**CMPT 412**

**Computational Vision**

**Report- Project5**

***3D Reconstruction***

**Task 3.1: Sparse reconstruction**

In this task, it is asked to first estimate the Fundamental matrix, compute point correspondence, then plot the results in 3D.

Below are the asked write-ups for **Sparse reconstruction**:

**Task 3.1.1: Implement eight point algorithm**

In this task of sparse reconstruction, it is asked to use the eight point algorithm to estimate the **Fundamental matrix** by using the point correspondences provided in “**someCorresp.mat”**. The code for the same is done in “**eightpoint.m**” script provided in the MATLAB folder. The fundamental matrix is estimated, and the visualizations are generated using “**displayEpipolarF**” function. The code for the same is done in the “**test.m**” script submitted in the MATLAB folder.

Below are the results of the **recovered F** and the **visualization of some epipolar lines:**

**Recovered F**

Graphical user interface, application, Word

Description automatically generated

**Visualization of some epipolar lines**

A picture containing engineering drawing

Description automatically generated

**Task 3.1.2: Find epipolar correspondences**

In this task, it is asked to implement a function "**epipolarCorrespondence**" that takes in two stereo images, a fundamental matrix, and a set of points in the first image, and returns the corresponding points in the second image. The function uses the epipolar geometry to reduce the search space for matching points in the second image. The code for the same is done in "**epipolarCorrespondence.m**" script.

The similarity metric used in the code is the Euclidean distance between the target window around the point in the first image and the candidate window around the point in the second image. In the code resulting distance metric represents the difference between the pixel values of the two windows being compared, with smaller values indicating higher similarity between the windows

However, there may be cases where the matching algorithm consistently fails.

* One such case is when there are occlusions in the scene, and a point in one image is not visible in the other image due to some object blocking its view. In such cases, the epipolar line assumption may not hold, and the algorithm may fail to find a corresponding point in the second image.
* Another case is when the images have low texture or similar patterns, making it difficult to distinguish between different points. In such cases, the similarity metric used may not be robust enough to differentiate between different points, resulting in wrong matches.

To visually test the function, “**epipolarMatchGui**” method is used which is implemented in “**epipolarMatchGui.m”** script. The visuals are generated, and the above functions are called in “**test.m**” script submitted in the MATLAB folder.

Below is the visual of the **epipolarMatchGui** running with the imp0lementation of **epipolarCorrespondence:**

A picture containing diagram

Description automatically generated

Visually it looks good. Below are the matched pixel coordinates in the first and second images respectively:

Table

Description automatically generated

Each row of “**coordsIM1**” corresponds to the pixel coordinates in the first image that match with the corresponding row of “**coordsIM2**” in the second image.

**Task 3.1.3: Write a function to compute the essential matrix**

In this task, it is asked to compute the **Essential Matrix**  to get the full camera projection matrices. The code for the same is done in “**essentialMatrix.m**” script. The method is called, and the essential matrix is generated. The code for the same is done in “**test.m**” script provided in the MATLAB folder.

Rectangle

Description automatically generated with low confidence

**Task 3.1.4: Implement triangulation**

In this task, it is asked to implement a function “**triangulate**” that takes in two sets of 2D image points and their corresponding camera projection matrices and returns a set of 3D points triangulated from those 2D points. The triangulation process involves finding the intersection of the two lines formed by projecting the 2D points onto their respective camera projection matrices. The code for the same is done in “**triangulate.m”** script provided in the MATLAB folder.

After calling the “**camera2(E)”** function, four extrinsic matrices were obtained. The chosen extrinsic matrix was based on the number of 3D points visible to both cameras, and the correct one was matrix number 4 because it was having the greatest number of points visible in front of both cameras, which was a total of 288 point correspondences.

The **reprojection error** was calculated for both sets of 2D points. The reprojection error using **pts1** was **0.5622**, and the reprojection error using **pts2** was **0.5667**.

**Task 3.1.5: Write a test script that uses templeCoords**

In this task, it is asked to write a test script called "**testTempleCoords.m**" that uses the functions implemented in the previous tasks to generate a full 3D reconstruction of a scene from two images and their corresponding point correspondences. The script is provided in the MATLAB folder.

Below are the 3 images of the final reconstruction of the **templeCoords** points from different angles:

**Image1**

Chart, radar chart

Description automatically generated

**Image2**

Chart, radar chart

Description automatically generated

**Image3**

**Chart, radar chart, scatter chart

Description automatically generated**

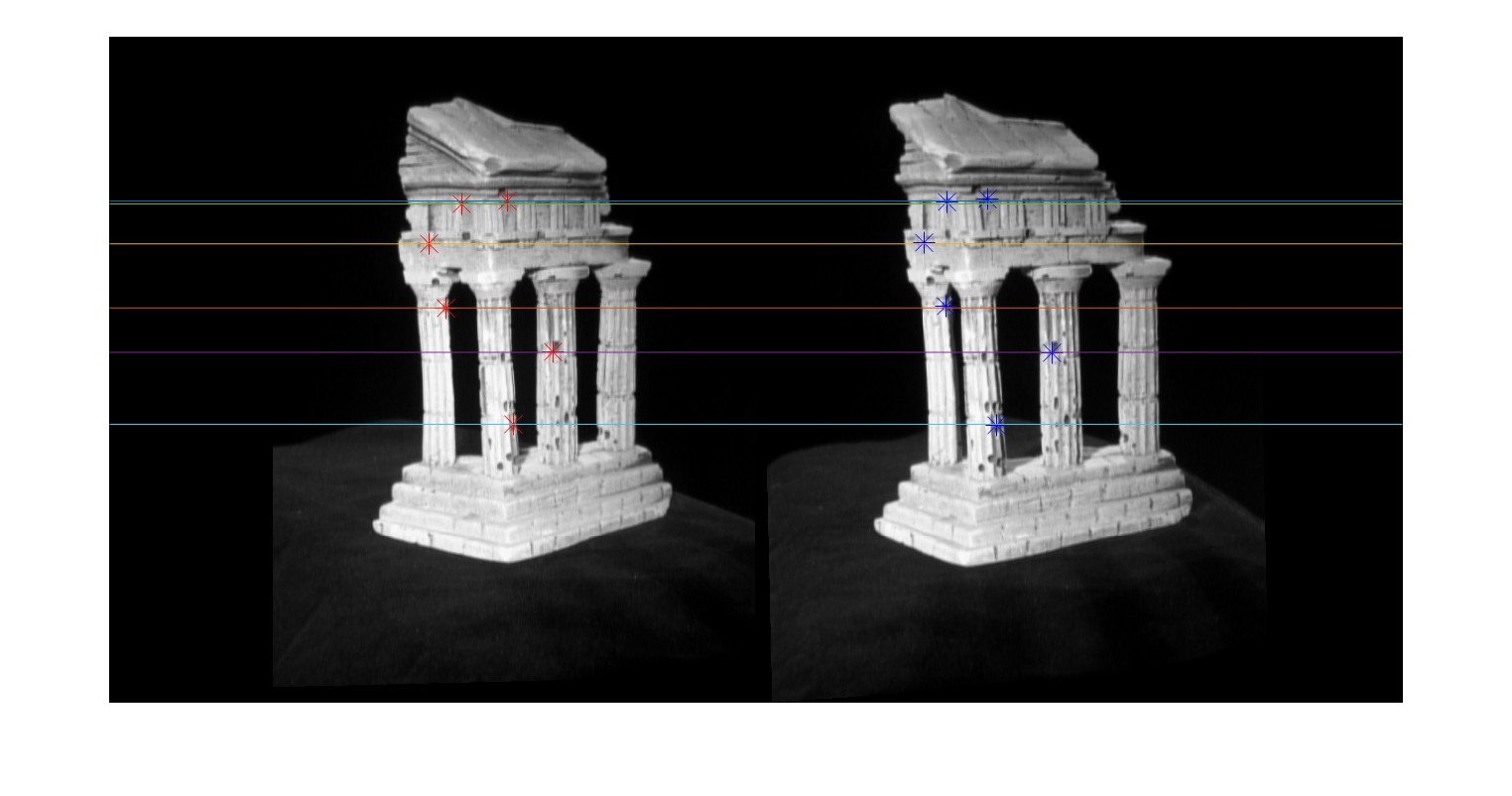
**Task 3.2: Dense Reconstruction**

In this task, it is asked to write a set of functions to perform a dense reconstruction on the temple images using rectification. The function will take the intrinsic and extrinsic parameters of the two cameras and computes the rectification parameters of the two images. The rectification should make the epipolar lines horizontal, making it easier to search for correspondences. After that rectify the images will be done.

**Task 3.2.1: Image Rectification**

In this task program of computing rectification matrix is written. The code is done in “**rectify\_pair.m**” script provided in the MATLAB folder.

Below are the results of the image rectification generated after running “**testRectify.m**” script:



**Task 3.2.2: Dense window matching to find per pixel density**

In this task, it is asked to write a program to create a disparity map from a pair of rectified images. The code for the same is done in “**get\_disparity.m**” script provided in the MATLAB folder. The disparity map image is generated after running“**testDepth.m**” script. The visual for the same is reported in the **Task 3.2.3** along with the visual of depth map.

**Task 3.2.3: Depth map**

In this task, it is asked to implement a function that creates a depth map from the disparity map. The “**get\_depth**” function is implemented in the “**get\_depth.m**” script and the image of depth map is generated after running “**testDepth.m**” script provided in the MATLAB folder.

Below are the images of both disparity and depth map:

**Disparity Map**

A picture containing diagram

Description automatically generated

**Depth Map**

**A picture containing graphical user interface

Description automatically generated**

**Task 3.3: Pose estimation**

In this task, it is asked to estimate both intrinsic and extrinsic parameters of camera given 2D point x on image and their corresponding 3D points X.

**Task 3.3.1: Estimate camera matrix**

In this task, it is asked to implement a function that estimates the camera matrix , given 2D and 3D points. The code for the same is done in “**estimate\_pose.m**” script. Below is the output obtained after running “**testPose.m**” script:

Text

Description automatically generated

**Task 3.3.2: Estimate intrinsic/extrinsic parameters**

In this task, it is asked to implement a function that estimates both intrinsic and extrinsic parameters from camera matrix. The code for the same is done in “**estimate\_params.m**” script. Below is the output obtained after running “**testKRt.m**” script:

Text

Description automatically generated

**Note:** Task 3.4 “Multi-view stereo” is a bonus task and is not attempted by myselm. The scripts for the implemented functions are saved in “matlab” folder submitted under submission.