Division of Engineering Science
UNIVERSITY OF TORONTO

ROB501: Computer Vision for Robotics

Assignment #1: Image Transformations

Fall 2020

The Background Story...

Alon Mosk, the intrepid CEO of SpaceY Corp., has launched a robotic mission to Mars! The SpaceY rover will scout the Red Planet for sites suitable for human colonization. Mosk has hired you as a rover navigation engineer, to develop portions of the visual navigation software that will be used to guide the rover and to avoid terrain hazards. You are excited to begin on this amazing programming adventure! (And also slightly nervous, since you have heard that Mosk is very fussy about code commenting and good programming practices....)

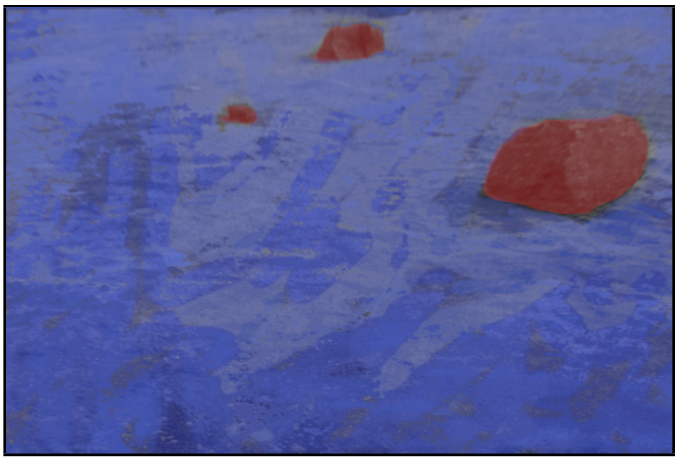
Assignment Overview and Learning Objectives

In this project, you will gain experience with the perspective transformation operation (discussed in the lectures), bilinear interpolation, and alpha blending. You will use the perspective transform to overlay one image on top of another image. The objectives are to:

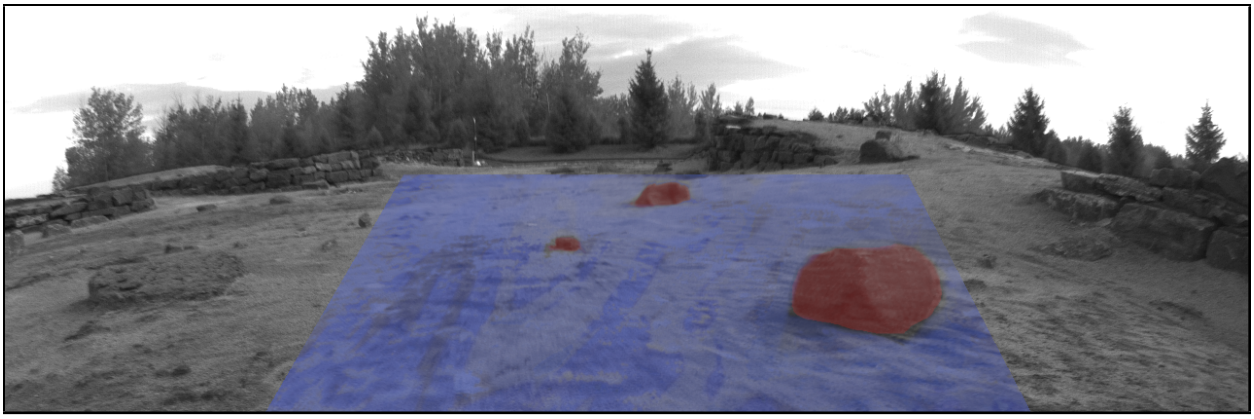
- aid in understanding perspective transformations or homographies (in two dimensions) and visualizing their application to images;
- experiment with inverse image warping and bilinear interpolation to overlay one image on top of another (respecting the appropriate scene geometry); and
- apply alpha blending to assist with visual interpretation by people (i.e., the rover operators).

The due date for project submission is **Tuesday, October 13, 2020, by 11:59 p.m. EDT**. All submissions will be in Python 3 via Autolab (more details will be provided in the lectures and on Quercus); you may submit as many times as you wish until the deadline. To complete the project, you will need to review some material that goes beyond that discussed in the lectures—more details are provided below.

(a) Rover Forward Navigation Camera Image



(b) Perspective False-Colour Obstacle Map



Portion of terrain overlay image showing rover hazards in red.

You have been contracted initially to produce a *terrain image overlay* tool. There are two images above: image (a) was captured by a small robotic rover at the Mars Emulation Terrain (MET) at the Canadian Space Agency (in Saint-Hubert, Quebec); image (b) is a false-colour rendering acquired from a different, near-overhead perspective—the colours identify different terrain surface characteristics and hazards.

Your assignment (should you choose to accept it) is to overlay the false-colour image on top of the rover camera image, respecting the proper image geometry (i.e., via a perspective homography). The overlay will incorporate *alpha blending* such that the false-colour rendering appears semi-transparent. Your overlay tool will help rover drivers to plan safe paths across varying terrain and will be a key part of the mission! The assignment has four parts, worth a total of **50 points**.

Please clearly comment your code and ensure that you only make use of the Python modules and functions listed at the top of the code templates. We will review and run your code.

Part 1: Perspective Transformations via the DLT

To carry out this exercise (Part 1), you will need to determine the perspective homography (warp) that transforms pixels from the (rectangular) false-colour map to the appropriate coordinates in the image from the rover navigation camera. The homography can be computed using the *Direct Linear Transform* (DLT) algorithm, given four point correspondences between the two images.

We did not review the DLT algorithm in the lectures, however it is straightforward and easy to implement in Python using `NumPy`. Details can be found in Section 2.1 of the (very useful) M.A.Sc. thesis written by Elan Dubrofsky of UBC, which is available on Quercus. For the moment, we will consider the four point correspondences to be exact—in later lectures, we will show how an overdetermined system of correspondences can be solved to produce an optimal homography estimate. For this part of the project, you should submit:

- a single function in `dlt_homography.py`, which computes the perspective homography between two images, given four point correspondences (where the ordering of the points *is* important).

Note that we are using four matching points in the DLT algorithm, and each point provides two constraints on the homography. However, there are nine numbers in the 3×3 homography matrix—recall that the a homography *is defined up to scale only* (any multiple of all the values in the homography is the same homography), and so you should normalize your matrix by scaling all of the entries such that the lower right entry in the matrix is 1.

Part 2: Bilinear Interpolation

With the perspective homography in hand, you can make use of the inverse warping and bilinear interpolation operations (discussed in the Szeliski text) to determine the best pixel value from the overhead image to map onto a corresponding pixel in the rover navigation image. Note that the overhead image is in colour (it has three bands: R, G, and B), and so the same transform must be applied to each band. For this part of the project, you should submit:

- a single function in `bilinear_interp.py`, which performs bilinear interpolation to produce a pixel intensity value, given an image and a subpixel location (point).

Part 3: Alpha Blending

Navigation overlays are easier for human operators to interpret when a portion of the background 'shows through,' that is, when the overlaid image is semi-transparent. This effect can be achieved through *alpha blending*, where the background (rover camera) image is blended with the foreground (warped, false-colour) image. Alpha blending can be accomplished using the following (per-pixel) formula:

$$I_o(x, y) = \alpha I_f(x, y) + (1 - \alpha) I_b(x, y)$$

where $I_f(x, y)$ is the intensity value at pixel location (x, y) in the foreground image, $I_b(x, y)$ is the corresponding intensity value in the background image, $I_o(x, y)$ is the resultant intensity value, and α is the blending factor. For this part of the project, you should submit:

- a single function in `alpha_blend.py`, which performs alpha blending of a foreground image and a background image to produce a composite image. The input images will be 8-bit greyscale or RGB (i.e., single-band or three-band).

Part 4: Terrain Overlays for Navigation

You're now ready to perform the complete overlay operation! Using the components you've built, you should: compute the perspective homography (once) that defines the warp between the rover navigation image and the false-color map, and then perform bilinear interpolation over all of the corresponding pixels to produce the output image. Before assigning the final pixel intensities, you should apply alpha blending.

Some portions of the code have already been filled in for you; in particular, the bounding box for the false-colour map, and the four pixel-to-pixel correspondences between the images, are available. For this (final) part of the project, you should submit:

- a single function in `terrain_overlay.py`, which uses the other functions above to produce the composite overlay that is fit for human consumption.

The composite image must be stored in colour and must be exactly the same size as the original rover navigation image (in terms of rows, columns, i.e., do not change the image size).

Grading

Points for each portion of the project will be assigned as follows:

- Perspective homography DLT function – **15 points** (3 tests $\times$ 5 points per test)

  Each test uses a different set of point correspondences. The root of the sum of squared projection errors (compared to the reference homography) must be below 0.1 (pixels) to pass.

- Bilinear interpolation function – **10 points** (5 tests $\times$ 2 points per test)

  Each test relies on a varied reference image and a different point location in that image. The absolute value of the brightness difference (between the reference, known brightness and the interpolated brightness) must be less than or equal to 1 to pass (e.g., if the reference value is 212, your function must report 211, 212, or 213 to pass the test).

- Alpha blending function – **5 points** (2 tests $\times$ 2.5 points per test)

  There are two tests, each using different input images. To pass each test, the pixel values in the blended image must match the pixel values in our reference solution exactly (since the alpha blending formula is very simple).

- Image composition script – **20 points** (4 tests $\times$ 5 points per test)

  There are three tests, each of which applies a more stringent criterion for matching between your overlay image and our reference solution (in terms of the absolute intensity difference between pixels in the warped region, evaluated by the mean and standard deviation). For now, the exact threshold parameters are being kept under wraps—if your support functions are working correctly, you should be able to pass the hardest test!

Total: **50 points**

Grading criteria include: correctness and succinctness of the implementation of support functions, proper overall program operation and code commenting, and a correct composite image output (subject to some variation). Please note that we will test your code *and it must run successfully*. Code that is not properly commented or that looks like 'spaghetti' may result in an overall deduction of up to 10%.