

# A Controller for Autonomous Landing of AR.Drone

Roman Barták, Andrej Hraško, David Obdržálek

Charles University in Prague, Faculty of Mathematics and Physics, Malostranské nám. 25, Praha, Czech Republic  
E-mail: bartak@ktiml.mff.cuni.cz, andrej@hrasko.eu, david.obdrzalek@mff.cuni.cz

**Abstract:** In recent years, many so called “robotic toys” appeared at the market. Though these devices are called toys and they are primarily intended for games, they provide a set of sensors and actuators that can be controlled from a computer. Hence these robotic devices may serve as non-expensive research tools, in particular for developing advanced software for controlling them. This paper describes a controller for landing of an AR.Drone, a quadricopter marketed by Parrot Inc. This controller is designed for fully autonomous landing to a given visual pattern and for hovering above the pattern. We will show how to collect input information for the controller and how to use it in a classical PID controller.

**Key Words:** AR.Drone, Landing, PID Controller

## 1 INTRODUCTION

Recent advancement of robotic research brought non-expensive hardware with sophisticated sensors and flexible motoric capabilities. Today robots are ready for performing advanced operations, but it seems that their major limitations lie in software. In our research, we are using an AR.Drone – a quadricopter capable of flying in any 3D direction (Figure 1). This drone has no manipulators; it is basically a flying robot with several sensors including two cameras. Moreover, an AR.Drone has some basic stabilization mechanisms and it can be controlled by setting pitch, roll, and yaw rather than controlling the speed of its rotors. This makes it easier to design software for AR.Drones as programmers can focus on higher levels of control. Still the quadricopter is performing in a real-life environment that is dynamic, continuous, partially observable, and stochastic [8]. So the software must run in real time, the environment is not always as the agent perceives it, and finally, the things do not always work as the agent intends.



Fig 1. AR.Drone by Parrot Inc.

In this paper we will describe a controller for autonomous landing of an AR.Drone. We will show how information about the landing point, which is obtained by analyzing

pictures from the drone camera, is transformed to distance measures used as an input for a PID controller. We will also present an application developed to demonstrate the proposed approach.

The paper is organized as follows. We will first describe hardware of the AR.Drone platform and the ways it communicates with a computer connected via WiFi. After that we will propose a method for measuring distance between the drone and the landing point. The next section will be devoted to the description of used PID controller. Presentation of the DroneLander – an application to present the proposed techniques in practice – and a summary of experimental results will conclude the paper.

## 2 AR.DRONE PLATFORM

AR.Drone by Parrot Inc. [7] is a high-tech flying toy (Figure 1) that can be used for augmented-reality games. Technically, it is a quadricopter with sensors and a controller. As this device is very light (360-400 grams) and therefore quite prone to wind disturbances, it is better suited for indoor environments. To a certain extent, it can operate outdoors as well. Its main battery allows operation for about 13 minutes. For our work, we used the original AR.Drone, but there exists a newer version AR.Drone 2 with better parameters. We describe here the selected parameters of the original AR.Drone as they influenced the decisions done.

AR.Drone is controlled by a 32-bit 468 MHz ARM9 RISC processor with 128MB DDR-RAM running at 200MHz. This processing architecture controls the basic operations of the drone including stabilization. It is possible to install own programs there but due to a limited computing power we have decided to use an external computer.

The core set of sensors consists of a 3-axis accelerometer, a 1-axis gyroscope, a 2-axis gyroscope, an ultrasound sensor (sonar), and a vertical downside camera. The 1-axis gyroscope measures yaw with the error  $12^\circ$  per minute during the flight or  $4^\circ$  per minute in the steady state. The 2-axis gyroscope measures pitch and roll with the error  $0.2^\circ$  per minute (see Figure 2 for the explanation of the yaw, roll, and pitch notions).

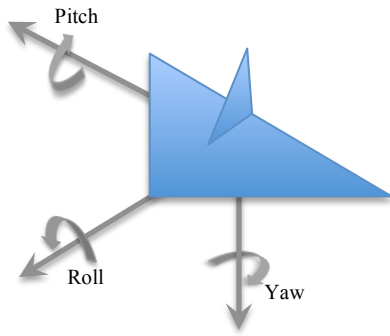


Fig 2. Roll, yaw, and pitch.

The ultrasonic sensor measures the altitude of the drone in the range 20-600 cm. The CMOS vertical camera directs downward; it has a resolution  $176 \times 144$  pixels corresponding to viewing angles  $45^\circ \times 35^\circ$  and provides a video-stream with 60 frames per second. The system uses this camera to estimate the speed of the drone via measuring the optical flow. It is also used for calibration of other sensors. There is a higher resolution horizontal camera too, but we will not use it for landing.

The quadricopter can be controlled via WiFi from an external device such as a computer. All the communication is done using three UDP channels. A *Command channel* allows sending commands to the drone. The device accepts the following commands with frequency 30Hz: takeoff, land, set limits (maximal declination, speed, and altitude), calibrate sensors, swap cameras (see below), set rotation speeds of rotors, set pitch, roll, yaw, and vertical speed. A *NavData channel* provides information about the drone state again with frequency 30Hz. The following information is provided: drone state (flying, steady, landing, takeoff, calibration, booting), sensor data (current pitch, roll, yaw, altitude, battery level, and speed in all axes). Finally, a *Stream channel* provides visual information from the cameras. Due to a limited bandwidth, only view from a single camera is available (or *picture-in-picture*).

Although the AR.Drone has a native command to land, it basically means go down without any control where to go. Therefore we decided to extend this functionality by defining a visual landing pattern and controlling the drone to land exactly at a given location.

### 3 MEASURING DISTANCE TO THE GOAL POSITION

To control the drone we first need to know its exact position with respect to the goal position. The difference between the current position and the goal position is then fed to a controller that, briefly speaking, minimizes this difference by controlling drone actions (see the next section). There exist many approaches for localizing drones. Some of them are using external navigation or specific sensors such as GPS; these are not appropriate for our purposes as AR.Drone does not use such sensors and it is mainly operating indoors. Others are based on purely visual navigation for example using an H-shaped landing pattern [10] or a specific circular pattern [5]. These seem more suitable for AR-Drone, but as they require better

visual sensors and they do not support horizontal orientation, we decided for another approach. Our motivation is exploiting an easy-to-made and customizable landing pattern that makes target localization reliable.

For landing as well as for hovering above the landing position we need to know where the landing point is located. We use a specific landing graphical pattern consisting of two colored discs. There are several reasons for choosing this pattern. First, it uniquely defines the landing point – we use the point between the discs (marked by a cross in Figure 3) – as well as the possible orientation. Second, it is easy and robust to identify the colored circles in the picture using the blob detection algorithm [9] and some simple picture pre-processing techniques. We gave the technical details of this method and its comparison with other methods in the paper [1].

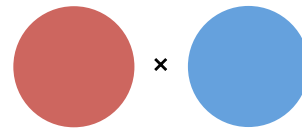


Fig 3. Landing pattern consisting of two colored discs (the cross identifies the point that we use for landing).

Obviously, to detect the landing pattern we use the bottom camera of AR.Drone, which is the main source of localization information for us. The picture processing software *OpenCV* [6] is used to identify the landing point in the picture, but we still need to convert this information to metric information about the distance between the drone and the landing point. In this section we shall describe the method to measure that distance.

Basically, we measure the distance using four attributes in the coordinate system: altitude (vertical distance in meters), front and side distances (horizontal distance in meters), and rotation (Figure 4). The information about altitude and rotation is given directly by the sensors; in particular the ultrasound sensor provides altitude in meters and the gyroscope provides yaw in degrees. It remains to compute the horizontal distances that are not directly accessible from the sensors. This will be done by analyzing the position of the landing point in the picture captured by the bottom camera, using information about altitude, and applying some trigonometry.

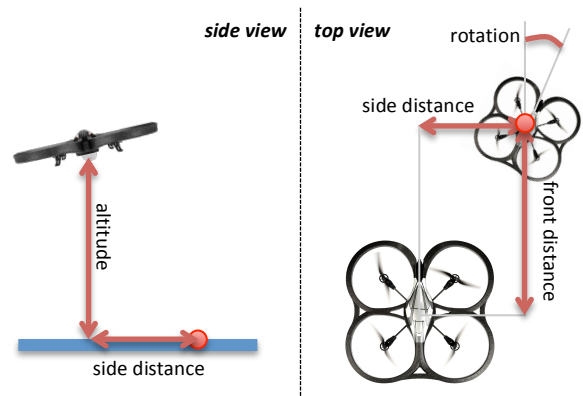


Fig 4. Relative position of drone and target.

Figure 5 shows the scheme of the situation that is used to compute the horizontal distances. Drone camera is at point D right above the point V; T is the target point;  $C_1$  and  $C_2$  are the edges of the area visible by the camera, and  $x$  is the altitude measured by the ultrasound sensor. In practice we use  $x$  to approximate the real altitude  $v$  as the angle  $\alpha$  is usually small and the ultrasound sensor has some small error anyway. Note however, that the camera does not provide the other measures directly; it gives a picture as shown in Figure 5 (top left). Points T, S,  $C_1$ ,  $C_2$  are seen as  $T'$ ,  $S'$ ,  $C'_1$ ,  $C'_2$  in the camera. We compute the real distance in meters between T and V using the formula:

$$dist(T,V) = v \cdot \tan(\alpha+\beta). \quad (1)$$

The ultrasound sonar gives altitude  $x$  (that approximates  $v$ ) in meters (with centimeter accuracy). Angle  $\alpha$  is also known from the sensors – it is either pitch for the forward/backward distance or roll for the left/right distance. Angle  $\beta$  is a relative angle between the actual drone position and the target so it proportionally corresponds to the distance between  $T'$  and  $S'$ . We measure this distance in pixels from the camera picture. This is not the real distance of points but the distance of their projections in the camera picture. We already know the pixel distance between points  $C'_1$  and  $S'$  (which is defined by the camera resolution) and the angle  $\gamma$  (view angle) from the camera properties ( $\pm 72$  pixels and  $\pm 17.5^\circ$  for the longitudinal axis,  $\pm 88$  pixels and  $\pm 22.5^\circ$  for the lateral axis). Hence can compute the angle  $\beta$  using the formula:

$$\beta = \arctan(\tan(\gamma/2) * pixel\_dist(T',S') / pixel\_dist(C'_1,S')). \quad (2)$$

For the forward/backward distance, the formula looks like:

$$\beta = \arctan(\tan(17.5) * vert\_pixel\_dist(T',S') / 72). \quad (2a)$$

For the side distance, the formula looks like:

$$\beta = \arctan(\tan(22.5) * horiz\_pixel\_dist(T',S') / 88). \quad (2b)$$

Since we have computed the real relative position of the target (defined by a distances/angle in meters/degrees between target and drone for each axis) we can make the drone to approach the target using this information – to minimize the distances/angles (called *errors*) to a zero.

## 4 DRONE CONTROLLER

During the landing procedure, we control the drone by repeating the following steps:

1. identifying the landing point in a video frame,
2. finding the error – the real distance in meters between the current position and the target position,
3. feeding the error to a controller,
4. using the controller output to move the drone.

As the main control mechanism, the *proportional–integrative–derivative (PID) controllers* are used. A PID controller is a generic control loop feedback mechanism widely used in industrial control systems [4]. Its input is a required value (attribute or property) of the controlled object; during the operation, the PID controller measures this value and based on the difference between it and the required value (the error) it changes the actuation.

We use four PID controllers to control the drone, namely for forward/backward movement, for left/right movement,

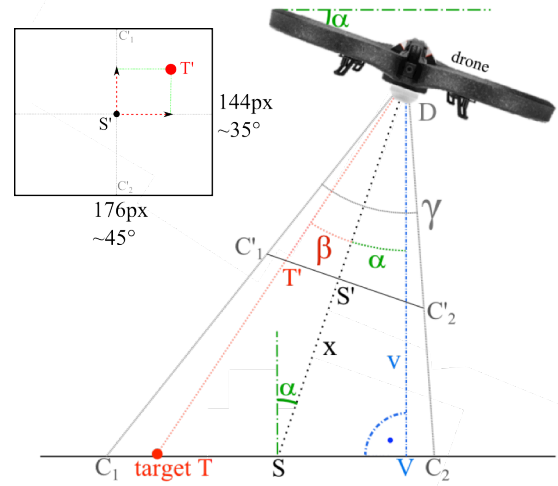


Fig 5. A schematic drone position (a side view) and the view from the bottom camera (top left). Points T, S,  $C_1$ ,  $C_2$  are seen as  $T'$ ,  $S'$ ,  $C'_1$ ,  $C'_2$ .

for rotation, and for altitude. The required value input for one PID controller is the target point  $x$  coordinate and the output is directly used for the low-level drone interface as pitch and so on: the target point  $y$  coordinate is transformed by the respective controller to roll, the rotation to yaw and the altitude to vertical speed, respectively. In the previous section, we described how to compute the errors for all these controllers. The computation procedures behind all these controllers are identical; the only difference is in the setting of proportional, integral and derivative gain constants for the four different controllers. Even these constants were tuned experimentally; they result in the PID controllers having sufficient effect on the drone flight. Note also that we are not directly controlling the rotation speeds of propellers; instead, the controllers output required pitch, roll, yaw, and the vertical speed according to the errors on their respective inputs. These outputs are then fed to the drone low-level control software described in Section 2, which results in the drone moving in the requested direction.

As mentioned above, the minimal altitude recognized by the ultrasonic sensor is about 25 cm. At this altitude the landing target is hardly visible by the bottom camera as the target pattern is too close to the camera. Also when light drones are close to the surface, their own propellers cause a lot of turbulence and the drones become very hard to control at low altitudes. Therefore we stop the landing procedure at the altitude of 30 cm where we switch off the motors. Hence this final stage of landing resembles a controlled fall. If one switches the drone motors at this low level, it does not usually impose any danger of breaking the drone structure. That is because the drone does not fall from high altitude and because the drone chassis is usually constructed shockproof enough to withstand exactly such kind of landing.

### 4.1 Unexpected Events

It may happen that the target is not identified in the current video frame. There can be several reasons for that situation including a noise in the picture causing the blob detection algorithm to fail and unexpected movement of the drone

(user intervention, wind, turbulences etc.). In the default mode, we simply assume that the target is still at the original position (although temporarily invisible) and we move the drone in the direction of the last time seen position of the target. However, if the target is not identified in ten consecutive video frames, we interrupt the landing procedure and switch the drone to the hovering mode to prevent unwanted movement. This approach recovers easily from mistakes in the pattern recognition procedure and ensures that the drone moves continuously.

For the second situation when the drone moves inappropriately to the intentions, we can optionally use a more proactive approach. Based on data from the accelerometer, we can detect the drone movement difference (measure passed distance by the accelerometer, this is also called accelerometer based odometry). We can compute the distance from the last known position of the target by summing all the acceleration values for each axis since the target got lost and adding this summations to the appropriate last known target position abscissa. By this information we have computed the estimated location of the target even if this location is outside the visibility area of the bottom camera. If such situation happens, we move the drone towards the expected location. However, if the target does not appear in the camera view according to the estimated location where it should already be visible, we interrupt this mode and switch the drone to the hovering mode to prevent unwanted moves like in the first case. The accelerometer can be used to balance sudden changes of drone location, but due to its nature, it cannot provide accurate measurements for a long time and hence it is used just as a short-term correction facility.

## 4.2 PID controller

A PID controller (see Figure 6) is a control loop feedback mechanism with specifically defined *transfer function* – a function that processes the input values to the output values in every cycle of the loop while aiming for minimizing the error. In our case the input is the computed error (distance/angle in meters/degrees, see the Section 3) and the output is a control value for the drone low-level controller. A point where the error of the system is minimal is called a *setpoint* and represents the target the controller wants to reach (and also the input value for the controller). The transfer function is divided to three parts – the so called proportional, integrative and derivative parts. Input of each part is the same as the input of the whole transfer function and the output of the transfer function is the sum of outputs of the three individual parts. It should also be noted that the input and output values do not necessarily have to have the same units; for example, in our case the input is the error of the current control system value but the output is the requested pitch, roll, yaw, and vertical speed as explained earlier in this section.

The output of the proportional part is directly dependent on its input multiplied by the constant  $P$  (*proportional gain*). We can see that this output value will always move the system in a direction towards the setpoint. The power of this movement depends on the value of the error and on the value of the constant  $P$ . When using this part on systems which are affected by inertia forces, there is a

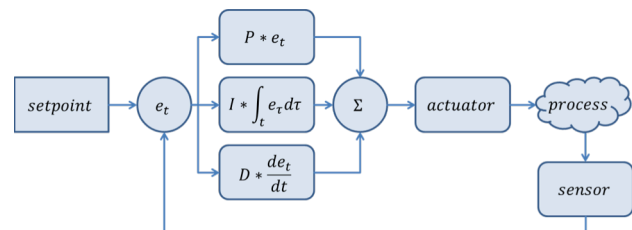


Fig 6. PID controller.

danger of reaching the setpoint but due to these inertia forces overshooting and departing it again, then stopping, returning and so on. That can lead in a long term to oscillation around the setpoint.

The output of the integrative part is the sum of all errors since the start multiplied by the constant  $I$  (*reset*). We can see that the magnitude of the output of this part grows over time if the error is not zero. This is useful for example in situations when the system can't reach the setpoint because of external impact. When the error does not minimize over time, the power of the movement generated by this part increases and consecutively can rise high enough to overcome the blocking forces.

The output of the derivative part is the magnitude of the error change (current cycle error value compared to the previous cycle error value) multiplied by the constant  $D$  (*rate*). The power of this movement depends on the speed of the system towards the setpoint ("velocity made good") and on the value of the constant  $D$ . Obviously using the output value of this part we can determine that the system is approaching the setpoint too quickly. In such case, we can start to slow down, which may prevent the unwanted oscillation around the setpoint as mentioned above. For example when the drone is near the target, but it still has high velocity, the result of the proportional part will be near the zero, but the result of the derivative part will be a higher value. The sum of these two values will be sent to the drone and will cause the breaking movement.

The outputs of the controllers are values saying how much percent of the maximum allowed tilt/vertical speed/angular speed the drone should execute. Simplified pseudo code of the PID algorithm is as follows:

```

do forever:
  read(current_position);
  current_error = setpoint - current_position;
  sum_error = sum_error + current_error;
  output = P*current_error + I*sum_error +
          D*(current_error - last_error);
  last_error = current_error;
  
```

When the constants  $P$ ,  $I$ , and  $D$  are set correctly and therefore each part has appropriate impact to controller output, the PID controller moves the drone towards the setpoint flawlessly and quickly without unnecessary movements and oscillations and stops and stays over it even when the flying conditions are not perfect.

In high abstraction we can imagine a use of the PID controller as an impact of a conjunction of three forces to the drone – each with unique characteristic. The first force tends to move the drone always directly to the target, the second force helps to reach the target when the difference between the position and the target is small and the third force slows down the drone as it approaches the target.

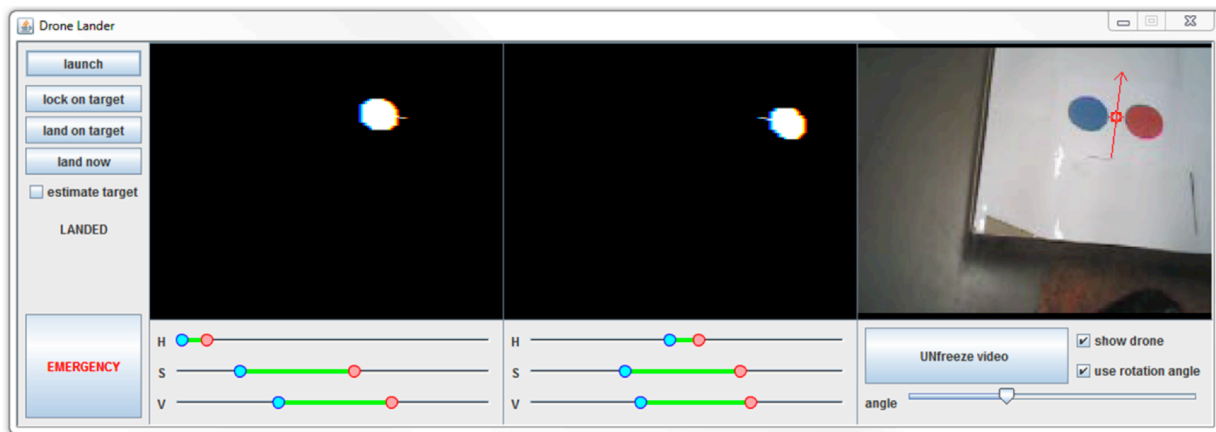


Fig 7. DroneLander user interface

### 4.3 Improving the controller

The outputs from the PID controllers may be post-processed before sending them to the drone. In our code, we have used for example the following improvements which do not change the overall operation but which fine-tune the flight:

```
yaw_out = 10 * yaw / max(horiz_dist, 0.1);
```

This makes the angular rotation speed dependent on the horizontal distance from the target. If the horizontal distance is higher than 0.1 meters, the angular rotation speed is decreased proportionally. Using this modification decreases the probability of losing the target while making risky moves such as rotation in situations when the target is near the middle of camera view. Another refinement concerns the vertical (altitude) speed:

```
alt_speed_out = 5 * alt_speed / max(angle, 5);
```

This decreases the altitude speed when the difference from the angle we want to reach is bigger than 5 degrees. This formula leads to slowdown of the drone so that the drone has more time to rotate and reach the required angle. These optimizations are used to enhance the stability and reliability of the landing procedure.

When the target is not recognized successfully, several problems may arise. When a wrong place is recognized as a target or the target is not where it was supposed to be (e.g. while rediscovering the target after inaccurate estimation), this target position change is fed to the controller. Since this wrong position might be far away from the last recognized position, the target shift can be interpreted by the controller as a jump. Even if this jump is not real, output of the derivative part of the controller will be high (because of the big change of the target position). Consecutively, the overall output of the controller would be very high which would result in rapid move of the drone. Such movements may be very dangerous for the system stability as such and transitively also for the chance of successful landing. This problem can be eliminated by setting a threshold for the maximum target position change speed (for example by limiting the maximum allowed value of the derivative part and/or by limiting the maximum allowed distance of two successive target positions) and therefore restricting the impact of the controller to the drone in such moments.

Secondly, the distance of the drone from the target is computed using data acquired from the sensors. Since the sensors are real-world sensors and not precise theoretical model sensors, their output is noisy (and may be in principle noisy with different deviations in every measurement). These deviations may produce incorrect error values used as inputs for the control loop, which may then result in dangerous jerky movements of the drone. This problem can be often eliminated by very simple filtering of the sensor data, for example by using floating average over a small number of consecutive measurements or using a Kalman filter if the characteristics of the variance of a measured values are known.

## 5 USER INTERACTION

For controlling the drone, we use the *ControlTower* application [3] that we extended by a new module named *DroneLander*. This module is invoked from *ControlTower* and provides its own user interface for defining the landing pattern and for basic control of the drone (buttons on the very left of Figure 7).

*DroneLander* uses the bottom camera of AR.Drone as the major information source for landing. The user can freeze a camera picture and set the HSV (hue, saturation, and value/brightness) parameters to identify two color blobs in the picture (left and center panels at Figure 7). Anytime when the software recognizes these two blobs in the video stream, it displays a red circle between them that marks the landing point (see the right most panel in Figure 7). The user can also optionally select the horizontal landing angle (yaw) of the drone marked by an arrow with respect to these two blobs (if not activated then the drone simply lands with any orientation).

The flight of a drone can be controlled manually through the keyboard or a game controller via the *ControlTower* application. When the user (pilot) spots the landing pattern in the picture from the bottom camera, it is possible to start the landing process. We provide two options for using the landing pattern. Either the drone lands onto the pattern (*land on target*) or the drone stays steadily directly above the pattern at the current height and at the required angle (if activated) (*lock on target*). The unexpected drone movements caused by external forces (wind, turbulence, user interaction etc.) are corrected by the landing software

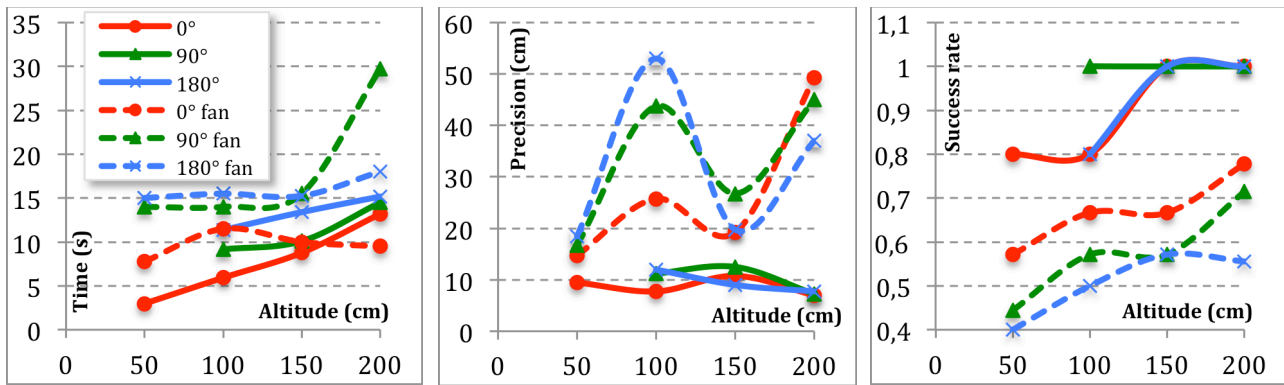


Fig 8. Dependence of landing time (left), precision (middle), and reliability (right) on the initial altitude (50, 100, 150, 200 cm) and orientation relatively to the target (0°, 90°, 180°).

via the implemented controller. When the landing pattern disappears from the camera view, the system has a simple function of trying to find the pattern by going in the direction where the pattern was lost (*estimate target*).

## 6 EXPERIMENTAL RESULTS

We evaluated the proposed autonomous landing software experimentally by measuring reliability, precision, and speed of the landing process under different conditions. In particular, we did two sets of experiments. In one set, there was no external disturbance of the landing process beyond the classical environmental conditions in the room. In the second set, we added a fan at the altitude of 50 cm and directed it towards the landing area in the distance about 150 cm. The motivation was to see how the drone controller balances external interference with the landing process. All experiments were done indoors; we placed the drone right above the landing area at different altitudes (50-200 cm) and with different rotation with respect to the landing pattern (0°, 90°, 180°). For each combination of parameters, we repeated the landing process until we got four successful landings. Then we measured the *success rate* as  $4/\#\text{landings}$ . For the successful landings we measured the *time to land* (in seconds) and the *distance* of the drone (its bottom camera) from the landing point (in centimeters).

Figure 8 shows the results as a function of the starting altitude. We differentiate between the initial orientations and having the fan switched on and off. The results confirm the expected behavior. Time to land increases with the increased altitude and with the increased initial rotation, as the drone needs to do more operations to land. When the fan is on (marked as *fan* in the figure), the landing time is also longer, which is expected. Adding wind also decreases significantly the success rate and the precision of landing, which is again an expected result. The reason is that AR.Drone is very light and it is easy to be blown away. This is especially critical close to the landing target as the drone may easily lose the target from the camera view.

In summary, under the normal conditions, the drone is landing reliably and with good precision. Under the windy

conditions, that are however not expected indoors, the success of landing decreases significantly. Using heavier drones would probably make it easier to resist wind.

## 7 CONCLUSIONS

Autonomous landing is one of the key operations of autonomous flying drones. It is especially important to use a flexible approach that does not require expensive ground equipment. In this paper we described a mechanism for fully autonomous landing of AR.Drone that uses a simple and easy-to-made graphical pattern to identify the landing area. We use only the existing sensors in AR.Drone including the bottom camera. The preliminary experimental results showed that the landing procedure is reliable under normal conditions.

## REFERENCES

- [1] R. Barták, A. Hraško, D. Obdržálek, On Autonomous Landing of AR.Drone: Hands-on Experience. In Proceedings of FLAIRS 2014, AAAI Press, 2014.
- [2] R. Brunelli, *Template Matching Techniques in Computer Vision: Theory and Practice*, Wiley, 2009.
- [3] *ControlTower*. Accessed March 10, 2014. <https://code.google.com/p/javadrone/wiki/ControlTower>.
- [4] M. King. *Process Control: A Practical Approach*. Chichester, UK: John Wiley & Sons Ltd., 2010.
- [5] S. Lange, N. Sünderhauf, P. Protzel, A Vision Based Onboard Approach for Landing and Position Control of an Autonomous Multirotor UAV in GPS-Denied Environments. *International Conference on Advanced Robotics*, pp. 1-6, 2009.
- [6] *OpenCV*. Accessed March 10, 2014. <http://opencv.org/>.
- [7] *Parrot Inc., AR.Drone 2.0*. Accessed March 10, 2014. <http://ardrone2.parrot.com/>.
- [8] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Third Edition. Prentice Hall, 2010.
- [9] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer Verlag, 2010.
- [10] S. Yang, S.A. Scherer, A. Zell, An Onboard Monocular Vision System for Autonomous Takeoff, Hovering and Landing of a Micro Aerial Vehicle. *Journal of Intelligent & Robotic Systems*, Volume 69, Issue 1-4, pp. 499-515, 2013.