A computer generated image of a machine

Description automatically generated

Introduction

* 1. Overview of the Sonar Drone Detector 3D (SoDDD 3D)

The SoDDD 3D maps a 3D space using an ultrasonic sensor and two servo motors. To send the data wirelessly, the system is connected to an Arduino with an XBee shield and XBee board attached. These communicate the necessary information via the Zigbee protocol. The ultrasonic sensor determines the distance to the nearest object. It uses one servo to horizontally rotate it and another to vertically rotate the sensor. This pan-tilt creates a neat 3D or 2D map of the terrain around it that will be stored as a matrix. To detect a drone, multiple or iterative “baseline” values are taken of the starting, drone-less environment. This baseline scan could theoretically encase an entire 13 ft. dome. However, it currently maps 75% of this dome due to the 135-degree rotation limit of the HS-422 Deluxe servo motors. In future, a similar system can be used to create a 2D or 3D scan with LiDAR. However, it avoids the incorrect readings that ultrasonic is subject to in wide, open spaces. This would require a larger array than an Arduino can handle, but the scan would reach up to 30 feet. MATLAB may be of use for such a task.

* 1. Purpose of the Documentation

The purpose of this documentation is to explain why this project is needed and how to produce it yourself. It is a common issue in engineering to lose knowledge of how to make a product. This causes unnecessary time wasted to “re-invent the wheel” instead of using acquired knowledge. This document should also fully explain the theory of the code and the current system’s limitations.

2.1 List of Required Components

* 2 HS-422 Deluxe Servo Motors
* 1 SR-04 Ultrasonic Sensor
* 1 Arduino Uno
* 1 Xbee Shield
* 1 XBee Module
* Soldering Kit
* Highly Accurate 3D Printer (Ultimaker preferred)
* Jumper Wires and Wire Headers

2.2 Wiring Diagram

A diagram of a circuit board

Description automatically generated

Note: This is the wiring diagram for the 3D configuration. In 2D, the horizontal rotation servo uses D10 so that the part does not require reassembly.

2.3 External Libraries or Dependencies:

The following libraries are all for the Xbee module except for ServoTimer2. ServoTimer2 must be used instead of the default servo library for Arduino because of a conflict between all known Zigbee Arduino library’s timer and the regular servo library’s timer.

ServoTimer2.h

* https://github.com/nabontra/ServoTimer2

XBee.h

* https://www.arduino.cc/reference/en/libraries/xbee-arduino-library/

Printers.h

* https://github.com/andrewrapp/xbee-arduino/blob/master/Printers.h

AltSoftSerial.h

* <https://www.arduino.cc/reference/en/libraries/altsoftserial/>

Newping

* https://www.arduino.cc/reference/en/libraries/newping/

Hardware Setup

3.0 Description of Each Component and Connection Instructions

There are 6 components to this build: an Arduino, Xbee Shield, XBee Module, two servo motors, and an ultrasonic sensor. There is also a case and servo assembly.

3.1 Case and Servo Assembly

3D print the following parts:

* + A. ultrasonic servo arduino holster 7.24
  + B. ultrasonic servo arduino holster bottom and arduino holder (not case!)
  + C. ultrasonic to servo 7.19.23-2
  + D. cchannel extruded 7.19
  + E. OPTIONAL: servo securer insert.04t OR .05t

Once A and B have been printed, attach the holster bottom to **either** of the two following spots:

A screenshot of a computer

Description automatically generated

It should lay flush with the bottom of the holster. I chose number 1. Now, drill two 3.2mm screws or some threaded inserts into each side.

A screenshot of a video game console

Description automatically generated

It is now time to wire the Arduinos!

3.2 Arduino Wiring

The Arduino is the microprocessor that allows the user to utilize the sensor and motors while communicating over serial. A 9V battery will be attached to the DC in of the Arduino to power the device. A XBee shield must then be attached to the Arduino. This may involve soldering depending on the pins of the purchased XBee Shield. Next, one should solder the 5V of the Arduino and the GND to two separate rows of pads on the XBee Shield. After the shield is attached and soldered, digital pins D2 and D3 must be attached to D8 and D9 respectively. This allows Arduino to communicate with the XBee module. Pin D10 should be attached to whichever servo is slotted directly into the giant holster. The important wire is the yellow cable (PWM) on the servo. Similarly, D13 should attach to the yellow cable of the outermost servo (omit in 2D configuration). Lastly, attach D11 to the “Trig” pin of the ultrasonic sensor. Similarly, attach D12 to the “Echo” pin. The system is now wired correctly!

3.3 Servo Assembly

Once the case has been printed, slot a servo into the holster. This servo should be plugged into pin D13 regardless of whether this is a 2D or 3D configuration. If in 2D configuration, simply attach the ultrasonic holder into the servo. If in 3D, use the servo-to-servo holder to connect the two servos to each other. Lastly, attach the ultrasonic sensor to the other servo and make sure it is pointed upwards.

4.0 Testing and Troubleshooting Hardware

All ultrasonic sensors and servos must be tested.

4.1 Servo Testing

One should use an example code to find the maximum servo rotation, in degrees. Using the ServoTimer2 library, one can use the servo.write command with values of 544 and 2400 to find the maximum and minimum rotations. These values were taken from the regular servo library and are the extreme times, in microseconds, for the pulse width of servos in the Arduino language. A translucent protractor can then be used to find the total rotation once the code has been built. Once one determines the maximum rotation, they should change the “servoMaxDegrees” variable in the ILOVEARDUINOSERVO2D or ILOVEARDUINOSERVO3D code.

4.2 Finding if an Ultrasonic Sensor is Consistent

According to my testing, there are two kinds of defective SR-04 ultrasonic sensors. The first is a measurement error. Some sensors add roughly 0.9 inches to the measurements. The other starts with adding 0.5 inches to the distance and increases as the distance between the sensor and the nearest object increases. The first is so minimal that it does not need to be accounted for. However, the second may be unreliable and should be avoided. Either way, to figure out which you have is quite simple. First, use a ruler and tape to place tape every 3 inches up to 18 inches. Next, run the ultrasonic test code included amongst the other files. Line up the farthest protruding parts of the ultrasonic sensor with the tip of the tape and write down each of your measurements. If it is a constant error <1-2 inches (with up to 10% deviation between measurements), you have a perfectly good ultrasonic sensor. If it ranges from 0.5 inches of error to >1 inch of error, your sensor may be unreliable.

The other common error for sensors is detecting objects at distances greater than 5 feet. To test for this, face the sensor at a wall that is between 8-9 feet away. If it can consistently (>75%) return values that are not equal to 0 or greater than 140in, your sensor can detect far away objects. Note, an ultrasonic sensor that was used in this project had its effective range decrease from 10 feet to less than 5 feet after extensive testing over the course of this summer. As such, sensors may need to be retested each summer.

Software Setup

5.1 Programming Language and Environment Used

Arduino was used for this project. An incomplete MATLAB code is also provided; however, attempting to use the XBee shield with MATLAB through XCTU caused two XBee modules to become “bricked”. A test ultrasonic code is also attached to find the maximum rotation of the sensors.

5.2 Installation Guide for Necessary Software

One should download all libraries from the links in 2.3. If the link is dead, the user should search, “Arduino ArduinoLibraryNameHere.h” on Google. Once the library is downloaded, one should go to Arduino IDE>Sketch>Include Libraries>Add .Zip Libraries and click on the files they downloaded. Note: one may have to extract the file from Github, find the condensed folder that the Arduino requires, Zip it themselves, and then install it if the folder is not built properly.

6.0 Code Explanation

The following will explain how the Arduino code works and its purpose.

6.1 High-Level Explanation of the Code's Purpose

The code attempts to detect drones via deviation from the starting map of the world. The beam width of a SR-04 sensor is 15 degrees.[[1]](#footnote-1) Dividing the semi-sphere into 15-degree chunks gives 12 values plus a “zero”. The result is a 13x13 baseline to map the entire dome of range. Each new row means an increase in horizontal rotation, and each column represents a change in vertical rotation. Note: in 2D, this is flipped.

6.2 Baselines

Sonar has reliability issues. Due to the inconsistency of how beams fire or bounce off of objects, there are sometimes multiple different values that an ultrasonic sensor returns despite firing from the same place. Because of this, we use 24 baselines to increase the probability that our map will contain both of these values. 3D does this by iterating one baseline many times while the 2D configuration has 24 “true” baselines. Since 2D only has one row, there is a function added to turns the two confusingly named baselines “baseline1” and “baseline2” into 24 baselines. We are now ready for detection.

6.3 What Qualifies as Detection?

Detection consists of anything in the scan array that is more than 10% closer than all four of the baseline values. There is a function called “deviatesEnough” that checks for this. We must also make sure to ignore distance readings that claim the drone is in a position that is farther than all of our baselines. This called for the creation of the “notTooFar” function. Then there are the extreme values. “Detections” whose value is greater than 150 inches are both invalid and unreliable because the maximum theoretical range is between 12 to 13 feet. If a reading is 0, it should be ignored because it is a reading error, or something is colliding with our sensor. Neither of those causes of a 0 value are ideal, but they should be ignored. It is worth noting that mismatching 0 values in the baseline are artificially changed to 169 to prevent a similar issue in baselines. Otherwise, if a 0 is detected in a baseline, the corresponding area will never truly be scanned again. Detections would simply be flagged as being too far away!

6.4 Zigbee Packet Structure:

Once a detection is made, a packet is sent via Zigbee serial to the coordinator Arduino. The coordinator knows what kind of packet it is receiving based off the length of the packet and the number contained by the first byte in the packet. The first byte is a type. Each sensor has a category and so does the jammer! The type for both versions of SoDDD is 4. The second and third bytes are the sensorID. This is a value that should be unique to each sensor, even if they are both SoDDD or LiDAR sensors. Fourth and fifth are the distance of the detection. Sixth and seventh are vertical rotation. This will always be outputted as a 0 if the 2D configuration is used, and ILOVEARDUINOSERVO2D is uploaded. Lastly, bytes eight and nine are the horizontal rotation. A useful visual depicts it below.

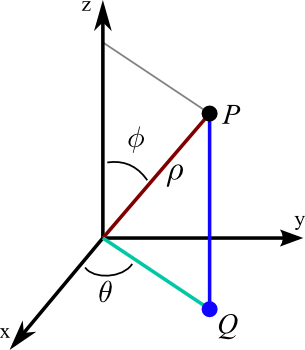
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Type | sensorID | | Distance | | Vertical Rotation | | Horizontal Rotation | |

6.5 Spherical Coordinates Calculations and Complications:

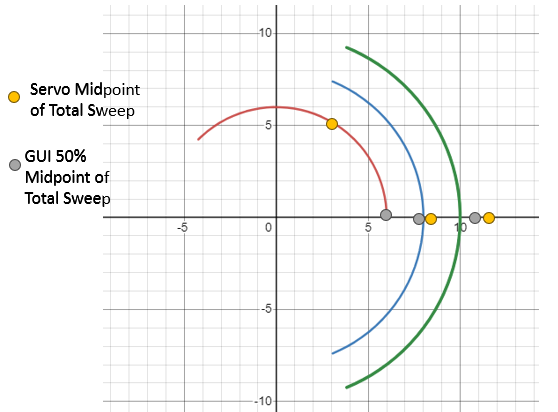
Calculating horizontal rotation and vertical rotation are not as simple as one would expect. This is because servos are defined by 0-180 degrees whether they are measuring phi or theta; however, spherical coordinates call for a horizontal rotation of 0-360 degrees and a vertical one of 0-180! Additionally, it was desired that the coordinator Arduino should receive the Arduino data with proper spherical coordinates while being centered. It must now be noted that servos are finicky. Although the HS-22 Deluxe’s claim of sweeping 180 degrees, they only have around 130 degrees in actuality![[2]](#footnote-2) The following procedure is used to find the coordinates:

1. Degree increments \* rows and columns
2. Centering: Rotation – servoMaxDegrees/2
3. Servos go from 0-130 degrees
4. RotationActual= -1 \* Rotation
5. If phi <0, phi= -phi and theta=theta+18

A few visuals showing the coordinate system and transformations are shown below:



Our spherical coordinate system



Transformations of the Center of Sweep

A graph of a graph with arrows and a red arrow

Description automatically generated

Transformations of the Direction of Sweep

7.1 Current Limitations and Future Improvements

The current limitations are due to two constraints: servo sweep and SoNAR performance. The HS-422 Deluxe servo motors do not rotate 180 degrees. If they did, one could scan an entire dome instead of ¾ dome. SoNAR performance similarly had issues. When using the SoNAR sensor in Fathomwerx, there were often erroneous detections when the drone was pointed 30-50 feet from a wall. It occasionally read the wall as being 5 feet away instead of returning a minimum or maximum value, I am unsure if this is purely to do with the wide space, the reflectivity of the walls, or something else. Additionally, powered drones seem to make this error far more common. For this reason, LiDAR may be better.

Arduino Uno’s only have 2KB of RAM. LiDAR sensors often have 1.5-2.5 degrees of beam width. The resulting 130x130 matrix is too large for even an Arduino Mega to handle. This amount of storage can be handled if one uses MATLAB due to the large RAM in a modern computer or Raspberry Pi. The issue with MATLAB is that improper setup of the XBee module with XCTU may brick the module. An incomplete MATLAB code is in this folder if this becomes desired. Alternately, one could perform 15-degree mini sweeps where the servos are moved in 1.5-degree increments. The lowest value of the ten measurements in each 15-degree mini sweep can be taken and plugged back into the provided code. This would allow minimal reworking of the current code to allow much greater detection distance, especially for smaller drones.

10.1 Summary of the Project's Success

10.2 Acknowledgments (if any)

10.3 Final Thoughts and Closing Remarks

Appendices

11.1 Additional Resources (links, references, etc.)

11.2 Glossary of Terms (if needed)

11.3 Additional Supporting Documents (e.g., data sheets)

1. It is less than 15 degrees in reality! Because of this, increments of 11.25 degrees are used.. [↑](#footnote-ref-1)
2. Because of this, there is a conversion between servo degrees and real degrees. This is used to move the servo 11.25 degrees. One servo degree is taken from the maximum – minimum pulse width and divided by 180. This value comes from the normal servo library. The “oneRealDegree” variable is based off of this and servoMaxDegrees. [↑](#footnote-ref-2)