In this lab you will be using color recognition for robot control.

This week you are expanding on your threaded main program. To get started you need to install:

opencv-python (the version should be >=3.3.x)

On the Mac::
```
sudo pip3 install opencv-python
```

On the Windows: Install MSVC build tools:
```
https://visualstudio.microsoft.com/downloads/#build-tools-for-visual-studio-2019
```
Then on the command line navigate to the Scripts folder in your python3 installation:
```
pip3 install opencv-python
```

During all lab work
You should keep notes that includes any data you collect or monitor during the lab -- for example, thresholds you decide to use in order to detect whether the robot is facing a white floor or the darker under the table space. In addition, this is a great place to include reminders to yourself -- commands you want to remember to copy-and-paste in the future or errors you don't want to make in the future! This section should be maintained and updated during the lab -- it can supplement any paper-and-pencil recording that you do.

Before you begin, you may wish to read the last section Submit so that you know what you'll be turning in. There are TWO extra requirements for this lab.

Video

Start by attaching your camera to the robot using the sticky velcro tape. You will want to point your camera upwards. It is okay to place your camera off-center.

Once your camera is plugged in you can turn on your RP. Eventually you should see the lights on the camera appear. Connect your laptop to the Robotics network and open your web browser to the address below where # is your RP number:

```
http://192.168.1.10#:8081/
```

Wave at yourself. If the video stream is working you can close the window and run the sample code. It should show you the same image twice. One window has sliders and you can adjust to highlight areas of interest based on the red channel.

Color

Vision is a remarkably rich source of input -- but it is also one of the most computationally challenging, because the things we might like to know about the environment (where a particular object is located, where there is freespace available to navigate) is wrapped up in very intricate ways among the pixels of an image.

To simplify the answers to these questions, we will use regions of uniform color as landmarks, e.g., the orange cone or red cups or blue bowls in the lab. Even in these cases, however, the

pixels those single-color objects produce can be difficult to extract using simple thresholding on each pixel's red, green, and blue coordinates.

As a result, you will also use the hue-saturation-value color space today: it is a transformation of red-green-blue (RGB) color space that tries, to a first approximation, to group perceptually similar colors in the same part of the space. The hue axis, which is circular, intends to represent roughly what we mean when we use different color names: green, cyan, yellow, etc. The value axis represents the amount of light present in a pixel -- that is, it's greyscale intensity, independent of hue. Finally, the saturation axis intends to capture roughly what we mean by the "richness" of a color, e.g., a deep scarlet would have a high saturation; a pale pink would have a low saturation. By using both HSV (hue-saturation-value) and RGB (red-green-blue) thresholds for a particular color, it should be possible to extract regions reliably.

Aside: OpenCV is a powerful image processing library that was originally developed by Intel in C. It is open source library under a very permissive license and widely used. Dr. Ben prefers it to Matlab processing for the speed you get from a native C/C++ codebase. We are using the Python wrapper, which means you have imagine from C++ documentation what the Python function will actually be. If you get stuck, open the Python3 interpreter, `import cv2`, and then do `help(cv2.`*`functionname`*`)`, this will give you the Python signature and sometimes better documentation of the specific function.
http://docs.opencv.org/3.3.0/

Action items: Help yourself identify thresholds. Start by adding sliders for Green and Blue and link them up correctly so that you can filter on these colors.

Next in the `doImgProc()` function, use `cv2.cvtColor(…)` to create a copy of the image that is in HSV color space. Then add sliders to filter the image on HSV values.

Object tracking like a puppy

You will track an object based on the center of mass of all pixels in the image that are “object-color”. Start by figuring the color of a tennis ball. Make sure you determine this in different lighting (under the light, far from the light, with the lights off, in a shadow of a person, etc.) as best you can. Encode these values in a ImageProc member variable. Use a dictionary like we did with thresholds.

Next, compute the center of mass of all of the pixels for this color. This is your object location, now you have to tell the robot to track it. The ***lofty*** goal is to show dynamic control of the robot by varying your speed based on the location of the tennis ball, the farther from the center of the image the faster your robot could move like so:

Near the center of the image	Move forwards at a normal speed
Far above the centerline	Move slower so you don’t hit it.
Far below the centerline	Move faster than normal to catch up
Left of center	Turn left
Right of center	Turn right
No tennis ball	Slowly search, stop turning after a complete turn

To help yourself in this challenge you will want to draw a crosshair where the center of mass is found. You can use `cv2.line(…)` there is also a function `cv2.circle(…)` that may be useful later.

For better tracking you may want a boolean in your imageProc thread showing if the object location was recently accessed. Make it false when you update position based on what you found in the image. Then in the state machine make it true when you drive the robot. If it is still true in the state machine later, just stop moving since there is no update since you last drove.

The problem with our image processing algorithm is that we only calculate the center of mass, so two tennis balls on opposite sides of the image would make it appear as if it was centered. You can test this by holding two tennis balls.

Goal: (ideal video) Show someone stealing the robot's attention (thereby showing successful tracking).

Goal 2: Track a beach ball. Due to reflections this could be a harder task. As part of this, have a key press change the tracking mode so that you can go back and forth between beach ball and tennis ball tracking.

Submit

You will earn a B if you have completed this lab (including a quality writeup and appropriate video evidence). To earn an A you must complete at least some amount of the **Bonus** tagged items. (How many? It is up to you! Tell me in your writeup why it was the right amount.) Also you are encouraged to make up your own bonuses. Be creative!

A writeup (each lab will have a writeup like this):

- In the **Progress** section, you should have a brief introduction that restates the tasks of the week's lab, along with a record of any notes, data, brainstorming ideas, and progress (positive and negative!) you made. For example, here you should explain the algorithm -- or algorithms -- you used for line-following. Briefly, what was your reasoning behind them? If you changed approaches, explain what happened to motivate that change. The Progress section does not have to be huge: usually 3-4 paragraphs is a good size to aim for, for each lab
- In the **Results and Media** section, include at least one paragraphs on your results, including (1) how your robot did (2) any surprises or changes that emerged as you worked on it and (3) things your robot can't do that you hoped it would, and (4) any problems that arose -- and whether/how they were overcome. Be sure to include:
    - Optional: Still pictures of your robot -- maybe a selfie with your team
    - **At least two screenshots of your system segmenting different objects - make sure that all of your team members are in at least one of them - either as background or as the things being tracked!**
    - At least two videos of your robot (10-20 seconds is best) - ideally, one with it running well and one where it's not, in order to contrast the two
    - For each video, you should include a caption: 3-4 sentences summarizing each of the videos: what it will show, how well the robot does, and (briefly), why.
    - You should include captions for each image, as well.
    - Video should be submitted as links.

- Finally, you should have a short **Reflection** section on how the lab went overall, including parts that were particularly frustrating (if any), any realizations/insights (positive or negative), and how you would try to extend your robot's capabilities if you had additional time. Also welcome but not required here are suggestions for other tasks that fit the lab's theme or other variations that would be interesting. (From these latter thoughts, you might converge on an idea you're excited about for a final project.)
    - Also, in this section, you should include at least a note on anything extra or different your team might have tried -- either something suggested in the lab or something you designed yourself. (I don't want to miss or forget these!)

Code? Your videos demonstrate your success and effort. If it isn't recorded, it won't be graded. You get credit for videos of failures too, so don't be afraid to share!