Division of Engineering Science
UNIVERSITY OF TORONTO

ROB501: Computer Vision for Robotics

Project #1: Image Transformations and Billboard Hacking

Fall 2021

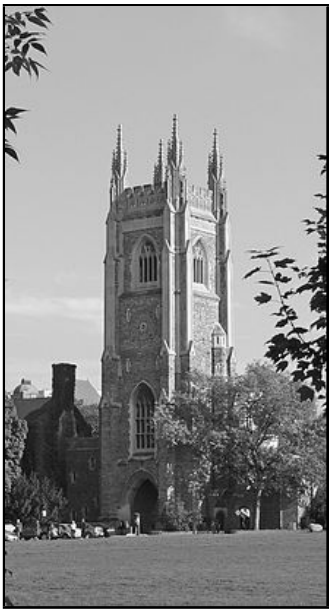
Overview

In this project, you will gain experience with the perspective transformation operation (discussed in the lectures), bilinear interpolation, and histogram equalization. You will use the perspective transform to replace a portion of an existing image with an alternate image. The goals 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 insert one image into another (respecting the appropriate geometry); and
- apply histogram equalization to improve the overall appearance (contrast) of an image.

The due date for project submission is **Thursday, October 3, 2021, by 11:59 p.m. EDT**. All submissions will be in Python 3 via Autolab (more details will be provided in class 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) Yonge & Dundas Square



(b) Soldiers' Tower

Your main project task is to perform some billboard hacking (this is a basic demonstration of the use of computer vision, and shows that it can be fairly easy to change 'reality'). There are two images above: image (a) is of Yonge and Dundas Square, an area that contains several large billboards, while image (b) is of Soldiers' Tower on the University of Toronto campus. Conveniently, the image of Yonge and Dundas Square has very limited radial distortion, which makes it suitable for our purposes. Your assignment (should you choose to accept it) is to replace the billboard advertisement for the "CN Tower Edge Walk" with the photo of Soldiers' Tower, such that the result looks *natural* (i.e., like the image of Soldiers' Tower is meant to be there). The project 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 view 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) Soldiers' Tower image (`uoft_soldiers_tower_dark.png`) to the appropriate coordinates in the Y&D Square image (`yonge_dundas_square.png`), and vice versa. 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, `dlt_homography.py`, which computes the perspective homography between two images, given four point correspondences (note that 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 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 lectures and in the Szeliski text) to determine the best pixel value from the Soldiers' Tower image to replace a pixel value in the Y&D Square image. Note that the Y&D Square image is in colour (it has three bands: R, G, and B), and so the same transform must be applied to each band (the Soldiers' Tower image is a greyscale image). For this part of the project, you should submit:

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

Part 3: Histogram Equalization

You will notice that the image file provided for this portion of the project, `uoft_soldiers_tower_dark.png`, is quite dark and has relatively low contrast. To fix this, you should implement the simple (discrete) histogram equalization algorithm discussed on page 107 of the Szeliski text (and in the course lectures). For this part of the project, you should submit:

- a single function in `histogram_eq.py`, which performs discrete histogram equalization on the input image (which will be 8-bit and greyscale).

Part 4: Billboard Hacking

You're now ready to perform the billboard hack! Using the components you've built, you should: enhance the contrast of the Soldiers' Tower image, compute the perspective homography (once) that defines the warp between the Y&D Square image and the Soldiers' Tower image, and then perform bilinear interpolation over all of the corresponding pixels to place Soldiers' Tower in the billboard position. Some portions of the code have already been filled in for you; in particular, the bounding box for the Edge Walk billboard, 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 `billboard_hack.py`, which uses the other functions above to produce the composite, 'hacked' image.

The composite image must be stored in colour and must be exactly the same size as the original Y&D image (in terms of rows and 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 square 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 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).

- Histogram equalization function – **10 points** (2 tests; 1 point and 9 points)

  There are two tests, one using the darkened version of the Soldiers' Tower image, and one using a hidden reference image (see points allocated above). To pass either test, only 10% or less of the equalized pixel intensity values may be greater than 3 units away from the reference intensity values (this is a fairly generous bound).

- Image composition script – **15 points** (3 tests $\times$ 5 points per test)

  There are three tests, each of which applies a more stringent criterion for matching between your hacked image and the 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%.