7MR10070: Software and Robotic Integration
Semester 2

Assignment 3

Written by Alexandros Megalemos

ROS slicer integration and robotic control

Introduction

Application Program Interface (API)

An API is an interface/toolset whose purpose is to provide developers/users with useful functionality in a simple manner. An API glues together / connects different pieces of software. A few examples of APIs are libraries such as numpy (for python), or more web-based APIs such as SOAP and REST.

One of the reasons to develop an API is so that other developers (or you), won't have to develop the same piece of code again. By simply "encapsulating" a piece of software/algorithm within an API, you can then reuse it without having to know how it works, just that it works.

Another reason to develop an API is security. For example, a company could give users free access to their database (not recommended) but that could lead to many problems, such as security breaches and loss of data (either on purpose or by mistake). What they could do instead is develop an API with, for example, a function called *getAvailableProducts()* which will handle all security issues as well as ensure that the correct data is returned.

A final reason to develop an API is automation and integration. For example, a certain third-party website could use the APIs of various airline websites to alert a user that a flight they're interested in has reached a certain price by a certain airline.

This leads to the next point which is that APIs should be treated as products and not as random collections of code to be forgotten.

Pros of an API:

- Automation – Allows computers to manage work instead of people
- Efficiency/Reusability – Developers do not have to develop the same functionality repeatedly
- Integration – Allows content from various parties to be connected into one
- Security – By allowing access only through the API, the provider makes sure that all necessary security measures are enforced (assuming they did a good job)

Cons of an API:

- Relying too much on an API could cause major issues. A publisher of an API could decide to change their implementation without warning, rendering your application useless
- Security – Trusting that you or the publisher of the API have/has taken all the necessary security precautions can be risky
- Loss of control – Once you have deployed an API, it is harder to make changes to it as you may mess up with other people's applications

Transfer Communication Protocol (TCP)

TCP is a transmission standard whose purpose is to connect different devices within network (which can be either private or public). It works with the Internet Protocol (IP) which defines how computers send data to each other. TCP is responsible for organising the data that is being transferred in a secure manner and it guarantees the integrity of the data.

TCP is connection-oriented which means that a connection needs to be established between the devices (server-client) before any data is sent. In order to establish this connection, a three-way handshake is used. The way this works is as follows:

1. A Source Machine sends an initial request by sending a message to a Destination Machine (SYN)
2. The Destination Machine acknowledges this request and agrees or disagrees to establish a connection by sending a message back (ACK)
3. The Source Machine acknowledges the Destination's agreement, and establishes the connection (SYN-ACK)

More checks can be added to establish a connection and increase security and data integrity. The main (and possibly only) drawback is the increase in latency.

TCP/IP works by breaking the message/data into smaller parts called packets, and then sending them one by one between the devices. Each packet carries information to help the devices identify which part of the message it is, from whom it was sent, when it was sent and so on. The TCP part is responsible for assembling the packets into the complete message once the transfer is finished. If for any reason a packet is missing, it is responsible for requesting it again.

TCP is important because it establishes a standard for how information is communicated over a network. This means that anyone using TCP knows the format in which the data will arrive, making it easier to interpret it.

Pros of TCP:

1. Reliability – Since it is a standard adopted by the whole industry, transferring data using TCP means that we can expect how the data is transferred.
2. Cross Platform – It works across different/heterogeneous networks
3. Open Protocol – No one owns TCP and therefore can be used by anyone
4. Scalable – Networks can be added without disruption to current services
5. Data integrity – TCP can detect if data is lost or damaged and can therefore be discarded and requested again

Cons of TCP:

1. Strict Protocol – It is not generic by design. Certain rules have to be followed to use it.
2. Bad for small networks – It is designed to be used by wide networks
3. Vulnerable to Denial of Service attacks – Attackers can abuse the handshake process to send many requests to establish a connection with the server, thus overloading it and rendering it unable to establish legit connections
4. Vulnerable to Connection Hijacking – Eavesdroppers can redirect packets if they can learn the sequence number from an ongoing communication
5. It is not a network – TCP is just a protocol. It only describes how a connection should be established.

Methods

We initialize a link between 3D Slicer and ROS using the *ros_igtl_bridge* package. Since I am using a locally run Slicer and running the ROS part on a virtual machine the following steps need to be taken:

Slicer Part

1. Launch Slicer 4.8.1
2. Go to your favourite IGTLink extension (SlicerIGT in this case)
3. Create a connection with Slicer as the server on port 18944

ROS Part

1. Start the VM with the network adapter attached to NAT
2. Launch the bridge file of the *ros_igtl_bridge*
3. Choose to run as client
4. Set IP to 10.0.2.2
5. Set port to 18944

A connection should be now established.

In order to send messages between the two we use the OpenIGTLink protocol. OpenIGTLink defines the message format that is used to transfer data between Slicer and ROS. The message consists of a header, an extended header, the content and some meta data. Furthermore, we need to define an importer in order to interpret the above message format. If we wanted to have two-way communication, we would also have to define an exporter but for the purposes of this assignment this has not been done.

The importer used for this assignment expects to receive a pose goal to go to.

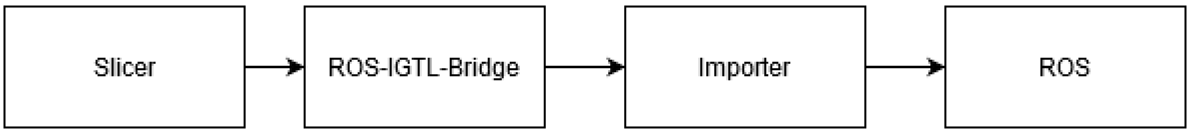


Figure 1: Connection between Slicer and ROS

Validation

Sending data from Slicer to ROS

First initialize the connection as mentioned in the methods section. Then:

Step 1:

1. In Slicer 4.8.1
2. Go to Markups
3. Create a new MarkupFiducials called “Goal”
4. Place a point in the workspace (I did this the other way around. I manually moved the robot in RVIZ and those coordinates instead)

Or simply use the provided scene “SlicerToRos” found in <directory>/Scenes

Step 2:

1. Launch a new terminal
2. Go to your workspace and source it
3. Enter “roslaunch panda_moveit_config demo.launch”

Step 3:

4. Launch a new terminal
5. Go to your workspace and source it
6. Enter “rosrun robot_control slicerToRos.py”

Step 4:

1. Go back to slicer
2. Go to IGT -> IGTLinkIF
3. Scroll down to I/O Configuration
4. Click on either “Goal1” or “Goal2” (if you used the supplied scene)
5. Click send (as seen here Figure 6: Slicer Scene)

If you now look at RVIZ, the robot should now be moving.

Position Validation

Default Position

Figure 2: Robot at default position shows our robot in the default position

At Goal 1

Figure 3: Robot at Goal 1 shows our robot at Goal 1 position after sending the points from Slicer

Going to Goal 2

Figure 4: Robot moving to Goal 2 shows the robot on its way to Goal 2. The trajectory planning helps us see the trajectory of the robot

At Goal 2

Figure 5: Robot at Goal 2, finally shows us the robot at Goal 2

Code / Git Repository

https://github.com/Meldanen/kcl/tree/master/robotics/slicerAndRos

Appendix

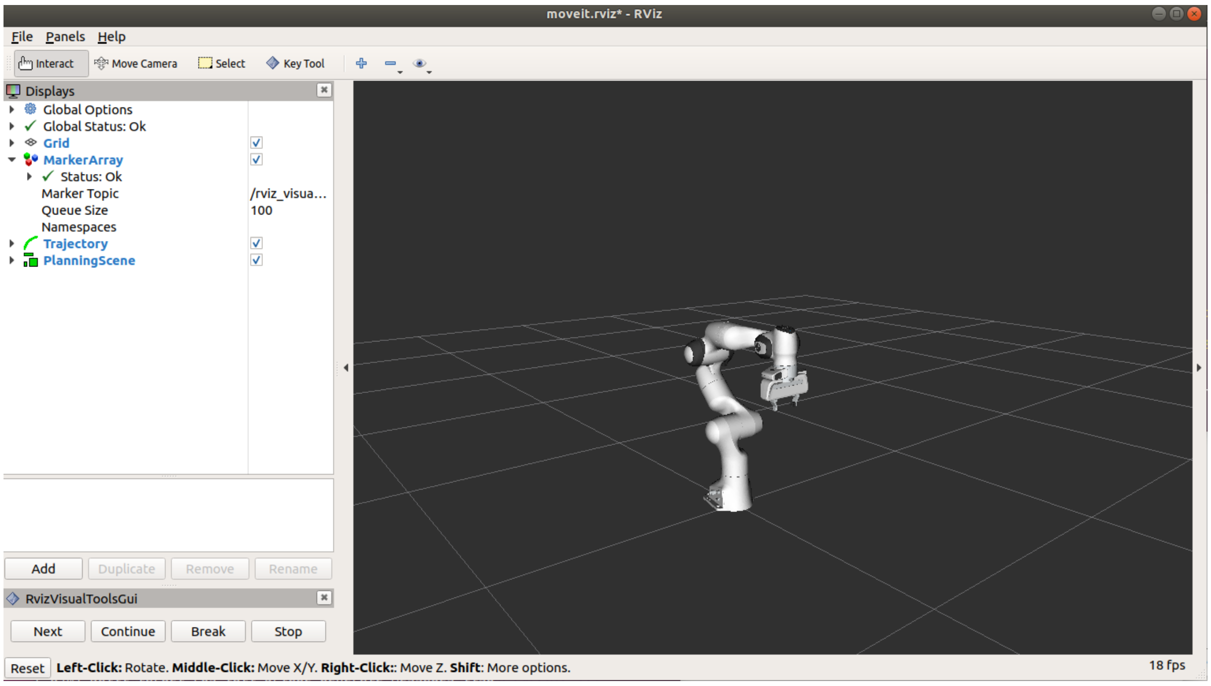


Figure 2: Robot at default position

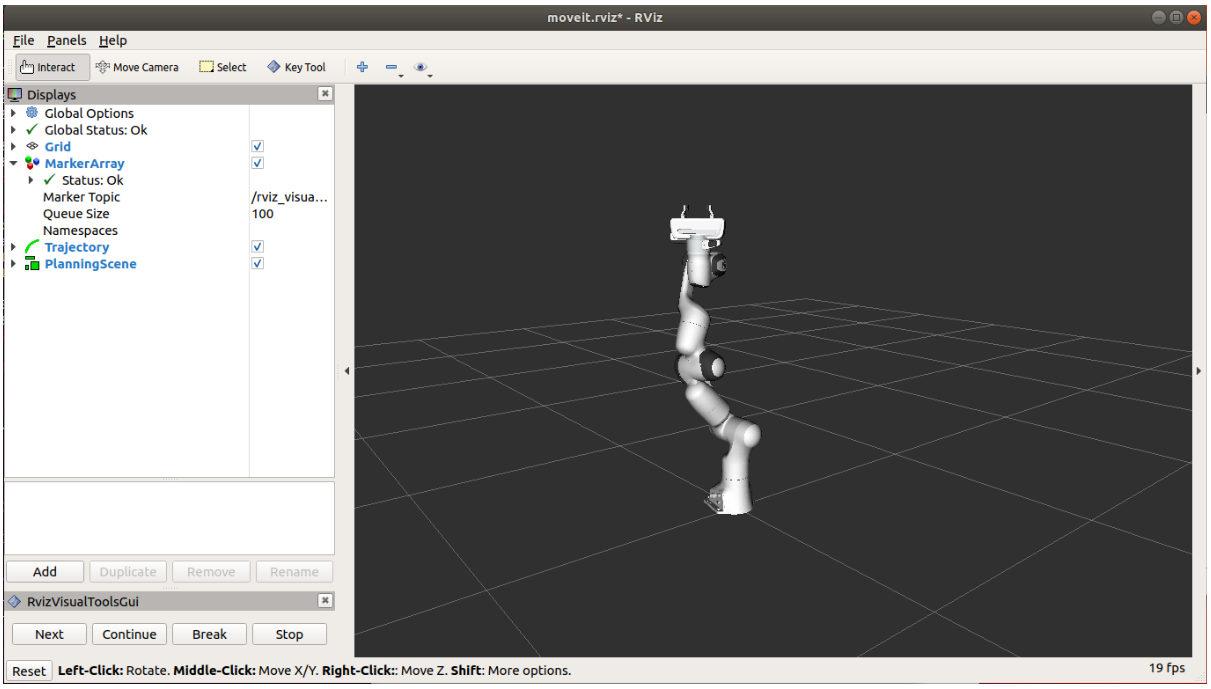


Figure 3: Robot at Goal 1

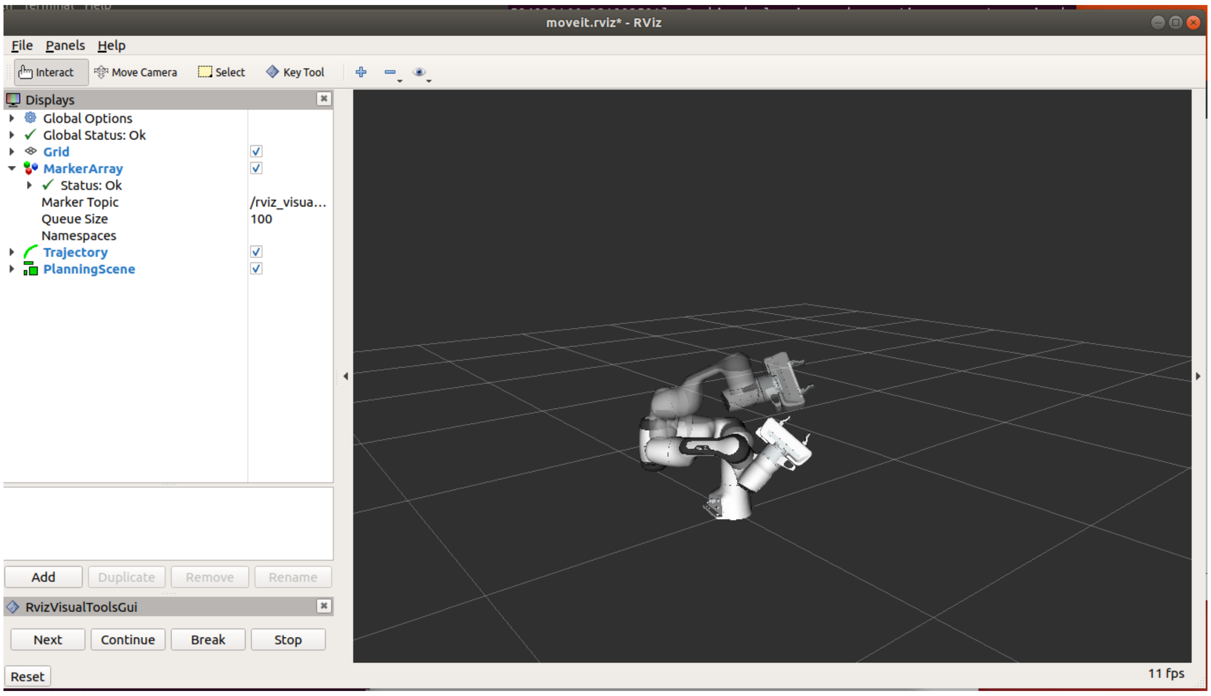


Figure 4: Robot moving to Goal 2

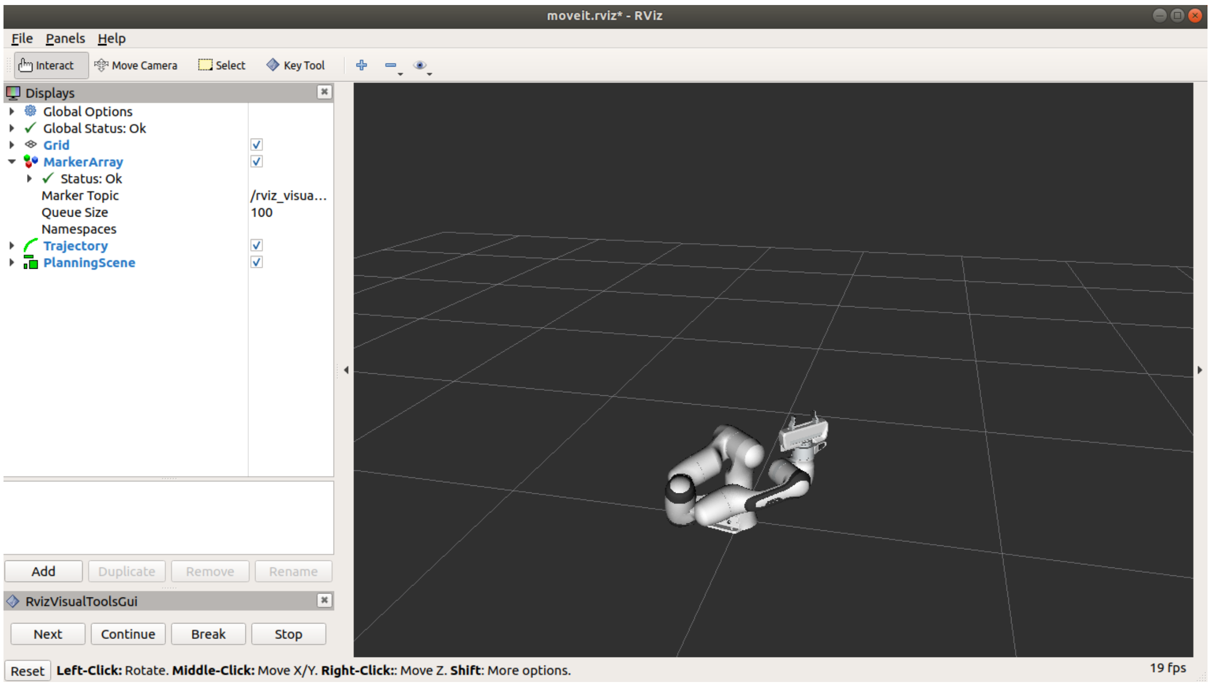
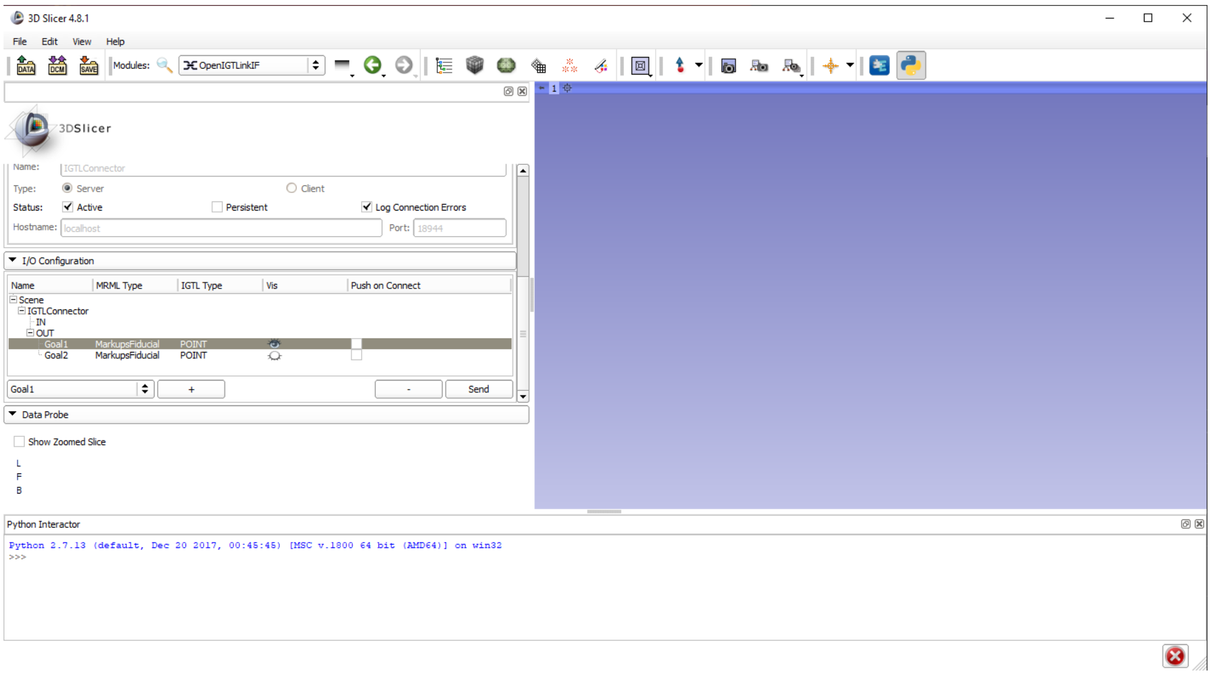


Figure 5: Robot at Goal 2



Name: IGTLConnector
Type: [✓ Server] Client
Status: [✓ Active] Persistent [✓ Log Connection Errors]
Hostname: localhost Port: 18944

Name	MRML Type	IGTL Type	Vis	Push on Connect
Scene				
IGTLConnector				
IN				
OUT				
Goal1	MarkupsFiducial	POINT		
Goal2	MarkupsFiducial	POINT		

Goal1 + - Send

Show Zoomed Slice

L
F
B

```
Python 2.7.13 (default, Dec 20 2017, 00:45:45) [MSC v.1800 64 bit (AMD64)] on win32
>>>
```

Figure 6: Slicer Scene