(Weeks 8-9)

TODO this week:

- Start thinking about your extension project
- Depth maps

Next week (if doing real space work as extension project):

- Real distances (triangulation)
- Calibration

Creating Depth Maps

3 Image Comparison

As we observed in the first part of the project, the cross correlation technique is a very powerful tool which can sensitively and accurately identify patterns in complex images. Consider the two views of New York as shown in Figure 4 below. As humans, most of us can easily recognise that both images are of the same scene, only taken from different angles or positions.

Your next task is to create a program which compares two images using cross correlation. For this part, you may choose to use any image pairs on the LMS.

Your program should:

1) Break up one image in to windows
2) Create a template from one window
3) Create a search region (larger than than the template) in the other image
4) Scan the template around the search region to find similar features using cross correlation
5) Return *dpx* and *dpy*, the difference in pixel location
6) Repeat this for all windows

***Note**: The term window refers to a small section of an image, like the orange box seen in Figure 4 & 5.*

Your function should have the structure:

```
[dpx,dpy] = myfn(imagea, imageb, wsize, xgrid, ygrid)
```

Where `wsize` is the window size in pixels, `xgrid` is the window centre pixel locations in the x direction, `ygrid` is the window centre pixel locations in the y direction.

Figure 4 Template created in left image (orange), search region created in right image (3 times larger than template, centered at the same pixel location). Source: wikipedia

Figure 5 Left image broken up in to windows

4 Cross Correlation Optimisation

There are several parameters, which need to be considered when applying the cross correlation technique. These include what is the ideal sized window to be used, and how best should it be scanned in search of matching. In this part of the project, we will investigate three optimisation strategies:

a) Window overlap
b) Search region
c) Multiple pass

Enhance the program that you have developed for part 3 above to implement each of these strategies. More details are provided below.

Optimisation strategies:

a) Variable window overlap.

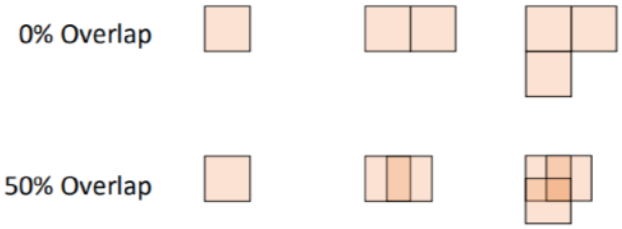

Figure 6 Example of Variable Window Overlap

b) Variable search region geometry.
Some examples of different search regions are shown below

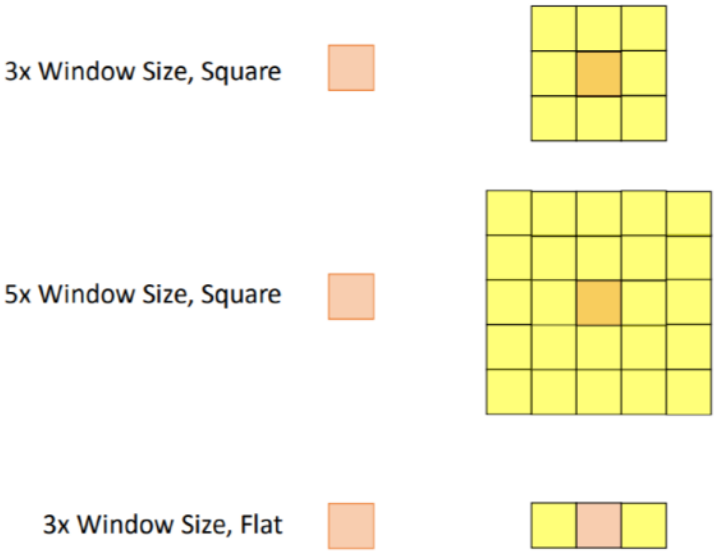

Figure 7 Example of Variable Search Region

c) Multi-Pass Cross Correlation.

This is the more complicated of the optimisation strategies. This process involves doing a course-to-fine multi-stage correlation. Consider the 2-pass cross correlation shown in Figure 6 as an example. In simple terms, the first pass is a broad guess at where the object has moved to and the second pass takes the information from this guess and provides finer detail.

For the first pass, the search region in the right image is centred at the same location as the template in the left image. This returns a pair (dpx, dpy) which is used as an estimate for the second pass.

In the second pass, the search region in the right image is centred at the location of the template plus (dpx, dpy).

Your new function should have the structure:

```
[dpx,dpy] = myfn(imagea, imageb, wsize, xgrid, ygrid, dpx_est, dpy_est)
```

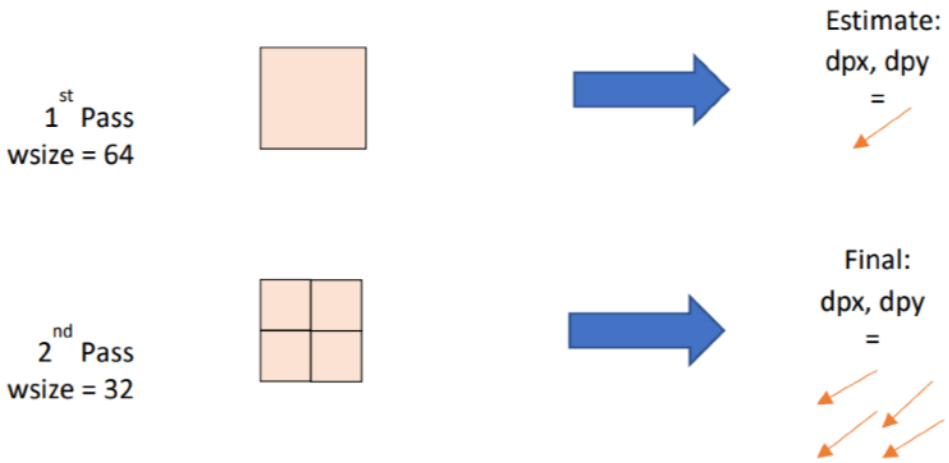

Figure 8 Multi-Pass Cross Correlation

Worksheet

In this part of the project, you can use any function except those which are from computer vision specific libraries. For example, you can use scipy.signal.correlate() with fft=True (faster) or your own spectral cross correlation function.

The spectral cross correlation function you have written needs to be submitted in your final report, so still need to get this working if you haven't already.

Task 1 - Setup

Write a function which imports all images in a folder.

Each folder will contain 2 images - the left and right camera images, which are marked with '_left.jpg' and '_right.jpg'. (most packages will read images regardless of filetype)

The images would then be converted to grayscale, and returned together

I.e

```
def load_images(filepath):

    # load image 1 (left image) as pattern

    # load image 2 (right image) as template

    # convert both to greyscale

    return pattern, template
```

Task 2 – Get windows

a) Write a function which accepts the left image (used as patterns) and splits it into a grid of non-overlapping window images.

The function should accept the left image, num y windows, num x windows.

The output should be a grid of images (numpy matrix its easiest), of shape (y, x).

You will need to think about the dimension of the windows (in terms of pixels) given the number of windows in x and y we want. If the image cannot be split evenly into that many windows in x and y because of the dimensions of the original image(pixels), you can either make the function exit with a printed error message, or you can be fancy and cut off the border of the image so the given num x, y windows works.

Think about how you can grab a subsection of a matrix with numpy!

b) Write a function which accepts the left image (used as patterns) and splits it into a grid of overlapping window images.

This will be the same as above, but a new argument needs to be given to the function – the overlap.

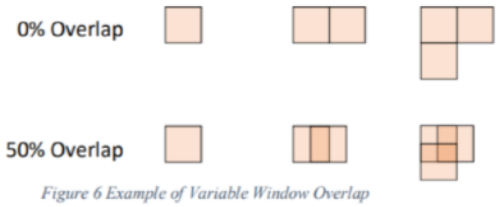

Figure 6 Example of Variable Window Overlap

This will allow us to increase the resolution of our depth map. We could just make our grid spaces smaller, but this method retains the pattern size. This provides more data (pattern is larger) when trying to locate it in the other image, and should provide more accurate cross correlation results.

Task 3 - Define search area

For each window in the left image, we will look for its location in the right image.

Write a function which accepts the right image, a location (center of search area), window dimensions, and number of windows in surrounding area to include in search are. It should return the search area in the right image.

There are 2 strategies to be explored:

- Square windows
- Horizontal windows

For each strategy, your function should accept the number of grid spaces in the right image (centered on the location in the left image) to search in.

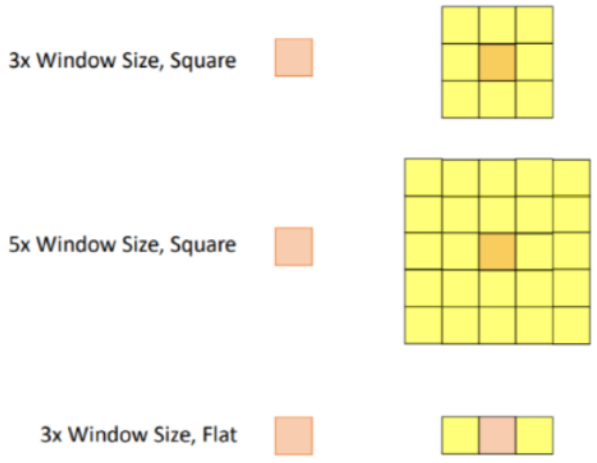

Figure 7 Example of Variable Search Region

For the images provided, you do not know if they lie on the same horizontal plane or not. Write both methods, then we will test them on the images.

We will also vary the window size to see which performs best on a given scene.

Task 4 – Multipass cross-correlation (optional)

This approach is to find a rough localisation for the pattern in the search area, then split the pattern into smaller windows and find the locations of each subwindow.

This approach attempts to improve resolution similar to the window overlap approach above.

Write a function which accepts the pattern (window of left image), the search area (from right image), and the best lag(x,y) for the pattern in the search area, and the smaller search area size in windows.

Your function should take that information, create a new, smaller search area centered on the best lag, split the pattern into 4 windows, then localise each window in the smaller search area.

Task 5 – offset for a window

Write a function which accepts a window (pattern) and a search area, and returns the best lag(x, y) for the window in the search area.

To do this, use a cross correlation function, then find the best lag as done in the wally puzzle. This is essentially just the wally puzzle code, but you can use libraries to perform the spectral cross correlation for you.

Task 6 – Putting it all together

The final function(s) should create a depth map from the left and right image.

After this stage, you are able to accept the left image, split it into windows, specify a search area for each window, and find the best lag for each window in the right image. Return a new matrix (of size x windows, y windows) where each index represents the offset for the corresponding window in the right image.

Remember that close objects have greater offset between the images, and distant objects have smaller offset. This will create the depth map. It can be visualised using matplotlib.

Links:

Improving speed through epi-polar geometry:

https://www.youtube.com/watch?v=O7B2vCsTpC0

Possible extension projects:

https://www.youtube.com/watch?v=bsA6RKUUA3M

https://au.mathworks.com/help/vision/ug/depth-estimation-from-stereo-video.html;jsessionid=ff7bc389b1c28cd2dff18a6ebb73