Using encoders to make turns

From ROBOTC API Guide
< Tutorials | Arduino Projects/Mobile Robotics/BoeBot
Arduino → Arduino Tutorials and Guided Projects → Parallax BoeBot + Arduino Shield, Mobile Robotics Platform → Using encoders to make turns

Contents

Another useful way to utilize an encoder is to use it to turn your robot a certain number of degrees. We have previously turned the robot by using time-based coding:

```
motor[leftServo] = -20;
motor[rightServo] = 20;

wait1Msec(500);

motor[leftServo] = 0;
motor[rightServo] = 0;
```

The problem with time based movement is that the distance traveled per second is directly tied to motor power, while motor power is directly tied to the remaining battery level. Therefore, time based movement is tied to the remaining battery level and will yield adverse results from a newly-charged battery and battery that has been drained. While the difference may be small in some cases, it is a source of error that can prevent accurate movement; this can be accounted for, however, by using an encoder.

Three ways of using encoders to make turns

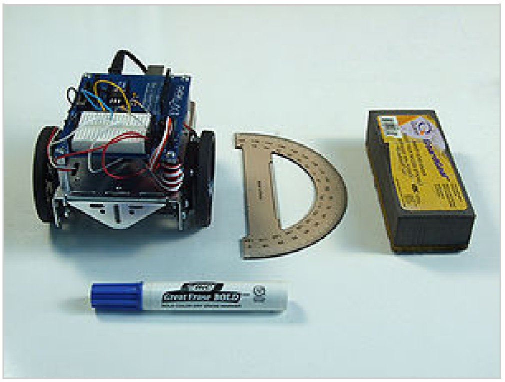
Now that the robot is equipped with encoders, we have a number of options regarding how to perform turns.

1. We can turn the robot until one wheel's encoder has ticked far enough, and hope the other wheel has gone the same distance. If you notice that your wheels tend to go roughly the same speed (A good indication of this is how straight your robot went without the benefit of proportional control) then this might be the simplest option. However, many motors have manufacturing tolerances which means the motors move at slightly different speeds when the same power is applied.
2. We can control each wheel independently throughout the turn - that is, we separately monitor each wheel for when its encoder has passed the tick point. This will result in a robot that has turned exactly the correct angle, however if the robots have slightly different speeds, one wheel will be turning after the other has stopped, leading to a slight change in position of the robot.
3. We can use *proportional control*. This is a complex method and since the BoeBot's servos are very accurate, we won't use this in our application.

For this lesson, we are going to choose option **2** as it is best suited for our purposes.

The Test

Just like in the previous lesson where we found the ratio of encoder ticks to inches traveled, we are going to find the ratio of encoder ticks to degrees turned. To do this, we will perform an experiment of sorts. It is possible to determine this ratio using trigonometry; however, the required calculations are beyond the scope of this lesson. Since our robot will always be the same size an experimental method is both easier and simpler.


The equipment needed

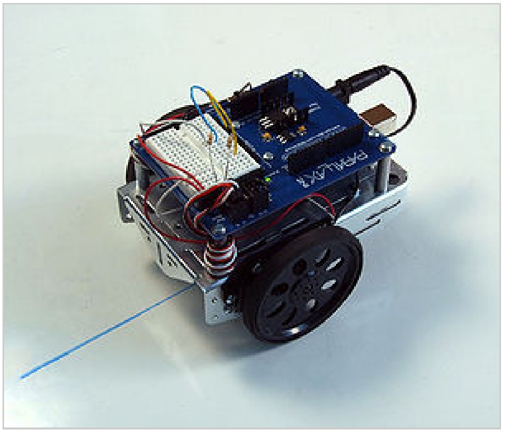
The Program

The first thing we need to do is write a program which will make the robot turn through a preset number of encoder ticks. We will turn for 8 encoder ticks to get enough motion to negate slop but not enough to be too large for the protractor. We will also control each motor independently so that we will have the most accurate turn.

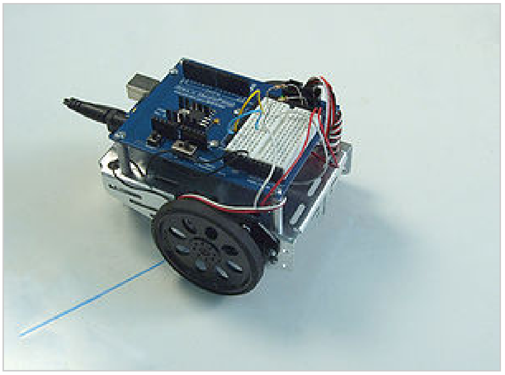
```
//used to store the count of the motor clicks
int leftEncoderCount = 0;
int rightEncoderCount = 0;

//a background task used to monitor the encoders and count the clicks
task encoderTask()
{
  // Get the state of the encoders so that we know when they have changed
  // for example if the last state was high and the current state is low
  // or vise-versa, then we know that the encoder incremented another click.
  int lastStateLeft = SensorValue[leftEncoder];
  int lastStateRight = SensorValue[rightEncoder];

  while(true)
  {

    //get the current state of the left encoder
    int state = SensorValue[leftEncoder];

    //has the left encoder changed states
    if (lastStateLeft != state) {
      //if so, then increment the leftEncoderCount by 1
      leftEncoderCount++;

      // then store the current state as the last state so that we can keep monitoring every change
      lastStateLeft = state;
    }

    //repeat for the right encoder.

    state = SensorValue[rightEncoder];
    if (lastStateRight != state) {
      rightEncoderCount++;
      lastStateRight = state;
    }

    //we add a pause between updates to keep from hogging the CPU and
    // so that we debounce the encoders.
    wait1Msec(10);
  }
}

task main()
{
  //Start the encoder monitoring task, which will run in the background.
  StartTask(encoderTask);

  //Reset encoders
  leftEncoderCount  = 0;
  rightEncoderCount  = 0;

  //Perform a point turn to the left. We will use lower power values for more accuracy.
  motor[leftServo] = -20;
  motor[rightServo] = 20;
  //Since the wheels may go at slightly different speeds due to manufacturing tolerances, etc.,
  //we need to test both encoders and control both motors separately. This may result in one motor
  //going for longer than another but it will ultimately result in a much more accurate turn.
  while(rightEncoderCount < 8 || leftEncoderCount < 8)
  {

    if(rightEncoderCount > 8)
      {
        motor[rightServo] = 0;
      }

    if(leftEncoderCount] > 8)
      {
        motor[leftServo] = 0;
      }
   }
}
```

The Experiment

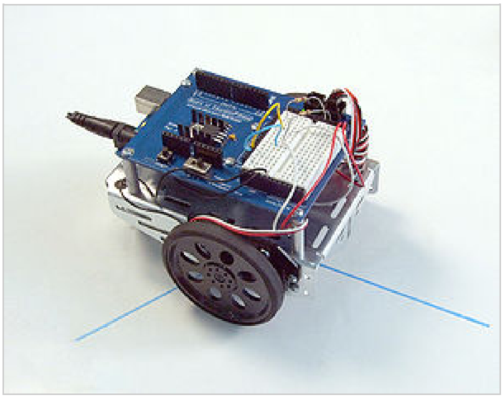
To use this code, the first thing we need to do is place the robot on a markable surface (e.g. whiteboard, a large sheet of paper, etc.) and mark a line perpendicular to the robot, straight through its center, as seen in the below image:


Draw a perpendicular line

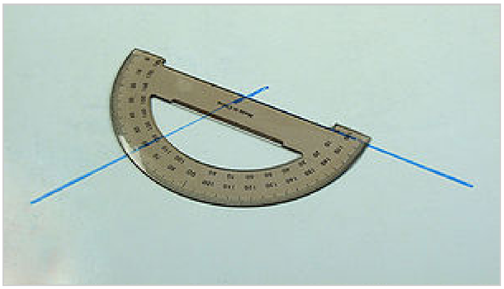
Before you run the program, position both wheels so that each encoder is exactly over a slot. This will help synchronize the encoders as we want as precise an experiment as possible. Now, run the program.


Run the program

Next, mark another perpendicular line in the same manner as you did before, only relative to the robot's new position.


Draw a second line

Finally, use a protractor to measure the angle formed between these two lines, and that will determine the angle through which the robot turned.


Measure the angle formed

The Results

Our tests resulted in an angle of around **119** degrees. It is highly recommended that you perform the test a few times for accuracy purposes.

Writing a function

Now that we know the angle our robot turned through with 8 encoder ticks, it will be simple to find the ratio of encoder ticks to degrees turned.

If the robot turned **119** degrees in **8** encoder ticks, then the ratio of encoder ticks to degrees turned must be

8 / 119 ≈ 0.067 encoder ticks per degree.

We can multiply this ratio by the desired amount of degrees we wish to turn, and that will output the amount of encoder ticks needed. Since the Arduino UNO with RobotC firmware cannot handle floating-point (decimal) operations, we will express this as:

***Encoder ticks needed* = (8 * *degrees to turn*) / 119**.

This is exactly the same operation however we multiply to get as large a number as possible before we complete a division. Keep in mind we are bound to lose an amount of accuracy because of the lack of floating-point division, and the low resolution of the encoders.

Now, all we need to do is convert the previously used code into a function with two parameters (degrees to turn and speed) using the formula we just calculated. We will write two functions, one for turning left, one for turning right. While it would be possible to use only one function, it can get confusing and complex, and when calling the function you will have to remember the direction of rotation. Having two functions makes it much clearer, easier, and more useful, even though there might be a few more lines of code.

```
void turnLeftDeg(int degrees, int power)
{
  //Reset encoders
  leftEncoderCount = 0;
  rightEncoderCount = 0;

  //Determine tickGoal
  int tickGoal = (8 * degrees) / 119;

  //Start the motors in a left point turn.
  motor[leftServo] = -1 * power;
  motor[rightServo] = power;

  //Since the wheels may go at slightly different speeds due to manufacturing tolerances, etc.,
  //we need to test both encoders and control both motors separately. This may result in one motor
  //going for longer than another but it will ultimately result in a much more accurate turn.
  while(rightEncoderCount < tickGoal || leftEncoderCount < tickGoal) {
    if(rightEncoderCount > tickGoal) {motor[rightServo] = 0;}
    if(leftEncoderCount > tickGoal) {motor[leftServo] = 0;}
  }
  //Make sure both motors stop at the end of the turn.
  motor[leftServo] = 0;
  motor[rightServo] = 0;
}
```

To turn right, we just change the function accordingly:

```
void turnRightDeg(int degrees, int power)
{
  //Reset encoders
  leftEncoderCount = 0;
  rightEncoderCount = 0;

  //Determine tickGoal
  int tickGoal = (8 * degrees) / 119;

  //Start the motors in a left point turn.
  motor[leftServo] = power;
  motor[rightServo] = -1 * power;

  //Since the wheels may go at slightly different speeds due to manufacturing tolerances, etc.,
  //we need to test both encoders and control both motors separately. This may result in one motor
  //going for longer than another but it will ultimately result in a much more accurate turn.
  while(rightEncoderCount < tickGoal || leftEncoderCount < tickGoal) {
    if(rightEncoderCount > tickGoal) {motor[rightServo] = 0;}
    if(leftEncoderCount > tickGoal) {motor[leftServo] = 0;}
  }
  //Make sure both motors stop at the end of the turn.
  motor[leftServo] = 0;
  motor[rightServo] = 0;
}
```

Test Program

Finally, we are going to test these functions by making the robot do four turns which should make it end up back at the starting position. All we need to do is put both functions in the program, then call them from within task main(). We also need to include the encoder monitoring task constructed earlier.

```
#pragma config(CircuitBoardType, typeCktBoardUNO)
#pragma config(PluginCircuitBoard, typeShieldParallaxBoeBot)
#pragma config(UART_Usage, UART0, uartSystemCommPort, baudRate200000, IOPins, dgtl1, dgtl0)
#pragma config(Sensor, dgtl2,  rightEncoder,   sensorDigitalHighImpedance)
#pragma config(Sensor, dgtl3,  leftEncoder,    sensorDigitalHighImpedance)
#pragma config(Motor,  servo_10,        leftServo,     tmotorServoContinuousRotation, openLoop, IOPins, dgtl10, None)
#pragma config(Motor,  servo_11,        rightServo,    tmotorServoContinuousRotation, openLoop, reversed, IOPins, dgtl11, None)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

//used to store the count of the motor clicks
int leftEncoderCount = 0;
int rightEncoderCount = 0;

//a background task used to monitor the encoders and count the clicks
task encoderTask()
{
  //get the state of the encoders so that we know when they have changed
  // for example if the last state was high and the current state is low
  // or vise-versa, then we know that the encoder incremented another click.
  int lastStateLeft = SensorValue[leftEncoder];
  int lastStateRight = SensorValue[rightEncoder];

  while(true)
  {

    //get the current state of the left encoder
    int state = SensorValue[leftEncoder];

    //has the left encoder changed states
    if (lastStateLeft != state) {
      //if so, then increment the leftEncoderCount by 1
      leftEncoderCount++;

      // then store the current state as the last state so that we can keep monitoring every change
      lastStateLeft = state;
    }

    //repeat for the right encoder.

    state = SensorValue[rightEncoder];
    if (lastStateRight != state) {
      rightEncoderCount++;
      lastStateRight = state;
    }

    //we add a pause between updates to keep from hogging the CPU and
    // so that we debounce the encoders.
    wait1Msec(10);
  }
}

void turnRightDeg(int degrees, int power)
{
  //Reset encoders
  leftEncoderCount = 0;
  rightEncoderCount = 0;

  //Determine tickGoal
  int tickGoal = (8 * degrees) / 119;

  //Start the motors in a left point turn.
  motor[leftServo] = power;
  motor[rightServo] = -1 * power;

  //Since the wheels may go at slightly different speeds due to manufacturing tolerances, etc.,
  //we need to test both encoders and control both motors separately. This may result in one motor
  //going for longer than another but it will ultimately result in a much more accurate turn.
  while(rightEncoderCount < tickGoal || leftEncoderCount < tickGoal) {
    if(rightEncoderCount > tickGoal) {motor[rightServo] = 0;}
    if(leftEncoderCount > tickGoal) {motor[leftServo] = 0;}
  }
  //Make sure both motors stop at the end of the turn.
  motor[leftServo] = 0;
  motor[rightServo] = 0;
}

void turnLeftDeg(int degrees, int power)
{
  //Reset encoders
  leftEncoderCount = 0;
  rightEncoderCount = 0;

  //Determine tickGoal
  int tickGoal = (8 * degrees) / 119;

  //Start the motors in a left point turn.
  motor[leftServo] = -1 * power;
  motor[rightServo] = power;

  //Since the wheels may go at slightly different speeds due to manufacturing tolerances, etc.,
  //we need to test both encoders and control both motors separately. This may result in one motor
  //going for longer than another but it will ultimately result in a much more accurate turn.
  while(rightEncoderCount < tickGoal || leftEncoderCount < tickGoal) {
    if(rightEncoderCount > tickGoal) {motor[rightServo] = 0;}
    if(leftEncoderCount > tickGoal) {motor[leftServo] = 0;}
  }
  //Make sure both motors stop at the end of the turn.
  motor[leftServo] = 0;
  motor[rightServo] = 0;
}

task main()
```

```
{
  //Start the encoder monitoring task, which will run in the background.
  StartTask(encoderTask);

  //Turn right 90 degrees with power 30
  turnRightDeg(90,30);

  //Wait a bit so the robot's momentum does not effect the next turn
  wait1Msec(500);

  turnLeftDeg(50,40);
  wait1Msec(500);
  turnRightDeg(160,20);
  wait1Msec(500);
  turnLeftDeg(200,30);
}
```