Vision techniques for on-board detection, following and mapping of moving targets

Petr Štěpán
Department of Cybernetics
Czech Technical University in Prague
stepan@fel.cvut.cz

Tomáš Krajník
Department of Computer Science
Czech Technical University in Prague
tomas.krajnik@fel.cvut.cz

Matěj Petrlík
Department of Cybernetics
Czech Technical University in Prague
petrlmat@fel.cvut.cz

Martin Saska
Department of Cybernetics
Czech Technical University in Prague
martin.saska@fel.cvut.cz

Abstract

This article presents computer vision modules of a multi-unmanned aerial vehicle (UAV) system, which scored gold, silver, and bronze medals at the Mohamed bin Zayed International Robotics Challenge (MBZIRC) 2017. This autonomous system, which was running completely on-board and in real-time, had to address two complex tasks in challenging outdoor conditions. In the first task, an autonomous UAV had to find, track, and land on a human-driven car moving at 15 km/h on a figure-eight-shaped track. During the second task, a group of three UAVs had to find small colored objects in a wide area, pick them up, and deliver them into a specified drop-off zone.

The computer vision modules presented here achieved computationally efficient detection, accurate localization, robust velocity estimation, and reliable future position prediction of both the colored objects and the car. These properties had to be achieved in adverse outdoor environments with changing light conditions. Lighting varied from intense direct sunlight with sharp shadows cast over the objects by the UAV itself, to reduced visibility caused by overcast to dust and sand in the air. The results presented in this paper demonstrate good performance of the modules both during testing, which took place in the harsh desert environment of the central area of United Arab Emirates, as well as during the contest, which took place at a racing complex in the urban, near-sea location of Abu Dhabi. The stability and reliability of these modules contributed to the overall result of the contest, where our multi-UAV system outperformed teams from world-leading robotic laboratories in two challenging scenarios.

1 Introduction

Advances in digital electronics meeting Moore's law allowed reliable autonomous operation of multi-rotor vehicles, which quickly became the most popular Unmanned Aerial Vehicles (UAVs) used in the scientific community. Compared to single- or double- rotorcraft, their mechanics are simpler. This makes their miniaturization easier and allows construction of small UAVs that can be fully autonomous and carry onboard all sensors and computational equipment. Therefore, using simple light cameras and advanced computer

vision approaches is currently one of the most promising directions being followed by the robotic community to achieve autonomous flight operations. In March 2017, the Mohamed Bin Zayed International Robotics Challenge (MBZIRC) 2017 was organized by the Khalifa University of Science in Abu Dhabi. The event aimed to accelerate the research of autonomous (multi) UAV systems, and the vision-based approaches were the principal sources of information for the autonomous flight. In this article, all vision techniques used by our team within the MBZIRC event are summarized. The MBZIRC competition had a significant impact on the robotics community, particularly within the field of UAVs, due to the ambitiously selected robotic challenges on the edge of current state-of-the-art chosen by a board of top scientists[1]. Two of the MBZIRC challenges, motivated by current industry needs, were designed for UAVs: autonomous landing on a moving vehicle and a task requiring a team of UAVs to collaborate to search, locate, track, collect, and relocate a set of static and moving objects.

The MBZIRC competition could be considered as a relevant and objective benchmark of these tasks, which are currently being solved by the robotic community since the teams had to achieve the given goal after only a few minutes of preparation before each trial. No repeated attempts due to technical difficulties or other problems, which is the standard practice in most of the robotics experiments, were allowed. Furthermore, the systems' robustness were thoroughly tested in an outdoor environment with changing environmental conditions. Out of 143 registered groups from almost all of the best robotic laboratories worldwide, 25 top teams were selected after several preliminary rounds to compete in the final competition in Abu Dhabi in March 2017[2]. The MBZIRC results verified the difficulty of the selected challenges and confirmed that the goals are at the cutting edge of the current robotic research. Out of the 25 participants, only two teams were able to land precisely in both final trials of the first challenge, and only four teams were able to transport at least one object into the desired location in the autonomous mode. The vision systems introduced in this paper were successfully employed in both of these challenges. The reliability and robustness of the object search and localization were demonstrated by the best score of our system in all trials of the Challenge 3. In the task of landing on a moving car, the robustness of the vision system was proven with successful landings in all three trials. These landings were all conducted in autonomous mode, which was a unique result in the competition. Moreover, the ability of fast car detection and landing was verified by the fastest attempt (25.1 s during Grand Challenge) achieved by our team, while the best times of other competitors were 42.3 s (Grand Challenge, University of Bonn) and 63.4 s (Beijing Institute of Technology, Landing Challenge).

In this article, we introduce vision-based techniques designed for the MBZIRC challenges. The first introduced system is capable of repeatable fully autonomous landing on a 15 km/h moving platform in an outdoor arena with the wind reaching $10-20$ km/h while achieving precision in the tens of centimeters. The second method was designed for a "Treasure hunt" challenge, where a fleet of three UAVs was able to cooperatively and entirely autonomously search for small objects in the challenging desert environment and collect them with an onboard gripper with centimeter precision.

The computer vision system described consists of two separate pipelines. Some steps of these two pipelines perform similar functions and thus are redundant. However, this redundancy was intended because if some components of one pipeline would exhibit insufficient performance during the contest itself, they could be easily substituted by components from the other one. The first pipeline is aimed to detect a pattern, composed of a black circle with an inscribed cross on a white background. The challenge rules specified the exact pattern parameters in advance and ensured that this high-contrast pattern with known dimensions could appear in the contest arena only once – on a moving vehicle for landing. Thus, the design and implementation of the first pipeline could be tailored specifically for this known pattern. The second processing pipeline was responsible for the 'Treasure hunt', where it had to perform colored object detection, tracking, and mapping. The exact parameters of these objects, e.g., shape, size, and score indication, were specified by the organizers during later stages of the contest preparation, and therefore, the individual processing steps of this pipeline had to rely on universal methods, which are easy to adjust to a variety of different object shapes and types. In particular, the original colored object detection pipeline assumed cuboid objects with

[1] `https://www.mbzirc.com/committee`

[2] Results of team CTU/UPENN/UOL from this qualification process can be found at `http://mrs.felk.cvut.cz/projects/mbzirc`

the score inscribed on their top in white Arabic numbers. However, shortly before the actual contest, the pipeline had to be changed to detect planar circular objects, with scores defined by their color.

1.1 Contribution beyond the state-of-the-art in UAV

Before the MBZIRC competition, few solutions for autonomous landing with visual feedback were described in literature (Saska et al., 2012),(Lange et al., 2009). Most of these systems were capable of landing only on a static or slowly moving pattern (Saripalli et al., 2003), (Fu et al., 2016), (Kim et al., 2016), and only in laboratory conditions, without the presence of wind and with stable light conditions. Our system has to perform a quick landing on a rapidly-moving vehicle, which requires the UAV to pitch and roll in angles sometimes exceeding 40 degrees. To prevent loss of landing target sight, we use a camera with a large field of view (FOV), fast frame rate, and high resolution.

The visual module for following and landing is presented in a growing body of literature (Hoang et al., 2017), (Gomez-Balderas et al., 2013), (Borowczyk et al., 2017), (Xu and Luo, 2016), (Lin et al., 2017), (Kim et al., 2014). In (Hoang et al., 2017) a system for ground vehicle following, based on Scale Invariant Feature Transform (SIFT) (Lowe, 2004), is described. In our case, the requirement of high resolution and framerate, along with the limited onboard computational power, prevent the use of methods like SIFT. In (Gomez-Balderas et al., 2013) the target detection is straightforward because the target is a red square in an indoor environment with constant, controlled lighting conditions. In (Borowczyk et al., 2017) the GPS and AprilTag (Olson, 2011) is used. This combination enables precise localization of the UAVs in the range of up to 5 m from landing platform. Similarly, (Xu and Luo, 2016) is using AprilTag achieved good performance for mutual localization of ground vehicle and UAV. This approach cannot be used for MBZIRC because the ground vehicle is marked by the circle with the cross inside. However, the method described in (Gomez-Balderas et al., 2013) and methods with AprilTag in (Borowczyk et al., 2017) and (Xu and Luo, 2016) use the Canny edge detection algorithm as the first stage of image processing. This is troublesome outdoors, where the presence of shadows and clutter causes the Canny method to detect a high number of edges, leading to excessive computational overhead in subsequent processing (Lightbody et al., 2017).

The authors of (Lin et al., 2017) present a similar approach for target detection, which is based on image segmentation and line detection to detect a landing pattern with the letter H inside a circle. To deal with the rapid movement of the target, our approach uses camera lenses with a FOV of more than 180° and a high frame rate camera with a global shutter. However, usage of lenses with this FOV requires specific techniques that deal with the image distortion. To reduce distortion we implemented an undistortion calculation method which is 20 times faster than the OpenCV fish-eye undistortion function, see Section 2.2.

In (Kim et al., 2016) and (Kim et al., 2014) an omnidirectional camera is used to detect the target platform. These articles describe the approach of team KAIST from MBZIRC competition, but only from their preliminary stage of development, when the red square was used as a target. Our approach uses more efficient computation of undistorted points and is more robust in the final detection.

The contribution of this paper is to present a solution that was very successful in the MBZIRC competition. The vision systems are a critical part of the whole control system, and the success of the system depends on thorough testing in realistic conditions. The requirements put on of vision-in-the-loop systems for autonomous flight are beyond efficiency metrics traditionally used in computer vision community, where detection is commonly evaluated using precision/recall on standardized datasets. In a contest-based evaluation, the methods have to satisfy real-time constraints on computationally restrained devices, to analyze images from low-end cameras, and to integrate results with sensory data, which are subject to glitches, delays, and hard-to-model noise. Furthermore, these methods have to deliver satisfactory results across a wide range of environmental conditions, which are often not captured in the training data. The quality and reliability of the presented approaches have been verified during the MBZIRC event by direct and fair comparison in equal conditions with tens of rivaling methods designed by reputable robotic laboratories worldwide. The solutions


Figure 1: The landing pattern from the UAV's perspective. Note the shadows cast by the UAV itself.

proposed here enable detection of a target object, precise estimation of its relative position and velocity, and accurate prediction of the target motion. The most important property of the proposed approaches is their computational efficiency and robustness in demanding outdoor environments with adverse illumination conditions. To facilitate the use of our vision system by other researchers, we provide their source codes as well as datasets collected during the contest at `http://github.com/gestom/MBZIRC_2017_vision`.

2 Landing pattern detection

The primary goal of landing pattern detection is to detect the target quickly and robustly. The target speed was high – 15 km/h (or 4.2 m/s) and considering the speed of the UAV (up to 25 km/h, i.e. 6.9 m/s) the computer vision system has to be able to deal with object's speed exceeding 10 m/s. Our tests indicated that UAV-mounted rolling-shutter cameras are subject to the 'jelly' effect (see Figure 9), which had a negative impact on our method's robustness. Furthermore, tests with 80° FOV cameras with standard framerates allowed us to land only on vehicles slower than 5 km/h. This led us to select a global shutter grayscale camera (mvBlueFOX-MLC200w), which can provide 93 images per second with resolution 752×480. To achieve a sufficient, 185° field of view, we equipped the camera with a small SuperFisheye lens (Sunex DSL215).

All control algorithms were performed on a machine which we have previously tested in a variety of weather conditions – Intel NUC embedded PC with Intel Core i7 5557U processor. This processor has two cores, four threads, and the computer runs three mission-critical control algorithms for trajectory planning, trajectory execution and position estimation using Real-time Kinematic Positioning of Global Positioning System (RTK-GPS), altitude sensors. It was decided to use only one thread for the image processing module to keep enough computation power for those control tasks. Our detection algorithm can recognize the landing pattern in one image using a single thread in less than 15 ms. Considering this detection frame rate and the aforementioned relative velocities, the relative position change of the landing target to the UAV is below 20 cm.

2.1 Algorithm overview

Landing pattern detection is based on standard computer vision approaches using the OpenCV (Bradski, 2000) library. The landing pattern (see Figure 1) consists of a cross within a circle. Therefore, the landing pattern localization algorithm combines outer circle detection with cross detection inside the circle. The detection has to be robust to various weather conditions, changes in light intensity, and shadows cast over the pattern. To reduce the effect of the sun's glare, we implemented a customized camera exposure control. The exposure of the camera was controlled so that the brightest point in the image is in the range 200 to 240 (the maximum brightness of a pixel is 255). This exposure control was especially efficient in suppressing brightness variations, and the images obtained from the camera were suitable for segmentation.

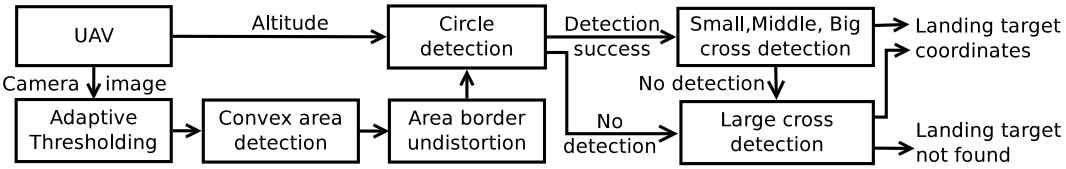

Figure 2: Vision pipeline for landing pattern detection.

The visual pipeline in Figure 2 starts with the adaptive thresholding that produces a binary image, followed by the detection of continuous areas with convex border. The steps mentioned above are performed on the original (radially distorted) image. Then, the border points of continuous areas that are larger than 10 pixels are undistorted using our improved undistort function, see Section 2.2. If the undistorted border points are ellipse-shaped, their inner area is tested for the presence of a cross. If no ellipse is found in the image and the UAV altitude is less than 1.5 m, the detection of the large cross, consisting of two pairs of parallel lines, is initiated. If either a large cross or ellipse with a cross inside is found, the target's coordinates are calculated.

The first step of pattern detection is an application of adaptive thresholding. The OpenCV library optimizes this step by using the integral image. Therefore the adaptive thresholding step is computationally efficient. In our case, the adaptive thresholding (AT) of an image with resolution 752×480 took less than 1 ms. The adaptive threshold is robust to changes in light intensity but can be used to segment correctly only the border of the pattern's circle. The box size defines the size of details that can be detected in the resulting image. The result of the adaptive thresholding with box sizes: 5 pixels, 11 pixels, and 25 pixels can be seen in Figure 3. Therefore the size of the block for adaptive threshold depends on expected pattern size, which again depends on UAV altitude over the landing pattern. The box size has to be an odd number in pixels and was continuously updated according to the current altitude of the UAV as follows:

- At altitudes exceeding 5 m we set the box size to 5 pixels because the pattern appears small, and the size of the circle in the image ranges from 10 to 20 pixels.
- Between 1.5 m and 6 m, the pattern is larger (20 to 200 pixels), and the box size is set to 11 pixels.
- Below 1.5 m, only part of the target's circle is visible (due to the UAV declination when pursuing a fast-moving object), and the box size is set to 25 pixels, which is optimal to detect the target's central cross.

Continuous areas with convex borders are searched for in binary images (see Figure 3) produced by the adaptive thresholding step. The border points of these areas are then undistorted and processed by ellipse and line fitting methods of the OpenCV.

2.2 Optimised fish-eye undistort

The identification of the lens was performed using OpenCV 3.2 and its fish-eye model (Kannala and Brandt, 2006). The original and undistorted images from the UAV's camera are in Figure 4. The size of the undistorted image is 1680×1290, which is too large to achieve real-time detection of the target. Since we do not want to lose details in the image center, we do not solve this by resampling the image. Instead of that, we perform undistortion only for points at the areas border.

For our application, the fish-eye undistortion provided by the OpenCV library is too computationally intensive, and thus, we implemented our own image undistortion method. The original fish-eye undistort function computes transformation from image coordinates (x, y) to undistorted coordinates as shown in Algorithm 1.

The approach described in Algorithm 1 is slow, mainly because of the iterative computation of θ_{10}, which

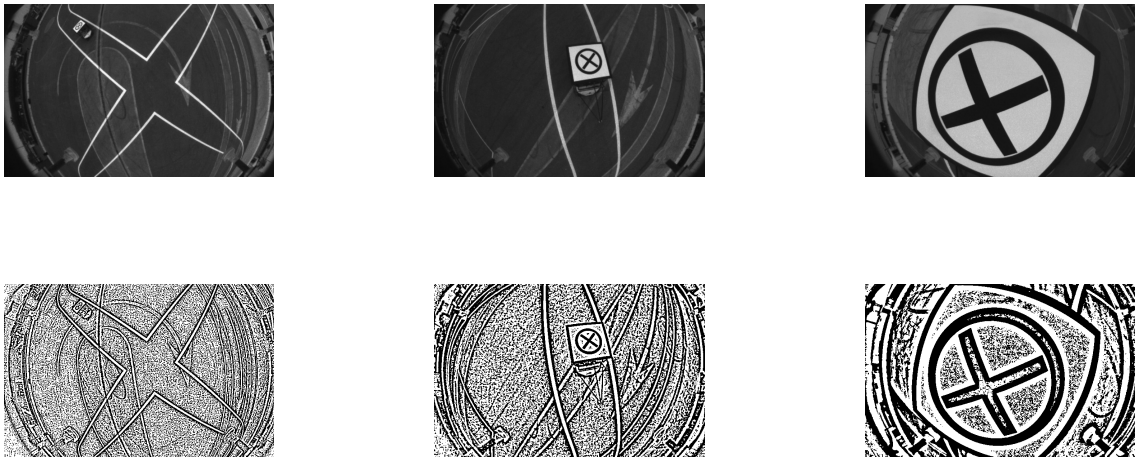

Figure 3: Adaptive thresholding of the target pattern. The top row shows original images from the camera, and the bottom shows results of the adaptive thresholding with box size 5, 11 and 25 pixels (left to right).

Figure 4: Image distortion specific to the fish-eye lens. Original image from the camera on the left and undistorted image on the right.

Algorithm 1: Standard fish-eye undistort

Input: $(x, y, c_{\{x,y\}}, f_{\{x,y\}}, k_{\{0\ldots3\}})$: x, y – coordinates to be undistorted; $c_{\{x,y\}}$ – optical center; $f_{\{x,y\}}$ – focal lengths; $k_{\{0\ldots3\}}$ distortion coefficients from camera calibration

Output: (x', y'): (x', y') – undistorted coordinates

$w_x \leftarrow (x - c_x)/f_x$ `// transform` x `to canonical coordinates`

$w_y \leftarrow (y - c_y)/f_y$ `// transform` y `to canonical coordinates`

$\theta_0 \leftarrow \sqrt{w_x^2 + w_y^2}$ `// compute distance from center`

`// #1 iteratively compute undistortion coefficient` θ_i `(10 times)`

for $i \in 0\ldots9$ **do**

  $\theta_{i+1} \leftarrow \theta_0/(1 + k_0 \cdot \theta_i^2 + k_1 \cdot \theta_i^4 + k_2 \cdot \theta_i^6 + k_3 \cdot \theta_i^8)$

$s \leftarrow \frac{\tan(\theta_{10})}{\theta_0}$ `// determine scale` s

$x' \leftarrow s\, w_x$ `// use scale` s `to calculate undistorted` x `coordinate`

$y' \leftarrow s\, w_y$ `// use scale` s `to calculate undistorted` y `coordinate`

requires 80 float multiplications, 11 float divisions, 1 square root and 1 tangent calculation. Therefore the original undistort function requires 93 float operations per coordinate.

Because the distortion coefficients $k_{\{0...3\}}$ are known in advance, we can precalculate the results of the scale s (see Step #1 of Algorithm 1) for all values of θ_0 with sufficient precision. For our image resolution 752×480 we perform this precalculation for 2000 values of θ_0, obtaining a function $\sigma(\theta_0)$, which sufficiently estimates the fish-eye undistortion for angles between 0 and 75°. Additionally, we neglect the square root function, because we can prepare scale values for the square distance from image origin. This allows optimized computation of undistorted coordinates of image coordinates (x, y) as shown in Algorithm 2.

Algorithm 2: Optimized fish-eye undistort

Input: $(x, y, c_{\{x,y\}}, f_{\{x,y\}}, k_{\{0...3\}})$: x, y – coordinates to be undistorted; $c_{\{x,y\}}$ – optical center; $f_{\{x,y\}}$ – focal lengths; $k_{\{0...3\}}$ distortion coefficients from camera calibration;**s**

Output: (x', y'): (x', y') – undistorted coordinates

$w_x \leftarrow (x - c_x)/f_x$ // transform x to canonical coordinates

$w_y \leftarrow (y - c_y)/f_y$ // transform y to canonical coordinates

$\theta_0 \leftarrow w_x^2 + w_y^2$ // compute square distance from center

$s \leftarrow \sigma(\theta_0)$ // use precomputed values of $\sigma(.)$

$x' \leftarrow s\, w_x$ // use scale s to calculate undistorted x coordinate

$y' \leftarrow s\, w_y$ // use scale s to calculate undistorted y coordinate

The optimised Algorithm 2 uses only 5 float multiplications/divisions and is ∼20 times faster than the conventional method shown in Algorithm 1. To assess its accuracy, we compared its undistortion results with the original method provided by the OpenCV library, see Table 1. For the central areas of the image, corresponding to a cone with apex angle 154°, the maximal error of our algorithm with the comparison to the original one is below 1 pixel. Then, it grows rapidly and towards the image borders, where the angle between the camera axis and pixel projection line exceeds 85°, the maximal error of our algorithm compared to the original undistortion algorithm is 9.4 pixels.

Table 1: Maximal error with respect to angle of pixel projection line

Angle of projection line [deg]	10	20	30	40	50	60	70	80	85
Maximal error [pixel]	0.001	0.006	0.018	0.052	0.123	0.281	0.628	1.270	9.404
Maximal error [%]	0.0003	0.002	0.01	0.03	0.06	0.09	0.13	0.18	0.78

2.3 Robust circle detection

The localization of the landing pattern is based on ellipse detection in images produced in the adaptive thresholding step. The ellipse candidates are selected from continuous areas with convex borders, found by a method similar to (Krajník et al., 2014). Selecting only convex objects eliminates a significant number of false detections and speeds up the target localization.

The points on borders of all convex objects are transformed to undistorted coordinates. Then we calculate the center, size, and rotation of the ellipse by using the OpenCV ellipse fitting method. Next, we rotate and rescale the resulting ellipse and detected border points in a way which projects the ellipse on a circle:

$$\begin{pmatrix} tr_x \\ tr_y \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & \frac{ell_{width}}{ell_{height}} \end{pmatrix} \cdot \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \cdot \begin{pmatrix} br_x - cen_x \\ br_y - cen_y \end{pmatrix}, \tag{1}$$

where (br_x, br_y) are undistorted coordinates of border points, (tr_x, tr_y) are coordinates of the transformed points, (cen_x, cen_y) are undistorted coordinates of the fitted ellipse center, α is detected ellipse rotation and $(ell_{width}, ell_{height})$ are the ellipse axes lengths.

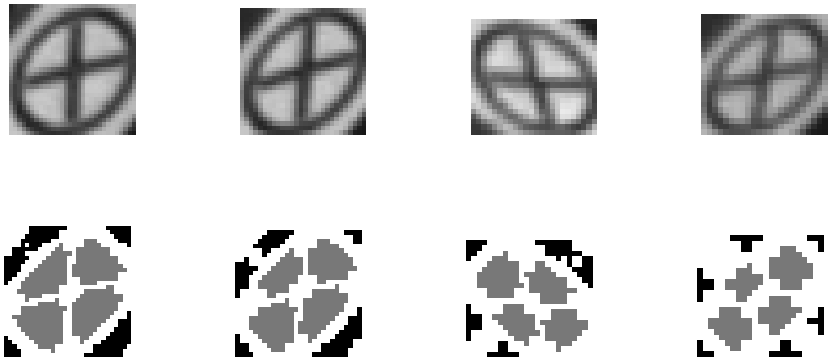

Figure 5: Morphological *closing* for cross detection: original images shown in the top row, processed images in the bottom row.

After the transformation, we calculate the distance of each detected border point (br_x, br_y) to the ellipse. This distance is determined as the distance of transformed border point (tr_x, tr_y) to the circle with center $(0, 0)$ and radius ell_{width}.

$$distance = \sqrt{tr_x^2 + tr_y^2} - ell_{width} \tag{2}$$

The detection is considered as valid if at least 95% of border points fall within 3.5 pixel distance to the ellipse.

2.4 Robust target detection

Typically, the UAV operates in environments where other circular objects are present. Therefore, to eliminate false positive detections, we have to detect the pattern's cross as well. The detection of the cross relies on three different methods, which are selected depending on the detected circle size. The methods differ because the target pattern appearance depends on UAV altitude. While the most robust cross detection method is based on Guo Hall thinning (Guo and Hall, 1989), it cannot be used for targets smaller than 30 pixels, as the cross lines become too narrow. On the other hand, the Guo Hall thinning for targets larger than 150 pixels becomes too slow.

If one of the axes of the ellipse (circle) is smaller than 30 pixels, then one line of the cross cannot be detected reliably, though the cross divides the circle into four areas of similar size. In such cases, the cross presence is verified by searching for these four areas. To do so, a mathematical morphology operation *closing* is applied to the image inside of the detected circle, and the number of closed areas inside the circle is computed. The cross is considered present if there are exactly four closed, similarly-sized areas inside the circle. The cross detection by *closing* is illustrated in Figure 5, which shows original images as well as the detected areas.

If the detected ellipse size is between 30 and 150 pixels, the cross is searched for by Guo Hall thinning (Guo and Hall, 1989), which is very robust, but computationally intensive, and thus applicable only to small-scale images. Once the Guo Hall thinning finds the cross lines inside the detected ellipse, we verify if these intersect close to the ellipse center. Figure 6 shows the original images from the camera in comparison to images after applying adaptive thresholding and Guo Hall thinning.

For circles larger than 150 pixels the Guo Hall thinning becomes too slow. For circles of this size, the cross is detected by searching for two pairs of parallel lines (see Figure. 7), which form a border of the cross. At least two pairs of parallel lines with correct size and distance need to be recognized to detect the cross and its center positively. The algorithm is based on border following, split and merge algorithm, and arc detection. For more details, see the source code of our detector at `http://github.com/gestom/MBZIRC_2017_vision`.

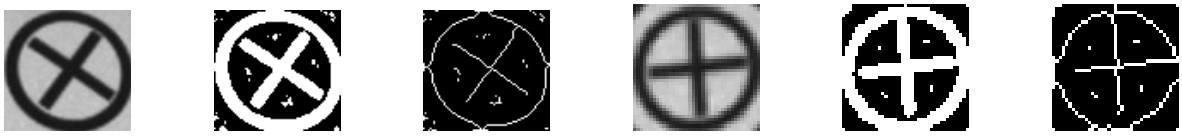

Figure 6: Two examples of landing pattern cross detection by Guo Hall thinning: original images, adaptive thresholding, and Guo Hall thinning.

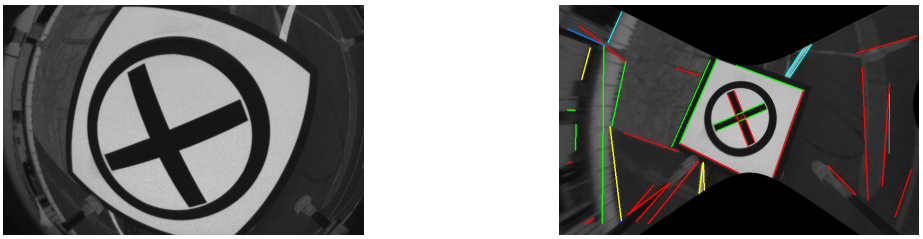

Figure 7: Cross detection by line search. The left image is the original image from the camera; the right image is undistorted showing detected lines by colours (tiny red cross shows position of the center).

The same algorithm for detection of lines and arcs is also used when the circle is only partially in the camera field of view. This situation is selected if no circle with the cross was detected by previous methods and the altitude is less than 2 m when the UAV is maneuvering aggressively to follow the vehicle. Figure 8 depicts a pattern that is only partially in the image (the right image contains only two visible lines of the cross). In the case of circle incompleteness, the landing pattern is considered to be visible if at least four border lines of the landing cross are detected.

2.5 Global pattern position

The final step is the calculation of the landing pattern's position in the global coordinate system. Suppose that we calculated the undistorted coordinates of the pattern's center $\mathbf{x}' = (x', y')$, we know that the pattern's relative position to the camera is $d\,\mathbf{x} = d\,(x', y', 1)$, where d is an unknown positive number corresponding to the pattern-camera distance. Knowing the UAV's camera position $\mathbf{u} = (u_x, u_y, u_z)$ and its rotation $\mathbf{R_u}$ in the

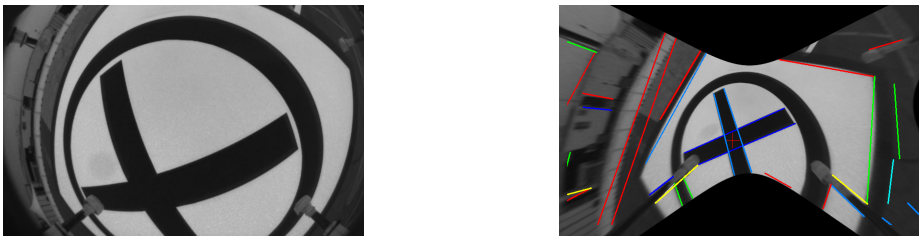

Figure 8: Line detection during the terminal phases of landing, when only part of the landing pattern is visible. The left image is the original image from the camera; the right image is undistorted showing detected lines by colors (tiny red cross shows the position of the center).

global coordinate frame, we can exploit the knowledge of camera altitude u_z and target height above ground t_z to eliminate the unknown distance d and calculate the landing pattern's global coordinates $\mathbf{g} = (g_x, g_y, g_z)$ as

$$\mathbf{g} = \mathbf{u} + (u_z - t_z)\,\mathbf{R_u}\,\mathbf{x}. \tag{3}$$

Equation 3 requires a precise estimate of the UAV's altitude. However, experiments showed that the altitude error could be up to 1 m when relying on the laser- or GPS-based altitude estimation. Therefore, for the pattern detection, we compute the distance to the pattern from the apparent size of the ellipse by a method similar to the one described in Section 3.5.

2.6 Visual system calibration

A successful landing depends on precise localization of the landing pattern in world coordinates. The precision of the localization depends not only on camera intrinsic calibration, but also on the accuracy of the camera pose estimate at the moment the processed image was acquired. This requires not only calibration of the camera coordinate system with the UAV coordinate system, but also synchronization of UAV's position to the time of image acquisition.

The following steps describe the parameters taken into account when calculating the global position of the vehicle:

- position translation of camera origin relative to the UAV body,
- angular shift of camera mount to UAV body,
- time shift of camera data to the time of known position and orientation of the UAV.

The position shift of the camera coordinate system to the UAV coordinate system was measured on a bench during the UAV construction. However, later tests showed that the best way to estimate the angular shift is to use real-world data. The time shift of camera data has to be determined by a special procedure because it depends on many hidden parameters of Robot Operating System (ROS), the software and the hardware used. We found it difficult to set all parameters at once, due to the entanglement of the time and angular shift. Consequently, we designed two separate flight scenarios to automate the estimation.

Both scenarios rely on RTK-GPS up to 3 cm localization precision. The UAV takes off from the center of the landing pattern, and therefore the target's position in world coordinate system can be automatically detected. The first scenario was designed to detect the angular shift of camera, so the UAV was flown as smoothly as possible along a predefined trajectory around the landing pattern. The second scenario employs aggressive maneuvers with a high angular rate of the UAV to detect the time shift of the camera image to UAV's position. From both scenarios, we receive a data set with camera images and the UAV's positions with a precise time stamp. Using these data, we can calculate angular shift of camera as minimizing position error of computed target position and real target position.

Finding the camera parameters splits into two parts. First, the yaw angle α and roll, pitch parameter d_x, d_y is estimated using the dataset from the first scenario. Since a calm, slow-moving flight was selected in the first scenario, the effect of the time delay on the calculated position is negligible. The effect of yaw angle α change on the detected target position is reflected by the following transformation:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}. \tag{4}$$

The point (x, y) is target position in image using undistorted coordinates with $(0, 0)$ in center of the image and point (x', y') is transformed position using yaw angle α. To estimate the angular correction using roll

angle $\beta = \tan \frac{d_x}{f_x}$ we apply transformation

$$x'' = \frac{(x' - d_x) \cdot f_x^2}{f_x^2 + d_x \cdot x'}. \tag{5}$$

To estimate the angular correction using pitch angle $\gamma = \tan \frac{d_y}{f_y}$ we apply transformation

$$y'' = \frac{(y' - d_y) \cdot f_y^2}{f_y^2 + d_y \cdot y'}. \tag{6}$$

Using these transformations we receive point (x'', y'') of corrected target position in undistorted coordinates system, that is then used to compute new global position of target in world coordinates system.

The search for the angular shift can be defined as an optimization task with variables α, d_x, d_y. The task was solved by finding parameters α, d_x, d_y such that they minimized the sum of squared differences between the computed target position and real target position for all data from the first scenario. Minimization is achieved by calculating the difference of positions for all combinations of values α from interval $< -5°, 5° >$ with step $0.5°$, values d_x, d_y from $< -50, 50 >$ with step 2 using focal length $f_x = f_y = 261$ pixels. The task was solved offline using recorded flight data.

In the second scenario, we aimed to determine the time delay of camera data relative to the UAV position. Similar to the previous step, we want to find the time delay Δt that minimizes the sum of squared differences between the computed target position using UAV's position from time $t + \Delta t$ and real target position for all data from the second scenario. As the second scenario is using more aggressive maneuvers, the time shift can be detected more precisely. Minimization is done by calculating the difference of positions for all combinations of values Δt from interval $< -0.1, 0.3 > s$ with 0.02 s steps. We are testing negative values for Δt because the UAV's position data has also a delay, which can exceed the delay of the image transfer. Our experiments show that for the fast BlueFox camera, the image delay caused by reading data from the sensor and transferring data to the computer can be smaller than the post-filtered delay of UAV's position.

3 Colored target localisation and motion estimation

The colored object detection and localization were intended for the 'Treasure hunt' scenario, where several UAVs had to search for small objects in the contest arena, pick them up, and deliver to a specific location. Some of these objects were supposed to be stationary, some were attached to moving robots, and some were too heavy to be carried by a single UAV – these were intended to be carried cooperatively by multiple UAVs. Each object had an assigned score, which the given team obtained if its UAVs brought the object to the designated drop–off zone.

Unlike the landing pattern, which consists of high-contrast black and white elements, the objects in the Treasure hunt scenario were supposed to be covered by a uniform color with an optional indication of the score. The organizers indicated the primary colors (red, green, blue, yellow, and black) and maximum sizes of these objects ~ 0.2 m, but the exact shape, color, and score indication were not announced initially. Thus, we designed the colored object detection pipeline in a way which would allow its easy modification after the disclosure of exact object specifications. Apart from the adaptability requirement, the image processing pipeline had to be computationally efficient, because picking up a small object in outdoor conditions requires fast vision-in-the-loop control. Furthermore, the requirement of outdoor operation meant that the algorithm had to be able to deal with varying illumination and shadows cast by the UAV over the objects during their pickup. Finally, since the camera used had a rolling shutter [3] the object detection method had to deal with the so-called 'jelly' or 'wobble' effect (see Figure 9) caused by vibrations induced by the UAV motors (Afolabi et al., 2015). To satisfy these requirements, we decided to base the colored object detection

[3]Earlier tests indicated that (in contrast to the landing pattern recognition), our color target localization achieved better performance with a high-resolution rolling-shutter camera than with a lower-resolution, global shutter camera of similar price.

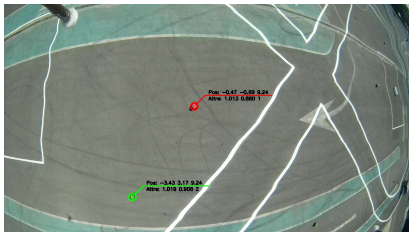

Figure 9: Object detection in onboard camera images affected by the 'jelly' or 'wobble' effect, which deforms lines (left image), as well as circular and square objects (right image). The detection results shown overlayed in the images indicate the 3d relative position (top line) and attributes like roundness, eccentricity, and type (values 1,2, and 3 for red, green, and blue static objects respectively; and 5 for the yellow, moving object.)

method on a computationally efficient segmentation algorithm originally used for black-and-white pattern detection (Krajník et al., 2014). The method combines flood fill techniques and on-the-fly calculation of the segments' statistical properties. The statistical properties are used in consecutive tests with increasing complexity and allow to efficiently determine if a candidate area represents a pattern with th desired shape. The advantage of these statistics is not only computational efficiency but also robustness to the 'jelly' effect. The object detection method can process the entire image, finding all relevant objects in the camera field of view, or it can be set to track only a specific number of objects, which boosts its computational speed by a factor of 2. The 'complete' mode, which processes the entire image, was used when the UAVs created a coarse overview map of the objects in the arena; while the 'tracking' mode was employed when descending to and picking up a specific object. Both of these settings proved especially useful during the contest because the map created during an initial high-altitude flyover allowed our UAV team to create an efficient plan for the objects' collection. Thus, the drones did not have to search for the objects randomly, but they could fly directly to the objects' approximate locations provided by the planning module.

The method presented here shares similarities with the solution used by another very successful team from the University of Bonn (Nieuwenhuisen et al., 2017). However (Nieuwenhuisen et al., 2017) based their segmentation on Maximally Stable Extremal Regions (MSER) (Matas et al., 2004) rather than on flood-fill technique from (Krajník et al., 2014), and they report that their method is slower than the camera frame rate.

3.1 Method overview

The method first retrieves an image from the UAV bottom camera and starts to label the image pixels into the object or background classes. Once it encounters an object-colored pixel, it initiates a flood-fill scheme to search for a continuous object-colored segment around it. Upon finding the segment, it establishes its bounding box, number of pixels, centroid, convexity, and compactness and uses these statistics to determine if the segment can correspond to an object of the desired shape. After that, using the known object size and camera parameters, the method calculates the relative 3d position of the object to the UAV. The relative position is transformed into global 3d coordinates based on the data from the UAV navigation system. Optionally, the method employs a naïve Bayesian classification scheme to determine the score obtainable by picking up the object. Finally, global 3d positions of the detected objects are forwarded to a mapping module, which integrates multiple detections of the objects into a single 3d representation, which is then used by the localization and planning systems. The performance of the method both during tests and the contest indicated not only good computational efficiency but also robustness to changing illumination.

3.2 Pixel labeling

The core of the method is efficient pixel-wise labeling based on a 3d RGB look-up grid, where each cell contains a label indicating if the RGB value represented by the cell coordinates could represent an object or not. Classifying a pixel is performed by retrieving a label from the grid cell with coordinates corresponding to the pixel's RGB values. The principal advantage of this method is that it can be used to implement any classification algorithm in constant time – one needs to precalculate the grid only during system initialization. In our case, the grid was initialized from a parametric model, where each object color was represented by a mixture of Gaussians in the hue-saturation-value color space. To allow further refinement of the color labeling, we developed a GUI, which displays the current results of the colored target localization and the hue-saturation projection of the 3d RGB look-up grid. This GUI can be used to alter the model's components or add and remove regions with a given label in the color grid. Thus, if a given object type could not be accurately detected, the user can delimit the image area containing the object, which triggers the display of the hue-saturation histogram within the area. The user can then indicate which colors (in hue-saturation space) in this area correspond to the object and which do not. Apart from labels indicating background or a given object class, the colormap can also contain 'ambiguous' labels, corresponding to colors which can but might not represent the searched object color. The 'ambiguous' labeled pixels were classified according to their neighboring pixels – this was particularly useful for white numbers with the object score, because the contest area also contained uncollectable white objects.

Not all pixels of the image need to be labeled – if the system works in the 'tracking' mode, the labeling is performed only as long as the method finds a specific number of objects. Thus, if the pixel labeling starts at some of the searched object's pixels, the method does not process any pixels apart from the ones that belong to the object itself, which results in a significant speedup.

3.3 Segmentation

In the first phase, Algorithm 4 uses the RGB grid to consecutively examine if the image pixels could belong to the searched object. Whenever a pixel is classified as being a pixel of a potential object, a flood-fill procedure shown in Algorithm 3 is initiated to search for a continuous segment of the object color. In other words, the method (Algorithm 4) labels the image pixels, and once an object pixel is detected, a queue-based flood-fill Algorithm 3 is initiated to find a segment around it. The queue used in Algorithm 3 contains positions of the segment's pixels and is implemented as a buffer with two pointers q_{start} and q_{end}. This ensures that at the end of the flood-fill calculation, the queue contains all the positions of the segment pixels, which are then used for further processing.

In particular, once the flood fill finishes, the segment is tested for a minimal size (in terms of the number of pixels belonging to the segment) and a roundness measure within acceptable bounds. These two simple constraints can be validated quickly, which results in a fast rejection of false positives. If either test fails, Algorithm 3 reports that the segment is not valid, and detection for other segments continues from the following pixel position (i.e., a pixel at the position $i + 1$) by Algorithm 4.

The roundness test, see Algorithm 3, is based on the pattern's bounding box dimensions and the number of pixels, where number of pixels s of an ellipse with dimensions b_u, b_v should be $s = \pi\, b_u\, b_v/4$. Therefore, the segment dimensions and number of pixels should satisfy the inequality

$$\rho_{tol} > \left| \frac{\pi}{4\,s}\, b_u b_v - 1 \right|, \tag{7}$$

where the value of ρ_{tol} represents a tolerance range, which depends on the camera radial distortion, possible pattern deformation, and spatial orientation.

If the segment passes the minimum size and roundness tests, its final validation is performed by calculating a more sensitive circularity measure described in the following subsection (see also Algorithm 4). Once it

Algorithm 3: Flood-fill segmentation

Input: $(p, rgb\_grid, object\_class)$: p – starting pixel position; $class$ – searched segment label
Output: $(\mathcal{S})$: $\mathcal{S}$ – positions of segment's pixels

$s_{id} \leftarrow s_{id} + 1$ // increment segment ID
$q_{end} \leftarrow q_{start} \leftarrow 0$ // initialise queue
$u_{max} \leftarrow u_{min} \leftarrow p_u$ // initialise bounding box
$v_{max} \leftarrow v_{min} \leftarrow p_v$ // initialise bounding box
pixel_label$[p] \leftarrow s_{id}$ // mark pixel as processed
queue$[q_{end}++] \leftarrow p$ // and push its position to the queue
// #3.1 perform the flood fill search
while $q_{end} > q_{start}$ **do**
  $q \leftarrow$ queue$[q_{start}++]$ // pull pixel from the queue
  // #3.2 and check its neighbours, (w is image width in pixels)
  foreach $offset \in \{+1, -1, +w, -w\}$ **do**
    $r \leftarrow q + offset$
    // #3.3 if a pixel is not labeled yet, then assign it a label
    if pixel_label$[r] = unknown$ **then**
      pixel_label$[r] \leftarrow$ rgb_grid(Image$[r]$)
    // #3.4 if it has object color, then
    if pixel_label$[r] = object_class$ *or ambiguous* **then**
      queue$[q_{end}++] \leftarrow r$ // add it to the queue
      pixel_label$[r] \leftarrow s_{id}$ // label it as belonging to the current segment
      // #3.5 update bounding box params
      $u_{min} \leftarrow \min(u_{min}, r_u),\ u_{max} \leftarrow \max(u_{max}, r_u)$
      $v_{min} \leftarrow \min(u_{min}, r_v),\ v_{max} \leftarrow \max(v_{max}, r_v)$
$\mathcal{S} \leftarrow \{\emptyset\}$
// # 3.6 test for the pattern size
if $q_{end} \geq min\_size$ **then**
  $\rho \leftarrow \pi b_u b_v/(4s) - 1$ // # 3.7 calculate roundness
  // # 3.8 test segment roundness
  if $\rho_{tol} > \|\rho\|$ **then**
    $\mathcal{S} \leftarrow$ queue // push queue to output and mark as valid

passes this test, the pattern is considered to be valid, and its centroid position is used as a starting point i_0 for the next detection run. If the method works in 'tracking' mode, the image processing finishes. If not, Algorithm 4 continues to search for objects in the rest of the image pixels.

3.4 Calculating ellipse semiaxes

To finally verify if the pattern is elliptic, we calculate its semiaxes' lengths and compare those to the number of segment pixels. The pixel positions to calculate the ellipse center u, v and the semiaxes $\mathbf{e_0}, \mathbf{e_1}$ are stored in the flood-fill queue formed by Algorithm 3. First, the ellipse center (u, v) is calculated as an arithmetic mean of the pixel positions. Then, the covariance matrix $\mathbf{C}$ is calculated as

$$\mathbf{C} = \begin{pmatrix} c_{uu} & c_{uv} \\ c_{uv} & c_{vv} \end{pmatrix} = \frac{1}{s} \sum_{i=0}^{s-1} \begin{pmatrix} u_i u_i & u_i v_i \\ u_i v_i & v_i v_i \end{pmatrix} - \begin{pmatrix} uu & uv \\ uv & vv \end{pmatrix}, \tag{8}$$

Algorithm 4: Pattern detection

Input: $(i_0, rgb\_grid, mode, \texttt{Image})$: i_0 – search start position ; rgb_grid – RGB lookup grid; `Image` – image to be processed,$mode$ – either search a 'complete' image or 'track' a single segment

Output: $(\mathcal{P})$: $\mathcal{P}$ – set of detected segments, each described by its centre $\mathbf{u}$, color $\mathbf{h}$, and semiaxes $\mathbf{e}_{0,1}$;

```
i ← i_0                                              // #4.1 set current position to search start
// #4.2 search through the image and label pixels
repeat
    if pixel_label[i] = unknown then
        if rgb_grid(Image[i]) = object_class then
            pixel_label[i] ← object_class
    // #4.3 if pixel at position i has object color, initiate flood fill
    if pixel_label[i] = object_class then
        S ← flood-fill_seg(i, rgb_grid, object_class)    // #4.4 perform flood fill, see Algorithm 3
        // #4.5 if the flood fill proposed a segment candidate
        if |S| > 0 then
            h ← Σ_{s∈S} HSV(Image[s]);          // #4.6a calculate mean HSV of the segment pixels
            u ← Σ_{s∈S} (s_x, s_y);             // #4.6b calculate mean of the segment pixels positions
            C ← cov_{s∈S}(s_x, s_y);// #4.6c calculate covariance of the segment pixels positions
            (λ_0, λ_1, v_0, v_1) ← eig(C);                  // #4.7 covariance matrix eigenanalysis
            e_{0,1} ← sqrt(λ_{0,1}) v_{0,1};                    // #4.8 determine ellipse semiaxes
            // #4.9 final circularity test
            if |S| ≈ 4 π |e_0 e_1| then
                P = P ∪ (u, h, e_0, e_1)    // add segment information to the set of segments found
                if mode = track then
                    break                                // if in 'track' mode, then terminate
    i ← (i + 1) mod sizeof(Image)                        // go to next pixel until you reach i_0
until i ≠ i_0;
```

where u_i and v_i are the pattern's pixel coordinates stored in the queue. Since the matrix $\mathbf{C}$ is two-dimensional, its eigenvalues λ_0, λ_1 are found as a solution of a quadratic equation:

$$\begin{aligned} \lambda_d &= \max\left(\sqrt{(c_{uu}+c_{vv})^2 - 4(c_{uu}c_{vv} - c_{uv}^2)}, 0\right), \\ \lambda_0 &= (c_{uu} + c_{vv} + \lambda_d)/2, \\ \lambda_1 &= (c_{uu} + c_{vv} - \lambda_d)/2. \end{aligned} \tag{9}$$

If λ_d equals 0, then the pattern is a perfect circle and we assume the corresponding eigenvectors $\mathbf{v_0}$,$\mathbf{v_1}$ to be $(1,0)^T$ and $(0,1)^T$ respectively. Otherwise, the eigenvectors are calculated by

$$\mathbf{v_0} = \frac{1}{\sqrt{c_{uv}^2 + (c_{uu}-\lambda_0)^2}} \begin{pmatrix} c_{uv} \\ \lambda_0 - c_{uu} \end{pmatrix}, \quad \mathbf{v_1} = \frac{1}{\sqrt{c_{uv}^2 + (c_{vv}-\lambda_1)^2}} \begin{pmatrix} \lambda_1 - c_{uv} \\ c_{uv} \end{pmatrix}. \tag{10}$$

Having the eigenvectors and eigenvalues, we can calculate the ellipse semiaxes $\mathbf{e_0}, \mathbf{e_1}$ by

$$\mathbf{e_i} = \sqrt{\lambda_i}\mathbf{v_i}. \tag{11}$$

The final test verifying the pattern roundness is performed by checking if the inequality

$$\rho_{prec} > \left| 4\,\pi\,\frac{|\mathbf{e_0}||\mathbf{e_1}|}{s} - 1 \right| \tag{12}$$

holds, where s is the pattern size in the number of pixels. Unlike in the previous roundness test (7), the tolerance value of ρ_{prec} is much lower because (11) establishes the ellipse dimensions with subpixel precision.

3.5 Position estimation in relative 3d space

Once the ellipse center, dimensions, and orientation are calculated, we determine its 3d position in the camera coordinate frame. First, we calculate the ellipse's characteristic equation in the canonical camera coordinates, which correspond to an ideal pinhole camera model with unit focal lengths and zero radial distortion. Then, we retrieve the position of the ellipse in 3d space utilizing its characteristic equation eigenanalysis.

3.5.1 Transformation to canonical camera coordinates

To compensate for the radial distortion of the image at the position of the detected ellipse, we calculate the camera characteristic equation from the coordinates of the pattern's canonical vertices. First, we calculate the image coordinates of the ellipse vertices $\mathbf{a_{0,1}}$ and co-vertices $\mathbf{b_{0,1}}$. We transform those to the canonical camera (i.e. camera with no radial distortion and unit focal lengths) coordinates $\mathbf{a'_{0,1}}, \mathbf{b'_{0,1}}$. These coordinates are then used to determine the pattern's canonical center and canonical semiaxes.

The ellipse center and semiaxes $\mathbf{u}$,$\mathbf{e_0}, \mathbf{e_1}$ are known and thus, the canonical (co-)vertices $\mathbf{a'_{0,1}}$ and $\mathbf{b'_{0,1}}$ are obtained by adding the semiaxes to the ellipse center and transforming the result:

$$\begin{array}{rcl} \mathbf{a'_{0,1}} & = & g'\left((u \pm e_{0x} - c_x)/f_x, (v \pm e_{0y} - c_y)/f_y\right) \\ \mathbf{b'_{0,1}} & = & g'\left((u \pm e_{1x} - c_x)/f_x, (v \pm e_{1y} - c_y)/f_y\right) \end{array},$$

where g' stands for the undistortion function and $f_{x,y}, c_{x,y}$ are the camera focal lengths and optical center respectively. Using the canonical coordinates of the ellipse vertices, the ellipse center $\mathbf{u'_c}$ and axes $\mathbf{e'_0}$, $\mathbf{e'_1}$ are then determined by

$$\begin{array}{rcl} \mathbf{e'_0} & = & (\mathbf{a'_0} - \mathbf{a'_1})/2, \\ \mathbf{e'_1} & = & (\mathbf{b'_0} - \mathbf{b'_1})/2, \\ \mathbf{u'_c} & = & (\mathbf{a'_0} + \mathbf{a'_1} + \mathbf{b'_0} + \mathbf{b'_1})/4. \end{array}$$

3.5.2 Characteristic equation

Each point u', v' lying on an ellipse satisfies its characteristic equation:

$$\begin{pmatrix} u' \\ v' \\ 1 \end{pmatrix}^T \begin{pmatrix} q_a & q_b & q_d \\ q_b & q_c & q_e \\ q_d & q_e & q_f \end{pmatrix} \begin{pmatrix} u' \\ v' \\ 1 \end{pmatrix} = \mathbf{X^T Q X} = 0, \tag{13}$$

where $\mathbf{Q}$ is a conic, and the parameters of the matrix $\mathbf{Q}$ are calculated from the ellipse center and axes by:

$$\begin{array}{rcl} q_a & = & +e'_{0u}e'_{0u}/|\mathbf{e'_0}|^2 + e'_{0v}e'_{0v}/|\mathbf{e'_1}|^2 \\ q_b & = & +e'_{0u}e'_{0v}/|\mathbf{e'_0}|^2 - e'_{0u}e'_{0v}/|\mathbf{e'_1}|^2 \\ q_c & = & +e'_{0u}e'_{0u}/|\mathbf{e'_1}|^2 + e'_{0v}e'_{0v}/|\mathbf{e'_0}|^2 \\ q_d & = & -u'_c q_a - v'_c q_b \\ q_e & = & -u'_c q_b - v'_c q_c \\ q_f & = & +q_a u'^2_c + q_c v'^2_c + 2q_b u'_c v'_c - 1 \end{array}. \tag{14}$$

3.5.3 Retrieving the 3d position

Finally, the position of the pattern is calculated by eigenvalue analysis of the conic $\mathbf{Q}$ (Yang et al., 2012). Let us denote the $\mathbf{Q}$'s eigenvalues and eigenvectors as $\delta_0, \delta_1, \delta_2$ and $\mathbf{q_0}, \mathbf{q_1}, \mathbf{q_2}$, respectively. Since the conic $\mathbf{Q}$ represents an ellipse, its signature is $(2, 1)$ and $\delta_0 \geq \delta_1 > 0 > \delta_2$. According to (Yang et al., 2012), the position of the circle in the camera coordinate frame is:

$$\mathbf{x_c} = \pm \frac{d}{\sqrt{-\delta_0 \delta_2}} \left(\mathbf{q_0} \delta_2 \sqrt{\frac{\delta_0 - \delta_1}{\delta_0 - \delta_2}} + \mathbf{q_2} \delta_0 \sqrt{\frac{\delta_1 - \delta_2}{\delta_0 - \delta_2}} \right), \tag{15}$$

where d is the real-world pattern diameter, and the sign is determined taking into account that the pattern is located in front of the camera, and therefore, the first component of the $\mathbf{x_c}$ vector should be positive. Finally, the position $\mathbf{x_c}$ of the target is transformed to the global coordinate frame using the TF framework of the ROS operating system. This allows to verify if the detected object lies on the ground plane – objects that are not close to the ground are discarded.

3.6 Digit recognition

Objects in the Treasure hunt have a value representing the score awarded after they are successfully grasped and dropped into the designated box. A preliminary version of the MBZIRC scoring mentioned that digits on the objects' surface will indicate the score, but no specific details were provided. Therefore, we developed an easily adaptable system for recognizing the scores of the detected objects. The advantage of our approach is that it can be used for both rectangular and round objects with any graphical representation of the object's score, such as Arabic or Roman numerals, binary codes, etc. Since the digit-based scoring system was not used in the final version of the MBZIRC 2017 rules, we provide only a coarse description.

The number-labeled objects had square, not circular cross-section, so we altered the tolerance of roundness test in Algorithm 3 and omitted the circularity test mentioned in Algorithm 4. Instead, we tested if the center of the detected segments contains brighter pixels corresponding to the white digits – if not, the segment was discarded. Furthermore, we did not determine these objects' 3d positions by the method from Section 3.5, but we estimated their distance based on their size only and then applied a method from Section 2.5.

Having obtained quadrilateral image projections of the square objects from the detection method, we first transform them into squares of uniform size. Then, we convert the image to grayscale and stretch its contrast by histogram equalization. Since we know the approximate ratio of the pixels that constitute the number to the rest of the object pixels, we calculate a threshold, which separates 15% of the brightest pixels from the background ones. By applying this threshold, we obtain a binary image. Finally, we use a pixel-wise naïve Bayesian classifier, trained on labeled binary images, to determine the object number. Since the digit on a square object can appear in four orientations, the Bayesian classifier is applied for all orientation and returns a result with the maximum likelihood. The stages of the number recognition are illustrated in Figure 10.

Figure 10: Digit recognition stages from left to right: image provided by the detection system with indicated corners, transformed image, grayscaled image, result of histogram equalisation, and thresholded image

3.7 Mapping module

The mapping module integrates the individual transformed detections into a global 3d coordinate frame shared by the UAV team. The map building algorithm used was inspired by a system described in (Kusumam et al., 2017), which proved reliable operation in field conditions. To associate the object detections coming from the perception system with the objects in the map, we assume that the maximal error of object position estimation is below a specific value, which we denote as e_{max}. Let us assume that the map is a set $\mathcal{M}$ where each element $\mathbf{m} \in \mathcal{M}$ contains the object's global 3d position $\mathbf{g_m}$, vector of likelihoods of all possible scores $\mathbf{s_m}$, object's rgb color $\mathbf{c_m}$, most likely score s_{max}, and number of detections n_m. Let us assume that the perception system analyzed an image from the UAV camera and that it detected a set of objects $\mathcal{D}$ (these have the same descriptions as the objects in the map). For each detected object $\mathbf{d} \in \mathcal{D}$, the mapping module finds the closest (in terms of distance between $\mathbf{c_d}$ and $\mathbf{c_m}$) object in the map.

If the distance between the detected $\mathbf{g_d}$ and mapped $\mathbf{g_m}$ object position is lower than e_{max}, we associate the detected object $\mathbf{d}$ with the one in the map $\mathbf{m}$ and update $\mathbf{m}$ accordingly (Algorithm 5). If the distance exceeds e_{max}, we simply add the detected object $\mathbf{d}$ to the map $\mathcal{M}$.

Algorithm 5: Update step of color target map

Input: $(\mathcal{D}, e_{max})$: $\mathcal{M}$ – mapped objects; $\mathcal{D}$ – detected objects; e_{max} maximal error of object localisation

an object $\mathbf{o} = (\mathbf{g}, \mathbf{s}, \mathbf{c}, n)$, where $\mathbf{g}$ is its global 3d position, $\mathbf{s}$ is a vector of its score likelihoods, s is the most likely score, $\mathbf{c}$ is its rgb color and n denotes the number of times the object was detected.

Output: $(\mathcal{M})$: $\mathcal{M}$ – updated map;

foreach $\mathbf{d} = (\mathbf{g_d}, \mathbf{s_d}, \mathbf{c_d}, n) \in \mathcal{D}$ **do**

- $(\mathbf{g_m}, \mathbf{s_m}, \mathbf{c_m}, n_m) \leftarrow \text{argmin}\{||\mathbf{g_i}, \mathbf{g_m}|| \; |\mathbf{g_m} \in \mathcal{M}\}$ `// #5.1 the closest object from` $\mathcal{M}$ `to` $\mathbf{d}$;
- `// #5.2 if the object was seen before`
- **if** $||(g_d, g_m)|| < e_{max}$ **then**
  - $\mathbf{g_m} \leftarrow (n\,\mathbf{g_m} + \mathbf{g_d})/(n+1)$; `// #5.3 update its position`
  - $\mathbf{c_m} \leftarrow (n\,\mathbf{c_m} + \mathbf{c_d})/(n+1)$; `// #5.4 update mapped object's color`
  - $\mathbf{s_m} \leftarrow \mathbf{s_m} + \mathbf{s_d}$; `// #5.5 update likelihoods of the scores`
  - $s_{max} \leftarrow \text{argmax}\{\mathbf{s_m}\}$; `// #5.6 update the most likely score`
  - $n \leftarrow n + 1$; `// #5.7 increment the number of observations`
- **else**
  - $\mathcal{M} \leftarrow \mathcal{M} \cup \mathbf{d}$; `// #5.8 add detected object to the map`

As the images from the bottom UAV camera are analyzed, and objects are detected, the Algorithm 5 determines if these belong to existing objects in the map and updates the map accordingly or uses the detections to initialize new objects in the map. Thus, each new detection of an object refines the map.

4 Experiments

4.1 Landing pattern detection

To determine the accuracy of the landing pattern localization system, we compared its position estimates with a ground truth provided by a RTK-GPS. In the experiments performed, the UAV was first centered at the landing pattern, and it used its RTK-GPS to measure its position. After that, it took off from the landing pattern and performed several maneuvers above it at different altitudes between 0 and 4 m.

In the first flight, the UAV maneuvers were performed with an acceleration limit of 0.2 ms^{-2}, while during the second flight the maneuvers were more aggressive. The positions of the landing pattern provided by the pattern localization system are shown in Figure 11. The Figure 11 indicates that when the acceleration of the UAV was limited, the pattern localization accuracy was ± 5 cm. However, aggressive flight with abrupt changes in UAV attitude affects the pattern position estimation negatively. This indicates that the error of the localization is significantly influenced by the accuracy of the synchronization of the camera and UAV position estimation system. Reducing this error would be very difficult without the use of a real-time operating system on board the drone.

The primary comparison of our algorithm was made at the MBZIRC competition. The team from the University of Bonn that (Beul et al., 2017) received the third place at autonomous landing. They used a system with two cameras, one with a wide (195°) and second with a narrow (65°) field of view. Their method first undistorts and rectifies the entire image and searches for circles using a Hough transform. From the method description, we deduce that their method does not process every image from their onboard, 40 FPS camera. The team from the University of Catania (Canteli and other, 2017) uses Tracking Learning

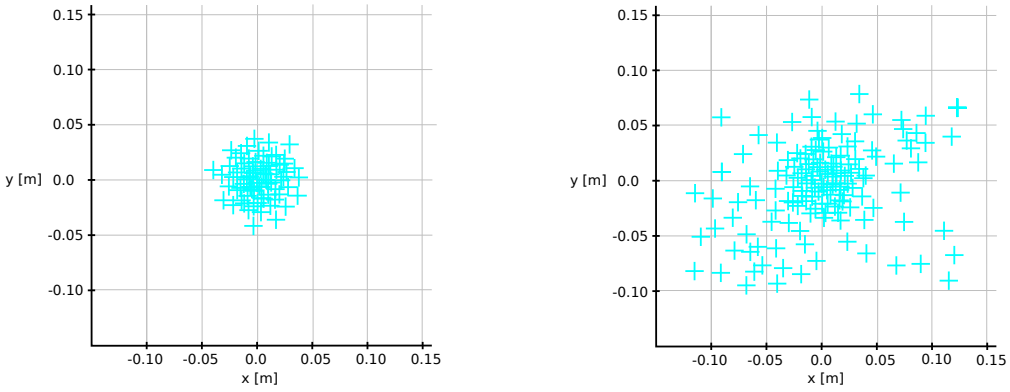

Figure 11: Experiment with limited acceleration (left) and with aggressive maneuvering (right), described in Section 4.1

Detection (Kalal et al., 2012) algorithm processing an image from a fish-eye camera. The vision system of the University of Zurich team (Falanga et al., 2017) combines two standard-FOV cameras: one looking downward and second angled down by 45° used for visual odometry. Their detection algorithm, which combines thresholding with segmentation, can process the captured images in 12 $ms \pm 5.7$ ms. The team from the University of Sevilla (Acevedo et al., 2017) is using an Ocam camera with a 110° field of view. Their method, which takes 20 ms, is based on contour-based localization of the landing's pattern cross. While the teams used different target detection methods, only those with fisheye cameras were able to land on vehicles moving at full speed of 15 km/h.

To evaluate the success rate of the detection algorithm, we select parts of data from MBZIRC competition where the target is visible. The data were stored from ROS to individual files called rosbags during the real competition. The data contains five experiments from Abu Dhabi with different weather, ranging from dark clouds to bright sun where the UAV cast shadows onto the landing pattern.

The dependence of the success rate of target detection on the UAV altitude is provided in Table 2. The first line contains success rates of low altitudes when the detection is problematic because the target does not fit in the camera's field of view and UAV landing gear is occluding the pattern partially. The second line in the table shows the detection results from optimal altitudes, where the detection rate is 100%. The last line shows the success rate for high altitudes when the target's image is small. The detection can be qualitatively evaluated using rosbags and videos available at `http://github.com/gestom/MBZIRC_2017_vision`.

Table 2: Success rate of landing pattern detection

	Training Day 1		*Challenge 1 Day 2*		*Challenge 1 Day 3*		*Grand challenge First run*		*Grand challenge Second run*	
Altitude	Total	Rate	Total	Rate	Total	Rate	Total	Rate	Total	Rate
0 - 2 [m]	253	47.4%	162	67.3%	337	67.4%	75	62.6%	66	69.7%
2 - 6 [m]	1149	100.0%	1724	100.0%	1114	100.0%	253	100.0%	330	100.0%
6 - 9 [m]	291	88.6%	64	95.3%	246	97.9%	60	98.3%	54	100.0%

The module for landing target detection is implemented as a ROS node, which subscribes to the camera topics and publishes the target pose in world coordinates. The entire computation, performed in one thread and measuring a time between the reception of the camera image and publication of the target position, takes approximately 12 ms per image. The final framerate achieved was 50 FPS, with 12 ms for the detection itself and 8 ms for ROS and Operating Systems overheads. Note that the framerate information was established from ROS logs collected during the contest and not from the rosbags. This is because we could not store more than 25% of the captured images due to hard drive data transfer limitation.

4.2 Colored object detection

To assess the robustness and accuracy of the color object detection pipeline, we have performed two separate experiments. The first experimental evaluation of the system was performed during preliminary outdoor tests in Czechia. The primary aim of this evaluation was to test the entire object detection pipeline, which, at the time, was supposed to detect cuboid objects with white digits on their top side. During the experiment, we placed 14 objects in the test area, ground-truthed their positions with a RTK-GPS, and then we performed three subsequent flyovers, which were planned by (Pěnička et al., 2017). The resulting dataset, which covers a continuous, 268 s long flight contained over 2652 images, which were captured by the UAV's bottom camera and associated with the drone's position estimate.

To evaluate the object detection system robustness, we counted the number of correct (1507), false positive (2), and false negative (126) detections – this corresponds to 99.86% precision and 92.28% recall of the detection system. The Figure 12 illustrates that another distinct object on the field caused the false positives, and grass occlusions and sun reflections typically caused false negatives. The results provided by the detection pipeline were further processed by the mapping module, which created a map containing all 14 objects, i.e., the entire mapping system achieved 100% precision and 100% recall, see video at `https://youtu.be/E1nUvw12WSM`. To estimate the accuracy of the localization, we calculated the differences between the objects' positions provided by the mapping module with the RTK-GPS-based ground truth; the mean difference was 0.36 m. We attribute this error to aggressive maneuvers of the UAV, causing incorrect transformation of the UAV-relative object positions to the global coordinate frame due to inaccurate time synchronization of the image capture and accelerometric data. To get further insight in the localization pipeline performance in terms of false positive removal, we also calculated the number of detected segments in the individual processing stages, i.e., the number of segments after the test for their size, roundness, digit presence and position, see Table 3.

Figure 12: False positive (left) and false negative (middle, right) examples of the first experiment.

Table 3: Number of segments after a given processing step

	Experiment and used objects			
	Preliminary testing		*Grand Challenge*	
	Cuboids with digits		*Circular objects*	
Processing step	Number	Reduction	Number	Reduction
Segmentation	1167425	—	1740678	—
Size test	41900	96.4%	26512	98.5%
Roundness test	40294	3.8%	19544	26.3%
Circularity test	—	—	4822	75.3%
Digit presence test	2940	92.7%	—	—
Position test	1507	48.6%	4051	16.0%

In the first experiment, we also evaluated the quality of the digit recognition, trained on a dataset collected a few days before. Despite issues caused by reflections and shadows, the system correctly classified 97 % of the object labels, see also `https://youtu.be/qAVKI_DQKw8`. The failures were typically caused by incorrect estimation of the segment corners, which prevented proper transformation of the segment to a square patch.

The second system evaluation is based on the data from the contest itself, where our multi-UAV system, which was using the object detection and mapping pipeline described, collected more objects from the contest arena than other teams, and won a gold medal of MBZIRC Treasure Hunt Challenge. We selected a 12 minute record from one of the UAV's bottom camera, which contains 19350 images. In this sequence, our system detected an object 4051 times and failed to detect an object 35 times. The false negatives were typically caused by a gross deformation of the object due to the jelly effect, or by wrong UAV altitude estimate caused by temperature-related altimeter malfunction, see Figure 14. Out of the 4051 detections, 4 were false positives: a blue, circular segment on a contest banner (1 detection) and a pole-like contest object (3 detections). In other words, the detection system achieved to 99.1% recall and 99.9% precision. By integrating these detection results into a global map, the precision and recall of the entire system achieved 100%. We provide a video indicating the detection results along with the map of the objects in the contest arena at `https://youtu.be/agR0juadp1g` and a representative snapshot in Figure 13. In this test, we did

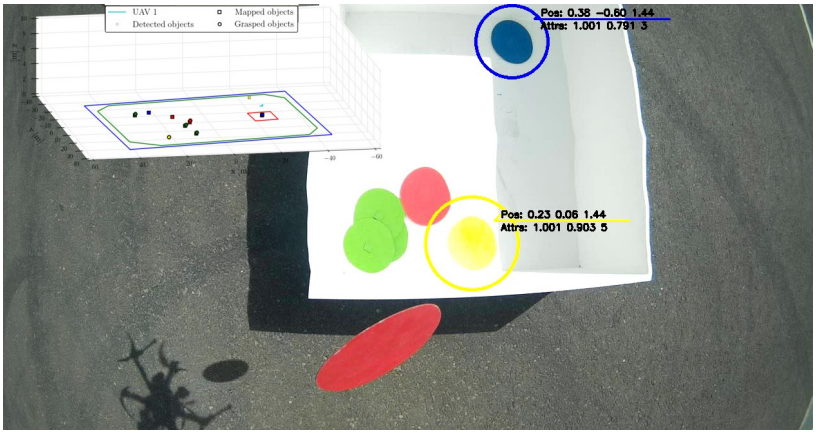

Figure 13: View from the UAV camera with object detection results and a map of the objects on the contest arena. The UAV releases the last object into the dropoff zone.

not have ground truth positions of the objects, but since the UAV used its gripper to pick them up, the object localization error was lower than 7 *cm*. (The object radius was 10 *cm*, and the gripper's center had to be 3 *cm* from the object edge for a successful grasp.) Similarly to the first experiment, we also calculated the number of detected segments in the individual processing stages, i.e., the number of segments after the test for their size, roundness, circularity, and position, see Table 3.

Figure 14: False positives (left,middle) and false negative (right) examples of the second experiment.

5 Conclusions

We presented methods for visual localization and tracking of objects, which were part of the multi-UAV system deployed in the Challenges 1 and 3 of the MBZIRC competition. These methods, structured in two separate visual processing pipelines corresponding to the two challenges, proved their ability to detect, localize, and track dynamic objects in adverse lighting conditions in outdoor, diverse environments.

The first processing pipeline had to reliably localize, track, and predict the movement of a rapidly moving vehicle, which carried a predefined fiducial marker. The accuracy and reliability of the localization and motion estimation ensured the ability of the UAV to land on the moving vehicle in a repeatable manner.

The second processing pipeline had to reliably recognize, locate, and track colored objects scattered over a large area. The information provided by the second pipeline was used to create a map of these objects so that the UAV team could plan how to collect these objects within a given time. Furthermore, the colored object detection had to provide accurate positions of the objects relative to the UAV, so that the visual information could be used to pick these objects up by the UAV gripper.

The efficiency of these methods is evaluated quantitatively in several outdoor experiments and qualitatively during MBZIRC contest, where the UAV system using these vision methods won gold, silver, and bronze medals. While the pipelines described here do not represent the bleeding edge of computer vision research, they were designed to be modular, versatile, and easy-to-modify. The versatility of the methods' modules allowed for efficient testing of alternative algorithms in field conditions, where extensive code modifications are not possible. Thus, we could rapidly identify, remove, and substitute components, which, while efficient, had reliability or robustness issues. These properties paid off during the contest, where we needed to alter these pipelines' components according to the unexpected conditions (e.g., the arena was not precisely flat, some of the colors were specular, banners around arenas generated false positives, etc.). The advantages of versatility and robustness of simpler algorithms over state-of-the-art methods along with the need for extensive testing were also stressed by another successful team of MBZIRC in (Nieuwenhuisen et al., 2017).

Another key factor was that the integrated working system was created several months before the contest and the system components were integrated continuously, which allowed 'anytime' field and simulation testing. Thus, the team members responsible for control and planning modules became aware of the vision systems' properties, and the vision system designers understood better the other modules' requirements. Moreover, the team members could not only alter their methods accordingly but also suggest modifications of other modules, which would improve the robustness and efficiency of the entire integrated system.

The source codes of both processing pipelines, along with the datasets used for experimental evaluation, are available at `https://github.com/gestom/MBZIRC_2017_vision`.

Acknowledgments

This project could not have achieved its success without the full cooperation of all members of the team, comprising of people from the Czech Technical University in Prague, University of Pennsylvania and University of Lincoln, UK. The work was supported by the Khalifa University project MBZIRC 2017, OP VVV MEYS Research Center for Informatics project CZ.02.1.01/0.0/0.0/16_019/0000765 and CTU grant no. SGS17/187/OHK3/3T/13. The work of Tomáš Krajník was supported by the Czech Science Foundation project 17-27006Y, the work of Martin Saska was supported by the Czech Science Foundation project 17-16900Y.

Bibliography

Acevedo, J. J., García, M., Viguria, A., Ramón, P., Arrue, B. C., and Ollero, A. (2017). Autonomous landing of a multicopter on a moving platform based on vision techniques. In *Iberian Robotics conference*, pages 272–282. Springer.

Afolabi, D., Man, K. L., Liang, H.-N., Guan, S.-U., and Krilavičius, T. (2015). 1543. monocular line tracking for the reduction of vibration induced during image acquisition. *Journal of Vibroengineering*, 17(2).

Beul, M., Houben, S., Nieuwenhuisen, M., and Behnke, S. (2017). Fast autonomous landing on a moving target at mbzirc. In *Mobile Robots (ECMR), 2017 European Conference on*, pages 1–6. IEEE.

Borowczyk, A., Nguyen, D., Nguyen, A. P., Nguyen, D. Q., Saussié, D., and Le Ny, J. (2017). Autonomous landing of a quadcopter on a high-speed ground vehicle. *Journal of Guidance, Control, and Dynamics*.

Bradski, G. (2000). The OpenCV library. *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, 25(11):120–123.

Canteli, L. and other (2017). Autonomous landing of a UAV on a moving vehicle for the MBZIRC. In *International Conference CLAWAR*, page 197. World Scientific.

Falanga, D., Zanchettin, A., Simovic, A., Delmerico, J., and Scaramuzza, D. (2017). Vision-based autonomous quadrotor landing on a moving platform. In *Int. Symp. on Safety, Security, and Rescue Robotics (SSRR)*.

Fu, M., Zhang, K., Yi, Y., and Shi, C. (2016). Autonomous landing of a quadrotor on an UGV. In *2016 IEEE International Conference on Mechatronics and Automation*, pages 988–993.

Gomez-Balderas, J. E., Flores, G., García Carrillo, L. R., and Lozano, R. (2013). Tracking a ground moving target with a quadrotor using switching control. *J. Intell. Robotics Syst.*, 70(1-4):65–78.

Guo, Z. and Hall, R. W. (1989). Parallel thinning with two-subiteration algorithms. *Commun. ACM*.

Hoang, T., Bayasgalan, E., Wang, Z., Tsechpenakis, G., and Panagou, D. (2017). Vision-based target tracking and autonomous landing of a quadrotor on a ground vehicle. In *American Control Conference*.

Kalal, Z., Mikolajczyk, K., and Matas, J. (2012). Tracking-learning-detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1409–1422.

Kannala, J. and Brandt, S. S. (2006). A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Kim, J., Jung, Y., Lee, D., and Shim, D. H. (2014). Outdoor autonomous landing on a moving platform for quadrotors using an omnidirectional camera. In *Int. Conf. on Unmanned Aircraft Systems (ICUAS)*.

Kim, J., Jung, Y., Lee, D., and Shim, D. H. (2016). Landing control on a mobile platform for multi-copters using an omnidirectional image sensor. *Journal of Intelligent & Robotic Systems*, 84(1):529–541.

Krajník, T., Nitsche, M., Faigl, J., Vaněk, P., Saska, M., Přeučil, L., Duckett, T., and Mejail, M. (2014). A practical multirobot localization system. *Journal of Intelligent & Robotic Systems*, 76(3-4):539–562.

Kusumam, K., Krajník, T., Pearson, S., Duckett, T., and Cielniak, G. (2017). 3d-vision based detection, localization, and sizing of broccoli heads in the field. *Journal of Field Robotics*, 34(8):1505–1518.

Lange, S., Sünderhauf, N., and Protzel, P. (2009). A vision based onboard approach for landing and position control of an autonomous multirotor uav in gps-denied environments. In *Int. Conf. on Adv. Rob. (ICAR)*.

Lightbody, P. et al. (2017). An efficient visual fiducial localisation system. *SIGAPP Appl. Comp. Review*.

Lin, S., Garratt, M. A., and Lambert, A. J. (2017). Monocular vision-based real-time target recognition and tracking for autonomously landing an uav in a cluttered shipboard environment. *Autonomous Robots*.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*.

Matas, J., Chum, O., Urban, M., and Pajdla, T. (2004). Robust wide-baseline stereo from maximally stable extremal regions. *Image and vision computing*, 22(10):761–767.

Nieuwenhuisen, M. et al. (2017). Collaborative object picking and delivery with a team of micro aerial vehicles at MBZIRC. In *European Conference on Mobile Robots (ECMR)*, pages 1–6. IEEE.

Olson, E. (2011). Apriltag: A robust and flexible visual fiducial system. In *International Conference on Robotics and Automation (ICRA)*, pages 3400–3407. IEEE.

Pěnička, R., Faigl, J., Váňa, P., and Saska, M. (2017). Dubins orienteering problem. *IEEE Robotics and Automation Letters*, 2(2):1210–1217.

Saripalli, S., Montgomery, J. F., and Sukhatme, G. S. (2003). Visually guided landing of an unmanned aerial vehicle. *IEEE transactions on robotics and automation*, 19(3):371–380.

Saska, M., Krajník, T., and Přeučil, L. (2012). Cooperative μUAV-UGV autonomous indoor surveillance. In *Systems, Signals and Devices (SSD), 2012 9th International Multi-Conference on*, pages 1–6. IEEE.

Xu, L. and Luo, H. (2016). Towards autonomous tracking and landing on moving target. In *2016 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, pages 620–628.

Yang, S., Scherer, S., and Zell, A. (2012). An Onboard Monocular Vision System for Autonomous Takeoff, Hovering and Landing of a Micro Aerial Vehicle. *Journal of Intelligent & Robotic Systems*.