ISSN 2076-2429 (print)
ISSN 2223-3814 (online)

UDC 62-529

S.G. Antoshchuk[1], DSc, Prof.,
O.M. Maksymov[1],
M. Wendl[2]
[1] Odessa National Polytechnic University, 1 Shevchenko Ave., Odessa, Ukraine, 65044; e-mail: m.alex.m11@gmail.com
[2] University of applied sciences Erfurt, 25 Altonaer Strasse, Erfurt, Germany, 99085; e-mail: max.wendl@fh-erfurt.de

AUTOPILOT MODEL FOR RETURNING AN UNMANNED AERIAL VEHICLE TO ITS STARTING POINT IN CASE OF ELECTROMAGNETIC NOISE

С.Г. Антощук, О.М. Максимов, М. Вендл. **Модель автопілоту, що повертає безпілотний літальний апарат в точку старту при виникненні електромагнітної перешкоди.** Розглянуто можливість повернення безпілотного літального апарата при виникненні електромагнітної завади, що блокує використання глобальної системи позиціонування та радіо керування. Показано, що в разі збору інформації про місцевість, над якою пролягає маршрут безпілотного літального апарата, за допомогою пасивних датчиків та камер можливо позиціонування апарату для повернення на місце старту. Проведено аналіз моделей, які дозволили створити симуляцію процесу польоту та позиціонування.
Ключові слова: автопілот, метод одночасного позиціонування та відтворення карти, комп'ютерний зір

S.G. Antoshchuk, O.M. Maksymov, M. Wendl. **Autopilot model for returning an unmanned aerial vehicle to its starting point in case of electromagnetic noise.** The possibility of returning an unmanned aerial vehicle in case of electromagnetic interference, which blocks the use of the global positioning system and radio control system, is considered. It has been shown that in the situation of gathering information about the area over which the route of unmanned aerial vehicle runs, using passive sensors and cameras, it is possible to position the machine to return to the starting point. An analysis of models, which allowed creating a simulation of flight process and positioning, was made.
Keywords: autopilot, simultaneous localization and mapping, computer vision

Introduction. Unmanned aerial vehicles (hereafter UAV) are very popular in our world. They are used in all parts of human life be it for taking pictures from perspectives men cannot reach or helping farmers by getting an overview about their land and the status of their plants. Also, they are used in the army like eyes for artillery. They are not big which makes them hard to notice in the sky and hard to hit by a machine gun. But there is one simple idea to fight them with: electromagnetic interference which aims not only at the radio control system of the drone but also at the video transmitter as well as at the global positioning system (GPS). GPS facilitates a simple autopilot and some base return systems. So, in case of using UAV's for the army like DJI Phantom or others their control and position can be lost if the enemy is using an electromagnetic noise weapon. The reason for this is that the UAV becomes "blind" in the air.

We present the idea of a return system that uses only the following passive independent sensors, camera, accelerometer, gyroscope, magnetometer and barometer (Fig. 1).

Analysis of literary data and formulation of the problem. The basic concept of returning the drone is using simultaneous localization and mapping (SLAM). When it is flying in the direction of the goal, a map of distinctive points in pictures provided by the drone camera is created.

Out of this information the real position of the drone in the environment can be identified. This can be made very accurate if the UAV still has GPS connection at this moment. When the drone loses the connection, it is able to find its global position with the help of the camera. It is searching for feature points in the current photo of the camera and calculates its global position out of it. This will be possible by comparing these points with the global map of distinctive points that was generated at the beginning when there was still GPS and is being edited and extended the whole time. If there is no GPS from the beginning, the first picture will define the global context.

DOI: 10.15276/opu.3.53.2017.13

To provide task sharing and fast processing the idea is to use a Nvidia Jetson TX2 module and an Arduino Micro at the same time to calculate the position and send movement controls to the motors. The Nvidia Jetson is very powerful and fast for its size. This helps to process the images, which are received from the camera, quickly as well as to compute the new movement vectors. The Arduino does not have such a strong hardware, but it is very small and energy-saving which is also an important factor when constructing a drone. The Arduino board will handle the stabilization of the UAV, using proportional-integral-derivative controllers (PID), as well as interpret the controls coming from the Nvidia module and convert them into movement.

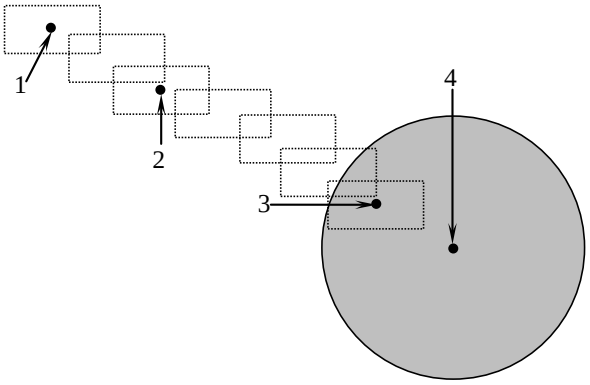

Fig. 1. Scheme of autopilot idea: starting point (1), flight with GPS autopilot or manual control (2), point of getting electromagnetic noise (3), electromagnetic noise generator (4)

Also a PID controller is needed for a point to point autopilot to handle deviation of the flying route. In order to establish a connection for communication between the Jetson TX2 and Arduino Micro software is required. Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions which attempt to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. It was determined that ROS can be used for the communication between the Arduino and the Nvidia Jetson which makes it possible to fly, take care of the camera and calculate the movement on different places and devices. This means an improvement in performance and speed of the computation and execution concerning the control of the drone.

Furthermore, something to process and analyze the pictures of the camera was required. Open Source Computer Vision Library (OpenCV) is a library of algorithms respective the development of computer vision and machine learning applications. It leans mostly towards real time vision applications and supports many programming languages as well as platforms. As they are on programming language side, C^{++}, C, Python, Java and MATLAB and on platform side Windows, Linux, Android and Mac OS. OpenCV therefore is very flexible which is helping the project in not being limited to one platform or programming language.

The OpenCV library can be used for camera calibration, feature detection, recognizing faces, tracking moving objects in series of images, stitching images together, finding similar images, removing red eyes from images etc. For the purpose of this project the features camera calibration, feature detection, tracking moving objects and finding similar images are very necessary and helpful.

With all this information a first approach to the topic was made and the first implementation is following. As the first approach it was decided to make a simulation with the help of Unity 3D simulator to test the concept, of returning a UAV to its starting point without using GPS, in an ideal environment, to find out if the plan is functional.

Purpose and objectives of the study

Make it possible to return a drone to its starting point in case of electromagnetic noise disturbing GPS and radio control by using passive sensors.

To achieve the purpose, the following tasks were identified:

– drone simulator with natural environment;

– point to point autopilot;

– autopilot environment to process data;

– camera model and camera calibration;

– feature points detection and mapping.

Approach and Realization

Drone simulator with natural environment. Previous research in the internet has shown , that such drone simulations in Unity already exist, but they do not simulate the behavior of each motor on its own, they are using only the drone center point for movement actions.

ISSN 2076-2429 (print)
ISSN 2223-3814 (online)

Fig. 2. Screenshot of Unity Drone simulator

Before implementing the idea in a real environment, we built a simulation with the help of the Unity 3D Simulator to understand the problem better and find solutions for the given problems. With this approach the problem occurred that the 3D drone model needs to behave like an actual drone. Also, the environment needs to have gravitation. However, Unity has a well-designed and easy to use implementation for this case. To simulate a bird's eye camera view the floor of Unity includes a satellite photo of Knox County Regional Airport that was found in the internet in high quality resolution (Fig. 2).

The drone and its physics are simulated with Nvidia PhysX. Every motor is simulated with a rotatory and vertical force. Our simulation is flying with applying forces to each single motor depending on the values the PID controller returns after it processed the input controls for movement. On Fig. 3 the double contoured PID controllers for pitch, yaw and roll can be found.

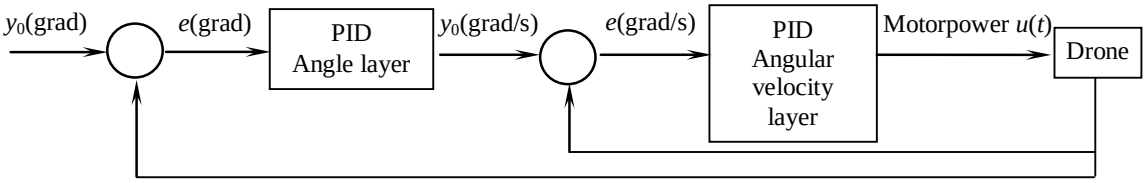

Fig. 3. Scheme of PID controllers of angle controlling

These three axes include six PID controllers. For height controlling the following contour is used (Fig. 4).

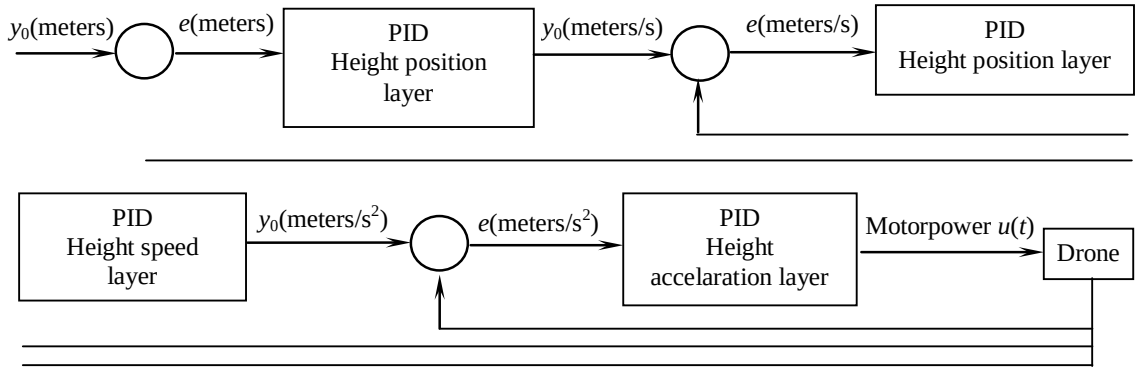

Fig. 4. Scheme of PID controller of height controlling

Altogether the drone stabilization uses nine PID controllers. The coefficients of them are shown in the Table 1.

After receiving all values from the bottom layer regulators, they are added or subtracted to each motor respecting the location of the drone. The initial power of the drone is height acceleration layer control.

Then each motor calculates vertical trust and relative torque depending on RPM (rotations per minute). RPM depend linearly on the power, the trust is quadratic dependent on RPM and torque is linearly dependent also on RPM.

Table 1

Coefficients of PID controllers used to stabilize drone in Unity Drone simulator

Name of PID controller	*K*-Proportional	*K*-Derivative	*K*-Integral
Pitch angular velocity layer	0.01	10	0.00000001
Pitch angle layer	0.1	0	0
Pitch angular velocity layer	0.01	10	0.00000001
Pitch angle layer	0.1	0	0
Yaw angular velocity layer	0.1	200	0.0
Yaw angle layer	0.1	0	0
Height acceleration layer	0.01	1	0.00001
Height speed layer	0.3	10	0.000001
Height position layer	0.1	10	0.000000001

Point to point autopilot. The class of point to point (hereafter PtP) autopilot is needed for the simple control of the drone like moving from A to B. It works according to the following steps:

1) Current position with time of position measuring. If it is time of position processing, not measuring, it won't work, because of wrong first and second derivative of position – speed and acceleration. ROS provided callback functions, when new position is arrived, but not provided time, when it was sent. The time of measuring is also sent with the vector of the position.

2) The destination position is also saved in a ROS topic. This is the result of a main program thread that calculates where it must fly.

In order to get the controls that would be sent to the drone (in our case via ROS-bridge to the Unity drone simulator) from that two points it uses another PID controller's contours (Fig. 5) which works with *X* (left and right movement) and *Z* (forward and backward movement) axis.

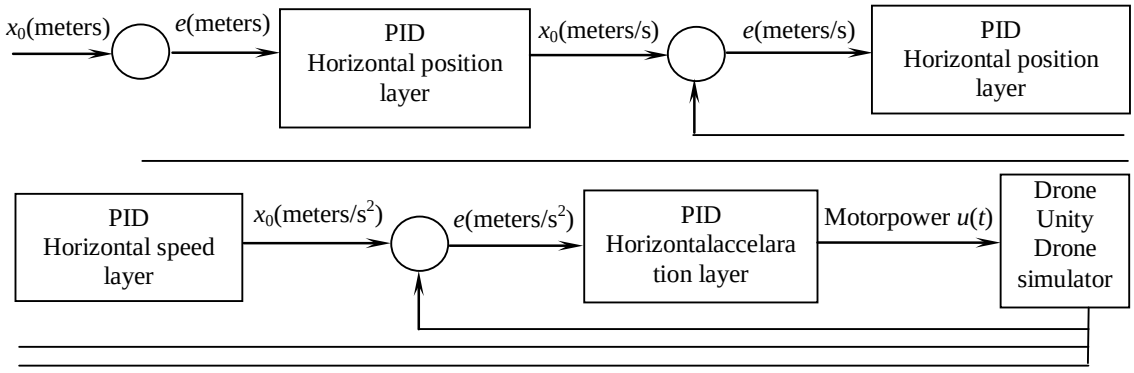

Fig. 5. Scheme of PID controllers of PtP autopilot

When the destination position is changed all PID controllers will be reset. However, this way only *X* and *Z* axis are controlled. The *Y* coordinate of destination point is sent directly to the controls quaternion of the drone. All coefficients of these controllers are received through single-handedly trial and error principle for each layer separately. The main goals to achieve were: speed of feedback and no rocking of the system. The discovered coefficients can be found in Table 2.

Table 2

Coefficients of PID controllers used to provide stable flight with PtP autopilot

Name of PID controller	*K*-Proportional	*K*-Derivative	*K*-Integral
Z acceleration layer	0.01	1	0.00001
Z speed layer	0.3	10	0.000001
Z position layer	0.1	10	0.000000001
X acceleration layer	0.01	1	0.00001
X speed layer	0.3	10	0.000001
X position layer	0.1	10	0.000000001

ISSN 2076-2429 (print)
ISSN 2223-3814 (online)

Autopilot environment. Now the side of the ROS environment needs to be set up. There is an existing ROS library called ROS bridge with the help of which it is possible to communicate between ROS and non-ROS programs. In our case it will be the Unity simulator. We found a C# library for ROS bridge which made it possible to send data from the drone, e. g. pictures of the camera to ROS and backwards.

It was decided to use ROS framework based on Python because Python is easy to use and very powerful. Also with the help of OpenCV it is possible to analyze and process images. A list of topics through which Unity and ROS can communicate was developed and is presented in Table 3.

Table 3

ROS topics, their data types and relations between drone and ROS environment

Name	Datatype	Drone	ROS
/drone/status/drone	UInt8	publish	subscribe
/drone/status/ros	UInt8	subscribe	publish
/drone/camera/1	sensor_msgs/CompressedImage	publish	subscribe
/drone/camera/calibration	sensor_msgs/CompressedImage	publish	subscribe
/drone/camera/undistort	sensor_msgs/CompressedImage	–	publish/subscribe
/drone/barometer	std_msgs/UInt32	publish	subscribe
/drone/rotation	geometry_msgs/Vector3	publish	subscribe
/drone/position	geometry_msgs/Vector3	publish	subscribe
/drone/position/destination	geometry_msgs/Vector3	–	publish/subscribe
/drone/control	geometry_msgs/Quaternion	subscribe	publish

Camera model and camera calibration. If there is feature detection and a global map of feature points is created it will be necessary to always have the map in scale of the real world. In case the drone still has GPS connection new feature points can easily be added to the map because the map and the real world have the same scale and the position of the UAV is known. When the GPS signal is lost the exact position of it in the world can be detected by finding its position in the map of feature points. This can be achieved by using the latest picture the camera sent, comparing the feature points of it with the global map and estimating the difference between the positions of the points. To get global position of feature point, pinhole camera [1, 2] was used (Fig. 6).

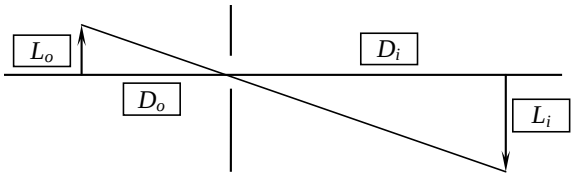

Fig. 6. Scheme of pinhole camera

The equation $L_i = L_o \cdot \frac{D_i}{D_o}$ de-scribes the relationship between the size of the object in real life and the size of the object in the image. For the drone this view is rotated 90° around the *z*-axis and D_i becomes the flight height of the drone, D_o represents the focal length of the camera. Since cameras in Unity do not support a focal length, the values of IMX219, a suitable camera for Arduino Micro and therefore a possible model for a real implementation, were used to ge-nerate the camera model.

L_o describes the distance of the object from the center of the picture which describes the position the drone is at and if the equation is converted to $L_i = L_o \cdot \frac{D_i}{D_o}$ the distance of the desired object in the real environment can be calculated.For mapping the feature points to real world objects and always having a map in scale to the real world this process is very necessary.

The pictures received from the camera also need to be normalized, which means every picture needs to have identical quality. Due to the fact that there will be geometric distortion [4] (Fig. 7, a) and blurring in a camera picture, camera calibration [5] is necessary. It will straighten the edges in the picture and adjust the focal point of the camera virtually since it cannot be changed in a real pinhole cam-

era. With the help of a chessboard and OpenCV this can be realized in Python. The algorithm finds the edges of the chessboard in a random picture. It needs to be defined how many rows and columns the board has. After this process a new camera matrix is returned which will be saved and used on every picture ROS receives from the drone in the future (Fig. 7, b).

a *b*

Fig. 7. Raw image from camera (a), undistorted image after camera calibration (b)

Feature detection and mapping. Now that every picture is flattened feature detection is possible. In general, a feature point is an "interesting" part of a picture. Normally they are described through edges, corners or blobs in a picture. There are many different algorithms which all use different criteria for feature detection. Many of them are provided by OpenCV. To decide, what algorithm will be used, tests were done. The tests were conducted with images of the size 512×512 pixels on average computer quality. The Table 4 shows: speed per frame – absolute total time in milliseconds spent on feature detection of a single frame; percent of tracked features – percent of successfully tracked features from original to transformed image. In an ideal situation, the value of this indicator should be near 100 %; average tracking error – average distance between the position of a tracked feature and the calculated position on a transformed frame. This indicator shows the accuracy of the feature detection. Large values indicate large numbers of false positive tracking or "drift" of feature points among frames. In each row with bold the four best results are selected. The winners in this competition are the FAST and the ORB algorithm. The problem of the FAST algorithm is that there will be returned a lot of feature points which also need to be processed.

Table 4

Comparison of feature detection algorithms provided by OpenCV

Name of values	FAST	GFTT	SIFT	SURF	MSER	STAR	ORB
Average number of detected keypoints	7807	577	1230	1153	230	173	692
Percent of tracked features	95.96	94.17	93.9	98.6	93.45	99.9	99.7
Average detection time in ms	13.8	32.2	502.9	364.4	117.3	27.6	25.3
Average feature point drift in pixels	0.144	0.812	0.2312	0.31	0.6875	0.826	0.0187

It was decided to use the Oriented FAST and Rotated BRIEF (ORB) [6] – algorithm because it had the best result from our point of view. A problem occurred, that there were lots of feature points in one picture which led to exponential growth of the processing time (Fig. 8). In order to not get massive amounts of features from every picture an algorithm, to sort out feature points which are very close to each other and only save one of them to the feature point list, was created.

Since feature points are unique the approach to create a three-dimensional map out of feature points was being established. On programming side this could be achieved through a three-dimensional array. The first and the second dimension can represent our map like a 2D coordinate system; for example, every index of it could be 10 meters in x and in y direction. The third dimension represents the array of feature points for every index in x and y direction.

ISSN 2076-2429 (print)
ISSN 2223-3814 (online)

Fig. 8. Feature detection with ORB algorithm. Feature matching of two images

Conclusion. As a result of the conducted studies it can be noted that creating a flight route on behalf of pictures and tracking the movement of a UAV as well as let it return home on its own, in theory and in an ideal environment, is possible. It was accomplished that a drone simulation environment was created which possesses gravity and offers a drone model. That model behaves like a real drone because it has independent motors with their own rotatory and vertical forces as well as several PID controllers for stabilization. A point to point autopilot provides simple movement from point A to point B, which is also realized through PID controllers. For the values the autopilot processes a ROS environment was created including a communication between Unity/C# and Python as well as good structured topics to exchange data. Furthermore a camera model including camera calibration has been created to provide a stable quality of pictures and also the length of pixels in real world scale. As the last point feature detection was established, to be able to locate the drone in a global context analyzing the difference between the position of feature points in subsequent pictures. This also includes creating a map of feature points which represents the route the drone is flying and makes it possible to always track the position on the way back.

Since all these features are working, the next task will be to create an interface to steer the drone manually. Using the map of feature points, an algorithm to allow the drone return to its starting point by itself needs to be built. After this first attempts to use a real drone can be made.

Література

1. Chris Patton. Photography Without Lenses. 2007. 39 p. URL: http://pinhole.stanford.edu/ images/cpbeyond.pdf (Last accessed 03.09.2017).
2. Camera Models and Fundamental Concepts Used in Geometric Computer Vision / Sturm P. et al. 2011. 187 p. URL: http://www.merl.com/publications/docs/TR2011-069.pdf (Last accessed 02.09.2017).
3. Wilhelm Burger. Zhang's Camera Calibration Algorithm:In-Depth Tutorial and Implementation. *Technical Report HGB16-05 16th May, 2016*. 2016. 55 p. URL: https://www.researchgate.net/ publication/303233579 (Last accessed 03.09.2017).
4. Juyang Weng, Paul Cohen, Marc Herniou (1992). Camera Calibration with Distortion Models and Accuracy Evaluation Distortion. *IEEE TRANSACTIONS ON PAmRN ANALYSIS AND MACHINE INTELLIGENCE*. Vol. 14, №10. 1992. P. 965–980. URL: https://www.cs.auckland.ac.nz/courses/ compsci773s1c/lectures/camera%20distortion.pdf (Last accessed 07.09.2017).
5. Oskarsson M. Two-View Orthographic Epipolar Geometry: Minimal and Optimal Solvers. *J Math Imaging Vis.* 2017. P. 1–11 DOI: 10.1007/s10851-017-0753-1 (Last accessed 05.09.2017).

6. Ethan Rublee, Vincent Rabaud, Kurt Konolige, Gary Bradski. ORB: an efficient alternative to SIFT or SURF. *Willow Garage*, Menlo Park, California. URL: http://www.willowgarage.com/sites/ default/files/orb_final.pdf (Last accessed 02.09.2017).

References

1. Chris, Patton. (2007). *Photography Without Lenses.* Retrieved from: http://pinhole.stanford.edu/ images/cpbeyond.pdf
2. Sturm, P., Ramalingam, S., Tardif, J-P., Gasparini, S., & Barreto, J. (2011). *Camera Models and Fundamental Concepts Used in Geometric Computer Vision.* Retrieved from: http://www.merl.com/ publications/docs/TR2011-069.pdf
3. Wilhelm Burger. (2016). *Zhang's Camera Calibration Algorithm: In-Depth Tutorial and Implementation*. Retrieved from: https://www.researchgate.net/publication/303233579
4. Juyang Weng, Paul Cohen, & Marc Herniou. (1992). Camera Calibration with Distortion Models and Accuracy Evaluation Distortion. *IEEE TRANSACTIONS ON PAmRN ANALYSIS AND MACHINE INTELLIGENCE*, 14, 10, 965–980. Retrieved from: https://www.cs.auckland.ac.nz/courses/ compsci773s1c/lectures/camera%20distortion.pdf
5. Oskarsson, M. (2017). Two-View Orthographic Epipolar Geometry: Minimal and Optimal Solvers. *J Math Imaging Vis*, 1–11. Retrieved from: https://doi.org/10.1007/s10851-017-0753-1
6. Ethan Rublee, Vincent Rabaud, Kurt Konolige, & Gary Bradski. ORB: an efficient alternative to SIFT or SURF. *Willow Garage*, Menlo Park, California. Retrieved from: http://www.willowgarage.com/ sites/default/files/orb_final.pdf

Received September 12, 2017

Accepted October 02, 2017