Stride Engineering Notebook - Indoor Navigation

Project Overview

Falls are the leading cause of injury and death for those 65 and older, accounting for 27,000 deaths every year in the United States alone according to the Centers for Disease Control and Prevention. Walkers are widely known to be an effective way of minimizing fall risk, but they still have two major limitations. First, if they are placed out of reach, people frequently choose not to use them, resulting in massively increased fall risk. Second, hand brakes on wheeled walkers tend to be hard to trigger in a pinch and frequently require significant grip strength to use effectively, also increasing fall risk. Our team plans to solve these problems of walkers being out of reach and hard to brake by creating an electronics-enhanced walker that incorporates all of the benefits of a standard walker, along with our unique Summon and Autobrake features. The Summon feature will use motors, software, and sensors to enable the walker to navigate itself to the user at the tap of a button or with a voice command. The Autobrake feature will use motors and pressure sensors to ensure that the walker will brake whenever the user's hands are not on the walker handles.

Summon Feature Summary

The main feature of our invention, the walker’s ability to self-navigate to the user at the tap of a button, is implemented with a short-range communication protocol. The walker receives data from a transmitter within a wrist-wearable device on the user, allowing the walker to know the user’s relative distance and direction. At first, we decided to create a bluetooth tracking system using Bluetooth Low Energy (BLE) chips on Arduino Itsy Bitsy boards by placing three BLE receivers located on the left, right, and front sides of the walker. After developing this system, we found that it was difficult to make the data as accurate as we wanted, so we decided to prototype with ultra-wideband (UWB) beacons instead. The UWB location system was significantly more accurate and what we ultimately decided to go with for our project.

Indoor Positioning Technologies

Types of indoor positioning methods / real-time locating systems (RTLS)

- Trilateration
    - GPS
    - UWB
- RSSI-based
    - BLE
    - Wifi
    - ZigBee
    - LoRa
- Triangulation
    - LiDARs
- Mixed
    - BLE + angle of arrival (AoA)
- Odometry
- Inertial
- Optical
    - QR codes
    - Optical flow
    - Motion capture
- Sensor fusion
    - IMU + ultrasonic
- Other
    - Li-fi

    - RFID
    - Magnetic

Considerations for methods / RTLS: update rate, price, power consumption, weight, size, tolerance to interferences, location vs. location + direction, embedded IMU, data communication to and from mobile beacons, line of sight

RSSI-based RTLS

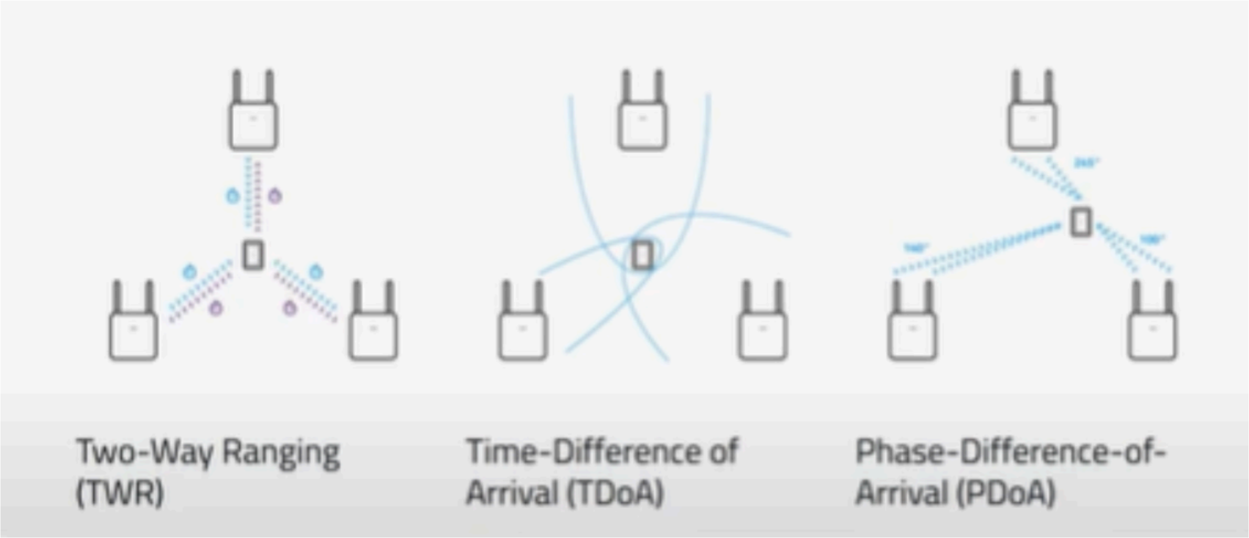
- Not designed for position
- Can estimate not high precision distance from target
- Potentially be negatively influenced by objects moving in-between transceivers (disrupt electromagnetic signals)

IMU-based RTLS

- Drifts a lot
- Magnetometers work poorly indoor
- Can be sufficiently precise
- Need constant drift elimination by external systems

Precise RTLS must have line of sight

UWB


Two-Way Ranging (TWR)
Time-Difference of Arrival (TDoA)
Phase-Difference-of-Arrival (PDoA)

LiDARs

- Precise but not designed for positioning and navigation
- Good for obstacle avoidance and detection
- Not designed for positioning
- Quality of positioning depend on complexity of environment

Visual positioning

- Quality depends on lighting and distance (few meters at most usually)

Takeaways

- RSSI based method could be beneficial
    - BLE, LoRa, Wifi specifically
    - Many Arduino modules that use these methods (could be more easily integrated)

BLE / Itsy Research

ItsyBitsy on Adafruit: https://learn.adafruit.com/introducting-itsy-bitsy-32u4/overview

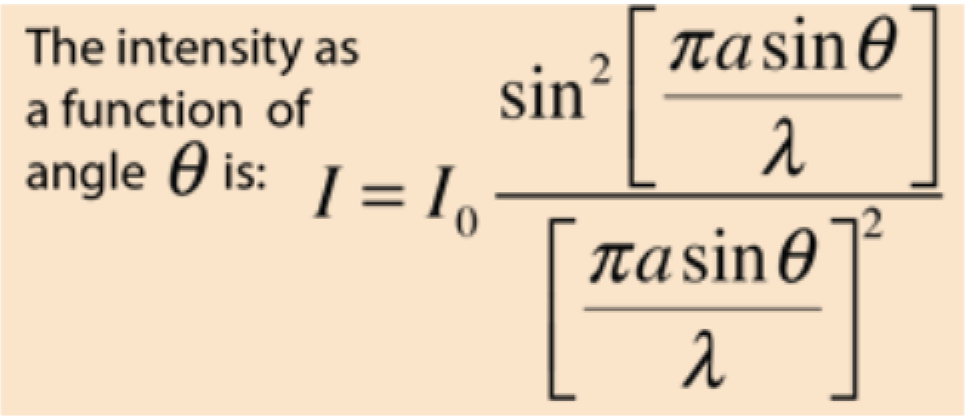
Specifications:

- ATmega32u4 onboard chip in QFN package
- USB bootloader with a nice LED indicator, AVR109 compatible (same as Flora, Feather 32u4, Leonardo, etc)
- Micro-USB jack for power, USB uploading and debugging, you can put it in a box or tape it up and use any Micro USB cable for when you want to reprogram.
- Can act as a USB HID Keyboard, Mouse, MIDI or plain USB 'CDC' serial device (default)
- Power with either USB or external output (such as a battery) into VBAT pin - it'll automatically switch over
- On-board red pin #13 LED
- 23 GPIO total - 6 analog in, 1x SPI port, 1x I2C port, 1x Hardware Serial port and 10 more GPIO, 4 of which have PWM
- Can drive NeoPixels, connect to sensors, servos, etc.
- Reset button for entering the bootloader or restarting the program.

BLE Housing Background

- Using Fraunhofer and simple single slit diffraction equations and assumptions (including treating the incoming RF signal as a series of plane waves at the slit)
- Enclosing a rough Faraday cage around the BLE receiver with only a single slit to allow in RF signals
  - Faraday cages do not have to be grounded in order to attenuate RF signals

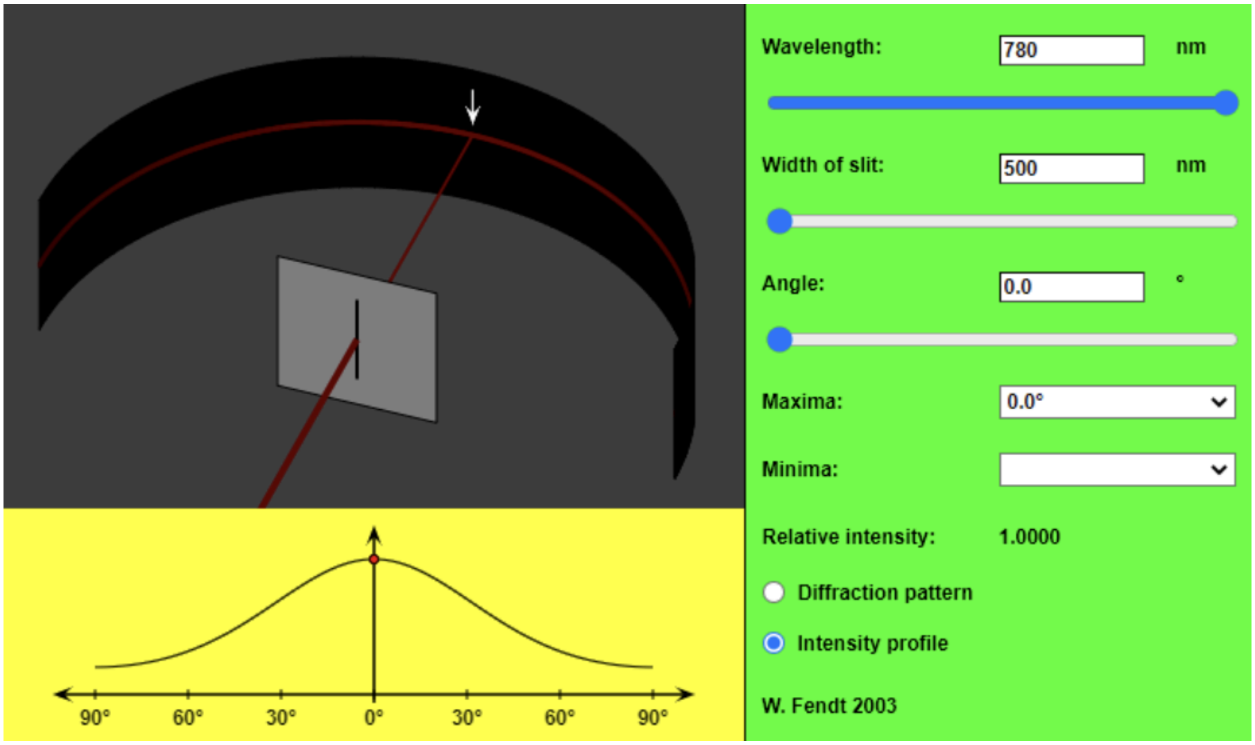
- Rotating the BLE receiver inside of its shielded housing results in the relative intensity of the RF signal reaching the receiver to drop varies per the equation below:



The intensity as a function of angle θ is:

$$I = I_0 \frac{\sin^2\left[\frac{\pi a \sin\theta}{\lambda}\right]}{\left[\frac{\pi a \sin\theta}{\lambda}\right]^2}$$

Effect of Slit Width vs. Wavelength



Wavelength: 780 nm

Width of slit: 500 nm

Angle: 0.0 °

Maxima: 0.0°

Minima:

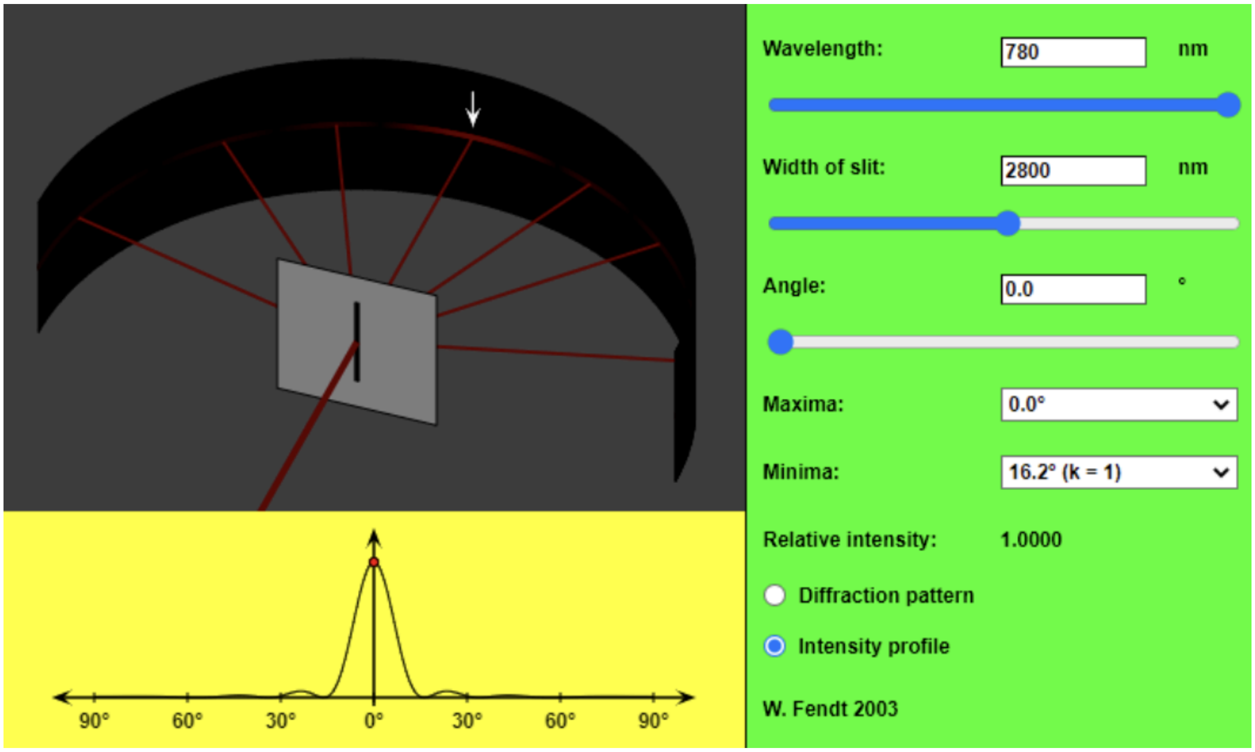
Relative intensity: 1.0000

○ Diffraction pattern

● Intensity profile

W. Fendt 2003

d < λ: No minima and little to no reduction in relative intensity



Wavelength: 780 nm

Width of slit: 2800 nm

Angle: 0.0 °

Maxima: 0.0°

Minima: 16.2° (k = 1)

Relative intensity: 1.0000

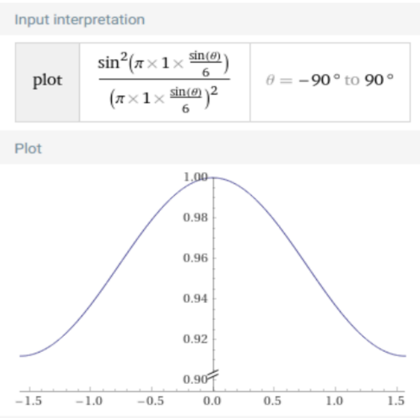
○ Diffraction pattern

● Intensity profile

W. Fendt 2003

d > λ: Diffraction pattern and steep reduction in relative intensity

A

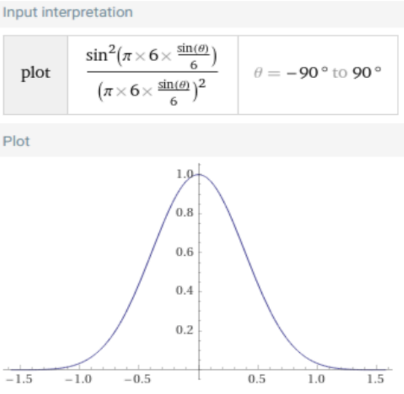


Input interpretation

plot $\frac{\sin^2(\pi \times 1 \times \frac{\sin(\theta)}{6})}{(\pi \times 1 \times \frac{\sin(\theta)}{6})^2}$ $\theta = -90°$ to $90°$

Plot

B

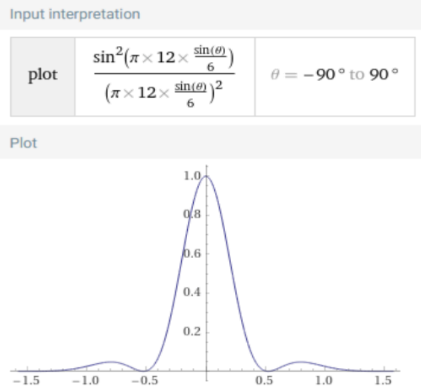


Input interpretation

plot $\frac{\sin^2(\pi \times 6 \times \frac{\sin(\theta)}{6})}{(\pi \times 6 \times \frac{\sin(\theta)}{6})^2}$ $\theta = -90°$ to $90°$

Plot

C



Input interpretation

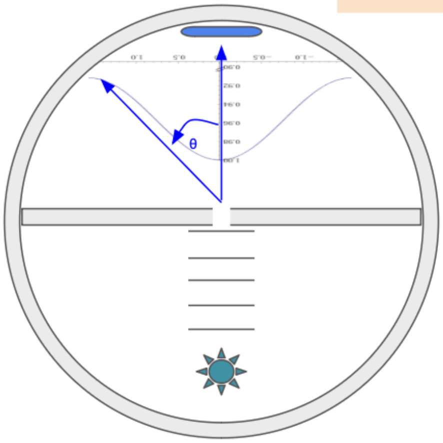
plot $\frac{\sin^2(\pi \times 12 \times \frac{\sin(\theta)}{6})}{(\pi \times 12 \times \frac{\sin(\theta)}{6})^2}$ $\theta = -90°$ to $90°$

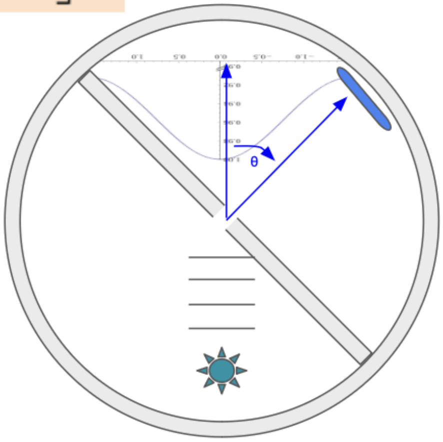
Plot

The slit size in the BLE housing is ~1 cm compared to the minimum BLE wavelength of 6 cm. This means that the relative intensity of pattern that we would expect to see should be similar to plot A.

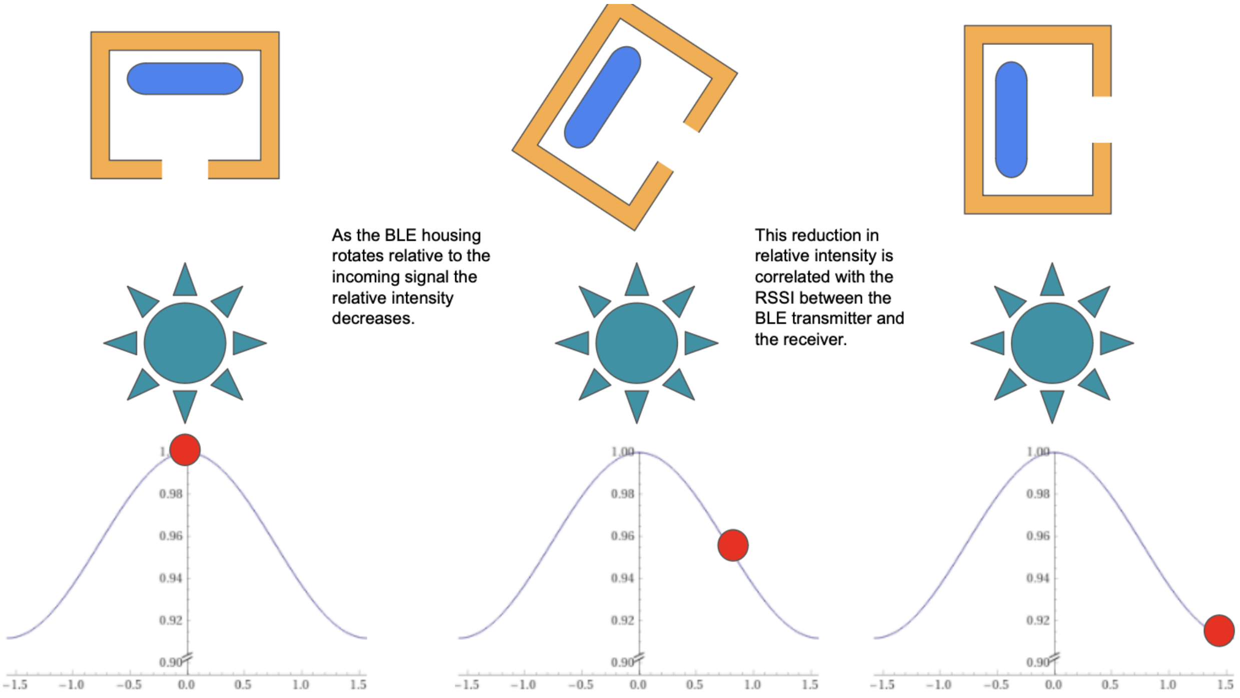
The intensity as a function of angle θ is:

$$I = I_0 \frac{\sin^2\left[\frac{\pi a \sin\theta}{\lambda}\right]}{\left[\frac{\pi a \sin\theta}{\lambda}\right]^2}$$



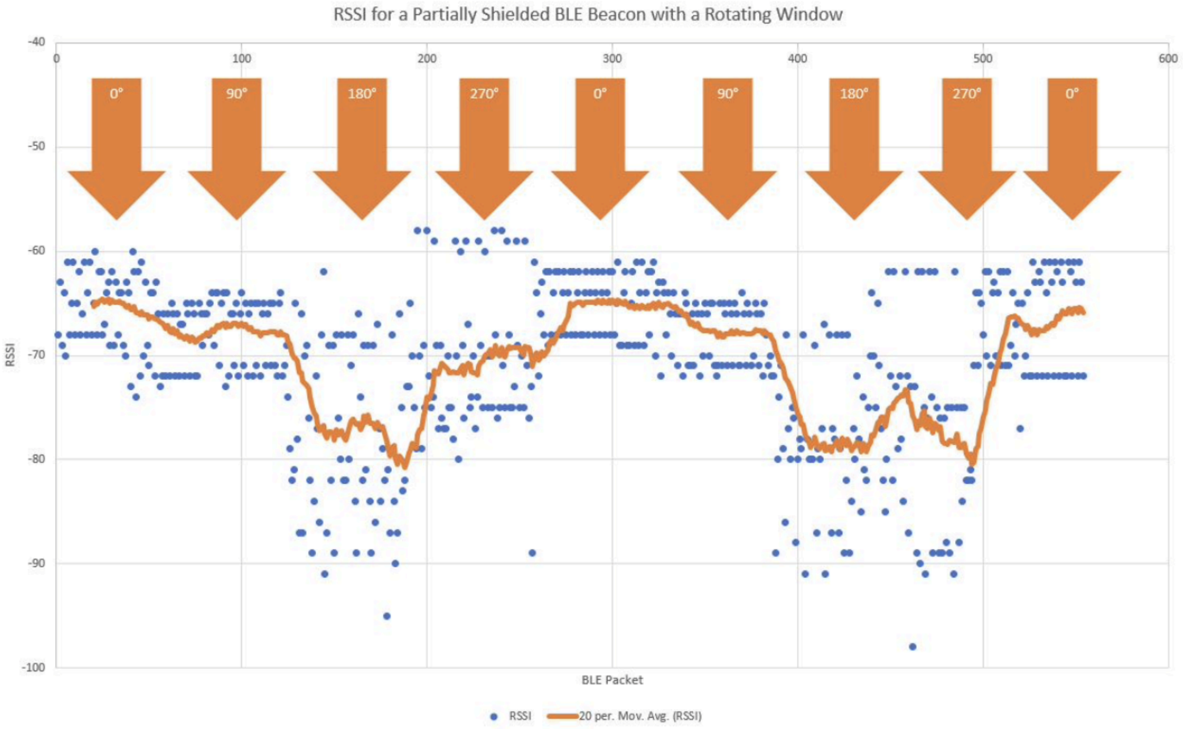


The theta in the relative intensity function is still valid regardless of which reference frame is rotating.



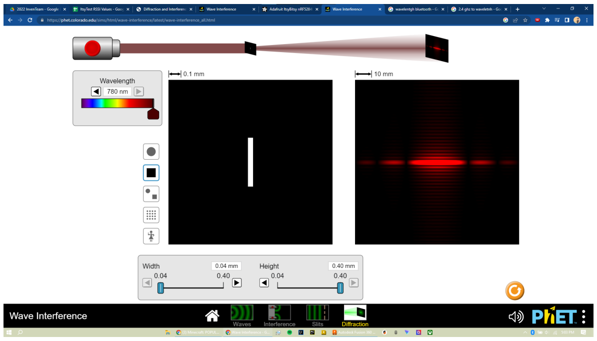
As the BLE housing rotates relative to the incoming signal the relative intensity decreases.

This reduction in relative intensity is correlated with the RSSI between the BLE transmitter and the receiver.



Simulation of Wave Interference

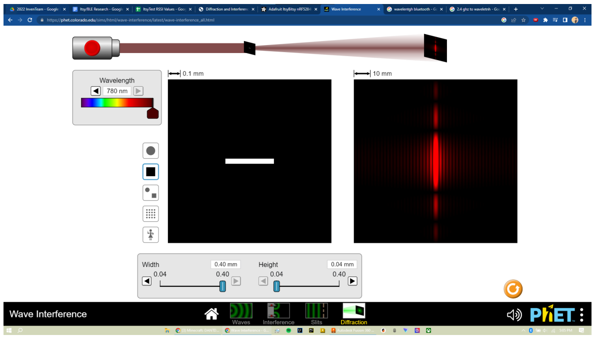
https://phet.colorado.edu/sims/html/wave-interference/latest/wave-interference_all.html


https://phet.colorado.edu/sims/html/wave-interference/latest/wave-interference_all.html
0.1 mm
10 mm
0.04
0.40
0.04
0.40
Wave Interference
Waves
Interference
Slits
Diffraction

Wavelength 780 nm

Width 0.04 mm

Height 0.40 mm


https://phet.colorado.edu/sims/html/wave-interference/latest/wave-interference_all.html
0.1 mm
10 mm
0.04
0.40
0.04
0.40
Wave Interference
Waves
Interference
Slits
Diffraction

Wavelength 780 nm

Width 0.40 mm

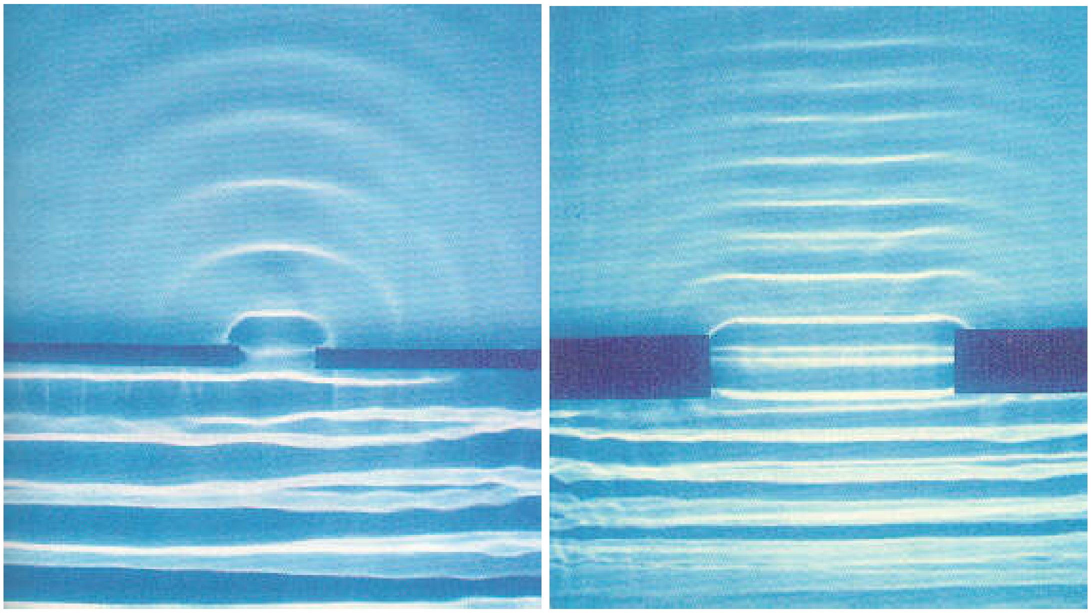
Height 0.04 mm

Along a vertical slit, waves diffract horizontally. Along a horizontal slit, waves diffract vertically. These tests were done in the visible spectrum, with its wavelength being orders of magnitude smaller than that of BLE signals, however the properties of diffraction stay the same.

Conclusion: In order to direct the BLE signals more effectively as we rotate from left to right, it may be more advantageous to have a horizontal slit rather than a vertical slit (meaning attempt to rotate the device by 90 degrees to see how it impacts the rotational effectiveness).

Notes on Diffraction

http://electron6.phys.utk.edu/phys250/modules/module%201/diffraction_and_interference.htm



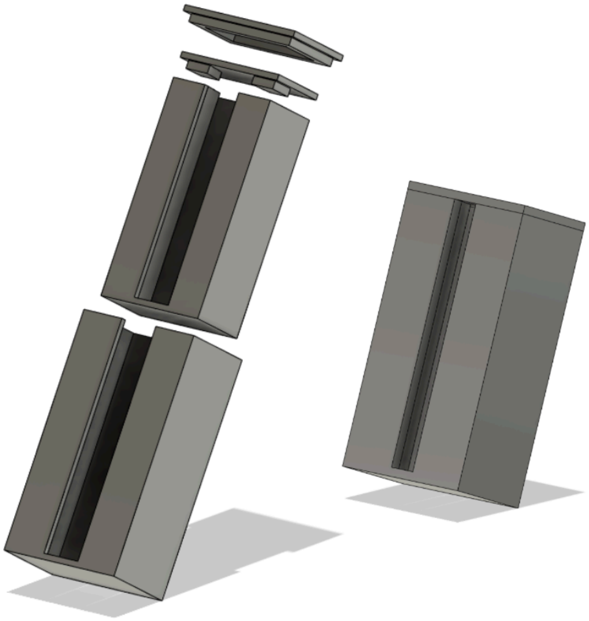
Somewhat counterintuitive, when a slit is smaller, the waves diffract along a greater angle, and when the slit is larger, the waves are more directed and diffract far weaker

Conclusion: If time permits, tests should be done to determine what size slit can be used to optimize for reduced diffraction, without sacrificing much directivity.

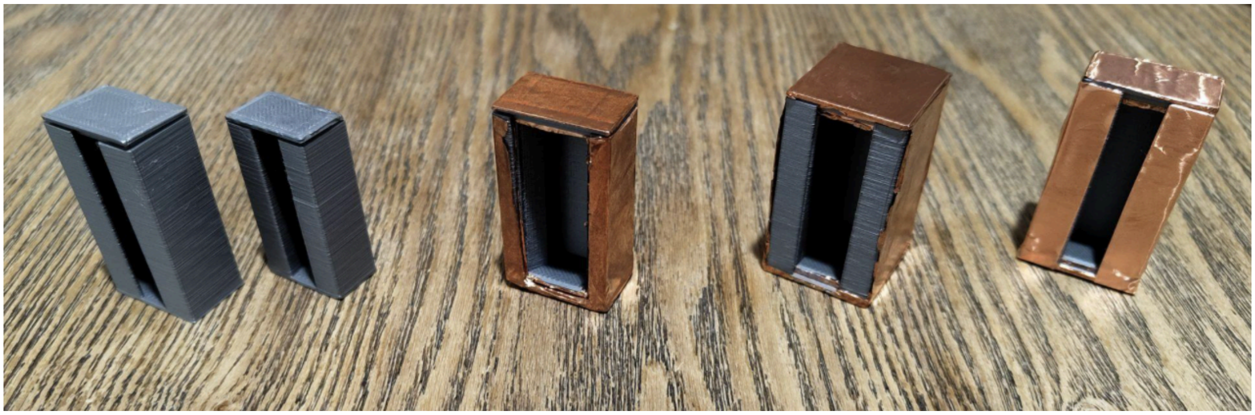
Developing BLE housing

To get the user's relative position, we transmit the strength of the signal in terms of Received Signal Strength Indicator (RSSI). Bluetooth signal is not directed in a specific direction and behaves more similarly to a field. To remedy this, we created housing units for the bluetooth receivers that are aimed at better directing the signal in one direction through a slit in the housing. With our three receivers, we are able to calculate the walker's position and direction relative to the user with several mathematical calculations in the program.

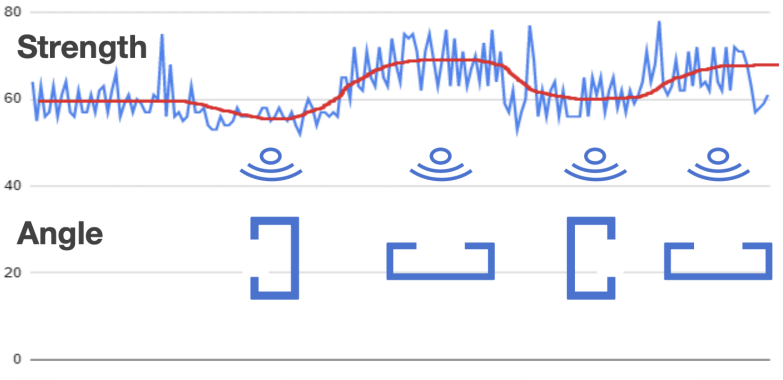
We spent an extensive time developing our housing units, similar to a faraday cage, to make the signal as focused as possible. To develop these housing units, we designed them using CAD software and 3d printed them.



The housing units contain two sleeves in order to wrap signal-focusing material in between the sleeves. It also has a slit in the center, which is where the signal comes out. We have extensively experimented with the dimensions of the slit. We noted that, contrary to our assumptions, a wider and shorter slit was more effective than a skinny and narrow slit. However, simply placing the bluetooth receivers inside a plastic box would not fix our problem with signal attenuation. Signal only bounces off metal material, more reactive metals being more effective at directly signal. We wrapped our housing units in copper tape, since copper is the most reactive, which can be seen in our several prototypes.



Graphical representation of data from the BLE receivers plotted in the Arduino IDE:



The differences in the y-value of the graph are from spinning the walker, indicating changes in direction.

Caveat with BLE method

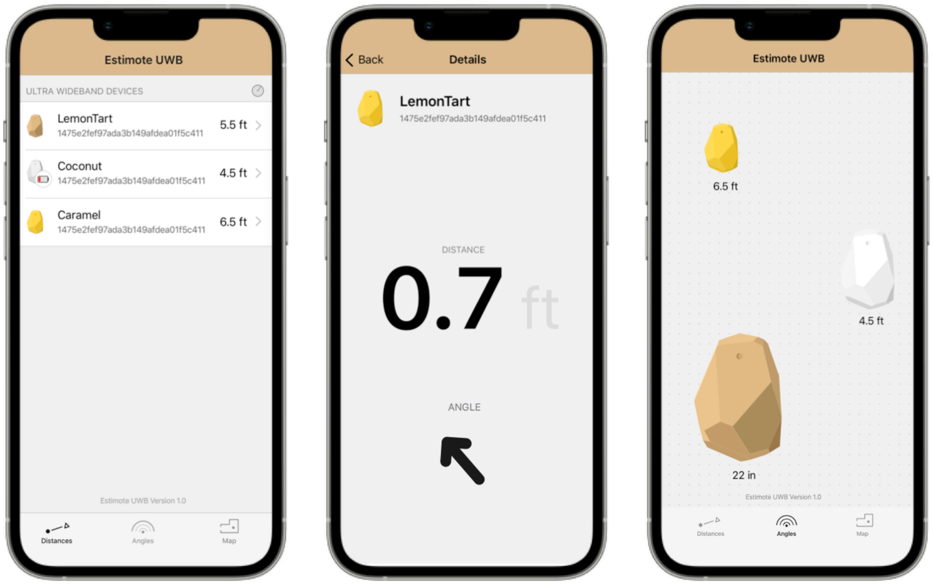
Even with several mathematical equations to manipulate the data to create a smooth line (rather than the fluctuating one as seen in the above graph), the RSSI values were still not accurate enough for the walker to locate the user. The values were off by several inches or feet, which could cause significant problems with our summoning feature. We only have a few weeks left before Eurekafest to solve this problem.

Ultra-wideband

Easiest way to solve accuracy problem: use existing technology that has been applied to indoor navigation and can reliable interact with Arduino to achieve our minimal viable product

Initial approach: use ultra-wideband beacons (similar to Apple's AirTags) to connect to an iOS app which connects via bluetooth to the Arduino board

Proven to work UWB beacon: https://estimote.com/



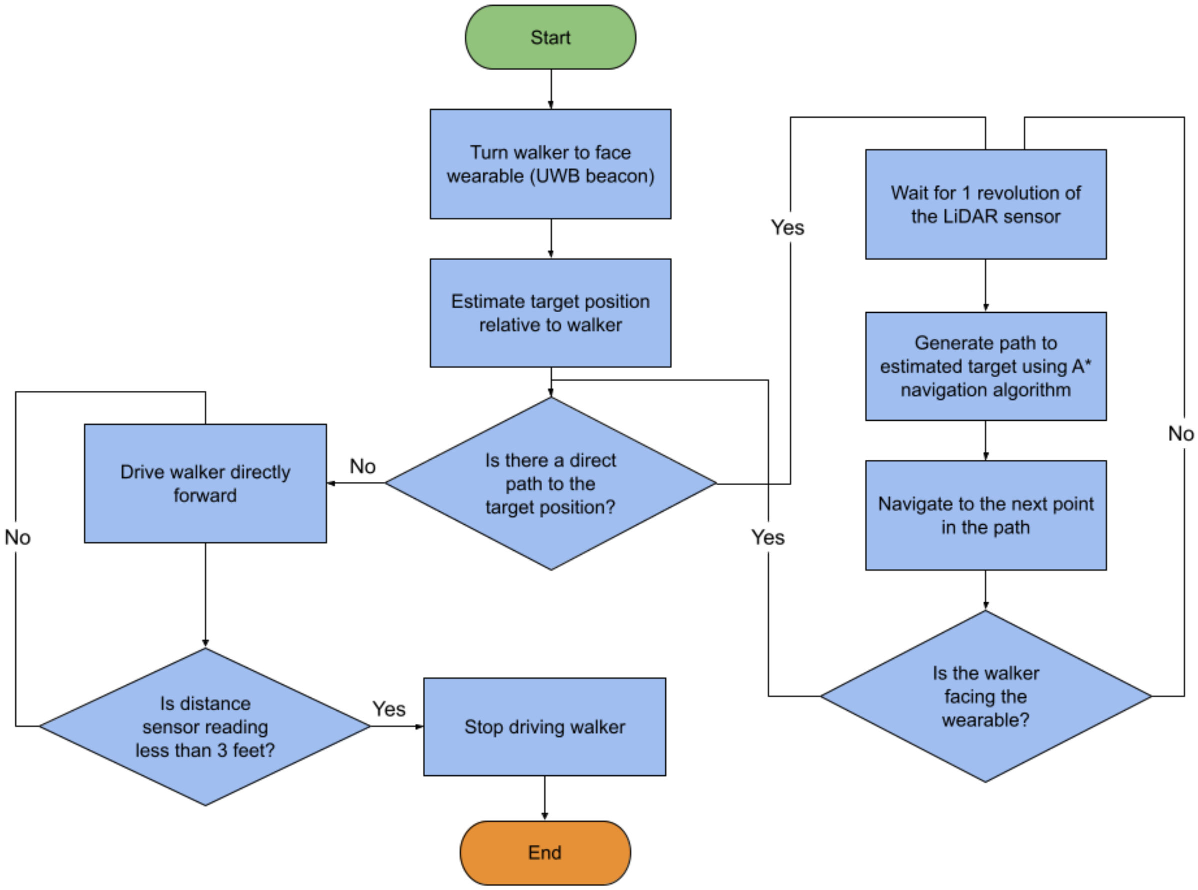
ULTRA WIDEBAND DEVICES	
LemonTart 1475e2fef97ada3b149afdea01f5c411	5.5 ft
Coconut 1475e2fef97ada3b149afdea01f5c411	4.5 ft
Caramel 1475e2fef97ada3b149afdea01f5c411	6.5 ft

How it works

Nearby devices that are UWB-enabled and have compatible apps connect to these beacons and compute precise distance and angle. Multiple antennas in the phone help to compute the angle of signals.

Flowchart of navigation algorithm

My team and I created the following flowchart to base our walker's navigation algorithm on.


Start
Turn walker to face wearable (UWB beacon)
Estimate target position relative to walker
Is there a direct path to the target position?
No
Drive walker directly forward
No
Is distance sensor reading less than 3 feet?
Yes
Stop driving walker
End
Yes
Wait for 1 revolution of the LiDAR sensor
Generate path to estimated target using A* navigation algorithm
Navigate to the next point in the path
Is the walker facing the wearable?
Yes
No

Arduino code before obstacle avoidance implementation

The Arduino code that I wrote for the navigation algorithm before obstacle avoidance implementation shown below is split up into snippets of code for each 2 boxes of the flowchart.

The following snippet of code establishes bluetooth connection with the iOS app using the Bluefruit chip. It then continuously checks for if data is sent from the iOS app to the Bluefruit chip; the data is the distance and angle of the walker relative to the UWB beacon as the UWB beacon is connected to the iOS app. Once the distance and angle data is received, the walker turns towards the UWB beacon using the angle data.

```
// C++ libraries
#include <iostream>
#include <sstream>

// BLE libraries
#include <Arduino.h>
#include <SPI.h>
#include "Adafruit_BLE.h"
#include "Adafruit_BluefruitLE_SPI.h"
#include "Adafruit_BluefruitLE_UART.h"
#include "BluefruitConfig.h"

// Motion/sensor libraries
#include <VescUart.h>
#include <SoftwareSerial.h>
#include <stdlib.h>
#include <MPU6050.h>

// Constants
const bool debugresponse = true;
const int vescbaudrate = 9600;
const int wheelDiameter = 6;            // Diameter of the wheel in inches
const float gearRatio = 2;              // Gear ratio of walker
const bool movedForward = false;        // Flag to track if the robot has completed
moving forward

// Variables
float distance = 0;               // Distance between walker and UWB beacon
float angle = 0;                  // Angle to turn between walker and UWB beacon

// Motor instances
VescUart vescML;
VescUart vescMR;
int oldSpeedL = 0;
int oldSpeedR = 0;

// Bluefruit initialization
Adafruit_BluefruitLE_UART ble(Serial1, BLUEFRUIT_UART_MODE_PIN);
// IMU sensor initialization
```

```
MPU6050 mpu;

// Error handler for BLE connection
void error(const __FlashStringHelper* err) {
 Serial.println(err);
 while (1);
}

// Establishes bluetooth connection with iOS app
void setupBluetooth() {
 // Initialization
 Serial.print(F("Initialising the Bluefruit LE module: "));

 if (debugresponse) {
   Serial.println("Start");
 }

 if (!ble.begin(VERBOSE_MODE)) {
   error(F("Couldn't find Bluefruit"));
 }
 Serial.println(F("OK!"));

 if (FACTORYRESET_ENABLE) {
   Serial.println(F("Performing a factory reset: "));
   if (!ble.factoryReset()) {
     error(F("Couldn't factory reset"));
   }
 }

 // Disable echo
 ble.echo(false);

 Serial.println("Requesting Bluefruit info:");
 ble.info();

 Serial.println(F("Please use Adafruit Bluefruit LE app to connect in UART mode"));
 Serial.println(F("Then Enter characters to send to Bluefruit"));
 Serial.println();

 ble.verbose(false);

 // Wait for connection
 while (!ble.isConnected()) {
   delay(500);
 }

 if (ble.isVersionAtLeast(MINIMUM_FIRMWARE_VERSION)) {
   Serial.println(F("Change LED activity to " MODE_LED_BEHAVIOUR));
   ble.sendCommandCheckOK("AT+HWModeLED=" MODE_LED_BEHAVIOUR);
 }
}

// Separates distance and angle data into two variables
void parseRecievedData(const std::string& input, float& firstFloat, float&
secondFloat) {
   std::istringstream iss(input);
   char delimiter;

   // Parse the floats
   if (iss >> firstFloat >> std::noskipws >> delimiter >> secondFloat >> std::ws &&
       delimiter == ';') {
       // Successfully parsed
       std::cout << "First Float: " << firstFloat << ", Second Float: " << secondFloat
<< std::endl;
   } else {
       // Failed to parse
       std::cerr << "Invalid input format: " << input << std::endl;
   }
}

// Function to get data from app
bool getData(char buffer[], uint8_t maxSize) {
 TimeoutTimer timeout(100);

 memset(buffer, 0, maxSize);
 while ((!Serial.available()) && !timeout.expired()) {
   delay(1);
 }

 if (timeout.expired()) return false;

 delay(2);
 uint8_t count = 0;
 do {
   count += Serial.readBytes(buffer + count, maxSize);
   delay(2);
 } while ((count < maxSize) && (Serial.available()));

 return true;
}

// Recieves distanct and angle data from the iOS app over bluetooth
float recieveData() {
 // Check for user input
 char inputs[BUFSIZE + 1];

 if (getData(inputs, BUFSIZE)) {
   // Send characters to Bluefruit
   Serial.print("[Send] ");
   Serial.println(inputs);

   ble.print("AT+BLEUARTTX=");
   ble.println(inputs);

   // Check response status
   if (!ble.waitForOK()) {
     Serial.println(F("Failed to send?"));
   }
 }

 // Check for incoming characters from Bluefruit
 ble.println("AT+BLEUARTRX");
 ble.readline();
 if (strcmp(ble.buffer, "OK") == 0) {
   // no data
   return;
 }

 // Data was found
 Serial.print(F("[Recv] "));
 Serial.println(ble.buffer);
 ble.waitForOK();

 // Parse through recieved string to get distance and angle
 parseFloatString(input, distance, angle);
```

```
  // If the recieved data is null or 0
 return 0;
}

// Set motor speeds
void setMotorSpeed(float left, float right) {
 vescML.setDuty(left / 100);
 vescMR.setDuty(right / 100);
}

// Motor debug information
void motordebug() {
 if (debugresponse) {
   if (vescML.getVescValues()) {
     Serial.print("Left RPM: ");
     Serial.print(vescML.data.rpm);
     Serial.print(" | Tachometer: ");
     Serial.println(vescML.data.tachometerAbs);
   } else {
     Serial.println("Left Data Failed!");
   }

   if (vescMR.getVescValues()) {
     Serial.print("Right RPM: ");
     Serial.print(vescMR.data.rpm);
     Serial.print(" | Tachometer: ");
     Serial.println(vescMR.data.tachometerAbs);
   } else {
     Serial.println("Right Data Failed!");
   }
 } else {
   // Do nothing if debug is disabled
 }
}

// Get intial angle of walker from IMU
float getCurrentAngle() {
 // Read accelerometer data
 mpu.readSensor();

 // Calculate the angle using arctan
 float angle = atan2(mpu.getAccelY(), mpu.getAccelX()) * RAD_TO_DEG;

 return angle;
}

// Turn walker to UWB beacon
void turnToAngle(float targetAngle) {
   const float rotationSpeed = 15.0;
   const float tolerance = 2.0;

   // Calculate the initial angle
   float initialAngle = getCurrentAngle();

   // Calculate the angle difference
   float angleDifference = targetAngle - initialAngle;

   // Ensure the angle difference is within the range [-180, 180]
   if (angleDifference > 180.0) {
       angleDifference -= 360.0;
   } else if (angleDifference < -180.0) {
       angleDifference += 360.0;
   }
   // Set the motor speeds to turn the walker
   while (std::abs(angleDifference) > tolerance) {
       setMotorSpeed(-rotationSpeed * angleDifference / 180.0, rotationSpeed *
angleDifference / 180.0);

       // Update the angle difference
       initialAngle = getCurrentAngle();
       angleDifference = targetAngle - initialAngle;

       // Ensure the angle difference is within the range [-180, 180]
       if (angleDifference > 180.0) {
           angleDifference -= 360.0;
       } else if (angleDifference < -180.0) {
           angleDifference += 360.0;
       }
   }

   // Stop the motors after reaching the target angle
   setMotorSpeed(0.0, 0.0);
}

// Move the walker a certain distance in inches
void moveInches(float distance, float speed) {
 float targetRotations = (distance / (3.1415 * wheelDiameter)) * gearRatio;

 while (getMotorRPM(vescML) < targetRotations) {
   setMotorSpeed(speed, speed);
 }
}

void setup() {
 // Establishes bluetooth connection
 setupBluetooth();

 // Initialize MPU-6050
 while (!mpu.begin(MPU6050_SCALE_2000DPS, MPU6050_RANGE_2G)) {
   Serial.println("Could not find a valid MPU6050 sensor, check wiring!");
   delay(500);
 }
}

void loop() {
 // Distance and angle was successfully recieved
 if (recieveData() != 0) {
   // Turn to angle
   turnToAngle(angle);
 }
}
```

This next snippet of code checks if there is a direct path to the user using reading from two infrared sensors and one ultrasonic sensor on the walker. If there is a direct path, the walker moves forward to the wearable device. If not, the walker stops. Future implementation of LiDAR sensing and the A* navigation algorithm would be inserted in place of stopping.

```
// Updates variable with ultrasonic distance reading
void loopUltrasonic() {
 // Clears the trigPin condition
 digitalWrite(UltrasonicTrig, LOW);
 delayMicroseconds(2);

 // Sets the trigPin HIGH (ACTIVE) for 10 microseconds
 digitalWrite(UltrasonicTrig, HIGH);
 delayMicroseconds(10);
 digitalWrite(UltrasonicTrig, LOW);

 // Reads the echoPin, returns the sound wave travel time in microseconds
 long duration = pulseIn(UltrasonicEcho, HIGH); // variable for the duration of sound wave travel

 // Calculating the distance
 distanceUltrasonic = duration * 0.034 / 2; // Speed of sound wave divided by 2 (go and back)


 if ( debugresponse ) {
   Serial.print("Ultrasonic distance: ");
   Serial.println(distanceUltrasonic);
 }


}


// Updates variable with IR distance reading
void loopIR() {
 float voltsL = analogRead(LeftInfrared) * 0.0048828125;  // value from sensor * (5/1024)
 distanceIRLeft = 13 * pow(voltsL, -1);


 float voltsR = analogRead(RightInfrared * 0.0048828125;  // value from sensor * (5/1024)
 distanceIRRight = 13 * pow(voltsR, -1);


 if ( debugresponse ) {
   Serial.print ( "IR distance left: " );
   Serial.print ( distanceIRLeft );
   Serial.print ( "IR distance right: " );
   Serial.println ( distanceIRRight );
 }
}


// Detects if there is a direct path without obstacles to the wearable
// YES -> drives straight
// NO -> LiDAR scanning and A* navigation algorithm implementation
bool isDirectPath(float targetDistance) {
 // Read ultrasonic sensor data
 loopUltrasonic();


 // Read IR sensor data
 loopIR();


 // Threshold distance for obstacle detection
 float obstacleThreshUltrasonic = 10.0; // inches
 float obstacleThreshIR = 10.0; // inches


 // Check if there are obstacles detected by either sensor
 if ((distanceUltrasonic < (targetDistance - obstacleThreshUltrasonic)) ||
(distanceIRLeft < (targetDistance - obstacleThresholdIR) || (distanceIRRight <
(targetDistance - obstacleThresholdIR))) {
   return false; // Obstacles detected, no direct path
 } else {
   return true;  // No obstacles detected, direct path exists
 }
}

void setup() {
 // Establishes bluetooth connection
 setupBluetooth();

 // Initialize MPU-6050
 while (!mpu.begin(MPU6050_SCALE_2000DPS, MPU6050_RANGE_2G)) {
   Serial.println("Could not find a valid MPU6050 sensor, check wiring!");
   delay(500);
 }
}

void loop() {
 // Distance and angle was successfully received
 if (receiveData() != 0) {
   // Turn to angle
   turnToAngle(angle);
   delay(3000);

   // There is a straight path
   if (isDirectPath(distance) == true) {
     // Move forward to wearable at 15% speed
     moveInches(distance - 3, 15);
   // Stop, insert future LiDAR implementation here
   } else {
     moveInches(0, 0);
   }
 }
}
```

The next bit of code carefully detects whether the walker is within a reachable range of the user, slowly moving forward until 3 feet away. After being within this threshold, it then stops.

```
// Walker inches forward until within 3 feet of the user
void slowlyMoveToUser() {
 loopUltrasonic();

 // Continue moving forward until the walker is within reach using Ultrasonic sensor
 while (distanceUltrasonic > 3) {
   setMotorSpeed(7, 7); // Moves at 7% speed

   delay(100);
 }

 // Stop walker
 setMotorSpeed(0, 0);
}

void setup() {
 // Establishes bluetooth connection
 setupBluetooth();

 // Initialize MPU-6050
 while (!mpu.begin(MPU6050_SCALE_2000DPS, MPU6050_RANGE_2G)) {
   Serial.println("Could not find a valid MPU6050 sensor, check wiring!");
   delay(500);
 }
}

void loop() {
 // Distance and angle was successfully received
 if (receiveData() != 0) {
   // Turn to angle
   turnToAngle(angle);
   delay(3000);

   // There is a straight path
   if (isDirectPath(distance) == true) {
     // Move forward to wearable at 15% speed
     moveInches(distance - 3, 15); // Threshold of 3 feet until slowlyMoveToUser
function

     // Slowly move forward until walker is within a reachable distance of the user
     slowlyMoveToUser();

   // Stop, insert future LiDAR implementation here
   } else {
     moveInches(0, 0);
   }
 }
}
```