Safety measures need to be followed before implementing any program to UR5:

- Do not directly upload any program to the robot. You need to first check the input (joint speed in case of ur_modern_driver) when you are uploading a code for the first time. First add a line to your code which gives the zero input and print them just one line before. Then run the code, check whether the inputs are in the range or not, then comment the line which was giving the zero input and upload the code again.

```
des_joint_vel = calc_q_dot(J,M,K_alpha,alpha_0,alpha)
print(t_now, des_joint_vel)
des_joint_vel = [0,0,0,0,0,0]
```

In the above example the final joint velocity to the robot is given by the variable “des_joint_vel”. So while testing the code comment the actual input and feed the zero values. Print the actual joint velocities just before that line and run the code and check whether the joint velocities are in the range (for joint speed the range is 0.3 rad/s).
If all the input velocities are in the range then comment the zero input line and feed the actual input values.

```
des_joint_vel = calc_q_dot(J,M,K_alpha,alpha_0,alpha)
print(t_now, des_joint_vel)
# des_joint_vel = [0,0,0,0,0,0]
```

- While testing a new code, always put one hand on the emergency stop button and keep an eye on the robot’s movement.
- Do not forget to scale the velocity in the teach pendant to 50% because at 100% velocity scale the movement of the robot is jerky. To compensate for this scaling we have multiplied the input velocities in all the code developed for this particular project by a factor of 2.

Prerequisites/Approach to proceed with document:

- One should have a basic understanding of a UR5 robot such as number of DoF, joint limit, workspace, payload, TCP configuration etc. Start with controlling the robot by its teach pendant. Refer to the following module available on Universal Robot's website: CB3 e-Learning
- Refer to the following link for installation and learning ROS:
  ROS Wiki
- For programming in ROS one needs to write the programs in either python or C++. It is recommended to write programs in python because you can find lots of readily available python libraries online. To learn the basic to advance python you can refer to any course available online. The link of one such free course is mentioned below:
  Beginner Friendly Full Python Tutorials(Teaser) | Python Tutorials For Absolute Beginners In Hindi#0

Chapter 1: Working with ROS

3.1 Getting started with ROS:

ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

ROS currently runs only on Unix-based platforms. Software for ROS is primarily tested on Ubuntu and Mac OS X systems, though the ROS community has been contributing support for Fedora, Gentoo, Arch Linux and other Linux platforms. However, If someone wants to use ROS using Windows 10 machine without installing Ubuntu OS, You can install Oracle VM virtual box and link the image of Ubuntu 18.04 LTS (Bionic) OS and install ROS Melodic. Refer 1.1 for installation of VM virtual box and linking of Ubuntu image.

3.2 Installation of Oracle VM Virtual box and Ubuntu Image on Windows 10

Here you will be provided a procedure to install this software on Windows 10 and run Ubuntu Image using the same.

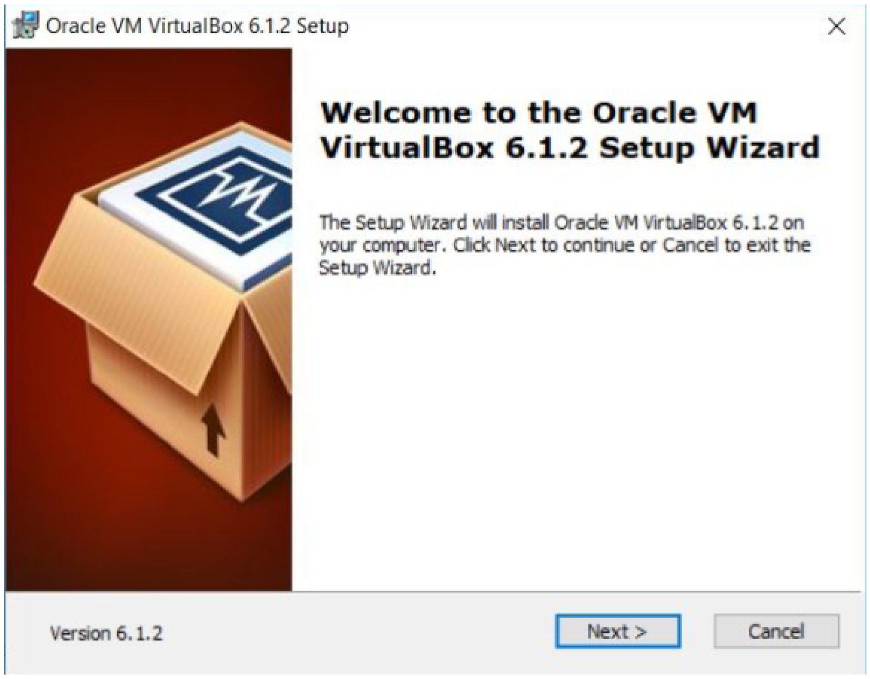
Dual booting the Windows 10

1. How to Dual boot Windows 10 and Ubuntu 18.04 (Complete Tutorial)
2. Follow the instructions provided in the above video for installation of Ubuntu 18.04 on Windows 10 operating system
3. After installation of Ubuntu 18.04, You can install the ROS by using the link provided below.
4. Ubuntu install of ROS Melodic (Install the ROS Melodic from the link)
5. You can also follow the installation instructions provided in the below video
6. How to Install ROS Melodic in Ubuntu | Complete steps shown in unix terminal co…

Installation procedure for Oracle VM Virtual Box

1. Download the Oracle VM Virtual Box from click here or from the link. (Downloads – Oracle VM VirtualBox)

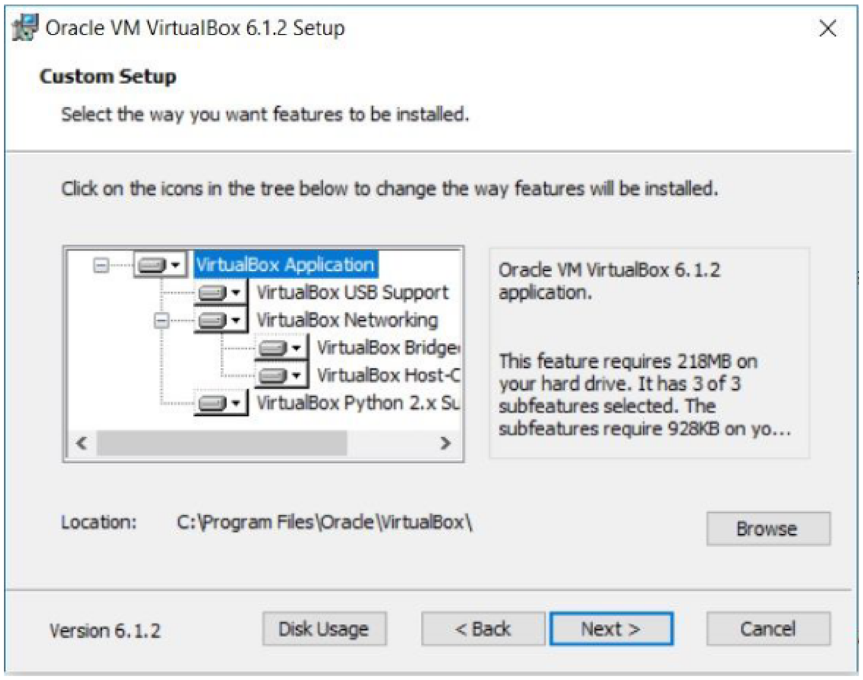
2. The file downloaded will have the file name format like VirtualBox-VersionNumber-BuildNumber-Win.exe. For example VirtualBox-6.1.4-136177-Win.exe. Double click on the installer to launch the setup wizard. Click on Next to continue.



Welcome to the Oracle VM VirtualBox 6.1.2 Setup Wizard

The Setup Wizard will install Oracle VM VirtualBox 6.1.2 on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

3. You will see a custom setup dialog box. There is not much to choose from. You can accept the default and click next. If you wish to change the installation directory, you can change it by clicking on the browse button and selecting the new directory and clicking OK. Normally you may leave it as the default as the whole installation process does not take much space on your hard drive.



Custom Setup

Select the way you want features to be installed.

Click on the icons in the tree below to change the way features will be installed.

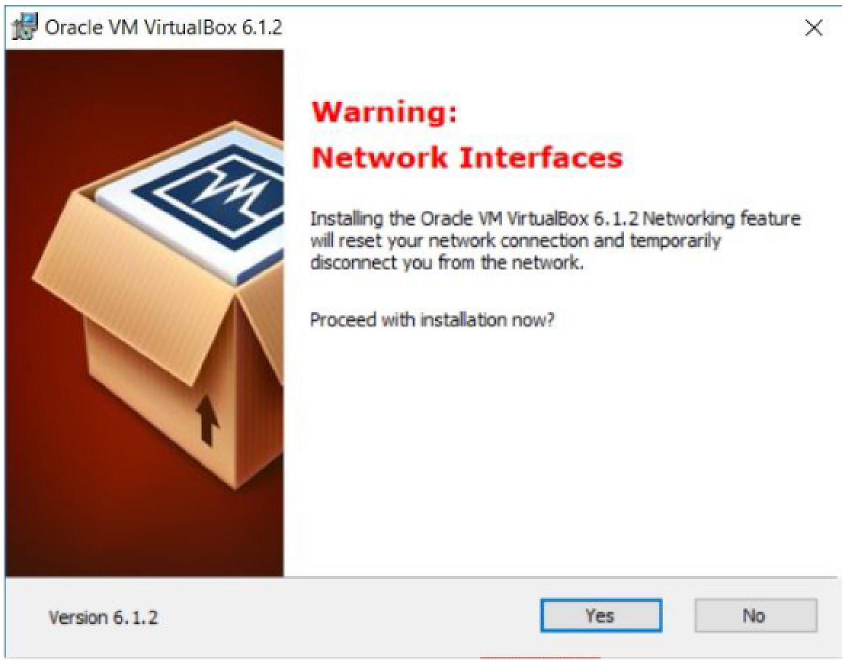
- VirtualBox Application
  - VirtualBox USB Support
  - VirtualBox Networking
    - VirtualBox Bridge
    - VirtualBox Host-C
  - VirtualBox Python 2.x Su

Oracle VM VirtualBox 6.1.2 application.

This feature requires 218MB on your hard drive. It has 3 of 3 subfeatures selected. The subfeatures require 928KB on yo...

Location: C:\Program Files\Oracle\VirtualBox\

4. This dialog box warns you about setting up a Network Interface. This means that VirtualBox will install network interfaces that will interact with the installed virtual machines and the host operating system which in our case is windows. This will temporarily disconnect you from the internet but that is OK, nothing to worry.



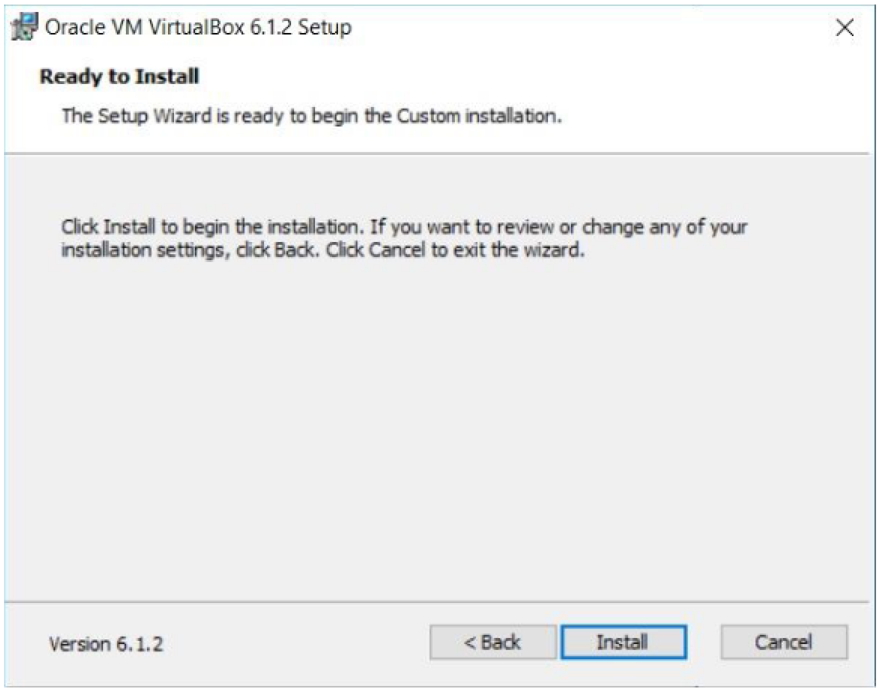
Warning:
Network Interfaces

Installing the Oracle VM VirtualBox 6.1.2 Networking feature will reset your network connection and temporarily disconnect you from the network.

Proceed with installation now?

Version 6.1.2

5. You will see ready to install a dialog box. Click Install to continue.



Ready to Install

The Setup Wizard is ready to begin the Custom installation.

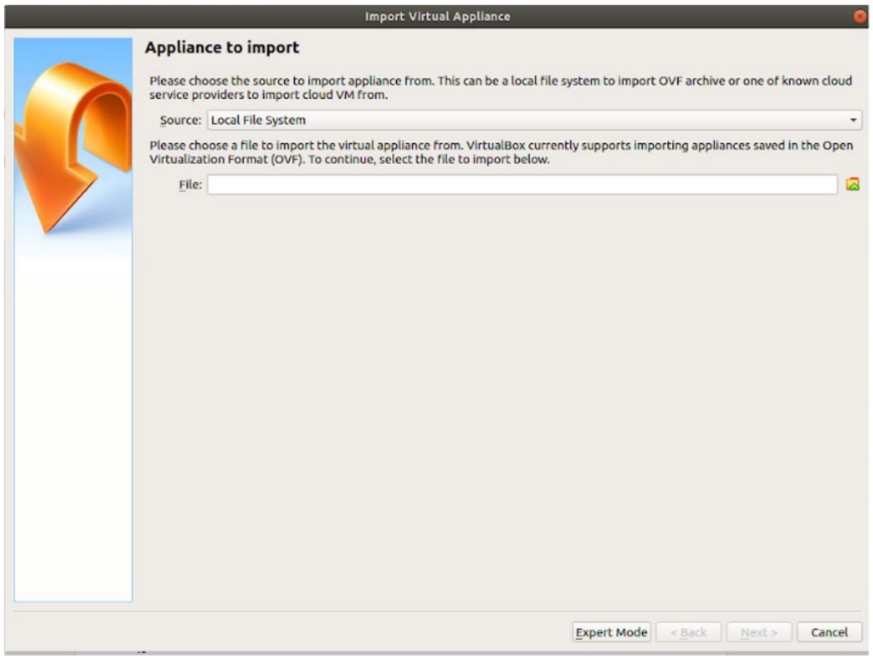
Click Install to begin the installation. If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.

Version 6.1.2

6. After installation, launch the application.

Import the Ubuntu Image in Oracle VM virtualbox

1. Download Ubuntu 18.04 LTS (Bionic) OS Image with pre installed ROS Melodic from here: https://drive.google.com/file/d/1SOYZSb0Od4PvqqrhlRA48Mv3pNMqVw2E/view?usp=sharing
2. After launching the application, go to “File” and click on “import appliance”.



Appliance to import

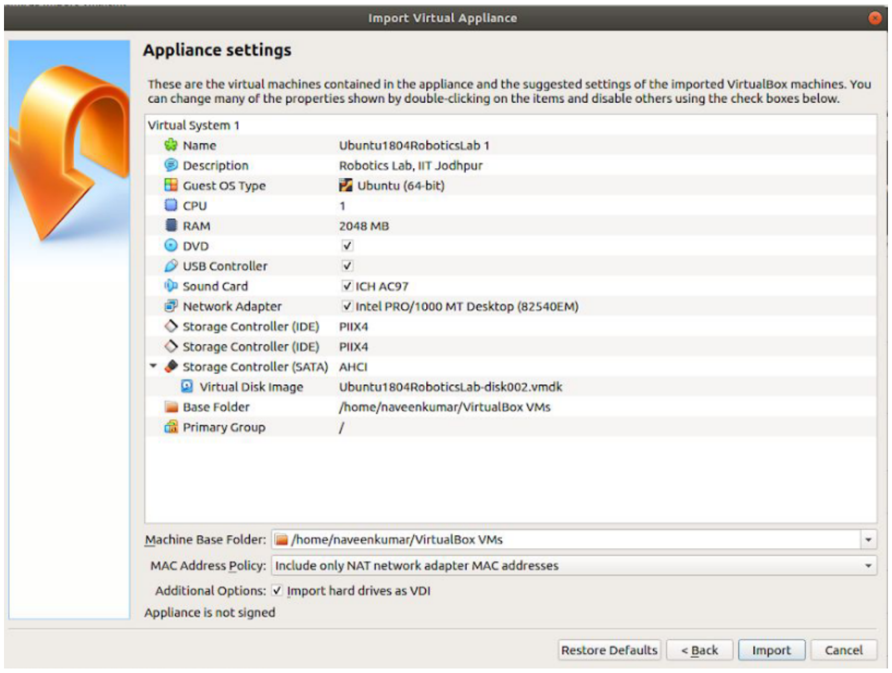
Please choose the source to import appliance from. This can be a local file system to import OVF archive or one of known cloud service providers to import cloud VM from.

Source: Local File System

Please choose a file to import the virtual appliance from. VirtualBox currently supports importing appliances saved in the Open Virtualization Format (OVF). To continue, select the file to import below.

File:

3. Choose the path of the file with its name by clicking on the file manager icon which has an arrow sign on it.
4. Click on next and then on the next window click on import.


Import Virtual Appliance

Appliance settings

These are the virtual machines contained in the appliance and the suggested settings of the imported VirtualBox machines. You can change many of the properties shown by double-clicking on the items and disable others using the check boxes below.

Virtual System 1

Name	Ubuntu1804RoboticsLab 1
Description	Robotics Lab, IIT Jodhpur
Guest OS Type	Ubuntu (64-bit)
CPU	1
RAM	2048 MB
DVD	✓
USB Controller	✓
Sound Card	✓ ICH AC97
Network Adapter	✓ Intel PRO/1000 MT Desktop (82540EM)
Storage Controller (IDE)	PIIX4
Storage Controller (IDE)	PIIX4
Storage Controller (SATA)	AHCI
Virtual Disk Image	Ubuntu1804RoboticsLab-disk002.vmdk
Base Folder	/home/naveenkumar/VirtualBox VMs
Primary Group	/

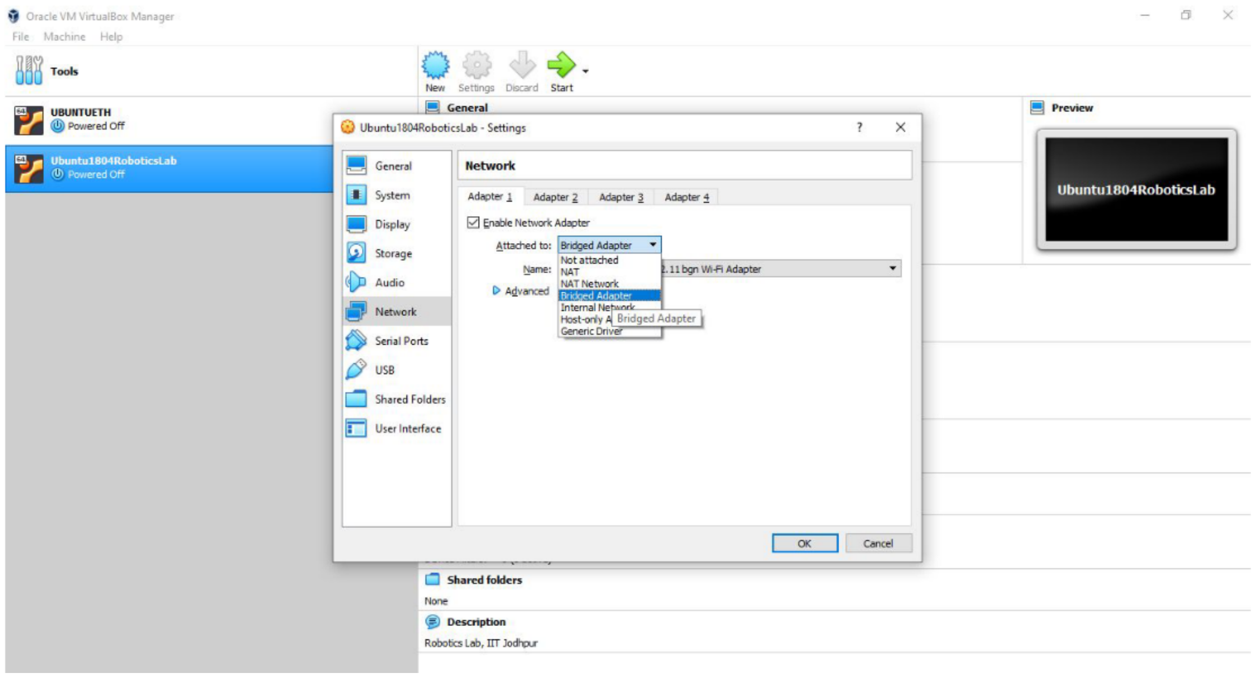
Machine Base Folder: /home/naveenkumar/VirtualBox VMs

MAC Address Policy: Include only NAT network adapter MAC addresses

Additional Options: ✓ Import hard drives as VDI

Appliance is not signed

5. Before you launch the virtual machine, you have to change the network adapter settings to make it compatible with your network for communication with your Android Mobile device.
6. Click on settings and go to Network options.



Network

✓ Enable Network Adapter

Attached to: Bridged Adapter

Name: 2.11 bgn Wi-Fi Adapter

Shared folders

None

Description

Robotics Lab, IIT Jodhpur

7. Change the settings from “NAT” to “Bridged Adapter”

8. Click on OK and launch the machine by clicking on the start button.

Troubleshooting:
1. The USB device doesn't mount automatically in the case of VirtualBox, install the VirtualBox extensions package for using the USB port.
2. Set up BIOS settings properly because some PCs don't allow virtual machines. For that you have to enable virtualization in the BIOS menu for that you have to check the documentation of your PC and Laptop.
3. You might have to disable secure boot if you are loading the virtual machine on a linux platform. Disabling secure boot is different according to your PC or Laptop. So, for that, you have to look at the documentation of your PC.

3.3 Installing ROS Melodic in Ubuntu OS:

1. Configure your Ubuntu repositories to allow "restricted," "universe," and "multiverse."
2. Enter the following command in the terminal to set up your computer to accept software from packages.ros.org.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" >
/etc/apt/sources.list.d/ros-latest.list'
```

3. Set up your keys:

```
sudo apt install curl # if you haven't already installed curl
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
```

4. Update your Debian package index using the following command:

```
sudo apt update
```

5. Enter the following command to install ROS, rqt, rviz, robot-generic libraries, 2D/3D simulators, and 2D/3D perception.

```
sudo apt install ros-melodic-desktop-full
```

6. It's convenient if the ROS environment variables are automatically added to your bash session every time a new shell is launched:

```
echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

7. To install this tool and other dependencies for building ROS packages, run:

```
sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-wstool
build-essential
```

8. Install and initialize rosdep:

```
sudo apt install python-rosdep
sudo rosdep init
rosdep update
```

9. If you just installed ROS from apt on Ubuntu then you will have a setup.*sh files in '/opt/ros/<distro>/', and you could source them like so:

```
$ source /opt/ros/melodic/setup.bash
```

10. Create a ROS workspace:

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
```

11. Source new setup.*sh file:

```
$ source devel/setup.bash
```

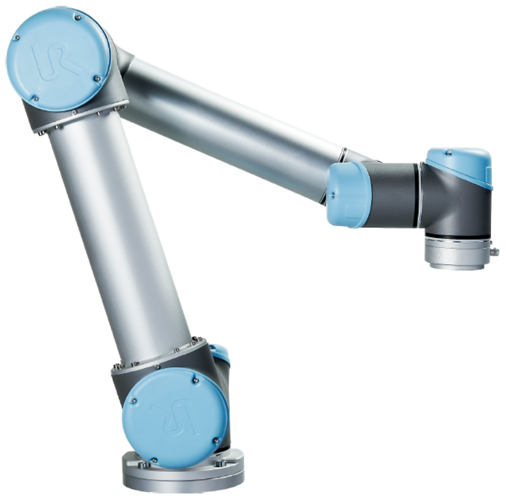
12. To make sure your workspace is properly overlayed by the setup script, make sure ROS_PACKAGE_PATH environment variable includes the directory you're in.

```
$ echo $ROS_PACKAGE_PATH
/home/youruser/catkin_ws/src:/opt/ros/melodic/share
```

Chapter 2: Introduction to UR5 (CB3)

Universal Robot CB3 tutorials
https://academy.universal-robots.com/free-e-learning/cb3-e-learning/

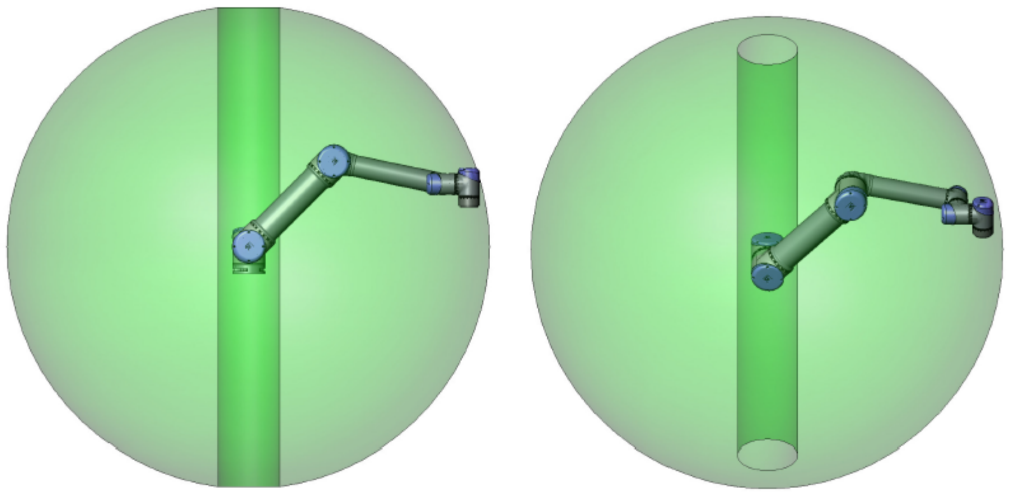
UR5 is a medium-sized, collaborative robotic arm developed by Universal Robots. It is ideal for automating low-weight processing tasks with its 5kg payload and 850mm reach radius. The robot consists essentially of six robot joints and two aluminum tubes, connecting the base with the tool of the robot. The robot permits the tool to be translated and rotated within the workspace.
ros



Technical Specifications:
Some of the specifications of the robot is enlisted in the table below:

Robot type	UR5
Weight	18.4 kg / 40.6 lb
Payload	5 kg / 11 lb
Reach	850 mm / 33.5 in
Joint ranges	$\pm$ 360° on all joints
Speed	Joint: Max 180 °/s. Tool: Approx. 1 m/s / Approx. 39.4 in/s.
Repeatability	$\pm$ 0.1 mm / $\pm$ 0.0039 in (4 mils)
Footprint	Ø149 mm / 5.9 in
Degrees of freedom	6 rotating joints
Control box size $(W \times H \times D)$	475 mm $\times$ 423 mm $\times$ 268 mm / 18.7 in $\times$ 16.7 in $\times$ 10.6 in
I/O ports	18 digital in, 18 digital out, 4 analogue in, 2 analogue out
I/O power supply	24 V 2 A in control box and 12 V/24 V 600 mA in tool
Communication	TCP/IP 100 Mbit: IEEE 802.3u, 100BASE-TX Ethernet socket & Modbus TCP
Programming	PolyScope graphical user interface on 12” touchscreen with mounting
Noise	Comparatively noiseless
IP classification	IP54
Power consumption	Approx. 200 W using a typical program
Collaboration operation	Collaborative operation according to ISO 10218-1:2011
Temperature	The robot can work in a temperature range of 0-50 °C
Power supply	100-240 VAC, 50-60 Hz
Calculated operating life	35,000 hours
Cabling	Cable between robot and control box (6 m / 236 in) Cable between touchscreen and control box (4.5 m / 177 in)

Workspace of the robot:
The workspace of the UR5 robot extends 850 mm from the base joint. It is important to consider the cylindrical volume directly above and directly below the robot base when a mounting place for the robot is chosen. Moving the tool close to the cylindrical volume should be avoided if possible, because it causes the joints to move fast even though the tool is moving slowly, causing the robot to work inefficiently and the conduction of the risk assessment to be difficult.



A control box with a teach pendant (a panel with the touch screen) is provided to move and manipulate the tool (End-effector).

Chapter 2: UR control using teach pendant

Chapter 3: UR control using remote access (ROS)

3.4 Installation of UR packages in ROS melodic:

1. Go to the website:
   Universal Robot
2. Open your terminal and type there:

```
sudo apt update
```

   In case you have any problem, copy the error message in the place it in google search bar and get the solutions.
3. Go to your workspace in src directory and enter the following code to download the package for universal robots:

```
$ cd ~/catkin_ws/src
git clone -b $DISTRO-devel https://github.com/ros-industrial/universal_robot.git
```

   Replace $DISTRO with the ROS version you are using such as Melodic or Kinetic.
4. Go the workspace (cd ..) and then enter the following command:

```
rosdep install --from-paths src --ignore-src --rosdistro $DISTRO
```

   Replace $DISTRO with the ROS version you are using such as Melodic or Kinetic.
5. In case there is any problem run the following code

```
rosdep update
```

   then repeat step 4.

“If both the code available is not working then first install rosdep then try these two code”

6. Now download the ur_modern_driver package from following the website:
   GitHub - ros-industrial/ur_modern_driver: (deprecated) ROS 1 driver for CB1 and CB2 controllers with UR5 or UR10 robots from Universal Robots

7. Extract the package and copy this folder into the universal_robot directory inside the src directory of your workspace.
8. Remove the ur_driver package from the src directory, your package contains ur_modern_driver package instead of ur_driver package.
9. So you have to check all package.xml files in the universal_robot packages, edit them and replace the words ur_driver by ur_modern_driver and save those package.xml files in each and every package.xml file.
10. The most important part:
    1) Go to the file name <<ur_hardware_interface.cpp>> which is present inside the src/universal_robot/ur_modern_driver/src directory.
    2) Open the file and you need to edit some parts.
    3) Inside the file replace "**controller_it->hardware_interface**" by "**controller_it->type**"

       This line begins at line number 180 of the file then you have to replace the "**controller_it->hardware_interface**" by "**controller_it->type**" in all the places it occurs in the file carefully and save the file.
    4) The above replace code is found as conditional statement with if

11. Now form your workspace you can build the package entering the code in your terminal:

```
$ catkin_make
```

12. Your package is ready and you can use it. In case of any error refer the following websites:
https://github.com/ThomasTimm/ur_modern_driver/blob/master/src/ur_hardware_interface.cpp
https://github.com/ThomasTimm/ur_modern_driver
Universal Robot
https://github.com/ros-industrial/universal_robot
https://github.com/ThomasTimm/ur_modern_driver/issues/58
https://github.com/iron-ox/ur_modern_driver/commit/883070d0b6c0c32b78bb1ca7155b8f3a1ead416c

3.5 Connecting to the hardware:

It has been assumed that you have installed packages required.
1. Connect your computer and UR5 manipulator via ethernet cable.
2. Use your teach pendant and Go To : SETUP Robot>>NETWORK
Now, select Static Address and fill the empty boxes with the values mentioned.

IP address	: 192.168.1.100
Subnet mask	: 255.255.255.0
Default gateway	: 0.0.0.0
Preferred DNS server	: 0.0.0.0
Alternate DNS server	: 0.0.0.0

3. Apply/Save the setting.
4. Now go to your PC's Network settings then Add connections
5. Choose a connection type : Ethernet
6. Give connection name : UR5(or something you want)
7. Click on drop down arrow in device and select the address
8. Go to IPv4 Setting tab
   Select Method : Manual
   Click on Add to add address of a device
   Fill in the boxes as Address, Netmask, Gateway as you have filled in the Network setting of the UR5.
9. Save the setting

Now you can connect to the UR5 manipulator.

3.6 Starting UR5:

It has been assumed that you have installed the universal_robot package in your workspace where ur_driver has been replaced by ur_modern_driver and necessary changes have been made.

1. Now go to the terminal and enter the following command:

```
roslaunch ur_modern_driver urXX_bringup.launch robot_ip:=ROBOT_IP_ADDRESS
```

Note here XX is either 5 or 10 depending upon what you have either UR10 or UR5 and IP address is your IP address. Now you are connected to the hardware.

2. If you do not have your ur_description installed please do so via

```
sudo apt install ros-<distro>-ur-description
```

Since now you are connected to the hardware you can use other nodes accordingly.

Ur_driver functionality:

- */joint_speed* : Takes messages of type *trajectory_msgs/JointTrajectory*. Parses the first JointTracetoryPoint and sends the specified joint speeds and accelerations to the robot. This interface is intended for doing visual servoing and other kinds of control that requires speed control rather than position control of the robot. Remember to set values for all 6 joints. Ignore the field joint_names, so set the values in the correct order.
- */ur_driver/URScript* : Takes messages of type *std_msgs/String* and directly forwards it to the robot. Note that no control is done on the input, so use at your own risk! Intended for sending movel/movej commands directly to the robot, conveyor tracking etc.
- Publishes robot joint state on */joint_states*
- *Publishes TCP force on /wrench*

- *Publishes IO state on /ur_driver/io_states (Note that the string /ur_driver has been prepended compared to the old driver)*
- *Action interface on /follow_joint_trajectory for seamless integration with MoveIt.*

Chapter 4: Various schemes developed for teledentistry operations

Teledentistry requires a robotic arm to be controlled using a haptic device so that force feedback from the patient’s teeth / any organ can be observed at the haptic interface of the operator (doctor / medical practitioner). If the force feedback is available from the robotic arm to the haptic device then a controller can be designed such that the operator can apply the required amount of force on the patient’s organ. This task is challenging compared to conventional systems in which the feedback and control is unidirectional. In this case feedback is taken both from the operator and the robot’s end-effector. A standard haptic device with 6 DOF (Degree of Freedom) gives position (x,y,z) and orientation (phi, th, psi) as the output along all the three orthogonal axes of cartesian coordinates. It also gives the feedback of forces (Fx, Fy, Fz) and torques (Mx, My, Mz) being applied on the linkages of the haptic devices by the operator.

Two devices (Manipulator arm and Haptic device) are main components of the teledentistry system. Integration and control between these two devices is the most challenging part of the whole task, as it requires interfacing as well as proper algorithms to complete the task and take care of the safety. Visual feedback, software interface for visualization of patient and environment etc are important as well but that can be done using standard devices with less effort that the former ones.

To begin with the work the task of teleoperation is split into several subtasks:

a) **Interfacing the robotic manipulator using a PC and developing separate software packages which contain various ROS nodes**

Interfacing the robotic manipulator includes UR package installation, connecting to the hardware etc. This section has already been explained in the previous chapter.

b) **Control of robotic arm using joint speed**

In this scheme one node (pub_vel) is created which subscribes on the topic */joint_states* and publishes the desired joint velocity on the topic /ur_driver/*joint_speed*. The desired joint velocity is randomly set in the program provided it should not exceed the maximum joint speed mentioned in the UR5 user manual. While conducting the experiment sudden jerks were observed when the input joint velocity is very high or the joint velocity publish rate is lower than 125 Hz (*/joint_state* update rate is 125 Hz). So in all the experiments maximum joint speed was set to 0.3 rad/s for all the joints. The topic */joint_states* in message type *trajectory_msgs/JointTrajectory* contains the current joint position, velocity and joint effort. In all the nodes mentioned in this document the joint velocity has been scaled up by a factor of 2. This is done to compensate for the velocity scaling of 50% in teach-pendant of UR5, which is set in order to avoid the jerks.

c) **Control of robotic arm using joint angle**

In this scheme the node (pub_ang) subscribes on the topic */joint_states* and publishes the desired joint velocity on the topic */joint_speed*. In this scheme, the user (programmer) randomly sets the desired joint angle provided it should be in the range of joint limit mentioned in the UR5 manual. The joint velocity from the desired joint angle and current joint angle is calculated as mentioned below.

$$q_{dot} = K_p(q_{curr} - q_{des})$$

The proportional constant Kp is set for each joint and tuned in experiment. It is ensured that the joint speed calculated from the above relation does not exceed the maximum limit mentioned in the UR5 manual.

d) Cartesian velocity control of robotic arm (teleoperation using joystick)

The ultimate aim of this project is to control the robotic arm using the haptic device, but as the haptic devices are very delicate and expensive, we can first use a low end device for teleoperation like a joystick. In this way we will not get the force feedback at the operator end but we can control the velocity of the robotic arm in the task space. Later we will extend the algorithm used in this section to controlling the robot using the haptic device.

i) UR5 interface with the ROS

(a) This step has already been explained in the previous section. Please refer to section

ii) Joystick interface with the ROS

(a) Start by installing the package. Open the terminal and enter following commands:

```
$ sudo apt install ros-<$DISTRO>-joy
```

Replace $DISTRO with the ROS version you are using such as Melodic or Kinetic.

(b) Connect the joystick to your computer and run the following command to see if Linux recognize your joystick:

```
$ ls /dev/input/
```

You will see a list of all the input devices as shown below:

```
by-id    event0  event2  event4  event6  event8  mouse0  mouse2  uinput
by-path  event1  event3  event5  event7  js0     mice    mouse1
```

From the above list it can be seen that the joystick devices are referred to by jsX. In this case it is referred to as js0.

(c) Check if the joystick is working by running the following command:

```
$ sudo jstest /dev/input/jsX
```

You will get the output of the joystick similar as shown below, depending on the type of joystick you are using. Move the joystick to see the data change.

```
Driver version is 2.1.0.
Joystick (Logitech Logitech Cordless RumblePad 2) has 6 axes (X, Y, Z, Rz, Hat0X,
Hat0Y)
and 12 buttons (BtnX, BtnY, BtnZ, BtnTL, BtnTR, BtnTL2, BtnTR2, BtnSelect, BtnStart,
BtnMode, BtnThumbL, BtnThumbR).
Testing ... (interrupt to exit)
Axes:  0:     0  1:     0  2:     0  3:     0  4:     0  5:     0 Buttons:  0:off  1:off  2:off  3:off  4:off
5:off  6:off  7:off  8:off  9:off 10:off 11:off
```

The joystick used in this experiment is Logitech Extreme 3D Pro (image shown below).

(d) Check the permission of the joystick to make it accessible for the ROS joy node.

```
$ ls -l /dev/input/jsX
```

You will get the output similar to shown below.

```
crw-rw-XX- 1 root dialout 188, 0 2009-08-14 12:04 /dev/input/jsX
```

If XX is rw: the joystick device is configured properly.

If XX is --: the joystick device is not configured properly and you need to:

```
$ sudo chmod a+rw /dev/input/jsX
```

(e) To get the joystick data published over ROS we need to start the joy node. First let's tell the joy node which joystick device to use. In this case it is js0.

```
$ roscore
$ rosparam set joy_node/dev "/dev/input/jsX"
```

Now, you can start the joy node.

```
$ rosrun joy joy_node
```

You will get the output similar to shown below.

```
[ INFO] 1253226189.805503000: Started node [/joy], pid [4672], bound on [aqy], xmlrpc
port [33367], tcpros port [58776], logging to [/u/mwise/ros/ros/log/joy_4672.log], using
[real] time

[ INFO] 1253226189.812270000: Joystick device: /dev/input/js0

[ INFO] 1253226189.812370000: Joystick deadzone: 2000
```

(f) Run the following command in a new terminal to see the data from the joystick.

```
$ rostopic echo joy
```

As you move the joystick around, you will get the output similar as shown below.

```
---
axes: (0.0, 0.0, 0.0, 0.0)
buttons: (0, 0, 0, 0, 0)
---
axes: (0.0, 0.0, 0.0, 0.12372203916311264)
buttons: (0, 0, 0, 0, 0)
---
axes: (0.0, 0.0, -0.18555253744125366, 0.12372203916311264)
buttons: (0, 0, 0, 0, 0)
---
axes: (0.0, 0.0, -0.18555253744125366, 0.34022033214569092)
buttons: (0, 0, 0, 0, 0)
---
axes: (0.0, 0.0, -0.36082032322883606, 0.34022033214569092)
buttons: (0, 0, 0, 0, 0)
```

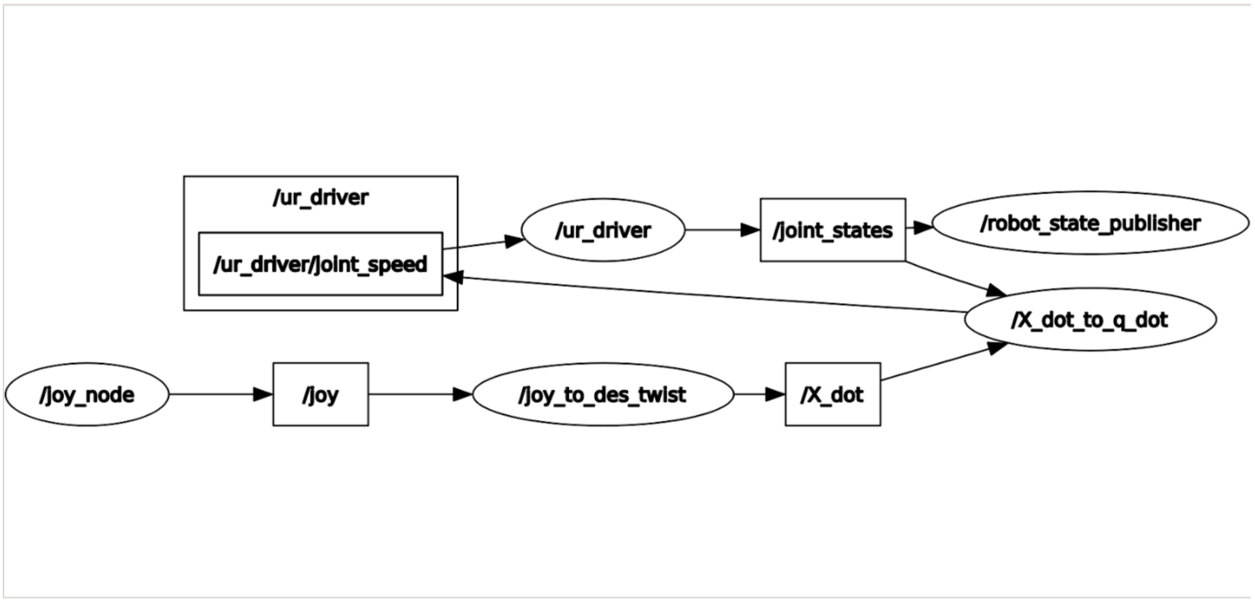
iii) Custom nodes for interfacing with UR5 and joystick

Our objective is to control the robot in the task space using a joystick. In other words, to give the position and orientation of the end-effector by joystick's output. There are a total six parameters (position in x, y, z and orientation about x, y, z that is phi, theta and psi) which we need to control for the task space control of the robot's end-effector. So we need to assign six channels (axes and/or buttons) for each parameter. The joystick used in this experiment has six axes and twelve buttons. Out of six only four axes give the continuous output while the other two give the discrete output. The button gives the output as either 0 or 1. For convenience three axes are assigned to control the orientation and a combination of a button and axes is assigned to control the position of the end-effector. The joy node, discussed in the previous section, will give the axes value varying from -1 to 1 and buttons value either 0 or 1.
Two custom nodes (node 2 and node 3) are created to get the joint speed from the joystick's output. The flowchart (rqt graph) is shown below:

Node 1: ***/joy_node***
Publication: */Joy* [*sensor_msgs*/*Joy*]

Node 2: ***/joy_to_des_twist***
Subscription: */Joy* [*sensor_msgs*/*Joy*]
publication: */X_dot* [*geometry_msgs*/*Twist*]

Node 3: ***/X_dot_to_q_dot***
Subscriptions: */X_dot* [*geometry_msgs*/*Twist*]
/Joint_states [*sensor_msgs*/*JointState*]
Publication: */ur_driver*/*joint_speed* [*trajectory_msgs*/*JointTrajectory*]



iv) **Limitation of current joystick and solution**
The joystick used for this experimental study has 6 axes and 12 buttons. Out of 6 only 4 axes give continuous output but 6 channels with continuous output are required for all 6 components of the end-effector velocity (3 linear velocity component and 3 angular velocity component). So, 3 axes with continuous output are selected for the 3 components of the angular velocity vector and a combination of a button with 3 axes is selected for the 3 components of the linear velocity vector. This way either linear or angular velocity can be controlled one at a time, both cannot be controlled simultaneously. To simultaneous control of linear and angular velocity 2 joysticks can be used one for position and the other one for the orientation of the end-effector.

v) **Limitations of the UR5 (joint angle and speed limit, singularity) and possible solution**

e) Jacobian trajectory tracking in cartesian space (Setting the home location of the dentistry tool (end effector of robotic arm):

For the teledentistry task the instrument in which the operator (doctor) is in direct contact with is a multi-degree of freedom haptic device and robotic manipulator will be near the subject (patient). The haptic device available for our task is Phantom Omni. It is like an advanced and sophisticated joystick having 6 DOF which can give feedback in motion along x, y, z position and roll, pitch, yaw angle. It also gives feedback of forces along all the axes. During the diagnosis/operation the operator can directly maneuver the robot arm using the motion of the haptic device and simultaneously experience the forces of the end-effector (tool) on the stylus. Experiments and control logic for motion control tasks without force feedback were performed using a 3 DOF joystick and UR5 robotic arm. But when the operator completes his task then he has to decide to put the robotic arm in a particular configuration. This configuration can be called as home location. Home location consists of predefined position and orientation of the robot. The position and orientation must be selected in such a way that the robot is away from the subject and other equipment and objects lying in the vicinity of the end effector but must be well within the workspace boundaries to avoid the singularities.

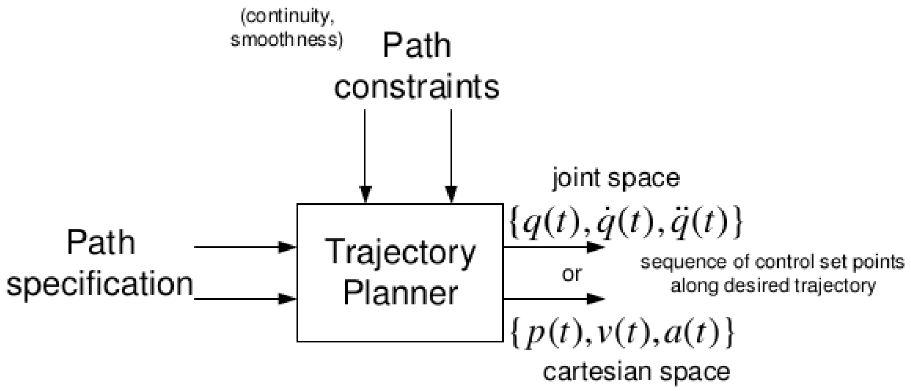
Once the home location (goal location) is provided the robotic arm must go to that home location in a controlled and safe manner. To perform this maneuver the robot must have a well known path that can be followed by its end effector. Path is a sequence of robot configurations without considering the time at which the configuration should be attained. Sequence from initial to final configurations must be in particular order in which the robot should move. Decision of obtaining this path is called **path planning**/**motion planning** / navigation problem / piano mover's problem. It is a kind of computational problem which can be solved by using some algorithms. These algorithms can be deterministic as well as probabilistic. Some of them use graph based search techniques as well. It finds a sequence of valid configuration from source to destination configuration. Inputs to the path planning algorithm are configuration space of the robot, obstacle space of the robot, initial and final configuration. Some of the popular methods for path planning are:

1. A*

2. Dijkstra
3. D*
4. Artificial potential field based path planning
5. RRT (Rapidly-exploring Random tree)
6. Genetic algorithm
7. Ant Colony algorithm
8. grassfire algorithm

When the knowledge of the environment is available using RGB and depth cameras these methods can be implemented. All these algorithms have different time and computational complexity but can be used when there are obstacles present within the workspace. In our experiments as there were no obstacles in the vicinity of the end-effector and home location, a straight line path joining the source to the destination was pre-assigned to the robot in the computer program. So in our experiments the path consists of configuration of the robot having position (x, y, z) varying linearly while the orientation of the robot from the initial to final position remains the same.

Once the path is decided the robot should be provided with a sequence of joint angles one after the other. This sequence must be having time at which the robot should achieve the configuration on the path. This problem is called **trajectory planning**. A trajectory consists of a sequence of robot configurations from initial to final configuration with a time stamp corresponding to each configuration at which the robot should align itself. Trajectory planning turns a specified path / sequence of configurations into a time-based sequence of joint position positions. UR5 also takes joint velocity as input so sequence of joint velocities can also be provided. Path planning takes path specification (sequence of configuration), path constraints (to maintain continuity and smoothness) along with some parameters related to time. The figure below shows the graphical representation of trajectory planning.



The derivatives of q (q is joint position) obtained from the trajectory planner output as shown in the above graph is given as input to the UR5 robot.

Trajectory planning can be broadly divided into two groups

1. Point to point trajectory planning
2. Continuous path trajectory planning

Point to point planning just needs initial and final points and the intermediate path is not critical. Continuous path trajectory planning is needed when there is a need to follow a complex path through 3-D space possibly with high speed.

Point to point trajectory can be done using several approaches such as:
1. Constant velocity trajectory
2. Cubic polynomial trajectory
3. Quintic polynomial trajectory
4. Trapezoidal / linear segment with parabolic blend

Implementing point to point planning is simple by fitting position and velocities to posynomials. Order of polynomials depends on the number of constraints in terms of position, velocity and acceleration.

When a path is described in terms of the number of points more than two then continuous path generation is used. Continuous path trajectory generation requires initial and final point on path along with intermediate waypoint(s) on the path. Some approaches of generating trajectories for this are:
1. Cycloidal trajectory
2. Piecewise cubic polynomial interpolation
3. Piecewise quintic polynomial interpolation
4. Multi-segment linear path with parabolic blends

Continuous path planning is used to deal with oscillatory behavior in the robotic arm when point to point planning is used. Also when the order of the polynomial becomes higher, numerical accuracy for computation of polynomial coefficients decreases. The resulting system of constraint equations becomes complex to solve when the point to point approach is chosen for a higher number of points. So usually when the points available on the path are more than two continuous path generation based trajectory planning should be used.

In our case only initial configuration of the robot i.e. the location where the operator stops its operation and the final location i.e. the home location is specified point to point trajectory planning can be used. But a better approach is to specify that a multiple number of intermediate points lying on the straight line from initial to final location should be specified to keep the motion of the end-effector constrained. This is done in order to keep the motion of the end-effector constrained as the tool will be attached to it which will be dangerous if the intermediate motion is not constrained.

A constant velocity as well as cycloidal trajectory based method was used to plan the trajectory in the homing task. We have discretized the path into a very high number of intermediate points. In our experiment **Jacobian based trajectory tracking** method is used to track the trajectory.

As this method directly generates the joint velocity to track the trajectory specified in cartesian space and the obtained joint velocities can be commanded to the individual joints using the ROS interface.

Jacobian based trajectory tracking algorithm
UR5 is a serial robot manipulator. Joint coordinates of the robot is given by vector q and cartesian space coordinates are given by vector x.

$$q = [\theta_1\ \theta_2\, \theta_3\, \theta_4\, \theta_5\, \theta_6]$$
$$x = [x\, y\, z\, w_x\, w_y\, w_z]$$

The vector of Cartesian space coordinates x of the trajectory is related to joint coordinates q by following equation
$$x = f(q)$$

Here $f(q)$ is a nonlinear differential function which is obtained from the forward kinematics using DH parameters of the manipulator.

Inverse of the above equation can be used when one has to map cartesian position to the joint space coordinates (joint angles).
$$q = f^{-1}(x)$$

Similarly if the linear velocity in cartesian space is specified by V, the joint velocity vector $\dot{q}$ can be related to it as
$$V = J\dot{q}$$

Which can be rewritten as follows
$$\dot{q} = J^{-1}V$$

Here J is called the **jacobian matrix**. The jacobian matrix **must be invertible** so that cartesian velocity can be mapped to the joint space velocity using the above equation. If the jacobian matrix is not invertible then for any given joint level velocity the robot will not be able to attain the desired cartesian space velocity thus will not reach the desired position in cartesian space.

In differential motion control, the desired trajectory is further divided into sampling points separated by time interval Δt between two consecutive points on the trajectory. If the time t_i is the time for joint $q(t_i)$, then the required $q(t_i + \Delta t)$ is calculated using the following equation

$$q(t_i + \Delta t) = q(t_i) + \dot{q}\Delta t$$

Above equation is written in joint space which can be further specified using position and velocity in cartesian as follows

$$q(t_i + \Delta t) = f^{-1}(x_{t_i}) + J^{-1} V \Delta t$$

This equation is a kinematic control law which is based on the Jacobian matrix of the robotic manipulator. This equation outputs the joint position but the UR5 takes joint velocity in our control scheme. So directly joint velocities were calculated for the manipulator.

f) Force/torque to motion control of the robotic arm

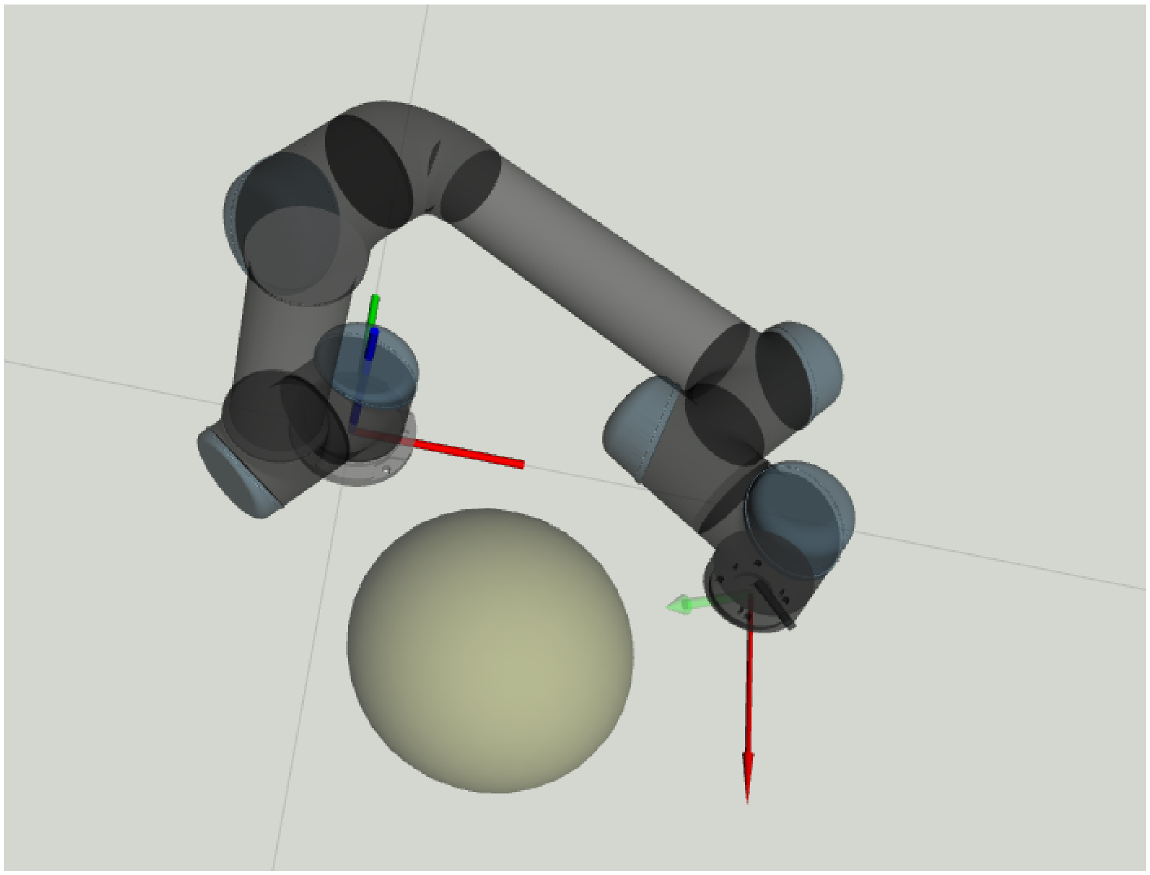
i) Using the load data available from joint torques of the robotic arm

ii) Using the load data available from load cell mounted on the robotic arm

g) Confining the workspace of the robot and its visualization in Rviz

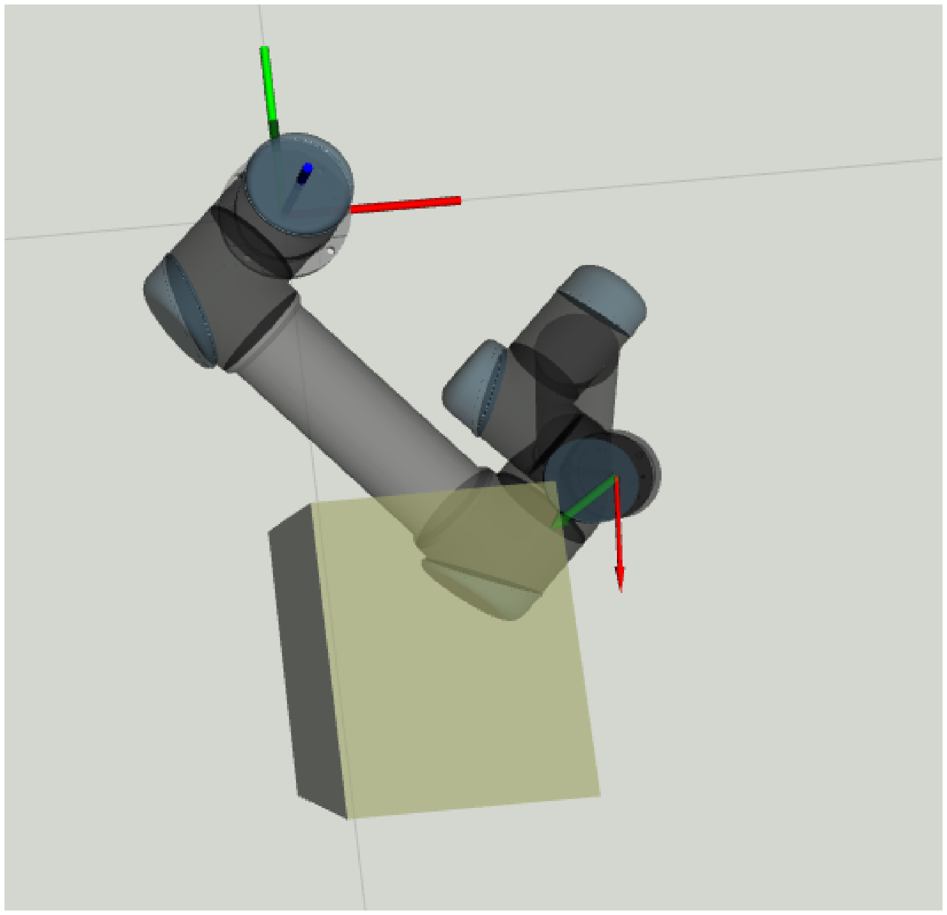
i) Spherical workspace:

For confining the robot's workspace in a sphere, we need two parameters that is center's coordinates and radius. When the EE is inside this workspace then it will move freely as commanded. The algorithm developed for this scheme ensures that if robot's EE is commanded to move away from defined workspace the joint velocity input must be zero and if we try to move it towards workspace then it should move towards the centroid of the workspace with a projected velocity of the EE. This is done because at the very start of some dentistry operation the robot will be outside of the mouth cavity, the dentist will need to bring the gripper/tool inside this cavity.
Refer to the below image taken from Rviz, the velocity vector shown in green is the actual velocity of the EE which is calculated by the developed algorithm and the velocity vector shown in red is the commanded velocity by the Joystick.



ii) Cuboidal workspace:

To define a cuboid in the space we need the dimension of its edges and we need to fix on of its vertices. All three edges of this cuboid is written in the form of vectors in the algorithm developed. To find whether the EE's position is inside this workspace or not, a volume based approach is used. Connect the EE's coordinates to all of the vertices of the cuboid, this way we will get six pyramids. If the volume of all six pyramids is less than the volume of the cuboid them the EE is inside the workspace and if its greater than cuboid''s volume them EE is outside the cuboidal workspace.
The move the EE back inside the workspace the same algorithm as that was used in the spherical workspace is used.



Touch haptic device user guide:

Features:

- Dexterous Serial manipulator design.
- Six degree-of-freedom positional sensing.
- Portable design and compact footprint for workplace flexibility.
- Compact workspace for ease-of-use.
- Comfortable molded-rubber stylus with textured paint for long term use and secure grip.
- Two switches on the stylus for ease of use and end-user customization.
- Multi-function indicator light.
- Wrist-rest to maximize user comfort.
- USB 2.0/3.0 compatibility required

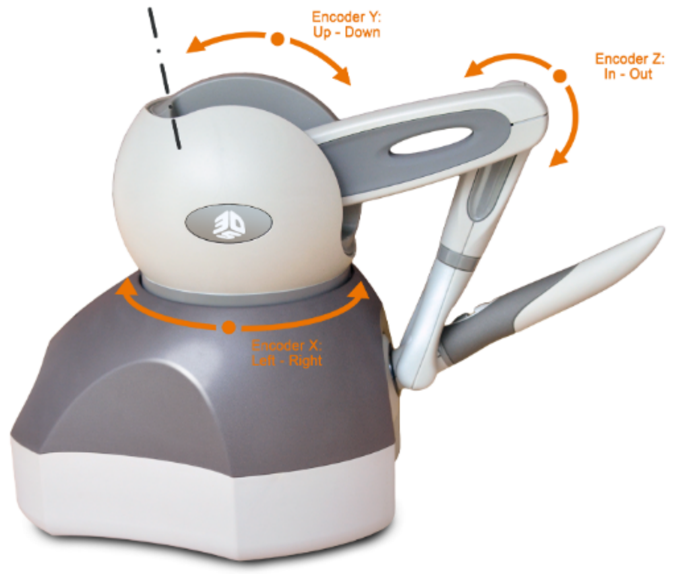
System requirements:

- Intel i5/i7 5th generation, or equivalent CPU
- 4 GB of RAM
- Win 7 64-bit, Win 8.1 64-bit, Win 10 64-bit
- Minimum 256 MB VRAM graphics card (Make sure you have the latest drivers)
- 512 MB of installation disk space
- Minimum display resolution of 1280 x 800
- USB 2.0 / 3.0 compatible port or USB hub

Encoder movement:

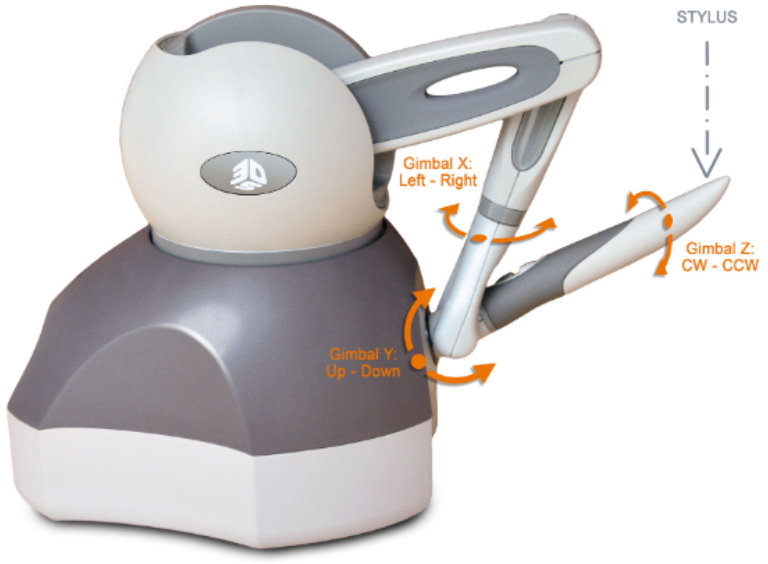
Macro movements: body/base, shoulder and elbow

Encoder articulation: Macro movements. Showing movement of the 3D Systems Touch arm and the body.



Micro movements: roll, pitch and yaw movement of stylus

Gimbal articulation: Micro movements. Showing movement of the 3D Systems Touch stylus.



Connections and installing drivers in linux:

Step 1: In the package you will get a Touch haptic device, a Type A to Type B USB cable, Universal power supply and a power cord. Connect the power cord to the universal power supply. Connect one end of the USB cable (type A) to your PC/laptop and the other end (type B) to the haptic device. Plug the power cord into the available outlet. The blue inkwell LED should quickly blink twice to indicate proper performance.

Step 2: Check if your system recognized the device or not. Open the terminal and run following command:

```
$ ls -l /dev/ttyACM*
```

you will get a result similar to this:

```
crw-rw---- 1 root dialout 166, 0 Jun  5 10:12 /dev/ttyACM0
```

Step 3: Run the following command:

```
$ udevadm info -a -n /dev/ttyACM0 | grep '{product}' | head -n1
```

If you get the output similar to the following then your device is properly connected:

```
ATTR{product}== "3D systems Touch 3d Stylus"
```

Another way to check whether the device is connected is to run the following command:

```
$ lsusb
```

You will get output similar to the following:

```
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 002: ID 13d3:56c9 IMC Networks
Bus 001 Device 004: ID 2988:0302
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

The output should show a device with ID 2988:0302. This is the ID of the Touch device manufactured by 3D Systems Inc.

Step 4: Once it is verified that the device is connected, we have to give read, write, execute permission for all user/group/other to access the device by using the following command:

```
$ sudo chmod 777 /dev/ttyACM0
```

Step 4: Run the following command to install the dependencies:

```
$ sudo apt-get install --no-install-recommends freeglut3-dev g++ libdrm-dev libexpat1-dev libglw1-mesa libglw1-mesa-dev libmotif-dev libncurses5-dev libraw1394-dev libx11-dev libxdamage-dev libxext-dev libxt-dev libxxf86vm-dev tcsh unzip x11proto-dri2-dev x11proto-gl-dev x11proto-print-dev
```

Step 5: Download and extract OpenHaptics and Haptic device drivers.

https://support.3dsystems.com/s/article/OpenHaptics-for-Linux-Developer-Edition-v34?language=en_US

Step 6: Run following commands and install OpenHaptics:

```
$ cd ~/openhaptics_3.4-0-developer-edition-amd64/
$ sudo ./install
# This gets installed in the following directory
/opt/OpenHaptics/
```

Step 7: Download and extract Geomagic driver. (Try to find the newer drivers over the internet, but if you couldn't find it, click on the following link to get the old device drivers. It will work fine.)

Haptic Drivers

Step 8: Install the Geomagic drivers:

```
$ cd ~/geomagic_touch_device_driver_2015.5-26-amd64/
$ sudo ./install
# This gets installed in the following directory:
/opt/geomagic_touch_device_driver/
```

Step 9: Run Geomagic_Touch_Setup in /opt/geomagic_touch_device_driver/. Ensure that the device serial number is displayed.

Step 10: Run Geomagic_Touch_Diagnostic in/opt/geomagic_touch_device_driver/
This can be used to calibrate the device, read encoders, apply test forces etc.

Interface with ROS:

Step 1: Go to your working directory/workspace:

```
$ cd ~/catkin_ws/src
```

Step 2: Run following command to install the repository:

```
$ wstool init .
$ wstool merge
https://raw.github.com/fsuarez6/phantom_omni/hydro-devel/phantom_omni.rosinstall
$ wstool update
```

Step 3: Check for any missing dependencies using rosdep:

```
$ source /opt/ros/$ROS_DISTRO/setup.bash
$ rosdep update
$ rosdep check --from-paths . --ignore-src --rosdistro $ROS_DISTRO
```

Step 4: After installing the missing dependencies, compile your ROS workspace.

```
$ cd ~/catkin_ws
$ catkin_make
```

Note: If you have created a different workspace, make sure to always source the appropriate ROS setup file or better add it in your .bashrc file.

```
$source ~/catkin_ws/devel/setup.bash
```

Step 5: Test the installation by running the following launch file:

```
$ roslaunch omni_common omni_state.launch
```

In case you find any issue, refer to the following links:

https://github.com/bharatm11/Geomagic_Touch_ROS_Drivers

https://fsuarez6.github.io/projects/geomagic-touch-in-ros/

User guide for Touch haptic device:

https://www.3dsystems.com/sites/default/files/2017-12/3DSystems-Touch-UserGuide.pdf

About package:

The data from the haptic device can be read from the following topics:

/phantom/button

/phantom/force_feedback

/phantom/joint_states

/phantom/pose

/phantom/state

Appendix:

Jacobian based control
Jacobian based trajectory tracking
PID controller