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

Function package location: /home/yahboom/YBAMR-COBOT-EDU-00001/src/yahboom_navrobo_other/yahboom_navrobo_visual

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. ARTags can be attached to objects or tagged on flat surfaces to calibrate the camera. Once the camera recognizes the ARTag, it can calculate the tag's position and pose in camera coordinates.

ar_track_alvar has 4 main functions:

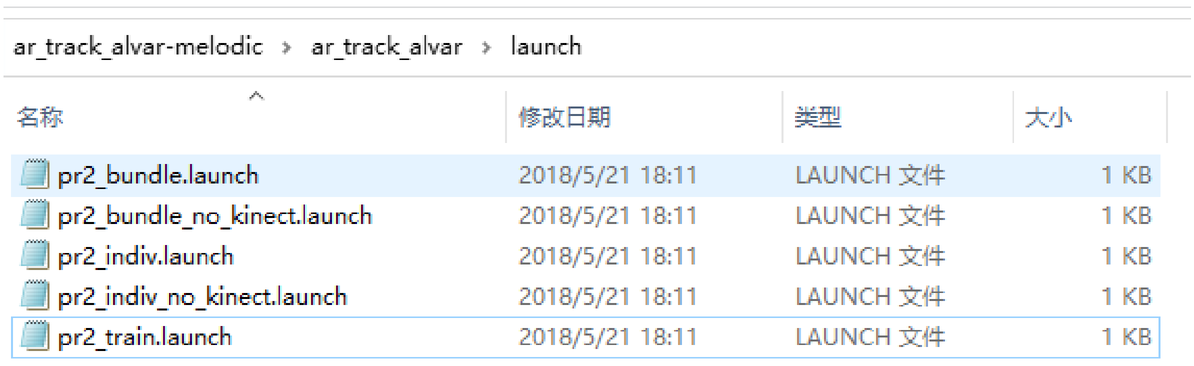
- Generates AR tags of different sizes, resolutions, and data/ID encodings.
- Identifies and tracks the pose of a single AR tag, optionally integrating kinect depth data (when kinect is available) for better pose estimation.
- Identifies and tracks the pose of a "bundle" of multiple tags. This allows for more stable pose estimation, robustness to occlusion, and tracking of multi-sided objects.
- Automatically computes the spatial relationship 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 recognition method that does not slow down significantly as the number of tags increases.

7.2. Create ARTag

7.2.1. Install the package

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



ar_track_alvar-melodic › ar_track_alvar › launch

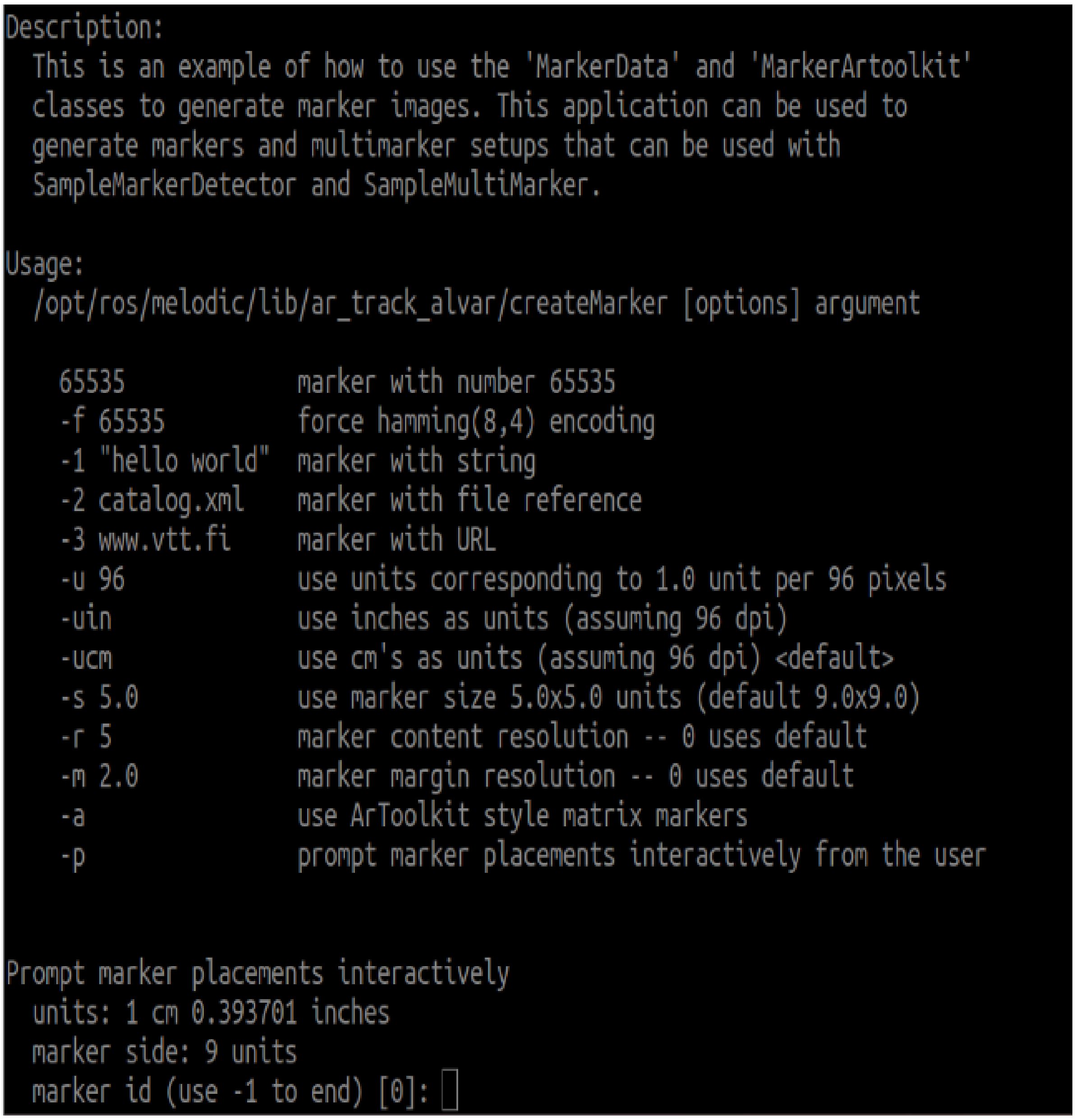
名称	修改日期	类型	大小
pr2_bundle.launch	2018/5/21 18:11	LAUNCH 文件	1 KB
pr2_bundle_no_kinect.launch	2018/5/21 18:11	LAUNCH 文件	1 KB
pr2_indiv.launch	2018/5/21 18:11	LAUNCH 文件	1 KB
pr2_indiv_no_kinect.launch	2018/5/21 18:11	LAUNCH 文件	1 KB
pr2_train.launch	2018/5/21 18:11	LAUNCH 文件	1 KB

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

7.2.2, Create AR QR code

- Generate multiple labels on one image

```
rosrun ar_track_alvar createMarker
```



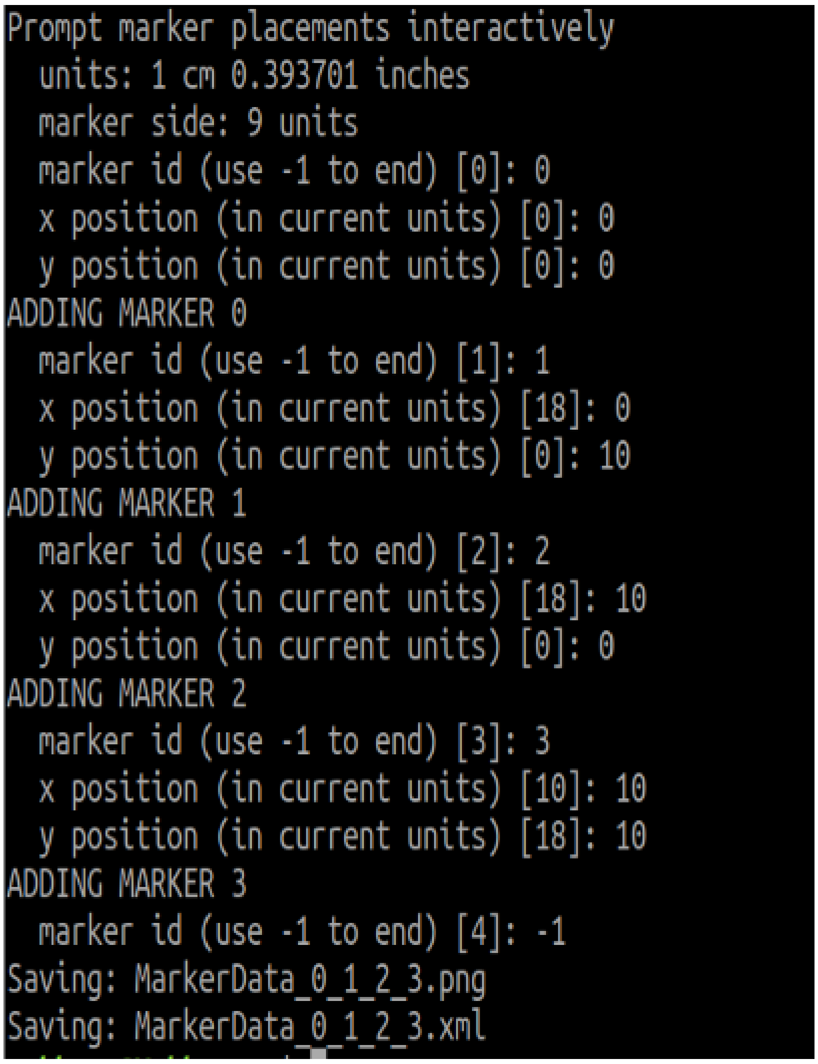
```
Description:
  This is an example of how to use the 'MarkerData' and 'MarkerArtoolkit'
  classes to generate marker images. This application can be used to
  generate markers and multimarker setups that can be used with
  SampleMarkerDetector and SampleMultiMarker.

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. You can generate one or more, and design the layout 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 + parameter to directly generate 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 with the number 33.

7.3, ARTag recognition

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

7.3.1, Start the recognition instance

Please do this step before running the program

```
sudo supervisorctl stop ChassisServer #Shut down the self-starting chassis
service
```

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

- The open_rviz parameter is opened by default.

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 blocks show the location of the AR QR code.
- TF: The display component of rviz, used to display the coordinate system of the AR QR code.
- Camera: The display component of rviz, showing the camera screen.
- world: world coordinate system.
- usb_cam: camera coordinate system.

7.3.2, launch file parsing

```
<launch>
    <!-- Set the camDevice parameter, 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 the camera image topic, camera intrinsic parameter topic, and 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 the camera node -->
    <include file="$(find usb_cam)/launch/usb_cam-test.launch"/>
    <!-- Set the correspondence between the camera coordinate system and the
world coordinate system -->
    <node pkg="tf" type="static_transform_publisher" name="world_to_cam" args="0
0 0.5 0 1.57 0 world usb_cam 10"/>
    <!-- Start the AR recognition node -->
    <node name="ar_track_alvar" pkg="ar_track_alvar"
type="individualMarkersNoKinect" respawn="false" output="screen">
        <param name="marker_size" type="double" value="$(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 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.

7.3.3, ar_track_alvar node

- **Subscribed topic**

Topic Name	Data Type
/camera_info	(sensor_msgs/CameraInfo)
/image_raw	(sensor_msgs/Image)

- **Published Topics**

Topic Name	Data Type
/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 a 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 the node graph

```
rqt_graph
```

7.3.5, 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 systems are set by ourselves.

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 name of the camera's coordinate system
- id: the number recognized is 3
- pose: the position 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