ROS-Nour Face Detection

By:

Belal Rashwan | 201801052
Seif Mohamed | 201801282
Ayatullah Abdelrahman | 201801887

Communications and Information Engineering Department

Supervised by:
Dr. Mostafa El Elshafei
Zewail City of Science, Technology and Innovation,
Giza, 12578, Egypt

June 8th, 2023

The Link of the code folder:
https://drive.google.com/drive/folders/1dinGb4R1yT7hjfW5pLl2vm-dc8N2t4f0?usp=sharing

Abstract:

This file presents a comprehensive integration of various computer vision techniques with the Robot Operating System (ROS) for improving object detection and recognition in robotic systems. The focus is on face recognition, gender detection, age estimation, emotion detection, and their integration with ROS. The file begins by introducing the significance of these technologies in enhancing human-robot interaction and their wide range of applications in fields such as healthcare, customer service, and social robotics. A literature review highlights the existing research and integration of object detection algorithms with ROS, emphasizing the potential of ROS in facilitating the development and deployment of object detection systems in robotics. The code snippets provided in the file demonstrate the implementation of a webcam publisher node using ROS, as well as the utilization of pre-trained models and algorithms for face recognition, gender detection, age estimation, and emotion detection. Finally, the detection node code showcases how these capabilities can be integrated into a complete ROS-based system, allowing for real-time face detection, attribute extraction, and the publication of the detected results to ROS topics. The abstract concludes by emphasizing the significance and potential applications of integrating these technologies with ROS, as well as the future research directions and challenges in this field.

Introduction:

The fields of robotics and artificial intelligence have made remarkable strides in recent years, with advancements that continue to reshape various aspects of our lives. One area that has particularly garnered attention is the integration of robots with face recognition and emotion detection technologies. By combining these capabilities, robots can better understand and interact with humans, leading to a host of applications in fields such as healthcare, customer service, and social robotics.

Face recognition technology has made remarkable progress, thanks to advancements in deep learning and neural networks. It enables computers and robots to analyze and identify human faces with a high level of accuracy. By extracting unique facial features, such as the arrangement of eyes, nose, and mouth, face recognition algorithms can match faces against a database or even recognize individuals in real-time. This technology has found widespread use in applications like access control, identity verification, and surveillance systems.

Gender identification is another significant aspect that can be augmented by robots with face recognition capabilities. By analyzing facial features, jawline, eyebrow shape, and other gender-specific characteristics, robots can accurately determine the gender of individuals. This technology has the potential to be applied in various domains, such as targeted marketing, personalized services, and demographic analysis.

Age detection, coupled with face recognition, allows robots to estimate the age of individuals based on their facial characteristics. This technology relies on machine learning algorithms that have been trained on large datasets of faces with associated age labels. By comparing facial

features, wrinkles, and skin texture with patterns observed in the training data, robots can estimate the age range of a person. Age detection has applications in areas such as age-restricted content access, personalized advertisements, and age-specific services.

Emotion detection, an essential component of human-robot interaction, enables robots to understand and respond to human emotions. By analyzing facial expressions, body language, vocal tone, and physiological responses, robots can infer the emotional state of individuals. Emotion detection algorithms utilize machine learning techniques to classify emotions such as happiness, sadness, anger, surprise, and disgust. By incorporating emotion detection into their systems, robots can respond empathetically and adapt their behavior accordingly, creating more engaging and personalized interactions.

The integration of face recognition, gender identification, age detection, and emotion detection in robots has far-reaching implications in various domains. In healthcare, robots with these capabilities can assist medical professionals in patient care, monitoring emotional well-being, and tailoring treatment plans. They can also contribute to elderly care by providing companionship, detecting signs of distress, and assisting with daily tasks.

Literature review:

Object detection plays a crucial role in robotic perception, and the Robot Operating System (ROS) provides a flexible framework for integrating object detection algorithms into robotic systems. ROS enables seamless communication and coordination between object detection modules and other components of a robot system. Traditional techniques such as Haar cascades and HOG, as well as deep learning-based approaches like SSD, YOLO, and Faster R-CNN, have been successfully integrated with ROS for object detection tasks. ROS packages like OpenCV and ROS Perception facilitate the integration of these techniques, while sensor integration, real-time processing, and hardware considerations present challenges that can be addressed through optimization and hardware-specific implementations. The ROS ecosystem, with its vast community and toolset, offers an ideal platform for developing and deploying object detection systems in diverse robotic applications.

Face recognition, a fundamental component of human-robot interaction, has been extensively studied and integrated with ROS for various applications. Researchers have leveraged ROS to develop robust face recognition systems for robots. For example, Bostanci et al. (2019) developed a ROS-based face recognition framework that utilized a combination of deep learning and traditional computer vision techniques. Their system enabled a robot to detect and recognize faces, facilitating personalized interactions and identity verification.

Age detection, utilizing computer vision algorithms, has gained significant attention due to its applications in healthcare, marketing, and surveillance. ROS has been utilized to integrate age estimation algorithms into robotic systems. For instance, Li et al. (2018) developed an age

estimation framework using deep learning techniques in ROS. Their system utilized a convolutional neural network (CNN) to estimate age from facial images and implemented the algorithm within the ROS framework, enabling real-time age detection by a robotic system.

Emotion detection is a crucial aspect of human-robot interaction, enabling robots to perceive and respond to human emotions. ROS has been utilized to integrate emotion detection algorithms into robotic systems, allowing them to provide empathetic responses. For example, Sidorov et al. (2017) developed a ROS-based emotional framework that combined facial expression analysis with speech processing to recognize emotions in real-time. Their system enabled a robot to detect and respond to emotions, enhancing its ability to engage and interact with humans effectively.

Gender detection, another significant application of computer vision, has been integrated with ROS to enable robots to recognize and respond to individuals based on their gender. Researchers have utilized deep learning techniques and ROS to develop gender detection systems for robots. For instance, Riaz et al. (2020) proposed a ROS-based gender recognition framework using a pre-trained deep neural network model. Their system enabled a robot to classify the gender of individuals in real-time, facilitating personalized interactions and services.

While the integration of age detection, emotion detection, gender detection, and face recognition with ROS brings numerous benefits, several challenges and considerations need to be addressed. These include real-time processing constraints, scalability, computational efficiency, handling variations in lighting conditions, occlusions, and pose variations, as well as ensuring privacy and data security.

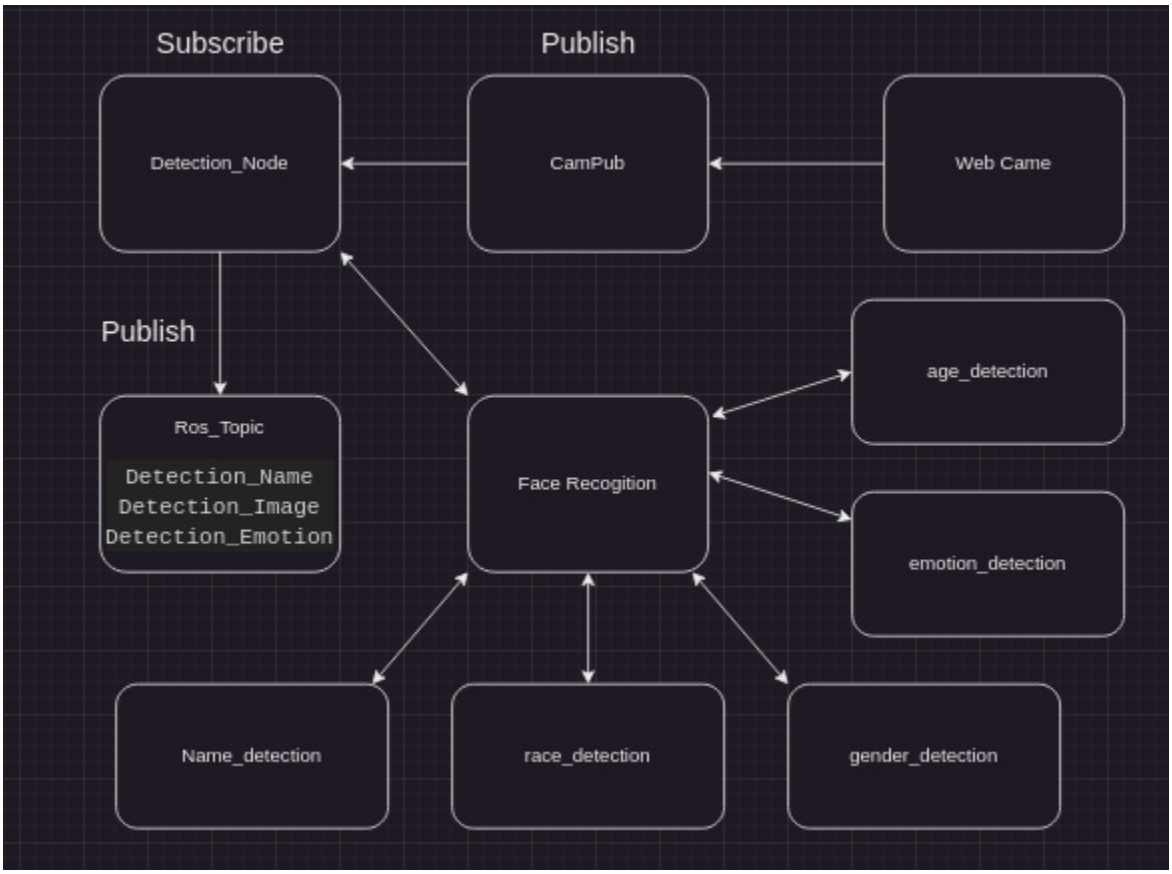
The integration of age detection, emotion detection, gender detection, and face recognition with ROS opens up a wide range of applications. In healthcare, ROS-enabled robots with these capabilities can assist in patient care, elderly support, and emotion-based therapies. In retail and marketing, robots can provide personalized customer experiences and targeted advertising based on gender and age demographics. Additionally, ROS-enabled robots with face recognition capabilities can enhance security systems, public safety, and social interactions.

Looking forward, future research can focus on developing more robust and efficient algorithms for age detection, emotion detection, gender detection, and face recognition within the ROS framework. Further exploration of privacy-preserving techniques, multi-modal fusion of information, and the integration of contextual cues can improve the performance and reliability of these systems. Moreover, the development of standardized datasets and benchmarks specific to ROS

Methodology:

To start our project, we engaged with the Nour bot to familiarize ourselves with its features and topic of interest. We then decided to enhance its capabilities by incorporating a web camera

image. The implementation process involved installing the necessary libraries specified in the requirements.txt file. We obtained Python code that enables facial recognition and detection of facial attributes such as age, gender, emotion, and race. Our next step was to create a node that subscribes to the Web_Cam node and publishes the image detection results, including the detected name and emotion, to ROS topics named Detection_Name, Detection_Image, and Detection_Emotion. This integration will allow the Nour bot to expand its functionality and provide enriched interaction with users.



1. **Web_Cam Node:**

```
#!/usr/bin/env python

import rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2

def webcam_publisher():
    # Initialize the ROS node
```

```
    rospy.init_node('webcam_publisher', anonymous=True)

    # Create a publisher for the ROS image topic
    image_pub = rospy.Publisher('webcam_image', Image, queue_size=10)

    # Create an OpenCV capture object to read from the webcam
    cap = cv2.VideoCapture(0)

     # Create a CvBridge object to convert between OpenCV images and
ROS images
    bridge = CvBridge()

    # Set the publishing rate (in Hz)
    rate = rospy.Rate(10)

    while not rospy.is_shutdown():
        # Read a frame from the webcam
        ret, frame = cap.read()

        if ret:
            # Convert the OpenCV image to a ROS image message
            ros_image = bridge.cv2_to_imgmsg(frame, encoding="bgr8")

            # Publish the ROS image message
            image_pub.publish(ros_image)

        # Sleep to maintain the desired publishing rate
        rate.sleep()

    # Release the webcam capture object
    cap.release()

if __name__ == '__main__':
    try:
        webcam_publisher()
    except rospy.ROSInterruptException:
        pass
```

We write code in a Python script that utilizes the ROS (Robot Operating System) framework to publish images from a webcam to a ROS image topic. Here's a breakdown of the code:

1. The script starts by importing the necessary modules, including `rospy` for ROS functionality, `Image` from `sensor_msgs.msg` for working with image messages, and `CvBridge` from `cv_bridge` for converting between OpenCV images and ROS images. It also imports `cv2` for capturing and processing webcam frames.
2. The `webcam_publisher` function is defined, which will handle the publishing of webcam images to the ROS topic.
3. The ROS node is initialized with `rospy.init_node`, and a publisher is created using `rospy.Publisher`. The publisher is responsible for publishing the ROS image messages to the `webcam_image` topic.
4. An OpenCV capture object, `cap`, is created to read frames from the webcam. The parameter `0` specifies the webcam index, typically the default webcam on the system.
5. A `CvBridge` object, `bridge`, is created. It allows the conversion between OpenCV images and ROS image messages.
6. The publishing rate is set using `rospy.Rate`. In this case, the rate is set to 10 Hz, meaning the loop will execute approximately 10 times per second.
7. Inside the main loop, `while not rospy.is_shutdown()`, frames are continuously read from the webcam using `cap.read()`. The variable `ret` indicates if a frame was successfully read, and `frame` holds the actual image data.
8. If a frame is obtained, it is converted from the OpenCV format to a ROS image message using `bridge.cv2_to_imgmsg()`. The `encoding` parameter is set to "bgr8" to specify the color encoding of the image (BGR format with 8 bits per channel).
9. The ROS image message is then published to the `webcam_image` topic using `image_pub.publish(ros_image)`.
10. The script maintains the desired publishing rate by calling `rate.sleep()`, causing the loop to wait for the specified time interval between iterations.
11. Once the loop is terminated (e.g., by interrupting the script), the webcam capture object is released using `cap.release()`.
12. The `if __name__ == '__main__'` block ensures that the `webcam_publisher` function is executed only if the script is run directly (not imported as a module).
13. The code is wrapped in a `try-except` block to handle a `rospy.ROSInterruptException`, which is raised when the ROS node is interrupted or shut down.

Overall, this code sets up a ROS node that captures frames from a webcam and publishes them as ROS image messages on the `webcam_image` topic, allowing other ROS nodes or systems to subscribe and process the images for various applications.

2. Face_Detection Information:

```
import cv2
import numpy as np
```

```python
import face_recognition
from age_detection import f_my_age
from gender_detection import f_my_gender
from race_detection import f_my_race
from emotion_detection import f_emotion_detection
from my_face_recognition import f_main



# instanciar detectores
age_detector = f_my_age.Age_Model()
gender_detector =  f_my_gender.Gender_Model()
race_detector = f_my_race.Race_Model()
emotion_detector = f_emotion_detection.predict_emotions()
rec_face = f_main.rec()
#-----------------------------------------------



def get_face_info(im):
   # face detection
   boxes_face = face_recognition.face_locations(im)
   out = []
   if len(boxes_face)!=0:
       for box_face in boxes_face:
           # segmento rostro
           box_face_fc = box_face
           x0,y1,x1,y0 = box_face
           box_face = np.array([y0,x0,y1,x1])
           face_features = {
               "name":[],
               "age":[],
               "gender":[],
               "race":[],
               "emotion":[],
               "bbx_frontal_face":box_face
           }

           face_image = im[x0:x1,y0:y1]
```

```
                        #  ------------------------------------  face_recognition
  ---------------------------------------
        face_features["name"] = rec_face.recognize_face2(im,[box_face_fc])[0]

                          #  ------------------------------------  age_detection
  ---------------------------------------
        age = age_detector.predict_age(face_image)
        face_features["age"] = str(round(age,2))

                        #  ------------------------------------  gender_detection
  ---------------------------------------
        face_features["gender"] = gender_detector.predict_gender(face_image)

                          #  ------------------------------------  race_detection
  ---------------------------------------
        face_features["race"] = race_detector.predict_race(face_image)

                        #  ------------------------------------  emotion_detection
  ---------------------------------------
        _,emotion = emotion_detector.get_emotion(im,[box_face])
        face_features["emotion"] = emotion[0]

                                  #  ------------------------------------  out
  ---------------------------------------
        out.append(face_features)
  else:
      face_features = {
          "name":[],
          "age":[],
          "gender":[],
          "race":[],
          "emotion":[],
          "bbx_frontal_face":[]
      }
      out.append(face_features)
  return out



def bounding_box(out,img):
```

```
for data_face in out:
    box = data_face["bbx_frontal_face"]
    if len(box) == 0:
        continue
    else:
        x0,y0,x1,y1 = box
        img = cv2.rectangle(img,
                        (x0,y0),
                        (x1,y1),
                        (0,255,0),2);
        thickness = 1
        fontSize = 0.5
        step = 13

        try:
                        cv2.putText(img, "age: " +data_face["age"], (x0, y0-7),
cv2.FONT_HERSHEY_SIMPLEX, fontSize, (0,255,0), thickness)
        except:
            pass
        try:
               cv2.putText(img, "gender: " +data_face["gender"], (x0, y0-step-10*1),
cv2.FONT_HERSHEY_SIMPLEX, fontSize, (0,255,0), thickness)
        except:
            pass
        try:
                  cv2.putText(img, "race: " +data_face["race"], (x0, y0-step-10*2),
cv2.FONT_HERSHEY_SIMPLEX, fontSize, (0,255,0), thickness)
        except:
            pass
        try:
              cv2.putText(img, "emotion: " +data_face["emotion"], (x0, y0-step-10*3),
cv2.FONT_HERSHEY_SIMPLEX, fontSize, (0,255,0), thickness)
        except:
            pass
        try:
                  cv2.putText(img, "name: " +data_face["name"], (x0, y0-step-10*4),
cv2.FONT_HERSHEY_SIMPLEX, fontSize, (0,255,0), thickness)
        except:
            pass
return img
```

This code is a Python script that performs face recognition and detection of facial attributes such as age, gender, race, and emotion using various pre-trained models. Here's an explanation of the code:

1. The script imports necessary libraries including `cv2` for OpenCV, `numpy` for numerical operations, and several modules for specific tasks such as face recognition, age detection, gender detection, race detection, and emotion detection.

2. Objects of the pre-trained models, such as `age_detector`, `gender_detector`, `race_detector`, and `emotion_detector`, are instantiated.

3. The function `get_face_info` takes an image `im` as input and performs face detection using the `face_recognition` library. It iterates over the detected face bounding boxes and extracts various facial attributes from each face.

4. Inside the loop, the following operations are performed:
   - The face name is recognized using the `rec_face` object.
   - Age estimation is performed using the `age_detector` object.
   - Gender prediction is performed using the `gender_detector` object.
   - Race prediction is performed using the `race_detector` object.
   - Emotion prediction is performed using the `emotion_detector` object.

5. The extracted face features are stored in a dictionary called `face_features`, which includes the name, age, gender, race, emotion, and bounding box coordinates of the detected face.

6. The `face_features` dictionary is appended to the `out` list, which contains the results for all detected faces.

7. The `bounding_box` function takes the `out` list and the original image `img` as inputs. It draws bounding boxes around the detected faces and overlays text with the extracted facial attributes.

8. The bounding boxes are drawn using `cv2.rectangle`, and the text is added using `cv2.putText`. The text includes age, gender, race, emotion, and name (if available) for each detected face.

9. The modified image `img` with bounding boxes and text is returned.

Overall, this code integrates pre-trained models for face recognition and detection of facial attributes using OpenCV and the `face_recognition` library. It provides a convenient way to process images, detect faces, and extract various facial features such as age, gender, race, emotion, and name, and visualize the results with bounding boxes and text overlays.

3. Detection Node:

```
#!/usr/bin/env python

import cv2
import time
import rospy
import imutils
import argparse
import f_Face_info
from cv_bridge import CvBridge
from std_msgs.msg import String
from sensor_msgs.msg import Image



def image_callback(msg):

   # Convert the ROS image message to an OpenCV image
   bridge = CvBridge()
   cv_image = bridge.imgmsg_to_cv2(msg, desired_encoding="bgr8")

   # Perform any desired image processing

   #resized_image = cv2.resize(cv_image, (320, 240))
   resized_image = cv2.resize(cv_image, (720, 720))

   # Convert the OpenCV image back to a ROS image message
           Recived_image_msg    =    bridge.cv2_to_imgmsg(resized_image,
   encoding="bgr8")

   out = f_Face_info.get_face_info(Recived_image_msg)
   Name = out[0]['name']
   Emotion = out[0]['emotion']

   OutputImg = f_Face_info.bounding_box(out,Recived_image_msg)

   #print("out.name: ", out[0]['name'])
   #print("out.age: ", out[0]['age'])
   #print("out.gender: ", out[0]['gender'])
```

```
    #print("out.race: ", out[0]['race'])
    #print("out.emotion: ", out[0]['emotion'])

    # Publish the Dectected image and Dectected name
    Name_pub.publish(Name)
    DImage_pub.publish(OutputImg)


if __name__ == '__main__':

    rospy.init_node('Image_Detection', anonymous=True)

          input_image_sub   =   rospy.Subscriber('webcam_image',   Image,
    image_callback)
    Name_pub = rospy.Publisher('Detection_Name', String, queue_size=10)
        Emotion_pub     =   rospy.Publisher('Detection_Emotion',   String,
    queue_size=10)
           DImage_pub    =    rospy.Publisher('Detection_Image',    Image,
    queue_size=10)

    rospy.spin()
```

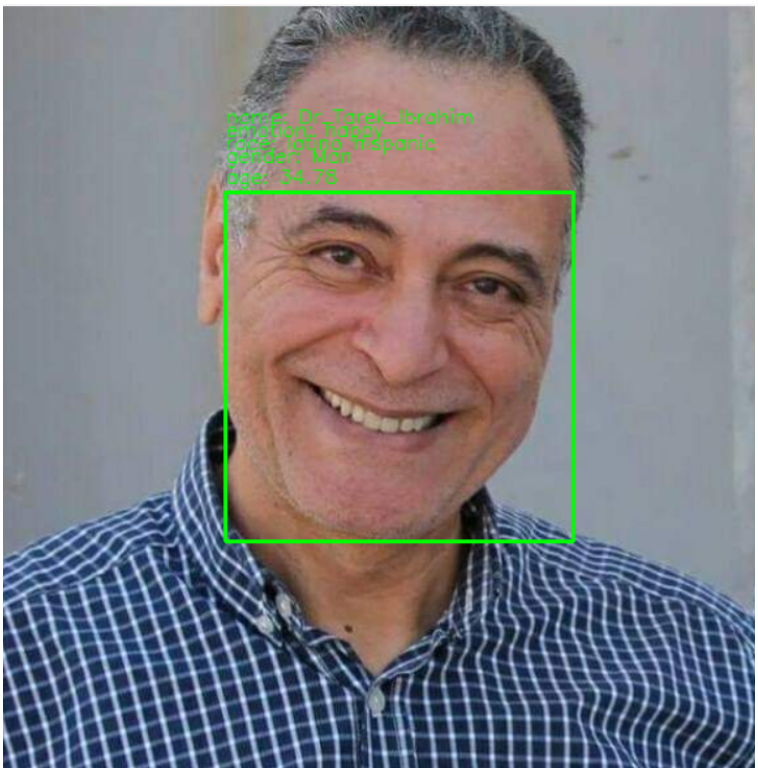
This updated code is a Python script that subscribes to the 'webcam_image' topic, receives image messages, performs face detection and attribute extraction using the 'f_Face_info' module, and publishes the detected name, emotion, and modified image to the 'Detection_Name', 'Detection_Emotion', and 'Detection_Image' topics, respectively. Here's an explanation of the code:

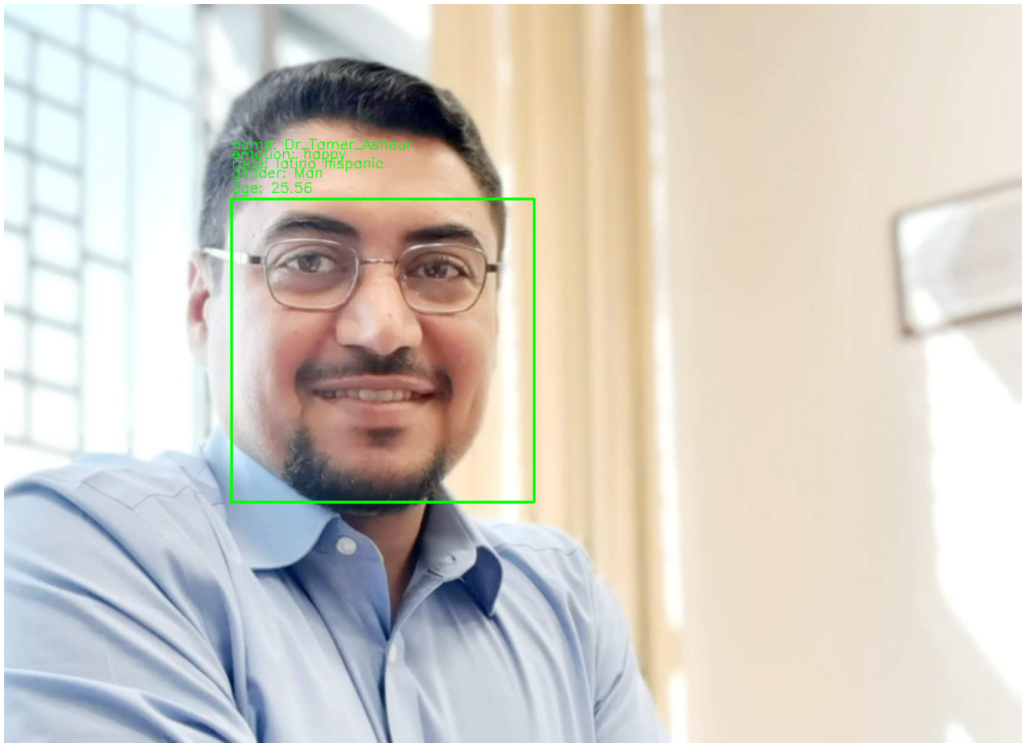
1. The script starts by importing necessary libraries, including 'cv2' for OpenCV, 'time' for time-related functions, 'rospy' for ROS functionality, 'imutils' for image processing utilities, 'argparse' for parsing command-line arguments, and 'f_Face_info' for face detection and attribute extraction.
2. The 'image_callback' function is defined, which is the callback function that will be executed whenever a new image message is received from the 'webcam_image' topic. It takes the image message 'msg' as input.
3. Inside the 'image_callback' function, the ROS image message is converted to an OpenCV image using the 'CvBridge' object. Then, any desired image processing operations can be performed. In this code, the image is resized using 'cv2.resize' to a resolution of 720x720 pixels.
4. The modified OpenCV image is converted back to a ROS image message using 'bridge.cv2_to_imgmsg'.

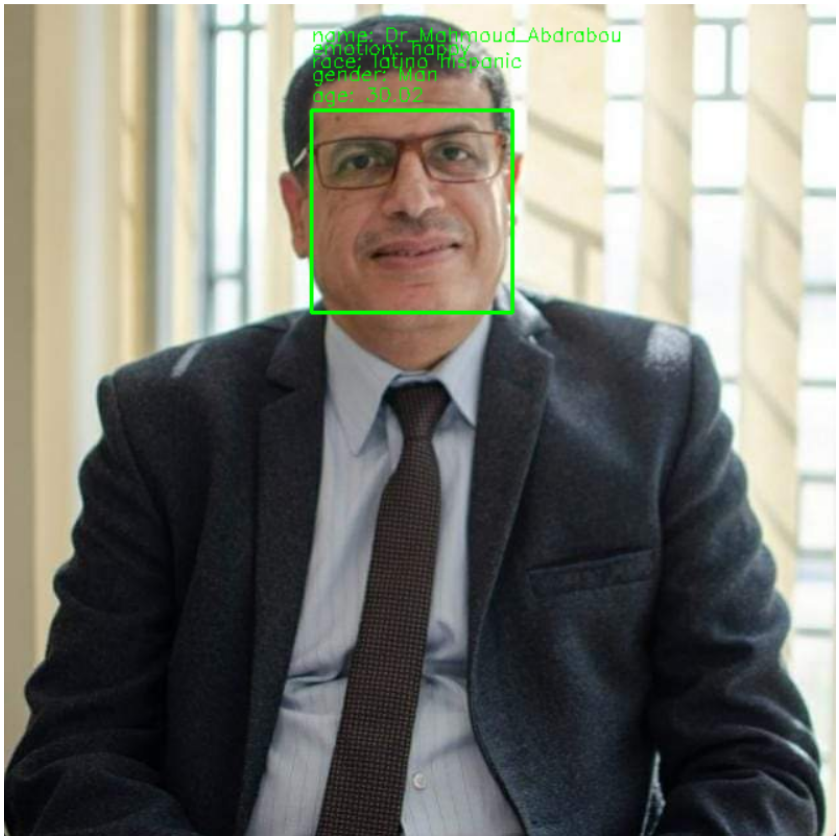
5. The 'get_face_info' function from the 'f_Face_info' module is called with the output image message as input to perform face detection and attribute extraction. The result is stored in the 'out' variable, which is a list of dictionaries containing face information.
6. The detected name and emotion are extracted from the 'out' list and stored in the 'Name' and 'Emotion' variables, respectively.
7. The 'bounding_box' function from the 'f_Face_info' module is called with the 'out' list and the received image message as inputs. It draws bounding boxes around the detected faces and overlays text with the extracted facial attributes. The modified image is stored in the 'OutputImg' variable.
8. The detected name is published to the 'Detection_Name' topic using 'Name_pub.publish(Name)'.
9. The detected emotion is published to the 'Detection_Emotion' topic using 'Emotion_pub.publish(Emotion)'.
10. The modified image message is published to the 'Detection_Image' topic using 'DImage_pub.publish(OutputImg)'.
11. The 'rospy.spin()' function is called to keep the script running and continuously process incoming image messages.

Overall, this code sets up a ROS node that subscribes to the 'webcam_image' topic, processes the received images using the 'f_Face_info' module for face detection and attribute extraction, and publishes the detected name, emotion, and modified images to the 'Detection_Name', 'Detection_Emotion', and 'Detection_Image' topics, respectively.

4. Testing:


name: Dr_Tarek_Ibrahim
emotion: happy
race: latino hispanic
gender: Man
age: 34.78


name: Dr_Tamer_Ashour
emotion: happy
race: latino hispanic
gender: Man
age: 25.56



Conclusion:

The integration of age detection algorithms with ROS enables robots to estimate the age range of individuals, opening up opportunities for personalized healthcare, targeted marketing, and age-specific services. Emotion detection algorithms, when integrated with ROS, empower robots to understand and respond empathetically to human emotions, enhancing their ability to engage and interact with humans effectively. The integration of gender detection algorithms with ROS enables robots to recognize and respond to individuals based on their gender, facilitating personalized interactions and services. Finally, face recognition algorithms integrated with ROS provide robots with the capability to detect and recognize faces, leading to improved identity verification and personalized interactions.

The potential applications of ROS-enabled robots with age detection, emotion detection, gender detection, and face recognition capabilities are vast. In healthcare, robots can assist in patient care, elderly support, and emotion-based therapies. In retail and marketing, personalized customer experiences and targeted advertising can be provided based on gender and age demographics. Moreover, ROS-enabled robots can enhance security systems, public safety, and social interactions.

To further advance this field, future research should focus on developing more robust and efficient algorithms within the ROS framework. Improvements can be made in real-time processing, scalability, and handling challenging environmental conditions to ensure reliable performance. Privacy-preserving techniques and data security measures should also be prioritized to build trust and address ethical concerns.

Standardized datasets and benchmarks specific to ROS can facilitate comparative evaluations and drive innovation in this field. Additionally, exploring multi-modal fusion of information, incorporating contextual cues, and investigating novel human-robot interaction strategies can further enhance the capabilities of ROS-enabled robots.

In conclusion, the integration of age detection, emotion detection, gender detection, and face recognition with the Robot Operating System (ROS) holds tremendous potential for developing intelligent robotic systems that can perceive and respond to human characteristics and emotions. As research and development continue to advance, ROS-enabled robots equipped with these capabilities are poised to play a significant role in various domains, contributing to personalized services, improved healthcare, and enriched human-robot interactions.