IMPLEMENTATION OF FORWARD KINEMATICS FOR 5-AXIS INDUSTRIAL ARM AND SIMULATION IN ROS

TAL Manufacturing Sol. Ltd. Internship Report

N.GOWTHAM (15R441)

gowtham27dec@gmail.com

PSG COLLEGE OF TECHNOLOGY

COIMBATORE

TAMILNADU – 641 004

TABLE OF CONTENT

1. INTRODUCTION

The purpose of this project is to implement Forward Kinematics of the 5 DOF industrial arm in ROS. This will be useful for testing and development. The simulation provides easy way to visualize the robot when the joint variables were given. The real time testing will be time consuming and costly. Hence the simulation make things easier.

ROS can be programmed in C++ and Python and most of the time combination of both. This project was programmed using C++ and utilized the feature like “dynamic reconfigure” to set the joint angles in real time and “Rviz”

1.1. Robot Operating System

ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management.

1.2. rviz

rviz (ROS visualization) is a 3D visualizer for displaying sensor data and state information from ROS. Using rviz, you can visualize Baxter's current configuration on a virtual model of the robot. You can also display live representations of sensor values coming over ROS Topics including camera data, infrared distance measurements, sonar data, and more.

1.3. NODE

A node is a process that performs computation. Nodes are combined together into a graph and communicate with one another using streaming topics, RPC services, and the Parameter Server. These nodes are meant to operate at a fine-grained scale; a robot control system will usually comprise many nodes. For example, one node controls a laser range-finder, one Node controls the robot's wheel motors, one node performs localization, one node performs path planning, one node provide a graphical view of the system, and so on.

The use of nodes in ROS provides several benefits to the overall system. There is additional fault tolerance as crashes are isolated to individual nodes. Code complexity is reduced in comparison to monolithic systems. Implementation details are also well hidden as the nodes expose a minimal API to the rest of the graph and alternate implementations, even in other programming languages, can easily be substituted.

2. SOFTWARE CONFIGURATION

2.1. Dynamic reconfigure

The angle values should be configured by the user at any time. The node which publishes the joint states to the robot joints will act as a server for the Dynamic reconfigure client. This system acts like a server client type of communication.

Dynamic reconfigure uses a custom defined data type (.cfg file). The data type in this case is a vector with names with axis names and values with angle values.

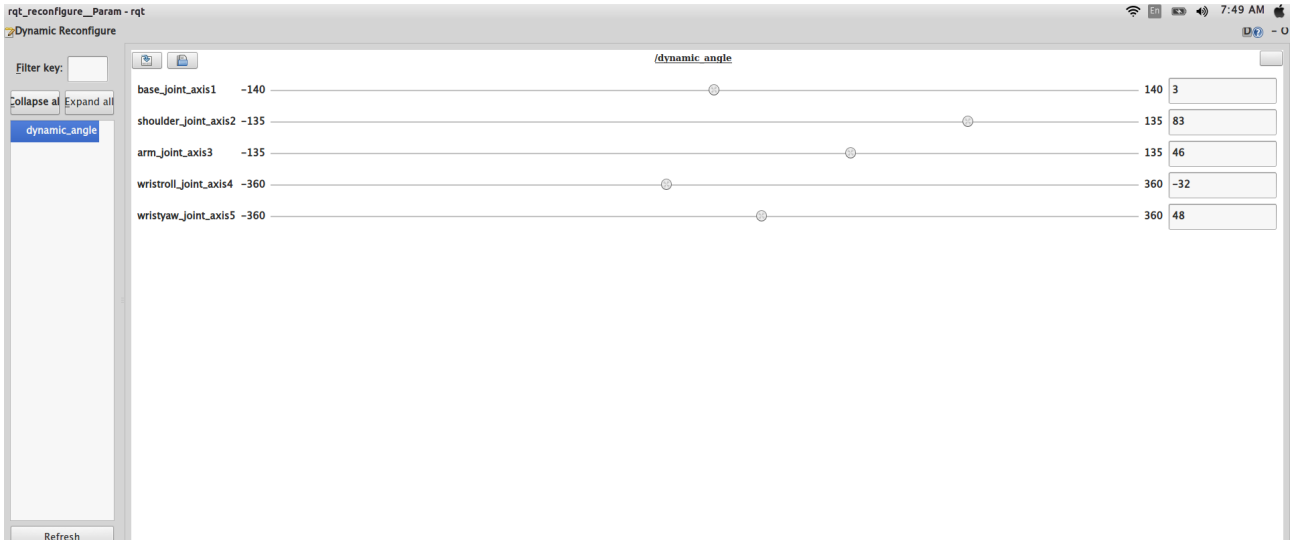

Figure 1: Slider to set joint angles

```
gowtham@gowtham: ~
gowtham@gowtham:~$ rostopic echo /dynamic_angle/parameter_updates
bools: []
ints:
  -
    name: base_joint_axis1
    value: 3
  -
    name: shoulder_joint_axis2
    value: 83
  -
    name: arm_joint_axis3
    value: 46
  -
    name: wristroll_joint_axis4
    value: -32
  -
    name: wristyaw_joint_axis5
    value: 48
strs: []
doubles: []
```

Figure 2: Joint vector with joint names and angle values

2.2. Dynamic angle node

This node is the main node which takes angle input from the dynamic reconfigure and publishes the angle in radian to rviz. Because rviz can only take joint angle input in radians. The node is programmed in c++ for better performance over python. The joint angles were published on the topic /joint_states .

The angle vector description

- **name** - a string which specifies the joint name under which this parameter should be stored
- **type** - defines the type of value stored, and can be any of int_t, double_t, str_t, or bool_t
- **description** - string which describes the parameter
- **default** - specifies the default value
- **min** - specifies the min angle value
- **max** - specifies the max angle value

Figure 3: Joint angles in degree

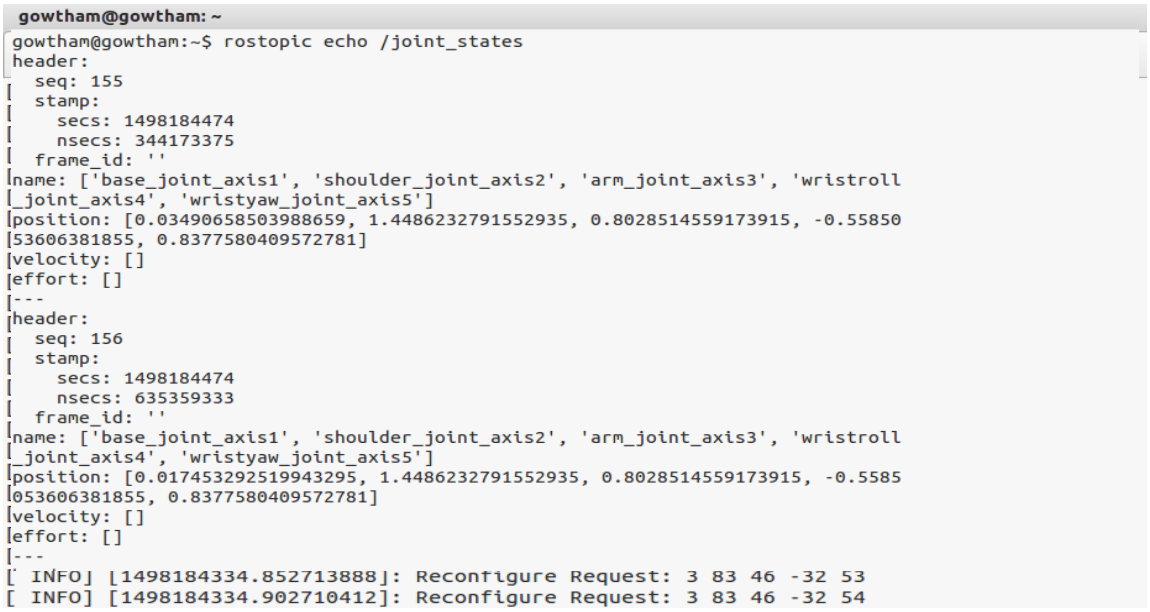

Figure 4: Joint angles in radian

2.3. URDF

The Universal Robotic Description Format (**URDF**) is an XML file format used in ROS to describe all elements of a robot. The solid works model can be exported to URDF file which can be visualized in rviz. The URDF can represent the kinematic and dynamic description of the robot, visual representation of the robot, and the collision model of the robot.

Link:

The link tag represents a single link of a robot. Using this tag, we can model a robot link and its properties. The modeling includes size, shape, color, and can even import a 3D mesh to represent the robot link. We can also provide dynamic properties of the link such as inertial matrix and collision properties.

Joint:

The joint tag represents a robot joint. We can specify the kinematics and dynamics of the joint and also set the limits of the joint movement and its velocity. The joint tag supports the different types of joints such as revolute, continuous, prismatic, fixed, floating, and planar.

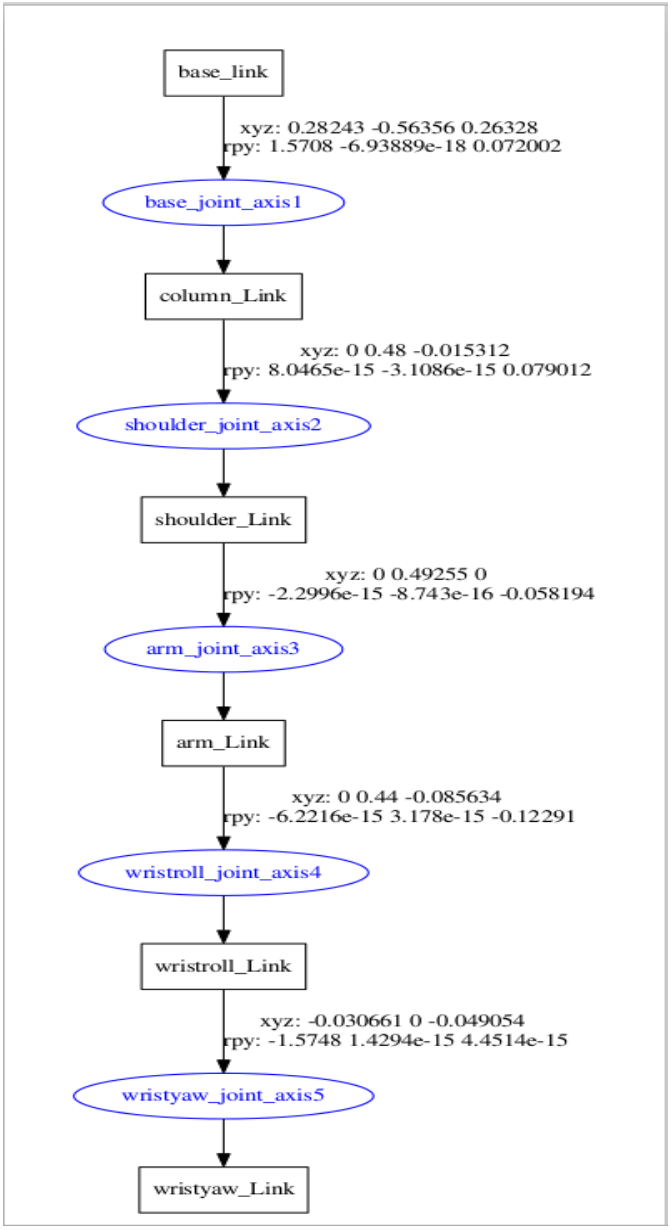

Figure 5: The urdf tree of the industrial arm

2.4. Solid works assembly to URDF

The links of the robot were drawn in solid works software. The links are assembled. By using the SW2URDF plug-in the assembly can be converted into URDF model. The limits of each joints were configured during the export.

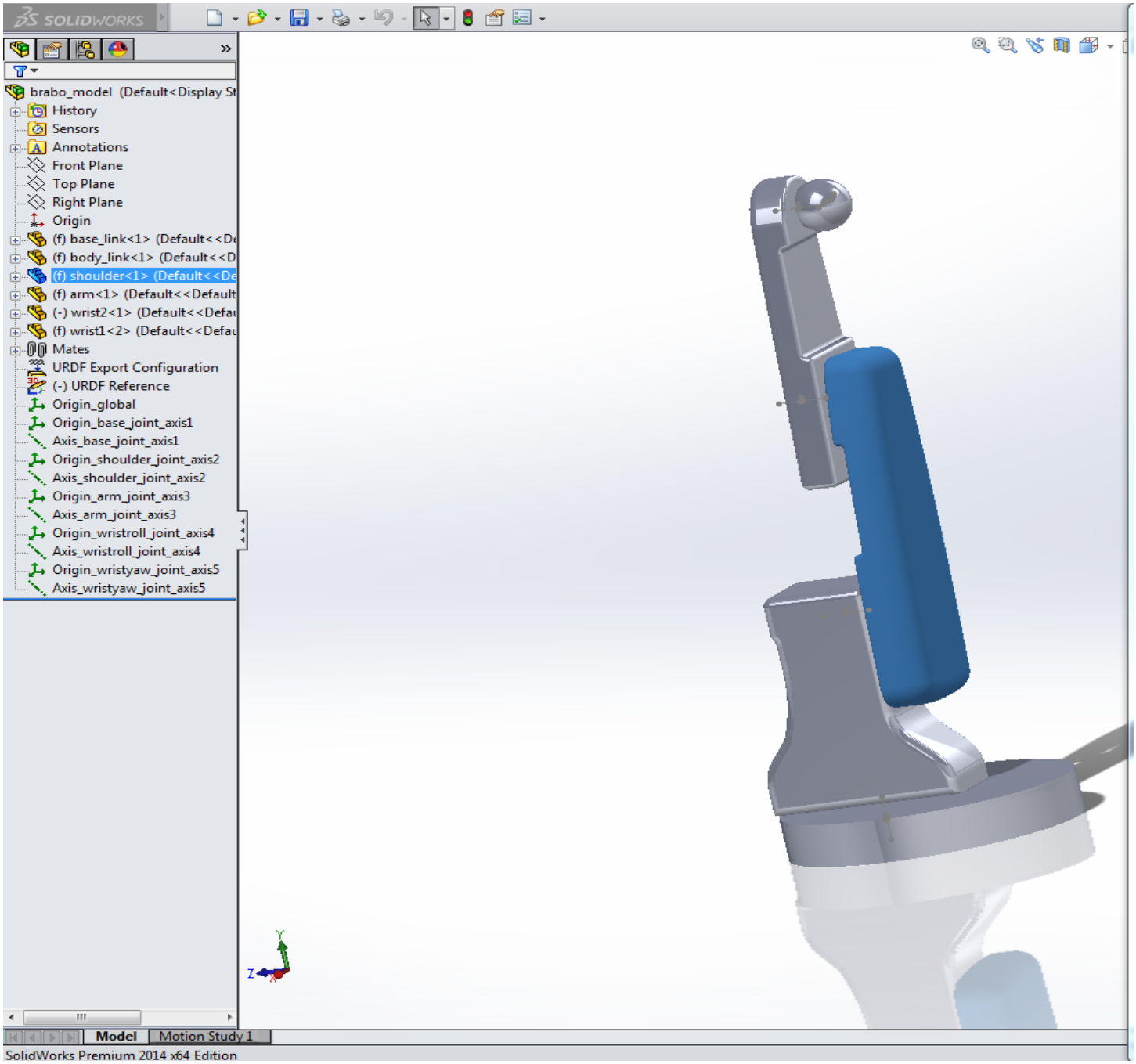

Figure 6: Model of the industrial arm

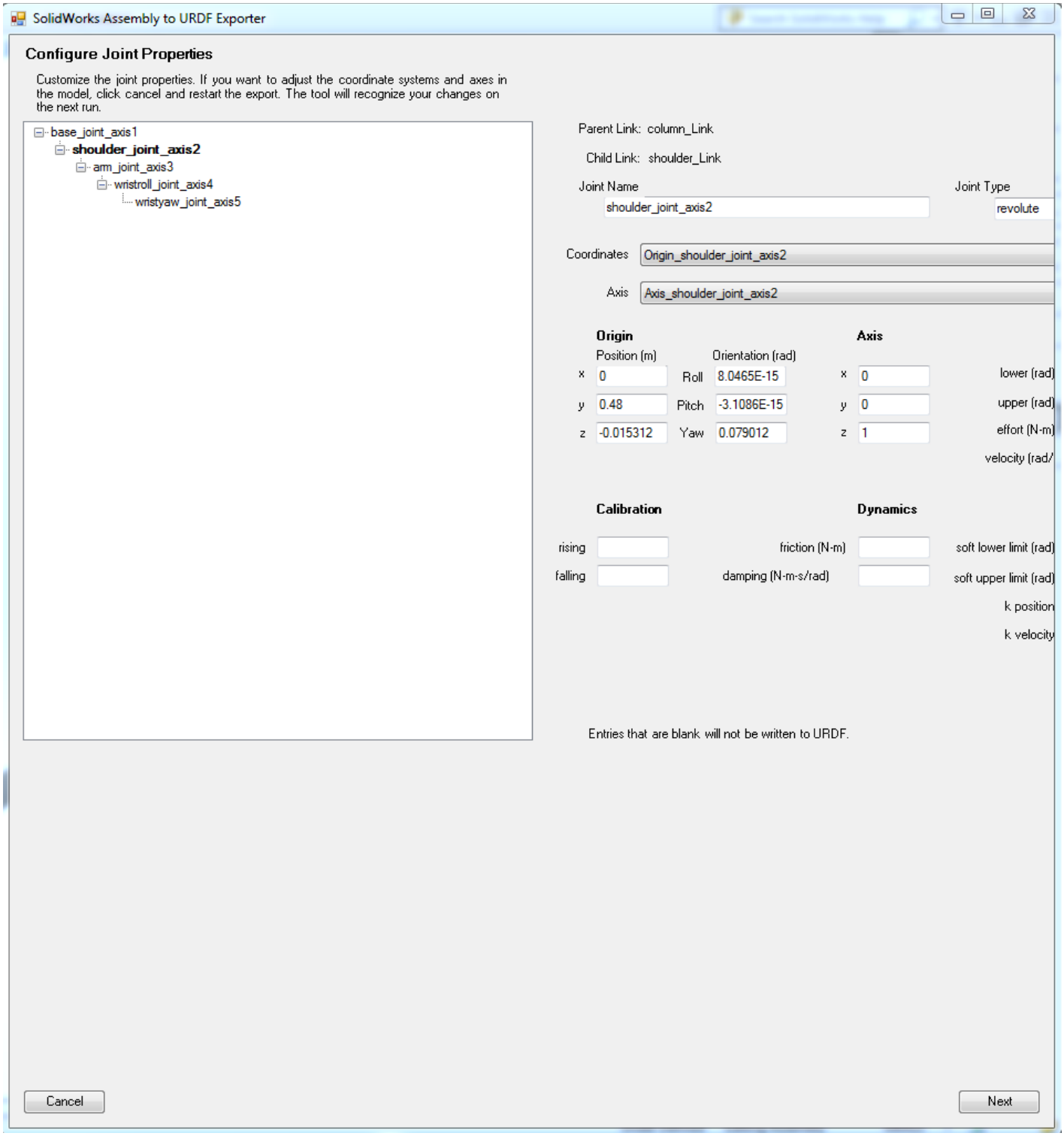

Figure 7: SW2URDF configuration window

2.5. VISUALIZATION

roscore

The roscore node is invoked. The roscore is the master node which manages all the nodes and makes the communication between the nodes.

Command: `$roscore`

rviz

rviz and the visualization can be opened by executing the following command

Command: `$source devel/setup.bash`

`$roslaunch brabo_model/brabo.launch`

dynamic_angle

The main node can be invoked by the following command

Command: `$source devel/setup.bash`

`$roslaunch dynamic_angle angles.launch`

arduino_motor_control

The robot_motor_control.ino is uploaded to arduino board using the arduino IDE. This makes the real time control of the robot possible. Arduino takes the angles in degrees and publishes pulses to the stepper motors.

Command: `$rosrun serial_python serial_node.py`

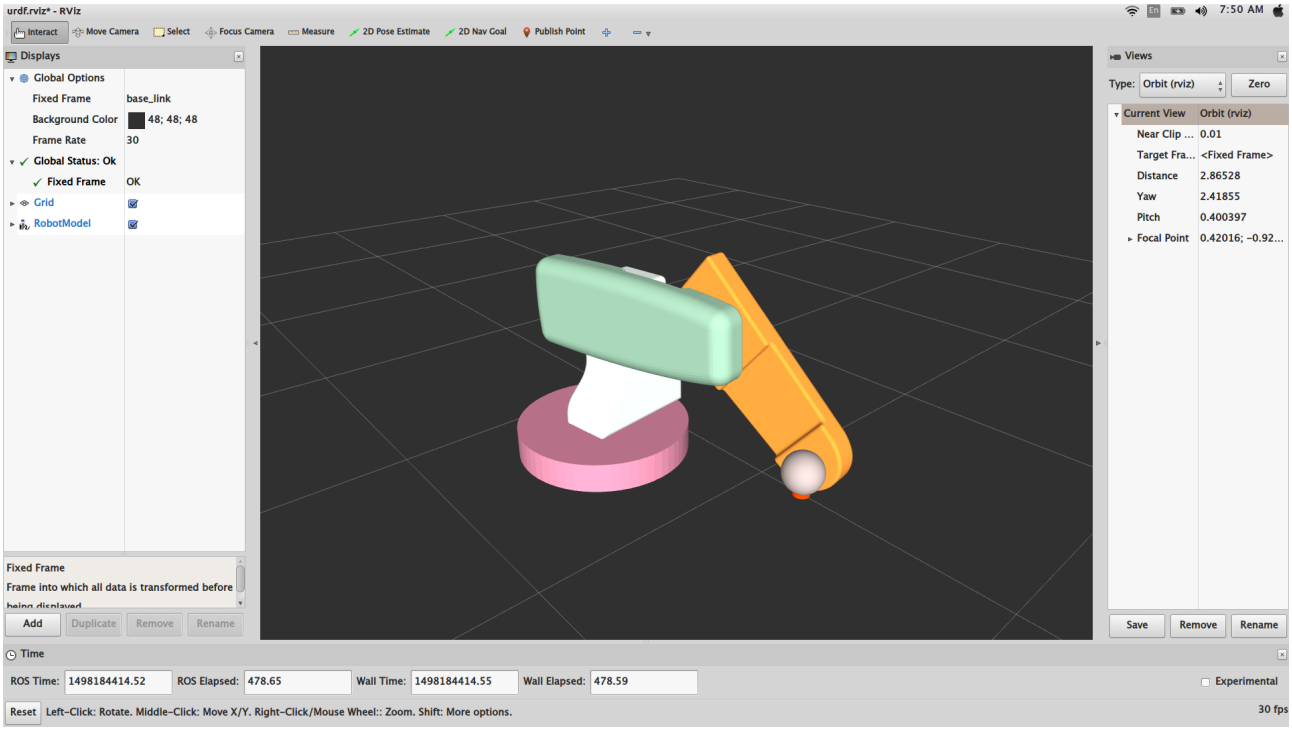

Figure 8: Simulation in rviz

3. BLOCK DIAGRAM

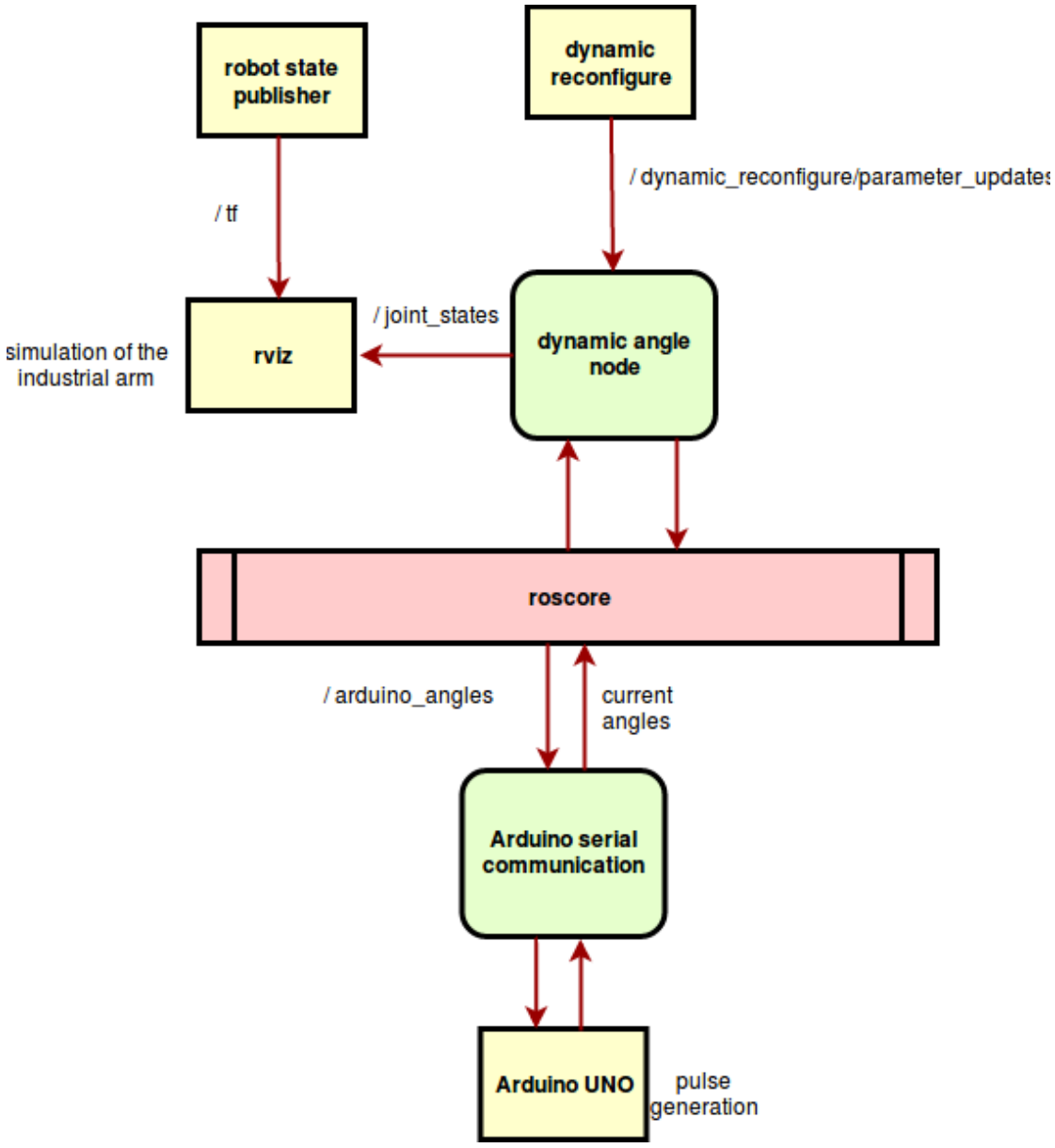

Figure 9: Block diagram of the system