J Intell Robot Syst (2014) 73:401–412
DOI 10.1007/s10846-013-9969-5

Framework for Autonomous On-board Navigation with the AR.Drone

Jacobo Jiménez Lugo · Andreas Zell

Received: 31 August 2013 / Accepted: 17 September 2013 / Published online: 19 October 2013

Abstract We present a framework for autonomous flying using the AR.Drone low cost quadrotor. The system performs all sensing and computations on-board, making the system independent of any base station or remote control. High level navigation, computer vision and control tasks are carried out in an external processing unit that steers the vehicle to a desired location. We experimentally demonstrate the properties and capabilities of three systems to autonomously following several trajectory patterns, visually estimate its position and detecting and following a person and evaluate the performance of the systems.

Keywords Micro aerial vehicle · Computer vision · Low cost · Pose estimation · Autonomous navigation · Quadrotor

1 Introduction

The popularity of the research on micro unmanned aerial vehicles has increased in the recent years. Quadrotors are a very popular choice among MAV platforms due to their robustness, mechanical simplicity, low weight and small size. Their agility and maneuverability have been demonstrated through triple flips [1], ball juggling [2] and aggressive maneuvering through windows [3]. These characteristics make the quadrotor an ideal platform for operations such as exploration, surveillance, search and rescue or even artistic performances.

Several quadrotor platforms are available for the research community. Commercial quadrotors from companies like Ascending Technologies[1] or Microdrones[2] are among the most frequently used in research due to their robustness, though they are expensive and the firmware is not always customizable. The other option for researchers is to develop their own platforms according to the application's needs. This idea was exploited in several projects, for example, the Stanford's STARMAC quadrotor testbed [4] created for multi agent experimentation or the pixhawk project from ETH Zurich [5] designed for on-board computer vision, which features a single board computer with a multi-core processor. Using an already available flight controller boards can speed up the process of building a quadrotor. Lim et al.

J. Jiménez Lugo (✉) · A. Zell
Cognitive Systems, University of Tübingen, Tübingen, Germany
e-mail: jacobo.jimenez@uni-tuebingen.de

A. Zell
e-mail: andreas.zell@uni-tuebingen.de

[1]www.asctec.de
[2]www.microdrones.com

[6] provide an overview of the characteristics of many open source platforms.

However, building a platform can be expensive and requires time and knowledge to design a flight controller. Low cost solutions are favored in scenarios with a high risk of losing the vehicle. For example, Tayebi et al. [7] exploited low cost gyroscopes for attitude estimation. Low cost sensors can be used for autonomous navigation of quadrotors. The infrared camera of the Nintendo Wii remote controller was used for autonomous take off and landing on a moving ground vehicle [8] and for following another quadrotor [9]. The camera tracked active markers on the targets to estimate the relative position of the quadrotor.

In 2010, the french company Parrot launched the AR.Drone, a $300 quadrotor that was developed for the mass market of video games. It quickly attracted the attention of the research community with features such as on-board cameras and stable hovering. The AR.Drone has become a popular platform for research and education. It has been used for object following, position stabilization and autonomous navigation [10]. Faigl et al. [11] used the AR.Drone for surveillance. The AR.Drone's camera was used to learn a feature map of a determined area. In subsequent flights the feature map was used together with the AR.Drone's odometry to navigate through the area. Bills et al. [12] classified indoor environments based on images from the AR.Drone. They used visual cues inherent in the environment to fly through it. Engel et al. [13] demonstrated figure flying using images from the AR.Drone as an input for the parallel tracking and mapping (PTAM) algorithm [14]. The output of PTAM was fused with the vehicle's inertial measurement unit (IMU) data to produce a more accurate position estimate. The mentioned AR.Drone applications depend on a base station that extracts relevant information for path planning, navigation, control algorithms and generate a steering command. The base station communicates with the AR.Drone through a wireless network, which limits the working distance of the system and introduces a delay in the information exchange of sensor data and control commands.

The motivation behind this work is to develop a framework to extend the AR.Drone. This framework will allow interaction with other sensors, additional processing for high level navigation tasks and enable autonomous flying. Additionally we show that figure flying and on-board computer vision are possible with a low-cost system just over $300. The system does not need a base station, relying only on on-board sensing and processing, thus minimizing the communication delays.

We present a framework for autonomous navigation of the AR.Drone with minimal hardware and software additions. The framework can be easily customized or upgraded. We implemented high level control and trajectory planning tasks for autonomous figure flying. This paper is an extended version of [15]. We additionally demonstrate two more systems with different hardware and applications. One that visually estimates the vehicle's distance to a known pattern and the other one that detects and follows a person, both using computer vision techniques with on-board processing only. We evaluated our system in a series of indoor and outdoor experiments.

The remainder of the paper is as follows. We provide an overview of the AR.Drone's hardware and software including vision and control algorithms in Section 2. We describe our system's architecture in Section 3. We present the evaluation results in Section 4 and conclude the paper in Section 5.

2 AR.Drone Platform

This section provides a brief overview of the AR.Drone quadrotor. A more complete description of the algorithms can be found in [16].

2.1 Hardware

The AR.Drone is a quadrotor with a size of 55 cm rotor-tip to rotor-tip and 380–420 grams of weight (see Fig. 1). The central cross of the AR.Drone is made of carbon fiber tubes joined by a central plastic plate. A brushless motor is mounted at the end of each arm. The electronics are housed by a plastic basket on the center of the cross and an Expanded Poly Propylene body.

The electronic hardware comprises two boards connected to each other and attached to the

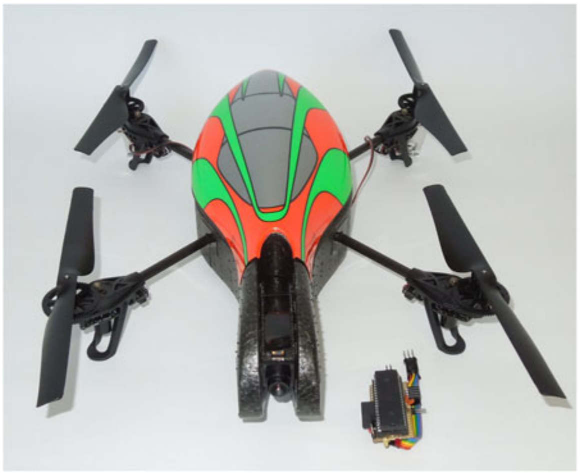


Fig. 1 Low cost quadrotor AR.Drone version 1.0 and 8-bit microcontroller. The AR.Drone used for trajectory following experiments. The microcontroller is in charge of path planning and control tasks.

plastic basket in the center of the cross to reduce vibrations. One board, the motherboard, contains the main processing unit (an ARM9 processor running at 468 MHz on AR.Drone 1.0 and an ARM Cortex-A8 at 1 GHz on AR.Drone 2.0), two cameras, a Wi-Fi module and a connector for software flashing and debugging. The cameras have different orientations. One is oriented vertically, pointing to the ground. It has an opening angle of 63°, a frame rate of 60 frames per second (fps) and a a resolution of 320 × 240 pixels. This camera is used to estimate the horizontal speed of the vehicle. The second camera is pointing forwards. It has an opening angle of 93°, a frame rate of 15 fps and a resolution of 640 × 480 pixels on version 1.0. Version 2.0 has an improved front camera that can deliver high definition images at 720p resolution at 30 fps . Both cameras can be used for detection of markers such as stickers, caps or hulls of other AR.Drones.

The second board, the navigation board, contains all the sensors necessary for the state estimation of the quadrotor. The sensor suite includes a low cost inertial measurement unit (IMU) that updates the state of the AR.Drone at 200 Hz. The IMU consists of a 3-axis accelerometer, 2-axis gyroscope to measure angular velocities on pitch and roll axes and a more precise gyroscope to measure the angular velocity on the yaw axis. The distance to the ground is measured by an ultrasonic sensor with a maximum range of 6 m. The AR.Drone version 2.0 also equips a barometric sensor to measure its height beyond the range of the ultrasonic sensor. This measurement is also used to determine the scene depth in the images of the vertical camera and to calculate vertical displacement of the vehicle.

2.2 Software

The AR.Drone can be remote controlled via a smartphone or a tablet PC. It can also accept commands or send data to a PC via an AR.Drone API. The AR.Drone can provide a vast amount of sensor sensor data to the user. These data is divided into two types of streams: a *navdata* stream that containing data related to the state of the vehicle and a *video* stream that provides encoded video from the cameras. The navigation data includes the status of the vehicle, motors and communications as well as raw and filtered IMU measurements, attitude (roll, pitch, yaw), altitude, linear velocities and an position with respect to take off point computed from visual information. The navigation data also includes information from the detection of visual tags. The data is subdivided into several options that group different sensor data. the user can request the options that contain sensor data of interest and receive them periodically. The received video can be from either of the two cameras or a picture-in-picture video with one of the camera images superposed on the top left corner of the other one.

The ARM processor runs an embedded Linux operating system that simultaneously manages the wireless communications, visual-inertial state estimation and control algorithms.

2.2.1 Navigation Algorithms

The AR.Drone provides sophisticated navigation algorithms and assistance in maneuvers such as take off, landing and hovering-in-position to ensure user's safety. To achieve a stable hovering and position control, the AR.Drone estimates its horizontal velocity using its vertical camera. Two different algorithms are used to estimate the horizontal velocity. One tracks local interest points

(FAST corners [17]) over different frames and calculates the velocity from the displacement of these points. It provides a more accurate estimate of the velocity and is used when the vehicle's speed is low and there is enough texture in the picture. The second algorithm estimates the horizontal speed by computing the optical flow on pyramidal images. It is the default algorithm during flight. It is less precise but more robust since it does not rely on highly textured or high-contrast scenes.

The AR.Drone uses inertial information from its IMU for estimating the state of the vehicle. It fuses the IMU data with information from the vision algorithms and an aerodynamics model to estimate the velocity of the vehicle.

2.2.2 Control

The control of the AR.Drone is performed in a nested fashion. The innermost loop controls the attitude of the vehicle using a PID controller to compute a desired angular rate based on the difference between the current estimate of the attitude and the attitude set point defined by the user controls. The second loop uses a proportional controller to drive the motors.

When the controls are released, the AR.Drone computes a trajectory that will take it to zero velocity and zero attitude in a short time. This technique is designed off-line and uses feed-forward control over the inverted dynamics of the quadrotor.

3 System Architecture

Our system consists of two main components: an external processing unit and a proxy program that serves as a bridge between the AR.Drone and the external processing unit (Fig. 2). We used three different devices as external processing unit. One is an ATMEL ATmega 1284 8-bit microcontroller (Fig. 1). It runs at 14.7456 MHz, has 128 Kb of memory, 32 I/O pins, counters and timers with PWM capabilities as well as two USART ports and a 10-bit analog to digital converter. It extends the capabilities of the AR.Drone to enable autonomous flying. The second unit is a Gumstix Overo Fire board (Fig. 3). This single-board computer outfits an ARM Cortex-A8 processor running at 700 MHz with 250 MB of RAM. An additional expansion board provides access to

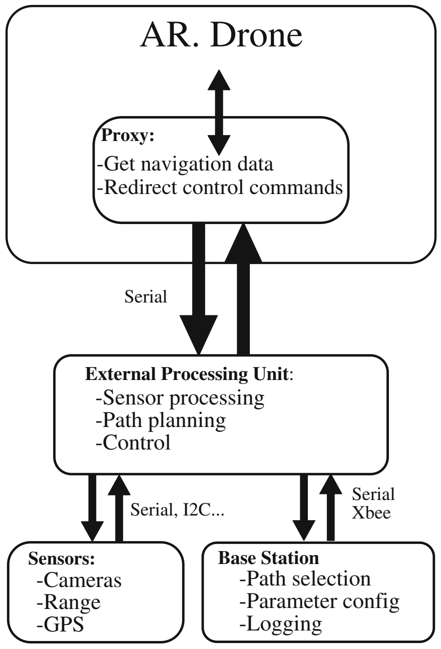


Fig. 2 System software components. The proxy application runs on the AR.Drone and enables exchange of information with the external processing unit. High level tasks for navigation and control run in the external processing unit. The external processing unit can interface with different sensors. An optional base station communicates with the external processing unit to configure the flight and log flight data

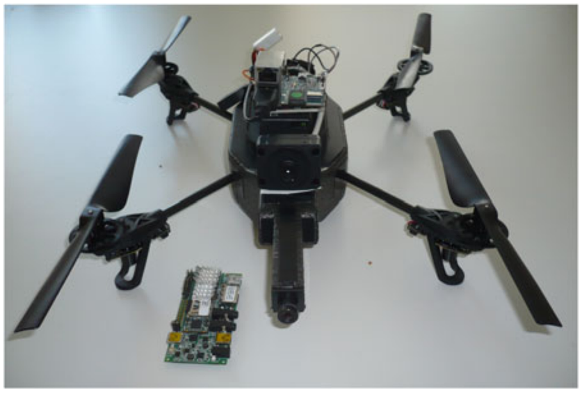


Fig. 3 On-board vision setup. To process images on board of the AR.Drone 2.0 an ODROID-U2 (mounted) or a Gumstix Overo Fire are equipped as external processing unit. To achieve a better frame rate a Pointgrey Firefly color camera is also added

USB ports and ehternet connections. With the expansion board the setup measures 105 × 40 mm and weights aproximately 35 g. The third unit is a Hardkernel ODROID-U2. It is a compact developing board with an ARM Cortex-A9 quadcore processor running at 1.7 GHz. Its dimensions are 48 × 52 mm and weights 40 g. It has 2 GB of RAM memory and a 64 GB card of flash memory. It has two USB ports, a UART port and ethernet connection. It runs an Ubuntu Linux system. The last two boards are used to process images on-board. These systems provide the additional computational power required for high level navigation tasks without the need of a ground station or any remote control device, thus widening its application range and eliminating delays that may be caused by the communication with the base.

AR.Drone Proxy We uploaded an external application to the AR.Drone to exchange information with the external processing unit, i.e. send steering commands, retrieve its status, sensor measurements, attitude and pose estimates. The external processing unit starts the application after the AR.Drone finishes its booting sequence. The application connects to the navigation and command ports of the AR.Drone main program and requests the navigation data (velocities, altitude, orientation and position from take off point). The application acts as a proxy: It listens on the navigation port for incoming data, sorts the data, packs it into a binary message and sends it to the external processing unit. At the same time it listens for AR.Drone's configuration or control commands coming from the external processing unit. These commands are directly forwarded to the AR.Drone's command port.

External Processing Unit To execute autonomous navigation, control algorithms and perform computer vision on-board, we attached the external processing unit's UART port to the serial debugging port located under the AR.Drone. With these algorithms running on an external processor all the computational resources on the AR.Drone are reserved for its pose estimation and control algorithms. An external processing unit allows us to interface with a wide variety of sensors for example cameras, range sensors, GPS, temperature sensors or radio communications. The number of sensors that can be added is limited by the unit's specifications, their power consumption and the payload of the AR.Drone. The system can be designed according to the mission objectives and budget. We designed three different systems with distinct characteristics. One is a low cost system capable of figure flying using a microcontroller as external processing unit. The other two systems are capable of processing images on-board to detect a known pattern or a person wearing an orange shirt. They use a Gumstix Overo Fire or an Odroid developing board as an external processing unit and an additional color camera.

Autonomous Figure Flying The hardware used for this system is the previously described microcontroller mounted on an AR.Drone version 1.0. The additional hardware weights approximately 25 g. The weight located close to the center of gravity of the AR.Drone to avoid affecting the balance and stability of the vehicle. There is no noticeable influence on flight performance with the outdoor protective hull and 10 cm height loss on the hovering after the take off which the controllers are able to correct. With our current setup, flight time is reduced by approximately 2–3 min due to the extra weight and power consumption of the microcontroller.

The software framework for autonomous figure flying runs on this microcontroller. It consists of three modules: position estimation, path planning and control. The first module exploits the AR.Drone's position estimates from vision algorithms (see Section 2). The position and heading are provided in world coordinates with the origin at the take off location and the height value as the current measurement over the ground. The proxy application gets these data from the AR.Drone's navigation data stream. Then the proxy application builds a binary message with the AR.Drone's status, its position and heading and sends it to the microcontroller. We treat the AR.Drone as an additional sensor providing position information to the planning and control modules running on the microcontroller.

The second module, path planning, computes a trajectory for the AR.Drone to follow, based on predefined shapes or user defined waypoints.

To follow a predefined shape, which can be a simple straight line, a circle or an eight shape, we calculated mathematical equations that parametrize the shape according to its size. To follow waypoints, we define a series of coordinates that outline a desired trajectory. Each waypoint w is defined as a triplet $w = [x, y, z]$ where x, y, z are in world coordinates with the origin as the take off location of the vehicle. The trajectory is computed as a series of smooth connected functions of time for each of the axes. The individual functions have the form $at + bt^2 + ct^3$ where the coefficients a, b and c are computed before the flight using the desired velocity of the vehicle. The yaw angle ϕ is computed so that the front camera of the AR.Drone points in the direction of the next waypoint. Navigating in this way keeps the camera pointing always in the direction of flight but is not optimal for the control of quadrotors. Yaw rotation produces centrifugal forces that push the vehicle away from the desired path requiring additional maneuvering to correct its trajectory.

In the third module, trajectory following control, we feed the AR.Drone's position estimate and the planned set point position to four PID controllers. The PID controllers compute commands for the roll, pitch and yaw angles and for the desired height of the vehicle independently from each other. These commands are then sent back to the AR.Drone via the proxy application that redirects it to the vehicle's command port. The control gains for each controller were experimentally determined.

Path planning and PID controller parameters can be configured from an optional base station. The base station communicates with the microcontroller on the AR.Drone via a radio module. It records flight data and displays useful information from the vehicle like its status and battery level and a visualization of the path.

Pose Estimation This system uses the Gumstix Overo Fire single-board computer mounted on an AR.Drone 2.0. This system is designed for on-board image processing. The front camera of the AR.Drone has a rolling shutter and delivers images at 15 Hz with a 1080p resolution. To ensure a better image quality and a faster frame rate we equipped a Pointgrey Firefly color camera with a global shutter that delivers VGA resolution images at a speed of 60 Hz.

The pose estimation of the vehicle is done in two steps. First we identify a pattern consisting of three markers. Then we compute the pose of the vehicle from the image and world location of the markers.

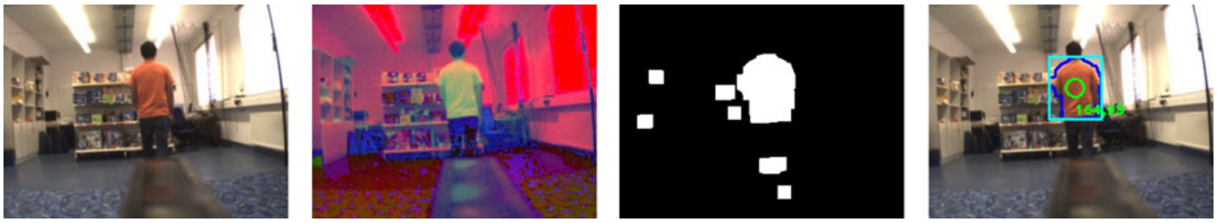
We have designed a pattern made from three orange table tennis balls as markers to easily identify these three points in the image.The marker detection is done by color segmentation with an efficient algorithm that allows the processing of every frame coming from the camera. To speed up the process, the algorithm is applied to the raw image rather than debayering the image. To further save computation time, we assume a minimum marker size of three pixel and only check every fourth pixel. The pixel's color is tested using a look up table. If the pixel's color match the marker color a floodfill algorithm is applied to incrementally to find a region. Next, regions are filtered by size to eliminate too large and too small regions. The aspect of the region is then tested to find those similar to circles using the Randomized Hough Transform. Finally the four largest remaining regions are taken as the markers.

The pose of the vehicle is computed from three points in the image whose world location is known. This is known as the perspective-n-problem (PnP). Once the markers have been found the pose of the vehicle relative to the pattern is estimated by solving the PnP problem. We used the approach described in [18].

This position is then fed to four PID controllers–one for each roll, pitch, yaw angles and another one for the thrust. The controllers compute commands to steer the vehicle to a desired position. The commands are sent to the AR.Drone at a speed of 60 Hz.

Person Detection This system uses an ODROID-U2 developing board mounted on an AR.Drone 2.0. This system is designed for on-board image processing.

The total weight of the additional hardware is 80 g. The board is mounted directly above the battery and the camera in front of the battery. To reduce the overall weight the hulls are not used.



(a) original (b) HSV (c) Thresholded (d) Result

Fig. 4 Pipeline of processing steps. The original image (Fig. 4a) is converted to HSV color space for color segmentation (Fig. 4b). Then the image is thresholded to obtain only objects matching an *orange color* (Fig. 4c). Finally, objects are filtered by size and aspect ratio to eliminate noise and objects with incorrect aspect. The largest object remaining is the *orange shirt* (Fig. 4d)

The system is able to detect a person wearing an orange shirt and keep a constant distance towards him. The detection is based on segmentation techniques to extract large regions that are homogeneous in color. The proposed segmentation pipeline is as follows: Once an image is received, it is scaled down to a 320×240 pixels resolution, as it is not necessary to extract small details of the scene. This speeds up the processing of the image and reduces noise. Then the image is converted to the HSV color space. The HSV color space separates color information from intensity information providing more robustness to light changes and shadows. After the color space conversion the image is thresholded to extract the regions matching the orange color. Remaining noise is further eliminated by a morphological opening operation. Once the regions have been extracted they are filtered once again by their dimensions and proportions. Regions that exceed the desired size or are too thin are eliminated. The largest remaining region is regarded as the orange shirt and a bounding box is calculated around it.

Figure 4 shows images of the various steps performed for shirt detection. In Fig. 4b one can see that the shirt color stands out in the image. In the thresholded image (Fig. 4c) several small objects appear from objects on the shelf on the background. This objects are then filtered out by size and proportions to leave only the shirt as the single object (Fig. 4d).

We use two PID controllers to compute commands for the yaw and the height to keep the center of the bounding box in the center of the image. A third PID controller computes commands for the roll angle to keep the height of the bounding box at a desired size. The height of the desired bounding box–computed offline–is calculated from the average of 100 measurements at the desired distance.

4 Experiments and Results

We tested the three systems considering different scenarios. One system to autonomously fly predefined figures with a microcontroller. The second uses computer vision to find three known markers and compute the vehicle's pose. The third uses onboard vision to follow a person wearing an orange shirt.

4.1 Autonomous Figure Flying

4.1.1 Experimental setup

To test the capabilities of our system and analyze its performance, we performed a series of autonomous flights. We used an AR.Drone version 1.0 communicating with the microcontroller as described in Section 3. Flight data are logged at a frequency of approximately 30 Hz on the base station that communicates with the microcontroller using a radio module. The flight experiments were performed in two different rooms, both equipped with a motion capture system that monitors the position of the vehicle at a frequency of 100 Hz with millimeter precision for accurate ground truth measurements. One environment was a small room where the motion capture

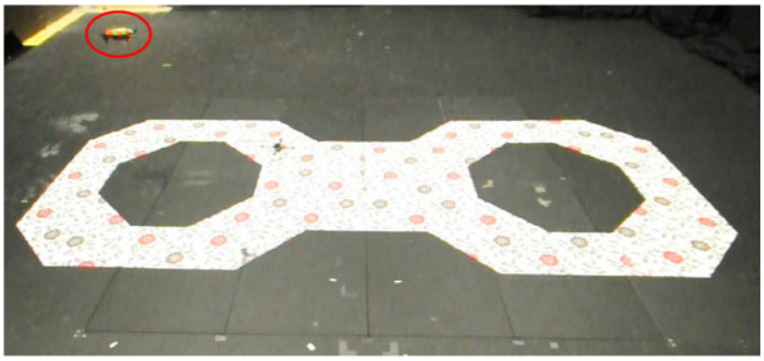


Fig. 5 Indoor environment for the path planning experiments. The room is equipped with a motion capture system to provide positioning ground truth. The figure of eight is marked on the floor to provide texture for the AR.Drone's vision algorithms

system covers a volume of 3 × 3 × 2 m^3. The room has a carpet that provides texture for the estimation algorithms of the AR.Drone. The second room covers a larger volume of 10 × 12 × 6 m^3. This room lacks texture on the floor so we laid colored paper on the floor forming the figure of an eight to provide the texture (see Fig. 5).

To test the trajectory following, we defined waypoints within a 2 × 2 m^2 area of the small room arranged to form different patterns. After loading parameters for the flight to the microcontroller, it calculated the trajectory to follow. We then commanded the AR.Drone to hover at 1.10 m height and then start the autonomous flight. The AR.Drone flew three figures with varying level of difficulty: a square, a zig-zag and a spiral. In the larger room, the AR.Drone was commanded to fly a predefined figure-eight of 5 m length and 1.75 m width over the marked figure on the floor of the room while keeping a constant height of 1.10 m.

4.1.2 Results

To evaluate the performance of our system, we analyzed the data collected from 20 flights. We first synchronized the data logged on the base station with the data provided by the tracking system. The data from both systems is synchronized by noting the start of the experiment (movement along x-y axis) on both data logs, the end of the AR.Drone's log and the total time of the experiment from these data and interpolating the tracking system's data at the time marked by the AR.Drone's data log timestamp. We measured the accuracy of the AR.Drone pose estimation by comparing it with ground truth data (see Table 1). The root mean square error on the short trajectories was smaller than 5 cm and on the longer trajectory around 10 cm on x and y axes while on the yaw angle the RMSE was around 3° and 10°, respectively. The difference in error values can be caused by drift in the sensors, which translates to a small drift of the position estimation. As it was expected, the height estimation from the ultrasonic range sensor yields comparable results in both experiments (around 1 cm). The errors and standard deviations in all axes showed that the AR.Drone's odometry provides enough accuracy for flying figures autonomously.

Figure 6 shows the measurements of the position and heading of the vehicle during a typical flight of the figure eight. A top view of one of the experiments where we flew the figure eight is shown in Fig. 7. Figure 8 shows the top view of the spiral, zig-zag and square figures flown in the small room.

Table 1 Error analysis of AR.Drone's odometry

	x [cm]	y [cm]	z [cm]	*yaw* [deg.]
Short trajectory				
RMSE	4.88	3.14	0.69	3.05
Std. dev.	2.77	3.01	0.78	1.50
Max error	11.81	10.32	6.74	6.40
Long trajectory				
RMSE	8.04	12.82	1.21	9.20
Std. dev.	6.04	11.57	1.18	5.80
Max error	25.26	40.60	10.28	17.74

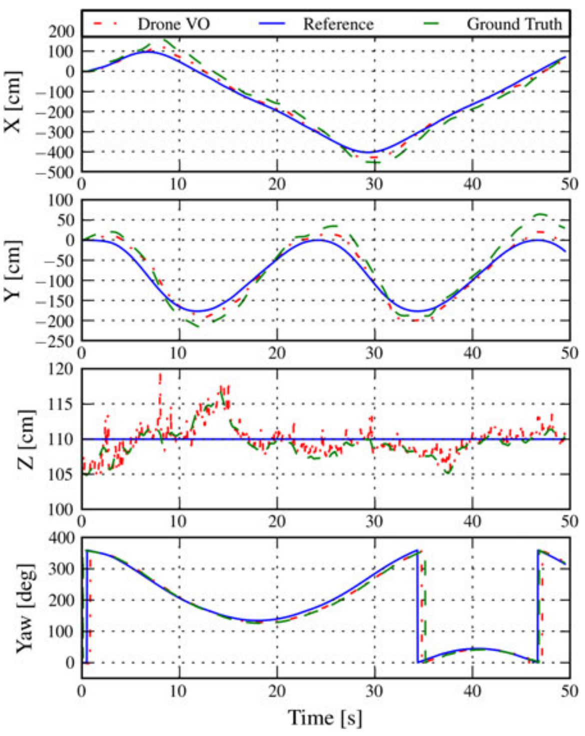


Fig. 6 Measurements from a typical trajectory following experiment. The AR.Drone was commanded to follow an eight shape trajectory of size 5 m × 1.75 m at an altitude of 1.10 m. Planned path (*solid blue*), AR.Drone's estimated position (*dash-dotted red*) and the ground truth (*dashed green*) are shown. Note that the yaw angle trajectory is smooth and the plot shows the wraparound at 360°

We analyzed the performance of the controllers for trajectory following by comparing the planned trajectory with the pose estimate of the AR.Drone (see Table 2). The controllers performed very similar on both short and long trajectories as suggested by the RMSE and standard deviation values on all axes. There was a slightly larger error and standard deviation on the shorter flights, since the trajectories involved more changes in the heading direction of the vehicle than in the larger figure-eight, and in the case of the spiral, the changes were increasingly faster. With these properties, our controllers are capable of autonomous operation of the AR.Drone in reduced and narrow spaces with average errors 5 times smaller than the size of the vehicle and maximum errors of about half a vehicle length.

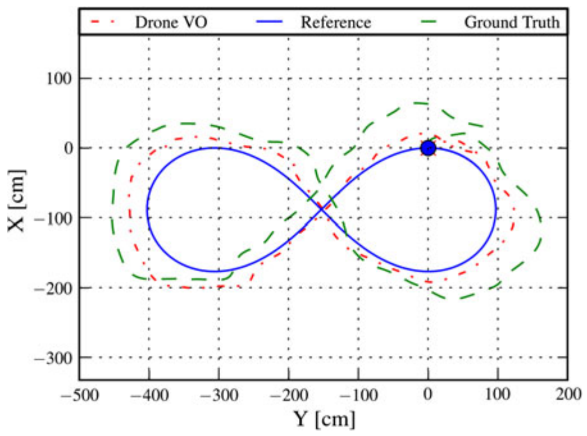


Fig. 7 X-Y projection of a typical eight shape trajectory flight In our experiments in the large room the AR.Drone followed an eight shape trajectory at a constant height of 1.10 m. The stating point is chosen so the AR.Drone starts with a yaw angle of 0°. Planned path (*solid blue*), AR.Drone's estimated position (*dash-dotted red*) and the ground truth (*dashed green*) are shown

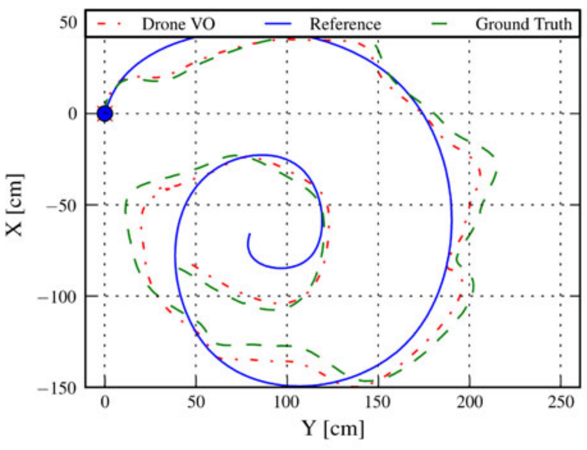
4.2 Pose Estimation

4.2.1 Experimental Setup

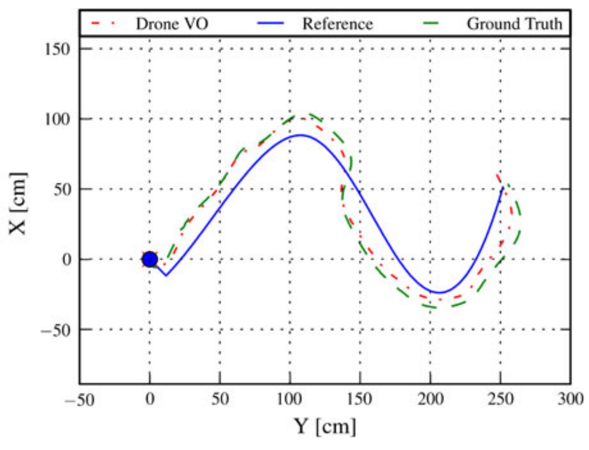
We used an AR.Drone 2.0 equipped with a Gumstix Overo Fire and a Pointgrey Firefly color camera as described in Section 3. This experiment consisted in letting the vehicle hover in position at a fixed distance from the pattern made with the orange markers. We fixed the pattern at 110 cm height and command the vehicle to hover at a distance of 160 cm directly behind the pattern. The position estimation computed by the vehicle is recorded and compared with the distance between the pattern and the vehicle measured by a motion capture system with millimeter precision. The controllers for x, y and z axis are evaluated by measuring the deviation from desired hover position over the flight. The yaw angle was not evaluated since it was designed to keep the pattern in the center of the image along the horizontal.

4.2.2 Results

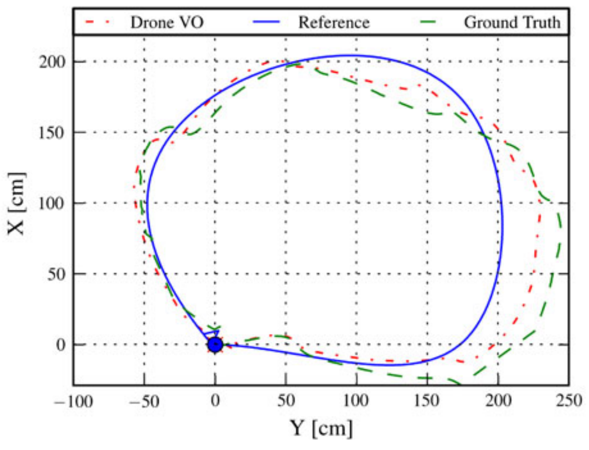
The overall position error was lower than 15 cm for the x and y axes and lower than 5 cm for



(a) spiral



(b) Zigzag



(c) Square

Fig. 8 X-Y projections of typical flights of the *spiral*, *zig-zag* and *square figures* flown in the small room. The different trajectories were computed from waypoints defined in the $3 \times 3 \times 2$ m^3 volume of the room

the z axis. Table 3 shows the details of 10 min of flight. Whereas the AR.Drone's visual odometry will drift after some time, this system is capable of holding the position relative to the pattern. Figure 9 shows the measurements of one typical hover flight. The position estimation provided by our system is matches the ground truth data provided by the tracking system.

Table 2 Error analysis of trajectory tracking

	x [cm]	y [cm]	z [cm]	yaw [deg.]
Short trajectory				
RMSE	7.55	12.02	3.06	4.02
Std. dev.	7.23	7.28	2.74	2.82
Max error	25.70	29.70	11.30	13.90
Long trajectory				
RMSE	7.58	11.34	3.14	5.93
Std. dev.	4.91	6.90	2.04	3.34
Max error	23.20	26.00	12.50	14.51

Table 3 Error analysis of position holding

	x [cm]	y [cm]	z [cm]
RMSE	11.53	10.22	3.26
Std. dev.	7.03	8.85	2.51
Max error	22.42	28.41	10.11

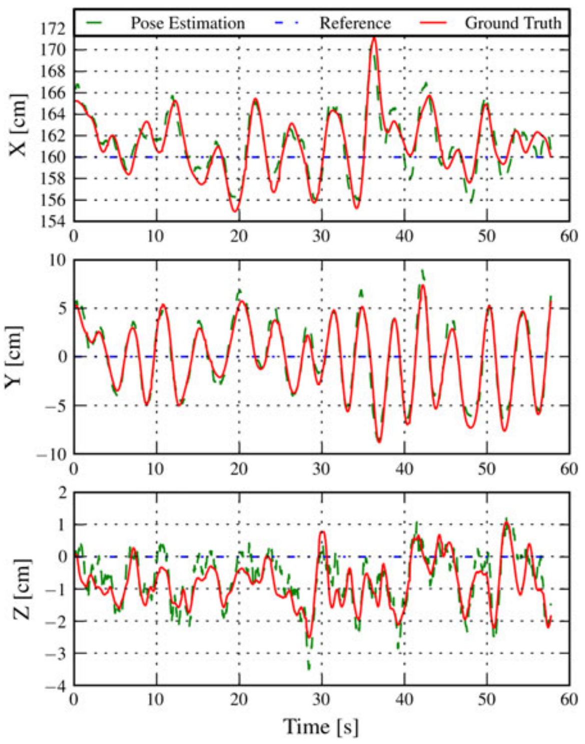


Fig. 9 Measurements a typical position holding experiment. The AR.Drone was commanded to hover at a fixed distance of 160 cm directly behind the pattern. Desired position (*dash-dotted blue*), AR.Drone's estimated position (*dashed green*) and the ground truth (*solid red*) are shown

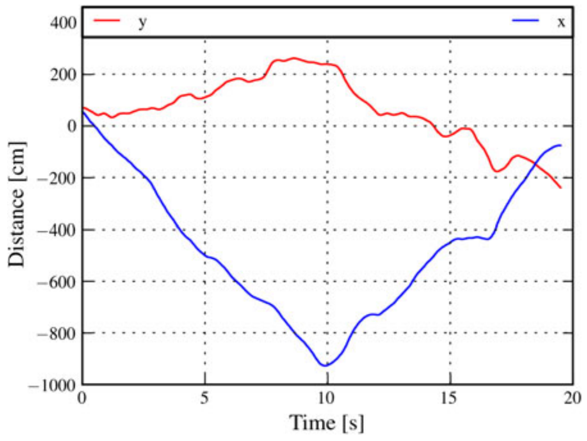


Fig. 10 Path recorded by the odometry of the AR.Drone while following a person walking a 10 m straight line in both directions

4.3 Person Following

4.3.1 Experimental Setup

We used an AR.Drone 2.0 equipped with a Hardkernel ODROID-U2 and a Pointgrey Firefly color camera as described in Section 3. In this experiment a person wearing an orange shirt walks a 10 m straight line while the vehicle follows him at a distance of approximately 2 m in an outdoor setup. Once he reaches the end, he turns around and walks back to the starting point. During the whole flight, the path of the vehicle is estimated by its visual odometry.

4.3.2 Results

Figure 10 shows the odometry data collected during one flight where the vehicle followed a person walking a straight line of approximately 10 m on both directions. The vehicle follows the path of a line in the x direction and stays within 1 m when it starts going back and finishes close to the starting x position. The y direction is not controlled actively since the vehicle lacks the knowledge of this dimension and instead controls its yaw to keep the person in the center of the picture. The vehicle drifts from the starting y position but stays in the desired range of 2 m following the person and finishes approximately around 2 m from the starting point.

5 Conclusion

We presented a framework that allows the low-cost quadrotor AR.Drone to equip additional processing power to perform autonomous flights, position estimation and person following. Our method uses on-board sensing only with all the computations performed on-board. The system can be used for extended distances since it does not require any base station or remote control to operate. It can be easily customized and extended with additional sensors or a more powerful external processing unit that can speed up the computations or carry out more computationally challenging tasks.

We evaluated the different systems capabilities within different settings. A simple low power system based on a microcontroller was tested by following predefined trajectories and a series of user defined waypoints. We used the AR.Drone's visual odometry which provided reliable estimates for this task. The trajectory tracking errors were small in comparison with the size of the quadrotor. This shows that our system is capable of flying in narrow environments. Two other systems to perform on-board computer vision were also tested. One to estimate the vehicle's position relative to a known pattern. And another one to find a person wearing a distinctive shirt and follow him. Both systems proved capable of processing the images and suitable to perform computer vision on-board.

Acknowledgements The authors would like to thank Volker Grabe and Dr. Paolo Robuffo Giordano from the Max Planck Institute of Biological Cybernetics for letting us use their tracking system room and Karl E. Wenzel for his valuable input and advice with the AR.Drone.

References

1. Lupashin, S., Schöllig, A., Sherback, M., D'Andrea, R.: A simple learning strategy for high-speed quadrocopter multi-flips. In: IEEE International Conference on Robotics and Automation, pp. 1642–1648 (2010)
2. Muller, M., Lupashin, S., D'Andrea, R.: Quadrocopter ball juggling. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 5113–5120 (2011)
3. Mellinger, D., Michael, N., Kumar, V.: Trajectory generation and control for precise aggressive manuevers with quadrotors. Int. J. Robot. Res. **31**(5), 664–674 (2012)
4. Hoffmann, G., Rajnarayan, D.G., Waslander, S.L., Dostal, D., Jang, J.S., Tomlin, C.J.: The Stanford testbed of autonomous rotorcraft for multi-agent control (STARMAC). In: 23rd Digital Avionics Systems Conference, Salt Lake City (2004)
5. Meier, L., Tanskanen, P., Fraundorfer, F., Pollefeys, M.: Pixhawk: a system for autonomous flight using onboard computer vision. In: IEEE International Conference on Robotics and Automation (ICRA), 2011, pp. 2992–2997 (2011)
6. Lim, H., Park, J., Lee, D., Kim, H.J.: Build your own quadrotor: open-source projects on unmanned aerial vehicles. IEEE Robot. Automat. Mag. **19**(3), 33–45 (2012)
7. Tayebi, A., McGilvray, S., Roberts, A., Moallem, M.: Attitude estimation and stabilization of a rigid body using low-cost sensors. In: 46th IEEE Conference on Decision and Control, 2007, pp. 6424–6429 (2007)
8. Wenzel, K., Masselli, A., Zell, A.: Automatic take off, tracking and landing of a miniature uav on a moving carrier vehicle. J. Intell. Robot. Syst. **61**, 221–238 (2011). doi:10.1007/s10846-010-9473-0
9. Wenzel, K.E., Masselli, A., Zell, A.: Visual tracking and following of a quadrocopter by another quadrocopter. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 4993–4998 (2012). doi:10.1109/IROS.2012.6385635
10. Saska, M., Krajník, T., Faigl, J., Vonásek, V., Preucil, L.: Low cost mav platform ar-drone in experimental verifications of methods for vision based autonomous navigation. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2012), pp. 4808–4809 (2012)
11. Faigl, J., Krajník, T., Vonásek, V., Přeučil, L.: Surveillance planning with localization uncertainty for UAVs. In: 3rd Israeli Conference on Robotics, Ariel (2010)
12. Bills, C., Chen, J., Saxena, A.: Autonomous mav flight in indoor environments using single image perspective cues. In: IEEE International Conference on Robotics and Automation, pp. 5776–5783 (2011)
13. Engel, J., Sturm, J., Cremers, D.: Camera-based navigation of a low-cost quadrocopter. In: Proc. of the International Conference on Intelligent Robot Systems (IROS) (2012)
14. Klein, G., Murray, D.: Parallel tracking and mapping for small AR workspaces. In: Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07), Nara, Japan (2007)
15. Jimenez Lugo, J., Zell, A.: Framework for autonomous onboard navigation with the ar.drone. In: 2013 International Conference on Unmanned Aircraft Systems (ICUAS), pp. 575–583 (2013)
16. Bristeau, P.-J., Callou, F., Vissiere, D., Petit, N.: The navigation and control technology inside the AR.Drone micro uav. In: 18th IFAC World Congress, pp. 1477–1484, Milano, Italy (2011)
17. Trajkovic, M., Hedley, M.: Fast corner detection. Image Vis. Comput. **16**(2), 75–87 (1998)
18. Masselli, A., Zell, A.: A novel marker based tracking method for position and attitude control of MAVs. In: Proceedings of International Micro Air Vehicle Conference and Flight Competition, pp. 1–6, DGON, Braunschweig (2012)