Patrol Turtlebot Technical Report

Tuan Trinh

Introduction: For this project, I implemented a patrolling robot moving freely in a given space, capturing unknown faces while avoiding obstacles.

Working pitch: I divided the project into two main tasks: capturing unknown faces/recognizing known ones, and the robot's movements. The code file that I have submitted contains 3 python files: *detect_with_CascadeClassifier.py*, *detect_with_ultraLight.py*, and *patrol_robot.py.* Each file is a different milestone of the project.

I. Facial recognition task.

- For the facial recognition task, the logic I implemented was a combination of faces detection, then, comparing detected faces to known faces. So, there were two subtasks: *detection* and *recognition*. For the latter task, I used the face_recognition library on https://github.com/ageitgey/face_recognition.

1. The *face_recognition* library: `import face_recognition`

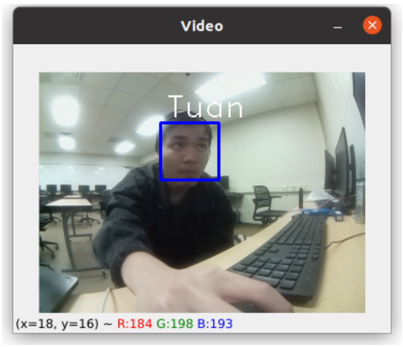
- They have already had example codes containing face detection working with live camera: https://github.com/ageitgey/face_recognition/blob/master/examples/facerec_from_webcam_faster.py
  However, when I tried the sample code on the camera of the turtlebot, the result was laggy and could not detect multiple faces well. Therefore, I had to think of another way to accomplish the subtask of face detection.

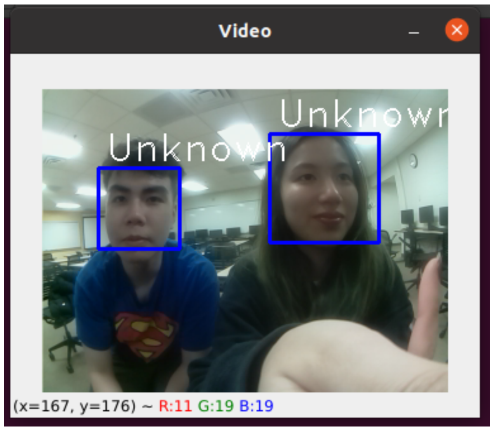
2. Face detection:

a) First attempt with cv2.CascadeClassifier: (detect_with_CascadeClassifier.py)

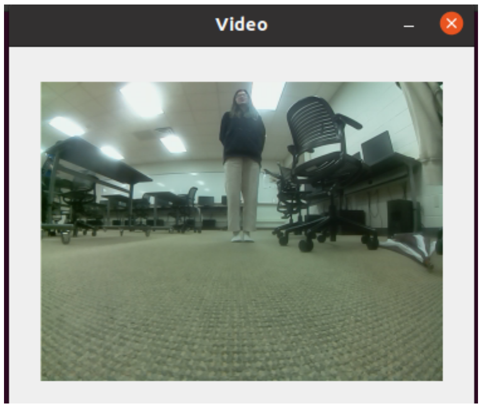
I think back to lab week 7, when we were given code examples for face detection and it worked quite well with detecting multiple faces, and it was not laggy. As a result, I combined face detection with cv2.CascadeClassifier and recognition with face_recognition library.

- Here are the results:
  Facial recognition using detection from cv2.CascadeClassifier





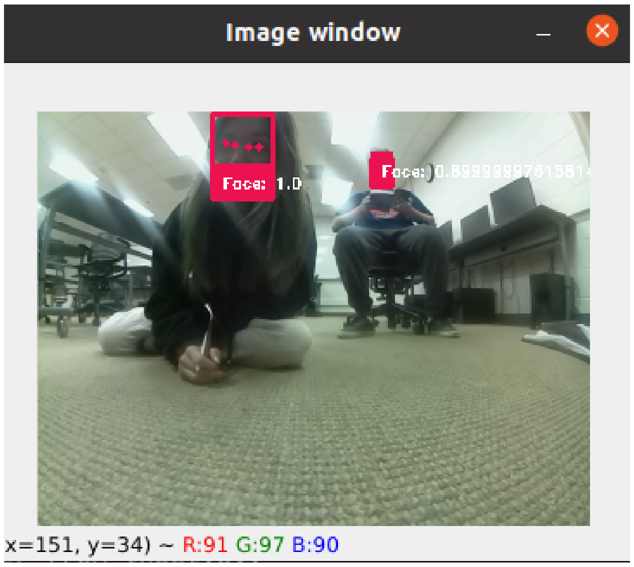
It worked quite well with detecting multiple faces and recognizing known faces. This is when I place the robot on the table, with a close up view of people's faces. However, when I placed the robot on the ground (with a face in the frame), the current face detection module was not able to detect face:



So, at this point, I have to change the face detection module.

b) **<u>Second attempt with the Ultralight Detector module.</u>** **(detect_with_ultraLight.py)**
https://github.com/Linzaer/Ultra-Light-Fast-Generic-Face-Detector-1MB
After I have done some research on face detection algorithms, I found that neural networks are the state of the art. However, most algorithms require a very high runtime, and finally, I was able to find this repository. They used SSD algorithms, which were optimized using ONNX. So, the algorithm was able to run really fast on CPU only, with better performances than the opencv CascadeClassifier.

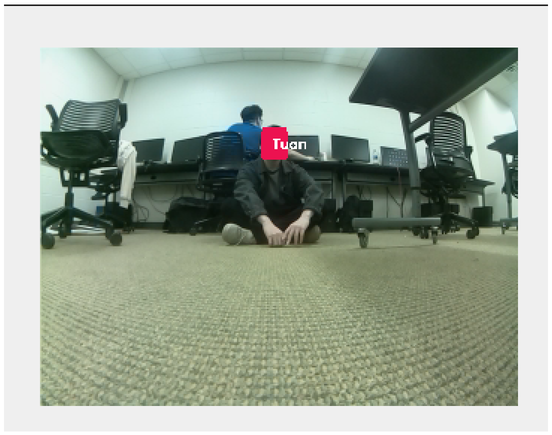
Here were the results of the Ultralight face detection module:

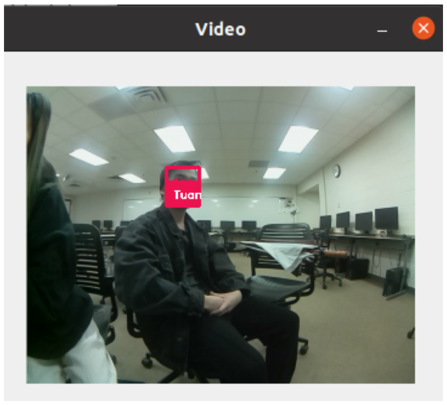


The module was able to detect faces even when a person was standing quite far from the camera.

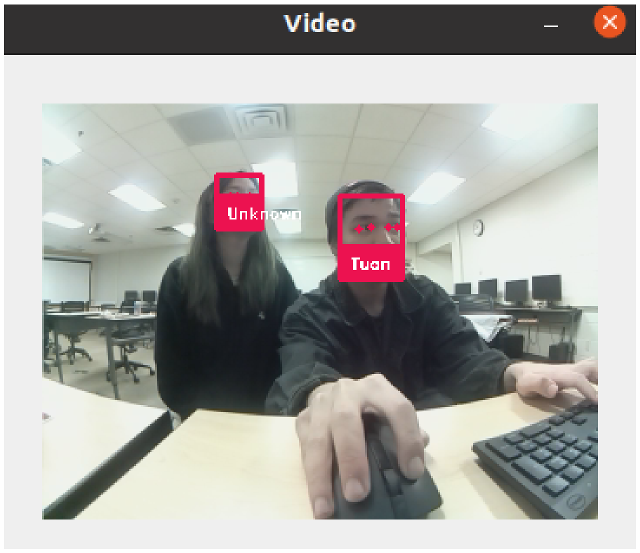
3. **Incorporate face detection with recognition:**
   patrol_robot.py

- The final task is to use the Ultralight face detection module with the face_recognition library.
- As you can see, the robot is now able to detect faces that were far from it, in some different angles.

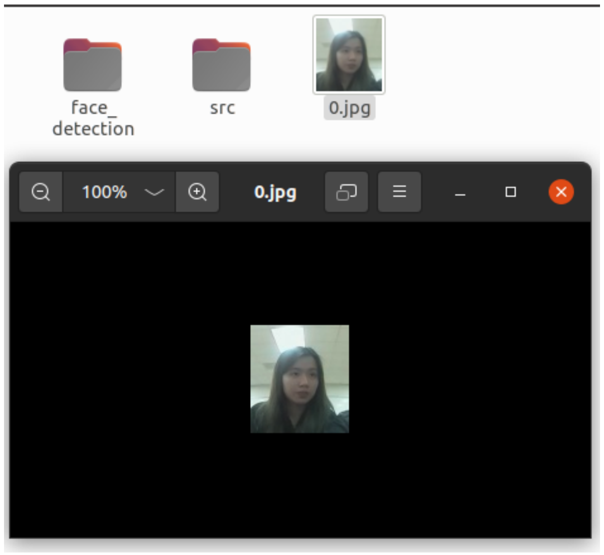




- It is also able to recognize multiple faces.



- I have also implemented a functionality for the robot to capture unknown faces and save them into a directory. I intended for the implementation so that each unknown face would be captured only one time.

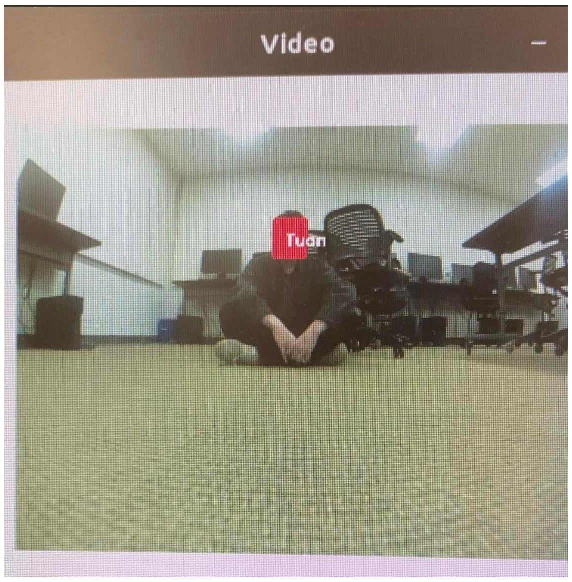


- Lastly, I add a callbackMove function to the robot so that it is able to avoid obstacles while capturing faces.

4. **Limitations of current codes, possible solutions, and future work.**

- Different angles are giving different identity → Solution: add multiple angle for one identity (one identity can have multiple images of multiple angle)

- With the current camera of the turtlebot, it is able to detect faces with the ultralight model when people are standing up. However, it is only able to recognize known faces if a person sits instead of stands in the frame.



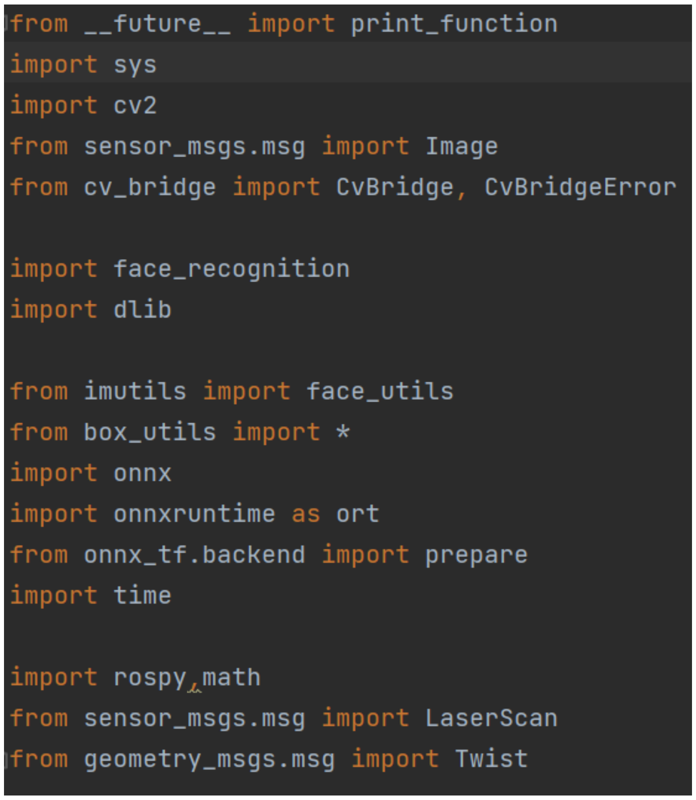
My guess for this problem is with the current facial recognition library used and the turtlebot itself. *face_recognition* is not a well-known library for its performance. Also, the turtlebot is too short, so that the image of a person's face (while he/she is standing in the frame) is too small for recognizing tasks. Thus, in the future, I would try to implement different recognition modules, and hopefully, have chances to work on a different robot with better resources.

II. Moving task.

- In the original idea, I was planning to have the robot move to a list of patrolling points in a mapped area. Those key points in a given space would allow the camera to capture faces' presence in the room. However, while working on the facial recognition task, I met up with problems of the robot not being able to recognize known faces if a person is standing up (as described above). Thus, I changed to a way more simple version with having the robot just move randomly in a given space, avoiding any obstacles with its laser scanner and capturing pictures of unknown faces with its camera.
  ⇒ *Future ideas:* I am looking forward to having the robot challenge unknown people by following them.

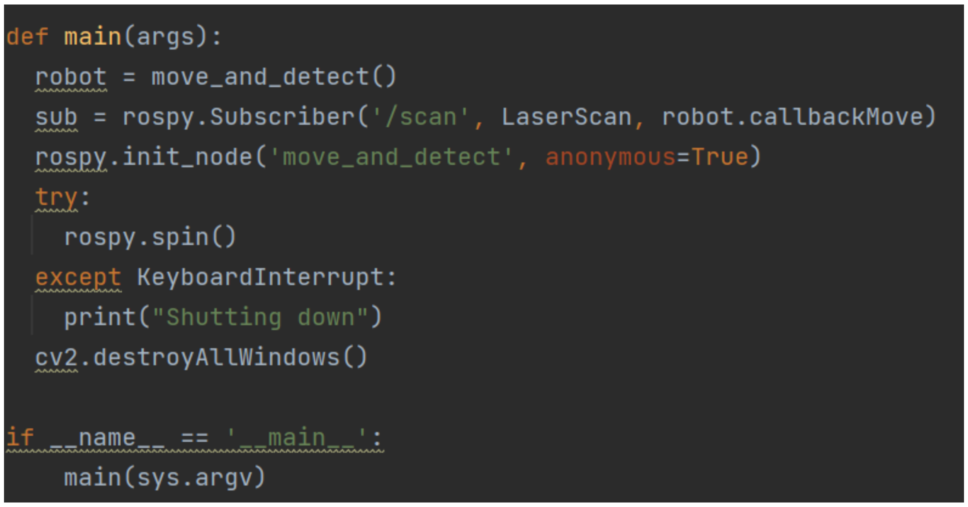
III. Codes explanation:

- I will briefly explain the codes for *patrol_robot.py* as it is the most general file, containing codes and ideas of the last two.



```
from __future__ import print_function
import sys
import cv2
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError

import face_recognition
import dlib

from imutils import face_utils
from box_utils import *
import onnx
import onnxruntime as ort
from onnx_tf.backend import prepare
import time

import rospy,math
from sensor_msgs.msg import LaserScan
from geometry_msgs.msg import Twist
```

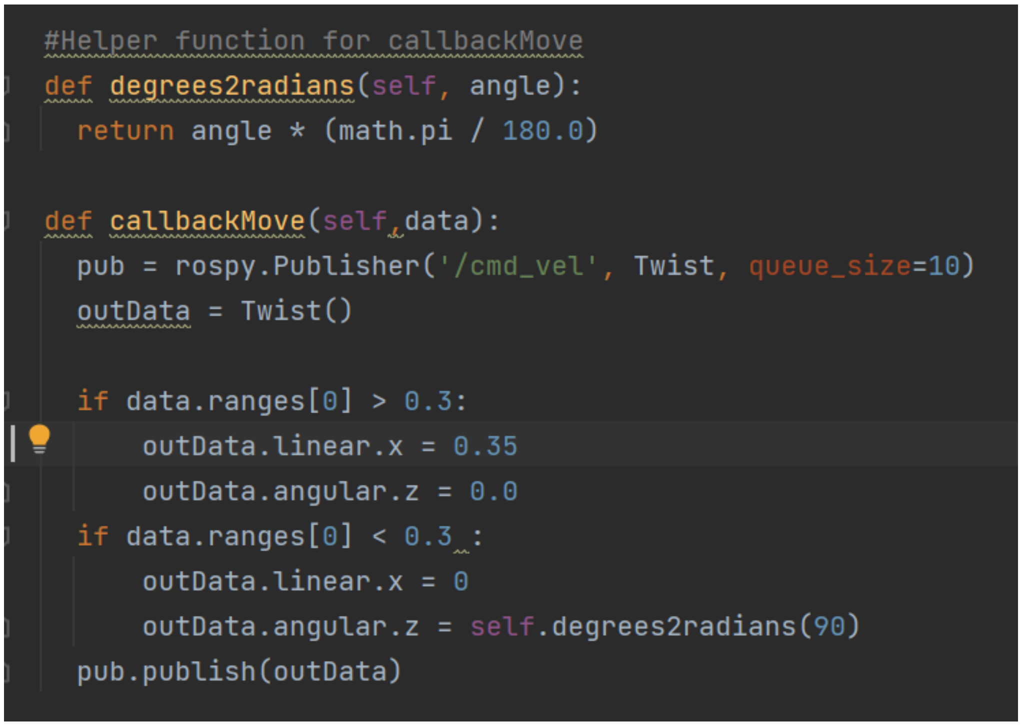
Importing needed packages.



```python
def main(args):
  robot = move_and_detect()
  sub = rospy.Subscriber('/scan', LaserScan, robot.callbackMove)
  rospy.init_node('move_and_detect', anonymous=True)
  try:
    rospy.spin()
  except KeyboardInterrupt:
    print("Shutting down")
  cv2.destroyAllWindows()

if __name__ == '__main__':
    main(sys.argv)
```

The main function to run the robot where sub would run callbackMove for the robot to move and avoid obstacles. At the same time, it would run another callback function for facial processing tasks.

- callbackMove()

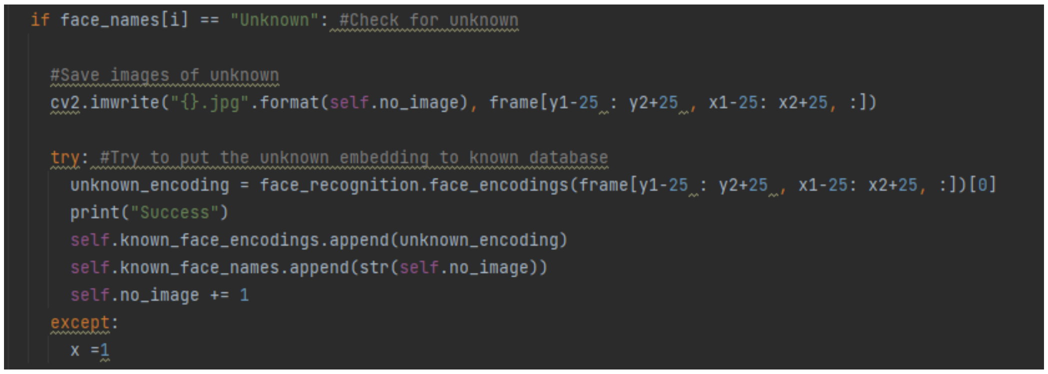


```python
#Helper function for callbackMove
def degrees2radians(self, angle):
  return angle * (math.pi / 180.0)

def callbackMove(self,data):
  pub = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
  outData = Twist()

  if data.ranges[0] > 0.3:
      outData.linear.x = 0.35
      outData.angular.z = 0.0
  if data.ranges[0] < 0.3 :
      outData.linear.x = 0
      outData.angular.z = self.degrees2radians(90)
  pub.publish(outData)
```

Simple algorithms used as in lab 5.

- Callback function for facial recognition task from line 82 to line 169:
    - The algorithms work by firstly, the face recognition model of the face_recognition library will calculate the embedding of the given known images in the database. Then, when footage from the camera is available, the ultralight model would detect faces from frames. Then, the model in the face_recognition library would calculate the embeddings of faces segments detected from camera's footage.

Distance scores will be calculated by comparing the embedding of the camera faces and the faces in the database (either by Euclidean distance of Cosine distance). If the distance between two embeddings is less than a tolerance (here we set the tolerance to be 0.45) then the two faces are assumed to be of the same identity. If the distance is more than the tolerance, the new face in the camera is set as unknown.

- Unknown faces will also be added to the database so that we can count and recognize unknown identities when we reencounter them again.
- 

```
if face_names[i] == "Unknown": #Check for unknown

  #Save images of unknown
  cv2.imwrite("{}.jpg".format(self.no_image), frame[y1-25 : y2+25 , x1-25: x2+25, :])

  try: #Try to put the unknown embedding to known database
    unknown_encoding = face_recognition.face_encodings(frame[y1-25 : y2+25 , x1-25: x2+25, :])[0]
    print("Success")
    self.known_face_encodings.append(unknown_encoding)
    self.known_face_names.append(str(self.no_image))
    self.no_image += 1
  except:
    x =1
```

IV. Demo videos

1. Robot moving and recognizing me: https://drive.google.com/file/d/1NgspxDMFinXuKBo5RMVXaw7L7rzF-v5z/view?usp=sharing
2. Robot avoids before hitting me: https://drive.google.com/file/d/1f0p-BWfWHv8tkDk7K6K0fZjKQmx9ZCVr/view?usp=sharing
3. Robot detects unknown face: https://drive.google.com/file/d/1ADHKJncYVDgyoNvO3RePXJPYPkvXzgbh/view?usp=sharing