# Robot Documentation

## Robot Class

Initalization:
Using the simplified `Robot` is simple all you have to do is include the `robot.h` file into where you want to use it.

```
#include <robot>
```

If you put the robot in a different folder/path, all you have to do is put the path to the `robot.h` file in. Example:

```
// robot.h is in the `include` folder
#include <include/robot>
```

Where ever you put your .ch program is fine, as long as you know how to navigate to the robot file. **IMPORTANT! YOUR PROGRAM MUST BE ON THE SAME DRIVE (E:/, F:/, etc.) AS THE ROBOT CODE!** Example, let's say that the robot code is in a flash drive (*D:\code\include\robot*) and that your program was in the root of the drive (*D:\my_program.ch* for example)

```
// At the top of your `my_program.ch` file
// Navigate through folders/directories with `/`
#include <code/include/robot>
```

Creating an instance of the robot class is also simple, it's done in the same way as the `CLinkbotI` class.

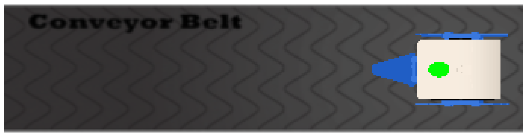
```
Robot myRobotName;
```

After creating an instance of the robot class you have to *initalize* it so that it knows where it is, what direction it's facing, and the radius of it's wheels. You can do this by calling the `init` (short for *initalize*) method on the robot that you created.

```
Robot robo;
robo.init(1, 5, 90, 1.75);
```

The first argument (aka parameter) of the `init` method is the robot's position on the **x-axis**, the second argument is the robot's position on the **y-axis**. The third parameter is the robot's inital rotation. Finally, the last argument is the radius of the wheels.

**This is how the rotation works:**

0° Rotation: 

90° Rotation:
Conveyor Belt

(et cetera)

The Basics:
The first basic method of the Robot class is `drive`.

```
Robot robo;
robo.drive(11.5);
```

The first and only argument of drive is the distance to drive, in inches. If the distance is negative then the robot will drive backwards.

Another basic method is `turn`.

```
Robot robo;
robo.turn(90);
```

Its first and only argument is the angle in degrees to turn. If the argument is positive, then the robot will turn right, if the argument is negative, then the robot will turn left.

The next simple method is `driveShape`

```
Robot robo;
robo.driveShape(TRIANGLE, 10);
```

The first argument to the `driveShape` method is the shape type. It supports all shapes from triangle to decagon, **the shape name must be in caps**. Circles are also supported. The second argument is the side length of the shape that you want the robot to drive, or if you want to draw a circle, the circumference. **IMPORTANT! THE CIRCUMFERENCE OF A CIRCLE CANNOT BE LESS THAN 2!**

Intermediate:
*These are the methods that you will probably use less often, and are a bit more challenging to use and understand.*

The first, and simplest method you'll find in this section is `driveTo`.

```
Robot robo;
robo.driveTo(3, 7);
```

The first argument is the *x-coordinate* that you want the robot to travel to, the second argument is the *y-coordinate* that you want the robot to travel to.

The next method is `turnArc`.

```
Robot robo;
robo.turnArc(10, 90);
```

The first argument is the radius of the arc that your robot will turn. The second argument is the degress that you want your robot to turn along that arc.

The next methods after that are `wait` and `waitMillis`. These are pretty much self explanatory.

```
Robot robo;
robo.wait(20); // Takes in the number of seconds
robo.waitMillis(250); // Takes in the number of milliseconds
```

## Advanced:

*These are the methods that you will probably hardly use, and are more challenging to use and understand. If you have any questions about these, **please ask me!***

The first more advanced methods are `getPosition` and `getRotation` . These methods will give you the robot's current position and rotation.

```
Robot robo;
// Do stuff with the robot...
vec2 currentPosition = robo.getPosition();
double currentRotation = robo.getPosition();
```

Using the rotation is simple, it's just a double that you can print out or do whatever with. For the position, it's a `vec2` or a vector of length 2. That just means that it's a 2D coordinate position with an `x` and a `y` . You can use it like so:

```
Robot robo;
// Do stuff with the robot...
vec2 currentPosition = robo.getPosition();

// Printing out the x and y of the position
// You can just use currentPosition.x and currentPosition.y like normal
// variables of the `double` type
printf("Position is (%lf, %lf)", currentPosition.x, currentPosition.y);
```

A vec2 has two variables that you could say are part of it, `x` and `y` . To access these variables you would type `{your vec2 name}.x` or `{your vec2 name}.y` .

The next more advanced method is `getRobot` . This is for getting access to the `CLinkbotI` variable that actually controls the robot. You might need this to use a function that is not included in my robot code, or to just manually control the robot.

```
Robot robo;
CLinkbotI *childRobot;
childRobot = robo.getRobot();

childRobot->driveDistance(10, 1.75);
childRobot->turnRight(90, 1.75, 3.69);
```

This is probably hard to understand because it's the first time that you have been introduced to a concept called pointers. If you want to run a function on the CLinkbotI, then you have to use `{name of your CLinkbotI}->{name of the function}` .

The last, and definitely most advanced method is `drivePathTo` . This method will have the robot find and drive a path to a certain point, given that the robot can only drive on a 2D grid. Here is an example:

```
Robot robo;
```

### Non-Blocking Functions

*These are the functions that will not block the execution of your program*

```
Robot robo;
robo.driveNB(10);
robo.turnNB(90);
robo.driveToNB(10, 10);
robo.turnArcNB(10, 90);
```

Last Updated *6:36 am | May 18th, 2019*