7.AR QR code tracking

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

7.1 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 pose 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 pose of the tag in the camera coordinates.

ar_track_alvar has 4 main functions:

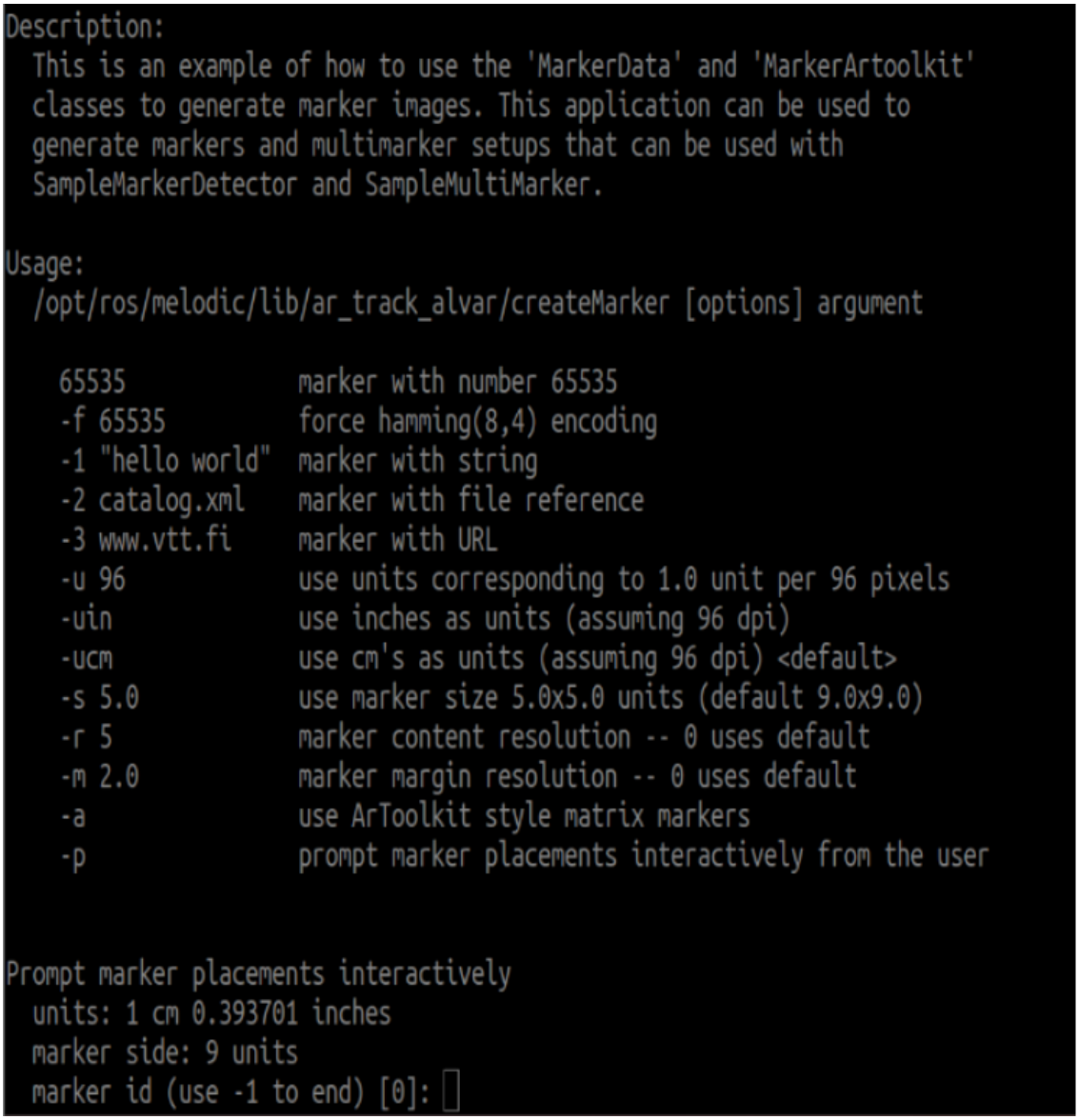
- Generate AR tags of different sizes, resolutions and data/ID encodings.
- Identify and track the pose of a single AR tag, and optionally integrate kinect depth data (when kinect is available) for better pose estimation.
- Use camera images to automatically calculate the spatial relationship between tags in a bundle so that users don't have to manually measure and enter tag positions 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 has adaptive thresholding to handle various lighting conditions, optical flow-based tracking for more stable pose estimation, and an improved tag recognition method that does not slow down significantly as the number of tags increases.

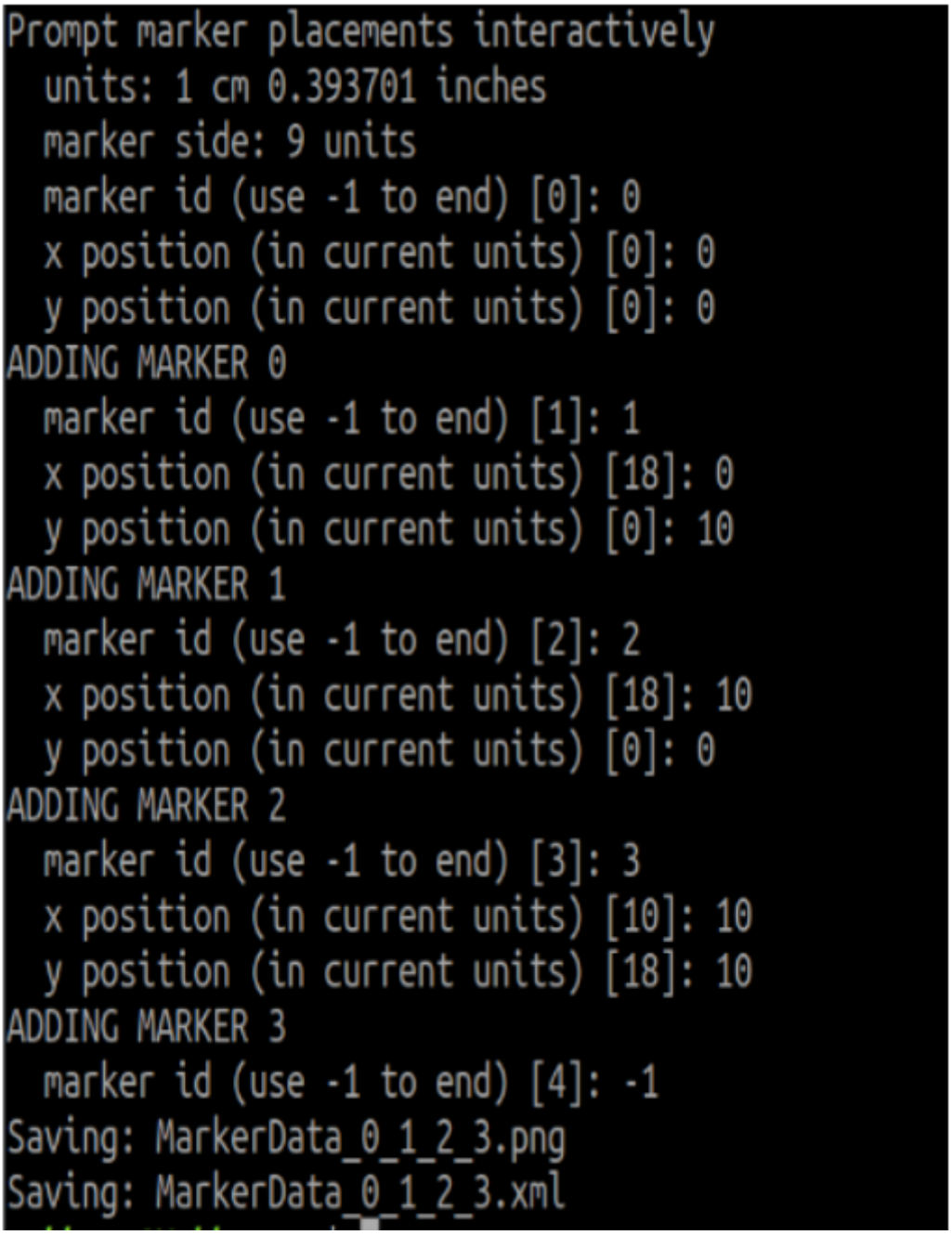
7.2. Create ARTag

- Generate multiple tags continuously 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
```

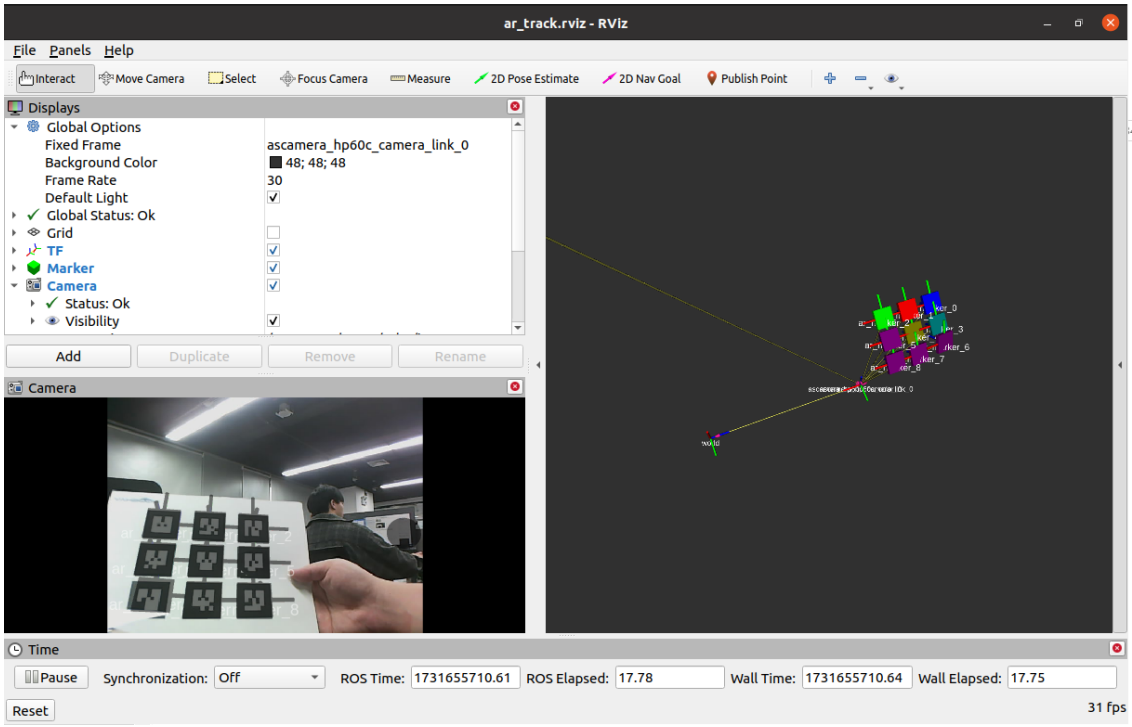
11: The number is the QR code of 11. -s: Specify the image size. 5: 5x5 image. 33: The number is the QR code of 33.

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

7.3, ARTag recognition and tracking

7.3.1, Start ARTag recognition and tracking

```
roslaunch ascam_visual ar_track.launch
```



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

- Image_Topic: Obbec's camera is [/ascamera_hp60c/rgb0/image].
- Marker: rviz display component, different blocks show the location of the AR QR code.
- TF: rviz display component, used to display the coordinate system of the AR QR code.
- Camera: rviz display component, displaying the camera screen.
- world: world coordinate system.
- camera_link: camera coordinate system.

launch file parsing,

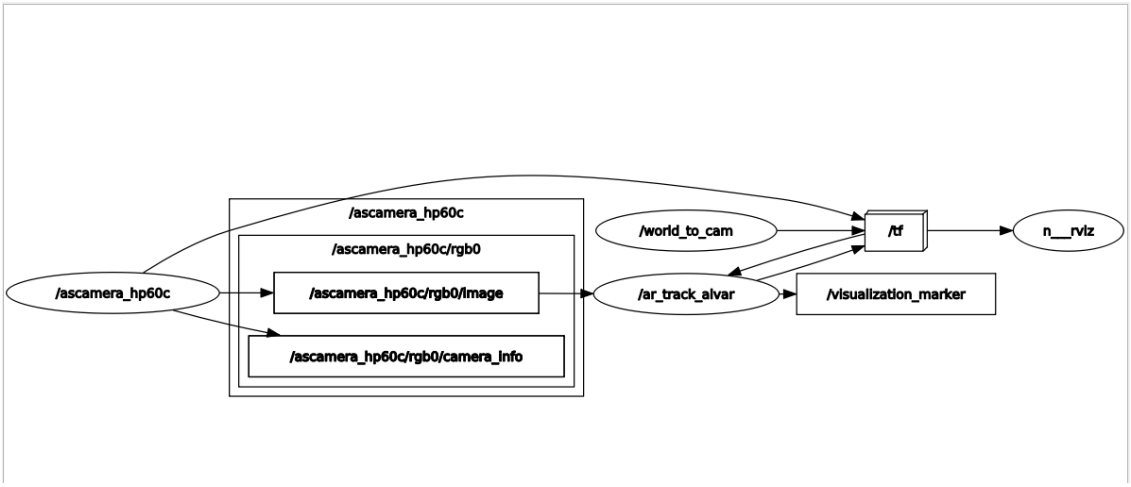
```
<launch>
    <arg name="open_rviz" default="true"/>
    <arg name="marker_size" default="5.0"/>
    <arg name="max_new_marker_error" default="0.08"/>
    <arg name="max_track_error" default="0.2"/>

    <arg name="cam_image_topic" default="/ascamera_hp60c/rgb0/image"/>
    <arg name="cam_info_topic" default="/ascamera_hp60c/rgb0/camera_info"/>
    <arg name="output_frame" default="/ascamera_hp60c_camera_link_0"/>

    <include file="$(find ascamera)/launch/hp60c.launch"/>
    <node pkg="tf" type="static_transform_publisher" name="world_to_cam" args="0
0 0.5 0 1.57 0 world ascamera_hp60c_camera_link_0 10"/>
    <node name="ar_track_alvar" pkg="ar_track_alvar"
type="individualMarkersNoKinect" respawn="false" output="screen">
        <param name="marker_size" type="double" value="$(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>
```

```
    <node pkg="rviz" type="rviz" name="rviz" args="-d $(find
ascam_visual)/rviz/ar_track.rviz" if="$(arg open_rviz)"/>
</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) : The topic name of the image used to detect the AR tag. This can be monochrome or color, but should be an unrectified image as rectification is performed in this package.
  - camera_info (string) : The topic name of the camera calibration parameters provided in order to rectify the image.
  - output_frame (string) : The coordinate position of the published AR tag in the camera coordinate system.

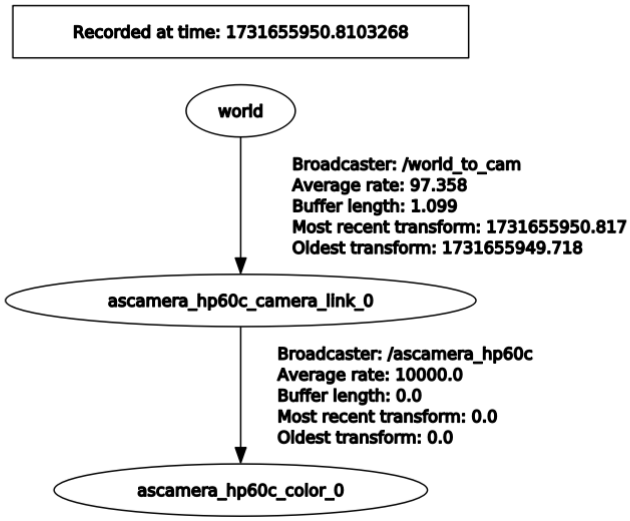
7.3.2, View the node graph

```
rqt_graph
```



7.3.4, View the TF tree

```
rosrun rqt_tf_tree rqt_tf_tree
```



7.3.3, View the output topic information

```
rostopic echo /ar_pose_marker
```

Shown as follows:

```
header:
  seq: 0
  stamp:
    secs: 1731656044
    nsecs: 675488770
  frame_id: "/ascamera_hp60c_camera_link_0"
id: 0
confidence: 0
pose:
  header:
    seq: 0
    stamp:
      secs: 0
      nsecs:         0
    frame_id: ''
  pose:
    position:
      x: -0.1652035738385046
      y: -0.002313690675123073
      z: 0.4325587570373727
    orientation:
      x: 0.9784503667767661
      y: 0.04034177085118151
      z: -0.04684219857707932
      w: 0.19701073501580588
```

- frame_id: camera coordinate system name
- id: the recognized number is 0
- 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