Sping 2018 CS755 Project Report Acoustic Sensing

Mingrui Han
George Mason University
4400 University Drive
Fairfax, VA
mhan8@gmu.com

Joshua Lilly
George Mason University
4400 University Drive
Fairfax, VA
jlilly3@gmu.com

ABSTRACT

More and more IoT devices are emerging into our daily life, aiming to ease our living styles. Devices such as smart televisions and smart fridges will have more functionality to provide a better user experience. A traditional controller that uses buttons cannot meet the expectation of the usage in all devices. In this paper, we implement AAMouse [13], a system that tracks devices using acoustic sensing. It enables users to use their mobile devices as a mouse to draw freely in the air. AAMouse achieves this with inaudible sound waves and Doppler shift to track the device movement. Human beings cannot hear sounds above 17kHz. However, these frequencies are supported in commercial devices. AAMouse takes advantage of the unused frequencies to do the tracing. Also, AAMouse does not require specialized hardware to operate. We implemented AAMouse in two platforms, on a PC using MATLAB and an Android tablet. To refine the tracking result, we applied Maximum Ratio Combining and a signal smoothing filter to achieve better accuracy. Then we conducted the evaluation experiment. The actual error is larger than the AAMouse paper by a significant amount. We also discuss the possible reasons for the error and possible solutions. In addition, the differences between our implementation and the original work is discussed as well.

Keywords

AAMouse, acoustic sensing, inaudible sound wave

1. INTRODUCTION

In this work, we implemented AAMouse, a system that utilizes acoustic signals and Doppler shift to track the device movement. AAMouse uses inaudible sound waves above 17kHz. Research indicates that human beings cannot hear the sounds above 17kHz. However, current commercial speakers support those frequencies. The unique property of AAMouse is that it can operates in mobile devices and it does not require any specialized hardware. A system that uses mobile devices to track its movement can offer a variety of usages. User would be able to control appliances from a distance by carrying their mobile devices. Such system can also enables the user to have a better user experience than the traditional remote controller. Moreover, mobile devices have IMU sensors built in, which can also be used to aid the accuracy of the tracking result.

The original AAMouse uses Short Time Fourier Transform, Doppler shift, Maximum Ratio Combining (MRC) Kalman Filtering and a Particle Filter using Monte Carlo Simulation. Our system is different than the original work.

Our system sends the sound wave and analyze the received signal using Doppler shift. The frequency shift will give information to calculate the velocity. We then used the velocity to calculate the distance that the device traveled. To refine the tracking result, we further apply MRC and a signal smoothing filter.

We implemented the system in PC using MATLAB and in Android using Java. During the experimental evaluation part, we observed that the actual system has larger error than the original paper. We provide detailed reasoning to explain what the potential causes are and what are the potential solutions in section 7.

The rest of this paper is organized as follows. Section 2 introduces some related works in this area. Section 3 provides some possible applications of this work in the future. Section 4 summarizes the contribution of each group member in this work. Section 5 provides an overview of the project, compared to the original work. Section 6 includes all the technical details of the implementation. We evaluated the system in section 7.

2. RELATED WORK

AAMouse is the first tracking system using acoustic signals. It uses the unused inaudible sound frequencies that are available in commercial devices. Other than AAMouse, some other works also utilize acoustic signals. CAT [4] propose a high precision acoustic tracking using the combination of FMCW with Doppler shift. It modifies the traditional FMCW which sends a chirp signal and applies the modified FMCW to a distributed system. Then it combines with Doppler shift to refine the results. The evaluation shows that this system outperforms AAMouse. Wang [9] proposed a device free tracking using inaudible sound wave phase information. This work differs from previous ones in the fact

that it does not track the movement of a device, instead, it tracks the movement of the finger. Device free tracking will have more applications in the future. For example, it can help to control a smart watch, and user can wear gloves. FinerIO [5] proposed a tracking system using OFDM and its echo profile. It is also a device free tracking system.

In RF-based schemes, ArrayTrack [11] uses WiFi to realize fine-grained tracking system with a median error of 23 cm using 16 antennas. RF-IDraw [8] adopts 8 RFID antennas with different spacing. This system has the median error of 3.7 cm. WiDraw [7] utilize the angle of arrive(AoA) based on CSI to enable hand-free drawing. Its median error is 5cm when using 25 WiFi transmitters. mTrack [10] has a high accuracy using 60 GHz RF signals. Tagoram [12] is also a RFID based tracking system that utilizes commercial off-the-shelf devices. Its median error is 12 cm. Compared with RF based tracking systems, audio tracking can achieve a fine-grained system with high accuracy in mm level.

IMU sensors can also be used for localization tracking. For example, Microsoft X-box Kinect uses a depth sensor and Nintendo Wii uses infrared cameras to track movement. Both system work for line-of-sight environments only and they suffer from error accumulation. Li [3] uses acceleration data from accelerator for localization tracking. However, it occurs significant error from measurement. IMU sensor can also be used to improve the tracking result.

3. APPLICATIONS

With the rise of the IoT we see a rise in smart devices such as smart TVs, smart fridges. Merely using a mouse with buttons it is not possible to meet the expected user experience offered by devices.

A traditional mouse requires a smooth surface to operate, which is feasible for a typical PC. However, when it comes to the smart TV or smart fridge, it is not suitable to have a flat surface near them and it is not suitable for user to maintain a fixed position. For example, an user may want to control a TV on couch, on desk, or on the dining table. Having a mouse in each of the location is not feasible in the real world. AAMouse can be applied in this scenario by enabling a mouse in the air. Users would be able to control appliances in non-standard places.

4. ROLES AND COLLABORATION

We discussed the project implementation details together, including what we should implement, what the time line would be, what the issues are, and what we should do to solve them. We discussed the technical concepts we needed, such as MRC, Kalman filter, STFT, Doppler Shift, and the particle filter, we discussed the implementation procedure, potential issues, and the possible solutions.

Joshua Lilly: Signal processing portion in MATLAB, Maximal Ratio Combining, Outlier removal, Kalman Filtering, Distance and point Calculation.

Mingrui Han: Implemented the system in an Android tablet. Review and discussed MATLAB code. Implemented MRC, distance calculation, following the MATLAB code. Research and implement FFT, signal filter, and other third party tools which java does not have.

5. PROJECT OVERVIEW

We implement a modified version of AAMouse on two platforms, PC (MATLAB) and an Android tablet. Our project is different than the original work in some ways, but it has the same concept as AAMouse. The key techniques used in AAMouse are, Doppler Shift, distance calculations, MRC, Kalman Filter and calibration phase. As we implemented the work, we found that the paper omits some details for implementation. Our system uses Doppler shift, distance calculation, MRC, and a signal smoothing filter. Details will be given in the later sections.

5.1 Original Work

The key idea of the original work is to use Doppler shift as an estimate of the device velocity. The Doppler shift is a well known effect where the frequency of a signal changes as a sender or a receiver moves. In our case, the sender is fixed, so we only consider the movement of the receiver. Because of its speed propagation and narrower bandwidth, acoustic signal can achieve higher accuracy than an RF signal. The detailed equation is shown below.

$$v = \frac{F^s}{F} \times C \tag{1}$$

F denotes the original frequency of the signal. F^s is the frequency shift of the signal. C is the signal speed, which is the speed of sound in this case. After the calculation of the velocity v, AAMouse multiplies it with time to obtain the distance of the device movement.

$$D = D_{previous} + v \times t \tag{2}$$

In 1D tracking, D is the result. In 2D tracking, D is the distance for each speaker. We have 2 distances of the device to each speaker and we have the distance between the two speakers. Thus we are able to track the device movement using basic trigonometry.

To improve the accuracy, AAMouse uses MRC. Measuring from a single frequency may not be reliable. Thus, AAMouse sends 10 frequencies, with a guard band of 200Hz. Then it applies MRC to average the received signal which is weighting each frequency by the inverse of the noise variance. After performing MRC, AAMouse uses a Kalman filter to smooth the estimation further using co-variance and measurement noise co-variance both to be 0.00001.

To find the distance between the speakers, AAMouse introduces a Doppler shift based calibration process. The TV emits the inaudible sound and the user scans the TV with their device. The user starts from the left end of the TV and moves to the right end of the TV. It finds the points where the value of Doppler shift changes from positive to negative. Those points indicate the time that uses spent to move the device between the speakers.

AAMouse finds the initial device location by applying a particle filter. It generates many particles uniformly distributed in the area. Each particle indicates a possible initial location of the device. In the next Doppler shift interval, the system checks the device movement of each particle. If the movement is not feasible, the particle will be filtered out.

5.2 Project Summary

We have implemented a modified version of AAMouse. Our system includes the key concept of the original work, which is the Doppler shift. Doppler shift in our system is basically the same as it in the AAMouse, except in the Android part, where there is no STFT library available. Originally AAMouse uses STFT to find the peak frequencies of the received signals. Our PC implementation does the same procedure as AAMouse, but our Android tablet uses FFT instead.

To refine the tracking result, original AAMouse uses maximum ratio combining. We also implemented such technique in both our system. We sent the signal in 10 different frequencies and average the received signal weighted by the inverse of the signal noise.

Next, AAMouse applied Kalman Filtering after the MRC to further smooth the signal. Our final evaluation does not employ the use of the Kalman Filtering, due to the fact that the authors did not specify the state transition function used in the Kalman Filter, which is needed by MATLAB. We did make an attempt using van der Pol state estimation, where the system is modeled as two non-linear ordinary differential equations [14].

$$\dot{x_1} = x_2$$

$$\dot{x_2} = (1 - x_1^2)x_2 - x_1$$

but this seemed to produce slightly worse results in the empirical evaluation. Which would suggest this is not the evaluation function the authors chose, or there is a bug in the implementation. On the Android side, there are not a lot of options to choose from. For example, Kalman Filtering does not exist on the Android platform. We used a signal filter from [2], called OneEuroFilter, to smooth the signal.

The distance calculation portion of the work are the same as AAMouse. We calculated the distance to each speaker and then calculated the intersection of the two circles from two speakers with the radius as the distance calculated before.

AAMouse implements q particle filter to find the initial location of the device. Due to the complexity of the particle filter concept and the missing details of the implementation in the original paper. We were not able to replicate the particle filter from AAMouse. Therefore, our system does not have the particle filter implemented, which causes large error.

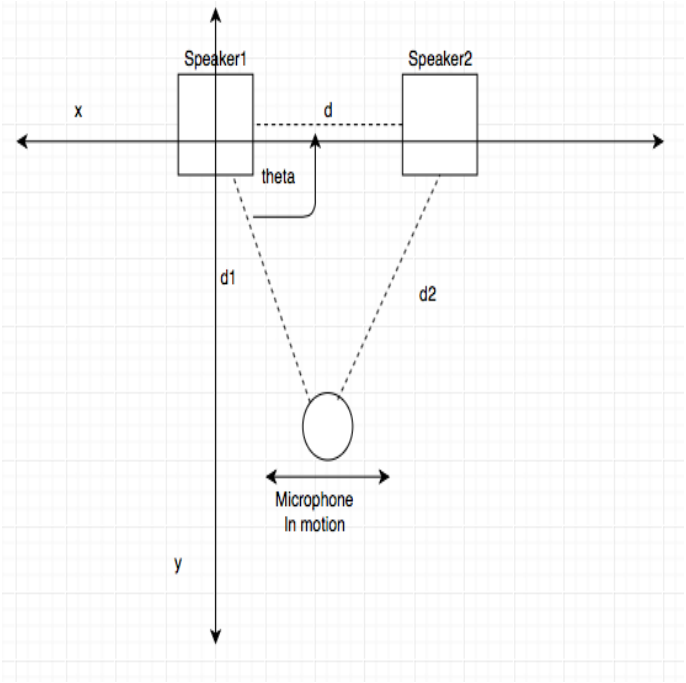
6. DESIGN AND IMPLEMENTATION

We implemented the system on two platforms, PC using MATLAB and Android using Java. In this section, we will discuss the implementstion details of our system. It includes the hardware we used, the experimental setup, the implementation approach we applied, the problems we faced and the solutions we used.

6.1 Setup PC

The PC implementation is implemented in MATLAB using a Macbook with 4 Gigabytes of memory and an Intel I5 processor with a clock speed 2.8 GHz. A simple microphone (Snowball Black Ice) is used for collection of the audio samples and two Logitech Z200 speakers are used to play the sounds. The distance between the speakers is measured to be 1 meter and the initial starting location for the microphone is approximately 1 meter from both speakers.

Figure 1: Experiment setup for PC



6.2 Technical Details Sound Generation

In order to generate sounds, MATLAB is used to generate 10 Sinusoidal waves, 5 for the left speaker and 5 for the right, each spaced 200 Hz apart. The process for generating the sounds can be seen in Algorithm 1.

6.3 Technical Details Receiver Design

The receiver which processes the audio signals uses a continuous loop to run for some variable amount of time. Inside of the loop we take 44100 samples from the microphone and perform a Short Time Fourier Transform with 44100 points. This also applies a Hamming window with 1764 points and an overlap of 50 percent. At this point the signal consists of all of the 10 frequencies transmitted from the speakers. We next perform MRC by imposing the idea of a channel. We do this by using a window of 50 Hz around the center frequencies of each of the 10 frequencies we generated. Then for each channel we estimate the noise variance by taking the sum of all of the signals outside of where we expect the signal to fall and then averaging this value by the number

Data: amp=1;
Data: fs=80000;
Data: duration=50;
Data: values=0:1/fs:duration;
for *frequency from 18000 to 18800* **do**
  $wave\_left =$
  $wave\_left + amp * sin(2 * pi * frequency * values)$;
  $frequency+ = 200$;
end
for *frequency from 19000 to 19800* **do**
  $wave\_right =$
  $wave\_right + amp * sin(2 * pi * frequency * values)$;
  $frequency+ = 200$;
end
$combined = [wave\_left(:), wave\_right(:)]$;
$write(file, combined)$;
Algorithm 1: How to generate Sinusoidal inaudible sounds

of points used. We then weight each channel by the inverse of this value, which is a rough approximation of the inverse of the noise variance. Next the frequencies are split into two sets. The weighted signals are then averaged with the other signals in their set and the the frequency exhibiting the strongest power is extracted from each set. We use these values to calculate the Doppler shift by finding the original frequency value it is closest to and taking the difference. If Doppler shift that is generated is greater than 10 Hz we throw it out. In the event we throw the point out, we extract the maximum power signal in each channel and calculate the difference between each value and it's closest original frequency value. Then the Doppler shift calculation that is closest to the previous loop's Doppler shift is selected as the Doppler shift. We specifically note that this technique of throwing out points in the event the Doppler shift is greater than 10 Hz may cause a biasing towards higher values for future calculations in the event no channel yields a Doppler shift estimate that is less than 10 Hz. Which is a potential source of error, however, the probability that this event will occur is somewhat small given the fact there are 10 frequency values. The Doppler shift obtained is then used to estimate a distance between each of the speakers. Then given distance of the speakers we estimate the angle between the speaker and the microphone and use that angle to then calculate an x y point. We also note that this point is the intersection of two concentric circles, each with their center at each of the two speakers. This means there are potentially two intersections. However, for simplicity and because the second intersection point tends to cause a larger distance jumps from the microphone we do not calculate the second intersection point in the event one exists.

6.4 Setup-Android

We used one Samsung SM-T320 tablet and two Logitech S-120 speakers for experiment hardware. The experiment setup is the same as the one mentioned above for our PC implementation. We keep the distance between the speaker to the device to be one meter and the distance between two speakers also to be one meter.

6.5 Technical Details-Android

The sampling rate we used is the same as the original paper, 44.1 KHz. It is also the default sampling rate supported by the device. By default, we uses 44,100 as the buffer size to store the audio samples. Such size is chosen in order to provide the fine-grained frequency resolution. Because in Android system, there is no signal processing libraries to use, we cannot call STFT function in one line as we did in PC side. Thus the first issue is how to calculate frequency shift in Doppler shift. We used a third party FFT tool from [1]. This FFT function will return the same number of points as the buffer size of the stored signal. The even index of the FFT point are the imaginary part and the odd index of the FFT point are the real part. We then found the frequencies with the greatest FFT value and subtracted it with the sending frequency to obtain the frequency shift. Then we are able to calculate the velocity.

To improve the accuracy of the system, we also implement MRC in Android side, the technical detail is the same as above for the PC section. After applying MRC, the original work uses Kalman filter to smooth the data. However, there is no such filter available in Android system. Instead, we applied a third party filter from [2] called OneEuroFilter to smooth the data.

After the post processing, we were able to get the frequency shift in each axis, averaging five sound frequencies. Then we calculate the distance using the same equations mentioned above.

6.6 Lessons learned

The knowledge we obtained from the project, specifically with the PC implementation in mind is we learned numerous pieces of signal processing theory, including why we would want to use a STFT as opposed to an FFT. We also learned tons about spacial and frequency diversity when we attempted to implement the maximal ratio combining portion of the project since most of the examples for diversity combining techniques center around spacial diversity as opposed to frequency diversity. Despite learning this, we still are yet to find a good example on Maximal Ratio Combining for frequency diversity. Hence it is possible our implementation is theoretically flawed. We also learned how to implement the Kalman filter in MATLAB, which is not necessarily a simple task despite there being numerous resources on how to do it. Modeling the problem as a state transition function turns out to require even more knowledge of signal processing, beyond what even to this point we have obtained. Despite this we did make an attempt to figure out how to come up with a model for our problem, unfortunately more research is necessary to build the Kalman filtering into the code. One piece of the of the project which we did not attempt to implement was the particle filtering for estimation of the starting location of the microphone. This is our largest source of error by far and is the primary reason for regular bad results. We further hypothesize that we could obtain significantly better results for the project if we would have implemented this portion. In Android side, The initial buffer size is chosen as 1759, which is the minimum buffer size by default calculation using Android AudioRecorder class. Using buffer size of 1759 is no doubt too small for FFT. Such value will lead to a very low frequency resolution, which will cause the distance calculation to be very inaccurate. Same theory is also mentioned in the original work. The correct buffer size should be 44100. However,

this value did not work at the first place and it will lead the app to stop working immediately. One suspicious cause is the intensive computing cost inside the while loop for the real time signal processing. Then we optimized the code to compute the maximum frequencies in linear time. This procedure allow the buffer size to be set as 44100. The app will run for approximate 15 seconds before it crushes. Even if the app will work for 15 seconds, this huge buffer size value will take one second to record. A long FFT does not allow us to track the device in real time. In one second, the device may move to other location. As a result, even though we were able to obtain the fine grained frequency shift using our Android device, the tracking result may not be accurate due to the long FFT.

7. EVALUATION

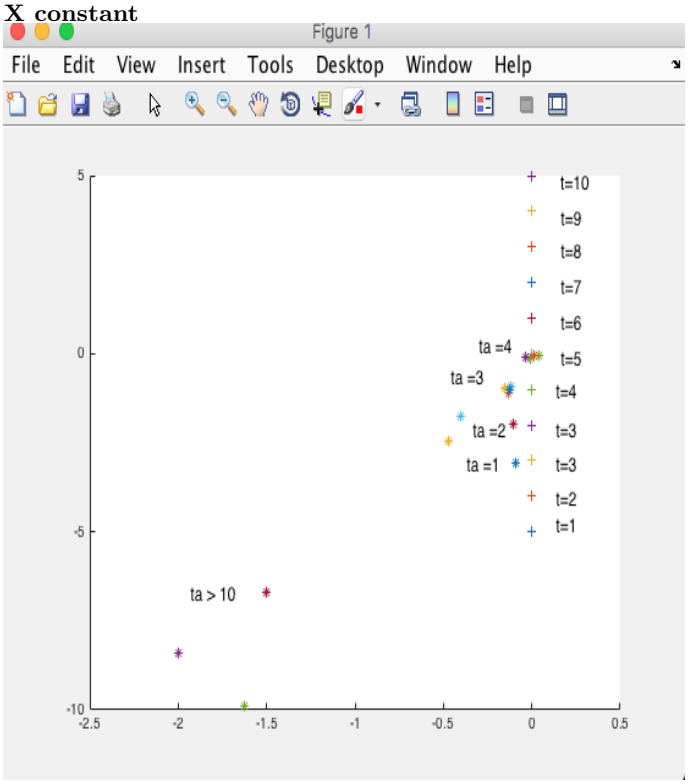
The evaluation was carried out on both Android and PC platforms. To evaluate the PC platform we evaluate the general ability of the platform to trace given shapes since distance will always be inaccurate due to the lack of knowing where in global space the fact we did not implement the particle filter. If one were to use distance as a metric the implementation obtains a massive error calculation. Hence the need for a more qualitative evaluation. Additionally, we note that the platform only gives a window of around 8 to 10 seconds before error in all variables, x, y theta and velocity propagates to the point where the system becomes completely incorrect and neither tracks distance nor general shape.

7.1 Performance Analysis - Motion Tracking

In figure 1 we can see the plot of tracing a simple straight line moving the microphone toward the speakers. We see the ground truth indicated by the + mark and the actual value of the microphone as *. Additionally, the time at which the point was obtained is included in the adjacent text, where t indicates the time at which the ground truth point would be obtained and ta represents the time at which the actual points are obtained. Additionally, due to a constant error of approximately 30 cm the graph takes this into account and normalizes the result by subtracting the error. From this image we see that a majority of the points near the beginning of the test are close to the ground truth values, however, as we see time approach the end of the test where $ta > 10$ we see very poor behavior from the system. This is most likely due to the error propagation.

In figure 3 we see an example of the system tracing a straight line in the X direction while Y. Figure 3 is the ground truth of what the plot should be and figure 4 is the actual obtained values. Note these are in separate plots due to the values on the x-axis being much larger than the ground truth. This is because the microphone was in motion before the the sampler started up and the first Doppler shift estimation distance had a large error. We notice that it is almost constant that if the microphone is in motion before the start of the trial, an error of 30 cm in either the x or y direction is obtained, so it is possible to correct this. If the microphone is not in motion we still obtain an error around 30 cm due to the fact that the estimation of the Doppler shift, where it is small when the microphone is not in motion, it is still non-zero. This in addition to the fact we don't know the start location is why we obtain erroneous results. One possible solution for this is to not take Doppler shift values that are close enough to zero and filter them out. This could potentially be done with Baysian Filters, including Kalman Filters.

Figure 2: Tracing line in Y direction, while holding X constant



7.2 Inaudible Tone Frequency Choices

Originally, we chose inaudible frequencies between 20kHz and 21.8 kHz. However, after attempting to use these frequencies it was observed that the power of the frequencies above 20kHz were very week. Due to this we switched to use the frequencies from 18kHz to 19.8kHz. The results of the PSD graphs can be seen in figures 4 and 5.

7.3 Potential Improvements

There are numerous potential improvements that could be made to the system in nearly every aspect. To start with a better implementation of Maximal Ratio Combining could be used, specifically one that does not discount the actual frequencies around the center of each channel for calculation of the noise variance. One other aspect for improvement is in the potential to further smooth the Doppler shift curve with Kalman Filtering; this however, would require much more research and evaluation in which transition function would best fit to the problem. Another improvement is answering the question of whether or not we actually need to calculate a global points in global space. One possible implementation could define both global and local coordinate spaces, where the local coordinate axis is with respect to the microphone and the global axis is with respect to the speakers. Then a method for deriving points in the global coordinate axis involves multiplying the result in the local

Figure 3: Ground truth for tracing a line in the X direction

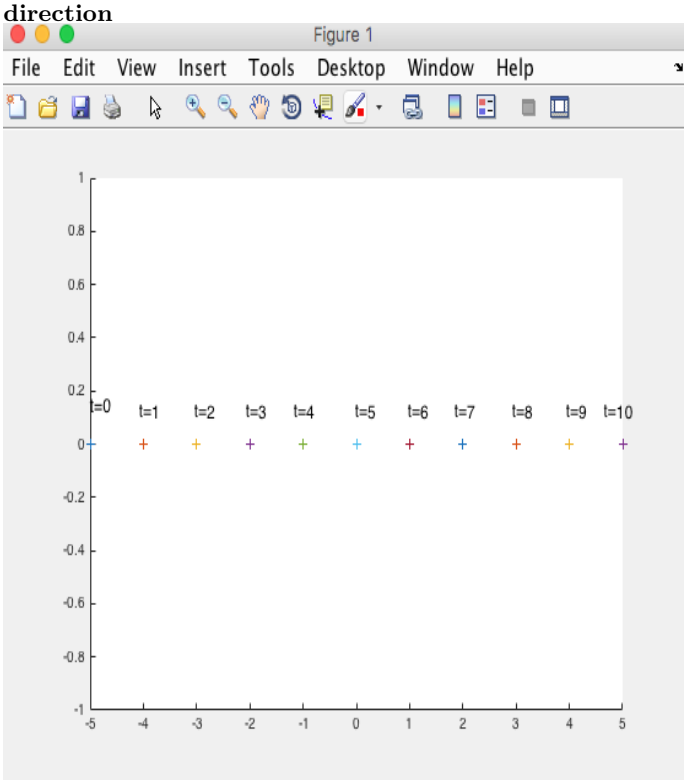
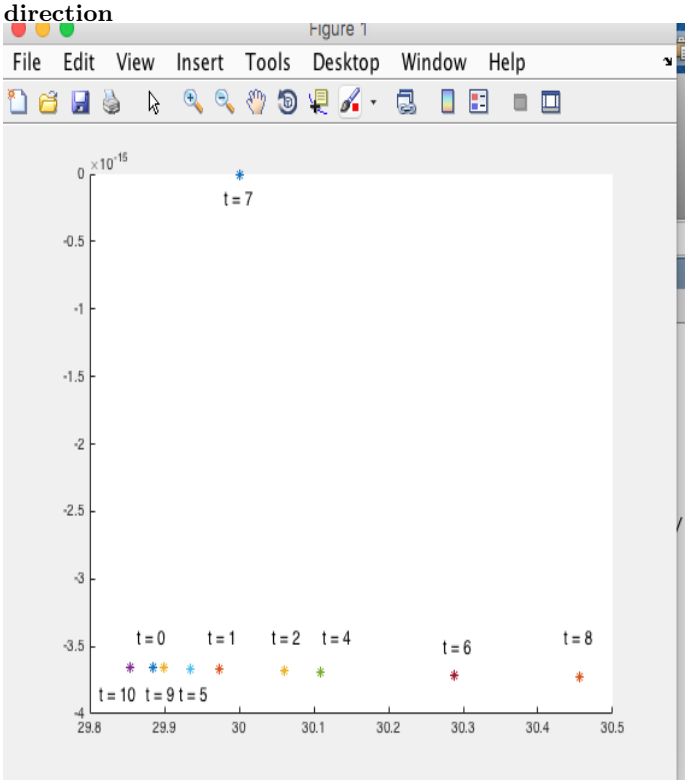


Figure 4: Ground truth for tracing a line in the X direction



coordinate axis by the transpose of the Jaobian. This idea comes from kinematics in robotic motion planning. Finally, one large issue we ran into when performing the evaluation is that the time to process the data in real time reached a point where the data that was being reported was old and mismatched in the timing. This is because the time it takes to complete one of the loops in the MATLAB program is approximately 1 second. The authors of the paper explicitly mention that 1 second is too long of a window in time for results to be accurate. One possible solution to this is to gather the data in a matrix in MATLAB for each instance of time and then calculate the x y pairs in a non-real time fashion. In Android side, we could use STFT packet instead of FFT to increase the accuracy because a long FFT cannot track the device in real time. However, currently there is no STFT library available for Android platform. We could implement our own STFT. The original work applied STFT to use 1764 audio samples to still achieve 1 Hz frequency resolution.

8. REFERENCES

[1] P. N. Amsen. Noise. https://github.com/paramsen/noise, 2017.

[2] R. de Courville. Signal filter (beta). https://github.com/SableRaf/signalfilter, 2015.

[3] F. Li, C. Zhao, G. Ding, J. Gong, C. Liu, and F. Zhao. A reliable and accurate indoor localization method using phone inertial sensors. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 421–430, New York, NY, USA, 2012. ACM.

[4] W. Mao, J. He, H. Zheng, Z. Zhang, and L. Qiu. High-precision acoustic motion tracking: Demo. In *Proceedings of the 22Nd Annual International Conference on Mobile Computing and Networking*, MobiCom '16, pages 491–492, New York, NY, USA, 2016. ACM.

[5] R. Nandakumar, V. Iyer, D. Tan, and S. Gollakota. Fingerio: Using active sonar for fine-grained finger tracking. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 1515–1525, New York, NY, USA, 2016. ACM.

[6] A. Rai, K. K. Chintalapudi, V. N. Padmanabhan, and R. Sen. Zee: Zero-effort crowdsourcing for indoor localization. In *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, Mobicom '12, pages 293–304, New York, NY, USA, 2012. ACM.

[7] L. Sun, S. Sen, D. Koutsonikolas, and K.-H. Kim. Widraw: Enabling hands-free drawing in the air on commodity wifi devices. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, MobiCom '15, pages 77–89, New York, NY, USA, 2015. ACM.

[8] D. Vasisht, J. Wang, and D. Katabi. Rf-idraw: Virtual touch screen in the air using rf signals. In *Proceedings of the 6th Annual Workshop on Wireless of the Students, by the Students, for the Students*, S3 '14, pages 1–4, New York, NY, USA, 2014. ACM.

[9] W. Wang, A. X. Liu, and K. Sun. Device-free gesture tracking using acoustic signals. In *Proceedings of the 22Nd Annual International Conference on Mobile Computing and Networking*, MobiCom '16, pages

Figure 5: PSD when using frequencies above 20kHz

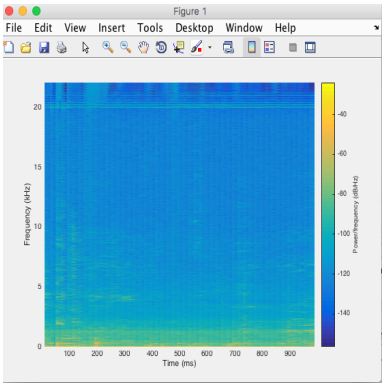
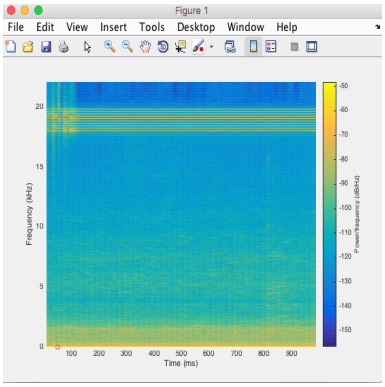


Figure 6: PSD when using frequencies above 18kHz to 19.8kHz



82–94, New York, NY, USA, 2016. ACM.

[10] T. Wei and X. Zhang. mtrack: High-precision passive tracking using millimeter wave radios. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, MobiCom '15, pages 117–129, New York, NY, USA, 2015. ACM.

[11] J. Xiong and K. Jamieson. Arraytrack: A fine-grained indoor location system. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, nsdi'13, pages 71–84, Berkeley, CA, USA, 2013. USENIX Association.

[12] L. Yang, Y. Chen, X.-Y. Li, C. Xiao, M. Li, and Y. Liu. Tagoram: Real-time tracking of mobile rfid tags to high precision using cots devices. In *Proceedings of the 20th Annual International Conference on Mobile Computing and Networking*, MobiCom '14, pages 237–248, New York, NY, USA, 2014. ACM.

[13] S. Yun, Y.-C. Chen, W. Mao, and L. Qiu. Turning a mobile device into a mouse in the air. In *MobiSys*, 2015.