Intro to Image Understanding (CSC420)

Available: November 13^{th}, 2020
Due Date: December 6^{th}, 10:59:00 pm, 2020
Total: 150 marks

Notes: You are encouraged to become familiar with the **Python** environments to use it for the experimental parts of the assignment. We EXPECT EVERYONE TO SUBMIT **ORIGINAL WORK** FOR ASSIGNMENTS. This means that if you have consulted anyone or any sources (including source code), you must disclose this explicitly. Anything you submit must reflect your work.

Submission Instructions:

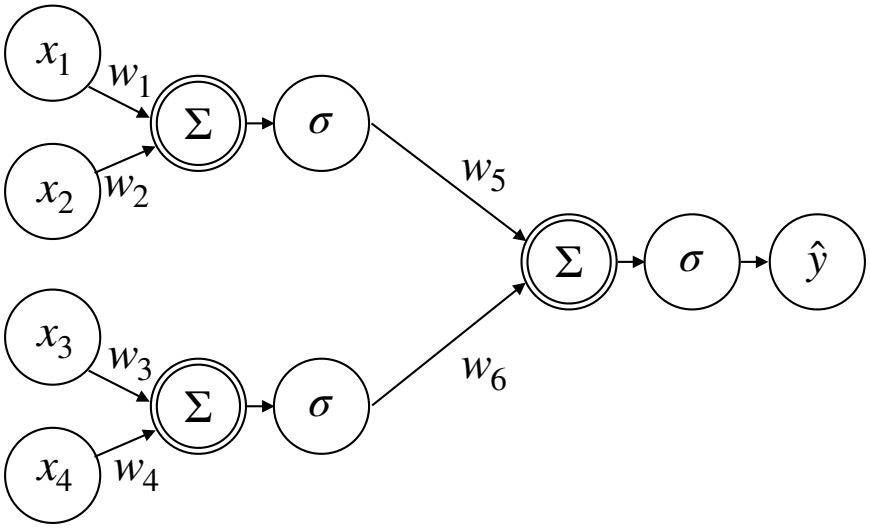
- Your submission should be in the form of an electronic report (PDF), with the answers to the specific questions (each question separately), and a presentation and discussion of your results. For this, please submit a file called "**report.pdf**" to MarkUs directly.
- Submit documented codes that you have written to generate your results separately. Please store all of those files in a folder called Assignment2, zip the folder, and then submit the file "**Assignment4.zip**" to MarkUs. You should also include a **"README.txt"** file (inside the Assignment 4 folder) which details how to run the submitted codes.
- Don't worry if you realize you made a mistake after submitting your zip file: you can submit multiple times on MarkUs. We always only mark the last submission.

Q1) Deep Learning (15 marks)

Consider the neural network architecture shown in the Figure below. Nodes denoted by x_i are input variables, and $\hat{y}$ is the output variable). The node Σ takes the sum of its inputs, and σ denotes the logistic function

$$\sigma(x) = \frac{1}{1+e^{-x}}.$$

Suppose the loss function used in training this neural network is an **L2** loss, i.e. $L(y, \hat{y}) = ||y - \hat{y}||_2^2$. Assume that the network is given a data point $(x_1, x_2, x_3, x_4) = (0.9, -1.1, -0.3, 0.8)$ with weights $(w_1, w_2, w_3, w_4, w_5, w_6) = (0.75, -0.63, 0.24, -1.7, 0.8, -0.2)$ true label 0.5. Compute the partial derivative $\frac{\partial L}{w_3}$, by using the back-propagation algorithm.
Hint: the gradient of an **L2** loss function $||y - \hat{y}||_2^2$ is $2||y - \hat{y}||$, and you do not need to write codes for this question! You can do it by hand.



Q2) Camera Models (35 marks)

Assume a plane passing through point $\vec{P_0} = [X_0,\ Y_0,\ Z_0]^T$ with normal $\vec{n}$. The corresponding vanishing points for all the lines lying on this plane form a line, called the horizon. In this question, you are asked to prove the existence of the horizon line by following the steps below:

1. **(15 marks)** Find the pixel coordinates of the vanishing point corresponding to a line L passing point $\vec{P_0}$ and going along direction $\vec{d}$.

   Hints: $\vec{P} = \vec{P_0} + t\,\vec{d}$ are points on line L, and $\vec{p} = \begin{pmatrix} \omega x \\ \omega y \\ \omega \end{pmatrix} = K\,\vec{P} = K \begin{pmatrix} X_0 + t\,d_x \\ Z_0 + t\,d_z \\ Z_0 + t\,d_z \end{pmatrix}$ are pixel coordinates of the same line in the image, and $K = \begin{pmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{pmatrix}$, where f is the camera focal length and is (p_x, p_y) is the principal point.

2. **(20 marks)** Prove the vanishing points of all the lines lying on the plane form a line.

   Hint: all the lines on the plane are perpendicular to the plane's normal $\vec{n}$; that is, $\vec{n} \,.\, \vec{d} = 0$, or $n_x\, d_x + n_y\, d_y + n_z\, d_z = 0$

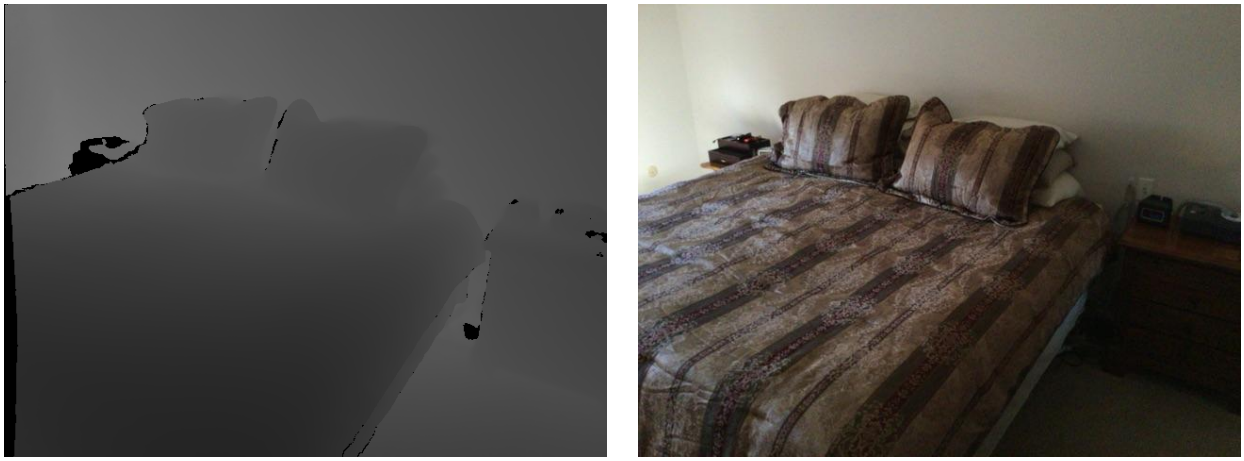
Q3) Projection (40 marks)

*For this Querstion, you are provided with set of input data (in a zip folder called **A4Q3.zip**). Please extract the package and follow the instructions below. Make sure to run your code on all the examples provided in the zip file.*

For this question we will elucidate the mapping from world to camera to pixel coordinates, using the matrix formulation we have used in class, as well as explore the consequences of certain extrinsic operations, in particular, rotations. You will be provided with rich data sets, each of which, for a particular 3D scene, contains the following:

1. A grayscale *depth* image of a particular size where at each pixel location one has a value proportional to the depth of the projected 3D scene point. Thus brighter points are further away from the camera. An example is shown in Figure 1(a).

2. A color image of the same size, which is registered to the above depth image. This should be interpreted as the appearance image which contains the red, green and blue channels of image irradiance for each corresponding 3D scene point. An example is shown in Figure 1(b).



a b

Figure 1: An example set of input images (grayscale depth map and color image).

3. A text file containing the intrinsic parameters, i.e., the camera calibration matrix **K** we discussed in class.

4. A text file containing the camera extrinsic parameters, i.e., the $R[I|-C]$ part of the finite projective camera model developed in class.

a) (15 marks) Your first task is to write a function (let us call that **"compute_point_cloud"**) which, for any pixel in the depth image, provides the coordinates of the associated 3D scene point (X, Y, Z) and the associated color channel values. You can assume that the world coordinate frame corresponds to the standard Cartesian frame with basis vectors $(1, 0, 0), (0, 1, 0), (0, 0, -1)$ corresponding to the x, y, and z directions. This function provides the coordinates of the associated 3D scene point (X, Y, Z) and the associated color channel values for any pixel in the depth image. You should save your output in an output file called **"pointCloud.txt"** inside the folder of the considered example besides the other files (**"depthImage.png"**, **"rgbImage.jpg"**, etc.) in the format of a N x 6 matrix where N is the number of 3D points with 3 coordinates and 3 color channel values:

```
X_1,Y_1,Z_1,R_1,G_1,B_1
X_2,Y_2,Z_2,R_2,G_2,B_2
X_3,Y_3,Z_3,R_3,G_3,B_3
X_4,Y_4,Z_4,R_4,G_4,B_4
X_5,Y_5,Z_5,R_5,G_5,B_5
X_6,Y_6,Z_6,R_6,G_6,B_6
.
.
.
.
.
X_N,Y_N,Z_N,R_N,G_N,B_N
```

b) (25 marks) You will now consider the 3 explicit rotations of the camera coordinate system: a pure rotation about its y-axis (pan), a pure rotation about its x-axis (tilt), and a pure rotation about its z-axis. These rotations are modeled by Ωt where Ω is a fixed rotation angle about one of the x, y, or z axes per step and t is the number of steps. Write a function which rotates a 3D scene point cloud (X, Y, Z) about the x, y, or z axes (as desired) by an amount Ωt (applies a sequence of rotations of Ω for t times). This function should output the new coordinates of the 3D scene point with respect to the camera reference frame. This function should accept an image input, a rotation value Ωt between 0 to $\pi/2$ (make sure to use radians not degrees) and a rotation axis which is either **'x'** or **'y'** or **'z'**. The function should produce two outputs: a data array (image) that contains the projected depth value (grayscale) and a data array (image) that contains the projected color of the 3D scene point. Now use this function along with the intrinsic parameters to consider a sequence of rotations $\Omega t \in (0, \pi/2)$ about the x, y, or z axes (as desired) of each 3D scene point, followed by a projection to create two images: 1) an image that contains the projected depth value (grayscale) and 2) an image that contains the projected color of the 3D scene point. In fact, for each scene and choice of axis, you will be creating two sequences of images. You can use existing libraries from Python or Matlab to create a movie out of each projected images. Just Google **How to make a video with Python (or Matlab)**. When viewed as movies, these sequences should look like the expected cases of panning, tilting or rolling the camera.

Q4) Homography (60 marks)

You are given three images `hallway1.jpg`, `hallway2.jpg`, `hallway3.jpg` which were shot with the same camera (i.e. same internal camera parameters), but held at slightly different positions / orientations (i.e. with different external parameters).

`hallway1.jpg` `hallway2.jpg` `hallway3.jpg`

Consider the homographies **H**,

$$\begin{bmatrix} \widetilde{w}\widetilde{x} \\ \widetilde{w}\widetilde{y} \\ \widetilde{w} \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

that map corresponding points of one image I to a second image $\widetilde{I}$, for three cases:

A. The right wall of I =`hallway1.jpg` **to the right wall of** $\widetilde{I}$ =`hallway2.jpg`.
B. The right wall of I =`hallway1.jpg` **to the right wall of** $\widetilde{I}$ =`hallway3.jpg`.
C. The floor of $\widetilde{I}$ =`hallway1.jpg` **to the floor of** $\widetilde{I}$ =`hallway3.jpg`.

For each of these three cases:

1. **(5 marks)** Use a Data Cursor to select corresponding points by hand. Select more than four pairs of points. (Four pairs will give a good fit *for those points*, but may give a poor fit for other points.) Also, avoid choosing three (or more) collinear points, since these do not provide independent information. This is trickier for case **C**.

2. **(15 marks)** Fit a homography **H** to the selected points. State what **H** is, and describe its effect using words such as *scale, shear, rotate, translate*, if appropriate.

3. **(5 marks)** Make a **figure** showing the I image with a colored square marking each of the selected points. In particular, for this question and the next one, convert the image I or $\widetilde{I}$ to grey level using an RGB to Gray function (or the formula $gray = 0.2989 \times R + 0.5870 \times G + 0.1140 \times B$), and plot colored squares using a plot command e.g. `plot(x,y,'rs')` in Matplotlib Library.

4. **(5 marks)** Make a **figure** showing the $\widetilde{I}$ image with red squares that mark each of the selected $(\widetilde{x}, \widetilde{y})$, and green squares that mark the locations of the estimated $(\widetilde{x}, \widetilde{y})$, that is, use the homography to map the selected (x, y) to the $(\widetilde{x}, \widetilde{y})$ space.

5. **(20 marks)** Make a figure showing a new image that is larger than the original one(s). The new image should be large enough that it contains the pixels of the I image as a subset, along with *all* the inverse mapped pixels of the $\widetilde{I}$ image. The new image should be constructed as follows:

   - RGB values are initialized to zero,
   - The red channel of the new image must contain the `rgb2gray` values of the I image (for the appropriate pixel subset only);
   - The blue and green channels of the new image must contain the `rgb2gray` values of the corresponding pixels $(\widetilde{x}, \widetilde{y})$ of $\widetilde{I}$. The correspondence is computed as follows: for each pixel (x, y) in the new image, use the homography $\mathbf{H}$ to map this pixel to the $(\widetilde{x}, \widetilde{y})$ domain (not forgetting to divide by the homogeneous coordinate), and round the value so you get an integer grid location. If this $(\widetilde{x}, \widetilde{y})$ location indeed lies within the domain of the $\widetilde{I}$ image, then copy the `rgb2gray`'ed value from that $\widetilde{I}(\widetilde{x}, \widetilde{y})$ into the blue and green channel of pixel (x, y) in the new image. (This amounts to an inverse mapping.)
     If the homography is correct and *if the surface were Lambertian** then corresponding points in the new image would have the same same values of R,G, and B and so the new image would appear be grey at these pixels.
   - Based on your results, what can you conclude about the relative 3D positions and orientations of the camera ? (Give only qualitative answers here.) What can you conclude about the surface reflectance of the right wall and floor, namely are they more or less Lambertian ?

(10 marks) Along with your writeup, hand in a program that you used to solve the problem. You should have a switch statement that chooses between cases **A, B, C**.

* *Lambertian reflectance is the property that defines an ideal "matte" or diffusely reflecting surface. The apparent brightness of a Lambertian surface to an observer is the same regardless of the observer's angle of view. Unfinished wood exhibits roughly Lambertian reflectance, but wood finished with a glossy coat of polyurethane does not, since the glossy coating creates specular highlights. Specular reflection, or regular reflection, is the mirror-like reflection of waves, such as light, from a surface. Reflections on still water are an example of specular reflection.*