VCU Electrical and Computer Engineering
College of Engineering

EGRE 347
Object Oriented Programming

Final Project

Externally Triggered Location Tracking

Mehmet Kutlug
Daniel J. Youngk

May the 4th, 2022

Pledge: Mehmet Kutlug, Daniel J. Youngk.

"On my honor, I have neither given nor received unauthorized aid on this assignment"

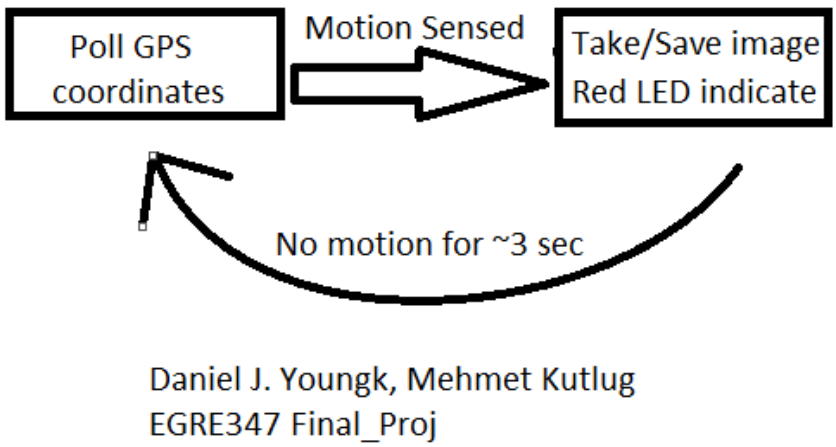
Abstract

In applications that require navigation, it is often useful to use a GPS tracker. Some applications require the location during certain events. In this application, a motion sensor triggers a sequence that saves the last known GPS coordinate, and an image for every second that motion is being sensed for a minimum of three seconds. The Linux OS is operated on a Raspberry Pi 4 machine, with a mix of C++ and Python languages used for program development. The program can operate autonomously using the "nohup" Linux terminal command. When motion is sensed, the GPS tracker saves the last known location to a .txt file. The module used here does not explore improvements on image recognition/sensor calibration, but simply uses the sensor [Part 1] as a simple external trigger. The possible applications of this system are not limited by attachment to stationary objects. For instance, the external trigger (GPIO) could attach to a metal detector to map high density locations. A possible improvement may be to have a location algorithm that takes into account nearby satellite connection. The sensing circuit operates autonomously, and continues to update the location log until terminated.

Software Introduction

The Raspberry Pi provides the environment for running C++/Python code, as well as GPIO terminals. Two terminals power red and green LEDs. The red LED indicates that the motion sensor is sensing motion. The green LED indicates program operation. One GPIO terminal is set as input from a motion sensor. The motion sensor is programmed with infrared detection software. For experimental purposes, the sensor is only used for simple motion detection.

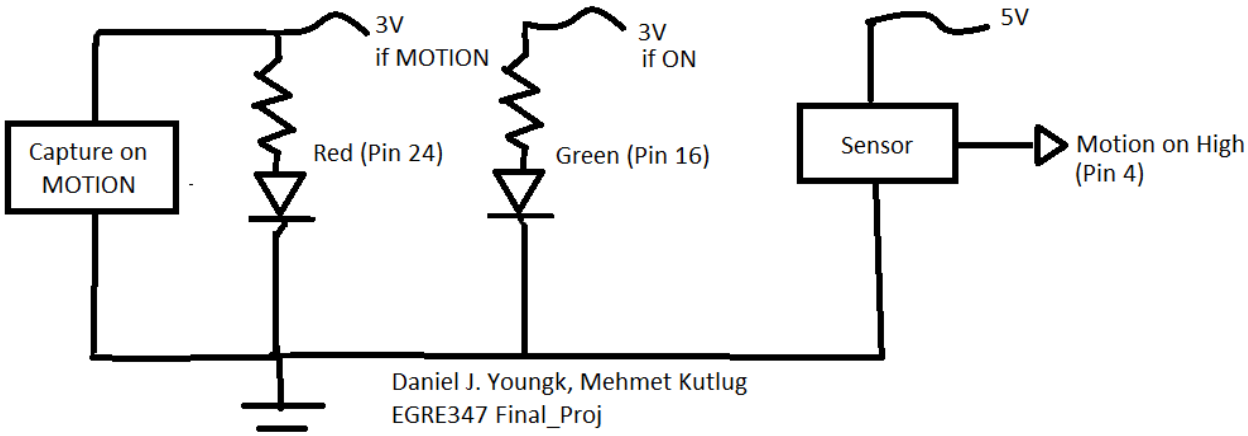
This module operates as a Mealy Machine. It transitions states based on the current input and current state. It can be abbreviated using the state machine diagram Figure 1.



[Figure 1] - Motion Sensor State Diagram

A software handler is used to shutdown the program upon the keystroke “ctrl-c” when operating through ssh terminal. More syntax for startup and shutdown can be found in the README file.

Hardware Introduction



[Figure 2] - Motion Sensor Breadboard Circuit

Part Number	Parts List		
	Description	Specification	Cost
1	Human Body Motion Module	HC-SR501 PIR	$7.99/3
2	GPS Message Receiver	VK-162 G-mouse	$13.99
3	Raspberry Pi	Scalable	~$15-$200

Part 1: A human body-tracking module produced by HiLetGo, which is a sensor manufacturing company in China. These are mass produced, and inexpensive.

Part 2: Serially interfaced GPS sensor that receives NMEA message protocol.
*The sensor is set to minimum delay and minimum sensitivity (~3 meters).

Part 3: Raspberry Pi 4 Model B, overkill for this application, but good for demonstration.

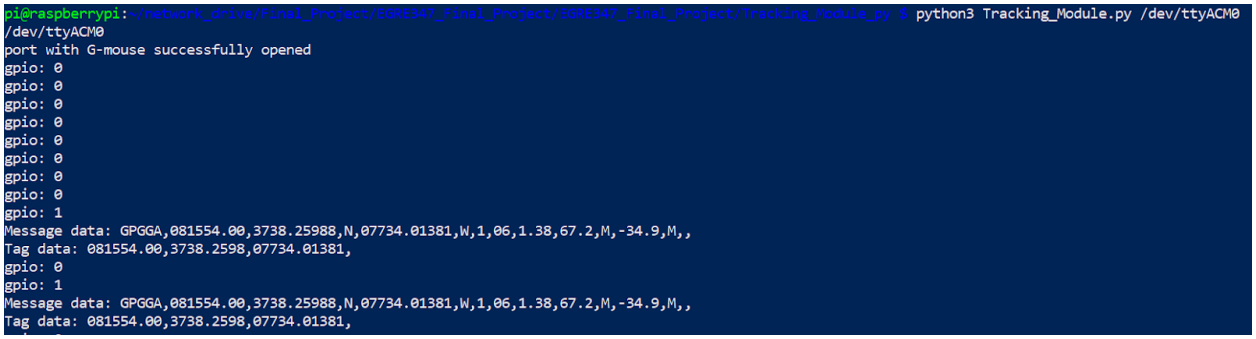
Running Program

C++

From the repository, open the Sparcer_cpp folder. From there, use make command in the console. Note that raspicam library has to be downloaded onto pi for compilation to succeed. From there, check path for GPS input using dmesg | grep tty (further instructions are located inside README.txt). Call program using ./sparce “GPS path”. While the program is running, if motion sensor is triggered, the red LED will turn on and photos will be saved with naming scheme of “MonthDayHourMinSec.ppm”. This name will then be recorded to a csv file named by default coordinates.csv. The program will open this file to append so that it doesn’t overwrite every time it’s run. Along with the name of the .ppm file, the GPS coordinates will also save to the file in GPGGA format, with each component comma separated. If the same process is applied but the program is called in the terminal with the phrase “nohup” in front of it, the program will continue to run in the background until the process is manually shut.

Python

Three main objects were developed for this device. The GPS_class is used to track live GPS coordinates, and format it appropriately. The second object is provided by the Sensor_class. A Sensor item has data members and member functions that initiate sensor, poll for input, or shut down all GPIO pins. The Sensor member functions’ main purpose is to send the GPS state machine the right information (i.e. Has motion been sensed in the past 3 seconds?).
The GPIO port used for sensor input is polled once every ~3 seconds. This is not optimal compared to C++, it is in fact more than 10 times slower. The tagging process is triggered by a rising-edge input as demonstrated in this test program output:



[Figure 3] - GPIO Input Rising Edge Demonstration

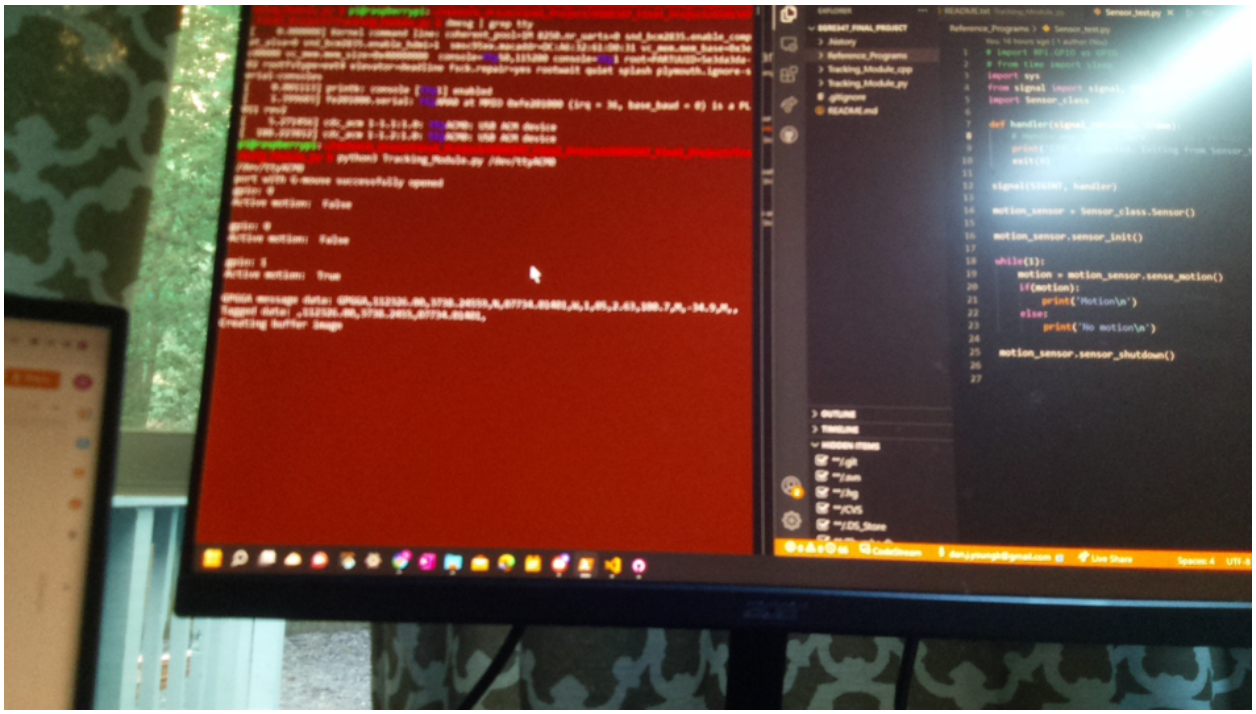
The tagged data is first delimited for columns by commas, and rows are separated by newline. The tagged data gets stored in “location_log.csv”, this file acts as a buffer. “location_log.csv” gets permanently deleted every time “Tracking_Module.py /dev/xxxxxxx” initiates.

Once an event is triggered, “location_log.csv” captures GPS and time data, while “buffer_image[n]” contains 1-3 camera images following the sensor trigger.

```
pi@raspberrypi:~/network_drive/Final_Project/EGRE347_Final_Project/EGRE347_Final_Project/Tra
cking_Module_py $ ls -l
total 1448
-rw-r--r-- 1 pi pi 290280 May  2 03:32 'buffer_image[0].jpg'
-rw-r--r-- 1 pi pi 276074 May  2 03:32 'buffer_image[1].jpg'
-rw-r--r-- 1 pi pi 287551 May  2 03:33 'buffer_image[2].jpg'
-rw-r--r-- 1 pi pi 286230 May  2 03:33 'buffer_image[3].jpg'
-rw-r--r-- 1 pi pi 292166 May  2 03:33 'buffer_image[4].jpg'
-rw-r--r-- 1 pi pi   6330 May  2 02:55  GPS_class.py
-rw-r--r-- 1 pi pi    265 May  2 03:33  location_log.csv
drwxr-xr-x 2 pi pi   4096 May  2 03:32  __pycache__
-rw-r--r-- 1 pi pi    924 May  2 03:17  README.txt
-rw-r--r-- 1 pi pi   2364 May  2 03:12  Sensor_class.py
drwxr-xr-x 2 pi pi   4096 May  2 01:46  Sensor_py
-rw-r--r-- 1 pi pi   1071 May  2 01:46  serial_read.py
-rw-r--r-- 1 pi pi   5688 May  2 03:07  Tracking_Module.py
```

[Figure 4] - Resulting Directory (~3 seconds of motion, ~20 seconds total runtime)

Although the program seems laggy with its 3 second gap between status updates, the image is taken within 3 seconds of sensing motion according to the image below.



[Figure 5] - PiCam image “buffer_image[0]” from demonstration video.

These example images remain in the submitted folder for reference.

CONCLUSION

A possible improvement may be to have a location algorithm that takes into account nearby satellite connection. There are several ways to improve upon an application-specific design for tracking location. The design for this module is not optimal, but portable for future projects. There are several places in the python scripts where functionality is not rugged. A lot of extra data is processed, and the whole process does not seem suited for fast-paces runtime applications, however Python is much easier to logically manipulate data with.

REFERENCES

Repository:
https://github.com/Dan-yelp/EGRE347_Final_Project

C++:
https://github.com/cedricve/raspicam
Library for using Camera with c++

Python:
https://docs.python.org/3/library/sys.html
.os for file manip

Linux:
https://www.cyberciti.biz/tips/nohup-execute-commands-after-you-exit-from-a-shell-prompt.html

Equipment:
https://www.amazon.com/dp/B07KZW86YR?psc=1&ref=ppx_yo2ov_dt_b_product_details
https://projects-raspberry.com/interfacing-hc-sr501-pir-motion-sensor-with-raspberry-pi/]\