ROS Perception in 5 Days

- Summary -

Estimated time to completion: **3 hours**

In this chapter, you will learn how to recognize different faces. Specifically, you will learn about:

- How to use a face_recognition Python module
- How to detect a face based on **one single picture**
- How to integrate it in ROS
- How to detect multiple faces

- End of Summary -

Chapter 5: People Perception Part 2 - Face Recognition

You now know how to detect different faces, but what if you want to recognize a familiar face? This is what you will learn here.

You will work with this scene: recognize three different people using a Fetch robot.

5.1 Create A Basic Package for this Unit and Download Extra Files

Let's create a package named **my_face_recogniser**:

▶ Execute in WebShell #1

In [ ]:
```
cd ~/catkin_ws/src/
catkin_create_pkg my_face_recogniser rospy sensor_msgs cv_bridge
cd ~/catkin_ws
catkin_make
source devel/setup.bash
rospack profile
```

Add to your package some example recognition images. You can get these images from the **person_img** folder of the *face_recognition_pkg* package. To copy this folder to your package, execute the following commands:

▶ Execute in WebShell #1

In [ ]:
```
cd ~/catkin_ws/src/
git clone https://bitbucket.org/theconstructcore/face_recognition_pkg.git
cp -r cd ~/catkin_ws/src/face_recognition_pkg/face_recognition_pkg/person_img
rm -rf face_recognition_pkg
```

5.2 Start The Face Recognition Package

For this, you will use the face_recognition python module (https://github.com/ageitgey/face_recognition).

It has many features apart from the face recognition feature, like face finding or applying digital makeup. But it provides a simple solution for rudimentary face recognition. Below, you have an example of how to recognize **one face**:

There are some elements to comment on:

In [ ]:
```
import sys
import os
import rospy
from sensor_msgs.msg import Image
import rospkg

print("Importing cv2")
import cv2
print("Importing cv2..Done")
import face_recognition
from cv_bridge import CvBridge, CvBridgeError
```

You will need these special packages, **cv2** (OpenCV), **cv_bridge** (CV_Bridge), and **face_recognition**, to recognize the faces. You will also need **rospkg** to easily find files that are inside other ROS packages.

In [ ]:
```
# get an instance of RosPack with the default search paths
rospack = rospkg.RosPack()
# get the file path for my_face_recogniser
self.path_to_package = rospack.get_path('my_face_recogniser')
print self.path_to_package
```

Next, **retrieve an image** for the person that you want to recognize. **It is important to note** that by providing only one image, the face recognition system won't make positive detections unless what it visualizes is very similar to the image provided. So, the more images you provide using different configurations, the better the detections. In this instance, one image is enough.

In [ ]:
```
# Load a sample picture and learn how to recognize it.
image_path = os.path.join(self.path_to_package,"scripts/standing_person.png")
```

Once you have that, it starts the real image processing:

In [ ]:
```
standing_person_image = face_recognition.load_image_file(image_path)
standing_person_face_encoding = face_recognition.face_encodings(standing_persc
```

You can see that you load the image into a **face_recognition** variable, and then you extract the encodings. Note that you are getting the first encoding [0]. This is because you only have one image of that person. More images will generate more encodings.

We initialize some variables, and then we underscale the image retrieved from the cameras. In this case, half of the original size. That way, you work with fewer pixels, making the recognition process faster. But, when you show the images captured, you will show the original size.

But this has a side effect: if you reduce the size **too much,** the face encoder has less information to work with, and therefore, it makes it harder to extract human features and match them with someone.

It has to be a compromise between performance and functionality. If you are too far away from your subjects, then you will have to work with bigger images.

With an image this small, it will be difficult for the algorithm to work. You will have to get closer.

In [ ]:
```python
# Initialize some variables
face_locations = []
face_encodings = []
face_names = []

# Resize frame of video to 1/2 size for faster face recognition processing
# If this is done be aware that you will have to make the recognition nearer.
# In this case it will work around maximum 1 meter, more it wont work properly
small_frame = cv2.resize(video_capture, (0, 0), fx=0.5, fy=0.5)
```

Let's process the image frame.

In [ ]:
```python
# Find all the faces and face encodings in the current frame of video
face_locations = face_recognition.face_locations(small_frame)
face_encodings = face_recognition.face_encodings(small_frame, face_locations)

if not face_encodings:
    rospy.logwarn("No Faces found, please get closer...")

face_names = []
for face_encoding in face_encodings:
    # See if the face is a match for the known face(s)
    match = face_recognition.compare_faces([standing_person_face_encoding], fa
    name = "Unknown"

    if match[0]:
        rospy.loginfo("MATCH")
        name = "StandingPerson"
    else:
        rospy.logwarn("NO Match")

    face_names.append(name)
```

First, extract the location of the faces. From those locations, extract the encodings. These encodings are like a "fingerprint" for each face, defining it. You do this for each image frame that you capture.

Next, for all the faces detected and, therefore, encodings, **compare** those encodings with the standing_person_face_encoding. If there is a match, you found the standing person, and you add that name to the face_names list.

Finally, display the results of the face recognition. This means showing the image captured and the locations where you detected a face, drawing a square with the name of the recognized face there.

In [ ]:

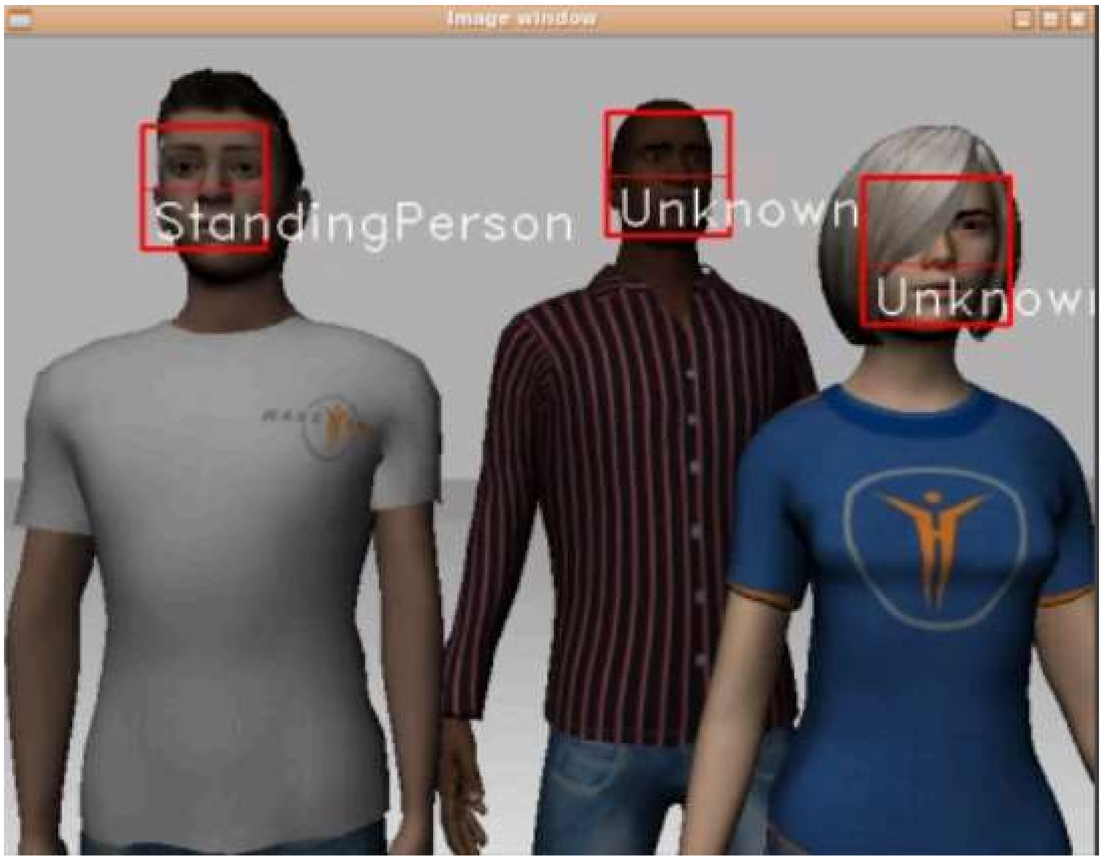
```
# Display the results
for (top, right, bottom, left), name in zip(face_locations, face_names):
    # Scale back up face locations since the frame we detected in was scaled t
    top *= 2
    right *= 2
    bottom *= 2
    left *= 2

    # Draw a box around the face
    cv2.rectangle(video_capture, (left, top), (right, bottom), (0, 0, 255), 2)

    # Draw a label with a name below the face
    cv2.rectangle(video_capture, (left, bottom - 35), (right, bottom), (0, 0,
    font = cv2.FONT_HERSHEY_DUPLEX
    cv2.putText(video_capture, name, (left + 6, bottom - 6), font, 1.0, (255,

# Display the resulting image
cv2.imshow("Image window", video_capture)
cv2.waitKey(1)
```

We then execute this at a certain rate using the **loop** method (5Hz in this case):

In [ ]:

```
def loop(self):

    while not rospy.is_shutdown():
        self.recognise(self.cam_image)
        self.rate.sleep()
```

The result of this should be the following:



This is the final code you should have, including all the parts explained:

▶ Execute in WebShell #1

In [ ]:
```
roscd my_face_recogniser
mkdir scripts
cd scripts
touch recognise_face.py
chmod +x recognise_face.py
```

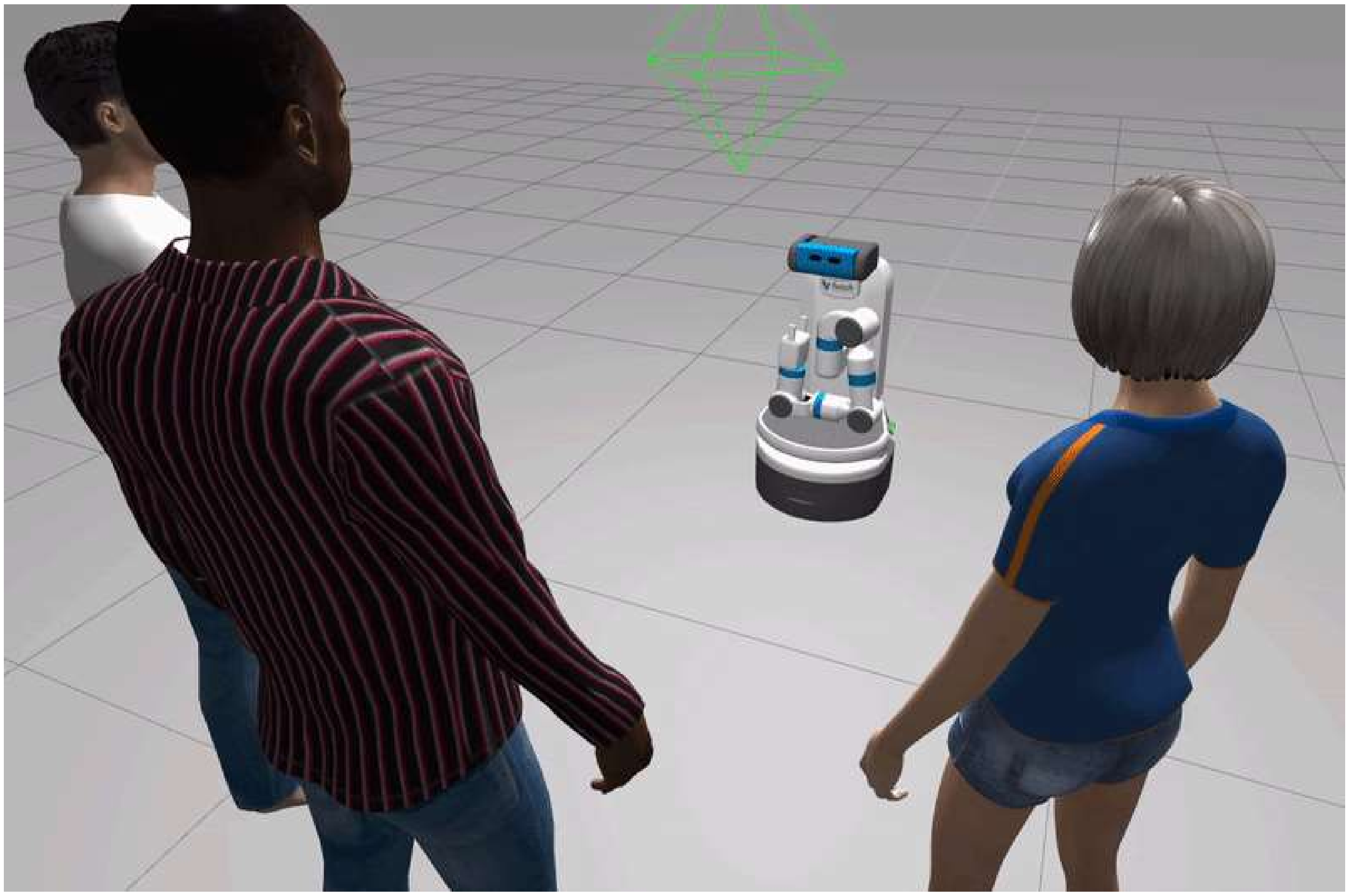
recognise_face.py

In [ ]:
```
#!/usr/bin/env python
import sys
import os
import rospy
from sensor_msgs.msg import Image
import rospkg

try:
    sys.path.remove('/opt/ros/kinetic/lib/python2.7/dist-packages')
except Exception as e:
    print(e)
    print("No path needed to be deleted")

print("Importing cv2")
import cv2
print("Importing cv2..Done")
import face_recognition
from cv_bridge import CvBridge, CvBridgeError

class FaceRecogniser(object):

    def __init__(self):
        rospy.loginfo("Start FaceRecogniser Init process...")
        # get an instance of RosPack with the default search paths
        self.rate = rospy.Rate(5)
        rospack = rospkg.RosPack()
        # get the file path for my_face_recogniser
        self.path_to_package = rospack.get_path('my_face_recogniser')

        self.bridge_object = CvBridge()
        rospy.loginfo("Start camera suscriber...")
        self.cam_topic = "/head_camera/rgb/image_raw"
        self._check_cam_ready()
        self.image_sub = rospy.Subscriber(self.cam_topic,Image,self.camera_cal
        rospy.loginfo("Finished FaceRecogniser Init process...Ready")

    def _check_cam_ready(self):
      self.cam_image = None
      while self.cam_image is None and not rospy.is_shutdown():
         try:
               self.cam_image = rospy.wait_for_message(self.cam_topic, Image,
               rospy.logdebug("Current "+self.cam_topic+" READY=>" + str(self.
            except:
                rospy.logerr("Current "+self.cam_topic+" not ready yet, retryir

    def camera_callback(self,data):
        self.cam_image = data

    def loop(self):

        while not rospy.is_shutdown():
            self.recognise(self.cam_image)
            self.rate.sleep()

    def recognise(self,data):

        # Get a reference to webcam #0 (the default one)
        try:
            # We select bgr8 because its the OpneCV encoding by default
            video_capture = self.bridge_object.imgmsg_to_cv2(data, desired_enc
        except CvBridgeError as e:
            print(e)


        # Load a sample picture and learn how to recognize it.
        image_path = os.path.join(self.path_to_package,"person_img/standing_pe

        standing_person_image = face_recognition.load_image_file(image_path)
        standing_person_face_encoding = face_recognition.face_encodings(standi


        # Initialize some variables
        face_locations = []
        face_encodings = []
        face_names = []

        # Resize frame of video to 1/2 size for faster face recognition proces
        # If this is done be aware that you will have to make the recognition
        # In this case it will work around maximum 1 meter, more it wont work
        small_frame = cv2.resize(video_capture, (0, 0), fx=0.5, fy=0.5)

        #cv2.imshow("SMALL Image window", small_frame)


        # Find all the faces and face encodings in the current frame of video
        face_locations = face_recognition.face_locations(small_frame)
        face_encodings = face_recognition.face_encodings(small_frame, face_loc
        if not face_encodings:
            rospy.logwarn("No Faces found, please get closer...")

        face_names = []
        for face_encoding in face_encodings:
            # See if the face is a match for the known face(s)
            match = face_recognition.compare_faces([standing_person_face_encoc
            name = "Unknown"

            if match[0]:
                rospy.loginfo("MATCH")
                name = "StandingPerson"
            else:
                rospy.logwarn("NO Match")

            face_names.append(name)


        for (top, right, bottom, left), name in zip(face_locations, face_names

            # Scale back up face locations since the frame we detected in was
            top *= 2
            right *= 2
            bottom *= 2
            left *= 2

            # Draw a box around the face
            cv2.rectangle(video_capture, (left, top), (right, bottom), (0, 0,

            # Draw a label with a name below the face
            cv2.rectangle(video_capture, (left, bottom - 35), (right, bottom),
            font = cv2.FONT_HERSHEY_DUPLEX
            cv2.putText(video_capture, name, (left + 6, bottom - 6), font, 1.0

        # Display the resulting image
        cv2.imshow("Image window", video_capture)
        cv2.waitKey(1)




def main():
    rospy.init_node('face_recognising_python_node', anonymous=True, log_level=

    face_recogniser_object = FaceRecogniser()
    face_recogniser_object.loop()
    cv2.destroyAllWindows()


if __name__ == '__main__':
    main()
```

With the help of a **pre-installed script in the system**, move the Fetch robot close to the people and raise its torso, using the following command:

▶ **Execute in WebShell #1**

In [ ]:
```
rosrun face_recognition_tc move_fetch_robot_client.py
```



Now you can execute the **recognise_face.py** script and see how it works. Rotate the Fetch robot so that it can try to recognize all three people in the scene.

▶ **Execute in WebShell #1**

In [ ]:
```
rosrun my_face_recogniser recognise_face.py
```

Rotate the Fetch robot with the keyboard, using the command below:

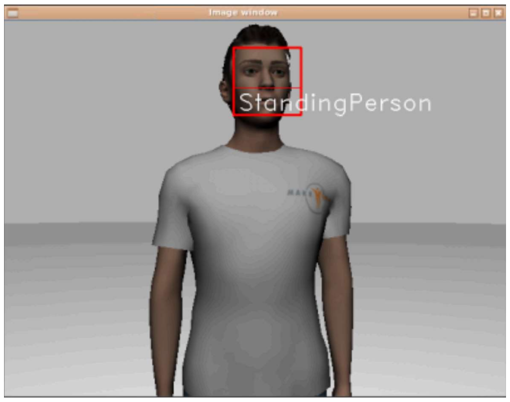
▶ Execute in WebShell #2

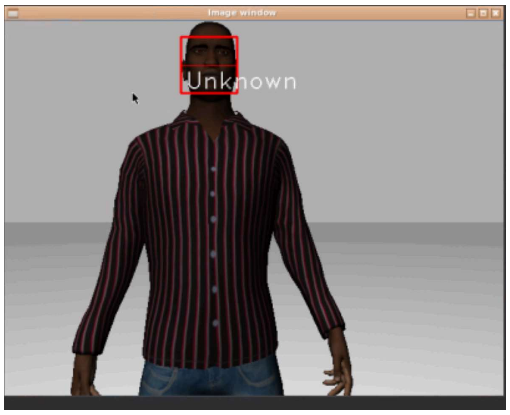
In [ ]:
```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

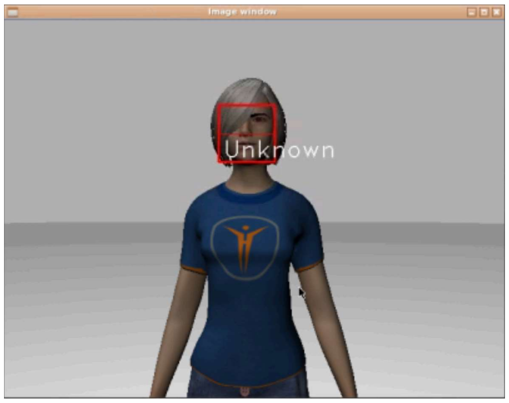
- Once you have checked that it only works with one person (which is called **standing person**), try to recognize the other two people. This means that you take a picture and modify what you think has to be changed so that it now recognizes all three people.

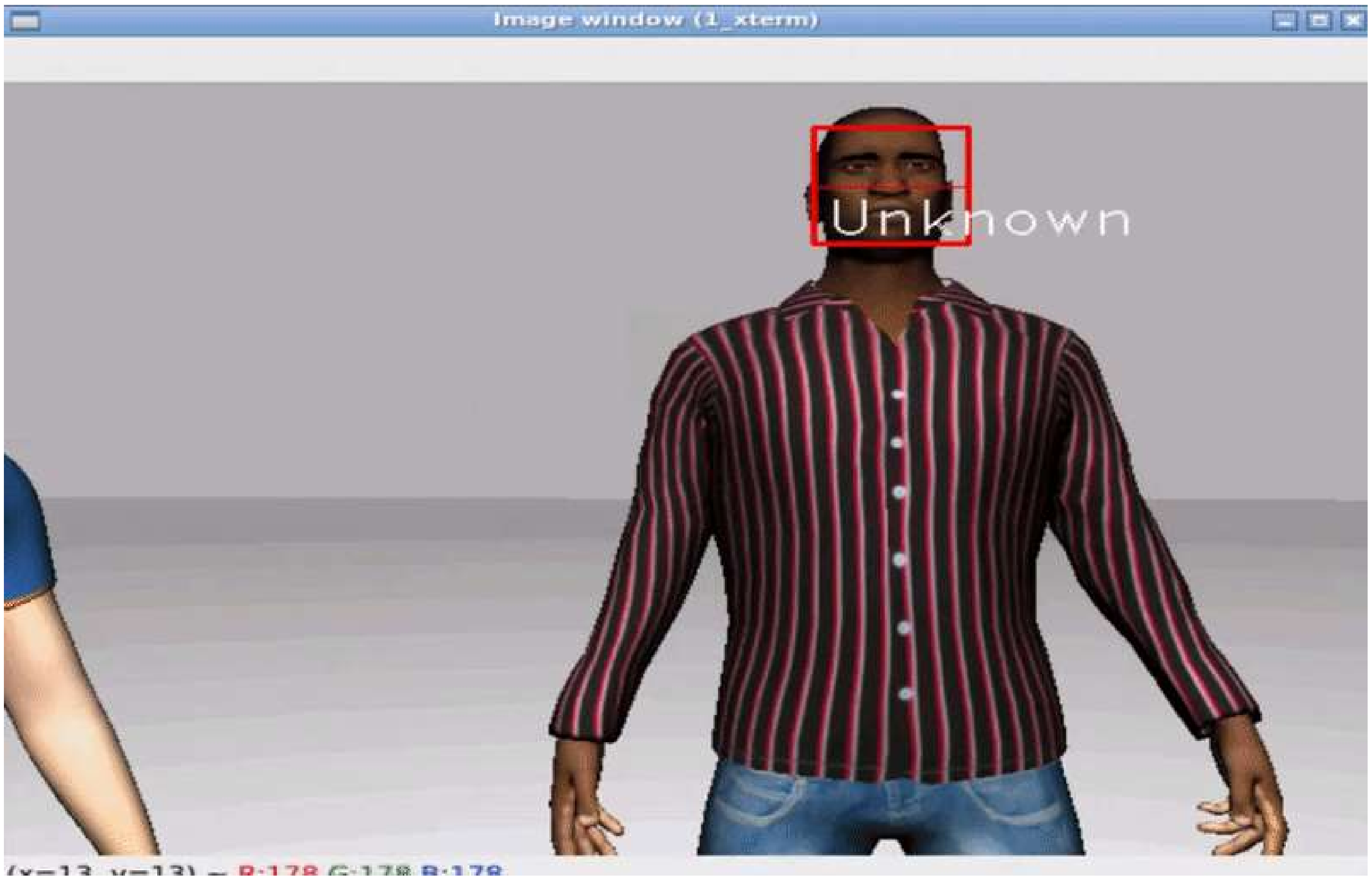
- Remember to click on the Graphical Interface icon to see the detection as an image:

- You will see that only one face has been recognized.









When a face is successfully recognized, you will receive the following message:

In [ ]:
```
[INFO] [WallTime: 1517396015.764194] [432.655000] MATCH
```

When a face is not correctly recognized, you will receive the following message:

In [ ]:
```
[WARN] [WallTime: 1517397392.869854] [606.444000] NO Match
```

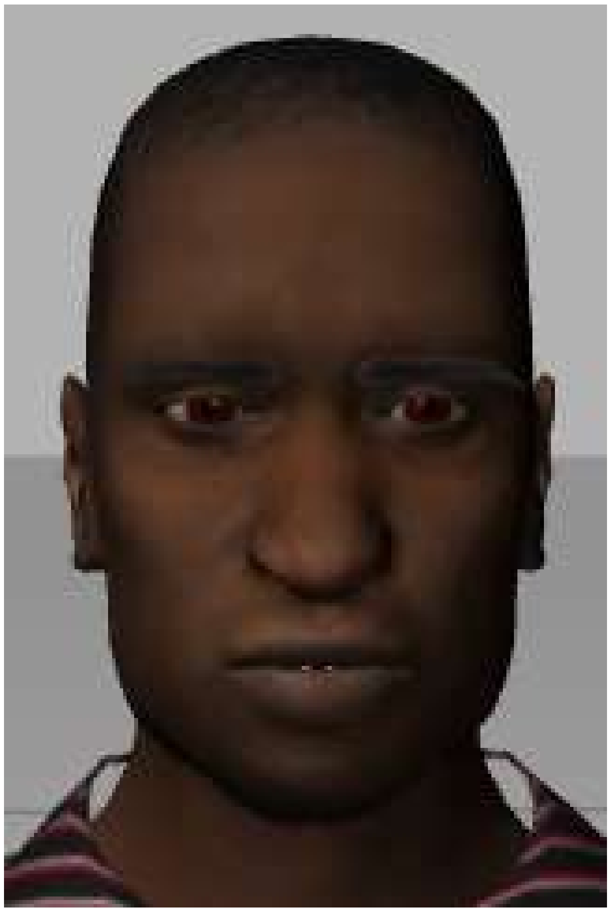
5.3 Multiple Face Detection at the Same Time

- Exercise 2.5.1 -

Do the following:

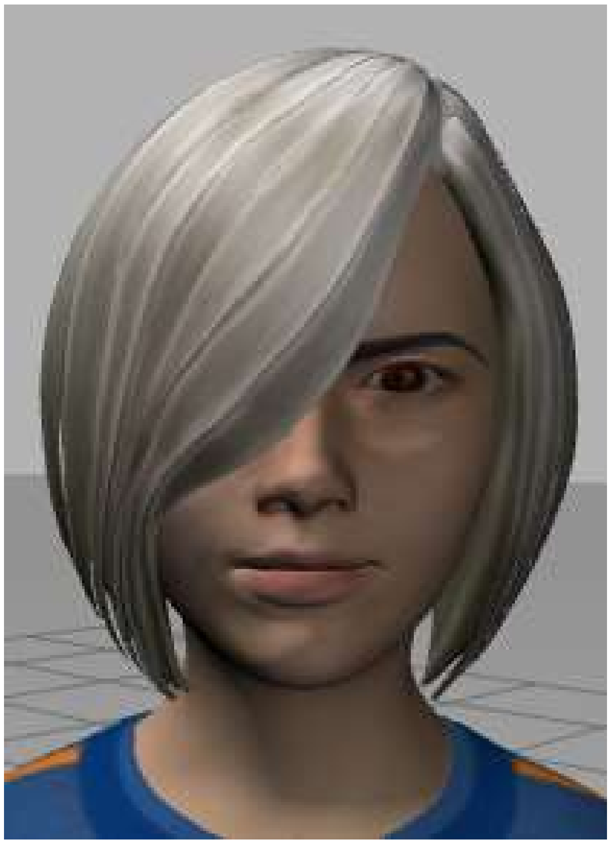
- Create a new script named **recognise_multiple_faces.py** that can recognize all three people at the same time.
- Use the images below for the face recognition. You can store them in the **person_img** folder in your **my_face_recogniser** package.

Bob:



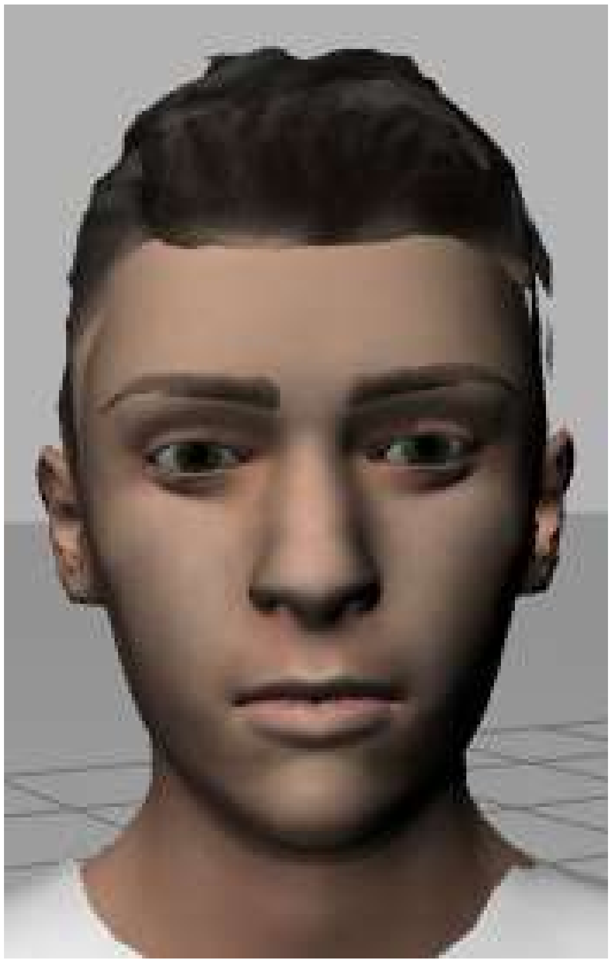
Bob

Naoko:



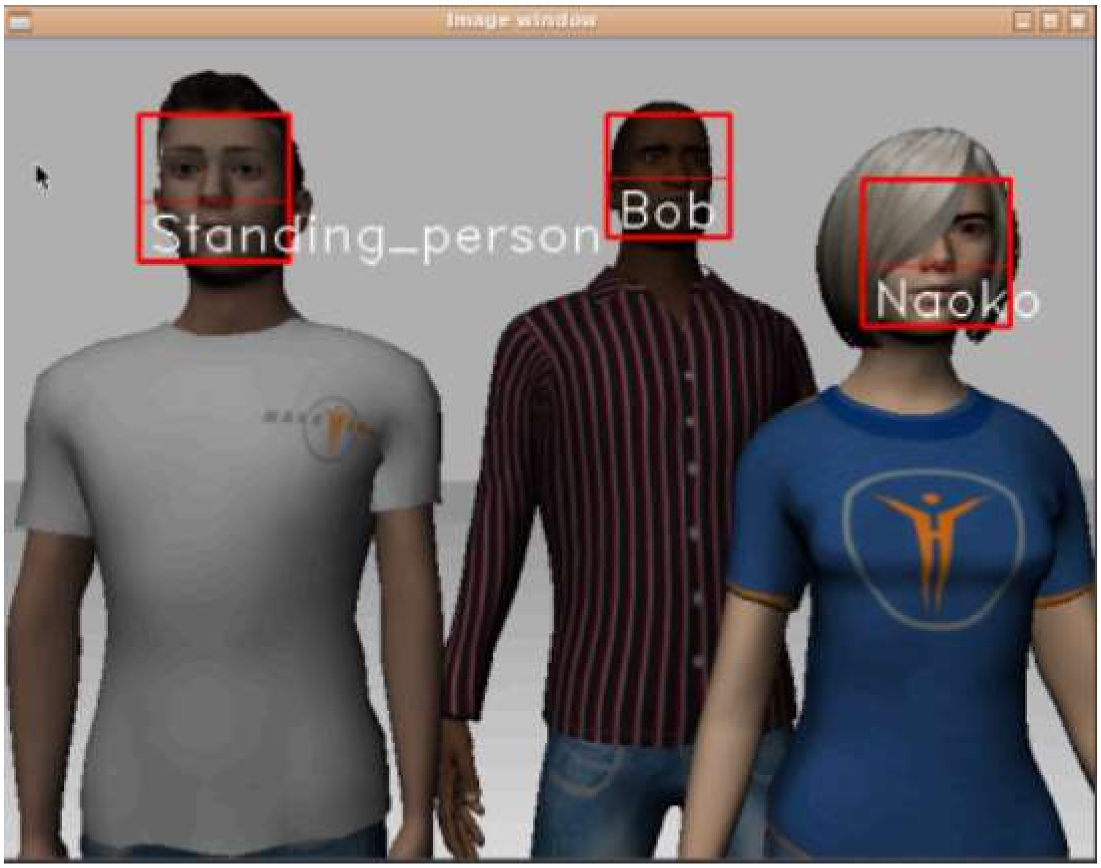
Naoko

Standing person:



Standing Person

- Exercise 2.5.1 -



Project

Select the project unit for this course. You can now do exercise to recognize a person's face and know who it is.

END Project