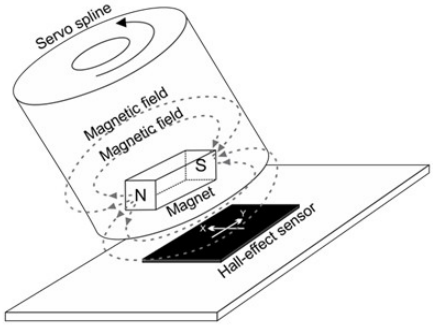
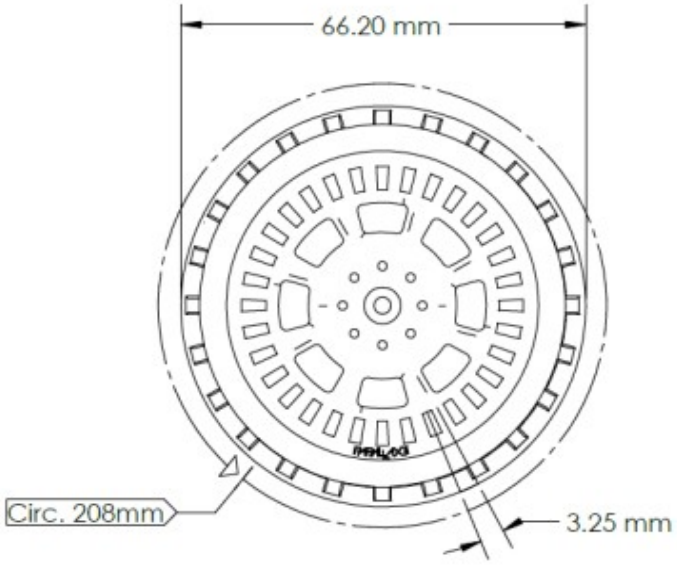
Making ActivityBot360 travel a specific distance J. La Favre March, 2026

The image to the right is the maze pattern for the 2025 NRC Robot Maze Contest, Middle School Division. Your assignment this year is to program the ActivityBot360 to solve the maze. You have already learned how to make the robot follow a wall using a Ping distance sensor, which keeps the robot centered in the lane it is traveling. You have also learned how to code for making the robot make a turn of a specific radius. In order to finish your maze-solving code, you will need to make the robot turn at specific places. This can be challenging, with many ways of coding a solution.

In these lessons you will learn two general approaches for coding WHEN to initiate a turn: **1)** odometry **2)** distance sensing (Ping sensor). In this lesson you will learn how to make the robot travel a specific distance by utilizing the odometry capabilities of the robot. If the distance traveled positions the robot where it must make a turn, then you have a solution for coding the initiation of the turn.



You are probably aware that automobiles have an instrument that tracks distance traveled. This instrument is named an *odometer.* The ActivityBot360 also has an odometer, although it is usually identified as an **encoder**. The encoder is composed of two parts, **1)** a magnet attached to the motor shaft **2)** a Hall-effect sensor located near the magnet. As the magnet spins, the Hall-effect sensor quantifies the rotation (it senses the rotation of the magnetic field of the magnet). The signal from the Hall-effect sensor is sent to the microprocessor of the robot, where software converts the signal into a distance traveled. Thus, the encoder works with software to create an odometer.



Dimensions of the ActivityBot360 wheel are provided in the image to the right. If both wheels make one rotation forward, the robot will travel a distance equivalent to the wheel circumference (208 mm or 20.8 cm). One rotation of the wheel is divided into 64 parts, each part named a **TICK**. One tick is equivalent to 5.625 degrees of wheel rotation (360 / 64). If we divide the circumference by 64 we get a value of 3.25 mm per tick. If we instruct the robot to move forward one tick, it will move forward 3.25 mm.

There are really only two kinds of drive functions available in the abdrive360.h library: **drive_speed()** and **drive_goto()**. So far you have learned how to use **drive_speed()**. This function specifies a **speed** of wheel rotation, which, combined with a **pause()** function, can be used to code for a distance to travel. However, the **drive_goto()** function provides for a more accurate distance specification.

Let us work out a solution to move the robot forward from the start of the maze to a location where it might begin to make its first turn. I will give you my measurements which I derived as follows. I placed the robot at the start position with its back touching the wall. Then I measured the distance from the center of the wheel to the end of the wall, where an opening to the left begins. The distance is about 26.5 inches (you should make your own measurement). Multiply by 2.54 to convert to centimeters (2.54 x 26.5 = 67 cm).

How do we code to make the robot move forward 67 cm (670 mm)? Recall that the robot moves forward 3.25 mm per tick. We need to convert distance in mm to distance in ticks, which we can do by dividing distance by 3.25.

670 mm / 3.25 mm/tick = 206 ticks

Here is the code to make the robot move forward 67 cm

drive_goto(206, 206);

Unfortunately, there is a problem with using the **drive_goto ()** function. We want to use wall-following code as the robot moves toward its first turn. There does not seem to be any simple way to accomplish this. While the robot is executing **drive_goto()**, wall-following code is excluded. But there is a way around this problem.

We can use another function related to the robot's odometry. That function is **drive_getTicks ()**. This function provides a way to retrieve the number of ticks the robot has traveled. You will use **drive_getTicks()** to make the robot travel a specified distance without using the **drive_goto()** function.

Using your own measurements for the distance to travel, write a code that will make the robot move forward from the start position until the center of the wheels are positioned at the place where a 90 degree right turn should be initiated. Make sure your code includes the libraries **simpletools.h** and **abdrive360.h**:

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

Then add the main() function:

```
int main()
{
   //the remainder of code goes here (see next page)
}  // end of main() function
```

In between the open and close curly { } braces, add your drive code as provided on the next page.

Use your distance measurement in ticks rather than 206 in code below..

```
int left = 0; //create variable to store number of ticks traveled by left wheel
int right = 0; //create variable to store number of ticks traveled by right wheel
while(left < 206) //run this loop as long as number of ticks traveled by
                  // left wheel is less than 206
   {
   drive_speed(64, 64); // drive forward at speed 64 both wheels
   pause(20); // wait 20 milliseconds
   drive_getTicks(&left, &right); // store # ticks traveled by wheels in
                                  // the variables left and right
   } //end of while loop
drive_speed(0, 0); // stop the robot
} // end of main() function
```

We need to examine the line **drive_getTicks (&left, &right)** in detail so that you can understand the code. The arguments contained inside the parentheses are **&left** and **&right**. The function has retrieved the number of ticks traveled by each wheel, but we need to specify WHERE to store those values. The variables where the values are to be stored are named **left** and **right**. But why is the ampersand (&) added in front of the variable name? The ampersand is used as a **pointer** in C language. In this function we need to POINT to the location where the variable is stored rather than just provide the name of the variable. In other words, the code is instructing that the values retrieved by the function should be stored in the variables located at specific locations in the microprocessor memory.

Also notice that when the variables **left** and **right** are created, values of zero (0) are assigned to each. Before the **while()** loop begins, it is necessary to ensure that the values stored in these variables are less than 206. Otherwise the loop would not run.

Load the program on your robot, place the robot at the start of the maze, then let the robot run the distance specified. Keep in mind here that your robot will not be executing any wall-following code. Therefore, it may wander close to one or the other of the walls of the lane. We won't worry about that for this exercise.

Where did your robot stop after running your code? Was it approximately where you would want it to be when it begins to turn left? If not, alter your distance traveled in the code and test again until you get the robot to travel the desired distance.

When you have the distance correct to position for a turn, add code to make a left turn of 90 degrees. After the turn is completed add code to stop the robot. Test this in the maze. Did the robot make a good left turn? If not, edit your code until your robot makes a good left turn.

The robot odometers keep a running total of how far each wheel has moved since the starting point of the maze. In developing your code to solve the maze, it is likely that you will want the robot to travel specific distances with starting points that are beyond the starting point of the maze. Let's take the example of the straight movement required after the first turn. The robot must move forward one maze cell (a cell is 254 mm long or 78 ticks) then make another left turn. We cannot use the code at the top of this page because the robot has moved beyond the maze start point and the odometer values are now not zero. You will need to create additional variables to store odometer readings at the

point in the maze where the robot is located after the first turn, where you wish to start measuring a distance forward. You will need one variable to store an odometer reading at the starting point of the movement. Then another variable that will store the distance measurement from the starting point. You will also need to add a mathematical expression to calculate the distance from the starting point. The code below will create the variables needed.

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

The code below will cause the robot to travel 78 ticks forward. The variable **prevTicks** stores the odometer reading of the left wheel at the moment the robot begins driving the 78 ticks segment. This serves to mark the point where the robot begins the forward movement. That movement stops when the variable ticksTraveled contains a value of 78 `while (ticksTraveled < 78 )`. The value stored in ticksTraveled is computed by this statement: `ticksTraveled = left - prevTicks;`. In other words, the value stored in prevTicks is subtracted from the value stored in left and the result stored in ticksTraveled. That value is the number of ticks traveled since the robot started the movement.

```
int main( )
   {
   drive_getTicks(&left, &right);         //store total distance traveled
   prevTicks = left;                      //store value of left in prevTicks

   while (ticksTraveled < 78 )            // drive straight 78 ticks
      {
      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
      }
   ticksTraveled = 0;                     // reset the counter
   drive_speed(0, 0);                     // stop the robot
   }
```

The code above will work if you use the left wheel odometer. The right wheel odometer counts in negative numbers, which requires a slightly different code. You will learn more about this in the lesson on making turns with an odometer.

It is a good idea to become familiar with the references that cover the libraries used with ActivityBot360. Specifically we are interested here with the reference for the **abdrive360.h** library of functions. You can check this reference at this link:

https://rawcdn.githack.com/parallaxinc/Simple-Libraries/5faf8977b1db2c8565204dd58d8f9c60d0a74257/Learn/Simple%20Libraries/Robotics/ActivityBot360/libabdrive360/Documentation%20abdrive360%20Library.html

File name: ActivityBot360_odometry.pdf
online location: https://www.lafavre.us/robotics/ActivityBot360_odometry.pdf
last update: March 18, 2026