CSE-593
Project Report

Submitted to
Professor Shubham Jain

Submitted by
Siri Siddalingappa Naganoor (115326282)

1. INTRODUCTION - SENSOR STUDY ON VELOCITY DETERMINATION IN AUTISTIC PATIENTS

2. SYSTEM ANALYSIS AND DESIGN

2.1 Software Requirements

2.1.1 Arduino IDE: A text editor, a message area, a text console, a toolbar with buttons for common functions, and a series of menus are all included in the Arduino Integrated Development Environment, also known as the Arduino Software (IDE). It establishes a connection to the Arduino hardware in order to communicate with it and upload programs to it.

2.1.2 Libraries:

a. VL53L4CD: Arduino library to support the VL53L4CD Time-of-Flight ranging sensor**.**

Link: https://github.com/stm32duino/VL53L4CD

b. ESP32: Arduino library for the ESP32, ESP32-S2, ESP32-S3 and ESP32-C3

Link:https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

2.1.3 Plugin:

a. ArduSpreadsheet: A plug-in for the Arduino Integrated Development Environment (IDE) that saves serial data as a CSV file that can be opened in any spreadsheet application, such as Microsoft Excel or LibreOffice.

2.1.4 Application:

The KBeacon bluetooth device's settings were configured using the KBeacon application.

Key Features:

- Scans MajorID and MinorID data from iBeacon devices;
- Scan Eddystone devices for information like UID/TLM/URL;
- Examining data from KSensor devices like temperature and accelerometer;
- Configure iBeacon parameters like UUID and MajorID;
- Configure the Eddystone device's parameters

2.2 Hardware Requirements

2.2.1 Ultrasonic or Proximity Sensor: The Ultrasonic Sensor HC-SR04 is a device that can determine a distance between two points. It sends out an ultrasound at a frequency of 40 000 Hz (40 kHz), which travels through the air and is reflected back to the module if there is anything in its path, such as an object or an obstacle. You can determine the distance by taking into account both the amount of time it takes and the rate at which sound travels.

VCC (1), TRIG (2), ECHO (3), and GND (4) are the pins that make up the HC-SR04 configuration. VCC receives a supply voltage of +5V, and you can connect the TRIG and ECHO pins to any Digital I/O on your Arduino Board

To generate the ultrasound, we must set the Trigger Pin to High for 10 seconds. This will send an 8-cycle sonic burst that will travel at the speed of sound and be received by the Echo Pin. The Echo Pin will output the time the sound wave traveled in microseconds.

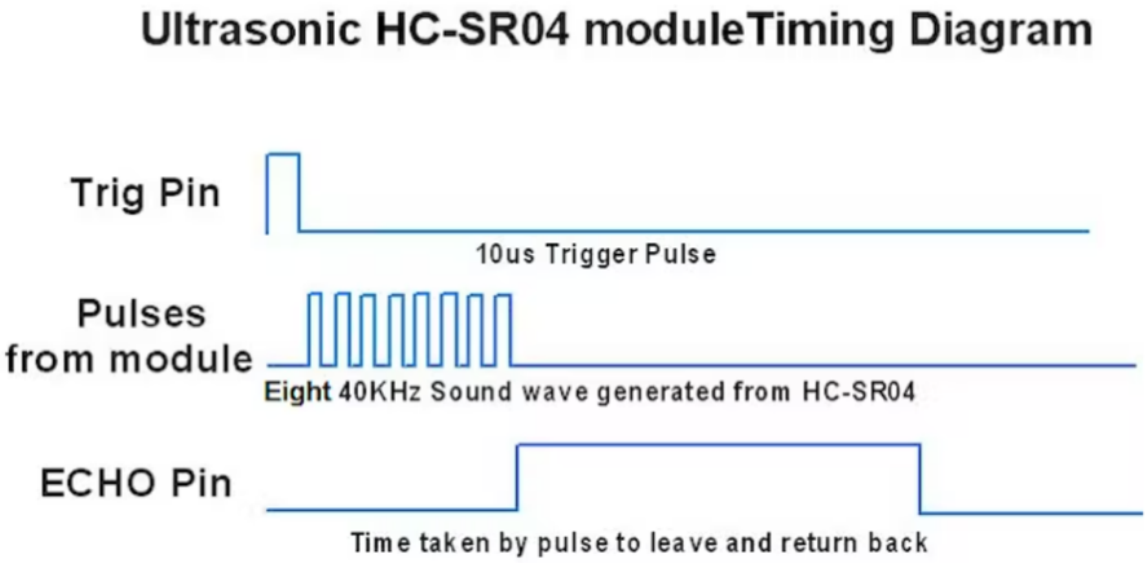


Fig 1. Ultrasonic HC-SR04 timing diagram

2.2.2 VL53L4CD Time-of-Flight ranging sensor: The VL53L4CD, which is specifically made for proximity and close-range measurements, offers incredibly precise distance measurements from as little as 1 mm up to 1300 mm. With a ranging speed of up to 100 Hz, a new generation of laser emitters with an 18° FoV improves performance in ambient light.

The VL53L4CD is perfect for use in battery-powered devices because of its extremely low power consumption and autonomous mode with programmable distance threshold. Since less potent and less expensive microcontrollers can be used, its fully embedded on-chip processing helps to reduce design complexity and BOM cost. The VL53L4CD records an absolute distance measurement regardless of the target color and reflectance, like all Time-of-Flight (ToF) sensors based on ST's FlightSense technology.

A VCSEL is built into every ToF sensor made by ST (vertical cavity surface emitting laser). It emits a 940 nm IR light that is completely safe for eyes and completely invisible (Class 1 certification).

2.2.3 Arduino Uno: The Arduino Uno is an open-source microcontroller board created by Arduino.cc and first made available in 2010. It is based on the Microchip ATmega328P microcontroller. Sets of digital and analog input/output (I/O) pins are present on the board, allowing it to be interfaced with different expansion boards (shields) and other circuits. The board has 6 analog I/O pins and 14 digital I/O pins, six of which can be used for PWM output. It can be programmed using the Arduino IDE (Integrated Development Environment) using a type B USB cable. It accepts voltages between 7 and 20 volts, but it can also be powered by an external 9-volt battery or by a USB cable.

2.2.4 ESP32: Espressif Systems created the ESP32, a single 2.4 GHz Wi-Fi and Bluetooth SoC (System On a Chip).

The ESP32 is made for Internet-of-Things (IoT) and wearable electronics applications. It has all the cutting-edge attributes of low-power chips, such as dynamic power scaling, multiple power modes, and fine-grained clock gating. For instance, ESP32 is only awakened periodically and when a specific condition is detected in a low-power IoT sensor hub application scenario. To reduce the amount of energy the chip uses, low-duty cycles are used.The power amplifier's output can also be changed, which aids in finding the best balance between communication range, data rate, and power usage. There are chips and modules available for the ESP32 series.

2.2.5 BC011 iBeacon BLE 4.0/5.0 MultiBeacon:

Broadcasts the Eddystone TLM, Eddystone URL, iBeacon, or Eddystone UID format of your choice. Every tenth transmission, Eddystone TLM transmits data on the battery's capacity, the temperature, and the number of broadcasts.

Free apps allow for customization of the beacon name, UUID, TX broadcast strength, broadcast interval, major/minor, etc.

3. IMPLEMENTATION

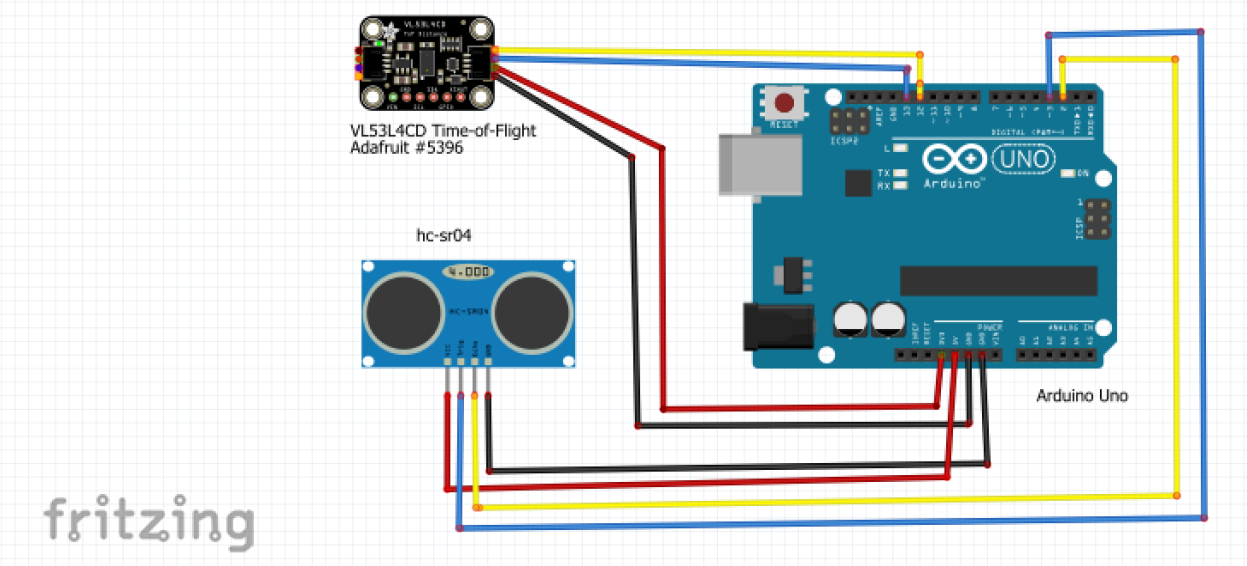


Fig 2. Hardware Implementation of the project with connections

3.1 System Implementation for Time-of-Flight, Ultrasonic Sensors with Arduino

1. **Install Arduino Ide** from the open source website arduino.cc

2. **Find the board** in tools and select the respective port, on connecting the Arduino Uno

3. **Connect** as shown in Fig 2.

4. **Install** the VL53L4CD library (Link: https://github.com/stm32duino/VL53L4CD)

5. **Upload** the code as shown in Appendix 1.

6. **Open the ArduSpreadsheet** from Tools and export the data (distance measurements from Time-of-flight and Ultrasonic sensor) collected to the .csv file to run post processing code.

3.2 System Implementation for ESP32 and iBeacon

1. Install the driver setup for ESP32

   Ref:https://espressif-docs.readthedocs-hosted.com/projects/arduino-esp32/en/latest/installing.html ,

https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/

2. After selection of the right board and port, upload the code found in Appendix 2.
3. Turn on the iBeacon by pressing the button until it starts to blink

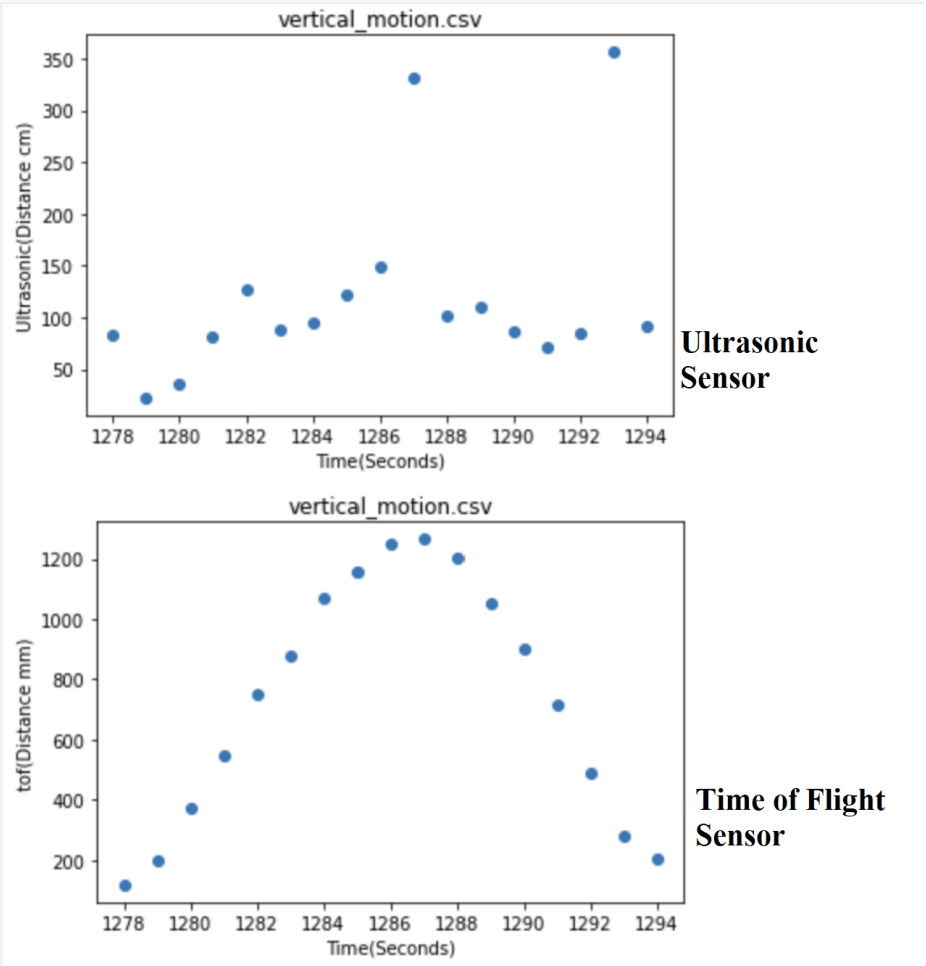
   Ref: https://bluecharmbeacons.com/bc011-ibeacon-multibeacon-quick-start-guide/

4. Ensure to calibrate the value of N(Environmental Factor) and update the "Distance" formula in the code accordingly
5. Once that is done, open ArduSpeadsheet and record the scan results of RSSI and Distance into a .csv file to run further data processing.

4. EVALUATION AND TESTING

4.1 Ultrasonic, Time-of-Flight Sensors and Arduino Uno Setup Results

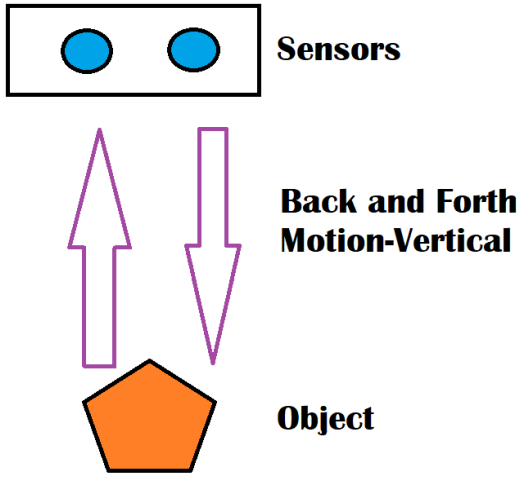
4.1.1 Vertical motion Test



Ultrasonic Sensor

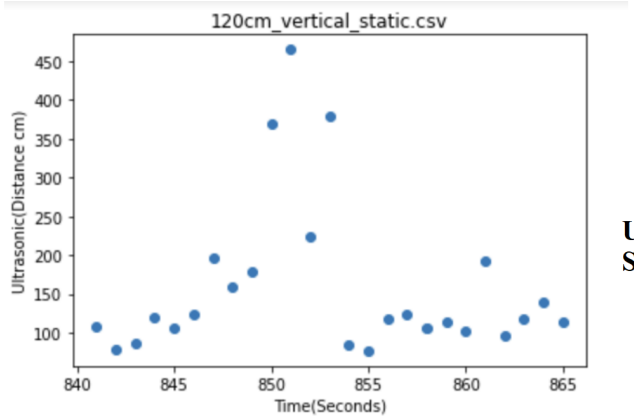
Time of Flight Sensor

Motion Representation:

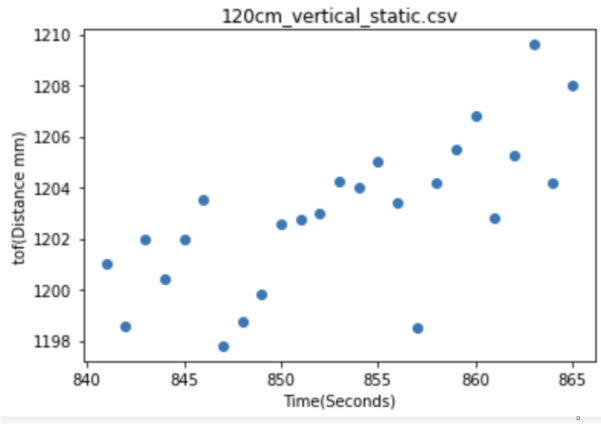


Inference: The TOF sensor represents a smoother curve with approximately accurate readings of the vertical motion of the object when compared to Ultrasonic sensor

4.1.2 Vertical Static at 120cm

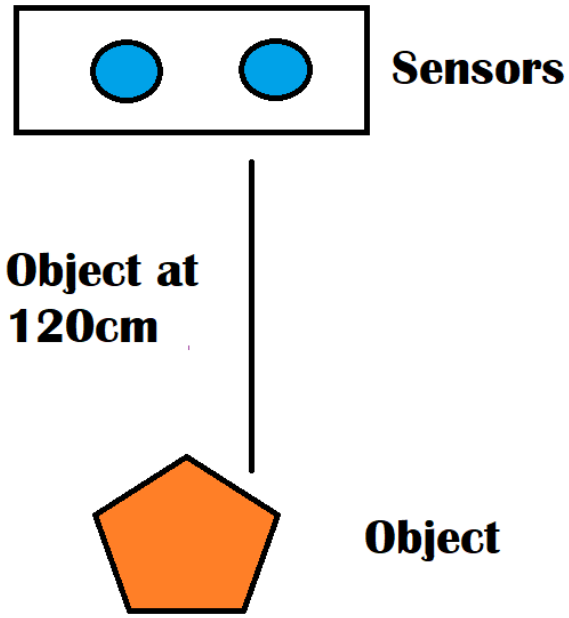


Ultrasonic Sensor



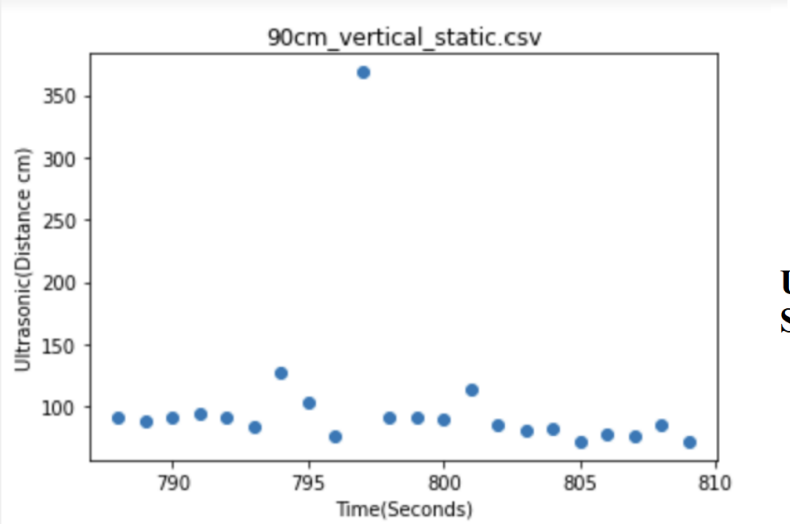
Time of Flight Sensor

Representation:

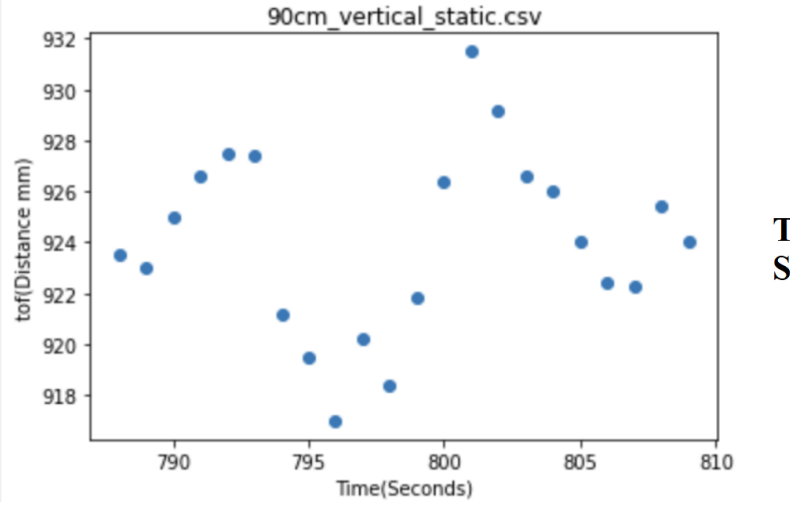


Inference: The majority of data points for the Time of Flight sensor lie between 119-121cm and 100-125 for the Ultrasonic sensor from the graph.

4.1.3 Vertical static at 90cm

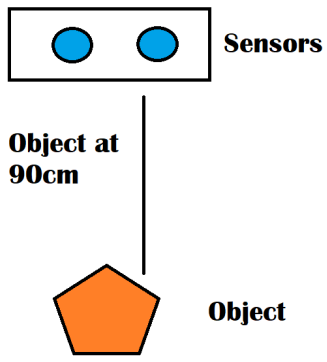


Ultrasonic Sensor



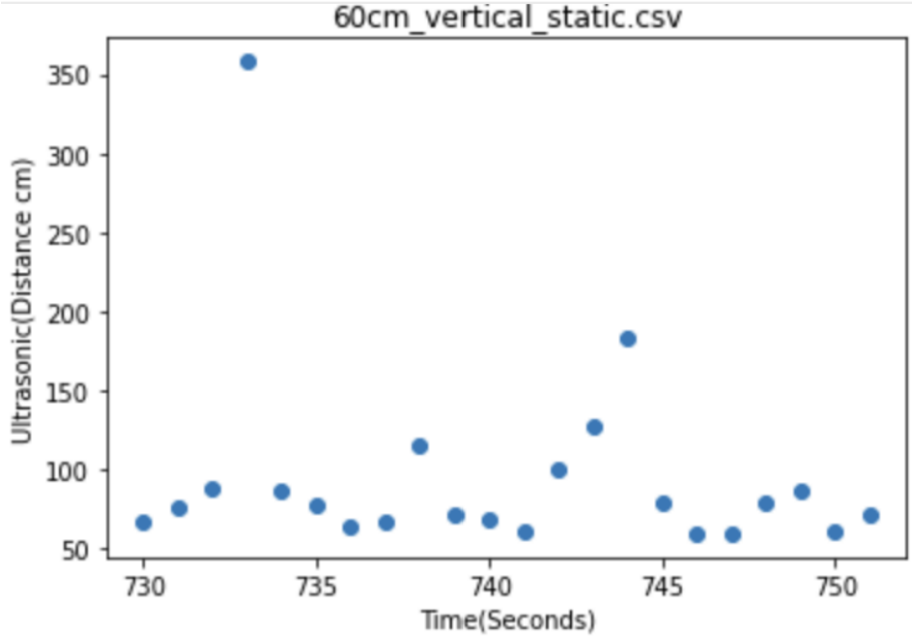
Time of Flight Sensor

Representation:

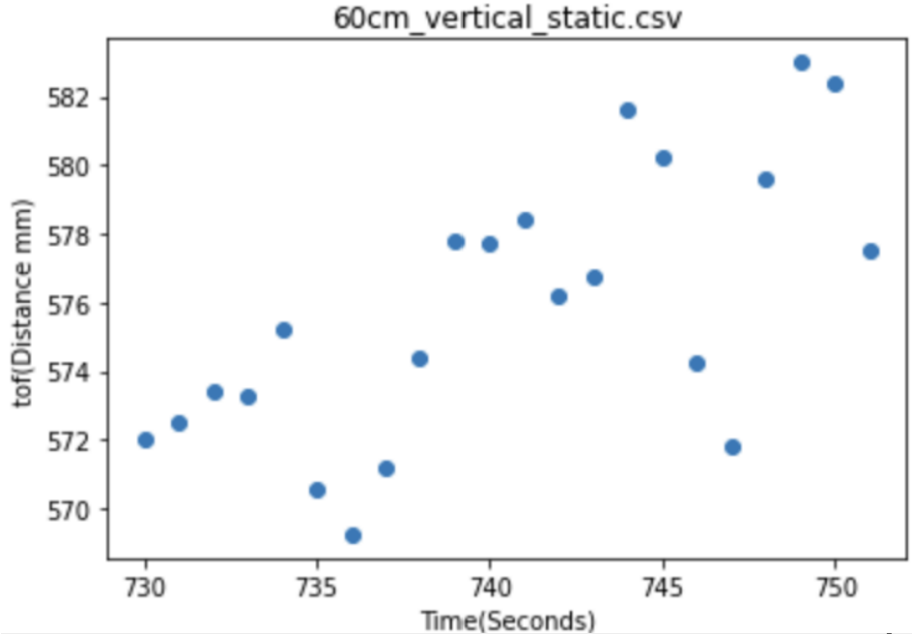


Inference: The majority of data points for the Time of Flight sensor lie between 90-94cm and 90-100 for the Ultrasonic sensor from the graph.

4.1.4 Vertical static at 60cm

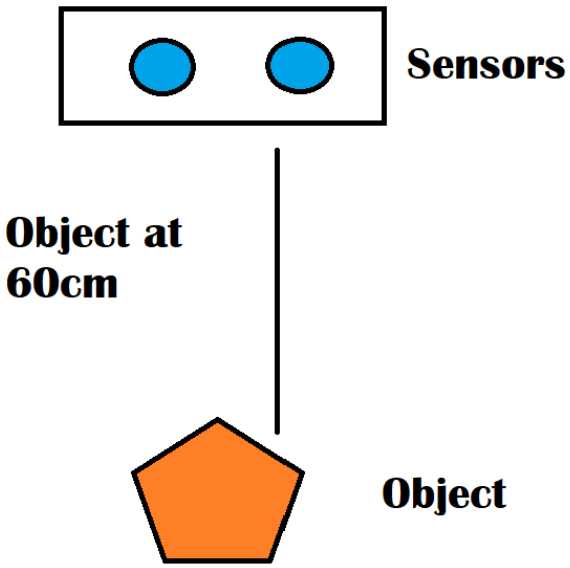


Ultrasonic Sensor



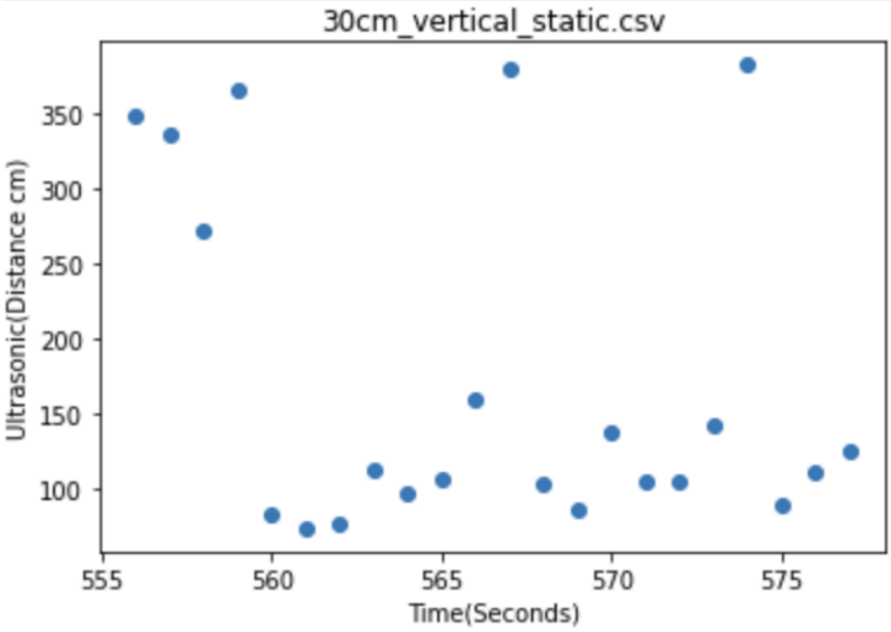
Time of Flight Sensor

Representation:

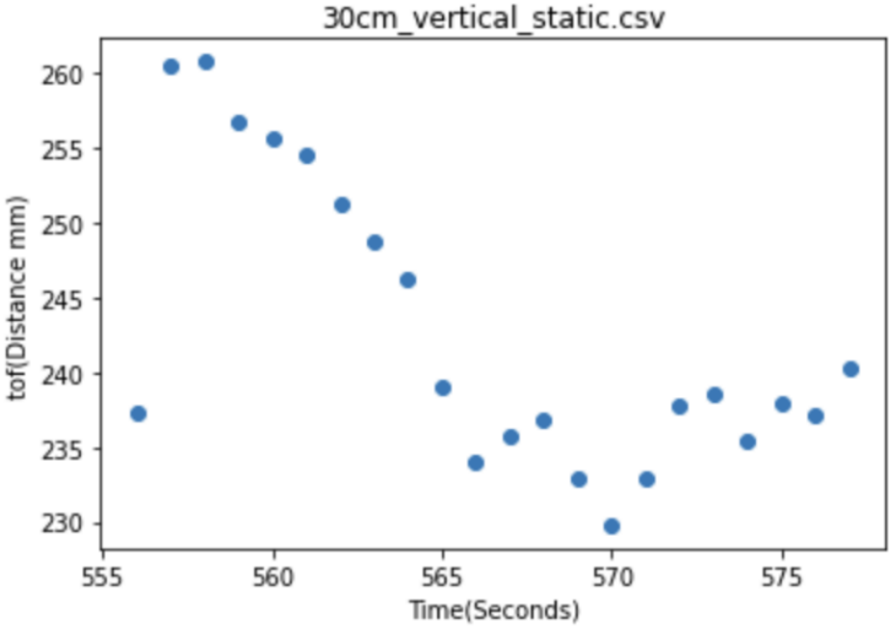


Inference: The majority of data points for the Time of Flight sensor lie between 57-60cm and 50-80cm for the Ultrasonic sensor from the graph.

4.1.5 Vertical static at 30cm

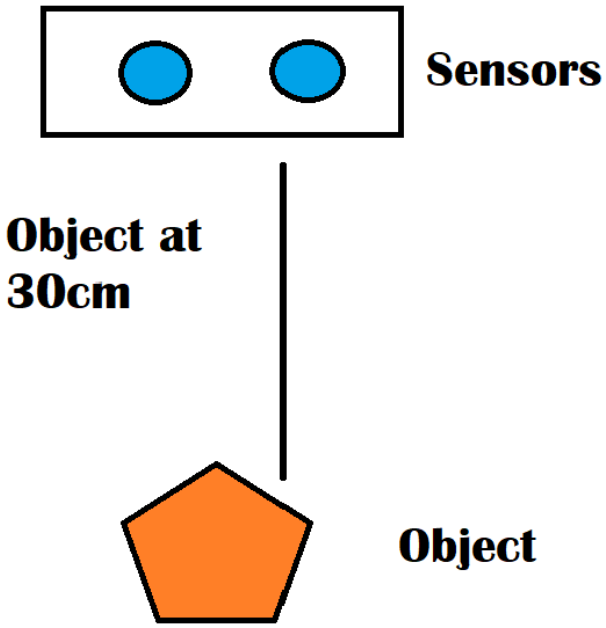


Ultrasonic Sensor



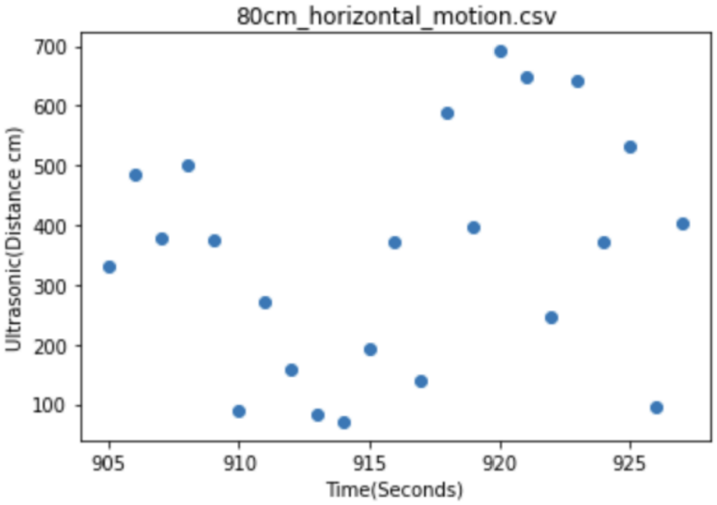
Time of Flight Sensor

Representation:

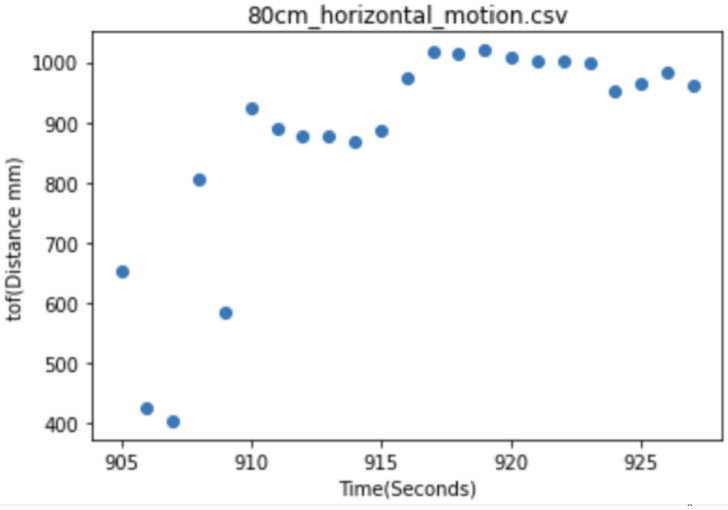


Inference: The majority of data points for the Time of Flight sensor lie between 23-26 cm and 50-150 cm for the Ultrasonic sensor from the graph.

4.1.6 Horizontal motion at 80cm

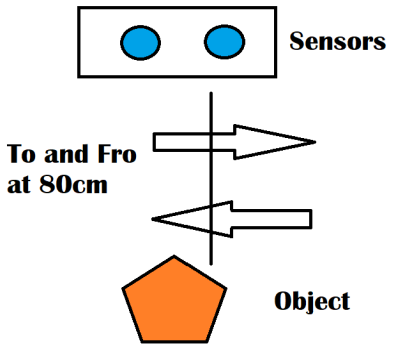


Ultrasonic Sensor



Time of Flight Sensor

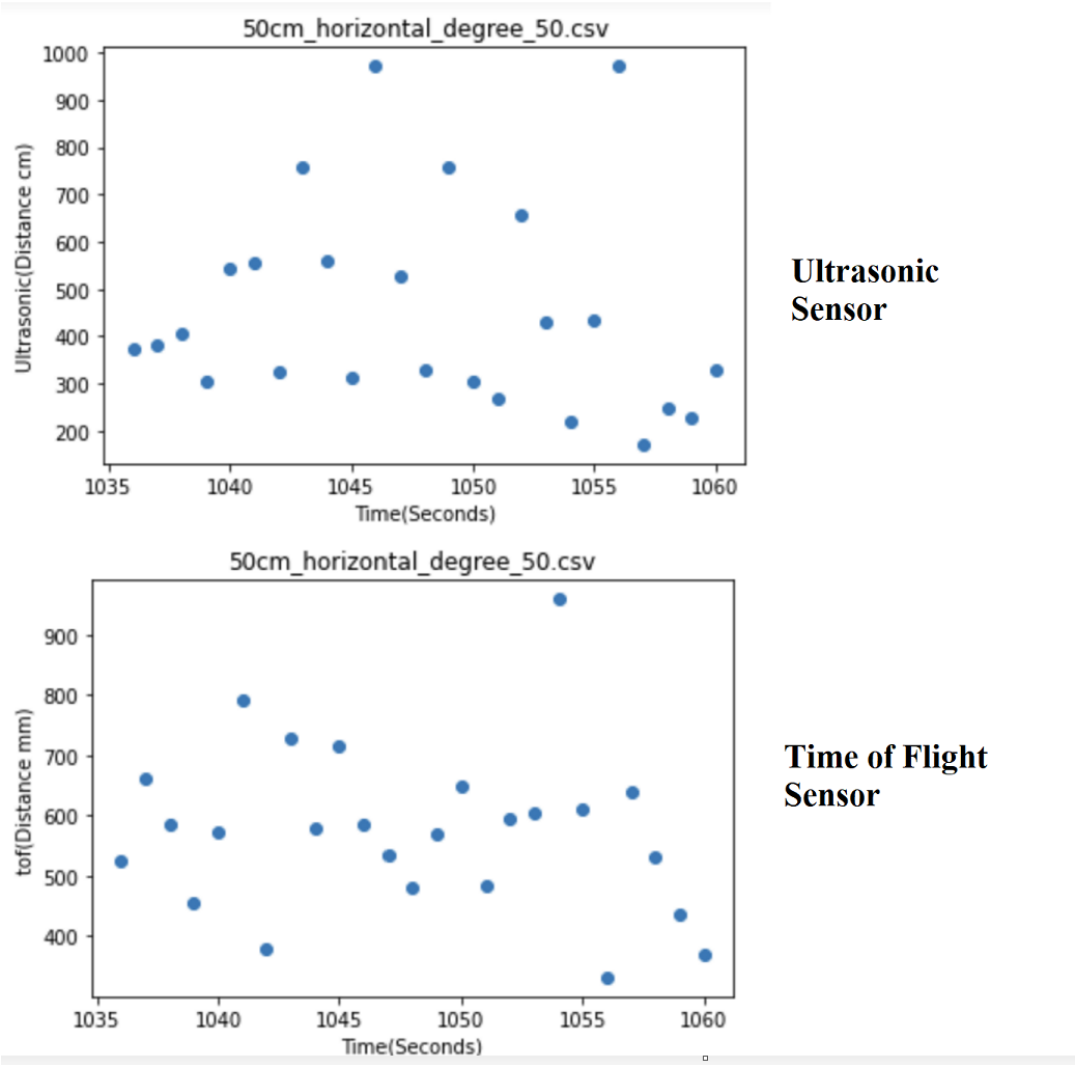
Representation:



Inference: The majority of data points for the Time of Flight sensor lie between 60-100 cm and >100 cm for the Ultrasonic sensor from the graph.
When the object is not in a straight line with the sensor, both the sensor readings are abrupt, indicating they measure only when in a straight line and not at an angle

The readings are slightly better for the TOF sensor in comparison to the Ultrasonic sensor.

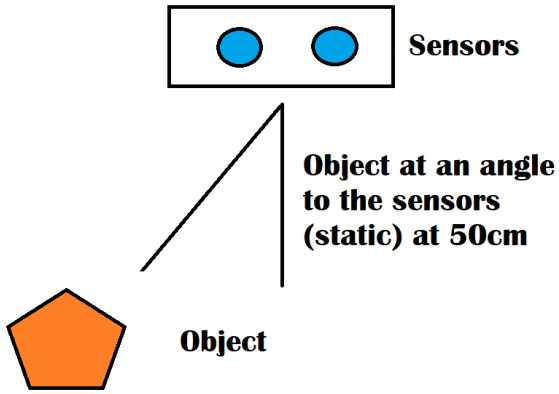
4.1.7 Angle at 50cm (Static)



Ultrasonic Sensor

Time of Flight Sensor

Representation:

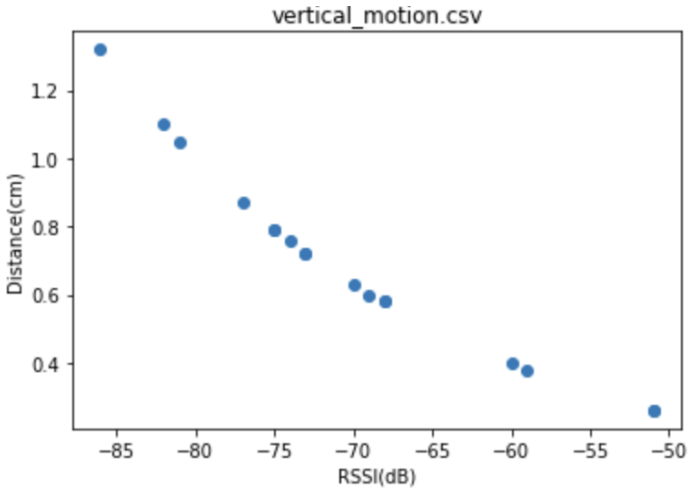


Inference: The majority of data points for the Time of Flight sensor lie between 40-80 cm and >200 cm for the Ultrasonic sensor from the graph.
When the object is not in a straight line with the sensor, both the sensor readings are abrupt, indicating they measure only when in a straight line and not at an angle

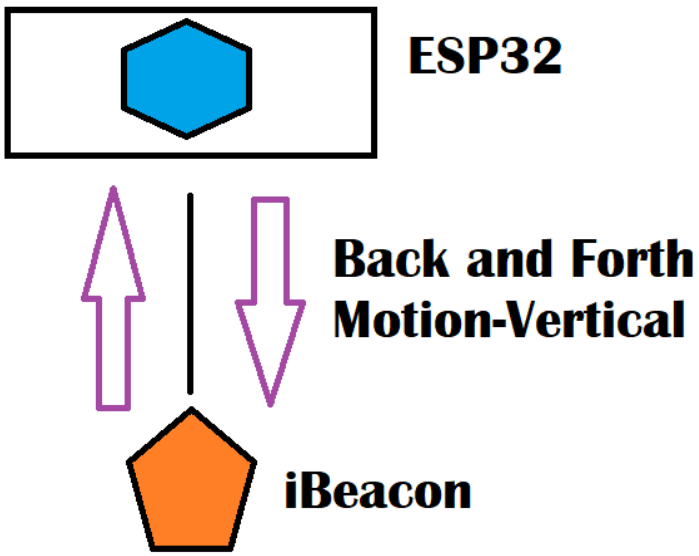
The readings are slightly better for the TOF sensor in comparison to the Ultrasonic sensor.

4.2 ESP32 and iBeacon Setup Results

4.2.1 Vertical motion Test

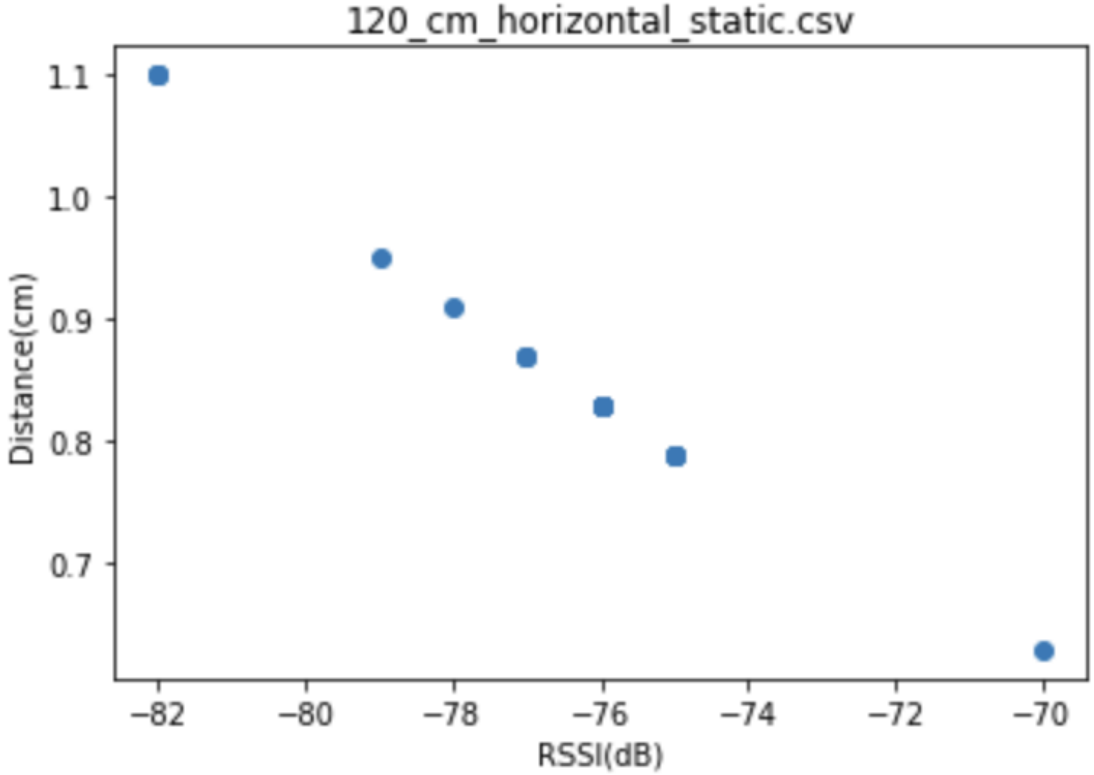


Representation:

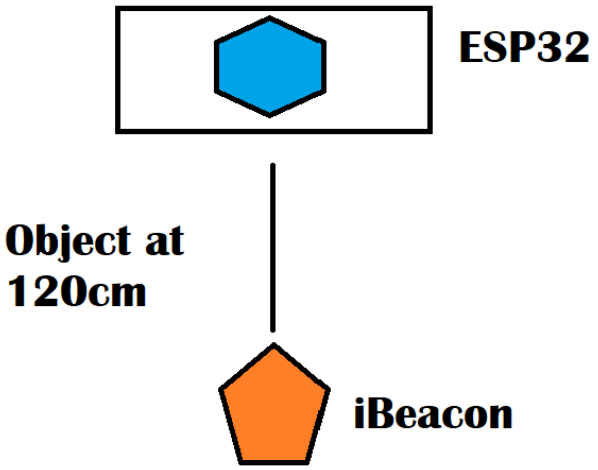


Inference: It detects the vertical motion(distance of beacon) close to accuracy of the measured points on ground.

4.2.2 Vertical Static at 120cm

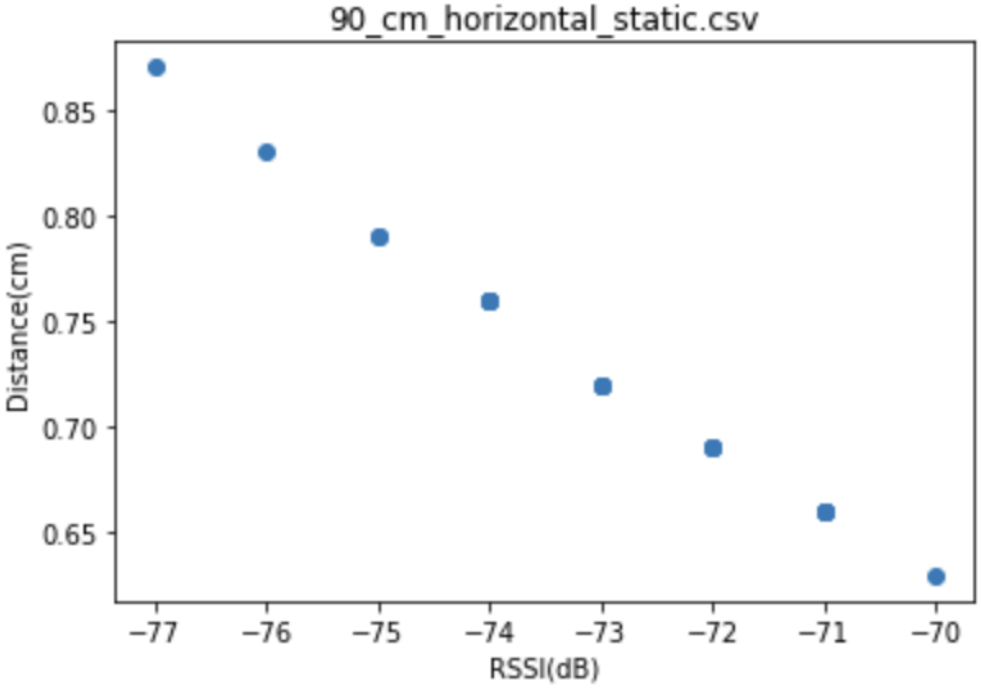


Reference:

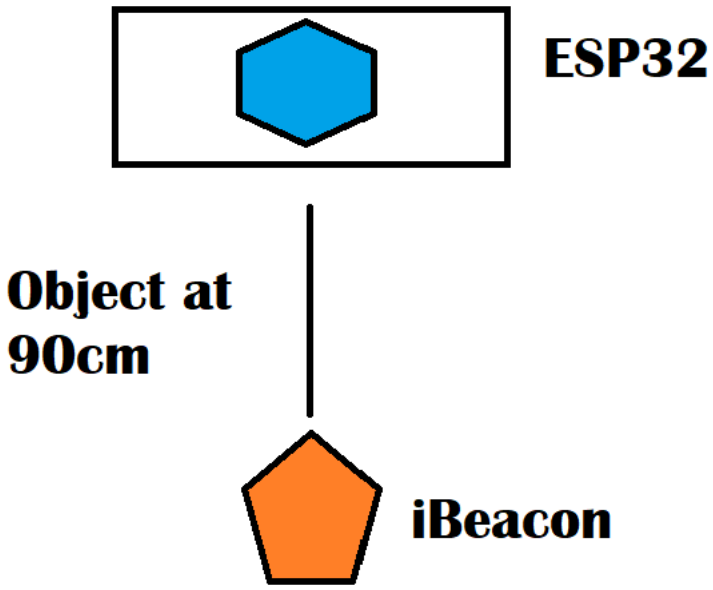


Inference: The ESP32 measures distance in the range of 80-110cm.

4.2.3 Vertical static at 90cm

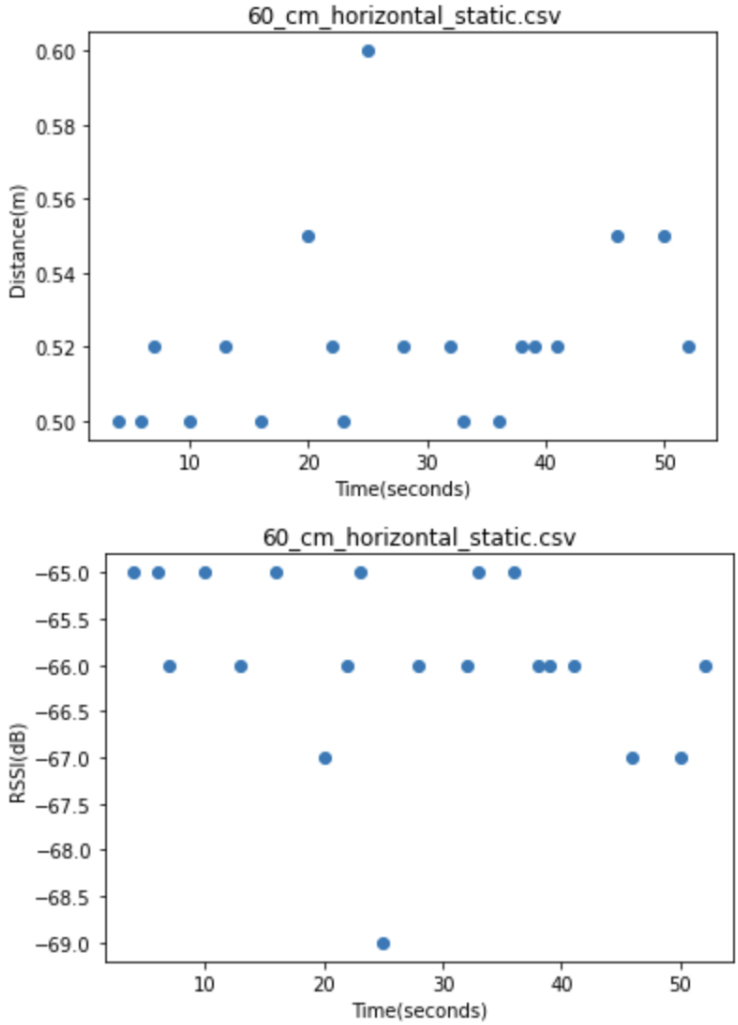


Representation:

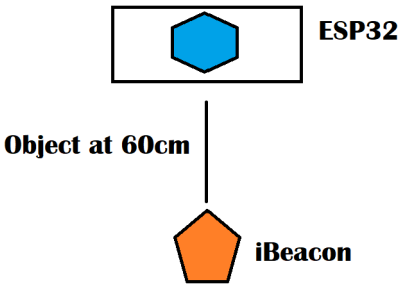


Inference:
The ESP32 measures distance in the range of 65-90cm.

4.2.4 Vertical static at 60cm

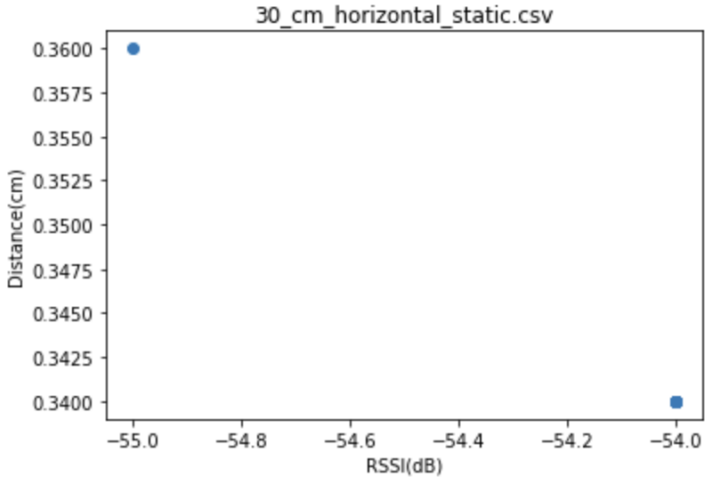


Representation:

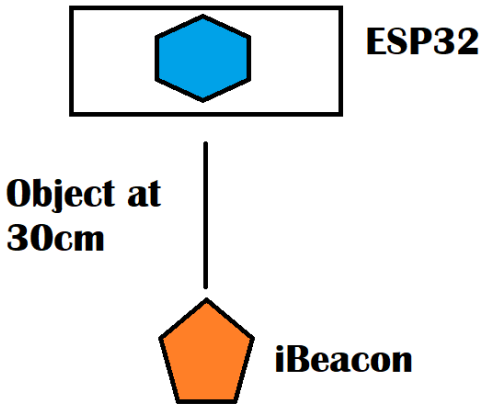


Inference: The ESP32 measures distance in the range of 50-60 cm and RSSI in the range (-68)-(-65) dB.

4.2.5 Vertical static at 30cm

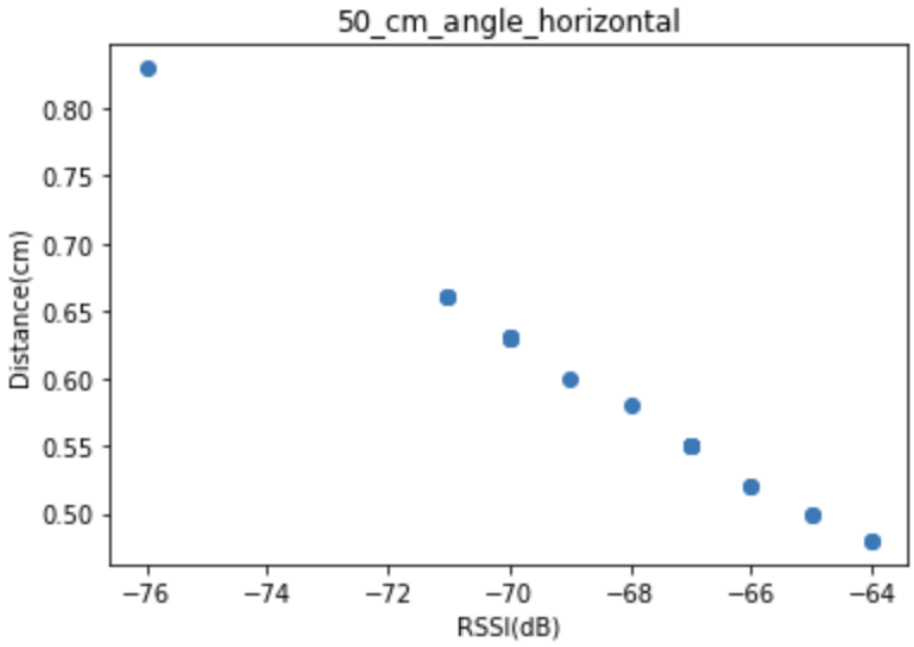


Representation:

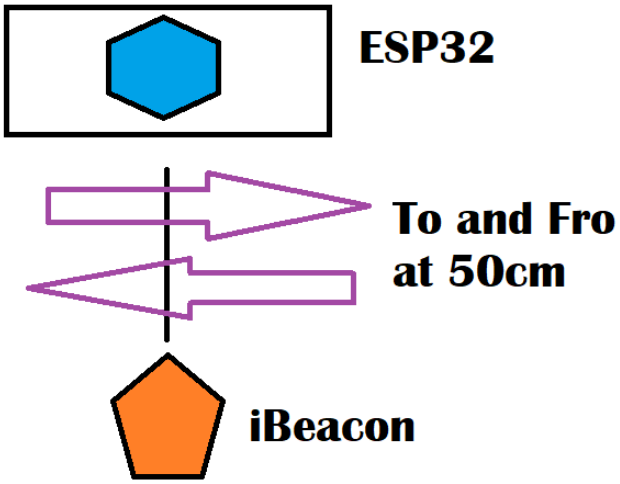


Inference: The ESP32 measures distance in the range of 34-36 cm.

4.2.6 Horizontal motion at 50cm



Representation:



Inference:
ESP32 measures distance independent of the angle between the iBeacon and itself. It measured distance in the range of 45-70cm

SUMMARY

Expected Distance(cm)	Ultrasonic Sensor	Time of Flight	ESP32
30cm	100-150 cm	23-26 cm	34-36 cm
60cm	50-80 cm	57-60 cm	50-60 cm
90cm	90-100 cm	90-94 cm	65-90 cm
120cm	100-125 cm	119-121 cm	80-110 cm

APPENDIX

1. **Ultrasonic, Time-of-Flight Sensor and Arduino IDE setup**

```
/* Includes ------------------------------------------------------------------*/
#include <Arduino.h>
#include <Wire.h>
#include <vl53l4cd_class.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <assert.h>
#include <stdlib.h>

#define DEV_I2C Wire
#define SerialPort Serial

#ifndef LED_BUILTIN
  #define LED_BUILTIN 13
#endif
#define LedPin LED_BUILTIN

// Components.
VL53L4CD sensor_vl53l4cd_sat(&DEV_I2C, A1);
```

```
/* Setup ---------------------------------------------------------------------*/
#define echoPin 2 // attach pin D2 Arduino to pin Echo of HC-SR04
#define trigPin 3 //attach pin D3 Arduino to pin Trig of HC-SR04

// defines variables
long duration; // variable for the duration of sound wave travel
int distance; // variable for the distance measurement
void setup() {
  // put your setup code here, to run once:
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an OUTPUT
  pinMode(echoPin, INPUT); // Sets the echoPin as an INPUT
  //Serial.begin(9600); // // Serial Communication is starting with 9600 of baudrate speed
  //Serial.println("Starting..."); // print some text in Serial Monitor

   // Led.
  pinMode(LedPin, OUTPUT);

  // Initialize serial for output.
  SerialPort.begin(115200);
  //SerialPort.println("Starting...");

  // Initialize I2C bus.
  DEV_I2C.begin();

  // Configure VL53L4CD satellite component.
  sensor_vl53l4cd_sat.begin();

  // Switch off VL53L4CD satellite component.
  sensor_vl53l4cd_sat.VL53L4CD_Off();

  //Initialize VL53L4CD satellite component.
  sensor_vl53l4cd_sat.InitSensor();

  // Program the highest possible TimingBudget, without enabling the
  // low power mode. This should give the best accuracy
  sensor_vl53l4cd_sat.VL53L4CD_SetRangeTiming(200, 0);

  // Start Measurements
  sensor_vl53l4cd_sat.VL53L4CD_StartRanging();
}
```

```
void TOF()
{
  uint8_t NewDataReady = 0;
  VL53L4CD_Result_t results;
  uint8_t status;
  char report[64];

  do {
    status = sensor_vl53l4cd_sat.VL53L4CD_CheckForDataReady(&NewDataReady);
  } while (!NewDataReady);

  //Led on
  digitalWrite(LedPin, HIGH);

  if ((!status) && (NewDataReady != 0)) {
    // (Mandatory) Clear HW interrupt to restart measurements
    sensor_vl53l4cd_sat.VL53L4CD_ClearInterrupt();

    // Read measured distance. RangeStatus = 0 means valid data
    sensor_vl53l4cd_sat.VL53L4CD_GetResult(&results);
    //snprintf(report, sizeof(report), "Status = %3u, Distance = %5u mm, Signal = %6u
kcps/spad\r\n",
            //results.range_status,
            //results.distance_mm,
            //results.signal_per_spad_kcps);
    //Serial.print("Distance(mm) from TOF: ");
    SerialPort.print(results.distance_mm);
    Serial.print("\t");
  }

  //Led off
  digitalWrite(LedPin, LOW);
}
void Ultrasonic() {
  delay(200);
  // Clears the trigPin condition
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin HIGH (ACTIVE) for 10 microseconds
  digitalWrite(trigPin, HIGH);
```

```
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // Reads the echoPin, returns the sound wave travel time in microseconds
  duration = pulseIn(echoPin, HIGH);
  // Calculating the distance
  distance = duration * 0.034 / 2; // Speed of sound wave divided by 2 (go and back)
  // Displays the distance on the Serial Monitor
  //Serial.print("Distance(cm) from Ultrasonic: ");
  Serial.println(distance);
  //Serial.println(",");

  //Serial.println(" cm");
}
void loop() {
  // put your main code here, to run repeatedly:
  TOF();
  Ultrasonic();

}
```

2. ESP32 and iBeacon Setup

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEScan.h>
#include <BLEAdvertisedDevice.h>

int scanTime = 30; //In seconds

class MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallbacks {

    void onResult(BLEAdvertisedDevice advertisedDevice) {
      //Serial.printf("Advertised Device: %s \n", advertisedDevice.toString().c_str());
      //BlueCharm_175540
      //N (Constant depends on the Environmental factor. Range 2-4)
      //Distance = 10 ^ ((Measured Power – RSSI)/(10 * N))
      //Measured Power(RSSI) at 1m = -72
      String DeviceName = advertisedDevice.getName().c_str();
      int rssi;
      //N varies from 2-4
```

```
      float Env_N= 5;
      int MeasuredPower = -80;
      float calculatedValue=0;
      float Distance = 0;
      if(DeviceName == "BlueCharm_178475"){
        //Serial.println("Found it");
        //Serial.printf("Advertised Device: %s \n", advertisedDevice.toString().c_str());
        rssi = advertisedDevice.getRSSI();
        calculatedValue = ((MeasuredPower + ((-1)*rssi)) / (10 * Env_N));
        //Serial.print("Rssi: ");
        //Serial.println(rssi);
        //Serial.print("CV: ");
        //Serial.println(calculatedValue);
        //Serial.print("Distance in M: ");
        Distance = pow(10,calculatedValue);
        Serial.print(rssi);
        Serial.print("\t");
        Serial.println(Distance);

        advertisedDevice.getScan()->stop();
      }

    }

};

void setup() {
  Serial.begin(115200);
  //Serial.println("Scanning...");

  BLEDevice::init("");
  BLEScan* pBLEScan = BLEDevice::getScan(); //create new scan
  pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());
  pBLEScan->setActiveScan(true); //active scan uses more power, but get results faster
  BLEScanResults foundDevices = pBLEScan->start(scanTime);
//  Serial.print("Devices found: ");
//  Serial.println(foundDevices.getCount());
//  Serial.println("Scan done!");
}
```

```
void loop() {
  // put your main code here, to run repeatedly:
  //Serial.println("Scanning...");

  BLEDevice::init("");
  BLEScan* pBLEScan = BLEDevice::getScan(); //create new scan
  pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());
  pBLEScan->setActiveScan(true); //active scan uses more power, but get results faster
  BLEScanResults foundDevices = pBLEScan->start(scanTime);
//  Serial.print("Devices found: ");
//  Serial.println(foundDevices.getCount());
//  Serial.println("Scan done!");
  pBLEScan->clearResults();
  delay(1000);
}
```