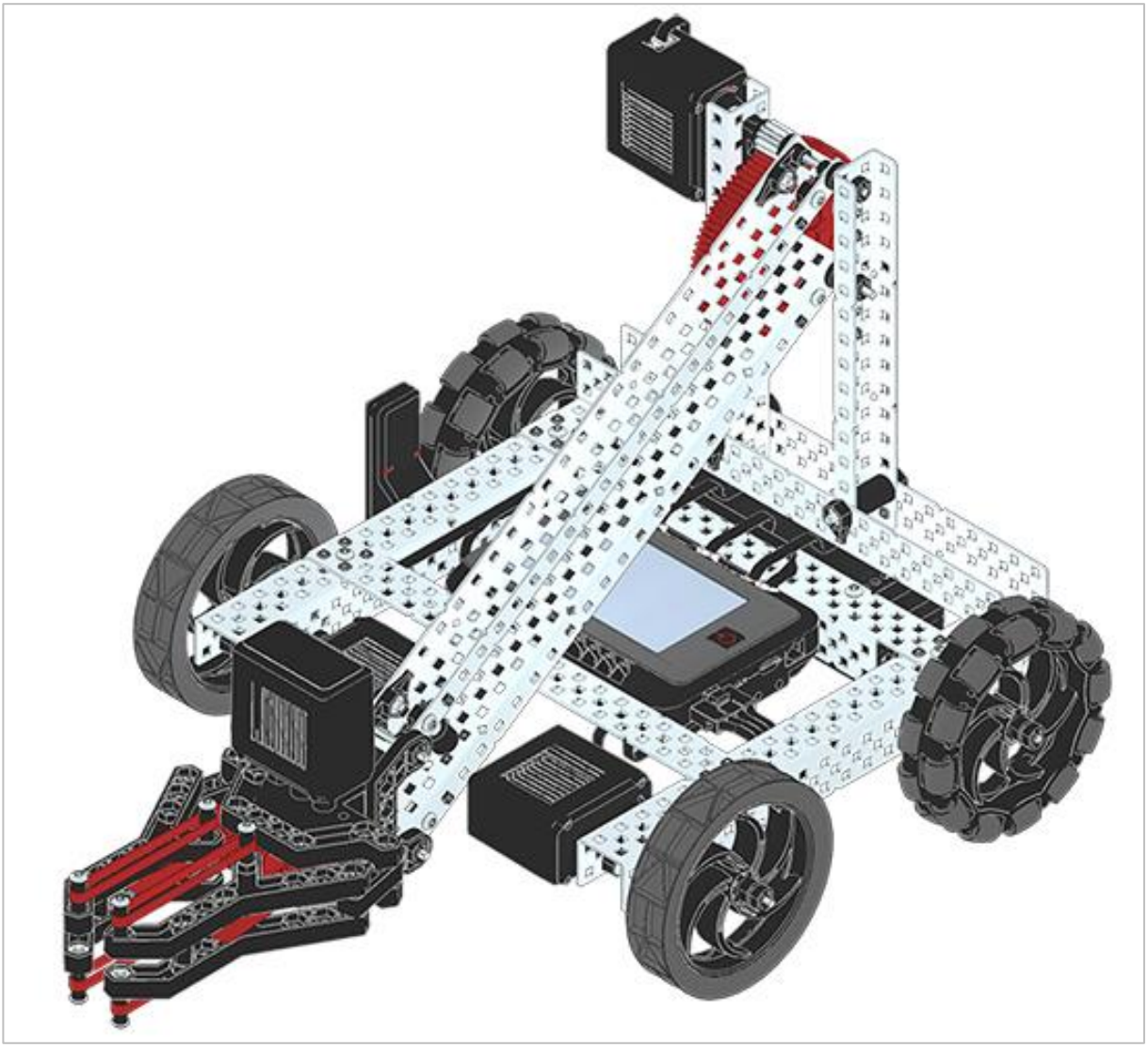
# Loop, There It Is!

Learn how to program loops to get your robot grooving!


Discover new hands-on builds and programming opportunities to further your understanding of a subject matter.

# The Completed Look of the Build



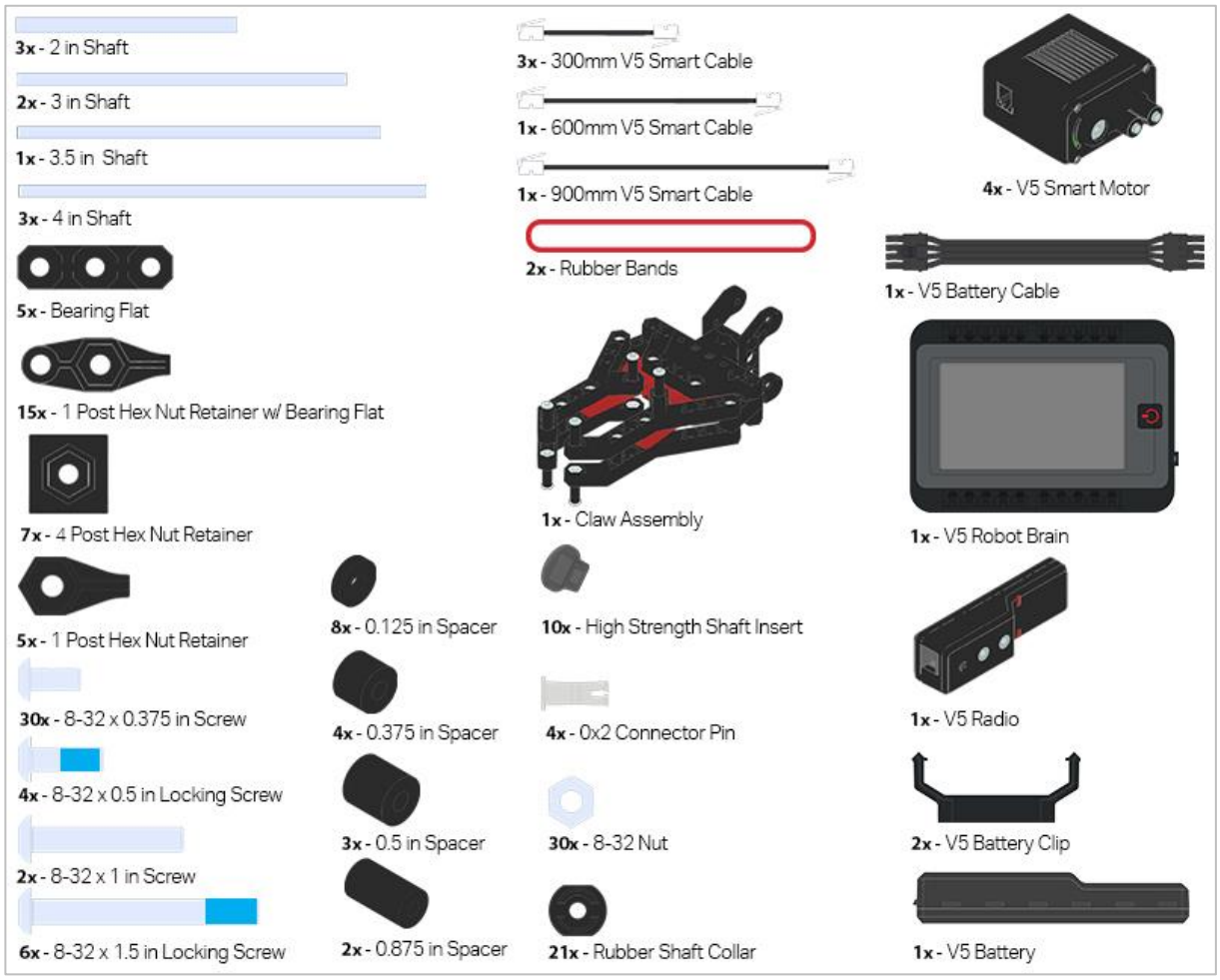
*Completed VEX V5 Clawbot*

The VEX V5 Clawbot is an extension of the VEX V5 Speedbot that can be programmed to move around and interact with objects.

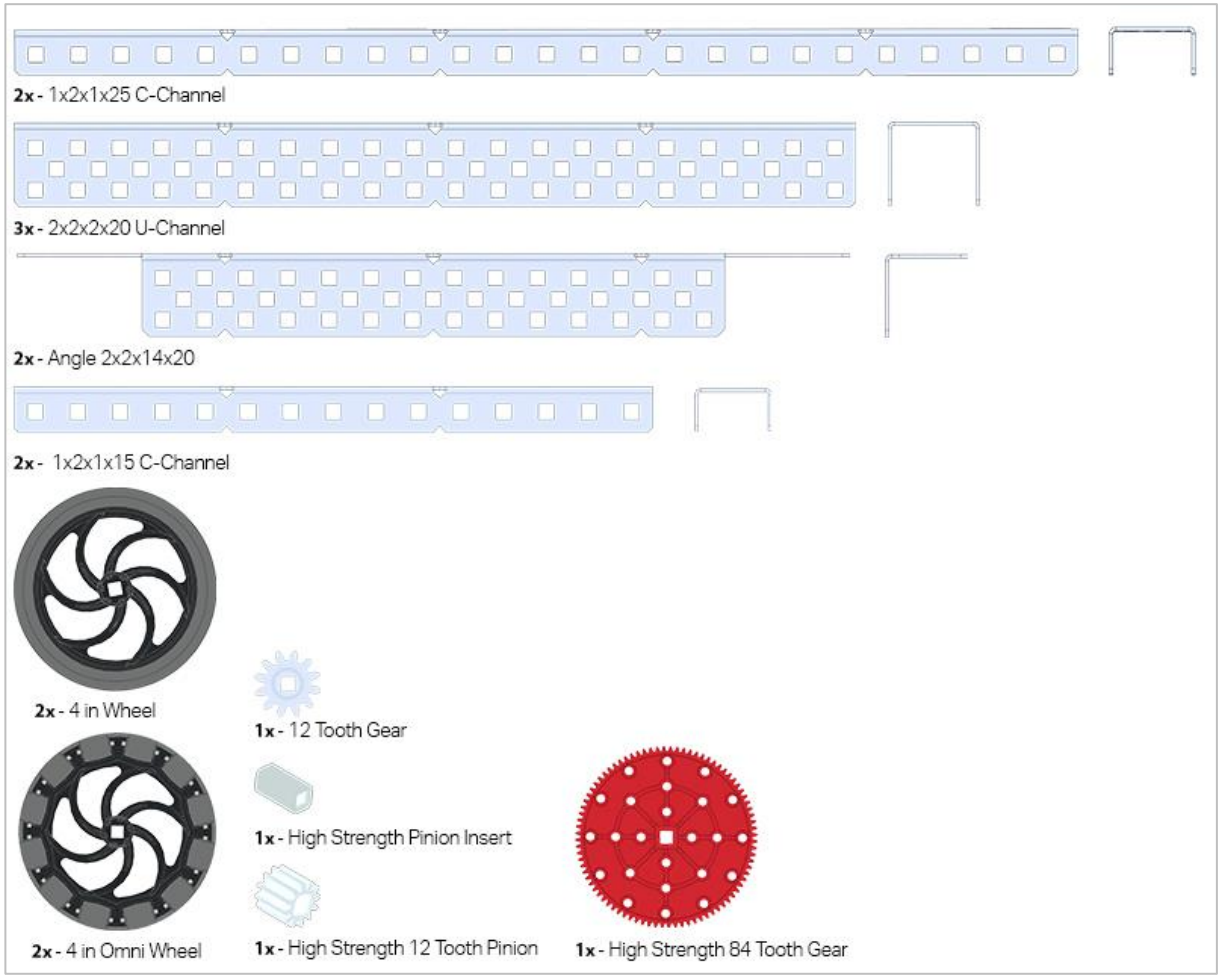
# Parts Needed: Part 1

**Can be built with:**

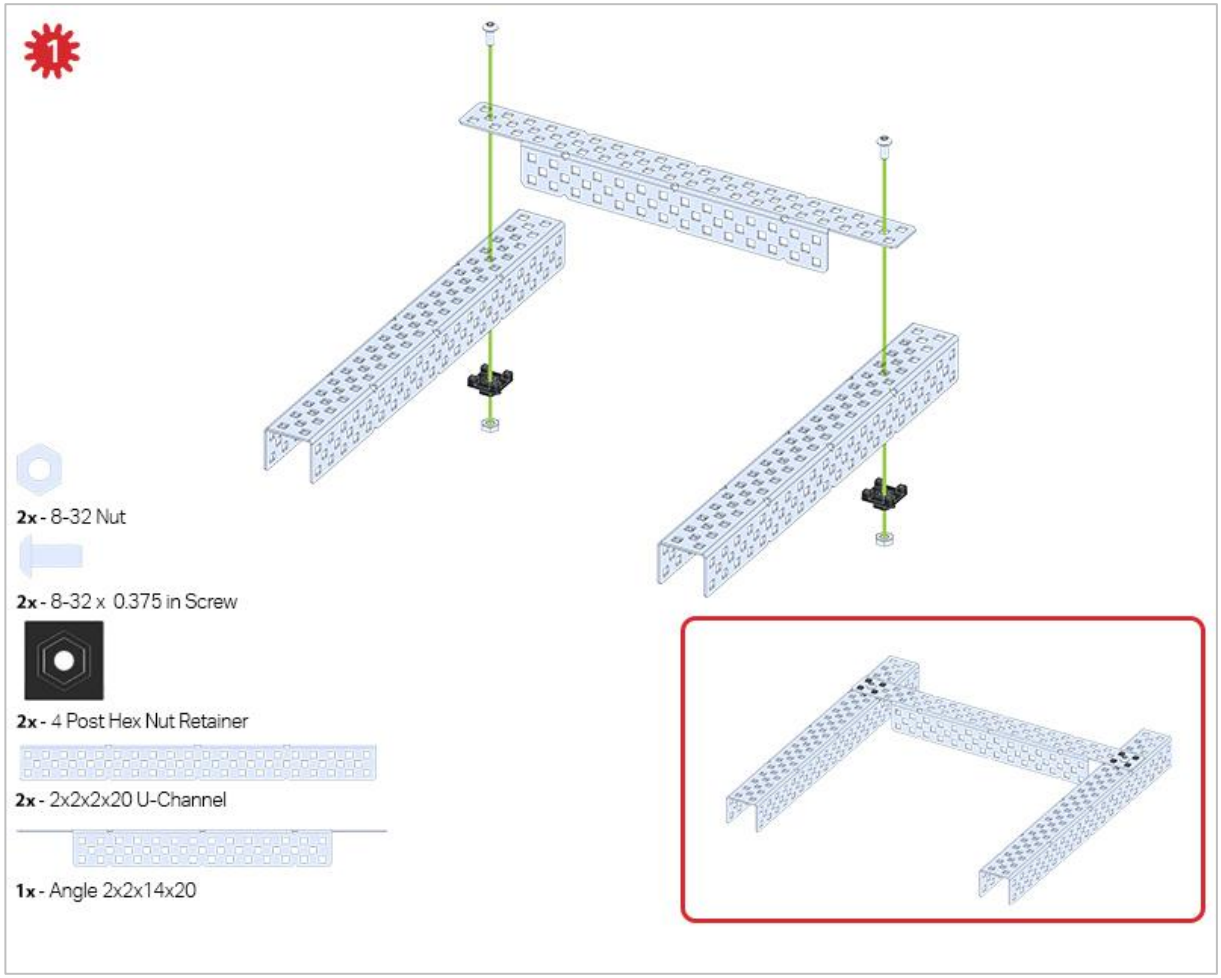
- VEX V5 Classroom Starter Kit



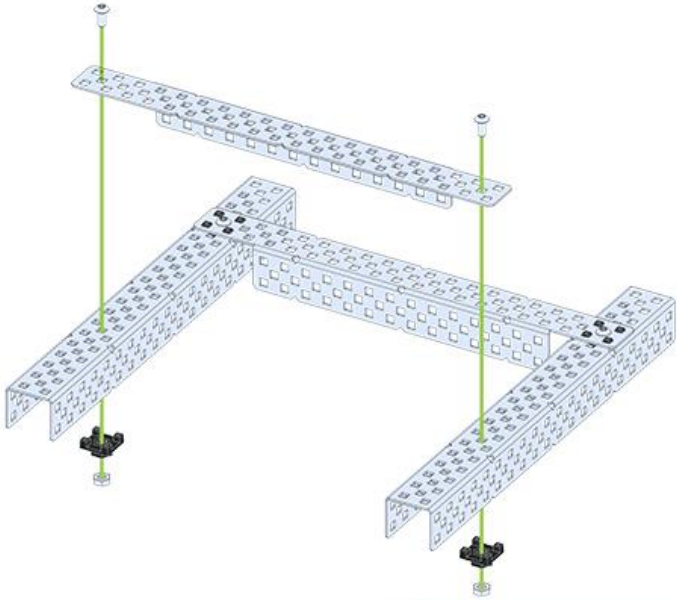
# Parts Needed: Part 2



# Build Instructions



2





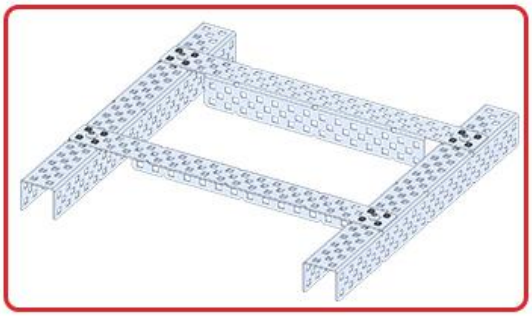
**2x** - 8-32 Nut



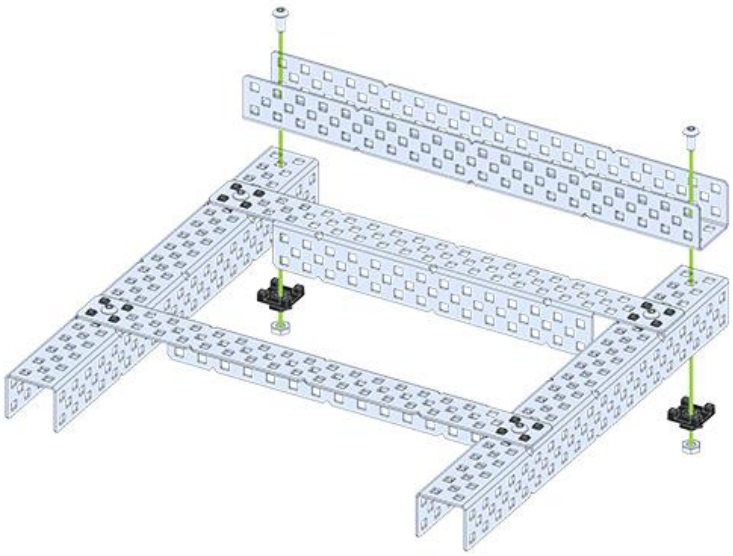
**2x** - 8-32 x 0.375 in Screw

**2x** - 4 Post Hex Nut Retainer

**1x** - Angle 2x2x14x20



3

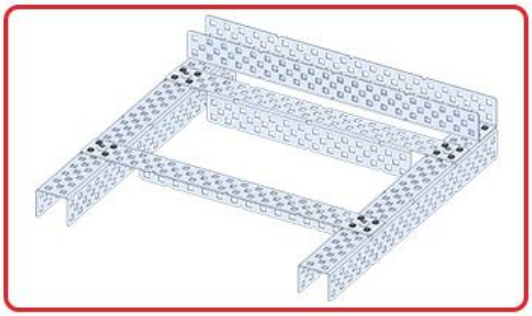


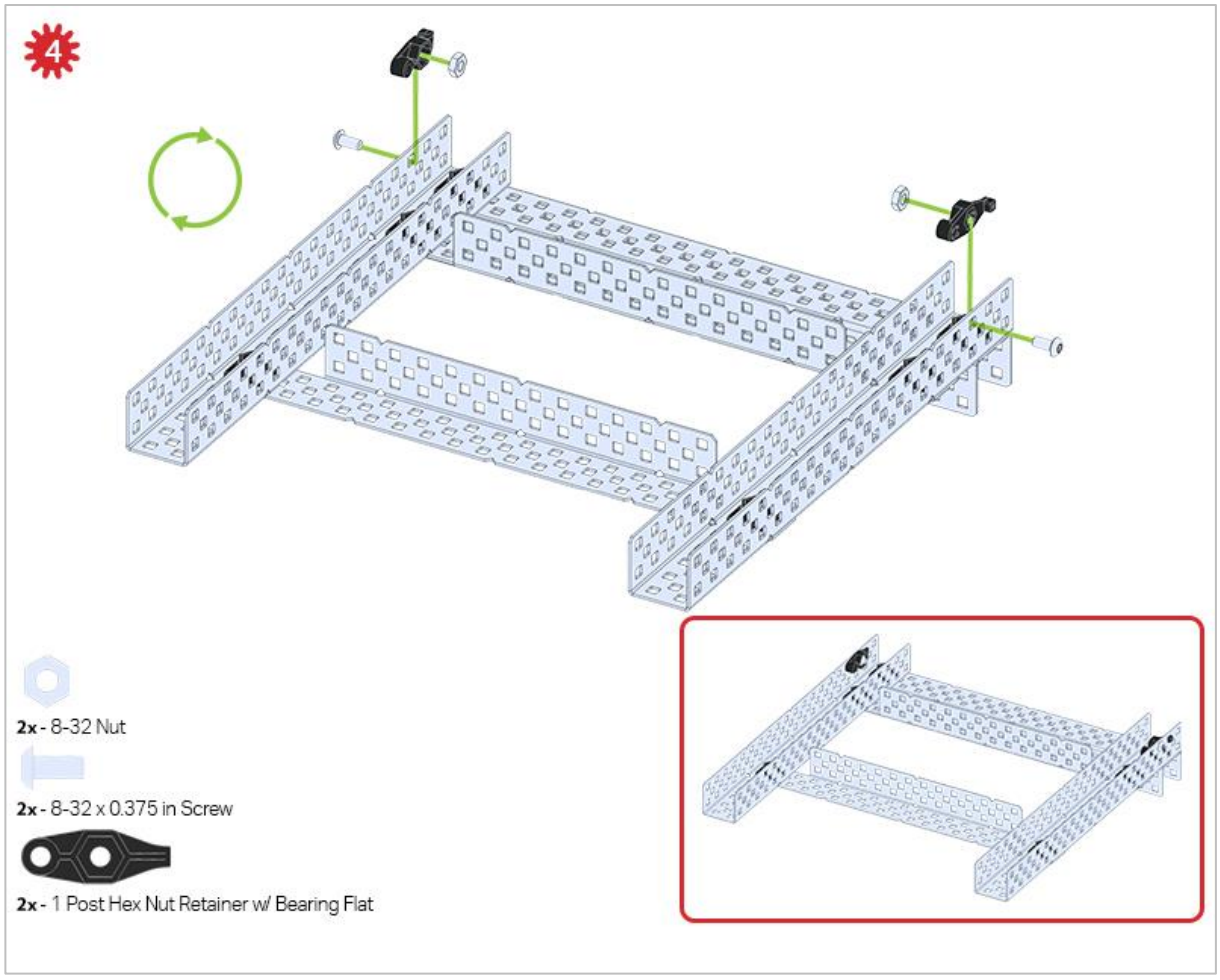
**2x** - 8-32 Nut

**2x** - 8-32 x 0.375 in Screw

**2x** - 4 Post Hex Nut Retainer

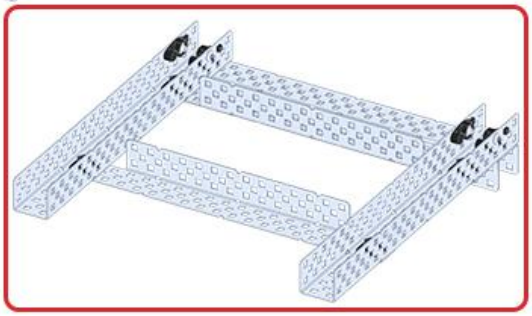
**1x** - 2x2x2x20 U-Channel






*The green icon indicates that the build needs to be flipped over (upside down).*

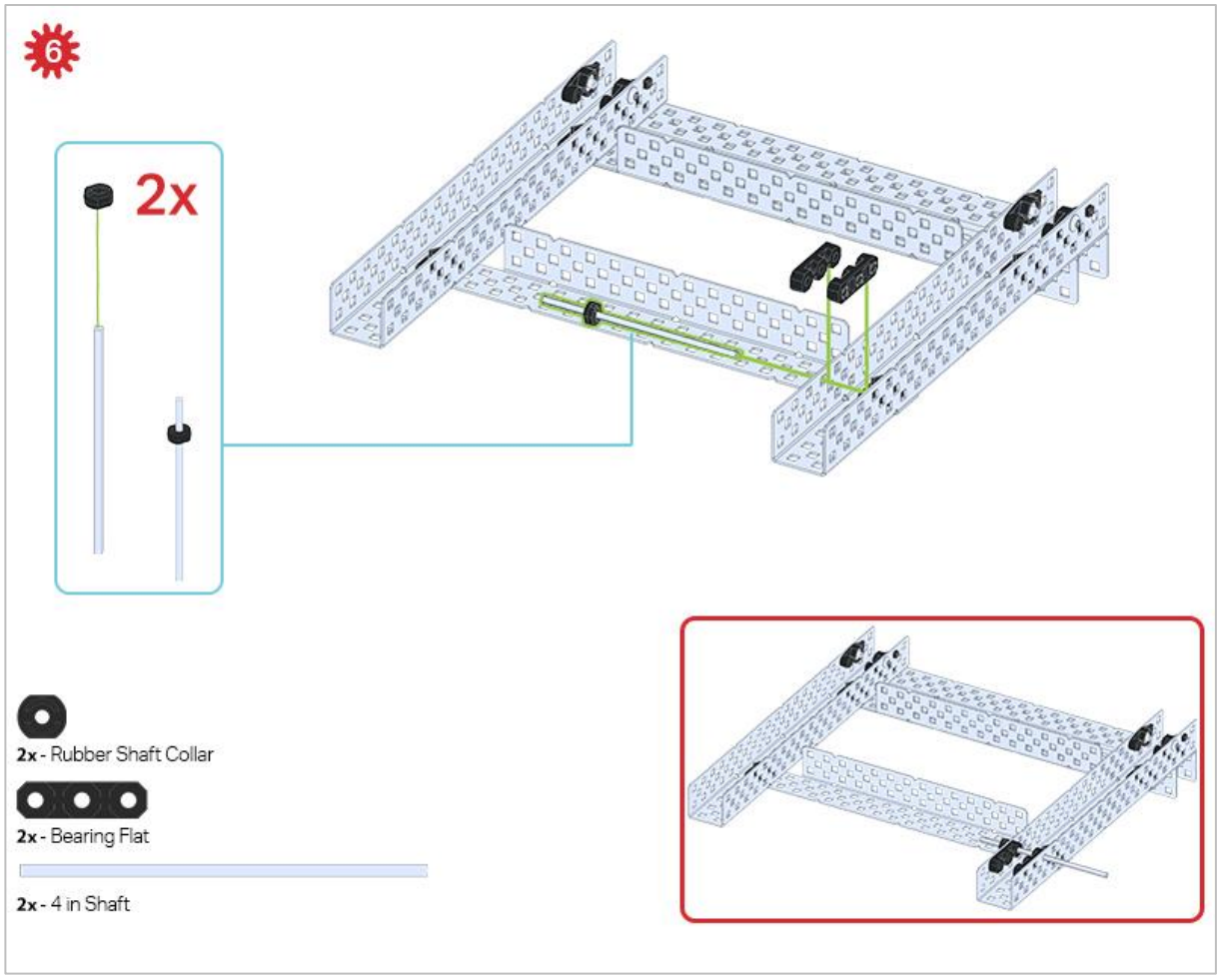
5



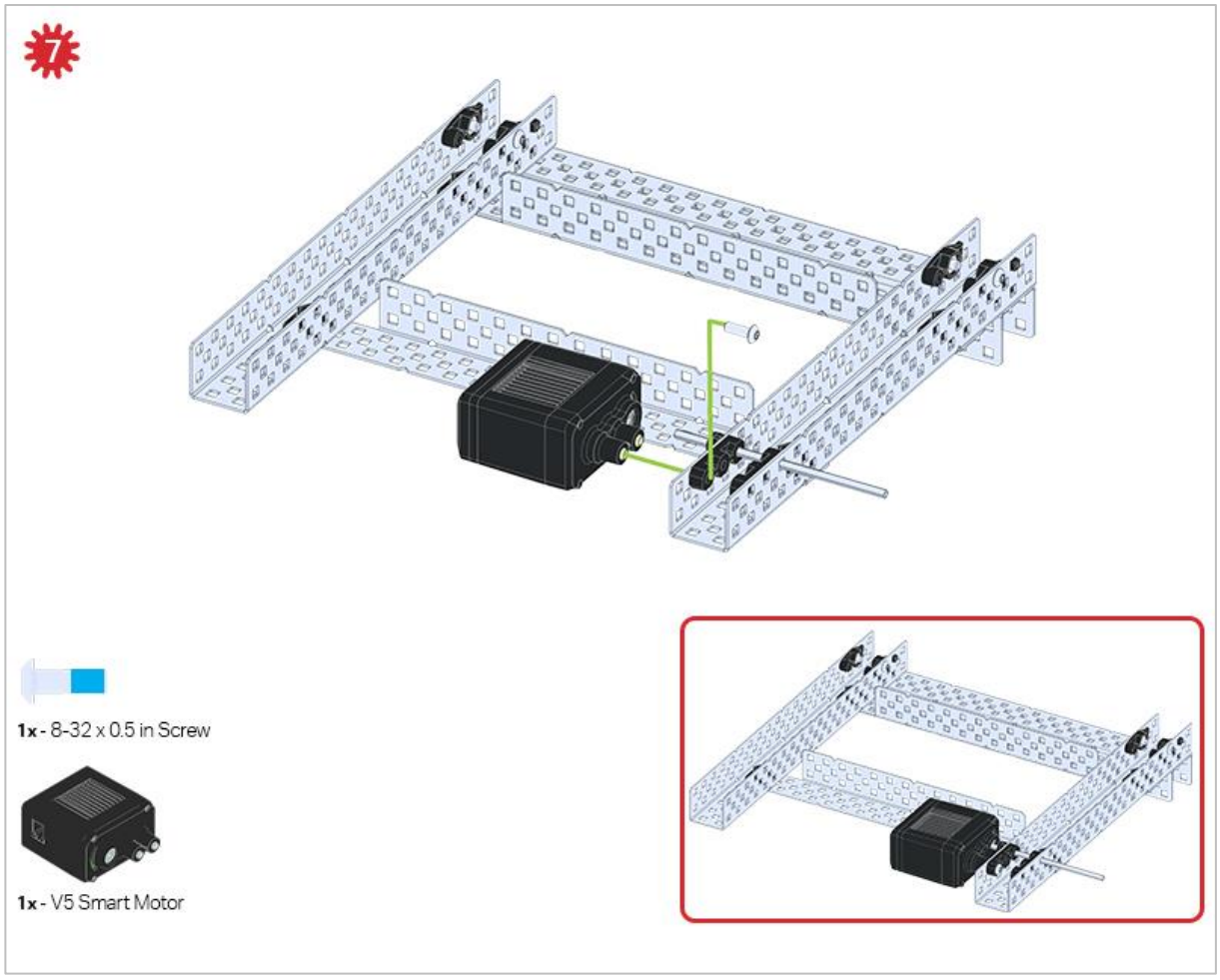
**2x** - 8-32 Nut

**2x** - 8-32 x 0.375 in Screw

**2x** - 1 Post Hex Nut Retainer w/ Bearing Flat

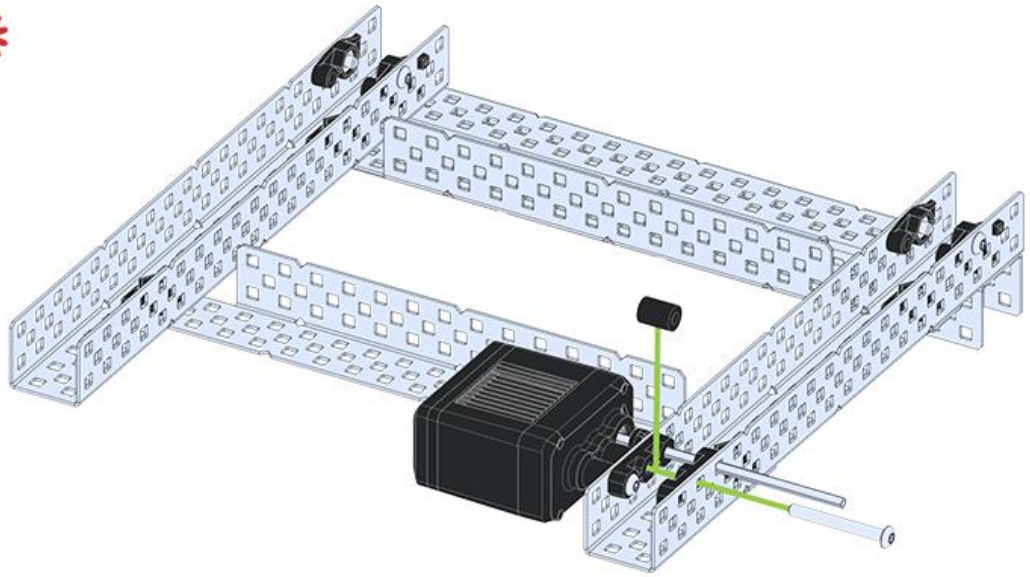



*Only one of the two sub-assemblies made in this step is used right now. The other will be used later in step 9.*



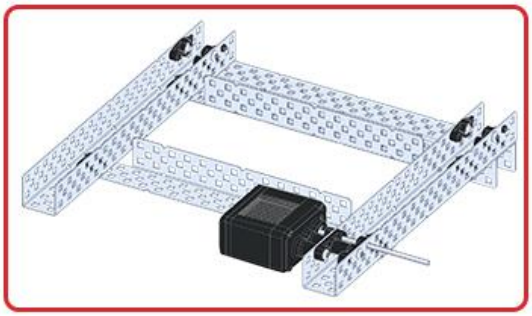

*Make sure your Smart Motors are oriented in the correct direction (screw holes facing the outside of the build and the shaft hole towards the inside).*

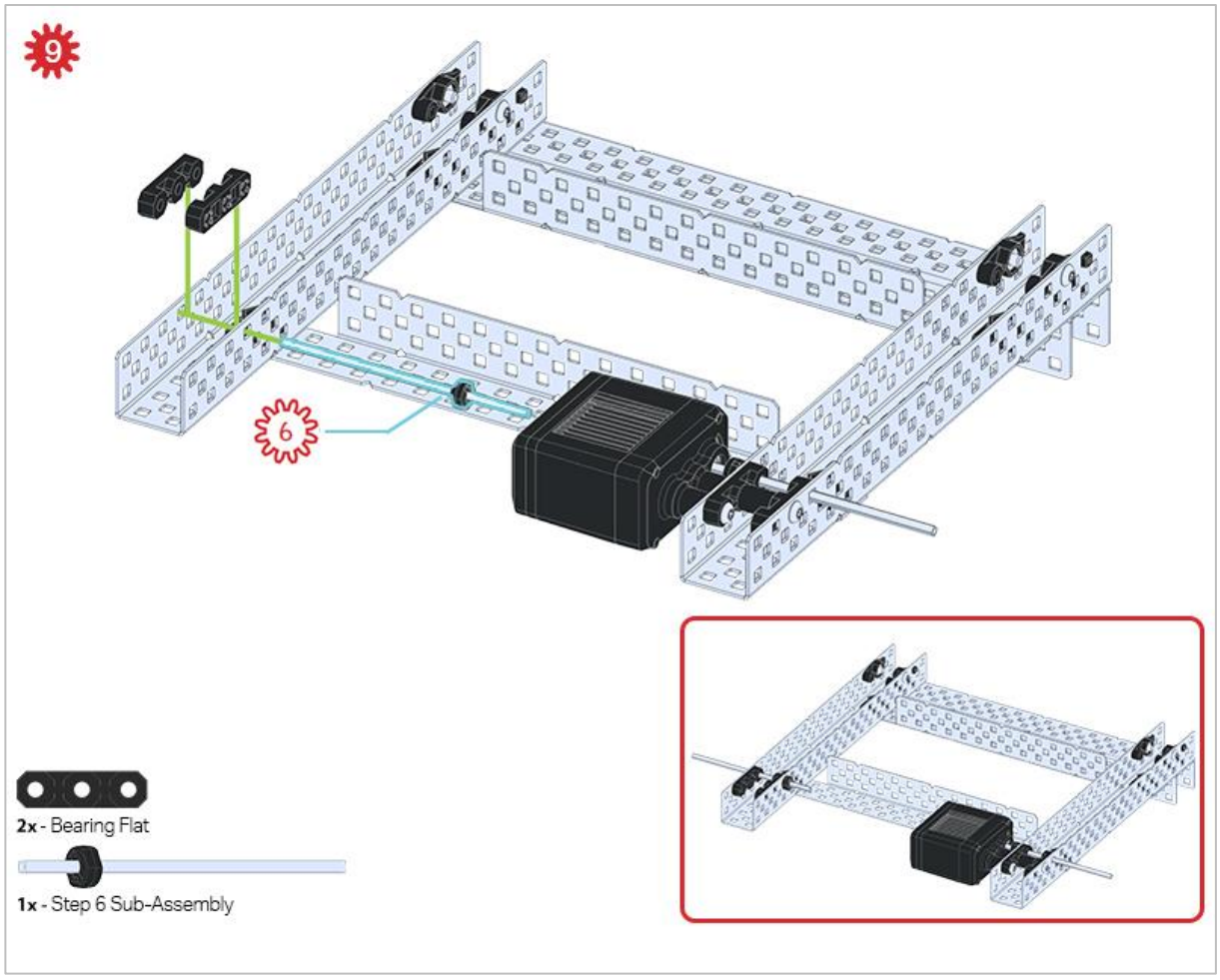
8

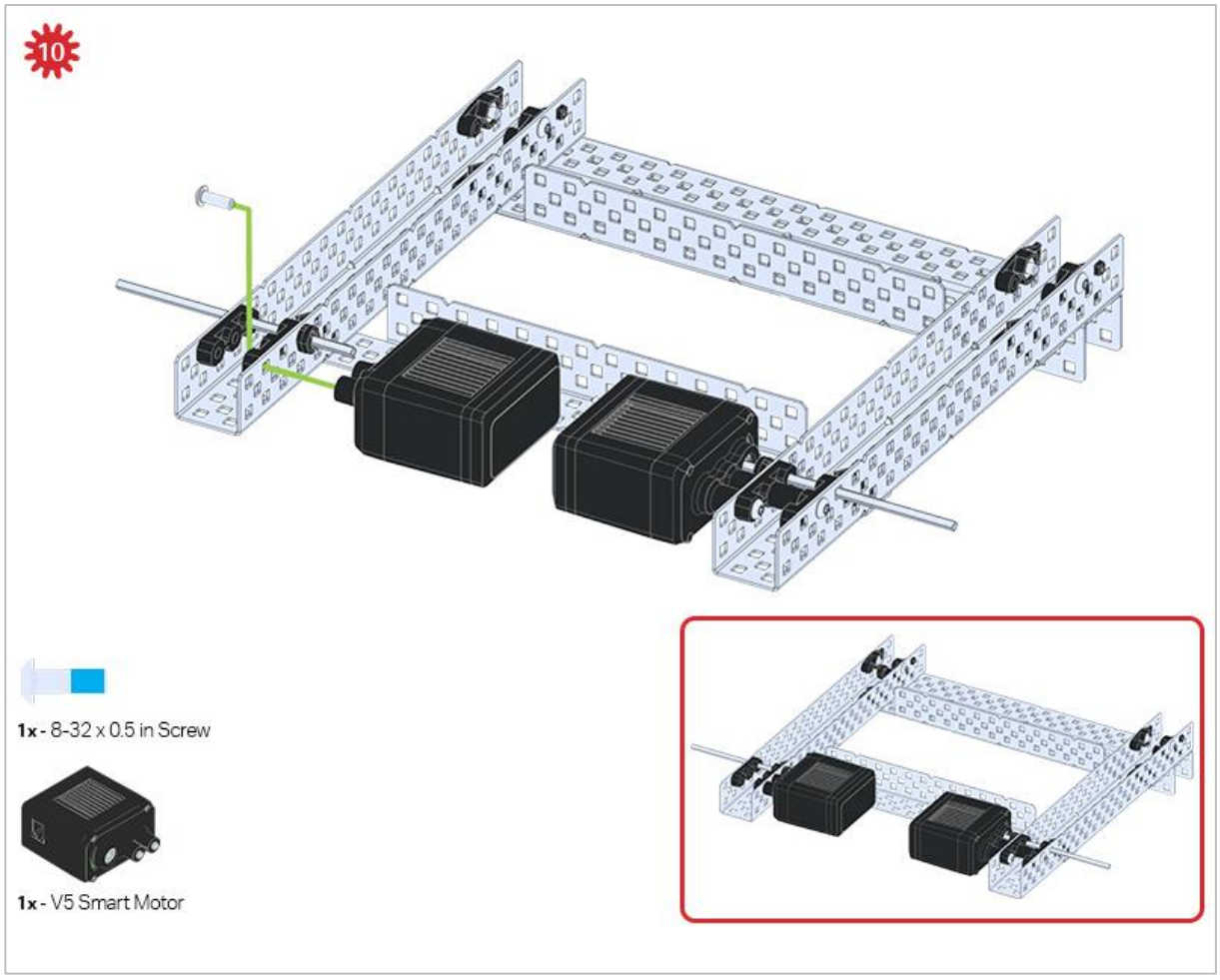


**1x** - 0.5 in Spacer

**1x** - 8-32 x 1.5 in Screw

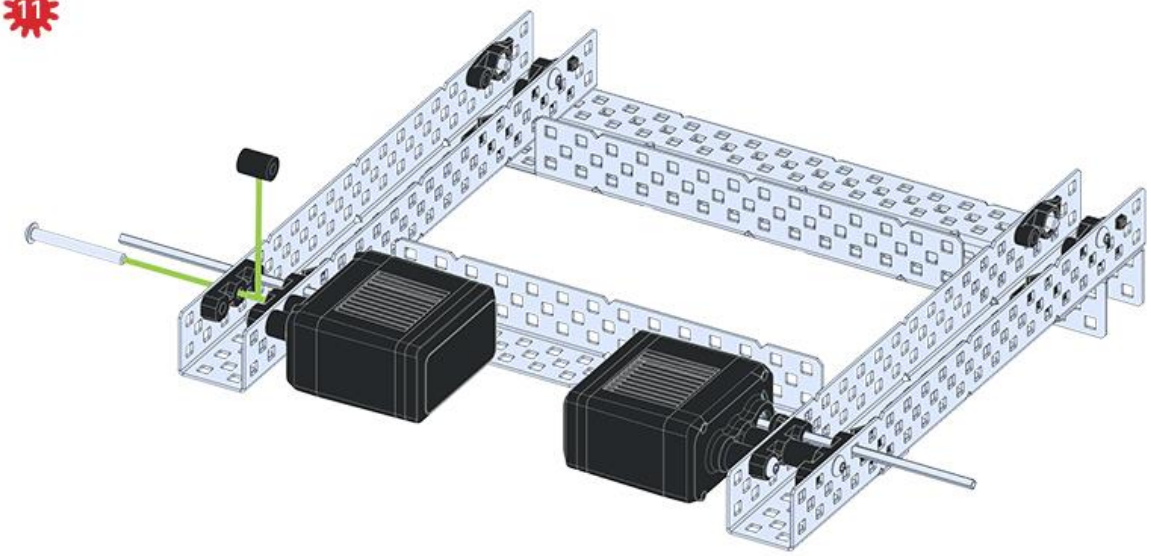



9
6
2x - Bearing Flat
1x - Step 6 Sub-Assembly



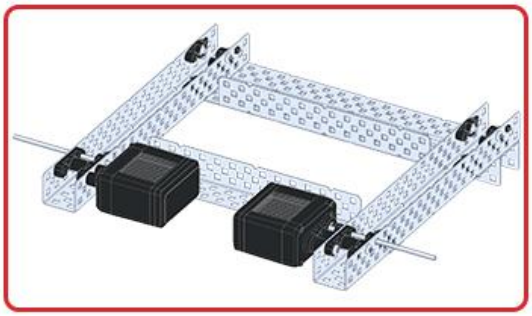

*Make sure your Smart Motors are oriented in the correct direction (screw holes facing the outside of the build and the shaft hole towards the inside).*

11

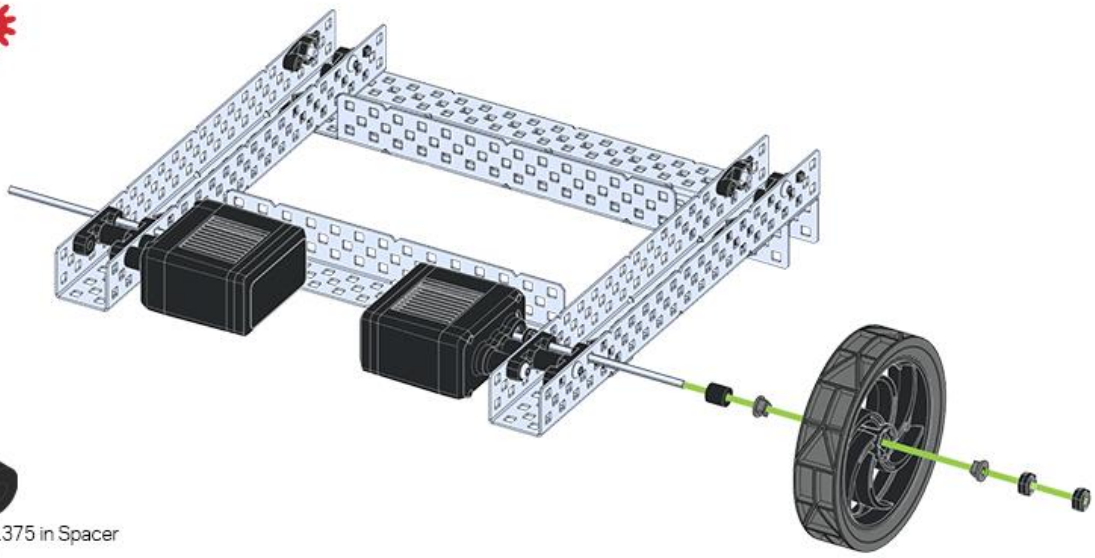


**1x** - 0.5 in Spacer

**1x** - 8-32 x 1.5 in Screw



12

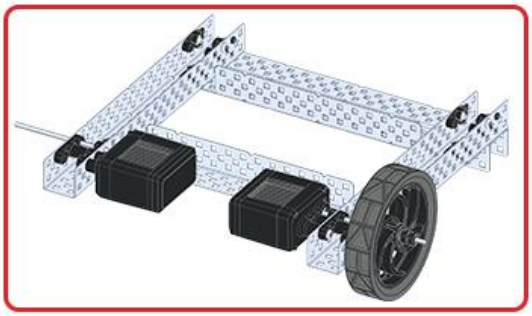


**1x** - 0.375 in Spacer

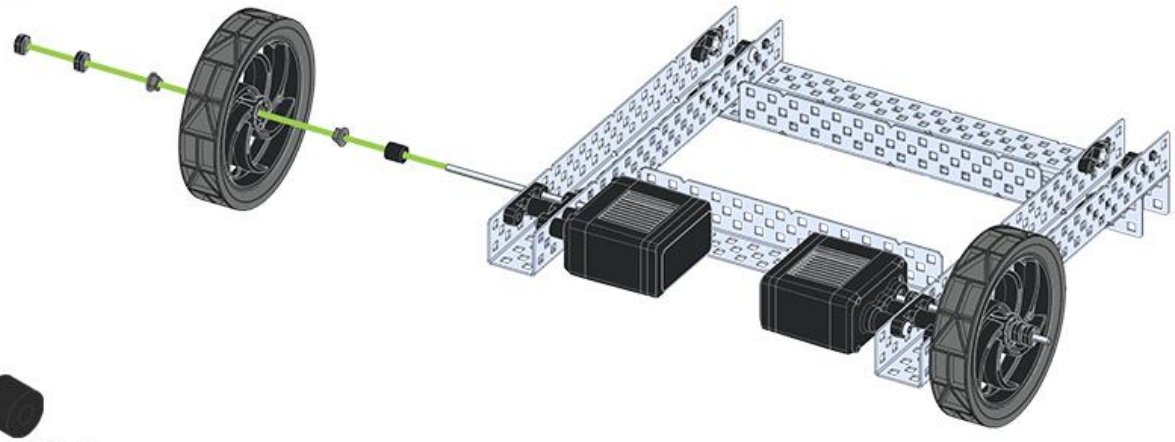
**2x** - Rubber Shaft Collar

**1x** - 4 in Wheel

**2x** - High Strength Shaft Insert



13



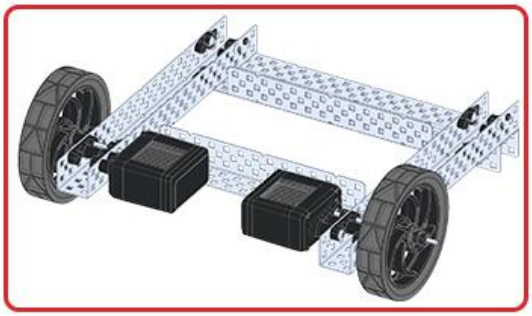


**1x** - 0.375 in Spacer

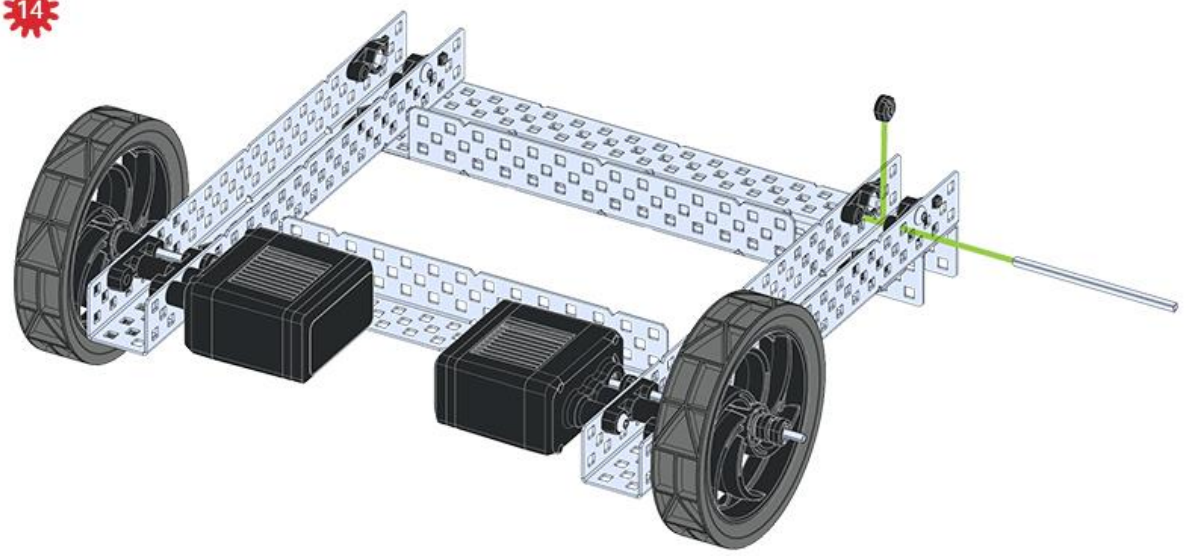
**2x** - Rubber Shaft Collar

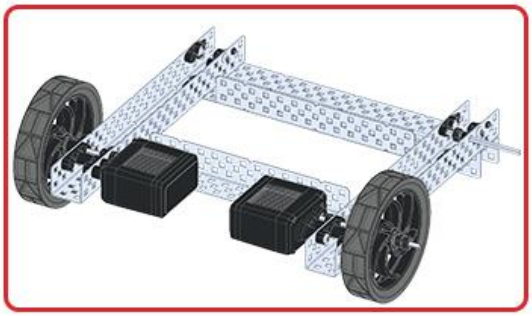
**1x** - 4 in Wheel

**2x** - High Strength Shaft Insert



14

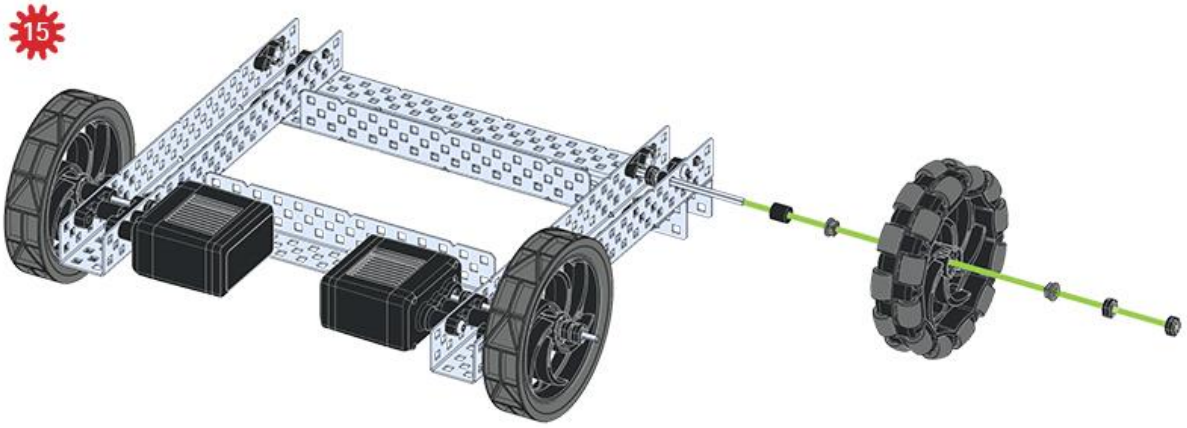




**1x** - Rubber Shaft Collar

**1x** - 3 in Shaft

15

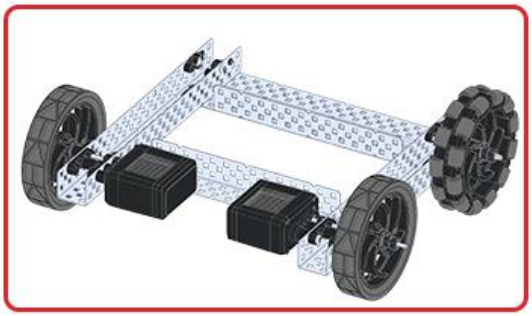


**1x** - 0.375 in Spacer

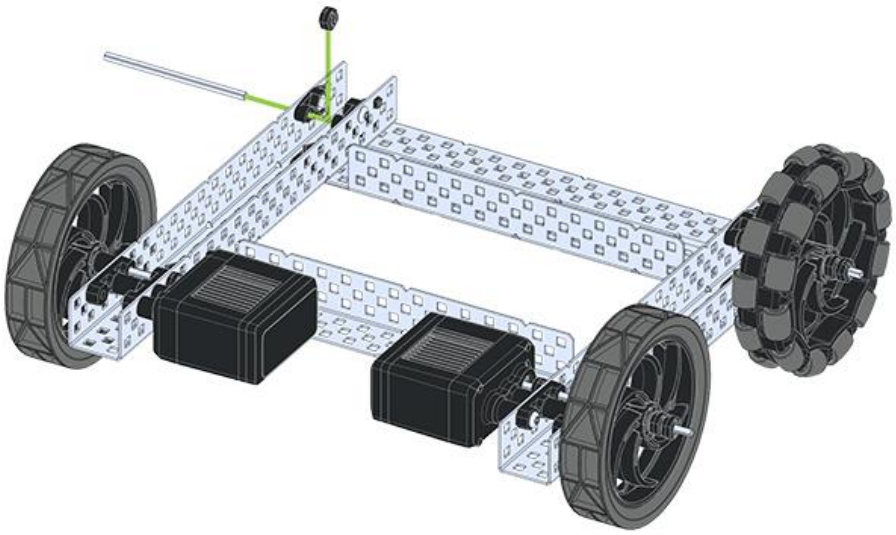
**2x** - Rubber Shaft Collar

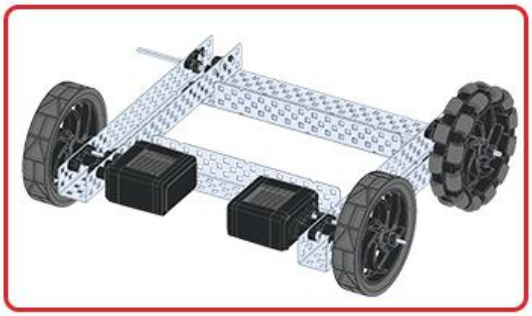
**1x** - 4 in Omni Wheel

**2x** - High Strength Shaft Insert



16

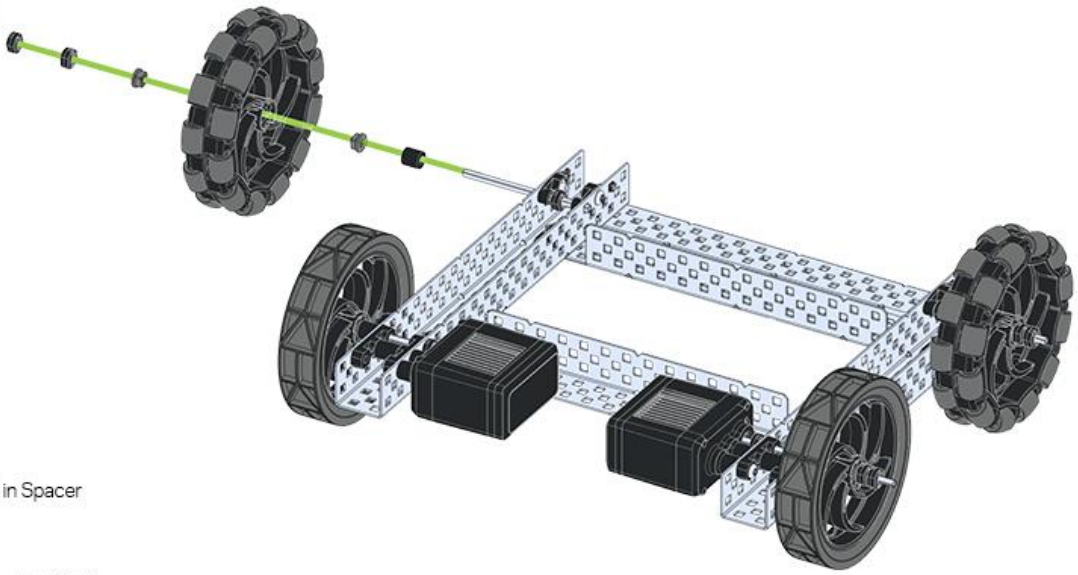




**1x** - Rubber Shaft Collar

**1x** - 3 in Shaft

17

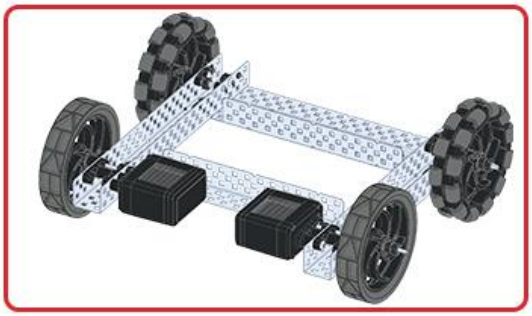


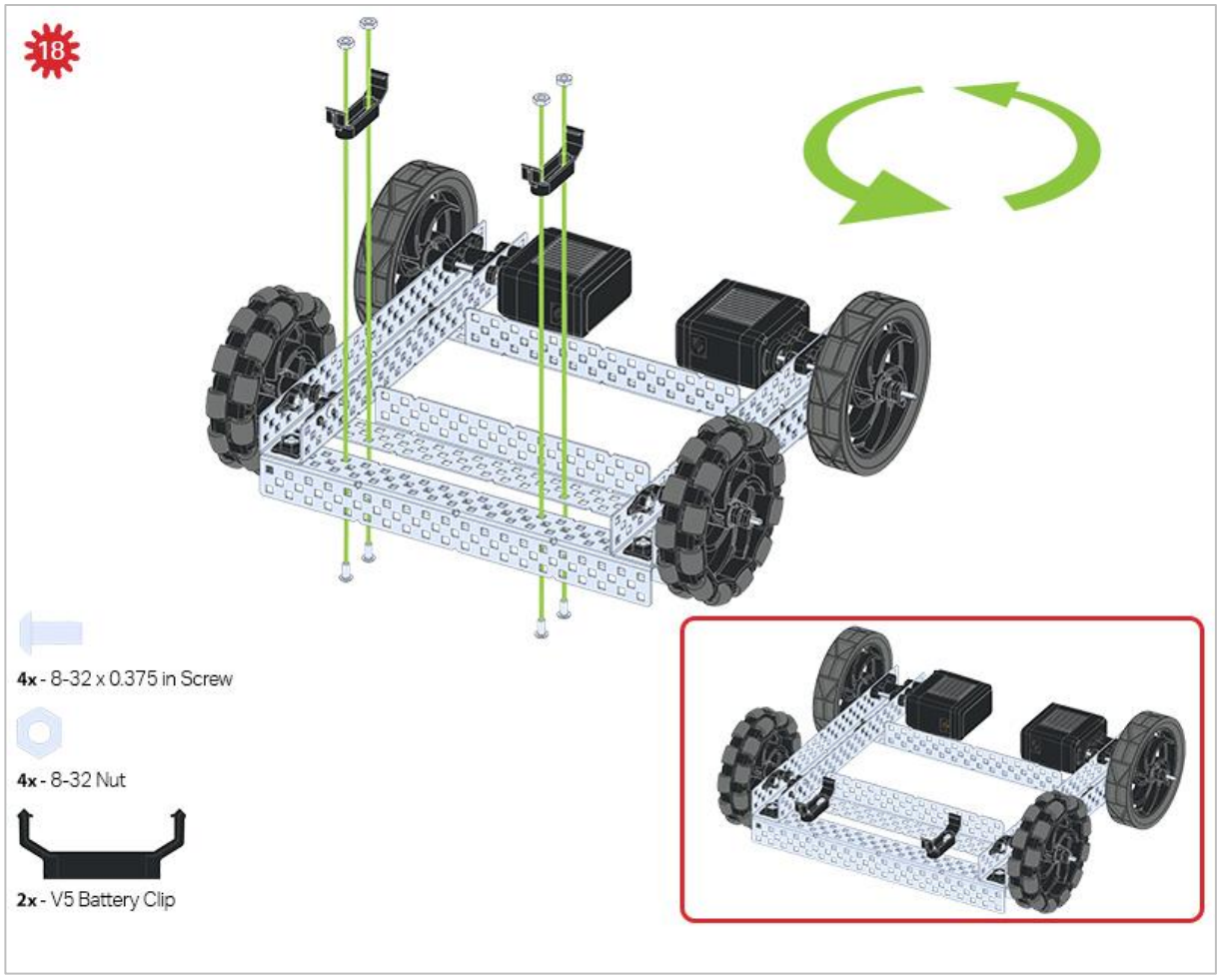
**1x** - 0.375 in Spacer

**2x** - Rubber Shaft Collar

**1x** - 4 in Omni Wheel

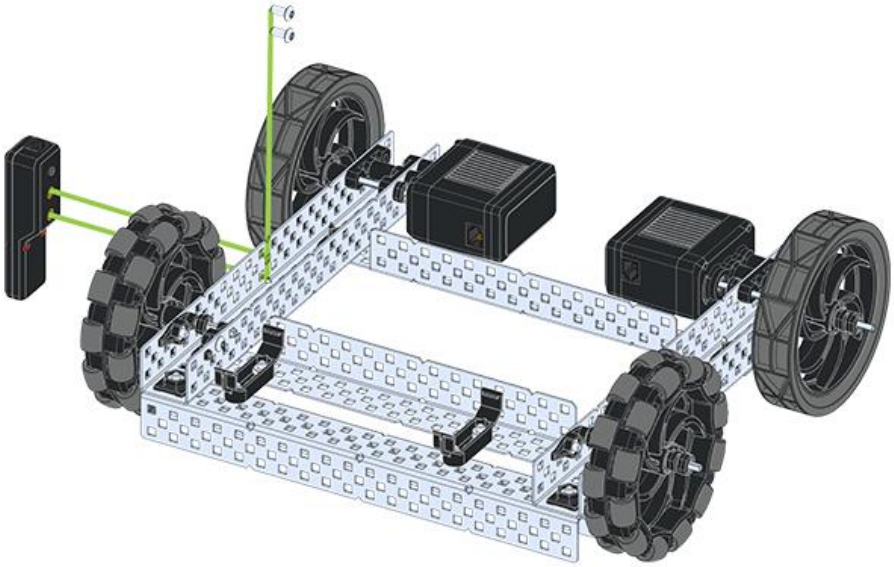
**2x** - High Strength Shaft Insert






*The green icon indicates that the build needs to be rotated (180 degrees).*

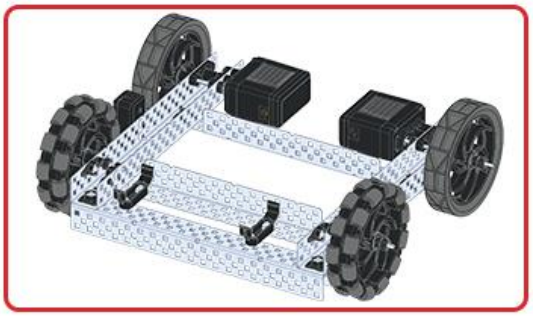
19

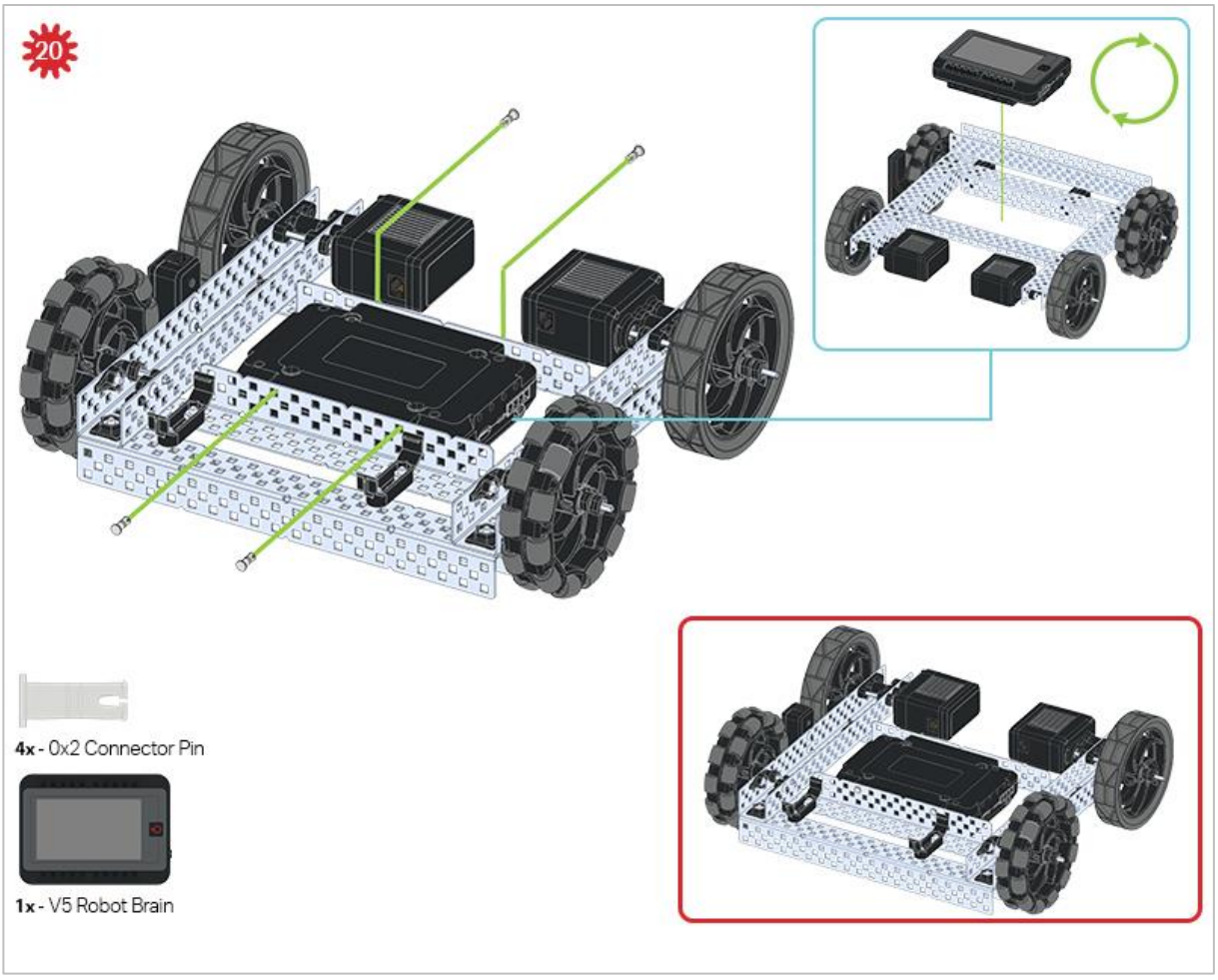


**1x** - V5 Radio

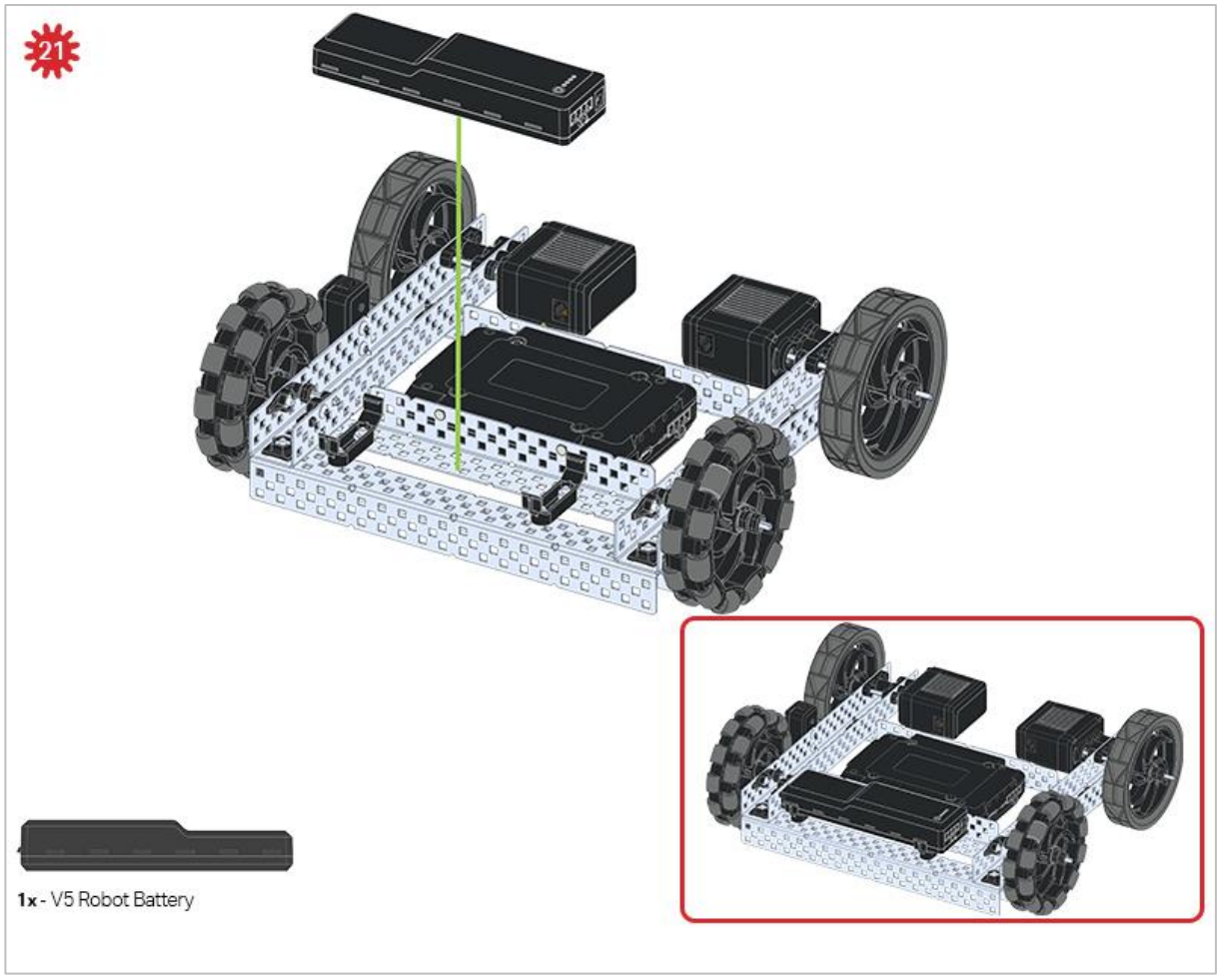


**2x** - 8-32 x 0.375 in Screw

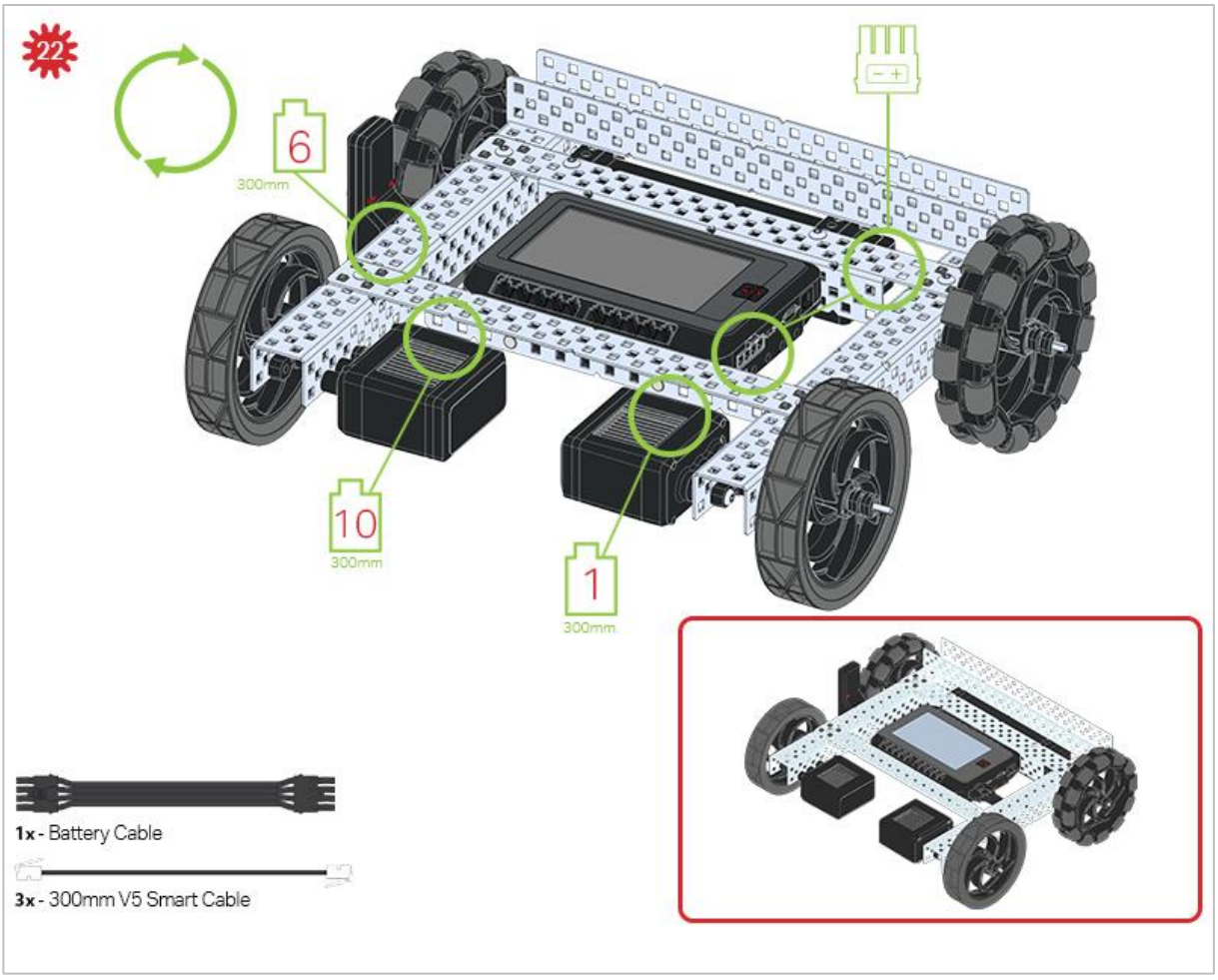




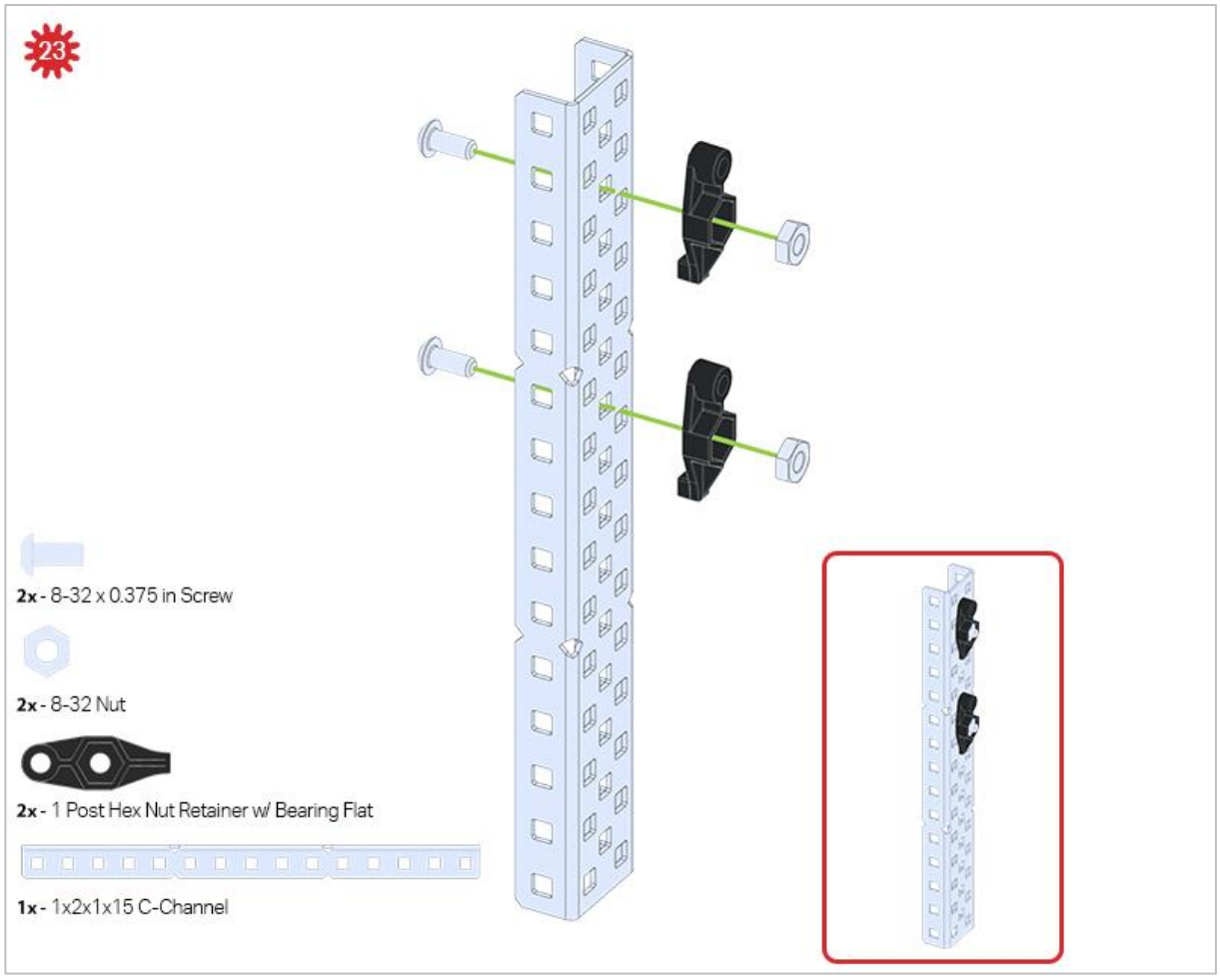

*The blue call out shows what the orientation of the Robot Brain should be if the build were flipped right side up. Make sure the 3 wire ports on the Robot Brain are facing the V5 Radio!*

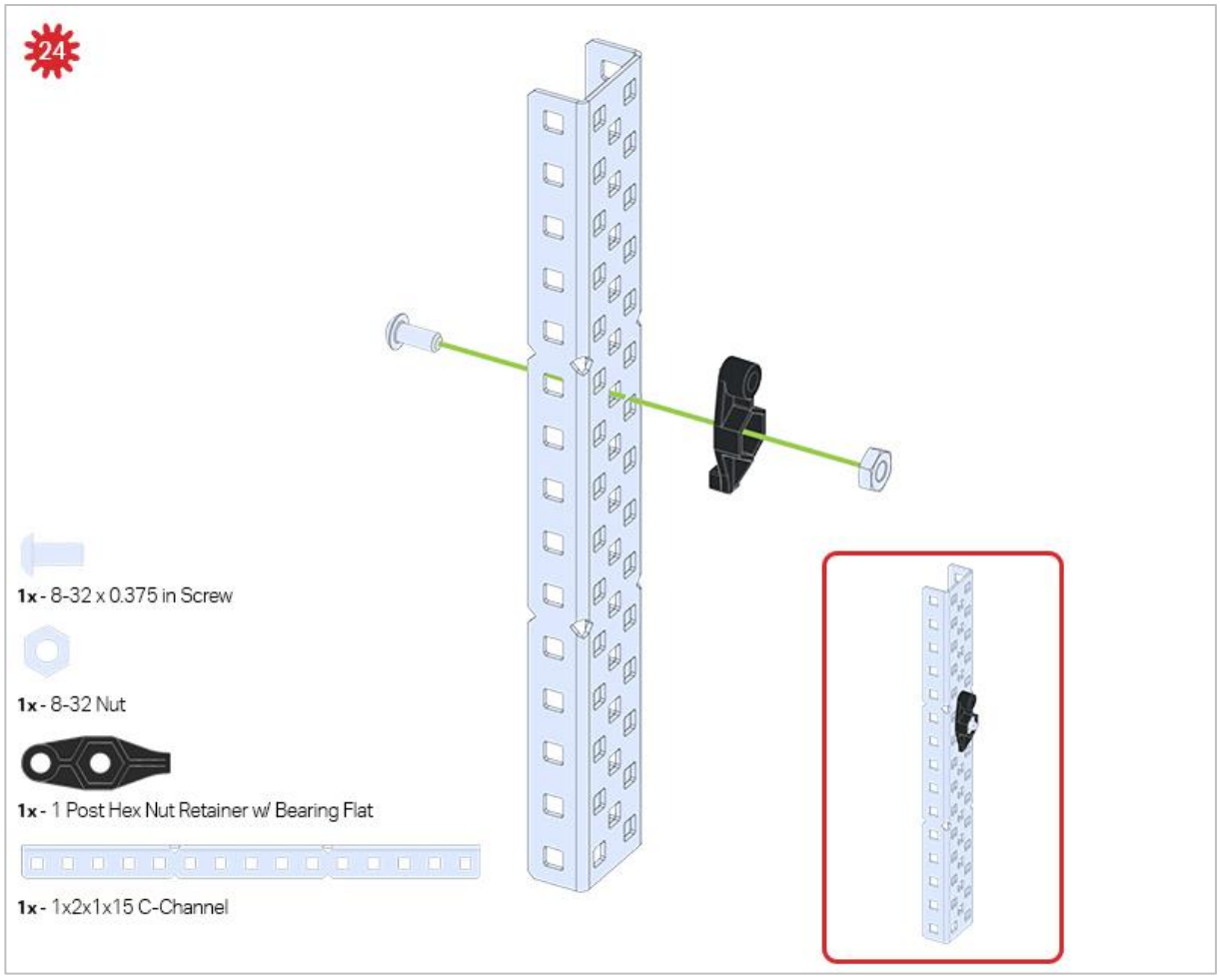



**1x** - V5 Robot Battery

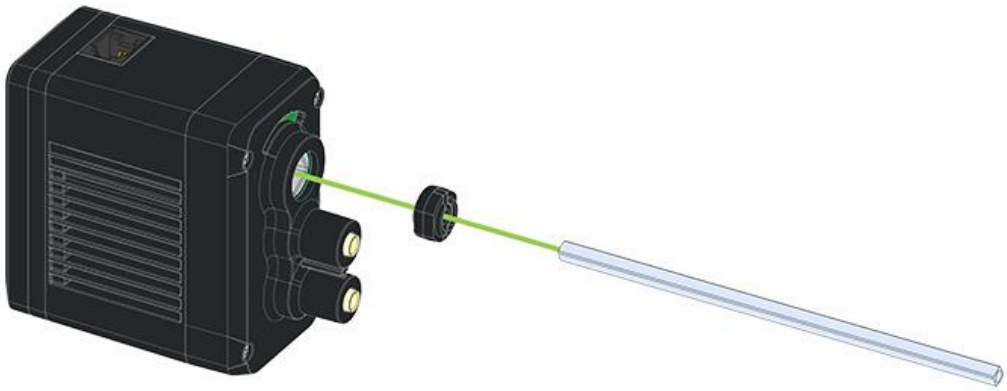



*The green call outs indicate which port on the Robot Brain to plug each device into using their respective cable.*


23
2x - 8-32 x 0.375 in Screw
2x - 8-32 Nut
2x - 1 Post Hex Nut Retainer w/ Bearing Flat
1x - 1x2x1x15 C-Channel


24
1x - 8-32 x 0.375 in Screw
1x - 8-32 Nut
1x - 1 Post Hex Nut Retainer w/ Bearing Flat
1x - 1x2x1x15 C-Channel

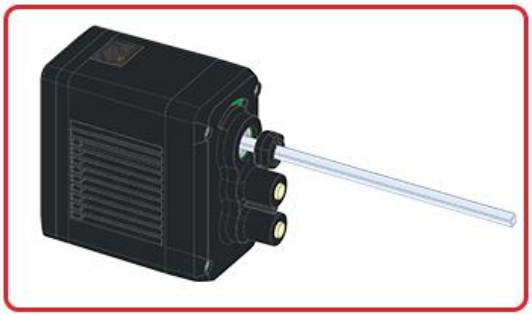
25

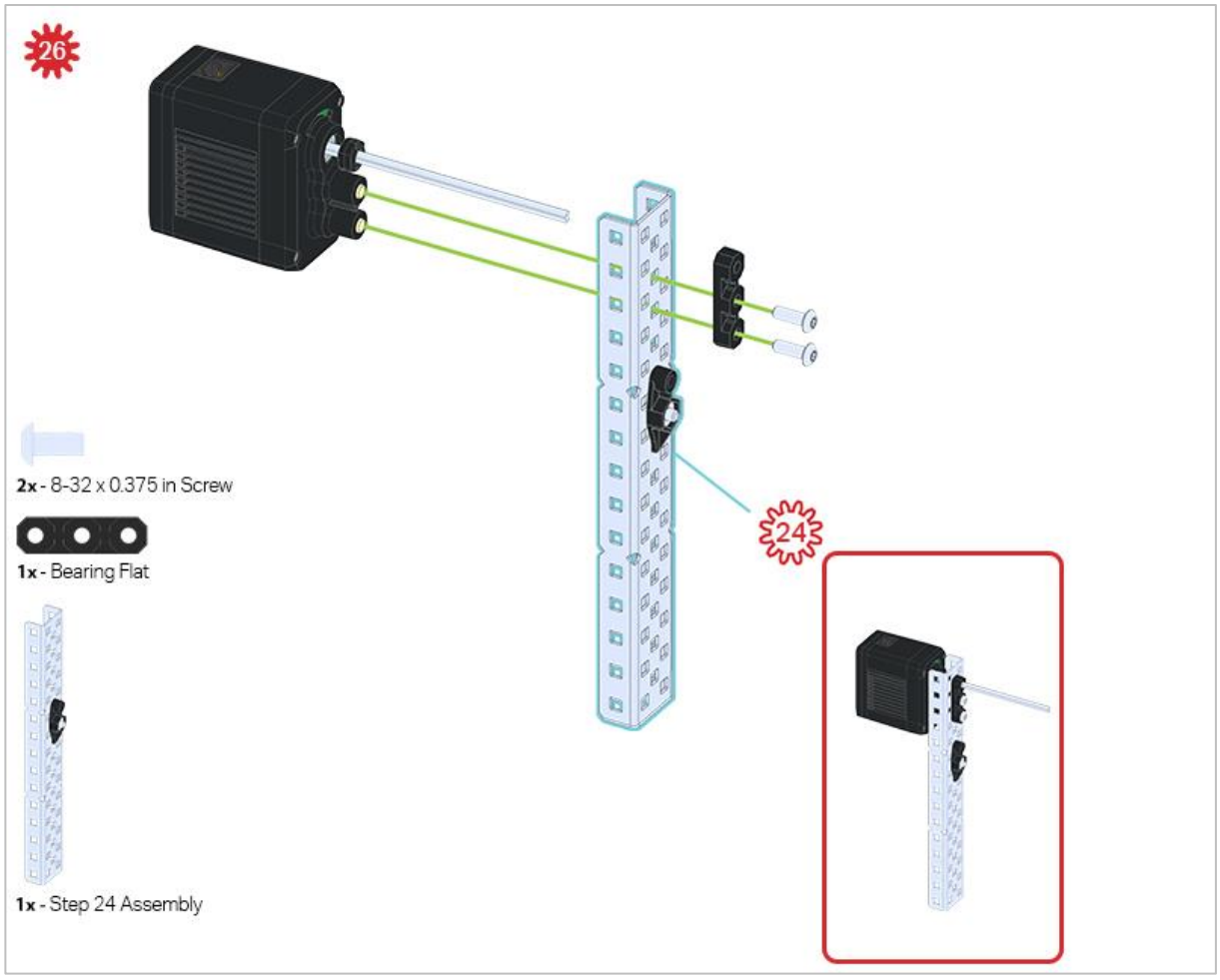


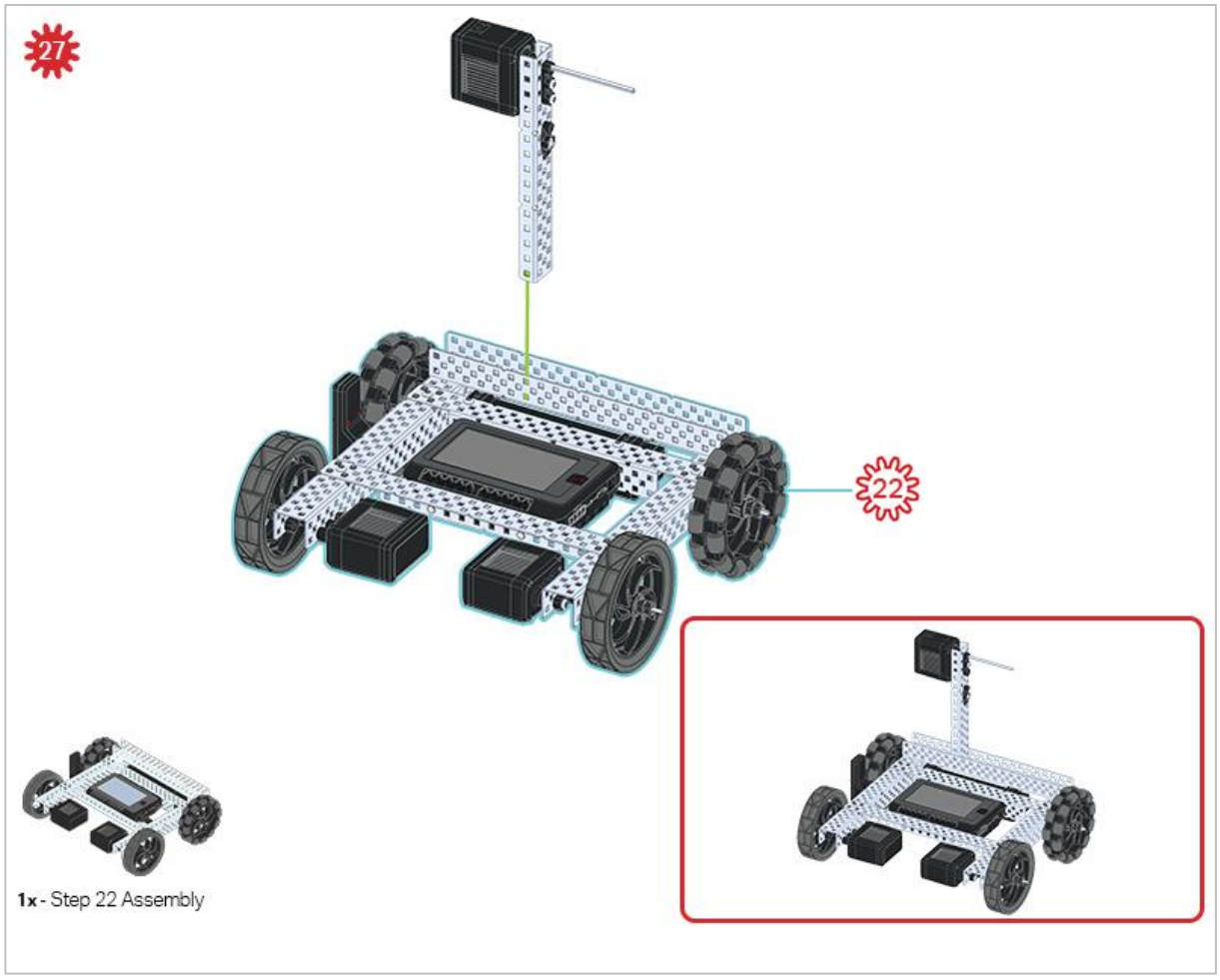
**1x** - Rubber Shaft Collar

**1x** - V5 Smart Motor

**1x** - 4 in Shaft




26
2x - 8-32 x 0.375 in Screw
1x - Bearing Flat
1x - Step 24 Assembly
24


27
22
1x - Step 22 Assembly

28

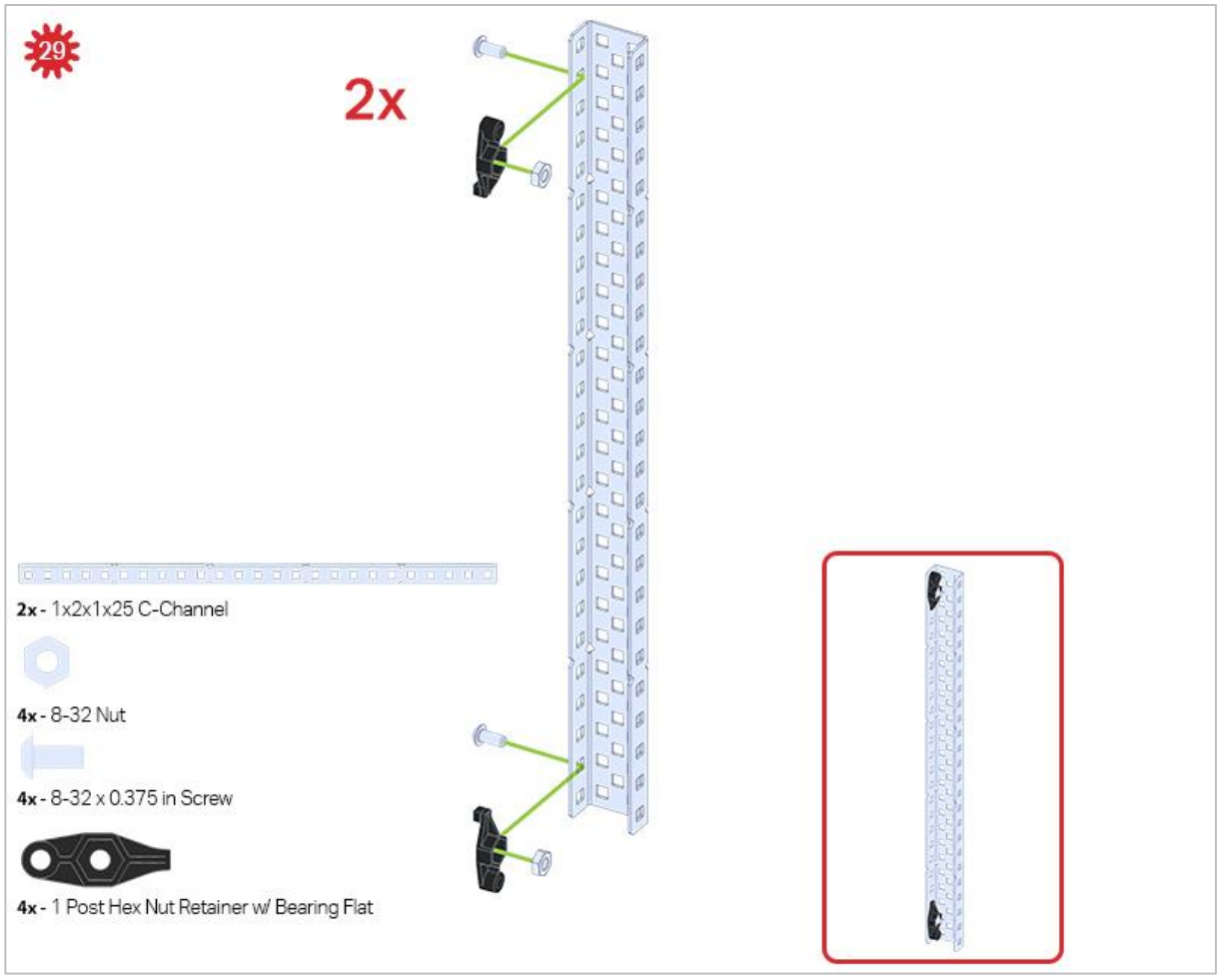


**1x** - 8-32 Nut

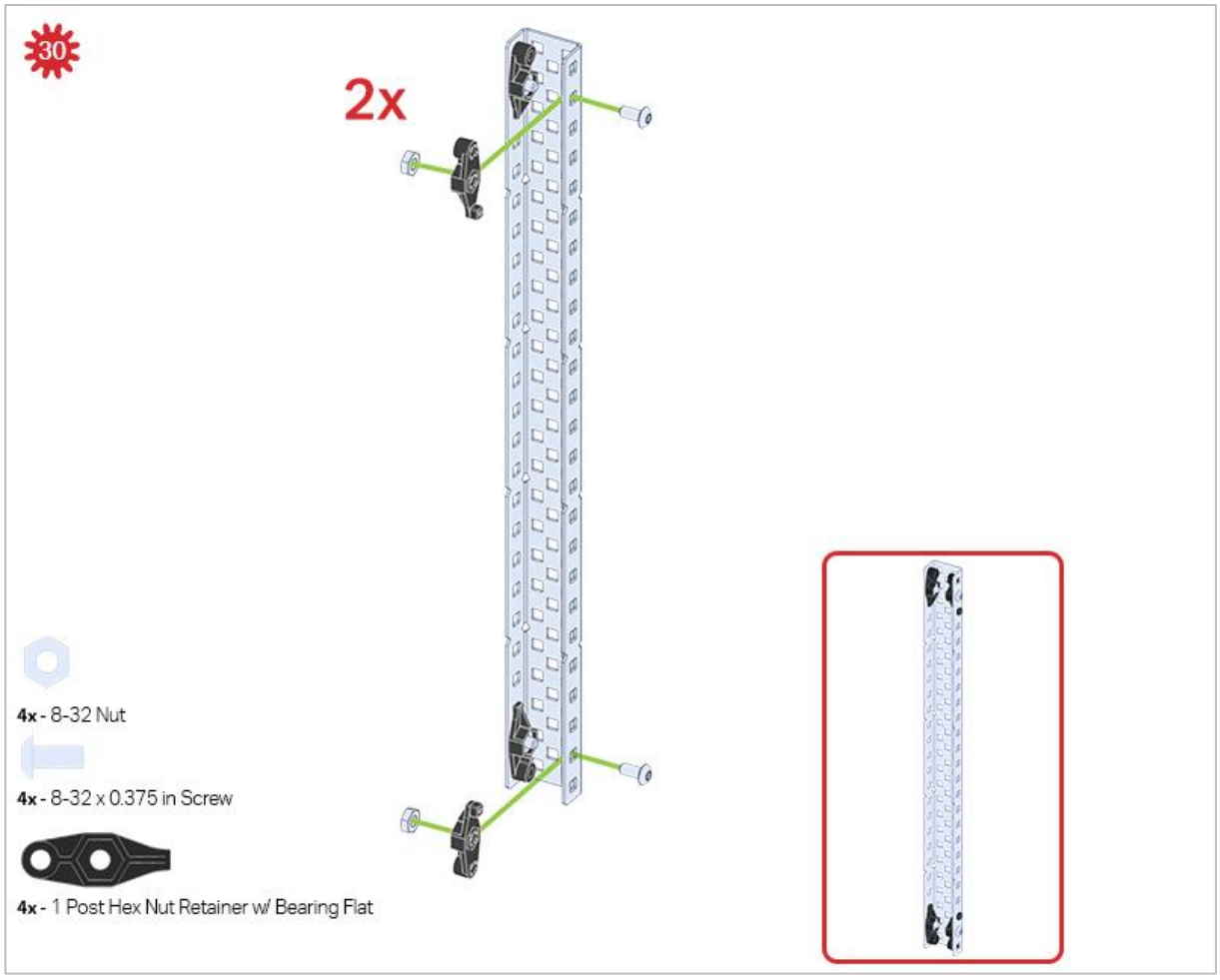
**1x** - 0.875 in Spacer

**2x** - 1 Post Hex Nut Retainer

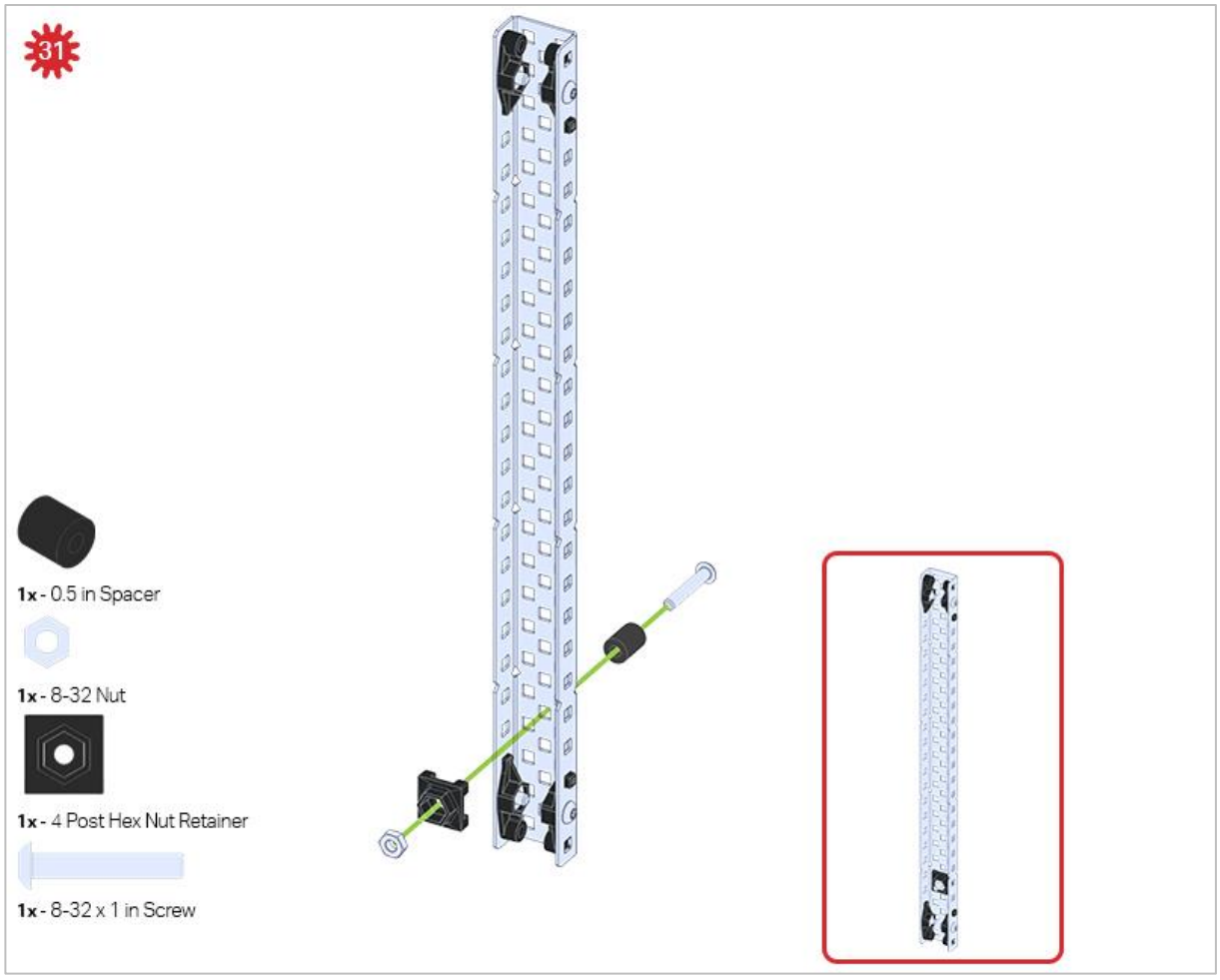
**1x** - 8-32 x 1.5 in Screw




*Be sure to make two assemblies in this step!*




*This step adds onto the two assemblies started in Step 29.*



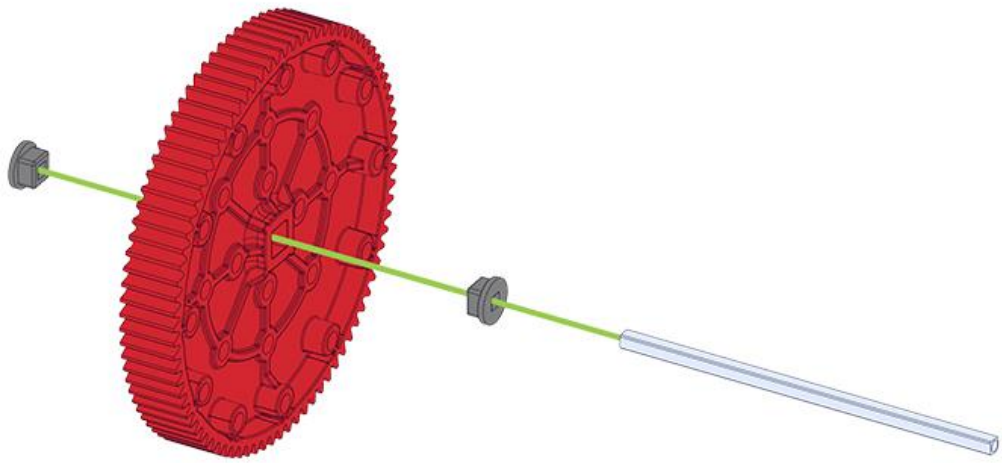

*Make sure to add this to only one of the two sub-assemblies you just made.*

32

**1x** - 8-32 Nut

**1x** - 1 Post Hex Nut Retainer

33

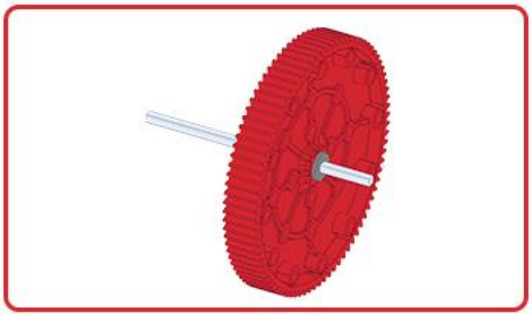


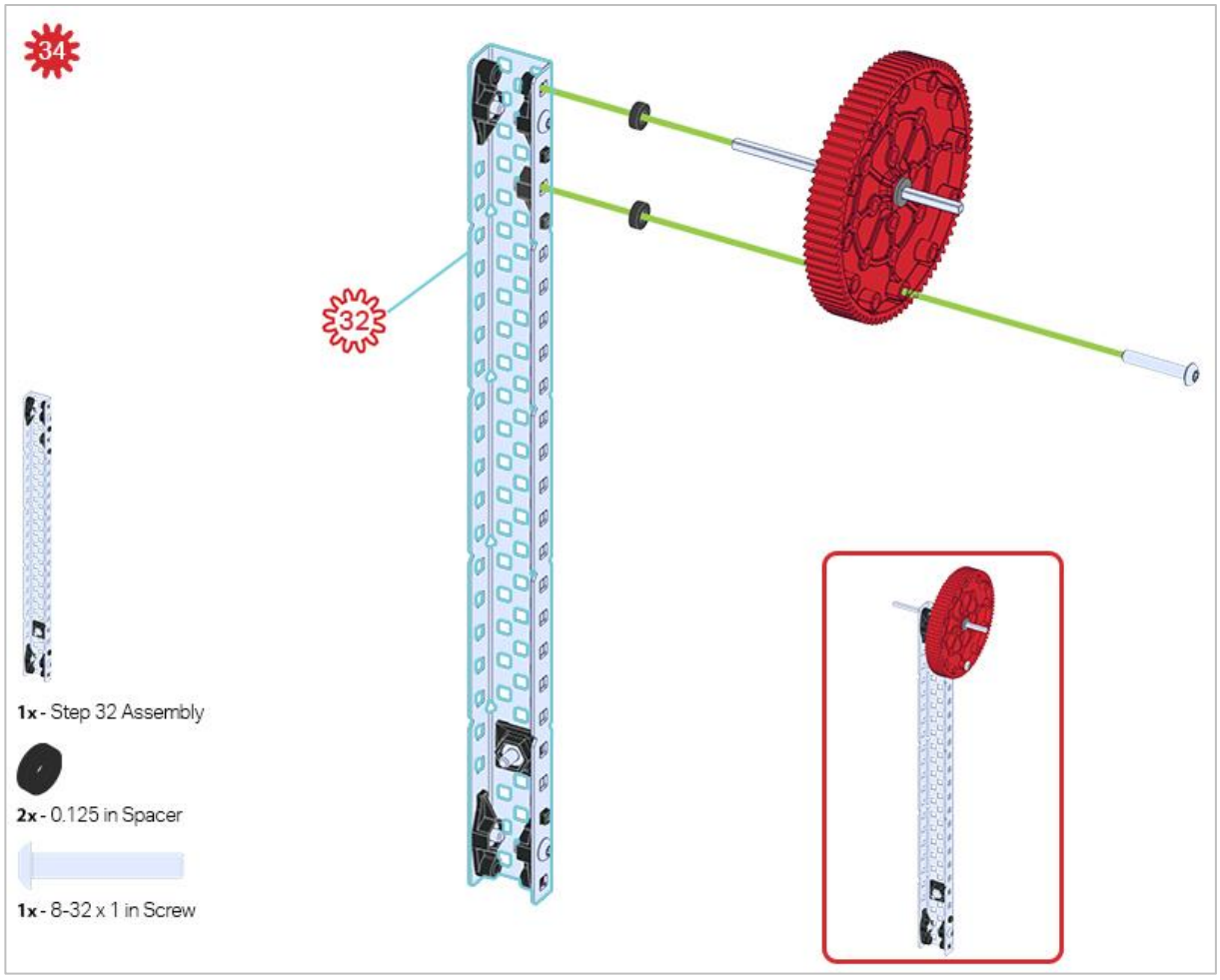
**1x** - High Strength 84 Tooth Gear

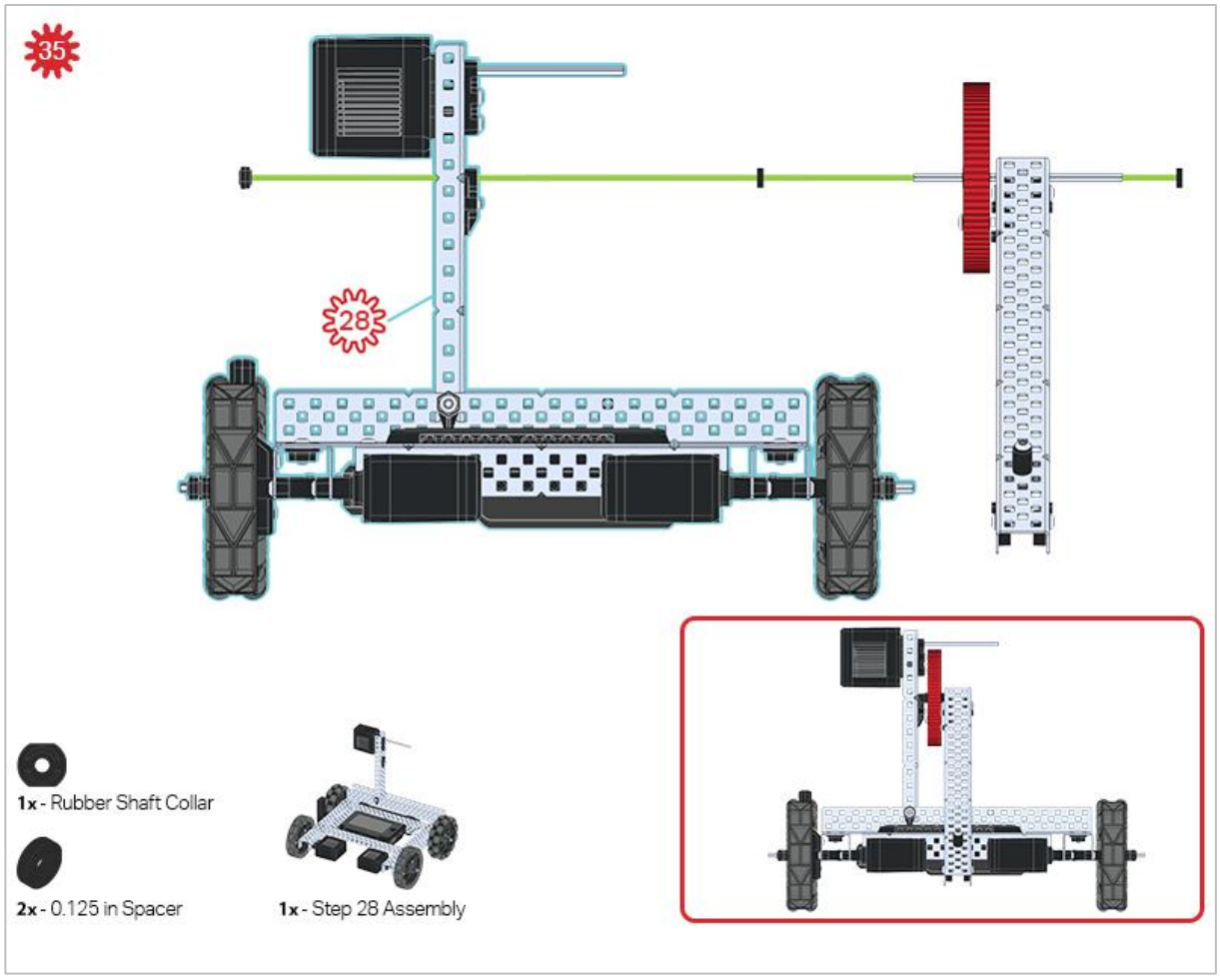
**2x** - High Strength Shaft Insert

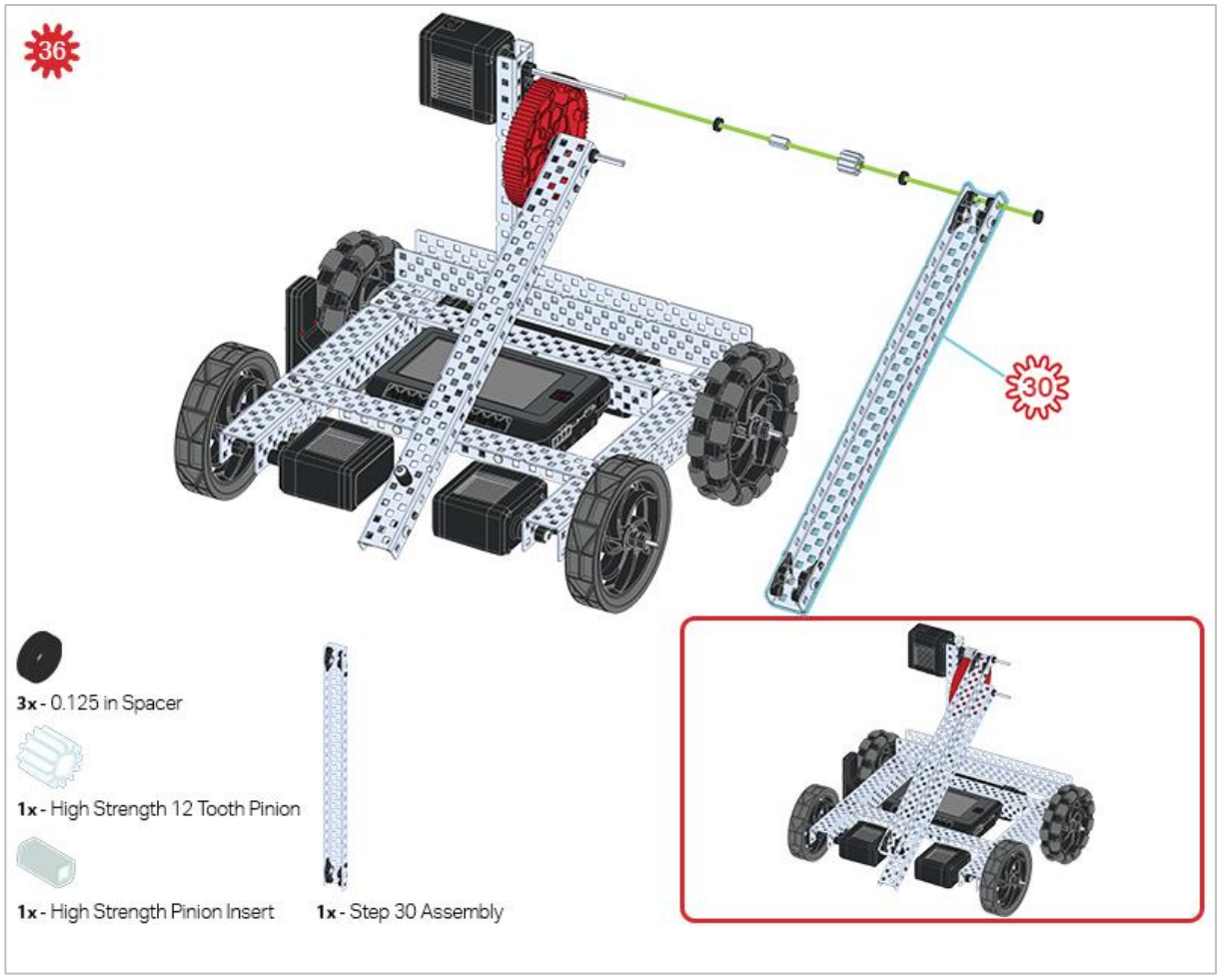


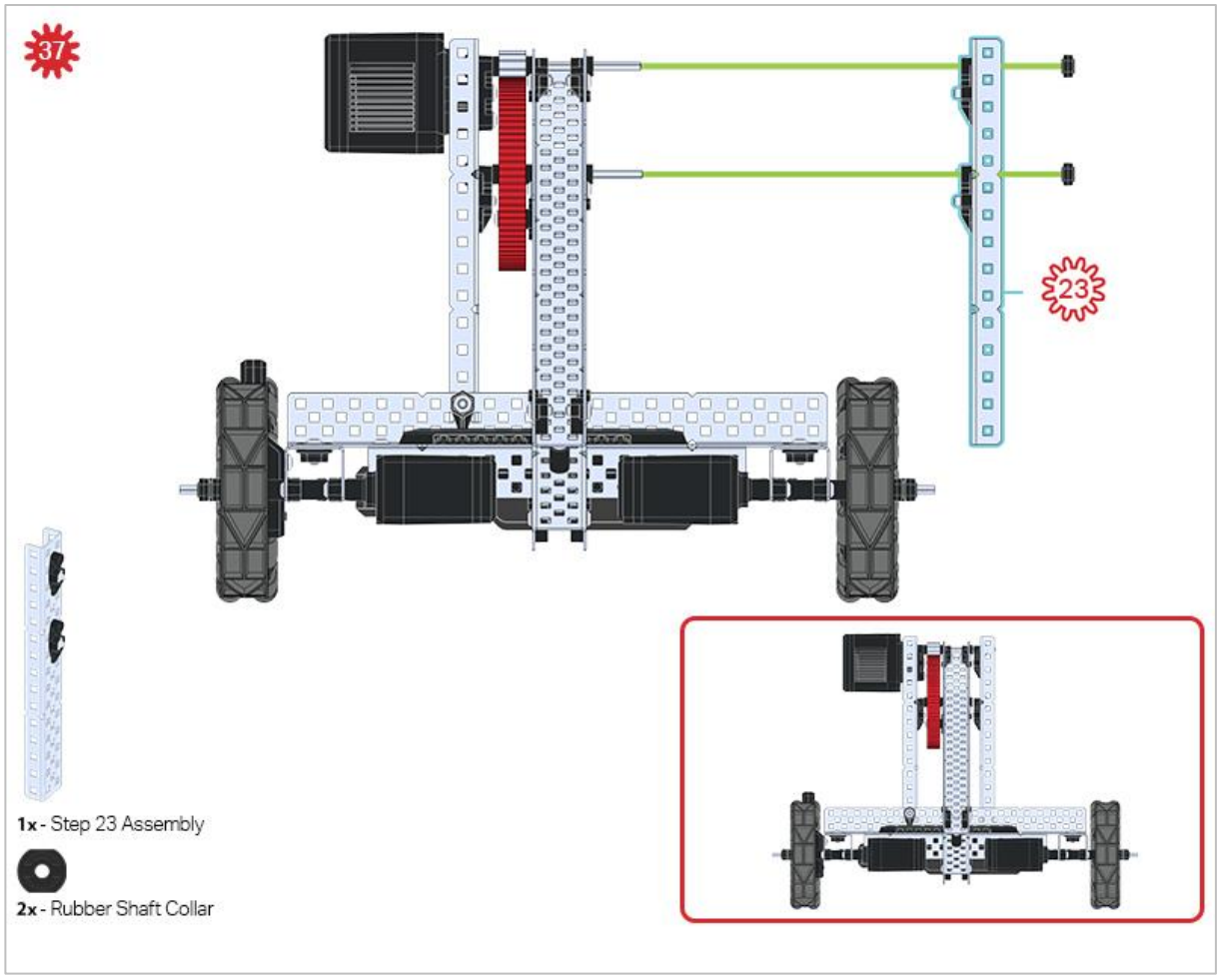
**1x** - 3.5 in Shaft




34
32
1x - Step 32 Assembly
2x - 0.125 in Spacer
1x - 8-32 x 1 in Screw


35
28
1x - Rubber Shaft Collar
2x - 0.125 in Spacer
1x - Step 28 Assembly

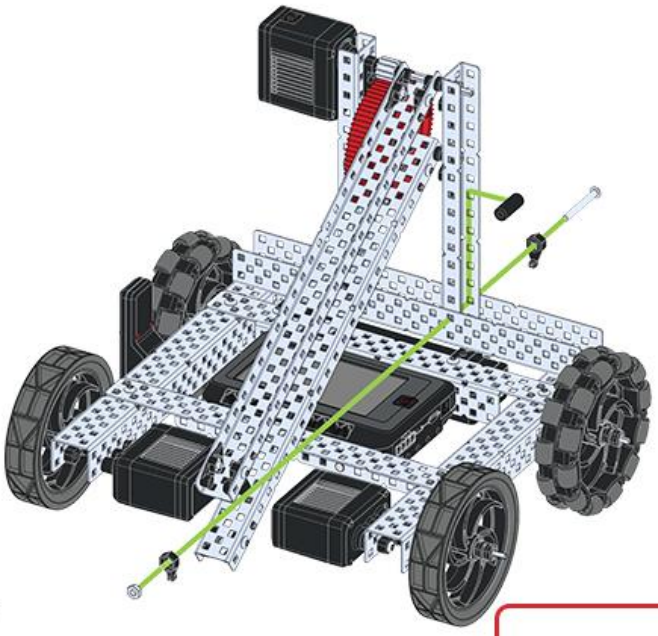

36
30
3x - 0.125 in Spacer
1x - High Strength 12 Tooth Pinion
1x - High Strength Pinion Insert
1x - Step 30 Assembly




**1x** - Step 23 Assembly

**2x** - Rubber Shaft Collar

38



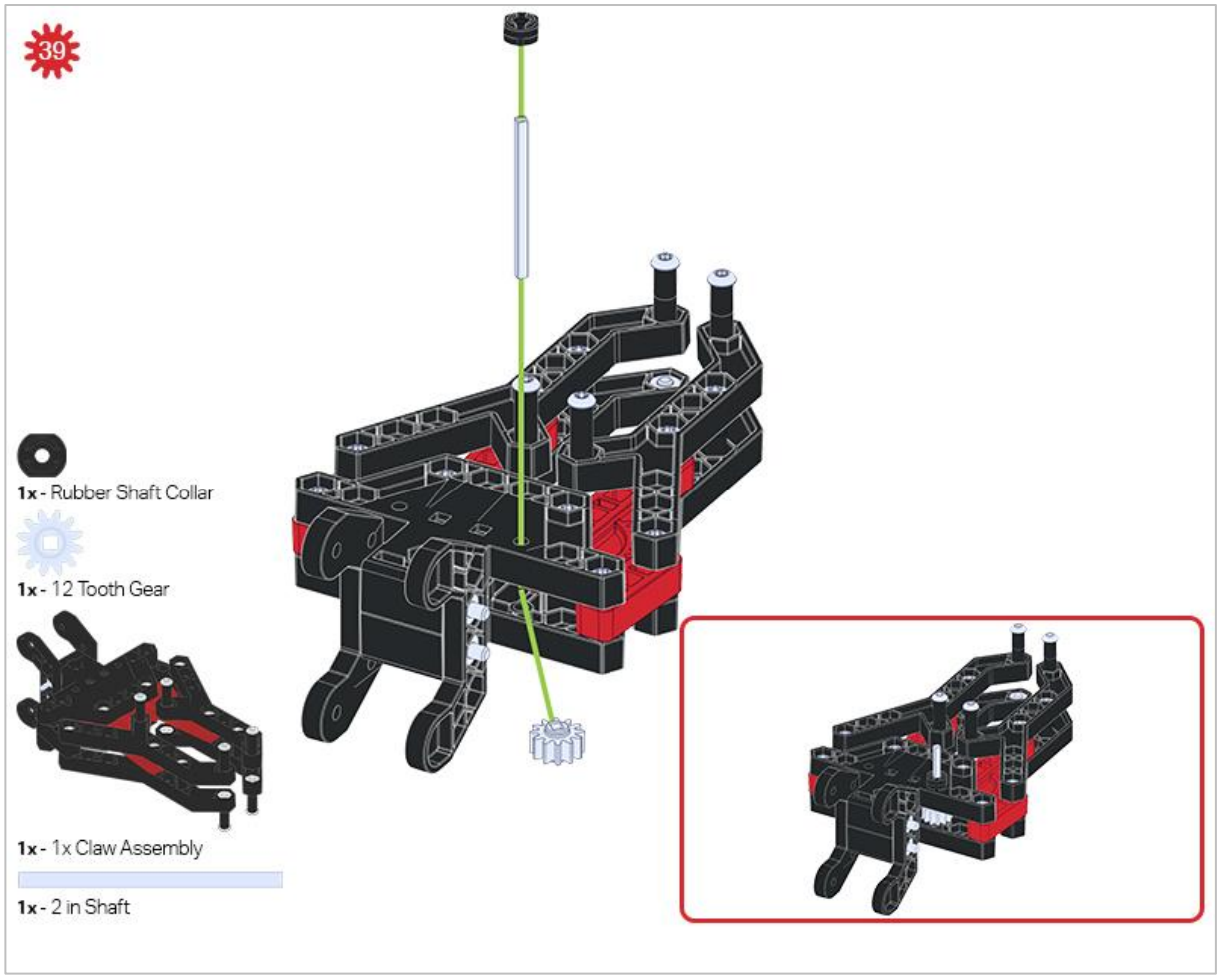
**1x** - 0.875 in Spacer



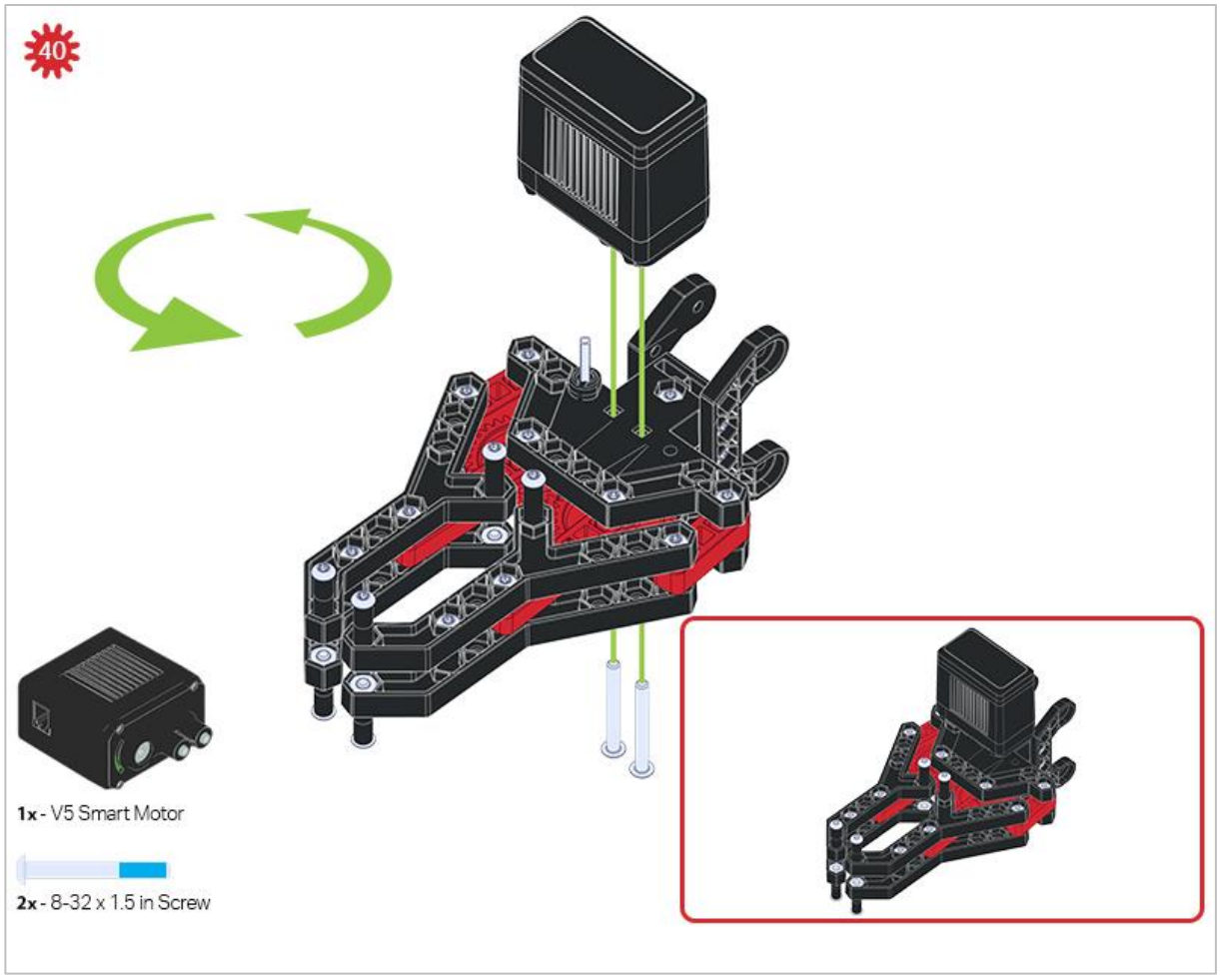
**1x** - 8-32 Nut

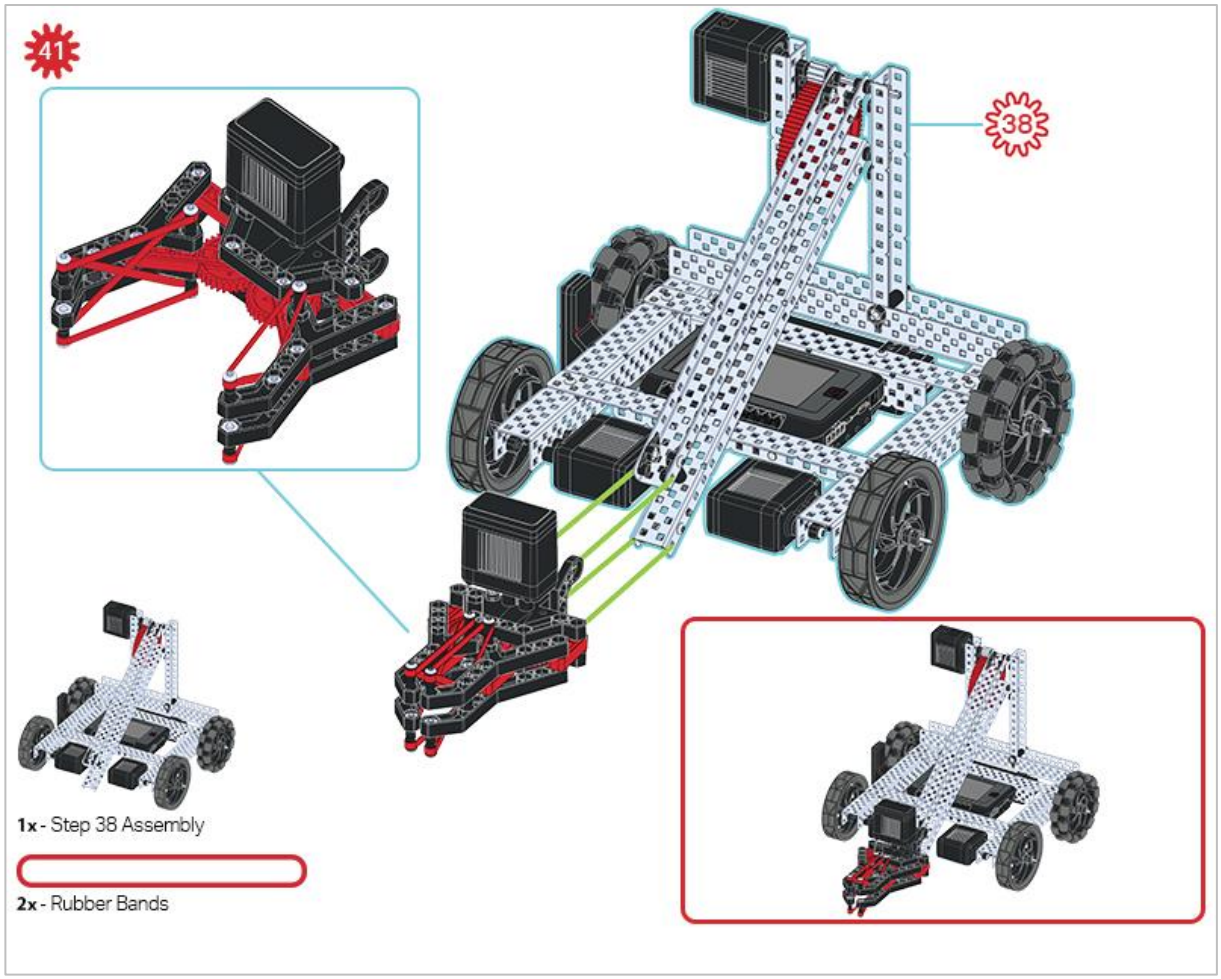
**2x** - 1 Post Hex Nut Retainer

**1x** - 8-32 x 1.5 in Screw



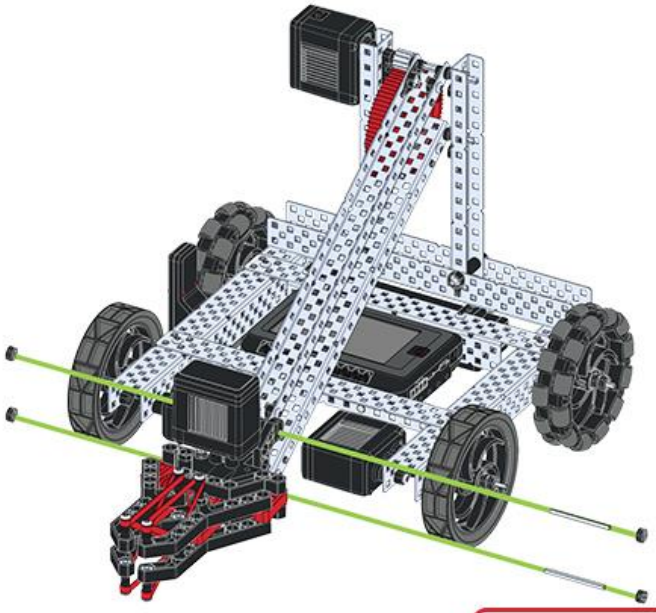

*Make sure the 12- tooth gear is installed on the right side of the claw.*


40
1x - V5 Smart Motor
2x - 8-32 x 1.5 in Screw



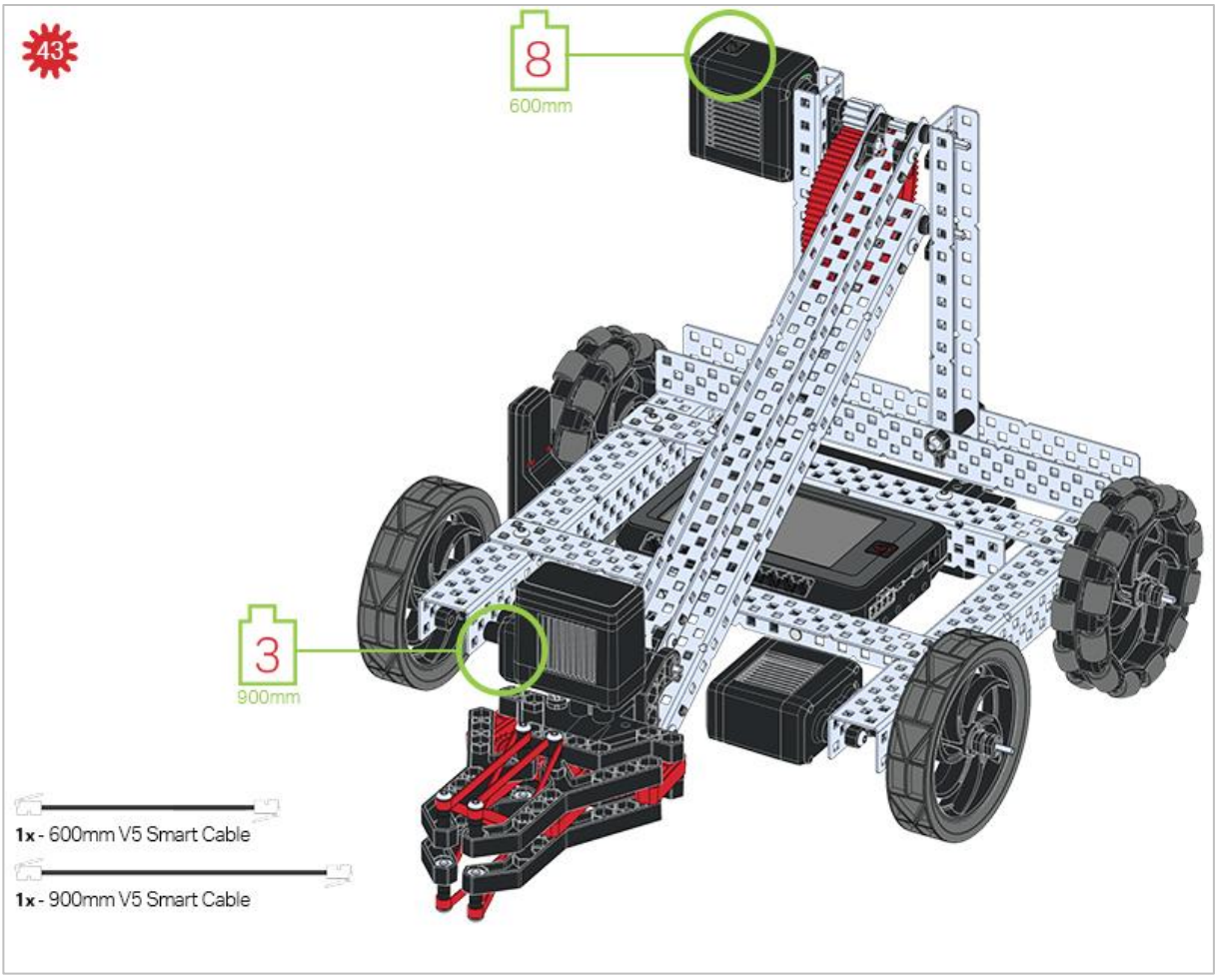

*Make sure that the port on the Smart Motor is facing the right side of the robot when the claw is installed (the same side as the V5 Radio).*

42



**4x** - Rubber Shaft Collar

**2x** - 2 in Shaft




## Build Instruction Tips

Check the Appendix for information on how to use the new Hex Nut Retainers.

# Exploration

Now that you've finished the build, test what it does. Explore your build and then answer these questions in your engineering notebook.

Where is the pivot point (the point the robot rotates about) for this robot build?

- Make a prediction and then manually move the 4” wheel on one side forward while at the same time moving the 4” wheel on the opposite side backwards at the same rate.

After manually moving both wheels, now describe where the pivot point is. How would the pivot point of the robot change if the Build design was changed so both of the 4” Omni Wheels (steps 14-17) were also powered by motors, making all 4 wheels powered?

- Make a prediction and then manually move both wheels on one side forward while at the same time moving both wheels on the opposite side backwards at the same rate.

After manually moving the wheels again, now describe where the pivot point is. Explain how a robot’s pivot point could change its behavior.

SPARK
PLAY

Test your build, observe how it functions,
and fuel your logic and reasoning skills
through imaginative, creative play.

# Robot Repetition

*Assembly line robots must perform actions repeatedly*

## Using Loops to Repeat Robotic Actions

Robots and computers are very good at consistency when performing actions multiple times. Computers use repetition to perform millions of calculations per second with incredible consistency. Since robots are built to interact with their environment and perform tasks precisely, they can be effectively used to repeat behaviors.

Behaviors to be repeated are grouped in programming structures called loops. The number of times and how fast they repeat depend on many factors that the programmer can specify.

Here are some examples of where repetition can be useful:

- Performing routine tasks multiple times
- Checking certain conditions multiple times to check for changes

# Programming Loops - VEXcode V5 Blocks

**The V5 Clawbot is ready to move!**

You can use the Help information inside of VEXcode V5 Blocks to learn about the blocks. For guidance in using the Help feature, see the Using Help tutorial.

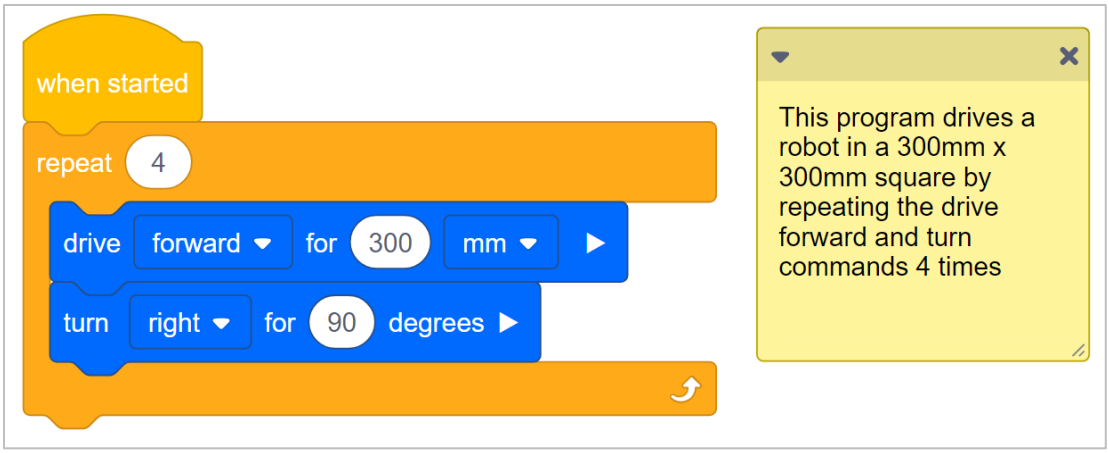

**Hardware/Software Required:**

| Quantity | Hardware/Other Items |
|---|---|
| 1 | VEX V5 Classroom Starter Kit (with up-to-date firmware) |
| 1 | VEXcode V5 Blocks (latest version, Windows, MacOS, Chromebook) |
| 1 | Engineering Notebook |
| 1 | Using Loops (Tutorial) |
| 1 | Repeating Actions (No Gyro) example project |

## 1. Let's start programming with loops.

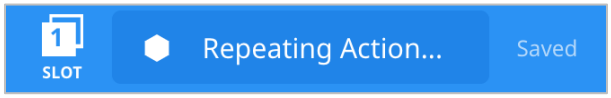
- Start by watching the **Using Loops** tutorial video.
- Open the **Repeating Actions (No Gyro)** example project.


- View the opened example project.




Do the following in your engineering notebook.

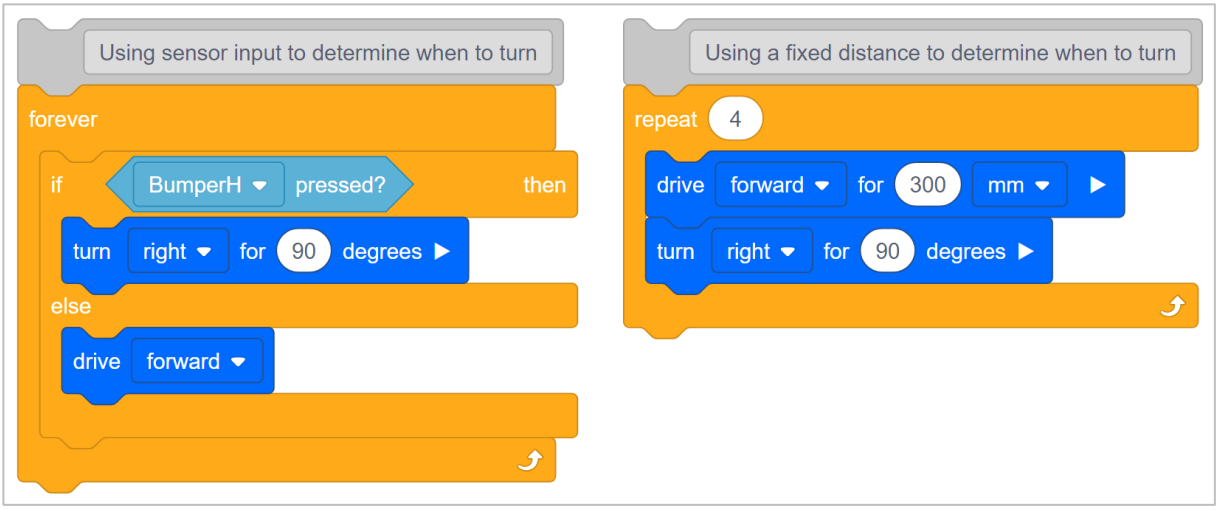
- Predict what the project will have the Clawbot do. Explain more than the fact that the project repeats.
  What is it repeating? What is the Clawbot doing?
- Write your prediction, but do not break the short project into more than two parts.
- Save, download, and run the Repeating Actions (No Gyro) example project.




- For help, see the tutorial in VEXcode V5 Blocks that explains how to Download and Run a Project.


- Check your explanations of the project in your engineering notebook and add notes to correct them as needed.

## 2. Run the project and observe the robot.



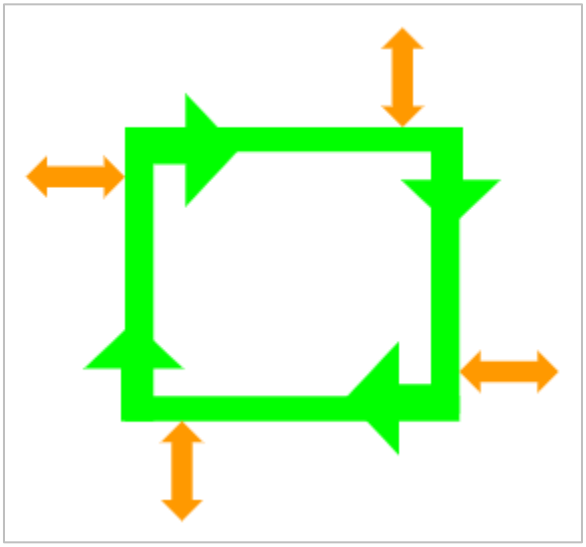

Look at the Repeating Actions (No Gyro) project (on the right) again. This project will repeat the forward and then turn behavior four times. A **repeat** block is used when you want to use a set of behaviors for a certain number of times. If the **repeat** block is replaced with a **forever** block, the robot would repeat the forward and then turn behaviors forever.

In the project on the left, a sensor's input is used to determine when to begin turning. The project on the right uses a fixed Drivetrain distance to determine when to begin turning.

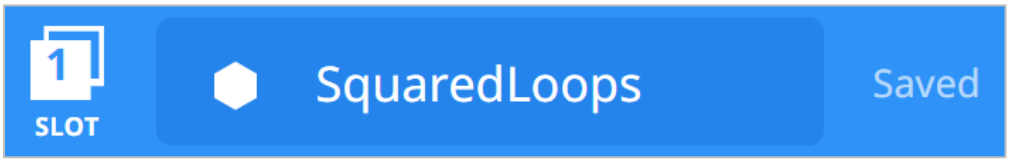
In order to continually check a sensor's input, an **if/else** block is used together with a **forever** block. In the project on left, the robot will turn right when the **BumperH** sensor is pressed, otherwise the robot will drive forward forever if the **BumperH** sensor is not pressed. To continually check the **BumperH** sensor's value, the **if/then** block is within a **forever** block.

The above project on the left is a practical use-case of a structure that repeats forever - using **forever** blocks and **if/then** blocks together. Imagine a self-driving sweeper that continues to drive forward until it runs into a wall or object, then turns before continuing to drive.

## 3. The Squared Loops Challenge!



- Have your Clawbot drive in a square.
- Before each turn, the claw must be opened and closed, and the arm must be raised and lowered.
- The Clawbot cannot drive along a side of the square more than once.
- You can use the Repeating Actions (No Gyro) example project as a starting point but save it as **SquaredLoops** before making any changes.




In your engineering notebook, plan the following:

- Plan out your solution and predict what each block in your project will have the Clawbot do.
- Download and run your project to test it before submitting it.
- Make changes to the project as needed and take notes about what was changed during testing.

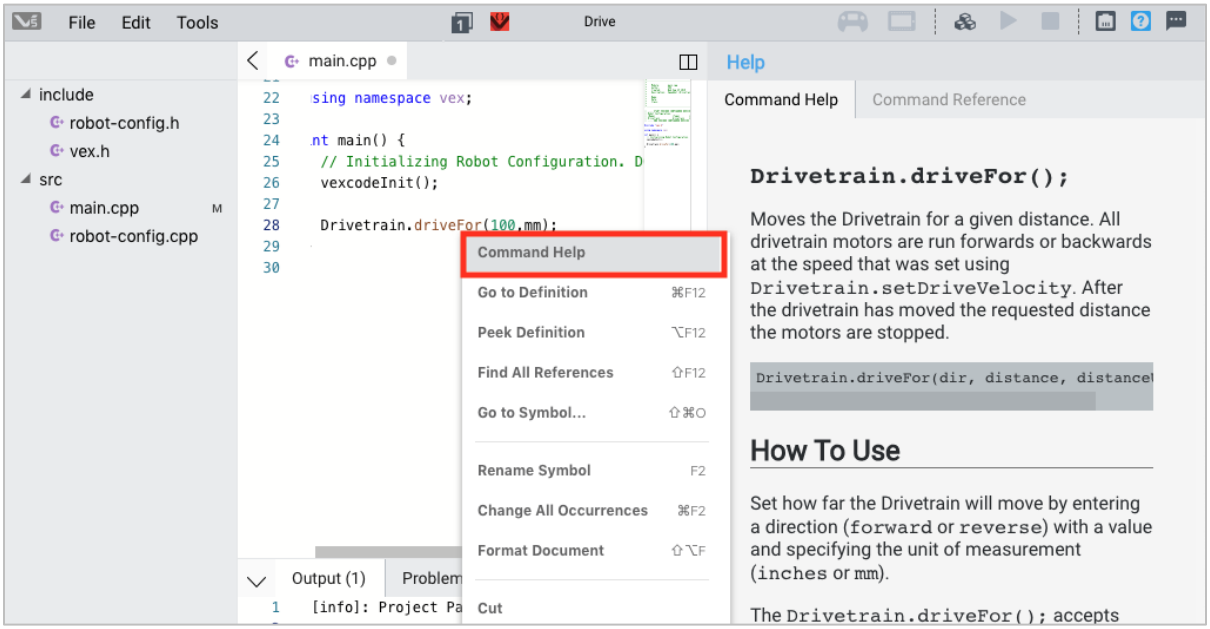
# Programming Loops - VEXcode V5 Text

**The Clawbot V5 is ready to move!**

This exploration will give you the tools to be able to start creating some cool projects that use loops.

- VEXcode V5 Text instructions that will be used in this exploration:
  - Drivetrain.driveFor(forward, 300, mm);
  - Drivetrain.turnFor(right, 90, degrees);
  - ClawMotor.spinFor(reverse, 70, degrees);
  - ArmMotor.spinFor(forward, 360, degrees);
  - while (true) {}
  - repeat (4) {}
  - wait(5, seconds);

To access additional information, right click on a command name in your workspace to see help for that command.



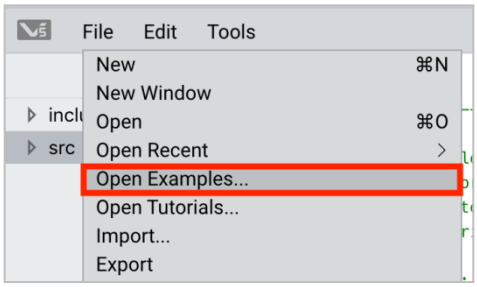

Make sure you have the hardware required, your engineering notebook, and VEXcode V5 Text downloaded and ready.

**Hardware/Software Required:**

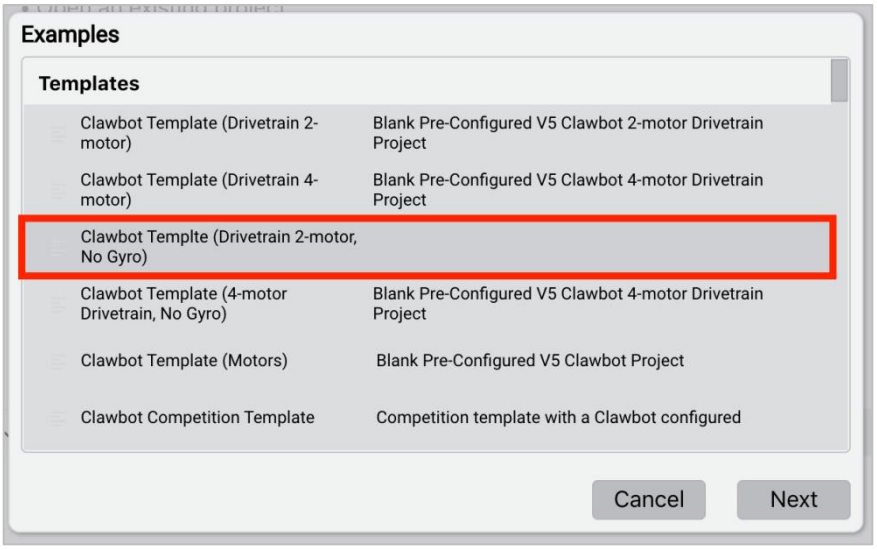
| Quantity | Hardware/Other Items |
|---|---|
| 1 | VEX V5 Classroom Starter Kit (with up-to-date firmware) |
| 1 | VEXcode V5 Text (latest version, Windows, MacOS) |
| 1 | Engineering Notebook |
| 1 | Clawbot Template (Drivetrain 2-motor, No Gyro) example project |

## 1. Let's start programming with loops.

- Before you begin your project, select the correct template project. The Clawbot Template (Drivetrain 2-motor, No Gyro) example project contains the Clawbot's motor configuration. If the template is not used, your robot will not run the project correctly.




- Select **File** and **Open Examples.**
- Scroll through the different **Example** projects. These projects demonstrate a variety of actions your Clawbot can perform. Select and open the **Clawbot Template (Drivetrain 2-motor, No Gyro)** example project.



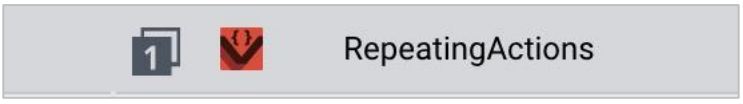

- Name the project RepeatingActions and select Create.


- Type the following code:

```
using namespace vex;

int main() {
  // Initializing Robot Configuration. DO NOT REMOVE!
  vexcodeInit();

  // drives forward and turns 90 degrees for 4 iterations
  repeat(4){
    Drivetrain.driveFor(forward, 300,mm);
    Drivetrain.turnFor(right, 90, degrees);
    wait(5, seconds);
  }
}
```

Look over the project and then do the following in your engineering notebook.

- Predict what the project will have the Clawbot do. Explain more than the fact that the project repeats.
  What is it repeating? What is the Clawbot doing?
- Write your prediction, but do not break the short project into more than two parts.
- Save, download, and run the Repeating Actions project.




- Check your explanations of the project in your engineering notebook and add notes to correct them as needed.

## 2. Run the project and observe the robot.

**Using sensor input to determine when to turn**

```
#include "vex.h"

using namespace vex;

int main() {
  // Initializing Robot Configuration. DO NOT REMOVE!
  vexcodeInit();
// The robot turns right 90 degrees if the Bumper Switch is pressed
// Otherwise, the robot drives forward
while(true){
  if(BumperB.pressing()){
    Drivetrain.turnFor(right, 90, degrees);
  }
  else{
    Drivetrain.drive(forward);
    }
  }
}
```

**Using a fixed distance to determine when to turn**

```
#include "vex.h"

using namespace vex;

int main() {
  // Initializing Robot Configuration. DO NOT REMOVE!
  vexcodeInit();

  //drives forward and turns 90 degrees for 4 iterations
  repeat(4){
    Drivetrain.driveFor(forward, 300, mm);
    Drivetrain.turnFor(right, 90, degrees);
    wait(5, seconds);
  }
}
```

Look at the Repeating Actions project (on the right) again. This project will repeat the forward and then turn behavior four times. A "`repeat`" loop structure is used when you want to use a set of behaviors a certain number of times.

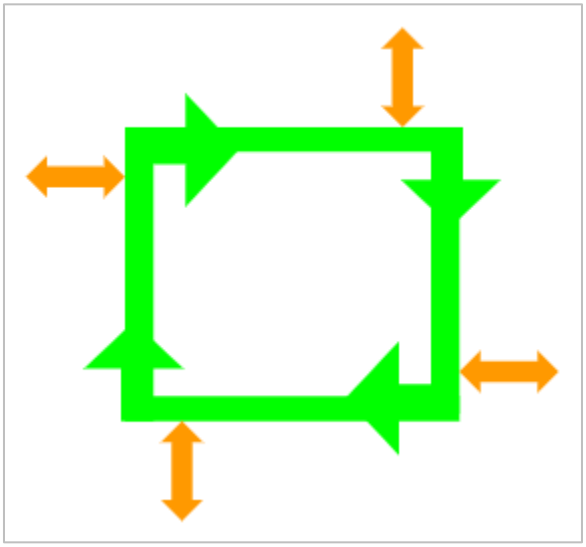
If the repeat structure is replaced with a "`while`" loop structure, the robot will repeat the forward and then turn behaviors "`while`" the condition is true. You can also set the condition to "`true`" to have the "`while`" loop continue forever.

In the project on the left, a sensor's input is used to determine when to begin turning. The project on the right uses a fixed Drivetrain distance to determine when to begin turning.

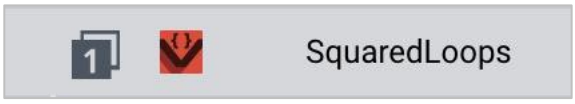
In order to continually check a sensor's input, an "`if/else`" statement is used together a "`while`" loop. In the project on left, the robot will turn right when the "`BumperB`" sensor is pressed, otherwise the robot will drive forward forever if the "`BumperB`" sensor is not pressed. To continually check the BumperB sensor's value, the "`if`" statement is within a "`while true`" loop.

The above project on the left is a practical use-case of a structure that repeats forever - using `while` loops and `if` statements together. Imagine a self-driving sweeper that continues to drive forward until it runs into a wall or object, then turns before continuing to drive.

## 3. The Squared Loops Challenge!



- Have your Clawbot drive in a square.
- Before each turn, the claw must be opened and closed, and the arm must be raised and lowered.
- The Clawbot cannot drive along a side of the square more than once.
- You can use the RepeatingActions project from above as a starting point but save it as **SquaredLoops** before making any changes.




In your engineering notebook, plan the following:

- Plan out your solution and predict what each instruction in your project will have the Clawbot do.
- Download and run your project to test it before submitting it.
- Make changes to the project as needed and take notes about what was changed during testing.


Become a 21st century problem solver
by applying the core skills and concepts
you learned to other problems.

# Robots in Manufacturing

*Robots working in a manufacturing plant.*

## Factory Robots

Factories first began using modern industrial robots in the early 1960's. These robots could do the dirty, dull, and dangerous jobs that were previously completed by humans. Since then, factories all over the world have invested millions of dollars to develop and build robots that can manufacture their products quickly and efficiently.

Factory robots are always being improved upon as new technology is expanded. New metals and materials allow robots to be used in high pressure or high temperature environments. Usually separated to keep human workers safe in case of an accident, factory robots are being made from new "softer" materials. These materials, such as rubber and plastic, can help reduce injuries in a robot/human collision. With the introduction of artificial intelligence and sensors, factory robots can be "taught" new ways of delivering these products overnight and adapt their movements in real time. This allows more productivity and precision.

Factory robots are used in the production of many products, but the top three robotic jobs in manufacturing are:

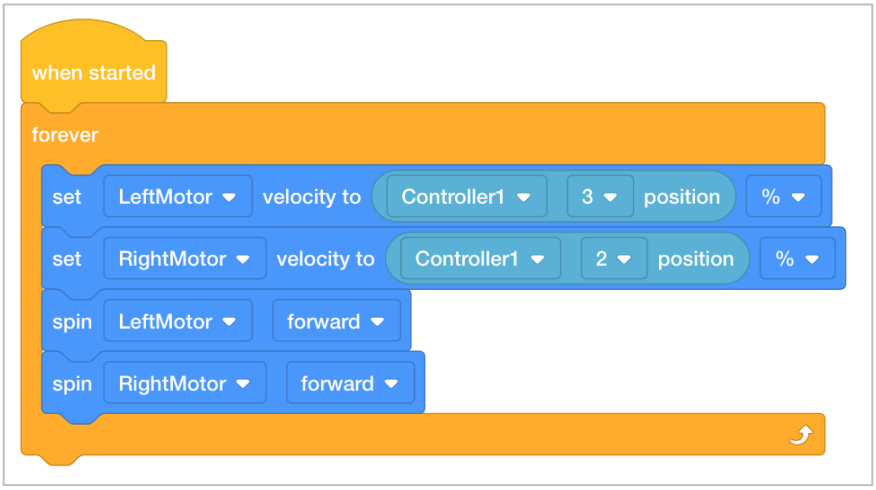
- Drilling
- Welding
- Painting and Sealing

# Controllers and Loops - VEXcode V5 Blocks

## Controllers and Loops

In competitions, teams must manipulate their robots wirelessly with controllers. The controller is programmed to update the robot based on input from the user. Loops are used in the project so that the robot repeatedly checks for updated input information. Loops allow the project to rapidly check which buttons have been pressed, or how far joysticks have been pushed. Once checked, this information is quickly relayed to the robot so that it responds quickly to the controller's instructions.

The following image shows the Tank Drive example project from VEXcode V5 Blocks. The *forever* loop in this project checks the positions of Axes 2 and 3 forever in order to set the velocity of the motors.




*Tank Drive example project from VEXcode V5 Blocks*

Loops are important even for autonomous programming without a controller. A loop helps to simplify and organize repeated commands within a project.

# Controllers and Loops - VEXcode V5 Text

## Controllers and Loops

In competitions, teams must manipulate their robots wirelessly with controllers. The controller is programmed to update the robot based on input from the user. Loops are used in the project so that the robot repeatedly checks for updated input information. Loops allow the project to rapidly check which buttons have been pressed, or how far joysticks have been pushed. Once checked, this information is quickly relayed to the robot so that it responds quickly to the controller's instructions.

The following image shows the Tank Drive example project from VEXcode V5 Text. The *forever* loop in this project checks the positions of Axes 2 and 3 forever in order to set the velocity of the motors.

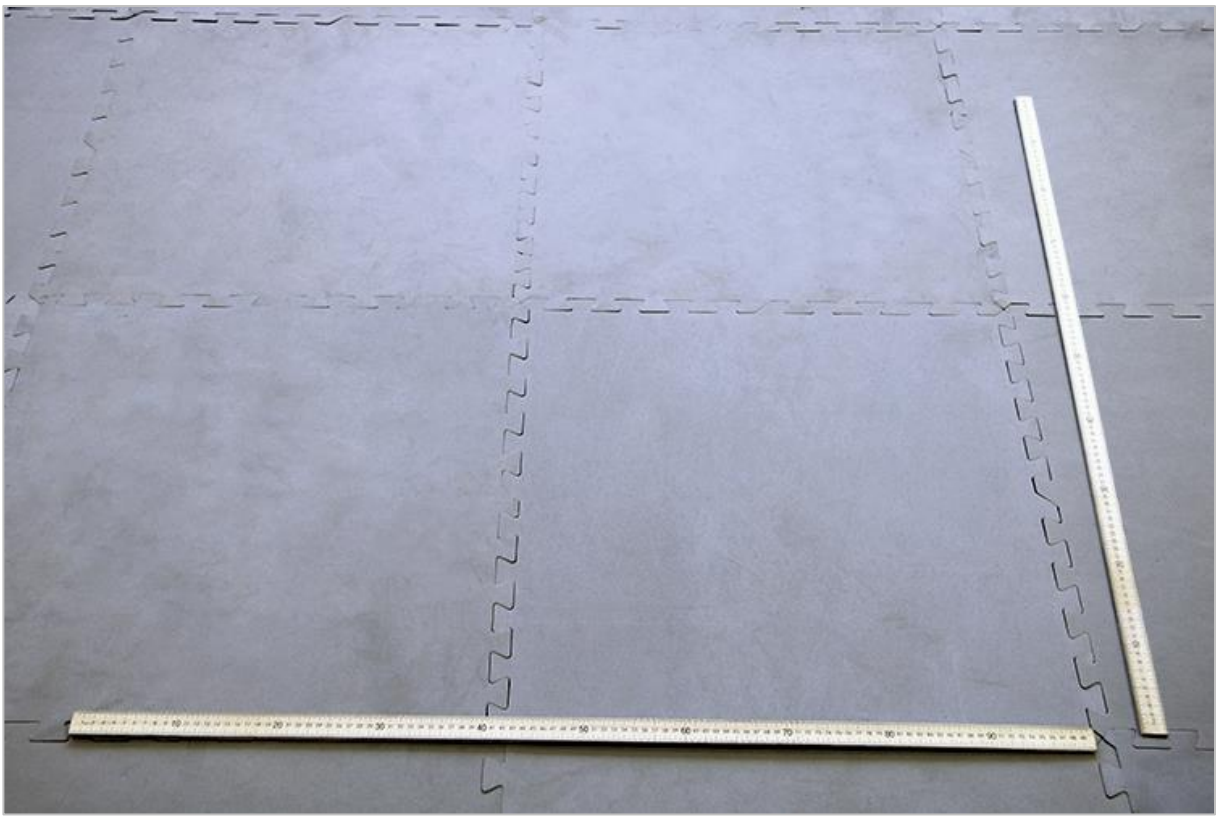
```
int main() {
  // Initializing Robot Configuration. DO NOT REMOVE!
  vexcodeInit();

  // Deadband stops the motors when Axis values are close to zero.
  int deadband = 5;
  while (true) {
    // Get the velocity percentage of the left motor. (Axis3)
    int leftMotorSpeed = Controller1.Axis3.position();
    // Get the velocity percentage of the right motor. (Axis2)
    int rightMotorSpeed = Controller1.Axis2.position();

    // Set the speed of the left motor. If the value is less than the deadband,
    // set it to zero.
    if (abs(leftMotorSpeed) < deadband) {
      // Set the speed to zero.
      LeftMotor.setVelocity(0, percent);
    } else {
      // Set the speed to leftMotorSpeed
      LeftMotor.setVelocity(leftMotorSpeed, percent);
    }
    // Set the speed of the right motor. If the value is less than the deadband,
    // set it to zero.
    if (abs(rightMotorSpeed) < deadband) {
      // Set the speed to zero
      RightMotor.setVelocity(0, percent);
    } else {
      // Set the speed to rightMotorSpeed
      RightMotor.setVelocity(rightMotorSpeed, percent);
    }
    // Spin both motors in the forward direction.
    LeftMotor.spin(forward);
    RightMotor.spin(forward);
    wait(25, msec);
  }
}
```

*Tank Drive example project from VEXcode V5 Text*

Loops are important even for autonomous programming without a controller. A loop helps to simplify and organize repeated commands within a project.


Is there a more efficient way to come to the same conclusion? Take what you've learned and try to improve it.

# Prepare for the Groove Machine Challenge



*Dance floor set up*

## Prepare for the Groove Machine Challenge!

In the Groove Machine Challenge, you will break up into groups and program your robot to go through a dance routine for a dance-off using your knowledge of loops.

For the challenge, you will need to clear a space on the floor large enough for a V5 Clawbot to move around for a dance routine without bumping into anything. A 1x1 meter area is recommended to give each Clawbot adequate space for moving.

# Design, Develop, and Iterate on your Project - VEXcode V5 Blocks

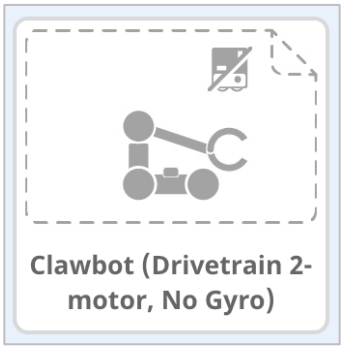
Answer the following questions in your engineering notebook as you design your project.

- What type of robot dance will you create? Explain with details.
- What types of loops will you use and why?
- What steps will you follow to test the dance? Explain with details.

In order to help you plan, click here for a few example dance moves that you might include in the Clawbot's dance.

Follow the steps below as you create your project:

- Plan out the dance using drawings and pseudocode.
- Use the pseudocode you created to develop your project using VEXcode V5 Blocks.
- Open the Clawbot (Drivetrain 2-motor, No Gyro) example project.

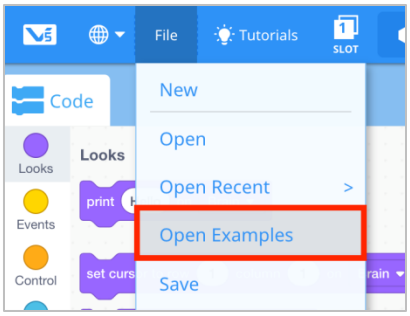



- Save your project as GrooveMachine.


- Run your project to test it often and iterate on it using what you learned from your testing.
- Share your final project with your teacher.

If you're having trouble getting started, review the following in VEXcode V5 Blocks:

- Example projects




- Using Loops tutorial


- Previous versions of your project
- The Help feature to learn more about the blocks

# The Groove Machine Challenge - VEXcode V5 Blocks

*V5 Clawbot with its arm up and claw open*

## The Groove Machine Challenge

In this challenge, you will break up into teams and program your robot to go through a dance routine using your knowledge of loops. Your teacher will set a time limit for developing/testing the dance and a time limit for the length of the dance. Everyone not on the competing head-to-head dance-off teams will judge the dances and vote on the team that they think is best.

**Rules:**

- Each Clawbot will dance one-at-a-time within the 1x1 meter area.
- The dancing continues until the Stop button on the Brain's screen is pressed to stop the project from running.
- The arm must be raised and lowered.
- The claw must open and close.
- The Clawbot must turn left and right.
- The Clawbot must drive forward and in reverse.
- The project needs to be stopped immediately if the Clawbot collides with anything or falls over. That is a losing dance.

# Design, Develop, and Iterate on your Project - VEXcode V5 Text

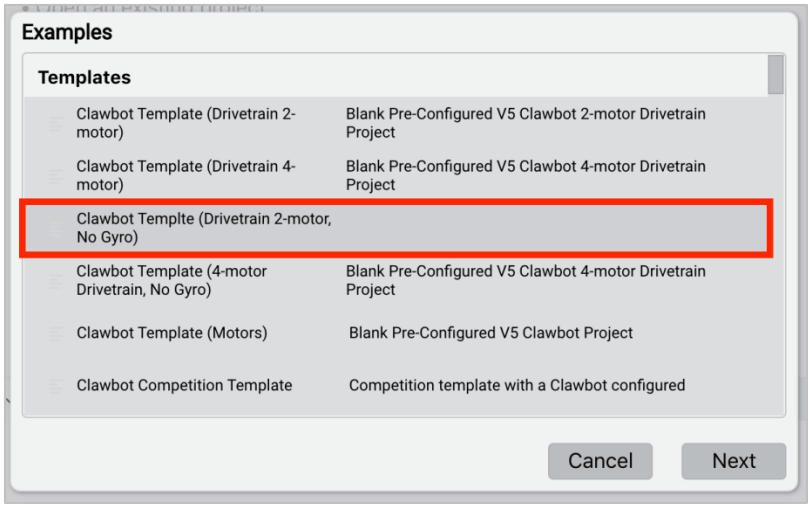
Answer the following questions in your engineering notebook as you design your project.

- What type of robot dance will you create? Explain with details.
- What types of loops will you use and why?
- What steps will you follow to test the dance? Explain with details.

In order to help you plan, click here for a few example dance moves that you might include in the Clawbot's dance.

Follow the steps below as you create your project:

- Plan out the dance using drawings and pseudocode.
- Use the pseudocode you created to develop your project using VEXcode V5 Text.
- Open the Clawbot Template (Drivetrain 2-motor, No Gyro) example project.

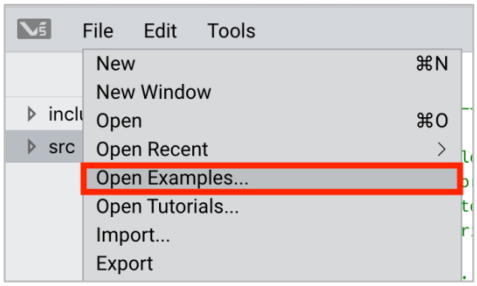



- Name the project GrooveMachine and select Create.

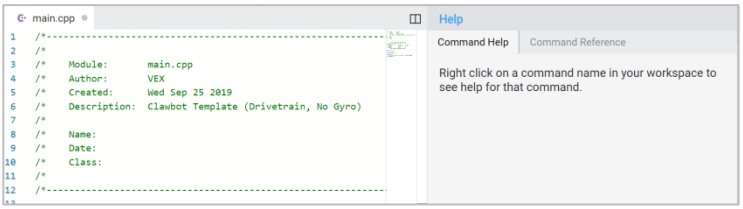

- Run your project to test it often and iterate on it using what you learned from your testing.
- Share your final project with your teacher.

If you're having trouble getting started, review the following in VEXcode V5 Text:

- Example projects:

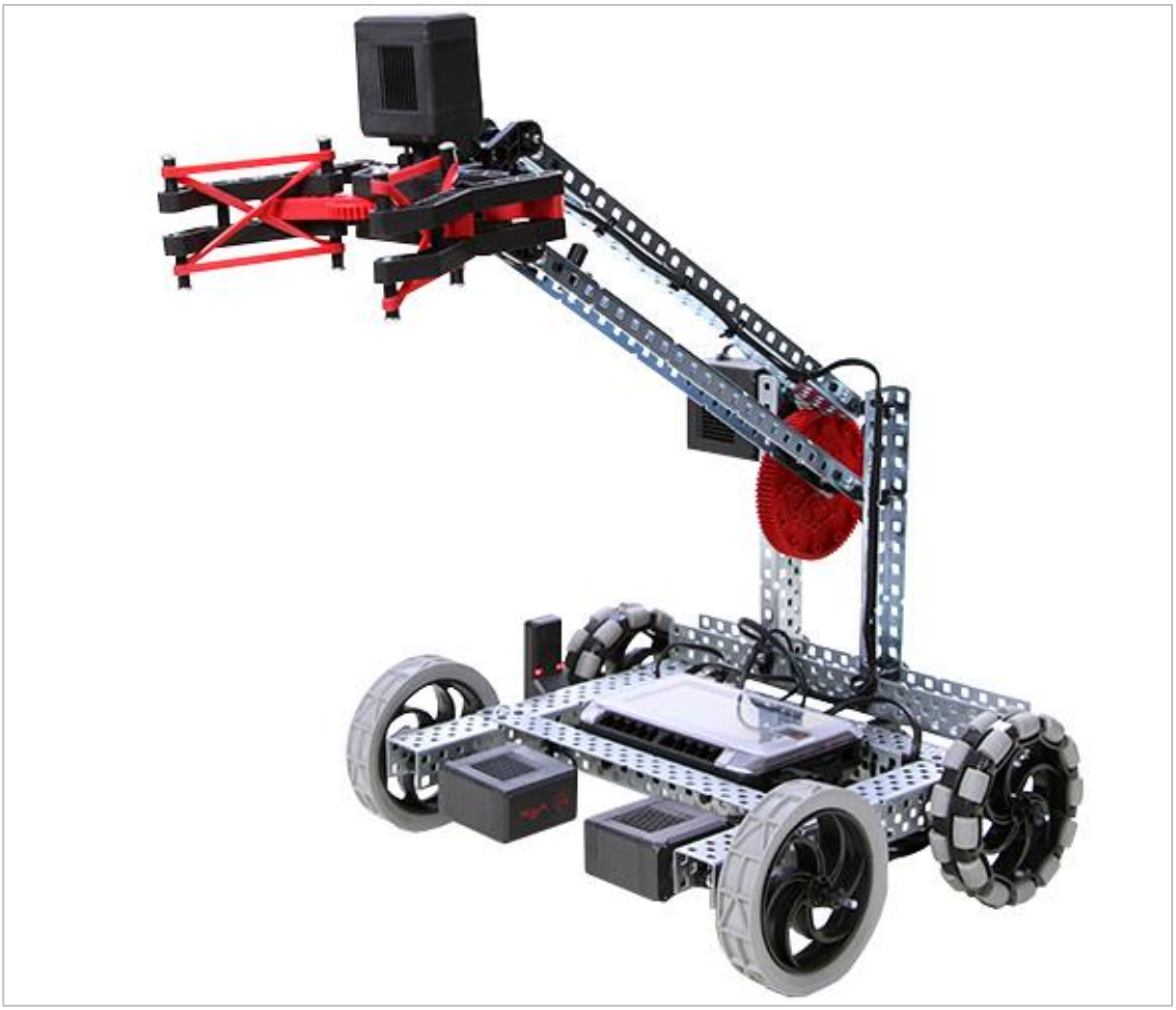



- To access additional information while creating a program, right click on a instruction in your work space to see additional information about it.




- Review previous versions of your RepeatingActions project to assist with creating your new one.

# The Groove Machine Challenge - VEXcode V5 Text



*V5 Clawbot with its arm up and claw open*

## The Groove Machine Challenge

In this challenge, you will break up into teams and program your robot to go through a dance routine using your knowledge of loops. Your teacher will set a time limit for developing/testing the dance and a time limit for the length of the dance. Everyone not on the competing head-to-head dance-off teams will judge the dances and vote on the team that they think is best.

**Rules:**

- Each Clawbot will dance one-at-a-time within the 1x1 meter area.
- The dancing continues until the Stop button on the Brain's screen is pressed to stop the project from running.
- The arm must be raised and lowered.
- The claw must open and close.
- The Clawbot must turn left and right.
- The Clawbot must drive forward and in reverse.
- The project needs to be stopped immediately if the Clawbot collides with anything or falls over. That is a losing dance.


Understand the core concepts and how to apply them to different situations. This review process will fuel motivation to learn.

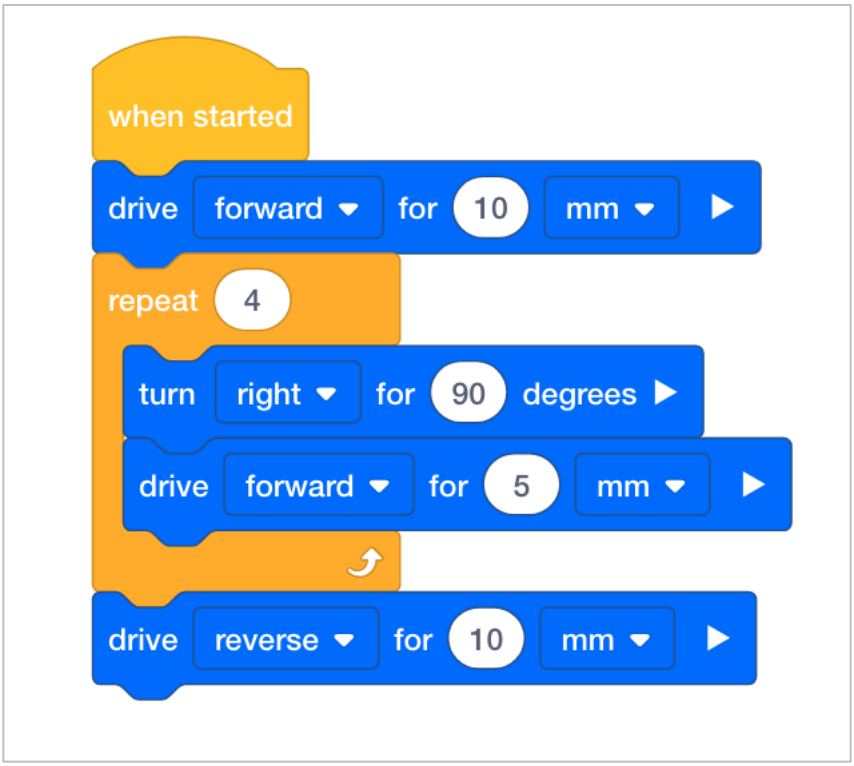
# Review - VEXcode V5 Blocks

1. **True or False: Controllers use loops in order to check for input responses from the buttons and joysticks.**
   - True
   - False

2. **Loops are _____.**
   - documents the data types available
   - conditionally executes a group of statements
   - statement(s) to handle errors that occur
   - programming structures that repeat behaviors

3. **True or False: In VEXcode V5 Blocks, the `repeat` and `forever` loop blocks repeat any blocks inside of it, and for the number of times specified. Blocks inside of the `repeat` and `forever` loop blocks are run in order from top to bottom.**
   - True
   - False

4. **In the following project, how many times does the robot turn right?**




- The robot will turn right 9 times because it is set to turn 90 degrees.
- The robot will turn right one time because there is only one `turn for` block in the project.
- The robot will turn right 4 times because the `repeat` block is set to repeat 4 times.
- The robot will turn right 5 times because the `drive for` block moves 5mm.

5. **True or False: Robots are not good at repetitive tasks like humans are because any slight error in their programming increases with each repetition.**
   - True
   - False

# Review - VEXcode V5 Text

6. **True or False: Controllers use loops in order to check for input responses from the buttons and joysticks.**
   - True
   - False

7. **Loops are _____.**
   - documents the data types available
   - conditionally executes a group of statements
   - statement(s) to handle errors that occur
   - programming structures that repeat behaviors

8. **True or False: In VEXcode V5 Text, the `repeat` and `forever` loop structures repeat any instruction inside of it, and for the number of times specified. Instructions inside of the `repeat` and `forever` loop structures are run in order from top to bottom.**
   - True
   - False

9. **In the following project, how many times does the robot turn right?**

```
#include "vex.h"

using namespace vex;

int main() {
  // Initializing Robot Configuration. DO NOT REMOVE!
  vexcodeInit();

  Drivetrain.driveFor(forward, 100, mm);
  repeat(4){
    Drivetrain.turnFor(right, 90, degrees);
    Drivetrain.driveFor(forward, 50, mm);
    wait(5, msec);
  }
  Drivetrain.driveFor(reverse, 100, mm);
}
```

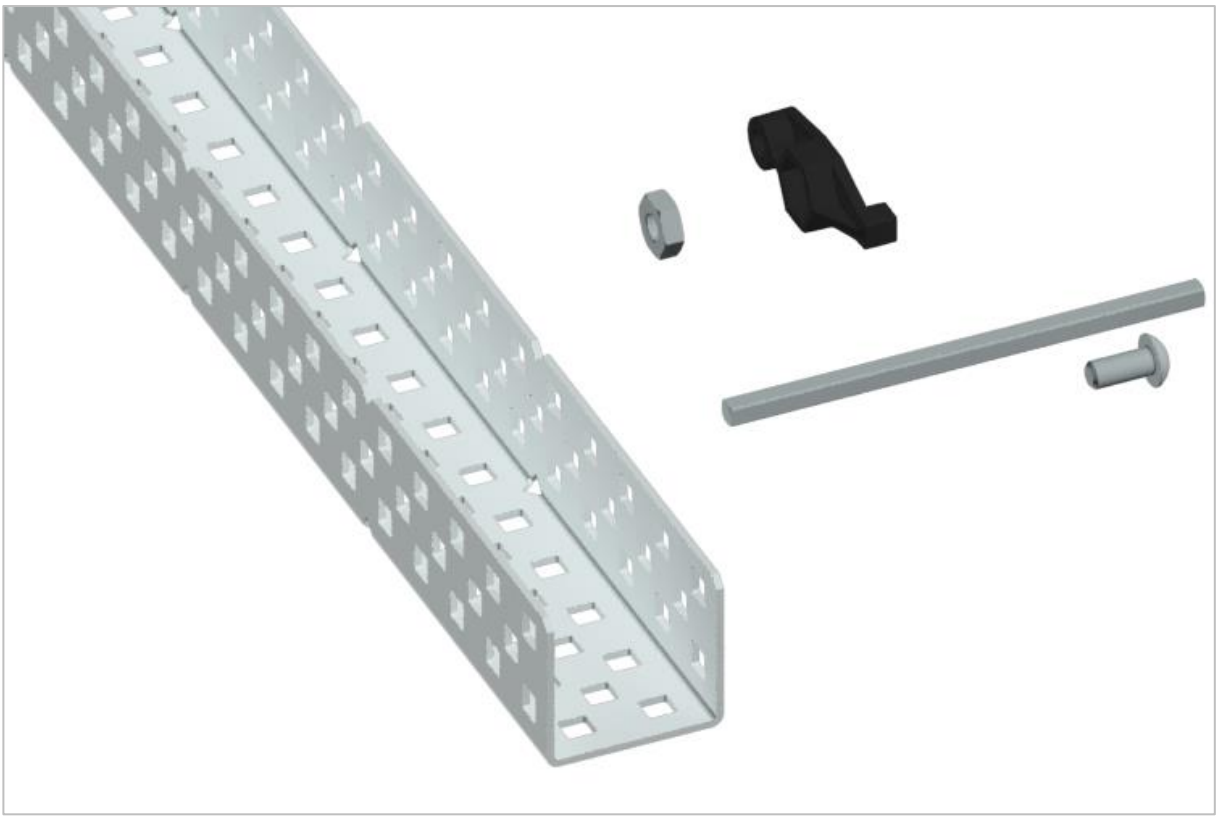
- The robot will turn right 9 times because it is set to turn 90 degrees.
- The robot will turn right one time because there is only one `turn for` instruction in the project.
- The robot will turn right 4 times because the `repeat` structure is set to repeat 4 times.
- The robot will turn right 5 times because the `drive for` instruction moves 5mm.

10. **True or False: Robots are not good at repetitive tasks like humans are because any slight error in their programming increases with each repetition.**
    - True
    - False

# APPENDIX

Additional information, resources, and materials.

# Using the 1 Post Hex Nut Retainer w/ Bearing Flat



*1 Post Hex Nut Retainer w/ Bearing Flat*

## Using the 1 Post Hex Nut Retainer w/ Bearing Flat

The 1 Post Hex Nut Retainer w/ Bearing Flat allows shafts to spin smoothly through holes in structural components. When mounted, it provides two points of contact on structural components for stability. One end of the retainer contains a post sized to securely fit in the square hole of a structural component. The center hole of the retainer is sized and slotted to securely fit a hex nut, allowing a 8-32 screw to easily be tightened without the need for a wrench or pliers. The hole on the end of the Retainer is intended for shafts or screws to pass through.

To make use of the retainer:

- Align it on a VEX structural component such that the end hole is in the desired location, and the center and end sections are also backed by the structural component.

- Insert the square post extruding from the retainer into the structural component to help keep it in place.
- Insert a hex nut into the center section of the retainer so that it is flush with the rest of the component.
- Align any additional structural components to the back of the main structural component, if applicable.
- Use an 8-32 screw of appropriate length to secure the structural component(s) to the retainer through the center hole and hex nut.

# Using the 4 Post Hex Nut Retainer

*4 Post Hex Nut Retainer*

## Using the 4 Post Hex Nut Retainer

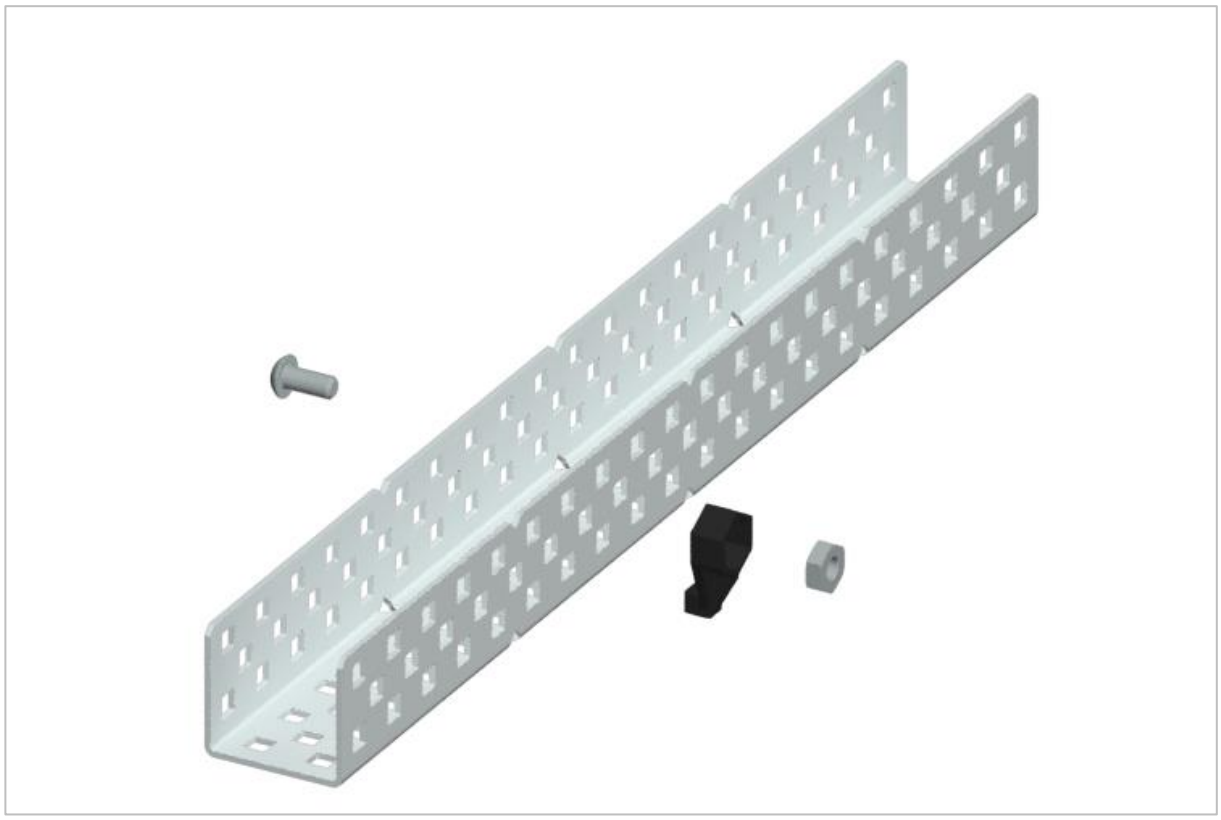
The 4 Post Hex Nut Retainer provides five points of contact for creating a strong connection between two structural components using one screw and nut. Each corner of the retainer contains a post sized to securely fit in a square hole within a structural component. The center of the retainer is sized and slotted to securely fit a hex nut, allowing a 8-32 screw to easily be tightened without the need for a wrench or pliers.

To make use of the retainer:

- Align it on a VEX structural component such that the center hole is in the desired location, and each corner is also backed by the structural component.
- Insert the square posts extruding from the retainer into the structural component to help keep it in place.
- Insert a hex nut into the center section of the retainer so that it is flush with the rest of the component.

- Align any additional structural components to the back of the main structural component, if applicable.
- Use an 8-32 screw of appropriate length to secure the structural component(s) to the retainer through the center hole and hex nut.

# Using the 1 Post Hex Nut Retainer



*1 Post Hex Nut Retainer*

## Using the 1 Post Hex Nut Retainer

The 1 Post Hex Nut Retainer provides two points of contact for connecting a structural component to another piece using one screw and nut. One end of the retainer contains a post sized to securely fit in the square hole of a structural component. The other end of the retainer is sized and slotted to securely fit a hex nut, allowing a 8-32 screw to easily be tightened without the need for a wrench or pliers.

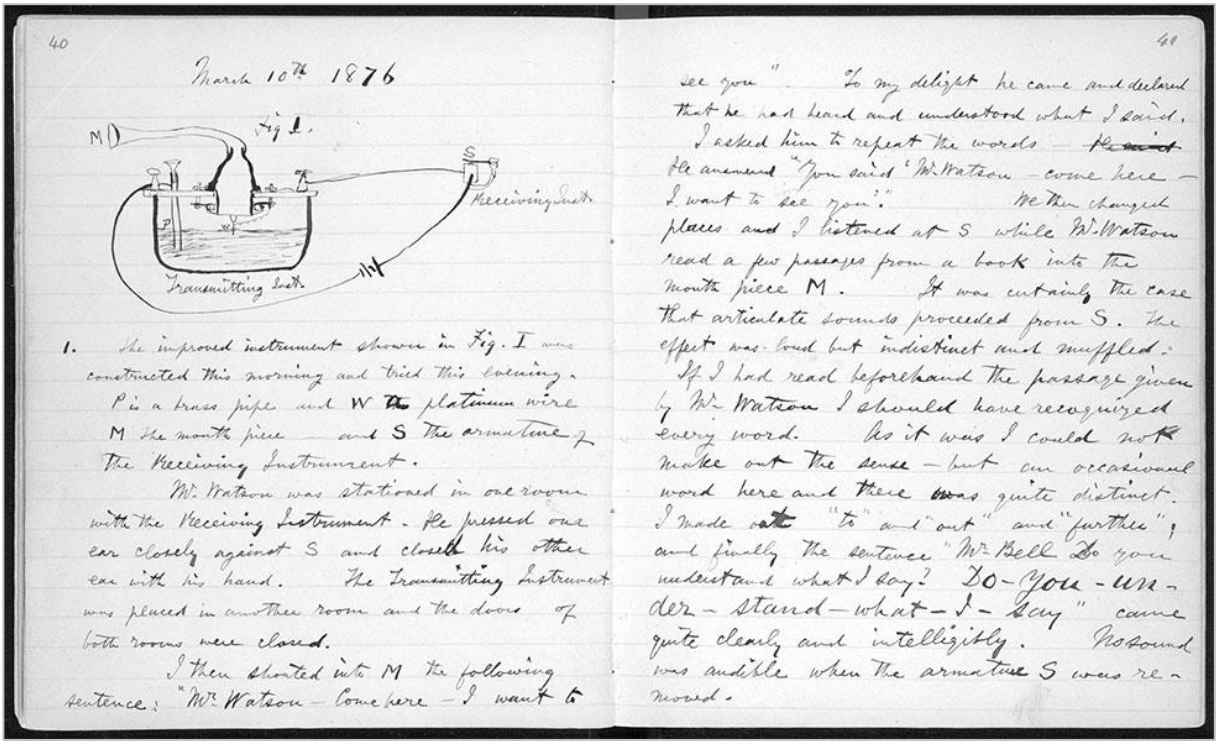
To make use of the retainer:

- Align it on a VEX structural component such that both ends are backed by the structural component and positioned to secure the second piece.
- Insert the square post extruding from the retainer into the structural component to help keep it in place.
- If the retainer is being used to secure two structural components, insert a hex nut into the other end of the retainer so that it is flush with the rest of the component. If used to secure

a different type of component, such as a standoff, it may be appropriate to insert the screw through this side.

- Align any additional components to the back of the main structural component, if applicable.
- If the retainer is being used to connect two structural components, use an 8-32 screw of appropriate length to secure the structural components through the hole and hex nut. If used to connect a different type of component, such as a standoff, secure it directly or with a hex nut.

# Engineering Notebooks



40

March 10th 1876

Fig 1.

M

S

Receiving Instr.

Transmitting Instr.

1. The improved instrument shown in Fig. I was constructed this morning and tried this evening. P is a brass pipe and W the platinum wire M the mouth piece — and S the armature of the Receiving Instrument.

Mr. Watson was stationed in one room with the Receiving Instrument - He pressed one ear closely against S and closed his other ear with his hand. The Transmitting Instrument was placed in another room and the doors of both rooms were closed.

I then shouted into M the following sentence: "Mr. Watson — Come here — I want to

41

see you" To my delight he came and declared that he had heard and understood what I said.

I asked him to repeat the words He answered "You said 'Mr. Watson — come here — I want to see you.'"

We then changed places and I listened at S while Mr. Watson read a few passages from a book into the mouth piece M. It was certainly the case that articulate sounds proceeded from S. The effect was loud but indistinct and muffled.

If I had read beforehand the passage given by Mr. Watson I should have recognized every word. As it was I could not make out the sense — but an occasional word here and there was quite distinct. I made out "to" and "out" and "further", and finally the sentence "Mr. Bell Do you understand what I say? DO-YOU-UN-DER-STAND-WHAT-I-SAY" came quite clearly and intelligibly. No sound was audible when the armature S was removed.

*Alexander Graham Bell's notebook entry from a successful experiment with his first telephone*

## An Engineering Notebook Documents your Work

Not only do you use an engineering notebook to organize and document your work, it is also a place to reflect on activities and projects. When working in a team, each team member will maintain their own journal to help with collaboration.

Your engineering notebook should have the following:

- An entry for each day or session that you worked on the solution
- Entries that are chronological, with each entry dated
- Clear, neat, and concise writing and organization
- Labels so that a reader understands all of your notes and how they fit into your iterative design process

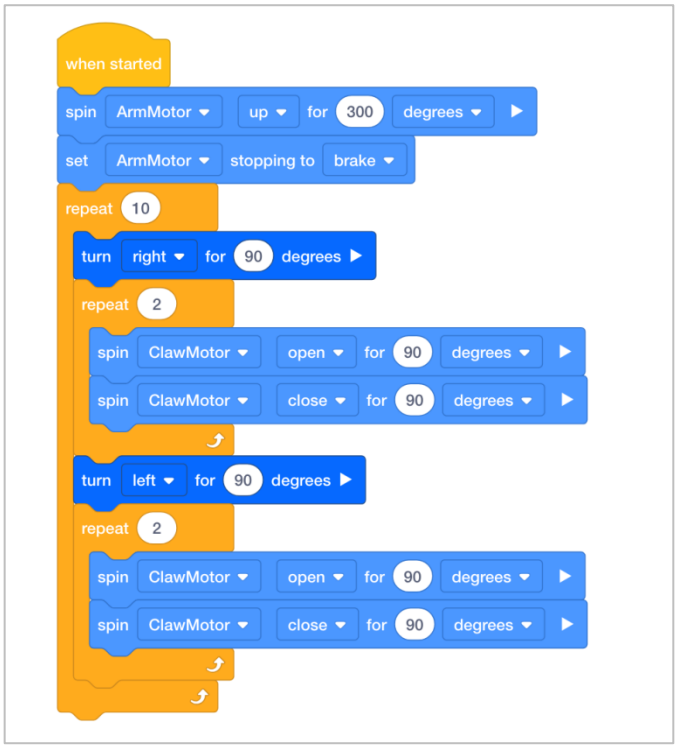
An entry might include:

- Brainstorming ideas
- Sketches or pictures of prototypes

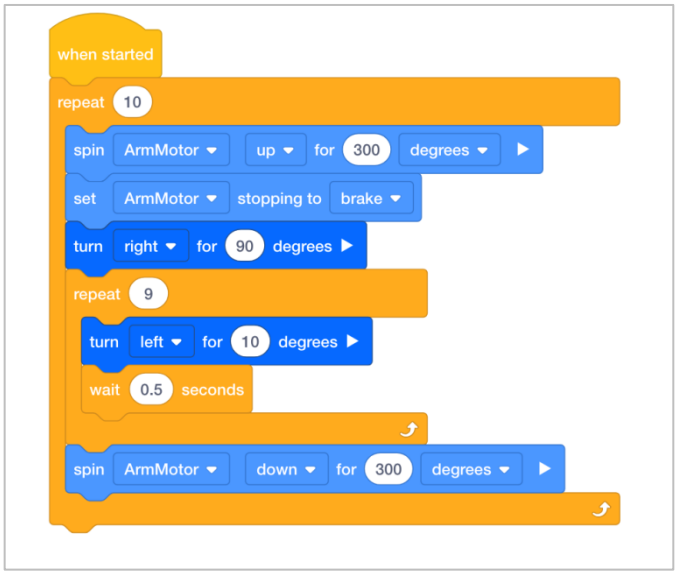
- Pseudocode and flowcharts for planning
- Any worked calculations or algorithms used
- Answers to guiding questions
- Notes about observations and/or conducted tests
- Notes about and reflections on your different iterations

# Example Dance Moves - VEXcode V5 Blocks

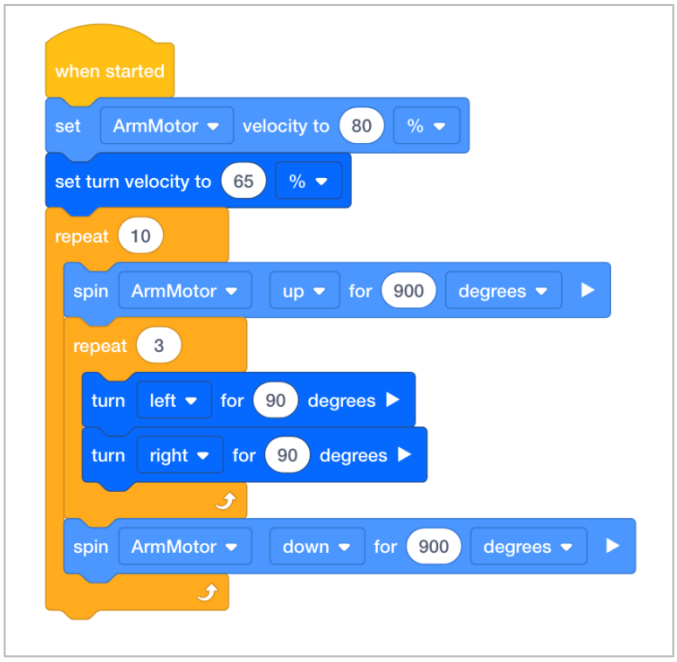
### Clap Your Hands!




### The Sprinkler




### Hands in the air!


when started
set ArmMotor velocity to 80 %
set turn velocity to 65 %
repeat 10
spin ArmMotor up for 900 degrees
repeat 3
turn left for 90 degrees
turn right for 90 degrees
spin ArmMotor down for 900 degrees

# Example Dance Moves - VEXcode V5 Text

**Clap Your Hands!**

```
#include "vex.h"

using namespace vex;

int main() {
  // Initializing Robot Configuration. DO NOT REMOVE!
  vexcodeInit();
  ArmMotor.spinFor(forward, 300, degrees);
  ArmMotor.setStopping(brake);
  repeat(10) {
    Drivetrain.turnFor(right, 90, degrees);
    repeat(2) {
      ClawMotor.spinFor(forward, 90, degrees);
      ClawMotor.spinFor(reverse, 90, degrees);
      wait(5, msec);
    }
    Drivetrain.turnFor(left, 90, degrees);
    repeat(2) {
      ClawMotor.spinFor(forward, 90, degrees);
      ClawMotor.spinFor(reverse, 90, degrees);
      wait(5, msec);
    }
  }
}
```

**The Sprinkler**

```
#include "vex.h"

using namespace vex;

int main() {
  // Initializing Robot Configuration. DO NOT REMOVE!
  vexcodeInit();
  repeat(10) {
    ArmMotor.spinFor(forward, 300, degrees);
    ArmMotor.setStopping(brake);
    Drivetrain.turnFor(right, 90, degrees);
    repeat(9) {
      Drivetrain.turnFor(left, 10, degrees);
      wait(0.5, seconds);
    }
    ArmMotor.spinFor(reverse, 300, degrees);
    wait(5, msec);
  }
}

```

**Hands in the air!**

```
#include "vex.h"

using namespace vex;

int main() {
  // Initializing Robot Configuration. DO NOT REMOVE!
  vexcodeInit();
    ArmMotor.setVelocity(80, percent);
    Drivetrain.setTurnVelocity(50, percent);

    repeat(10) {
      ArmMotor.spinFor(forward, 500, degrees);
      repeat(3) {
        Drivetrain.turnFor(left, 90, degrees);
        Drivetrain.turnFor(right, 90, degrees);
        wait(5, msec);
    }
    ArmMotor.spinFor(reverse, 500, degrees);
    wait(5, msec);
    }
```