Indoor Localization system using Time of Flight in UWB spectrum

Anirban Ghosh, Albert Davies, Tejus Siddagangaiah
Electrical and Computer Engineering Department, Carnegie Mellon University, Pittsburgh, PA 15213 USA

Positioning systems based on GPS have become ubiquitous over the past few decades. Due to the attenuation of GPS signals, they cannot be used in an indoor setting for localization. In recent years, multiple attempts have been made to design a localization system for indoor applications. Indoor localization has the potential for trans-formative impact in a wide variety of applications like augmented reality, indoor navigation, asset tracking and advertising. Due to the inherent nature of the applications, high accuracy as well as energy efficiency is required. Based on the target application, various methods and system architectures have been proposed using broadcast-based technologies(RFID [1], BLE [2], WiFi [3], Ultrasound [4]) and motion based algorithms(Inertial based sensing) [5].

In our work, we develop a system for Indoor Localization to track mobile TAG nodes in an indoor setting with multiple anchor nodes whose coordinates are known. We use Decawave's DWM1000 modules to perform accurate ranging and Arduino UNO WiFi modules to control the modules and perform basic computation. We achieve a best case and average accuracy of 10cm and 20 cm respectively across each axes. [1]

I. Introduction

[1] surveys various RFID based localization strategies. RFID based localization schemes suffer from need for several active RFID anchors to be able to localize accurately. WiFi based localization schemes require intensive calibration in the area of localization. Several methods have been proposed to crowd-source the RSSI values from different WiFi access points [3]. Another approach to this problem has been to use two different mediums and measure the time difference of arrival of signals in both the mediums. This is the approach used by [2] wherein the different speed of travel of bluetooth and ultrasonic waves are used to estimate distance. Inertial based localization strategies [5] fail to achieve sub metre accuracy due to drift from initial calibration that compounds with time. A recent development in the indoor localization domain has been the success of UWB based methods [6]. The promise of UWB based localization was witnessed when DWM1000, a UWB localization chip won the Microsoft indoor localization competition in the year 2015. Some of the current commercial positioning systems include the Ubisense technology, Time Domain technology, Decawave technology, Zebra technology, Nanotron technology and Apple's iBeacon. The advantages and disadvantages of these systems are investigated in [7]

II. Architecture

This section describes the overall system architecture as well as some of the design decisions we took during the course of this project. We begin by describing the different localization algorithms. Next, we describe the software architecture and finally describe the hardware design choices.

A. Localization Algorithm

There are two predominant methods used in the field of localization (both indoor and outdoor). The first is time-of-flight measurement. This computes the point-to-point distance between two nodes based on the round trip time of a time-stamped packet travelling in a known medium. Alternatively, time-difference-of-arrival uses the difference in time of arrival between multiple packets sent from the Tag to time-synchronized anchor nodes. These schemes and their corresponding equations are explained in [8]

1) Time of Flight

As shown in Figure 1, the time-of-flight (TOF) method computes the distance between the two nodes by measuring the round-trip time of a time stamped packet. As given by equations 1 and 2 node 1 sends the packet (t1 at $t1_{tx}$) and waits for node 2 to respond (t2 at $t2_{rx}$). Upon receiving the packet, node 2 responds with another packet, which contains the time difference between when node 2 receives the first packet and sends the response. When node 1 receives node 2's response, it calculates the difference between when it sent the first packet and when it receives the response. Node 2's response latency, which is embedded in the response packet is then subtracted from this time to obtain the true round trip time. The packet travels twice the distance between the nodes, so the distance between the two nodes is equal to half the round trip time multiplied by the speed of the packet in the given medium, as given by 2. In case of the DWM1000, the speed of the packet is the speed of light in air and in case of ultrasonic methods, this is the speed of sound in air and so on. In 3D space, the distance computed between an anchor and a tag defines a sphere where the tag must lie. The tag must then periodically find its distance from four different anchor nodes placed around the room. The 3D location of the tag will then be the intersection of the four spherical surfaces generated in this process. The advantage of the TOF method is that it is much simpler to implement well and does not require time synchronization between the nodes. However, since the round trip time is measured, this method requires the delay added due to transducers (RF circuits in our case) and the response time of node 2 to be modelled accurately. In the case of the DWM1000 modules, these parameters are factory calibrated and don't cause any accuracy issues.

$$T = (t2_{rx} - t1_{tx}) - (t2_{tx} - t1_{rx}) \quad (1)$$

[1] Git Repository: https://github.com/tejus26/Indoor-Localization.git

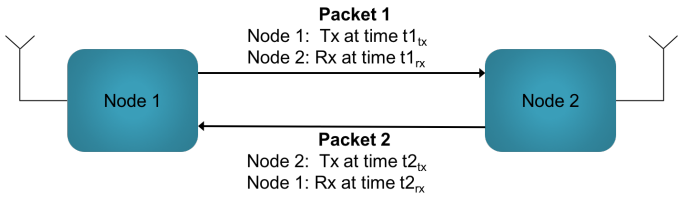

Fig. 1. Illustration of Time of Flight algorithm

$$Distance = \frac{Speed\ of\ Light \times T}{2} \tag{2}$$

2) Time Difference of Arrival

The time-difference-of-arrival (TDOA) method is comparitively a more scalable algorithm for localization. In this method, the target tag periodically sends beacons to anchors in its vicinity. When the anchors have time synchronized clocks, the time difference of arrival of the packet at between each pair of anchor nodes defines a hyperbolic surface on which the target tag must lie. The tag's 3D location can be computed by finding the intersection of all these hyperbolic surfaces. However, it does require very precise time synchronization between the clocks of every anchor in the network [9]. Specifically for the DWM1000, the medium is an electromagnetic wave, so any error in time synchronization gets multiplied by the speed of light. We chose to start our implementation with the TOF method because it is much simpler to implement and the DWM1000's factory calibration takes care of a lot of the error-causing factors that the TOF method is susceptible to. While we did keep TDOA as a stretch goal, our primary objective was still to get the entire system up and running with the best possible accuracy achievable with TOF since time-synchronizing all these wireless nodes is a hard problem.

B. System Architecture

The goal of the project was to implement a system which achieves high 3D localization accuracy and low enough power consumption to be powered by a Li-Ion battery. We used Time-of-Flight to measure the distance of the tag node with respect to each one of the anchor nodes and then used a WiFi network to transmit the distance data from the tag to the backend software running on a PC. The PC runs a Python script to connect to the mobile node and collects the distance data with Anchor ID. This data is then processed by a series of signal processing steps (described in section III) to localize the target and give the user an intuitive and interactive graphical output about the tag's location. Decawave claims a 1D distance measurement accuracy of ≈10cm using these sensors. After calibration, we acheived a best case accuracy of ≈10cm on all 3 dimensions, and an average accuracy of ≈20cm on all 3 dimensions, which is at par with the device manufacturer's claims.

The 3D location of any node can be found if the distance from four anchor nodes of fixed location in 3D space is known. In the TOF method, for example, the distance "d" from one anchor node defines a sphere with radius "d" around that node. The surface of this sphere is the locus of points where the tag must lie. The spheres formed by the distances from two anchors intersect at a circle, the perimeter of which is the locus of points where the tag must lie. A third anchor generates another sphere which meets this circle at two possible points. A fourth anchor is needed to find which of these points the tag must lie in. Ofcourse, in real world systems, there is a finite measurement error, and none of these spheres are clearly defined. Therefore, a closed form solution of the system of equations formed by these four spheres may not exist all the time. Therefore, a least-squares equation solver is used to minimize the set of equations and find the most probable location of the tag. The system setup of our system is shown in Figure 2 with three anchor nodes.

To obtain the distance measurements, we use time of flight of

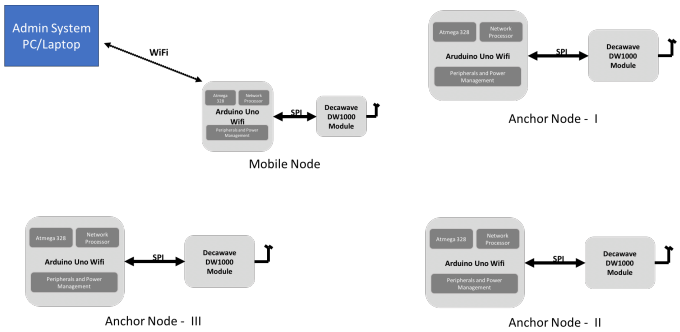

Fig. 2. Complete system architecture

radio waves over the UWB spectrum. The DWM1000 sends very short chirps of radio signals in the UWB spectrum. The benefits of this include a very fine time resolution, energy efficiency, and most importantly, robustness to interference in harsh environments [6]. UWB is also particularly resilient to multi-path fading effects. In this project, we use the Decawave ScenSor DWM1000 Modules for UWB TOF measurements. Using these modules the distance of the mobile node from each of the anchor nodes is determined. This information is accumulated at a gateway for the multilateration computation. For this we interface DWM1000 with Arduino Uno Wifi using the SPI interface supported by DWM1000.

C. Software Design

1) Backend Software

Python was chosen as the language for backend computation for the following reasons:

- Scipy and Numpy packages provide ready access to minimization functions required for localization algorithm
- Matplotlib package provides an environment to scatter plot the tag location in 3D
- Python being a high level language is best suited for prototyping. The tradeoff in speed was tolerable in this application as the point did not have to be plotted in realtime.
- Availability of Telnet library to connect to the Tag listening socket and read the distance values.

2) Decawave Library

The Arduino DWM1000 library hosted by Thomas Trojer on github was used [10]. This library had the 2 way ranging code implemented with instructions on how to interface the DWM1000 with the Arduino. The library was still active as of writing this report.

3) WiFi Library

The standard library which ships with Aruduino Uno Wifi was used. It came with a reference program to relay the serial readings over Wifi. This library also makes the Arduino Uno a WiFi access point. The advantage of this compared to connecting to a public access point was the static IP allocation of the Tag node. Since the Tag was hosting the access point it always had the first IP in the subnet eliminating the need for having access to the public DHCP server.

D. Hardware Design

1) Micro-controller Platform Decision

When we initially started the project we had a set of specifications in mind for the microcontroller and wireless communication device. We required a low power microcontroller board with low power WiFi connectivity. The MCU should have at least one SPI and one UART port for interfacing with the Decawave and debugging in addition to whatever interface it requires for WiFi. One solution was to use a low power MCU platform like Arduino or TI's MSP430/432 Launchpad and an external WiFi board like the Arduino WiFi sheild or the CC3100 BoosterPack. However, we already had a Decawave board and a set of level shifter ICs and we did not want to add an additional board (and therefore another point of failure). We then refined our requirement and looked for single board solutions that incorporated a WiFi radio along with a low power MCU. While the Raspberry Pi is an obvious solution to this, we decided against it since the Raspberry Pi comes with a 1GHz ARM processor and high speed WiFi radio, both of which were an overkill for this application and had a power consumption that was definitely out of budget. Finally, we narrowed down to the CC3200 Launchpad and the Arduino UNO WiFi. We initially selected the CC3200 since it incorporates a single chip WiFi+MCU and we thought its power consumption would be lower, and the powerful 80MHz ARM Cortex M4 on the CC3200 might be needed for our algorithms. However, upon carefully analyzing our options, we concluded in favor of the Arduino for the following reasons:

Library Support: The Decawave SPI driver and 1D ranging libraries were readily available for Arduino platforms and not directly available for the CC3200. While the CC3200 does support the "Arduino-like" Energia platform, significant porting effort may have been needed just to get the CC3200 to talk to the Decawave. The Arduino was preferable since the Arduino SPI libraries take care of the hardware abstractions and board-to-board dependencies.

On-chip Flash: The CC3200 also suffers from another drawback, which is the fact that it does not come with any on-chip flash. While this is taken care of in the launchpad with the help of an external flash on the board, an MCU with on-chip flash benefits from easier programming and deployment. We saw this as a major concern for the later stages of the the project where we would need to test and calibrate multiple of these boards and program them to make tweeks during testing time. It is more convenient to have a device with built-in flash rather than requiring the additional step of using TI's UNIFLASH to program the external flash on the CC3200 board.

Power consumption: The CC3200 has an active power consumption of about 15.3mA at 3.3V without the WiFi radio on, which translates to an active power consumption of 50.49mW. When the WiFi radio is active at peak power, the total current consumption is 278mA at 3.3V, which is 917.4mW of peak active power excluding the power consumption of the external flash, which draws an additional standby current during runtime. The ATMega328P consumes about 5.2mA at 5V when active, which translates to about 26mW active power consumption. We needed to use 2 Arduinos in order to not stress the RAM on a single Arduino and obtain stable operation. Therefore, the combined power consumption of both Arduinos is 52mW. The Arduino responsible for WiFi transmission was an Arduino UNO WiFi with an ESP8266 WiFi Radio. The ESP8266 WiFi SoC on the Arduino UNO WiFi board has a peak power consumption of 170mA at 3.3V, and therefore has a peak power consumption of 561mW. Therefore, our combined active power consumption for the 2 Atmel microcontrollers along with the ESP8266 is 613mW. This makes the combined Arduino solution about 33% more power efficient than the CC3200 solution, even without considering the CC3200's external flash's standby current.

The main reason for choosing the CC3200 initially was

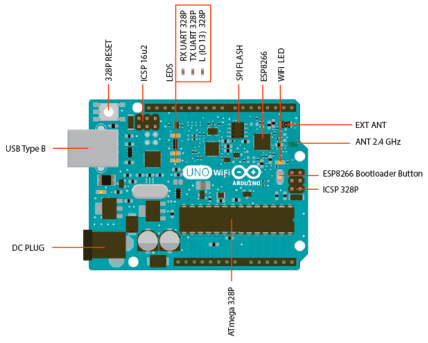

Fig. 3. Board layout of Arduino Uno Wifi SoC

that we were able to get a powerful MCU and a WiFi chip in a low cost, low power board. However, we realized that it was more suitable to run the math-heavy least-squares linear equation solver needed for accurate localization on the backend software on a PC. This gives us more flexibility to use the vast collection of mathematical libraries and filters available in NumPy while also reducing our MCU's CPU utilization. Since we no longer needed such a powerful MCU, the Arduino's processing power and memory was sufficient

for our purposes. Based on the above reasons, we concluded that the Arduino UNO WiFi was the right device for the job. The block diagram of Arduino UNO WiFi is shown in Figure 3

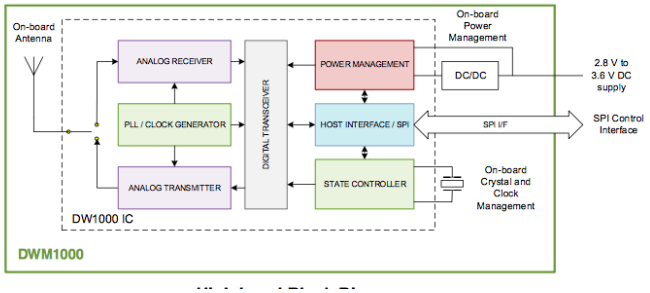

Fig. 4. Block Diagram of DW1000 UWB transceiver

2) DWM1000 Module

Due to the advantages mentioned in the previous sections, using a UWB based sensor helps us achieve better accuracy and also avoid multi-path fading. DWM1000 modules from Decawave claim an accuracy of ≈10cm between two nodes. DWM1000 modules are sensors which use DW1000 chips with RF and clock circuitry. The modules can be interfaced over SPI and requires a 3.3V external power supply. The block diagram of DWM1000 module is shown in Figure 4. Time-of-flight applications are sensitive to clock drifts which can reduce the accuracy significantly. To alleviate the effect of clock drifts, the clocks on the modules are factory trimmed to within +/- 2ppm under typical conditions. Due to the reasons stated above, DWM1000 matches with our requirement. Other advantages of using DWM1000 modules are listed below:

- Upto to 6.8Mb/s achieved in Wireless Sensor Networks
- Communication range of upto 290m
- Low power consumption
- Compact size
- RF design is not required
- Integrated antenna

Each node (both tag and anchors) have Arduino UNO WiFi board and a Decawave module interfaced over SPI. The hardware architecture of each node is shown in Figure 5

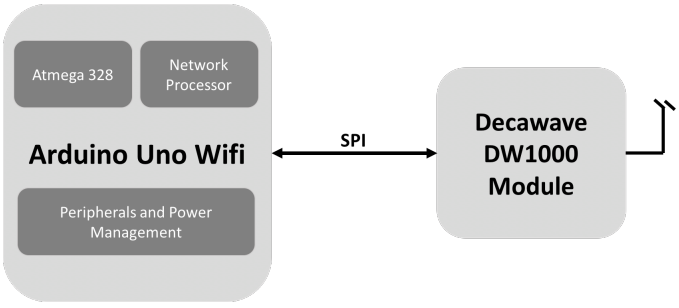

Fig. 5. System Block Diagram of Each Node

III. Implementation

As shown in figure 2, we use multiple anchor nodes with known positions to locate a mobile node. We use dynamic network discovery to connect tags with anchor nodes and perform 2-way ranging with each anchor node. The calculated distance between the tag and each anchor is sent from the tag to an admin laptop over WiFi. The distances from four anchor nodes is used to perform localization. The error of localization in our implementation is as low as **10 cm on each axis** in the best case. On an average, we are able to locate the mobile node **within 20 cm on all three axes**.

A. Network Discovery and Communication

A mobile tag node performs network discovery and initiates 2-way ranging with each of the anchor nodes. *TAG* node controls the entire network to calculate the point-to-point distances between the anchor nodes and itself. As soon as the *TAG* node is turned on, it broadcasts a **POLL** packet to all its neighbours. All the *ANCHOR* nodes that receive the broadcast packet, respond with a **POLL_ACK** message. The mobile node uses these acknowledgement messages to prepare a list of anchor nodes in its range. It is important to note that the mobile node sends one broadcast message and receives multiple acknowledgment messages.

Using the network discovery performed using **POLL** and

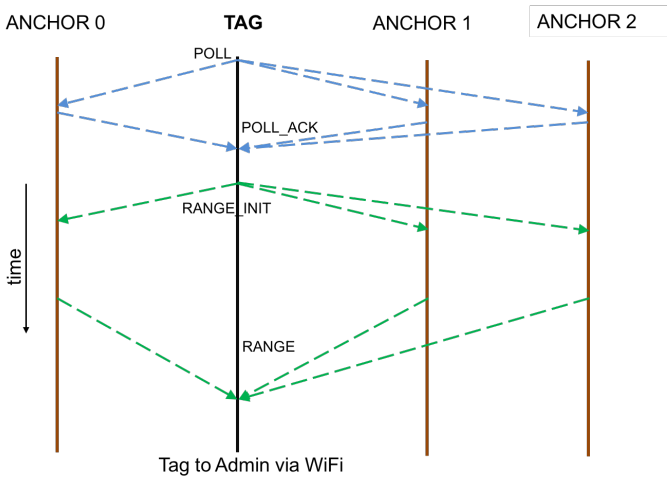

Fig. 6. Network discovery and two-way ranging protocol

POLL_ACK messages, the mobile node maintains a list of anchor nodes and performs 2-way ranging with each of those nodes. Once the anchor nodes respond with **POLL_ACK** message, it waits to receive the **RANGE_INIT** packet from the mobile node. **RANGE_INIT** and **RANGE** packets are exchanged between the nodes using timestamps to perform 2-way ranging. As shown in figure 6 the messages exchanged between the mobile node and the anchor nodes involve multiple steps and are performed one after the other.

If an anchor node goes out of range after responding with the acknowledgement message, the mobile node marks it as an inactive node and deletes the corresponding anchor node from its network list. Thus, our communication protocol can discover dead anchor nodes immediately.

B. Ranging Calibration

Although the timestamps used by DWM1000 modules to perform 2-way ranging are quite accurate, there is an error in the point-to-point distance calculated due to the communication protocol overhead. Comparison between the actual distance and the calculated distance was made. It is observed that for small ranges, the percentage error in the distance measurement is quite high and this error reduces as the distance between two nodes increase. The error is calculated for distances from few centimeters to 10 meters and the plot is shown in figure 7 & 8.

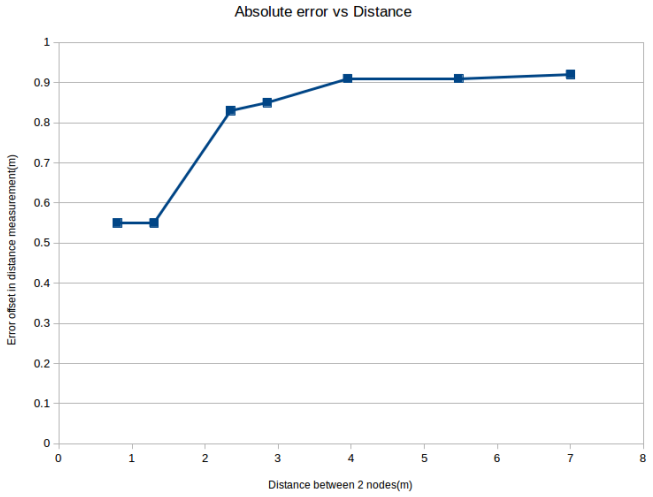

Fig. 7. Absolute error between 2 nodes while doing 2-way ranging

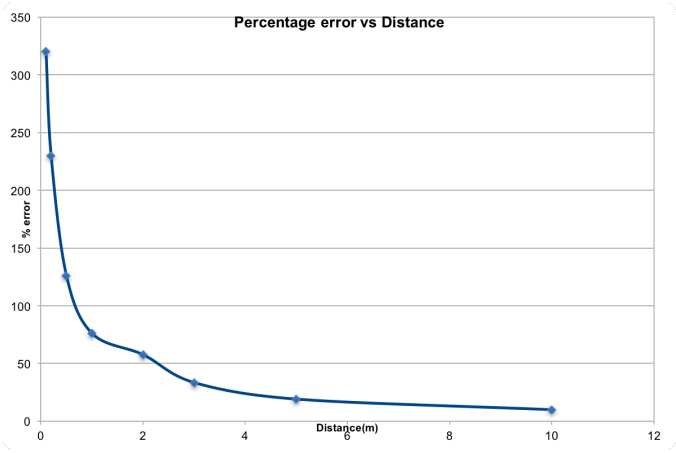

Fig. 8. Percentage error between 2 nodes while doing 2-way ranging

We use this data to calibrate the ranges sent from mobile node to the admin laptop. The point-to-point distance between the tag node and each anchor node is calibrated using a piecewise linear look-up table. Error calibration using this technique increased the localization accuracy significantly. The error calibration increased the accuracy from 50 cm on all three axes to 20 cm on an average. This also made our equation solver stable and converge sooner.

C. Localization

The equations for localization were solved in Python using the Localization library [11]. This library uses SciPy and NumPy libraries to find a least square error estimate of the three co-ordinates. The total error was defined as shown in Equations 4 and 3

$$D_i = \sqrt{(X - x_i)^2 + (Y - y_i)^2 + (Z - z_i)^2} \tag{3}$$

where, D_i is the distance of the tag from anchor i.

$$error = \sum_i (D_i - r_i)^2 \forall\ anchor\ i \tag{4}$$

where, r_i is the range measured by DWM1000 to anchor i.

This error was minimized using $scipy.optimize.minimize$ with the initial estimate $X = 0, Y = 0, Z = 0$.

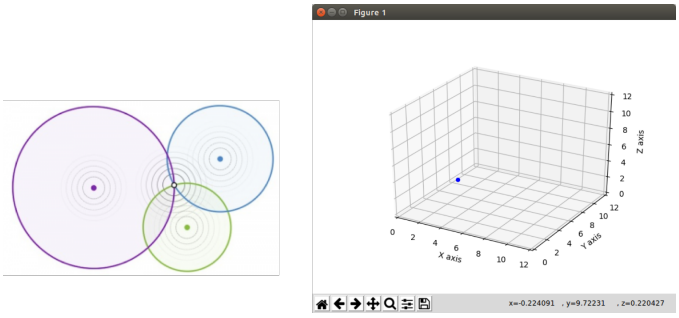

Fig. 9. 3-D Localization

Further information on the equations used and the corresponding equations for TDOA can be found in [8]

D. Placement of Anchor Nodes

Different cofigurations of placement of anchor nodes was tried out. When all four anchor nodes were placed in one plane, the error in distances from each anchor gets compounded along the perperdicular dimension. Therefore the best possible placement was achieved by placing the 4 anchors as orthogonal as possible as shown in Figure: 10. The lengths along each dimension were chosen for convenience of placing anchor nodes.

E. Median Filtering

To eliminate noise iin the samples, median filtering was performed on a sliding window. Median is particularly resilient to sparse errors. Hence the last 5 calculated co-ordinates were held in a queue. The present co-ordinate was chosen to be the median of these 5 values. This made our co-ordinates very robust.

IV. Challenges

We faced a number of challenges during the course of this project. These are listed below:

- DWM1000 did not come with a base board to mount on a breadboard. This base board was ordered separately.

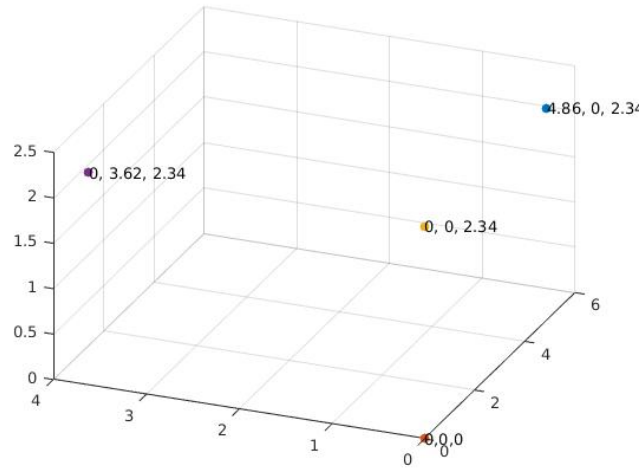

Fig. 10. Placement of four Anchor Nodes. All dimensions are in meters.

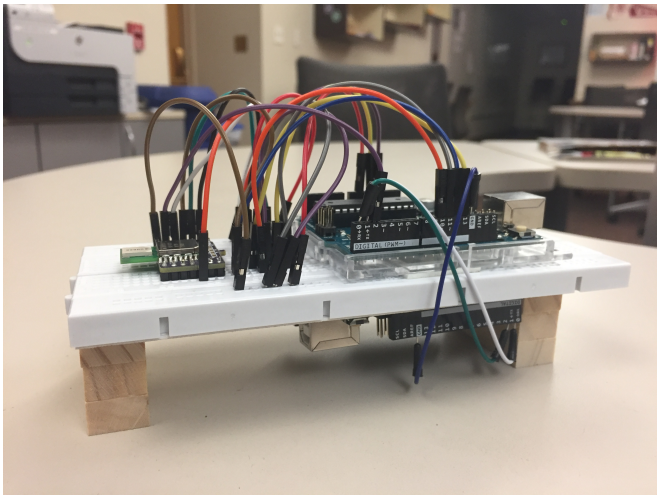

Fig. 11. TAG node: DWM1000 and Arduino on top; Arduino UNO Wifi is mounted on the bottom

- DWM1000 works on 3.3V while the Arduino Uno works on 5V. We had to procure level shifters to downshift the required SPI lines.
- The library for 2-way ranging was not stable with more than 2 anchors. These were identified as arising due to race conditions in the library code. After considerable probing, we were able to narrow down the bug to the variable whose access was shared.
- The distance measurements given by DWM1000 was initially off by about 1m. After plotting the actual distance vs the distance measured by DWM1000, it was evident that there is an offset between the two values. These offsets were recorded for each piecewise interval. A function was written which would convert the uncalibrated distance measured by DWM1000 to the actual distance by subtracting the corresponding offset.
- The Arduino Uno board did not have enough RAM to run both the ranging code as well as the WiFi library. The workaround for this problem was to attach another Arduino to act as a WiFi network processor for the tag. This Arduino would receive the ranging distance value over UART and transmit the data over WiFi.
- The addition of another slave Arduino for WiFi relaying brought in additional constraints in keeping the Tag physically small and concise. We had to design custom legs for the breadboard to house both the Arduinos without bloating the Tag dimensions.

V. Future Work

Our system can be improved further which are mainly based on the challenges we faced during our implementation stage. The current system is tested with a single mobile node and four anchor nodes. The system can be tested with multiple mobile nodes to check the performance. However, we believe that the four anchor nodes will not be able to service the range requests from multiple tag nodes, resulting in inaccurate ranges and delayed localization.

The current solution uses a two board approach to provide connectivity to the localization system over WiFi. This is mainly due to the limited RAM available on the Arduino boards. The DWM1000 library can be ported to a more powerful SoC like CC3200 to achieve a one board solution. With a more powerful SoC like CC3200, the localization equation solving can be done on the SoC itself. This eliminates the need for a separate computation backend server.

Time-Difference-of-Arrival algorithm can be used to perform localization with multiple mobile nodes. This will reduce the load on the anchor nodes. We expect this system to be more scalable as the number of mobile nodes increase. However, this approach requires accurate time synchronization between the mobile nodes. For time synchronization [9] proposes few methods which involve a master anchor transmitting clock synchronization packets every fixed interval. This packet is used by the other slave anchors to synchronize their time with the master anchor node. We also propose an iterative process to synchronize the clocks by localizing the 4th anchor(known co-ordinates) using the other 3 and changing the clock skews to increase the localization accuracy in every iteration.

Our system currently requires measuring the precise locations of the anchors and programming it into the Tag before the deployment of the system. A SLAM inspired approach can be taken wherein the user can place the anchors around the room. The Tag then guides the user to different landmarks around the room. This information is then used to learn the dimensions of the room. This approach is demonstrated and further elucidated in [2].

Security is another aspect that must be considered for real world deployment. A malicious agent must not be able to know the location of the Tag. This will require encrypting the messages transferred between the anchor and the tags.

VI. Conclusion

In conclusion, with our design choices and system architecture, our system is able to perform localization of mobile nodes in an indoor environment successfully. Our implementation is a completely self sufficient system and makes no assumptions regarding the availability of WiFi, internet or power source. Therefore the system can be used to perform localization in any indoor setting. A key observation from this project is that placement of anchor nodes have a significant effect on the accuracy of the system. We have achieved a best case accuracy of 10cm and 20cm accuracy on an average. Figure 11 shows our final Tag assembly.

VII. Milestones

The following were the key milestones during the course of the project:

Week	**Progress**
Mar 27-31	Basic Arduino Code test, Solder 2x DW1000 modules, breadboard prototyping, Python interface
Apr 3-7	Mid term demo. Level shifters interface, DW1000 baseboards arrive, Tag and anchor nodes bring up
Apr 10-14	Develop TOF code with 5 DW1000 modules working at once
Apr 17-21	Change library for robust multi-anchor network connectivity and addressing
Apr 24-28	Test TOF Code with 4 anchor+1 tag. Accuracy measurements, integrating solver code with python interface code and GUI extension
May 1-5	Prep for final demos, code and H/W cleanup, calibrations, WiFi Integration

VIII. Work Partitioning

The work was divided between the team members as follows:

Work Items	Team Member
Python UI, data processing/Algorithm, socket programming for WiFi connection, DWM1000 offset calibration, Arduino-DWM1000 library bug fixes	Tejus
Embedded programming - DWM1000 Interface, TOF Algorithm, DWM1000 offset calibration, Anchor placement tuning	Albert
Embedded Programming - WiFi Chip Interface, Data Transfer to PC Over WiFi, Arduino-DWM1000 library bug fixes	Anirban
Board bring up, HW interface, other setup	All

References

[1] M. Bouet and A. L. Dos Santos, "Rfid tags: Positioning principles and localization techniques," in *Wireless Days, 2008. WD'08. 1st IFIP*. IEEE, 2008, pp. 1–5.

[2] P. Lazik, N. Rajagopal, O. Shih, B. Sinopoli, and A. Rowe, "Alps: A bluetooth and ultrasound platform for mapping and localization," in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2015, pp. 73–84.

[3] Z. Yang, C. Wu, and Y. Liu, "Locating in fingerprint space: wireless indoor localization with little human intervention," in *Proceedings of the 18th annual international conference on Mobile computing and networking*. ACM, 2012, pp. 269–280.

[4] P. Lazik, N. Rajagopal, B. Sinopoli, and A. Rowe, "Ultrasonic time synchronization and ranging on smartphones," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2015 IEEE*. IEEE, 2015, pp. 108–118.

[5] 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*. ACM, 2012, pp. 421–430.

[6] T. Ye, M. Walsh, P. Haigh, J. Barton, and B. O'Flynn, "Experimental impulse radio ieee 802.15. 4a uwb based wireless sensor localization technology: Characterization, reliability and ranging," in *ISSC 2011, 22nd IET Irish Signals and Systems Conference, Dublin, Ireland. 23-24 Jun 2011*. Insitution of Engineering and Technology, 2011.

[7] M. Yavari and B. G. Nickerson, "Ultra wideband wireless positioning systems," *Dept. Faculty Comput. Sci., Univ. New Brunswick, Fredericton, NB, Canada, Tech. Rep. TR14-230*, 2014.

[8] B. Gaffney, "Considerations and challenges in real time locating systems design," *DecaWave white paper, Dublin*, 2008.

[9] C. McElroy, D. Neirynck, and M. McLaughlin, "Comparison of wireless clock synchronization algorithms for indoor location systems," in *Communications Workshops (ICC), 2014 IEEE International Conference on*. IEEE, 2014, pp. 157–162.

[10] T. Trojer, *Arduino DW1000*, 2017 (accessed March 9, 2017). [Online]. Available: https://github.com/thotro/arduino-dw1000

[11] K. Shadi, *Python Localization*, 2017 (accessed March 9, 2017). [Online]. Available: https://pypi.python.org/pypi/Localization