CS 223b Homework 1 (Due January Wednesday, 23th)

J. Košecká

e-mail your answers to cs223b@gmail.com

Real-world implementation is essential in computer vision. Vision algorithms are often implemented in software using either MATLAB or Intels Open Computer Vision (OpenCV) library for C/C++. The purpose of this section is to gain an initial familiarity with these two environments. We ask that you implement the three vision algorithms below. To encourage breadth, two of the first three algorithms must be implemented in OpenCV. Your code must be clearly structured and well-commented. If obtaining access to MATLAB is an issue for you, please email the course staff as soon as possible.
FAQ: Use gray-scale images where each pixel has a single intensity on [0:255].
FAQ: It is up to you which algorithms you implement in OpenCV and which you implement in MATLAB. Sometimes OpenCV will be a cleaner choice than MATLAB or vice-versa. This will not affect your grade.
FAQ: To check your work, you are free to implement each algorithm in every environment.

1. (10) **The Log-Polar Transform** The log-polar transform (see Figure 1) is a simple operation that changes the coordinate system of an image from Cartesian to log-polar. Log-polar coordinates have several interesting properties. First, when optical flow is computed as the camera moves along the optical axis (as is common in robotics), the optical flow vectors are radial from the image center. However, in log-polar coordinates, the flow vectors are vertical. This can make the detection of moving objects more efficient computationally. Second, the log polar transform converts an image to a form that is rotation and scale invariant. This can be very helpful for object detection. Finally, the log-polar transform emulates how images appear on the back of the human retina. (The brain then transforms the image to the Cartesian-like way we perceive it.) The log-polar transform is `destination_image[`ϕ, ρ`] = source_image[x, y]` where $\phi = tan^{-1}(y/x)$ and $rho = 60 * \log_e \sqrt{x^2 + y^2}$. Leave any out-of-bounds pixels black. Implement the transform on the image http://cs223b.stanford.edu/homework/ass1/1.1/input.png. Submit your result as a PNG and the source code you used to generate it. An example of the log-polar transform (on a different image) is included at the end of this document.

2. (10) **Histogram Equalization** Often, we encounter an image whose dynamic range (ie: contrast) is compressed (see Figure 2). For example, in an 8-bit gray-scale image, only a narrow range of the 256 possible intensity values might be used. In that case, an image can contain a significant amount of detail that is not apparent visually. One approach for enhancing detail is called histogram equalization. It is straightforward to perform. First, we generate a histogram H of the intensities in the image. Specifically, we have one bin for each intensity 0-255. The value in each bin is the number of pixels in the image with that intensity. Second, we normalize the histogram such that the sum of the 256 values in bins 0-255 is 255. Third, we generate a second histogram H' where $H'[i] = \sum_{0 \leq j \leq i} H[j]$ for all $0 \leq i \leq 255$. Finally, `destination_image[x, y] = H'[source_image[x, y]]`. Implement the transform on the image http://cs223b.stanford.edu/homework/ass1/1.3/input.png. Submit your result as a PNG and the source code you used to generate it. An example of histogram equalization (on a different image) is at the end of this document. FAQ: It is helpful to keep floating-point precision until you write the new image in the last step. Otherwise, integer rounding will throw you off.

3. (5) **Contrast Stretching** Another approach to detail enhancement in the face of dynamic range compression (see Section 1.3) is contrast stretching. Contrast stretching is even easier than histogram equalization: `destination_image[x, y] = (source_image[x, y] - image_min)*255/(image_max - image_min)` where image min and image max are the minimum and maximum intensity values present in the image. Implement the transform on the image http://cs223b.stanford.edu/homework/ass1/1.4/input.png. Submit your result as a PNG and the source code you used to generate it.

4. (10) **Perspective Projection** It is often useful while testing some of the algorithms to simulate the perspective projection process. Write a MATLAB function, which implements the image formation process. Write a function $x = \texttt{project}(\texttt{X}, \texttt{R}, \texttt{T}, \texttt{K})$ which takes as an input image coordinates of 3D points in the world coordinate frame and generates pixel coordinates of the projected points in the image, assuming that (R, T) is the displacement of the camera coordinate frame with respect to the world frame, K is the matrix of intrinsic image parameters, and X is a $3 \times n$ vector of the coordinates of 3D points. To test the function consider a unit cube placed in the origin of the world coordinate system (specified by 8 vertices $[0, 0, 0]', [1, 0, 0]', etc$, assume that the camera is translated along z-axis by some amount and rotated around x-axis by angle 20∘. You can assume that matrix K is $[800, 0, 250; 0, 800, 250; 0, 0, 1]$. Generate the image of the cube. Its enough when you plot the vertices of the cube and optionally connect them by line to visualize it better. Submit the MATLAB code and generated MATLAB figure. It is commonly assumed that in the coordinate system of the camera the z-axis if pointing towards the scene and y down and x to the right.

   Solution:

```
function xim = project(X,R,T,K)


np = size(X,2);
xim = K*(R*X + T*ones(1,np));
xim(1,:)  = xim(1,:)./xim(3,:);
xim(2,:)  = xim(2,:)./xim(3,:);
xim(3,:)  = xim(3,:)./xim(3,:);
```

5. (10) **Rigid Body Transformations** Consider rigid body transformations in the plane. Draw a right triangle defined by three points $A = (2, 1), B = (4, 1), C = (4, 6)$.

   - Consider a rotation matrix
   $$T_1 = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

   a. What is the determinant of the matrix ? Determinant of the matrix is 1.

   b. Apply the rotation matrix to the triangle and show the result.

   - Consider transformation matrix
   $$T_2 = \begin{bmatrix} \sin\theta & \cos\theta \\ \cos\theta & -\sin\theta \end{bmatrix}$$

   a. What is the determinant of the matrix ? Determinant of the matrix is -1.

   b. Apply the transformation matrix to the triangle and show the result. Is T_2 rigid body transformation (i.e. can you move the triangle from initial position to the final without leaving the plane ? Yes T_1 is a rigid body transformation. What is the difference between T_1 and T_2, how are the results different? T_2 is not a rigid body transformation, because the determinant of the T_2 is -1, hence T_2 is not a rotation matrix. T_2 is a mirror reflection.

6. (5) **Rigid body transformations composition** Suppose that you are given the relative displacement between the coordinate frame {1} and {2}, $g_{21} = (R_{21}, T_{21})$ expressed in the frame {2} and relative displacement between the frame {3} and the frame {2} expressed in the frame {2}, and denoted by $g_{23} = (R_{23}, T_{23})$. What is the relative displacement g_{31}, between the frame {1} and frame {3} expressed in the frame {1} ?. a) Draw a figure; and write g_{31} in terms of given transformations/displacements. b) Write down explicitly what is the rotational and translational part of g_{31}, in terms of given rotations R_{ij} and $T'_{ij}s$.

Solution: We are given:

$$X_2 = R_{21}X_1 + T_{21} \quad (1)$$
$$X_2 = R_{23}X_3 + T_{23} \quad (2)$$

and we want the relationship between X_3 and X_1.

$$X_3 = R_{23}^T X_2 - R_{23}^T T_{23} \text{ by inverting eq. [2]} \quad (3)$$
$$X_3 = R_{23}^T R_{21} X_1 + R_{23}^T T_{21} - R_{23}^T T_{23} \text{ by composition of rigid body motions} \quad (4)$$

Hence the answer is $R_{31} = R_{23}^T R_{21}$ and $T_{31} = R_{23}^T T_{21} - R_{23}^T T_{23}$

Note: Commonly used convention for rigid body motion when multiple frames are involved is for (R_{21}, T_{21}) to be the transformation which transforms coordinates from frame 1 to frame 2, where T_{21} is the coordinate of the origin of the frame 1 expressed in the frame 2 and R_{21} is the rotation of frame $\{1\}$ wrt. frame $\{2\}$. The notation in the original problem statement was inconsistent with this convention as it stated that (R_{21}, T_{21}) was expressed in frame $\{1\}$, which would change the first equation (and the rest accordingly) to $X_1 = R_{21}X_2 + T_{21}$. The composition would then be $R_{31} = R_{21}R_{23}^T$ and $T_{31} = -R_{21}R_{23}^T T_{23} + T_{21}$.

7. (10) **Vanishing Point** Straight line in 3D world is projected in to a straight line in the image. The projections of two parallel line intersect in the image at so called *vanishing point*.

   a) Show (mathematically) that projections of parallel lines in the image intersect in a point.

   b) Compute for a certain family of parallel lines where in the image will the vanishing point be.

   c) When does the vanishing point of the lines in the image lie at infinity (i.e. they do not intersect)?

   Solution: Assume parametric equation of a line expressed in the camera coordinate frame where $\mathbf{X_0} = [X_0, Y_0, Z_0]^T$ is a point on the line (base point) and $\mathbf{v} = [v_1, v_2, v_3]^T$ is a unit vector representing the direction of the line, we have $\mathbf{X} = \mathbf{X_0} + \lambda\mathbf{v}$. For any point on the line its projection is (assuming focal length is 1 and the optical axis is aligned with z-axis)

$$x = \frac{X_0 + \lambda v_1}{Z_0 + \lambda v_3} \text{ and } y = \frac{Y_0 + \lambda v_2}{Z_0 + \lambda v_3}$$

   To analyze the behavior of the projection is to see how the projection of a point varies at $\lambda \to \infty$. This behavior will depend only on the direction of the line, not the base point. First divide both nominator and denumerator by λ

$$x = \frac{X_0/\lambda + v_1}{Z_0/\lambda + v_3} \text{ and } y = \frac{Y_0/\lambda + v_2}{Z_0/\lambda + v_3}$$

   a) If we set $\lambda \to \infty$ the point all parallel lines have to go through is

$$x = \frac{v_1}{v_3} \text{ and } y = \frac{v_2}{v_3}$$

   b) If $v_3 \neq 0$ then the projection is a finite point. Note that since the projection of the point at infinity does not depend on the base point, all lines which have the same direction will meet in the image at the same point.

   c) If $v_3 = 0$ then the denominator becomes 0 and the coordinates of the projection go to ∞, i.e. when directions of a set of parallel lines do not have any component along z-axis (or in other words are parallel to the image plane) the projection of their intersection does not lie in the image.

Figure 1: Log-polar transformation: before and after

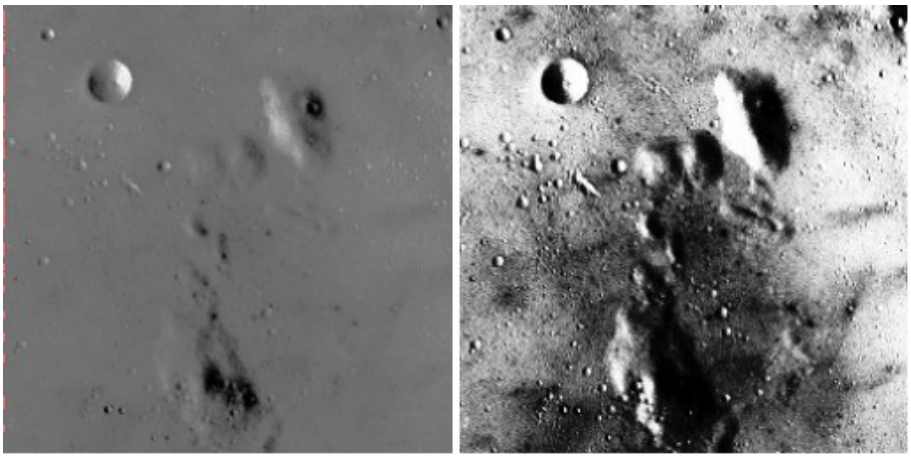

Figure 2: Histogram equalization: before and after