Test Data and Procedure

To test the working of this object detection system we had to consider various features from hardware and software to individual components working and the result provided, the error in the results. Some of the false negative and false positive and margin of error.

Various Testing Phases and Procedures:

1. The first factor was the wiring and connection of each component to Beaglebone Black; - Every component first needed to be connected to the Beaglebone black and work properly at the respective pin it was connected to whether it was to a GPIO output or to a GND VCC and a PWM we needed to make sure that the connections were working properly and we combined this with the second step of checking whether the component was working well too or not.
2. Second factor was individual components working on their own; - In order for the entire project to work first we needed to make sure that each and very component of the system worked perfectly individually as in we could find some fault equipment if present early on in the process.
   We tested each and every part of the system first the LEDs by blinking them then the buzzer by buzzing it, then we tested a button working in sync with both of them, then we tested the servo motor by running various run time systems from rotating it at a particular angle as input to rotating it continuously from 0 to 180 and then 180 to 0 again finally we tested the ultrasonic sensor at various distances and more ahead.
3. The third step in the testing phase was to check that the code written for the individual components was proper and efficient as moving forward a lot of this code will be reused for the overall implementation of the system and making sure that there are no hidden bugs at this point of time would help sort of and pin point the mistakes early on in the process.
4. The fourth step was to start combining the components one by one and testing their working together: First we combined the LEDs with the buzzers to test their simultaneous working, then we added a push button to turn them on using two threads simultaneously, then we added the servo motor to run continuously first from 0 to 180 degrees and then from 180 to 0 and so on.
5. The ultrasonic sensor had to be tested rigorously: So, we first mounted the ultrasonic sensor and tested it individually if it was giving the right distance, then we added on to the tests by putting objects at various angles to test how efficiently it could detect. Then we tried various lighting areas to make sure that there wasn't anything interfering with the ultrasonic sensor's waves.
6. The 3rd party repository pulled in from Github needed to be tested by checking if all the conditions ere satisfied and all the necessary installations were done before we could start combining the items all together.

7. Finally, we connected all the components together and made sure that each and every component worked properly as before as intended. Even after connecting all the components to the Beaglebone Black and breadboard the components first had to be made sure to work correctly and then when this was done, we proceeded to the next step.
8. Checking if all the components worked properly together ;- The hardware had to be made sure that it worked properly together and also the code running had to be tested in various ways to make sure that there were no bugs and edge cases.
9. All the 4 threads had to be working with the mutex implemented to make sure that the distance had been accessed in proper order one after another so that there isn't a race case and mutex locks were to be implemented.
10. Finally, when all the code was correct, we needed to make sure that the LED's were glowing red when the sensor detected and object and glow green when there wasn't anything nearby. Also, the buzzer buzzed when the object was detected and stopped when it moved away from the set sensor.
11. We also tested at various lengths what delay was taking place and what was the time taken to change the change the LEDs and buzzers.

Nominal Testing Procedure after complete assembly:

After complete assembly of the project, we started testing it from various distances and weather it reacted fast or not.

First, we started with the distance we kept an object at various distance from the sensor and noted how much it gave in the output and how far we had kept the object

Next, we noted if it detected if it was too far from the sensor or if it was too close to the sensor.

Then we tested if the red LED would glow instantaneously when we kept an object near the sensor and the green glowed instantly when we kept the object out of the decided 30 cm area.

We also started testing with the 30 cm limit by keeping it as close as 0.5cm to glow the LED to as far as 70 cm to test if it accurately was detecting the object.

We also ran the similar codes in python as well as C to check for the quickness/efficiency and time complexity of the program and came to the conclusion that python was almost slower in seconds as compared to the quick millisecond reaction time of the C program.

Off-Nominal Testing Procedure:

Testing in different lighting conditions: we tried testing it underly dimly lit room and it performed comparatively less accurately but not project breaking. Also there were some instances where the sensor didn't collect the data properly from the object distance. In a dimly lit or dark room.

Testing in noisy environments: when kept various objects near the sensor the sensor tended to make errors and sometimes giving the closest object as the value. And didn't consider the various objects at various distances.

Testing with different surface materials: when various other kinds of items from bottles to books were kept the result was constant as it didn't seem much of a difference but there might be a chance of a different kind of result when a transparent or a translucent item might reflect back the sensor wave.

Testing with different power supply voltages: this could have been done but the risk of circuiting the Beaglebone we decided to avoid it but moving forward this could be a factor causing issues to the ultrasonic sensor.

Data validation: We Validated the data collected by the sensor against other sources of data or standards to ensure its accuracy and reliability. We kept the object at counted lengths from the sensor and checked if the sensor gave accurate distance answer or not. And if it wasn't accurate what was the error range and fi it could be due to what factors.

The above tests show the various distances at which the object was kept from the sensor and it detected and when no object was kept.
There was error at various times for example it showed 1562.59 cm when nothing was kept in front of it and facing the ceiling.

Also, there was error when the object was too close to the sensor or if there were multiple minor obstructions in the way of the object even if it was just farther than 30 cm distance for standard testing.

Coverage:

Path Coverage: Path coverage refers to the percentage of possible execution paths that have been executed during the testing of the code. The code for this project has various N number of paths but one major starting branch. Approximately 80-90% of the paths must have been tested due to major paths having infinite while loops. There might be some paths where there is an equipment error and it cannot be recreated and possibly not tested hence the Path coverage must be around 85-90%.

Branch Coverage: Branch coverage is a measure of how much of a program's branching code has been tested. The code for this project has a few major branches which have been tested rigorously for example a branch where the distance is less than 30 so the red LED branch must be selected with the buzzer turning on. Hence similarly most of the major branches of the code have been tested rigorously but there might be some branches with errors in writing in GPIO which might not have been tested clearly hence it might have around 90% major Branch Coverage

Code Coverage: Code coverage is a measure of the amount of code in a software application that has been executed during a particular test run. For this project the major code lines are for GPIO setup and functions and threads hence majority of the code need to be running to be tested. But there are pieces of code for example GPIO write error which have been less tested and hence the Code Coverage of this project might be about 90-95% due to the high number of code lines and less amount of error cases and more of include cases and while loops and threads and important functions.

In conclusion: 85-90% Path Coverage, around 90% Branch Coverage and around 90-95% Code Coverage must have been tested.

Sample test data output:

When we run the code and put an object at various distance from the ultrasonic sensor, we got the following results.

```
debian@beaglebone:~/project$


trigger: [P8_17]

echo: [P8_16]

Setup completed!

Distance: [65.09] cm.

Distance: [235.88] cm.

Distance: [227.96] cm.

Distance: [169.49] cm.

Distance: [668.68] cm.

Distance: [703.58] cm.

Distance: [153.68] cm.

Distance: [154.9] cm.

Distance: [151.93] cm.

Distance: [141.97] cm.

Distance: [713.41] cm.

Distance: [149.05] cm.
```

```
Distance: [715.57] cm.
Distance: [140.78] cm.
Distance: [104.04] cm.
Distance: [224.93] cm.
Distance: [169.74] cm.
Distance: [624.78] cm.
Distance: [169.09] cm.
Distance: [5.18] cm.
Distance: [234.23] cm.
Distance: [151.69] cm.

debian@beaglebone:~/project$
trigger: [P8_17]
echo: [P8_16]
Setup completed!
Distance: [344.55] cm.
Distance: [222.82] cm.
Distance: [726.32] cm.
Distance: [169.1] cm.
Distance: [106.1] cm.
Distance: [84.54] cm.
Distance: [14.71] cm.
Distance: [152.72] cm.
Distance: [550.04] cm.
Distance: [93.32] cm.
Distance: [13.39] cm.
Distance: [295.2] cm.
Distance: [1562.59] cm.
Distance: [137.95] cm.
```