Beacon Indoor Navigation System

Andre E. Compagno, Joshua J. Facchinello, Jonathan E. Mejias, and Pedro Perez

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — This paper discusses the design methodology for creating an indoor navigation system using beacons that emit Bluetooth signals to a listening Android device, which provide important localization information. Topics included in the beacon design range from proper antenna design to utilizing solar cells and various batteries for power to interface with the Nordic nrf51822 system on a chip QFN package. Topics involved in the application software design include Bluetooth signal processing within the Android operating system, algorithms for user localization, optimal pathfinding for navigation, DynamoDB utilization for retrieving environment information, and user interface design for ease of use.**

***Index Terms* — Indoor navigation, Bluetooth, beacons, antennas, low power.**

I. Introduction

This project provides an inexpensive, low-power, and expandable infrastructure for an indoor navigation system. The goal of this system is to allow a user that has an Android device to be able to connect to a building's navigation network, input where they need to go within the building, then be guided by audio through the building to their desired destination. GPS is not a viable option for moving through a structure indoors due to the possibility of signal loss and the inability to localize a user in a building with multiple floors due to GPS only working in latitude and longitude and not three dimensions, which is the reason for using beacons. Android was chosen as the application platform due to its wide prevalence in the mobile device market as well as its open-source model. The specific device chosen for navigation in particular will be Google Glass, which will allow navigation information to be easily conveyed to the user.

The hardware design for the beacons, which will be placed throughout a building, consist of using the Nordic nrf51822 SoC, containing an ARM Cortex M0 processor as the controller, which is powered by both a Lithium-ion battery and a solar cell with the solar cell being the primary source. At a determined rate, the Bluetooth signal is emitted from an omnidirectional antenna. The Google Glass device then receives the signal along with a received signal strength indication (RSSI) value. Using the RSSI value the user's approximate location within the building is snapped to the known location of the beacon that is closest to the user. Then, with a destination given by the user, a path is constructed between the user and their destination. The user is then guided by Glass to the destination with the application tracking the user's progress warning the user if they go off the path. Upon reaching the desired destination, the process is complete.

II. Apple iBeacon Protocol

iBeacon is a Bluetooth Low Energy (LE) protocol developed by Apple in order to extend Location Services in iOS [1]. Fig. 1 displays all of the data included in an iBeacon message.

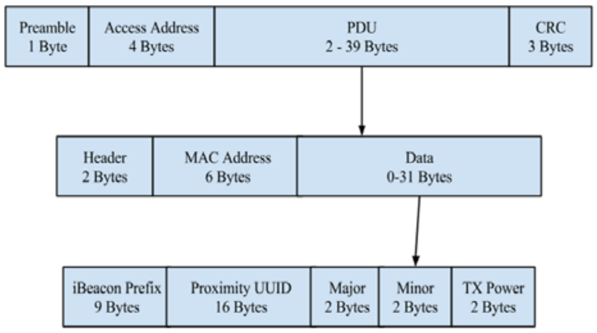


Fig. 1. iBeacon advertisement package

The iBeacon protocol carries three separate values: the Universally Unique Identifier (UUID), the Major value, and the Minor value. The UUID is a sixteen byte ID. The UUID defines a broad group of iBeacons and would be common to all iBeacons being used by a specific application or location. The Major value is a two byte unsigned integer and is usually used to define a more specific group of iBeacons within the group defined under a UUID. The Minor value is also a two byte unsigned integer that defines a specific iBeacon within a group of iBeacons with the same Major value. There are other values, which are not explicitly specified by Apple. One of these values is the calibration RSSI. This value is the two's complement of the measured RSSI value at one meter. It is used to calculate the estimated distance between the user and the beacon. Due to the imprecise signal readings, using the beacon signals to directly approximate user distance is impractical, thus forcing proximity to be used as the method for localizing the user.

If the signals were more reliable, trilateration could be done to more accurately determine the user's location. The iBeacon prefix is nine bytes of information including manufacturer information and Bluetooth signal flags.

III. Amazon's DynamoDB for Building Information

In order for the user to be able to navigate through a building, the application must have access to the building information that is important for navigating. Important aspects of traversing through a building include room names, number of floors, beacon positions, and a collection of virtual nodes that map out the structure of the building, which is used within the pathfinding module. DynamoDB will be responsible for holding all of this building information and for giving the information whenever the application requests it.

DynamoDB is a NoSQL database provided by Amazon, which offers a generous free-tier service. Using the service also remains simple due to the Amazon Web Services (AWS) SDK, which provides an API for creating and querying tables using the Java programming language.

Upon startup of the application, the application waits for a certain number of different beacon signals to be received. Once the number is reached, the UUID, which uniquely specifies the building the user is in, with the highest frequency amongst the set of beacon signal readings is assumed to specify the building the user is currently located in. This UUID is then used as a primary key to query for all of the building's navigation information. The application waits until the query is successful and the information is received before parsing it and using it for the rest of the application's processes.

Another purpose for the database is to hold the tags of specific nodes that are in the building. Within the pathfinding module discussed later, some virtual nodes will contain special names or tags that describe the particular node and its role within the building. Tags can be viewed as words that describe the node's location within the building. For example, if the building being mapped is a school, one of the nodes might be at the entrance to a classroom. Some tags for that node might be the room number of the classroom, the name of the teacher that teaches within the classroom, or the purpose of that classroom, such as biology lab or chemistry lab. In order for the mapping from tags to nodes to work properly, each node must have tags that are unique from any other node within the building, so that a unique tag maps to one and only one node. Only nodes that mark building landmarks need to have tags. The rest of the nodes do not need tags, because they are only used to traverse to the landmark nodes as sort of a middle point. Some examples of landmark nodes are nodes that are entrances to rooms, elevators, staircases, or any node that can be used as a possible destination of the user. Once the tag-to-node mappings are stored in the database, the database will be able to accept a tag as an input and return the node mapped to that tag. This specific query will be used to determine where the user would like to go within the building. The application will ask the user where they would like to go and the user will be able to input tags for their desired destination. If the tag cannot be found in the system, then the user will be notified. The user will also be able to view all of the tags or rooms within the building in a list format, so that if the user's given tag does not match any in the database, they can still select the room they want from the list if somehow they did not input the right tag.

Ultimately, if anything within the navigation system is needed to be stored, it will be held in the database. Currently, there is just the beacon and node information stored, but if any other data is needed to be added to make the system work more efficiently, it can be added to the database.

IV. Application User Interface Design

The Google Glass application is designed to be usable by both visually impaired and non-visually impaired users. Text and images will be displayed on the screen, but the device will also communicate with the user by speaking any text currently being displayed. In most cases, the user will communicate with the application by simply using their voice to speak a key phrase. The only times the user will need to use the touchpad on the device is to wake the device in order to start the application and exit the application. Visually, the application will mostly consist of brightly colored text and images on a black background. This is due to the way in which the display on the device works. The black sections will actually look transparent, while the other sections will be visible. Having a brightly colored background will direct too much light into the user's eyes making it uncomfortable, especially in a dark environment.

Whenever the device is on the home screen, as indicated by the clock and the "ok glass" prompt, the user can simply speak the key phrase "ok glass" in order to access the list of application-specific key phrases. This list includes many different key phrases specified by applications which correspond to an action the device carries out. In order to start the directions application, the user will simply say "get indoor directions." This will start the indoor navigation application which then prompts the user to speak their desired destination. The application then checks that the location is valid by cross-referencing the database. If the destination is invalid, the application prompts the user and shortly after closes. In order to specify a new destination, the user will need to start the application again from the home screen. If the destination is valid, the application starts generating directions from the user's current location to their specified destination.

Once the application has generated the directions for the user, it will start prompting the use to follow said directions. The application will display a marker on the screen to visually guide the user. The application will also periodically speak to the user by dictating the next step of the directions. If at any time the user starts going off-course, the application will prompt the user that they are in a warning zone, both by displaying a message and speaking to the user to get back on course. Once the user gets back on course, the application continues to display and dictate the directions as it was before. If the user continues moving more and more off-course, they will be notified that they are off-course and new directions are generated. Once the user reaches the destination, the user is notified and the application exits. If at any time the user wants to exit the application, they can simply swipe down on the touchpad located on the side of the device.

V. Beacon Detection Library

Within the application is a beacon detection library that was built around the Android's Bluetooth LE SDK and is used to facilitate the process of scanning, filtering, and interpreting the beacon's modules.

Even though it has already been supported by the hardware in the past, the Bluetooth LE support was not introduced into the Android operating system until the release of Android 4.3 (Jellybean). With the update, Google introduced the startLeSan method into the BluetoothAdapter class, which allows the developer to scan for Bluetooth LE devices and consume the data being advertised by those devices. Although this allowed for some Bluetooth LE support, it was still behind the Bluetooth LE APIs offered by other mobile operating systems, like iOS and Windows Phone 8. Since Android only allows for scanning, it means it can only act as a central device and does not support acting as a peripheral device. With the recent release of Android 5.0 (Lollipop), Google has added software support for the emulation of multiple simultaneous peripheral devices.

In order to use Bluetooth LE in an Android application, two system permissions are needed to be declared in the application manifest: android.permission.BLUETOOTH and android.permission.BLUETOOTH_ADMIN. If the permissions are not declared, the operating system will not allow the application to access the Bluetooth hardware. All Bluetooth activity is done through the BluetoothAdapter. This object is shared across all applications that use Bluetooth.

Fig. 2 shows a high level view of the architecture of Bluetooth LE in Android. With permissions correctly set in the application, the BluetoothAdapter can be extracted from the BluetoothManager. If the device does not support Bluetooth or Bluetooth is not enabled, the BluetoothManager returns null. In order to define what happens whenever a device is found, an LeScanCallback object is created with the onLeScan method overridden. Whenever a Bluetooth LE device is found, the onLeScan method from the LeScanCallback object is called. In order to start the scan, the startLeScan method form the BluetoothAdapter is called with the LeScanCallback object passed in as a parameter.

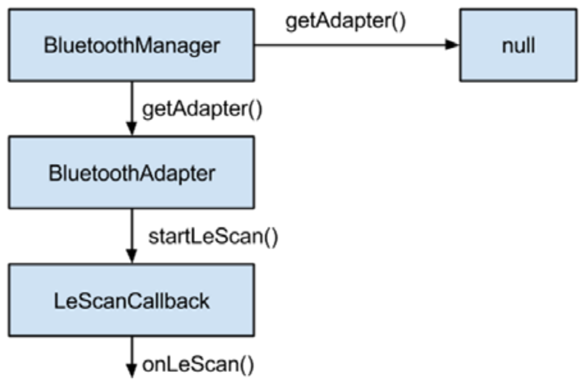


Fig. 2. Architecture of Bluetooth LE in Android

While scanning for beacons, all Bluetooth LE devices show up in the scan. This requires that the application filter out the miscellaneous Bluetooth LE devices that are unrelated to the indoor navigation system. This check is performed when the data scanned in is being transformed into a Beacon object. The method uses a Regex expression to check the advertisement data. If the check fails, a null value is returned instead of the Beacon object telling the application to ignore the data scanned.

With each device scanned, three parameters are received: a BluetoothDevice object, the RSSI value, and a byte array with the advertisement data. The advertisement data is converted from a byte array to a String so the Regex expression can be used. The Regex expression being used is the following: "^0201[a-f0-9]{2}1aff[a-f0-9]{4}0215[a-f0-9]{42}.*". The "0201" section is found in all beacons. This is followed by "[a-f0-9]{2}" which matches by any two hexadecimal digits which represent flag values in the advertisement. The following section, “1aff” is also constant across all beacons. This is followed by the company identifier “[a-f0-9]{4}” which is made up of any four hexadecimal digits. The next three digits, “0215”, correspond to the iBeacon advertisement indicator and is constant across all beacons. Lastly, the “[a-f0-9]{42}” section ensures that there are forty-two hexadecimal digits. This corresponds to the UUID, Major value, Minor value, and the calibration RSSI value.

VI. User Localization

For user localization, trilateration was researched as a possible method in which to calculate the user’s location. Trilateration takes the known beacon positions and the approximate distances between the beacons and the user to calculate the user’s approximate position within a given space. However, due to the imprecise distance approximations between the beacons and the user determined from the Bluetooth signals, trilateration was found to be an inadequate method for localization.

The next option was to use the user’s proximity to the beacons as a method for determining their location. Instead of calculating the user’s location, the user’s location is set to the position of the closest beacon. As beacon signals are received by the Android device, the average for each beacon is used to determine whichever one is closest. The downside of this approach is that it requires many more beacons than when using trilateration due to the beacons actually becoming part of the path of the user. The approach is not the most optimal, but it is ultimately more effective than trilateration.

An optimization to this proximity method that is done to lower the number of beacons needed for a given area is to spread the beacons farther apart from one another, take the average of each incoming beacon signal, take the two closest beacons and then determine if they are within a given threshold value. If both beacons are within the threshold, then the user can be snapped to a location exactly in between both beacon locations. This nearly doubles the area the same number of beacons can cover compared with the previous method of snapping the user to the closest beacon while not jeopardizing localization accuracy. With the previous method, beacons spaced three meters apart produced adequate results. However, with the optimization, beacons spaced five meters apart produced comparable results to the previous method. In an effort to have more resolution for the user’s location, the optimization was tried with beacons three meters apart, but it produced a user location with too much noise constantly snapping back and forth between the two closest beacons and the position in between both beacons. For this reason, a beacon spacing of five meters is used.

VII. Pathfinding

The pathfinding module is responsible for using the collection of virtual nodes queried from DynamoDB that map the layout of the building the user is in to generate an approximate shortest path from the user’s location to their desired destination. Pathfinding within any system involves moving from one point to another within some sort of space. The process involves determining the area one can traverse, as well as determining obstacles that block the path needed to move from one location to another, if possible.

Most pathfinding algorithms use a collection of nodes giving the positions of all obstacles and open areas within the traversal space. Within the beacon navigation system, these nodes represent either open, walkable areas within a building, or walls that block off movement from the user. The pathfinding module takes in the node closest to the user determined by the user localization module and the desired destination node to find the shortest viable path between both nodes. If a path cannot be found, the user will be notified that there is no valid path that can be used to reach the user’s destination. This, however, should never happen, because a given room should always be able to be visited if the building is of sound design and the data is without error. If the destination node is located on a different floor than the user is currently on, it should be able to guide the user to the staircase or elevator that would lead to taking the shortest, most efficient path. However, the actual algorithm that handles finding the shortest path between two nodes does not have knowledge of multiple floors, but rather searches a two-dimensional space that maps out a floor. Another object must be added to handle connecting the paths of each floor into one multi-floor path, which is discussed later on in this section when discussing floor sequencing.

Pathfinding within the indoor navigation system will require a significant amount of computing power. Because of this, the module should be limited to run only when needed for the user to be able to reach their destination. Updating the user’s path constantly would use excess battery within the Android device and possibly slow down the system. For this reason, the path is not calculated at a rate, but is only calculated towards the beginning following the user’s destination input and whenever the user goes off the path. Functionality for determining if a user is off-course is discussed in the next section.

There are a variety of different pathfinding algorithms that have been used in many systems throughout the past. A number of tweaks and variations have been made to long-used pathfinding algorithms that introduce small improvements that give more efficiency to particular use cases. However, even though an algorithm may add to a previous one, it does not automatically make the newer one better. Each specific system must look at what the requirements are that system and which algorithm is better suited for the system. This not only holds true for pathfinding algorithms, but also any other type of algorithm in general.

A* was heavily considered as a possible pathfinding algorithm for the navigation system. A* uses a few distance approximations to generate the shortest possible path, or close to it, from one node to another within a graph of nodes. A* traverses through the graph by looking at each of the neighboring nodes to the current node and checking if the distance from the neighboring node to the starting node using the current path, and the approximate distance from the neighboring node to the destination node is smaller than when calculating it from a different path. If the sum of these two distances is greater than when following a previous path, the neighboring node's values are left alone. If the sum is less, then the neighboring node's two distance values are updated to the current calculation and the parent is updated to be the current node. The parent is used to store the path so that when backtracked to the start node, the path taken can be constructed. Once all neighboring nodes to the current node have been updated, the node that has the smallest sum of the two distance values is made to be the current node. This process loops until the destination node is reached. The path can then be reconstructed by backtracking through the parent nodes.

Similar to A* is Theta*, which follows the exact same process with the added step of checking line of sight between the current node's parent and the neighboring nodes. If the current node's parent is within line of sight of the neighboring node and the path to the neighboring node is shorter following the current path, then the neighboring node's parent is set to the current node's parent. If there is no line of sight, then the current node is set as the neighbor's parent as long as taking the current path is shorter than taking a previous, different path. In the end, this results in paths that have fewer turns and longer straightaways. Paths with fewer turns are easier to give directions for as adjacent nodes in the calculated path do not have to be adjacent nodes in the graph, which gives results similar to how a human would walk through a building. They would not turn every meter to stick with the A* path, but rather just walk straight and turn only when they really need to. Fig. 3 shows the difference between the A* and Theta* algorithms when calculating a path between two nodes. The path is in green, while the start and destination nodes are in red, the obstructed nodes that cannot be walked through are in gray, and the open, walkable nodes are white.

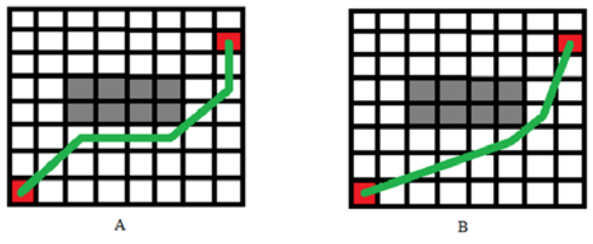


Fig. 3. A* (A) versus Theta* (B) path comparison

Another aspect of pathfinding that remains to be discussed the floor sequencing for paths that move across multiple floors. The floor sequencer is responsible for stringing together the paths generated from Theta* into a multi-floor path that the user can follow. Because Theta* only calculates the path of one floor at a time and the system needs to be able to calculate the shortest path across multiple floors, an extra step must take place to connect the different paths from each floor into one long path. The link between the floors are staircase and elevator nodes that must be stored with the building information inside DynamoDB. In order to link Theta* paths together, an adjacency list of elevator and staircase nodes is constructed, so that if a user is on one floor and needs to get to another, every possible route of staircases and elevators will be calculated that an take the user to their destination. Fig. 4 provides a visualization of how the adjacency list connects different floors.

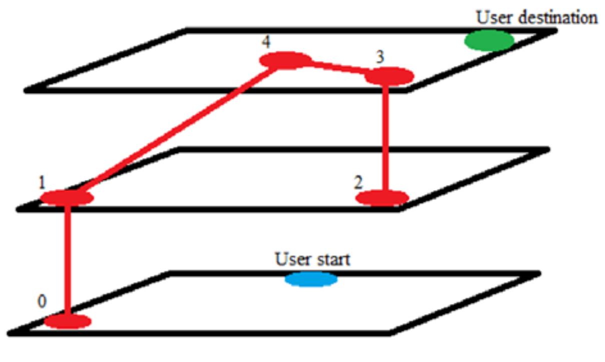


Fig. 4. A building with three floors and connections

Once all possible routes are calculated, each connector node, which is red in Fig. 4, is used as a start and end node and is passed into Theta*. Once Theta* returns, the path is appended onto the previous paths calculated by Theta*. The floor sequencer then returns the multi-floor path that is calculated the fastest as the approximate shortest path.

VIII. User State Tracking

The application must also be able to handle the possibility of the user moving off-course from the determined path, which is handled in the user state tracking module. Because the nodes in a Theta* calculated path can vary in distance from one another a perimeter system is used to surround the user as they move from one node to another. Fig. 5 shows how the perimeter works by surrounding the two current nodes of the path the user is currently on. One node is the previous node the user passed, while the other node is the node the user still needs to reach.

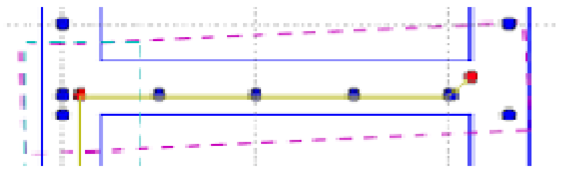


Fig. 5. A graph showing the perimeter that surrounds the two nodes that mark the node that the user has passed and the node they are headed to. The perimeter is the dashed purple rectangle with the two nodes being red in the figure. The blue dots represent beacons and the blue lines represent the walls of a building.

Initially a circular perimeter was thought to have been a simple solution. By placing the center at the center between both nodes, a radius threshold could be used to ensure that the user stayed within the perimeter. However, because nodes can be very far apart, it would result in large space to the sides, which would not be effective at warning the user that they are headed in the wrong direction. Thus, a rectangular perimeter is used. To be more specific, two rectangular perimeters are used. The inner perimeter marks the boundary between an on-course and warning state for the user, whereas the outer perimeter marks the boundary between the user being in a warning zone where the application warns them they are close to being off-course, and being off-course.

IX. RF Optimization

Indoor environments pose great challenges for the implementation of wireless systems. Our system utilizes the Bluetooth LE protocol which operates at around the 2.4 GHz frequencies. Using this protocol is hugely advantageous in that it is supported by the majority of modern handheld and wearable wireless consumer electronics which increases the possible user base considerably. This protocol also allows for superior low power usage which suits our low bandwidth, low power specifications of our system. A few drawbacks in working with this system is the inherent occupied spectrum at around 2.4 GHz – i.e. Bluetooth shares radio spectrum with other wireless communications protocols such as WiFi. At this frequency, indoor Bluetooth LE signals quickly attenuate and limit the effective distances of each beacon.

Due to these disadvantages, great consideration was exercised when deploying the indoor navigation system beacons. Through hypotheses and testing, it was determined that the best location for each individual beacon is on the ceiling, center of the hallway. Beacons placed on walls proved highly ineffective due to the nature of microwave radio propagation in an indoor environment – e.g. multipath fading effects were easily observed in this configuration. Beacons were also most effective placed above the average pedestrian height in order to reduce the attenuation stemming from foot traffic. It was also largely helpful to run a rudimentary site survey to locate any potential sources of interference such as WiFi transmitters; beacons were not placed near strong sources of interference in order to obtain optimal RF performance.

Another consideration for beacon placement is the displacement between each beacon node in our system. The goal is to be able to maximize the distance between each node without sacrificing accuracy of the indoor navigation system in order to keep implementation costs down. The software algorithm for indoor mapping relies on the perceived RSSI and thus the distance between these beacons are limited by the useful data reading of the RSSI. At a certain displacement, RSSI values became unreliable. We gathered experimental data and used statistical analysis (i.e. curve fitting, linear regression) in order to optimally determine the distance between each beacon to be approximately five meters. Further optimization of our system can improve these figures.

X. Hardware Design

Due to the nature of high speed digital electronics and operating in the microwave radio frequency, great care was taken in the layout of the PCB, choice of passive components, and choice of antenna.

Our beacon is powered by the Nordic nrf51822 which provides a differential antenna output with an embedded low noise power amplification for the antenna. A pi circuit shown in Fig. 6 matches the antenna output to 50 ohms and also converts the differential output to a single ended output which allows for increased compatibility with many 2.4 GHz antenna designs. The capacitors and inductors of the pi network are all kept as close as possible to the chip in order to limit parasitic and stray effects of long feed lines. Using freeware antenna design software such as AppCad, the antenna output copper trace is specified at 30 mil thickness in accordance to our board thickness, material, wavelength, impedance, etc. Other electronics are kept far away from the pi network to further reduce possible interference. Vias are limited in order to further reduce potential change to the RF environment. Decoupling capacitors are terminated as soon as possible near the chip and power traces are kept short in order to further keep unwanted electromagnetic interferences away.

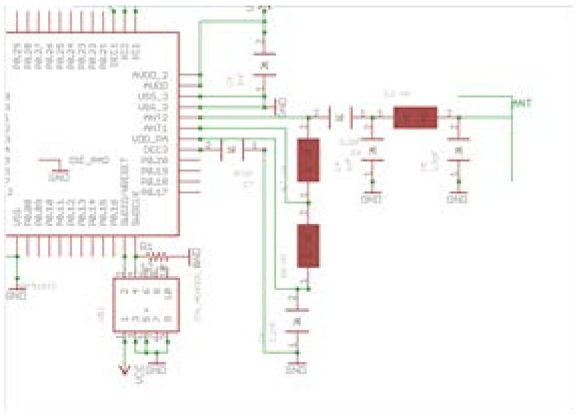


Fig. 6. The pi network matching the antenna output to 50 ohms and providing a single-ended antenna output is shown.

Our beacons utilize a PCB trace antenna, shown in Fig. 7 is in the “Inverted-F” configuration which is a standard in many wireless handheld applications. This design choice allows for decreased form factor which allows for the device to remain miniature. The basic operations of the Inverted-F Antenna (IFA) rely on the capacitive effects between the long arm of the antenna and the ground plane in addition to the inductive effects of the shorting trace. The ground plane copper pour is abstained from being directly under the antenna. This antenna provides an omni-directional radiation pattern making it useful for ease of placement and 360 degree coverage of indoor corridors and atriums.

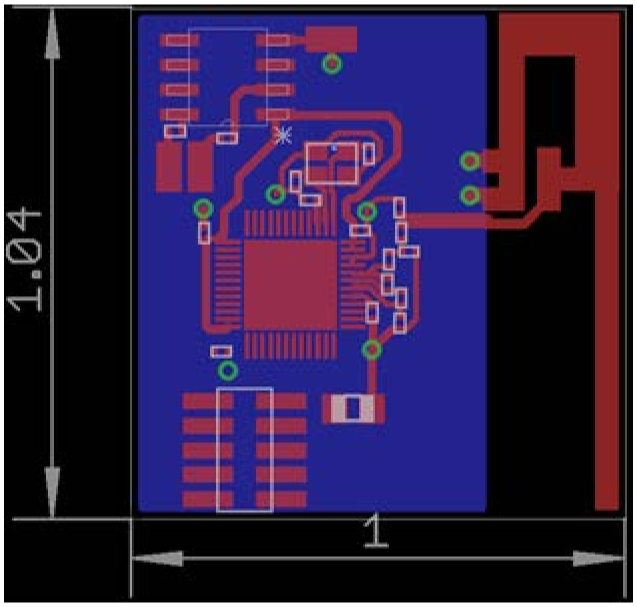


Fig. 7. The overall layout of the PCB including the inverted-F antenna. Note the tight spacing of passive components, short traces, and vias close to the chip. Board measurements are in inches: 1 x 1.04 inches.

A 1 oz. pour copper plane is used on the bottom layer with an overall 2-layer PCB thickness of 1.6 mm FR4. The copper plane serves as a ground plane allowing for the reduction of electronic noise caused by high speed digital electronics and also works in conjunction with the PCB trace antenna. Alternatives considered include using Teflon for overall increased RF performance and utilizing 4-layer PCB for more efficient routing. These, however, would unnecessarily increase our budget.

XI. Power Design

The nrf51822 QFN chip is meant for low power applications. The chip requires 1.8 – 3.6 volts to operate and a minimum of 10 mA. The 2032BP 3V button batteries have been chosen as a source because they provide 3.3V fresh out of the package and can still function when slightly worn out. Also, the same battery is already used within the industry for beacons that are manufactured. Since many beacons will be set up throughout a building for long periods of time, efficiency of the batteries must be considered.

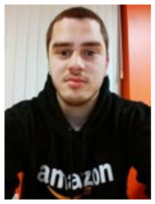
Solar power is also incorporated in the power design. The MP3-37 solar strip was chosen due to its small and flexible profile as well as its output voltage. Being four and a half by one inches in dimensions, the flexibility allows for the creation of a beacon housing that is not bulky. The solar strips have been tested indoors and outdoors; outside with direct sunlight, a voltage of 3.6 volts can be obtained. Indoors, depending on the location of the strip, 2.8 to 3.1 volts can be obtained. The solar strip will be beneficial for devices set on the ceiling near a light source and those placed near a window. Also, improving the beacons with solar energy use is a new effort that has not yet been done with beacons in the industry, which still must rely on battery power, which can diminish quickly depending on the beacon's settings.

A chip that can manage interchanging between the button battery and the solar strip was chosen. This chip, the ICL7673, provides a battery backup system for the beacon. The beacon's primary source of voltage will be the solar strip with secondary (or backup system) being the battery. The chip works by multiplexing both power sources into the nrf51822 SoC. The ICL7673 gives priority to the solar cell. If the solar cell is producing enough power for the beacon to operate, then solar power is used. However, if the solar cell is not producing enough power, then the battery is used as the source of power. When the battery wears out and the solar strip provides suboptimal power, between 1.8 and 2.4 volts, the device will still work, but the signal will be weak. Another power consideration is when the device is programmed or flashed to act as a Bluetooth device. The input voltage must be 3.3 volts in order for the pin headers to transmit the information to the system. In order to meet this requirement from the pin header datasheet, a power supply will be used to supply 3.3 volts in order to flash the beacon to act as a Bluetooth device.

VII. Conclusion

This discussion shows the basic overview of the indoor navigation system and all of the system's main processes and components. Though beacons with Bluetooth remain a challenge for indoor localization, certain techniques discussed can be used to create an adequate indoor navigation system that can guide a user easily through a building. With improved Bluetooth signal detection and precision, a more effective navigation system could be developed. Furthermore, research into other possible mediums for localization other than Bluetooth could also be done in the future to possible improve the effectiveness of the indoor navigation system.

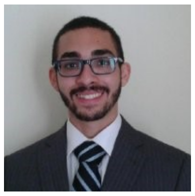
Biography



Andre Compagno is currently a senior at the University of Central Florida. He plans to graduate with his Bachelor of Science in Electrical Engineering in May of 2015. He completed an internship during the summer of 2014 and received a full-time offer.

Josh Facchinello is currently a senior at the University of Central Florida. He plans to graduate with his Bachelor of Science in Computer Engineering in May of 2015. He is currently interning at Lockheed Martin with UCF's YES Program and will work full-time with Lockheed Martin upon his graduation.

Jonathan Mejias is currently a senior at the University of



Central Florida College of Electrical Engineering and Computer Science and is pursuing a Bachelor of Science in Electrical Engineering. He has a concentration in communications in addition to past industry experience working with AT&T.

Pedro Perez is an Electrical Engineering student at the University of Central Florida and will be graduating in the spring of 2015. He will pursue a career in power engineering and work towards obtaining a PE license in the future.

References

[1] Apple Inc, 2 June 2014. [Online]. Available: https://developer.apple.com/ibeacon/GettingStarted-with-iBeacon.pdf. [Accessed November 2014]