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)

1 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.

2 LAB PARTS

2.1 DETERMINING THE CALIBRATION COEFFICIENT AND MINIMUM SENSING DISTANCE OF MB1010 SONAR SENSOR

Table 1 : 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

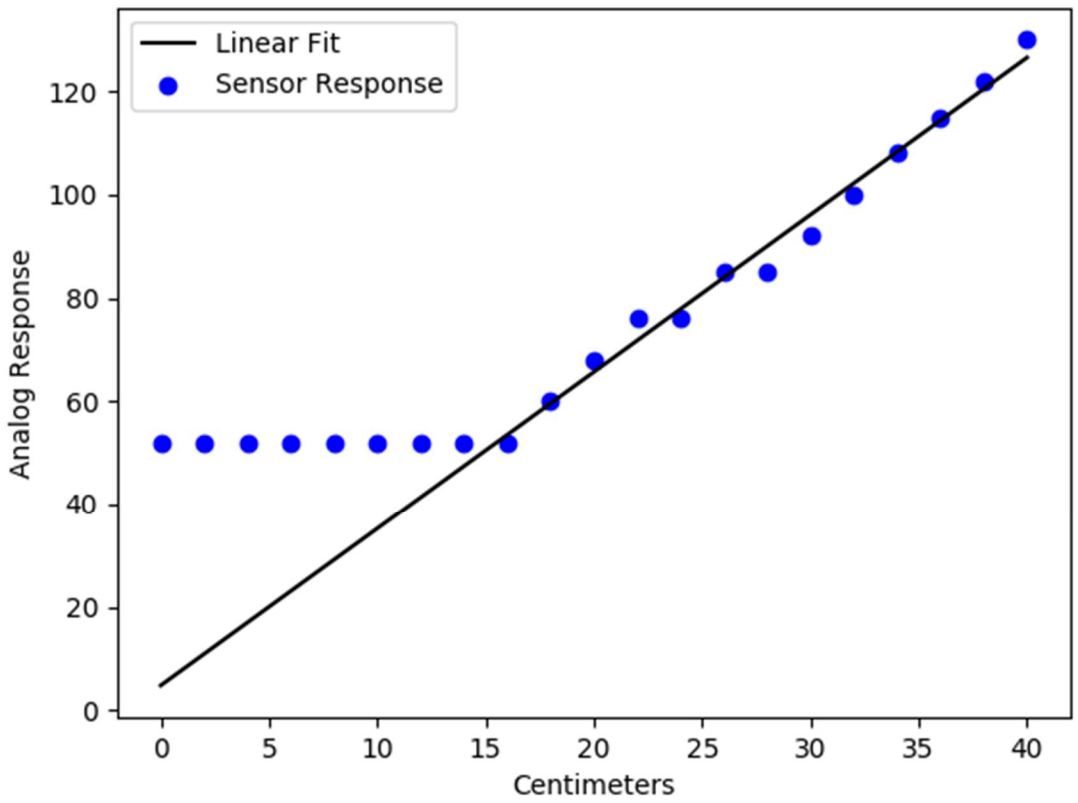

Figure 1 : Linear Regression for Calibration Coefficients

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.

2.2 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.

```
void RotationalTracking(void) {
    int imageHalfWidth = IMAGE_WIDTH / 2;
    int objectXLocation = imageHalfWidth;
    int objectCount = 0;
    int tolerance = 20;
    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);
            printf("\nThe object's x location is %d", objectXLocation);

            // Check if the objectXLocation is outside of the tolerance
            if ( objectXLocation < imageHalfWidth - tolerance || objectXLocation > imageHalfWidth + tolerance ) {
                if (objectXLocation < imageHalfWidth) {
                    // Object is to the left.
                    leftMotorPower = 0;
                    rightMotorPower = 100 - NormalizeSensorReading(objectXLocation, 0, imageHalfWidth);
                } else {
                    // Object is to the right.
                    leftMotorPower = NormalizeSensorReading(objectXLocation, imageHalfWidth, IMAGE_WIDTH);
                    rightMotorPower = 0;
                }
                demoMotor(LEFT_MOTOR, RIGHT_MOTOR, leftMotorPower, rightMotorPower);
            } else {
                ao();
            }
        } else {
            ao();
        }
    }

    //close the connection with the camera.
    camera_close();
}
```

Figure 2 : Rotational Tracking Function

2.3 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.

```
void TranslationalTracking(void) {
    int imageHalfHeight = IMAGE_HEIGHT / 2;
    int objectYLocation = imageHalfHeight;
    int objectCount = 0;
    int tolerance = 3;
    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
            objectYLocation = get_object_center_y(CHANNEL_NUMBER, 0);
            printf("\nThe object's y location is %d", objectYLocation);

            // Check if the objectYLocation is outside of the tolerance
            if ( objectYLocation < imageHalfHeight - tolerance || objectYLocation > imageHalfHeight + tolerance ) {
                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);
                }
                demoMotor(LEFT_MOTOR, RIGHT_MOTOR, leftMotorPower, rightMotorPower);
            } else {
                ao();
            }
        } else {
            ao();
        }
    }

    //close the connection with the camera.
    camera_close();
}
```

Figure 3 : Translational Tracking Function

2.4 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.

2.4.1 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();
```

}

2.5 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.

2.5.1 Thread Driver Function

```
void ObstacleAvoidanceWhileTracking(void) {
    // create threads
    thread sonarThread = thread_create(SonarObstacleAvoidanceThread);
    thread cameraThread = thread_create(RotationalAndTranslationalTrackingThread);

    // start threads
    thread_start(sonarThread);
    thread_start(cameraThread);

    // Run for 5 minutes
    wait_for_milliseconds(60000 * 5);

    // Destroy threads
    thread_destroy(sonarThread);
    thread_destroy(cameraThread);
}
```

Figure 4 : Thread Driver Function

2.5.2 Avoidance Thread Function

```
void SonarObstacleAvoidanceThread(void) {
    int sonarAnalogReading = 0;
    float sensorDistance = 0;
    int sonarThreshold = 20;

    while(1) {
        // Get sensor reading
        sonarAnalogReading = analog(SONAR_SENSOR);

        sensorDistance = ConvertSonarToDistance(sonarAnalogReading);

        if (sensorDistance < sonarThreshold) {
            ao();
            gv_sonarEvent = 1;
            printf("\nSonar Event is occuring!");

            if(gv_lastX < IMAGE_WIDTH / 2) {
              // Back up
              goStraightMav(LEFT_MOTOR, RIGHT_MOTOR, 2000, -500);
              // Turn left
              goDemobotMav(LEFT_MOTOR, RIGHT_MOTOR, 1000, 0, 1000);
              // Go straight
              goStraightMav(LEFT_MOTOR, RIGHT_MOTOR, 2500, 500);
              // Turn Right 90
              goDemobotMav(LEFT_MOTOR, RIGHT_MOTOR, 1000, 1000, 0);
            } else {
              // Back up
              goStraightMav(LEFT_MOTOR, RIGHT_MOTOR, 2000, -500);
              // Turn Right 90
              goDemobotMav(LEFT_MOTOR, RIGHT_MOTOR, 1000, 1000, 0);
              // Go straight
              goStraightMav(LEFT_MOTOR, RIGHT_MOTOR, 2500, 500);
              // Turn Left
              goDemobotMav(LEFT_MOTOR, RIGHT_MOTOR, 1000, 0, 1000);
            }

            gv_sonarEvent = 0;
        }
    }
}
```

Figure 5 : Sonar Obstacle Avoidance Thread Function

2.5.3 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();
}
```

2.6 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.

3 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.

4 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.