8.3、ARTag

8.3.1、Overview

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

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

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 coding system and QR code are difference, it is mostly used in camera calibration, robot positioning, augmented reality (AR) and other applications.

One of the most important functions is to recognize the pose relationship between the object and the camera. ARTag can be affixed to the object, or an ARTag label can be affixed to the flat surface to calibrate the camera. After the camera recognizes ARTag, it can calculate the position and posture of the tag in camera coordinates.

ar_track_alvar has 4 function:

- Generate AR tags of different sizes, resolutions and data/ID encoding.
- Recognize and track the pose of a single AR tag, and optionally integrate kinect depth data (when kinect is available) for better pose estimation.
- Recognize and track the pose of a "bundle" composed of multiple tags. This allows for more stable pose estimation, robustness to occlusion, and tracking of multilateral objects.
- Use camera images to automatically calculate the spatial relationship between the tags in the bundle, so that users do not have to manually measure and enter the tag position in the XML file to use the bundle function.

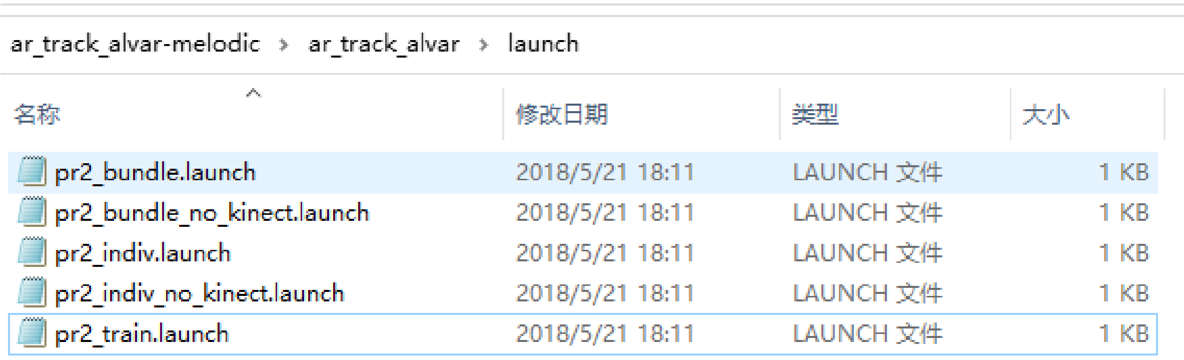
Alvar is newer and more advanced than ARToolkit. ARToolkit has always been the basis of several other ROS AR label packages. Alvar has adaptive threshold processing to handle various lighting conditions, optical flow-based tracking to achieve more stable pose estimation, and an improved tag recognition method that does not slow down significantly as the number of tags increases.

8.3.2、Create ARTag

1. Install software pack

Note: If you use jetbotmini official image, you can skip this step.

```
sudo apt install ros-melodic-ar-track-alvar
```



ar_track_alvar is an open source marker library that provides examples of pr2+kinect.

2. Create AR QR Code

- Continuously generate multiple labels on one picture

```
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: Specify the image size.

5: 5x5 picture.

33: The number is 33 QR code.

8.3.3、ARTag recognition

1. ar_track_alvar node

- **Subscribed topic**

Topic	Data type
/camera_info	(sensor_msgs/CameraInfo)
/image_raw	(sensor_msgs/Image)

- **Published Topics**

Topic	Data type
/visualization_marker	(visualization_msgs/Marker)
/ar_pose_marker	(ar_track_alvar/AlvarMarkers)

- **Provided tf Transforms**

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

Multiple QR codes: Provide the 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.

2. launch file parsing

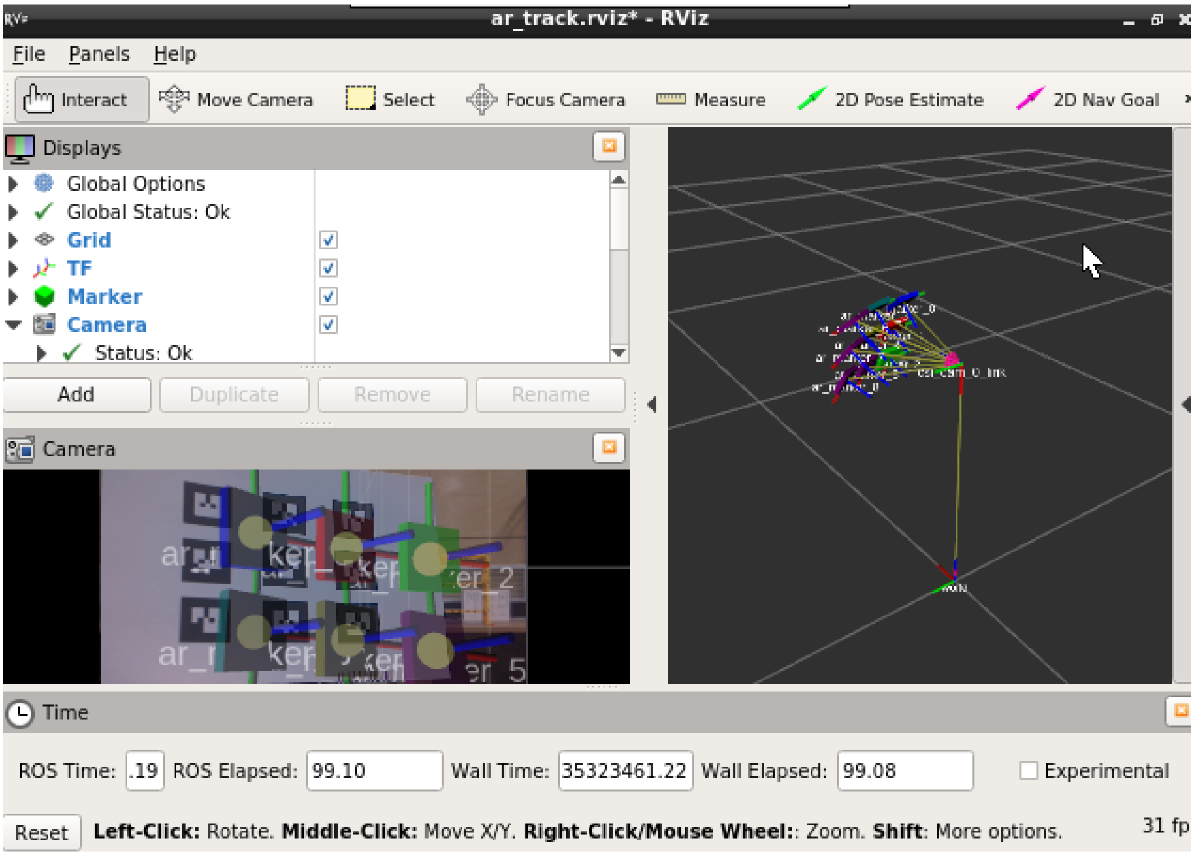
```
<launch>
    <arg name="open_rviz" default="true"/>
    <arg name="marker_size" default="5.0"/>
    <arg name="max_new_marker_error" default="0.08"/>
    <arg name="max_track_error" default="0.2"/>

    <include file="$(find jetson_nano_csi_cam)/launch/jetson_csi_cam.launch"/>
    <node  pkg="tf" type="static_transform_publisher" name="world_to_cam"
          args="0 0 0.5 0 1.57 0 world csi_cam_0_link 10"/>
    <node name="ar_track_alvar" pkg="ar_track_alvar"
type="individualMarkersNoKinect" respawn="false" output="screen">
        <param name="marker_size" type="double" value="$(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="/csi_cam_0_link"/>
        <remap from="camera_image" to="/csi_cam_0/image_raw"/>
        <remap from="camera_info" to="/csi_cam_0/camera_info"/>
    </node>
    <group if="$(arg open_rviz)">
        <node pkg="rviz" type="rviz" name="rviz" args="-d $(find
jetbot_ros)/launch/rviz/ar_track.rviz"/>
    </group>
</launch>
```

Node parameters:

- `marker_size` (double) -- The width in centimeters of one side of the black square marker border
- `max_new_marker_error` (double) -- A threshold determining when new markers can be detected under uncertainty
- `max_track_error` (double) -- A threshold determining how much tracking error can be observed before an tag is considered to have disappeared
- `camera_image` (string) -- The name of the topic that provides camera frames for detecting the AR tags. This can be mono or color, but should be an UNrectified image, since rectification takes place in this package
- `camera_info` (string) -- The name of the topic that provides the camera calibration parameters so that the image can be rectified
- `output_frame` (string) -- The name of the frame that the published Cartesian locations of the AR tags will be relative to

3. Start up camera to to recognize

```
roslaunch jetbot_ros ar_track.launch
```

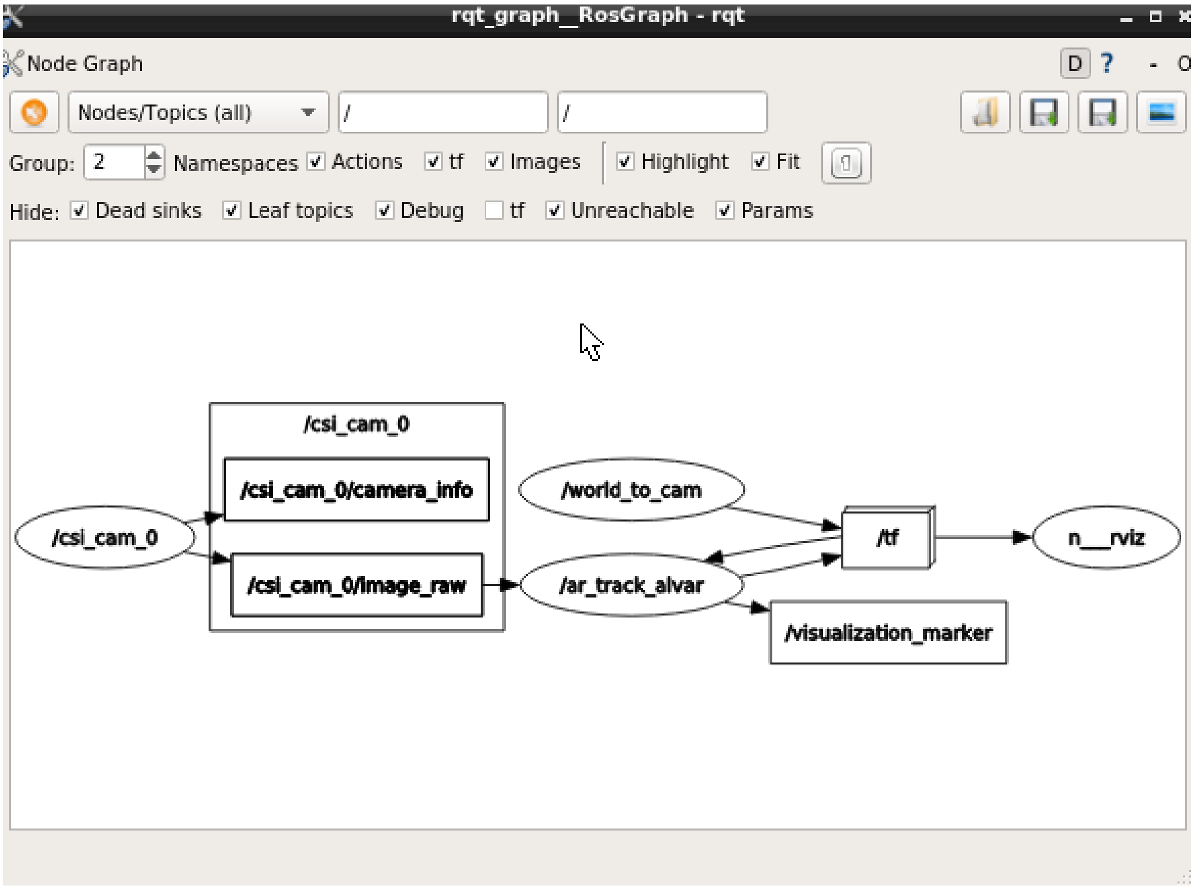


In rviz, you need to set the corresponding camera topic name.

- Image_Topic: Camera is【/csi_cam_0/image_raw】.
- Marker: The display component of rviz. Different squares show the location of the AR QR code.
- TF: The display component of rviz, which be used to display the coordinate system of the AR QR code.
- Camera: The display component of rviz, which displays the camera screen.
- world: World coordinate system.
- usb_cam/camera_link: Camera coordinate system.

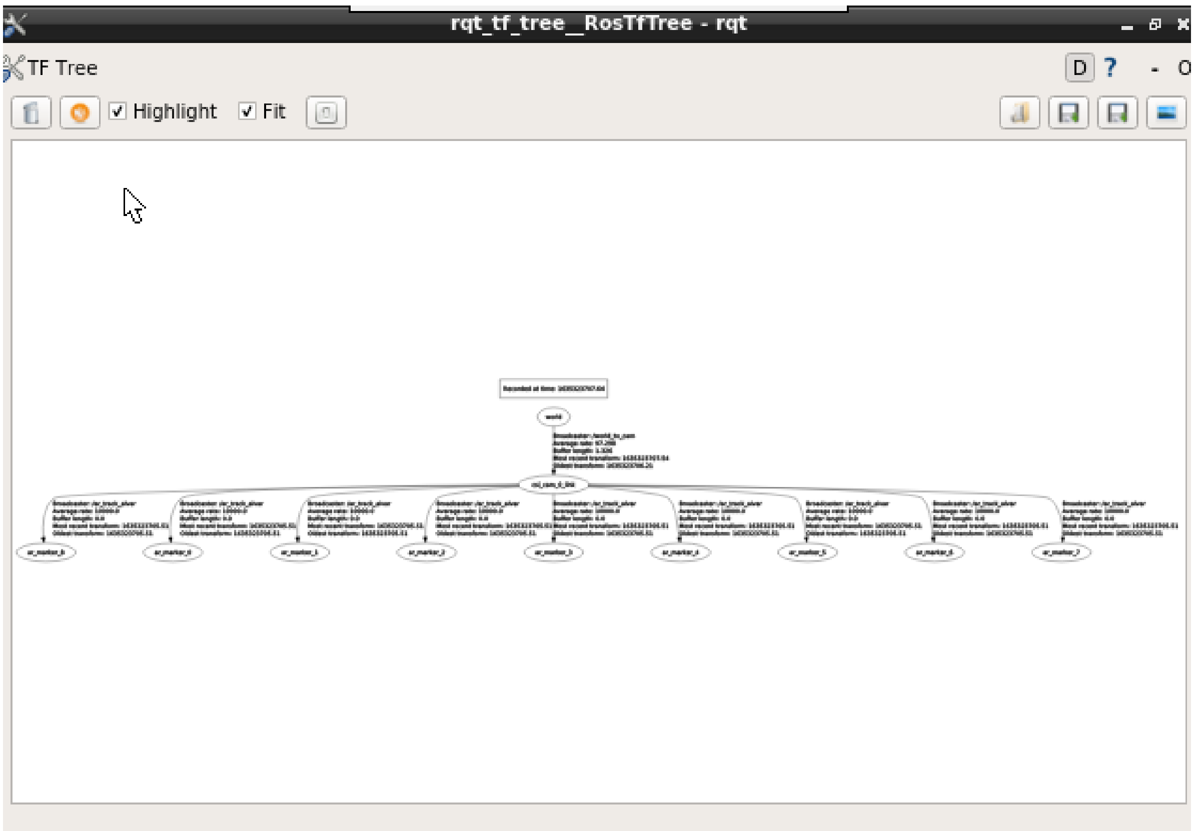
4. View node graph

```
rqt_graph
```



5. View tf tree

```
rosrun rqt_tf_tree rqt_tf_tree
```



Through rviz, we can see the relative position of the QR code and the camera very intuitively.

6. Viewing output information

```
rostopic echo /ar_pose_marker
```

Shown as follows:

```
    header:
      seq: 0
      stamp:
        secs: 1635323907
        nsecs: 351972489
      frame_id: "/jetson_csi_cam"
    id: 8
    confidence: 0
    pose:
      header:
        seq: 0
        stamp:
          secs: 0
          nsecs:         0
        frame_id: ''
      pose:
        position:
          x: 3.25242099926e-312
          y: 9.05198983881e-313
          z: 6.87688519034e-313
        orientation:
          x: 3.05175975132e-312
          y: -2.63006491287e-312
          z: 5.93903099628e-312
          w: 1.0
```

- frame_id：Name of the camera's coordinate system
- id：The recognized 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 orientation of the QR code coordinate system relative to the camera coordinate system

code path: /home/jetson/workspace/catkin_ws/src/jetbot_ros/launch/ar_track.launch