7. AR QR code

7.1. Overview

wiki: http://wiki.ros.org/ar_track_alvar/

Source code: https://github.com/ros-perception/ar_track_alvar.git

Feature pack location: ~/yahboomcar_ws/src/yahboomcar_visual

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

ar_track_alvar has 4 main functions:

- Generate AR tags of different sizes, resolutions, and data/ID encodings.
- Recognize and track the pose of a single AR tag, optionally integrating kinect depth data (when kinect is available) for better pose estimation.
- Recognize and track poses in "bundles" consisting of multiple tags. This allows for more stable pose estimation, robustness to occlusions, and tracking of multilateral objects.。
- Automatically calculate spatial relationships between tags in bundles using camera images, so users don't have to manually measure and enter tag locations in an XML file to use the bundle feature.

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 does not slow down significantly as the number of tags increases.

7.2. Create ARTag

7.2.1. Install software package

file path,

```
~/software/ar_track_ws/src/ar_track_alvar/ar_track_alvar
```

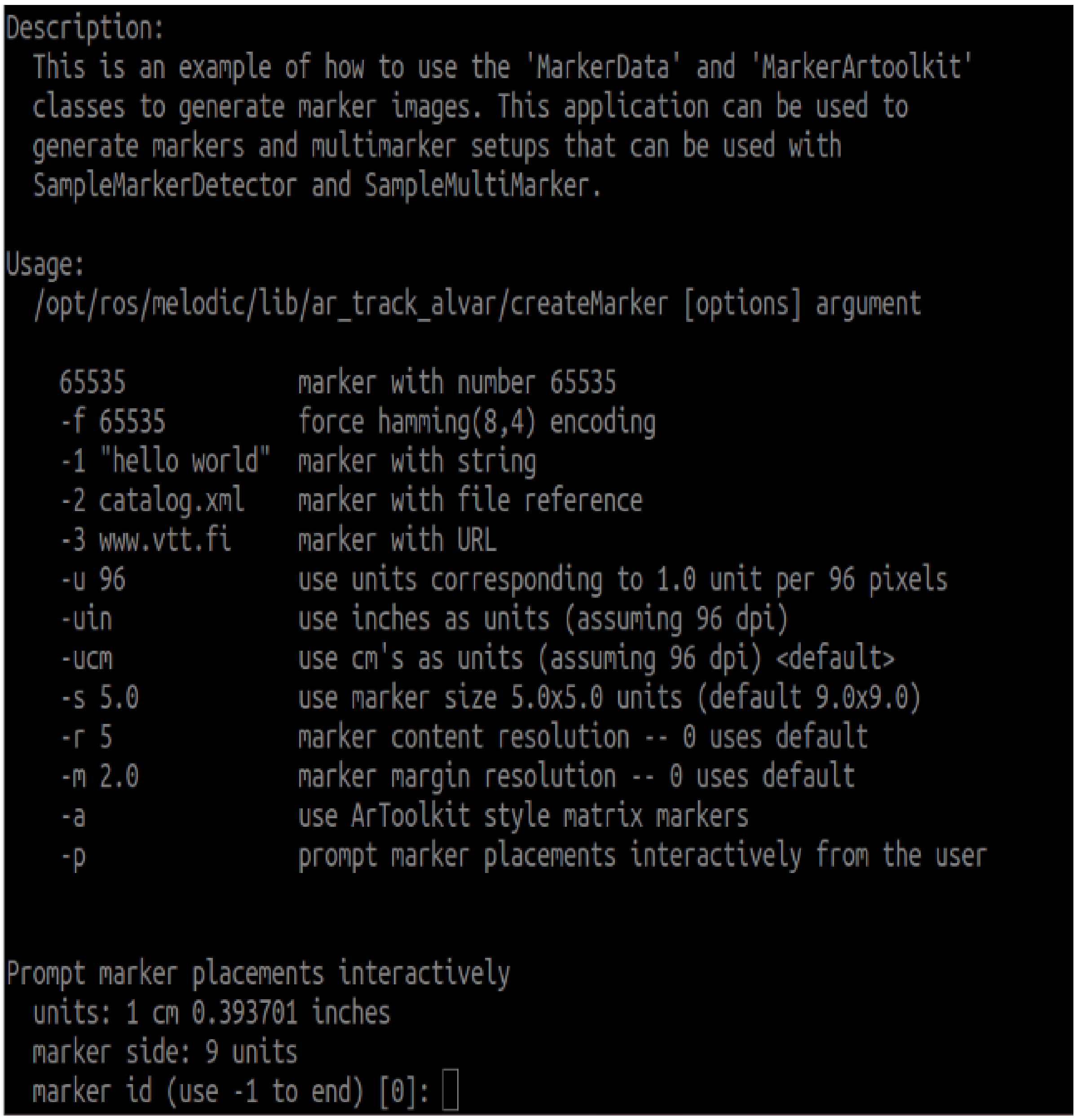
ar_track_alvar is an open source marker library, which provides examples of pr2+kinect. The first use case of this package is to identify and track gestures from (possibly) multiple AR tags, each considered individually.

7.2.2. Create AR QR code

```
#You need to enter docker first, perform this step more
#If running the script to enter docker fails, please refer to Jetson Orin-Docker/05, Enter the robot's docker container
~/run_docker.sh
#Multiple ros commands require multiple terminals to enter the same docker container for execution, please refer to Jetson Orin-Docker/05, Section 5.8 tutorial
```

- Generate multiple labels in a row on an image

```
rosrun ar_track_alvar createMarker
```



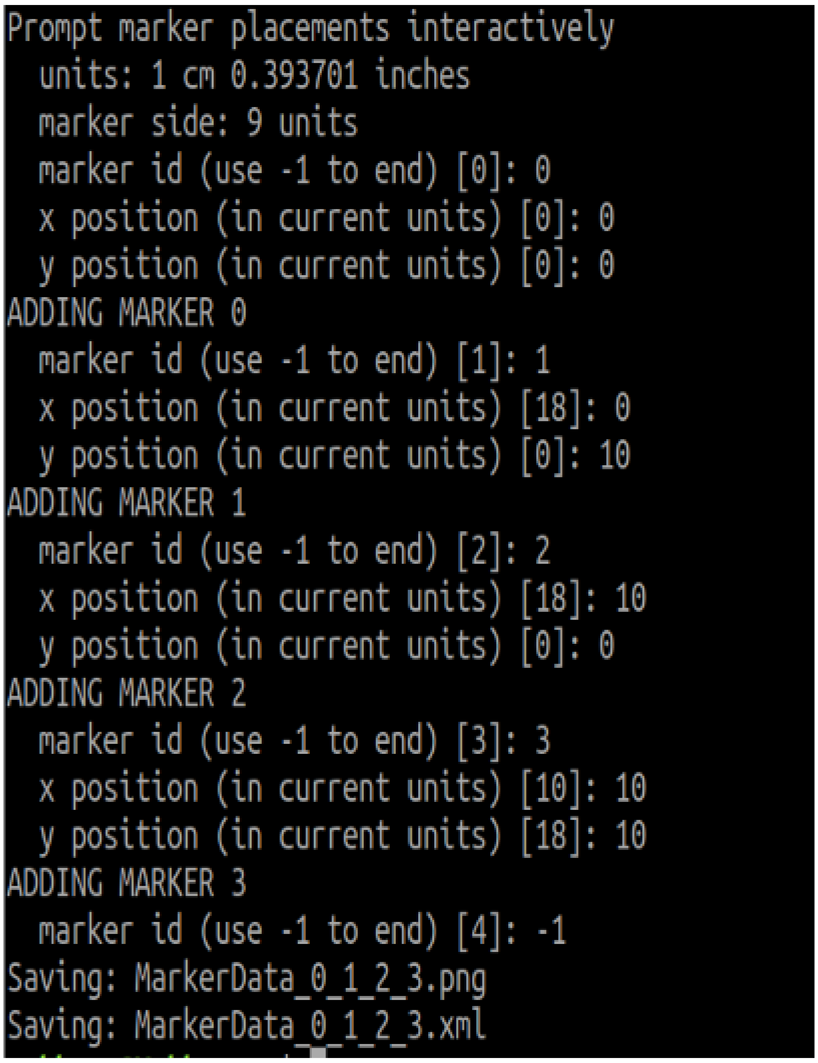
```
Description:
  This is an example of how to use the 'MarkerData' and 'MarkerArtoolkit'
  classes to generate marker images. This application can be used to
  generate markers and multimarker setups that can be used with
  SampleMarkerDetector and SampleMultiMarker.

Usage:
  /opt/ros/melodic/lib/ar_track_alvar/createMarker [options] argument

    65535             marker with number 65535
    -f 65535          force hamming(8,4) encoding
    -1 "hello world"  marker with string
    -2 catalog.xml    marker with file reference
    -3 www.vtt.fi     marker with URL
    -u 96             use units corresponding to 1.0 unit per 96 pixels
    -uin              use inches as units (assuming 96 dpi)
    -ucm              use cm's as units (assuming 96 dpi) <default>
    -s 5.0            use marker size 5.0x5.0 units (default 9.0x9.0)
    -r 5              marker content resolution -- 0 uses default
    -m 2.0            marker margin resolution -- 0 uses default
    -a                use ArToolkit style matrix markers
    -p                prompt marker placements interactively from the user


Prompt marker placements interactively
  units: 1 cm 0.393701 inches
  marker side: 9 units
  marker id (use -1 to end) [0]:
```

You can enter [ID] and location information here, and enter [-1] to end. One or more can be generated, and the layout can be designed by yourself.



```
Prompt marker placements interactively
  units: 1 cm 0.393701 inches
  marker side: 9 units
  marker id (use -1 to end) [0]: 0
  x position (in current units) [0]: 0
  y position (in current units) [0]: 0
ADDING MARKER 0
  marker id (use -1 to end) [1]: 1
  x position (in current units) [18]: 0
  y position (in current units) [0]: 10
ADDING MARKER 1
  marker id (use -1 to end) [2]: 2
  x position (in current units) [18]: 10
  y position (in current units) [0]: 0
ADDING MARKER 2
  marker id (use -1 to end) [3]: 3
  x position (in current units) [10]: 10
  y position (in current units) [18]: 10
ADDING MARKER 3
  marker id (use -1 to end) [4]: -1
Saving: MarkerData_0_1_2_3.png
Saving: MarkerData_0_1_2_3.xml
```

- Generate a single number

Command + parameters directly generate digital pictures; for example

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

11： The number is the QR code of 11. -s： Specifies the image size. 5： 5x5 picture. 33： The number is the QR code of 33. The generated pictures are stored in the directory of the current terminal.

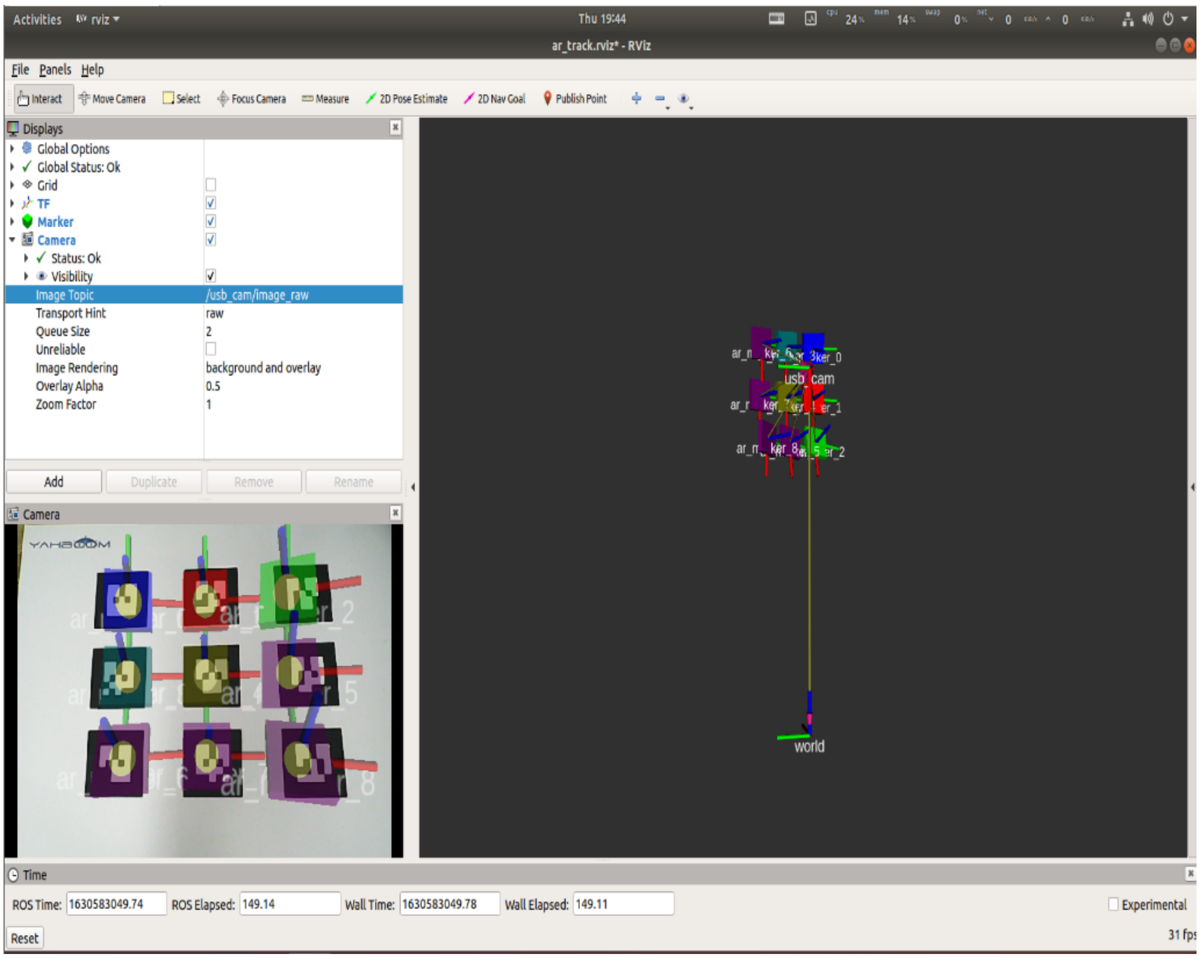
7.3. ARTag identification

Note: When starting the camera, the camera calibration file needs to be loaded, otherwise it cannot be recognized.

7.3.1. Start the identification instance

```
roslaunch yahboomcar_visual ar_track.launch open_rviz:=true
```

- The open_rviz parameter is turned on by default.



Global Options	
Global Status: Ok	
Grid	☐
TF	✓
Marker	✓
Camera	✓
Status: Ok	
Visibility	✓
Image Topic	/usb_cam/image_raw
Transport Hint	raw
Queue Size	2
Unreliable	☐
Image Rendering	background and overlay
Overlay Alpha	0.5
Zoom Factor	1

ROS Time: 1630583049.74 ROS Elapsed: 149.14 Wall Time: 1630583049.78 Wall Elapsed: 149.11

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

- Image_Topic：The camera topic is [/usb_cam/image_raw].
- Marker：The display component of rviz. Different squares display the location of the AR QR code.
- TF：The display component of rviz, used to display the coordinate system of AR QR codes.
- Camera：The display component of rviz, which displays the camera screen.
- world：world coordinate system.
- usb_cam：camera coordinate system.

7.3.2. launch file analysis

```
<launch>
    <!-- Set camDevice parameters，the default is USBCam -->
    <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"/>
    <!-- Set camera image topic, camera internal reference topic, camera frame --
>
    <arg name="cam_image_topic" default="/usb_cam/image_raw"/>
    <arg name="cam_info_topic" default="/usb_cam/camera_info"/>
    <arg name="output_frame" default="/usb_cam"/>
    <!-- Start camera node -->
    <include file="$(find usb_cam)/launch/usb_cam-test.launch"/>
    <!-- Set the corresponding relationship between the camera coordinate system
and the world coordinate system -->
```

```
    <node pkg="tf" type="static_transform_publisher" name="world_to_cam" args="0
0 0.5 0 1.57 0 world usb_cam 10"/>
    <!-- Start AR recognition node -->
    <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>
    <!-- start rviz -->
    <node pkg="rviz" type="rviz" name="rviz" args="-d $(find
yahboomcar_visual)/rviz/ar_track.rviz" if="$(arg open_rviz)"/>
</launch>
```

Node parameters:

- marker_size (double)： Width (in centimeters) of one side of the black square marker border.
- max_new_marker_error (double)： Threshold that determines when a new marker can be detected under uncertain conditions.
- max_track_error (double)： A threshold that determines how many tracking errors can be observed before the marker disappears.
- camera_image (string)： Provides the image topic name used to detect AR tags. This can be monochrome or color, but should be an uncorrected image since correction is done in this package.
- camera_info (string)： Subject name that provides camera calibration parameters to correct images.
- output_frame (string)： Publish the coordinate position of the AR label in the camera coordinate system.

7.3.3. ar_track_alvar node

- **Subscribed topic**

topic name	type of data
/camera_info	(sensor_msgs/CameraInfo)
/image_raw	(sensor_msgs/Image)

- **Published Topics**

topic name	type of data
/visualization_marker	(visualization_msgs/Marker)
/ar_pose_marker	(ar_track_alvar/AlvarMarkers)

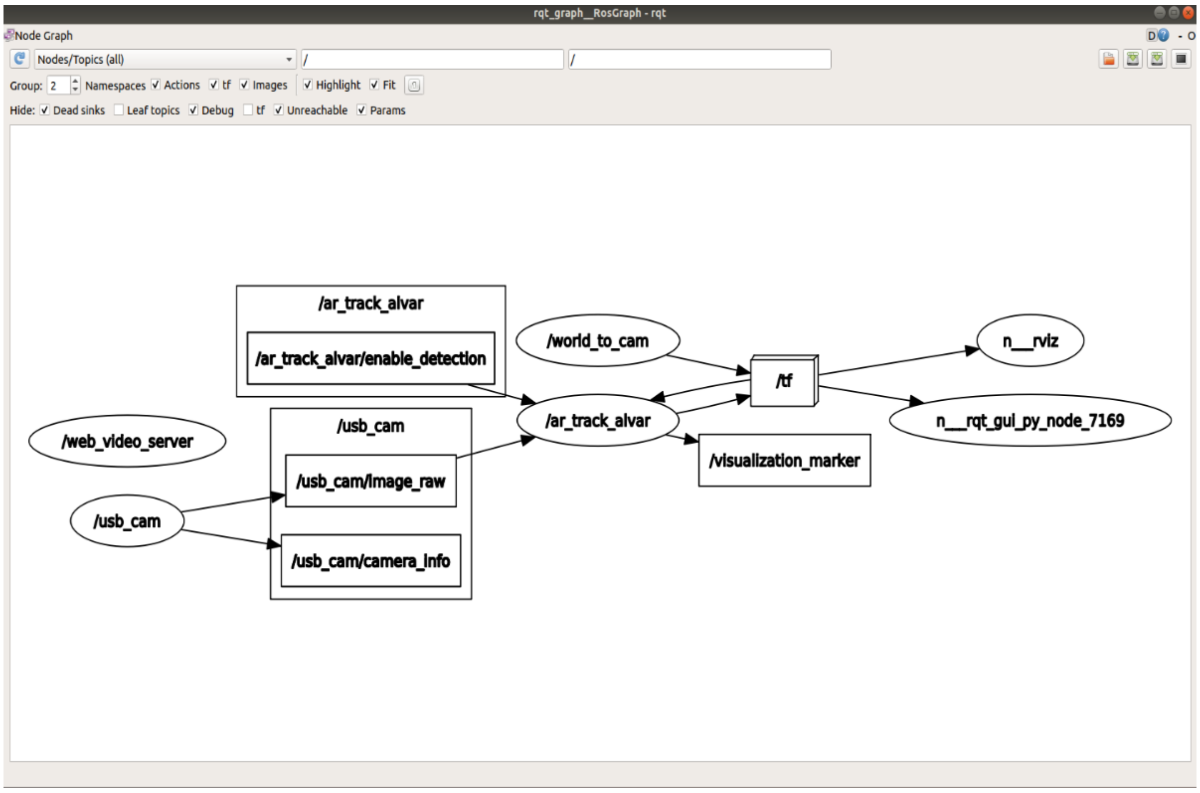
- **Provided tf Transforms**

Single QR code: Camera coordinate system → AR tag coordinate system

Multiple QR codes: Provides transformation from the camera coordinate system to each AR tag coordinate system (named ar_marker_x), where x is the ID number of the marker.

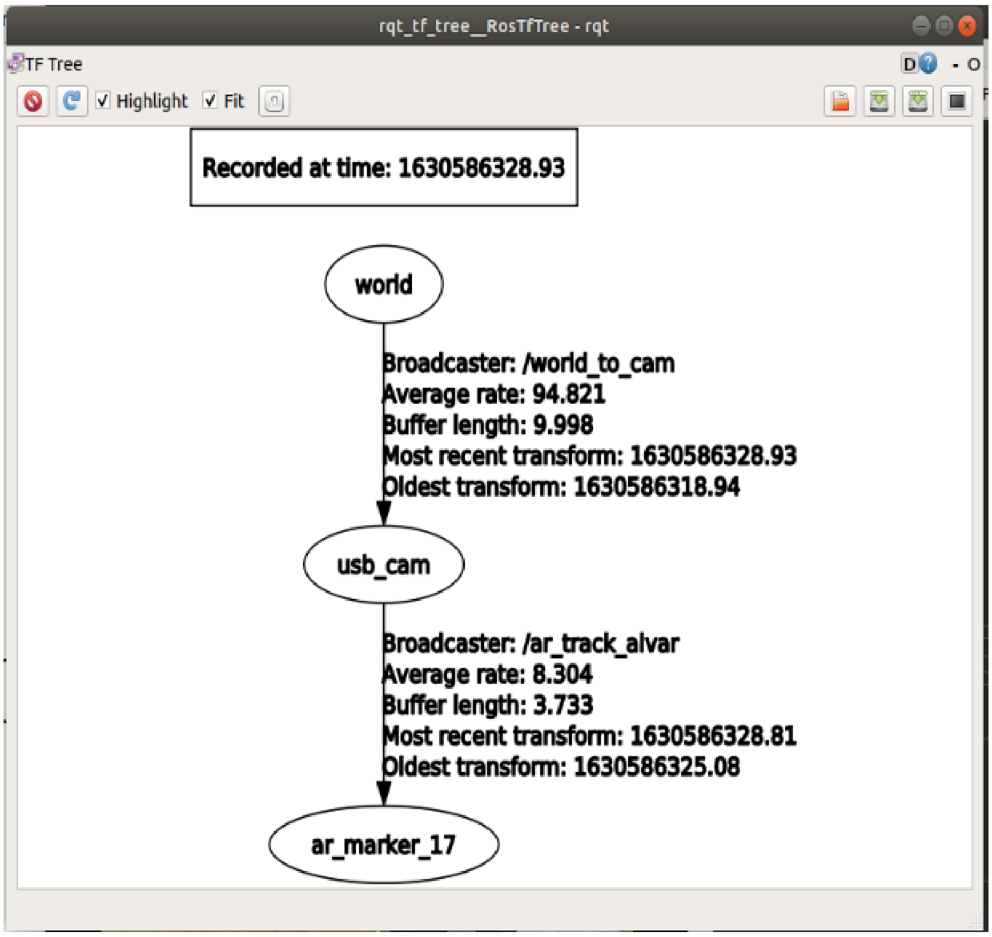
7.3.4. View node graph

```
rqt_graph
```



7.3.5. View tf tree

```
rosrun rqt_tf_tree rqt_tf_tree
```



Recorded at time: 1630586328.93

world

Broadcaster: /world_to_cam
Average rate: 94.821
Buffer length: 9.998
Most recent transform: 1630586328.93
Oldest transform: 1630586318.94

usb_cam

Broadcaster: /ar_track_alvar
Average rate: 8.304
Buffer length: 3.733
Most recent transform: 1630586328.81
Oldest transform: 1630586325.08

ar_marker_17

Through rviz, we can intuitively see the relative position of the QR code and the camera. The camera and world coordinate system are set by yourself.

7.3.6. View output information

```
rostopic echo /ar_pose_marker
```

The display is as follows:

```
    header:
      seq: 0
      stamp:
        secs: 1630584915
        nsecs: 196221070
      frame_id: "/usb_cam"
    id: 3
    confidence: 0
    pose:
      header:
        seq: 0
        stamp:
          secs: 0
          nsecs:         0
        frame_id: ''
      pose:
        position:
          x: 0.0249847882514
          y: 0.0290736736336
          z: 0.218054183012
        orientation: 
          x: 0.682039034537
          y: 0.681265739969
          z: -0.156112715404
          w: 0.215240718735
```

- frame_id：The coordinate system name of the camera
- id：The identified number is 3
- pose：pose of the QR code
- position：the position of the QR code coordinate system relative to the camera coordinate system
- orientation：the attitude of the QR code coordinate system relative to the camera coordinate system