# V5 VEXcode: Movement

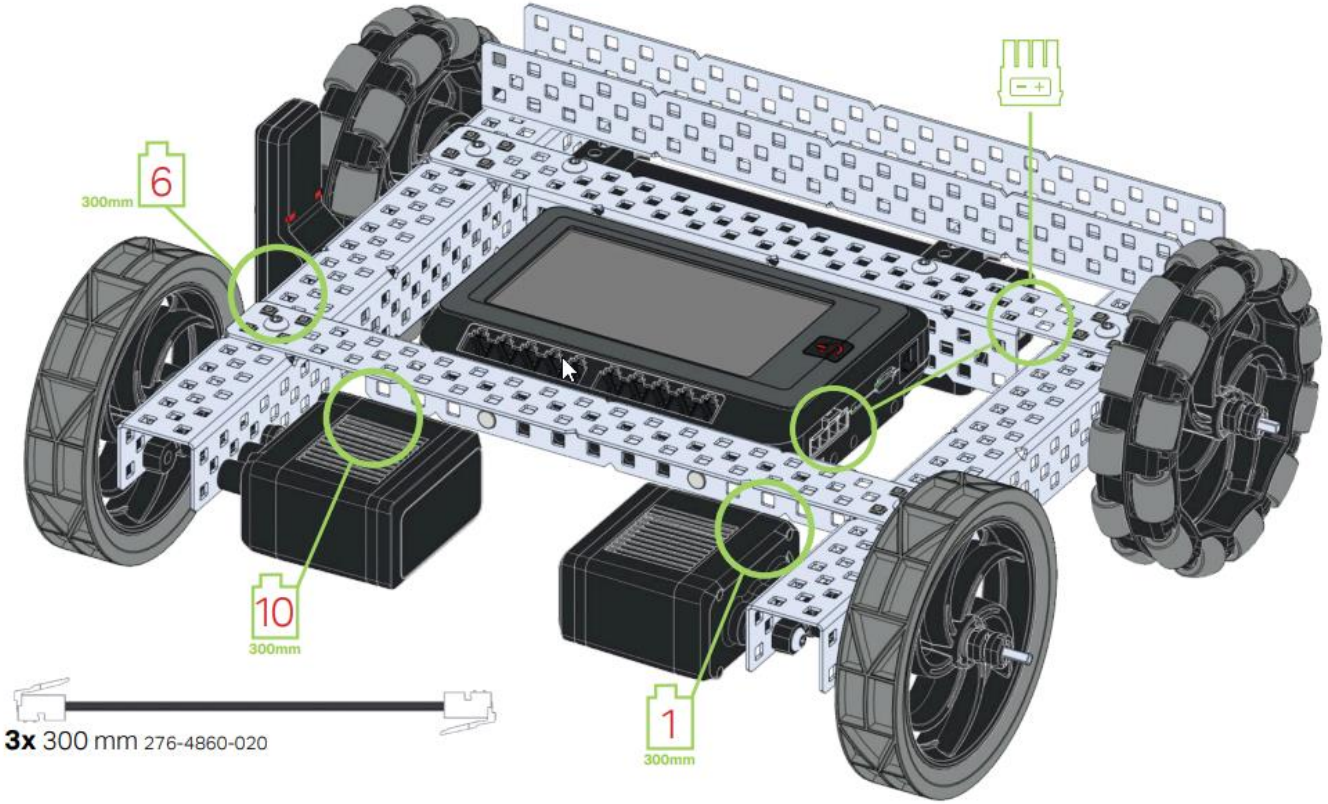
Motor Setup

.rotateFor()

.spin()

# Using the Clawbot

- Clawbot default values
  - RightMotor = Port 10
  - LeftMotor = Port 1
- Motors come with the green (18:1 or 200 RPM) gear cartridge

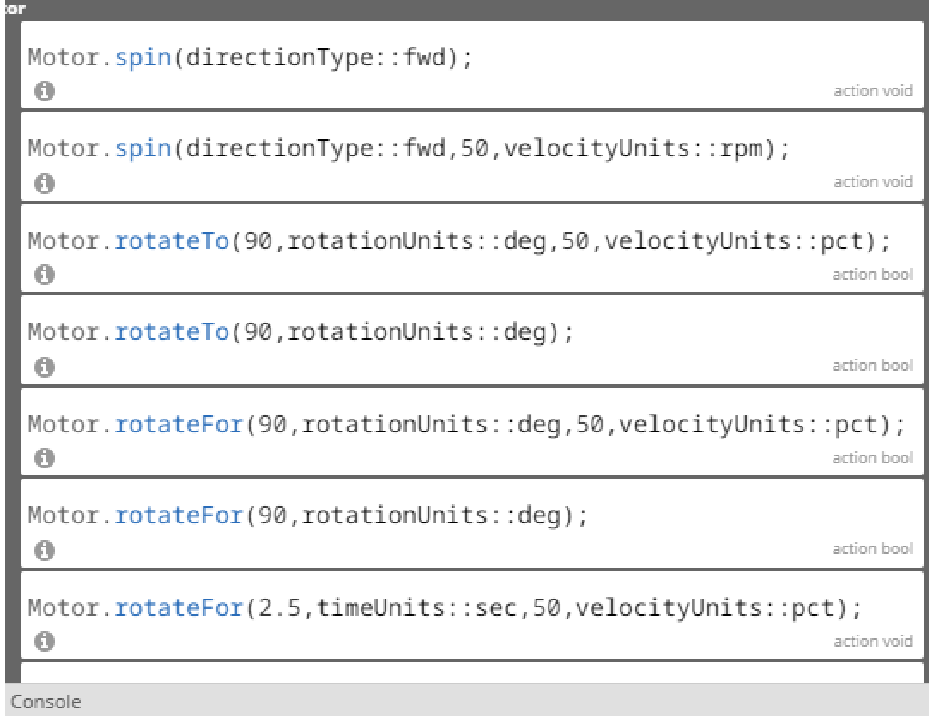


# Need to configure the robot in V5 Coding Studio

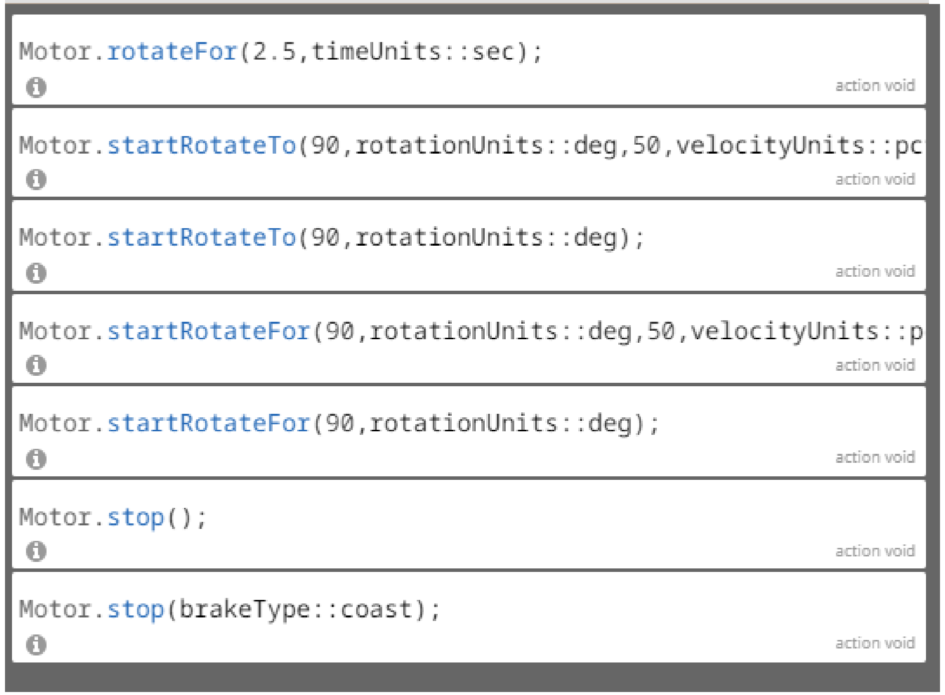
- `Defining the motors: Before task main()`
  - **`motor RightMotor = motor( PORT1 );`**
    - **`RightMotor`** is the what the motor is named in this example. The programmer can pick a different name. It must start with a letter, no spaces, no punctuation, not a reserved word, not used somewhere else and is <u>descriptive</u>.
    - **PORT1** defines the port where the motor connects to the brain.
  - **`motor LeftMotor = motor( PORT10, true );`**
    - **LeftMotor** is the what the motor is named in this example. The programmer can pick a different name. It must start with a letter, no spaces, no punctuation, not a reserved word, not used somewhere else and is <u>descriptive</u>.
    - **PORT10** defines where the motor connects to the brain
    - **true** sets this motor to be reversed.

# Programming the Motor

- We will focus on the Motor.rotateFor() command for movement.
- There are several movement commands.
- Help.vexcodingstudio.com for a reference to other Motor commands. (Settings, actions, sensing commands)



```
Motor.spin(directionType::fwd);
Motor.spin(directionType::fwd,50,velocityUnits::rpm);
Motor.rotateTo(90,rotationUnits::deg,50,velocityUnits::pct);
Motor.rotateTo(90,rotationUnits::deg);
Motor.rotateFor(90,rotationUnits::deg,50,velocityUnits::pct);
Motor.rotateFor(90,rotationUnits::deg);
Motor.rotateFor(2.5,timeUnits::sec,50,velocityUnits::pct);
```



```
Motor.rotateFor(2.5,timeUnits::sec);
Motor.startRotateTo(90,rotationUnits::deg,50,velocityUnits::pc
Motor.startRotateTo(90,rotationUnits::deg);
Motor.startRotateFor(90,rotationUnits::deg,50,velocityUnits::p
Motor.startRotateFor(90,rotationUnits::deg);
Motor.stop();
Motor.stop(brakeType::coast);
```

# Motor.rotateFor();

- **Motor.rotateFor(double rotation , rotationUnits units, boolean waitForCompletion = true);**
- Used to rotate the left and right motors for a specific target rotational distance.
- Motor.rotateFor can be used either as a blocking or non-blocking command.
- Can be a non-blocking by including ‘false’ as the waitForCompletion .
- This command can be used not only with wheel motors but also with arm or claw motors, allowing them to be moved specific distances while safely avoiding overextensions without the need for a limit switch.

# Motor.rotateFor() Example.  Assuming LeftMotor and RightMotor were already configured in the program.

… 180 degrees …

… and will not block the next command from starting.

- LeftMotor.rotateFor( 180, rotationUnits::deg, 50, velocityUnits::pct, false );

The LeftMotor will…

…rotateFor()…

… at 50% velocity…

… 180 degrees …

… and will block the next command from starting until after this command is completed.

- RightMotor.rotateFor( 180, rotationUnits::deg,50, velocityUnits::pct );

The LeftMotor will…

…rotateFor()…

… at 50% velocity…

# Motor Command rotateFor() with all of its options. Assuming *Motor* is defined previously as a motor.

A double (real) value that describes **how many units** will be completed. ( 10, 4.5, -20,…)

| units | Description |
|---|---|
| **deg** | A rotation unit that is measured in **degrees**. |
| **rev** | A rotation unit that is measured in **revolutions**. |
| **raw** | A rotation unit that is measured in **raw data form**. |

```
Motor.rotateFor(double rotation, rotationUnits:: units, double velocity, velocityUnits:: units_v, bool waitForCompletion=true);
```

A double (real) value that describes the velocity. ( 10, 4.5, -20,…)

| units | Description |
|---|---|
| **pct** | A velocity unit that is measured in percentage. |
| **rpm** | A velocity unit that is measured in rotations per minute. |
| **dps** | A velocity unit that is measured in degrees per second |

Optional: If left off then it will complete this command before starting the next command.
**false** = It will start the next command immediately after starting this command.

# ‘rotateFor’ Examples

```
int main() {
   LeftMotor.rotateFor(3.5, rotationUnits::rev, 75, velocityUnits::rpm);

   LeftMotor.rotateFor(360, rotationUnits::deg, 80, velocityUnits::dps, false);
   RightMotor.rotateFor(720, rotationUnits::deg, 80, velocityUnits::dps);

   LeftMotor.rotateFor(3.5, rotationUnits::rev, false);
   RightMotor.rotateFor(3.5, rotationUnits::rev);//3.5 revolutions

   RightMotor.rotateFor(3500, timeUnits::msec, 70, velocityUnits::pct);//3.5 seconds
   RightMotor.rotateFor(3.5, timeUnits::sec);
}
```

The LeftMotor will rotate 3.5 revolutions at 75 revolutions per minute (rpm).

The LeftMotor will rotate 360 degrees at 80 degrees per second (dps) and will not wait until the command is finished before going to the next command.

The RightMotor will rotate 720 degrees at 80 degrees per second (dps) and will complete this command before going to the next command.

The LeftMotor will rotate 3.5 revolutions at the default speed or speed set by the Motor.setVelocity command and will not wait until the command is finished before going to the next command.

The RightMotor will rotate 3500 milliseconds (ms) and 70% of the maximum speed.

The RightMotor will rotate 3.5 seconds (sec) at the default speed.

# Using the built in motor encoder to calculate the rotations to complete a distance

- With the rotateFor() command you can use the built in motor encoder to control how far the robot moves.
- We know
  - 360 degrees = 1 revolution
  - Since the wheel is connected directly to the motor.
    - One motor revolution = One wheel revolution
  - One wheel revolution = circumference of the wheel distance traveled
  - Circumference of the wheel = PI * wheel diameter

- **Mini Challenge 1:**
  - Write a program to have the robot move exactly **one yard** without guessing.
  - No testing on the course
  - Check answers on the floor.

Example: Calculating the revolutions needed to travel 5 feet with a 4-inch diameter wheel directly connected to the motor.

5 feet*(12 inches/ 1 foot)*(1 Revolution/ PI*4 inches) = 4.77 revolutions

# Mini Challenge 2: 90 degree turns

- Write a program
  - 90 degree turn in each direction
  - Save it in slot 2
  - Give a short description
  - Try to calculate
  - Check and modify as needed

# Motor.spin()

- The .rotateFor() function stops the motor after completing the command that can make travel inconsistent.
- The Motor.spin() function works like the motor[] = 100 command in RobotC.
- It will start the motor spinning and then go onto the next command.
- The motor will continue spinning until the motor receives another command.

# Motor.spin() Syntax

dir options

| | |
|---|---|
| fwd | Forward |
| rev | Reverse |

*Optional*: A double (real) value that describes the velocity. ( 10, 4.5, -20,…) If not included it uses the predefined velocity and units.

- LeftMotor.spin(**directionType**:: dir*, double velocity,* ***velocityUnits::*** *units*);

| units | Description |
|---|---|
| **pct** | A velocity unit that is measured in percentage. |
| **rpm** | A velocity unit that is measured in rotations per minute. |
| **dps** | A velocity unit that is measured in degrees per second |

- LeftMotor.spin(directionType::rev, 50, velocityUnits::pct);

# Motor.spin example program

```
int main()
{          // Point Turn
           LeftMotor.spin(directionType::rev, 50, velocityUnits::pct);
           RightMotor.spin(directionType::fwd, 50, velocityUnits::pct);
           task::sleep( 2000 );
           //Swing Turn
           LeftMotor.spin(directionType::fwd, 0, velocityUnits::pct);
           RightMotor.spin(directionType::fwd, 50, velocityUnits::pct);
           task::sleep( 2000 );
           //Straight using the setVelocity function
           LeftMotor.setVelocity(50, velocityUnits::pct);
           RightMotor.setVelocity(50, velocityUnits::pct);
           LeftMotor.spin(directionType::fwd);
           RightMotor.spin(directionType::fwd);
           task::sleep( 2000 );
           //Stop both motors.
           LeftMotor.stop();
           RightMotor.stop();
}
```

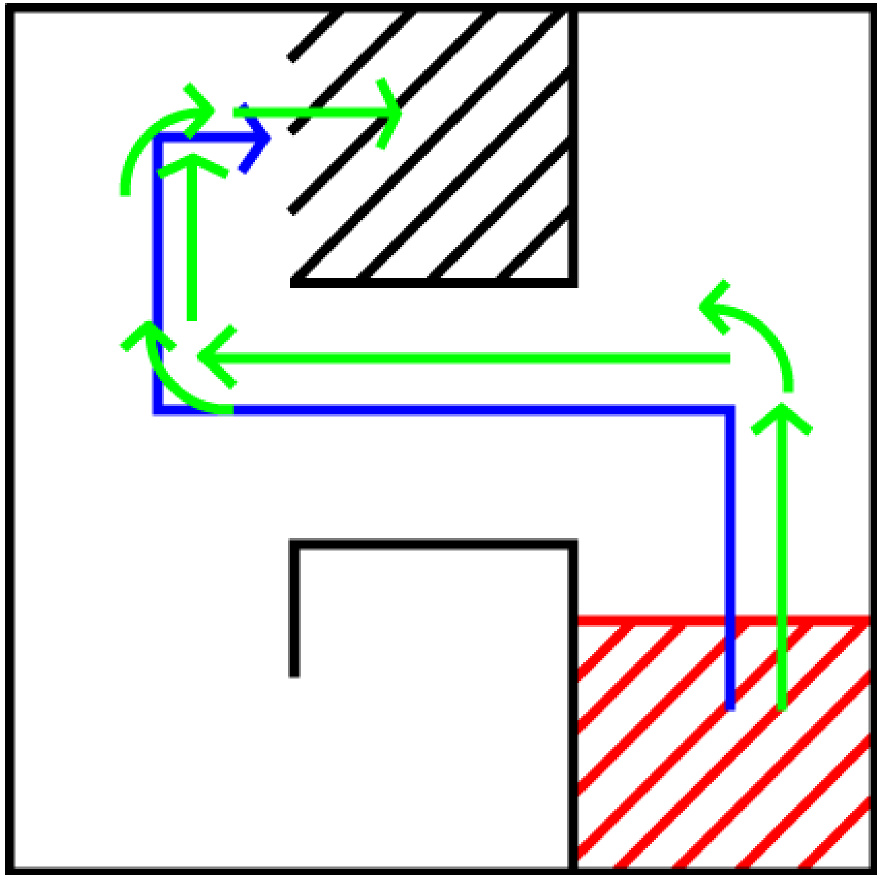
# *Using Motor.spin() and the while loop to go for a distance*

```
int main()
{
        //Start the motors spinning
        LeftMotor.spin(directionType::fwd, 50, velocityUnits::pct);
        RightMotor.spin(directionType::fwd, 50, velocityUnits::pct);

        //While the rotations are less than 360 degrees, continue
        while (RightMotor.rotation(rotationUnits::deg)<360){}

        //When the RightMotor is no longer less than 360 degrees, stop both motors
        LeftMotor.stop(); //Stop both motors.
        RightMotor.stop();
}
```

```
int main()
{
        Brain.Screen.print("Starting Degrees %.1f", RightMotor.rotation(rotationUnits::deg));
        task::sleep(2000);
        LeftMotor.spin(directionType::fwd, 50, velocityUnits::pct);
        RightMotor.spin(directionType::fwd, 50, velocityUnits::pct);
        Brain.Screen.newLine();
        while (RightMotor.rotation(rotationUnits::deg)<360)
        {
                task::sleep(20);
                Brain.Screen.print("Degrees %.1f", RightMotor.rotation(rotationUnits::deg));
                Brain.Screen.setCursor(2,1);
        }
        LeftMotor.stop();//Stop both motors.
        RightMotor.stop();
        Brain.Screen.newLine();
        Brain.Screen.print("Ending Degrees %.1f", RightMotor.rotation(rotationUnits::deg));
}
```

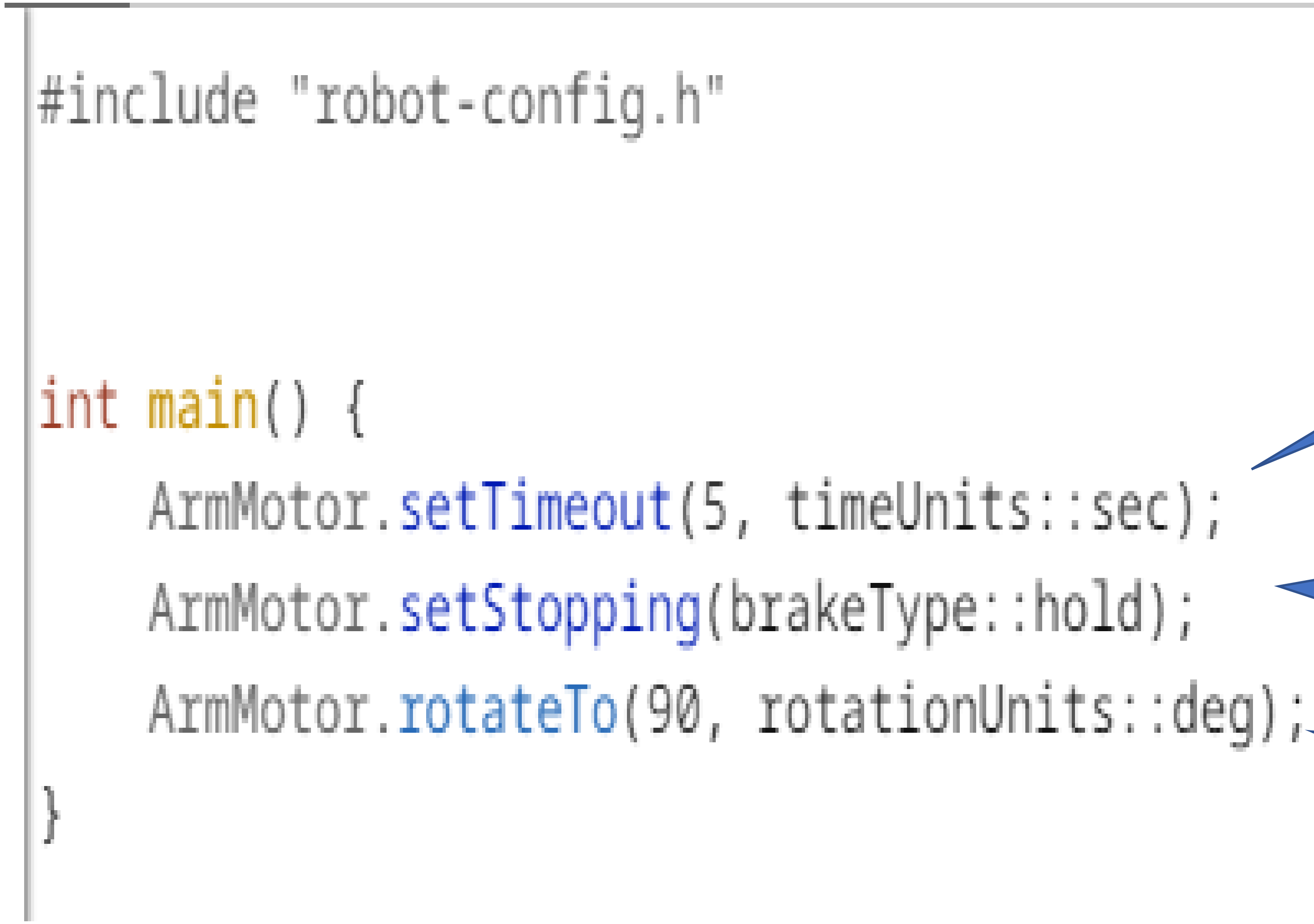
Same as previous slide with code to display current rotations.

# Labyrinth Challenge



- Complex Behavior
  - Complete the maze
- Simple Behaviors

# Programming Arms: Handling a range of motion limitation.



```
#include "robot-config.h"


int main() {
    ArmMotor.setTimeout(5, timeUnits::sec);
    ArmMotor.setStopping(brakeType::hold);
    ArmMotor.rotateTo(90, rotationUnits::deg);
}
```

Sets the motor to stop spinning if it gets stuck

Sets the motor to maintain its position after the movement is complete.

Rotates to the **absolute** position of 90 degrees

# Using the Motor for Arm Movement

```
motor ArmMotor = motor(PORT8);

int main()
{
    ArmMotor.setTimeout(5, timeUnits::sec);
    ArmMotor.setStopping(brakeType::hold);
    ArmMotor.rotateTo(90, rotationUnits::deg);

}
```

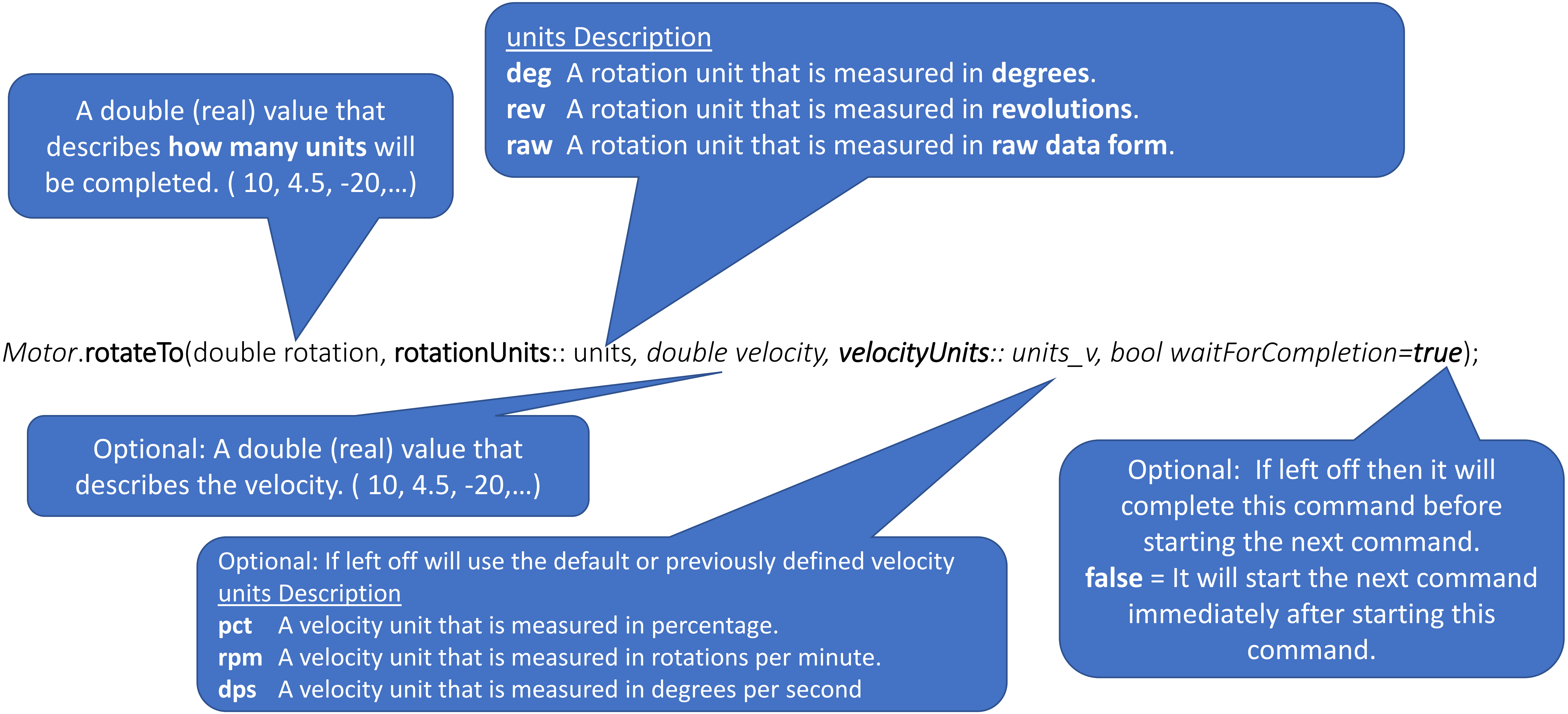
The motor setup is just like the motor setup when motors are used for driving.

Sets the motor to stop spinning if it gets stuck

Sets the motor to maintain its position after the movement is complete.
brakeType::hold
brakeType::coast
brakeType::brake

Rotates to the **absolute** position of 90 degrees.  So the arm could move up or down.

# Motor Command rotateTo() with all of its options. Assuming *Motor* is defined previously as a motor.



*Motor*.**rotateTo**(double rotation, **rotationUnits**:: units*, double velocity,* ***velocityUnits****:: units_v, bool waitForCompletion=****true***);

A double (real) value that describes **how many units** will be completed. ( 10, 4.5, -20,…)

<u>units Description</u>

**deg** A rotation unit that is measured in **degrees**.

**rev** A rotation unit that is measured in **revolutions**.

**raw** A rotation unit that is measured in **raw data form**.

Optional: A double (real) value that describes the velocity. ( 10, 4.5, -20,…)

Optional: If left off will use the default or previously defined velocity

<u>units Description</u>

**pct** A velocity unit that is measured in percentage.

**rpm** A velocity unit that is measured in rotations per minute.

**dps** A velocity unit that is measured in degrees per second

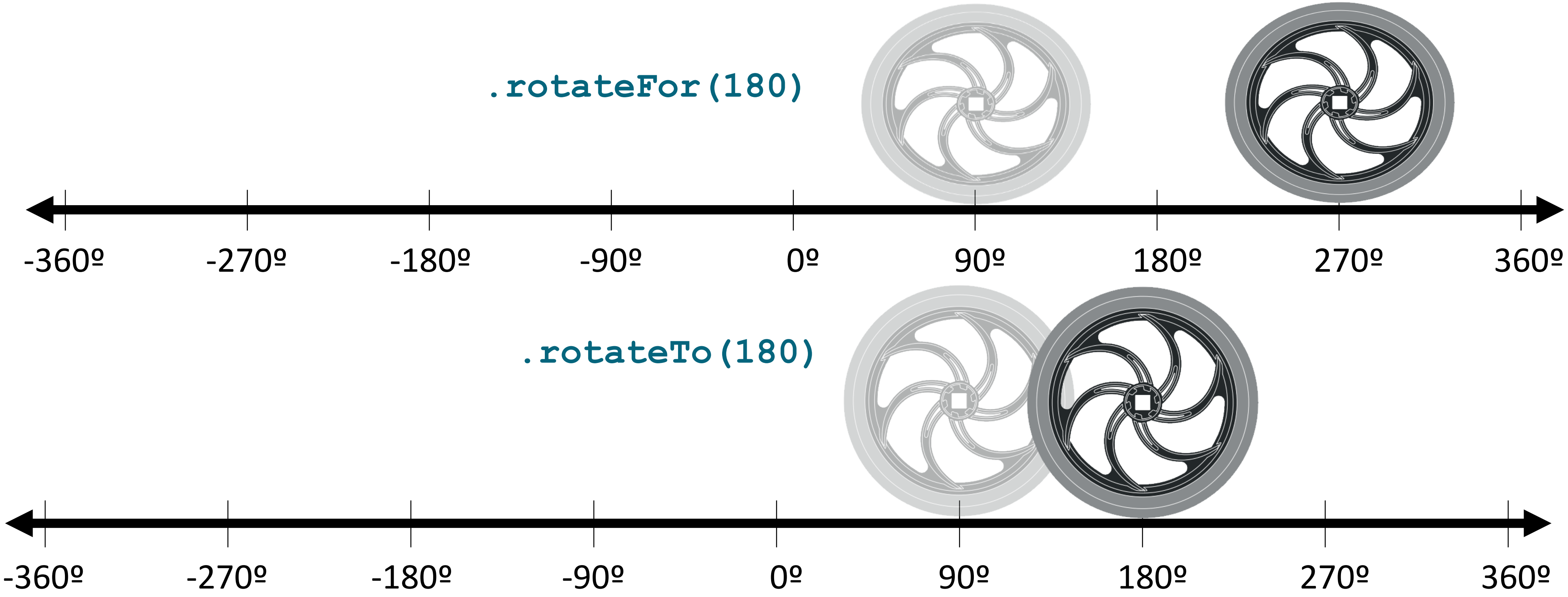
Optional: If left off then it will complete this command before starting the next command.

**false** = It will start the next command immediately after starting this command.

# rotateTo() vs rotateFor()

- rotateFor() rotates a motor the full specified amount regardless of the current motor encoder reading.

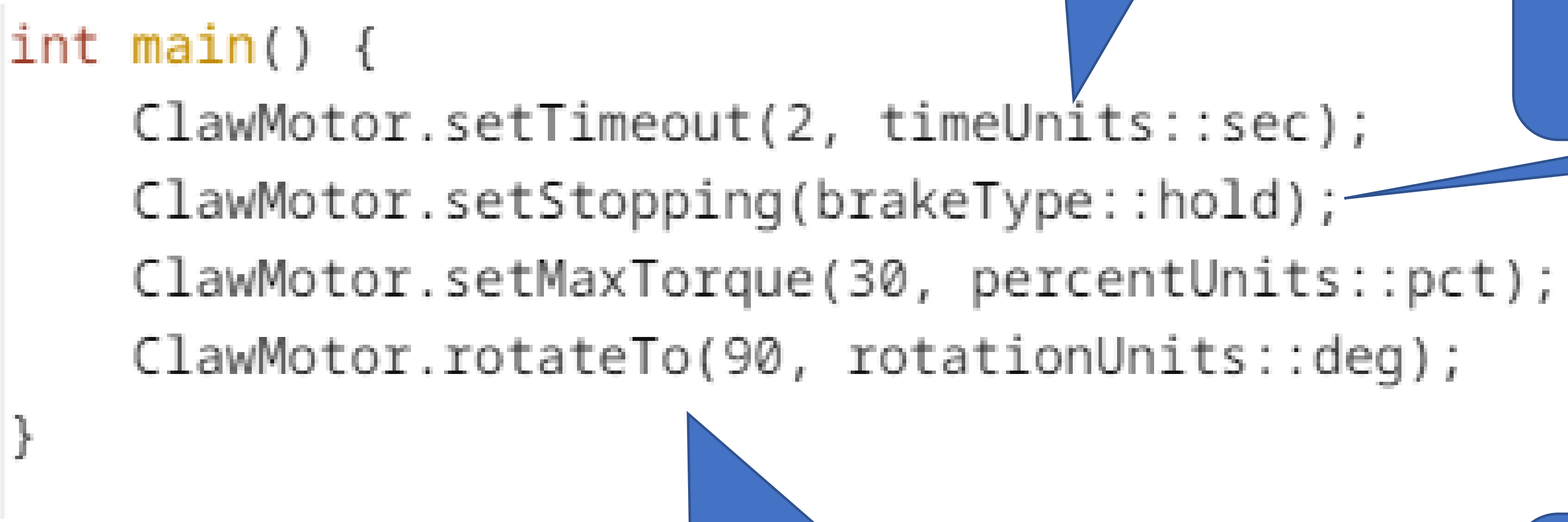
- rotateTo() rotates a motor to a specific motor encoder reading.



# Programming Hands/Grabbers

Sets the motor to stop spinning if it gets stuck after 2 seconds

Sets the motor to maintain its position after the movement is complete.



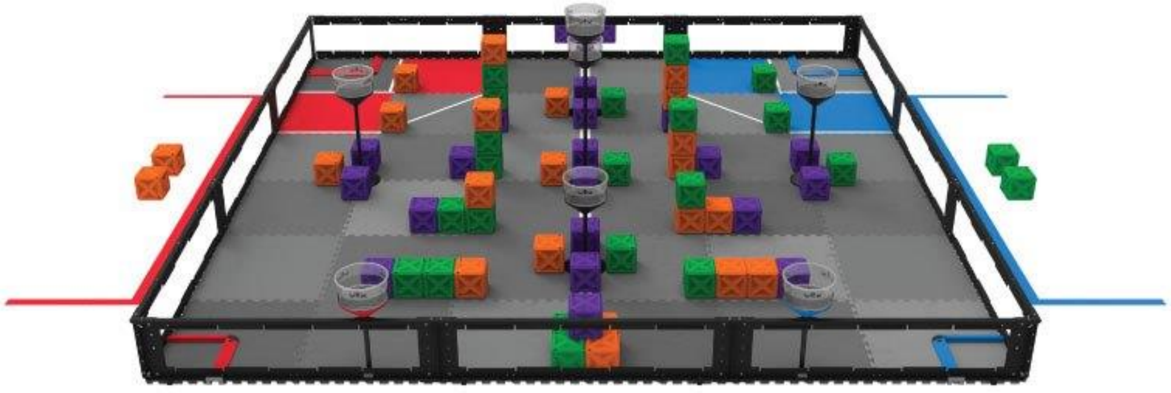
```
int main() {
    ClawMotor.setTimeout(2, timeUnits::sec);
    ClawMotor.setStopping(brakeType::hold);
    ClawMotor.setMaxTorque(30, percentUnits::pct);
    ClawMotor.rotateTo(90, rotationUnits::deg);
}
```

Rotates to the **absolute** position of 90 degrees

Sets the maximum amount of torque the motor will use.

# Exercise:

Using the mini cubes and the modified field, score as many points as you can autonomously.



- Each Cube scored in a Goal Zone is worth a base of one (1) point.
- For each Cube of a given color that is Placed into a Tower, the point value for Cubes of that color increases by one (1) point.
- For example, if there are three (3) green Cubes Placed in Towers at the end of the Match, then all green Cubes Scored in Goal Zones are worth four (4) points.