Homework 4

ME 416 - Prof. Tron

2024-04-12

The two problems in this assignment are independent. Problem 1 applies Euler's method to approximately track the position of the robot (*dead reckoning*). Problem 2 shows two applications of *color-based thresholding*, the most basic image segmentation technique.

Preparation. Use `rastic_cl ws update` to update the software on your `roslab` account.

Problem 1: Euler-based odometry from encoders

Goal Implement a simple pose estimation routine by integrating the analytical model for the encoders. The implementation will use two nodes:

1) `encoders_publisher`, which continuously reads changes in the encoder readings, and publishes (at 50 Hz) estimates of the wheel speeds to the topic `/encoders` with type `MotorSpeedsStamped`;

2) `odometry_encoders`, which reads the values from `/encoders`, and uses them together with the model of the differential drive robot to maintain and estimate of its 3-D pose (with respect to its initial position).

The result is then visualized using RViz, an important tool in ROS that is used to display 3-D geometrical quantities.

Preparation. Copy the provided file `me416_lab/me416_lab/odometry_encoders_template.py` to `me416_lab/me416_lab/odometry_encoders.py`

Question node 1.1 (0.5 points). For this node, we start from the differential robot model $\dot{z} = A(\theta)z$ seen in class and in previous assignments. The node `odometry_encoders` will maintain a state estimate $\dot{z}$, and update it using Euler's method (in the callback `encoders_callback`( )), using the data received from the encoders, received from the topic

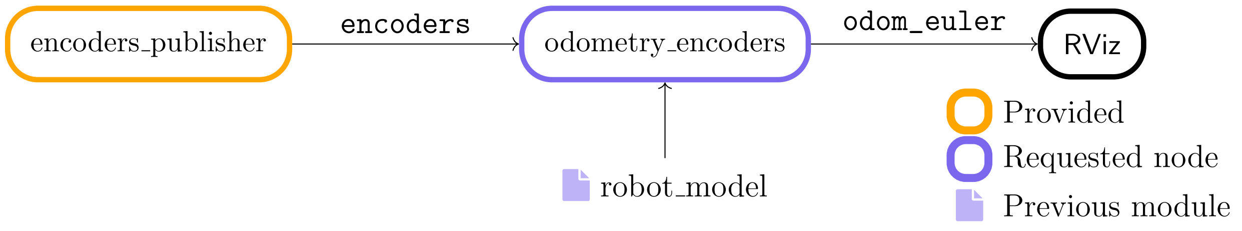


Figure 1: ROS graph of the nodes and files used in Problem 1

`/encoders`. The estimated pose is published at a fixed rate on the topic `/odom_euler` by the timer callback `timer_callback`( ).[1]

The callbacks `encoders_callback`( ) and `timer_callback`( ) are described in detail further below.

File name: `odometry_encoders.py`
Class name: `Odometry`
Subscribes to topics: `encoders` (type `me416_msgs/MotorSpeedsStamped`)
Publishes to topics: `odom_euler` (type `geometry_msgs/PoseStamped`) Frequency: 2 Hz.
Description: At initialization, the node should

1) Create a publisher for the topic `/odom_euler`.

2) Create a subscriber for the topic `/encoders` with `self.encoders_callback`( ) as the callback, and a timer that calls `self.timer_callback`( ) at 0.5 s intervals.

3) Initialize a state estimate `self.z_state` corresponding to $z(0) = 0$.

4) Create an attribute from the class `StampedMsgRegister` (imported from the `robot_model` module, this will be used by `self.encoders_callback`( )).

Question node 1.2 (0.5 points). Next, it is necessary to implement a callback for the messages from the encoders node.

File name: `odometry_encoders.py`
Class name: `Odometry`

Function name: `encoders_callback`
Description: The function updates `self.z_state` by using the function `euler_step`( ) (imported from `robot_model`) using the following inputs:

- The $[3 \times 1]$ vector with the current state z is given by `self.z_state`.
- The $[2 \times 1]$ vector with the current inputs u is build by concatenating the `left` and `right` fields of the received message `msg`, and then dividing by a constant `u_scale` (start with `u_scale=3600`; this will be tuned in Question report 1.1).
- The `step_size` is the time elapsed between the current message `msg`, and the message received in the previous call to the callback; this can be computed using the `replace_and_compute_delay`( ) from the attribute of type `StampedMsgRegister`.

Input arguments

- `msg` (type `MotorSpeedsStamped`): message from the `/encoders` topic.

[1]We separate the update and the publishing, so that an estimate of the pose is published even if no updates are received on the topic `/encoders`

Note: remember that `step_size` will not be valid for the first message received by the callback is. In this case, simply skip the update for `self.z_state`.

optional You should also skip the update if the `step_size` is much larger than 0.02 s; such a large delay is probably caused by the encoders' node being restarted, and would cause a large, erroneous jump in the estimate.

optional You can call the method `timer_callback`( ) explicitly after the update to reduce the lag in the odometry update.

Question node 1.3 (0.5 points). To complete the node, it is necessary to implement a timer callback that regularly publishes the odometry estimate.

File name: `odometry_encoders.py`
Class name: `Odometry`

Function name: `timer_callback`
Description: The function performs the following steps:

1) Create a message object of type `PoseStamped`.
2) Extracts the angle `theta` from the current state estimate `self.z_state`.
3) Fills the field `pose.orientation` of the message by creating a quaternion from Euler angles (this part has been already implemented for you).
4) Fills the field `pose.position` with the other entries of the current state estimate `self.z_state`.
5) Stamps the message (i.e., fills `header.stamp` as discussed in a previous assignment, and sets `header.frame` to the string `'map'`).
6) Publishes the message.

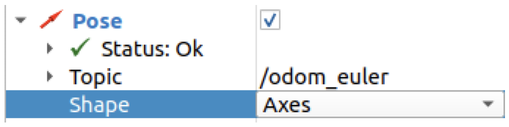
Note: the string `'map'` will tell RVizto visualize the pose in the standard world reference frame (which is called `'map'`).

Question report 1.1. Given what seen in class, explain why the function `euler2quat`( ) in the code provided for `timer_encoder`( ) is called with two arguments set to zero. What do they correspond to?

Question video 1.1. Run the following two nodes together: `odometry_encoders`, `encoders_publisher`; then, run RViz by using the command `rviz2`. In RViz, setup the visualization with the following steps:

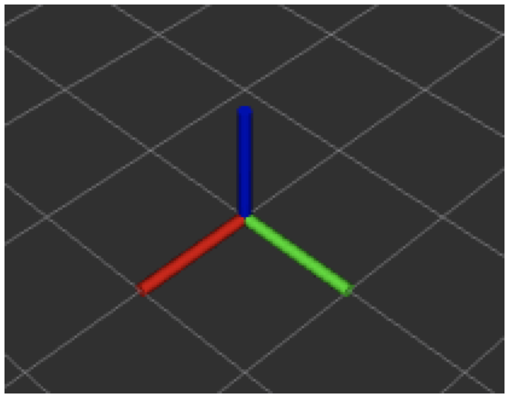
1) Press the Add button, then select Pose under the /odom_euler topic, and press the OK button.
2) From the drop-down tree in the lefthand panel, choose Axes for the Shape property (Fig. 2a).

You should see a reference frame where the colors RGB correspond to the xyz axes, respectively (Fig. 2b).



Pose	✓
Status: Ok	
Topic	/odom_euler
Shape	Axes

(a)



(b)

Figure 2: Visualization in RViz.

- If you are running the code on `roslab.bu.edu`, you should see the robot rotating around the left wheel (i.e., a point along the positive y axis).
- If you are running the code on the real robot, you should see the frame move when the wheels spin. Move the robot using either `scripted_op` or `key_op`, and you should see the movement replicated in RViz. Adjust the value of `u_scale` to adjust the scale of the motion.

Problem 2: Color-based line tracking

Goal In this problem, you will first write a small binary classifier (which can be seen as a decision tree that combines six linear classifiers). You will play the role of a machine learning algorithm, and find the classifier parameters by hand. The code for applying and testing the classifier, together with some utility functions to extract and visualize the horizontal centroid of an image mask, will be written in the `image_processing` module (directory `row_ws/src/me416_lab/image_processing`). You will then use these functions to track a colored line in a video stream through the `image_segment` node. See fig. 3 for an overview of how the code is structured.

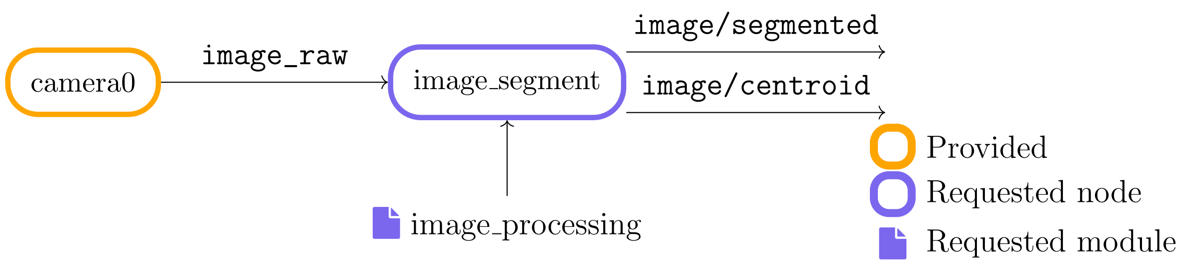
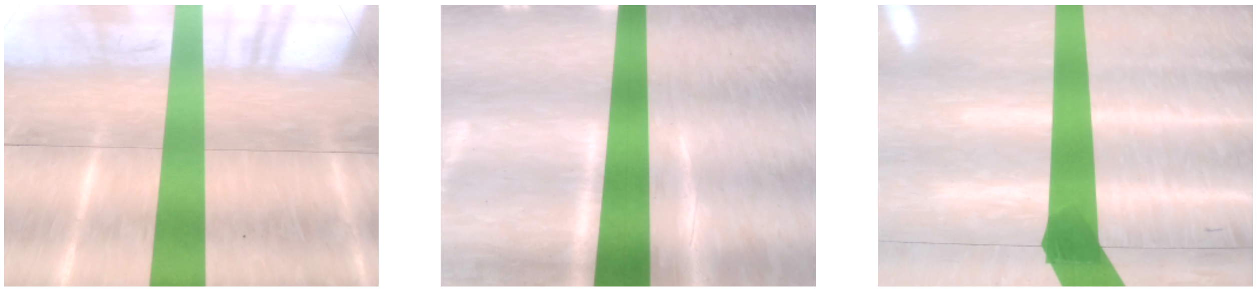


Figure 3: ROS graph of the nodes and files used in Problem 2



(a) Training (b) Cross validation (c) Testing

Figure 4: Provided images

2.1 Obtaining a dataset

Goal To train, validate, and test a color-based classifier for pixels, you will need a dataset. You have three options for acquiring the dataset.

- If you want to use the real robot and the track in RASTIC, go to Section 2.1.1.
- If you want to use pre-recorded data, go to Section 2.1.2.

2.1.1 With a real robot and the track in RASTIC

Preparation. Find the U-shaped green track in the robot arena in RASTIC.

Preparation (preparing the robot). Place the robot on the test track, so that the line is visible in the middle of the camera frame. Run the command `ros2 launch me416_lab camera_launch.py` to start the camera in one terminal, and then run the utility `rqt` to visualize the live view. Adjust the tilt of the camera toward the floor as much as possible, so that the edge of robot frame is just out of the picture. You can visualize different parts of the track by either driving around with your scripts, or simply moving the robot by hand.

Preparation (collecting images). You can save a snapshot of the live camera view by using the button on the screen. Take three pictures from different points on the track, and call them `line-train.png`, `line-cross.png`, and `line-test.png`. As a suggestion, `line-train.png` and `line-cross.png` should be taken with slightly different lighting conditions (e.g., see Fig. 4a and Fig. 4b).

2.1.2 Prerecorded data

Preparation. Use the three files `line-train.png`, `line-cross.png`, and `line-test.png` provided in the `data` subdirectory of the `me416_lab` package (Figures 4a-4c).

2.2 Color-based image segmentation

Goal First build a training/testing set from an image, then manually create a classifier, and finally apply the classifier on new images.

2.2.1 Initial parameters for the classifier

Question report 2.1. Create a Python script `image_dataset.py` that uses the functions `imread( )`, `imwrite( )` and NumPy array slicing to create rectangular cropped versions of the main training and testing files into the *six* image files listed below. Refer to a previous in-class activity for details on how to perform these operations.

- `train_positive.png`, `cross_positive.png`, and `test_positive.png` that contain pixels only from the track line (white in the provided files), from the files `line-train.png`, `line-cross.png`, and `line-test.png`, respectively.
- `train_negative.png`, `cross_negative.png`, and `test_negative.png` that contain pixels only from the background (the floor tiles in the provided files), from the files `line-train.png`, `line-cross.png`, and `line-test.png`, respectively.

Include the listing of the script and the six images in the report. These six images represent the *training*, *cross-validation*, and *testing* datasets for the rest of the assignment. *Note:* Using the PNG format is preferable because it is lossless, but the JPEG format is also acceptable.

The tilde shortcut for the home directory `~` will not work for paths specified in a Python script. To avoid problems, it is best to specify the paths for the image files using absolute paths. For instance, use `/home/tron/rastic_ws/` instead of `~/rastic_ws/`.

optional Repeat the process with images from different parts of the track, so that you have multiple training and testing files. In general, more data will give you a more accurate evaluation of the classifier that you are required to build in the next questions.

optional To make the classification in the next questions more reliable, transform the image to the HSV colorspace before saving the image. See the documentation at `https://docs.opencv.org/3.4.0/d7/d1b/group__imgproc__misc.html#ga397ae87e1288a81d2363b61574eb8cab`, using `cv::COLOR_BGR2HSV` as the value for the `code` argument.

Question report 2.2. Use the provided script `image_set_statistics.py` (under the `me416_lab/scripts` subdirectory, run `./image_set_statistics.py -h` for more detailed instructions) on the files `train_positive.png` and `train_negative.png`. This function will show you the histogram of values for each channel over all the pixels of the image, and also visualize each pixel as a 3-D point.

Your goal is to devise a set of thresholds on each channel to obtain a sufficiently high precision with substantial recall of pixels on the track (positive examples) versus the background (negative examples). For each channel, you need to choose three lower thresholds `threshold_low[0]`, `threshold_low[1]`, `threshold_low[2]`, and three upper thresholds, `threshold_high[0]`, `threshold_high[1]`, `threshold_high[2]` (one pair of thresholds for each channel) such that the positive samples fall between the two thresholds for each channel.

Include a screenshot of the output of the `image_set_statistics.py` script, adding on the figure where you decided to place the thresholds (do not use the cross-validation image yet, this will be done in Question report 2.5), and indicate also the exact values you choose in your report.

Question code 2.1. Copy the provided file `me416_lab/image_processing/image_processing_template.py` into the file `me416_lab/image_processing/image_processing.py` (all the functions below will refer to this file). Make a function that returns the threshold identified in the previous question.

> File name: `image_processing.py`
> Function name: `classifier_parameters`
> Description: Returns two sets of thresholds indicating the three lower thresholds and upper thresholds for detecting pixels belonging to the track with respect to the background.
> Input arguments
>
> - None
>
> Returns arguments
>
> - `threshold_low` (length 3, type `tuple of floats`), `threshold_high` (length 3, type `tuple of floats`): sets of lower and upper thresholds for each channel.

Preparation. The set of thresholds from the previous question defines a cube in color space. Points (pixels) whose coordinates (colors) fall inside this cube are classified positive. With respect to the theory, you can see this process as a cascade where each classifier is linear and compares one of the entries against a threshold.

2.2.2 Image segmentation

Question code 2.2. This question creates a basic function that segment an image using simple thresholding.

> File name: `image_processing.py`
> Function name: `image_segment`
> Description: Take an image and two bounds value as the input, return a second image indicating which pixels fall in the given bounds.
> Input arguments
>
> - `img` (type `np.array`): a color image.
> - `threshold_low` (length 3, type `tuple of floats`), `threshold_high` (length 3, type `tuple of floats`): sets of lower and upper thresholds for each channel as in Question code 2.1.
>
> Returns arguments
>
> - `img_segmented` (type `np.array`): an image with one color channel where a pixel is white if the corresponding pixel in `img` is classified as positive (falls in the bounds), and black if they are classified as negative.

🔎 This function needs to some consideration regarding performance. Loops in Python are relatively slow. Hence, for performance reasons, you will need to use the OpenCV function `cv2.inRange`(`img, threshold_low, threshold_high`) (see also the documentation at

https://docs.opencv.org/3.4.0/d2/de8/group__core__array.html#ga48af0ab51e36436c5d04340e0
or use `cv2.inRange?` in Python)

2.2.3 Evaluating the segmentation

Question provided **2.1.** Use the following utility function to count pixels in binary images.

> File name: `image_processing.py`
> Header: `pixel_count` ( )
> Input arguments
>
> - `img` (type `np.array`): an image with one color channel.
>
> Description: Count the number of pixels that belong to the positive class (i.e., that are non-zero) versus those that belong to the other class (zero, black).
> Returns arguments
>
> - `nb_positive` (type `int`), `nb_negative` (type `int`): the number of pixels in the image belonging to the positive and negative classes.

Question report **2.3.** Consider the following pairs of images:

1) `'train_positive.png'`, `'train_negative.png'`;

2) `'cross_positive.png'`, `'cross_negative.png'`.

Write a Python script or use IPython to run the function `pixel_count` ( ) on each image and report:

- Number of points for each image.
- Total number of positives and negatives for each image.
- Number of false positives and false negatives for each image (for this, you need to consider that files ending in `_positive` have been cropped to contain only positive examples, and files ending in `_negative` only negative examples).
- Precision and recall for each pair.

Include the images and the values in your report using a table so that they can be easily interpreted.

Question report **2.4.** Write a Python script or use IPython to call the function `image_segment` ( ) on the images `line-training.png`, `line-cross.png`. Visualize the results using the function `cv2.imshow` ( ), and the function `cv2.imwrite` ( ) to save them to files (you can choose any file name, please refer to a past in-class activity on how to use these functions). Include the images in the report.

You should get a very good segmentation for the training image, but the segmentation for the cross-validation image might be substantially wrong. You will revisit this in the next question.

Question report **2.5.** Now,syou will verify the reliability of your classifier using the rest of your cross-validation dataset. Use the provided script `image_set_statistics.py` on the files `train_positive.png`, `train_negative.png`, `cross_positive.png`, and `cross_negative.png`. Include the new figure in the report. Using this figure, and the results from the previous questions, comment on whether the classifier obtained by using the training set alone is enough to obtain reliable results also on the cross-validation dataset. If not, go back to the function from Question code 2.1 and tune the values until you obtain a result that you think is acceptable; **report all the thresholds you tried before settling to the final values**. Remember that not all features (channels) might be useful for classification. If you change the thresholds, please include also the new segmentation results (as in Question report 2.4).

Question report 2.6. Compute the statistics listed in Question report 2.3 on the pair of images `'testing_positive.png'` and `'testing_negative.png'`. Compare these statistics with the training statistics in your report (you should not cheat by changing the thresholds after finishing training to maximize your testing performance).

2.2.4 Centroid extraction

Question provided 2.2. The following utility function draws a vertical line across the image

> File name: `image_processing.py`
> Header: `image_line_vertical` (`img,x_line`)
> Input arguments
> - `img` (type `NumPy array`): A color image.
> - `x_line`: The x coordinate of the line.
>
> Returns arguments
> - `img_line` (type `NumPy array`): A color image that contains the original image with the line overimposed.
>
> Description: Use `cv2.line` ( ) to draw the line. The returned image is a copy to avoid modifying the original.

Question provided 2.3. Make a function that transforms a grayscale (1-channel) image into a color (3-channels) image.

> File name: `image_processing.py`
> Header: `image_one_to_three_channels` (`img`)
> Input arguments
> - `img` (type `NumPy array`): An image with a single channel.
>
> Returns arguments
> - `img_three` (type `NumPy array`): An image with a three channels.
>
> Description: Transforms a grayscale (1-channel) image into a color (3-channels) image by repeating the information across the three channels.

Question code 2.3. Make a function that computes, in a robust way, the horizontal centroid of all the white pixels.

File name: `image_processing.py`
Header: `image_centroid_horizontal` (`img`)
Input arguments

- `img` (type `NumPy array`): An image that contains only pixels that are either white or black (no other colors).

Returns arguments

- `x_centroid` (type `int`): The median of the x coordinates of all white pixels. If there are no white pixels in the image (i.e., if the list is empty), the function should return zero instead of computing the median.

Description: This function should accumulate the x coordinates of all white pixels in the image, and then return the median of this list. Note: it is useful to know that if `a` is a list, `a.append(b)` appends `b` to the list `a`; moreover, if `c` is a NumPy array, `numpy.median(c)` returns the median of that array. Note that you need to cast the median from `float` to `int` to make it work.

Question provided 2.4. The following function performs a sequence of test operations on the test image. The function can be found in the file `image_processing_template.py`, but it should be copied to your file `image_processing.py`.

🔎 When running the function, either run it from the same directory where the file `line-test.png` is present, or change the argument to the call to `cv2.imread`( ) to point to the absolute location of that file.

File name: `image_processing.py`
Header: `image_centroid_test`( )
Input arguments

- None

Description: The function performs the following sequence of operations:

1) Load the file `line-test.png` from Question report 2.1.

2) Run `image_segment`( ) to produce a segmented image of the line from the background.

3) Run `image_centroid_horizontal`( ) to compute the horizontal centroid of the line.

4) Run `image_one_to_three_channels`( ) to transform the segmented image from 1 to 3 channels ("color" image).

5) Run `image_line_vertical`( ) to add a line on the segmented "color" image at the computed centroid.

6) Display the results (both the original image and the segmented image with the centroid line).

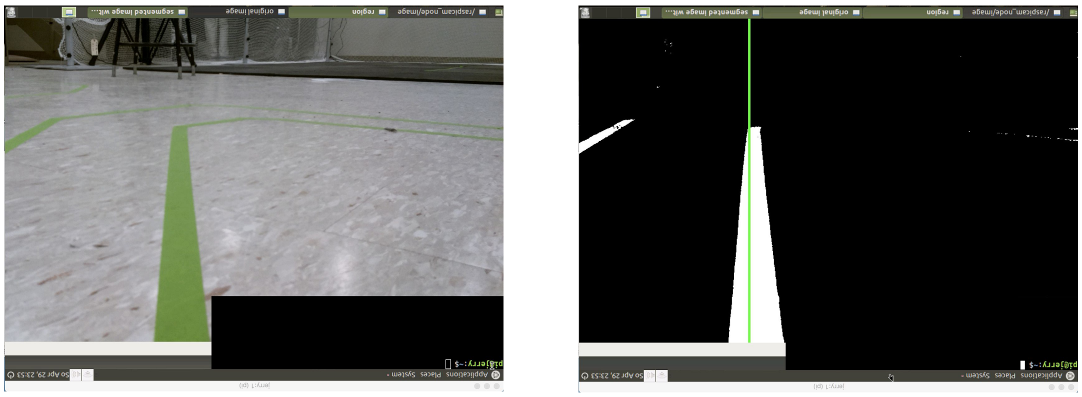


Figure 5: Example of original image, and segmented image with a line.

Question **report** **2.7.** Write a Python script or use IPython to call the function `image_centroid_test()`. Run the script, and include a screenshot of the result in your report. An example of the expected output is shown in Figure 5.

2.2.5 *A line tracking node*

Question **node** **2.1.** You are now ready to make a ROS node to perform real-time extraction of the centroid of the line from the images acquired by your robot.

> File name: `image_segment.py`
>
> Subscribes to topics: `/image_raw` (type `sensor_msgs/Image`)
>
> Publishes to topics:
>
> - `/image/segmented` (type `sensor_msgs/Image`)
> - `/image/centroid` (type `geometry_msgs/PointStamped`)
>
> Description: This node should import the file `image_processing.py` to use all the functions that you have developed so far. For each new image received from the camera, the node should perform the same operations described in Question provided 2.4, but without visualization of the results. Instead, the node should:
>
> *1)* Publish the segmented image with the line on the topic `/image/segmented`.
>
> *2)* Insert the computed horizontal centroid in the `x` field of a `PointStamped` message, add the current time to the `header.stamp` field, and then publish the message on the topic `/image/centroid`

You can use the provided file `image_flipper.py` as a starting template for the node. Include the listing of the file `image_segment.py` in your report.

Question **video** **2.1.** Using multiple terminals, follow the instructions below.

- If you want to use pre-recorded data, download the `ros2 bag` file `line.db3` (the link is available on Blackboard). Use the command `ros2 bag play --loop line.db3` to replay the recorded messages.

- If you are using the real robot, launch the `camera.launch` file and manually move the robot. Then, perform the following.

Run the `image_segment.py` node, together with the `rqt` utility to see the contents of the topic `/image/segmented` (if you are on the robot, run also the `ros2 topic echo /image/centroid` command to see the computed centroid). Check that the node is able to track the line robustly and without too much delay.

Make sure that, in the video, the visualized outputs of the segmented image and the computed centroid are visible at the same time, so that they can be compared.

🔎 You might need to change the thresholds of your classifier to account for changes in lighting conditions. This is an intrinsic problem of color-based thresholding.

Question video 2.2. Run a terminal with the command `ros2 topic hz /image/segmented` to visualize the speed at which you can process images. Make sure that the result is visible in the video. Note that, for this particular question, you do not need to run `rqt` to visualize images (this will give you slightly better rates). Please post your rates on the discussion board on Gradescope; the group(s) with the highest rate will receive an extra point for this question.

2.2.6 *Chroma key*

Question code 2.4. This function will not be used directly on our ROSbot, but it is nonetheless fun to see the results, and it will be used in a future in-class activity. **rtron**
This function definition cannot be fully tested

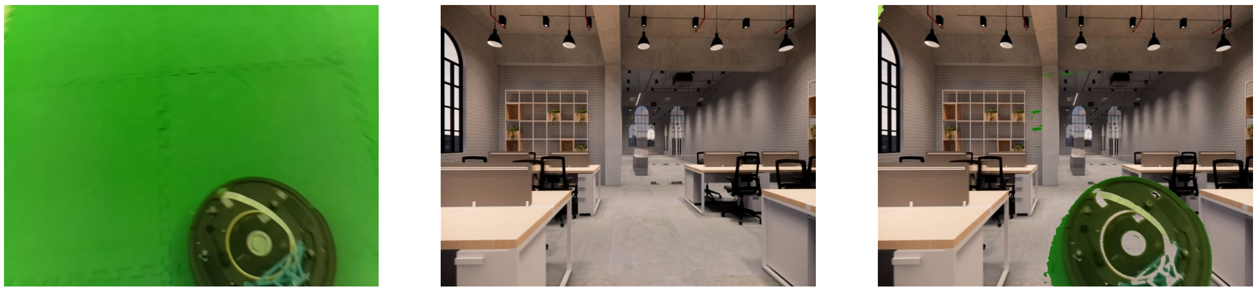
File name: `image_processing.py`
Function name: `image_mix`
Description: Given an image of an object (a robot) on a colored background, and a new background image as inputs, output a new image that combines the two by

1) Using `image_segment`( ) on `img_object`.
2) Creates a new image, of the same dimensions as `img_object`, such that
   - Every negative pixel (i.e., which is not part of the background) is kept equal to the corresponding one from `img_object`.
   - Every positive pixel (i.e., which is part of the background) is taken from the corresponding one in `img_background`.

⚠ To create `img_mix`, you will need to use either the attribute `copy`( ) of `img_object`, or `np.array`( ) to avoid modifying the original image in `img_object`.

Input arguments

- `img_object` (type `np.array`): a color image with an object on a solid-color background.
- `img_background` (type `np.array`): a color image of an arbitrary background, with dimensions larger than `img_object`.



(a) `img_object` (b) `img_background` (c) `img_mix`

Figure 6: Example of inputs and output for the function `img_mix`( ).

- `threshold_low` (length 3, type `tuple of floats`), `threshold_high` (length 3, type `tuple of floats`): sets of lower and upper thresholds for each channel for the background color.

Returns arguments

- `img_mix` (type `np.array`): an image where the colored background is substituted with the given background.

Question report 2.8. Open the robot dataset on the shared Google Drive folder at `https://drive.google.com/drive/folders/12Mvh-06oI0f70UzKuKN3Br-SPL0tvyNX`. Download at least two images from each of the folders `robot_training`, `robot_testing`, `background_training`, and `background_testing`, and upload them to `roslab` (in whatever location is most convenient for you). You should end up with four training images (two of the robot, two background), and four testing images. Write a Python script (or, less recommended, use IPython) to call the function `image_mix`( ) to combine all possible combinations of, separately, the training images, and testing images. You should end up with eight new combined images. Include all the original and obtained images in your report (organizing them in a clear way).

Hint for question node 1.1: The subscriber/publisher and update structure of the node is conceptually similar to `listener_accumulator` from Homework 1.

Hint for question report 2.2: Keep in mind two things:

1) The narrower the threshold, the higher the precision (what is classified as a line is really part of the line), but also the lower the recall (you might not find all the pixels in the line).

2) The distribution of the two classes might significantly overlap in one of the channels; that is not a problem, as long as there is at least one channel (dimension) in which the two do not overlap.

Hint for question code 2.4: To simplify the implementation of this function, examine the documentation of `numpy.where`( ).