ELEC6910R Final Project Report

JIANG, Jingbo and CHENG, Wei
Jingbo.jiang@connect.ust.hk   wchengad@connect.ust.hk

1 Grouping and Work Assignment

JIANG, Jingbo and CHENG, Wei both from ECE department form a group. JIANG, Jingbo is from VLSI Research Lab and devote his effort in deep learning and LDPC codes. CHENG, Wei is under the supervision of Prof. SHI, Ling and Prof. FANG, Lu, his research interest is machine learning and control.

The project is separated into several parts and equally assigned according to group member's specialties, specifically:

- JIANG, Jingbo
  - Image detection and localization from robot view
  - Trained a Sklearn cascade regression model for image identification
  - Image identification using ORB and Canny block matching
  - Face identification using the pre-trained model
- CHENG, Wei
  - Robot keyboard control and SLAM using existed packages
  - Moving object (yellow ball) detection in camare coordinate
  - Implement and Evaluated three different controller, naive controller, PID controller and Kalman Filter - LQR controller
  - Control panel, GUI design, project hierarchy design

2 System Overview

Due to the system limitation of the group members, this system environment is not the recommended system setting which is listed as follows, but achieved a comparable performance

- Operation System: Ubuntu 14.04
- ROS Version: Indigo
- V-Rep Version: Education v3.4.0 Linux
- Language: Ros Indigo built-in Python2.7

The file hierarchy of the project is illustrated by the following file tree, where three packages and one launch file is included, each package generates by *catkin_create_pkg* command and includes one ROS node and all nodes are launched together by one launch file. Noted that the keyboard controller, hector mapping slam package are installed separately in the root directory of the system which are not contained in the project file hierarchy but also be started in the same launch file.

```
src
|
├─ main_node
|   ├─ src
|   |   └─ main_node.py
|   ├─ CMakeLists.txt
|   └─ package.xml
├─ visual_servoing
|   ├─ src
|   |   └─ visual_servoing.py
|   ├─ CMakeLists.txt
|   └─ package.xml
├─ face_detector
|   ├─ src
|   |   └─ face_detector.py
|   ├─ pictures
|   |   └─ ...
|   ├─ CMakeLists.txt
|   └─ package.xml
└─ main.launch
```

The relationship between launched nodes and topics are visualized in Fig. 2 using rqt_graph command. More specifically,

- *keyboard* node: this node is started using installed package *key_teleop*, which generates a command panel letting user control the robot via keyboard, the published topic */key_teleop* is then remapped to */cmd_vel* which can be subscribed by vrep_ros_interface.
- *hector_mapping* node: this node is also started using the install package *ros-indigo-hector-slam*, which subscribe the */vrep/laser_scan*, and reconstruct the scene, then publish reconstructed map and robot localization, finally display in *rviz*.
- *vrep_ros_interface* node: this node is the interface between ros nodes and V-REP simulation environment, the specific publish and subscribe topics have been listed in the project desription.

- *main_node* node: this node provides Graphical User Interface (GUI) for user to switch between different demo tasks and project options, display the object or face detection output, tracking error graph, etc. This node publish some enable/disable messages for functional node for example *visual_servoing* when GUI button is clicked or checked, and subscribe the results from functional nodes, then display them on the screen.
- *visual_servoing* node: this node subscribes the */vrep/image*, detects the moving yellow ball, then controls the robot by publishing the control signal */vrep/cmd_vel*. Three alternative tracker can be choose by GUI in *main_node*, tracking errors are back plotted to the GUI.
- *face_detector* node: this node also subscribes the */vrep/image*, identifies the given pictures in robot view, and also detects faces in the robot view. Bounding boxes of the picture and detected faces are displayed through GUI, and the 3D locationa of identified pictures are visualized in *rviz*.

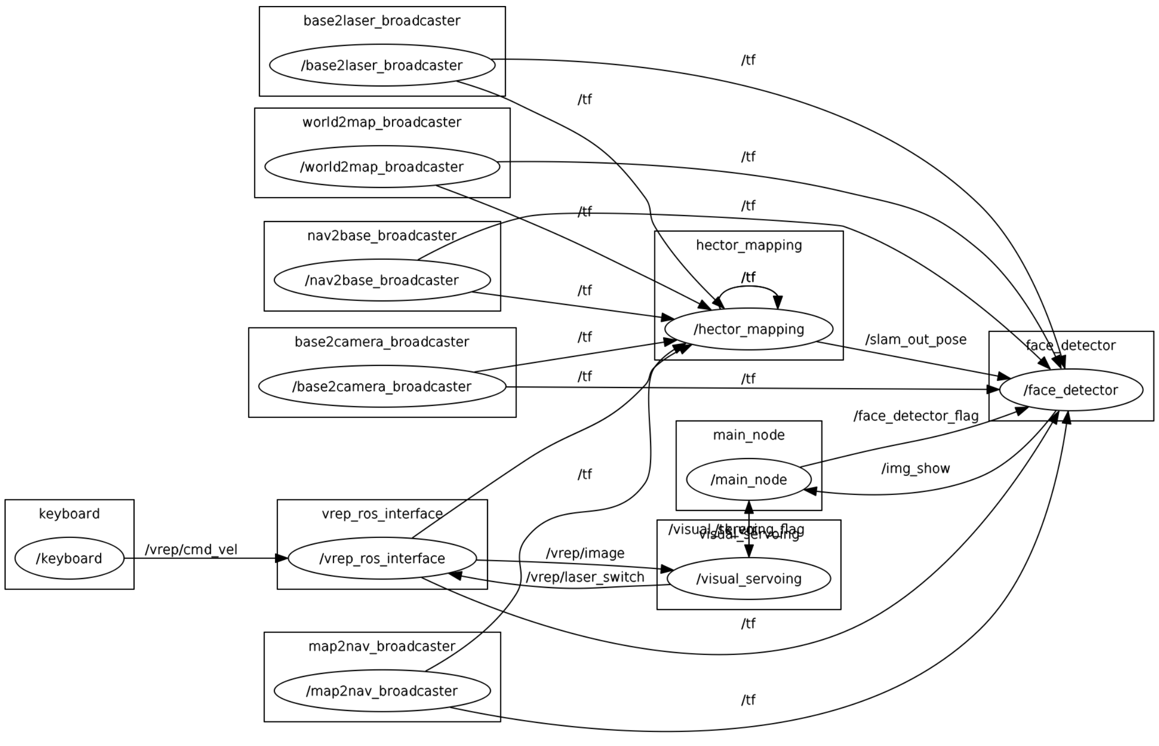


Figure 1: rqt_graph of the system.

3 Robot Control and Mapping

This section introduce the implementation of robot control and mapping using the existed package.

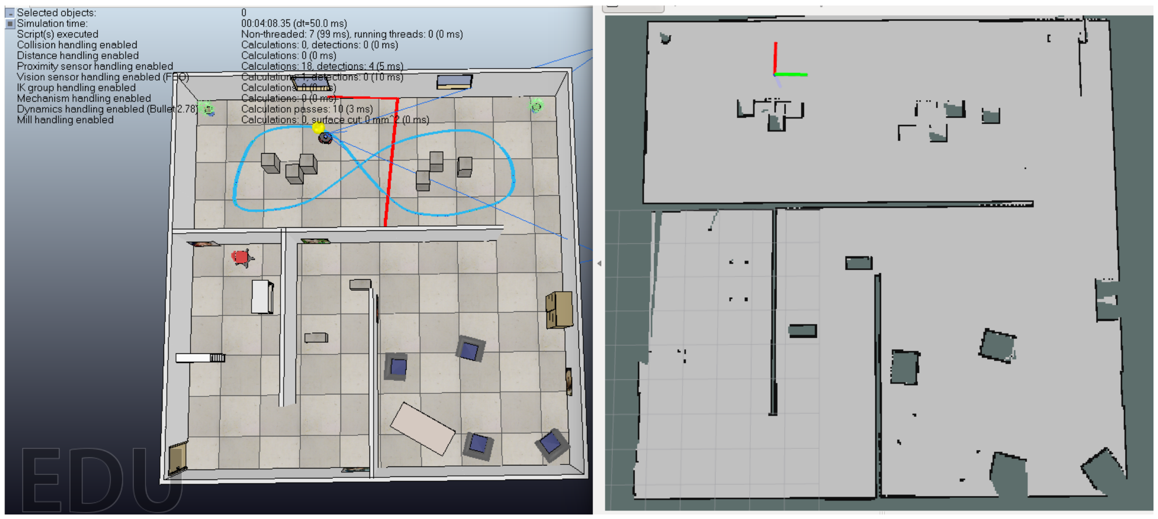
3.1 Robot Control

Robot control is performed by launching a *key_teleop* node which package can directly be obtained by installing a rospack. The published topic by *key_teleop* is */key_vel* which can not be subscribed by the *vrep_ros_interface*. Thus, it has to be remapped to */vrep/cmd_vel* in the launch file. Two variable is feasible, robot yaw speed and robot y speed controlled by left/right and up/down arrow on keyboard respectively.

3.2 Mapping and Localization

The functionality of SLAM is realized by ros-indigo-hector-slam package, and the node hector_mapping subscribe the laser scan date generated in V-Rep simulation platform and reconstruct the 2D map and localize the robot's transformation (translation and rotation), both are visualized using rviz, another useful ros package for robotics visualization. In our launch file, the transformation relationship is packed using ros tf2 packages, the relative coordinate transformation of different components of the scene are constructed in a twist fashion.

The 2D mapping and robot localization results are shown in Fig. 2, the reconstructed 2D map and estimated robot location is accurate by comparing to the scene in V-Rep visually. The mapping accuracy may degraded when robot motion is to large, meanwhile the moving yellow ball also may affect the robot localization then finally affect the reconstructed map.



```
Selected objects:                          0
Simulation time:                           00:04:08.35 (dt=50.0 ms)
Script(s) executed                         Non-threaded: 7 (99 ms), running threads: 0 (0 ms)
Collision handling enabled                 Calculations: 0, detections: 0 (0 ms)
Distance handling enabled                  Calculations: 0 (0 ms)
Proximity sensor handling enabled          Calculations: 18, detections: 4 (5 ms)
Vision sensor handling enabled (FBO)       Calculations: 1, detections: 0 (10 ms)
IK group handling enabled                  Calculations: 0 (0 ms)
Mechanism handling enabled                 Calculations: 0 (0 ms)
Dynamics handling enabled (Bullet 2.78)    Calculation passes: 10 (3 ms)
Mill handling enabled                      Calculations: 0, surface cut: 0 mm^2 (0 ms)
```

Figure 2: Left: Robot and virtual indoor scene in V-Rep; Right: Reconstructed map in localized robot in *rviz*.

4 Visual Servoing

This section introduces the visual servoing design, which including two aspect, object detection and tracking, we designed three types of tracking controller, Naive controller, PID controller and Kalam Filter

- LQR controller, and finally the performance of the PID and LQR controllers are evaluated by tracking error estimation.

4.1 Object Detector

As the task is to identify only the yellow ball, designing a general object detector is not necessary, so our object detector is implemented using simple *cv2* api, quick but accurate and efficient.

When received one message, first the ros message is transformed to *cv2* images via *cvBridge* package in ros. Then flip the image and transfer it into HSV color space to tackle the color variation on the yellow ball due to reflection. Then the mask image is generated by threshold upper and lower bound of yellow color in HSV. To extract clean mask, tow morphological transformations, erosion and dilation are applied. Contour is then extracted together with the object bounding box and center, as shown in left subplot in Fig. 4 and Fig. 5.

4.2 Object Trackers

The object tracking problem be can separated into two independent control problem with regard to the robot's motion mode. The control signals, speed of yaw and y in robot body coordinate, should be calculated under some control criteria. In order to track a moving object, we set the purpose of control is to minimize the position error of object center and camera center in image plane, and to minimize the size error of the object shown in image plane. By setting this target, the robot can track the moving object with a certain distance and follows the objects motion.

Suppose the object center lies at (x, y) in image plane, and the bounding box of the object has a size r, both units are pixels. The desired object center and object size are $\bar{x}$, $\bar{(r)}$ So the control error the controller want to minimize is

$$\text{err\_x} = \frac{x - \bar{x}}{\bar{x}}$$
$$\text{err\_r} = \frac{r - \bar{r}}{\bar{r}}$$

where we set $\bar{x} = 256$ and $\bar{r} = \frac{1}{8}$ in our implementations.

Under this criteria, we designed three different controllers to tackle the tracking problem.

4.2.1 Navie Controller

The most naive way to control the robot is to assign a fixed control relationship between estimated error and control signal. The control speed of yaw is modeled as linear depending on the error of object center, and the control speed of y is formulated as the l-0 norm of error of object size, which are

$$\text{speed\_x} = L * \text{err\_x}$$
$$\text{speed\_yaw} = D * |\text{err\_r}|_0 * sign(\text{err\_r})$$

where L and D are control parameters control the direction and response level of control on different error.

4.2.2 PID Controller

A proportional–integral–derivative (PID) controller is a control loop feedback mechanism widely used in industrial control systems and a variety of other applications requiring continuously modulated control. A PID controller continuously calculates an error value $e(t)$ as the difference between a desired setpoint and a measured process variable and applies a correction based on proportional, integral, and derivative terms (denoted P, I, and D respectively). The PID control is defined ad

$$u(t) = K_p e(t) + K_d e'(t) + K_i \int e(t)dt$$

where $e(t)$ is the defined err_x and err_r in each PID controller and the integrator $K_i * \int e(t)dt$ should be clipped within bounded value b. Please check our submitted code, for detailed implementation.

4.2.3 Kalman Filter - LQR Controller

When we model the robot motion system as a state-space dynamic system, we can write the state and measurement model as

$$x = Ax + Bu + w$$
$$y = Cx + v$$
$$w \sim N(0, Q),\ v \sim N(0, R)$$

The optional control strategy is to use LQR controller which is concerned with operating a dynamic system at minimum cost, the cost function can be defined as

$$\underset{u(t),\ldots,u(0)}{\arg\min} = J(u(t), \ldots, u(0)) = x(t)^T A x(t) + \int x(t)^T Q x(t) + u(t)^T R u(t) dt$$

This problem can be solved using Dynamic Programming (DP), and the optimal control value $u(t)$ at time t is

$$u^\star(t) = -Lx(t)$$
$$L = (B^T P B + R)^{-1} B^T P A$$

where P is the estimated covariance of the state x, which can be estimated and updated by a Kalman Filter (KF) by the following transition model and measurement model

$$\begin{aligned}
\hat{x}_{k|k-1} &= A_{k-1}\hat{x}_{k-1|k-1} \\
P_{k|k-1} &= A_{k-1}P_{k-1|k-1}A'_{k-1} + Q \\
K_k &= P_{k|k-1}[P_{k|k-1} + R]^{-1} \\
\hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k[y_k - \hat{x}_{k|k-1}] \\
P_{k|k} &= P_{k|k-1} - K_k P_{k|k-1}
\end{aligned}$$

where in our application, we set the measurement y as the error measurement on object center and object size. And the initial state $x(0) = 0, P(0) = 0.1$, in both yaw and y controllers. For more details of our realization, please refer to our submitted codes.

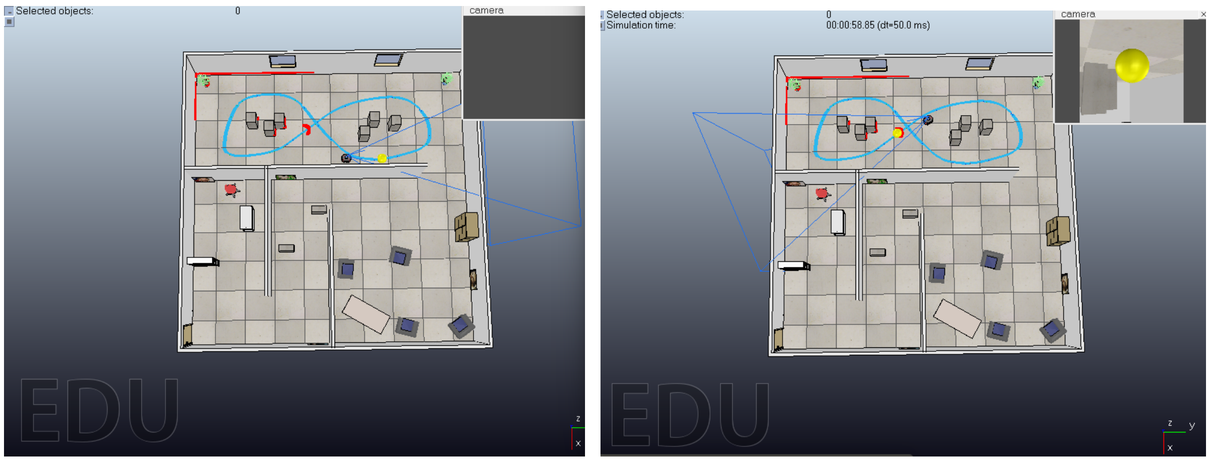


Figure 3: Starting and ending points of PID tracker and LQR tracker.

4.2.4 Comparison between Controllers

Finally, we conducted an experiment on the tracking performance of the PID controller and LQR controller, robots are initially placed in the same starting and ending location shown in Fig. 3.

The results are shown in Fig. 4 and Fig. 5, the left subplot display the robot view of tracking target, and the right view is the tracking error of object center and object size which are in red and blue line respectively.

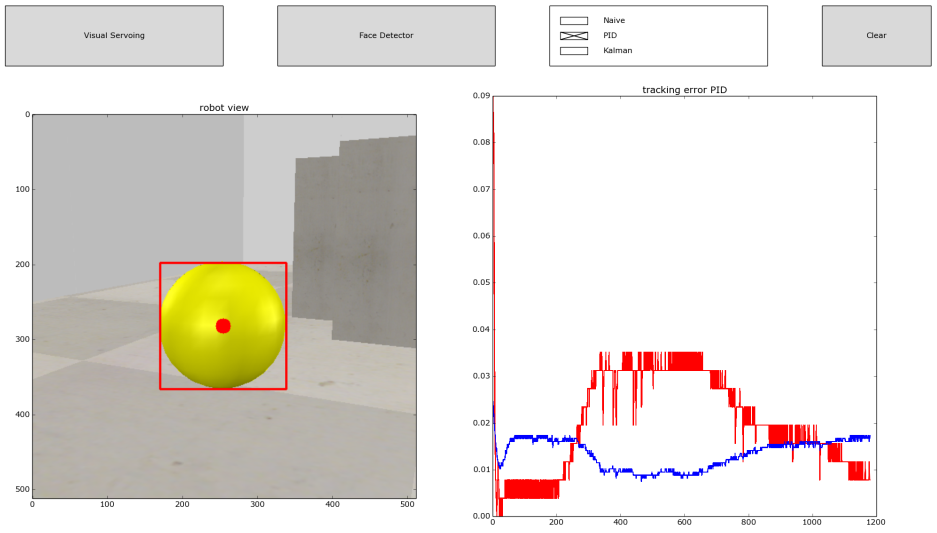


Figure 4: Left: Robot view in tracking task using PID controller; Right: Visualization of control error of PID controller, err_x in red and err_r in blue.

In control of yaw speed, the parameters of PID controller are well tuned, so that the overshoot of the initial pose, which can be consider as the impulse response, is relatively low comparing to the LQR controller who has a high overshoot and oscillation. When the robot starting to turning around, the maneuverability of the LQR is better demonstrated by the low error rate in later part.

In control of y speed, both tracker bounds the error rate in a low level.

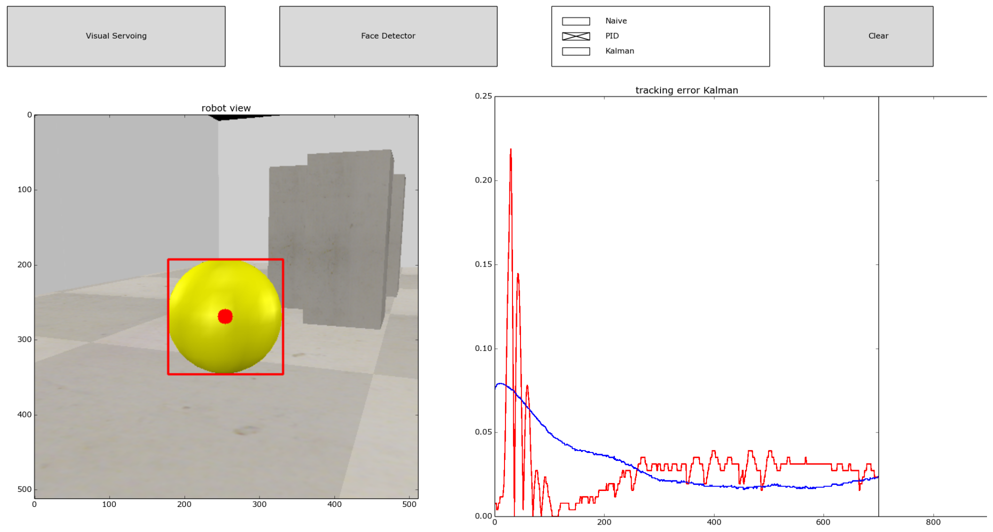


Figure 5: Left: Robot view in tracking task using LQR controller; Right: Visualization of control error of LQR controller, err_x in red and err_r in blue.

5 Image Identification and Face Detection

This section describes the image identification and face detection tasks, different methodology in our process of the project are also discussed.

We should first state our difficulty on this task, because of the limitation caused by the operation system. Due to the desktop of our group has to maintain in Ubuntu 14.04, so the supported ros version is only Indigo which only has OpenCV2 APIs and no *opencv_contrib_python* supported in our environment (low level numpy version conflicting). After a long time trial of OpenCV3 ros python installation, we gave up our original idea of using SIFT feature extraction and then perform homography to find a known object with rotation and distortion. Thus, the following desribed method are both not optimal, due to their limitation on rotation and distortion, which mean it can only give a rectangular bounding box and estimated relative picture to camera rotation is identical.

5.1 Image Identification using Regression Model

We first tried to use regression model to identify the given image from robot view, we applied a logistic model (or logit model) which is a widely used statistical model that in its basic form uses a logistic function to model a binary dependent variable.

We use the python module *sklearn.linear_model.LogisticRegression* to firstly train a regression model given the five training data, then try to inference the image patch using the model and compare the output possibility with a threshold. While in real simulation, we found that the accuracy and stability is with less satisfactory, maybe due to the reason that training data is not enough.

5.2 Image Identification using Multi-scale Template Matching

Then we tired feature extraction and template matching method. Firstly the 5 reference image is firstly loaded resized and transformed into a 100 gray scale templates, then the ORB feature and edge map are extracted by ORB feature extractor and Canny detector, such that the database of the template features are prepared. When the image captured by the robot triggers a callback, t the image is the resized into multi-scale level, then both ORB features and Canny edge maps are extracted and compared to the database. The matched template with greatest similarity will be identified, and the position of the detected image will be resacled to the original scale level and finally tagged on the input image with a bounding box. This multi-scale template matching method is simple but quick, because of both the ORB feature and the Canny detector requires minimum calculation. The detection results are demonstrated in Fig. 6 shown in red bounding boxes.

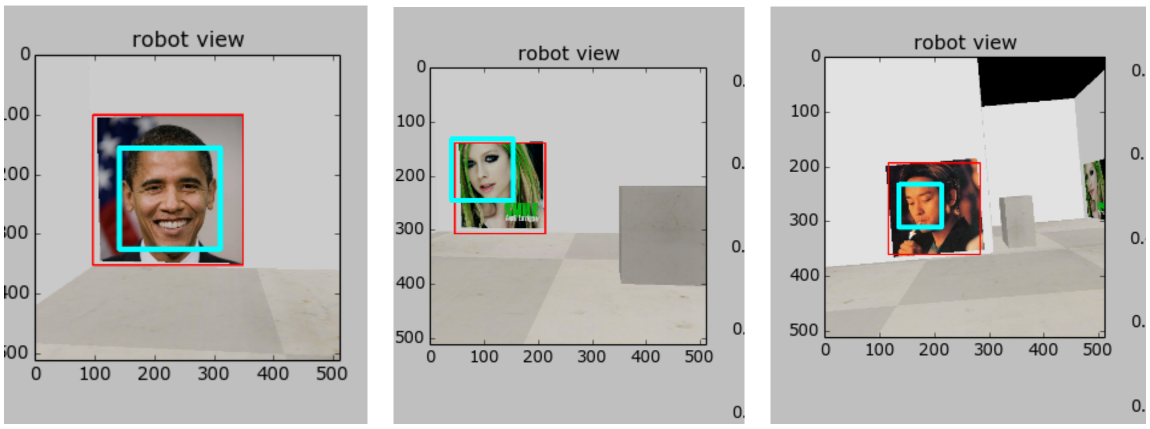


Figure 6: Image identification and face detection with detected bounding box in red and blue respectively.

5.3 Face Detection

We designed our face detector using the OpenCV CascadeClassifier, a pre-trained model *haarcascade_frontalface_default* is loaded which robust enough for our application. The receive sensor image is firstly transformed into grayscale image then sent into the classifier, and the bounding boxes of the detected faces are displayed in blue rectangles shown in Fig. 6.

6 Possible Add-up Point - GUI Design

To better demonstrate and evaluate the functionality of our project, we built a *main_node* which contains functionality selection, intermediate result display and statistics analysis via a GUI panel as shown in Fig. 4 and Fig. 5.

When the demonstration starts, both *visual_servoing* and *face_detection* node are not activated on their launch. We designed two buttons to enable the two functionality separately, when a button is clicked, the *main_node* will publish a ros *std_msgs.msg.Bool* message, then the two node will select to enable or disable its execution depending on the subscribed message.

Output results of detection including picture, faces and moving balls are required to be visualized, so in *visual_servoing* and *face_detection* node will send a *sensor_msgs.msg.Image* when they generates an output. The *main_node* subscribe this type of topic, and plot them on the control panel.

In visual servoing task, we need to analyze the control error and evaluate the performance of our designed controller, the *main_node* maintain two buffers to save the received errors which are transmitted via *std_msgs.Float32* messages. A click button is also designed to clear the buffers which is placed in rightmost corner as shown in Fig. 4. The selection of different trackers are decided by three check buttons in Fig. 4, and it triggers a publisher to send a *std_msgs.msg.String* message, *visual_servoing* changes its tracker when receives this message.

7 Functionality Inventory

Finally, to conclude, we listed the project requirement and identified our completion status with ✓and × marks.

- Basic functions:
  - ✓ Use your launch file launch ros and ros nodes (including rviz), start V-REP simulation
  - ✓ Control the mobile robot move in the environment by keyboard
  - ✓ At same time, the grid map should be shown in rviz.
  - × At same time, if the target images occurs in the filed of view of vision sensor, its location should be estimated and marker should be shown in rviz
  - ✓ Move robot to upper room and switch to auto-tracking mode. The robot should automatically track and follow the yellow ball without keyboard control
- Add-pu points:
  - ✓ Besides detection, you can also recognize the faces on the wall.
  - × Automatic exploration of the environment.
  - ✓ Controller performance evaluation for the tracking problem.
  - ✓ Any further contributions not included in the standard tasks and specifications.
- Possible Add-up points
  - ✓ To better demonstrate and evaluate the functionality of our project, we built a *main_node* which contains functionality selection, intermediate result display and statistics analysis via a GUI panel