Experiment 11 – Locating Multiple Targets with IR Sensor

Source code written by Dylan Vo and Charlie Escude.

The first scanning task of the robot is to locate the targets, obstacles, and mothership within the playing field. This is done by rotating the robot with the varsIntTurn(,) function and simultaneously reading the IR sensor. Detection levels are set with the threshold variable.

Based on code developed in Experiment 10, the object detection is based on detecting the leading and trailing edges of the intervals when the sensor spot is on the object. During the interval while the spot is on the object, the sensor reading is above the detection level threshold.

The program below tests the sensor level against the threshold and compares the results to the prior "laststate" results. If the last state was off, or below threshold, and the current state is on, or above threshold, a leading edge is detected and the location is recorded in the firstedge[] array. If the last state was on and the current state is off, the trailing edge is detected and the location is recorded in the secondedge[] array. When the trailing edge is detected the width of the object is calculated and compared to a minimum value. If the object is rejected the detection process is simply reset and the object count is not updated.This process efficiently rejects noise from the sensor and eliminates spurious object detection. Once an object is detected and validated the object counter, "objectcount" is incremented for detection of the next object and the distance to the brightest spot on the object is calculated. Each time the sensor state changes the "laststate" variable is updated.

Once the turn is complete, the function outputs the number of detected objects and the width, center location, and measured distance of each target.

Following the locate_multi() function are two profiling functions used to record and output the averaged sensor readings during rotation. The user must specify the direction of the turn, the distance in degrees for the turn, and the speed of the turn.

Procedure

Locating multiple targets

1. Copy locate_multi() function code.
    a. Reconcile motor bytes in varsIntTurn call
    b. Place global variables above setup()
    c. Compile
2. Setup targets, obstacles, and mothership
    a. 5 – 6 obstacles
    b. 1 – 2 targets
    c. Mothership
    d. 24" – 60" from robot
    e. 90 degree spread
3. Upload and run program
    a. Upload
    b. Turn on serial monitor
    c. Turn on robot

4. Verify results
    a. Verify # of objects
    b. Read widths and differentiate mothership

Scanning and recording

1. Copy profile functions
2. Compile profile functions
3. Upload and run program
    a. Point robot
    b. Upload code
    c. Turn on serial monitor
    d. Turn on robot
4. Copy data to Excel
    a. Select serial monitor window with mouse
    b. Ctrl-a
    c. Ctrl-c
    d. Open Excel workbook
    e. Select insertion point
    f. Ctrl-v
5. Graph data

Source code

```
// Global Variables – place above setup()
const int objcnt = 100;
long firstEdge[objcnt], secondEdge[objcnt], width[objcnt], center[objcnt], distToTarget[objcnt];
int objectcount;

// Function locate_multi
// scan and locate objects within range
void locate_multi(){
  float centerAngle;
  bool laststate = false;
  long sum = 0;
  long threshold = 450;
  long maxDist = 0;
  int i;
  objectcount = 0;
// Turn on interrupt driven motion
varsIntTurn(rotl, 90, 500, 1.0, right_motors, left_motors); // default is 90 degrees
long a = analogRead(A4);
if(a < threshold)
  laststate = false;
// While robot is turning, read sensors and test data
while(steps > 0){
  sum = 0;
  for(i = 0; i < 16; i++){     // sum 16 sensor readings
    sum += analogRead(A4);
  }
  a = sum/i;                      // average sensor readings
  if(laststate && a > maxDist)
    maxDist = a;
  if(a > threshold && !laststate)  // look for leading edges
{
  firstEdge[objectcount] = steps;
  laststate = true;
}
if(a < threshold && laststate){    // look for trailing edges
  secondEdge[objectcount] = steps;
  center[objectcount] = (firstEdge[objectcount] + secondEdge[objectcount]) / 2;
  width[objectcount] = (firstEdge[objectcount] - secondEdge[objectcount]);
  laststate = false;
  if(width[objectcount] > 20){      // debounce noise based on width criteria
    distToTarget[objectcount] = calScaleLong * pow(maxDist, calPowerLong);
    objectcount++;
    maxDist = 0;
    }
  }
}
Serial.println(objectcount);
for( i = 0; i < objectcount; i++){
  Serial.print(width[i]);
  Serial.print("\t");
  Serial.print(center[i]/steps_per_degree);
  Serial.print("\t");
  Serial.println(distToTarget[i]);
  }
}


// Utility functions
// Object Profiling function using long-range IR sensor
void profileLong(byte dir, float deg,int del){
  const int sizeArray = 2000;
  int scanArray[sizeArray];
  int i = 0;
  int j;
  long sum;
  varsIntTurn(dir, deg, del, 1.0, rightmotors, leftmotors);
  while(steps > 0){
    sum = 0;
    for (j = 0; j < 16; j++){
      sum += analogRead(A4);
    }
    if (i < sizeArray)  scanArray[i++] = sum/j;
  }
  for (int j = 0; j < i; j++){
    Serial.println(scanArray[j]);
  }
}


// Object Profiling function using mid-range IR sensor
profileMid(byte dir, float deg,int del){
  const int sizeArray = 2000;
  int scanArray[sizeArray];
  int i = 0;
  int j;
  long sum;
  varsIntTurn(dir, deg, del, 1.0, rightmotors, leftmotors);
  while(steps > 0){
    sum = 0;
    for (j = 0; j < 16; j++){
      sum += analogRead(A0);
    }
    if (i < sizeArray)  scanArray[i++] = sum/j;
  }
  for (int j = 0; j < i; j++){
    Serial.println(scanArray[j]);
  }
}
```