2. AR QR code

2.1. 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 fiducial marker system, which can be understood as a reference for other objects. It looks similar to a two-dimensional code, but its encoding system and two-dimensional code are still very large. The difference is mostly used in camera calibration, robot positioning, augmented reality (AR) and other applications. One of the most important functions is to identify the pose relationship between the object and the camera. An ARTag can be attached to the object, or an ARTag label can be attached to a plane to calibrate the camera. After the camera recognizes the ARTag, the position and pose of the tag in the camera coordinates can be calculated.

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 "clusters" consisting of multiple labels. This allows for more stable pose estimation, robustness to occlusion, and tracking of polygonal objects.
- The spatial relationship between tags in a bundle is automatically calculated using camera images so that users do not 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 tagging packages. Alvar features adaptive thresholding to handle various lighting conditions, optical flow-based tracking for more stable pose estimation, and an improved label recognition method that does not slow down significantly as the number of labels increases.

2.2. Create ARTag

2.2.1. install software package

melodic:

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

noetic:

The noetic system does not have a command to install ar-track-alvar, only from code source

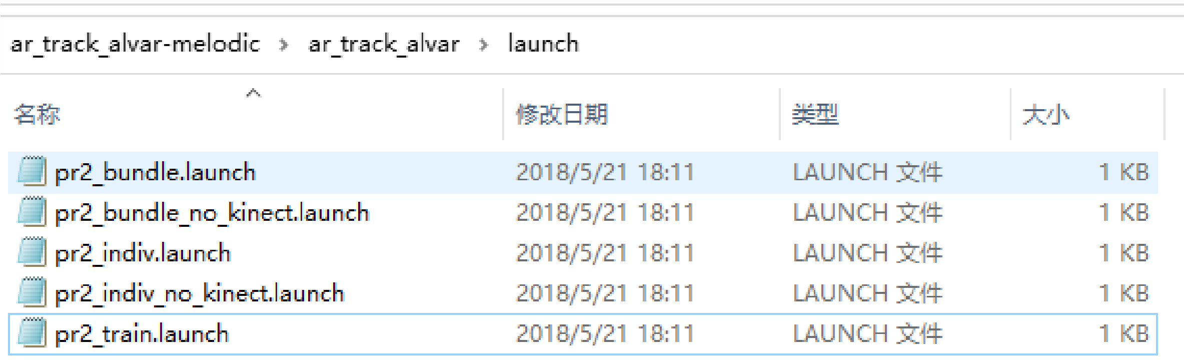
```
sudo apt install python3-colcon-common-extensions
```

```
# in the src folder of your workspace

git clone https://github.com/machinekoder/ar_track_alvar.git -b noetic-devel

# Then recompile the workspace

catkin build
```



ar_track_alvar is an open source marker library that provides examples of pr2+kinect. The first use case of this package is to recognize and track the pose of (potentially) multiple AR tags, each of which is considered individually.

2.2.2. Create AR QR code

- Generate multiple labels in a row on an image

```
rosrun ar_track_alvar createMarker
```

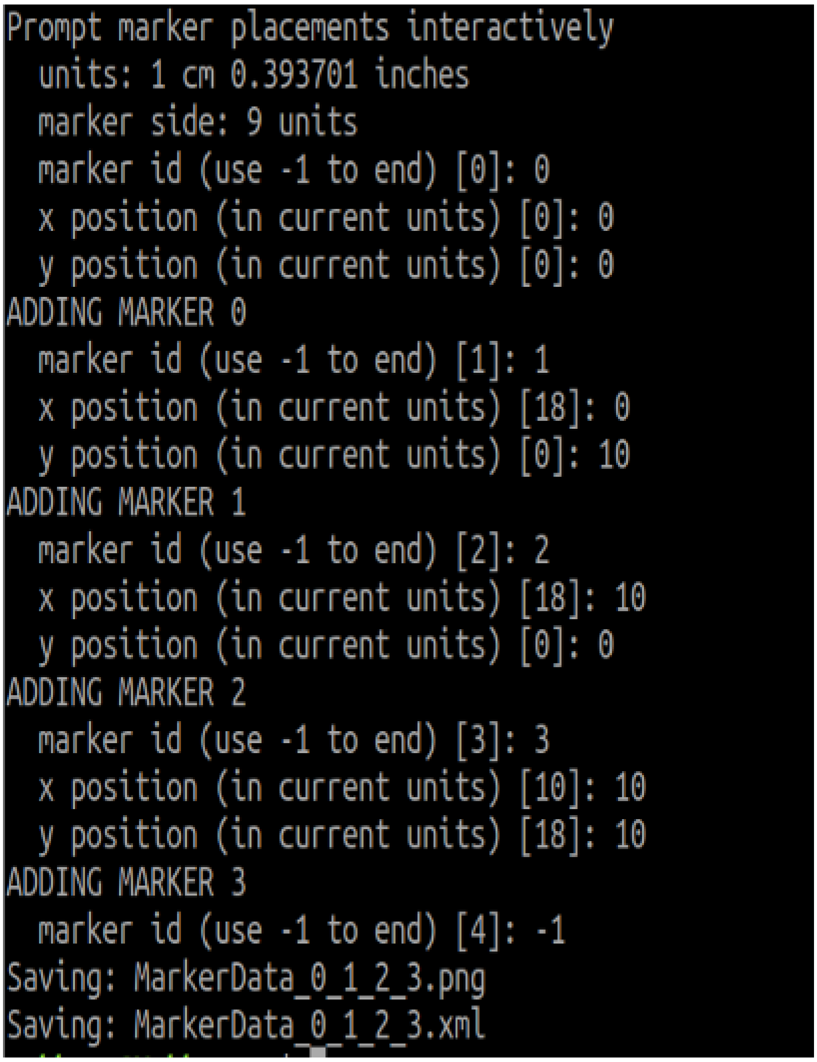
```
Description:
  This is an example of how to use the 'MarkerData' and 'MarkerArtoolkit'
  classes to generate marker images. This application can be used to
  generate markers and multimarker setups that can be used with
  SampleMarkerDetector and SampleMultiMarker.

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; e.g.

```
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.

2.3, ARTag identification

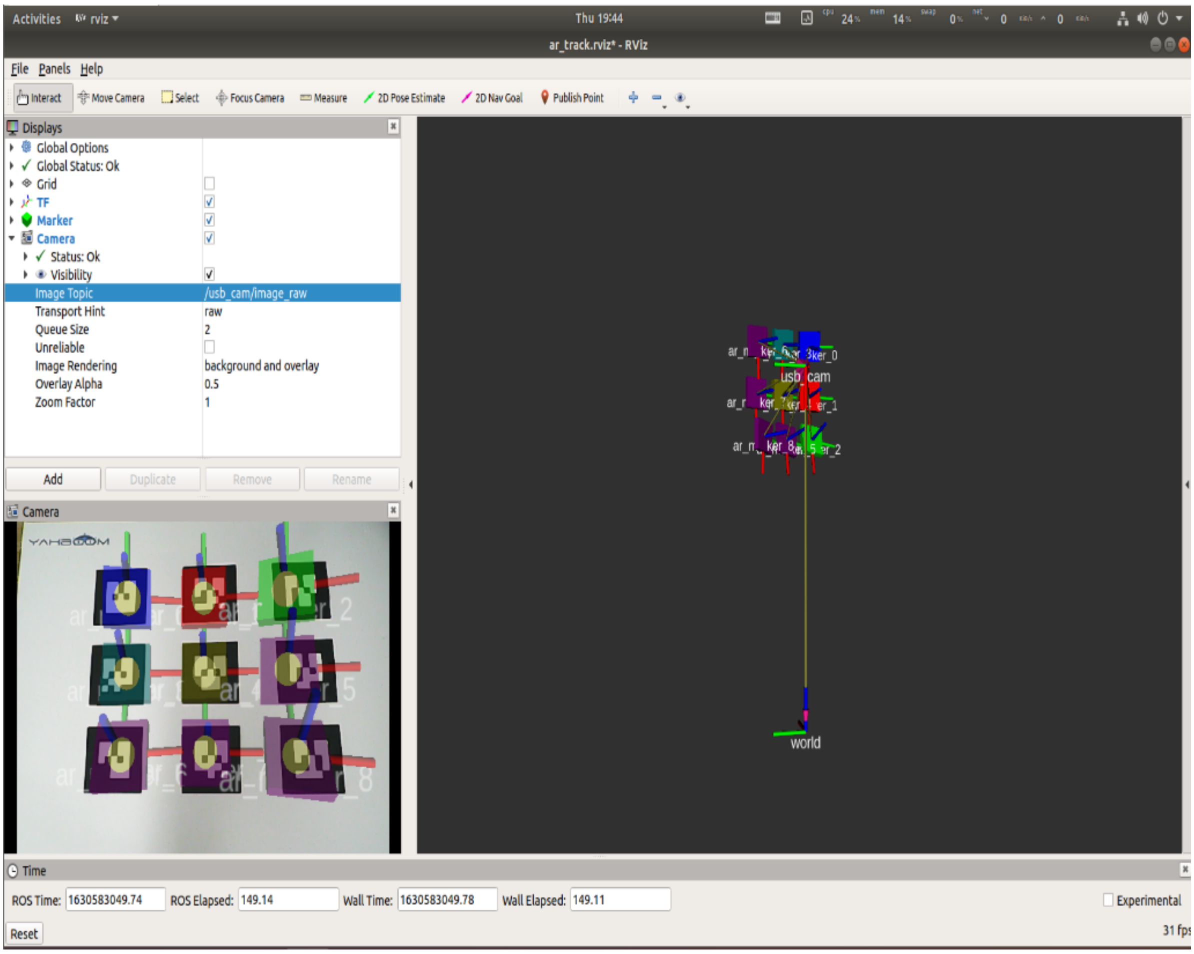
Note: When starting the camera, you need to load the camera calibration file, otherwise it will not be recognized.

2.3.1. Start the identification instance

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

- The open_rviz parameter is turned on by default.
- If it is a monocular camera or Raspberry PI CSI camera, you need to change 【ar_track_usb.launch】 to 【ar_track.launch】

  If it is a Jetson CSI camera, you need to change 【ar_track_csi.launch】 to 【ar_track.launch】



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

- Image_Topic: If it's a monocular camera or Raspberry PI camera, check [/usb_cam/image_raw]. If it's a Jetson CSI camera, check [/csi_cam_0/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 code.
- Camera: The display component of rviz, which displays the camera screen.
- world: world coordinate system.
- usb_cam: Camera coordinate system.

2.3.2. launch file analysis (Take monocular camera/Raspberry PI CSI camera as an example)

```
< 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 correspondence 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) : The width (in centimeters) of one side of the border of the black square marker.
- max_new_marker_error (double): Threshold to determine when a new marker can be detected under uncertainty.
- max_track_error (double) : A threshold that determines how many tracking errors can be observed before markers disappear.
- 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.

2.3.3. 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)

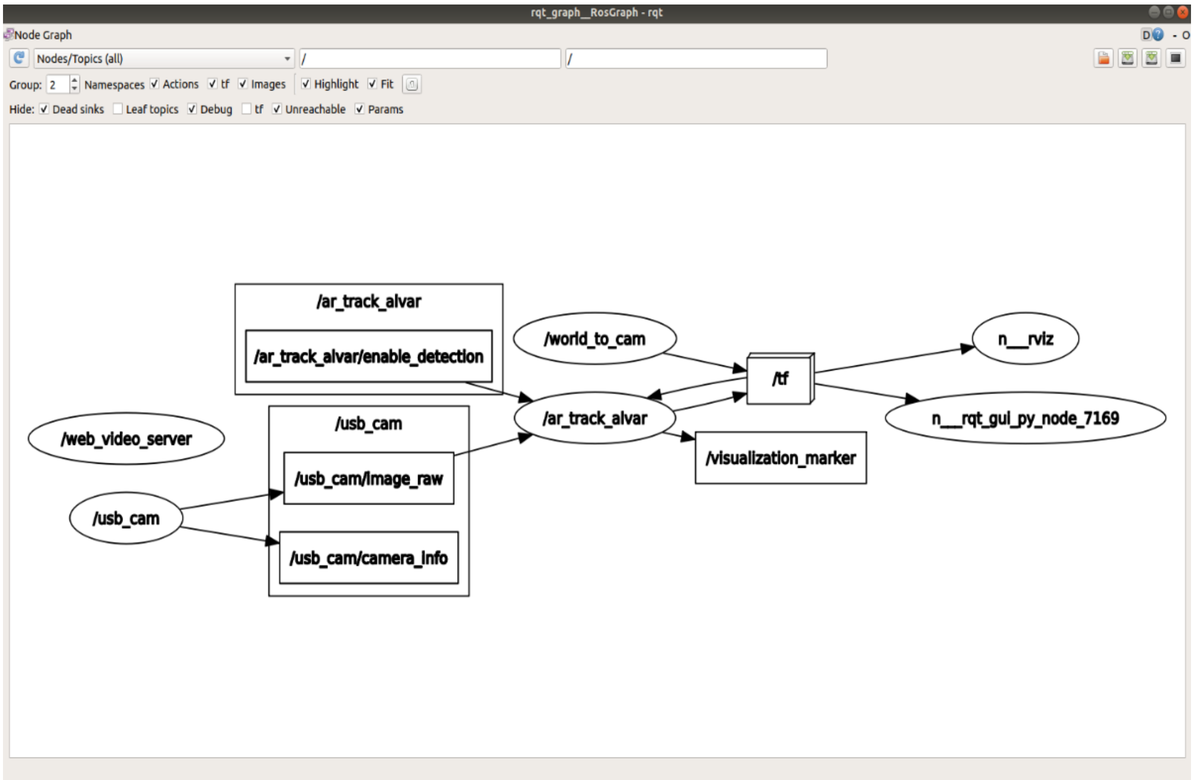
- ## Provided tf Transforms

A single QR code: camera coordinate system → AR tag coordinate system

Multiple QR Codes: Provides a transformation from the camera coordinate system to each AR marker coordinate system (named ar_marker_x), where x is the marker's ID number.

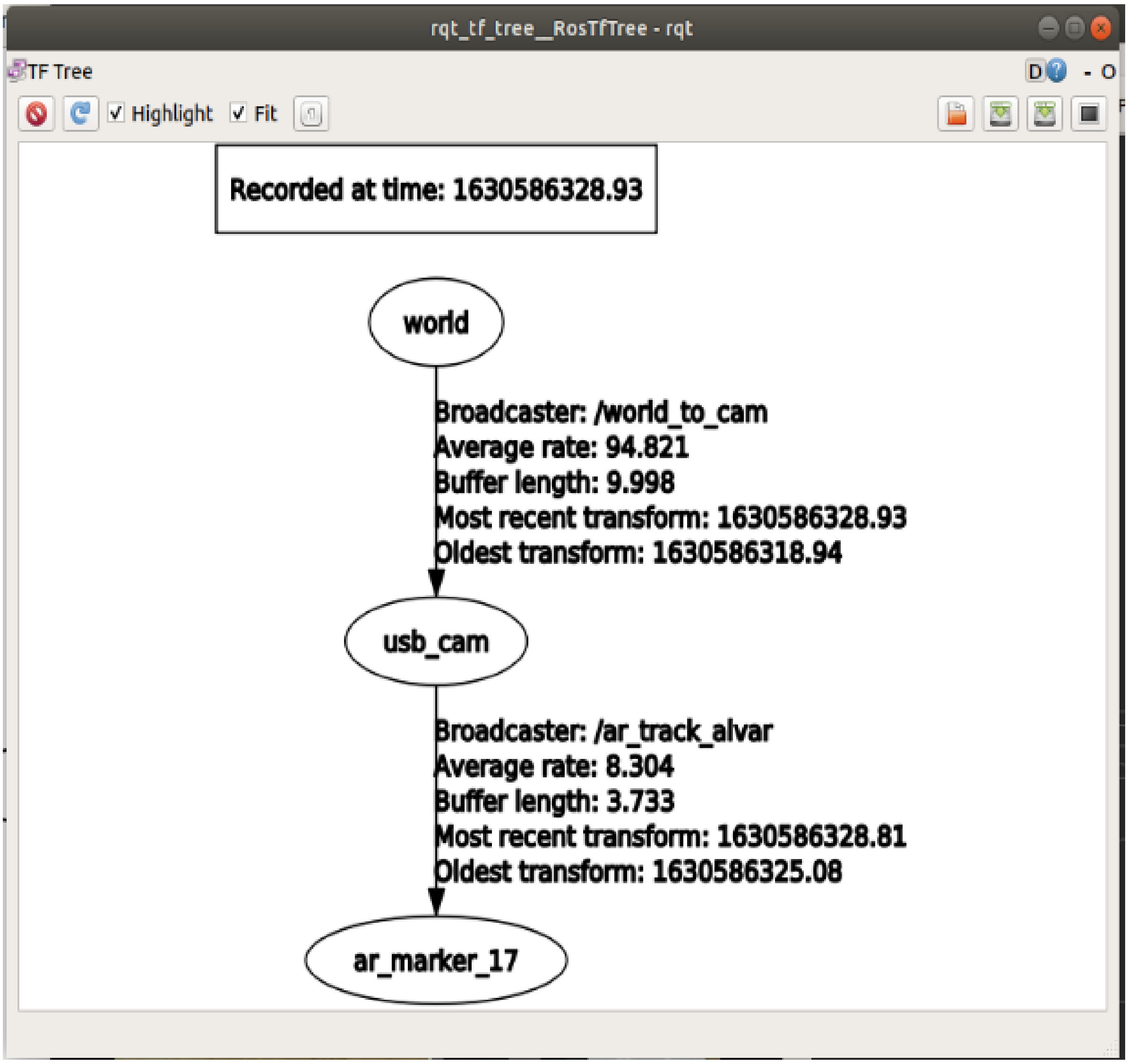
2.3.4. View node graph

```
rqt_graph
```



2.3.5. View tf tree

```
rosrun rqt_tf_tree rqt_tf_tree
```



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 themselves.

2.3.6. 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
with: 0.218054183012
        orientation:
          x: 0.682039034537
y: 0.681265739969
with: -0.156112715404
in: 0.215240718735
```

- frame_id: the coordinate system name of the camera
- 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