Assignment Brief	
Title:	Computer Vision Assignment 2
Submission Deadline:	21st March 2024, 16:00
Submission:	Online (DLE)
Contribution to Module Grade:	60%
Individual/Group Assignment:	Individual
Module:	ROCO321
Module Leader:	Dr Dena Bazazian
Lab Technician:	James Rogers

Overview

These labs involve using the owl robot that can be booked out from stores. Please **work individually** with the robot and **use the same owl each week**. **Submitted reports are individual pieces of work.**

Marking

Assignment 2 is worth 60 marks (60% of the module).

Tasks 3 and 4 are worth 25 marks each and are broken down as such:

- **Solution (7 marks):**
  - **(0-2)**: Code barely fulfils the task specification.
  - **(3-5)**: Code fulfils the task specification.
  - **(6-7)**: Code fulfils task specification and extra features have been added that improves the program.

- **Method (7 marks):**
  - **(0-2)**: Little to no effort explaining the code or background on theory.
  - **(3-5)**: Some theory, and an explanation of the code.
  - **(6-7)**: A comprehensive background in relevant theory, and a detailed explanation of the code with appropriate justifications.

- **Evaluation (7 marks):**
  - **(0-2)**: Little to no effort assessing the performance of your solution.
  - **(3-5)**: A measure of performance is identified (such as: error rate, false positives / negatives, etc.) and data is collected to assess your solution.
  - **(6-7)**: A measure of performance is well explained and justified, and a significant amount of data is collected to assess your solution. This may involve making your own data sets to evaluate your solution in a broader scope.

- **Conclusion (4 marks):** Based on your evaluation (with data and numbers to back it up), what conclusions can you make from your solution, and what would you improve on?

An **extra 10** marks are awarded for including the following in the report:

- **Videos (2 marks):** Must contain links to YouTube videos of your code working. The videos don't need to be complicated, just a short clip demonstrating your code functioning as the task describes.
- **Flow Diagrams and Code Snippets (3 marks):** Flow diagrams and snippets will be needed to explain your code, marked on how clear and informative they are.
- **References (3 marks):** All quotes, information, and diagrams used in the report should be clearly cited.
- **Acceptable English and Grammar (2 marks):** Not marked strictly but do proofread before submitting.

Task 3: Cross Correlation Tracker (25 marks)

Create a tracking program like task 2, only this time use cross correlation to locate your target. Expand the tracking code to move both eyes to converge on the target and estimate the distance using the eye angles and trigonometry.

Tasks:

- Open the Task3.pro file in the ROCO321Part2 folder for the task template.
- Use cross correlation on the left and right eye to locate the target in both camera frames.
- Draw some kind of marker on the detected target.
- Move both the left and right eyes to try and centre the target in each frame.
- Using the servo angles, estimate the distance to the target.

Notes

The code in the template will first give you an interface to select the target pattern you want to track. **Align your target within the box on the screen and press SPACE to select it.** After this, it will enter a loop where the camera frames are displayed to you, along with a cropped image of your target. This loop is where you will write your tracking code.

The target you pick to track will drastically affect the quality of the tracking. For example, a flat colour with no markings will not be suitable as many parts of the image (such as the wall) will match this pattern. **Something like a cross or shape drawn on white with a black marker would work better,** as the unique pattern will be unlikely to appear elsewhere in the image. It might take some experimenting to find a pattern that works for you.

Cross correlation in OpenCV is done with the matchTemplate() function, which is used like so:

```
matchTemplate(videoFrame, targetImage, matchOutput, match_method);
```

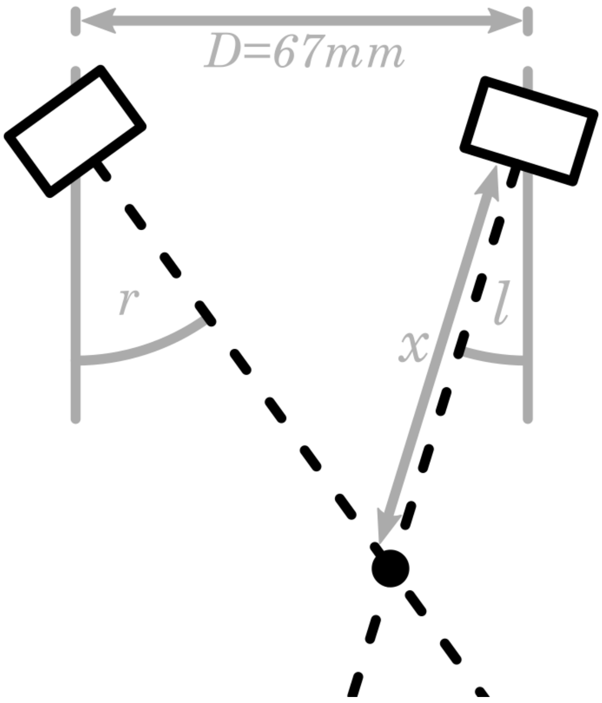
In this case use "TM_SQDIFF_NORMED" as the match method. The image this match method produces shows the error of the match over the whole image. In the example below, **white areas show where the target didn't match the image at all, and darker areas show better matches.** As you can see there is a very dark spot, indicating a point of very low error. This is the best match for the target over the whole image, and what you want to find.

To find this minimum error point, use "minMaxLoc". Note you must define the min/max value variables (double) and the min/max location variables (Point) before using this function as it outputs via reference:

```
minMaxLoc(Input_Image, &Min_Val, &Max_Val, &Min_Loc, &Max_Loc);
```

Min_Loc is the location of the top left corner of your target (or at least the closest match to it) and Min_Val stores the error value of the match. You can use this error value against a threshold to check if the target is visible at all.

Do this for both the left and right camera frames, and use these positions to move the eyes so that they both track the target. This will cause the eyes to converge onto the object like a human would and from this you can estimate depth. Note, you may need to follow the owl servo calibration guide to make sure the angles are accurate.



Use the getServoAngles() owl member function to get values in radians for r and l as show above. Calculate the value of x (distance to the left eye) and display this on the console or camera frame. How accurate is this method of depth perception?

Task 4: Disparity Mapping (25 marks)

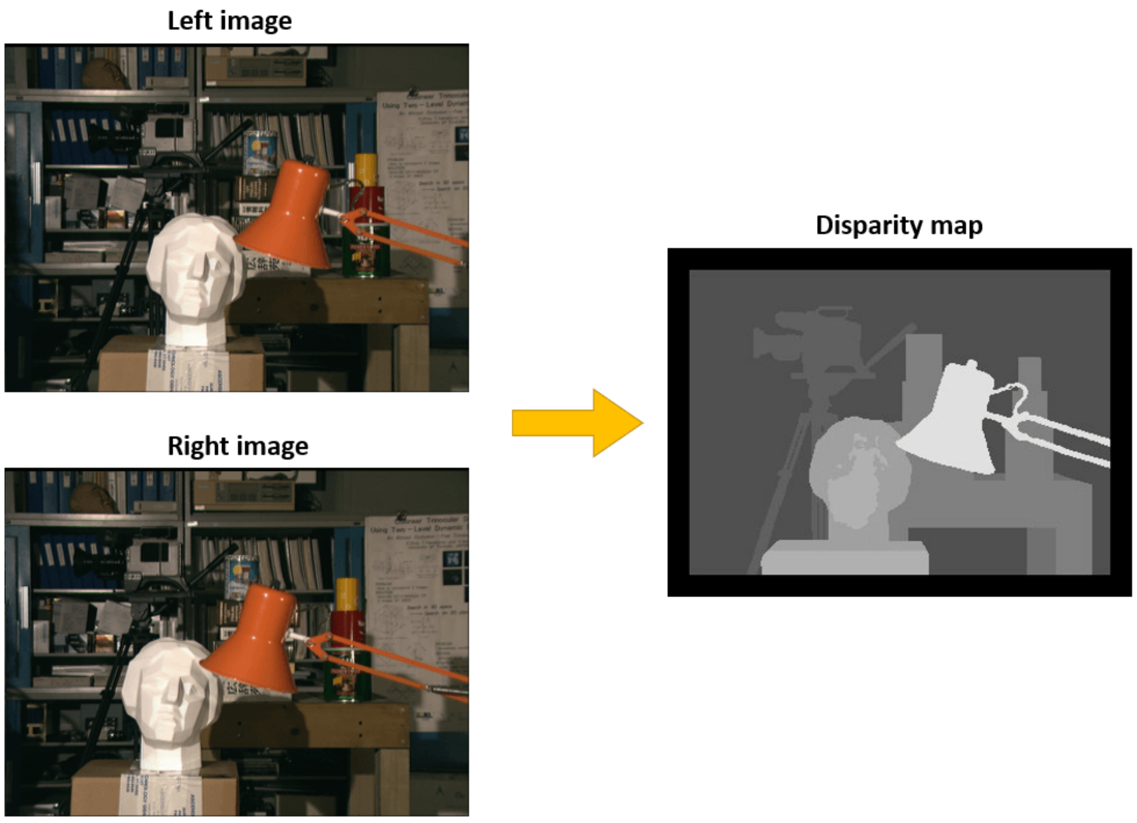
Disparity mapping takes cross correlation to the next level by matching patches of pixels in the left and right views to find the perspective shift over the whole image. This is extremely sensitive to camera misalignments and lens distortions however, so **there will be a calibration process to get it working properly.**

Tasks:

- Calibrate the servos of the owl bot to be parallel.
- Calibrate the cameras and create the extrinsics and intrinsics files.
- Produce a disparity map from the left and right images.
- Calibrate distance measurements to known targets.
- Compute a distance map from the disparity map.
- Display the distance in cm of the centre of the screen using opencv drawing functions.

Notes

As mentioned above, disparity mapping uses cross correlation to match thousands of targets between the left and right images. The resulting disparity map describes how much objects shift position from one eye to the other. In the example below, the lamp has the biggest difference between the left and right images, hence it appears brighter in the disparity image.

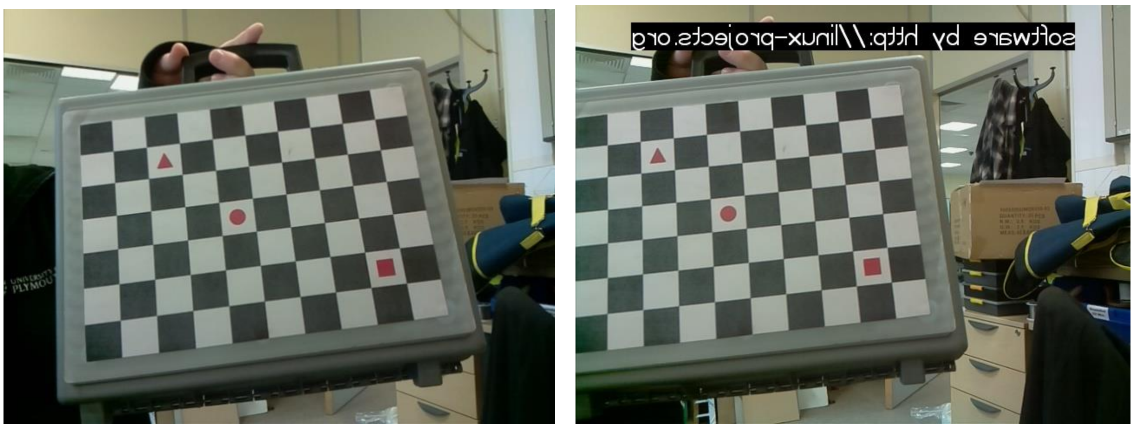


Performing thousands of cross correlation operations would normally be absurdly computationally expensive. However, several optimisations can make it run in real time. One of which, is **assuming that there is no lens distortion (straight objects appear straight in the frame), and that the cameras are perfectly horizontally aligned.**

The first part of this task is to correct for these distortions. Step one is to calibrate the servos so that the cameras are parallel. **This process is explained in the "Owl Servo Calibration Guide" PDF.**

After doing this you'll need to calibrate the cameras. This is done by taking images of the calibration target on the owl robot box. **In total you'll need about 30 pairs of images** with the following requirements:

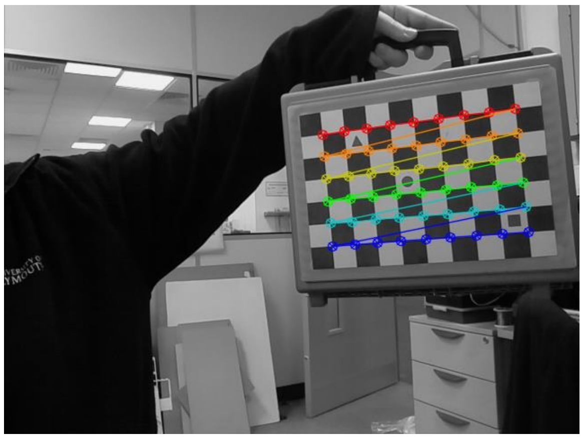
- Images are in focus with no motion blur.
- The calibration target is completely in shot in both camera views.
- Have the box at different distances and angles in each image pair.
- Make sure to move the target in different areas of the screen, around each corner, around the centre, etc.



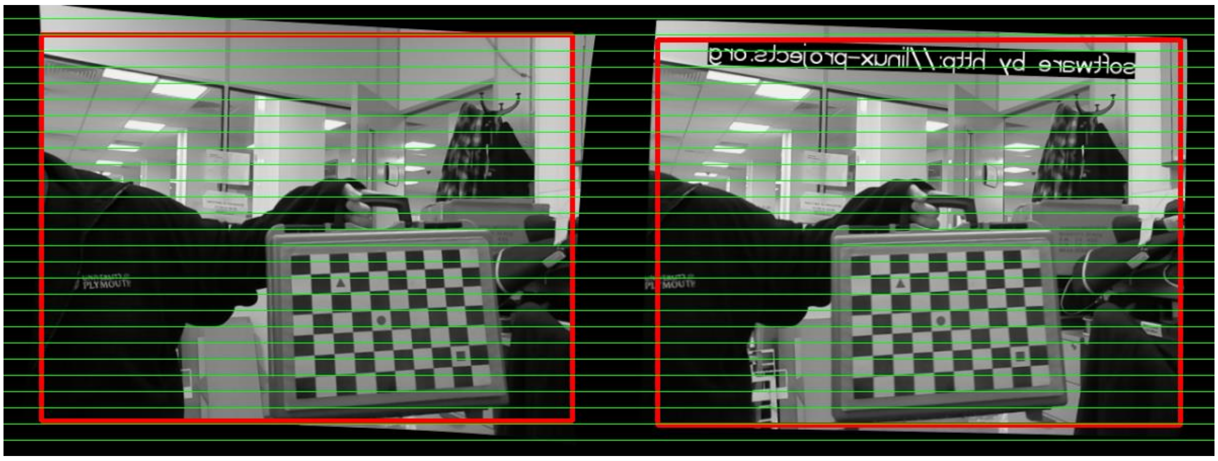
To help you take these images, a project called "Stereo Image Capture" has been included in the ROCO321Part2 folder. An example of a good calibration set has also been included for reference, though **these images will not work for your owl,** you can still use them to test the calibration software.

Once you have your calibration image set, **open the "Stereo Calibration" project**. This program will detect the checkerboard in each image pair and calculates the physical misalignments (extrinsic error) and lens distortion (intrinsic error).

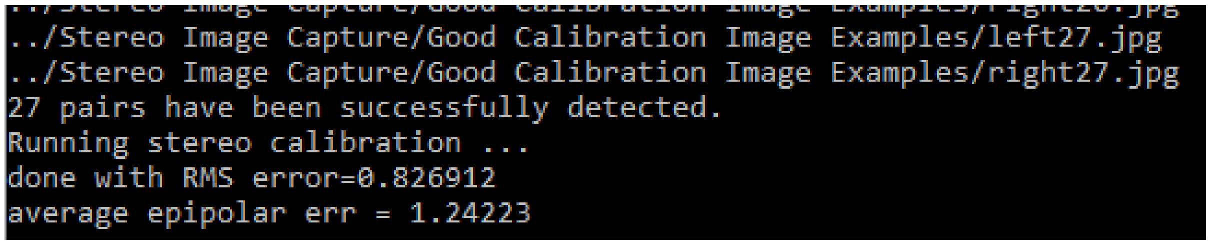
In Qt, under "Other files", you'll find a file called **"image_list.xml".** This lists the paths to all of the images in your calibration set. By default, the list links to the example image list so you can test the software is working, but **you'll need to change these paths to link to your own images.**



When you run the program, you'll see it first detect the checkerboard in each image (as shown above). **Watch as it does this to make sure it's successfully detected each time.** After this, it will compute the distortion needed to correct for the error. You should end up with a display as seen below. Epipolar lines shown in green demonstrates what the calibration does, **objects that appear in the left image should appear on the exact some y coordinate as the right image.** This leaves only x coordinate differences behind which correspond to the distance of the target.



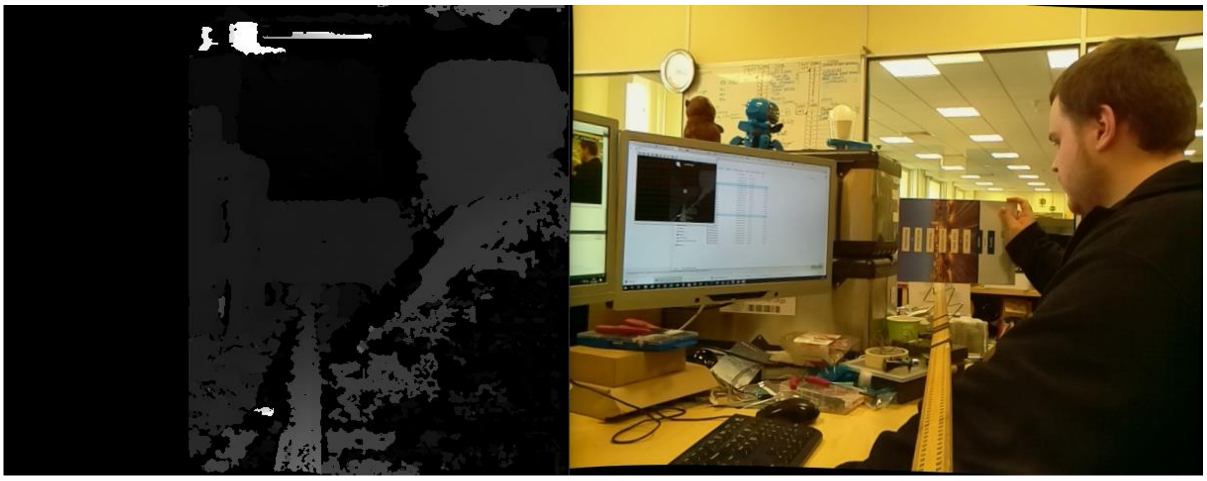
After calibration two values are returned to measure how good the alignment is. **Aim to get similar or better values that the ones below.** It can be tricky to get it accurate and often the only solution is to try a different set of calibration images.



```
../Stereo Image Capture/Good Calibration Image Examples/left27.jpg
../Stereo Image Capture/Good Calibration Image Examples/right27.jpg
27 pairs have been successfully detected.
Running stereo calibration ...
done with RMS error=0.826912
average epipolar err = 1.24223
```

After this process, you should find two files in your ROCO321Part2 folder, **extrinsics.xml and intrinsics.xml.** Keep these files safe as they store the calibration data from this step.

Finally, you can now start making disparity maps! **Open the Task 4 project and run the program,** it should find your calibration files and start computing the disparity map from the left and right views.



You should have similar or better quality than the view above. Black areas are patches of pixels that couldn't be matched in the left and right frames, and some areas might have spurious values due to false positive matches. So try to find settings and matching methods that work best for you. Under the "create block matcher" section of the program, there are a number of values you can adjust to change the quality of the image so **some experimentation may be needed.**

The disparity image you are left with is a 16-bit single channel image called "disp". There is also an 8-bit version called "disp8" but this is just for visualisation reasons, don't use it for measurements as it has reduced resolution. Information on how to read values from a 16-bit Mat is outlined in the opencv basics guide. **Distance and Disparity are inversely proportional,** as described in the following equation:

$$Disparity = \frac{Baseline \cdot focal\ length}{Distance}$$

Baseline represents the distance between the two cameras, and focal length relates to the lens and sensor of the camera. However, all you need to know is that **these values are constants.** Therefore, the problem becomes:

$$Disparity = \frac{C}{Distance}$$

$$\therefore \boldsymbol{Disparity \cdot Distance = C}$$

This means that if you find the disparity of an object, and measure its distance from the cameras, you can calculate this constant. This is quite sensitive to noise so **do this for a number of different distances to achieve an average**, but they should all be similar values.

With this constant term, you can convert the disparity values from the disparity map, into distance values on a **distance map.** Using values on this distance map, display the distance to objects in the centre of the screen using opencv drawing functions. **How accurate are your measurements?**

Deadline and Submission

Deadline on **21st March 2024, 16:00.** The deadline information for submission is on the ROCO321 DLE page via the submission link.

Plagiarism

This is an individual assignment and must reflect the work of that individual. Thus, while you may discuss this assignment in general with your colleagues and give each other technical help, your scientific literature review must be entirely your own work.

The University treats plagiarism very seriously. If you cannot satisfy me that your work is your own, formal plagiarism procedures will be started.

The penalty for submitting work which is wholly or partially the work of someone else is usually, at least, a mark of zero for the assignment. Do not be tempted to help a colleague by giving them your work, as both parties will be guilty of an assessment offense, and both face the risk of a zero mark. Please refer to your student handbook for guidance as to what constitutes original / individual work.

Module Learning Outcomes Assessed

- ALO-1: Demonstrate a systematic understanding of the underpinning vision theories and techniques applicable in object recognition.
- ALO-2: Critically evaluate current research and methods in artificial vision.
- ALO-3: Design and implement an original practical vision system.