Memo-ECE425-LAB2

Date: 1/9/2022

To: Professor Berry

Team: Leonardo - Nathan Lee, Shantao Cao

The purpose of this Lab was to get the robot to recognize obstacles and avoid the object using the proper method. For this Lab, we used LEONARDO’s four IR sensors and calibrated them accordingly. The obstacles avoidance behavior used different levels of obstacle avoidance and each level built off each other. The result is that LEONARDO (LEO) can execute a smart homing function and arrive at a specified location while avoid random obstacles in its path.

**Prelab: State Diagram and Simulation**

The obstacle avoidance behavior was built on levels, each one being built on the previous. The first level was to prevent the robot from hitting obstacles. This was done through aggressive kid and shy kid behaviors. Aggressive kid behavior stops the robot once an obstacle is recognized and shy kid behavior is built on that by being repelled from the obstacle. The planning was done through a state diagram and can be seen in the first figure in Appendix A: avoid obstacle. This behavior was also simulated in Tinkercad (Appendix B) before applying the behaviors on LEO. The next level was to add a wander function with the shy kid behavior to have the robot move around a space while avoiding obstacles (Appendix A: smart wander). The final level was to have the robot go to a set location while avoiding obstacles in its path (Appendix A: smart homing).

**IR Sensor Calibration**

For this lab, we only utilized the four IR sensors on LEO. We calibrated the front IR and the left IR sensors. The back IR uses the same calibration equation as the front IR. The same applies for the left and right IR sensors. Calibration was done through taking data of the analog values to their distance measurement, then linearizing the data and creating a line of best fit (Appendix C)[[1]](#footnote-1). The moving average of 200 data points was used to reduce noise in the calibration of the IR sensors. The linearization coefficients were experimentally determined to maximize the R-squares value to 4 decimal places. The IR sensors were calibrated from 1 in to 20 in. The resulting calibration equations:

|  |  |  |
| --- | --- | --- |
|  |  | (1) |
|  |  | (2) |

After implementing the calibration equation for the IR sensors, the distance measurement is discrete to the nearest inch with an accuracy of 1 inch. The moving average is set to 10 data points for all levels.

The IR sensors was enough for LEO to sense its terrain and the use of the additional sonar sensors would inhibit its behavior. For the sonar sensors, we have tried using the NewPing arduino library and it gave us accurate distance readings. The NewPing Library uses the standard time of flight sensing with a ping and a trigger but has some overhead compensation and better calibration done to the library. During implementation of the sonar sensors for the shy kid behavior, one of the two sonar sensors would trigger and give us false obstacle readings. The IR sensors proved to be enough for this lab.

**Avoid Obstacle**

The most basic form of obstacle avoidance starts with recognizing the object. This behavior is shown through aggressive kid: the robot would continually move forward until an obstacle is detected in front of it and stop all movement; the robot will continue to move forward when the object is removed. LEO has successful executed the aggressive kid behavior.

Building on top of aggressive kid behavior is the shy kid behavior. The robot would detect obstacles around it and move away from the obstacle. We initially used a proportional controller for the speed of each wheel relative the distance from the obstacle. This method proved well for 1- and 4-wall, but LEO could not turn tight enough to avoid the 2- and 3-wall situations. This proved to be most problematic with the 3-wall situation because LEO would get stuck in a loop of going back and forth. We switched to a state machine controller using obstacle detection based off the position of the walls. LEO would turn to a direction away from the walls and move away from it. This approach proved to be successful for the shy kid behavior.

There are still a few issues for obstacle avoidance behavior. The major issue is that LEO’s field of view is very limited because it only has 4 single point IR sensors. This means that the corners of LEO’s frame may collide with obstacles. This is solved through increasing the obstacle detection threshold to make LEO much more caution around obstacles. Another issue was the sensors would misread distances because the obstacle is at an angle to the sensor. This issue results in a bad sensor reading and result in an unpredicted movement. This was improved by editing the obstacles and LEO’s behavior to always point the sensors perpendicular to the obstacles.

**Smart Wander**

The next level above the shy kid behavior is implementing it onto a wandering algorithm. LEO should wander using a random algorithm and avoid obstacles when any obstacles is detected. The random wander algorithm for LEO is based on a random number generator to determine a direction to take: 0 for forward, 1 for left and 2 for right. Using a timer interrupt for sensor checking and a state machine, this lets us easily add layers on top of the obstacle avoidance behavior. A state machine is implemented to have LEO change between random wandering and obstacle avoidance behavior. With the use of Boolean statements, LEO was able to check its current state and switch to the next state: switching between randomly wandering and obstacle avoidance.

**Smart Homing**

The final and highest level for this lab is smart homing. LEO would have a designated point to move to and avoid any obstacles in its path. Because of LEO’s limited field of view and the IR sensors limitation, we assumed that the obstacles would be straight lines in the x-y coordinate system. This helps LEO’s pathing algorithm by easily detecting the obstacles walls by reading correct distance values and having the ability to read all the walls of the obstacle. Figure 1 shows the Subsumption architecture and further explains how the smart homing behavior works.

Sensor Readings

Go to goal

Avoid Obstacle

Actuators

**Figure 1:** Subsumption architecture for smart homing. The robot will follow the go to goal behavior until the avoid obstacle behavior will take priority.

Overall, LEONARDO was able to execute all layers of obstacle avoidance from aggressive kid to smart wander and smart homing. As the behaviors became more advance, the implementation began to veer further away from theory. Aggressive and shy kid implementation followed their theory similarly. Smart homing implementation required some compromises, especially with potential field navigation, to execute properly.

**Appendix A: PreLab State Diagrams**

**Avoid Obstacle:**

Diagram

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| States | Input | Output | Next State |
| Random Direction | N/A, No Obstacle | Direction | Go to Goal |
| Go to Goal | Direction | Power Motor | Obstacle |
| Obstacle | Sensor Data | Yes  No | Obstacle Direction  Random Direction |
| Obstacle Direction | Sensor Data & Obstacle | Front, Left, and Right  Front and Left, Left  Front, Right, Front and Right | Turn 180 Degrees and Go Forward  Go Back and Turn Right  Go Back and Turn Left |
| Turn 180 Degrees and Go Forward | Front, Left, and Right | Power Motor | Obstacle |
| Go Back and Turn Right | Front and Left, Left | Power Motor | Obstacle |
| Go Back and Turn Left | Front, Right, Front and Right | Power Motor | Obstacle |

**Smart Wander:**

Diagram

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| States | Input | Output | Next State |
| Stop Moving | N/A, No Obstacle, All 4 Directions | Stop Motor | Obstacle |
| Obstacle | Sensor Data | Yes  No | Direction  Stop Moving |
| Direction | Sensor Data & Obstacle | Only 3 and less directions  All 4 Directions | Direction Vector  Distance  Stop Moving |
| Direction Vector | Only 3 and less directions | Calculate Obstacle Direction Vector | Go with speed and Opposite Direction |
| Distance | Only 3 and less directions | Calculate Obstacle Distance:  10<d<30  D<10 | Slow Speed  Fast Speed |
| Slow Speed | 10<d<30 | Set slow motor speed | Go with speed and Opposite Direction |
| Fast Speed | D<10 | Set fast motor speed | Go with speed and Opposite Direction |
| Go with speed and Opposite Direction | Obstacle Direction Vector and Speed | Power Motor | Obstacle |

**Smart Homing:**

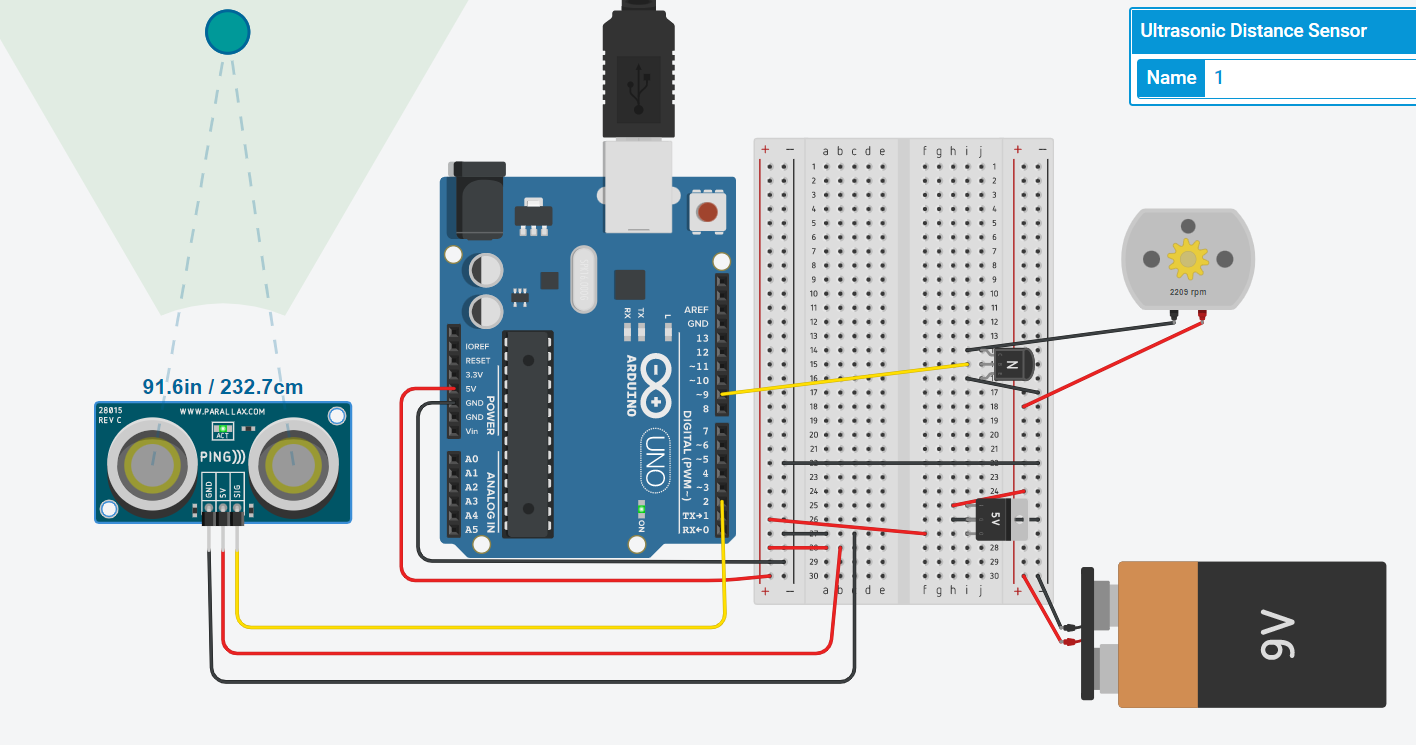
Diagram

Description automatically generated

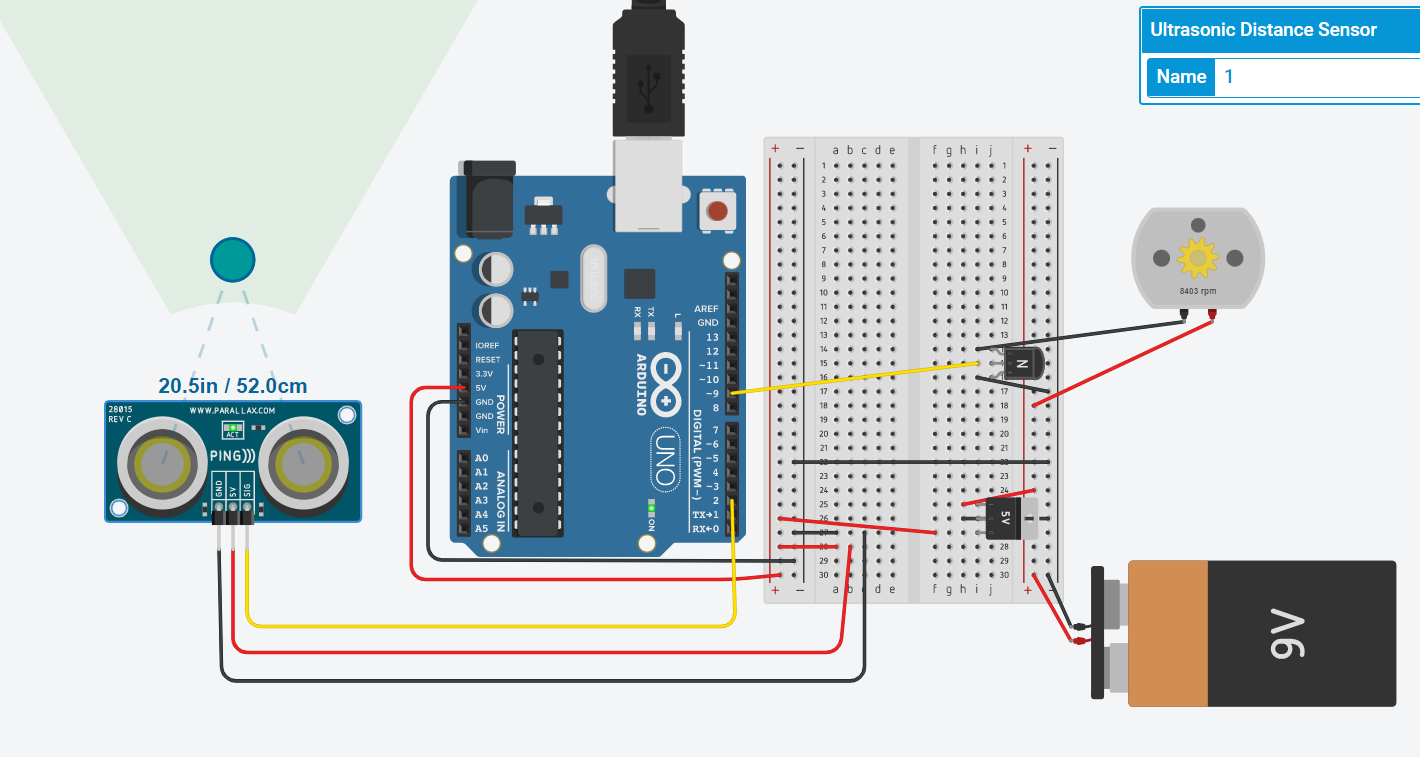
|  |  |  |  |
| --- | --- | --- | --- |
| States | Input | Output | Next State |
| Input Target Position | Target Position, Not Arrived | Save target position  Provide robot relative direction to target | Obstacle |
| Obstacle | Sensor Data | Yes  No | Obstacle Direction  Point to Target Position and Go 1 second |
| Obstacle Direction | Sensor Data & Obstacle | Front, Left, and Right  Front and Left, Left  Front, Right, Front and Right | Turn 180 Degrees and Go Forward  Go Back and Turn Right  Go Back and Turn Left |
| Turn 180 Degrees and Go Forward | Front, Left, and Right | Power Motor | Calculate and Save Direction and Distance to target |
| Go Back and Turn Right | Front and Left, Left | Power Motor | Calculate and Save Direction and Distance to target |
| Go Back and Turn Left | Front, Right, Front and Right | Power Motor | Calculate and Save Direction and Distance to target |
| Point to Target Position and Go | No Obstacle | Power Motor | Calculate and Save Direction and Distance to target |
| Calculate and Save Direction and Distance to target | Moving Direction and Distance | Calculate and Save Direction and Distance for comparison | Arrived |
| Arrived | Robot location and target position | Compare:  Yes  No | Stop  Input Target Position |

**Appendix B: Tinkercad Simulation**

**Shy Kid**



*The object is 91.6 inches away from the sensor. The motor is spinning at 2200 RPM*



*The object is 20.5 inches away from the sensor. The motor is spinning at 8400 rpm.*

**Shy Kid TinkerCad Code:**

int distanceThreshold = 0;

int cm = 0;

int inches = 0;

void setup()

{

pinMode(9, OUTPUT);

}

long readUltrasonicDistance(int triggerPin)

{

pinMode(triggerPin, OUTPUT); // Clear the trigger

digitalWrite(triggerPin, LOW);

delayMicroseconds(2);

// Sets the trigger pin to HIGH state for 10 microseconds

digitalWrite(triggerPin, HIGH);

delayMicroseconds(10);

digitalWrite(triggerPin, LOW);

pinMode(triggerPin, INPUT);

// Reads the echo pin, and returns the sound wave travel time in microseconds

return pulseIn(triggerPin, HIGH);

}

void setmotorspeed(int speed){

analogWrite(9,speed);

}

void loop()

{

// set threshold distance to activate LEDs

distanceThreshold = 300;

// measure the ping time in cm

cm = 0.01723 \* readUltrasonicDistance(2);

// convert to inches by dividing by 2.54

inches = (cm / 2.54);

Serial.print(cm);

Serial.print("cm, ");

Serial.print(inches);

Serial.println("in");

if (cm > distanceThreshold) {

setmotorspeed(0);

}

if (cm <= distanceThreshold && cm > distanceThreshold - 100) {

setmotorspeed((300-cm)/2);

}

if (cm <= distanceThreshold - 100 && cm > distanceThreshold - 250) {

setmotorspeed((300-cm)/2);

}

if (cm <= distanceThreshold - 250 && cm > distanceThreshold - 350) {

setmotorspeed(255);

}

if (cm <= distanceThreshold - 300) {

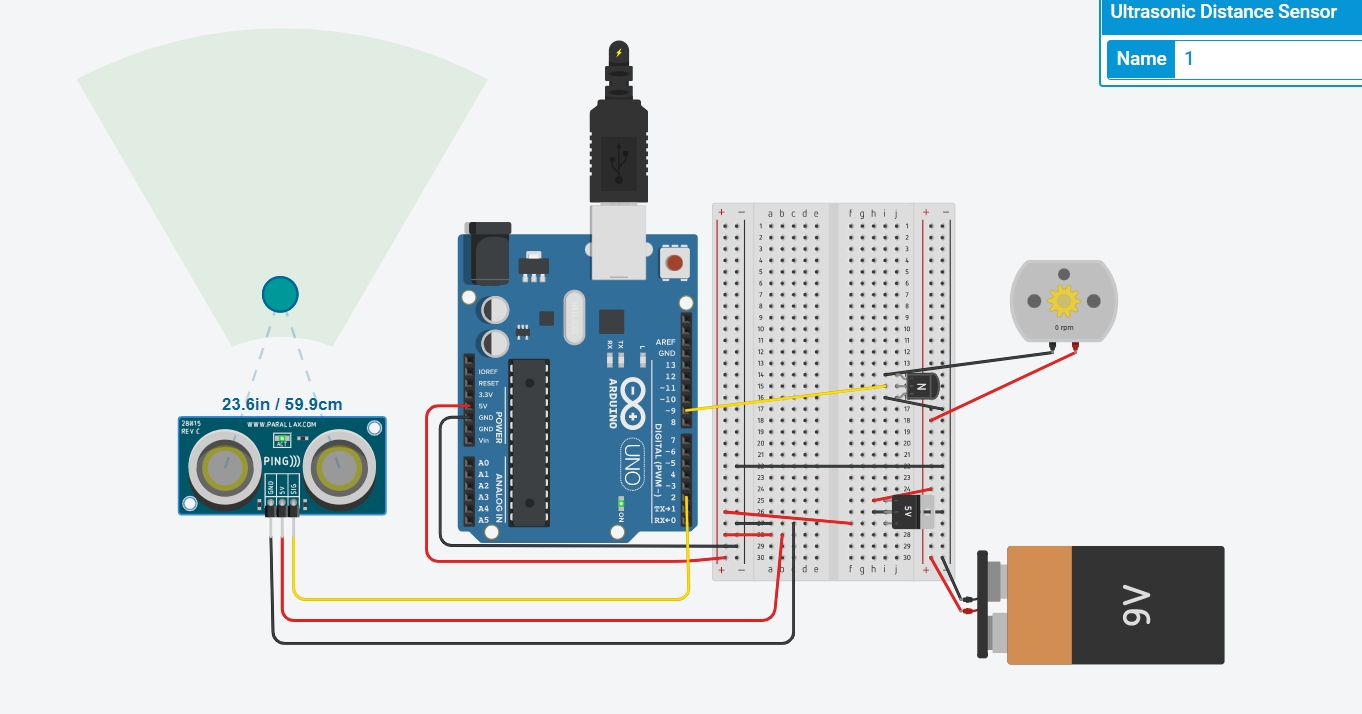
setmotorspeed(255);

}

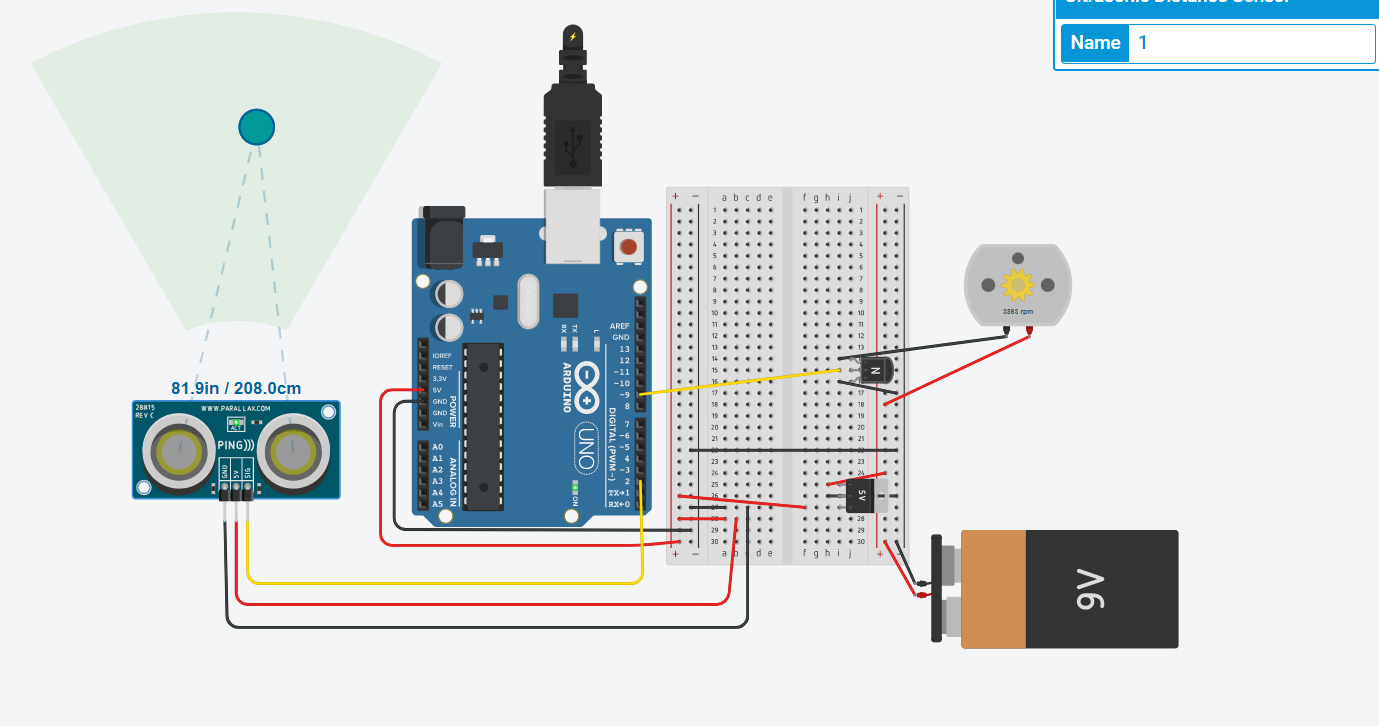
delay(100);

}

**Aggressive Kid**

**

*The object is 23.6 inches away from the sensor. The motor is spinning at 0 rpm.*



*The object is 81.9 inches away from the sensor. The motor is spinning at 3800 rpm.*

**Aggressive Kid TinkerCad Code:**

int distanceThreshold = 0;

int cm = 0;

int inches = 0;

void setup()

{

pinMode(9, OUTPUT);

}

long readUltrasonicDistance(int triggerPin)

{

pinMode(triggerPin, OUTPUT); // Clear the trigger

digitalWrite(triggerPin, LOW);

delayMicroseconds(2);

// Sets the trigger pin to HIGH state for 10 microseconds

digitalWrite(triggerPin, HIGH);

delayMicroseconds(10);

digitalWrite(triggerPin, LOW);

pinMode(triggerPin, INPUT);

// Reads the echo pin, and returns the sound wave travel time in microseconds

return pulseIn(triggerPin, HIGH);

}

void setmotorspeed(int speed){

analogWrite(9,speed);

}

void loop()

{

// set threshold distance to activate LEDs

distanceThreshold = 350;

// measure the ping time in cm

cm = 0.01723 \* readUltrasonicDistance(2);

// convert to inches by dividing by 2.54

inches = (cm / 2.54);

Serial.print(cm);

Serial.print("cm, ");

Serial.print(inches);

Serial.println("in");

if (cm > distanceThreshold) {

setmotorspeed(255);

}

if (cm <= distanceThreshold && cm > distanceThreshold - 100) {

setmotorspeed((cm-100)/2);

}

if (cm <= distanceThreshold - 100 && cm > distanceThreshold - 250) {

setmotorspeed((cm-100)/2);

}

if (cm <= distanceThreshold - 250 && cm > distanceThreshold - 350) {

setmotorspeed(0);

}

if (cm <= distanceThreshold - 350) {

setmotorspeed(0);

}

delay(100);

}

**Appendix C: IR Sensor Calibration**

Reference: <https://acroname.com/articles/linearizing-sharp-ranger-data>

|  |  |  |
| --- | --- | --- |
|  |  |  |

Using the equation above, the distance is converted to Voltage. The k value is experimentally found to maximum the R-squared value to three decimal points using the guess and check method.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | |  |
|  |  | |  |
|  |  | |  |
|  |  |  |  |

The calibration equations above use the linearized data to convert Voltage, or Analog Output, to a Distance.

**Left and Right IR Sensor Calibration:**

The *m’* value is 2500 and the *b’* value is -32.25 for the Left and Right IR sensors. The resulting calibration equation is

|  |  |  |
| --- | --- | --- |
| Distance (IN) | Analog Output (Left,Right) | Linearized Voltage (k=1.8) |
| ~~1~~ | ~~593~~ | *Too Close* |
| 2 | 670 | 0.263158 |
| 3 | 560 | 0.208333 |
| 4 | 449 | 0.172414 |
| 5 | 388 | 0.147059 |
| 6 | 343 | 0.128205 |
| 7 | 308 | 0.113636 |
| 8 | 280 | 0.102041 |
| 9 | 255 | 0.092593 |
| 10 | 240 | 0.084746 |
| 11 | 223 | 0.078125 |
| 12 | 207 | 0.072464 |
| 13 | 196 | 0.067568 |
| 14 | 187 | 0.063291 |
| 15 | 180 | 0.059524 |
| 16 | 170 | 0.05618 |
| 17 | 164 | 0.053191 |
| 18 | 157 | 0.050505 |
| 19 | 153 | 0.048077 |
| 20 | 144 | 0.045872 |

**Front and Back IR Sensor Calibration:**

The *m’* value is 2000 and the *b’* value is 50 for the Left and Right IR sensors. The resulting calibration equation is

|  |  |  |
| --- | --- | --- |
| Distance (IN) | Analog Output (Left,Right) | Linearized Voltage (k=1.5) |
| 1 | 667 | 0.4 |
| 2 | 512 | 0.285714 |
| 3 | 368 | 0.222222 |
| 4 | 272 | 0.181818 |
| 5 | 225 | 0.153846 |
| 6 | 190 | 0.133333 |
| 7 | 165 | 0.117647 |
| 8 | 138 | 0.105263 |
| 9 | 119 | 0.095238 |
| 10 | 107 | 0.086957 |
| 11 | 96 | 0.08 |
| 12 | 87 | 0.074074 |
| 13 | 79 | 0.068966 |
| 14 | 73 | 0.064516 |
| 15 | 68 | 0.060606 |
| 16 | 60 | 0.057143 |
| 17 | 57 | 0.054054 |
| 18 | 56 | 0.051282 |
| 19 | 55 | 0.04878 |
| 20 | 54 | 0.046512 |

1. https://acroname.com/articles/linearizing-sharp-ranger-data [↑](#footnote-ref-1)