Franka Emika 3D Slicer Integration

This project demonstrates some possibilities of using 3D Slicer for tracking and control of the Franka Emika robot arm

Overview

The project draws some ideas from a Slicer plugin for the Kuka LightWeightRobot. We setup a model of the robot workspace in 3D slicer then we setup a tracker for the robot end effector relative to it's workspace using registration methods and we show the realtime position (and pose) of the robot end effector in 3D Slicer. Also, we set up a workflow to select workspace positions in 3D slicer and make the robot move to those positions in the physical workspace. An improvement to this project would be using a model of the whole robot in 3D slicer as the plugin referenced above does.

Key Features

- Realtime tracking of the robot end effector position *relative* to the robot workspace.
- Moving the robot to points in the physical workspace selected from the workspace model in 3D Slicer.

Our Setup

In development, we had a Franka Emika robot arm, a realtime linux box connected to it for control of the robot using the libfranka library, and we had a third computer (a laptop) running the 3D slicer software. We connected the laptop to the realtime linux machine using an ethernet cable for high frequency communication and safety (WiFi could be unsafe).

Software Used

- 3D Slicer
- SlicerOpenIGTLink: a 3D Slicer plugin.
- OpenIGTLink: communication interface.
- ZeroMQ: messaging library.
- libfranka

Implementation

There are 3 C++ files in the program

- `final_project.cpp`: This file contains the code that is run on the linux realtime kernel. It functions to control the robot using the libfranka library. The point registration is performed here. It also communicates with programs on the laptop sending the (transformed) robot positions

and recieving workspace positions to move the robot to and moves the robot.

- `Tracker.cpp`: This file contains the code that performs the tracking of the robot end effector. The main ideas implemented here are gotten from the examples in the OpenIGTLink repository. The program continually recieves the robot EE position using zmq and sends them to 3D Slicer using the OpenIGTLink protocol. This program runs on the laptop.
- `Reciever.cxx`: This file contains the code that sends workspace points to the program running on the realtime linux machine. The main ideas implemented here are gotten from the examples in the OpenIGTLink repository. This program listens to 3D slicer for points, receives them from 3D slicer using OpenIGTLink and sends them to the program on the linux kernel using zmq.

Running the Programs

- Compile the C++ files on their respective machines as described above.
- When the robot is running in fci mode, run `./final_project <host>` with the ip address of the robot.
- Follow the steps and register points in the workspace
- Then run `./Tracker <port>` and `./Reciever <port>` with different free ports.
- Open 3D Slicer and install the SlicerOpenIGTLink plugin then create connections to the appropriate ports
- Load the workspace model into 3D slicer.
- You should see the tracker in the workspace in 3D slicer and you should be able to send points and see the robot move.

Note on Mathematics

Notice that in this project, we transform points from the image space to the physical space (when moving the robot to desired points) and we transform physical space points to image space points. One way to do this would be to perform the registration algorithm twice but that could be cumbersome. So we use the same R and t in the two tranformations.

From image to physical: $y = Rx + t$ (the usual transformation)

From physical to image: $x = R^{-1}(y - t)$. We get this by solving for x in the registration algorithm. We know that this is well defined and R has an inverse because it is a memeber of the special orthogonal group since it is a rotation matrix.

Also, we are able to get the transformed robot poses into the workspace by using the same method to transform the physical robot pose

$$R_{image} = R^{-1} \times R_{phy}$$