Tyler Thompson

3/24/2023

tjt5498@psu.edu

**Lab 12**

**Objective**

Create a cruise control algorithm for the robot. The algorithm will be built using PID control. This algorithm will be tested on different surfaces to ensure that it works properly. The results will show that the robot maintains a constant speed and goes back to that constant speed regardless of how much power is needed. This will be compared to the robot not using the algorithm to demonstrate the effectiveness and use of this program. The materials needed are wheel encoders for the robot which will allow the speed to be measured by the amount of times the wheels have rotated. The timer library is imported into the program to allow time to be calculated with the use of interrupts. In addition to that the list library is also used to allow the data to be held in a list for more efficient storage.

**Results**

|  |  |
| --- | --- |
| Picture of the robot  A picture containing wall, indoor  Description automatically generated | This is the robot properly set up with all required equipment. |

|  |  |
| --- | --- |
| Surfaces used for the algorithm  A picture containing building, floor  Description automatically generated | Here it is shown that the robot will need to traverse from a thick surface all the way to a smooth surface. The robot will need to try and return to the speed that it was set at when encountering the types of surfaces in this demonstration. |

**Algorithm Used**

|  |  |
| --- | --- |
| Robot decreasing speed  <https://share.icloud.com/photos/09eMCJ3CxGOm3af3Kh0Fz1L5w> | The robot was set to start out at a high speed so that the algorithm can put it back to the speed that was intended. After the speed was reached it was maintained. |
| Robot increasing speed  <https://share.icloud.com/photos/001rVE2lJT6kbSOrrfJQZnOvg> | The robot was set to start out at a low speed so that the algorithm can put it back to the speed that was intended. After the speed was reached. it is maintained. |
| The surface trial and the algorithm maintaining a constant speed  <https://share.icloud.com/photos/0d1-xV1-mARg05d1va7f6he_w> | The robot starts in a rough surface where it has a lot of power initially. The algorithm detects that the robot must go slower so the motor power is adjusted. The robot then reaches the new surface where it is much easier for the robot to move so the robot is moving faster than intended. The algorithm then corrects this and allows the robot to maintain the constant speed that was set. |

**Data With Algorithm**

Chart, line chart, scatter chart

Description automatically generated

We can see from the data that the algorithm worked perfectly. The average RPM was 340 when it adjusted back to the normal speed. We can see that when the other surface was encountered, the algorithm made a smooth transition to fix the speed in this case.

**Algorithm Not Used**

|  |  |
| --- | --- |
| Robot moving with not algorithm  <https://share.icloud.com/photos/0ce3tOaASQba78JMu3HLX5wYw> | Here we can see the robot start out moving at a slow speed with no algorithm. The robot struggles to move on the thick surface even though it should be moving a lot faster. The robot then encounters the next surface and becomes a lot faster. There is 2 distinct different speeds in this test since the robot will not try and correct how fast it is moving. The power is set as the same in both motors. |

**Data Algorithm Not Used**

**Chart, line chart, scatter chart

Description automatically generated**

In the case of the algorithm not being used we can see the robot has 2 distinct speeds that it travels in. That speed depends on the surface and it is shown that the RPM is much slower when it is on the thick surface. When the robot encounters the smooth surface it easily becomes faster.

**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

**Pseudocode**

// This takes in the current speed and the speed the robot should be going.

CorrectSpeed(inputSpeed, robotSpeed) {

speedToCorrect = robotSpeed // This is the speed the robot should be going  
 if (input Speed is a high percentage above the speedToCorrect) {  
 more correction to slow down motors

}

else if (inputSpeed is a medium percentage above the speedToCorrect) {  
 medium correction to slow down motors

}

else if (inputSpeed is a low percentage above the speedToCorrect) {  
 less correction to slow down motors

}

else if (inputSpeed is a high percentage below the speedToCorrect) {  
 more correction to speed up motors

}

else if (inputSpeed is a medium percentage below the speedToCorrect) {  
 medium correction to speed up motors

}

else if (inputSpeed is a low percentage below the speedToCorrect) {  
 less correction to speed up motors

}

else {

maintain current motor power

}

return new motor power to correct for the main program

**Code**

// This is the function that enables the cruise control. It also adds data to console.

void cruiseControl()

{

  Timer1.detachInterrupt();  // Stop the timer

  go\_Advance(currentPower); // Maintain a constant speed.

  float checkSpeed = (counter1 / diskslots) \* 240; // This is because the robot is check 4 times in 1s in this case.

  float errorCorrection = checkSpeed / sameSpeed; // Determines the amount of error that needs to be fixed.

  if (printTime >= 6) { // This keeps track of the amount of time data has printed to the list.

    printTime = 0;

  }

  if (checkSpeed > 100 && printTime == 5) { // This means data is ready to be added to the list.

  speedData.add(checkSpeed);

  }

  // This is the algorithm that fixes the speed and keeps it constant.

  if (errorCorrection <= 0.5) { // This makes it increase a lot.

    currentPower = currentPower + 6;

    go\_Advance(currentPower);

  }

  else if (errorCorrection <= 0.85) { // This makes it increase slightly.

    currentPower = currentPower + 5;

    go\_Advance(currentPower);

  }

  else if (errorCorrection <= 0.9) { // This makes it increase slightly.

    currentPower = currentPower + 4;

    go\_Advance(currentPower);

  }

  else if (errorCorrection <= 0.95) { // This is a very minor increase.

    currentPower = currentPower + 3;

    go\_Advance(currentPower);

  }

  else if (errorCorrection <= 0.97) { // This is a very minor increase.

    currentPower = currentPower + 2;

    go\_Advance(currentPower);

  }

  else if (errorCorrection <= 1.0) { // This keeps the speed the same.

    go\_Advance(currentPower);

  }

  else if (errorCorrection <= 1.03) { // This keeps the speed the same.

    go\_Advance(currentPower);

  }

  else if (errorCorrection <= 1.05) { // This slightly decreases the speed.

    currentPower = currentPower - 3;

    go\_Advance(currentPower);

  }

  else if (errorCorrection <= 1.1) { // This slightly decreases the speed.

    currentPower = currentPower - 4;

    go\_Advance(currentPower);

  }

  else if (errorCorrection <= 1.5) { // This makes it decrease a lot.

    currentPower = currentPower - 6;

    go\_Advance(currentPower);

  }

  counter1 = 0;  //  reset counter to zero

  counter2 = 0;  //  reset counter to zero

  checkTime = checkTime + 1;

  printTime = printTime + 1;

  Timer1.attachInterrupt( cruiseControl );  // Enable the timer

}

void setup()

{

  Serial.begin(9600);

  pinMode(BUZZ\_PIN, OUTPUT);

  digitalWrite(BUZZ\_PIN, HIGH);

  Timer1.initialize(250000);

  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

   Timer1.attachInterrupt(cruiseControl);

     // Enable the timer  (this supports serial print for testing)

}

void loop()

{

  // This prints the data to the console after the robot has gathered the results.

  if (checkTime > 200 && !checked) {

    Serial.print(speedData.getSize());

    Serial.print(" ");

    for (int i = 0; i < speedData.getSize(); ++i) {

      Serial.print(speedData.getValue(i));

      Serial.print(" ");

    }

    checked = true;

  }

}

**Conclusion**

This algorithm truly demonstrates the power of PID in programming. The tests concluded that the algorithm did work and the data was very conclusive. Without the algorithm the robot has 2 different speeds where as with the algorithm the speed deviated away just slightly which was when the new surface was encountered and then the original speed was maintained. Algorithms are extremely important in the real world and have so many practical uses.

This is just a simple example of this PID control and depending on what needs to be solved a different control algorithm might be needed. The combination of multiple algorithms allows for the creation of powerful software and robots that can perform incredible tasks. In the real world this algorithm can be replicated on a more practical scale and still provide great results.