Field Robotics, vol. 2 pp. 807-842. May 2022.
DOI: 10.55417/fr.2022027

Target Chase, Wall Building, and Fire Fighting: Autonomous UAVs of Team NimbRo at MBZIRC 2020

Marius Beul, Max Schwarz, Jan Quenzel, Malte Splietker, Simon Bultmann, Daniel Schleich, Andre Rochow, Dmytro Pavlichenko, Radu Alexandru Rosu, Patrick Lowin, Bruno Scheider, Michael Schreiber, Finn Süberkrüb, and Sven Behnke
Autonomous Intelligent Systems Group
University of Bonn
Bonn, Germany
mbeul@ais.uni-bonn.de

Abstract

The Mohamed Bin Zayed International Robotics Challenge (MBZIRC) 2020 posed diverse challenges for unmanned aerial vehicles (UAVs). We present our four tailored UAVs, specifically developed for individual aerial-robot tasks of MBZIRC, including custom hardware- and software components.

In Challenge 1, a target UAV is pursued using a high-efficiency, onboard object detection pipeline to capture a ball from the target UAV. A second UAV uses a similar detection method to find and pop balloons scattered throughout the arena.

For Challenge 2, we demonstrate a larger UAV capable of autonomous aerial manipulation: Bricks are found and tracked from camera images. Subsequently, they are approached, picked, transported, and placed on a wall.

Finally, in Challenge 3, our UAV autonomously finds fires using LiDAR and thermal cameras. It extinguishes the fires with an onboard fire extinguisher.

While every robot features task-specific subsystems, all UAVs rely on a standard software stack developed for this particular and future competitions. We present our mostly open-source software solutions, including tools for system configuration, monitoring, robust wireless communication, high-level control, and agile trajectory generation.

For solving the MBZIRC 2020 tasks, we advanced the state of the art in multiple research areas like machine vision and trajectory generation. We present our scientific contributions that constitute the foundation for our algorithms and systems and analyze the results from the MBZIRC competition 2020 in Abu Dhabi, where our systems reached second place in the Grand Challenge. Furthermore, we discuss lessons learned from our participation in this complex robotic challenge.

1 Introduction

The Mohamed Bin Zayed International Robotics Challenge (MBZIRC) aims to advance the state of the art in autonomous mobile robotics through robot competitions. Unique to this challenge is the focus on field robotics and the diversity of tasks that need to be carried out. In the 2nd MBZIRC competition, participants were required to track, follow, and interact with a target unmanned aerial vehicle (UAV), find

(a) Jelly (b) Chaser (c) Lofty (d) Splasher

Figure 1: Our UAVs in action during the MBZIRC 2020.

and pop balloons, build walls using a ground/aerial robot fleet, and finally detect and fight fires in indoor and outdoor environments.

In this article, we will describe the approach of Team NimbRo to the aerial-robot tasks of the MBZIRC 2020, detailing the fleet of UAVs that was developed (see Fig. 1). We will focus on the three sub-challenges individually, highlighting the specific solutions found for the complex problems they posed, but also discuss our overall approach and identify lessons learned from our successful participation in the competition—Team NimbRo reached second place in Challenge 2 (wall building) and second place in the combined Grand Challenge.

In addition to the overall system integration, our contributions include:

- Custom-tailored UAV hardware designs for the individual challenges,
- a common *software stack* for mission control and system monitoring, including a hardened FSM framework with accompanying design and visualization tools, robust network communication and remote supervision tools, as well as a universal model predictive controller,
- real-time *onboard* target perception and trajectory control for Challenge 1 (balloon hunt),
- onboard object detection, 6D pose estimation, and aerial manipulation control for Challenge 2 (wall building), and
- fully *autonomous* fire detection and extinguishing for Challenge 3 (fire fighting).

Many of the employed algorithms are based on state-of-the-art techniques, and their development advanced their field significantly. Since detailing every approach would go beyond the scope of this article, we highlight key aspects of our methods and refer to corresponding papers where applicable.

We evaluate our approach with real-robot experiments and report results from the MBZIRC 2020 competition. A description of our unmanned ground vehicle (UGV) approaches for MBZIRC 2020 can be found in Lenz et al. (2021).

2 Related Work

To our knowledge, no comprehensive analysis or review of the MBZIRC 2020 competition has yet been published. Still, many of the sub-problems that appeared in the individual challenges are highly active research areas, and we provide an overview of the current state of the art.

Aerial Search & Rescue: Michael et al. (2012) report on field experiments with UAVs and UGVs inside earthquake-damaged buildings. The authors provide details about their collaborative mapping approach and report results from the experiments in the form of maps generated by the individual robots and as a team. Rodriguez et al. (2019) use a convolutional neural network (CNN) to detect circular and other pre-trained 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. (2016) propose an encoder-decoder structure for contour detection

of generic foreground objects from the PascalVOC dataset (Everingham et al., 2010). Maninis et al. (2018) use a CNN architecture to detect object contours at multiple scales together with their orientations, based on a generic backbone CNN, like ResNet (He et al., 2016). We follow a similar approach but train our contour detection network to detect only contours of one class of objects—the balloons (see Sec. 4.3).

Lightweight computer vision models that can be executed efficiently also on mobile or embedded systems with limited computational power have been of increasing research interest during recent years. The MobileNet architectures (Howard et al., 2017; Sandler et al., 2018), for example, significantly reduce the number of parameters in a 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. 4.3), keeping the number of parameters and the necessary computational power low. Furthermore, specialized inference accelerators like the Google Edge TPU can be used for efficient processing with a 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 by Jacob et al. (2018).

Also, fast real-time trajectory generation and control is an active area of research. Specifically, as a result of the MBZIRC 2017, various groups presented advanced control approaches for UAVs. Baca et al. (2017) report the approach to landing on a moving platform during the MBZIRC 2017, employed by their team including the CTU Prague, UPenn and UoL.

Similarly, Cantelli et al. (2017) and Battiato et al. (2017) from the University of Catania report their systems, including their control approach. Falanga et al. (2017) from the University of Zürich plan jerk-minimizing trajectories using a fast analytic polynomial generation method similar to ours. Outside of the MBZIRC, many groups employ polynomial trajectories for UAV control. For a comparison of these approaches, see the work of Ezair et al. (2014).

Aerial Manipulation: Aerial manipulation has been investigated for some time (Ruggiero et al., 2018). Complex systems with fully actuated multi-DoF robotic arms have been built (Huber et al., 2013; Kim et al., 2013). Michael et al. (2011) as well as Tran et al. (2021) demonstrated carrying large items with multiple UAVs. Lindsey et al. (2012) demonstrated the assembly of structures with teams of small UAVs. This work relied on an external motion capture system and self-locking magnetic part connectors. Goessens et al. (2018) present a feasibility study of constructing real-scale structures with UAVs based on self-aligning Lego-like brick shapes.

Baca et al. (2020) present the UAV system for the MBZIRC 2020 wall building task of the team from CTU Prague, UPenn, and NYU. The team won the Wall-building as well as the Grand Challenge. Their approach for the treasure-hunt challenge in MBZIRC 2017 is described in (Loianno et al., 2018).

Real et al. (2021) demonstrates the approach from the University of Seville (GRVC) together with Tecnico Lisboa and CATEC. The authors placed 14nd in the Wall-building challenge and 4th in the Grand Challenge, together with Tecnico Lisboa.

A predecessor of our work is Challenge 3 of the MBZIRC 2017, where a team of UAVs was supposed to collect discs. Our entry (Beul et al., 2019) was quite successful and reached third place in this challenge, behind ETH Zürich (Bähnemann et al., 2019) and CTU Prague, UPenn, and UoL (Spurnỳ et al., 2019).

In contrast, the 2020 edition of the MBZIRC featured much heavier and larger objects, which could only be grasped on a specific spot and had to be placed in a specified pose. To this end, we designed a magnetic gripper that is guided using visual servoing and has five passive DoFs that allow flexibility during grasping but facilitate precise placement.

Krizmancic et al. (2020) describe a planning system for UAV-UGV cooperative wall building. Their planner allows exploitation of the unique characteristics of each platform. In contrast, our UAV for wall building uses a simple greedy behavior for selecting the next task—which was sufficient for MBZIRC 2020 since we

only had one UAV and the UAV/UGV walls were separate.

Fire Fighting: Cooperative monitoring and detection of forest and wildfires with autonomous teams of UAVs (Bailon-Ruiz and Lacroix, 2020) or UGVs (Ghamry et al., 2016) gained significant attention in recent years (Delmerico et al., 2019). While UGVs are able to carry larger amounts of extinguishing agents or drag a fire hose (Liu et al., 2016a), payload limitations impede the utility of UAVs. Aydin et al. (2019) investigated the deployment of fire-extinguishing balls by a UAV. Ando et al. (2018) developed an aerial hose pipe robot using steerable pressurized water jets for robot motion and fire extinguishing.

MBZIRC 2020 competitors Spurny et al. (2021); Jindal et al. (2021) from CTU Prague, UPenn and NYU present their fire-fighting system. Their UAV is capable to autonomously approach and extinguish fires in GNSS-denied environments after detecting a suitable building entrance. The authors report results from simulations, field tests, and from the MBZIRC 2020 competition where they (together with UPenn and NYU) placed 4th in the fire-fighting and won the Grand Challenge.

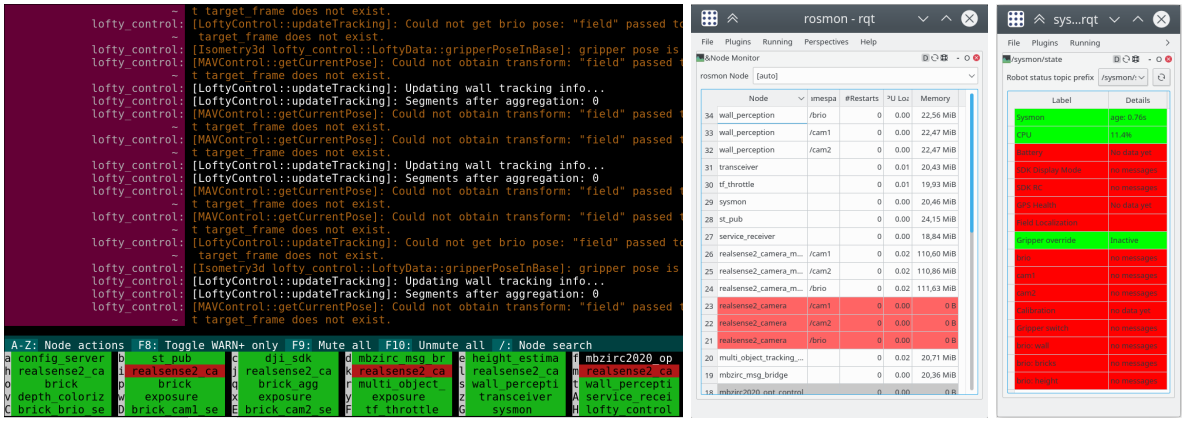
In urban environments, thermal mapping (Cho et al., 2015) is commonly used for building inspection. It relies on simultaneously captured color and thermal images from different poses and employs standard photogrammetry pipelines. Schönauer et al. (2013) provide real-time assistance for firefighters via thermal augmentation of live images within room-scale environments. In contrast, Borrmann et al. (2013) mounted a terrestrial LiDAR, a thermal camera, and a color camera on a UGV to obtain colorized point clouds, enabling automated detection of windows and walls (Demisse et al., 2015). Fritsche et al. (2017) cluster high-temperature points from fused 2D LiDAR and mechanically pivoting radar to detect and localize heat sources. Similarly, Rosu et al. (2019) acquire a thermal textured mesh using a UAV equipped with LiDAR and thermal camera and estimate the 3D position and extent of heat sources.

New challenges arise where robots have to operate close to structures. Therefore, UAVs are often equipped with cameras and are remote-controlled by first responders. In contrast, autonomous execution was the goal for Challenge 3 of the MBZIRC 2020. Team Skyeye (Suarez Fernandez et al., 2020) used DJI Matrice 600 and DJI Matrice 210 v2 UAVs equipped with color and thermal cameras, GPS, and LiDAR. A map is prebuilt from LiDAR, IMU, and GPS data to allow online Monte Carlo localization and path planning with Lazy Theta*. Fires are detected via thresholding on thermal images. The fire location is estimated with an information filter from either projected LiDAR range measurements or the map. A water pump for extinguishing is mounted on a pan-tilt unit on the UGV while being fixed in place on the UAV.

Although our general approach is similar to team Skyeye, we rely more heavily upon relative navigation for aiming at the target after initial detection and less on the quality of our map and localization. In comparison, the protruding nozzle on our UAV allows for a safer distance from fire and walls while aiming. Furthermore, where distance measurements are unavailable, we use the known heat source size for target localization. We detect the holes of outdoor facade fires in LiDAR measurements and fuse these with thermal detections, which allows us to spray precisely on target, where the surrounding flames would offset thermal-only aiming.

3 Common Hardware and Software

Instead of starting development from scratch, we based our UAVs on our systems built for the MBZIRC 2017 competition (Beul et al., 2019). We equip commercially available DJI platforms with specific sensors, actuators, and computing power for their particular task. Depending on the required payload, we chose a DJI product of suitable size and lift capability, such as the DJI Matrice 100, DJI Matrice 210 v2, and DJI Matrice 600. Locking into a particular vendor allowed us to share design resources and necessary equipment such as batteries, chargers, and remote controls. Furthermore, our trained safety pilots can switch between the different robots without problems, significantly increasing flexibility during testing and in the actual competition. For detailed descriptions of the hardware platforms for each challenge, we refer to Sections 4.1, 5, 6.1, and 7.1.



(a) `rosmon` terminal interface

(b) `rosmon` GUI with process overview

(c) System monitor GUI with high-level checks

Figure 2: Process and system monitoring interfaces. In all views, the red color indicates process or component failures.

Our software components are also an evolution of our MBZIRC 2017 entry. We built our software on top of the ROS middleware (Quigley et al., 2009), the de-facto standard framework for robotic applications. In addition to the wealth of available components in the ROS ecosystem, we developed several essential tools and libraries that allowed us to quickly implement robust software for the competition.

3.1 Robust Network Communication

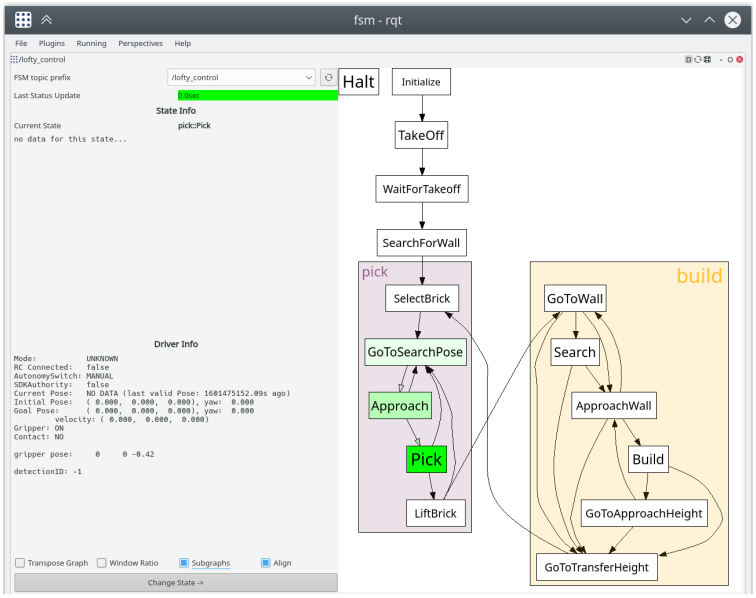
A key component in UAV fleets is communication. Robots may communicate with each other about object detections, goal and task assignments, their current position, and a multitude of other data. Furthermore, since the robots operate under human supervision, they will send monitoring data to the supervisors.

ROS offers built-in network transparency, but it is ill-suited to unreliable wireless networks due to its reliance on TCP and handshaking protocols. Our group has developed the `nimbro_network`[1] package as a robust replacement for ROS network transparency. Our robots are separate ROS systems (a so-called multimaster approach), which allows them to operate even without a network connection. The `nimbro_network` stack creates statically-configured connections between the robots, which forward the chosen ROS topic over the network connection. It uses forward error correction (FEC) rather than resending to deal with packet loss, which reduces latency. All robots stream monitoring data to a central operator station on the ground. The mission specialists then connect to this operator station through Ethernet using standard ROS network transparency. For a detailed list detailing the advantages and disadvantages of `nimbro_network`, see the referenced website.

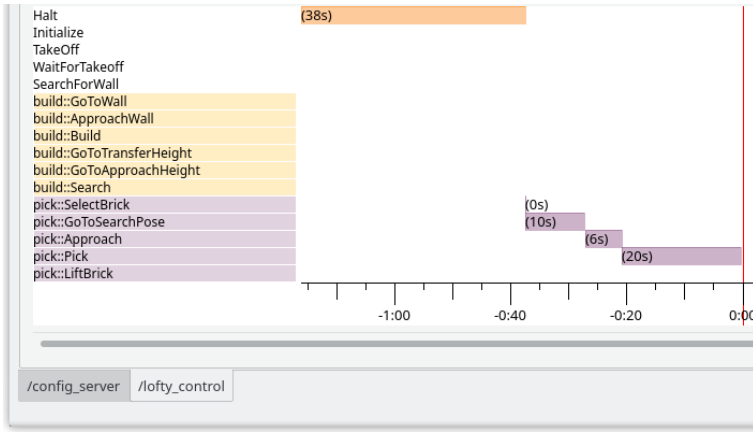
3.2 Runtime Control and Supervision

Running a complex robot software stack, especially from remote, is a complicated task. We use a combination of a local, onboard launch system and remote high-level system monitor. Our launch system, `rosmon`[2], is a robust replacement for the standard `roslaunch` tool, which offers several useful features for remote robot operation: A ROS interface for process status and process control, terminal and remote GUIs for process control (see Fig. 2), automatic log, and process core dump collection in case of failures.

[1] `https://github.com/AIS-Bonn/nimbro_network`
[2] `https://github.com/xqms/rosmon/`



(a) FSM GUI with state-specific information such as robot and brick poses during picking.



(b) Time-line plot with state activations.

Figure 3: `nimbro_fsm2` GUI showing the state machine of our UAV Lofty (see Sec. 6). The currently active state "Pick" is shown in dark green, while the previous two states are shown in lighter shades of green.

While `rosmon` allows the supervisors to ascertain whether all required processes are still running, it is impossible to see if the components are running *correctly*. Does the camera driver produce images? Is the GPS signal stable? For such questions, we built a flexible monitoring system consisting of a local ROS node that runs predefined checks every second and a remote GUI component that shows the results to the user in an intuitive way (see Fig. 2c).

3.3 Finite State Machines

Finite state machines are a standard tool for the definition and control of robot behavior. For relatively constrained tasks such as the ones defined by MBZIRC, they allow fast construction of behaviors. Instead of working with standard ROS tools such as `SMACH`, a Python-based FSM framework, we decided to develop our own `nimbro_fsm2` library with a focus on compile-time verification. Since testing time on the real robots is limited and simulation can only provide a rough approximation of the real systems, the robot will likely

Table 1: Trajectory parameters at MBZIRC 2020.

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

encounter untested situations during a competition run. We trade the ease-of-use of dynamically typed languages and standard frameworks against compile-time guarantees to guard against unexpected failures during runtime.

The `nimbro_fsm2` library supports FSM definition in C++. The entire state graph (see Fig. 3) is discovered and verified at compile time using C++ metaprogramming features. `nimbro_fsm2` is open-source[3].

Our library also automatically publishes monitoring data so that a human supervisor can see the current status. An accompanying GUI displays this data and can trigger manual state transitions, which is highly useful during testing.

Metaprogramming techniques for FSMs, including compile-time optimizations, have been investigated by, e.g., Juhász et al. (2008). Unfortunately, the authors do *not* provide an implementation, and in contrast to our approach, no user interface is provided.

The combination of our tools for multi-robot communication, transparent state execution, and easy supervision is groundbreaking. Although we developed these tools for the tasks posed in robotic competitions, their potential use reaches far beyond these demonstrations. With robotic systems becoming more complex, featuring more degrees of freedom, and robots communicating with each other, keeping the overview over these systems becomes increasingly difficult. Our monitoring and communication shaping approaches help to not get lost in complexity and to scale to systems of systems with a large number of robots.

3.4 Trajectory Generation and Control

Our UAVs need to be fast and agile in order to score competition points. At the same time, they need to precisely arrive at target positions. Our trajectory generation and control method[4] is based on the method that already reliably worked during MBZIRC 2017 (Beul et al., 2017; Nieuwenhuisen et al., 2017). For the MBZIRC 2020, we only changed parameters to adapt the behavior to the specific challenges. The parameters for the horizontal x- and y-Axis and the vertical z-Axis are presented in Tab. 1. Our method is described in detail by Beul and Behnke (2016) with the extensions from Beul and Behnke (2017). For brevity, in this section, we cover only the most essential 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 $(p, v, a)^\intercal_{UAV}$ 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 UAV flies on a relatively straight path.

We directly use this trajectory generation method as a model predictive controller (MPC), running in a closed loop at 50 Hz on all UAVs. As stated above, our MPC generates jerk commands, but the low-level flight controllers expect pitch resp. roll commands. We therefore assume pitch and roll to directly relate to $\theta = \text{atan2}(a_x, g)$ and $\phi = \text{atan2}(a_y, g)$. Thus, we send smooth pitch θ and roll ϕ commands for horizontal movement and smooth climb rates v_z instead.

[3] `https://github.com/AIS-Bonn/nimbro_fsm2`
[4] `https://github.com/AIS-Bonn/TopiCo`

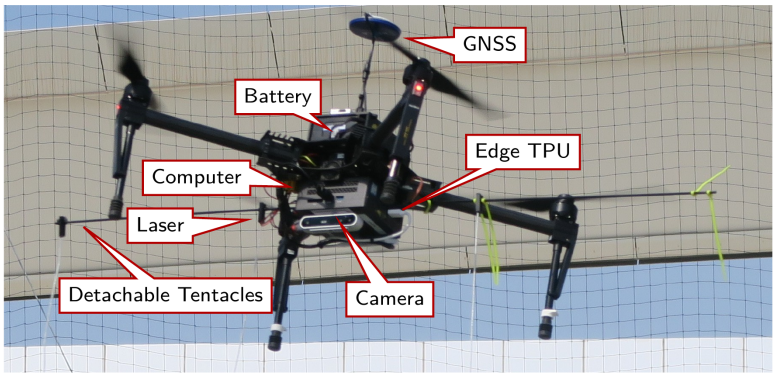


Figure 4: Design of our UAV "Jelly" equipped with four detachable spiked tentacles, an Intel® RealSense™ D415 camera, a Google Edge TPU, a laser height sensor, and a lightweight but powerful onboard computer.

Although an arbitrary number of axes can be controlled by the above-mentioned method, we do *not* consider the yaw-axis to be synchronized with the x, y, and z-axis. For simplicity, we use proportional control for the yaw-axis.

Like our tools for robot communication and supervision, also our universal MPC contributed to making the effort of operating multiple very diverse UAVs tractable. Due to its inherent dynamic model independence, we used it on all UAVs without having to keep track of individual parameter sets like PID gains. It also reduced the control system complexity since we didn't have to model challenge-specific dynamics like UAVs' changing weight when dropping a brick.

We see our MPC as a prime example of how the scientific contribution of a novel method leads to added value in the real world. By releasing our tools as open-source to the public, we transfer this value to the robotics community and, therefore, advance the entire field which adds value to our *own* systems in turn.

4 Balloon Hunt

The first task of the MBZIRC 2020 Challenge 1 required teams to find and pop balloons in an outdoor arena of size 90×40 m. Five green balloons with approximately 60 cm diameter were randomly placed inside on top of 2.5 m long poles. Although the total challenge time was set to 15 min, the task had to be completed much faster and autonomously to receive a high score.

The task mainly promoted the development of advanced (visual) perception methods, multi-UAV planning, and aerial manipulation capabilities that are beneficial for real-world applications like e.g., crop dusting.

Up to three UAVs could be used to complete the challenge, but we found it sufficient to use only one UAV. While global navigation satellite system (GNSS) positioning was available, the use of differential GNSS was penalized. For this task, we designed the UAV "Jelly" including fast TPU-based perception, robust filtering of sensor data, and fast trajectory generation and control (Beul et al., 2020).

4.1 Hardware

Jelly, shown in Fig. 4, is based on the DJI Matrice 100 platform. We equipped it 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 RAM. Balloons are perceived by an Intel® RealSense™ D415 depth camera with the assistance of a Google Edge TPU USB accelerator. For precise height estimation, Jelly uses a downward-facing LIDAR-Lite v3.

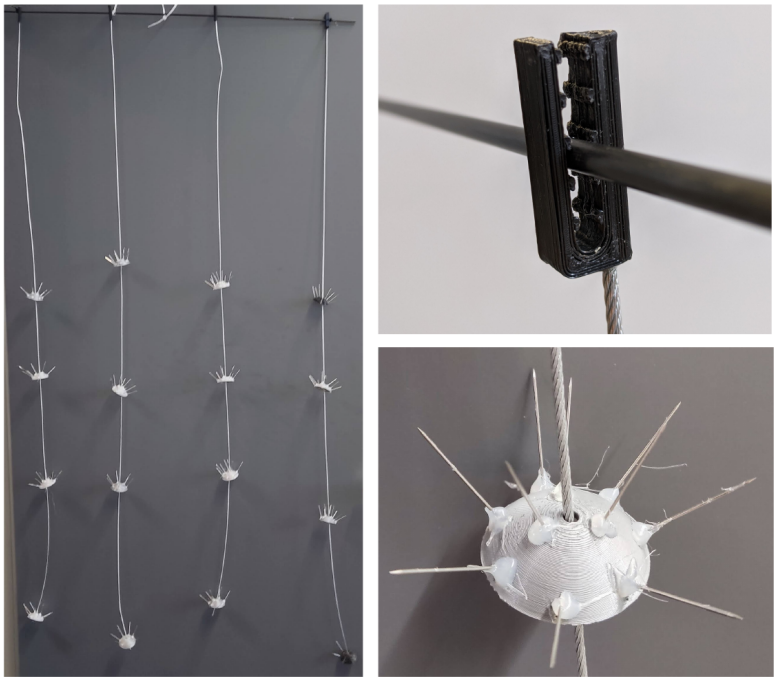


Figure 5: Detachable spiked tentacles of our UAV "Jelly".

Balloons are punctured with four detachable spiked 1.4 m long tentacles mounted on a horizontal bar and spaced out at a distance of 30 cm, as shown in Fig. 5. When a force of more than 2 N is applied to a tentacle (e.g., by entangling with the poles), it detaches, preventing Jelly from crashing. On each cable, we mounted four needle-spiked hemispheres with a 15 cm distance. Using flexible popping hardware, Jelly complied with the size restrictions of $1.2 \times 1.2 \times 0.5$ m, still offering a forgiving popping system that does *not* require centimeter-level precision.

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 UAV. 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 UAV. Fig. 6 gives an overview of the information flow in our system.

4.2 Mission Control State Machine

Jelly's behavior is controlled by a state machine that serves as a generator for waypoints and headings for the subsequent control layers. It also ensures that Jelly does *not* exceed arena limits and stays within a defined altitude corridor so that it always stays above the balloon mounting poles and below the 5 m minimum altitude of the other subchallenge's UAVs. Fig. 7 shows a flowchart of our state machine, which consists of two alternating parts: Search and Pop. In search mode, Jelly flies a repeating creeping-line pattern along the field's long axis, thereby scanning the entire arena. In Pop mode, Jelly flies a trajectory that drags the tentacles through detected balloons.

Jelly's velocity in search mode is tuned to 5.0 m/s, and the altitude is 4.0 m so that balloons can be reliably searched from a safe height. Our balloon detector produces reliable position estimates at ranges over 30 m. During the Grand Challenge, the search pattern comprised two search lanes, spaced at 10 m from the arena limits, which proved sufficient. As depicted in Fig. 6, all balloon detections are filtered as described in Sec. 4.4

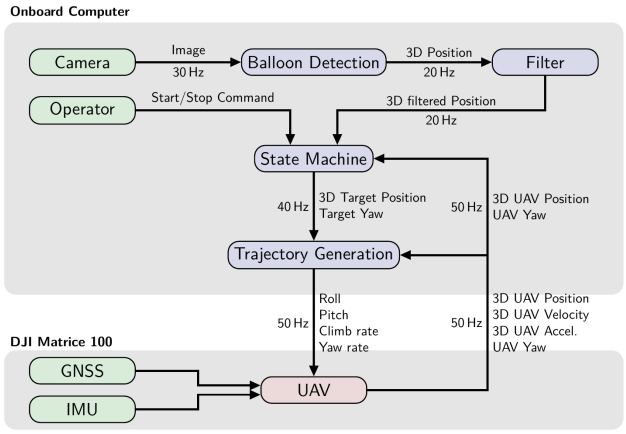


Figure 6: Structure of our software stack. Green boxes represent external inputs like sensors, blue boxes represent software modules, and the red box indicates the UAV flight control. Position, velocity, acceleration, and yaw are allocentric.

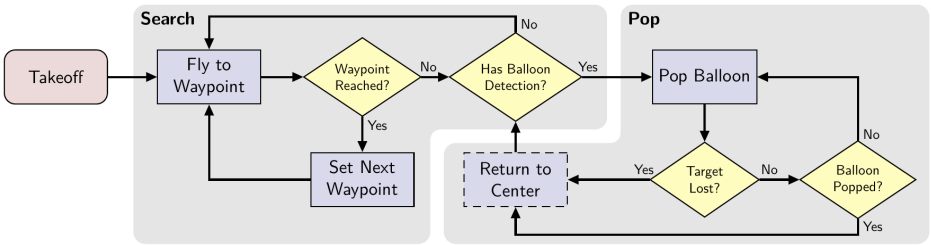


Figure 7: Flowchart of our state machine for balloon popping.

before being processed by the state machine. The filter provides a list of verified balloon positions within the arena limits, including those currently out of view. Once the state machine receives at least one detection, it proceeds to approach the closest target. Jelly does *not* consider observation waypoints for the detection but only searches for balloons while flying from one waypoint to the next. Jelly does *not* stop at the waypoints since when it reaches the current waypoint (within a certain radius), the transition is triggered, and the next waypoint is set active.

In Pop mode, we compute a straight-line trajectory, such that the center of the tentacles passes through the balloon center at a non-zero velocity. The tentacles' upward-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 Jelly passes over the estimated center of the balloon instance within a 0.5 m radius. Should the balloon still be 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 canceled. As an addition, in the Grand Challenge, after each attempt to pop a balloon, Jelly returns to the center of the field to prevent flying into the scaffolding protruding the arena. This method of handling the field's non-convexity is simple, but it introduces additional flying time compared to real obstacle avoidance. On the other hand, it is easy to implement and reliable. After returning to the center, Jelly targets the subsequent closest balloon provided by the filter or resumes with the search pattern if there are no viable balloon hypotheses.

4.3 Balloon Perception

Our approach for detecting balloons in images is based on deep learning methods and split into an inference and a postprocessing step. During the inference step, we employ a neural network for semantic segmentation. 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. 10. The balloon detection itself is carried out in the postprocessing step, which provides information like the number of balloons, their confidence values, and their positions in camera coordinates.

Balloon Outline Segmentation Network The neural network structure is based on the first three blocks of ResNet-18 (He et al., 2016). See Fig. 9 for a visualization of our network. The network outputs a binary segmentation consisting of background and balloon outline classes. For training, the network is initialized from ResNet-18 pre-trained with ImageNet. We use a dataset consisting of 10 k synthetic and 300 real images. The synthetic images were generated by a lightweight physically-based renderer[5] (Rosu and Behnke, 2020). A 60 cm diameter sphere is randomly placed in different HDR environments, as shown in Fig. 8. The images also include spherical shapes that are not green and do not correspond to a balloon to reduce the number of false positives. The real images were recorded with the same Intel® RealSense™ D415 camera, which we used during the competition. To enhance the network's robustness even further, we added noise to the synthetic data as described by Carlson et al. (2019). We trained the network with an image size of 960×540 px.

Postprocessing The postprocessing step aims to detect balloons in the binary segmentation output, as shown in Fig. 10. The binary segmentation provides contours of the balloons, which can still be noisy or could be false positives on the background. The valid balloon detections are extracted from the segmentation output based on a connected component analysis. In the first step, connected components are extracted, and valid components are filtered by a minimum number of pixels. Then, for each valid component, a circle is fitted based on N points sampled equally distributed from the connected component. Detections are filtered by a size threshold and a threshold on the residuals of the fitted circle, which give a measure for the quality of the estimated circle. A 3D position estimate can be calculated based on the detected balloon radius. The resulting 3D balloon center points are then further processed by an allocentric filter (cf. Sec. 4.4).

[5] `https://github.com/RaduAlexandru/easy_pbr`

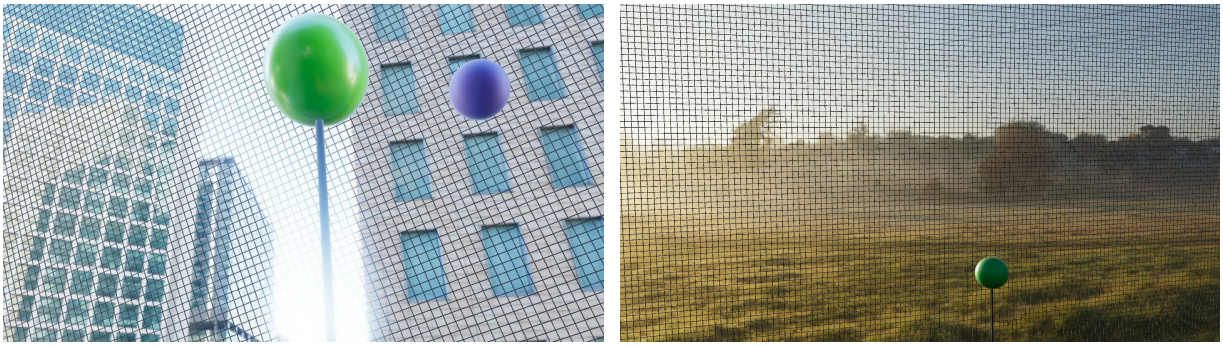


Figure 8: Synthetic training frames for balloon detection.

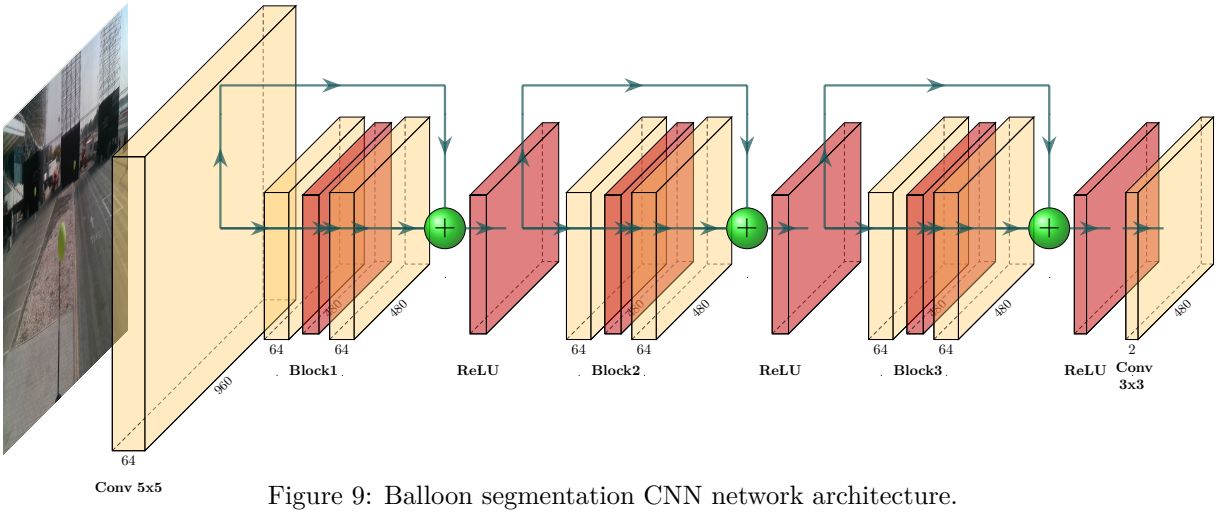
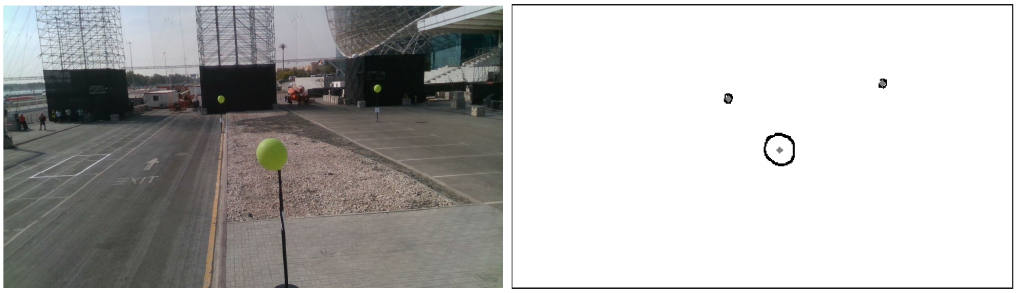


Figure 9: Balloon segmentation CNN network architecture.



(a) Input image. (b) Balloon detections.

Figure 10: Balloon perception. The detected balloon outlines are drawn with black lines, and the centers are marked with gray points. All visible balloons are detected correctly, even at large distances.

The entire pipeline is very time efficient and runs with an average processing time of 45 ms per frame on the onboard CPU with TPU acceleration. The quantization of the network for processing on the Edge TPU did *not* lead to a decrease in prediction quality. Finetuning during the competition resulted in a significant decrease in background noise and enhanced the network's balloon outline detection. Consequently, parameters like residual threshold and minimum connected component size in postprocessing were adapted so that we archived stable balloon detections even at large distances of up to 50 m.

4.4 Allocentric Balloon Filter

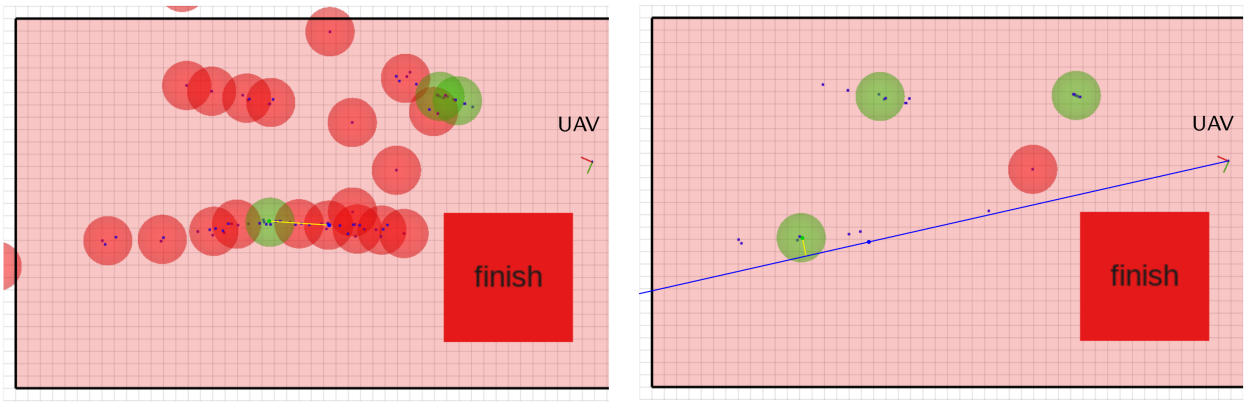
For each image frame, the balloon perception (Sec. 4.3) outputs a list of current balloon detections $d_1^i, \ldots, d_n^i$, described as egocentric 3D positions in camera coordinates. These are processed by a filter to reject outliers and to aggregate them into a list of hypotheses $\mathcal{H}$ of possible balloon positions. Each hypothesis $\mathcal{H}_i \in \mathcal{H}$ consists of

- a history $\mathcal{D}_i := (d_1^i, \ldots, d_8^i)$ 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 UAV position and forwarded to the state machine.

We transform the egocentric detections of the balloon detector into allocentric field coordinates. Since the height of the balloons was predefined to be at 2.5 m, all detections outside a height corridor from 1.5 to 5.0 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 $i^* = \arg\min_i \{dist(d, P_i)\}$. If the distance is smaller than a threshold of 2.0 m, we assign d to $\mathcal{H}_{i^*}$. Otherwise, we create a new hypothesis. Finally, hypotheses are merged when their estimated balloon positions come closer than 2.0 m.

The choice of the distance measure $dist(\cdot)$ 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 $2.5\,\text{m} + \frac{0.6\,\text{m}}{2} = 2.8\,m$, $dist(\cdot)$ 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. 11a). Instead, we cast a ray τ in the detection direction and define $dist(\cdot)$ as the distance between the estimated balloon position and τ. The latter distance metric better reflects our balloon detector's characteristics since it has a high accuracy in the image plane but is relatively inaccurate in the depth dimension. This change results in more accurate hypotheses assignments, as shown in Fig. 11b.



(a) Euclidean distance on ground plane. Exemplarily, the Euclidean distance of a detection is shown in yellow. It exceeds the threshold and, thus, a new hypothesis is created.

(b) Distance to detection ray metric. The distance (yellow) to the detection ray (blue) is smaller and does not exceed the threshold. Thus, no new hypothesis is created.

Figure 11: Evaluation of different distance metrics. Detections are shown as blue dots. Hypotheses are shown as spheres, colored green if at least eight detections are assigned to them and red otherwise. The ray metric (b) better reflects the characteristics of our balloon detection pipeline.

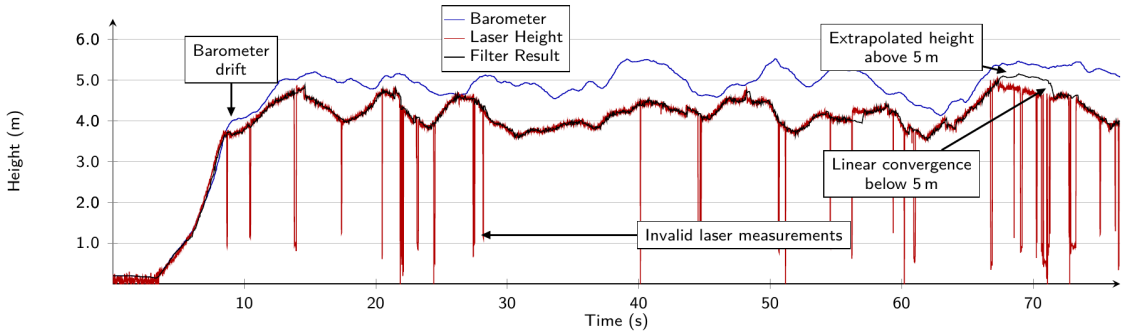
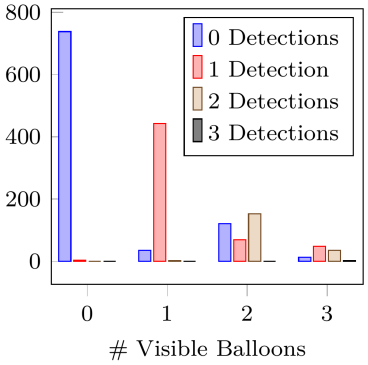


Figure 12: Laser height estimation during Run 2. The estimated height (black) is either based on barometer data (blue) or laser measurements (red).

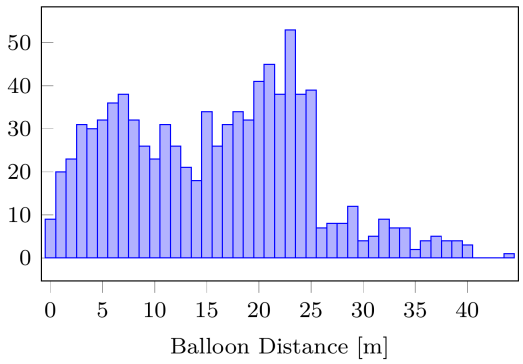
Once Jelly reaches a position above an estimated balloon, we assume the balloon to be popped and remove the corresponding hypothesis. If popping was *not* successful, the balloon is detected again later, and thus a new hypothesis for this balloon will be added (cf. Sec. 4.2).

4.5 Laser Height Filter

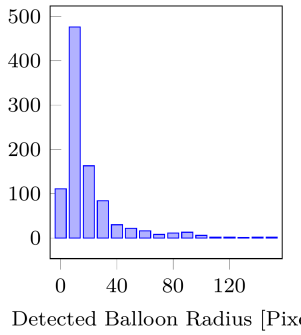
Precise height estimation can make the critical difference between popping a balloon, missing a balloon, or hitting a pole. We therefore use measurements of a downward-facing LIDAR-Lite v3 as the primary height source. The laser height filter determines whether the laser measurements are valid (i.e., within expected boundaries) 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 m/s. Fig. 12 depicts height measurements and the filtered height estimates during Run 2. During takeoff, height estimation is based on barometer data since laser measurements are unreliable for too close distances. Once Jelly reaches an estimated height of 1.0 m, height estimation is based on filtered laser data. Above 5.0 m, the laser becomes unreliable in the bright outdoor conditions, and the height estimate is



(a) Number of detections compared to number of visible balloons per frame.



(b) Number of detections compared to distance to closest ground truth balloon position.



(c) Number of detections compared to pixel radius of detection.

Figure 13: Histograms for the number of balloon detections.

Table 2: Assignments of all balloon detections to the different balloon hypothesis during Run 3.

	Balloon 1	Balloon 2	Balloon 3	Balloon 4	Balloon 5	No Assignment
#Assignments	58	229	117	252	248	45

extrapolated using the change in barometric height measurements.

4.6 Evaluation

We operated Jelly for balloon popping in three competition runs during MBZIRC 2020. A video showcasing our Grand Challenge Run can be found on our website[6].

To evaluate the performance of our balloon perception (cf. Sec. 4.3), we manually check the balloon detections for the Grand Challenge Run, during which a total of 1662 image frames have been processed by our vision pipeline. In total, 1460 balloons occur in these images, of which 949, i.e., 64 %, are correctly detected, while there are only five false detections. Fig. 13a shows histograms of the number of balloon detections per frame. Frames where no balloon is visible are correctly classified in 99.6 % of all cases. If a single balloon is visible, we achieve an accuracy of 92.3 %. However, frames with more than one balloon often contain more distant balloons, which are more difficult to detect. Thus, detections are missed more frequently in these cases.

To analyze for which distances our vision pipeline works reliably, we plot the number of detections against the distance to the corresponding balloon (see Fig. 13b). As ground truth, we use the last filtered position estimate immediately before the balloon is punctured. We reliably detect balloons up to a distance of 24 m, but can even detect balloons at a distance of up to 44.5 m. The corresponding pixel sizes are shown in Fig. 13c.

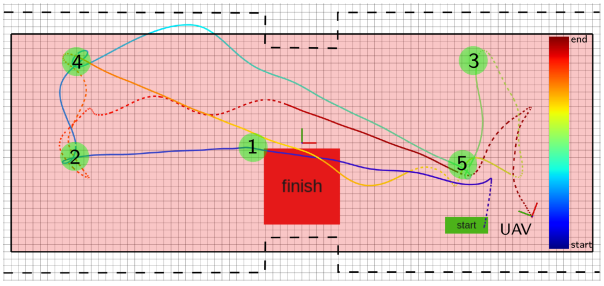
The number of detections at distances of 25–40 m is still sufficient to generate valid hypothesis with our allocentric detection filter (cf. Sec. 4.4). Thus, balloon hypotheses were added to the world model shortly after takeoff during all runs. During the second run, all five balloons were known to the filter only 15 s after takeoff. 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; Jelly then flies on a search pattern for a short time only until it detects the remaining targets. During the Grand Challenge Run, 95.3 % of all detections could be successfully assigned to one of the five balloons. The corresponding numbers are reported in Tab. 2.

[6] `https://www.ais.uni-bonn.de/videos/fr_2021_mbzirc`

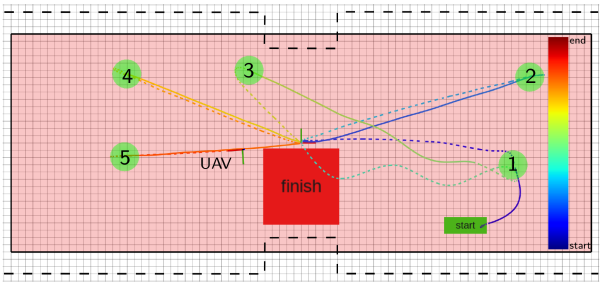
Table 3: Time and number of tries needed to puncture the individual balloons.

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

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



(a) Run 2: The UAV 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.



(b) Run 3: The UAV 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.

Figure 14: 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.

During the three different competition runs, we successfully punctured 15 balloons using only 18 tries. The times at which the respective balloons were punctured and the corresponding number of attempts are given in Tab. 3. In the first run, two balloons were popped after 30 s. Then a reset occurred for 8 min, as Jelly 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 min 28 s, but only 1 min 28 s flight time. In the second run, the first two balloons were punctured immediately. Then, however, two balloons were missed—the puncturing did not work due to a suboptimal flight pattern (see Fig. 14a). By repeating the search pattern and re-approaching the missed targets, in this run, all balloons were punctured after a total duration of 1 min 40 s. In the final run during the Grand Challenge, all balloons were punctured in the first attempt. The time between two consecutive balloons was very similar (12–15 s). Between Balloon 2 and 3, Jelly flew a search pattern to discover the remaining ones, explaining the longer time interval. The challenge was completed after a total time of 1 min 21 s.

During the first and second run, Jelly 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 Jelly flying dangerously close to the arena borders. It could even have led to a crash during Run 2 if the controller had chosen to pierce Balloons 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 arena's non-convex shape (see Fig. 14a).

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. 14b) 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. Moreover, the star-shaped flight pattern avoids trajectories close to the arena borders

Figure 15: Image sequence of popping the first balloon in the Grand Challenge. Jelly (marked with the red circle) 1) takes off to 4.0 m. 2) After detecting the first balloon, the robot targets a position 2.0 m behind and 0.7 m above balloon's center. 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 s.

and leads to safe flight paths despite the non-convex arena outline without any additional obstacle avoidance system (cf. Sec. 4.2).

See Fig. 15 for an exemplary balloon popping sequence. Overall, we placed 5th in Challenge 1, including the Ball Interception Challenge, and 2nd in the Grand Challenge, including five other sub-challenges.

5 Moving Target Interception

The second task of the MBZIRC 2020 Challenge 1 required a robot to intercept a moving target UAV in an outdoor arena of size 90×40 m. A yellow ball with approximately 13 cm diameter was attached below the target UAV, which moved with up to 10 m/s on a 3D figure-eight trajectory in the arena. The teams had to track the target, detach the ball without damaging the target UAV, and finally deliver it to a drop-off zone.

The task mainly promoted the development of advanced (visual) perception methods, fast and precise UAV control, and aerial manipulation capabilities that are beneficial for real-world applications like e.g., drone defense.

Our plan was to command our UAV to wait on a corner point of the figure-eight with the camera directed towards the field center and search for the ball. After the target UAV passes from behind and the ball is detected, our UAV pitches to the maximum allowed magnitude of 35° to gain speed on the figure-eight's diagonal to intercept the ball.

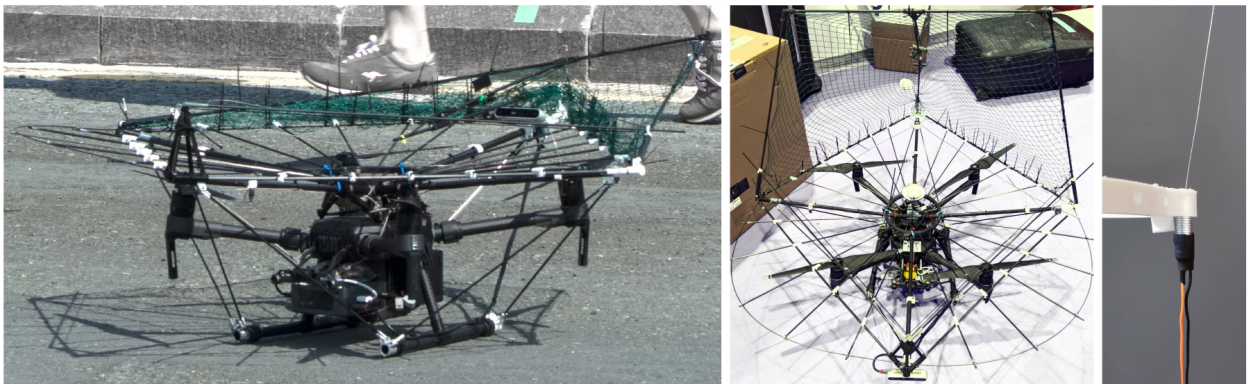


Figure 16: Design of our UAV "Chaser" equipped with a foldable net to capture the target, a fast onboard computer, an Intel® RealSense™ D415 camera and a Google Edge TPU.

5.1 Hardware

For this task, we designed our integrated UAV "Chaser" shown in Fig. 1b, which is based on the DJI Matrice 210 v2 platform. It is equipped with a compact, but fast onboard PC (Intel® NUC), an Intel® RealSense™ D415 depth camera for visual perception, and a Google Edge TPU USB accelerator for visual inference. The target ball is detached and caught by a foldable net, which, in a folded state, respects the required maximum takeoff size and, after takeoff, automatically extends by a spring release to offer a large volume for catching the target ball. The overall mechanism is shown in Fig. 16 and attaches on top of our UAV. Once the ball is caught by the net, it will drop onto the funnel-like contraption with radial struts and roll towards the center where multiple omnidirectional switches were installed. The switch consists of spring steel wire within a metal tube. Upon contact, the wire pushes against the tube and closes the circuit and an Arduino Nano micro controller will signal the onboard computer. To ensure safe interaction with the ball's attachment cable, the struts connect outwards to a round prop-guard steering the cable away from the propeller. Additional connections between the contraption and the landing gear improves stability and prevents oscillation of the flexible construction. The outer net struts can rotate backwards and together with the hinge in the middle of the central strut; the net becomes foldable. A spring at the central strut tensions and unfolds these struts. We prevent unfolding with a small chord from the strut to below the landing gear. During takeoff, the chord releases automatically and the spring unfolds and tensions the net. Zip ties attach the net to its frame while zip tie pairs at the bottom frame pointed upwards prevent the net from being dragged into the rotors before unfolding. Initially, we mounted the D415 at the rear-end on the funnel below the net, but moved it to the front due to USB interfering with the GNSS and the contraption being partially visible in the camera image. Since onboard GNSS remained unusable, we added in the middle a separate DJI N3 flight controller with protruding antenna and extra shielding below.

5.2 Target Detection

We employ a deep learning-based vision pipeline for the perception of the moving target. RGB images of the RealSense™ camera are processed by a detector based on the lightweight SSD architecture (Liu et al., 2016b) using an efficient MobileNetV2 (Sandler et al., 2018) backbone and provides bounding box detections for the yellow ball and the drone (see Fig. 18). The network input resolution is set to 848×480 px, processing cropped regions of the camera images of 1280×720 px resolution. The region of interest to pass to the detector is determined based on the currently tracked detection from the previous frame. When no hypothesis is known, the entire image is analyzed in four separate tiles.

The detector was trained before the competition on synthetic images of the target UAV and the ball, and fine-tuned with a few background images from the real arena on site. To prevent the network from learning to only detect dark patches in the sky, like cranes, we added negative training examples of objects like tea

Figure 17: Synthetic training images for target detection of drone and yellow ball.

pots, submarines and lamps with an attached yellow ball. In total, we used approximately 20 k synthetic and 1000 real images. Some examples are depicted in Fig. 17.

The vision model runs in 8-bit quantized mode on the Edge TPU USB accelerator, achieving the full camera frame rate of 30 Hz, thus sufficient for fast and reactive flight control based on the detections. The distance towards the detected UAV and ball is estimated by fusing the depth reading from the RealSense™ RGB-D camera with the distance estimated from the known diameter of the target ball and the camera parameters.

We want to stress that although we use our detection pipelines for detecting objects relevant to the MBZIRC 2020, our practical method and the theoretical contribution it is based on are *not* limited to these demonstrations. In contrast to classic computer vision approaches that are often custom-tailored for detecting specific patterns like the landing pattern during MBZIRC 2017 (Beul et al., 2019), our pipeline makes no assumption about the specific object it detects. Here again, scientific contributions transfer to real-life applications that result in added value.

5.3 Allocentric Detection Filter

The vision model outputs lists of copter and ball detections for each image frame. As already discussed for the Balloon Hunt Challenge (cf. Sec. 4.4), the depth estimation of the visual perception model can be noisy. However, precise allocentric position information and additional velocity estimates are necessary to successfully intercept the target. Thus, we process our detections with an allocentric filter before forwarding them to the Mission Control module. Our filter module consists of two steps: First, the most probable ball and copter detections are selected from the detection list based on the previous estimates. Afterwards, an Extended Kalman Filter (EKF) is applied to the position information of the detections to generate velocity estimates.

When selecting the most probable ball and copter detections, we first remove outliers by only considering copter detections if they are within a distance of 5 m around the previous estimate. From those, we choose the one with highest confidence score as provided by the detector. Afterwards, we process the ball detections in a similar way but utilize the fact that the ball has to be close to the target copter. Thus, if a valid copter detection for the current frame was found, we consider all ball detections within a distance of 5 m around the current copter detection. Otherwise, we search for a ball detection close to the previous estimated ball position. If multiple valid detections exist, we select the one with highest confidence score.

Differentiating the yellow ball from the green balloons of the first Challenge 1 task (cf. Sec. 4) is very challenging for our vision pipeline. Thus, we additionally search for all ball detections that might actually correspond to balloons. We estimate the distance between camera and each ball detection using the size of the bounding box and the known balloon diameter. If the projection into allocentric 3D coordinates results in a feasible balloon position, i.e., if the height is within a corridor from 1.5 to 5.0 m, we probably misclassified a balloon as ball, and thus discard the detection.

The selected detections are processed by two independent instances of an EKF, one for ball and copter

Figure 18: Detections of the target UAV (green bounding box) and ball (red bounding box) in the RGB image, with detection probabilities. The yellow bounding box shows the detection of the tracked ball from the previous frame, used to determine the cropped region of the current frame passed to the detector.

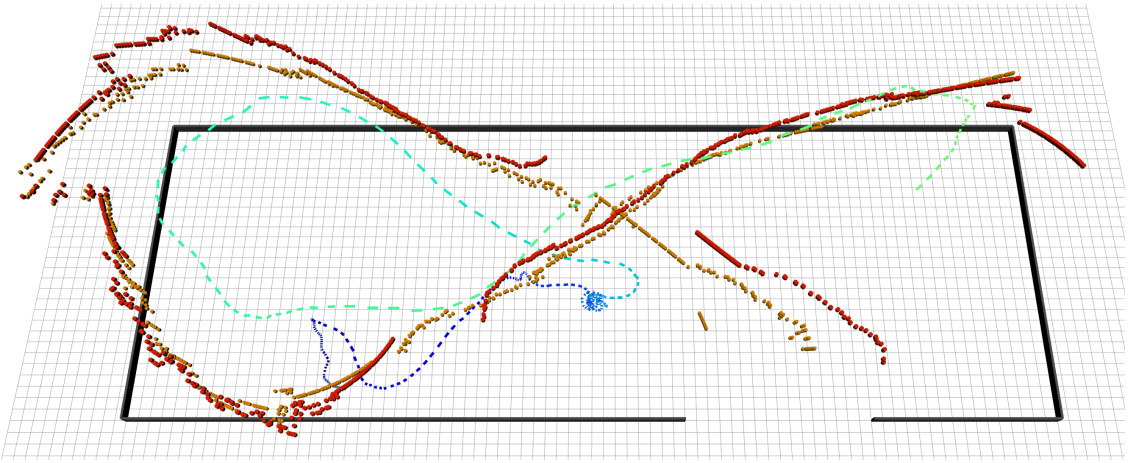


Figure 19: Tracked trajectory of target UAV (red) and ball (yellow), flight path of our UAV (dashed line), GNSS-based geofencing (gray). After takeoff, our UAV ascends to the search height and flies towards a waiting point (dark-blue dashed line). After the first detection of the target UAV and ball (bottom right), our UAV turns and follows the targets. The chase is abandoned at the top-right corner, as our UAV needs to brake before the turn. The three-dimensional figure-eight shape of the target's trajectory is clearly discernible.

detections, respectively. Here, we utilize the implementation from the robot_localization library (Moore and Stouch (2014)). The EKF output is forwarded to the Mission Control module. Additionally, the current estimates are used by our vision pipeline to crop a region of interest from the camera images, as described in Sec. 5.2.

5.4 Evaluation

To evaluate the performance of our perception pipeline, we manually check the detections for a portion of an autonomous flight during one challenge run. The analyzed sequence has a duration of 43 s and 1304 frames. It starts with the moment when Chaser reaches the initial search pose and ends when Chaser aborts the

Table 4: Evaluation of the vision pipeline for the moving target interception task.

	#Frames Visible	#True Detection	#False Detection	#No Detection	#Frames Invisible	#No Detection	#False Detection
Copter	566	456	15	95	738	707	31
Ball	557	450	15	92	747	718	29

Table 5: Evaluation of the filter accuracy for distinguishing ball and balloon detections.

True Label	# Classified as "Balloon"	# Classified as "No Balloon"
"Balloon"	43	8
"No Balloon"	8	490

first chase by breaking on the end of the figure eight's diagonal. If multiple detections for the same frame exist, we only consider the one with highest confidence score. The performance is similar for copter and ball detections (see Tab. 4). The perception pipeline works robustly and detects the moving target in about 80 % of all cases where it is visible. Only in about 3.5 % of all analyzed frames does the most confident detection not correspond to the target copter or ball. Some example images, where the moving target is detected at a considerable distance even in front of cluttered backgrounds, are shown in Fig. 18.

Distinguishing the yellow target ball from the green balloons of the other Challenge 1 task was challenging under the harsh sunlight. During the analyzed sequence, our vision pipeline reports 594 ball detections, of which 51 actually correspond to balloons. However, our allocentric filter (Sec. 5.3) is able to detect those wrong classifications in most of the cases, as shown in Tab. 5. For both, precision and recall the filter achieves scores of 84 %. Note, that in contrast to Tab. 4, we count all ball detections per frame and not only the most confident one.

Based on the filtered detections, the target's figure-eight trajectory could be tracked in 3D space, and flight trajectories were generated to follow the target UAV and accelerate below it to detach the target ball (see Fig. 19). Unfortunately, in all trials, the acceleration was insufficient to gain enough speed until Chaser had to break to prevent flying into the net on the end of the diagonal. In multiple tries, Chaser reliably achieved velocities exceeding the 10 m/s of the target UAV and also came close to it but never managed to intercept the ball before breaking. With 6.2 kg and $1.1 \times 1.1 \times 1.0$ m size and due to the wind resistance of the large netting construction on top, Chaser was simply not agile enough for the proposed task.

3 min before the end of our run, we decided to abort the autonomous attempts and salvage points by manually approaching the target. For this, we prepared a bare DJI Matrice 100 in advance and removed all unnecessary hardware to reduce weight and increase agility. Although the UAV featured no catching hardware, the intention was to at least detach the ball from the target UAV, which would yield at least partial points.

With the stripped-down UAV, our trained safety pilot was able to manually detach the ball with only 8 s flight time, securing a 5th place in Challenge 1 together with the balloon-hunting task.

6 Wall Construction

In Challenge 2, participants were required to build walls out of supplied bricks, both with a UGV and a team of up to three UAVs. The task setting was particularly interesting because it required complete autonomy, robustness under real-world outdoor conditions with harsh sunlight and wind, and independence from any external reference system besides the globally available GPS.

In the following, we will describe our UAV "Lofty". For more detailed information about Lofty, we refer to Lenz et al. (2020).

Since the initial rules specified a shared wall where UAVs and UGV could collaborate, we concentrated our efforts on the UGV design. In a late rule revision, UAV and UGV walls were separated, making it clear to us that UAV points had to be scored in order to win. Thus, our UAV design focused on a minimal solution that could achieve almost full points: We decided to ignore the orange bricks of 1.8 m length, intended to be carried by two UAVs. We designed our system to only support the red (0.3 m), green (0.6 m), and blue

(a) Full assembly (b) Magnetic gripper with passive compliance

Figure 20: Wall building UAV hardware design. The four telescopic rods are shown in the fully extended configuration.

(1.2 m) bricks.

6.1 Hardware Design

Because of the weight of the larger bricks and their size, we decided to use a large UAV, the DJI Matrice 600, for this task. The DJI Matrice 600 offers sufficient payload and battery life (roughly 20 min in our configuration).

A key component for aerial manipulation is the robotic gripper. UAVs pose unique constraints when compared with ground-based manipulation. The gripper has to be lightweight in order to fit inside the payload constraints. Furthermore, a certain flexibility and mechanical compliance is desired for two reasons: First, this allows a grasp to succeed even if the approach was not fully precise. Secondly, a rigid connection between the UAV and the ground can be perilous since UAVs usually tightly and dynamically control their attitude to hold position. One can easily imagine situations where the UAV has to drastically change attitude in response to wind gusts, and of course, hindrance by the gripper system should be limited. However, during the placement phase of the pick-and-place operation, we require exact control of the target object. Here—at least while the target object is still in the air—we want a rigid attachment to the UAV. To resolve these seemingly contradicting goals, we designed the gripper system to be rigid only while load is applied, i.e., the brick is hanging below Lofty.

Our gripper design (see Fig. 20) consists of four carbon fiber telescopic rods, which hold the magnetic gripper plate below Lofty. When the rods are fully extended, the gripper plate is in a fixed pose and can only move upwards. The more the gripper plate is pressed upwards (e.g., due to contact with a brick), the more it can move sideways and rotate due to the gained movement range in each rod. The gripper is equipped with a switch to detect successful grasping.

Since the standard foldable landing legs on the DJI Matrice 600 would interfere with the gripper, we replaced them with fixed landing legs (see Fig. 20).

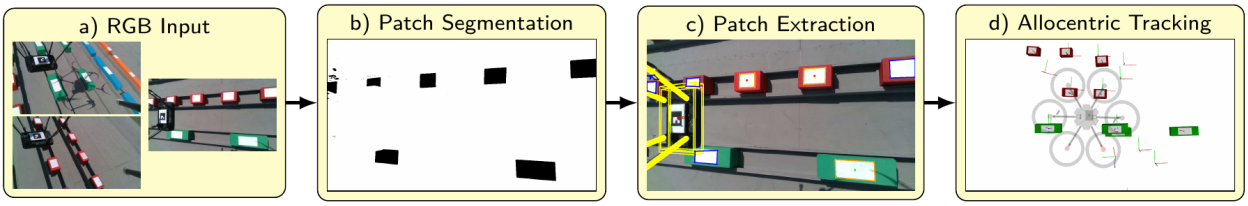


Figure 21: Camera-based UAV brick perception pipeline. a) Input frames from all cameras. b) White patch segmentation. c) Patch corner extraction & pose estimation. Patch contours in orange (verified) and blue (wrong shape). Brick type is indicated by a colored center point. The gripper is overlaid in yellow. d) Tracking of detections from all three cameras in GPS frame. Detections are shown as bricks, while tracked hypotheses are shown as coordinate axes.

6.2 Brick Perception

The competition task involves two perception challenges: finding and precisely localizing the bricks and localizing with respect to the target wall. Similar to the gripper system, the UAV places unique constraints on the perception system. Since the gripper is mounted directly beneath Lofty, any brick observation close to the gripper must be conducted from the side. The necessary off-center mounting of the sensors severely limits the sensor weight. We chose the Intel® RealSense™ D435 RGB-D camera as a primary sensor for its low weight and its capability to work in sunlight. To achieve good coverage of the terrain below Lofty and to be able to observe large parts of the wall during the placement process, we mounted three D435 sensors (see Fig. 20 and 21).

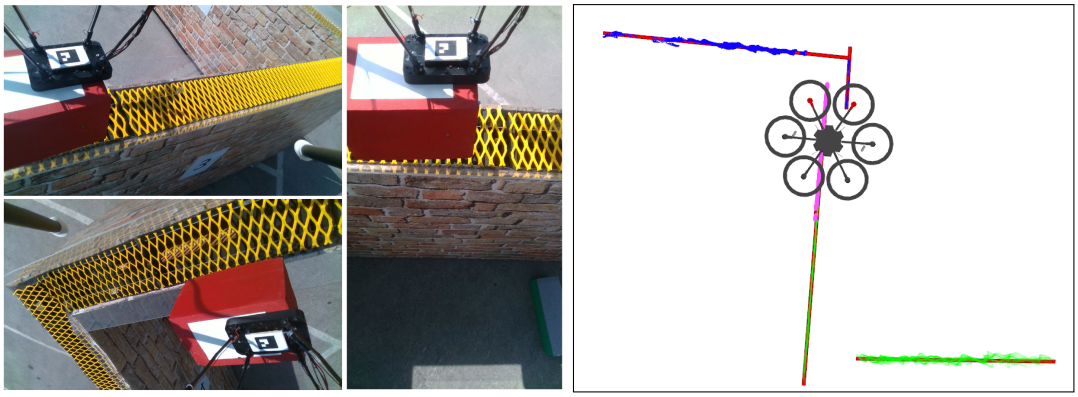
The gripper is visible in all camera images and would lead to confusion with bricks. For this reason, we mounted an ArUco marker (Romero-Ramirez et al., 2018) on it. The marker pose can be efficiently estimated in each of the three cameras and is low-pass filtered to obtain a robust estimate of the gripper pose below Lofty. We exclude pixels in the immediate vicinity of the detected gripper from further processing.

Since the white patches on the bricks are quite distinctive (see Fig. 21), we use them to detect the bricks and estimate their pose. In a first step, we convert the input image (resolution $1280 \times 720\,\text{px}$) to the HSV color space. To detect high-saturation pixels (the colored bricks) in the neighborhood, we downsample the input image to half resolution and run a box filter with kernel size $290 \times 290\,\text{px}$ to obtain a local saturation average $\bar{S}$ and local value average $\bar{V}$. A pixel p is classified as a *patch* if $S(p) < \bar{S}(p) - \lambda_S \wedge V(p) > \bar{V}(p) + \lambda_V$, or, in other words, the saturation is less than the local average and the value (brightness) is larger than the local average by user-specified thresholds. This simple segmentation method is modeled after the ones used for detecting chessboard patterns and leads to highly robust performance (see Fig. 21).

Contours with exactly four corners (after contour simplification) are processed further: We check that each corner has a *patch* pixel on the inside and a high-saturation pixel on the outside at a specified distance of $d = 4\,\text{px}$. The high-saturation pixels on the outside are independently classified into the four possible colors. If all agree, the brick is detected. Finally, a PnP solver is used to determine the 6D pose of the brick from the recovered 2D-3D correspondences. Here, we assume that the longer side in the 2D image corresponds to the longer brick side in 3D—an assumption, which is only violated at extreme viewing angles. To fuse the detections from all three cameras and track bricks over time, we apply a basic Multi-Hypothesis Tracking (MHT) method with one Kalman filter per hypothesis.

6.3 Brick Placement

Since not much was known about the target wall for brick placement prior to the competition apart from its shape, we decided to use only the wall geometry for detection and localization. Since there is no larger structure in the vicinity of the wall, we felt this would be sufficient. Of course, the bright yellow mesh grids on top of the wall (see Fig. 22) would have been a good feature as well, but these were not known to the



(a) RGB images from all three cameras.

(b) Top-down view of detected wall points per camera (green, blue, purple) and detected wall segments (red lines).

Figure 22: Wall localization.

participants beforehand.

Our wall perception module estimates the height above ground from the depth image of the downward-facing camera. 3D points measured from each camera are then filtered so that only points above 1.0 m and below 1.7 m remain. We then project the data to 2D, where we extract lines using RANSAC. Each line, if fit correctly, corresponds to a side view of one wall segment (see Fig. 22). The system is initialized with a user-specified initial wall pose relative to the starting pose, which serves as the search pose. The wall pose is updated any time two parallel line segments of valid length with 4 m distance are found. Under the assumption that the wall did not rotate 180° relative to the initialization, this is unambiguous. Carrying a brick, Lofty targets a specific pose on one of the wall segments to place the brick as indicated by the wall plan given by the challenge organizers. During close approach, the detected segment closest to the expected segment pose is identified, and the goal position is projected onto this segment.

6.4 High-level Control

The high-level control module is implemented in an FSM framework. It is supplied with the target wall pattern as defined by the organizers of the competition. The basic cycle of events is designed as follows:

1. Fly to the last known pile pose and fly a search pattern until the next brick requested by the pattern is found.
2. Grasp the brick and lift it.
3. Fly to the target position (relative to the last known wall pose) and look for a wall segment.
4. Approach the projected position on the wall segment and place the brick.

Similar to our MBZIRC 2017 approach (Beul et al., 2019), we utilize a “cone of descent” during grasping and placement, in which Lofty is allowed to descend towards the target pose. If it drifts outside of the cone, it has to stay at that height until the disturbance is countered. The cone angle is 10° with a hysteresis of 3° to prevent oscillations. The cone was shifted such that at the target height, it had a radius of 9 cm, which we determined as the maximum allowable deviation that would still allow successful magnetic grasping.

6.5 Evaluation

During the MBZIRC 2020, Lofty performed in six arena runs: three rehearsal runs, two Challenge 2 runs, and the final Grand Challenge run. A video showcasing the evaluation can be found on our website[7].

We used the rehearsal runs to get used to the conditions in Abu Dhabi and continuously improved our pick success rate (see Fig. 23). During our first Challenge 2 run, we only picked one red and one green brick due to difficulties with our magnetic gripper. Both bricks were dropped close to but not *on* the wall due to wall tracking problems. The wall tracking module had not been tested until this point due to short development time and lack of suitable testing opportunities at the competition.

After improving our gripper overnight, we managed to pick four red bricks and one green brick and placed two red bricks successfully during our second Challenge 2 run. The other bricks were sadly dropped right next to the wall due to another wall tracking problem. This run was scored as 1.33 points, which secured a second place in Challenge 2, next only to the Prague-UPenn-NYU team.

In the Grand Challenge, Lofty managed to pick a red brick but placed it a bit too high, and it fell off the wall. After a long pause to allow our Challenge 1 UAV to operate, it started again and picked up a green brick. Sadly, it falsely detected a W-shaped wall behind the Arena netting. Due to a rushed setup sequence, both wall search pose and the geofencing was not set up correctly and did not prevent the false detection nor Lofty from flying into the net. After a short, unsuccessful rescue attempt during a reset, we had to leave it there for the rest of the Grand Challenge.

Overall, Lofty executed 132 pick attempts in Abu Dhabi, of which 22 were successful, which gives a success rate of 16.7 %. Since a failed attempt took 12 s on average, this limited the number of attempts we had for placing bricks on the wall. The number of pick attempts increased throughout the competition (see Fig. 23) as the rest of the system became more robust. There are two peaks in the duration histogram for failed picks: One at roughly three seconds which corresponds to tracking failures during the initial approach, and a larger one around 10 s, which corresponds to misaligned picks or magnet failures.

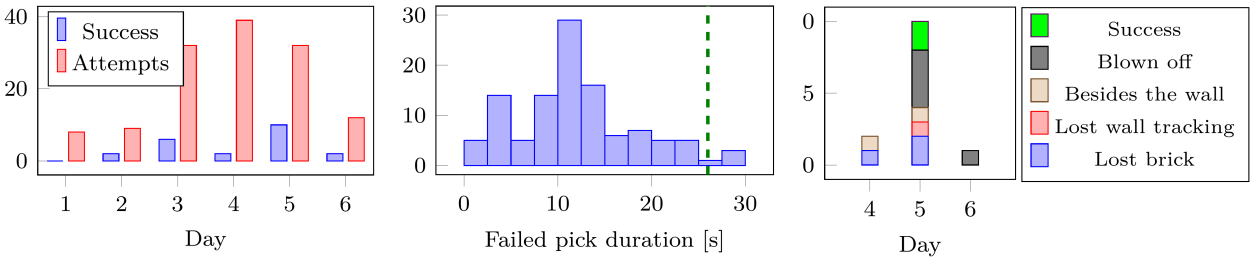
While we cannot provide a detailed accuracy analysis of the brick pose estimator due to missing ground truth, we can draw some conclusions regarding the perception module. In 57 % of the pick attempts, the gripper made contact with a brick. This establishes a lower bound on the detection performance. Out of these, 29 % resulted in a successful pick, which indicates sufficient precision for magnetic gripping. Of course, other failure causes such as insufficient magnet power or wind gusts will also have reduced this fraction.

We also show placement results in Fig. 23. In our best run, our second Challenge 2 run, Lofty attempted ten placements, out of which 2 succeeded. The most prevalent failure reason was slightly off-center placement, which resulted in the placed brick being blown off the wall by Lofty's rotor wash.

7 Fire-Fighting

Challenge 3 of the MBZIRC 2020 targeted an urban fire-fighting scenario for autonomous robots to foster and advance the state of the art in perception, navigation, and mobile manipulation. Participants had to detect, approach, and extinguish multiple simulated fires around and inside a building with up to 3 UAVs and one UGV. Each fire provided a 15 cm circular opening with a heated plate recessed 10 cm on the inside. Holes on the outside facade were surrounded by a propane fire ring, while indoor fires had a moving silk flame behind the heating element. For these tasks, we developed our UAV "Splasher", described in the following.

[7] `https://www.ais.uni-bonn.de/videos/fr_2021_mbzirc`



(a) Pick success rate over the duration of the competition.

(b) Histogram of failed pick durations. The average successful pick is shown in green.

(c) Placement results.

Figure 23: Challenge 2: Pick & place robustness.

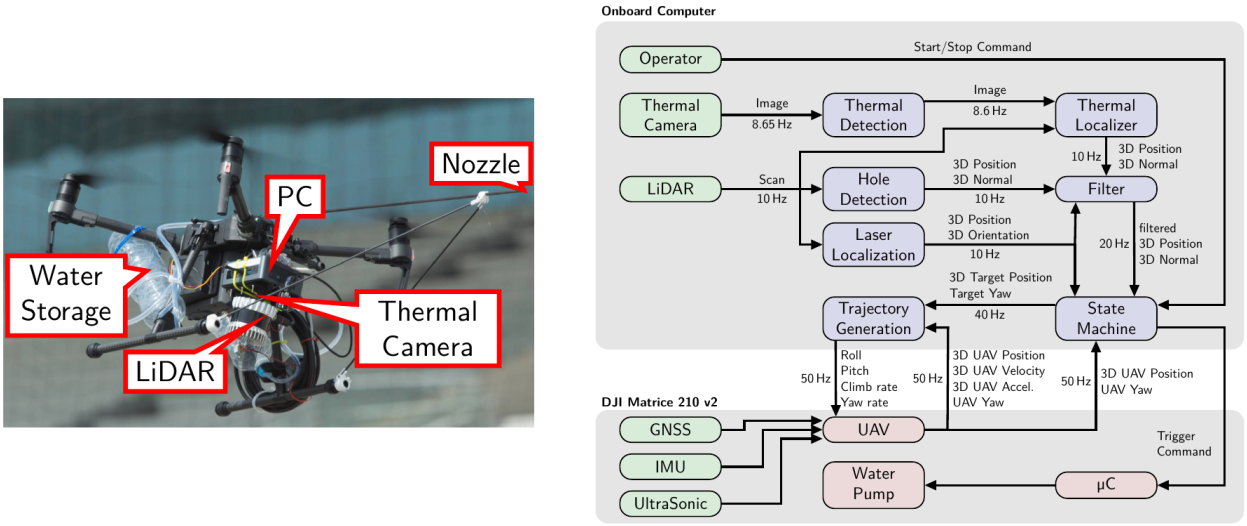


Figure 24: Hard- and software design of our fire-fighting UAV “Splasher”.

7.1 Hardware Design

Fig. 24 shows the setup of Splasher. It is based on a DJI Matrice 210 v2 and is equipped with an Intel® Bean Canyon NUC8i7BEH with a Core™ i7-8559U processor and 32 GB RAM. We combine an Ouster OS1-64 LiDAR and a FLIR Lepton 3.5 thermal camera with 160×120 px resolution for perception and localization of the fires. The DJI Matrice 210 v2 provides GNSS-based ego-motion estimates. We deactivated the obstacle avoidance of the DJI Matrice 210 v2 in the forward direction to get close enough to fires for extinguishing.

The water supply is stored in two downward-facing 1.5 l PET-bottles attached between the rear frame arms and the landing gear. The screw caps are connected via a flexible hose to a windscreen washer pump. We mounted a carbon tube on top of Splasher as an extended nozzle. It is supported by two additional carbon struts from the landing gear. The high location compensates for the height difference in the water jet parabola, allows to perceive the fire with the sensors below, and prevents the water from flowing out on its own during flight maneuvers. We chose a 10° downturn for the nozzle, LiDAR, and thermal camera to fly above the fire while maintaining observability.

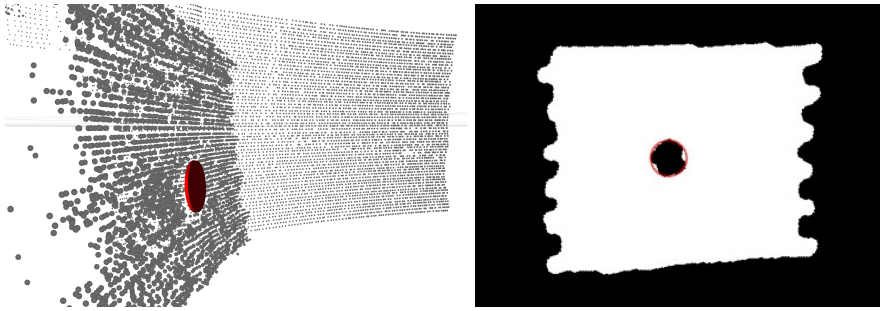


Figure 25: Detected hole (red disk) inside a point cloud and points projected into the virtual camera after morphological transformation and the detected circle.

7.2 Laser Localization

GNSS-based localization is subject to local position drift and is unreliable close to structures. The usage of RTK-/D-GPS was allowed but penalized. Hence, we localize Splasher w.r.t. the building using LiDAR. In a test run, we collected data to generate a Multi-Resolution Surfel Map (MRSMap) of the arena with the approach from Droeschel and Behnke (2018). During localization, we register the current LiDAR point cloud against the map. The DJI Matrice 210 v2 provides an estimate of its ego-motion from onboard sensors, which we use to update the initial pose. After registration, we update an offset transformation that compensates for the local position drift of the GPS. The translation between consecutive offset updates is bounded to 30 cm to prevent potential jumps from incorrect registration.

7.3 Hole & Thermal Detection

The water jet exiting the nozzle is very narrow, so precise aiming is required in order to successfully extinguish a fire with a limited amount of water. We apply a hole detection algorithm on the LiDAR point clouds to use for relative navigation (see Fig. 25). First, planes are extracted using RANSAC. For each plane, we project the contained points into a virtual camera perpendicular to it. Morphological transformations are applied to the resulting image in order to close gaps between projected points. Shape detection is used on the image to find potential holes. After re-projecting the circles into 3D, we discard overly large or small holes, as the scenario specifies a constant 150 mm diameter for all target holes.

Our thermal detector applies lower (23000) and upper bounds (65535) to threshold[8] the intensity. We find contours in the thresholded image. For each contour, we compute the area, bounding box, center of intensity, as well as min, max, and mean intensity for further processing. We only retain contours of a specific size. After that, we project the LiDAR point cloud into the thermal image and filter points outside of the detected bounding box. Finally, we calculate the centroid and normal direction of the remaining points.

The thermal and the hole detector both output lists of detections $d_i = (p_i, n_i)$, each consisting of a position p_i and a normal n_i. A filtering module processes these detections to reject outliers and combines both detection types into an estimate of the position and orientation of the currently tracked fire. To do this, we keep track of a history $\mathcal{H} = ((p_1, n_1), \ldots, (p_{10}, n_{10}))$ of ten recent valid detections and estimate the target position as running averages over the detection history.

Mind that the detection history may contain both thermal and hole detections. Thermal detections are necessary to determine which of the possible targets is currently on fire. However, we found that hole detections give a more precise estimation of the target position and especially of its orientation. Hence, we use thermal detections to initialize the target estimate and subsequently update it using hole detections if available. Since the detection history contains both types of detections, they are *implicitly* fused by the filter.

[8]The intensity thresholds are unitless.

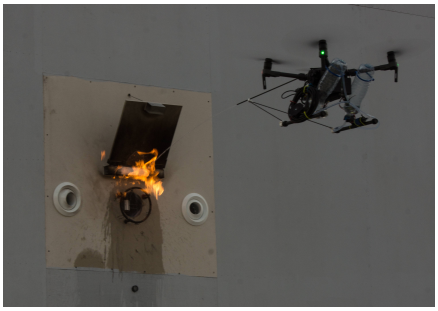


Figure 26: Splasher extinguishing a fire during the MBZIRC 2020.

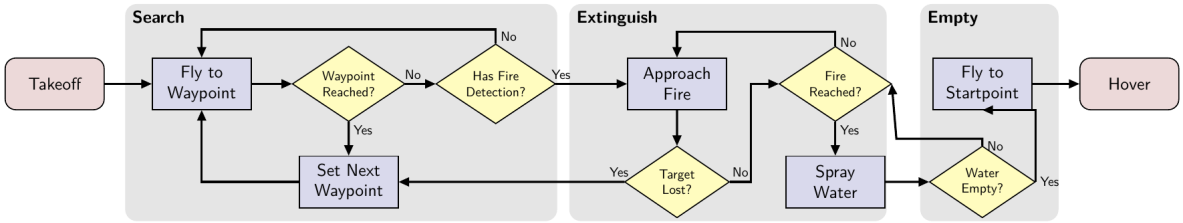


Figure 27: Flowchart of Splasher's state machine.

If one of the detection systems fails and, thus, outputs no detections, the history is only filled with the other type of detection. During regular operation, however, the filtered fire position comprises both detections.

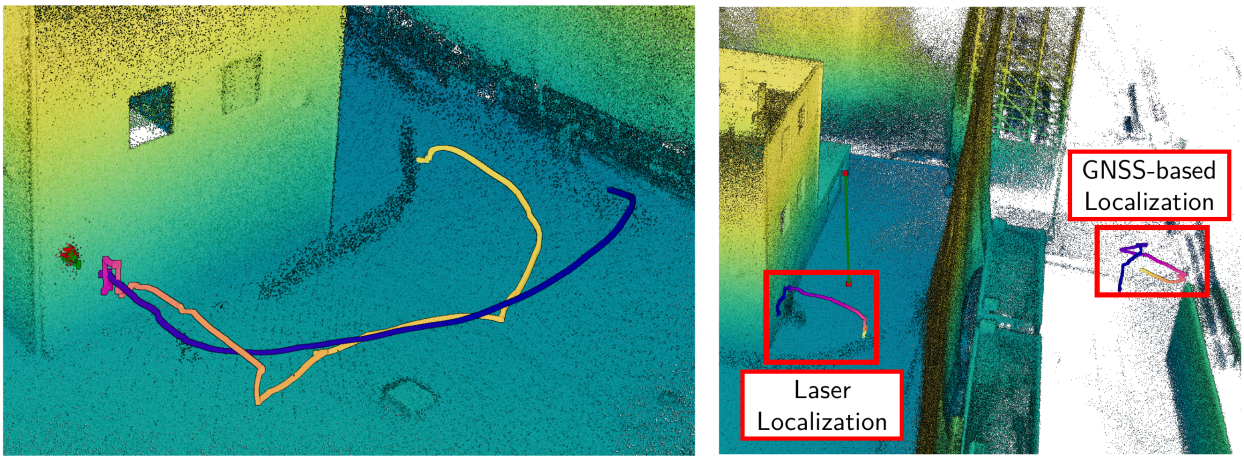
Although we keep track of the most recent thermal detection, we only add it to the detection history $\mathcal{H}$ if there has not been a valid hole detection within the last second. Thus, we ensure that we can still estimate target positions if the hole detector fails, but otherwise only use the more precise information from hole detections. In the case of multiple holes close to the target position, the estimate might drift away from the target if we only use hole detections. Furthermore, we have to recover if initial heat detections are wrong or if the fire was extinguished in the meantime. To address these issues, we only add hole detections to the detection history $\mathcal{H}$ if there has been a heat detection within the last second and the detected hole position lies within a radius of 1.0 m around the latest heat detection.

One might think that the open fire (Fig. 26) would negatively affect LiDAR performance due to smoke and heat. To the contrary, a thorough investigation of the LiDAR scans after the competition trials yielded no significant impact on LiDAR performance.

7.4 Fire-fighting control

The high-level control of Splasher is performed by a Finite State Machine (FSM). The FSM uses inputs from components described above to produce navigation waypoints to locate and approach fires, as well as to control the water spraying during extinguishing. It also ensures that Splasher stays within the arena limits and altitude corridor. The diagram of the FSM is shown in Fig. 27.

***Search* state:** Splasher flies around the building in order to detect and localize a fire on all sidewalls. The route around the building is manually defined as a linked list of waypoints specified in the building frame, obtained from laser localization. While moving between waypoints, the detection and filter pipeline collects data. Some waypoints are marked as *Observation* waypoints. Upon arrival at an *Observation* waypoint, located in front of the known hole locations, Splasher hovers for a predefined duration of 2 s, looking for fires. Splasher switches into the *Extinguish* state if a detection was observed at least five times in a row and it is within the allowed height and angle boundaries.



(a) Fire detection and UAV localization during manual fire extinguishing of the windy ground-level facade fire during the second challenge day. The trajectory is color-coded by time, yellow to blue.

(b) Incorrect GNSS-based UAV localization during the Grand Challenge. The laser localization shows the trajectory before crashing into the building.

Figure 28: UAV localization.

***Extinguish* state:** The purpose of this state is to arrive at the extinguishing position without losing the detected fire on the way. We do so by steering the robot relative to the detected fire (i.e., visual servoing to the target position). To do so, we first transform the egocentric fire detection into an allocentric frame. Based on the allocentric fire detection, we derive an extinguishing waypoint that lies relative to the detection (horizontal offset: 2.1 m, vertical offset: 0.35 m), which Splasher targets. Once Splasher reaches the goal pose, the pump is started. It is stopped as soon as Splasher deviates from the current goal pose more than a predefined threshold for either position or heading. Splasher keeps track of the fire and updates the goal pose accordingly. If Splasher did *not* lose the target during extinguishing, the pump is stopped after all water has been sprayed and we enter the *Refilling* state.

***Refilling* state:** We do not *directly* measure water content but measure the time the pump is active. Once the water reserve is depleted, Splasher flies back to the starting position and hovers there awaiting manual landing and refueling of the water storage.

7.5 Evaluation

In the first scored challenge run, software issues and wrong predefined search poses prevented Splasher from detecting any fire. On the second challenge day, we experienced incorrect height estimates. We attribute this to the ultrasonic sensor measuring the building wall rather than the ground, thus estimating the height too low. In hindsight, we believe that a height sensor with a smaller opening angle (like employed in Challenge 1) could have solved this problem, but we were not aware of the problem's severity during the competition.

We noticed that during all trials Splasher flew too high to detect the fire. We then switched to manual mode and were able to fill the container of the windy ground-level facade fire with 322 ml of water near the end of the second challenge run. The manually flown trajectory and localized detections of the fire are shown in Fig. 28. After the Grand Challenge, we found out that our LiDAR-based localization was disabled the whole time.

As described in Sec. 3.2, we employ tools that should catch errors like disabled components. Unfortunately, we did *not* include the laser localization in our supervision system since we never expected such a core component to be subject to human error.

(a) World model of the scene. (b) Bird's-eye view. (c) Onboard thermal camera. (d) Onboard RGB-camera.

Figure 29: Autonomous fire extinguishing on mockup target with heating element. Red line segments indicate the UAV's heading at various numbered waypoints (orange). The waypoints are targeted in ascending order on the colored trajectory. 1) The UAV starts 2,3) It searches for the heat source 4) It found the heat source 5) It approaches the wall and extinguishes the fire.

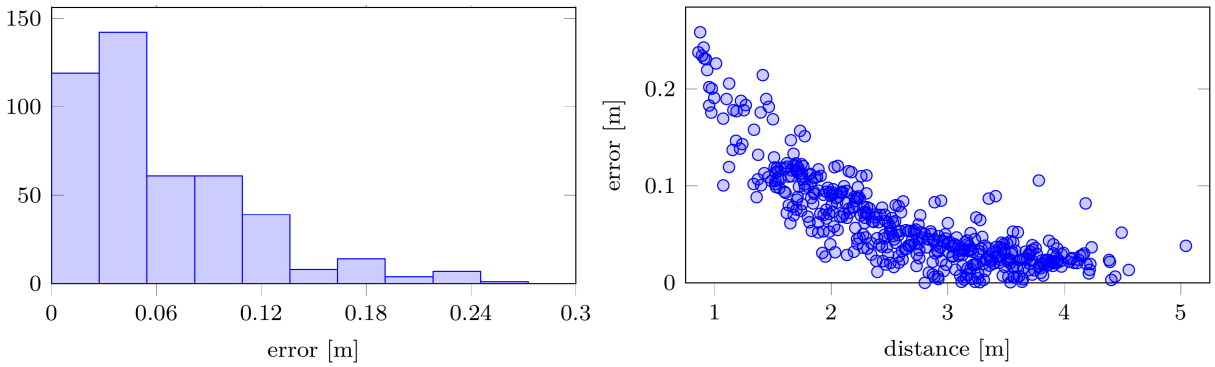
Shortly after the Grand Challenge started, a series of unfortunate circumstances led to Splasher crashing into the building, as shown in Fig. 28. The laser localization was disabled due to human error, and position estimates relied solely on the GNSS-based ego-motion estimates. To compensate for position drift, we added a static offset to the GPS poses, which was calculated as the difference between the predefined start position and the GPS pose when the challenge started. However, the GPS signal strongly drifted while computing the offset. Thus, a large offset was added to the GPS pose, resulting in an initial localization error of approximately 20 m. Moreover, it continued to drift afterwards. The canyon-like starting position between the building and the operator booth, as well as interference with our UGV, might be responsible for the drift.

After repair, we performed further lab tests to showcase Splasher's abilities on a mockup target (Fig. 29). After lift-off, Splasher flies towards multiple predefined search waypoints (Fig. 29a, Poses 1–3). The heat source is first detected at Waypoint 3. Now Splasher begins to navigate only relative to the detection and turns towards it before flying closer (Pose 4) and extinguishing the fire (Fig. 29b–Fig. 29d). A video showcasing the evaluation can be found on our website[9]. We repeated the experiment on the mockup seven times. In all cases, the target was first detected on Pose 3. It took on average 1.01 s from first detection to a stable filter estimate and already 4.96–6.60 s later the target was reached and the pump started spraying water for on average 14.57 s. The mean GPS drift from first detection until spraying stops was 0.68 m while our UAV remained in a stable position relative to the heat source. This highlights the necessity of relative navigation for such a high-precision task.

In a separate test, we evaluate the accuracy for our estimated distance towards the heat source from projecting LiDAR into thermal detections at different ranges. We derive ground truth on the mockup from fitting a plane to its outside wall. The histogram in Fig. 30a shows that the majority of all estimates has an error below 0.05 m and less than 90 % have more then 0.12 m error. Counterintuitively, close range measurements were least reliable, as visualized in Fig. 30b. We found that more projected LiDAR measurements stem from the backside of the mockup and since we fuse measurements within the bounding box, the distance is overestimated and the error increases. In general, the accuracy is sufficient to approach the target until reliable hole detections are available.

Fig. 31 compares the estimated distances of hole and thermal detections. We combined the trial runs on Day 1 and 2 of MBZIRC in Fig. 31a. The thermal detection provided first distance measures at up to 16.5 m which is four times the maximal distance for the hole detection with 4.07 m. In contrast, during our trials with the mockup target (Fig. 29), both methods provided detections with a maximum distance of approx. 5 m. We attribute the difference in behavior to the heat sources and sensor combination. The low thermal sensor resolution allowed us to detect the mockup's heating element at a similar range to the hole detection.

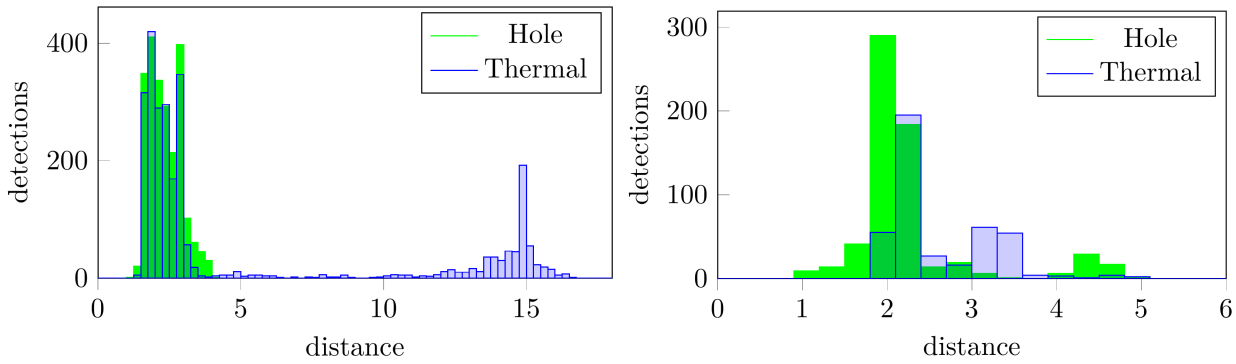
[9] `https://www.ais.uni-bonn.de/videos/fr_2021_mbzirc`



(a) Error histogram.

(b) Error over distance.

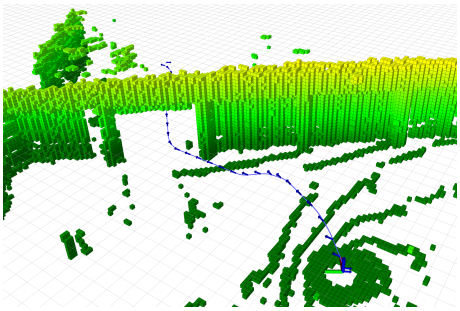
Figure 30: Evaluation of distance for thermal detections with depth obtained by projecting LiDAR on the mockup target against ground truth from plane estimation. The error histogram shows sufficient accuracy to approach the target while the error increases at close range by partially measuring the backside through the hole.



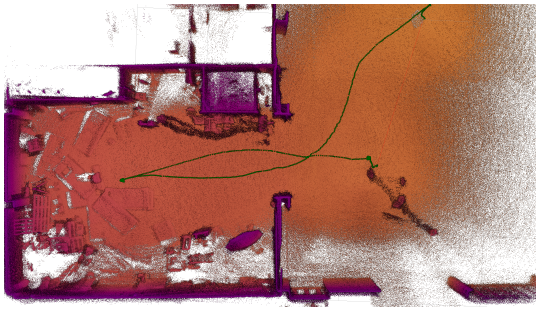
(a) MBZIRC Challenge Day 1 and 2.

(b) Mockup trials.

Figure 31: Distance histogram for thermal and hole detections combined for MBZIRC Trial Days 1 and 2 (left) as well as the trials with the mockup target.



(a) Initially planned trajectory.



(b) Top-down view of an aggregated point cloud and a trajectory (green) generated using our LiDAR-based odometry.

Figure 32: Autonomous indoor flight experiment.

During the challenge, the heat source was detectable further away due to the surrounding fire. The apparent difference in the absolute number of detections in Fig. 31b originates from the higher LiDAR scan frequency. Furthermore, we attribute for thermal detections the generally larger distance estimates at close range to measurements on the backside of the mockup.

7.6 Autonomous Indoor Flight

During the competition, we employed Splasher only for outdoor fire extinguishing and addressed indoor tasks using our UGV. This allowed us to use a combination of GPS and LiDAR localization. However, we experienced severe localization errors due to limited GPS visibility and human errors. Although we later proved the viability of our approach on a mockup on an open field, we decided to increase the robustness of our localization method such that it can even be applied to challenging indoor scenarios. Thus, we developed a method for LiDAR-based odometry.

We model surfaces within LiDAR scans with normal distributions derived from measured points on a uniform sparse voxel grid. Our odometry (Quenzel and Behnke, 2021) uses a sliding registration window to simultaneously register multiple surface element (surfel) maps against a local surfel map. A continuous-time Lie group B-Spline (Sommer et al., 2020) describes the UAV trajectory within the sliding registration window. After a certain traveled distance, we add the last scan in a keyframe-based sliding window approach to the local surfel map.

We exchanged our previous localization method with the new LiDAR-based odometry. An Extended Kalman Filter fuses the resulting position measurements with IMU data to generate estimates of the high-dimensional UAV state. Additionally, we use the method of Schleich and Behnke (2021) to plan a collision-free trajectory instead of manually defining waypoints.

We evaluated our updated system by autonomously exploring the inside of an industrial building. The UAV started outside and had to enter through a gate to reach a manually defined observation pose, as depicted in Fig. 32a. There, it rotated to generate an overview of the environment and left the building again. Fig. 32b shows an aggregated point cloud of the environment and a flight trajectory generated using our LiDAR-based odometry. Additional example images of our UAV during the flight are shown in Fig. 33.

(a) Our UAV at the indoor observation pose.

(b) Our UAV autonomously exits the building while being supervised by a safety pilot.

Figure 33: Our UAV during autonomous flight in a GNSS-denied environment.

Table 6: Points (Ranks) at MBZIRC 2020 Grand Challenge.

Team	Ch. 1	Ch. 2	Ch. 3	Time left	Sum of Ranks	GC Rank
CTU Prague & UPenn & and NYU	**72.0 (1)**	0.0 (8)	12.5 (2)	10	**11**	**1**
Team NimbRo (Bonn)	30.0 (4)	**0.0 (7)**	**12.5 (1)**	**20**	12	2
UPM & UPO & PUT & CNRS	40.5 (2)	0.0 (9)	0.1331 (5)	0	16	3

8 Lessons Learned

To bring our performance in context, we depict the Grand Challenge scores of the best teams in Tab. 6. One can see that no team was able to score in the wall-building challenge[10]. The rank was determined as a sum of the individual challenge ranks. While in Challenge 1, no tiebreaker was needed, our rank (7) in Challenge 2 and Challenge 3 (1) was mainly determined by the time left.

We take the opportunity to identify the key strengths and weaknesses of our system and development approach. We also want to identify aspects of the competition that could be improved to increase scientific usefulness in the future.

Our hardware designs proved themselves during the competition after small adaptions to the conditions at hand. For example, after solving initial problems with our magnets, especially the passive gripper turned out to be an advantage over other teams, who could not manipulate the heavier bricks.

The biggest issue shortly before and during the competition was unavailable testing time. Robust solutions require full-stack testing under competition constraints. Since we postponed many design decisions until the competition rules were settled, we could not test our intricate design thoroughly. In hindsight, simpler designs with fewer components, which would have required less thorough testing, could have been more successful in the short available time frame.

While we tested our UAVs on small mockups in our lab and on an open field, the competition environment was very different and led to system failures. For example, GPS visibility was severely limited near the mandatory start position, which led to initialization errors. An initialization-free or delayed initialization scheme, which we implemented later, or even a GPS drift detection would have improved robustness. Furthermore, relative navigation enables task fulfillment with unreliable pose information, and low-level obstacle avoidance is mandatory in such a scenario.

Also, improved visualization of the UAV state and perception could have helped to detect problems early on. This includes a thorough application of our supervision system for every component to catch errors like

[10]Only one of 17 teams was able to score in this subchallenge.

the unintentionally disabled laser localization subsystem.

The competition also placed an enormous strain on the involved personnel. For safe operation, one safety pilot was required per UAV, plus at least one team member supervising the internal state of the autonomous systems and being able to reconfigure the system during resets. For example, in the Grand Challenge, this led to the situation that a safety pilot had to run from one arena to another depending on which subchallenge was active, unnecessarily delaying the run. While reducing the number of required human operators per robot is an admirable research goal, this is not possible in the near future due to safety regulations. We thus feel that future competitions should keep the required human personnel in mind so that small teams can continue to participate.

What proved to be useful in the context of competitions is to prepare low-effort backup strategies in advance, like assistance functions for manual mode. Also, reducing high-level control to a bare minimum, instead of universal strategies, makes the systems easier to test and reduces potential errors.

For example, in the balloon hunting challenge, our approach could not incorporate the arena's non-convex shape. Instead of using an untested elaborated path planning approach, we used the low-effort strategy of inserting an intermediate waypoint in the middle of the arena, which required minimal testing.

This edition of the MBZIRC suffered from overall low team performance, to the extent that the Grand Challenge price money was not paid out on the jury's recommendation. This underperformance of all teams points to systematic issues with the competition. From a participants' perspective, we think late changes to the rules have certainly contributed to this situation. A pre-competition event such as the Testbed in the DARPA Robotics Challenge can help identify critical issues with rules and material early in the competition timeline.

Another issue was the required effort to participate in all the different sub-challenges. The MBZIRC 2020 defined seven different tasks. Ideally, one would develop specialized solutions for all of these. Focusing the competition more on general usability, i.e., defining multiple tasks that can *and should* be completed by one platform would lower the barrier for participants.

9 Conclusion

We demonstrated successful hardware design, perception and control methods, high-level control, and system integration for a highly complex robotic challenge. The lessons learned discussed above in Section 8 already contain hints about individual improvements that can be applied to sub-components, team strategy, or competition organization.

On a more global scale, we firmly believe that robotic competitions such as MBZIRC are key for our community, as they force researchers to test their algorithms under real-world, integrated conditions. Public benchmarks are the only way to properly evaluate systems in this direction. Contrary to comparable challenges, MBZIRC provides very varied and complex challenges. While this is highly interesting and provides research challenges, it would be good to have similar tasks from one competition edition to the next to give the community time to learn from their promising approaches.

Finally, while we made technical contributions to various aspects in each sub-challenge, it is clear that all of them warrant further research to increase flexibility and applicability to other domains.

Acknowledgments

We thank all members of our team NimbRo for their support before and during the competition. This work has been supported by a grant from the Mohamed Bin Zayed International Robotics Challenge (MBZIRC).

References

Ando, H., Ambe, Y., Ishii, A., Konyo, M., Tadakuma, K., Maruyama, S., and Tadokoro, S. (2018). Aerial hose type robot by water jet for fire fighting. *IEEE Robotics and Automation Letters (RA-L)*, 3(2):1128–1135.

Aydin, B., Selvi, E., Tao, J., and Starek, M. J. (2019). Use of fire-extinguishing balls for a conceptual system of drone-assisted wildfire fighting. *Drones*, 3(1):17.

Baca, T., Penicka, R., Stepan, P., Petrlik, M., Spurny, V., Hert, D., and Saska, M. (2020). Autonomous cooperative wall building by a team of unmanned aerial vehicles in the mbzirc 2020 competition. *arXiv e-prints*, arXiv:2012.05946.

Baca, T., Stepan, P., and Saska, M. (2017). Autonomous landing on a moving car with unmanned aerial vehicle. In *Proc. of European Conf. on Mobile Robots (ECMR)*.

Bähnemann, R., Pantic, M., Popović, M., Schindler, D., Tranzatto, M., Kamel, M., Grimm, M., Widauer, J., Siegwart, R., and Nieto, J. (2019). The ETH-MAV team in the MBZ international robotics challenge. *Jnl. of Field Robotics*, 36(1):78–103.

Bailon-Ruiz, R. and Lacroix, S. (2020). Wildfire remote sensing with UAVs: A review from the autonomy point of view. In *Proc. of Int. Conf. on Unmanned Aircraft Systems (ICUAS)*.

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

Beul, M. and Behnke, S. (2016). Analytical time-optimal trajectory generation and control for multirotors. In *Proc. of Int. Conf. on Unmanned Aircraft Systems (ICUAS)*.

Beul, M. and Behnke, S. (2017). Fast full state trajectory generation for multirotors. In *Proc. of Int. Conf. on Unmanned Aircraft Systems (ICUAS)*.

Beul, M., Bultmann, S., Rochow, A., Rosu, R. A., Schleich, D., Splietker, M., and Behnke, S. (2020). Visually guided balloon popping with an autonomous MAV at MBZIRC 2020. In *Proceedings of the IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*.

Beul, M., Houben, S., Nieuwenhuisen, M., and Behnke, S. (2017). Fast autonomous landing on a moving target at MBZIRC. In *Proc. of European Conf. on Mobile Robots (ECMR)*.

Beul, M., Nieuwenhuisen, M., Quenzel, J., Rosu, R. A., Horn, J., Pavlichenko, D., Houben, S., and Behnke, S. (2019). Team NimbRo at MBZIRC 2017: Fast landing on a moving target and treasure hunting with a team of micro aerial vehicles. *Jnl. Field Rob.*, 36(1).

Borrmann, D., Elseberg, J., and Nüchter, A. (2013). Thermal 3D mapping of building façades. In *Proceedings of Int. Conf. on Intelligent Autonomous Systems (IAS)*, pages 173–182. Springer.

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

Carlson, A., Skinner, K., Vasudevan, R., and Johnson-Roberson, M. (2019). *Modeling Camera Effects to Improve Visual Learning from Synthetic Data*, pages 505–520.

Cho, Y. K., Ham, Y., and Golpavar-Fard, M. (2015). 3D as-is building energy modeling and diagnostics: A review of the state-of-the-art. *Advanced Engineering Informatics*, 29(2):184 – 195.

Delmerico, J., Mintchev, S., Giusti, A., Gromov, B., Melo, K., Horvat, T., Cadena, C., Hutter, M., Ijspeert, A., Floreano, D., et al. (2019). The current state and future outlook of rescue robotics. *J. of Field Robotics*, 36(7):1171–1191.

Demisse, G., Borrmann, D., and Nüchter, A. (2015). Interpreting thermal 3D models of indoor environments for energy efficiency. *J. Intell. Robot. Syst.*, 77(1):55–72.

Droeschel, D. and Behnke, S. (2018). Efficient continuous-time SLAM for 3D lidar-based online mapping. In *Proceedings of IEEE Int. Conf. on Robotics and Automation (ICRA)*.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2010). The Pascal Visual Object Classes (VOC) Challenge. *Int J Comput Vis*, 88(2):303–338.

Ezair, B., Tassa, T., and Shiller, Z. (2014). Planning high order trajectories with general initial and final conditions and asymmetric bounds. *The International Journal of Robotics Research*, 33(6):898–916.

Falanga, D., Zanchettin, A., Simovic, A., Delmerico, J., and Scaramuzza, D. (2017). Vision-based autonomous quadrotor landing on a moving platform. In *Proceedings of the IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*.

Fritsche, P., Zeise, B., Hemme, P., and Wagner, B. (2017). Fusion of radar, LiDAR and thermal information for hazard detection in low visibility environments. In *Proceedings of the IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, pages 96–101.

Ghamry, K. A., Kamel, M. A., and Zhang, Y. (2016). Cooperative forest monitoring and fire detection using a team of UAVs-UGVs. In *Proc. of Int. Conf. on Unmanned Aircraft Systems (ICUAS)*, pages 1206–1211. IEEE.

Goessens, S., Mueller, C., and Latteur, P. (2018). Feasibility study for drone-based masonry construction of real-scale structures. *Automation in Construction*, 94.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR*, abs/1704.04861.

Huber, F., Kondak, K., Krieger, K., Sommer, D., Schwarzbach, M., Laiacker, M., Kossyk, I., Parusel, S., Haddadin, S., and Albu-Schäffer, A. (2013). First analysis and experiments in aerial manipulation using fully actuated redundant robot arm. In *Proceedings of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2704–2713.

Jindal, K., Wang, A., Thakur, D., Zhou, A., Spurny, V., Walter, V., Broughton, G., Krajnik, T., Saska, M., and Loianno, G. (2021). Design and deployment of an autonomous unmanned ground vehicle for urban firefighting scenarios. *arXiv e-prints*, arXiv:2107.03582.

Juhász, Z., Sipos, Á., and Porkoláb, Z. (2008). *Implementation of a Finite State Machine with Active Libraries in C++*, pages 474–488. Springer Berlin Heidelberg, Berlin, Heidelberg.

Kim, S., Choi, S., and Kim, H. J. (2013). Aerial manipulation using a quadrotor with a two DOF robotic arm. In *Proceedings of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*.

Krizmancic, M., Arbanas, B., Petrovic, T., Petric, F., and Bogdan, S. (2020). Cooperative aerial-ground multi-robot system for automated construction tasks. *IEEE Robotics and Automation Letters*, 5(2):798–805.

Lenz, C., Quenzel, J., Periyasamy, A. S., Razlaw, J., Rochow, A., Splietker, M., Schreiber, M., Schwarz, M., Süberkrüb, F., and Behnke, S. (2021). Autonomous wall-building and firefighting: Team NimbRo's UGV solution for MBZIRC 2020. Accepted for Field Robotics.

Lenz, C., Schwarz, M., Rochow, A., Razlaw, J., Periyasamy, A. S., Schreiber, M., and Behnke, S. (2020). Autonomous wall building with a UGV-UAV team at MBZIRC 2020. In *Proceedings of the IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*.

Lindsey, Q., Mellinger, D., and Kumar, V. (2012). Construction with quadrotor teams. *Autonomous Robots*, 33(3):323–336.

Liu, P., Yu, H., Cang, S., and Vladareanu, L. (2016a). Robot-assisted smart firefighting and interdisciplinary perspectives. In *Proceedings of the International Conference on Automation and Computing (ICAC)*, pages 395–401. IEEE.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016b). Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer.

Loianno, G., Spurny, V., Thomas, J., Baca, T., Thakur, D., Hert, D., Penicka, R., Krajnik, T., Zhou, A., Cho, A., Saska, M., and Kumar, V. (2018). Localization, grasping, and transportation of magnetic objects by a team of MAVs in challenging desert-like environments. *IEEE Robotics and Automation Letters*, 3(3):1576–1583.

Maninis, K.-K., Pont-Tuset, J., Arbeláez, P., and Gool, L. V. (2018). Convolutional Oriented Boundaries: From Image Segmentation to High-Level Tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 40(4):819 – 833.

Michael, N., Fink, J., and Kumar, V. (2011). Cooperative manipulation and transportation with aerial robots. In *Autonomous Robots*, volume 30, pages 73–86.

Michael, N., Shen, S., Mohta, K., Mulgaonkar, Y., Kumar, V., Nagatani, K., Okada, Y., Kiribayashi, S., Otake, K., Yoshida, K., Ohno, K., Takeuchi, E., and Tadokoro, S. (2012). Collaborative mapping of an earthquake-damaged building via ground and aerial robots. *J. of Field Robotics*, 29(5):832–841.

Moore, T. and Stouch, D. (2014). A generalized extended Kalman filter implementation for the robot operating system. In *Proceedings of Int. Conf. on Intelligent Autonomous Systems (IAS)*. Springer.

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

Quenzel, J. and Behnke, S. (2021). Real-time multi-adaptive-resolution-surfel 6D LiDAR odometry using continuous-time trajectory optimization. In *Proceedings of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*.

Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). ROS: An open-source robot operating system. In *ICRA Workshop on Open Source Software*.

Real, F., Ángel R. Castaño, Torres-González, A., Capitán, J., Sanchez-Cuevas, P. J., Fernandez, M. J., Villar, M., and Ollero, A. (2021). Experimental evaluation of a team of multiple unmanned aerial vehicles for cooperative construction. *IEEE Access*, 9:6817–6835.

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

Romero-Ramirez, F. J., Muñoz-Salinas, R., and Medina-Carnicer, R. (2018). Speeded up detection of squared fiducial markers. *Image and Vision Computing*, 76:38–47.

Rosu, R. A. and Behnke, S. (2020). Easypbr: A lightweight physically-based renderer. In *Proceedings of the International Conference on Computer Graphics Theory and Applications (GRAPP)*.

Rosu, R. A., Quenzel, J., and Behnke, S. (2019). Reconstruction of textured meshes for fire and heat source detection. In *Proceedings of the IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, pages 235–242. IEEE.

Ruggiero, F., Lippiello, V., and Ollero, A. (2018). Aerial manipulation: A literature review. *Rob. and Automation Letters*, 3(3).

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520.

Schleich, D. and Behnke, S. (2021). Search-based planning of dynamic MAV trajectories using local multiresolution state lattices. In *Proceedings of IEEE Int. Conf. on Robotics and Automation (ICRA)*.

Schönauer, C., Vonach, E., Gerstweiler, G., and Kaufmann, H. (2013). 3D building reconstruction and thermal mapping in fire brigade operations. In *Proceedings of the 4th Augmented Human International Conference*, pages 202–205. ACM.

Sommer, C., Usenko, V., Schubert, D., Demmel, N., and Cremers, D. (2020). Efficient derivative computation for cumulative B-Splines on Lie groups. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Spurnỳ, V., Báča, T., Saska, M., Pěnička, R., Krajník, T., Thomas, J., Thakur, D., Loianno, G., and Kumar, V. (2019). Cooperative autonomous search, grasping, and delivering in a treasure hunt scenario by a team of UAVs. *Jnl. Field Rob.*, 36(1).

Spurny, V., Pritzl, V., Walter, V., Petrlik, M., Baca, T., Stepan, P., Zaitlik, D., and Saska, M. (2021). Autonomous firefighting inside buildings by an unmanned aerial vehicle. *IEEE Access*, 9:15872–15890.

Suarez Fernandez, R., Rodríguez Ramos, A., Alvarez, A., Rodríguez-Vázquez, J., Bavle, H., Lu, L., Fernandez, M., Rodelgo, A., Cobano, A., Alejo, D., Acedo, D., Rey, R., Martinez-Rozas, S., Molina, M., Merino, L., Caballero, F., and Campoy, P. (2020). The SkyEye team participation in the 2020 Mohamed Bin Zayed International Robotics Challenge. In *Mohamed Bin Zayed International Robotics Competition (MBZIRC) Symposium*.

Tran, V. P., Santoso, F., Garratt, M. A., and Anavatti, S. G. (2021). Distributed artificial neural networks-based adaptive strictly negative imaginary formation controllers for unmanned aerial vehicles in time-varying environments. *IEEE Transactions on Industrial Informatics*, 17(6):3910–3919.

Yang, J., Price, B., Cohen, S., Lee, H., and Yang, M.-H. (2016). Object Contour Detection with a Fully Convolutional Encoder-Decoder Network. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 193–202, Las Vegas, NV, USA. IEEE.