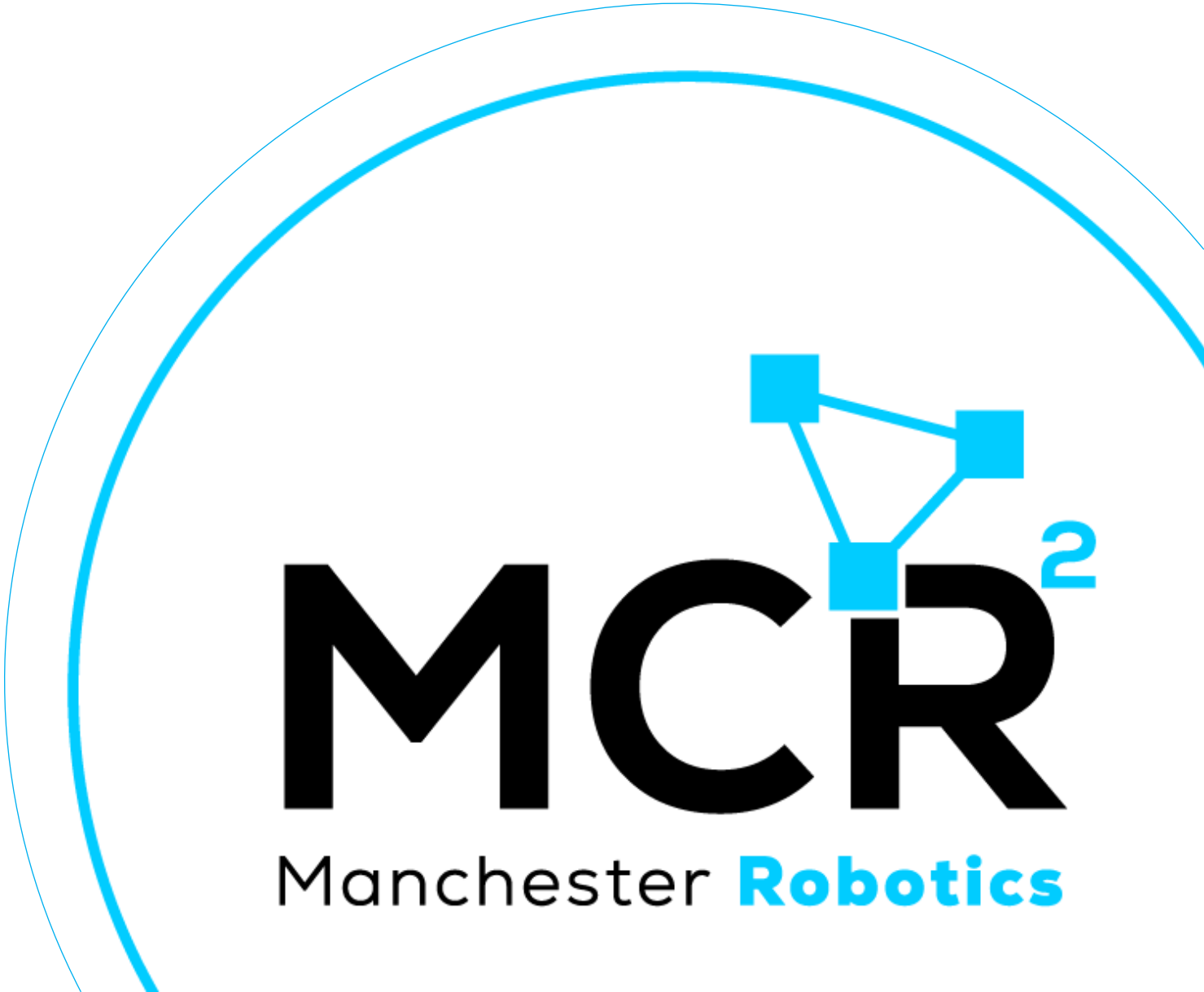
{Learn, Create, Innovate};

RVIZ Examples

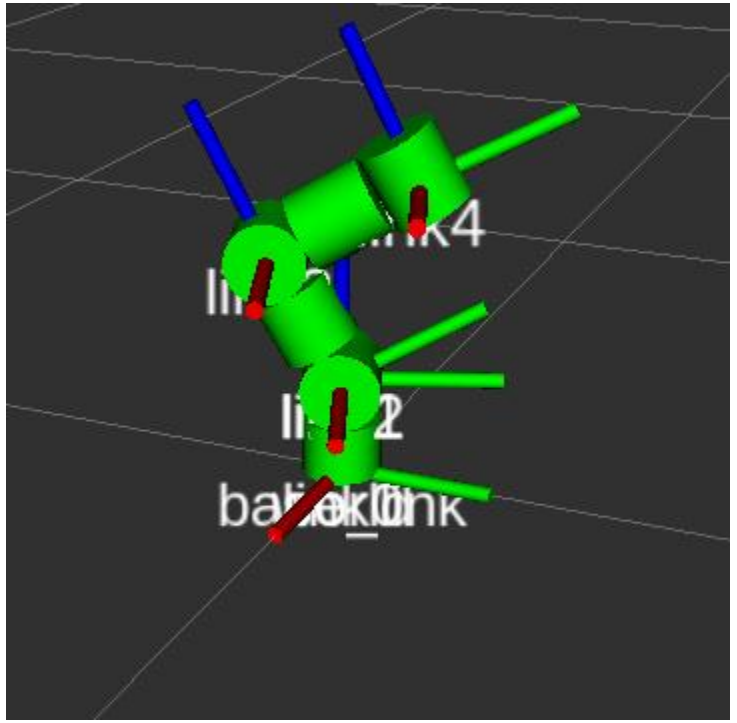
Manipulator Modelling in RVIZ



Introduction

Introduction

- In this tutorial, the user will build a simple robot manipulator, skeleton using basic markers from RVIZ.
- This tutorial will guide the user through all the necessary steps to simulate the manipulator using the concepts of transforms and markers in ROS.



Creating the package

Requirements

- Ubuntu in VM (MCR2 VM) or dual booting
- ROS installed (if not follow the steps in this link and select full installation)
- Workspace "catkin_ws" created following the steps here (if you are using the VM this is already done for you).

Creating a package

- Create a package called "*simple_manipulator*". The dependencies used are *rospy*, *std_msgs*, *geometry_msgs*, *visualization_msgs tf2_ros tf_conversions*. Open a terminal and type the following

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg simple_manipulator std_msgs rospy
  geometry_msgs visualization_msgs tf2_ros tf_conversions
  tf2_geometry_msgs
```

Beware that the command "`catkin_create_package`" must be run inside the "src" folder.

- Once the package is created you will be able to see the package folder in ~/catkin_ws/src.
- Build the package you just created and add it to your environment (more information here)

```
$ cd ~/catkin_ws
$ catkin_make
$ source devel/setup.bash
```

Creating the package

Creating and Configuring the Node

- Create a folder for the Python scripts that will be used for each node and generate a Python script called “manipulator.py” inside the scripts folder.

```
$ cd ~/catkin_ws/src/simple_manipulator
$ mkdir scripts
$ cd scripts
$ touch manipulator.py
```

- Since the “talker.py” is an executable script, you need give permission to ubuntu to run it.

```
$ sudo chmod +x manipulator.py
```

- Open the file CMakeLists.txt in the folder “~/simple_manipulator/CMakeLists.txt” and find the following line

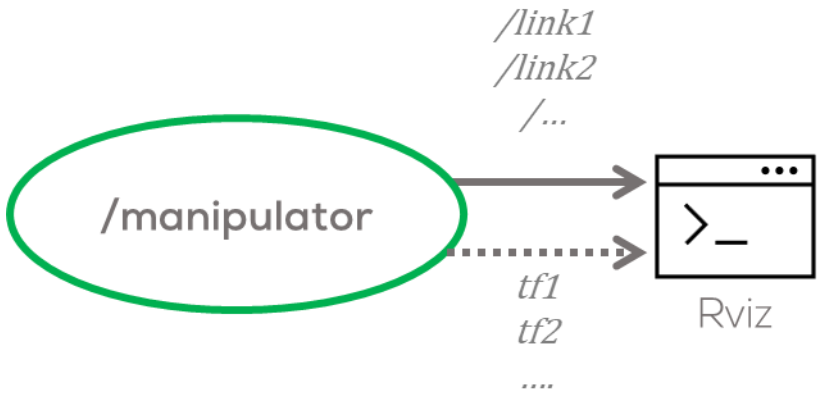
```
catkin_install_python(PROGRAMS scripts/manipulator.py
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)
```

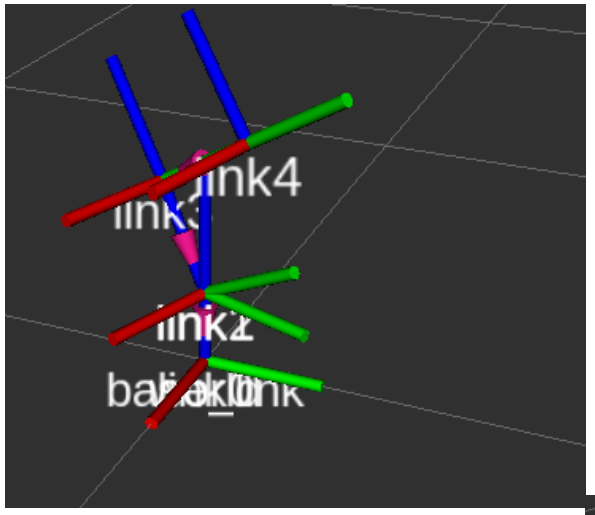
- Uncomment it (remove the #) and add the highlighted line in yellow. This makes sure the python script gets installed properly and uses the right python interpreter

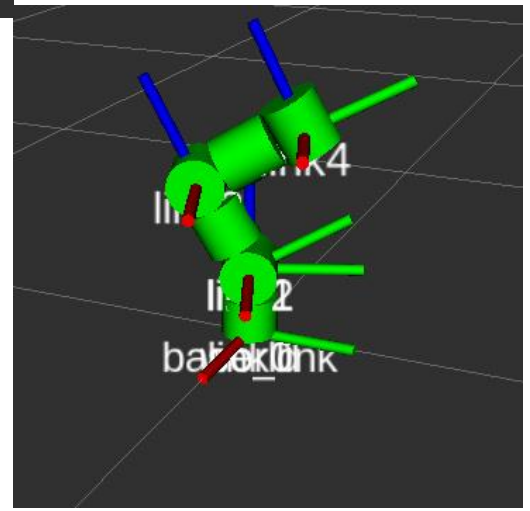
Creating the node

Coding the node

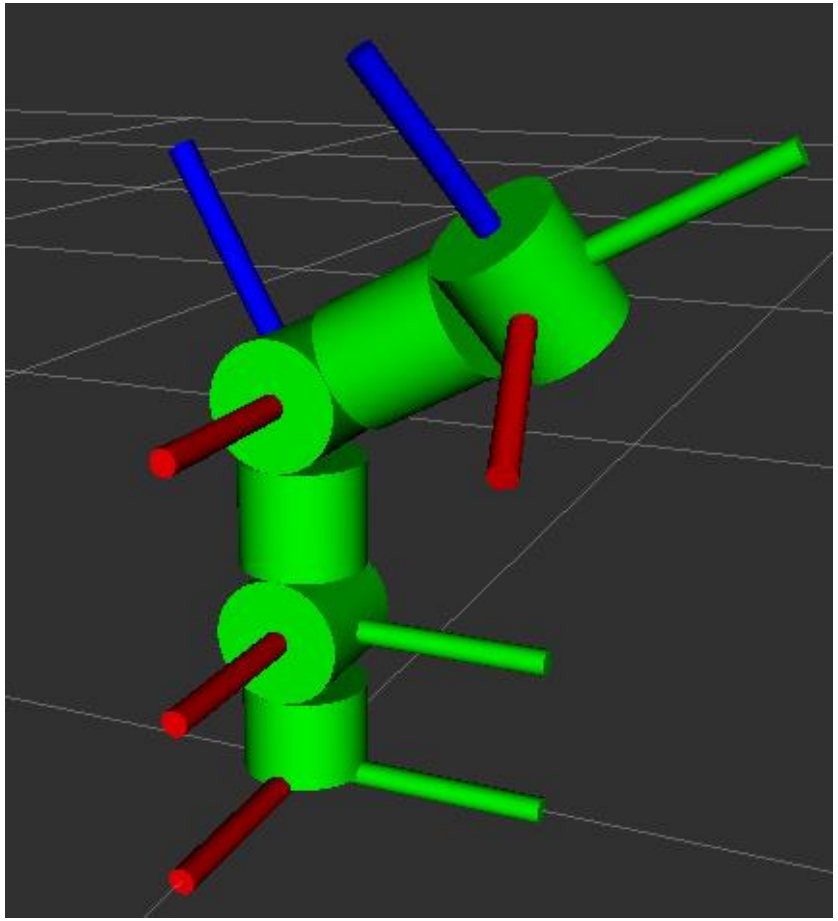
- This node will configure and broadcast/publish the necessary transforms, and visualisation messages to visualise a simple manipulator.
- A graphical representation of this task will look as follows

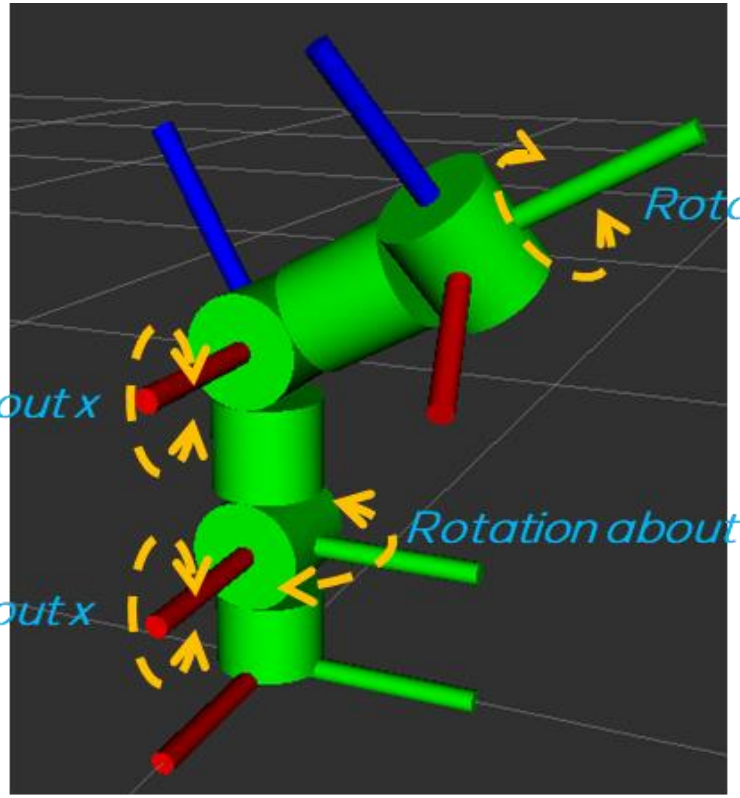


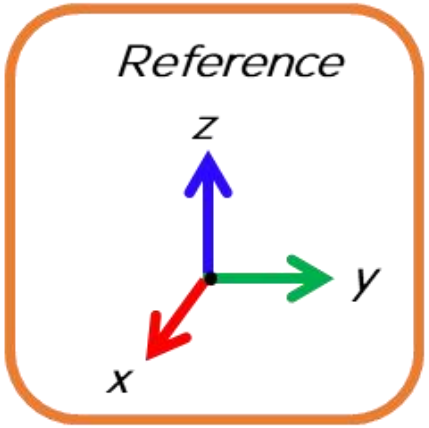




Robot Description







Code

Code Development

- The code development will be divided into two parts.
- In the first part, the previously defined transforms will be coded and tested to make sure they work properly.
- In the second part of this tutorial, the markers necessary for visualising the manipulator will be defined.

Code Development

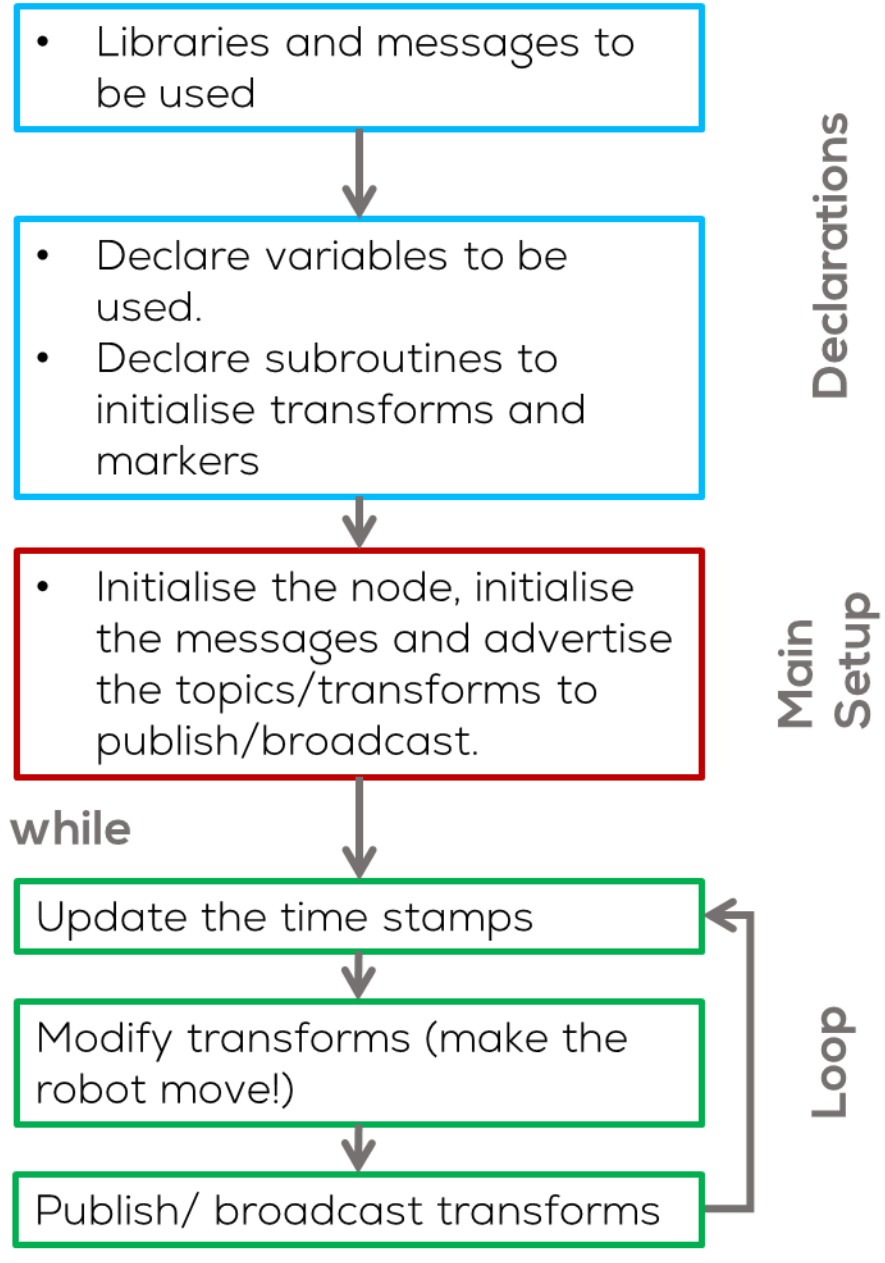
- The complete code structure to be used for this ROS node will be divided into three different sections, declaration, main setup and loop.
  - Declaration: In this section, the user will define the variables to be used, type of messages, transforms, and functions to initialise them.
  - Main setup: In this section the node will be configured, initialised and the publishers and subscribers will be defined.
  - Loop: In this section, the markers and transforms will be updated to make the manipulator move.
- Each of these sections will be filled during this tutorial.

```
#!/usr/bin/env python
import rospy
…
```

```
# Declare Marker and transform messages
base_marker = Marker()
baseLink_l0_tf = TransformStamped()
…

#Declare Transform Messages
baseLink_l0_tf = TransformStamped()
….

#Functions to Initialise markers
def init_base_marker():
          initialise a Marker (marker in visualisation_msgs)
…

#Functions to Initialise transforms
def init_baseLink_l0_tf():
           initialise Transforms
…
```

```
MAIN
     init_node
     loop_rate
     call functions to initialise tf and markers
     Setup publishers and broadcasters
```

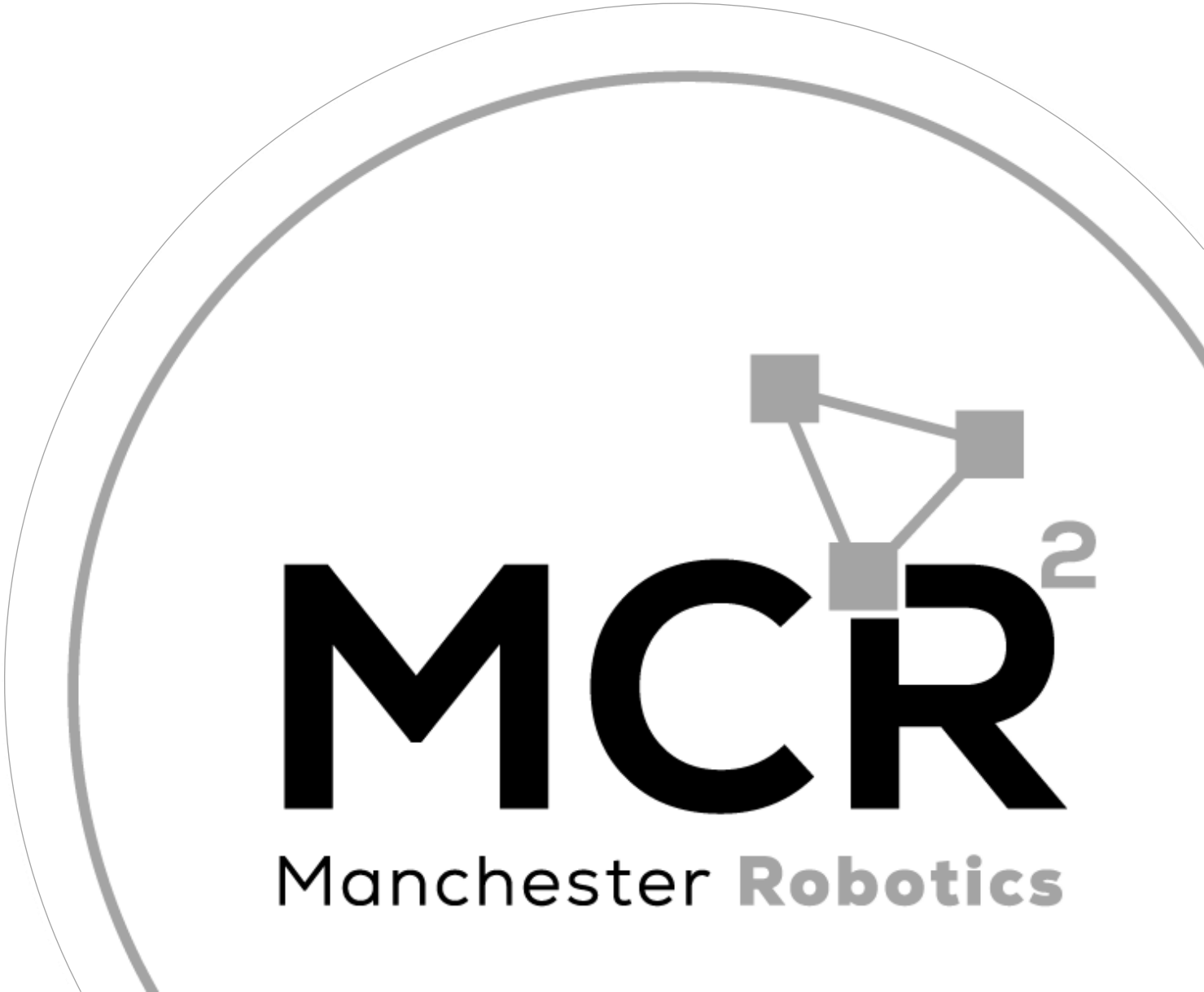
```
WHILE
     update time stamp of the markers
     Modify transforms and update time stamp
     Publish markers
     Broadcast transforms
     SLEEP
```



Simple Manipulator

Part 1: Transforms

{Learn, Create, Innovate};

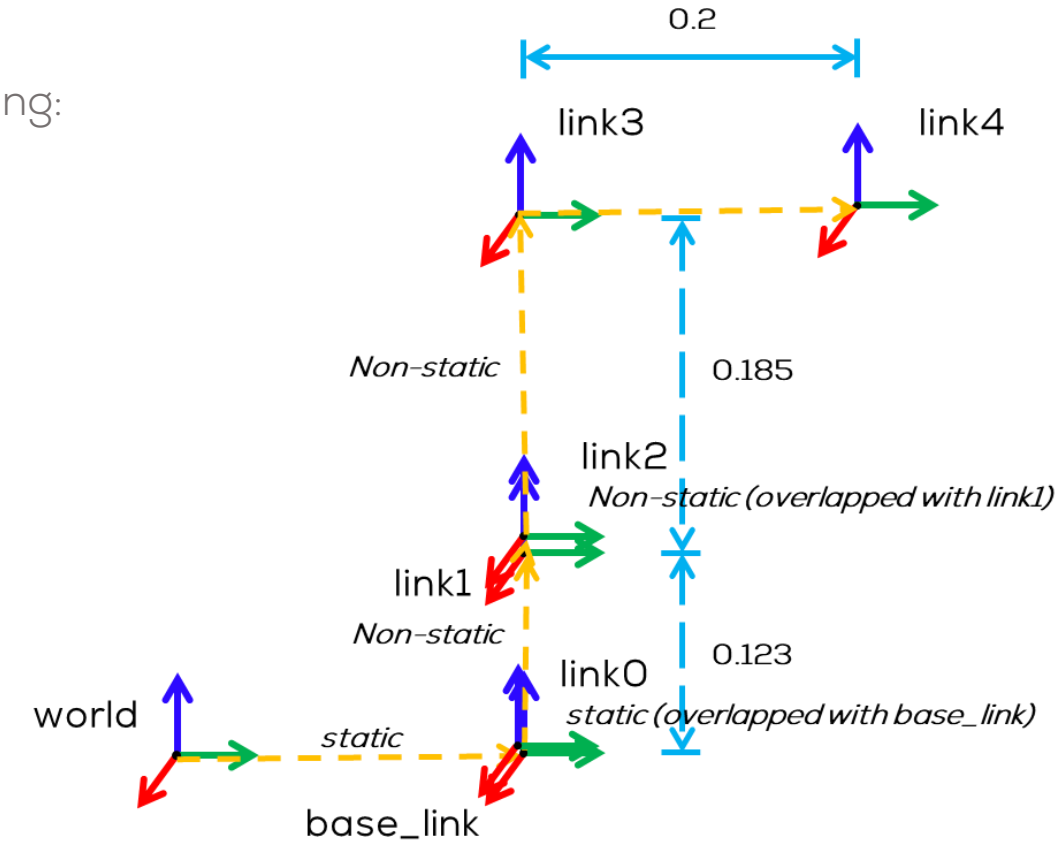


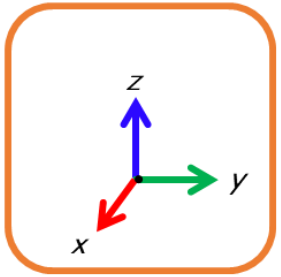
Transforms to be broadcasted

Transforms

The transforms to be defined are the following:

- Transform 1:
  - Father frame: world
  - Child frame: base_link
  - Type: static
  - Translation: x, y, z= user-defined
  - Rotation: User defined
  - Information: Position of the manipulator
- Transform 2:
  - Father frame: base_link
  - Child frame: link0
  - Type: Static
  - Translation: x=0, y=0, z=0
  - Rotation: r=0 , p=0, yaw=0
  - Information = Overlapping base_link



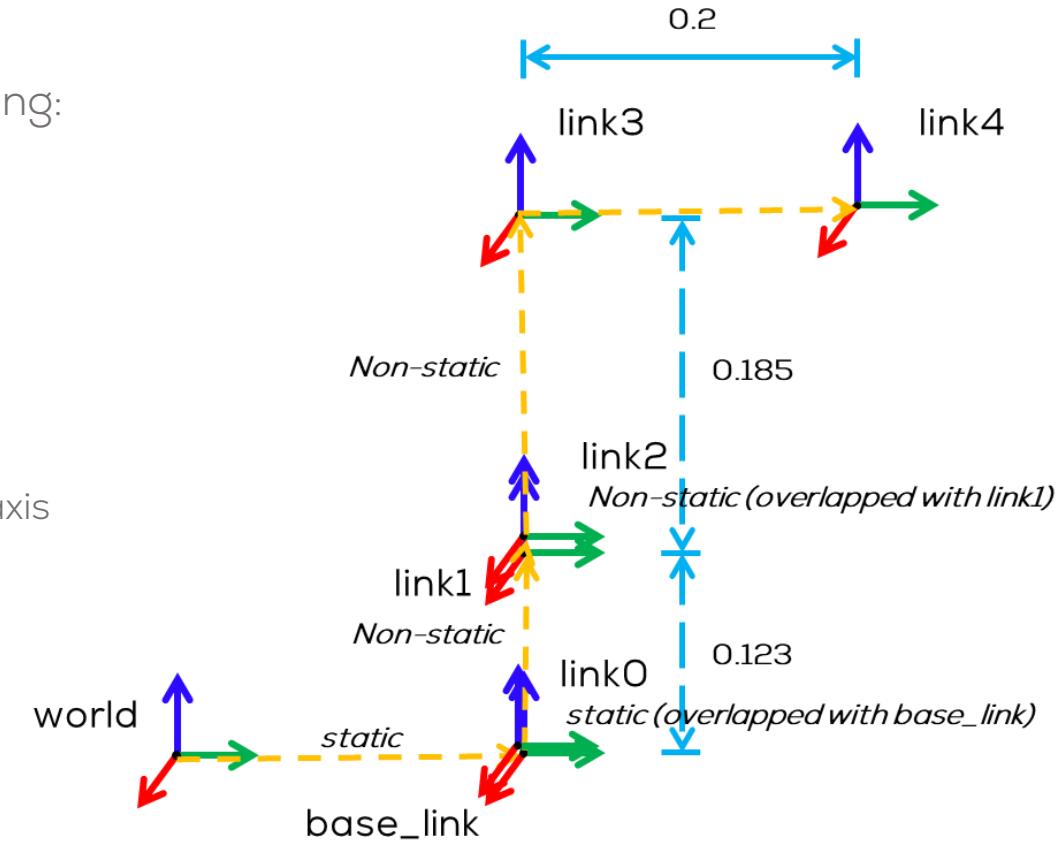


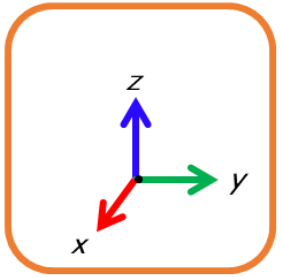
Transforms to be broadcasted

Transforms

The transforms to be defined are the following:

- Transform 3:
  - Father frame: link0
  - Child frame: link1
  - Type: Non-Static
  - Translation: x=0, y=0, z=0.123
  - Rotation: r=0 , p=0, yaw=0
  - Information: Joint 1 rotation about the z-axis
- Transform 4:
  - Father frame: link1
  - Child frame: link2
  - Type: Non-static
  - Translation: x=0, y=0, z=0
  - Rotation: r=0 , p=0, yaw=0
  - Information: Joint 2, Overlapping link1, rotation about the x-axis



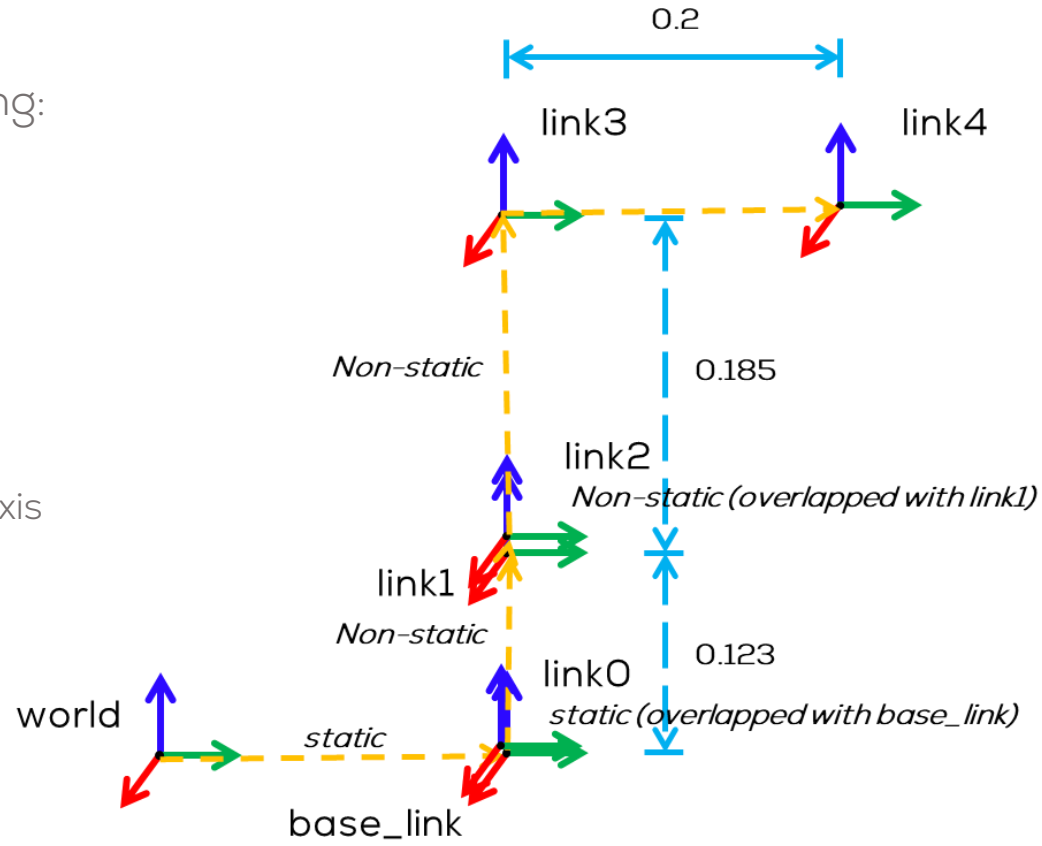


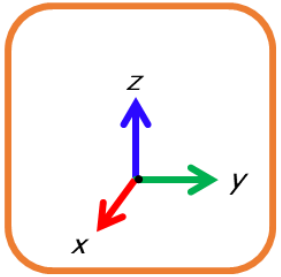
Transforms to be broadcasted

Transforms

The transforms to be defined are the following:

- Transform 5:
  - Father frame: link2
  - Child frame: link3
  - Type: Non-Static
  - Translation: x=0, y=0, z=0.185
  - Rotation: r=0 , p=0, yaw=0
  - Information: Joint 3 rotation about the x-axis
- Transform 6:
  - Father frame: link3
  - Child frame: link4
  - Type: Non-static
  - Translation: x=0, y=0.2, z=0
  - Rotation: r=0 , p=0, yaw=0
  - Information: Joint 4, Rotation about the y-axis





Code Part 1: Transforms

Code

- Open the file “manipulator.py”
- Copy the following template and save.

Be careful with the tabs!

```
#!/usr/bin/env python
import rospy
import numpy as np
from geometry_msgs.msg import TransformStamped
from visualization_msgs.msg import Marker
from tf2_ros import TransformBroadcaster, StaticTransformBroadcaster

# Setup parameters, transforms, variables and callback functions here (if
required)

# Declare Marker messages

#Declare Transform Messages

#################### Functions to Initialise Markers#####################
#Functions to initialise markers

########################################################################

################## ## Initialise Transform Messages#####################

#Functions to initialise transforms

########################################################################
```

```
if __name__=='__main__':
    #Initialise and Setup node
    rospy.init_node("manipulator_ex")
     # Configure the Node
    loop_rate = rospy.Rate(100)

    #Functions to initialise markers and transforms

    #Setup Publishers,subscribers and transform broadcasters

    try:
        #Run the node
        while not rospy.is_shutdown():

            #Declare time variable
            t = rospy.Time.now().to_sec()

            #Update the markers time stamp

            # Define the transform movements

            #Update the transforms time stamps and movements

            #Broadcast transforms

            #Publish markers

            loop_rate.sleep()

    except rospy.ROSInterruptException:
        pass
```

Code Part 1: Transforms

Running the node

- Open a terminal and re-build your package

```
$ cd ~/catkin_ws
$ catkin_make
$ source devel/setup.bash
```

- Run ROS

```
$ roscore
```

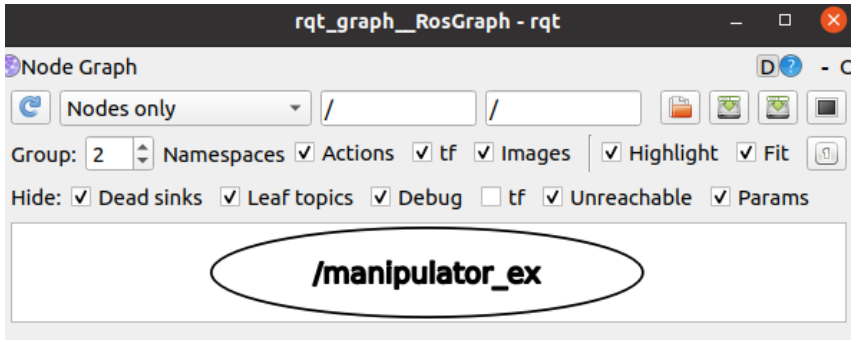
- Open a new terminal and run the node you just made using the following command

```
$ rosrun simple_manipulator manipulator.py
```

- Open a new terminal and run the rqt_graph

```
$ rosrun rqt_graph rqt_graph
```

Results



- You won't be able to see anything else since there is no program; this is just a checkpoint to make sure everything is running smoothly.

Code Part 1: Transforms

Code (Steps)

1. In the file “manipulator.py” declare the first transform message “baseLink_l0_tf” to be used in the “**Declarations**” section (yellow section).

```
#Declare Transform Messages
baseLink_l0_tf = TransformStamped()
```

2. Define the function “init_baseLink_l0_tf()” to initialise the transform message in the “**Declarations**” section.

```
#################### Initialise Transform Messages ######################
def init_baseLink_l0_tf():
    baseLink_l0_tf.header.frame_id = "base_link"
    baseLink_l0_tf.child_frame_id = "link0"
    baseLink_l0_tf.header.stamp = rospy.Time.now()
    baseLink_l0_tf.transform.translation.x = 0
    baseLink_l0_tf.transform.translation.y = 0
    baseLink_l0_tf.transform.translation.z = 0.0
    baseLink_l0_tf.transform.rotation.x = 0
    baseLink_l0_tf.transform.rotation.y = 0
    baseLink_l0_tf.transform.rotation.z = 0
    baseLink_l0_tf.transform.rotation.w = 1
```

3. In the “**main setup**” section, call the previously defined function

```
#Functions to initialise markers and transforms
init_baseLink_l0_tf()
```

4. Define a transform broadcaster for the previously defined transform (“**Main setup**” section).

```
#Setup Publishers, subscribers and transform broadcasters here
    bc_baselink = StaticTransformBroadcaster()
```

5. In the “**Loop**” section, update the time stamp of the transform before broadcast it.

```
 #Update the transforms
 baseLink_l0_tf.header.stamp = rospy.Time.now()

 #Broadcast transforms
 bc_baselink.sendTransform(baseLink_l0_tf)
```

Code Part 1: Transforms

Running the node

6. Open a terminal and re-build your package

```
$ cd ~/catkin_ws
$ catkin_make
$ source devel/setup.bash
```

7. Run ROS

```
$ roscore
```

8. Open a new terminal and run the node you just made using the following command

```
$ rosrun simple_manipulator manipulator.py
```

9. Open a new terminal and run the rviz

```
$ rosrun rviz rviz
```

10. In the “Fixed Frame” option in RVIZ GUI (top left corner) change it to “base_link”.

11. Press de “Add” button in the bottom left corner of the RVIZ GUI, look for “TF” option on the menu and add it by selecting it and pressing “OK”

Results

Code Part 1: Transforms

Code

- Repeat steps 1-5 of the previous two slides, and add the following transformations, in their respective sections

- Declariation Section

```
#Declare Transform Messages
l0_l1_tf = TransformStamped()
l1_l2_tf = TransformStamped()
l2_l3_tf = TransformStamped()
l3_l4_tf = TransformStamped()
l4_l5_tf = TransformStamped()
```

```
#################### Initialise Transform Messages ######################
def init_baseLink_l0_tf():
. . . (the definition is here but not enough space in the slide)

def init_l0_l1_tf():
    l0_l1_tf.header.frame_id = "link0"
    l0_l1_tf.child_frame_id = "link1"
    l0_l1_tf.header.stamp = rospy.Time.now()
    l0_l1_tf.transform.translation.x = 0
    l0_l1_tf.transform.translation.y = 0
    l0_l1_tf.transform.translation.z = 0.123
    l0_l1_tf.transform.rotation.x = 0
    l0_l1_tf.transform.rotation.y = 0
    l0_l1_tf.transform.rotation.z = 0
    l0_l1_tf.transform.rotation.w = 1
```

```
def init_l1_l2_tf():
    l1_l2_tf.header.frame_id = "link1"
    l1_l2_tf.child_frame_id = "link2"
    l1_l2_tf.header.stamp = rospy.Time.now()
    l1_l2_tf.transform.translation.x = 0
    l1_l2_tf.transform.translation.y = 0
    l1_l2_tf.transform.translation.z = 0.0
    l1_l2_tf.transform.rotation.x = 0
    l1_l2_tf.transform.rotation.y = 0
    l1_l2_tf.transform.rotation.z = 0
    l1_l2_tf.transform.rotation.w = 1

def init_l2_l3_tf():
    l2_l3_tf.header.frame_id = "link2"
    l2_l3_tf.child_frame_id = "link3"
    l2_l3_tf.header.stamp = rospy.Time.now()
    l2_l3_tf.transform.translation.x = 0
    l2_l3_tf.transform.translation.y = 0
    l2_l3_tf.transform.translation.z = 0.185
    l2_l3_tf.transform.rotation.x = 0
    l2_l3_tf.transform.rotation.y = 0
    l2_l3_tf.transform.rotation.z = 0
    l2_l3_tf.transform.rotation.w = 1

def init_l3_l4_tf():
    l3_l4_tf.header.frame_id = "link3"
    l3_l4_tf.child_frame_id = "link4"
    l3_l4_tf.header.stamp = rospy.Time.now()
    l3_l4_tf.transform.translation.x = 0
    l3_l4_tf.transform.translation.y = 0.2
    l3_l4_tf.transform.translation.z = 0
    l3_l4_tf.transform.rotation.x = 0
    l3_l4_tf.transform.rotation.y = 0
    l3_l4_tf.transform.rotation.z = 0
    l3_l4_tf.transform.rotation.w = 1
```

Code Part 1: Transforms

Code

- Repeat steps 1-5 of the previous two slides, and add the following transformations, in their respective sections
- Main Setup Section

```
#Functions to initialise markers and transforms
    init_baseLink_l0_tf()
    init_l0_l1_tf()
    init_l1_l2_tf()
    init_l2_l3_tf()
    init_l3_l4_tf()
```

```
#Setup Publishers, subscribers and transform broadcasters here
    bc_baselink = StaticTransformBroadcaster()
    bc_l0 = TransformBroadcaster()
    bc_l1 = TransformBroadcaster()
    bc_l2 = TransformBroadcaster()
    bc_l3 = TransformBroadcaster()
    bc_l4 = TransformBroadcaster()
    bc_l5 = TransformBroadcaster()
```

Loop Section

- Add the rotations for the other transforms (the previous was static)
- Rotations are defined in quaternions so we must use the tf_conversion library to convert from Euler angles to quaternions

```
# Define the transform movements
q_l1 = tf_conversions.transformations.quaternion_from_euler(0, 0, 0.5*np.sin(0.5*t))
q_l2 = tf_conversions.transformations.quaternion_from_euler(0.5*np.sin(t), 0, 0)
q_l3 = tf_conversions.transformations.quaternion_from_euler(0.5*np.cos(t), 0, 0)
q_l4 = tf_conversions.transformations.quaternion_from_euler(0, 0.5*np.cos(t), 0)
```

Code Part 1: Transforms

Loop Section

- Update the transforms

```
#Update the transforms time stamps and movements
            baseLink_l0_tf.header.stamp = rospy.Time.now()

            l0_l1_tf.header.stamp = rospy.Time.now()
            l0_l1_tf.transform.rotation.x = q_l1[0]
            l0_l1_tf.transform.rotation.y = q_l1[1]
            l0_l1_tf.transform.rotation.z = q_l1[2]
            l0_l1_tf.transform.rotation.w = q_l1[3]

            l1_l2_tf.header.stamp = rospy.Time.now()
            l1_l2_tf.transform.rotation.x = q_l2[0]
            l1_l2_tf.transform.rotation.y = q_l2[1]
            l1_l2_tf.transform.rotation.z = q_l2[2]
            l1_l2_tf.transform.rotation.w = q_l2[3]

            l2_l3_tf.header.stamp = rospy.Time.now()
            l2_l3_tf.transform.rotation.x = q_l3[0]
            l2_l3_tf.transform.rotation.y = q_l3[1]
            l2_l3_tf.transform.rotation.z = q_l3[2]
            l2_l3_tf.transform.rotation.w = q_l3[3]

            l3_l4_tf.header.stamp = rospy.Time.now()
            l3_l4_tf.transform.rotation.x = q_l4[0]
            l3_l4_tf.transform.rotation.y = q_l4[1]
            l3_l4_tf.transform.rotation.z = q_l4[2]
            l3_l4_tf.transform.rotation.w = q_l4[3]
```

Loop Section

- Broadcast the transforms

```
#Broadcast transforms
            bc_baselink.sendTransform(baseLink_l0_tf)
            bc_l0.sendTransform(l0_l1_tf)
            bc_l1.sendTransform(l1_l2_tf)
            bc_l2.sendTransform(l2_l3_tf)
            bc_l3.sendTransform(l3_l4_tf)
```

Code Part 1: Transforms

Running the node

6. Open a terminal and re-build your package

```
$ cd ~/catkin_ws
$ catkin_make
$ source devel/setup.bash
```

7. Run ROS

```
$ roscore
```

8. Open a new terminal and run the node you just made using the following command

```
$ rosrun simple_manipulator manipulator.py
```

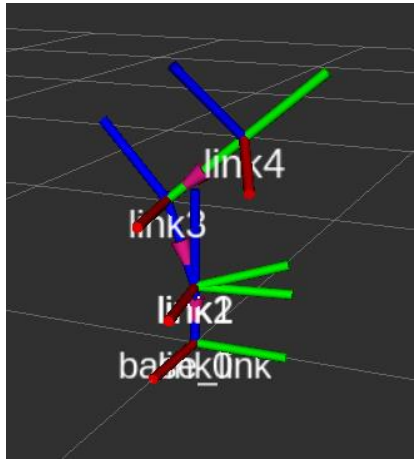
9. Open a new terminal and run the rviz

```
$ rosrun rviz rviz
```

10. In the “Fixed Frame” option in RVIZ GUI (top left corner) change it to “base_link”.

11. Press de “Add” button in the bottom left corner of the RVIZ GUI, look for “TF” option on the menu and add it by selecting it and pressing “OK”

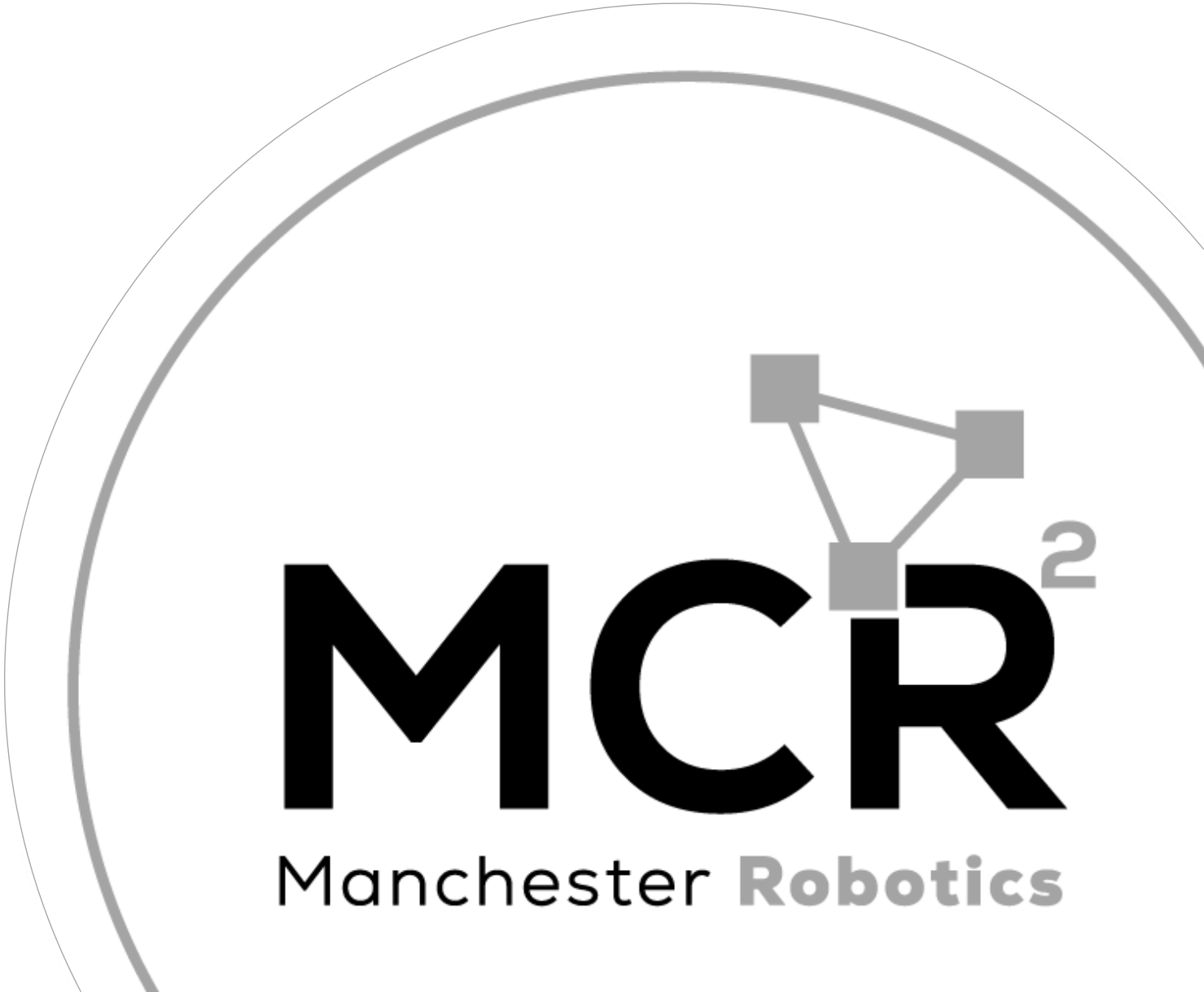
Results



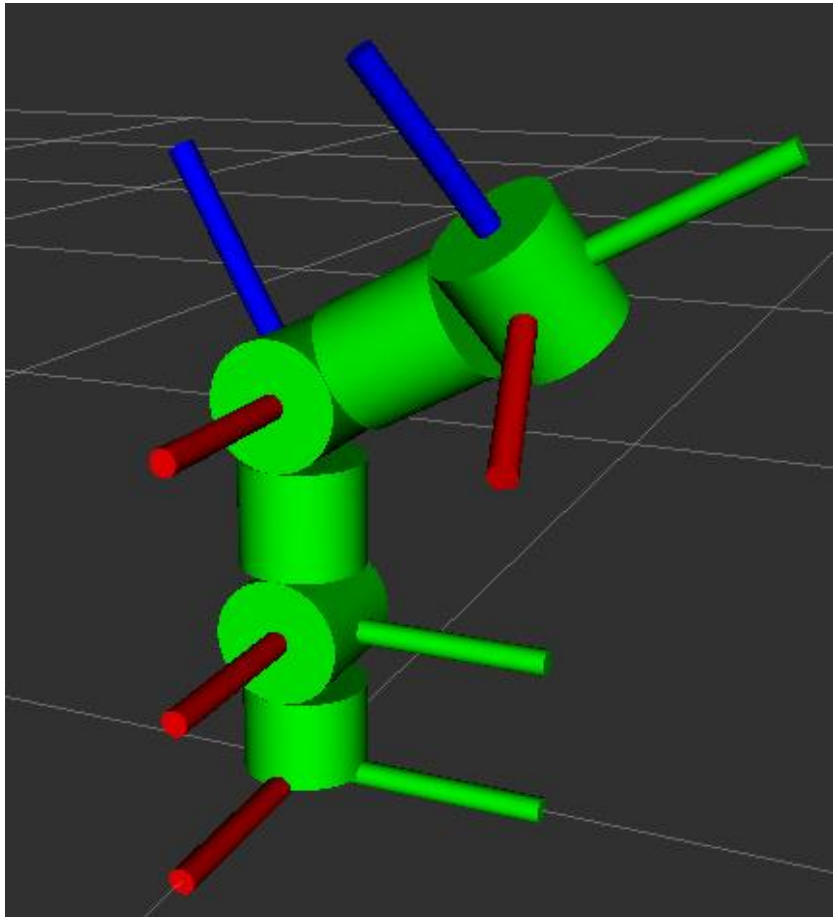
Simple Manipulator

Part 2: Markers

{Learn, Create, Innovate};

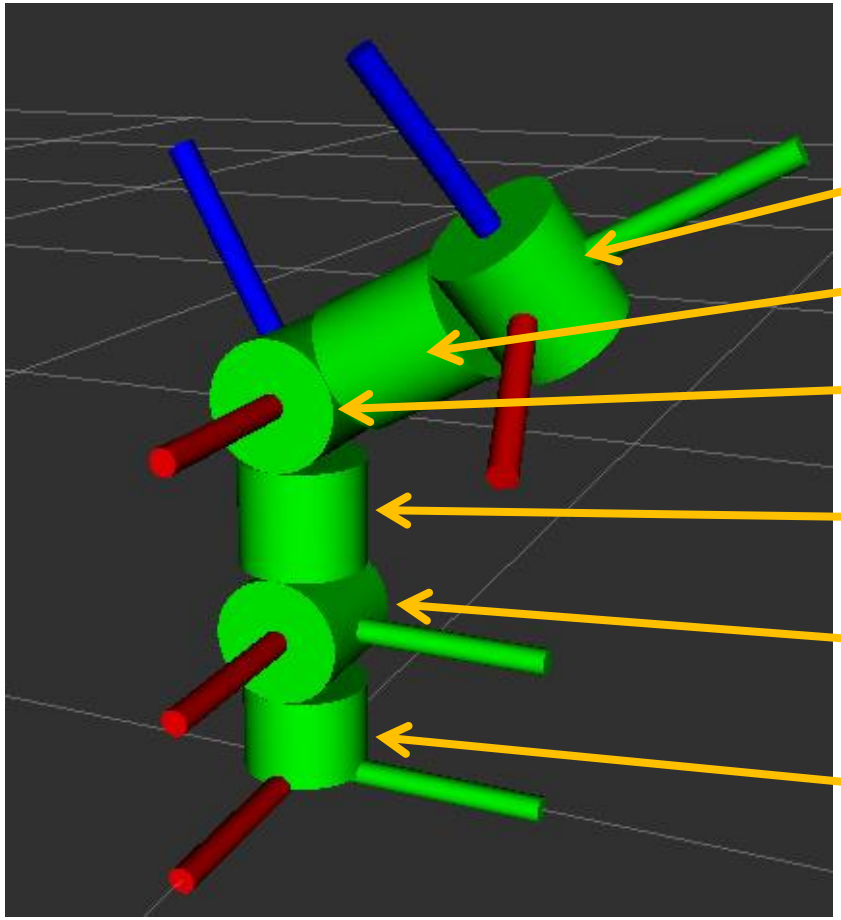


Robot Description



Description

- Six markers will be added to form the manipulator.
- The marker, in this case, represents the joints and the body of the manipulator.
- The markers to be used are cylinders, the user can select any other shape.
- Each marker will be attached to a frame in a different position.
- The movement of the frame will move the marker accordingly.



arm5_marker:

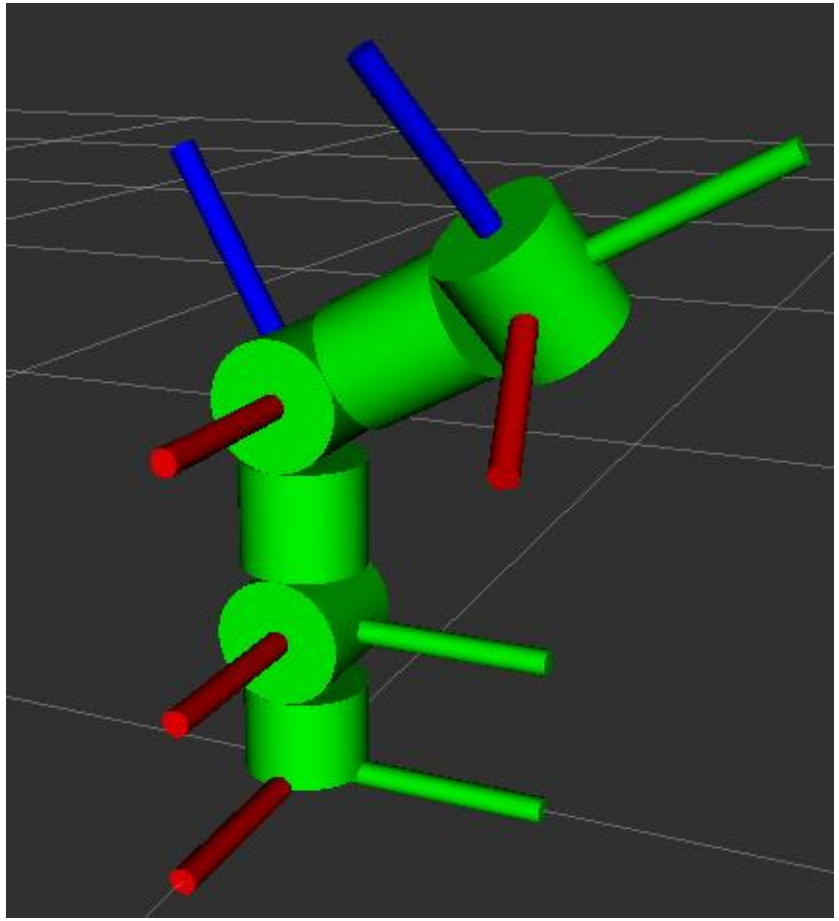
arm4_marker:

arm3_marker:

arm2_marker:

arm1_marker:

base_marker:



arm2_marker: frame: link2 Type: cylinder Height: 0.085 Diameter: 0.1 Position: x=0, y=0, z=0.0925 Rotation: r=0, p=0, y=0	arm5_marker: frame: link4 Type: cylinder Height: 0.1 Diameter: 0.1 Position: x=0, y=0, z=0 Rotation: r=0, p=0, y=0
arm1_marker: frame: link2 Type: cylinder Height: 0.1 Diameter: 0.1 Position: x=0, y=0, z=0.0365 Rotation: r=0, p=pi/2, y=0	arm4_marker: frame: link3 Type: cylinder Height: 0.1 Diameter: 0.1 Position: x=0, y=0.1, z=0 Rotation: r=pi/2, p=0, y=0
base_marker: frame: link0 Type: cylinder Height: 0.073 Diameter: 0.1 Position: x=0, y=0, z=0.0365 Rotation: r=0, p=0, y=0	arm3_marker: frame: link3 Type: cylinder Height: 0.1 Diameter: 0.1 Position: x=0, y=0, z=0 Rotation: r=0, p=pi/2, y=0

Code Part 2: Markers

Code (Steps)

1. In the file “manipulator.py” declare the first marker message “baseLink_l0_tf” to be used in the “Declarations” section (yellow section).

```
# Declare Marker messages
base_marker = Marker()
```

2. Define the function “def init_base_marker()” to initialise the marker message in the “Declarations” section.

```
########## Functions to Initialise Markers#################################

#Functions to initialise markers
def init_base_marker():
    # Declare the link0 Marker Message
    base_marker.header.frame_id = "link0"
    base_marker.header.stamp = rospy.Time.now()
    base_marker.id = 0
    base_marker.type = 3
    base_marker.action = 0
    base_marker.pose.position.x = 0.0
    base_marker.pose.position.y = 0.0
    base_marker.pose.position.z = (0.123-0.05)/2
    base_marker.pose.orientation.x = 0
    base_marker.pose.orientation.y = 0
    base_marker.pose.orientation.z = 0
    base_marker.pose.orientation.w = 1
    base_marker.scale.x = 0.1
    base_marker.scale.y = 0.1
    base_marker.scale.z = 0.123-0.05
    base_marker.color.r = 0.0
    base_marker.color.g = 1.0
    base_marker.color.b = 0.0
    base_marker.color.a = 1.0
    base_marker.lifetime = rospy.Duration(0)
```

Code Part 2: Markers

3. In the "**main setup**" section, call the previously defined function

```
#Functions to initialise markers and transforms
init_base_marker()
```

4. Define a publisher for the previously defined marker ("**Main setup**" section).

```
 #Setup Publishers, subscribers
 pub_link0 = rospy.Publisher('/link0', Marker, queue_size=1)
```

5. In the "**Loop**" section, update the time stamp of the marker before publishing it.

```
 #Update the transforms
 baseLink_l0_tf.header.stamp = rospy.Time.now()

#Publish markers
 pub_link0.publish(base_marker)
```

Running the node

6. Open a terminal and re-build your package

```
$ cd ~/catkin_ws
$ catkin_make
$ source devel/setup.bash
```

7. Run ROS

```
$ roscore
```

8. Open a new terminal and run the node you just made using the following command

```
$ rosrun simple_manipulator manipulator.py
```

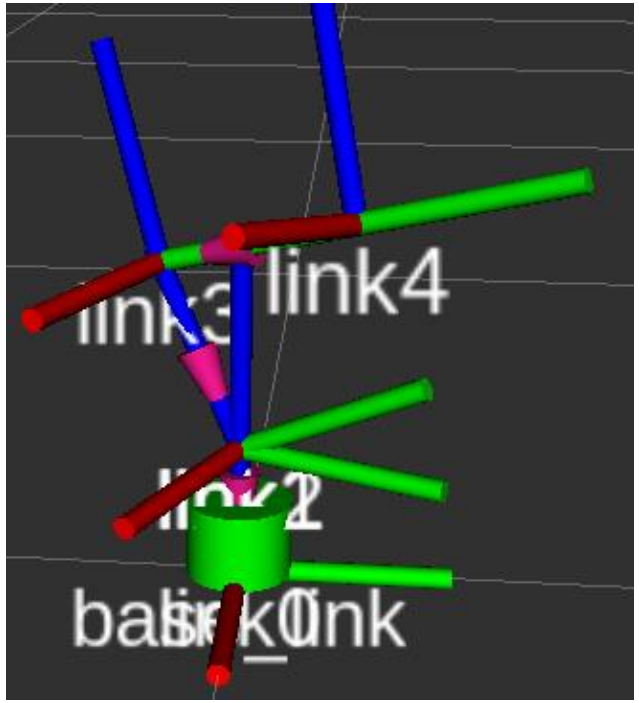
9. Open a new terminal and run the rviz

```
$ rosrun rviz rviz
```

Code Part 2: Markers

10. In the “Fixed Frame” option in RVIZ GUI (top left corner) change it to “base_link”.
11. Press de “Add” button in the bottom left corner of the RVIZ GUI.
12. Look for “TF” option on the menu and add it by selecting it and pressing “OK”
13. Repeat step 11 and select “By topic” at the top part of the pop-up window. Select “/link0/Marker” and add it.

Results



Code Part 2: Markers

Code

- Repeat steps 1-5 of the previous two slides, and add the following markers, in their respective sections
- Declaration Section

```
# Declare Marker messages
base_marker = Marker()
arm1_marker = Marker()
arm2_marker = Marker()
arm3_marker = Marker()
arm4_marker = Marker()
arm5_marker = Marker()
```

```
#Functions to initialise markers
def init_base_marker():
. . . (the definition is here but not enough space in the slide)

def init_arm1_marker():
    # Declare the link1 Marker Message
    q_m1 = tf_conversions.transformations.quaternion_from_euler(0, np.pi/2, 0)
    arm1_marker.header.frame_id = "link2"
    arm1_marker.header.stamp = rospy.Time.now()
    arm1_marker.id = 0
    arm1_marker.type = 3
    arm1_marker.action = 0
    arm1_marker.pose.position.x = 0.0
    arm1_marker.pose.position.y = 0.0
    arm1_marker.pose.position.z = 0.0
    arm1_marker.pose.orientation.x = q_m1[0]
    arm1_marker.pose.orientation.y = q_m1[1]
    arm1_marker.pose.orientation.z = q_m1[2]
    arm1_marker.pose.orientation.w = q_m1[3]
    arm1_marker.scale.x = 0.1
    arm1_marker.scale.y = 0.1
    arm1_marker.scale.z = 0.1
    arm1_marker.color.r = 0.0
    arm1_marker.color.g = 1.0
    arm1_marker.color.b = 0.0
    arm1_marker.color.a = 1.0
    arm1_marker.lifetime = rospy.Duration(0)
```

Code Part 2: Markers

Code

- Declaration Section

```
def init_arm2_marker():
    # Declare the link2 Marker Message
    arm2_marker.header.frame_id = "link2"
    arm2_marker.header.stamp = rospy.Time.now()
    arm2_marker.id = 0
    arm2_marker.type = 3
    arm2_marker.action = 0
    arm2_marker.pose.position.x = 0.0
    arm2_marker.pose.position.y = 0.0
    arm2_marker.pose.position.z = (0.185-0.1)/2+0.05
    arm2_marker.pose.orientation.x = 0
    arm2_marker.pose.orientation.y = 0
    arm2_marker.pose.orientation.z = 0
    arm2_marker.pose.orientation.w = 1
    arm2_marker.scale.x = 0.1
    arm2_marker.scale.y = 0.1
    arm2_marker.scale.z = 0.185-0.1
    arm2_marker.color.r = 0.0
    arm2_marker.color.g = 1.0
    arm2_marker.color.b = 0.0
    arm2_marker.color.a = 1.0
    arm2_marker.lifetime = rospy.Duration(0)
```

```
def init_arm3_marker():
    # Declare the link3 Marker Message
    q_m3 = tf_conversions.transformations.quaternion_from_euler(0, np.pi/2, 0)
    arm3_marker.header.frame_id = "link3"
    arm3_marker.header.stamp = rospy.Time.now()
    arm3_marker.id = 0
    arm3_marker.type = 3
    arm3_marker.action = 0
    arm3_marker.pose.position.x = 0.0
    arm3_marker.pose.position.y = 0.0
    arm3_marker.pose.position.z = 0.0
    arm3_marker.pose.orientation.x = q_m3[0]
    arm3_marker.pose.orientation.y = q_m3[1]
    arm3_marker.pose.orientation.z = q_m3[2]
    arm3_marker.pose.orientation.w = q_m3[3]
    arm3_marker.scale.x = 0.1
    arm3_marker.scale.y = 0.1
    arm3_marker.scale.z = 0.1
    arm3_marker.color.r = 0.0
    arm3_marker.color.g = 1.0
    arm3_marker.color.b = 0.0
    arm3_marker.color.a = 1.0
    arm3_marker.lifetime = rospy.Duration(0)
```

Code Part 2: Markers

Code

- Declaration Section

```
def init_arm4_marker():
    # Declare the link4 Marker Message
    q_m4 = tf_conversions.transformations.quaternion_from_euler(np.pi/2, 0, 0)
    arm4_marker.header.frame_id = "link3"
    arm4_marker.header.stamp = rospy.Time.now()
    arm4_marker.id = 0
    arm4_marker.type = 3
    arm4_marker.action = 0
    arm4_marker.pose.position.x = 0.0
    arm4_marker.pose.position.y = (0.2-0.1)/2+0.05
    arm4_marker.pose.position.z = 0.0
    arm4_marker.pose.orientation.x = q_m4[0]
    arm4_marker.pose.orientation.y = q_m4[1]
    arm4_marker.pose.orientation.z = q_m4[2]
    arm4_marker.pose.orientation.w = q_m4[3]
    arm4_marker.scale.x = 0.1
    arm4_marker.scale.y = 0.1
    arm4_marker.scale.z = 0.2-0.1
    arm4_marker.color.r = 0.0
    arm4_marker.color.g = 1.0
    arm4_marker.color.b = 0.0
    arm4_marker.color.a = 1.0
    arm4_marker.lifetime = rospy.Duration(0)
```

```
def init_arm5_marker():
    arm5_marker.header.frame_id = "link4"
    arm5_marker.header.stamp = rospy.Time.now()
    arm5_marker.id = 0
    arm5_marker.type = 3
    arm5_marker.action = 0
    arm5_marker.pose.position.x = 0.0
    arm5_marker.pose.position.y = 0.0
    arm5_marker.pose.position.z = 0.0
    arm5_marker.pose.orientation.x = 0
    arm5_marker.pose.orientation.y = 0
    arm5_marker.pose.orientation.z = 0
    arm5_marker.pose.orientation.w = 1
    arm5_marker.scale.x = 0.1
    arm5_marker.scale.y = 0.1
    arm5_marker.scale.z = 0.1
    arm5_marker.color.r = 0.0
    arm5_marker.color.g = 1.0
    arm5_marker.color.b = 0.0
    arm5_marker.color.a = 1.0
    arm5_marker.lifetime = rospy.Duration(0)
```

Code Part 2: Markers

- In the "**main setup**" section

```
#Functions to initialise markers and transforms
init_base_marker()
init_arm1_marker()
init_arm2_marker()
init_arm3_marker()
init_arm4_marker()
init_arm5_marker()
```

```
 #Setup Publishers, subscribers
pub_link0 = rospy.Publisher('/link0', Marker, queue_size=1)
pub_link1 = rospy.Publisher('/link1', Marker, queue_size=1)
pub_link2 = rospy.Publisher('/link2', Marker, queue_size=1)
pub_link3 = rospy.Publisher('/link3', Marker, queue_size=1)
pub_link4 = rospy.Publisher('/link4', Marker, queue_size=1)
pub_link5 = rospy.Publisher('/link5', Marker, queue_size=1)
```

- In the **Loop** section

```
 #Update the markers
base_marker.header.stamp = rospy.Time.now()
arm1_marker.header.stamp = rospy.Time.now()
arm2_marker.header.stamp = rospy.Time.now()
arm3_marker.header.stamp = rospy.Time.now()
arm4_marker.header.stamp = rospy.Time.now()
arm5_marker.header.stamp = rospy.Time.now()

#Publish markers
pub_link0.publish(base_marker)
pub_link1.publish(arm1_marker)
pub_link2.publish(arm2_marker)
pub_link3.publish(arm3_marker)
pub_link4.publish(arm4_marker)
pub_link5.publish(arm5_marker)
```

Code Part 2: Transforms

Running the node

- Open a terminal and re-build your package

```
$ cd ~/catkin_ws
$ catkin_make
$ source devel/setup.bash
```

- Run ROS

```
$ roscore
```

- Open a new terminal and run the node you just made using the following command

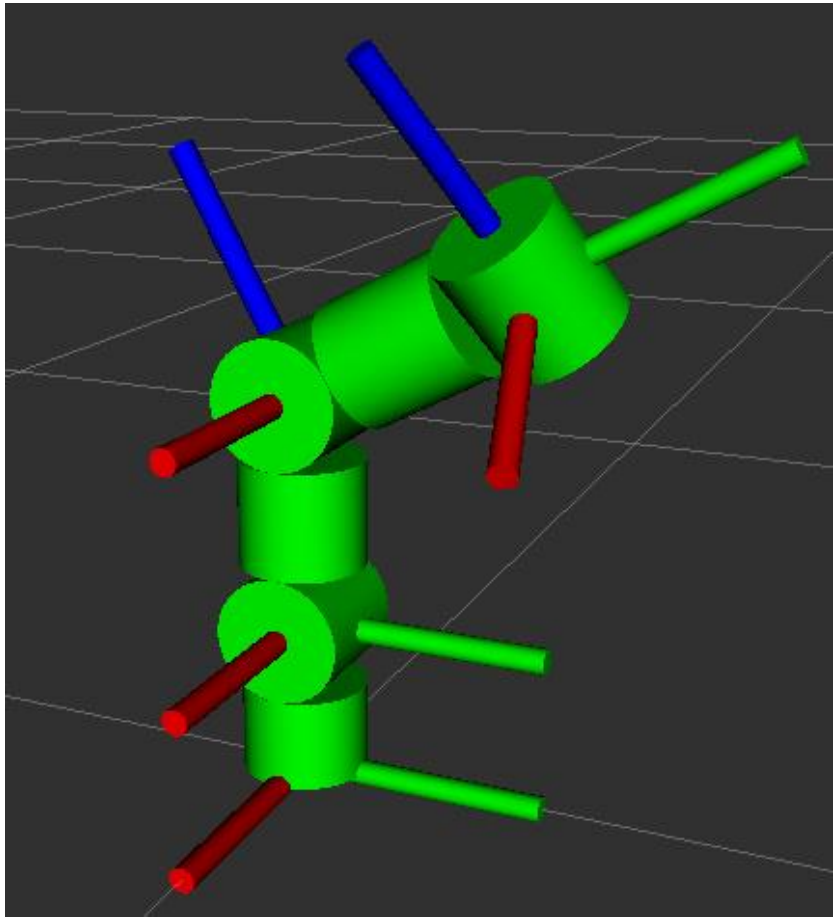
```
$ rosrun simple_manipulator manipulator.py
```

- Open a new terminal and run the rviz

```
$ rosrun rviz rviz
```

- In the “Fixed Frame” option in RVIZ GUI (top left corner) change it to “base_link”.
- Press de “Add” button in the bottom left corner of the RVIZ GUI, look for “TF” option on the menu and add it by selecting it and pressing “OK”
- Repeat the previous step and select “By topic” at the top part of the pop-up window. Select and add all the markers.
- You can save RVIZ configuration by going to File>>Save Config As.
- Set a name “robot_rviz.rviz”
- It is recommended for you to save on a folder called ”rviz” inside your package “simple_manipulator”

Code Part 2: Transforms



Launch File

Define a launch file

- Open a terminal

```
$ cd ~/catkin_ws/src/simple_manipulator
$ mkdir launch
$ cd launch
$ touch robot.launch
```

- Open the file and write the following
- In this file a static transform from the frame “world” to “base_link” is being created (change the “Fixed Frame” in Rviz).

```
<?xml version="1.0" ?>
<launch>
    <node name="robot" pkg="simple_manipulator" type="manipulator.py" output="screen"/>
    <arg name="rvizconfig" default="$(find simple_manipulator)/rviz/robot_rviz.rviz" />
    <node name="rviz" pkg="rviz" type="rviz" args="-d $(arg rvizconfig)" required="true" />
    <node pkg="tf2_ros" type="static_transform_publisher" name="world_tf" args="0 0 0 0 0 0   world   base_link"/>
</launch>
```

Results