**Covering the Uncovered Areas Using Epipolar Line Features to Generate Newly Calculated Views and Drone-based 2D Mapping from Multiple Positions.**

XIN REN, Johns Hopkins University  
PRIYANKA VIJAY HUBLI, Johns Hopkins University  
MINSUNG C. HONG, Johns Hopkins University

Google Street View does a good job in creating a virtual tour. However, users are constrained to limited viewing spots. Here we will intend to take a totally different approach from Google Street View in building a vision map for Johns Hopkins University Homewood campus by using epipolar features. This would be an added functionality to a larger scale mappings like the works of Jeffrey Martin and his group on creating large detailed gigapixel photos. Compared to the conventional method of capturing the 360 degree view by capturing images from high towers using robotic platforms like CLAUSS industry’s products, we attempted on using UAV drone to overcome the limitation on reaching high altitude to create a 2D and 3D mapping of the Homewood Campus. Also, using a UAV drone to perform aerial mapping gives us a solution to the restriction in the mobility viewers have as in London 320 Gigapan, where users can see from just one center position point. This report reflects our primary effort in building a database and the functionality to construct views for the uncovered area to be implemented to the aerial map generated from a UAV drone. Limitations and drawbacks to this approach are also discussed.

**KEYWORDS**

Computer vision, epipolar, panorama, drone-mapping, photo-consistent scene

**1 INTRODUCTION**

Inspired by the works of Jeffrey Martin on his 360 gigapan maps throughout the globe, we began this project to map out the Johns Hopkins’ Homewood campus with an aerial UAV drone. Jeffrey Martin’s works are done by first finding a tall tower in center of cities and use a 360 degree rotator robot with a very nice camera attached on top to take multiple layers of images (~8000 images) to stitch them all up with a vigorous algorithm. Although this method outputs an unforgettable panoramic image with a tremendous resolution even to distances 25km away, the downside of this method is that one is limited to being placed in just one central position to take images from. Viewers cannot change their position where their central positions are located to different positions on the map. This is the bottleneck of the current gigapixel mapping strategies. However, using an aerial drone eliminates this bottleneck issue, granting the photographer to reach altitudes far greater than regular skyscrapers as well as the freedom to take rotating multi-layered panoramic images for 2D mapping and even 3D mapping of selected areas basically from anywhere. This would generate datasets from multiple coordinates, which can be set as movable positions where users online can move into and out from to view the same area with greater perspective. Our goal in this project is to create a bird-eye view online application that will show large sets of panoramas captured using a DJI Phantom3 SE drone from multiple different central positions on Johns Hopkins Homewood campus. Difficulties for aerial drone mapping are in short battery flight time, drone-stability being highly dependent on the weather as windy weather or any kind of precipitation would make data collection difficult, and limitations in changing focal lengths. The figure below shows our big picture goal of drone-based mapping compared to the current Jeffrey Martin’s ground-based mapping method.

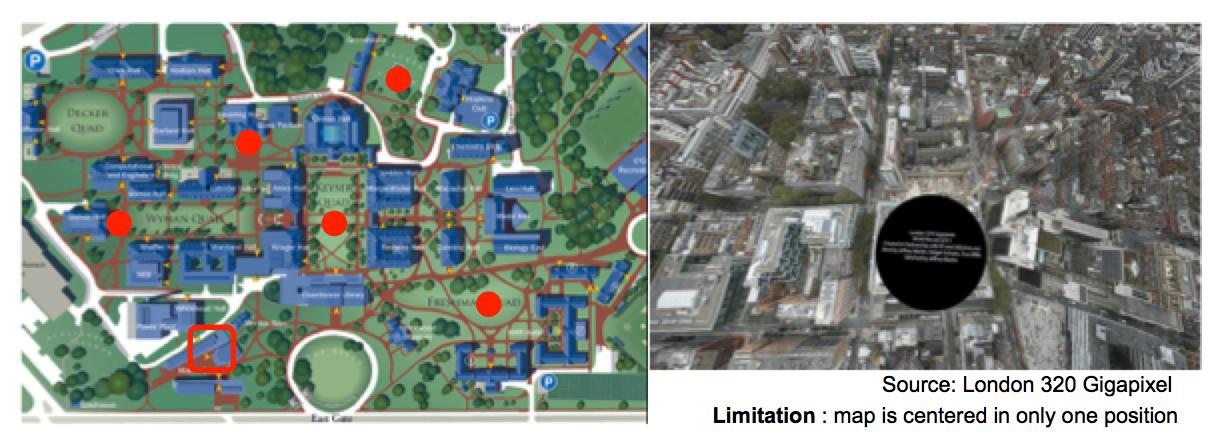


Fig. 1. Picture on the left is showing the central positions(marked with red circle) where our drone will map out the campus from, and the picture on the right is London 320 Gigapan where only one central viewing position is located on.

An additional functionality we attempted on adding is enabling horizontal view of certain areas marked from the bird-eye view map, which when clicked will change the view of the viewers as walker’s point of view, not in a bird-eye view. This is somewhat similar to the Google Street View, which provides panoramic views from various spots along streets, thus creating a vivid virtual tour in a horizontal view. Google’s approach has been to use a fisher-eye camera to collect images in all directions from spots along the streets they covered. For each spot, a spherical panorama is created, thus enabling the user to view the street view in all directions at a given spot along the street. However, limitations are that first, users can select only from limited spots, where panoramas have been created from. And secondly, at least eight images are required for one spherical panorama so a large amount of images need to be collected.

Here we intend on trying a totally different approach from that of Google’s in creating a horizontal virtual tour. Our approach is to collect images from several views for one scene and collect the position and orientation for each image using epipolar features, and thus constructing a database for a given location. Our approach will enable the viewers to take a look, for the first time, at a view that has not been taken by the camera during data collection. The algorithm we developed will generate a new uncovered view that a user will see. The algorithm for generating the uncovered views selects appropriate images from the database and utilizing another algorithm, which uses epipolar features, will generate a newly calculated view.

This paper includes our primary effort in collecting the data required and building the 2D drone-based map and the epipolar feature-based horizontal vision mapping for a certain area in Homewood campus.

**2 DATABASE CONSTRUCTION**

2.1 Image Collection

Two different camera imaging methods have been used in our project for collecting our image datasets. The main goal that we aimed of achieving for the duration of this project was to get the epipolar feature based extraction of new views of image points that are not captured in our dataset. For this part, we had to calibrate our cameras for every new focal length and principle point needed. We made our own checkerboard with block square lengths to be sufficiently large enough for the feature detection to work the long focal length calibration. We divided our calibration steps to three distance lengths: short, medium and long. The beauty of this project is to focus on keeping the image resolution as good as possible for these short, medium and long distance-located objects. The camera used for image collection is Canon EOS Rebel T5i. So far, we captured 159 images to be used in our datasets to be used for the epipolar line-based new image view generation.

For another ambitious goal of ours is to achieve a 2D mapping of our Hopkins Homewood campus with a drone from high altitude. We aimed of adding more central positions that had not been done in the work of Jeffrey Martin on his megacity 320 Gigapixel projects. Jeffrey Martin’s work is definitely something we cannot get close to replicating. However, the plus sides we see in this project are the freedom in mobility to take this drone up to altitude from range of 0m ~ 2000m high, and the freedom in mobility to map out in multiple central points. The drone used for this project is Phantom 3 SE from DJI with 4K resolution with 30 frames/second imaging capability. Altitude for the image collection for this project has been set to 150m. Drone-based aerial mapping allows us far greater freedom in mobility to generate 360 degree mappings from many different positions.

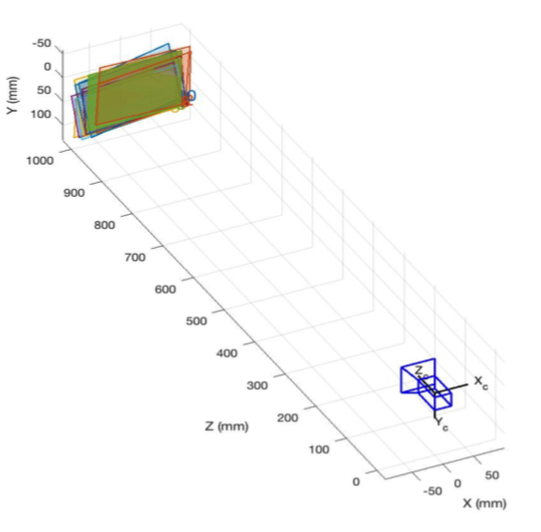
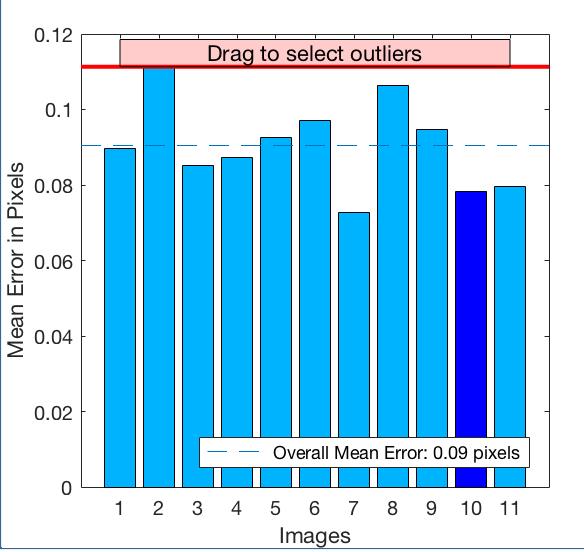
 

Fig. 2. Figure of the Phantom 3 SE (on the left) that’s used in our project, and the outputted overall calibration images layout with reprojection error graph.

2.2 Image Processing

*2.2.1 Calibration*

Image calibration is performed using Matlab’s built-in Image-processing package. Collecting about 10 to 20 images per calibration, we were able to obtain the intrinsic parameters (focal lengths and principle points) for the varying distances (short, medium, and long) needed to capture for our database collection. This process would output another graph that shows the mean error each calibration image has in the calibration process. Threshold value for mean projection error has been automatically set to filter out those images with higher errors. We also had to convert the outputted camera intrinsics (focal length) into real world units as matlab doesn’t know our camera (Canon EOS Rebel T5i) CMOS sensor data. The CMOS in the camera we used has in total 18.5 Megapixels with pixel dimension of 4.3 This is ~0.002074mm/pixel as each pixel width is ~2.074 , which is ~0.002074mm.

The following table shows the intrinsic parameters we have obtained for our images:

|  |  |  |  |
| --- | --- | --- | --- |
| distances | short(~3.5m) | medium(~5m) | long(~9m) |
| focal length | 10.1549mm | 6.6484mm | 6.0672mm |
| principle points | (2446.062,1419.337) | (2603.887,1704.639) | (2574.230,1620.666) |

Table 1. Table of our calibration dataset for the three distance categories

*2.2.2 Feature Matching and Extrinsic Parameters extraction:*

Feature detection is performed in every pair of images using SURF detection. An example of the output of our detection platform is shown on the following figures on the right. We have decided to keep track of the orientations of each corners in order to select out 100 strongest features. Upon detecting features from each image in the pairs, feature matching is performed that matches the feature pairs. The output of this feature detection is shown on the figure 3 below. There are many outliers that needs to be rid of in order to obtain just the right matches. We take care of this as well for every image pairs. Figure 4 shows the output of the function to rid off the outliers for us to obtain just the right matches we need.



Fig. 3. 2x6 montage of the images for one of the locations covered.

**

Fig. 4. Detected features using SURF detection method

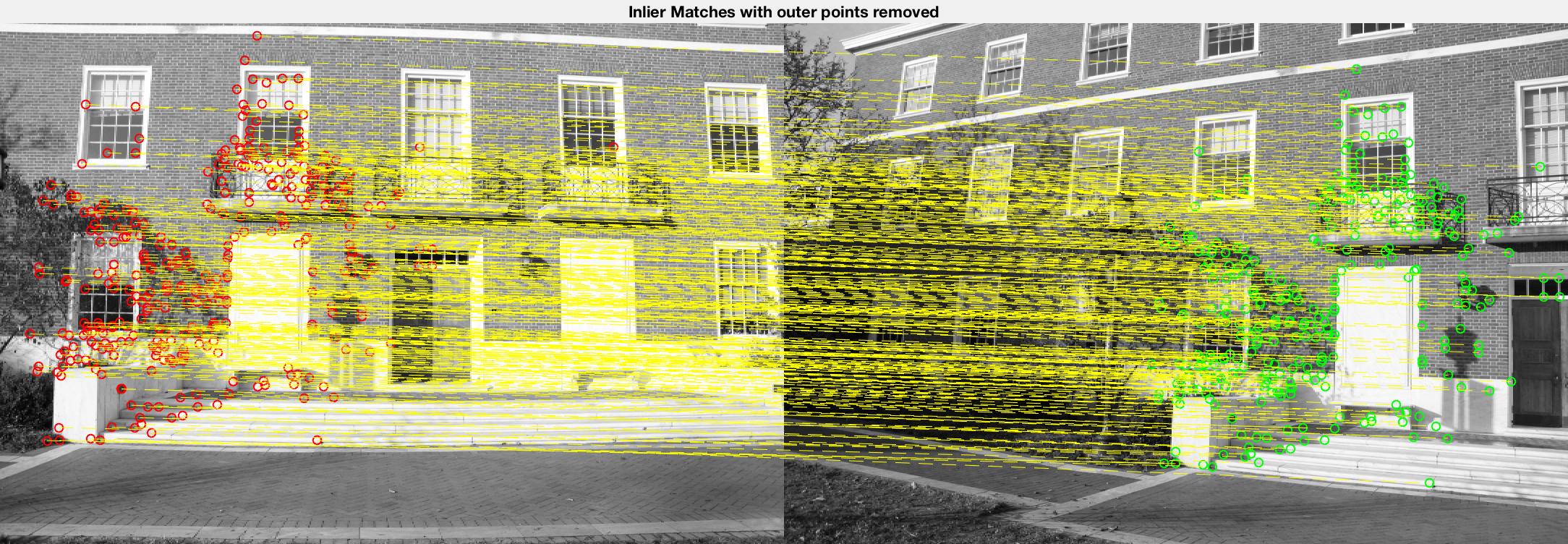


Fig. 5. Feature matching results. Left image shows feature matching without outliers removed. Right image is with outliers removed by using RANSAC feature matching

**3 IMAGE SELECTION**

3.1 Method

After the successful collection of the Database and calculation of the Intrinsic parameters of the camera, we have to use an algorithm which chooses the most suitable images to generate a photo-consistent scene from the point of view chosen by the user.

The Database is created such that the images are categorised according to the positions they were taken from and the focal lengths used. Depending on the location and orientation the user requests to view the scene, this algorithm will choose the relevant images and passes on to the ‘Generation’ Algorithm. This is done by using the concept of nodes in a graph. As we already know the positions from which the Database of images were created, these positions form the nodes of the graph. First, to accurately position the nodes of the graph, we need to find the relative translation and rotation of the nodes from a reference node. The transformation from the reference position to the other positions is calculated by taking the average transformation between the images using the Essential function as described above.

Secondly, the relative transformation is calculated for the location the user requests. Using this transformation, the new position is added as a node in the graph. The orientation at which the user wants to view the scene is used to narrow down the images needed to be selected.

The nearest nodes and the nodes which contribute to the view from the new position is determined using the shortest distance method in the direction specified. This narrows down the Database to just the images taken from these positions. However, all the images taken from these positions are not relevant to us as this collection has images taken 360around the position. Therefore, the number of images can further be reduced. This is done by selecting only the images captured in the provided direction. This gives the relevant images needed for reconstruction.

3.2 Selection Algorithm

Here we will provide the details of the algorithm. First, the camera is calibrated and we have its intrinsic matrix. The Database is created and the images are labeled according to the position and orientation. The essential matrix can be found by using the Essential() function. And the user provides the position and orientation at which they want to view the scene.

**ALGORITHM 1:** Selection Algorithm

*ref\_img ←* Read reference image

**for** ***each*** *position* ***in*** *Database*, **do**

*transformation ← Avg ( Essential(ref\_img, all\_images in position))*

**end**

*plot ←* Plot the nodes in the graph

*given\_pos ←* position from where the user wants to view the scene, add this to the graph

**for** ***each*** *position* ***in*** *Database*, **do**

**if** *distance* ***is in*** *Range,* **do**

*nodes*  ← All the nodes closest to the *given\_pos*

**end**

**end**

**for** ***each*** *nodes* ***in*** *graph*, **do**

**if** *orientation* ***is in*** *given\_pos\_direction,* **do**

*images*  ← All the images closest to the *given\_pos* and in its direction

**end**

**return** *images*

**end**

3.3 Test Results

The Selection Algorithm is highly susceptible to errors in the Database. This can be seen in the images shown below. Ideally, the below two graphs should be similar. However, the collection of the database has to be accurate. To compensate for the inconsistencies in the database, a few images taken from position 7 were added to 8 and the images taken from position 3 were added to position 5. Hence, the errors in locating position 5 and 8 have occured. Therefore it can be inferred that Database collection is a very crucial step.

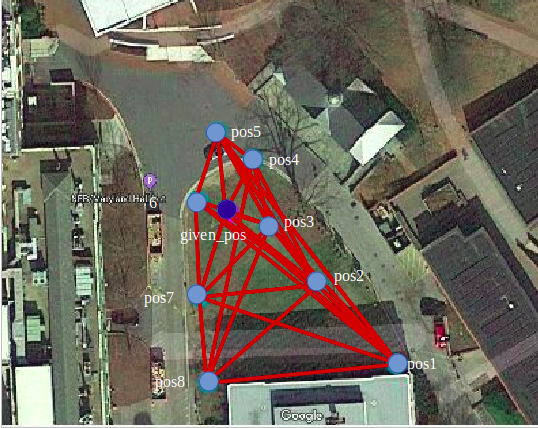


Fig. 6. The representation of the positions from where the images were captured. Given\_pos is from where the user has requested to view the scene.

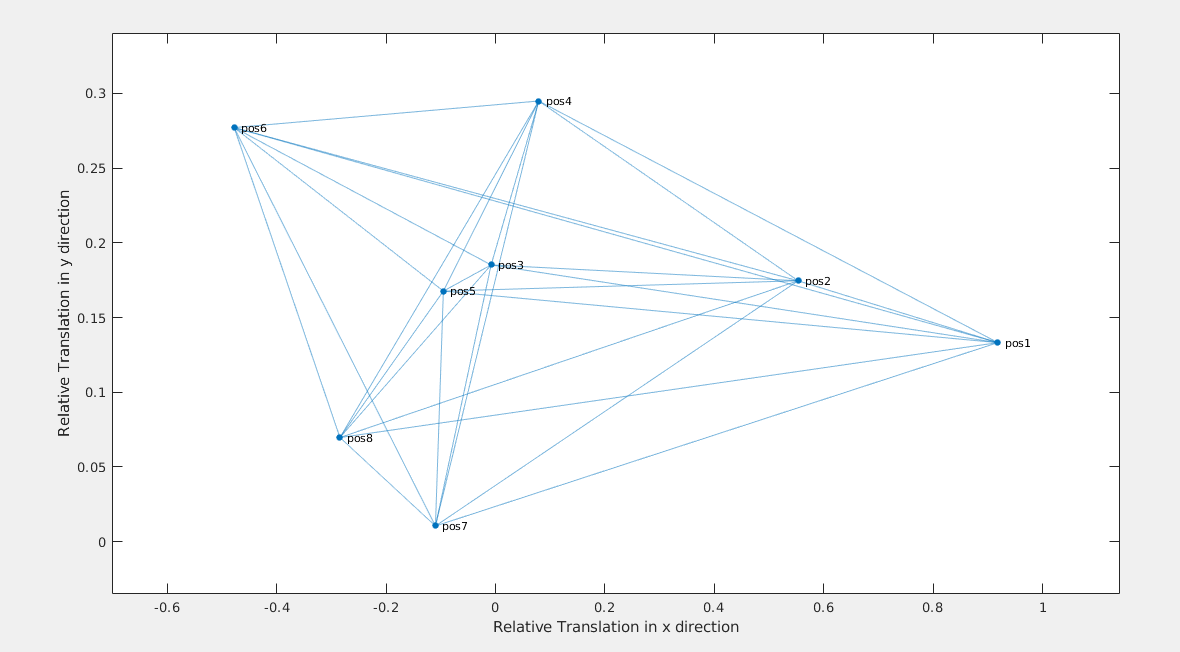


Fig. 7. The generation of the graph after calculating the average transformation.

Now, when the user requests to view a scene from a particular location, the position of this location with respect to the reference image is calculated and added as a node to the graph above. To be clear, there are multiple reference images as one image cannot cover all the scenes. Hence, there is a reference image for each set of orientations. For the given\_pos, the graph is generated and the images from the nodes represented are chosen and further filtered as explained above.

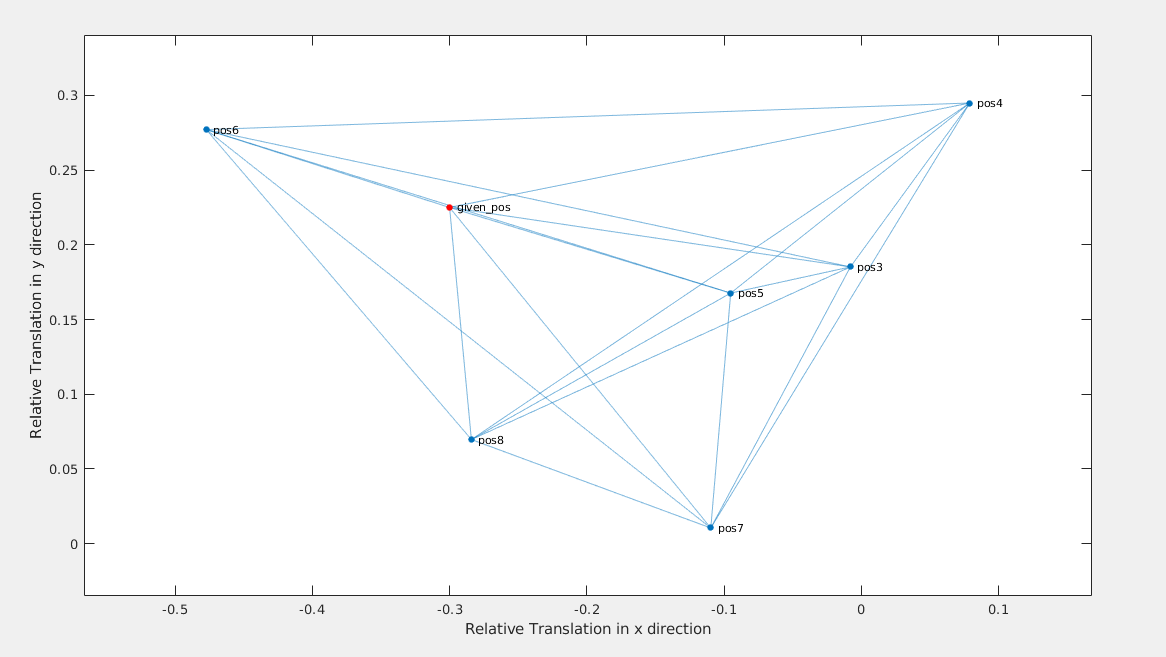
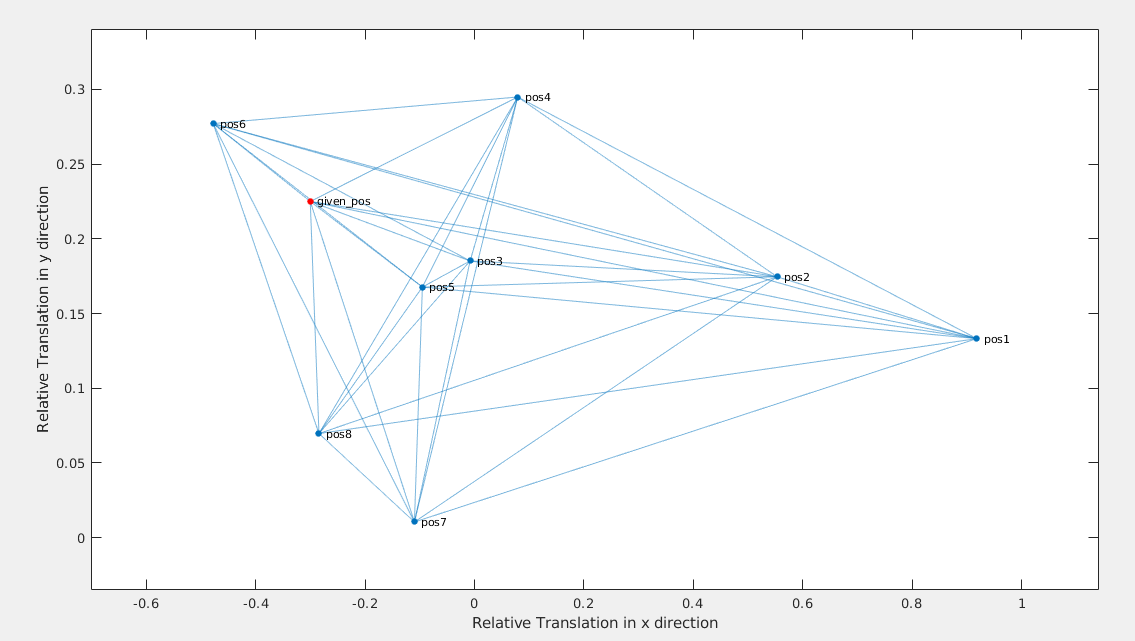


Fig. 8a, 8b. The given\_pos is added as a node in the graph in Fig a. After the selection algorithm, the selected nodes are depicted in Fig b. The orientation at which the user wants to view the scene is in positive y direction and normal to the x direction ⇑.

The results from the Algorithm shows us that there are nodes which may not be necessary but still get selected due to its close proximity to the given\_pos. For example, in the above graph, position 7 and 8 may not be necessary. However, they are as close to the given\_pos as position 3 and 5. Hence, eliminating these positions is not feasible.

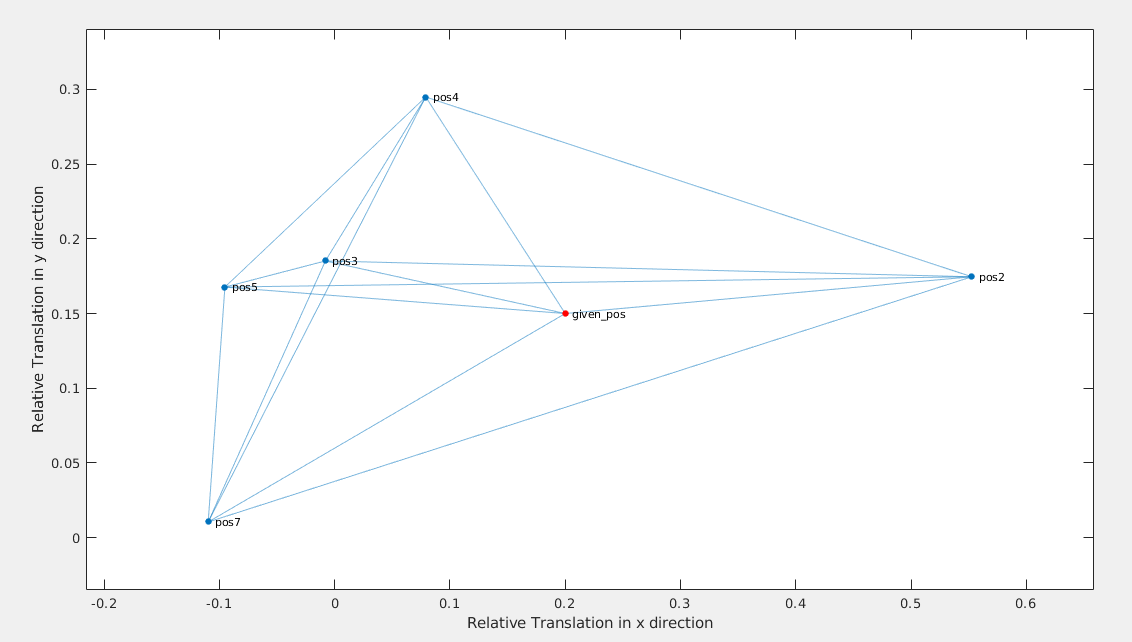
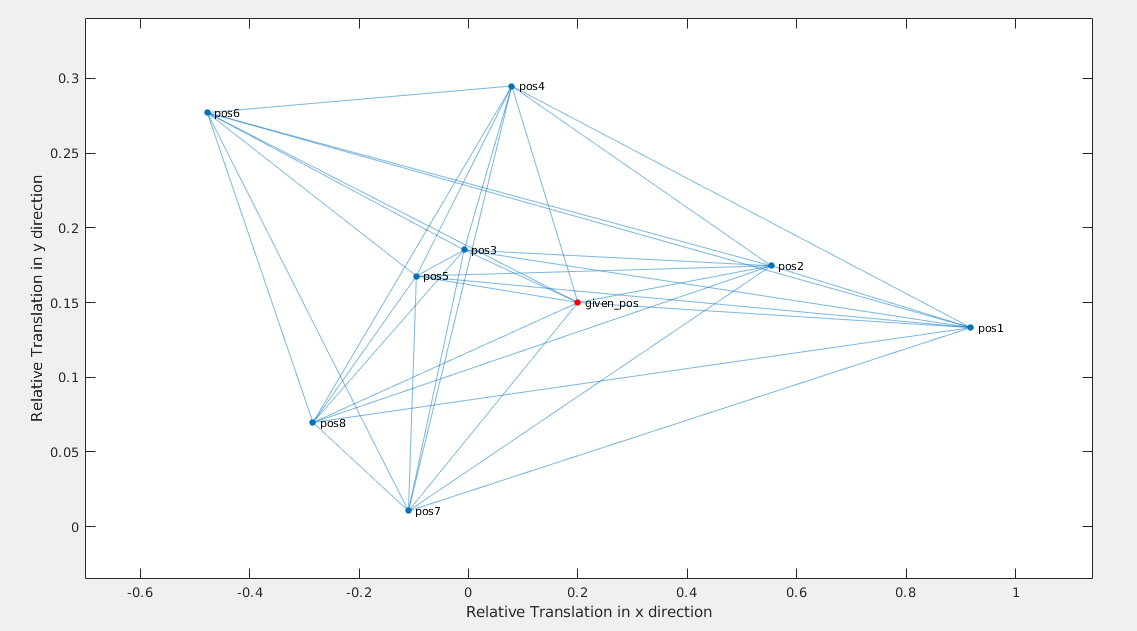


Fig. 9a,9b. The given\_pos is added as a node in the graph in Fig a. After the selection algorithm, the selected nodes are depicted in Fig b. The orientation at which the user wants to view the scene is in negative x direction and normal to the y direction ⇐.

Consider another example where the given\_pos is as defined in Fig a. Then the selection algorithm returns the nodes in Fig b. These nodes do not have recurring images. Therefore, this is a quite accurate result for the Selection Algorithm.

Overall, it can be said that the Selection Algorithm is prone to errors when the Database has inconsistencies and it fails to be efficient for a few positions. For not so large Databases, this algorithm can still be used. However, if the number of images increases then this would be computationally heavy unnecessarily.

**4 VIEW RECONSTRUCTION**

4.1 Overview

Assume the user has given the preferred viewing position and orientation and selection algorithm has successfully picked out the relevant images in the database, our next task is to generate a photo-consistent scene from the point of view chosen by the user. There are several approaches to this problem. Here we will try a totally different approach to get a photo-consistent scene with minimum photos and as fast as possible. We do not require the scene to be exactly the same as the image from the same camera viewpoints, but we need the image to be decent and accurate.

We can summarize our problem as, given two images with known relative rotation and translation, how to reconstruct a scene for a given viewpoint. We mainly take advantage of the following fact. Suppose we have *Point A* on *Image A* and *Line A* is the corresponding epipolar line on the scene to be reconstructed, *Point B* on *Image B* and *Line B* is the corresponding epipolar line on the scene. If *Point A* and *Point B* are the matching points for two images, then *Line A* and *Line B* has an intersection and this intersection point is also the matching point for *Point A* or *Point B*. In this way, we can scan all the points in *Image A* and *Image B* and thus reconstruct the scene point by point. However, this is super computationally expensive and we can only reconstruct the overlapping parts of *Image A* and *Image B.*

As a result, instead of trying to reconstruct the scene point by point, we can do it line by line. Choose a pair of matching points in *Image A* and *Image B* to get their corresponding epipolar lines, which contain overlapping information. Get two pairs of matching points along the epipolar lines and we will obtain two intersection points on the scene to be reconstructed. These two intersection points are enough to define a line. Since we already know the content of the line from *Image A* and *Image B*, we can fill in the line with the content.

Picking several points along the epipolar line on *Image A* and also a few on *Image B* and repeat the process, we can reconstruct the scene pretty decently with only two images if they contain enough information. However, this is still not fast enough. We can play the trick in sampling. A point will not be sampled again if it is already on or near an already sampled epipolar line. We can achieve our goal now with accuracy and speed. At last we will filter the scenes to fill in holes and to eliminate artifacts.

4.2 Generation Algorithm

Here we will provide the details of the algorithm. First, the camera is calibrated and we have its intrinsic matrix. The relative position between two given images and are pre-calculated and given in database. Thus the essential matrix between *Image A* and *Image B* can be calculated and obtained. Since the user has given us the position and orientation of viewpoint, we can similarly calculate and .

**ALGORITHM 2:** Generation Algorithm

*start*  ← a point out of ***BorderRange***

**for each intervaled** *point* ***on*** *EpipolarLine*, **do**

*pts1*  ← *point*

*pts2*  ← *H (points)*

*Line1*  ← epipolar line given *pts1*

*Line2*  ← epipolar line given *pts2*

*Intersect\_point*  ← intersection of *Line1 Line2*

**if** *Intersect\_point* ***is in BorderRange***, **do**

*end* ← *Intersect\_point*

**if** *start* ***is in BorderRange****,* **do**

*Line\_toBeFilled*  ← line ( *start, end* )

*Line\_toFill*  ← line ( previous used *point* ***on*** *EpipolarLine, pts1*)

fill the content of *Line\_toFill* to *Line\_toBeFilled*

*start* ← *Intersect\_point*

**end**

**end**

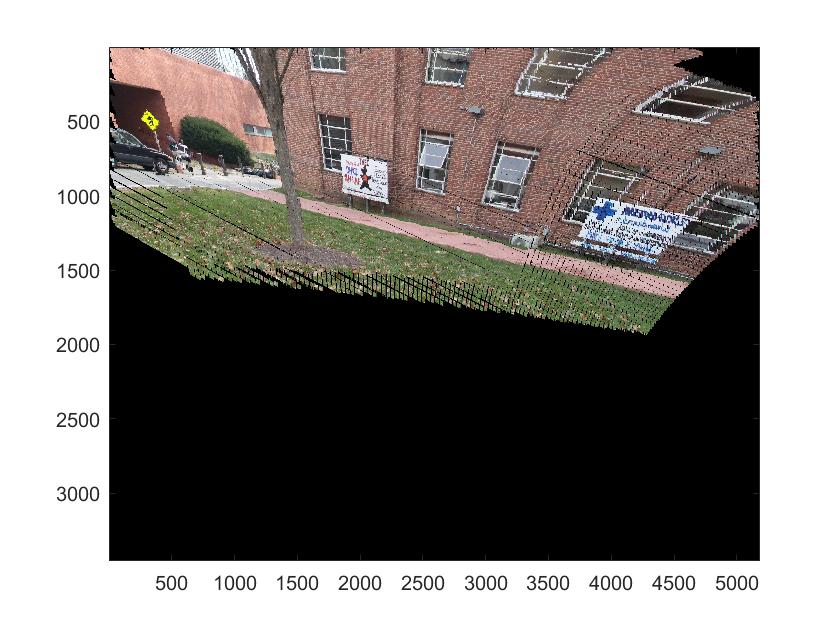
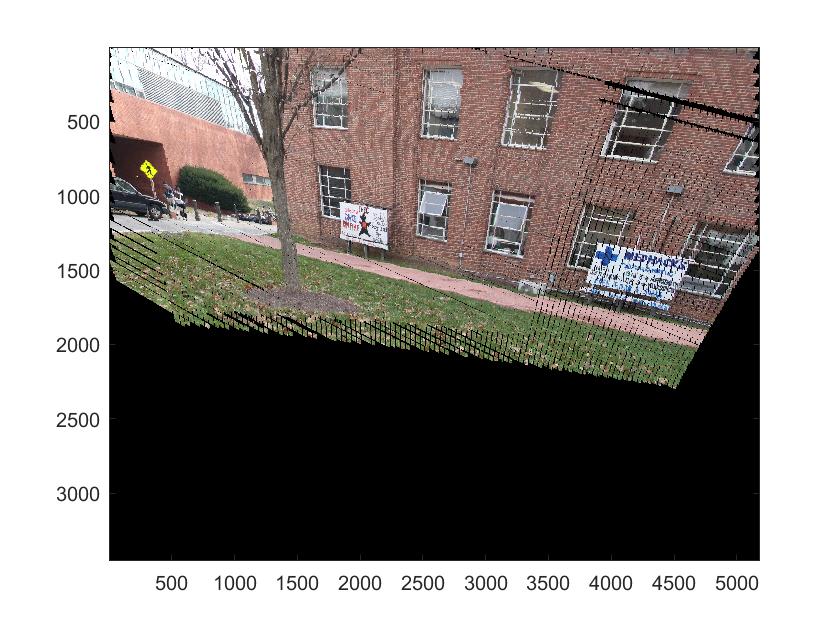
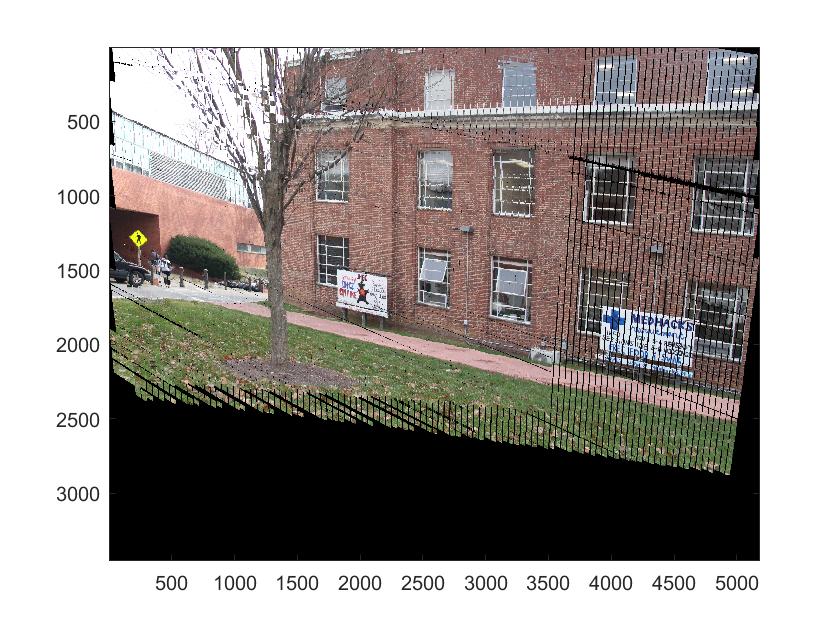
**end**

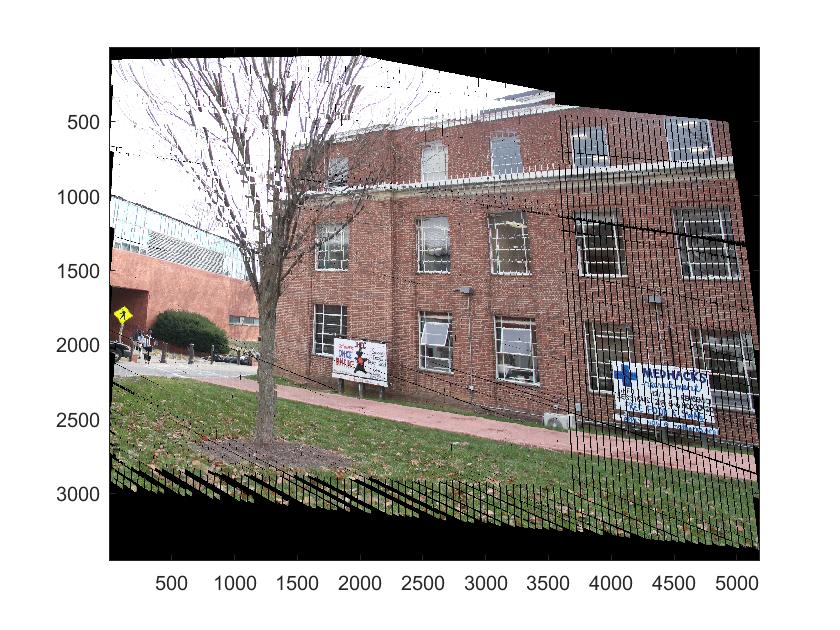
This is the algorithm for filling the content of one epipolar line to the scene. The repetitive steps are similar. To check if an epipolar line needs to be repeated for the step, set a matrix the same size of the 2D image initialized as zeros. When an epipolar line is evaluated, convert the corresponding elements in to 1. If an epipolar line is within 10 pixels from an already evaluated epipolar line, there is no need to evaluate it.

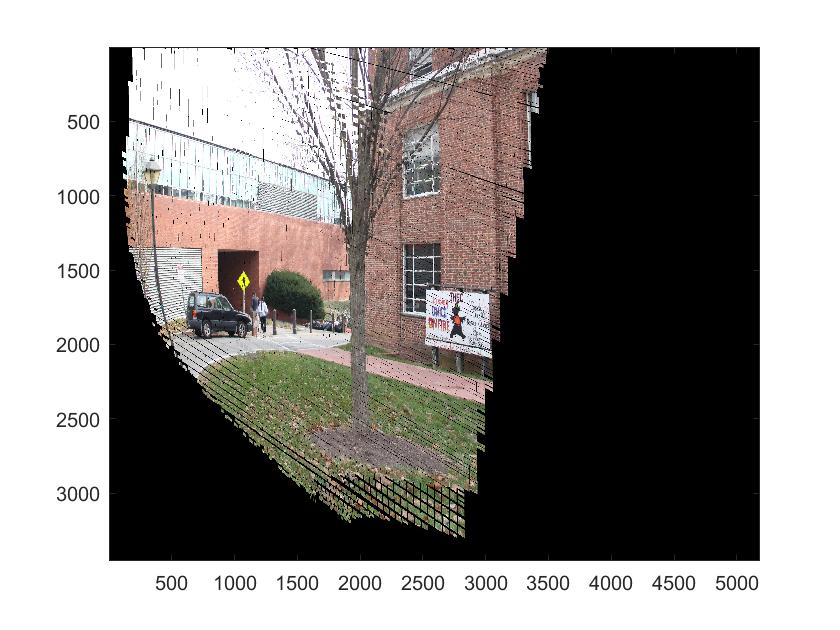
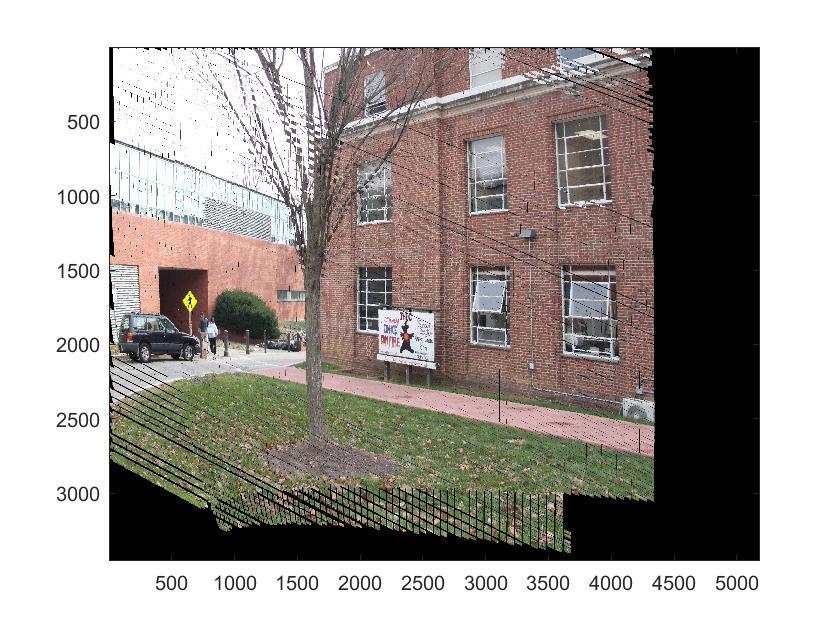
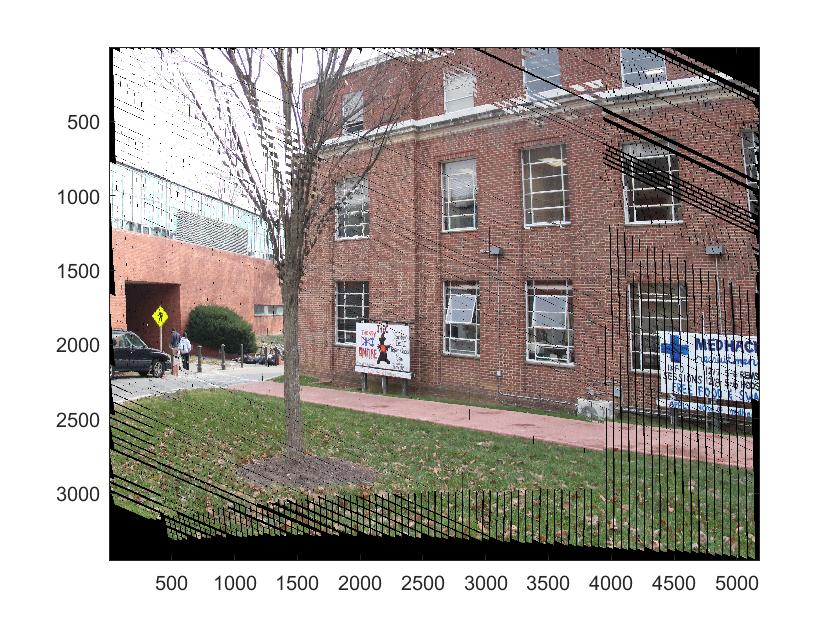
4.3 Test Results

As shown in figure x, two images work reasonably well when the view is only slightly diverted from the given ones. While the distortion gets larger, black area begin to emerge as the information contained is not enough. What’s more, the scene gets warped if the diversion is large, which makes the image look very unreasonable.









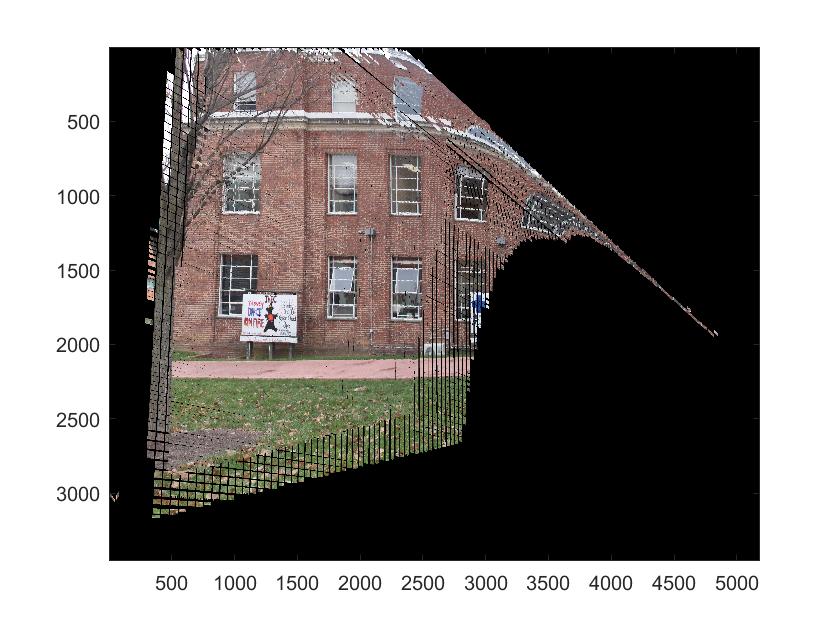
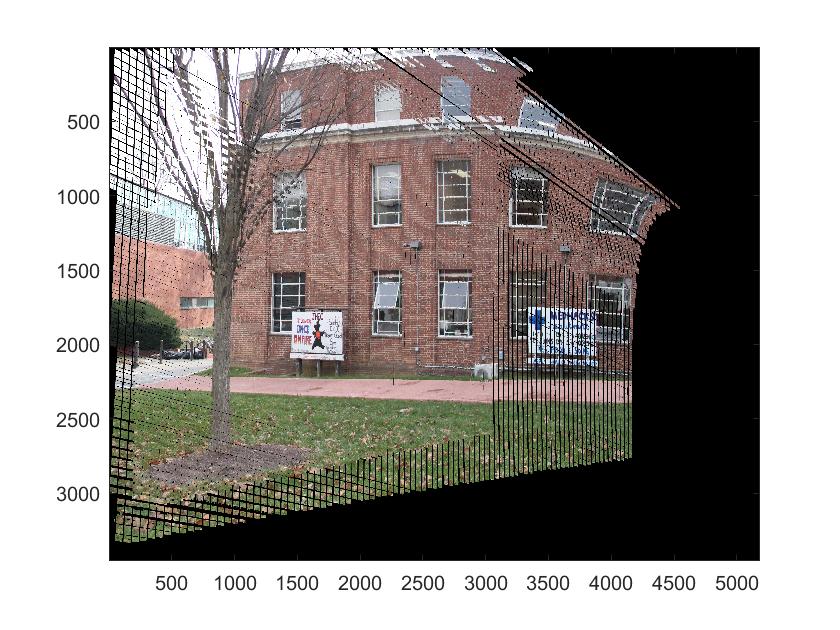
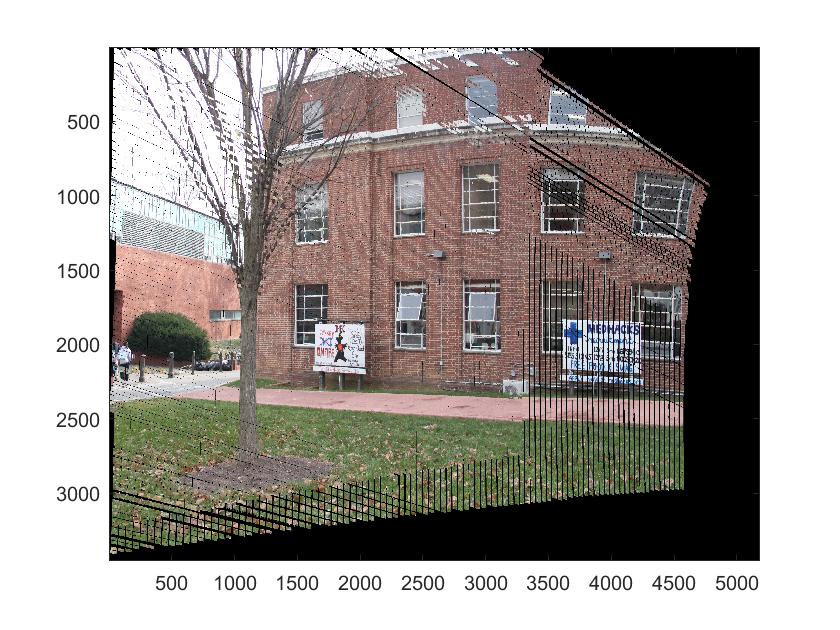
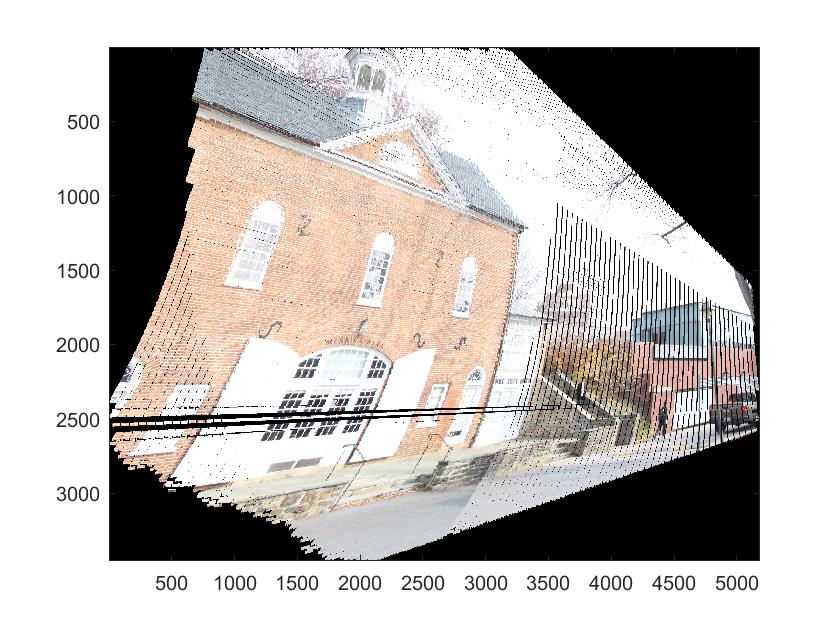


Fig. 10. First row: given images. Second row: a view diverted for +5 deg (x axis); a view diverted for +20 deg (x axis); a view diverted for +30 deg (x axis). Third row: a view diverted for -5 deg (x axis); a view diverted for -20 deg (x axis); a view diverted for -30 deg (x axis). Fourth row: a view diverted for +5 deg (y axis); a view diverted for +10 deg (y axis); a view diverted for +20 deg (y axis). Fifth row: a view diverted for -5 deg (x axis); a view diverted for -10 deg (y axis); a view diverted for -20 deg (y axis).

Ideally, multiple images should be used to reconstruct a complete scene. However, when multiple images are used, error in matching and fitting accumulate and fail to give us a good results. Our next step is trying to understand why scenes constructed using multiple images are off and how to solve the error accumulation problem.

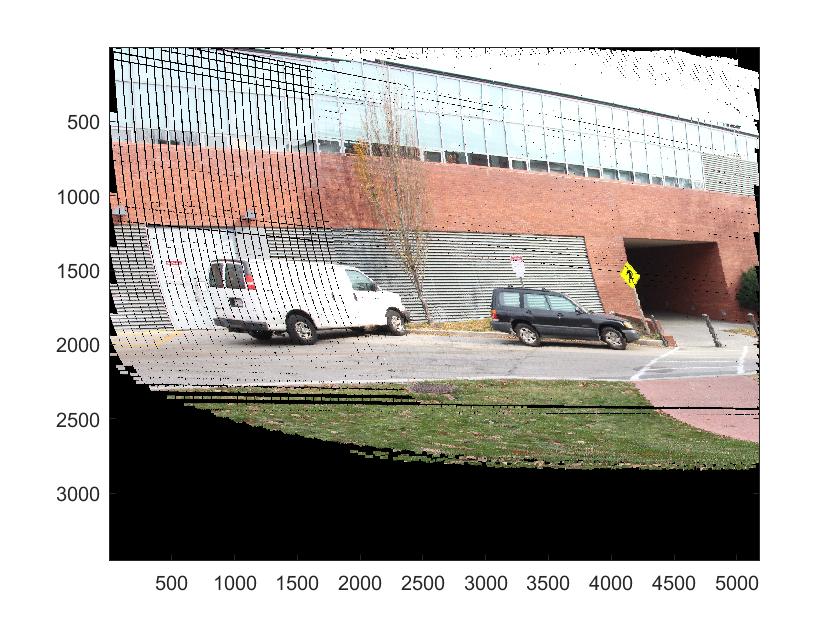
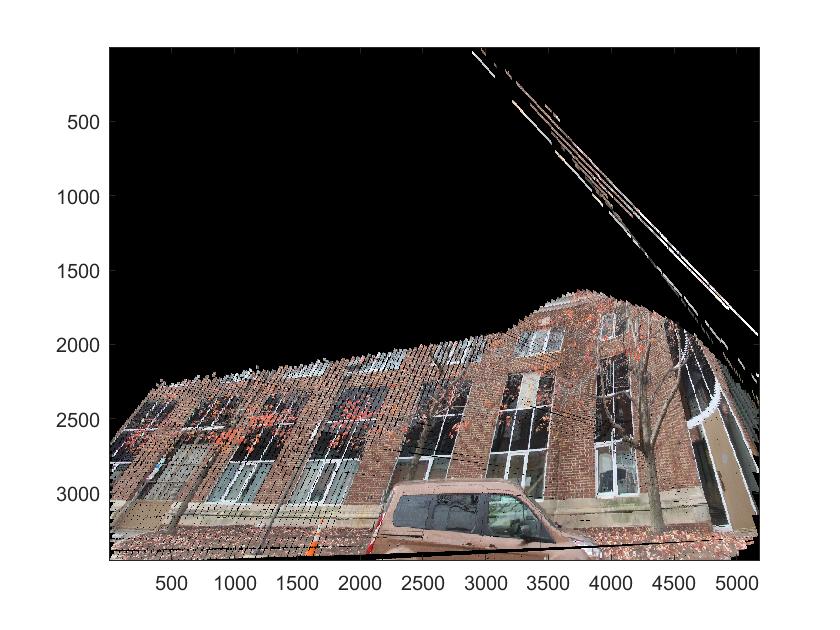
 

Fig. 11. Some of other results

5. MAPPING WITH DRONE

Although this method of aerial mapping with an UAV drone has the mobility advantage compared to the conventional fixed-point 360 mapping from tall skyscrapers, there were several difficulties faced with this method as well. Timing was a big factor for this method. Drone was flown up to 150m above sea-level to obtain our dataset. Although this drone (DJI’s Phantom 3 SE) allows 4K imaging resolution, the quality of the images depended heavily on the intensity of the light from the sun. Also, as the drone rotates about the yaw-axis, if the weather is windy the pressure from the wind causes the drone to drift away significantly.

Future works with UAV are 3D mapping the landscape being covered, and developing an online web platform where users can access these panoramas from a viewing window that can be controlled and be able to move into several available central positions, where other panoramas have been stored at.

6. CONCLUSION

The project was clearly divided into 3 main different parts. One being calibrating the camera, collection of data and deducing the translation and rotation between the images. The second being the selection algorithm and lastly, the generation algorithm. During the development of this project, there were different hurdles which we came across. To concisely put it, first,the collection of data should be methodical. Errors in collection of data leads to the errors in all the consequent processes. Second, the selection algorithm can be optimized to work for large databases. Third, the generation algorithm generates images where the scene gets warped if the diversion is large, which makes the image look very unreasonable. Therefore, this works well when the view is only slightly diverted from the given ones. To conclude, this method of generating images works well with a few constraints added and with accurate data sets.

REFERENCES

|  |  |
| --- | --- |
| [1] | Patricia S. Abril and Robert Plant. 2007. The patent holder’s dilemma: Buy, sell, or troll? *Commun. ACM* 50, 1 (Jan. 2007), 36–44. DOI: http://dx.doi.org/10.1145/1188913.1188915 |
| [2] | I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. 2002. Wireless Sensor Networks: A Survey. *Comm. ACM* 38, 4 (2002), 393–422. |
| [3] | David A. Anisi. 2003. *Optimal Motion Control of a Ground Vehicle*. Master’s thesis. Royal Institute of Technology (KTH), Stockholm, Sweden. |
| [4] | P. Bahl, R. Chancre, and J. Dungeon. 2004. SSCH: Slo.ed Seeded Channel Hopping for Capacity Improvement in IEEE 802.11 Ad-Hoc Wireless Networks. In *Proceeding of the 10th International Conference on Mobile Computing and Networking* (MobiCom’04). ACM, New York, NY, 112–117. |
| [5] | Kenneth L. Clarkson. 1985. *Algorithms for Closest-Point Problems (Computational Geometry)*. Ph.D. Dissertation. Stanford University, Palo Alto, CA. UMI Order Number: AAT 8506171. |
| [6] | Jacques Cohen (Ed.). 1996. Special Issue: Digital Libraries. *Commun. ACM* 39, 11 (Nov. 1996). |
| [7] | Bruce P. Douglass. 1998. Statecarts in use: structured analysis and object-orientation. In *Lectures on Embedded Systems*, Grzegorz Rozenberg and Frits W. Vaandrager (Eds.). Lecture Notes in Computer Science, Vol. 1494. Springer-Verlag, London, 368–394. DOI:http://dx.doi.org/10.1145/3-540-65193-429 |
| [8] | Ian Editor (Ed.). 2008. *The title of book two* (2nd. ed.). University of Chicago Press, Chicago, Chapter 100. DOI: http://dx.doi.org/10.1145/3-540-09237-4 |