16-281 Lab 2

Spring 2026

DUE: 11:59 PM on Wednesday Jan. 28.

Description

In this lab, you are tasked with creating and calibrating a vision pipeline in Python. Once it is complete, you will apply it to several inputs. This lab is intended to provide an opportunity to apply computer vision algorithms learned in lecture, as well as gain programming experience in Python.

Any questions regarding the lab should be directed to the lab TAs or posted to Piazza. As questions come in, clarifications will be added to the handout and the Lab 2 Clarifications Piazza post.

Note: Like in hw2, you are not allowed to use library functions that trivialize the problem.

The list is not a complete list!
If you're unsure whether a function is allowed, ask on Piazza or consult a TA during office hours.

Examples of Python functions (packages) you **can't** use:	**Examples** of Python functions (packages) you **can** use:
• scipy (package) • skimage (package) • cv (package) • cv2.findContours • cv2.connectedComponents • cv2.connectedComponentsWithStats • cv2.threshold • cv2.adaptiveThreshold • cv2.erode • cv2.rotate • cv2.getRotationMatrix2D • cv2.drawContours • cv2.watershed • ……	• numpy (package) • cv2.GaussianBlur • cv2.cvtColor • ……

Suggested Workflow

1. Download handout zip
    a. Read through and understand ALL files
2. Take student images, resize if necessary
3. Add images to student_images directory
4. Fill in images.dat (as specified below)
    a. Placeholder examples given, replace them with your own values
    b. Distance in inches, threshold value should be six floats between 0-1
5. Complete threshold_image
    a. Complete threshold_image
    b. Tweak threshold value(s)/algorithm until looks good
    c. Test each image individually
6. Complete segment_image
    a. Complete segment_image
        i. Try the algorithms discussed in class
    b. Complete calculate_centroids
        i. The circle representing the centroid should be near the exact center of the ball
        ii. Test each image individually
7. Complete test_pipeline
    a. Complete centroids_to_dist
        i. Use your images to find what the focal length value of student images should be
        ii. Changing the TA focal length will result in a **zero** on the lab
    b. Complete image_to_dist
        i. Put everything together!
        ii. Test whole system using infrastructure/test_pipeline
    c. Create zip file using zip_submission(andrewID)

Style (10 points)

The code will be read and graded for style and legibility. In particular, the course staff will look for:

- Clear, descriptive variable names.
- Comments
- ……

Part 1: Building the Pipeline

1.1 Collecting Samples (10 points, 10 images)

In the REL, there are two jigs with tennis balls mounted on them. You should collect 5 images from each with your phone, at various distances from the jig. Take the pictures in the REL (you'll want the pictures to be as similar as possible to the ones provided to you). Try to range from 2 feet to 10 feet from the jig. **Carefully record how far each image is from the jig, in inches.** For reference, the balls are:

- **5 inches** apart on the small jig (5 images)
- **12 inches** apart on the large jig (5 images).

The images should be saved to the student_images folder and named [small|large]_[1|2|3...].[ext] (for example, small_1.jpg, large_4.jpg, small_2.jpg)

PLEASE SAVE YOUR FILES IN .jpg FORMAT!

1.1.1 .dat file

There should also be an images.dat file in the student_images folder. You can edit the file by using vscode or other similar softwares to open it as a text document. Each uncommented line should have a filename, then a space, then the distance from the camera to the jig (in inches), then a space, then a number. The last 6 numbers are optional thresholds that you can use as you wish as inputs to the following section (1.2).

NOTE:

1. DO NOT REMOVE/EDIT/COMMENT OUT THE FIRST FIVE LINES. THE CODE WILL HAVE AN ERROR IF YOU DO THIS!
2. IT IS SUGGESTED THAT YOU SHOULD NOT REMOVE/COMMENT OUT THE REST OF LINES. THE CODE MAY HAVE ERROR BEHAVIOR IF YOU DO THIS!

For example:

```
# Sample images.dat file
small_1.jpg 50 0 0 0 0 0 0
small_2.jpg 75 0 0 0 0 0 0
small_3.jpg 144 0 0 0 0 0 0
large_1.jpg 50 0 0 0 0 0 0
```

Important Note: each student image file cannot exceed **1 megabyte** in size. For each megabyte that goes over this limit, 1 point will be deducted from the lab. Scores will not be reduced to below 0 points.

Deliverables:
10 images in the student_images folder and a corresponding images.dat

Testing:
To verify that the course infrastructure can see the images and the text file, run verify_config.py. Furthermore, the TA images use the same file format to run our tests (the images whose distances you don't know have a distance of 0).

1.2 Thresholding (20 points)

In this section, you will implement one function: threshold_image in the script threshold_image.py. Given 6 thresholds between 0 and 1, you should threshold the image as described in lecture. In particular, you should consider the many different ways to turn your image into a binary image (RGB, HSV, etc.).

Since the threshold is a float between 0 and 1, remember to potentially shift your threshold to 0-255 as needed.

Your images may need different threshold values, which you should determine experimentally for each of your images. Once you find the thresholds you feel best highlights the tennis balls while masking the background, punch it in as the last 6 terms of the appropriate images.dat entry, as values from 0-1. Don't forget to do this for the TA images as well! You don't have to use all 6 numbers if you don't want to. Just ignore the other numbers you inputted, but DO NOT change the function header NOR the file format.

Deliverables:

1. Update the two images.dat files with your threshold values
2. Implement the threshold_image function which converts your image to a binary image using a threshold (array of six values). The **output** should be an image of **size w*h*1**:
   NOTE: THE OUTPUT IMAGE MUST BE A GRAYSCALE IMAGE WITH BINARY ELEMENTS (0 OR 255)

Testing:

Run threshold_image.py and a folder named "output_images_thresholded" will be created in the same directory with the thresholded images in the folder. **MAKE SURE IMAGES LOOK CORRECT BEFORE CONTINUING!**

1.3 Segmentation (20 points)

In this section, you will implement two functions: segment_image and calculate_centroids in the script segment_image.py.

segment_image should take a **binary image after thresholding** as an input and segment it into a labeled image, where each segment has a distinct label. We recommend that you DO NOT IMPLEMENT SEGMENTATION RECURSIVELY.

calculate_centroids should take a segmented image as an input and compute all the distinct centroids, then filter out those that are too large or too small.

Deliverables:

1. Implement the segment_image function which converts a binary image to an image with labels **(different intensity in a grayscale image)**.
   ANY SOLUTION THAT LABELS NUMBERS ON THE IMAGE WILL RESULT IN A ZERO.
2. Implement the calculate_centroids function which outputs a 4x2 matrix with the locations of the centroids in (x, y) format. **BLUE CIRCLES WILL BE LABELED AT THE POSITIONS YOU IDENTIFIED IN THE PROVIDED CODE.**

Testing:
Run segment_image.py and a folder named “output_images_segmented” will be created in the same directory with the segmented and labeled images in the folder.
We will grade this by inspection: first, that segment_image produces a correctly segmented image (for 10pts), and that calculate_centroids outputs reasonable values. (for 10pts). You could comment out some lines in the function segment_file_and_save for partial credit OR the ease of debugging.

1.4 Building the Pipeline

In this section, you will implement two functions: centroids_to_dist and image_to_dis in the scripttest_pipeline.py.

centroids_to_dist takes the centroids from calculate_centroids and the ball separation as input and uses that information to compute the distance. The ball_separation parameter will be the distance between the balls in inches (the small jig is 5in, the large jig is 12in).
CHANGING THE TA FOCAL LENGTH WILL RESULT IN A ZERO FOR THE LAB.

Calculating distance. Your phone camera is not a true pinhole camera, but the images we are working with are nicely behaved enough (tennis balls centered and parallel to the camera) that we can approximate the camera as a pinhole. So, you can use the ratio presented in class for determining the rig’s distance. To find the “focal length” of this hypothetical pinhole camera the easiest way is to manually search for the value until your images work. The value listed by your phone manufacturer is a good initial guess.

image_to_dist will run your entire pipeline with an image. Using an input image, a threshold value, a ball separation distance, and isTA boolean, run the functions that you wrote in previous sections.

Deliverables:
1. Implement the centroids_to_dist function which takes four centroids in (x, y) format, the ball separation, and isTA, and outputs the distance to the jig.
2. Implement the image_to_dist function which takes an rgb image of a jig as an input and outputs the distance to the jig.

Testing:
Testing and evaluation of 1.4 will be done in Part 2.

Part 2: Testing Your Pipeline

In this section, you will run your pipeline on several challenge images, for which you do not know the correct distance. The output of your pipeline is a distance, which will be compared to the TA solution after submission.

There are four stages on which you will run your image pipeline. Level 0 uses the images you took in 1.1. Levels 1-3 are of increasing difficulty using TA images that we provide to you. However, you do not

know the correct distance for every image (you should still inspect the output as a sanity check). For each section, we will compute the percent difference from the correct value, average them, and use them to compute your score as follows:

Grade on a section: round(weight * 4 * max((25% - avg_error, 0)).
Example: round(20pts * 4 * max(25% - 13.5%, 0)) = 9pts

2.1 Level 0: Proof of Concept (10 points)

In this section, we verify basic functionality of your vision pipeline. You have to correctly find the distance for your ten sample images (for up to 20 points).

2.2 Level 1: No Modification (20 points)

In this level, you must correctly find the distance of 10 unmodified TA images.

2.3 Level 2: Rotate (15 points)

In this level, the images from **2.2** have been rotated an arbitrary amount. This is intended to test that your segmentation and trigonometry code doesn't make any unnecessary assumptions.

2.4 Level 3: Noise (15 points)

In this level, the images from **2.2** have had random noise added. This helps test the robustness of your general algorithm to noisy images. The average error will be computed from all the errors.

Submission

Make sure all images are uncommented in all images.dat files. Then, you should zip your submission with the file name "YOUR_ANDREW_ID_lab2.zip". **All codes and images (including original and processed images) should be submitted.**
Upload it to Gradescope

Grading

This lab is out of 100 points. There are 120 points available, to be allotted as follows:

Section	Value
Collecting Data	10
Thresholding	20
Segmentation	20

Level 0	10
Level 1	20
Level 2	15
Level 3	15
Style	10

After grades are released, you have one week to dispute the grade before it is final, regardless of any mistake on our part. You may dispute a grade by coming to lab TA's office hours, or setting up an appointment via email (emails are on the course website).

How to Get a Non-zero Grade

Each of your images need to be < 1 megabyte in size (-5 for each MB)

Do not change any of the function headers

Your entire pipeline (for all images) must run in <12 mins, each minute over is -1 point.

Fill in images.dat as specified above

Do not change the focal length for the ta images in centroid_to_dist (It should be equal to 1450)

Do not use packages (functions) that simplifies the nature of the problem

Do not label numbers on the image for segmentation

Write your NAME AND ANDREW ID in each script that requires your implementation

Take/Write your OWN WORK. Using images from other students except from yourself will also be considered as a violation

Tips

- Jigs should be in center of frame (less lens distortion)
- Include extensions (.jpg)
- Follow the given steps
- Use the same units (inch) throughout
- If your code takes too long, your images may be too big or you may not be using the best data structure
- Remember to test your whole submission before uploading
- START EARLY!