# Integrating sensor and motion models to localize an autonomous AR.Drone

Dijkshoorn, N.; Visser, A.

DOI
10.1260/1756-8293.3.4.183
Publication date
2011
Document Version
Author accepted manuscript
Published in
International Journal of Micro Air Vehicles

# Integrating Sensor and Motion Models to Localize an Autonomous AR.Drone

**Nick Dijkshoorn and Arnoud Visser**

Reprinted from

International Journal of
**Micro Air Vehicles**

Volume 3 · Number 4 · December 2011

**Multi-Science Publishing**
**ISSN 1756-8293**

# Integrating Sensor and Motion Models to Localize an Autonomous AR.Drone

**Nick Dijkshoorn and Arnoud Visser**
Intelligent Systems Lab Amsterdam, Universiteit van Amsterdam,
Science Park 904, Amsterdam, the Netherlands
Email: a.visser@uva.nl

## ABSTRACT

This article describes a method to develop a generic approach to acquire navigation capabilities for the standard platform of the IMAV indoor competition: the Parrot AR.Drone. Our development is partly based on simulation, which requires both a realistic sensor and motion model. The AR.Drone simulation model is described and validated. Furthermore, this article describes how a visual map of the indoor environment can be made, including the effect of sensor noise. This visual map consists of a texture map and a feature map. The texture map is used for human navigation and the feature map is used by the AR.Drone to localize itself. To do so, a localization method is presented. An experiment demonstrates how well the localization works for circumstances encountered during the IMAV competition.

## 1. INTRODUCTION

Nowadays, small quadrotors with on-board stabilization like the Parrot AR.Drone can be bought off-the-shelf. These quadrotors make it possible to shift the research from basic control of the platform towards applications that make use of their versatile scouting capabilities. Possible applications are surveillance, inspection and search & rescue. Still, the limited sensor suite and the fast movements make it quite a challenge to fully automate the navigation for such platforms. One of the prerequisites for autonomous navigation is the capability to make a map of the environment.

The paper proceeds as follows. In Section 2 we give a short overview of related work. Section 3 describes our methods. First the simulation model of the AR.Drone is outlined in Section 3.1. Section 3.2 describes how the AR.Drone's pose can be estimated. This pose is used to build two visual maps: a texture map (Section 3.3.1) and a feature map (Section 3.3.2). The texture map is used for human navigation and the feature map can be used to localize the AR.Drone (Section 3.4). We present experiments and results in Section 4. This section demonstrates how well the localization works for circumstances encountered during the IMAV competition. Conclusions and future work are covered in Section 5 and 6.

## 2. RELATED WORK

Our approach was inspired by Steder *et al*. [1], who presented a system that allows aerial vehicles to acquire visual maps of large environments using comparable setup with an inertial sensor and a low-quality camera pointing downward. In their approach the inertial sensor was used to estimate a number of parameters in the spatial relation between two camera poses, which reduces the dimensionality of the pose to be estimated. Equivalent with our approach, Steder uses Speeded-Up Robust Features (SURF) [2] that are invariant with respect to rotation and scale. By matching features between different images, one can estimate the relative motion of the camera and construct the graph that serves as input to the TORO-based network optimizer [3].

While Steder uses a Euclidean distance measure to compare the correspondences of the features in two images, Caballero *et al*. [4] indicate that this is a last resort. They are able to make a robust estimation of the spatial relationship on different levels: homogeneous, affine and Euclidean. First an attempt is made

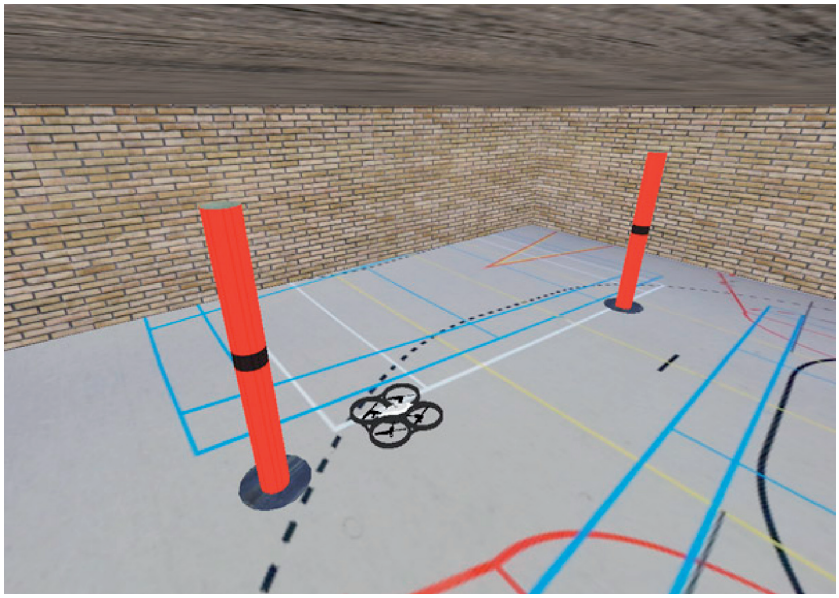

**Figure 1:** 3D model of a gym with realistic ground and wall textures which represents the circumstances during the Micro Air Vehicle pylon challenge.

to estimate complete homogeneous transformation (8 degrees of freedom). When the number of correspondences is too low an affine transformation (6 degrees of freedom) is estimated. Only when the features in the image are really noisy a Euclidean transformation (4 degrees of freedom) is estimated.

Since the 2011 summer edition of the Micro Air Vehicle competition, a team can choose to either use a standard platform or create their own. Prior to 2011 no standard platform existed in the competition, resulting in participants to focus on constructing optimal sensor configurations. For instance, [5] describes a quadrotor equipped a Hokuyo laser scanner, allowing many techniques developed for ground robotics to be applied on their flying robot. Another example is the approach from [6], where an omnidirectional camera was mounted on the platform, which was used to acquire a heading estimation from the optical flow. At the Spring 2011 Micro Air Vehicle competition[1] a Parrot AR.Drone was extended with an infrared camera to be able to follow people [7]. Because the AR.Drone is a quite recent development, the number of studies based on this platform is limited. A recent publication is from Cornell University [8], where an AR.Drone is used to automatically navigate corridors and staircases based on visual clues.

The other teams during the 2011 summer edition of the IMAV indoor competition relied on detecting the brightly coloured pylons to navigate their drones. In the 2010 edition the PixHawk team [9] already demonstrated that autonomous navigation is possible by making a visual map from the ground, although they needed artificial texture on the ground. In the method described in this article the natural texture of the ground in the gym is used.

As indicated in [10], an accurate simulation of a quadrotor is a valuable asset, which allows safe and efficient development of control algorithms. Additionally, it gives direct access to ground truth values and allows to design repeatable experiments. Validation of the sensor and actuator models is essential to guarantee the utility of both development and experimentation. Another useful feature is that the simulation environment is equipped with an editor, which allows detailed modifications to the simulated environment. The details needed for the experiment can be easily added as illustrated in Figure 1.

## 3. METHODS

In this section, we describe the AR.Drone simulation model, which includes a sensor and motion model. Furthermore, we describe how a visual map of the indoor environment can be made and how this map can be used by the AR.Drone to localize itself.

[1] http://www.springimav2011.org

## 3.1. Simulation Model

An accurate simulation of a quadrotor is a valuable asset, which allows safe and efficient development of control algorithms. Additionally, it gives direct access to ground truth values and allows to design repeatable experiments. The simulation environment selected is USARSim [11], which is based on the Unreal Tournament game engine[2]. It allows physical realistic simulations and a versatile environment editor.

### 3.1.1. Motion Model

The AR.Drone is a stabilized system (Figure 2). When no control signals are given the quadrotor hovers on the same location, which is accomplished by a feedback loop which uses the sonar (for altitude) and the bottom camera (for horizontal position). The simulation makes use of this assumption. When no control signal is given, the AR.Drone stays at the same location. When a control signal for a longitudinal or lateral velocity is given, it calculates the force needed to reach that velocity (and assuming that the drag force $D_b$ increases linearly with the velocity). When the control signal stops, the drag force $D_b$ slows the quadrotor down until it hovers again. The USARSim quadrotor model uses the Karma physics engine (part of the Unreal Engine [12]) to simulate the force and torque acting on the aircraft. Yet, only the overall thrust is calculated, the differential thrust is not used. When moving in the horizontal plane, a real quadrotor changes its angle of attack (which is the defined as the angle between direction of motion $e_V$ and the body frame $e_N$ [13]). The Karma physics engine does not need this angle to calculate the resulting horizontal movement. Yet, this angle of attack has direct consequences for the viewing directions of the sensors, so the roll and the pitch should be adjusted in correspondence with horizontal movements. The active control of the AR.Drone is incorporated in the value of the dragforce $D_b$.

Control signals for vertical and rotational movements (around the $z$-axis) are calculated in the same manner. For vertical movements not only the drag force $D_b$ is taken into account. In this case also the gravitational force $mg$ is included in the equation. Rotations around the $z$-axis stop quite quickly when no control signal is given. For this rotational movement a strong drag force $D_r = 20 \times D_b$ is used to model the additional inertia.

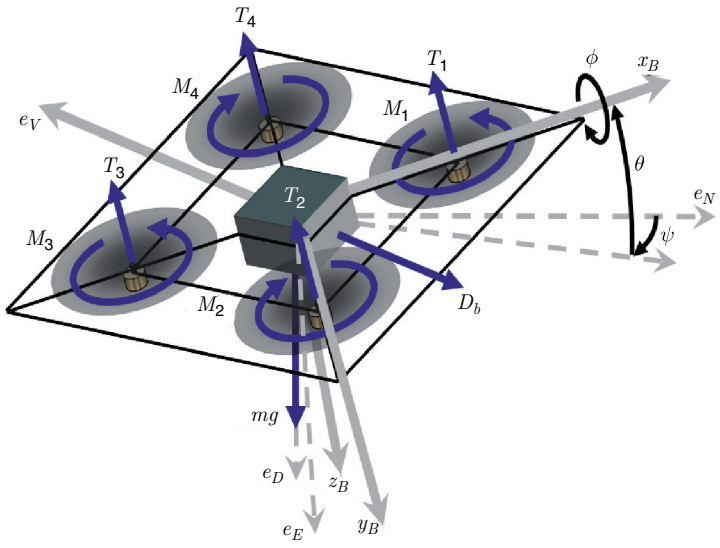


**Figure 2:** Free body diagram of a quadrotor helicopter [14]. Note that a right-handed orthogonal coordinate system is used with the $z$-axis pointing down. Each of the 4 motors has a thrust $T_i$ and momentum $M_i$. Together the motors should generate sufficient vertical thrust to stay airborne, which is indicated by the gravity force $m_g$ in the direction $e_D$. Differential thrust between the motors can provide roll $\phi$ and pitch $\theta$ torques, which lead to an angle of attack $\alpha$. This can result in fast movements of the helicopter (e.g. in the horizontal plane) in the direction $e_V$ which a resulting drag force $D_b$.

[2] http://www.unrealengine.com/

**Figure 3:** 3D model of the Parrot AR.Drone. This is a model optimized for simulation, based on the highly detailed model provided by Parrot SA.

The result is a simulation model (Figure 3), which maneuvers in a way similar to the actual AR.Drone, as demonstrated in Section 4.1. Both the simulated and real system have the same dimensions $(0.525, 0.515, 0.115)m$. The principal elements of inertia are calculated correspondingly to $(0.0241, 0.0232, 0.0451)kg \cdot m^2$, assuming a homogeneous distribution of the mass.

#### 3.1.2. Sensor Model

The USARSim simulator has a set of configurable sensors, which include all the sensors from the AR.Drone. Each mounted sensor is triggered at fixed time intervals (every 200 $ms$ by default). The AR.Drone's time interval is reduced to 5 $ms$, producing up to 200 sensor measurements per second.

The sonar sensor emits a number of traces, which in combination form a cone. First the sensor sends out one trace in the direction of its orientation remembering the measured range. Then it does several conical measurements and calculates the minimal measured distance. If the angle between a trace and the surface normal is smaller than angle $\alpha_{max\ Incidence}$ the measurement is ignored.

The acceleration sensor computes the body accelerations using the AR.Drone's velocity:

$$a_t = (v_t - v_{t-\Delta t}) / \Delta t \tag{1}$$

where $\Delta t$ is the time between measurements and $v$ is the velocity of the AR.Drone. The velocity of an object is modeled explicitly by the simulator.

The camera is modeled by creating a viewpoint in the Unreal engine. The field of view is set to 64 degrees and the resolution is set to $176 \times 144$ pixels, matching the specifications of the AR.Drone's bottom camera. USARSim supports dynamic lightning to produce realistic images and shadows. However, the camera sensor lacks a noise model and doesn't emulate the effect of automatic white balancing that is performed in most cameras (including the AR.Drone's camera). Our previous paper [15] briefly describes the influence of white balancing on image stitching and how to mimic the real images as close as possible.

### 3.2. Pose Estimation

In order to initialize a visual localization and mapping algorithm, a good estimate of the position based on the internal sensors of the AR.Drone is essential. The AR.Drone has several sources of information available about its movements and the majority of its computing power is dedicated to combine these sources into combined estimates. Here this information is combined into a state vector, with all information needed to initialize visual localization and mapping.

The AR.Drone is equipped with a number of sensors, which give frequent updates about the motion. The inertial sensor measures the body accelerations and angular velocities. A proprietary filter on board of the AR.Drone converts the angular velocities to an estimated attitude (orientation). A sonar sensor is used to measure the altitude of the vehicle. In addition to the body accelerations, the AR.Drone sends an estimate of its estimated body velocity. This estimate is based on the inertia measurements and optical flow obtained from the relative motion between camera frames. The details of this algorithm are proprietary information of the manufacturer.

The information from the sensors are used in an Extended Kalman Filter (EKF) to estimate the current position. The EKF has been considered [16] the facto standard in the theory of nonlinear state estimation. Our EKF state vector comprises a position vector $p^W$, velocity vector $v^W$, acceleration vector $a^W$ and attitude (orientation) vector $q^W$. All vectors are 3-dimensional Cartesian coordinates.

$$x = [p^W v^W a^W q^W] \tag{2}$$

The filtered attitude (orientation) from the proprietary onboard filter is written directly to the attitude vector $q^W$ of the state. Steder et al. [1] found that roll and pitch measurements are accurate, even for low-cost sensors and can be directly integrated into the state. The body accelerations can be used to estimate the current velocity of the vehicle. However, we found that the low-cost acceleration sensor from the AR.Drone provides unreliable data. This is modeled by a large uncertainty in the covariance matrix $Q$ of the EKF.

Unlike the acceleration data, the velocity estimate sent by the AR.Drone is accurate enough for integration into the state. Based on the previous position $p_{t-1}$, the state's velocity $v_t$ and time between measurements, the new position $p_t$ can be calculated as follows:

$$p_t = p_{t-1} + v_t \times \Delta t + a_t \times 0.5\Delta t^2 \tag{3}$$

where $\Delta t$ is the variable time (seconds) between the last two measurements. The body accelerations or velocity is sufficient to estimate the vehicle's altitude. However, the sonar's altitude measurement can be integrated into the EKF to improve the altitude estimate. The sonar sensor is not sensitive to drift because it provides an absolute altitude measurement. Relying only on the sonar sensor is not optimal since the measurements depend heavily on material and structure of the floor.

## 3.3. Visual Mapping

### 3.3.1. Texture Map

Now that we have an estimate of the AR.Drone's position and attitude in the world, we can use this information to build a visual map of the environment. The AR.Drone is equipped with a down-looking camera that has a resolution of $176 \times 144$ pixels. The frames captured by this camera can be warped on a flat canvas to create a visual map. Directly merging the frames on the canvas is not possible, because individual frames can be taken from a broad range of angles and altitudes. Instead, perspective correction is applied and all frames are normalized in size and orientation.

Before describing how a frame is warped, some basics about image formation are explained. The AR.Drone's camera is modeled using a pinhole camera model (Figure 4). In this model, a scene view

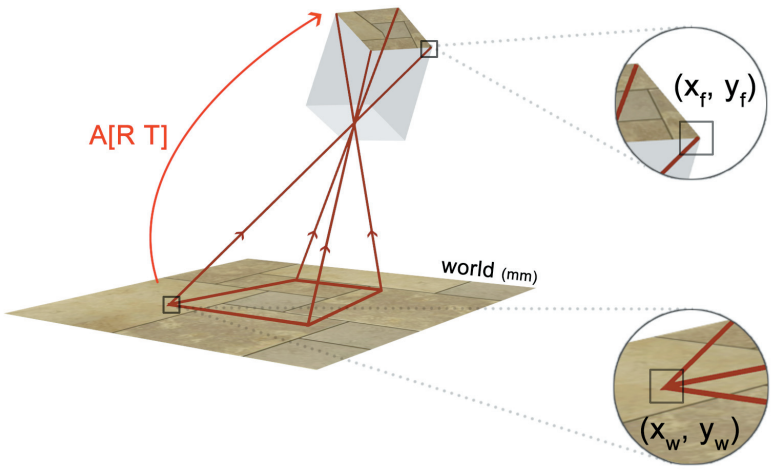


**Figure 4:** In the pinhole camera model, a scene view is formed by projecting 3D points into the image plane using a perspective transformation.

is formed by projecting 3D points into the image plane using a perspective transformation. 3D camera coordinates are typically described with homogeneous coordinates.

$$\begin{bmatrix} x_f \\ y_f \\ 1 \end{bmatrix} = A\begin{bmatrix} R \mid t \end{bmatrix}\begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{4}$$

where $x_f$ and $y_f$ represent a 2D point in pixel coordinates and $x_w$, $y_w$ and $z_w$ represent a 3D point in world coordinates. The $3 \times 3$ *camera intrinsic* matrix $A$ includes the camera's focal length and principal point. The $3 \times 4$ joint rotation-translation matrix $[R|t]$ includes the *camera extrinsic* parameters, which denote the coordinate system transformations from 3D world coordinates to 3D camera coordinates. Equivalently, the extrinsic parameters define the position of the camera center and the camera's heading (attitude) in world coordinates.

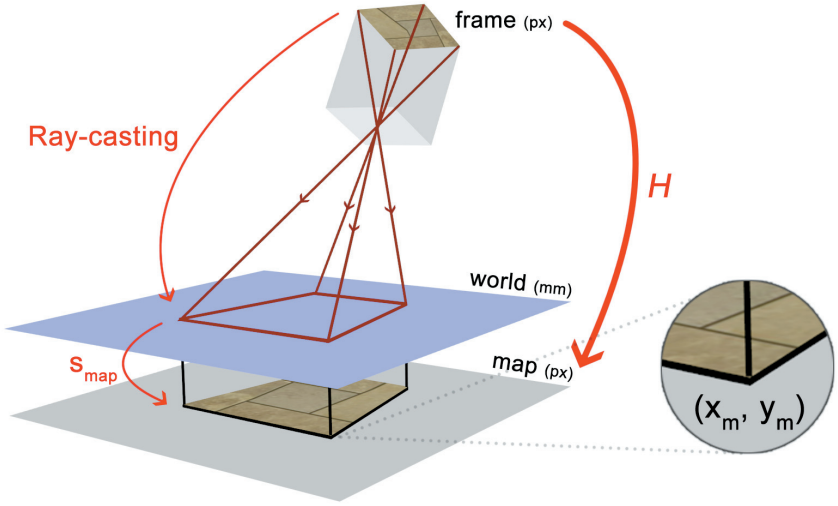


**Figure 5:** Visual map: warping a camera frame on the map's canvas. The frame's corners ($px$) are mapped to 2D world coordinates that lie on the world's plane. The 2D world coordinates of the frame are mapped to the map's canvas coordinates ($px$).

In order to warp a camera frame on the correct position of the canvas, we need to know which area of the world (floor) is captured by the camera. This is the inverse operation of the image formation described above. Instead of mapping 3D world coordinates to 2D pixel coordinates, 2D pixel coordinates are mapped to 3D world coordinates ($mm$). It is impossible to recover the exact 3D world coordinates, because a pixel coordinate maps to a line instead of a point (i.e., multiple points in 3D world coordinates map to the same 2D pixel coordinate). This ambiguity can be resolved by assuming that all 3D world points lie on a plane ($z_w = 0$), which makes it possible to recover $x_w$ and $y_w$. The 2D world coordinates of the frame's corners are obtained by casting rays from the four frame's corners (top of Figure 5). The 2D world coordinate that corresponds to a frame corner is defined as the point ($x_w$, $y_w$, 0) where the ray intersects the world plane ($z_w = 0$).

$$\begin{bmatrix} x_w \\ y_w \\ 0 \end{bmatrix} = \frac{(p_0 - l_0) \cdot n}{l \cdot n} \tag{5}$$

where $n = [0\ 0\ -1]^T$ is a normal vector to the world plane and $p_0 = [0\ 0\ 0]^T$ is a point on the world plane. $l$ is a vector in the direction of the ray and $l_0 = p^W$ is a point where the ray intersects the camera plane. $l$ is computed as follows:

$$l = \left| RA^{-1} \begin{bmatrix} x_f \\ y_f \\ 1 \end{bmatrix} \right| \tag{6}$$

Both the camera's extrinsic and intrinsic parameters are required for this operation. The extrinsic parameters (position and attitude of the camera) are provided by the EKF state vector. The camera intrinsic parameters are estimated using OpenCV's camera calibration tool[3]. The implementation is based on the work from [17, 18].

Now, a relation between the pixel coordinates and world coordinates is known. However, the 2D world coordinates ($mm$) need to be mapped to the corresponding 2D pixel coordinates ($x_m$, $y_m$) of the map's canvas. A canvas with a fixed resolution of 4.883 $mm/px$ is used.

$$\begin{bmatrix} x_m \\ y_m \end{bmatrix} = s_{map} \cdot \begin{bmatrix} x_w \\ y_w \end{bmatrix} \tag{7}$$

where $s_{map}$ = 1/4.884.

Now, the map's pixel coordinates of the frame corners are known (bottom of Figure 5). In order to transform a frame to the map's canvas, a relation between the frame's pixels and map's pixels is required. A transformation from the frame's corner pixel coordinates and the corresponding pixel coordinates of the map's canvas can be computed.

$$\begin{bmatrix} x_{m,i} \\ y_{m,i} \\ 1 \end{bmatrix} \sim H \begin{bmatrix} x_{f,i} \\ y_{f,i} \\ 1 \end{bmatrix} \tag{8}$$

The local perspective transformation $H$ is calculated by minimizing the back-projection using a least-squares algorithm. We used OpenCV's $s\ find\ Homography$[3] to find the perspective transformation. This transformation describes how each frame's pixel needs to be transformed in order to map to the corresponding (sub) pixel of the map's canvas. The transformation is used to warp the frame on the map's canvas. By warping the frame on a flat canvas, implicit perspective correction is applied to the frames. We used OpenCV's $warp\ Perspective$[4] to warp the frame on the map's canvas.

In this way a texture map can be built. This map consists of a set overlapping textures. The placement and discontinuities of the overlapping textures could be further optimized by map stitching, as described in [15]. Here the texture map is used for human navigation. For automatic navigation a feature map is used, as described in the next section.

### 3.3.2. Feature Map

In addition to the visual map described above, a grid of image features (feature map) is created using a method we have developed. This feature map will be used for localization purposes, as described in Section 3.4. The inertia measurements provide frequent position estimates. If the AR. Drone is able to relate a video frame to a position inside the feature map, the vehicle is able to correct this drift long enough to build a map as large as the indoor arena of the IMAV competition (as described in Section 4).

A 2D grid with a fixed resolution of 100 $\times$ 100 $mm$ per cell is used. From each camera frame we extract Speeded-Up Robust Features (SURF) [2] that are invariant with respect to rotation and scale. Each feature is an abstract description of an "interesting" part of an image (e.g., corners). A feature is described by a center point in image sub-pixel coordinates and a descriptor vector that consists of 64 floats.

[3] http://opencv.itseez.com/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

[4] http://opencv.itseez.com/modules/imgproc/doc/geometric_transformations.html

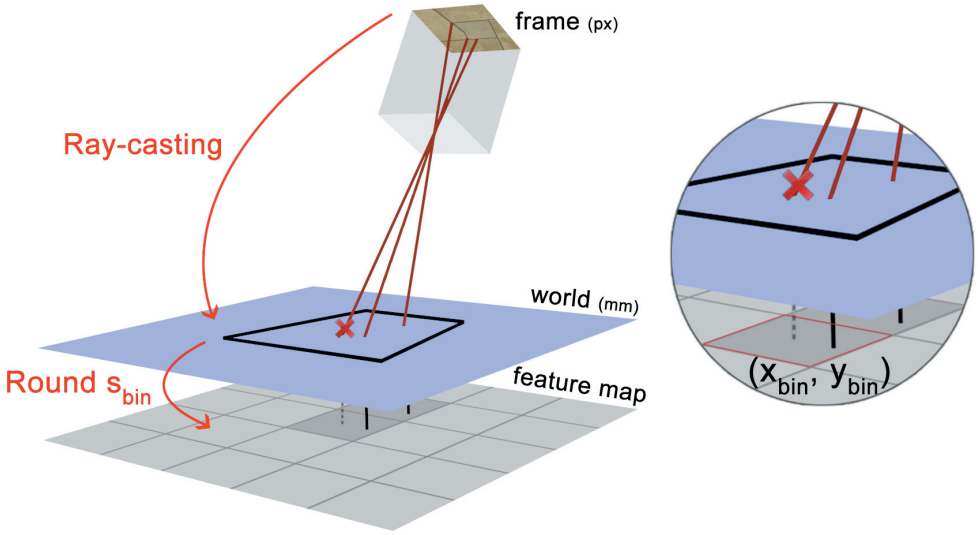


**Figure 6:** Feature map: adding visual features to a feature grid. Each feature found in the camera frame is mapped to the corresponding 2D world coordinates. The 2D world coordinates are mapped to the grid cell the feature belongs to. For each cell, only the best feature is stored.

Each feature that is detected in a camera frame is mapped to the corresponding cell of the feature map. A feature's center point ($x_f$, $y_f$) is transformed to its corresponding position in 2D world coordinates ($x_w$, $y_w$), visible in the top of Figure 6. This is done by casting a ray from the features pixel coordinates in the frame. The method is similar to the method used for casting ray's from the frame's corners (Section 3.3.1). Finally, the 2D world position ($x_w$, $y_w$) of each feature is transformed to the corresponding cell indices ($x_{bin}$, $y_{bin}$), visible in the bottom of Figure 6.

$$\begin{bmatrix} x_{bin} \\ y_{bin} \end{bmatrix} = Round(s_{bin} \cdot \begin{bmatrix} x_w \\ y_w \end{bmatrix}) \tag{9}$$

where $s_{bin} = 0.01$.

For each cell, only the best feature (e.g. with the highest response) is kept and the other features are dropped. If a cell already contains a feature descriptor ($grid_{x,y} \neq \emptyset$), the cell is ignored.

$$grid_{x,y} = \begin{cases} \arg\max_{d \in Dx,y} response(d) & \text{if } grid_{x,y} = \emptyset \\ grid_{x,y} & \text{else} \end{cases} \tag{10}$$

where $D_{x,y}$ is the set of features that is mapped to cell $x_{bin}$, $y_{bin}$.

The remaining (best) feature descriptors, including the corresponding world coordinates ($x_w$, $y_w$), are added to the corresponding grid cells.

## 3.4. Localization

The feature map created in the previous section can be used for absolute position estimates, at the moment of loop-closure (when the AR.Drone is above a location where it has been before). This allows to correct the drift that originates from the internal sensors of the AR.Drone. The inertia measurements provide frequent position estimates. However, the estimated position will drift over time, because the errors of the inertia measurements being accumulated. The feature map that is created can be exploited to reduce this drift, because localization against this map provides absolute positions of the vehicle.

These absolute positions are integrated into the EKF and improve the estimated position and reduce the covariance of the state.

When a camera frame is received, SURF features are extracted. Each feature consists of a center position in pixel coordinates ($x_f$, $y_f$) and a feature descriptor. A feature's center point ($x_f$, $y_f$) is transformed to its corresponding position in 2D world coordinates ($x_w$, $y_w$), visible in the top of Figure 6. This is done by casting a ray from the features pixel coordinates in the frame. The method is similar to the method used for casting ray's from the frame's corners (Section 3.3.1).

The next step is matching the feature descriptors from the camera frame against the feature descriptors from the feature map. When the feature map is quite large, this process becomes slow. However, the estimated position of the vehicle can be used to select a subset of the feature map. This can be done by placing a window that is centered at the vehicle's estimated position. The covariance of the estimated position can be used to determine the size of the window. The set of frame descriptors (query descriptors $D_q$) is matched against the map descriptors (training descriptors $D_t$). Matching is done using a brute force matcher that uses the $L^2$ norm as similarity measure. For each query descriptor $d_q$ from the frame, function $C(d_q)$ selects the training descriptor $d_t$ from the map that minimizes the $L^2$ norm:

$$C(d_q) = \arg \min_{d_t \in D_T} L2(d_q, d_t) \tag{11}$$

where $D_T$ is the set of map descriptors within a window around the estimated position. The $L2$ distance between two descriptors $a$ and $b$ is defined as:

$$L2(a,b) = \sqrt{\sum_{i=1}^{N} \left| a_i - b_i \right|^2} \tag{12}$$

where $N = 64$ is the length of the SURF descriptor vector.

Each query descriptor (frame) is matched against the descriptor from the training descriptors (map) that is most similar. Please note it is possible that multiple descriptors from the frame are matched against a single descriptor from the map.

For each match $C(d_q, d_t)$ the 2D world coordinates ($x_{w,d_q}$, $y_{w,d_q}$) and ($x_{w,d_t}$, $y_{w,d_t}$) of both descriptors are already computed. These point pairs can be used to calculate a transformation between the query points (frame) and training (map) points. This transformation describes the relation between the EKF estimated vehicle position (described in Section 3.2) and the position according to the feature map.

Different types of transformations can be used to describe the relation between the point pairs (e.g., perspective transformation or affine transformation). However, if not all of the point pairs fit the transformation (due to outliers), the initial transformation estimate will be poor. We use Random Sample Consensus (RANSAC) [19] to filter the set of matches in order to detect and eliminate erroneous matches (point pairs). RANSAC tries different random subsets of corresponding point pairs. It estimates the transformation using this subset and then computes the quality of the computed transformation by counting the number of inliers.

Since the point pairs are normalized, the perspective deformation and scale differences between the point pairs (matches) are already removed. The remaining degrees of freedom are the translation and rotation between the point pairs. Perspective transformation and affine transformation provide more degrees of freedom than required. This is potentially dangerous, because an incorrect transformation may still results in a high percentage of inliers. For example, this may happen when multiple query descriptors (frame) are matched against a single training descriptor (map). In that case, a perspective or affine transformation is found that scales all points to a single position. This transformation does not correspond with the actual translation and rotation, but is a valid transformation according to the RANSAC algorithm.

Instead of computing a full perspective, affine or Euclidean transformation, here only the translation in $x$ and $y$ direction is estimated. The rotation is estimated independently, as described at the end of this section. The translation is estimated by taking a subset of 3 random point pairs. The translation for those points is computed as follows:

$$T_x = \frac{\sum_i^N (x_{w,d_t,i} - x_{w,d_q,i})}{N} \tag{13}$$

$$T_y = \frac{\sum_i^N (y_{w,d_t,i} - y_{w,d_q,i})}{N} \tag{14}$$

where $N = 3$ is the number of point pairs.

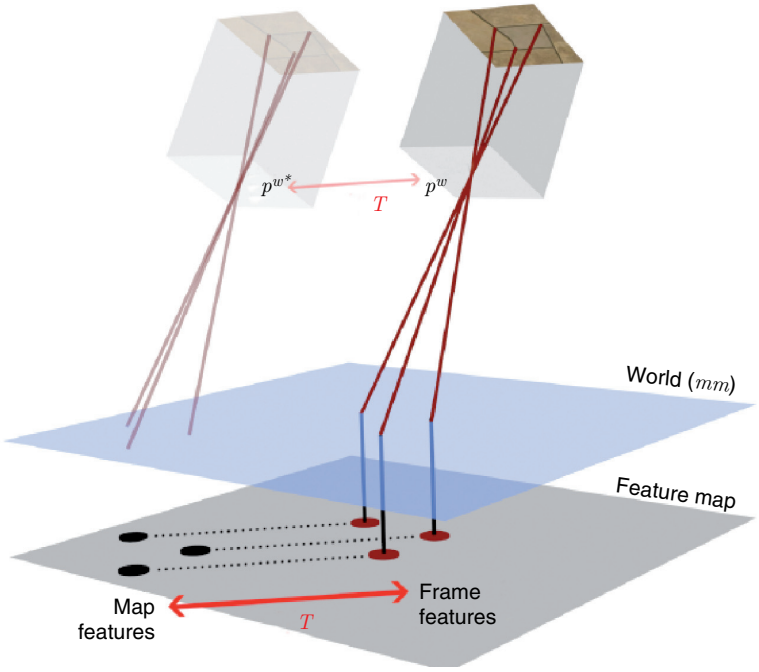


**Figure 7:** Localization: features from the camera frame (red circles) are matched against the features from the feature map (black circles). All the feature points (pixels) are converted to 2D word coordinates. The translation $T$ between both feature sets is calculated and used to correct the estimated position of the AR.Drone.

The standard deviation of translation $T$ is defined as:

$$\sigma_x = \sqrt{\sum_i^N ((x_{w,d_t} - x_{w,d_q}) - T_x)^2} \tag{15}$$

$$\sigma_y = \sqrt{\sum_i^N ((y_{w,d_t} - y_{w,d_q}) - T_y)^2} \tag{16}$$

The confidence $c$ of translation $T$ is computed using the standard deviation:

$$c_T = 1 - \sigma_x / \theta_d - \sigma_y / \theta_d \tag{17}$$

where $\theta_d = 200$.

After RANSAC has performed all iterations, the translation with highest confidence $T_{best}$ is used as final translation. A big advantage of this two degrees of freedom approach is that it is very efficient, allowing a great number of RANSAC iterations to find the optimal subset of point pairs. If the

confidence $c_T$ of the translation $T_{best}$ exceeds a threshold, the transformation $T$ is added to the estimated position $p^W$ to compute the corrected position $p^{W*}$. This corrected position is integrated in the EKF as measurement with low covariance. Please note this method requires that the estimated yaw of the vehicle is close to the actual yaw. A large difference between estimated and actual yaw results in low confidence $c_T$. Localization on regular basis should sufficiently correct errors in the estimated yaw.

If the confidence is exceptionally high (i.e., good matches are found), the selected point pairs are used to determine the rotation between the estimated yaw and the real yaw of the vehicle. This rotation is used to correct the drift in yaw. The rotation is computed using Kabsch algorithm [20]. This method computes the optimal rotation matrix that minimizes the RMSD (root mean squared deviation) between two paired sets of points. Like the translation, the rotation is added to the estimated rotation to calculate the correct rotation. This rotation is directly written to the EKF state.

This localization method allows us to exploit the feature map created in Section 3.3.2 and get absolute position estimates which can be used to autonomously navigate the AR.Drone, as demonstrated in the next section.

# 4. RESULTS

## 4.1. Validation of the Simulation Model

To evaluate the USARSim simulation model, a set of maneuvers is flown with the actual AR.Drone and simulated AR.Drone. The differences between the maneuvers are studied in detail. To enable multiple repetitions of the same maneuver it is described as a set of time points (milliseconds since initialization) each coupled to a movement command. We wrote wrappers for the AR.Drone programming interface and for USARSim interface which read these scripts and output a control signal, using the system clock to manage the timing independently from the game engine and the AR.Drone hardware. Orientation, altitude and horizontal speed are recorded at a frequency of 200 Hz during the maneuvers. These are gathered through the AR.Drone's internal sensors and the build-in algorithms, which are also used by the controller to operate the drone. The filtered output of the MEMS gyroscope is used for estimating orientation. The filtered output of the ultrasound distance sensor is used for estimating altitude. The optical flow algorithm using the bottom camera is used for estimating the horizontal (linear and lateral) speeds. The simulator has equivalent sensors. In addition, simulation can provide ground-truth data. Also for the real maneuvers an attempt was made to generate ground truth via an external reference system; the movements were recorded with a synchronized video system consisting of four firewire cameras, capturing images at 20 frames per second at a resolution of $1024 \times 768$ pixels. The position of the AR.Drone in each frame has been annotated by hand.

Corresponding to NIST guidelines [21] a set of experiments of increasing complexity was performed. For the AR.Drone four different experiments were designed. The first experiment is a simple hover, in which the drone tries to maintain its position (both horizontal and vertical). The second experiment is linear movement, where the drone actuates a single movement command. The third experiment is a small horizontal square. The last experiment is a small vertical square.

### 4.1.1. Hovering

Quadrotors have hovering abilities just like a helicopter. The stability in maintaining a hover depends on environmental factors (wind, underground, aerodynamic interactions) and control software. If no noise model is explicitly added, the USARSim model performs a perfect hover; when no control signal is given the horizontal speeds are zero and the altitude stays exactly the same.

For the AR.Drone, this is a good zero-order model. One of the commercial features of the AR.Drone is its ease of operation. As part of this feature it maintains a stable hover when given no other commands, which is accomplished by a visual feedback loop. So, the hovering experiment is performed indoors with an underground chosen to have enough texture for the optical flow motion estimation algorithm.

As experiment the AR.Drone maintains a hover for 35 seconds. This experiment was repeated 10 times, collecting 60.000 movement samples for a total of 350 seconds. Over all samples the mean absolute error in horizontal velocity (the Euclidean norm of the velocity vector) is 0.0422 $m/s$ with a sample variance of 0.0012 $m^2/s^2$. From the samples we obtain the distribution of the linear and lateral velocity components.

From the velocity logs the position of the AR.Drone during the 35-second flight was calculated. The mean absolute error of the horizontal position is $0.0707m$ with a sample variance of $0.0012m^2$.

### 4.1.2. Horizontal Movement

In this experiment the AR.Drone is flown in a straight line. It is given a control pulse (i.e., pitch) with a constant signal for 5 different time periods: 0.1s, 1s, 2s, 3s, and 5s. Each pulse is followed by a null signal for enough time for the AR.Drone to make a full stop and a negative pulse of the same magnitude for the same period, resulting in a back and forth movement. In Figure 8 the red line shows the control signal, the blue line the response (i.e., velocity) of the AR.Drone. The experiment was repeated for 5 different speeds. The control signal $s$ specifies the pitch of the AR.Drone as a factor (between 0 and 1) of the maximum absolute tilt $\theta_{max}$, which was set to the default value[5]. The trials were performed with the values of 0.05, 0.10, 0.15, 0.20, 0.25 for the control signal $s$.

Robots in USARSim are controlled with a standardized interface, which uses SI units. A robot in USARSim expects a DRIVE command with a speed in $m/s$ and not the AR.Drone native signal $s$. Thus in order to fly comparable trials the relation between the AR.Drone's angle of attack $\alpha$ and the corresponding velocity $v$ has to be investigated. When flying straight forward in no-wind conditions, the angle of attack $\alpha$ is equivalent with the pitch $\theta$. In order to do this the samples from the logs where the AR.Drone has achieved maximum velocity have to be selected. Closer inspection of the velocity logs show that in each trial there is still constant increase of velocity for the first three pulses. For the last two pulses there is obvious plateauing, which indicates that the last two seconds of the five-second pulses is a good indication for the maximum velocity. Therefore the velocity at those last two seconds was used to compute mean absolute speeds $\overline{v}$, which are combined with the mean absolute pitch $\overline{\theta}$ as measured by the MEMS gyroscope. The estimates for $\overline{v}$ and pitch $\overline{\theta}$ are presented in Table 1 together with their standard deviation. Extrapolating the mean pitch $\overline{\theta} \simeq 7.5°$ at control value $s = 0.25$ to the maximum control signal gives an indication of the AR.Drone's maximum pitch $\theta_{max} \simeq 30°$ value. For typical usage, the angle of attack never exceeds 12° degrees.

To convert the AR.Drone's control signal $s$ to USARSim commands $v$ a least-squares fit through the points of Table 1 is made for the linear function $v = c \cdot \theta$, which gives us $c = 0.2967$. Equation 18 gives the final conversion of a control signal $s$ to a velocity $v$ in $m/s$ given the AR.Drone's maximum pitch $\theta_{max}$ in degrees.

$$v = 0.2967 \cdot s \cdot \theta_{max} \tag{18}$$

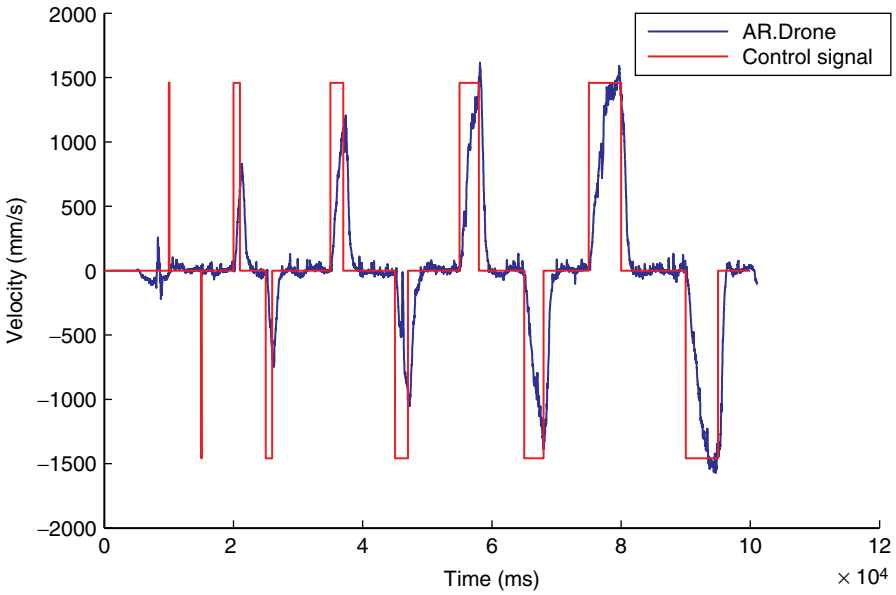


Figure 8: Velocity of the real AR.Drone (blue) on a number of control pulses (red) with a pitch of s = 0.15 × θmax.

[5] ARDrone firmware (1.3.3)

Table 1: Averaged velocity $\bar{v}$ measured at the end of a 5 seconds pulse of the control signal $s$, including the corresponding pitch $\bar{\theta}$ as measured by the gyroscope.

| | **Control signal $s$** | | | | |
|---|---|---|---|---|---|
| | 0.05 | 0.10 | 0.15 | 0.20 | 0.25 |
| $\bar{v}$ (m/s) | 0.4044 | 0.6284 | 1.4427 | 1.7587 | 2.2094 |
| $\sigma_v$ (m/s) | 0.096 | 0.226 | 0.070 | 0.126 | 0.165 |
| $\theta$ (deg) | 1.4654 | 2.9025 | 4.1227 | 5.7457 | 7.4496 |
| $\sigma_\theta$ (deg) | 0.455 | 0.593 | 0.482 | 0.552 | 0.921 |

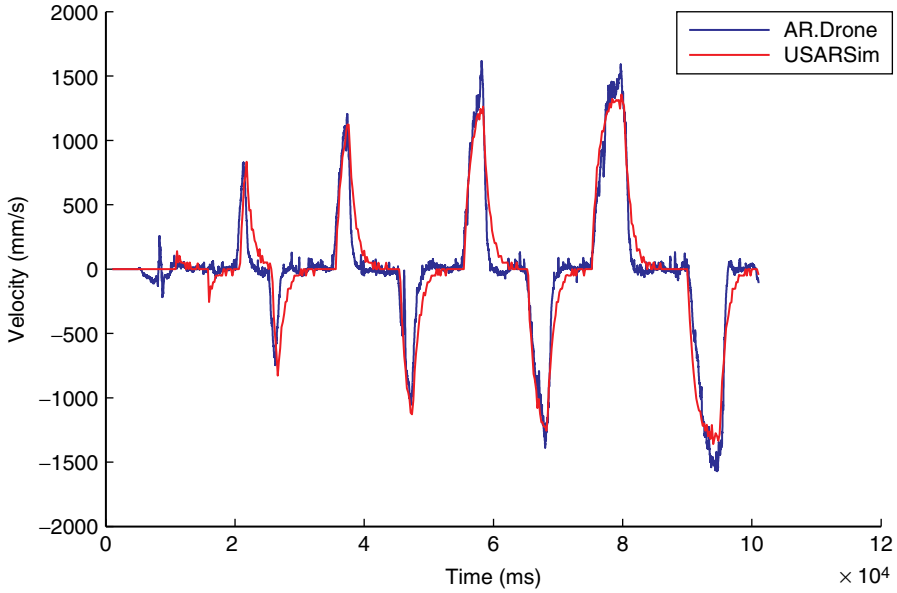


**Figure 9:** Velocity of the real (blue) and simulated AR.Drone (red) on the same control pulses as shown in Figure 8.

The USARSim model has a parameter $P_\theta$ for calculating the angle (radian) given the velocity, which is the value $\hat{P}_\theta = 0.057$, as used in subsequent simulations.

The next experiment checks the acceleration of the real and simulated AR.Drone. First we give an estimate of how quickly the AR.Drone's controller changes its pitch to match the commanded pitch and how well it can keep it. For this we select all samples from 100 ms after the start of the 2s, 3s and 5s pulses till the first sample at which the commanded pitch has been reached. This corresponds to the time-span between which the AR.Drone has started to act on the change in the control signal until it reaches the commanded pitch. The result is illustrated in Figure 9.

As one can see, the acceleration has for the real and simulated AR.Drone nearly the same slope. The deceleration of the simulated AR.Drone is slightly lower. In the real AR.Drone the feedback loop (based on the optical flow of the ground camera) actively decelerates the system. Overall, the dynamic behavior of the simulator closely resembles the dynamic behavior of the real system. Additionally, tests with more complex maneuvers (horizontal and vertical square) have been recorded, but unfortunately not yet analyzed in detail.

## 4.2. Mapping

An experiment has been carried out to evaluate the mapping approach. This experiment is performed with the AR.Drone and the simulated AR.Drone using USARSim.

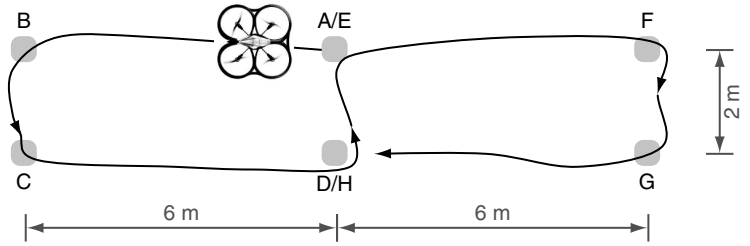


**Figure 10:** Top view of the experiment's floorplan. The AR.Drone flew an 8-shape. The labels A-G represent six landmarks that provide ground truth to evaluate the accuracy of a map.

A sports hall is used as indoor testing environment. This environment closely resembles the environment of the IMAV2011 indoor competitions. Six magazines are laid out on the floor at the locations A, B, C, D, F, G in Figure 10. These magazines are used as landmarks that are easily recognizable in the map. The distances between these landmarks are known, providing ground truth. The AR.Drone flew in an 8-shape above the floor to capture an intermediate loop (point A and E in Figure 10). In addition to the landmarks, the lines on the floor provide visual clues about the overall quality of a created map. Care has been taken to mimic the gym floor in the simulated environment. The floor of the simulated environment consists of an image taken at the IMAV2011 competitions. The image was taken from the gym tribune and warped with an image editor to remove the perspective view as much as possible. Unfortunately, some warping artifacts are visible on the simulated gym floor.

The mapping method is performed on the floor as described above. The distances between the landmarks inside the generated map (red lines) are compared to the know ground truth to compute the error of the map.

The results of this experiment can be found in Table 2 and Figures 11 (real AR.Drone) and 12 (simulated AR.Drone). Both the simulated and real AR.Drone produce a map with enough quality for navigation in the IMAV Pylon challenge. The visual map could be further optimized by a method equivalent to TORO [3]. The visual map created by the simulated AR.Drone contains fewer errors than the map of the real AR.Drone. Despite the simulator's inertia measurements are noise-free, there are

Table 2: Accuracy of the visual map by measuring the distance between landmarks. The maps are created using the AR.Drone (Figure 11) and the USARSim simulator (Figure 12).

| **Landmarks** | **A-B** | **B-C** | **C-D** | **D-E** | **E-F** | **F-G** | **B-G** | **C-F** |
|---|---|---|---|---|---|---|---|---|
| **AR.Drone** | | | | | | | | |
| mean error (m) | 0.042 | 0.653 | 0.087 | 0.726 | 0.350 | 0.579 | 0.491 | 0.388 |
| error (%) | 0.71 | 32.65 | 1.44 | 36.28 | 5.84 | 28.95 | 4.03 | 3.19 |
| **USARSim simulator** | | | | | | | | |
| mean error (m) | 0.280 | 0.095 | 0.319 | 0.010 | 0.172 | 0.021 | 4.566 | 0.496 |
| error (%) | 4.66 | 4.74 | 5.31 | 4.98 | 2.85 | 1.1 | 3.75 | 4.1 |

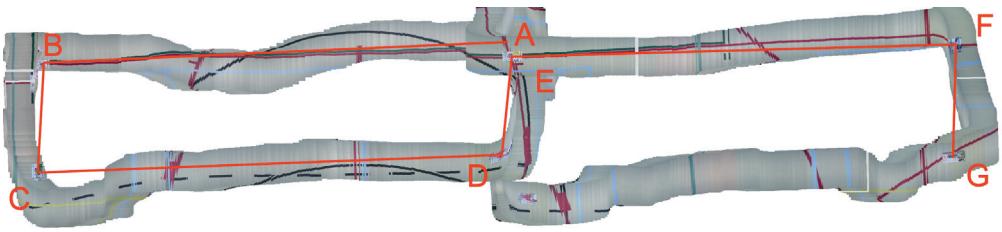


**Figure 11:** Visual map created by the mapping method. Camera images are taken by the AR.Drone, flying at approximately 70 cm altitude.

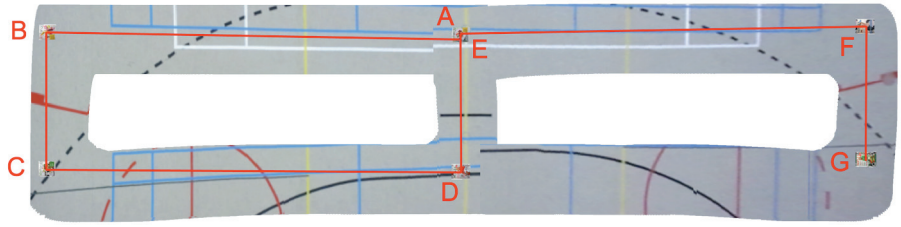


**Figure 12:** Visual map created by the mapping method. Camera images are taken by the simulated AR.Drone, flying at approximately 85 cm altitude.

small errors in the map. These errors are due to integrating the acceleration measurements to velocities and positions. The largest errors are in the long stretches of the trajectory, because these parts of the trajectory are longer affected by integration errors due to the larger traveling times.

The position of the real AR.Drone was estimated using the AR.Drone's velocity estimates produced by the proprietary onboard filter. This estimate is based on inertia measurements and optical flow obtained from the relative motion between camera frames. These measurements suffer from noise, resulting in larger map errors. The AR.Drone's proprietary optical flow algorithm is probably having difficulties with the gym floor. When flying along a single line or parallel lines (e.g. path B-C in Figure 11), no apparent motion can be observed from the camera frames. These circumstances mainly occured when flying along the short stretches in the trajectory, producing the largest errors in this direction. This behavior is not observed for the simulated AR.Drone, because it lacks an optical flow method to estimate the velocity.

### 4.3. Localization

Another experiment has been carried out to evaluate the localization approach. This experiment resembles quite closely the IMAV Pylon Challenge. This experiment is performed with the simulated AR.Drone only to demonstrate the possibilities for localization when a reasonable accurate visual map (errors in the order of 5%, as found in the previous section) is available. The USARSim simulator provides accurate ground truth that is used to evaluate the performance.

For this experiment, the trajectory from the mapping experiment (Figure 10) is repeated several times. Eleven 8-shapes (called rounds) were flown autonomously. During the first round, the map is created. The remaining 10 rounds are flown with mapping turned off and localization turned on. The $x$, $y$ and $z$ distance between the estimated location and real location (ground truth) is measured and

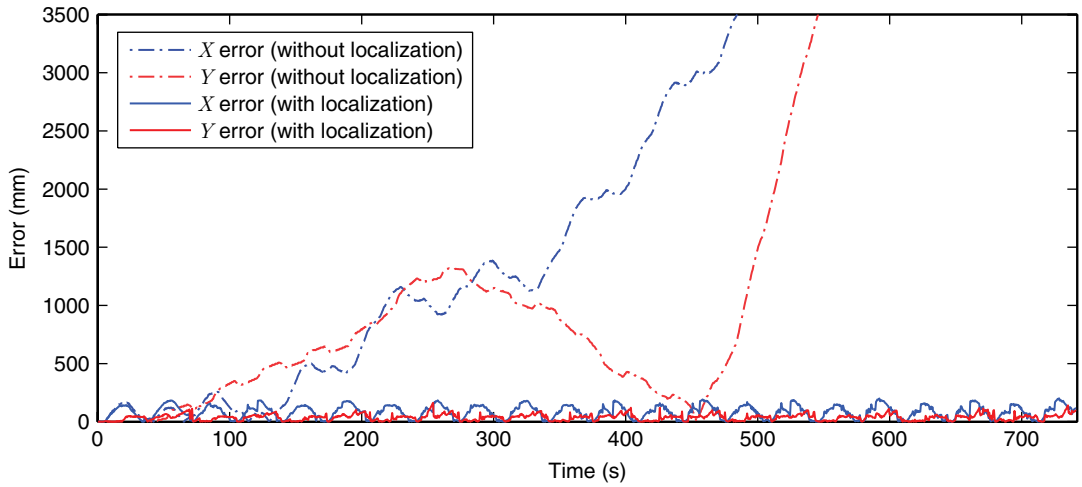


**Figure 13:** Error between the estimated position and actual position (ground truth). The experiment is performed with visual localization (solid lines) and repeated without visual localization (dashed lines).

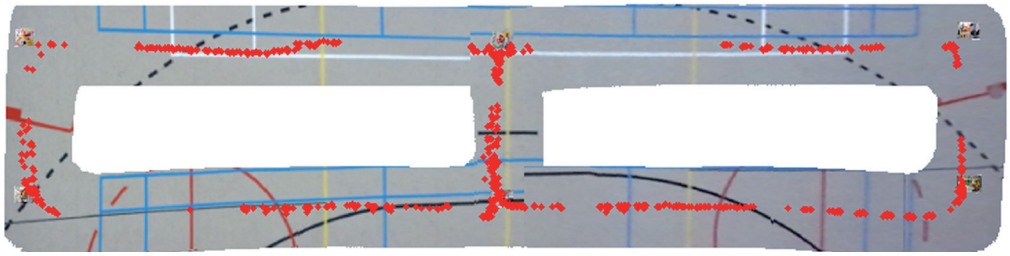

**Figure 14:** Visual map with localization positions. Each red dot represents a position of the AR.Drone where visual localization was performed.

used as error measure. This experiment is repeated with localization turned off, meaning that the location is estimated using the inertia measurements only.

The results of this experiment can be found in Figure 13. Without localization, the error of the estimated position increases over time (dashed lines). Around 280 seconds the drift in Y direction seems to reverse. This drift in reversed direction is causing a decreasing Y error until the error is completely compensated. However, the drift is continuing and the error increases again. With localization turned on, the error remains low during the whole flight. The first 75 seconds correspond to the first 8-shape that is used create a map. During this mapping flight, the error of the estimated position increases and the quality (accuracy) of the map decreases. Again, reversed drifting causes a temporary drop in error around 35 seconds. The repetitive error pattern suggests that the error of the estimated position largely depends on the quality (accuracy) of the map, if localization can be performed regularly.

Figure 14 indicates the positions where the AR.Drone was able to localize itself. Localization was performed at almost every position of the 8-shape trajectory. Not only the magazines, but also the lines on the gym floor provide sufficient information for localization. At some positions localization was not possible, because the camera observed insufficient image features. For example, when lines without intersections are observed. These results confirm that the localization method is able to deal with circumstances encountered during the IMAV competition.

## 5. CONCLUSION

The validation effort of the hovering and the forward movement shows that the dynamic behavior of the simulated AR.Drone closely resembles the dynamic behavior of the real AR.Drone. Further improvement would require more system identifications (e.g., rotational movements, wind-conditions) and include the characteristics of the Parrot's proprietary controller into the simulation model. Our research would benefit from a realistic modeling of the noise aggregation in the inertia sensor.

The mapping method is able to map areas visually with sufficient quality for both human and artificial navigation purposes. Both the AR.Drone and USARSim simulator can be used as source for the mapping algorithm. The visual map created by the simulated AR.Drone contains fewer errors than the map of the real AR.Drone. The difference can be explained by the measurement noise produced by the real AR.Drone. The optical flow algorithm in the AR.Drone's firmware should improve the accuracy, but is probably having difficulties with the gym floor.

The localization method is able to significantly reduce the error of the estimated position when places are revisited. The lines on the gym floor provide sufficient information for robust localization. It was demonstrated that autonomous navigation is possible by visual localization and mapping, with the natural texture of the IMAV indoor competition. This removes the reliance on the detection of the artificial landmarks in the IMAV Pylon challenge.

## 6. FUTURE WORK

Future work will use additional data sources to improve the accuracy of the estimated position. The vehicle's motion can be computed from consecutive video frames. Our previous paper [15] describes a method for this. A loop-closure method can be used to optimize the map when places are revisited. Other future work is building an elevation map using the sonar's altitude measurements. Merging it with the texture map results in a textured 3D map of the floor, similar to [1]. This map can be used to help ground robots navigate.

A further improvement would be to include the images of the high-resolution front camera into the process. Those images can be used to calculate the optical flow from monocular stereo vision, which serves as the basis for both creating a disparity map of the environment. The disparity map can be used to detect obstacles ahead, which can be used in autonomous navigation. When combined with a time to contact method, the measurements can be used for a coarse 3D reconstruction of the environment and another source for estimating the egomotion.

Once the visual map exists, this map can be extended with range and bearing information towards landmarks (as the pylons from the IMAV challenge). On close distance the bearing information could be derived directly from perception. By modeling this as an attractive force, the correct heading could be spread over the map by value iteration. In addition, obstacles visible in the disparity map of the front camera could be used as basis for a repulsive force. Both forces could be combined to a force field that guides a robot on the map. The availability of a realistic simulation model will contribute to the training of these machine-learning approaches.

## ACKNOWLEDGEMENTS

We like to thank Parrot S.A. for providing an AR.Drone for the competition. This research is partly funded by the EuroStars project 'SmartINSIDE'. We like to thank Carsten van Weelden for his experiments to validate the motion model of the AR.Drone. Martijn van der Veen and Robrecht Jurriaans took the initiative to compete in the IMAV competition and did both independent research on obstacle avoidance [22] and force field navigation [23].

## REFERENCES

[1] B. Steder, G. Grisetti, C. Stachniss, and W. Burgard. Visual SLAM for flying vehicles. *Robotics, IEEE Transactions on*, 24(5): 1088–1093, 2008.

[2] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3): 346–359, 2008.

[3] G. Grisetti, S. Grzonka, C. Stachniss, P. Pfaff, and W. Burgard. Efficient estimation of accurate maximum likelihood maps in 3d. In *IROS*, pages 3472–3478, San Diego, CA (USA), 2007.

[4] F. Caballero, L. Merino, J. Ferruz, and A. Ollero. Unmanned Aerial Vehicle Localization Based on Monocular Vision and Online Mosaicking. *Journal of Intelligent and Robotic Systems*, 55(4): 323–343, 2009.

[5] Abraham Bachrach, Ruijie He, and Nicholas Roy. Autonomous flight in unknown indoor environments. *International Journal of Micro Air Vehicles*, l(4): 217–228, December 2009.

[6] John Stowers, Andrew Bainbridge-Smith, Michael Hayes, and Steven Mills. Optical flow for heading estimation of a quadrotor helicopter. *International Journal of Micro Air Vehicles*, 1(4): 229–239, December 2009.

[7] E. Jovanov, M. Milosevic, R. Tilly, M. Truex, and C. Jones. Autonomous Personnel Tracking Help AR.Drone. In *International Micro Air Vehicle Conference*, May 2011.

[8] Cooper Bills, Joyce Chen, and Ashutosh Saxena. Autonomous mav flight in indoor environments using single image perspective cues. In *International Conference on Robotics and Automation (ICRA)*, 2011.

[9] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys. Pixhawk: A system for autonomous flight using onboard computer vision. In *IEEE Int. Conf. Robotics and Automation* (*ICRA*'11), May 2011.

[10] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar. The grasp multiple micro-uav testbed. *Robotics Automation Magazine*, *IEEE*, 17(3): 56–65, sept. 2010.

[11] Workshop on robots, games, and research: Success stories in usarsim. In Stephen Balakirky, Stefano Carpin, and Mike Lewis, editors, *Proceedings of the International Conference on Intelligent Robots and Systems (IROS* 2009*)*. IEEE, October 2009.

[12] Stefano Carpin, Jijun Wang, Michael Lewis, Andreas Birk, and Adam Jacoff. High fidelity tools for rescue robotics: Results and perspectives. In Ansgar Bredenfeld, Adam Jacoff, Itsuki Noda, and Yasutake Takahashi, editors, *RoboCup* 2005*: Robot Soccer World Cup IX*, volume 4020 of *Lecture Notes in Computer Science*, pages 301–311. Springer Berlin / Heidelberg, 2006.

[13] T. Yechout, S. Morris, D. Bossert, and W. Hallgren. *Introduction to Aircraft Flight Mechanics: Performance, Static Stability, Dynamic Stability, and Classical Feedback Control*. American Institute of Aeronautics and Astronautics, Reston, VA, 2003.

[14] Gabriel M. Hoffmann, Haomiao Huang, Steven L. Wasl, and Claire J. Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. In *Proc. of the AIAA Guidance, Navigation and Control Conference*, 2007.

[15] Arnoud Visser, Nick Dijkshoorn, Martijn van der Veen, and Robrecht Jurriaans. Closing the gap between simulation and reality in the sensor and motion models of an autonomous ar.drone. In *Proceedings of the International Micro Air Vehicle Conference and Flight Competition (IMAV*11*)*, September 2011.

[16] S.J. Julier and J.K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 92(3): 401–422, 2004.

[17] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22: 1330–1334, 2000.

[18] Jean-Yves Bouguet. *Visual methods for three-dimensional modeling*. PhD thesis, California Institute of Technology, May 1999.

[19] M.A. Fischler and R.C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6): 381–395, 1981.

[20] W. Kabsch. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A*, 32(5): 922–923, Sep 1976.

[21] Adam Jacoff, Elena Messina, Hui-Min Huang, Ann Virts, and Anthony Downs. Standard Test Methods for Response Robots. ASTM International Committee on Homeland Security Applications, January 2010. subcommittee E54.08.01.

[22] Robrecht Jurriaans. Flow based obstacle avoidance for real world autonomous aerial navigation tasks. Bachelor's thesis, Universiteit van Amsterdam, August 2011.

[23] Martijn van der Veen. Optimizing artificial force fields for autonomous drones in the pylon challenge using reinforcement learning. Bachelor's thesis, Universiteit van Amsterdam, July 2011.