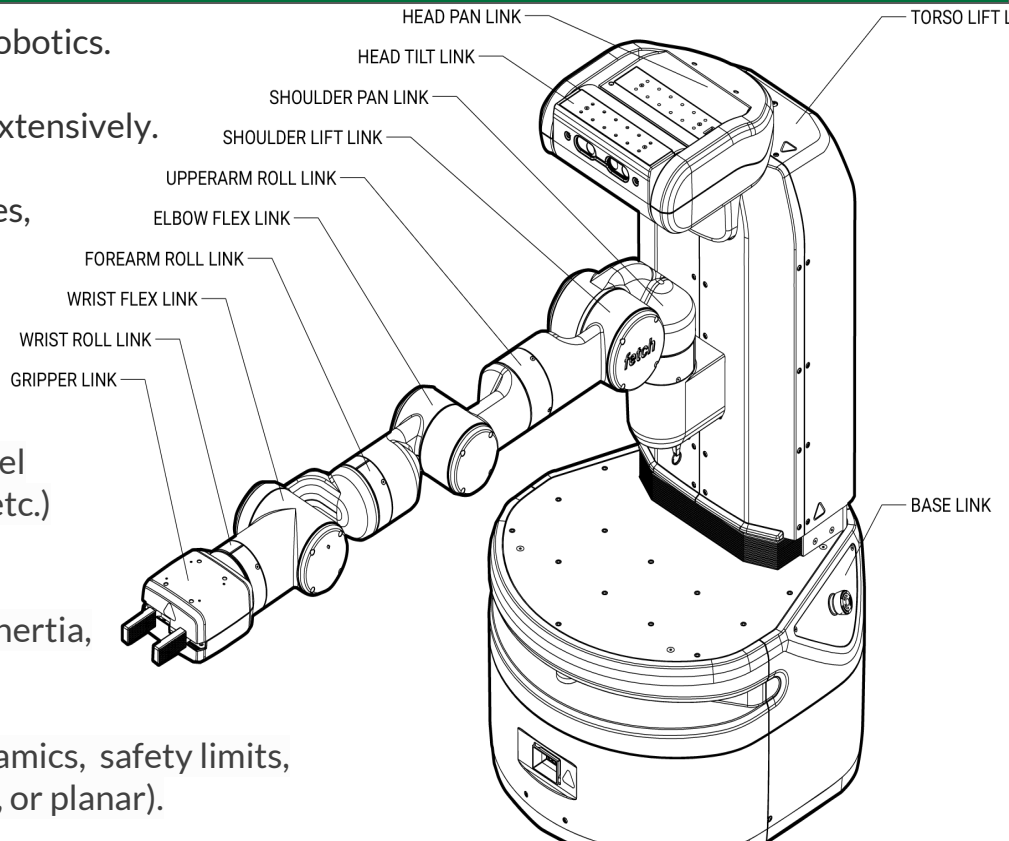
UNC CHARLOTTE
College of Computing and Informatics

Fetch Robot
Pick and Place Demo

By:
Hima Bindu Sigili
Vinay Branham Reddy

Introduction

- Fetch is a mobile manipulation platform built by Fetch Robotics.
- The Fetch software system uses ROS (Indigo Version) extensively.
- All Fetch capabilities are thus available via ROS messages, services or actions.
- The Fetch kinematics are defined by using the concepts of joints, links, and coordinate frames.
- The robot URDF (unified robot description format) model specifies the attributes (kinematic tree, names, ranges, etc.) of the joints, links, and frames of the robot.
- A link element in the URDF describes a rigid body with inertia, visual features, and coordinate frames.
- A joint element in the URDF defines the kinematics, dynamics, safety limits, and type (revolute, continuous fixed, prismatic, floating, or planar).



Connecting to Network

- Once the robot is turned on and the robot is on the network (connect to the eduroam wireless network)
- If you are unable to access the robot through ssh due to your network settings you will need to connect an HDMI monitor, USB keyboard and USB mouse to the side panel ports.
- Use them to select your network from the networking menu, ssh into the computer of the robot using the fetch user account and IPv4 Network address from remote computer.
- For instance: `>$ ssh niner@10.38.5.0`
- you will need an Ubuntu machine (14.04) with ROS indigo installed to properly communicate with the robot.

Joystick and Teleop

- To pair the controller with the robot, press the middle button (16) once the robot has powered on. The controller will vibrate once successful. To drive the robot base, hold the primary deadman button (button 10 above) and use the two joysticks. The left joystick controls turning velocity while the right joystick controls forward velocity.
- To tuck the arm, press and hold button 6 for one second. This will trigger the tuck_arm server to tuck the arm. While tucking the arm, Fetch will avoid collisions with itself, however it will not be using any active perception, so be sure to keep the space in front of the robot clear when running the tuck arm. While the arm is tucking, you will have to hold the deadman
- To Teleop from remote computer : >$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py

Visualization

From remote computer you can manually set up your RVIZ visualization

>$exportROS_MASTER_URI=http://<robot_name_or_ip>:11311

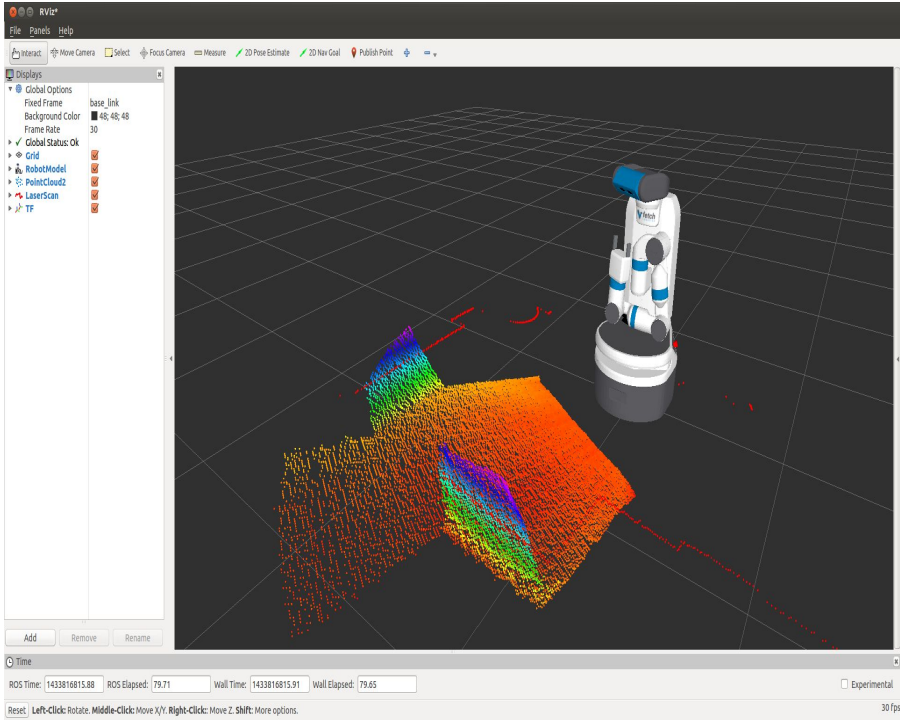
>$ rosrun rviz rviz

Or Re-run RVIZ with a configuration file using the command line. The default .rviz configuration file for Fetch can be loaded using:

>$ roscd fetch_navigation/config.

>$export ROS_MASTER_URI=http://<robot_name_or_ip>:11311

>$ rviz -d navigation.rviz



Building a Map

Building a Map:

When running navigation on a robot, first you will need to build a map,
>$ roslaunch fetch_
Once you launch build_map, you will want to tele-operate the robot around and build the map, which can be visualized in RVIZ.

While driving the robot around, you can view the map in RVIZ.
Once you are happy with the map, you can save the map:
>$ rosrun map_server map_saver -f <map_directory/map_name>

Navigation

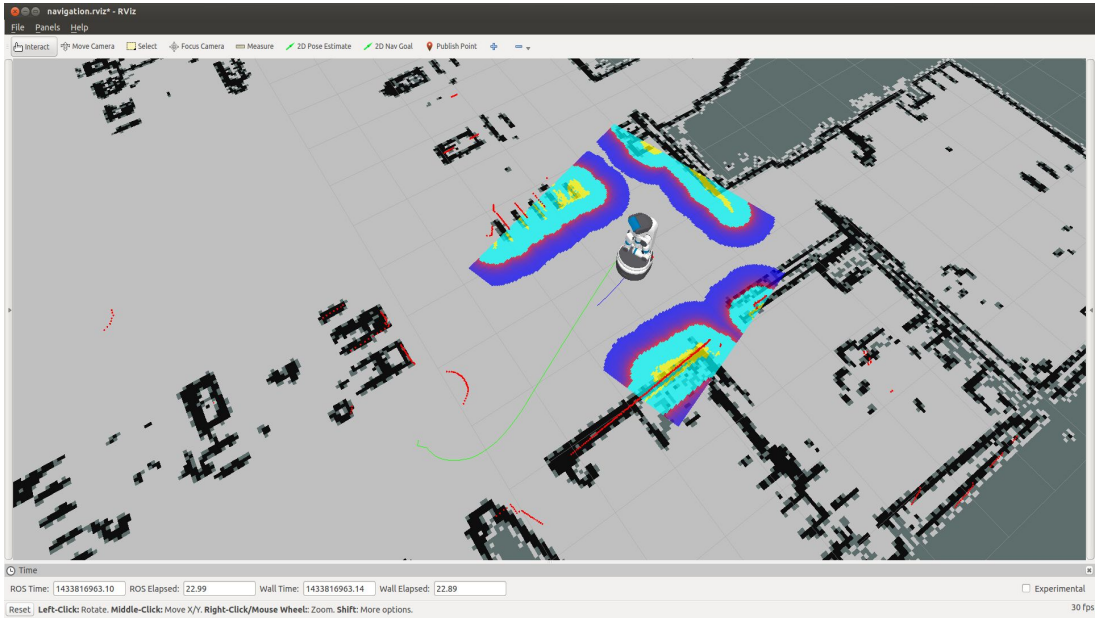
Navigation:

Then you will need to supply the map to the navigation launch file from the fetch_navigation package.

>$ roslaunch fetch_navigation fetch_nav.launch map_file:=/path/to/map.yaml

For instance: >$ roslaunch fetch_navigation fetch_nav.launch map_file:=/home/niner/maps/my_map.yaml

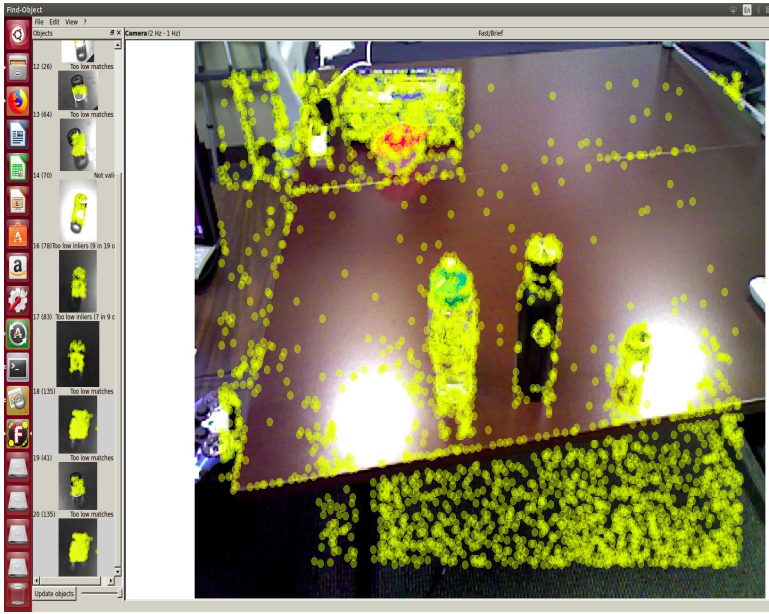
The easiest way to send a goal to the navigation stack is using RVIZ and the 2D Nav Goal button.



Perception

3D Object Finder:

We used find_object_3d ros package for having the position of the objects in the 3D space, to be able to grasp them. The only real difference with the 2D detection will be the sensors involved and the fact that the ObjectPoseStamped will be transformed into TF.



How to launch?

>$ roslaunch my_object_recognition_pkg my_find_object_3d.launch

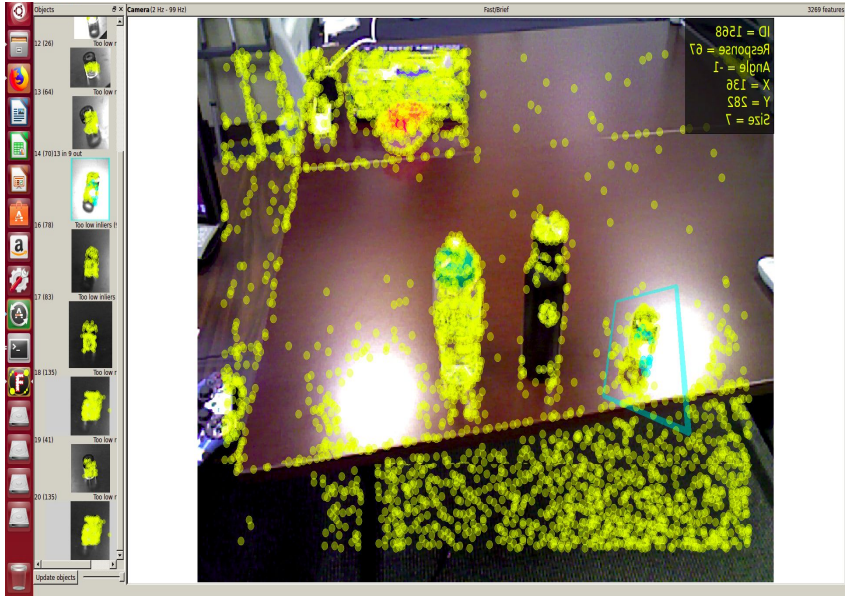
To teach the Fetch robot to recognize a particular object, select the **Edit->AddObjectFromScene**. In the Adding Object menu, follow the steps to select a region of the image that you consider to be the object.

Once done, you should be detecting the object.

This system compares the image with the saved ones and looks for matches. If it matches in enough points, it considers it the desired object.

The last step is to save all of the objects added. There are 2 main ways:

- Saving the Objects as images: File-->Save_Objects. This will save all of the images taken in a folder
- Saving the Whole session: File-->Save_Session. This will save a binary with all of the images and settings. This is the most compact way of doing it, although you won't have access to the images of the objects. It depends on your needs



The rostopic "/Objects" publishes the object's position (ObjectStampedPose that be transformed into tf) and to which later we subscribe and use it for grasping.

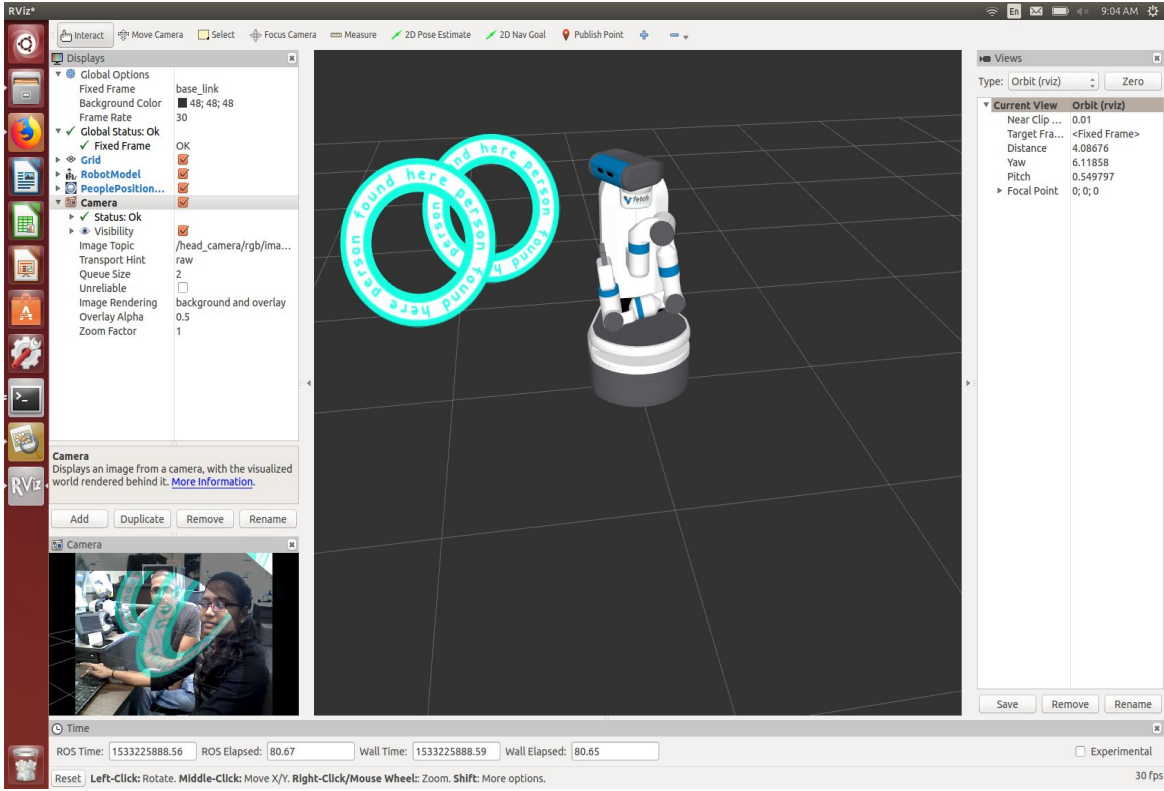
Face Detection

We used face detector package to detect multiple faces in the environment. The first thing to note is that this is not run-of-the-mill RViz marker. This is because it's using the jsk_rviz_plugins package. We have installed this package.

- **PeoplePositionMesurementsArray**: This is the blue circle drawn around the position of the face detected. It's the main data we are looking for.
- **Camera**: Just the RGB camera, as a reference, which RViz superimposes the PeoplePositionMesurementsArray data

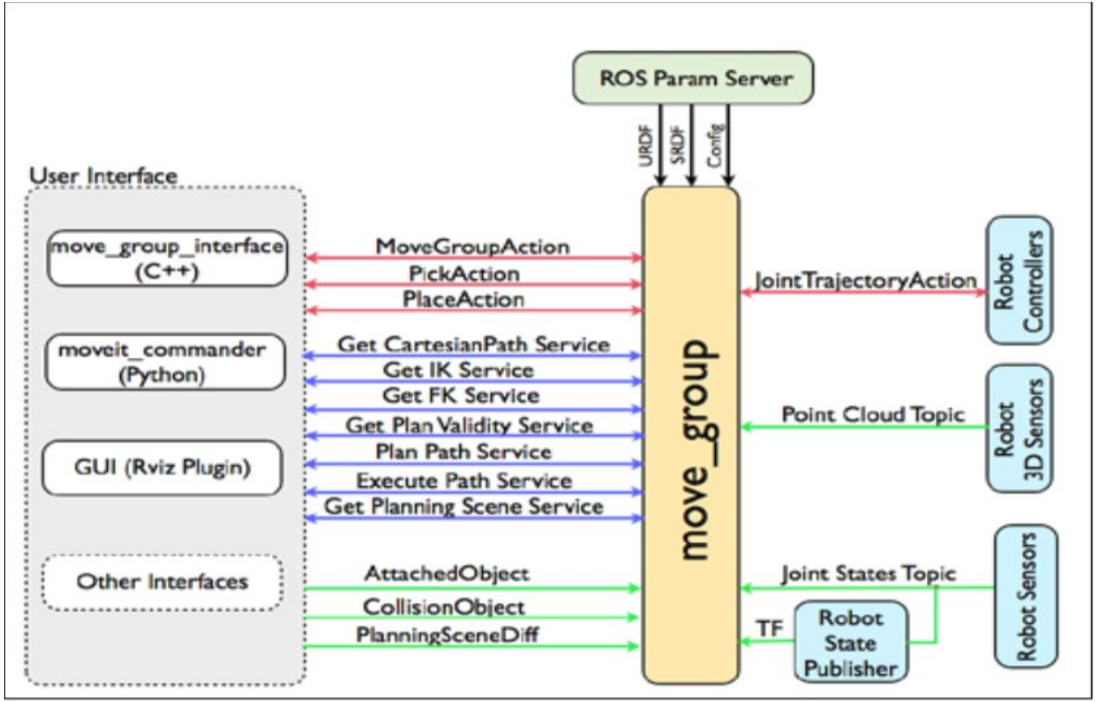
How to launch?

>$roslaunch my_face_detector face_detection_tc.launch



Manipulation

Moveit Architecture:



Manipulation

How to Create Move_it package:

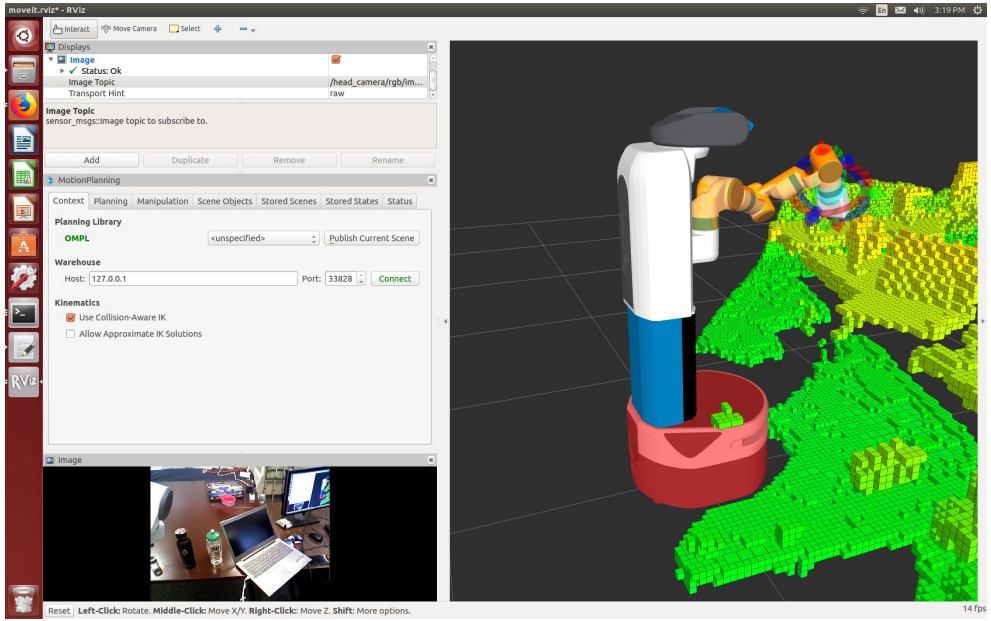
Manipulation

As we are working on an actual robot, we have created and configured a controllers.yaml file inside the config folder of the moveit package. This file contains the details of action servers that helps control the joints in case of a real robot. Then, we have created fetch_moveit_controller_manager.launch file to load the controllers.yaml file that we created.

Next, we have created a file called joint_names.yaml in the config directory that defines all the names of the joints of the robot. Then, we have created a launch file called my_planning_execution.launch. This launch file loads the joint_names.yaml file that we just created.

Finally, we have launched the my_planning_execution.launch file to launch the moveit rviz and plan a trajectory. We have also written a python script in my_motion_scripts directory to move the arm to a certain point in the space.

Adding Perception to Moveit



To add a sensor to Moveit package, we have created a new file called sensors_rgbd.yaml inside the config folder. This file configures a plugin to interface the 3D sensor with the MoveIt. Then we have created a fetch_moveit_sensor_manager.launch.xml inside the launch directory to load the yaml file we created.
Now, the scene would be similar to this.

we are using PointCloudUpdater plugin to bring the PointCloud obtained from the camera.

Grasping

Used the fetch's own grasping client. There is a demo for this and can be found at:

https://github.com/fetchrobotics/fetch_gazebo/blob/gazebo2/fetch_gazebo_demo/scripts/pick_place_demo.py

Pick and Place

Pick a desired object

Questions??


Thank
You!!!