Individual Lab Report - 4

Autonomous Reaming for Total Hip Replacement

IPSTER | ARTHuR

Sundaram Seivur

Team C:

Kaushik Balasundar | Parker Hill | Anthony Kyu
Sundaram Seivur | Gunjan Sethi

March 23rd 2022

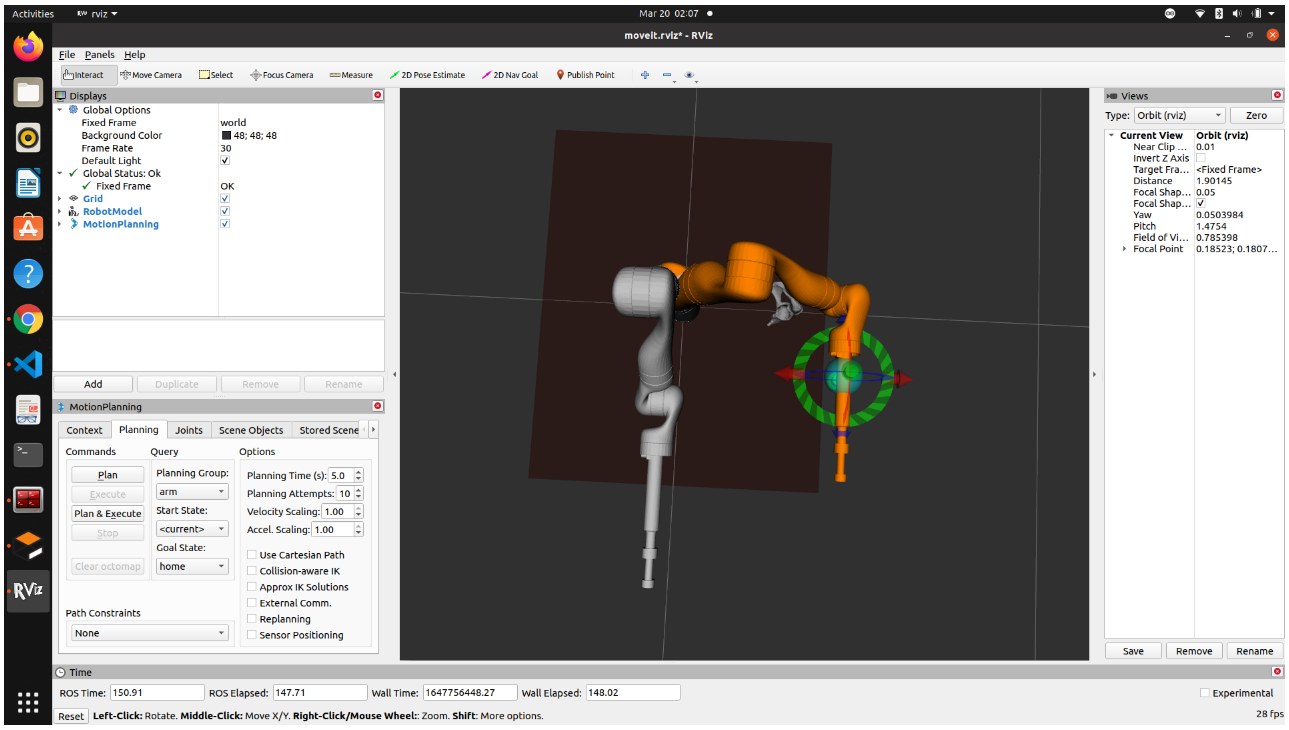
Carnegie Mellon University
The Robotics Institute

Contents

1 Individual Progress

The last few weeks after progress review 2 have been hectic yet satisfying.

I have made progress in the trajectory planning subsystem. I started with connecting the Gen3 with a computer via the ethernet and configured the IP for the computer to connect with the arm. Once I set up the arm, I validated if I was able to read the joint position, velocity and accelerations and also send back commands to the arm via the KORTEX API which is Kinova's Web App.
I worked on generating an IKFast Plugin for the arm. This was not very straight forward as there wasn't enough documentation on how to do this irrespective of the manufacturer, and the available documentation was outdated. This needed me to set up an docker image with ROS Indigo and Ubuntu 16.04, which was the last available documentation version. I had to run this multiple times and run into errors to understand the intricacies to set up this plugin for the Gen3 arm.
I majorly worked on setting up the Pilz Industrial motion planner in MoveIt!. We decided not to use the deafult Open Motion Planning Library (OMPL) as our primary use for MoveIt! is to generate trajectories and not do collision avoidance. To do this, I was able to install the planner easily but to configure the planning pipeline in the Kinova ROS code was challenging. I had to make changes to the Move Group files to load Pilz as a motion planner instead of OMPL and change the Kinova Launch files to load from the pilz planning pipeline.



Global Options
Fixed Frame: world
Background Color: 48; 48; 48
Frame Rate: 30
Default Light: ✓
Global Status: Ok
Fixed Frame: OK
Grid: ✓
RobotModel: ✓
MotionPlanning: ✓

Planning Group: arm
Start State: <current>
Goal State: home
Path Constraints: None
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 1.00
Accel. Scaling: 1.00
Use Cartesian Path
Collision-aware IK
Approx IK Solutions
External Comm.
Replanning
Sensor Positioning

Type: Orbit (rviz)
Current View: Orbit (rviz)
Near Clip ...: 0.01
Invert Z Axis
Target Fra...: <Fixed Frame>
Distance: 1.90145
Focal Shap...: 0.05
Focal Shap...: ✓
Yaw: 0.0503984
Pitch: 1.4754
Field of Vi...: 0.785398
Focal Point: 0.18523; 0.1807...

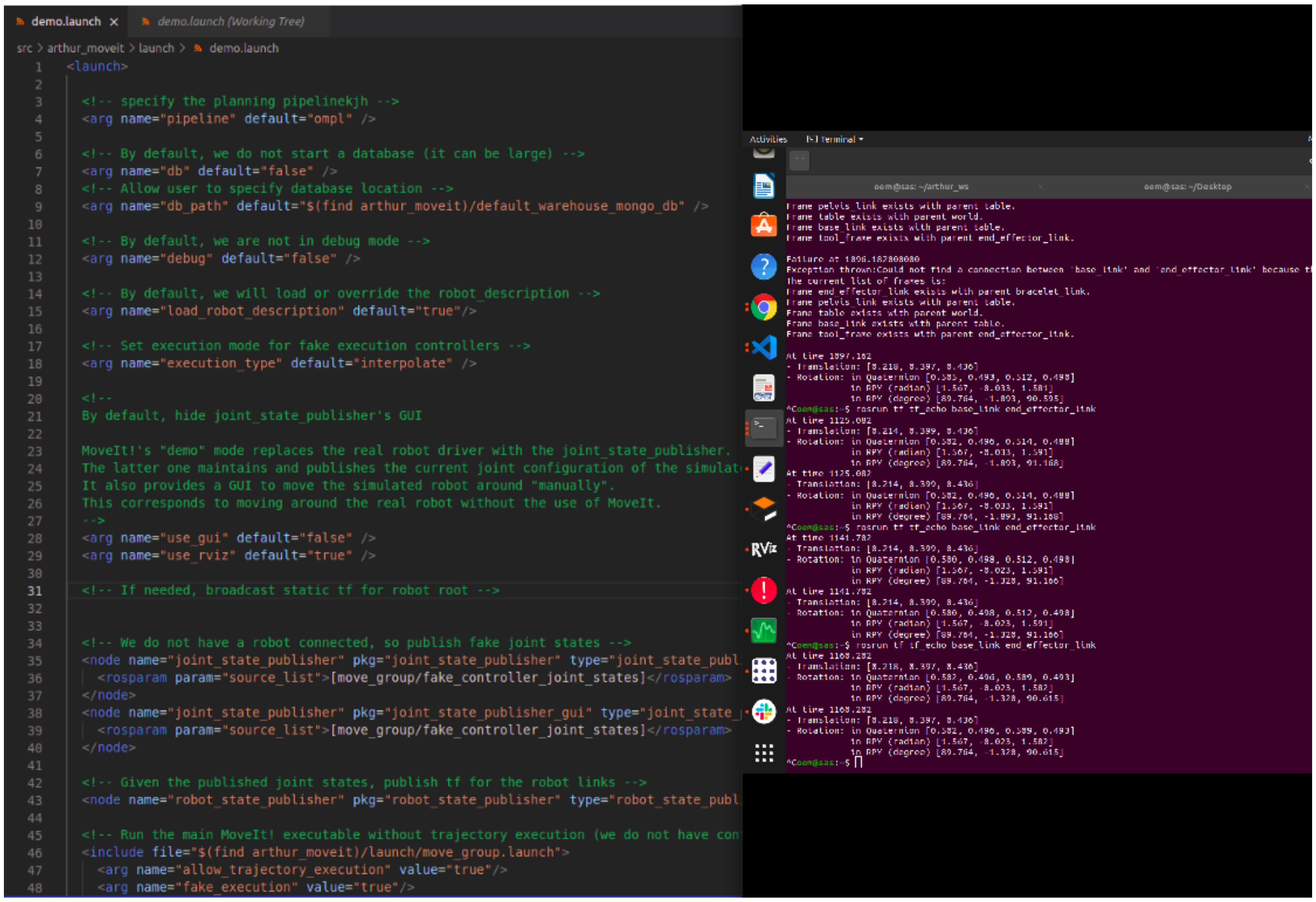
ROS Time: 150.91 ROS Elapsed: 147.71 Wall Time: 1647756448.27 Wall Elapsed: 148.02

Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options. 28 fps

Figure 1: Pilz Industrial Motion Planner

With all these developments, I wrote a script to send commands to the arm for basic motions such as go to home position, reach a particular joint state, reach a cartesian point etc., all of this using both OMPL and Pilz planner. I wrote the script such that Pilz planner loaded as default, however, if Pilz was unavailable due to unforeseen reasons, OMPL would load as the planner. Fig.

2



demo.launch × | demo.launch (Working Tree)

src > arthur_moveit > launch > demo.launch

```
<launch>

  <!-- specify the planning pipelinekjh -->
  <arg name="pipeline" default="ompl" />

  <!-- By default, we do not start a database (it can be large) -->
  <arg name="db" default="false" />
  <!-- Allow user to specify database location -->
  <arg name="db_path" default="$(find arthur_moveit)/default_warehouse_mongo_db" />

  <!-- By default, we are not in debug mode -->
  <arg name="debug" default="false" />

  <!-- By default, we will load or override the robot_description -->
  <arg name="load_robot_description" default="true"/>

  <!-- Set execution mode for fake execution controllers -->
  <arg name="execution_type" default="interpolate" />

  <!--
  By default, hide joint_state_publisher's GUI

  MoveIt!'s "demo" mode replaces the real robot driver with the joint_state_publisher.
  The latter one maintains and publishes the current joint configuration of the simulat
  It also provides a GUI to move the simulated robot around "manually".
  This corresponds to moving around the real robot without the use of MoveIt.
  -->
  <arg name="use_gui" default="false" />
  <arg name="use_rviz" default="true" />

  <!-- If needed, broadcast static tf for robot root -->


  <!-- We do not have a robot connected, so publish fake joint states -->
  <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publ
    <rosparam param="source_list">[move_group/fake_controller_joint_states]</rosparam>
  </node>
  <node name="joint_state_publisher" pkg="joint_state_publisher_gui" type="joint_state_
    <rosparam param="source_list">[move_group/fake_controller_joint_states]</rosparam>
  </node>

  <!-- Given the published joint states, publish tf for the robot links -->
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publ

  <!-- Run the main MoveIt! executable without trajectory execution (we do not have con
  <include file="$(find arthur_moveit)/launch/move_group.launch">
    <arg name="allow_trajectory_execution" value="true"/>
    <arg name="fake_execution" value="true"/>
```

```
Frame pelvis_link exists with parent table.
Frame table exists with parent world.
Frame base_link exists with parent table.
Frame tool_frame exists with parent end_effector_link.

Failure at 1096.182808080
Exception thrown:Could not find a connection between 'base_link' and 'end_effector_link' because t
The current list of frames is:
Frame end_effector_link exists with parent bracelet_link.
Frame pelvis_link exists with parent table.
Frame table exists with parent world.
Frame base_link exists with parent table.
Frame tool_frame exists with parent end_effector_link.

At time 1097.162
- Translation: [0.218, 0.397, 0.436]
- Rotation: in Quaternion [0.505, 0.493, 0.512, 0.498]
            in RPY (radian) [1.567, -0.033, 1.581]
            in RPY (degree) [89.764, -1.893, 90.595]
^Coem@sas:~$ rosrun tf tf_echo base_link end_effector_link
At time 1125.082
- Translation: [0.214, 0.399, 0.436]
- Rotation: in Quaternion [0.502, 0.496, 0.514, 0.488]
            in RPY (radian) [1.567, -0.033, 1.591]
            in RPY (degree) [89.764, -1.893, 91.168]
At time 1125.082
- Translation: [0.214, 0.399, 0.436]
- Rotation: in Quaternion [0.502, 0.496, 0.514, 0.488]
            in RPY (radian) [1.567, -0.033, 1.591]
            in RPY (degree) [89.764, -1.893, 91.168]
^Coem@sas:~$ rosrun tf tf_echo base_link end_effector_link
At time 1141.782
- Translation: [0.214, 0.399, 0.436]
- Rotation: in Quaternion [0.500, 0.498, 0.512, 0.498]
            in RPY (radian) [1.567, -0.023, 1.591]
            in RPY (degree) [89.764, -1.328, 91.166]
At time 1141.782
- Translation: [0.214, 0.399, 0.436]
- Rotation: in Quaternion [0.500, 0.498, 0.512, 0.498]
            in RPY (radian) [1.567, -0.023, 1.591]
            in RPY (degree) [89.764, -1.328, 91.166]
^Coem@sas:~$ rosrun tf tf_echo base_link end_effector_link
At time 1168.282
- Translation: [0.218, 0.397, 0.436]
- Rotation: in Quaternion [0.502, 0.496, 0.509, 0.493]
            in RPY (radian) [1.567, -0.023, 1.582]
            in RPY (degree) [89.764, -1.328, 90.615]
At time 1168.282
- Translation: [0.218, 0.397, 0.436]
- Rotation: in Quaternion [0.502, 0.496, 0.509, 0.493]
            in RPY (radian) [1.567, -0.023, 1.582]
            in RPY (degree) [89.764, -1.328, 90.615]
^Coem@sas:~$
```

Figure 2: Cartesian pose and ROS node

As I was able to generate trajectories in MoveIt! and visualize the motion on Gazebo, I saved the generated trajectory in the joint space as joint positions, velocities and accelerations as a rosbag. These waypoints were read from the MPC controller and was followed closely with a small error, but within our performance requirements. I also visualized the pose of the end-effector from the controller to validate and compute the error between the desired end state from the Planner and the actual state reached by the controller.

2 Challenges

I had a couple of challenges while setting up the IKFast plugin. While trying to install and configure it, I had to run into a lot of errors due to poor and old documentation. I had to load a docker image and load the URDF of our arm. Due to environment mismatch, I had to try a couple of times to get it right.

There were quite a few challenges in setting up the Pilz industrial planner set up and configured with MoveIt!. The Pilz planner again has insufficient document and the examples given don't work for all manipulator manufacturers. It took significant effort and debugging to get the planning pipeline set up in parallel to the OMPL planner. I also had to collaborate with members of another team working on similar problems to debug issues and accelerate.

I wrote a node to send commands to the arm via a script. This was easy to do for the default commands set up by Kinova and the default planner. However, when I tried to the same thing using the Pilz planner, there were a number of problems as the examples don't translate directly to custom cases. I had to interpret each functions working, sometimes even guess what some function might do and build from there.

As a team, we had challenges with the MPC controller to converge. Although Anthony worked on it individually, I collaborated with him closely to send the right set of trajectories so that we are able to understand the limitations of the code as it is written now.

3 Team Work

3.1 Anthony Kyu

Anthony Kyu 10:26 AM Anthony worked on further developing the Model Predictive Controller, refactoring the code to use previously overlooked functions implemented in the RigidBodyDynamics.jl Library, and simplifying the dynamics model to only include joint positions and velocities in the state to simplify the overall MPC to get an initial code base. Then, in order to get the MPC to converge, Anthony worked to create an appropriate initial guess for the input/torque trajectory based on the given state trajectory, working on several approaches. Once the MPC converged Anthony worked on visualization, optimization and initial code implementation of the MPC into ROS using RobotOS.jl library in Julia. Anthony also collaborated with Sundaram to get a sample trajectory to work with and sanity checked Parker's PCB.

3.2 Kaushik Balasundar

Kaushik worked alongside Gunjan in developing the landmark capture capability to convert the marker pose into pointcloud2 messages. This was then visualized on RViz and verified with ground truth to ensure the data is of the right scale. He also modified the URDF to incorporate new design changes and also added a force-torque plugin into the Gazebo simulation. He assisted Sundaram in extracting a trajectory planned by the Pilz planner for preliminary testing of the MPC controller.

3.3 Gunjan Sethi

Gunjan worked on developing the landmark capture functionality along with Kaushik. During this, the main tasks involved understanding the PointCloud2 message type in ROS, prototyping a simple pointcloud collection function and visualizing it in RViz. Later, they performed a test to verify the scale of the collected pointcloud and to study the effect of orientation of registration probe on the collected points. Gunjan also prepared slides for and presented the project management portion of the PDR.

3.4 Parker Hill

Parker worked on creating a end-effector reaming adapter for this progress review, going through the process of designing, 3D printing, assembling, and analyzing the assembly. He also worked on the PCB assignment, going through the steps to create the board, find parts, and verify performance for submission. Finally, Parker helped to perform the motor speed and torque test as well as helped to create a ROS node which runs uses Julia.

4 Future Plan

As part of our sprint for progress review 4, I will work on making the planning pipeline autonomous such that a transform can be taken from the perception subsystem which would be the end point of the trajectory. I will write to a script to subscribe to this transform and plan a trajectory between the current point and the end point. I will also validate the motion followed by the arm in comparison to the generated trajectory and compute the error. To do this, I will use the RPG trajectory evaluation package. I will also add new states to the waypoints such as the end-effector position, velocity and acceleration and the forces/torque at the end-effector. I will also write a publisher node such that the controller can directly subscribe to the waypoints generated by the planner.