7MR10070: Software and Robotic Integration
Semester 2

Assignment 2

Written by Alexandros Megalemos

Robotic Simulation

Introduction

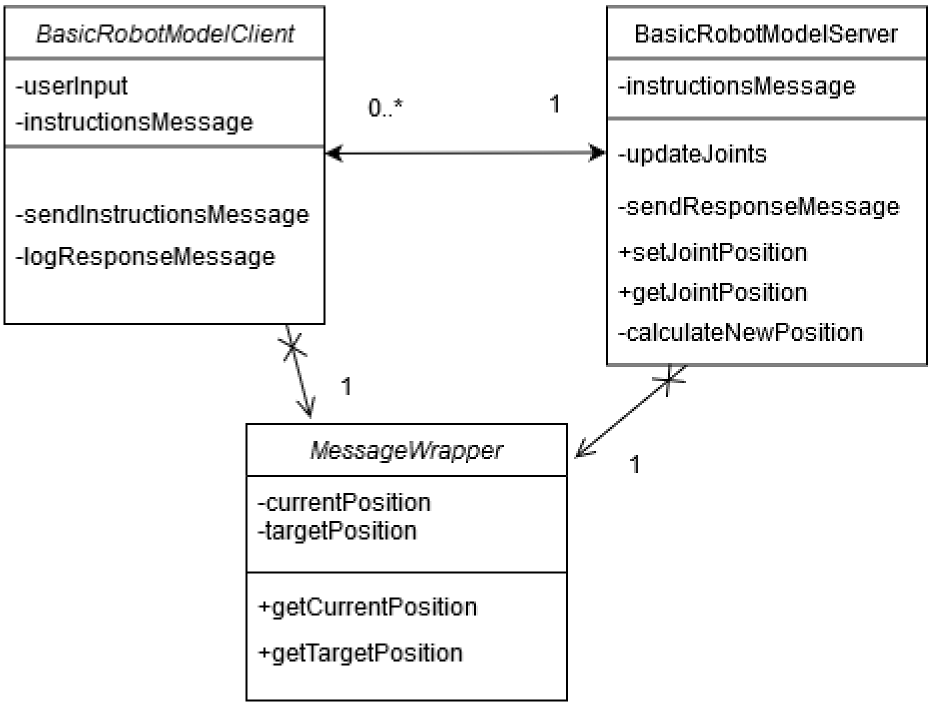


Figure 1: Pseudo UML of the relationship between Client and Server

Figure 1: Pseudo UML of the relationship between Client and Server shows a "pseudo" UML diagram of the system. Some of the functions were omitted for clarity. The basic idea of the system is to have a client-server interface, where the client sends commands to the server and if possible, the server executes them. While only one server should exist, we could theoretically multiple clients sending requests to the server. Both nodes use the MessageWrapper class which hides ugly parts of the message and provides utility methods for it.

BasicRobotModelClient

In general, we don't want the Client to include any logic (regarding kinematics) and so it has no access to any actual logic in terms of calculating kinematics and estimating their validity. The only thing it can do it receive input from the user (i.e. using a terminal or a GUI) and pass (publish) it over to the Server. In order to be able to pass over further input, it needs to receive a reply from the Server, telling it that either: a) The input was valid and the robot has moved to the desired position or b) The input was invalid and it should request a different set of inputs from the user.

The client allows to user to select between two modes. The first allows the user to specify the joint angles directly and the second one allows the user to specify the end effector's desired coordinates.

BasicRobotModelServer

This is where the kinematics are essentially calculated and passed over to the robot. It waits to receive a message that contains all the relevant info (*xyz* or *joint position*) from the Client, checks that the position is valid and then sends a request to the joint state publisher to move the robot. Once the robot has moved to desired position, it sends a message back to the Client to let it know of the current status of the robot.

Robot Model

The model is a simple robot comprised of a base, two joints and two links. Its .xml can be found in the Appendix under Figure 3: Robot URDF/XARCO

Methods

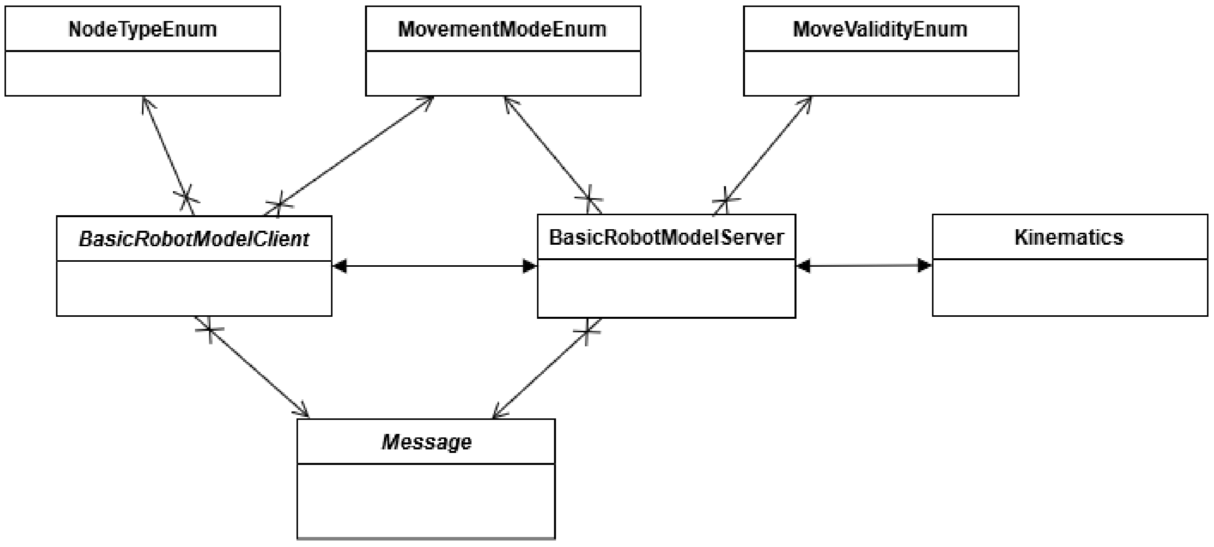


Figure 2: Pseudo UML of the whole system

Client

The client/controller node is called **BasicRobotModelClient.py**. It contains a single class called **RobotModelClient** which itself is comprised of the **init()** function, the **start()** function, a **callback()** function and a **getInput()** function

Init()

- Initialises the node
- Initialises the publisher for a two-way communication between client and server
- Creates an instance of the message wrapper

Start()

- Runs if the node is running (i.e. indefinitely)
- The main function where messages are presented to the user and input is requested

getInput()

- Performs checks to make sure that the input is inserted in a numerical format. If it's not, it rejects it and asks the user to repeat

Server

The server/subscriber node is called **BasicRobotModelServer.py**. It contains a single class called **RobotState** which itself is comprised of the following functions:

Init() and its sub-functions initJointState() and initPublishersAndSubscribers

- Initialises the node
- Initialises the joint state variables which contain the name of the joints and their positions
- Initialises the publishers and subscribers so that the server can communicate with the JointState and the Client
- Creates an instance of the message wrapper

setResponseMessage()

- Sets the response message to send back to client

Start()

- Initiates the program
- Calls the updateJoints() function
- Calls the publishMessage() function

Callback()

- Retrieves the new position of the joints
- Calls the setResponseMessage() function

getRobotConfiguration()

- Gets the current robot configuration based on the joint state position

getJointStatePosition()

- Retrieves the new position of the joints using the Kinematics class

publishMessage()

- Helper function to publish the message to the subscriber

updateJoints()

- Publishes the new positions to the JointStatePublisher node

isValidPosition()

- Checks if the requested position is valid

setMessage()

- Sets the attributes for the message wrapper class

Message Wrapper

The message wrapper class is called **Message.py**. It used in order to hide some functions from the server and client nodes. It acts as a bean-like class with setters and getters to satisfy the fields provided by the **InstructionsMessage.message** file.

Decision behind using only a *.message* file and not an *.srv* file:

The initial plan was to use both or just an .srv file, but after tinkering with the implementation for a bit, I realized that all the information provided by Client to the Server would be useful to be returned back (instead of a simple “Move successful, currently at xyz”) to the Client. Therefore, it didn’t make much sense to either have to maintain both an *.srv* and a *.message* file or to have an *.srv* message where both the instruction and response are the same. Using the .message format was the simplest solution.

Furthermore, it contains some utility methods such as:

clearMessage()

- Clears all fields of the message

isUsingJointPosition()

- Checks if the joint position mode is active by calling the **MovementModeEnum** class

isUsingEndEffectorPosition()

- Checks if the end effector mode is active by calling the **MovementModeEnum** class

Enums

Various enums are used to remove hard coded strings and hide some logic from the client and server nodes.

MoveValidityEnum

Simple enum that contains 3 fields. One to specify that a valid position was requested, one to specify that an invalid position was requested and a not available position in case it is not available yet.

Furthermore, it contains:

isValid()

- Utility function to quickly check if the position is set to valid

isInvalid()

- Utility function to quickly check if the position is set to invalid

NodeTypeEnum

Simple enum that contains node names that would otherwise be hard coded. These fields could have been included as constants in their respective *.py* files, but this implementation removes unnecessary clutter from those files and creates a centralized place where all of them can be included.

MovementModeEnum

Enum to signify the mode selected by the user. It contains 3 fields. One to specify that node mode has been set yet, one to signify that the joint position mode was selected and one to signify that the end effector mode was selected.

Furthermore, it contains:

isJointPosition()

- Utility function to quickly check if the joint position mode is set

isEndEffectorPosition()

- Utility function to quickly check if the end effector position mode is set

It also contains a **Utils** class to help get the enum entry through a label or id. This is useful when translating between the server/client response.

Kinematics

The kinematics class contains methods to calculate the kinematics required based on the user's requests. It is used to separate logic that is not needed to be seen by the server.

getForwardKinematics()

- Calculates the forward kinematics

getInverseKinematics()

- Calculates the inverse kinematics

isValidHeight()

- Checks if the requested position is valid based on the robot's physical properties

Validation

The interface was tested by manual testing. This was mostly done by doing sanity checks on the different kinds of inputs as well as supplying the robot with invalid and valid positions.

Input Validation

Valid Input

Figure 6: Request valid position (end effector mode) shows the two-way communication between Client and Server when valid input is used by the user.

Furthermore, *Figure 4: Robot in RVIZ* shows the robot in its default position and *Figure 11: Robot State after moving (2)* after using the input showed in figure 6. *Figure 10: Robot state after moving,* shows the robot after using the joint position mode with inputs joint1 = 3 and joint2 = 2

Invalid Input

Figure 8: Request non-numeric node (same check used for all inputs) shows validation performed at Client level when invalid input is provided, such as non-numeric characters. The same validation is performed for all fields.

Figure 9: Request invalid mode shows the validation performed at the Client level when an invalid mode is provided.

Position Validation

Valid Position

Figure 6: Request valid position (end effector mode) and Figure 7: Request valid position (Joint position mode) show the two-way communication between Client and Server when a valid position is requested for both modes. The validation is performed on the Server level.

Invalid Position

Figure 5: Request invalid position (end effector mode) shows the two-way communication between Client and Server when a valid position is requested. The validation is performed on the Server level.

Code / Git Repository

https://github.com/Meldanen/kcl/tree/master/robotics/ros/assignment_ws

Use instructions

Setup

The model path specified in <workspace>/launch/display.launch is hardcoded as I run into some issues using relative paths on windows. This was fine on Linux, but I later switched to using Windows as it was easier to visualise.

Server

While within the main workspace, type the command **roslaunch basic_robot display.launch** in a terminal

Client

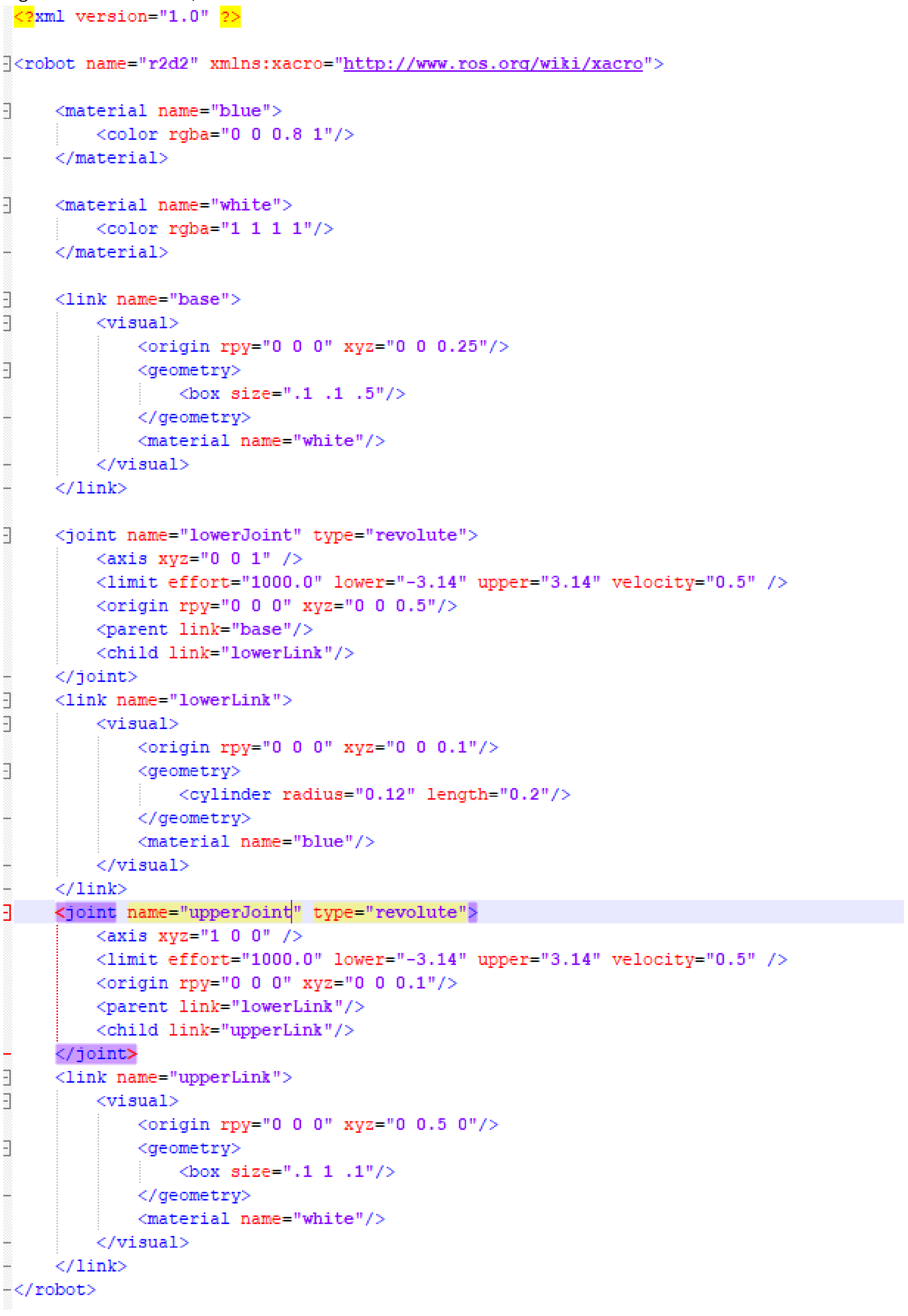
While the server is running, type the command **rosrun basic_robot src/BasicRobotModelClient**

Notes:

I noticed sometimes that on Windows, the first command issued by the Client to the Server is not registered. If you notice this issue, please just type in the same parameters again to go to the desired position. There is no need to re-run the Client's command.

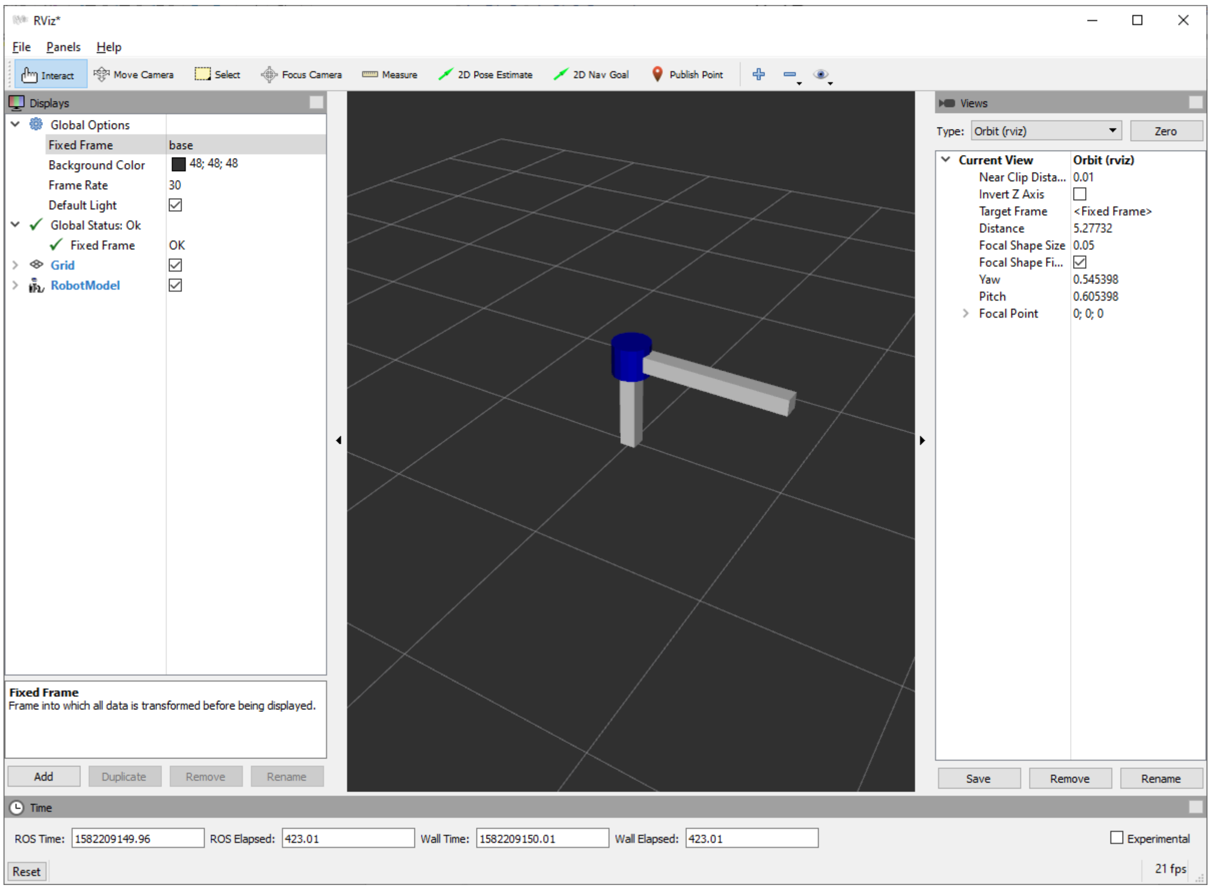
Appendix

Figure 3: Robot URDF/XARCO



```xml
<?xml version="1.0" ?>

<robot name="r2d2" xmlns:xacro="http://www.ros.org/wiki/xacro">

    <material name="blue">
        <color rgba="0 0 0.8 1"/>
    </material>

    <material name="white">
        <color rgba="1 1 1 1"/>
    </material>

    <link name="base">
        <visual>
            <origin rpy="0 0 0" xyz="0 0 0.25"/>
            <geometry>
                <box size=".1 .1 .5"/>
            </geometry>
            <material name="white"/>
        </visual>
    </link>

    <joint name="lowerJoint" type="revolute">
        <axis xyz="0 0 1" />
        <limit effort="1000.0" lower="-3.14" upper="3.14" velocity="0.5" />
        <origin rpy="0 0 0" xyz="0 0 0.5"/>
        <parent link="base"/>
        <child link="lowerLink"/>
    </joint>
    <link name="lowerLink">
        <visual>
            <origin rpy="0 0 0" xyz="0 0 0.1"/>
            <geometry>
                <cylinder radius="0.12" length="0.2"/>
            </geometry>
            <material name="blue"/>
        </visual>
    </link>
    <joint name="upperJoint" type="revolute">
        <axis xyz="1 0 0" />
        <limit effort="1000.0" lower="-3.14" upper="3.14" velocity="0.5" />
        <origin rpy="0 0 0" xyz="0 0 0.1"/>
        <parent link="lowerLink"/>
        <child link="upperLink"/>
    </joint>
    <link name="upperLink">
        <visual>
            <origin rpy="0 0 0" xyz="0 0.5 0"/>
            <geometry>
                <box size=".1 1 .1"/>
            </geometry>
            <material name="white"/>
        </visual>
    </link>
</robot>
```

Figure 3: Robot URDF/XARCO



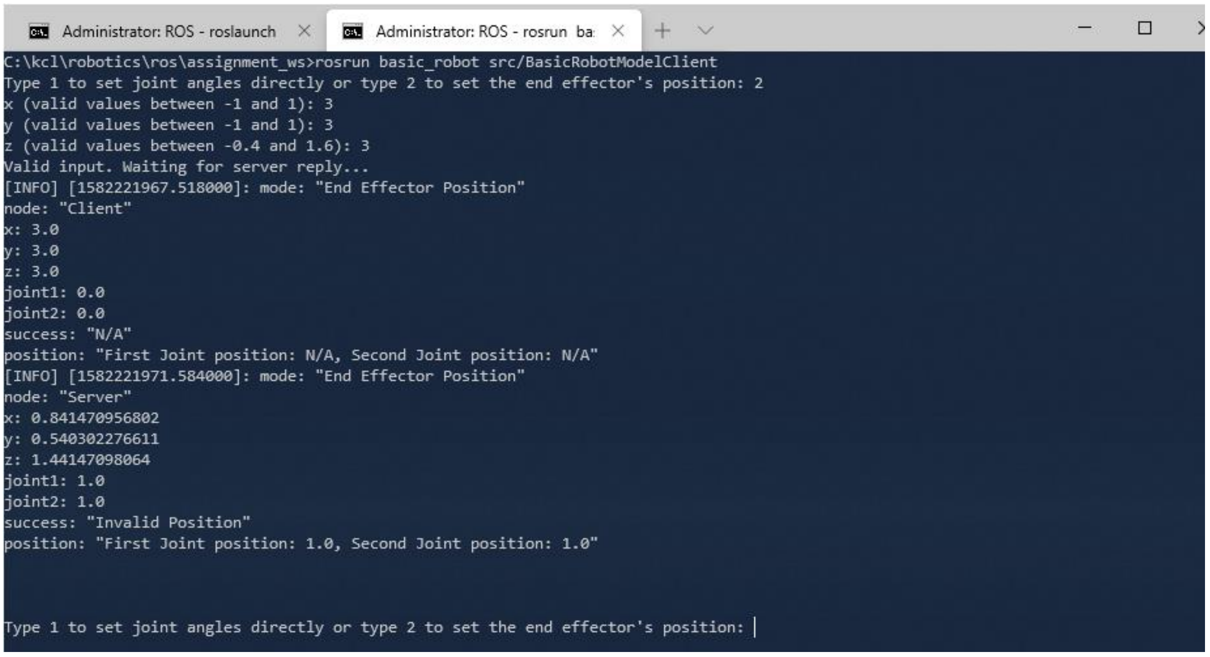
Global Options
Fixed Frame base
Background Color 48; 48; 48
Frame Rate 30
Default Light ✓
Global Status: Ok
Fixed Frame OK
Grid ✓
RobotModel ✓

Fixed Frame
Frame into which all data is transformed before being displayed.

Type: Orbit (rviz)
Current View Orbit (rviz)
Near Clip Dista... 0.01
Invert Z Axis ☐
Target Frame <Fixed Frame>
Distance 5.27732
Focal Shape Size 0.05
Focal Shape Fi... ✓
Yaw 0.545398
Pitch 0.605398
Focal Point 0; 0; 0

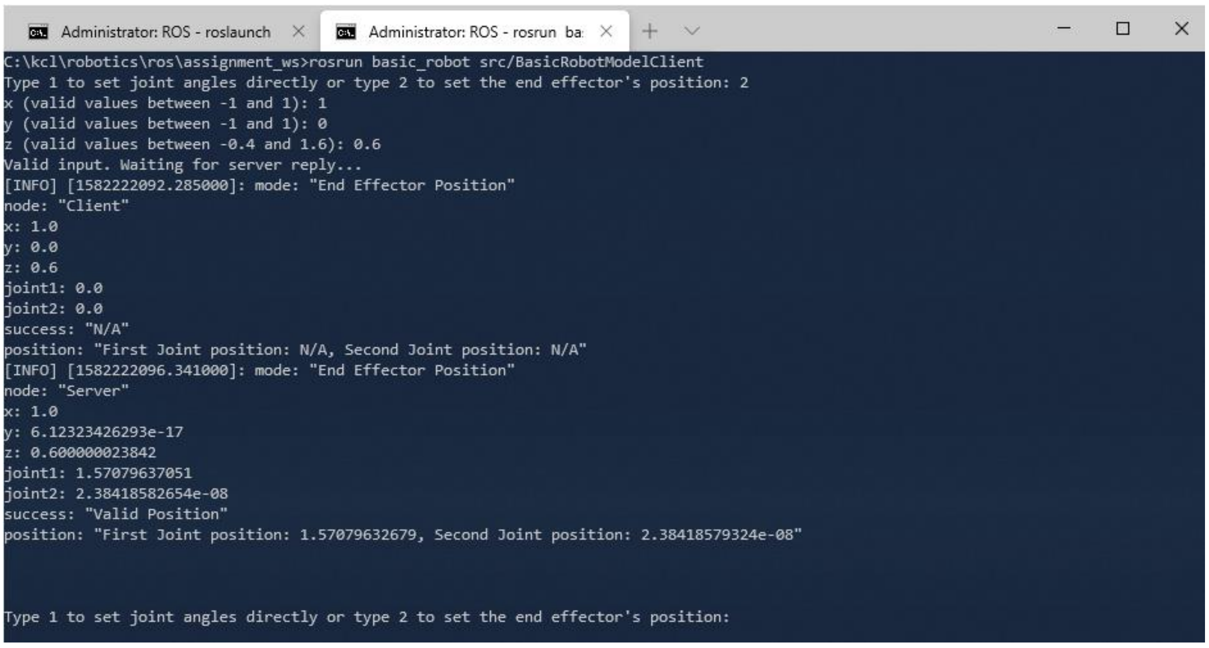
ROS Time: 1582209149.96 ROS Elapsed: 423.01 Wall Time: 1582209150.01 Wall Elapsed: 423.01

Figure 4: Robot in RVIZ


Administrator: ROS - roslaunch
Administrator: ROS - rosrun ba

```
C:\kcl\robotics\ros\assignment_ws>rosrun basic_robot src/BasicRobotModelClient
Type 1 to set joint angles directly or type 2 to set the end effector's position: 2
x (valid values between -1 and 1): 3
y (valid values between -1 and 1): 3
z (valid values between -0.4 and 1.6): 3
Valid input. Waiting for server reply...
[INFO] [1582221967.518000]: mode: "End Effector Position"
node: "Client"
x: 3.0
y: 3.0
z: 3.0
joint1: 0.0
joint2: 0.0
success: "N/A"
position: "First Joint position: N/A, Second Joint position: N/A"
[INFO] [1582221971.584000]: mode: "End Effector Position"
node: "Server"
x: 0.841470956802
y: 0.540302276611
z: 1.44147098064
joint1: 1.0
joint2: 1.0
success: "Invalid Position"
position: "First Joint position: 1.0, Second Joint position: 1.0"


Type 1 to set joint angles directly or type 2 to set the end effector's position:
```

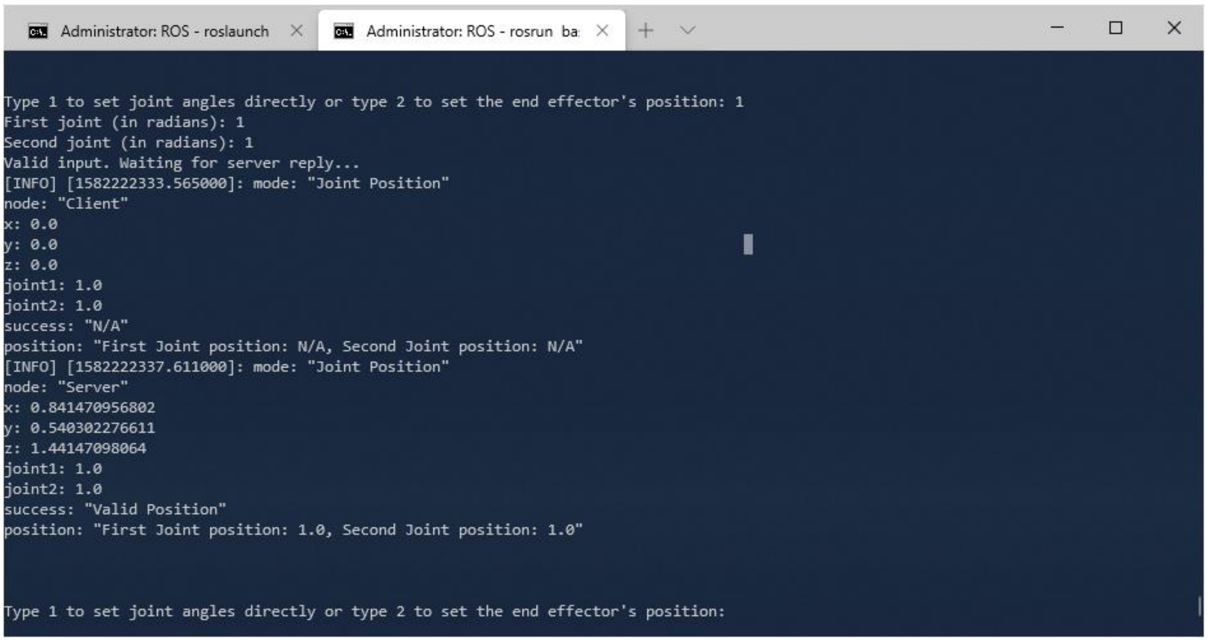
Figure 5: Request invalid position (end effector mode)



Administrator: ROS - roslaunch
Administrator: ROS - rosrun ba:

```
C:\kcl\robotics\ros\assignment_ws>rosrun basic_robot src/BasicRobotModelClient
Type 1 to set joint angles directly or type 2 to set the end effector's position: 2
x (valid values between -1 and 1): 1
y (valid values between -1 and 1): 0
z (valid values between -0.4 and 1.6): 0.6
Valid input. Waiting for server reply...
[INFO] [1582222092.285000]: mode: "End Effector Position"
node: "Client"
x: 1.0
y: 0.0
z: 0.6
joint1: 0.0
joint2: 0.0
success: "N/A"
position: "First Joint position: N/A, Second Joint position: N/A"
[INFO] [1582222096.341000]: mode: "End Effector Position"
node: "Server"
x: 1.0
y: 6.12323426293e-17
z: 0.600000023842
joint1: 1.57079637051
joint2: 2.38418582654e-08
success: "Valid Position"
position: "First Joint position: 1.57079632679, Second Joint position: 2.38418579324e-08"


Type 1 to set joint angles directly or type 2 to set the end effector's position:
```

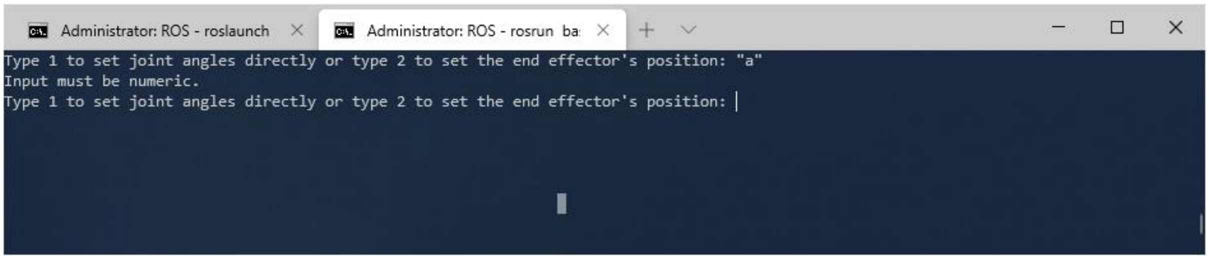
Figure 6: Request valid position (end effector mode)



Administrator: ROS - roslaunch
Administrator: ROS - rosrun ba:

```
Type 1 to set joint angles directly or type 2 to set the end effector's position: 1
First joint (in radians): 1
Second joint (in radians): 1
Valid input. Waiting for server reply...
[INFO] [1582222333.565000]: mode: "Joint Position"
node: "Client"
x: 0.0
y: 0.0
z: 0.0
joint1: 1.0
joint2: 1.0
success: "N/A"
position: "First Joint position: N/A, Second Joint position: N/A"
[INFO] [1582222337.611000]: mode: "Joint Position"
node: "Server"
x: 0.841470956802
y: 0.540302276611
z: 1.44147098064
joint1: 1.0
joint2: 1.0
success: "Valid Position"
position: "First Joint position: 1.0, Second Joint position: 1.0"


Type 1 to set joint angles directly or type 2 to set the end effector's position:
```

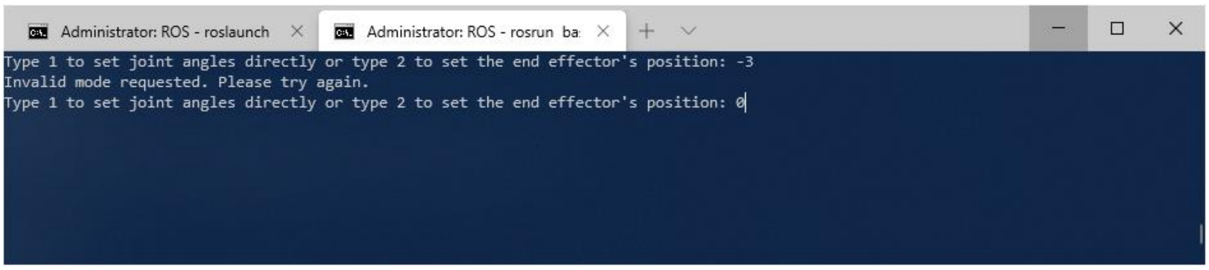
Figure 7: Request valid position (Joint position mode)



Administrator: ROS - roslaunch
Administrator: ROS - rosrun ba:

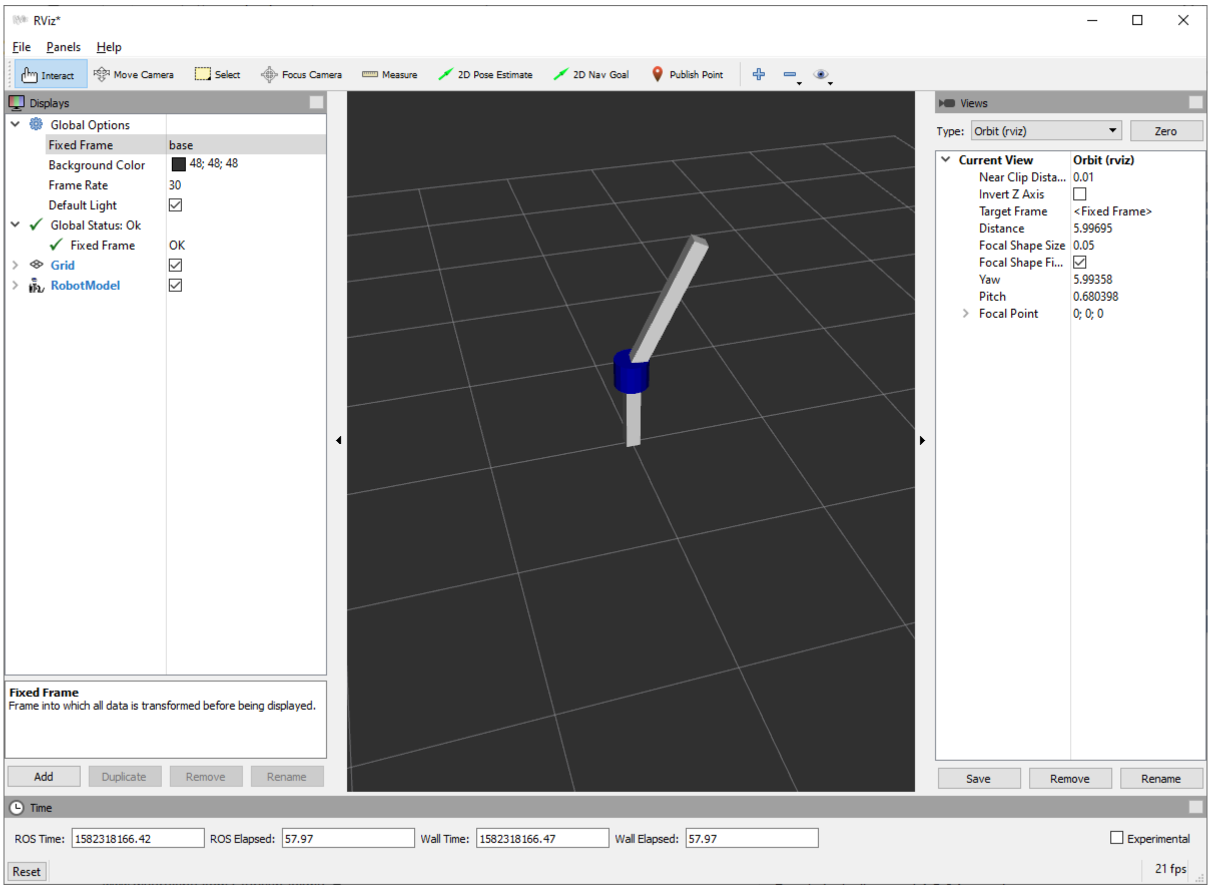
```
Type 1 to set joint angles directly or type 2 to set the end effector's position: "a"
Input must be numeric.
Type 1 to set joint angles directly or type 2 to set the end effector's position:
```

Figure 8: Request non-numeric node (same check used for all inputs)



```
Type 1 to set joint angles directly or type 2 to set the end effector's position: -3
Invalid mode requested. Please try again.
Type 1 to set joint angles directly or type 2 to set the end effector's position: 0
```

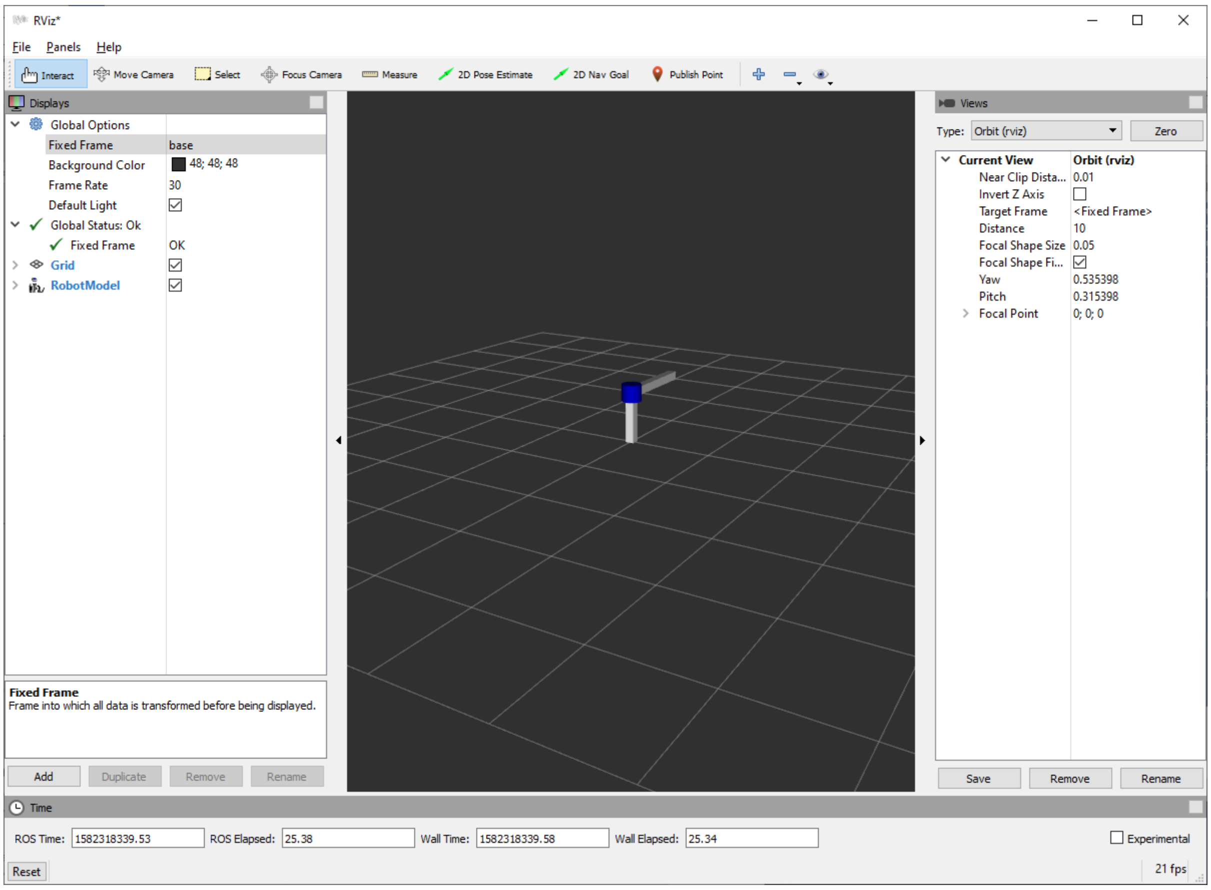
Figure 9: Request invalid mode



RViz*
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Nav Goal Publish Point
Displays
Views
Add Duplicate Remove Rename
Save Remove Rename
Time
Reset

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

Fixed Frame
Frame into which all data is transformed before being displayed.

Type: Orbit (rviz)   Zero
Current View         Orbit (rviz)
  Near Clip Dista... 0.01
  Invert Z Axis      ☐
  Target Frame       <Fixed Frame>
  Distance           5.99695
  Focal Shape Size   0.05
  Focal Shape Fi...  ✓
  Yaw                5.99358
  Pitch              0.680398
  Focal Point        0; 0; 0

ROS Time: 1582318166.42  ROS Elapsed: 57.97  Wall Time: 1582318166.47  Wall Elapsed: 57.97  Experimental
21 fps
```

Figure 10: Robot state after moving



Global Options	
Fixed Frame	base
Background Color	■ 48; 48; 48
Frame Rate	30
Default Light	☑
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	☑
RobotModel	☑

Fixed Frame
Frame into which all data is transformed before being displayed.

Type: Orbit (rviz)

Current View	Orbit (rviz)
Near Clip Dista...	0.01
Invert Z Axis	☐
Target Frame	<Fixed Frame>
Distance	10
Focal Shape Size	0.05
Focal Shape Fi...	☑
Yaw	0.535398
Pitch	0.315398
Focal Point	0; 0; 0

ROS Time: 1582318339.53 ROS Elapsed: 25.38 Wall Time: 1582318339.58 Wall Elapsed: 25.34 ☐ Experimental

21 fps

Figure 11: Robot State after moving (2)