Overview

Environment setup

Test environment:

- Ubuntu 20.04
- ROS-noetic
- V-REP PRO EDU 3.62 Ubuntu 18.04
- OpenCV version 4.2.0

External pacakge:

- hector-mapping

Setup

1. run `roscore`
2. launch the simulation `cd <V-REP-dir>` , then `./vrep.sh <location of environment file>`, then start the simulation
3. `roslaunch final_3210 main.launch`. It will pop out 4 windows
    - rviz: for rviz visualization
    - teleop: for keyboard teleop control
    - location: print out location and judge the room label
    - Capture - Face detection: image get from the camera. It will mark the recognized face.

Design and Implement

1 keyboard control

by Jinyun

Copy from the `teleop_twist_keyboard.py` from package `teleop_twist_keyboard`. Modified a bit to switch on and off the visual servo. Press `a` to enable, `s` to stop.

2 build map with laser

by Jinyun & Xinyuan

3 judge location

by Jinyun

4 image recognition and localization

by Xinyuan

1. Detection

We use OpenCV Haar Cascade method for face detection. The data pre-trained is downloaded from here. We chose the file `haarcascade_frontalface_default.xml`.

The detection is based on gray-scale, and able to detect multiple faces, we choose only the largest one and resize it to a scale of 50 x 50. The gray-scale resized region of interest (ROI) is then passed to the recognition model, because the recognition model also work on only the gray-scale image.

2. Recognition

   We use OpenCV Eigenfaces method for face recognition.

   The Eigenfaces method use PCA to recognize different faces. We need to first prepare a dataset to train the model. Before the first time we run the recognition, we collected 16 detection results for each image and save it in `imgs` dir. These results are all 50 x 50 gray-scale images output by the detection model.

   When the program start, the Eigenfaces model will load all the training dataset and train itself. Then we can directly use it to recognize the label of incoming images.

3. Localization

   Let the center of the ROI be $P_c = (x_c, y_c)$. As the size of image is 512 x 512, the camera has Perspective angle $\frac{\pi}{4}$, the relative angle to the left-up corner is

   $\theta_x = \frac{\pi}{4} * \frac{x_c}{512} - \frac{\pi}{8}$

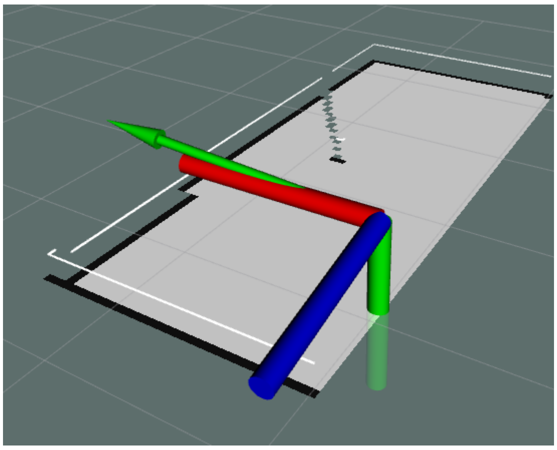
   $\theta_y = \frac{\pi}{4} * \frac{y_c}{512} - \frac{\pi}{8}$

   where positive means from left to right and up to down.

   With θ_x, we can get the ground distance between the robot and the image by looking up the corresponding laser data. Let the distance be d. the length form the camera to P_c is

   $l = \frac{d}{cos(\theta_y)}$

   In a coordinate where X is front, Y is left, and Z is up, The rotation q to move the P_c to the center in Row-Pitch-Yaw is $(0, \theta_y, -\theta_x)$. However, the coordinate of `camera_link` is not the case. The X(red) axis is correct, but the other two axises are rotated -100 degree relative to X axis. Let this rotation be q_n, Row-Pitch-Yaw form $(\frac{-100\pi}{180}, 0, 0)$. So the final rotation for `camera_link` is $q_n^{-1} * q * q_n$. The green arrow stands for the X-axis of the new frame, where P_c is located at $(l, 0, 0)$



5 visual servo

by Jinyun

6 launch file

by Xinyuan

`main.launch` includes 3 files, `teleop_and_tracking.launch`, `slam.launch`, and `face.launch`.

`teleop_and_tracking.launch` launch `teleop_twist_keyboard.py` and the visual servo node `track`

`slam.launch` publish a `static_tf_transform` from `odom` to `base_link` and launch `hector-mapping`. `hector-mapping` will publish the transform from `map` to `odom`

`face.launch` specify the dataset location and launch the node `face`.

Conclusion

References