Design Flow of Single Camera Motion Estimation Using GPU Accelerators

Guojun Yang and Jafar Saniie

Embedded Computing and Signal Processing (ECASP) Research Laboratory (http://ecasp.ece.iit.edu)

Department of Electrical and Computer Engineering

Illinois Institute of Technology, Chicago IL, U.S.A.

***Abstract*—Autonomous navigation and localization remains a challenging task for a moving object. Computer-vision based solutions have been proven effective when estimating the location of a moving object from an on-board camera using feature points. These feature points can be created in the image where the gradients change in all directions and offer unique values. By observing the position changes of feature points, the motion of the camera can be determined. As a result, feature detection and matching play critical role when estimating the motion of camera using video. In this paper, a GPU-accelerated feature detecting, and matching motion estimation system is introduced and tested using a drone platform. During our test, the GPU-accelerated system showed a 4-times increase in speed when compared with CPU based approach.**

***Keywords*—SLAM; GPU acceleration; CUDA**

I. Introduction

SLAM (Simultaneous localization and mapping) refers to the process localizing an object itself in an unknown environment and creating a map of the environment at the same time. Camera motion estimation is an important step in vision-based SLAM applications. In those environments where GPS signals are unavailable, SLAM is one of the few alternative solutions toward localization problems [1][2][3]. SLAM can be achieved using ultrasonic sensor, lidar and camera. Camera is the most adaptive sensor among those SLAM sensors. To be more specific, each ultrasonic sensor can generate one distance measurement; even with an array of ultrasonic sensors, the resolution of their measurements is inferior. A typical Lidar sensor makes measurements by scanning an environment using multiple laser emitters. However, Lidar sensors are expensive and bulky in size; they also require huge throughput for data processing. In comparison, cameras can measure distance of objects in their field of view. Cameras are widely accessible, affordable, and most of all, more versatile, with the help of machine learning techniques, objects can be detected and annotated in camera footages.

To estimate the motion of a moving object, it requires a system to extract 3D information. However, extracting 3D information from a single regular camera is challenging. Stereo vision cameras are able to calculate depth information using the offset of an object in a pair of images. To estimate the motion of a single regular camera, it requires a camera to compare the changes in the scene [4][5][6][7].

This paper focuses on using GPU to accelerate the motion estimation algorithm when using video clips taken by a regular camera. There are a number of related research works that use single camera to extract 3D information or estimate the motions. In Noah Snavely's research [5], a point cloud of a scene can be built using isolated photos taken by different cameras. Researchers from Microsoft [8] uses videos shot on regular camera to estimate the trajectory of camera to stabilize the time-lapse video.

To evaluate the performance of the system, we implemented the motion estimation algorithm using both CPU and GPU. Then video footages collected from a drone were used to benchmark both CPU and GPU systems. The performance of the algorithm will be compared and discussed in the result section.

II. Estimating Motion Using Video

Estimating the position and attitude of an object is critical for navigation and localization applications. Using GPS is the most common solution to the positioning problem. The GPS service is able to determine the position of an object in meters. However, GPS readings are prone to delays and refresh slowly, both of which compromise the confidence in localization while navigating a moving object. To compensate that problem and retrieve up-to-date estimation on location and attitude of an object, accelerometers and gyroscopes are often adopted. Between each GPS update, a system can calculate its changes in position and attitude using inertial components. Using GPS combined with inertial components has one major defect; these devices perform poorly indoor. For indoor navigation applications, GPS signals are not available. Accelerometers and gyroscopes can theoretically estimate the position and attitude of an object, however, error in such estimation will accumulate, unless the attitude and position can be rectified with GPS and magnetometer. Therefore, for a localization system to operate indoor, other techniques need to be adopted. There are other techniques proposed to solve the indoor navigation problems. Those solutions rely on either beacon or designated reference objects (e.g., using ultrasounds or Wi-Fi).

Computer vision-based solution is the most promising for indoor localization and mapping. When humans navigating indoor, we mostly relying on visual perceptions. Hence, it is possible to use visual information alone to navigate indoor. Moreover, cameras are widely accessible, and their performance are only getting better. Moreover, if we were able to extract motion information from videos, there is tremendous data readily available. That large amount of data opens great possibilities for creating 3D model of scene, people and etc. Most importantly, using the vision-based method to estimate motion requires no modification to its environment, which allows the system to be adaptive and operate with minimum limitations.

A. *Background for Estimating Motion from Video*

By comparing the corresponding feature points between frames, the motion of the camera can be estimated. In this section such processes will be explored. Every time a camera takes a photo of a scene, 3D information is projected to a 2D plane. Therefore, using only one image it's impossible to determine the motion or 3D information from the scene.

To recover the 3D information from images, two or multiple images on the same scene are required [9]. Motion estimation and mapping are the last stage of the whole process. Motion estimation and mapping is achieved by adopting multi-view geometry. Multi-view geometry helps to calculate the motion of a camera in terms of rotation and translation, which is represented by the extrinsic matrix of the camera for each frame. As shown in Figure 1, there are 3 different photos taken by the same camera at different locations t_1, t_2 and t_3 with different poses R_1, R_2 and R_3. X is a keypoint in real world, which is projected on photos taken at locations 1 to 3. Each projection is shown on camera planes as X_1, X_2 and X_3.

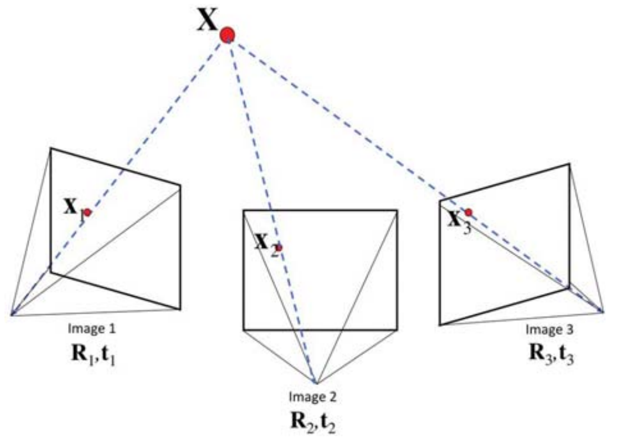

Figure 1. Illustration of multi-view geometry

Once 6 pair of matched feature points have been found, the fundamental matrix can be estimated using Equation 1. x_1 and x_2 are the pair of matched feature points in image plane. While F is the fundamental matrix to be estimated. To find a solution of F, multiple correspondence are required to minimize the error of the fundamental constrain, as shown in Equation 2 [10].

$$x_1^T F x_2 = 0 \quad (1)$$

$$\min \sum_j ||x_1^j - F x_2^j||^2 \quad (2)$$

Once the fundamental matrix that generates minimum error is found, essential matrix E can be calculated as:

$$E = K_1^T F K_2 \quad (3)$$

Essential matrix can be decomposed to translation of rotation of the camera. By repeating above process, the motion of the camera can be recovered.

III. Matching Detected Features

A. *Feature Detector*

Feature detecting is the prerequisite for finding correspondence between frames. In principle, this is achieved by finding unique gradient patterns and matching them among multiple frames. An example of such gradient patterns can be corners. As shown in Figure 2, feature x_i is detected in original image (shown as yellow line) using kernel ω (black circle). Its corresponding feature located in the second image (shown as red). Between the original image and the second image, the shift of feature x_i is described by vector u (shown as red arrow). As shown in Figure 2 (b), when using edge-like features, its corresponding feature in candidate frame remain ambiguous. As shown in Figure 2 (c), while using texture-less areas as features, the ambiguity increases further. Therefore, edges and texture-less areas are not suitable features for matching. Figure 2 (a) shows the matching of corner-like features. Corner-like features generate gradient change in both horizontal and vertical directions. Such changes in gradient provide rigid constrain when performing matching.

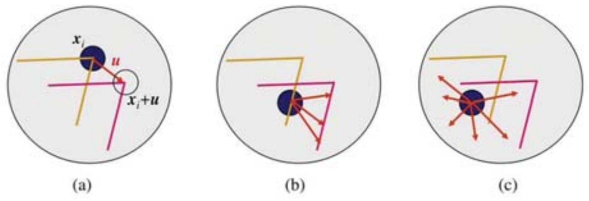

Figure 2. Corner, line and solid gradient

However, corner-like features are rotation variant. As the camera rotates, the correspondence of corners may no longer be established. Besides rotation, the scale of object also changes while the camera is moving. To make a feature detector invariant to scale changes and rotation, it needs to generate the feature on multiple scales and create a descriptor describing the direction of each feature point. SIFT (Scale-Invariant Feature Transform) feature detector as an example, detects feature points on multiple scales, and records gradient changes in all directions. As shown in Figure 3, once a feature point is detected, direction of gradient within an 8 by 8 square will be calculated (shown on left). Those gradients will then be organized as histograms for future matching (shown on right).

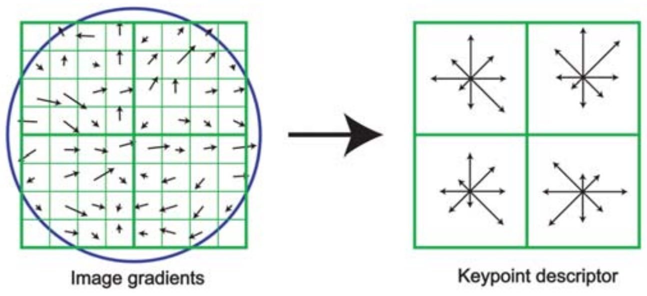

Figure 3. Feature descriptor of a keypoint [11]

B. *Feature Matching*

The correspondence of feature points is established through the feature matching. As mentioned in above, each feature point associates to a descriptor. The hamming distance of the same feature point in different frames should remain small. Hence, by calculating the hamming distances between all candidate feature points, correspondence can be found.

In a typical SLAM application, to estimate the motion of the camera, feature points extracted from a new frame need to be compared with their counterparts in the previous frame. By finding the motion of the camera in two frames repeatedly, the motion of the camera can be recovered. By recovering the motion of the camera incrementally, the trajectory of a vehicle can be determined.

In theory, to recover the 5 DoF (Dimension of Freedom) motions of the camera, at least 6 pairs of matched feature points are needed. However, there are outliner (e.g. wrong matches) among all matched feature points. As a result, merely 6 pairs of feature points would not suffice SLAM applications. Moreover, the process of matching feature point has complexity of $O(n^2)$, hence, the computation time for feature matching will grow exponentially as the number of feature points grow linearly.

IV. GPU Acceleration

Unlike CPU, GPU are designed for graphical processing. As a result, GPU has different architecture than CPU. In general, GPUs are often equipped with multiple cores. Each core can perform simpler tasks compared to CPU. However, by utilizing multiple cores simultaneously, a GPU can provide better throughput when performing certain tasks. As shown in Figure 4, a CPU is more powerful, while GPU has more cores.

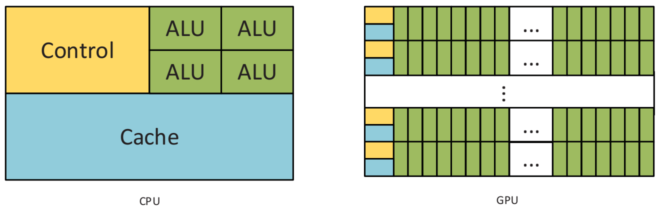

Figure 4. CPU and GPU architecture comparison

The capability of each individual core on a GPU limits its capability to achieve single yet complex problems. While performing parallelizable yet simple tasks, the GPU can always outperform CPU.

Through parallelization, the performance of certain tasks can be enhanced drastically. Using matrix multiplication as an example, when multiplying 2 100-by-100 matrix m and n with floating point numbers, the result matrix will be a 100-by-100 matrix k. Each element in the product matrix k is the result of 100 multiplication and 99 summation operations. Through a single-core CPU, those operations will be processes sequentially. However, with hundreds of cores, a GPU can process hundreds multiplication operations and summation operations simultaneously. Ideally, by paralleling matrix multiplication task, the time used in computation can be cut down by the number of cores in a GPU. When performing motion estimation, tasks such as detecting and matching feature points can be easily parallelized. By leveraging the multi-tread advantage of a GPU, those tasks can be carried out more quickly.

However, some tasks are not easy to be parallelized. Hence, not all tasks in motion estimation can be implemented on GPU. When using GPU and CPU working in coordination, data transition between the two devices cause delays. Thus, they need to be minimized. As shown in Figure 5, tasks will be split between CPU and GPU. By performing the feature detecting and matching on GPU, CPU can perform time-sensitive tasks with less delay, which benefits varies motion estimation applications.

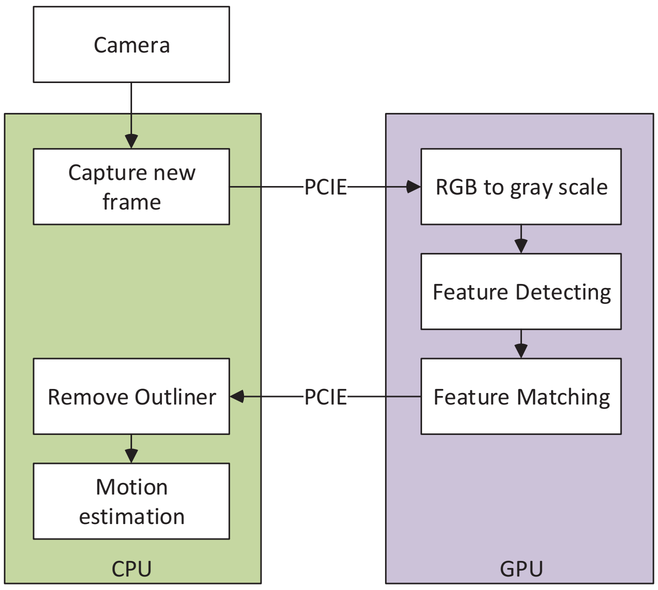

Figure 5. Workflow of motion estimation

V. Result

To test the effectiveness of GPU accelerated motion estimation, we built a drone survey system. In the survey system, a drone was used to collect aerial footage with a regular RGB camera. The footage then was transmitted to a desktop computer powered by a 6th generation i7 CPU and Nvidia GTX 1070 GPU. To compare the difference in performance, the motion estimation program was implemented using CPU [12] as a control group. CUDA library was used when implementing the GPU algorithm. To simplify the system, mapping of detected feature points and ICP (Iterative Closest Point) was ignored.

During the motion estimation process, no IMU (inertial measurement unit) or GPS was used. As a result, the motion of the camera on the drone was estimated up to scale. As shown in

Figure 6, the calculated trajectory shows the drone finished a round course in the sky.

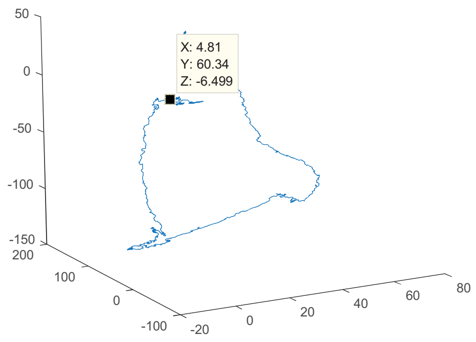

Figure 6. Estimated camera motion (scale)

As shown in Figure 7 and Figure 8, feature points are extracted and matched in the drone footage. These matched feature points are able to help the algorithm to estimate the motion of the drone and the camera mounted on it. Moreover, these two processes differentiate the performance of the CPU method and GPU method.

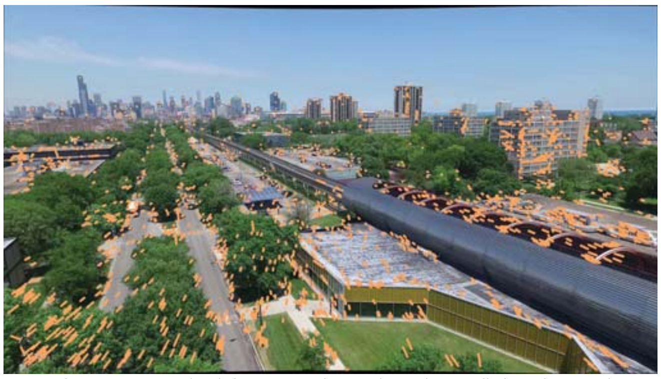

Figure 7. Matched feature points when drone flying forward

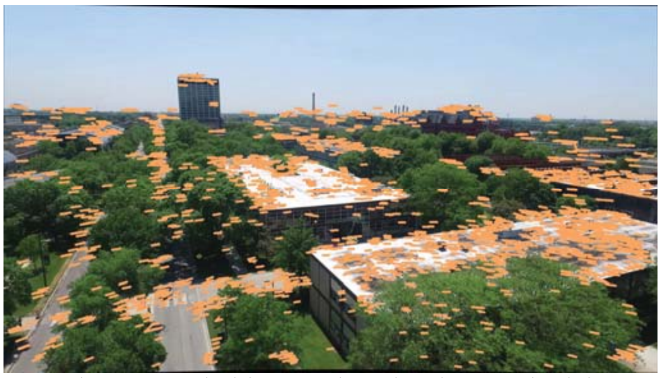

Figure 8. Matched feature points when drone turning left

All footages are shot at 1920 by 1080 pixels 60 frame per second. 4 different footages were used was to stress-test CPU and GPU implementation. Each has been processed with 100 loops. As shown in Table I**Error! Reference source not found.**, the average processing time of a single loop (in seconds) shows the GPU reduces the processing time of CPU to roughly 1/5.

TABLE I. PROCESSING TIME FROM THE STRESS-TESTS (SECONDS)

	Footage 1	**Footage 2**	**Footage 3**	**Footage 4**
Length	81.78	32.83	27.69	32.22
CPU	1548.040	881.291	472.211	372.661
GPU	310.597	136.596	108.926	121.305

VI. CONCLUSION

By performing feature detecting and matching on GPU, the motion estimation system was able to yield 4.98 times higher FPS (frame per second). Other parts of SLAM applications, such as ICP can be implemented on GPU to improve the performance further. Moreover, in this experiment we used a desktop computer, but portable platform such as Nvidia Jetson is also equipped with GPU, which makes it possible for building a real-time vision-based SLAM system on a portable platform.

REFERENCES

[1] B. Joydepp and V. Manuela, "WiFi Localization and Navigation for Autonomous Indoor Mobile Robots," in *IEEE International Conference on Robotics and Automation*, Anchorage, 2010.

[2] G. Yang and J. Saniie, "AR Marker Assisted Indoor Navigation for Visually Impaired," in *2017 IEEE International Conference on Electro/Information Technology*, Lincoln, 2017.

[3] B. Kim, M. Kwak, J. Lee and T. T. Kwan, "A multi-pronged approach for indoor positioning with WiFi, magnetic and cellular signals," in *2014 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, 2014.

[4] S. Y. Bao, M. Bagra and S. Savarese, "Semantic structure from motion with object and point interactions," *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops),* pp. 982-989, 2011.

[5] A. Saxena, M. Sun and A. Ng, "Make3D: Learning 3D Scene Structure from a Single Still Image," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 31, no. 5, pp. 824-840, 2009.

[6] N. Snavely, S. M. Seitz and R. Szeliski, "Photo Tourism: Exploring Photo Collections in 3D," *ACM Trans. Graph.,* vol. 25(3), pp. 835-846, 2006.

[7] H. Lim, J. Lim and H. J. Kim, "Real-Time 6-DOF Monocular Visual SLAM in a Large-Scale Environment," in *2014 IEEE International Conference on Robotics & Automation (ICRA)* , Hong Kong, 2014.

[8] J. Kopf, M. F. Cohen and R. Szeliski, "First-person Hyper-lapse Videos," *Proceedings of ACM SIGGRAPH 2014 ACM Transactions on Graphics (TOG),* vol. 33, no. 4, pp. 78:1-10, 2014.

[9] L. Quan and Z. Lan, "Linear N-point camera pose determination," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 21, no. 8, pp. 774-780, 1999.

[10] R. Hartley and A. Zisserman, Multiple view geometry in computer vision, Cambridge University Press, ISBN: 0521540518, 2004.

[11] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," vol. 60, no. 2, pp. 91-110, 2004.

[12] G. Yang, Z. Zhou, T. Gonnot and J. Saniie, "Design flow of motion based single camera 3D mapping," in *2015 IEEE International Conference on Electro/Information Technology (EIT)*, 2015.