AR-Drone as a Platform for Robotic Research and Education

Tomáš Krajník, Vojtěch Vonásek, Daniel Fišer, and Jan Faigl

The Gerstner Laboratory for Intelligent Decision Making and Control
Department of Cybernetics, Faculty of Electrical Engineering
Czech Technical University in Prague
{tkrajnik,vonasek,danfis,xfaigl}@labe.felk.cvut.cz

Abstract. This paper presents the AR-Drone quadrotor helicopter as a robotic platform usable for research and education. Apart from the description of hardware and software, we discuss several issues regarding drone equipment, abilities and performance. We show, how to perform basic tasks of position stabilization, object following and autonomous navigation. Moreover, we demonstrate the drone ability to act as an external navigation system for a formation of mobile robots. To further demonstrate the drone utility for robotic research, we describe experiments in which the drone has been used. We also introduce a freely available software package, which allows researches and students to quickly overcome the initial problems and focus on more advanced issues.

1 Introduction

A quadrotor helicopter or quadcopter is an aerial vehicle propelled by four rotors. The propellers have a fixed pitch, which makes the quadcopter mechanically simpler than an ordinary helicopter. However, the quadcopter is inherently unstable, and therefore, its control is rather difficult. The progress on the field of control engineering allowed to deal with inherent instability of the quadrotors, and therefore, they have started to appear in military, security and surveillance systems.

Nowadays, the quadcopters can perform quick and complex maneuvers [1], navigate autonomously in structured [2] and unstructured [3] environments and cooperate in manipulation and transportation tasks [4]. However, the commercial quadrotor helicopters are too expensive to be used by students or small research teams. Although there exist several quadcopter toys, these are too small to carry necessary sensor equipment. In recent years, several community projects aimed to develop an affordable quadrotor helicopter have appeared [5]. However, these projects are still in progress and have not filled the gap between expensive commercial platforms and sensorless toys.

In the autumn of 2010 an affordable quadcopter, equipped with the necessary sensors and with a suitable software interface has appeared on the market. Originally intended as a high-tech toy for augmented reality games, the drone quickly caught attention of universities and research institutions, and nowadays is being used in several research projects. At the Cornell university, the AR-Drone has been used for experiments in UAV visual autonomous navigation in structured environments [6]. Moreover, machine learning approaches were applied to predict the position errors of the UAV following a desired flight path [7]. Other research groups used the drone as an experimental platform for autonomous surveillance tasks [8], human-machine interaction [9], and even as a sport assistant [10], which aids the athletes by providing them external imagery of their actions.

Starting to work with the drone might be time consuming, because one has to solve several implementation and 'low level' issues. Moreover, the drone itself is an unstable system and its control is not as easy as control of the ground robots. We have used the AR-Drone prototypes since March 2010, and therefore, we have gained experience with utilizing the drone as a platform for research and education. During the last year, we have been contacted by several students, who have been starting to use the AR-Drone in their projects. Most of them have similar problems with drone control, sensory data processing and usage of the provided software. Our aim is to help the AR-Drone users to quickly overcome these issues.

The hardware and firmware of the platform is described in the next section, which also provides basic information about the drone API. After that, we describe how we implemented the basic tasks of position control, object tracking, and autonomous flight. The following section briefly summarizes experiments the drone has been used for. The last section concludes benefits of the drone usage in robotic education and research.

2 The AR-Drone platform

In this chapter, we make a brief introduction to the AR-Drone platform. We will describe not only its hardware, but also the way it can be controlled.

2.1 Hardware

The AR-Drone (see Fig. 1) is an electrically powered quadcopter intended for augmented reality games. It consists of a carbon-fiber support structure, plastic body, four high-efficiency brushless motors, sensor and control board, two cameras and indoor and outdoor removable hulls. The control board not only ensures safety by instantly locking the propellers in case of a foreign body contact, but also assists the user with difficult maneuvers such as takeoff and landing. The drone operator can set directly its yaw, pitch, roll, and vertical speed and the control board adjusts the motor speeds to stabilize the drone at the required pose. The drone can achieve speeds over 5 m.s^{-1} and its battery provides enough energy up to 13 minutes of continuous flight.

Drone control computer is based on the ARM9 processor running at 468MHz with 128 MB of DDR RAM running at 200MHz. The manufacturer provides a software interface, which allows to communicate with the drone via an ad-hoc WiFi network. The API not only allows to set drone required state, but also provides access to preprocessed sensory measurements and images from onboard cameras.

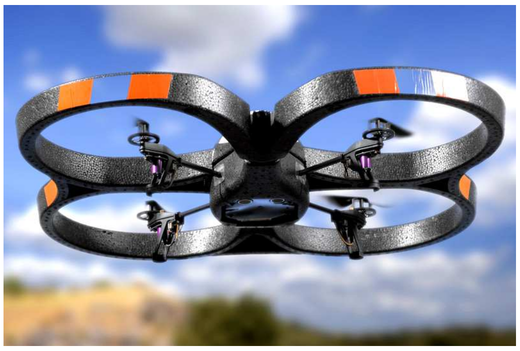


Fig. 1: The AR-Drone quadcopter

The drone sensory equipment consists of a 6-degree-of-freedom inertial measurement unit, sonar-based altimeter, and two cameras. The first camera with approximately $75° \times 60°$ field of view is aimed forward and provides 640×480 pixel color image. The second one is mounted on the bottom, provides color image with 176×144 pixels and its field of view is approximately $45° \times 35°$.

While data from the IDG-400 2-axis gyro and 3-axis accelerometer is fused to provide accurate pitch and roll, the yaw is measured by the XB-3500CV high precision gyro. The pitch and roll precision seems to be better than 0.2° and the observed yaw drift is about 12° per minute when flying and about 4° per minute when in standby.

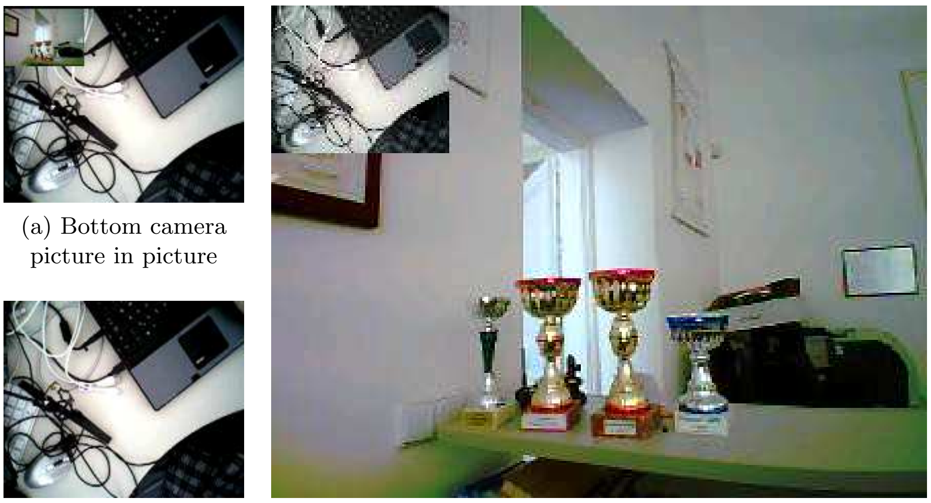
2.2 Software

The control board of the AR-Drone runs the BusyBox based GNU/Linux distribution with the 2.6.27 kernel. Internal software of the drone not only provides communication, but also takes care of the drone stabilization, and provides so-called assisted maneuvers. The bottom camera image is processed to estimate the drone speed relative to the ground, and therefore, the drone is more stable than other quadcopters.

After being switched on, an ad-hoc WiFi appears, and an external computer might connect to it using a fetched IP address from the drone DHCP server. The external computer then can start to communicate with the drone using the interface provided by the manufacturer. The interface communicates via three channels, each with a different UDP port.

Over the *command* channel, a user controls the drone, i.e., requests it to takeoff and land, change configuration of controllers, calibrate sensors, set PWM on individual motors etc. However, the most used command sets the required pitch, roll, vertical speed, and yaw rate of the internal controller. The channel receives commands at 30 Hz.

The *navdata* channel provides the drone status and preprocessed sensory data. The status indicates, whether the drone is flying, calibrating its sensors, the current type of altitude controller, which algorithms are activated etc. The sensor data contain current yaw, pitch, roll, altitude, battery state and 3D speed estimates. Both status and sensory data are updated at 30 Hz rate. Moreover, the drone can run a simple analysis of the images from the frontal camera and search for a specially designed tags in the images. In the case the tags are detected, the navdata contains estimates of their positions.


(a) Bottom camera picture in picture

(b) Bottom camera

(c) Frontal camera picture in picture

Fig. 2: Images provided by the drone in various modes of operation

.

The *stream* channel provides images from the frontal and/or bottom cameras. The frontal camera image is not provided in actual camera resolution, but it is scaled down and compressed to reduce its size and speed up its transfer over WiFi. As a result, the external computer obtains a 320×240 pixel bitmap with 16bit color depth. A slight disadvantage of the camera system is that a user cannot obtain both camera images at a time. Rather than that, the user has to choose between bottom and forward camera or go for two picture in picture modes, see Fig. 2. Switching the modes is not instant (takes $\sim$ 300 ms) and during the transition time, the provided image contains invalid data.

Since the control board is accessible by *telnet*, the drone user can log in and change settings of the onboard operating system and adjust configuration files of the drone internal controllers. Moreover, it is possible to cross-compile an application for the ARM processor and run it directly on the AR-Drone control board. In this case, one can access the drone cameras and onboard sensors directly without a delay caused by the wireless data transfer. Thus, one can achieve faster control loops and experiment with a low level control of the drone. Even when a custom application is running on the platform control board, the internal controllers, which take care of the drone stability, can be active. However, the memory and computational limits of the control board have to be taken into account when developing an application, which should run onboard the drone.

For our purposes, we have created a simple application, which uses all three aforementioned channels to acquire data, allows drone control by a wireless joystick and performs a simple image analysis. This piece of freely available software [11] serves as a base for more complex applications, which provide the drone with various degrees of autonomy. The software does not require any nonstandard libraries and works both under GNU/Linux and Windows environments.

3 Autonomous navigation

In this chapter, we will show how to implement several autonomous behaviours. We will start by a simple position control, continue with hovering over a moving object and traveling along a predefined path.

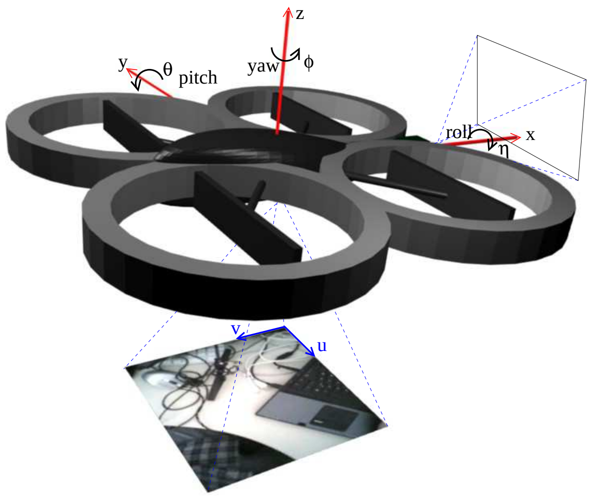


Fig. 3: Coordinate system of the drone

3.1 Dynamic model of the drone

In order to design controllers for the drone, it is desirable to know its dynamic model parameters. Instead of modeling the drone like a standard quadrotor helicopter, i.e., considering its propeller speeds as inputs and angles as outputs, we model the drone including its internal controller. Since the internal controller is able to set and keep the desired angles and vertical speed, we do not have to deal with complexity of the drone model [12]. Instead of it, we model the drone as a system, which has the desired pitch, roll, yaw rate, and vertical speed as its input, and its actual angles, speed, and position as its states, see Fig. 4.

Since the position of the drone is a pure integration of its speeds, we can further simplify the model, and consider only the yaw and speeds as its state variables. Moreover, we can consider that the forward speed is given by the drone pitch, and ignore the influence of other inputs. The same can be done for the drone side speed and roll, yaw and yaw rate, and altitude and vertical velocity. Therefore, we can decompose the drone dynamic model in four first- or second-order dynamic systems and identify their parameters separately. This decoupled model allows to design separate controllers for each such aforementioned tuple of state and input variables.

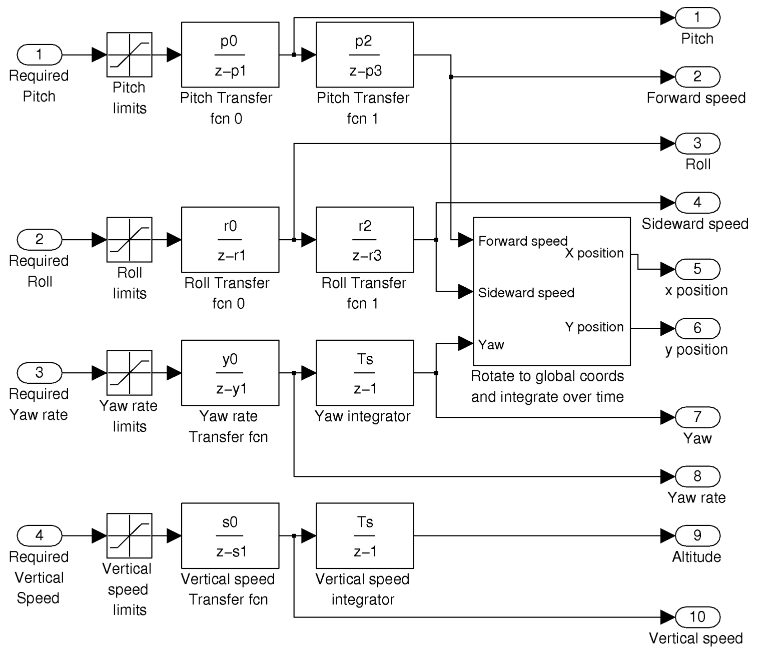

1 Required Pitch
Pitch limits
p0 / z−p1 Pitch Transfer fcn 0
p2 / z−p3 Pitch Transfer fcn 1
2 Required Roll
Roll limits
r0 / z−r1 Roll Transfer fcn 0
r2 / z−r3 Roll Transfer fcn 1
3 Required Yaw rate
Yaw rate limits
y0 / z−y1 Yaw rate Transfer fcn
Ts / z−1 Yaw integrator
4 Required Vertical Speed
Vertical speed limits
s0 / z−s1 Vertical speed Transfer fcn
Ts / z−1 Vertical speed integrator
Forward speed
Sideward speed
Yaw
X position
Y position
Rotate to global coords and integrate over time
1 Pitch
2 Forward speed
3 Roll
4 Sideward speed
5 x position
6 y position
7 Yaw
8 Yaw rate
9 Altitude
10 Vertical speed

Fig. 4: A structure of the drone model

To identify the parameters of the forward speed-pitch model, we have let the drone hover over a spot, then requested a -7.5° pitch and gathered the drone navdata. From the gathered data, we have extracted sequences of required pitch ϕ'_i, actual pitch ϕ_i, and forward velocity v_i. Using the model in Fig. 4, we established the following linear equations

$$\begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_n \end{pmatrix} = \begin{pmatrix} \phi_0 & \phi'_0 \\ \phi_1 & \phi'_1 \\ \vdots & \vdots \\ \phi_{n-1} & \phi'_{n-1} \end{pmatrix} \begin{pmatrix} p_1 \\ p_0 \end{pmatrix},$$

and calculated p_1 and p_0 by means of least squares. The parameters p_3 and p_2 can be calculated by a similar equation from the ϕ_i and v_i. To verify the model parameters, we compared the measured and simulated data, see Fig. 5.

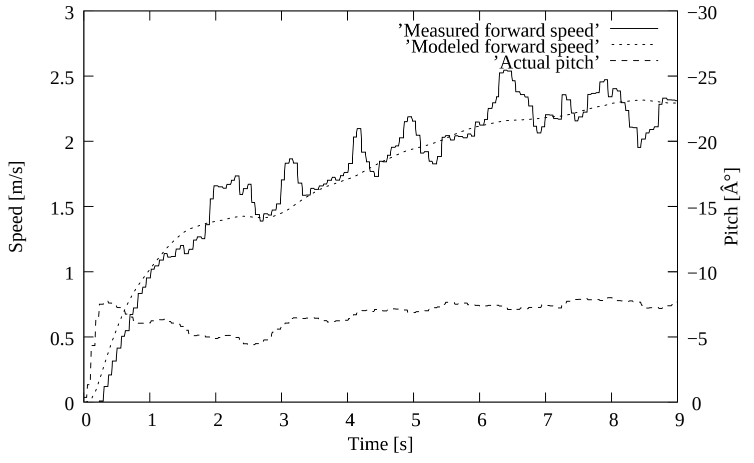


Fig. 5: Measured and simulated forward speed

To establish the remaining parameters of the dynamic model ($r_{0\ldots3}$, $y_{0,1}$, $s_{0,1}$), a similar procedure was performed.

One has to consider that the drone model parameters differ for indoor and outdoor hulls. Moreover, the parameters vary slightly for different drones as well, and therefore, it is recommended to perform the identification procedure for each drone separately.

3.2 Position control

Since the model parameters were calculated, we can start with controller design. Assume, that the drone is at a position (x, y, z), its yaw is ϕ, and we want to implement a controller, which moves it to a different position (x_d, y_d, z_d) with a yaw ϕ_d. In our drone model, neither the drone altitude z nor its yaw ϕ are influenced by other state variables, and therefore, their controllers can be designed independently.

The yaw subsystem has a very quick response, and therefore, a simple proportional controller is sufficient. Altitude dynamics is a bit slower because the drone inertia cannot be omitted. Therefore, we have implemented a simple PD controller for the purpose of altitude stabilization. The vertical speed (i.e., the differential channel) does not have to be estimated from height measurements, we can rather use the vertical speed from the *navdata* provided by the drone. Since both altitude and yaw subsystems contain a pure integrator, their controllers do not have to contain an integration channel for achieving zero control error. Therefore, the structure of both yaw and altitude controllers is quite simple, see Fig. 6.

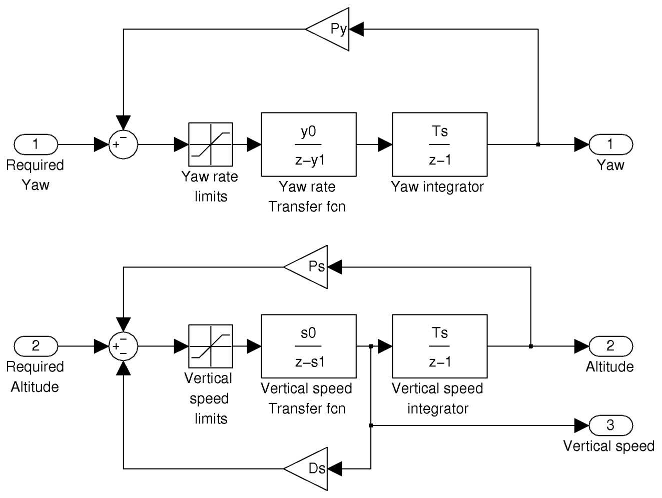


Fig. 6: Structure of the yaw and altitude controllers

When designing the position controller, we assumed, that the yaw rate (i.e., turning speed of the drone) will be small, and therefore, the pitch would influence only the forward speed and the roll only the sideward speed of the drone. To reach a desired position (x_d, y_d), we first transform it to the coordinates relative to the drone (x_r, y_r). After that, we use the x_r as an input value of the pitch controller and y_r as an input value for the roll controller. Not only have these subsystems the same structure, but also their dynamics is similar, and therefore, their controllers will be the same. Since we know the dynamics of the system and have access to the current pitch and forward speed, we can easily implement a linear controller, see Fig. 7. Note that the proportional feedback loop is implemented in the calculation of relative coordinates (x_r, y_r) from the desired and current drone position.

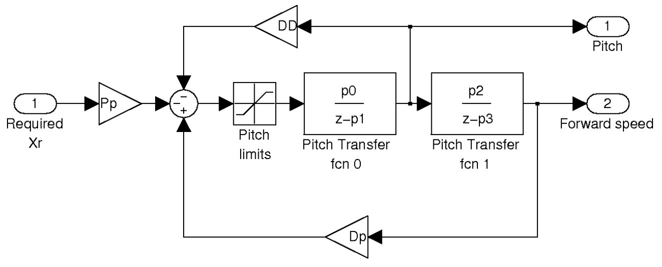


Fig. 7: Structure of the pitch (and roll) controller

Since the model parameters are known, the gains of the individual controllers can be chosen by any of the plethora methods. We have selected the controller gains using the method of pole placement [13].

With this position controller, the drone is able to quickly reach the requested position and stabilize itself without oscillations. However, there are several issues regarding the position stability. The drone position is estimated by integration of the *navdata* speeds and yaw, which are not precise. Therefore, the IMU-based position estimation is subject to drift. When flying indoors at speeds about $1-2\ m.s^{-1}$, the overall precision of the position estimation is approximately 10% of the traveled distance. Moreover, the sonar-based altimeter error is higher than 20%, and the error seems to be influenced by the type of the surface below the drone. The yaw, which is measured by integrating a gyro signal, is also subject to drift. Therefore, this kind of position estimation is not suitable for long-term autonomous operation. However, the position controller works fine in short term and is used as a base for other, more complex behaviours.

3.3 Hovering over an object

To keep the position estimation system stable, one needs an external reference. One of the most simple ways is to place a distinct pattern on the ground, and base the drone position estimation on the pattern image coordinates. This would allow not only hovering over a fixed position, but also takeoff and landing on a colored heliport, see Fig. 8. To do this, we just need to estimate the tracked object coordinates and feed them to the position controller.

First of all, we have to measure the drone bottom camera field of view and establish relation between image and real-world coordinates. Then, we have to setup the blobfinding algorithm to search for the desired color pattern.

In autonomous operation, the bottom camera image is searched for continuous blocks ‘blobs’ of a given color. The coordinates u, v of the pixel in the center of the largest blob are calculated. These coordinates are transformed to a real world (x_o, y_o) coordinates (we assume that the tracked object altitude is zero) by the equation

$$\begin{pmatrix} x_o \\ y_o \end{pmatrix} = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix} \left[\begin{pmatrix} 0 & k_u \\ k_v & 0 \end{pmatrix} \begin{pmatrix} c_u - u \\ c_v - v \end{pmatrix} + \begin{pmatrix} \sin(\theta) \\ \sin(\eta) \end{pmatrix} \right] z + \begin{pmatrix} x \\ y \end{pmatrix}, \quad (1)$$

where x, y, z, ϕ, θ, η are drone current position, height, yaw, pitch and roll, c_u and c_v are image center coordinates and k_u, k_v are camera parameters, see also Fig. 3. The k_u and k_v are approximately equal to the ratio of camera field of view to the camera resolution. Resulting object positions x_o, y_o are sent to the drone position controller.

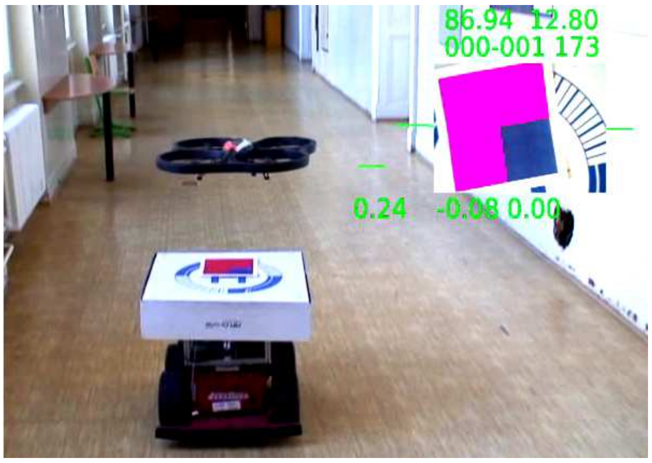


Fig. 8: AR Drone landing on a moving robot

This kind of position control works well at altitudes over 0.4 m. Below these altitudes, the area viewed by the bottom camera is small and the tracked object can be easily lost. Moreover, the air flow caused by the drone reflects from the ground and affects drone behaviour. Therefore, it is desirable to hover in higher altitudes and switch off the position control during takeoff and landing maneuvers. Our experience shows that a reasonable altitude for object tracking is over 1.5 m.

One could think that object tracking can be achieved by much simpler means than the ones described above, e.g., the required pitch and roll can be directly calculated from object image coordinates. Imagine such a controller in a situation, when an object is detected in the top part of the image and the drone is requested to move forwards. To move forward, the drone is pitched down, which causes the object to move further to the image top border, which causes even higher forward speed. This positive feedback loop produces oscillations and makes the control unstable. Even when the controller would compensate for drone pitch, tilting the drone usually causes it to lose the tracked object out of sight. Due to these facts, the described position control is more reliable than simpler methods.

3.4 Visual based navigation

Another way to localize the drone in long term is to use the forward camera and some of the monocular-based navigation methods used for ground based robots. We have implemented a map and replay method described in [14]. The method relies on the Speeded Up Robust Features (SURF) [15] extracted from the frontal camera image. In this method, the drone is first guided by a human through an environment and creates a landmark map. After that, the map is used by the drone to repeatedly travel the taught path. The path is divided in several straight segments, each with it's own submap.

When autonomously traversing a particular segment, the currently perceived image features are matched to the mapped ones to estimate the heading relative to the segment. The distance traveled along the segment is estimated purely by dead reckoning. The navigation method is provably stable for nondegenerate polygonal paths, since the heading corrections can suppress the position uncertainty originating from the dead reckoning. The advantage of the method is its simplicity — the heading is determined by a simple histogram voting procedure — and robustness.

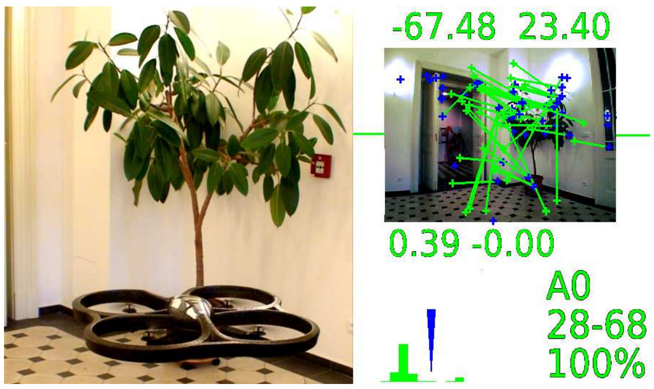


Fig. 9: Autonomous visual navigation of the drone

For the purpose of the autonomous navigation, the position control was slightly changed. When traversing a particular segment, the height control works as described in Section 3.2, the input for the yaw controller is determined from the image information, the roll position controller is disabled, and the pitch controller input equals the currently traveled segment length. Due to the low precision of the IMU based distance estimation, the drone localization error is high. However, the localization error does not diverge and is kept within sufficient limits allowing the drone to autonomously navigate along the taught path.

4 Experiments

This section briefly describes two experiments performed with the drone. Videos of these experiments are available on our youtube channel[1]. In the first experiment, the drone was used as a mobile external localization system. The second experiment was concerned with performance of the AR-Drone in an autonomous surveillance task.

4.1 Mobile localization system

Since the drone is capable to determine positions of distinct colored objects from its bottom camera image, we have used it as an external localization system. In our experiment, we tested a formation control method [16], which allows to adaptively change formation shape according to the environment constraints. Our control method is based on the leader-follower approach [17], where one of the robots (the leader) plans and executes the path, while the following robots keep a certain relative position to the leader. However, the followers might be separated from the leader by obstacles, and therefore, unable to establish their relative position accordingly. Using GPS to solve this problem would not be possible, because its accuracy is insufficient to keep a precise formation shape. To solve this, the leading robot carried the drone, which took off and provided localization when needed. Therefore, the drone was hovering above the leader and provided the followers with their positions relative to the formation leader. Using the provided and desired positions, the followers adjusted their speeds to maintain the formation.

Equation (1) shows that measurement of the drone altitude z considerably influences the results of the position estimation. Hovering over the leading robot was achieved by the method described in Section 3.3. To calculate relative follower positions, one has to measure heading of the leader. Moreover, the followers need to know their heading to compute their speed adjustments. Therefore each robot is distinguished by two rectangular areas with the same color, covering the top of it, see Fig. 10.

Since the drone altimeter is imprecise, we had to estimate the drone altitude from the leader top covering, in particular from the distance of its colored rectangles in image coordinates. This way, the altitude estimation error was about 3 % and the follower position estimation error was approximately equal to 0.05 m.

[1] http://www.youtube.com/imrfel

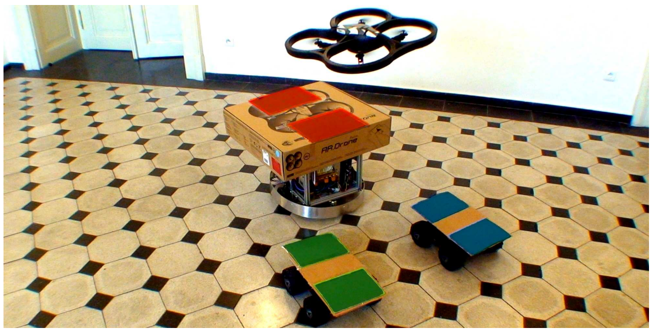


Fig. 10: The drone taking off from the formation leader

The mobile robot formation used in the experiment was composed of two followers and a leader, which carried a heliport with the drone. The following robots did not use odometry or any additional position estimation system.

4.2 Autonomous Surveillance

Consider an autonomous surveillance scenario, where the task of the drone is to monitor a set of objects of interest (OI). In particular, the drone has to fly autonomously over the objects and use its bottom camera to capture the OI images as frequently as possible. One of the possible solutions is to formulate the task as Traveling Salesman Problem [18], establish the order in which the OIs should be visited and use the navigation method described in Section 3.4 to move between the OIs.

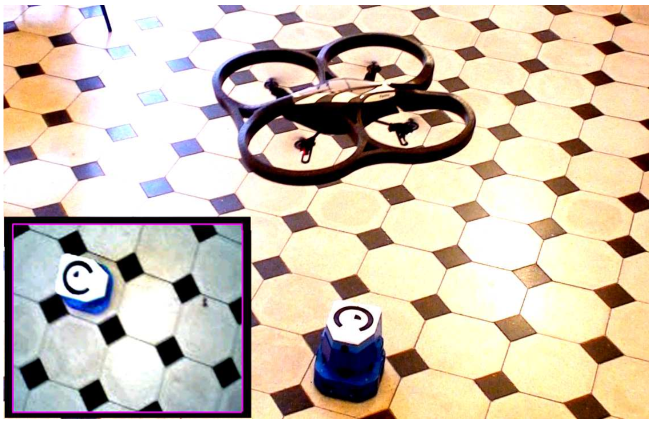


Fig. 11: The drone taking photo of the OI

However, the drone lacks precise position reference and might miss the desired OI. The frequency of goal visits then depends not only on the path length, but also as on the probability of arriving at a sufficient distance to the particular OI to take its photo. Therefore, one should plan a path over the OIs while considering the uncertainty of drone position estimation over the IOs.

As noted in Section 3.4, the drone position uncertainty is increasing in the direction of its movement and decreasing in the perpendicular direction [14]. Therefore, we have proposed to place an auxiliary waypoint before each OI and used a SOM based approach [8] to plan the sequence of waypoints the drone should pass through.

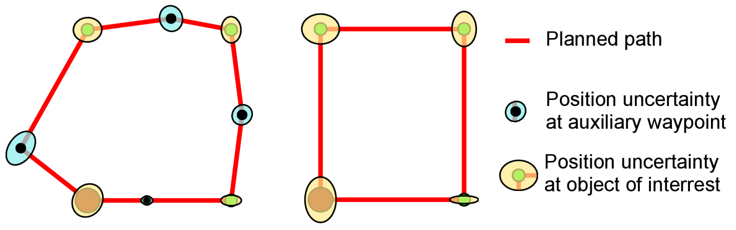

Planned path
Position uncertainty at auxiliary waypoint
Position uncertainty at object of interrest

Fig. 12: Planned paths considering and not considering the position uncertainty
.

We tested the naïve and the planning method with auxiliary waypoints for four OIs, see Fig. 12. The drone was then taught the planned paths and the success rate of goal visits was measured. The experiment showed that considering the localization uncertainty in path planning significantly increases the probability of goal visits [8].

5 Conclusion

This paper introduces the AR-Drone quadrotor helicopter as a platform utilizable in robotic research. Not only the hardware of the drone, but also its software, and communication are discussed. We described how to model this quadcopter mathematically, how to establish the model parameters, and how to use the model in position controller design. Moreover, we have shown how to implement object tracking and autonomous flight. We also presented a freely available software package, which allows to control the drone from a PC, obtain sensory data and run basic image analysis. Using this application, researchers and students can quickly overcome the initial problems and focus on more advanced issues. To further demonstrate the drone utility in robotic research, we present two experiments in which the drone has been used.

Acknowledgements. The work presented in this paper has been supported by the EU FP7-ICT project No. 216342, by the Ministry of Education of the Czech Republic grant MEB111009, and by the grants of the Czech Technical University in Prague No. SGS10/185 and No. SGS10/195

References

1. Mellinger, D., Michael, N., Kumar, V.: Trajectory generation and control for precise aggressive maneuvers with quadrotors. In: International Symposium on Experimental Robotics, Delhi, India (2010)
2. Achtelik, M., Bachrach, A., He, R., Prentice, S., Roy, N.: Stereo vision and laser odometry for autonomous helicopters in GPS-denied indoor environments. In: SPIE Unmanned Systems Technology XI. Volume 7332., Orlando, F (2009)
3. Blöandsch, M., Weiss, S., Scaramuzza, D., Siegwart, R.: Vision based MAV navigation in unknown and unstructured environments. In: IEEE Int. Conf. on Robotics and Automation. (2010) 21 –28
4. Michael, N., Fink, J., Kumar, V.: Cooperative manipulation and transportation with aerial robots. Autonomous Robots **30** (2011) 73–86
5. Multicopter: List of helicopter projects (2011) `http://multicopter.org/wiki/`.
6. Bills, C., Chen, J., Saxena, A.: Autonomous MAV flight in indoor environments using single image perspective cues. In: IEEE Int. Conf. on Robotics and Automation. (2011)
7. Bills, C., Yosinski, J.: MAV stabilization using machine learning and onboard sensors. Technical Report CS6780, Cornell University (2010)
8. Faigl, J., Krajník, T., Vonásek, V., Přeučil, L.: Surveillance planning with localization uncertainty for mobile robots. In: 3rd Israeli Conference on Robotics. (2010)
9. Ng, W.S., Sharlin, E.: Collocated interaction with flying robots. Technical Report 2011-998-10, Dept. of Computer Science, University of Calgary, Canada (2011)
10. Higuchi, K., Shimada, T., Rekimoto, J.: Flying sports assistant: external visual imagery representation for sports training. In: 2nd Augmented Human International Conference, New York, NY, USA, ACM (2011) 7:1–7:4
11. Krajník, T.: Simple 'getting started' applications for AR-drone (2011) `http://labe.felk.cvut.cz/~tkrajnik/ardrone`.
12. Šolc, F.: Modelling and control of a quadrocopter. Advanced in Military Technology **1** (2007) 29–38
13. Kailath, T.: Linear Systems. Prentice Hall (1980)
14. Krajník, T., Faigl, J., Vonásek, V., Košnar, K., Kulich, M., Přeučil, L.: Simple yet stable bearing-only navigation. Journal of Field Robotics **27** (2010) 511–533
15. Bay, H., Ess, A., Tuytelaars, T., Van Gool, L.: Speeded-Up Robust Features (SURF). Computer Vision and Image Understanding **110** (2008) 346–359
16. Saska, M., Vonásek, V., Přeučil, L.: Roads sweeping by unmanned multi-vehicle formations. In: IEEE Int. Conf. on Robotics and Automation. (2011)
17. Consolini, L., Morbidi, F., Prattichizzo, D., Tosques, M.: Leader-follower formation control of nonholonomic mobile robots with input constraints. Automatica **44** (2008) 1343–1349
18. Spitz, S.N., Requicha, A.A.G.: Multiple-Goals Path Planning for Coordinate Measuring Machines. In: IEEE Int. Conf. on Robotics and Automation. (2000) 2322–2327