Harnessing Spatial Processing to Find Cameras

Siddharth Kaza
Junior CE Student
Champaign, USA
kaza3@illinois.edu

Jaden Ambrocio Alcantara
Senior CS Student
Champaign, USA
ca23@illinois.edu

ABSTRACT

The aim of this experiment was to create a Raspberry Pi Spy detector camera. Students utilized the sensors on the Raspberry Pi's Sensehat, including accelerometers and gyroscopes, in combination with cameras. Their task was to apply their knowledge of the Inertial Measurement Unit (IMU) from the class to estimate the movement of a person holding the Raspberry Pi and, in conjunction with the RSSI values, determine the Raspberry Pi's location.

On Day 1, the hidden Raspberry Pi was transmitting packets constantly, however on the second day, a new challenge was imposed, in which the camera would only transmit packets on detection of motion, furthermore decoy cameras were added so that students would have to rely on their Spy Camera Finder, instead of their eyes. Our team decided to use a combination of the IMU sensor on the Raspberry Pi as well as the joystick that was present on the SenseHat, and in this paper, we will explain our strategy.

KEYWORDS

RSSI, Kalman Filtering, IMU, Step Detection, Raspberry Pi

ACM Reference Format:
Siddharth Kaza and Jaden Ambrocio Alcantara. 2023. Harnessing Spatial Processing to Find Cameras. In *Proceedings of Using Raspberry Pi's to Find Spy Cameras (CS437 Midterm Competition).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Our paper delves into the utilization of wireless sensing technology for the detection of hidden transmitting devices and cameras. Our primary motivation is to counteract the potential threats posed by individuals with malicious intent who employ wireless spy technology on unsuspected individuals. Wireless sensing takes advantage of the multi-path propagation characteristics of existing infrastructure, network protocals, and when coupled with additional hardware sensors such as an Inertial measurement unit (IMU) sensor, wireless sensing can pinpoint the presence of concealed objects within a given environment.

Given the recent advancements in localization algorithms and wireless sensor network systems, applicable to both indoor and outdoor settings, our paper endeavors to make a valuable contribution to the ongoing progress in this field.

2 SYSTEMS OVERVIEW

Several ideas were put forward during the initial project brainstorming phase. Over the first few weeks of the class, students gained knowledge about Raspberry Pi's SenseHat and how to utilize its various sensors within Python code. As time passed, the focus transitioned to the IMU or Inertial Measurement Unit, and how one could harness the readings from the IMU in order to estimate movement accurately. Beyond the IMU, the curriculum also covered topics like wireless signals and wireless sensing.

Initially, our project approach heavily favored relying solely on the IMU. However, it became evident rather quickly that the IMU's data exhibited significant sporadic behavior, making it an unreliable source of information. While some of the data was occasionally accurate, more often than not, it introduced substantial outliers into our dataset. Furthermore, the data analysis technique of double integration would no longer work since we would not be moving linearly in the challenge. If we were to only move in a straight line then it would give us accurate position and acceleration data, but once you add multiple dimensions its accuracy rapidly deteriorates.

In this implementation of our code, we use the joystick to indicate the direction that we would be moving, and we put a flag into our CSV so that when we were parsing our data in our analysis algorithm, we would account for the acceleration in that direction as our movement data. Furthermore, we coupled this with the step detection algorithm, so that we could accurately detect the number of steps that we took, and use the direction flag that we inserted in our CSV file to understand how far, and in which direction we were moving. to estimate the distance we moved we had Jaden (our walker), walk at a normal gait, and measure on average how large his footsteps were, this measurement was then used in order to account for the distance moved with each step using the step detection algorithm. We then process all the flags that we add into our code, in our analysis program.

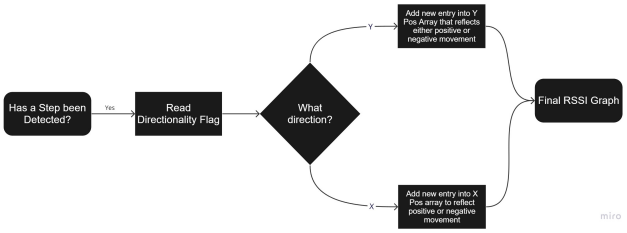

Figure 1: The workflow of our RSSI Analysis algorithm and how we graph position data.

3 DATA BENCHMARKING

Below are some of the outputs that we have from our data analysis algorithms. These are the outputs of the program of Analysis-Day1.py, which is our analysis algorithm for Day 1.

CS437 Midterm Competition, October 16–October 19, 2023, Urbana, IL
2023. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

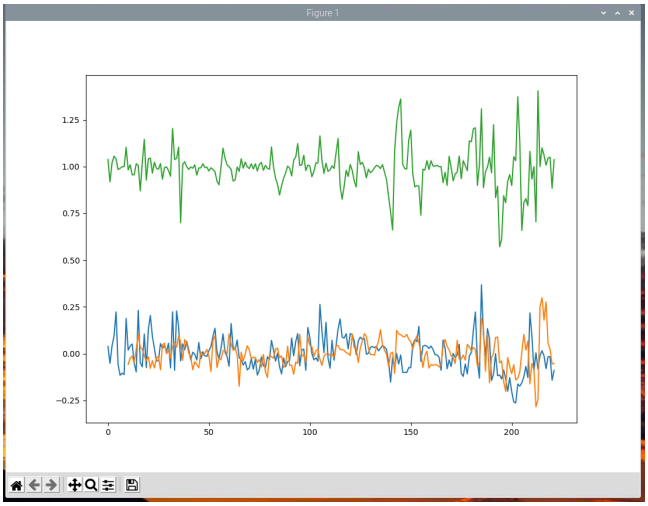

Figure 2: Raw Accelerometer Data from Day 1 Competition.

3.1 Accelerometer Data

Below is the graph of our raw acceleration data, you can see the z movements when Jaden was walking around the competition area, however, one issue was the spikes at the end. We consider the spike an outlier since this is when our time had run out, and Jaden swiftly turned so he could exit the competition area. In our data analysis processing, we account for this and make sure the record time stamps so the outlier data can either be filtered out in code or removed from the CSV.

3.2 Interpreted and smoothed Data

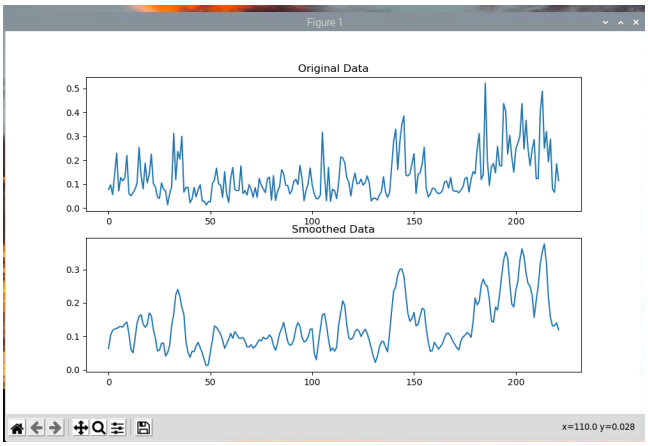

Figure 3: Accelerometer data interpreted with two different filters.

The set of graphs shows the process for reducing the noise of the raw accelerometer readings. First, we calibrate the raw reading using The top graph shows the normalized values of accelerometer read values. , the bottom graph is data smoothed using a Savitzky-Golay smoothing filter. Savitzky-Golay filtering is a signal-processing technique used to smooth or filter noisy data. It is particularly useful when dealing with data that contains random variations or noise, and the goal is to obtain a smoother representation of the underlying trends in the data. One can see the distinct difference in the graphs, and the data that is smoothed using the Savitzky-Golay filter is far more readable and usable.

3.3 Step Detection Algorithim

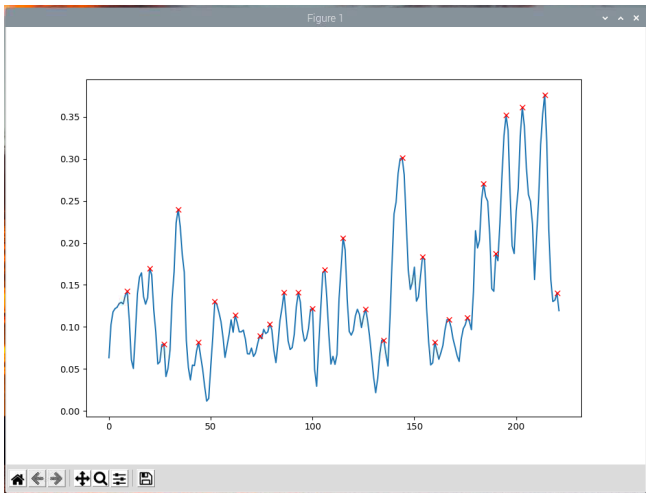

Figure 4: The result of the Step Detection Algorithm.

Now that we have much smoother, cleaner, and more usable data, we can start applying our analysis techniques to it. Our first technique was step detection, done by implementing the given code and modifying it to suit our needs with our parameters. You can see each of the X marks, which shows each time a step was taken. This number is incredibly close to the actual number of steps Jaden took during his search for the Pi. Since it is accurate, we can use this information, along with the directionality flag that we inject into our CSV file to be able to accurately track the position in which Jaden is walking. Each step is built upon the other to give us better, more accurate data.

4 EVALUATION

As previously mentioned, we used IMU data to calculate steps taken using a simple Step Detection Algorithm using z-axis values on the IMU readings recorded in a CVS file. Using the number of steps detected, we can plot a path in a graph and highlight different average RSSI received by the Raspberry Pi in the time frame in between each step. We plotted this processed data in a scatter plot with dots colored based on the RSSI range. For Day 1 of the competition, given the constant transmission of packets our processed data showed a clear path around the different zones of the environment. However, for Day 2 of the competition, given a poor walking strategy and inexperienced walker, our step detection had very noisy IMU data which affect the number of steps we could process.

4.1 DAY 1 RSSI VS Position Graph

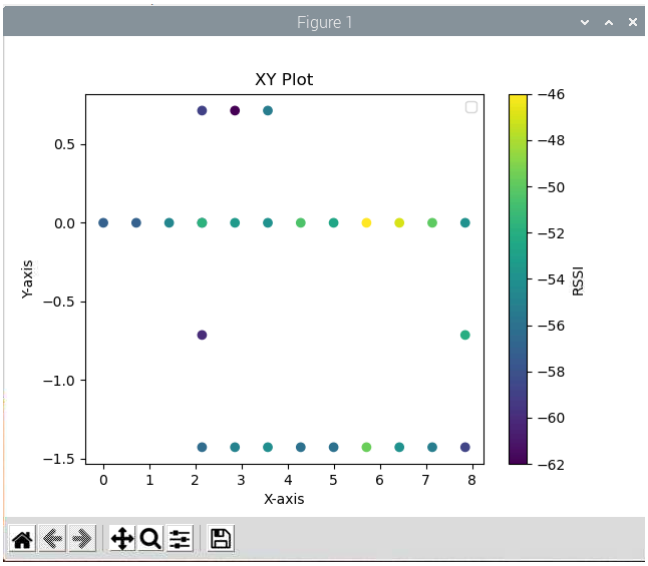

Figure 5: Step VS RSSI Graph.

Above is the final graph analyzed from Day 1. Coordination calculations used axes relative to the Raspberry Pi, therefore, in the following graphs, the x and y coordinates are swapped to the real world x and y coordinate of the competition space. Through all our data, we now have the number of steps, and thanks to the direction flag, we can obtain the axis that we are moving in. Each step also has an RSSI value associated with it. With our extrapolated X and Y step data, a graph is made with a plot showing you each step you took, and a color grading is applied to said graph, to show you the range of RSSI values you received at each step. Given that the Raspberry Pi can receive multiple packets with in the same step, we used timestamps at each direction change to calculate the amount of RSSI values read at each step. The final RSSI value displayed on the graph is the averages of those values in that step window.

In this graph from Day 1, we can see that we received the best signal when we walked forward and right. However, there were some flaws in our Day 1 walking strategy, as well as some missed inputs, hence you can see there are 0 Y steps, but when we inspected our CSV data, we concluded that during the walking path we had planned out in Zone 1, was when we had the best RSSI values, and this is also what led to our final guess of it being in Zone 1, which was the correct zone. We had a localization of 2.1. We looked into this error and we believe that the combination of missed inputs due to collecting RSSI and IMU data in the same thread along with inaccurate step detection data due to the walker not exaggerating each step. We will outline strategies for improvement in the following sections.

4.2 DAY 2 RSSI VS Position Graph

As mentioned in the previous subsection, poor planning and data loss lead to the reported y-axis being inaccurate. For Day 2, these issues were magnified by increased poor planning as well as a lack of understanding of the step detection algorithm. When the walker enter the environment, the walker held the Raspberry Pi steady as possible to not cause irregularities in the accelerometer readings since we had learned that it was already so immensely sensitive. However, we did not account how this would affect the Z-axis readings of the IMU data. We needed more Z-axis accelerometer movement for steps to be accurately detected, so our Day 2 data shown below is incredibly inaccurate and resulted in us getting the zone completely wrong. Later on in the lab, we tried moving with exaggerated Z movements at each step and found this to be a far more accurate measure of steps. Below is the graph of our Day 2 Data, and you can see how few steps appear on the graph, demonstrating the lack of z motion and failure of detection of a clear path unlike Day 1.

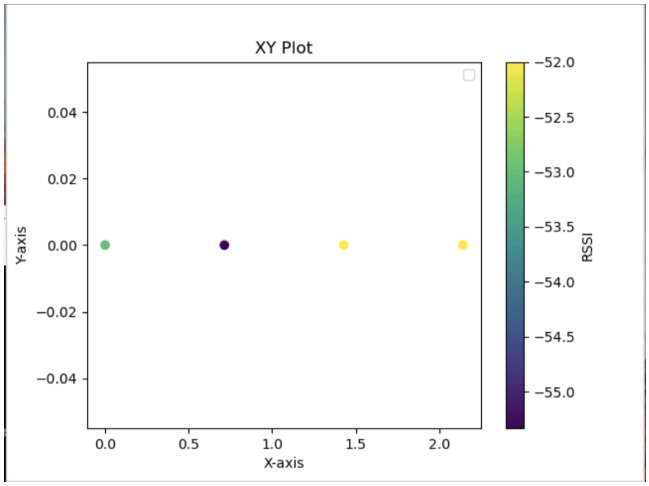

Figure 6: Day 2 Step Detection Graph

Another mistake that we made was running the wrong code for Day 2, we meant to run the Day 2 code we created, in which we had programmed an LED to change color depending on the RSSI value that it received, and as we get better and worse signal it would change accordingly, giving us better localization, however we ended up running our Day 1 code during the competition, which gave us absolutely no visual feedback, meaning that the walking strategy was rendered irrelevant, and we were just trying to walk around in a panic over-relying on communication with the observer to move around the environment. Due to the limited and inaccurate processed data, we were left estimated making an educated guess on the location of the Raspberry Pi based when the Pi received the burst of pack and an estimation of the position the walker was at that given time frame.

Lastly, we overlooked the packet transmission delay, which led to our misinterpretation of the zone's location. Additionally, we failed to consider the camera's field of view. When Jaden crossed the boundary between Zone 1 and Zone 2, we received a burst of packets. However, we later realized that this burst resulted from Jaden's movement in front of the camera before making the turn, and the packet transmission was simply delayed.

In hindsight, running the Day 2 code, which relied on RSSI strength and provided a visual indicator, our walking path could have been adjusting in real time leading to a better estimation even with limited data. With this approach, we could have completed a walking path, established a baseline when we received an RSSI packet burst, and then narrowed down the location.

4.3 Improvements and Changes

Given the failures and inaccuracy of Day 2's processed data, we modified our code writing a new scripts, CollectionFinal.py and AnalysisFinal.py. In the scripts, we combined our improved RSSI LED indicator code. The main improvement was adding a pixel color indicator for increasing or decreasing reading along with our previous color indicator based on set ranges of good and bad RSSI values. Lastly, the new collection script uses threads to collect data separately which meant our analysis had minor changes. However, the main approach of using step detection remained the same.