Telerobotic Control in Virtual Reality

Zaid Gharaybeh

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science

University of Washington

2019

Committee:

Howard Chizeck, Chair

Andy Stewart

Program Authorized to Offer Degree:
Electrical and Computer Engineering

University of Washington

Abstract

Telerobotic Control in Virtual Reality

Zaid Gharaybeh

Chair of the Supervisory Committee:
Professor Howard Chizeck
Department of Electrical and Computer Engineering

The U.S. Army Corps of Engineers and the Navy have identified more than 400 underwater sites potentially contaminated with munitions [22]. These munitions were left as a result of past military activities. They are an explosive hazard and their contents pose an environmental risk. An effective ROS (Robot Operating System) package that interfaces with Simulink/Matlab was developed to teleoperate an underwater robotic system in Virtual Reality for the remediation of these munitions. The key advantage of this VR teleoperation method is more effective and intuitive control in comparison to teleoperation methods that utilize a 2D monitor and joystick/keyboard/manipulandum input device. The ROS package was built to work with the Oculus DK2 VR headset and the Razer^{TM} Hydra hand controllers.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1
INTRODUCTION

1.1 Motivation

Due to past military training and weapons testing activities, undetonated munitions were left in many places, whose area is on the scale of millions of acres. Many of these munitions exist in underwater environments such as ponds, rivers, and coastal areas. They are an explosive hazard and environmental hazard due to their constituents leaking out over time. Current methods for the remediation of these munitions include sending divers to manually dismantle them or triggering them to explode using blow-in-place strategies[22] as shown in figure 1.1. Blow in place strategies are extremely environmentally harmful as all the chemicals locked in the munition are instantly released into the water, and the shock waves released by the explosion harm nearby marine life. Manual remediation is dangerous to the divers as inadvertent detonation of the munitions could lead to dire results. A cleaner and safer method is needed.

Teleoperation is the control of a robot/machine from a distance. Telerobotic systems include robotic systems that are remotely operated to conduct critical tasks that are difficult to automate and that rely on expert knowledge. Utilizing an expert-driven telerobotic system to conduct the remediation of these underwater undetonated munitions mitigates these individual and environmental safety concerns. A telerobotic approach is much safer than manual remediation approaches because there is no more need for personnel to be present on site. It offers an environmentally cleaner approach than blow-in-place strategies because it does not entail the detonation of anything and no harmful chemicals are released into the water.

Telerobotic systems can also be augmented to afford greater precision, dexterity, and

[23]

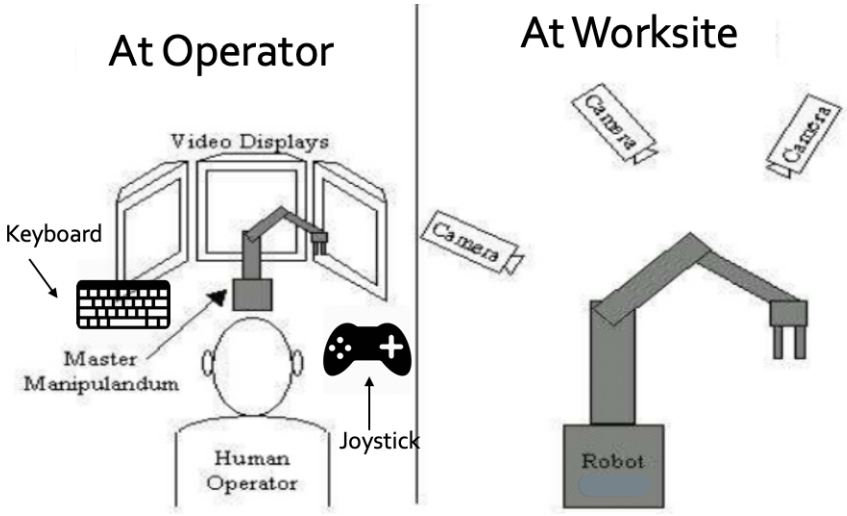
Figure 1.1: Manual Remediation

adaptability by employing advanced robotic technologies. Virtual fixtures and haptic feedback, for instance, when instantiated by experts can be used to assist the teleoperator when making physical contact with the munitions.

Teleoperation is therefore a feasible alternative approach to retrieve those munitions. However, conventional teleoperation approaches that utilize 2D monitor and input methods (see figure 1.2) are not easily controllable due to a lack of depth perception, static views, and awkward/unintuitive and ineffective input control methods. But conducting the teleoperation in Virtual Reality can mitigate these drawbacks, as is detailed in the next section. Control methods to teleoperate a robotic system in virtual reality are developed for the remediation of these underwater munitions. These methods are generalizable to the control of any 6DoF robot arm.

1.2 Project Goal

Develop a method to teleoperate a robotic system in virtual reality to safely retrieve these underwater munitions.



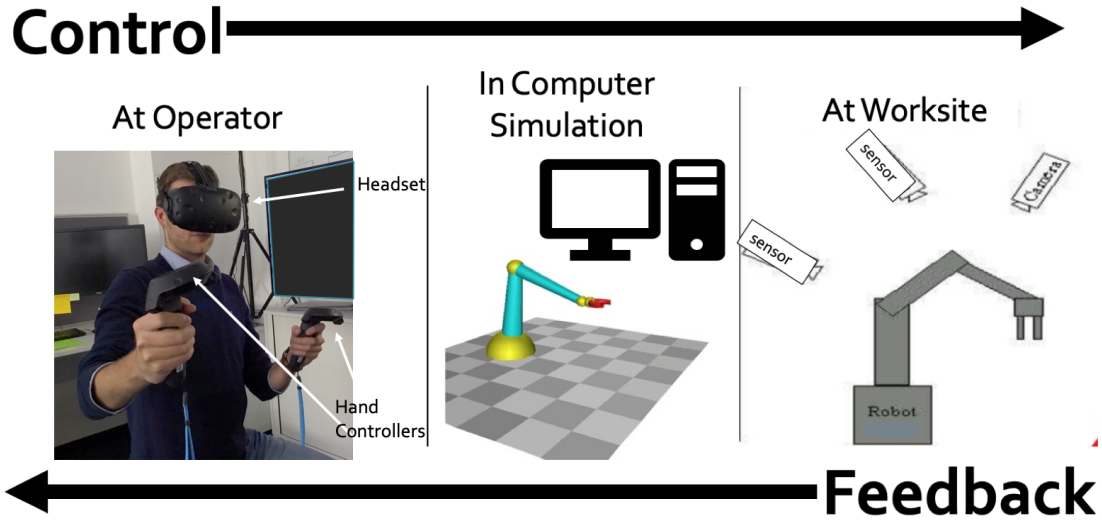
[11]

Figure 1.2: Teleoperation

1.3 Teleoperation Background

Teleoperation is the remote control of a robot/machine. The operator has a way to view the work site and interact with the environment by controlling the robot using various input devices such as mouse/keyboard, joystick, or master manipulandum such as the Phantom Omni.

Traditional teleoperation approaches usually utilize a 2D monitor for the operator to view the worksite environment and a mouse/keyboard, joystick, or master manipulandum as the input device to control the robot. The issue with these approaches is that they make it difficult to control the robot and increase the chance of making errors, for several reasons. Firstly, using a 2D monitor decreases the operator's ability to perceive depth. Robust depth perception is important when interacting and controlling objects in 3D space. Secondly, if the cameras used at the worksite are stationary, it will lead to static views on the 2D monitor for the operator (in other words, operator can only observe worksite environment from static frames). This forces the operator to have to continuously mentally rotate/scale/translate the robot he sees on the monitor to conduct the desired movement using the input device. Lastly,



[1][2]

Figure 1.3: Virtual Reality Teleoperation

the input methods (keyboard, mouse, joystick) are awkward and difficult to use because they don't make use of humans' natural ability to manipulate objects in 3D space. A drawbacks of using a master manipulandum such as the Phantom Omni is the ability to only control via moving the end-effector/difficulty controlling individual joints.

Conducting the teleoperation task in Virtual Reality can mitigate these drawbacks.

In virtual reality (VR), the operator wears a VR Headset that tracks the position and orientation of his head in real time. He holds two hand controllers, one in each hand, that are also equipped with real-time position and orientation tracking and provide input methods. Through his headset, he sees an immersive 3D simulation of the worksite (underwater environment) and interacts with a simulated 3D model of the worksite robot. He also sees the hand controllers in real time in the simulation as visual representations of their positions and orientations relative to his head. He then performs the remediation task by controlling the simulated robot using the hand controllers. The real robot follows the motion of the simulated robot.

VR teleoperation brings back depth perception. The use of a simulated 3D model of

the worksite and robot allow for free and unconstrained movement of the operator's frame of view. The use of hand controllers with 3D position and orientation tracking enables effective and natural/intuitive interaction with simulated 3D objects. For these reasons, VR teleoperation delivers more effective and intuitive control than the 2D monitor + input methods discussed earlier.

1.4 Contributions

- Telerobotic control methods in Virtual Reality of 6DoF 6Rotary robot arm implemented in ROS/ROS-Visualizer (RVIZ) as a ROS package.

- Generalizable to any 6DoF 6-joints robot arm

1.5 Literature Review

1.5.1 Related Work

It has been previously shown that non-expert users are more efficient with a VR teleoperation interface, and that they prefer using one, compared to a keyboard and monitor interface[6]. We believe, with proper implementation of control methodologies, this extends to experienced users. Previous works that used Virtual Reality for teleoperation include the "Homunculus" model[8] where a view from a 2D camera from Baxter's head was projected to virtual reality, creating a an experience where the user feels like he is the robot. Other works used a live point cloud from an RGB-D camera to perform complex manipulation tasks using deep imitation learning[24], also from a fixed frame.

1.6 Remediation Hardware

The hardware being tested for the remediation task resides in UW's Applied Physics Lab (APL). The hardware includes the Robot Arm (Schilling Titan4 Manipulator), which is a 6DoF 6-Rotary robot arm, LiDAR sensor, and cameras. Testing is done by attaching the robot arm to a frame within working range of a large open-top container filled with gravel

Figure 1.4: Hardware: Schilling Titan4 Manipulator

[13]

holding a dummy munition, and submerging the frame at UW's Ocean Sciences Building water tank/pool, and conducting the remediation.

These devices interface with a machine holding commercial software from Olis Robotics. The Olis software directly controls the Schilling Robot Arm.

1.7 Setting Up Software

1.7.1 Target Platform

This software platform targets:

- Oculus Version: Oculus DK2
- OS: Ubuntu 14.04
- ROS Version: Indigo

Figure 1.5: Hardware: LiDAR Scanner

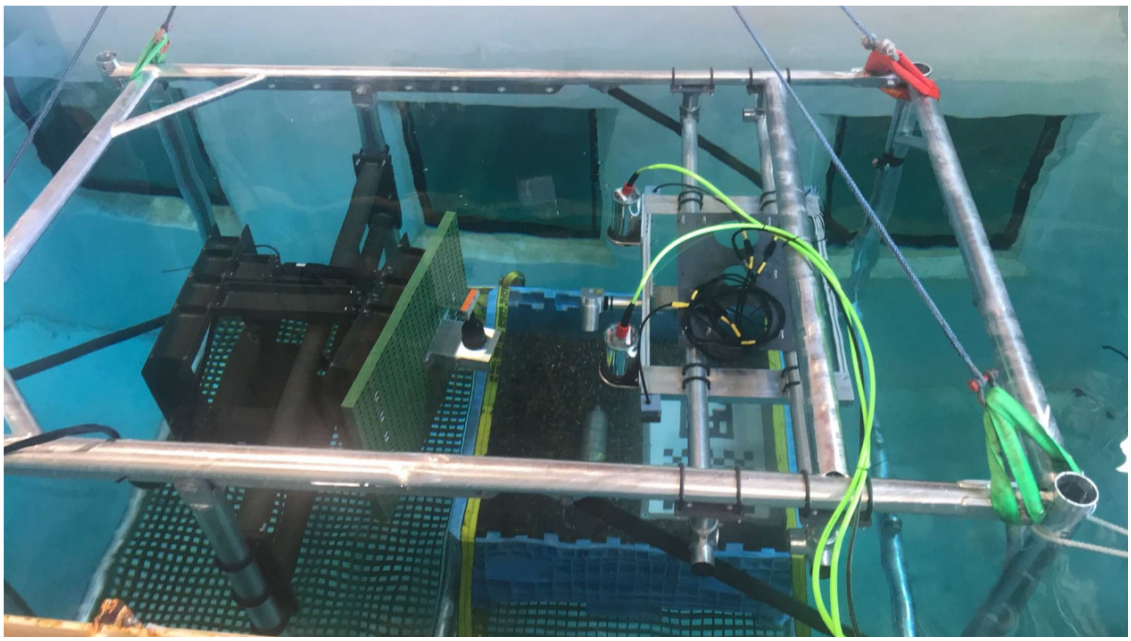


Figure 1.6: Hardware: Testing at UW Ocean Sciences Building's (OSB) immersion tank

Figure 1.7: Hardware: Frame for Robot Arm for Testing Purposes

1.7.2 Launching Software Steps

Here are the steps to set up and launch the software. Firstly, make sure that the Oculus and Hydra are properly recognized by the operating system: the rules file should be modified in /etc/udev/rules.d/ - check this link for more information:
https://github.com/OSUrobotics/ros_ovr_sdk/issues/6

1. Make sure the oculus_rviz_hydra_teleoperation, razer_hydra, ros_ovr_sdk, and oculus_rviz_plugins packages are built in your catkin workspace.

2. Align the Oculus headset straight in front of you facing downwards, and parallel to your body, as is described in figure 2.27

3. In a sourced terminal, run the following ros nodes, where oculus_hydra_rviz_teleoperation is the name of the package. Alternatively, just run the shell script provided you modify the paths inside
   rosrun ros_ovr_sdk ovrd

   roslaunch razer_hydra hydra.launch publish_tf:=true

rosrun oculus_hydra_rviz_teleoperation hydra_presence

rosrun oculus_hydra_rviz_teleoperation rviz_oculus_hydra_nav_2

rosrun oculus_hydra_rviz_teleoperation IK_direct_schilling_control_matlab

rosrun oculus_hydra_rviz_teleoperation jaw_grab

rosrun oculus_hydra_rviz_teleoperation schilling_trigger_control

rosrun oculus_hydra_rviz_teleoperation IK_path_schilling_control_matlab

rosrun oculus_hydra_rviz_teleoperation schilling_stream

sleep 8;

roslaunch oculus_hydra_rviz_teleoperation rviz_no_gui.launch model:=PATH-TO-PACKAGE /mesh/schilling_detached_gantry.urdf

rosrun oculus_hydra_rviz_teleoperation virtual_fixture_coords rosrun oculus_hydra_rviz_teleoperation virtual_fixture_markers

rosrun oculus_hydra_rviz_teleoperation undo_motion

rosrun oculus_hydra_rviz_teleoperation lidar_publish "package://oculus_hydra_rviz_teleoperation/mes colored.stl"

4. Run the matlab IK solver file (change path inside)

5. In RVIZ, add marker display. Add Marker array display. Add Oculus display. Within the Oculus display options, check Render to oculus. Then choose the oculus_mount frame to mount the oculus on the correct frame.

To run with gantry attached, launch the schilling_gantry.urdf file in the rviz launch command instead, and, in addition to the above, run the following ros nodes:

1. simulink_To_ROS_message_parser.cpp

2. run the gantry_dynamics_control.slx simulink model with inf as the end time

1.7.3 Connecting to Worksite Machine

- SSH to the ROS machine from the machine interfacing the schilling manipulator.
- Run "rostopic echo /schilling_stream" to get stream of joint values of simulated robot
- Parse output stream to control schilling
- To visualize LiDAR scan, transfer LiDAR point cloud file to the package in the ROS machine and run lidar publish command with name of file as argument (see section 2.5.4.
- To set/reset simulated schilling's state, run Set/Reset joint values command detailed in section 2.5.3.

Please read section on Communication with hardware for more details

1.8 Specifications of Development PC

- Memory: 8GB
- Processor: Intel Core i7-3930K CPU @ 3.20GHz x 12
- Graphics: Gallium 0.4 on AMD Cayman (DRM 2.43.0, LLVM 3.8.0)

Chapter 2

METHOD

2.1 Approach

- Build model of robot in simulation, work-site robot follows motion of simulated robot.
- Populate simulation with sensor data of worksite.
- Use off the shelf VR headset and hand controllers (Opted for Oculus DK2 headset and Razer Hydra controllers)
- Develop methods to control the robot in VR using the hand controllers that leverage the operator's natural instincts in manipulating objects in 3D from any viewing frame
- Generalizable to any 6DoF 6-joints arm
- Implement everything teleoperation related in the most popular robot software development platform, Robot Operating System. All ROS nodes were written in ROSCPP.

See figure 2.1 for block diagram of complete system. The hardware interfaces with the Olis Robotics Commercial software, which remotely connects to the ROS machine and visualizes in VR. Within the ROS machine, 5 major components were developed, represented by the yellow rectangles. Please note that the underwater dynamics were implemented for the gantry's motion and not the robot arm. The robot arm is assumed to be position controlled.

See figure 2.4 to view all input methods of Razer Hydra controllers.

- A: Analog Stick/Button 6

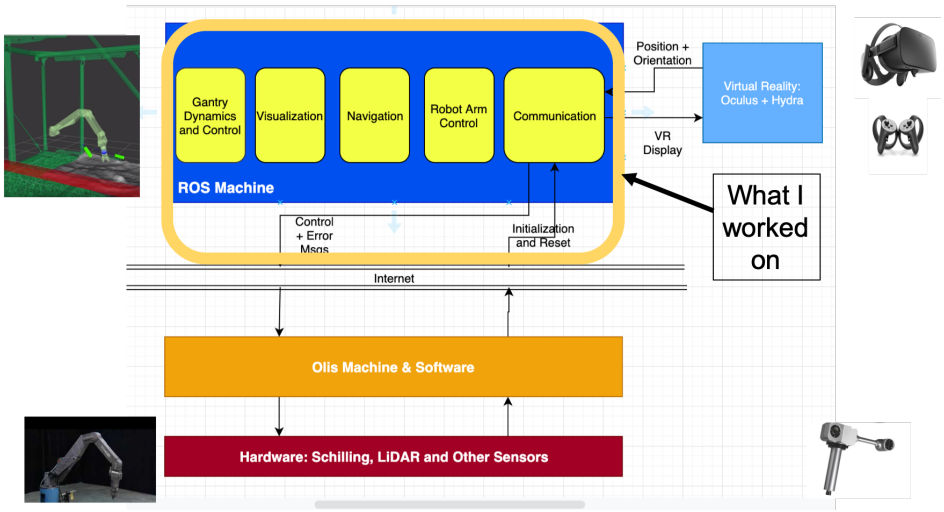


Figure 2.1: Block Diagram of Full Stack

Figure 2.2: Oculus DK2

Figure 2.3: Razer Hydra

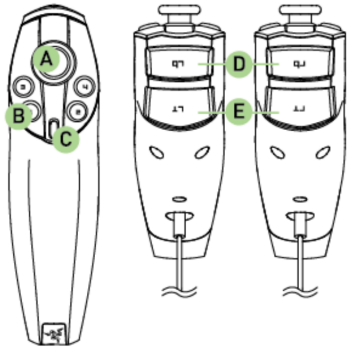


Figure 2.4: Hydra Controls

[12]

- B: Buttons 1:4
- C: Button 5
- D: Bumper Buttons
- E: Trigger Buttons

2.1.1 Design Choices

Robotics Systems Software: Why ROS?

The most popular software framework for the development of robot software solutions is Robot operating system (ROS). ROS is a flexible framework for writing robotics software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms [18]. A ROS implementation primarily exists as *nodes*, which are simple programs, *subscribing* and *publishing* messages of specified types to different *topics*. Any node can subscribe and publish to any number of topics. Any message published to a topic is received by all nodes who subscribe to that topic. This highly modular framework is suitable for our implementation, and allows for easy integration of additional advanced components, making this software package upgrade and developer friendly.

However, ROS lacks the capability to simulate real-time dynamical systems. To simulate the dynamics of our underwater environment for the motion of the gantry, Simulink was used. Simulink, developed by MathWorks, is a graphical programming environment for modeling, simulating and analyzing multidomain dynamical systems. Its primary interface is a graphical block diagramming tool and a customizable set of block libraries[10]. Furthermore, Matlab/Simulink supports communication to ROS through it's Robotics Systems Toolbox. Lastly, the ROS Visualizer environment (RVIZ) is a robust tool for 3D visualizations of robotic systems and simulated environments, and has all desired features for our applica-

tion. There also exists a plugin, oculus_rviz_plugins, that projects the RVIZ visualization to the Oculus DK2 VR headset.

VR Headset and Controllers: Why DK2 and Razer Hydra?

The Razer Hydra was used as the VR hand controller. It is a motion and orientation detection game controller. According to its developers, Sixense TrueMotion, "using magnetic motion sensing, ..., the Razer Hydras compact base station can compute the exact location and orientation of controllers in your hands, down to a millimeter and degree, ..., Combined with Razer-grade ultra-low latency, you get a fluid and unhindered experience thats accurate and precise, like a mirror reflection of you in the game world."[5] Unlike the Oculus Touch, which is the newest Oculus Hand controller that comes with the CV1, the Razer Hydra is compatible with Linux, and there exists an open source ROS node for its integration in ROS. The Razer Hydra also has several more inputs than the Oculus Touch. The Touch only has 2 Buttons on each controller, while the Hydra has 5, see figure 2.4.

The Oculus DK2 served as the VR headset. There exists a tool, oculus_rviz_plugins, that projects the RVIZ simulated environment to the Oculus DK2. The newest Oculus headset (CV1) is incompatible with Linux, and is only officially supported to run on Windows at the time of writing.

By choosing the Oculus DK2 and Razer Hydra, full ROS locality of all running software modules was achieved. This made it easy to develop and test the software components, while still delivering good performance.

Furthermore, a tactile-feedback system, called Reactive Grip, can be attached to the Razer Hydra to enable haptic feedback on the device[15]. This enables easy development and testing of haptic feedback/virtual fixture functionality. Haptic feedback was not implemented in this work but can be a next step in the project.

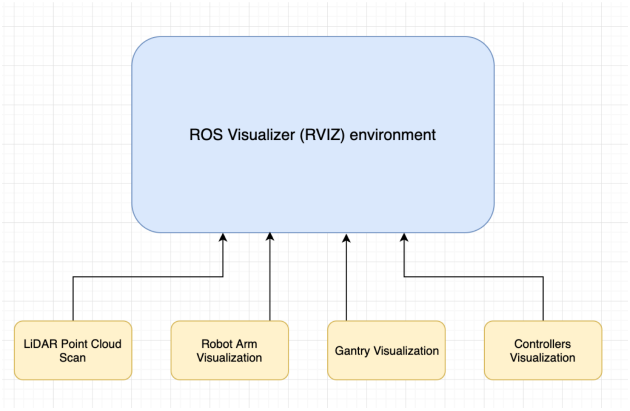


Figure 2.5: Visualization Components

2.2 Visualization

The LiDAR scan, Robot Arm (Schilling), Hydra Controllers, and Gantry (more on the gantry later) are visualized in the ROS Visualizer (RVIZ). The LiDAR scan is visualized in RVIZ as a static point cloud. The Razer Hydra controllers are simulated as ROS Marker objects[1] that have the same shape as the controllers. Robot models of the Schilling Robot Arm and Gantry are generated in ROS, and they are visualized in RVIZ based on their joint configurations. Additionally, the frame used for testing purposes was visualized from a mesh file. This section first gives a background on RVIZ, and then details how each component was visualized.

2.2.1 ROS Visualizer: RVIZ Background

Rviz is a 3D visualizer for the Robot Operating System (ROS) framework, and uses the tf transform system for transforming data from the coordinate frame it arrives in into a global reference frame[19]. tf is a package that lets the user keep track of multiple coordinate frames over time. tf maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc between any two coordinate

[1]Markers are an RVIZ display type that allows points, shapes, and lines to be displayed in RVIZ.

frames at any desired point in time[20].

2.2.2 Razer Hydra Visualization in RVIZ

Real time representations of the Razer Hydras are visualized in RVIZ to allow the operator to see their state and assist him in interacting with the simulation. Two hydra visual models from a .STL mesh file are visualized and published to RVIZ as Marker objects within a Marker Array message at a location that is below and slightly in front of the Operator's default viewing frame in Virtual Reality, which is the Oculus' mounting frame (oculus_mount tf frame). This facilitates the feeling that the hand controllers are the operator's hands, while at the same time enabling him to easily see the hand controllers' states. It is also the standard way of holding the Hydra Controllers. The location and orientation of the markers relative to that position are updated in real time by parsing the hydra_calib topic. Additionally, two Arrow Marker objects are projected from each controller, one of adjustable length, and added in the Marker Array. Moreover, TF frames were published representing the transformation between the Oculus Mount frame and the hydra markers, the Oculus Mount frame and the end of the hydra markers, and the Oculus Mount frame and the end of the Arrows projected from the hydra markers. These will be used to enable functionalities in other nodes.

ROS Node: hydra_presence

This ROS node subscribes to the hydra_calib topic and publishes to the visualization_marker_array topic which allows RVIZ to display the markers. It also publishes several tf transformation messages. It loads the .stl model of the hydra's, and from the hydra_calib topic, it retrieves in real time the position and orientation of the hydra controllers. The orientation is in quaternion form. It visualizes the .stl model with the position/rotation information as a marker object below and in front of the oculus_mount frame. The offset to the oculus_mount frame is modify-able by changing the offset constants' values.

It also adds arrow Markers projected from each controller pointing along the direction the controllers. Finally, it uses the value of the right joystick to adjust the length of the right

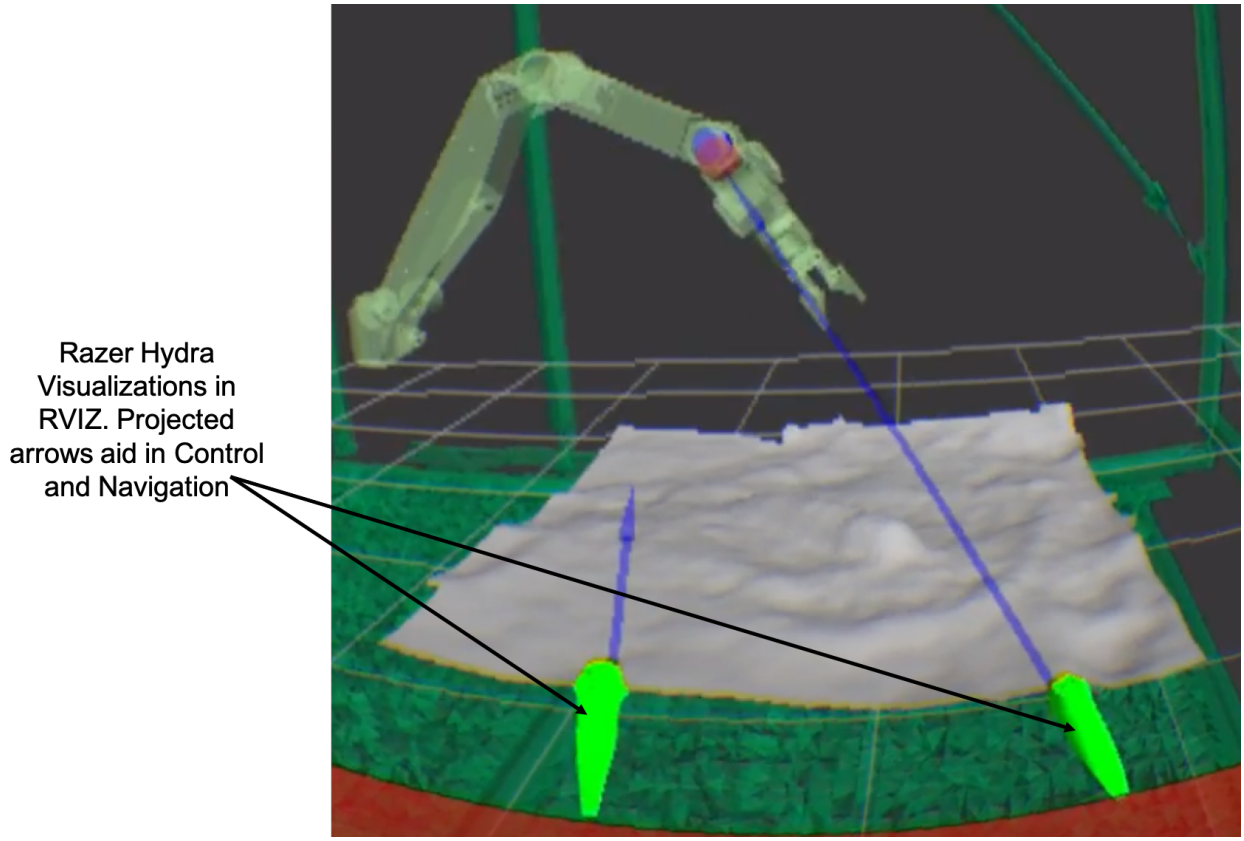


Figure 2.6: Hydra's Presence in RVIZ are a Reflection of the Razer Hydra Controller's Positions and Orientations in Real Time

arrow in real time. The length of the right arrow serves many functionalities detailed later.

2.2.3 Robot Arm (Schilling Titan4 Manipulator) and Gantry Visualization

URDF files[2] representing a model of the Schilling's and Gantry's joints and links was created and loaded into RVIZ. The Gantry's URDF file was created from scratch, while the Schilling's was based on a simplified CAD model of the links, as well as the joint motion ranges detailed in its schematic. The URDF files for all these components were combined as a single URDF file which is passed as an argument to the RVIZ launch command when

[2]Unified Robot Description Format (URDF), which is an XML format for representing a robot model

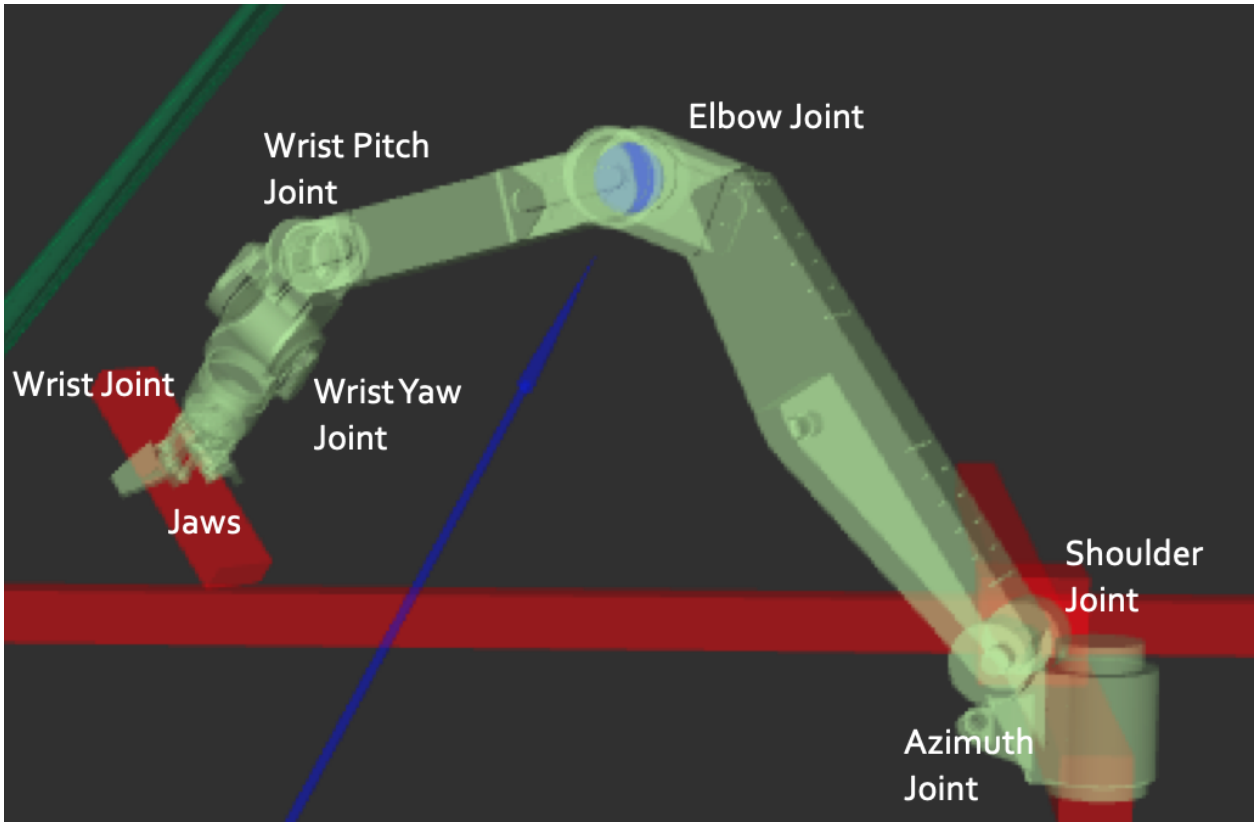


Figure 2.7: Schilling's Visualization

initiating the program. This loaded URDF file also includes additional joints with no visual components acting as axes for mounting the Oculus Mounting frame (oculus_mount) for navigation purposes.

2.2.4 LiDAR Visualization

There was a lot of difficulty trying to get RVIZ to properly display the LiDAR point cloud scan from the .ply file provided. Converting it to an RVIZ/URDF compatible format (.stl or .dae) resulted in the point cloud being too dark (Navy colored) and invisible when looking at it from below. Using Blender, the .ply file was converted to .dae and the format was messed around with until it finally displayed an acceptable opacity and color in RVIZ. This is the

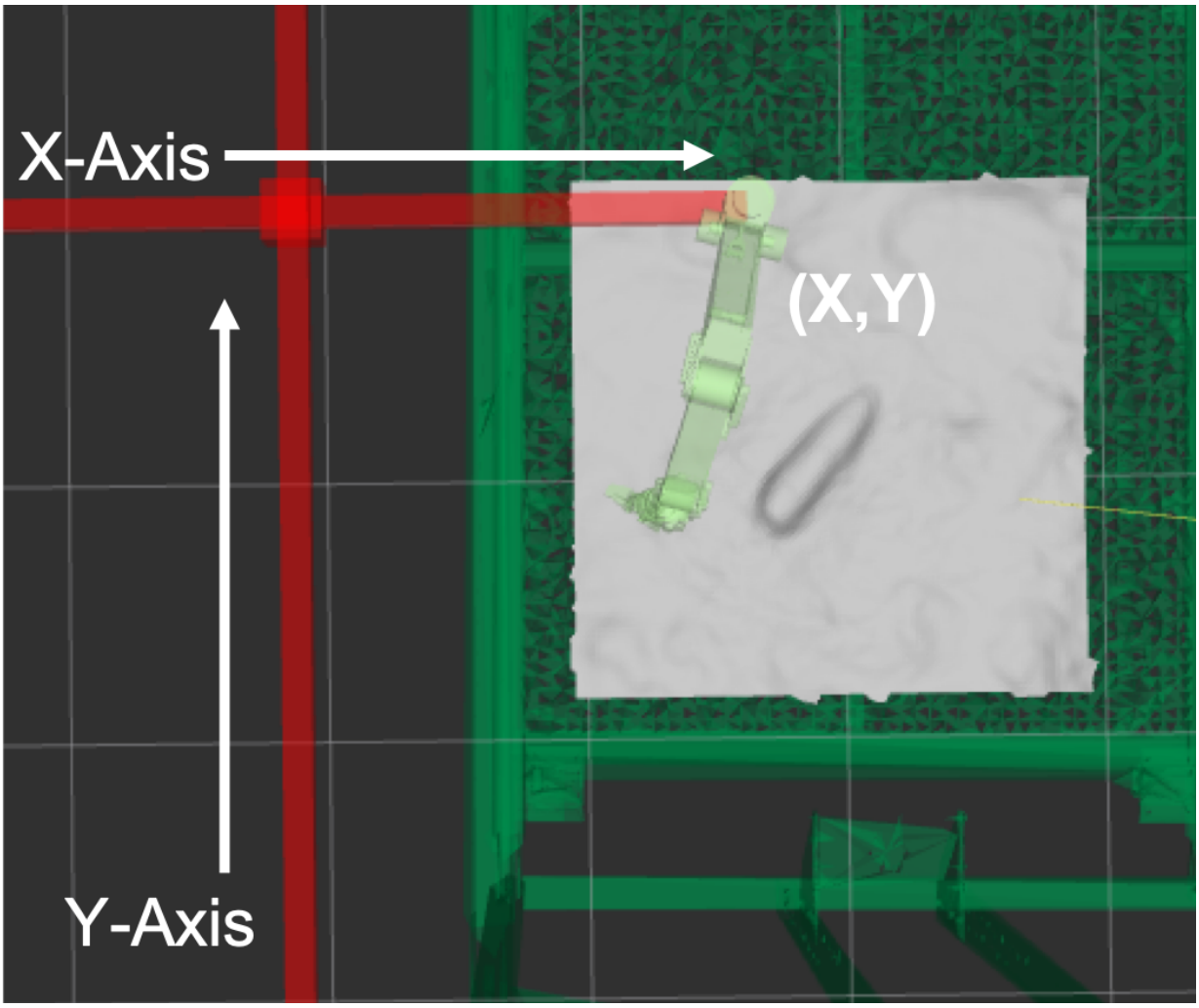


Figure 2.8: Top View: Robot Arm connected at base to Gantry (red rods) end effector, munition below. Green is frame used at UW test bed.

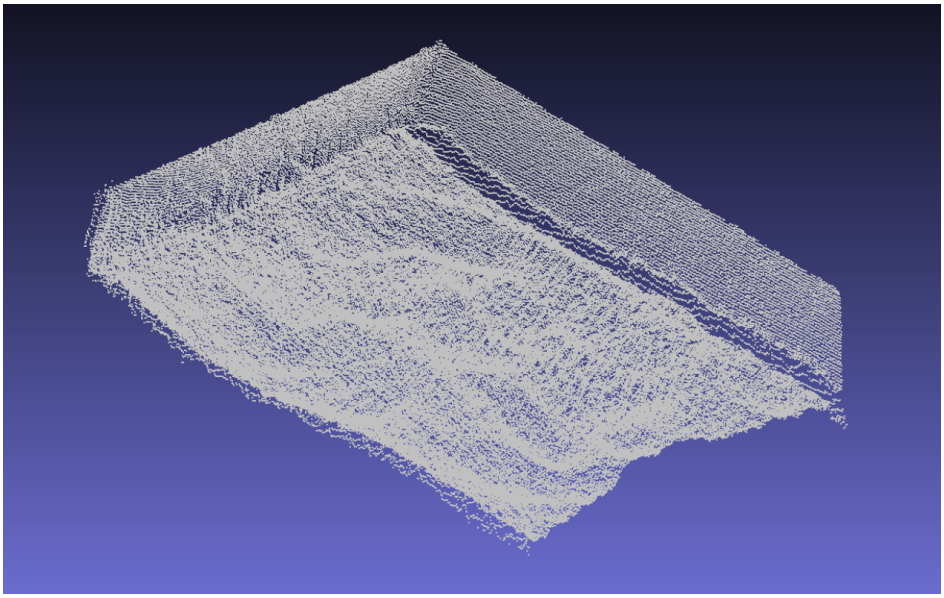


Figure 2.9: PointCloud Scan of Munition and Surroundings by LiDAR

"fromstl.dae" file.

2.3 Navigation

2.3.1 Setting up free moving tf frame Mount for Oculus

A 3-axis (X,Y,Z) cartesian-coordinate space was chosen to be the method by which the Oculus frame moves in RVIZ. 3 orthogonal prismatic joints (base_to_xaxis, xaxis_to_yaxis, and yaxis_to_zaxis), as well as one continuous rotary joint about the zaxis (zaxis_to_tmp_frame) were added to the urdf file that RVIZ loads to serve as tf frames that can carry the Oculus to any point in RVIZ and have any orientation with respect to its yaw. The Oculus display should be mounted on the oculus_mount frame to allow for this. These joints' values are modified in the navigation nodes to execute all movement capabilities that are implemented.

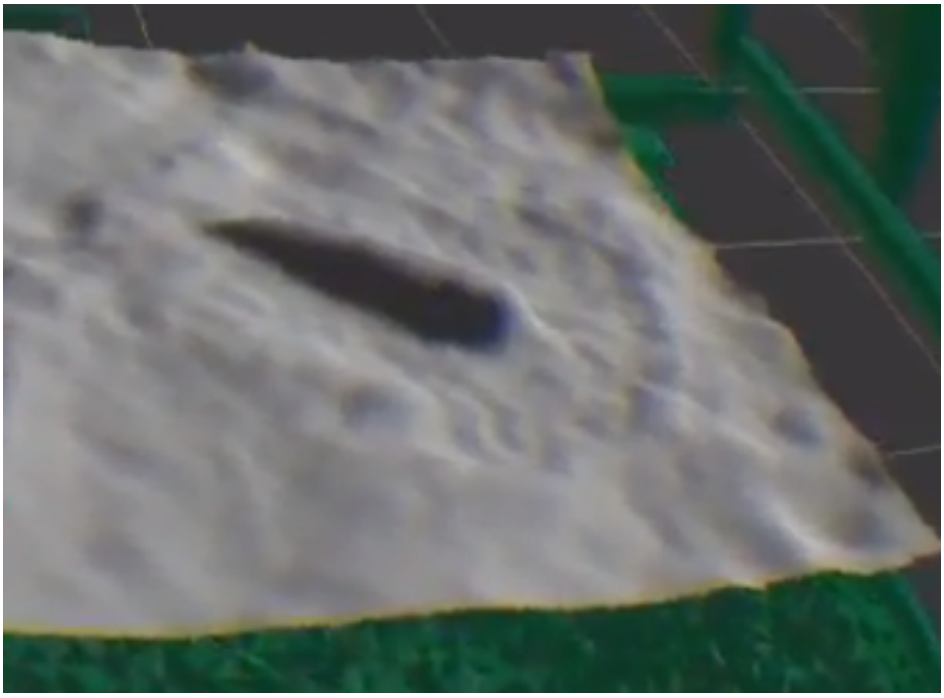


Figure 2.10: LiDAR pointcloud scan visualized in RVIZ

2.3.2 Navigation: Zoom, Translate, Rotate

2 ROS nodes have been written for the functionality of freely navigating/moving the camera in RVIZ using the Razer Hydra, one with a different method of translating, but same method for rotating and zooming in, from the other.

The user can rotate the view about the z-axis (turn left/right) by holding the left bumper and moving the left handle left or right, followed by releasing the bumper to stop rotating.

To zoom in, the user holds the handles close together, presses both bumpers, extends the distance between the handles, and releases the handles. To zoom out, the user presses both bumpers with the hydra handles spaced out, brings the handles together, and releases the bumpers. The zoom direction is the current center of the Oculus' view (the center of the screen in virtual reality) i.e. if the user changes his gaze, the zoom direction changes. This enables robust zooming.

The first method of translation is done by holding the right bumper and moving the right handle in any direction, followed by releasing the bumper to stop translating. The position in RVIZ then moves accordingly to the XYZ motion of the right hydra. The direction of translation is based on the default current viewing angle (the angle if the user is looking straight). The way the first translation implementation works can best be described as "holding a point in space and moving it around" which leads to the camera moving accordingly.

Another translation implementation was done by holding the right bumper and tilting the right handle along its yaw or pitch, followed by releasing the bumper. The position in RVIZ then rotates about the tip of the right hydra's arrow in virtual reality. This is best described as "holding a point in space and changing the angle by which you're looking at it while keeping it at a constant distance away from yourself".

The user can reset the view to be facing the Schilling arm above the point cloud representation by clicking both joystick buttons (buttons 6).

ROS Node: rviz_oculus_hydra_nav_1

This ROS node handles the navigation in RVIZ using the Razer Hydra. It subscribes to 3 ROS topics: /hydra_calib, which is the hydra node that publishes the state of the device, /tf, which publishes the transformations for all coordinate frames, and /joint_states, which publishes the state of all joints (such as rotation for rotary joints and displacement for prismatic joints). It publishes to 1 topic, /joint_states, to move the position and orientation of the Oculus in the virtual environment by modifying the values of the joints that hold the oculus_mount frame.

From the /tf topic, the ROS node continuously extracts the orientation of the Oculus, which by default is in the form of a quaternion. Conversion to Euler angles (roll, pitch, yaw) is done in the toEulerAngle function according to equation shown in figure 2.15.

From the /js topic, the ROS node extracts the *current* position and orientation of the Oculus Mount frame in the visualization, which are the values before any navigation command is being executed. The mount's position in the virtual world is determined by the

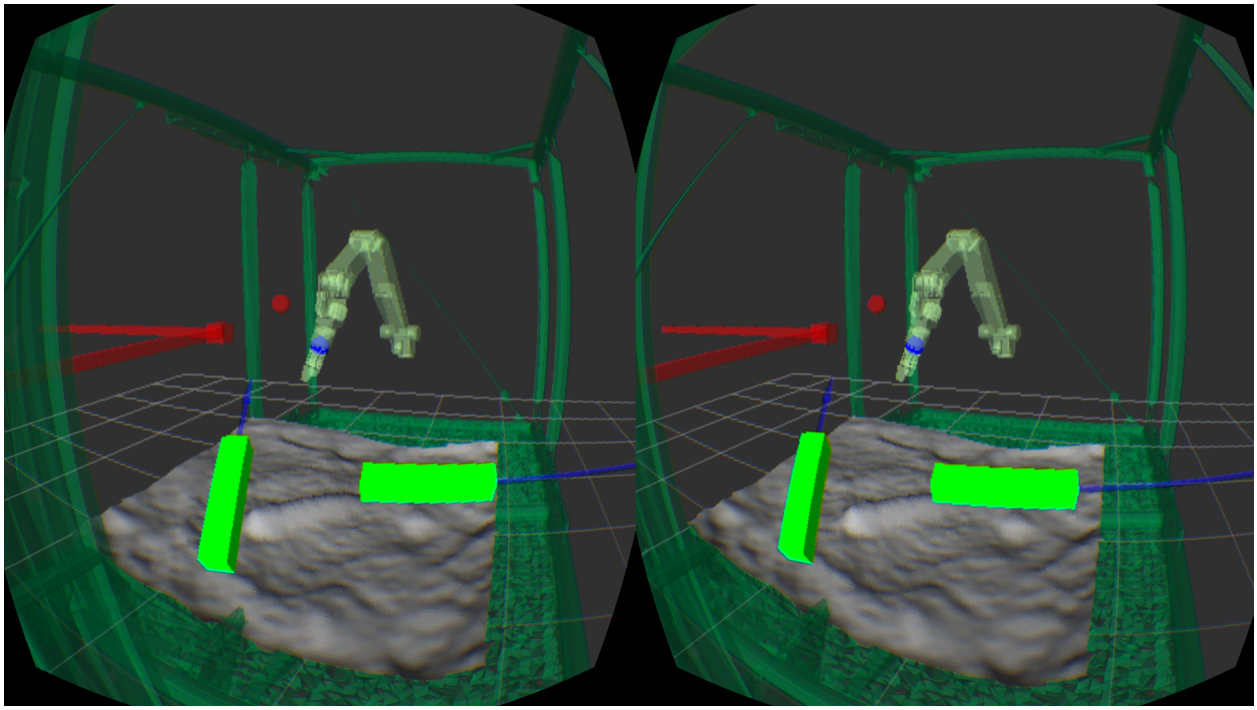


Figure 2.11: Rotation Before

links "base_to_xaxis", "xaxis_to_yaxis", and "yaxis_to_zaxis", and it's rotation wrt the z-axis is determined by "zaxis_to_myoculus". These values are stored in global variables for use by the callback functions.

From the /hydra_calib topic, the ROS node receives the values of the buttons, triggers, and bumpers of the hydra. First, let's consider rotation of the Oculus Mount frame wrt the z-axis. To rotate clockwise, the user presses the left bumper and rotates the left hydra handle about its yaw. The ROS node uses the current value (prior to any buttons being pressed), and the yaw angle of the hydra as the bumper is pressed, to update in real time the zaxis_to_myoculus frame.

Zooming in/out of the Oculus's camera in RVIZ is done in a similar fashion. The ROS node registers the current position (x,y,z), and orientation (roll,pitch,yaw) of the Oculus to calculate the direction in which the zooming will take place. Then, it measures the spreading

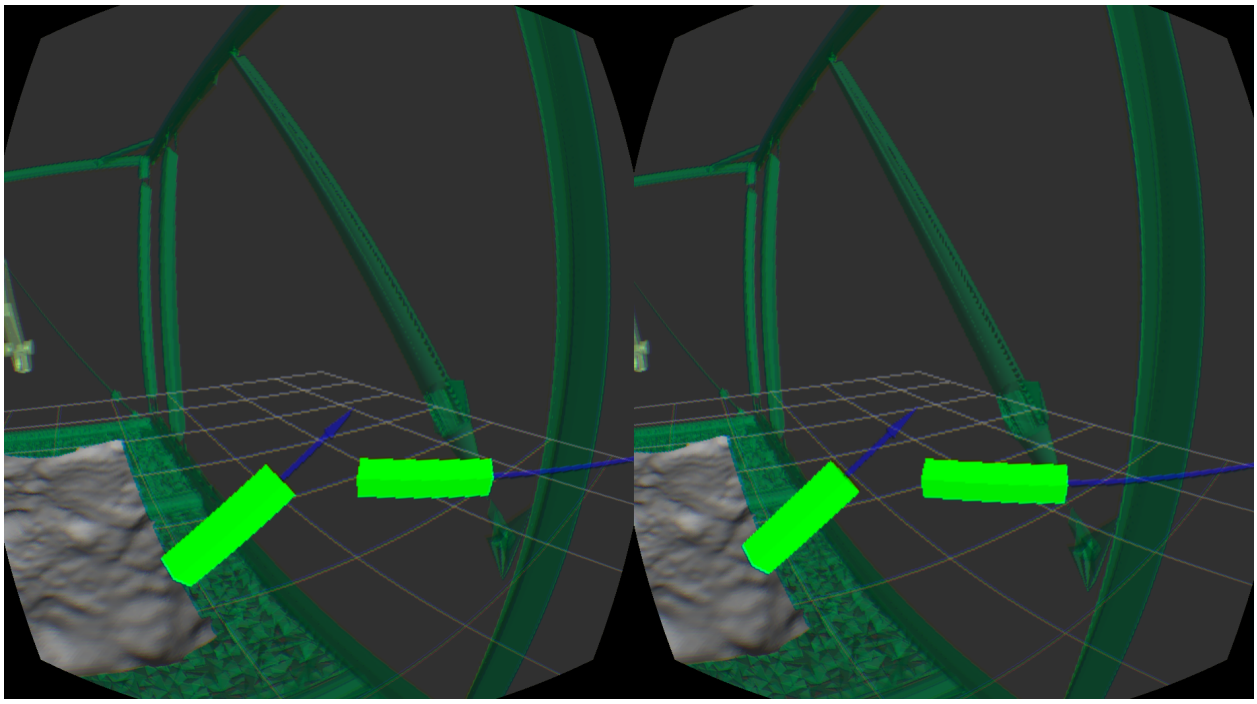


Figure 2.12: Rotation After

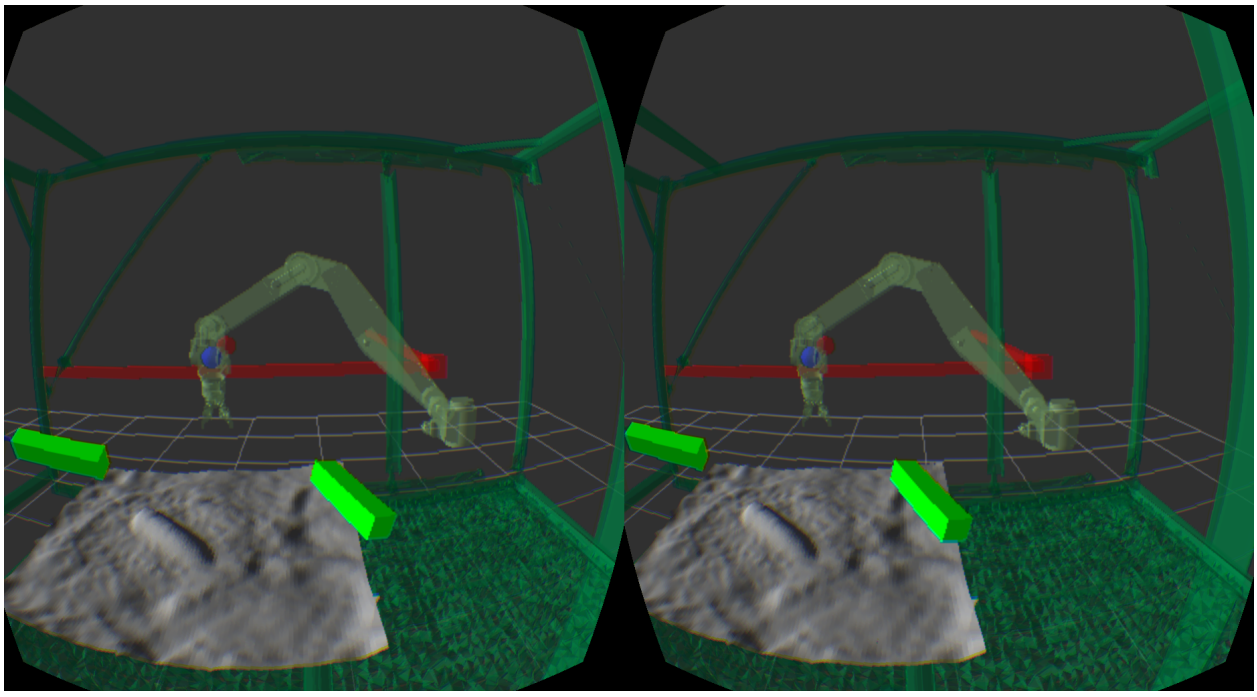


Figure 2.13: Translation Before Using Second Nav Implementation

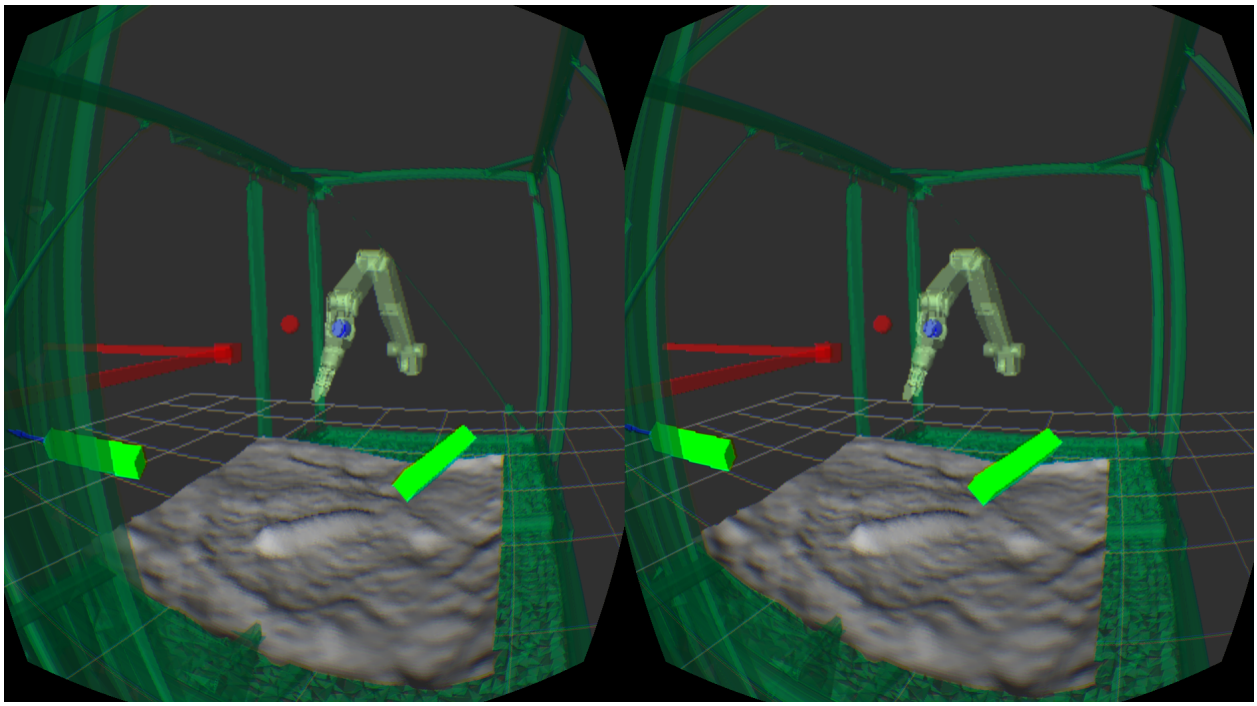


Figure 2.14: Translation After Using Second Nav Implementation

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \operatorname{atan2}(2(q_0 q_1 + q_2 q_3), 1 - 2(q_1^2 + q_2^2)) \\ \operatorname{asin}(2(q_0 q_2 - q_3 q_1)) \\ \operatorname{atan2}(2(q_0 q_3 + q_1 q_2), 1 - 2(q_2^2 + q_3^2)) \end{bmatrix}$$

Figure 2.15: Conversion to Euler Angles

[17]

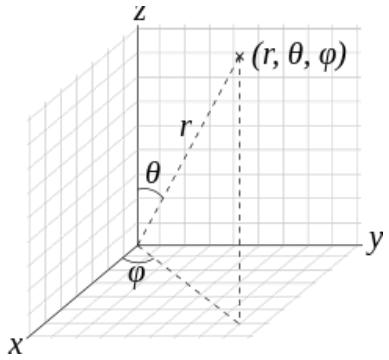


Figure 2.16: Spherical Coordinates: Origin is Tip of Right Hydra's Arrow, current (r,theta,phi) position is Teleoperator's current View. r is kept constant while theta, phi adjust to move camera along sphere's surface

out/bringing in distance between the two Hydras (y) and adds to the *current* values of the x,y,z positions of the oculus mount frame to move in the gaze direction.

ROS Node: rviz_oculus_hydra_nav_2

This implements everything the same way as the node above, except for translation, for which the rotation along the surface of a sphere in the 3D space is the way of translating in the virtual space. The center of this sphere is the tip of the right hydra's arrow in RVIZ. The user can increase the length of this arrow by moving the joystick up, and decrease by moving it down, thereby changing the radius about which the rotation happens. The node keeps track of the length of the arrow (radius) and changes the pitch (θ) and yaw (ϕ) values when the translation is active. These yaw and pitch changes correspond to a new point on the sphere about the origin (tip of the arrow), and the oculus_mount frame's new position wrt to the base frame is computed and published as a joint state message that updates the xaxis,yaxis, and zaxis values on which the oculus_mount frame is mounted, moving the frame.

When the user rotates about a point using the above method, his orientation is also adjusted such that he is always facing the center of the sphere *but only wrt to the change*

in yaw ϕ. For example, if the translation results in a position change corresponding only to a change in θ (pitch), the orientation does not change. This is because the Oculus frame (which is mounted on the oculus_mount frame) in RVIZ always maintains a pitch = 0 wrt the base frame if the user is looking forward, even if the Oculus TF frame is attached to a frame that is rotated about the X or Y axes wrt to the base frame. In other words, the only way the user can "look down" in VR is by actually tilting his head down, and that effect cannot be replicated by mounting the oculus frame to a frame that is tilted down. This for some reason does not apply to the yaw (if the oculus is mounted on a frame that rotates about the zaxis wrt the base frame, the default view of the user changes).

Depth perception of the right hydra's arrow was improved by adding a translucent ball at the end of the right hydra's arrow that would be visible/invisible if it is front/behind the schilling or other objects. This also enhances the accuracy of the perception of the length of the arrow, as the ball looks smaller when it is further away.

2.4 Robot Arm Construction and Control

2.4.1 Schilling URDF File Construction

A URDF[3] of the Schilling arm was created using the following schematic and simplified CAD models of the Schilling's links. With the exception of the jaw, the simulated robot is identical to the real Schilling.

The Schilling robot arm is assumed to be position controlled in the ROS layer.

All of the Schilling joint transformations where constructed based on the above schematic except for the jaws, whose transformations were determined visually and fixed. The joints, from base to end effector, are Azimuth, Shoulder, Elbow, Wrist Pitch, Wrist Yaw, Wrist, and the Jaw joints.

---

[3]Unified Robot Description Format (URDF), which is an XML format for representing a robot model

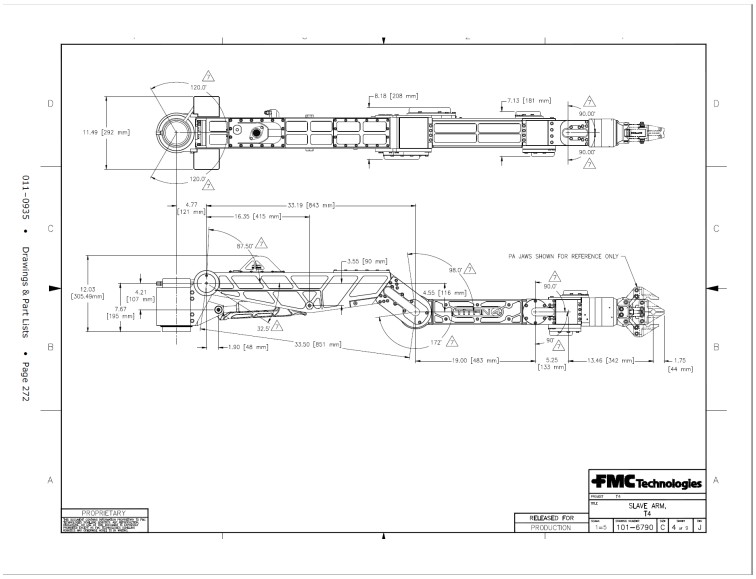


Figure 2.17: Schilling Schematic

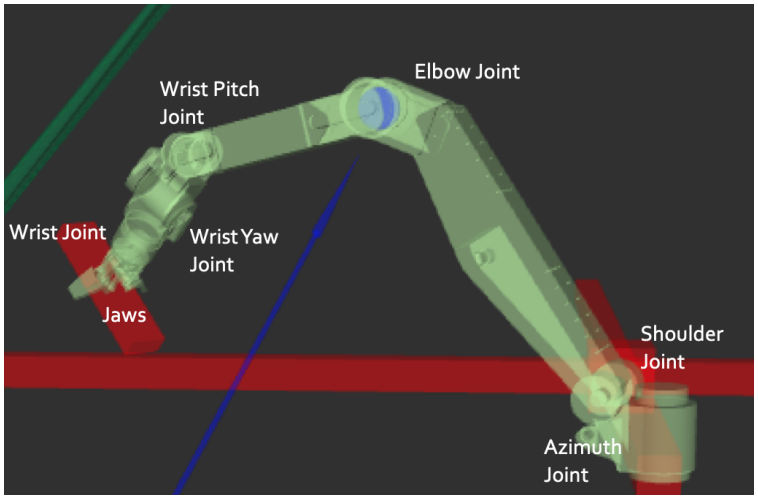


Figure 2.18: Schilling Model Visualization

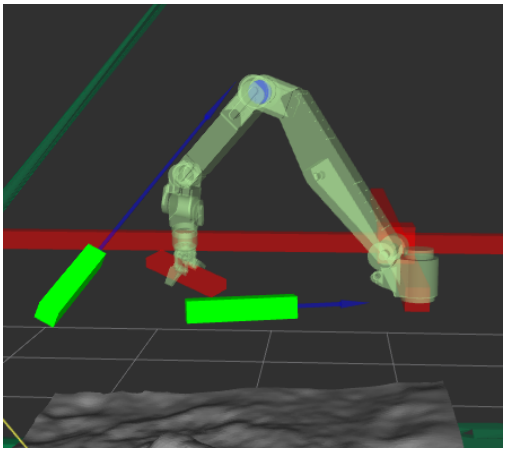


Figure 2.19: Schilling Individual Joint Control Before

2.4.2 Robot Arm Individual Joint Control Using Oculus and Hydra

To perform individual joint control/forward kinematics on the Schilling, a unique methodology was implemented. The joint closest to where the user is pointing at from the right hydra is continuously calculated and lit up with a translucent blue sphere. Once the user is pointing at the desired joint, to move the joint, the user activates the right Hydra's Trigger and rotates the hydra along its x-axis (similar to rotating a screwdriver) to intuitively rotate the joint in the simulation.

To allow for fine-grain turning, the rate of rotation is scaled by the value of the trigger which has values [0,1]. If the trigger is pressed only very lightly, then with a full rotation of the Hydra, the joint will only partially rotate.

ROS Node: schilling_trigger_control

The node subscribes to hydra_calib and joint_states topics, and listens to the tf transforms between the following frames:

- The transforms between the right_hydra_frame and all movable joints in the robot except the jaws.

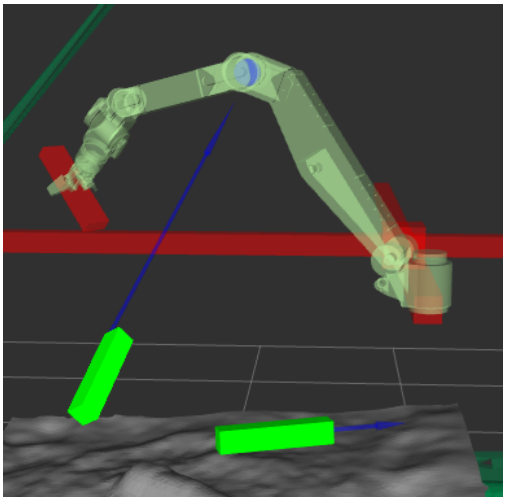


Figure 2.20: Schilling Individual Joint Control After

- The transforms between all movable joints in the robot except the jaws and right_hydra_frame (inverse of the above transforms).

Firstly, the node determines which rotary joint is the user pointing at using the right hydra's arrow representation in RVIZ. This is done by calculating the length of the Opposite with respect to the triangle that is the y and z displacements for the transform between the right_hydra_edge and the Schilling's joints. The joint who's said Opposite is the smallest is the joint closest to where the arrow is pointing.

Once the joint is determined, the node determines which side the operator is looking at the joint from, which translates to does the right_hydra_frame resolve to a point on the positive or negative zaxis of the joint, since the rotation of joints is about their z-axes. This calculation uses the rest of the transforms.

The node visualizes the joint closest to where the user is pointing at by publishing a blue sphere Marker object in real time at the joint.

The joint_states callback keeps track of the current values of all joints at all times. These values are used by the hydra_calib callback function which deals with actual movement of the joints. It first checks to see if the right trigger is pressed. If it is then it compares the

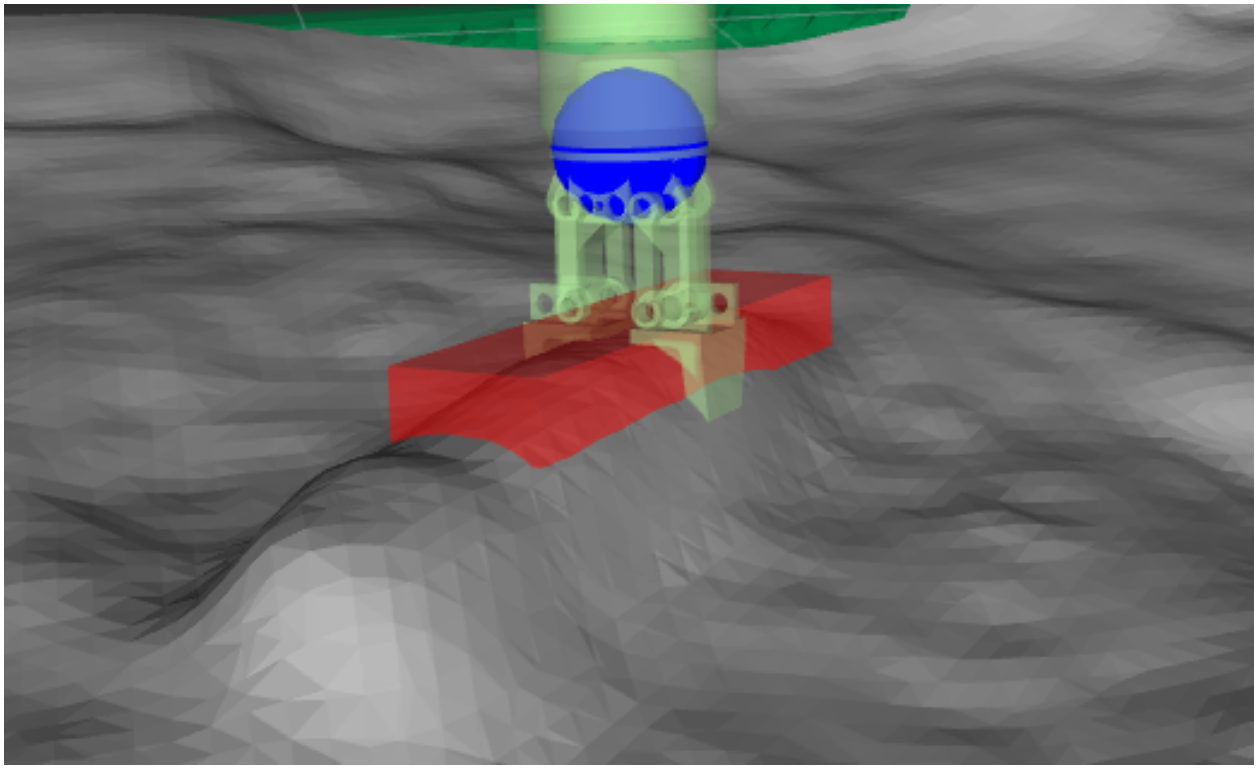


Figure 2.21: Locking Jaw to Secure Munition

difference in roll between the current state of the right hydra and the previous state when the last hydra_calib callback happened. Then it updates the joint value = current joint value + roll_difference*TriggerValue. Finally it updates the hydra previous roll state at the end of the callback for use in the next one.

ROS Node: jaw_grab

Controlling the Jaw joint was implemented in a different way: Button 1 of the left Hydra opens the Schilling's jaw and Button 2 closes it. Once the jaw is holding the munition properly (at the correct place), the user can click Button 5 of the left Hydra to lock the joint. A red rectangular cuboid appears between the jaws signifying the locked joint that moves with the end effector, and the jaws can no longer be opened or closed.

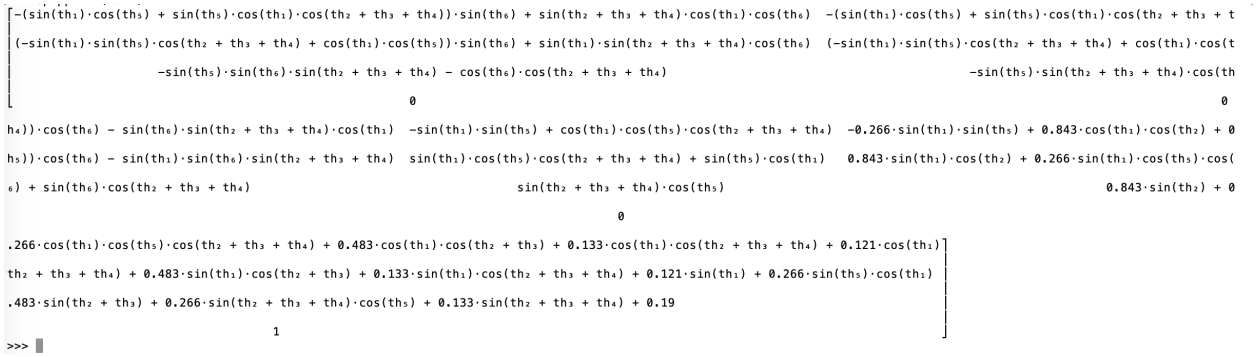


Figure 2.22: Transformation between base and end effector

2.4.3 Robot Arm Inverse Kinematic Control

Inverse Kinematic Control of the end effector was implemented in two ways. The first is "holding" the end effector using the right Hydra's edge and moving it around in virtual reality from any position/orientation in RVIZ. The second is drawing a path using the right Hydra's arrow in RVIZ and having the end effector follow it with the first point on the path being the current end effector's location. The current point being sent to be computed by the IK engine is lit up using a sphere marker as the arm is following the path. This enables the operator to visualize where solutions to the IK don't exist.

A closed form solution was attempted firstly using a python script. The transformation from a base frame to the end effector was computed symbolically and simplified using Sympy's Simplify method. However, a solution to it could not be found by hand. Here is the transformation between the base and the end effector:

IK was solved using matlab's built-in IK solver using the BroydenFletcherGoldfarbShanno (BFGS) algorithm, which is an iterative algorithm for solving unconstrained nonlinear optimization problems. The IK solver uses the robot model in the file "schilling_modified_jaw_extended.urdf" file, which is the Schilling Arm model with an additional fixed frame to model the end effector. The speed of this algorithm was fast enough (17 Hz) to allow for real time IK of the 6 DoF Schilling arm if a close initial guess is provided. This initial guess for each IK solution is

easily provided for by the current joint states of the schilling, since both IK implementations assume smooth motion along a path from start to end point (no jumping). The initial guess also leads to the matlab node to choose the solution that is closest to the current joint states of the robot when multiple solutions exist for a given pose. Moreover, if the IK computation takes too long, the matlab node does not publish the joint angle solutions. This is because when it takes too long it usually means that the solution does not exist or it is at a singularity that could partially compromises the pose and so it chooses angles that are incorrect and sometimes random.

Note that if you have the IK .m file output any kind of output to the screen within the IK loop the frequency of solving it decreases substantially, so make sure all lines of code within the loop of the Schilling_IK_ROS_Script.m file end with a semicolon to repress the output.

Direct IK: ROS Node: IK_direct_schilling_control_matlab

This implements the first method of IK: real time control of the end effector by dragging and dropping the Right Hydra in RVIZ from any position. Motion along the X, Y, and Z axes with respect to the Base frame of the right hydra's edge when the IK button (right hydra button 5) is activated was determined and the position values were sent to matlab for the IK computation using BFGSGradientProjection algorithm. The resulting joint states were published from matlab to ROS in real time, moving the robot with the same cartesian motion as the right hydra from any position. Moreover, the user can directly "grab" the end effector as if it is in his hands by moving near it and placing the right hydra's visualization at the end effector. This gives the feeling that the operator is holding the end effector with his hands.

The files associated with this are the IK_direct_schilling_control_matlab.cpp file and the IK_matlab.m file.

IK_direct_schilling_control_matlab.cpp is a ROS node that keeps track of the right razer hydra's movement changes in X,Y, and Z wrt to the base frame (which would be similar to calculating them wrt to the schilling mount frame) while the direct IK button (hy-

dramsg.Paddles(2).Buttons(6) in the matlab node) is clicked. It also publishes these cartesian displacement values as a ROS::Float64MultiArray, in addition to the current position of the end effector wrt to the schilling mount frame, to the IK_matlab topic.

The matlab IK node is continuously subscribed to that topic to compute the IK joint values and publishes them back to ROS using BFGSGradientProjection. When the IK button is clicked, the translation and orientation information is published and added to a transformation matrix to compute the IK joint angle solutions in the matlab node, which publishes back the resulting joint values as joint_states messages.

Path IK:ROS Node: IK_path_schilling_control_matlab

This implements the second method of IK: moving end effector along path that was drawn by the right Razer Hydra. The user clicks button number 1 of the right hydra (hydramsg.Paddles(2).Buttons(2) in the matlab node) to initiate the movement along the drawn path. To draw the path, the user clicks on button number 4 on the right razer hydra. An initial point is always displayed for all paths, signifying the starting position of the end effector. Then, if the button remains clicked, a path of many points, with a granularity of 1 point per 0.01 units of distance in RVIZ, is drawn in the 3D space. If the user releases the button, moves the razer hydra in the 3D space, and clicks the button again, a straight path is created that connects the current position of the hydra to the last point in the path, and the user can resume extending it. The two files involved in this functionality are the IK_path_schilling_control_matlab.cpp ROS file and the IK_ROS_script.m file.

In the IK_path_schilling_control_matlab.cpp ROS file, the X,Y, and Z coordinates of every point in the path is stored in 3 vectors; x,y,z. The first element in each of these vectors correspond to the current position of the end effector wrt to the schilling mount frame. The node continuously monitors the pathclick button of the right razer hydra and adds the points' coordinates to the three vectors (x,y,z) and also to a ROS::visualization::Points marker to display in RVIZ. Once the path creation is done, the user can begin the execution of the path by pressing button 1 of the right hydra, which publishes the coordinates to the

IK_matlab topic at a rate of 10Hz. The matlab node receives these values and computes the IK solutions.

This allows the user to change the orientation of the end effector mid path.

2.4.4 Drawing Bounding Boxes

The functionality to draw bounding boxes using the right hydra's arrow's tip in virtual reality was implemented. The right hydra and its arrow act as a "pen" using which the user draws these rectangular bounding boxes. The boxes are intended to be used as virtual fixtures in future work. The operator can draw/remove an unlimited number of bounding boxes of any size to RVIZ.

To draw a bounding box, the operator clicks the draw button. This instantly places a vertex of the bounding box at the position in 3D space of the tip of the "pen" when the draw button was clicked. It also places the opposite vertex at the current position of the tip of the "pen"; continuously following it as it moves, with the translucent bounding box constantly moving. Once the teleoperator clicks the draw button again, the second vertex' position is solidified at that position, solidifying the bounding box. The boxes are translucent, and the operator can navigate as well as increase/decrease the right hydra's arrow's length before solidifying the second vertex' position. This increases accuracy and ease of drawing, as the operator can view the bounding box as it's being drawn from any view. The operator can remove the previously drawn bounding box by clicking the remove button on the left hydra.

ROS Nodes: virtual_fixtures_coords and virtual_fixtures_markers

The virtual_fixtures_coords node initializes a tf transform listener to get the position of the tip of the right hydra's arrow. This will be the "pen" which the operator uses to draw the bounding boxes. The node subscribes to the hydra_calib topic to detect the drawing/removing boxes button clicks. The node publishes to the "virtual_fixture_coords" a vector that contains all the xyz positions of all bounding boxes that currently exist. The virtual_fixtures_markers node subscribes to the virtual_fixture_coords topic and visualizes the

bounding boxes in real time as translucent red Marker Cube objects by publishing them as a Marker Array.

2.5 Communication

2.5.1 Razer Hydra ROS node

aleeper's ROS package for razer hydra motion controller was used to read in real time the state of the Hydra's joysticks, triggers, bumpers, and buttons. The package publishes the X and Y values [0,1] representing the state of the joysticks, and a value [0,1] representing the state of the left and right triggers. The rest of the buttons and bumpers are published as binary on/off.

2.5.2 RVIZ Environment Projected to Oculus

To visualize the simulated environment in Oculus in real time, a plugin, Oculus-rviz-plugins, developed by OSU-Robotics, was installed[3]. This allows the teleoperator to view the virtual environment in 3D from any angle. By default, the Oculus is not attached to any coordinate frame. The user can attach it to any available tf frame desired by choosing the frame from a drop down menu in RVIZ. Furthermore, the plugin creates a ROS node that publishes the roll, pitch, and yaw of the Oculus in real time. Before launching the plugin (rosrun oculus_ovr_sdk ovrd), the user must have the Oculus Headset in front of him, facing downwards (perpendicular to the floor), and not rotated:

ROS Node and package: oculus_ovr_sdk

"A ROS package which builds the Oculus SDK as a shared library and outputs a link to the appropriate includes/libraries that other packages can use." [16]. It publishes the oriantation of the Oculus to the specified topic. Running and terminating the node produces the output shown in the two figures on my machine.

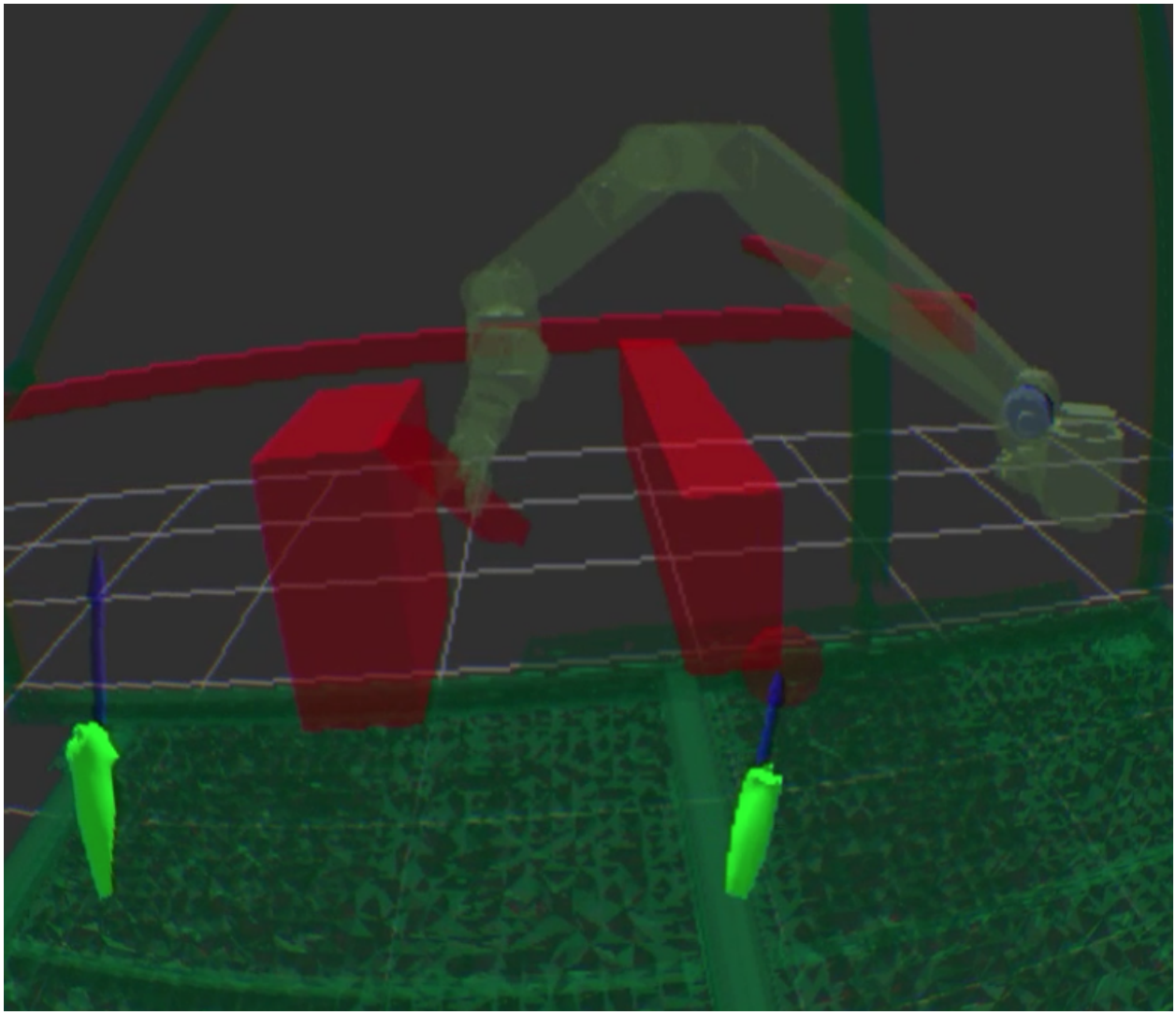


Figure 2.23: Bounding Boxes

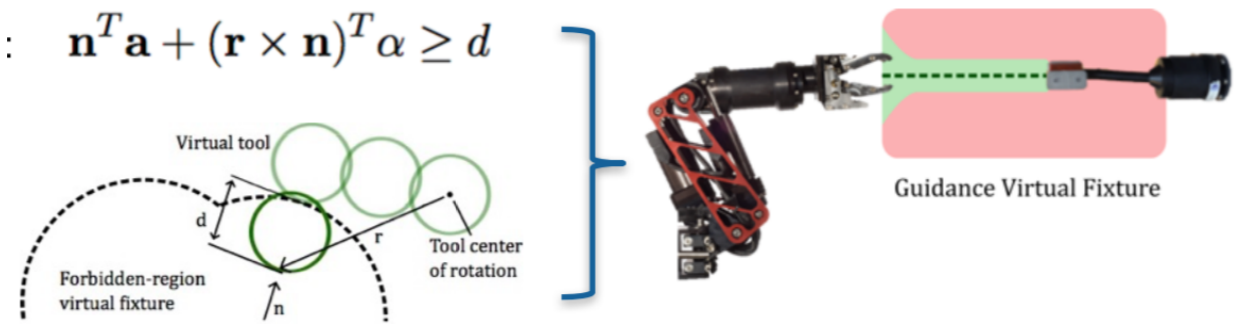


Figure 2.24: Virtual Fixtures and Haptic Feedback

[23]

ROS package: oculus_rviz_plugins

This package should be built in the ROS workspace in addition to the oculus_ovr_sdk package. It enables the Oculus view in RVIZ.

2.5.3 Communication With Hardware: Control

The IP address of the machine hosting the ROS-Simulation was made static to allow an ssh connection request from the machine that is interfacing with the Schilling hardware over the internet. The hardware interfacing machine then had access to the stream of the Schilling joint values by echoing the schilling_stream topic using the following command:

rostopic echo /schilling_stream

That machine would then parse the stream output of the topic and redirect it to control the schilling.

A safety mechanism is implemented in the schilling_stream node such that it will not publish joint values of the schilling to the stream if any of the joints move too fast (or jump) in the simulation. Instead, an error message, "ERROR: joint state maximum step value exceeded" will be displayed in the visualization, and the stream stops. Jumps in the schilling's joints' values occur when the IK solver encounters singularities or when the simulated Schilling's end effector traverses a path, using inverse kinematics, that goes outside

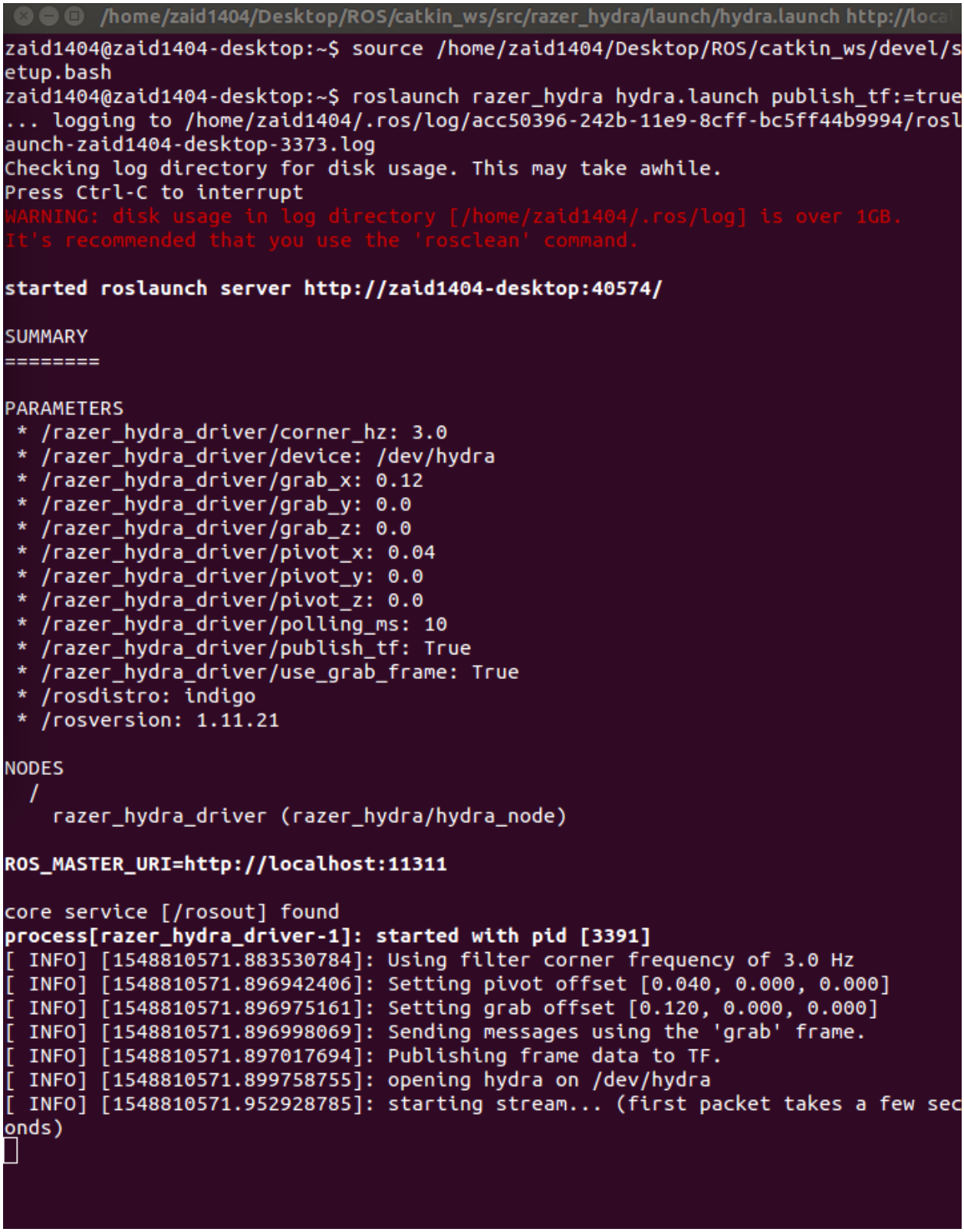


Figure 2.25: Terminal Info Running Hydra Node

Figure 2.26: Orientation of Oculus before Launching Oculus Node

[17]

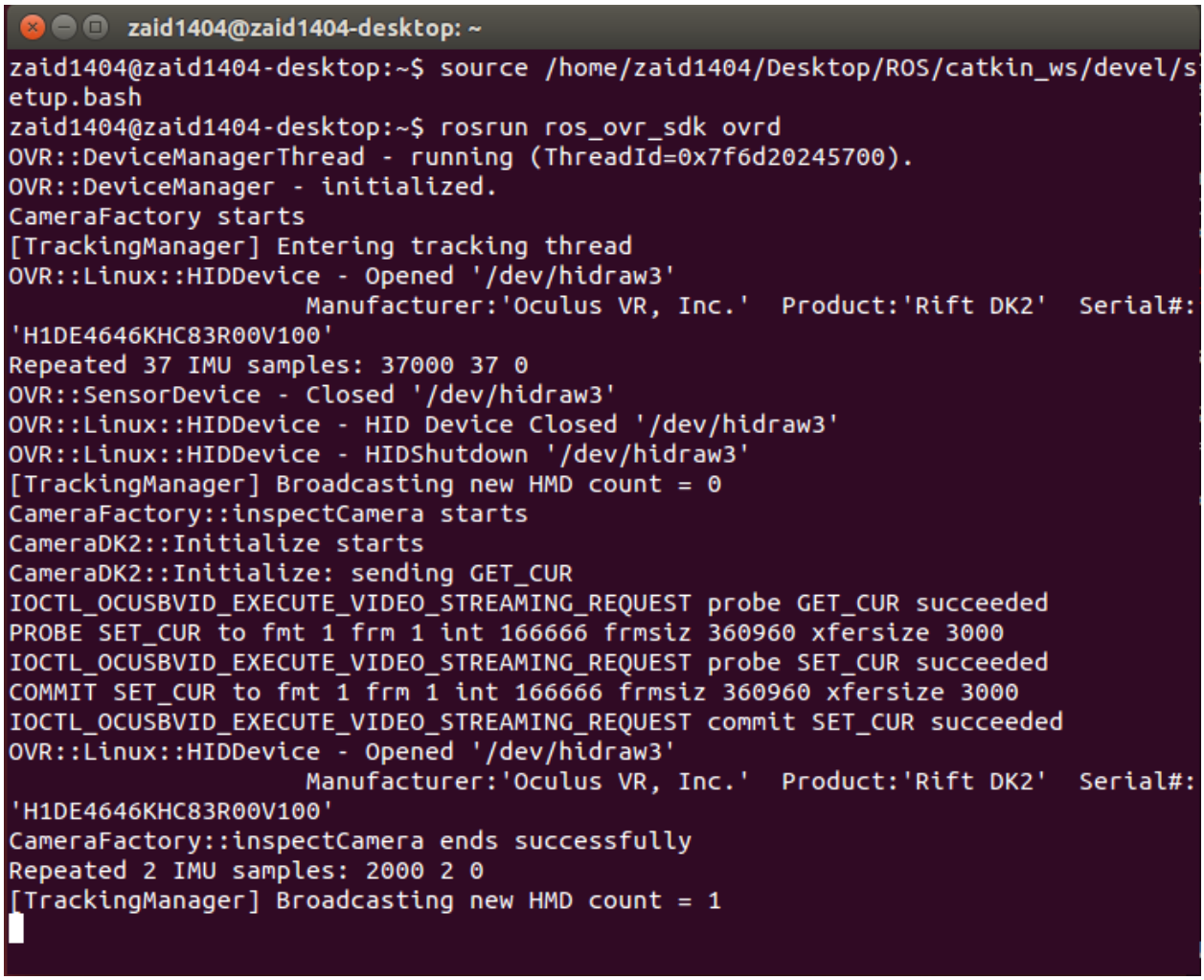


Figure 2.27: Running oculus_ovr_sdk

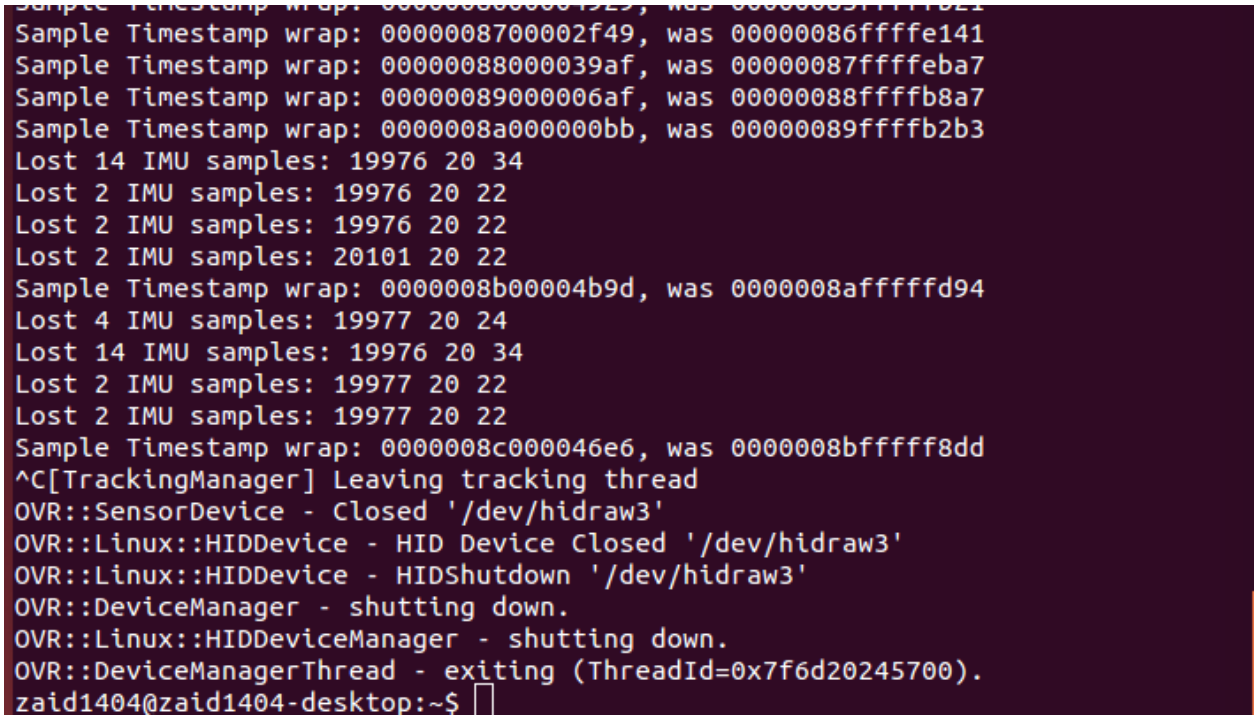


Figure 2.28: Shutting Down oculus_ovr_sdk

of its workspace and comes back in at a different point. The max jump threshold value is modifyable in the schilling_stream.cpp file. When a max jump error occurs, it results in the real schilling being out of sync with the simulated schilling during the teleoperation.

To reset the max jump error and when setting/resetting the joint values of the simulated schilling to match the real schilling's state, the hardware-interfacing machine will need to retrieve the joint values of the real schilling (or whatever desired values) and run this command in the ROS-machine command line:

rostopic pub -1 /schilling_set sensor_msgs/JointState 'header: auto, name:
['schilling_mount_to_azimuth', 'ShoulderJoint', 'ElbowJoint', 'WristPitchJoint',
'WristyawJoint', 'Jaw0_to_endeffector'], position: [a,b,c,d,e,f], velocity: [], effort: []'
where a,b,c,d,e,f are the desired joint states.

This sets/resets the state of the simulated schilling and remove any max jump error if present.

ROS Node: schilling_stream

This node parses the joint state publisher to get just the joint states (joint values) of the schilling, and publishes them to the "schilling_stream" topic. It also keeps track of the difference between each joint angle from the current published message and the previous published message. If the difference is larger than the max jump threshold for any joint, it issues an error message which appears as very large text displayed in RVIZ above the Schilling, and stops the stream until the joints are reset using the rostopic pub method described above, at which point it resumes the stream.

The max jump threshold value is modifiable in the schilling_stream.cpp file.

2.5.4 Communication With Hardware: Visualization

To visualize the LiDAR point cloud in RVIZ, the LiDAR file is sent over the internet to the ROS machine, and the lidar_publish ros node is run with an argument of the name of the file:

rosrun oculus_hydra_rviz_teleoperation lidar_publish

"package://oculus_hydra_rviz_teleoperation/mesh/meshes/NAME-OF-FILE.stl"

2.6 Gantry Dynamics and Control

A simple 2D Cartesian Robot (X,Y) Gantry was modelled and visualized in RVIZ and Simulink. It serves a simple purpose: bringing the robot arm to an optimal position near the munition to conduct the remediation. It is assumed to be force controlled. To enable position control of the gantry, a model of the underwater dynamics was created in simulink. The motion of the gantry's visualization in RVIZ is constrained by these dynamics.

2.6.1 Underwater Dynamics in Simulink

The underwater environment's dynamics are implemented in a Simulink model as part of a closed loop control system to control the XY Cartesian robot's position (gantry). It is assumed that axes are decoupled., so two separate closed loop control systems were implemented, one for the x-axis and one for the y-axis.

Firsty, the underwater environment creates a damping force on the robotic system proportional to the velocity. Secondly, the underwater environment creates a force due to underwater currents and water movements. These can be viewed as two distinct phenomena: 1) Low force random (with respect to the magnitude and direction) time-varying currents modelled as white noise for both the x and y axes of the gantry (r(t)) and 2) A stable current c(t) with a specific force and direction whose values remain fairly constant over time, and this was modelled as two non-time-varying uniform random variables with respect to the X and Y axes of the gantry (this assumes the general direction of the currents don't change within a single dismantling/remediation session and that the changing shape of the robot does not contribute to a large change of the force, which is false) and the values are sampled at the start of the Simulink simulation. The last force on the gantry is the engine force e(t).

It is assumed that the Cartesian robot uses a motor who's force we can control directly.

Therefore, the equation that models the underwater dynamics of our system is:

$$e(t) + c(t) + r(t) = ma(t) + bv(t)$$

And in the laplace domain:

$$E(s) + C(s) + R(s) = m * X(s) * s^2 + b * X(s) * s$$

$$F(s) = m * X(s) * s^2 + b * X(s) * s$$

$$X(s)/F(s) = G_p(s) = 1/(ms^2 + bs)$$

A digital PID controller was implemented in Simulink to achieve low overshoot, zero steady state error (assuming no white noise), and quick response.

Initial conditions are assumed to be zero since the gantry, upon reaching the next point in the path, stops. This prevents error from accumulating as the gantry moves along the points in the path.

Because we don't have the exact measurements for the drag coefficient and reference area of the robotic system, placeholders have been left in the simulink model's underwater dynamics block until such data is acquired. A placeholder was also left for the gain of the motors of our Cartesian robot until they are determined.

Here is the closed loop control system of the Cartesian robot for just one axis. Since the white noise disturbance is just a model, the same white noise Simulink block can be applied to both the x and y axes of the gantry. However, since C(t) is direction dependent, it will resolve to two not necessarily equal values for the x-axis and y-axis, so two non-time-varying random variables were sampled from a uniform distribution.

Simulink Node: gantry_dynamics_control

Matlab's Robotics Systems Toolbox includes a pipe for communicating with ROS Indigo. The pipe works on both standard Matlab functions as well as simulink blocks. The Simulink

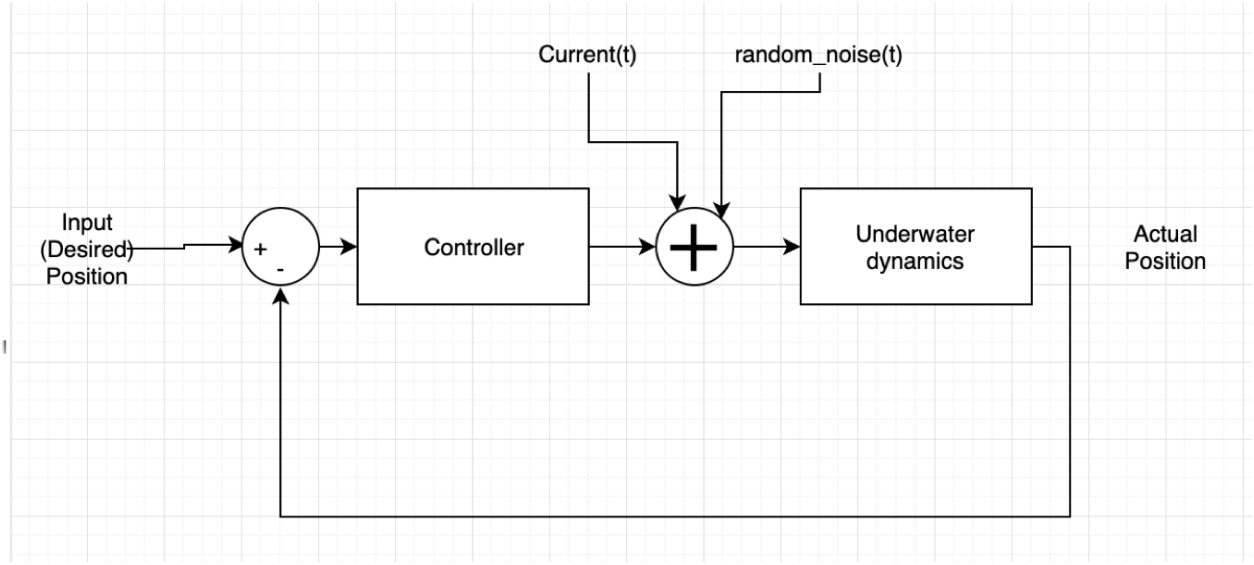


Figure 2.29: Closed loop control system wrt one axis

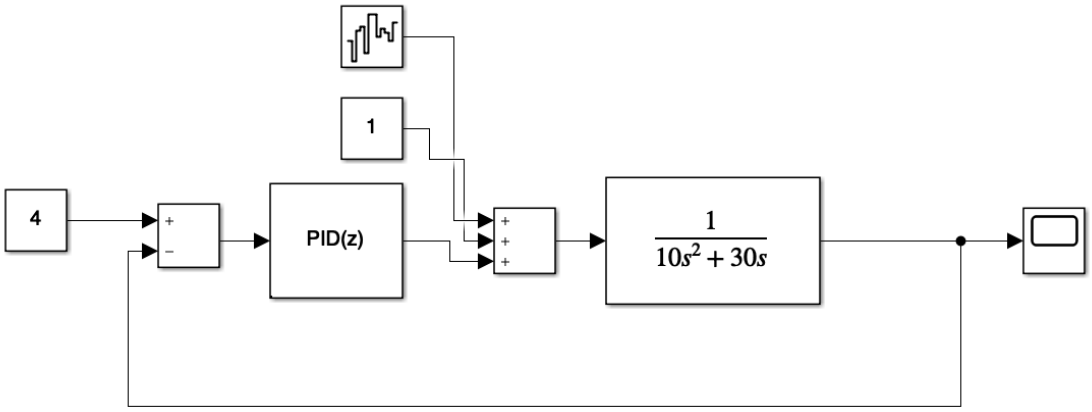


Figure 2.30: Simulink feedback loop

[17]

file consists of functionality that implements gantry motion planning in 2 modes of operation: Direct and Path. The variable that detects that is directOrMarker. In the Direct method, the gantry is directly operated by the Razer Hydra's joystick/keyboard's arrow keys. In the Path mode, a path consisting of multiple points is drawn and the motion begins executing when the engine is activated (via engine enabler command).

Based on the mode, the X and Y arrays are filled with values, either of the path (each point in the path corresponds to 1 value in the X and Y arrays at the same index), or of the next point (in direct mode), at which all the values would be of the next point. Simulink arrays are static sized, so all the indices after the last point's index are filled with the value of the last point. The marker array path points come from the markerArray ROS topic. The direct motion point comes from the goToPublisher ROS topic.

Once the X and Y values are determined, the subsystem in simulink executes the motion control. The counter i keeps track of which index along the path is the gantry currently going towards. Another variable, StoppedPos, keeps track of the current position of the gantry in case the user disables the engine to stop the motion during execution at any time.

During execution, an X and Y coordinate are fed to each feedback control loop and when the gantry reaches these coordinates, the i index is incremented and the next set of coordinates are fed. The stopping condition (the point at which i is incremented) is met only when these two constraints are met for both the X and Y loops: the distance between the desired location and actual location is less than 0.05, and the gradient of the distance (velocity) when the distance is less than 0.05 has been positive at least twice and negative at least twice. This is to ensure that the system stabilized at the current coordinates before moving on to the next one.

simulink_To_ROS_message_parser

This node takes the output of the Simulink node (x,y positions of the gantry) and creates a joint state message with these values and publishes them to the joint_states topic to move the gantry.

*2.6.2 Marker Publish in RVIZ to Display Path *change to just gantry visualization*

RVIZ's Marker Publish functionality is used to implement the Gantry's path planning. Path planning of the gantry was implemented using the Right Hydra's joystick. The displacement between the points is scaled by the magnitude of the joystick value, so the line segments connecting each 2 consecutive points may not have equal length.

Note that in the ROS nodes that deal with creating paths using markers for the gantry, the number of markers published is constant, and all the extra points are placed at the location of the last point in the path e.g. if the path is just one straight line from the current position to another, one point will be published at the current path and all the rest will be published at the destination. This doesn't cause any problems neither visually nor with execution. The reason for this implementation is that Simulink assumes a constant size of its array blocks that subscribe to ROS. This constant can be changed, but one must make sure that the same value is used in both the ROS line marker publishing nodes for the gantry as well as the Simulink file.

ROS Node: joystick_marker_pub

This ROS node is used to create the path to be taken by the gantry. The node assumes a constant number of points that describe the path (20), subscribes to the hydra_calib topic, and publishes to the markerArray topic. The node assumes that the following node (gantry_path_visualizer) is running. Once this node is run, it creates the path by listening to the values of the joystick and modifying an array that contains the X,Y coordinates of the path and publishes the array as a visualization_msg and displays the path as it updates it. Simulink, as well as the gantry_path_visualizer node receive this array, with simulink executing the path and points_and_lines2 displaying it.

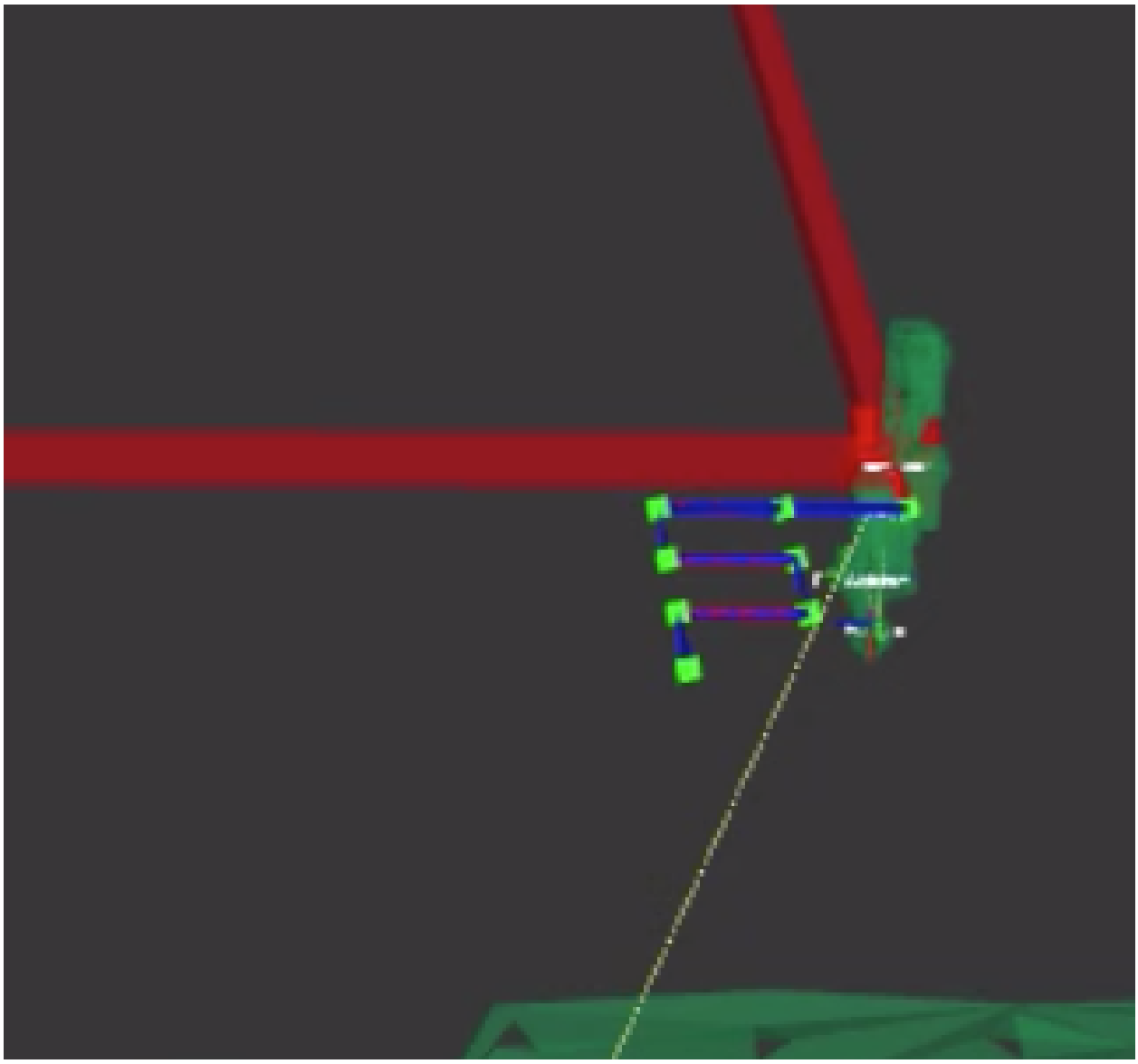


Figure 2.31: Path of gantry using keyboard's arrow keys

ROS Node: arrow_keys_path_pub

This implements the same functions as the joystick_marker_pub but using the arrow keys of the keyboard.

ROS Node: gantry_path_visualizer

This node receives the array that contains the XY coordinates of the points along the path from the markerArray topic and creates point markers at the desired coordinates as well as line markers that connects these points, and publishes them to the visualization_msgs topic, which is the default topic that displays markers in RVIZ.

2.6.3 Direct Control of Gantry

In addition to path planning of the gantry using markers, a way to directly move the gantry was implemented using the Hydra's joystick and keyboard's arrow keys.

ROS Node: arrow_keys_direct_pub

This allows for the control of the gantry using the arrow keys of the keyboard directly (without creating prior path). It listens to the arrow keys' values and publishes the next point to Simulink through the goToPublisher.

ROS Node: joystick_direct_pub

This allows for the control of the gantry using the Hydra's joystick directly (without creating prior path). It listens to the joystick's values and publishes the next point to Simulink through the goToPublisher.

ROS Node: gantry_control_modes

This node keeps track of the mode of control -direct or following path. It also keeps track of whether the engine is on. It sends these values to Simulink by publishing to the /director-

marker topic.

2.7 Hydra Controls

The controls are:

- Translation using right bumper
- Zoom in/out using right and left bumpers simultaneously
- Rotation using left bumper
- Reset view using left joystick button and right joystick buttons simultaneously
- increasing right hydra's arrow's length using right joystick (move up to increase down to decrease)
- Forward Kinematic control using right trigger
- IK control using right button 0 (center button)
- Adding points for the path using right button 4, reset using right button 3
- Drawing Bounding boxes using right button 1, removing using right button 2.
- Gantry drawing path/direct control using right joystick or keyboard arrow keys.
- Gantry engine enabler using right button 1
- Gantry control mode (path vs direct) right Button 2

Chapter 3

RESULTS

3.1 rqt_graph with and without gantry

rqt_graph provides a GUI plugin for visualizing the ROS computation graph.[1] For reference, the ROS computation graph for the two cases are shown: case 1) gantry not loaded and 2) gantry loaded.

3.2 Performance of RVIZ Virtualization in Oculus

Using the rviz to Oculus plugin, the RVIZ environment was visualized on the Oculus DK2. The resolution of the simulation was as sharp and only limited by the resolution of the DK2 itself. The rate at which oculus orientation data is published is 60Hz, and based on my user experience, was accurate.

3.3 Performance of Razer Hydra in RVIZ

Using the Razer Hydra node, the location and orientation data of the Hydras were published in real-time in ROS at 245Hz. The position and orientation values themselves were accurate based on my user experience. For example, if the two Hydra's are brought together then they also meet perfectly next to each other in RVIZ. However, there are some instabilities that cause the position/orientation values to jump to a random value if the teleoperator had the Hydras outside of the working range. Because of this, the teleoperator should ensure that his arms are positioned in the middle of the workable range where this does not happen. The user's arms should not be below a table with respect to the Hydra's orb. Experimentation can be done by the teleoperator to determine the best position. I found it to be best when

[1] http://wiki.ros.org/rqt_graph

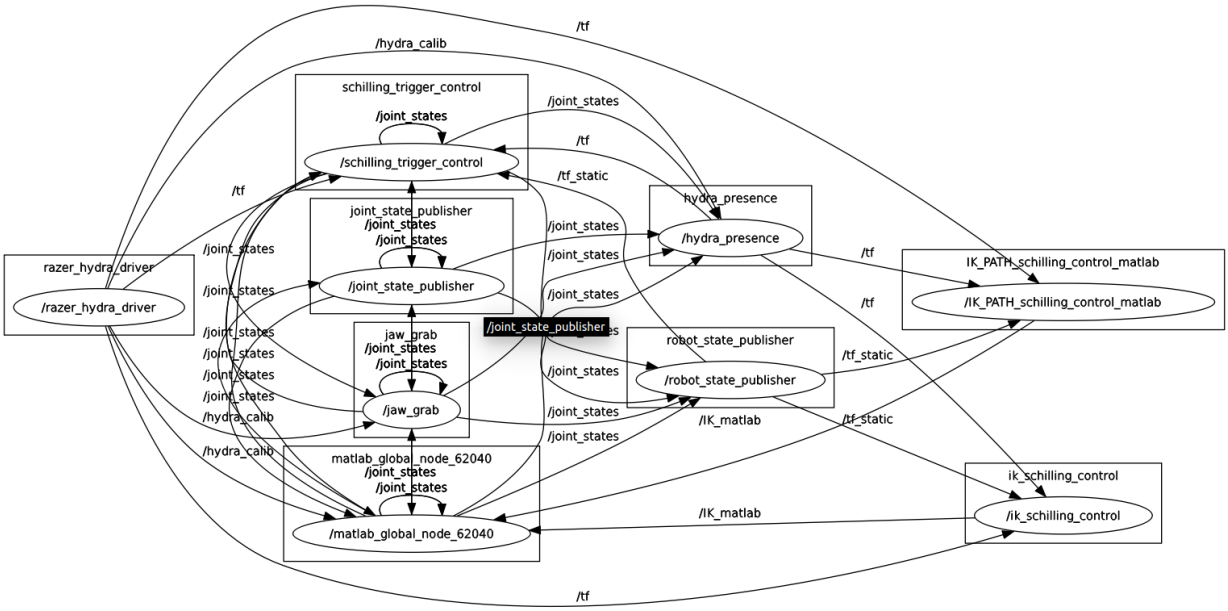


Figure 3.1: rqt_graph without gantry components

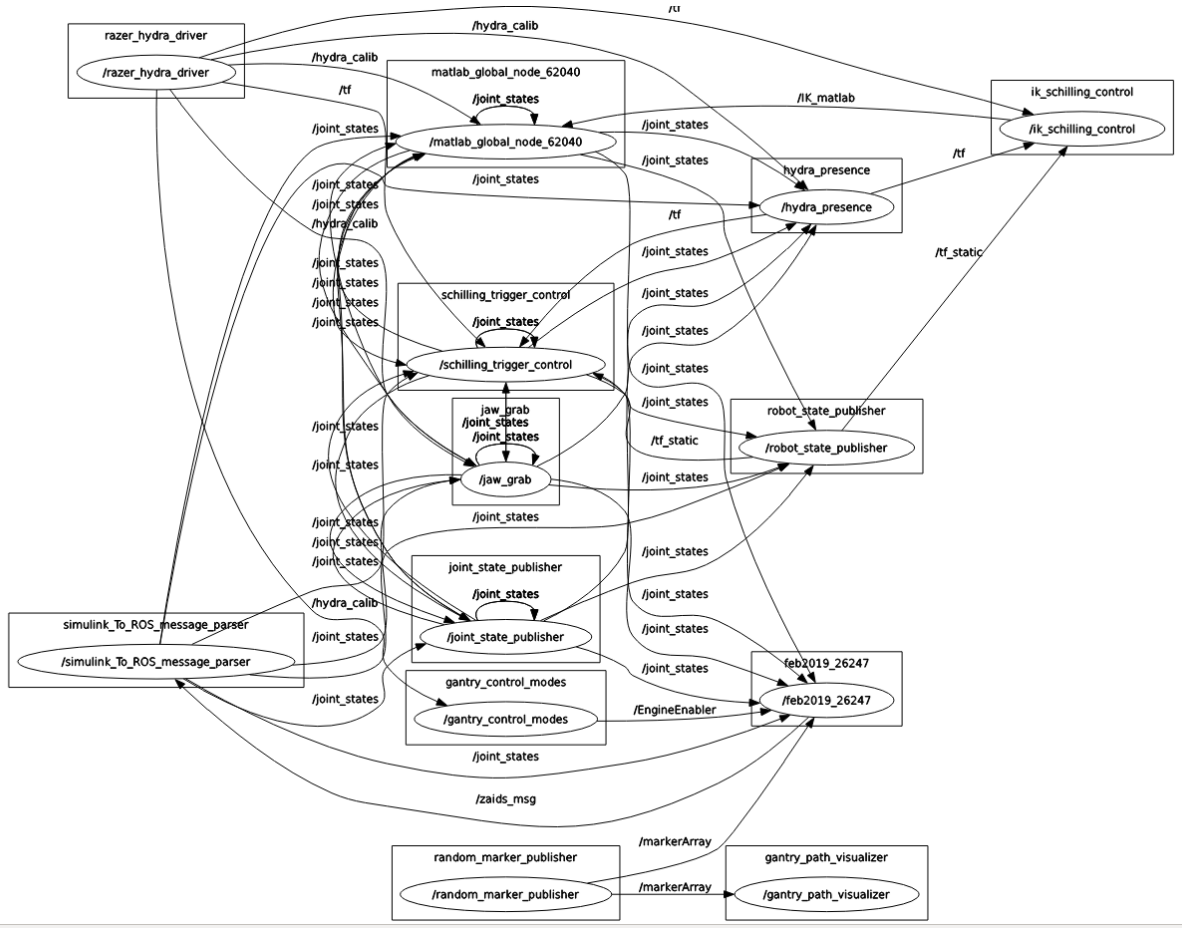


Figure 3.2: rqt_graph with gantry and running all features

standing up, with a slight bend in the elbows, and with the orb on a table in front of the user. Moreover, the Razer Hydra visualizations' positions' offsets with respect to the headset can be modified in the hydra_presence file to move the controllers to the most natural position for the teleoperator.

3.4 Performance of Navigation Using Razer Hydra in RVIZ

The navigation implementation was described as "intuitive" by demo participants. However, first time users seemed to struggle, partially because there are numerous controls, but also because they lacked experience in VR. However, a seasoned user will be able to seamlessly navigate the virtual environment using either zoom,rotation,translation implementation.

3.5 Performance of Control Using Razer Hydra in RVIZ

All control methods were designed to leverage humans' natural ability to manipulate objects in 3D space and were made to be as mentally untaxing as possible, while still delivering robust performance. Control functionalities were also described as "intuitive" by inexperienced demo participants.

3.5.1 Performance of IK Solver in Matlab

IK solutions are computed fast enough to allow for smooth and responsive movement of the end effector in the Cartesian space. Solutions were computed at a rate of 17.65Hz using the BFGS Algorithm, and 18.5Hz using LevenbergMarquardt algorithm.

3.6 Generalizability of ROS Package to Other Applications

The Navigation component of this work is generalizable to any RVIZ simulated environment for any VR telerobotic application.

By modifying the Robot Arm model in the URDF files, one could use this ROS package for the teleoperation of any 6DoF 6-joint robot arm. The "schilling_gantry" (URDF file containing everything that is loaded into RVIZ), "schilling_arm_modified_jaw_extended"

(URDF file used by Matlab node for Inverse Kinematics), and "schilling_detached_gantry" (URDF file containing everything except gantry that is loaded into RVIZ) URDF files should be modified to model the desired robot arm. The names of the joints (azimuth, shoulder, elbow, etc..) should be kept the same, however, as several ROS nodes parse these names. This allows the entire package to be generalizable to any VR telerobotic application with a 6DoF 6-joint Robot.

3.7 Underwater Dynamics Simulink

With the use of underwater sensors and encoders, the system can fill in the value of the disturbances in the Simulink model. And with analysis of the shape of the gantry and the robot arm, an estimate of the drag coefficient could be derived and integrated in the Simulink model to successfully control the gantry. A "lock" on the gantry would also be useful for when it reaches the perfect location to conduct the remediation.

Chapter 4

CONCLUSIONS

Virtual Reality telerobotic control using off the shelf VR hardware implemented on ROS and Matlab/Simulink delivers effortless and more robust control compared to traditional teleoperation approaches. A ROS package was developed for the VR telerobotic control of a 6DoF 6-joint robot arm using the Oculus DK2 and Razer Hydra.

4.1 Future Work

4.1.1 IK Integration With Gantry

So far the IK node only computes the Schilling Joints. But this could be extended to include the gantry forming an 8 DoF robot, significantly increasing the robot workspace. The Gantry may come into effect when a singularity is reached at the workspace boundary with respect to the 6 DoF.

4.1.2 Integration with more advanced hardware such as the Oculus CV1 and the Oculus Touch

The Oculus Touch and CV1 are only compatible with Windows. If one could utilize a low latency and high thoroughput windows-linux pipe, either on the same machine running both OS's simultaneously, or on different machines, to visualize RVIZ/Ros elements in windows, while streaming the Oculus Headset's and Touch's position and orientation data to linux, the operator would get an improved display and better, more responsive controllers.

4.1.3 Haptic Feedback

Haptic feedback as well as force feedback could be implemented in many circumstances such as:

- When IK control hits a singularity
- With guidance virtual fixtures
- When collisions occur with bounding boxes

4.1.4 Partial or complete automation of the remediation process

Creating a model that automates some or all aspects of the remediation process is a big next step for this project.

4.1.5 AR implementation

Implementations for Augmented Reality headsets such as the Hololens or Magic leap may deliver advantages over the fully immersive VR experience of the Oculus.

4.1.6 Complete Generalizability to Teleoperate any Robotic System

The nodes in this package could be modified to make it fully robot agnostic.

BIBLIOGRAPHY

[1] https://upload.wikimedia.org/wikipedia/commons/e/ee/Reality_check_ESA384313.jpg.

[2] http://www.math.union.edu/~dpvc/talks/2000-11-23.MTCM/robot.html.

[3] Aleeper. razer_hydra ros node. https://github.com/aleeper/razer_hydra, 2019 (accessed April 7, 2019).

[4] Jose-Luis Blanco. A tutorial on se (3) transformation parameterizations and on-manifold optimization, 2019 (accessed April 7, 2019).

[5] Deep ROV cursos. Titan 4 manipulator. https://www.youtube.com/watch?v=ZaQLyIvC5aw, 2019 (accessed April 7, 2019).

[6] Elizabeth Phillips George Konidaris David Whitney, Eric Rosen and Stefanie Tellex. Comparing robot grasping teleoperation across desktop and virtual reality with ros reality., 2017.

[7] ECA Group. subsea manipulator arms. https://www.ecagroup.com/en/solutions/subsea-electrical-manipulator-arms, 2019 (accessed April 7, 2019).

[8] Daniela Rus Jeffrey I Lipton, Aidan J Fay. Baxter's homunculus: Virtual reality spaces for teleoperation in manufacturing, 2017.

[9] Mathworks. Ros support. https://www.mathworks.com/hardware-support/robot-operating-system.html, 2019 (accessed April 7, 2019).

[10] Mathworks. Simulink. https://www.mathworks.com/products/simulink.html, 2019 (accessed April 7, 2019).

[11] Mariacarla Memeo. Teleoperation.

[12] OSUrobotics. ros_ovr_sdk. https://github.com/OSUrobotics/ros_ovr_sdk, 2019 (accessed April 7, 2019).

[13] Razer. Razer hydra guide. https://dl.razerzone.com/master-guides/Hydra/HydraOMG-ENG.pdf, 2019 (accessed April 7, 2019).

[14] Razer. Razer hydra portal 2 bundle. https://www2.razer.com/au-en/gaming-controllers/razer-hydra-portal-2-bundle, 2019 (accessed April 7, 2019).

[15] Roadtoovr. Reactive grip brings tactile feedback to the razer hydra, other motion input devices [video]. https://www.roadtovr.com/tactical-haptics-reactive-grip-razer-hydra-gdc-2013/, 2019 (accessed April 7, 2019).

[16] OSU Robotics. Oculus rviz plugins. https://github.com/OSUrobotics/oculus_rviz_plugins, 2019 (accessed April 7, 2019).

[17] ROS. Displaytypes/marker. http://wiki.ros.org/rviz/DisplayTypes/Marker, 2019 (accessed April 7, 2019).

[18] ROS. Ros home page. http://www.ros.org/, 2019 (accessed April 7, 2019).

[19] ROS. Rviz. https://github.com/ros-visualization/rviz, 2019 (accessed April 7, 2019).

[20] ROS. tf. http://wiki.ros.org/tf, 2019 (accessed April 7, 2019).

[21] SERDP. Munitions underwater. www.serdp-estcp.org/Program-Areas/Munitions-Response/Munitions-Underwater, 2019 (accessed April 7, 2019).

[22] serdp estcp. Munitions in the underwater environment. https://www.serdp-estcp.org/Featured-Initiatives/Munitions-Response-Initiatives/Munitions-in-the-Underwater-Environment, 2019 (accessed April 7, 2019).

[23] Andy Stewart. Augmennted co-robotics for remediation of military munitions underwater, dr. andrew stewart university of washington applied physics laboratory brief to the scientific advisory board, 2019 (accessed April 7, 2019).

[24] Owen Jow1 Dennis Lee1 Xi Chen Ken Goldberg1 Pieter Abbeel Tianhao Zhang, Zoe McCarthy. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation, 2018.

[25] Jean Vertut. Teleoperation and robotics: Applications and technology. vol. 3. Springer Science Business Media.

Appendix A

WHERE TO FIND THE FILES

- Github.

  `https://github.com/zaid-g/ROS_oculus_hydra_teleoperation`