Student Name: *Furqan Ahmed* Student ID#: 1000927009.

MET 3D Point Cloud Construction Using Bundle Adjustment

1 Project Problem Definition

One of the main problems in robotic navigation involves generating a map of an unknown environment and locating the position of the robot within this environment. This computational process is known as Simultaneous Localization And Mapping (SLAM) and using a vision based approach to solve this problem is known as Visual SLAM.

For this project we will use bundle adjustment techniques to solve the following problem:

"How to find the orientation of a robot in a given frame and characterise the environment it is in?"

Bundle adjustment solves this problem by determining 3D positions of landmark points and camera poses from a set of image feature correspondences across multiple images. Starting from an initial guess for camera pose and landmarks (3D points in the world), non linear least square optimization can be used to minimize the reprojection error of landmark points by adjusting the initial pose and landmark estimates.

2 Project Methodology

For this project we used the monocular image data from run 4 of the CPET dataset. We specifically used images from frames 50 to 110. We choose to use the monocular image data, as opposed to the stereo setup as this provided a more generalised problem statement. A well performing implementation for a monocular case can be readily applied to extract scene information from any set of images of a given scene. Although this produces additional challenges of obtaining ground control points and retrieving scale, the more general nature of this approach seemed more appealing than the stereo bundle adjustment problem.

The main steps used for solving the bundle adjustment problem can be summarised as follows:

1. Identify point correspondences between camera frames using a SIFT descriptor.
2. Use SIFT point correspondences to determine the essential matrix between the two camera frames. 5-point algorithm can be used for this.
3. Retrieve the rotation matrix and unit translation vector from the essential matrix.
4. Determine distance travelled between frames by using the rover velocity estimate from wheel encoders for the time window of the selected frames. This distance is then multiplied with the unit translation vector to obtain the actual translation between frames.
5. Estimate 3D location of landmark points using triangulation
6. Repeat the above steps for all combinations of frames being considered.
7. Convert 3D points and frame transformation matrices with respect to the first frame and use these results as initial guess for non linear optimization.
8. Compute the sparsity structure of the jacobian matrix
9. Solve the system and retrieve final camera poses and 3D points

The code implementation of the bundle adjustment problem is described in more detail next.

2.1 Frames selection

The monocular image set contain images captured at every 0.2 second intervals and given the rover had an average speed of 0.5 m/s for run 4, this translates to a new frame captured at every 0.1 meters. For our problem, we decided to use images from every 5^{th} frame or after about 0.5 meters. This step size was picked as it gave a reasonable trade off between number of points analysed and consistency of

landmarks between frames. Also each frame was paired with 5 consecutive frames for identifying point correspondences.

Given that this was a downward facing camera, only the terrain is exposed in about half the picture. For this reason, we decided to ignore pixels a y coordinate of 500. This allowed us to speed up computation without losing too many important features.

2.2 Point correspondences between frames

Point correspondences between image pairs was determined by comparing the SIFT feature descriptors between the two images. SIFT descriptors were generated using the *SIFT_create()* function from the OpenCV library and the descriptors were compared using a kNN search with k=2 (2 nearest feature descriptors in image 2 are selected given a descriptor in image 1).

Bad matches are filtered using the Lowe's ratio test – for a given descriptor in image 1, if the ratio of norms of the two descriptors in image 2 is greater than 0.7, then this match is rejected. The probability of making a wrong feature association is higher if two descriptors are very similar to each other.

The code implementation of the point correspondence can be found in *Feature_Matching.py*.

2.3 Finding the Essential Matrix

Given point correspondences between two images and the camera calibration matrix, the essential matrix for the transformation between the two cameras can be determined using the well known 5-pt algorithm. The OpenCV function *findEssentialMatrix* accomplishes this and was used in the project to retrieve the essential matrix. It also runs RANSAC to filter outlier points that resulted in inconsistent matrix results.

2.4 Retrieving pose from essential matrix

The translation between the two camera frames can be recovered from the essential matrix via SVD. This results in 4 possible solutions but 3 of these solutions result in triangulation happening behind the camera (negative depth value). Hence the chirality of the 4 solutions needs to be checked and the correct result selected. We used the *recoverPose* function from OpenCV to accomplish this. The resultant translation vector is however only defined up to a scale.

The magnitude of the translation vector was determined by using the velocity estimations from the rover. Knowing the time stamps of the frames being considered, the average velocity in this interval can be retrieved from *velocity_estimate.txt* and the total distance covered can be calculated. This is accomplished by the *distance* function found in *observations.py*.

2.5 Point Triangulation

Having determined the camera transformation between and feature correspondences, we can now triangulate the 3D position of a given feature correspondence. The theory discussed in reference [1] was coded in the *triangulate* function to accomplish this. The function accepts the feature correspondences, the rotation matrix and translation vector and solves the over constrained triangulation problem (3 unknowns in 4 equations) in a least squares sense.

2.6 Set up for bundle adjustment

The steps discussed in sections 2.2 to 2.5 are performed for each image pair determined in section 2.1. The functions coded in *Feature_Matching.py* accomplish this task for a given image pair.

The camera transforms, 3D points and 2D correspondences obtained from the above process are sorted by the code in *observations.py.* Firstly, all the camera transforms and 3D points are transformed with

respect to the initial frame by appropriately chaining the transforms between frames. Finally the data is sorted in the following format:

Table 1: 2D Point Correspondence List

Camera Index	Point Index	2D Image Coordinates
0	0	(x_{00},y_{00})
1	0	(x_{10},y_{10})
⋮	⋮	⋮
N	M	(x_{NM},y_{NM})

Table 2: Camera Parameters

Camera Index	Camera Transforms
0	1x6 parameter vector
⋮	⋮
N	1x6 parameter vector

Table 3: 3D Point Coordinates

Point Index	3D Point Coordinates
0	X_1, Y_1, Z_1
⋮	⋮
M	X_M, Y_M, Z_M

The camera parameters and point coordinates are the initial guesses for Bundle Adjustment optimization. The 2D image coordinates will be used to calculate the reprojection error for the coordinates estimated from the camera model using the camera parameters and 3D points.

2.7 Jacobian Sparsity Structure

The jacobian sparsity structure is computed using the camera and point index information stored in *2D point correspondences*. The sparsity structure of the jacobian is well known and will not be explained in detail here. The code implementation for this can be found in the *bundle_adjustment_sparsity* function.

One thing to note here is that the entries corresponding to the first camera's parameters were set to 0. This is because our system is defined with respect to the first camera and hence it's transformation matrix has to be an identity matrix and can't be modified during the optimization process.

2.8 Non Linear Optimization

Having collected the initial guesses, structured our data and defined the Jacobian structure, we can now initialize the non linear least squares optimization. The *least_squares* function in *scipy.optimize* is used to perform the optimization. The function calculates the Jacobians numerically at every step. The sparsity of the Jacobian can be passed to the function as a sparse matrix with appropriately filled 0s and 1s.

The optimization routine and subsequent processing of results can be found in *main_script.py*. The optimization procedure coded here was built by referencing an example problem shown in reference [2].

3 Project Evaluation and Results

3.1 Project evaluation metrics

The model performance was evaluated using the following metrics:

1. **Reprojection error** – this criterion evaluates how well the projected points from the model compare with the measured 2D points. The quality of the initial guess and the performance of optimization routine can be analysed by evaluating the reprojection error before and after the optimization.
2. **Camera position error** – this criterion evaluates how well the predicted camera frame positions compare with the ground truth data. The ground truth data for this comparison was obtained from the *global_pose_utm.txt* file. The pose here is given with respect to the rover frame. This information was transformed to the monocular camera frame by using the rover to monocular transform. The camera positions retrieved at the required time step were normalised against the initial frame which serves as the origin in our model. The code for obtaining the ground truth can be found in the *get_ground_truth* function.

Comparing the point cloud results with the benchmark data proved to be difficult. We hence relied on heuristics to decide if the point cloud produces reasonable results.

3.2 Project Results

Some details of our bundle adjustment model are as follows:

Number of cameras:	13
Number of points:	1030
Total number of parameters:	3168
Total number of residuals:	4646

Reprojection errors:

The reprojection errors are summarised in the figure below. Blue lines represent the errors with the initial guess. As can be seen, the initial errors are quite high, especially at the further end of point indexes. These are points from the later camera frames, where the error is expected to grow as we move further away from the initial frame. The high errors for some points (>2500 pixels) are most likely due to outlier points being retained in the initial guess.

The orange line represents the errors after bundle adjustment and we can see a drastic improvement in the results. The maximum reprojection error here is 166 pixels with an RMS value of 7.14 pixels. Given the lack of robust kernels and other outlier handling techniques in our implementation, this result is deemed acceptable.

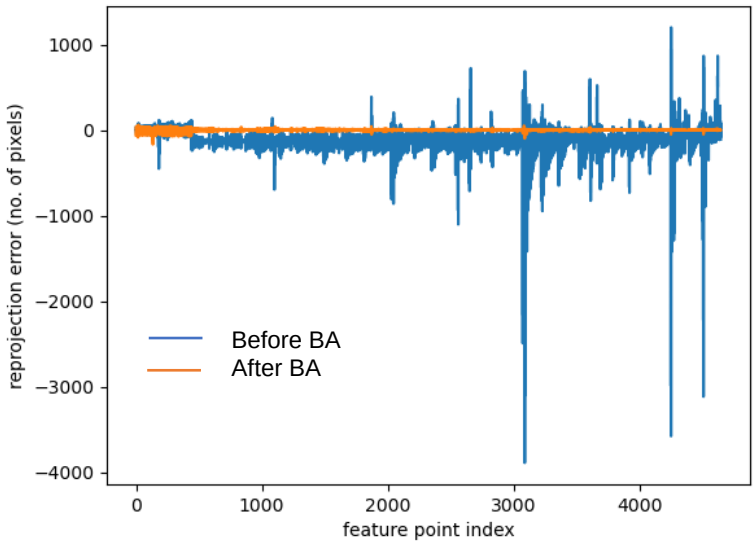


Figure 1: Reprojection errors

Camera position errors:

The reference frame that describes the camera positions roughly relates to the global reference frame as follows.

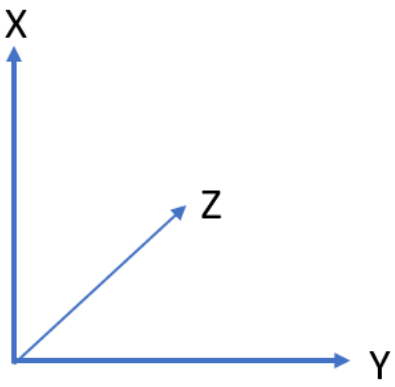


Figure 2: BA Camera Reference Frame

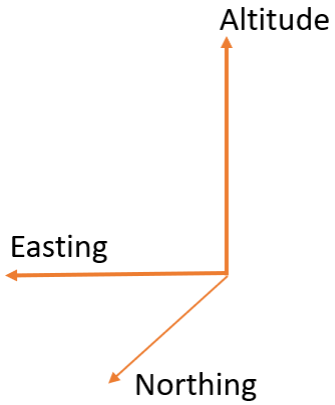
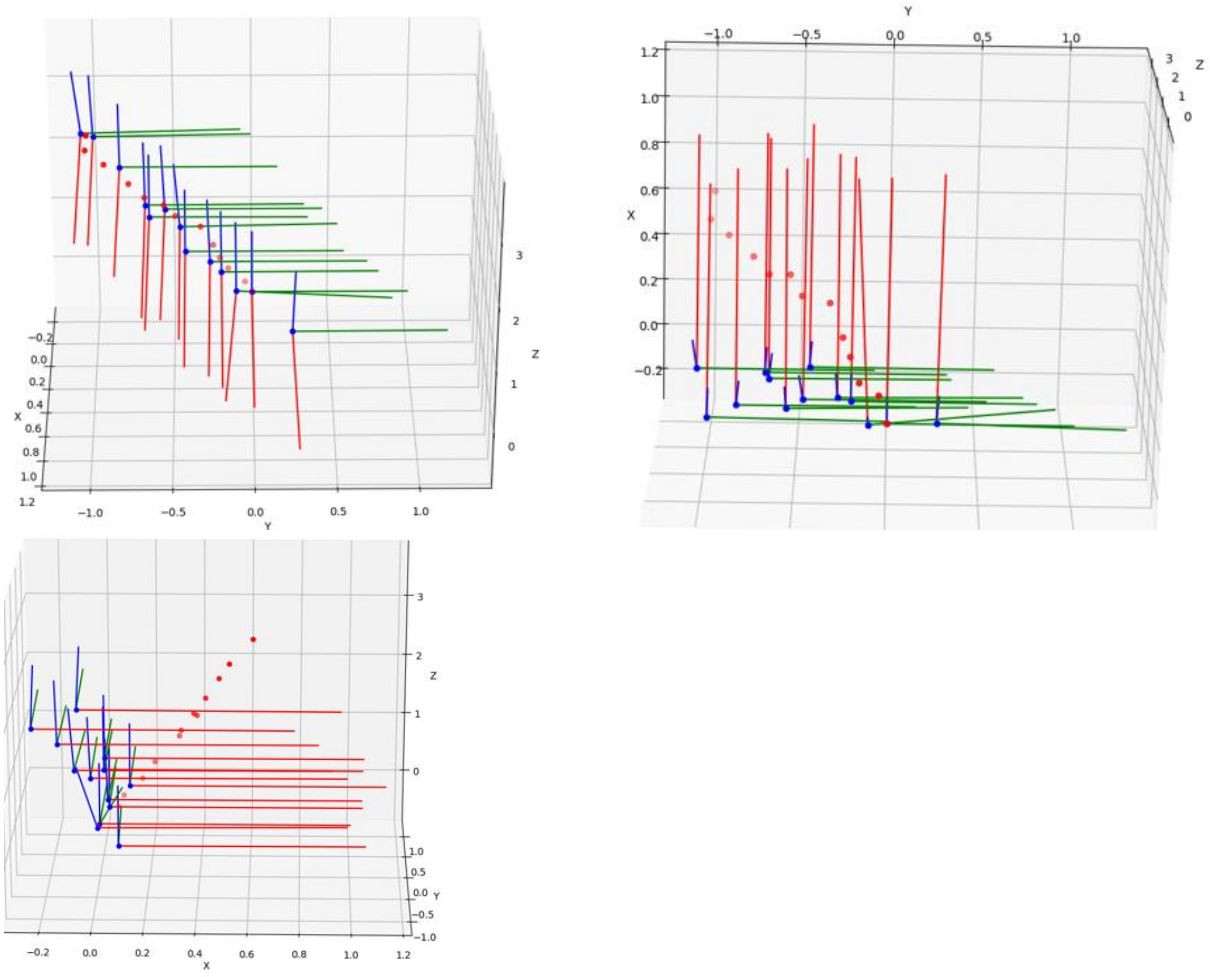


Figure 3: Global Reference Frame

For comparing camera positions with the ground truth, the ground truth coordinates were transformed appropriately to match the BA reference frame.

The figures below show the camera positions for all 13 camera frames. The solid red dots represent the ground truth camera position. Significant errors in position coordinates are seen between the two. The movement trajectory for the BA solution however matches well with the ground truth. Each subsequent frame moves in the positive Z direction (-ve northing) and to the left (+ve easting) and follows an overall trajectory that matched the ground truth. The trajectory in the X direction (altitude) is however very inconsistent.

Figure 4: Camera positions (3 views)



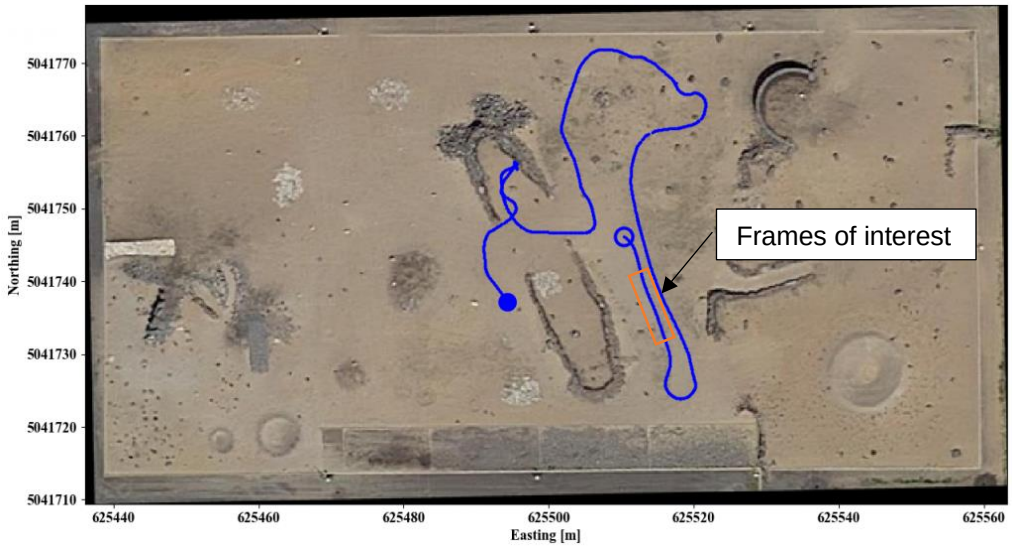


Figure 5: Run 4 travel path

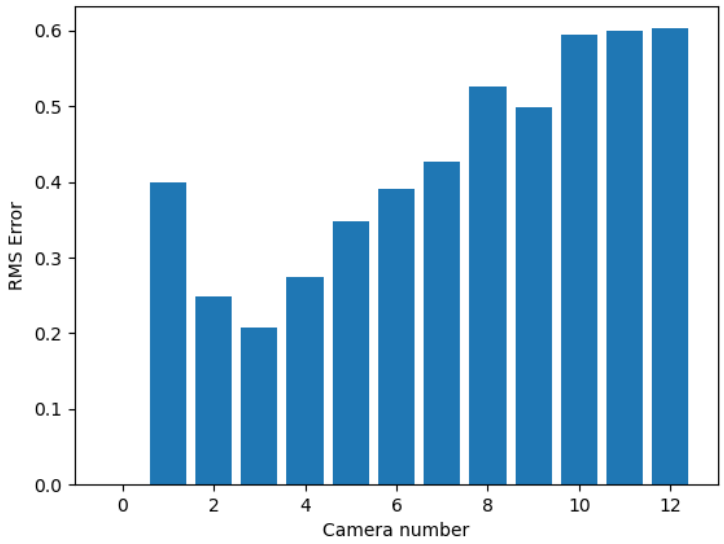


Figure 6: RMS error for camera positions

Point cloud evaluation:

Due to space restrictions on the computer being used, we were unable to download the point cloud ground truth data. This prevented us in from fixing certain 3D points within our BA model and also from comparing the model point cloud results to the ground truth. The results obtained here are photogrammetric and our defined only upto a scale.

Looking at the point cloud, we can still make some observations. Much of the points generated lie close to the camera origin (0,0,0). This corresponds to dense feature points extracted from the terrain. The points are more spread out as we move away from the camera centres, corresponding to more sparse features elsewhere. There is not enough detail to make out any specific features such as treelines or walls.

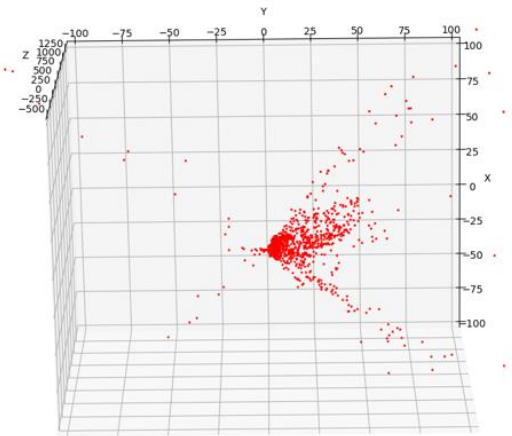


Figure 7: Point cloud results from BA

Figure 8: Terrain image

We were unable to get reasonable results with higher number of frames. Bumping up the number of frames to 20 resulted in a jacobian matrix with over 50 million entries (around 400 mb) and system was

unable to allocate enough memory to process this matrix. We also tried reducing the number of features tracked between frames in order to reduce the size of the jacobian but this resulted in very poor initial guesses and final results exhibited grossly high errors.

References:

1. Tomasi, Carlo. *The 8 Point Algorithm*. Duke.edu, www2.cs.duke.edu/courses/spring19/compsci527/notes/longuet-higgins.pdf.
2. “Large-Scale Bundle Adjustment in Scipy¶.” *Large-Scale Bundle Adjustment in Scipy - SciPy Cookbook Documentation*, scipy-cookbook.readthedocs.io/items/bundle_adjustment.html.
3. Lamarre O, Limoyo O, Marić F, Kelly J. The Canadian Planetary Emulation Terrain EnergyAware Rover Navigation Dataset. The International Journal of Robotics Research. 2020;39(6):641-650. doi:10.1177/0278364920908922 2.
4. Á. P. Bustos, T. Chin, A. Eriksson and I. Reid, "Visual SLAM: Why Bundle Adjust?," 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 2019, pp. 2385-2391, doi: 10.1109/ICRA.2019.8793749.
5. 4. “Bundle Adjustment III - Multi-View Geometry.” Coursera, www.coursera.org/learn/roboticsperception/lecture/yuawM/bundle-adjustment-iii.