EEL 4930/5934: Autonomous Robots
HH1: Hands-on Homework #1 (Spring 2023)

Tasks Overview:

A. Prepare Workspace: ROS, Catkin, and Python-OpenCV Packages
B. Interface webcam / usb camera in ROS
   i. Initiate camera and visualize image topics
   ii. Subscribe to image topic and extract data: OpenCV-Bridge
   iii. Perform image processing: detect face draw bounding boxes (in OpenCV)
C. Publish the output image (with face boxes) as a topic: visualize topics in `rqt_image_view`
D. Write a single launch file for the whole project, *ie, that does the following*
   i. Starts the `usb_cam` node (for step B.i)
   ii. Start the `face_detector` node (for step B.ii, B.iii, and C)
   iii. Start the `rqt_image_view` node for visualization

Grading Breakdown

EEL 4930	EEL 5934
• Part A: 25% • Part B: 50% (20% + 20% + 10%) • Part C: 25% • Part D: extra! (not required, may get bonus points)	• Part A: 20% • Part B: 45% (15% + 20% + 10%) • Part C: 20% • Part D: 15%

References:

- Lecture 1-2 contents and ROS wiki
- Recommendations:
  - Use a linux laptop (virtual OS is fine) and its built-in camera
  - Alternatingly use a PC or Raspberry PI (3 or 4) or Jetson nano (use any USB camera)

Submission: [Through Canvas only; Due: Feb 7, 2023 by 11.55pm]

- A single zip file with no more than 10MB size
  - A readme.txt with your name, GatorID, ROS version, OS version, etc.
  - Your ROS package (only your new Catkin package, do not include anything else)
  - A PDF of step-by-step demo with screen-shots of terminal outputs
- Assignment more than 10 MB file size will get negative penalty (-10% to -50%)

Part A: Prepare Workspace: ROS, Catkin, and Python-OpenCV Packages

- Install Python and OpenCV libraries (if you do not have them already)
  - Get Python (3.8 or 3.9): `sudo apt install python3`
  - Verify the installation: `python3 --version`
  - Get OpenCV 3.2.x: `sudo apt install python3-opencv`
  - Verify the installation: `python3 -c "import cv2; print(cv2.__version__)"`

- Install ROS (if you do not have them already)
  - Installation: https://wiki.ros.org/ROS/Installation
  - Make sure to install the correct distribution for your platform (see Lecture 2 slides)
  - **ROS Noetic**
    - Primarily targeted at the Ubuntu **20.04** (Focal); should work with Raspberry Pi 4s
    - Follow the installation instruction and reference video
  - **ROS Melodic:**
    - Primarily targeted at the Ubuntu **18.04** (Bionic); should work with Raspberry Pi 3s
    - Follow the installation instruction

- **Create and setup a Catkin workspace:**
    - Follow the CreateWorkspace Tutorial and reference video
    - Practice a couple of sample projects (talker/listener, turtlesim, etc.)

Part B: Interface webcam / usb camera in ROS

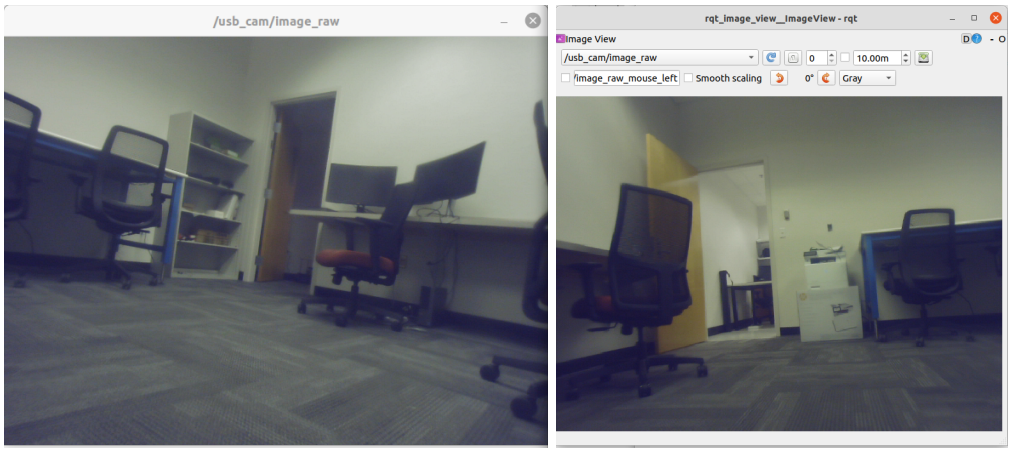
- Install the usb_camera package; *ie*: `sudo apt install ros-noetic-usb-cam`
- If you are using external usb cameras
    - Plug the camera and check which usb bus is reading it (`lsusb` command)
- Initiate camera by running the `usb_cam` package (which will start the `usb_cam` node)
    - You can use both `rosrun` or `roslaunch` to do this
    - Check the image topics once the camera is initiated: `rostopic list` (se below)

```
2: /opt/ros/noetic/share/usb_cam/launch/usb_cam-test.launch http://localhost:11311
boxiao@ece-p206c-magellanic:~/catkin_ws/src/my_face_detection$ roslaunch usb_cam usb_cam-test.launch
... logging to /home/boxiao/.ros/log/65fc4e68-984b-11ed-908e-3d0d7e24ac5e/roslaunch-ece-p206c-magellanic-12659
02.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ece-p206c-magellanic:35913/

3: boxiao@ece-p206c-magellanic: ~
boxiao@ece-p206c-magellanic:~$ rostopic list
/rosout
/rosout_agg
boxiao@ece-p206c-magellanic:~$ rostopic list
/image_view/output
/image_view/parameter_descriptions
/image_view/parameter_updates
/rosout
/rosout_agg
/usb_cam/camera_info
/usb_cam/image_raw
/usb_cam/image_raw/compressed
/usb_cam/image_raw/compressed/parameter_descriptions
/usb_cam/image_raw/compressed/parameter_updates
/usb_cam/image_raw/compressedDepth
/usb_cam/image_raw/compressedDepth/parameter_descriptions
/usb_cam/image_raw/compressedDepth/parameter_updates
/usb_cam/image_raw/theora
/usb_cam/image_raw/theora/parameter_descriptions
/usb_cam/image_raw/theora/parameter_updates
boxiao@ece-p206c-magellanic:~$
```

    - You can visualize the image data using `rqt_image_view` (see below)



- Now create your own ROS package which will
    - Subscribe to the image topic of interest, *ie*, `/usb_cam/image_raw`
    - Convert the ROS image data to OpenCV image data
        - By using Open-CV bridge (see this tutorial)
        - CvBridge is a ROS library that provides an interface between ROS and OpenCV

Here is a sample piece of code, that does the following

- Initiates a ROS node named `'my_node'`
- This node Subscribes to the image topic of interest, *ie*, `/usb_cam/image_raw`
- Converts the ROS image data to OpenCV image data
    - `imRos = rospy.Subscriber(topic, Image, self.imaCallBack, queue_size=3)`
    - The `imaCallBack` function is called every time there is data in this specific topic name
- The `imaCallBack` function gets `inp_im` witch is he ROS image data
- So it is converted to OpenCV image data (*eg*, Numpy array)
    - **`imCV`** `= self.bridge.imgmsg_to_cv2(inp_im, "bgr8")`

```
import cv2
import rospy
from sensor_msgs.msg import Image
from threading import Lock
from cv_bridge import CvBridge, CvBridgeError

class ImagePipeline:
    def __init__(self):
        self.mutex = Lock()
        rospy.init_node('my_node', anonymous=True)
        self.bridge = CvBridge()
        topic = '/usb_cam/image_raw'
        imRos = rospy.Subscriber(topic, Image, self.imaCallBack, queue_size=3)

        self.ImOut = rospy.Publisher('/out/image', Image, queue_size=3)

        try:
            rospy.spin()
        except KeyboardInterrupt:
            print("Rospy Spin Shut down")

    def imageCallBack(self, inp_im):
        try:
            imCV = self.bridge.imgmsg_to_cv2(inp_im, "bgr8")
        except CvBridgeError as e:
            print(e)
        if imCV is None:
            print ('frame dropped, skipping tracking')
        else:
            self.ImageProcessor(imCV)
```

Hence, now you do your processing by implementing `self.ImageProcessor(imCV)`

- Detect faces in **`imCV`** image and draw bounding boxes by using OpenCV (see this tutorial); steps:
    - Download the OpenCV cascade face detection model
    - Declare `faceCascade = cv2.CascadeClassifier('model_path')`
    - Convert image to gray `gray = cv2.cvtColor(imCV, cv2.COLOR_BGR2GRAY)`
    - Detect face
      ```
      faces = faceCascade.detectMultiScale(gray, scaleFactor=1.1,
                                           minNeighbors=5, minSize=(30, 30),
                                           flags = cv2.cv.CV_HAAR_SCALE_IMAGE)
      ```
    - Draw bounding boxes
      ```
      for (x, y, w, h) in faces:
          cv2.rectangle(imCV, (x, y), (x+w, y+h), (0, 255, 0), 2)
      ```

Part C: Publish the output image (with face boxes) as a topic: visualize topics in rqt_image_view

- Finally you can publish the output image as a ROS topic
- You already have the data structure in place
    - `self.ImOut = rospy.Publisher('/out/image', Image, queue_size=3)`
- Note that we now need to convert it back!
    - Convert OpenCV image data to ROS image data
    - Use the `CvBridge().cv2_to_imgmsg(.)` function
    - Then publish the `self.ImOut.publish(.)` function
- Learn how to publish your processed image as a ROS topic this way!
- Then visualize the image topics (input/output) by using `rqt_image_view`
    - Point your webcam/camera to your face and see the feed in `/usb_cam/image_raw` topic
    - You should see the corresponding output in the `/out/image` topic

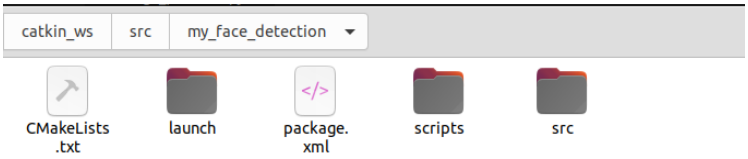
Part D: Write a single launch file that for the whole project

Notice that the whole process needs to run several ROS nodes.

- The `usb_cam` node
- Your ROS node (`my_node` or whatever you name it)
- And the `rqt_image_view` node for visualization

ROS launch files allow you do initiate all these nodes through a single launch file

- Write a launch file that achieves this!
- Then test it using `roslaunch [your_package_name] [launch_file_name]`
- Your package directory should look like the following



Remember the submission instructions

- Submit through Canvas only
- Due: Feb 7, 2023 by 11.55pm

- Submit a single zip file with no more than 10MB size
    - A readme.txt with your name, GatorID, ROS version, OS version, etc.
    - Your ROS package (only your new Catkin package, do not include anything else)
    - A PDF of step-by-step demo with screen-shots of terminal outputs

- Assignment more than 10 MB file size will get negative penalty (-10% to -50%)