ENGN2560 Computer Vision
Lab05: Correspondences from Stereo Vision

The goal of this lab is to:

- Learn how correspondences can be established from small baseline stereo vision with unknown camera poses using optical flow algorithm.
- Learn how correspondences can be established from wide baseline stereo vision with known camera poses using inverse depths and epipolar geometry.

Problem 1. Multi-Resolution Optical Flow In the lecture, we learned that Lucas-Kanade optical flow algorithm finds the flow of a point from an image to another image based on the assumption of photometric consistency. Assume that the brightness is constant across images, Lucas-Kanade optical flow is however useful when the baseline of the two images is very small. To provide a robust optical flow estimation, a coarse-to-fine solution is often used in practice, where the points are first up-scaled from the original resolution (coarse) to low resolution (fine), and then find their flows from low resolution (fine) to the original resolution (coarse). This hierarchical process of an optical flow approach is called a multi-resolution optical flow. The goal of this problem is to implement this approach to track corner features using 6 consecutive images, Figure 1, by following the steps below. Implement your code in the provided `P1_main.m` file. (40 Points)

1. **Construct an Image Pyramid for the First View:** A coarse-to-fine solution requires an image pyramid, a multiple levels of images with different scales (resolution), such that points can be tracked from the top level to the bottom level. An illustration is given in the blackboard pictures of the lecture. Let the original image `I{1}` be the lowest level of a pyramid. The scale of an image `I{2}` that is one level above is one quarter of the scale of `I{1}`, *i.e.*, the number of rows and columns are half of those in `I{1}`. You are allowed to use the MATLAB function `impyramid` to construct an image pyramid. Specifically, you can use `I{2} = impyramid(I{1}, 'reduce');` to generate `I{2}`. The parameter `PARAMS.LK_OPTICAL_FLOW_NUM_OF_LEVELS` defines how many levels you need to build. Construct an image pyramid for the first view.
2. **Detect Corner Features:** Use MATLAB function `detectHarrisFeatures` to find corner features on the first image.

Figure 1: (Left to right) images 1, 3, 5, and 6 of an image sequences where the baseline between every two images is small.

3. **Downscale Corner Feature Locations:** For each corner feature, scale down its location from the bottom levtl to the top level of the pyramid. Specifically, multiply the x and y coordinate of corner feature locations by a factor of 0.5.

4. **Track Corners by Multi-Resolution Lucas-Kanade Optical Flow:** Loop over all the rest of the views in the image sequence. For each view,

   (a) Build an image pyramid of that view as you did in the first step.

   (b) Create a second loop which starts from the top level image to the bottom level image. In each level,

      i. Track all corner features via Lucas-Kanade optical flow algorithm, provided by the `LucasKanadeOpticalFlow.m` file. Carefully read the instructions in that file to properly structure your input data. The output is an array of displacements in x and y directions of all corner features.

      ii. Add the displacements returned by the `LucasKanadeOpticalFlow` function to the corner feature locations to get updated corners.

      iii. Check if the corner flows outside the image boundary. If so, remove those corners.

      iv. Upscale the updated corner locations by the same scale you used when building the image pyramid. Specifically, multiply the x and y coordinate of corner feature locations by a factor of 2. These up-scaled corners are used in the next loop as the initial locations for the Lucas-Kanade algorithm in step *i*.

      v. Repeat steps *i* to *iv* until all levels are visited.

   (c) Now the corners are flown from one image to its next image. They are located in the bottom level of the image pyramid. To continue flowing to subsequent images, downscale the corner feature locations again as you did in step 3 so that they are located in the top level of the image pyramid.

   (d) Repeat steps (a)-(c) until the last image is reached.

5. **Visualize the Corners Across Images:** Plot the corners on the first image and the last image. Make each corner correspondence an indivisual color. An example result is given in Figure 2(a). In addition, superimpose all corners from the six images to the first image which gives a clear observation of how the flows look like, *e.g.* Figure 2(b).

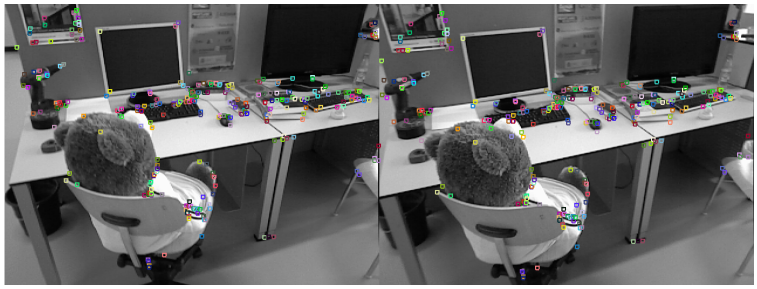
(a)  (b) 

Figure 2: (a) The corners are flown from the first image (left) to the last image (right) with corresponding colors. (b) When superimposing all corners from the six images to the first image, the flows of the corners can be easily observed.

Answer the following questions:

1. By visual inspection, are all the corner features flowing to their correct positions from the first image to the last image? If not, what factors do you think might contribute to the problem? (5 Points)

2. Reduce the number of levels in your image pyramid to 1, *i.e.* we do not do a coarse-to-fine approach but enforce the optical flow to work on the original image directly. Run your code again and visualize the results. Which approach is better? (5 Points)

Problem 2: Stereo Matching Based on Inverse Depths Without known camera poses, Lucas-Kanade optical flow allows us to find correspondences across very small baseline stereo images. Now, assume that the camera poses are known, given a wide baseline stereo image, how can we establish correspondences? In this problem, we will answer this question using the projections from inverse depth samples, and construct correspondences based on the photometric consistency constraint.

As illustrated in Figure 3, given relative pose $(\mathcal{R}, \mathcal{T})$ between two images I_1 and I_2, a point γ_1 on image I_1 gives rise to a 3D point $\rho_1\gamma_1$ which can be projected to the second image I_2. If the depth ρ_1 is known, then γ_2 can be easily found, constructing a correspondence (γ_1, γ_2). However, without this prior information, an approach is to first sample the depth values (black dots on the line ρ_1 in Figure 3) to generate a set of 3D points $\rho_1\gamma_1$, and then project those 3D points to the second image. The projected point γ_2 is in correspondence to γ_1 if they satisfy the photometric consistency constraint, *i.e.*, $I_1(\gamma_1) = I_2(\gamma_2)$.

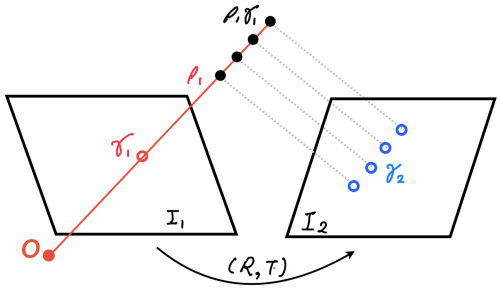


Figure 3: An illustration of finding a correspondence of γ_1 on image I_1 from many γ_2 on image 2 as a set of points projected from $\rho_1\gamma_1$ with various values of ρ_1, given know relative pose $(\mathcal{R}, \mathcal{T})$.

An example of a set of projections from $\rho_1\gamma_1$ using uniformly sampled depth values are demonstrated in Figure 4 (a) and (b), where a point γ_1 on a dice on image 1, Figure 4(a), gives rise to multiple $\rho_1\gamma_1$ with different ρ_1 values which are projected to the second images, Figure 4(b). The depth values are sampled every 10 cm from 1 meter to 5 meters. You may

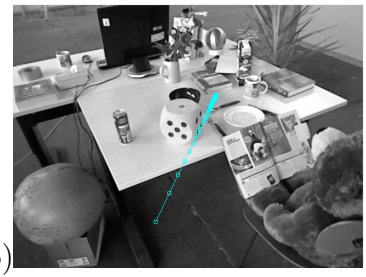
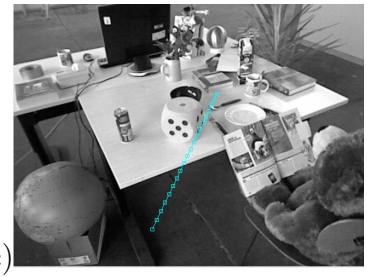
(a)  (b)  (c) 

Figure 4: (a) A point (cyan) on the dice of the first image gives rise to many 3D points by varying depths. The uniformly sampled depths in the representation space (3D space) creates a log sampling of projections γ_2 in the observation space (the second image) in (b), whereas the sampling of inverse depths in the representation space gives uniform projections on the observation space in (c).

observe that the projections are not uniformly sampled, but are rather log sampled! This is where the *inverse depths* come into play: when parameterize the depth by its inverse, the sampling of every 10 cm becomes a log sampling rate in the representation space, which attributes to uniform sampling in the observation sapce in image 2, Figure 4(c).

Using the projections γ_2 on the second image, we can find the best match to γ_1 based on photometric consistency. In this problem, you are tasked to implement this to find the correspondences of corner points in image 1 to image 2 via inverse depths. Specifically, follow the steps below: (30 Points)

1. **Detect Corner Features:** Use MATLAB function `detectHarrisFeatures` to find corner features on the first image. You have done this in the previous problem.

2. **Pad Two Images:** Rather than comparing $I_1(\gamma_1)$ and $I_2(\gamma_2)$ to check their photometric consistency, a robust approach is to find the consistency of values in a $w \times w$ *window* (or a *patch*) attached to each point. The patch size w is defined by `PARAMS.PATCH_SIZE`. However, if there is a corner feature located near the image boundary, the corresponding windowed values can not be accessed if the window covers areas outside the image boundary. This can be resolved by padding the image before doing the rest of the steps. See how images can be padded from the provided `LucasKanadeOpticalFlow.m` file, lines 26-28.

3. **Create `griddedinterpolant` Structures for Each Image:** Since the corners on image 1 and the projected points on image 2 are in subpixel locations, their photometry values can be acquired by interpolating the photometry of their nearby pixels. We will use `griddedinterpolant` structures for the two images to quickly do photometry interpolation. See how this can be done in the provided `LucasKanadeOpticalFlow.m` file, lines 35-39.

4. **Find Correspondence of Each Corner:** Loop over all corners γ_1 on image 1. In each loop,

   (a) Get the coordinates of the patch attached to the corner point γ_1. See how this can be done in the provided `LucasKanadeOpticalFlow.m` file, lines 48-54.

   (b) Acquire the interpolated photometric values of the patch. See line 57 of the provided `LucasKanadeOpticalFlow.m` file.

   (c) Loop over all sampled inverse depth values defined by `PARAMS.DEPTH_SAMPLES`. For each depth ρ_1,

      i. Compute $\rho_1\gamma_1$ and project it to the second image using the provided camera intrinsic matrix and the relative pose. Denote the projected point as γ_2.

      ii. Check if γ_2 is within the image boundary. If not, jump to the next ρ_1 and continue the loop.

      iii. Acquire the interpolated photometric values of the patch attached to γ_2. This is the same as step (b).

      iv. Calculate the average Sum of Squared Distance (SSD) value from the two patches of γ_1 and γ_2. Refer to the Appendix of how the average SSD can be calculated.

v. Check if the average SSD value is below a threshold defined by `PARAMS.SSD_THRESH`. If so, store the average SSD value and the corresponding index of the depth ρ_1.

vi. Repeat steps *i.* to *v.* until all sampled inverse depths are visited.

(d) Check whether there is at least one γ_2 that passes the average SSD threshold. If not, return to step (a). Else, for all candidates γ_2, the one with the smallest SSD is the corresponding point of γ_1.

(e) Repeat steps (a) to (d) until the end of the loop.

5. **Visualize the Correspondences:** Finally, show the matches of the corners. You can use the provided function `draw_Feature_Matches` from Lab02 to help the visualization. An example result is shown in Figure 5

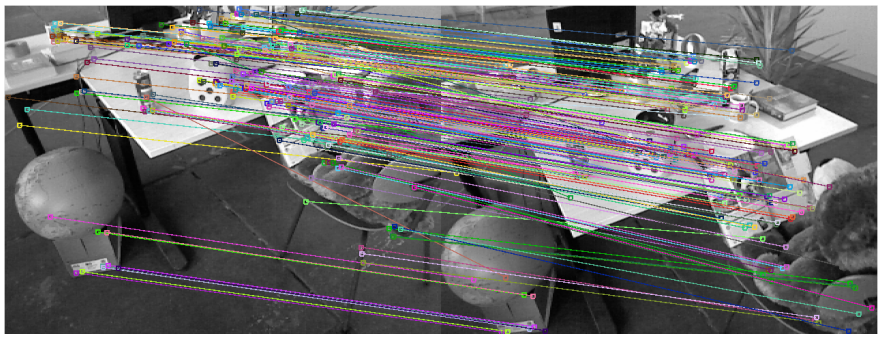


Figure 5: The correspondences of stereo images by projecting corners from the left image to the right image with varying inverse depth values.

Put your code in the provided `P2_main.m` file and show your matching result in your report.

Problem 3: Stereo Matching Based on Epipolar Geometry In problem 2, we have witnessed how corner features correspondences can be found across a pair of images based on photometric consistency. However, by visual inspection on Figure 5, a few mismatches can be observed. They are not interest points on the second image, *e.g.* the surface of a table or a chair. This raises a question of whether we can find correspondences from corners on image 1 and corners on image 2 based on photometric consistency. This problem thus tasked you to use epipolar geometry to find corner feature correspondences, *i.e.*, check the photometric consistency from corners on image 2 that lie on the corresponding epipolar line. We will use the same images from problem 2. Follow the steps below. (20 Points)

1. **Detect Corner Features and Pad Two Images:** Find corner features on *both* images using `detectHarrisFeatures`. Pad two images as you did in step (b) of problem 2.

2. **Create `griddedinterpolant` Structures for Each Image:** This is the same as step (c) of the previous problem.

3. **Find Correspondence of Each Corner:** Loop over all γ_1 on image 1. In each loop,

   (a) Get the coordinates of the patch and their corresponding photometry values of the corner point γ_1. This is the same as step 4(a) and 4(b) of the previous problem.

   (b) Compute epipolar line coefficients and the distances from the corners γ_2 on image 2 to the epipolar line. Take only the corners γ_2 with distance to the epipolar line smaller than a threshold defined by `PARAMS.POINT_TO_EPIPOLAR_LINE_DIST`.

   (c) Check if there is at least one γ_2 that passes the distance to epipolar line constraint. If not, return to step (a). Else, loop over all candidate corners γ_2. For each γ_2, do steps 4(c)-*iii.* to v from the previous problem.

   (d) Check whether there is at least one γ_2 that passes the average SSD threshold. If not, return to step (a). Else, for all candidates γ_2, the one with the smallest SSD is the corresponding point of γ_1.

   (e) Repeat steps (a) to (d) until the end of the loop.

4. **Visualize the Correspondences:** Finally, show the matches of the corners just like you did in the previous problem. An example result is shown in Figure 6

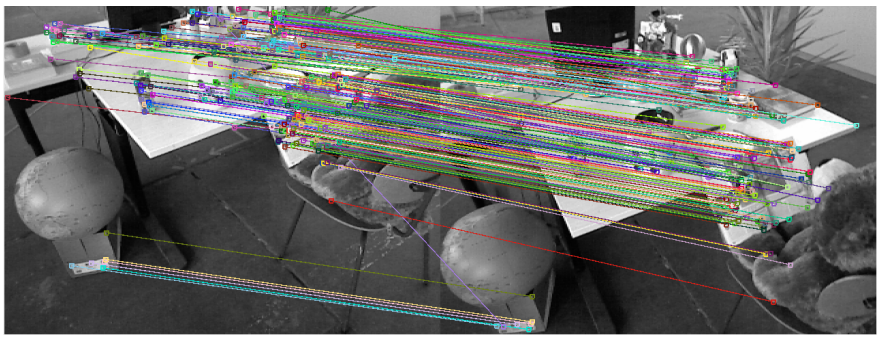


Figure 6: The correspondences of corners in a pair of images based on the epipolar geometry.

Appendix

Let $\gamma = (x, y)$ be a feature on image I, $\bar{\gamma} = (\bar{x}, \bar{y})$ be a feature on another image $\bar{I}$, and the image intensities of the window W attaching two features be $I_w(\gamma) = \{I(\xi, \eta) | (\xi, \eta) \in W(\gamma)\}$ and $\bar{I}_w(\bar{\gamma}) = \{I(\xi, \eta) | (\xi, \eta) \in W(\bar{\gamma})\}$. The average Sum of Squared Distance (SSD) measures the similarity between windowed γ and $\bar{\gamma}$:

$$SSD = \frac{1}{W^2} \sum_{i \in W} \sum_{j \in W} \left(I(x + i, y + j) - \bar{I}(\bar{x} + i, \bar{y} + j) \right)^2 \tag{1}$$