EEC172 Final Project: 2D Ultrasonic Sonar Scanner

Braedon Hansen,
Haaris Tahir-Kheli

June 9, 2025

1 Description

For the final project, we chose to build a 2-dimensional sonar scanner for mapping areas of a room or house. We chose to do this because all currently available products of the same nature are very expensive and not accessible to most consumers. It contains an ultrasonic distance sensor and 6-axis inertial measurement unit (IMU) for mapping surroundings, and a organic light emitting diode (OLED) display for showing the state of the device, powered by a Texas Instruments CC3200 LaunchPad development board. The data was sent to Amazon Web Services (AWS) for storage, where it could be downloaded for processing and visualization.

1.1 Website

Here is our EEC172 Final Project Webpage.

2 Design

Here is how we met each of the design requirements for the final project:

- 2 Hardware Communication Protocols: I^2C for the IMU, SPI for the OLED.
- Web Service: AWS IoT, Lambda, S3 Bucket.
- 2 Sensing Devices: IMU, Ultrasonic Distance Sensor

2.1 Functional Specification

The high-level functionality of the device begins with booting. During booting, an initial reading of the IMU is taken and used as a calibration value for the remainder of the loop. The OLED is initialized as well, loading to a blank screen.

Once booting is complete, the device goes into its `Polling` state, showing a 'P' on the OLED, collecting a data point from the IMU and distance sensor, and looping until it collects 100 samples. Then, it attempts to connects to AWS, skipping to the `Done` state if there is a failure. If AWS connects successfully, an 'S' is displayed and each point is sent to an AWS IoT Device Shadow, which triggers a Lambda function that appends a .CSV file in an S3 Bucket.

Finally, after all data has been sent, it enters a `Done` state, which waits for reset while displaying a 'D' on the OLED.

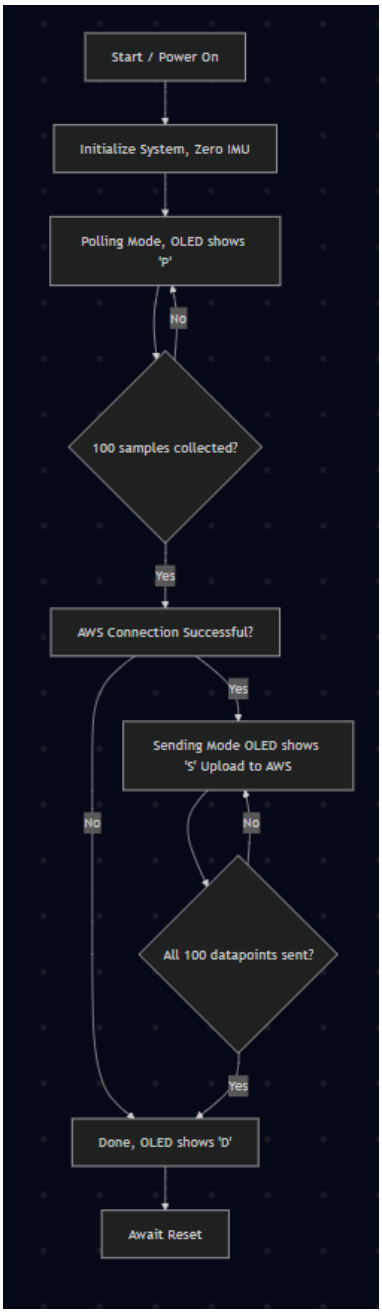

Figure 1: Functional Flowchart

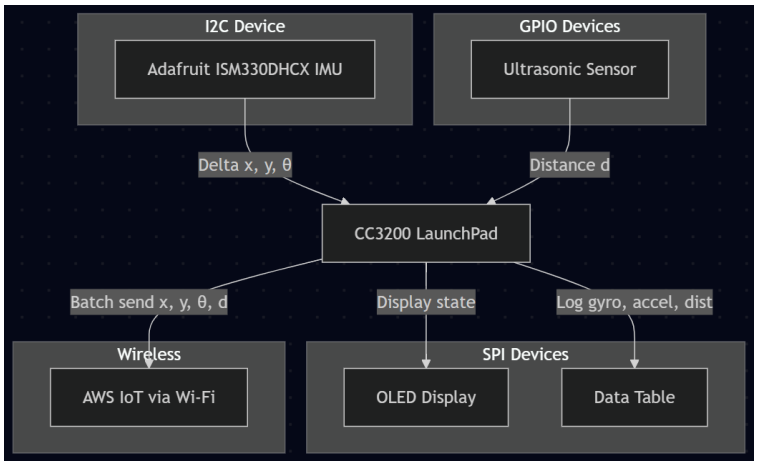

Figure 2: System Block Diagram

2.2 System Architecture

Pin	Peripheral Function	Description
PIN_15	GPIO Output	Trigger Pulse (Ultrasonic)
PIN_63	GPIO Input	Echo Input (Ultrasonic)
PIN_55	UART0 TX	UART0 Transmit (USB-Serial)
PIN_57	UART0 RX	UART0 Receive (USB-Serial)
PIN_01	I^2C SCL	I^2C Clock line for IMU
PIN_02	I^2C SDA	I^2C Data line for IMU
PIN_07	SPI MOSI	OLED SPI Data (MOSI)
PIN_05	SPI CLK	OLED SPI Clock (SCLK)
PIN_62	GPIO Output	OLED DC (Data/Command)
PIN_18	GPIO Output	OLED Reset (R)
PIN_61	GPIO Output	OLED Chip Select (CS)

As shown in the above table, the device uses 11 GPIO pins on the CC3200 LaunchPad. The ultrasonic distance sensor uses pins 15 & 63 for `Trigger` and `Echo`, respectively. The UART for debugging uses the standard 55 & 57. The IMU uses two pins for I^2C, with 1 and 2 as `Clock` and `Data`, respectively. Finally the OLED uses 5 pins for SPI, using 7 for `Master Out Slave In`, 5 for `Clock`, 62 for `Data/Command`, 18 for `Reset`, and 61 for `Chip Select`.

3 Implementation

3.1 Hardware

3.1.1 Sensors

- Distance Sensor
- IMU

The `Trigger` pin on the distance sensor is configured using an interrupt handler on a peripheral clock. This is run every 0.05 seconds, setting a frequency of 20Hz. When this is triggered, it pulls the `Echo` pin high. The signal remains high until an echo is received, at which point, the signal is dropped. Another peripheral clock is used to time that difference, with an interrupt set to trigger at both edges. In this interrupt, the falling edge is used to trigger a reading. The distance is derived using the speed of sound and the time of travel. Then an IMU value is read and both data points are saved together.

At a high level, one timer sets a trigger speed, and the reciept of the return signal triggers the function to save the data from both sensors.

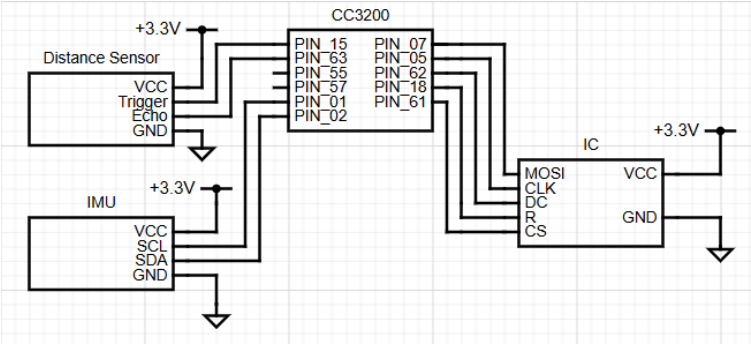

Figure 3: Circuit Diagram

3.1.2 Actuators

- OLED Display

The OLED display is used to show the user the state of the device while it is in use. It simply receives a signal at the beginning of each state loop telling it to display a character until a new one is sent. This is important because it tells the user when the device is recording data and when it is done and the data is prepare.

3.2 Software

- Code Composer Studio (CCS) C code
- AWS
- Python
- MATLAB

The software system starts at the CC3200, where all the data is recorded, according to the hardware explanation above. Once the full data length is recorded, in a C struct string, each data point is parsed into a JSON message which is then sent to an AWS IoT Core Device Shadow. Though it only remains in the shadow for a moment before it is replaced, a Lambda function is triggered at each update. This function first clears an S3 bucket for the new batch of data then adds creates a .CSV and adds the new data to the bottom of the file.

Once all 100 data points have occupied the .CSV, the data can be downloaded from S3 via a Python program that accesses the bucket directly. Then, a MATLAB program extracts the data from the .CSV before filtering, transforming, and plotting the data. For filtering, 4 main filters were used to reduce noise.

First, a basic outlier filter that removed all data points where the distance was measured as 50 inches was implemented. This is due to the fact that the there was no point during testing where the sensor was actually more than 50 inches away from a wall, meaning a reading of 50 inches or higher for distance was erroneous.

Next, a median filter, with a window size of 3, was used to smooth out potential outliers in the IMU data. The median filter went through all 7 vectors of data, examining consecutive groups of 3 elements, and selecting the median of them, to create 7 new vectors of filtered data.

After that, the data was run through a Madgwick filter. This is a sensor fusion algorithm specifically designed for IMU data. The Madgwick filter works by taking the actual Euler angles from the IMU, converting them into quaternions, estimating the current Euler angles (as a quaternion), and then performing gradient descent using the actual and estimate Euler angles to minimize the error of the drift that occurs as the parasitic noise of the IMU begins to affect the data.

Finally, the distance vector is run through an Enhanced Kalman filter. This filter attempts to create a state-space model to estimate independent white noise in the measurement system, and reduce the bias. It

is nonlinear, as opposed to a standard Kalman filter, and effectively attempts to linearize a model based on previous value–akin to training data for a machine learning algorithm, but on a smaller scale.

4 Challenges

The largest challenge we faced in the development of this project was parasitic noise from the IMU. Because the sensor is naturally slightly noisy, like any sensor, the signal is not a perfect representation of the real world. While this is generally not an issue for orientation estimation, it becomes a significant issue when converting to position data.

To calculate position, the acceleration data must be integrated twice and added to all previous values. This double integration causes the noise to be squared as well, and the summing operation compounds the noise further. This compounding noise makes it increasingly difficult to maintain an accurate position estimate after more than a few seconds.

To counteract this, a number of filtering methods were applied to the data. First, all values under $0.02g$ in linear acceleration were ignored. This was intended to be a very simple high-pass filter, which erased any noise while the device was stationary, while allowing for larger (human) movements to make it through. This error is compounded exponentially when deriving position from an accelerometer, due to the double integration calculation. Next, a feature to allow the data to be visualized without the (X,Y) data was implemented, so the data could be visualized strictly with the more accurate gyroscope measurements. This was helpful for measurements where the device was not moved as much, and most of the movement was rotational. Finally, 4 layers of filtering were applied to the data to further reduce noise: a basic outlier filter, a median filter of length 3, a Madgwick filter for accelerometer and gyroscopic data, and an Enhanced Kalman filter for the distance data.

5 Future Work

This project has plenty of room to grow. A 9 DoF IMU with higher precision would allow better measurements and more accurate filtering. This would permit much longer readings, potentially long enough for a whole room or a small 3-D scan. Building the device a better case which could hold the battery inside, rather than using a large external battery would also be a good improvement. The user interface could be improved on the device side by implementing a simpler reset protocol, allowing for multiple scans without downloading the data and resetting the device. Finally, if data length was improved, the device could include a MicroSD card to store larger datasets while polling, dramatically expanding the practical applications of the device.

6 Bill of Materials

—**Component**	**Price**	**Notes**
HC-SR04 Ultrasonic Sensor	$3.95	Already owned
Adafruit ISM330DHCX - 6 DoF IMU	$19.95	Purchased
Solid-Core Wire Spool - 25ft - 22AWG	$2.95	Already owned
Adafruit Perma-Proto PCB - Single	$4.50	Already owned
Total (purchased)	**$19.95**	

7 Division of Work

Haaris:

- Worked on implementing AWS integration with the device, by using a combination of IoT device shadows and a AWS Lambda function, to output data received from the device into a .csv file via a AWS s3 bucket.

- Wrote the code for the filters used in the MATLAB visualization code. A library was used for Madgwick Filter, but the Enhanced Kalman Filter code was written from scratch, to fine-tune for the needs of this project.

Braedon:

- Wrote the first draft of the code to collect data from the sensors.
- Configured the hardware implementation of the device.
- Created first draft of MATLAB visualization code.