# 34. AR QR code tracking

AR function package location: ~/ArTrack_ws/src/ar_track_alvar/ar_track_alvar

## 34.1. Overview

ARTag (AR tag, AR means "augmented reality") is a fiducial marking system, which can be understood as a reference for other objects. It looks similar to a QR code, but its encoding system is very different from that of a QR code. It is mostly used in camera calibration, robot positioning, augmented reality (AR) and other applications. One of its important functions is to identify the position relationship between the object and the camera. ARTag can be attached to an object or an ARTag tag can be attached to a plane to calibrate the camera. After the camera recognizes the ARTag, it can calculate the position and posture of the tag in the camera coordinates.

ar_track_alvar has 4 main functions:

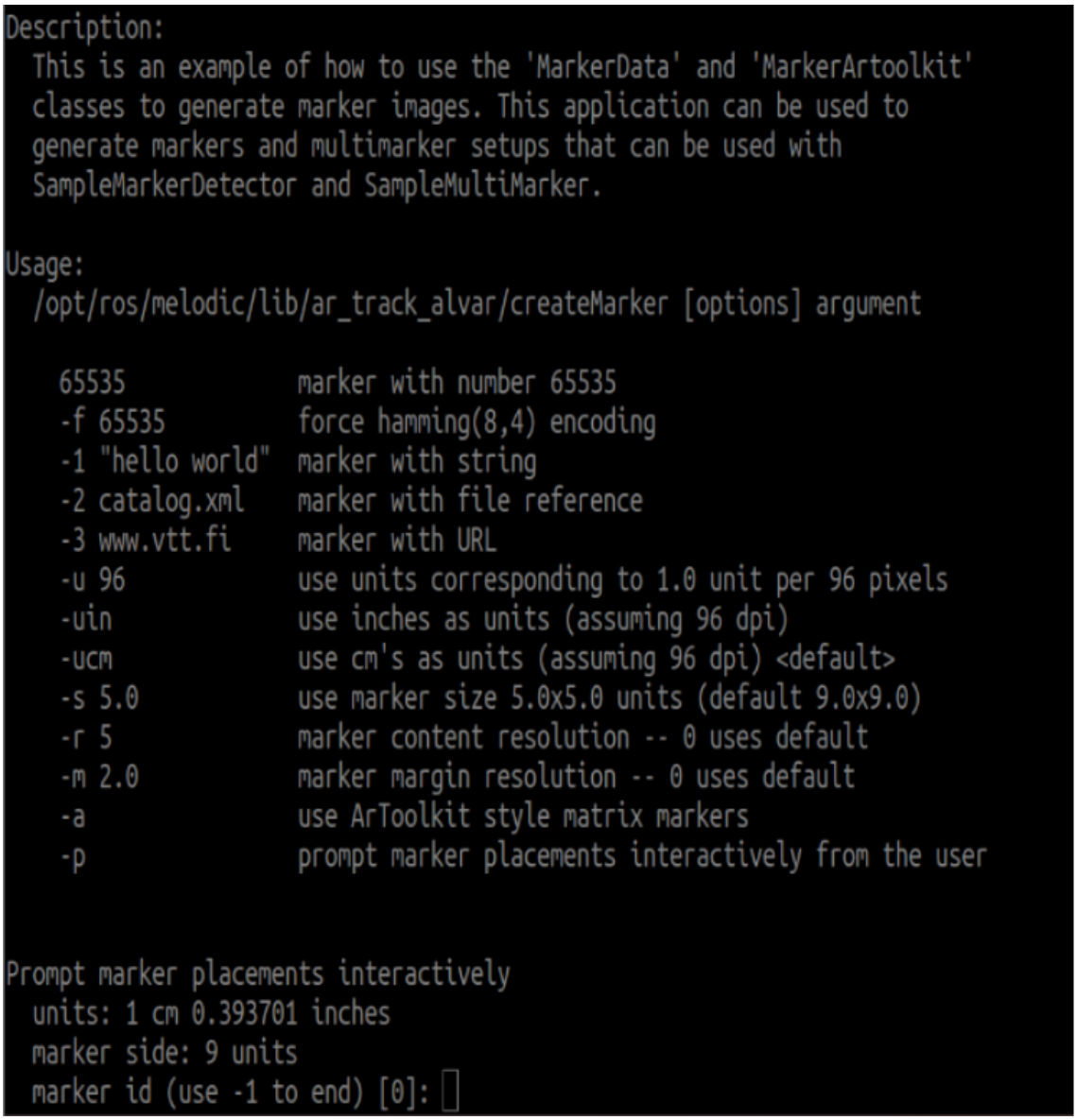
- Generate AR tags of different sizes, resolutions and data/ID codes.
- Identify and track poses of individual AR tags, optionally integrating kinect depth data (when kinect is available) for better pose estimation.
- Automatically compute spatial relationships between tags in a bundle using camera images so that users don't have to manually measure and enter tag positions in an XML file to use bundle functionality.

Alvar is newer and more advanced than ARToolkit, which has been the basis for several other ROS AR tag packages. Alvar features adaptive thresholding to handle various lighting conditions, optical flow-based tracking for more stable pose estimation, and an improved tag identification method that doesn't slow down significantly as the number of tags increases.

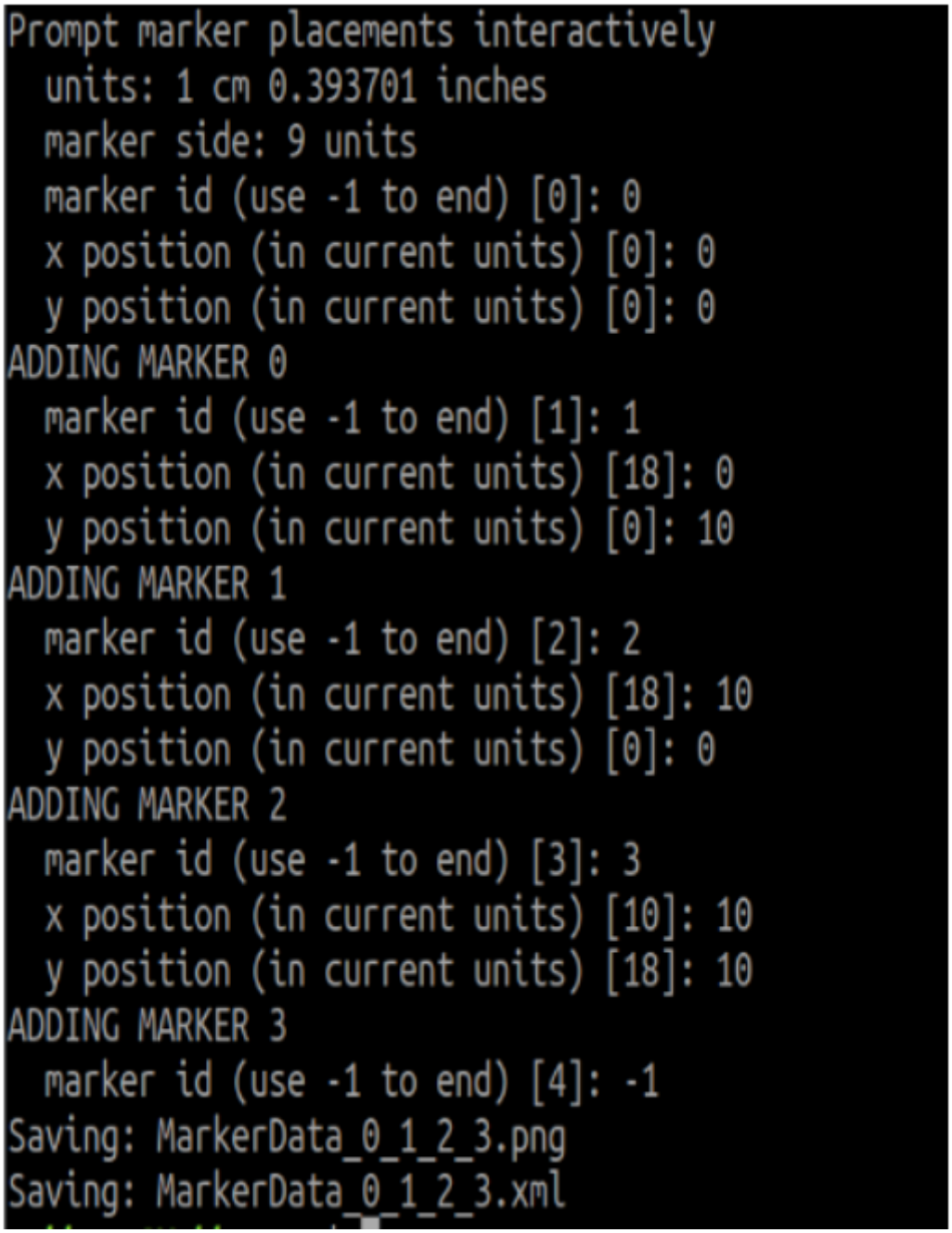
## 34.2. Create ARTag

- Continuously generate multiple tags on one image

```
roscore
rosrun ar_track_alvar createMarker
```




You can enter [ID] and location information here, and enter [-1] to end. You can generate one or more, and design the layout yourself.




- Generate a single number

Command + parameter directly generates a digital image; for example,

```
rosrun ar_track_alvar createMarker 11
rosrun ar_track_alvar createMarker -s 5 33
```

11: The QR code of the number 11. -s: Specify the image size. 5: 5x5 image. 33: The QR code of the number 33.

The generated AR QR code is saved in the directory of the terminal running the command.

## 34.3, ARTag recognition and tracking

Note: When starting the camera, you need to load the camera calibration file, otherwise it cannot be recognized. The supporting virtual machine has calibrated the parameter file required when starting usb_cam. If there is no such parameter file, you need to calibrate it according to the following tutorial.

### 34.3.1, Camera internal parameter calibration

Install the camera calibration function package,

```
sudo apt install ros-noetic-camera-calibration -y
sudo apt install ros-noetic-usb-cam
```

Use usb-cam to drive the camera,

```
roslaunch usb_cam usb_cam-test.launch
```

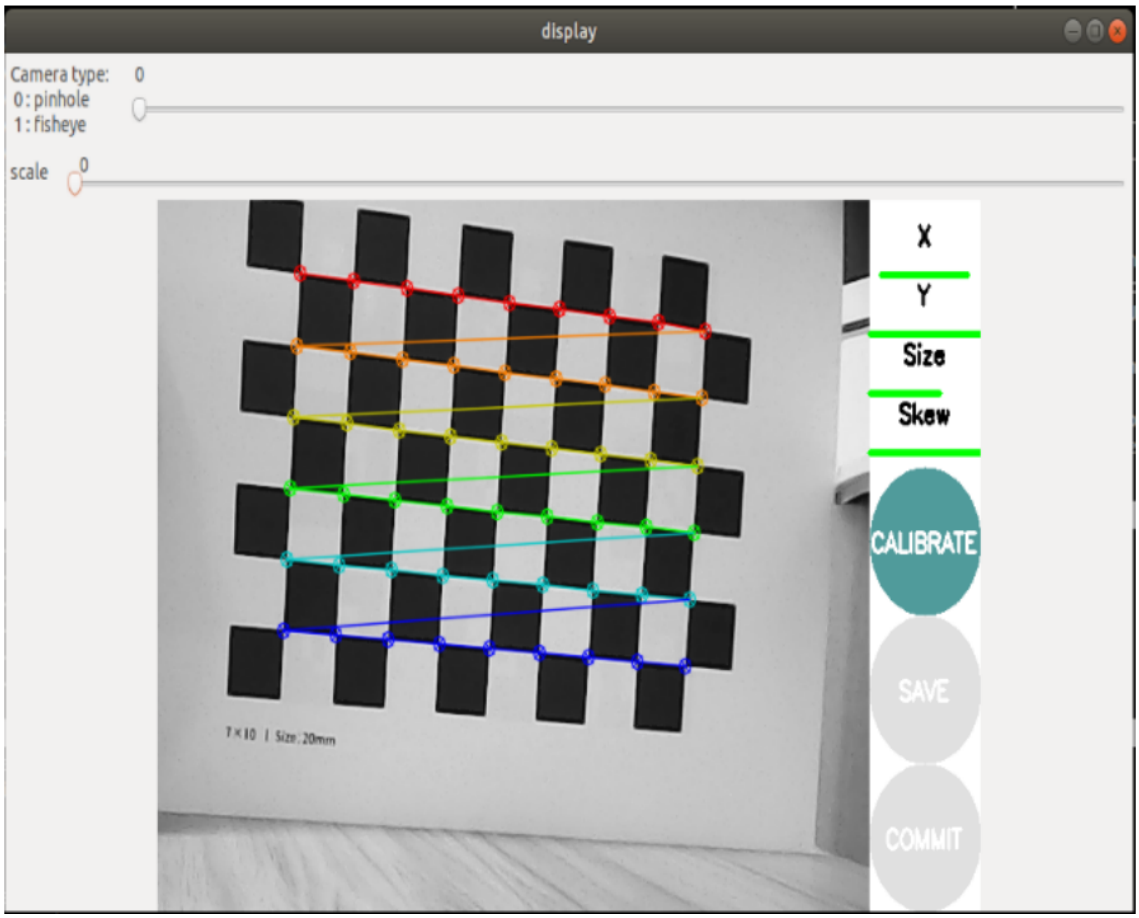
Start the calibration node program,

```
rosrun camera_calibration cameracalibrator.py image:=/usb_cam/image_raw --
size 9x6 --square 0.02
```

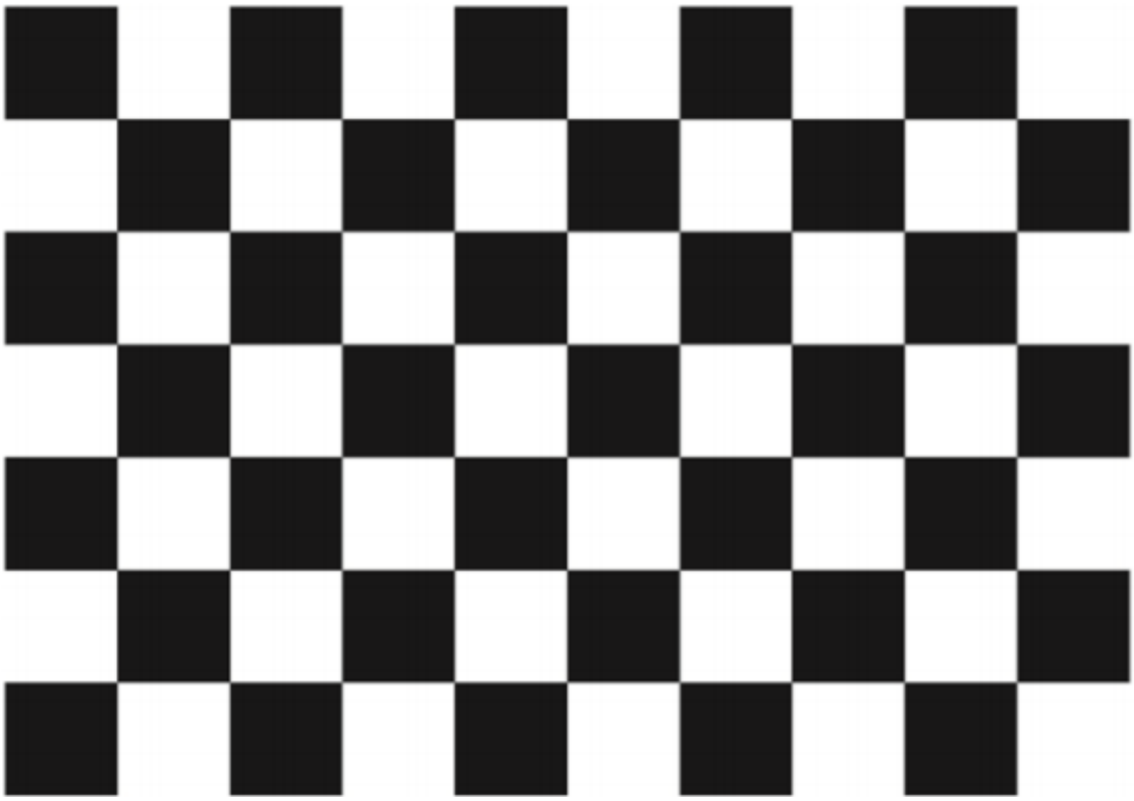
size: the number of internal corner points of the calibration chessboard, for example, 9X6, the corner points have six rows and nine columns.

square: the side length of the chessboard, in meters.

image: set the image topic published by the camera.




Place the prepared calibration chessboard (7*10|size:20mm) in front of the camera,



X: The chessboard moves left and right in the camera's field of view

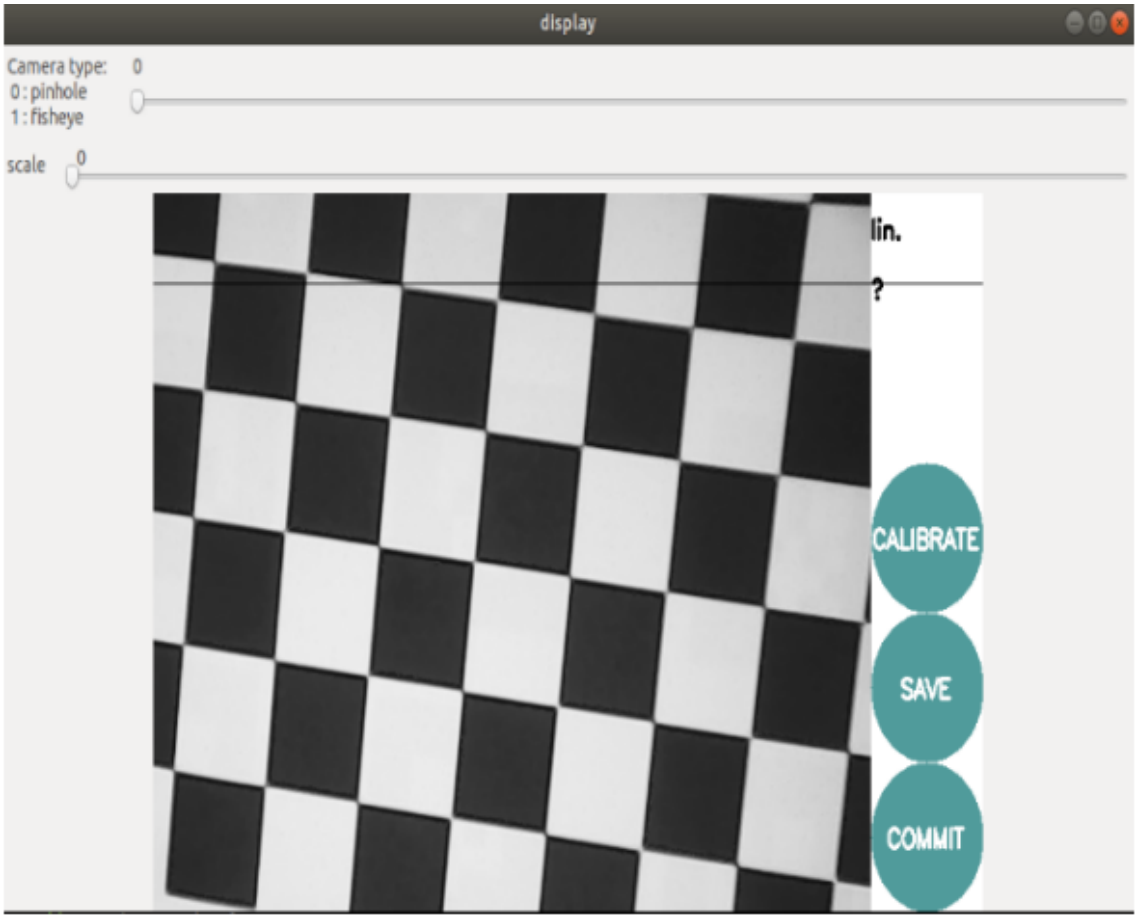
Y: The chessboard moves up and down in the camera's field of view

Size: The chessboard moves forward and backward in the camera's field of view

Skew: The chessboard tilts and rotates in the camera's field of view

After successful startup, put the chessboard in the center of the screen and change different postures. The system will recognize it autonomously. The best situation is that the lines under [X], [Y], [Size], and [Skew] change from red to yellow and then to green as data is collected, and fill them as much as possible.

Click [CALIBRATE] to calculate the camera internal parameters. The more pictures there are, the longer it will take. Just wait. (Sixty or seventy pictures is enough. Too many will cause the camera to get stuck.)




Click [SAVE] to save the calibration results. The calibration results are saved in the directory of the terminal where the calibration program is run. Use the following command to move it out.

```
sudo mv /tmp/calibrationdata.tar.gz ~
```

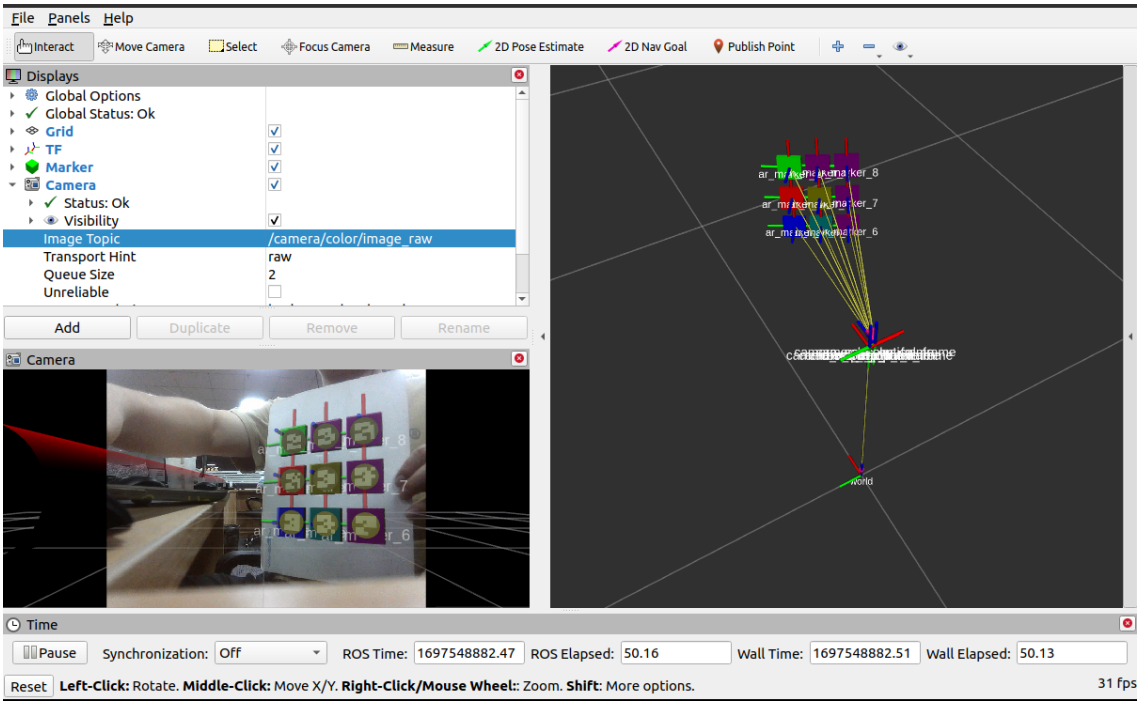
After decompression, there are the pictures just calibrated, an ost.txt file and an ost.yaml file. The yaml file is what we need, but it needs to be modified before it can be used.

- Change ost.yaml to head_camera.yaml
- Move the file to the ~/.ros/camera_info folder

```
cd ~/.ros
sudo mkdir camera_info
sudo cp ~/calibrationdata/head_camera.yaml ~/.ros/camera_info
```

### 34.3.2, Start ARTag recognition and tracking

```
roslaunch astra_visual ar_track.launch
```




In rviz, you need to set the corresponding camera topic name,

- Image_Topic: Obbec's camera is [/camera/color/image_raw].
- Marker: The display component of rviz, different blocks show the location of the AR QR code.
- TF: Display component of rviz, used to display the coordinate system of the AR QR code.
- Camera: Display component of rviz, displaying the camera screen.
- World: World coordinate system.
- Camera_link: Camera coordinate system.

launch file parsing,

```
<launch>
<arg name="open_rviz" default="true"/>
<arg name="marker_size" default="5.0"/>
<arg name="max_new_marker_error" default="0.08"/>
<arg name="max_track_error" default="0.2"/>
<!-- Dabai_dcw2 camera configuration parameters-->
<arg name="cam_image_topic" default="/camera/color/image_raw"/>
<arg name="cam_info_topic" default="/camera/color/camera_info"/>
<arg name="output_frame" default="/camera_link"/>
<!-- Camera launch node-->
<include file="$(find orbbec_camera)/launch/orbbec_camera.launch"/>
<node pkg="tf" type="static_transform_publisher" name="world_to_cam"
args="0 0 0.5 0 0 0 world camera_link 10"/>
<node name="ar_track_alvar" pkg="ar_track_alvar"
type="individualMarkersNoKinect" respawn="false" output="screen">
<param name="marker_size" type="double" value="$(arg marker_size)"/>
<param name="max_new_marker_error" type="double" value="$(arg
max_new_marker_error)"/>
<param name="max_track_error" type="double" value="$(arg max_track_error)"/>
<param name="output_frame" type="string" value="$(arg output_frame)"/>
<remap from="camera_image" to="$(arg cam_image_topic)"/>
<remap from="camera_info" to="$(arg cam_info_topic)"/>
</node>
```

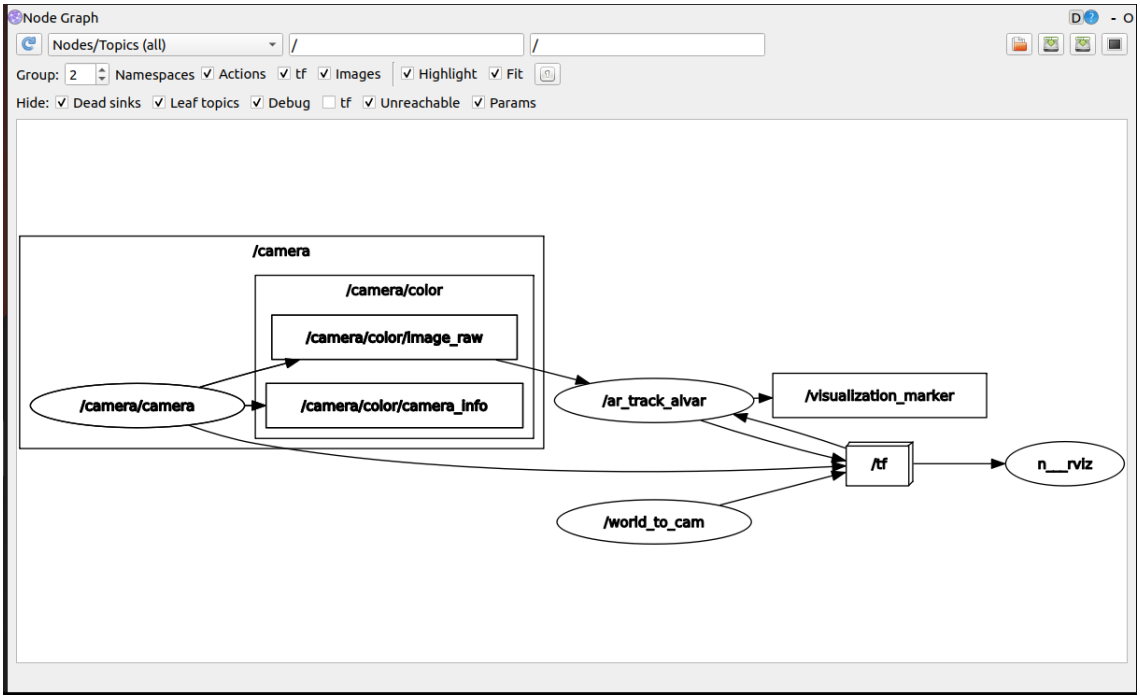
```
<group if="$(arg open_rviz)">
<node pkg="rviz" type="rviz" name="rviz" args="-d $(find
astra_visual)/rviz/ar_track.rviz"/>
</group>
</launch>
```

Node parameters:

- marker_size (double) : The width of one side of the black square marker border in centimeters.
- max_new_marker_error (double) : A threshold for determining when a new marker can be detected under uncertainty.
- max_track_error (double) : A threshold for how much tracking error can be observed before a marker disappears.
- camera_image (string) : Provides the subject name of the image used to detect AR tags. This can be either monochrome or color, but should be an unrectified image, as rectification is performed in this package.
- camera_info (string) : Provides the camera calibration parameters in order to rectify the subject name of the image.
- output_frame (string): The coordinate position of the published AR tag in the camera coordinate system.

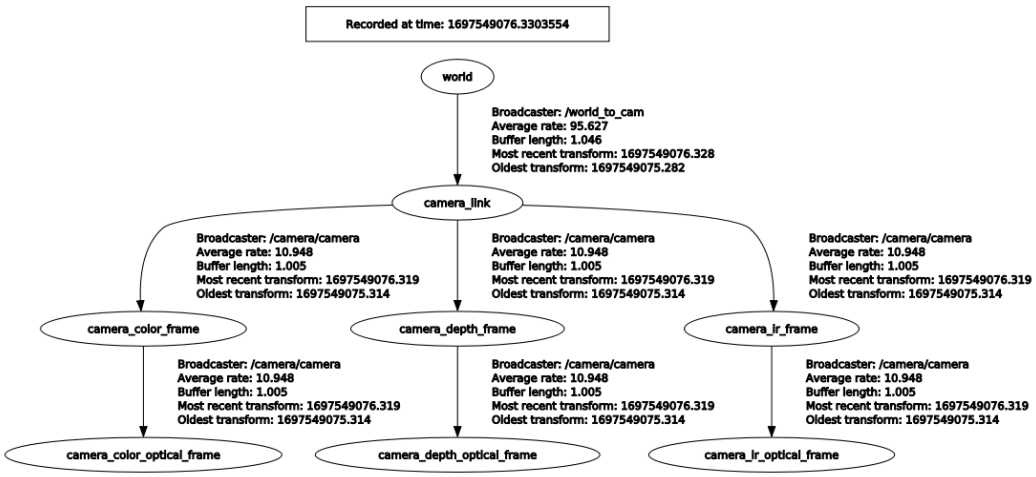
### 34.3.3, View the node graph

```
rqt_graph
```




### 34.3.4, View the TF tree

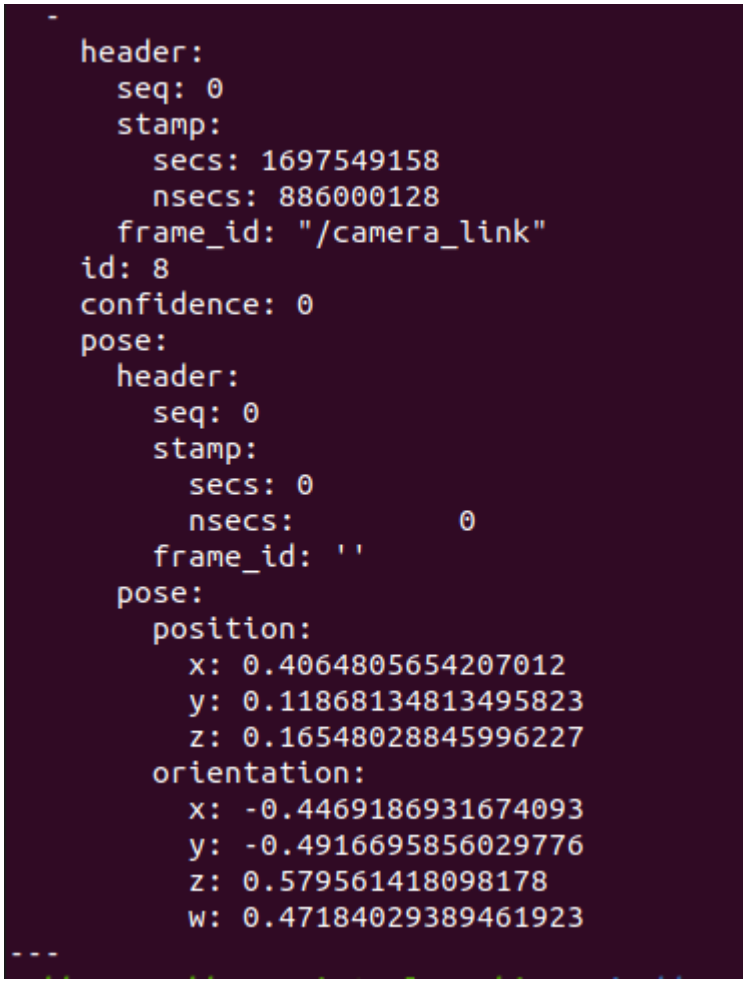
```
rosrun rqt_tf_tree rqt_tf_tree
```




## 34.3.5, View the output topic information

```
rostopic echo /ar_pose_marker
```

Shown as follows:




- frame_id: camera coordinate system name
- id: the recognized number is 8
- pose: QR code pose

- position: the position of the QR code coordinate system relative to the camera coordinate system
- orientation: the orientation of the QR code coordinate system relative to the camera coordinate system