Fanuc Robot Manual

Documentation Package April 2021

University of Idaho VMAL

Table of Contents

Table of Contents

How to Download and Install the Fanuc ROS Packages/Files (Photo Guide Below)

1. Obtain the github link from the UI CDA lab

2. Download the code base and put place the three folders (fanuc, fanuc_demo, and rosLaunch) into the src folder of your ROS catkin workspace

3. Navigate into your fanuc_demo/src folder and run this command (to make all python scripts executable)

```
>> chmod +x *.py
```

4. You need two additional ROS packages to get the main programs to run. These are the industrial core and moveit packages:

```
>> sudo apt-get install ros-melodic-industrial-core

>>sudo apt-get install ros-melodic-moveit
```

How to Download the Fanuc ROS Packages/Files

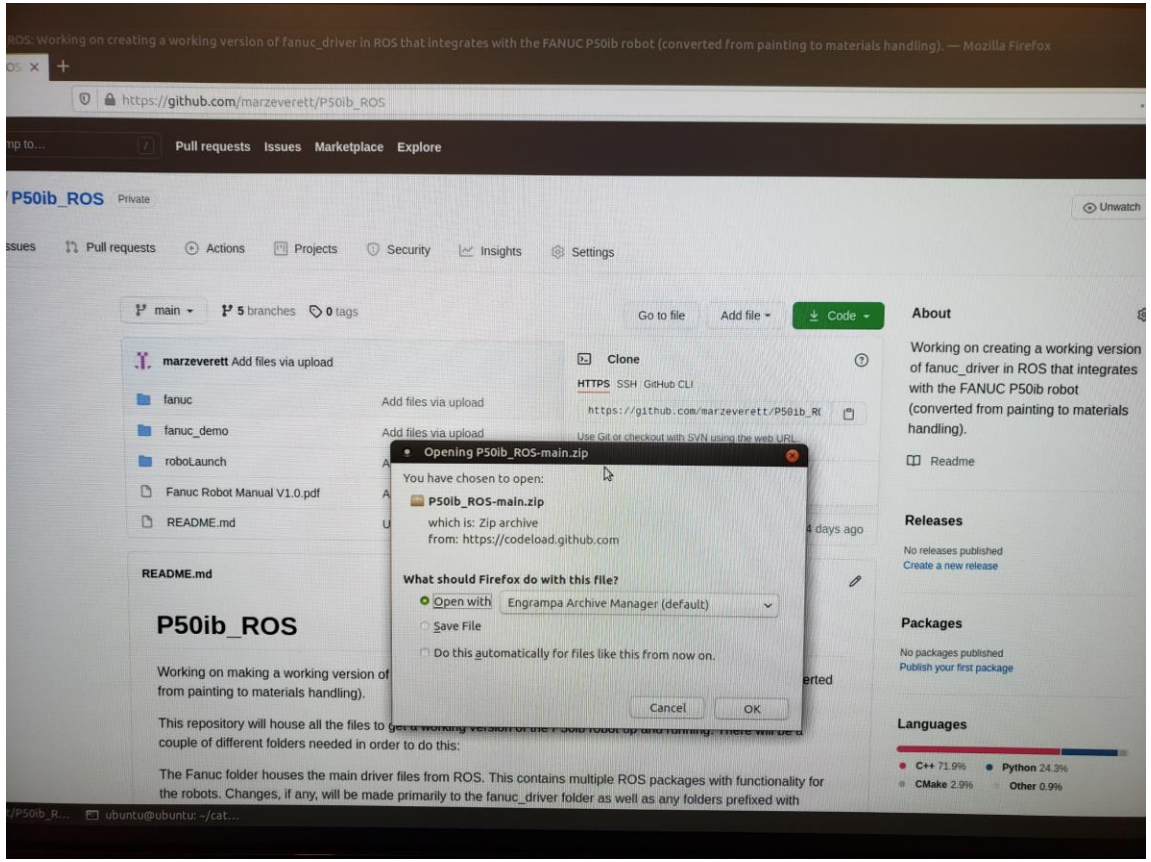
(You may have to do a sudo apt-get update to install everything correctly).

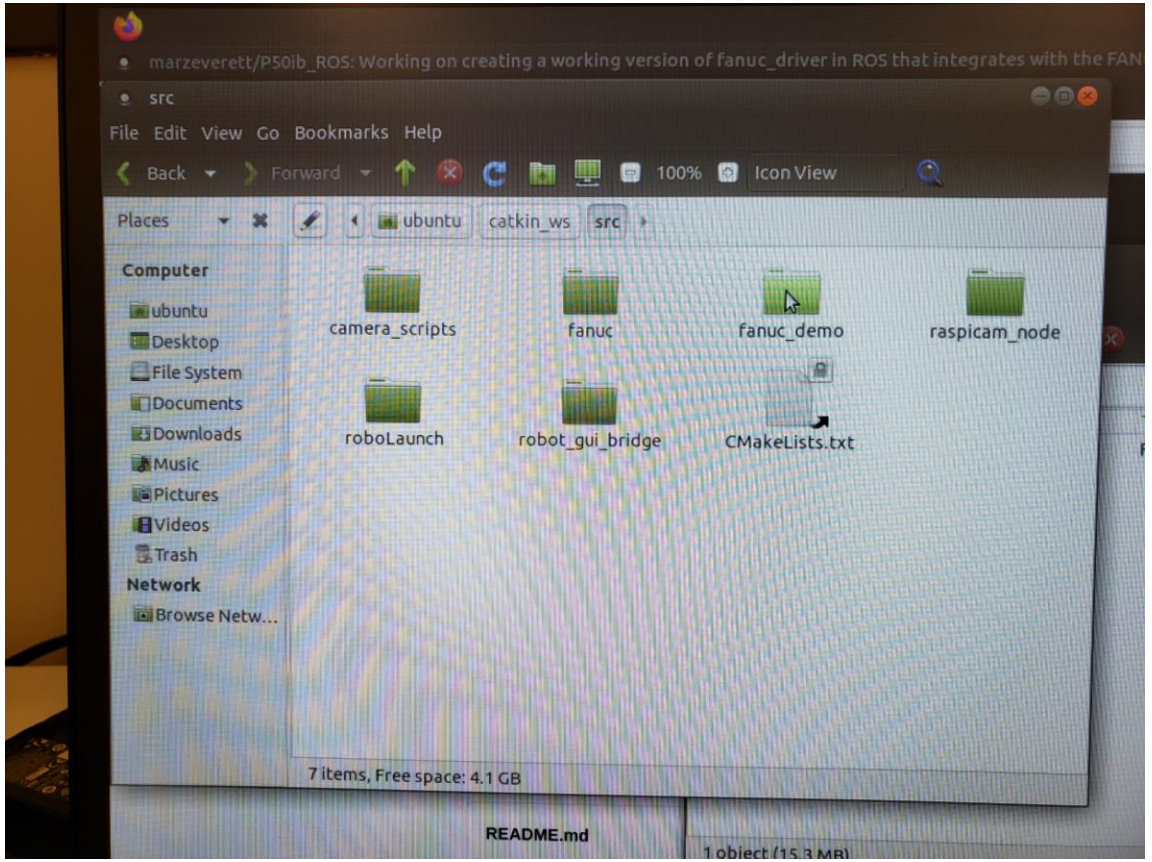
7. Navigate to the base of your catkin workspace, and do a catkin_make

```
>> catkin_make
```

Everything should now build correctly.

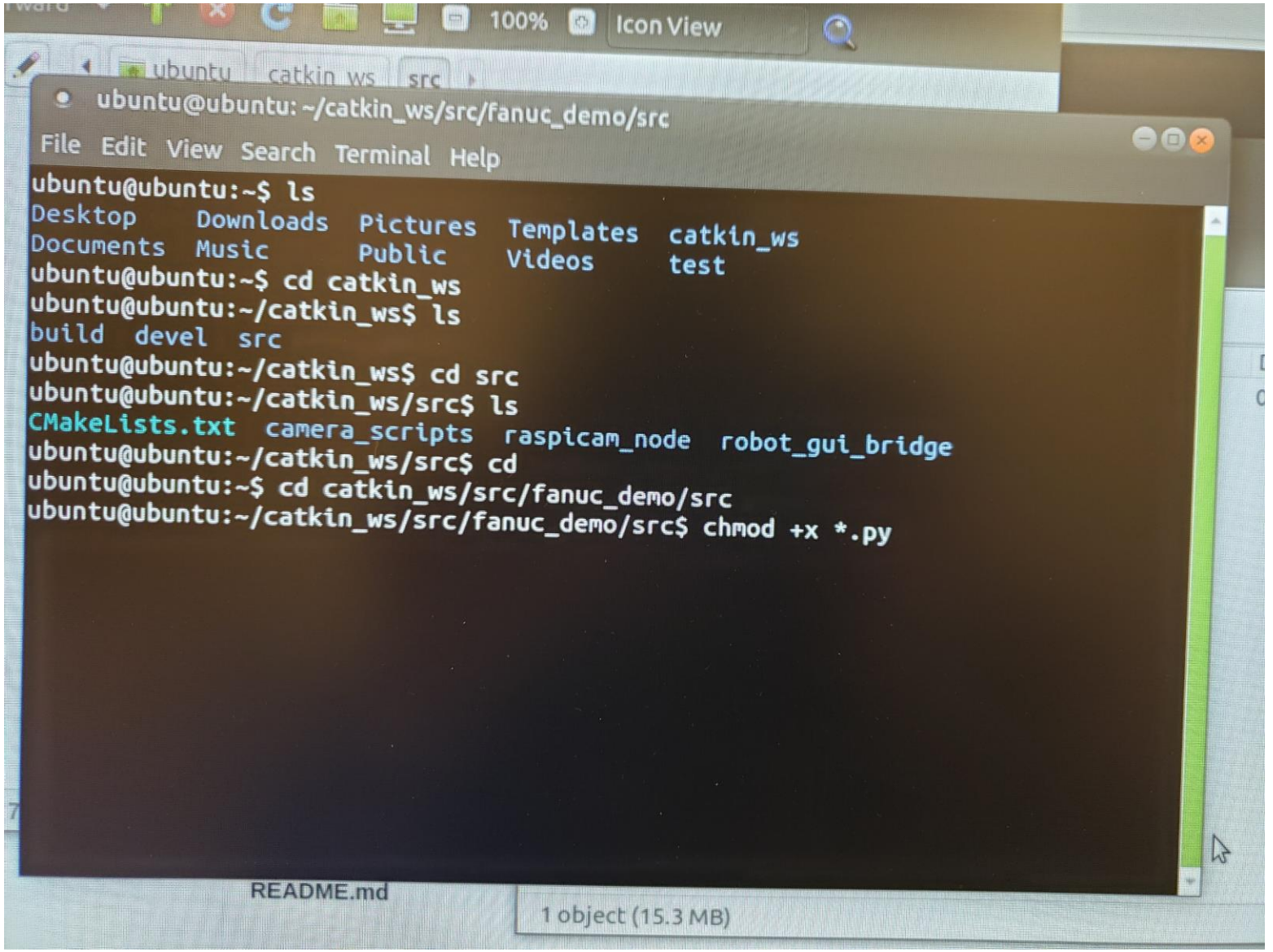
Download the code base and put place the three folders (fanuc, fanuc_demo, and rosLaunch) into the src folder of your ROS catkin workspace





Navigate into your fanuc_demo/src folder and run this command (to make all python scripts executable)

>> chmod +x *.py

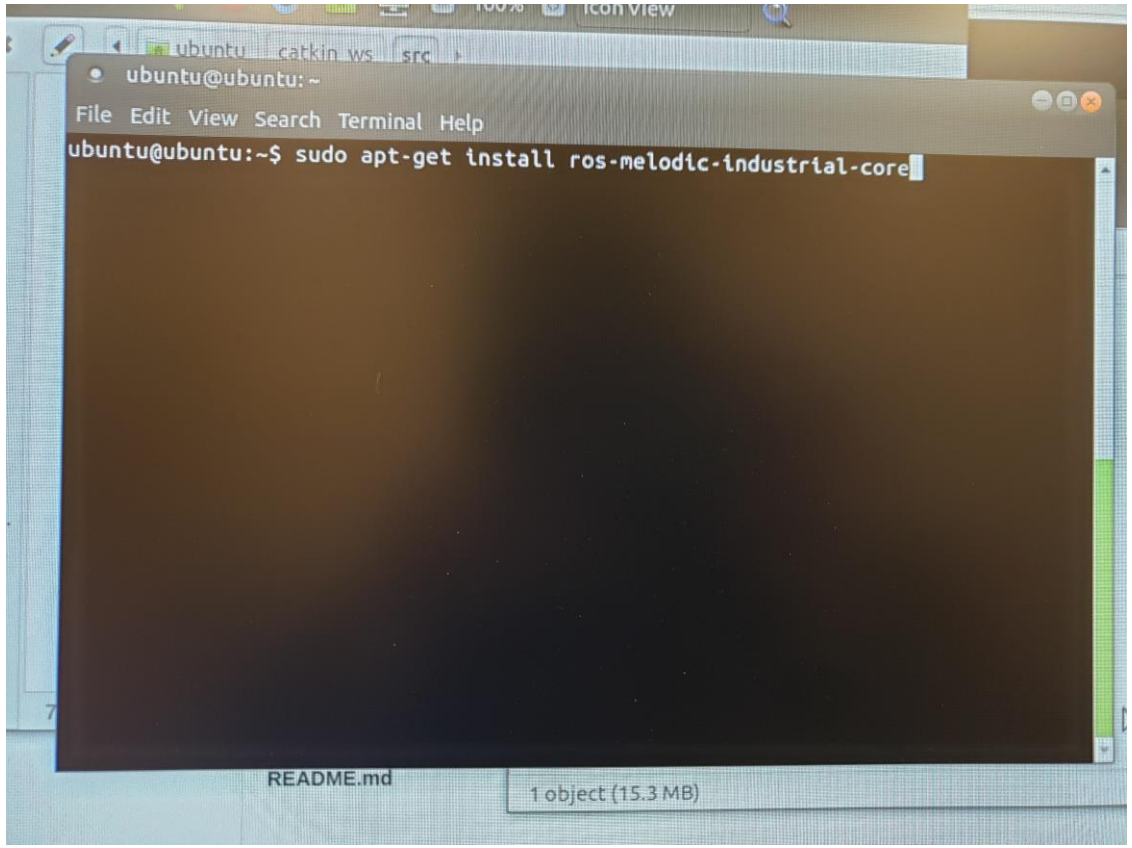


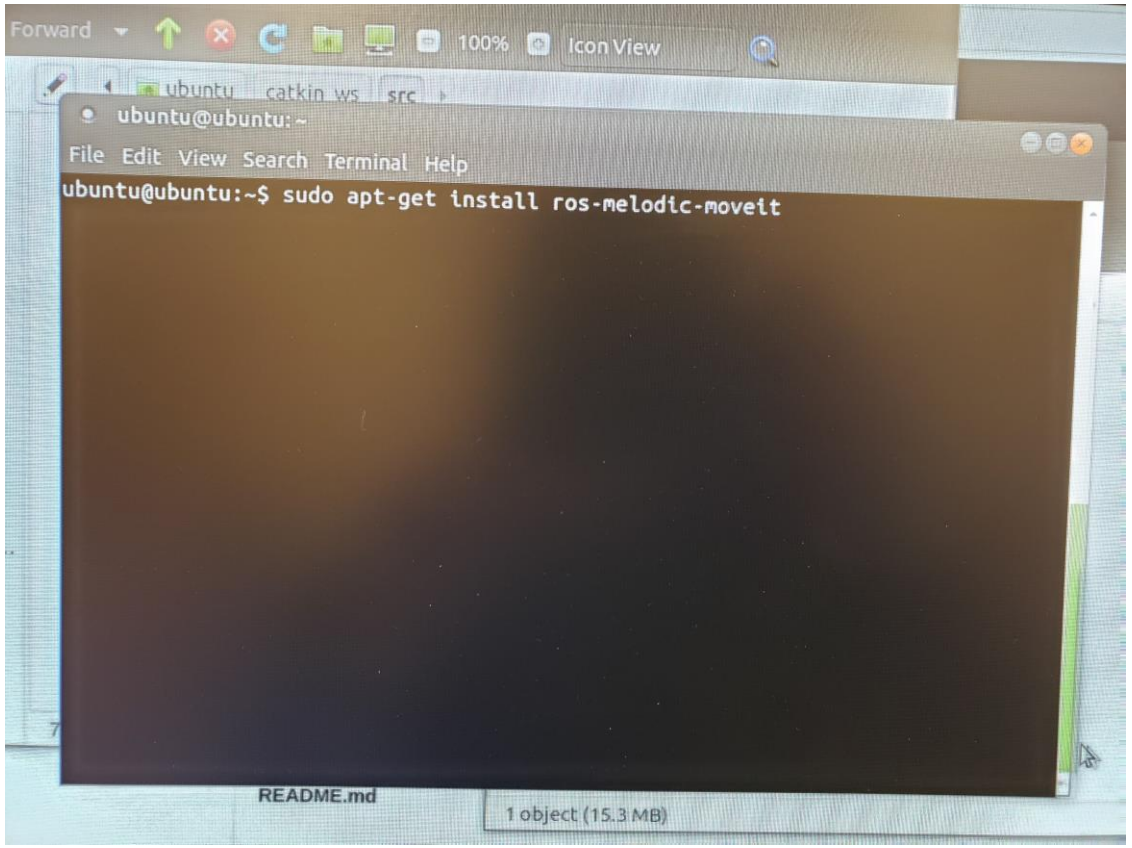
You need two additional ROS packages to get the main programs to run. These are the industrial core and moveit packages:

>> sudo apt-get install ros-melodic-industrial-core

>>sudo apt-get install ros-melodic-moveit

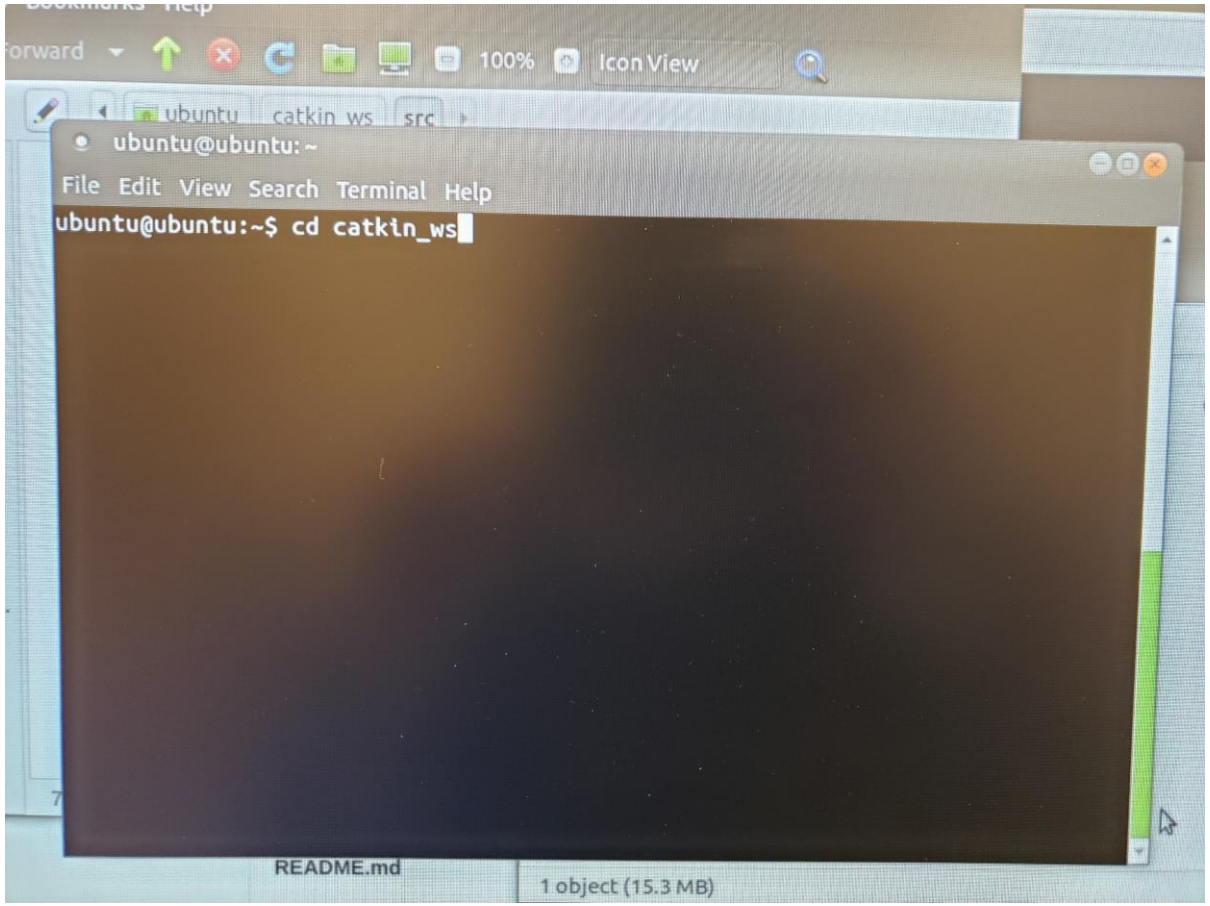
(You may have to do a sudo apt-get update to install everything correctly).

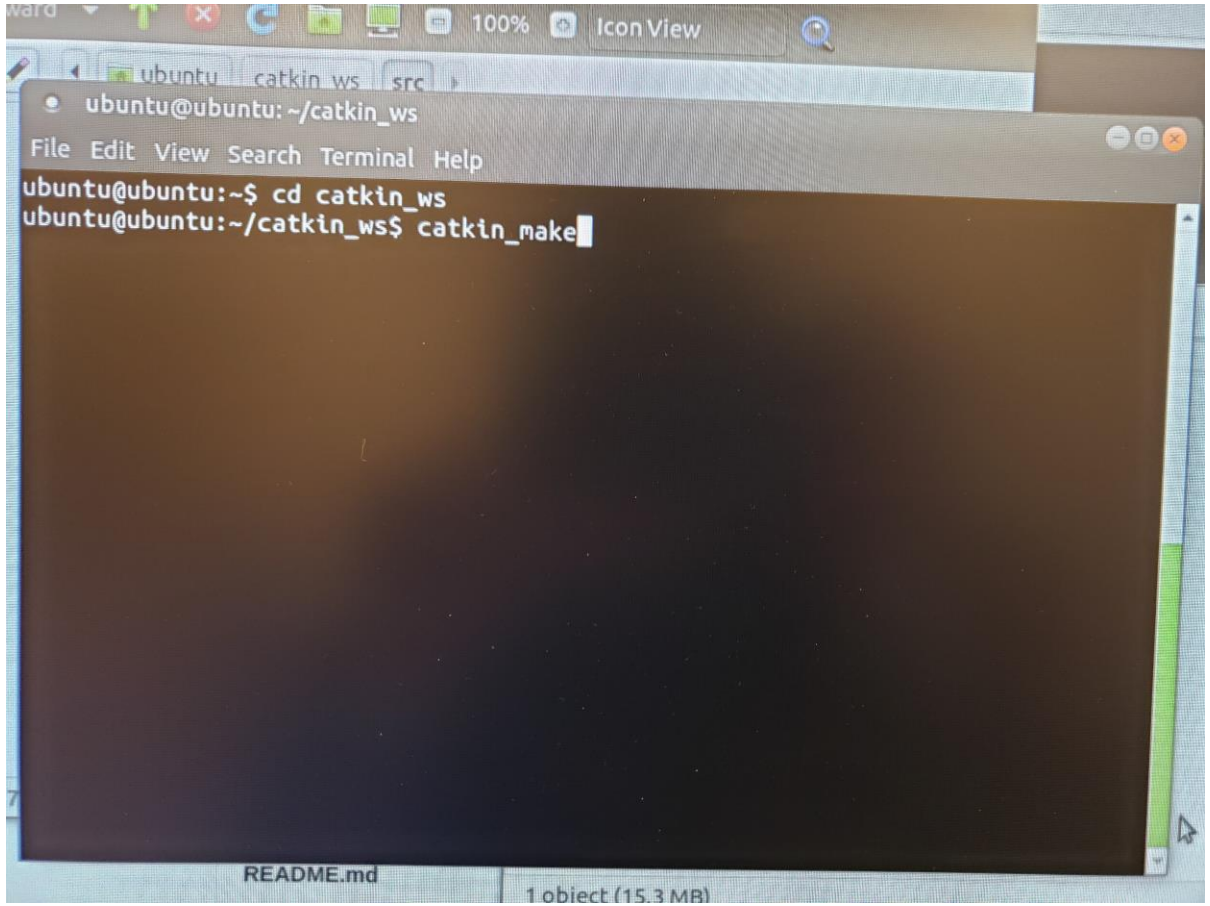




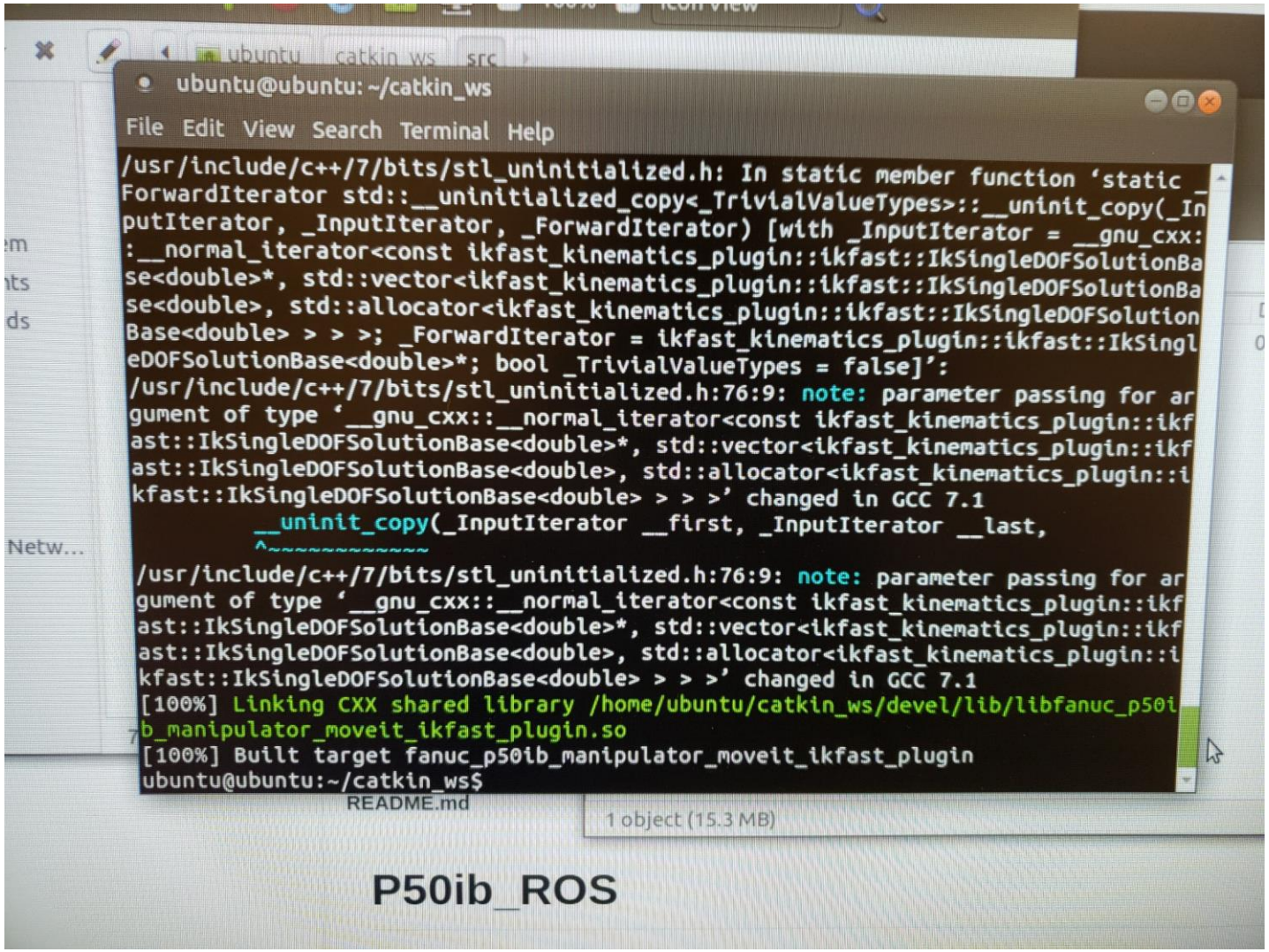
Navigate to the base of your catkin workspace, and do a catkin_make

```
>> catkin_make
```

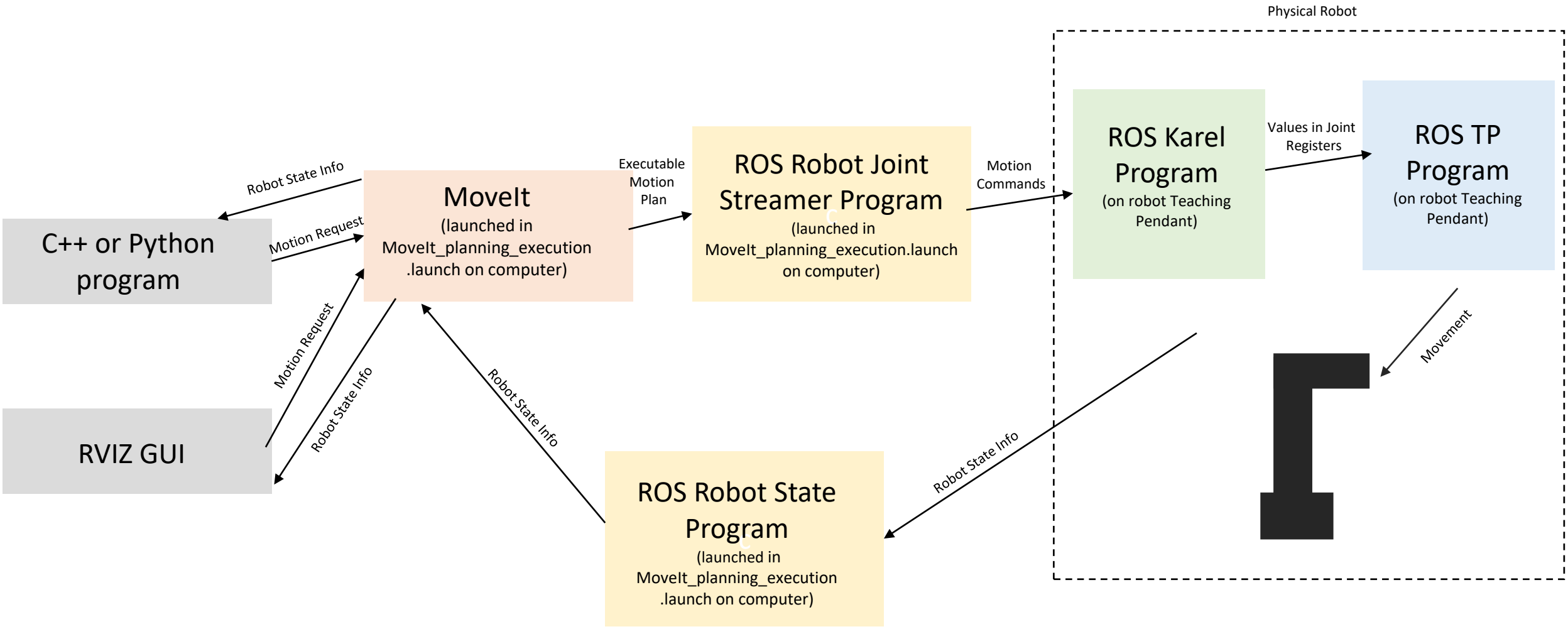




Everything should now build correctly.



Robot Driver Architecture



FANUC Hardware Control

- The following apply to using the hardware physical robot with the teaching pendant/controller

Steps for Running the FANUC Robot

Start-Up

1. Start up the Robot
2. When you are out of the cage, put the robot in Auto mode
3. Start the ROS program on the FANUC teaching pendant (the physical robot)
4. Start the robot state program on a computer (best practice is to use ROSBOX 2)

Takedown:

1. Run your program on your machine.
2. Close the robot state program
3. Abort the ROS program on the FANUC teaching pendant
4. Turn the robot to T1 mode
5. Turn off the robot

How to Start Up the Robot (Photo Guide Below)

- To start up the robot, follow these steps:

1. Make sure the robot is OFF or in T1 mode. There is a power button on the controller that will be lit up if the robot is on. You can also check the dial to see what mode the robot is in.
2. Unlock the robot cage and enter (ONLY IF THE ROBOT IS OFF OR IN T1!!). Grab the teaching pendant and screwdriver, and then exit the cage.
3. Turn the robot on by pressing the power button, and use the screwdriver to turn it to auto mode. Once the robot is in auto mode, DO NOT ENTER THE CAGE.

Make sure the robot is OFF or in T1 mode. There is a power button on the controller that will be lit up if the robot is on. You can also check the dial to see what mode the robot is in.

Dial

Power Button. Not lit up if power is off.

Unlock the robot cage and enter (ONLY IF THE ROBOT IS OFF OR IN T1!!). Grab the teaching pendant and screwdriver, and then exit the cage.

Turn the robot on by pressing the power button, and use the screwdriver to turn it to auto mode. Once the robot is in auto mode, DO NOT ENTER THE CAGE.

Power Button. Not lit up if power is off. If lit up, power is on

Screwdriver turns dial. Turn it to “auto” mode.

How to Bring Up the ROS Program on the Robot (Photo Guide Below)

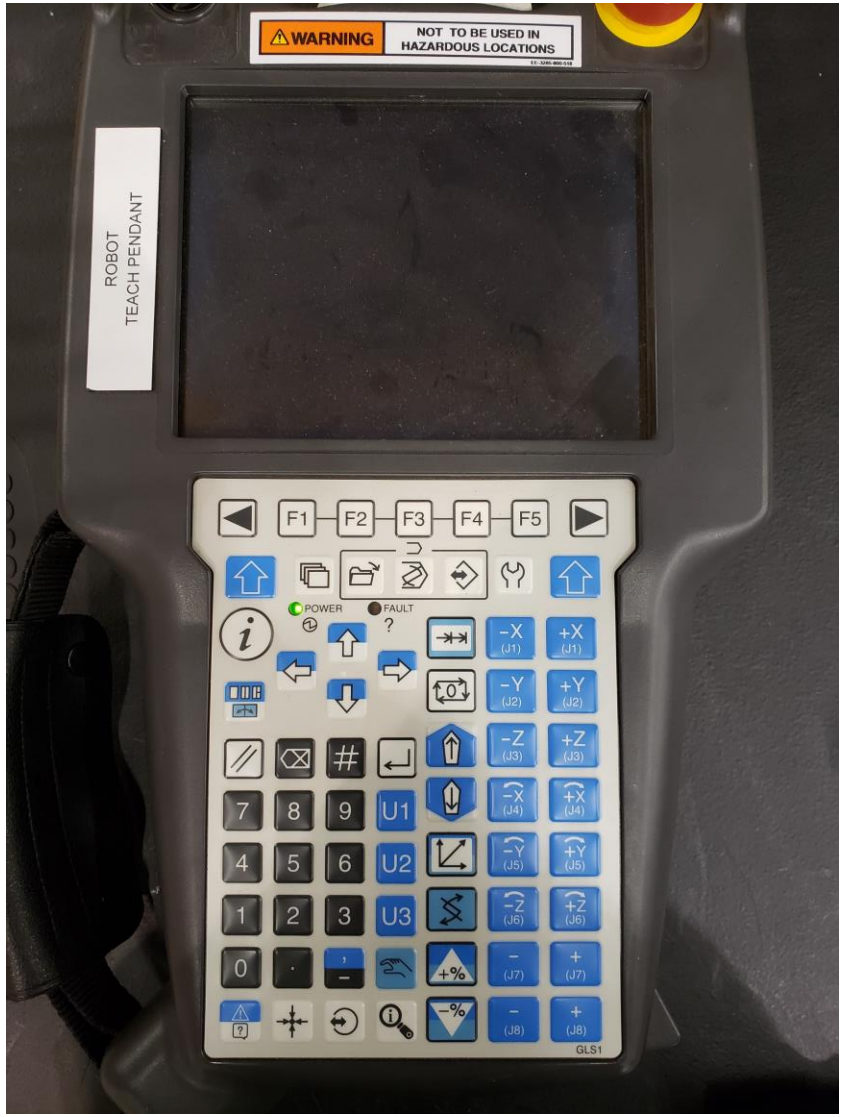
- To bring up the ROS Program on the robot, you will need to use the teaching pendant. (Note: Even if the switch is turned to “off”, the teaching pendant is on if the screen is on. If the switch is “off”, you just can’t operate in T1 or T2 mode. If it is “on”, you can’t operate it in auto mode.)
- First, if the robot is in auto mode, make sure the teaching pendant is switched off. You will still use the teaching pendant, but this lets the teaching pendant know you are in the right mode.
- Clear any faults. (Go to section teaching pendant tips—clear faults)
- Adjust the speed to what you want (Teaching pendant tips-change speed)
- Press the select button

How to Bring Up the ROS Program on the Robot

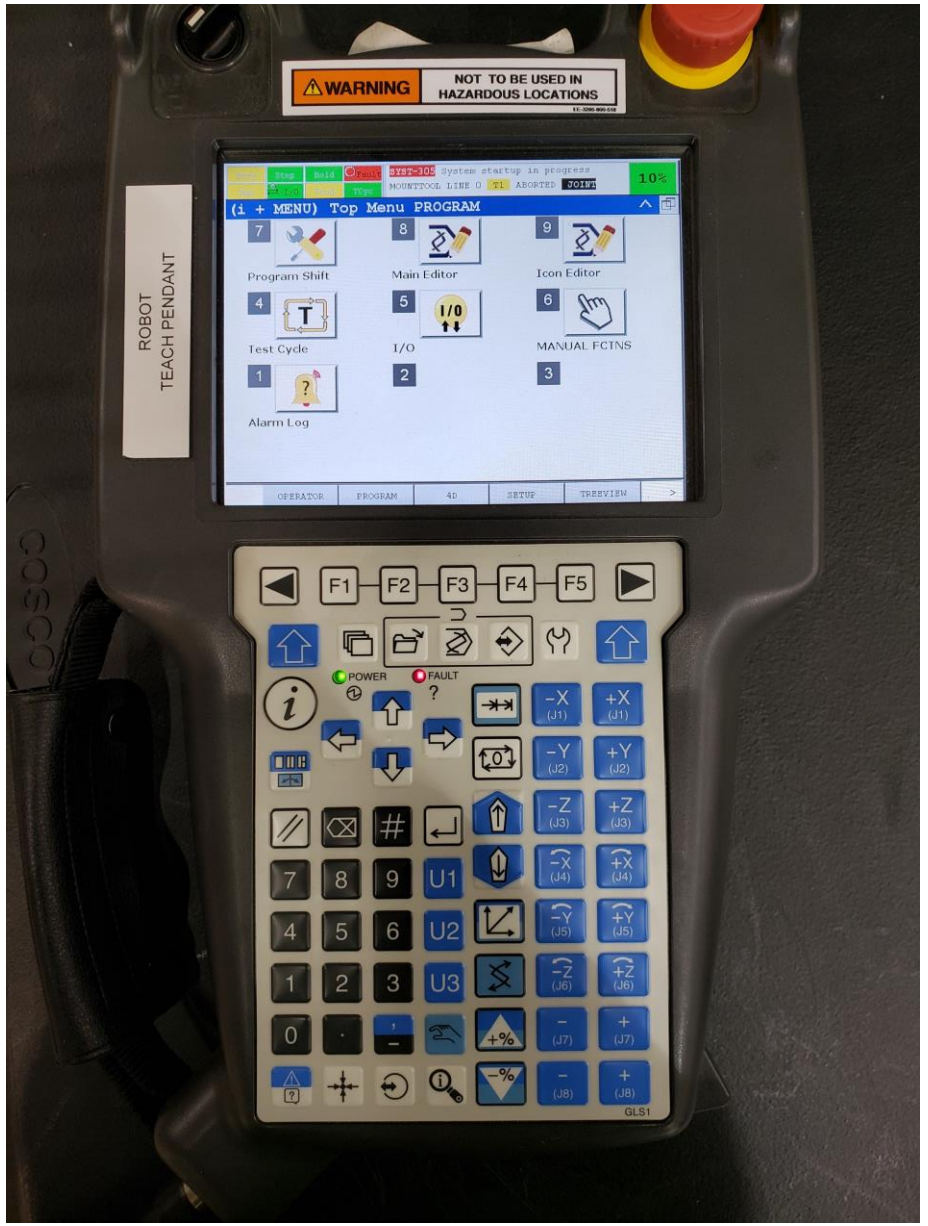
- Use the up and down arrow keys to find the ROS program (you can use the shift key to move up and down by bigger chunks)
- Press enter
- (Now, leave the teaching pendant alone)
- Go to the control panel
- Press the green “Cycle Start” button
- If everything worked, the button will light up green and you will get a message on the Teaching Pendant screen

To bring up the ROS Program on the robot, you will need to use the teaching pendant. (Note: Even if the switch is turned to “off”, the teaching pendant is on if the screen is on. If the switch is “off”, you just can’t operate in T1 or T2 mode. If it is “on”, you can’t operate it in auto mode.)

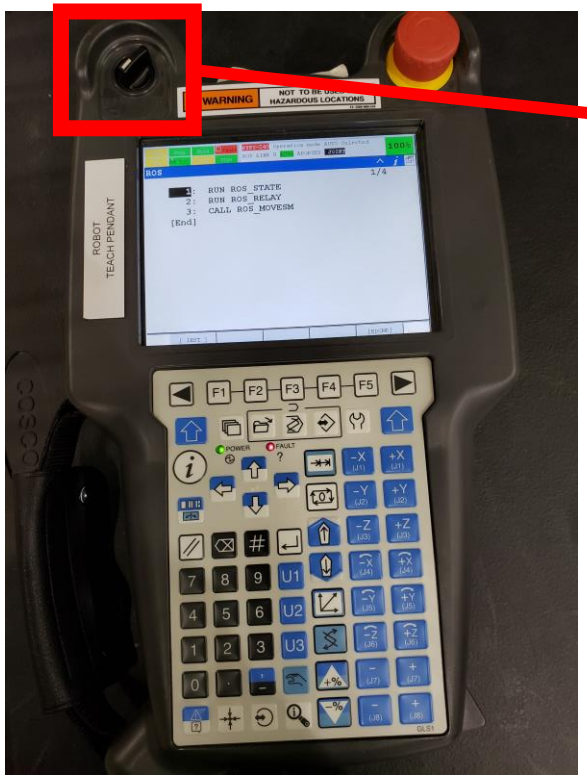
Teaching Pendant (Robot is OFF)



Teaching Pendant (Robot is ON)

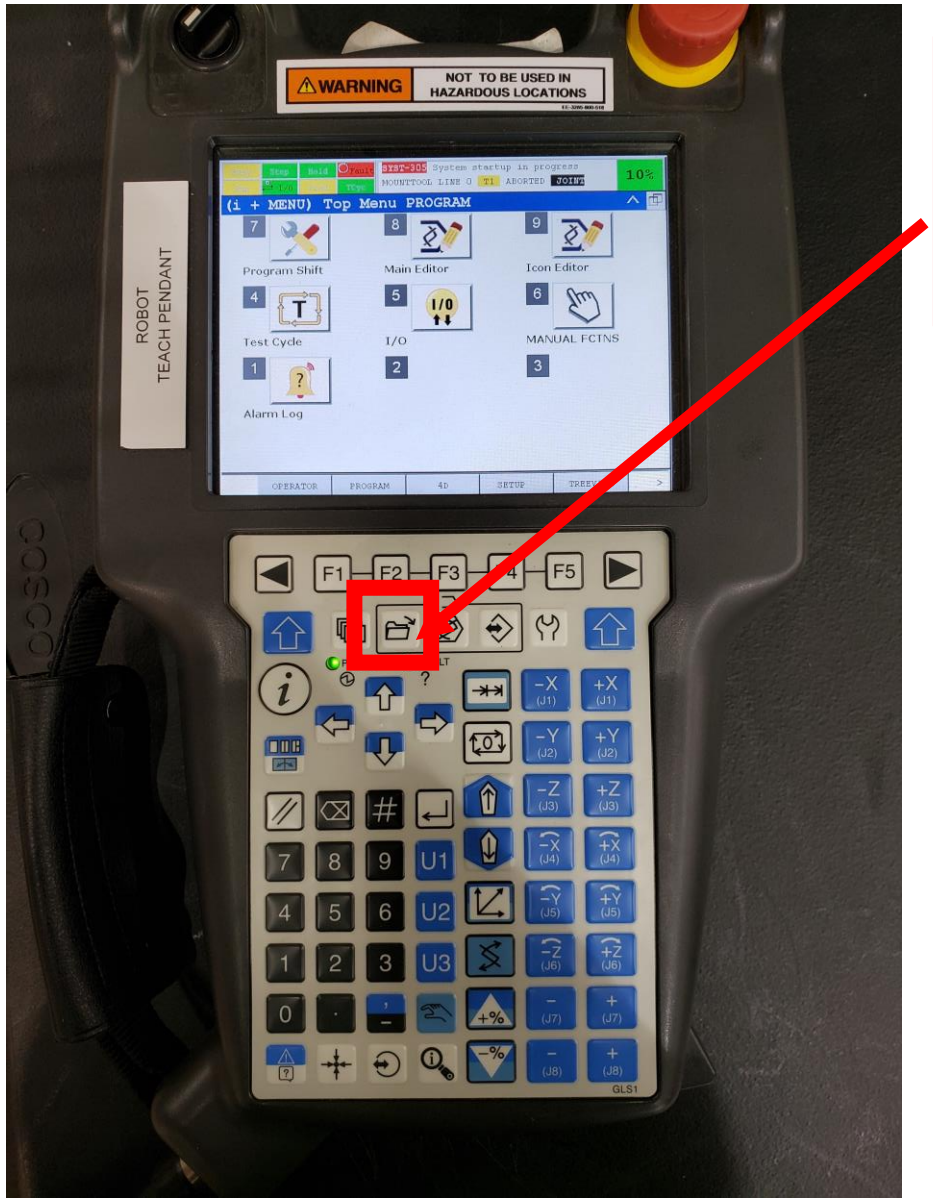


First, if the robot is in auto mode, make sure the teaching pendant is switched off. You will still use the teaching pendant, but this lets the teaching pendant know you are in the right mode.

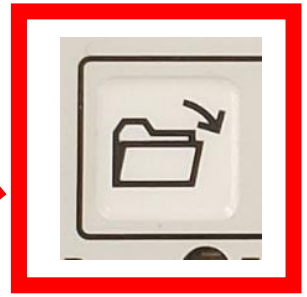


Teaching Pendant is switched OFF.

Clear any faults. (Go to section teaching pendant tips—clear faults)
Adjust the speed to what you want (Teaching pendant tips-change speed)
Press the select button



Select Button

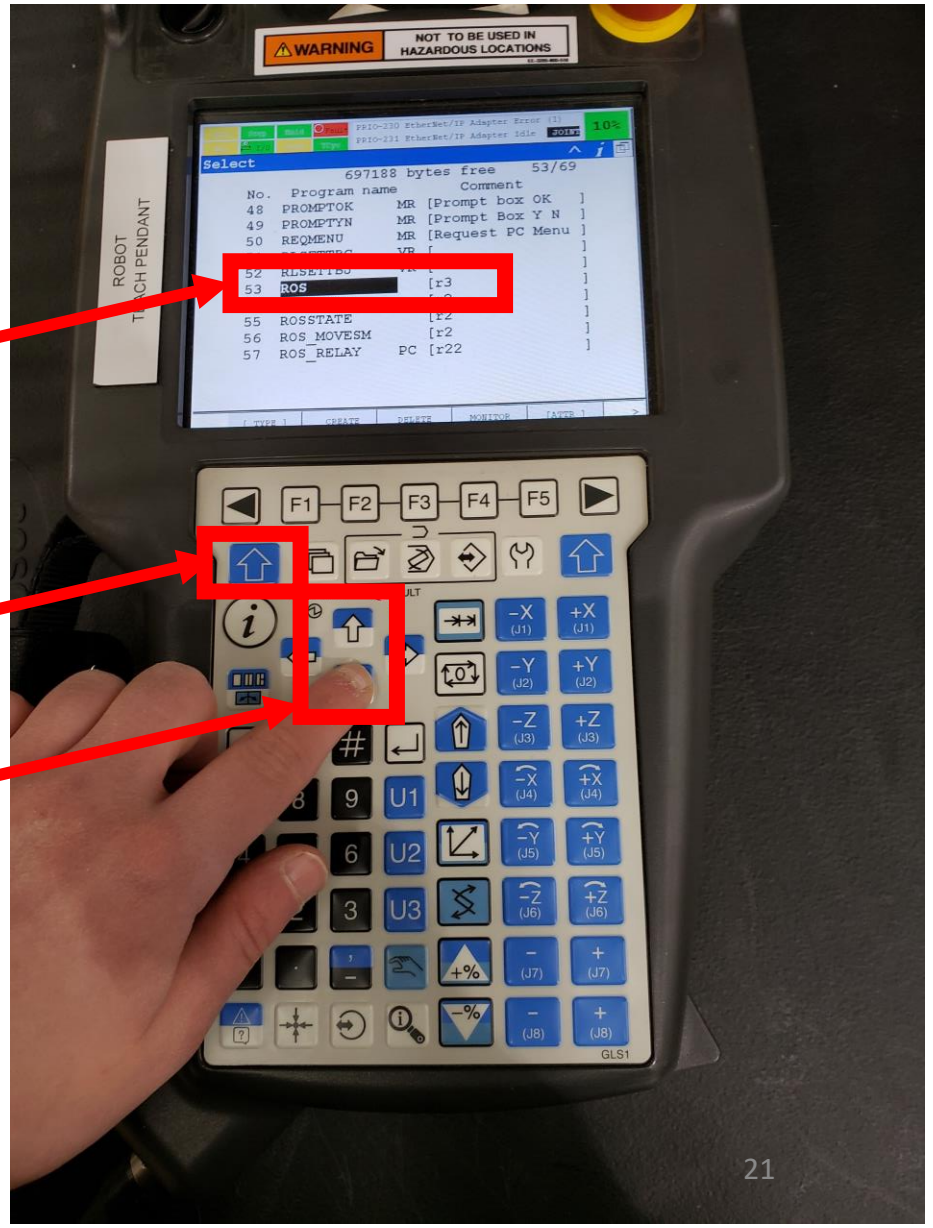


ROS Program to Run

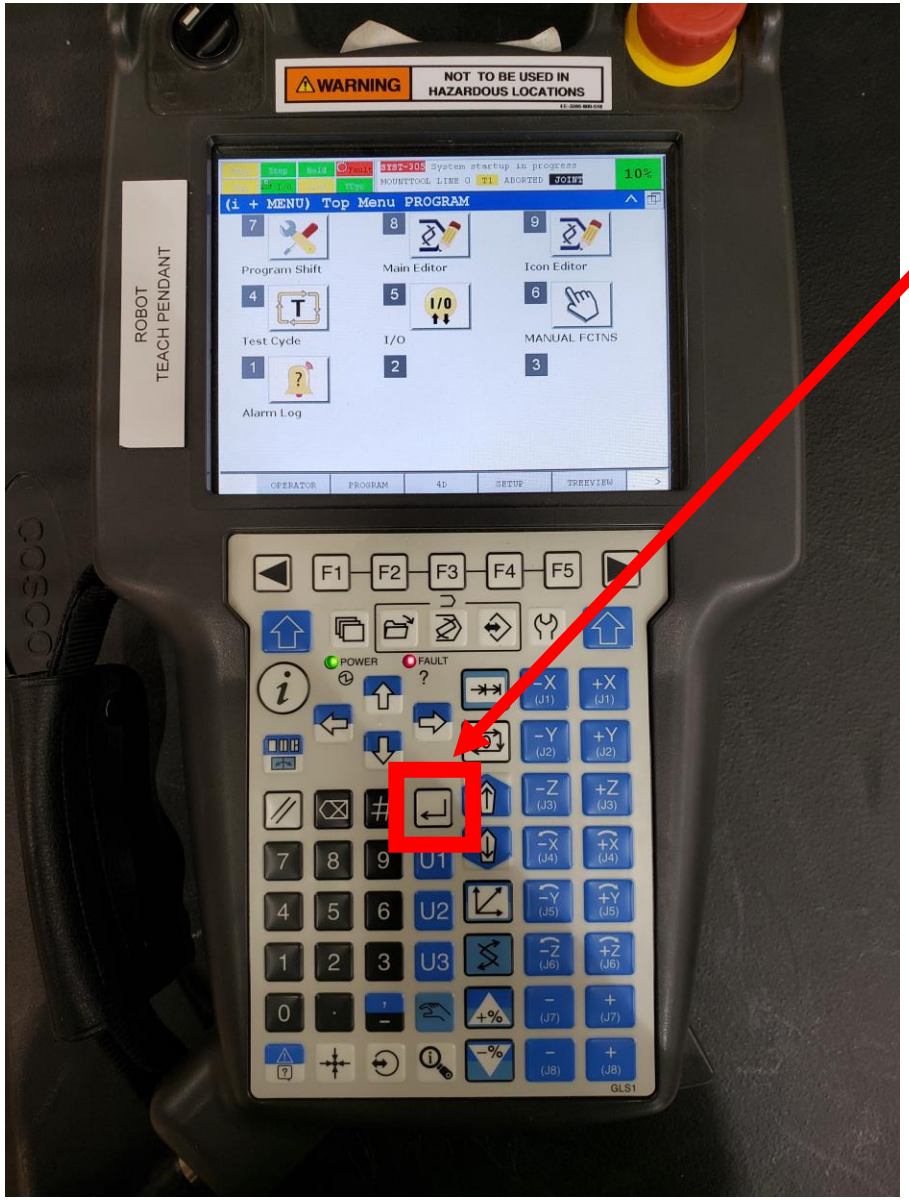
Press and Hold SHIFT key to move up and down by bigger chunks

Up/Down Arrow Keys

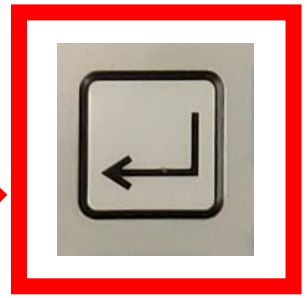
Use the up and down arrow keys to find the ROS program (you can use the shift key to move up and down by bigger chunks)

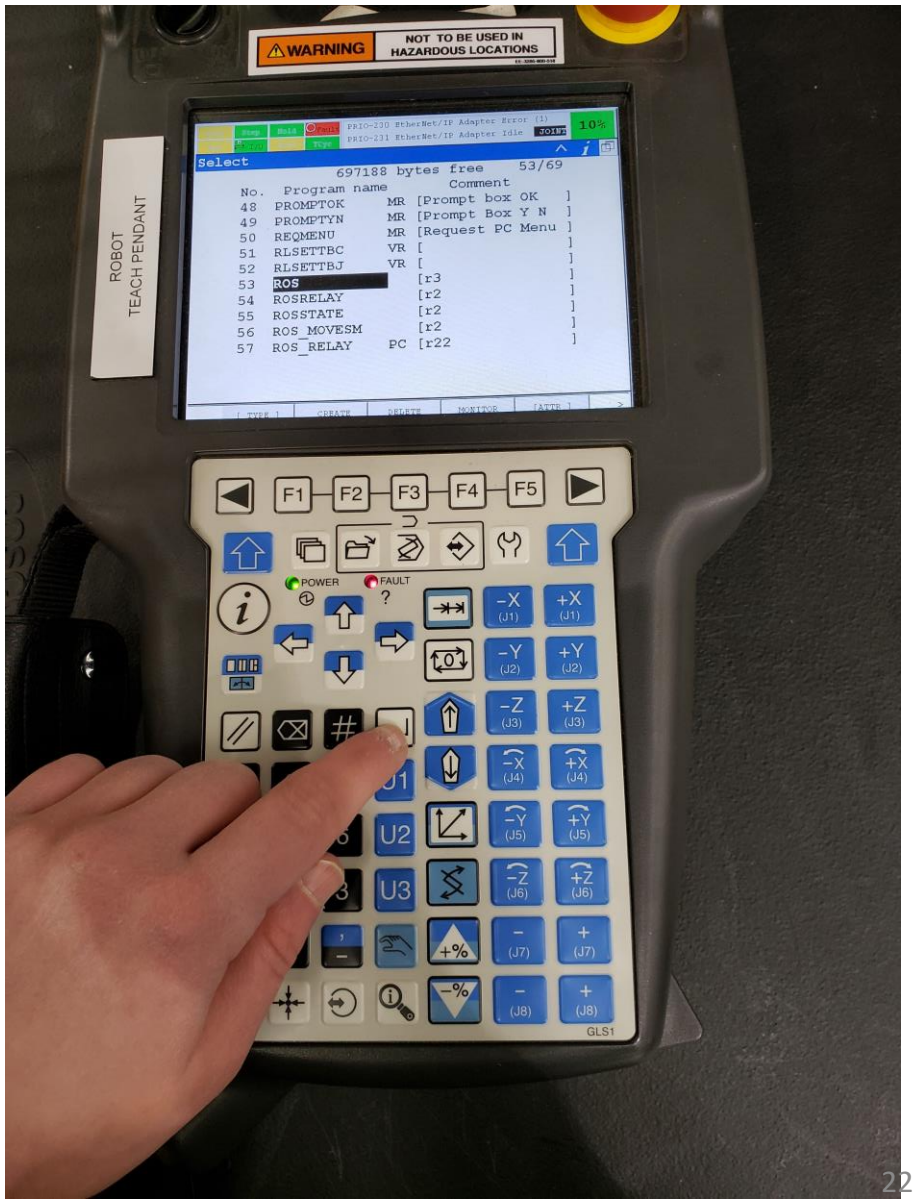


Press enter
(Now, leave the teaching pendant alone)



Enter Button



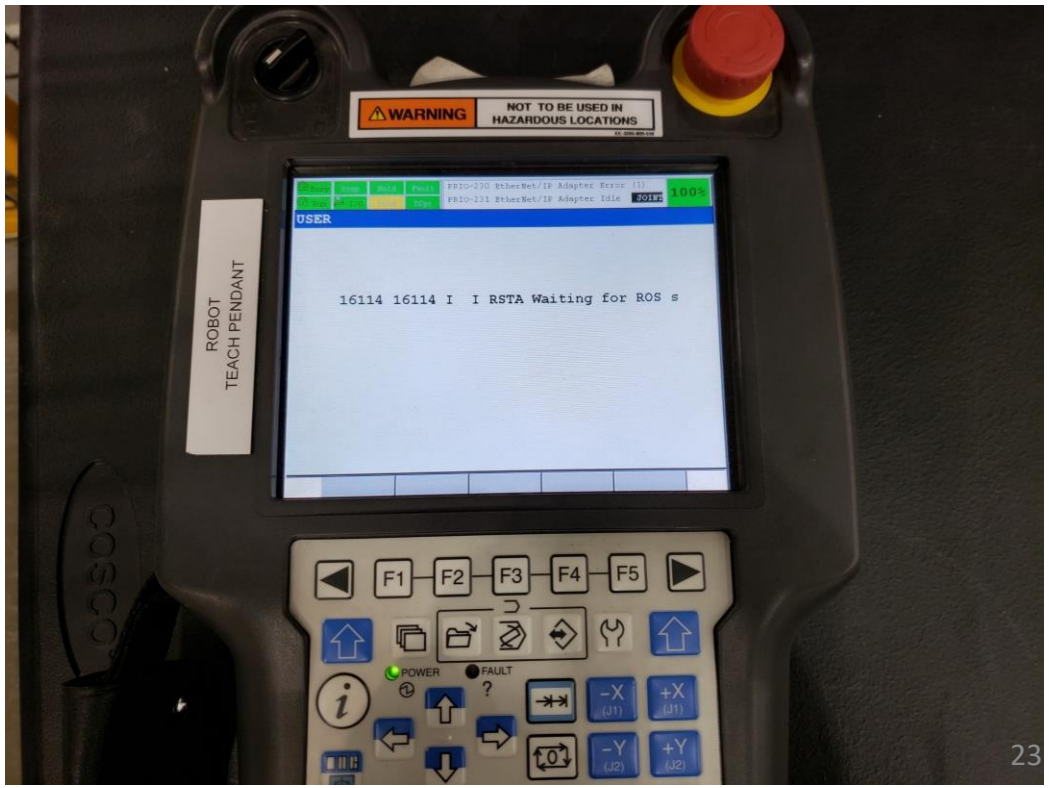


Go to the control panel
Press the green “Cycle Start” button
If everything worked, the button will light up green and you will get a message on the Teaching Pendant screen

Cycle Start Button—starts programs loaded on teaching pendant in “auto” mode

Cycle Start Button Lights up Green when program successfully started

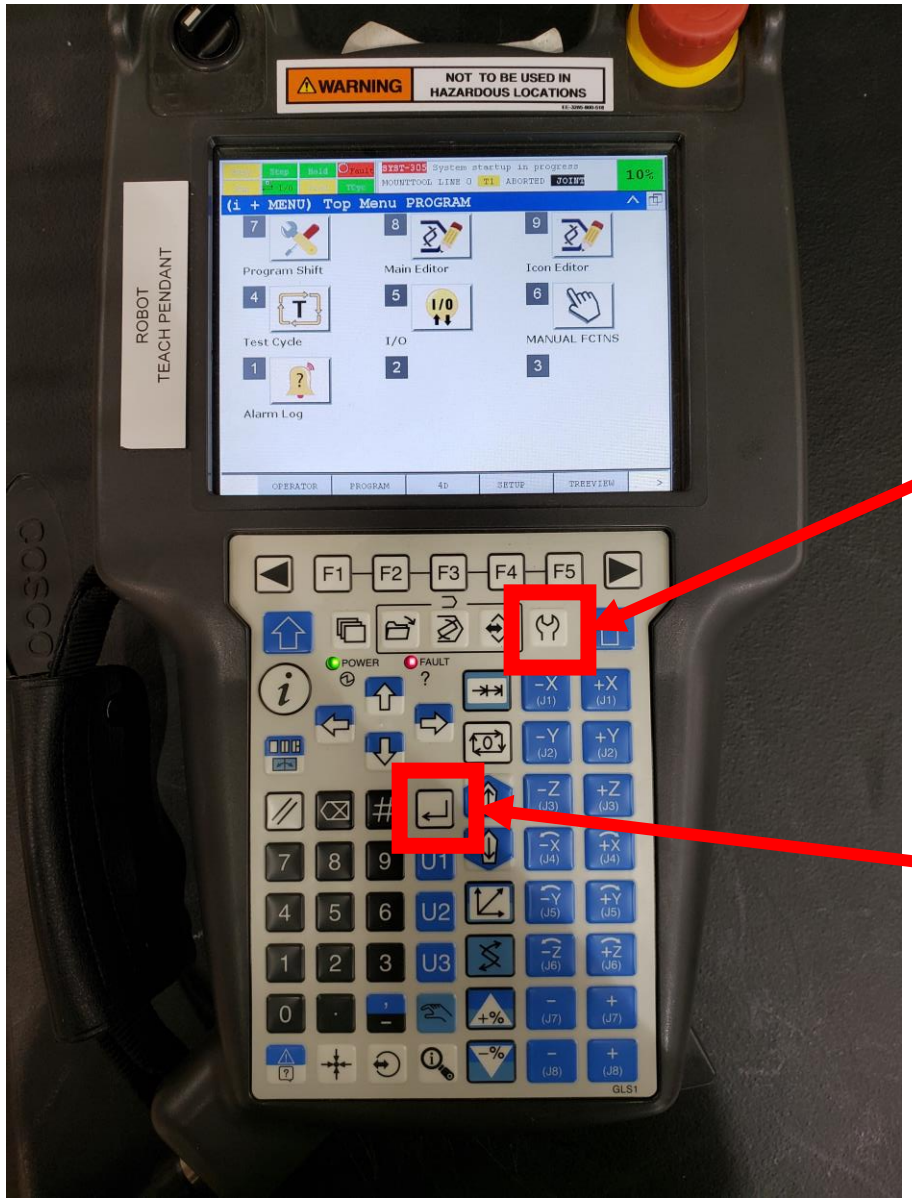
Teach Pendant now displays this message



How to Close the Robot State Program on the Robot (Photo Guide Below)

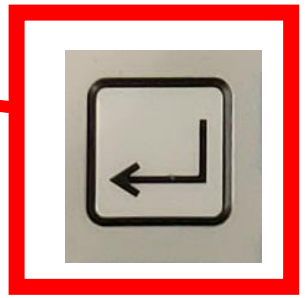
- To stop the program: Press the function key and then enter on “abort”. Do this twice.

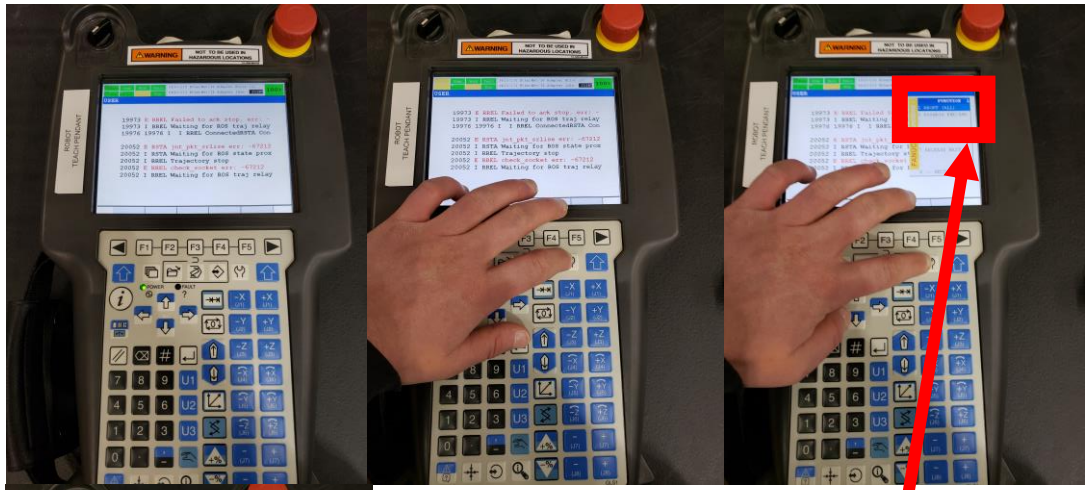
To stop the program: Press the function key and then enter on “abort”. Do this twice.



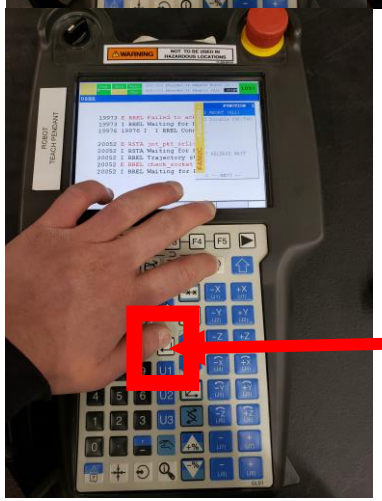
Function key

Enter key





Function key brings up program menu



Press enter on first option “Abort”

How to Turn the Robot Off (Photo Guide Below)

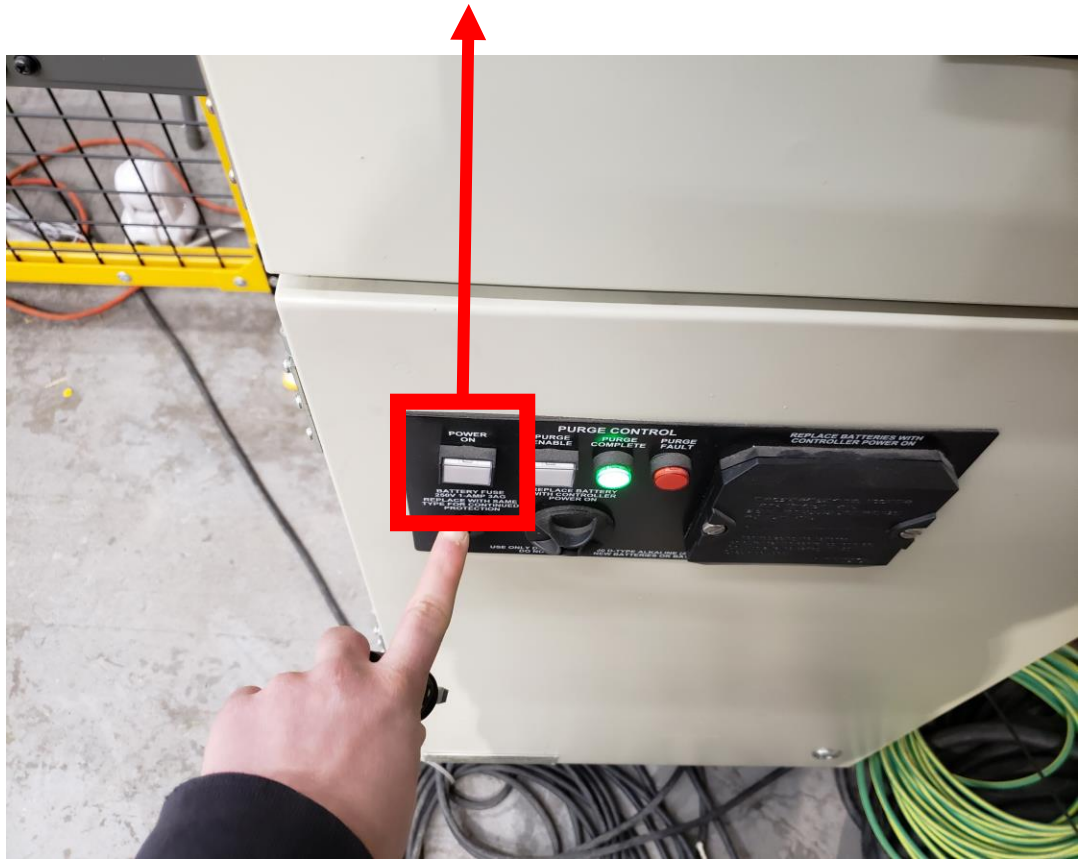
- To turn the robot off:
- First, stop any programs running on the teaching pendant (function-abort (twice for the ROS program, once for all others) (See Teaching Pendant Tips—abort a program)
- Next, use the screwdriver to put the robot into T1
- Finally, press the power button. The light will turn off, and the robot is now shutdown.

First, stop any programs running on the teaching pendant (function-abort (twice for the ROS program, once for all others) (See Teaching Pendant Tips—abort a program)
Next, use the screwdriver to put the robot into T1

Use screwdriver to change to T1 (safe testing mode)

Finally, press the power button. The light will turn off, and the robot is now shutdown.

Light off = robot power is off



Teaching Pendant Tips

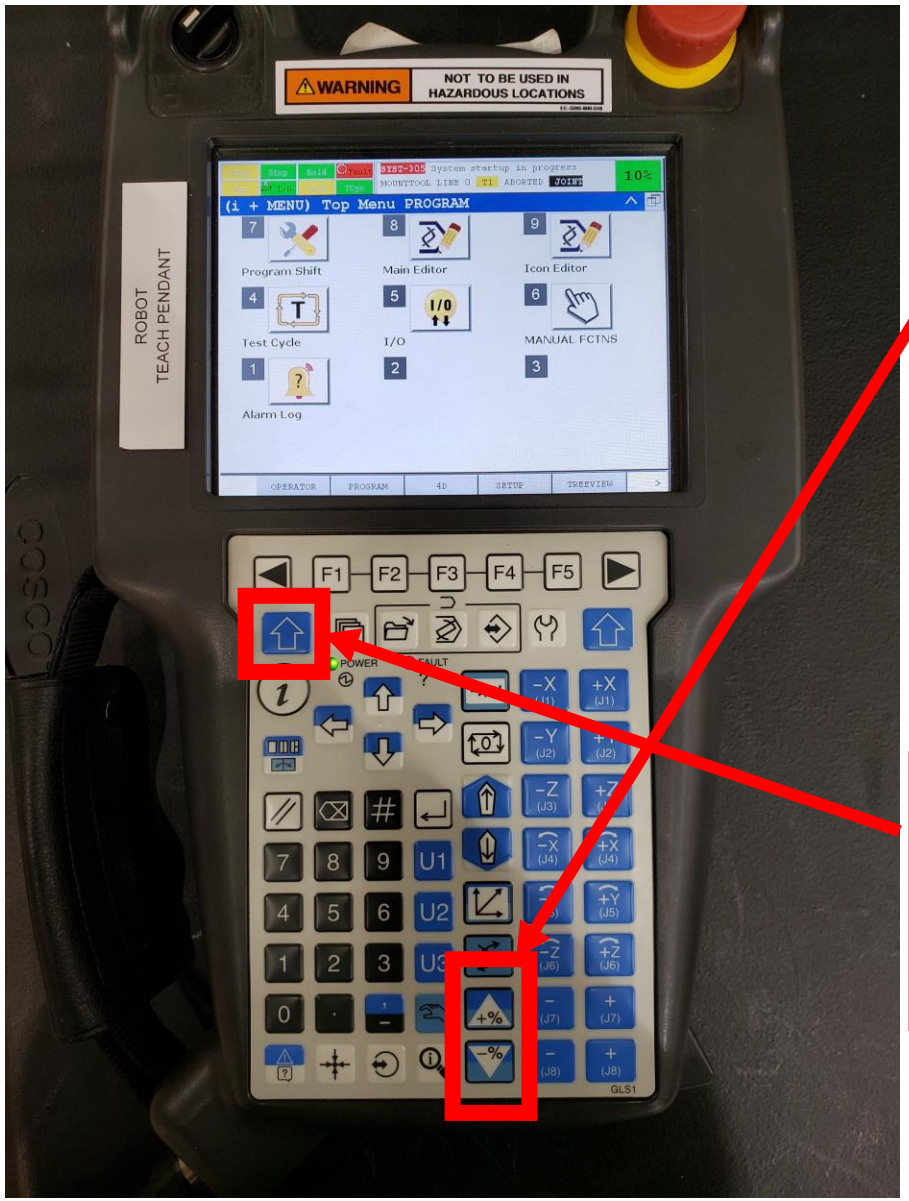
- Change speed
- Clear a fault
- Abort a program
- Jog the Robot (Move it’s joints via the teaching pendant)

Teaching Pendant Tips—Change Speed (Photo Guide Below)

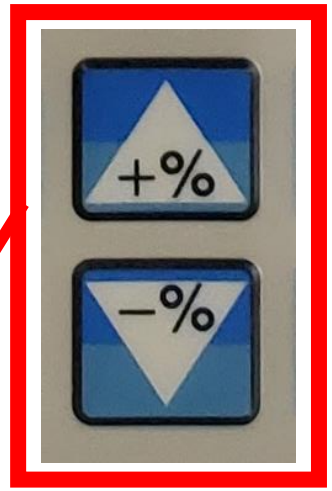
- Press the up/down speed keys to change the speed. Pressing and holding the “shift” key while you do this lets you change the speed by greater amounts.

Teaching Pendant Tips—Change Speed

Press the up/down speed keys to change the speed. Pressing and holding the “shift” key while you do this lets you change the speed by greater amounts.

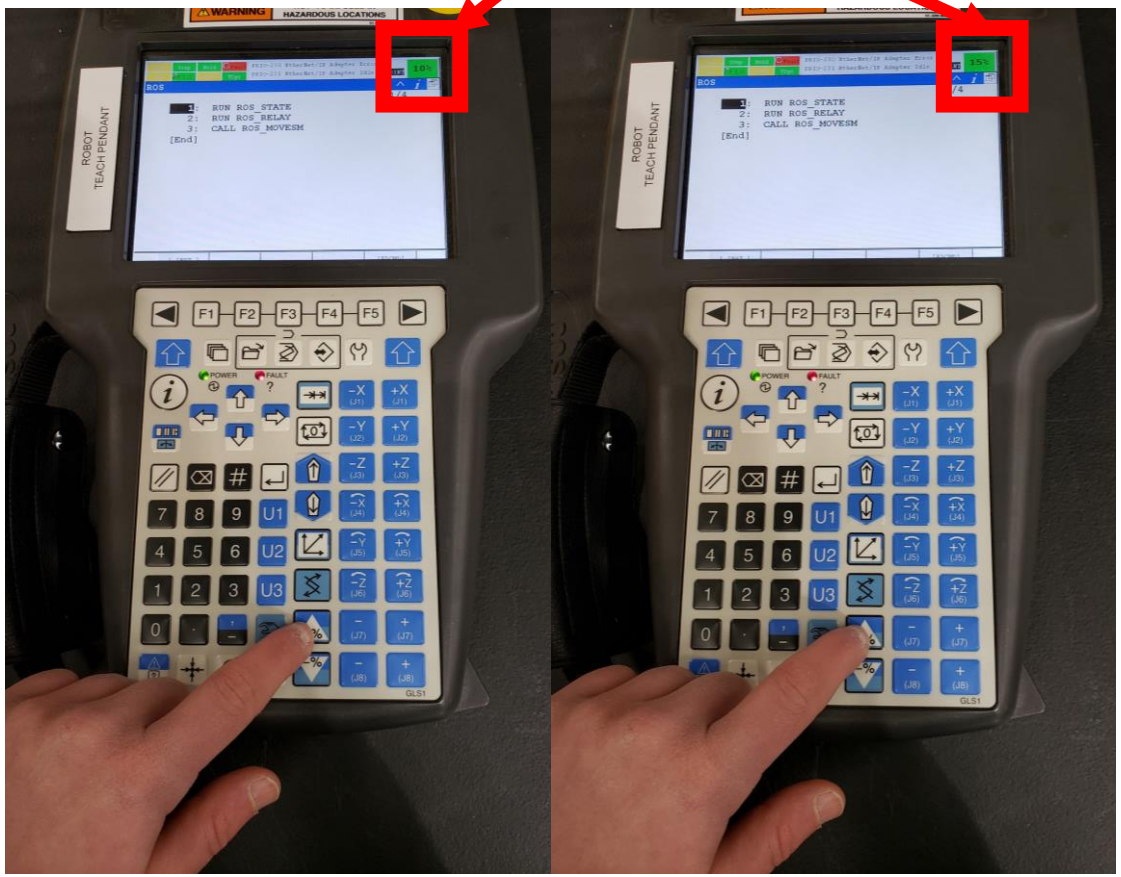


Up/Down Speed Keys



Shift Key

Speed Increase



Teaching Pendant Tips—Clear a Fault (Photo Guide Below)

- Press and hold the shift key and press the ‘reset’ key. Note: If you are not in auto mode yet (but the teach pendant is turned off), the fault will persist unless you are in auto mode.

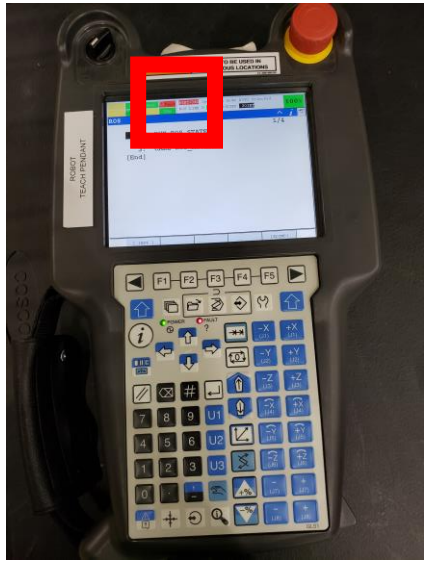
Teaching Pendant Tips—Clear a Fault

Press and hold the shift key and press the ‘reset’ key. Note: If you are not in auto mode yet (but the teach pendant is turned off), the fault will persist unless you are in auto mode.

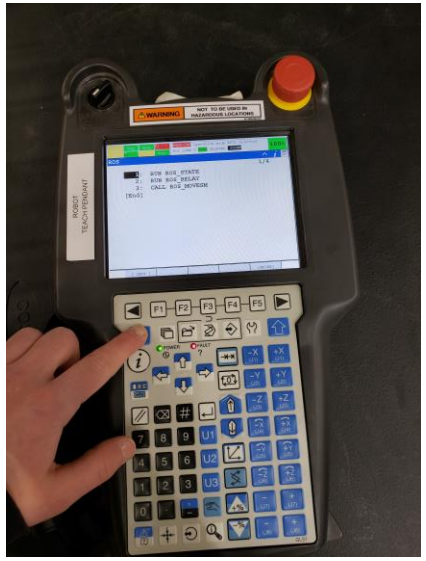
Reset Key

Shift Key

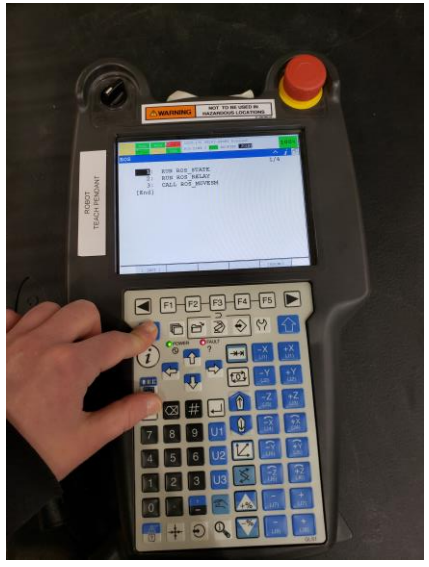
Fault (Red Text at top)



Shift pressed



Reset pressed while shift is pressed



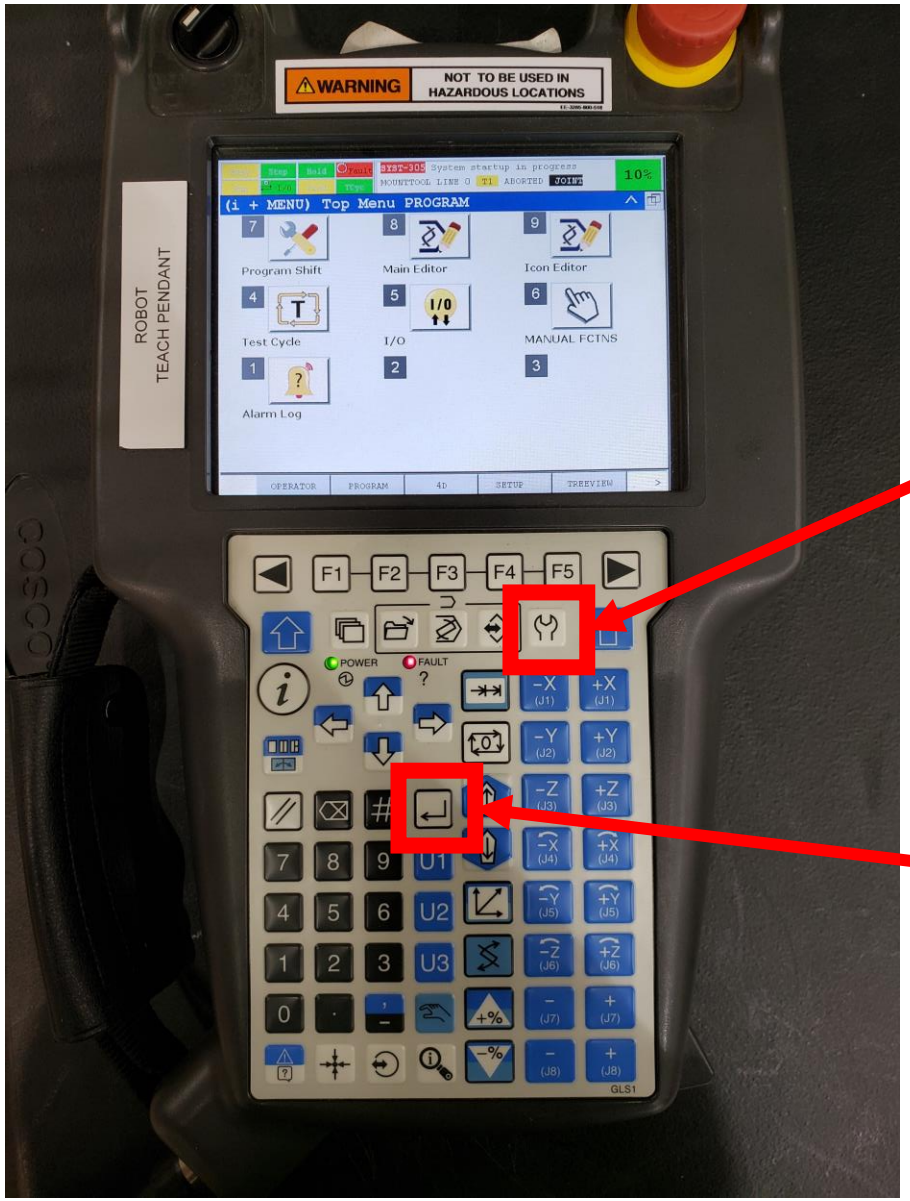
Fault cleared (red is now green text at top)

Teaching Pendant Tips—Abort a Program (Photo Guide Below)

- Press the function key. The first option that will be highlighted when the menu comes up will be “Abort”. Press Enter. Do this twice if the program you are aborting is the “ROS” program.

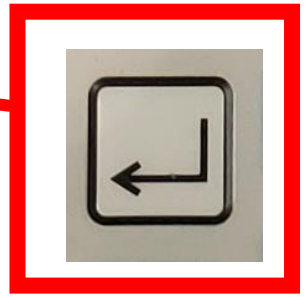
Teaching Pendant Tips—Abort a Program

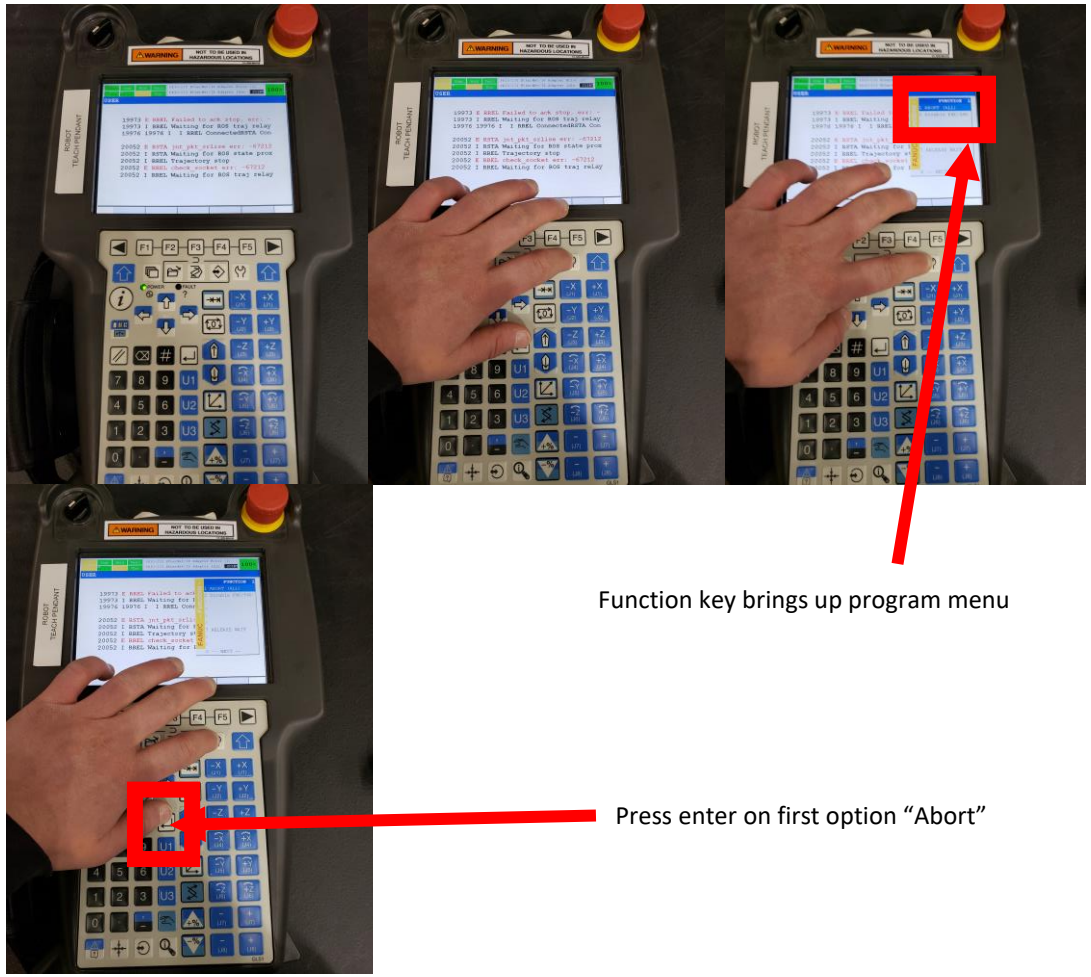
Press the function key. The first option that will be highlighted when the menu comes up will be “Abort”.
Press Enter. Do this twice if the program you are aborting is the “ROS” program.



Function key

Enter key





Function key brings up program menu

Press enter on first option “Abort”

Teaching Pendant Tips—Jog the Robot (Photo Guide Below)

- Make sure the robot is in T1 or T2 mode.
- Turn on the teaching pendant.
- With both hands, press down the Deadman's switch (located on the back). This will need to be held down the entire time operating the robot in T1 or T2 mode.
- Clear any faults (Make sure deadman’s switch is held down still!)
- While pressing the shift key, press any of the joint +/- keys to move the joints. (Make sure deadman’s switch is held down still)
- You can adjust the speed for jogging if desired. T1 limits the robot’s speed.

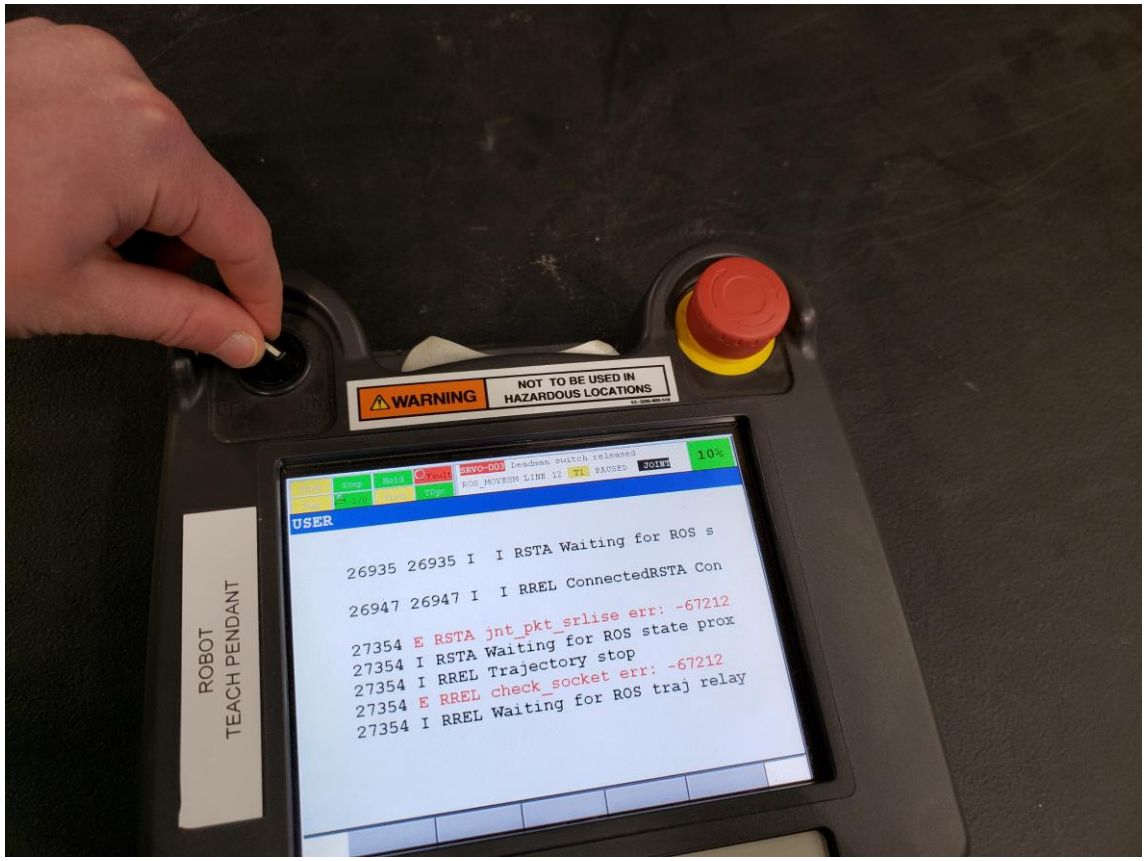
Teaching Pendant Tips—Jog the Robot

Make sure the robot is in T1 or T2 mode.

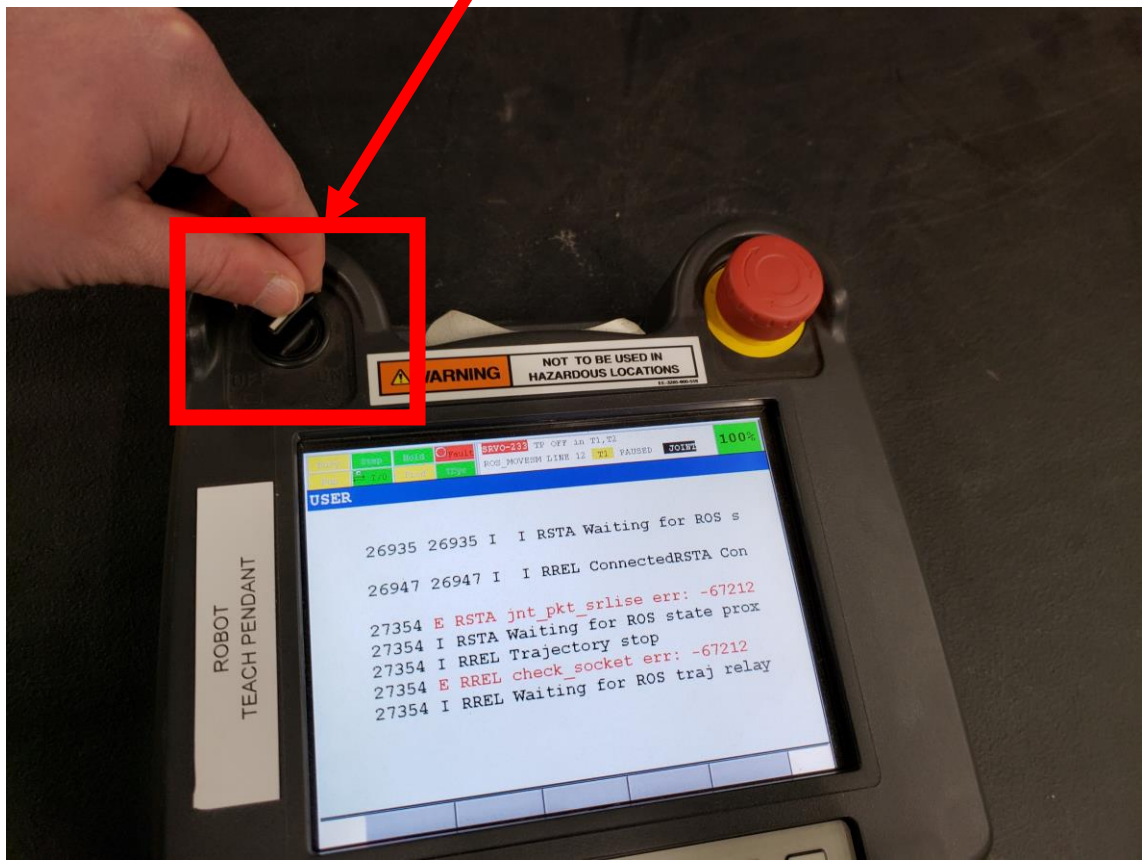
Make sure mode is T1 or T2.

Teaching Pendant Tips—Jog the Robot

Turn on the teaching pendant.



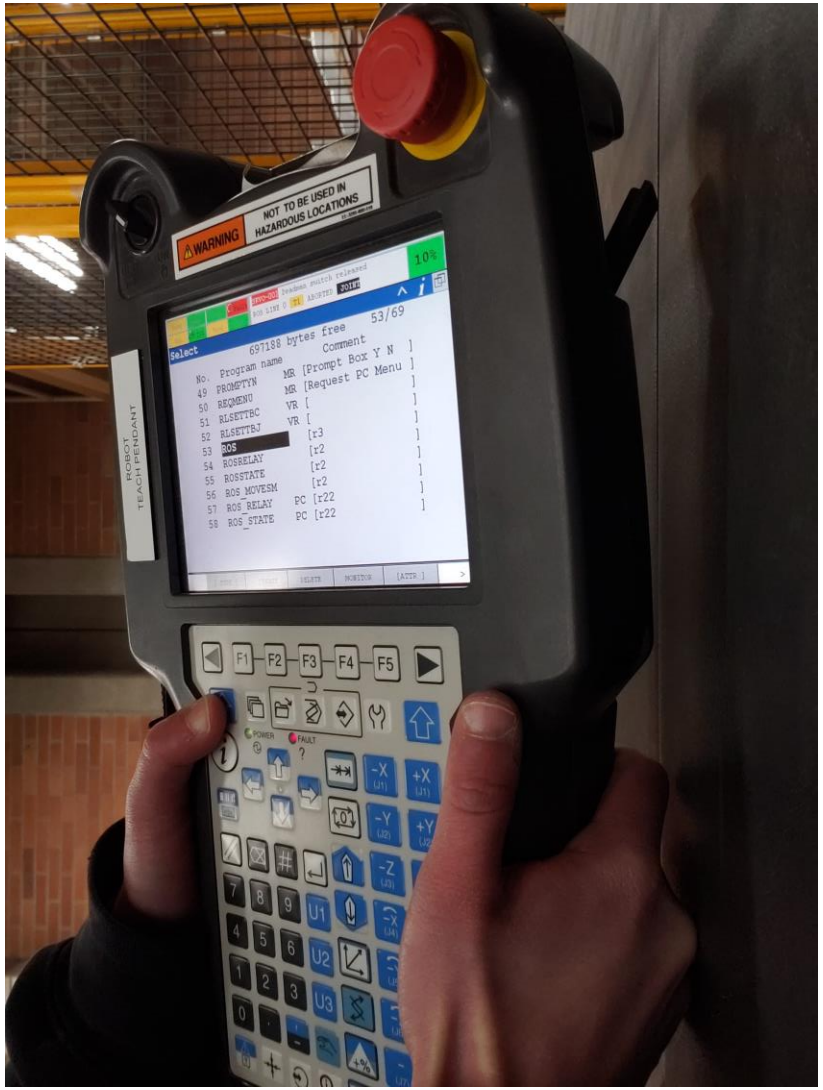
TP “on” position



Teaching Pendant Tips—Jog the Robot

With both hands, press down the Deadman's switch (located on the back). This will need to be held down the entire time operating the robot in T1 or T2 mode.

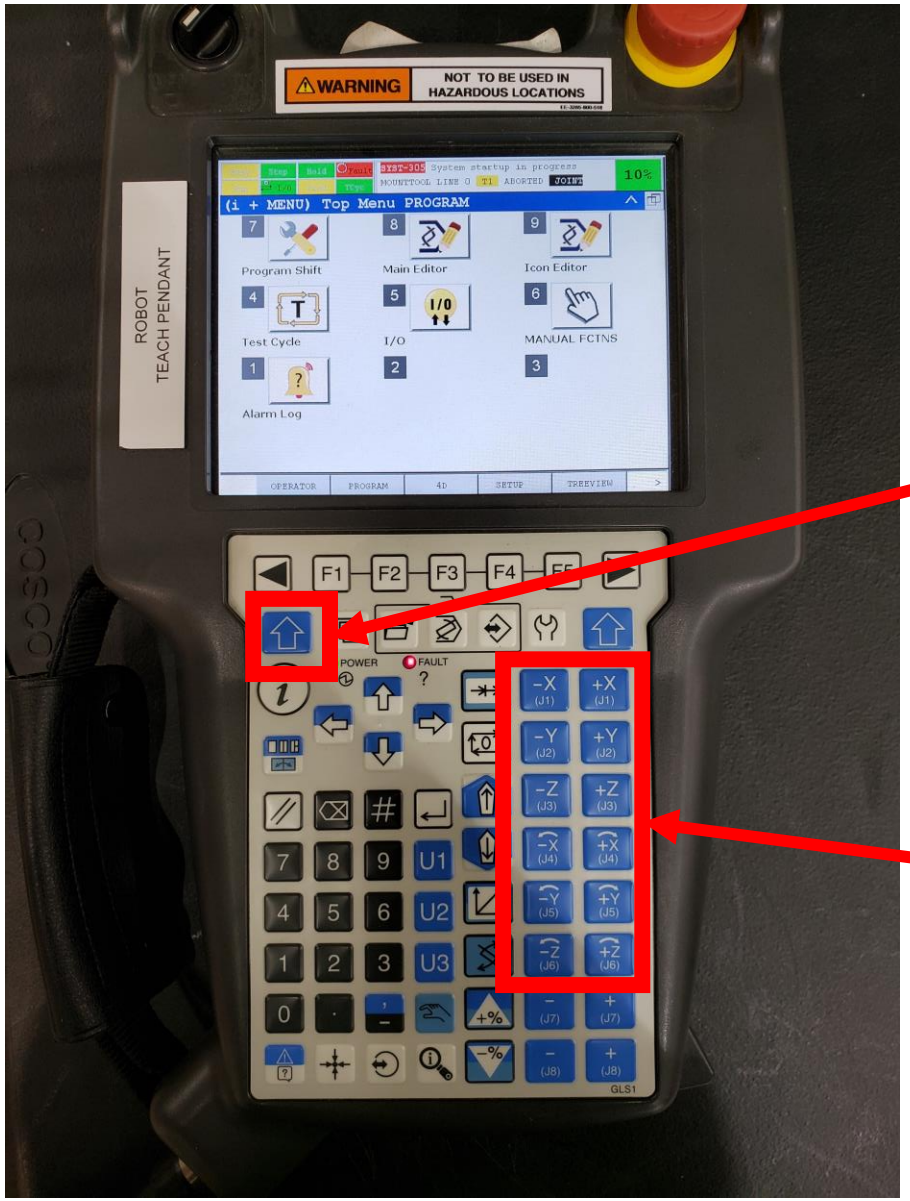
Deadman's switch on back



Pressing Deadman's switch With both hands

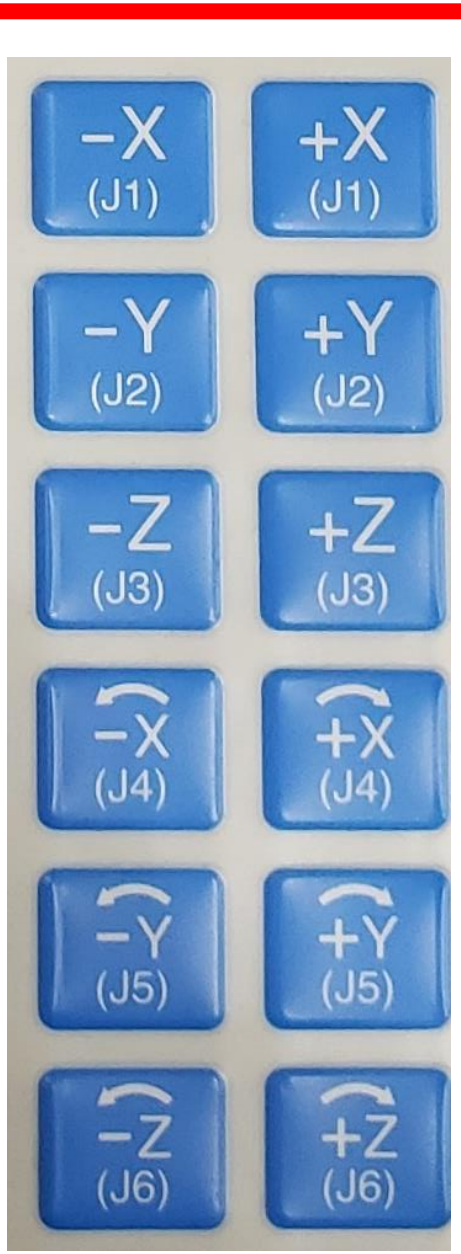
Teaching Pendant Tips—Jog the Robot

Clear any faults. While pressing the shift key, press any of the joint +/- keys to move the joints.
You can adjust the speed for jogging if desired. T1 limits the robot's speed. (Make sure deadman's switch is held down still!)

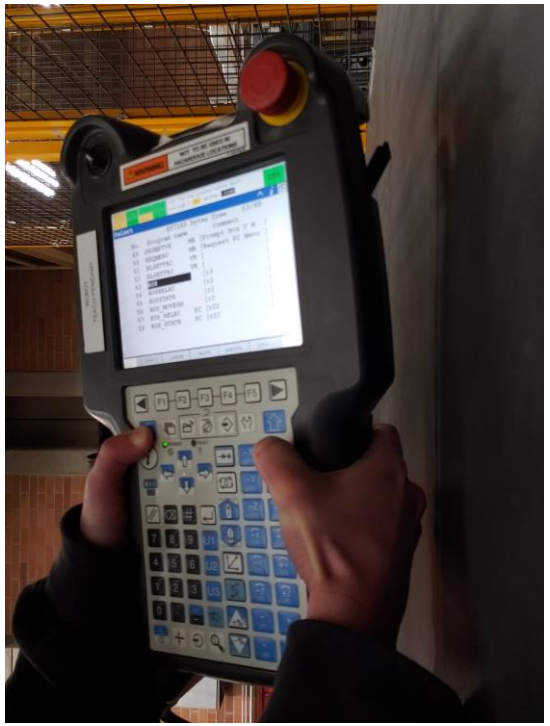


Shift key

Joint Keys (in joint order, top = base rotation)



Jogging the Robot



Using the Simulator

- Using the simulator is very similar to using the actual robot, but the control panel and teaching pendant are represented virtually within the software “RoboGuide”. Use the steps on the following slides to use the simulator software in RoboGuide.

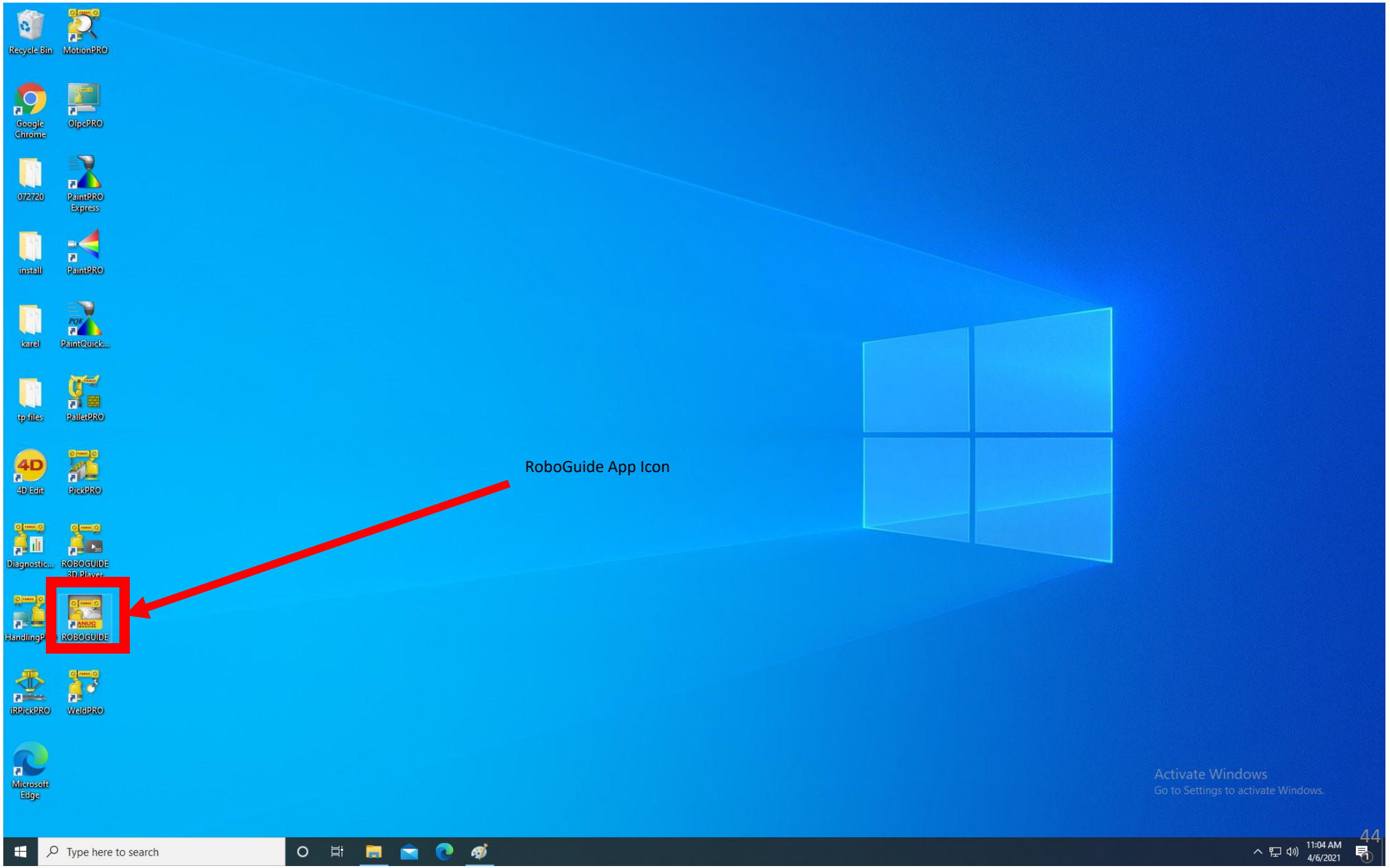
Using the Simulator

- Start the “RoboGuide” software on the Windows computer you are using.
- The RoboGuide Application will come up with one or more options for workcells. The “KCL_Robot” workcell should work, so click this.
- The robot cell will load. This may take a few minutes.
- The workcell will come up with a 3D model of the robot. The teaching pendant and control panel may already be on the screen. If so, great. If not, you will want to bring them up.

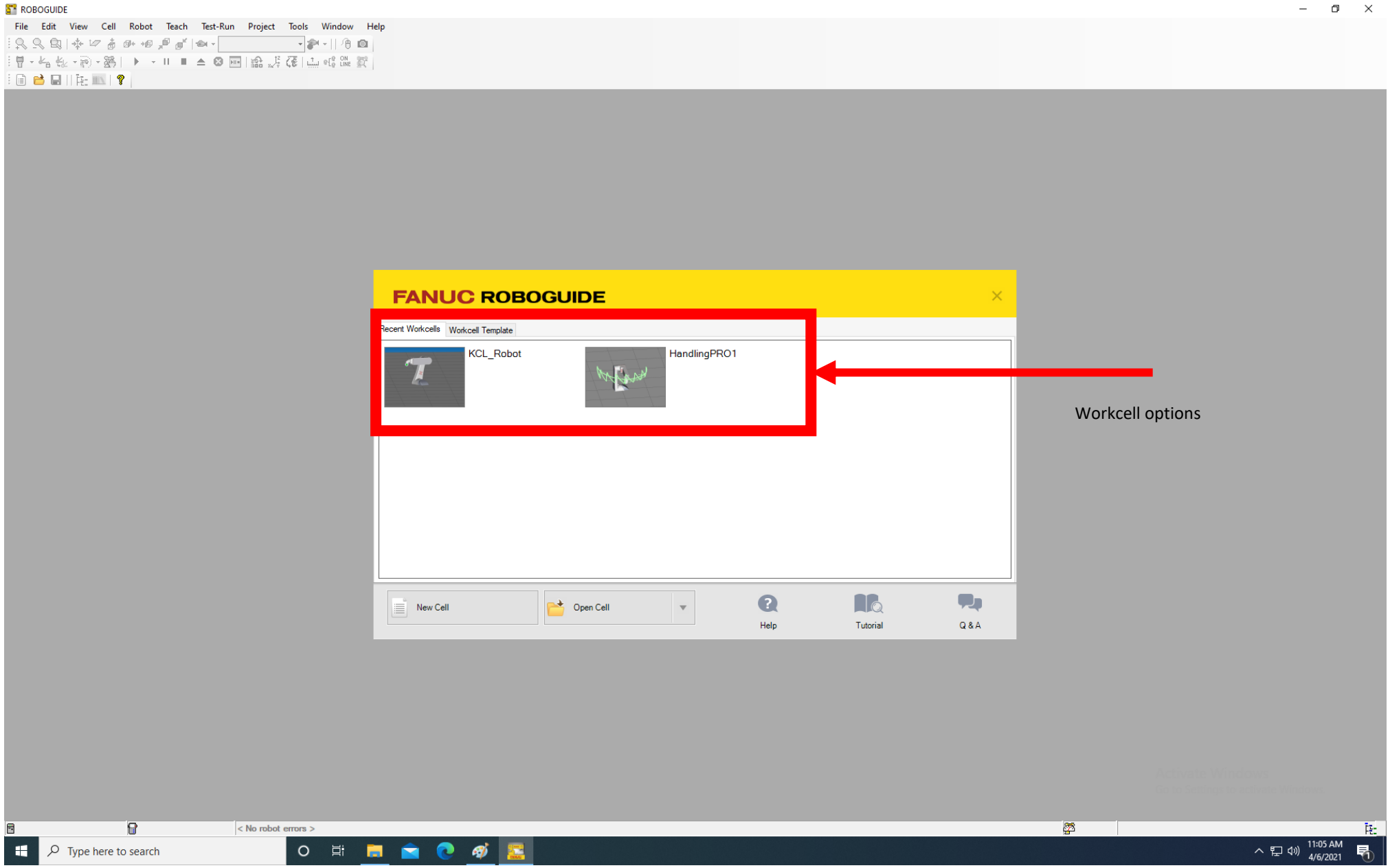
Using the Simulator

- Click the teaching pendant icon to bring up the virtual teaching pendant.
- You can then follow the same steps on the virtual teaching pendant as the physical robot teaching pendant to bring up the ROS program.
- Click the Run Panel icon to bring up the virtual control panel.
- Click the Triangle “Run” button on the virtual control panel to start the program. This is analogous to the green “Cycle Start” button on the physical robot controller.

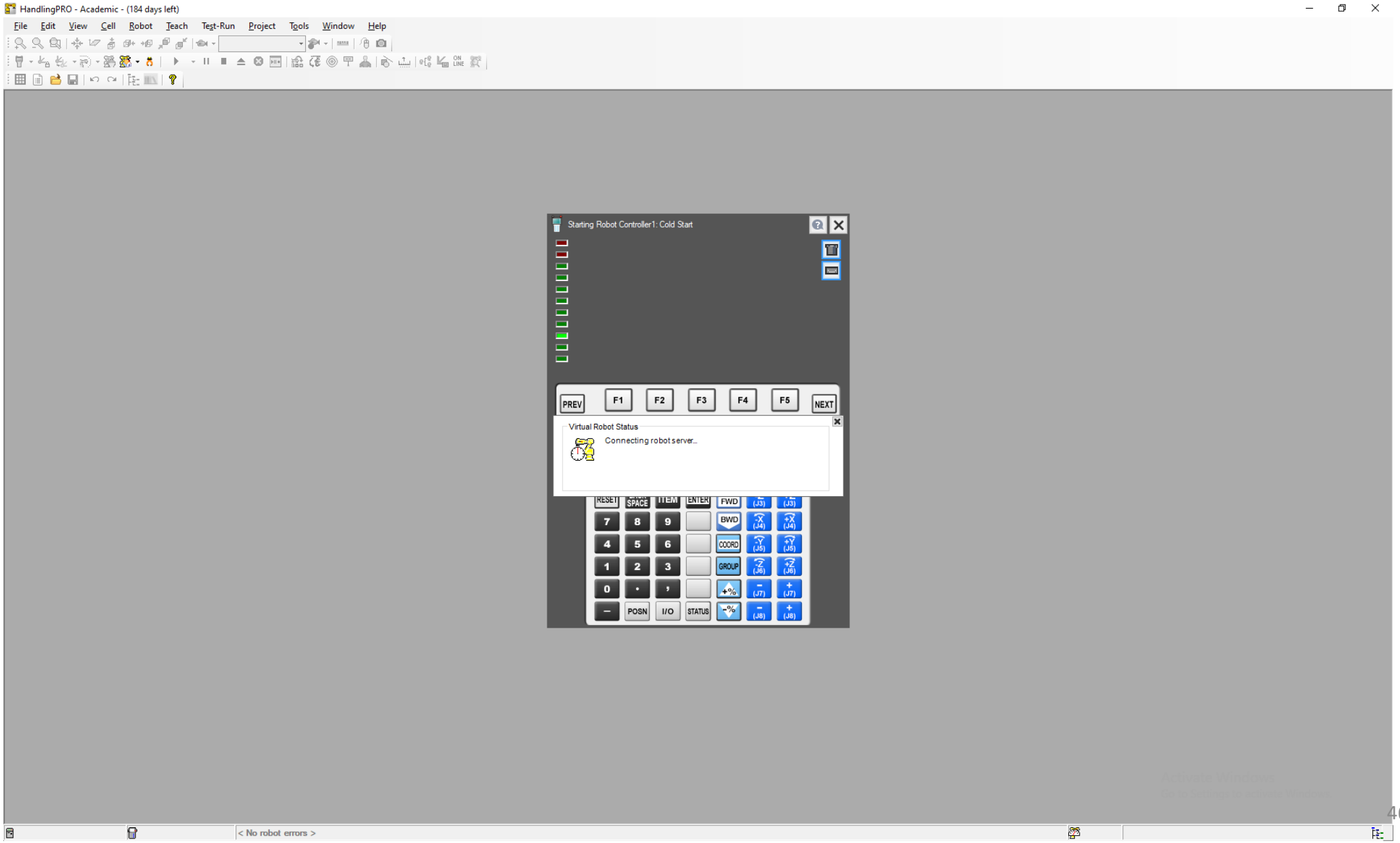
Start the “RoboGuide” software on the Windows computer you are using.



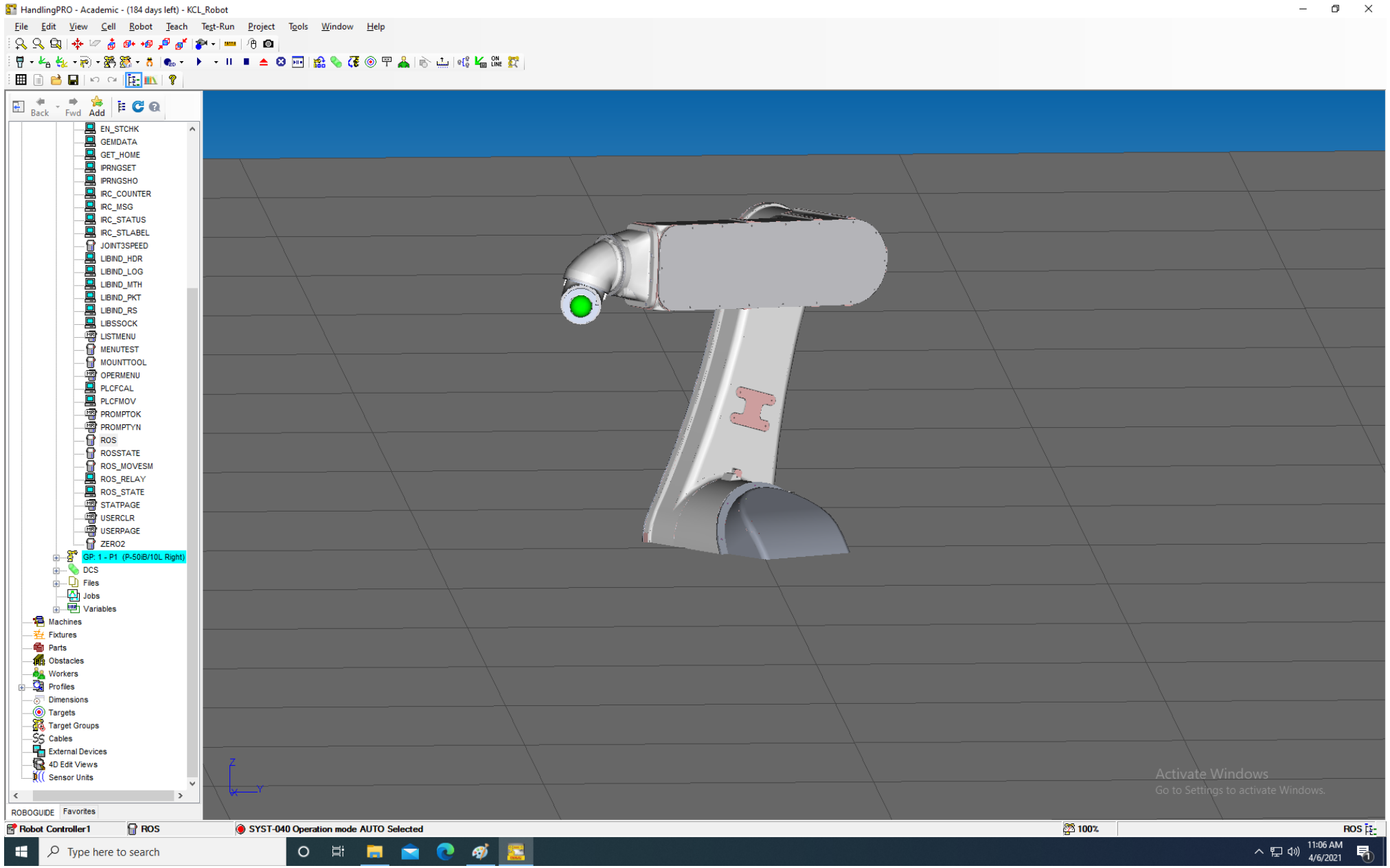
The RoboGuide Application will come up with one or more options for workcells. “KCL_Robot” workcell should work, so click this.



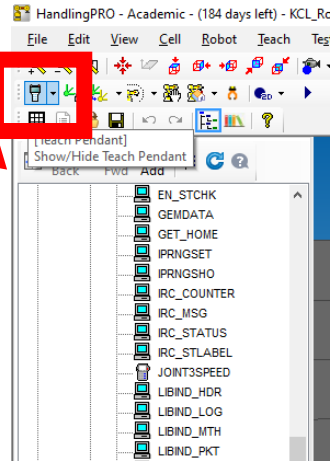
The robot cell will load. This may take a few minutes.

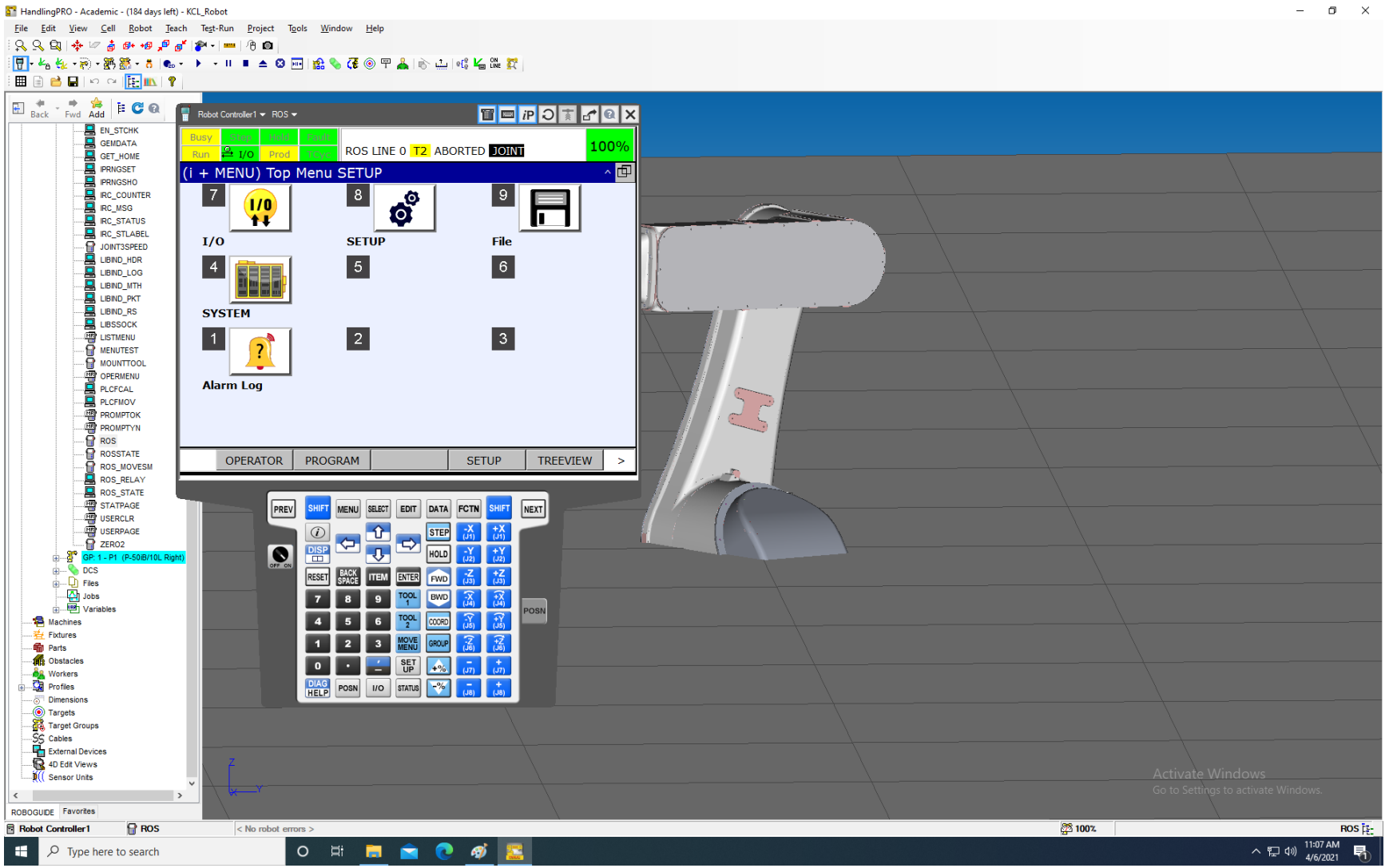


The workcell will come up with a 3D model of the robot. The teaching pendant and control panel may already be on the screen. If so, great. If not, you will want to bring them up.

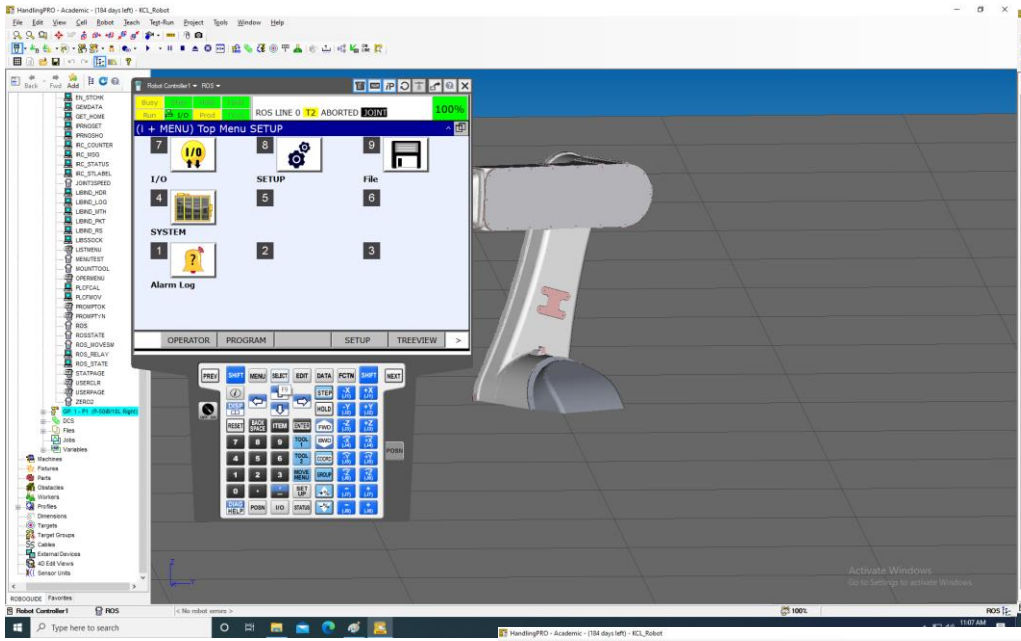


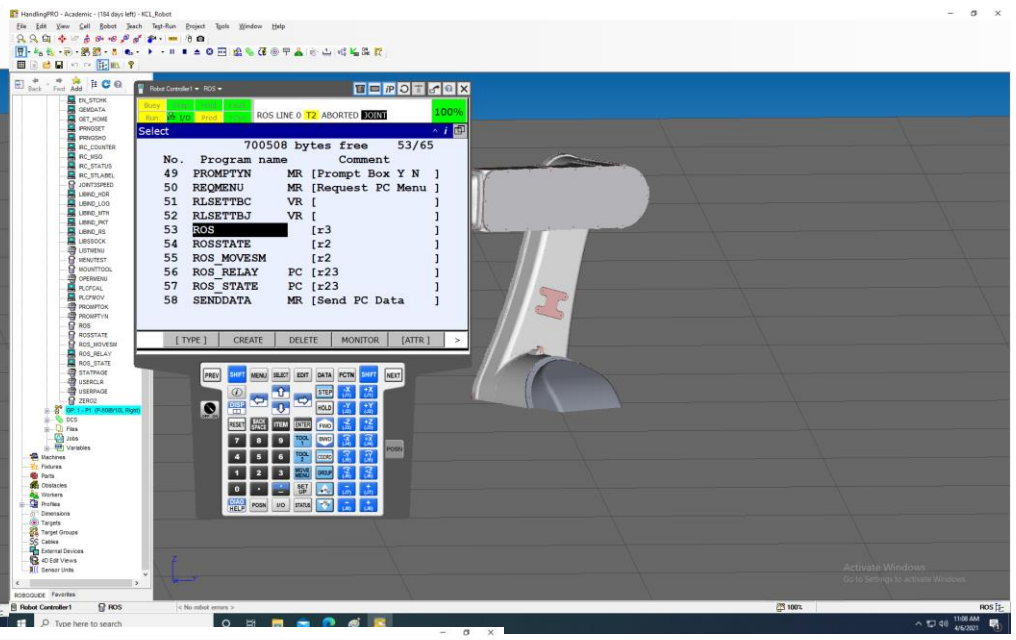
Click the teaching pendant icon to bring up the virtual teaching pendant.

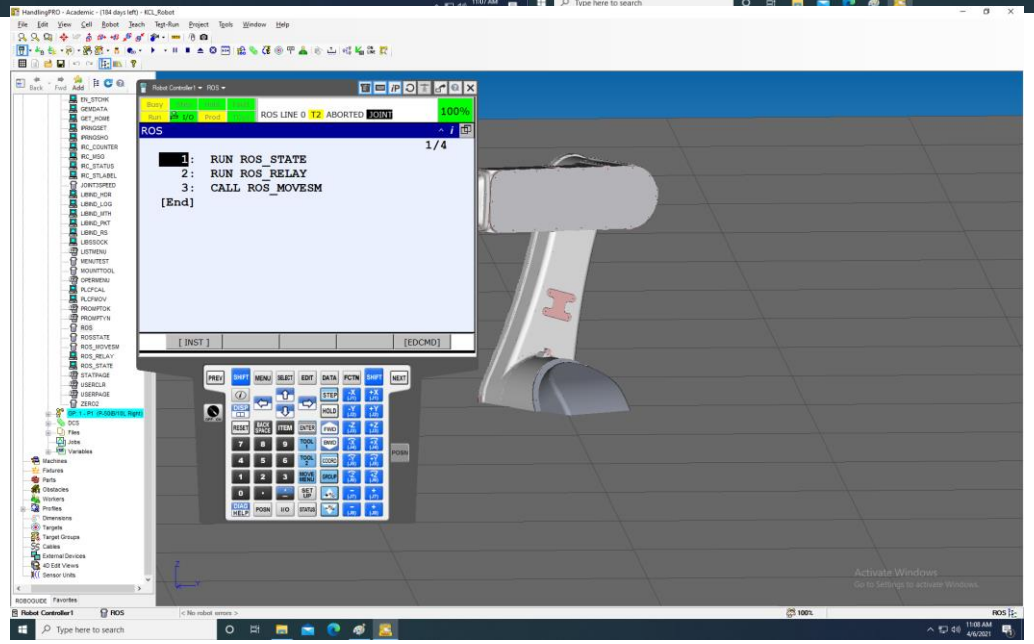




You can then follow the same steps on the virtual teaching pendant as the physical robot teaching pendant to bring up the ROS program.

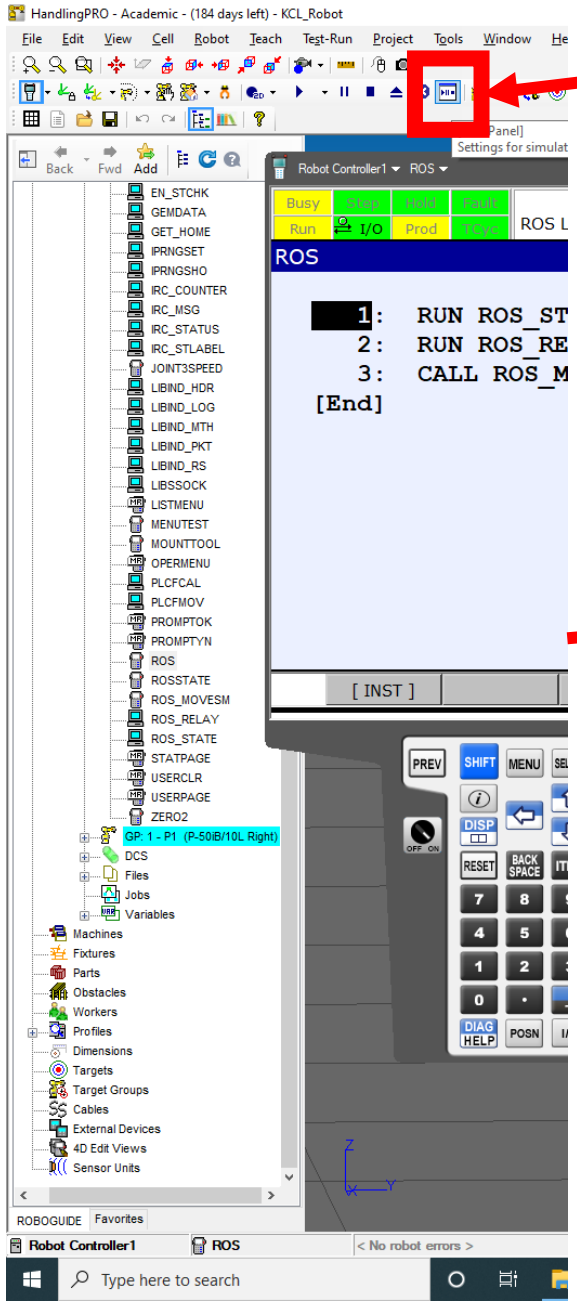


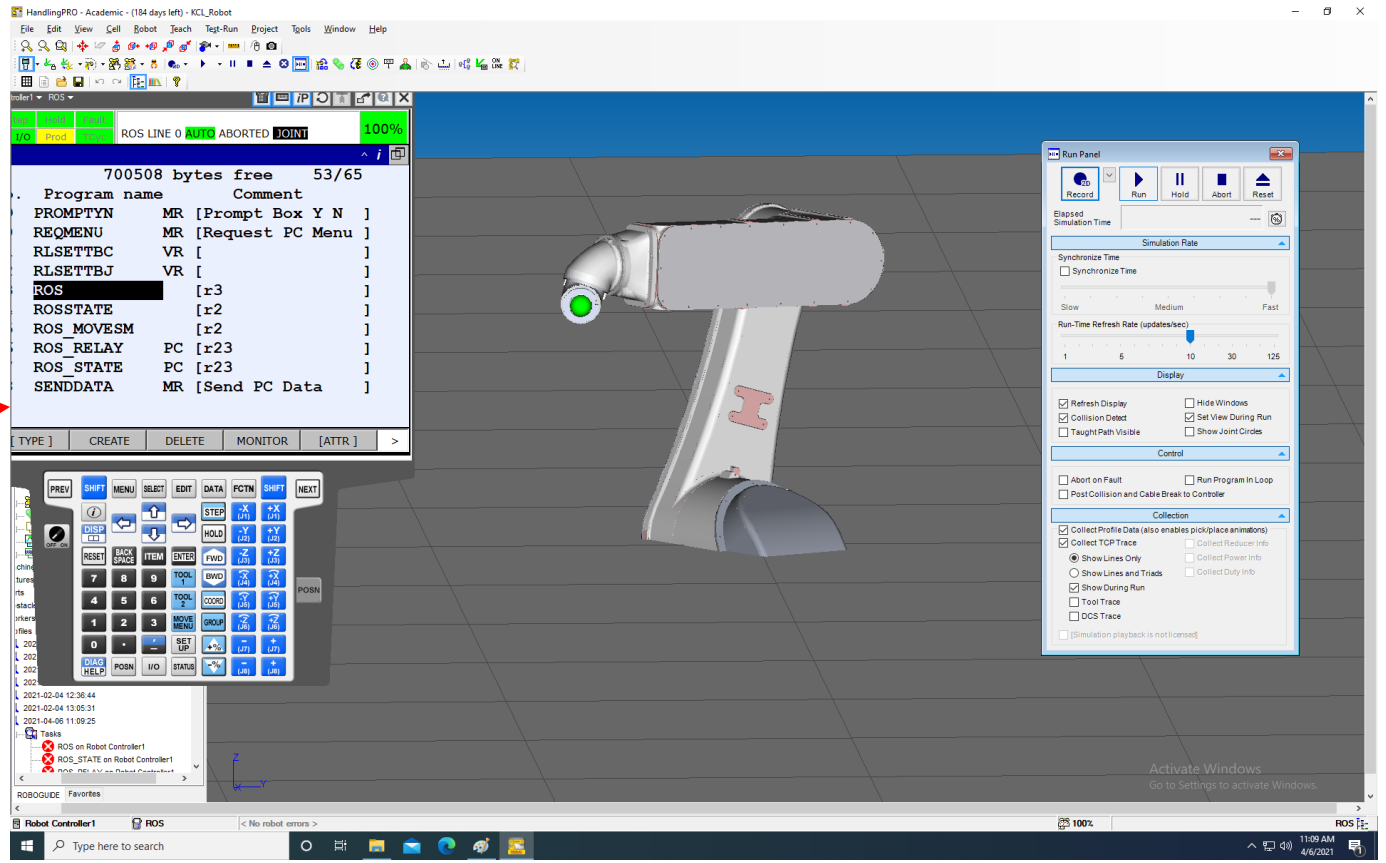




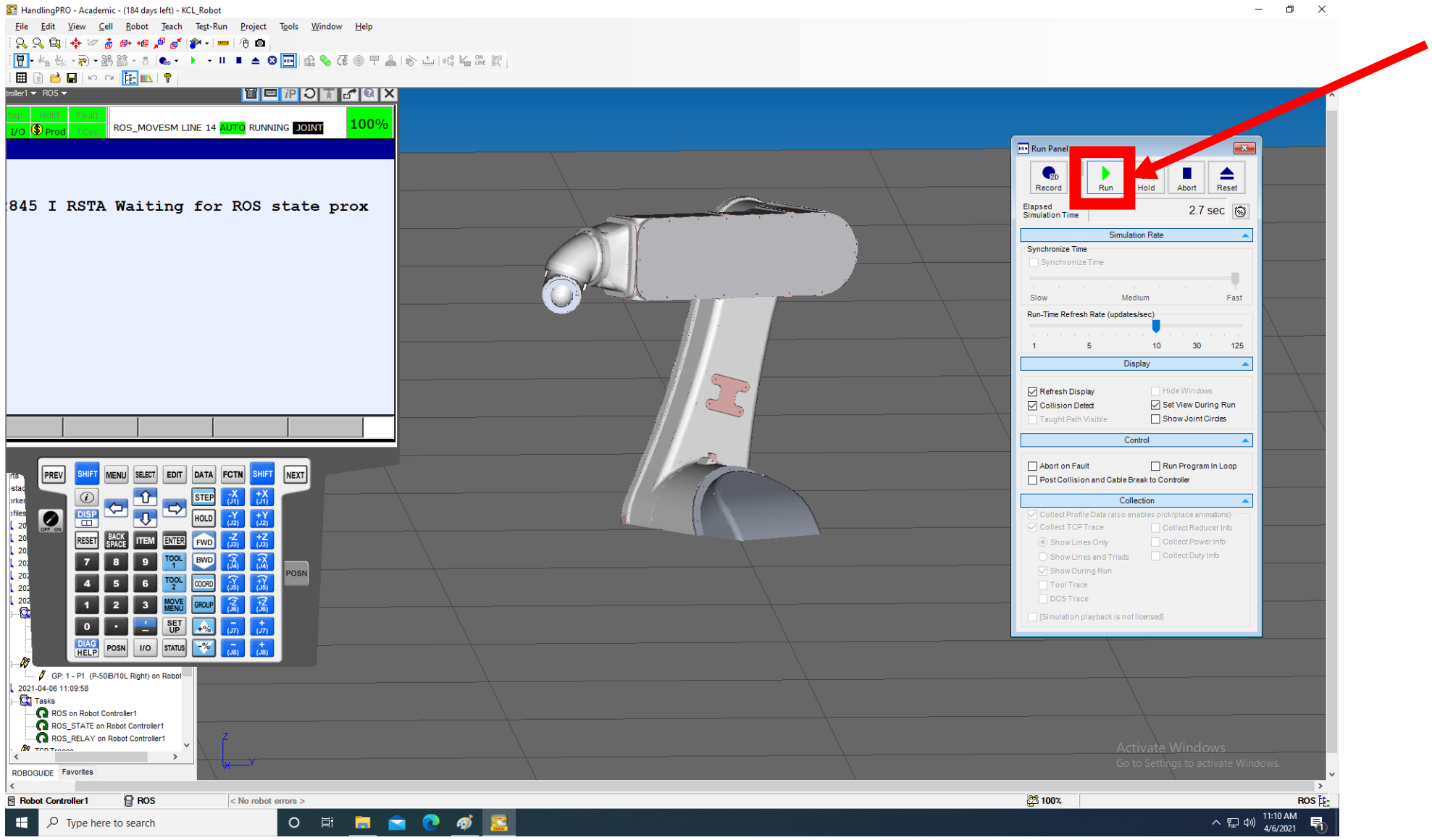
Click the Run Panel icon to bring up the virtual control panel.

Run panel icon





Click the Triangle “Run” button on the virtual control panel to start the program. This is analogous to the green “Cycle Start” button on the physical robot controller.



Controlling the FANUC robot in software

- The following applies to controlling the robot using an outside computer

How to Launch the Robot State Program—Controller Computer

- This is a multi-step process. Follow the instructions in the following slides.

What is the Robot State Program?

- The robot state program launches the ROS-FANUC driver on the ROS end. This program listens for messages sent by the MoveIt program and executes them.
- You can optionally specify to launch RVIZ with the robot state program as well (generally, you will want to launch RVIZ, and it does so by default). You can control the robot's movements by communicating to the robot state program in two ways:
- 1. Use the RVIZ program that was launched (useful for testing)
- 2. Use the Python controller class (what we generally do for software/demos)

Namespaces in the Robot State Program

- When you launch the robot state program, you will do so in one of the four namespaces:
- real_robot – The actual, physical robot. You will need a password to launch this, and your ROS environment should be connected to the main VMAL network (as the master).
- sim_1 –The simulator inside the Den Robot Lab
- sim_2 –The simulator outside in the Den common area
- just_rviz –This only launches rviz (doesn't hook up to real simulator or controller)

Namespaces in the Robot State Program

- When you are launching the robot state program, MAKE SURE YOU ARE LAUNCHING IT IN THE CORRECT NAMESPACE. The bash files you will use are labeled with their corresponding namespace. Make sure you are using the correct one! For testing, you should use the just_rviz namespace.
- If you launch the robot state in the wrong namespace, you could start controlling the wrong robot—This could mess with someone else trying to control the robot/simulator that you are accidentally inside. Always be sure you are in the right namespace.

Launching the Robot State Program

- There are several bash scripts that launch the correct robot state program inside the ‘roboLaunch’ folder (this can be setup anywhere in your computer—it doesn’t actually have to be inside a fanuc package.

- All of the bash files launch the moveit_planning_execution.launch folder with the parameters specified. Each file has a slightly different set of parameters and/or IP Address.

Description of the Robot State Shell Scripts

- The shell scripts have the following command to launch the moveit_planning_execution.launch file
- Roslaunch fanuc_p50ib_moveit_config moveit_planning_execution.launch module:={parameter} sim:={parameter} robot_ip:={parameter}
- All bash files that aren’t controlling the real robot will also have the argument use_bswap:= false
- Bash files that have no_rviz in the name will have the additional arguments use_rviz:=false and use_bswap:=false
- just_rviz.sh does not have the robot_ip argument.

Arguments to the launch file description

- module:=
- This argument describes the namespace of the robot. For example: module:=sim_1
- sim:=
- This argument specifies whether or not ros treats the robot as a simulator IN ROS. This argument is only true for the just_rviz.sh. Even though the simulators are, in fact, simulators, since they are OUTSIDE ROS they are not treated as a simulation WITHIN ros. Ex: sim:=false
- robot_ip:=
- This argument is the IP address of whatever you are talking to (which simulator or robot). just_rviz.sh does not have this argument. Ex: robot_ip:=129.101.98.238
- use_bswap:= This argument must be present and set to false for all files that aren't the real robot. Ex: use_bswap:=false
- use_rviz:=
- This argument specifies whether or not to bring up the RVIZ environment. For a raspberry pi, you probably do not want to bring up RVIZ (computationally expensive).

Shell Scripts

- real_robot.sh – this brings up the actual, physical robot state program on whatever machine you launch it on. This file is password protected. It is recommended to run this program on Rosbox 2 (the master computer inside the cage), because this computer has an ethernet connection with the robot and is therefore more stable to communicate with it. In the future, it is likely that this particular environment will be forced to run on the roscore computer. (password protected)
- real_robot_no_rviz.sh – same as real_robot.sh but without rviz (password protected)
- sim_1.sh – this brings up the robot state program connected to the first simulator
- sim_1_no_rviz.sh – same as sim_1.sh but without rviz
- sim_2.sh –this brings up the robot state program connected to the first simulator
- just_rviz.sh –this brings up the robot state program as a simulation within ROS, only connected to RVIZ.

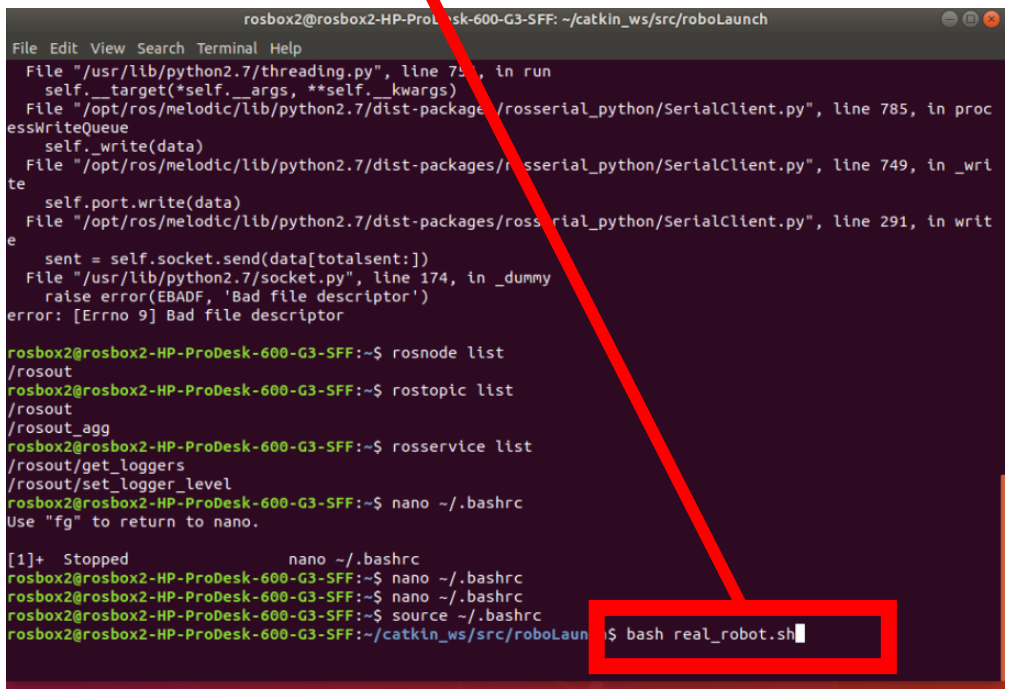
Actually Launching the Robot State Program

- To launch the robot state program for whatever robot or simulation you wish to connect to, run the appropriate shell script:

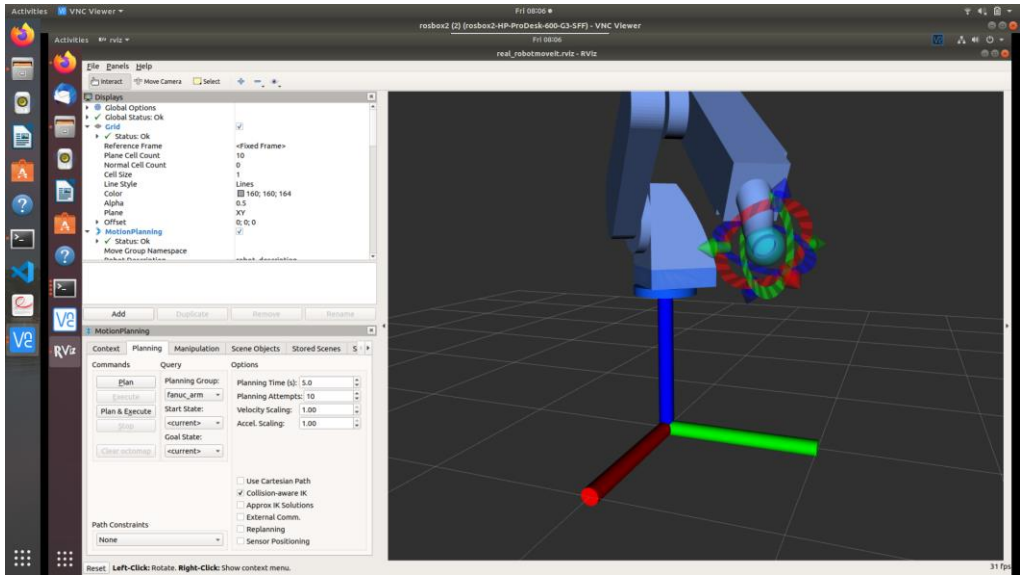
- >> bash real_robot.sh

- This should bring up RVIZ (if enabled) and print out a connection message to your terminal.

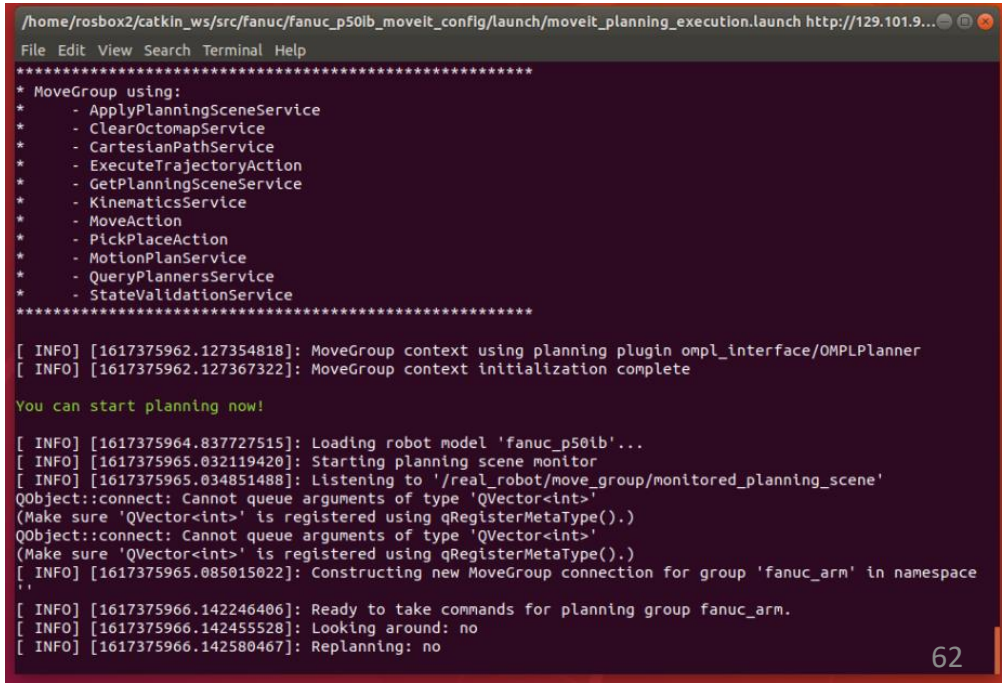
Launch robot state program—
>> bash real_robot.sh



This brings up RVIZ



And prints a
terminal Message



Controlling the Robot

- There are two ways to control the FANUC robot
- 1. Via RVIZ (good for testing, finding coordinates that are feasible in the robot state)
- 2. Via Python programming with the FanucInterface Class (what we typically use)

Controlling the Robot Via RVIZ

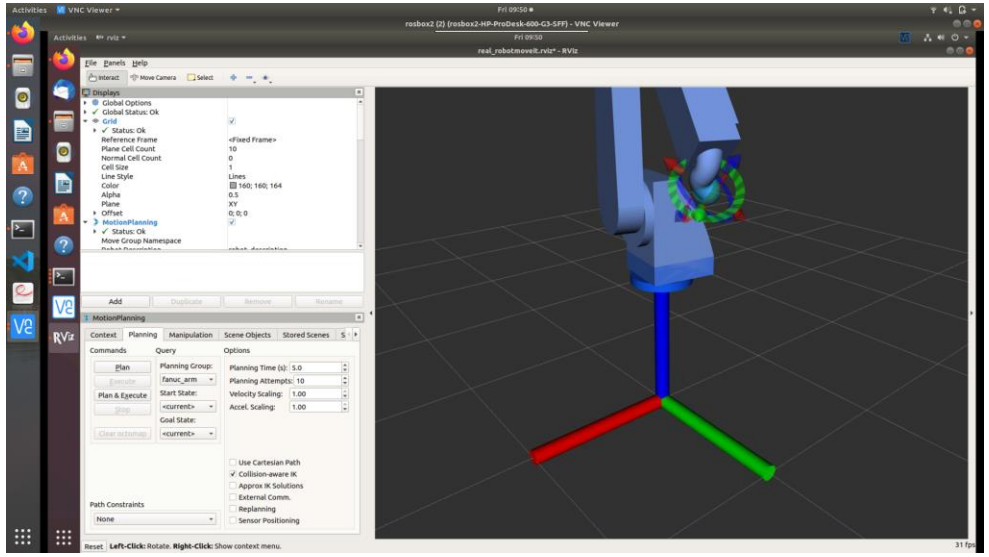
- It is useful to control the robot via rviz when attempting to see what cartesian point the robot is at given any particular position. This is especially useful when trying to determine the end-effector's orientation to reach a given point.

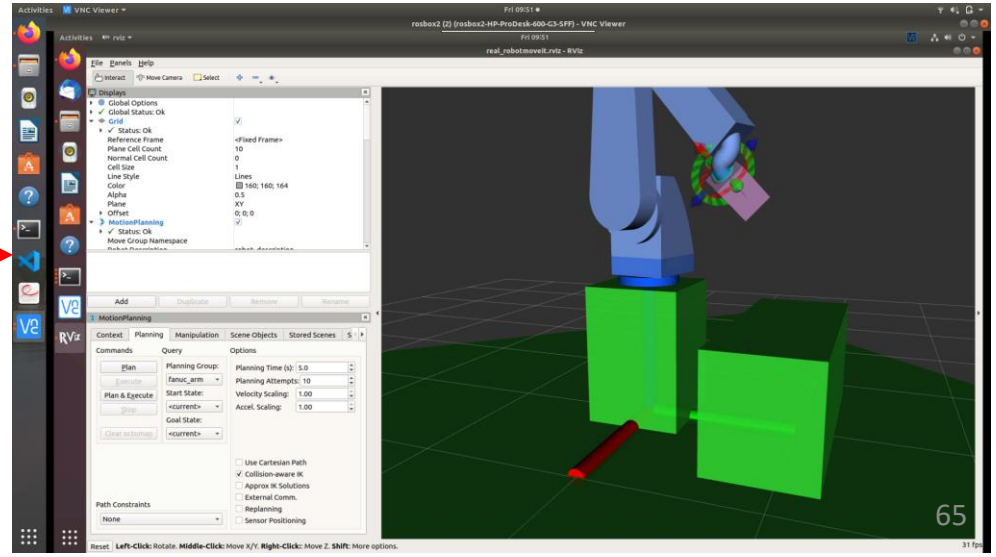
- To start, go to the roboLaunch folder and execute the bash script corresponding to whatever robot/simulation you want to control.

>> bash just_rviz.sh

Controlling the Robot Via RVIZ

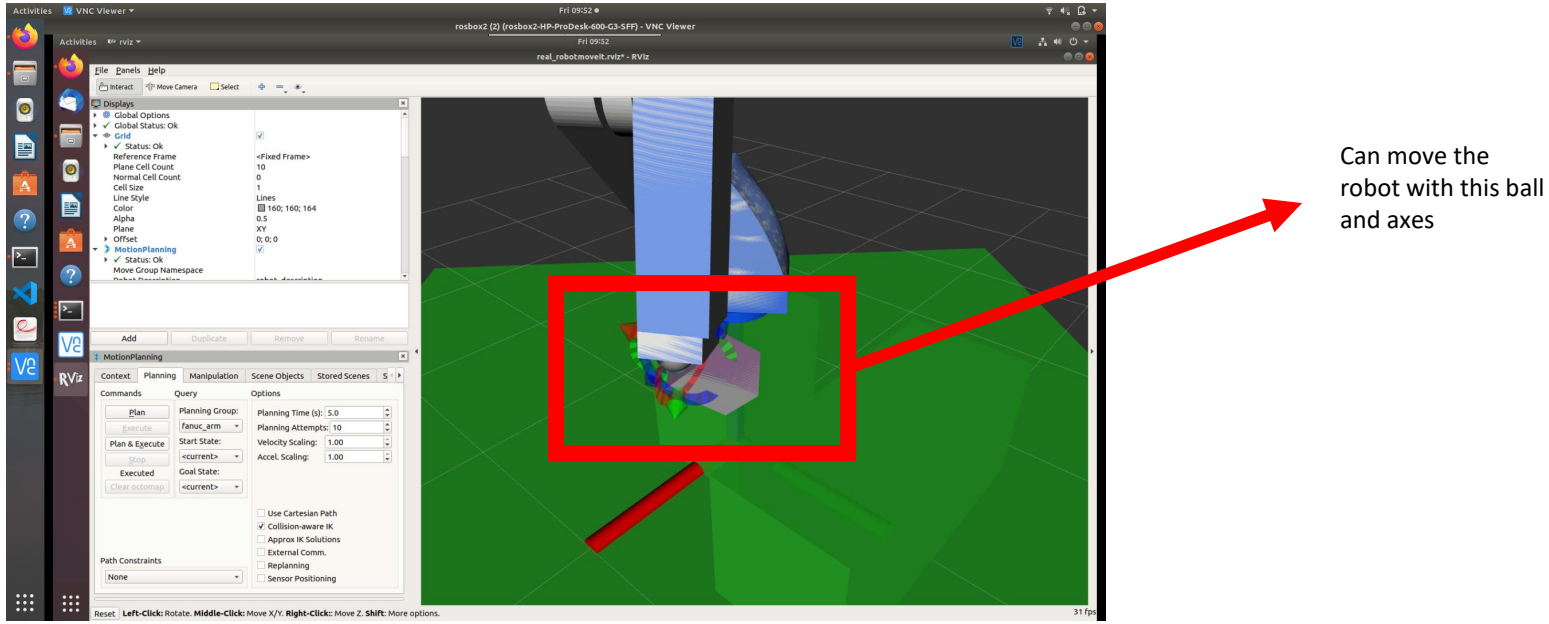
- RVIZ will come up. Make sure you also load in the scene objects to make sure you are working around the table and floor. To do this, run the setupScene.py script in the fanuc_demo package in the correct namespace.
- >> rosrun fanuc_demo setupScene.py __ns:=just_rviz





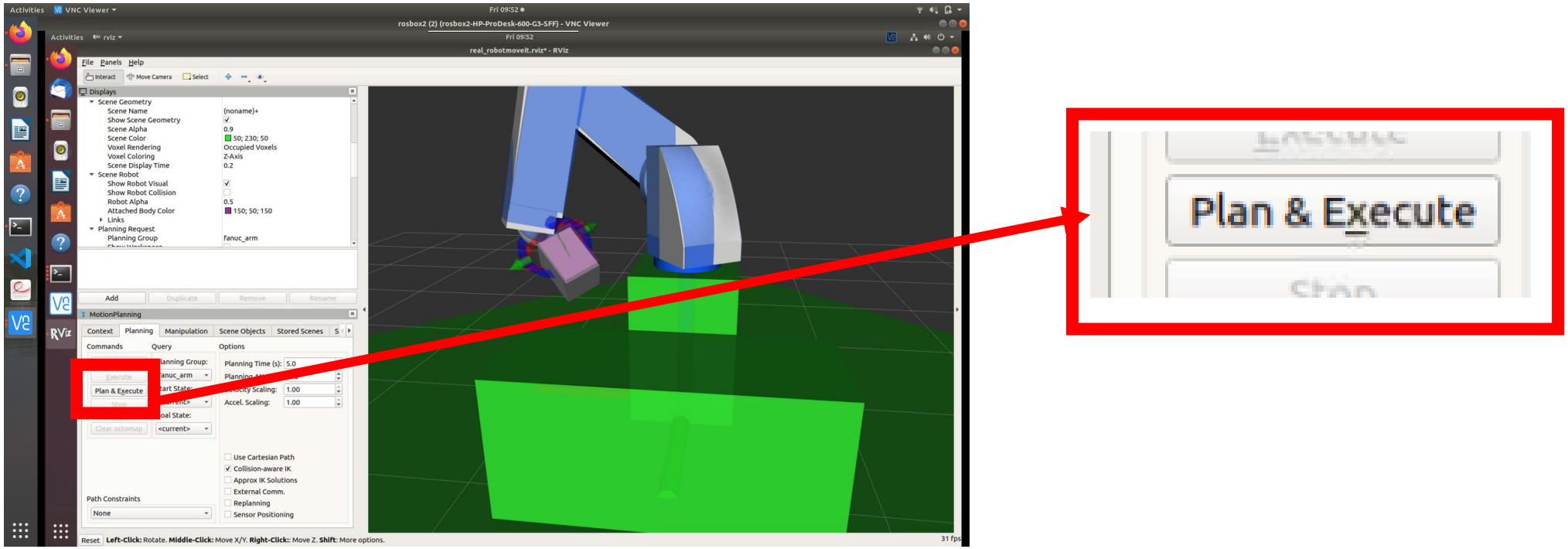
Controlling the Robot Via RVIZ

- You can grab the ball in the center of the end of the robot arm (onscreen) and drag it around to move the arm into different positions. The slider rings and arrows around the ball help you control a specific position/rotation axis.

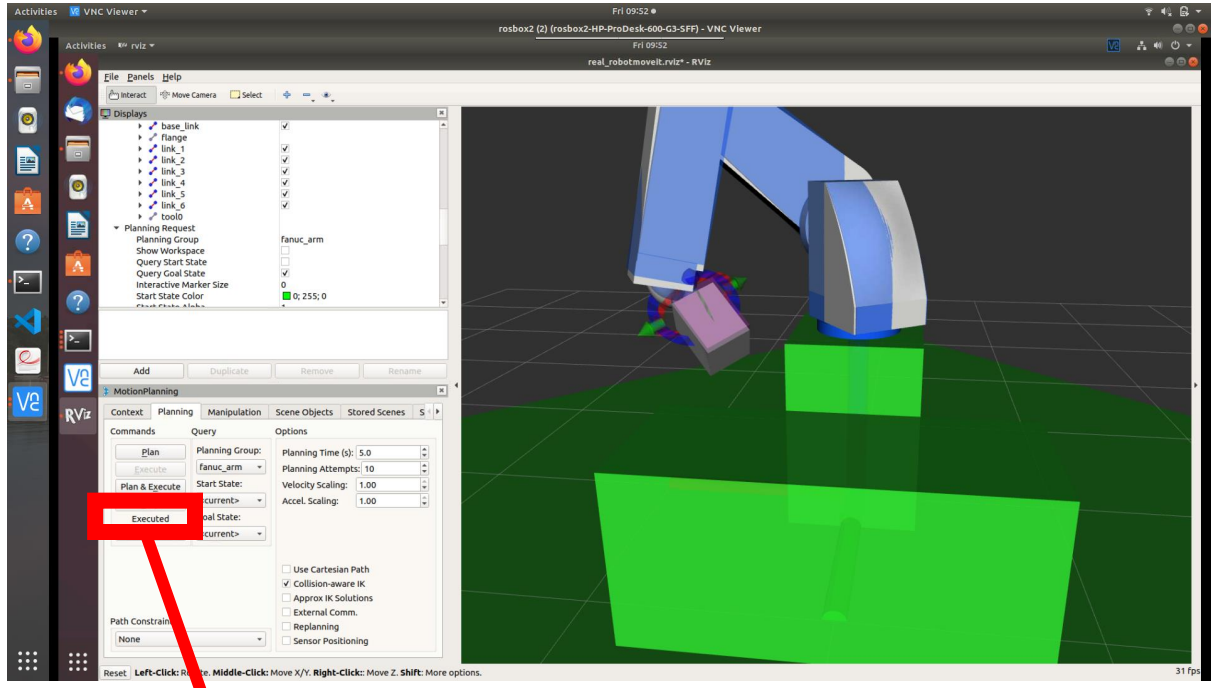


Controlling the Robot Via RVIZ

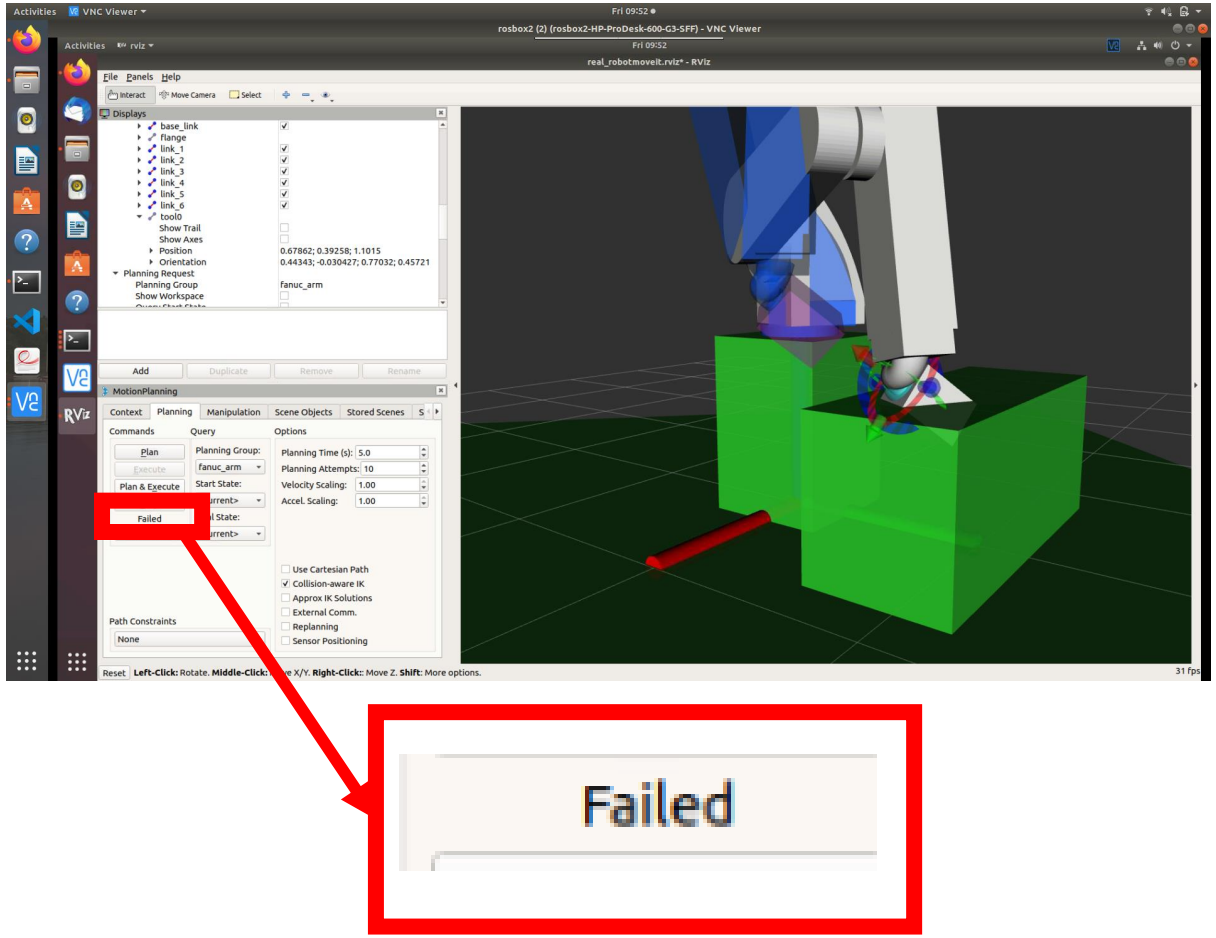
- When you have a position you like, click “Plan and Execute” in the Planning Tab of the Motion Planning window to have the robot move there. If is a feasible move, you will get a “Executed” message back. If not (or if there is a connection error), you will get a “failed” message back.



Successful Motion Execution



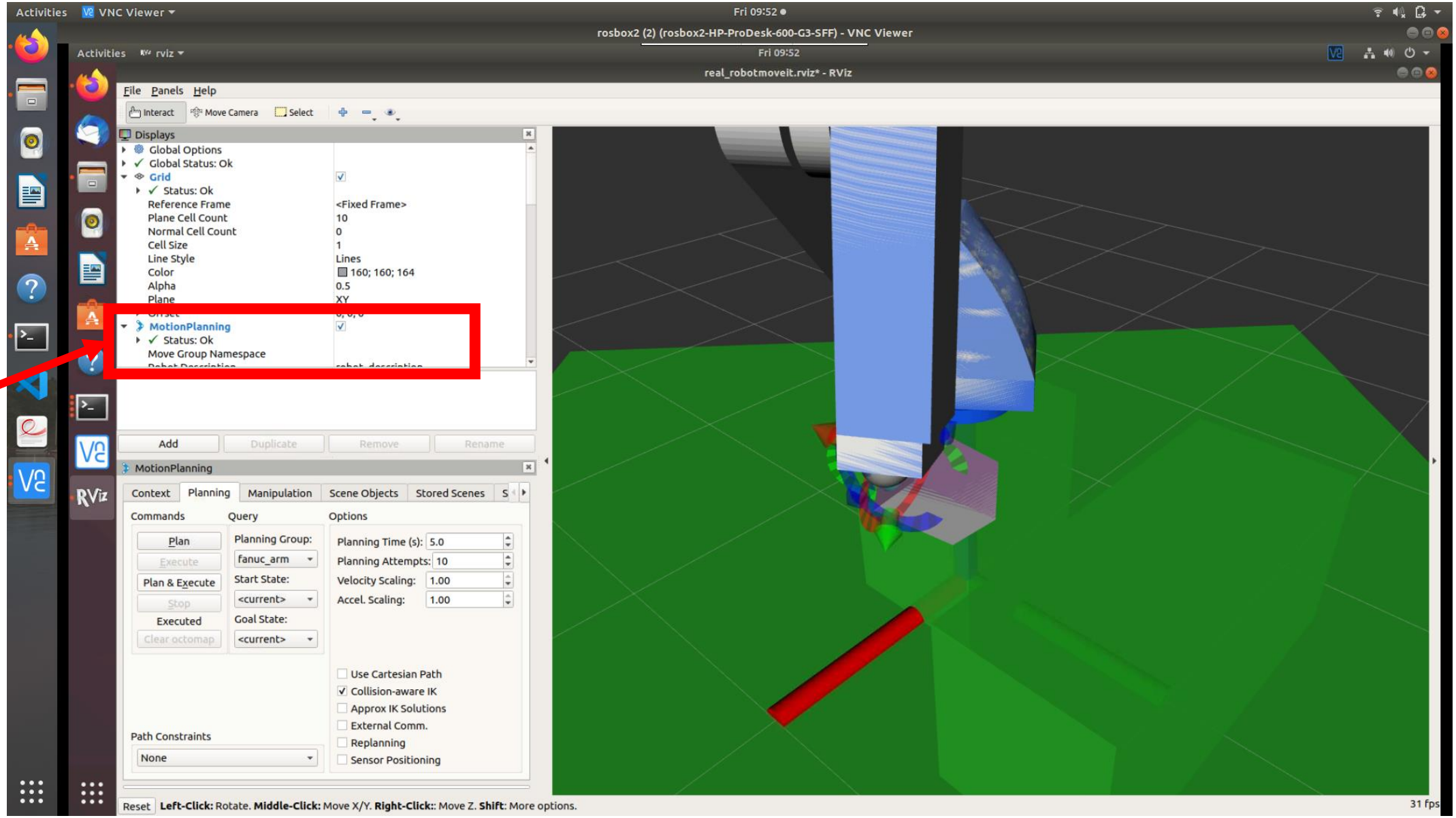
Failed motion execution
(In this case, because of the collision with the table that would happen if the motion took place.)



Controlling the Robot Via RVIZ

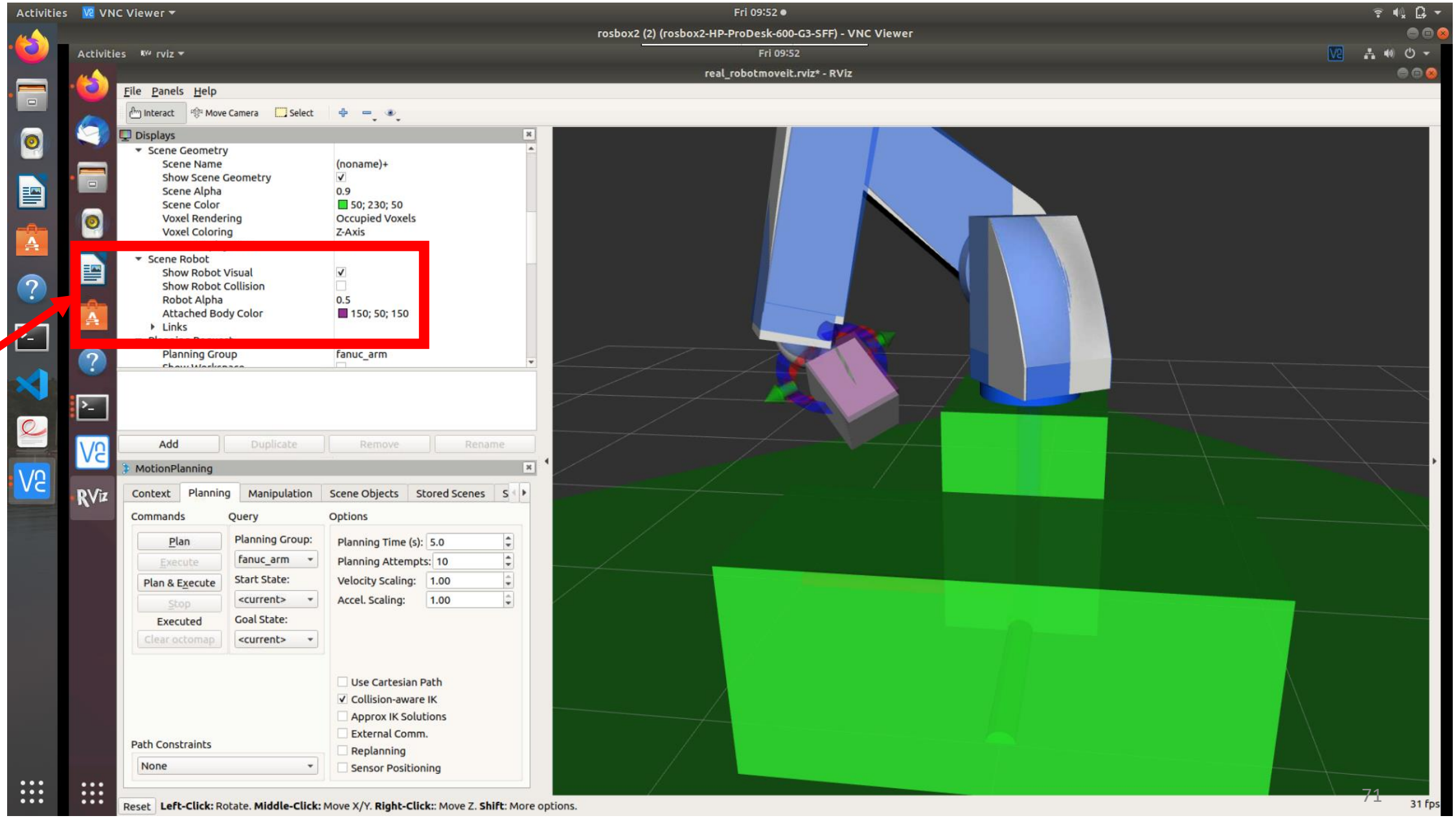
- To get the coordinate of the end-effector (used for cartesian planning), go to the Displays Window, scroll down to the Motion Planning tab until you reach ‘Scene Robot’, click the drop down arrow on ‘Scene Robot’ , click the drop down arrow on ‘Links’, click the drop down arrow on ‘tool 0’, and the position and orientation coordinates are now visible.

To get the coordinate of the end-effector (used for cartesian planning), go to the Displays Window, scroll down to the Motion Planning tab until you reach ‘Scene Robot’



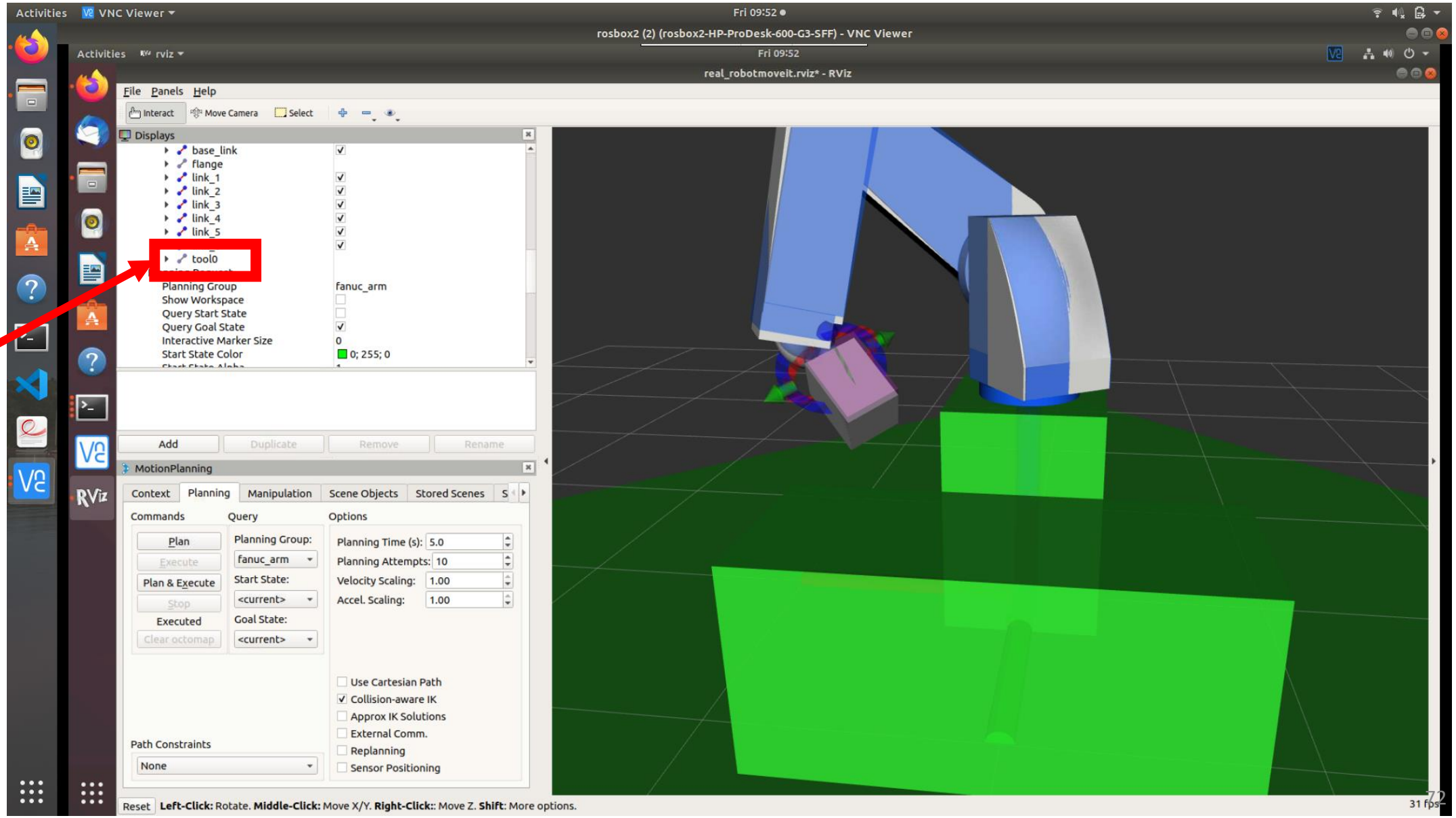
Scroll

Click the drop down arrow on ‘Scene Robot’ , click the drop down arrow on ‘Links’



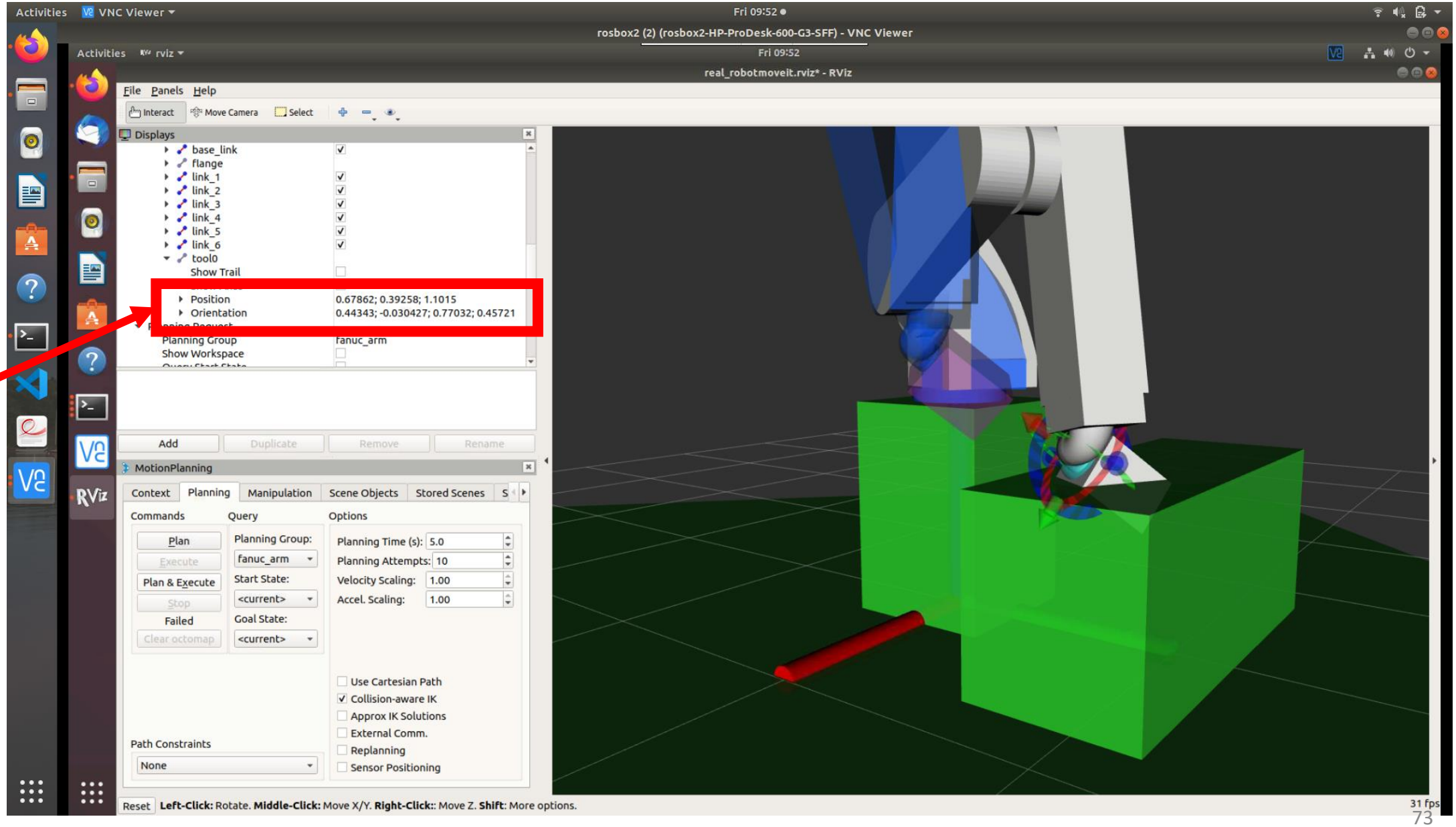
Click drop-down on ‘scene robot’ (if not already expanded) and ‘links’

Click the drop down arrow on ‘tool 0’ (the end-effector)



End Effector Link

The position and orientation coordinates are now visible.



Top row:
x, y z,
position in
that order
Bottom
row:
x, y, z, w
orientation
in that order

Writing Code to Control the Robot

This is the primary method we will be using to control the robot and will be used for the first Lab assignment.

How we program the robot

- In our program setup, we do not communicate directly with the robot itself through ROS. Instead, we communicate through ROS to the MoveIt node we are running, which communicates to the ROS/Fanuc Driver. We send MoveIt the commands to execute, and MoveIt transforms those commands into motions the robot can actually execute.

- Because of this, in the python programs we use to control the robot via ROS, we will need an instance of a class called FanucInterface. Fanuc Interface is a class that nicely packages all of the MoveIt objects we will access.

The FanucInterface Class

- The FanucInterface class is defined in a file called controller.py. All python scripts wanting to move the robot should import this file.
- The FanucInterface class, when initialized, starts the moveit_commander node. It also creates three other objects that are part of moveit_commander, the RobotCommander (assigned to class variable robot), the PlanningSceneInterface (assigned to class variable scene), and the MoveGroupCommander (assigned to the class variable move_group). It also starts a publisher for trajectory, which allows the display of movements into RVIZ

The FanucInterface Class

- In general:
- self.robot is the object that will be acted on to get robot state information when programming
- self.move_group is the object that will be acted on to actually move the robot
- self.scene is the object that will allow us to change the robot's planning scene (adding objects, etc).

Using the FanucInterface Class

- You will generally be using the high-level functions in the FanucInterface class rather than directly using variables like move_group and scene.
- This high-level functions and instantiation of the FanucInterface Class object are discussed next.

Using the FanucInterface Class

- To create a FanucInterface class object in your python script, assign the class to a variable like below. A good variable name, for instance, would be “fanuc”. Make sure you have controller.py imported.

- Import controller
- Fanuc = controller.FanucInterface()

Setting up the Robot Scene

- ALWAYS make sure you setup the robot scene before moving the robot. This will make sure you don’t accidentally karate chop the table in half or break the hand (speaking from experience here).

- fanuc.setupScene()

- If the hand is not attached to the robot, you can pass the function the argument: hand=False

Moving the Robot

- To actually move the robot, you can use joint coordinates or cartesian coordinates. For sophisticated robot action, cartesian coordinates will generally be used. Demo programs like the dancing program may use joint coordinates.

Moving the Robot – Joint Coordinates

- Joint coordinate moves use the class function go_to_joint_coordinates. This function takes in a list of six values for each of the six joints, where the joint position represented in radians. For instance, to zero out the robot:

- joint_goal = [0, 0, 0, 0, 0, 0]
- fanuc.go_to_joint_state(joint_goal)

Moving the Robot – Cartesian Coordinates

- Cartesian coordinate moves use the class function go_to_pose_goal. This function takes in a list of 3, 6, or 7 points.
- If 3 points, these will correspond to the x, y, and z position of the last joint (end-effector) of the robot in cartesian space
- If 6 points, the last three points will correspond to the x, y, and z orientation of the end-effector
- This function may change a bit in the future because I am not totally thrilled with how it is setup. It may be more useful to send it x,y,z, and w points only.

- pose_goal = [-0.5, 0.5, 1.2]
- fanuc.go_to_pose_goal(pose_goal)

- Note: Some of the points may be impossible to reach given a certain orientation of the robot. I recommend you figure out this points using rviz first before trying them out to make sure they are feasible.

Cartesian Space of the Robot

- The origin of the robot cartesian space is directly below and in the center of the base of the robot.
- When facing the robot from outside the cage, the positive x axis goes to the left, the positive y axis goes toward you, and the positive z axis goes straight up.
- Cartesian units are in meters.
- The robot's table corner points are (with a buffer of 10 cm): [-0.765, 0.6075, 0.39], [-0.765, 1.2225, 0.39], [0.765, 1.2225, 0.39], [0.765, 0.6075, 0.39]

Description of the Robot Controller File

- The robot controller file provides an neater interface to the functionality of the moveitcommander objects that we use to talk to the robot via ROS. Ideally, everyone will be programming with the same controller file. However, this file is by no means perfect and will likely be changed to add functionality as we go on. There are functionalities of moveit we aren't yet using and functionalities that could be better organized. With that in mind, the following is a description of some of the more important robot controller file functions.

Description of the Robot Controller File

- go_to_joint_state(joint_goal) : Takes in a list of six joint coordinates, and has the robot move to that location
- zeroOut(): Moves the robot to the 0 location for it's joints
- randomJointMove(): moves the robot to a random joint position
- printPt(self, wpose, message): Prints the point from the pose passed in with the user-defined message
- go_to_pose_goal(points): Takes a list of 3, 6, or 7 points (position, position + roll, pitch, yaw, or position + orientation in quarternion form) and moves the robot to these points in cartesian coordinates

Description of the Robot Controller File

- positionAnyOrientation(pos): takes a list of x, y, z points and moves the robot to that point without caring about orientation
- orientationAnyPosition(ori): takes a list of x, y, z, w quarternion points and moves the robot to that orientation without caring about position
- rpyOrientationAnyPosition(ori): takes a list of roll, pitch, yaw points and moves the robot to that orientation without caring about position
- goToRandomPose(): moves the robot to a random cartesian position

Description of the Robot Controller File

- setPositionTolerance(tolerance): set the tolerance for the robot's position accuracy
- setOrientationTolerance(tolerance): set the tolerance for the robot's orientation accuracy

Possible Sample Test Code File

```
Import controller
#Make the controller object
Fanuc = controller.FanucInterface()
#Set up the scene – very important!!
Fanuc.setupScene()
#ZeroOut Position
Fanuc.zeroOut()
#Move it to a cartesian coordinate (this is just an examples– you should
#find a valid coordinate using RVIZ first)
Fanuc.go_to_pose_goal([0.2, 0.3, 1.2])
```

Running your FANUC ROS scripts

- Run your python scripts through ROS using the normal commands. However, make sure you are running the scripts in the correct workspace. Also, ALWAYS make sure the scene objects are present before running a script. If you must restart the robot state program, make sure all robot python scripts are STOPPED. IMPORTANT: do NOT have a python robot control script running while restarting the robot state program! This will send commands to the robot without the scene objects loaded and cause crashes
- Make sure your python scripts are executed in the corresponding namespace (two underscores + ns:= + namespace name
- >> rosrun fanuc_demo cartMove.py __ns:=sim_1
- >> roslaunch fanuc_demo follow.launch __ns:=sim_2

Lab 1—Getting Started

- For lab 1, you will:
- A. Write a program in python and ROS to move the robot in a square pattern above (and not too close to!) the table. You will use cartesian coordinates
- B. Execute the program on the simulator
- C. Start the robot
- D. Start the ROS state program on the robot
- E. Start the ROS state program on a controller computer
- F. Execute your program on the real robot.

Lab 1-Hints

- The controller file has a lot of handy functions, including one to move the robot in cartesian coordinates
- Try to figure out the four or more points for your square pattern for the robot in RVIZ before writing the program. Make sure they are valid points the robot can actually move to.
- The orientation of the end effector may give you trouble. There are a few fixes: 1) Specify the orientation of the end effector when controlling the robot, or 2) set the tolerance for the end effector to be much less strict, or 3) use the function that moves the robot to the point using any orientation of the end effector.

Lab 1-Hints

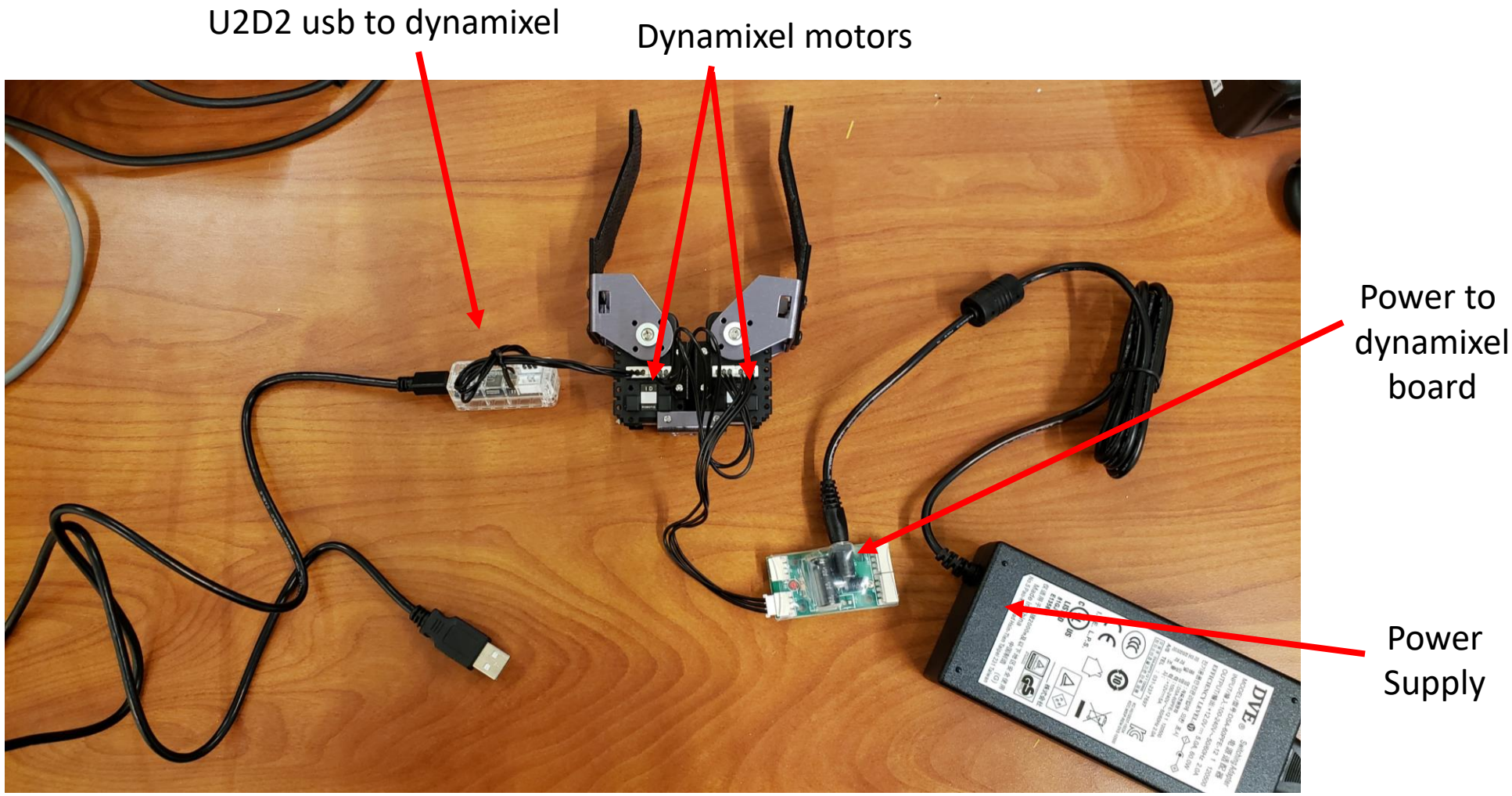
- Make sure you use the simulator before the actual robot!
- Make sure in your testing environment, the scene has been setup so you don’t accidentally hit the table, robot base, or hand.
- If you get stuck, contact Marz — slack channel or ever6812@vandals.uidaho.edu

Gripper

- The gripper is composed of two dynamixel AX-18F (https://emanual.robotis.com/docs/en/dxl/ax/ax-18a/) model motors. It runs on a 12V power supply and is controlled via serial USB interface to a computer (like a raspberry pi). The controller is a U2D2 device: https://emanual.robotis.com/docs/en/parts/interface/u2d2/

- The gripper is also controlled via ROS. The control file uses the dynamixel SDK for ROS, github here: https://github.com/ROBOTIS-GIT/DynamixelSDK

Gripper


U2D2 usb to dynamixel
Dynamixel motors
Power to dynamixel board
Power Supply

Gripper

- To control the gripper, you will use the dynamixel_controller.py file found in fanuc_demo/src directory. Import this into your program.
- You will need to make an instance of the dynamixelMotorControl class
- #initialize motor model AX-18F with a baudrate of 1000000 on port USB0 using packet protocoal 1.0
- Gripper = dynamixelMotorControl(“AX-18F”, 1000000, ‘/dev/ttyUSB0’, 1.0)

Gripper

- Use the “syncWrite” command to control the motors
- Pass in the list of motor IDs, the destination you are writing too, and a list of the commands in motor ID order
- “Goal_Position” Destination changes the motor location
- “Moving_Speed” changes the speed

```
gripper.syncWrite([1, 2], “Goal_Position”, [484,558])
```

- When you finished, end the gripper object

```
gripper.end()
```

Hand Gripper

- Coming Soon
- Nikolai, this is all you

Demo Programs

- Two demo programs we run on the robot include:
- Follow Me Program (Have the robot follow you with his head around the lab)
- Disco Dancing Program (Have the robot dance to a song)

Follow Me Program

- You will need OpenCV installed in order to make use of the FollowMe program.

Follow Me Program

- To launch the followMe Program:
- VNC into ROSBOX2 (cage computer)
- Once there, go to the roboLaunch folder and launch

  >> bash real_robot.sim
- WAIT UNTIL THE SCENE IS SETUP
- Also, start the cartesian coordinate receiver node with the command

  >> rosun fanuc_demo cartMove.py __ns:=real_robot

Follow Me Program

- On your control computer, type:

  >> rosrun fanuc_demo followMe.py __ns:=real_robot

- And (using your computer's webcam) position it somewhere where you can be 'seen' by the robot

Dancing Program

- The dancing demo program is composed of 5 Nodes:
  - audioNode.py (where song is played and beats detected)
  - audioAnalysis.py (beats are analyzed for frequency, commands sent)
  - dmxNode.py (commands received, passed to DMX light)
  - robotNode.py (commands received, passed to robot)
  - robotHandNode.py (functionality coming soon)
- And 2 libraries
  - controller.py (should be identical to main controller.py file, moves robot)
  - movesLibrary.py (different types of dance moves)

Dancing Program

- To run the dancing program, you will need the following python packages/libraries installed:
- aubio
- Pyaudio (may want to install as sudo apt-get install python-pyaudio rather than pip install)
- numpy
- wave
- pydub
- ola
- While these are mostly simple installs, the ola package setup takes more work. See the following section.

Dancing Program-DMX Setup

- To use the DMX node, you will need to download the OLA (Open Lighting Architecture) software and begin the ola daemon. Instructions for this are found here: https://www.openlighting.org/ola/getting-started/
- You will need to setup the architecture for the Velleman controller device. Instructions are listed below from this webpage: https://www.openlighting.org/ola/getting-started/device-specific-configuration/

Linux
You need a udev rule like this in /etc/udev/rules.d/10-local.rules

```
# udev rules file for the velleman dmx device SUBSYSTEM=="usb|usb_device", ACTION=="add", ATTRS{idVendor}=="10cf", ATTRS{idProduct}=="8062", GROUP="plugdev"
```

Then make sure the user olad runs as is a member of plugdev.

Dancing Program-DMX

- Make sure patch your device to a DMX universe (probably universe 1): https://www.openlighting.org/ola/getting-started/using-ola/
- Now your device should be setup. The DMX node uses the python ola interface, more information here: https://wiki.openlighting.org/index.php/OLA_Tips_%26_Tricks

- NOTE: Due to a port issue, it is usually not feasible to have the OLA daemon and the Dynamixel protocol running at the same time.

Dancing Program-DMX

- To connect the DMX light: plug the DMX cable into light. (Light should be configured as d001 or loop – I'm not actually sure which way is right-side up.
- Plug other end of cable into velleman controller. Plug USB end of velleman controller into computer.
- This should already be set up in rosbox2.

(Terrible picture quality)
Velleman DMX controller

Dancing Program

- To launch the dancing program:
  - Start the robot state program on ROSBOX2 (cage computer)

    >>bash real_robot.sh
  - WAIT UNTIL THE SCENE IS SETUP
  - On the computer you are using, run the command:

    >> roslaunch fanuc_demo disco.launch __ns:=real_robot
  - (Note: This will not start the robotHandNode as this hasn’t been integrated yet)

Robot URDF/meshes

- The robot urdf/meshes define how the robot looks in RVIZ and aid in simulation and planning. The meshes in the urdf have been measured to track with the real robot's measurements in order to provide the most accurate representation and to avoid collisions with the robot and its environment. The urdf/meshes are found in the fanucp50ib_support package.

Testing/Troubleshooting/Other

- There are two red emergency stop buttons on the robot system. One is on the teaching pendant, and I believe only works if the teaching pendant is on. The other is on the controller panel and will shut down the robot immediately. If you want the robot to move or the teaching pendant to operate, these button should be pushed out.

Testing/Troubleshooting/Other

- If the emergency stop buttons were pressed, you will have to restart all programs: first the ROS program on the fanuc robot, then the ROS state robot program on the control computer, then your own program. This is usually the case for any kind of fault or stopping of the physical robot ROS program.