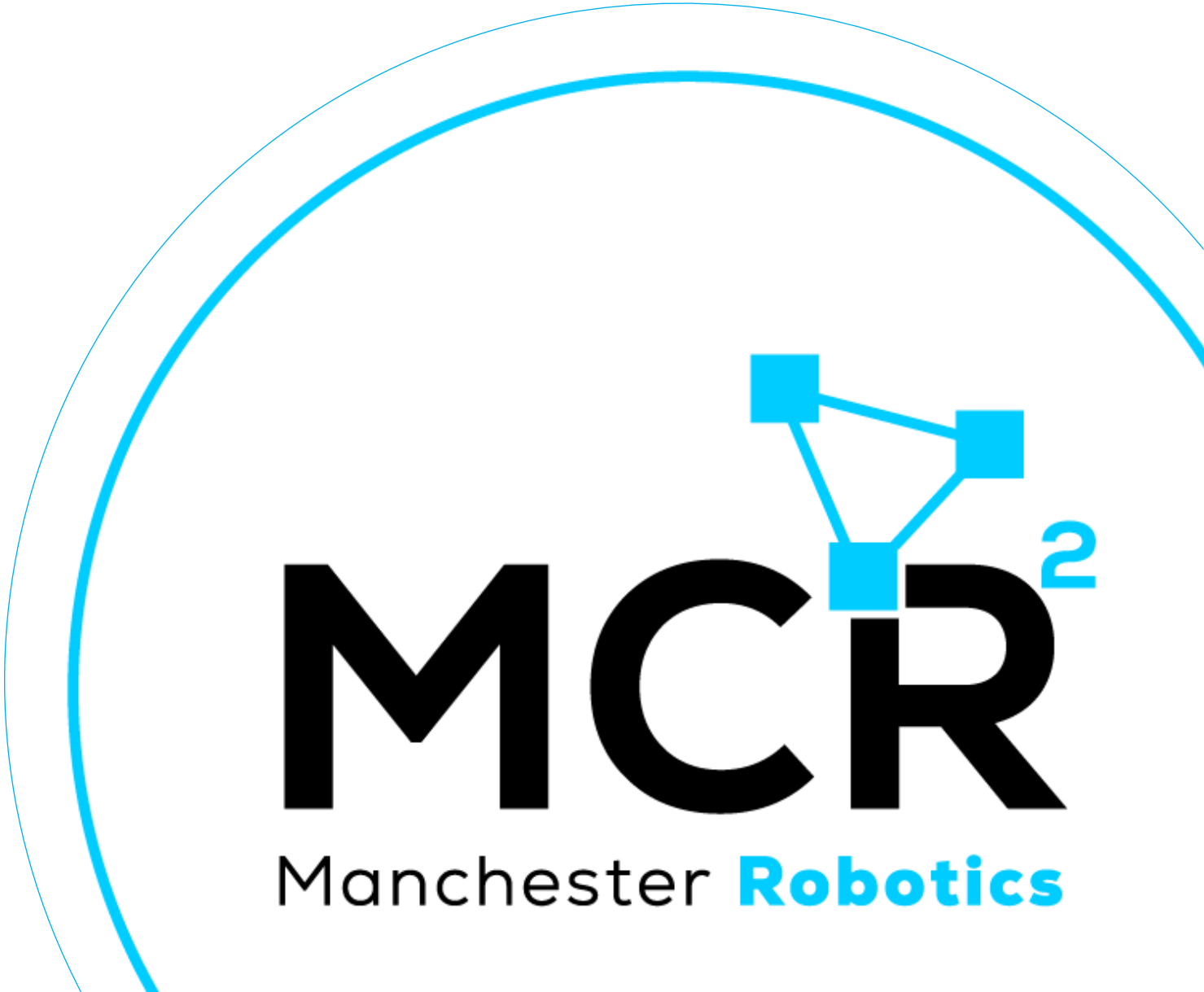
{Learn, Create, Innovate};

ROS

Visualisation Tools

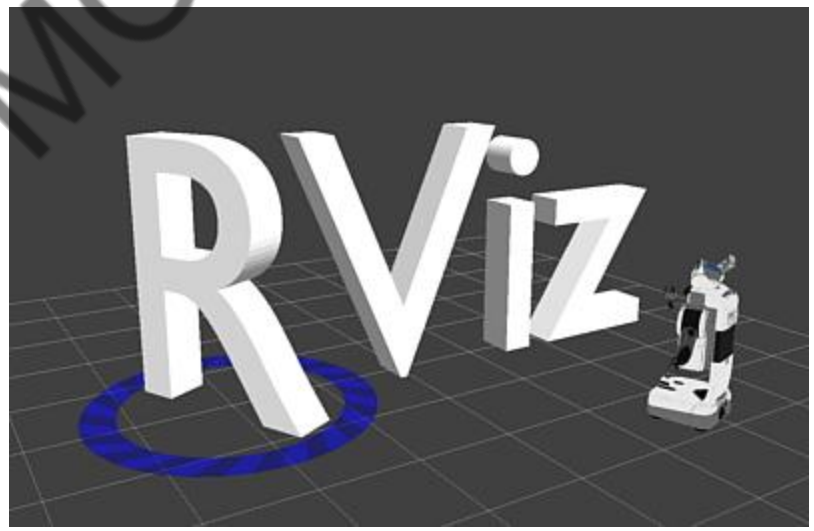


RVIZ

What is RVIZ?

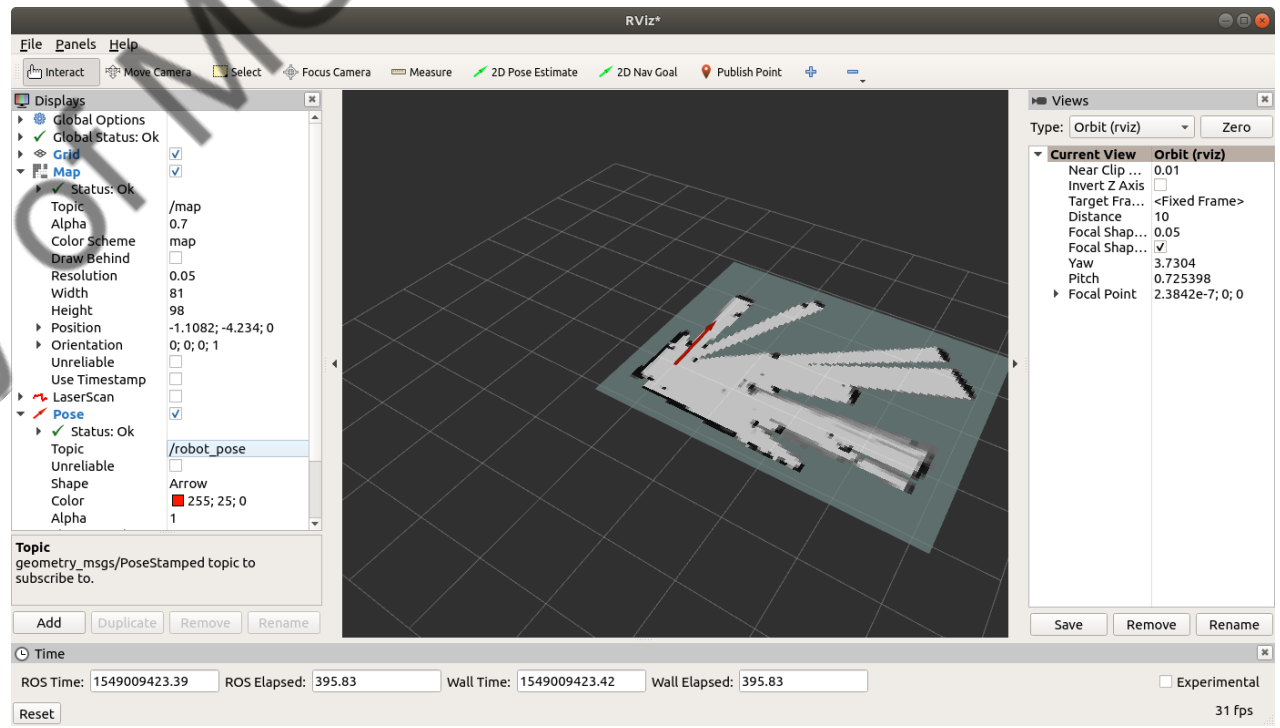
- RVIZ (ROS Visualization)
- Is a 3D visualisation environment
- Made to simplify debugging using visual tools.
- RVIZ allows the user to see what the robot is seeing, thinking and doing.

  "See the world through the robot's eyes."



RVIZ

- There are two main ways of putting data into RVIZ.
  - Via messages, where it understands sensors and state information, like laser scans, point clouds, cameras, and coordinate frames.
    - They have specialised displays to let the user configure how to view that information.
  - Information markers, letting the user send cubes, arrows and lines coloured however you want.
- The combination of sensor data and custom visualisation markers makes RVIZ a powerful tool for robotic development.



RVIZ

Quick Start (USB camera)

- Download the rospackage usb_cam

```
sudo apt install ros-<$DISTRO>-usb-cam
```

- Run the node

```
ros2 run usb_cam usb_cam_node_exe
```

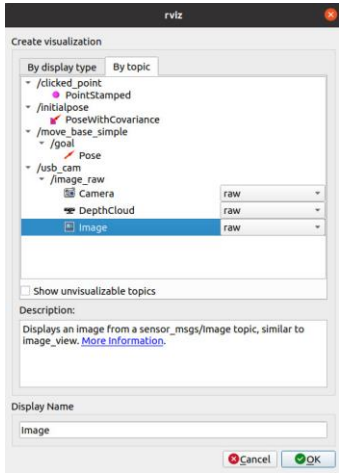
- Check that the topics are being published

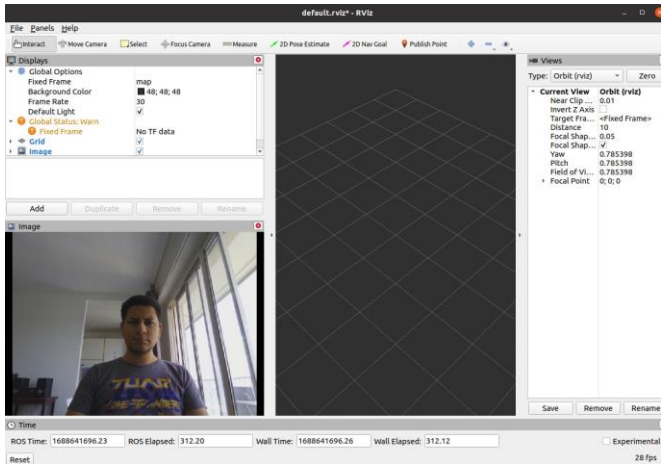
```
ros2 topic list
```

- Start RVIZ

```
ros2 run rviz2 rviz2
```

- Press the “add” button
- Go to the tab ”By topic”
- Add the topic Camera, located under the topics /usb_cam -> /image_raw

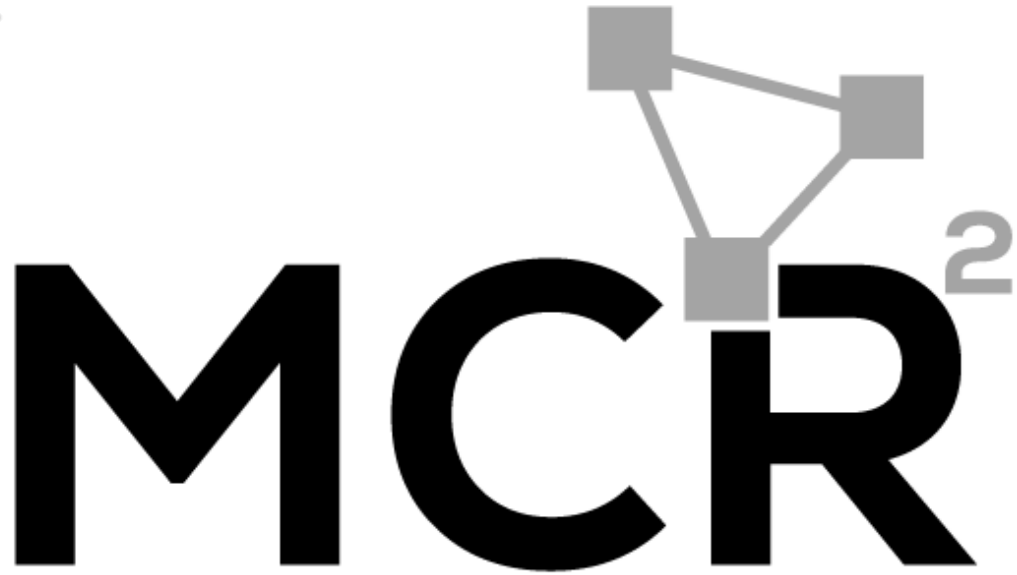




RVIZ

Markers

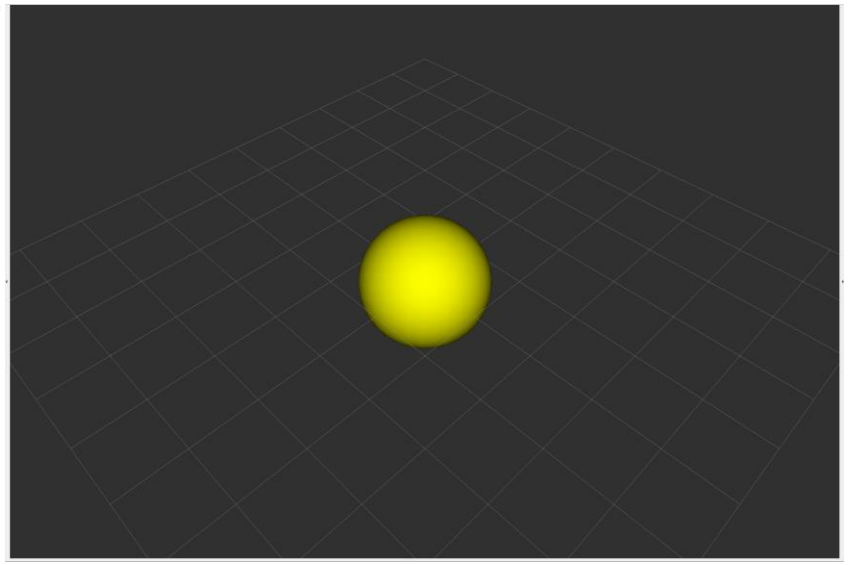
{Learn, Create, Innovate};



Markers

What are markers?

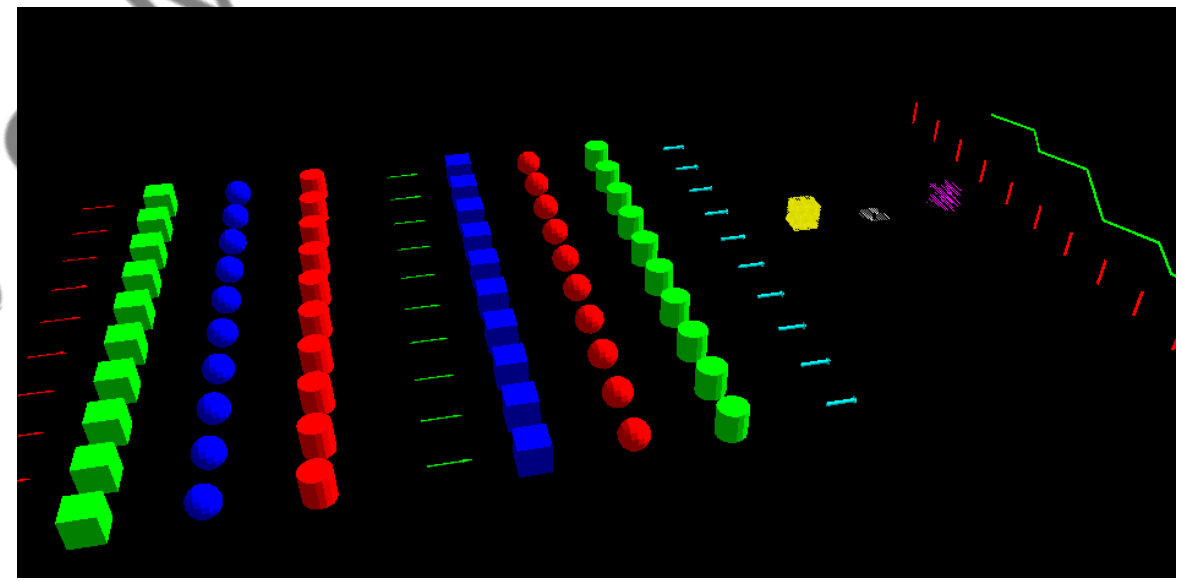
- One of the key features of RViz is the ability to visualize markers.
- Markers are graphical objects that represent different types of data in the 3D space.
- They can display points, lines, meshes, text, and more.
- Markers are typically published as ROS messages and can be subscribed to by RViz for visualisation.
- RViz provides a user-friendly interface for adding, configuring, and visualizing markers.



Markers

Type of markers?

- There are different types of markers that help us to visualise information in RViz.
- The basic markers are points, cubes, spheres, arrows, and lines.
- The user can set up and define its own markers and make them move.



Markers

- Markers are defined as ROS messages (composed of submessages) in which the user inputs the type of marker, configuration and characteristics. More information here and here

```
uint8 ARROW=0, uint8 CUBE=1, uint8 SPHERE=2, uint8 CYLINDER=3, uint8 LINE_STRIP=4
uint8 LINE_LIST=5, uint8 CUBE_LIST=6, uint8 SPHERE_LIST=7, uint8 POINTS=8, uint8 TEXT_VIEW_FACING=9, uint8 MESH_RESOURCE=10,
uint8 TRIANGLE_LIST=11

uint8 ADD=0, uint8 MODIFY=0, uint8 DELETE=2,
uint8 DELETEALL=3

Header header                       # header for time/frame information
string ns                           # Namespace to place this object in... used in conjunction with id to create a unique name for the object
int32 id                          # object ID useful in conjunction with the namespace for manipulating and deleting the object later
int32 type                        # Type of object
int32 action                        # 0 add/modify an object, 1 (deprecated), 2 deletes an object, 3 deletes all objects
geometry_msgs/Pose pose                 # Pose of the object
geometry_msgs/Vector3 scale             # Scale of the object 1,1,1 means default (usually 1 meter square)
std_msgs/ColorRGBA color            # Color [0.0-1.0]
duration lifetime                   # How long the object should last before being automatically deleted.  0 means forever
bool frame_locked                   # If this marker should be frame-locked, i.e. retransformed into its frame every timestep

#Only used if the type specified has some use for them (eg. POINTS, LINE_STRIP, ...)
geometry_msgs/Point[] points
#Only used if the type specified has some use for them (eg. POINTS, LINE_STRIP, ...)
#number of colors must either be 0 or equal to the number of points
#NOTE: alpha is not yet used
std_msgs/ColorRGBA[] colors

# NOTE: only used for text markers
string text

# NOTE: only used for MESH_RESOURCE markers
string mesh_resource
bool mesh_use_embedded_materials
```

Markers

Key Aspects

- When using Markers three key aspects must be taken into consideration.

Header

- Header Message: in this section the user define the marker frame and the time stamp.
- Markers must be attached to a frame of reference!

```
Header header # header for time/frame information
```

```
chassis = Marker()

chassis.header.frame_id = "base_link"
chassis.header.stamp = self.get_clock().now().to_msg()
```

Pose

- Pose Message, in this section the user define the pose (position and orientation) of the marker with respect to the frame of reference stated on the Header ("base_link"). The position is given by a point and the orientation by a quaternion.

```
geometry_msgs/Pose pose  # Pose of the object
```

```
chassis.pose.position.x = 0.205/2
chassis.pose.position.y = 0.0
chassis.pose.position.z = 0.0
chassis.pose.orientation.x = 0.0
chassis.pose.orientation.y = 0.0
chassis.pose.orientation.z = 0.0
chassis.pose.orientation.w = 1.0
```

Markers

Key Aspects

- When using Markers three key aspects must be taken into consideration.

Header

- Header Message: in this section the user define the marker frame and the time stamp.
- Markers must be attached to a frame of reference!

```
Header header # header for time/frame information
```

```
chassis = Marker()

chassis.header.frame_id = "base_link"
chassis.header.stamp = self.get_clock().now().to_msg()
```

Pose

- Pose Message, in this section the user define the pose (position and orientation) of the marker with respect to the frame of reference stated on the Header ("base_link"). The position is given by a point and the orientation by a quaternion.
- If the pose varies through time, it will be done w.r.t the header's frame of reference.

```
geometry_msgs/Pose pose  # Pose of the object
```

```
chassis.pose.position.x = 0.205/2
chassis.pose.position.y = 0.0
chassis.pose.position.z = 0.0
chassis.pose.orientation.x = 0.0
chassis.pose.orientation.y = 0.0
chassis.pose.orientation.z = 0.0
chassis.pose.orientation.w = 1.0
```

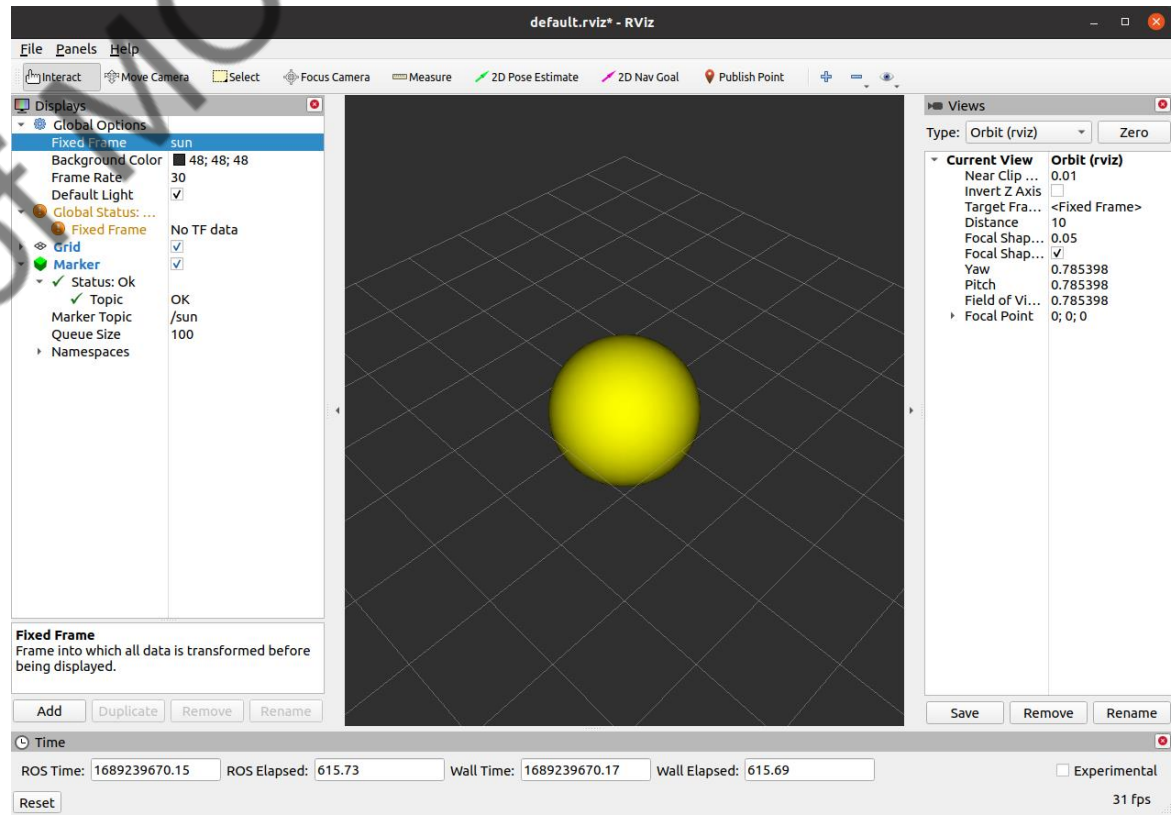
Markers

Scale

- Vector3 message in this section the user defines the Scale/Size of the object in each of its dimension's x,y,z.

```
geometry_msgs/Vector3 scale # Scale of the object
 1,1,1 means default (usually 1 meter square)
```

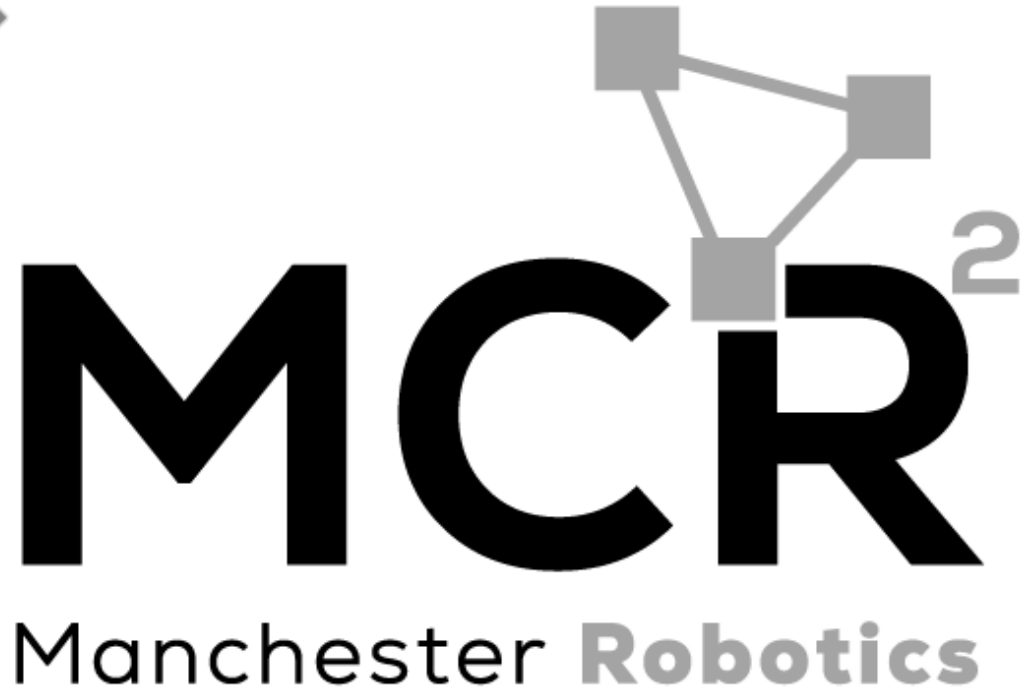
```
chassis.scale.x = 0.205
chassis.scale.y = 0.41
chassis.scale.z = 0.2
```



Activity 1

Markers

{Learn, Create, Innovate};



Marker Generator

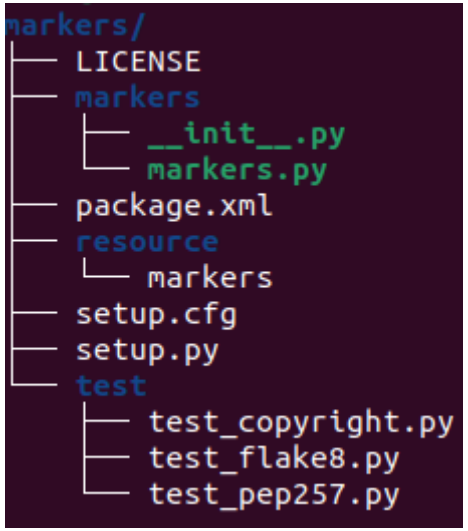
Declaring a Static Transform from Launch File

- Make a package called “markers” and a node called “markers.py” with the following dependencies:
  - geometry_msgs, python3-numpy, rclpy, tf2_ros_py, ros2launch, std_msgs python3-transforms3d visualization_msgs

```
$ ros2 pkg create --build-type ament_python markers --license
Apache-2.0 --node-name markers --dependencies geometry_msgs
python3-numpy rclpy tf2_ros_py ros2launch std_msgs python3-
transforms3d visualization_msgs --description TF2_Examples --
maintainer-name "Mario Martinez" --maintainer-email
mario.mtz@manchester-robotics.com
```

- Do not forget to give executable permissions to the newly created files

```
$ chmod +x markers/markers/*
```



Marker Generator

- Make a launch folder and a launch file called "simple_marker_launch.py"

```
#replace "YOUR_WS" with the name of tour workspace

$ cd ~/<YOUR_WS>/src/markers
$ mkdir launch
$ touch simple_marker_launch.py
$ chmod +x markers/launch/*
```

- Change the "setup.py" to find the launch files

```
from setuptools import find_packages, setup
import os
from glob import glob

package_name = 'tf_examples'

setup(
    name=package_name,
    version='0.0.0',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
        (os.path.join('share', package_name, 'launch'), glob(os.path.join('launch', '*launch.[pxy][yma]*'))),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='Mario Martinez',
    maintainer_email='mario.mtz@manchester-robotics.com',
    description='TF2_Examples',
    license='Apache-2.0',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'static_tf = tf_examples.static_tf:main'
        ],
    },
)
```


```
from setuptools import find_packages, setup
import os
from glob import glob
...

 data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
        (os.path.join('share', package_name, 'launch'), glob(os.path.join('launch', '*launch.[pxy][yma]*'))),
    ],
```

Marker Generator

- Open the file *markers.py*
- Define a new marker called marker in the constructor.

```
# Message Declaration
self.marker = Marker()
```

- Define its publisher

```
self.publisher = self.create_publisher(Marker, '/marker', 10)
```

- Initialise the marker (if needed) on the constructor
  - For this exercise the marker is defined w.r.t "map" frame.
- Define a timer to update the marker pose (next slide)
- Publish the marker

```
        #Variables to be used
        self.omega = 0.5
        self.intial_pos_x = 0.0
        self.intial_pos_y = 0.0
        self.intial_pos_z = 1.0

        #initialise the marker(the pose and orientation will be
changed on the callback function)
        self.marker = Marker()
        self.marker.header.frame_id = "map"
        self.marker.header.stamp = self.get_clock().now().to_msg()
        self.marker.id = 0
        self.marker.type = Marker.CUBE
        self.marker.action = Marker.ADD
        self.marker.pose.position.x = self.intial_pos_x
        self.marker.pose.position.y = self.intial_pos_y
        self.marker.pose.position.z = self.intial_pos_z
        self.marker.pose.orientation.x = 0.0
        self.marker.pose.orientation.y = 0.0
        self.marker.pose.orientation.z = 0.0
        self.marker.pose.orientation.w = 1.0
        self.marker.scale.x = 0.2
        self.marker.scale.y = 0.5
        self.marker.scale.z = 0.2
        self.marker.color.r = 1.0
        self.marker.color.g = 1.0
        self.marker.color.b = 0.0
        self.marker.color.a = 1.0
```

```python
import rclpy
from rclpy.node import Node
from visualization_msgs.msg import Marker
import transforms3d
import numpy as np
from rclpy.duration import Duration

class MarkersPublisher(Node):
    def __init__(self):
        super().__init__('marker_publisher')
        self.publisher = self.create_publisher(Marker, '/marker', 10)

        #Variables to be used
        self.omega = 0.5
        self.intial_pos_x = 0.0
        self.intial_pos_y = 0.0
        self.intial_pos_z = 1.0

       #initialise the marker
        self.marker = Marker()
        self.marker.header.frame_id = "map"
        self.marker.header.stamp = self.get_clock().now().to_msg()
        self.marker.id = 0
        self.marker.type = Marker.CUBE
        self.marker.action = Marker.ADD
        self.marker.pose.position.x = self.intial_pos_x
        self.marker.pose.position.y = self.intial_pos_y
        self.marker.pose.position.z = self.intial_pos_z
        self.marker.pose.orientation.x = 0.0
        self.marker.pose.orientation.y = 0.0
        self.marker.pose.orientation.z = 0.0
        self.marker.pose.orientation.w = 1.0
        self.marker.scale.x = 0.2
        self.marker.scale.y = 0.5
        self.marker.scale.z = 0.2
        self.marker.color.r = 1.0
        self.marker.color.g = 1.0
        self.marker.color.b = 0.0
        self.marker.color.a = 1.0
        timer_period = 0.1 #seconds
        self.timer = self.create_timer(timer_period, self.timer_cb)
        self.i = 0

    #Timer Callback
    def timer_cb(self):
        time = self.get_clock().now().nanoseconds/1e9

        q = transforms3d.euler.euler2quat(0, 1.57, self.omega*time)
        self.marker.header.stamp = self.get_clock().now().to_msg()
        self.marker.pose.position.x = self.intial_pos_x + 0.5*np.sin(self.omega*time)
        self.marker.pose.position.y = self.intial_pos_y + 0.5*np.cos(self.omega*time)
        self.marker.pose.position.z = self.intial_pos_z
        self.marker.pose.orientation.x = q[1]
        self.marker.pose.orientation.y = q[2]
        self.marker.pose.orientation.z = q[3]
        self.marker.pose.orientation.w = q[0]

        self.publisher.publish(self.marker)

def main(args=None):
    rclpy.init(args=args)
    node = MarkersPublisher()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    finally:
        if rclpy.ok():  # Ensure shutdown is only called once
            rclpy.shutdown()
        node.destroy_node()

if __name__ == '__main__':
    main()
```

Marker Generator

- Compile the program

```
cd ~/<YOUR_WS>

colcon build

source install/setup.bash
```

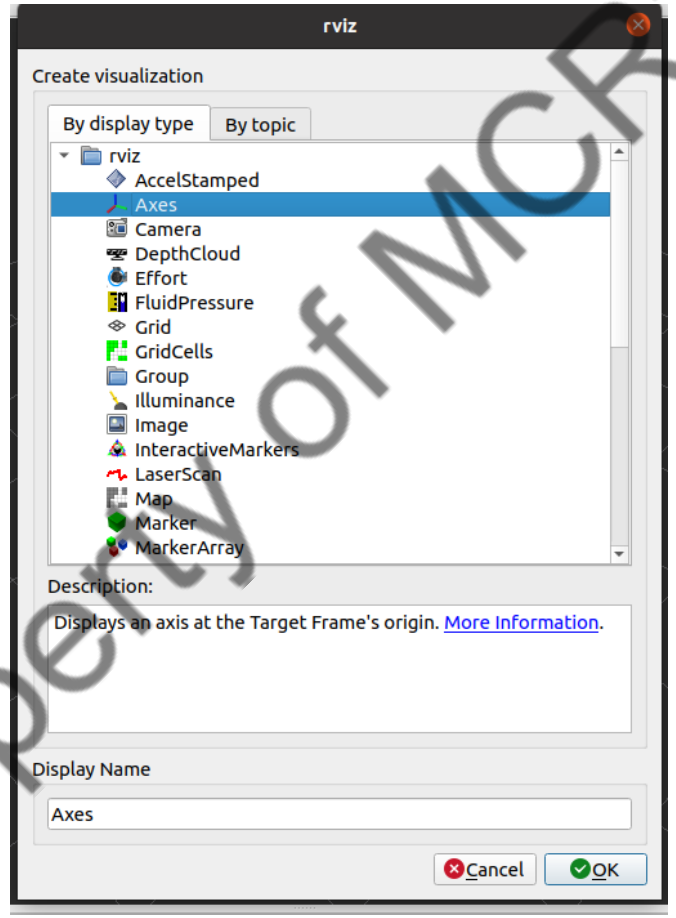
- Run the node

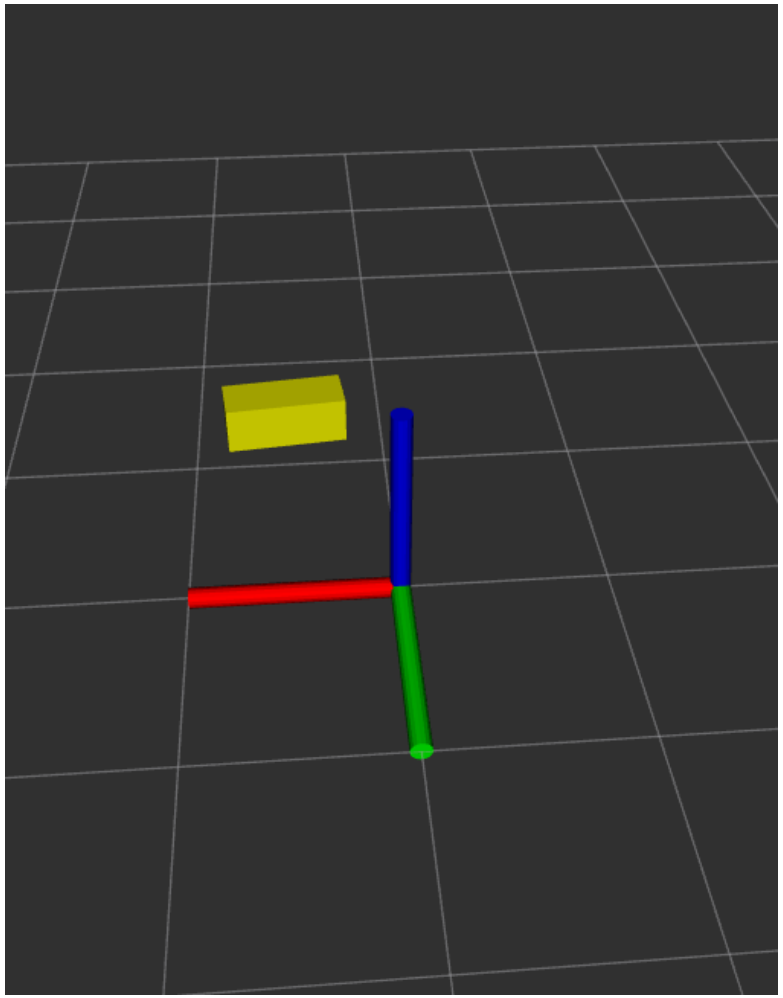
```
ros2 run markers markers
```

- Start RViz

```
ros2 run rviz2 rviz2
```

- Add the marker
  - Press Add
  - ››By topic››/marker›› marker
- Change the fixed frame on top of RViz to “map”
- Visualise the Fixed Frame by adding “Axes” from the “Add” Menu

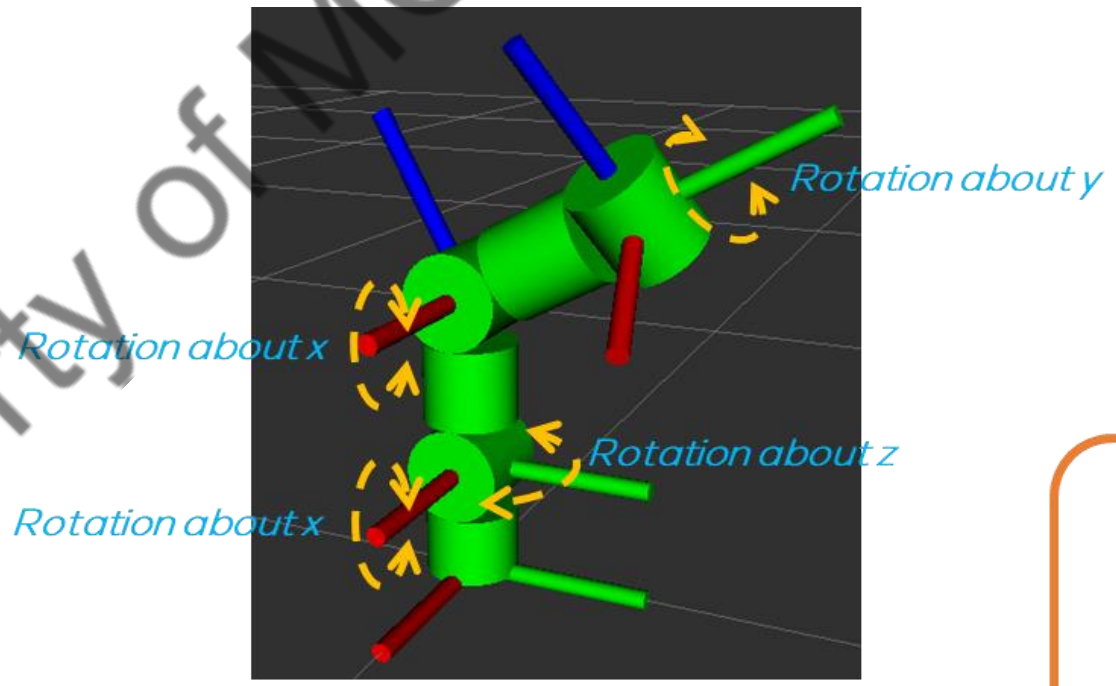


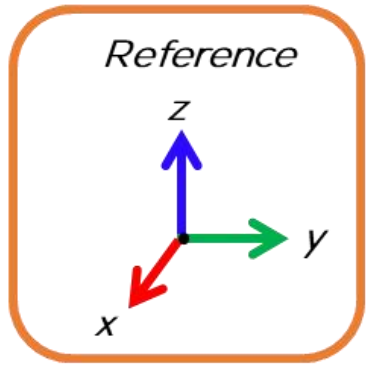


Markers and Transforms

Scale

- One of the best practices in ROS is to attach markers to frames of reference at their origin.
- The movement of the markers should be performed by the frames of reference rather than by the marker itself.
- In other words, the frame must move while the marker is attached to its centre.
- In the previous example the marker was moving around a frame of reference (not recoomended), better to declare a frame, move the frame and keep the marker attached to its origin.

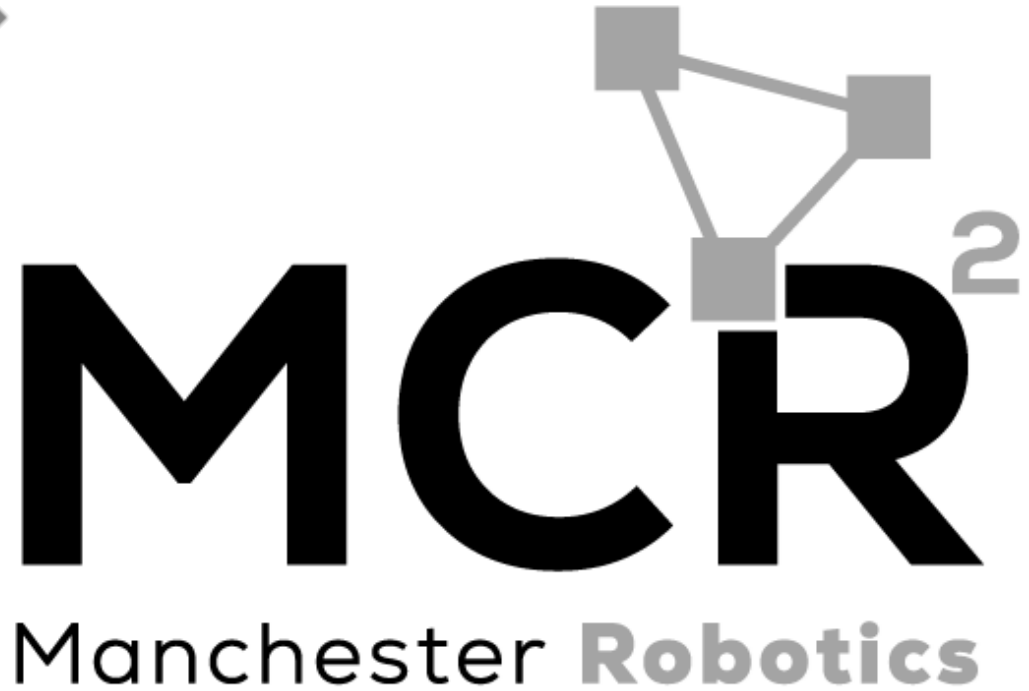




Activity 2

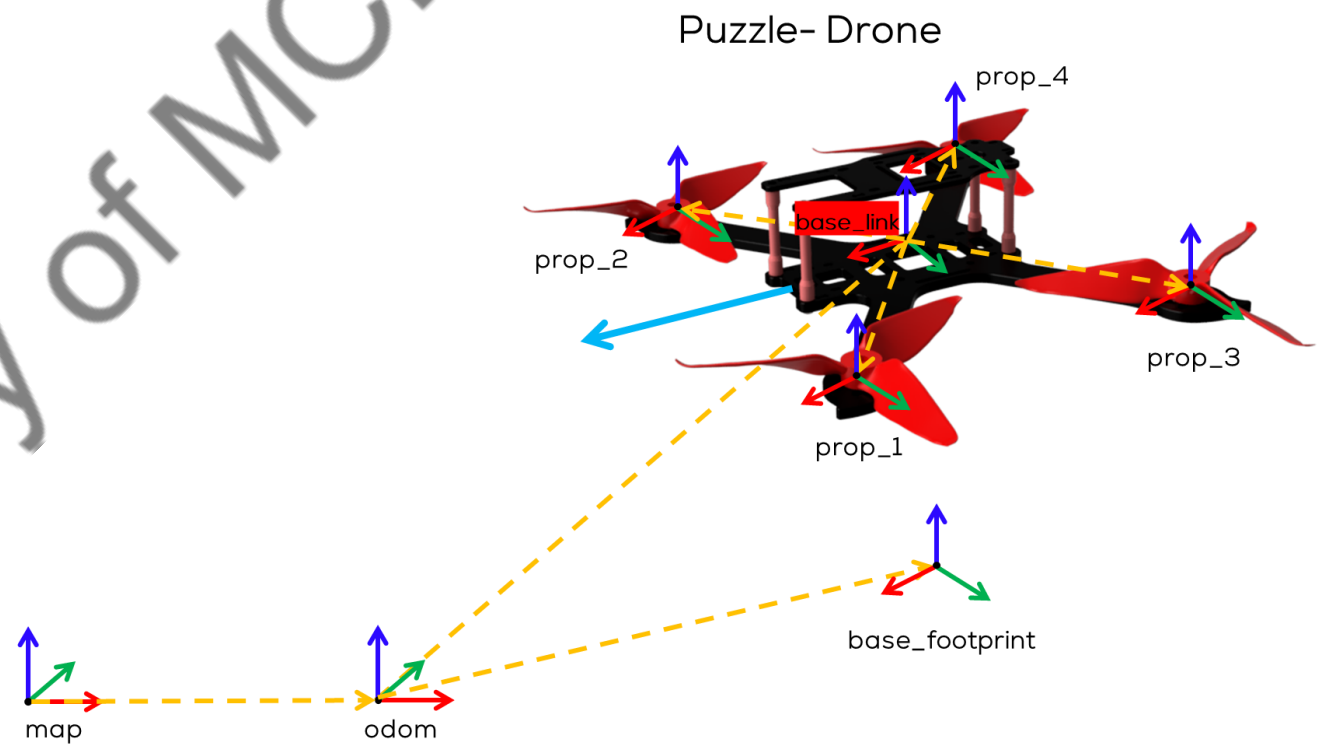
Multiple Markers

{Learn, Create, Innovate};



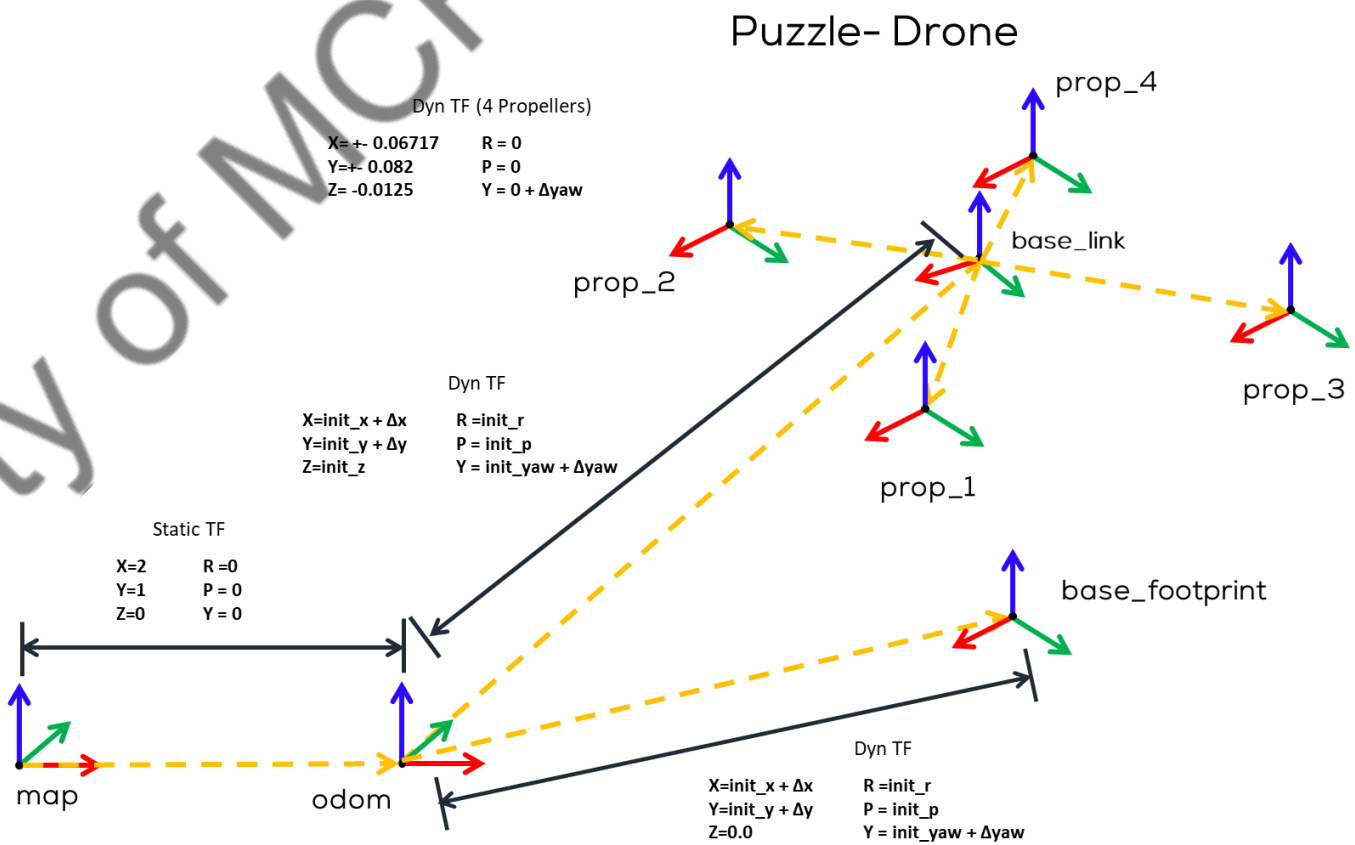
Multiple Markers

- In this activity a simple Puzzle Drone will be modelled.
- To decrease the complexity and decrease computational power, only the basic parts od the Puzzle Drone will be modelled (excl. motors, cables, PCB, etc.)
- The transforms will be set up as follows.
- The markers will be attached to each one of the transforms.
- For this case, a “mesh” type marker will be used.



Multiple Markers

- The following coordinates will be used to define the Drone as Transforms.
- Once the transforms are defined, the markers will be set up to lay on top of the transforms.
- The user should manipulate/control the transforms only, not the Markers.
- For this exercise, the basic on how to define transforms and markers will be shown.
- For this activity, the full code is on GitHub (too long to be on a ppt.)



Multiple Markers

- Make a new node called “PuzzleDrone.py” inside the “markers” package developed previously.

```
$ touch ~/<YOUR_WS>/src/markers/markers/PuzzleDrone.py
```

- Make a launch file called “puzzledrone_launch.py”

```
$ mkdir ~/<YOUR_WS>/src/markers/launch
$ touch ~/<YOUR_WS>/src/markers/launch/puzzledrone_launch.py
```

- Give execution permission to both files
- Modify the previously created “setup.py” as follows

```
data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
        (os.path.join('share', package_name, 'launch'), glob(os.path.join('launch', '*launch.[pxy][yma]*'))),
        (os.path.join('share', package_name, 'config'), glob(os.path.join('config', '*.[yma]*'))),
        (os.path.join('share', package_name, 'rviz'), glob(os.path.join('rviz', '*.rviz'))),
        (os.path.join('share', package_name, 'meshes'), glob(os.path.join('meshes', '*.stl'))),
    ],
```

Multiple Markers

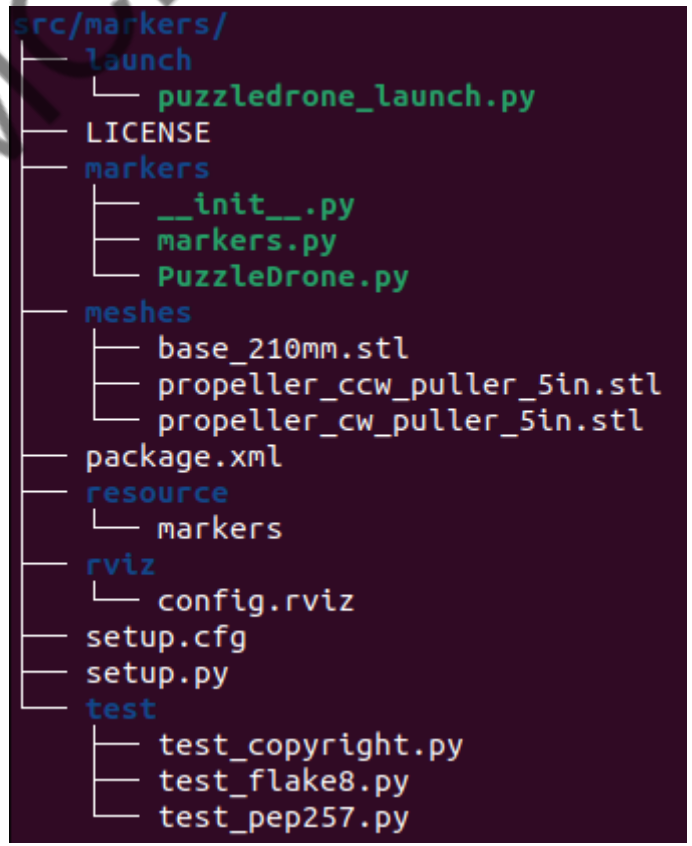
- Make a “meshes” directory in the package “markers”

```
$ mkdir ~/<YOUR_WS>/src/markers/meshes
```

- Download inside the folder ”meshes” the three “.stl” files in GitHub.

```
base_210mm.stl
propeller_ccw_puller_5in.stl
propeller_cw_puller_5in.stl
```

- The folder tree should look as in the image
- Open the “PuzzleDrone.py”
- Type the following code (node initialisation)



Multiple Markers

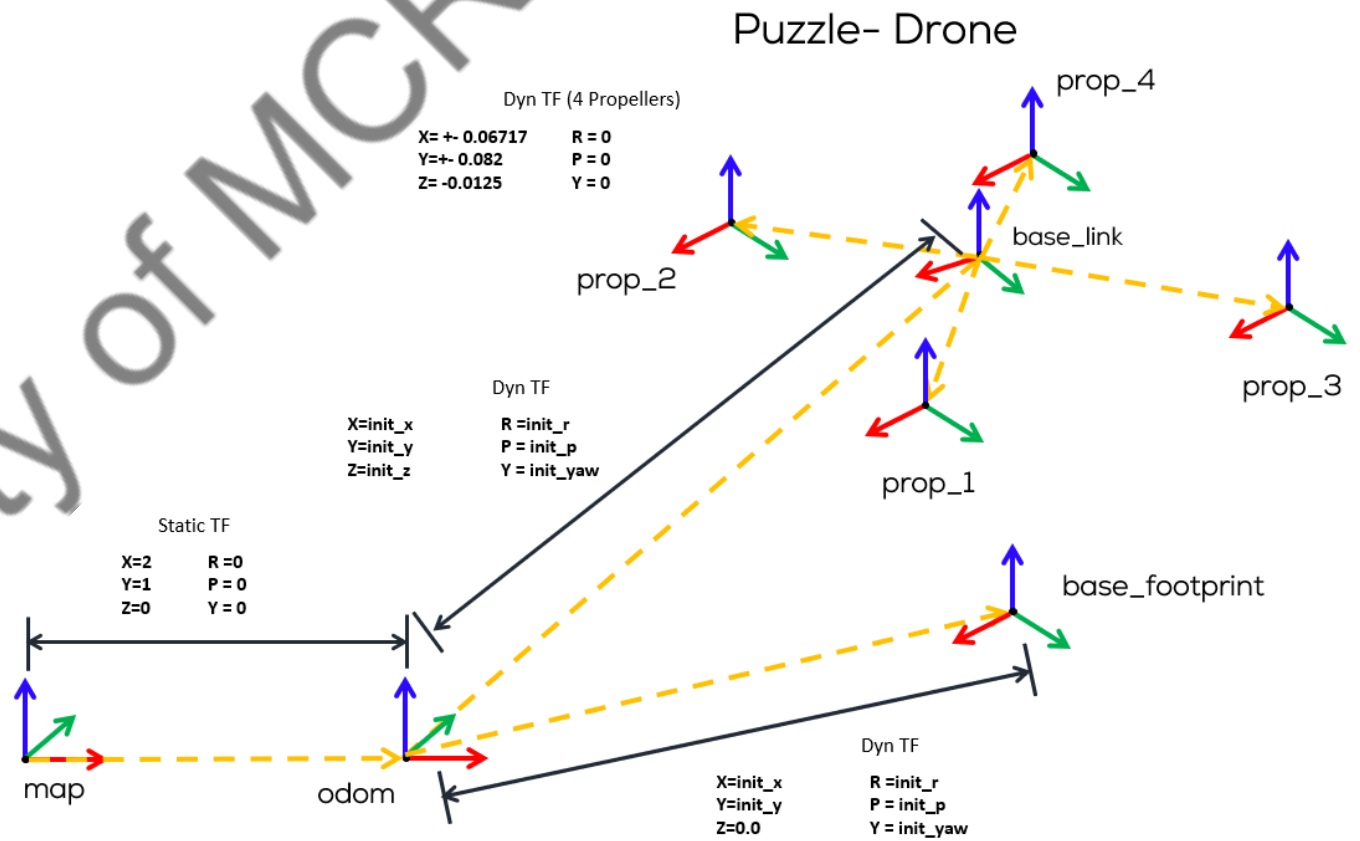
```python
import rclpy
from rclpy.node import Node
from tf2_ros import TransformBroadcaster
from geometry_msgs.msg import TransformStamped
from visualization_msgs.msg import Marker
import transforms3d
import numpy as np

class DronePublisher(Node):

    def __init__(self):
        super().__init__('frame_publisher')

        #Drone Initial Pose
        self.intial_pos_x = 1.0
        self.intial_pos_y = 1.0
        self.intial_pos_z = 1.0
        self.intial_pos_yaw = np.pi/2
        self.intial_pos_pitch = 0.0
        self.intial_pos_roll = 0.0

        #Angular velocity for the pose change and propellers
        self.omega = 0.5
        self.omega_prop = 100.0

        #Define Transformations
        self.define_TF()
        #Define Markers
        self.define_markers()
```

```python
        #Create Transform Boradcasters

        #Create Markers Publishers

        #Create a Timer
        timer_period = 0.01 #seconds
        self.timer = self.create_timer(timer_period, self.timer_cb)

    #Timer Callback
    def timer_cb(self):
        #Callback to be filled

    def define_markers(self):
        #Initialise the markers here

    def define_TF(self):
        #Create Trasnform Messages here

def main(args=None):
    rclpy.init(args=args)
    node = DronePublisher()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    finally:
        if rclpy.ok():  # Ensure shutdown is only called once
            rclpy.shutdown()
        node.destroy_node()

if __name__ == '__main__':
    main()
```

Transforms

- Each transform will be declared as follows
- All transforms will be defined inside the function “define_TF”

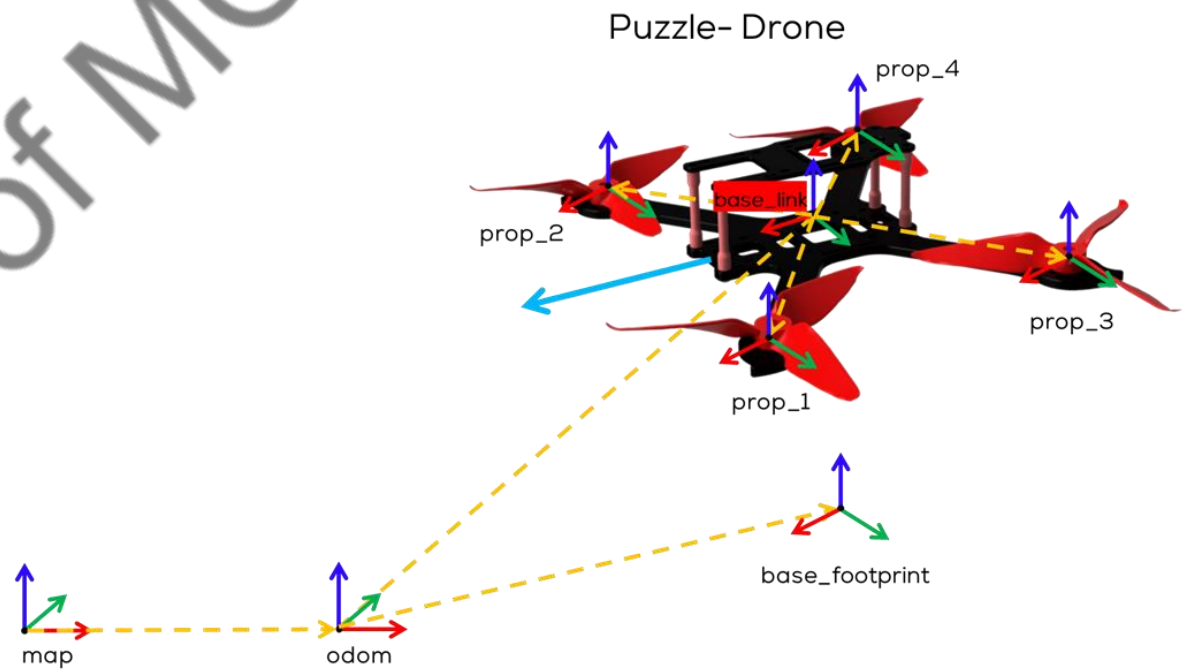
```
def define_TF(self):
    #Create Trasnform Messages
    self.base_footprint_tf = TransformStamped()
    self.base_footprint_tf.header.stamp = self.get_clock().now().to_msg()
    self.base_footprint_tf.header.frame_id = 'odom'
    self.base_footprint_tf.child_frame_id = 'base_footprint'
    self.base_footprint_tf.transform.translation.x = self.intial_pos_x
    self.base_footprint_tf.transform.translation.y = self.intial_pos_y
    self.base_footprint_tf.transform.translation.z = 0.0
    q_foot = transforms3d.euler.euler2quat(self.intial_pos_roll,
self.intial_pos_pitch, self.intial_pos_yaw)
    self.base_footprint_tf.transform.rotation.x = q_foot[1]
    self.base_footprint_tf.transform.rotation.y = q_foot[2]
    self.base_footprint_tf.transform.rotation.z = q_foot[3]
    self.base_footprint_tf.transform.rotation.w = q_foot[0]


... #Define the rest of the transforms
```



Multiple Markers

- The same as with Transforms, the markers must be defined.
- Each marker will be related to a frame and to a mesh (in this case simple marker will not be used)

Marker	Frame/link to attach	Mesh to attach
base	base_link	base_210mm.stl
prop1	prop_1	propeller_ccw_puller_5in.stl
prop2	prop_2	propeller_cw_puller_5in.stl
prop3	prop_3	propeller_ccw_puller_5in.stl
prop4	prop_4	propeller_cw_puller_5in.stl



Multiple Markers

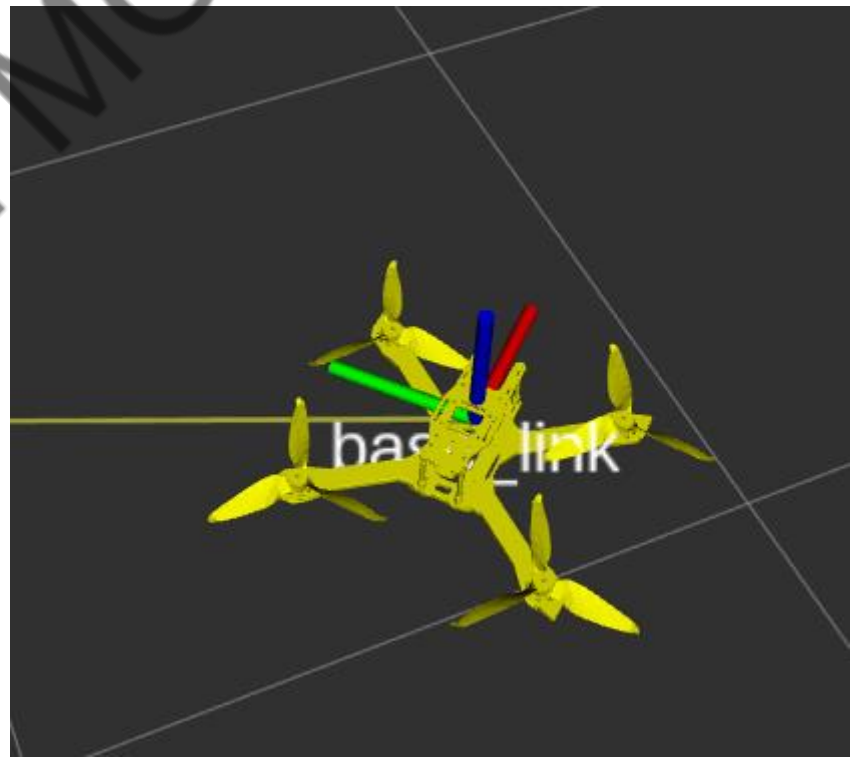
```python
def define_markers(self):

    #initialise the marker
    self.base = Marker()
    self.base.header.frame_id = "base_link"
    self.base.header.stamp = self.get_clock().now().to_msg()
    self.base.id = 0
    self.base.type = Marker.MESH_RESOURCE
    self.base.mesh_resource = "package://markers/meshes/base_210mm.stl"
    self.base.action = Marker.ADD
    self.base.pose.position.x = 0.0
    self.base.pose.position.y = 0.0
    self.base.pose.position.z = -0.0205
    q_base_marker = transforms3d.euler.euler2quat(1.57, 0.0, 1.57)
    self.base.pose.orientation.x = q_base_marker[1]
    self.base.pose.orientation.y = q_base_marker[2]
    self.base.pose.orientation.z = q_base_marker[3]
    self.base.pose.orientation.w = q_base_marker[0]
    self.base.scale.x = 1.0
    self.base.scale.y = 1.0
    self.base.scale.z = 1.0
    self.base.color.r = 1.0
    self.base.color.g = 1.0
    self.base.color.b = 0.0
    self.base.color.a = 1.0
... #Define the rest of the Markers
```

- Some markers requires a small adjustment, because the coordinate frame of the .stl file is not the same as the coordinate frame in ROS.
- In the example the position and orientations of the markers changed to make the coordinate of the .stl to match with the one defined by ROS standards.

Marker	Adjusted Pose
base	x= 0, y=0, z=-0.0205, r=1.57, p= 0.0, yaw=1.57
prop1	x= 0, y=0, z=-0.004, r=0.0, p= 0.0, yaw=0.0
prop2	x= 0, y=0, z=-0.004, r=0.0, p= 0.0, yaw=0.0
prop3	x= 0, y=0, z=-0.004, r=0.0, p= 0.0, yaw=0.0
prop4	x= 0, y=0, z=-0.004, r=0.0, p= 0.0, yaw=0.0

Multiple Markers

- Update the dynamic transforms and the time stamp of the markers.
- The “TF’s” must be updated inside the timer callback.
- For the “markers” only the timestamp must be updated, since they are fixed to a Frame.
- The user must always control the Frames, not the markers.
- If using meshes, the tessellation in the stl files must be as low quality as possible to save computing power for other ROS Tasks (This is just to visualise!)



Multiple Markers

```python
import rclpy
from rclpy.node import Node
from tf2_ros import TransformBroadcaster
from geometry_msgs.msg import TransformStamped
from visualization_msgs.msg import Marker
import transforms3d
import numpy as np

class DronePublisher(Node):

    def __init__(self):
        super().__init__('frame_publisher')

        #Drone Initial Pose
        self.intial_pos_x = 1.0
        self.intial_pos_y = 1.0
        self.intial_pos_z = 1.0
        self.intial_pos_yaw = np.pi/2
        self.intial_pos_pitch = 0.0
        self.intial_pos_roll = 0.0

        #Angular velocity for the pose change and propellers
        self.omega = 0.5
        self.omega_prop = 100.0

        #Define Transformations
        self.define_TF()
        #Define Markers
        self.define_markers()

        #Create Transform Boradcasters
        self.tf_br_base = TransformBroadcaster(self)
        ... #Define the other broadcasters for each transform

        #Create Markers Publishers
        self.base_marker_pub = self.create_publisher(Marker, '/base_marker', 10)
        ... #Define the other publishers for each marker

        #Create a Timer
        timer_period = 0.01 #seconds
        self.timer = self.create_timer(timer_period, self.timer_cb)

    #Timer Callback
    def timer_cb(self):

        time = self.get_clock().now().nanoseconds/1e9

        #Update the time stamp of each marker
        self.base.header.stamp = self.get_clock().now().to_msg()
        ... #Update the time stamp of th other markers

        #Update the Transforms
        self.base_link_tf.header.stamp = self.get_clock().now().to_msg()
        self.base_link_tf.transform.translation.x = self.intial_pos_x + 0.5*np.cos(self.omega*time)
        self.base_link_tf.transform.translation.y = self.intial_pos_y + 0.5*np.sin(self.omega*time)
        self.base_link_tf.transform.translation.z = self.intial_pos_z
        q = transforms3d.euler.euler2quat(self.intial_pos_roll, self.intial_pos_pitch,
self.intial_pos_yaw+self.omega*time)
        self.base_link_tf.transform.rotation.x = q[1]
        self.base_link_tf.transform.rotation.y = q[2]
        self.base_link_tf.transform.rotation.z = q[3]
        self.base_link_tf.transform.rotation.w = q[0]
        ... #Update the other TF's

        self.tf_br_base.sendTransform(self.base_link_tf)
        ... #Broadcast the other TF's

        self.base_marker_pub.publish(self.base)
        ... #Publish the Other Markers
```

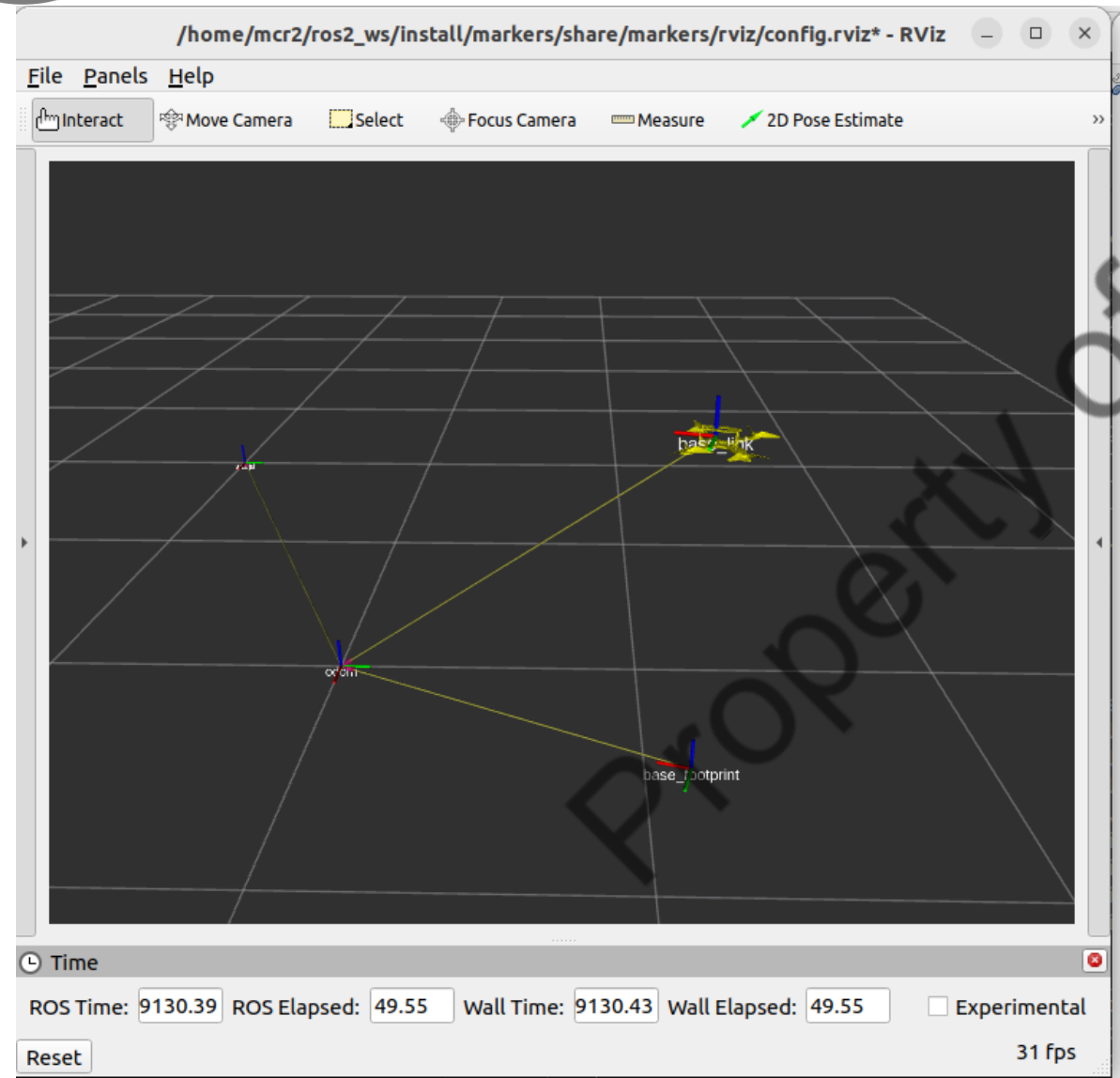
Multiple Markers

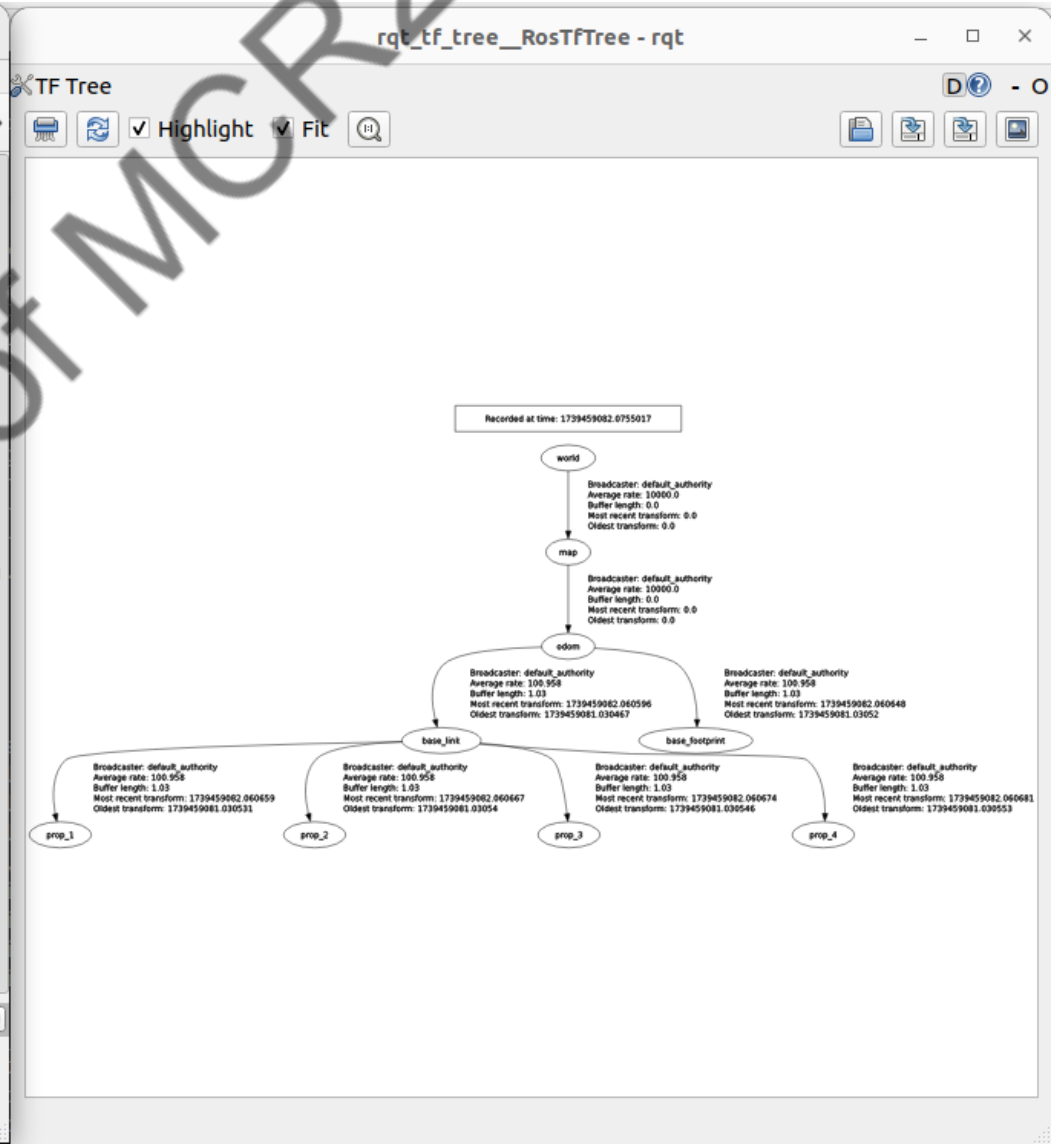
- Open the launch file previously done.
- Define two static transforms
  - world -> map (same pose)
  - map -> odom `[x= 2, y=1, z=-0.0, r=0, p=0.0, yaw=0]`

```
static_transform_node = Node(
                            package='tf2_ros',
                            executable='static_transform_publisher',
                            arguments = ['--x', '2', '--y', '1', '--z', '0.0',
                                         '--yaw', '0.0', '--pitch', '0', '--roll', '0.0',
                                         '--frame-id', 'map', '--child-frame-id', 'odom']
                            )

static_transform_node_2 = Node(
                            package='tf2_ros',
                            executable='static_transform_publisher',
                            arguments = ['--x', '0', '--y', '0', '--z', '0.0',
                                         '--yaw', '0.0', '--pitch', '0', '--roll', '0.0',
                                         '--frame-id', 'world', '--child-frame-id', 'map']
                            )
```

- Add the "PuzzleDrone" node to the launch file.
- You can add RVIZ and TF_Tree to verify the transforms.
- Build the package
- Launch the package
- Add the TF's to RVIZ and the Markers

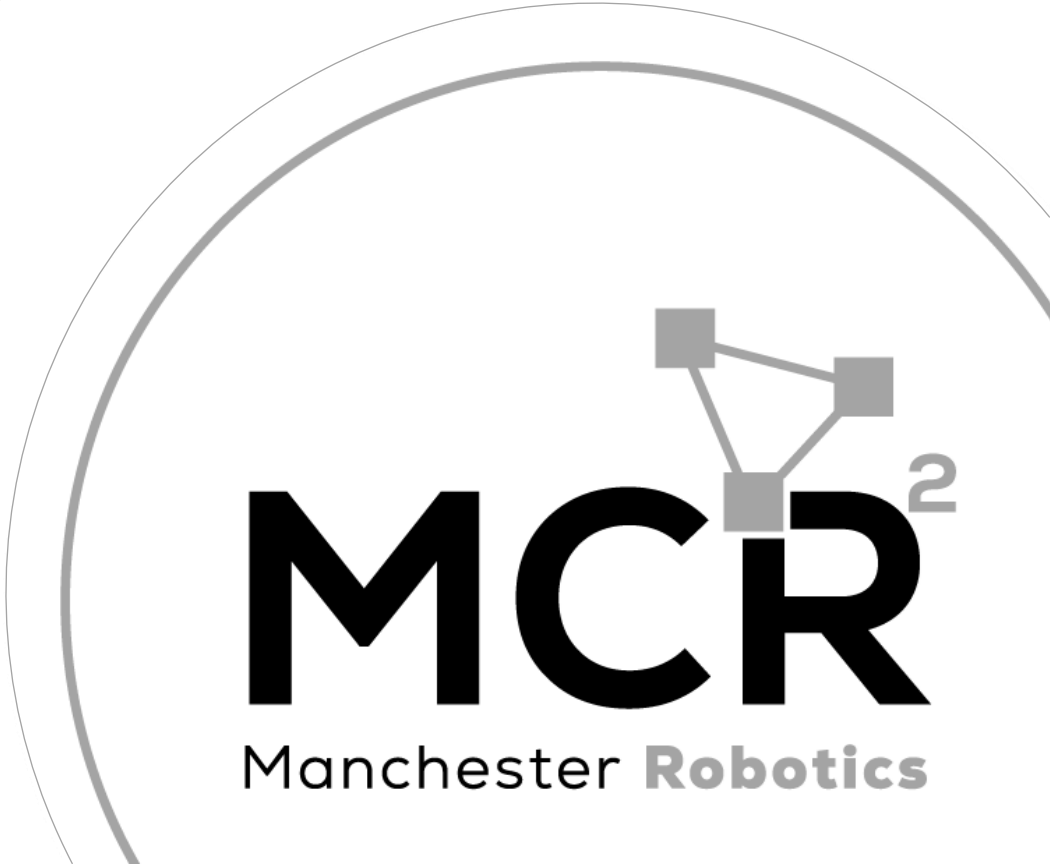
Results





Thank you

{Learn, Create, Innovate};



T&C

Terms and conditions

{Learn, Create, Innovate};

MCR^2

Manchester Robotics

Terms and conditions

- *THE PIECES, IMAGES, VIDEOS, DOCUMENTATION, ETC. SHOWN HERE ARE FOR INFORMATIVE PURPOSES ONLY. THE DESIGN IS PROPRIETARY AND CONFIDENTIAL TO MANCHESTER ROBOTICS LTD. (MCR2). THE INFORMATION, CODE, SIMULATORS, DRAWINGS, VIDEOS PRESENTATIONS ETC. CONTAINED IN THIS PRESENTATION IS THE SOLE PROPERTY OF MANCHESTER ROBOTICS LTD. ANY REPRODUCTION, RESELL, REDISTRIBUTION OR USAGE IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF MANCHESTER ROBOTICS LTD. IS STRICTLY PROHIBITED.*
- *THIS PRESENTATION MAY CONTAIN LINKS TO OTHER WEBSITES OR CONTENT BELONGING TO OR ORIGINATING FROM THIRD PARTIES OR LINKS TO WEBSITES AND FEATURES IN BANNERS OR OTHER ADVERTISING. SUCH EXTERNAL LINKS ARE NOT INVESTIGATED, MONITORED, OR CHECKED FOR ACCURACY, ADEQUACY, VALIDITY, RELIABILITY, AVAILABILITY OR COMPLETENESS BY US.*
- *WE DO NOT WARRANT, ENDORSE, GUARANTEE, OR ASSUME RESPONSIBILITY FOR THE ACCURACY OR RELIABILITY OF ANY INFORMATION OFFERED BY THIRD-PARTY WEBSITES LINKED THROUGH THE SITE OR ANY WEBSITE OR FEATURE LINKED IN ANY BANNER OR OTHER ADVERTISING.*