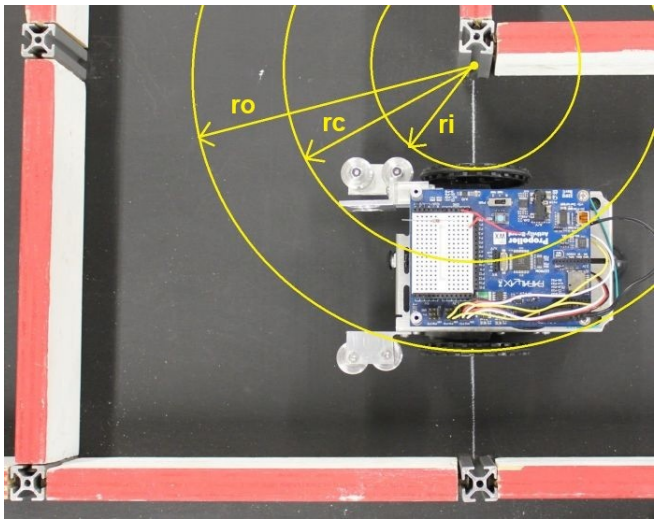
Previously you learned how to code the robot to make a turn. You have also learned how to use odometry to make the robot travel straight forward a specific distance. In this lesson you will use odometry to make the robot turn.

Figure 1: 2025 NRC Robot Maze - middle school

There are three types of turns the robot can execute: 1) zero-radius turn **2)** pivot turn **3)** arc turn. In the previous lesson on turning I suggested that the arc turn is the better turn for solving the maze quickly and we will use this type of turn in this lesson.

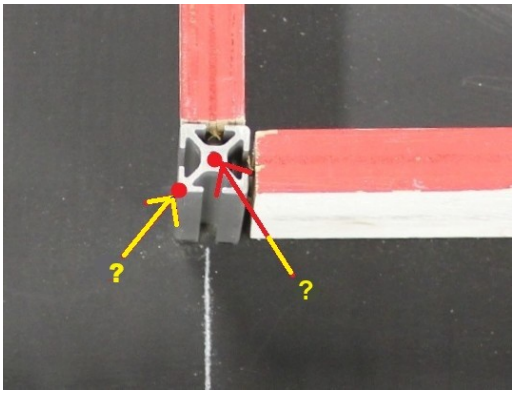
Making Arc Turns (review)

When the robot wheels are spinning at different speeds, the robot travels a curved path. We can specify the shape of the curved path by using the mathematics of circles. The size of a circle is conveniently specified as its **radius** (r), *the distance from the center of the circle to the circumference*. In the case of the robot, we are interested in three radii, **ri,** the radius to the inner wheel, **rc**, the radius to the center of the robot and **ro**, the radius to the outer wheel. The distance between the robot wheels is about 10.2 cm. From this we can derive all three radii for a turn if we know only one radius.

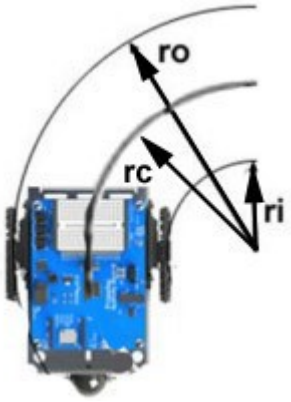


Where should the center of the circle be?

Two candidates for the circle center point are **1)** corner of the wall post or **2)** center of the wall post. I would argue that the center of the wall post is better. This allows for a larger radius turn where the speed of the inner wheel is slightly greater than with a shorter radius turn. In addition, the turn should then start earlier, at the center of the corner post (white line) rather than at the corner of the post.



Let us take the third turn in the maze as an example, a **90 degree turn to the right**. Suppose we want the robot centered in the lane at the beginning of the turn and to end up centered in the intersecting lane at the end of the turn. Then we can set rc as half the width of the lanes. The lanes are 25.4 cm wide (including half the wall thickness on each side). Then rc = 25.4 / 2 = 12.7 cm. We need to calculate ro and ri:



ro = rc + ½ 10.2 cm = 17.8 cm
ri = rc – ½ 10.2 cm = 7.6 cm

Now calculate the circumference of the circle with the radius of the outer wheel, ro.

2 x ro x 3.14 = 112 cm

The outer wheel must travel 112 cm around the circle of radius 17.8 cm to complete one revolution (360 degrees). A 90 degree turn is one quarter of a full circle:

112 cm / 4 = 28 cm

The outer wheel must travel 28 cm to make a 90 degree turn with a radius of 17.8 cm. We can use odometry to make the outer wheel travel 28 cm. However, we also need to make the robot turn with a radius of 17.8 cm by specifying different speeds for the left and right wheels. Suppose we want the robot to turn right. Then the left wheel must turn faster than the right wheel. Also suppose we want the left (outer wheel) to have a speed of 64 (half maximum speed). Then we need to calculate the speed required for the right (inner wheel).
The difference is a ratio of the two radii:

ri / ro = ratio of inner wheel
7.6 cm / 17.8 cm = 0.427

Multiply the ratio by speed of outer wheel to calculate speed of inner wheel:

0.427 x 64 ticks/sec = 27 ticks/sec

We will use the function **drive_getTicks()** to make the left wheel travel a specific distance along the curved path of the turn. Keep in mind that the robot odometer keeps a running total of the distance traveled by each wheel from the point at which the robot first moves. We want to track the number of ticks the left wheel moves starting at the **beginning** of the turn. In order to do this it is necessary to take an odometer reading at the beginning of the turn and store the value in a variable that will mark the beginning point of the turn. Then as the robot turns, do periodic odometer readings which will represent the total distance traveled. Subtract the odometer reading at the beginning of the turn from the total distance to get the number of ticks traveled since the beginning of the turn. When that result yields 86 ticks (280 mm/3.25 mm per tick) the turn has completed and the robot should then execute the next required move to continue through the maze. The full code is on the next page.

Code for a right turn by odometry, adjusted by testing in the maze

```
#include "simpletools.h"

#include "abdrive360.h"


int left = 0;           //variable to store ticks traveled by left wheel
int right = 0;          //variable to store ticks traveled by right wheel
int prevTicks = 0;      //variable to store ticks at start of a movement
int ticksTraveled = 0;  //variable for ticks traveled in a movement


int main( )
   {
   drive_getTicks(&left, &right);         //store total distance traveled
   prevTicks = left;                      //store value of left in prevTicks
   while (ticksTraveled < 75 )            // drive straight 75 ticks (about 244 mm)
      {
      drive_speed(64, 64);          // drive straight, both wheels same speed
      pause (20);                         // wait 20 milliseconds
      drive_getTicks(&left, &right);      //store total distance traveled
      ticksTraveled = left - prevTicks;   //calculate distance moved
      }
   drive_getTicks(&left, &right);         //store total distance traveled
   prevTicks = left;                      //store value of left in prevTicks
   ticksTraveled = 0;                     // reset the counter
   while(ticksTraveled < 87)              // calculated value is 86
      {
      drive_speed(64, 29);                // right calculated value is 27
      pause (20);
      drive_getTicks(&left, &right);      //store total distance traveled
      ticksTraveled = left - prevTicks;   //calculate distance moved
      }
   drive_getTicks(&left, &right);         //store total distance traveled
   prevTicks = left;                      //store value of left in prevTicks
   ticksTraveled = 0;                     // reset the counter
   while(ticksTraveled < 75)              // drive straight 75 ticks (about 244 mm)
      {
      drive_speed(64, 64);                      // drive straight
      pause (20);
      drive_getTicks(&left, &right);            //store total distance traveled
      ticksTraveled = left - prevTicks;         //calculate distance moved
      }
drive_speed(0, 0);                       //stop the robot
}
```

Discussion of code on page 3

The code causes the robot to make three movements: 1) robot moves straight forward 75 ticks (each tick is 3.25 mm), 2) robot makes a 90 degree right turn, 3) robot moves straight forward 75 ticks, then stops. By experiment I have found that coding a turn is more accurate if you first make the robot move straight forward a distance of about one maze cell, then make the turn and move straight forward again another cell distance. If you test your turning code with the robot stopped at the beginning point of the turn, then just make the turn and stop, the robot will turn more than the desired number of degrees. Why? Because the robot does not stop immediately when the program executes the **drive_speed(0, 0)** function. There is a deceleration which causes the robot to coast past the point of 90 degrees. When your robot is running the maze it should not stop before it makes each turn or stop after each turn. It should be running continuously from start to finish. Therefore, the best way to judge your turn code is by making the robot move forward before the turn and also after the turn.

Before the **main()** function there are four variables initialized with values of zero. The variable named **left** stores the odometer reading of the left wheel encoder and the variable named **right** stores the odometer reading of the right wheel encoder. Keep in mind that these odometer values are the **total** distance traveled by each wheel. We will need to measure distances other than the total distance, which necessitates that we use additional variables. The variable named **prevTicks** is used to store an odometer reading at the start of a movement, for example the start of a turn. The fourth variable named **ticksTraveled** stores the distance value the robot has made since the start of the movement.

Let's look at the first two lines inside the main() function:

```
drive_getTicks(&left, &right);
prevTicks = left;
```

The first line causes the current odometer readings to be stored in the variables **left** and **right**. The robot has not moved yet so the two odometer readings stored should have a value of zero. The second line causes the value stored in **left** to also be stored in the variable **prevTicks**. We are using the left wheel encoder for odometry because it is the wheel traveling the larger radius of turn and thus longer path yielding greater accuracy.

The next line of code is below.

```
while (ticksTraveled < 75 )
```

This loop will continue to run until the value stored in the variable **ticksTraveled** is 75 or greater. Notice that the drive code causes the robot to move straight forward `drive_speed(64, 64);` for 20 milliseconds and then takes another odometer reading `drive_getTicks(&left, &right)` and then makes a calculation of how far the robot has moved: `ticksTraveled = left - prevTicks` . Remember the value stored in **prevTicks** is

the odometer reading at the start of the move and the value in **left** represent the total odometer reading. By subtracting the value in **prevTicks** from the value in **left**, we obtain the distance traveled in the current movement. The value is stored in the variable named **ticksTraveled**.

Now we are at the bottom of the first while loop and the program jumps back to the top of the loop and checks to see if the value stored in **ticksTraveled** is less than 75. If it is, then the loop continues to execute, doing each of the steps I have outlined above. When the value stored in **ticksTraveled** is 75 or more, then the while loop terminates and the program proceeds to the code below:

```
while(ticksTraveled < 87)                // calculated value is 86
      {
      drive_speed(64, 29);               // right calculated value is 27
```

On page 2 we calculated that the left wheel should move 86 ticks to complete the 90 degree turn. However, in testing my robot, I found that a value of 86 made the robot turn less than 90 degrees and that the value of 87 was better. We also calculated that the robot wheel speeds should be 64 and 27 but in testing I found that the robot turned with too tight a radius. In adjusting the right wheel speed I found that 29 was a better speed. In testing your robot it is likely that you will also need to adjust the values a small amount to get the most accurate turn. And if we were to test multiple robots, it is likely that the adjustments required to make an accurate turn will be slightly different. If you want all of your robots to run exactly the same, then be prepared to spend thousands of dollars for each one. In our case, GEAR selected robots for you to use that are more economical, but still provide acceptable performance.

Making the robot turn left with odometry.

In order to cause the robot to move forward, the left wheel must rotate in a counterclockwise direction while the right wheel rotates in a clockwise direction. Both wheel motors are manufactured to be the same. When rotating counterclockwise, the odometer provides positive numbers for the distance traveled while if rotating clockwise, the odometer provides negative numbers. When the robot turns left, the right wheel will be the outer wheel and we want to use its encoder. But that encoder will be providing negative numbers for distance traveled. It will be necessary to adjust the turn code to handle negative numbers. The next page provides the code for turning left and you will notice that it handles negative numbers. A discussion of the code is provided starting on page 7.

Code for a left turn by odometry, adjusted by testing in the maze

```
#include "simpletools.h"
#include "abdrive360.h"

int left = 0;                //variable to store ticks traveled by left wheel
int right = 0;               //variable to store ticks traveled by right wheel
int prevTicks =0;            //variable to store ticks at start of a movement
int ticksTraveled = 0;       //variable for ticks traveled in a movement

int main( )
{
  drive_getTicks(&left, &right);               //store odometer readings
  prevTicks = right;               // use right motor encoder as odometer

  while (ticksTraveled > -73)   //right encoder counts are negative numbers
  {
    drive_speed(64, 64);                       // drive straight
    pause (20);                                // wait 20 milliseconds
    drive_getTicks(&left, &right);             //store odometer readings
    ticksTraveled = right - prevTicks;         //using right motor encoder
  }

  drive_getTicks(&left, &right);               //store odometer readings
  prevTicks = right;                           //using right motor encoder
  ticksTraveled = 0;                           // reset the counter

  while(ticksTraveled > -91)                   // calculated to be -86
  {
     drive_speed(29, 64);                      //calculated left is 27
     pause(20);                                // wait 20 milliseconds
     drive_getTicks(&left, &right);            //store odometer readings
     ticksTraveled = right - prevTicks;        //using right motor encoder
  }


  drive_getTicks(&left, &right);               //store odometer readings
  prevTicks = right;                           // using right motor encoder
  ticksTraveled = 0;                           // reset the counter

  while(ticksTraveled > -73)
  {
    drive_speed(64, 64);                       // drive straight
    pause (20);                                // wait 20 milliseconds
    drive_getTicks(&left, &right);             //store odometer readings
    ticksTraveled = right - prevTicks;         //using right motor encoder
  }

  drive_speed(0, 0);                           // stop robot
}
```

Discussion of code on page 6

Below you will find the beginning code for a left turn. The second line causes the odometer value of the right wheel to be stored in the variable **prevTicks**. Instead of using the left wheel encoder, as we did for turning right, now we are using the right wheel encoder for turning left. Notice that all lines in the code where an encoder value is used, these are now the right wheel encoder, not the left wheel.

```
drive_getTicks(&left, &right);
prevTicks = right;
```

Below you will find the next line of code. Notice that the value inside the parentheses of the **while** is a **negative** value. This is necessary because we are using an encoder now that counts in negative numbers. Also, the **while** loop continues while the value stored in **ticksTraveled** is *greater* than -73. This seems to be the reverse of the code for a right turn. But consider how the counts accumulate from the right encoder. If we set the right encoder to a value of zero at the start of a left turn, then after the robot moves one tick forward with the right wheel, the value from the right encoder will be -1. We need the robot to keep turning until the right encoder provides a number of -73. A value of -1 is greater than a value of -73. All numbers between zero and -73 are greater than -73 (-1, -2, -3, etc.).

```
while (ticksTraveled > -73)
```

Below you will find the code for the second while loop, that makes the robot turn left. The condition inside the parentheses is of the same structure as the first while loop, accounting for the fact that we are using an encoder that provides negative numbers. The calculated value for distance traveled along the outer wheel radius should be 86 ticks (-86 ticks if we are using the right wheel encoder). However, in testing my robot, I found that -91 ticks provided a better value.

```
while(ticksTraveled > -91)                    // calculated to be -86
```

Below find the code inside the above while loop that provides the wheel speeds for the left turn. The calculated speed for the left wheel should be 27 but I found in testing a better value was 29. When you test your robot in the maze, you may find slightly different numbers that provide the best left turn.

```
drive_speed(29, 64);                    //calculated left is 27
```

File name: TurningCornerByOdometry.pdf
online location: https://www.lafavre.us/robotics/TurningCornerByOdometry.pdf
last update: 3/13/26