Hong Kong University of Science and Technology
ELEC6910R Robotic Perception and Learning
Spring 2018

ELEC6910R Project Report

Name: Yetao LYU (20486418)
Yu LEI (20486418)
Chung Hee KIM(20080862)

Date: 05/31/2018

Subject Instructor: Prof. LIU Ming

TA: HUANG Kan

Overview

In this project, a mobile car is available to be controlled in the simulation environment via keyboard or visual servoing to handle tasks including SLAM, face detection and face recognition. A demo video has been submitted to TA to present the whole procedure from initialization to the runtime. Basic tasks and bonus which have been implemented in this project are listed as below:
All basic requirements:

1. Build 2D grid map with laserscan data and show it via rviz
2. Control the mobile robot in the simulation environment with keyboard
3. Image recognition and localization.
4. Visual servoinging
5. Current room number recognition
6. A top-level launch file

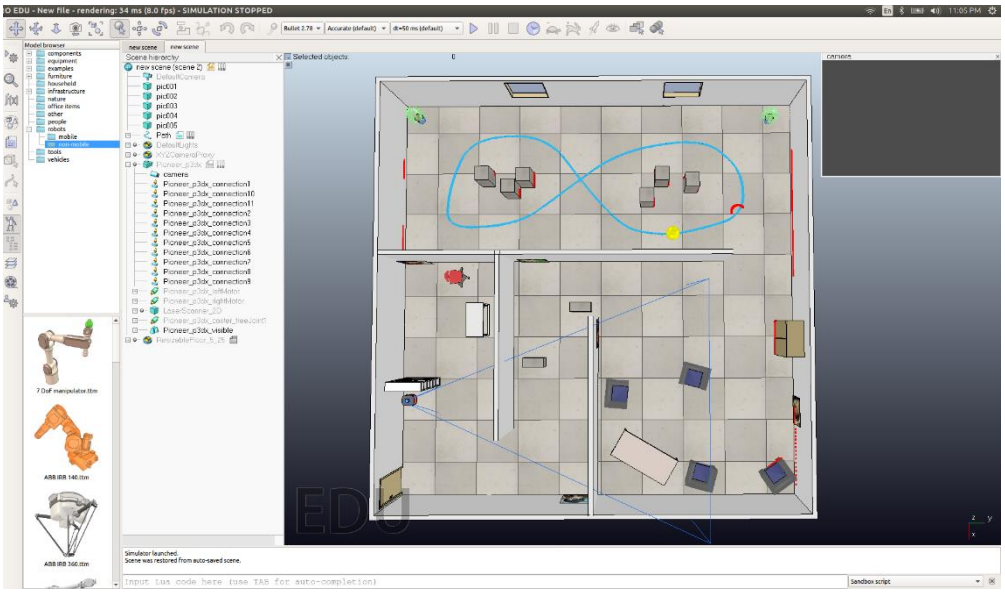
Bonus:

1. Face recognition
2. Controller changes itself to tracking mode automatically
3. Meanshift implementation

The whole project is divided into three major parts: the first part is the SLAM task which is implemented by Chung Hee KIM, the second part is the face detection and recognition programmed by Yetao LYU, and the last part is the visual servoing. We have two versions for the last part, the first version is implemented by Chung Hee KIM with the ability to switch mode automatically and the second version is realized by Yu LEI including a meanshift algorithm and standard servoing process.

Environment setup

The simulation environment used for this project is V-REP, which is a robot simulator. It is based on distributed control architecture where each object can be individually controlled through various methods including plugin, embedded script, remote API clients, or a ROS node. Powerful features are included in the simulator such as physics engines (Bullet Physics, ODE, Newton, Vortex Dynamics), collision detections, forward/inverse kinematics to list a few. In this project, an environment was given as env.ttt file as shown below.



The robot was controlled by interfacing V-REP with ROS through the vrep_ros_bridge and RosInterface plugins. The main application of the vrep_ros_brige plugin is to provide communication interface between V-REP and ROS such that the simulation can be externally controlled using ROS messages and services.

RVIZ (ROS visualization) is a 3D visualizer for displaying various sensor data and state information from ROS. In this project, we utilized RVIZ to display the map and location of the portrait images on the map.

To realize SLAM, various packages are available for use. For the purpose of this project, the group used hector_slam. Hector_slam was developed by Kohlbrecher et al. [1] which uses map gradients for pose estimation along with high update rate of laser scanners. Further detail will be elaborated in later section.

OpenCV (Open Source Computer Vision Library) was utilized for robotic perception including visual servoing and image recognition. OpenCV is an open source computer vision and machine learning software library. The library includes comprehensive set of classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used for various purposes including recognizing faces, identifying objects, tracking movements etc.

To control the robot with the keyboard, a simple package available for use called teleop_twist_keyboard was used. The package is a generic keyboard teleop for twist robots which was perfect for the project application.

System design and modules implement

a) SLAM

Simultaneous Localization and Mapping (SLAM) is vital to mobile robots. To interact with the environment, the robot must know what is around itself (Mapping) and know where it is in the environment (Localization). Various algorithms exist to achieve SLAM, including Kalman Filters, Particle Filters etc. but fundamentally they are all Bayesian Filter Processes. In the Robot Operating System (ROS), several packages are available to achieve SLAM. Our group narrowed our options to two candidates: Gmapping and Hector slam. Initially, Gmapping was attempted but Gmapping requires odometry which is combined with the laser scan due to the slow update rate. Since the odometry was not given in the pre-set environment of V-REP, we decided to use Hector Slam, which only requires the laser scan without the odometry to achieve SLAM in 2D environments.

Hector slam utilizes occupancy grid maps and a scan matching process to build the map. The approach is based on scan matching optimization of the alignment of beam endpoints with the map learnt so far [1]. For better achievement, the point cloud from the 2D laser scan can be pre-processed but in the case of Hector slam, only the endpoints within a certain threshold is filtered and used for the scan matching process. The scan matching optimization is achieved by minimizing the following function:

$$\xi^* = \underset{\xi}{\operatorname{argmin}} \sum_{i=1}^{n} \left[1 - M(S_i(\xi))\right]$$

where ξ is the pose of the robot with respect to the world coordinate, Si(ξ) is the world coordinates of scan endpoints, and M(Si(ξ)) returns the map value at the coordinates given by Si(ξ) [1]. By minimizing the pose of the robot, the rigid transformation of the current scan and the previous map can be obtained, and consequently, the laser scan can be matched to the map.

As mentioned before, Hector slam builds a 2D occupancy grid map as shown in the figure above which publishes a /map topic (nav_msgs/OccupancyGrid). Each cell is colour coded based on probability of occupation, where black means the cell is occupied, light grey means a cell is free and dark grey means

the cell was not scanned yet. To build the map, Hector slam subscribes to the /vrep/scan topic (sensor_msgs/LaserScan) and processes the data as described above. In addition to the map, Hector slam is also able to publish a /slam_out_pose topic (geometry_msgs/PoseStamped) which is utilized for localizing the robot in the map, such as identifying which room the robot is currently in.

b) Image Recognition and localization

In this project, we need to detect five faces in the room. They are actually quite challenging faces. As is shown below, there are several difficulties in these five images. including pose variance, partial face, cartoon face.





For the face detection, I use the opencv's function detectMultiScale to find the face in the camera image. For the feature detector, I tried several xml file provided by the opencv and chose the best one. I chose LBP cascade for face detector and Harr-like feature for eye detector. The detection result is as follow.

After the face is detected, we can extract the face image from the camera image and pass it to the face classifier. In this project, in order to get the bonus point, I write the face classifier using PCA by myself. The only library I use in this part is Eigen which I use it for solving the eigenvector of the covariance matrix. PCA is taught by Prof. Liu in this course. It is a powerful dimension reduction tool for high-dimensional feature. It is also very famous in the field of face recognition and widely known as eigenfaces. My procedure of writing code in this project is as below.

```
for(int i=0; i<TRAINNO; i++){
  stringstream filename;
  filename << "/home/i4tech/catkin_ws/src/ELEC6910/face_detector/training/" << i+1 << ".jpg";
  Mat face;
  try {   // Surround the OpenCV call by a try/catch block so we can give a useful error message!
    face = imread(filename.str(), CV_LOAD_IMAGE_GRAYSCALE);   // Read the file
  }
  catch (cv::Exception &e) {}
  if (!face.data){                              // Check for invalid input
    cout << "Could not open or find the image" << endl;
    exit(1);
  }
  trainFace.push_back(face);
  //normalizedTrainFace.push_back(normalizeFace(face));
}
```

First, I collect all the training faces, one for which human and there are five in total. The training faces I collected is as below.

```
for (int i = 0; i < faceSize*faceSize; i++){
  meanFace[i] = 0;
}
for (int i = 0; i < TRAINNO; i++){
  for (int j = 0; j < faceSize*faceSize; j++){
    meanFace[j] += trainFace[i].data[j];
  }
}
for (int i = 0; i < faceSize*faceSize; i++){
  meanFace[i] /= TRAINNO;
}
```

Second, I calculate the mean face of these five training faces. All the training face and testing faces will be demeaned first by this mean face in order to remove the DC component in the feature hyperplane.

```
MatrixXf covFace(TRAINNO, TRAINNO);
covFace = trainingset.transpose()*trainingset;
//cout << "Here is the matrix m:\n" << covFace << std::endl;
//cout << "The matrix m is of size " << covFace.rows() << "x" << covFace.cols() << std::endl;
//Get the eigen value and eigen vector of the covFace
SelfAdjointEigenSolver<MatrixXf> eigensolver(covFace);
if (eigensolver.info() != Success) abort();
//cout << "The eigenvalues of covFace are:\n" << eigensolver.eigenvalues() << endl;
for (int i = 0; i < TRAINNO - 1; i++){
  eigenValue(i) = eigensolver.eigenvalues()(i + 1);
}

for (int i = 0; i < TRAINNO - 1; i++){
  eigenVector.col(i) = trainingset * eigensolver.eigenvectors().col(i + 1);
  eigenVector.col(i) /= sqrt(eigenValue(i));
}
```

Third, the covariance matrix of the training set is calculated and it eigenvalues and eigenvectors are computed for extracting the important component for these faces. Since we only have 5 images, we only have 4 eigen faces.

```
// Get the low-dimensional coefficients of train faces

if (deleteEigenOrder){    //if delete several eigenvectors of large eigenvalue
  trainCoef = (eigenVector.transpose() * trainingset).block(0, 0, TRAINNO - 1 - deleteEigenNo, TRAINNO);
}
else{          //if delete the first several eigenvectors of small eigenvalue
  trainCoef = (eigenVector.transpose() * trainingset).block(deleteEigenNo, 0, TRAINNO - 1 - deleteEigenNo, TRAINNO);
}

if (deleteEigenOrder){    //if delete several eigenvectors of large eigenvalue
  testCoef = (eigenVector.transpose() * testingset).block(0, 0, TRAINNO - 1 - deleteEigenNo, 1);
}
else{          //if delete the first several eigenvectors of small eigenvalue
  testCoef = (eigenVector.transpose() * testingset).block(deleteEigenNo, 0, TRAINNO - 1 - deleteEigenNo, 1);
}
```

Fourth, we project the training faces and testing faces through the eigenfaces to get a low-dimensional feature vector. We can also delete some small eigenvector since they didn't affect the result too much.

```
float euclideanDistance_min = 1000000000;
for (int j = 0; j < TRAINNO; j++){
  VectorXf distance(coefNo);
  distance = testCoef.col(0) - trainCoef.col(j);
  //cout << "train No. " << j << "\tdistance: " << distance << endl;
  float euclideanDistance = 0;
  for (int k = 0; k < coefNo; k++){
    euclideanDistance += (distance(k) * distance(k));
  }
  euclideanDistance = sqrt(euclideanDistance);
  if (euclideanDistance < euclideanDistance_min){
    matchedFace = j+1;
    euclideanDistance_min = euclideanDistance;
  }
}
```

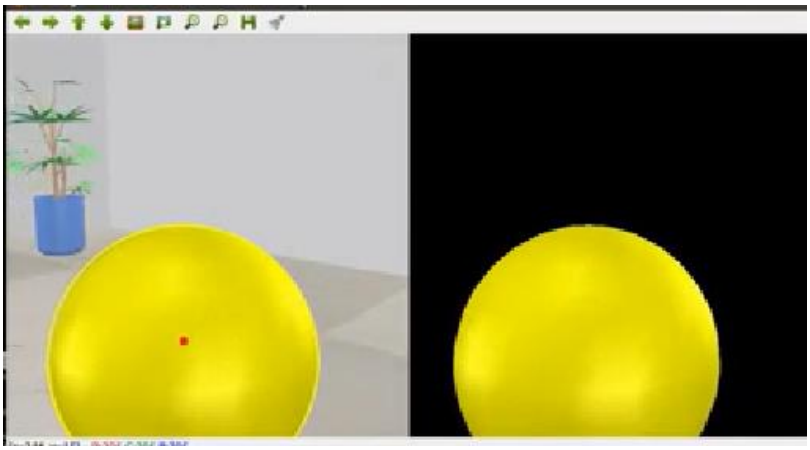
Last but not least, we compare the Euclidean distance between the feature of training faces and testing face so we can find which training face is most similar to the test face detected in the camera image. In this way, we recognize the identity of the face in the room.

c) Visual Servoing

The basic idea of visual servoing is to control the movement trajectory of the car by displaying a yellow ball as the target. Once the yellow ball moves in the environment, the car is supposed to track the ball with a smooth trajectory, the stable angular and linear velocity.

To achieve the visual servoing goal, the first step is to recognize the yellow ball in the captured image. We convert the image from RGB field to HSV field to have a better classification result. After gathering the image in HSV field, a yellow pass filter can be applied to the image by setting the upper and lower bound color value.

The second part is to calculate the center position and radius of the yellow ball in the image. One simple but workable algorithm is calculating the mean position value of all yellow pixels. And calculate the radius with the circle area formula.



Then it follows the control module. In this module we design PI controller to control the linear speed and angular speed of the car. Linear speed is controlled by the shift distance between the center of the ball and the center of the image. Angular velocity is controlled by the size of the ball, because the size of the ball in the image is in inverse proportion to the distance between the robot and the ball. There exists one trick that the parameters of PI controller may requires to be changed according to VREP's frame

update rate, because this FPS also influences the period of the visual callback function as well as the logical part of the PI controller.

We also try to evaluate meanshift algorithm of the ball tracking task, and finish one standard meanshift module. First we initialize the module with a yellow ball image, which is called the target pdf (probability density function), as known as the color distribution histogram. Then at the time of switching to the visual servoing mode, the ball is required to be detected using the color and center detection method described in the last paragraph. After acquiring an initial target bounding rectangle, the following tracking task can be handled with meanshift module, we just compute the most similar candidate ball center point in the searching window with new pdf. However, in VREP, such an ideal simulation environment, color filtering performs much better than meanshift. As a result, we didn't integrate the meanshift into our final project. The corresponding code can be found in an individual test directory if anyone want to evaluate it.

Conclusion

In this project, we learned plenty of things in different fields, including face recognition in the field of computer vision, PID in the field of control theory, SLAM in the field of mapping. We were given a very powerful environment VREP which can simulate a car when it moves in an interesting room. We really appreciated the great work of professor and TA, which enable us to play with this fruitful robotics project.

References

[1] S. Kohlbrecher, O. Von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable SLAM system with full 3D motion estimation," *9th IEEE International Symposium on Safety, Security, and Rescue Robotics. SSRR*, pp. 155–160, 2011.