Hospitably

An Indoor Localization Application

By

Arad Saebi, 404-498-230

Shengkang Chen, 004-498-661

Gary Bui, 904-672-175

Victor Hsiang, 904-150-802

Introduction

Hospitably is a web application that is designed to create an automated hospital experience. The initial motivation behind this project was that hospitals can be very confusing environments for visitors, and there may not be enough staff or resources to help and guide them. Furthermore, there are not many public applications that allow for indoor localization. Hospitably will not only provide a reliable source of navigation for users while they traverse the endless monotonous hallways, but also has the potential to streamline a patient's visit to the hospital, including checking in, accessing patient information, and monitoring patient activity.

The focus of our project was within the scope of navigation by utilizing WiFi address and signal strength mapping. Furthermore, we used an IMU to enrich our navigation algorithm, as it provides data from which we can determine the distance a user travels. The combination of WiFi localization and IMU tracking gives an accurate representation of where the user is standing while they are indoors. Outside the context of hospitals, our project has the potential to track or localize inside any building with WiFi connectivity. As long as the reference data points are mapped out, it is simple to switch out the map in the cloud database and run an indoor localization system anywhere.

Overview

The navigation is based on Kalman Filter which takes into account of both measurement and propagation. In our case, we use the Intel Edison to Wifi scan and collect reference data and motion sensing using 9DOF mounted on shoes to collect propagation data. Using a probabilistic method, we are able to compare a user's WiFi scan to our reference map, hosted on the Firebase

Database, to determine location. Meanwhile, 9DOF can identify user's footsteps and turns to estimate user's change in location.

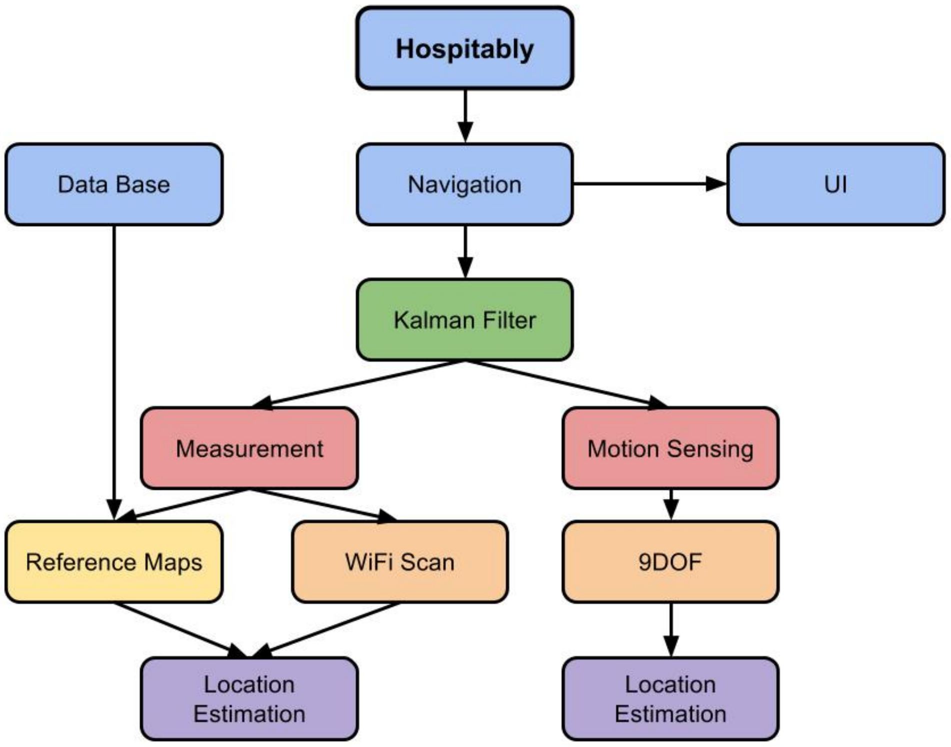


Figure 1: the Flowchart of Hospitably

Although both WiFi scan and motion sensing can provide localization results. We cannot depend solely on either one of them. Motion sensing can provide constant updates on estimated location. However, its error will accumulate, which is unreliable for long-term localization. WiFi scan can can provide localization results without accumulated errors, but it takes a period of time, about 8 seconds in our case, to scan and process the data, which is unacceptable for simultaneous localization. Also, if users are not stationary while scanning, its accuracy will decrease noticeably. The Kalman Filter allows us to achieve simultaneous localization using both WiFi scan and motion sensing. We use Kalman Gain(K) to weight both localization results to get better accuracy. For example: localization result will rely more heavily on the motion sensing localization result as users move faster or it will rely more heavily on WiFi scanning results if the user is moving slower.

Framework:

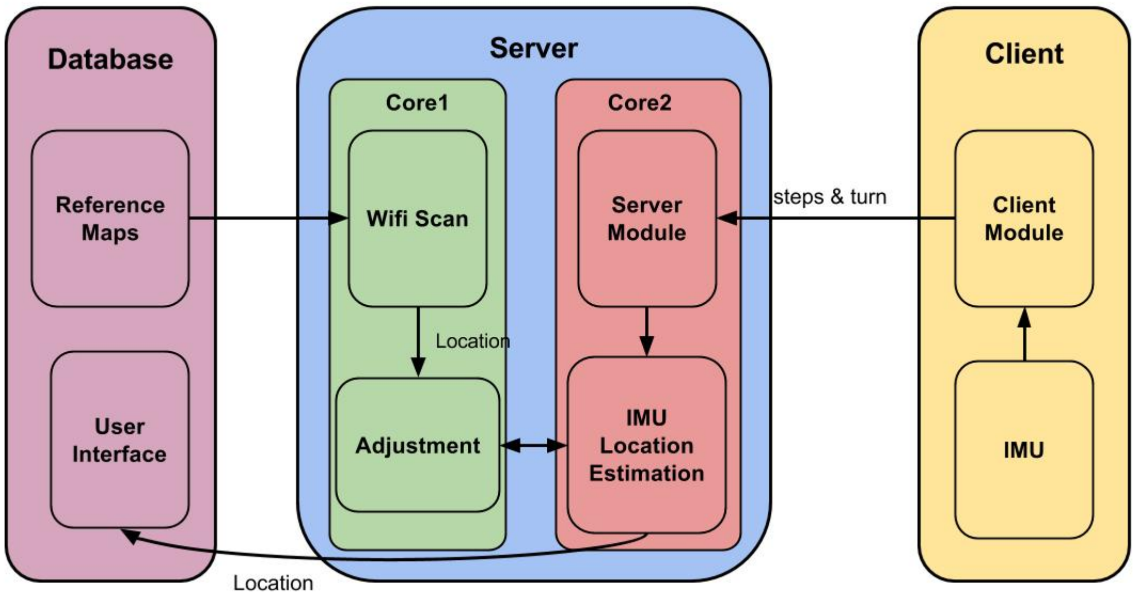


Figure 2: The Framework of Hospitably

There are three main components of the system: database, server and client. We use Firebase Database as our database, the Intel Edison board that perform WiFi scan as the server and the 9DOF as the client. In the initialization, communication between these components will be set up; the server will ask for user's height to calculate his/her foot stride and orientation then server will request reference map from the database and perform WiFi scan to determine the starting location. During the localization process, the IMU will send the user's step count and turning periodically to the server using client module and server module. The server will do parallel programming: one core receives the data from IMU and calculates new estimated location based on previous location and user’s movements, then it will send the location data to the user interface through database. The other core performs WiFi scan localization at the same time. Once WiFi scan localization is finished, it will adjust the current estimated location based on Kalman Filter to prevent accumulated errors.

Motion Sensing

We used the 9DOF IMU Sparkfun Kit for Intel Edison for our IMU tracking. The 9DOF uses the LSM9DS0 9 Degrees of Freedom for full range motion sensing. This features a 3D digital linear accelerometer, 3D digital gyroscope, and a 3D digital magnetometer. For the purposes of our indoor localization application we were interested in tracking the motion of walking in real time, or at least as close to real time as possible. The IMU should be securely strapped with the Intel Edison chip facing upwards to the front of the user’s foot in order to minimize noise while walking. We started by using the open source code from “Musings from Stephanie,” a professional from Intel and hobbyist, who set up and configured the IMU in python. This allowed us to gather raw data. To process the data, we initially attempted to implement Magdwick’s filter to convert the raw data into quaternions which can be represented by euler angles. However, for the purpose of this project we decided that took too much processing power and was unnecessary. Since our project allows IMU tracking localization to be corrected by WiFi positioning we are able to accomplish our goal even without the most sophisticated data processing algorithm. We then decided to go in a different direction and use a pedometer algorithm to determine the change in distance. Our IMU tracking is broken into two parts: step counting and turn detecting.

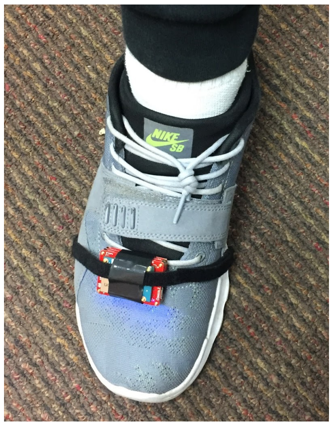


Figure 3: Photo of how the IMU should be secured

One overarching assumption we started with is that when a person is walking, according to the IMU frame, there should be a repetitive pattern in the Z direction. Therefore, we set the IMU device so that the Intel Edison chip is facing upwards, as shown in Figure 3. From our data we were able to determine that this is true, however, when a person walks there is no one distinctive peak to indicate one step. Alternatively, there is a zero-velocity update when the person steps with the foot not attached to an IMU. We are able to use this zero velocity update in order to determine when a person is beginning a step and ending their step. It is important to determine the beginning and end of a step because the IMU gathers data at discrete time intervals, so there are multiple points at which acceleration will satisfy the threshold that correspond to the same step. By gathering training data with different subjects, we were able to determine acceptable thresholds to indicate that a step was being taken and then used the zero-velocity update to indicate that the step was ending**.** We determined the thresholds by analyzing the training data in MATLAB. We wanted the distinction between the IMU's noise during zero velocity update and an actual step to be very distinctive. Another aspect of the step counter is that we did not want to mistake someone tapping their foot for steps. This is done two ways: the zero-velocity update and peak accelerations. In order to be registered as a step, the program must recognize that a step is not currently being observed, the acceleration in the Z direction must reach the threshold, and the step must end at the zero-velocity update threshold. In the case of tapping feet, the zero-velocity update is not achieved and peak accelerations are generally not achieved either. The pseudo-code for our step counter is:

```
steps = 0      #counts the number of steps
isStepping = false     #this is a variable to determine if we are in
the middle of a step
a0 = Acceleration in Z direction @ t = 0
      if acceleration < threshold and isStepping == false
            steps++
            isStepping = true
      if acceleration is within "zero velocity update" threshold and
isStepping == true
            isStepping = false
```

We take a similar approach to detect if the person is turning. For the turning detection we looked at the gyroscope output for the Z axis, since turning left or right results in a rotation in the Z axis. We found that a negative change in Z gyro signals a turn to the left and a positive change in Z gyro signals a turn to the right. We also noticed that when a person walks there is noise from their foot shaking, so we set a threshold for when the person turns at least 45 degrees. Since we are working in Engineering IV, we know the user will be confined by the hallways and should only turn in 90 degree intervals. In order to make the code more robust we added a high pass filter to the gyroscope to achieve a zero-angular velocity update. The pseudo-code is as follows:

```
turn = ""
startTurn = false #this is a variable to determine if we are in the
middle of a turn
If gyroZ > gyro threshold and startTurn = = false
      turn  = right
      startTurn  = true
else if gyroZ < negative gyro threshold and startTurn  == false
      turn = left
```

```
                startTurn = true
        if gyroZ is within zero angular velocity update
                startTurn = false
```

We send both the number of steps and if the user turned to the server where they will be used to calculate position. The benefit of analyzing the sensor data in only the Z axis is that the IMU can be oriented in any direction if the Intel chip is oriented upwards.

WiFi Scan Localization

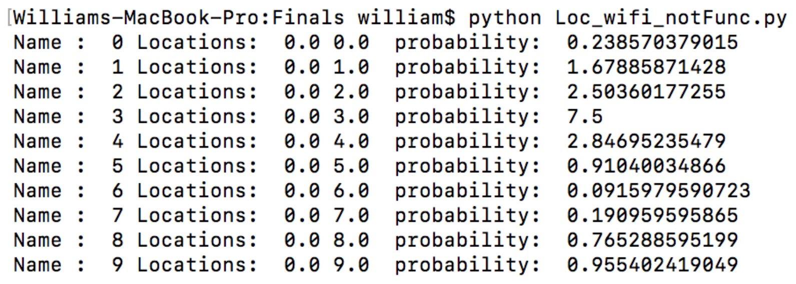
In order to perform localization based on WiFi scan, it required two parts: WiFi reference maps and WiFi scan results. For the purpose of wifi mapping we used a shell script to collect all the wifi information that we needed such as ESSID and power in units of dBm at a single point and ran it in Intel Edison. We later devised the 4th floor of Engineering IV into 61 points with each 2 points being distanced at around 10 ft and ran this script to collect WiFi strength datas through those points to have a complete reference map. We later repeated the whole process 10 times to in order to obtain the mean and standard deviation of wifi signal strength at every reference point on the reference map.

The shell script we used for gathering WiFi information was the following:

```
        #!/bin/bash
        FILENAME="scan_results.txt"
        iwlist wlan0 scan | grep 'Address:\|Signal' | sed 's/.*Address: //;
s/.*\([0-9]\{2\}\) dBm/\1/' | sed 'N; s/\n/ /' | sort > "scan_results.txt"
```

The scan command will approximately takes 5 seconds to run and as we later on found out is the most dominant term in terms of delay for WiFi scan localization.

To estimate the current location given WiFi scan results, we need to first go through the scan results and reference points in the reference map to find the matched MAC address between reference points and scan results. For each matched MAC address, we can calculate the probability for each matched MAC address at each reference point based on its mean and its standard deviation. Then, we calculate the sum of the probabilities of each matched address on each point. After that, we find the weighted centroid of the cluster set of points with highest probabilities.



```
Williams-MacBook-Pro:Finals william$ python Loc_wifi_notFunc.py
Name :  0 Locations:  0.0 0.0  probability:  0.238570379015
Name :  1 Locations:  0.0 1.0  probability:  1.67885871428
Name :  2 Locations:  0.0 2.0  probability:  2.50360177255
Name :  3 Locations:  0.0 3.0  probability:  7.5
Name :  4 Locations:  0.0 4.0  probability:  2.84695235479
Name :  5 Locations:  0.0 5.0  probability:  0.91040034866
Name :  6 Locations:  0.0 6.0  probability:  0.0915979590723
Name :  7 Locations:  0.0 7.0  probability:  0.190959595865
Name :  8 Locations:  0.0 8.0  probability:  0.765288595199
Name :  9 Locations:  0.0 9.0  probability:  0.955402419049
```

Figure 5: screenshot for part of the list of sum of the probabilities at point 3

The above figure is a screenshot for part of the list of sum of the probabilities when we do the WiFi scan at “Position 3.” It is clear that “Position 3” has the highest probability and the positions near “Position 3” also have fairly high value. After we take the centroid of of the cluster set of points with highest sum of the probabilities, “Position 1” to “Position 5” in this case, our estimation is [0, 2.79] and point 3 is at [0, 3]. This is a difference of 2.1 feet and is very accurate for indoor localization.

User Interface

While we initially had planned on building the front end of our application in Android Studio, complications with accessing WiFi information as well as endless obstacles in environment set-up led us to consider other options. We discovered a web application development framework called Ionic 2. Built on top of AngularJS and written in Typescript, Ionic 2 provides developers with a pretty high level of abstraction in terms of development. The structure of the code is intuitive and allowed us to put together a simple user interface without many complications.

Our navigation user interface is very intuitive: the user is provided a map of the area they are localizing. Reference points have been mapped out so that each 10 foot distance represents a new position. The system will report to you which position you are in. In this way, you can visually see what part of the building you are in and find your destination that way. While we had envisioned a more user-friendly and intuitive interface, we did not foresee the other aspects of our project bottlenecking the front-end development process. Oftentimes, the development process required for us to just be able to pass information throughout our system and debugging always took priority over user interface.

Communication

Our backend is handled by a combination of python and the online database Firebase. Firebase allows for access across multiple platforms and offers a plethora of API capabilities for most application development languages. We had to connect to Firebase via three different methods. First, our reference map was parsed into a dictionary in Python and uploaded into our Firebase through the command line. Next, we had to be able to send our localization data into the Firebase so it could subsequently be displayed in our Ionic 2 application, which was the last form of communication we had to establish with the database. Firebase allowed us to communicate between many components of our system and proved to be a versatile tool.

The Communication between Intel Edison boards is fairly straightforward. Client (IMU) and Server (Wifi scan) setup the connection based on handshake protocol. Then, IMU will keep sending steps counts and turns to the server periodically.

Problems encountered and workarounds

We initially had wanted to be able to map our reference points based on the physical location of our network access points. We soon realized how difficult it would be to actually figure out where the access points were. Furthermore, for our application to be scalable and robust, we would need to be able to create our map without knowing where our access points were. We decided to use a combination of MAC addresses and signal strength to map our reference points to predetermined landmarks in our physical environment. Furthermore, we decided that we could use the movement of the IMU to track the velocity of our user and add to the precision of our localization algorithm. The motivation to use a relative positioning system came from the article *EZ Localization*, which outlined the design of the "EZ Localization" application. We are using a user's IMU data to calculate how fast they are moving and when they make turns. We then calculate the person's position based on the initial WiFi signal that they received and how fast they're moving.

Originally we tried to map the entire 4th floor with 10 points. We even tried to find a person's location with 30 reference points on the map. However, the low accuracy of the results convinced us to sample more points and we finally came across 61 points which gave us a robust and fairly accurate results.

This segues into another issue we ran into, which was finding a reliable source of data from the IMU to receive a usable velocity and angular velocity in our calculations. We experimented with different ways to determine the user's velocity, namely their actual acceleration along a plane and their stride count. Because we felt that the IMU's acceleration readings were not reliable enough for us to derive a usable velocity, we instead used stride count.

We did this because the impulse response behavior of a step in the IMU was much easier to identify than that of the constant acceleration in any given plane.

Some of the other limitations for our IMU tracking are that we do not know the user's initial orientation.We attempted to solve this by creating a compass based off of the IMU's magnetometer, but because we are indoors the magnetometer does not give accurate readings. Another approach we tried was to smooth the gyroscope data to calculate angles, but we could not create or find an algorithm that was computationally inexpensive for our system. Instead of running the risk of slowing down our system, we simply prompt the user for input on their initial orientation and detect changes in the Z gyro to determine if we made a turn or not. The benefits of how we approached the IMU tracking are that it is fairly accurate, handles in almost real time, and it is more user friendly because the IMU does not have to be worn in a very specific way.

A common issue we ran into while testing and debugging was the presence of interference from other devices nearby. When we had two team members testing the data during after hours when there were less people and devices in the halls, our WiFi readings tended to be most accurate. However, as soon as our two other teammates entered the vicinity, it would oftentimes throw off the localization readings. We adjusted for this with the introduction of the IMU in our system, allowing for location tracking based on steps taken on top of the WiFi localization. While this led to the dilemma of how we wanted our system to weigh the readings of each component, it provided us more flexibility and robustness in terms of providing an accurate representation of the user's location.

Another problem is the processing time for WiFi localization. Initially the system took about 5 seconds to scan the WiFi signals and 20 seconds to process. This too long for real time indoor localization. Instead reading from text files of the reference points, they are all load into dictionary type variables in initialization for faster processing. We also cleaned up the files inside the Edison Board. Processing speed has improved dramatically; now it only take about 3 seconds to process instead of 20 seconds.

As we predicted, setting up what we needed for our Android application had a lot of obstacles and pitfalls. We soon learned that it would be difficult to use the WiFi Manager API inside an emulator, as typical emulators abstract the internet connection as a cellular connection. It would be impossible to identify actual WiFi networks inside the emulator, and so we needed to get our hands on an actual device. Regardless, the source code we found online did not do what we expected and we were forced to move on in developing other aspects of the Android application. We eventually had to rethink the flow of our system as Android development became more complex. We arrived upon a web application platform called Ionic 2. Ionic 2 provided an easy, intuitive way to build a simple user interface and link it up to our Firebase database.

One of our greatest obstacles was the use of so many programming languages that many of us had never used. This only served to cause confusion in how we would end up linking the different components of our system together. Eventually, after a lot of research and experimentation, we were able to simplify the relationships between the different parts of our system and link everything up. We discovered the risk of having every member working on different components without enough communication. It was very easy for us to start specializing in our portions of the project and falling behind in understanding the other parts of the system. This would prove to be detrimental later on in the course of our project when we needed to start connecting our components. When this happened, debugging and testing our code got more complex and convoluted. The combination of all our small bugs and working through those bugs took up a lot of our bandwidth as a team. However, we soon picked up the pace and started spending more time working together. This ultimately facilitated a more efficient workflow for our team.

The challenge we have for the framework is the balance between communication and process. Originally, we wanted the client (IMU) to do the calculation determine the user's movements on x-axis and y-axis based on steps and turns, then send it to the server to calculate current location. We noticed that the large amount of informations needed to send to server would sometimes cause the IMU to lag and cause miscommunication or inaccurate data. This dragged down the accuracy of the localization results. Now, the client (IMU) only needs to send the user's steps and turns every 2 second or when a turn is made, and the server takes cares of the calculation of the movements. This eliminates the miscommunications, but there are lots of processing burden on the server side.

Experimental verification

For the IMU the data is outlined in Figures 6-9.

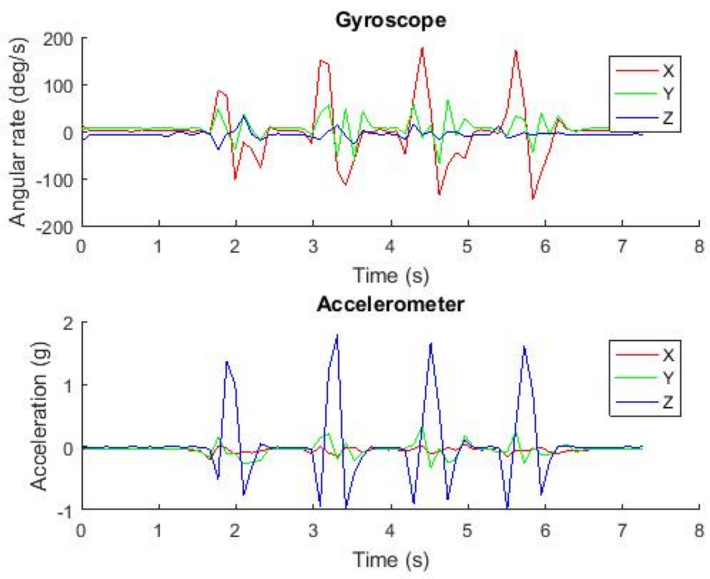


Figure 6: Walking Testing Data

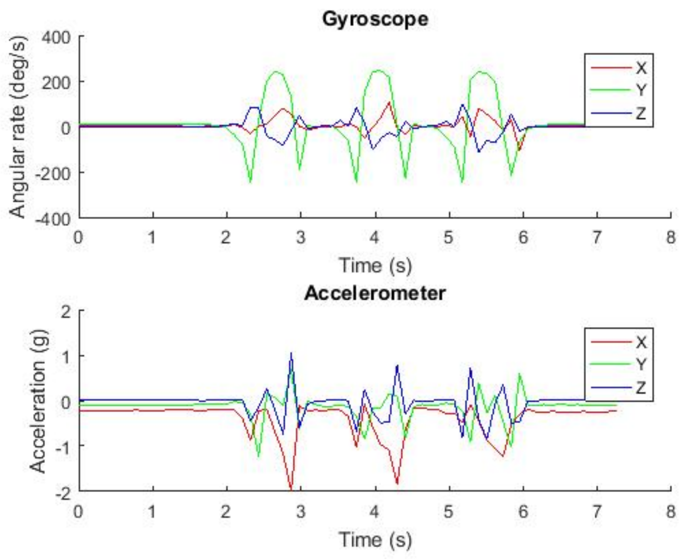


Figure 7: Actual Walking Data

Figure 6, on the left, is our testing data in which we simply moved the IMU up and down four times to simulate walking. From that data we were able to see how the IMU should ideally act, and we were able to determine the zero velocity update thresholds. Figure 7. on the right. is our actual training data when the IMU was strapped to the user's foot and they took three steps. We can see that there are still clear sections in the Z direction that indicate that steps were being taken. This data allowed us to determine our step thresholds. By detecting when the step starts and ends we are able to categorize the data as steps.

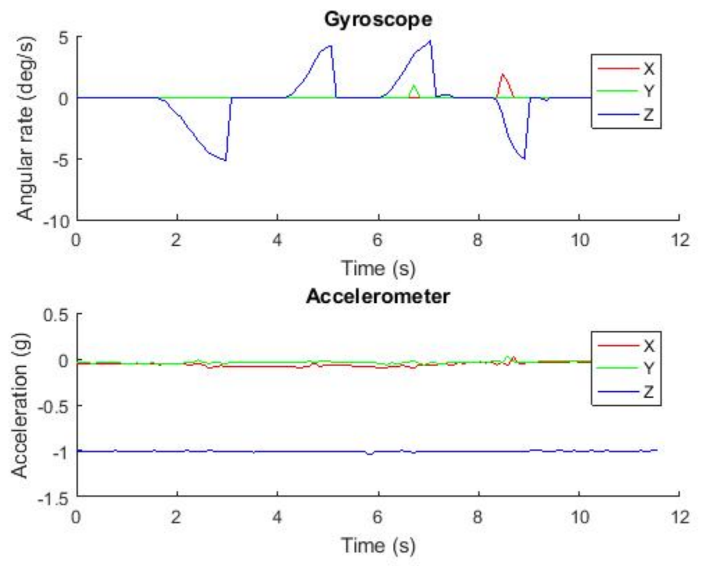


Figure 8: Turning Testing Data

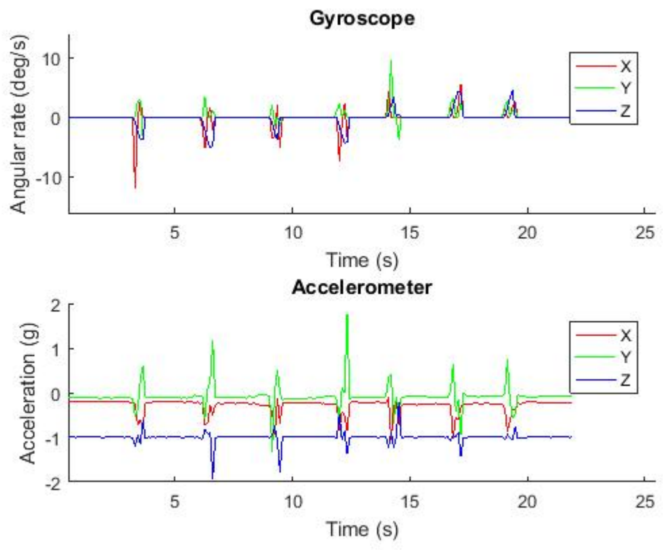


Figure 9: Actual Turning Data

Figure 8, on the left, is the turning testing data where we had the IMU flat on the table and simply rotated it left, stopped, back to center, stopped, right, stopped, and back to center. From the testing data we were able to determine that turns,left and right, can be defined as changes in angular rate in the Z axis, where negative changes indicates left and positive changes indicate right. Figure 9, on the right, shows the training data where the user turned to the left four times and then to the right three times. From the graph it is clear that the relationship still hold and should be reliable with the right thresholds.

Using MATLab we were able to create a simulation environment using Kalman filter to test our positioning algorithm. Figure 4 is the output of our algorithm mapped over the fourth floor of Engineering IV. The green dots are the actual physical location of the reference points, which are unknown to the user, while the blue line is our simulation of how the user moved through the building. Since the map is not to scale and is outdated, there is some mismatch of where the green dots are mapped. However, it still shows that our algorithm is capable of constructing a pathway with acceptable process noises and measurement noises.

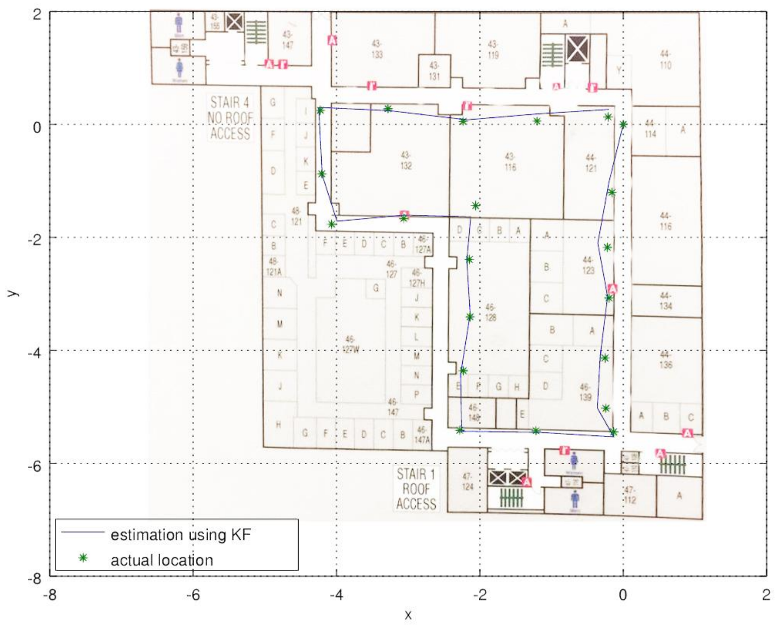


Figure 10: Output of Kalman Filter mapped to Engineering IV's fourth floor

Moreover, we did extensive testing on WiFi scan localization. For stationary scanning, we did several complete runs on 4th floor in Engineering IV building. Under low interference, we have fairly accurate results.

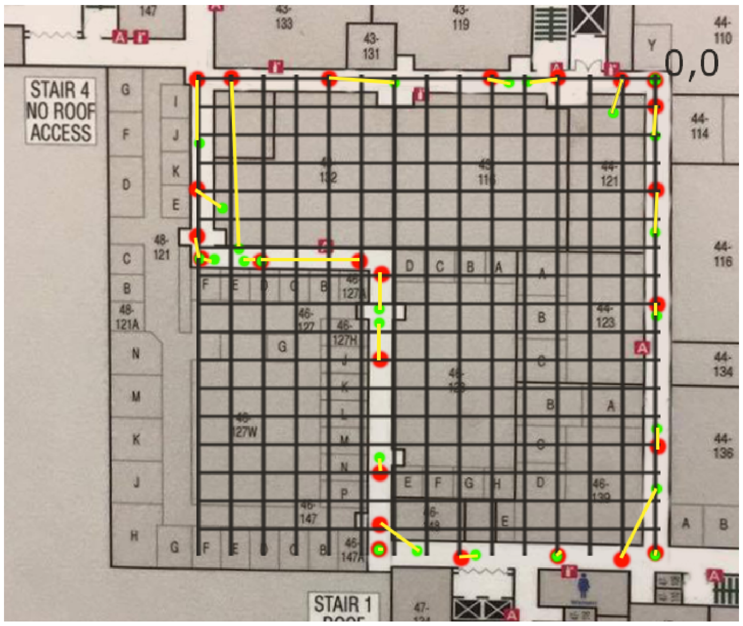


Figure 11: complete test run on 4th Floor Eng4 for stationary WiFi scan

The above figure is one of the complete test runs on 4th Floor Engineering IV building for stationary WiFi scan localization under low interference, where green dots are localization results, red dots are groundtruth, yellow lines are the deviation. As indoor localization, this is fairly accurate. We also did several testing runs on non-stationary scan, we noticed that as user moving faster the localization results became less accurate.

After the integration with database, WiFi scan and IMU, we created the completed system of *Hospitably*. We did several testings on the system, and Figures 12 and 13 show that the system can perform fairly accurate localization with low interference and clear footsteps detection.

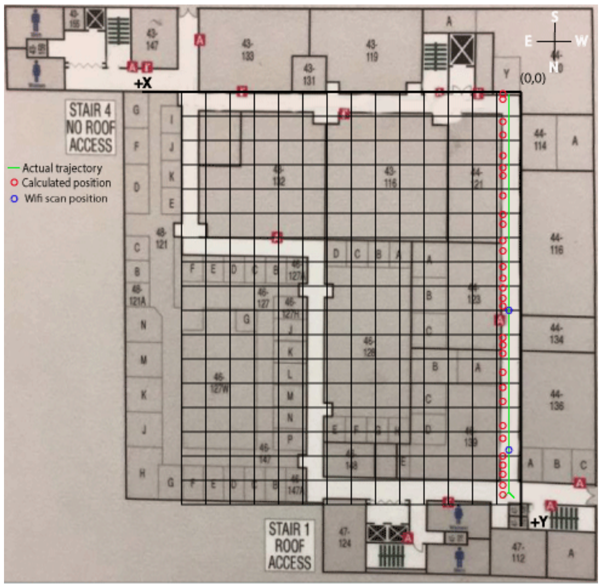
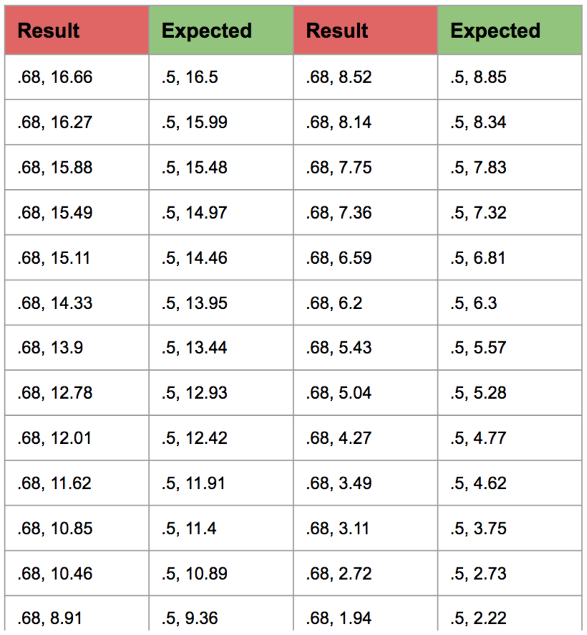


Figure 12: Trial 1 testing complete system



Result	Expected	Result	Expected
.68, 16.66	.5, 16.5	.68, 8.52	.5, 8.85
.68, 16.27	.5, 15.99	.68, 8.14	.5, 8.34
.68, 15.88	.5, 15.48	.68, 7.75	.5, 7.83
.68, 15.49	.5, 14.97	.68, 7.36	.5, 7.32
.68, 15.11	.5, 14.46	.68, 6.59	.5, 6.81
.68, 14.33	.5, 13.95	.68, 6.2	.5, 6.3
.68, 13.9	.5, 13.44	.68, 5.43	.5, 5.57
.68, 12.78	.5, 12.93	.68, 5.04	.5, 5.28
.68, 12.01	.5, 12.42	.68, 4.27	.5, 4.77
.68, 11.62	.5, 11.91	.68, 3.49	.5, 4.62
.68, 10.85	.5, 11.4	.68, 3.11	.5, 3.75
.68, 10.46	.5, 10.89	.68, 2.72	.5, 2.73
.68, 8.91	.5, 9.36	.68, 1.94	.5, 2.22

Figure 13: Experimental data from Trial 1

The above figures are the results from Trial 1 testing. We walked through the entire hallway of 4th floor Engineering IV building. The results show that the whole system is functional for localization: IMU provided reliable datas for footsteps and WiFi scan results was able to adjust current location estimation. The deviances on localization results on this trial are within 5 feet, which is very accurate for indoor localization.

Future plans

In order to make the IMU code more adaptable, we would like to make more use of the sensor fusion to gain more accurate angular movements. For example, we can measurment other signal strength like setting up bluetooth beacons instead of depending solely on WiFi signals.

As so much of our project was based on the actual technology at work with the WiFi and IMU localization, it was difficult to sink time into designing the user interface to reflect what we wanted in our final product. In the future, we would like to implement the actual hospital application we had initially planned to make. The goal was to streamline an everyday person's day at the hospital, including signing in, access to patient information, and health monitoring, on top of the indoor localization. Ultimately we want to create an immersive, hospitable hospital experience.

Another improvement we would like to pursue is improving the algorithm used to track rotation. We can definitely choose better thresholds for decision making and fine-tune our entire setup, including how we choose to physically fasten the IMU. A huge issue that we came across during testing was that as the IMU wobbled ever so slightly, it would create enough noise to affect our readings significantly. Understanding this, we would like to go back and collect more data in order to improve our algorithm.

In general, we feel that the more data we can collect, the better our localization algorithm will become. With more data points, we can have better averages for our reference map and ultimately just get a better feel for how we can better weigh the expectations of our WiFi localization values against the expectation of our IMU values.

Finally, this entire system currently takes at least two Edisons and realistically two laptops at the a time. We would hope to be able to do more research into how we can get our application on a mobile platform. The biggest obstacle we will probably have to face will be accessing WiFi information from the each device's native system.

References

1. "How to Create a Wi-Fi Android App Scanner." *YouTube*. Start Android, 21 Sept. 2016. Web. 07 Dec. 2016.
2. Admin. "How to Create a Wi-Fi Scanner App for Android." *FanDROID.info*. N.p., 05 Oct. 2016. Web. 07 Dec. 2016.
3. "SubPos - A "Dataless" Wi-Fi Positioning System." *YouTube*. Linuxconfau2016, 05 Feb. 2016. Web. 07 Dec. 2016.
4. Chandra, Ranveer, Jitu Padhye, and Alec Wolman. "BeaconStuffing: Wi-Fi Without Associations - Microsoft Research." *Microsoft Research*. IEEE HotMobile, 1 Feb. 2007. Web. 07 Dec. 2016.
5. Chintalapudi, Krishna, Anand Padmanabha Iyer, and Venkata N. Padmanabhan. *Proceedings of MobiCom '10 and MobiHoc '10: The 16th Annual International Conference on Mobile Computing and Networking and the 11th ACM International Symposium on Mobile Ad Hoc Networking and Computing, Chicago, IL, USA, 20-24.09.2010*. New York: A.C.M., 2010. 20 Sept. 2010. Web. 7 Dec. 2016.
6. Faragher, Ramsey. "Understanding the Basis of the Kalman Filter Via a Simple and Intuitive Derivation [Lecture Notes]." *IEEE Signal Processing Magazine* 29.5 (2012): 128-32. Sept. 2012. Web. 7 Dec. 2016.
7. Stephanie. "Sparkfun 9dof Block." *Musings from Stephanie*. N.p., n.d. Web. 23 Mar. 2017.