COMP90072 - Stereo Vision Part 2

Stereo Vision

Expected completion date: 30/4/18

Learning outcome goal for Part 2 is an understanding of:

- Code structure
- File handling
- Image Handling
- Pixel Manipulation
- Producing Figures
- Basic Modelling
- Optimisation

Required Mathematical Understanding

- Linear Algebra (assumed knowledge)
- Cross Correlation
- Gaussians
- Surface Fits

Unlike in Part 1, Part 2 does not ask for explicit inclusion of code in the report. As a general rule all code should be included in the appendix and some code or code snippets included in the body of the report.

For this Part of the assignment, you may use *any* functions in MATLAB other than those in the Computer Vision System Toolbox.

Introduction

Predators, with front facing eyes having been using the benefits of stereo vison for tens of millions of years to accurately determine the location of prey[1]. Since it's invention, photography has allowed us to capture flat, 2D representations of the real world with no explicit depth information[2].In this Part of the project, you're going to create a program which mimics the depth understanding of a human (see Figure 1) and can accurately analyse and represent depth information from stereo image pairs.

Building from your code in Part 1 (in particular the rocket man example), these are the next steps moving towards a 3d Stereo Vision system:

1) Create a program which can detect features on a calibration plate
2) Create a program which compares two same size images to each other using cross correlation
3) Create a program to batch process calibration data
4) Create a model which translates from *pixel space* to *real space*
5) Test system on provided computer generated images
6) Optimise System with Multi-Pass, variable overlap, variable window size changes
7) Demonstrate efficient computational usage and improved accuracy with your optimisation

To provide spatial information to your program, it is necessary to calibrate your system to the real world. This is done by using a calibration plate photographed at a multiple known distances and fitting a model. See Stereo_Camera_Calibration.pdf for more information.

The first step in calibrating a 3d vision system is detecting the markings on a calibration plate.

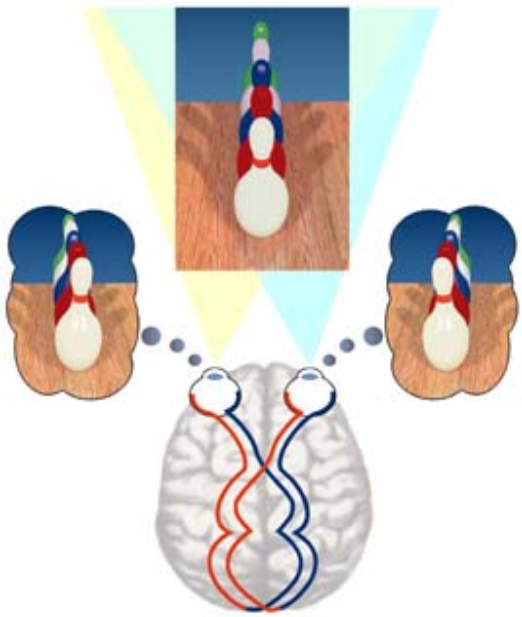

Figure 1 How your brain turns stereo vision in to a single image with depth information. Source: http://www.strabismus.org/all_about_strabismus.html

[1] https://www.newscientist.com/article/dn17453-timeline-the-evolution-of-life/

[2] (see https://www.google.com.au/search?q=forced+perspective&tbm=isch for some examples).

1 Dot Detection Algorithm

There are many ways to detect a dot in space, but for this project we're going to use a Gaussian peak detection method. This method uses a 2D Gaussian as a template and passes it over an image (search region) to find dots. This process is almost identical to searching for the Rocket Man in part 1.

Find and plot the locations of the dots from *cal_image_left_2000.tiff* on an equal axis.

To create the Gaussian template (Figure 2), the function *meshgrid* will be useful.

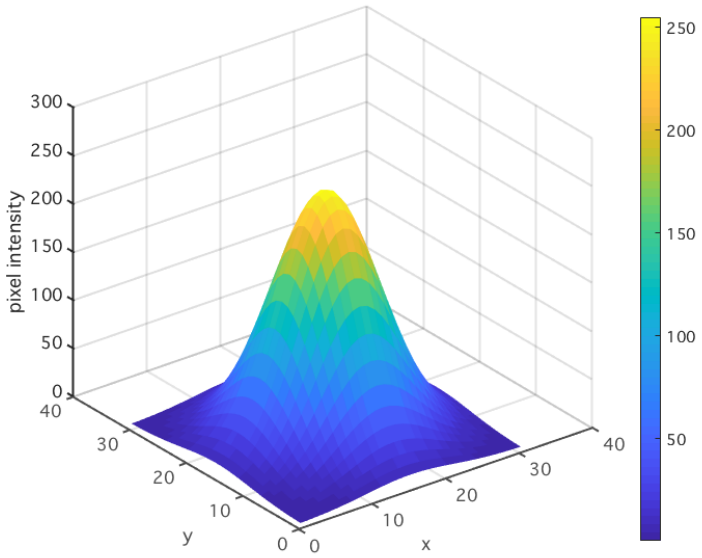

Figure 2 Gaussian

2 Create Calibration Model

Create a program which imports all the calibration images and records the pixel and real locations of the dots. Use a fitting tool to create a 4D surface fit which connects all pixel space to real space within your calibrated zone.

To achieve the fit, it may be useful to download functions from the MATLAB file exchange, as the inbuilt functions are limited. *polyfitn* is particularly useful for creating n-dimensional fits.
Some included functions which may satisfy your fitting needs are *nlinfit* and *griddatan*.

Your program should read in the multiple files automatically and should not require any manual entry after initialisation.

About Calibration Images:

There are a number of calibration images provided. These images are from a left and right camera, viewing the same calibration target at a stereo angle of approximately ±9◦. The calibration target has white dots spaced by 50 mm in the x (horizontal) and y (vertical) directions. The calibration target is shifted to various z locations, starting at a distance of 2000 mm from the camera, and shifted in 20 mm increments towards the camera (in the negative z direction). The calibration file names contain the name of the stereo camera (i.e left or right) and the z location of the calibration target (i.e 2000)

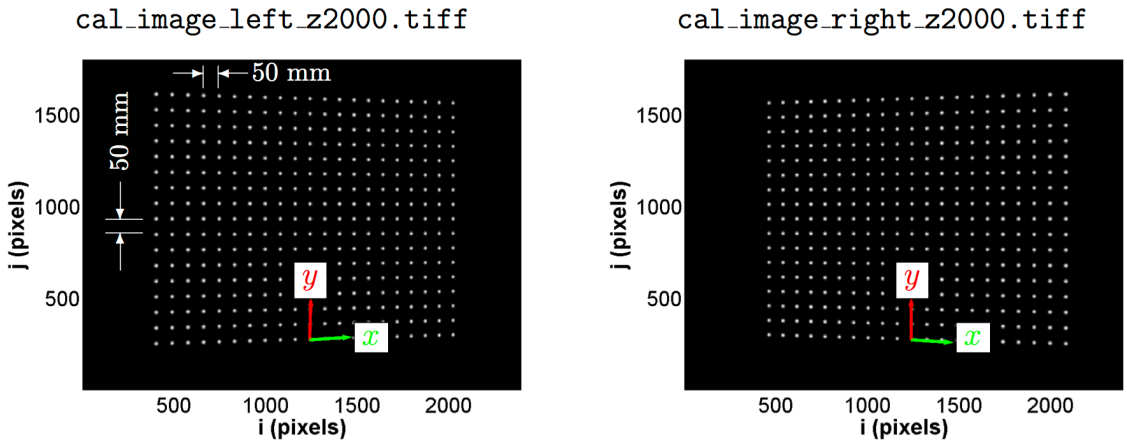

Figure 3 Calibration Images

Each of the identified dots has a known (x, y, z) in real space (in mm). The datum is as shown in Figure 3, with x=0mm , y=0mm located at the 11th dot from the left, in the lowest row of dots. The coordinates of each common dot from the left image (i_l,j_l) and the right image (i_r,j_r) are uniquely associated with a given (x,y,z) in real space. For example the coordinates of the lowest left-hand dot in the left and right images shown in Figure 3 correspond to the real space coordinates (-500, 0, 2000).

3 Image Comparison

Note: The term window refers to a small section of an image, like the orange box seen in Figure 4 & 5.
Create a program which compares two images using cross correlation.

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

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

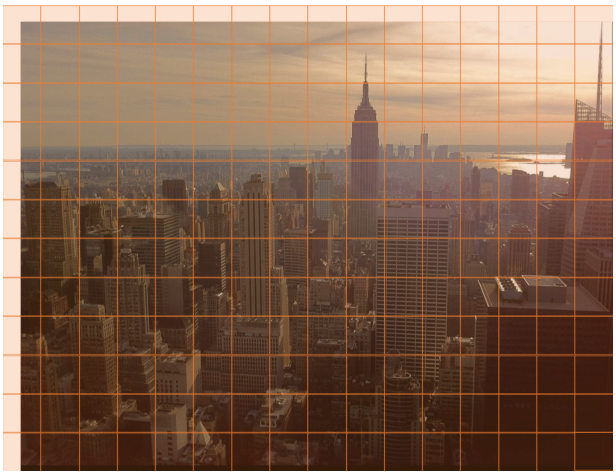

Figure 5 Left image broken up in to windows

4 Cross Correlation Optimisation

Implement three optimisation strategies:

a) Variable window overlap.

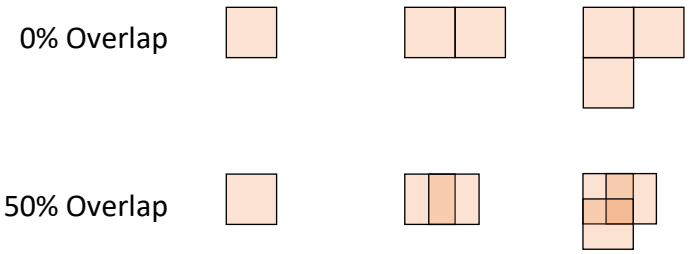

Figure 6 Example of Variable Window Overlap

b) Variable search region geometry.

Some examples of different search regions are shown below

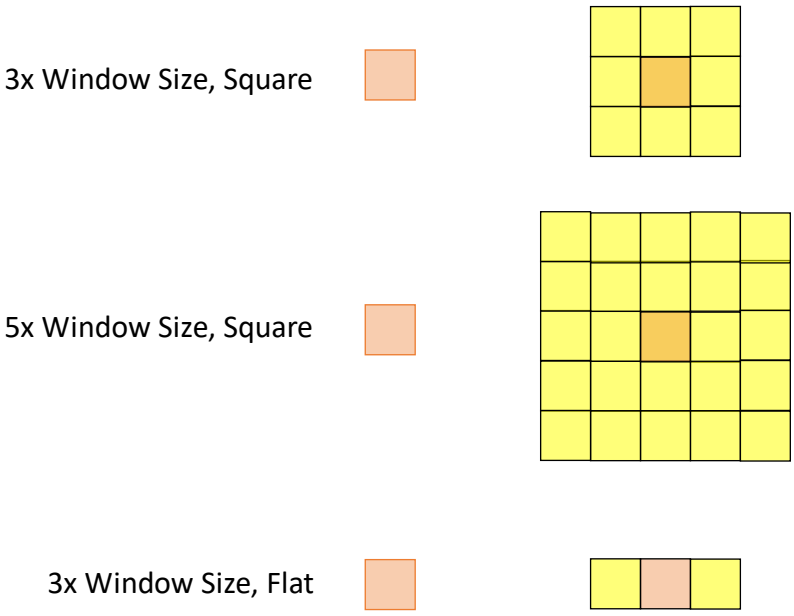

Figure 7 Example of Variable Search Region

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.
The first pass returns a dpx and dpy which are used as an estimate for the second pass.
For the first pass, the search region in the right image is centred at the same location as the template in the left image.
For the second pass, the search region in the right image is centred at the location of the template plus dpx and dpy.
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.

Your new function should have the structure:

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

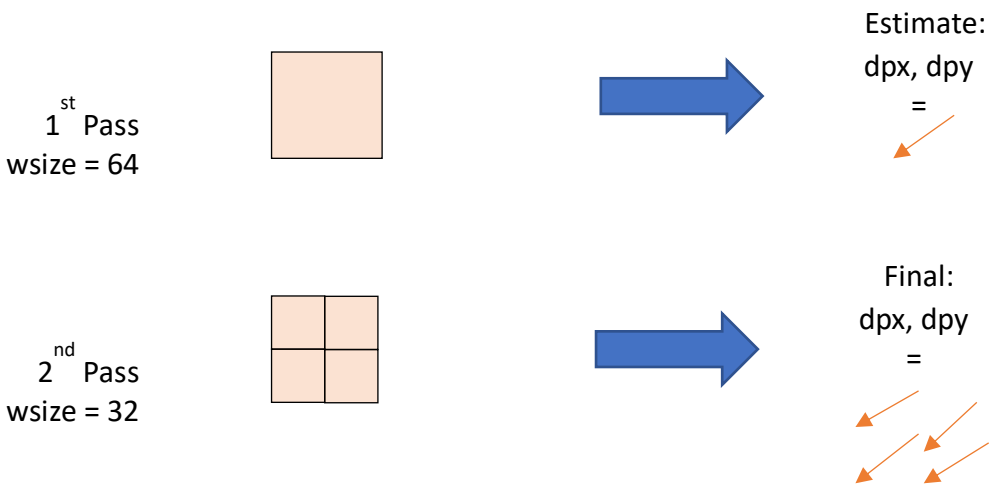

Figure 8 Multi-Pass Cross Correlation

For splitting your estimate up and assigning the value to a new window, a Kronecker product might be useful (MATLAB function *kron*)

5 Test Scan on Computer Generated Calibrated Images

Create 3D reconstructions of the 3 test image pairs provided and display your results in an intuitive and clear way.
The result for test image pair 1 is shown below in Figure
When testing image pairs 2 and 3, it may be necessary to remove any spurious vectors. A spurious vector is a dpx or dpy result which is artificially high due to errors in the cross correlation.
Discuss the effects that different overlap, search region and passes have on the result.
Discuss the limitations of your program.

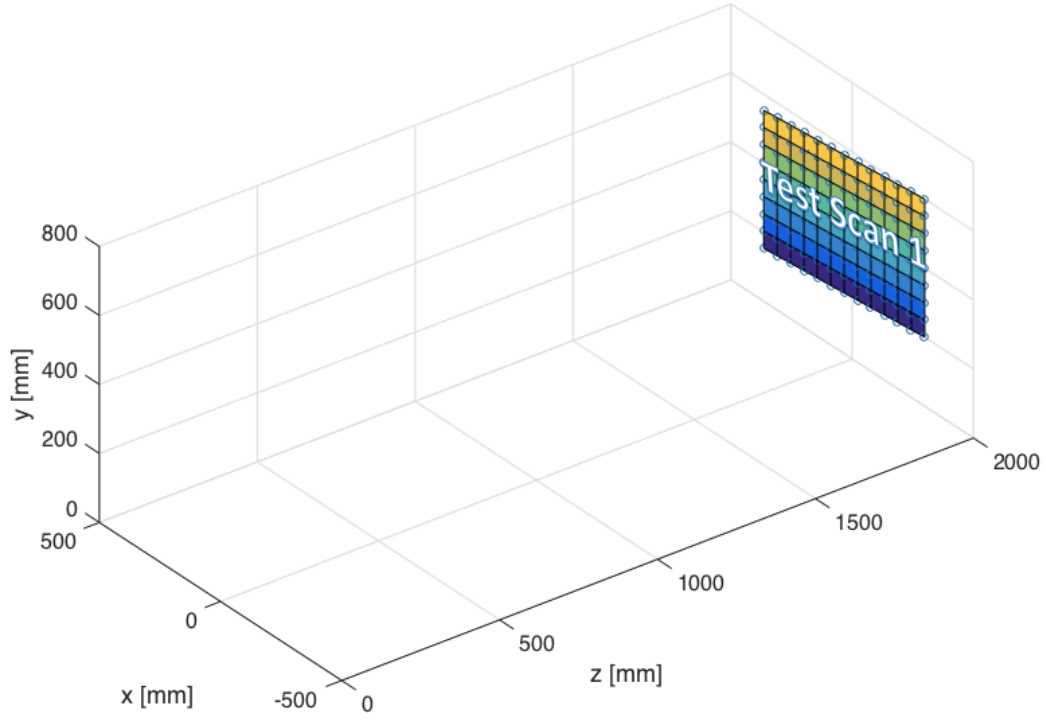

Figure 9 Test Scan 1 Result

6 BONUS: Optimised Test Scan

Demonstrate how your code achieves a more accurate and faster result.
Update your calibration model to detect the centre of a dot at a sub-pixel level. Update your general cross correlation to work on a sub-pixel level.
Which parameters achieve the fastest acceptable result?
Which parameters achieve the most accurate result?