avatar_face_recognition

Author: Neelkumar Mistry
Email: mistry.neel92@gmail.com

==========

This face recognition is based on pre-trained Neural-Net and face_recognition of dlib. It is wrapped to work with ROS. It takes images from a video stream topic and generates:

- **json string containing the names of each person in the frame ("Unknown" for someone it is not able to recognize) and if the person is wearing a mask or not.**
- **face bounding box dimensions (row, col, height, width)**
- **face image as a compressed image**

All three things in the list above is put into an array.

==========

Node Files

rebuild_database.py

This node acts as a teacher node for our avatar. It introduces new faces to avatar and stores the facial measurements data in a pickle file to use it later for face recognition.

All face images we want to introduce to avatar are stored in the directory: "/catkin_ws/src/avatar/faces". Under this directory, we make a folder for each person we want to introduce to avatar and then save multiple images of that person under that folder.

Next, a json file is saved at /home/ros/catkin_ws/src/avatar/info.json which links the folder names in "faces" directory with person names.

For example: if we want to specify that folder named "Neel" contains pictures of a person named "Neel Mistry", then our json file data would look like below:

{
"Neel": {"name": "Neel Mistry"}
}

Once, we are done with these preparatory steps, we can run the node and train avatar on new faces. The node first grabs the paths of our input images and reads the json file data. Then it enumerates through all the dataset images and convert them to dlib ordering (RGB). Using face_recognition with "hog" method, it detects the bounding box coordinates of each face and creates encodings for them. Then, each encoding is appended in a list of encodings. Similarly, corresponding person names for those encodings are also appended in another list. Finally, these lists are merged together to create a dictionary and then that dictionally is serialized in a pickle file. This pickle file acts as a database for avatar.

movie_2_image.py

Publishes to topic: /camera/image_raw/compressed

This node uses "sample.mov" (location: /catkin_ws/src/avatar_face_recognition/sample/sample.mov) and creates compressedImage objects from it. Then it sets the three fields of compressedImage to the following:

- Header field is stamped with current timestamp.
- Format field is set to jpeg.
- For data field, it encodes the cv2 image to a jpg, generates a numpy array and converts it to a string.

In short, this node takes a .mov video file as an input, generates compressedImage string messages of jpg at the set rate and publishes the message to the topic mentioned above.

cam_2_image.py

Publishes to topic: /camera/image_raw/compressed

This node performs identical work compared to movie_2_image.py node, except this node takes input from live webcam instead of a pre-recorded movie file.

face_finder_haar.py

Subscribes to topic: /camera/image_raw/compressed
Publishes to topic: /face_finder/faces

This node subscribes to a compressed image producing topic, reads the numpy array data saved to the compressedImage. Next thing it does is, it converts the image frames to gray-scale using cv2 utilities. Then, using gray-scale image and haar cascade, it finds all the faces in image and embeds each face in appropriately sized bounding box.

It then uses our pickle file, which contains the encoding of all faces that avatar currently recognizes. It compares each face in the image with faces in pickle file and calculates Euclidean distance between them. If this distance passes our threshold criteria, then it assigns that name to that face. (Less distance means more match.) If it does not find a match qualifying our threshold criteria, then the name assigned is "Unknown". All the names are saved in a list.

Then, using "Shapely" library's "box" feature, it finds out if any of the face boxes in the image overlaps with other. If the overlap is more than set percentage then it compares the Euclidean distances for both face-boxes and keeps the one with smaller number. The other face-box is discarded at this stage.

Next, it reads thru all the names in the list looking for any duplicate entries. If it finds a person's name twice, it keeps the one with smaller Euclidean distance and discards the other one. Note that two or more entries are allowed for "Unknown".

Finally, it saves names, face-box parameters (row, col, height, width) and compressed image for face into a "FaceArray" message (please see /catkin_ws/avatar_face_recognition/msg for more details on this message type.) and publishes the message to publisher topic.

****This node has haar based face detection which fails to detect faces with mask. Due to this reason, we developed another node to detect faces with/without mask. That's why, we no longer use this node - it is obsolete.*****

face_finder.py

Subscribes to topic: /camera/image_raw/compressed
Publishes to topic: /face_finder/faces

This node performs exactly same task as face_finder_haar.py, except it does not use haar to detect faces in image frames. Instead, it used pre-trained Neural-Net to detect faces. Rest of the process remains same.

mask_detector.py

Subscribes to topic: /face_finder/faces

Publishes to topic: /mask_detector/faces_with_mask

This node subscribes to a "FaceArray" producing topic and further processes the compressed images to detect if the face in the image is having a mask on or not. Then It upends that information along with confidence value (e.g. No-Mask 89.9%) to the json string and publishes the modified "FaceArray" to publisher topic.

face_2_screen.py

Subscribes to topic: /mask_detector/faces_with_mask

Publishes to: Screen

This node subscribes to “FaceArray” message publishing topic. It reads the published data, takes compressed images, stacks them horizontally in a window, and displays face images on screen.

Launch Files

databaseBuilder.launch

Use this file to create/update your face encoding database. This file launches the following nodes.

- add_new_face_client.py
- rebuild_database.py

realsense_input.launch

Use this file to obtain input from web-cam. This file launches the following nodes.

- cam_2_image.py
- image_2_screen.py

movie_input.launch

Use this file to obtain input from pre-recorded video file. This file launches the following nodes.

- movie_2_image.py
- image_2_screen.py

recognizer.launch

Use this file to recognize the faces and detect mask in a video stream. This file launches the following nodes.

- face_finder.py
- mask_detector.py
- face_2_screen.py