IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), Abu Dhabi, UAE, 2020.
Best Paper Award.

Visually Guided Balloon Popping with an Autonomous MAV at MBZIRC 2020

Marius Beul, Simon Bultmann, Andre Rochow, Radu Alexandru Rosu,
Daniel Schleich, Malte Splietker, and Sven Behnke

Abstract— **Visually guided control of micro aerial vehicles (MAV) demands for robust real-time perception, fast trajectory generation, and a capable flight platform. We present a fully autonomous MAV that is able to pop balloons, relying only on onboard sensing and computing. The system is evaluated with real robot experiments during the Mohamed Bin Zayed International Robotics Challenge (MBZIRC) 2020 where it showed its resilience and speed. In all three competition runs we were able to pop all five balloons in less than two minutes flight time with a single MAV.**

I. Introduction

In order to advance the state of the art in autonomous mobile robots, the Mohamed Bin Zayed International Robotics Challenge (MBZIRC) 2020 [1], which took place in February 2020 in Abu Dhabi posed multiple challenges. One of the tasks was to pop balloons in an outdoor arena of size $90\,\mathrm{m} \times 40\,\mathrm{m}$. Five green balloons with approximately $60\,\mathrm{cm}$ diameter were randomly placed inside on top of $2.5\,\mathrm{m}$ long poles. Although the total challenge time was set to $15\,\mathrm{min}$, the task had to be completed much faster and autonomously to receive a high score. Up to three micro aerial vehicles (MAV) could be used to complete the challenge, but we found it sufficient to use only one MAV. While global navigation satellite system (GNSS) positioning was available, the use of differential GNSS was penalized. In this paper, we present our integrated MAV system "Jelly", specifically designed to pop balloons including

- a custom-tailored hardware design,
- fast perception accelerated by a Tensor Processing Unit,
- robust filtering of sensor data, and
- fast trajectory generation and control.

We evaluate our approach with real robot experiments and report results from the MBZIRC 2020 competition. Fig. 1 shows Jelly popping one balloon with its tentacles.

II. Related Work

At present time, no other group has presented complete systems to this specific task. However, much related research deals with subtasks posed in this MBZIRC 2020 challenge like visual object detection, lightweight computer vision models for deployment on MAVs, or precise MAV control.

This work has been supported by a grant of the Mohamed Bin Zayed International Robotics Challenge (MBZIRC) and the German Federal Ministry of Education and Research (BMBF) in the project "Kompetenzzentrum: Aufbau des Deutschen Rettungsrobotik-Zentrums (A-DRZ)"

Institute for Computer Science VI, Autonomous Intelligent Systems, University of Bonn, Endenicher Allee 19a, 53115 Bonn, Germany, mbeul@ais.uni-bonn.de

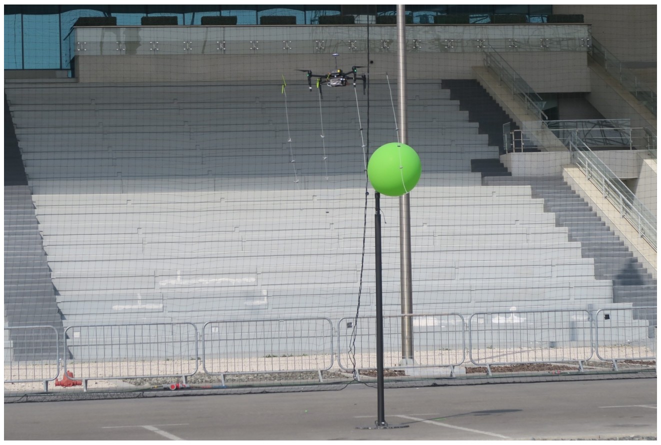


Fig. 1: Our MAV "Jelly" just before it pops the balloon with its spiked tentacles.

Rodriguez et al. [2] use a convolutional neural network to not only detect circular objects, but also other pretrained objects in real time. Circular objects can be easily detected and differentiated from non-circular ones based on the shape of their contour. Yang et al. [3] propose an encoder-decoder structure for contour detection of generic foreground objects from the PascalVOC dataset [4]. Maninis et al. [5] use a CNN architecture to detect object contours at multiple scales together with their orientations, based on a generic backbone CNN, like ResNet [6]. We follow a similar approach, but train our contour detection network to detect only contours of one class of objects—the balloons (see Sec. III-C).

Lightweight computer vision models that can be executed efficiently also on mobile or embedded systems with very restricted computational power have been of increasing research interest during recent years. The MobileNet architectures [7], [8], for example, greatly reduce the number of parameters in a convolutional neural network (CNN) by replacing standard convolutions with depthwise-separable convolutions. For our vision system, we employ a standard ResNet architecture but with very few layers (cf. Sec. III-C), keeping the number of parameters and the necessary computational power low. Furthermore, specialized inference accelerators like the Google Edge TPU [9] can be used for efficient processing with limited size and energy budget. To make a trained CNN model compatible with the Edge TPU, weights and activations need to be quantized to 8-bit integer values, e.g. using the quantization scheme described in [10].

Also, fast real-time trajectory generation and control is an active area of research. Specifically, as a result of

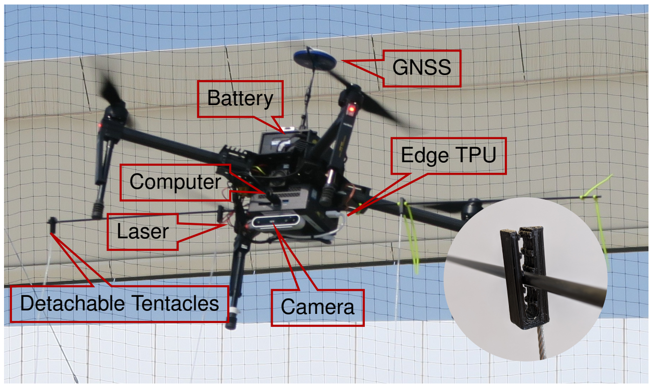


Fig. 2: Design of our MAV "Jelly" equipped with four spiked tentacles, an Intel RealSense D415 camera, a Google Edge TPU, a laser height sensor and a lightweight but powerful onboard computer. Bottom right: closeup of a 3D-printed detachable tentacle clamp with release force set to medium.

MBZIRC 2017, various groups presented advanced control approaches for MAVs. The team from Czech Technical University Prague reports their approaches to landing on a moving platform during the MBZIRC 2017 in [11]. Also, Cantelli et al. and Battiato et al. from the University of Catania report their systems [12], [13] including their control approach. Falanga et al. [14] of University of Zurich plan jerk-minimizing trajectories using a fast analytic polynomial generation method similar to ours. Also, outside of MBZIRC, many groups employ polynomial trajectories for MAV control. For a comparison of polynomial trajectory generation algorithms, see the works of Ezair et al. [15].

III. System Setup

In the following sections, we first describe the hardware of our MAV in Sec. III-A. We continue by presenting the mission control state machine in Sec. III-B. Our balloon perception pipeline including an allocentric filter is detailed in Sec. III-C and Sec. III-D. GNSS-based state estimation is supported by a laser height filter which we describe in Sec. III-E. Lastly, our trajectory generation and control method is presented in Sec. III-F.

A. Hardware

Our MAV, shown in Fig. 2, is based on the DJI Matrice 100 platform. It is equipped with a small but fast Gigabyte GB-BSi7T-6500 onboard PC with an Intel® Core™ i7-6500U CPU running at 2.5/3.1 GHz and 16 GB of RAM. Balloons are perceived by an Intel RealSense D415 depth camera [16] with the assistance of a Google Edge TPU USB accelerator [9]. For precise height estimation, the MAV uses a downwards facing LIDAR-Lite v3 [17].

Balloons are punctured with four detachable spiked 1.4 m long tentacles, mounted on a horizontal bar with a distance of 30 cm. When an adjustable force (set to ≈2 N during the challenge) is applied to a tentacle, e.g., by entangling with the poles, it is removed, preventing the MAV from crashing. On each tentacle, four needle-spiked hemispheres are mounted with 15 cm distance. By using flexible popping hardware, our MAV complied to the size restrictions of $1.2\,\mathrm{m} \times 1.2\,\mathrm{m} \times 0.5\,\mathrm{m}$, still offering a forgiving popping system that does not require centimeter-level precision.

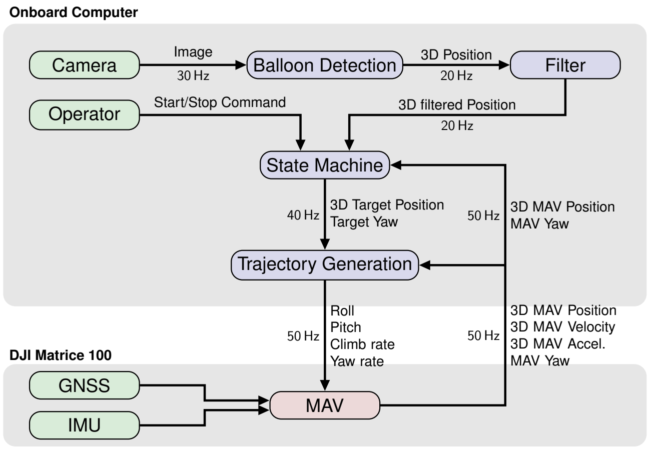


Fig. 3: Structure of our method. Green boxes represent external inputs like sensors, blue boxes represent software modules, and the red box indicates the MAV flight control. All software components use ROS as middleware. Position, velocity, acceleration, and yaw are allocentric.

For allocentric localization and state estimation, we employ the filter onboard the DJI flight control that incorporates GNSS and IMU data.

To make all components easily transferable between the test area at our lab and also different arenas on site, we defined all coordinates (x, y, z, yaw) in a field-centric coordinate system. The center and orientation of the current field were broadcasted by a base station PC to the MAV. Since we do not make any assumption about the allocentric movement of the target, navigation is purely relative to the detected target and not affected by global inaccuracies. In contrast to other teams, we did not use advanced satellite-based localization methods like Real Time Kinematic positioning (RTK-GPS) that need multiple GPS antennas on the MAV.

Fig. 3 gives an overview of the information flow in our system. We use the robot operating system (ROS) as middleware on the MAV and the ground control station. We communicate over WiFi with a robust UDP protocol, developed for connections with low bandwidth and high latency [18].

B. Mission Control State Machine

The behavior of the MAV is controlled by a state machine that serves as a generator for waypoints and headings for the subsequent control layers. It also ensures that the MAV does not exceed arena limits and stays within a defined altitude corridor, so that it stays always above the balloon mounting poles and below the 5.0 m minimum altitude of the other subchallenge's MAVs. Fig. 4 shows a flowchart of our state machine, which consists of two alternating parts—Search and Pop. In search mode, the MAV flies a repeating creeping-line pattern along the long axis of the field, thereby scanning the entire arena. In Pop mode, the MAV flies a trajectory that drags the tentacles through detected balloons.

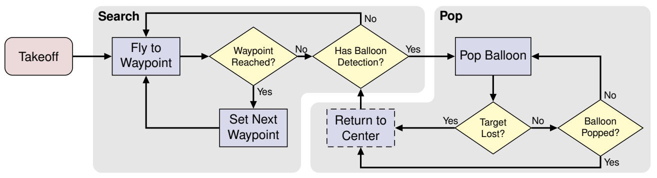


Fig. 4: The flowchart of our state machine.

The MAV velocity in search mode is tuned to $5.0\,\mathrm{m/s}$ and the altitude is $4.0\,\mathrm{m}$, so that balloons can be reliably searched from a safe height. Our balloon detector produces reliable position estimates at ranges over $30\,\mathrm{m}$. During the Grand Challenge, the search pattern comprised two search lanes, spaced at $10\,\mathrm{m}$ from the arena limits, which proved sufficient. As shown in Fig. 3, all balloon detections are filtered (Sec. III-D) before being processed by the state machine. The filter provides a list of verified balloon positions which are within the arena limits, including those which are currently out of view. Once the state machine receives at least one detection, it proceeds to approach the closest target.

In Pop mode, a straight-line trajectory is computed, such that the center of the tentacles passes through the balloon center at a non-zero velocity. The tentacles' upwards facing needles are dragged into the balloon surface, effectively puncturing it. As the forward-facing camera cannot perceive the balloon all the way, it is assumed to be popped once the MAV passes over the estimated center of the balloon instance within $0.5\,\mathrm{m}$ radius. Should the balloon be still intact, due to unsuccessful puncturing or missing the intercept point, it will be tackled again later as the search pattern repeats and the balloon will inevitably be re-detected. If the target is lost during the approach, e.g. because the filter discarded a false positive, the attempt is cancelled. As an addition in the Grand Challenge, after each attempt to pop a balloon, the MAV returns to the center of the field in order to prevent flying into the scaffolding protruding into the arena. This method of handling the non-convexity of the field is simple but it introduces additional flying time as compared to real obstacle avoidance. On the other hand, it is easy to implement and reliable. After returning to the center, the MAV targets the subsequent closest balloon provided by the filter or resumes with the search pattern, if there are no viable balloon hypotheses.

C. Balloon Perception

Our approach for detecting balloons in images is based on deep learning methods and split into an inference and a postprocessing step (cf. Fig. 5). During the inference step a neural network for semantic segmentation is employed. Since we aim to detect multiple balloons, we perform a binary segmentation of the raw input image and extract the balloon outlines as shown in Fig. 8. The balloon detection itself is carried out in the postprocessing step, which provides information like the number of balloons as well as their confidence values and positions in camera coordinates.

a) Balloon Outline Segmentation Network: The structure of the neural network is simple and based on the first three blocks of ResNet-18 [6]. See Fig. 6 for a visualization of our network. Each block consists of two 3×3 convolutions and calculates

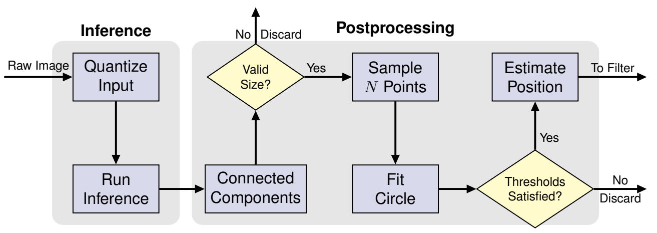


Fig. 5: Perception pipeline: Input is the raw image and output is a number of balloon detections with 3D-positions in the camera coordinate system.

$$x_{\mathrm{out},i} = \mathrm{ReLu}\left(x_i + \mathrm{conv}_2\left(\mathrm{ReLu}\left(\mathrm{conv}_1\left(x_i\right)\right)\right)\right) . \qquad (1)$$

The last convolutional layer performs a reduction from 64 to $C = 2$ feature maps $f_c, c \in \{0, \ldots, C-1\}$. These feature maps are used for a binary classification consisting of background and balloon outlines classes. The network is trained with weighted Negative Log-Likelihood Loss (wNLL), after calculating the pixel-wise $\mathrm{LogSoftmax}$ from the output feature maps f_c:

$$q_{i,c} = \mathrm{LogSoftmax}\left(x_{i,c}\right) = \log\left(\frac{e^{x_{i,c}}}{\sum_{c'=0}^{C-1} e^{x_{i,c'}}}\right) , \qquad (2)$$

$$\mathrm{wNLL}(x_i, y_i) = -w_{y_i} q_{i,y_i} , \qquad (3)$$

for every pixel $x_{i,c}$ in the feature maps f_c and ground truth label y_i. The weights w_c are used to compensate the class imbalance of the training set.

The weights of the first convolutional layer are initialized from ImageNet-pretrained ResNet-18. The network uses a receptive field of 5 and a stride of 2 to sample down the input image, since ResNet uses time consuming convolutions and inference has to run in real time. This initialization results in faster training and better inference results than random initialization. For training, a dataset consisting of 10,000 synthetic and 300 real images was used, mostly consisting of single balloons and just a few of them containing multiple balloons. The synthetic images were generated by a lightweight physically-based renderer [19]. A 3D mesh of a $60\,\mathrm{cm}$ diameter sphere is randomly placed in different HDR environments in order to capture realistic lighting as shown in Fig. 7. To reduce the number of false positives, the images also include spherical shapes that are not green and do not correspond to a balloon. The real images were recorded with the same Intel RealSense D415 camera which was used during the competition. To enhance generalization of the network even further, we added noise to the synthetic data as described in [20].

The network was trained with image size 960×540. The ground truth size is 480×270 and shows white balloon outlines with a thickness of 1 pixel on a black background. However, we decided to use a 3×3 dilation on the output

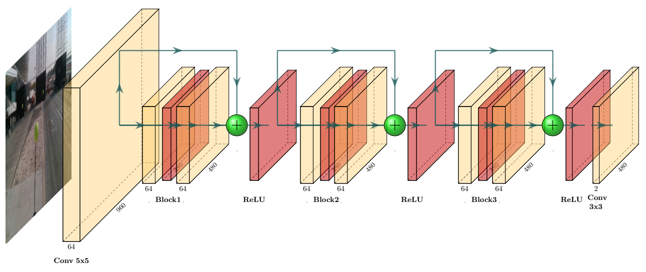


Fig. 6: Architecture of the ball outline segmentation network.

image to enhance chances to detect a balloon as a fully connected component and make the detections more invariant to noise. Training was accomplished with a batch size of $B = 12$ and a learning rate of $lr = 0.001$ using the Adam optimizer on a Nvidia GeForce GTX 1080 Ti. In total, pretraining took 130 epochs, followed by finetuning during the competition for another 50 epochs. On site, we added 250 additional annotated images, which our MAV captured during rehearsals, resulting in a total training time of $9.5\,\mathrm{h}$.

b) Postprocessing: The purpose of the postprocessing step is to detect balloons in the binary segmentation output as shown in Fig. 8. In the best case, the binary segmentation provides all outlines of the balloons with no noise in the background. Since the circle Hough transform is very time consuming and inefficient for oval shapes (like balloons often are) we use a pipeline which processes the segmentation output in several steps. In the first step, connected components are extracted and valid components are filtered by a minimum number of pixels. Furthermore, N points are sampled equally distributed on each connected component, starting with the largest one descending. To fit a circle into the sampled points $x_i, i \in \{1, \ldots, N\}$ we estimate a center point c and the radius

$$R_{\text{mean}}(c) = \frac{1}{N} \sum_{i=1}^{N} \|x_i - c\| \,. \tag{4}$$

To estimate the optimum circle center $\hat{c}$ we minimize the residuals

$$r(c) = \sum_{i=1}^{N} \left(\|x_i - c\| - R_{\text{mean}}(c) \right)^2 , \tag{5}$$

$$\hat{c} = \underset{c \in \mathbb{R}^2}{\arg\min}\, r(c) \,. \tag{6}$$

An estimate of the quality of the fitted circle is given by the normalized residuals $r_{\text{norm}}(\hat{c}) = \sqrt{\frac{r(\hat{c})}{N}}$. Detections can be filtered by a size threshold λ_{radius} and a residual threshold parameter λ_{res}, which are empirically determined by evaluating recorded test data. Fitted circles $\hat{c}$ are accepted as valid detections if $R_{\text{mean}}(\hat{c}) < \lambda_{\text{radius}}$ and $r_{\text{norm}}(\hat{c}) < \lambda_{\text{res}}$.

The detector outputs 2D balloon centers and their radii in pixel coordinates. A 3D position estimate can be calculated based on the balloon radius in the detections R_{mean} and in real $R_{\text{real}} = 30\,\mathrm{cm}$ which was fixed and known. The additional information R_{real} motivated us to estimate the depth of balloons without using the given depth image of our Intel RealSense Camera, since we observed noisy depth and a not negligible computation effort. At first, we project the balloon center point $r_1 = (c_u, c_v, 1)$ and the point at the right balloon outline $r_2 = (c_u + R_{\text{mean}}, c_v, 1)$ into 3D camera coordinates at unit depth using the camera matrix

$$K = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix} .$$

The transformed points are $p_{1/2} = K^{-1} r_{1/2}$, with angle $\alpha = \arccos\left(\frac{p_1 \cdot p_2}{\|p_1\| \|p_2\|}\right)$ between them. Further, the scalar s, which transforms p_1 from unit depth to the 3D balloon center point $P_{\text{m}} = s p_1$ in metric scaling is calculated as $s = \frac{R_{\text{real}}}{\tan \alpha}$. The resulting 3D balloon center points are then further processed by an allocentric filter (cf. Sec. III-D).

The entire pipeline is very time efficient and runs with an average processing time of $45\,\mathrm{ms}$ per frame on the used Intel i7-6500U CPU with the Google Edge TPU connected over USB 3.0. Since the balloons are static, this computation time is more than sufficient as results during the contest confirmed. The quantization of the network for processing on the Edge TPU did not lead to any decrease in prediction quality. The small receptive field allows to easily detect multiple balloons in images as shown in Fig. 8.

Finetuning during the competition resulted in a significant decrease of background noise and enhanced the balloon outline detection of the network. Consequently, parameters like residual threshold and minimum connected component size in postprocessing were adapted, so that balloons were detected even at large distances of up to $50\,\mathrm{m}$.

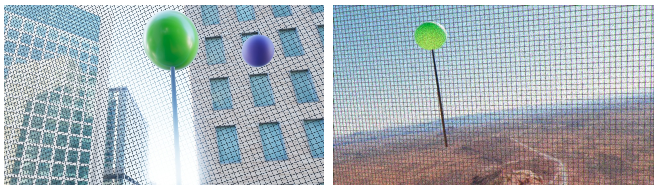
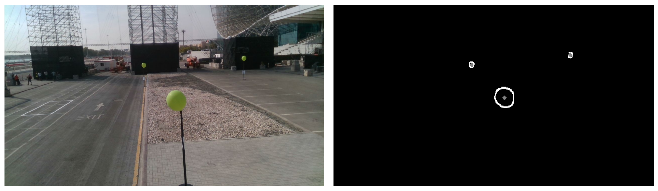


Fig. 7: Synthetic images generated using EasyPBR [19].



(a) Input image (b) Balloon detections

Fig. 8: Balloon perception: The detected balloon outlines are drawn with white lines and the centers are marked with gray points. All visible balloons are detected, even at large distances, without false positives in the background.

D. Balloon Filter

For each image frame, the balloon perception (Sec. III-C) outputs a list of current balloon detections, described as egocentric 3D positions in camera coordinates. These are processed by a filter to reject outliers and to aggregate them into a list of hypotheses $\mathcal{H}$ of possible balloon positions. Each hypothesis $\mathcal{H}_i \in \mathcal{H}$ consists of

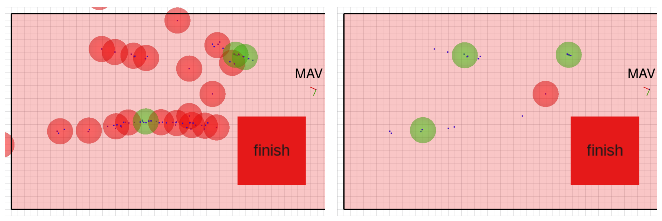


Fig. 9: Evaluation of different distance metrics. Detections are shown as blue dots. Hypotheses are shown as spheres, which are colored green if at least eight detections are assigned to them and red otherwise. **Left:** Euclidean distance on ground plane. **Right:** Distance to detection ray.

- a history $\mathcal{D}_i := (d^i_1, \dots, d^i_8)$ of the last eight detections that were assigned to it,
- an estimate of the balloon position $P_i := \frac{1}{|\mathcal{D}_i|} \sum_{d \in \mathcal{D}_i} d$, calculated as the running average over the detection history, and
- a counter for missed detections.

All hypotheses with at least eight detections are sorted with increasing distance to the current MAV position and forwarded to the state machine. In the following, we describe the assignment of detections to the different hypotheses and the removal of hypotheses which were created by false detections based on the missed detection counter.

a) Detection Assignment: In a first step, the egocentric detections of the balloon detector are transformed into allocentric field coordinates. Since the height of the balloons was predefined to be at $2.5\,\mathrm{m}$, all detections outside a height corridor from 1.5 to $5.0\,\mathrm{m}$ are discarded.

For each remaining detection d, we determine the closest hypothesis $\mathcal{H}_{i^*}$ by minimizing the distance between the detection and all position estimates i.e., choosing $i^* = \arg\min_i\{\mathit{dist}(d, P_i)\}$. If the distance is smaller than a threshold of $2.0\,\mathrm{m}$, we assign d to $\mathcal{H}_{i^*}$, otherwise we create a new hypothesis. Finally, hypotheses are merged when their estimated balloon positions become closer than $2.0\,\mathrm{m}$.

The choice of the distance measure $\mathit{dist}(\cdot)$ is important to reduce the influence of detection noise and thus to achieve accurate assignments. Since the center height of all balloons is fixed to $2.5\,\mathrm{m} + \frac{0.6\,\mathrm{m}}{2} = 2.8\,\mathrm{m}$, $\mathit{dist}(\cdot)$ can be chosen as the Euclidean distance on the ground plane to eliminate noisy height measurements. However, due to the noisy depth estimation of the egocentric detections, this may result in multiple different hypotheses for the same balloon (Fig. 9 (left)). Instead, we cast a ray τ in the direction of the detection and define $\mathit{dist}(\cdot)$ to be the distance between the estimated balloon position and τ. This results in more accurate hypotheses assignments as shown in Fig. 9 (right).

b) Hypotheses Removal: Once the MAV reaches a position above an estimated balloon, we assume the balloon to be popped and remove the corresponding hypothesis. If popping was not successful, the balloon will be detected again later and thus a new hypothesis for this balloon will be added (cf. Sec. III-B).

To further remove hypotheses created by false detections, we estimate the visibility of each hypothesis by projecting the estimated balloon coordinates onto the image plane of the camera. Whenever a hypothesis is assumed to be visible but is not detected in the current frame, the corresponding missed detection counter is incremented. If the number of missed detections exceeds a threshold of 30, the corresponding hypothesis is removed.

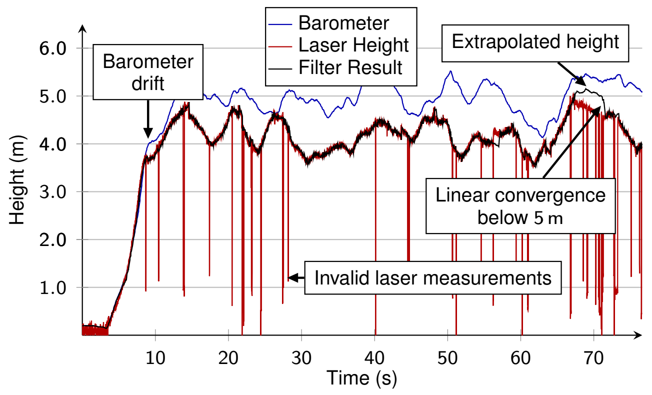


Fig. 10: Laser height correction. During takeoff, height estimation (black) is based on barometer data (blue) since laser measurements (red) are unreliable for too close distances. Once the MAV reaches an estimated height of $1.0\,\mathrm{m}$, height estimation is based on filtered laser data. Above $5.0\,\mathrm{m}$ the laser becomes unreliable in the bright outdoor conditions and the height estimate is extrapolated using the change in barometric height measurements. Excerpt of Run 2.

E. Laser Height Filter

Precise height estimation can make the critical difference between popping a balloon, missing a balloon, or hitting a pole. We therefore use measurements of a downwards facing LIDAR-Lite v3 as primary height source.

Detecting outliers and fusing the laser measurements with barometer data using an EKF would require to manually estimate sensor covariances. However, we aim to use simple methods which allow fast debugging during the competition. Thus, we choose an approach similar to the one that already proofed successful in MBZIRC 2017 (see [21]).

A laser height filter determines whether the laser measurements are valid and thus can be used as height estimation. However, when the laser measurements are assumed invalid, we extrapolate the latest height estimate using the change in the fused GNSS and barometric height. As soon as the laser measurement is assumed valid again, we immediately correct our height estimate to the laser height or—if the extrapolated height drifted too much—linearly interpolate, allowing a maximum slope of $1.5\,\mathrm{m/s}$.

To determine whether a given laser measurement is valid, we check the following criteria:

- Laser measurements below $1\,\mathrm{m}$ are assumed to be invalid as they might be caused by measuring the spiked tentacles of our MAV.
- Laser measurements become unreliable for very low or large distances. Thus, we only consider laser measurements if the current height estimate is within 1.0 and

TABLE I: Parameters used at MBZIRC 2020.

Parameter	Axis	Value	Parameter	Axis	Value
v_{max}	X,Y	$5.0\,\mathrm{m/s}$	v_{max}	Z	$1.0\,\mathrm{m/s}$
a_{max}	X,Y	$4.0\,\mathrm{m/s^2}$	a_{max}	Z	$10.0\,\mathrm{m/s^2}$
j_{max}	X,Y	$5.0\,\mathrm{m/s^3}$	j_{max}	Z	$50.0\,\mathrm{m/s^3}$

$5.0\,\mathrm{m}$. Mind that this estimate might be extrapolated using GNSS/ barometric data.

- To reject outliers, we discard all laser measurements that differ more than $15\,\mathrm{cm}$ from the latest valid laser measurement. The first valid laser measurement is bootstrapped by collecting ten measurements which pass the above criteria and selecting the one with the most inliers within a $15\,\mathrm{cm}$ range. To be able to recover after a longer sequence of invalid measurements, this initialization process is repeated each time 100 subsequent measurements (which corresponds to a time interval of $1\,\mathrm{s}$) have been rejected.

Fig. 10 depicts height measurements and the filtered height estimates during Run 2.

F. Trajectory Generation and Control

Since the total time needed for the challenge is crucial and the MAV has to precisely hit the balloons, our method for trajectory generation and control is based on the method that already reliably worked during MBZIRC 2017 (see [22] and [21]). The method is described in detail in [23] with the extensions from [24]. For reasons of brevity, in this section, we cover only the most important aspects of the algorithm.

Based on a simple triple integrator model, our method analytically generates third-order time-optimal trajectories that satisfy input ($j_{min} \leq j \leq j_{max}$) and state constraints ($a_{min} \leq a \leq a_{max}$, $v_{min} \leq v \leq v_{max}$). Trajectories are computed from the current state $(p, v, a)^{\intercal}_{MAV}$ to the target state $(p, v, a)^{\intercal}_{target}$. The X,Y, and Z axis are synchronized to arrive at the target state at the same time. By doing so, the MAV flies on a relatively straight path.

We directly use this trajectory generation method as a model predictive controller (MPC), running in a closed loop with $50\,\mathrm{Hz}$. Our hardware does not support direct execution of sent jerk commands. We therefore assume pitch and roll to directly relate to $\theta = \mathrm{atan2}(a_x, g)$ and $\phi = \mathrm{atan2}(a_y, g)$. Thus, we send smooth pitch θ and roll ϕ commands for horizontal movement and smooth climb rates v_z instead. We use the parameters presented in Tab. I.

Although an arbitrary number of axes can be controlled by the above-mentioned method, we do not consider the yaw-axis Ψ to be synchronized with the x, y and z-axis. For simplicity, we use proportional control for the yaw-axis. The yaw rate setpoint $\dot{\Psi}_{setp} = K_p \cdot (\Psi_{target} - \Psi_{MAV})$ with $K_p := 2.5$ for Run 1 and 2 and $K_p := 1.0$ for the Grand Challenge is sent to the MAV flight controller. The gain was reduced during the competition because of safety concerns (cf. Sec. IV-B).

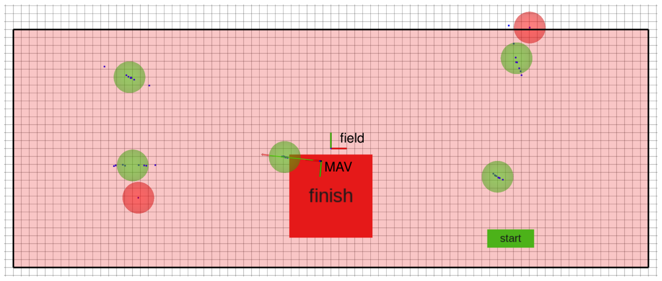


Fig. 11: World model 15 s after takeoff during Run 2: Individual detections (blue squares) are accumulated into balloon hypotheses. All five balloons are detected (green circles). Outlier detections result in unconfirmed balloon hypotheses (red circles). The MAV is about to approach the first balloon. The goal pose (red arrow) is set right behind a balloon hypothesis. The field area is shaded in red, its origin is the *field* frame.

IV. Evaluation

Our MAV system for balloon popping was operated in three competition runs during MBZIRC 2020, which are evaluated below. A video showcasing the evaluation of our Grand Challenge Run can be found on our website[1].

A. Time Until Completion

As our vision system detects the balloons at large distances (cf. Sec. III-C), balloon hypotheses are added to the world model shortly after takeoff during all runs. During the second run, all five balloons were known to the filter only 15 s after takeoff (see Fig. 11). In the other runs, only a fraction of the balloons was inside the field of view directly after takeoff. The known target hypotheses are approached right away, the MAV then flies on a search pattern for a short time only until it detects the remaining targets. The times at which the respective balloons were punctured are given in Tab. II.
In the first run, two balloons were popped after $30\,\mathrm{s}$. Then a reset occurred for $8\,\mathrm{min}$, as the MAV had gotten stuck in the Net at the arena borders due to an error in the GNSS-based geofencing. The challenge was completed after $9\,\mathrm{min}\ 28\,\mathrm{s}$, but only $1\,\mathrm{min}\ 28\,\mathrm{s}$ flight time.
In the second run, the first two balloons were punctured right in sequence. Then, however, two balloons were missed—the puncturing did not work due to a suboptimal flight pattern (see Sec. IV-B). By repeating the search pattern and re-approaching the missed targets, in this run, all balloons were punctured after a total duration of $1\,\mathrm{min}\ 40\,\mathrm{s}$.
In the final run during the Grand Challenge, all balloons were punctured in the first attempt. The time between two consecutive balloons were very similar, (12-15 s). Between Balloon 2 and 3, the MAV flew a search pattern to discover the remaining ones, which explains the longer time interval. The challenge was completed after a total time of $1\,\mathrm{min}\ 21\,\mathrm{s}$, the shortest duration of all three competition runs.

[1]www.ais.uni-bonn.de/videos/ssrr_2020_mbzirc

TABLE II: Timings of the balloon popping.

	Balloon 1	Balloon 2	Balloon 3	Balloon 4	Balloon 5
Run 1*	23 s	30 s	9 min 4 s	9 min 11 s	9 min 28 s
Run 2	17 s	23 s	51 s	1 min 15 s	1 min 40 s
Run 3	11 s	27 s	56 s	1 min 8 s	1 min 21 s

(*) 8 min reset time between 2nd and 3rd balloon.

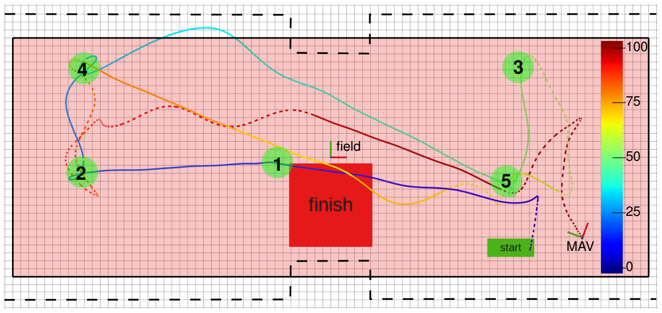
B. Flight Path

During the first and second run, the MAV always chose the direct path between two consecutive balloons. In the ideal case, this results in the shortest duration between two consecutive balloons (e.g. 6 s between the first and second balloon in Run 2). However, this can lead to the MAV flying dangerously close to the arena borders and could even have led to a crash during Run 2, had the controller chosen to pierce Balloon 3 and 4 in direct sequence. Our simple GNSS-based geofencing system which restricts the allowed flying area to a single rectangle could not correctly model the non-convex shape of the arena (see Fig. 12 (a)). Furthermore, the above described behavior results in the MAV turning right over the balloons as it rotates to approach the next target. This is a suboptimal flight pattern as it prevents the dangling tentacles from piercing the balloons successfully. During the second run, two balloons were missed due to this suboptimal maneuvering and needed to be approached a second and even a third time.

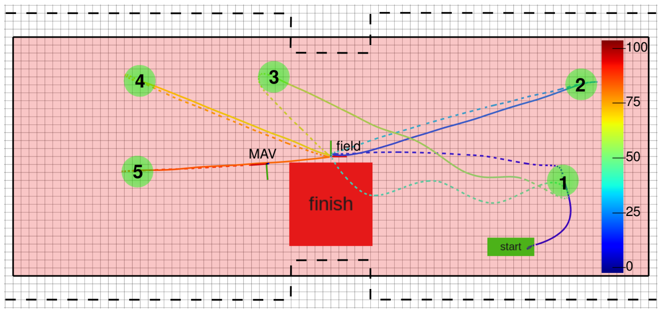
Turning above the balloons also had an impact on the safety of the flight maneuver. In Run 2, after popping Balloon 4, the MAV quickly turned while accelerating towards Balloon 5. Turning was so fast with $K_{p,yaw} = 2.5$, that the MAV was not able to redirect the acceleration fast enough due to the low jerk limit. Thus, the MAV first flew into a wrong direction—even entering the safety margin of the field—before correcting itself (see Fig. 12 (a) top-left). The trajectory rollout showed that the MAV was completely aware that it would leave the allowed area, but it simply could not compensate the erroneous acceleration fast enough. We prevented this possible safety hazard by setting $K_{p,yaw} = 1.0$, thus slowing down the yaw rate.

To overcome the remaining problems, an additional waypoint was added in the middle of the arena after each balloon for the final run. This results in a star-shaped flight pattern (see Fig. 12 (b)) and the balloons being passed in a straight line, without turning above them. Consequently, each balloon was pierced in the first attempt. The time between two consecutive balloons is slightly higher than it was before in the ideal case, but almost constant for each target, as no misses occur (see Sec. IV-A). Therefore, the challenge could be completed faster using the new behavior. Moreover, the star-shaped flight pattern avoids trajectories close to the arena borders and leads to safe flight paths despite the non-convex arena outline without any additional obstacle avoidance system (cf. Sec. III-B).

We show snapshots of the performance in Fig. 13. With this performance, we placed 5th in Challenge 1 including another subchallenge and 2nd in the Grand Challenge including five other subchallenges.



(a) Run 2: The MAV chooses the direct path between consecutive balloons. In some cases, it turns directly above the balloons which prevents the piercing tentacles from working correctly. Balloons 4 and 5 need to be passed two resp. three times until successful puncturing. Flight path leaves allowed area at top left.



(b) Run 3: The MAV passes through the arena center after each balloon. It moves straight through the balloons, which leads to them being pierced reliably at the first attempt. Paths close to the boundaries are avoided by this strategy.

Fig. 12: Comparison of flight paths between Runs 2 and 3 (colored by time). Solid line: Pop mode, dashed line: Search mode. The allowed flying area is shaded in red, the physical arena boundaries are marked with a dashed black line. Balloon hypotheses are displayed as green circles.

V. Lessons learned

As a retrospect towards our design choices, we deem the choice of using flexible tentacles that are dangling below the MAV to be a good decision. The flexible tentacles ensure that Jelly cannot get accidentally caught in the pole and additionally allow to have imperfect positioning with respect to the balloon. Competing teams that have experimented with rigid mechanism to pop the balloon struggled as the MAV would collide with the pole if the position estimate is not accurate enough.

We also observed that multiple teams have opted to use a NVIDIA Jetson platform for running the MAV, including balloon detection networks [25], [26]. This might be an interesting direction to explore as the Edge TPUs, while powerful, are also limited in the architectures that can be executed on them.

VI. Conclusion

We have provided detailed insight into our robust MAV setup for quickly and robustly popping balloons. The viability of our approach has been demonstrated in a real-

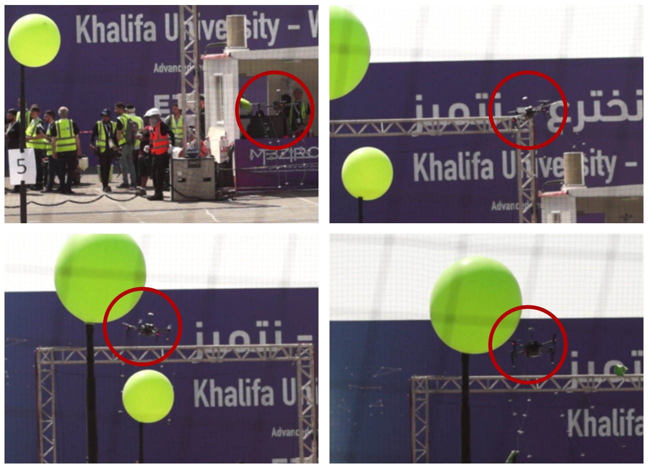


Fig. 13: Image sequence of popping the first balloon in the Grand Challenge (from top-left to bottom-right). The MAV (marked with the red circle) 1) takes off to $4.0\,\mathrm{m}$. 2) After detection of the first balloon, the MAV targets a position $2.0\,\mathrm{m}$ behind and $0.7\,\mathrm{m}$ above the center of the balloon. 3) It further accelerates to pass through the balloon with significant velocity. 4) It successfully pops the balloon. The entire shown process only takes $4.9\,\mathrm{s}$.

world scenario during the MBZIRC 2020 at which our MAV consistently performed as one of the fastest and most resilient among all competitors.

In particular, the robust vision pipeline including the meticulous filtering of outliers combined with a robust fault-tolerant hardware and an overall simple system architecture that allowed for quick adjustments in the field made our approach to this challenge a huge success. The same counts for the laboriously handcrafted laser height filter that rejected a large number of outliers from the noisy sensor and a well-tested state machine that could be easily adapted during the individual runs. Also, our low-level control approach proofed to be reliable, precise and fast. It guided Jelly under real-world conditions, including noisy sensor information and external disturbances.

We believe that our contribution, and in general all experience from the MBZIRC 2020, will inspire new ideas on how to operate flying robots in dynamic, real-world environments.

References

[1] MBZIRC, "MBZIRC challenge description," https://www.mbzirc.com/challenge/2020, 2019, accessed: 2020-03-16.

[2] D. Rodriguez, H. Farazi, G. Ficht, D. Pavlichenko, A. Brandenburger, M. Hosseini, O. Kosenko, M. Schreiber, M. Missura, and S. Behnke, "RoboCup 2019 AdultSize winner NimbRo: Deep learning perception, in-walk kick, push recovery, and team play capabilities," *RoboCup 2019: Robot World Cup XXIII*, pp. 631–645, 2019.

[3] J. Yang, B. Price, S. Cohen, H. Lee, and M.-H. Yang, "Object contour detection with a fully convolutional encoder-decoder network," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 193–202.

[4] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes (VOC) challenge," *Int. Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, 2010.

[5] K.-K. Maninis, J. Pont-Tuset, P. Arbeláez, and L. V. Gool, "Convolutional Oriented Boundaries: From Image Segmentation to High-Level Tasks," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 40, no. 4, pp. 819–833, 2018.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[7] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, arXiv:1704.04861.

[8] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[9] Google, "EdgeTPU USB Accelerator," https://coral.ai/docs/accelerator/datasheet, 2020, accessed: 2020-03-20.

[10] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[11] T. Baca, P. Stepan, and M. Saska, "Autonomous landing on a moving car with unmanned aerial vehicle," in *European Conf. on Mobile Robots (ECMR)*, 2017.

[12] L. Cantelli, D. Guastella, C. D. Melita, G. Muscato, S. Battiato, F. D'Urso, G. M. Farinella, A. Ortis, and C. Santoro, "Autonomous landing of a UAV on a moving vehicle for the MBZIRC," in *20th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines (CLAWAR)*, 2017.

[13] S. Battiato, L. Cantelli, F. D'Urso, G. M. Farinella, L. Guarnera, D. Guastella, C. D. Melita, G. Muscato, A. Ortis, F. Ragusa, and C. Santoro, "A system for autonomous landing of a UAV on a moving vehicle," in *Image Analysis and Processing (ICIAP)*, 2017.

[14] D. Falanga, A. Zanchettin, A. Simovic, J. Delmerico, and D. Scaramuzza, "Vision-based autonomous quadrotor landing on a moving platform," in *IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, 2017.

[15] B. Ezair, T. Tassa, and Z. Shiller, "Planning high order trajectories with general initial and final conditions and asymmetric bounds," *The Int. J. of Robotics Research*, vol. 33, no. 6, pp. 898–916, 2014.

[16] Intel, "Depth Camera D415 — Intel Realsense Depth and Tracking Cameras," https://www.intelrealsense.com/depth-camera-d415, 2020, accessed: 2020-03-20.

[17] Garmin, "Garmin Lidar-Lite v3," https://buy.garmin.com/de-DE/DE/p/557294, 2020, accessed: 2020-03-20.

[18] M. Schwarz, T. Rodehutskors, D. Droeschel, M. Beul, M. Schreiber, N. Araslanov, I. Ivanov, C. Lenz, J. Razlaw, S. Schüller, D. Schwarz, A. Topalidou-Kyniazopoulou, and S. Behnke, "NimbRo Rescue: Solving disaster-response tasks through mobile manipulation robot Momaro," *J. of Field Robotics*, vol. 34, no. 2, pp. 400–425, 2017.

[19] R. A. Rosu, "EasyPBR," https://github.com/RaduAlexandru/easy_pbr, 2020, accessed: 2020-03-20.

[20] A. Carlson, K. A. Skinner, R. Vasudevan, and M. Johnson-Roberson, "Modeling camera effects to improve visual learning from synthetic data," in *Computer Vision - ECCV Workshops*, 2018.

[21] M. Nieuwenhuisen, M. Beul, R. A. Rosu, J. Quenzel, D. Pavlichenko, S. Houben, and S. Behnke, "Collaborative object picking and delivery with a team of micro aerial vehicles at MBZIRC," in *European Conf. on Mobile Robots (ECMR)*, 2017.

[22] M. Beul, S. Houben, M. Nieuwenhuisen, and S. Behnke, "Fast autonomous landing on a moving target at MBZIRC," in *European Conf. on Mobile Robots (ECMR)*, 2017.

[23] M. Beul and S. Behnke, "Analytical time-optimal trajectory generation and control for multirotors," in *Int. Conf. on Unmanned Aircraft Systems (ICUAS)*, 2016.

[24] ——, "Fast full state trajectory generation for multirotors," in *Int. Conf. on Unmanned Aircraft Systems (ICUAS)*, 2017.

[25] L. A. Tony, S. Jana, A. Bhise, V. BV, M. S. Gadde, D. Ghose, R. Krishnapuram, *et al.*, "Vision based target interception using aerial manipulation," *arXiv:2009.13066*, 2020.

[26] R. Suarez Fernandez, A. Rodríguez Ramos, A. Alvarez, J. Rodríguez-Vazquez, H. Bavle, L. Lu, M. Fernandez-Cortizas, A. Rodelgo, A. Cobano, D. Alejo, D. Acedo, R. Rey, S. Martinez-Rozas, M. Molina, L. Merino, F. Caballero, and P. Campoy, "The Skyeye Team participation in the 2020 Mohamed Bin Zayed International Robotics Challenge," 02 2020.