TTIC 31040: Introduction to Computer Vision
Spring 2021

Problem Set #2

Out: April 5, 2021

Due: Thursday April 15, 11:59pm
Instructions: Turn in a write-up of your solution (as markdown cells) along with the code and the results as an iPython (Jupyter) notebook, including any figures or tables you generate. Please name your notebook the following:
`firstname-lastname-ps2.ipynb` and upload it on Canvas. Make sure the notebook you are submitting has been fully run, so that all the results are displayed in it.
Collaboration policy: it is OK (and encouraged) to discuss the problem set and any ideas for solving it with other students. However, in the end you must write your solution on your own. This includes writing any code and running any experiments.

1 Vanishing points and lines

For this entire problem set, we will be assuming the pinhole camera model as covered in class.

Problem 1 **[10 points]**
Prove that all lines corresponding to a given direction in 3D space, i.e., $\{\mathbf{a} + t\mathbf{n} \,|\, \forall \mathbf{a} \in \mathbb{R}^3\}$, with $\mathbf{n}$ non-orthogonal to the optical axis of the camera, have the same vanishing point in the image plane.

Advice: consider the (mathematical) limit of the projection of a point on a line as the depth of the point increases.

End of problem 1

Problem 2 **[10 points]**
Consider a set of parallel lines within a (non fronto-parallel) 3D plane; these lines are associated with a vanishing point (as proven in the previous problem). Prove that the vanishing points for all such sets for a given 3D plane form a 2D line in the image plane.

End of problem 2

Line parameterization We will work with the following parameterization of a line in Euclidean space: given a point $\mathbf{a} = (x, y)$ on the line and a direction vector $\mathbf{n} = (n_1, n_2)$, the line is the set of points $\mathbf{p}$ satisfying $\mathbf{p} = t\mathbf{n} + \mathbf{a}$ for $t \in \mathbb{R}$.

Distance from point to line The distance from the point $\mathbf{p}$ to the line $\mathbf{a} + t\mathbf{n}$ is given by:

$$D(\mathbf{a} + t\mathbf{n}, \mathbf{p}) = \|(\mathbf{a} - \mathbf{p}) - ((\mathbf{a} - \mathbf{p}) \cdot \mathbf{n})\mathbf{n}\| \tag{1}$$

Expanding this out we get:

$$D(\mathbf{a} + t\mathbf{n}, \mathbf{p}) = (\mathbf{a} - \mathbf{p})^T(\mathbf{I} - \mathbf{n}\mathbf{n}^T)(\mathbf{a} - \mathbf{p}) \tag{2}$$

(Note that this is all in the Euclidean space, not the projective space; so for lines/points in the image, $\mathbf{p}$, $\mathbf{a}$, $\mathbf{n}$ all have just two coordinates). You can (and should) understand how this is derived, e.g., from the Wikipedia page.
Now we will tackle the problem of finding the intersection of a number of lines. For an arbitrary set of N (unique) lines, they are unlikely to intersect at a unique point, even if mathematically they are projections of parallel lines in the scene. The main reason for this is numerical error in identifying the points that define the image lines.

Problem 3 **[10 points]**

Write a function that finds the best approximation for the intersection of a set of lines, defined in the least squares sense: For a set of N lines $\{(\mathbf{a}_i, \mathbf{n}_i)\}$, minimize the sum of squared distances from the found intersection to the lines in the set,

$$\mathbf{p}^* = \arg\min_{\mathbf{p}} \sum_{i=1}^{N} D(\mathbf{p}; \mathbf{a}_i, \mathbf{n}_i)^2 \tag{3}$$

You can solve this (convex) problem by setting $\frac{\partial D}{\partial \mathbf{p}} = 0$ and findind the least squares solution. You are allowed to use `np.linalg.lstsq`or a similar implementation (no need to implement it yourself).

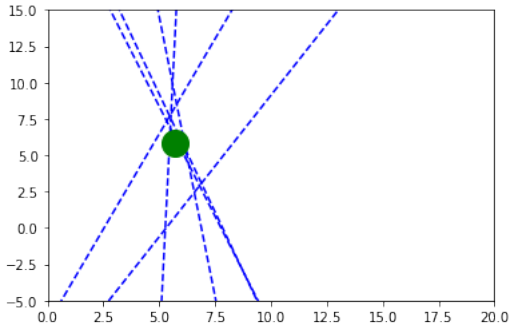

Figure 1: Finding the closest intersection to a set of lines. These are random lines for illustration only, not an actual example of lines converging on a vanishing point in an image.

End of problem 3

2 Vanishing Points: warm-up

The next set of problems will use the image of an American football field to estimate some properties of the scene from pixels. Sports is a major application of computer vision, and modern games are replete with vision-processed images, and it's common to see lines being drawn on the field or augmented reality applications. We'll do something similar in the next two problems.
For this problem, you are given a set of pixels denoting the start and end of straight lines in the image, and will use these to estimate the vanishing points as well as the horizon line.

Problem 4 **[10 points]**

The start and end points of the lines are stored in `points.txt` as a Python dictionary, where each pixel is named according to its line ID and whether it's a starting or ending point, e.g. `s11` corresponds to the starting point for line 1 in line group 1. We have already grouped the lines for you into three groups (though detecting and grouping straight lines in an image into sets of paralle lines is an interesting task in and of itself).

Use the provided function `draw_line` to visualize the lines.

End of problem 4

Problem 5 **[10 points]**

Find the vanishing points. As you saw in the lecture, vanishing points are found at the intersection of

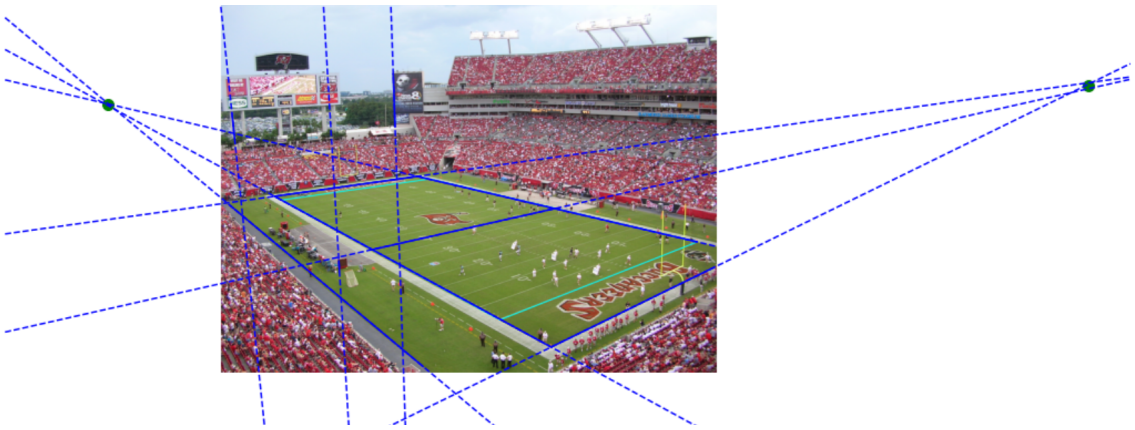

Figure 2: Visualization of two of the estimated vanishing points (the vertical vanishing point is much further away from the image plane – why? Explain in your writeup.)

parallel lines – like the ones you drew in the first part of this problem. Use your code from Problem 3 to find the intersection of lines to compute the location of the 3 vanishing points.

Your output should look like Figure 2. Observe that one of the vanishing points is far from the others – why?**not sure what you are getting at here**

End of problem 5

Problem 6 **[5 points]**
Use at least two vanishing points from sets of parallel lines on the ground to draw the horizon line. Does it correspond to your "intuitive" notion of a horizon? Write down any observations you might have.

End of problem 6

3 Playing fields and homographies

Refer to Figure 3 for a representation of an American football field. The field is measured in Imperial units, specifically *yards*[1]. The field is 100 yards long, with 5 yard lines dividing the field horizontally, and every 10 yards there's a helpful marker on the field outlining the distance from the endzone. (The endzones on either side of the field are 10 yards long, so the full rectangle is sometimes quoted as being 120 yards long).
In all of the following questions, we will use the following assumption: a football field is 100 yards long and 53 yards wide (giving it the aspect ratio of approximately 1.89). The true field is a rectangle, yet its image is distorted by perspective projection. First we will figure out how to undistort the field, then we will measure quantities on the field in world units. We will be working with the picture in `stadium.png`.

Problem 7 **[15 points]**
Load up `corners.txt` and draw the four corners on the field as dots, then use `cv2.polylines` to draw the polygon enclosing the field. Pick one of the corners in the image on the polygon to be the corner of the new rectangle. Compute the distance in pixels between that corner and the adjacent corner, and then use the aspect ratio of the field to compute where the other two corners of the rectangle should be. Then compute a mapping (homography) from the original rectangle found in `corners.txt` to the new "rectified" rectangle field. Your output should look like Figure 4.

End of problem 7

[1] A yard is about 0.914 meters

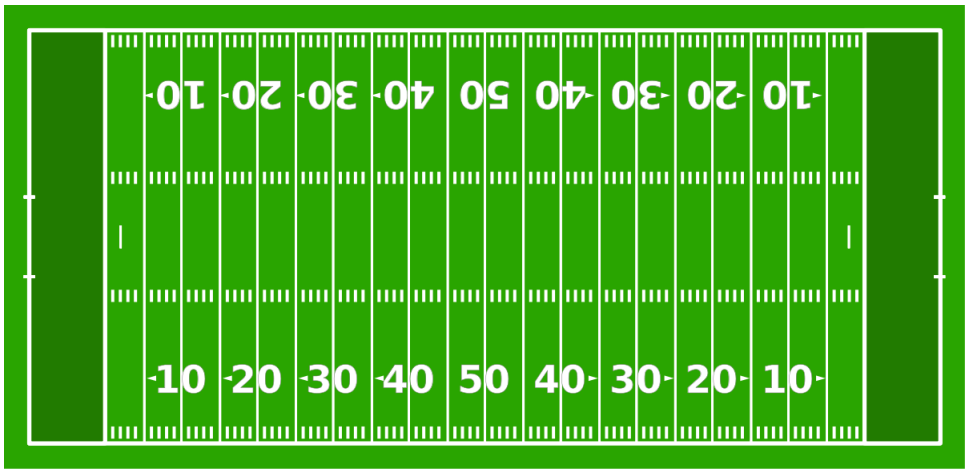

Figure 3: An American football field. Note the 5 yard lines.

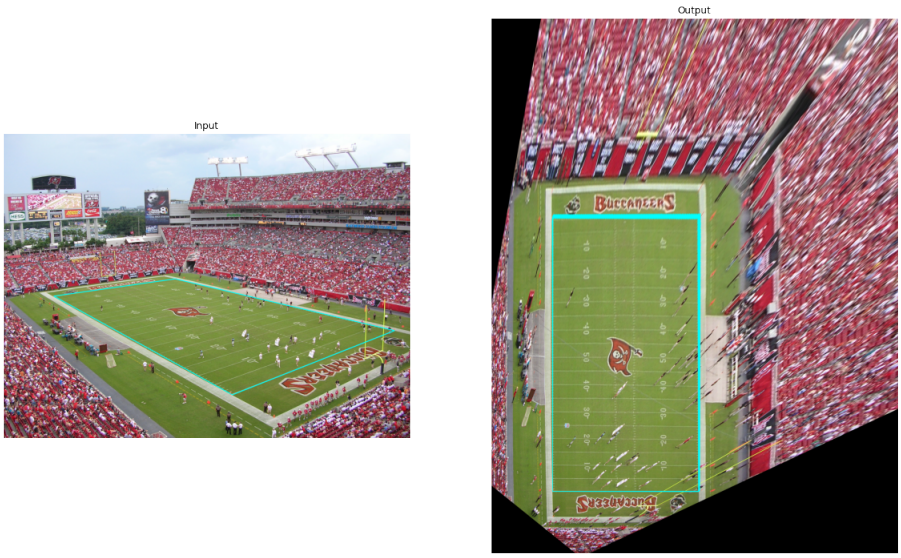

Figure 4: Warp the image on the left to generate the image on the right.

Problem 8 **[10 points]**

Load the dictionary in `players.txt`; it specifies positions (in the image) of three people on the field. Draw these positions, and the lines between them, as in Figure 5. Choose a 2D coordinate system for the field (i.e. the origin of the coordinate system should be one of the corners of the real field, and the axes should be aligned to the field's edges) and compute a homography between the field and image field. Then,

- Report the position (in yards) of each of the players
- Report the distance between the players in yards
- The center of the field should be $(height/2, width/2)$ in yards. Report the pixel position (in the original image) of the projection of this center.

End of problem 8

Figure 5: The pixel coordinates and lines between several players/referees on the field.

Problem 9 **[5 points]**
You might have seen sports broadcasts where lines are highlighted on the field to show the part of the field where an event took place. Using our knowledge of the homography between the physical field and the image, we will draw lines to highlight the yard lines in the field.

Hint: your solution should consist of a single for loop, skipping every 5 *yards from* 0 *to* 100.

End of problem 9

4 Single-View Metrology: A Rocket Factory

The geometry of vanishing points allows us to consider a variety of real-world applications. Space Exploration Technologies (SpaceX) is a cutting-edge American rocket company, and it has attracted a considerable internet fandom. Pictures inside SpaceX like Figure 7 are throughly analyzed to try to find the dimensions of new parts and test articles.
This picture was taken before the maiden launch of the Falcon Heavy rocket, and features the nosecone of a Falcon Heavy booster next to regular Falcon 9 "interstage"[2] You will use the known height of the interstage to measure the height of the new Falcon Heavy booster nosecone (in the image, it's the carbon composite cone that's half painted white).

Problem 10 **[15 points]**
Load `spacex_factory.txt` (a dictionary with the same naming convention as the one in Problem 2) and estimate the vanishing points.

[2]A lightweight cylinder that connects the first stage to the second stage, housing the second stage's rocket engine before stage separation.

Figure 6: Drawing the yard lines.

Next, load the dictionary `rocket_metrology.txt`. You are given a few positions in the image from which you can compute the image cross ratio, which will allow you to use a known height of one object to estimate the height of another. You are given the pixel representing the top of the interstage (`top_interstage`)the top of the nosescope (`top_noscone`)as so forth. Use the technique covered in class (relying on the projective invariant over four points) and the known height of the interstage ($R = 4.4$ meters) to estimate the height of the nosecone.

End of problem 10

Figure 7: SpaceX composites factory, building payload fairings (nosecones) and other light-weight stage elements.