Lab 2 Report: ROS Installation and `/tf` and Homogeneous Transformations Experiment

Yilin Zhang 23020036094[1]. Lab group: 31.

I. Installing ROS

I followed the official documentation on ROS.org website. [1] It says better to set a mirror while in China, so I chose ustc mirror as I did before. [2] Ustc also offers mirror for `rosdep init`, which is great! [3]

Those bash code above are completely listed in appendix A. I don't see the meaning of this, but it is what you want.

II. ROS Intro

A. ROS file system structure

1) General structure: Along all these files, the ROS packages are the most important and basic units. They contain nodes, libraries, configeration files and others. Just as figure 1 shows, there are mainly four structures: **package manifest**, **messages**, **service**, **code** and others.

According to the *VNAV* website, a package we will develop would be like: [4]

```
.
+-- two_drones_pkg
    |-- CMakeLists.txt
    |-- README.md
    |-- config
    |   +-- default.rviz
    |-- launch
    |   +-- two_drones.launch
    |-- mesh
    |   +-- quadrotor.dae
    |-- package.xml
    +-- src
        |-- frames_publisher_node.cpp
        +-- plots_publisher_node.cpp
```

2) The workspace: In short, the workspace is a directory under `$HOME/$`, such as `/catkin_ws`. It serves as the foundational organizational unit for ROS development. It contains one or more ROS packages, each of which holds source code, configuration files, and other necessary resources. The workspace provides a unified environment in which these packages can be built together—typically via `catkin_make`. This centralized structure is especially useful when managing multiple interdependent packages, as it streamlines compilation and ensures consistent build settings across the entire project.

B. ROS Master, nodes and topics

1) ROS Master: Any ROS system must have one and only one master. A ROS Master transfer imformation about the topics to all nodes. And to start a master we could simply run `roscore` inside a new terminal, the result are pasted in code B.

[1]Yilin zhang is with Faculty of Robotics and Computer Science, Ocean University of China and Heriot-Watt University, China Mainland `zyl8820@stu.ouc.edu.cn`

2) ROS Nodes: A ROS system could have several nodes. One thing to say, turtlesim is a very classic teaching example in all tutorial. It would open a blue window with a turtle in the middle, and we can control the turtle with *wsad* on our keyboard. To create a turtlesim node, we run `rosrun turtlesim turtlesim_node` in a new terminal, and the screen would be like figure 2.

3) ROS Topics: A ROS topic is a channel, and all messages are sent here. In the exercise after, there is a good example.

C. Anatomy of a ROS node

Unlike most ai practice, we use *C++* code instead of *Python*. Although Python has a better support in AI, C++ runs way more faster.

In a simple C++ file, firstly, it should contains a ROS header such as:

```
#include <ros/ros.h>
```

Above all, a `ros::init` call initialize the node:

```
ros::init(argc, argv, "example_node");
```

It is important that the node's name must be unique. This initialization does not contact the master, instead, we use

```
ros::NodeHandle n;
```

This calls a `ros::start()`, and when the last `ros::NodeHandle` is destroyed, it calls `ros::shutdown()`. Usually we want to run our node at a given frequency, to set the node frequency we use

```
ros::Rate loop_rate(50);
```

which is setting the desired rate at 50 Hz.

D. Launch files

The launch files for a package are typically found within its `launch` directory and use the `.launch` file extension. If a package includes such a file, you can start it using the `roslaunch` command.

```
roslaunch <package_name> <launch_file>
```

III. Exercise

A. Basic ROS commands

1) Deliverable 1 - Nodes, topics, launch files: **1.** List the nodes running in the two-drone static scenario.

I used the `rqt_graph` method in a new terminal, unchecked the Debug option, and downloaded the figure, shown in figure 7.

1) /av1broadcaster
2) /av2broadcaster
3) /plots_publisher_node
4) /rviz

2. How could you run the two-drone static scenario without using the roslaunch command? List the commands that you would have to execute, in separate terminals, to achieve the same result.

Just simply run every node in *question 1* one by one. Such as:

```
rosrun av1broadcaster
rosrun av2broadcaster
rosrun plots_publisher_node
rosrun rviz
```

3. List the topics that each node publishes / subscribes to. What nodes are responsible for publishing the av1, av2, frames? Which topic causes rViz to plot the drone meshes?

In the figure that `rqt_graph` method in *question 1*, we can switch the *Node only* to the *Node/Topics (all)* mode. We can see that

- Node `/tf` are resbonsible for publishing the av1 and av2;
- Node `/visuals` points to `/rviz` and it points to `/rosout`.

4. What changes if we omit `static:=True`? Why?

You can't see those two drones, because in `two_drones.launch`C-C1 it says so. The first line says `static` is set to `false` by default. And the code generation of two drones are in a *if* condition fork. If static is false, these code will not run.

B. Publishing the transforms using tf

1) Deliverable 2 - Publishing transforms: We use this time to minus the last time, like

```
double time = (ros::Time::now() -
↪ startup_time).toSec();
```

which returns a ros::Time object. Then we use `toSec()` to transfer it to a second time, and store it into a double variable. The second step *av1* and *av2*'s origin, rotation and frame_ids are set. And finally, they are published by using `sendTransform` as a method in `tf2_ros::TransformBroadcaster`. The whole `C++` code are pasted in appendix C-C2.

2) Changing the rViz fixed reference frame: I changed the Fixed Frame in Global Options in the left from *world* to *av1*, and suddenly av1 stopped moving! The world starts to spit and *av2* moves in a circular shape.

3) Deliverable 3 - Looking up a transform: I added the code below to transfer frames. The whole code file are pasted in appendix C-C3.

```
transform =
↪ parant->tf_buffer.lookupTransform(
        ref_frame,
        dest_frame,
        ros::Time(0),
        ros::Duration(0.1)
);
```

C. Let's do some math

1) Deliverable 4 - Mathematical derivations: We are required to explicitly write down all the homogeneous transformation matrices used in the process and precisely outline the logic and algebraic steps taken.

1. In the problem formulation C-B, we mentioned that AV2's trajectory is an arc of parabola in the $x - z$ plane of the world frame. Can you prove this statement?

Since

$$cos(2t) \; = \; 1 \; - \; 2\,sin^2(t),$$

we have

$$z \; = \; 1 \; - \; 2\,x^2, \tag{1}$$

which is an arc of parabola.

2. Compute $o_2^1(t)$, i.e., the position of AV2 relative to AV1's body frame as a function of t.

We know that:

$$T_2^1 \; = \; (T_1^w)^{-1} \cdot T_2^w. \tag{2}$$

And since

$$\begin{cases} R_1^w = \begin{bmatrix} \cos t & -\sin t & 0 \\ \sin t & \cos t & 0 \\ 0 & 0 & 1 \end{bmatrix}, \\ o_1^w = [\cos t, \sin t, 0]^\top, \end{cases}$$

we have its homogeneous transformation as

$$T_1^w = \begin{bmatrix} R_1^w & o_1^w \\ \mathbf{0}^T & 1 \end{bmatrix} = \begin{bmatrix} \cos t & -\sin t & 0 & \cos t \\ \sin t & \cos t & 0 & \sin t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

In robotics

$$T^{-1} = \begin{bmatrix} R^\top & -R^\top \cdot o \\ 0^\top & 1 \end{bmatrix},$$

so,

$$T_1^{w-1} = \begin{bmatrix} \cos t & -\sin t & 0 & -1 \\ \sin t & \cos t & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3}$$

Similarly, T_2^w's homogeneous transformation is

$$T_2^w = \begin{bmatrix} I & o_2^w \\ \mathbf{0}^T & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \sin t \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \cos(2t) \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4}$$

Thus,

$$\begin{aligned} T_2^1 &= (T_1^w)^\top \, T_2^w \\ &= \begin{bmatrix} \cos t & -\sin t & 0 & -1 \\ \sin t & \cos t & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & \sin t \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \cos(2t) \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos t & -\sin t & 0 & \cos t \sin t - 1 \\ \sin t & \cos t & 0 & \sin^2 t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \end{aligned}$$

That is,

$$o_2^1(t) = \begin{bmatrix}\cos t \cdot \sin t - 1 & -\sin^2 t & \cos 2t\end{bmatrix}^\top. \quad (5)$$

3. Show that $o_2^1(t)$ describes a planar curve and find the equation of its plane Π.

Since

$$\begin{aligned} z(t) &= \cos 2t \\ &= 1 - 2\sin^2 t \\ &= 1 + 2y(t), \end{aligned}$$

we have the every point of the curve satisfying the linear equation

$$z - 2y = 1, \quad (6)$$

which is also the equation of o_2^1's plane Π.

4. Rewrite the above trajectory explicitly using a 2D frame of reference (x_p, y_p) on the plane found before. Try to ensure that the curve is centered at the origin of this 2D frame and that x_p, y_p are axes of symmetry for the curve.

The "HINT" says the new origen of the frame is

$$p' = (-1, -\frac{1}{2}, 0)^\top. \quad (7)$$

So we have

$$\begin{aligned} o(t) &= o_1^1(t) - p' \\ &= \begin{bmatrix}\cos t \cdot \sin t - 1 \\ -\sin^2 t \\ \cos 2t\end{bmatrix} - \begin{bmatrix}-1 \\ -\frac{1}{2} \\ 0\end{bmatrix} \\ &= \begin{bmatrix}\cos t \cdot \sin t \\ -\sin^2 t + \frac{1}{2} \\ \cos 2t\end{bmatrix} \end{aligned} \quad (8)$$

In frame AV1 we define

$$\begin{cases} x_p = \begin{bmatrix}1 & 0 & 0\end{bmatrix}^\top, \\ y_p = \begin{bmatrix}0 & \frac{\sqrt{5}}{5} & \frac{2\sqrt{5}}{5}\end{bmatrix}^\top, \\ z_p = \begin{bmatrix}0 & -\frac{2\sqrt{5}}{5} & \frac{\sqrt{5}}{5}\end{bmatrix}^\top. \end{cases}$$

So,

$$R_p^1 = \begin{bmatrix}x_p^1 & y_p^1 & z_p^1\end{bmatrix} = \begin{bmatrix}1 & 0 & 0 \\ 0 & \frac{\sqrt{5}}{5} & -\frac{2\sqrt{5}}{5} \\ 0 & -\frac{2\sqrt{5}}{5} & \frac{\sqrt{5}}{5}\end{bmatrix}. \quad (9)$$

Therefore,

$$\begin{aligned} o_2^p(t) &= R_p^1 \cdot o(t) \\ &= \begin{bmatrix}1 & 0 & 0 \\ 0 & \frac{\sqrt{5}}{5} & -\frac{2\sqrt{5}}{5} \\ 0 & -\frac{2\sqrt{5}}{5} & \frac{\sqrt{5}}{5}\end{bmatrix} \cdot \begin{bmatrix}\cos t \cdot \sin t \\ -\sin^2 t + \frac{1}{2} \\ \cos 2t\end{bmatrix} \\ &= \begin{bmatrix}\frac{1}{2}\sin 2t \\ -\frac{\sqrt{5}}{2}\cos 2t \\ 0\end{bmatrix}. \end{aligned} \quad (10)$$

5. Using the expression of $o_2^p(t)$, prove that the trajectory of AV2 relative to AV1 is an ellipse and compute the lengths of its semi-axes.

In **question 4**, we reached that

$$o_2^p(t) = \begin{bmatrix}\frac{1}{2}\sin 2t \\ -\frac{\sqrt{5}}{2}\cos 2t \\ 0\end{bmatrix},$$

Obviously, that is

$$\frac{{x_p}^2}{\frac{1}{4}} + \frac{{y_p}^2}{\frac{5}{4}} = 1, \quad (11)$$

and

$$a = \frac{1}{2}, b = \frac{\sqrt{5}}{2} \quad (12)$$

2) Deliverable 5 - More properties of quaternions: In the lecture notes, we have defined two linear maps Ω_1: $\mathbb{R}^4 \to \mathbb{R}^{4\times4}$, and Ω_2: $\mathbb{R}^4 \to \mathbb{R}^{4\times4}$, such that for any $q \in \mathbb{R}^4$, we have:

$$\Omega_1(q) = \begin{bmatrix}q_4 & -q_3 & q_2 & q_1 \\ q_3 & q_4 & -q_1 & q_2 \\ -q_2 & q_1 & q_4 & q_3 \\ -q_1 & -q_2 & -q_3 & q_4\end{bmatrix},$$

$$\Omega_2(q) = \begin{bmatrix}q_4 & q_3 & -q_2 & q_1 \\ -q_3 & q_4 & q_1 & q_2 \\ q_2 & -q_1 & q_4 & q_3 \\ -q_1 & -q_2 & -q_3 & q_4\end{bmatrix}.$$

The product between any two unit quaternions can then be explicitly computed as:

$$q_a \otimes q_b = \Omega_1(q_a)q_b = \Omega_2(q_b)q_a.$$

1. For any unit quaternion q, both $\Omega_1(q)$ and $\Omega_2(q)$ are orthogonal matrices, i.e.,

$$\begin{aligned} \Omega_1(q)^T\Omega_1(q) = \Omega_1(q)\Omega_1(q)^T = I_4, \\ \Omega_2(q)^T\Omega_2(q) = \Omega_2(q)\Omega_2(q)^T = I_4. \end{aligned}$$

Intuitively, what is the reason that both $\Omega_1(q)$ and $\Omega_2(q)$ must be orthogonal?

To make it easy, it can directly calculated. Only when q is a unit quaternion, we have the result of

$$\Omega_1(q)^\top\Omega_1(q) = ||q||^2 I.$$

And for $\Omega_2(q)$, the result would be the same.

2. For any unit quaternion q, both $\Omega_1(q)$ and $\Omega_2(q)$ convert q to be the unit quaternion that corresponds to the 3D identity rotation, i.e.,

$$\Omega_1(q)^T q = \Omega_2(q)^T q = [0, 0, 0, 1]^T.$$

I cannot tell why, but it can alwo be calculated:

$$\begin{bmatrix} q_4 & -q_3 & q_2 & q_1 \\ q_3 & q_4 & -q_1 & q_2 \\ -q_2 & q_1 & q_4 & q_3 \\ -q_1 & -q_2 & -q_3 & q_4 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

3. For any two vectors $x, y \in \mathbb{R}^4$, show the two linear operators commute, i.e.,

$$\Omega_1(x)\Omega_2(y) = \Omega_2(y)\Omega_1(x),$$
$$\Omega_1(x)\Omega_2(y)^T = \Omega_2(y)^T\Omega_1(x).$$

Since,

$$\Omega_1(x)(\Omega_2(y)v) = x \otimes (v \otimes y)$$
$$\Omega_2(y)(\Omega_1(x)v) = (x \otimes v) \otimes y,$$

for every $v \in \mathbb{R}^4$ it should be right.

That is $\Omega_1(x)\Omega_2(y) = \Omega_2(y)\Omega_1(x)$. Similarly, for $\Omega_1(x)\Omega_2(y)^T = \Omega_2(y)^T\Omega_1(x)$, the result would be the same.

3) [Optional] Deliverable 6 - Intrinsic vs Extrinsic rotations: In OUC class this is no "optional". Though I'm very interested in it, I have no time to finish another deliverable.

IV. Some words

This week's work is somehow extremely difficult. Mostly because of the math part. There's a lot of equations and theories that I have never heard before. Everyday in our wechat group, there's someone ran out of anger, talking dirty to this course. Yet, everyone seemed to finish their lab reports. I love my cute classmates not only because of their true heart, not afraid of upper power, but also their true grit, spirit. We overcomed all these difficulties...

References

[1] ROS.org. (2025) Ubuntu install of ros noetic. [Online]. Available: https://wiki.ros.org/cn/noetic/Installation/Ubuntu

[2] U. M. Help. (2025) Ros. [Online]. Available: https://mirrors.ustc.edu.cn/help/ros.html

[3] ——. (2025) Ros distributions. [Online]. Available: https://mirrors.ustc.edu.cn/help/rosdistro.html

[4] M. I. of Technology. (2025) Mit16.485 - visual navigation for autonomous vehicles. [Online]. Available: https://vnav.mit.edu/labs_2023/lab2/ros101.html#general-structure

Appendix A
Installing ROS

These are the actual bash code I used:

Set up keys:

```
gpg --keyserver 'hkp://keyserver.ubuntu.com:80'
↪   --recv-key
↪   C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
gpg --export
↪   C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654 | sudo
↪   tee /usr/share/keyrings/ros.gpg > /dev/null
```

Setup sources.list:

```
sudo sh -c 'echo "deb
↪   [signed-by=/usr/share/keyrings/ros.gpg]
↪   https://mirrors.ustc.edu.cn/ros/ubuntu
↪   $(lsb_release -sc) main" >
↪   /etc/apt/sources.list.d/ros-latest.list'
```

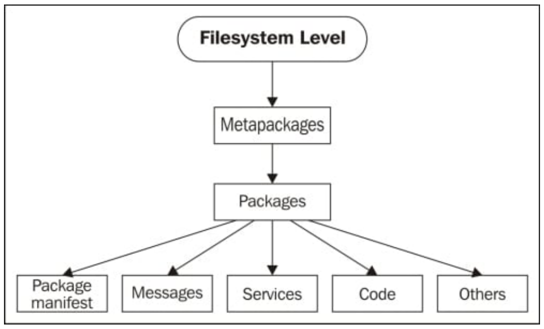
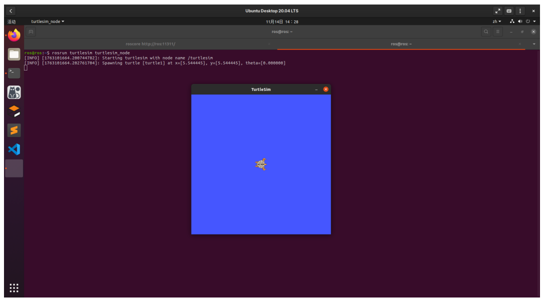


Fig. 1. A tree map of ROS files and folder on the disk



```
ros@ros:~$ rosrun turtlesim turtlesim_node
[INFO] [1763101664.200744782]: Starting turtlesim with node name /turtlesim
[INFO] [1763101664.202761704]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]
```

Fig. 2. A new turtlesim node shows on the screen. [4]

Then we do the Desktop-Full Install:

```
sudo apt update
sudo apt install ros-noetic-desktop-full
```

And we should setup the environment. By using `.bashrc` it will be done automaticly.

```
echo "source /opt/ros/noetic/setup.bash" >>
↪   ~/.bashrc
source ~/.bashrc
```

For `rosdep` installation, we use:

```
sudo mkdir -p /etc/ros/rosdep/sources.list.d/
sudo curl -o
↪   /etc/ros/rosdep/sources.list.d/20-default.list
↪   https://mirrors.ustc.edu.cn/rosdistro/rosdep/so
↪   urces.list.d/20-default.list
sed -i 's#raw.githubusercontent.com/ros/rosdistro/m
↪   aster#mirrors.ustc.edu.cn/rosdistro#g'
↪   /etc/ros/rosdep/sources.list.d/20-default.list

export ROSDISTRO_INDEX_URL=https://mirrors.ustc.edu
↪   .cn/rosdistro/index-v4.yaml
rosdep update

echo 'export ROSDISTRO_INDEX_URL=https://mirrors.us
↪   tc.edu.cn/rosdistro/index-v4.yaml' >> ~/.bashrc

sudo apt-get update
sudo apt install python3-rosdep python3-rosinstall
↪   python3-rosinstall-generator python3-wstool
↪   build-essential

sudo apt install python3-rosdep

sudo rosdep init
rosdep update
```

Appendix B
ROS Intro

A. Quick overview of tf tools

1) Using `rqt_tf_tree`:

```
ros@ros:~$ roscore
... logging to /home/ros/.ros/log/363e3778-c11e-11f⌋
↪  0-acb5-39e7c8db108d/roslaunch-ros-9803.log
Checking log directory for disk usage. This may take
↪  a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ros:44839/
ros_comm version 1.17.4


SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.17.4

NODES

auto-starting new master
process[master]: started with pid [9811]
ROS_MASTER_URI=http://ros:11311/

setting /run_id to
↪  363e3778-c11e-11f0-acb5-39e7c8db108d
process[rosout-1]: started with pid [9884]
started core service [/rosout]
```

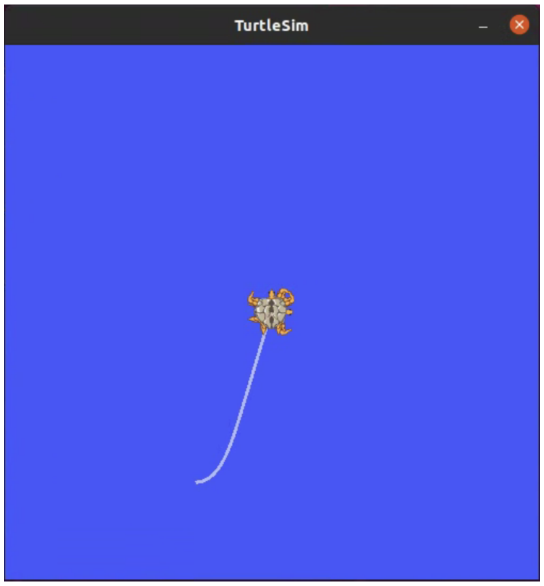


Fig. 3. The window of running `turtle_tf_demo.launch`

2) Using `tf_echo`:

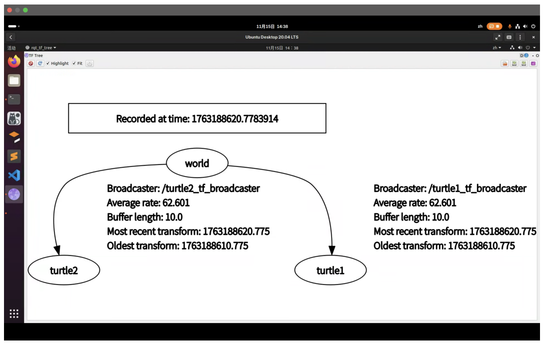
```
ros@ros:~$ rosrun tf tf_echo turtle1 turtle2
At time 1763188806.764
- Translation: [0.000, 0.000, 0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.707,
↪  0.707]
            in RPY (radian) [0.000, -0.000, 1.571]
            in RPY (degree) [0.000, -0.000, 90.000]
```

Appendix C
Exercise

A. Setup workspace

1) Create the catkin workspace: I have done this before, so it may be a little different to the tutorial.

```
ros@ros:~$ mkdir -p ~/vnav_ws/src/
```



Recorded at time: 1763188620.7783914

world

Broadcaster: /turtle2_tf_broadcaster
Average rate: 62.601
Buffer length: 10.0
Most recent transform: 1763188620.775
Oldest transform: 1763188610.775

Broadcaster: /turtle1_tf_broadcaster
Average rate: 62.601
Buffer length: 10.0
Most recent transform: 1763188620.775
Oldest transform: 1763188610.775

turtle2

turtle1

Fig. 4. The window of running `rqt_tf_tree`

```
ros@ros:~$ cd ~/vnav_ws/
ros@ros:~/vnav_ws$ catkin init
Catkin workspace `/home/ros/vnav_ws` is already
↪  initialized. No action taken.
---------------------------------------------------⌋
↪  ---
Profile:                     default
Extending:          [cached] /opt/ros/noetic
Workspace:                   /home/ros/vnav_ws
---------------------------------------------------⌋
↪  ---
Build Space:        [exists] /home/ros/vnav_ws/build
Devel Space:        [exists] /home/ros/vnav_ws/devel
Install Space:      [unused]
↪  /home/ros/vnav_ws/install
Log Space:          [exists] /home/ros/vnav_ws/logs
Source Space:       [exists] /home/ros/vnav_ws/src
DESTDIR:            [unused] None
---------------------------------------------------⌋
↪  ---
Devel Space Layout:          linked
Install Space Layout:        None
---------------------------------------------------⌋
↪  ---
Additional CMake Args:       None
Additional Make Args:        None
Additional catkin Make Args: None
Internal Make Job Server:    True
Cache Job Environments:      False
---------------------------------------------------⌋
↪  ---
Buildlisted Packages:        None
Skiplisted Packages:         None
---------------------------------------------------⌋
↪  ---
Workspace configuration appears valid.
---------------------------------------------------⌋
↪  ---
```

2) Getting the Lab code and Building the code: Then we copy the code from `lab2` to the `src/` directory, and we build and source this new shell.

```
ros@ros:~/vnav_ws$ cp -a
↪  ~/labs/lab2/two_drones_pkg/ ~/vnav_ws/src/
ros@ros:~/vnav_ws$ catkin build
---------------------------------------------------⌋
↪  ---
Profile:                     default
Extending:          [cached] /opt/ros/noetic
Workspace:                   /home/ros/vnav_ws
---------------------------------------------------⌋
↪  ---
Build Space:        [exists] /home/ros/vnav_ws/build
Devel Space:        [exists] /home/ros/vnav_ws/devel
Install Space:      [unused]
↪  /home/ros/vnav_ws/install
Log Space:          [exists] /home/ros/vnav_ws/logs
Source Space:       [exists] /home/ros/vnav_ws/src
DESTDIR:            [unused] None
```

```
--------------------------------------------------↵
↪ ---
Devel Space Layout:          linked
Install Space Layout:        None
--------------------------------------------------↵
↪ ---
Additional CMake Args:       None
Additional Make Args:        None
Additional catkin Make Args: None
Internal Make Job Server:    True
Cache Job Environments:      False
--------------------------------------------------↵
↪ ---
Buildlisted Packages:        None
Skiplisted Packages:         None
--------------------------------------------------↵
↪ ---
Workspace configuration appears valid.
--------------------------------------------------↵
↪ ---
[build] Found 1 packages in 0.0 seconds.
[build] Package table is up to date.
Starting  >>> two_drones_pkg
Finished  <<< two_drones_pkg                [ 0.1
↪ seconds ]
[build] Summary: All 1 packages succeeded!
[build]   Ignored:   None.
[build]   Warnings:  None.
[build]   Abandoned: None.
[build]   Failed:    None.
[build] Runtime: 0.1 seconds total.
ros@ros:~/vnav_ws$ echo "source
↪ $HOME/vnav_ws/devel/setup.bash" >> ~/.bashrc
ros@ros:~/vnav_ws$ source ~/.bashrc
```

3) A two-drone scenario - The static scenario and rViz: With the command

```
roslaunch two_drones_pkg two_drones.launch
↪ static:=True
```

we are able to add `/tf` in the Display channel as figure, and then *CTRL + s* to save the config.

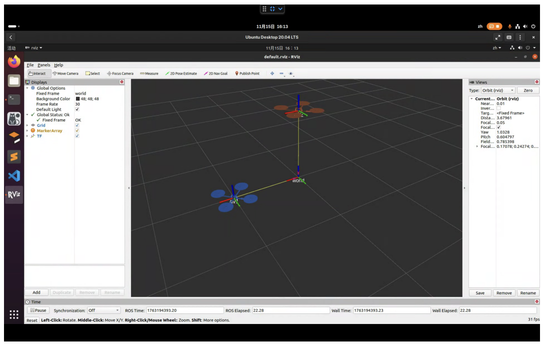


Fig. 5. The RViz window after added `/tf` option.

B. Problem formulation

1) Positions: In the world frame, AV1 and AV2's origins are given by:

$$o_1^w = [\cos(t), \sin(t), 0]^T, \text{and} \quad (13)$$
$$o_2^w = [\sin(t), 0, \cos(2t)]^T, \quad (14)$$

where t denotes time. [4]

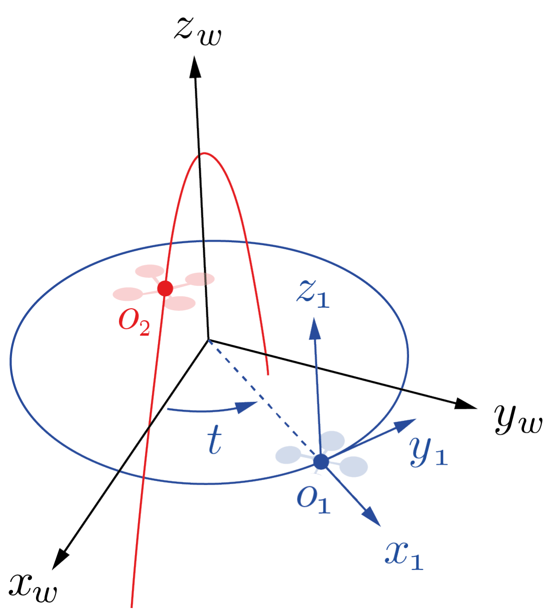


Fig. 6. AV1 [blue] and AV2 [red] are following different trajectories: a circle and an arc of parabola, respectively. [4]

2) Orientations:

- AV1's reference frame is such that y_1 stays tangent to AV1's trajectory for all t and z_1 is parallel to z_w for all t (i.e., equivalently, roll = pitch = 0, yaw = t)
- AV2's reference frame moves with pure translation and we can assume that its axes are parallel to the world axes for all times t [4]

C. Basic ROS commands

1) Deliverable 1 - Nodes, topics, launch files:

```
<launch>
  <arg name="static" default="false"/>

  <!-- Transform publishers !-->
  <group if="$(arg static)">
    <node pkg="tf2_ros"
    ↪ type="static_transform_publisher"
    ↪ name="av1broadcaster" args="1 0 0 0 0 0 1
    ↪ world av1"/>
    <node pkg="tf2_ros"
    ↪ type="static_transform_publisher"
    ↪ name="av2broadcaster" args="0 0 1 0 0 0 1
    ↪ world av2"/>
  </group>

  <node name="frames_publisher_node"
  ↪ pkg="two_drones_pkg"
  ↪ type="frames_publisher_node" unless="$(arg
  ↪ static)"/>

  <!-- Marker publisher -->
  <node name="plots_publisher_node"
  ↪ pkg="two_drones_pkg"
  ↪ type="plots_publisher_node"/>

  <!-- Visualizer -->
  <node name="rviz" pkg="rviz" type="rviz" args="-d
  ↪ $(find two_drones_pkg)/config/default.rviz"/>
</launch>
```

2) Deliverable 2 - Publishing transforms:

```
#include <ros/ros.h>
```

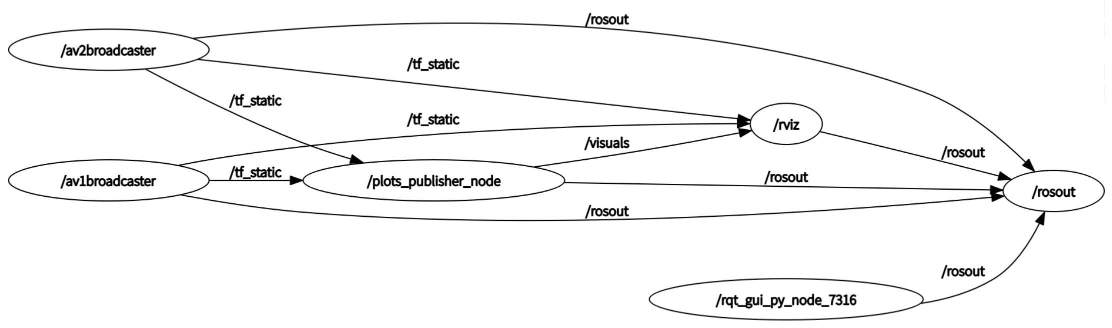


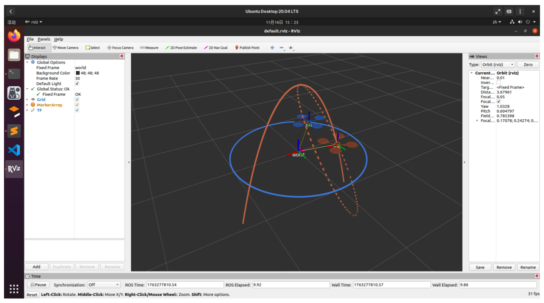
Fig. 7. The nodes are generated by using `rqt_graph`

```
#include <tf2_ros/transform_broadcaster.h>
#include <tf2/LinearMath/Quaternion.h>
#include <tf2_geometry_msgs/tf2_geometry_msgs.h>

#include <iostream>
#include <cmath>

class FramesPublisherNode {
 private:
  ros::NodeHandle nh;
  ros::Time startup_time;

  ros::Timer heartbeat;
  tf2_ros::TransformBroadcaster br;

 public:
  FramesPublisherNode() {
    // NOTE: This method is run once, when the node
    ↪ is launched.
    startup_time = ros::Time::now();
    heartbeat =
        nh.createTimer(ros::Duration(0.02),
        ↪ &FramesPublisherNode::onPublish, this);
    heartbeat.start();
  }

  void onPublish(const ros::TimerEvent&) {
    // NOTE: This method is called at 50Hz, due to
    ↪ the timer created on line 16.

    // 1. Compute time elapsed in seconds since the
    ↪ node has been started
    //    i.e. the time elapsed since startup_time
    ↪ (defined on line 8)
    //   HINTS:
    //   - check out the ros::Time API at
    //     http://wiki.ros.org/roscpp/Overview/Time
    ↪ #Time_and_Duration
    //   - use the - (subtraction) operator between
    ↪ ros::Time::now() and startup_time
    //   - convert the resulting Duration to
    ↪ seconds, store result into a double

    double time = (ros::Time::now() -
    ↪ startup_time).toSec();

    // Here we declare two
    ↪ geometry_msgs::TransformStamped objects,
    ↪ which need to be
    // populated
    geometry_msgs::TransformStamped AV1World;
    geometry_msgs::TransformStamped AV2World;
    // NOTE: fields in a ros message default to
    ↪ zero, so we set an identity transform by
    //       setting just the w component of the
    ↪ rotation
    AV1World.transform.rotation.w = 1.0;
    AV2World.transform.rotation.w = 1.0;

    // 2. Populate the two transforms for the AVs,
    ↪ using the variable "time"
    //    computed above. Specifically:
    //      - AV1World should have origin in
    ↪ [cos(time), sin(time), 0.0] and
    //        rotation such that:
    //         i) its y axis stays tangent to the
    ↪ trajectory and
    //        ii) the z vector stays parallel to that
    ↪ of the world frame
    //      - AV1World should have frame_id "world"
    ↪ and child_frame_id "av1"
    //
    //      - AV2World shoud have origin in
    ↪ [sin(time), 0.0, cos(2*time)], the
    //        rotation is irrelevant to our purpose.
    //      - AV2World should have frame_id "world"
    ↪ and child_frame_id "av2"
    //    NOTE: AV1World's orientation is crucial
    ↪ for the rest fo the assignment,
    //          make sure you get it right
    //
    //    HINTS:
    //    1. check out the ROS tf2 Tutorials:
    ↪ http://wiki.ros.org/tf2/Tutorials,
    //       https://wiki.ros.org/tf2/Tutorials/Addi
    ↪ ng%20a%20frame%20%28C%2B%2B%29#The_Code
    //    2. consider the setRPY method on a
    ↪ tf2::Quaternion for AV1
    //    3. the frame names are crucial for the
    ↪ rest of the assignment,
    //       make sure they are as specified, "av1",
    ↪ "av2" and "world"

    // AV1World
    AV1World.header.stamp = ros::Time::now();
    AV1World.transform.translation.x = cos(time);
    AV1World.transform.translation.y = sin(time);
    AV1World.transform.translation.z = 0.0;

    tf2::Quaternion q;
    q.setRPY(0, 0, time+M_PI_2);
    AV1World.transform.rotation = tf2::toMsg(q);

    AV1World.header.frame_id = "world";
    AV1World.child_frame_id = "av1";

    // AV2World
    AV2World.header.stamp = ros::Time::now();
    AV2World.transform.translation.x = sin(time);
    AV2World.transform.translation.y = 0.0;
    AV2World.transform.translation.z = cos(2*time);

    AV2World.header.frame_id = "world";
    AV2World.child_frame_id = "av2";
    // 3. Publish the transforms using a
    ↪  tf2_ros::TransformBroadcaster
    //    HINTS:
    //         1. you need to define a
    ↪  tf2_ros::TransformBroadcaster as a member of
    ↪  the
    //              node class (line 11) and use its
    ↪  sendTrasform method below

    br.sendTransform(AV1World);
    br.sendTransform(AV2World);
  }
};

int main(int argc, char** argv) {
  ros::init(argc, argv, "frames_publisher_node");
  FramesPublisherNode node;
  ros::spin();
  return 0;
}
```

3) Deliverable 3 - Looking up a transform:

```
#include <geometry_msgs/Point.h>
#include <ros/ros.h>
#include <tf2_ros/transform_listener.h>
#include <visualization_msgs/MarkerArray.h>

#include <iostream>
#include <list>

class PlotsPublisherNode {
  ros::Time startup_time;
  ros::Timer heartbeat;
  ros::NodeHandle nh;
  ros::Publisher markers_pub;
  tf2_ros::TransformListener tf_listener;
  tf2_ros::Buffer tf_buffer;
  int num_trails;

  class TrajTrail {
    PlotsPublisherNode* parent;
    static int id;
    std::list<geometry_msgs::Point> poses;
    std::string ref_frame, dest_frame;
    size_t buffer_size;

    std::string ns;
    float r, g, b, a;

    visualization_msgs::Marker marker_out;

    void update() {
      // NOTE: you need to populate this transform
      geometry_msgs::TransformStamped transform;
      try {
        // ~~~~~~~~~~~~~~~~~~  BEGIN OF EDIT SECTION
        ↪  ~~~~~~~~~~~~~~~~~~~~~

        /* The transform object needs to be
        ↪  populated with the most recent
         * transform from ref_frame to dest_frame as
         ↪  provided by tf.

         * Relevant variables in this scope:
         *   - ref_frame, the frame of reference
         ↪  relative to which the trajectory
         *                 needs to be plotted
         ↪  (given)
         *   - dest_frame, the frame of reference of
         ↪  the object whose trajectory
         *                  needs to be plotted
         ↪  (given)
         *   - parent->tf_buffer, a tf2_ros::Buffer
         ↪  object (given)
         *   - transform, the
         ↪  geometry_msgs::TransformStamped object
         ↪  that needs to be populated
         *
         * HINT: use "lookupTransform", see
         * https://wiki.ros.org/tf2/Tutorials/Writi
         ↪  ng%20a%20tf2%20listener%20%28C%2B%2B%2
         ↪  9#TheCode
         */

        transform =
        ↪  parent->tf_buffer.lookupTransform(
          ref_frame,
          dest_frame,
          ros::Time(0),
          ros::Duration(0.1)
        );

        // ~~~~~~~~~~~~~~~~~~~~  END OF EDIT SECTION
        ↪  ~~~~~~~~~~~~~~~~~~~~~
        while (poses.size() >= buffer_size) {
          poses.pop_front();
        }

        geometry_msgs::Point tmp;
        tmp.x = transform.transform.translation.x;
        tmp.y = transform.transform.translation.y;
        tmp.z = transform.transform.translation.z;
        poses.push_back(tmp);
      } catch (const tf2::TransformException& ex) {
        ROS_ERROR_STREAM("transform lookup failed:
        ↪  " << ex.what());
      }
    }

  public:
    TrajTrail() : parent(nullptr){};

    TrajTrail(PlotsPublisherNode* parent_,
              const std::string& ref_frame_,
              const std::string& dest_frame_,
              int buffer_size_ = 160)
        : parent(parent_),
          ref_frame(ref_frame_),
          dest_frame(dest_frame_),
          buffer_size(buffer_size_) {
      if (buffer_size <= 0) {
        ROS_ERROR_STREAM("invalid buffer size!
        ↪  defaulting to 10");
        buffer_size = 10;
      }

      marker_out.header.frame_id = ref_frame;
      marker_out.ns = "trails";
      marker_out.id = parent->num_trails++;
      marker_out.type =
      ↪  visualization_msgs::Marker::LINE_STRIP;
      marker_out.action =
      ↪  visualization_msgs::Marker::ADD;
      marker_out.color.a = 0.8;
      marker_out.scale.x = 0.02;
      marker_out.lifetime = ros::Duration(1.0);
    }

    void setColor(float r_, float g_, float b_) {
      marker_out.color.r = r_;
      marker_out.color.g = g_;
      marker_out.color.b = b_;
    }

    void setNamespace(const std::string& ns_) {
    ↪  marker_out.ns = ns_; }

    void setDashed() { marker_out.type =
    ↪  visualization_msgs::Marker::LINE_LIST; }

    visualization_msgs::Marker getMarker() {
      update();
      marker_out.header.stamp = ros::Time::now();
      marker_out.points.clear();
      for (auto& p : poses)
      ↪  marker_out.points.push_back(p);
```

```
      if (marker_out.type ==
      ↪  visualization_msgs::Marker::LINE_LIST &&
      ↪  poses.size() % 2)
        marker_out.points.resize(poses.size() - 1);

      return marker_out;
    }
  };

  TrajTrail av1trail;
  TrajTrail av2trail;
  TrajTrail av2trail_rel;

public:
  PlotsPublisherNode() : tf_listener(tf_buffer),
   ↪  num_trails(0) {
    startup_time = ros::Time::now();
    markers_pub = nh.advertise<visualization_msgs::⌋
    ↪  MarkerArray>("visuals", 0);
    heartbeat =
        nh.createTimer(ros::Duration(0.02),
        ↪  &PlotsPublisherNode::onPublish, this);
    heartbeat.start();
    av1trail = TrajTrail(this, "world", "av1", 300);
    av1trail.setColor(0.25, 0.52, 1.0);
    av1trail.setNamespace("Trail av1-world");
    av2trail = TrajTrail(this, "world", "av2", 300);
    av2trail.setColor(0.8, 0.4, 0.26);
    av2trail.setNamespace("Trail av2-world");
    av2trail_rel = TrajTrail(this, "av1", "av2",
    ↪  160);
    av2trail_rel.setDashed();
    av2trail_rel.setColor(0.8, 0.4, 0.26);
    av2trail_rel.setNamespace("Trail av2-av1");

    ROS_INFO_STREAM("Waiting for av1 and av2
    ↪  transforms to be broadcast...");
    while (ros::ok()) {
      const bool av1_present =
      ↪  tf_buffer.canTransform("av1", "world",
      ↪  ros::Time(0));
      const bool av2_present =
      ↪  tf_buffer.canTransform("av2", "world",
      ↪  ros::Time(0));
      if (av1_present && av2_present) {
        ROS_INFO_STREAM("Necessary frames are
        ↪  present, starting!");
        break;
      }
    }
  }

  void onPublish(const ros::TimerEvent&) {
    visualization_msgs::MarkerArray visuals;
    visuals.markers.resize(2);
    visualization_msgs::Marker&
    ↪  av1(visuals.markers[0]);
    visualization_msgs::Marker&
    ↪  av2(visuals.markers[1]);
    av1.header.frame_id = "av1";
    av1.ns = "AVs";
    av1.id = 0;
    av1.header.stamp = ros::Time();
    av1.type =
    ↪  visualization_msgs::Marker::MESH_RESOURCE;
    av1.mesh_resource = "package://two_drones_pkg/m⌋
    ↪  esh/quadrotor.dae";
    av1.action = visualization_msgs::Marker::ADD;
    av1.pose.orientation.w = 1.0;
    av1.scale.x = av1.scale.y = av1.scale.z =
    ↪  av1.color.a = 1.0;
    av1.color.r = 0.25;
    av1.color.g = 0.52;
    av1.color.b = 1.0;
    av1.lifetime = ros::Duration(1.0);
    av2 = av1;
    av2.header.frame_id = "av2";
    av2.ns = "AVs";
    av2.id = 1;
    av2.color.r = 0.8;
    av2.color.g = 0.4;
    av2.color.b = 0.26;

    // Trails
    visuals.markers.push_back(av1trail.getMarker());
    visuals.markers.push_back(av2trail.getMarker());
    visuals.markers.push_back(av2trail_rel.getMarke⌋
    ↪  r());
    markers_pub.publish(visuals);
  }
};

int main(int argc, char** argv) {
  ros::init(argc, argv, "plots_publisher_node");
  PlotsPublisherNode node;
  ros::spin();
  return 0;
}
```


Ubuntu Desktop 20.04 LTS
11月16日 15：23
default.rviz - RViz
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Nav Goal Publish Point
Displays
Views
world
Add Duplicate Remove Rename
Save Remove Rename
Time

```
Global Options
  Fixed Frame        world
  Background Color   48; 48; 48
  Frame Rate         30
  Default Light      ✓
✓ Global Status: Ok
  ✓ Fixed Frame      OK
Grid                 ✓
MarkerArray          ✓
TF                   ✓

Type: Orbit (rviz)
Current...  Orbit (rviz)
  Near...   0.01
  Inver...
  Targ...   <Fixed Frame>
  Dista...  3.67961
  Focal...  0.05
  Focal...  ✓
  Yaw       1.0328
  Pitch     0.604797
  Field...  0.785398
  Focal...  0.17078; 0.24274; 0....

Pause  Synchronization: Off  ROS Time: 1763277810.54  ROS Elapsed: 9.92  Wall Time: 1763277810.57  Wall Elapsed: 9.86
Reset  Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.  31 fps
```

Fig. 8. The path of both avs can be seen after added codes above.