

7.Face recognition

7.1 Overview

wiki: http://wiki.ros.org/opencv_apps

Source code: https://github.com/ros-perception/opencv_apps.git

Most of the code was originally taken from:<https://github.com/Itseez/opencv/tree/master/samples/cpp>

Functional package: `~/software/library_ws/src/opencv_apps`

The topic subscribed by this function package is `/image`. What we have to do is to open the camera node, write a topic that converts the camera topic into a `/image` node, and publish the `/image` topic.

Turn on the camera and publish the path of the node of the `/image` topic:

```
~/dofbot_ws/src/dofbot_visual/scripts/pub_image.py
```

The `opencv_apps` program provides various nodes that internally run the functions of `opencv` and publish the results to a ROS topic. When using the `opencv_apps` program, you only need to run a launch file according to your own business needs, so that you no longer have to write program codes for these functions.

ROS Wiki has related node analysis, topic subscription and topic publishing of the corresponding node, introduction of related parameters, etc. See the ROS Wiki for details.

Contents

1. Introduction, usage
2. Edge Detection Nodes
 1. edge_detection
 2. hough_lines
 3. hough_circles
3. Structural Analysis Nodes
 1. find_contours
 2. convex_hull
 3. general_contours
 4. contour_moments
4. People/Face Detection Nodes
 1. face_detection
 2. face_recognition
 3. people_detect
5. Motion Analysis Nodes
 1. goodfeature_track
 2. camshift
 3. fback_flow
 4. lk_flow
 5. phase_corr
 6. simple_flow
6. Object Segmentation Nodes
 1. segment_objects
 2. watershed_segmentation
7. Image Filter Nodes
 1. rgb_color_filter
 2. hls_color_filter
 3. hsv_color_filter
8. Simple Image Processing Nodes
 1. adding_images

7.2 Use

Step 1: Start the camera

```
roslaunch dofbot_visual opencv_apps.launch img_flip:=false
```

- img_flip parameter: whether the image needs to be flipped horizontally, the default is false.

The [usb_cam-test.launch] file opens the [web_video_server] node by default, and you can directly use the [IP:8080] web page to view images in real time.

Step 2: Activate the face recognition function

```
roslaunch opencv_apps face_recognition.launch # face recognition
```

Almost every case of ros+opencv application will have a parameter [debug_view], Boolean type, whether to use Opencv to display pictures, which is displayed by default.

If no display is required, set it to [False], for example

```
roslaunch opencv_apps contour_moments.launch debug_view:=False
```

However, after starting in this way, some cases cannot be displayed in other ways, because in the source code, some [debug_view] is set to [False], which will turn off image processing.

7.3 Display method

- `rqt_image_view`

Enter the following command to select the corresponding topic

```
rqt_image_view
```

- `opencv`

The system displays it by default, no need to do anything.

- Web viewing

(Same as under LAN) Enter IP+port in the browser, for example:

```
192.168.2.116:8080
```

For the specific ip, use your current virtual machine ip.

7.4 Face recognition display

This case is based on autonomous training and real-time recognition by collecting images of people in real time, and the steps are slightly complicated.

| Parameter | Type | Default | Analyze |
|--------------------------------|--------|---------|--|
| <code>~approximate_sync</code> | bool | false | Subscribe to the topic [camera_info] to get the default coordinate system ID, otherwise use the image information directly. |
| <code>~queue_size</code> | int | 100 | Queue size for subscribing topics |
| <code>~model_method</code> | string | "eigen" | Face recognition method: "eigen", "fisher" or "LBPH" |
| <code>~use_saved_data</code> | bool | true | Load training data from <code>~data_dir</code> path |

| Parameter | Type | Default | Analyze |
|-----------------------|--------|---------------------------|---|
| ~save_train_data | bool | true | Save training data to ~data_dir path for retraining |
| ~data_dir | string | "~/opencv_apps/face_data" | Save training data path |
| ~face_model_width | int | 190 | Width of training face images |
| ~face_model_height | int | 90 | Training face image height |
| ~face_padding | double | 0.1 | Filling ratio of each face |
| ~model_num_components | int | 0 | The number of components of the face recognizer model (0 is considered unlimited) |
| ~model_threshold | double | 8000.0 | Face recognition model threshold |
| ~lbph_radius | int | 1 | Radius parameter (for LBPH method only) |
| ~lbph_neighbors | int | 8 | Neighborhood parameters (only for LBPH method) |
| ~lbph_grid_x | int | 8 | Grid x parameter (only for LBPH method) |
| ~lbph_grid_y | int | 8 | Grid y parameter (only for LBPH method) |
| ~queue_size | int | 100 | Image subscriber queue size |

Operation steps:

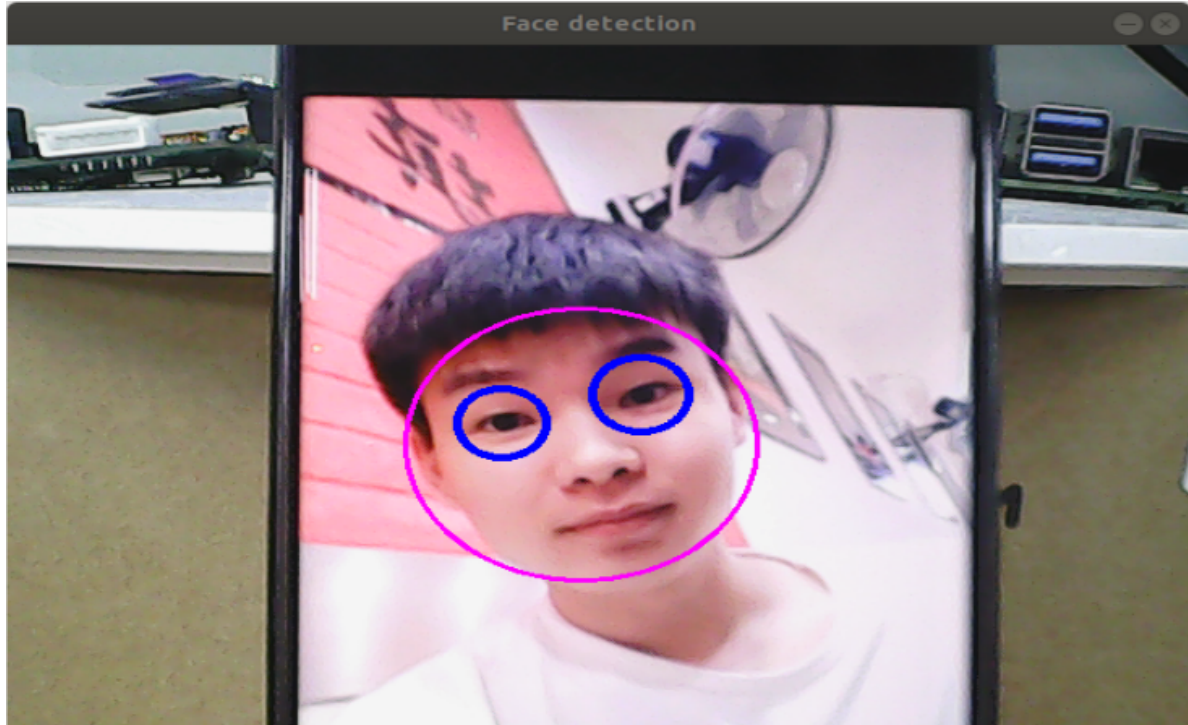
1. First, enter the character's name after the colon in the picture below: Yahboom
2. Confirm name: y
3. Then place the face in the center of the image and click OK.
4. Add a photo in a loop: y, click to confirm.
5. To end image collection, enter: n and click Confirm.

6. Close the launch file and restart.

If you need to enter the recognized identifications, cycle through steps 1 to 5 until all identified persons have been entered, and then proceed to step six.

```
face_recognition_trainer.py
Please input your name and press Enter: Yahboom
Your name is Yahboom. Correct? (y/n): y
Please stand at the center of the camera and press Enter:
taking picture...
One more picture? (y/n): y
taking picture...
One more picture? (y/n): y
taking picture...
One more picture? (y/n): y
taking picture...
One more picture? (y/n): y
taking picture...
One more picture? (y/n): y
taking picture...
One more picture? (y/n): y
taking picture...
One more picture? (y/n):
```

Step 3: Ensure that faces can be recognized



Final recognition effect

