CS380 final report

Braylee Cumro and Rhys Jordan

Overview

For this final project, we made a trail camera/game camera using a raspberry pi 4. This system includes two sensors, a motion sensor and a temperature sensor, as well as a camera module. The main goal was to make a system that took pictures when it detected motion with the motion sensor. The temperature sensors' job is to make sure that the important pieces do not turn on or attempt to work when it is too cold for them to function properly. We determined the temperature ranges for the different components using data sheets for each piece. These data sheets will be explained in the finite state machine. The system also has a notification system for when the states change due to the temperature. The images were given timestamps and the temperature as a text caption in the top left corner of the pictures.

The second part of this project is the base station. To extract both the images and log file we built a script using node.js that downloads all the files from a specified directory on the raspberry Pi to a specified directory on the local device. The directory on the raspberry pi is then cleared to make room for more photos. Then we built a basic web app to display the images and the log file. All the logs are stored in a SQL database.

FSM

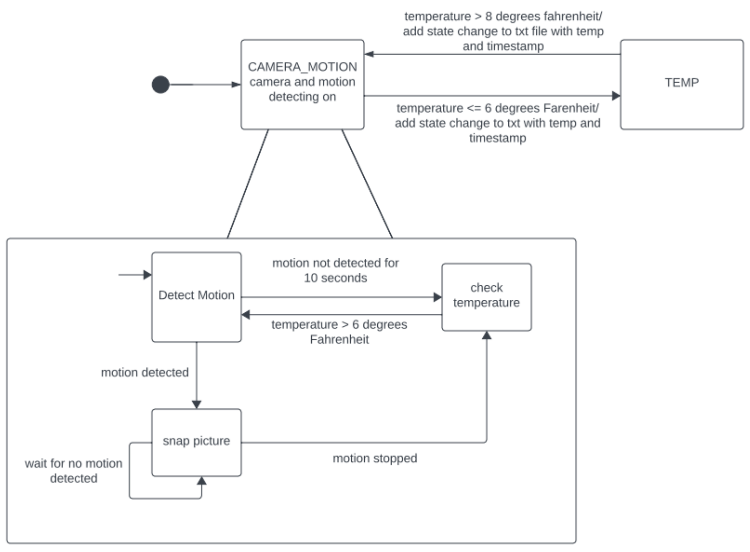

temperature > 8 degrees fahrenheit/ add state change to txt file with temp and timestamp
CAMERA_MOTION camera and motion detecting on
TEMP
temperature <= 6 degrees Farenheit/ add state change to txt with temp and timestamp
Detect Motion
motion not detected for 10 seconds
check temperature
temperature > 6 degrees Fahrenheit
motion detected
snap picture
wait for no motion detected
motion stopped

Figure 1: FSM

Figure one shows the finite state machine of this project. We have two main states, CAMERA_MOTION and TEMP. The switch between these states is controlled by the temperature sensor. The temperature for each switch was determined by looking at the data sheets for each of the devices we are using and finding where they stopped working correctly. The data sheets we used are in the references labeled one for the pir/motion sensor, two for the V2 camera module, and three for the temperature sensor. Of these devices, the pir sensor set the temperature limit of the camera module with a range of 5 degrees Fahrenheit to 158 degrees Fahrenheit. We put a buffer on this by looking for six degrees to turn it off. To turn the motion sensor and camera back on we wait for the temperature to be at least eight degrees. The two-degree gap is to attempt to prevent hysteresis from occurring. These two states are the two defined in the switch function. The python version we were using did not have python's version of a switch, so we had to build our own because upgrading the Python version was causing other issues.

The inner states of CAMERA_MOTION is achieved with if statements. Pythons equivalent of a switch statement was in a version of Python that broke some of the libraries we were using. It was cleaner to build the main switch from scratch and use the if statements for minor inner states. This state starts by looking for motion detected. It waits ten seconds in this state. If it detects motion it takes a picture and annotates it with the timestamp and temperature before moving to the state that checks if the temperature is too cold. It only moves to check the temperature once there is no more motion detected (motion stopped error) as a way to limit pictures being taken for the same motion/action the motion sensor picks up. If no motion is detected before the timeout, it moves directly to check the temperature. If the temperature is still above 6 degrees it goes back to waiting for motion; however, if the temperature drops below 6 degrees it switches to the TEMP state to wait for it to warm up.

When the main state is changed and when the system first turns on, it documents the change in a text file. This text file contains lines with the timestamp and temperature of the system when it entered that state.

Circuit and Sensors

The circuit for this project has six wires connected to two different sensors. The first of these sensors is a tmp 102 sensor. This sensor has three wires it needs: SCA, SCL, and ground. On a raspberry pi module 4 pin 3 is SDA and pin 3 is SCL. The ground pin can be connected to any of the ground pins of the raspberry pi but in our case, we used pin 9 as our ground for this sensor. The wire set up for this sensor can be found in figure two. Note that our sensor automatically connected itself to address 0x48 of the I2C interface of the raspberry pi. An image of this sensor is on the circuit in figure two (the top red one).

The second sensor was a passive infrared (PIR) motion sensor. This sensor also has three wires needed to set it up. The three pins on the motion sensor are shown in figure 3. To see the name of these pins the dome of the sensor can be removed to shoe the labels. The pin labeled VCC needs to be hooked to 5v of power and in our case, we used pin 2 to get this power. The pin labeled OUT is used as the pin to read data from. Any of the generic GPIO pins would work for this, but our specific code set this up to pin 11 or GPIO 17. Finally, this sensor has a GRD that needs to be hooked up to another one of the ground pins on the board. In our case we choose pin six as this particular ground. An image of this sensor is shown in figure 3.

Another device we had connected to the raspberry pi was a camera. This camera was the raspberry pi V2 camera module. The connection for this device is pointed out in figure two. Note that the blue section of the connection faces the USB ports when hooked up correctly. An image of this camera module is found in figure four.

Figure five points to other useful connections on the raspberry pi module. Our design works best with the internet and the raspberry pi has an ethernet port as shown in this diagram; however, we never used this port just a WIFI connection. There are 4 USB connection ports. These ports are typically used to connect a keyboard and mouse to the device when programming it. Finally, the port labeled HDMI 10 is the display port. Any monitor can be connected to this port to get the desktop view of the raspberry pi. This connection is needed to program the device in order to access the command line and other features of a raspberry pi. This port is a micro-HDMI port.

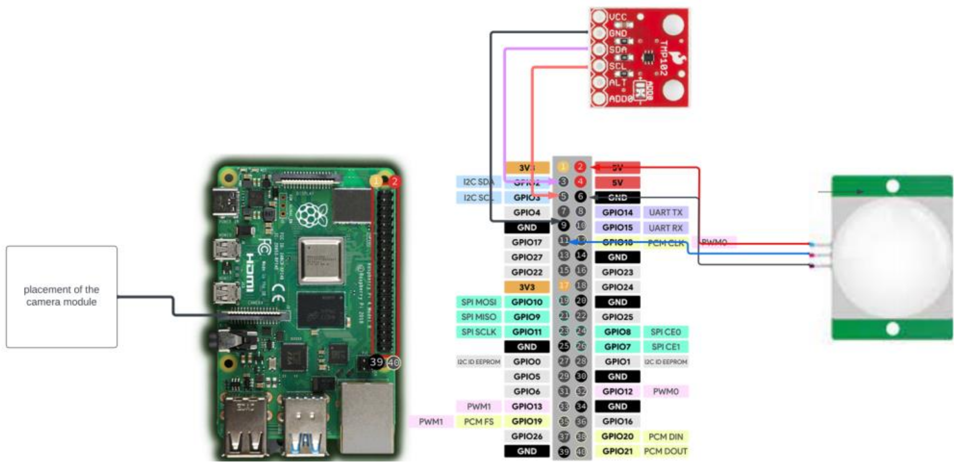


Figure 2: Circuit design

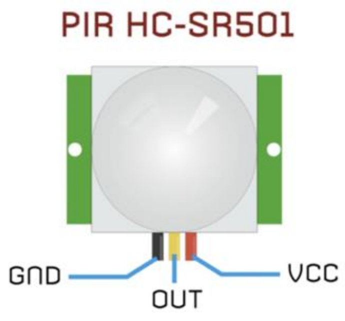


Figure 3: motion sensor pins

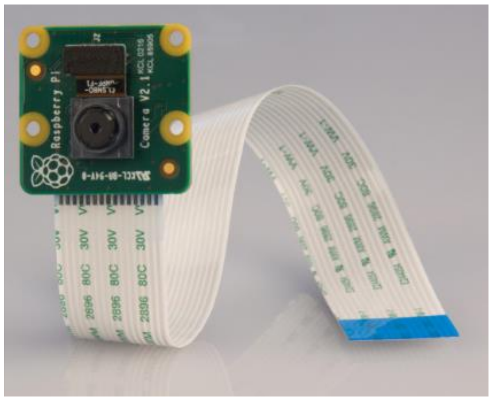


Figure 4: V2 camera module

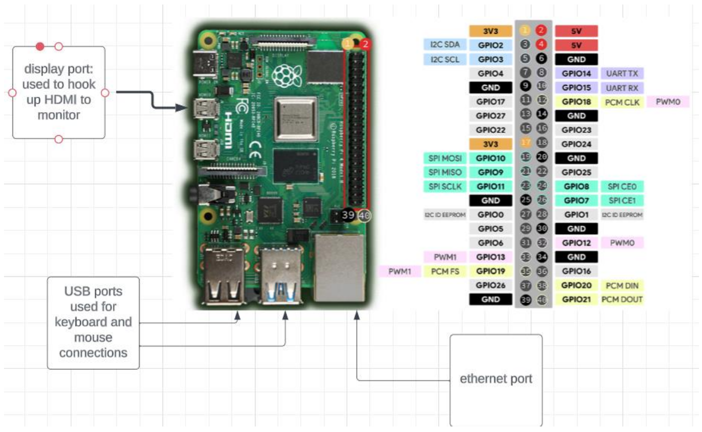

display port: used to hook up HDMI to monitor
USB ports used for keyboard and mouse connections
ethernet port

Figure 5: other useful port connections

Setup- raspberry pi

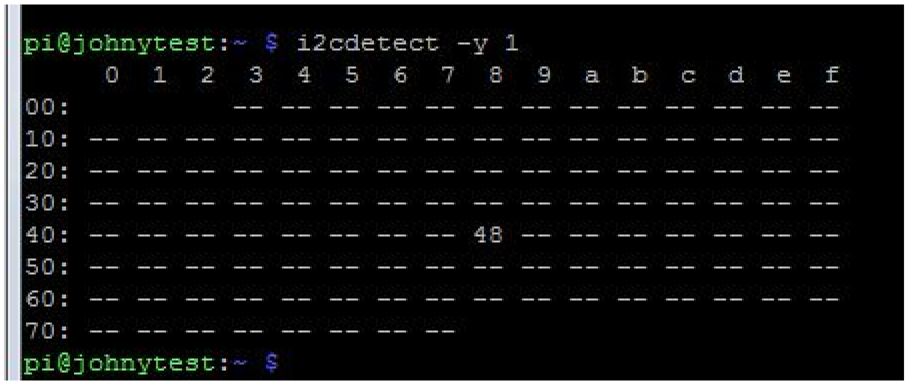
The raspberry pi we were using was a raspberry pi 4 with the Raspbian GNU/ Linux 11 (Bullseye) operating system. This operating system is on a micro-SD card in the raspberry pi that was flashed using the raspberry pi imager. The version of Python we were running was 3.9.2.

There are three interfaces that we had turned on for this project. These interfaces can be turned on using the raspberry pi configurations option and moving to the interfaces tab. These three interfaces were SSH, VNC, and I2C. After turning these interfaces on the raspberry pi must be rebooted and then these interfaces will remain on unless you explicitly turn them off again.

The main Python code has the libraries picamera2, gpiozero, time, smbus, datetime, picamer2.encoders, NumPy, and PIL. Of these, with the image I flashed for our operating system I only had to install the smbus package, but the others may be missing depending on the image you get. The camera library picamera2 only works when the legacy camera library is turned off. By default, most images have this off, but if it is not this can be turned off by running “sudo raspi-config" at the command line and finding the legacy camera under interfaces. Once you power cycle the device to make the change permanent. On top of these libraries, we have one resource file that we are pulling from the device to get the font of the text on our images. This resource is the ClearSans-Thin.ttf file in the font's directory of the raspberry pi. This reference is made on line 50 of the code. This should be a default font in the raspberry pi if it is flashed with the same operating system.

Our design runs the python script on reboot or start. This was done by adding a Cron Job to the raspberry pi. This Cron job was added by accessing the crontab –e and adding the line "@reboot python /home/class380/Desktop/cs380_finalProject.py &" to that file. This code is runnable without this setup; however, we added this for the final demo as well as a way to easily deploy this if it were an actual product.

The final requirements for the main Python code are that the motion sensor must be on GPIO pin 17, the temperature sensor should also be in the address of 0x48 of the I2C, and the code assumes that there exists a directory /home/class380/Desktop/dataFinal/. To check the address of the temperature sensor in the I2C you can run the command "sudo i2cdetect -y 1". If it is in the correct address the output of this command should appear like figure 6. The class380 directory was made available by making the username of the machine class380. If this directory does not exist because of the username of the machine lines 17, 41,46, and 53 use this file path.



```
pi@johnytest:~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- 48 -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
pi@johnytest:~ $
```

Figure 6: correct I2C setup

Setup- base station

The base station has two different parts that require different set ups. One is to run the web app and the other is needed to run the node.js script. The first step is to get the data off the raspberry pi which requires the use of the scp.js script located in the static folder (TrailCameraWepApp/static/scp.js). To run this script the user needs to have node.js installed which can be installed on Linux using the command "sudo apt install nodejs". Once nodejs is installed then you can run this script in the command line with the command "node scp.js". You must run this command from the TrailCameraWepApp/static/ directory for the images to end up in the right folder to be displayed on the web page correctly.

For the web app it uses python flask to run a development server. The user must install python flask with the command "pip install flask". Then a database is created using SQLAlchemy which can be installed with "pip install flask_sqlalchemy". Then you must first run the createDB.py script to create the database then run app.py to get the website running. The web page is hosted on the development server given when the python script runs. That python script must be running for the web page to run.

Testing

In order to test the python code, with the design we had the program would start anytime the raspberry pi was started or rebooted. If this is not set up to run this code, you can go to the navigation screen of the raspberry pi. In this screen, there is a tab labeled programming and that has a Thonny IDE option. Once Thonny is open you can upload the script into the IDE. The run button at the top of the IDE can start the program. If you need to stop the program in this mode, there is a keyboard interrupt built in by pressing CTRL+ C. The code has a continuous loop that runs until this keyboard interrupt happens.

If the code is running correctly, a text file named systemStatus should get a new line that shows the time stamp followed by a comma followed by the temperature followed by a comma and finally the system's state. Every time the system state changes, this file should get a new line with the same format. If the system is in the CAMERA_MOTION state, the device should take a picture, annotate it with the timestamp and temperature and save it to the same directory the text file resides in.

To test the base station there must be new data on the raspberry pi to collect. Then the scp.js script can be run to download everything from the specified directory on the raspberry pi to the directory on the local device. If this is the first time running the app then creatDB.py must be run to create the database. Once that is done then run the app.py script and go to the link in the terminal. This should pull up the web app in the browser. Under the pictures tab all the pictures taken should be displayed. The logs tab displays the logs which after hitting update should be the extent of the current logs.

Bugs/Limitations

A limitation on the Python script is that the current setup uses Datetime to get the time for the device. This works well when the device has some sort of internet connection where this can be updated when needed by the device itself. If this were to be deployed for long periods of time in a remote location a real time clock would need to be added. However, its current configuration would work well as doorbell camera or security camera on a property since you would be able to connect to the internet of the house when needed.

Another possible limitation is that the setup is dependent on a specific setup of directories (/home/class380/Desktop/dataFinal/). In theory, if you were making multiple of these devices it would have the same operating system flashed on it and the same directory setup due to that; however, if you were not aware of that directory system the code would not work without a few modifications. This setup lets us work with SSH and SCP to pull images and data off of the raspberry pi for the base station.

There are two other limitations to making this a deployable product. The first limitation is that we do not have a way to hold the camera in a certain position. We have been using electrical tape or prop the camera up to take the picture the way we want. This can be solved with a case for the camera or by 3d printing a holder for the camera. The second of these limitations is how much battery the device will take up. When we are in the state that waits for it to get warm enough to use the motion sensor, we could have saved battery by putting the device in some sort of sleep mode for a while. The temperature in most cases would not change quick enough that you would need this loop constantly taking up the battery. Similarly, if we wanted a final project with a better battery life, we could have the pir wake up the device from one of the sleep modes to take a picture and put it back in that mode when it's done when in the other state. This would use the pir as a sort of interruption that wakes up the device only when it needs to take a picture.

There is one possible bug with the temperature sensor. There was a time where the sensor would sometimes only want to get the data from the sensor once and then would get a read error. After searching for possible causes, I was told to switch out the wiring and this problem seemingly went away; however, I am not one hundred percent sure this problem is totally fixed so just in case that was a bug that this system was struggling with.

There are many limitations to the base station. The biggest limitation is the need to be on a secure network to pull data from the raspberry pi. We had trouble reliably using ssh to access the raspberry pi because of the schools' network. I used my hotspot as a work around, but it wasn’t the most reliable. To improve this, it would be better to implement communication between the base station and the raspberry pi through something like cellular or satellite.

One more limitation is that it runs on a development server which could not be used if this web application were to be accessible to the public. The base station simply serves as a UI to display the information from the raspberry pi but is not very user friendly. The logs are put into a database but there is no way to track which logs have already been written so they can be written multiple times. The use of a database allows us to query things such as the time stamps and temperature but there is no UI for such an application.

Another big limitation is the need to run the node.js script which is not very user friendly. The user must have the script in the right folder with the right libraries installed to get the pictures off of the device. To run the script there must be knowledge of the command line which is not common.

Conclusion

For this final project, we were able to write software for the raspberry pi that made a motion sensor triggered camera. There are some limitations we were not able to remove at the time we were working on this project. If we had more time, we would have made a more robust system and had more things happen in the state when the temperature is too cold for the motion sensor to work. One of the things that set us back on adding more to the project was the problem of raspberry pi depreciating the old camera library which had to be turned on with the legacy camera mode. We ended up having to change the camera library to run the program at reboot with the Cron job because the legacy camera mode turned off each time the device was power cycled, and this was not something we were able to turn on programmatically. Modifications to the Python code would include adding an RTC and including more sensors and data that may be helpful when there is no image available to the user.

The is a very basic user interface to display images and logs but lacks real world application. If we had more time to develop then I would have liked to add more functionality to getting data off the raspberry pi and utilize the database to make the data more accessible.

Overall, this system functions as a basic trail camera or simple motion detector. The base station is able to get data off the raspberry pi and display it. Although some improvements could be made, it is a basic functional IoT device.

References

[1] https://www.mpja.com/download/31227sc.pdf

[2] https://docs.rs-online.com/3b9b/0900766b814db308.pdf

[3] https://www.sparkfun.com/datasheets/Sensors/Temperature/tmp102.pdf