,Implementation of Trapped Personnel Detection and Evacuation Guidance in Indoor Fire Scene Based on Quadrotor UAV

Danyu Bi
College of Electronics and Information Engineering
Tongji University
Shanghai, China
bidanyu@hotmail.com

Youling Yu
College of Electronics and Information Engineering
Tongji University
Shanghai, China

Yong Shen
College of Electronics and Information Engineering
Tongji University
Shanghai, China

Claudio Melchiorri
Dept. of Electrical, Electronic and Information Engineering
Univ. of Bologna
Italy

***Abstract*—This paper presents a solution to trapped personnel detection and evacuation guidance in indoor fire scene based on quadrotor UAV. Our approach purely relies on UAV's own monocular camera as the main sensor, and therefore UAV can search and rescue in unknown, GPS-denied fire environments. In this paper, an effective method was proposed, which separated the mission into two parts: one was trapped personnel detection by using face detection algorithm, the other one was evacuation guidance, which was actually scale-aware navigation and path planning by using visual SLAM system. Costly computations were carried out on an external laptop that communicated with the quadrotor over wireless. The solution can be used directly with a Parrot AR.Drone quadrotor.**

Keywords—Quadrotor UAV, Face detection, Navigation, visual SLAM, Search and Rescue.

I. INTRODUCTION

This Aerial robotics is a fast-growing field of robotics, and multi-rotor aircrafts are rapidly growing in popularity and now universally recognized for its applications in the commercial and military markets. The quadrotor, a unique type of Unmanned Aerial Vehicle (UAV), is an under-actuated system with four inputs and six outputs [1]. The advantages of the quadrotor UAVs lie in their Vertical Take Off and Landing (VTOL) ability, maneuverability due to their inherent dynamic nature, low-cost and small-size flying platform, simple structure and good stability [2].

In reality, it's really dangerous for human beings to do search-and-rescue works in the firing buildings which have the possibilities of collapse. Meanwhile the complicated unknown indoor environment greatly increases the difficulties for human search-and-rescue works. That is why robot is chosen instead. Compared with ground vehicles, the main advantage of flying devices is their extension from 2D plane into 3D space by changing their flight altitude. Among the flying devices, quadrotor UAVs are dominant over traditional helicopters for their fixed rotor propulsion mode. Therefore, quadrotor UAVs that can autonomously operate in indoor environments are envisioned to be useful for search and rescue activities.

The paper is structured in the following manner. In Sec. II we introduce our emergency services scenario. In Sec. III we consider the trapped personnel detection phase of the mission in more details and in Sec. IV, we consider the evacuation guidance phase of the mission in more details, while in Sec. V presents the integrated solution and practical experiments and results. Sec. VI concludes with final remarks and plans for future activity.

II. AN EMERGENCY SERVICE SCENARIO

A particular situation is assumed: There are thousands of people in one shopping mall on weekends, and there are numerous rooms in this building, such as toilets and offices. Suddenly, somewhere in the building catch fire and people inside is trapped and being more and more scared. It is really difficult for rescuers to search all the rooms and rescue trapped people as soon as possible. Therefore, the quadrotor UAVs can provide essential supports and greatly increase the working efficiency of human rescue activities.

The emergency service mission of quadrotor UAV is shown as *Fig. 1.*

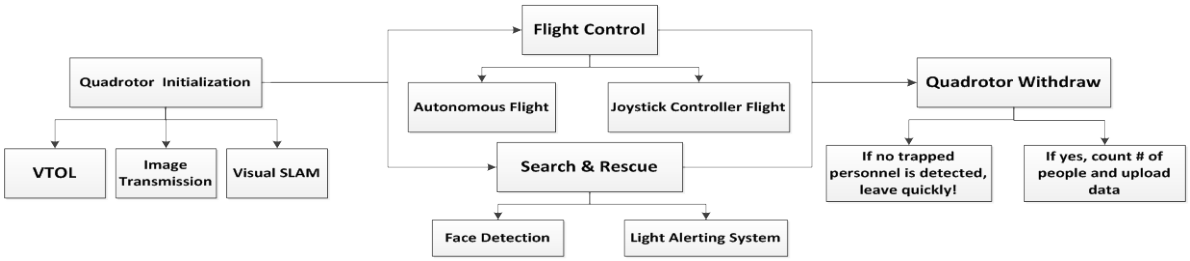

Fig. 1. Flow chart of emergency service mission

A. *Quadrotor Initialization*

Once the Fire alarm is on, the quadrotor UAV which is already equipped in the building will take off. In the first 2-3 seconds, the UAV will initialize itself quickly, including parameters initialization, image transmission, vertical takeoff & hovering and locate itself in unknown environment (Assuming the room is messy because of the fire).

B. *Flight Control*

Once the initialization work is done, the UAV will execute a flight plan automatically by stored mission or manually by remote joystick controller. Meantime, the UAV will do face detection work to detect if there are any trapped people in the fire scene. If the quadrotor UAV finds anyone, it will start light alerting system immediately.

C. *Quadrotor Withdraw*

After the quadrotor UAV finishes the detecting task, it will report both the position and face portrait of each trapped person, and finally count and upload the total number of them remotely by WIFI. When multiple quadrotor UAVs search in multiple rooms simultaneously, the feedback information they provide will help rescuers to make an excellent plan in a short time. And then quadrotor UAV itself will get away from the fire scene as soon as possible.

III. TRAPPED PERSONNEL DETECTION

A. *Face Detection*

The face detector uses a cascade of boosted classifiers working with Haar-like features, which has been initially proposed by Paul Viola [3] and improved by Rainer Lienhart [4]. The cascade classifier can be initialized with different XML files that define the object we want to detect. These files were created by training machine learning algorithms on hundreds or even thousands of images that either contain a face or do not. The learning algorithm is then able to extract the features that characterize faces and the results are stored in XML format. After a classifier is trained, it can be applied to search for the face in the whole image, which can move the search window across the image and check every location using the classifier. The classifier can be easily resized in order to find the objects of interest at different sizes, which is more efficient than resizing the image itself. The word "cascade" in the classifier name means that the resultant classifier consists of several simpler classifiers (stages) that are applied subsequently to a region of interest until at some stage the candidate is rejected or all the stages are passed. The word "boosted" means that the classifiers at every stage of the cascade are complex themselves and they are built out of basic classifiers using one of four different boosting techniques (weighted voting). Currently Discrete Adaboost, Real Adaboost, Gentle Adaboost and Logitboost are supported. The basic classifiers are decision-tree classifiers with at least 2 leaves. Haar-like features are the input to the basic classifiers, and are calculated as *Fig. 2*. The current algorithm uses the following Haar-like features, which is specified by its shape, position with the region of interest and the scale [5].

B. *Solution*

In our paper, the node used in trapped personnel detection is located in the file face_detector.py. The raw images are mainly subscribed from the appropriate UAV's video driver. And the number of detected faces is published to other nodes. The flow chart is shown as *Fig. 3*. In the face detection task, the front and profile face of trapped people will be detected by using two different XML files to initialize the cascade classifier. In addition, the cascade classifiers require some initialized parameters to set their speed and the probability of correctly detecting a target. In our solution, the minSize and maxSize parameters set the smallest and largest faces that will be accepted. The ScaleFactor parameter acts as a multiplier to change the image size as the detector runs from one scale to the next. The smaller this number, the finer the scale pyramid used to scan for faces but the longer it will take on each frame. After the raw images are subscribed by our node, they are converted into grayscale version, which the algorithms of Haar cascade detector run on. Then the histogram of the gray-scale image is equalized, which is a standard technique for reducing the effects of changes in overall lighting.

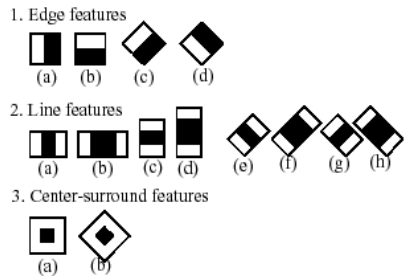

Fig. 2. Haar-like features.

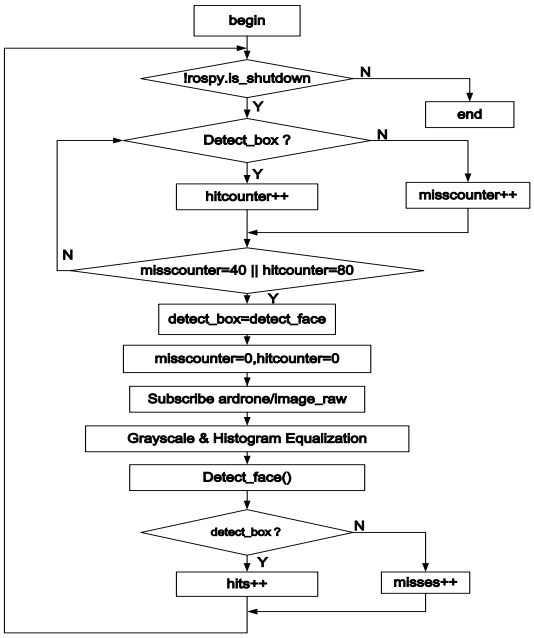

Fig. 3. Flow chart of face_detector.py node

The pre-processed image is sent to the detect_face() function which will be described later. If a human face is detected in this image frame, the variable self.detect_box will be set to 1, otherwise will be set to 0. The variable hits will return the total number of detected faces. Although ScaleFactor parameter is set to a suitable value, the accumulated delay from both image transmission and face detector calculation will be considerable after several frames. Therefore, we propose a method to solve this problem. Two new variables are introduced, one is misscounter_limit which means the number to count when detect no face, and the other one is hitcounter_limit which means the number to count when detect a face. We now don't detect every frame of raw images, but according to whether detecting a face or not, we skip hitcounter_limit or misscounter_limit frames and then detect again.

C. Experiments and Results

In our paper, Robot Operating System (ROS) is used on Ubuntu 14.04, which is an open-source, meta-operating system for robot software development. ROS provides standard operating system services and the primary goal of ROS is to support code reuse in robotics research and development [6]. ROS distributes computation in form of "nodes", where nodes exchange information through "topics". "Topic" is a name of data stream and data information is defined by "message". Implementation of "nodes" could be done in C++ or Python. OpenCV is a library of programming functions mainly aimed at real-time computer vision for 2D image processing and machine learning, which is one of the three pillars of computer vision in the ROS community. ROS passes around images in its own sensor_msgs/Image message format, but many users want to use images in conjunction with OpenCV. CvBridge in *Fig. 4* is a ROS library that provides an interface between ROS and OpenCV [7].

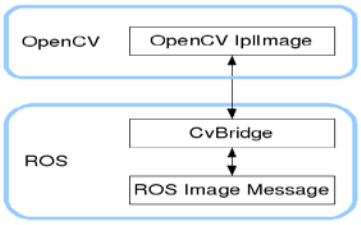

Fig. 4. Interface OpenCV with ROS

In order to ensure the real-time of our face detector, relative small time internal between the detection of two separate frames named miscounter_limit is necessary to be chosen. However, the smaller the miscounter_limit is, the more considerable the accumulated delay will be. Therefore, we need to find out the most suitable value. The experiment is as follows: Our node is set to run detect_face() function every N frames, and detects no face for totally 30 times. At the same time, another node is set to run without doing any face detection. Then we will record how many frames have been run at last in these two situations. The error between them means the accumulated delay is caused by N frames, which equals to miscounter_limit, and N is tested from 20 to 90.

According to the real-time requirement and the results of experiments in *Fig. 5*, miscounter_limit is set to N = 40. As to the setting of the other parameter, hitcounter_limit, we can measure the relationship between its value and the time internal, which means how long the detect_face() function will run again after detecting one face. The results are as Table1 shows. The hitcounter_limit is set to N = 80. *Fig. 6* shows the experimental video's screenshots. Once the node face_detector.py find a face, it will draw a rectangle to show the detected face, and keep it for a few seconds. At the meantime, a topic named hitscount will be published to other nodes.

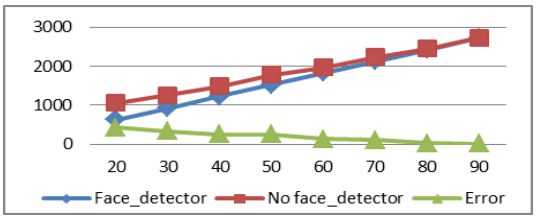

Fig. 5. Face detector for 30 times

TABLE I. PARAMETER HITCOUNTER_LIMIT

hitcounter_limit	Time Internal(s)
50	1.7
60	2
70	2.3
80	2.6
90	3.1
100	3.3
120	7

Fig. 6. Video's screenshot when detecting face

IV. TRAPPED PERSONNEL DETECTION

A. Flight Path Planning

According to the emergency service mission we mentioned in section 2, once the fire alarm is on, our quadrotor UAV will take off and detect trapped people autonomously inside the fire scene. Therefore it's really important to set reasonable flight path planning, both for detection and evacuation guidance work. In the fire scene, we assume the environment need to be detected is tiny and clutter. The quadrotor rotating itself by 360 degrees at the original point is the easiest method to search the whole room in a short time. However it will be dangerous for quadrotor to hover at one point for a long time, since goods inside firing room may fall down and smash thes quadrotor. Therefore, the flight plan of autonomous quadrotor will be a rectangle in a small scope, as shown in *Fig. 7*.

B. Navigation System

Navigation & path planning require both a solution to the so-called simultaneous localization and mapping (SLAM) problem as well as robust state estimation and control methods. For solving the SLAM problem on quadrotor UAVs, various kinds of sensors such as laser range scanners [8], monocular cameras [9], stereo cameras [10], and RGB-D sensors [11] have been explored in the past. In our paper, we prefer to choose monocular camera as onboard sensor, because they provide rich information at a low weight, power consumption, size and cost [12]. According to Jakob Engel et al., monocular cameras have the following advantages over other sensors: they provide rich information at a low weight and cost and in contrast to depth sensors, a monocular SLAM system is not intrinsically limited in its visual range, and therefore can operate both in small, confined and large, open spaces. The drawback however is that the scale of the environment cannot be determined from monocular vision alone, such that additional sensors, such as the IMU or air pressure sensor, are required.

Our navigation system consists of three major components: a monocular SLAM implementation for visual tracking, an Extended Kalman Filter (EKF) for data fusion and prediction, and PID control for pose stabilization and navigation. All computations are performed off-board. Since SLAM updates entire map for every frame and needs sparse map of high-quality features, which is so expensive that cannot fulfill real-time requirement in UAV navigation system. We used Parallel Tracking and Mapping (PTAM) instead, which is a camera tracking system for augmented reality and first implemented by Klein and Murray [13] . PTAM provides 3D position information of the camera by using key-frames which are dense map of low quality features. PTAM also splits the tracking and mapping into two threads, as *Fig. 8* shows, and implements multi-stages algorithm to fulfill the task.

C. Solution

In our paper, the node used in trapped personnel evacuation guidance is located in the file autonomous_flight.py. The flow chart is shown as *Fig. 9*. We combine navigation system described in Sec. IV in our autonomous flight controller. Once quadrotor take off, the node will initialize PTAM at first. Since PTAM will capture the first two frames of images as key frames, the quadrotor will fly up and down about 1 meter to make the scale estimate more accurate. We publish our flight plan to navigation system and then subscribe the feedback information /predicted_pose from it. According to the error between predicted position and desired position, this node will publish topics cmd_vel to control motors of quadrotor.

The integrated solution of the emergency service mission described in Sec. II is shown as *Fig. 10*.

Here the specific flight path planning is set as we discussed in Sec. IV inside the autonomous flight controller. Once the quadrotor takes off, navigation system will provide predicted position of itself by fusing monocular SLAM described in Sec. IV and IMU data. And the autonomous flight controller will send commands to control our quadrotor according to feedback information from navigation system. Meanwhile, the node face_detector.py in Sec. III is running simultaneously, and will report total number of trapped people to remote computer when quadrotor leave the room. In addition, once emergency situation happens, people can manual control quadrotor remotely by using joystick controller.

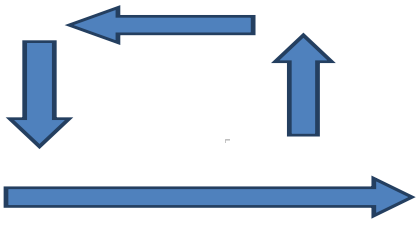

Fig. 7. Path planning of quadrotor.

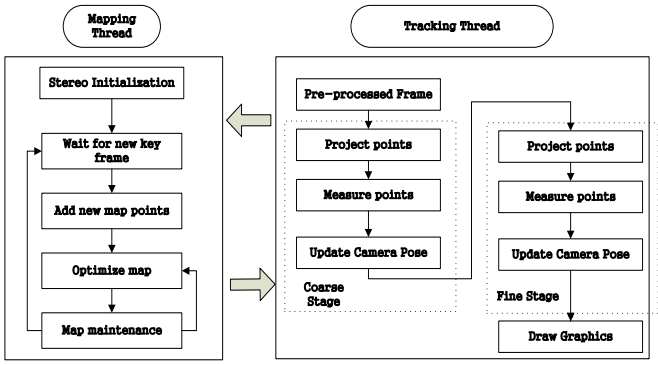

Fig. 8. PTAM: mapping & tracking thread.

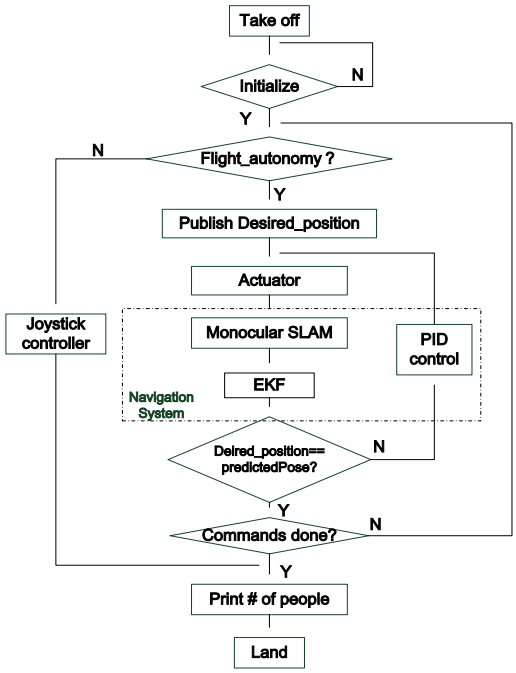

Fig. 9. Flow chart of autonomous_flight.py node.

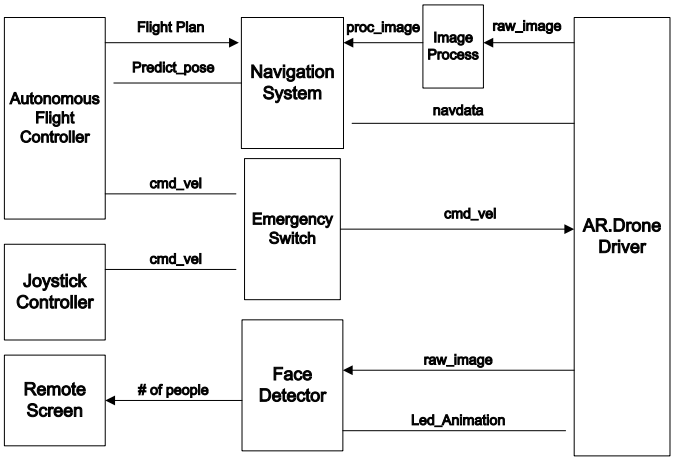

Fig. 10. Flow Chart of Emergency Service Scenario.

D. Experiments & Results

The hardware platform consists of a Wifi-controlled AR.Drone quadrotor with cameras attached to it, a joystick to control quadrotor in emergency situation and ground-based laptop to perform computing off-board. The hardware test system is shown in *Fig. 11*. In our experiment, our Monocular SLAM solution is implemented firstly in a tiny, clutter indoor environment. *Fig. 12* shows the commands of linear velocity which is published by autonomous flight controller. And *Fig. 13* shows the predicted position of quadrotor. The practical flight trajectory is shown in *Fig. 14*, which shows that the quadrotor finish the task as we expect.

Fig. 11. Hardware test System.

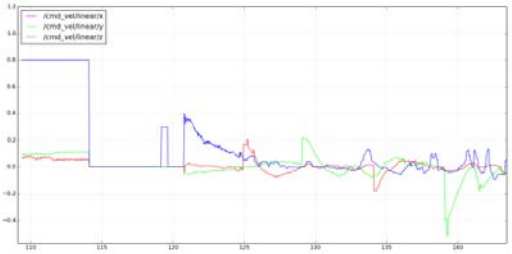

Fig. 12. Linear Velocity in Monocular SLAM Solution.

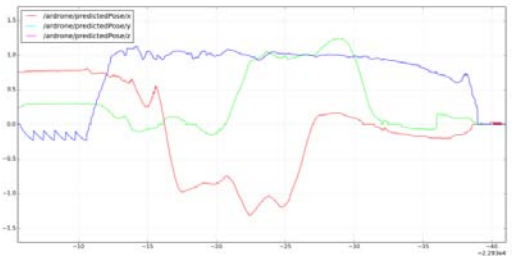

Fig. 13. Predicted Pose in Monocular SLAM Solution.

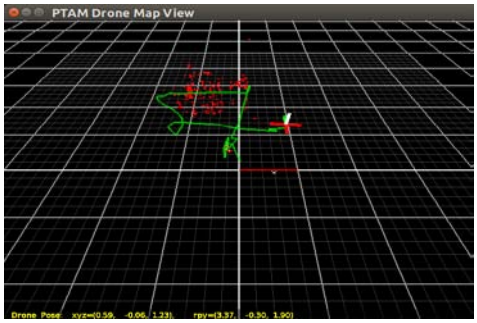

Fig. 14. Predicted Pose in Monocular SLAM Solution.

Our solution is compared with open loop motor control, which is simpler method to run the flight path planning. In the *Fig. 15*, the quadrotor UAV is controlled to finish the flight path planning by sending cmd_vel to its driver directly without any feedback. However, comparing with *Fig. 14*, it is quite clearly that flight trajectory in open-loop motor control is not as accurate as our solution. Because the quadrotor is sensitive to wind or other factors which will cause it diverges its original flight trajectory.

Fig. 15. cmd_vel in Open-loop Motor Control Solution.

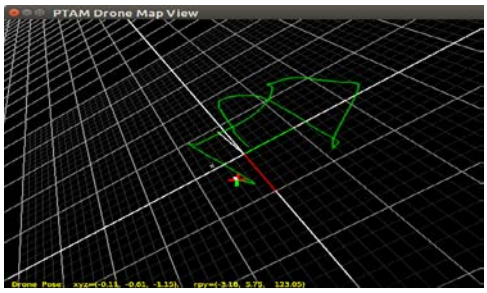

Fig. 16. Flight Trajectory in Open-loop Motor Control.

V. CONCLUSION

In this paper, we have described an emergency services scenario and presented corresponding mission requirement for trapped personnel detection and evacuation guidance in indoor fire scene based on quadrotor UAV. We introduced the parrot AR.Drone quadrotor as a hardware platform and ROS as a software platform to implement the mission. Currently, the integrated solution has been developed and tested in practical indoor environment. With these experimental results, we demonstrated that our solution is feasible and effective. We believe in the future our solution is worthy for search-and-rescue work. Further improvements could be how to control several quadrotors to do search-and-rescue works in different rooms simultaneously, and adding more functions & methods for evacuation guidance, such as music soothe.

References

[1] S. Gupte, P. I. T. Mohandas and J. M. Conrad, "A survey of quadrotor Unmanned Aerial Vehicles,", 2012, pp. 1-6.

[2] R. Mahony, V. Kumar and P. Corke, "Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor," IEEE Robotics&Automation Magazine, vol. 19, pp. 20-32, 2012-01-01 2012.

[3] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features,", 2001, pp. I-511-I-518 vol.1.

[4] R. Lienhart and J. Maydt, "An extended set of Haar-like features for rapid object detection,", 2002, pp. I-900-I-903 vol.1.

[5] "Cascade Classification,", 2015. http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html.

[6] "Robot Operating System (ROS)," http://www.ros.org/wiki/.

[7] "vision_opencv," http://wiki.ros.org/vision_opencv.

[8] S. Grzonka, G. Grisetti and W. Burgard, "Towards a navigation system for autonomous indoor flying,", 2009, pp. 2878-2883.

[9] M. W. Achtelik, S. Lynen, S. Weiss, L. Kneip, M. Chli, and R. Siegwart, "Visual-inertial SLAM for a small helicopter in large outdoor environments,", 2012, pp. 2651-2652.

[10] F. Fraundorfer, L. Heng, D. Honegger, G. H. Lee, L. Meier, P. Tanskanen, and M. Pollefeys, "Vision-based autonomous mapping and exploration using a quadrotor MAV,", 2012, pp. 4557-4564.

[11] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, "Visual odometry and mapping for autonomous flight using an RGB-D camera," in International Symposium on Robotics Research (ISRR), 2011, pp. 1-16.

[12] J. Engel, J. Sturm and D. Cremers, "Scale-aware navigation of a low-cost quadrocopter with a monocular camera," ROBOTICS AND AUTONOMOUS SYSTEMS, vol. 62, pp. 1646-1656, 2014-01-01 2014.

[13] G. Klein, G. Klein, D. Murray, and D. Murray, "Parallel Tracking and Mapping for Small AR Workspaces,", 2007, pp. 225; 1-10; 234.