EEL 4930/5934: Autonomous Robots
HW **#1**: **Camera Interfacing in ROS** (Spring 2025)

Task Overview:

A. Prepare Workspace: ROS, and Python-OpenCV Packages
B. Interface webcam / USB camera in ROS
   i. Initiate camera and visualize image topics
   ii. Subscribe to the 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, *i.e., that does the following*
   i. Starts the `usb_cam` node (for step B.i)
   ii. Start the `face_detector` node (for steps 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: not required, 5% bonus points	• **Part A:** 20% • **Part B:** 45% (15% + 20% + 10%) • **Part C:** 25% • **Part D:** 10%

References:

- Lecture 1-2 contents and ROS wiki. **Video demo:** https://youtu.be/26vqSGt_iV0?t=277
- Recommendations: use a Linux laptop (virtual OS or docker container is fine) and its built-in camera

Submission: [Canvas only; Due: February 3 (Monday) by 11.55pm]

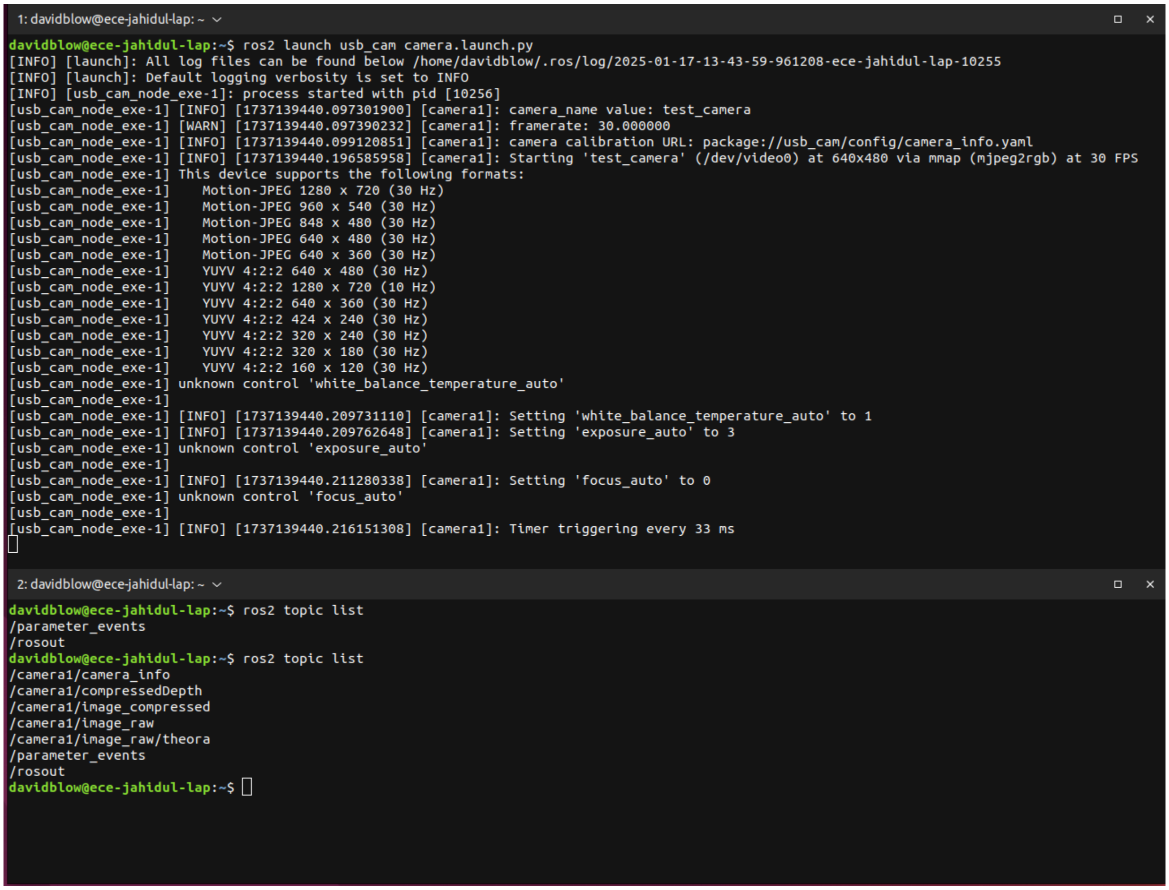
- A single zip file with no more than 10MB in size
  - A readme.txt with your name, UF email, ROS version, OS version, etc.
  - Your ROS package
  - A PDF of step-by-step demo with screen-shots of terminal outputs
- Assignments of more than 10 MB in file size will get a negative penalty (up to -25%)

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

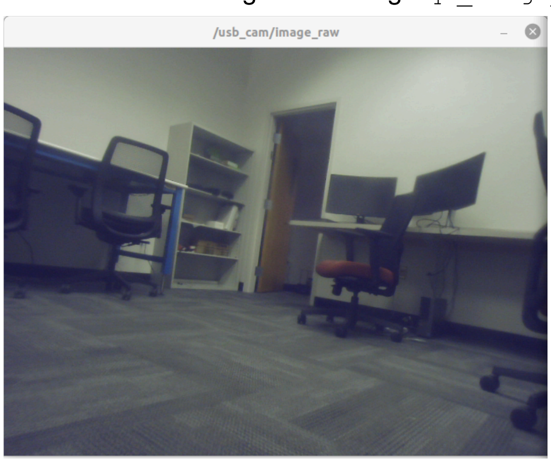
- Install Python and OpenCV libraries (if you do not have them already)
  - Get Python (3.8+): `sudo apt install python3`
  - Verify the installation: `python3 --version`
  - Get OpenCV 3.2.x: `sudo apt install python3-opencv`
  - Verify the installation: `python3 -c "import cv2; print(cv2.__version__)"`
- Install ROS (if you do not have them already)
  - Installation: https://docs.ros.org/en/humble/Installation.html
  - Make sure to install the correct distribution for your platform (see Lecture 2 slides)
  - **ROS2 Humble:**
    - Primarily targeted at the Ubuntu **22.04** (Jammy)
    - Follow the installation instructions and reference video to install ROS Humble
    - Practice a couple of sample projects (talker/listener, turtlesim, etc.)
      - Highly recommend going through beginner tutorials

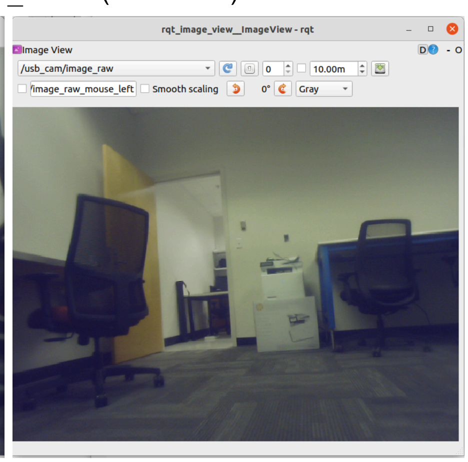
Part B: Interface webcam / USB camera in ROS

- Install the usb_camera package; *ie*: `sudo apt install ros-humble-usb-cam`
- If you are using external USB cameras
  - Plug the camera and check which USB bus is reading it (`lsusb` command)
- Initiate the camera by running the `usb_cam` package (which will start the `usb_cam` node)
  - You can use both `ros2 run` or `ros2 launch` to do this
  - Check the image topics once the camera is initiated: `ros2 topic list` (see below)



- 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
    - `subscription=`**`self.create_subscription`**`(Image, topic,self.listener_callback, queue_size=3)`
    - The `listener_callback` function is called every time there is data in this topic name
- The `listener_callback` function gets `inp_im` which is the ROS image data
- So it is converted to OpenCV image data (*eg*, Numpy array)
    - **`imCV`** `= self.bridge.imgmsg_to_cv2(data)`

```
import rclpy # Python library for ROS 2
from rclpy.node import Node
from sensor_msgs.msg import Image # Image is the message type
from cv_bridge import CvBridge #Convert between ROS and OpenCV Images
import cv2

class ImageSubscriber(Node):
  def __init__(self):
    # Initiate the Node class's constructor and give it a name
    super().__init__('image_subscriber')
    self.subscription = self.create_subscription(Image, '/usb_cam/image_raw',
        self.listener_callback, 3)
    self.subscription # prevent unused variable warning

    # Used to convert between ROS and OpenCV images
    self.bridge = CvBridge()

  def listener_callback(self, data):
    # Convert ROS Image message to OpenCV image
    imCV = self.bridge.imgmsg_to_cv2(data)
```

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 = self.create_publisher(Image,'/out/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 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)
- The `rqt_image_view` node for visualization

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

- Write a launch file that achieves this!
- Then test it using `ros2 launch [your_package_name] [launch_file_name]`

Video demo: https://youtu.be/26vqSGt_iV0?t=277
