Adv Syst Sci Appl 2018; **03**:17–28
Published online at http://ijassa.ipu.ru/index.php/ijassa/article/view/583

Visual odometry approaches to autonomous navigation for multicopter model in virtual indoor environment

Alexander V. Abdulov[1], Alexander N. Abramenkov[1] ,*Andrey A. Shevlyakov[1]

[1] *Institute of Control Sciences of the Russian Academy of Sciences, Moscow, Russia*

Abstract: Using simulators and virtual environments can simplify the development and testing of various control, behavior and navigation scenarios significantly. In this article we describe our experience using the ROS environment and the Gazebo simulator to develop an autonomous multicopter UAV that can navigate an unknown building and locate an exit using computer vision techniques.

Keywords: virtual simulation, visual odometry, UAV, autonomous navigation

1. INTRODUCTION

As more powerful embedded computers appear, along with smaller cameras, autonomous indoor navigation for mobile robots becomes a sufficiently interesting and tractable problem, inciting new research [8, 13].

Since global positioning signal is usually unavailable indoors, mobile robots face hard problems of determining their position in reference to some external beacons, route planning and actuating movement. The essential difficulty is not navigation per se, but locating beacons using available sensors. Autonomous mobile robots may face additional challenges in dynamic environments.

Today's technology allows for relatively easy development of autonomous mobile robots that can navigate using only onboard sensors [1, 9]. However, there are no universal algorithms for navigation in unknown scenes with dynamic obstacles.

Simulation software exists to facilitate the development and lower the entry threshold of knowledge (e.g. DIY robotics). Well-designed simulators (see Gazebo [11]) make rapid testing possible, as well as robot model building and their deployment in realistic scenarios.

In this article we share our experience if solving a problem of autonomous navigation in virtual environment using widespread visual odometry methods. We proceed with the problem and scenario description, which is considered as a test ground for two different approaches. We discuss strong points and limitations of the methods, as well as evaluation of their applicability and overall difficulties of this type of research.

*Corresponding author: aash29@gmail.com

2. PROBLEM STATEMENT

As an example of a navigation problem, we chose the following mission: a multicopter of a specified model must autonomously locate a window in an unknown building and leave via this window, without colliding with obstacles or using any kind of external positioning system. Onboard sensors include IMU (accelerometers and gyroscopes) and computer vision sensors (LiDAR and video cameras).

Building in question is an enclosed virtual hangar with one open window (see fig. 2.1). The hangar has a rectangular plan. Texture, size and color of walls may vary. Obstacles are represented by cylindrical pillars of varying height, diameter and color. Pillars are static, but their position is random. They may also be combined to form more complicated structures. The window is rectangular, although its position and dimensions are not known.

As the mathematical models of sensors and whole UAVs are usually available, we shall not pay too much attention to the hardware. For our research we chose a quadrotor scheme (fig. 2.2) as the most widespread and small enough (less than 0.5 m in dia) to operate indoors. A multitude of virtual sensors can be combined and installed onboard.

Virtual multicopter movement is controlled by feeding an embedded controller the reference values of linear and angular velocity. The make of embedded controller is fixed, thus we implement the autonomous movement with higher-level PID controllers that have to be tuned separately.

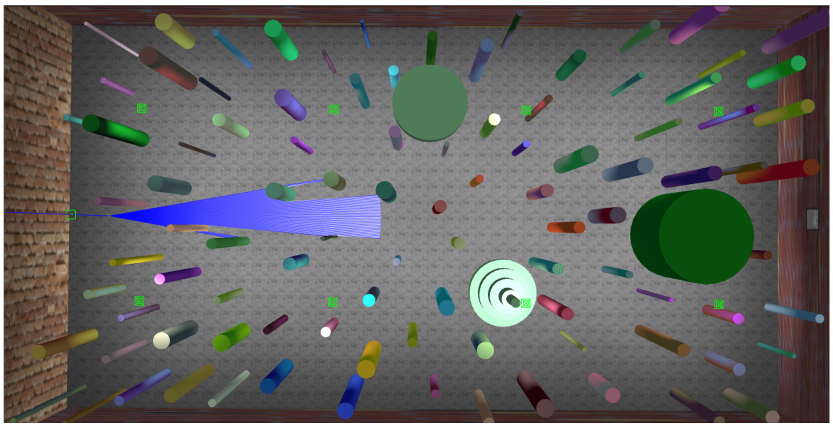


Fig. 2.1. Top view of the virtual hangar

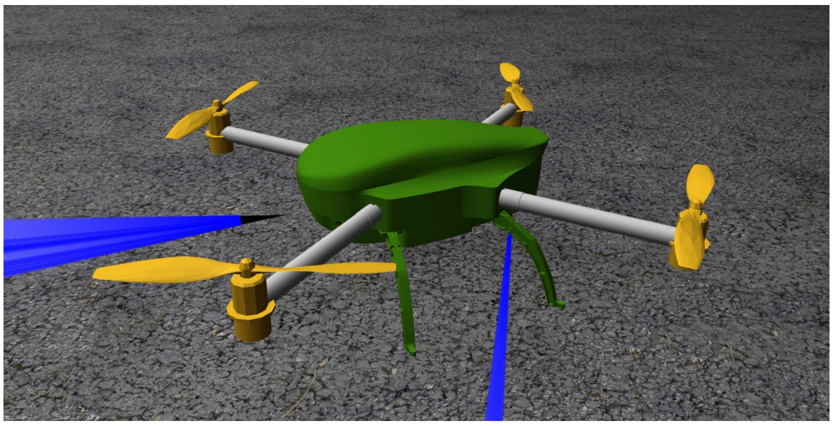


Fig. 2.2. Multicopter model

3. PROPOSED SOLUTION

3.1. Technology

Simulation is run on Ubuntu 16.04 OS with ROS Kinetic installed. ROS (Robot Operating System) is a framework for robot programming. It is based on graph architecture, where data processing is performed in the nodes that can receive and transmit messages to other nodes. There are many useful packages for ROS, which can also be written from scratch using standard interface. C++ and Python are supported. We should mention separately the MAVROS package that provides the data exchange interface between ROS and autopilot firmware using MAVLink (Micro Air Vehicle Link) protocol, which is designed for UAV communication.

As a simulator we chose the Gazebo software due to the availability of necessary UAV and sensor models, as well as ROS integration. Additionally, we use OpenCV library for image processing and DLib library for numerical optimization.

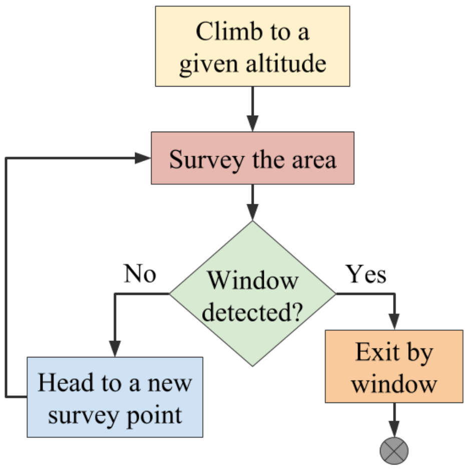
3.2. Flight strategy

High-level strategy for a chosen UAV model is described as a finite state machine and contains the following steps (see fig. 3.3a).

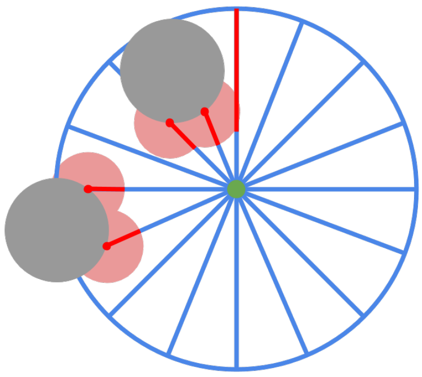
1. Climb to a given altitude.
2. Survey the area. On this step the multicopter rotates (up to 360 degrees). Two tasks are accomplished – searching for window and building a local obstacle map.
3. If a window is found, multicopter changes heading to face the window, and next step is performed; otherwise, multicopter flies forward in the direction chosen according to the algorithm and surveys again (step 2).
4. Head to the window avoiding obstacles and exit the building.

Local obstacles are mapped using frontal laser LiDAR (see fig. 3.3b). For each heading we save the distance to the nearest obstacle and add a safety margin which allows to avoid collisions. A new direction of movement is chosen as the clearest one and closest to the course set.

On a lower level, we must control and stabilize the UAV model by flight altitude, yaw angle and linear velocity.



(a) Flowchart of states change



(b) Obstacles are grey, safety margins are pink

Fig. 3.3. Flight strategy and local obstacles map

3.3. Angle stabilization

Orientation estimate obtained via MEMS gyroscopes can accumulate error due to “zero drift” problem. Algorithms exist for fusing different sensors (accelerometers, gyroscopes and magnetometers) to refine estimates. For example, Madgwick AHRS [12] algorithm or extended Kalman filter [14].

We should note that the model firmware used IMU readings to interpret the control signals. In case of yaw angle error this feature did not allow to control the UAV correctly and had to be accounted for while specifying control signal. Since in case of simulation the magnetometer readings were relatively accurate, they were used to rectify the yaw estimate.

3.4. Altitude stabilization

It is reasonable to use a downward-facing range-finder (altimeter) to stabilize altitude. An ultrasound range-finder can be used, but a more narrow-bear laser range-finder is preferable. Additionally, orientation of the aircraft must be accounted for.

On the basis of this estimate, it is relatively simple to tune the coefficients for a PID controller which can adequately stabilize the reference altitude (see fig. 3.4). Here and below, we denote: $Target$ – reference value; *Real* – ground truth, obtained from simulator; PID – sensor estimate, fed to the PID controller.

3.5. Velocity stabilization

In GPS-denied indoor spaces we need to use other information to obtain current velocity estimate. In principle, IMU can be used, but due to large errors it needs to be amended by other means. Visual odometry is a promising approach [3]. A sequence of images captured by camera can be a main or auxiliary source of position information. Depending on number of cameras and their configuration, various methods can be used. We’ll consider two approaches, mono and stereo visual odometry, that necessitate installing one or two calibrated cameras respectively.

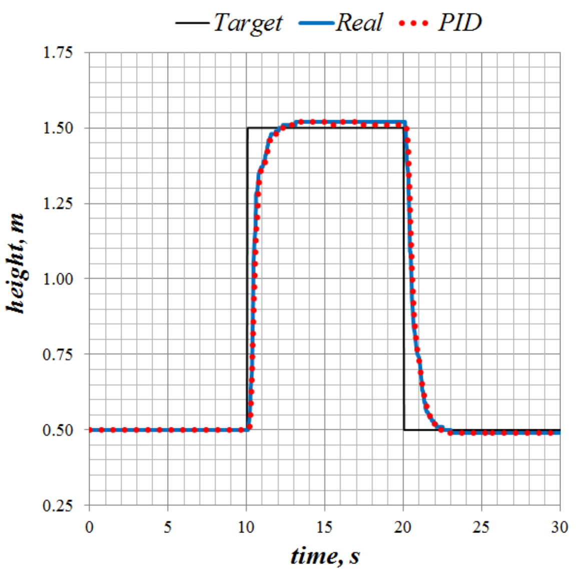


Fig. 3.4. Flight altitude stabilization by PID

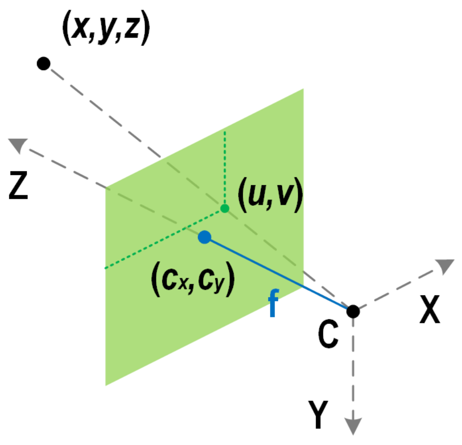


Fig. 3.5. Perspective projection camera model

3.6. Camera model

Visual odometry operates on a basis of a camera model. Often this is a relatively simple perspective projection model (see fig. 3.5), represented by a matrix (3.1) of intrinsic parameters (focal length and principal point). In more complicated cases, distortion transformations are also accounted for.

$$C = \begin{vmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{vmatrix} \tag{3.1}$$

Usually the intrinsic parameters are determined by camera calibration. For a virtual camera we chose image center as the principal point (c_x, c_y) and obtain focal length $f = f_x = f_y$ from equation (3.2), knowing resolution s and field of view (FoV) angle α.

$$\alpha = 2 \cdot \arctan \frac{s}{2 \cdot f} \tag{3.2}$$

Equation (3.3) gives us the relation of point (x, y, z) in 3D scene and its projection (u, v) on image plane up to a scale coefficient w. This is a composition of rotation matrix R and translation vector T.

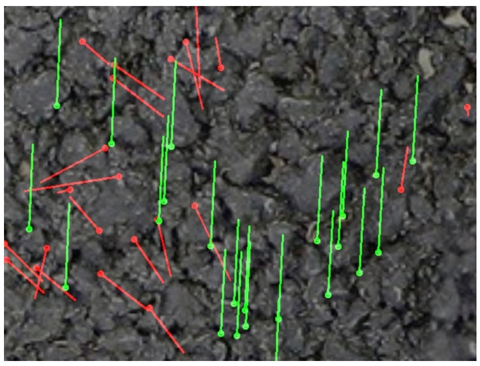
$$w \cdot \begin{vmatrix} u \\ v \\ 1 \end{vmatrix} = C \cdot \begin{vmatrix} R & T \end{vmatrix} \cdot \begin{vmatrix} x \\ y \\ z \\ 1 \end{vmatrix} \tag{3.3}$$

3.7. Monocular visual odometry

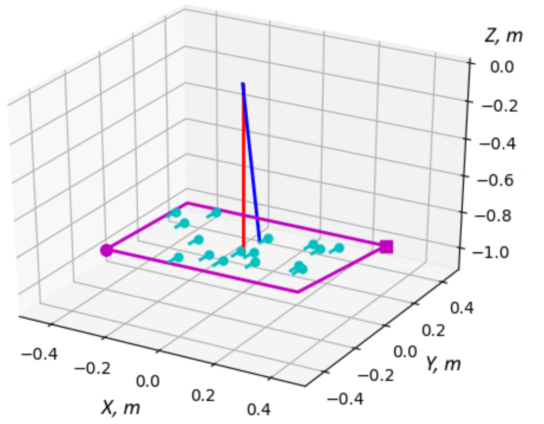
Consider a case of one camera, laser altimeter and 6-axis IMU. If a camera is downward-facing, then we can assume that most of the observed feature points belong to one plane. This is a reasonable assumption if the flight altitude is low, and it was almost always valid in our simulation.

Knowing altitude h and camera orientation quaternion Q_{IMU}, we can compute floor plane equation $a \cdot x + b \cdot y + c \cdot z + d = 0$ coefficients, as indicated in (3.4). First three coefficients are obtained by rotation operation $\circ$ a unit vector pointing along an axis Z.

$$(a, b, c) = Q_{IMU}^{-1} \circ (0, 0, 1); \quad d = -h \cdot c \tag{3.4}$$



(a) Keypoints optical flow with homographic filtering between two consequent images of floor; matched points in green, filtered points in red



(b) Keypoints flow projection on the floor plane

Fig. 3.6. Monocular odometry approach to velocity estimation

Taking coordinates (u, v) of floor image feature points and substituting (3.5) in (3.6), we reconstruct their position (x, y, z) in camera frame. Since frame used in aircraft firmware does not match the camera frame, some axes and signs are inverted in equations for x and y.

$$\hat{u} = \frac{u - c_x}{f_x}; \quad \hat{v} = \frac{v - c_y}{f_y}; \tag{3.5}$$

$$z = \frac{d}{a \cdot \hat{u} + b \cdot \hat{v} + c}; \quad x = -z \cdot \hat{v}; \quad y = -z \cdot \hat{u} \tag{3.6}$$

As a result, linear velocity estimate is carried out using feature keypoints position (FAST [5], good features to track [7]) from to consequent measurements. Correspondence between feature points can be found, for example, by optical flow or by comparing descriptors [4]. We used a well-established Lucas-Kanade algorithm [2], due to its modest computational requirements and adequate performance in case of small movements. The implementation of this algorithm is available in OpenCV library.

It is useful to apply additional filters to reject outliers: mean velocity, direction, etc. An assumption of all points belonging to one plane allows us to successfully use the filter based on homography matrix (see fig. 3.6a), obtained on RANSAC [10] method.

This approach is schematically depicted on fig. 3.6b. In this case, multicopter speed is towards negative direction of axis Y. Thus, projections of feature points are moving in the opposite direction. Red line is normal distance to the floor, blue line is its estimate by altimeter and IMU readings, magenta plane is the observed floor plane, cyan markers are feature points and their movement.

In general, monocular approach allows us to estimate velocity while flying at low altitude (less than 1 m), and is computationally inexpensive. Flying higher leads to poor performance. This is due to two factors:

1. Aircraft rotation at higher altitude leads to more displacement for optical flow to handle, which leads to more errors and outliers.
2. Flying higher increases the probability of obstacles in view field, which invalidates the assumption of all points belonging to the floor plane.

Here the task of odometry is not fully solved. Only velocity is estimated, and then stabilized by a suitably tuned PID controller, without trajectory construction.

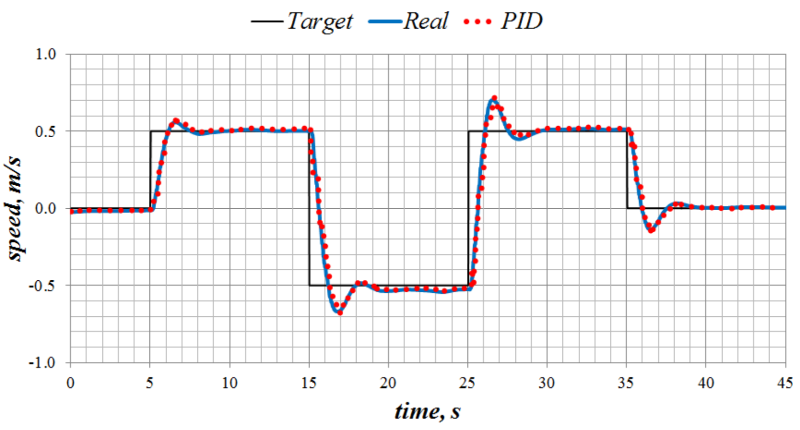


Fig. 3.7. Velocity stabilization via monocular odometry approach

Therefore, the accuracy of this method makes it possible to successfully use it. Speed estimates are adequate, as can be seen on fig. 3.7. Stabilization time is acceptable, although overshooting is present.

3.8. Stereo visual odometry

Stereo visual odometry is an alternative approach to estimating velocity. In this work we used a stereoscopic method based on reprojection error minimization [6]. It consists of several steps:

1. For each feature point P^{3D} need to find correspondence between its projections on current left and right frames of a stereo camera (p^l_t, p^r_t) and on one of the previous frames (p^l_{t-1}). This can be seen in fig. 3.8. As in monocular case, we use Lucas-Kanade optical flow.
2. Using the established correspondence between the projections of all feature points on the left (u_l, v_l) and right (u_r, v_r) frames of a stereo camera with base b, the triangulation problem (3.7) is solved to restore their position (x, y, z) in 3D space (see fig. 3.9).

$$(x, y, z) = \frac{b}{u_l - u_r} \cdot (u_l - c_x, v_l - c_y, f) \tag{3.7}$$

3. Knowing current position on each feature point $p^{3D}_{i,t}$ in 3D space, and its projection $p^l_{i,t-1}$ at a previous time step, linear displacement T and rotation R are estimated by minimizing (3.8) reprojection error.

$$\underset{R,T}{\operatorname{argmin}} \sum_i \| p^l_{i,t-1} - \hat{C}(R \cdot p^{3D}_{i,t} + T) \| \tag{3.8}$$

Equation (3.8) contains a projection operator $\hat{C}$ that maps a point from 3D space to the image plane according (3.3). We used the euclidian distance $\|\cdot\|$ as a norm.

This approach is more versatile, and does not demand additional hardware apart from two properly fixed cameras (downward or forward facing). The main limitation is the necessity of multiple observed feature points. Their allowed distance depends on stereo camera base.

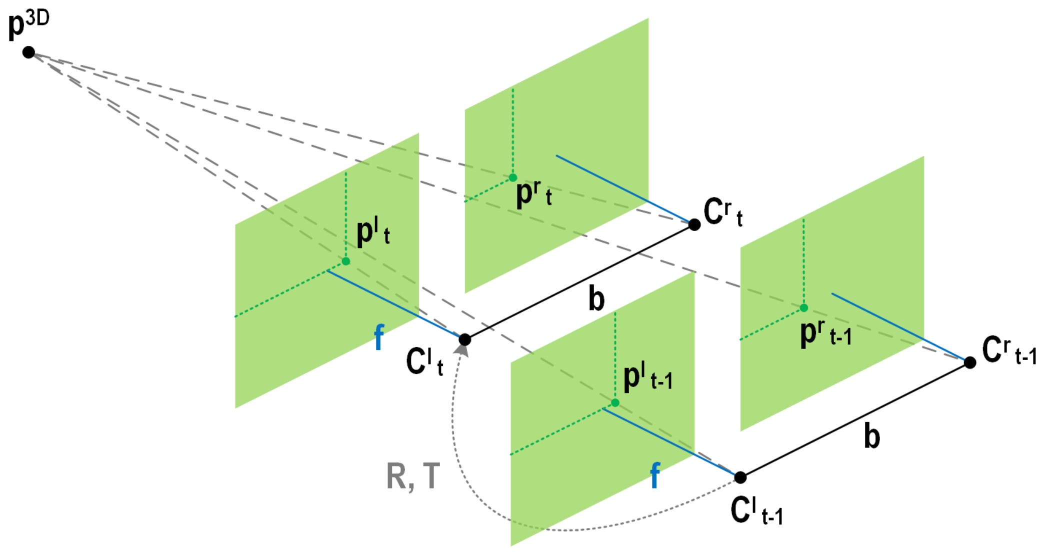


Fig. 3.8. Schematic representation of 3D point projection on stereo camera matrix during movement. C_t^l, C_t^r – left and right cameras at time t. C_{t-1}^l, C_{t-1}^r – left and right cameras at time $t-1$

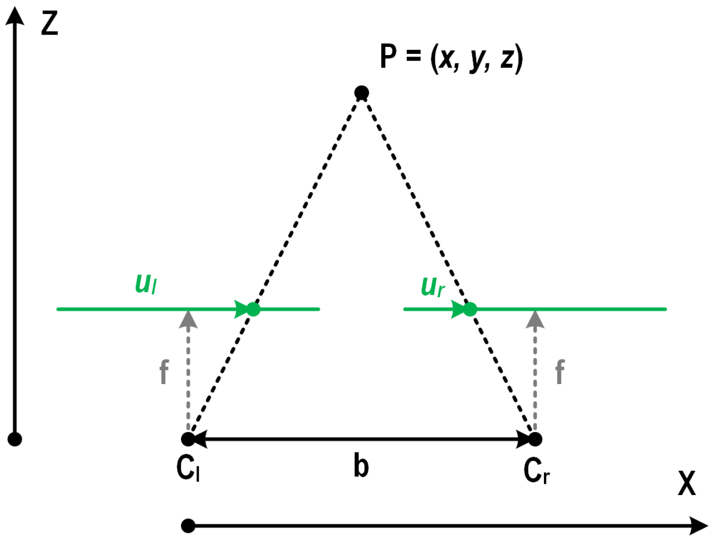


Fig. 3.9. Triangulating P for a stereo camera with base b

Stereo visual odometry approach is also more computationally expensive – for the same video parameters (648x486, 36fps) processing was 2-3 times longer. In our case, monocular odometry allowed for real time computation, while stereo odometry had to skip a half of available frames. It is possible, however, to decrease the precision in order to speed up the computations.

In the examples considered, forward facing camera did not always perform well due to lack of texture on pillars which yielded few feature points. Downward facing camera was more stable, although there was a minimum necessary altitude that allowed both cameras to see the same objects.

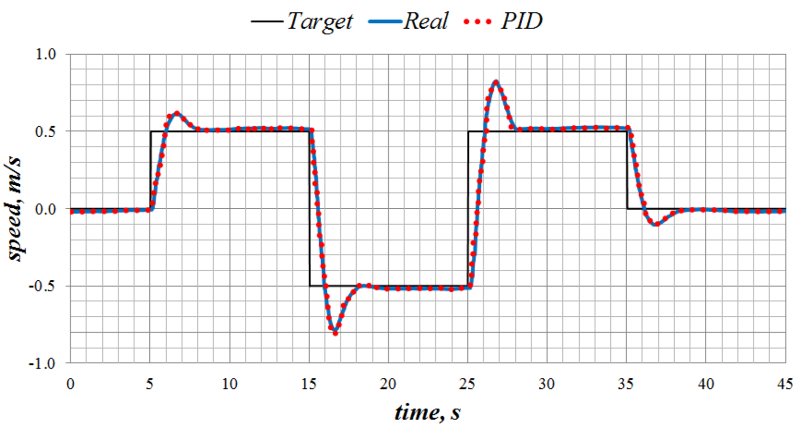


Fig. 3.10. Velocity stabilization via stereo odometry approach

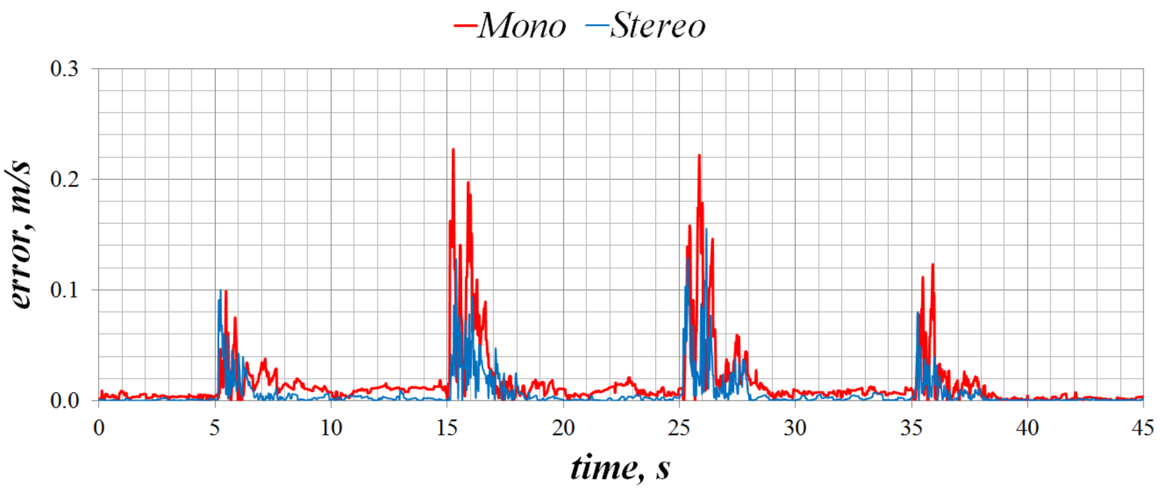


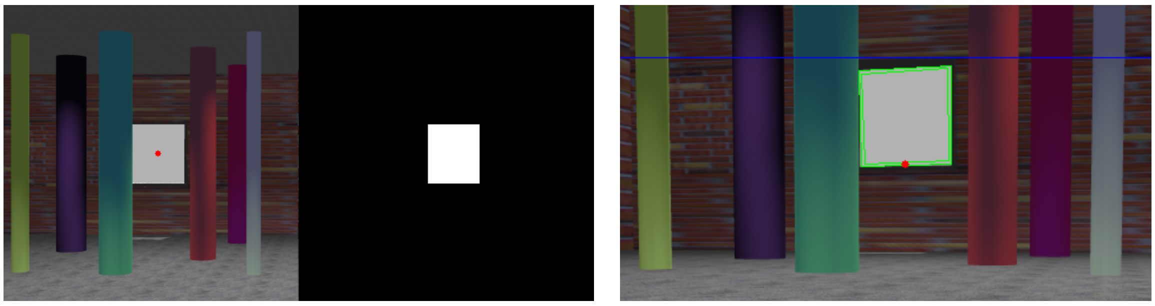
Fig. 3.11. Errors of calculated velocities for monocular and stereo approaches, $error = |PID - Real|$

Figure 3.10 shows the velocity stabilized with tuned PID controllers via the estimate obtained by this approach. Velocity estimation is somewhat better than in monocular case – the plots of $Real$ and PID nearly match (see fig. 3.11).

4. SEARCH FOR WINDOW

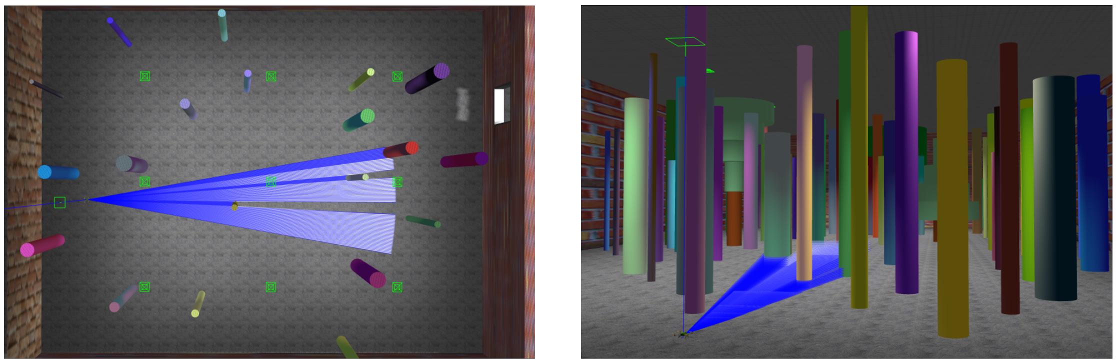
To detect a window, we used color segmentation. Assuming that the background visible through the window was a different color than the rest of the scene, we needed to find a suitably colored shape (see fig. 4.12a). Furthermore, shape filtering, e.g. rectangle detection (see fig. 4.12b), can be used.

At a last stage, when the aircraft approaches a window, it can become disoriented as number of feature points decreases or they disappear entirely. In this case, we solve a ballistics problem – aim for the window and fix a exit speed, disabling regular velocity controllers.



(a) Color segmentation and threshold binarization (b) Specifically-colored rectangle searching

Fig. 4.12. Searching for window



(a) Simple (b) Difficult

Fig. 4.13. Examples of the scenes

5. SIMULATION RESULTS

As a result of several simulation trials, we established that the described approach solves a problem of exiting the building through the window. Two scenes of different layouts were explored.

Simple scene dimensions are 40x50x10 m (see fig.4.13a). Obstacles are represented by cylindrical pillars of different radii. Aircraft could detect the window directly from the starting point and approached it while avoiding the obstacles.

Difficult scene dimensions are 50x90x15 m, obstacles are simple pillars and composite objects of several different-sized cylinders, imitating trees. (see fig. 2.1, 4.13b). In this case, obstacles blocked the window completely, thus search strategy had to be activated. The parameters chosen resulted in a fairly conservative behavior – there was no collisions, but search required a sufficiently long time.

Thus, a relatively simple search algorithm, that didn't map the scene or localize of the aircraft, could accomplish the task of exiting the building.

6. TECHNICAL DIFFICULTIES

We encountered some complications while using the Gazebo simulator, which should be accounted for. Notably, high computational requirements, which resulted in unstable work. Sometimes, a kind of "numerical wind" was apparent, a seemingly random perturbations that forced the aircraft to crash.

The effect could be reproduced by running both client and server parts of the simulator on the same medium-powered computer. For all our experiments we used an Intel Core i7-4510U CPU @ 2.00GHz, 8GB RAM PC. Additional disturbances were created by other applications running in the background.

Practically, Gazebo is not the most comfortable software to use in case of multiple test runs. Each run required restarting the whole virtual environment and took a lot of time, which combined with the "numerical wind" made our work more difficult – especially the tuning of PID controllers.

We encountered various other difficulties, including the necessity to convert between every coordinate frame involved (world, aircraft, sensor). This, however, is anticipated by MAVLink protocol which offers the possibility of specifying control in several coordinate frames (NED, ENU, etc.)

7. RESULTS

Numeric simulation using the ROS framework, the Gazebo simulator and the rest of the corresponding ecosystem allowed us to test the applicability of the chosen algorithms in specified conditions. Use of this class of simulators and frameworks can be of much help while developing real hardware and software platforms for UAVs.

REFERENCES

[1] A. Weinstein, A. Cho, G. Loianno, & V. Kumar. (2018). Visual inertial odometry swarm: An autonomous swarm of vision-based quadrotors. *IEEE Robotics and Automation Letters*, *3*(3), 1801–1807.

[2] B.D. Lucas, & T. Kanade. (1981). An iterative image registration technique with an application to stereo vision. *International Joint Conference on Artificial Intelligence*, 647–679.

[3] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, ... J.J. Leonard. (2016). Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age. *IEEE Transactions on Robotics*, *32*(6), 1309–1332.

[4] D. Bekele, M. Teutsch, & T. Schuchert. (2013). Evaluation of binary keypoint descriptors. In *2013 IEEE International Conference on Image Processing* (pp. 3652–3656). doi:10.1109/ICIP.2013.6738753

[5] E. Rosten, & T. Drummond. (2006). Machine learning for high speed corner detection. *9th European Conference on Computer Vision*, *1*, 430–443.

[6] H. Badino, & T. Kanade. (2011). A head-wearable short-baseline stereo system for the simultaneous estimation of structure and motion. *IAPR Conference on Machine Vision Applications (MVA),Nara, Japan.*

[7] J. Shi, & C. Tomasi. (1994). Good features to track. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 593–600.

[8] K. Mo, H. Li, Zh. Lin, & J. Lee. (2018). The adobeindoornav dataset: Towards deep reinforcement learning based real-world indoor robot visual navigation. *arXiv preprint arXiv:1802.08824.*

[9] L. Silvestri, L. Pallottino, & S. Nardi. (2017). Design of an indoor autonomous robot navigation system for unknown environments. In *International Conference on Modelling and Simulation for Autonomous Systems* (pp. 153–169). Springer.

[10] M.A. Fischler, & R.C. Bolles. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, *24*(6), 381–395. doi:10.1145/358669.358692

[11] N. Koenig, & A. Howard. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2149–2154.

[12] S. Madgwick. (2010). An efficient orientation filter for inertial and inertial/magnetic sensor arrays. Retrieved from http://x-io.co.uk/res/doc/madgwick_internal_report.pdf

[13] Sh. Yang, S.A. Scherer, X. Yi, & A. Zell. (2017). Multi-camera visual slam for autonomous navigation of micro aerial vehicles. *Robotics and Autonomous Systems*, *93*, 116–134.

[14] X. Li, M. Chen, & L. Zhang. (2016). Quaternion-based robust extended kalman filter for attitude estimation of micro quadrotors using low-cost mems. *Control Conference (CCC), 2016 35th Chinese*, 10712–10717.