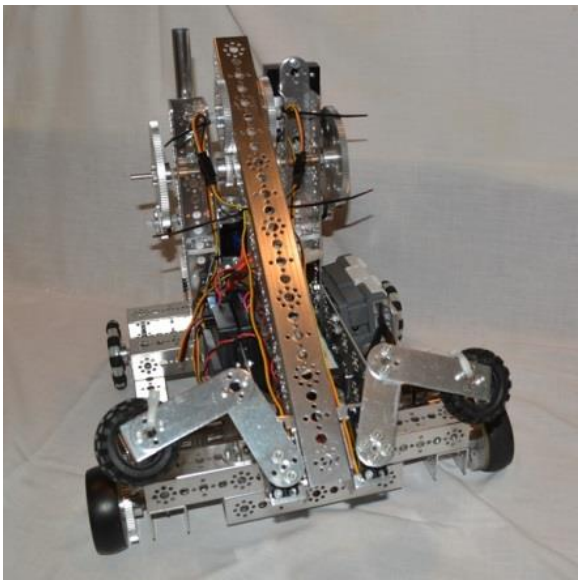
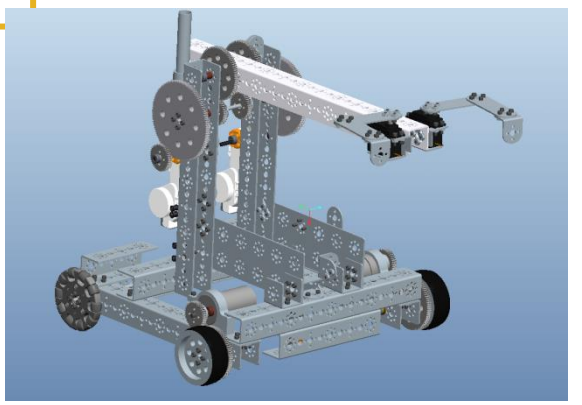
FIRST Tech Challenge®
Push Bot: A Manual for TETRIX





Developed By FTC Team 0003 Australia
Revised for TETRIX by former FTC Team 2843
Hollywood, Maryland
United States of America
September 2014

Revision History		
Revision	**Date**	**Description**
1	8/15/2014	Initial Release – by FTC Team 003 Australia, The Southport School
2	9/1/2014	Replaced MATRIX content with TETRIX content by former FTC Team 2843, Under the Son

Contents

Introduction

What is the FIRST Tech Challenge?

FIRST® Tech Challenge is a student-centered activity that focuses on giving students a unique and stimulating experience. Each year, Teams participate in a new Game that requires them to design, build, test, and program Autonomous and Driver-operated Robots that must perform a series of tasks.

The Playing Field for the Game consists of the *FIRST* Tech Challenge Game Pieces set up on a foam-mat surface, surrounded by a metal and Lexan Field frame. Each Tournament features Alliances, which are comprised of two Teams, competing against one another on the Playing Field. Teams work to overcome obstacles and meet challenges, while learning from and interacting with their peers and adult Mentors. Students develop a greater appreciation of science and technology and how they might use that knowledge to impact the world around them in a positive manner. They also cultivate life skills such as:

Details about setting up a Playing Field can be found on the FTC website after the yearly Game challenge Kickoff.

- Planning, brainstorming, and creative problem-solving.
- Research and technical skills.
- Collaboration and Teamwork.
- Appreciation of differences and respect for the ideas and contributions of others.

To learn more about FTC and other *FIRST* Robotics competitions, visit www.usfirst.org.

FIRST Tech Challenge (FTC) Core Values

Volunteers are integral to the *FIRST* community. The *FIRST* Tech Challenge relies on Volunteers to run the program at many levels, from managing a region to Mentoring an individual Team. FTC Affiliate Partners coordinate the program in each region or state. These FTC Partners fundraise, run Tournaments, hold workshops and demonstrations, market FTC locally, handle public relations, and recruit Volunteers and Teams. They are a tremendous resource for Mentors and FTC would not exist without them.

FIRST asks everyone who participates in FTC to uphold the following values:

- We act with integrity
- We are a Team.
- We do the work to get the job done with guidance from our Coaches and Mentors.
- We respect each other in the best spirit of Teamwork
- We honor the spirit of friendly competition.
- What we learn is more important than what we win.
- We behave with courtesy and compassion for others at all times
- We share our experiences with others.
- We display Gracious Professionalism in everything we do.
- We have fun.
- We encourage others to adopt these values.

What is the FIRST Tech Challenge PushBot: A Manual for TETRIX?

In early 2014, a design challenge was set by *FIRST,* for experienced FTC teams to design a basic robot and manual for Rookie FTC teams. The Robot and manual had to provide rookie teams with simple, basic instructions and images for building a PushBot that can do well in competition for any challenge.

The Push Bot (PB) needed to meet all FTC competition guidelines using the current Game Manual and Forum rules, but ONLY the competition kit and materials.

As part of the task, the Push Bots needed to have basic programming for autonomous mode; as well as a program for drive train. The Code in ROBOTC should include comments describing what each line of code means and does. The LabVIEW® code had to be copied into a document and comments written alongside.

Team 2843 United States – Push Bot Robot.

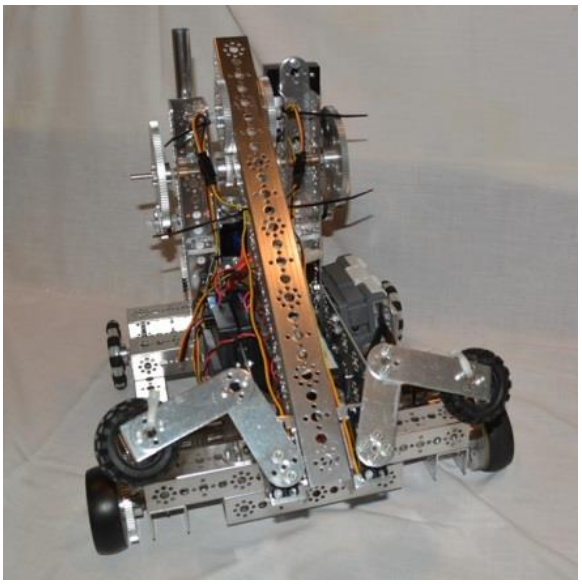
Robot Name: PB (Push Bot)

Purpose: Assist new FTC competitors to develop a sense of how an FTC Robot is constructed with the TETRIX Robotic Construction system and is programmed in LabVIEW® and ROBOTC.

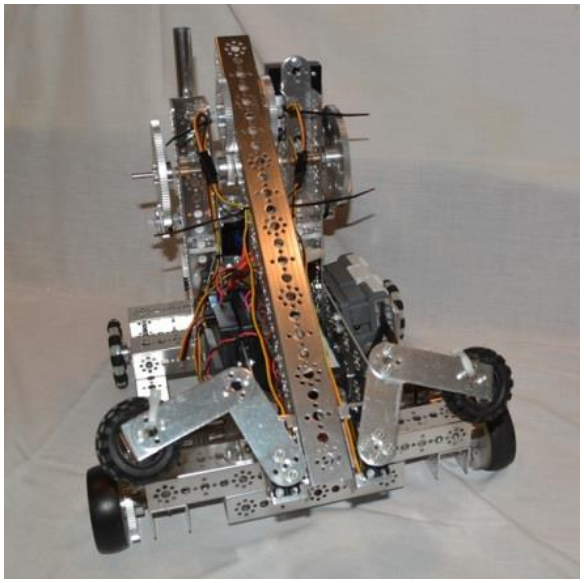
Design: The Push Bot has 2 DC motors, 2 servos and 2 LEGO motors. Its design is based around a rectangular frame. This allows structural Engineering principles to be explored by teams. The robot has an ‘arm’ powered by LEGO motors, a hand that is driven by servos, and feet powered with DC motors. The ’arm’ is geared down to allow the lower powered LEGO motors to lift it easily. PB has a front panel which can be used to ‘push’ items to various positions on the FTC Game floor. This guide will assist rookie FTC competitors gain an understanding of how a simple robot built with the TETRIX Construction System can be programmed in LabVIEW® or ROBOTC. It will also guide teams through using the Samantha module and pre game setup.

The robot is constructed with the TETRIX kit of parts, the resource kit, and the LEGO Mindstorms kit.

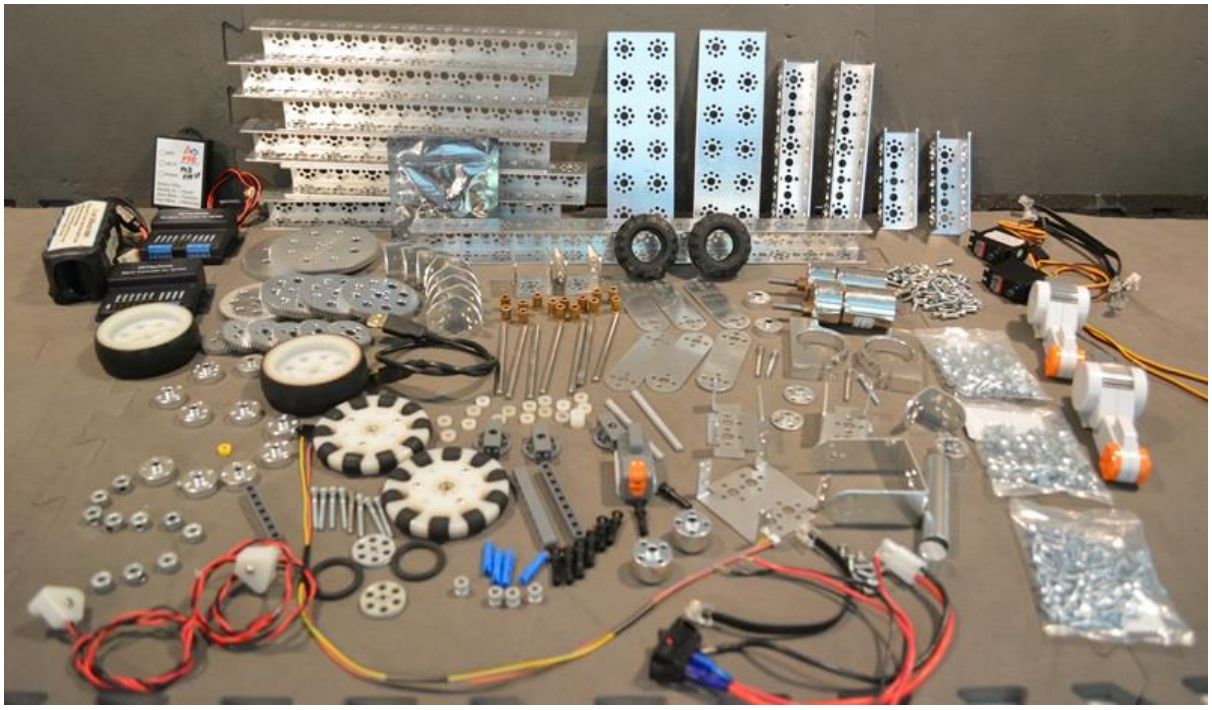
See http://www.usfirst.org/roboticsprograms/ftc/tetrixkop



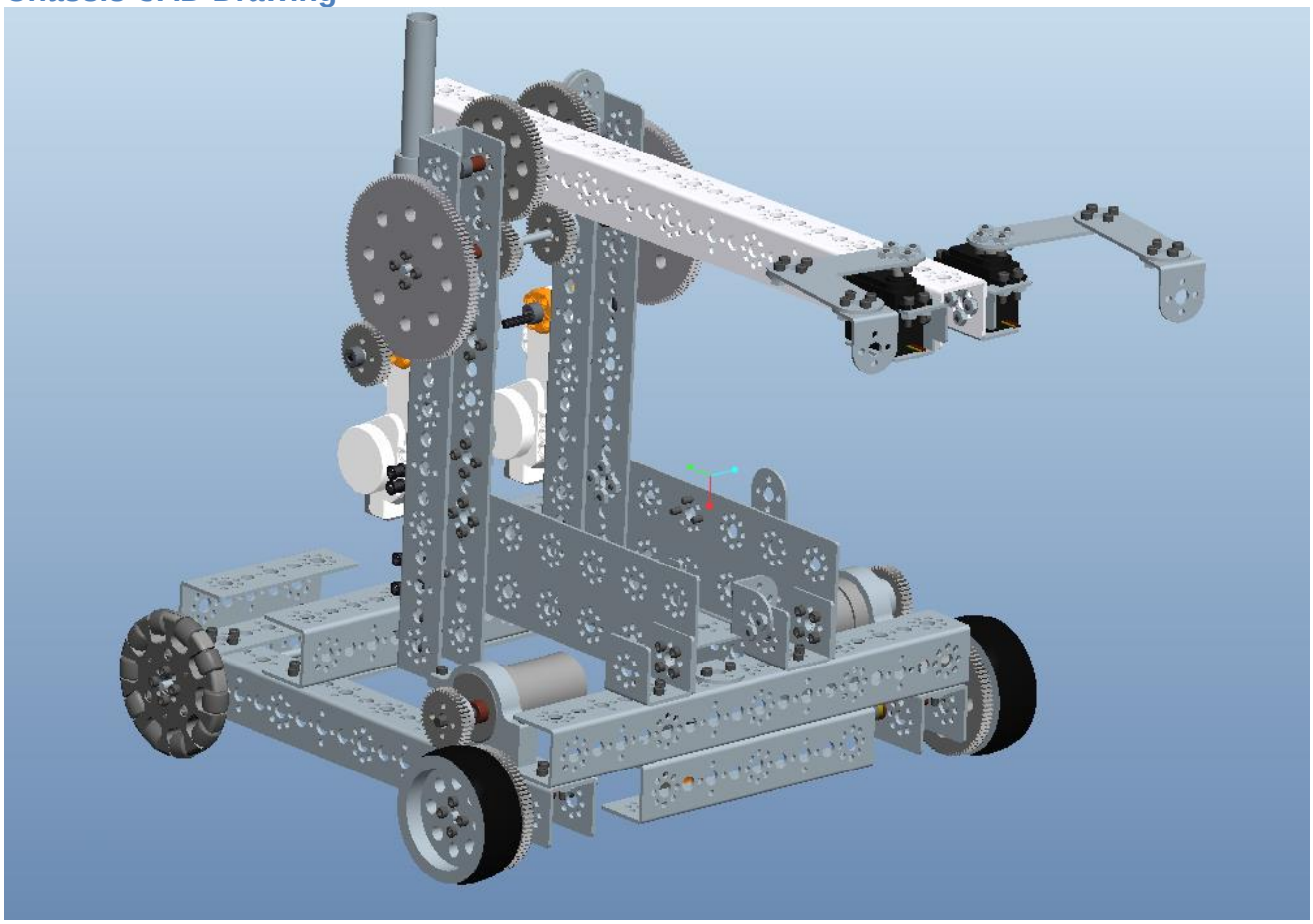
Building the Chassis



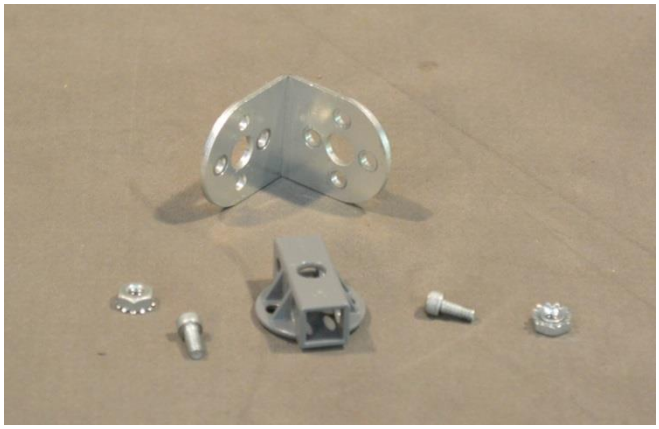
This section will outline the construction of the Chassis, Sensors, NXT and Samantha module. The chassis is constructed using the Kit of Parts and Resource TETRIX sets. The standard tools included in the kit of parts set will be needed to build the chassis. All parts needed for the construction are listed in the instructions (most are shown below).

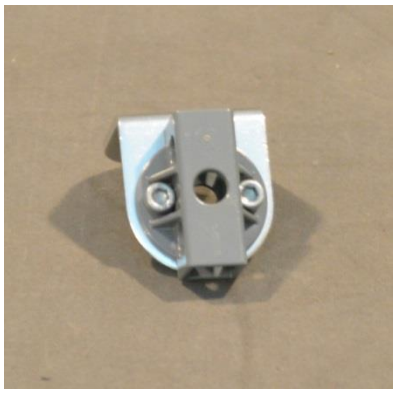


Chassis CAD Drawing

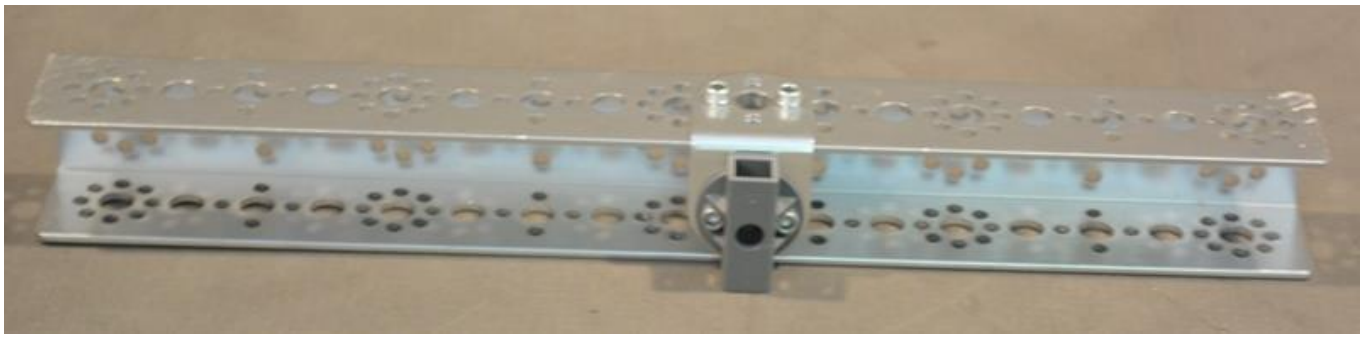


Step 1: L Bracket (1), 1/2” socket head cap screws (2), kep nuts (2), hard point connector (1)

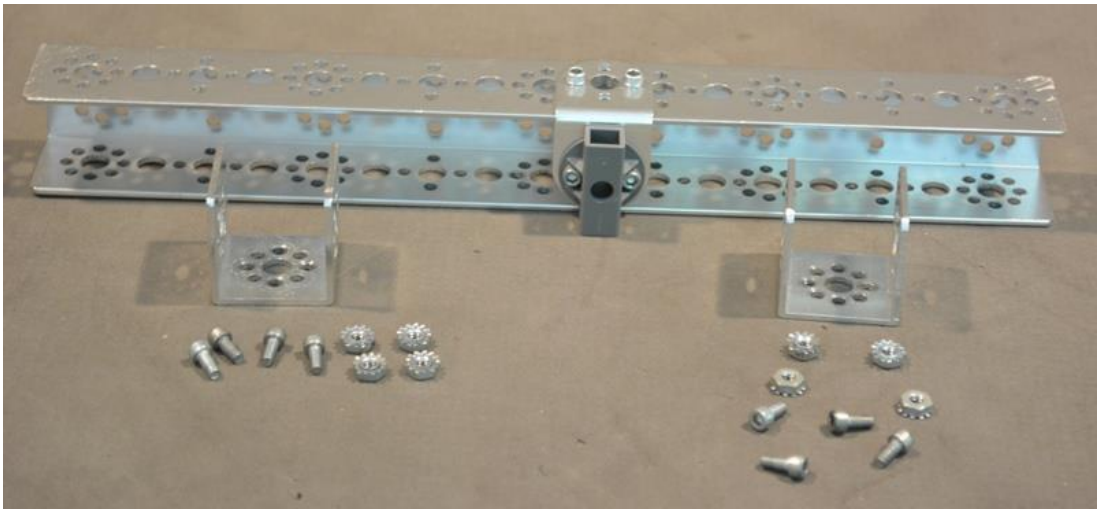


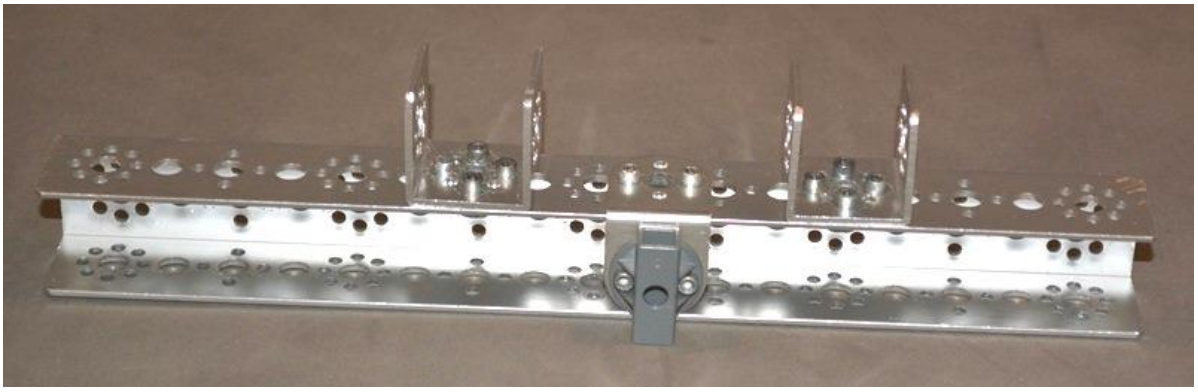


Step 2: assembly from step 1, 288mm channel (1), 1/2” socket head cap screws (2), kep nuts (2)

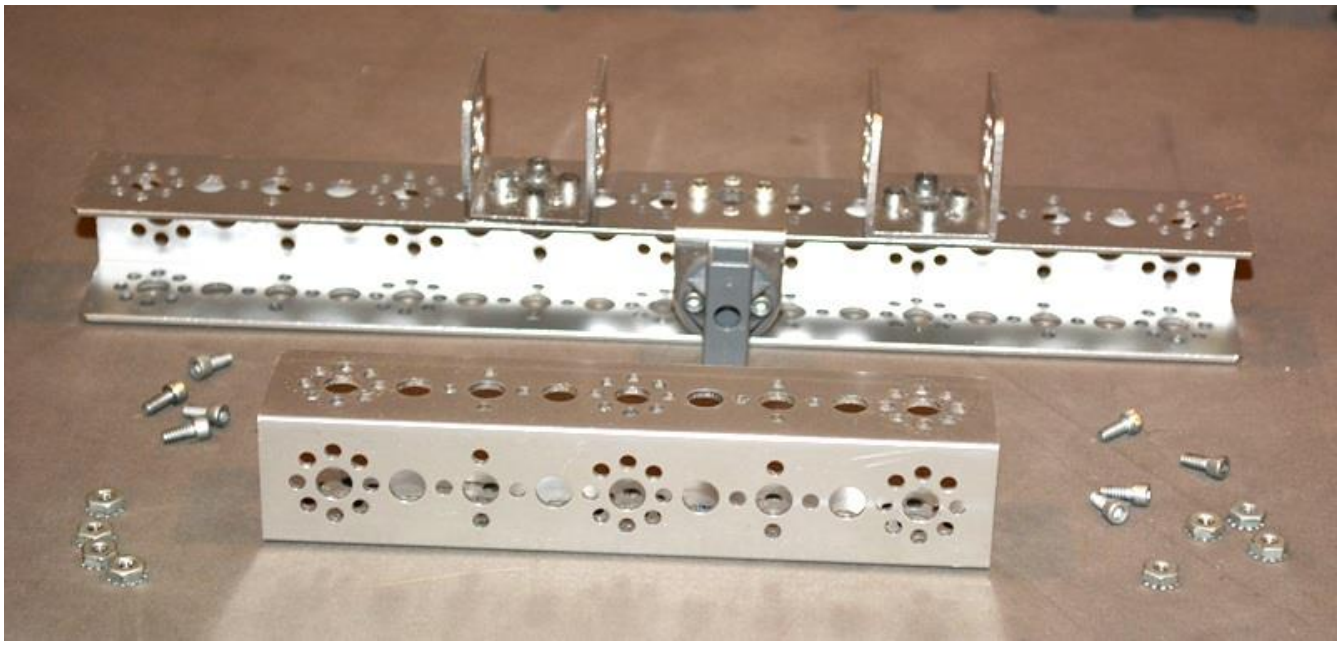


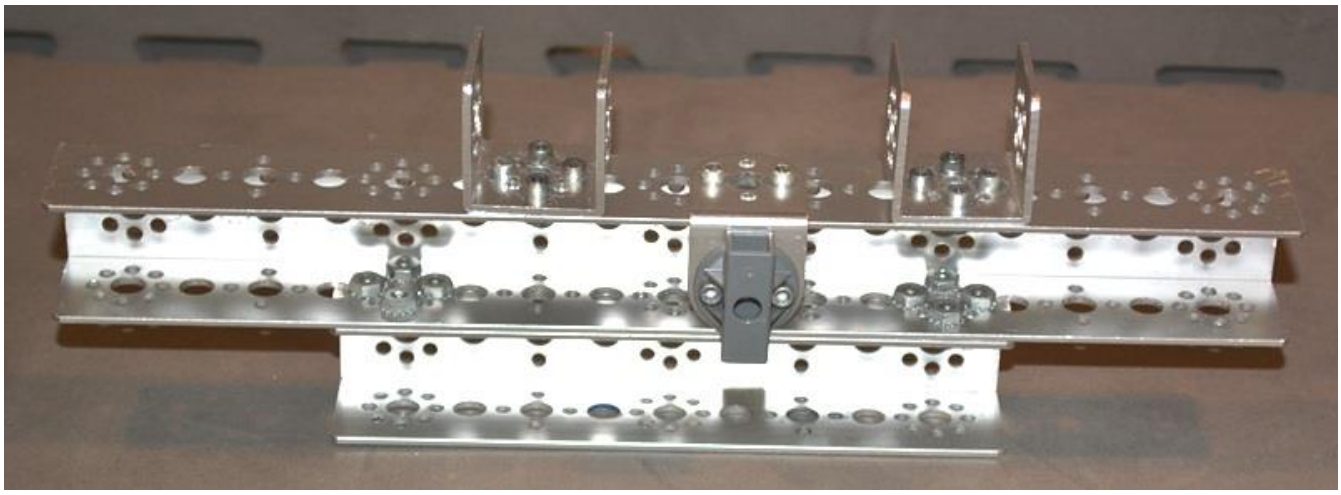
Step 3: assembly from step 2, 32mm channel (2), 5/16” socket head cap screws (8), kep nuts (8)



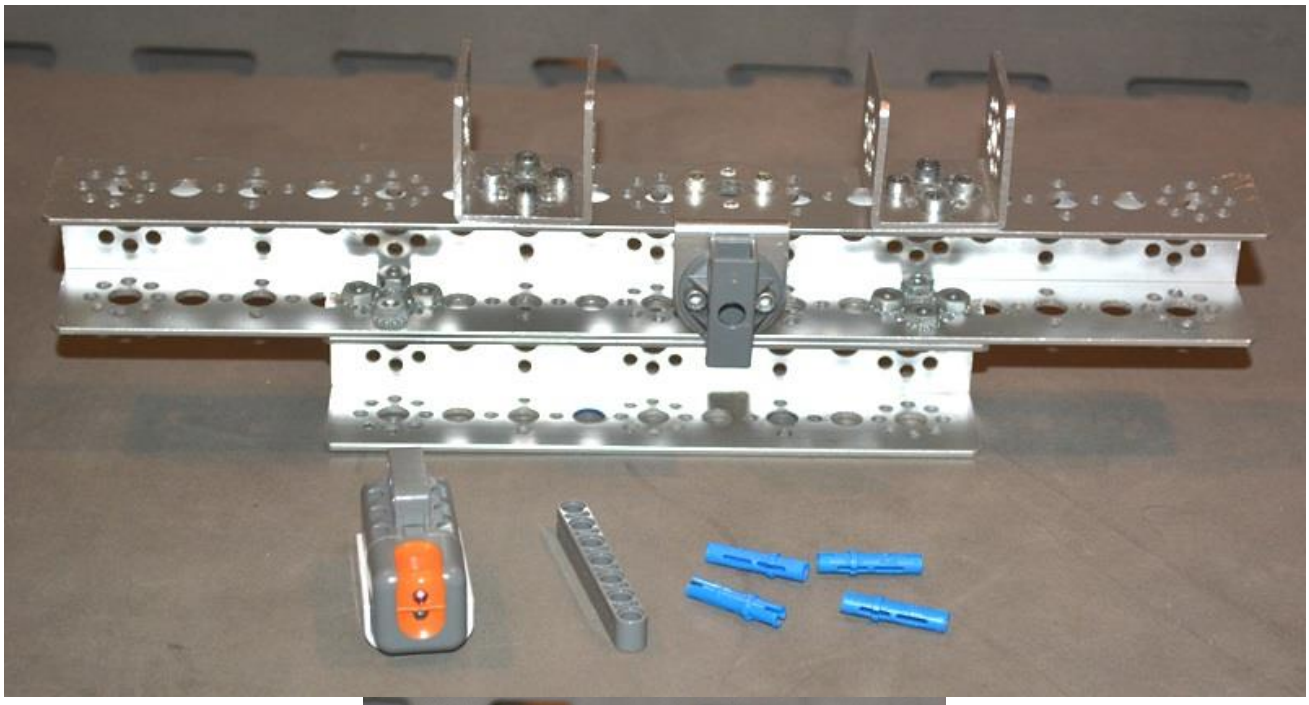


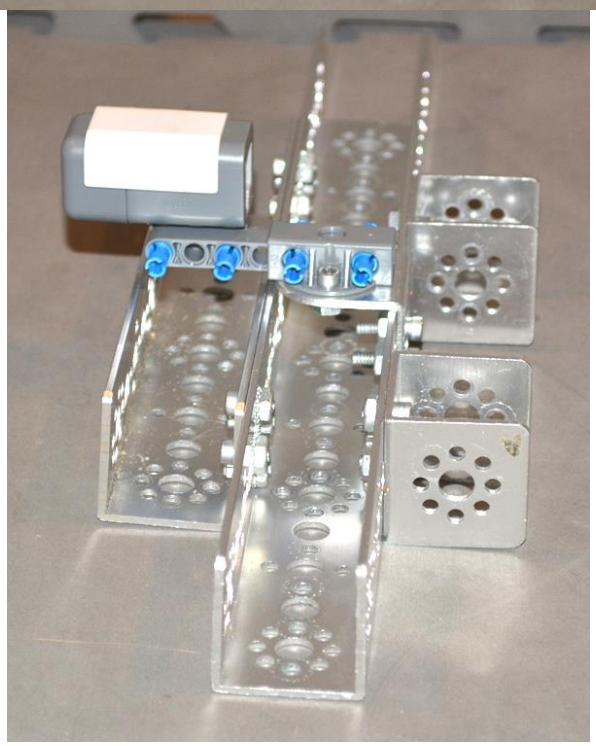
Step 4: assembly from step 3, 160mm channel (1), 5/16” socket head cap screws (8), kep nuts (8)



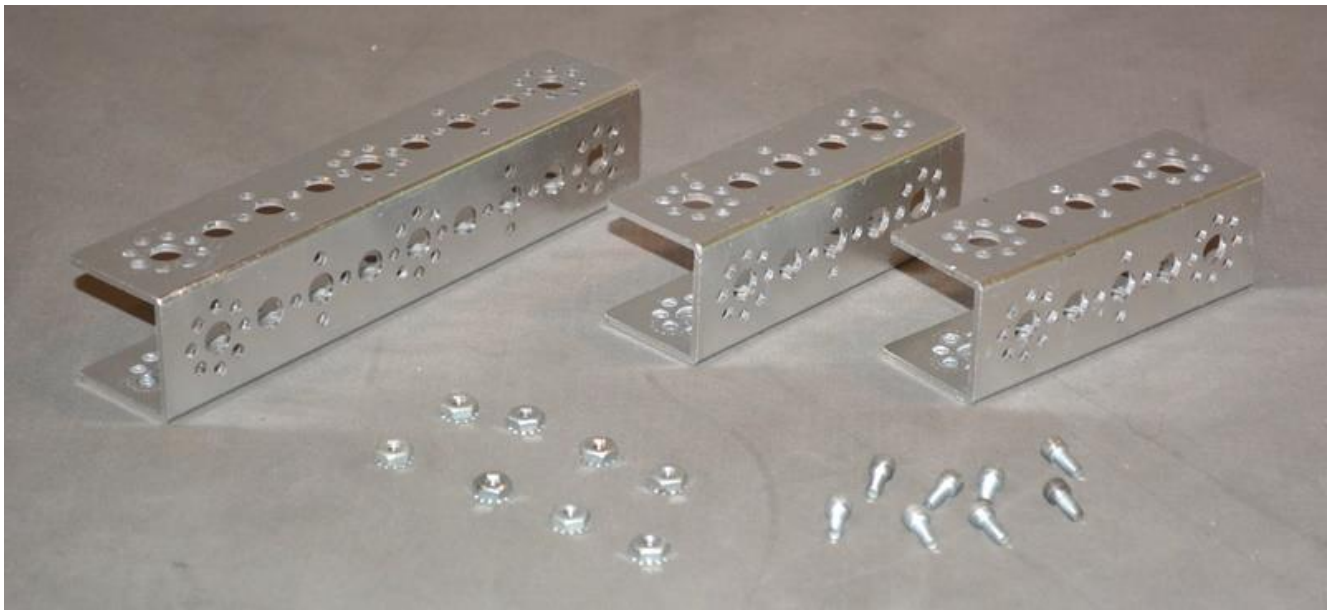


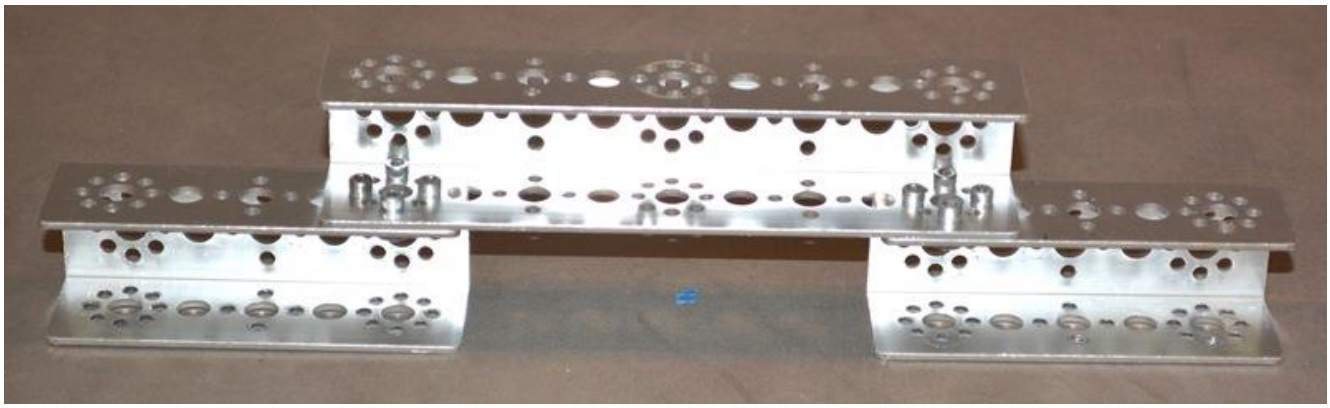
Step 5: assembly from step 4, light sensor (1), 7-module dark grey (1) beam, 3-module blue connector peg with friction (4)



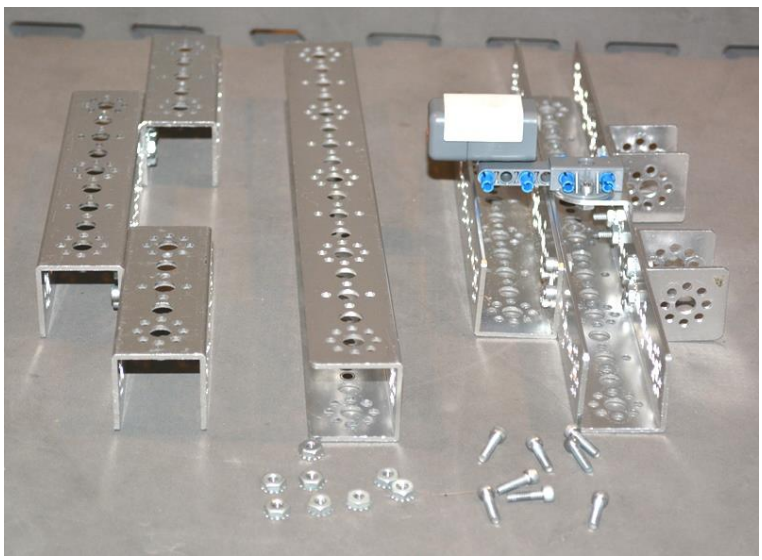


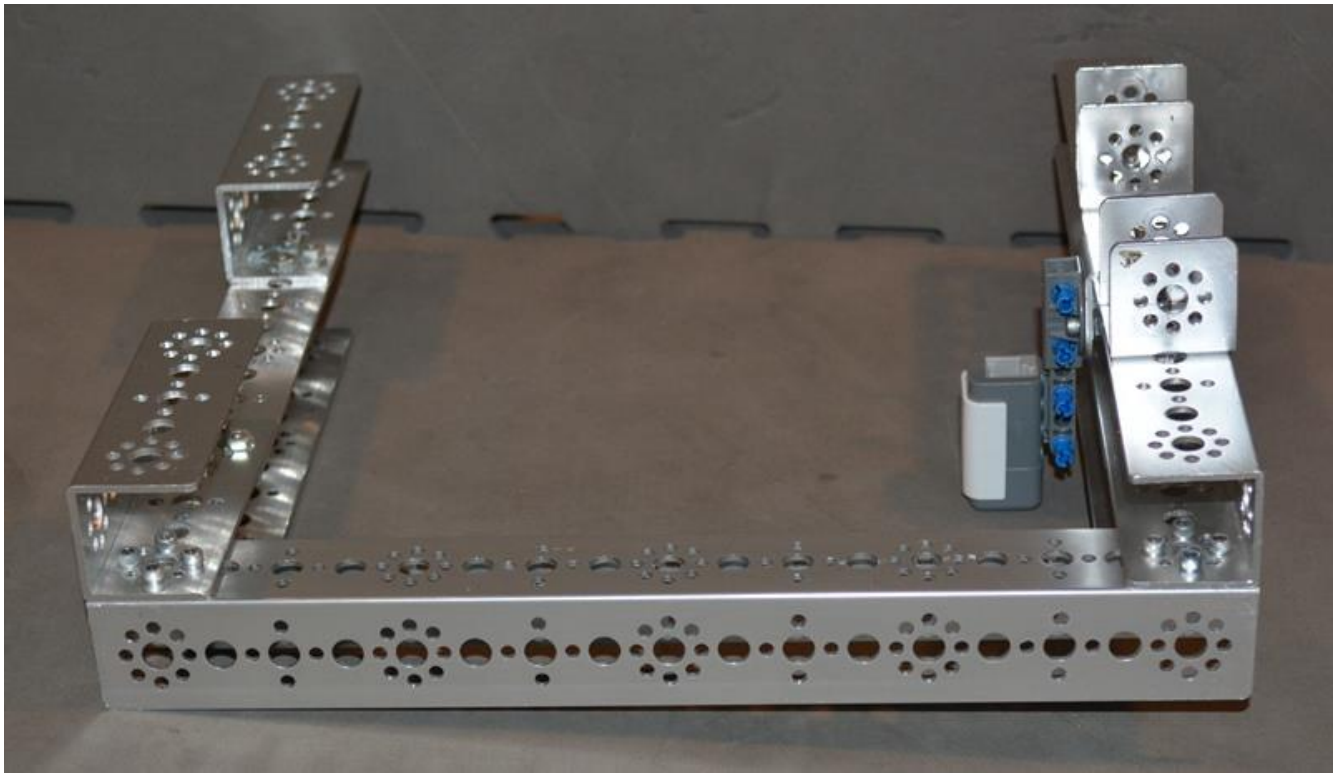
Step 6: 160mm channel (1), 96mm channel (2), 5/16” socket head cap screws (8), kep nuts (8)



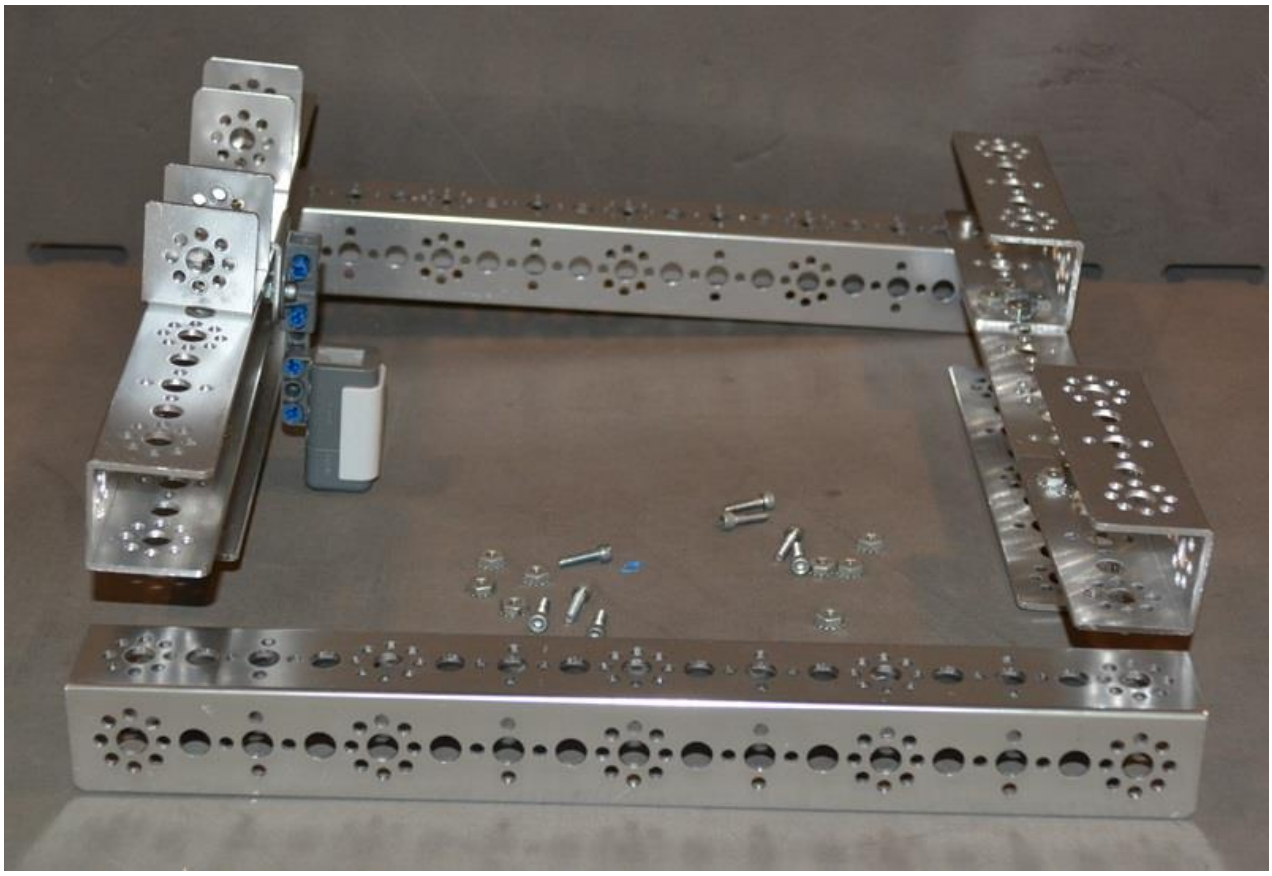


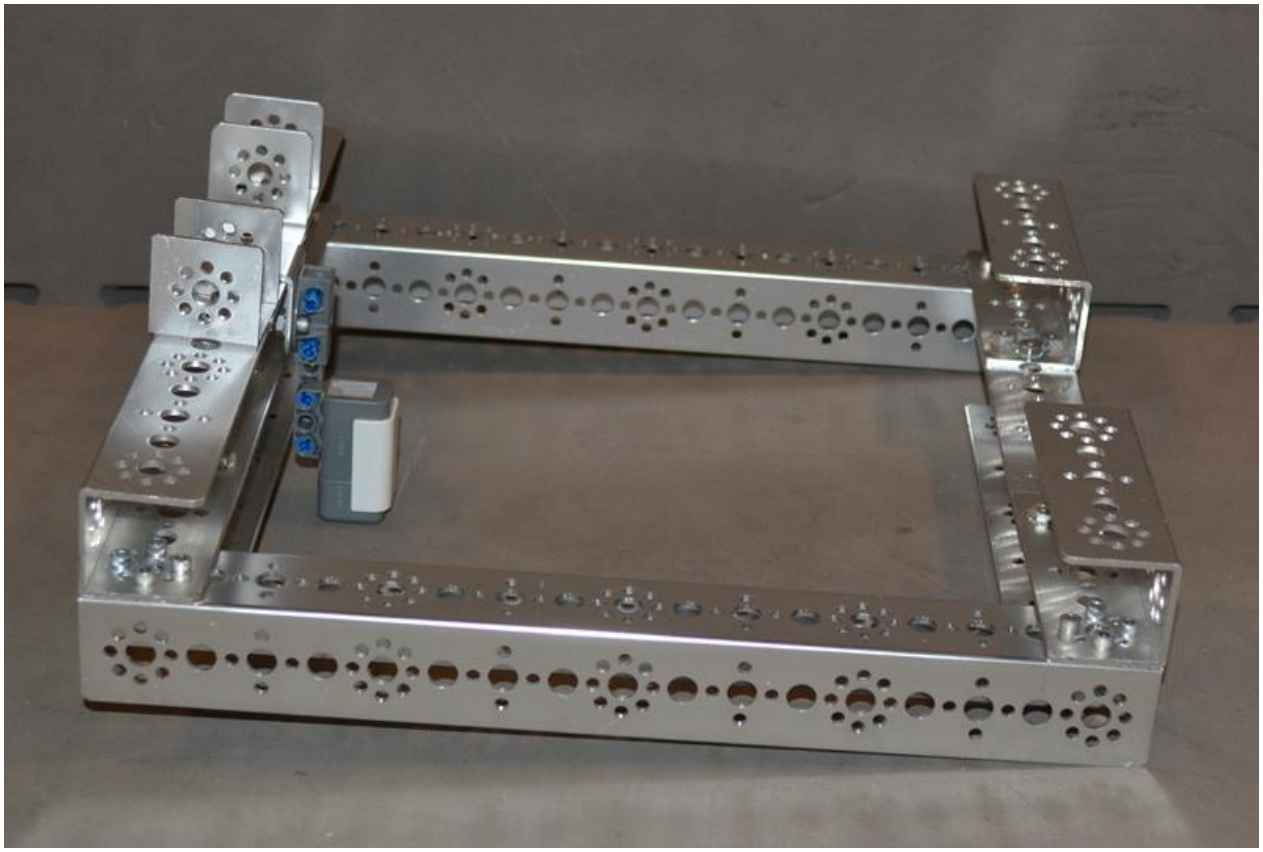
Step 7: assemblies from steps 5 and 6, 288mm channel (1), 1/2” socket head cap screws (8), kep nuts (8)



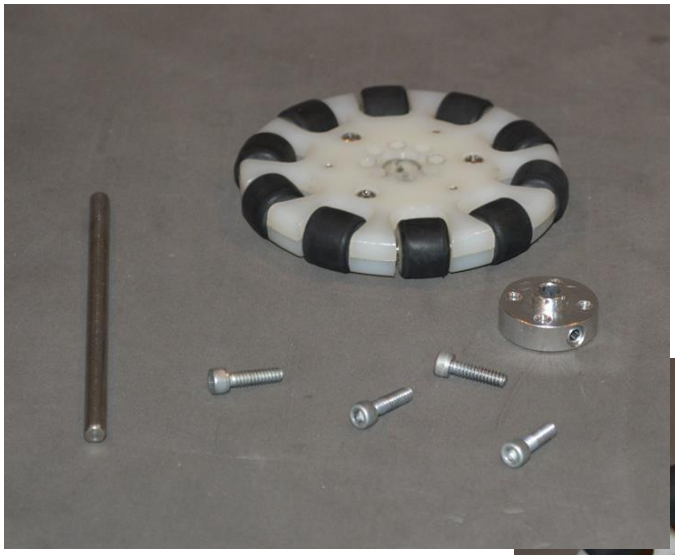


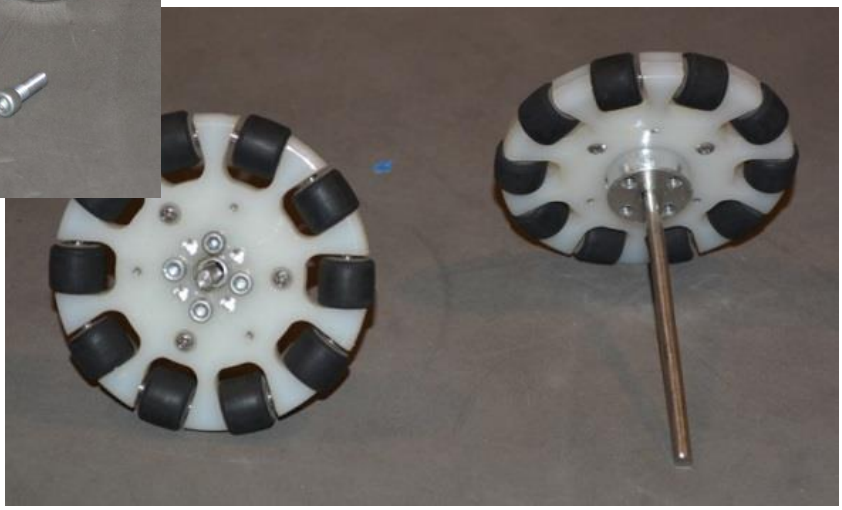
Step 8: assembly from step 7, 288mm channel (1), 1/2” socket head cap screws (8), kep nuts (8)



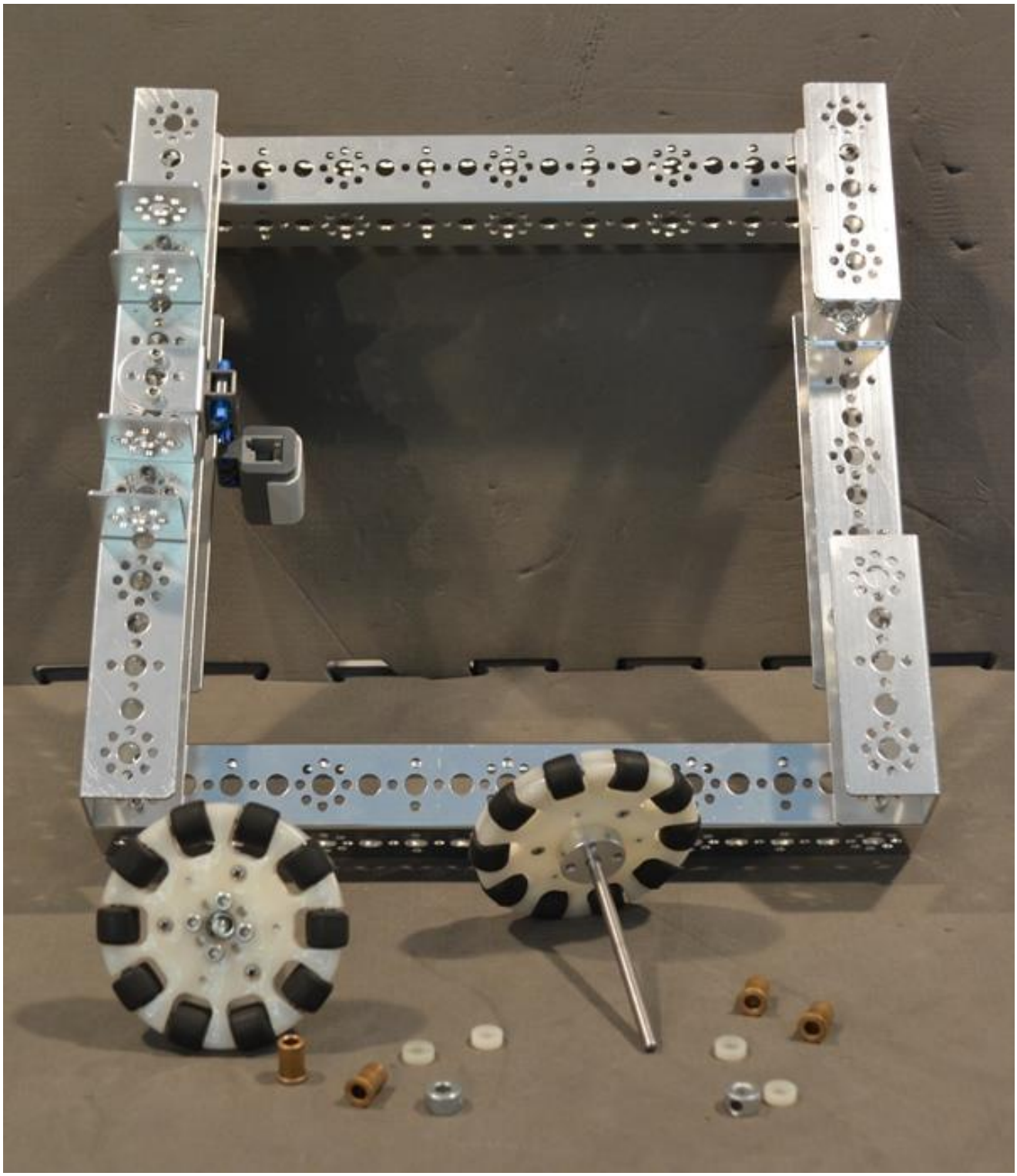


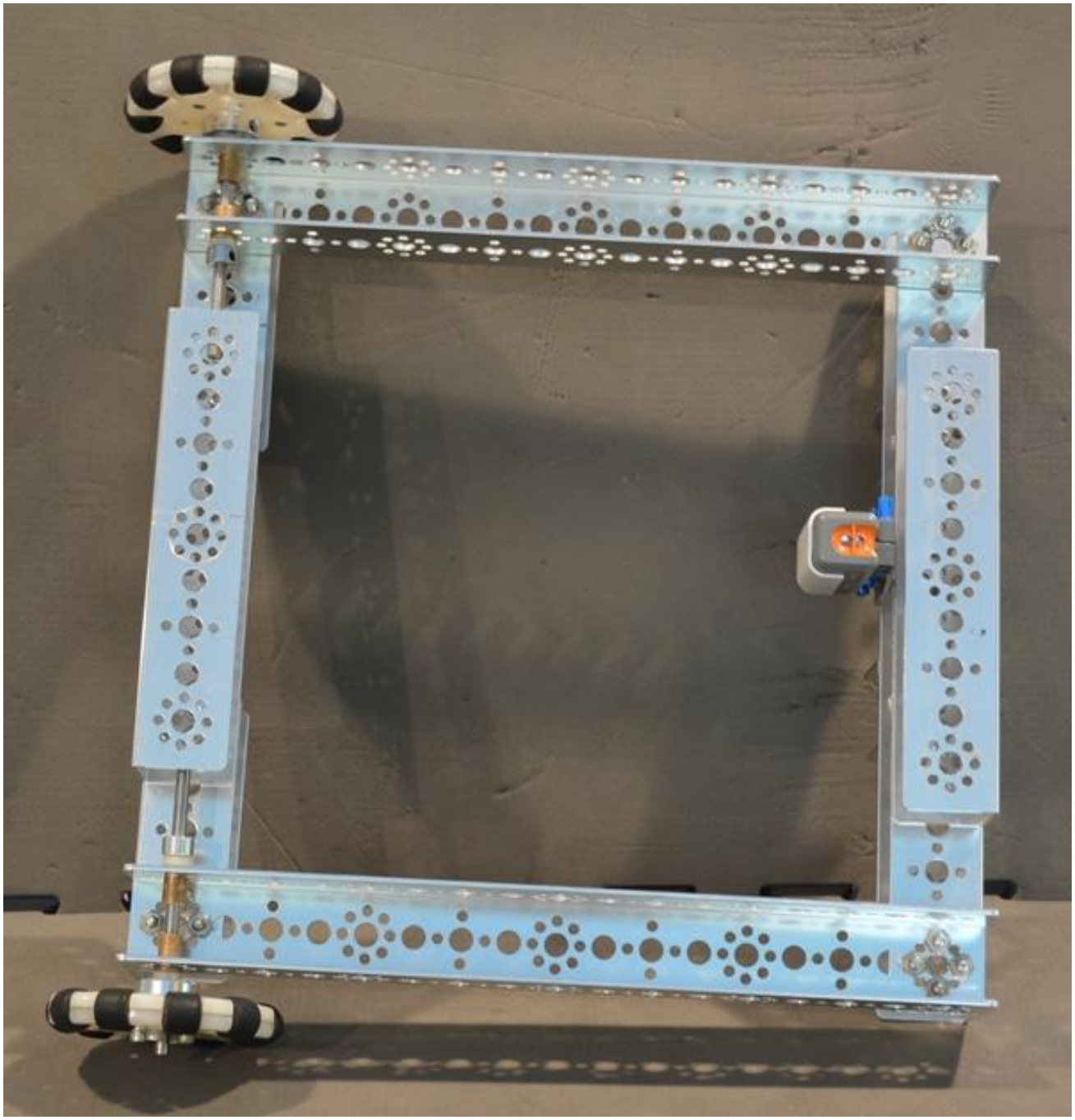
Steps 9: (make two):100mm axle (1), omni-wheel (1), axle hub (1), 1/2” socket head cap screws (8), kep nuts (8)





Step 10: assemblies from steps 8 and 9, bronze bushing (4), axle spacer (4), axle set collar (2)

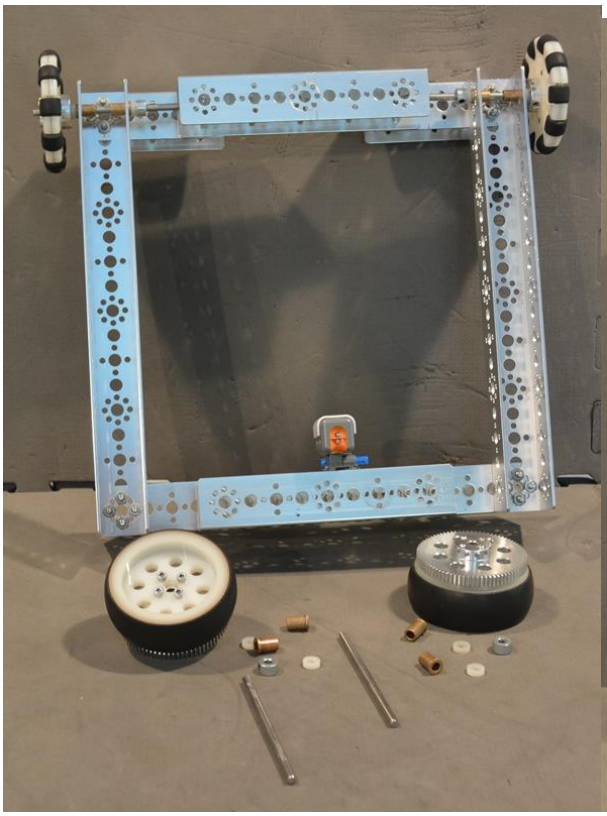


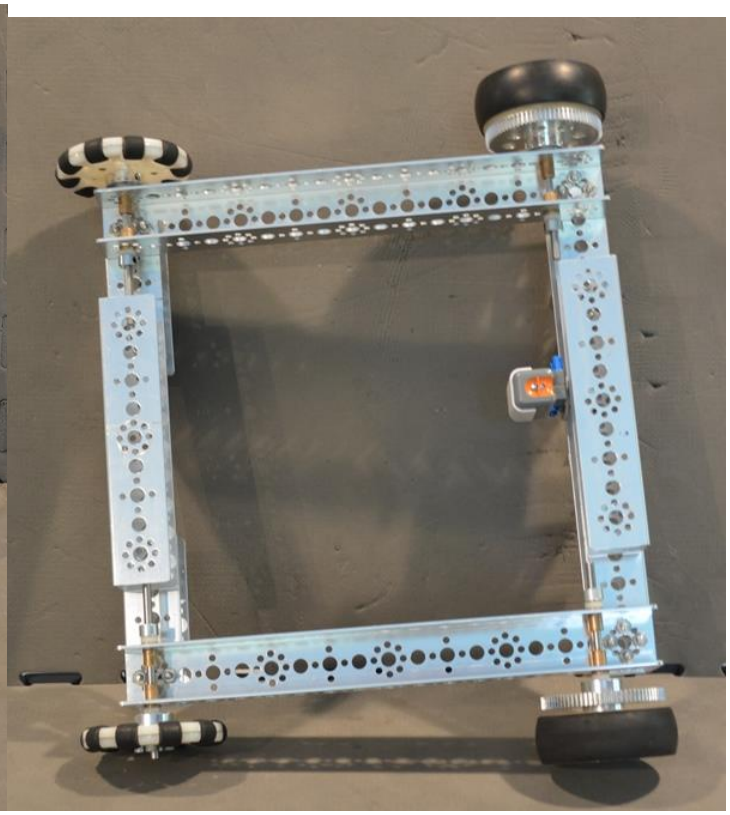


Step 11: 3” wheel (2), gear hub spacer (2) (and bolts that are packaged with the spacer), 80-tooth gear (2), axle hub (2)

The order is wheel, spacer, gear, hub.

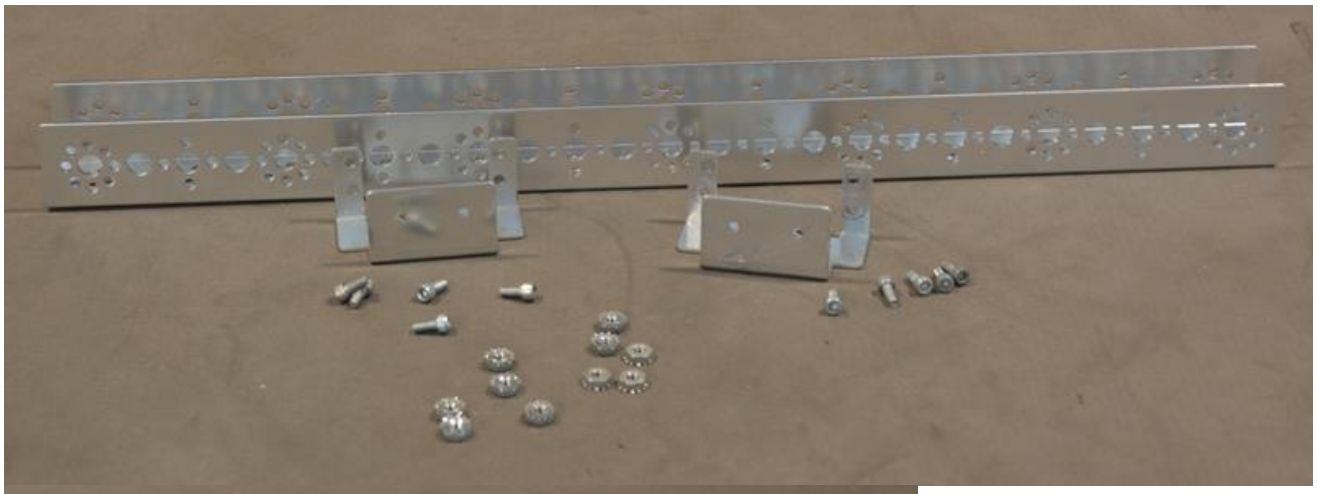
Step 12: assemblies from steps 10 and 11, 100mm axle (2), bronze bushing (4), axle spacer (4), axle set collar (2)

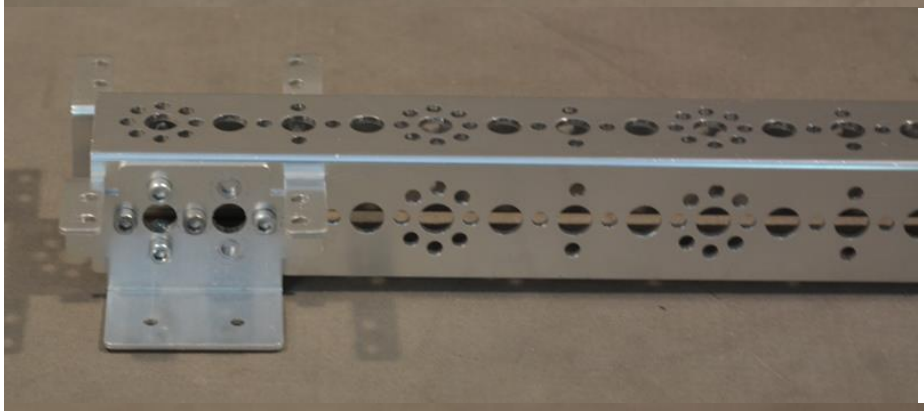


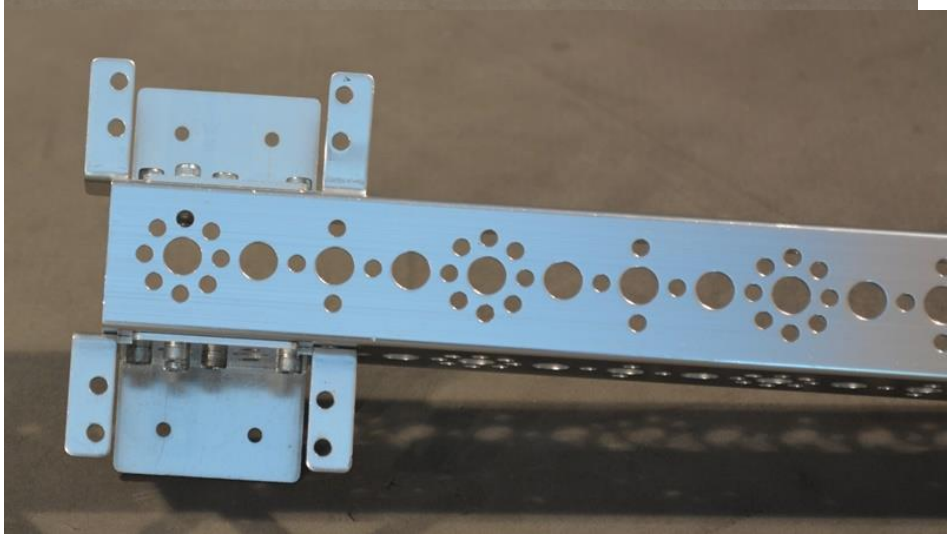


Note that the wheels are <u>not</u> at the end of the channel.

Step 13: 416mm channel (1), single-servo motor bracket (2), 5/16” socket head cap screws (8), kep nuts (8)

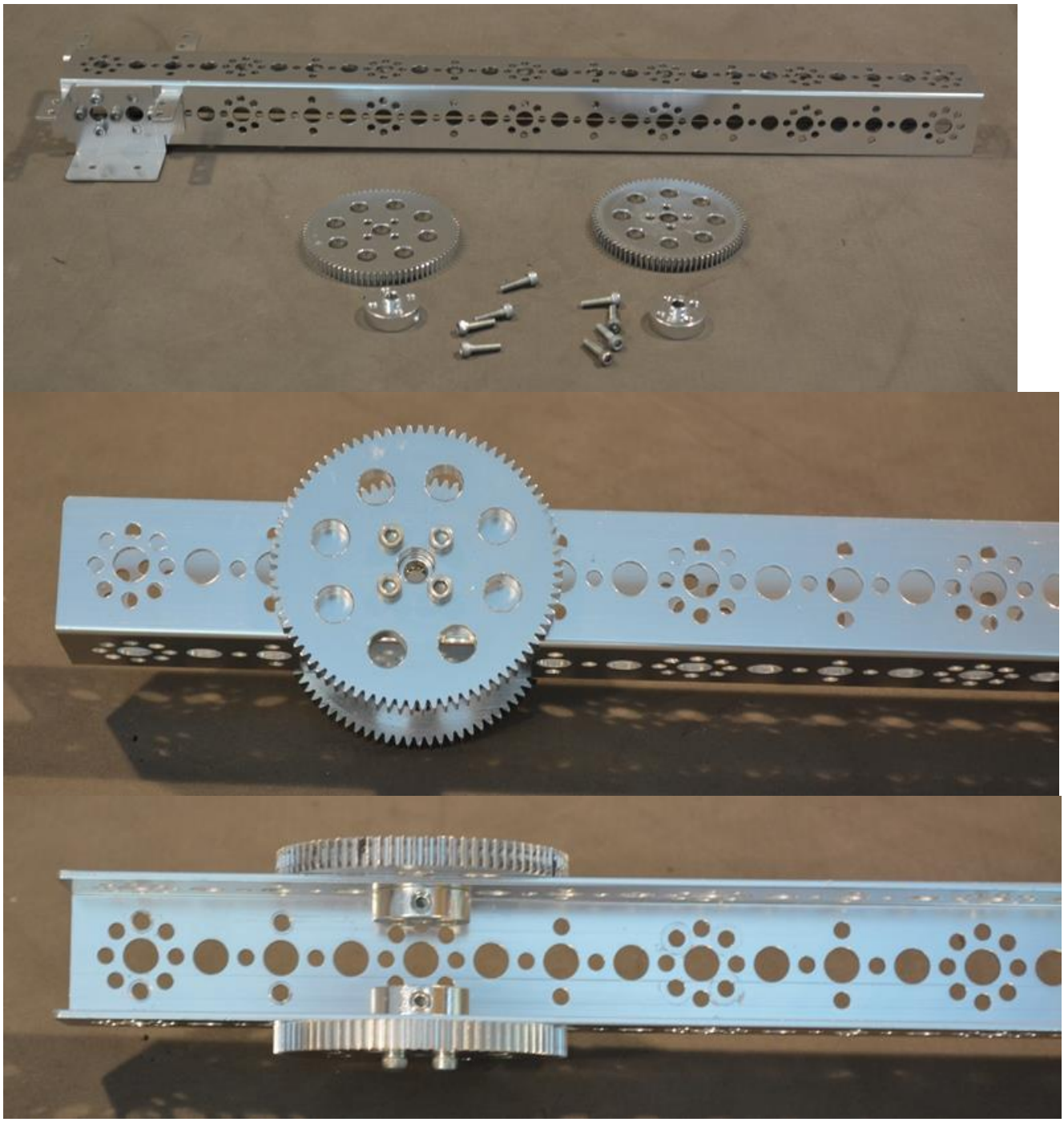






Do not make this part twice --- The two bottom photos show different views of the same assembly.

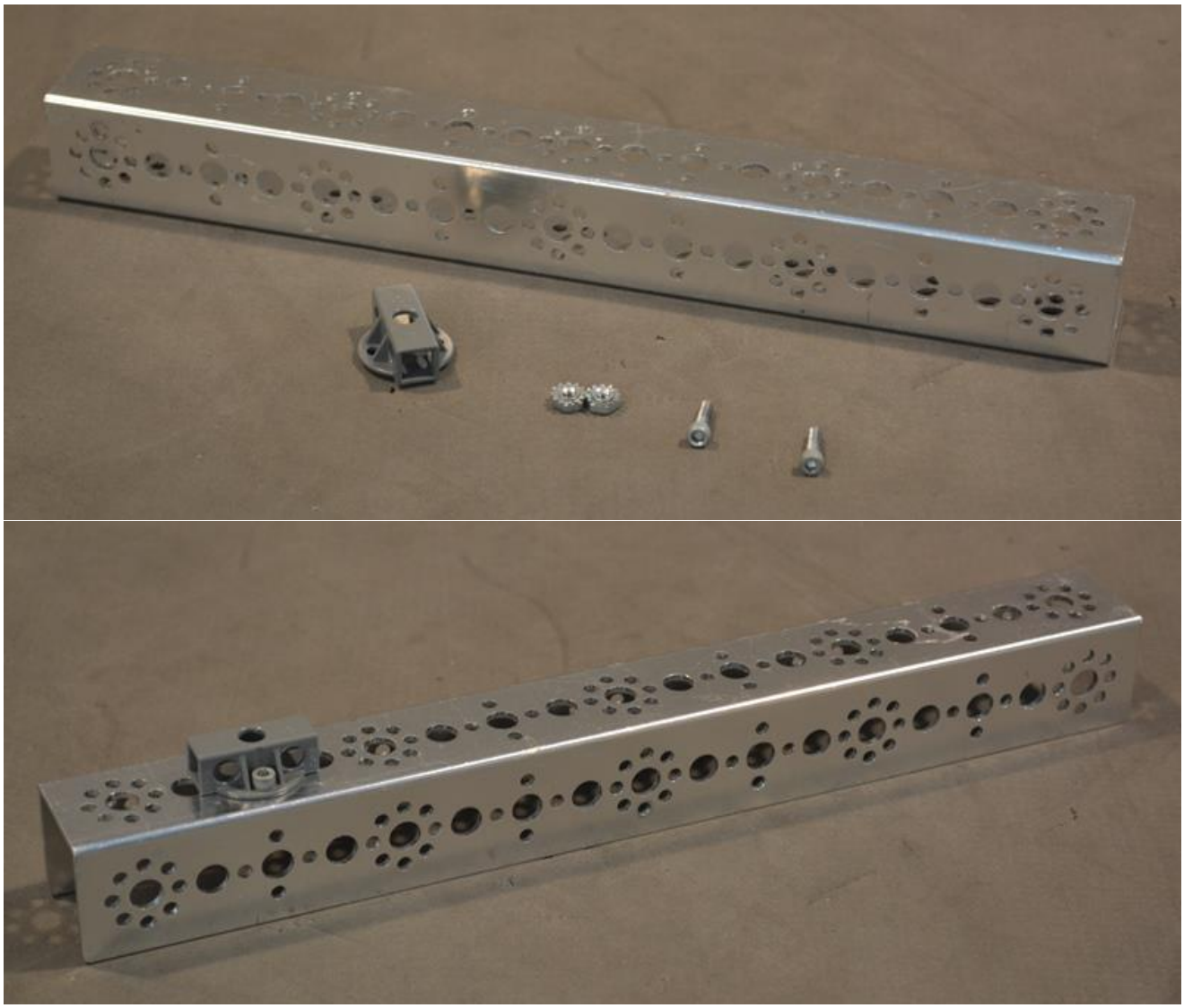
Step 14: assembly from step 13, 80-tooth gear (2), axle hub (2), 1/2” socket head cap screws (8)



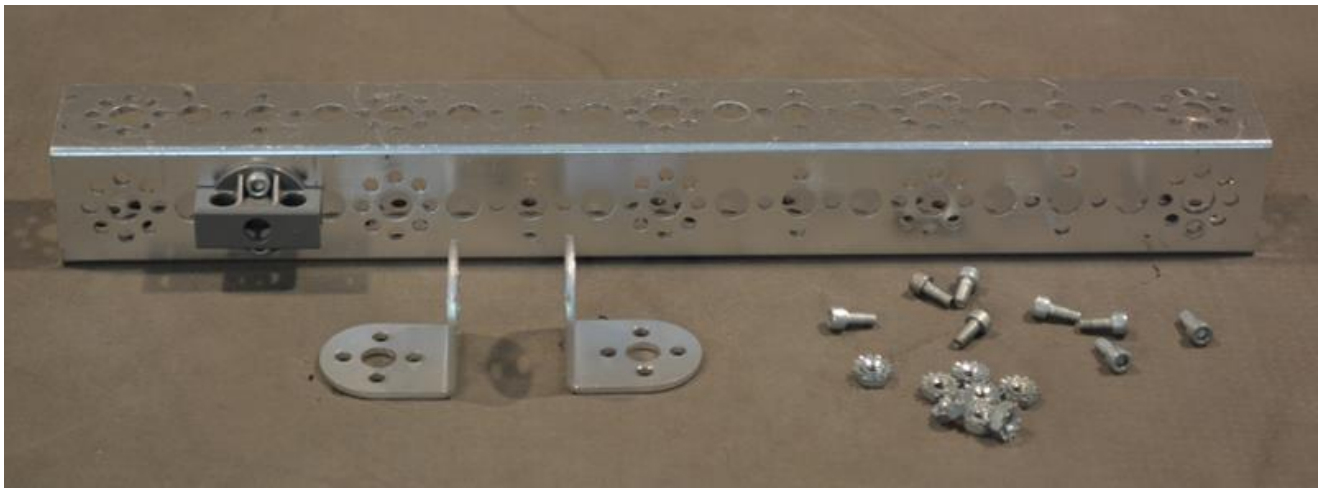
Do not make this part twice --- The two bottom photos show different views of the same assembly.

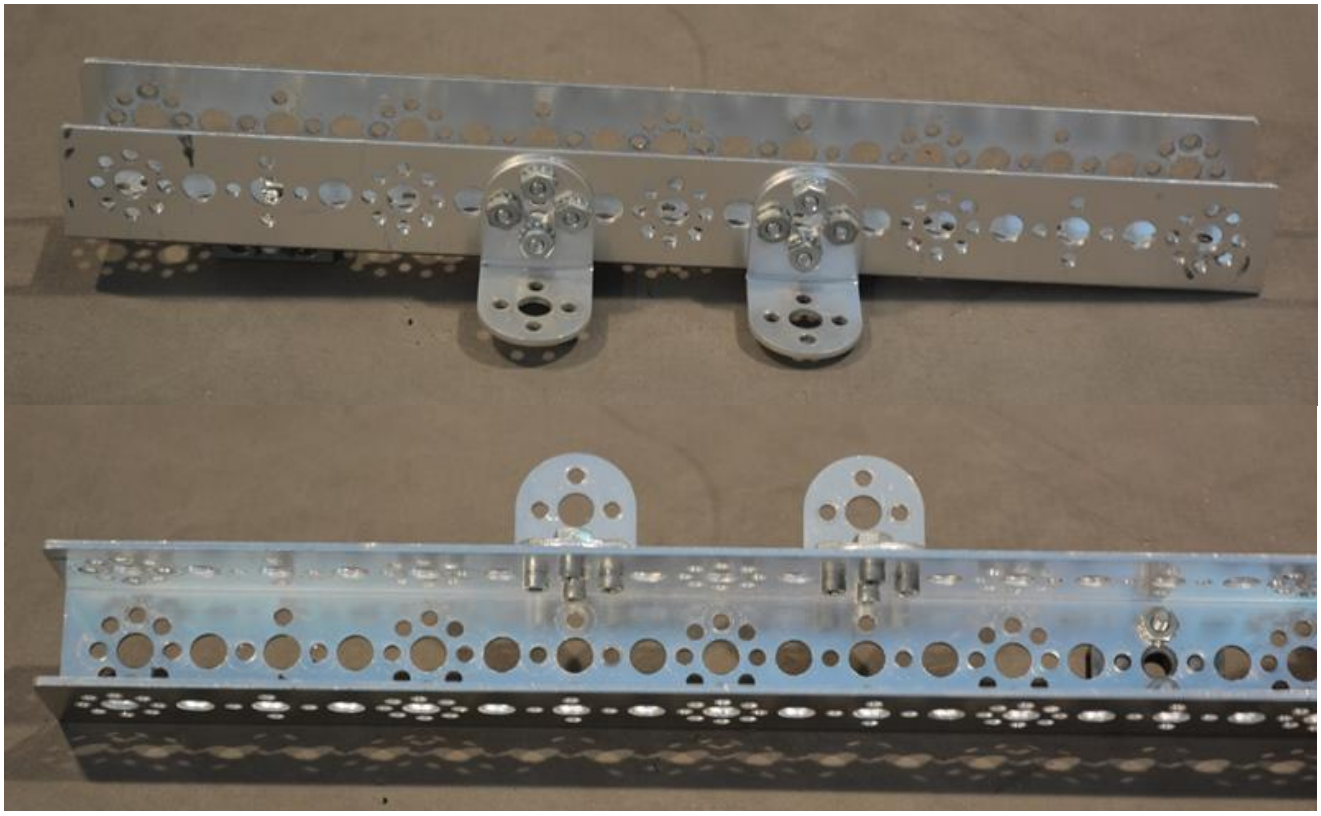
Note that the gears are on the other side of the beam from the servo motor brackets.

Step 15 (make two): 288mm channel (1), 1/2” socket head cap screws (2) , 1/2” socket head cap screws (2), kep nuts (2), hard point connector (1)



Step 16: one of the assemblies from step 15, L bracket (2), 5/16” socket head cap screws (8), kep nuts (8), hard point connector (1)

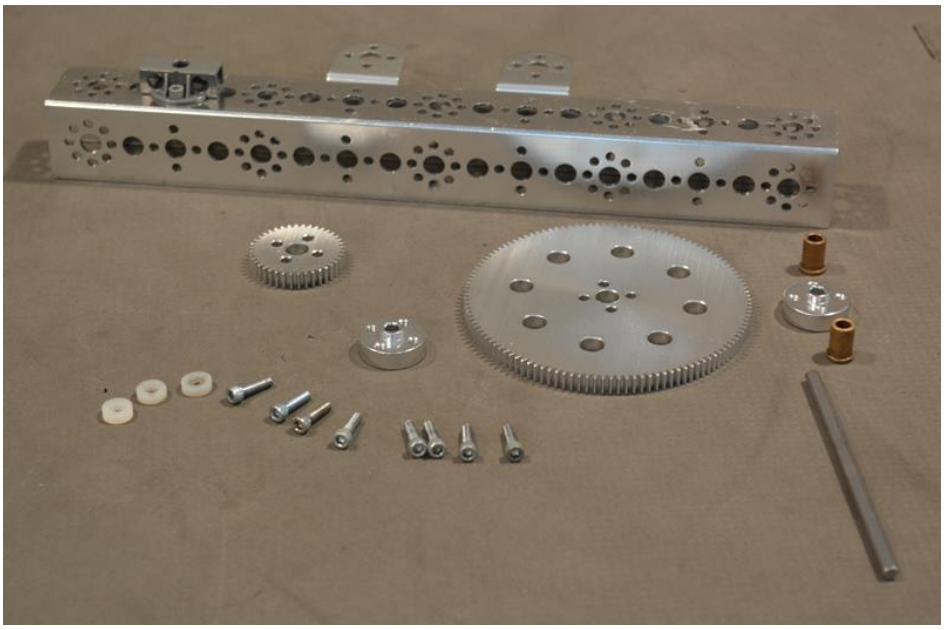


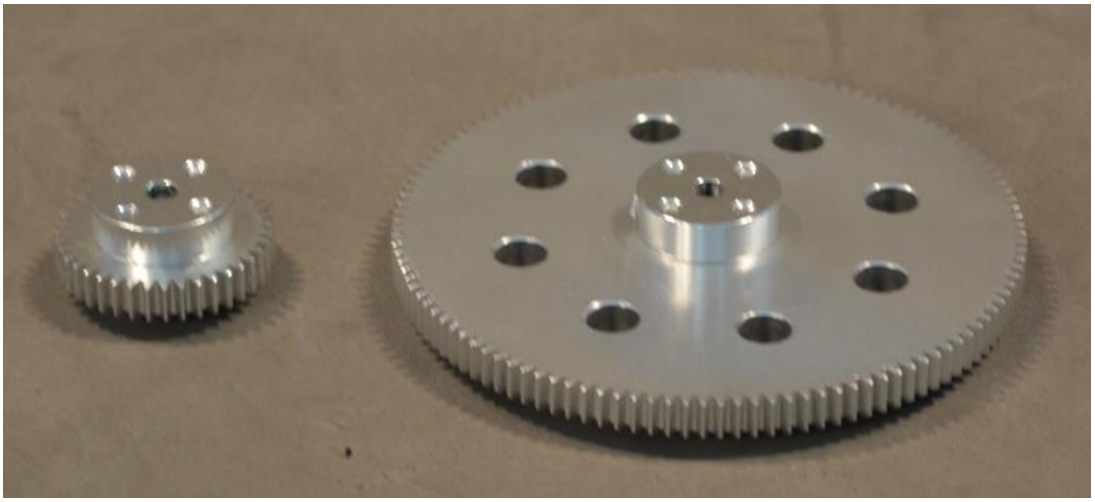


Do not make this part twice --- The two bottom photos show different views of the same assembly.

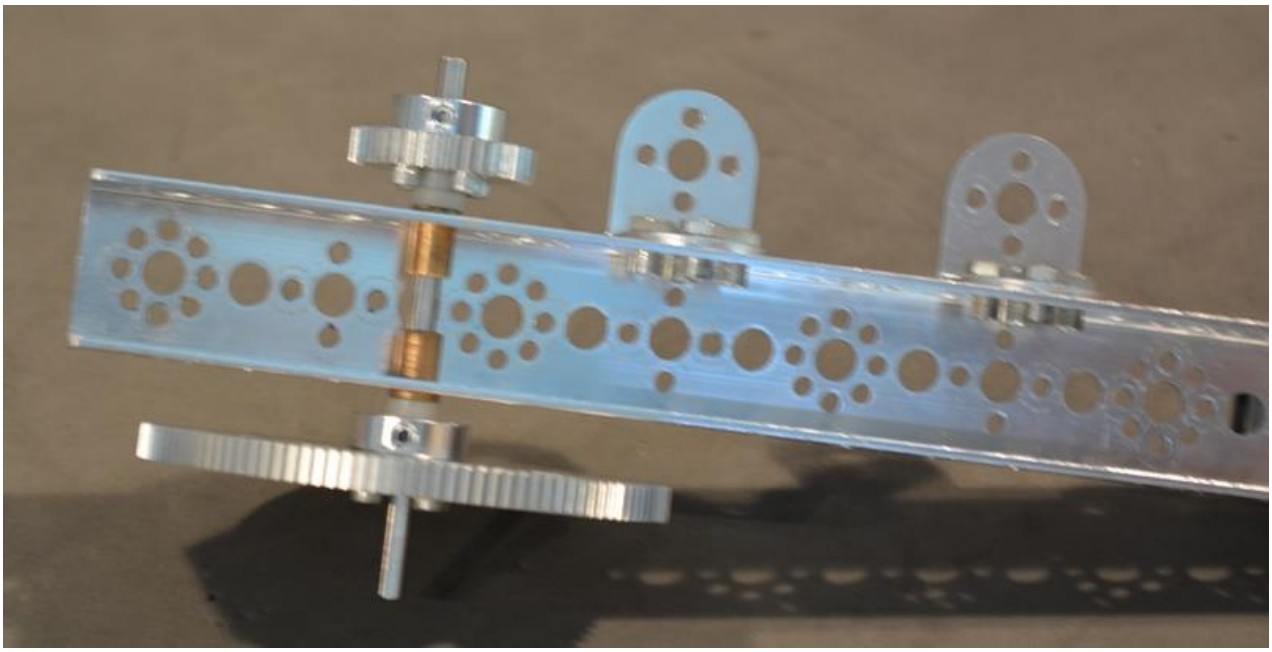
Note that the heads of the bolts are on the inside of the channel. The kep nuts are on the outside of the channel. When this assembly and the other tower assembly (next step) are mounted on the wheel base, the L brackets will be next to each other and an Allen wrench will not be able to fit between the beams.

Step 17: assembly from step 16, 120-tooth gear (1), 40-tooth gear (1), 100mm axle (1), axle hub (2) 3/8” axle spacer (3), bronze bushing (2), 5/16” socket head cap screws (8)



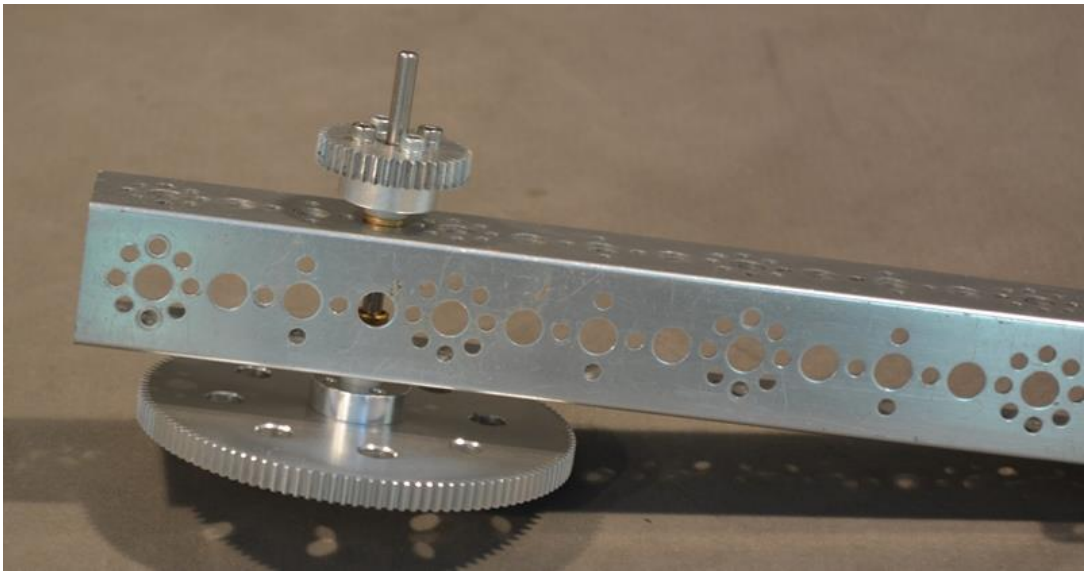


Use 4 of the bolts to secure the axle hubs to the gears (above) before completing this step (below).



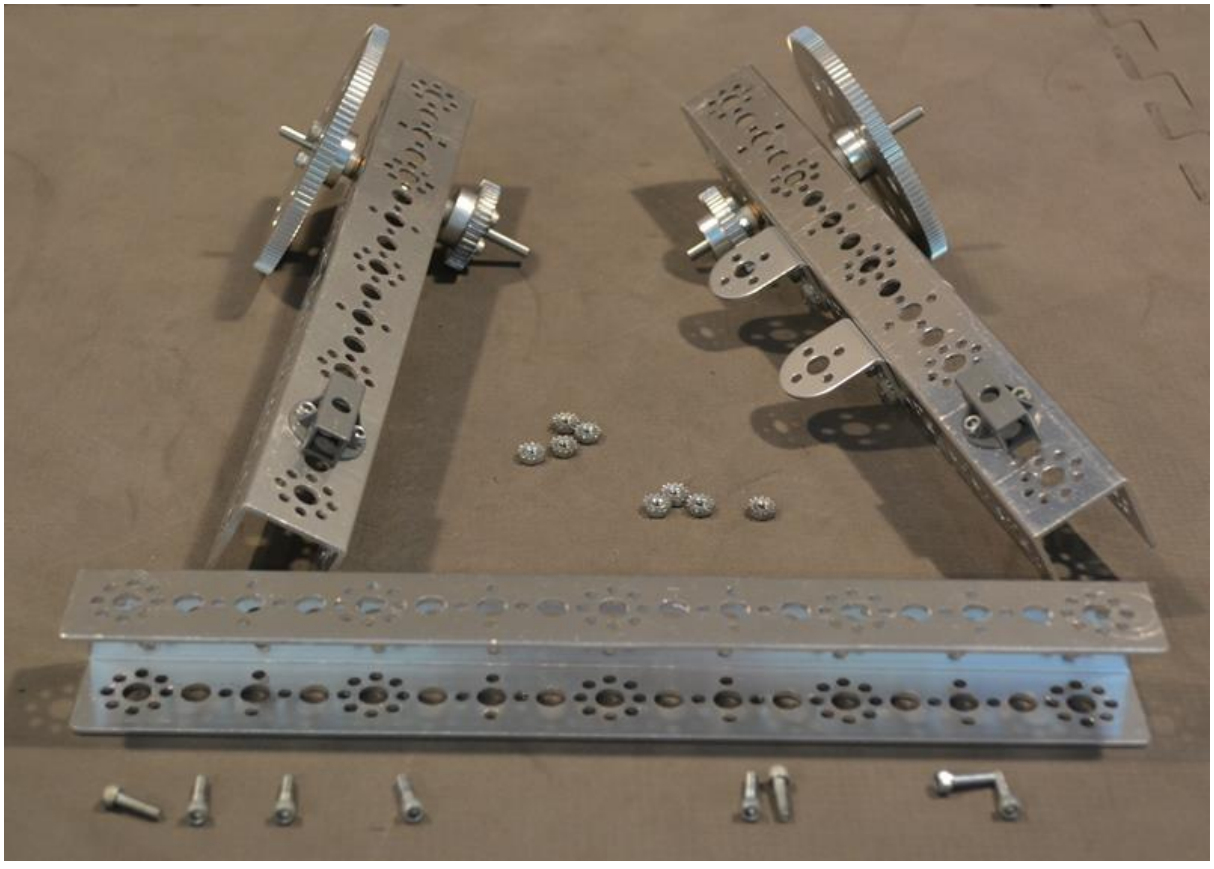
Note the sequence from top to bottom is axle hub, 40-tooth gear, spacer, bronze bushing, side of channel, bronze bushing, second bronze bushing, spacer, axle hub, and 120-tooth gear. The sequence is important, so these gears align with the LEGO motor gears.

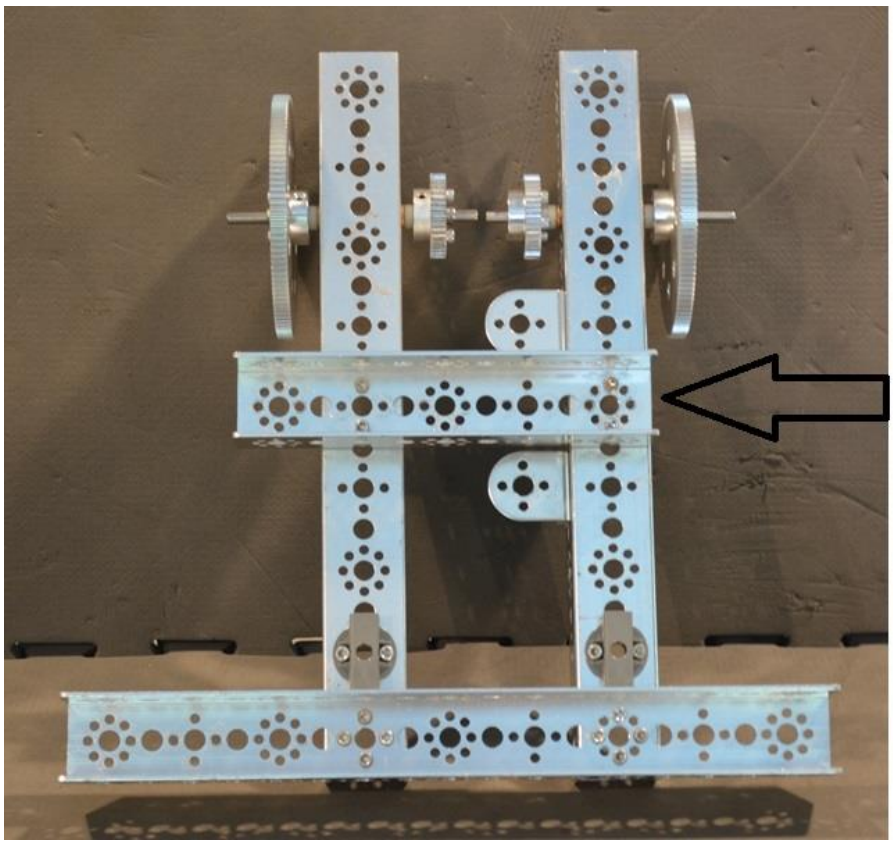
Step 18: the other assembly from step 15, 120-tooth gear (1), 40-tooth gear (1), 100mm axle (1), axle hub (2), 3/8” axle spacer (3), bronze bushing (2), 5/16” socket head cap screws (8) (these parts are shown in step 17) Follow the steps from step 17 to build the below assembly. Compare the image below with the image in step 19, which shows both completed assemblies.



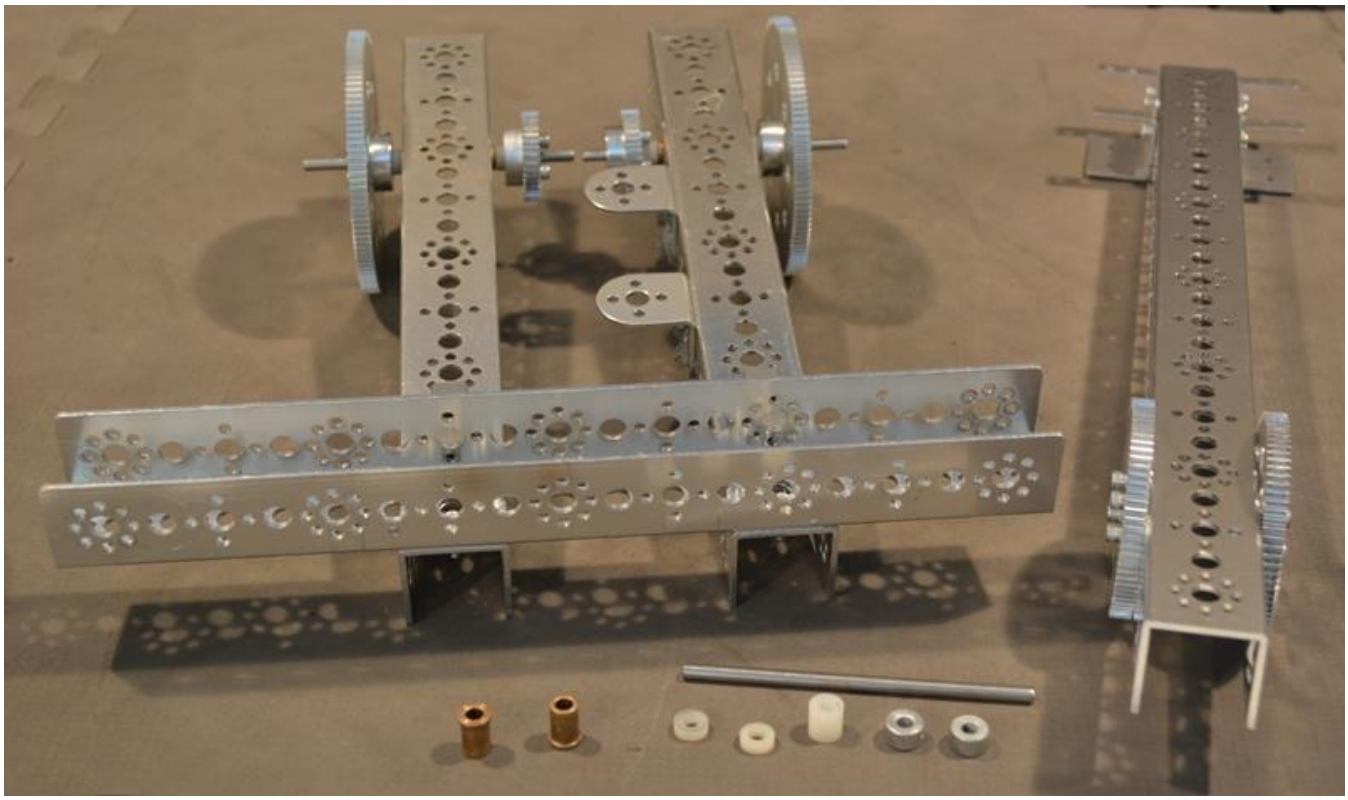
Note the sequence from top to bottom is 40-tooth gear, axle hub, spacer, bronze bushing, side of channel, bronze bushing, second bronze bushing, spacer, axle hub, and 120-tooth gear. The sequence is important, so these gears align with the LEGO motor gears.

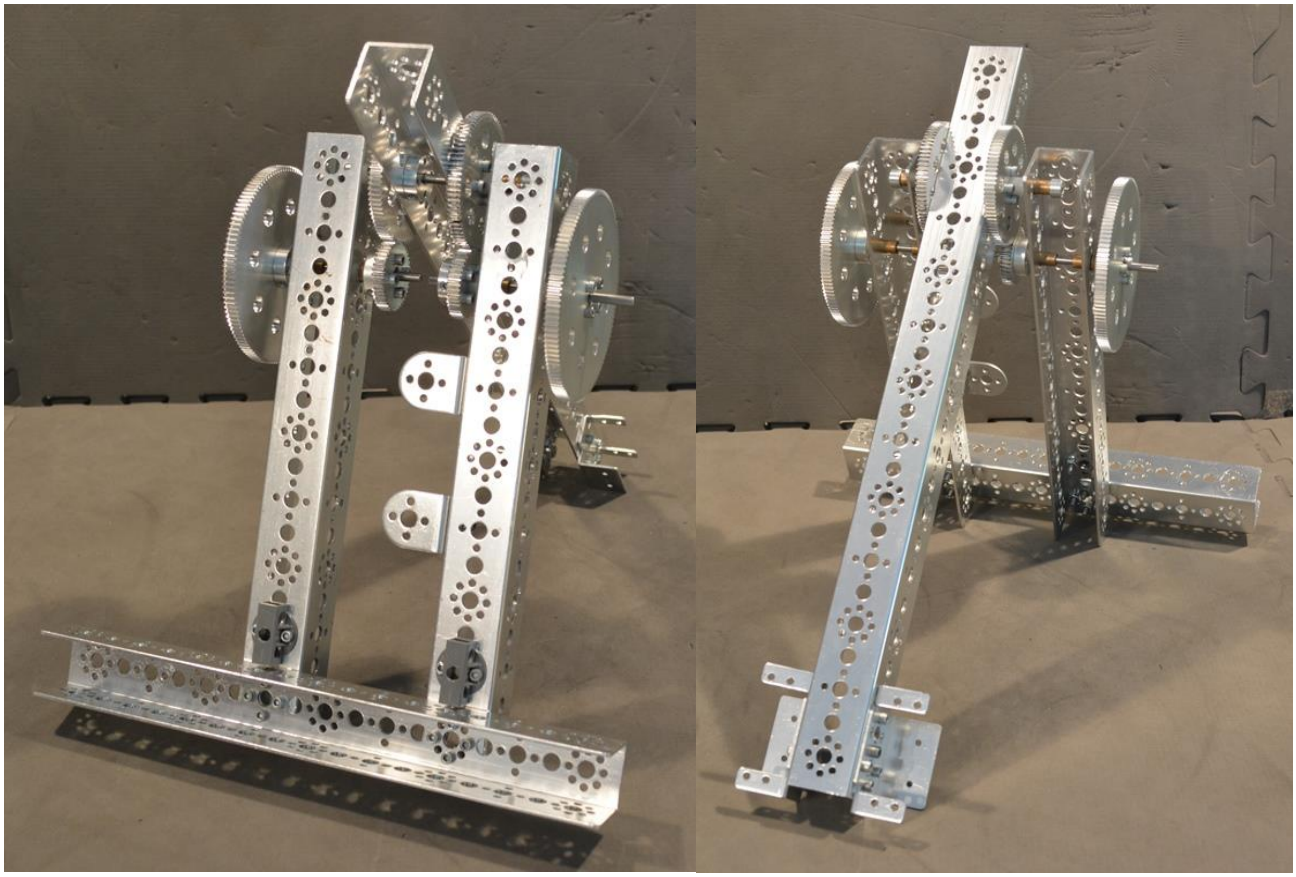
Step 19: assemblies from step 17 and 18, 288mm channel, 5/16” socket head cap screws (8), kep nuts (8)

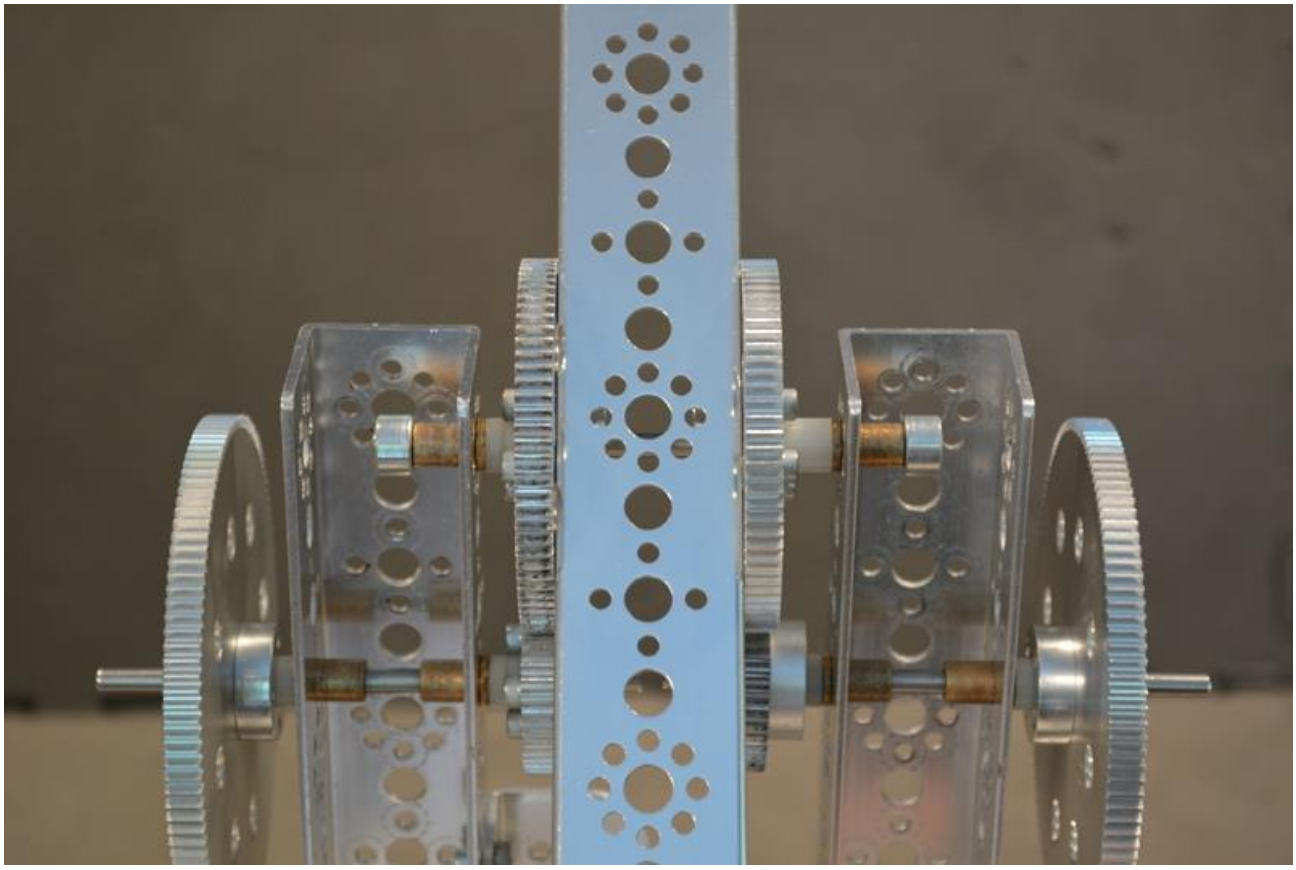




Step 20: assemblies from steps 14 and 19, 100mm axle (1), bronze bushing (2), 3/8” axle spacer (1), 1/8” axle spacer (2), axle set collar (2)

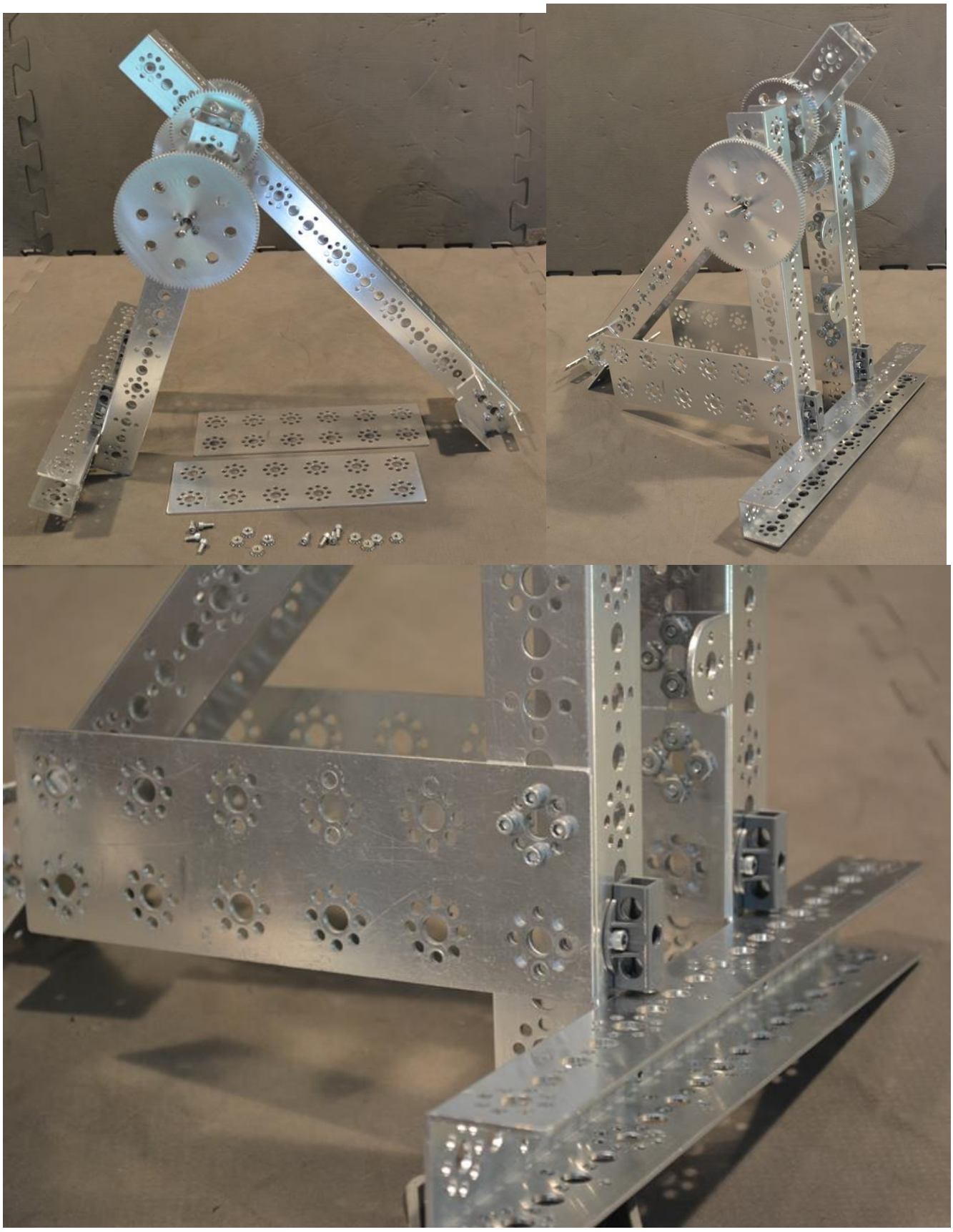






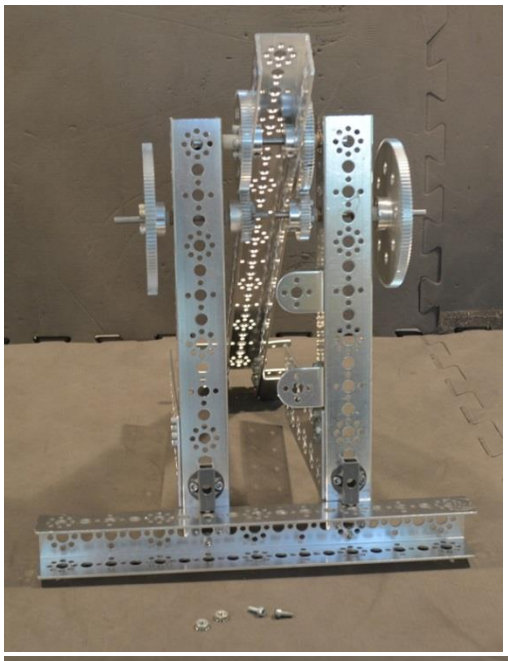
These three images show different views of the same assembly.

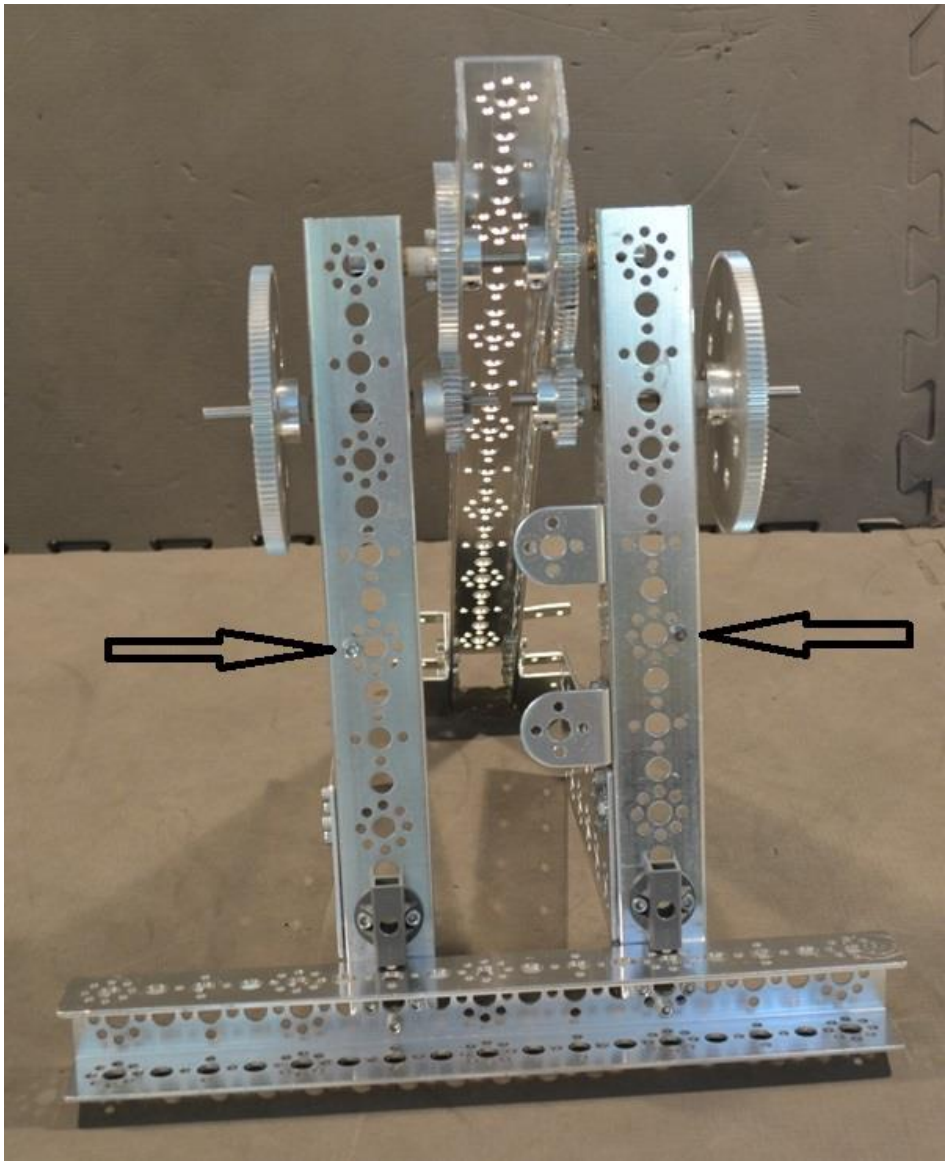
Step 21: assembly from step 20, 100mm axle (1), flat building plate (2), 1/2” socket head cap screw (8), kep nut (8)



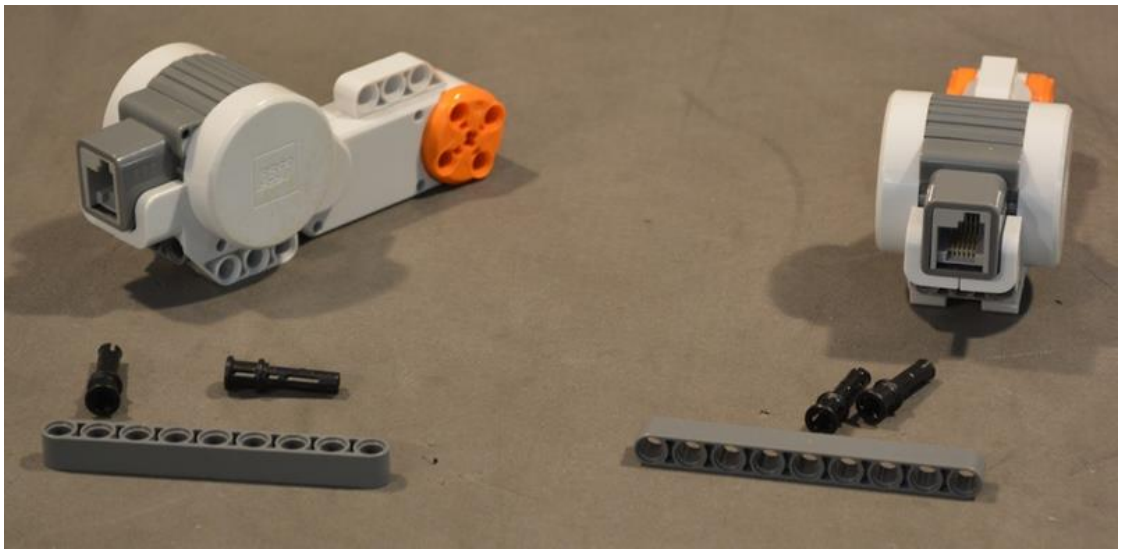
The top right image is the completed assembly. The bottom image shows a closer view of the completed assembly.

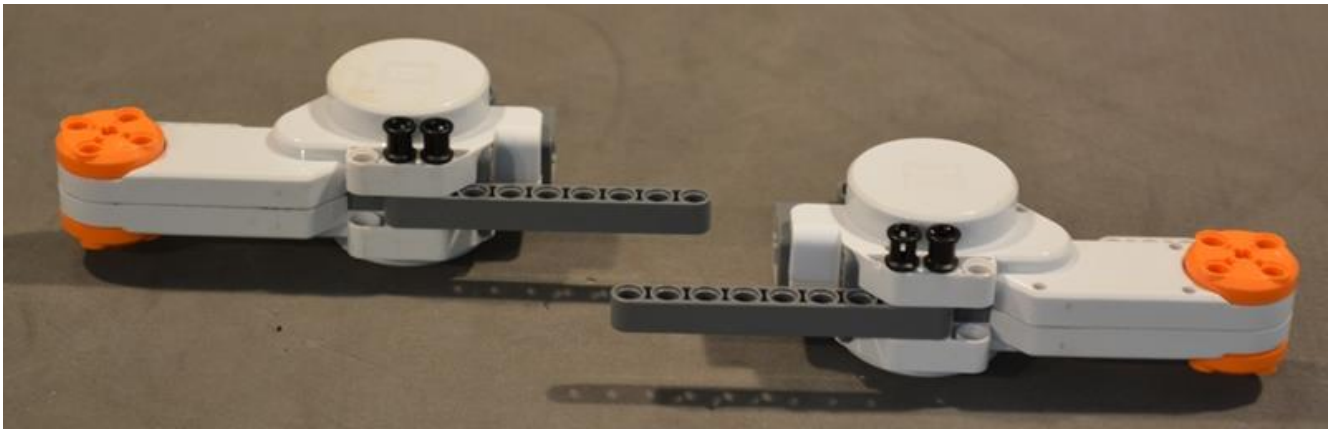
Step 22: 1/2” socket head cap screw (2), kep nut (2)



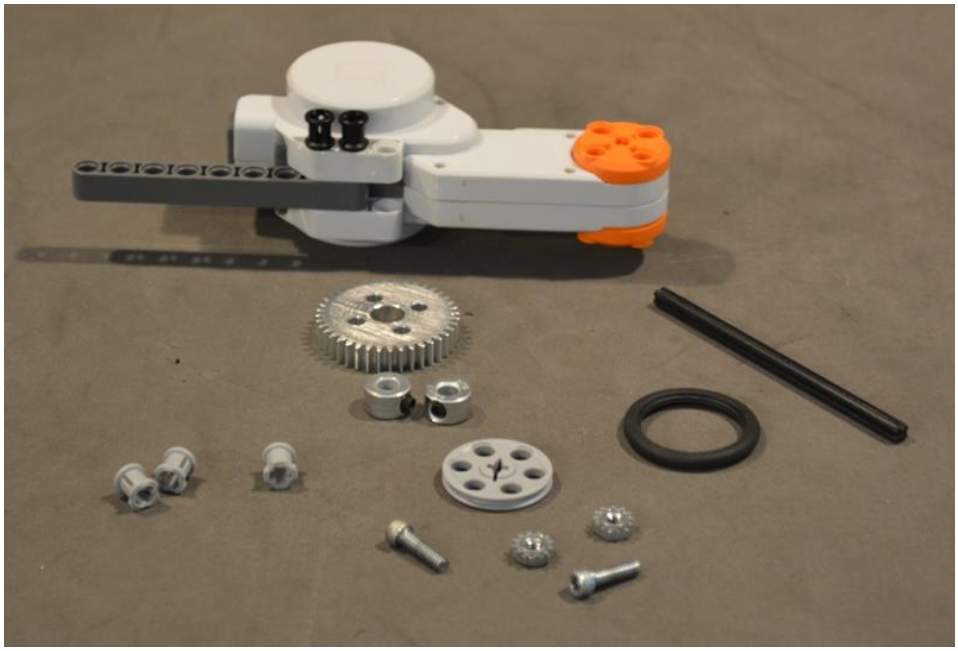


Step 23: NXT interactive servo motor (2), 9-module dark grey beam (2), black connector peg with bushing (2)

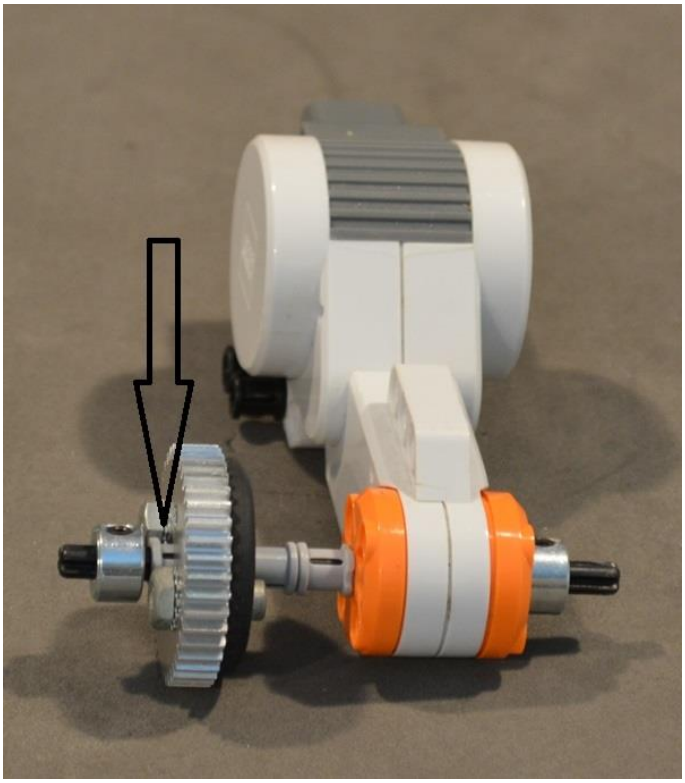




Step 24: one of the assemblies from step 23, 40-tooth gear, axle set collar (2), ½" socket head cap screw, key nut (2), grey bushing (3), 24X4 grey hub (1), 30 4X4 black tire (1), 10-module black axle (1)

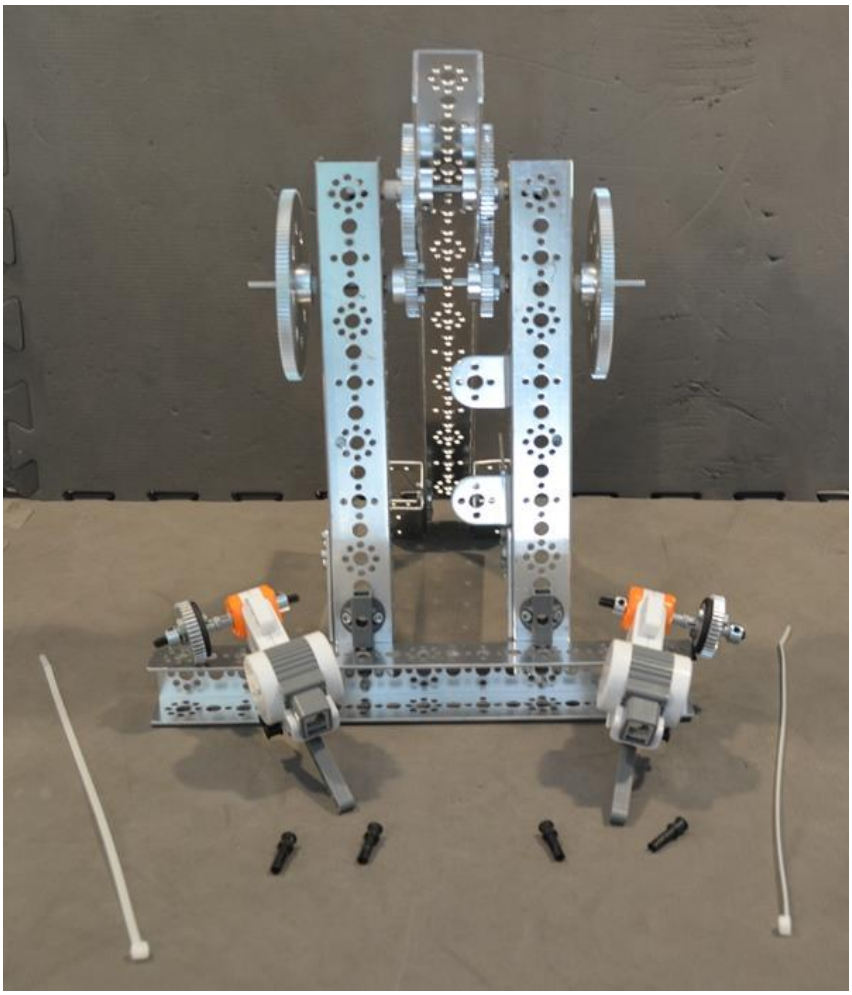


Two bushings are to the right of the hub (wheel) and one (marked by the arrow) is partially inside the gear.

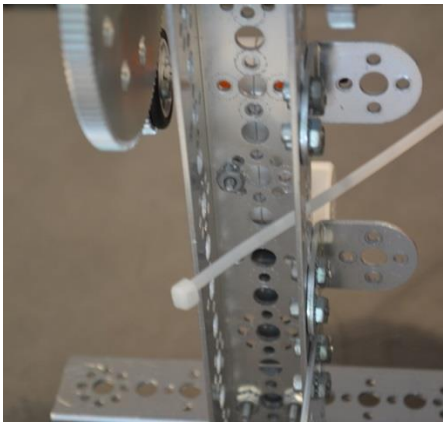


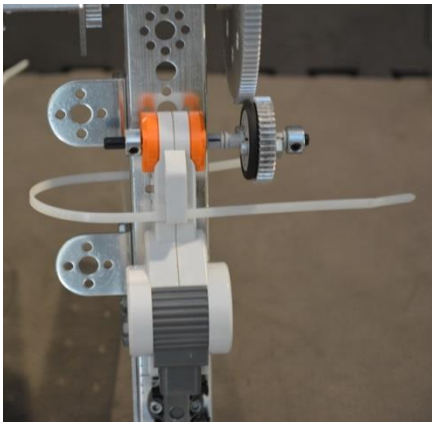
Step 25: the other assembly from step 23, 40-tooth gear, axle set collar (2), ½" socket head cap screw, key nut (2), grey bushing (3), 24X4 grey hub (1), 30 4X4 black tire (1), 10-module black axle (1) ------ The image of parts is the same as step 24. Both completed assemblies are shown below.

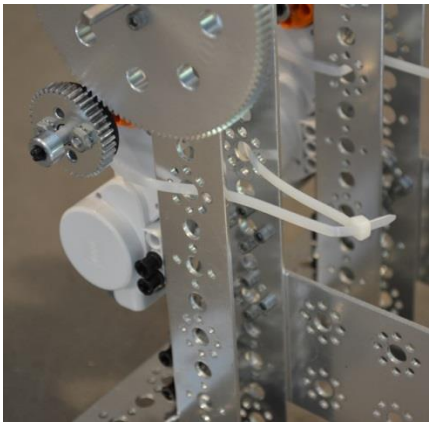
Step 26: the assemblies from step 22, 24 and 25, black connector peg with bushing (4), 11” flexible nylon cable tie (2)

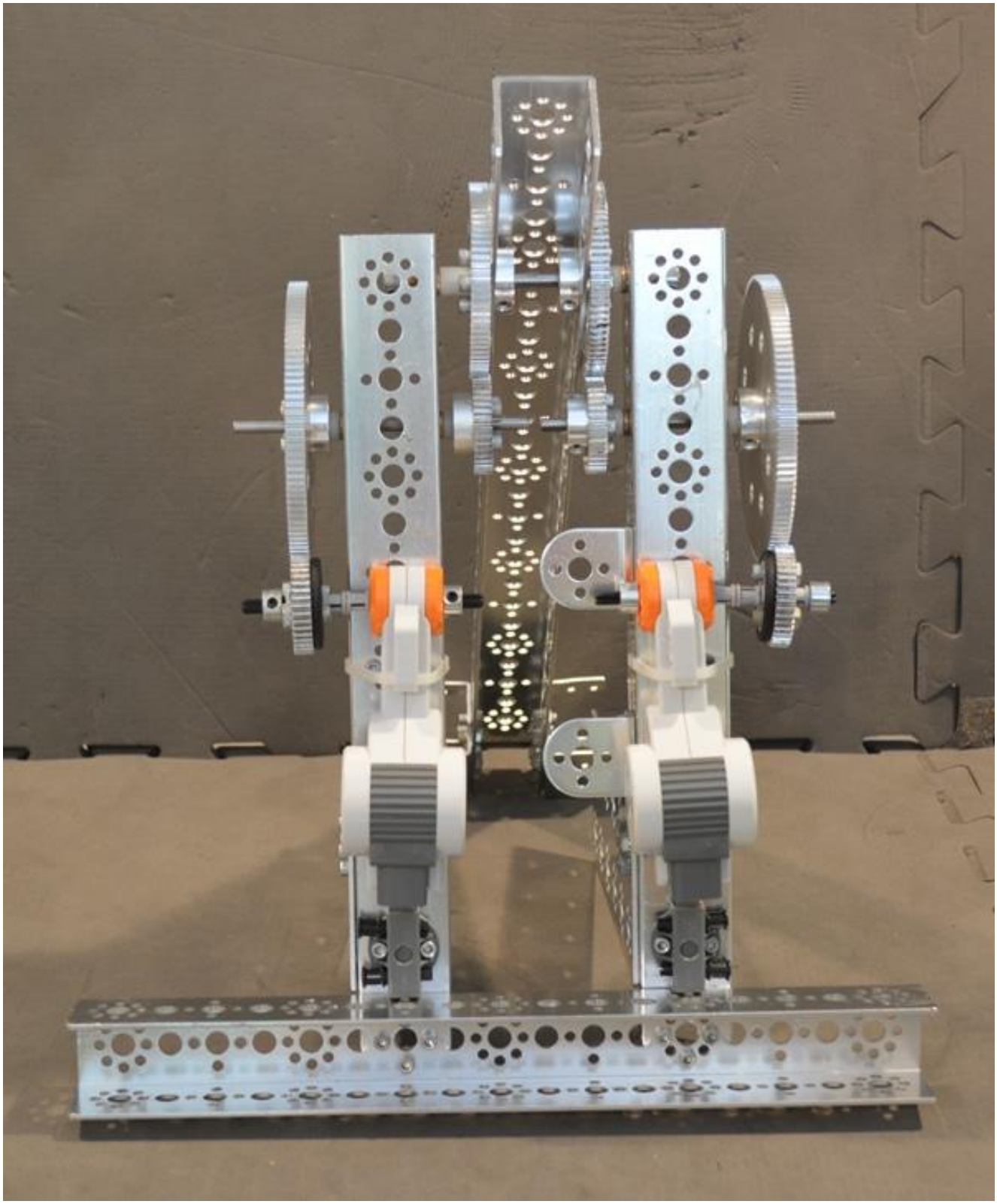


Secure the beams attached to the motors to the hard point connector. Match the 40-tooth gears on the motors to the 120-tooth gear on the tower (as show in the completed assembly on the next page). From the side opposite the LEGO motors, insert the pointed end of the zip into the large hole (left). Insert the pointed end of the zip tie through the lower hole in the motor (middle). Insert the pointed end through the large hole in the channel and secure it through the head of the zip tie (right). Pull the zip tie tight and cut the pointed end close to the head.

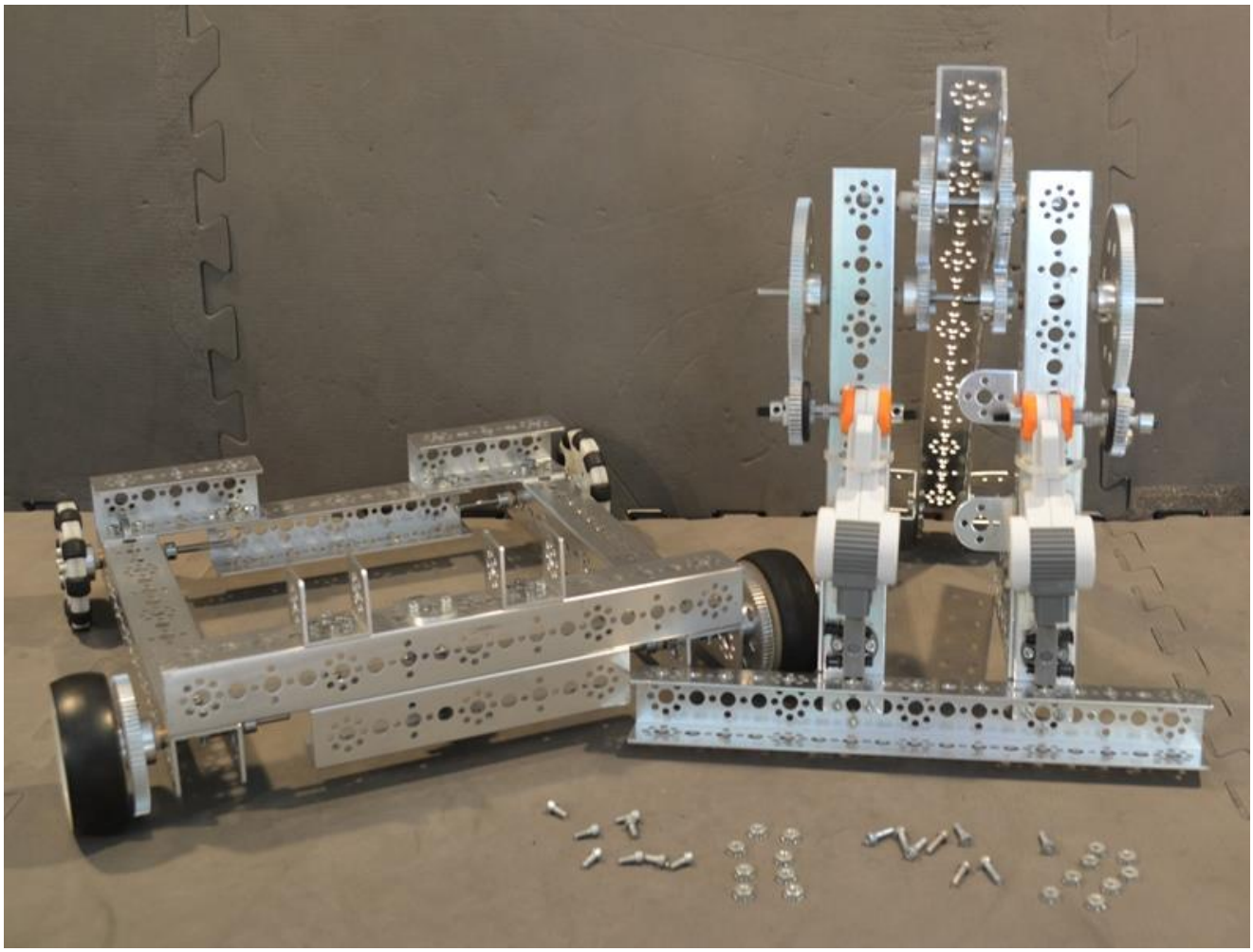


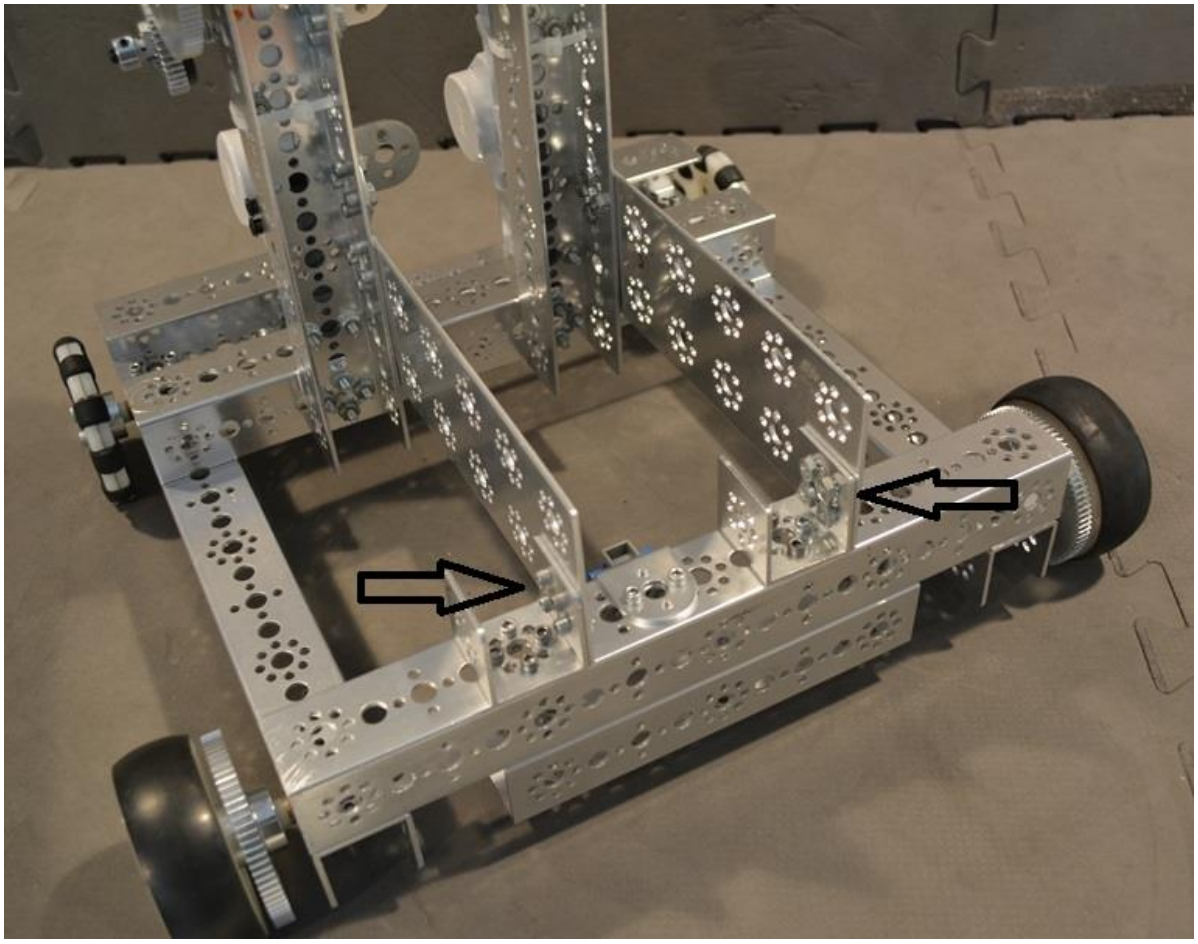




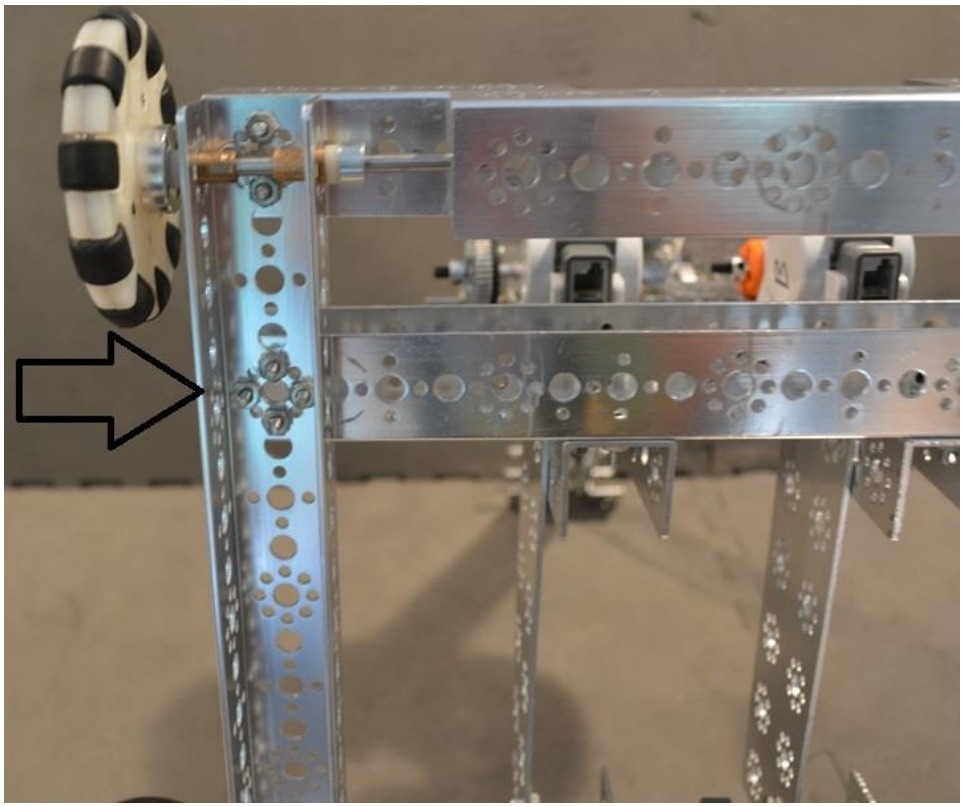


Step 27: assemblies from step 12 and 26, ½" socket head cap screw (8), 5/16” socket heat cap screw (8), kep nut (16)

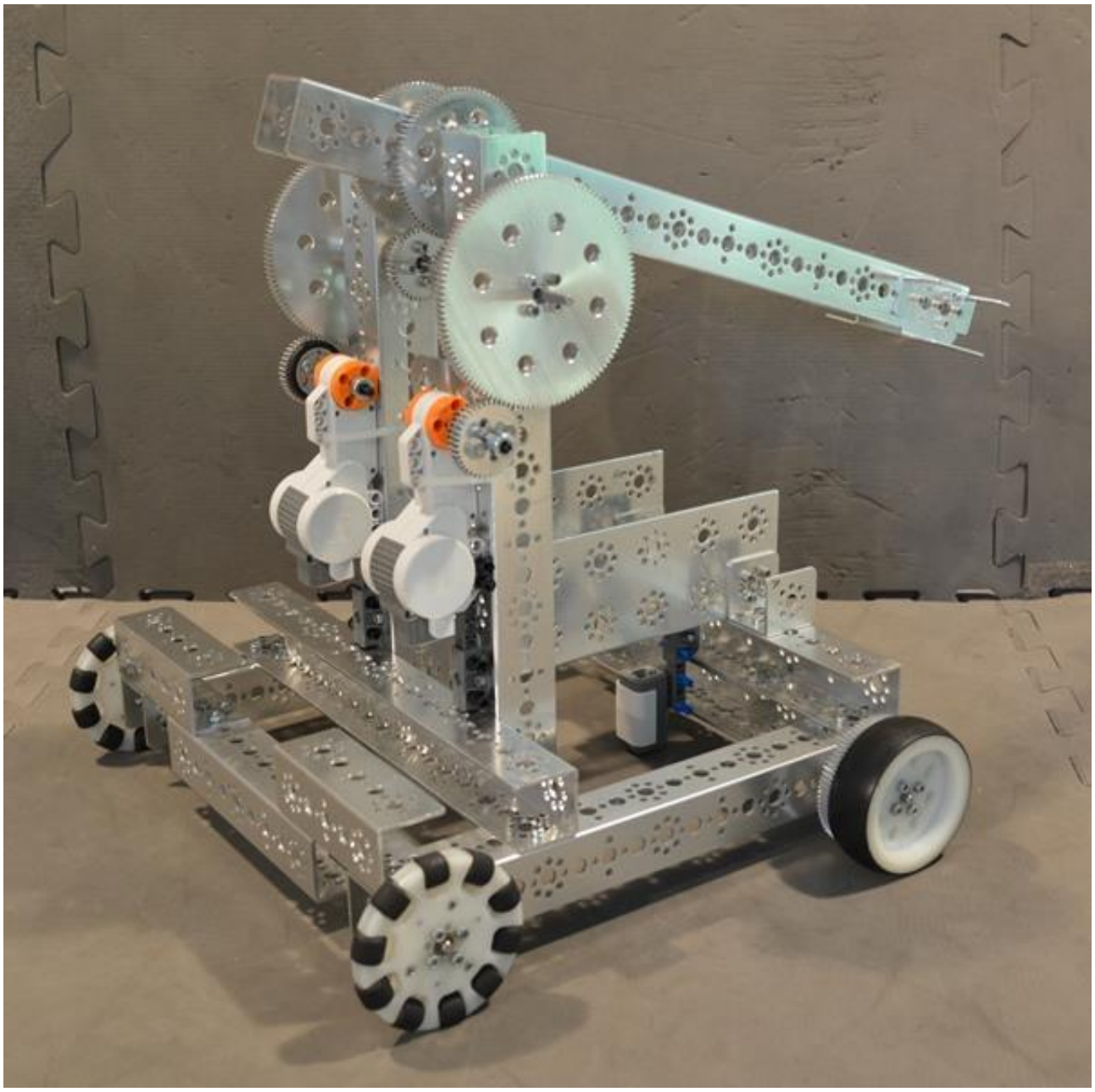




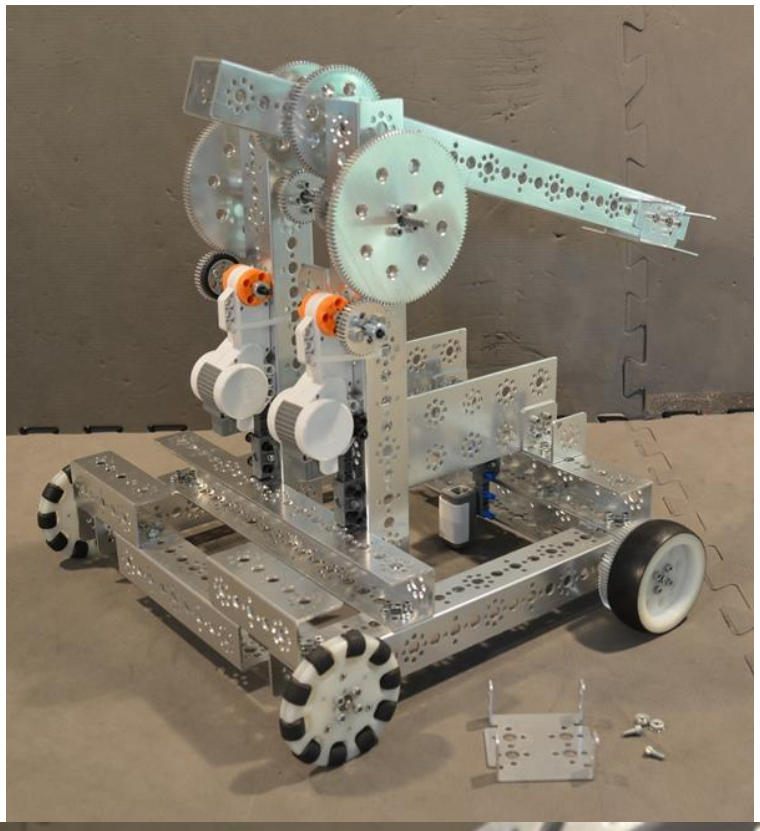
The arrows in the image above mark the location of the 5/16” screws. The ones below show where of the 1/2" screws are secured. The other end of the horizontal beam will have the other 4.

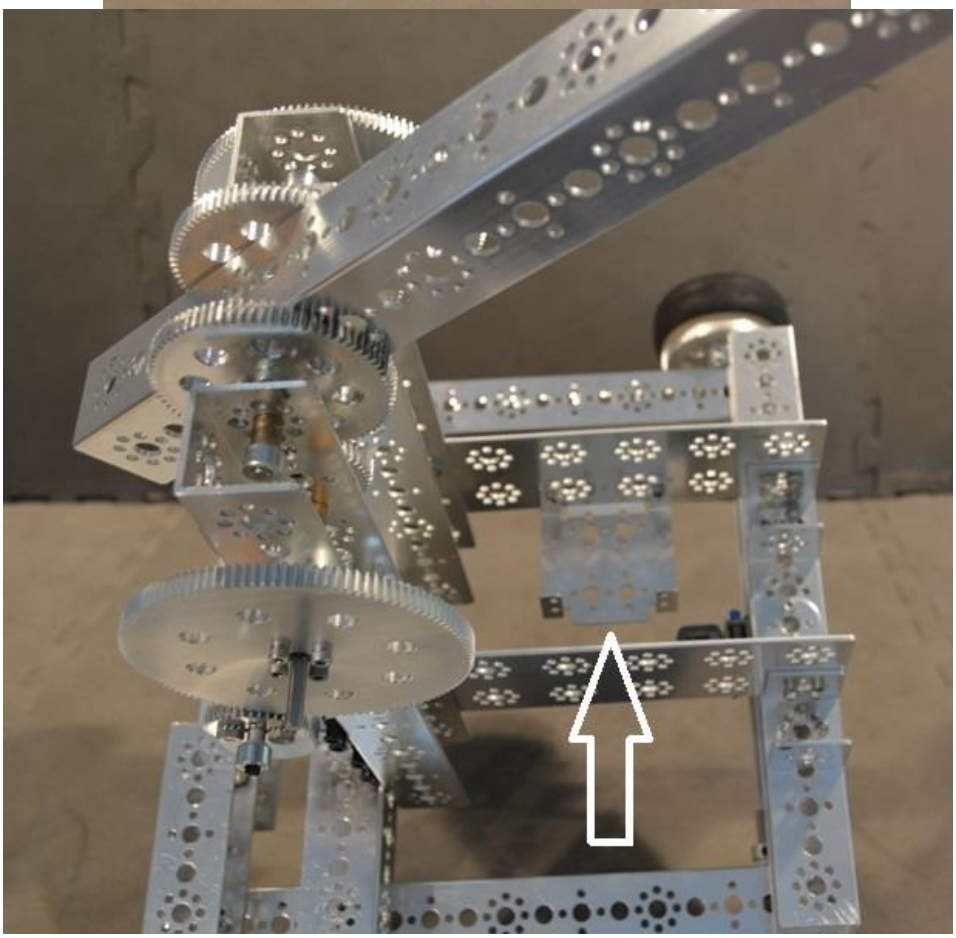


This image shows another view of the completed assembly.



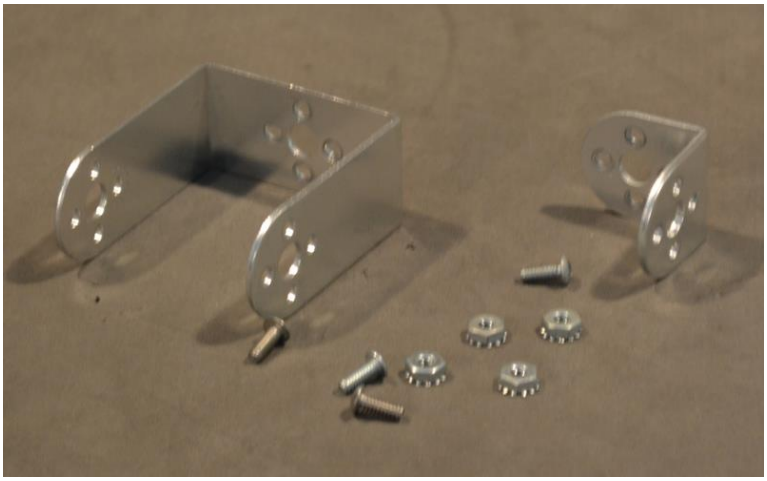
Step 28: assembly from step 27, dual-servo motor bracket, 5/16” socket head cap screw (2), kep nut (2)

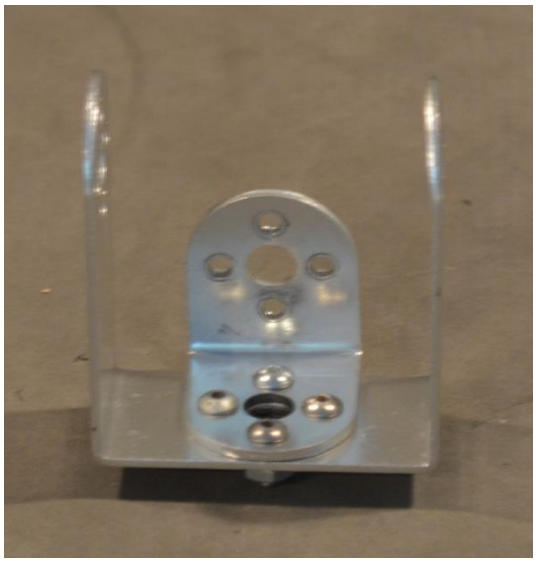




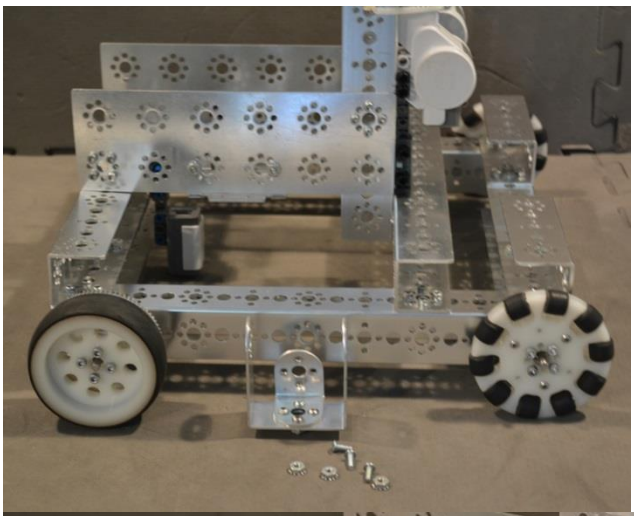
Step 29: the metal portion of the servo joint pivot bracket (1), L bracket (1), 3/8” button head cap screw (4), kep nut (4)

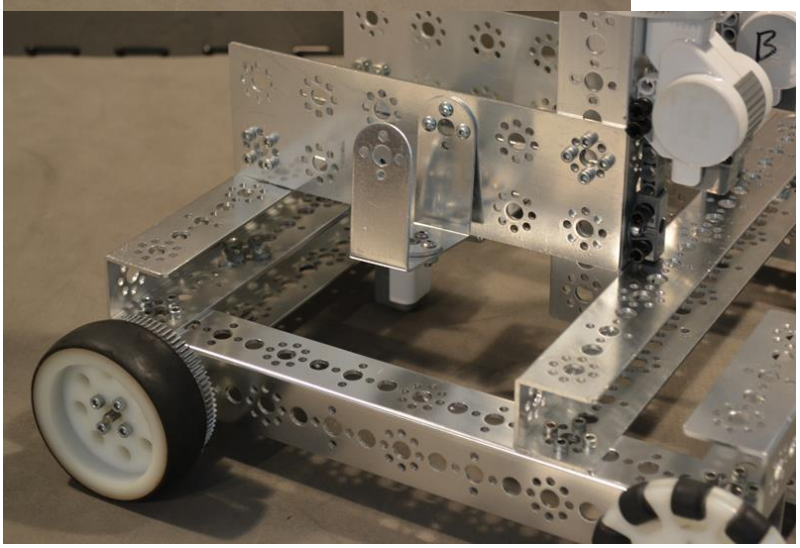
It is important to use button head screws for this step due to space limitations.



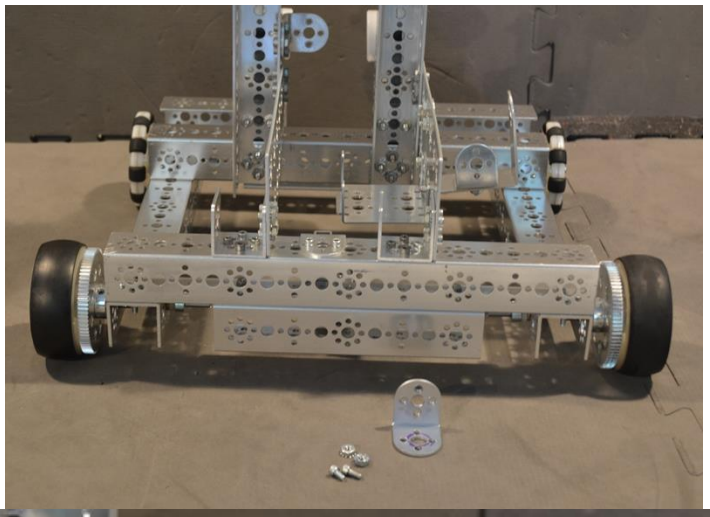


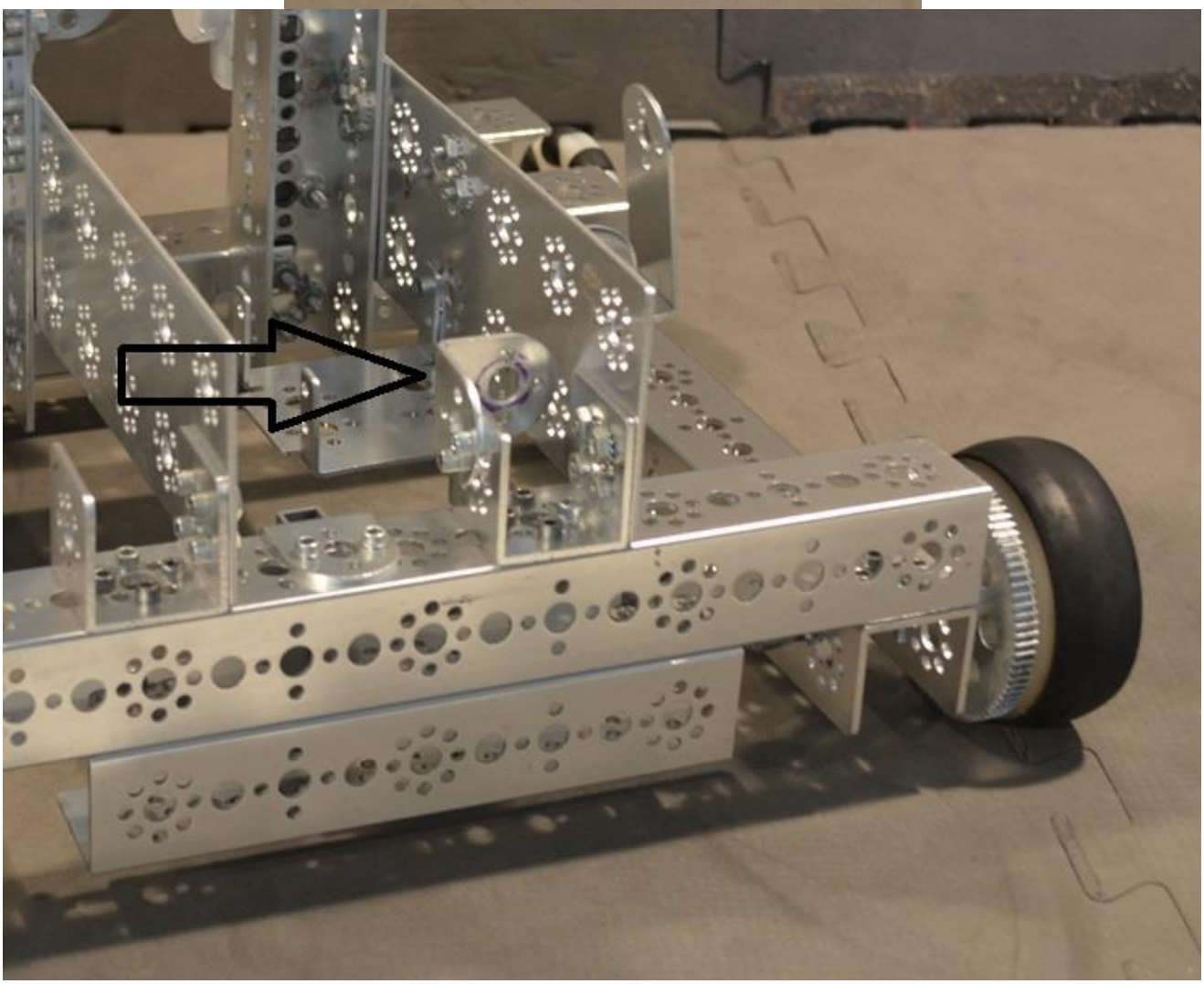
Step 30: assembly from step 28, 3/8” button head cap screw (3), kep nut (3)



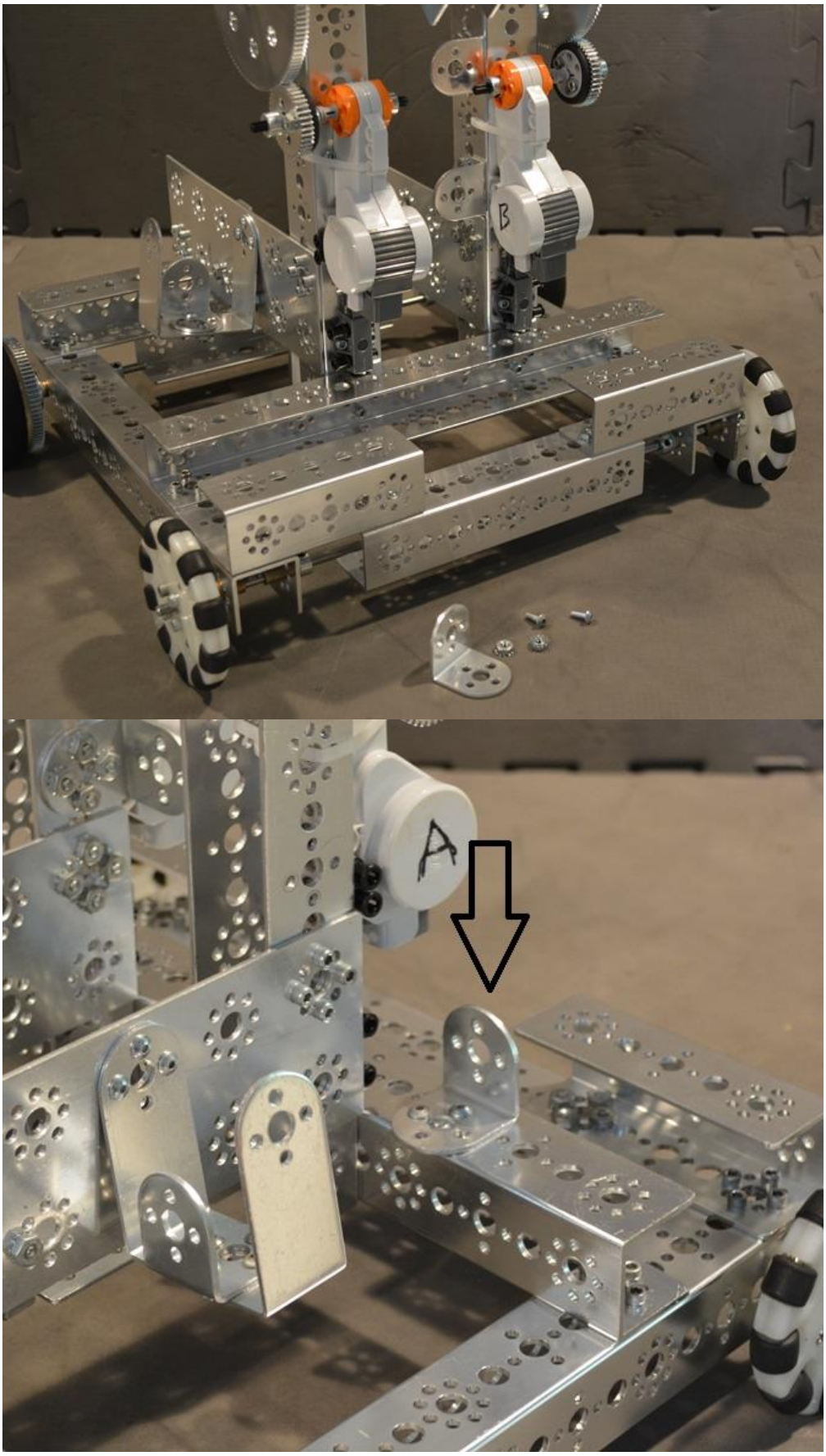


Step 31: assembly from step 30, L bracket (1) 5/16” socket head cap screws (2), kep nut (2)

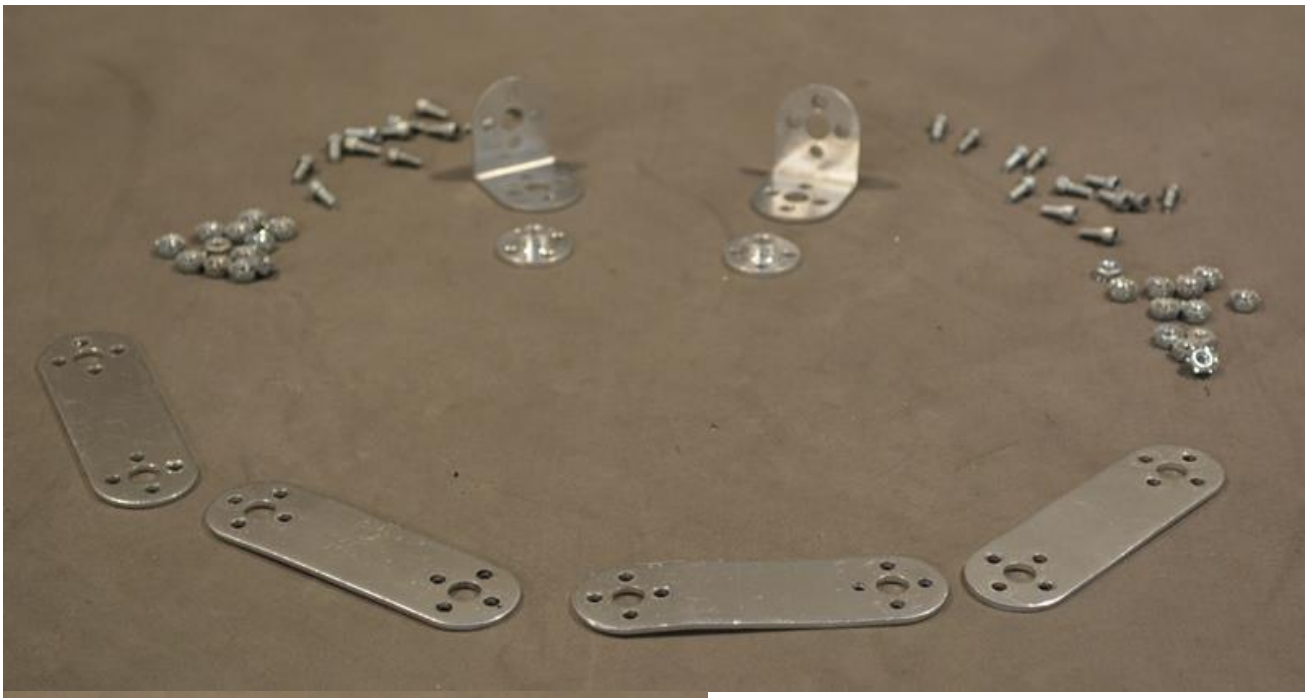




Step 32: assembly from step 31, L bracket (1), 3/8” button head cap screw (2), kep nut (2)

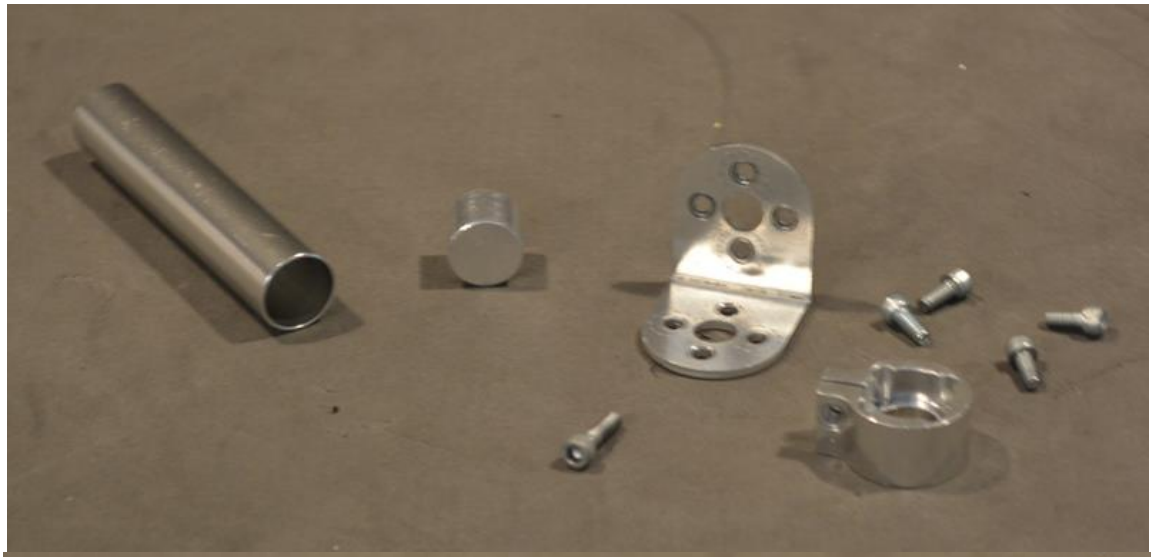


Step 33: flat bracket (4), L bracket (2), servo horn (from the servo motor packet) (2), 5/16” socket head cap screw (24), kep nuts (24). These parts will make two assemblies.

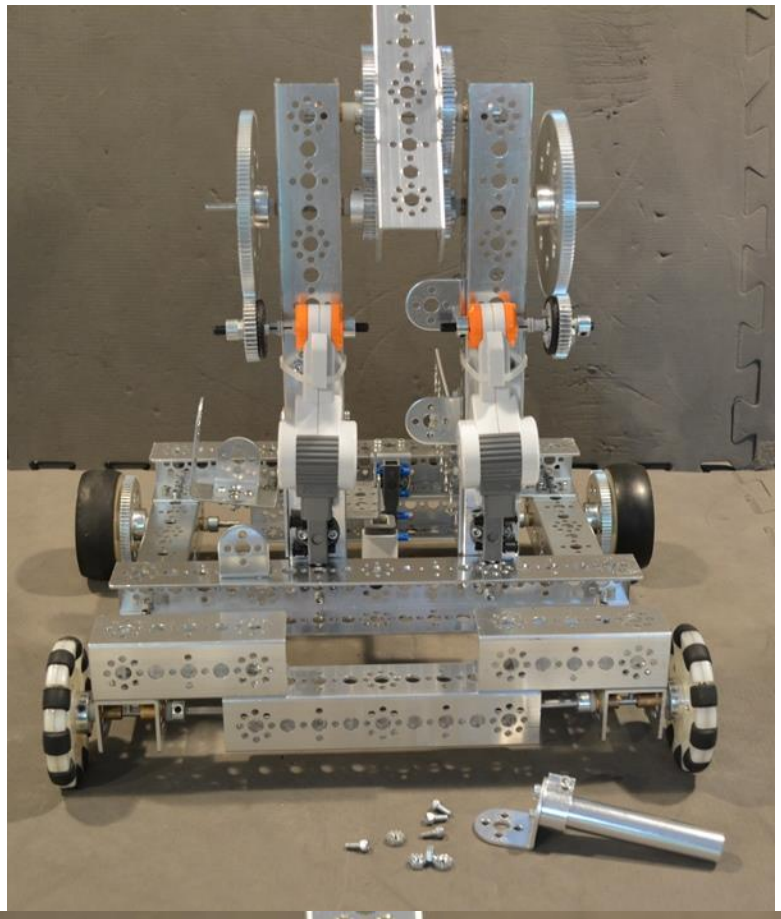


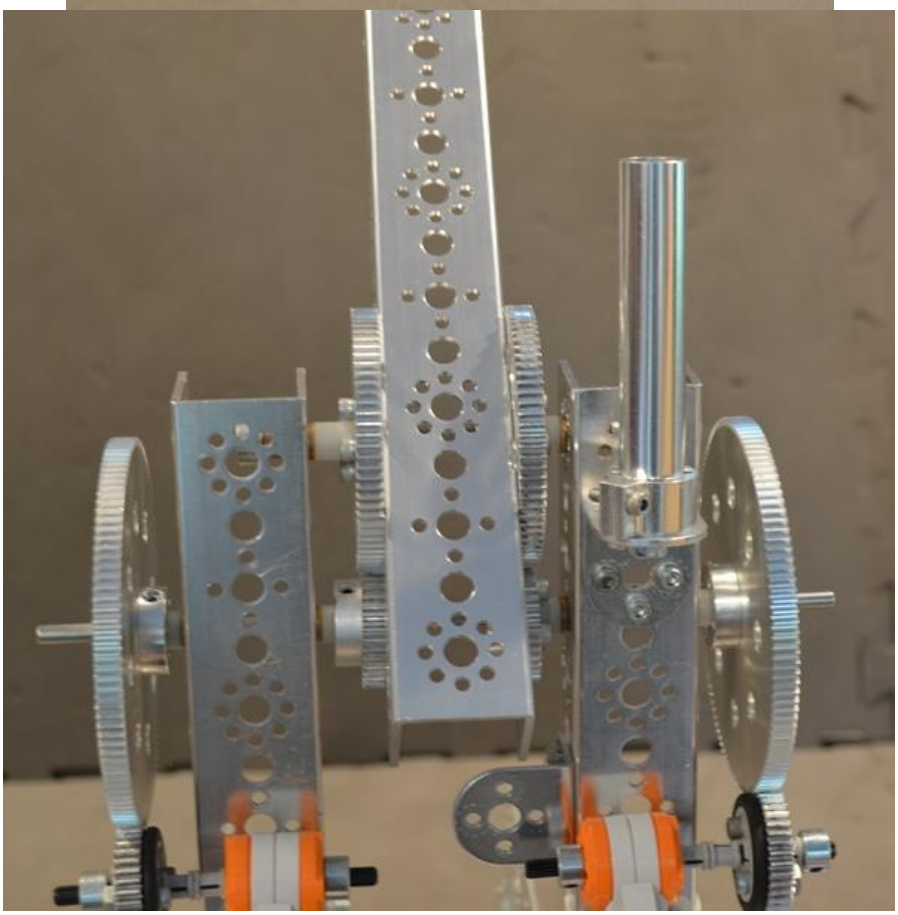
The addition of L brackets is shown in the above image.

Step 34: 80mm tube (1), tube plug (1), L bracket (1), 1/2” socket head cap screw (1), 5/16” socket head cap screw (4), tube clamp (1)

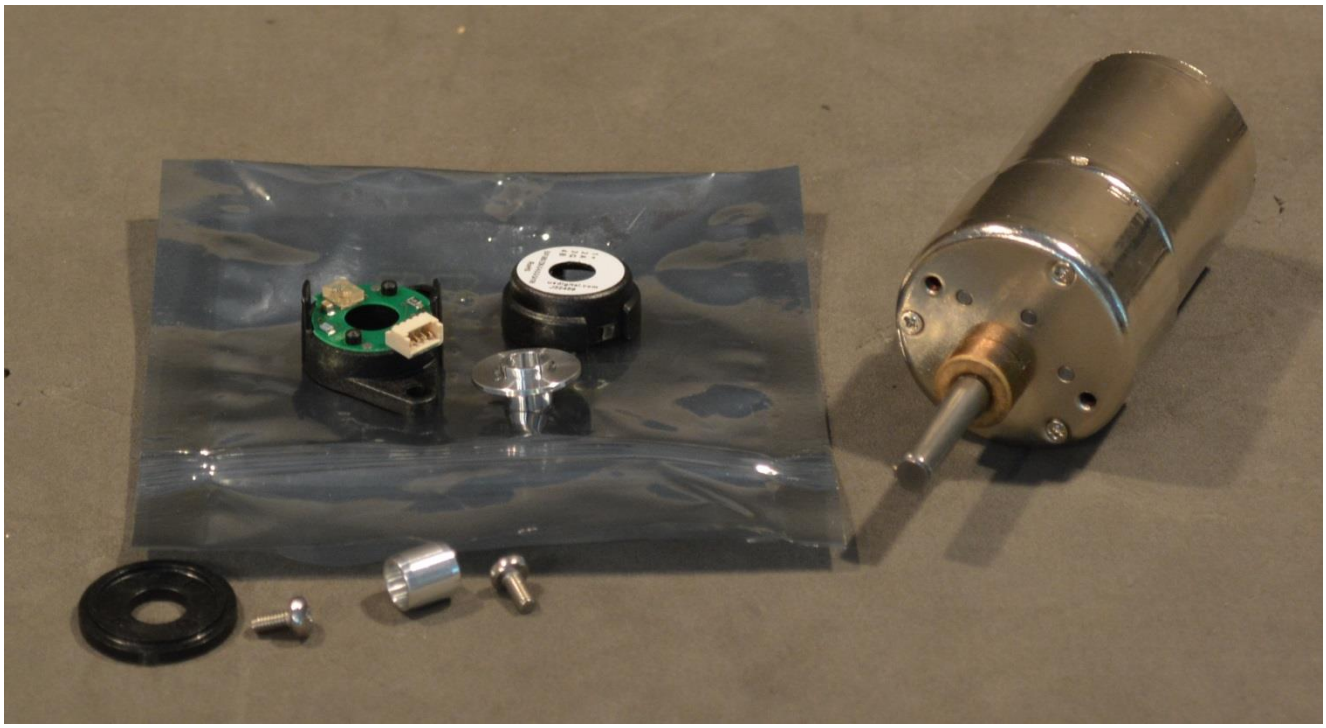


Step 35: assemblies from step 32 and 34, 5/16” socket head cap screw (4), kep nut (4)

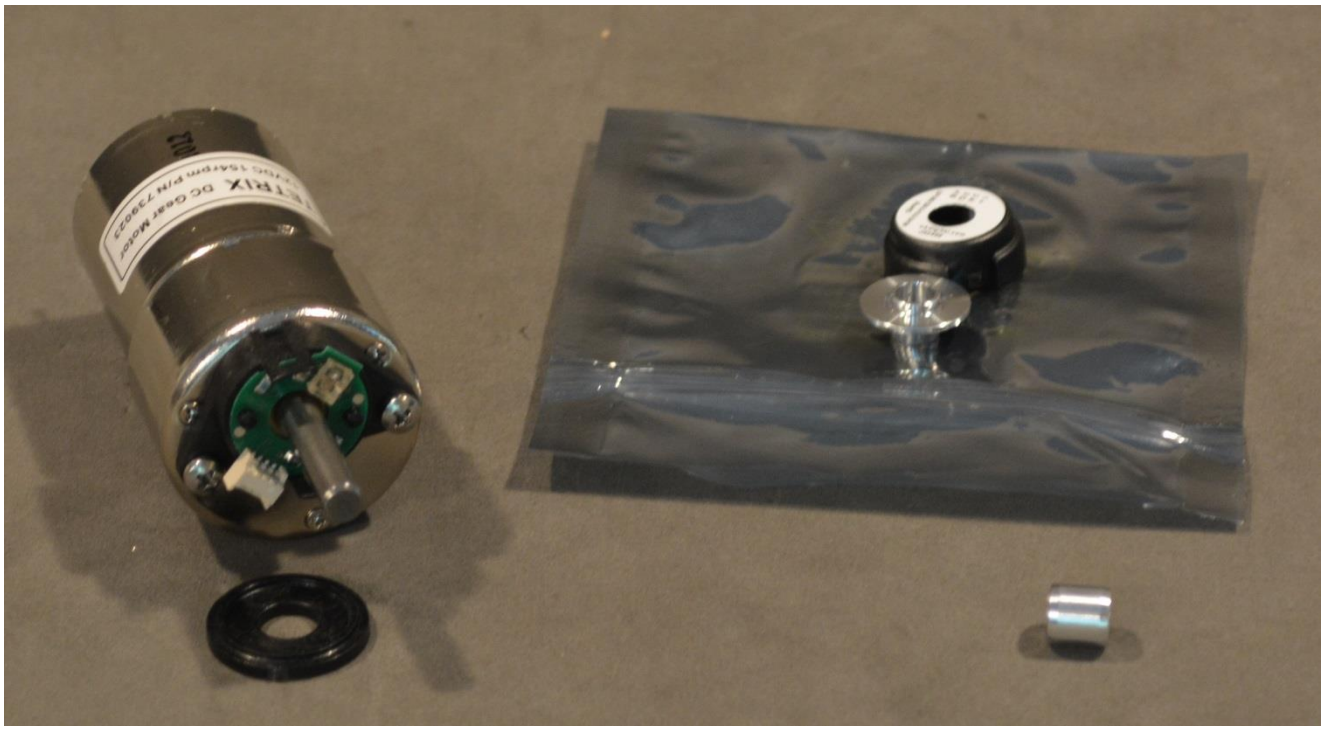




Step 36 (make two): DC drive motor (1), motor encoder pack (1), the provided wires will be used in a later step.



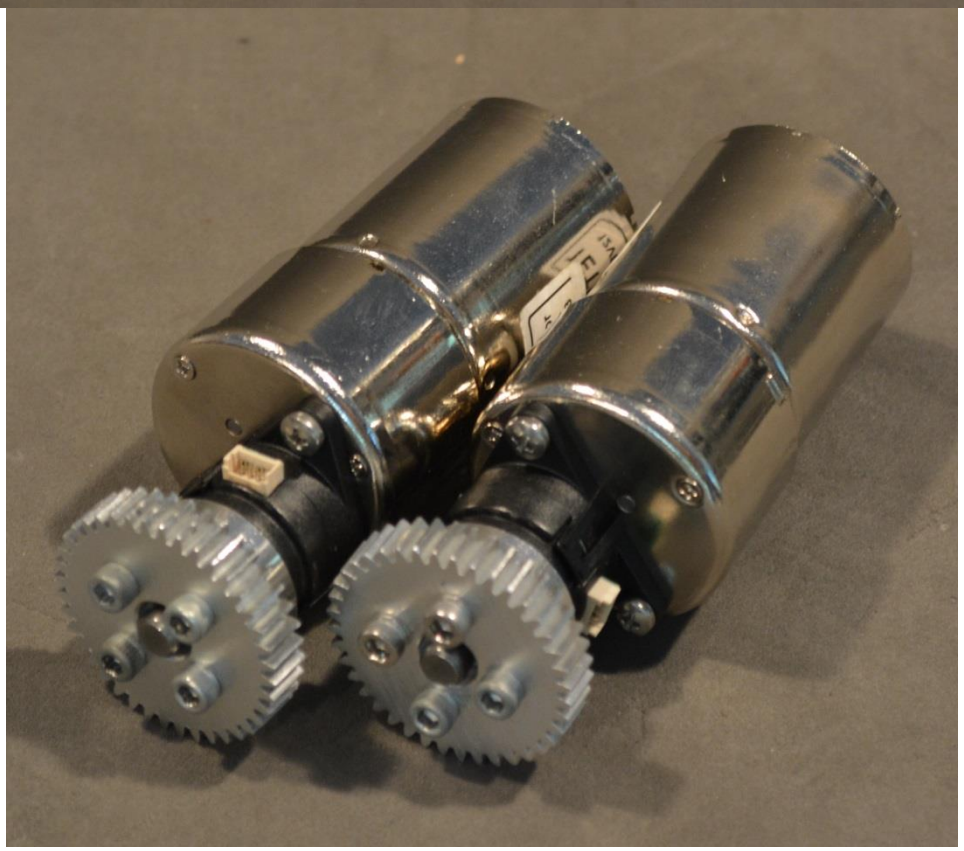
Use provided screws to attach encoder to motor.



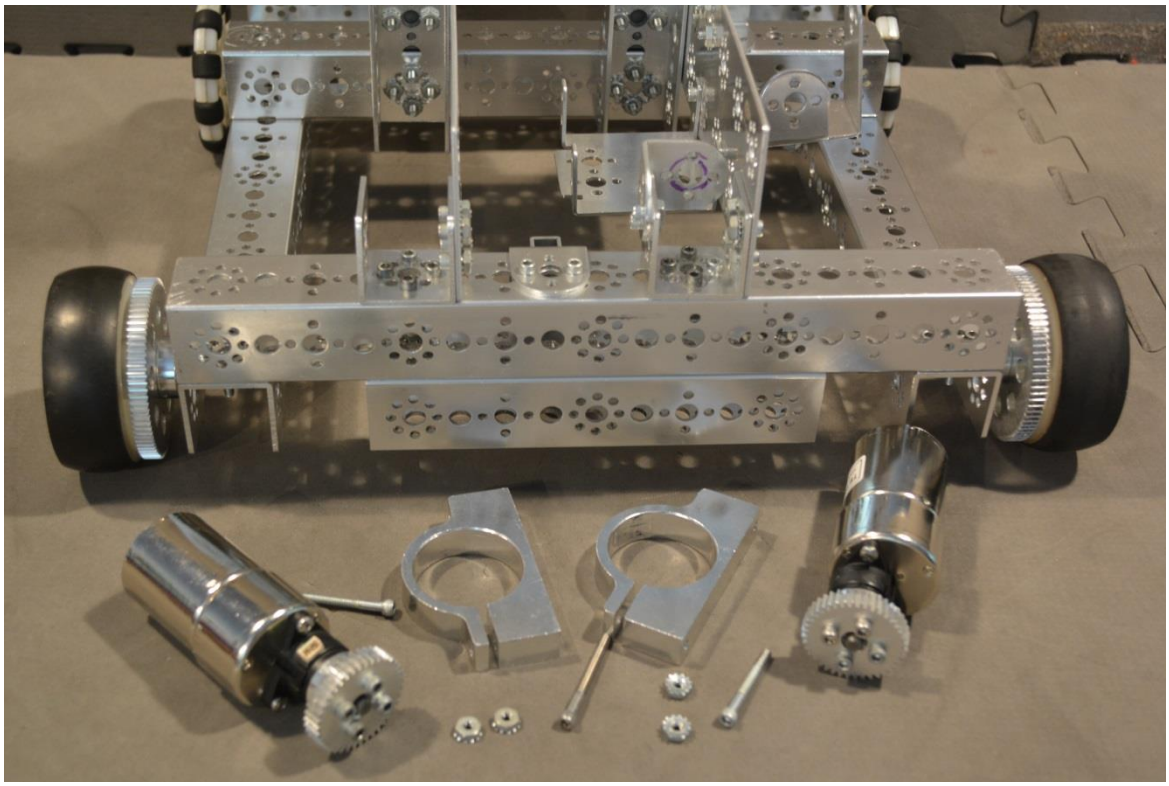
Use provided spacer tool to push disc onto motor shaft (left image). The right image shows the disc on the shaft after the spacer tool has been removed.

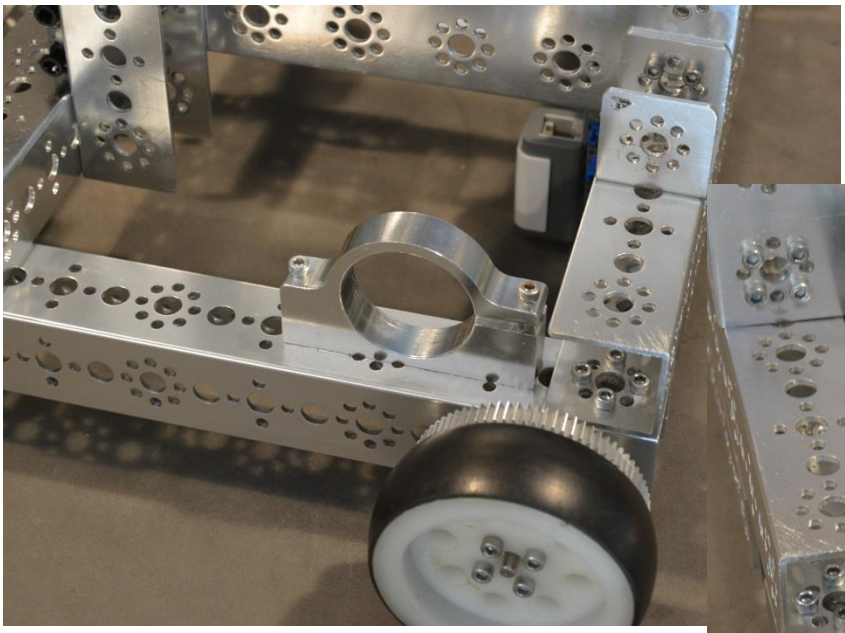
Snap cover onto the encoder.

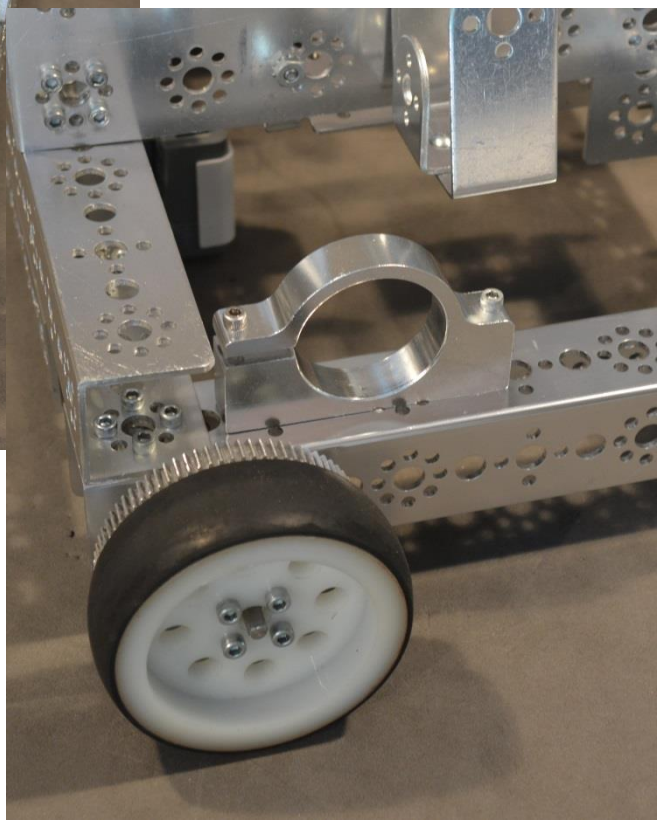
Step 37: assemblies from step 36, 40-tooth gear (2), motor shaft hub (2), 1/2" socket head cap screw (4)



Step 38: assemblies from step 37, motor mount – includes shown socket head cap screws (2), kep nut (4)

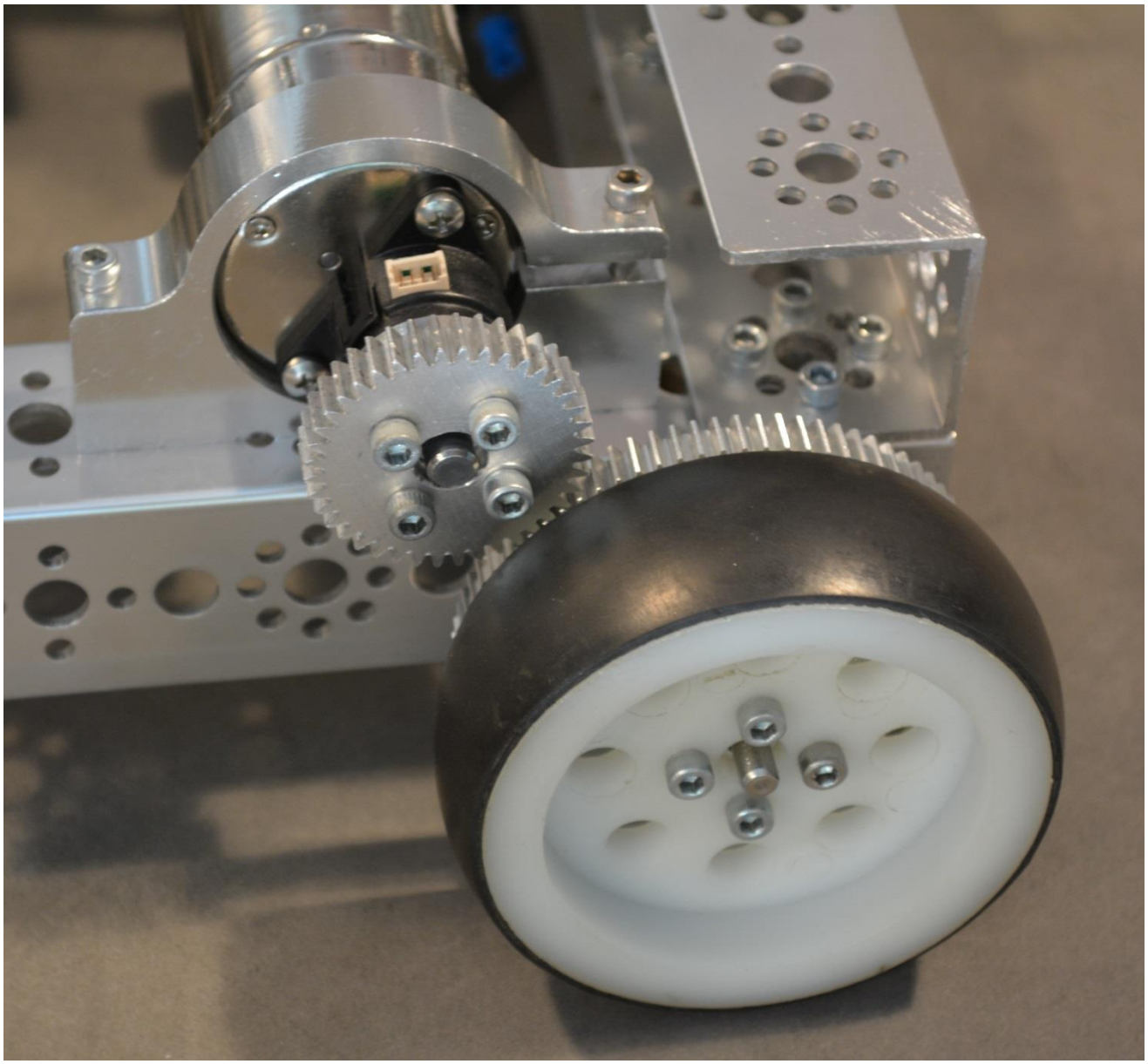




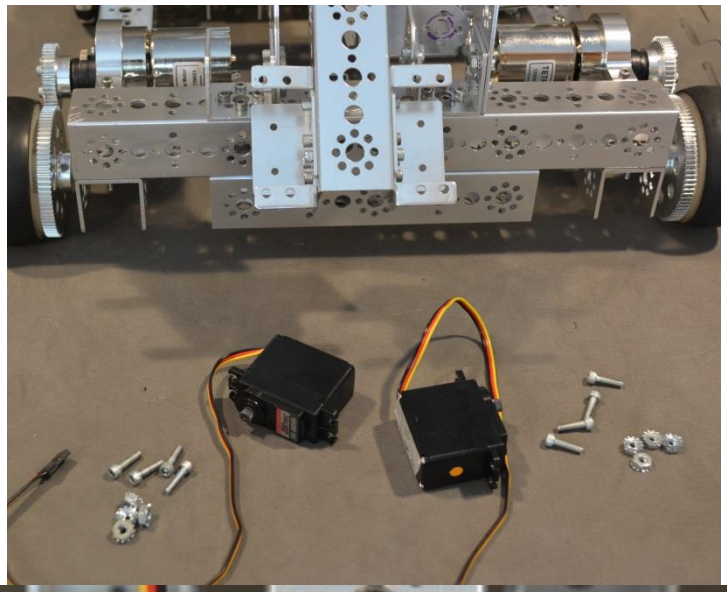


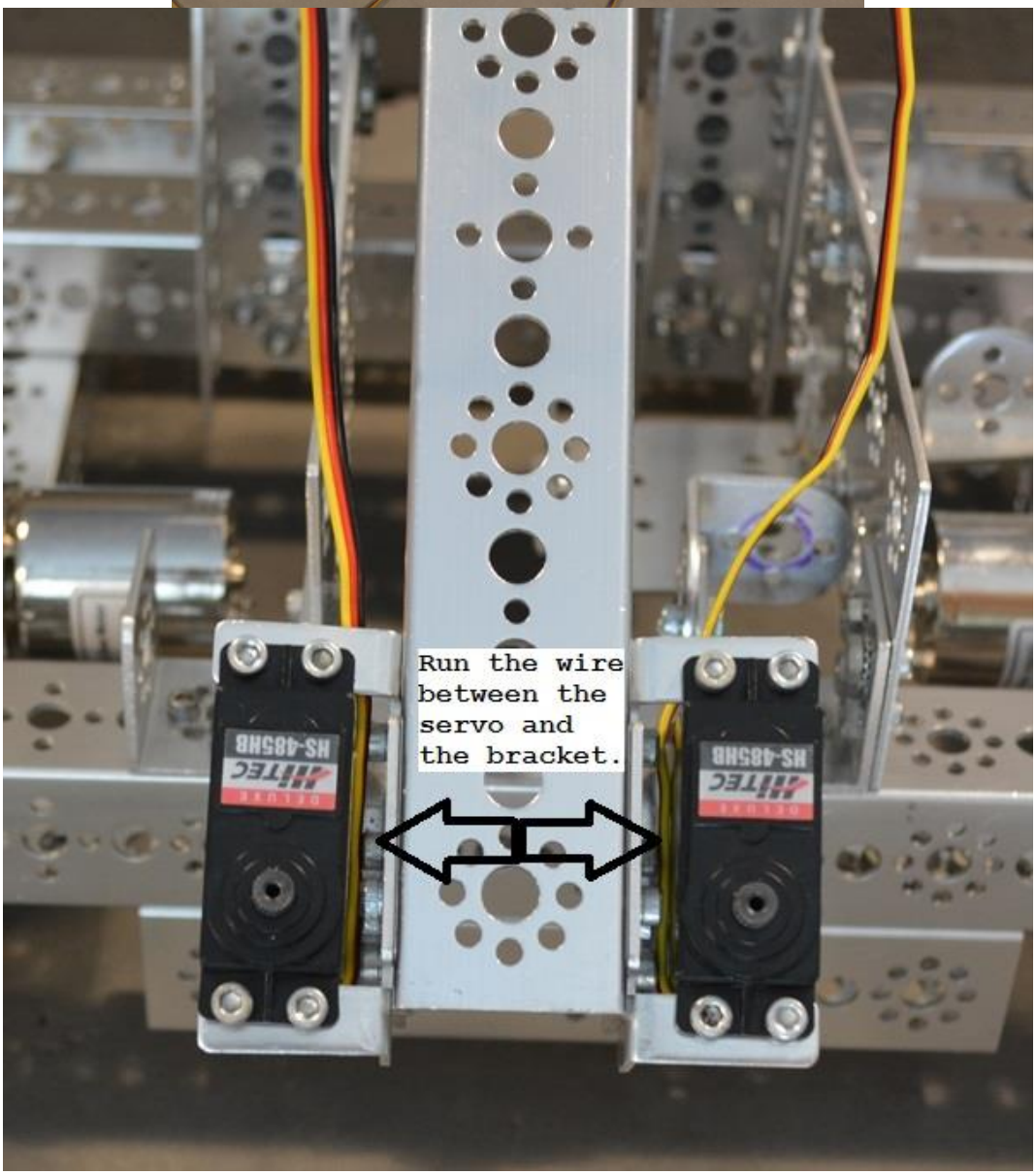
Note that the longer screw of the motor mount faces the front of the robot. The split portion of the motor mount also faces the front of the robot. This is important, so the gear will mesh better with the gear on the wheel.

Rotate motor until the gears mesh. Do not rotate it to the point that the gears bind.

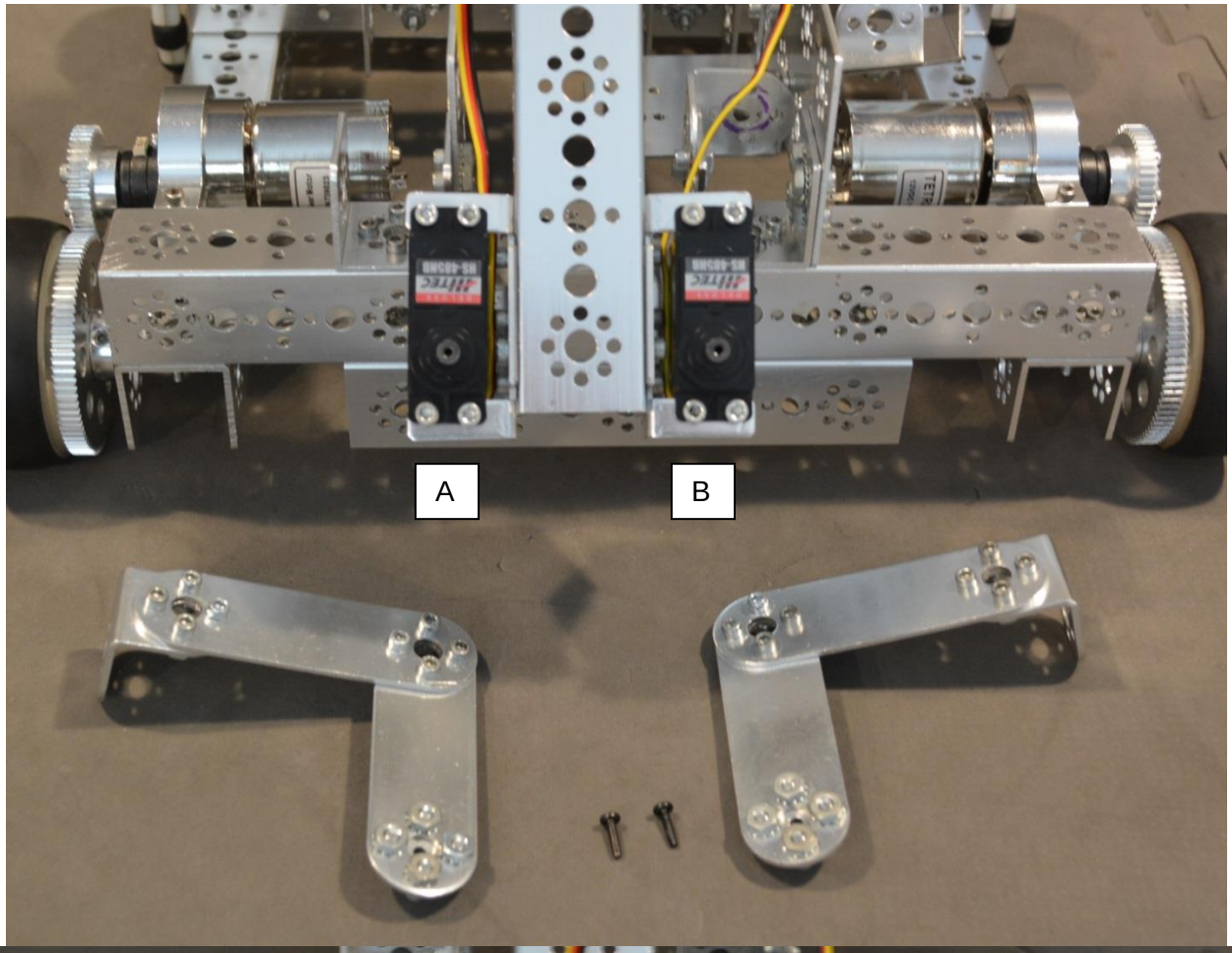


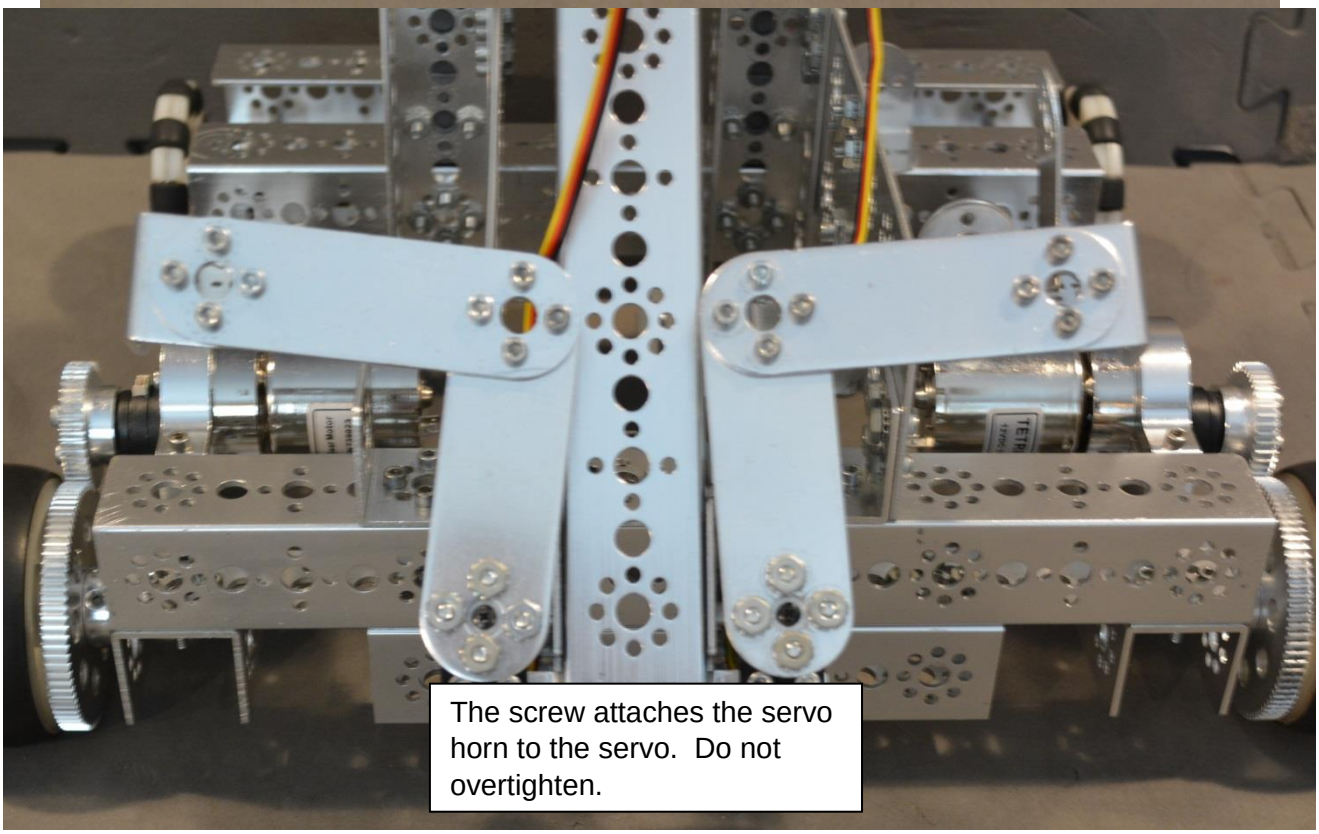
Step 39: assembly from step 38, 180 degree servo motor with horn (2), 1/2" socket head cap screw (8), kep nut (8). Remove plastic servo horn, but retain the screws for the next step (40).



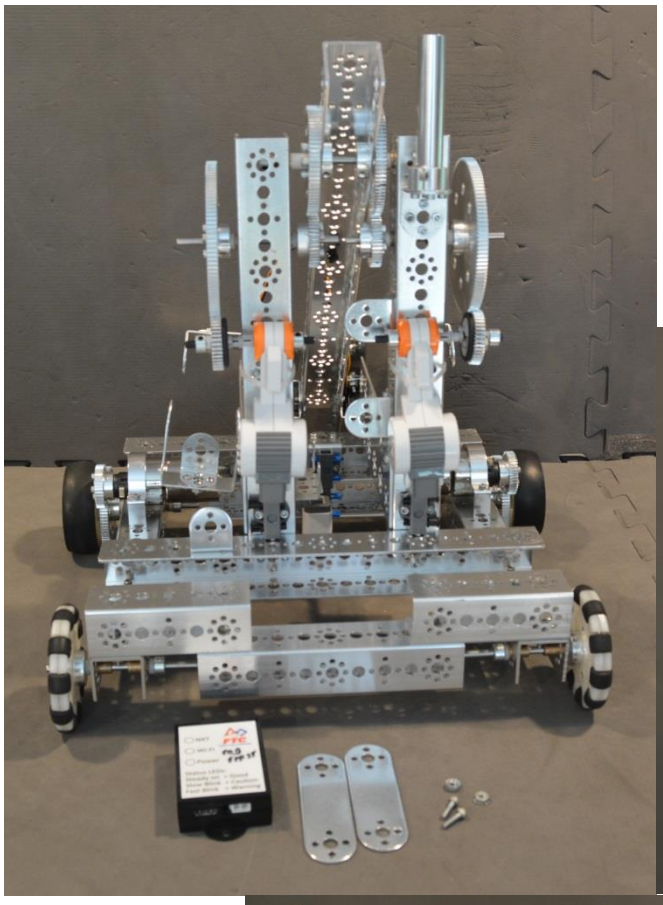


Step 40: assemblies from steps 33 and 39. Use the screws that come with the servo for this step. Use the servo horn on the flat bracket assemblies to rotate servo A fully counter-clockwise and servo B fully clockwise.

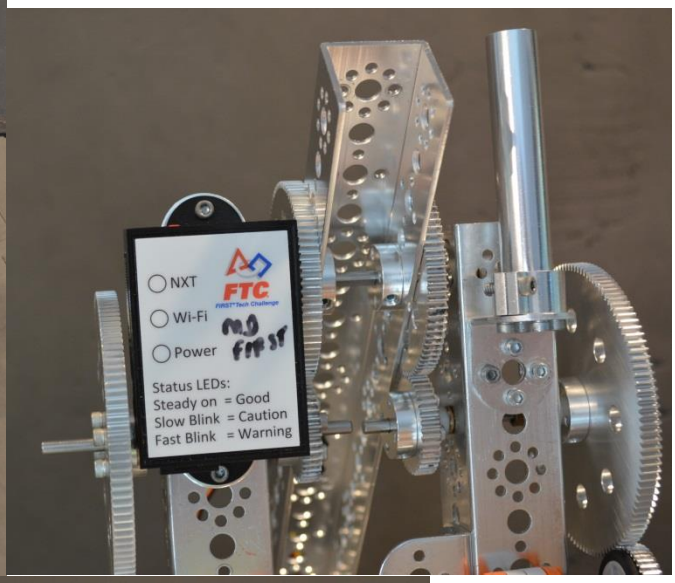


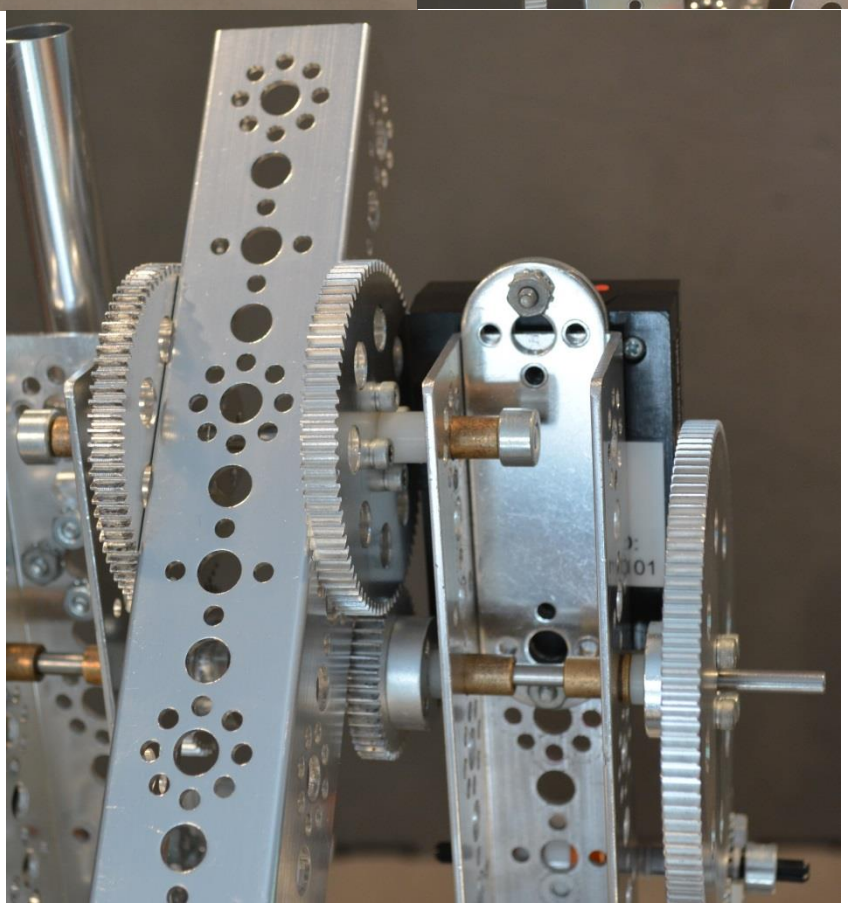


Step 41: Samantha module – the power and data cables will be installed in a later step, flat bracket (2), 1/2" socket head cap screw (2), kep nut (2)

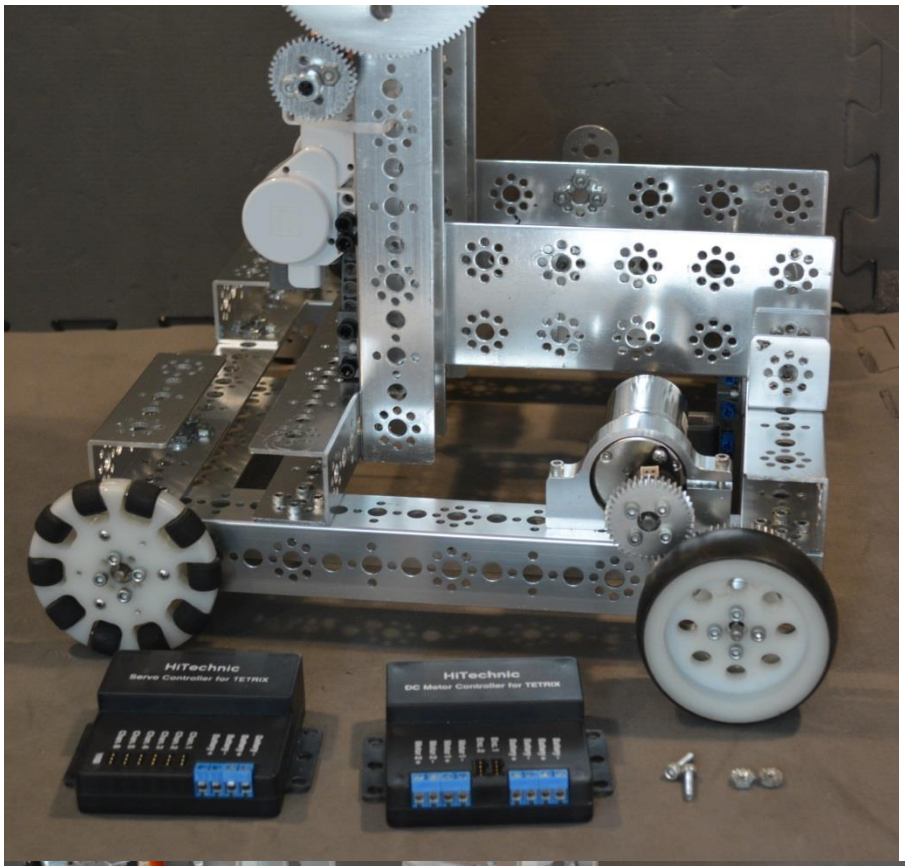


One of the brackets is mounted inside the channel. The other is on the outside, directly behind the Samantha. The module can't be mounted lower and directly on the channel, because the USB connector housing would hit the LEGO motor.

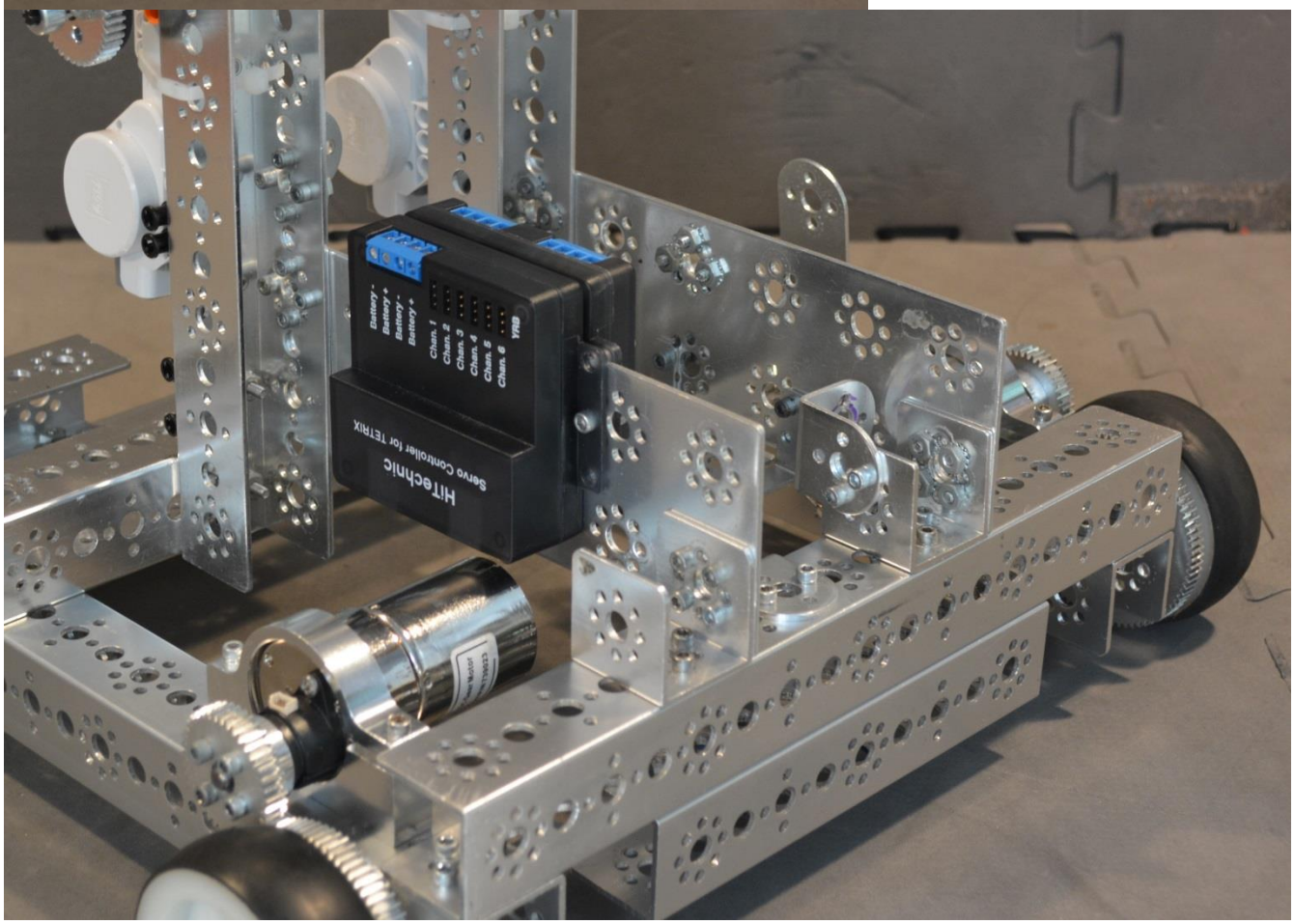




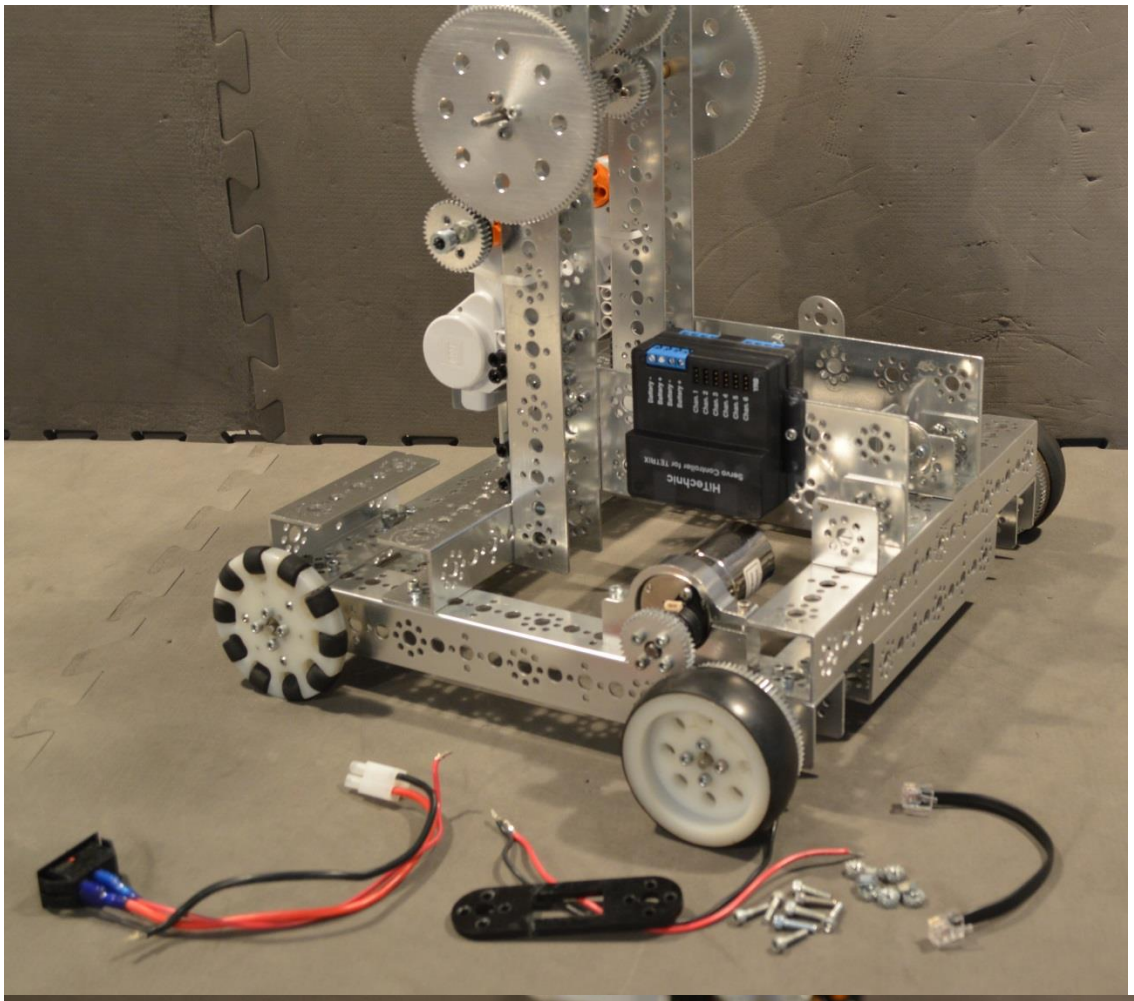
Step 42: assembly from step 41, DC motor controller – the power and data wires will be used later (1), servo controller (1), 1/2" socket head cap screw (2), kep nut (2)

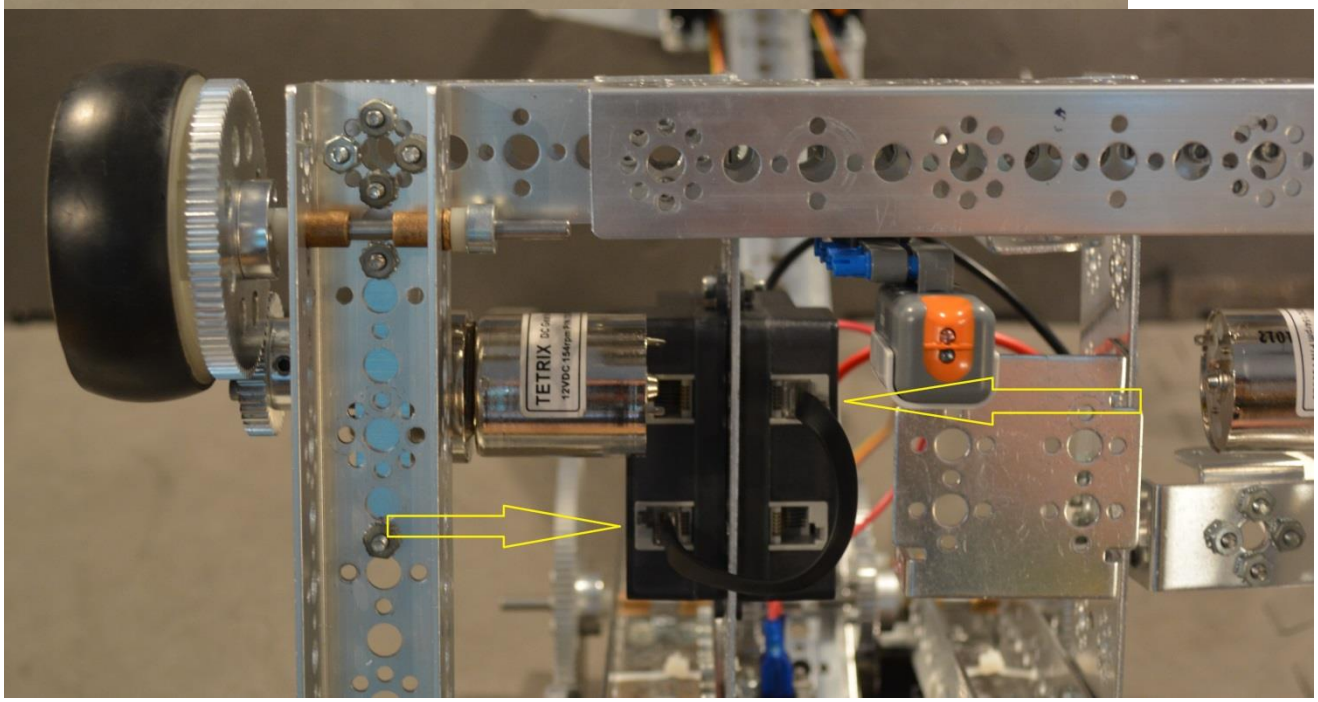


Mount the motor controller on the inside, so it is closer to the battery and the servo on the outside so it is easier to work with the servo wires



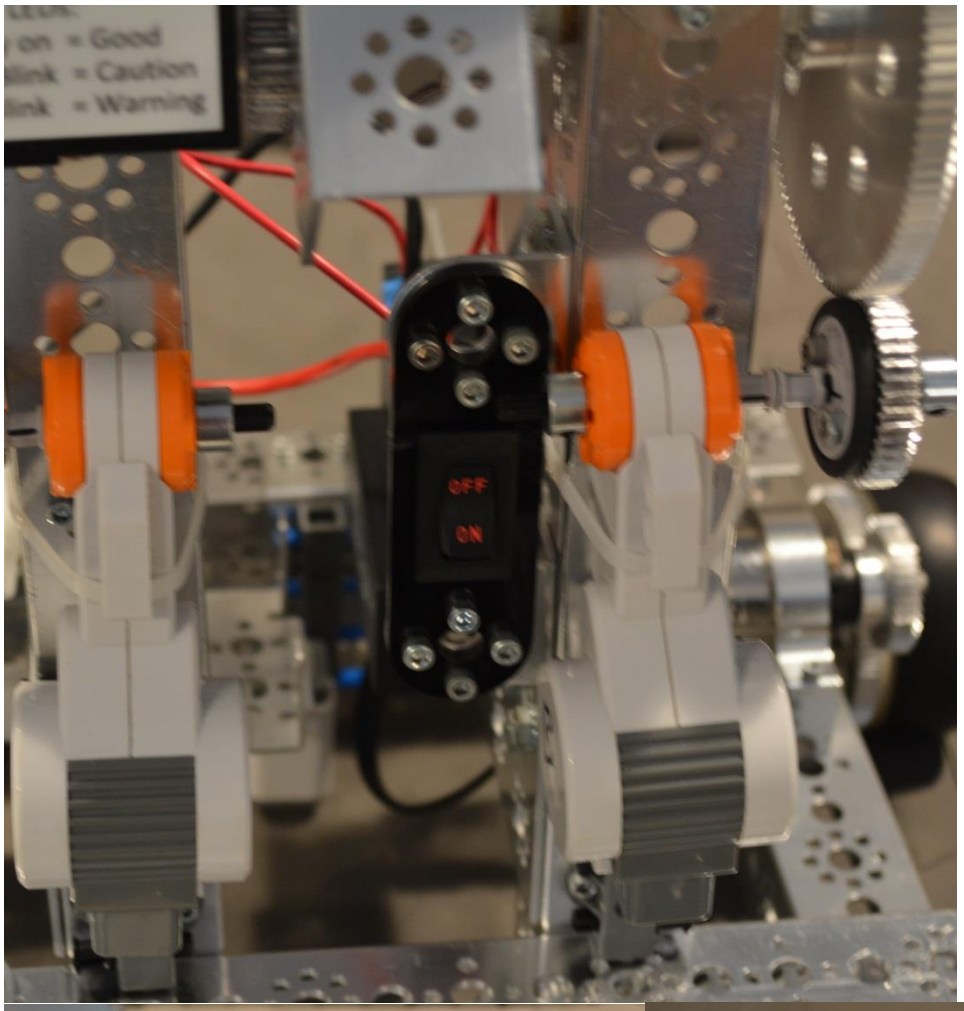
Step 43: assembly from previous step, on/off switch with wires attached, mounting bracket, power and data wires from the motor controller, 1/2” socket heat cap screws (8), kep nut (8)



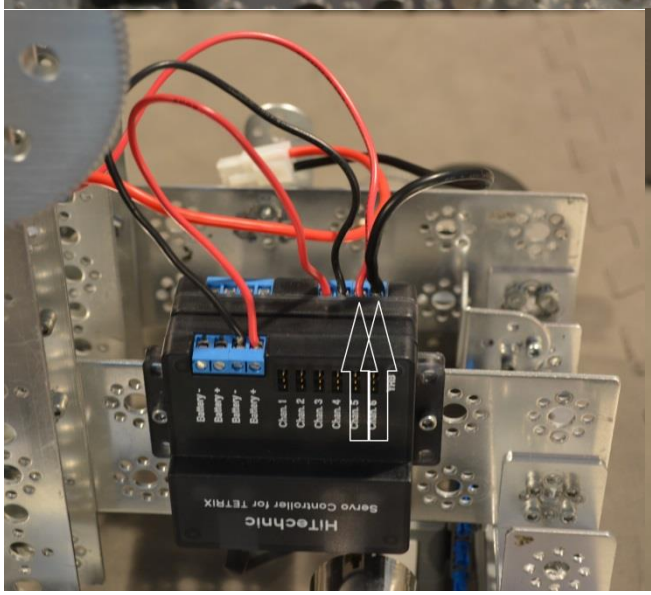


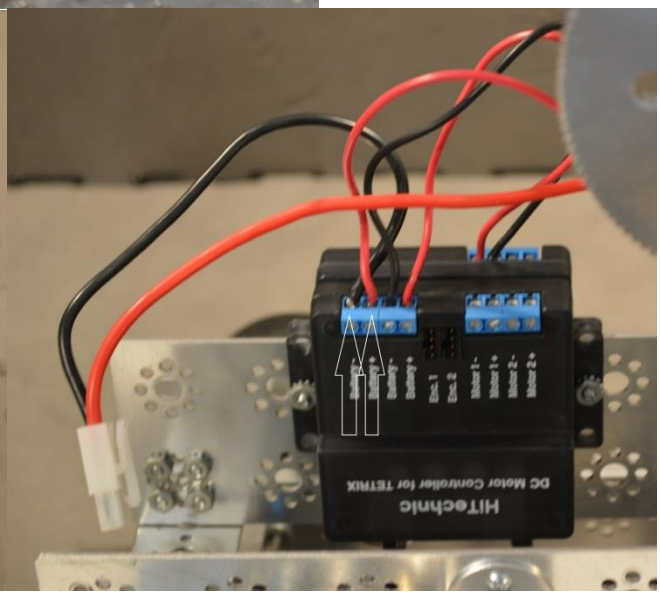
In the above picture, the data cable is connecting the DC Motor Controller and the Servo Controller.

Step 43 (continued)



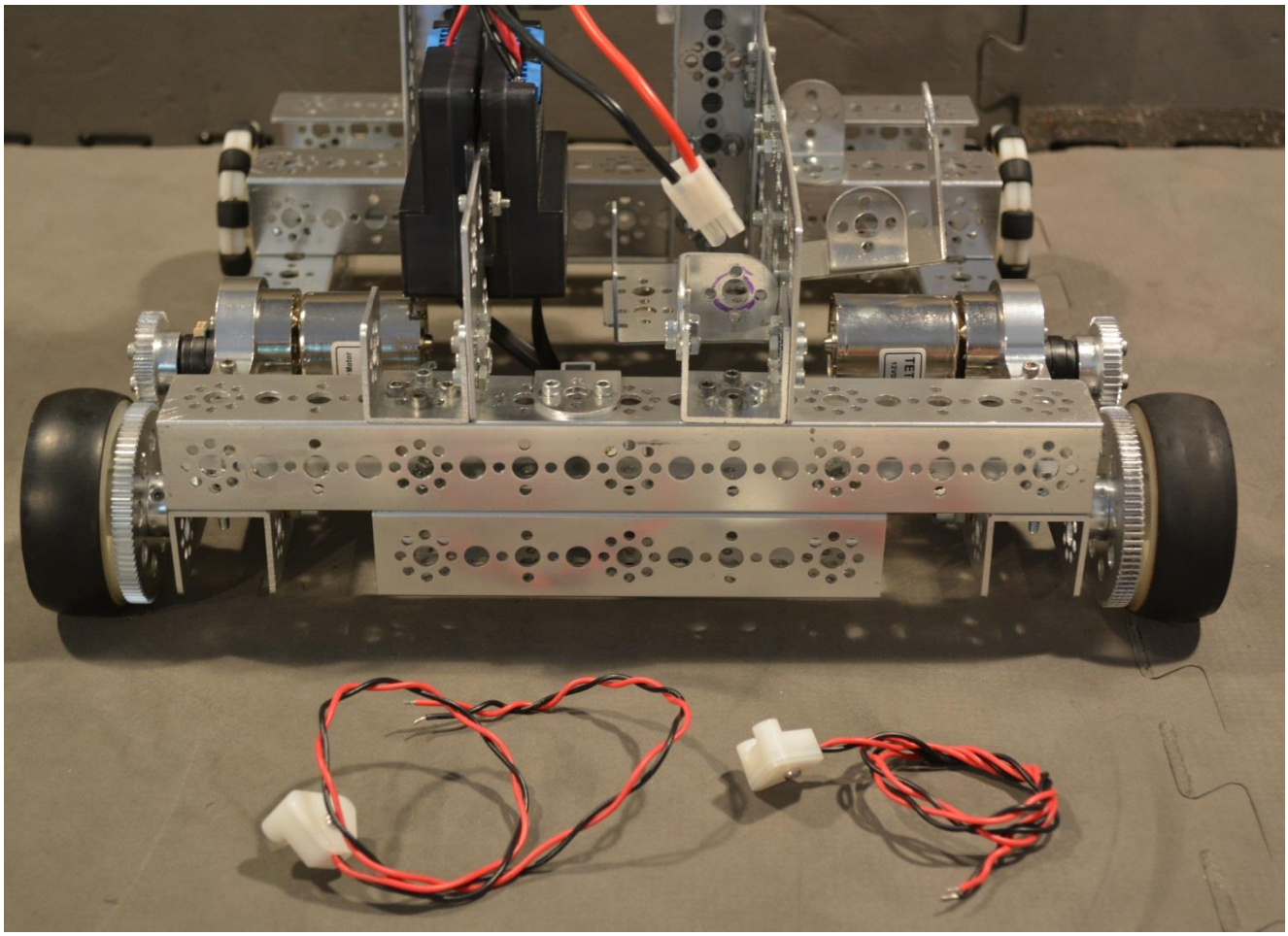
By convention, red wires are used for positive and black wires are used for negative.



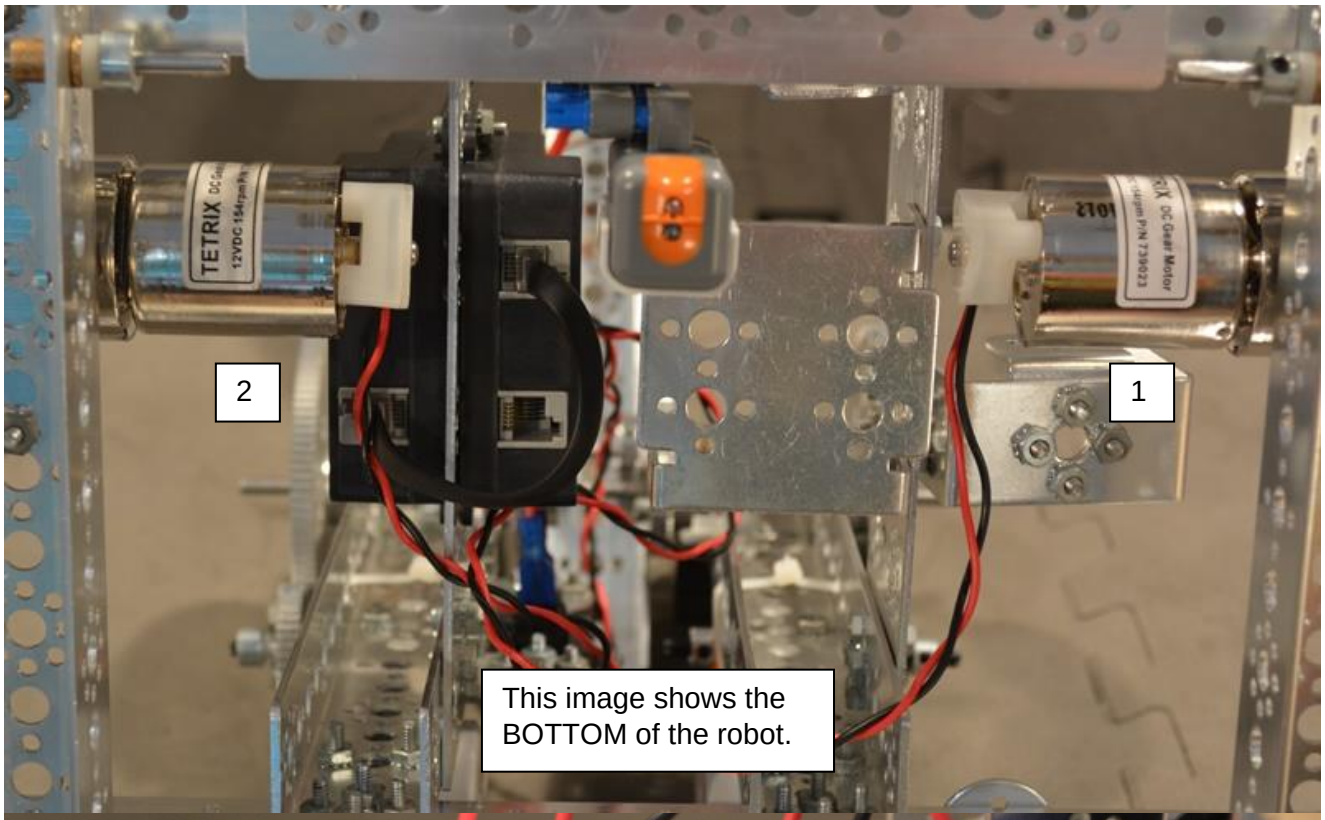


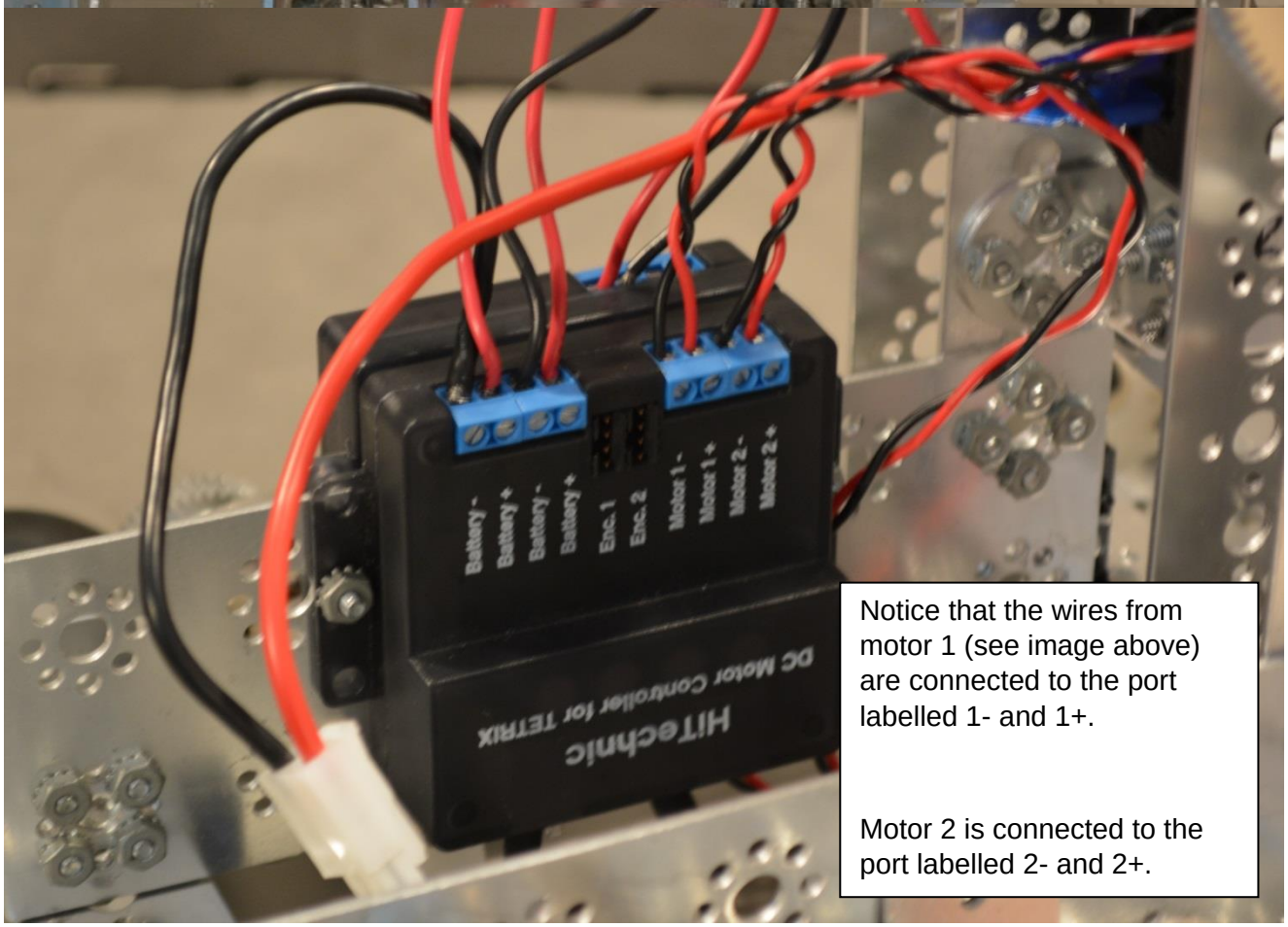
These pictures show how the switch is wired into the DC Motor Controller. The left picture is the view of the servo controller. The right is the view of the DC Motor Controller. The white arrows show the switch wires. The other wires run power from the DC Motor Controller to the Servo Motor Controller. This is called “daisy chaining the power”, because the power flows from the battery to the DC Motor Controller to the Servo Controller.

Step 44:

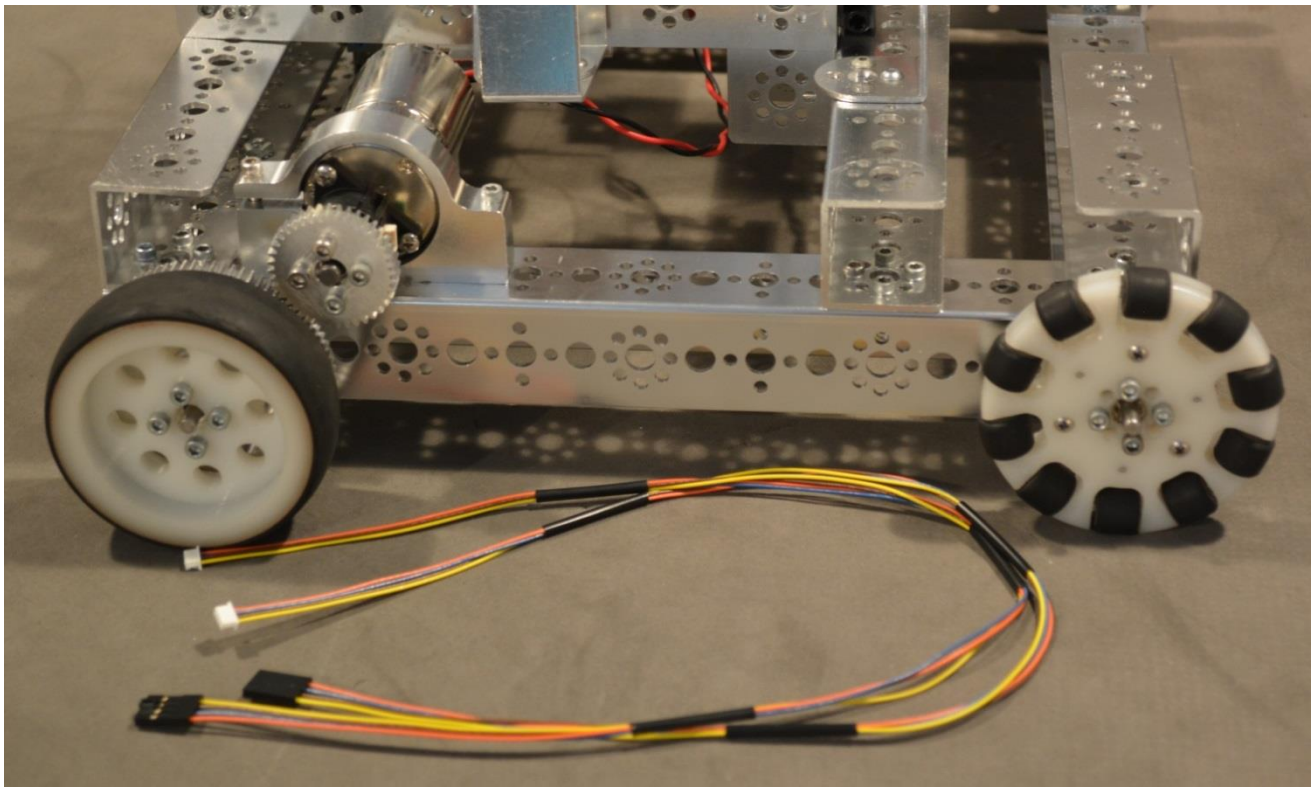


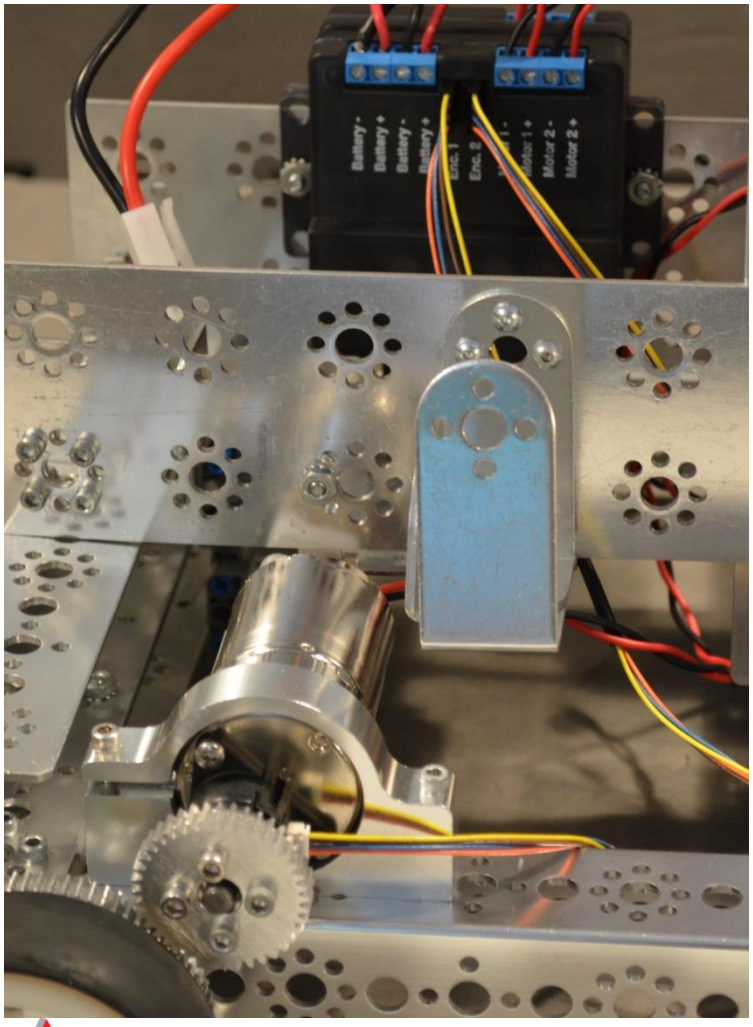
Step 44 (continued)





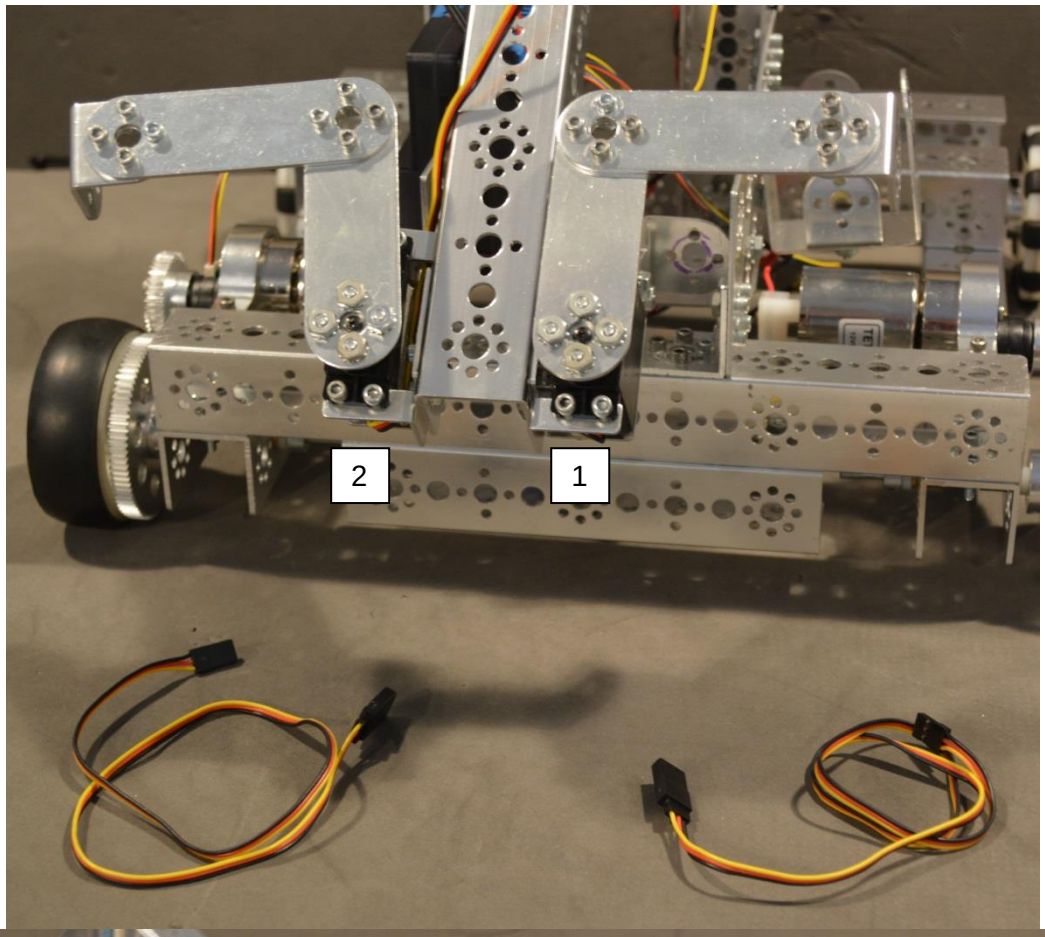
Step 45: assembly from the previous step, encoder wires from a previous step.

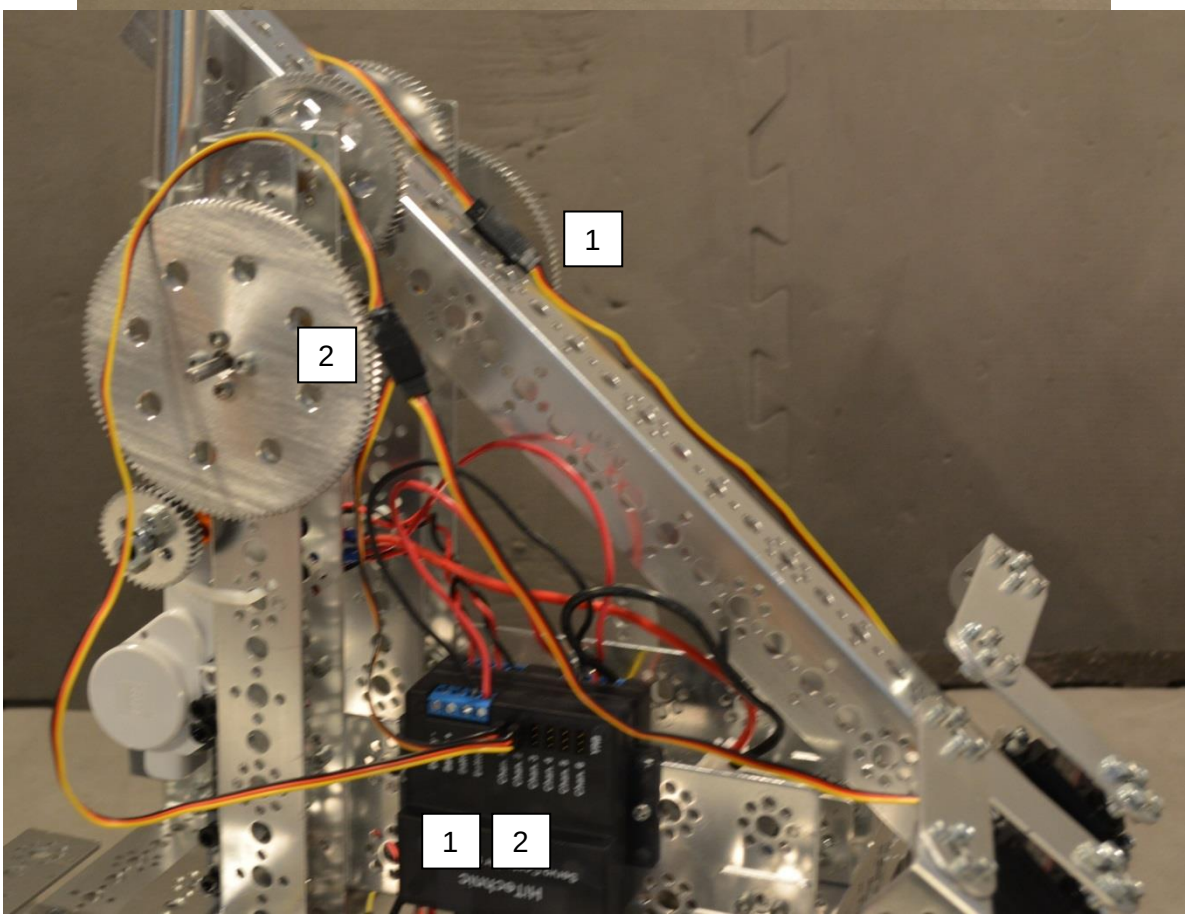




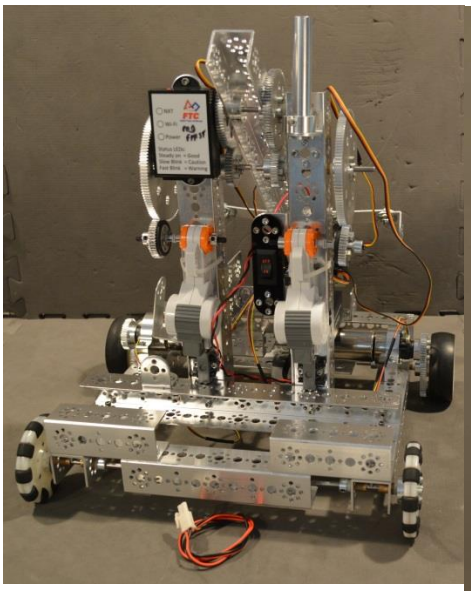
The wire from motor 1 (see step 44 – also shown in the images above and left) must be connected to port 1. The wire from motor 2 must be connected to port 2.

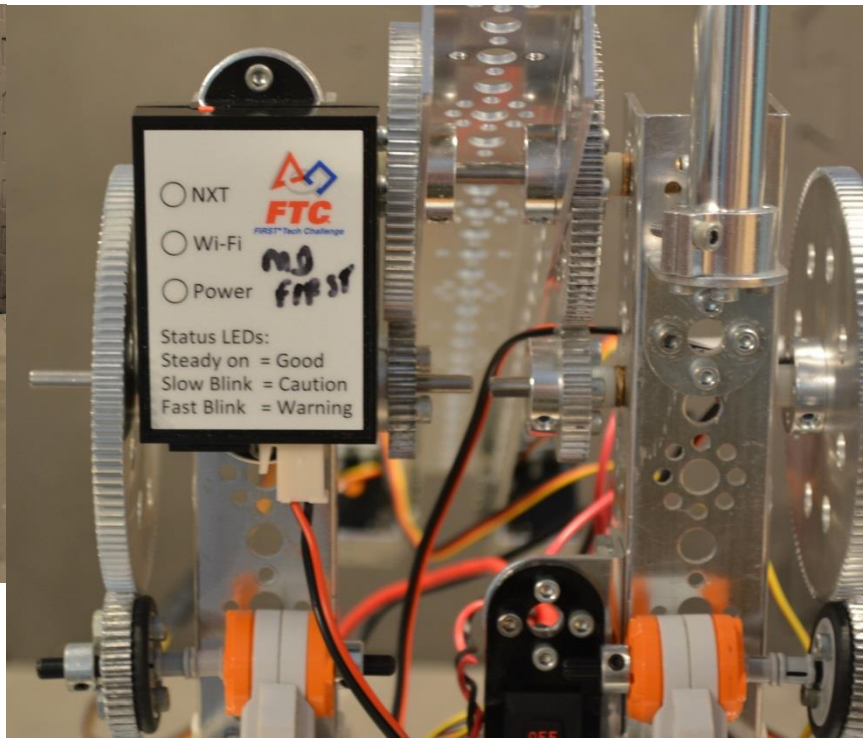
Step 46: assembly from the previous step, servo extension (2)



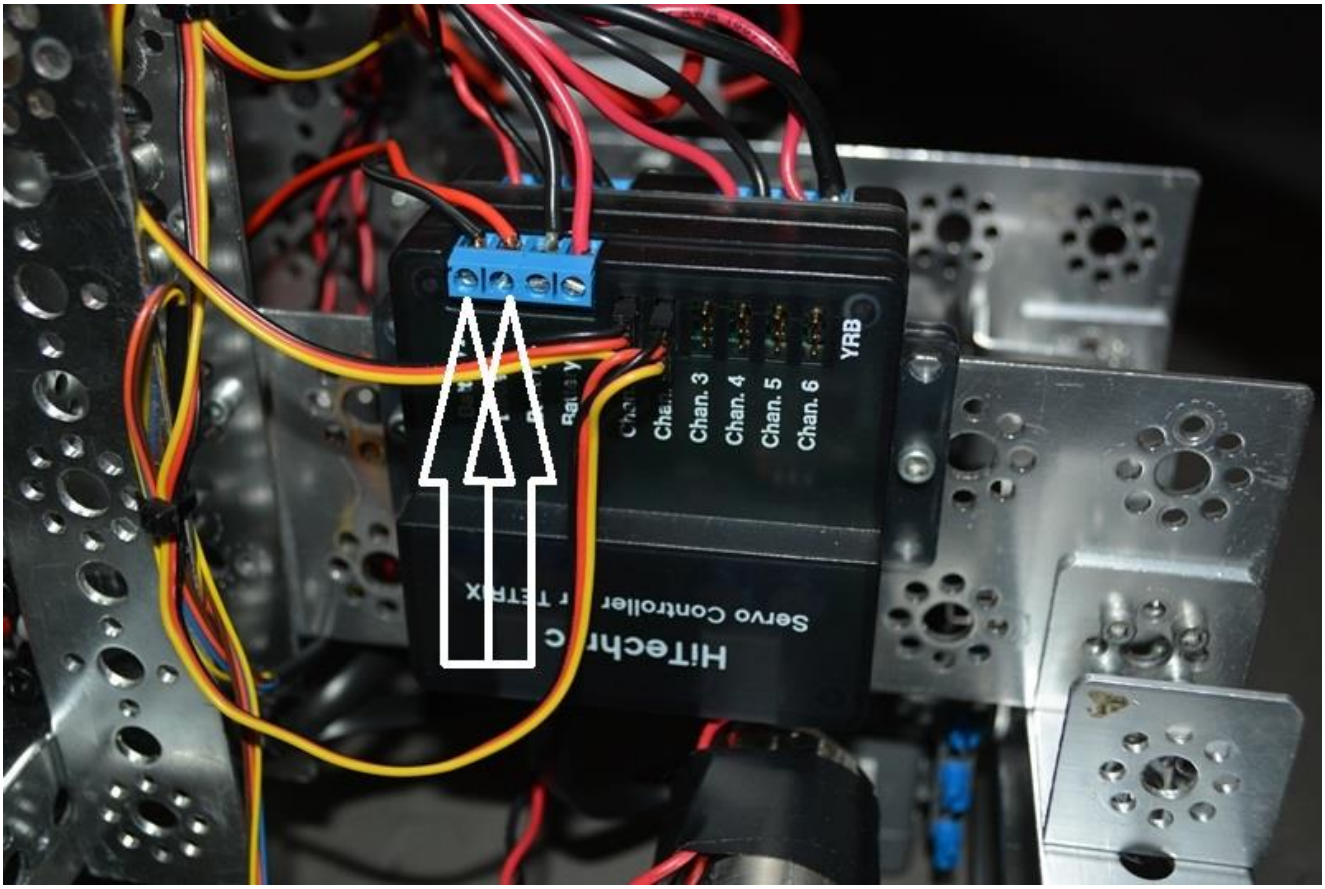


Step 47: assembly from the previous step, Samantha power cable from a previous step

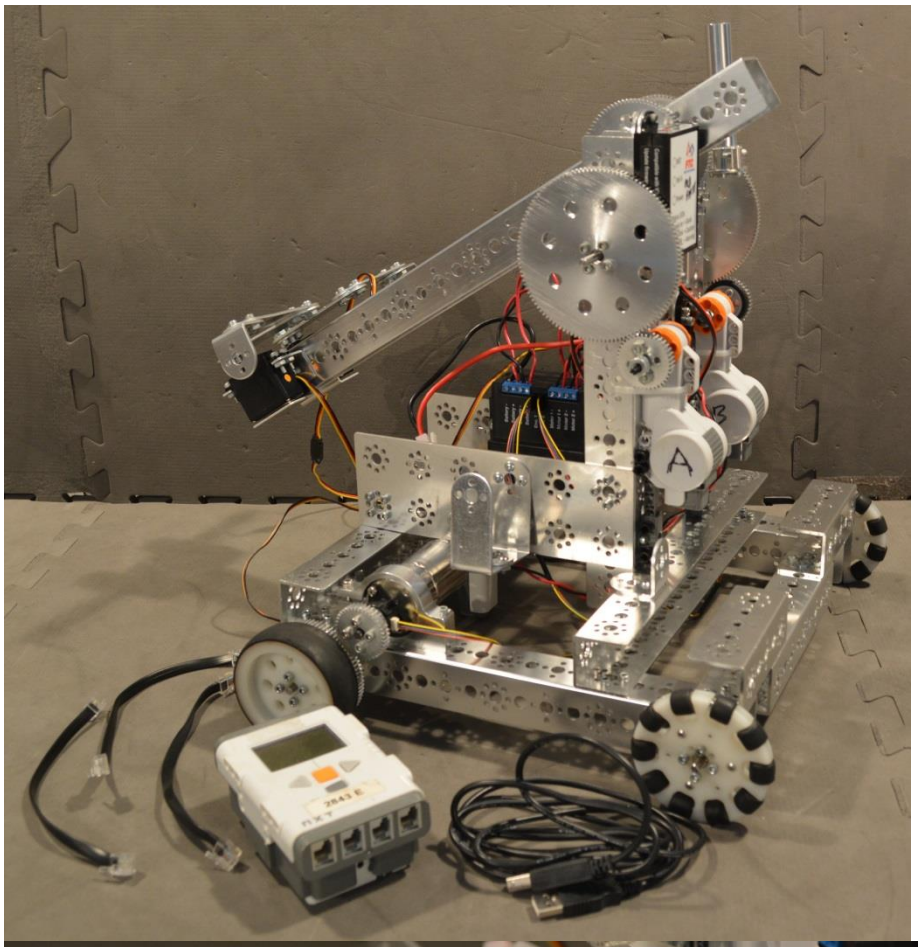




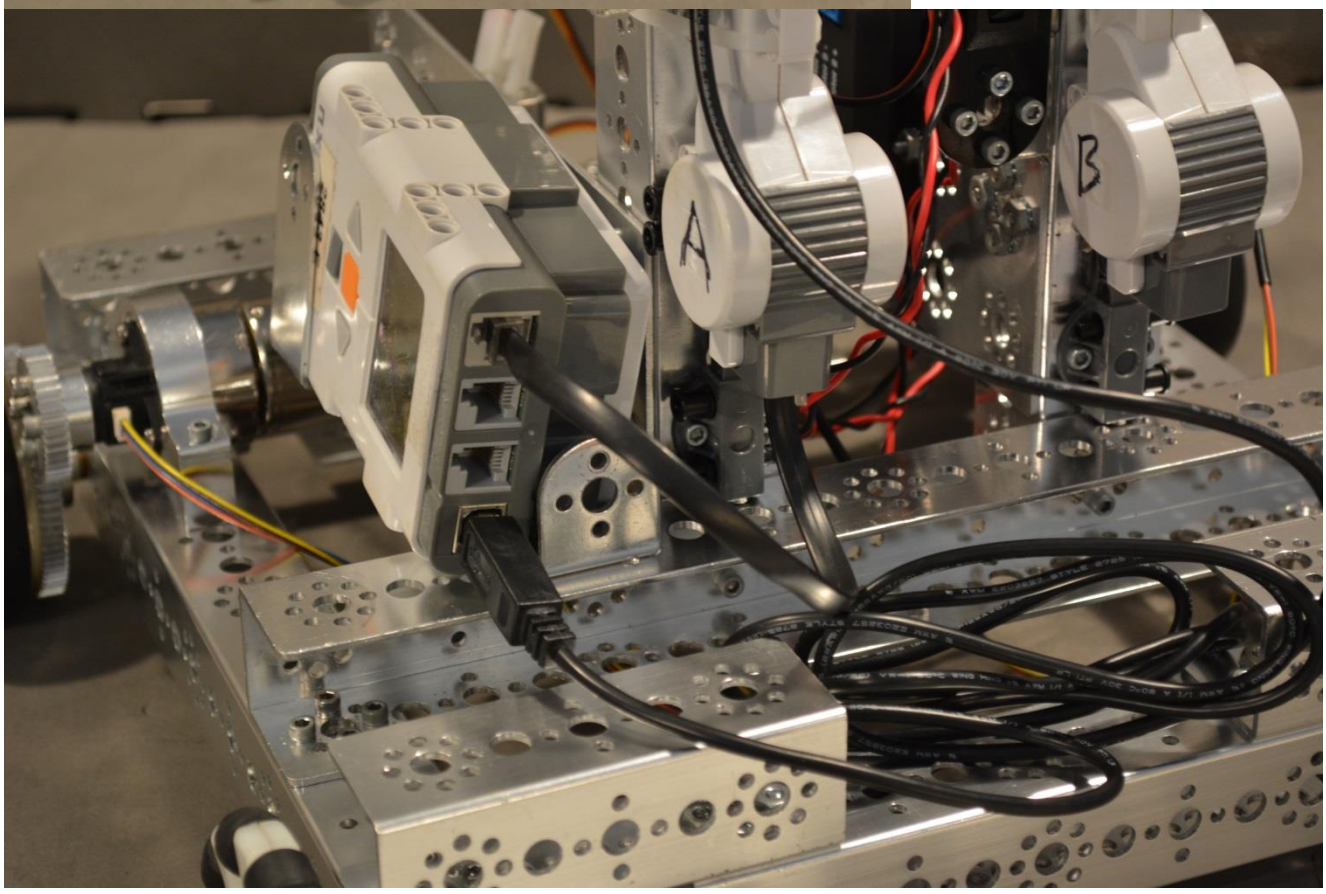
The bare end of the cable will connect to the servo controller's battery ports (battery 1- and battery 1+).



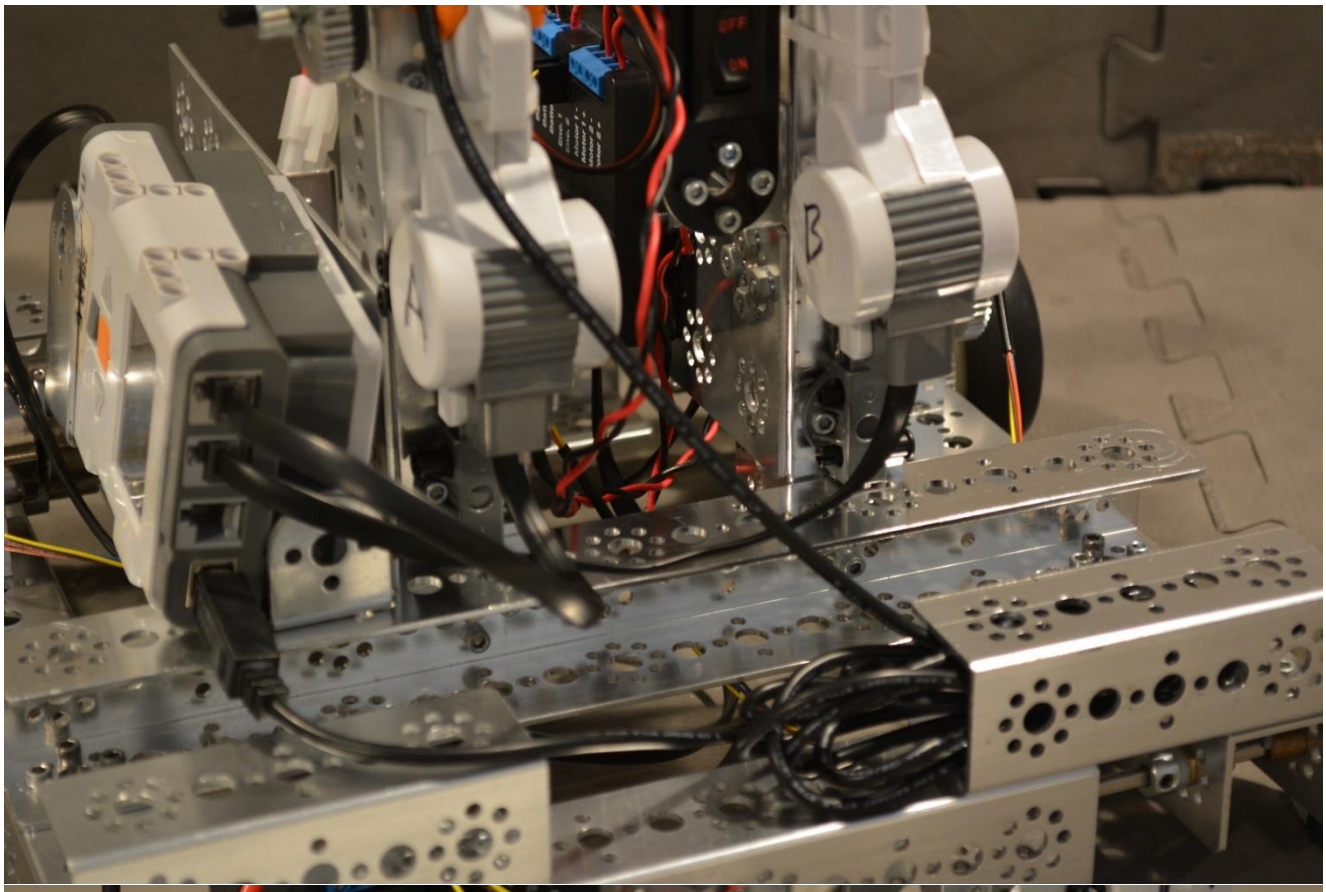
Step 48: assembly from the previous step, NXT, data cables from the motor controller or servo control packages, Samantha data cable (from a previous step)

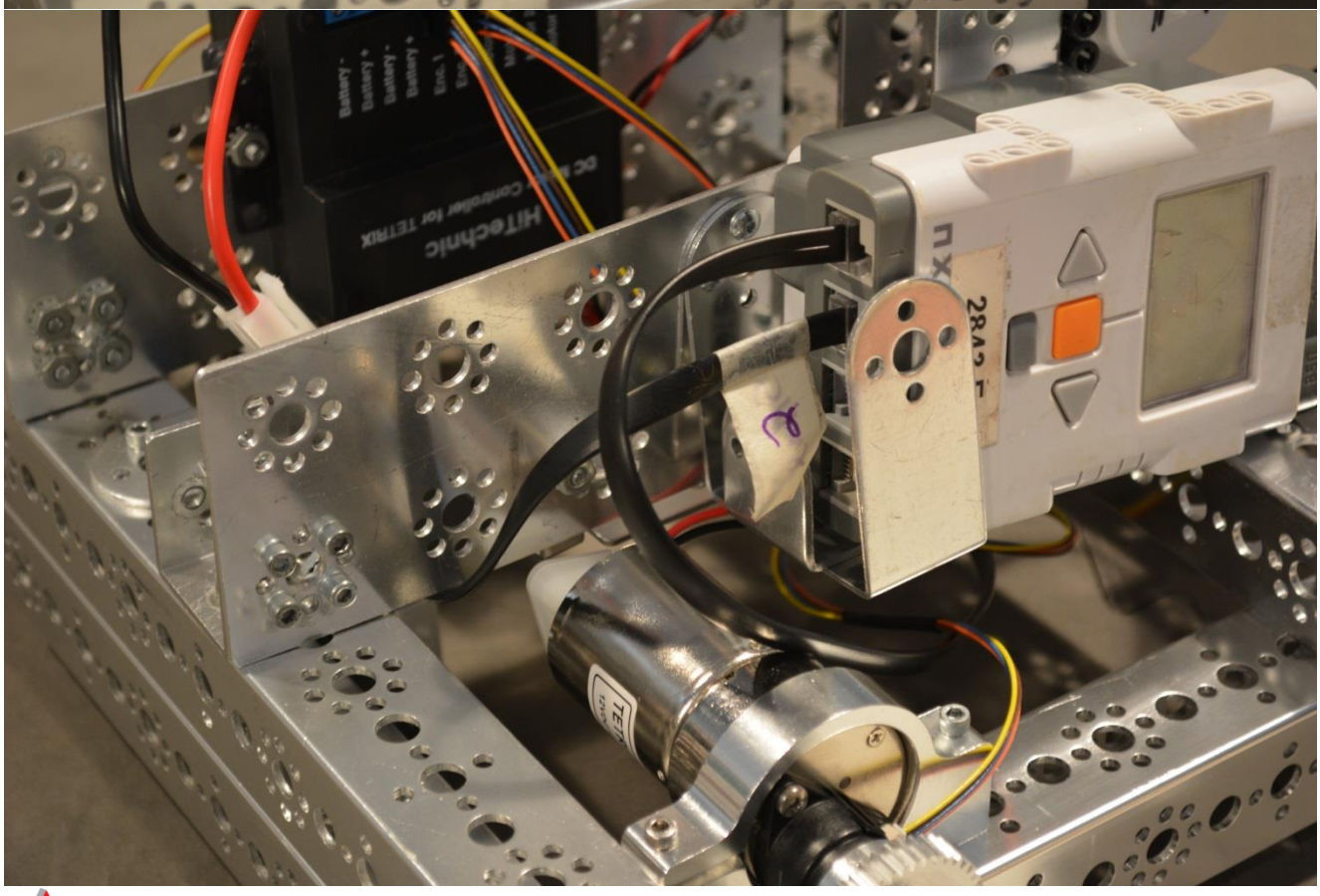


Plug LEGO motor A into the NXT port marked A.

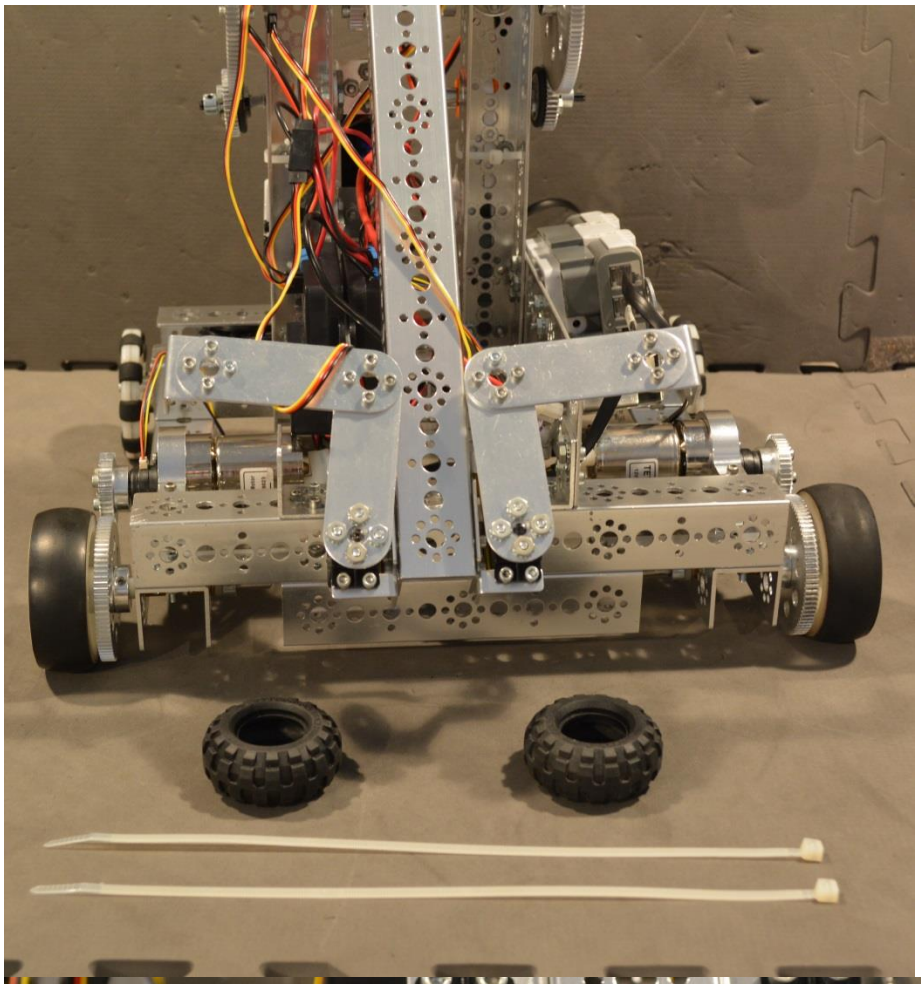


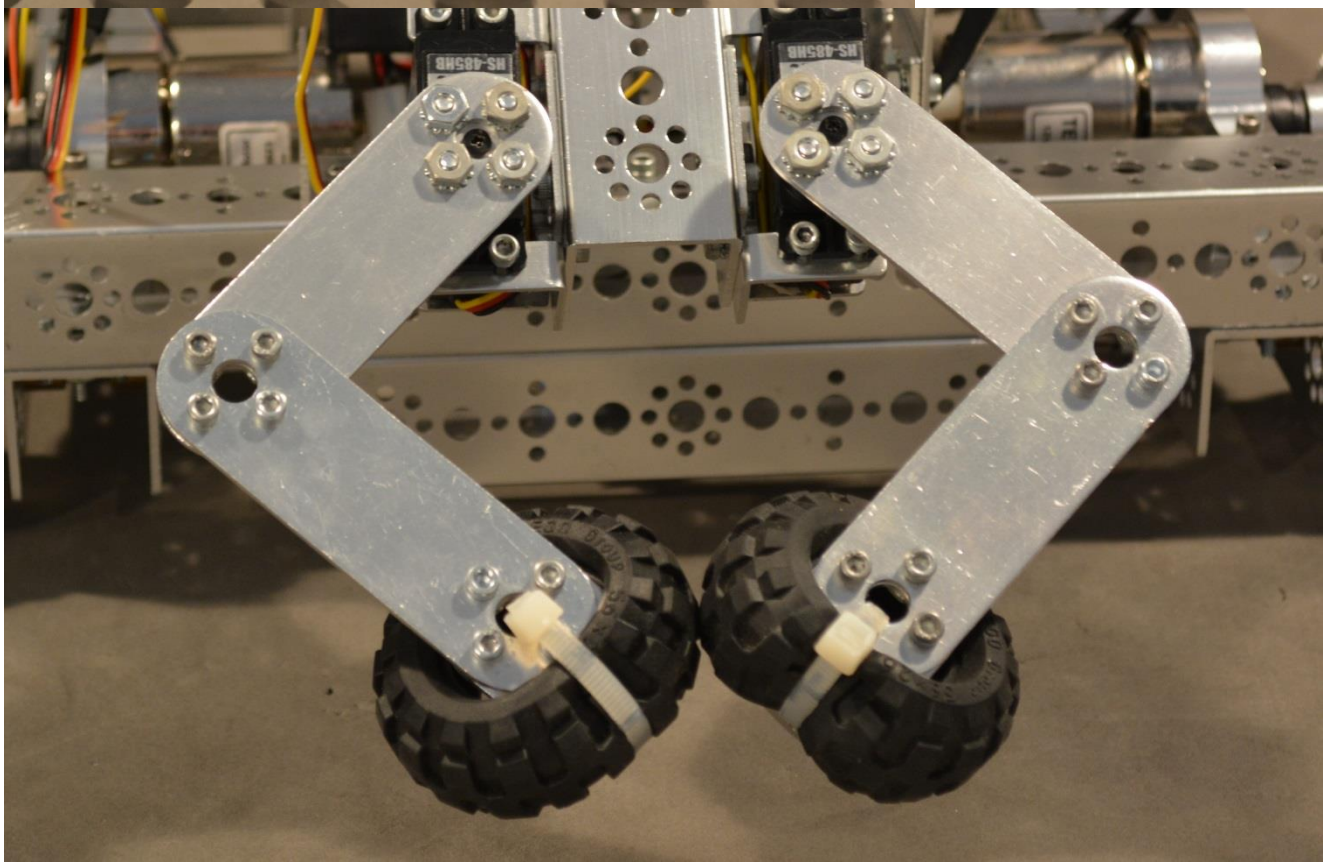
Step 48 (continued) – Plug LEGO motor B into the NXT port marked B (top image). Plug on end of a data cable into the motor control data cable port and the other end into port 1 of the NXT (bottom image). The bottom image also shows that the light sensor is connected to port 2 of the NXT.



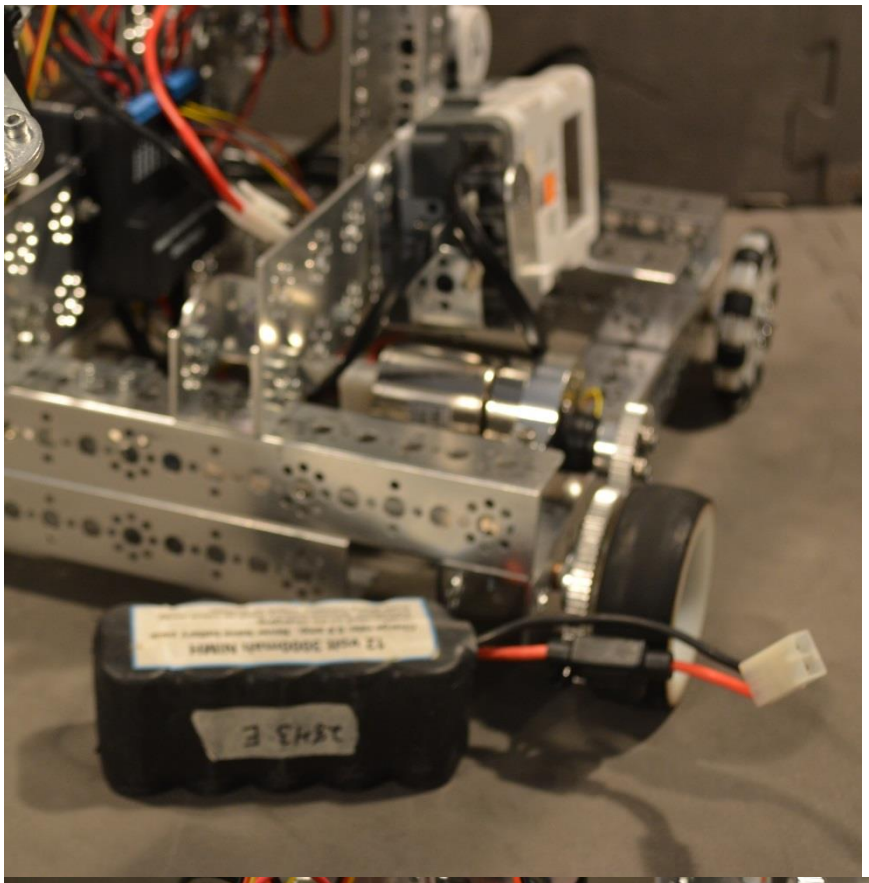


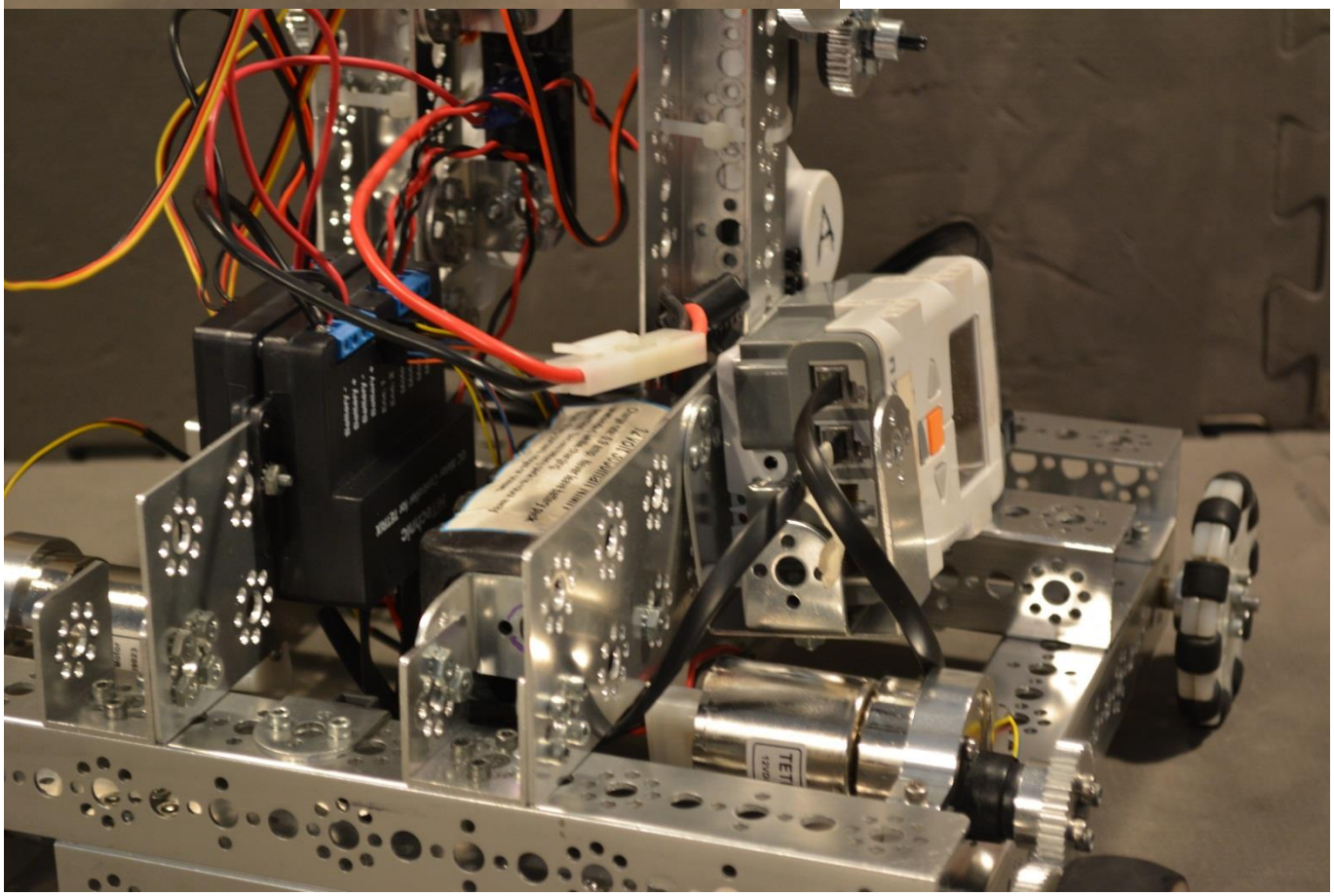
Step 49: assembly from the previous step, 56 X 26 black tire (2), zip tie (2)



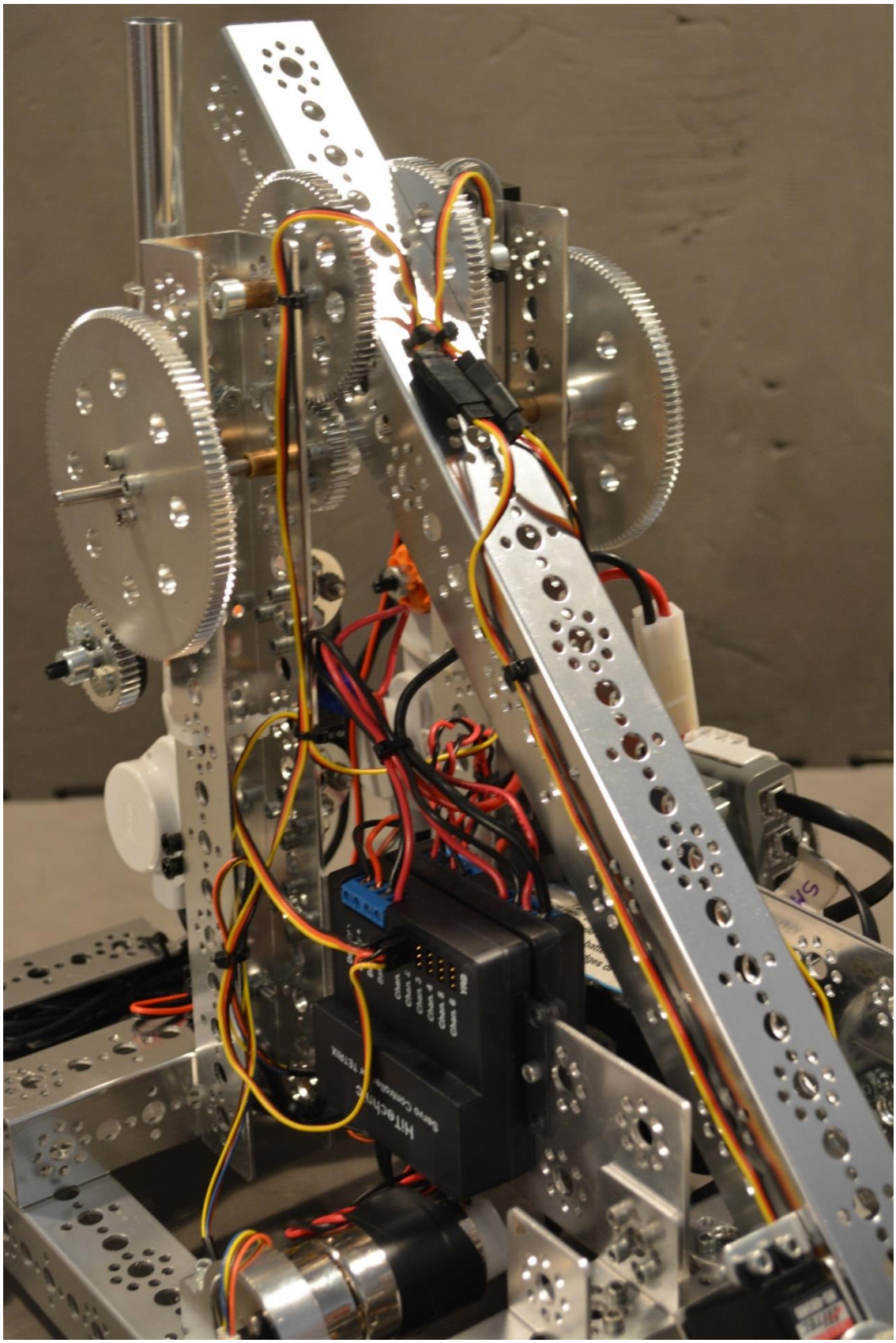


Step 50: assembly from the previous step, rechargeable battery pack

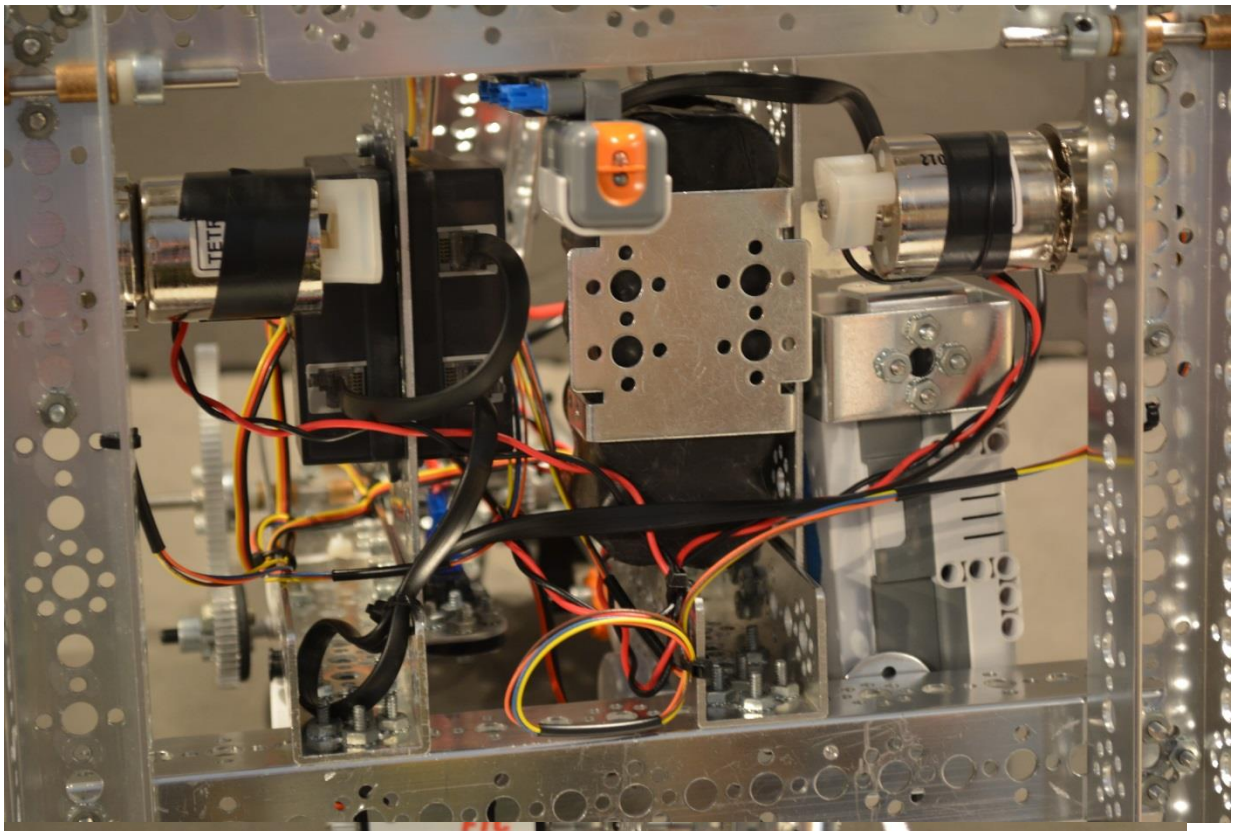


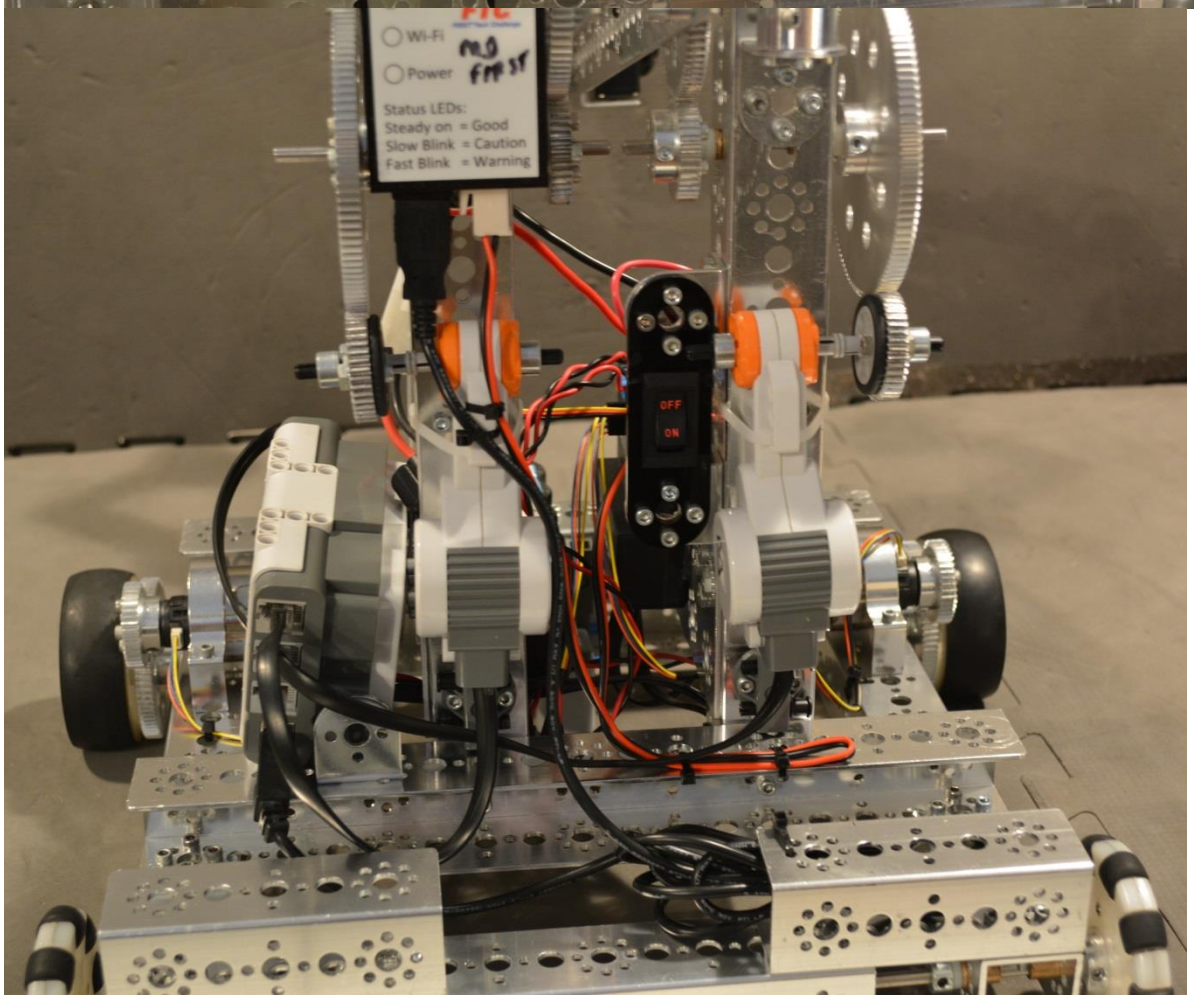


Step 51 (optional): This step shows an optional, but strongly recommended. It requires parts that are NOT part of the kits. On this robot, small black zip ties were used to secure the wires to the metal channels. Tape, string or zip ties (as long as it is allowed by the game manual) can be used to secure the wires. Robots come into close contact with other robots and game elements during a match. To prevent entanglement, secure all loose wires.

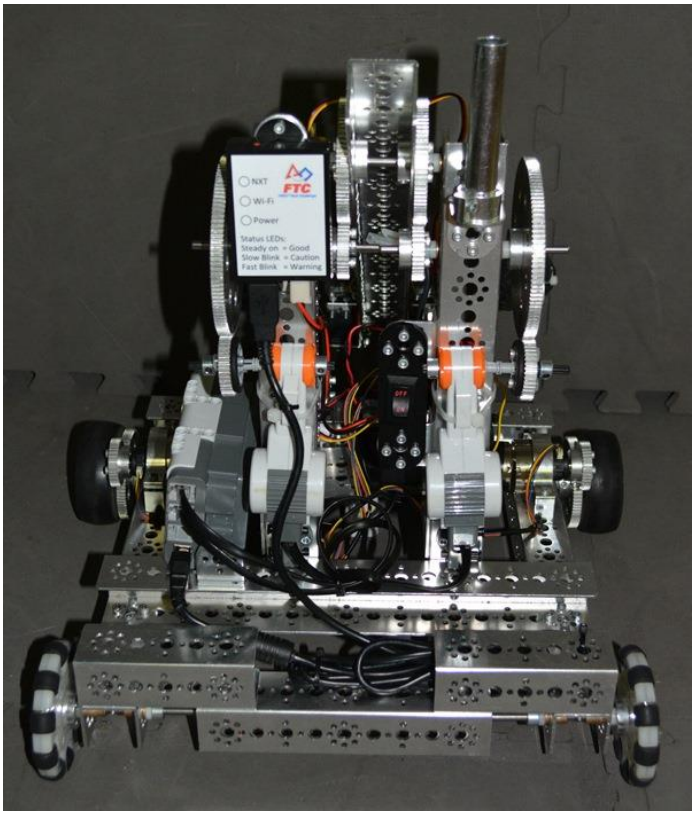


Step 51 (continued): Notice the electrical tape securing the power wires to the DC motor.





Push Bot Movement



The LabVIEW® and ROBOTC programming sections show how a Logitech controller (F310) can be used to steer the robot. The left joystick controls the 'left' motor and the right joystick controls the 'right' motor. The top image shows the back of the robot. Contrary to unspoken convention, the omni-wheels are on the back of the robot and the traction wheels are on the front of the robot. The NXT is on the left side of the robot.

Push both joysticks away from your body and the robot drives forwards. Pull both joysticks toward your body and the robot drives backwards. Push one joystick up and the other down and the robot turns.

The LabVIEW® and ROBOTC programming sections show how a second Logitech controller (F310) can be used to raise and lower the arm and open and close the ‘hand’.

The ‘arm’ allows the robot to lift items from its lowest level and to at least 26” above the ground. The arm uses a 1:6 gear-ratio, which enables the arm to lift moderately heavy items. Push the left joystick away from the body to raise the arm; toward the body to lower the arm.

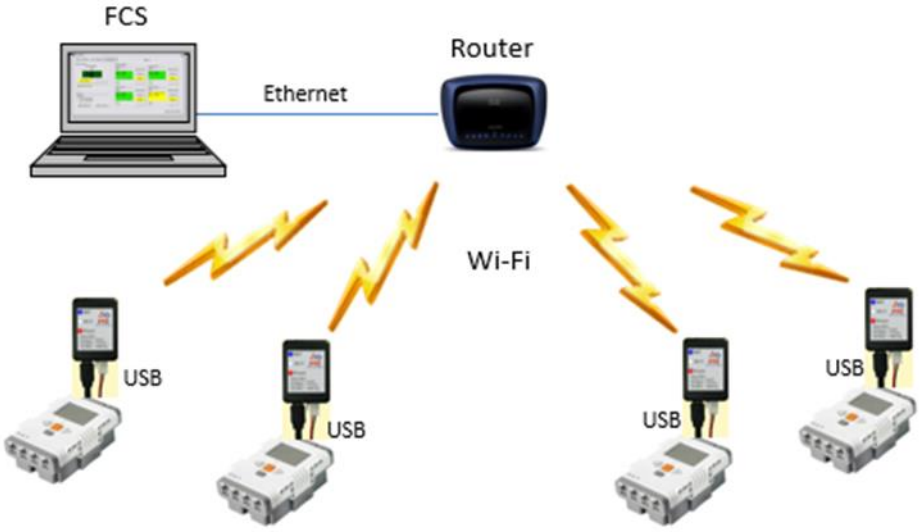
The ‘hand’ is capable of opening and closing instantly or gradually. Buttons ‘6’ opens the hand; button ‘8’ closes the hand. Button 5 opens the hand slowly and button 7 closes it slowly.

Wireless Communication

The Samantha Module explained

For the past few years, FIRST has used the Samantha Module as a key part of the robot control system to allow the field to control and monitor the robot and to pass the joystick commands to your program. As your lifeline to talking to your creation, part of your success comes from giving that communication channel the best chances of being stable and reliable. In addition to being used by the Field Control Software (FCS), both ROBOTC and LabVIEW® will connect wirelessly to your robot using the Samantha Module.

In understanding the role that this small black box plays, let's take a quick look at what it does and how you can make it be a part of your success strategy. In simple terms, you should think of the Samantha Module as if it were an antenna for your LEGO® Mindstorms® NXT. Like any antenna, it works the best when it is clear of any metal and has a good view of the network talking to it. If you have ever had a cell phone call drop out, you know that even with the best of technologies, it is not possible to guarantee a connection 100% of the time.



With this in mind, here are some tips which we hope give you the best chance of being successful at your tournaments. Just remember that you want to keep it:

Secure --- Powered --- Accessible --- Visible

Keeping it Secure

Without a doubt, the number one problem that happens at tournaments is teams breaking the USB connector on the Samantha module. What you want to do is make sure that the USB cable connecting into the Samantha Module is tight and secure. In addition, you can make the FTC Game Inspectors happy with you by taking a nice USB extension cable and cable-tying it in place to the Samantha Module and making the other end easy for the inspectors to access so they can program the network information quickly.

Keep it Powered

If you lose power to the Samantha Module, the field can no longer talk to your robot and it is off to fend for itself. Make sure you have a number of spare 9V batteries to power the Samantha Module during the Tournament day. If your robot seems to lose connectivity whenever it hits a bump, you probably have a loose connection between the Samantha module and the 9V battery box that you need to find.

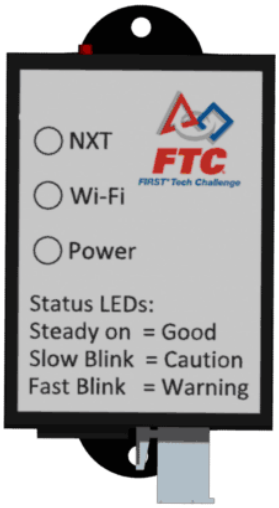
Keep it Accessible

If you hold the Samantha Module, the actual antenna is about half the size of a postage stamp and is located almost immediately below the FTC logo on the front of the case. At all costs, you want to have this as free from metal obstructions as you can and mounted as high up as possible.

Keeping it Visible

Even though there are only three LEDs on the Samantha Module, FIRST deliberately made them extra bright so that they can clearly communicate the status of the robot to you and the field personnel. You can see the codes documented on the *FIRST* web site. The easiest way to read this is: Red LED on and solid means you have power. White blinking means the module sees the network, while White solid means it is connected to the field. Blue solid means it is talking to the NXT. So Red, White and Blue all on means you are ready to go.

If anything goes wrong, you should be able to look at the lights on the Samantha module and quickly determine if you are at least talking to the robot.



Blue solid means the module is talking to the NXT.

White blinking LED means the module sees the network, White solid means it is connected to the field.

Red LED on and solid means you have power.

The Samantha Module and the TETRIX Controller

[To Do]

The TETRIX Robotics kit includes an integrated motor controller that has 4 motor and 4 servo ports. This integrated motor controller was not designed to allow the Samantha wireless communication module to be connected to the same power supply as the motor controller. The power circuit for the motor controller shares the same ground plane as the sensor port that connects the controller to the MINDSTORMS NXT Brick. If the Samantha device is powered from the same battery as the motor controller, then under certain circumstances a large current could inadvertently flow through the NXT Brick and destroy the controller, the NXT Brick, and the Samantha module instantly.

The manufacturer strongly recommends against powering the Samantha module from the same battery as the motor controller.

A 9V alkaline battery can be used to power the Samantha module directly. The Samantha Module is supplied with a 9V battery box. The Samantha Module is connected to the NXT via a USB cable. The NXT Brick is connected to the MATRIX motor controller through Sensor Port 1. The motor controller has its own separate, larger 9V battery pack.

Some important items to note when powering the Samantha module using a separate 9V battery:

Teams should make sure to disconnect the battery in between matches to avoid draining the battery.

Teams should also disconnect the battery in between matches to minimize the traffic that is on the competition field's wireless network.

The Field Control System (FCS) software polls the Samantha module to check the battery voltage of the robot. If the Samantha is powered by a separate 9V battery, the FCS will not be able to check the actual voltage of the robot's main power supply. Instead, the FCS will only check the voltage of the 9V alkaline battery.

The FCS will highlight this lower-than-expected voltage condition in red in the FCS window.

Samantha Module Installation - Best Practices

The Samantha module provides a highly reliable WiFi connection between the FTC robot and the Field Control System. However, the integrity of this communication link is dependent on the USB connection between the Samantha module and the NXT brick.

During FTC matches robots are subjected to rapid accelerations and decelerations, jarring impacts, rapid turns, etc. These actions could lead to a strain on the USB connection which in turn could cause the cable to become loose.

Loose cables could cause:

A loss in communication between the Samantha and the NXT.

A loss of tele-op/Remote Control for up to 10 seconds while the link is reestablished.

Permanent loss of communication.

The NXT to lock-up.

Guidelines to Avoid Communication Issues

1. Mount the module vertically with the USB port facing downward. The cable should hang down straight with no strain put on it in any direction.

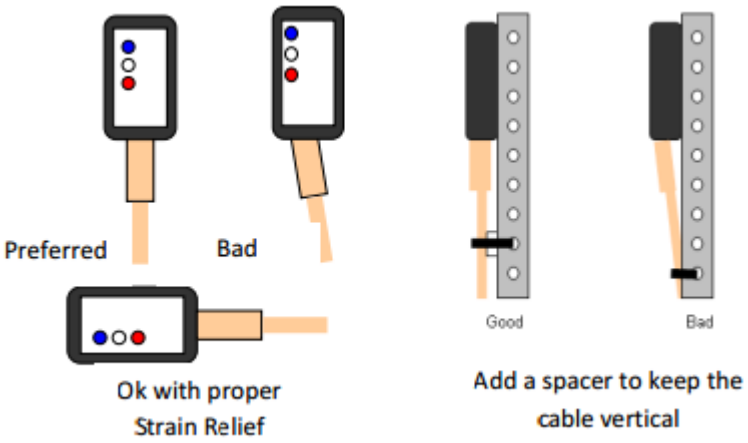
2. Horizontal mounting has the most potential for loosened connections, but this can be mitigated with proper strain relief.

3. Prevent any relative motion between the NXT and the cable by using zip ties to hold the cable in place. Make sure the NXT is securely mounted as it will tend to shift during rapid robot maneuvers.

4. Use as short a USB cable as possible, usually 30cm, to eliminate a large mass of cable swinging around your robot and pulling on the connections. After you have coiled the cable make sure to zip tie it securely in several locations to prevent it from shifting during a match.

5. Mount the module as high as possible on your robot. This will improve WiFi reception and make it easier for the refs and FTAs to see the lights, which will help them diagnose any potential communication issues.

6. Do not bury the module behind a mass of metal, like the chassis or large sheets of aluminum. This will create a Faraday cage and prevent the WiFi signals from reaching the module.



Things to Know about the Samantha Module

- There is a button on the Samantha module that you must press at certain times. Make sure you can get to it.
- There is a USB connector (Female-A) on the Samantha module which you must access during software inspection.
- You may wish to direct connect your NXT to your laptop for programming during a competition. You must unplug Samantha from the NXT to do this and reconnect it before your next match.
- Samantha has 3 LED lights that give lots of information. Make sure they are easily viewed.

Conclusion: Protect, but DON'T BURY your Samantha module inside your robot!

- You can use Samantha right out of the box, but you will need to make some changes to your home network router.

  OR

- You can use Samantha with your existing home network (as is), but you will need to make some Samantha reconfiguration.
- Your Samantha module may be modified by the software inspectors at each event. You may need to change it back if you want to use after the event.
- You can do all your development without ever using Samantha, HOWEVER, you MUST use Samantha at your competitions.
- Recommend testing your Samantha well in advance of your first event to insure that it works!!!!

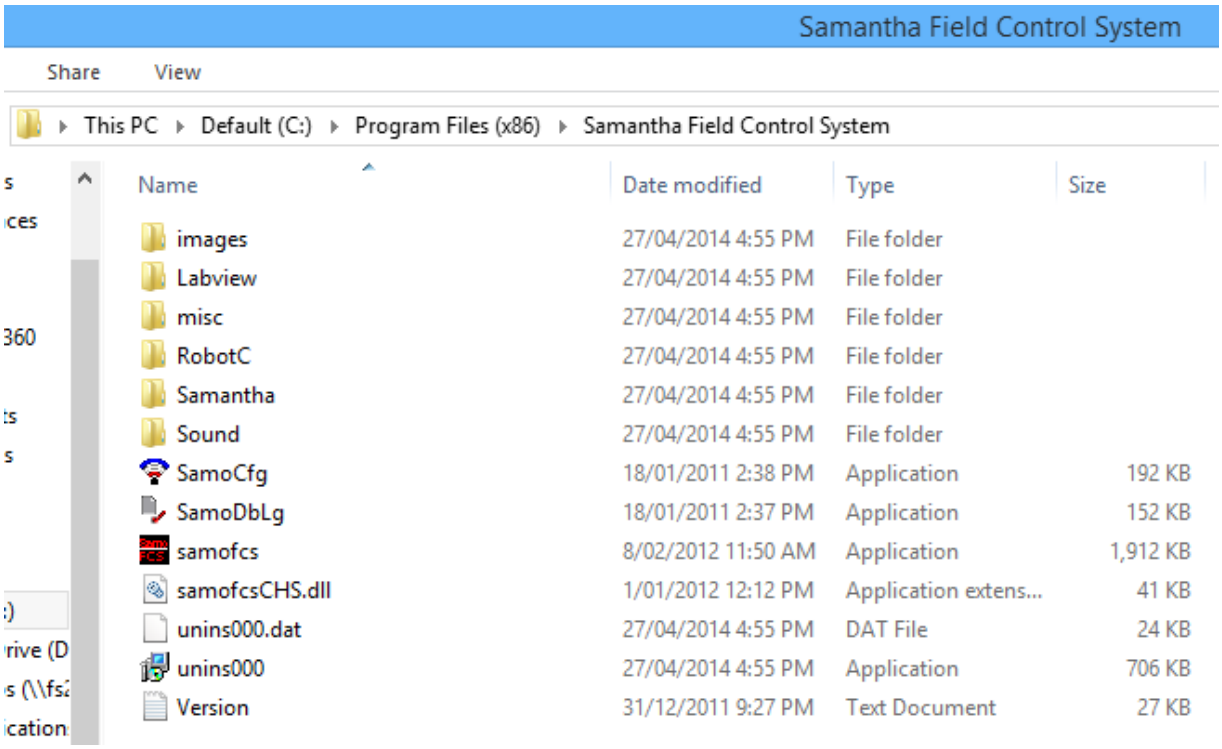
Setting Up the Samantha Module

1. Download and install FTC's Field Control Software package. This can be found on FIRST's site (usfirst.org).

2. Find out the Wireless network you intend to use SSID, encryption type, and password.

3. You will need a 4GB or smaller USB flash drive, formatted to FAT32 file format. To format the flash drive, navigate to the Start menu, select Computer, right click on the flash drive's name, then click Format. In the drop down box, select 'FAT32' as the file format, and check the 'Quick Format' box (for a faster format). Leave the Flash Drive plugged in for step 4.

4. Open the Samantha Field Control System Folder on your computer and run the program called "SamoCfg".



5. Enter all the information for your network and click. Two message boxes will pop up, the first one informing you that the password you entered will be converted to an encrypted key and the second informing you that the config file has been saved to your flash drive. Close the program.

6. Your Flash Drive will as need the Samantha Module's firmware on it. Navigate to the Samantha Field Control System Folder (shown above). Open the Folder called 'Samantha' and copy the Samantha.hex file into your flash drive.

Flash the Samantha Module

You must now update the Samantha Module's firmware and install the config file you created using the Network Config tool; otherwise the Samantha Module will not recognize your network.

1. Unplug the USB flash drive from your computer and (making sure the battery is unplugged) plug it into the Samantha Module. Plug in the battery while holding down the red/black button on top of the module. Release the button approximately 3 seconds after the red light appears on the module.

2. The Samantha Module will flash the red light twice in rapid succession and then flash the white repeatedly. You may also see the flash drive's LED flickering (if applicable). This means the Samantha Module is installing the firmware. When it's done, it will flash the red, white, and blue lights in sequence, then repeat the process. Be patient; this process normally takes 20-30 seconds and as

long as 2-3 minutes. When it is done it should have a solid red light and a blinking white light (meaning it is connected to the network).

Remember:

- Don't forget no numbers or other characters in the SSID
- The SSID is case-sensitive in the Samantha Configuration Utility
- Use a flash drive ≥2GB to configure your Samantha
- Having an indicator light on your flash drive is useful to confirm your module is reading the configuration
- If you switch networks on your computer, the FCS needs to be restarted
- Position your WiFi router close to the field for a good connection
- Use cable/zip ties to hold down any excess USB cable so it cannot shake if the robot takes a hit.
- Don't mount the Samantha module on metal backing, as radio signals are affected
- Because the antenna are parallel with the long edge, mount it with the narrow, flat side parallel to the ground
- Make your power switch easily accessible. If the NXT freezes, the motors with continue moving until the main power switch is shut off
- The red button and lights on the Samantha must be easily accessible.

Samantha Router Configuration Guide

Setting up a FCS Router for Teams to Practice with.

These instructions are specific for the Linksys E1200 router.

Connect an Ethernet cable from PORT 1 (do not plug into the Internet port) on the router to the computer that will control the FCS program for your event. Be sure to turn off the wireless network adapter on your computer.

Open an Internet browser window and enter 192.168.1.1 into the URL bar. This is the default address of the router. (If nothing comes up, ensure that you have plugged the Ethernet cable into PORT 1 and not the port labelled "Internet").

The router should display a Welcome screen. Do not install the Connect software. You will instead configure the router manually.

At the bottom of the welcome screen, click on the link that reads "Continue with an open and unsecured network (not recommend)"

The web browser should display another screen that indicates that the network is not secure.

Check the checkbox ("I understand that my network…") and press Continue to proceed with the setup.

The router should prompt you for the router's user name and password. Enter in the default User ID of admin and the default password of admin and then press the "OK" key to login. Once you've logged in, you might need to reload the page "192.168.1.1" in your browser to continue.

The first time you connect to you router you will see a pop-up warning "Using this advanced utility to change your router settings could disable your network". Check the "Do not show me this again" box and then click OK to dismiss it.

Create a secure password. Click on the administration tab to change the admin password and disable Access via Wireless. Save settings by pressing the "Save Settings" button.

After the changes have been successfully saved, hit the “Continue” button and log in with User ID “admin” and the password you just defined.

When the dialog opens, click on the Setup tab, then Basic Setup.

1. Ensure the Internet Connection Type is set to Automatic Configuration - DHCP

2. Change the IP address to 192.168.2.1.

3. Set the starting IP address and the max number of users to 100.

4. Set the time zone appropriately.

5. Critical: Save the settings. A message will appear asking you to “Continue” Click Continue to refresh the router to the new IP address setting.

6. Open an Internet browser (such as Internet Explorer or Firefox) and enter 192.168.2.1 into the address bar to continue configuring the router.

7. Login to the router using “admin” and the password you established.

8. The Wi-Fi Protected Setup is selected by default, so you must click on the wireless tab and select the manual setup bullet:

The E1200 is a 2.4GHz router only. You do not need to disable the 5GHz band for the E1200. You must, however, configure the 2.4GHz band settings for the E1200:

1. Select: “Wireless-B only” from the drop down menu next to Network Mode.

2. Enter FTC_FIELD into the Network Name (SSID): box.

3. Note: Generally, channels 5 or 8 are good choices as they are least used channels for public WiFi networks.

4. Click on Save Settings

5. Hit the “Continue” button and log in with “admin” and the password you defined.

Setup Security

Click on the wireless Security tab under the main Wireless tab.

1. Change the security mode to WPA2 Personal.

2. Change the passphrase to something unique and give it to the software inspectors and field tech advisor. Do not share it with the teams. Passphrase must be at least 8 characters and contain no spaces. For extra security, use at least one upper and one lower case alpha character, at least one number and one special character such as !@#$%^&*.

3. Click on Save Settings

4. Reboot the router one more time by returning to the Setup tab, scrolling to the bottom of the screen and clicking on “reboot”.

When the router configuration file returns after rebooting, your FTC_FIELD router is ready to use.

CRITICAL STEP: Write down these settings and keep them safe for reference.

Robot Wiring

TETRIX Push Bot Wiring Layout

NXT Connections

Samantha Module. Plugs into NXT USB connection. Powered via the servo controller.

LEGO NXT with Rechargeable Battery

Hitechnic light sensor plugs in to port 2 of the NXT

DC Motor Controller plugs into port 1 of the NXT.

Servo Motor Controller plugs into the data port of the DC motor controller. The power comes from the DC motor controller and supplies the Samantha with power.

Turning 'ON' Bluetooth on the NXT

To test your ROBOTC or LabVIEW® code, Bluetooth will need to be enabled on the NXT.

Follow the steps below to enable Bluetooth on the NXT.

Step 1 – Turn on the NXT - Press the orange Button.

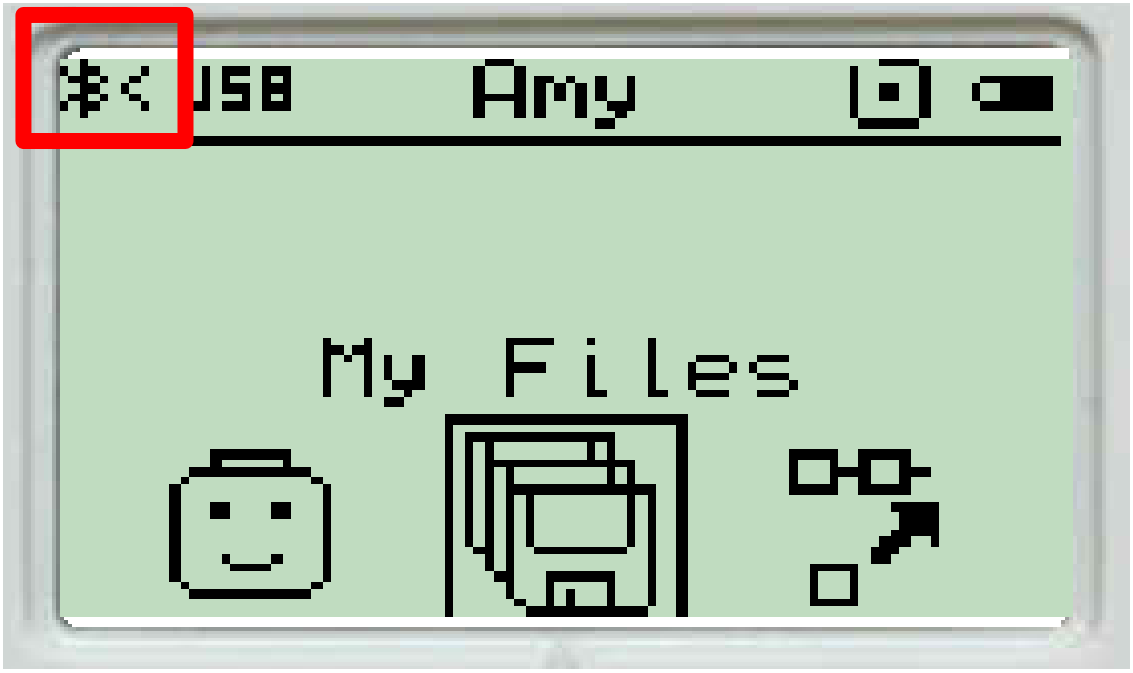
Step 2 – Scroll across to the Bluetooth Menu – Use Side Buttons. Press Orange Button to select 'Bluetooth Menu'

Step 3 – Scroll across to the 'On/Off' Menu and press Orange Button to select.

Step 4 – Press Orange Button to select 'On'.

Step 5 – Reselect Bluetooth Menu and Scroll across to ‘Visibility’. Use Orange Button to select ‘Visible’. Click on Dark Grey Button to run to Menu.

Step 6 – Check that Bluetooth and Visibility is Turned on as per image below.



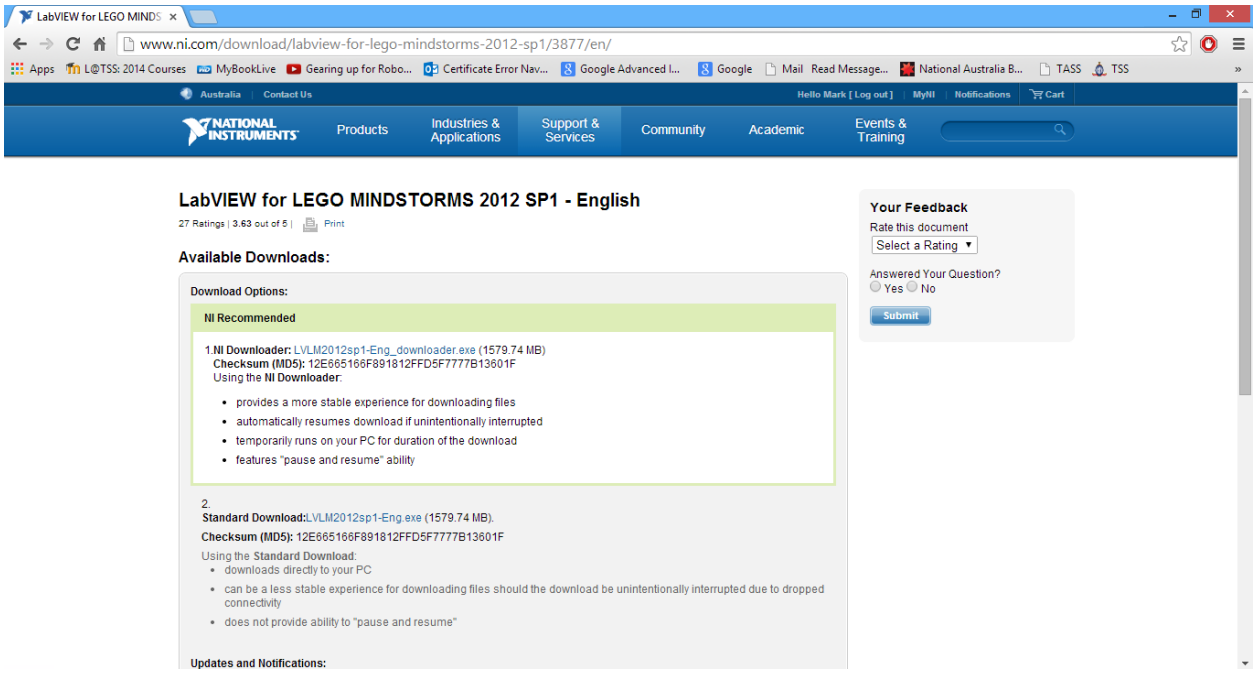
Programming the Robot with Labview

Setting up LabVIEW for LEGO MINDSTORMS 2012.

Install LabVIEW for LEGO MINDSTORMS 2012 from the DVD included in your FTC Competition Kit or from the URL below

http://www.ni.com/download/labview-for-lego-mindstorms-2012-sp1/3877/en/

Use the 'NI Downloader: LVLM2012sp1-Eng_downloader.exe (1579.74 MB)' version



.

There is a free trial license period but an FTC Team has one licensed version of the software included in their Competition Pack.

Please Note: THE SOFTWARE MAY ONLY BE USED FOR PURPOSES OF A *FIRST* COMPETITION AND NOT FOR ANY OTHER PURPOSE, including, without limitation, use in a classroom or lab, research, professional, commercial, or industrial purposes. You hereby understand and agree that your license will automatically expire upon the conclusion or termination of your membership in a Team or should you stop being a Mentor or are no longer connected with FIRST; upon any such expiration, you must promptly uninstall all copies of the SOFTWARE from your applicable computer(s).

In addition to the LabVIEW for LEGO MINDSTORMS 2012 software, FTC teams will need to install the LabVIEW Mindstorms Competition Toolkit for FIRST Tech Challenge. The 2013-2014 can be found here

https://decibel.ni.com/content/docs/DOC-17997

Use a web search engine for find the current version of 'LabVIEW Software Setup for *FIRST* Tech Challenge' and install.

Install the MINDSTORMS Competition Toolkit.

2. Download and Install The MINDSTORMS Competition Toolkit 2013 - 2014 (MCT)

Updated 11/6: MCT 2013-2014 now includes NXT Module 2012 SP1 f3 patch

Download and Install the MCT AFTER you Install LabVIEW for LEGO MINDSTORMS 2012

MINDSTORMS Competition Toolkit (Windows)

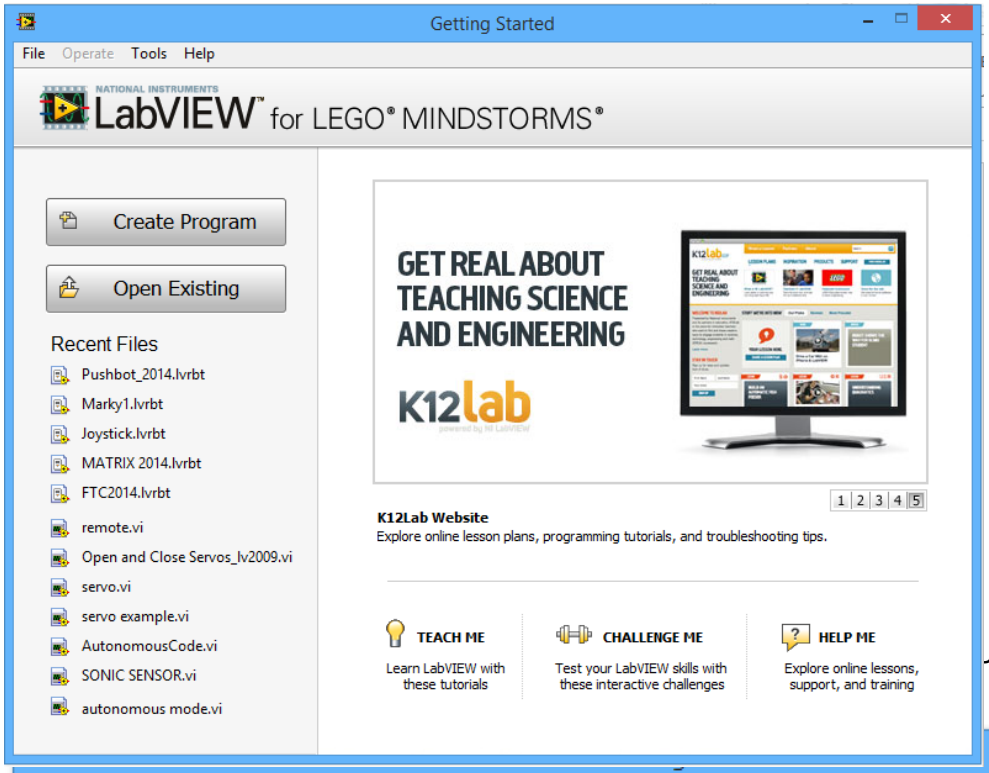
MINDSTORMS Competition Toolkit (Mac OS X)

Note: If you are having trouble with code generation or MATRIX motor support, please reinstall the MCT. The installer was updated on 9/10 to fix an install issue.

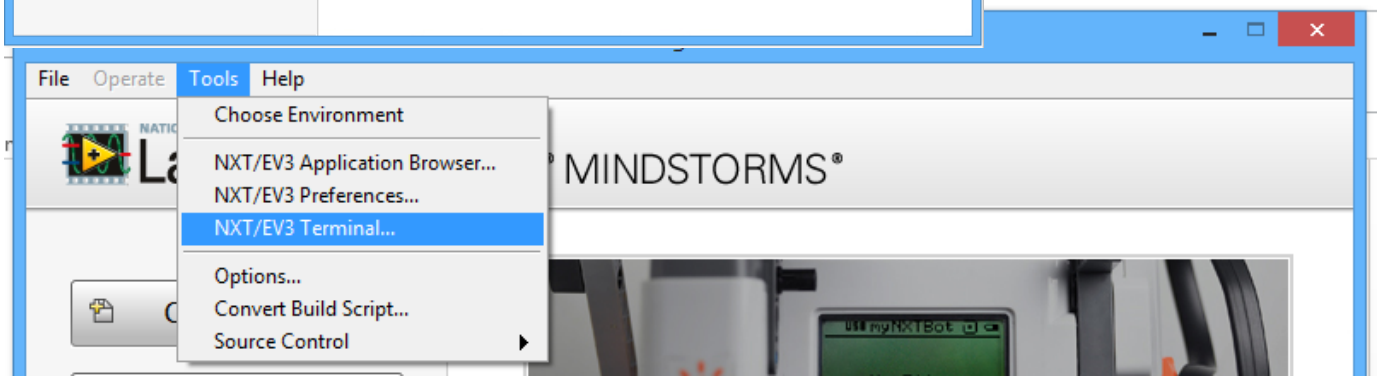
Contents

- NXT Fantom driver
- Samantha Module Support
- LVLM 2012 patches & updates
- MATRIX Motor Support

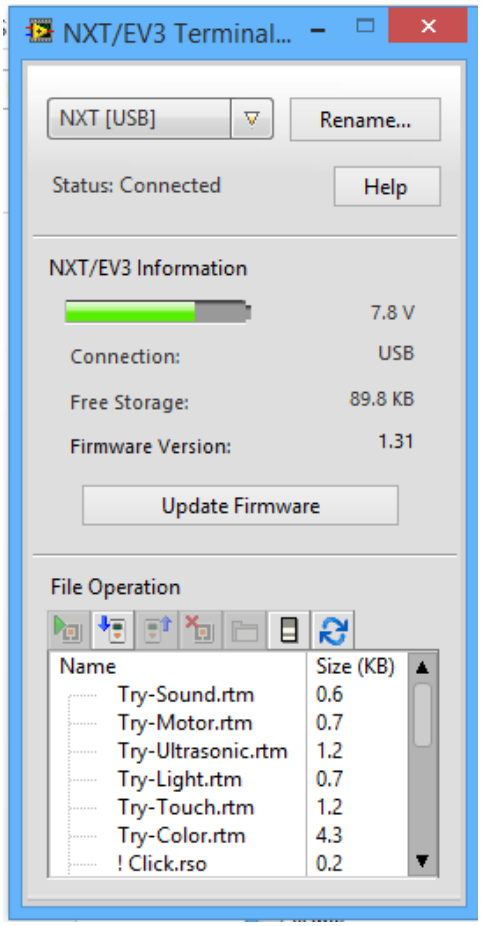
Downloading the LabVIEW Firmware' on an NXT.

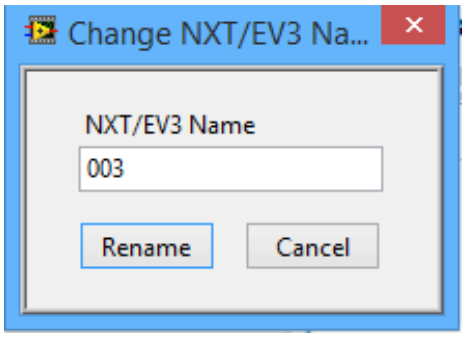


USB cable and turn the

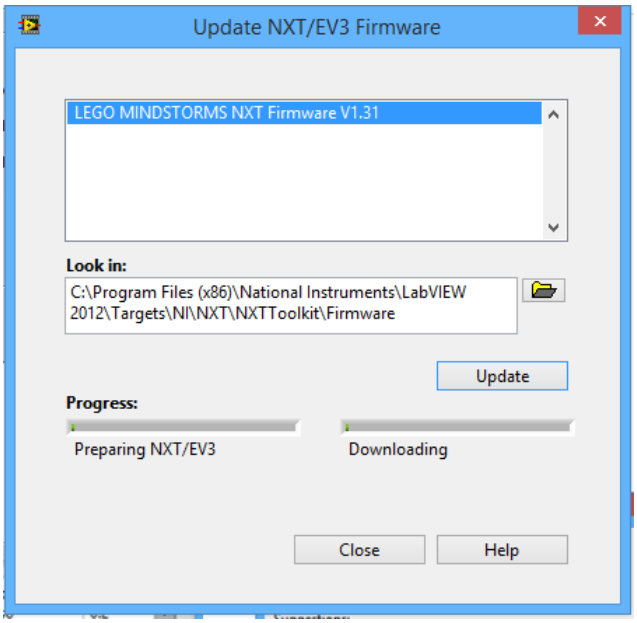


Select ‘Tools’ and ‘NXT/EV3 Terminal’.



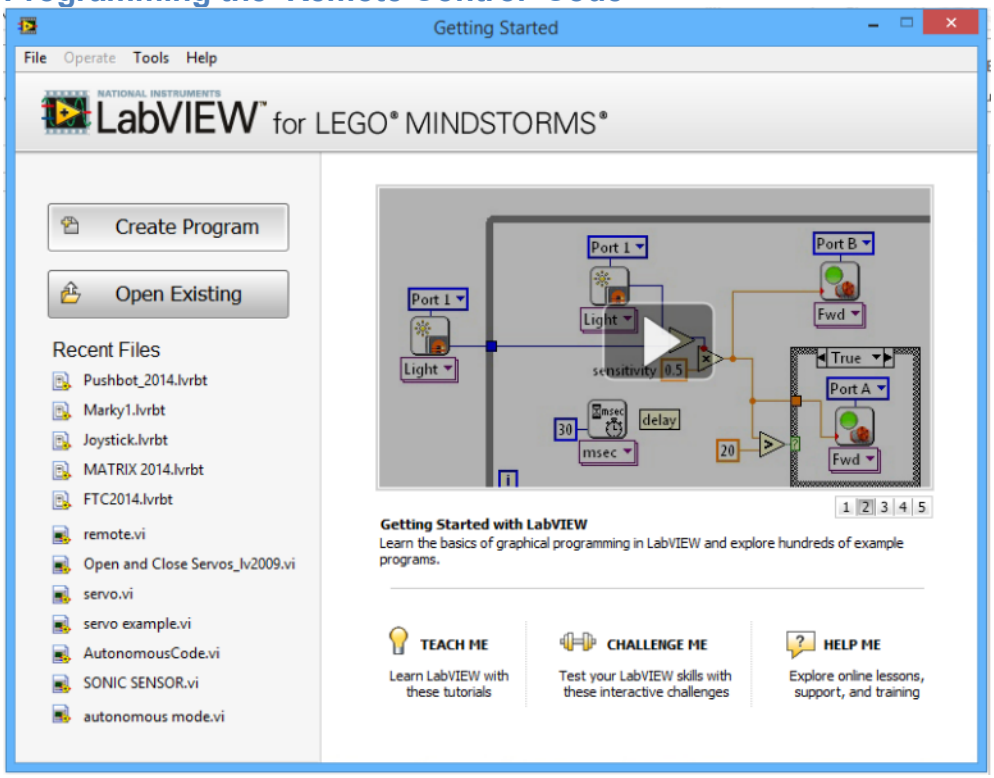


If an NXT has been attached via the USB cable, CLICK ‘Rename’ and enter your FTC team number. If this is the first time you have used Labview you must update the Labview Firmware on the NXT.

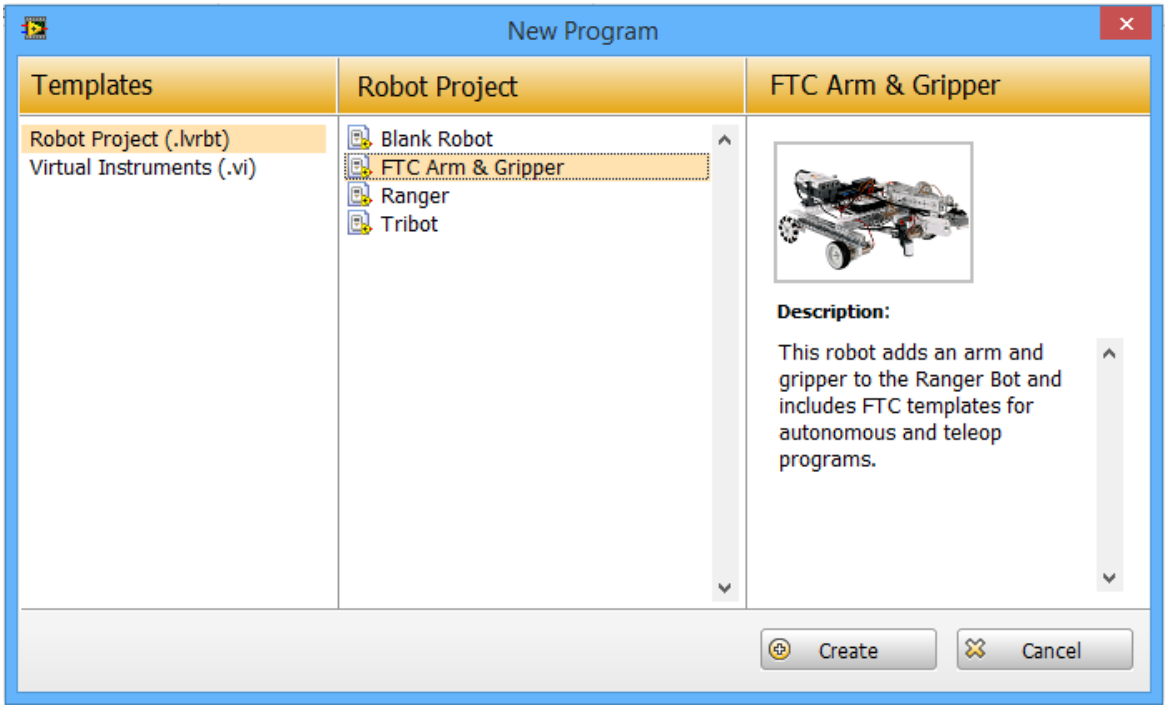


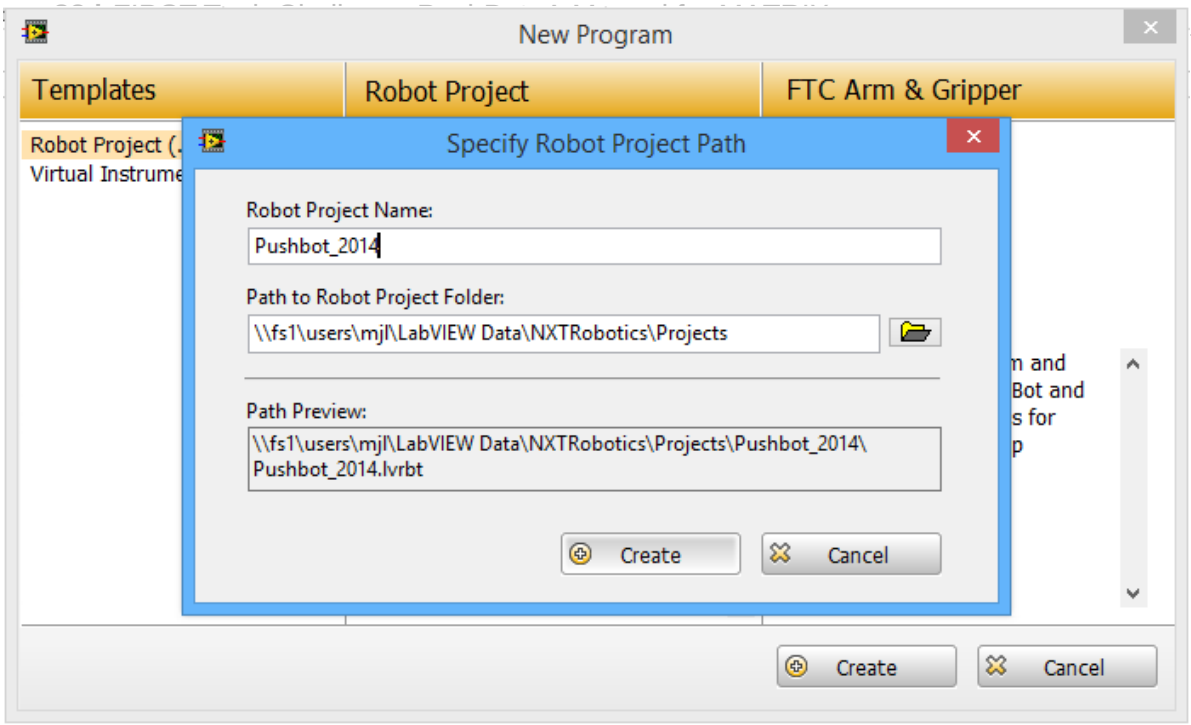
Select ‘Update’ and wait for the Firmware to download and select ‘Close’.

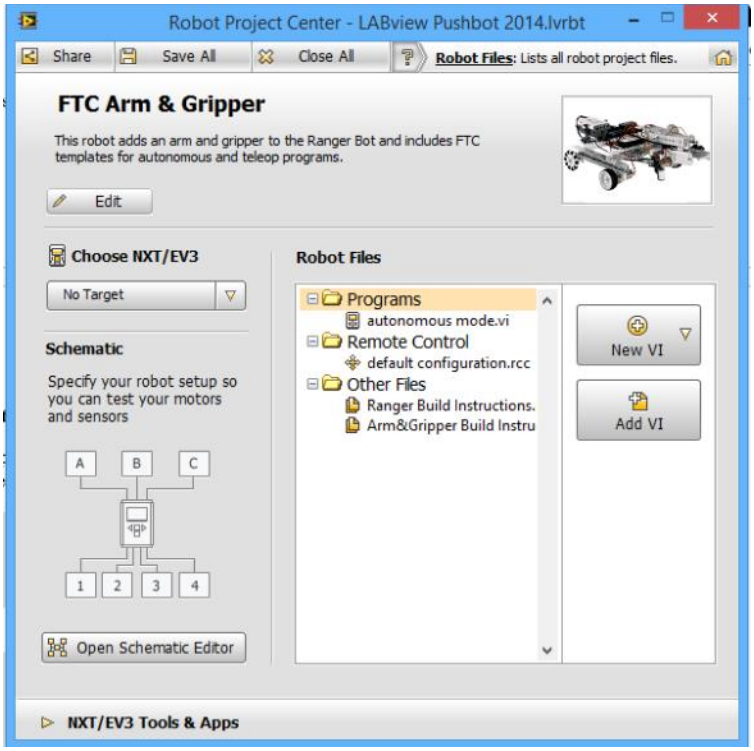
Programming the 'Remote Control' Code



Select 'Create Program'.





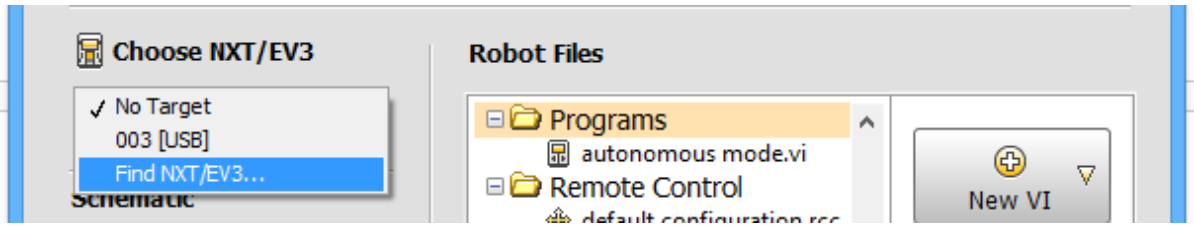


This is the Robot Project Center. All the Remote Control and Autonomous Code is stored here.

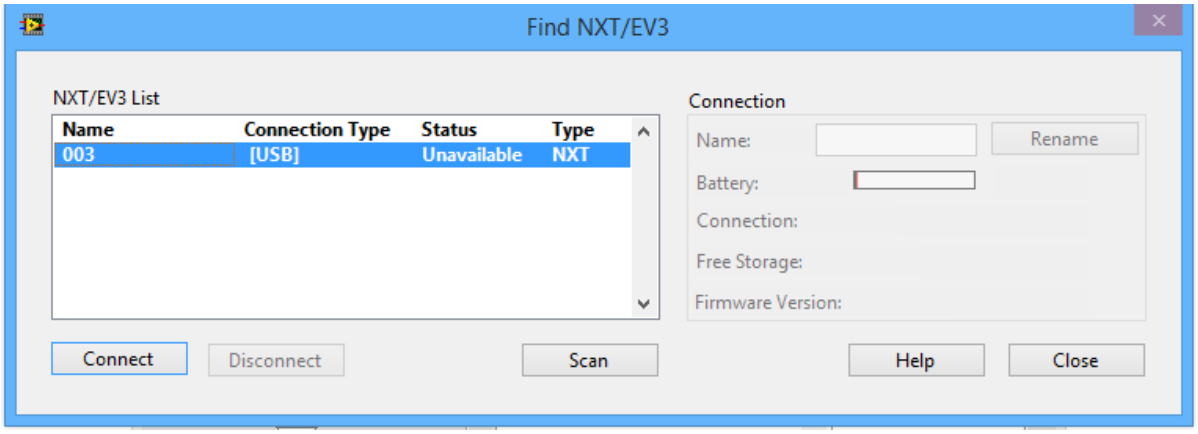
This is where the Robot communication and Inputs and Outputs are setup.

Setting Up Bluetooth Communications.

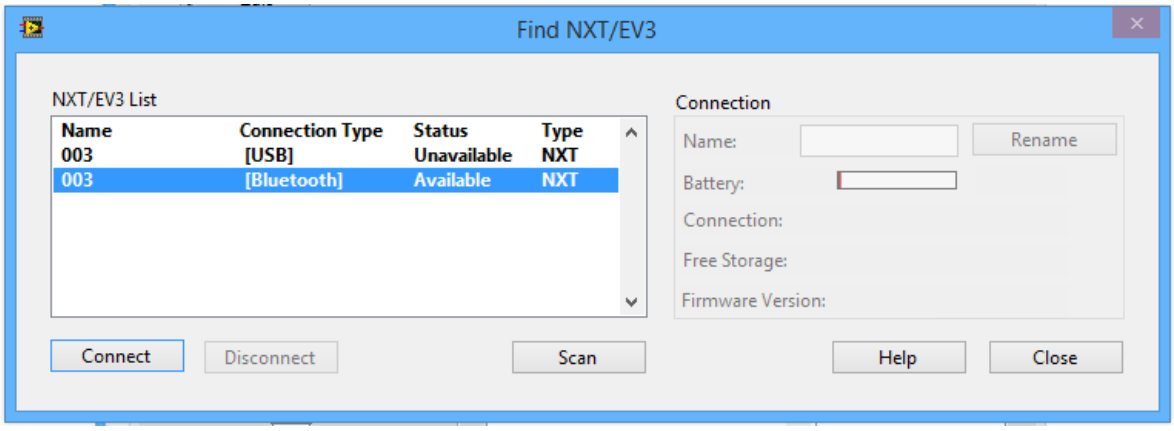
Unplug the USB cable from the NXT and ensure Bluetooth and Visible has been selected on the NXT. Use the Drop down Menu under 'Choose NXT/EV3' to select 'Find NXT/EV3'.



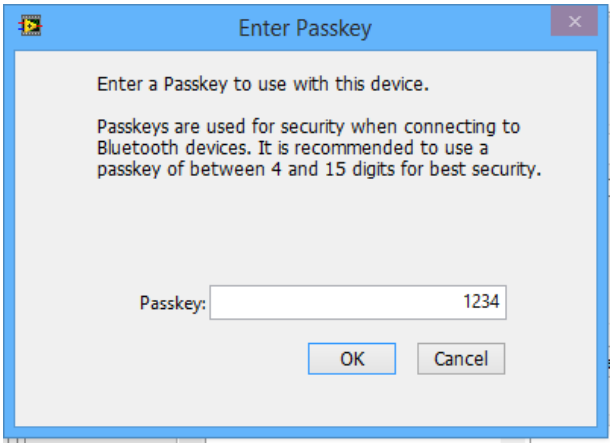
Click on 'Scan'



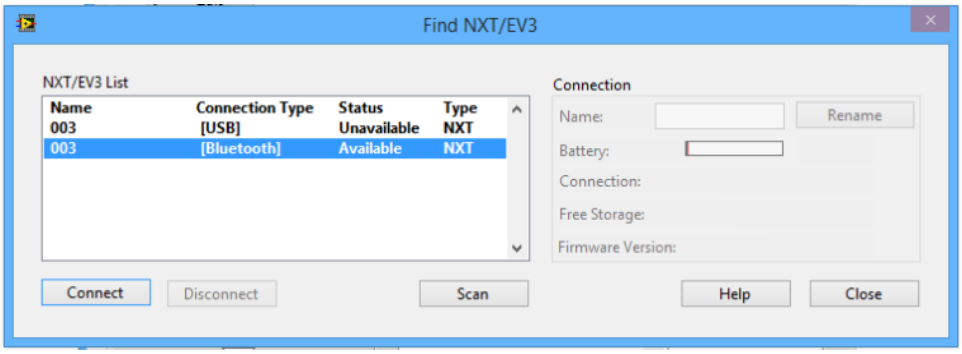
Allow access through Windows Firewall if requested.



Select the desired NXT and select 'Connect'



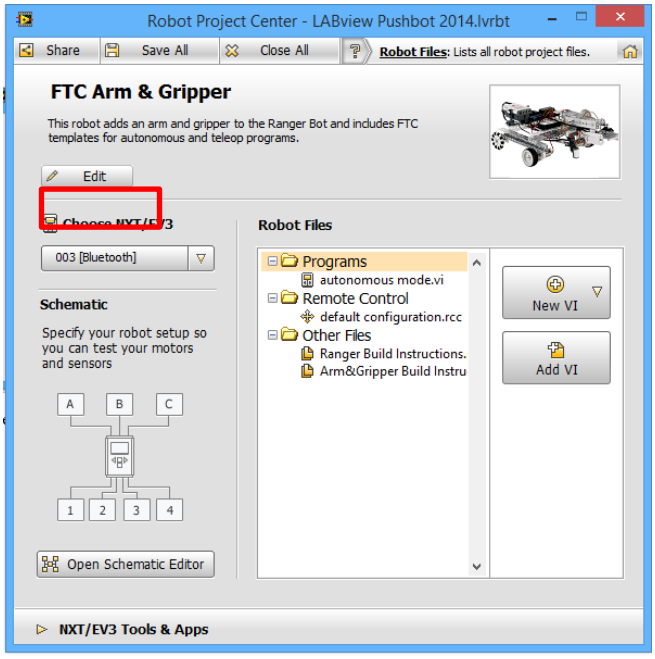
Enter NXT passcode (normally 1234) and select 'OK'.



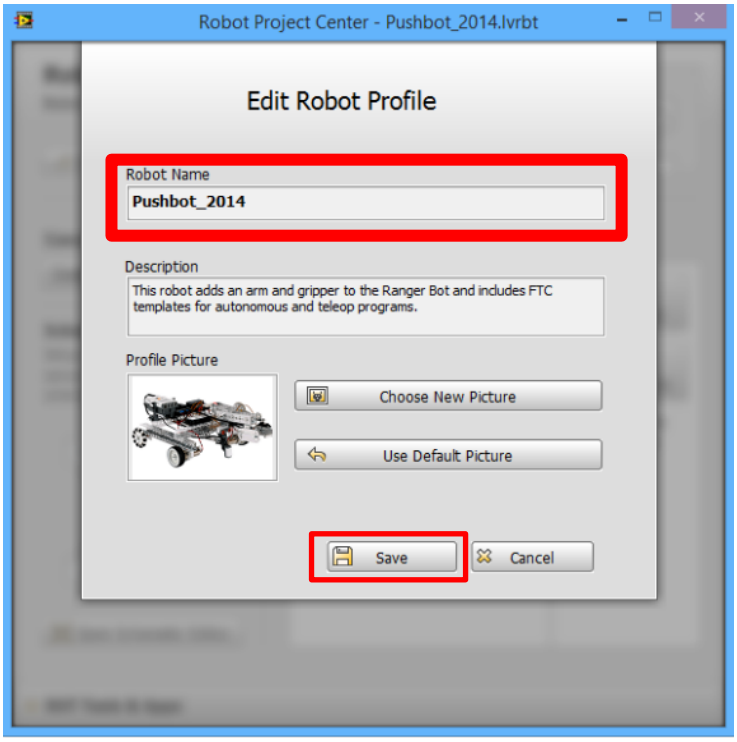
Then ‘close’.

Ensure your robot name is selected under ‘Choose NXT/EV3’.

Editing the Robot’s Name



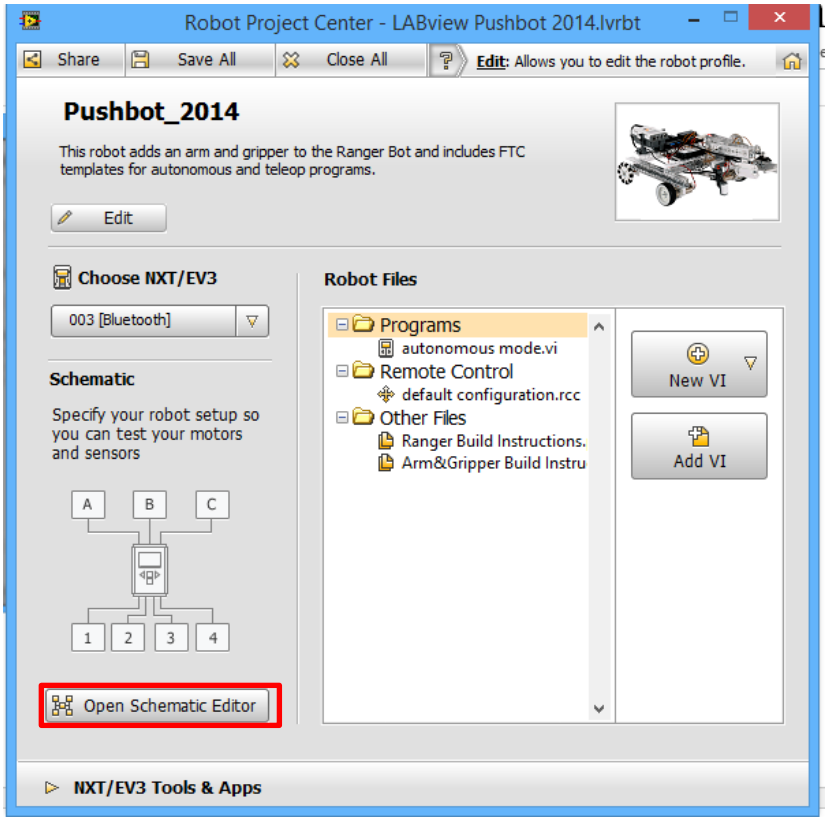
Select ‘Edit’



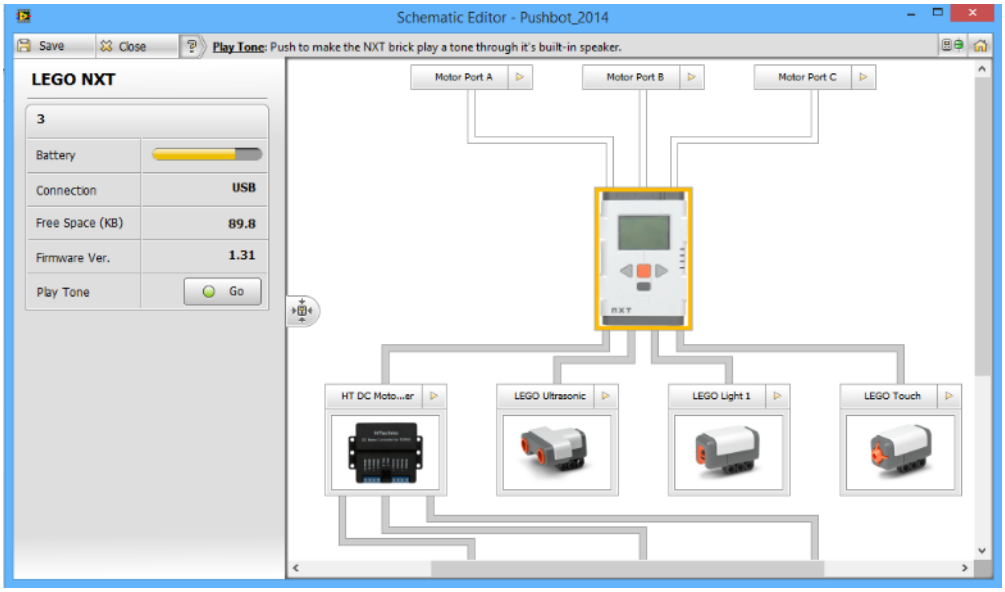
Enter a new name for the Robot. A new image for the Robot can be chosen under 'Choose New Picture'. Select 'Save' once all edits have been made.

Configuring the Robot's Inputs and Outputs.

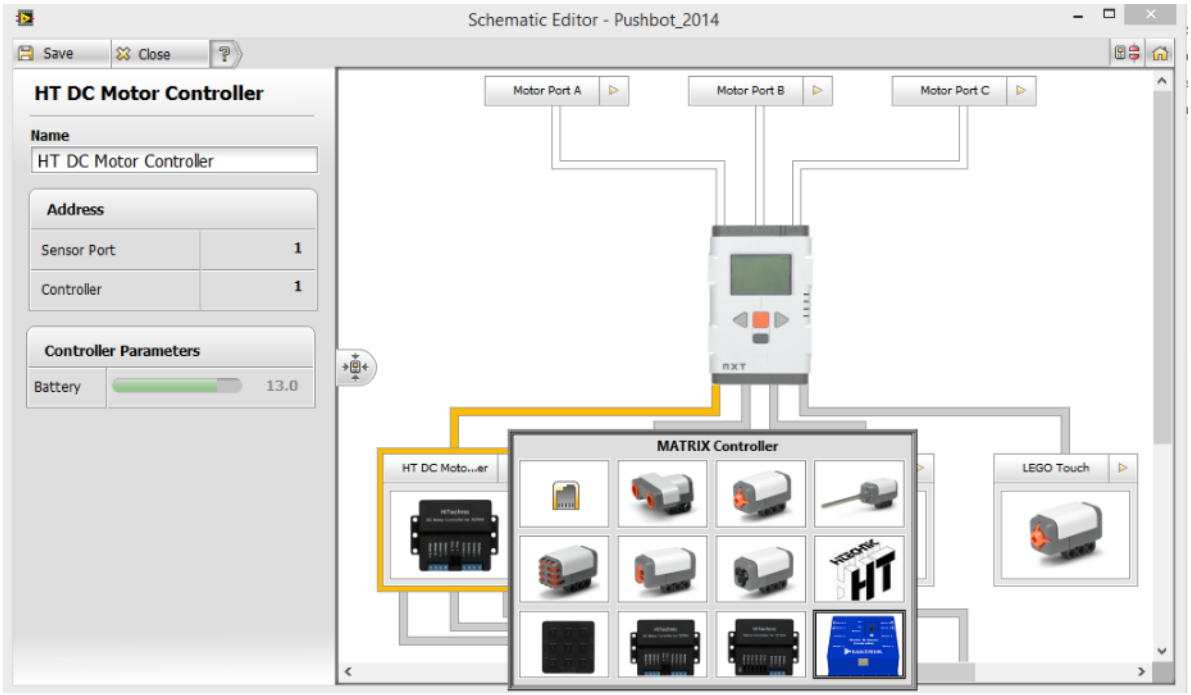
All the robot's sensors, controllers, motors and servos have to laid out in the Schematic Editor.



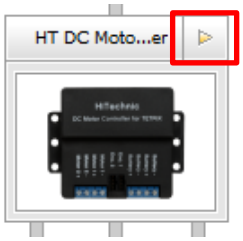
Select 'Open Schematic Editor'



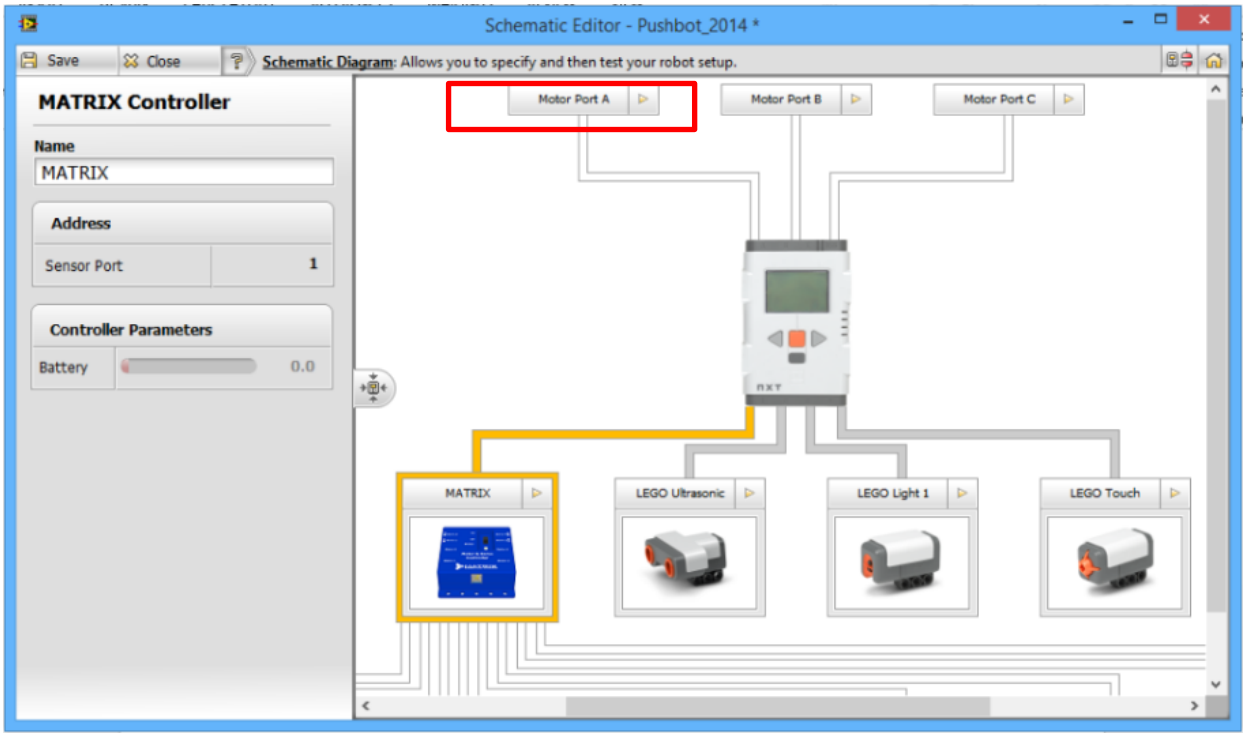
This is the Schematic Editor Window. The Controllers, Sensors, Motors and Servos needed to be attached and named.



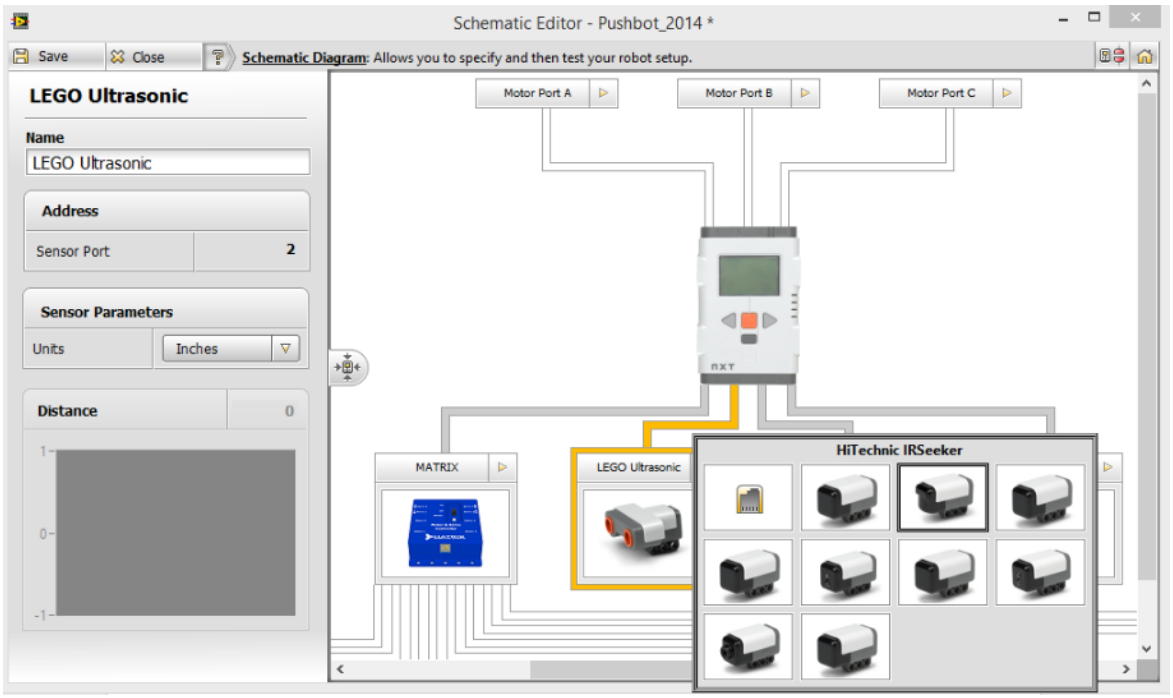
Select HT DC Motor Controller as indicated above and click on the ‘side arrow’ as shown below. Select the MATRIX Controller.



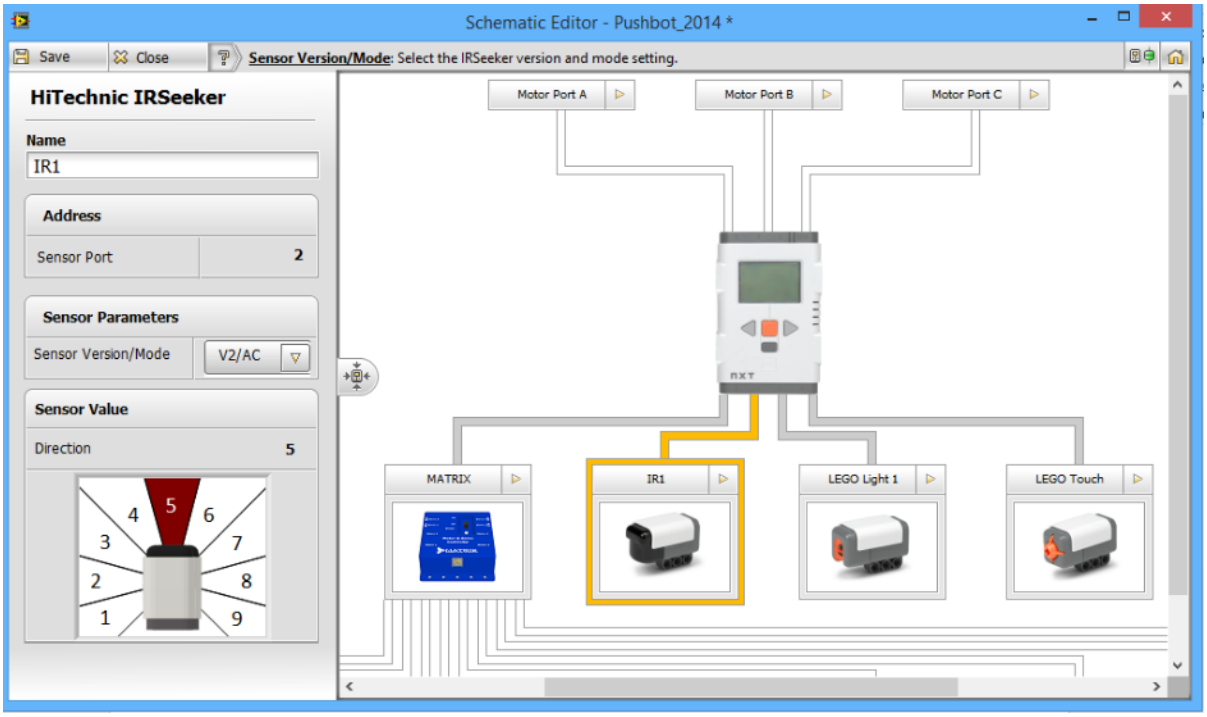
Label the MATRIX Controller as ‘MATRIX’



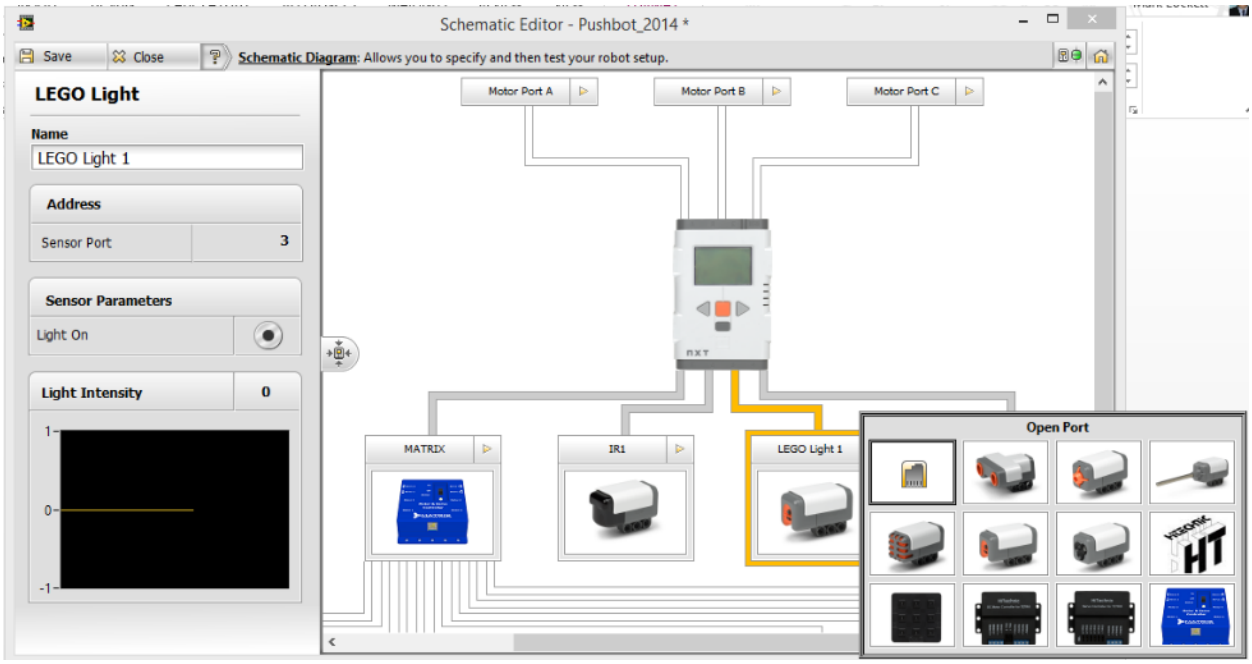
Select the LEGO Ultrasonic Sensor and click on the 'HITECHNIC HT' box as shown above.



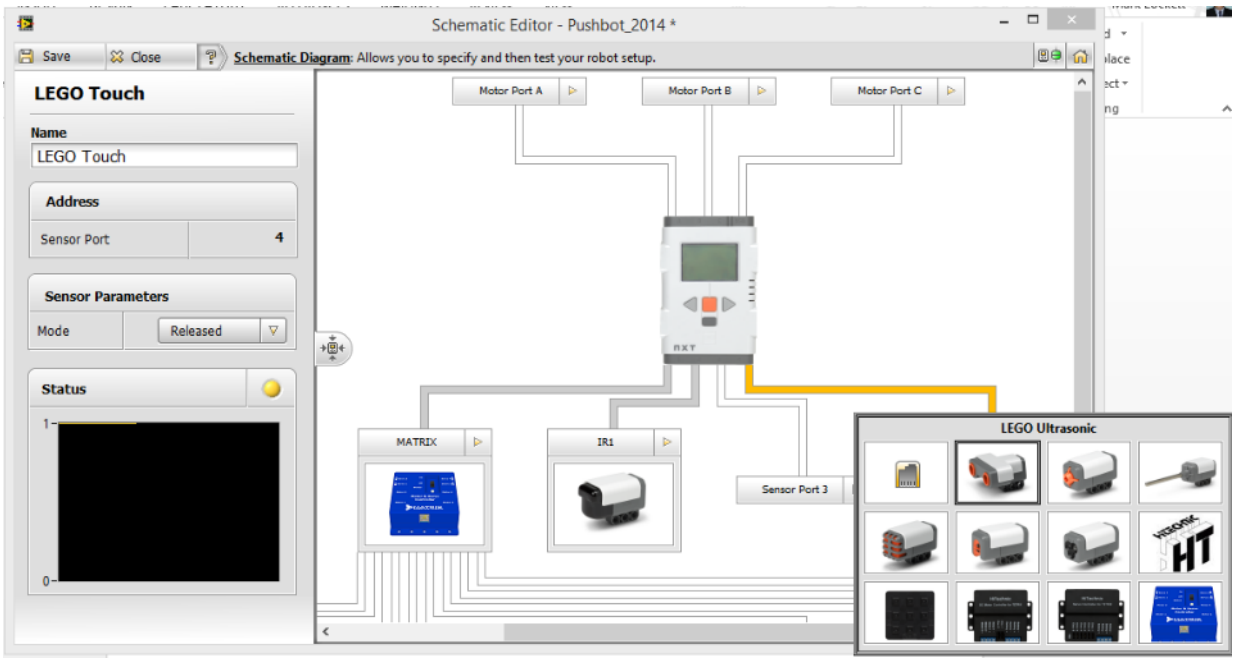
Select the 'HiTechnic IRSeeker'.



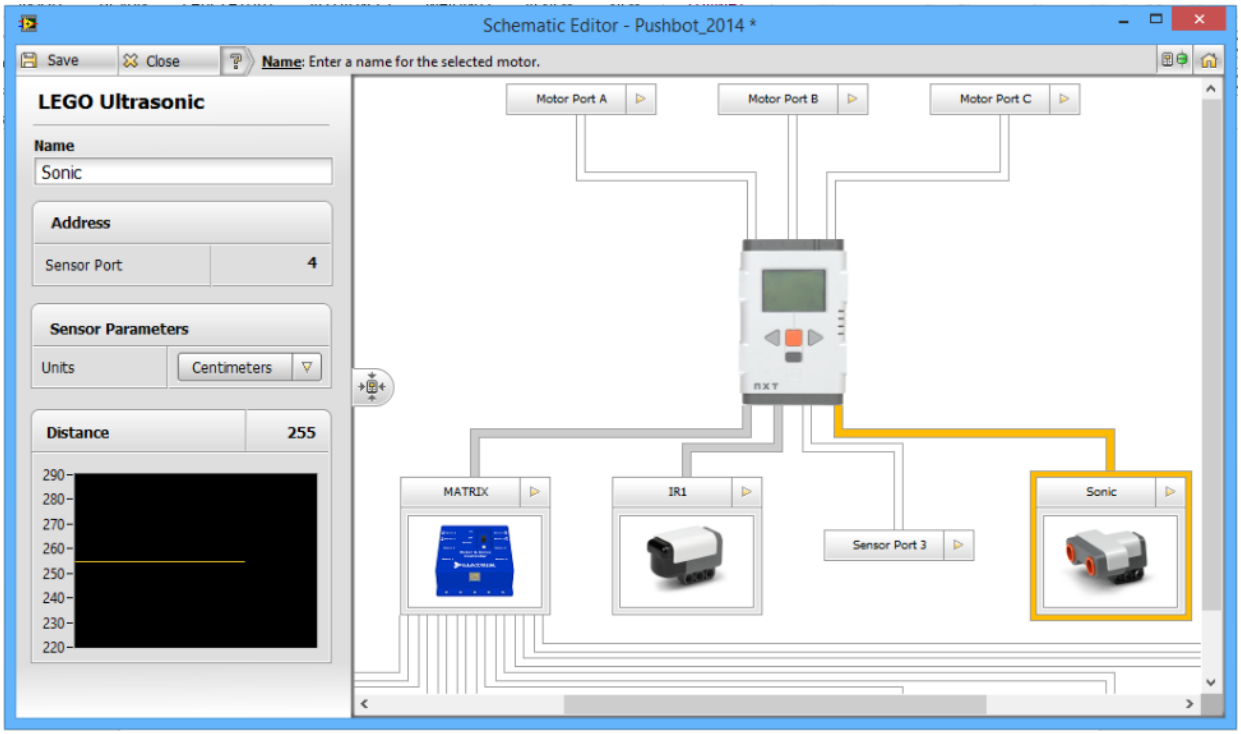
Rename the Sensor as ‘IR1’



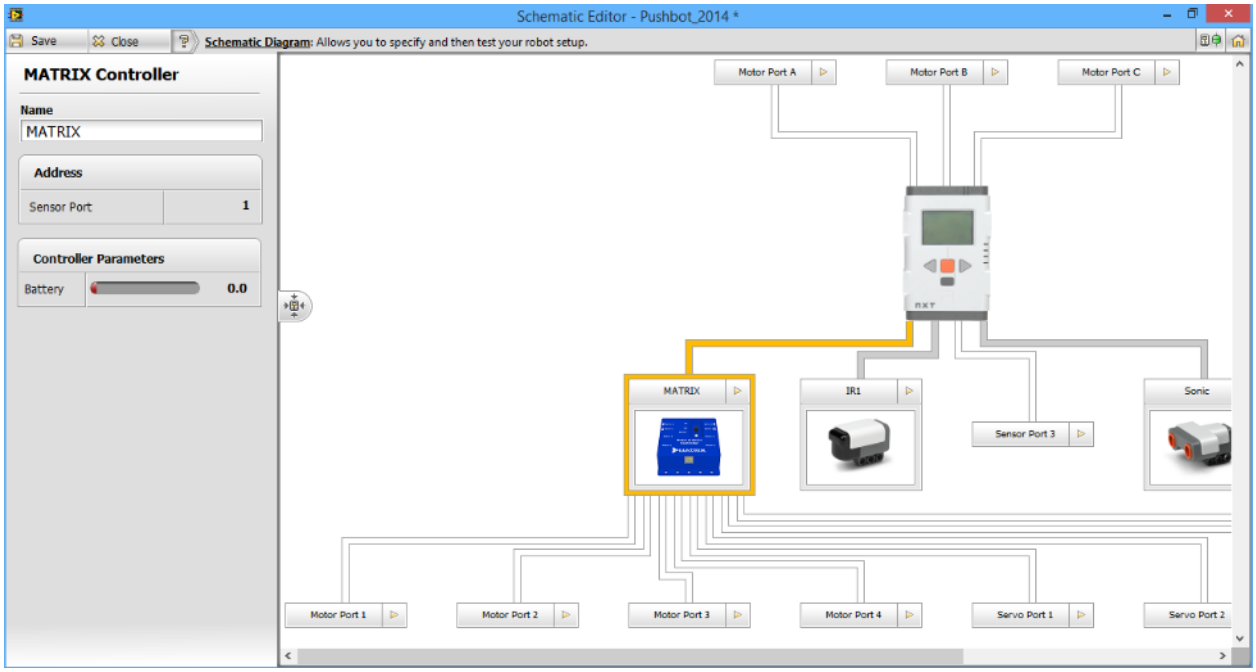
Select the LEGO Light 1 sensor and select ‘Open Port’.



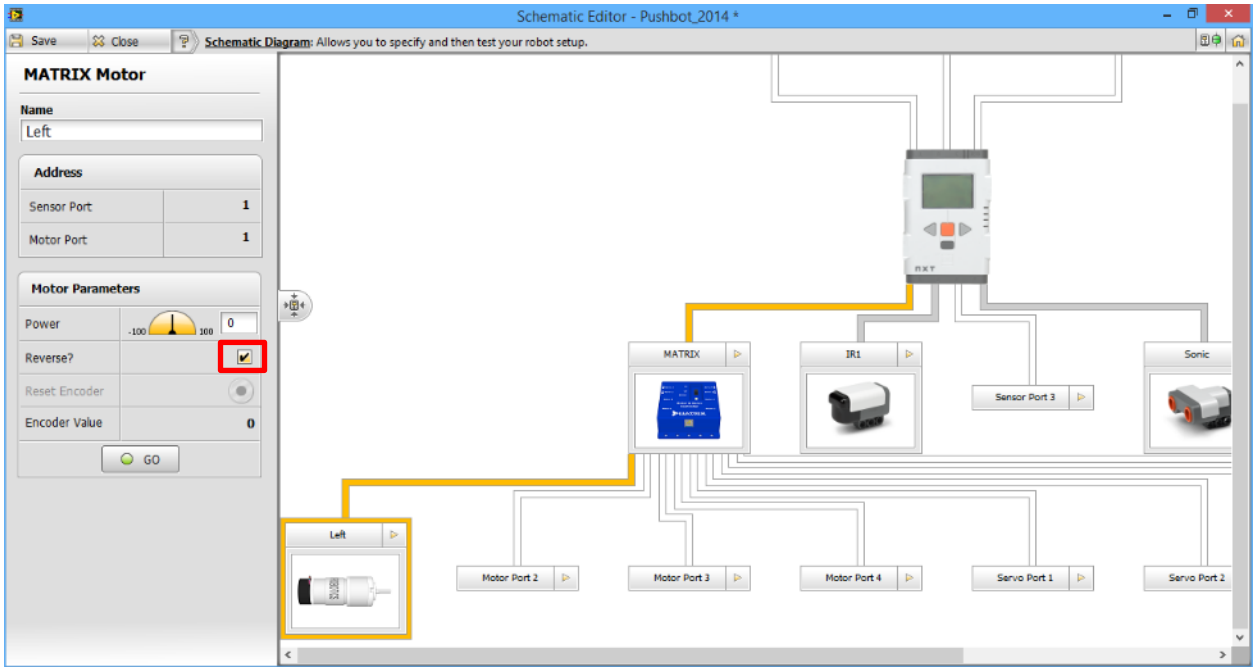
Select the LEGO Touch Sensor and choose the LEGO Ultrasonic sensor instead.



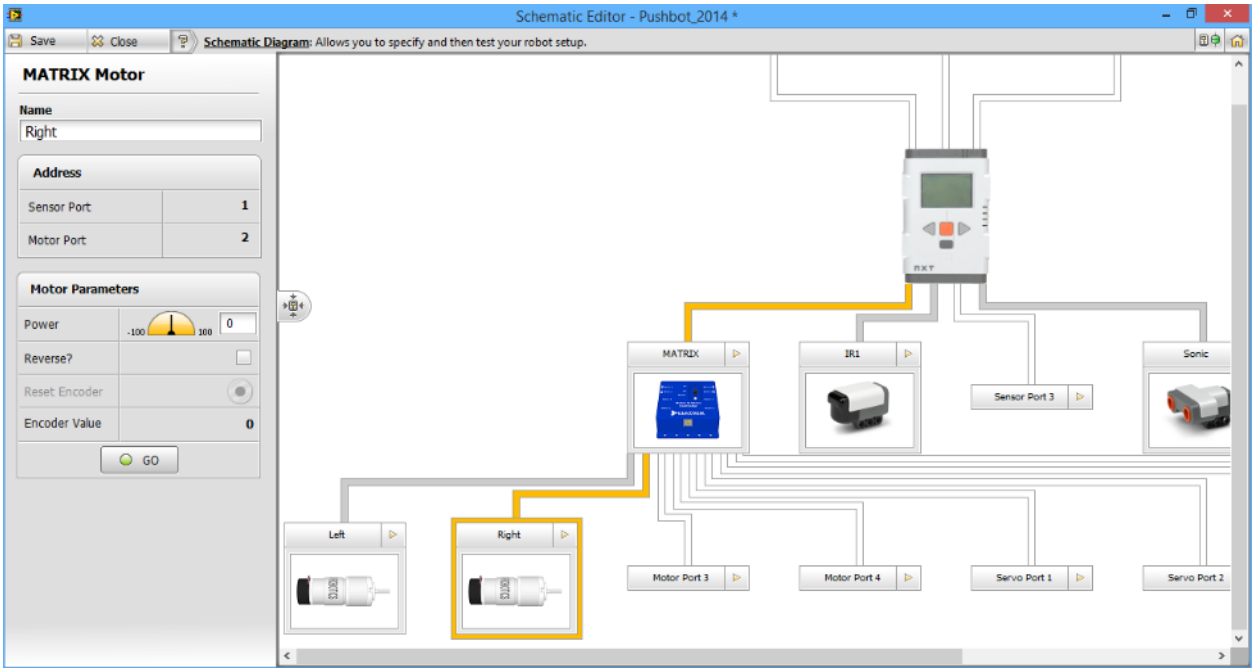
Rename the LEGO Ultrasonic sensor as ‘Sonic’.



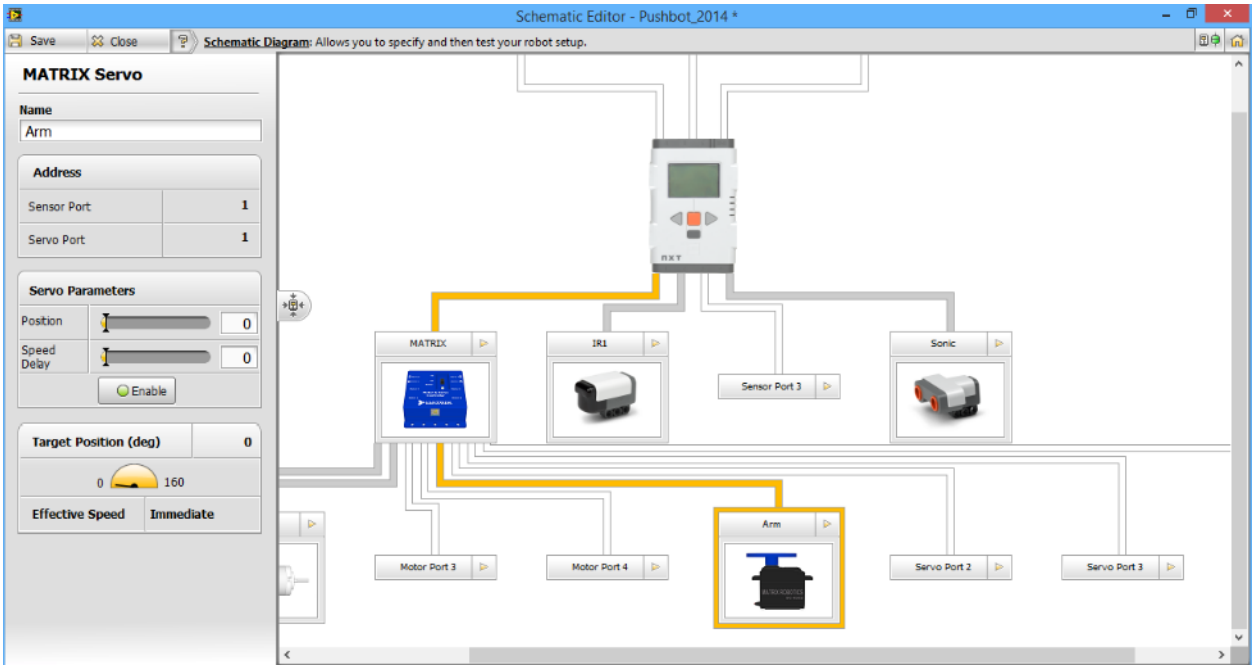
Select the MATRIX Controller and adjust the window so the Motor and Servo Ports are visible.



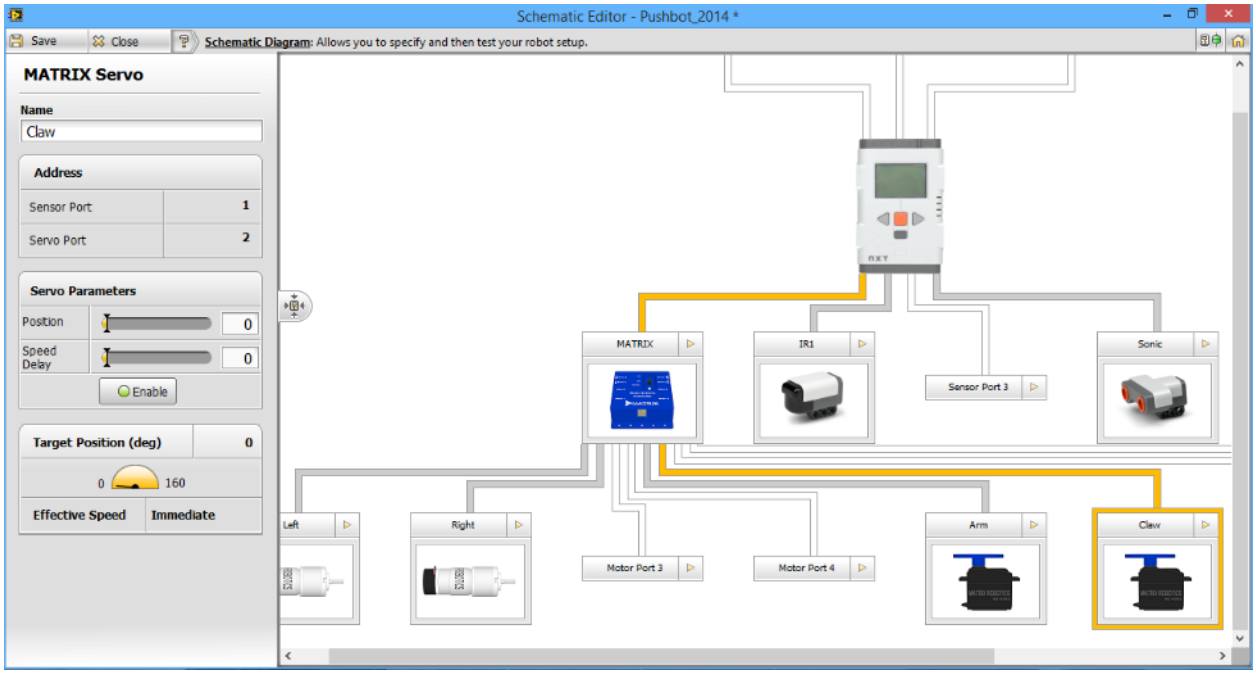
Select ‘Motor Port 1’ and change to MATRIX Motor. Rename as ‘Left’. Click on the Reverse box.



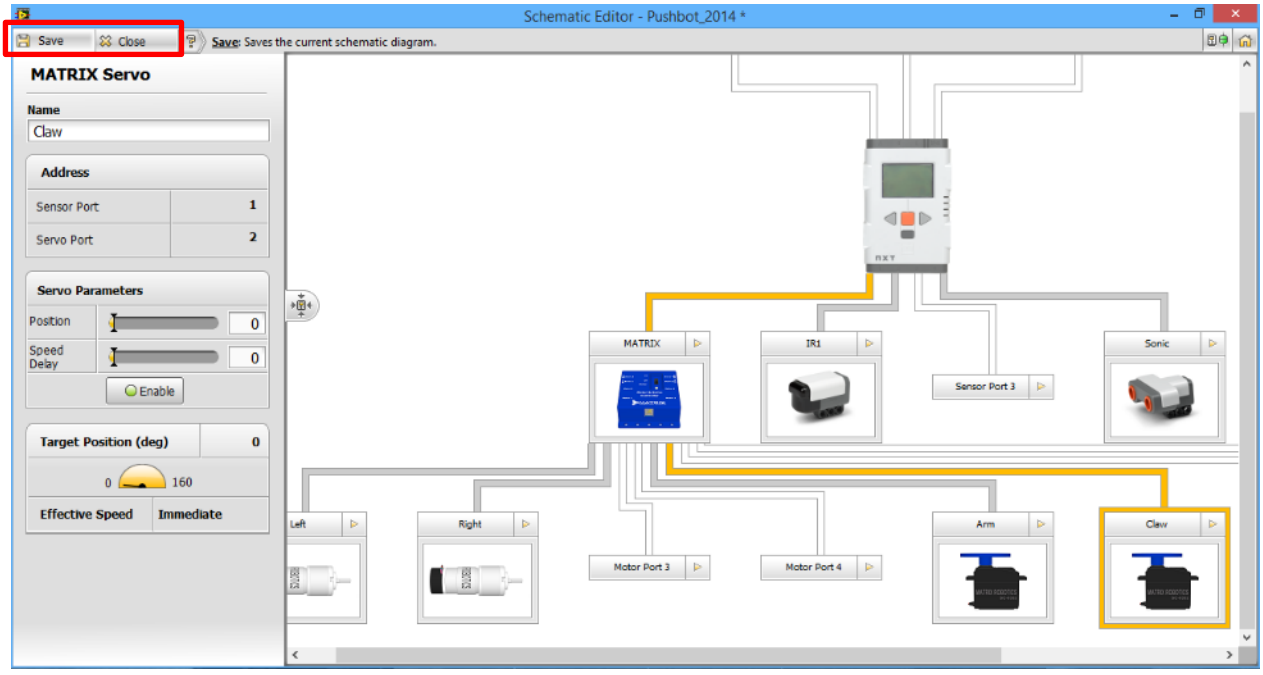
Select ‘Motor Port 2’ and change to MATRIX Motor. Rename as ‘Right’.



Select Servo Port 1 and change to MATRIX Servo. Rename as ‘Arm’



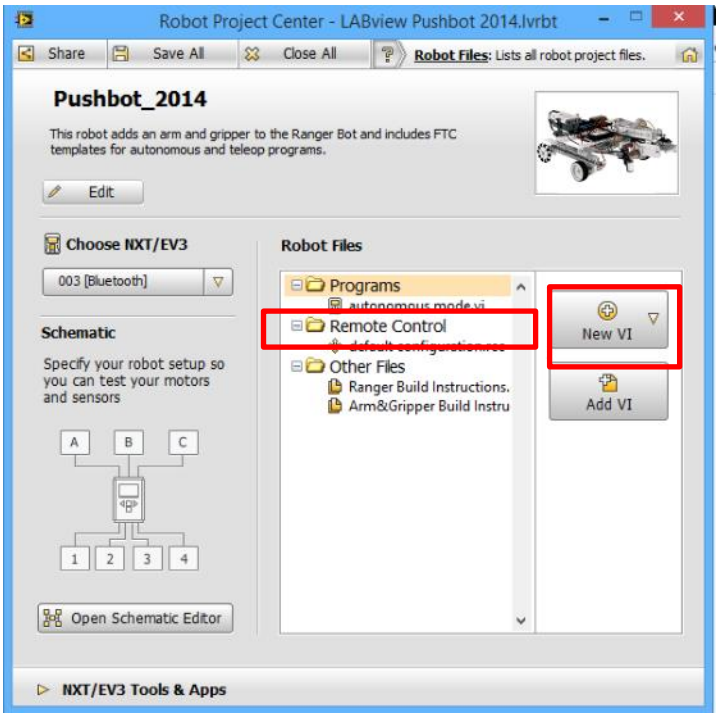
Select Servo Port 2 and change to MATRIX Servo. Rename as ‘Claw’



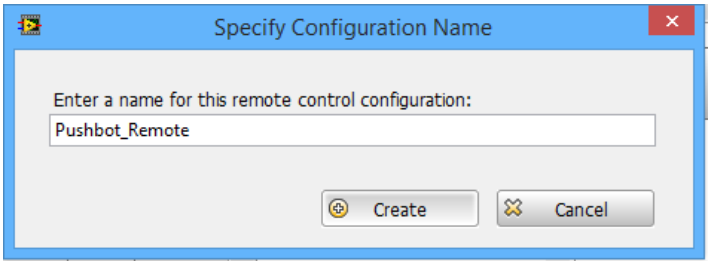
‘Save’ and ‘Close the Schematic Editor.

Driving Remote Control Setup

The 2 minute Remote Control Period Setup and Code will now be explained. Connect two USB Logitech F210 Joypads. Ensure the Joypads are switched to 'D' on the back.

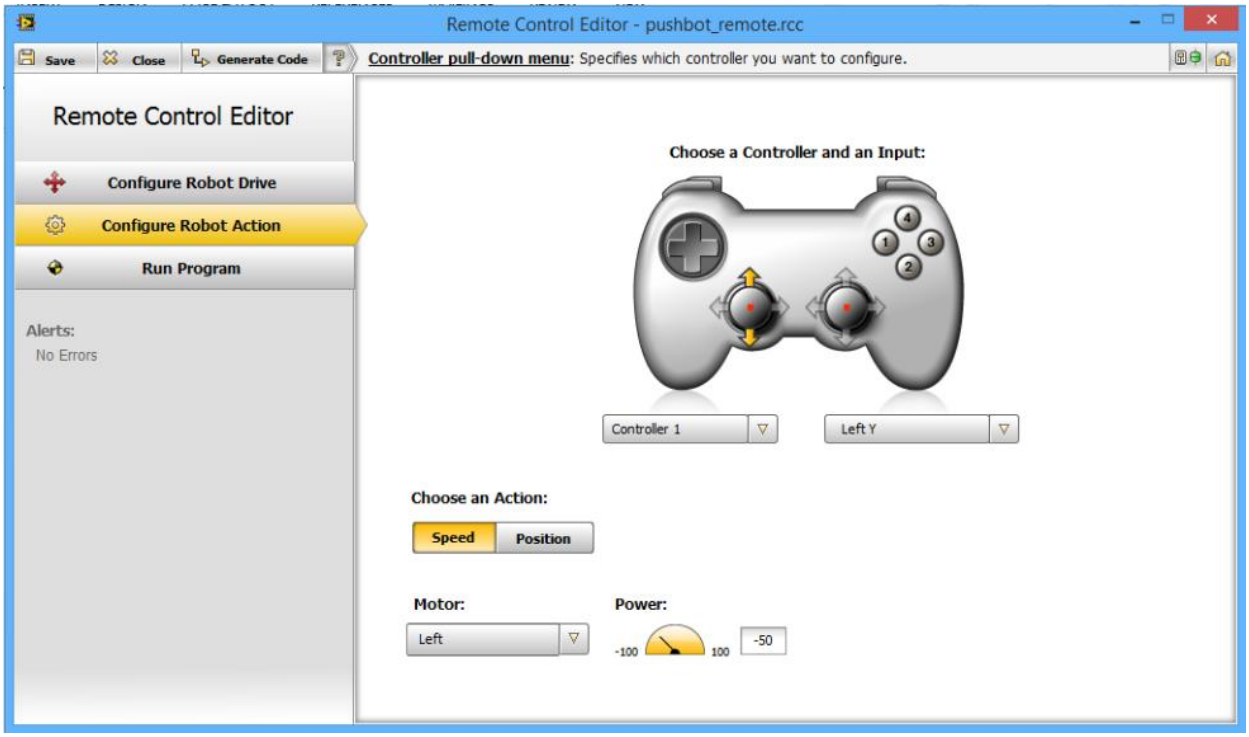


Select 'Remote Control' and 'New'.

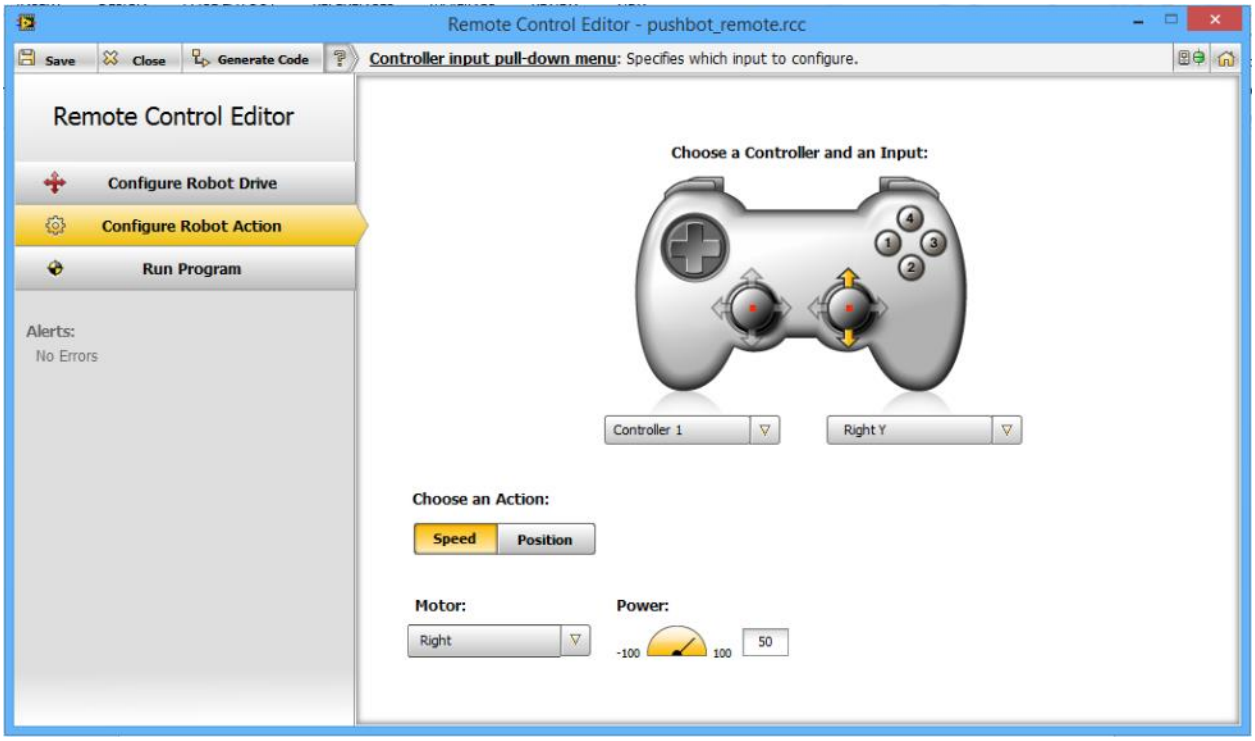


Enter a name for the Remote Control Configuration and select 'Create'.

The Robot Driving and Servo Control will be configured through the ‘Configure Robot Action’ setup. Leave the ‘Configure Robot Drive’ as ‘None’.



Select Joypad Controller 1 and press the Left Joystick upwards. This will select the ‘Left Y’ setting. Select ‘Speed’ and change the Motor to ‘Left’. Set the Power to -50.



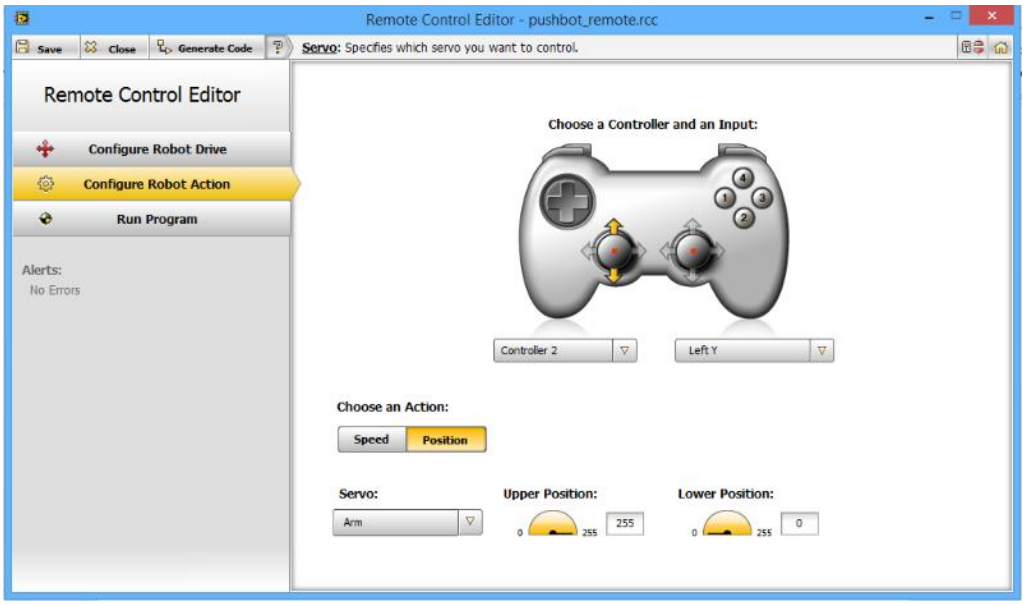
Select Joypad Controller 1 and press the Right Joystick upwards. This will select the 'Right Y' setting. Select 'Speed' and change the Motor to 'Right'. Set the Power to 50.

Testing the Driving Remote Control Code

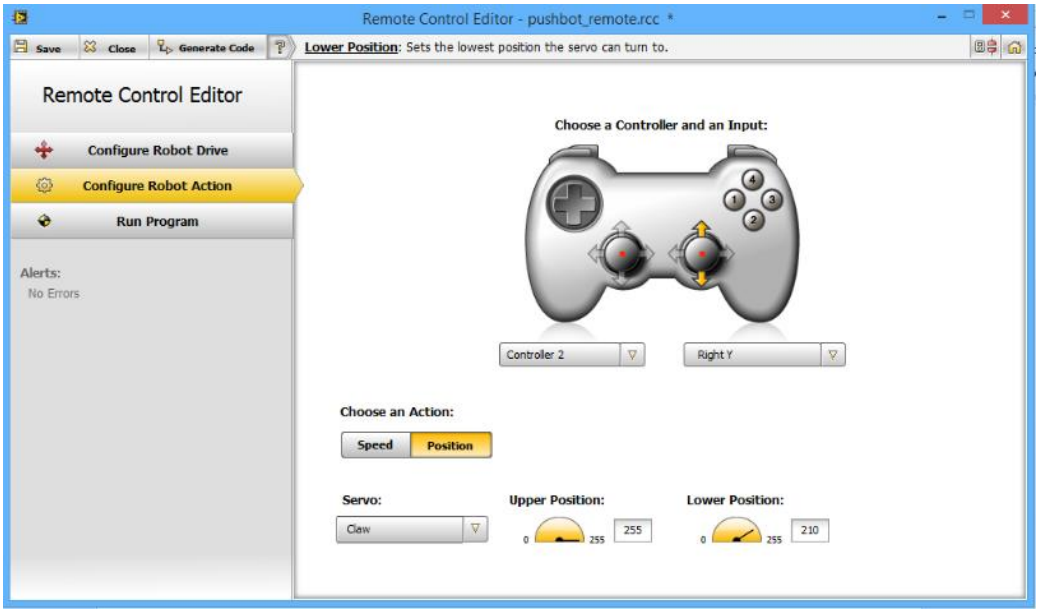
Save the Configuration by clicking on 'Save' in the Top left-hand Corner. Select 'Run Program' and 'Start Test'. Joypad 1 Left and Right Joysticks can now be used to drive the robot around. Both Joysticks UP is Forward, Both Joysticks Down is Backwards and One Joystick UP and the other one DOWN is Turn.

Servo Remote Control Settings

To control the Arm and Claw the Servos need to be configured. The second Joypad will be used to control the Arm and claw.

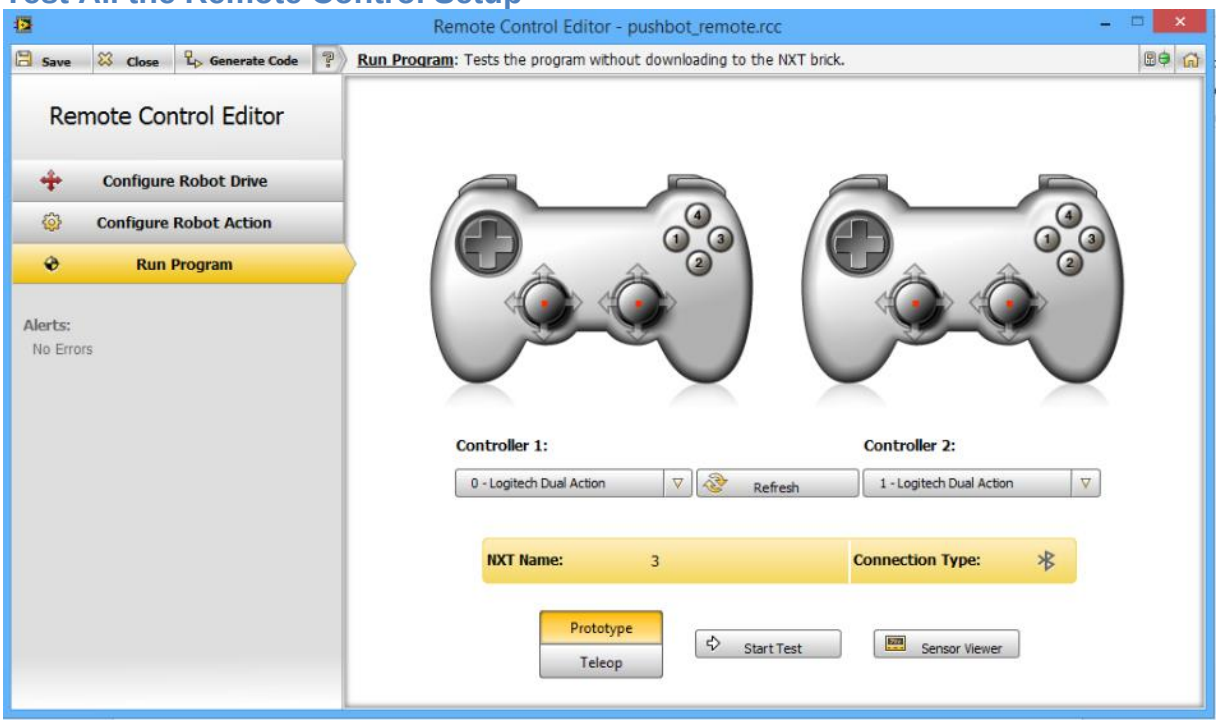


Select Joypad Controller 2 and press the Left Joystick upwards. This will select the ‘Left Y’ setting. Select ‘Position’ and change the Servo to ‘Arm’. Set the Upper Position to 255 and the Lower Position to 0.



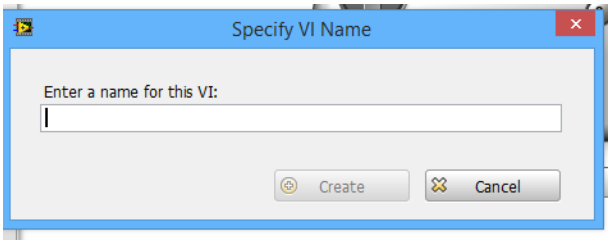
Select Joypad Controller 2 and press the Right Joystick upwards. This will select the ‘Right Y’ setting. Select ‘Position’ and change the Servo to ‘Claw’. Set the Upper Position to 255 and the Lower Position to 210.

Test All the Remote Control Setup

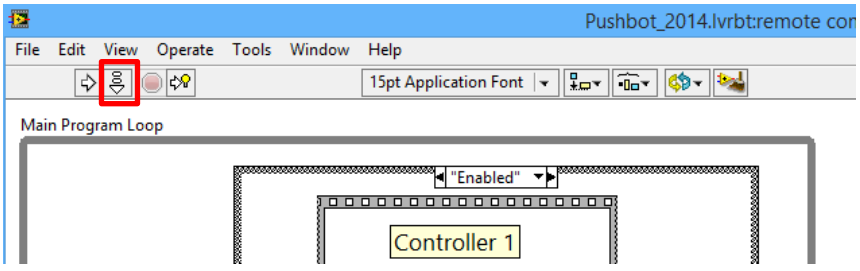


Again to test the Code, save the configuration by clicking on 'Save' in the Top left-hand Corner. Select 'Run Program' and 'Start Test'. Joypad 1 Left and Right Joysticks can now be used to drive the robot around and Joypad 2 can be used to control the Arm and Claw Servos.

Once the code is tested and working well, click on Generate Code.



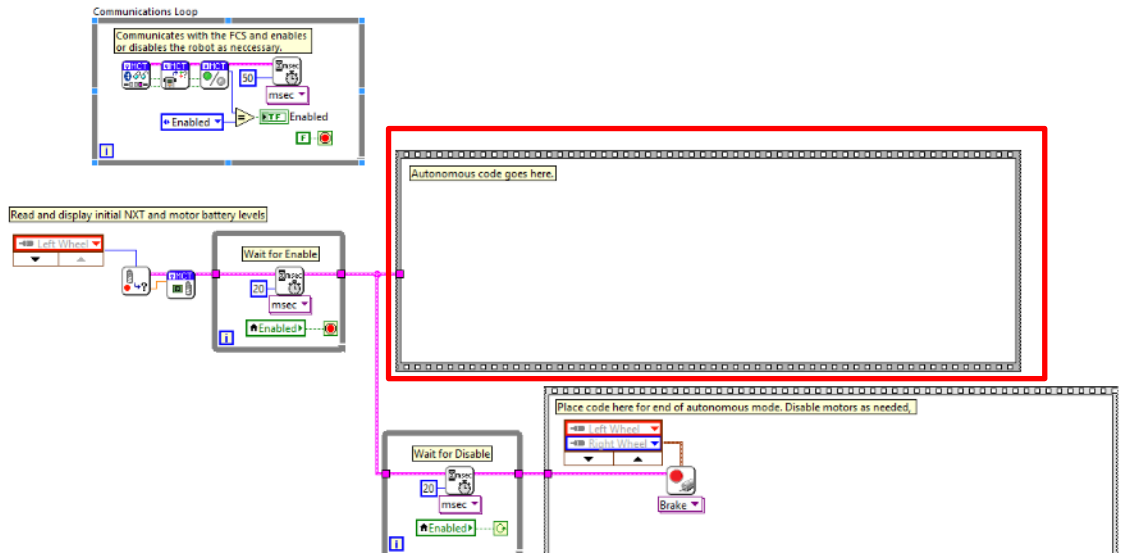
Enter a name for the code and click 'Create'. This will generate the VI code for the Remote Control. Write down all the Remote Control Actions for the Joypads. The Remote Control can be downloaded to the NXT using 'Deploy'. Saves any changes when closing the code.



Autonomous Code

Autonomous Code is used in the first part of an FTC Game. The Robot autonomously drives around the field to complete various 'missions' These may include finding an Infra-Red Beacon and depositing a 'block', positioning the Robot on a platform or ramp or upending Crates.

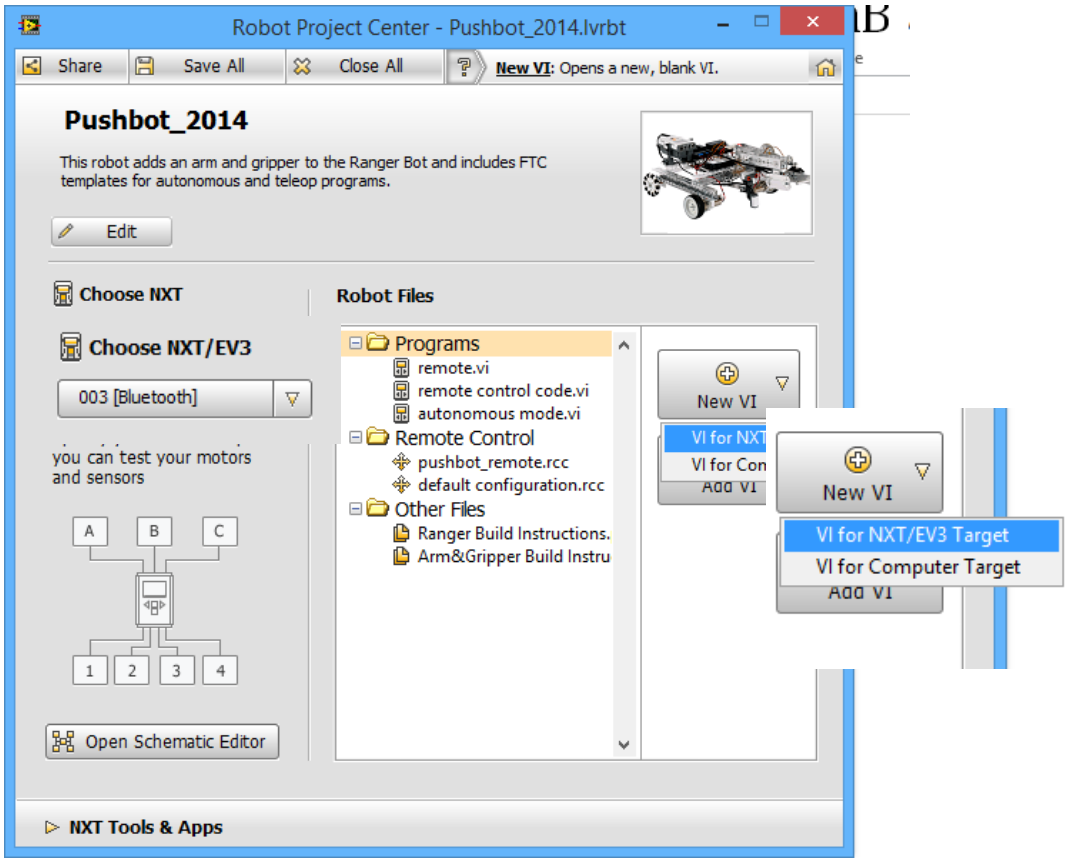
For the FTC Game, the code will have to be copied to an Autonomous Template. The Autonomous is placed the Red Square as shown below.

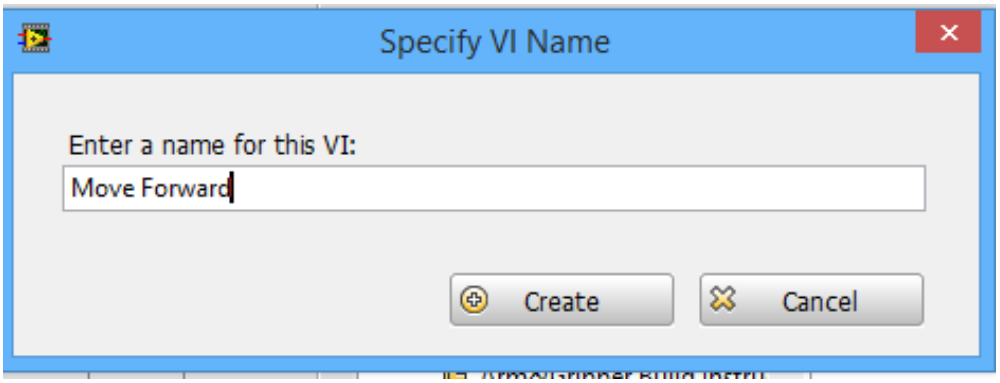


To write and test Autonomous code, the above template will not be used.

Creating Autonomous Code.

In the Robot Project Center, Select 'Programs' and click on 'New VI' and select 'VI for NXT Target'

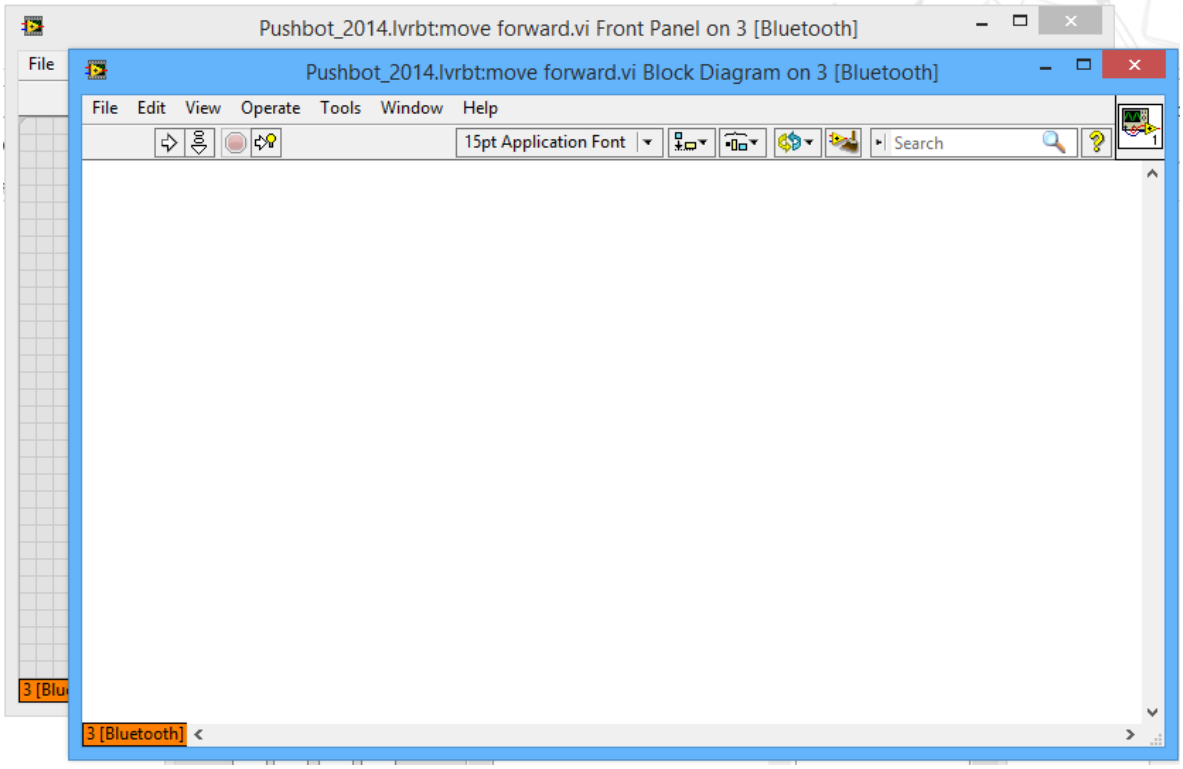




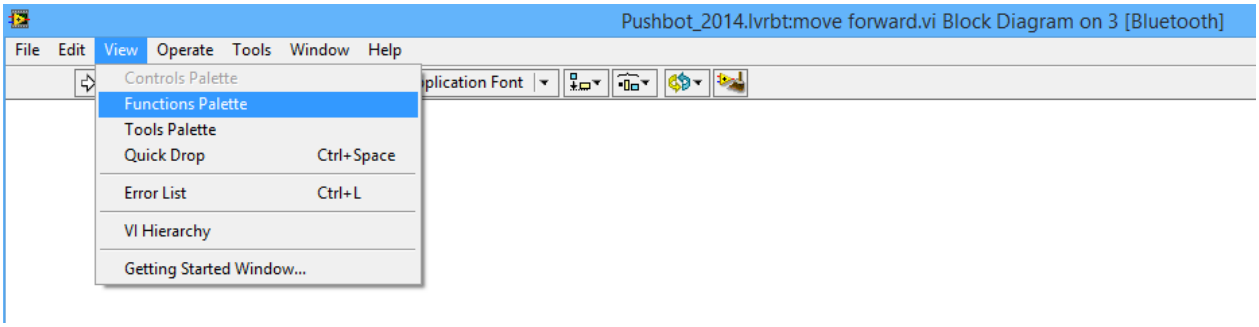
Enter a name for the VI and select ‘Create’

The Front Panel and Block Diagram.

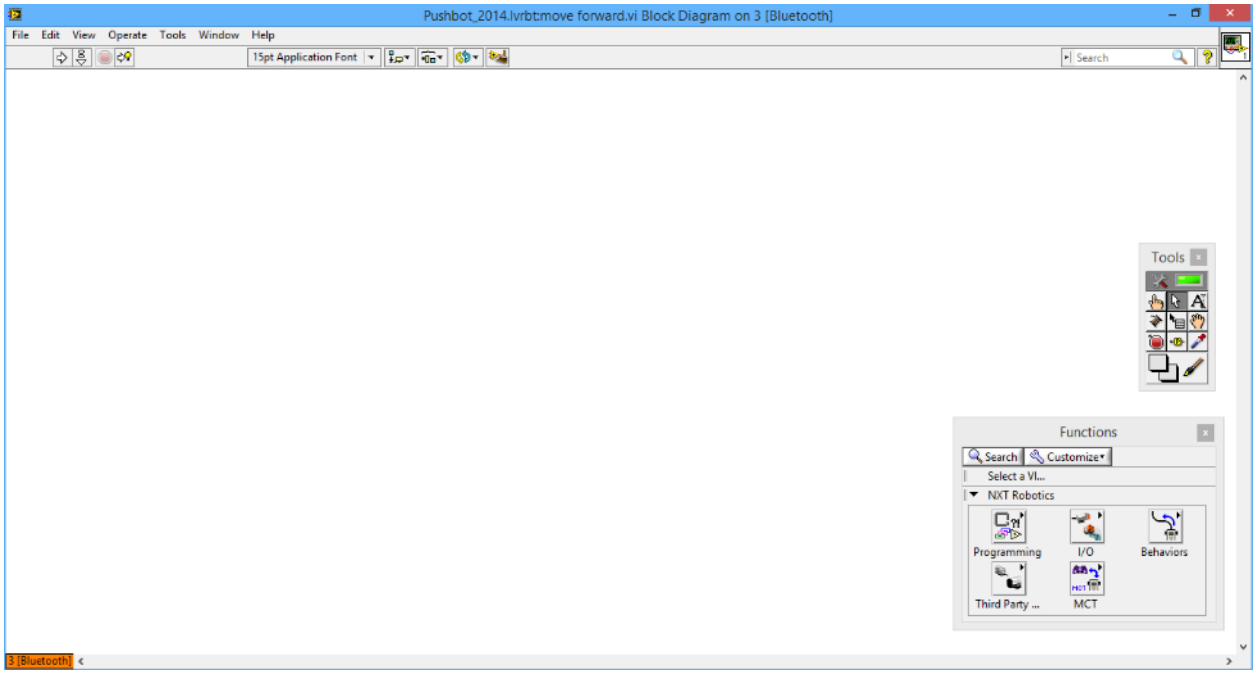
All Code for the Autonomous in written in the Block Diagram Window as shown below.



The Tools and Function Palettes now need to be opened.



Select ‘View’ and click on ‘Functions Palette’ and then select ‘Tool Palette”



These will be the most useful parts of the Tool Palette

Operating changes the value of a control.

Positioning positions, resizes, and selects objects.

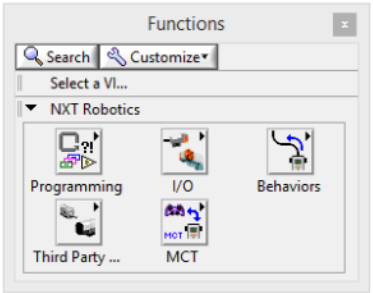
Labeling creates free labels and captions, edits existing labels and captions, or selects the text within a control.

Wiring wires objects together on the block diagram.

Object shortcut menu opens the shortcut menu of an object.

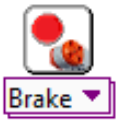
Scrolling scrolls the window without using the scroll bars.

The Function Palette has many sub palettes. The search option is helpful when a block name is known but not the location.

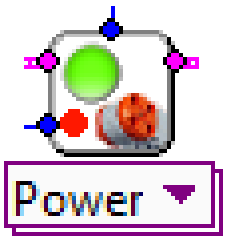


Driving Forward for 5 seconds.

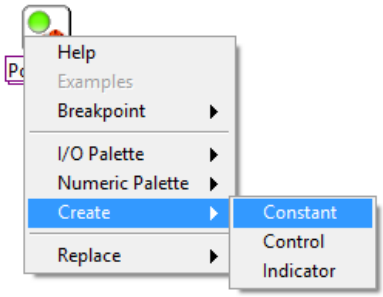
Open the 'Third Party Motors' sub palette in the Functions Palette and drag in a 'Move Motors' block and a 'Stop Motors' Block. Click on the NXT Robotics Header to back to the Main Function Palette as shown above. Select the I/O sub palette and drag in a 'Wait For' block and then

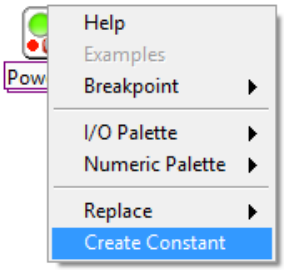


When you click on a block with the Wiring Tool (see useful parts of the Tool Palette on the previous page), the wiring options for the block are visible as shown below.

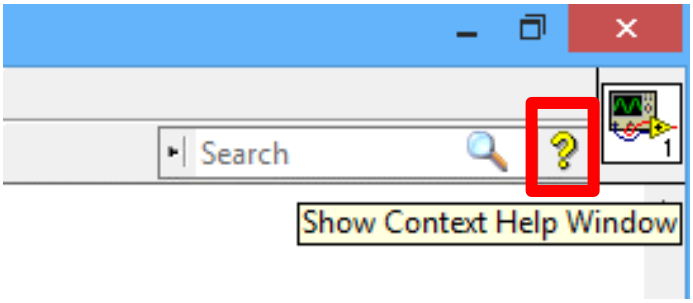


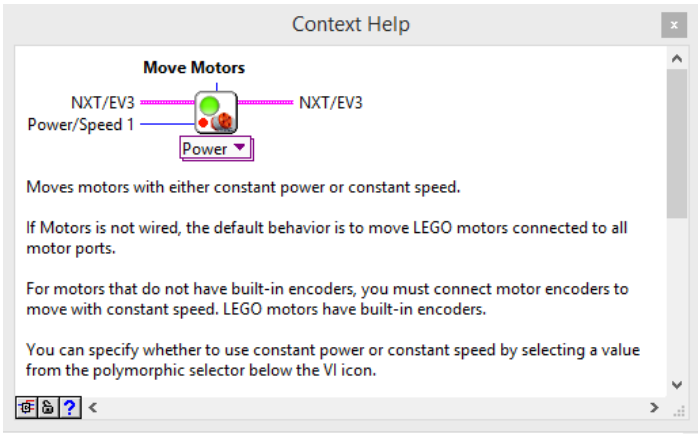
To create a wiring input or output, 'Right Mouse Click' on the Block with the Wiring Tool and select 'Create' and 'Constant' (or with some wiring inputs just 'Create Constant').





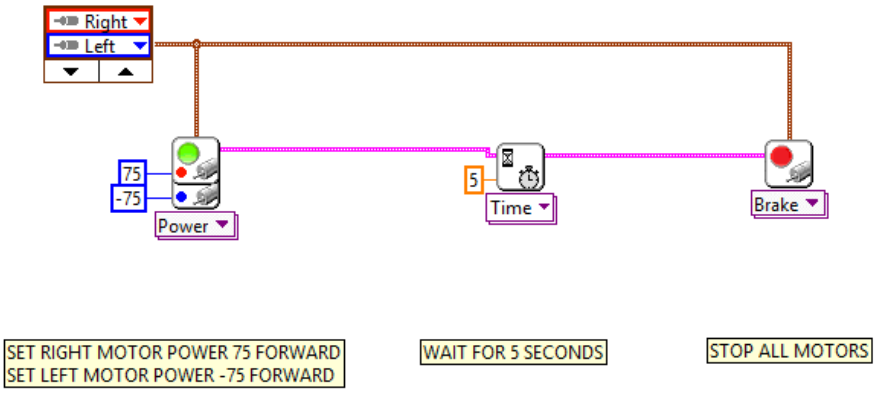
To find out the type of Inputs and Outputs and their ranges use Help in top right-hand side of the Block Diagram Window and click on any block.



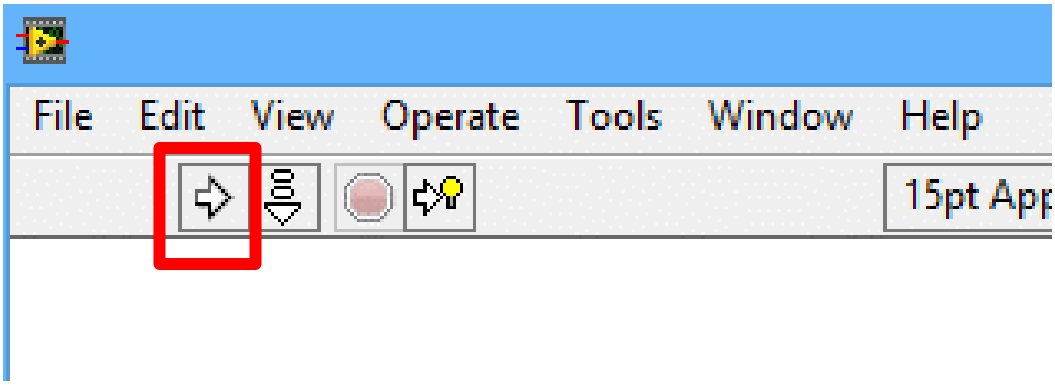


The 'Move Motor' block Help Window.

Note: The blocks will change to the icons shown below once the correct inputs are wired in.



Use the wiring tool to create all the Block Inputs and Outputs as above. The Operating Tool is used to modify all the Inputs and Outputs to each block.



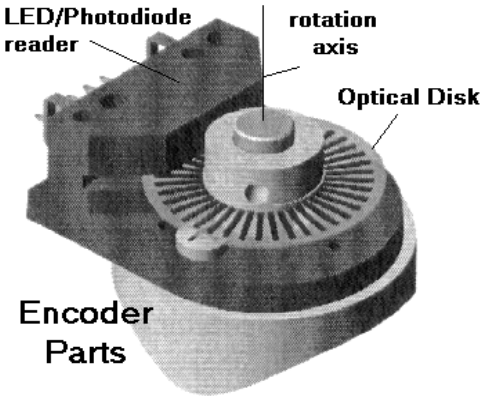
Click on 'Run' to download and run the program. Ensure the NXT and MATRIX Controller is still on.

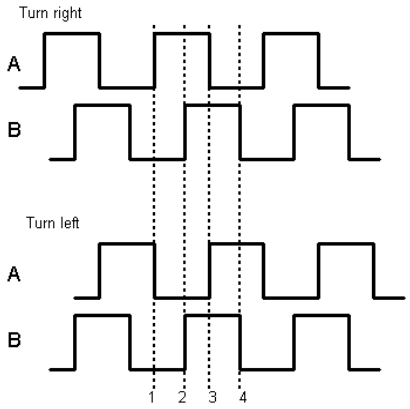
The distance travelled by the robot is dependent on the MATRIX Battery power level when using 'wait for time' commands. This is not ideal when the robot is required to travel a specific distance every time. To solve this problem motor encoders are used.

Motor Encoders.

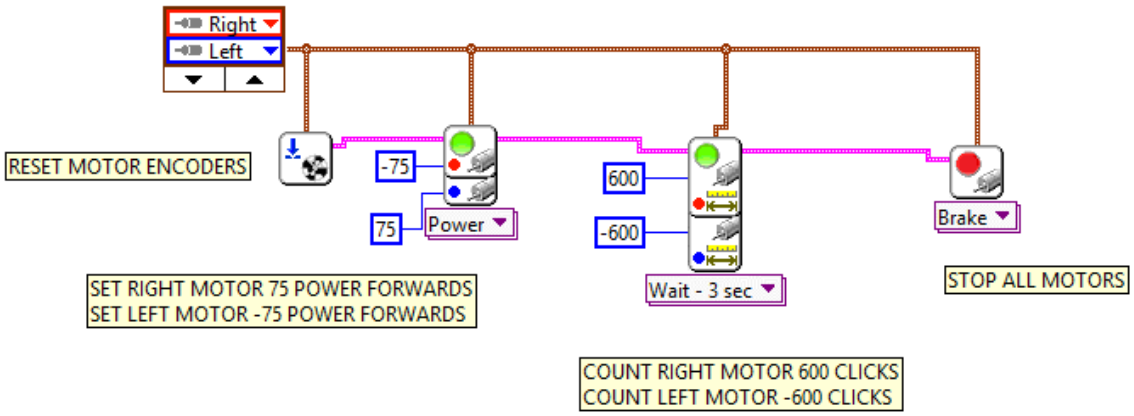
Motor Encoders are used to 'count' how far the motors have turned. The LEDs and photodiodes produce a pulse of square waves which the robot 'counts' to work out the number of rotations travelled.

The MATRIX Motor Encoders use a count of 600 'clicks' for every 1 motor rotation.





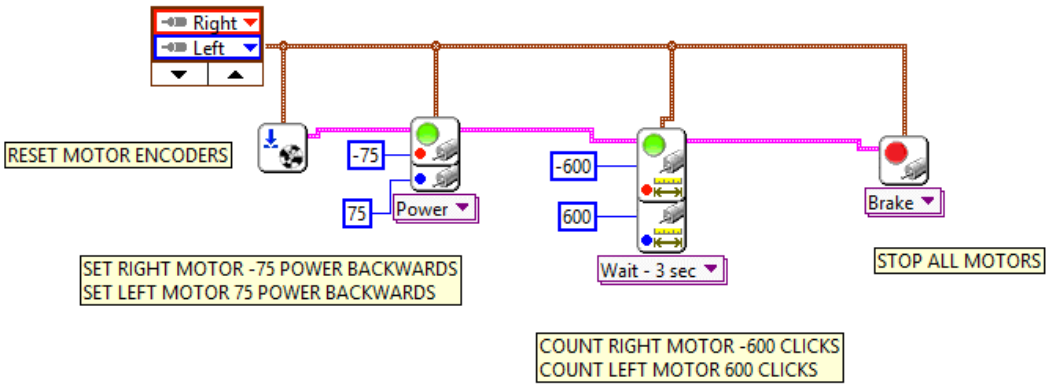
From the Robot Project Center, Open a new 'Program' 'New VI' Program and call it 'Forward Encoder'



Select the 'Third Party Motors' palette. The encoders have to be reset using the 'Reset Encoder' block. Add a 'Motor Move' Block a 'Fixed Distance' block and the 'Stop Motors' block. Add the Input and Output Ports as shown above. For every 600 'clicks' of the encoder the Motor will rotate once. This equates to 300mm of distance travelled by the robot using the 95mm wheel attached to the robot.

Reversing Using Encoders

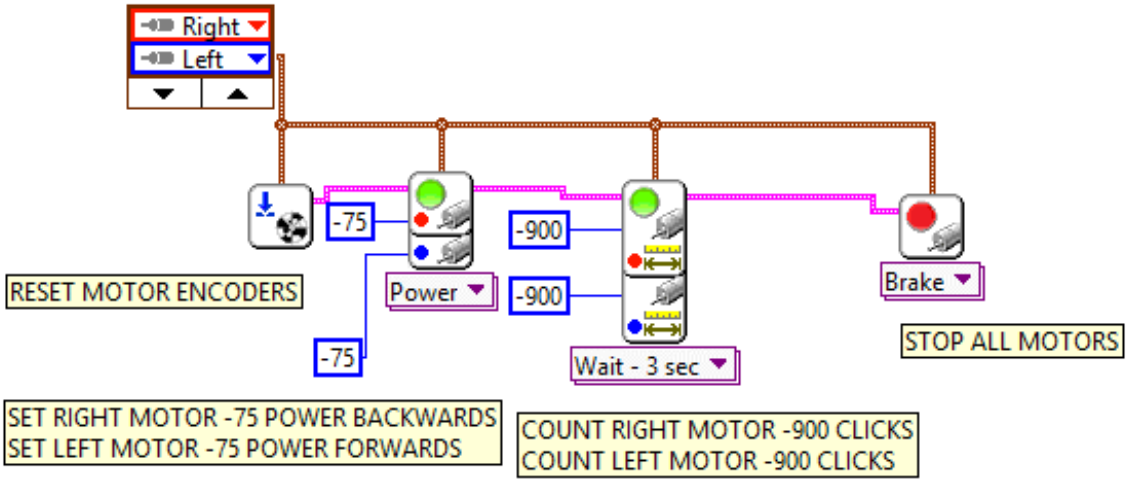
From the Robot Project Center, Open a new 'Program' 'New VI' Program and call it 'Reverse Encoder'.



This program is similar to the driving Forward with Encoders but now the Motors have been reversed.

Turning 90 degrees to the Right with Encoders.

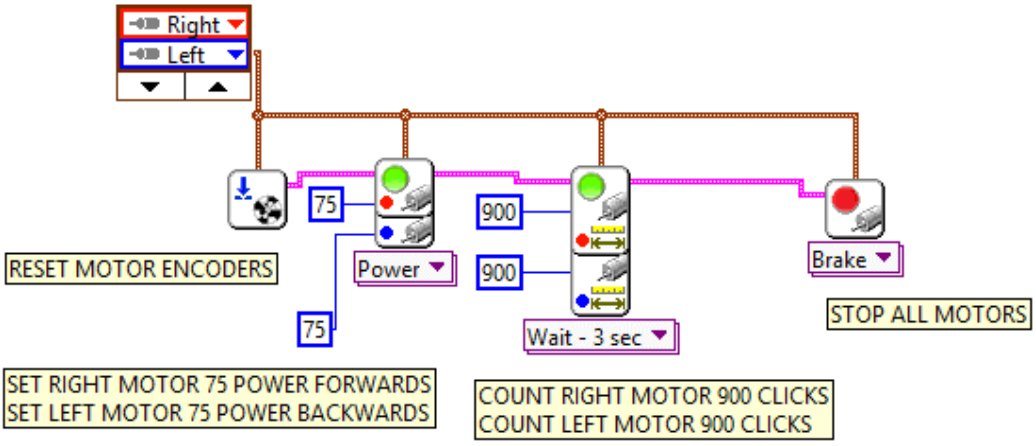
From the Robot Project Center, Open a new ‘Program’ ‘New VI’ Program and call it ’90 degree Turn Right Encoder’



The direction of rotation on the ‘Motor Move’ turns the robot right and the ‘Fixed Distance’ block waits for -900 ‘clicks’ before stopping all Motors. The amount the robot turns can be changed by increasing or decreasing the Input Value into the ’Fixed Distance’ block.

Turning 90 degrees to the Left with Encoders

From the Robot Project Center, Open a new 'Program' 'New VI' Program and call it '90 degree Turn Left Encoder'

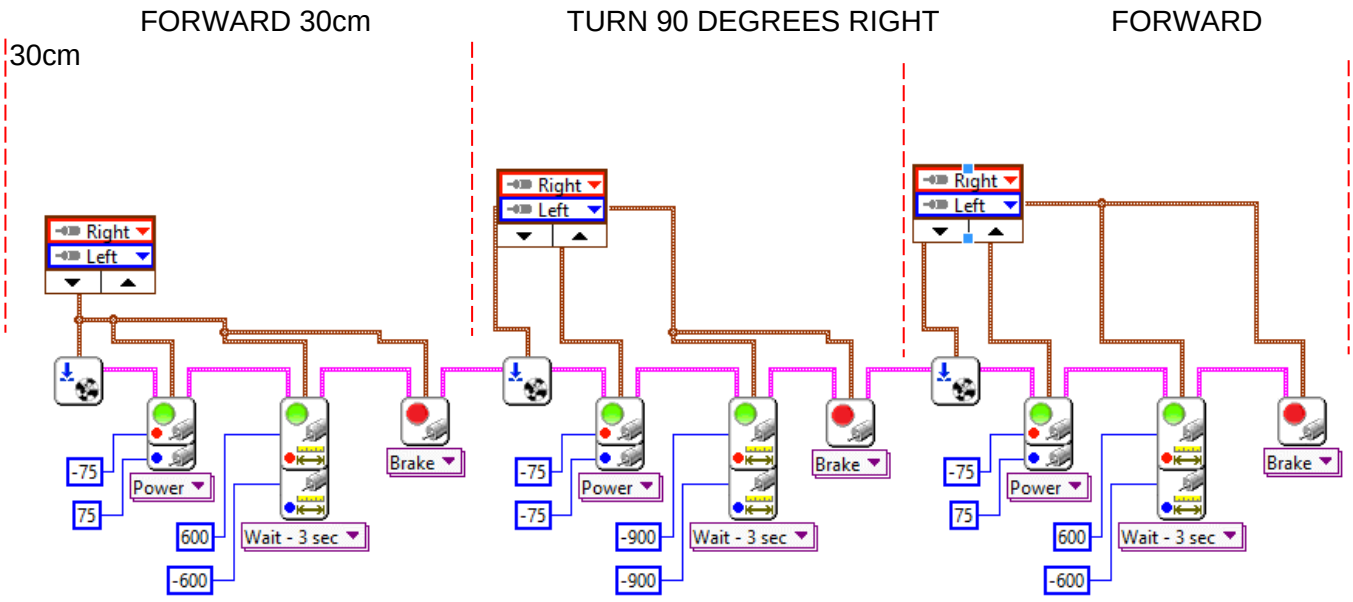


The direction of rotation on the 'Motor Move' has been reversed and the 'Fixed Distance' waits for 900 'clicks'.

Moving the Robot in Autonomous Mode.

The four Encoder Movement programs shown above can be linked together to form complex movements.

Example

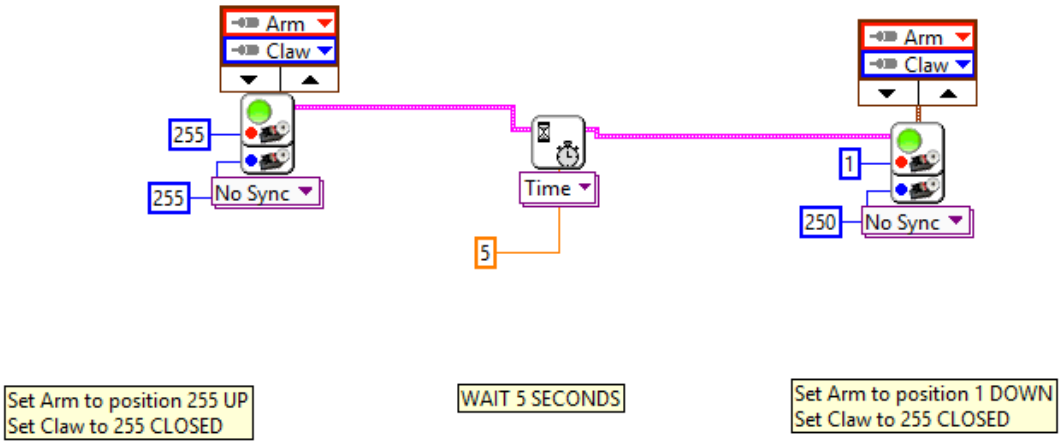


Using Servos in the Autonomous Period.

Servos are constructed from three basic pieces; a motor, a potentiometer (variable resistor) that is connected to the output shaft, and a control board. The potentiometer allows the control circuitry to monitor the current angle of the servo motor. The motor, through a series of gears, turns the output shaft and the potentiometer simultaneously. The potentiometer is fed into the servo control circuit and when the control circuit detects that the position is correct, it stops the motor. If the control circuit detects that the angle is not correct, it will turn the motor the correct direction until the angle is correct. Normally a servo is used to control an angular motion of between 0 and 180 degrees. It is not mechanically capable (unless modified) of turning any farther due to the mechanical stop built on to the main output gear.

The output values from servos range from 0 to 255 (Approximately 0.7 servo value per 1 degree of rotation). Servos need a small amount of time to reach their position so 'wait for time' commands are used after setting a servo position. Two servos are used in the Robot, one for the arm and the other for the claw. The arm servo has been geared down to allow for the weight of the claw and objects that it picks up.

From the Robot Project Center, Open a new 'Program' 'New VI' Program and call it 'Servo Test'



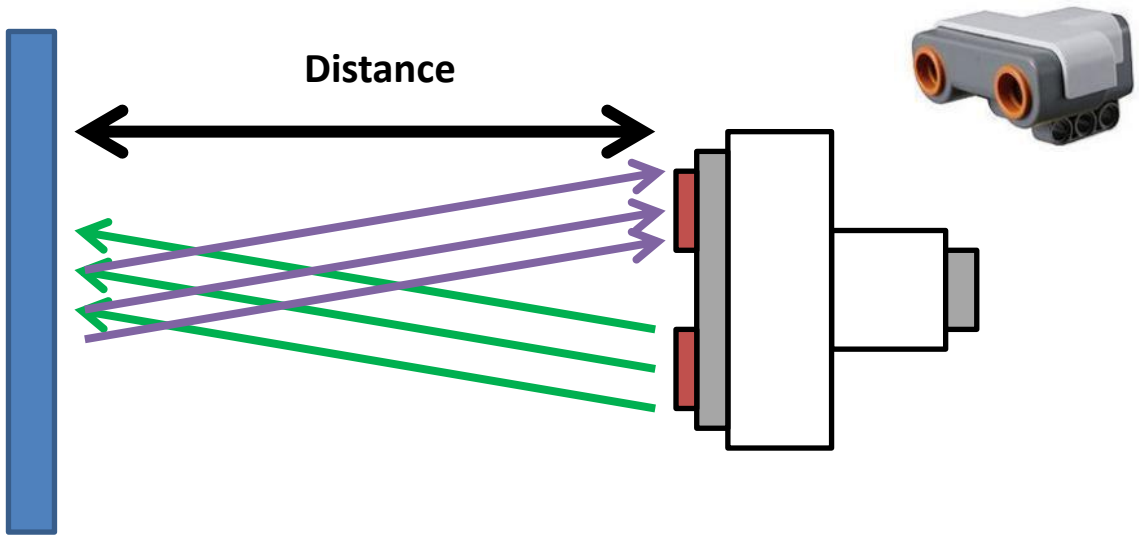
The program above, positions Servo 'Arm' and Servo 'Claw' at servo position 255, waits for 5 seconds and then repositions Servo 'Arm' at servo position 255 and keeps Servo 'Claw' at servo position 255.

Using Sensors in the Autonomous Period.

The robot has been fitted with 2 sensors, the LEGO NXT Ultrasonic and the Hitechnic Infra-Red seeker sensor.

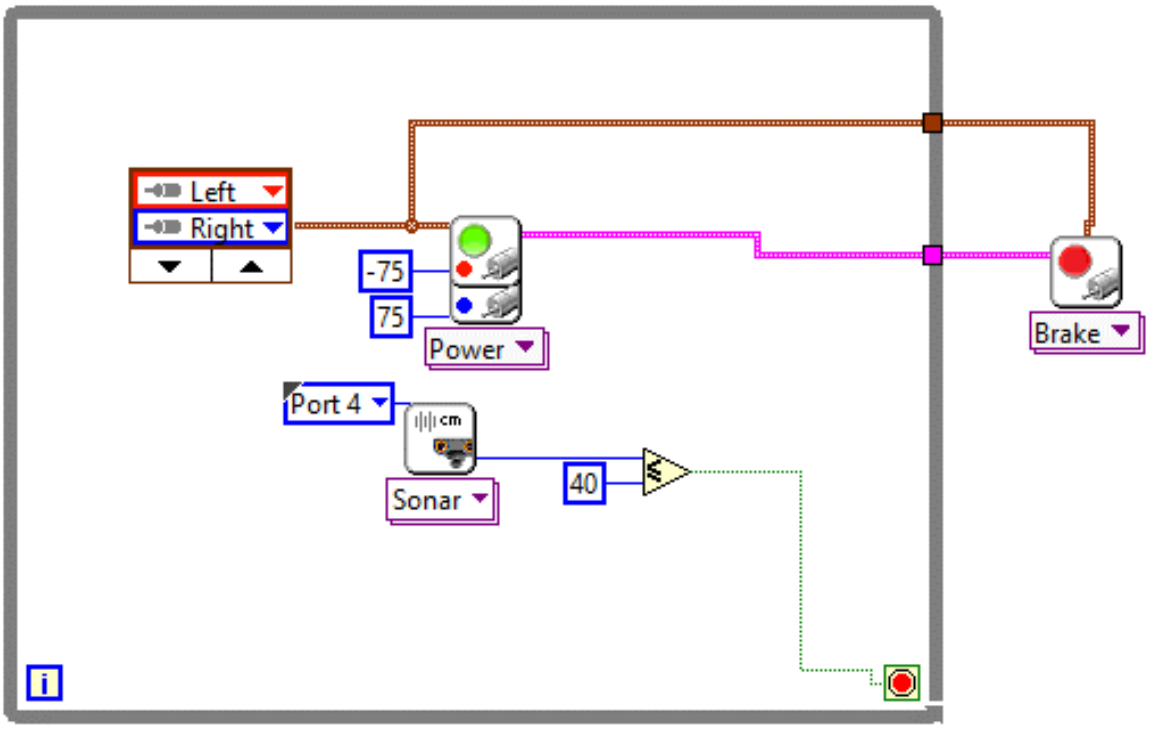
The Ultrasonic Sensor

The LEGO ultrasonic sensor is a range sensor. It measures the distance between the sensor and an object in front of the sensor. It does this by sending short beeps and measuring how much time it takes for the beep to bounce back. Knowing the speed of sound it can then calculate the distance to the object. It works just like the sonar of a submarine.



Ensure the Ultrasonic sensor has been plugged in NXT Port 4.

From the Robot Project Center, Open a new 'Program' 'New VI' Program and call it 'Sonic'

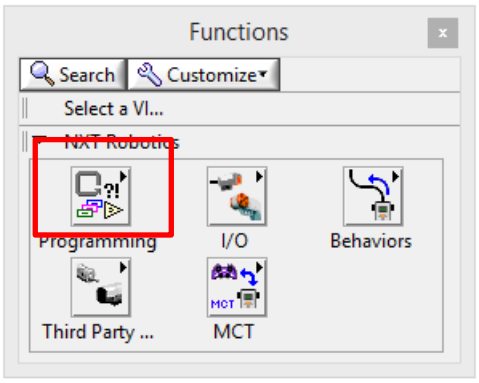


Drive forward while Sonar Sensor is Greater than 40cm

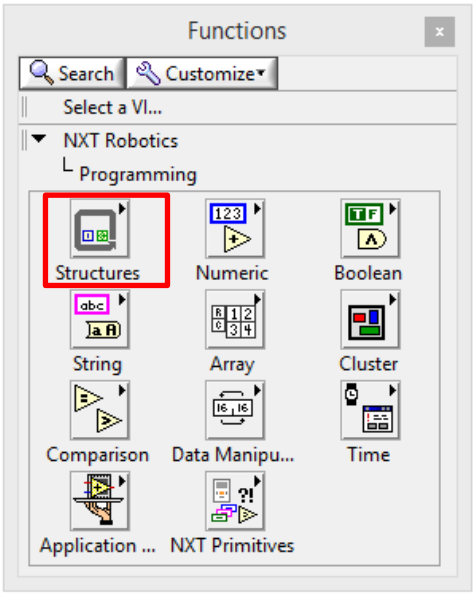
In this program, the Robot will drive forward until the ultrasonic detects a wall 40cm in front of it, the robot will then stop.

This program uses a ‘while loop’ to continually monitor the sensor until the ultrasonic see less then 40cm.

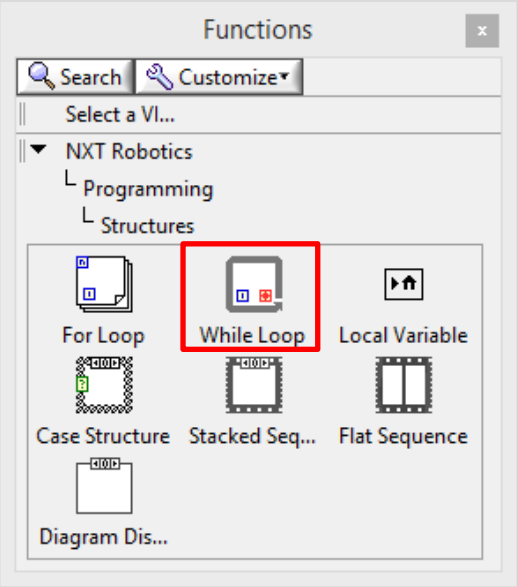
Open the Function Palette and select the ‘NXT Programing’ Sub palette.



Then select the ‘Structures’ sub palette

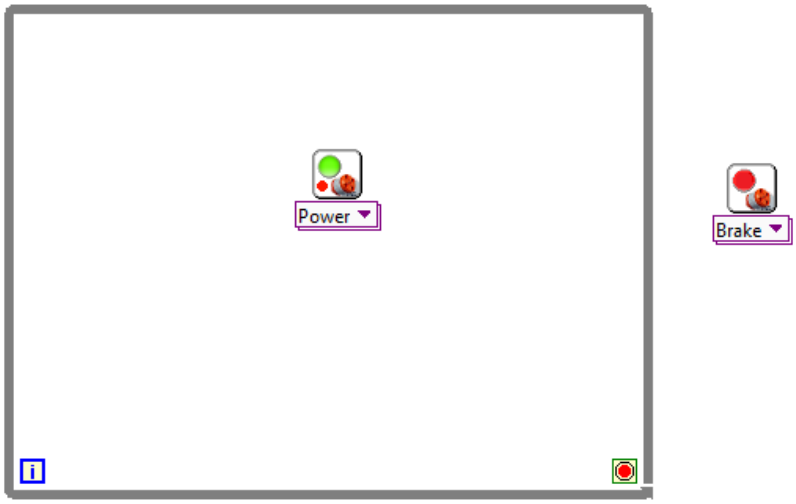


Then select the ‘While Loop’ block.

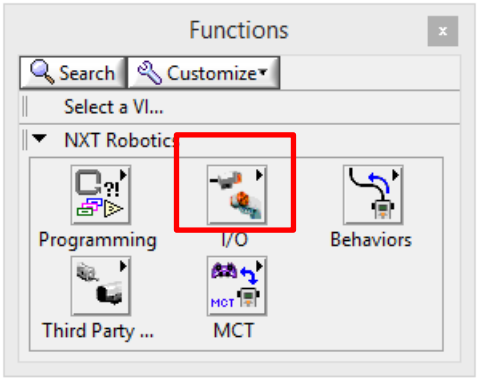


Drop and drag a 'While Loop' Rectangle. If you find you run out of space inside this rectangle, you can click+drag while holding in the "CTRL" key to add space to the diagram.

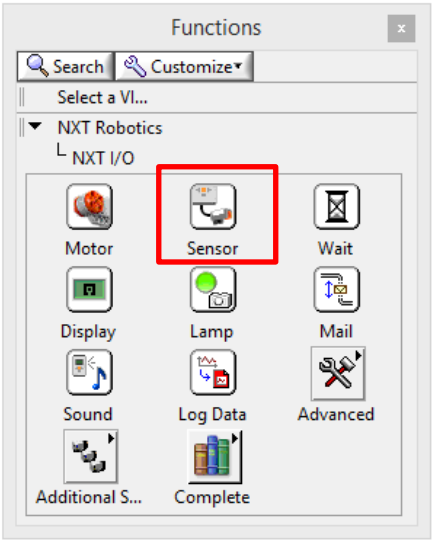
Add a 'Motor Move' block on the inside of the loop and a 'Stop Motor' block on the outside of the loop.



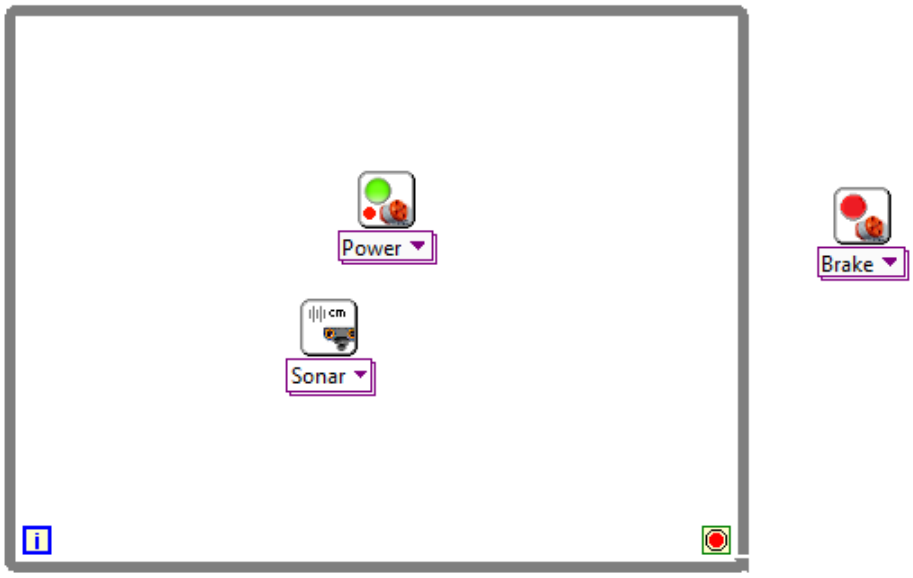
Go Back to the NXT Robotics Palette and select the 'I/O' Sub palette.



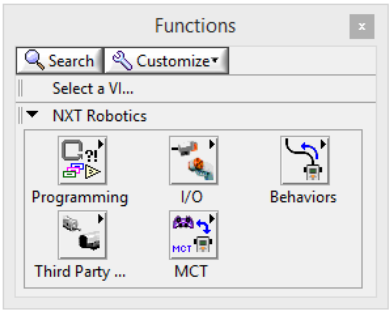
Select the ‘Sensor’ Block and drop it inside the Loop



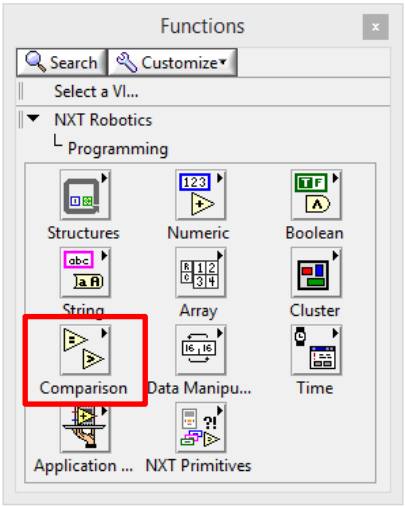
Modify the Sensor block to ‘Read Ultrasonic’ using the drop down menu box.

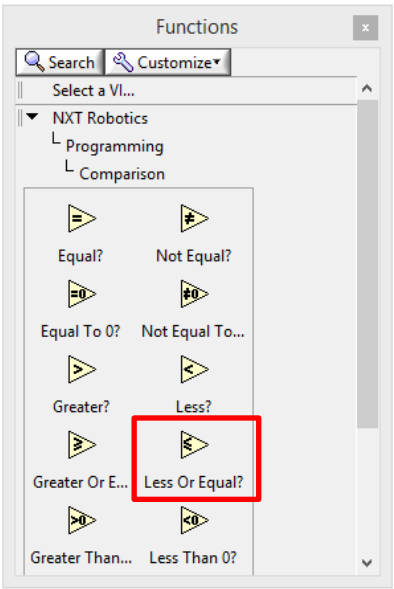


Go back to the NXT Robotics Palette and select the ‘Programming’ sub palette.

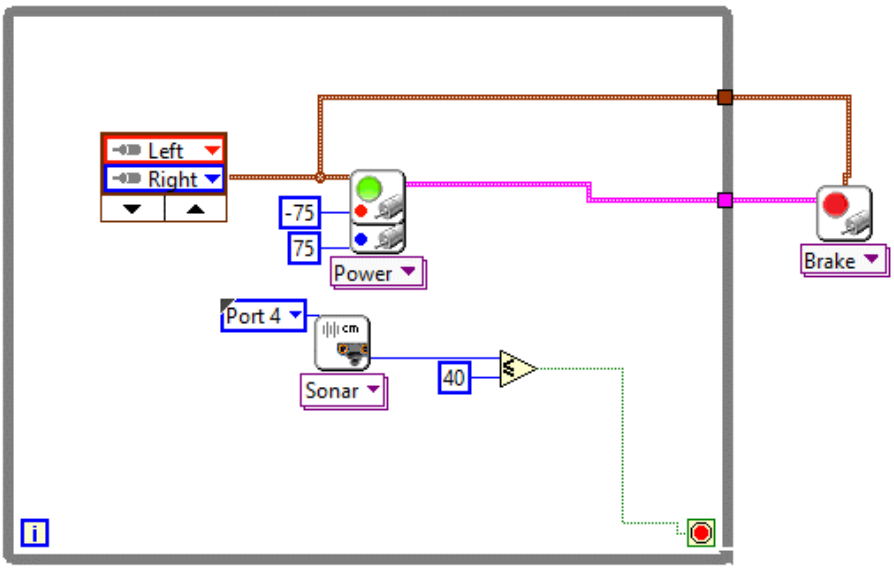


Select the ‘Comparison’ sub palette.





Select the 'Less or Equal' Block and place it inside the Loop.

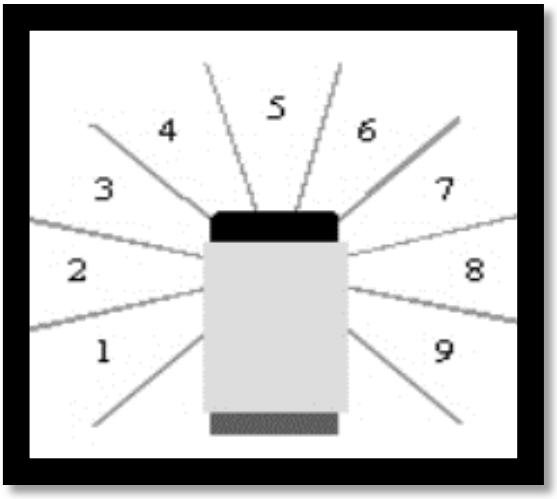


Drive forward while Sonar Sensor is Greater than 40cm

Modify the blocks so it looks like the program above. Download and Run the program. Adjust the Less than 40 so the robot stops closer or further away from an object.

The Infra-Red Seeker Sensor

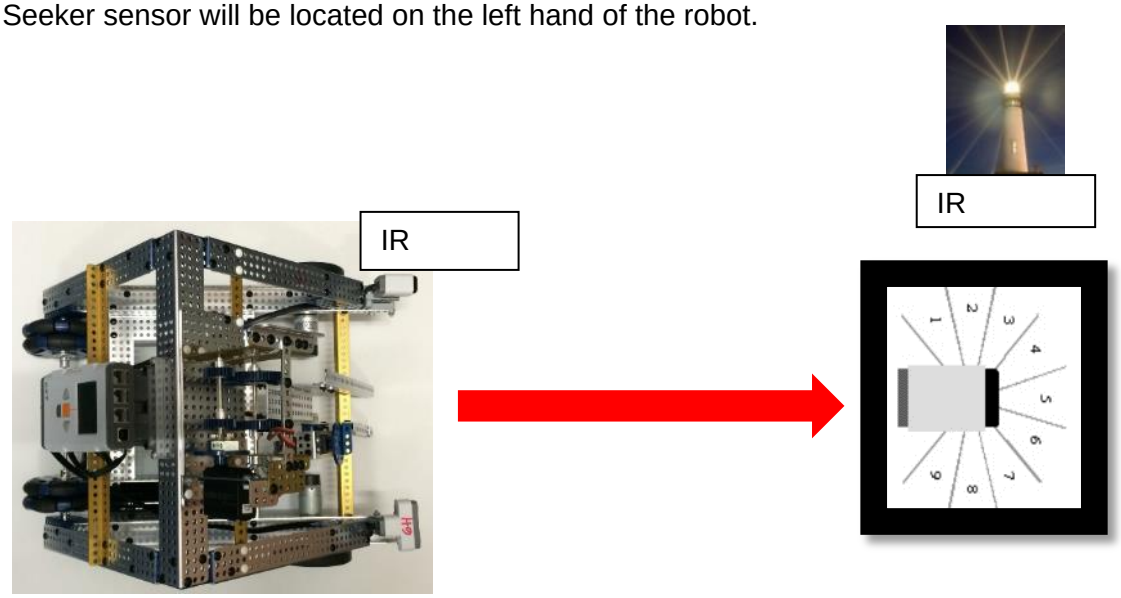
The Infra-Red (IR) Seeker Sensor has a 240 degree sensing ability. This allows the sensor to 'seek' an IR source and then pin point its location.



The Infra-red sensor has a range of 0 to 9. The 1 to 9 reading shows the position of the infra-red source in relation to the sensor. The IR source could be the FTC IR beacon used in the Autonomous time period, a Robocup Junior IR Soccer ball, most TV remotes or the LEGO® Power Functions remotes. A simple candle could also be used as an IR source.

Detecting an IR Source

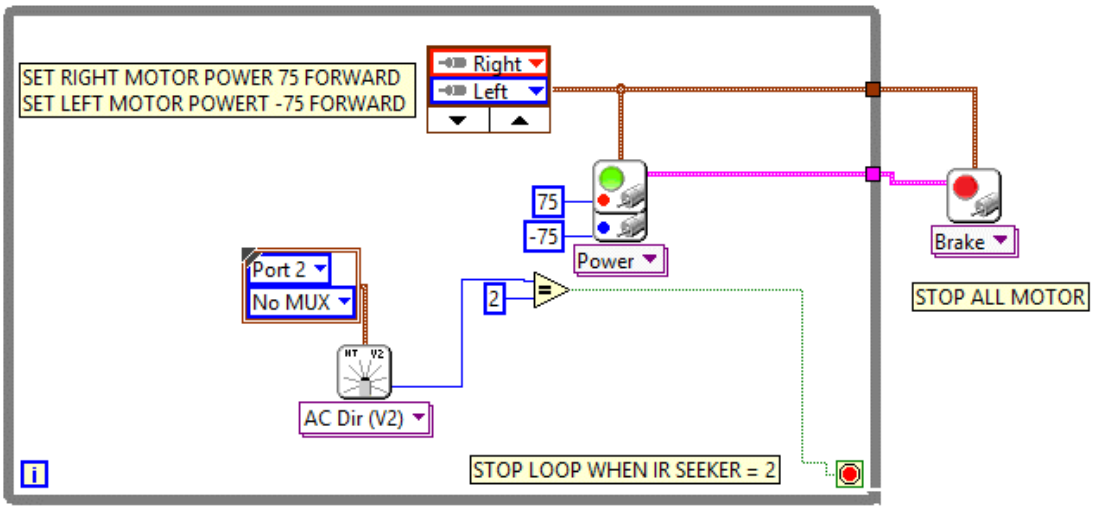
The robot will drive forward until it is perpendicular to the Infra Source (FTC IR Beacon). The IR Seeker sensor will be located on the left hand of the robot.



This will mean that the IR Seeker will read a value of '2' when the robot is perpendicular to the IR Beacon.

From the Robot Project Center, Open a new 'Program' 'New VI' Program and call it 'IRSeeker'.

The program below, drives the Robot forward until the IR Seeker Sensor sees a value of '2', the robot will then stop.



Add the 'While Loop' as shown in the Sonic Sensor program above. Insert a 'Motor Move' block and 'Stop Motor' block and setup as shown. The 'Equal' block can be found under the 'Comparison' Sub palette. The IRSeeker block is found under 'I/O', 'Additional Sensors and Motors' Sub palette then 'HiTechnic Sensors' and select the 'HT Read' block. Setup the program as above. The 'HT Read' block should be setup as 'Read IRseeker V2' and select 'AC' and 'Direction'

Download and run the program. An Infra Source will need to be used for the robot to find and stop at.

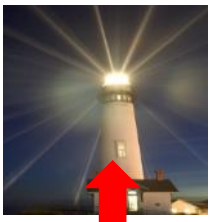
Using the IR Seeker and Ultrasonic sensor to search for and stop in front of the IR Beacon

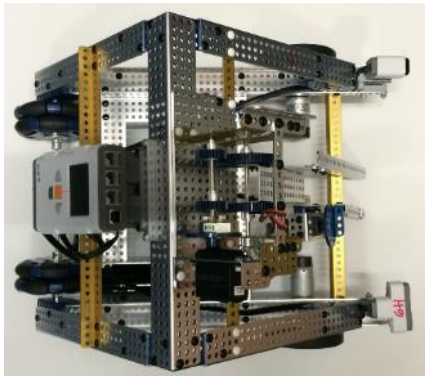
From the Robot Project Center, Open a new 'Program' 'New VI' Program and call it 'IRSeeker Turn and Sonic'.

The Sensor programs and Encoder programs can be added together to produce complex movements.

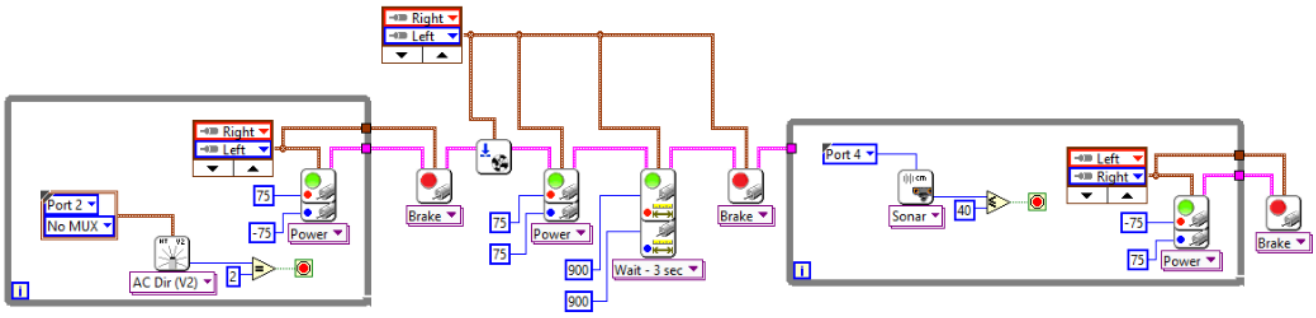
Objective:

1. Forward until perpendicular to the IR Beacon
2. Turn 90.
3. Forward until the Ultrasonic sensor is 40cm away from the wall/beacon.





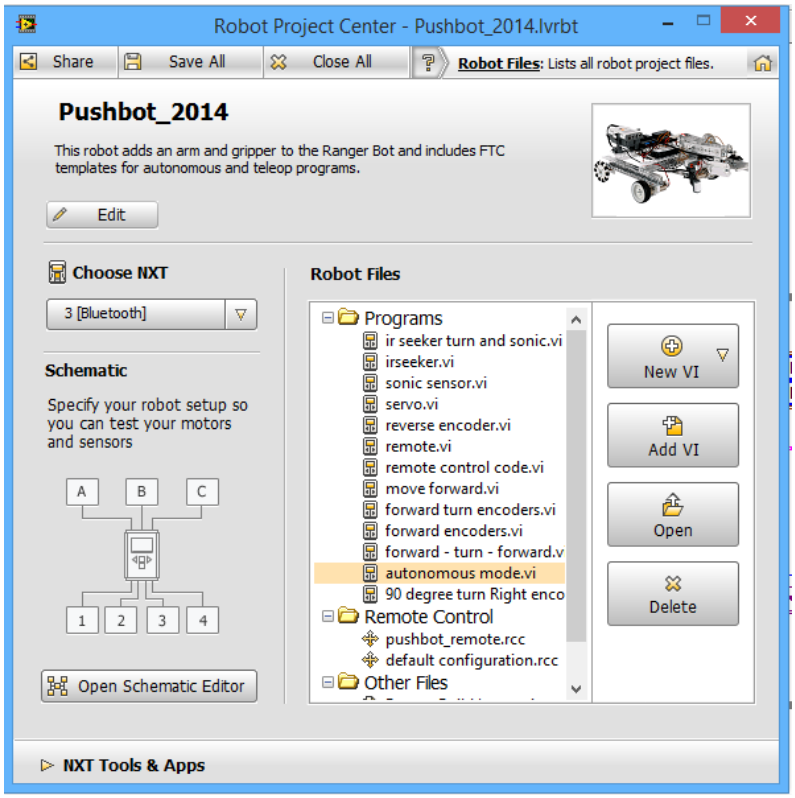
This will program will use the IR seeker to see the IR beacon and the Ultrasonic Sensor to see the wall and use the motor encoders to turn 90 degrees.



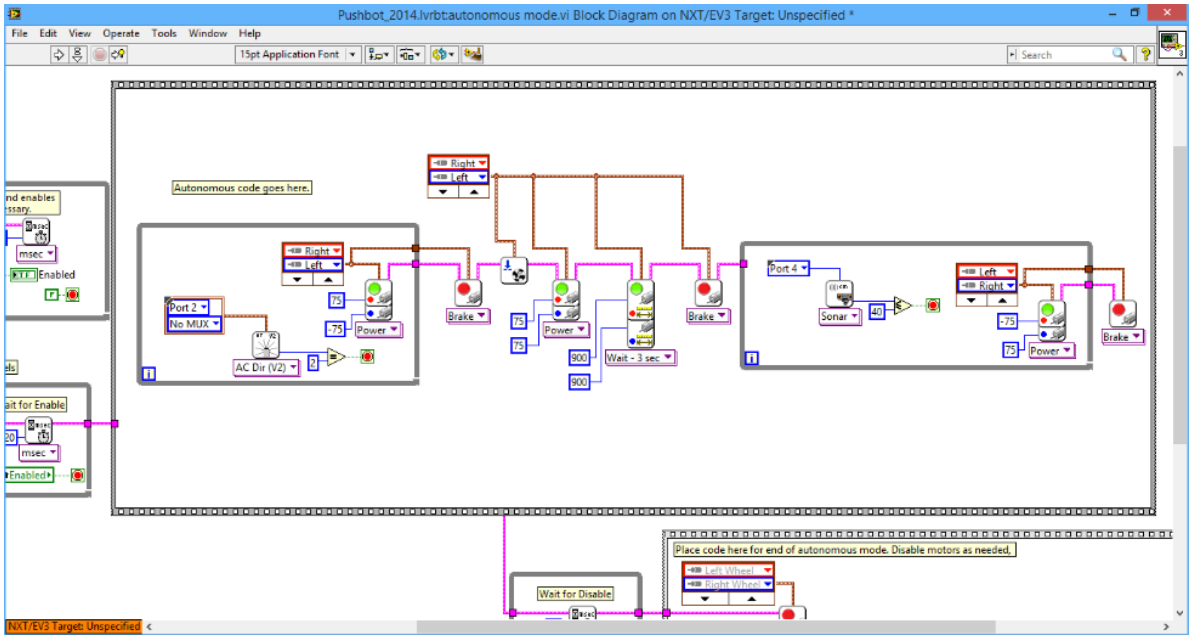
Download and run the code. Once it works copy the code.

Creating the FTC Autonomous Program

From the Robot Project Center open the autonomous mode.vi program.



Paste the Copied Code into the Autonomous Part of the Template.



This file can now be saved and Downloaded to the NXT.

The NXT should have 2 files downloaded on it, an Autonomous Mode and the Remote_Control_Code.

These two files will be used in the setup for the Samantha Module.

Programming the Robot with ROBOTC

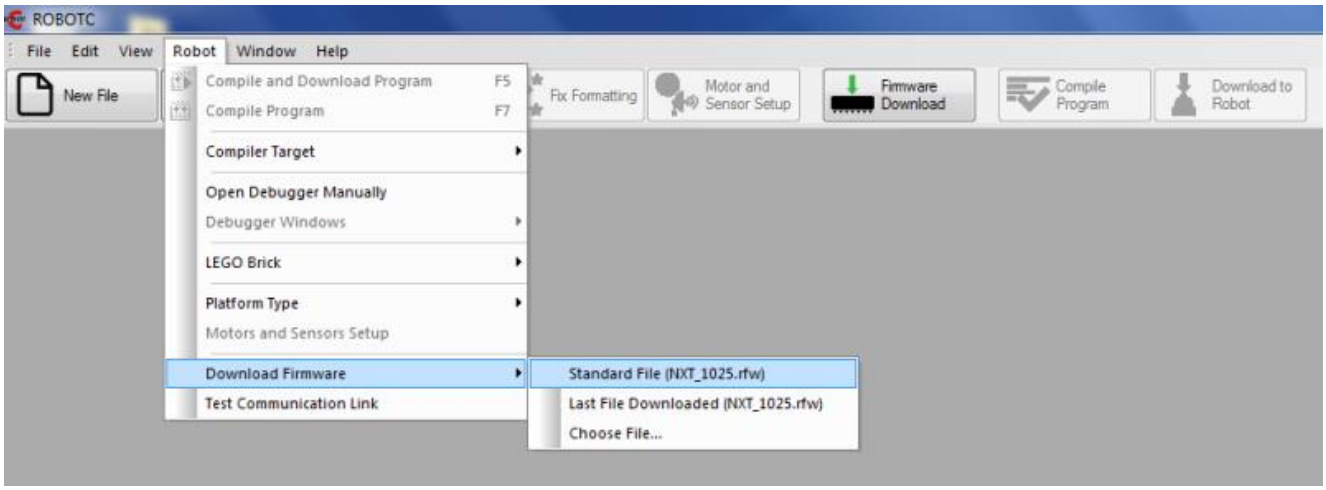
ROBOTC (trademarked as ROBOTC and frequently spelled ROBOTC) is an integrated development environment that is used to program and control the LEGO NXT (and EV3 in Version 4) using a programming language based on the C programming language.

Download the latest version of ROBOTC for the LEGO Mindstorm: http://www.ROBOTC.net/

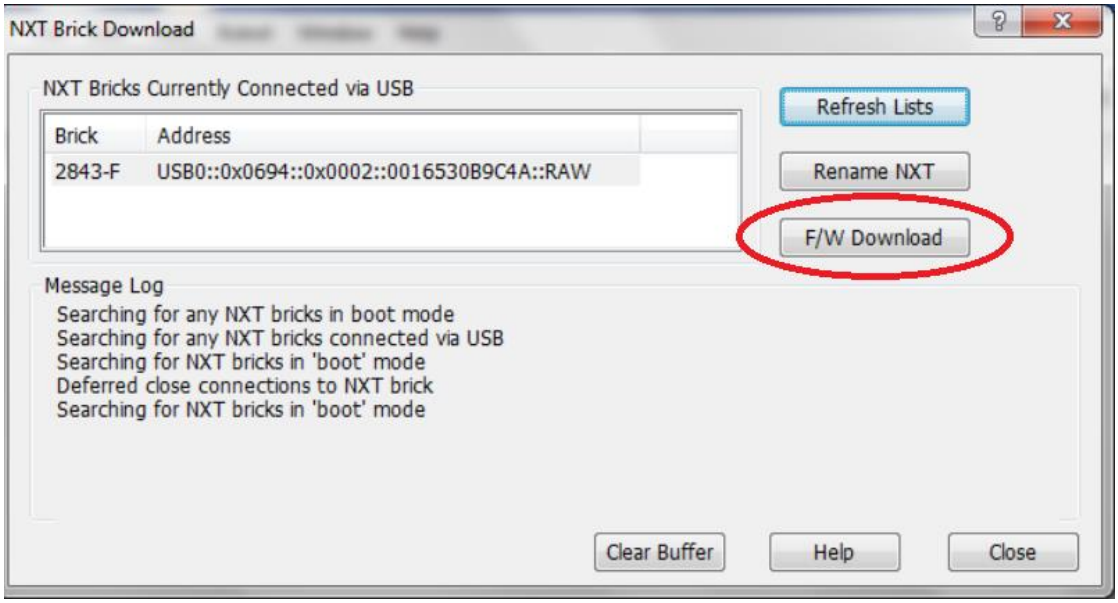
ROBOTC will only run on Windows based PCs. There is a 30 day free trial version or a licence can be purchased through the website. This programming section is written for ROBOTC Version 4.25 and includes the LEGO NXT and EV3 platforms. However, the EV3 is not allowed for FTC use at this time.

Getting Started with ROBOTC

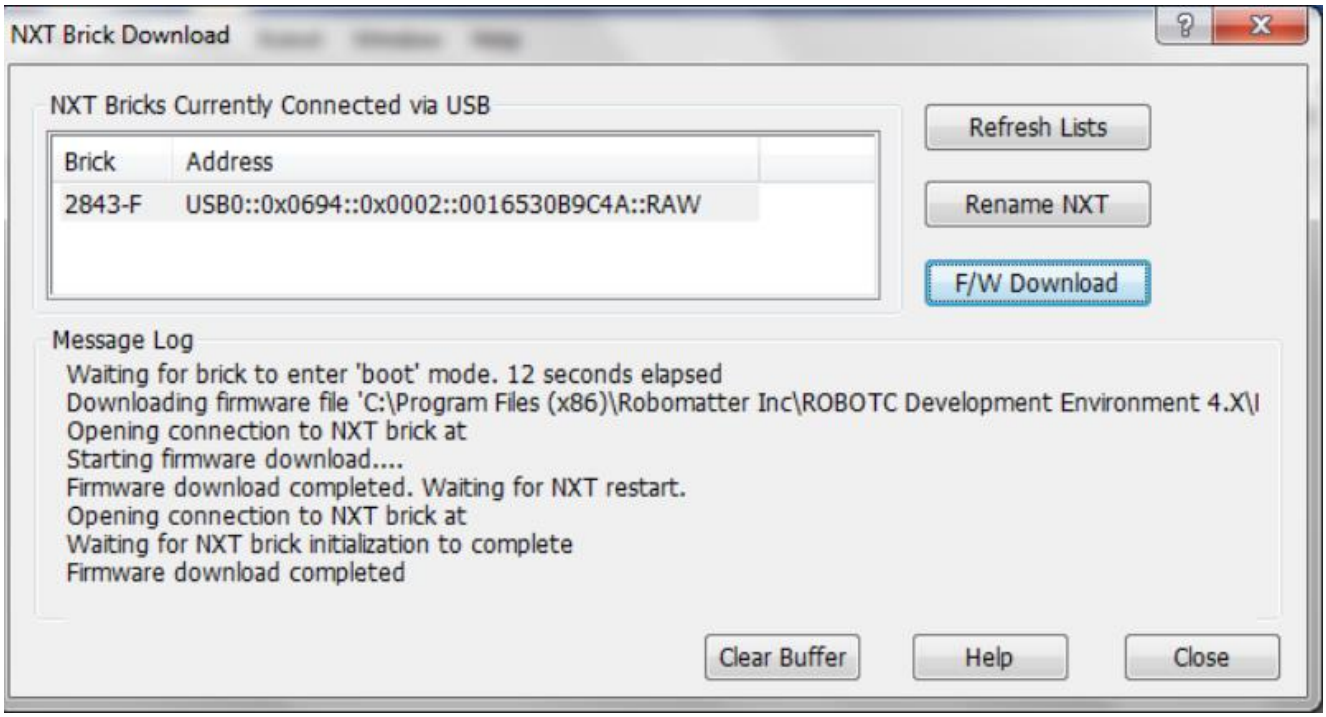
Each version of ROBOTC uses different firmware on the NXT brick. This firmware can be downloaded to the NXT by using the NXT Brick Download dialog. Use the diagram below to open the dialog.



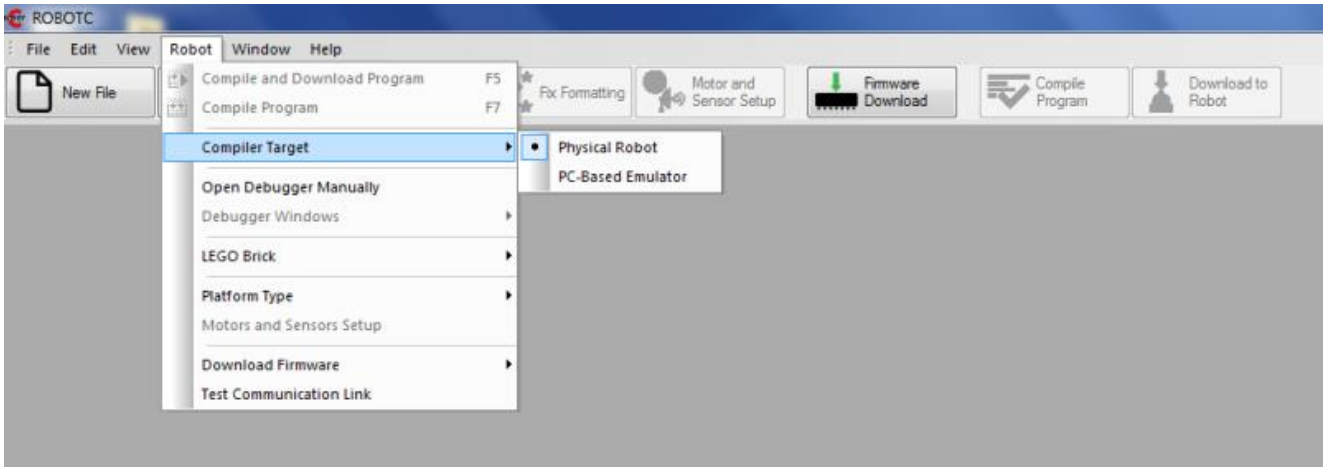
Connect an NXT to the computer with a USB cable and turn the NXT on. If your NXT isn't shown in the list, then click the "Refresh Lists" button. When your NXT is shown in the list, click the F/W Download button (circled in the image below).



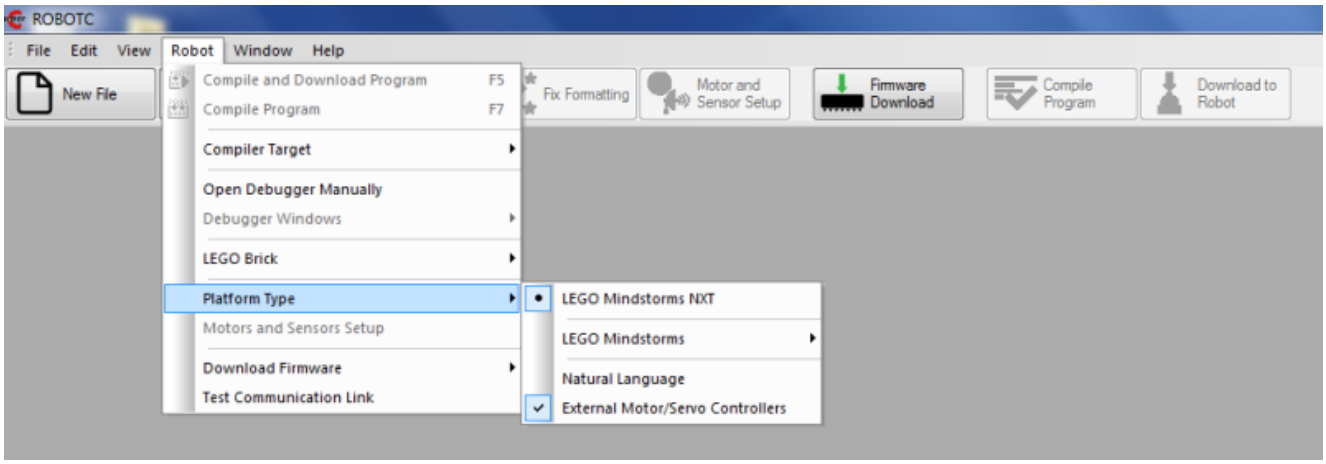
After the firmware has downloaded, the dialog will be as shown below. Click the close button to dismiss the dialog.



ROBOTC is capable of emulating a robot, but for the purposes of this document, verify that it is working with a physical robot.

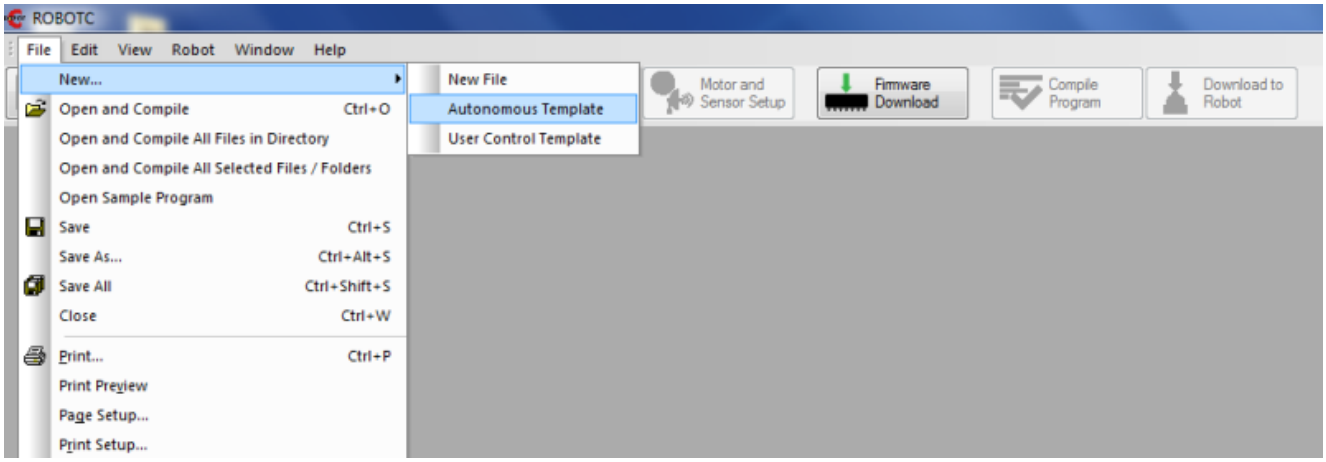


For the NXT to use the TETRIX controllers the platform type must be LEGO Mindstorms NXT and the External Motor/Servo Controllers option must be checked.

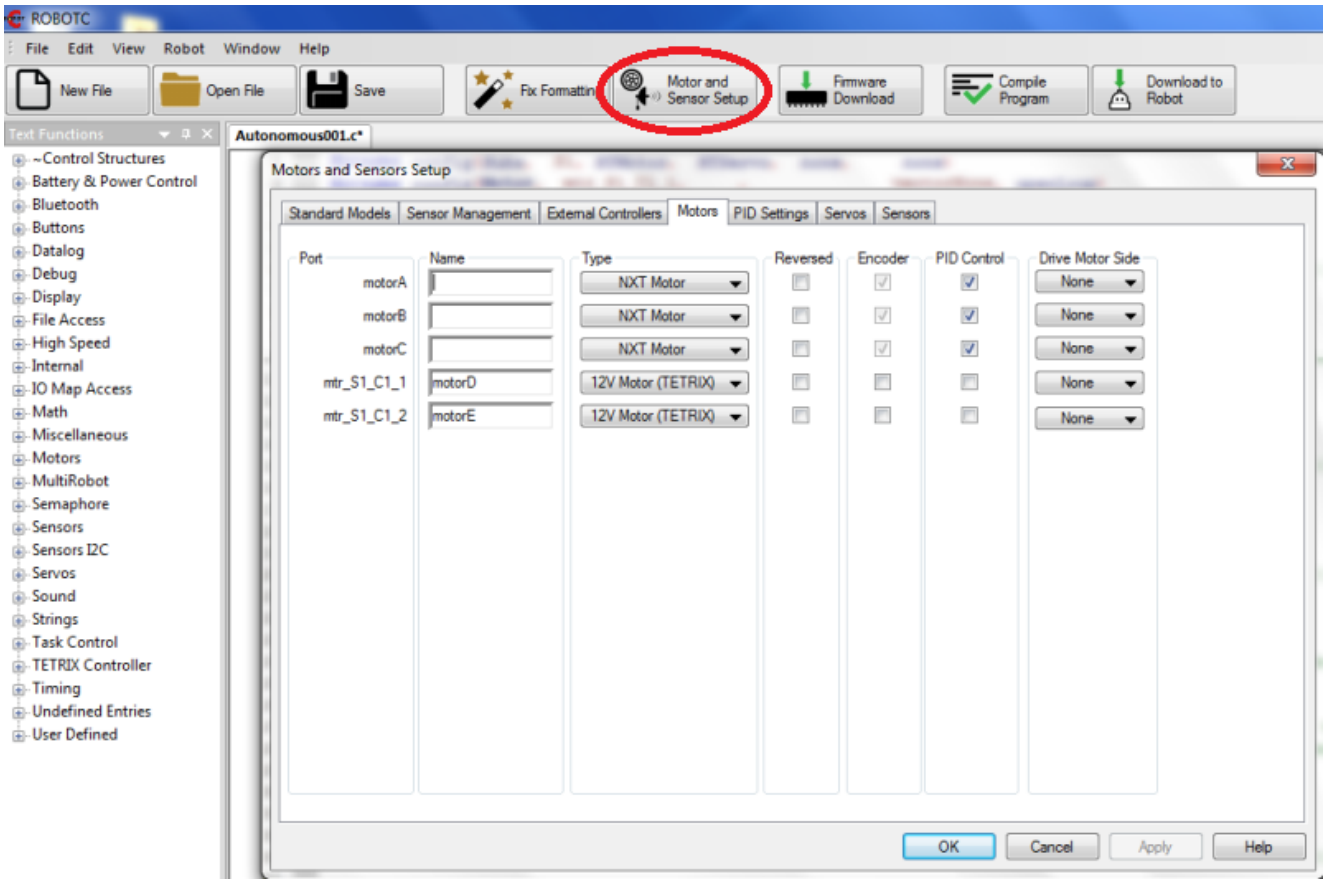


Creating an autonomous template with ROBOTC

The next step is to configure the number and type of controllers and sensors. This will be demonstrated by creating an autonomous template.

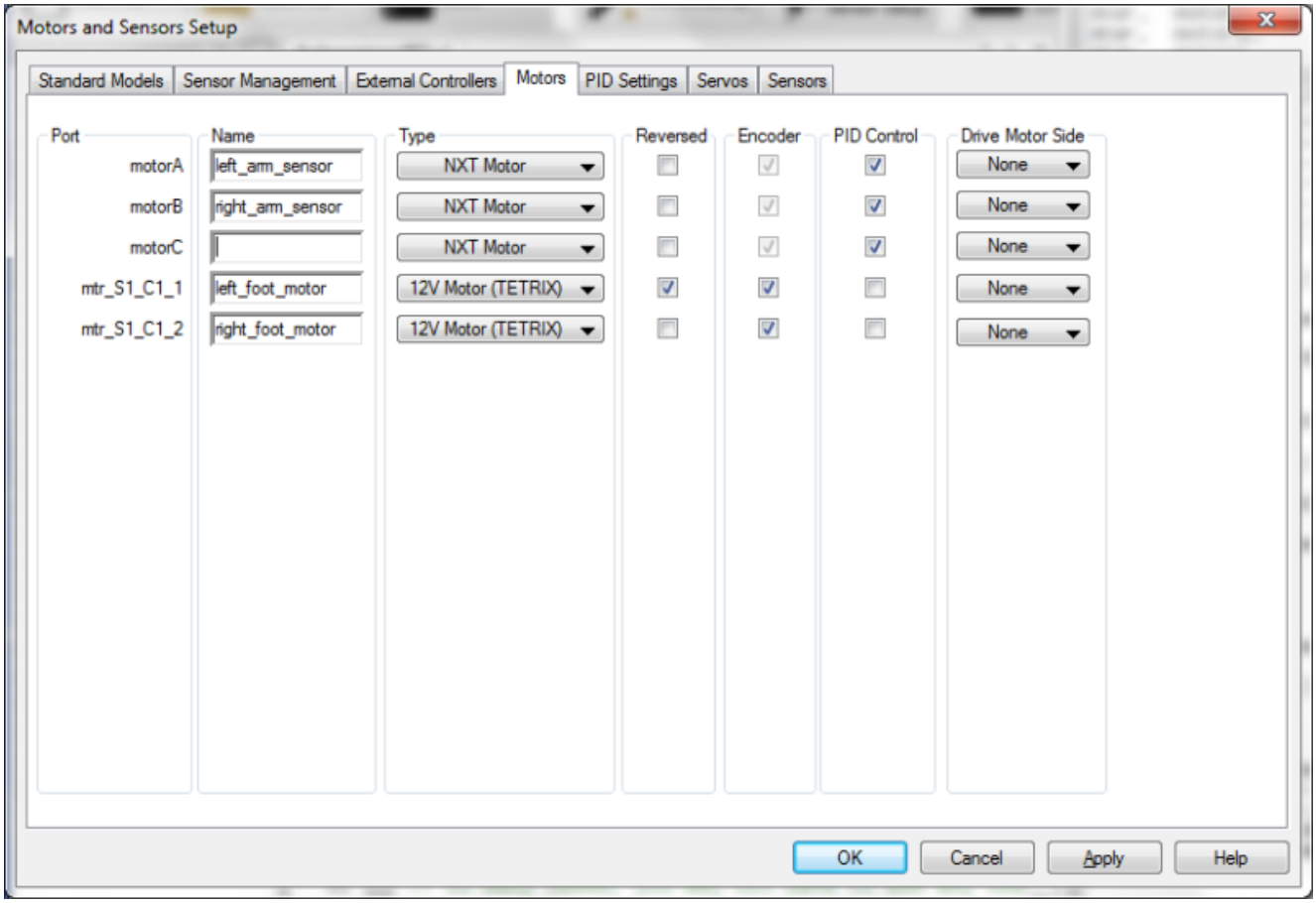


Next, select the Motor and Sensor Setup button on the icon ribbon (see the red circle in the image below). Notice that the Motors tab is selected. It shows that there are 3 LEGO motors and 2 DC motors. This is because the "Standard TETRIX Configuration. One TETRIX Motor Controller and Servo Controller on sensor port #1" radio button is selected on the External Controllers tab.. The Push Bot uses this configuration. If you change the number of motor or servo controllers on your robot, then you will need to update the configuration using this dialog.



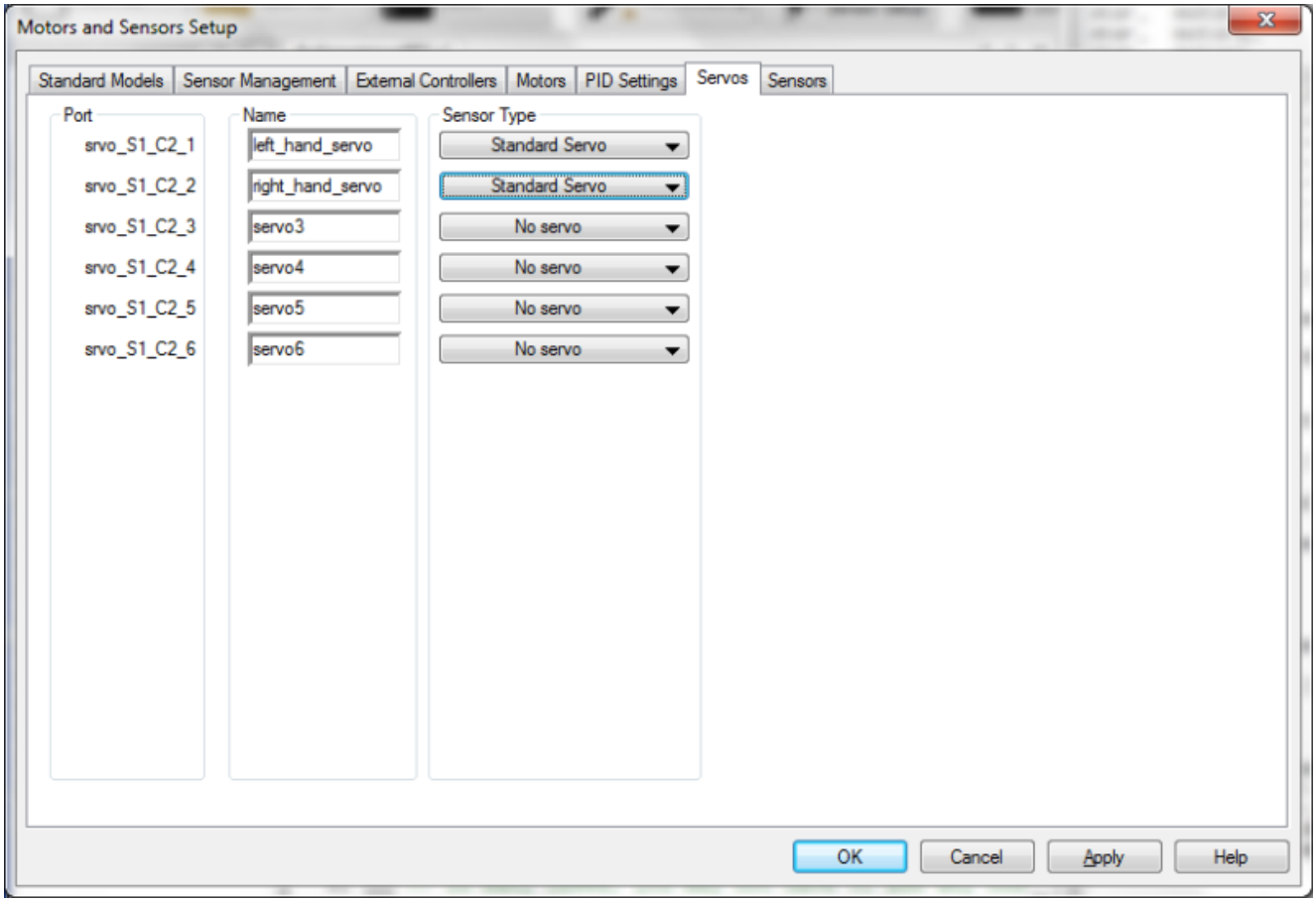
Enter information into the dialog. After the information is entered, the dialog should be the same as the image below.

- motorA – left_arm_motor, NXT Motor, PID Control
- motorB – right_arm_motor, NXT Motor, PID Control
- motorC – leave this empty
- mtr_S1_C1_1 – left_foot_motor, 12V Motor (TETRIX), Reversed, Encoder
- mtr_S1_C1_2 – right_foot_motor, 12V Motor (TETRIX), Encoder



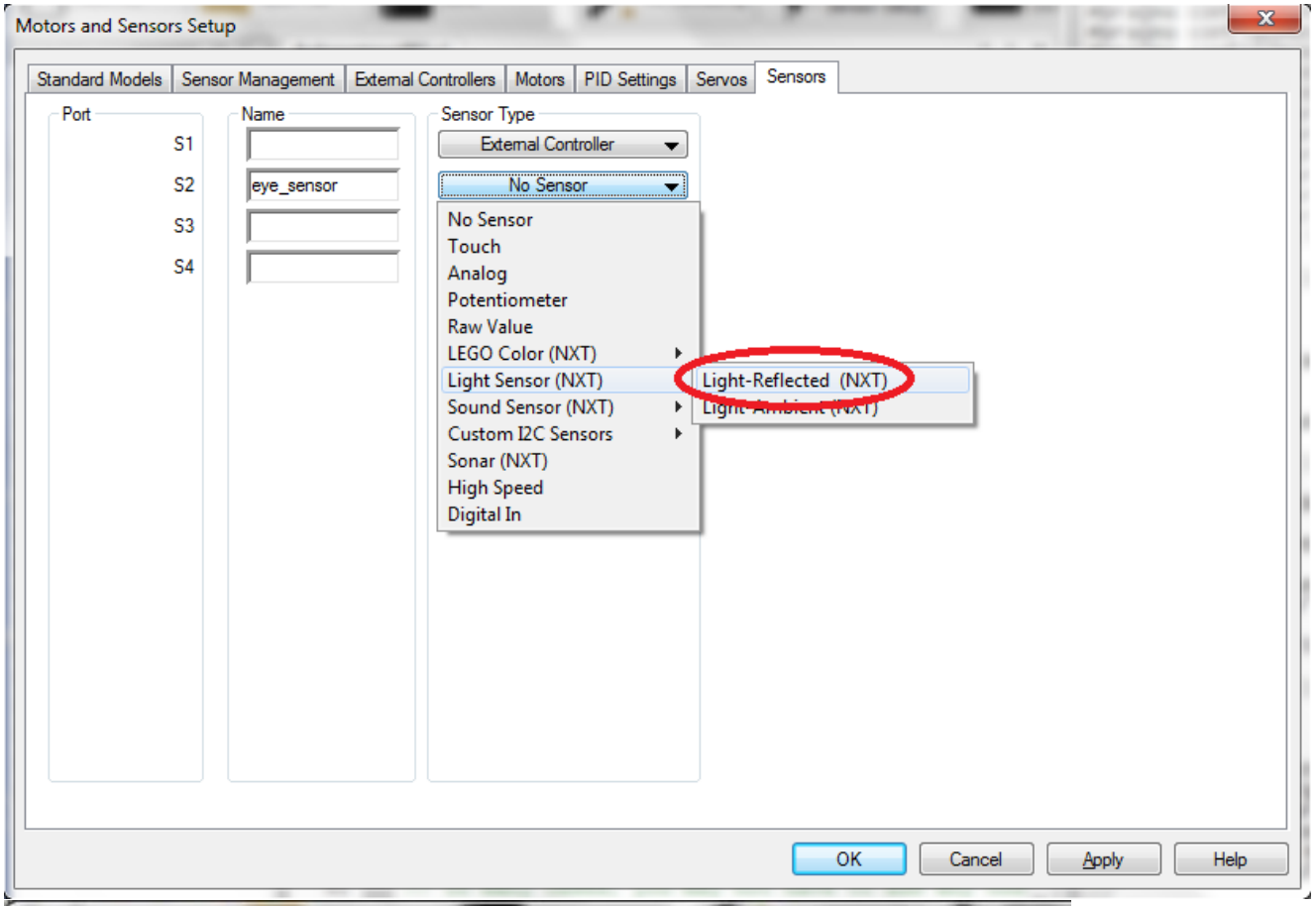
Select the Servos tab. Enter information into the dialog. After the information is entered, the dialog should be the same as the image below.

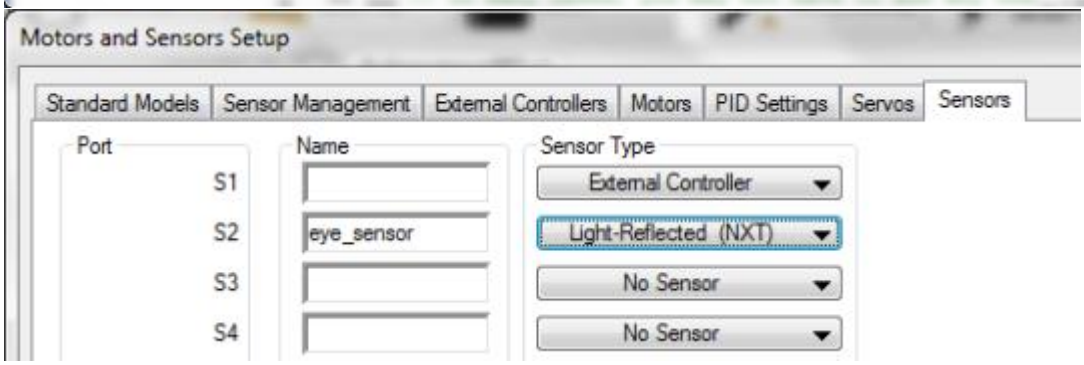
- srvo_S1_C2_1 – left_hand_servo, Standard Servo
- srvo_S1_C2_2 – right_hand_servo, Standard Servo



Select the Sensors tab. Enter information into the dialog. After the information is entered, the dialog should be the same as the image below.

- srvo_S1_C2_1 – left_hand_servo, Standard Servo
- srvo_S1_C2_2 – right_hand_servo, Standard Servo

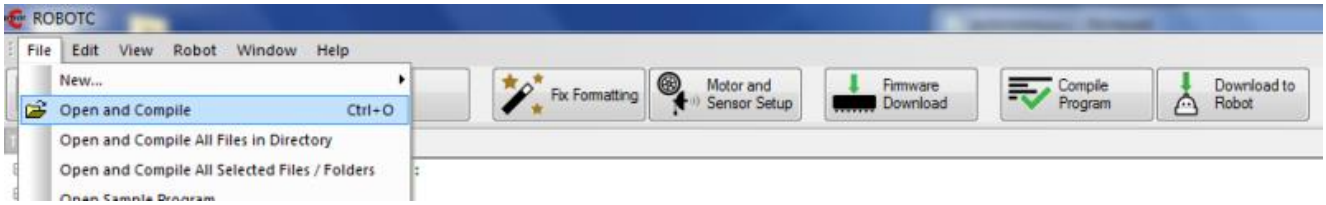




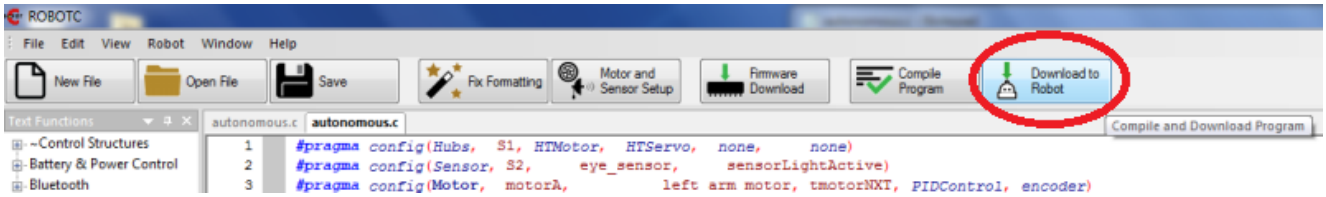
The autonomous template now has new code at the top of the file. To save the file for later use, click File and then "Save As". Save the file – this document saves the files as autonomous.c (see appendix B). The complete autonomous code is shown in appendices C-G. Once this code is downloaded onto computer, open autonomous.c.

Downloading code to the NXT

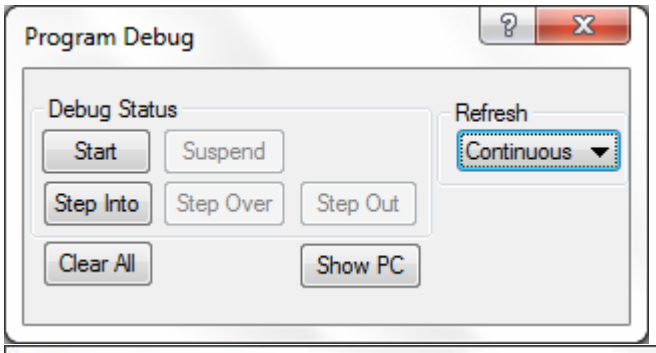
Open the autonomous.c file provided with this document by clicking File | Open and Compile. There will be some compile warning concerning unreferenced functions. These can be safely ignored when compiling the autonomous code.

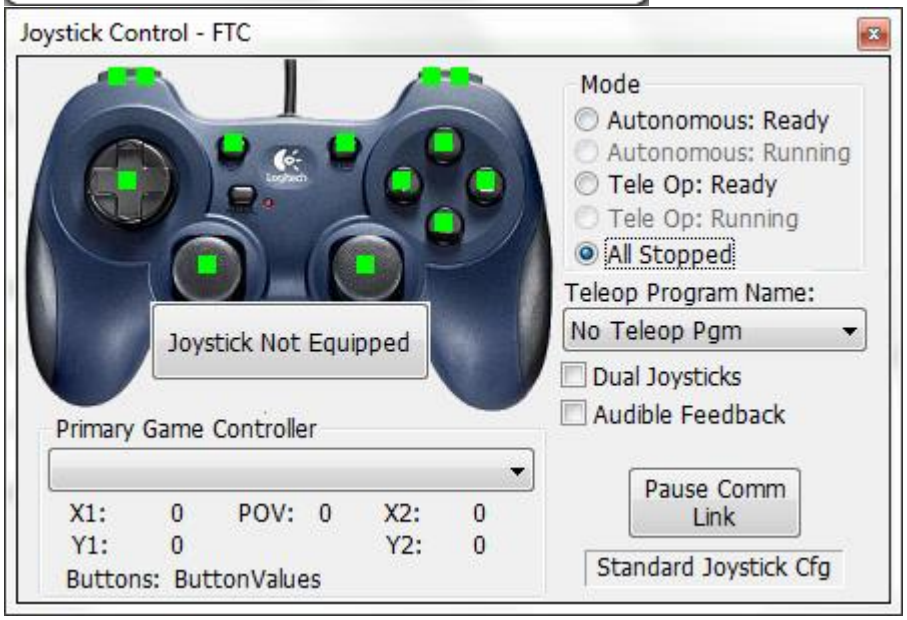


Click the “Download to Robot” button on the icon ribbon.



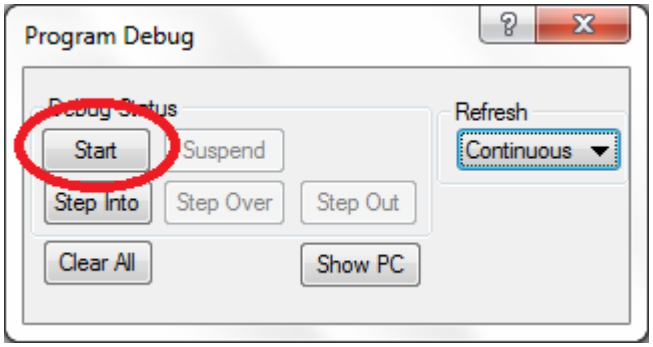
When the software has been downloaded to the NXT, the Program Debug and Joystick Control windows will be displayed.





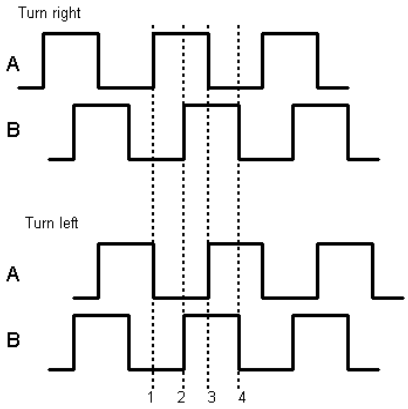
Enable the second joystick by clicking on the Dual Joysticks checkbox of the Joystick Control dialog.

Then select 'start' to run the code. NOTE - Ensure the battery switch is turned 'ON' on the robot.



Motor Encoders.

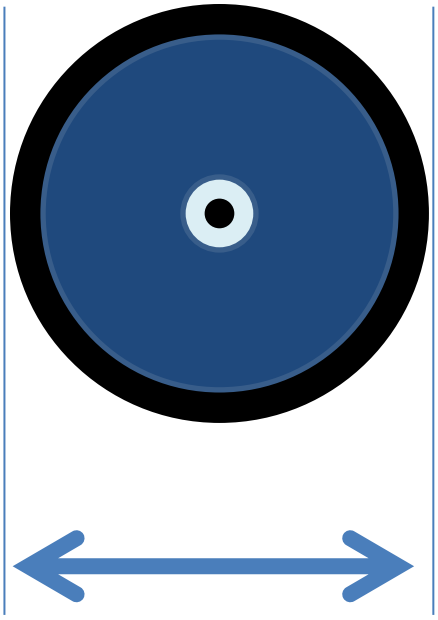
The distance travelled by the robot is dependent on the DC battery power level when using 'wait for time' commands. This is not ideal when the robot is required to travel a specific distance every time. To solve this problem motor encoders are used.



Motor Encoders are used to 'count' how far the motors have turned. The LEDs and photodiodes produce a pulse of square waves which the robot 'counts' to work out the number of rotations travelled.

The TETRIX Motor Encoders use a count of 1440 'clicks' for every 1 motor rotation. LEGO motors use 360 counts per revolution.

Calculating Distance using Motor Encoders



Diameter 3"

To calculate the distance travelled by 1 motor rotation, the robot wheel circumference is required. For TETRIX Robots, the 'Large Wheel' diameter is 3 inches.

Using: Wheel Circumference = ∏X Diameter

= 3.14 x 3.

The approximate wheel circumference is 9.42".

So 1440 'clicks' from the Motor Encoder is equal to 9.42 inches of distance travelled by the robot.

Driving forward using DC motor encoders.

This code uses a While Loop to count 1440 encoder clicks or 9.42" of distance travelled while the robot travels forward at speed 30. Once the 1440 'clicks' have been counted the motors are set to speed 0 or stop.

```
//
// Reset the encoders to zero.
//
nMotorEncoder[left_foot_motor] = 0;
nMotorEncoder[right_foot_motor] = 0;

//
// Power the motors at 30% until the encoders reach 1 revolution.
//
while (
    (nMotorEncoder[left_foot_motor] < 1440) &&
    (nMotorEncoder[right_foot_motor] < 1440))
{
    motor[left_foot_motor] = 30;
    motor[right_foot_motor] = 30;
}

//
// Stop the motors.
//
motor[left_foot_motor] = 0;
motor[right_foot_motor] = 0;
```

Driving backward using DC motor encoders.

The condition of the while loop (from "< 1440" to "> -1440") and the motor power levels (from "= 30" to "= -30") can be changed to drive the robot backward.

```
//
// Reset the encoders to zero.
//
nMotorEncoder[left_foot_motor] = 0;
nMotorEncoder[right_foot_motor] = 0;

//
// Power the motors at 30% until the encoders reach 1 revolution.
//
while (
    (nMotorEncoder[left_foot_motor] > -1440) &&
    (nMotorEncoder[right_foot_motor] > -1440))
{
    motor[left_foot_motor] = -30;
    motor[right_foot_motor] = -30;
}

//
// Stop the motors.
//
motor[left_foot_motor] = 0;
motor[right_foot_motor] = 0;
```

Turning using the DC motor encoders.

Modify the conditions and motor powers, such that one side (i.e. left) is positive and the other side is negative (right).

```
//
// Reset the encoders to zero.
//
nMotorEncoder[left_foot_motor] = 0;
nMotorEncoder[right_foot_motor] = 0;

//
// Power the motors at 30% until the encoders reach 1 revolution.
//
while (
    (nMotorEncoder[left_foot_motor] < 1440) &&
    (nMotorEncoder[right_foot_motor] > -1440))
{
    motor[left_foot_motor] = -30;
    motor[right_foot_motor] = 30;
}

//
// Stop the motors.
//
motor[left_foot_motor] = 0;
motor[right_foot_motor] = 0;
```

By using the forward, reverse and turning programs above, simple robot movements can be created. See the functions inside the BasicFunctions.h file. The drive_both_wheels, TurnRight, TurnLeft functions use the DC motor encoders to drive the robot to a specified distance. The move_arm

function accepts three different values to raise the arm, lower the arm, and to stop the arm if it is in motion (this last function is more appropriate to manual.c, but is beyond the scope of this document).

The program below, shows how the 'blocks' can be combined to create an autonomous program.

```
//
// Raise the arm.  Make sure that the arm is in its lowest position before
// running this autonomous program.
//
raise_arm ();

//
// Drive forward twelve inches.
//
drive_both_wheels (75, 75, 12 * kInch);

//
// Rotate right 90 degrees.  This will cause one motor to turn in one
// direction, but the other will turn in the opposite direction.
//
TurnRight (90 * kDegrees);

//
// Drive forward twelve inches.
//
drive_both_wheels (75, 75, 12 * kInch);

//
// Rotate right 90 degrees.  This will cause one motor to turn in one
// direction, but the other will turn in the opposite direction.
//
TurnRight (90 * kDegrees);

//
// Drive forward until a white line is found.
//
find_line ();

//
// Lower the arm and close the hand.
//
lower_arm ();
move_hand (e_hand_command_close);

//
// Drive forward twelve inches.
//
drive_both_wheels (75, 75, 12 * kInch);

//
// Rotate left 90 degrees.
//
TurnLeft (90 * kDegrees);
```

Using Sensors

A sensor is a converter that measures a physical quantity and converts it into a signal which can be read by the robot.

The TETRIX robot has been fitted with 1 sensor: the LEGO NXT light sensor. The function find_line function inside light.h drives the robot backwards until a white line is found.

```
void find_line (void)

{
    motor[c_left_foot_motor] = -75;
    motor[c_right_foot_motor] = -75;

    while (motor[c_left_foot_motor] != 0)
    {
        if (SensorValue (c_eye_sensor) > c_eye_value)
        {
            motor[c_left_foot_motor] = 0;
            motor[c_right_foot_motor] = 0;
        }
    }

} // find_line
```

This code can be adapted to drive forward by changing the sign of the number being written to the motor variables.

```
void find_line (void)

{
    motor[c_left_foot_motor] = -75;
    motor[c_right_foot_motor] = -75;

    while (motor[c_left_foot_motor] != 0)
    {
        if (SensorValue (c_eye_sensor) > c_eye_value)
        {
            motor[c_left_foot_motor] = 0;
            motor[c_right_foot_motor] = 0;
        }
    }

} // find_line
```

Servo Control using Robot C

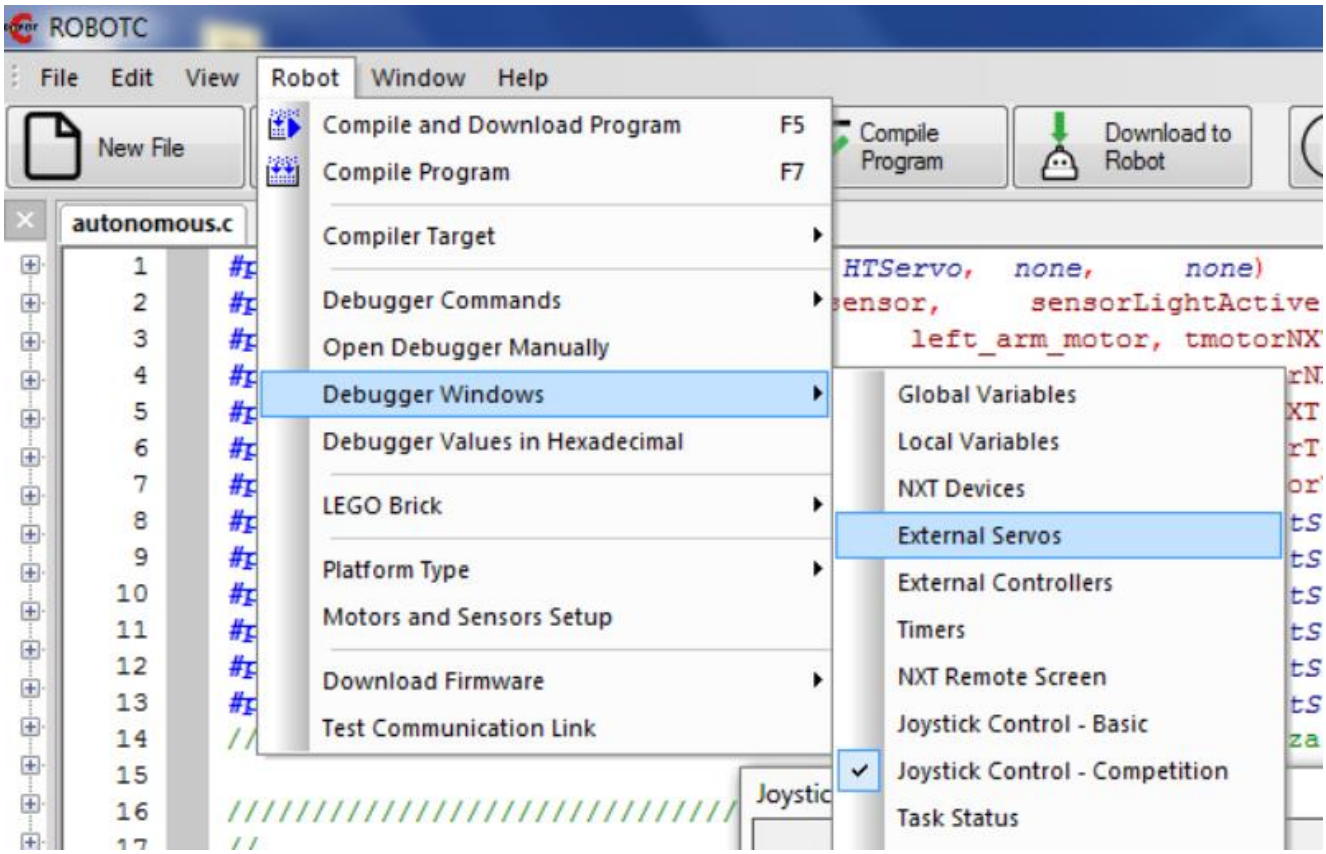
Servos are constructed from three basic pieces; a motor, a potentiometer (variable resister) that is connected to the output shaft, and a control board. The potentiometer allows the control circuitry to monitor the current angle of the servo motor. The motor, through a series of gears, turns the output shaft and the potentiometer simultaneously. The potentiometer is fed into the servo control circuit and when the control circuit detects that the position is correct, it stops the motor. If the control circuit detects that the angle is not correct, it will turn the motor the correct direction until the angle is correct. Normally a servo is used to control an angular motion of between 0 and 180 degrees. It is not

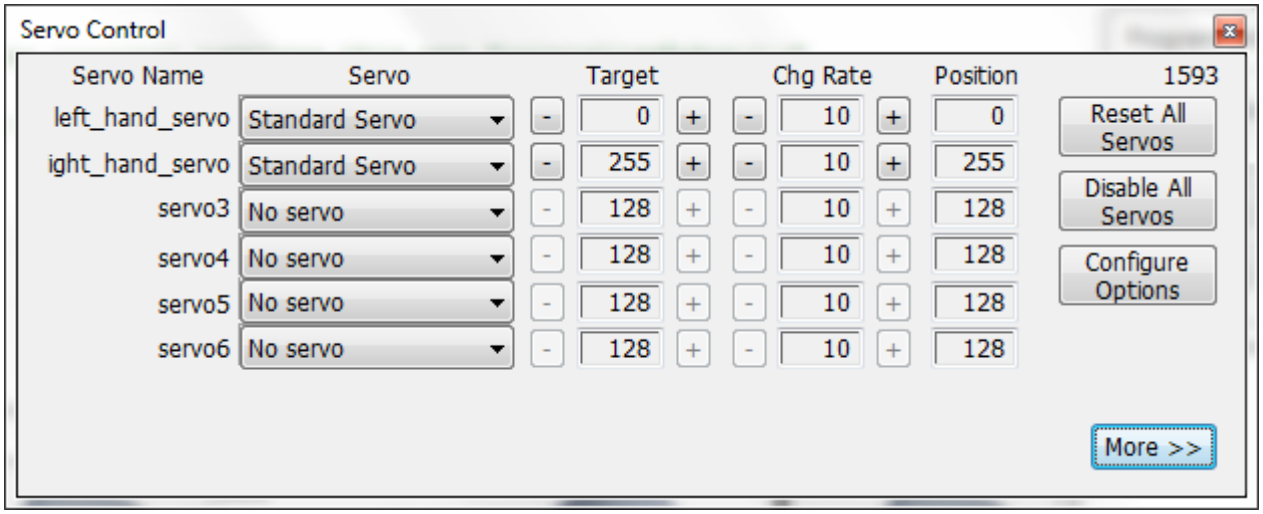
mechanically capable (unless modified) of turning any farther due to the mechanical stop build on to the main output gear.

The output values from servos range from 0 to 255 (Approximately 0.7 servo value per 1 degree of rotation). Servos need a small amount of time to reach their position so 'wait for time' commands are used after setting a servo position. Two servos are used in the Robot, one for the arm and the other for the claw. The arm servo has been geared down to allow for the weight of the claw and objects that it picks up.

Setting up Servo positions

ROBOTC has a Servo 'debug' tool which allows the servo positions to be adjusted and recorded. Open the Servo Debugging Window



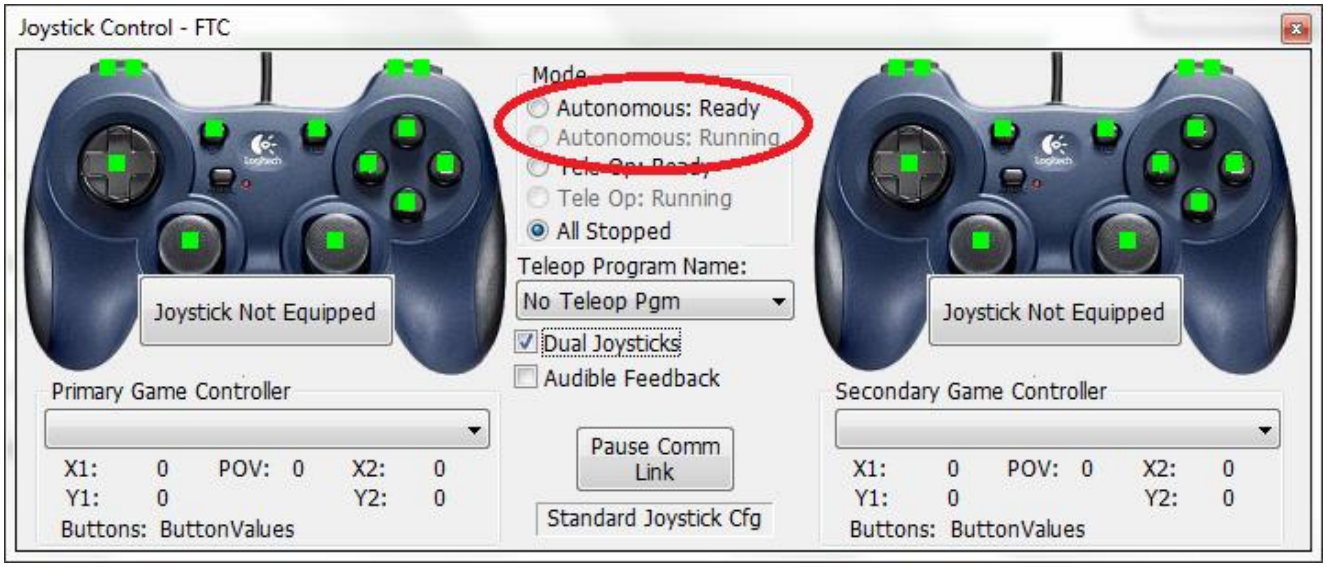


This window allows the servos to be manually moved and the positions recorded. The hand servos may need to be adjusted to get the maximum movement range. The 'left_hand_servo' should be zero when it is fully open. The 'right_hand_servo' should be 255 when fully open. When fully closed, the 'left_hand_servo' should be 180 and the 'right_hand_servo' should be 75 (see the constants in the constants.h file).

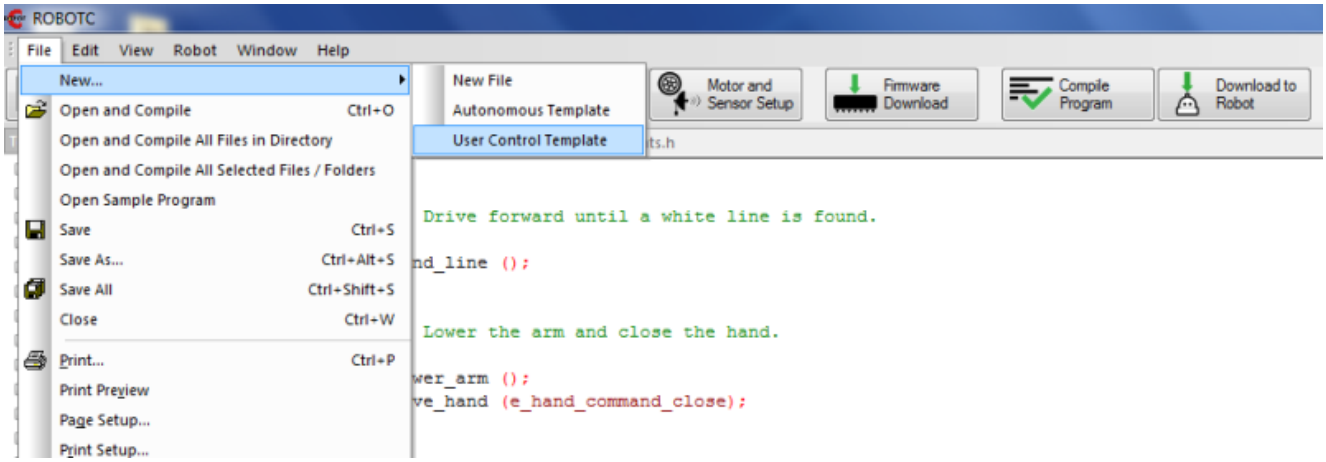
The initialize.h file contains a function called initialize. It moves the servos to the fully open position, so the robot will fit inside the 18" cube, which is required before the match can begin. All initialization code that moves the servos at start-up requires a warning label, as shown below (see Game Manual Part 1). The label must be placed near the robot's main power switch.

Running autonomous code.

To start the autonomous routine, use the joystick control dialog. Click the 'Autonomous: Ready' radio button and then click the 'Autonomous: Running' radio button.



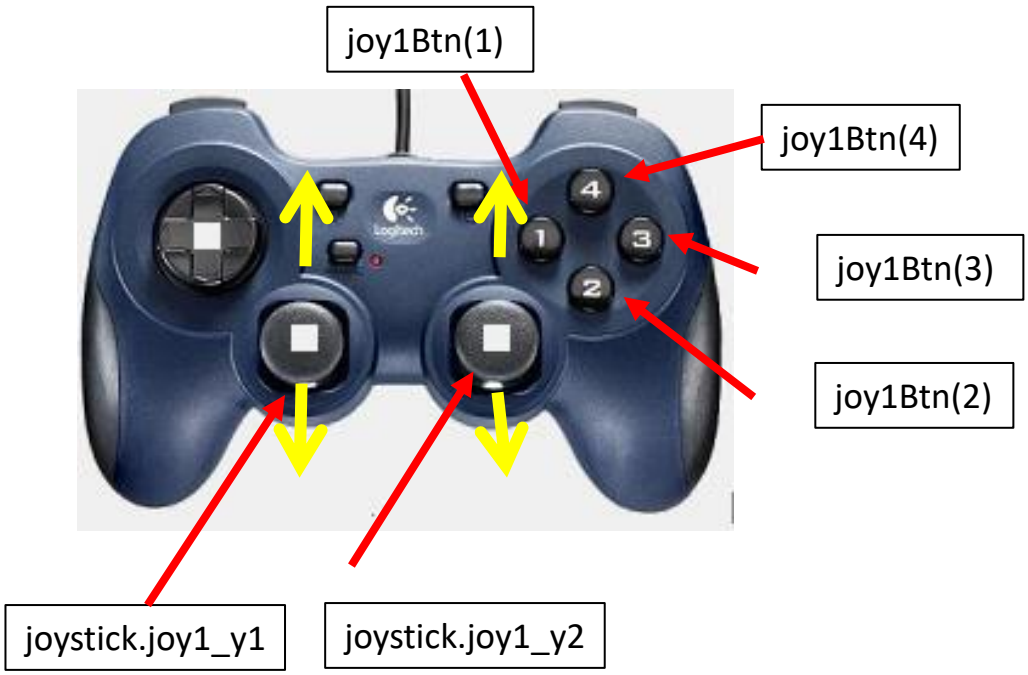
Creating a manual template with ROBOTC



The "#pragma" statements at the top of the autonomous.c file are the same for the purposes of this document as those in the manual.c file. Appendix I contains the manual (aka tele-op) code for the Push Bot. Compile and download manual.c to the NXT in the same fashion as the autonomous.c file. If you change the configuration of the motors, servos, or sensors, then use the Motor and Sensor Setup dialog to make the necessary changes to the autonomous.c and manual.c files.

Joypad Controllers (Logitech F310)

The image below outlines the button layout and the related button 'names' used in ROBOTC



First Joystick Controller Readings

joystick.joy1_y1 - Value of the Y-Axis on the Left Joystick on Controller #1. Range: -128 to +127.

joystick.joy1_y2 - Value of the Y-Axis on the Right Joystick on Controller #1. Range: -128 to +127.

These values are converted into -100 to +100 values in the convert_joystick_to_motor function inside motor.h, which is included into manual.c. To prevent the motors from receiving a very small amount

of power when the joysticks aren't being actively pushed, the values between -28 and +28 are ignored and the motor power is set to zero. This is called a dead zone and it prevents motor burnout.

Second Joystick Controller Setup

Joystick 2 is configured using the same terms but changing joy1Btn() to joy2Btn() or joystick.joy1_yX to joystick.joy2_yX.

Notice that button 6 and 8 of the second controller operate the hand (see the "Manage the hand servo positions" block inside the main function inside manual.c. Buttons 5 and 7 also operate the hand, but only partially open or close the hand.

FTC NXT Pre-Game Setup

Before an FTC match, Teams need to choose a program to run for the autonomous section of the game and a program to be run in the user control stage. The User Control stage is selected via the Program Chooser.

The Program Chooser

The Program Chooser is a program that allows users to assign programs on the NXT brick for tele-op use. Because autonomous programs are chosen before the start of an FTC match, the program chooser enables mixing-and-matching of programs and helps users account for game variables such as start location, alliance members, opposing alliance members, game object positions, etc. before a match starts.

On your NXT run the "Program Chooser" program. Go to the 'Try Me' Menu and scroll across until you find 'Program Chooser'. Select it with the orange button.

This program will prompt you to choose which of the programs on your NXT is your Teleop program. Use the two left/right arrows on your NXT brick to select your teleop program. Then hit the orange button. The program will prompt you to hit the orange button to confirm again, and then the program will exit.

Connect your robot to your FCS, and on your NXT brick choose your autonomous program and run it. Your robot will not make any actions at this time.

Start the match on your FCS. Your robot should now perform its autonomous program and then halt when the program is through, or the 30 seconds are up.

Your FCS will now pause between auto and teleop modes and display 2 minutes on the clock. Your NXT will automatically run the teleop program on its own without any input by you.

Continue the match on the FCS and you should have teleop control of your robot now.

The autonomous program is selected via the normal 'My Files', Software Files' Menus.

Appendix A: Bill of Materials and Tools List

Count	Part
6	CHANNEL_288_2011
9	BRACKET_L-ANGLE_2011
3	LEGO_HPC_2011
94	SCREW_SHCS_6-32_X_0_3125_2011
73	SCREW_SHCS_6-32_X_0_5_2011
148	KEPS_NUT_6-32_2011
2	CHANNEL_160_2011
2	CHANNEL_96_2011
2	DC_MOTOR_MOUNT_2011
2	SCREW_SHCS_6-32_X_1_5_2011
10	SCREW_SHCS_6-32_X_1_25_2011
2	WHEEL_76MM_3_INCH_2011
2	HUB_SPACER_2011
4	GEAR_80_2011
10	HUB_D-SHAFT_2011
6	HUB_D-SHAFT_SCREW_2011
15	NYLON_SPACER_SMALL_2011
14	BRONZE_BUSHING_2011
5	AXLE_D-SHAFT_2011
10	AXLE_SET_COLLAR_2011
4	OMNI_WHEEL_3_IN_HUB_2011
20	OMNI_WHEEL_3_IN_ROLLER_2011
12	OMNI_WHEEL_3_IN_SCREW_2011
1	BEAM_7_2011
4	CONN_3_2011

1	SENSOR_LIGHT_2011
1	NYLON_SPACER_LARGE_2011
1	CHANNEL_416MM
2	BRACKET_SERVO_SINGLE_2011
2	BEAM_9_2011
8	CONN_AXLE_FEMALE_2011
2	NXT_MOTOR_2011
4	NXT_MOTOR_WHEEL_2011
6	GEAR_40_2011
2	GEAR_METRIC_120_2011
2	PLATE_64MM_X_192MM_2011
1	BRACKET_SERVO_DOUBLE_2011
1	BRACKET_SERVO_PIVOT_MULTI_2011
9	SCREW_BHCS_6-32_X_0_375_2011
2	SERVO_2011
2	SERVO_METAL_HORN_2011
6	BRACKET_FLAT_2011
2	CHANNEL_32_2011
2	DC_MOTOR_2011
2	HUB_MOTOR_SHAFT_2011
1	TUBE_CLAMP_2011
1	TUBING_PLUG_2011
1	TUBE_80MM_2011
6	BUSHING_2011
2	PULLEY_2011
2	PULLEY_TYRE_2011
2	AXLE_10_2011

Appendix B: Initial Autonomous.c

```
#pragma config(Hubs,  S1, HTMotor,  HTServo,  none,     none)
#pragma config(Sensor, S2,     eye_sensor,     sensorLightActive)
#pragma config(Motor,  motorA,          left_arm_motor, tmotorNXT, PIDControl, encoder)
#pragma config(Motor,  motorB,          right_arm_motor, tmotorNXT, PIDControl, encoder)
#pragma config(Motor,  mtr_S1_C1_1,     left_foot_motor, tmotorTetrix, openLoop, reversed, encoder)
#pragma config(Motor,  mtr_S1_C1_2,     right_foot_motor, tmotorTetrix, openLoop, encoder)
#pragma config(Servo,  srvo_S1_C2_1,    left_hand_servo,      tServoStandard)
#pragma config(Servo,  srvo_S1_C2_2,    right_hand_servo,     tServoStandard)
#pragma config(Servo,  srvo_S1_C2_3,    servo3,               tServoNone)
#pragma config(Servo,  srvo_S1_C2_4,    servo4,               tServoNone)
#pragma config(Servo,  srvo_S1_C2_5,    servo5,               tServoNone)
#pragma config(Servo,  srvo_S1_C2_6,    servo6,               tServoNone)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

////////////////////////////////////////////////////////////////////////////////////////////////////
//
//                           Autonomous Mode Code Template
//
// This file contains a template for simplified creation of an autonomous program for an TETRIX robot
// competition.
//
// You need to customize two functions with code unique to your specific robot.
//
////////////////////////////////////////////////////////////////////////////////////////////////////

#include "JoystickDriver.c"  //Include file to "handle" the Bluetooth messages.


////////////////////////////////////////////////////////////////////////////////////////////////////
//
//                                    initializeRobot
//
// Prior to the start of autonomous mode, you may want to perform some initialization on your robot.
// Things that might be performed during initialization include:
//   1. Move motors and servos to a preset position.
//   2. Some sensor types take a short while to reach stable values during which time it is best that
//      robot is not moving. For example, gyro sensor needs a few seconds to obtain the background
//      "bias" value.
//
// In many cases, you may not have to add any code to this function and it will remain "empty".
//
////////////////////////////////////////////////////////////////////////////////////////////////////

void initializeRobot()
{
  // Place code here to sinitialize servos to starting positions.
  // Sensors are automatically configured and setup by ROBOTC. They may need a brief time to stabilize.

  return;
}


////////////////////////////////////////////////////////////////////////////////////////////////////
//
//                                         Main Task
//
// The following is the main code for the autonomous robot operation. Customize as appropriate for
// your specific robot.
//
// The types of things you might do during the autonomous phase (for the 2008-9 FTC competition)
// are:
//
//   1. Have the robot follow a line on the game field until it reaches one of the puck storage
//      areas.
//   2. Load pucks into the robot from the storage bin.
//   3. Stop the robot and wait for autonomous phase to end.
//
// This simple template does nothing except play a periodic tone every few seconds.
//
// At the end of the autonomous period, the FMS will autonmatically abort (stop) execution of the
program.
//
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////

task main()
{
  initializeRobot();

  waitForStart(); // Wait for the beginning of autonomous phase.

  //////////////////////////////////////////////////////////
  //////////////////////////////////////////////////////////
  ////                                                  ////
  ////    Add your robot specific autonomous code here.  ////
  ////                                                  ////
  //////////////////////////////////////////////////////////
  //////////////////////////////////////////////////////////

  while (true)
  {}
}
```

Appendix C: constants.h

```
//-------------------------------------------------------------------------------
//
// constants.h
//
// This file provides variables that are used in both the autonomous and manual
// files.
//

//-------------------------------------------------------------------------------
//
// Define drive wheel (aka feet) constants.
//
const tMotor c_left_foot_motor = left_foot_motor;
const tMotor c_right_foot_motor = right_foot_motor;

//-------------------------------------------------------------------------------
//
// Define LEGO motor (aka arm) constants.
//
const tMotor c_left_arm_motor = left_arm_motor;
const tMotor c_right_arm_motor = right_arm_motor;

//-------------------------------------------------------------------------------
//
// Define servo motor (aka hand) constants.
//
const TServoIndex c_left_hand_motor = left_hand_servo;
const TServoIndex c_right_hand_motor = right_hand_servo;

const int c_left_hand_motor_fully_open_position = 0;
const int c_left_hand_motor_fully_closed_position = 180;
const int c_right_hand_motor_fully_open_position = 255;
const int c_right_hand_motor_fully_closed_position = 75;

//-------------------------------------------------------------------------------
//
// Define light sensor (aka eye) constants.
//
const tSensors c_eye_sensor = eye_sensor;
const int c_eye_value = 52;
```

Appendix D: BasicFunctions.h

```
/*!-----------------------------------------------------------------------------
//
// @file
//    BasicFunctions.h
//
// @brief
//    Contains all Tele-Operated code and some autonomous functions.
//
// @author RLS - Renee L. Spangler
//
// @version 2009 - 2010, updated 2014-08-24
//
// @remarks
//    This program has all of the robot devices' code except the autonomous
//    sequences.
*/

const float kInch = (((1440.) / (3.1415926535 * 4.)) * (3./1.95));
const float kDegrees = (2850. / 90.);

//-----------------------------------------------------------------------------
//
// drive_both_wheels
//
void drive_both_wheels (
    int a_left_power_and_direction,
    int a_right_power_and_direction,
    float a_counts)

{
    if (a_counts > 0)
    {
        int max_delay = 3;
        int right_last_encoder = 32767;
        int right_delay = 0;
        int right_encoder = 0;
        int left_last_encoder = 32767;
        int left_delay = 0;
        int left_encoder = 0;

        long limit = a_counts;

        nMotorEncoder[c_left_foot_motor] = 0;
        nMotorEncoder[c_right_foot_motor] = 0;
        wait1Msec (100);

        bool right_done = false;
        bool left_done = false;

        while ((right_done != true) || (left_done != true))
        {
            wait1Msec (30);
            if (right_done != true)
            {
                right_encoder = abs(nMotorEncoder[c_right_foot_motor]);
                if (right_encoder < limit)
                {
                    if (right_last_encoder != right_encoder)
                    {
                        right_delay = 0;
                        motor[c_right_foot_motor] = 100 * a_right_power_and_direction;
                        right_last_encoder = right_encoder;
                    }
```

```
                else if (right_delay > max_delay)
                {
                    motor[c_right_foot_motor] = 0;
                    right_done = true;
                    motor[c_left_foot_motor] = 0;
                    left_done = true;
                }
                else
                {
                    right_delay++;
                }
            }
            else
            {
                motor[c_right_foot_motor] = 0;
                right_done = true;
            }
        }
        else
        {
            motor[c_right_foot_motor] = 0;
        }

        if (left_done != true)
        {
            left_encoder = abs(nMotorEncoder[c_left_foot_motor]);
            if (left_encoder < limit)
            {
                if (left_last_encoder != left_encoder)
                {
                    left_delay = 0;
                    motor[c_left_foot_motor] = -85 * a_left_power_and_direction;
                    left_last_encoder = left_encoder;
                }
                else if (left_delay > max_delay)
                {
                    motor[c_left_foot_motor] = 0;
                    left_done = true;
                    motor[c_right_foot_motor] = 0;
                    right_done = true;
                }
                else
                {
                    left_delay++;
                }
            }
            else
            {
                motor[c_left_foot_motor] = 0;
                left_done = true;
            }
        }
        else
        {
            motor[c_left_foot_motor] = 0;
        }
    }
    motor[c_left_foot_motor] = 0;
    motor[c_right_foot_motor] = 0;
    nMotorEncoder[c_left_foot_motor] = 0;
    nMotorEncoder[c_right_foot_motor] = 0;
}
```

```
} // drive_both_wheels

//-----------------------------------------------------------------------------
//
// Turn Commands
//
void TurnRight (float Degrees)

{
    drive_both_wheels (1, -1, (Degrees * kDegrees));

} // TurnRight

void TurnLeft (float Degrees)

{
    drive_both_wheels (-1, 1, (Degrees * kDegrees));

} // TurnLeft

//-----------------------------------------------------------------------------
//
// move_arm
//
//-------
enum e_arm_command
{
    e_invalid_arm_command,

    e_arm_command_stop,
    e_arm_command_raise,
    e_arm_command_lower,

    e_count_arm_command
};

e_arm_command move_arm (e_arm_command a_command)

{
    e_arm_command l_command = a_command;

    switch (l_command)
    {
    case e_arm_command_stop:
    default:
        motor[c_left_arm_motor] = 0;
        motor[c_right_arm_motor] = 0;
        nMotorEncoder[c_left_arm_motor] = 0;
        nMotorEncoder[c_right_arm_motor] = 0;
        break;
    case e_arm_command_raise:
        if (nMotorEncoder[c_left_arm_motor] < -550)
        {
            motor[c_left_arm_motor] = 0;
            motor[c_right_arm_motor] = 0;
            nMotorEncoder[c_left_arm_motor] = 0;
            nMotorEncoder[c_right_arm_motor] = 0;
            l_command = e_arm_command_stop;
        }
        break;
    case e_arm_command_lower:
        if (nMotorEncoder[c_left_arm_motor] > 550)
        {
            motor[c_left_arm_motor] = 0;
```

```
            motor[c_right_arm_motor] = 0;
            nMotorEncoder[c_left_arm_motor] = 0;
            nMotorEncoder[c_right_arm_motor] = 0;
            l_command = e_arm_command_stop;
        }
        break;
    }

    return l_command;

} // move_arm

//-----------------------------------------------------------------------------
//
// lower_arm
//
//-------
void lower_arm (void)

{
    motor[c_left_arm_motor] = 75;
    motor[c_right_arm_motor] = 75;
    while (move_arm (e_arm_command_lower) != e_arm_command_stop)
    {
        displayTextLine (2, "%d   %d", nMotorEncoder[c_left_arm_motor],
nMotorEncoder[c_right_arm_motor]);
    };

} // lower_arm

//-----------------------------------------------------------------------------
//
// raise_arm
//
//-------
void raise_arm (void)

{
    motor[c_left_arm_motor] = -50;
    motor[c_right_arm_motor] = -50;
    while (move_arm (e_arm_command_raise) != e_arm_command_stop)
    {
        displayTextLine (2, "%d   %d", nMotorEncoder[c_left_arm_motor],
nMotorEncoder[c_right_arm_motor]);
    };

} // raise_arm

//-----------------------------------------------------------------------------
//
// move_hand
//
//-------
enum e_hand_command
{
    e_invalid_hand_command,

    e_hand_command_open,
    e_hand_command_close,
    e_hand_command_partially_open,
    e_hand_command_partially_close,

    e_count_hand_command
```

```
};

void move_hand (e_hand_command a_command)

{
    //
    // These variables help control the hand servo motors.
    //
    static int left_hand_motor_position = c_left_hand_motor_fully_open_position;
    static int right_hand_motor_position = c_right_hand_motor_fully_open_position;

    switch (a_command)
    {
    case e_hand_command_open:
        left_hand_motor_position = c_left_hand_motor_fully_open_position;
        right_hand_motor_position = c_right_hand_motor_fully_open_position;
        break;
    case e_hand_command_close:
        left_hand_motor_position = c_left_hand_motor_fully_closed_position;
        right_hand_motor_position = c_right_hand_motor_fully_closed_position;
        break;
    case e_hand_command_partially_open:
        //
        // This increment block allows the user to control how far open or
        // shut the claws are.  This block opens the claws by an increment.
        // The increment is set to 2 here, but if you want the claw to
        // open more quickly then set it to a higher number here.  Alternately,
        // decrease the wait and the end of the while (true) block.
        //
        left_hand_motor_position = left_hand_motor_position - 2;
        right_hand_motor_position = right_hand_motor_position + 2;
        break;
    case e_hand_command_partially_close:
        left_hand_motor_position = left_hand_motor_position + 2;
        right_hand_motor_position = right_hand_motor_position - 2;
        break;
    }

    //
    // Make sure that the servo values are within acceptable limits.
    //
    if (left_hand_motor_position < c_left_hand_motor_fully_open_position)
    {
        left_hand_motor_position = c_left_hand_motor_fully_open_position;
    }
    else if (left_hand_motor_position > c_left_hand_motor_fully_closed_position)
    {
        left_hand_motor_position = c_left_hand_motor_fully_closed_position;
    }

    if (right_hand_motor_position > c_right_hand_motor_fully_open_position)
    {
        right_hand_motor_position = c_right_hand_motor_fully_open_position;
    }
    else if (right_hand_motor_position < c_right_hand_motor_fully_closed_position)
    {
        right_hand_motor_position = c_right_hand_motor_fully_closed_position;
    }

    //
    // Command the servos to a position.
    //
    servoTarget[c_left_hand_motor] = left_hand_motor_position;
    servoTarget[c_right_hand_motor] = right_hand_motor_position;
```

```
} // move_hand
```

Appendix E: light.h

```
//-------------------------------------------------------------------------
//
// light.h
//
// This file provides functions that relate to light sensor.
//

//-------------------------------------------------------------------------
//
// found_line
//
//--------
// Call this function every iteration.  Provide it with value that represents
// the average between a value when the light sensor is placed over a light
// object and the value when placed over a dark object.  This number will vary
// depending on the environment light conditions.  Calibrate the light sensor
// before the competition and during lunch at the competition if the field
// personnel allow it.
//--------
void find_line (void)

{
    motor[c_left_foot_motor] = -75;
    motor[c_right_foot_motor] = -75;

    while (motor[c_left_foot_motor] != 0)
    {
        if (SensorValue (c_eye_sensor) > c_eye_value)
        {
            motor[c_left_foot_motor] = 0;
            motor[c_right_foot_motor] = 0;
        }
    }

} // find_line
```

Appendix F: initialize.h

```
//-------------------------------------------------------------------------
//
// initialize
//
//-------
// Initialize all motors, sensors, and internal variables.
//--------
void initialize (void)

{
    //
    // This allows the LEGO motors to maintain arm position.  Without it, the
    // arm might be too heavy and will slowly rotate downwards unless the driver
    // continually applies power to the LEGO motors.
```

```
    //
    bFloatDuringInactiveMotorPWM = false; // Brake the motors

    //
    // Initialize the foot motors.
    //
    nMotorEncoder[c_left_arm_motor] = 0;
    nMotorEncoder[c_right_foot_motor] = 0;

    //
    // Initialize the arm motors.
    //
    nMotorEncoder[c_left_arm_motor] = 0;
    nMotorEncoder[c_right_arm_motor] = 0;

    //
    // Initialize the hand motors.
    //
    // As part of initialization, move the servos to the open position.  The
    // servos need to be open, so the robot remains less than 18 inches deep.
    //
    move_hand (e_hand_command_open);

    //
    // Set the amount of clicks required to turn the NXT (aka brick) off.  This
    // prevents the NXT from being accidentally turned off when the robot hits a
    // game element, field element or another robot.
    //
    nNxtExitClicks = 3;

} // initialize
```

Appendix G: autonomous.c (Complete)

```
#pragma config(Hubs,  S1, HTMotor,  HTServo,  none,     none)
#pragma config(Sensor, S2,     eye_sensor,     sensorLightActive)
#pragma config(Motor,  motorA,          left_arm_motor, tmotorNXT, PIDControl, encoder)
#pragma config(Motor,  motorB,          right_arm_motor, tmotorNXT, PIDControl, encoder)
#pragma config(Motor,  motorC,           ,             tmotorNXT, openLoop)
#pragma config(Motor,  mtr_S1_C1_1,     left_foot_motor, tmotorTetrix, openLoop, reversed)
#pragma config(Motor,  mtr_S1_C1_2,     right_foot_motor, tmotorTetrix, openLoop)
#pragma config(Servo,  srvo_S1_C2_1,    left_hand_servo,      tServoStandard)
#pragma config(Servo,  srvo_S1_C2_2,    right_hand_servo,     tServoStandard)
#pragma config(Servo,  srvo_S1_C2_3,    servo3,               tServoNone)
#pragma config(Servo,  srvo_S1_C2_4,    servo4,               tServoNone)
#pragma config(Servo,  srvo_S1_C2_5,    servo5,               tServoNone)
#pragma config(Servo,  srvo_S1_C2_6,    servo6,               tServoNone)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

////////////////////////////////////////////////////////////////////////////////////
///////
//
//                           Autonomous Mode Code Template
//
// This file contains a template for simplified creation of an autonomous program for an
TETRIX robot
// competition.
//
// You need to customize two functions with code unique to your specific robot.
//
```

```
////////////////////////////////////////////////////////////////////////////////////////////
///////

#include "JoystickDriver.c"  //Include file to "handle" the Bluetooth messages.

#include "constants.h"  // Must be included before other user-defined headers.

#include "BasicFunctions.h"
#include "light.h"

#include "initialize.h" // Must be included after other user-defined headers.

////////////////////////////////////////////////////////////////////////////////////////////
///////
//
//                                    initializeRobot
//
// Prior to the start of autonomous mode, you may want to perform some initialization on your
robot.
// Things that might be performed during initialization include:
//   1. Move motors and servos to a preset position.
//   2. Some sensor types take a short while to reach stable values during which time it is
best that
//      robot is not moving. For example, gyro sensor needs a few seconds to obtain the
background
//      "bias" value.
//
// In many cases, you may not have to add any code to this function and it will remain
"empty".
//
////////////////////////////////////////////////////////////////////////////////////////////
///////

#ifdef d_code_moved
This template function has been moved to the constants.h header file, so it can
be used for both manual and autonomous purposes.
void initializeRobot()
{

} // initializeRobot
#endif d_code_moved

////////////////////////////////////////////////////////////////////////////////////////////
///////
//
//                                        Main Task
//
// The following is the main code for the autonomous robot operation. Customize as appropriate
for
// your specific robot.
//
// The types of things you might do during the autonomous phase (for the 2008-9 FTC
competition)
// are:
//
//   1. Have the robot follow a line on the game field until it reaches one of the puck
storage
//      areas.
//   2. Load pucks into the robot from the storage bin.
//   3. Stop the robot and wait for autonomous phase to end.
//
// This simple template does nothing except play a periodic tone every few seconds.
//
```

```
// At the end of the autonomous period, the FMS will autonmatically abort (stop) execution of
the program.
//
////////////////////////////////////////////////////////////////////////////////////////////
///////

task main()
{
    initialize ();  // In the original template this was "initializeRobot();"

    waitForStart (); // Wait for the beginning of autonomous phase.

    //
    // Raise the arm.  Make sure the arm is in its lowest position before
    // running this autonomous routine.
    //
    raise_arm ();

    //
    // Drive forward twelve inches.
    //
    drive_both_wheels (75, 75, 12 * kInch);

    //
    // Rotate right 90 degrees.  This will cause one motor to turn in one
    // direction, but the other will turn in the opposite direction.
    //
    TurnRight (90 * kDegrees);

    //
    // Drive forward twelve inches.
    //
    drive_both_wheels (75, 75, 12 * kInch);

    //
    // Rotate right 90 degrees.  This will cause one motor to turn in one
    // direction, but the other will turn in the opposite direction.
    //
    TurnRight (90 * kDegrees);

    //
    // Drive forward until a white line is found.
    //
    find_line ();

    //
    // Lower the arm and close the hand.
    //
    lower_arm ();
    move_hand (e_hand_command_close);

    //
    // Drive forward twelve inches.
    //
    drive_both_wheels (75, 75, 12 * kInch);

    //
    // Rotate left 90 degrees.
    //
    TurnLeft (90 * kDegrees);

    //
    // Drive forward twelve inches.
    //
```

```
    drive_both_wheels (75, 75, 12 * kInch);

} // main
```

Appendix H:motor.h

```
//-------------------------------------------------------------------------
//
// motor.h
//
// This file provides functions that relate to driving the DC motors.
//

//-------------------------------------------------------------------------
//
// convert_joystick_to_motor
//
int convert_joystick_to_motor (short a_joystick_value)

{
    //
    // Set the default speed and direction to 0 and one, respectively.
    //
    float speed = 0;
    short direction = 1;

    //
    // If the specified joystick value is [-27,27], then set the speed to the
    // absolute value of the joystick value and subtract 28.  This margin allows
    // for a dead zone (noise) within the joystick hardware.
    //
    if ((a_joystick_value < -28) || (a_joystick_value > 28))
    {
        speed = abs (a_joystick_value) - 28;
    }
    //
    // If the specified joystick value is greater than zero, then set the
    // direction to a negative value (-1) (i.e. pushing a joystick away from the
    // human has a negative value, so negate it to drive the robot "forward").
    // This will depend on how your robot is wired and/or how the pragmas are
    // configured.
    //
    if (a_joystick_value > 0)
    {
        direction = -1;
    }

    //
    // Return the speed multiplied by the direction.
    //
    return (speed * direction);

} // convert_joystick_to_motor
```

Appendix I:manual.c

```
#pragma config(Hubs,  S1, HTMotor,  HTServo,  none,     none)
```

```
#pragma config(Sensor, S2,     eye_sensor,          sensorLightActive)
#pragma config(Motor,  motorA,          left_arm_motor, tmotorNXT, PIDControl, encoder)
#pragma config(Motor,  motorB,          right_arm_motor, tmotorNXT, PIDControl, encoder)
#pragma config(Motor,  motorC,           ,             tmotorNXT, openLoop)
#pragma config(Motor,  mtr_S1_C1_1,     left_foot_motor, tmotorTetrix, openLoop, reversed)
#pragma config(Motor,  mtr_S1_C1_2,     right_foot_motor, tmotorTetrix, openLoop)
#pragma config(Servo,  srvo_S1_C2_1,    left_hand_servo,           tServoStandard)
#pragma config(Servo,  srvo_S1_C2_2,    right_hand_servo,          tServoStandard)
#pragma config(Servo,  srvo_S1_C2_3,    servo3,                tServoNone)
#pragma config(Servo,  srvo_S1_C2_4,    servo4,                tServoNone)
#pragma config(Servo,  srvo_S1_C2_5,    servo5,                tServoNone)
#pragma config(Servo,  srvo_S1_C2_6,    servo6,                tServoNone)
//*!!Code automatically generated by 'ROBOTC' configuration wizard               !!*//

////////////////////////////////////////////////////////////////////////////////////////
///////
//
//                           Tele-Operation Mode Code Template
//
// This file contains a template for simplified creation of an tele-op program for an FTC
// competition.
//
// You need to customize two functions with code unique to your specific robot.
//
////////////////////////////////////////////////////////////////////////////////////////
///////

#include "JoystickDriver.c"  //Include file to "handle" the Bluetooth messages.

#include "constants.h"  // Must be included before other user-defined headers.

#include "BasicFunctions.h"
#include "motor.h"

#include "initialize.h" // Must be included after other user-defined headers.

////////////////////////////////////////////////////////////////////////////////////////
///////
//
//                                    initializeRobot
//
// Prior to the start of tele-op mode, you may want to perform some initialization on your
robot
// and the variables within your program.
//
// In most cases, you may not have to add any code to this function and it will remain
"empty".
//
////////////////////////////////////////////////////////////////////////////////////////
///////n

#ifdef d_code_moved
This template function has been moved to the constants.h header file, so it can
be used for both manual and autonomous purposes.
void initializeRobot()
{

} // initializeRobot
#endif d_code_moved

////////////////////////////////////////////////////////////////////////////////////////
///////
//
//                                        Main Task
```

```
//
// The following is the main code for the tele-op robot operation. Customize as appropriate
for
// your specific robot.
//
// Game controller / joystick information is sent periodically (about every 50 milliseconds)
from
// the FMS (Field Management System) to the robot. Most tele-op programs will follow the
following
// logic:
//   1. Loop forever repeating the following actions:
//   2. Get the latest game controller / joystick settings that have been received from the
PC.
//   3. Perform appropriate actions based on the joystick + buttons settings. This is usually
a
//      simple action:
//      *  Joystick values are usually directly translated into power levels for a motor or
//         position of a servo.
//      *  Buttons are usually used to start/stop a motor or cause a servo to move to a
specific
//         position.
//   4. Repeat the loop.
//
// Your program needs to continuously loop because you need to continuously respond to changes
in
// the game controller settings.
//
// At the end of the tele-op period, the FMS will autonmatically abort (stop) execution of the
program.
//
////////////////////////////////////////////////////////////////////////////////////////////
///////

task main()
{
    initialize ();  // In the original template this was "initializeRobot();"

    waitForStart ();   // wait for start of tele-op phase

    while (true)
    {
        //
        // Read the current values of the joystick controllers.
        //
        hogCPU (); // Prevent the messaging task from writing to it while it is being read.
        getJoystickSettings (joystick); // Added to template for kick-off presentation
        releaseCPU ();

        //
        // Manage power to the drive wheels.
        //
        motor[c_left_foot_motor] =
            convert_joystick_to_motor (joystick.joy1_y1);

        motor[c_right_foot_motor] =
            convert_joystick_to_motor (joystick.joy1_y2);

        //
        // Manage the power and direction of the LEGO arm motors, which make the
        // arm move up and down.
        //
        int l_motor = convert_joystick_to_motor (joystick.joy2_y1);
        motor[c_left_arm_motor] = l_motor;
```

```
        motor[c_right_arm_motor] = l_motor;

        //
        // Manage the hand servo positions, which make the hand open and close.
        //
        if (joy2Btn (6) != false)
        {
            move_hand (e_hand_command_open);
        }
        else if (joy2Btn (8) != false)
        {
            move_hand (e_hand_command_close);
        }
        else if (joy2Btn (5) != false)
        {
            move_hand (e_hand_command_partially_open);
        }
        else if (joy2Btn (7) != false)
        {
            move_hand (e_hand_command_partially_close);
        }

        //
        // This wait is important for several reasons.  It provides the message
        // a chance to perform its processing.  It also determines how fast the
        // servos open and close (see the increment blocks in move_hand).
        //
        wait1Msec (10);

    } // while (true)

} // main
```

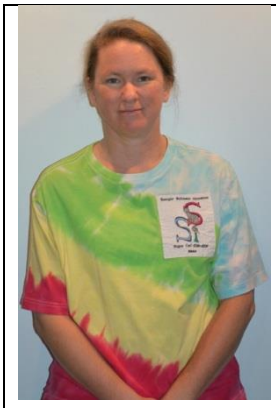
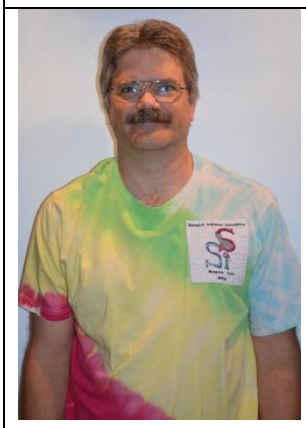
Appendix J: Project Team Profiles

Mary Spangler

Year 11 student at Leonardtown High School.

FTC Competitor for 5 years on Team 2843, Under the Son

CADD drawings and Bill of Materials

	**Lydean Spangler** Maryland FTC Planning Committee Co-Chair ssi@the-spanglers.net
	**David Spangler** Maryland FTC Planning Committee Co-Chair ssi@the-spanglers.net