Tyler Thompson

2/10/2023

tjt5498@psu.edu

**Lab 9**

**Objective**

Have the robot perform a square and record the results. The speed will be changed throughout the tests to see how it affects the code and results. A measurement will be taken to record how far it was from the start as well as the angle of the end position. A few calculations will be need to be done in order to get the distance required to move the robot which in this case will be the amount of wheel rotations. The materials for this lab include the encoders that were added to the robot.

**Results**

|  |  |
| --- | --- |
|  | Here we can see the encoders successfully installed onto the robot. The hardware works properly. The code written will work efficiently with the hardware to allow for accurate results. The encoders can be seen fitted properly next to the motors. |

|  |  |
| --- | --- |
| Robot moving in square at 250 speed  <https://share.icloud.com/photos/0b1SgvPLOnSsac9nTaKdJIkHg> | Here the robot can be seen completing the square at 250 speed. The rotations are still the same for each test case just the speed is changed. The robot is shown completing the square while being only 2 inches off the starting point which is rather impressive. The angle was rather wide on this test and ended up being 6 degrees off the initial target that was intended. This test was rather good for a first test and more tests will conclude that the code is well written but the amount of sliding with the wheels may make each test slightly moved from the initial spot. The speed being fast also has a lot to do with this happening. |
| Robot moving in square at 150 speed  <https://share.icloud.com/photos/0cbRfMANTKTTwjLOSG9VbCl4Q> | This test shows the robot moving at 150 speed and completing the square. The rotations are still the same for each test only the speed is changed. The robot was only 1 inch away from the final target which is promising considering it is the same code. The angle is also only 3 degrees moved from the initial point. This shows how much the speed has to do with an accurate test since each surface the robot is on may alter the way the robot behaves. The slower the robot the less likely inconsistencies can happen. This test shows that the code works and allows the robot to perform the action. |

|  |  |
| --- | --- |
| The 3 test cases with 250 speed  A picture containing floor, building  Description automatically generated | The first test case for 250 speed included being 2 inches off the initial spot. The angle was 6 degrees from the target. The next test was only 1 inch from the initial spot and the angle was 4 degrees off the spot. This was the best test case. The final test was 1.4 inches of the initial spot and the angle was 2 degrees off. This was also a really good test except t was just slightly further. Overall the average for the test cases was about 1.4 inches from the initial target and the average angle was about 4 degrees away from the initial spot. These tests were really accurate but showed some inconsistency due to the speed that was used. |
| The 3 test cases with 150 speed  A picture containing building, floor, lumber  Description automatically generated | The first test case for the 150 speed was the slightly off the initial point with a distance of 1.6 inches and the angle was around 3 degrees away from the initial spot. The next test case was 2.1 inches from the initial spot and had an angle of 1 degree which is the best of all the test cases. The final test was 1.7 inches off the initial spot and had an angle of 4 degrees away from the initial spot. The average distance of all the test cases was 1.8 inches and the average angle was about 3 degrees away from the initial spot. These test cases were slightly better than the other series of tests and this is because of the speed being less. This allows for less inconsistency with the test results. |

**System Block Diagram**

Right Wheel Encoder

Control Software

Arduino

Left Wheel Encoder

Motor Control H Bridge

Front Left Motor

Back Right Motor

Front Right Motor

Back Left Motor

**All Calculations Needed**

The diameter of the wheels is around 2.6 inches for each of them. This means that the circumference which is diameter multiplied by pi is around 8.1 inches and using this calculation the amount of wheel encoder counts can be used to determine how many times the wheel needs to rotate to move. The wheel encoder has 20 slots on it meaning that once the interrupt reaches 20 counts for an encoder than that wheel has turned 1 time. This means the wheel needs to turn 3 times to go about 2 feet from where it started. With the test of running around 100 encoder counters the average the robot moved was 43 inches so the data was correct. For the pivot only 1 rotation of the wheels were needed so that is 20 encoder counts. With this data being calculated the code could be easily written.

**Pseudocode**

// This function checks how many times the wheel has rotated and compares it to the value passed.

// A boolean value is returned to determine if that many rotations happened.

checkDistance(float wheelRotation) {

    // This checks to see if the rotations happens.

    if ((counter1 / diskslots) greater than wheelRotation or (counter2 / diskslots) greater than wheelRotation) {

      counter1 = 0 // Reset counter 1 so it can be called again.

      counter2 = 0 // Reset counter 2 so it can be called again.

      return true

    }

    else {

      return false

    }

}

// Turns the robot the amount of rotations.

void turnRobot(float turnRotation) {

  // Checks to see if that amount of rotations has happened.

  // If it has then the function call will stop.

  while (not checkDistance(turnRotation)) {

    move\_Right // turn the car right a fixed amount of time.

  }

}

// This is the function that allows the square to happen.

// It calls other functions to allow it to work properly.

void makeSquare(float wheelRotation) {

    int path = 0; // This value is needed to determine if the square is complete.

    while (path less than 3) { // Checks to see if the square is complete.

      if (checkDistance(wheelRotation)) { // Turn the robot.

        turn Robot

        path++; // After a turn this is increased.

      }

      else {

        go forward // Makes the robot go forward.

      }

    }

}

void setup()

{

  Serial.begin(9600);

  pinMode(BUZZ\_PIN, OUTPUT);

  digitalWrite(BUZZ\_PIN, HIGH);

  Timer1.initialize(1000000); // set timer for 1sec

  attachInterrupt(digitalPinToInterrupt (MOTOR1),

     ISR\_count1, RISING);

    // Increase counter 1 when speed sensor pin goes High

  attachInterrupt(digitalPinToInterrupt (MOTOR2),

     ISR\_count2, RISING);

   // Increase counter 2 when speed sensor pin goes High

  Make Square // Calls the function so the robot can complete the task.

  stop

**Code**

// This function checks how many times the wheel has rotated and compares it to the value passed.

// A boolean value is returned to determine if that many rotations happened.

boolean checkDistance(float wheelRotation) {

    // This checks to see if the rotations happens.

    if (float((counter1 / diskslots)) >= wheelRotation || float((counter2 / diskslots)) >= wheelRotation) {

      counter1 = 0; // Reset counter 1 so it can be called again.

      counter2 = 0; // Reset counter 2 so it can be called again.

      return true;

    }

    else {

      return false;

    }

}

// Turns the robot the amount of rotations.

void turnRobot(float turnRotation) {

  // Checks to see if that amount of rotations has happened.

  // If it has then the function call will stop.

  while (!checkDistance(turnRotation)) {

    go\_Right();

  }

}

// This is the function that allows the square to happen.

// It calls other functions to allow it to work properly.

void makeSquare(float wheelRotation) {

    int path = 0; // This value is needed to determine if the square is complete.

    while (path <= 3) { // Checks to see if the square is complete.

      if (checkDistance(wheelRotation)) { // Turn the robot.

        stop\_Stop();

        delay(100);

        turnRobot(1);

        stop\_Stop();

        delay(100);

        path++; // After a turn this is increased.

      }

      else {

        go\_Advance(); // Makes the robot go forward.

      }

    }

}

void setup()

{

  Serial.begin(9600);

  pinMode(BUZZ\_PIN, OUTPUT);

  digitalWrite(BUZZ\_PIN, HIGH);

  Timer1.initialize(1000000); // set timer for 1sec

  attachInterrupt(digitalPinToInterrupt (MOTOR1),

     ISR\_count1, RISING);

    // Increase counter 1 when speed sensor pin goes High

  attachInterrupt(digitalPinToInterrupt (MOTOR2),

     ISR\_count2, RISING);

   // Increase counter 2 when speed sensor pin goes High

  makeSquare(3); // Calls the function so the robot can complete the task.

  stop\_Stop();

}

**Conclusion**

This robot operation shows the power of the encoders that were installed. This allows the robot to not use delays which means it can move around the same distance for each time. This is really powerful and unlocks the true power of applications that can be created with the robot. The results with this application were much more consistent than when time was used to make the robot do a square. This means we can duplicate accurate results frequently and not have to worry about the calculations other than the wheel rotations when creating them. All the functions used in this application are universal and can be shared between applications so it could be beneficial to create libraries and import them when creating a new application. The data for this application was really promising and many more improvements can come with updated code.