Coursework 2 - Robotic Simulation

Introduction

The ROS (Robot operating system) environment provides a flexible and easily accessible framework to create and write robot software. It encourages collaborative robotic software development and provides many tools, libraries and collections to minimise the complexity involved in creating a robust robotic platform.

Both the classes Robot Model and Robot State are known as the core classes which provide access to a robot's kinematics. The class RobotModel stores all of the information regarding the respective relationships between all the links and joints of the Robot, as well as the joint limit properties from the Universal Robotic Description Format file (URDF). A URDF file is an XML file that is able to describe all the elements of the robot, and it can specify the robots kinematic and dynamic properties provided it is a single isolated robot structure, but it cannot identify this about the robot in the real-world environment. The Robot model class also categorizes the links and joints of the robot into specific planning groups using an SRDF.

The Robot State contains information about the robot at a certain point in time and the join position vectors, velocities and accelerations which are used to obtain kinematic information about the robot. The robot state class also holds helper functions for setting the arm location with respect to the end effector. The Rviz file is a three dimensional tool in ROS applications that is used for visualization. It provides a view of your robot model, capture sensor information from robot sensors, and replay captured data.

There were a few communication protocol methods to choose from while building this robot using the Ros software. One of which is called Publish-subscribe, and in this architecture framework there is a central part called a broke and this is responsible for receiving and distributing all of the data. Publish-subscribe clients are able to publish data to

the broker and subscribe to get data from the broker. The client that publishes data is only sending the data through when it has changed, and this client which is a subscriber to the broker also receives the data when it has changed. This data is not stored by the broker - it is just moved between the publisher and subscriber as and when it is needed.

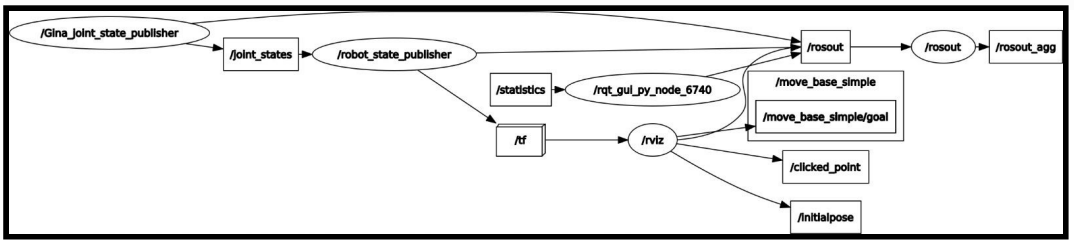
Another network that exists is a client/service system and this is what a ROS service is. A ROS service works on the basis that a client sends a request and blocks this request until a response has been received. ROS services are usually used for quick computational tasks. There is a two-way method of communication, where a message is the request and another message is the response. A ROS service server can only exist as a singularity but a server is able to have many clients.

For the purpose of building this specific robot model a Publisher/subscriber network protocol was used. Publisher/subscriber is quite flexible as you can subscribe to multiple things and publish various things. A client/server is more suitable when explicit requests for specific data is required, but in this robot models situation the movements of the robot do not need to be recorded and can be updated many times, this is why publisher/subscriber was the best method to use.

A robot state publisher was used as part of creating this robot structure, and it subscribes to a topic that is called joint_states (Gina_joint_state_publisher). The joint_state is the controller in this robot model case. The robot state publisher is a subscriber and it subscribes to joint_states which is a topic. Sensor_msgs allows access to the topic joint_states contains a message called JointStates. JointStates contains information such as position, value, velocity and the name of the joints, when the publisher is created we override the value, velocity and effort.

The index of the named joints corresponds to its specific position within the robots structure, values are accepted from the user that corresponds to the index, Std_msgs allows the most updated points and position of the robot in the virtual environment to be viewed.

THe information in the message is used to calculate the forward kinematics and the RViz is updated accordingly.



A diagram outlining the sequence of the data flow is shown above. Gina_joint_state publisher give data to the join_states node and then the robot_state_publisher, and then tf **f** which is a package that lets the user keep track of multiple coordinate frames over time conducts statistics to transform string input data into float coordinates and this is published to the Rviz file to update the robot in the ROS environment.

Methods

The robot was designed using the urdf XML file called GinaRobot. In this the structure of the robot is initialised, and the file 'my robot manipulator' is linked in through a ROS xacro file. The first link is the base link which is fixed to stabilize the robot, made of a cylinder with a radius of 0.8 and a length of 0.15. The link is named and initialized, and its geometry is set to cylinder with the dimensions and visual parameters specified. My robot design had 3 DoF - design composition allows the joints to rotate around each of the x,y and z axis. The robot consists of five joints with 6 links in total.

Then joint 1 is initialized and set to be a revolute joint, and named base_link_link_1 to specify the positioning of this joint between the base link and the first link and it is set to move around the z axis. The upper and lower limit of the joint is st between -3.14 and 3.14

to ensure the robot does not collide with itself once it attempts to change position according to the user input, the parent link is set as the base_link and the child link is link 1.

The first link after the base link is then created, it is also a cylinder but has a radius of 0.35m and a length of 0.4m. Then the joint my_first_link1_link1 between link 1 and 2 is created and set to revolve along the y axis. Link 2 is then initialized to be joined to this aforementioned link, the set shape for link 2 is a box with x/y/z dimensions of 0.2, 0.4 and 0.8. Link2_link3 is created and set as revolute in order to revolve around the y axis. The same steps as above are repeated for link 3 and 4 as well as the joints link2_link3, link_3_link4 and link4_link5. The final link (end effector) is a small cube/box with dimension set as 0.1, 0.1, 0.1 for the length, width and height (x/y/z).

The GinaJointState code is the publisher and it starts off by having the relevant packages and classes imported in. Rospy is imported as well as JointState(velocity/position/effort) from sensor_msgs and Header from sts_msgs (to publish a message). Within the function def talker the class publisher is initialised with the message to be sent between the publisher and subscriber set out below. To instantiate the first joint a JointState instance called jointstate1, the header file is used to create a header for jointstate1 and another header file is initialised which is called joint1_header.stamp and this file contains a timestamp to tell you what time the message was published to ensure the subscriber recieved the most recent message from the publisher.

A while loop is initialised and set to work as long as rospy is still running, and within this loop all 5 joints are set up to receive and respond to user input. In joint1_name all of the joint names to be used are listed in order. The raw input received is a string input from the user about the desired coordinates, a transformation occurs and is stored as a New coordinate which has the corresponding float coordinates. The same process is repeated for Joint 2 and and limits of 1.571 and -1.571 is set to ensure the robot does not over extend itself past the origin or far enough to hit itself. An error message is set up to print if the user enters values outside of this domain. This process is repeated for joints one to 5.

Validation

Figure 1 below shows the initialised position of my robot. This test ensures that when the ROS environment is first loaded in the robot is always initialized to 0,0,0 origin.

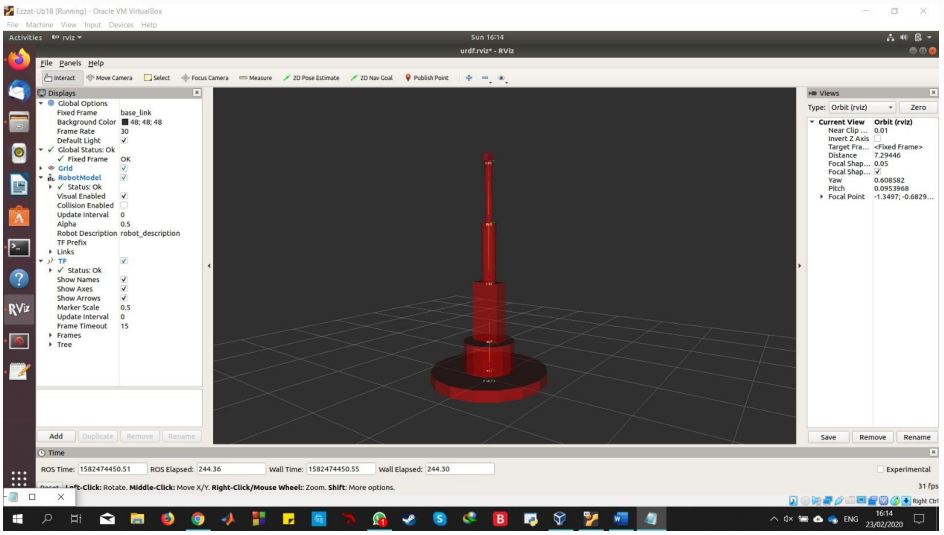


Figure 1 shows the starting position of my loaded robot design.

Figure 2 shows the robots finalised position when the user input received is 0 at joint 1, 0.5 at joint 2 and 1 at joint 3 and 1 at joint 4.

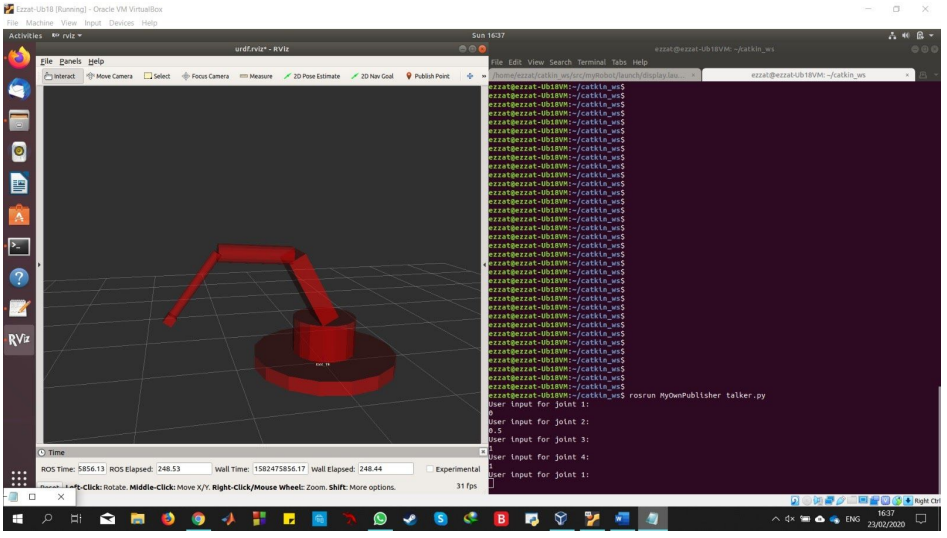


Figure 2 shows the position of my loaded robot design at [0,0.5,1,1].

Figure 3 shows the robots maxed out position when a few user inputs received: 0 at joint 1, 20 at joint 2 and 0.3 at joint 3 and -5 at joint 4 and 15 at joint 5. This proved the software responds to incorrect values correctly and prompts the user to fix these accordingly.

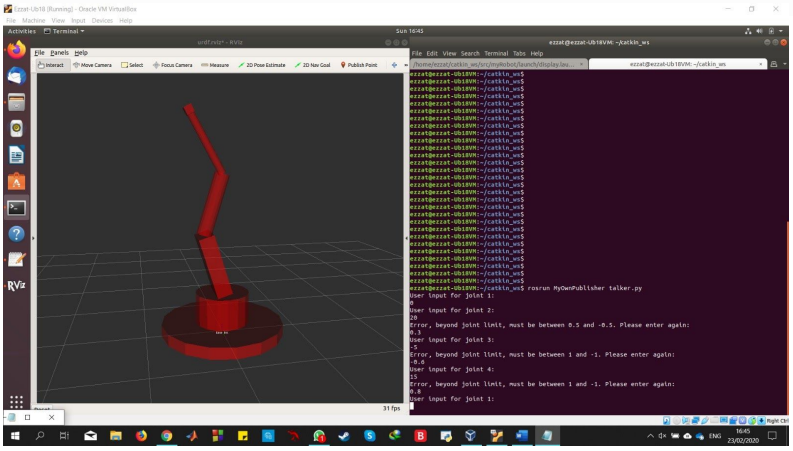


Figure 3 shows the error messages when user input is out of bounds.

Figure 4 shows the robot maxed out at varying 90 degrees, proving it can alter its design successfully without collapsing onto itself.

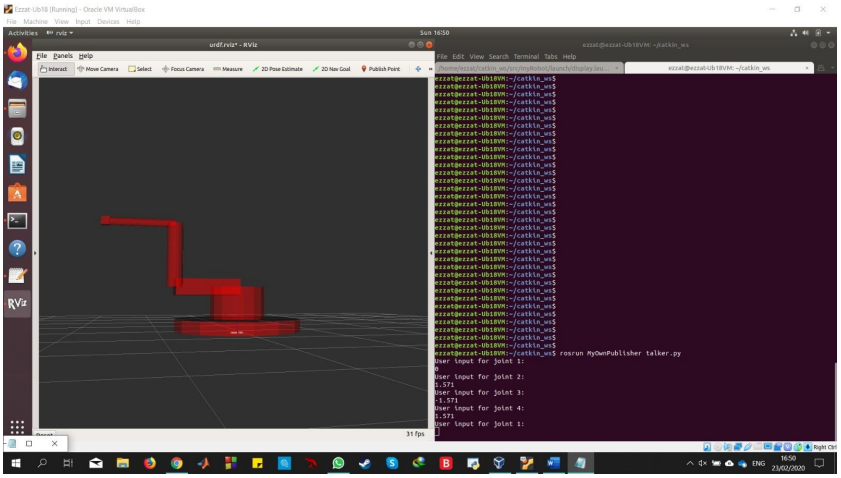


Figure 4 shows the maxed out angles, proving the robot can mould itself at 90 degree angles.

ReadMe

Open a new terminal and run

$ source devel/setup.bash

After this run the launch file

$ roslaunch GinaRobot display.launch.

https://github.com/GSaj2020/CW2-Robot-Model.git