the python libraries you need installed. Then the first time starting up the web app you must run the createDB.py to set up the database needed in the web app. After that has been run you can run the app.py which will then give you a link in the terminal which will bring you to the home page of the web app. This python script must be running the whole time when using the app. Beware this is a DEVELOPMENT SERVER and should NOT be used in actual applications.

It is important that all scripts are run in the directory they live in. All images from the raspberry pi must be put in the images directory or they will not display properly. Below is how the what the directory TrailCameraWebApp/ should look.

```
TrailCameraWebApp/
instance/
Static/
images/
                        logs/
                            systemStatus.txt
                        *all .jpg and .txt files pulled from pi*
node_modules/
package.json
package-lock.json
scp.js
style.css
        templates/
        base.html
        display.html
        home.html
        logs.html
        app.py
```

createDB.py

Test:

1. New data is in the raspberry pi
2. Run scp.js
3. Run createDB.py if database does not exist
4. Run app.py
5. Go to the pictures tab
6. All pictures from the raspberry pi should be displayed
7. Go to logs tab
8. Hit update button
9. All logs should be displayed

Database:

All of the logs when pulled to the local device are saved in an SQL database. The database is created when running createDB.py and then used in the app.py. The upside of having this information in a database is that you use queries to get specific information.

ReadMe must be included all key information, such as required libraries and versions, environment setup, required software tools, and installation/startup/testing instructions.