Real-Time UAV Global Pose Estimation Using 3D Terrain Engine

Ali Abbas, Assef Jafara, Zouhair Dahrouja
HIAST Higher Institute for Applied Science and Technology,
Damascus, P.O.box: 31983, Syria.
ali.abbass@hiast.edu.sy

***Abstract*—We present a new approach that automatically estimates global pose for a UAV in real-time using 3D terrain engine. Inaccurate auxiliary sensors on the UAV were used to obtain initial real camera pose that moves the virtual camera inside the 3D terrain engine. We, then automatically found multiple matches between the two images to find the 3D coordinates of the matches using the 3D terrain engine. Finally, we tested the co-planarity of the 3D points under the camera, depending on this test. We used coplanar or non-coplanar algorithm to estimate accurate global camera pose. We executed feature detection, description and pair wise matching algorithms on GPU to get a suitable frame rate (12 FPS) needed in the navigation applications. The proposed approach has been tested on a synthetic and real data. Experimental results proved the feasibility and robustness of the proposed approach, and the precision was the same order as the 3D terrain engine used. Finally, we can say that the 3D terrain engine succeeded when other methods failed.**

***Index Terms*—UAV; Pose Estimation; 3D Terrain Engine; Local Features.**

I. INTRODUCTION

During the past decades, the UAV (Unmanned Aerial Vehicle) has been used in monitoring and reconnaissance operations, such as fire detection and monitoring of oil pipelines and border areas. However, these tasks were only done by rich countries because of the high costs for these UAVs. Recent developments in material science, control engineering and communications have led to the development of a low cost UAV capable of carrying a digital camera and fitted with a communication system and a set of sensors that helps to determine the location and orientation of the UAV, such as GPS (Global Positioning System) and IMU (Inertial Measurement Unit). However, cheap sensors suffer from the problem of low accuracy and high sensitivity to noise. The UAV itself is unstable and under the influence of wind. All of these reasons make it impossible to accurately estimate the location and angles of the UAV when using these sensors only. For this reason, we used digital camera mounted on the UAV as an optical sensor to improve the estimation accuracy.

The rest of this paper is organized as follows. We first define problem formulation, related works, coordinate systems, real and virtual camera models, 3D terrain engine and calibration process, then we discuss our approach steps, our testing environment and finally the experimental results are presented followed by conclusions at the end.

II. PROBLEM FORMULATION

Given an image from a UAV, our goal is to estimate its global pose in real-time using a 3D terrain engine. We assumed that the camera's FOV (Field of View) is known, as well as an approximate pose taken from a set of inaccurate sensors mounted on the UAV, GPS for location, IMU for angles. Given these hypotheses, we were looking for the accurate location *(longitude, latitude, altitude)* and the accurate rotations *(azimuth, elevation, roll)* that map the camera frame to the frame of the 3D terrain engine.

III. RELATED WORKS

In general, to solve the problem of global pose estimation for a camera with 6-DOF (Degrees of Freedom), we need a 3D model (consists minimum from 3 points) which we can automatically find their 2D projections on the image plane. In the case where the camera is mounted on a UAV, two dominant approaches help to solve this problem: the first one is based on the assumption that the earth model is a plane textured with satellite Geo-referenced images [1] (3D model here is a plane). This approach turns to image-image registration problem, which basically uses feature-based methods such as SIFT (Scale Invariant Feature Transform [2]) or SURF (Speeded Up Robust Features [3]) techniques. The second one is proposed by Hyon Lim et al. for real-time camera localization inside 3D model reconstructed from off-line using SFM (Structure from Motion) (3D model here is a point cloud). This algorithm efficiently combines key-point tracking in video with 2D-3D point matching, without using salient features. This approach works well when the camera faces a robust represented part from 3D point cloud [4]. To avoid the limitations of the previous approaches, we proposed a new approach, which depends on a 3D terrain engine. The 3D terrain engine allows us to deal with a complex model of the earth and process multiple views of the same target. After finding the correspondences between the 3D-2D points, the problem turns to estimate pose from those correspondences. The traditional solutions for this problem can be divided into two groups: closed-form solutions and iterative solutions. In [5], there is a thorough comparison between different closed-form solutions. In this work, we used iterative algorithm, which uses multiple points to handle errors of the camera's measurements [6].

IV. PROBLEM GEOMETRY

Here, we discuss the coordinate systems, real camera model (which give us real image), virtual camera model

(which give us virtual image), 3D terrain engine and calibration process.

A. Coordinate systems

We used geographic coordinates *(longitude, latitude, altitude)* to represent a point over the earth surface, and we used normalized Euclidean coordinates *(X,Y,Z)* for rendering earth and implementing algorithms. We also used UTM (Universal Transverse Mercator) [7] map projection to project geographic coordinates on earth tangent plane to calculate azimuth angle relative to the northpole and elevation angle from the horizontal plane. To do this, we defined a non-linear transform *LL2UTM* to convert from geographic to UTM coordinates *(easting, northing) = LL2UTM(lon, lat)* and another non-linear transform *UTM2LL* to convert from UTM to geographic coordinates *(lon, lat) = UTM2LL(easting, northing)*.

B. Geographic to Euclidean coordinates conversion

We defined a non-linear transform LLA2XYZ to convert from geographic to Euclidean coordinates (X, Y, Z) = LLA2XYZ(lon, lat, alt) by the following equations:

$$r = \frac{alt}{R_{earth}} + 1 \tag{1}$$

$$Y = r\cos\left(\frac{\pi}{2} - lat\right) \tag{2}$$

$$Z = -Y\cos(lon)\tan\left(\frac{\pi}{2} - lat\right) \tag{3}$$

$$X = Z\tan(lon) \tag{4}$$

where: R_{earth} = 6378137 is Earth radius in meters.

C. Euclidean to Geographic coordinates conversion

We defined a non-linear transform *XYZ2LLA* to convert from Euclidean to geographic coordinates *(lon, lat, alt) = XYZ2LLA(X, Y, Z)* by the following equations:

$$r = \sqrt{Z^2 + X^2 + Y^2} \tag{5}$$

$$alt = R_{earth}(r - 1) \tag{6}$$

$$ion = \arctan\left(\frac{X}{Y}\right) \tag{7}$$

$$lat = \frac{\pi}{2} - \arccos\left(\frac{Y}{r}\right) \tag{8}$$

D. Azimuth and Elevation angles calculation

If we have two geographic points (lon1, lat1, alt1) and (lon2, lat2, alt2), we need to calculate azimuth and elevation angles between them; hence, we defined a function (azi, ele) = AziEleAngles(lon1, lat1, alt1, lon2, lat2, alt2) implemented by the following equations:

$$(e_1, n_1) = LLT2UTM(ion_1, lat_1) \tag{9}$$

$$(e_2, n_2) = LLT2UTM(ion_2, lat_2) \tag{10}$$

$$len = \sqrt{(e_2 - 2_1)^2 + (n_2 - n_1)^2} \tag{11}$$

$$elevation = \arctan\left(\frac{alt_2 - alt_1}{len}\right) \tag{12}$$

$$azimuth = \arctan\left(\frac{e_2 - e_1}{n_2 - n_1}\right) \tag{13}$$

E. Real camera model (real image)

We represented the points in the homogeneous coordinates, where a 3D point v in world coordinate system is represented as (X, Y, Z, 1)T and its projection on the image plane v' is represented in the camera coordinate system as (x, y, 1)T and given by:

$$v' = Rv + T \tag{14}$$

where $T=(T_x, T_y, T_z)^T$ is a translation vector and R is 3×3 rotation matrix, and this mapping can be defined by the perspective projection equation as:

$$v' = P_m v \tag{15}$$

where P_m is the 3×4 projection matrix and can be decomposed as:

$$P_m = K[R \mid T] \tag{16}$$

where K is a 3×3 upper triangular matrix, specifying the internal camera calibration parameters.

$$K = \begin{bmatrix} f_x & \alpha & P_x \\ 0 & f_y & P_y \\ 0 & 0 & 1 \end{bmatrix} \tag{17}$$

where f_x, f_y are the focal lengths in the x and y directions, α is the skew parameter, and (p_x, p_y) is the principal point location. Since the camera is known a priori, it may be calibrated off-line to find f and the other components of K.

Given P_m and the depth *winZ* at each pixel *(x, y)* in the image from 3D terrain engine (see Figure 2), the corresponding 3D point v can be obtained using equation (15).

F. Virtual camera model (virtual image)

In this research, we used OpenGL library to represent virtual camera, which is defined via view parameters *eye* $e(e_x, e_y, e_z)$, *view* $v(v_x, v_y, v_z)$, *right* $r(r_x, r_y, r_z)$ and *up* $u(u_x, u_y, u_z)$ measured in world space (see Figure 1).The view matrix M is given by:

$$K = \begin{bmatrix} r_x & r_y & r_z & A \\ u_x & u_y & u_z & B \\ -v_x & -v_y & -v_z & C \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{18}$$

where $A=-(e.r)$, $B=-(e.u)$ and $C=-(e.v)$.

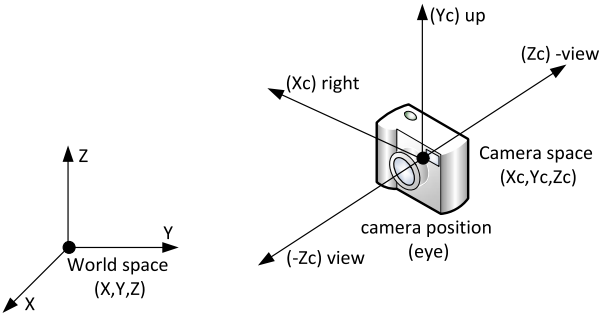

Figure 1: Virtual camera model (Virtual Image)

G. Extracting camera pose from view matrix

We found the camera position by:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} r_x & r_y & r_z \\ u_x & u_y & u_z \\ -v_x & -v_y & -v_z \end{bmatrix} \begin{bmatrix} A \\ B \\ C \end{bmatrix} \quad (19)$$

Next, we obtained the geographic coordinates for the camera by $(lon_c, lat_c, alt_c) = XYZ2LLA(X_c, Y_c, Z_c)$, to find the azimuth and elevation angles. We searched for a point (X_v, Y_v, Z_v) placed at a distance $D = 0.00001$ from the camera position toward the view vector:

$$\begin{bmatrix} X_v \\ Y_v \\ Z_v \end{bmatrix} = \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} + D \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \quad (20)$$

Then we obtained geographic coordinates by $(lon_v, lat_v, alt_v) = XYZ2LLA(X_v, Y_v, Z_v)$. Finally, we searched for the angles using the function $(azimuth, elevation) = AziEleAngles(lon_c, lat_c, alt_c, lon_v, lat_v, alt_v)$.

Now, to calculate roll angle, we searched for a vertical point placed at a distance *100* meters from the camera position using $(X_{vert}, Y_{vert}, Z_{vert}) = LLA2XYZ(lon_c, lat_c, alt_c + 100)$. Then we projected this point on a plane defined by the camera position as the center and the view vector v as a normal vector to obtain a new point $g(X_g, Y_g, Z_g)$. Next we defined a new vector G from the camera position to this point $G(X_g - X_c, Y_g - Y_c, Z_g - Z_c)$. Finally, we searched for a roll angle by calculating the angle between G and $u(u_x, u_y, u_z)$ vectors.

H. 3D terrain engine

We defined a 3D terrain engine in this work by a 3D software model or representation of the Earth built using OpenGL library (see Figure 2). It provides the user with the ability to freely move around in the virtual environment by changing the viewing angle and position and could capture current virtual image and depth map in real-time.

We presented the earth as a unit sphere covered by a 3D mesh generated from a DEM (Digital Elevation Model) and textured by satellite images.

We can extract current OpenGL model-view, projection and view-port matrices using the following OpenGL commands:
glGetDoublev(GL_MODELVIEW_MATRIX,modelview)
glGetDoublev(GL_PROJECTION_MATRIX,projection)
glGetIntegerv(GL_VIEWPORT,viewport), and we can change the camera view using the command gluLookAt(ex,ey,ez,vx,vy,vz,ux,uy,uz). We can also capture current virtual image and depth map using the commands glReadPixels(0,0,w,h,GL_BGRA_EXT,GL_UNSIGNED_BYTE,virtual_image)
glReadPixels(0,0,w,h,GL_DEPTH_COMPONENT,GL_FLOAT,depth_map). Finally, we can find a 3D point (X, Y, Z) which have a 2D projection (x, y) on the virtual image and winZ as the depth using the following command gluUnProject(x,y,winZ,modelview,projection,viewport,X,Y,Z).

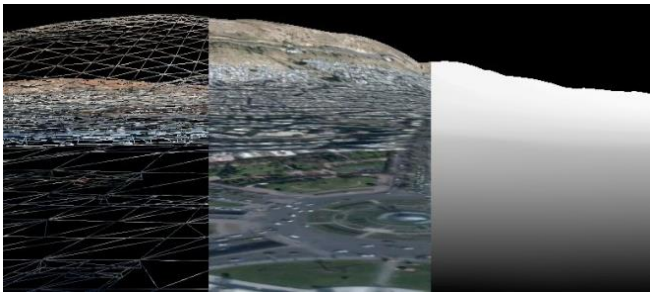

Figure 2: 3D Terrain Engine (Virtual image and depth map)

I. Camera calibration

Before using our system, we must calibrate the camera to find intrinsic parameters; hence, we used the calibration method described in [8] to simultaneously estimate the focal length and principal point parameters. Standard values were assumed for the remaining intrinsic parameters (i.e. zero skew, zero radial distortion and unit aspect ratio), or we can use Matlab calibration toolbox [9].

V. PROPOSED APPROACH

The 3D Terrain Engine enabled us to get multi-views at real time to implement the proposed computer vision algorithms. Now, we will explain some processes before describing the approach steps, such as feature extraction and pairwise feature matching.

A. Feature Extraction

We evaluated many local features to find correspondences between real and virtual images, and we found that SIFT key-point detector and descriptor [2] is very suitable because it is invariant to scale and rotation, and it is partially invariant to viewpoint and illumination changes [10]. It has also been found to be highly distinctive and repeatable in performance evaluation [11], although SIFT needs more time to calculate on CPU (Central Processing Unit) compared with the other methods. Therefore, we implemented it on a GPU (Graphics Processor Unit) to represent point features in real-time.

B. Pairwise Feature Matching

We evaluated many pairwise feature matching algorithms implemented on a CPU, and we found that the linear search algorithm is the best one in accuracy. However, it is the worst in performance, when we used a large number of key-points [12,13]. To solve this problem, we used the sufficient number of key-points (in our work, we just need 200 key-points) and we implemented the algorithm on GPU to work in real-time.

C. Approach steps

We resume the proposed approach by the following steps (see Figure 3) taking in consideration the co-planarity of the 3D points (Because when dealing with planes, there are doubt about the pose because several poses that are very different have the same perspective projection [14]).

- Calibrate the camera to find its internal parameters, then calculate its horizontal field of view FOV_h from its focal length f and the image width in Pixels w using the following equation $FOV_h = 2\arctan(w/2f)$ then change *FOV* for the 3D terrain engine using FOV_h value.
- Estimate initial pose *(longitude, latitude, altitude)* and *(azimuth, elevation, roll)* from GPS/IMU.
- Move and rotate the virtual camera according to initial pose.
- Capture the real image from the mounted camera and the virtual image from the 3D terrain engine. Next, find best correspondences between the two images automatically using SIFT and pairwise matching algorithms, after filtering them by sorting matches distances in ascending order. Then, take the first *N=30* points that spread in the whole image (we reject points that are near taken points).
- Estimate the robust Homography matrix *H* from the correspondences by rejecting the outliers using RANSAC (RANdom SAmple Consensus) [15]. Apply the Homography matrix to reject the corresponding points that have a projection distance more than a predefined threshold (2 pixels).
- Get the 3D coordinates of the accepted correspondences from the 3D terrain engine.
- Test if these points are coplanar. If coplanar, then rotate the virtual camera multiple times to obtain new multi-views. Next, apply coplanar calibration algorithm with multiple views to estimate rotation matrix and translation vector [16]. If it is non-coplanar, apply the non-coplanar calibration algorithm with one view [6].
- Use the estimated rotation matrix and the translation vector to find view matrix *M*. Then, extract the global camera pose using the equations shown in section *Extracting camera pose from view matrix*.

VI. Testing Environment

We created a new testing environment *MapViewer.exe* using C++Builder XE5 (see Figure 4). This testing environment consists of 2D top-view satellite images, 3D terrain engine (virtual camera), 3D path planner and image database.

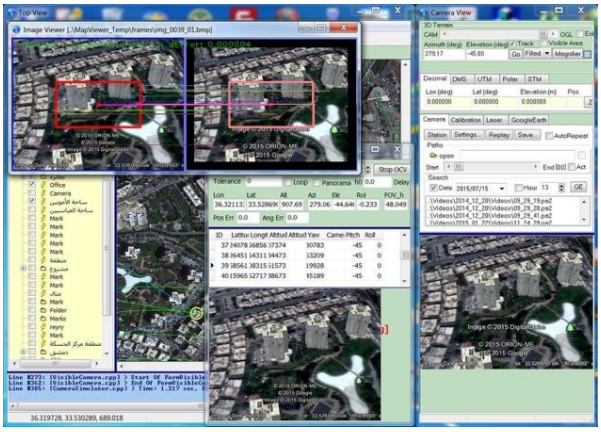

Figure 4: Our Testing Environment (MapViewer)

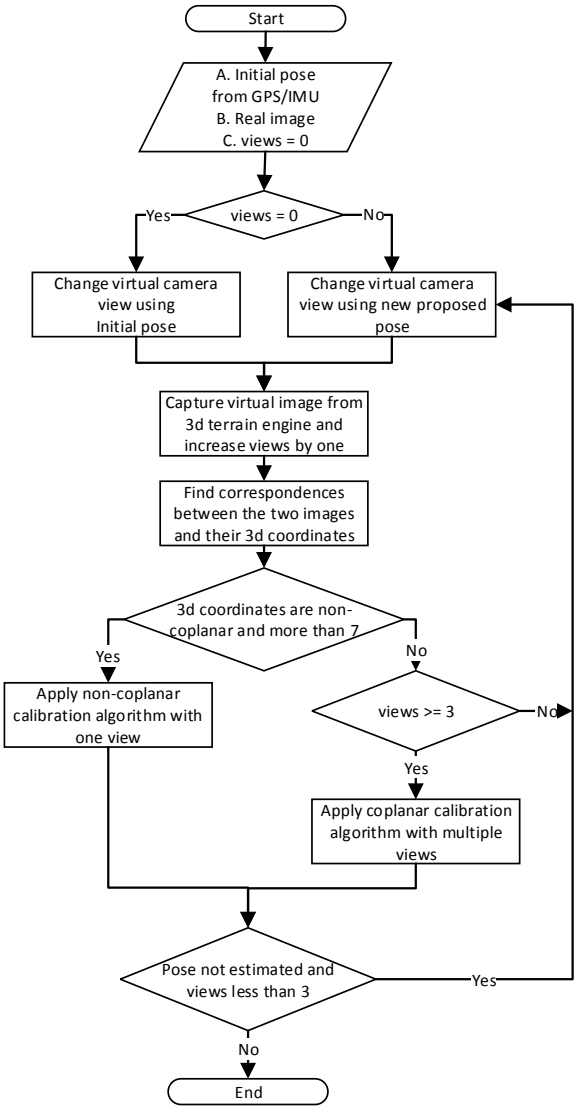

Figure 3: Our pose estimation approach

A. 3D path planning

We created a set of tools that let us to design a UAV path in the 3D environment to test our approach in different scenarios like those in the real world (see Figure 5). These paths were generated with UAV speed (*150 km/h*), elevation speed (*1 m/s*). We also made sure that the path did not intersect the terrain and each point has a line of sight with the ground station.

VII. EXPERIMENTAL RESULTS

We tested our approach on synthetic and real images with dimensions of 384×288 pixels on a PC with *Intel CPU i7 4-cores, NVIDIA GeForce GTX 570 graphics card, 4GB RAM* and *Windows7 64-bits operating system*, the frame rate was *12 FPS*.

A. Synthetic test

We used the testing environment to generate new scenarios for the UAV paths, without adding any noise to use them as a reference (see Figure 5). Then we tested the proposed approach in two cases, Ideal and Noisy.

i. Ideal case

In the ideal case, we tested the approach without adding any noise (see Figure 6). The errors after applying our pose estimation algorithm are shown in Table 1.

Figure 5: 3D path generated by our UAV path planner

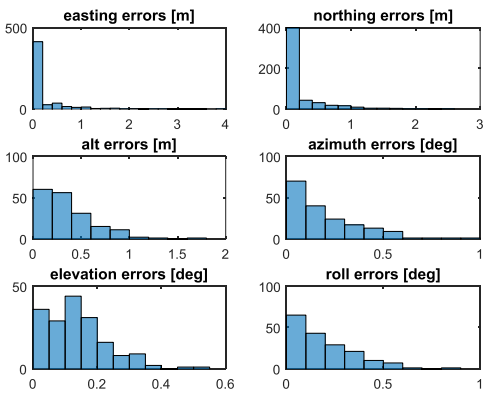

Figure 6: Estimated POSE parameters errors (Ideal Case), where horizontal axis shows errors and the vertical axis shows occurrence count

Table 1
Ideal case errors

Easting [m]	Northing [m]	Altitude [m]
0.2	0.2	0.5
Azimuth [deg]	Elevation [deg]	Roll [deg]
0.3	0.2	0.3

ii. Noisy case

In the noisy case, we added a Gaussian noise with *mean=0.0* and *stddev=10* meters for position and *stddev=10* degrees for angles (see Figure 7 and Figure 8). The errors after applying our pose estimation algorithm are shown in Table 2.

B. Real test

We tested our approach on real images taken from a UAV. Each image has inaccurate pose, which we used as initial pose to our algorithm. Because we did not know the accurate pose for those images, (unexpected sensor noise and weather conditions) we could not compare our results with them like in ideal case. Therefore, we validated our results by making a comparison between two images: the real and the corrected virtual image (virtual image after changing 3D engine using estimated pose). The re-projection error was so small (*2* pixels) and acceptable for Navigation or Target Localization applications (see Figure 9). We could see that the real image coincides with the 3D Terrain Engine Image, so we could get the geographic coordinates for any pixel in the real image. There were cases where we could not find good correspondences between the real and virtual image, especially when there was a long period between the time of capturing of the two images. We could solve this problem by updating the 3D terrain engine periodically. In the worst case, we could fly the UAV to capture images from the work area before the real mission. We could also generate an Ortho-photo (for example, using Agisoft PhotoScan) to update the 3D terrain engine (if we use Google Earth, we can do that using KML files).

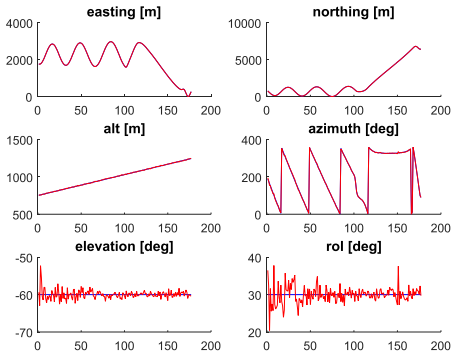

Figure 7: Estimated POSE parameters (Noisy Case), where horizontal axis shows frame index and the vertical axis shows values

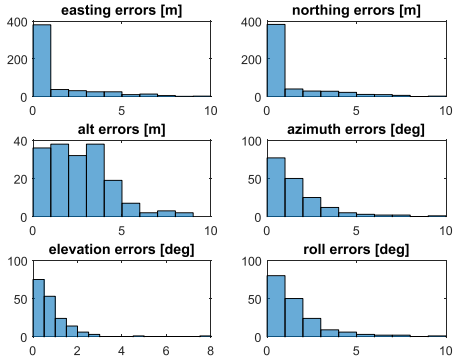

Figure 8: Estimated POSE parameters errors (Noisy Case), where horizontal axis shows errors and the vertical axis shows occurrence count

Table 2
Noisy case errors

Easting [m]	Northing [m]	Altitude [m]
1.0	1.0	4.0
Azimuth [deg]	Elevation [deg]	Roll [deg]
2.0	1.0	2.0

VIII. CONCLUSION

This paper presented a new approach to automatically estimate in real-time a global pose for a UAV using 3D terrain engine. Our system works well when the UAV flies at a suitable altitude This depends on the 3D terrain engine used, for example when we used a 3D terrain engine built on a DEM, we must fly with high altitudes to compensate the insufficient 3D model accuracy. However, when we used a 3D City Model as engine, this constraint is not needed. Also, our approach is near real-time (12 FPS) and can be used in navigation and multiple targets localization. In the future, we will use 3D City models to enhance the results and we will work on auto-calibration algorithms using 3D terrain engine to allow user to change the field of view on-line, without the need to re-calibrate off-line. Finally, we can say that the 3D terrain engine succeeded when other methods failed.

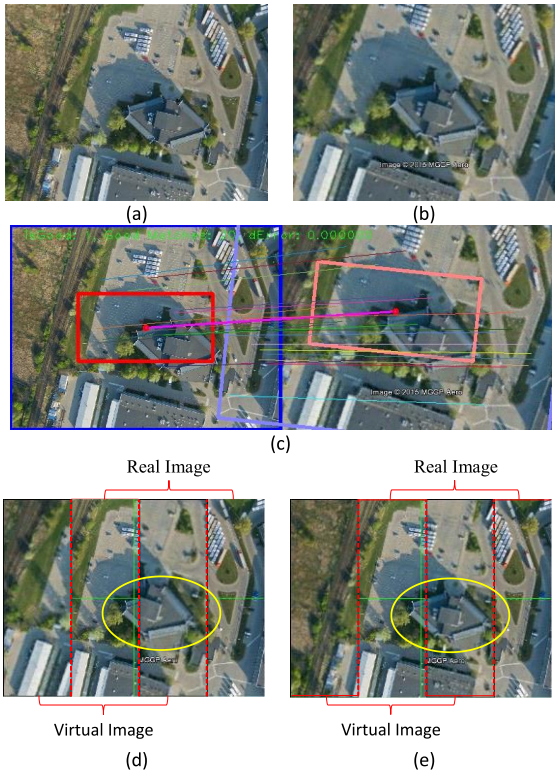

Figure 9: Real test: (a) Real image, (b) Virtual image, (c) matching between images (a-b), (d) Real image (even slides) and Virtual image (odd slides) before applying our algorithm, (e) Real image (even slides) and Virtual image (odd slides) after the algorithm

REFERENCES

[1] Son K.-H., Hwang Y., and Kweon I. S., "Uav global pose estimation by matching forward-looking aerial images with satellite images," *in Intelligent Robots and Systems*, 2009. IROS 2009. IEEE/RSJ International Conference on. IEEE, pp. 3880–3887, 2009.

[2] Lowe D. G., "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[3] Bay H., Ess A., Tuytelaars T., and Van Gool L., "Speeded-up robust features (surf) ," *Computer vision and image understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[4] Lim H., Sinha S. N., Cohen M. F., and Uyttendaele M., "Realtime image-based 6-dof localization in large-scale environments," *in IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2012)*, June 2012.

[5] Haralick B. M., Lee C.-N., Ottenberg K., and Nolle M., "Review and analysis of solutions of the three point perspective pose estimation problem," *International journal of computer vision*, vol. 13, no. 3, pp. 331–356, 1994.

[6] Lourakis M. and Zabulis X., "Model-based pose estimation for rigid objects," *in Computer Vision Systems. Springer*, pp. 83–92, 2013.

[7] Snyder J. P., "Map projections–A working manual," US Government Printing Office, pp. 1395, 1987.

[8] Irschara A., Kaufmann V., Klopschitz M., Bischof H., and Leberl F., Towards fully automatic photogrammetric reconstruction using digital images taken from UAVs. Na, 2010.

[9] Heikkila J. and Silven O., "A four-step camera calibration procedure with implicit image correction," *in Computer Vision and Pattern Recognition Proceedings, IEEE Computer Society Conference on. IEEE*, pp. 1106–1112, 1997.

[10] Lingua A., Marenchino D., and Nex F., "Performance analysis of the sift operator for automatic feature extraction and matching in photogrammetric applications," *Sensors*, vol. 9, no. 5, pp. 3745–3766, 2009.

[11] Mikolajczyk K., Tuytelaars T., Schmid C., Zisserman A., Matas J., Schaffalitzky F., Kadir T., and Van Gool L., "A comparison of affine region detectors," *International journal of computer vision*, vol. 65, no. 1-2, pp. 43–72, 2005.

[12] Silpa-Anan C. and Hartley R., "Optimised kd-trees for fast image descriptor matching," *in Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pp. 1–8, 2008.

[13] Muja M. and Lowe D. G., "Fast approximate nearest neighbors with automatic algorithm configuration," *VISAPP*, vol. 2, no. 1, 2009.

[14] Petersen T., A comparison of 2d-3d pose estimation methods. Master's thesis, Aalborg University-Institute for Media Technology Computer Vision and Graphics, Lautrupvang, vol. 15, pp. 2750.

[15] Fischler M. A. and Bolles R. C., "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[16] Zhang Z., "A flexible new technique for camera calibration," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 11, pp. 1330–1334, 2000.