A multithreaded algorithm of UAV visual localization based on a 3D model of environment: implementation with CUDA technology and CNN filtering of minor importance objects

Alexander Buyval, Mikhail Gavrilenkov
Department of computer science,
Bryansk State Technical University[1],7 Oktyabrya boulevard, Bryansk, 241035,Russian Federation

Evgeni Magid
Intelligent Robotic Systems Laboratory, Higher Institute for Information Technology and Information Systems (ITIS),
Kazan Federal University[2], 35 Kremlyovskaya street, Kazan, 420008, Russian Federation
E-mail: alexbuyval@gmail.com, gavrilenkov@umlab.ru, dr.e.magid@ieee.org
[1]http://iipo.tu-bryansk.ru[2] http://kpfu.ru/eng/itis

Abstract

Visual based navigation plays an important role in localization and path planning, especially in GPS-denied environments. This paper presents a visual based localization algorithm for a UAV within an indoor environment. The algorithm uses multithreaded computing CUDA technology and CNN-preprocessing filtering, which is responsible for filtering out dynamic objects. The algorithm is simulated in ROS/Gazebo environment with two different approaches – one uses CPU only and the other uses CPU and GPU - and their performance is compared.

Keywords: localization of UAV; particle filter; ROS; Gazebo, CUDA; CNN.

1. Introduction

Robot self-localization and simultaneous localization and mapping (SLAM)[1] play important role in path planning and navigation for multiple purposes, including such broad fields as service robotics, industrial robotics, search and rescue robotics, and others[2]. Within indoor environment, which are typically GPS-denied, visual localization is often a natural substitution of a GPS approach. Due to visual sensors' long range, high resolution, low energy consumption, and affordable price visual localization is often preferred over laser range finder or sonar based localization solutions. Selection of a passive visual sensor ranges from cheap and simple monocular cameras to stereo or depth cameras, like Kinect, or event-driven cameras. We may distinguish two typical subclasses of visual localization methods: the first subclass uses special visual markers with known positions and thus requires some additional marking of environment prior to its use; the second subclass uses native visual features.

This paper focuses on visual localization approach, which uses object edges as key features, and proposes several improvements to a particle filter based algorithm. The algorithm uses multithreaded CUDA technology and CNN-preprocessing filtering to exclude dynamic objects. We assume that while an initial 3D model of environment is available, the scene may undergo minor dynamical changes, e.g. new objects may appear in the scene as the time passes. The localization is performed in two steps. Initially, a neural filtering module detects new objects in the scene and filters them out. Next, a multithreaded edge-computing module processes filtered data and compares it with the initial model. The algorithm is simulated in ROS/Gazebo environment with two

different approaches – one uses CPU only and the other uses CPU and GPU – and their performance is compared.

2. Basic localization algorithm

To solve a localization problem our system uses image edges as visual features. Based on location model, it compares a received from a camera real image and a simulated image of a pre-stored 3D model. The algorithm considers a number of robot position hypotheses and renders a set of images from the 3D model that correspond these hypotheses. Each image corresponds to an image that would be obtained in the estimated point under a particular hypothesis. Thus, at each algorithm iteration, we compare one image from the camera to a set of simulated images, estimate edge matching between them and a level of their similarity.

Edge features are formed from typical elements of indoor environment: conjunctions of walls, ceiling and floor, window and door frames, etc. Such edges are rather steady features. They are steady in time and, as a rule, illumination changes do not influence their perception. Experimental work showed that it is enough to consider these basic elements, which in turn significantly simplifies a 3D model of the room.

2.1. *Using a particle filter for robot localization*

For processing of images and locomotion sensory data, which are followed by localization hypotheses, we use particle filter approach that has proven to perform well in similar tasks. This approach provides a reasonable set of location hypotheses and allows for nonlinear models of the system and sensory input.

2.2. *The particle filter measurement model*

To estimate each hypothesis probability (a particle), edges from the on-board camera images are compared with edges of synthetic images from the 3D model, applying the nearest edges method. This method estimates edges' similarity using a set of line normals that are constructed from synthetic images' edges[3-5]. Figure 1 presents the result of rendering a synthetic image of a room at the position, which corresponds to a particular hypothesis. Edge features, which have been found in the synthetic image and correspond to real world edges, are shown in green; blue lines depict edges of the real world camera image, projected onto the synthetic image; line

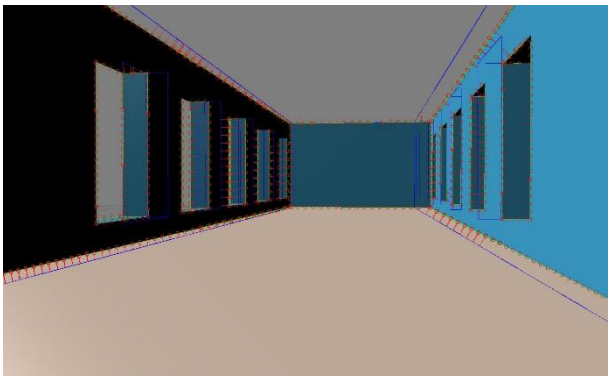

Fig. 1. Calculation of images similarity using edges.

normals are shown as red segments. The algorithm uses the extracted edges to perform the following calculations:

(i) *(Each) line weight* calculated as follows:

$$g(d) = exp\left(-\frac{d^2}{2\sigma^2}\right), \quad (1)$$

where d is the normal's length, σ is the parameter that determines the weight of the normal and depends on the normal length d. Parameter σ allows to scale (to increase or decrease) the influence of long normals.

(ii) *Total weight* is calculated as follows:

$$l = \frac{\sum_{i=0}^{S} g(d_i)}{S}, \quad (2)$$

where S is the total number of lines in the simulated image.

(iii) *Total probability hypotheses* is calculated by combining the weights of each line:

$$W = \propto \cdot exp\left(k\frac{\sum_{n=0}^{m} l_n}{m}\right), \quad (3)$$

where m is the count of lines, and $\propto, k$ are empirically established parameters.

3. Particle filter implementation

The most important particle filter task is updating the particle weights. At the first stages of the algorithm, we detect edges within an image and find straight lines. We implemented the system as a robot operating system (ROS[6]) package, and a number of computer vision functions from OpenCV library were utilized. The system uses a constant number of particles, which allows

static memory allocation and decreases memory management time.

3.1. *Comparative analysis of the CPU and GPU implementations*

Figure 2 presents a flowchart of particle weights calculation, which employs eq.(1-3). The main drawback of the algorithm is its complexity of *O(N*K*M)*, where *N* is a number of particles, *K* is a number of straight segments within a simulated image, *M* is a number of segment normals for a single straight line.

We parallelize the algorithm with NVIDIA's CUDA technology, which allows to receive a parallel computing system that works according to SIMT-principle (Single-Instruction, Multiple-Thread), when one instruction is simultaneously executed by multiple independent threads (Fig.3). We apply CUDA technology for particle weight

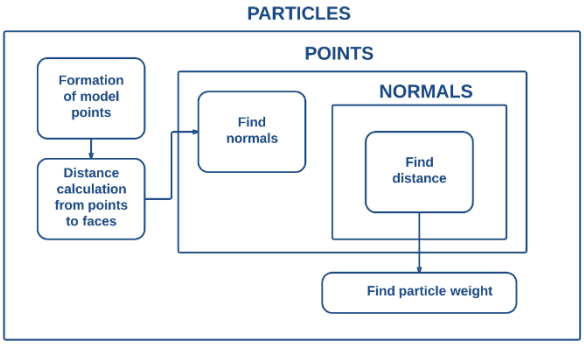

Fig. 2. The scheme of calculating the weights of the particles using a CPU only.

calculation through usage of a separate block for each particle and separate GPU core for each calculation of normal line's weight (Fig.4).

3.2. *Simulation results*

We implemented two versions of the system in ROS: one uses CPU only, while the other employs both CPU and GPU for calculations. Experiments were performed in ROS/Gazebo environment. Table 1 presents the two implementations' comparison within four different experiments: *N* denotes a number of the experiment, t_{CPU} and t_{GPU} denote calculation time in ms for CPU only system and CPU with GPU system implementation respectively, *ratio* denotes the ratio of t_{CPU} to t_{GPU}. The later system (CPU with GPU) demonstrated significantly

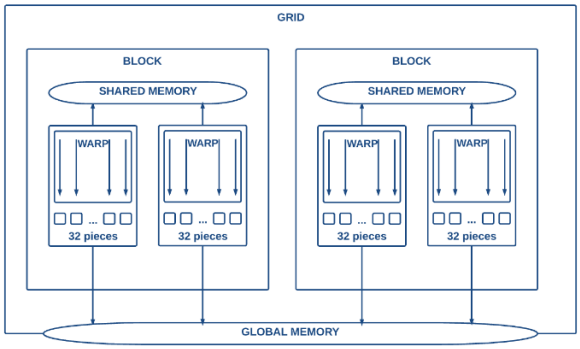

Fig. 3. CUDA threads hierarchy.

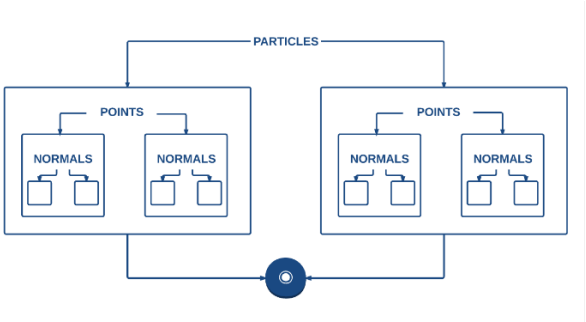

Fig. 4. The scheme of calculating the weights of the particles using a CPU together with GPU scheme.

better execution time, which surpassed the CPU-only system in almost 17 times in average.

Table 1. Performance comparison results, ms.

N	1	2	3	4
$\mathbf{t_{CPU}}$	123,6	102,83	150,12	131,84
$\mathbf{t_{GPU}}$	7,39	6,54	8,44	7,45
Ratio	16,7	15,7	17,7	17,6

4. Neural image filtering

The presented approach works well when a 3D model of environment is static and almost identical to the actual environment. However, in real world scenarios, most scenes are rather dynamic, i.e. new objects that were not captured during the 3D model building process appear and may even move through the actual environment (for example, the black sofa and the PC blocks in Fig.5 are new objects). To improve robot localization in such cases, all new and moving objects should be excluded from the scene analysis. Yet, if many temporary objects are excluded, too many edges (that belong to these objects)

may be excluded as well and there is a risk of obtaining a very small number of detected edges.

One of the most effective ways to find objects within an image is to employ a convolutional neural network (CNN). To initialize and train, these networks use image databases, which form a dictionary with each word corresponding to a graphical image[7]. The properly adjusted and trained CNN allows finding objects and excluding them from an image in real time. At the preliminary stage, the CNN preprocesses data prior to edge detection in order to eliminate potentially dynamic objects within the environment. Next, edge detection is performed for an "empty" room, followed by robot localization. Figure 5 demonstrates CNN image preprocessing results, where the black sofa (i.e., a "dynamic" object) was properly detected. CNN returned a two-dimensional mathematical description of an object,

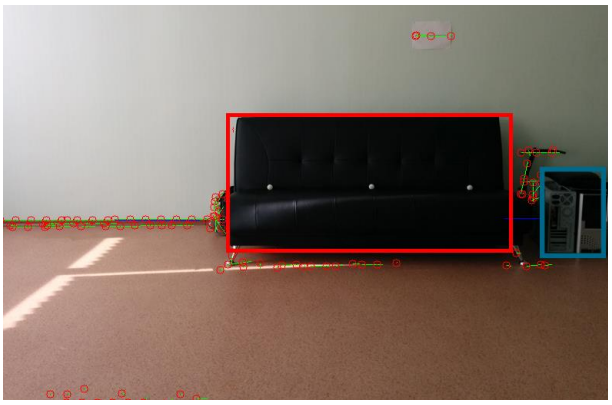

Fig. 5. Example of objects excluding by CNN: the black sofa (in red rectangular) and the PC blocks (in blue rectangular).

and the image area that belongs to the sofa object was eliminated from further processing.

Object detection and identification time is proportional to system hardware capabilities. After successful object detection, as a result of learning process, the CNN weights are redistributed. For all further appearances of the sofa object CNN performs quick identification in real time and thus localization process time decreases significantly.

Integrating such CNN module into a UAV onboard control system in addition to robot localization would allow a specific object detection, tracking a moving object[8] and other useful functionality.

5. Conclusions

This paper focuses on visual localization approach, which uses environment edges as key features. Our approach compares an obtained from a camera image with a simulated image of indoor environment's 3D model. We suggested a particle filter based algorithm improvement through CNN-preprocessing filtering, which eliminates dynamic objects from an image, and usage of GPU in UAVs localization system. Using parallel threads allowed increasing of localization system speed in almost 17 times in average. In addition, CNN filtering decreased negative influence of dynamic objects on localization process.

Acknowledgements

Part of the work was performed according to the Russian Government Program of Competitive Growth of Kazan Federal University.

References

1. A. Buyval, I. Afanasyev and E. Magid, Comparative analysis of ROS-based SLAM-related methods for autonomous indoor navigation, in *Int. Conf. on Machine Vision* (Nice, France, 2016)
2. E. Magid and T. Tsubouchi. Static Balance for Rescue Robot Navigation: Discretizing Rotational Motion within Random Step Environment, in *Lecture Notes in Artificial Intelligence*, 6472, (2010), pp. 423-435.
3. S. Nuske, J. Roberts and G. Wyeth, Outdoor visual localization in industrial building environments, in *IEEE Int. Conf. on Robotics and Automation*, (Pasadena, CA., 2008), pp. 544-550.
4. S. Nuske, J. Roberts and G. Wyeth, Robust outdoor visual localization using a three‐dimensional‐edge map, in *J. of Field Robotics* **26**(9), (2009),pp. 728-756.
5. A. Buyval and M. Gavrilenkov, Vision-based pose estimation for indoor navigation of unmanned micro aerial vehicle based on the 3D model of environment, in *IEEE Int. Conf. on Mechanical Engineering, Automation and Control Systems* (Tomsk, Russia, 2015).
6. E. Fernandez, S. Crespo, A. Mahtani and A. Martinez, Learning ROS for Robotics Programming Second Edition, (Birmingham, UK, 2015).
7. J. Redmon and S. D. Allen, You only look once: Unified, Real-Time Object Detection. arXiv preprint arXiv: 1506.02640v4 (2015).
8. G. Klein and D. Murray,Full-3D edge tracking with a particle filter, in *British Machine Vision Conference*,(UK, Edinburgh,2006).