Lab 6: Sonar (Ultrasonic Range Finders) and Camera Sensors

*ECE 564: Fundamentals of Autonomous Robots Lab*

*Team 1: Jacob Cassady, Chase Crutcher, and Olalekan Olakitan Olowo*

*October 19, 2019*

*The group members have worked together and face-to-face at all stages of this project work. The contributions of members to the report and to the codes are equal.*

*(Initials of group members)*

# Introduction

This lab is used to show that sonar sensors and cameras can be used as tools to extract information about the environment the robot is placed in. Both tools can be used for things such as range finding, object detection-avoidance, and target tracking. While both tools are limited to the extent at which they can provide the Demobot with all of its surroundings, this lab shows what information they can give the Demobot and how it can help the robot do certain tasks when placed in an open world environment.

# Lab Parts

## Determining the calibration coefficient and minimum sensing distance of MB1010 sonar sensor

Table : Distance vs. Analog Response

|  |  |
| --- | --- |
| Distance (cm) | Analog Response |
| 0 | 52 |
| 2 | 52 |
| 4 | 52 |
| 8 | 52 |
| 10 | 52 |
| 12 | 52 |
| 14 | 52 |
| 16 | 52 |
| 18 | 60 |
| 20 | 68 |
| 22 | 76 |
| 24 | 76 |
| 26 | 85 |
| 28 | 85 |
| 30 | 92 |
| 32 | 100 |
| 34 | 108 |
| 36 | 115 |
| 38 | 122 |
| 40 | 130 |

A close up of a map

Description automatically generated

Figure 1 : Linear Regression for Calibration Coefficients

The graph shown in **Figure 1** above shows the linear relationship between the sonar analog response with respect to distance in centimeters. The calibration coefficient m = 3.04371 with an offset of 4.81585. The minimum distance response from the sensor resulted in a distance of 15.502 cm.

## Rotational tracking using Camera

The rotational tracking algorithm gets the x location of the largest object matching the channel number of interest. It then determines if this x location is within an interval. If it is, it does nothing. If it isn’t it rotates proportional to the difference between its objective x location and its current x location.

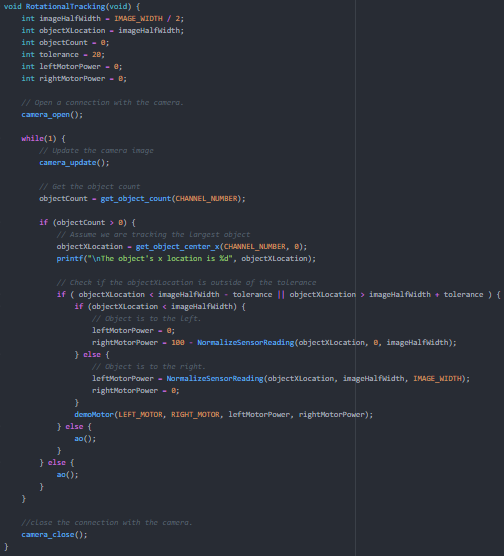


Figure 2 : Rotational Tracking Function

## Translational tracking using Camera

The translational tracking algorithm gets the y location of the largest object matching the channel number of interest. It then determines if this y location is within an interval. If it is, it does nothing. If it isn’t it moves forward or backward proportional to the difference between its objective y location and its current y location.

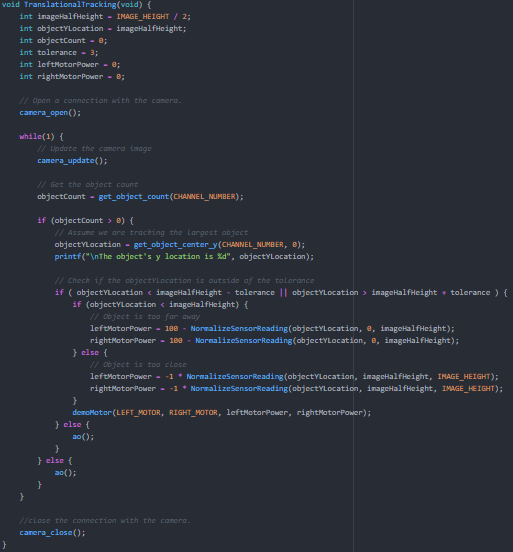


Figure 3 : Translational Tracking Function

## Rotational and Translational tracking using Camera

The rotational and tracking algorithm makes use of both of the previous algorithms by taking 50% of its motor power from each and adding them together. First, the translational power is determined using the same algorithm as the translational tracking algorithm, and then divided in half. The rotational power portion is then calculated with a maximum of 50 and a minimum of 0. The rotational and translational values are then summed and the resultant powers are fed into the motors.

### Function

void RotationalAndTranslationalTracking(void) {

int imageHalfHeight = IMAGE\_HEIGHT / 2;

int objectYLocation = imageHalfHeight;

int imageHalfWidth = IMAGE\_WIDTH / 2;

int objectXLocation = imageHalfWidth;

int objectCount = 0;

int rotationalTolerance = 20;

int translationalTolerance = 8;

int leftMotorPower = 0;

int rightMotorPower = 0;

// Open a connection with the camera.

camera\_open();

while(1) {

// Update the camera image

camera\_update();

// Get the object count

objectCount = get\_object\_count(CHANNEL\_NUMBER);

if (objectCount > 0) {

// Assume we are tracking the largest object

objectXLocation = get\_object\_center\_x(CHANNEL\_NUMBER, 0);

objectYLocation = get\_object\_center\_y(CHANNEL\_NUMBER, 0);

printf("\nThe object's x location is %d", objectXLocation);

printf("\nThe object's y location is %d", objectYLocation);

// Check if the objectYLocation is outside of the tolerance

if ( objectYLocation < imageHalfHeight - translationalTolerance || objectYLocation > imageHalfHeight + translationalTolerance ) {

if (objectYLocation < imageHalfHeight) {

// Object is too far away

leftMotorPower += 100 - NormalizeSensorReading(objectYLocation, 0, imageHalfHeight);

rightMotorPower += 100 - NormalizeSensorReading(objectYLocation, 0, imageHalfHeight);

} else {

// Object is too close

leftMotorPower += -1 \* NormalizeSensorReading(objectYLocation, imageHalfHeight, IMAGE\_HEIGHT);

rightMotorPower += -1 \* NormalizeSensorReading(objectYLocation, imageHalfHeight, IMAGE\_HEIGHT);

}

}

printf("\nY Power component is %d", leftMotorPower);

leftMotorPower /= 2;

rightMotorPower /= 2;

// Check if the objectXLocation is outside of the tolerance

if ( objectXLocation < imageHalfWidth - rotationalTolerance || objectXLocation > imageHalfWidth + rotationalTolerance ) {

if (objectXLocation < imageHalfWidth) {

// Object is to the left.

rightMotorPower += 100 - NormalizeSensorReading(objectXLocation, 0, imageHalfWidth) - 50;

} else {

// Object is to the right.

leftMotorPower += NormalizeSensorReading(objectXLocation, imageHalfWidth, IMAGE\_WIDTH) - 50;

}

}

// Move motor appropriately.

if (leftMotorPower == 0 && rightMotorPower == 0) {

ao();

} else {

printf("\nleftMotorPower before bind %d", leftMotorPower);

printf("\nrightMotorPower before bind %d", rightMotorPower);

leftMotorPower = bind(leftMotorPower, -100, 100);

rightMotorPower = bind(rightMotorPower, -100, 100);

printf("\nleftMotorPower after bind %d", leftMotorPower);

printf("\nrightMotorPower after bind %d", rightMotorPower);

demoMotor(LEFT\_MOTOR, RIGHT\_MOTOR, leftMotorPower, rightMotorPower);

}

} else {

ao();

}

}

//close the connection with the camera.

camera\_close();

}

## Obstacle avoidance while tracking a target

The obstacle avoidance while tracking a target algorithm is similar to the rotational and translational tracking algorithm. The rotational and translational tracking algorithm is reformatted to run as a thread. A sonar thread is added to handle the obstacle avoidance. Whenever the sonar produces a response signaling an obstacle is close, a flag is raised causing the Rotational and Translational Tracking Thread to stop sending commands to the motor. The obstacle avoidance thread then finishes a routine to move itself away from the obstacle. Finally, the flag is lowered after the routine is finished and the rotational and translational tracking thread continues control.

### Thread Driver Function

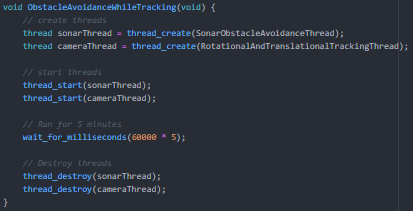


Figure 4 : Thread Driver Function

### Avoidance Thread Function



Figure : Sonar Obstacle Avoidance Thread Function

### Rotational and Translational Tracking Thread

void RotationalAndTranslationalTrackingThread(void) {

int imageHalfHeight = IMAGE\_HEIGHT / 2;

int objectYLocation = imageHalfHeight;

int imageHalfWidth = IMAGE\_WIDTH / 2;

int objectXLocation = imageHalfWidth;

int objectCount = 0;

int rotationalTolerance = 20;

int translationalTolerance = 8;

int leftMotorPower = 0;

int rightMotorPower = 0;

// Open a connection with the camera.

camera\_open();

while(1) {

while(!gv\_sonarEvent) {

// Update the camera image

camera\_update();

// Get the object count

objectCount = get\_object\_count(CHANNEL\_NUMBER);

if (objectCount > 0) {

// Assume we are tracking the largest object

objectXLocation = get\_object\_center\_x(CHANNEL\_NUMBER, 0);

objectYLocation = get\_object\_center\_y(CHANNEL\_NUMBER, 0);

gv\_lastX = objectXLocation;

gv\_lastY = objectYLocation;

// Check if the objectYLocation is outside of the tolerance

if ( objectYLocation < imageHalfHeight - translationalTolerance || objectYLocation > imageHalfHeight + translationalTolerance ) {

if (objectYLocation < imageHalfHeight) {

// Object is too far away

leftMotorPower += 100 - NormalizeSensorReading(objectYLocation, 0, imageHalfHeight);

rightMotorPower += 100 - NormalizeSensorReading(objectYLocation, 0, imageHalfHeight);

} else {

// Object is too close

leftMotorPower += -1 \* NormalizeSensorReading(objectYLocation, imageHalfHeight, IMAGE\_HEIGHT);

rightMotorPower += -1 \* NormalizeSensorReading(objectYLocation, imageHalfHeight, IMAGE\_HEIGHT);

}

}

leftMotorPower /= 2;

rightMotorPower /= 2;

// Check if the objectXLocation is outside of the tolerance

if ( objectXLocation < imageHalfWidth - rotationalTolerance || objectXLocation > imageHalfWidth + rotationalTolerance ) {

if (objectXLocation < imageHalfWidth) {

// Object is to the left.

rightMotorPower += 100 - NormalizeSensorReading(objectXLocation, 0, imageHalfWidth) - 50;

} else {

// Object is to the right.

leftMotorPower += NormalizeSensorReading(objectXLocation, imageHalfWidth, IMAGE\_WIDTH) - 50;

}

}

// Move motor appropriately.

if (leftMotorPower == 0 && rightMotorPower == 0) {

ao();

} else {

leftMotorPower = bind(leftMotorPower, -100, 100);

rightMotorPower = bind(rightMotorPower, -100, 100);

demoMotor(LEFT\_MOTOR, RIGHT\_MOTOR, leftMotorPower, rightMotorPower);

}

} else {

ao();

if(gv\_lastX < IMAGE\_WIDTH / 2) {

// Pan to the left looking for the object.

goDemobotMav(LEFT\_MOTOR, RIGHT\_MOTOR, 600, 0, 1000);

} else {

// Pan to the right looking for the object.

goDemobotMav(LEFT\_MOTOR, RIGHT\_MOTOR, 600, 1000, 0);

}

}

}

wait\_for\_milliseconds(100);

}

//close the connection with the camera.

camera\_close();

}

## Extra Credit

The extra credit portion is shown in the code above in section 2.5.3 of this document. When no object is found, the robot pans in the direction the object was seen last. This ensures the robot searches for the object whenever it is loses sight instead of sitting still.

# Conclusion

This lab shows how the sonar sensor and camera were used with the Demobot to help it perform certain tasks. The senor and Camera were mounted to the Demobot and tested to determine the calibration coefficient and minimum sensing distance so as to help perform later tasks. Once the sensor, camera, and wallaby microcontroller were all working together, the camera was then used to target certain colors and the sonar sensor was used to detect and avoid obstacles while tracking a target. This lab shows that the use of these two tools prove crucial to any robot trying to operate in an open world environment. The final demonstration of the lab activity was shown to the TA at its completion.

# Suggestions

Make the assigned colors the Demobot is supposed to look for colors that people would not ordinarily wear so the camera does not detect someone's shirt or other article of clothing. Also if possible the provision of an intentional isolated space or perhaps an open field to properly test for the distance reach of the sonar sensors as the sensor had to be extended outside a window for it not to pick up distance of nearby furniture or objects not considered during the experiments.