Test Report – Second Prototype

To: Professor Pisano

Team: 7 (MuseumMate)

Date: 03/07/2024

Group Members:

John Culley jculley@bu.edu

Kai Imery kimery@bu.edu

Yanyyang Zhang yz21@bu.edu

Ananth Sanjay ananths@bu.edu

Kwadwo Osafo kosafo@bu.edu

1.0 Equipment and Setup Summary

The prototype we are demonstrating can be split between two broad categories; hardware and software. The software consists of separate applications that run the server (backend), frontend, handheld user devices (TourTags), and beacons. The backend consists of various code that describes the functions of our networking and storage capabilities. Our handheld devices (TourTags) send data to a server on our backend through UDP and the data received is stored in an InfluxDB database through the use of an API. The server is also capable of using trilateration to determine what room a given TourTag is in, given the nearest three beacons. The front end consists of a React.js application which allows users to log into our system and receive positioning data corresponding to their respective TourTags. It also includes several quality-of-life features, which include, but are not limited to, text-to-speech features in multiple languages, multimedia overlays giving additional information about specific exhibits, and a dashboard for system administrators. The last part of our software consists of code that allows the TourTags to detect UWB signals between themselves and the beacons, which allows them to compute an accurate measurement of the distance between themselves and the beacons. This data is then processed and sent to the server through the use of WiFi at recurring intervals.

Our hardware has shifted significantly since our first prototype test, mainly due to the replacement of Adafruit Huzzah32 ESP32 Feather Boards with Makerfabs ESP32 UWB, which was done in order to utilize UWB technology to determine distances between TourTags and the beacons, which we deduced will be more robust and accurate than BLE technology. Additionally, we have included a Li-Ion Charger in the TourTag design, which allows for charging of the handheld device without needing to detach the lithium-ion battery from the device. The ESP32 UWB is split to serve as either a TourTag or a beacon (housed in a 3D-printed enclosure). The beacons are powered by USB 5V power supply units connected directly to electrical sockets, and the TourTags are powered by PKCELL LP552535, which is a rechargeable lithium-ion mobile power supply unit. RFID modules will also be attached to the TourTags to demonstrate their functionality.

The physical setup of our prototype consisted of plugging the UWB beacons into electric sockets at various predetermined locations in the lab. These predetermined locations are shown in **Figure 1.** The TourTag was also connected to a mobile rechargeable power unit and placed in a 3D printed enclosure. Next, the backend (Node.js server), the frontend (React Native application), and the ChatGPT server (Golang) were started on two of our computers. The log output from our respective applications allowed us to verify whether each part of the system was integrated properly at startup. After confirming successful connections, we proceeded to demonstrate the location services through the app by revealing the change from 1.1 to 1.2 as the TourTag was moved to the other side of the lab. Then, and RFID tag was scanned, revealing multimedia content retrieved from the server. With this content, the language, text to speech, and ChatGPT services were properly tested. Below is a detailed list of all the materials used for our setup:

Hardware:

- **Makerfabs ESP32 UWB**
- **Adafruit Li-Ion/Li-Poly Charger [v1.2]**
- **3D Printed Enclosure for Beacons**
- **PCB for Handheld Device**
- **USB 5V Power Supply**
- **PKCELL LP552535 3.7V 420mAh**
- **Passive RFID Tags**
- **RC522 RFID Scanner Module**

Software:

Server:

- **Node.js**
- **Express.js**
- **InfluxDB**
- **MinIO**

ChatGPT Server:

- **Gin Framework**
- **ChatGPT API (3.5 Turbo)**
- **Golang**

Frontend:

- **React Native**
- **Expo**
- **React**

User Device:

- **UWB Module**
- **Battery Charging System**
- **RFID Reader**
- **Connection to Campus WiFi**
- **UDP Client**

Beacon:

- **UWB Module**

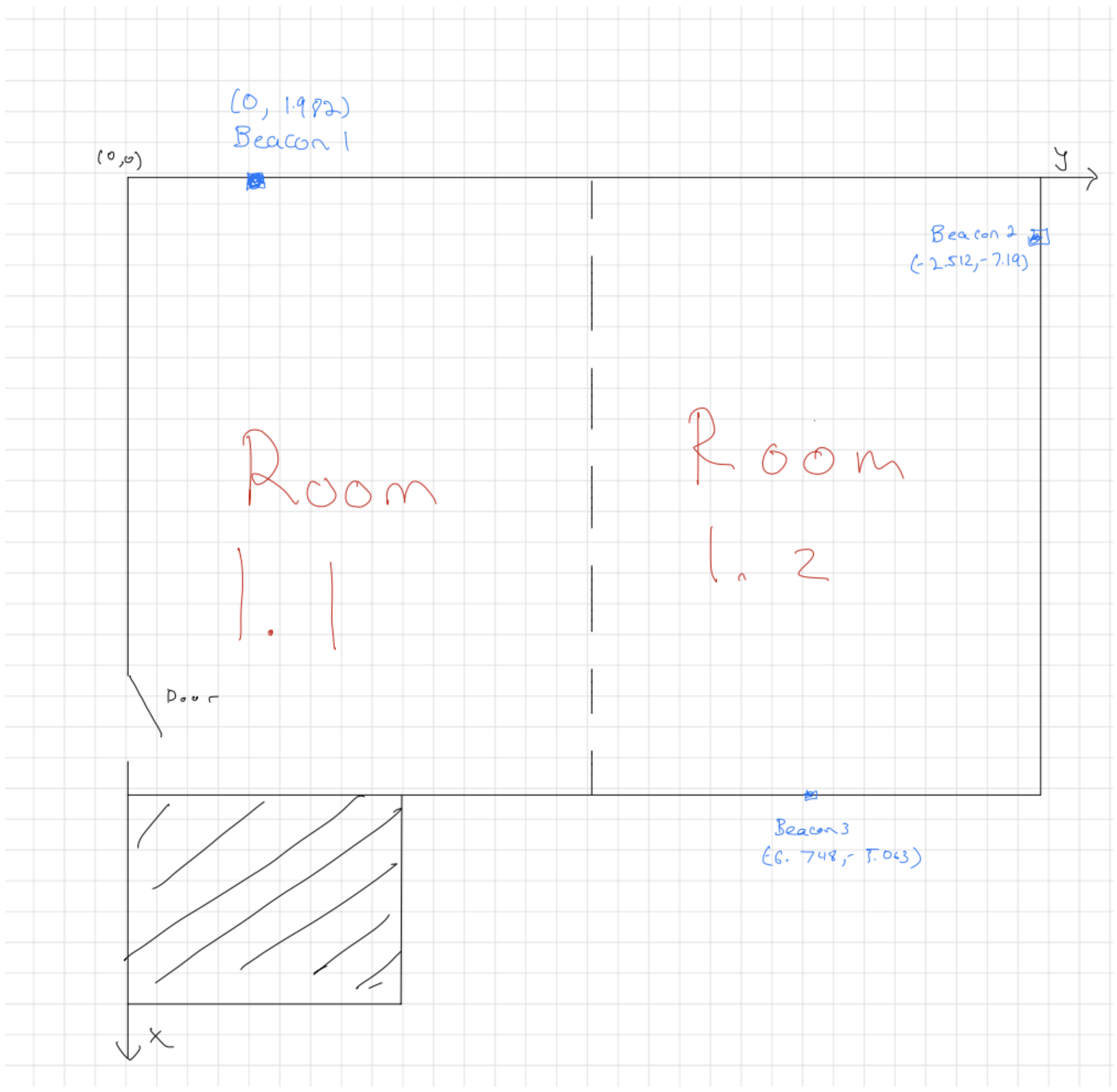

Figure 1: Locations of beacons and room sections

2.0 Testing and Measurement Summary

Included in our test plan was a list of measurable criteria that defined vital components of our system and was proof of proper functionality of our prototype. Below is the list of proposed measurable criteria. It is important to note that references to the number corresponding to each measurable criteria in the list will be made in the sections it precedes.

Hardware:

1. Handheld (User) devices accurately deduce the distance between themselves and a given beacon within approximately +/- 0.2m error bounds.

2. 3D printed enclosures should not interfere with the WiFi and UWB antennas of the enclosed beacon
3. RFID tag is successfully scanned by handheld device, and the corresponding RFID tag ID is recorded.
4. Updated distance measurements from UWB readings should be sent to the central server upon the detection of signals from 3 unique beacons in a single sampling interval.
5. Contents of packets sent via UDP to the central server should be consistent with predetermined protocols.

Server:

1. Successfully receives UWB data via UDP from user device
2. Stores and processes UWB data with a queue that is updated each time new data is received
3. API Testing:
    a. /location/:userID (GET)
        i. Returns user location to the front end
    b. /path/:userID (GET)
        i. Given a userID and room, return route to get to destination
    c. /tsp-path (POST)
        i. Given an array of rooms, return the shortest path to visit all the rooms
    d. /rfid/:bucketName (GET)
        i. Given a bucket name, return all associated multimedia from minIO
4. In 5-second intervals, occupancy by room is obtained and sent to InfluxDB to be timestamped

ChatGPT Server:

1. API Testing:
    a. /chat (POST)
        i. Returns the response from ChatGPT based on the prompt given in the request
        ii. Upon successful request, response is either retrieved from cache or if it is a new prompt it is generated with the ChatGPT API and returned

Front-End:

1. Initializes the app and swiftly navigates to the HomeScreen within 2 seconds, offering primary navigation.
2. Activates the BarcodeScanner to utilize the device's camera for scanning a barcode and displaying results within 5 seconds.
3. Displays a list of available tours, including TimedTour and Explore, within 3 seconds upon selecting TourTypes.
4. Accurately locates the user's device and retrieves the corresponding map image within 6 seconds using the CurrentLocation feature.
5. Ensures smooth screen transitions without freezes exceeding 2 seconds and responsive interactive elements within 2 seconds.
6. API Testing:
    a. /location/:userID (GET)

i. Retrieves the current location of the user's device and updates the display or UI elements.

b. /path/:userID (GET)

   i. Provides a navigation route for a specified userID to a selected room within the app.

c. /tsp-path (POST)

   i. Receives a user's selected rooms and computes the shortest visitation path.

d. /rfid/:bucketName (GET)

   i. Returns all multimedia associated with a given bucket name from minIO.

e. WebSocket ws://10.192.45.20:8080

   i. Establishes real-time connection for RFID data, updating UI with new object info.

f. Google Translate API https://translation.googleapis.com/language/translate/v2

   i. Translates text to selected user language for improved accessibility.

g. GoLang API http://10.192.39.193:4040/chat (POST)

   i. Accepts prompts and returns informative responses for interactive user engagement.

Our testing procedure began with setting up and starting the hardware elements and the backend and frontend. Next, the application was rendered on a mobile phone in order to demonstrate the user interface to the system. The mobile application allowed the user to use the phone's camera to scan a QR code on the handheld device (TourTag) enclosure, which in turn allowed the application to identify what TourTag the user possesses, and this information is used to query the backend for the specific location information of the TourTag. The functionality of various buttons and screens on the mobile application was also demonstrated. The application was then navigated to a screen which displayed a map similar to that in Figure 1 which was configured to serve as a live map with an indicator showing what room the user is currently in.

The next step in our demonstration involved moving the TourTag to room 1.2 in order to demonstrate the functionality of the live map feature. Our TourTag obtained UWB distance values from the beacons in the room, which continuously broadcast the signals. Every 10 seconds, the distances measured were sent to a server on our backend over WiFi through UDP in the form {beaconID, userID, distance}. The data received is then processed within the server to update the distance values for each of the TourTags and through trilateration determines the x and y coordinate of the device in the room. Based on the diagonal coordinates of the room, it is then determined which room the TourTag is in. The frontend is then able to query the backend through an HTTP endpoint for the location information. The frontend was configured to allow for strict adherence to the measurable criteria in order to eliminate any latency and successfully meet the expectations.

The last step in our demonstration consisted of connecting an RC522 RFID module to one of our TourTags to demonstrate the functionality of the RFID aspect of our system. The TourTag was tapped against an RFID chip and the corresponding multimedia was populated on the mobile app. Here, the many languages, text to speech, and ChatGPT integrations were successfully tested.

2.0 Conclusions

Overall, our demonstration was a success and we were able to meet all the proposed measurable criteria. We received useful feedback from the reviewers of our prototype. The reviews consisted of suggestions about how often the location is being updated for each TourTag. There are many factors that need to be considered when determining how often the location should be updated. It is very important for the TourTag to preserve battery life and collect enough data samples before sending it to the server but it is equally important that the location is being updated fast enough to not interfere with the user experience. The feedback mentioned that the ten seconds we were previously using was too slow, so we have now implemented a new system that plans to optimize all of these important factors. The new implementation is a form of dynamic pooling. We first attempted to understand our product through a visitor's experience and determined that the most crucial time that the location needs to be updated quickly is when the visitor is navigating the museum. Because of this, we have implemented two different polling speeds: fast and slow. The TourTag will by default be in slow mode which is now going to update the location every 10 seconds, which is most suitable for visitors enjoying one room and exploring the museum with no destination in mind. This will allow for the TourTag to preserve precious battery life when it is not navigating. The fast mode aims at providing faster location updates to the visitor when they are navigating the museum. This is implemented by determining which TourTag is in navigation mode and turning their setting to fast, then when the TourTag reaches their destination, the mode is switched back to slow. The fast mode will update the location every 2 seconds which we believe is fast enough to not hinder the experience of the visitor while they attempt to navigate the museum.