

The
University
Of
Sheffield.

Automatic Control &
Systems Engineering

ACS6503: Manipulator Robotics Laboratory Sheet 3

Updated: 04 December 2023

Contents

1.	Introduction	3
2.	Laboratory Procedures.....	4
2.1	Familiarisation with the Environment	4
2.2	Beginning Using ROS.....	5
2.3	Beginning Using ROS Part II	8
2.4	Launching the Mover6 within ROS	8
2.4	Linking Matlab with the Mover6 Driver	10

1. Introduction

This lab exercise is to extend the previous work in operating the Mover6 robot arm. In this lab we will continue to control the arm, however, we will use the Robot Operating System to allow us to program arm and interface to other computational environments.

Dr Jonathan Aitken

Module Leader

jonathan.aitken@sheffield.ac.uk

2. Laboratory Procedures

In this laboratory you'll be using a mixture of different control methodologies for simulating and controlling the Mover6 arm. Further information can be found on this arm in the documentation which is available through Blackboard. You should familiarise yourself with this information before you begin using the equipment.

2.1 Familiarisation with the Environment

The Robot Operating System (ROS) is a commonly used piece of middleware that provides access to a wide range of tools that allow robots to be simulate, programmed, and accessed through appropriate drivers. Typical applications can be written in C++ or Python, or through connected applications like Matlab/Simulink and the Robot Systems Toolbox.

There is a substantial amount of information available on ROS. It is worthwhile to review some of the online resources:

- The main Robot Operating Systems webpage: <http://wiki.ros.org>; this details the packages that are available and details how to setup the middleware, configure the environment and provides a collection of basic tutorials.
- In this lab we are using the Mover6. This arm has an existing Github page (https://github.com/CPR-Robots/cpr_robot)

The desktop machines you have been provided with have an install of Ubuntu, with the Robot Operating System configured with a development stack already in place. As this is a shared computing resource you will need to backup your files at the end of the lab, and remove them from the configured stack. You will be using Ubuntu during this lab.

In order to start the robot and connect it to the machine we first need to run a setup file (this will also enable ROS, as part of the command it will ask for the password for the machine in the lab this is "acselab"):

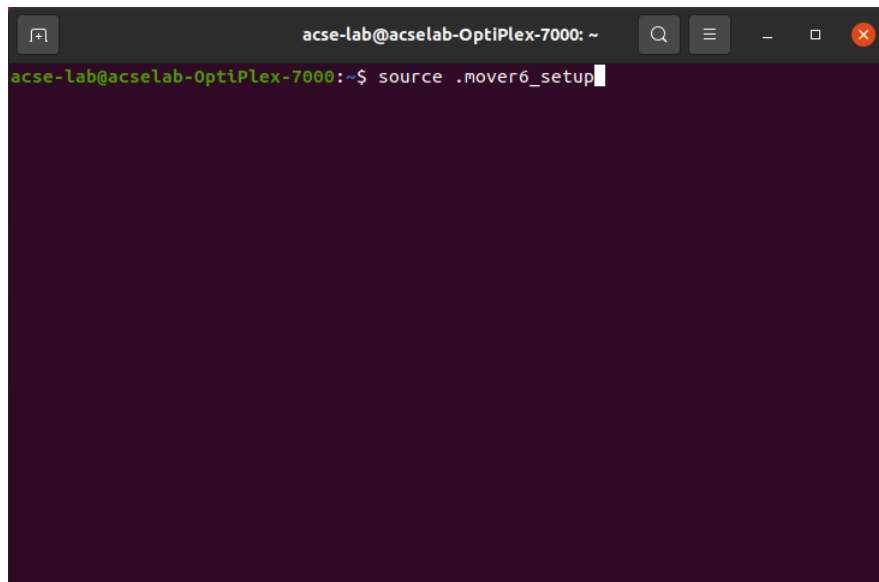
A terminal window with a dark background and light text. The window title bar shows 'acse-lab@acselab-OptiPlex-7000: ~'. The terminal content shows the command 'acse-lab@acselab-OptiPlex-7000:~\$ source .mover6_setup' followed by a cursor. The terminal is otherwise empty.

Figure 1: Robot and ROS setup Command

You will need to do this in each terminal window that you use or ROS will not be enabled

Note your workspace is catkin_ws2 – You will not be able to use catkin_ws, compile operations are not permitted there

2.2 Beginning Using ROS

To start the Rosmaster issue the command “roscore” within the terminal. Once issued the terminal will look as indicated in Figure 2. This shows that the master is in operation, and gives some basic information about the version of Ros that is running (1.14.3) and the Distribution (Melodic).

```
File Edit View Search Terminal Help
roscore http://jonathan-VirtualBox:11311/
jonathan@jonathan-VirtualBox:~$ roscore
... logging to /home/jonathan/.ros/log/b49fdeb0-1840-11ea-b189-080027967b39/roslaunch-jonathan-VirtualBox-5561.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://jonathan-VirtualBox:39401/
ros_comm version 1.14.3

SUMMARY
=====
PARAMETERS
 * /rostdistro: melodic
 * /rosversion: 1.14.3

NODES
auto-starting new master
process[master]: started with pid [5571]
ROS_MASTER_URI=http://jonathan-VirtualBox:11311/

setting /run_id to b49fdeb0-1840-11ea-b189-080027967b39
process[rosout-1]: started with pid [5582]
started core service [/rosout]
```

Figure 2: Starting the Rosmaster

Once the master is started we can begin to start running nodes. A node is essentially a piece of executable code that runs, however, communication with the master is added in so that it can interact with other nodes that are running. Typically we can write nodes in C++ or Python. In this lab we’ll be using C++ for development, however, through the communication protocols it is just lightweight XML that is exchanged therefore the languages are interoperable.

Open a new terminal window, and let’s start the turtle simulator (as shown in Figure 3). This is done by using the “roslaunch” command. As can be seen in Figure 3 the “roslaunch” command takes two arguments. The first is the name of the development stack that contains the node (“turtlesim”), the second is the name of the node to start (“turtlesim_node”). It is often useful to remember that the tab key can be used to autocomplete the stack and node name. Indeed it can be used to complete all elements typed at the command line.

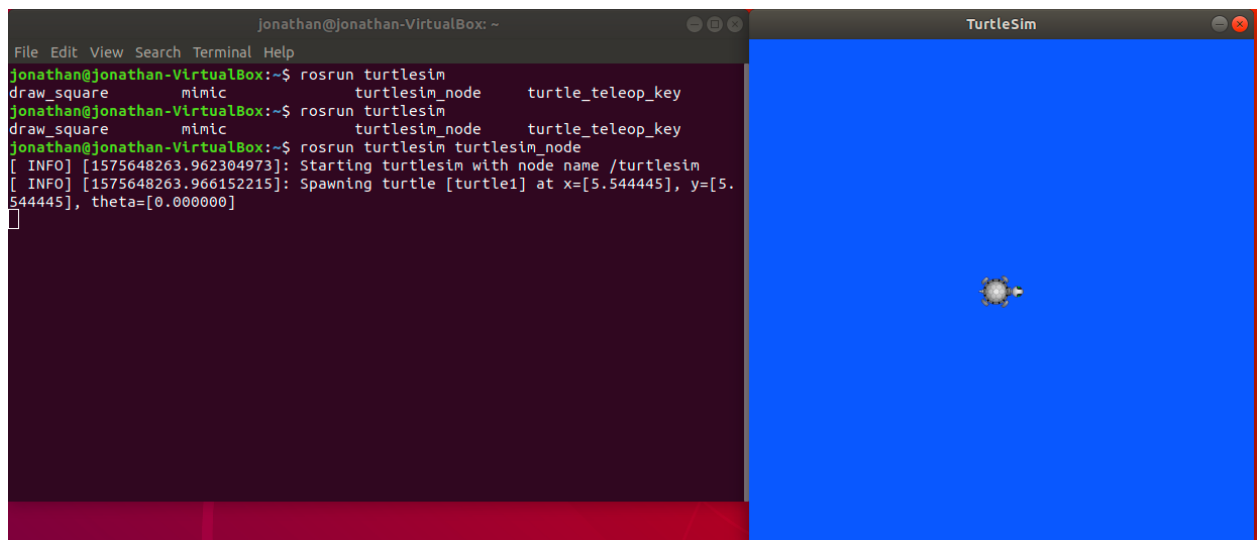


Figure 3: Starting the Turtlesim node

Let's now start a controller for moving the turtle around, this can be done starting a teleop command, as shown in Figure 4:

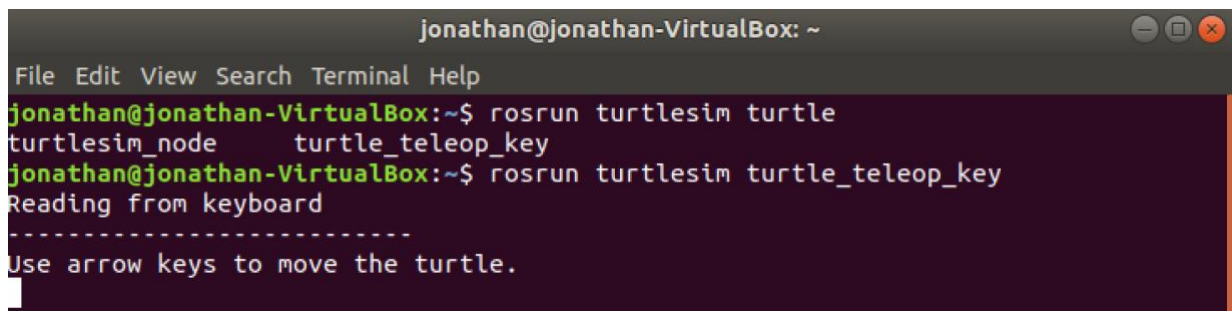


Figure 4: Starting the Keyboard Controller

This connects a node that takes keyboard inputs and translates them into messages sent on a command topic, we can find some more information about this by issuing the commands shown in Figure 5. This shows that a total of 5 topics have been created. The topic we are interested in is `/turtle1/cmd_vel`, if we look for some more information on this we can see that it is of type `geometry_msgs/Twist` and that it is subscribed to by the simulator, and published to by the our teleoperation node.

```
jonathan@jonathan-VirtualBox: ~
File Edit View Search Terminal Help
jonathan@jonathan-VirtualBox:~$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
jonathan@jonathan-VirtualBox:~$ rostopic info /turtle1/c
/turtle1/cmd_vel      /turtle1/color_sensor
jonathan@jonathan-VirtualBox:~$ rostopic info /turtle1/cmd_vel
Type: geometry_msgs/Twist

Publishers:
* /teleop_turtle (http://jonathan-VirtualBox:41829/)

Subscribers:
* /turtlesim (http://jonathan-VirtualBox:33187/)
```


Figure 5: Topic Information

If we look at the information posted on this topic when we use the arrow key to move the turtle (as shown in Figure 6). This shows two things, when we press an arrow key this node sends a message to the simulator. The form of this message is fixed, in this case it is a “geometry_msgs/Twist”, which is formed up of 6 components, a linear and rotation movement about the X-, Y-, and Z- axes.

If we send a message on this topic of the correct type we can then cause the turtle to move; ROS can send messages like this from the command line as well as in nodes, shown in Figure 7. In general we could write a new node to command the turtle and publish information as we require it!

```
jonathan@jonathan-VirtualBox: ~
File Edit View Search Terminal Help
jonathan@jonathan-VirtualBox:~$ rostopic echo /turtle1/
/turtle1/cmd_vel      /turtle1/color_sensor /turtle1/pose
jonathan@jonathan-VirtualBox:~$ rostopic echo /turtle1/c
/turtle1/cmd_vel      /turtle1/color_sensor
jonathan@jonathan-VirtualBox:~$ rostopic echo /turtle1/cmd_vel
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
---
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0
```

Figure 6: Inspecting the Topic

A terminal window titled 'jonathan@jonathan-VirtualBox: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the command 'rostopic pub /turtle1/cmd_vel geometry_msgs/Twist "linear: x: 2.0 y: 0.0 z: 0.0 angular: x: 0.0 y: 0.0 z: 0.0"' being entered. The output shows the message being published and latched, with a prompt to press ctrl-C to terminate.

```
jonathan@jonathan-VirtualBox: ~  
File Edit View Search Terminal Help  
jonathan@jonathan-VirtualBox:~$ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist "linear:  
t "linear:  
  x: 2.0  
  y: 0.0  
  z: 0.0  
angular:  
  x: 0.0  
  y: 0.0  
  z: 0.0"  
publishing and latching message. Press ctrl-C to terminate
```

Figure 7: Manually Publishing

2.3 Beginning Using ROS Part II

Task 1:

In general <http://wiki.ros.org/ROS/Tutorials> provides a good basis for beginning to use ROS.

As an orientation complete sections 2, 3, 4, 5, 6, 7, 10 and 11. Section 1 has been completed for you on the host machines. These section will begin to orient you with ROS, and with both topics and services.

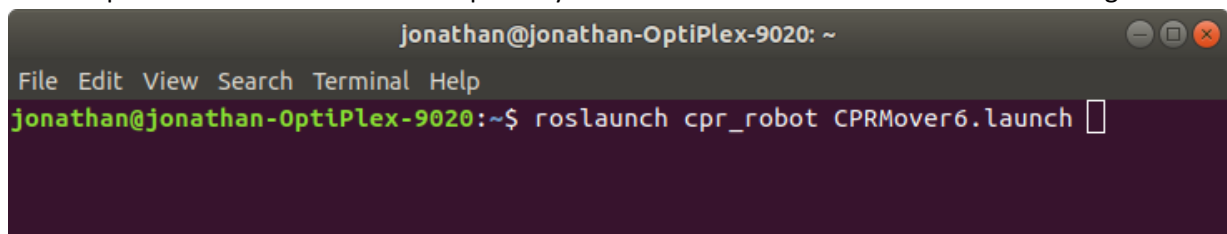
As an addendum to this it is beneficial to also follow the tutorials for connecting Matlab nodes to ROS: <https://uk.mathworks.com/help/ros/ug/get-started-with-ros.html>

The remainder of this lab exercise will be given as a mini task which links together the previous labs undertaken, with a basic knowledge of ROS.

2.4 Launching the Mover6 within ROS

The Mover6 package can be launched easily within ROS. The workspace on the machines have been pre-configured to connect to the can bus for communication with the arm.

NB: When you start a terminal the first time after a boot, you will be asked to enter the password for the computer. Once this has been completed you can start the Mover6 driver as shown in Figure 8.

A terminal window titled 'jonathan@jonathan-OptiPlex-9020: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the command 'roslaunch cpr_robot CPRMover6.launch' being entered.

```
jonathan@jonathan-OptiPlex-9020: ~  
File Edit View Search Terminal Help  
jonathan@jonathan-OptiPlex-9020:~$ roslaunch cpr_robot CPRMover6.launch
```

Figure 8: Starting the Mover6 Driver in ROS

This will start RVIZ, as show in Figure 9. This is a visualisation of the robot, but will connect to the robot once the "Enable" button is pressed; as can be seen the sliders and "STOP" buttons are greyed out indicating their inactivity.

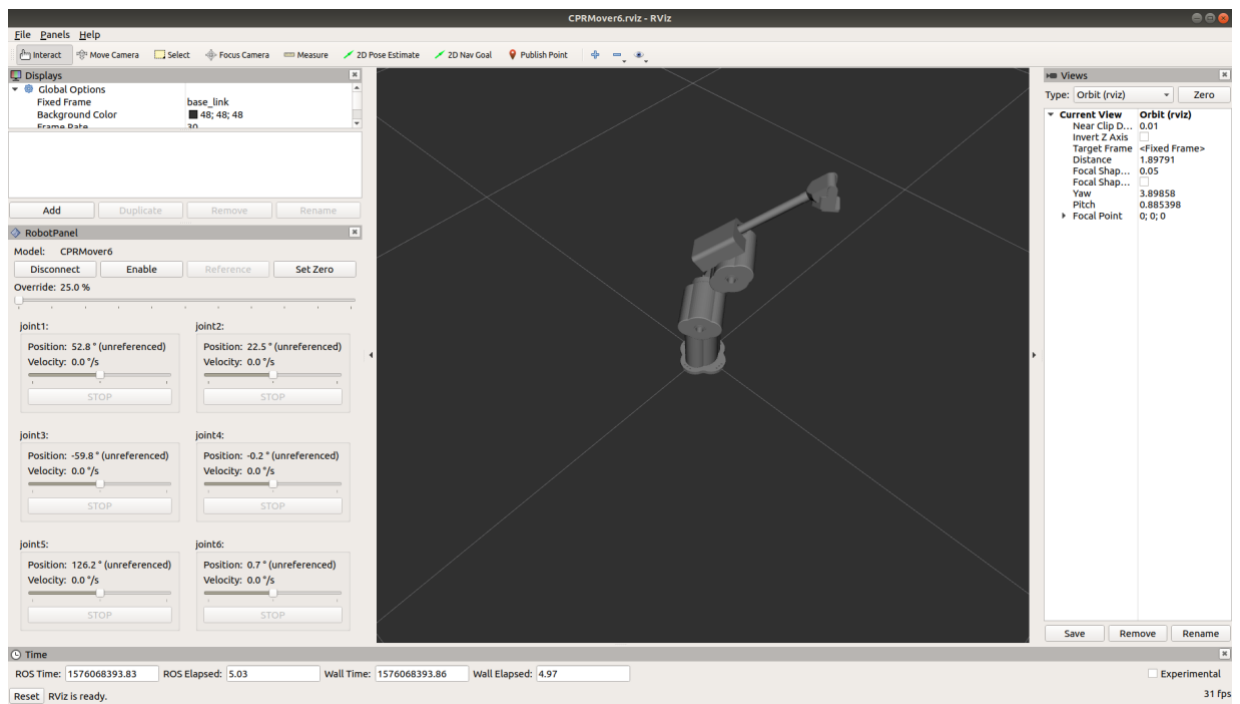


Figure 9: RVIZ Started

The robot can be enabled by pressing the “Enable” button. This will activate the motors as shown in Figure 10. These sliders now provide velocity control of the joints, with the joint positions indicated in each panel.

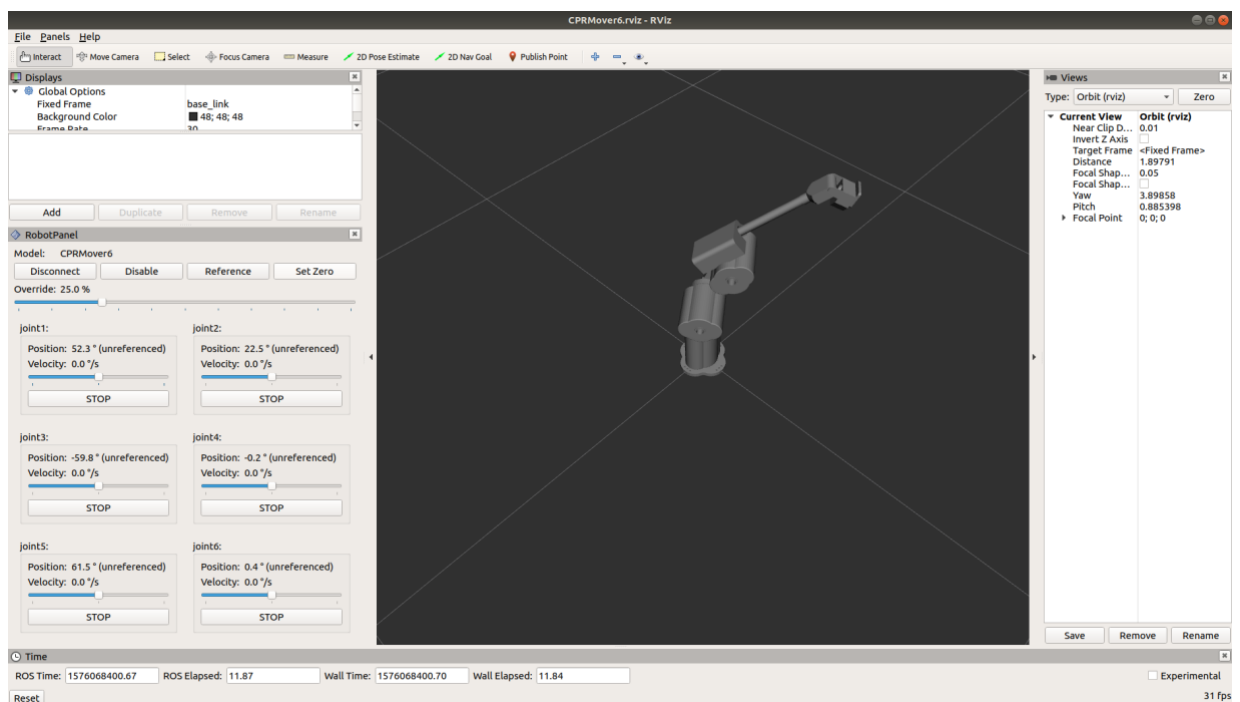


Figure 10: Mover 6 Active

Task 2:

Master the arm using the same process shown in Lab 2 to set the joint positions. Note that the sliders do not move the joints to an angle, but set a joint velocity. Therefore you will have to use a low override setting and press stop when the joint is in the correct position.

When correctly setup into the home position (see Lab 2 notes, Task 3) press “Set Zero”.

2.4 Linking Matlab with the Mover6 Driver

The main exercise for this lab is to compute the joint angles for the Mover6 using the Inverse Kinematics process developed in Lab Session 1. This will then be communicated to a control node in ROS, you have been provided with a basic node which can set the angle of Joint 1, you will need to expand this node to set the angle for all six joints. Once this has been completed this node can be extended to take communication from the node generating the inverse kinematics within Matlab.

Task 3:

Begin by inspecting the topics produced by the Mover6 driver you will see that there is a topic which accepts a `control_msgs::JointJog` message. The sample stack you have been provided with (“basic_movement”) contains a source file (“GoalMovementMover6”). You can build this file by copying the stack into the “src” folder of the workspace and issuing the “`catkin_make`” command in the top level.

Observe that joint 1 of the arm moves to the desired angle; this can be seen in the RVIZ window monitoring the Joint angle. This process can be repeated by jogging joint 1 to a different position and re-running the node.

Adjust the code within this node so that it is capable of setting all 6 joint angles and verify that it sets them correctly.

NB: Carefully set the gains of each line so that maximum velocities aren’t exceeded. This gain is represented by the value 0.25 in the line:

```
msg_start.velocities.push_back(0.25*(joint1_demand-joint1)/abs(joint1_demand-joint1));
```

NB: Ensure that you keep the E-Stop Button by you so that you can cut power to the robot in any situation where the motion is not as you expect.

If you need to use the E-Stop close all terminal windows, release the E-stop and then re-open one and launch the RVIZ Mover6 tool to reconnect.

At this stage the joint angles of the arm can be correctly set from within the code. Now we need to expose the joint angles to a ros topic so that we can set them from outside the system.

Task 4:

1. Extend the code in “GoalMovementMover6” by creating a new callback function which receives a `std_msgs/Float32MultiArray.msg`, process this message using similar code to the “jointsCallback” function, to extract 6 variables “joint1_demand”, “joint2_demand”, “joint3_demand”, “joint4_demand”, “joint5_demand” and “joint6_demand”.

Hint: The iterator in jointsCallback uses a double type; consider the base data type you are using for this message

2. Declare a new subscriber which connects to a topic “/joint_demands” and points to the callback in step 1.

3. Setup a Boolean variable that will only trigger movement once a new demand has been received.

4. Verify from the command line that your new callback function correctly sets the joint angles of the robot using the command line to generate the message.

NB: This can be done using the line (setting joint1, joint2, joint3, joint4, joint5, and joint6 to values in the range $-\pi/2$ to $\pi/2$ appropriately:

“rostopic pub /joint_demands std_msgs/Float32MultiArray '{data: [joint1, joint2, joint3, joint4, joint5, joint6]}'”

You should set the arm to a random configuration by using the RViz tool and then command the arm back to a joint position, e.g. the home position:

“rostopic pub /joint_demands std_msgs/Float32MultiArray '{data: [0, 0, 0, 0, 0, 0]}'”

The arm is now setup to take an array which can be passed in from a ROS Topic. This array will be an array of 6 floats which represents the demand joint angles. Obviously we need an inverse kinematic solver to generate the joint angles. However, this can be done from Matlab.

Task 5:

Connect Matlab to ROS, so that Matlab connects as a node within your existing network running the Mover6 driver. You should be able to see the topics on the Matlab command prompt using the command “rostopic list”.

1. Create a new matlab function using the skeleton file CartesianMover6.m using the process that you used to calculate the inverse kinematics in Lab 1.

Hint: X, Y and Z displacements form up standard transformation matrices. A, B and C are the Euler angles or rotation about Z, Y, and X respectively. Therefore if you calculate one transformation matrix for each component and multiply together in the order XYZABC – you will obtain the final desired end effector p[osition and pose which can be fed into the inverse kinematics solver to give you a set of joint angles.

2. Package this up into a ROS message of type `std_msgs/Float32MultiArray` and broadcast it to you “GoalMovementMover6” node. Verify that the joints go to the angles that you expect.

