Problem Statement

Detecting WhyCon markers

From the tutorials, you have learned

- Basics of ROS : creating workspace, creating package, launch file, writing a subscriber
- WhyCon markers : topics which the package subscribes and publishes

Problem Statement:

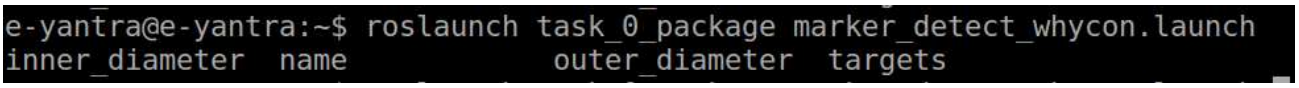
Fetch WhyCon marker coordinates of the static markers placed in the Gazebo world by subscribing to the markers' corresponding topics and print the coordinates.

Procedure:

1. Copy the sr_task_0 folder present in your Task download into the *catkin_ws/src* directory. We will be operating on this package folder for this task.
2. Please go through the procedure mentioned in Read_me.pdf before proceeding.
3. As this folder contains just .launch and .py files (files which are not compiled, there is no need to run the catkin_make command). You may still do so if you wish.
4. Take a look at the "*marker_detect_whycon.launch*" present inside the launch folder.
5. In it you will find that most of the *.launch* file requirements mentioned in *Introduction_to_WhyCon_markers.pdf* are already present.
6. The lines of remap (lines 32-33) are commented, **please uncomment and complete them**. Instructions can be found in the *Introduction_to_WhyCon_markers.pdf* file.
7. Launch roscore in a terminal.
8. Next launch Gazebo simulator by typing the following command (you need this only for the very first time, subsequently you do not need to perform this step, only step 9 will work)
   ```
   > gazebo
   ```
9. To see the world scene of Task0 in gazebo type the following command, which will launch your Gazebo world
   ```
   > roslaunch sr_task_0 task_0.launch
   ```

The above step might not work for the first, you must rerun until you see the Gazebo world with multiple whycon marker in it. If the marker do not appear, make sure you have performed instructions describe Read_me.pdf .

SIDE NOTE: If you press tab after you enter the above command ending with a space, you will see the various arguments that the .launch file accepts. An example for the whycon launch file is given in the screenshot below.



```
e-yantra@e-yantra:~$ roslaunch task_0_package marker_detect_whycon.launch
inner_diameter   name               outer_diameter   targets
```

Figure 1: WhyCon launch file options

These parameters control various properties of the WhyCon node. You can change these parameters if you wish, if they are not specified in the command line, they are assigned certain default values which you can find in the .launch file

10. Subsequently run the command, roslaunch task_0_package marker_detect_whycon.launch targets:=5
11. If you have remapped the topic correctly then you should see 5 whycon markers displayed in a window titled “whycon/image_out”.
12. Type rostopic list in a different terminal window to find the topics being published by the overhead camera overhead_cam.
13. The following topics get published on running the simulation:
    a) **/gazebo/overhead_cam/image_rect_color**
        i. This message is of type *sensor_msgs/Image*
        ii. This corresponds to the image published by the camera in the Gazebo world.
        iii. Run the following command on a terminal to see the image published by the overhead camera:

```
>  rosrun image_view image_view image:=/gazebo/overhead_cam/image_rect_color
```

    b) **/gazebo/overhead_cam/camera_info**
        i. This message is of type *sensor_msgs/CameraInfo*
        ii. This corresponds to the camera information being published by the camera in the Gazebo world.
14. The launch file must ideally run the simulator alongwith the required nodes in the whycon package to detect markers in the image published by the overhead camera in Gazebo and should output a window of detected WhyCon markers as shown in Figure 2.

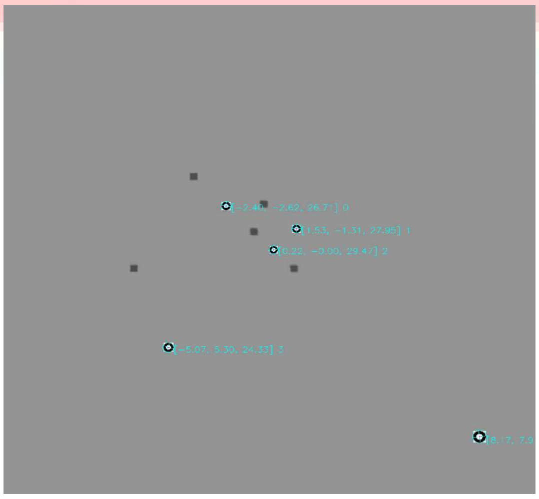
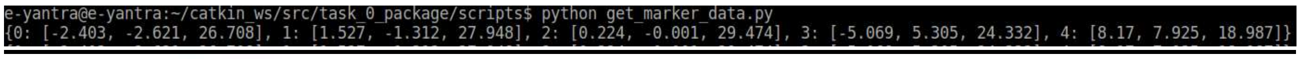


Figure 2 : WhyCon output image

15. Open the python script named "***get_marker_data.py***" in the "scripts" folder present in the folder, "task_0_package". This rosnode must subscribe to the ***/whycon/poses*** and output the **position(x,y,z)** of . You should create a dictionary for the WhyCon markers; in which the key field is the marker id and values field corresponds to a table containing position (x,y,z). For hints to complete the subscriber function you can visit the file Getting_familiar_wtht_ROS.pdf



```
e-yantra@e-yantra:~/catkin_ws/src/task_0_package/scripts$ python get_marker_data.py
{0: [-2.403, -2.621, 26.708], 1: [1.527, -1.312, 27.948], 2: [0.224, -0.001, 29.474], 3: [-5.069, 5.305, 24.332], 4: [8.17, 7.925, 18.987]}
```

Figure 3 : Terminal output image

Points to remember:

- You are not supposed to change any of the templates.
- Make sure you remap the topics properly to detect markers.
- Run "*rosmsg show topic_type*" to see what is the message type of the corresponding topic.
- Run "*rqt_graph*" in a separate terminal to check your topic and node relations. It should look something like in Figure 4.

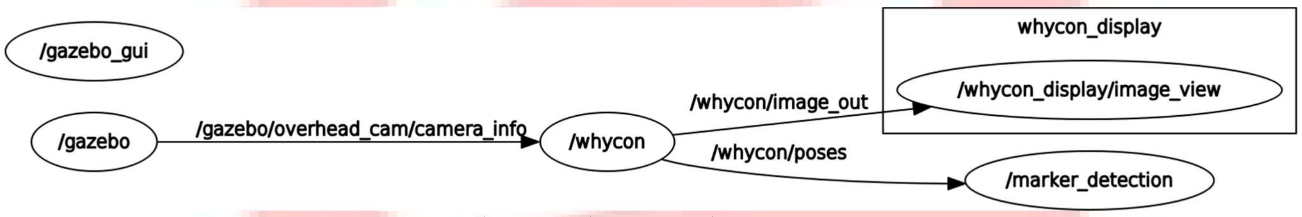


Figure 4: The rqt_graph

Submission Instructions:

Follow the instructions below to submit your Task.

1. Bag File:

a) Launch your *marker_detect.launch* file after running simulation in Gazebo:

```
> roslaunch marker_detect.launch
```

b) Run your python script, *get_marker_data.py* by running the following command in another terminal:

```
> rosrun get_marker_data.py
```

Run the rosbag command to record the topics for a specific duration. The following command records the topics */whycon/poses* and for 5 seconds and saves it with a name *markers.bag* in the directory from where you executed the command:

```
> rosbag record -O markers.bag --duration=5 /whycon/poses
```

c) Compress the *markers.bag*

```
> rosbag compress -j markers.bag
```

d) This will compress the *markers.bag* and now the size will be below 1-2MB. Look at the size of the bag file in its properties to distinguish between the original and the compressed bag file.

2. Code:

a) Do not rename your Python script, retain the original name.

Compress these three files into a .zip file before uploading. Name the .zip file as your <team_id>.zip. For eg: if your Team ID is SR#105, then rename it as 105.zip

NOTE: You must upload all of the following: (i) the compressed bag file (ii) screenshot of your PC screen containing both the terminal printing the marker coordinates from the python script and the images of detected WhyCon markers in .png format, titled: task_0_scrsht (iii) your completed python script in order to be evaluated. Please place all these files inside a .zip file before uploading. Name the .zip file as your <team_id>.zip. Please follow the naming convention strictly as specified in each step.

Instructions for uploading the folder will be provided on the portal.

Bonus Problem Statement:

We will implement the Babylonian method for finding square roots- an exercise which you may have performed in school- using two ROS nodes with publishers and subscribers. In this exercise we iteratively arrive at the square root of a number by applying a formula. More details of this algorithm can be found on this URL.

This exercise is intended mainly as an exercise to give you hands on exposure to writing Python scripts with ROS integration. This task will not be graded.

The structure of our nodes will look roughly like this, however there are multiple possibilities and you are free to experiment.

Node 1: Will be the target-containing Node. It will have the value of the number whose square root is to be calculated and a compare function to compare the value calculated by this algorithm to the output of Python's sqrt(NUMBER) function up to a certain precision (say 10).

Node 2: Will have a subscriber who given the value of the current iteration and the number whose square root is to be calculated will publish the updated value of the iterant.

The Algorithm will look roughly like this:

1. Input a value whose square root is to be calculated along with a seed/guess/estimate value to start the iteration.
2. Node 1 publishes these values.
3. Node 2's subscriber takes these values, computes an updated estimate and publishes the updated estimate.
4. Node 1 subscribes to these values, the subscriber compares this with a threshold as mentioned above and if not precise enough publishes the fresh estimate again.
5. Steps 3 and 4 repeat until desired precision is reached.
6. Display the output and number of iterations it took to reach the desired precision.

Choose appropriate names and data types for the topic. A sample solution will be released at the end of Task 0.