UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Team of multiple autonomous UAVs interacting with static objects

Author:
Yurii STASINCHUK

Supervisor:
Dr. rer. nat Martin SASKA

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Faculty of Applied Sciences
Department of Computer Sciences

Lviv 2020

Declaration of Authorship

I, Yurii STASINCHUK, declare that this thesis titled, Team of multiple autonomous UAVs interacting with static objects and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences
Department of Computer Sciences

Bachelor of Science

Team of multiple autonomous UAVs interacting with static objects

by Yurii STASINCHUK

Abstract

The Mohamed Bin Zayed International Robotics Challenge (MBZIRC) is a prestigious competition, aimed at furthering the state-of-the-art in the field of autonomous robotics. This thesis presents my part of the solution to one of the tasks in the MBZIRC 2020 competition, which landed us a second place in Challenge 1 and a first place in the Grand Challenge of the competition.

Specifically, this thesis focuses on the balloon popping task, which had to be solved quickly and robustly in order to compete with systems from other expert robotic teams from the whole world.

In this task, a team of cooperating Unmanned Aerial Vehicles (UAVs) had to autonomously search and destroy balloons, placed in a designated arena, as fast as possible. A software and hardware overview of the used UAV platform is presented and the detection, estimation and planning algorithms of our winning solution are described in detail. Evaluation of the described methods on data from the competition is presented.

Acknowledgements

I would like to thank my supervisor Martin Saska for the patience and sharing his knowledge. Tomas Bača and Matouš Vrba for sharing their expertise and helping me to go through the challenge. Furthermore, I want to thank all members of the Multi-Robot Systems group that believed in me and gave me a chance to participate in the MBZIRC2020 challenge. This also would not be possible without support from the Faculty of Applied Sciences, who gave me a chance to change my studies. My final thanks go to my mother and my friends, who supported me throughout all my studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Mohamed Bin Zayed International Robotics Challenge (MBZIRC) is a prestigious competition, aimed at furthering the state-of-the-art in the field of autonomous robotics. The competition contains three separate challenges and the grand challenge as a combination of all three of the challenges done simultaneously. This thesis mainly focuses on solving the planning and color calibration for the balloon hunting task of Challenge 1 of the MBZIRC 2020.

Unmanned aerial vehicles (UAVs) recently saw a rise in usage across different fields. The utility of UAVs is rising, and lots of industrial applications done and planned [5] in the future.

The first challenge was motivated by intruder drone attacks that were devastating for middle east oil refineries, which couldn't be overcome with missiles since the drone targets are very small, and can surpass the defenses. The attacks caused losses of oil and economic issues for the region[1]. The challenge is meant to enhance research of defending from small UAVs.

Challenge 1 had two tasks that were solved simultaneously by three UAVs. The first part was to catch a ball that was attached to the intruder UAV. The second task was to locate and destroy static targets, represented by balloons, while actively avoiding obstacles in the area. Each target was tethered to the ground via a metal pole (see Figure 1.1). This thesis focuses on solving the balloon hunting task using an autonomous multi-robotic approach with two UAVs.

The length of the metal poles is undefined and may vary from 1 to 5 meters. The poles were placed inside an area (later referred to as arena) 100 meters long and 60 meters wide, and had a ceiling 20 meters high. The arena itself was shaped as a non-convex polygon (see Figure 1.2), whose edges were covered with a net to protect the viewers in case UAV goes out of control.

The goal of the thesis was to develop the necessary methods and algorithms for autonomous searching and elimination of the balloon targets, which would solve the task as fast as possible. The challenge rules state that overall up to 3 UAVs can be used to complete the whole task. In this thesis a solution for 2 UAVs is developed.

Chapter 1 provides a detailed overview of the problem and related state of the art methods and research. Details of the hardware platform are provided in Chapter 2. Chapter 3 provides overview of the high level state machine that was used to solve the MBZIRC balloon hunting task. Chapter 4 describes the balloon detection algorithm. In Chapter 5 generation of trajectories for arena scanning and target approach is explained. Experimental evaluation of the solution is described in Chapter 6.

[1] https://www.cnbc.com/2019/09/15/saudi-stock-market-dives-crude-to-jump-after-attack-on-oil-plants.html

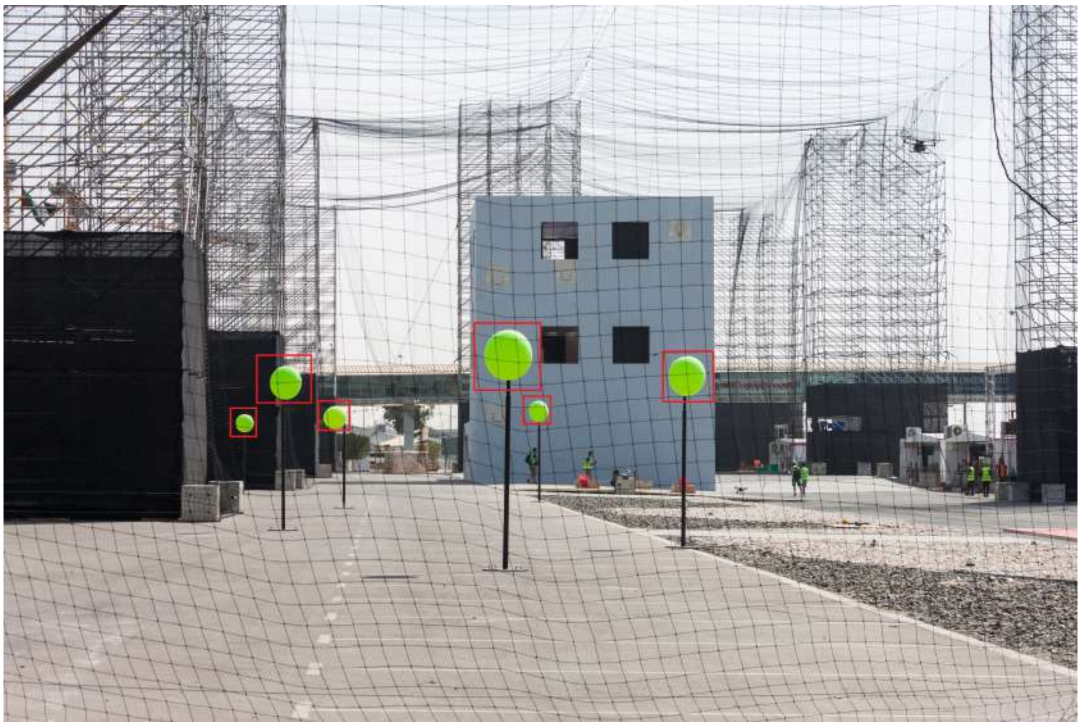


FIGURE 1.1: The target balloons are tethered to the poles in the Challenge 1 arena. The targets are highlighted using red rectangles.

1.1 Related Works

Although the task of the MBZIRC 2020 first Challenge is very specific, it is similar to the "Treasure Hunt" task that was introduced at the previous competition that was held in 2017. There are different solutions to that problem, for example [20] and [4]. They have one thing in common - mapping of the objects and then proceed to interact with them. This can be done with the current task, but there are only five targets, so in this thesis, the greedy approach was selected to search and destroy without mapping.

The search and target tracking scenarios are also studied using UAVs [9] and [10], but most of the work is aiming to solve the task cooperatively, with the dynamic interaction of the UAVs, which is not crucial for the thesis task. The task can be distributed via the team of UAVs, and they will perform them faster, using more straightforward and faster methods and not relying on heavy communication.

The task is mainly motivated by the problems of small intruder UAVs, which can surpass the defenses. This has also been known in the area, and there are different researches done to help to solve this issue like in [21].

For robust and accurate detection of the objects HSV [11] and La*b* [7] models are used. Using these models color together with color segmentation provided accurate and fast target detection. Though there are multiple solutions to this problem, for example, using the neural networks like in [6] or [8], the neural networks can be specifically trained for the balloons, and give satisfactory results. Though, the neural networks require a high amount of computational power or a Graphical Processing Unit (GPU) for these purposes. As described in chapter 2, UAV does not have such capabilities, so the color segmentation solution was chosen.

1.2 Preliminaries

The algorithms presented in this thesis were deployed using a software system, which is described in this section. Notably, usage of the Robot Operation System (ROS) [17], was used to implement the MRS system to control the drone. The MPC tracker [3] and the SO(3) controller [14] is provided.

Robot Operating System

ROS is an open-source robotics middleware (collection of software frameworks for robot software development). ROS serves as a structured communication layer above the host operating system. The system is released under BSD license and is opened to contributors.

In this thesis, the described algorithm was developed using ROS melodic version installed on the Ubuntu 18.04 operating system. This framework provides a system for communication between nodes (which represent different packages). ROS is an open-source system, so using this framework, it easy to connect different hardware to it because someone has adapted it for that, like a camera, for example (details in section 2.3). Each node can stream and listen to data from other nodes. This gives the distribution of work that is done and increases the flexibility of the system.

ROS integrates an API to represent various coordinate systems (frames) in the world (see Figure 1.4 for visualization of the frame). The API may be used to easily transform data between the frames e.g. from a sensor frame to a static world frame. Relevant frames for the purpose of this thesis are:

- **GPS frame** - GPS fixed frame. Usually the origin of this frame is placed inside of the area where UAV will operate, in our case near the center of the MBZIRC arena. This frame is the core of the flight for this thesis, because GPS navigation was used to perform all the tasks.

- **Arena frame** - The arena origin is very close to the GPS origin, but it's frame is rotated so the axis is parallel to borders of the arena itself. This frame was used to plan all the trajectories for the balloon hunting scenarios.

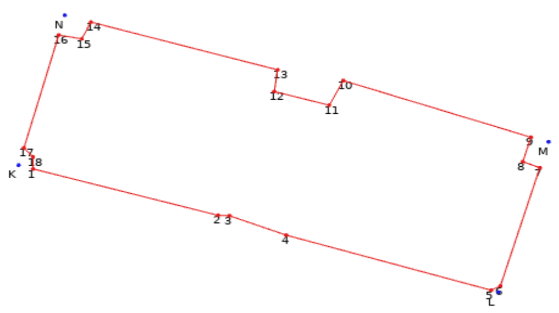


FIGURE 1.2: A top-view layout of the MBZIRC 2020 Challenge 1 operation arena as it was manually measured using a GPS (a photo of the arena is in Figure 1.1). Edges of the arena are represented by a red line, the arena has 18 vertices, each one is numbered, the K,L,M,N blue dots represent corners of the arena rectangle.

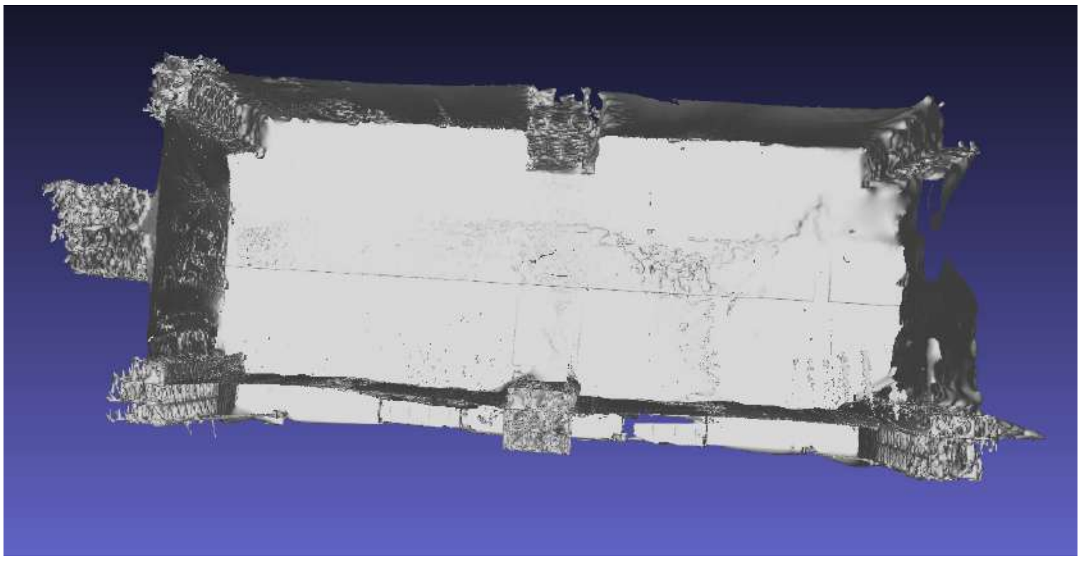


FIGURE 1.3: Top-view of the MBZIRC 2020 Challenge 1 operation arena 3D model as it was scanned using a Leica 3D scanner[2]

- **UAV frame** - The origin of this frame is located in the center of the flight control unit (FCU) of the UAV, and it is oriented with respect to the current orientation of the UAV.
- **Camera frame** - The detections of the balloons, obtained from a camera sensor are expressed in this frame.

1.3 MRS Control pipeline

In this thesis, the MRS control system [2] was used to control the drone. The system is released and available on GitHub [3]. In the system UAV pose is represented by translation $\mathbf{r} = [x, y, z]^T$, and an orientation $\mathbf{R}(\phi, \theta, \psi)$. These represent the relation between the world frame $\omega = \{e_1, e_2, e_3\}$ and the body frame $\mathbf{b} = \{b_1, b_2, b_3\}$ (see Figure 1.4).

[2] `https://leica-geosystems.com/products/laser-scanners/scanners/blk360`
[3] `https://github.com/ctu-mrs/uav_core`

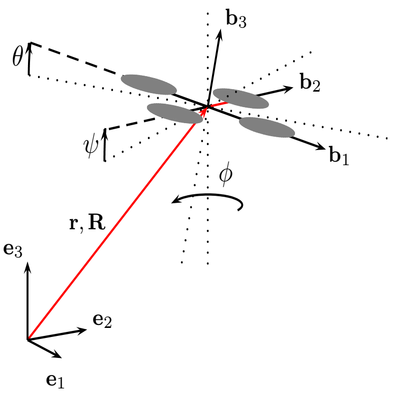


FIGURE 1.4: Taken from [2], where translation $\mathbf{r} = [x, y, z]^T$, and an orientation $\mathbf{R}(\phi, \theta, \psi)$ is the orientation, they represent relation between the world frame $\mathbf{!} = \{e_1, e_2, e_3\}$ and the body frame $\mathbf{b} = \{b_1, b_2, b_3\}$. Courtesy of [16].

Following this UAV state vector is as follows

$$\mathbf{x} = (\mathbf{r}, \mathbf{R}) \tag{1.1}$$

The MRS control system provides an API for positioning and commanding the drone via ROS services. The main module with which the user interacts is the Mission planner. It receives trajectories or UAV desired position. It will reshape the trajectory and execute it with current constraints. Constraints are limitations for UAV speed and acceleration in horizontal, vertical planes, as well as for limitation for how fast the heading can change. In Table 1.1, the constraints that are used in this thesis are shown.

The sweeping constraints are meant for fast change of heading on the turns (see ϕ in Figure 1.4), but without significant acceleration to avoid tilt (see ψ in Figure 1.4) of the UAV. This is done to ensure that the camera of the drone is always looking forward and searching for targets.

The attack constraints have less limitation in speed and acceleration, to ensure fast approaching and destroying of the target.

Name	**Sweeping**	**Attack**
Horizontal		
speed	5.0 m/s	8.3 m/s
acceleration	1.5 m/s^2	3.0 m/s^2
Vertical		
Ascend		
speed	2.0 m/s	2 m/s
acceleration	3.0 m/s^2	3.0 m/s^2
Descend		
speed	2.0 m/s	2 m/s
acceleration	3.0 m/s^2	3.0 m/s^2
Heading		
speed	3.14 rad/s	3.14 rad/s
acceleration	5.0 rad/s^2	5.0 rad/s^2

TABLE 1.1: The dynamic constraints used by the MPC Tracker.

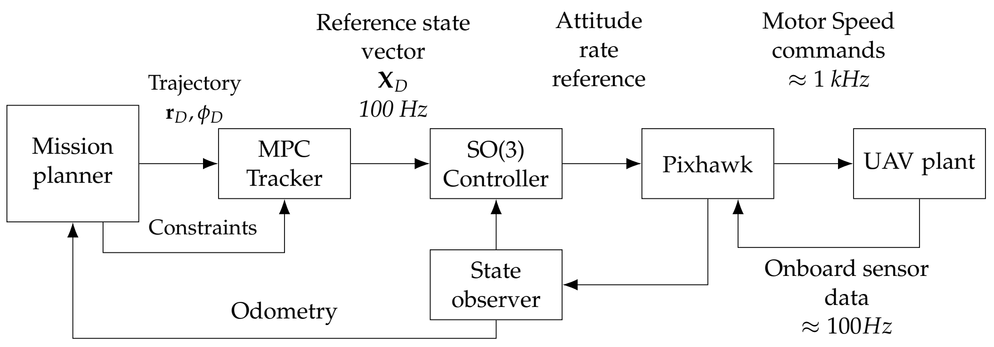

Reference state vector $\mathbf{X}_D$ 100 Hz
Attitude rate reference
Motor Speed commands ≈ 1 kHz
Trajectory $\mathbf{r}_D, \phi_D$
Mission planner
MPC Tracker
SO(3) Controller
Pixhawk
UAV plant
Constraints
State observer
Odometry
Onboard sensor data ≈ 100Hz

FIGURE 1.5: MRS control pipeline taken from [2]

The result of MPC tracker is the control signals which are required to be supplied at $\approx 100Hz$, and send to the SO(3) as shown in Figure 1.5. For controlling the Electronic Speed Controllers (ESC) of the UAV a open-source hardware platform Pixhawk [15] was used.

Feedback from the Pixhawk onboard sensors such like Global Position System (GPS), as well as Inertial measurement unit (IMU) are received and processed to get accurate odometry [16], for localization purposes, the data is also send back to the mission planner, so it can correct the UAV position.

Chapter 2

Hardware setup

This chapter provides an overview of the hardware configuration that was used for MBZIRC 2020.

2.1 UAV platform

The MRS CTU group has spend years in development of their own base design for UAV platform. This includes the hardware (frames, motors, sensors, computers and else), and the software (control pipeline, drivers for the sensors).

Most of the hardware that was used on the UAV platform consists of off-the-shelf components that are mounted together using 3D printed parts. The base of the drone is its frame, in this case a Tarot T650 quadrocopter frame, with Tarot motors that are connected to Pixhawk 4 flight controller unit [1] and an Intel NUC [2] onboard computer (see Figure 2.1).

[1] https://docs.px4.io/v1.9.0/en/flight_controller/pixhawk4.html

[2] https://www.intel.com/content/www/us/en/products/boards-kits/nuc/kits/nuc8i7beh.html

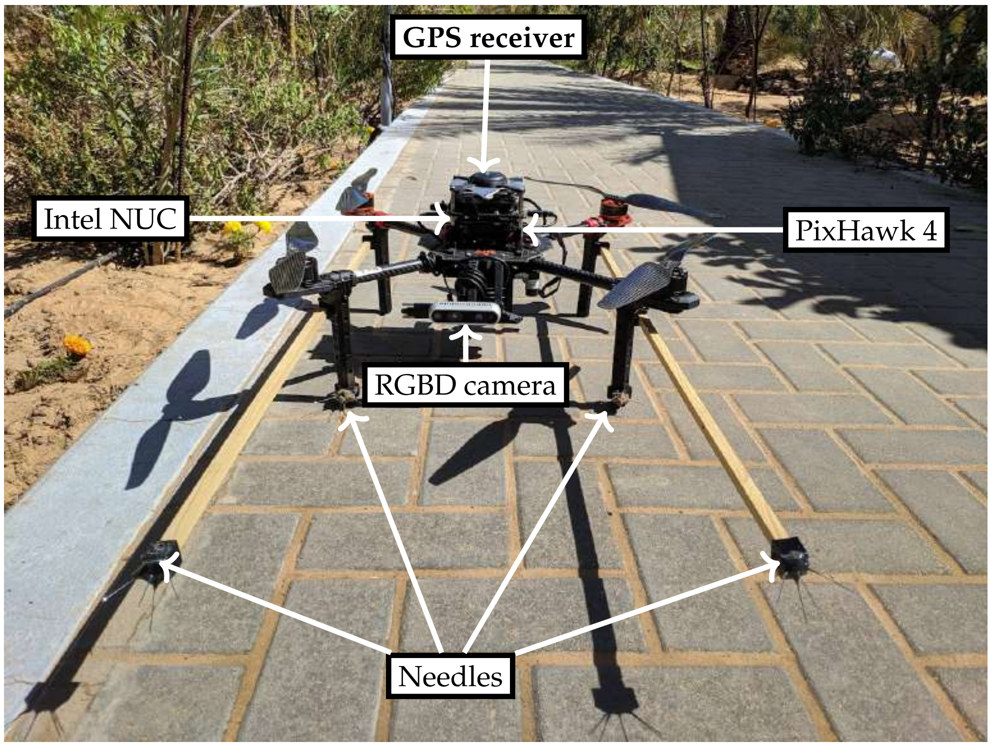


FIGURE 2.1: The UAV platform developed and used by the Multi Robot Systems group at MBZIRC 2020

The Pixhawk 4 itself contains several basic sensors, such as a GPS with magnetometer, gyroscope,an IMU, and a barometer which are used to stabilize and navigate the vehicle. Pixhawk is connected to the motors through ESCs, and driving the brushless motors according to received attitude commands from the control pipeline which runs running on onboard computer.

The computer that was used is an Intel NUC8i7BEH with an Intel i7-8559U CPU, 8 GB of RAM and an onboard integrated GPU Intel Iris Plus Graphics 655. The computer was running the Ubuntu 18.04 LTS operating system.

2.2 Target elimination tools

The specified target for the challenge is a balloon filled with air with diameter of 60cm (see Figure 1.1). The task of the challenge was to destroy all the balloons in the defined arena. Balloon is a soft target and is easily shred with using propellers. However, this elimination method is problematic due to the possibility of the balloon remains getting entangled in the UAV causing a crash. During testing of the popping method we have encountered this situation several times (see Figure 2.2).

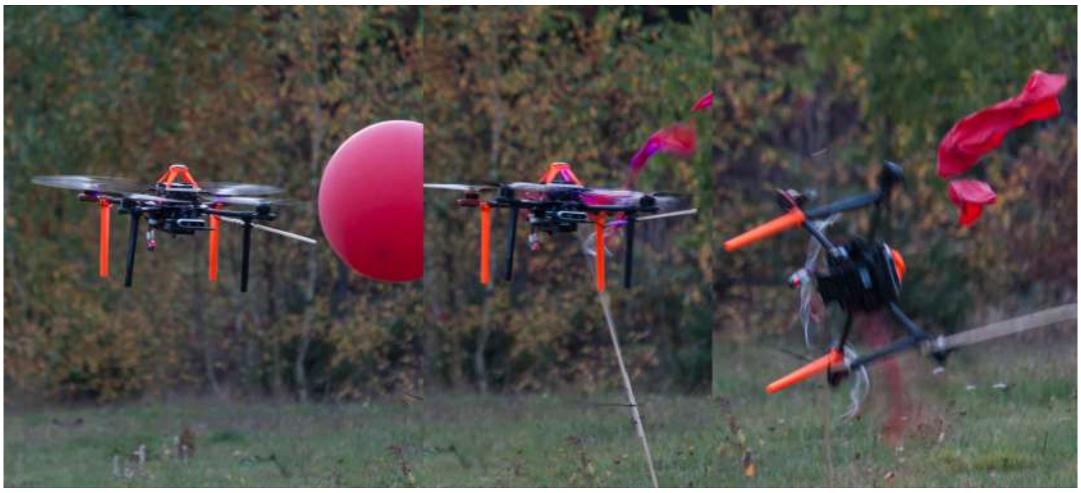


FIGURE 2.2: Drone is falling due to the balloon shred parts got stuck in the motor.

After experiencing a couple of crashes, two wooden poles were added to the drone , and 3D printed holders (see figure 2.3b) with attached needles on them (see Figure 2.1). Due to a restriction of the maximal size of the drone (it must fit into 1.2m wide, 1.2m long and 0.5m box),[3] the poles were **100cm** long and the needles were ≈ **3.5cm**. Because the poles were going through the whole drone, the point of interaction was moved **30cm** forward from the drone props. Besides the poles, needles were also added on the front legs of the drone (figure 2.3a), to ensure that even if the of the target height was estimated wrong, the drone will destroy the target anyway.

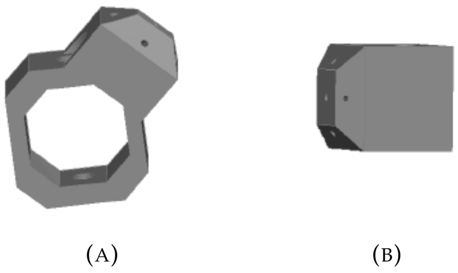


FIGURE 2.3: 3D model of needle holder for the legs (A), and for the poles (B)

This helped a lot in target elimination, but in rare occasions crash was still possible. For example when the needles got stuck in the balloon surface (see Figure 2.4 and video of the fall `https://youtu.be/keHxKxgUdtA`).

[3] `https://www.mbzirc.com/challenge/2020`

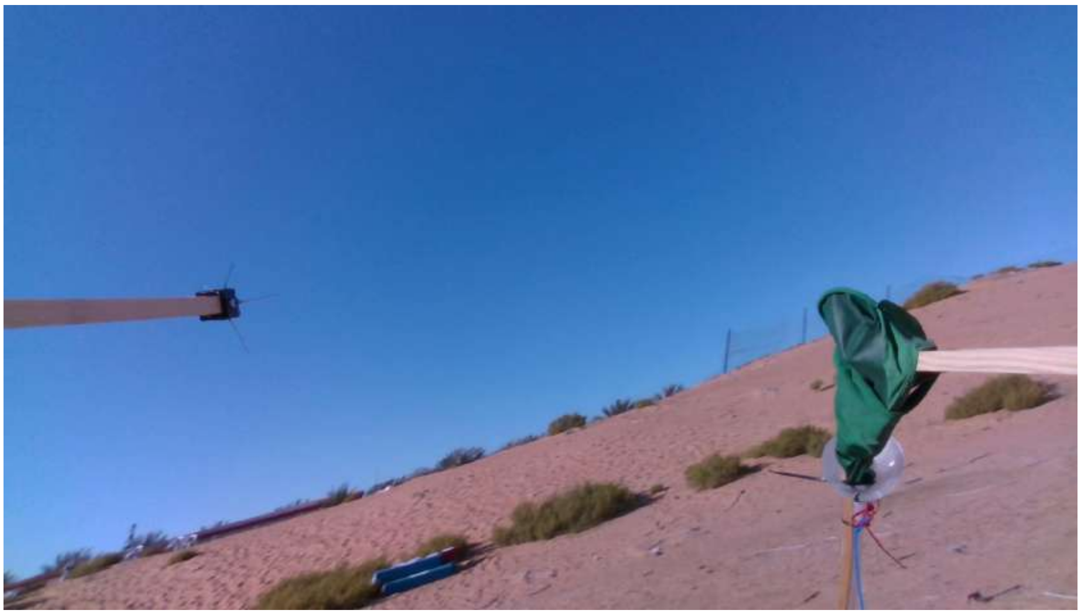


FIGURE 2.4: A photo from the drone onboard camera, when the balloon got entangled around the pole with needles.

2.3 Camera

Intel RealSense D435[4] (see Figure 2.1) camera was used to provide color and depth images for detection of the targets. It provides image with 1920**x**1080 resolution (expressed in pixels) with 69.4° **x** 42.5° **x** as Horizontal **x** Vertical correspondingly field of view (FOV). The color image has 1920x1080 (in pixels) resolution and is supplied at maximum of 30 frames per second (FPS) in this configuration. This stereo camera also supplies depth image with 87.4° **x** 58° FOV which is supplied in 848**x**480 resolution at 30 FPS.

This camera was extensively used in different environments, especially in desert areas. All the tests were done a month before the competition in an area near Abu-Dhabi, and the camera has shown good resistance against the heat and sand, but it has proven to have reliability issues, due to which the camera could turn off in the middle of the flight. Due to that, an abort system was built into the state machine to track if it is working and abort the mission in case of camera failure (subsection 3.5.3).

[4]`https://www.intelrealsense.com/depth-camera-d435/`

Chapter 3

High level state machine

This chapter provides a detailed description of the state machine. The state machine controlled the balloon popping mission based on data from various modules of the system (see Figure 3.1). Output of the state machine is a setpoint trajectory for the UAV control pipeline (described in section 1.3).

3.1 State machine pipeline

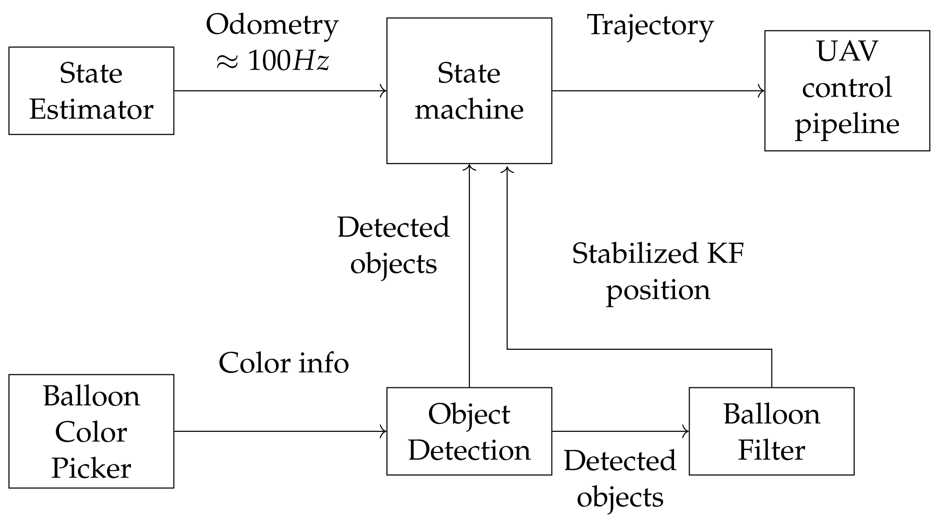


FIGURE 3.1: State machine pipeline.

- **Object Detect** - is a computer vision package developed at MRS[1] to perform detection of circular objects for MBZIRC 2020 competition. It is described in chapter 4.
- **Balloon Filter** - is an implementation of a linear Kalman filter, which is used to track selected target acquired from data given from **Object Detect** and estimate its position. Its basic algorithm is described in chapter 4.
- **Balloon Color Picker** - this module is used for assistive selection valid color range of the target for the competition. Its implementation is described in more chapter 4.

The state machine also receives data from the partner UAV and how the mission is going. If the partner drone stopped transmitting information, the state machine

[1]http://mrs.felk.cvut.cz/

will adapt to the changes and expand the mission (see subsection 3.5.4). The state machine decides whether to proceed with mission or abort because of sensor failure (see subsection 3.5.3). However, the primary motivation of the state machine is to execute the balloon popping without getting stuck in one of the states. For this matter, the state machine also controls the time of each state. If it is too long, the state machine will restart the mission and mark the selected target as a forbidden zone (see subsection 3.5.2).

The state machine is started right after takeoff by automatic start. Automatic start is a module that checks UAV state before takeoff and triggers the takeoff if all sensors are working, and the drone is ready (see section 3.6).

3.2 Communication between UAVs

To manage tasks between UAVs, reliable communication is required. The rules of the MBZIRC competition specified that all teams are given a separate powerful 5GHz dedicated Wi-Fi network. Thus, we could rely on a communication channel between several UAVs. However, it also adds a vulnerable point into the solution—a method to deal with a situation when a drone loses the communication link needed to be implemented. The proposed solution is described in subsection 3.5.4.

The software solution responsible for managing the communication between UAVs is NimBro communication solution [18]. It is a ROS package that allows to transport specified messages with data over IP between computers with different instances of the ROS.

To reduce the load of the communication in the given network, only selected topics were exchanged between the UAVs:

Odometry - actual position of the UAV in the **gps origin** frame to enable collision avoidance [2] (see [2] for a description of the collision avoidance mechanism).

UAV status - this message contains information about UAV flight status, if it is flying normally. This message is the most important for the state machine. The logic of processing this message is described in subsection 3.5.4.

3.3 State Machine Parameters

All parameters of the state machine are in Table 3.1, and core variables in Table 3.2.

Symbol	Value	Meaning
H	5 m	Searching height.
v	3 m	Default velocity for trajectory generation.
v_a	4 m	Velocity for attack trajectory generation .
v_s	3 m	Velocity for searching trajectory generation .
a_{max}	2 m	Maximal acceleration.
d_b	4 m	Distance to the balloon, where the UAV should wait.
d_o	5 m	Overshoot distance after popping the balloon.
d_e	3 m	Detection error distance threshold, in meters.
d_a	3 m	Accuracy threshold of position when the drone should be waiting.
D_f	0.5 m	Dead band factor for dealing with positive feedback from detections.
D_p	11 m	Distance when the UAV start descending to target height.
D_k	5.5 m	Distance when the UAV should switch from tracking the target from PCL to output from balloon filter.
o_h	0.3 m	Height offset when executing the attack trajectory.
o_o	0.15 m	Height offset at the end of the overshoot after popping.
t_f	15 s	Time of existence of a forbidden area, after it will be deleted, and the drone will be able to try destroy target again.
t_m	3 s	Maximal time of not detecting a selected target.
t_w	4 s	Maximal waiting time for receiving data from the balloon filter.
t_s	15 s	Maximal time for waiting to receive message from another UAV.
T_c	30 s	Maximal time of being in the **CHECKING BALLOON** state.
T_g	30 s	Maximal time of being in the **GOING TO BALLOON** state.
T_d	20 s	Maximal time of being in the **DESTROYING** state.
r_f	4.5 m	Radius of a forbidden area.
n_f	2 m	Number of tries to approach a target before adding it to the forbidden list.

TABLE 3.1: Parameters of the state machine.

Name	**Dimension**	**Meaning**
b_c	$\mathbb{R}^{3\times1}$	Closest target position.
b_p	$\mathbb{R}^{3\times1}$	Current target position obtained from the ebject detector.
b_k	$\mathbb{R}^{3\times1}$	Current target position obtained from the balloon position filter.

TABLE 3.2: Core variables of the state machine.

3.4 State Machine Flowchart

There are 5 states:

- **IDLE** - This is the initial state of the state machine.
- **GOING AROUND** - in this state the arena is scanned for suitable targets.
- **CHECKING BALLOON** - after finding a target, this is the next state. Its purpose is to check whether the detection is a false-positive. This state may be actiavted during every other state of the algorithm. It is made to avoiding switching to the **DESTROYING** state, when the target is a false detection.
- **GOING TO BALLOON** - after passing the checks in the **CHECKING BALLOON** state, trajectory is passed to Mission planner [2], to get close to the balloon at a given distance in the parameters.
- **DESTROYING** - this state controls to last step of the algorithm, destroying the target.

3.4.1 IDLE state

This state is triggered on by Autostart (see section 3.6) once the UAV is in the air.

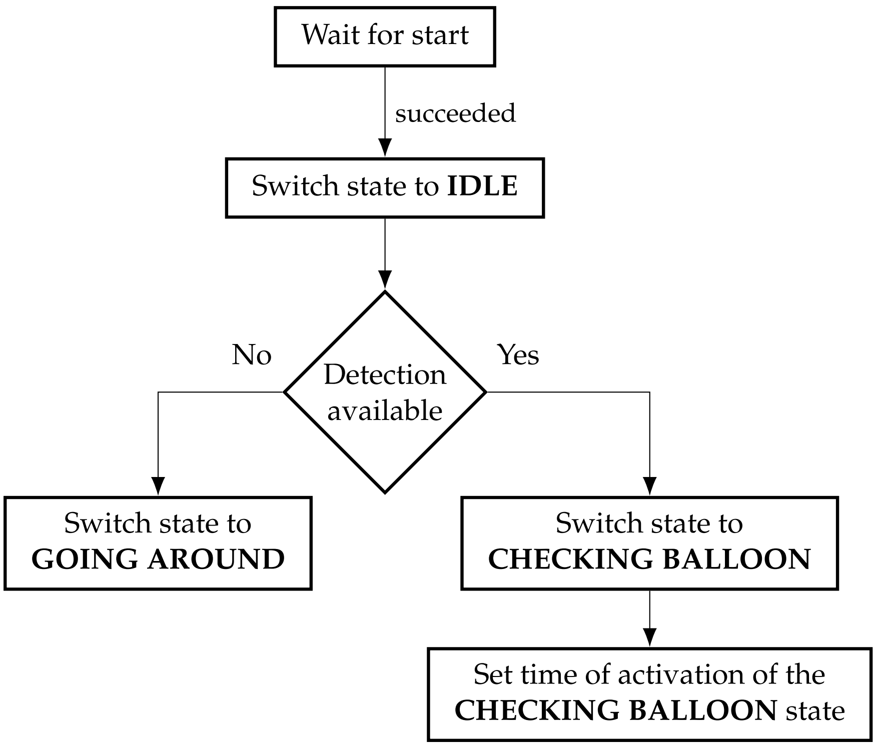

Wait for start
succeeded
Switch state to IDLE
No
Detection available
Yes
Switch state to GOING AROUND
Switch state to CHECKING BALLOON
Set time of activation of the CHECKING BALLOON state

FIGURE 3.2: **IDLE** state diagram.

In Figure 3.2, flowchart of the **IDLE** state is visualized. Once the state machine is started, there is one choice that is made in the state - which state should be triggered next. If there are some detections from **Object Detect** that are in drone's operational zone (described in subsection 3.5.4), then the **CHECKING BALLOON** state is activated, and an activation timer for the Jobs manager (section 3.5) is enabled. Otherwise the **GOING AROUND** state is triggered to perform a search of the arena.

3.4.2 GOING AROUND state

This state performs scan of the arena. Its workflow is shown in Figure 3.3. The scanning is described in section 5.3. Before starting the scan, Sweeping constraints are activated. This ensures, that the drone camera won't be looking downwards, because of aggressive pitching of the UAV. At the same time the turns should be performed with aggressive yawing, since the turns are at the end of the arena and there is no reason to look at the net.

If a target is detected during this state a stopping manoeuvre commanded. The stopping procedure is here due to the possible false-detections that may occur with sudden change of light. After the drone is stopped, the state is changed to **CHECKING BALLOON**. After switching to the **CHECKING BALLOON** state, a state timer is set, (its usage is described in section 3.5).

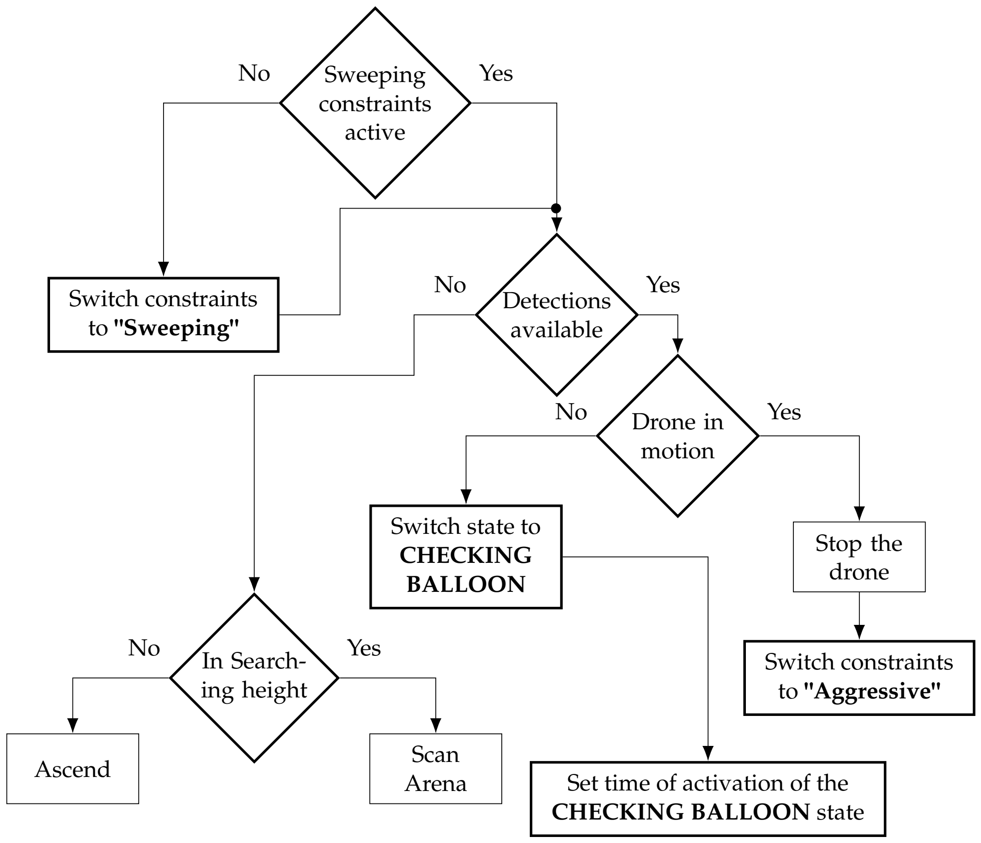

Sweeping constraints active
No
Yes
Switch constraints to "Sweeping"
Detections available
No
Yes
Drone in motion
No
Yes
Switch state to CHECKING BALLOON
Stop the drone
Switch constraints to "Aggressive"
In Searching height
No
Yes
Ascend
Scan Arena
Set time of activation of the CHECKING BALLOON state

FIGURE 3.3: **GOING AROUND** state diagram.

Otherwise, if there is no available detection, before performing the scan, the UAV moves to be above the searching height H . This is to avoid the poles to which the targets are tethered, (see in Figure 1.1). The height of the poles is unknown and according to the description of the challenge it can vary up to 5 meters (since target maximal height is 5 meters). H was set to 5.5 meters to avoid the possibility of collisions. The ascend command is send to the mission planner. Once the UAV altitude is higher than H , the scan procedure is activated, and a scanning trajectory is sent to the mission planner. The scan trajectory is sent repeatedly every time it is finished, to ensure continuous scanning of the arena.

3.4.3 CHECKING BALLOON state

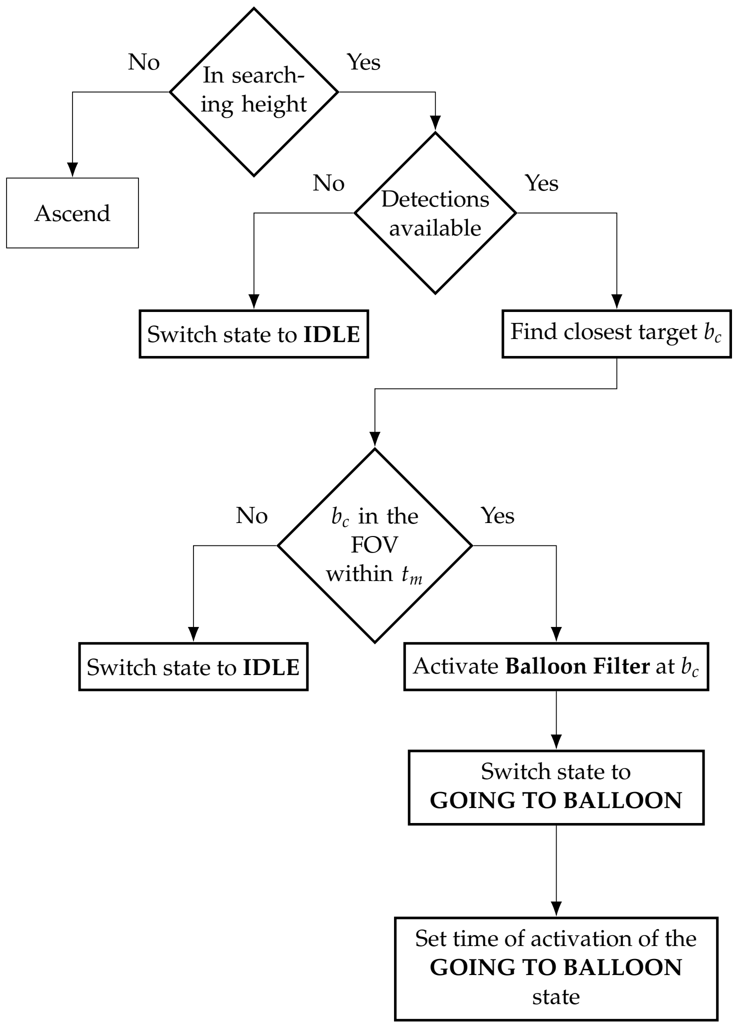

No
In searching height
Yes
Ascend
No
Detections available
Yes
Switch state to **IDLE**
Find closest target b_c
No
b_c in the FOV within t_m
Yes
Switch state to **IDLE**
Activate **Balloon Filter** at b_c
Switch state to **GOING TO BALLOON**
Set time of activation of the **GOING TO BALLOON** state

FIGURE 3.4: **CHECKING BALLOON** state diagram.

If the height is correct, a check for available detections is made. This is done to avoid any false-positive detections, artifacts in the camera, etc. If there are detections available , the closest target to the drone's current position is selected.

Since the UAV is a non-stationary vehicle, its stopping movement can cause a great tilt, which may cause that in one moment the target is in the camera FOV and next frame it is completely out. The stat is reset to **IDLE** only if the target is not detected for a t_m duration. This added flexibility to the system, in situation when drone has seen the target, but stopped couple meters ahead of it. It is shown in chapter 6.

Procedure of activation of the **Balloon Filter** module is described in section 4.2. This state does not check whether the filter is working or not, it is done later in the **GOING TO THE BALLOON** (subsection 3.4.4) state. After activating the **GOING TO THE BALLOON** state the corresponding execution timer is started for the Job manager (see section 3.5).

3.4.4 GOING TO THE BALLOON state

This state to commands the drone to approach the balloon. Its workflow is described in Figure 3.5.

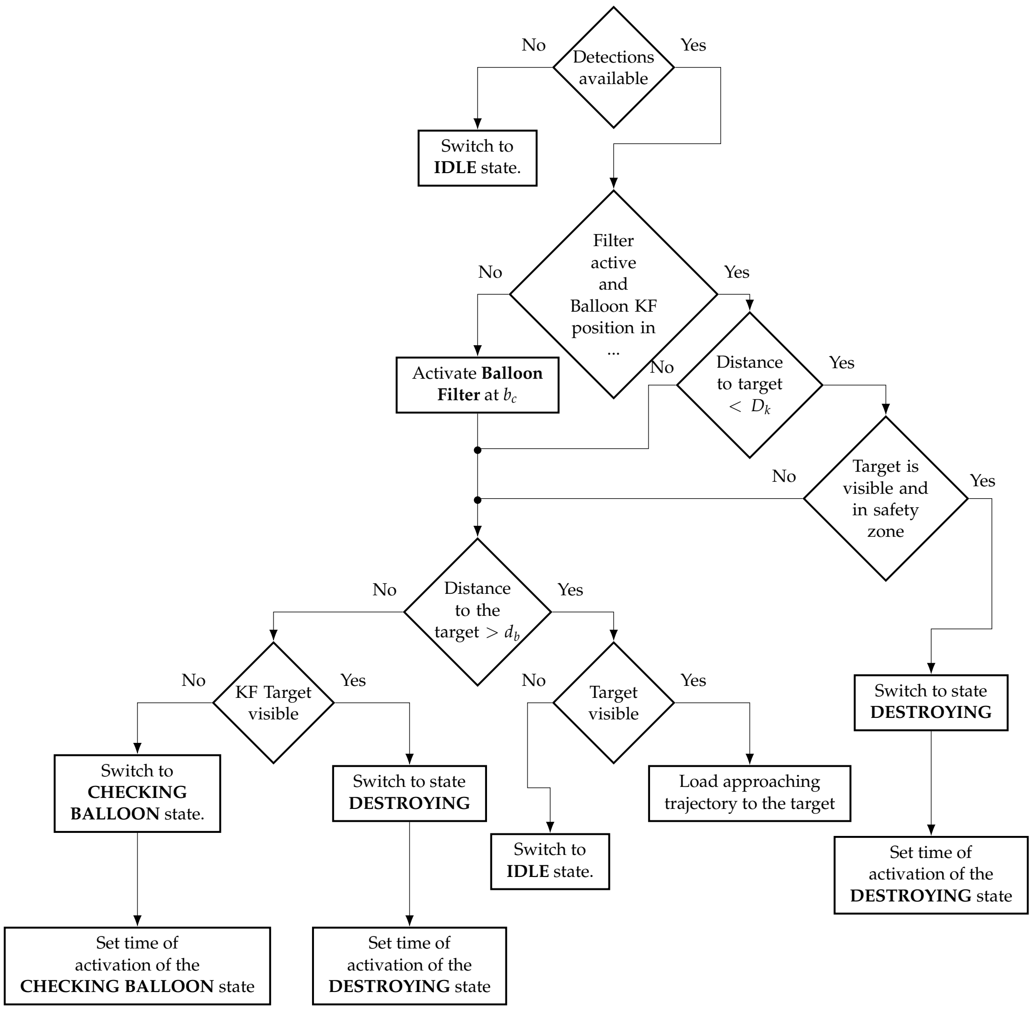

No
Detections available
Yes
Switch to **IDLE** state.
No
Filter active and Balloon KF position in ...
Yes
Activate **Balloon Filter** at b_c
No
Distance to target $< D_k$
Yes
No
Target is visible and in safety zone
Yes
No
Distance to the target $> d_b$
Yes
No
KF Target visible
Yes
No
Target visible
Yes
Switch to state **DESTROYING**
Switch to **CHECKING BALLOON** state.
Switch to state **DESTROYING**
Load approaching trajectory to the target
Switch to **IDLE** state.
Set time of activation of the **DESTROYING** state
Set time of activation of the **CHECKING BALLOON** state
Set time of activation of the **DESTROYING** state

FIGURE 3.5: **GOING TO BALLOON** state diagram.

If there are any detections, the next step is to check whether the **Balloon Filter** is working, and if it is publishing the stabilized position of the selected target. If this is satisfied, next step is made for faster switch to destroying state, if the distance to target is less then D_k (Table 3.1). This part makes the transition between two states more smooth, since destroying state has very aggressive constraints.

If the filter is not working, or the distance is still bigger then D_k, the activation procedure of the filter is repeated. The approaching trajectory is loaded only if the target is visible (within t_m). If the target is visible the selected target is updated, and an approaching trajectory is loaded.

If the target is close enough, the visibility check is executed again. The difference is that it will check whether the target position obtained from Balloon Filter is visible or not. If the target is visible, the **DESTROYING** state is turned on. If not, the **CHECKING BALLOON** state is turned on to select the target again. After activating the **DESTROYING** or **CHECKING BALLOON** state the corresponding execution timer is started for the Job manager (see section 3.5).

3.4.5 DESTROYING state

This state is visualized in Figure 3.6. When this state is active, the trajectories are generated with the reference obtained from **Balloon Filter**. Before sending attacking trajectories, the target position is checked if it is in the list of targets from **Object Detect** for a t_m time, then the attacking maneuver is sent.

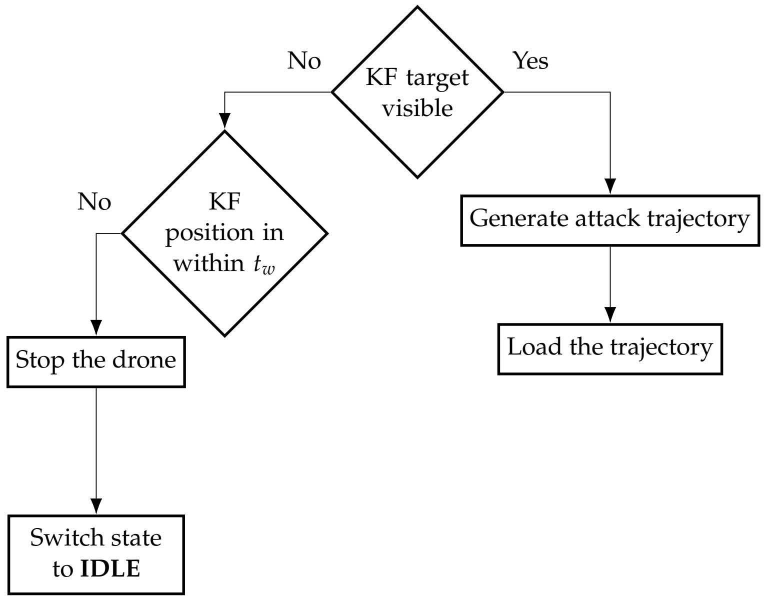


FIGURE 3.6: **DESTROYING** state diagram.

The detections are incoming at approximately 30 Hz, the same is for the stabilized position, so the trajectories are sent nearly at the same rate. The continuous loading of the trajectory and attack constraints ensure that the drone is able to adapt to a sudden change of the target position, but only if UAV is more than 0.5 meters away from the target, else the Balloon Filter won't have enough number of detections and the position won't be updated.

3.5 Job manager

This part describes parts of the algorithm which were running beside the state machine in parallel. These routines may interrupt the state machine and change its procedures.

3.5.1 State timer

It is important that UAV does not get stuck in one of the states, to prevent deadlocks and improve robustness. Every state has a maximal time of execution - T_c, T_g, T_d . Every time one of the states is switched on, timer is set (as is shown in Figure 3.2, Figure 3.3, Figure 3.4, Figure 3.5), and if the timer exceeds its limit, the current selected target b_c is marked as a forbidden zone, and state is switched to **IDLE**.

3.5.2 Forbidden zones

Every detection that comes from the Object Detect module goes through a filter. Detections that are out of the drone designated work area are filtered out, as well as detections that are in a forbidden zone. The forbidden area is a sphere characterized by a position vector $[x, y, z]$, a radius r_f and a time of life t_f. Once the execution timer of the state exceeds its limit it creates a forbidden zone, and after the t_f is exceeded it is deleted. This gives flexibility to the state machine, so the UAV can skip the problematic target (because it took too much time to eliminate it, or it was a false-positive) and search and focus on other ones.

3.5.3 Abort system

The state machine is strongly dependent on the computer vision part, since without a camera stream it is literally blind. Due to this fact and because during experiments RealSense proved to have reliability issues an abort system was designed. If, there was no data incoming from the camera for 1 second (in normal mode the camera produced images at $\approx 30Hz$), the Jobs manager initiated for emergency land of the UAV.

3.5.4 Multi-Robot Scenario

MBZIRC 2020 arena has the shape of a non-convex polygon 100 meters long and 60 meters wide. For solving the whole challenge, 3 UAVs can be used at most. To solve the balloon hunting task, two drones were selected. There are different ways how a multi robot scenario could be addressed in this situation: mapping the area and splitting the targets between UAVs like the MRS team did in previous MBZIRC 2017 treasure hunt challenge [20], dynamic TSP, etc. Since there are only 5-6 targets for the whole challenge a greedy solution was chosen.

The solution that was used in this thesis is to split the arena in two equal parts. In Figure 3.7 these two areas are visualized. The red polygon is the working area for UAV1 and the yellow one is the working area for UAV2. All detections coming from Object Detect are filtered out if they are outside of the current working area for the respective of the UAV.

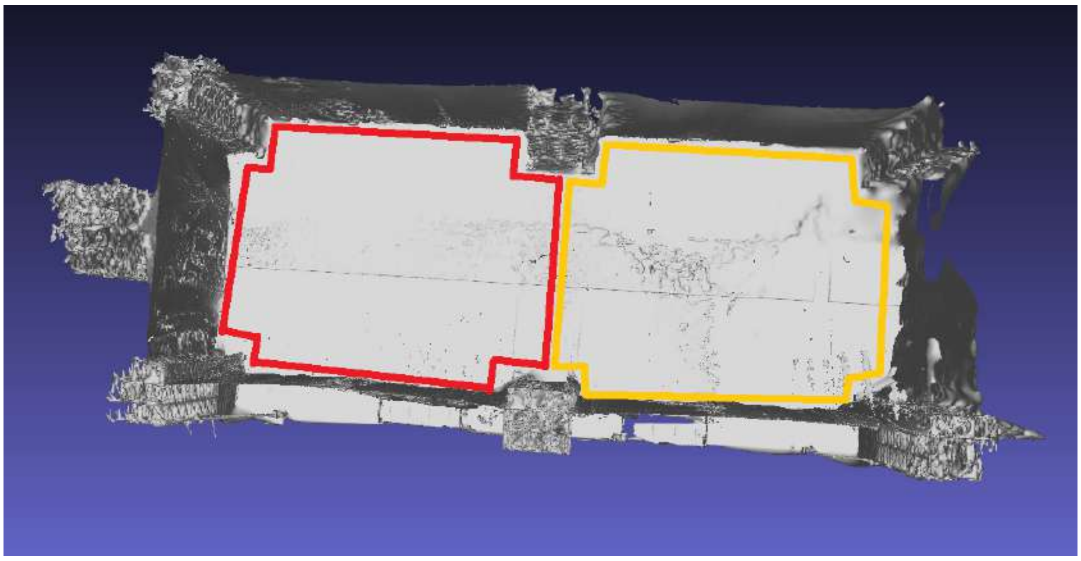


FIGURE 3.7: The top view on the arena 3D model. Working area for UAV1 is highlighted by a red polygon and working area for UAV2 is highlighted by a yellow polygon.

There is a continuous communication link between the drones, as described in section 3.2. The UAV status is checked by the Jobs manager and processed as shown in Figure 3.8.

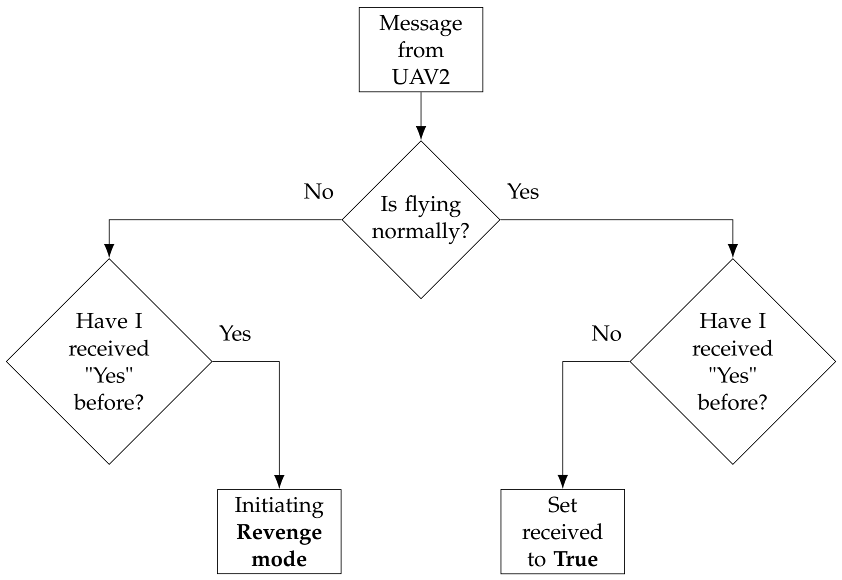

Message from UAV2
Is flying normally?
No
Yes
Have I received "Yes" before?
Yes
Initiating **Revenge mode**
Have I received "Yes" before?
No
Set received to **True**

FIGURE 3.8: UAV status message processing.

Initially the parameter **"flying normally"** is **False**, only after UAV takeoff the parameter is changed to **True**. So, a drone failure situation is detected after the takeoff succeeded.

The **Revenge mode** is a situation when another drone has landed due to some reason (can be a camera failure or a safety reason done by the MRS control pipeline), and the drone that received the message needs to expand its working area to the whole arena as shown in Figure 3.9. This mode can also be triggered if one of the drones is not communicating. Due to the reliability of Wi-Fi communication in the arena, the drones were able to communicate with each other through the whole area. Taking this fact into consideration, we can assume that if a drone isn't communicating, it crashed, and the computer turned off. There is a waiting period before activating this mode if no message has been received for t_s, the Jobs manager changes the working is to the whole arena.

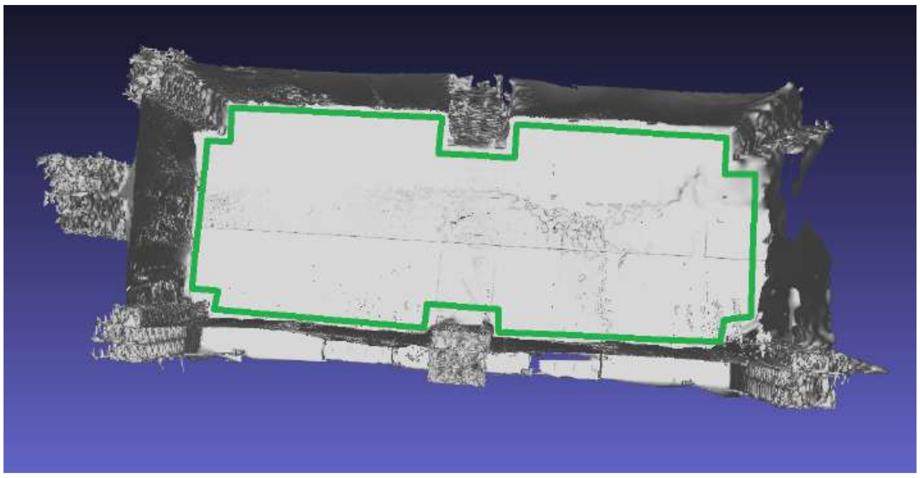


FIGURE 3.9: A top view of the arena. The big working area is highlighted by polygon of green color.

3.6 Automatic start

The state machine can only be started when the UAV is in the air, and it does not control takeoff and landing. To manage this, the **Automatic Start** ROS package from the MRS group was used. The control of takeoff and landing is done in the control pipeline [2], and the **Automatic start** triggers the start of takeoff and landing in the control pipeline which then proceeds. Once the takeoff is done, and the drone is in the air, **Automatic start** starts the state machine, which continues with the task.

After the takeoff, **Automatic start** launches a timer, and when this time expires, it will call for a landing. Using this, there was no need to implement a timer or a finishing procedure in the state machine itself. Usually, the timer was set for 4-5 minutes, but almost every time it was not used since the drones finished the task faster (this is shown in chapter 6).

Chapter 4

Target detection and Estimation

This chapter provides an overview of **Object Detect**, **Balloon Filter**, and description of the **Balloon Color Picker**, which are modules of the system that are related to target detection and estimation of its position.

4.1 Object Detect

The Object Detect is a package developed at the MRS group[1] for detection colorful round objects, like the one that was used as a target for the MBZIRC 2020 Challenge one.

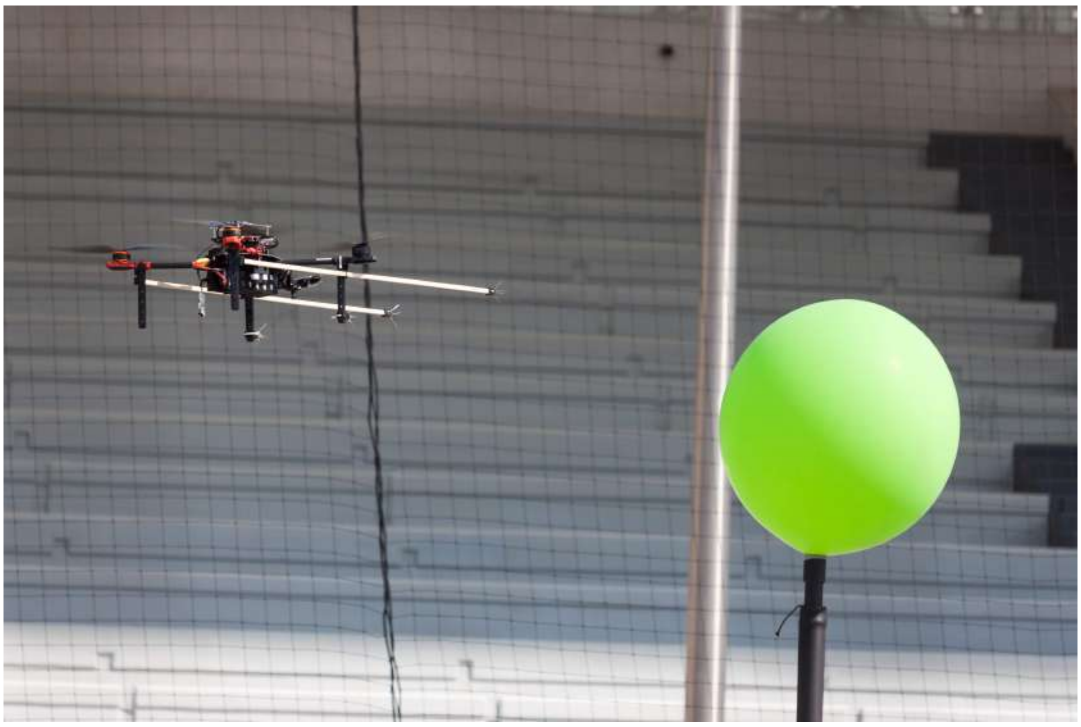


FIGURE 4.1: A Green balloon that served as the target for the balloon sub-task of the first Challenge of MBZIRC 2020.

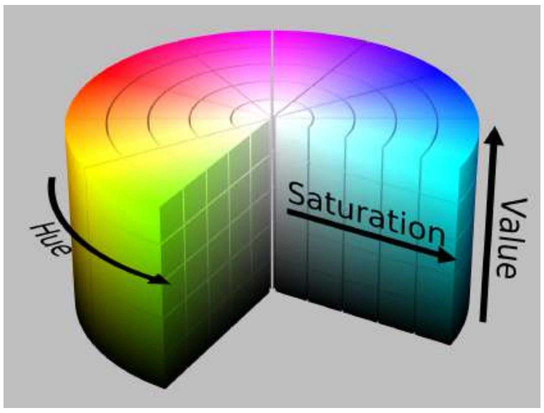
In the MBZIRC 2020 Challenge one target was a green balloon 60cm in diameter as seen in Figure 4.1.

The first stage of the Object Detect algorithm is color segmentation. Image taken by onboard camera (section 2.3) is 1920x1080 pixels size, and each pixel is described by combination of red, green, blue colors - RGB color model. This color model does

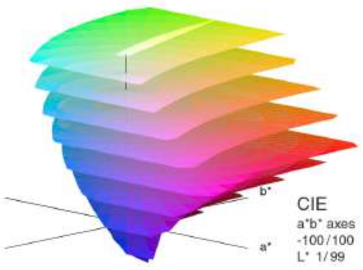
[1]http://mrs.felk.cvut.cz/

not separate hue information from the intensity (lighting), which is a problem for color segmentation, since the hue is one of the main features of the object, and in the case of RGB model it changes together with light. On the other hand there are other color representation models, which are more suitable such as HSV or L*a*b.

The **HSV** color space [11] represents the color as hue, saturation and value. This model is based on a cylindrical representation of the color (see in figure 4.2a), where hue is the angular dimension, saturation starts from the center with a value 0 and in the edges ends with 1, value is a representation of lightness, so in the bottom the cylinder it is darker. The color goes lighter towards the top.



(A)



(B)

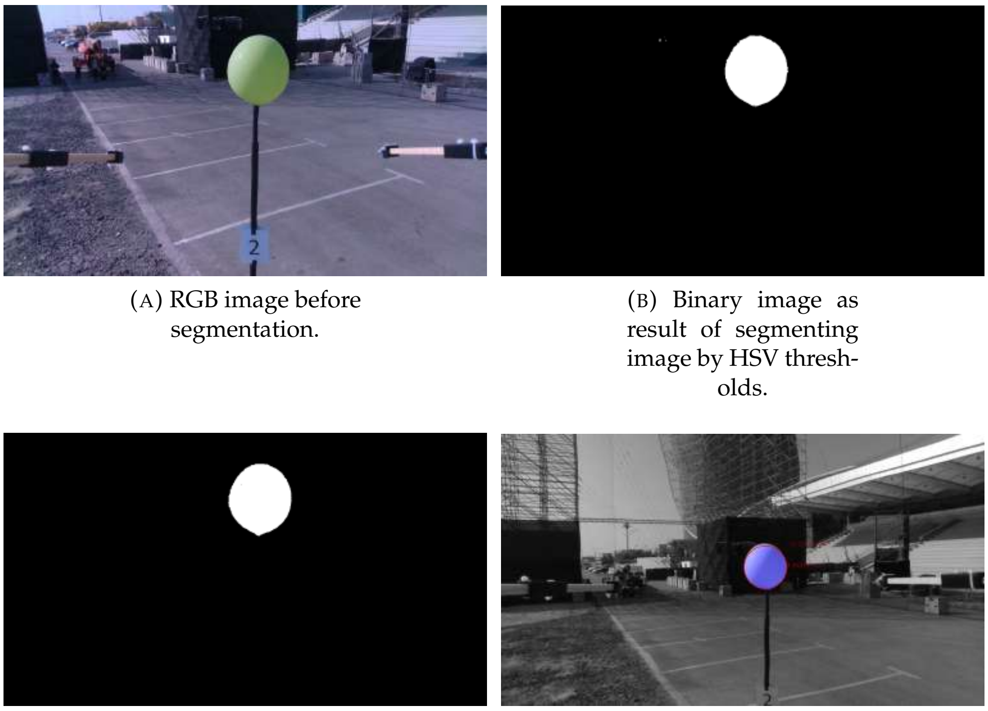
FIGURE 4.2: (A) Representation of the HSV model in a cylindrical shape[2]. (B) Representation of L*a*b space in 3D. The L axis is from 1 to 99, and a*.b* are from -100 to 100, taken from [19].

CIELab [7] (CIE La*b referred to as LAB further) is a color space that is specified by the International Commission on Illumination (from french name Commission internationale de l'eclairage, (CIE)). The motivation behind its creation was to make a model that represents the colors as a human eye perceives them. There are three coordinates in LAB (see figure 4.2b):

- **L** - Lightness of the color, it starts with black at 0, and goes lighter up to 100 where it is white.
- **a*** - represents a transition from red to green, with negative values as green, and positive as red.
- **b*** - represents a transition from blue to yellow, where negative values represent blue, and positive represent yellow.

Using these color spaces Object Detect generates Look Up Table (LUT) from the color ranges obtained from the color picker section 4.3. Incoming image is filtered using the LUT and as a result binary image is created (see Figure 4.3). The next step is circle detection from resulting binary image, after this step knowing size of the object distance is estimated using methods from [21].

[2]https://commons.wikimedia.org/wiki/File:HSV_color_solid_cylinder_saturation_gray.png#filehistory



(A) RGB image before segmentation.

(B) Binary image as result of segmenting image by HSV thresholds.

(C) Binary image as result of segmenting image by LAB thresholds.

(D) Debug image from Object Detect, the detected object is highlighted by a red circle.

FIGURE 4.3: Result of Object Detect image processing.

4.2 Balloon Filter

Balloon Filter is a ROS package developed at the MRS CTU group. It features the implementation of a linear Kalman filter [13] with a mode of a static object in three dimensions. The state machine uses this package to obtain a stabilized position of the target when destroying the balloon.

The output of the filter is the filtered position of one target. Input is a set of 3D positions of detected balloons. Once the state machine has initiated the filter, it uses the closest detection to the initial position, which is within d_a distance.

4.3 Balloon Color Picker

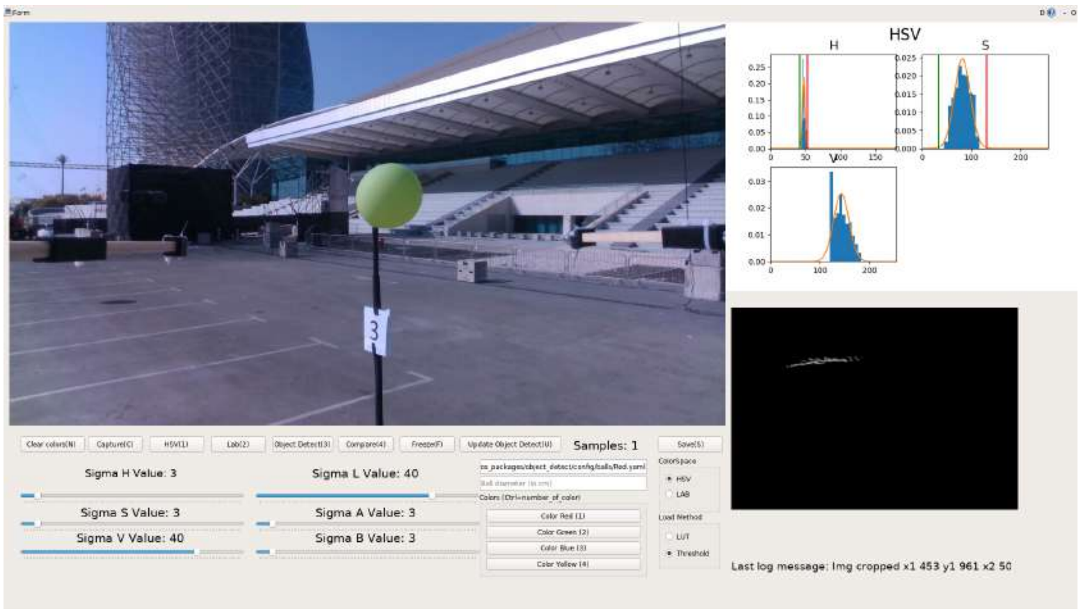
Often desired colors for color segmentation are selected manually by the user. This takes much time and must be done whenever the light conditions are different from the previous tuning. **Balloon Color Picker** (later referred to as the plugin) serves as a semi-automatic tool to select an object and pick its color accurately. The output of the tool is a lookup table (LUT) of valid colors for color segmentation of the object, which may be directly used by the Object Detect package (see section 4.1).

The plugin consists of two separate parts - a graphical user interface (GUI) based on RosQt[3] and a computational package, which performs the color calculations.

[3]`http://wiki.ros.org/rqt`

4.3.1 GUI

Motivation for the design of the plugin is to provide a fast and convenient way of picking the color of a desired object. It should be able to use a video stream from any camera that is running using ROS and output a representation of the picked color. The GUI is shown in Figure 4.4.



Samples: 1

Sigma H Value: 3
Sigma S Value: 3
Sigma V Value: 40
Sigma L Value: 40
Sigma A Value: 3
Sigma B Value: 3

os_packages/object_detect/config/balls/Red.yaml

ColorSpace: [✓ HSV] LAB

Load Method: LUT [✓ Threshold]

Last log message: Img cropped x1 453 y1 961 x2 50

FIGURE 4.4: Balloon color picker plugin GUI.

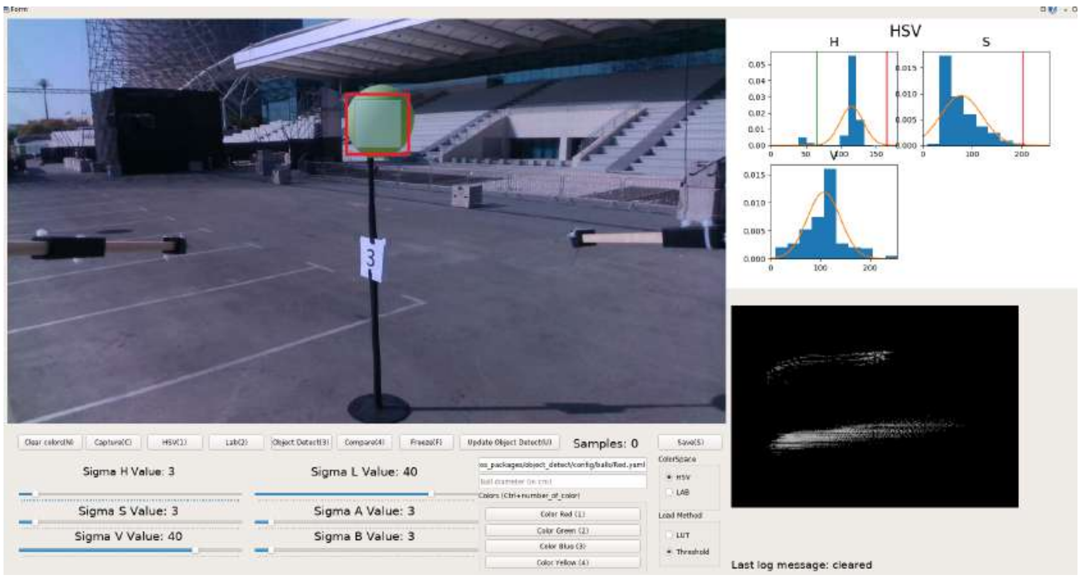
The GUI interface these elements:

1. Video stream - a live video stream from a camera topic and the segmentation result using the selected color values.
2. Clear colors(N) - a button for deleting the current color data. Can also be triggered by pressing **"N"** on the keyboard.
3. HSV(1) - a button that changes current video stream from colorful image to the segmented image by the HSV color space. Can also be triggered by pressing **"1"** on the keyboard.
4. Lab(2) - a button that changes current video stream from colorful image to the segmented image by LAB. Can also be triggered by pressing **"2"** on the keyboard.
5. Object Detect(3) - a button that changes current video stream from colorful image to the debug image from **Object Detect**. Can also be triggered by pressing **"3"** on the keyboard.
6. Freeze(F) - a button that freezes the video stream. Can also be triggered by pressing **"F"** on the keyboard.
7. Update Object Detect(U) - a button that sends the picked color LUT to the **Object Detect** package. Can also be triggered by pressing **"U"** on the keyboard.
8. The current number of taken samples of the color.

9. Save(S) - a button that saves the current picked color LUT to a file. Can also be triggered by pressing **"S"** on the keyboard..
10. A group of sliders for the H, S, V channels respectively. These sliders serve to change the allowed color range in the respective dimensions of the HSV color space..
11. A group of sliders for L,A,B channels correspondingly. These sliders serve to change the allowed color range in the respective dimensions of the LAB color space.
12. Input bar for the destination of the file that will contain the resulting color data.
13. Input bar for the diameter of the tracked object (expressed in centimeters).
14. A list of predefined color naming buttons. By pressing them, user can change the name of the resulting file. Can by also triggered by a combination of **"Ctrl"**+number of the desired color.
15. Buttons that change the current color space view (Figure 4.6).
16. The load method - buttons that specify what type of color data is send to Object Detect.
17. The histogram block - a group of three histograms charts for each channel - H, S, V for HSV view and L, A, B for LAB view.
18. 2D histogram of H,S channels in case of HSV view and A,B channels for LAB view. This area is used to select the color ranges for the LUT.
19. A log message - verbose output of the last action.

Color selection

Main feature of the plugin is the color selection functionality. The desired color is pciked by selecting an area on the image, like it is shown Figure 4.5. The user can draw a rectangle over the image (item 1), using the mouse. The selected region is then send to the computation to calculate the color data.



Samples: 0

Sigma H Value: 3

Sigma S Value: 3

Sigma V Value: 40

Sigma L Value: 40

Sigma A Value: 3

Sigma B Value: 3

os_packages/object_detect/config/balls/Red.yaml

ColorSpace: [✓ HSV] LAB

Load Method: LUT [✓ Threshold]

Last log message: cleared

FIGURE 4.5: The balloon color picker plugin GUI. Process of selecting the color data. The current selection is highlighted by a red rectangle.

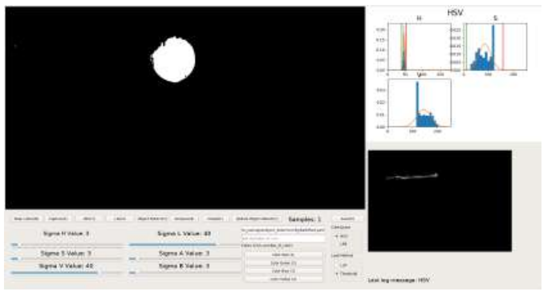
Once the user starts selecting the video stream is frozen and the image does not change until end of the cropping (user releases the mouse). User can also trigger the freeze manually by using the **"Freeze(F)"** button (item 6).

Managing the color range

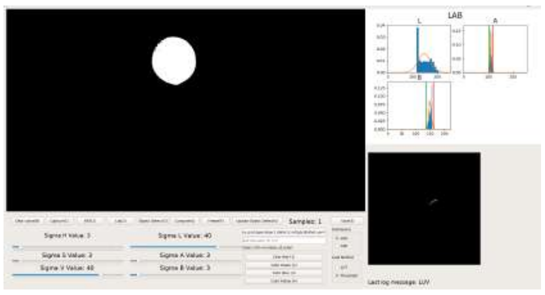
Once the color data has been selected (section 4.3.1), histogram of each channel (H, S, V or L, A, B) is displayed as bar charts on the plots tab item 17. The user can change the range of the computed thresholds using sliders for each channel (item 10, item 10). The effect of changing the value range on the sliders is displayed in the histogram plots (item 17) - green vertical line as the lower range and a red vertical line as the upper level. Users can also switch to HSV, or LAB view (item 3, item 4) that will show how the image can be segmented using the currently selected thresholds. To display how the target detection and position estimation will work, the user can send the color thresholds to the Object Detect package (item 7) and switch to Object Detect view (item 5) that will show a debug image with selected thresholds. All the views are displayed in Figure 4.6

Besides changing the thresholds with sliders, the user can directly select an area on the LUT (item 17). LUT, in this case, is displayed as a 2D histogram of two channels (H, S for HSV and A, B for LAB), the description of its generation is in subsection 4.3.2. Users can select a rectangle region on the histogram and deselect rectangle region, so only the needed part of the histogram is selected. User can change the way the color threshold data is sent to Object Detect, by thresholds or sending directly the generated LUT histogram by using the button "Load Method" (see item 16).

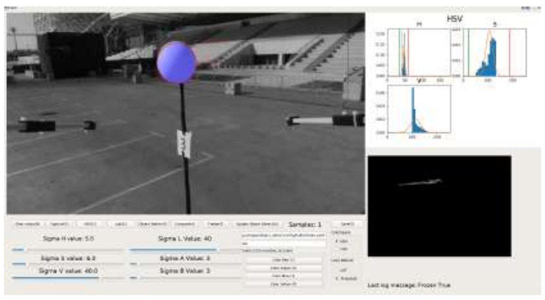
The histogram selection method is used when there is a need for a very accurate color segmentation. Especially when the surroundings may have the same color, but a different shade than the object.



(A) HSV view



(B) LAB view



(C) Object Detect view

FIGURE 4.6: Balloon Color Picker plugin different GUI views.

4.3.2 Computation

Once the user has selected an area of the input image, that part is sent to the computation node. The resulting portion of the image is converted to two color spaces - HSV and LAB. The image data is split into channels. The result is six distributions of the channel data.

Limits of the valid color ranges are calculated using mean and standard deviation estimation of the data. The mean is obtained as

$$\overline{x} = \left(\sum_{i=1}^{n} x_i \right) \frac{1}{n}, \tag{4.1}$$

where x: is one sample of the channels data, n is size of the data. The standard deviation is obtained as

$$\sigma_x = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \overline{x})^2}{n-1}}. \tag{4.2}$$

For the purpose of the limits calculation, a normal distribution of the color data is assumed.

A slightly different approach has to be to calculate the hue channel since it is distributed circularly, as shown in figure 4.2a. Values of the hue have to be treated as angles on a circle. One way to calculate a circular mean is to calculate mean of the cosines and sines of the angles, and then calculate the angle of the resulting vector, using arcus tangens:

$$\overline{x} = \operatorname{atan2}\left((\sum_{i=1}^{n} \sin x_i)\frac{1}{n}, (\sum_{i=1}^{n} \cos x_i)\frac{1}{n} \right). \tag{4.3}$$

The standard deviation of a circular quantity is obtained by:

$$\bar{s} = \sum_{i=1}^{n} \sin x_i, \tag{4.4}$$

$$\bar{c} = \sum_{i=1}^{n} \cos x_i, \tag{4.5}$$

$$\sigma = \sqrt{-2 \log \sqrt{\bar{s^2} + \bar{c^2}}}. \tag{4.6}$$

The histogram charts for every channel of two color spaces are shown in Figure 4.7. The probability density curves drawn over histograms to show the resulting mean and standard deviation. Every histogram was obtained on a green color, the same object that is in Figure 4.6, Figure 4.4, it is clearly seen how the data can be different depending what region of the object you select.

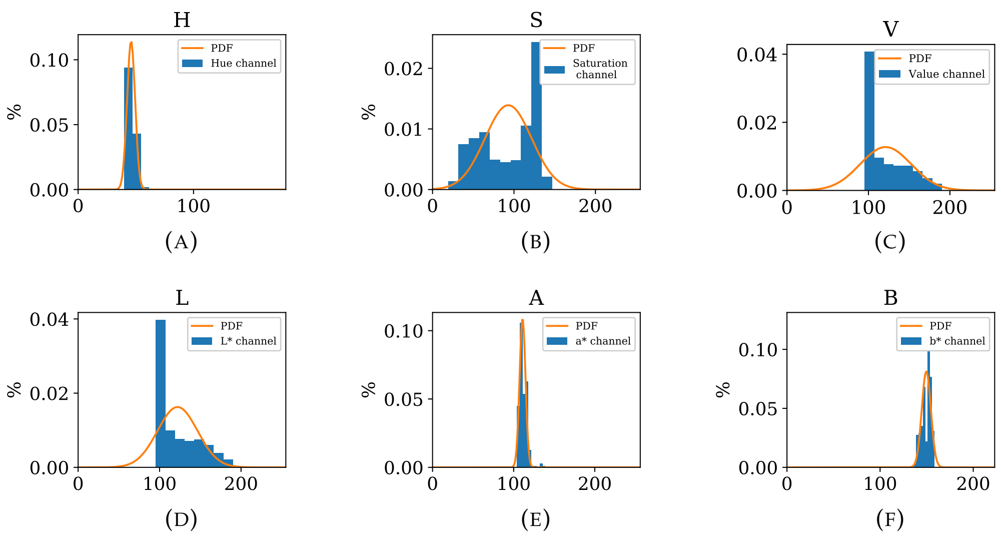


FIGURE 4.7: Histograms of green color for H, S, V ((A), (B), (C)) and for L, a*, b* ((C),(E),(D)) channels.

To get a more accurate representation of color regions,the user can accumulate the selected regions by selecting the object multiple times from different angles.

The accumulated mean is calculated as:

$$\overline{m}_n = \frac{\overline{m}_o + \overline{m}}{n} \tag{4.7}$$

where $\overline{m}_o$ is the mean for the previous selected area and the $\overline{m}$ for the newly selected area. The accumulated standard deviation is obtained as

$$\sigma_n = \sigma_o^2 + \sigma^2 + \overline{m}_o + \overline{m} - 2 \cdot \overline{m}_n \tag{4.8}$$

where σ is the standard deviation for the newly selected area and $_o$ for the old one, and $\overline{m}_n$ is the accumulated one. Only for the Hue channel the mean and standard deviation are calculated the same way as previously.

The thresholds for color segmentation are obtained by subtracting and adding a scaled standard deviation to the mean for each channel

$$lower_c = \overline{m_c} - \sigma_c \cdot multiplier_c \tag{4.9}$$

$$upper_c = \overline{m_c} + \sigma_c \cdot multiplier_c \tag{4.10}$$

where $multiplier_c$ is the value on the slider for channel c that the user uses to increase or decrease the range, see section 4.3.1.

This kind of thresholding creates a rectangular selection in the respective color space (or a cylindrical sector in the case of HSV). In Figure 4.8 it may be observed that the spread of the color data (white dots) is not rectangular. The green rectangle represents the region that was selected by calculating the thresholds and increasing the value of the multiplier. The detection of the object is still very good, but in this case there is not much green color in the surroundings.

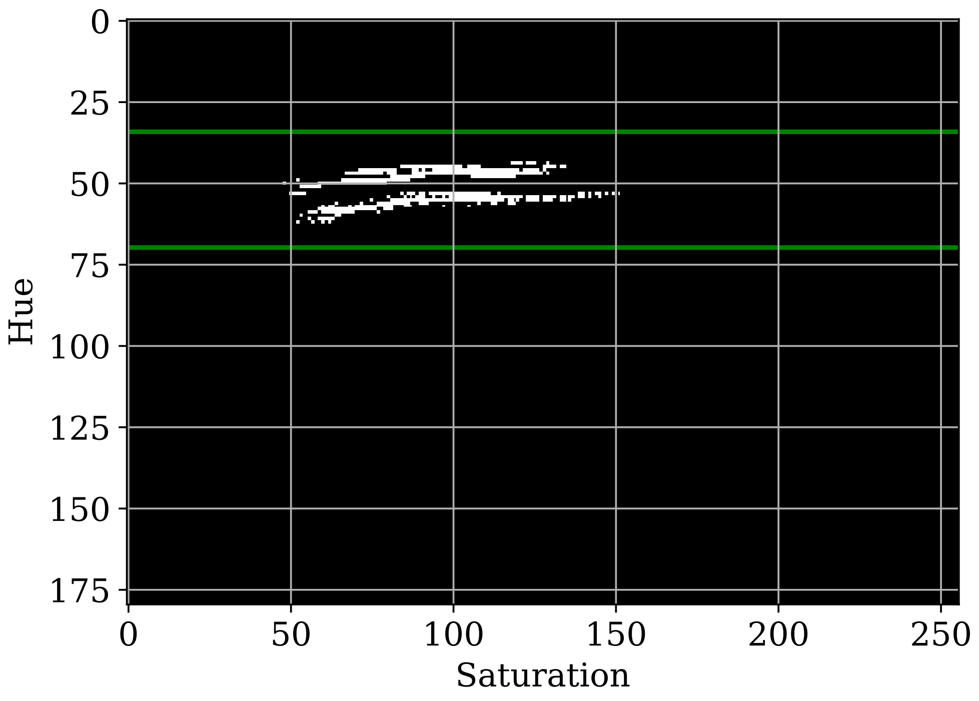


FIGURE 4.8: A 2D histogram of hue and saturation of a green colored object. The green rectangle represents selected region.

For more challenging surroundings (like detecting a yellow ball in a desert), the area on Figure 4.8 can be selected directly to have more accurate representation of the object's color. This is implemented by selecting area on the histogram see item 17.

In this case, the result is a binary mask of the histogram, where the selected color range will be marked as 1, and not used area a 0, this is the proposed version of the LUT. Using this approach, an object may be selected from different angles, resulting in accurate color thresholds.

Chapter 5

Trajectory generation

This chapter describes the generation of approach, attack, and scanning trajectories. These methods are essential for the state machine. At first, the approach trajectory generation, described in section section 5.1, since this is the most complex trajectory in terms of coping with oscillation of the target, and followed by the description of the attack trajectory section 5.2 — finally, the scanning trajectory generation described in section 5.3.

5.1 Approaching trajectory

Thee**Object Detect** package produces estimates of target positions, which one filtered and primary target is selected by the state machine (described in subsection 3.4.3). The resulting reference position

$$\mathbf{b_p} = \begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix} \tag{5.1}$$

is not the final destination of the approach trajectory. The UAV should idle at d_b distance from the balloon. The UAV keeps the target in center of the camera, so the heading is calculated as

$$\phi = \text{atan2}\left(y_p - y_d, x_p - x_d\right), \tag{5.2}$$

where the x_d and y_d are coordinates the UAV's position in the world frame.

5.1.1 Deadband

The detection from the **Object Detect** comes at ≈30 Hz, and the resulting reference has ≈ 10 cm jumps due to detection noise in position of the target. This leads to an unstable reference being sent to the vehicle, which escalates into a positive-feedback and oscillation of the vehicle together with the reference. To prevent the UAV from oscillating a solution called deadband [12] is used.

A basic idea of deadband is to define a neutral zone where UAV does not respond to changes of the reference. In our case, we want to penalize the distance to the balloon but restrict moves around the target.

If the new reference $\vec{r}$ is in range of D_f the reference is updated and projected to a plane, defined by UAV's position and the D_f. The reference projection is obtained using Equation 5.4, where $\vec{t}$ is a result of Equation 5.3, and $\vec{x_d}$ is UAV position.

$$\vec{t} = \left|\vec{x_d} - \vec{b_p}\right| \tag{5.3}$$

$$\vec{p} = (\vec{t} \cdot \vec{t}^{\,T}) \cdot (\vec{r} - \vec{x_d}) \tag{5.4}$$

If $||\vec{p}||$ is smaller than the D_f, the reference needs to be moved to prevent oscillation. The updated reference $\vec{r_n}$ is obtained in Equation 5.6 by projecting it onto the deadband plane via projector $\vec{p_n}$ (Equation 5.5, where I_3 is an identity matrix of size 3 and $\vec{t}$ is normalized vector to the target), the result is shown in Figure 5.1.

$$\vec{p_n} = I_3 - (\vec{t} \cdot \vec{t}^{\,T}) \tag{5.5}$$

$$\vec{r_n} = \vec{p_n} \cdot (\vec{r} - \vec{x_d}) + \vec{x_d} \tag{5.6}$$

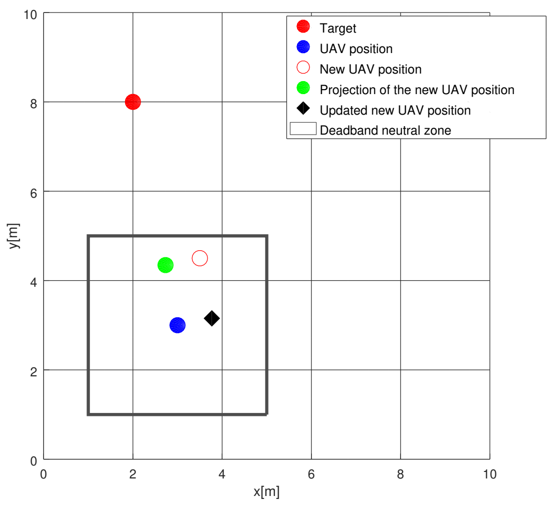


FIGURE 5.1: Illustration of the proposed deadband implementation.

5.1.2 Planning

Since the target is stationary, the approach trajectory can be a straight line. But, the metal poles that hold the balloons may differ in height. The distance between the UAV and the target determines the height of the UAV to the target. If the UAV is further from the goal than D_p, the height is the current one. Otherwise, the height is the target one. There is no need to plan the descend, since we can send the trajectory to the MPC tracker in the UAV control pipeline and it will modify the trajectory to be feasible. The UAV approaches the balloon at distance d_b and wait for next command from state machine. To find this position and plan trajectory to it, algorithm 1 is used. The resulting approach trajectory is visualized in Figure 5.2.

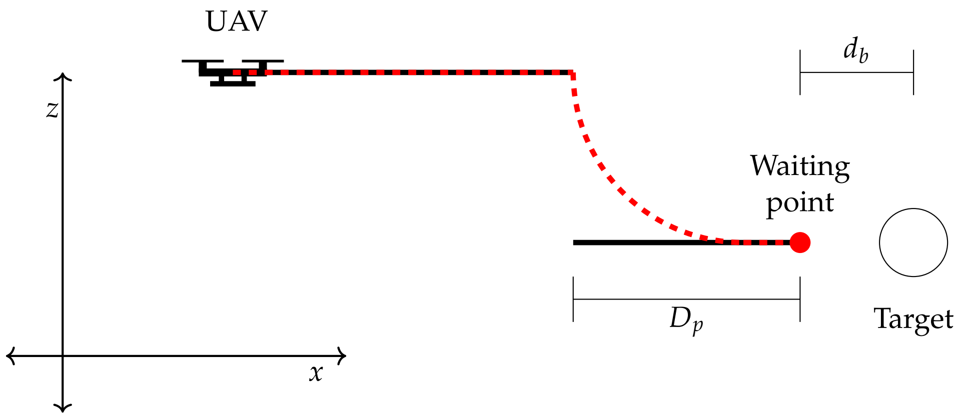


FIGURE 5.2: Approach trajectory planning to a waiting point (red dot) at a distance d_b to the target. Descent to the height of the target starts at D_p from the height H. The solid black line is the generated trajectory; the dashed red one is the actual UAV trajectory, after processing by the MPC tracker.

```
input : $\vec{b_p} = \begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix}$ - position of the target
        $\vec{p} = \begin{bmatrix} x_d \\ y_d \\ z_d \end{bmatrix}$ - UAV position
        $d_b$ - distance from the target where vehicle should be stationary
        $v$ - desired velocity of the resulting trajectory
        $v_d$ - desired velocity of the vehicle
        $a_{max}$ - maximal acceleration of the vehicle
        $\phi$ - heading to the target
output: $traj$ - trajectory to approach the target

begin
    $\vec{dir} := \vec{b_p} - \vec{p}$
    $dist := \left|\left|\vec{dir}\right|\right|$
    $dist := \frac{(dist - d_b)}{dist}$
    $\vec{g} := dist \cdot \vec{dir} + \vec{p}$
    $\vec{g} := deadBand(\vec{g}, \vec{b_p})$
    $\vec{dir} := |\vec{g} - \vec{p}|$
    $traj.append(x_d, y_d, z_d, \phi)$
    $\vec{cur} := \vec{p}$
    $vel := v_d$
    while $\vec{cur} \neq \vec{g}$ do
        $vel := vel + a_{max} * dt$
        if $vel > v$ then
            $vel \leftarrow v$
        end
        $\vec{cur\_dir} := vel * \vec{dir}$
        $\vec{cur} := \vec{cur} + \vec{cur\_dir}$
        $\vec{cur}[2] := z_p$
        if $dist \geq D_p$ then
            $cur[2] := z_d$
        end
        $traj.append(\vec{cur}[0], \vec{cur}[1], \vec{cur}[2], \phi)$
    end
end
```

Algorithm 1: Approach trajectory generation algorithm. Parameters are described in Table 3.1

5.2 Attack trajectory

The attack trajectory planning is similar to the approach trajectory planning (section 5.1). The difference is in speed, distance where the new reference is set and two height offsets. The algorithm 1 is used, but the distance parameter is passed negative d_o, so that the new reference will appear behind the target, and the speed is changed to v_a. The biggest difference are the height offsets. The first one is added to the height of the target - o_h, the second one is added to the last point of the trajectory o_o. The result of this modification is shown in Figure 5.3.

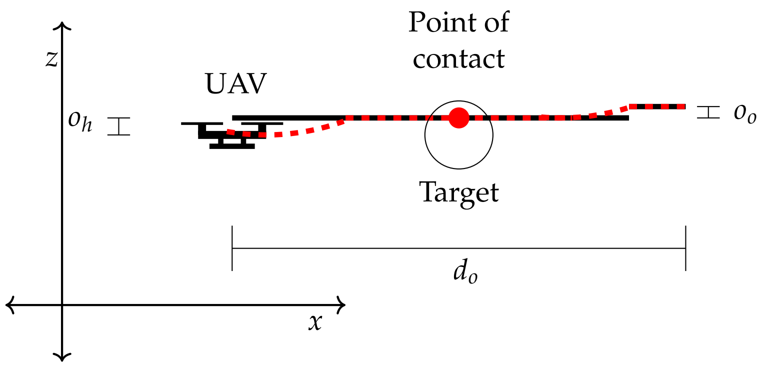


FIGURE 5.3: Attack trajectory planning. Solid black line is the planned trajectory, and the red dashed line is the resulting trajectory after reshaping by the MPC. The red dot is the point of interaction with the target, it has o_h offset from the center. o_o is an overshoot offset added to the last point of the trajectory. d_o is the overshoot distance parameter from Table 3.1.

Those two offsets are serving as a safety precaution since if the height estimation is wrong, the UAV will crash into the metal pole, as shown in Figure 5.4.

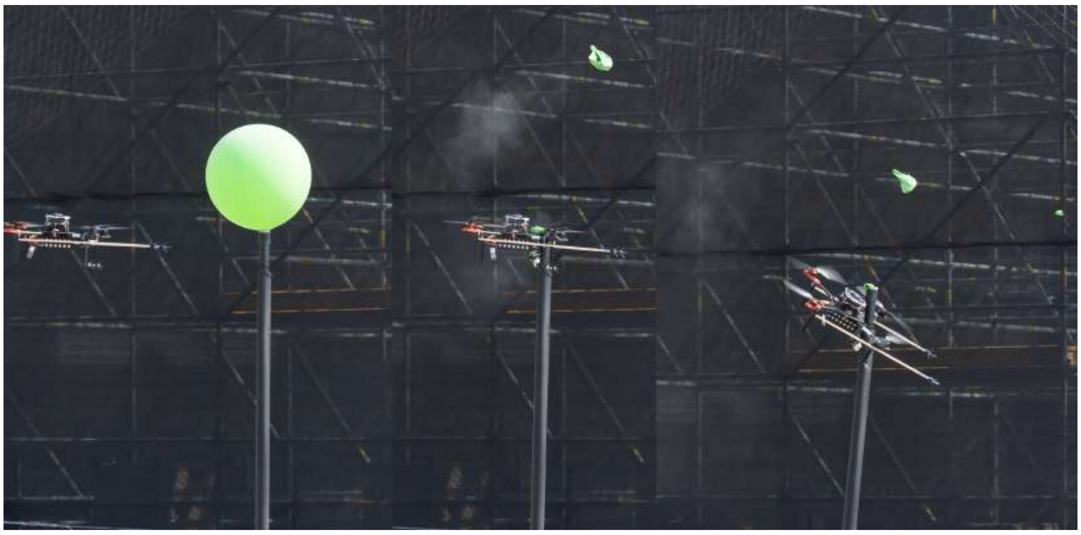


FIGURE 5.4: The UAV crashes into the metal pole because of a wrong estimated height.

5.3 Scan trajectory planning

The arena is shaped as a non-convex polygon (see Figure 1.2). The aim of the scanning trajectory is to go through the whole arena and give the camera as good view coverage as possible. According to this goal, a planning algorithm was designed:

1. Go to one edge of the arena.
2. Move s_{scan} distance the along X-axis.
3. Go to the other edge of the arena.
4. Move s_{scan} distance along the X-axis.

5. Go to step 1 unless the end of the arena is reached.

An illustration of this plan is shown in Figure 5.5. The arrows represent the heading of the UAV in the current step of the plan—almost all the way through the arena, the UAV faces the direction of the flight, except the short turns. During the turns, UAV is heading to the inner part of the arena. This is made to give the camera a view on the inner part of the arena.

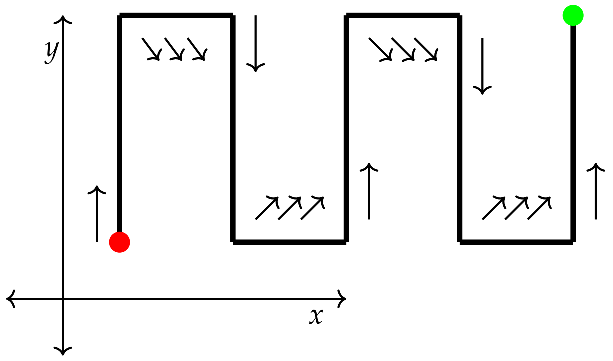
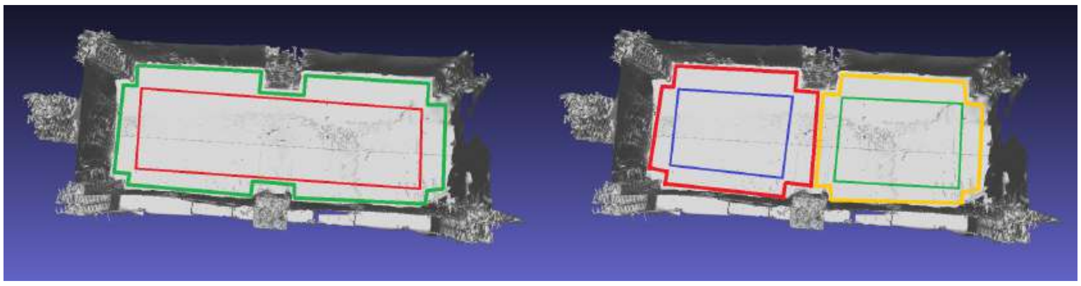


FIGURE 5.5: Illustration of the proposed scanning plan, the red dot is starting point and green one is the finish. The arrows represent heading of the UAV on the move.

Since the arena is a non-convex polygon, its width differs in each point. To overcome this issue and not complicate the algorithm scanning is performed in a rectangular shape area inside of the working arena, as shown in Figure 5.6.



(A) Illustration of a big working area Figure 3.9. The scanning area is highlighted by red rectangle.

(B) Working areas illustration from the Figure 3.7 scanning area is highlighted by green rectangle for UAV1, and scanning area for UAV2 by blue rectangle.

FIGURE 5.6: Illustration of scanning area drawn over 3D model of the arena.

The scan trajectory consists of straight lines, with constant heading and constant speed. The generation of such lines is described in algorithm 2. This algorithm mentions the function *pointInArena*, which checks whether the point is inside the scanning area so that the trajectory does not get out from the scanning rectangle. The algorithm also works when If the starting or finishing point is outside of the scanning area, the result will contain only part of the trajectory that fits into the rectangle.

The scan planning algorithm is shown in algorithm 3. The resulting trajectory is an offset from the rectangle, so that if the Global Position System (GPS) drifts the UAV does not hit the net. The algorithm will produces trajectory from the current UAV's position to the furthest side of the arena. To reduce i the tilt of the vehicle while turning on the corners the speed is decreased by c_f. The algorithm also uses $getHeading(\vec{a}, \vec{b})$ function, which calculates the angle between two points according to Equation 5.7.

$$\phi = \text{atan2}\,(y_b - y_a, x_b - x), \tag{5.7}$$

where x_a, y_a and x_b, y_b are coordinates of the starting and finishing positions respectively. The result of algorithm 3 is shown in Figure 5.5

```
input : $\vec{a}$ - starting point of the trajectory
        $\vec{b}$ - finishing point of the trajectory
        $v$ - velocity of the trajectory
        $\phi$ - heading of the trajectory
output: $traj$ - trajectory from $\vec{a}$ to $\vec{b}$

begin
    $\vec{d} := \left|\vec{b} - \vec{a}\right| * v$
    $\vec{c} := \vec{a}$
    if $pointInArena(\vec{c})$ then
        $traj.append(\vec{c}, \phi)$
    end
    while $\vec{c} \neq \vec{b}$ do
        $\vec{c} := \vec{c} + \vec{d}$
        if $pointInArena(\vec{c})$ then
            $traj.append(\vec{c}, \phi)$
        end
    end
end
```

Algorithm 2: Basic trajectory generation.

```
input : $x_{min}, x_{max}, y_{min}, y_{max}$ - limits of the arena
        $f$ - detection range of the camera
        $v_s$ - velocity of scanning trajectory
        $H$ - scanning height
        $d_{offset}$ - offset from the borders of the arena
        $x_d$ - UAV position
        $c_f$ - arena corner speed decrease factor $\vec{p} = \begin{bmatrix} x_d \\ y_d \\ z_d \end{bmatrix}$ - UAV position
output: $traj$ - scanning trajectory
begin
    $left := x_{max} - d_{offset}$
    $right := x_{min} - d_{offset}$
    $top := y_{max} - d_{offset}$
    $bot := y_{min} + d_{offset}$
    $\vec{c} = \vec{p}$
    $steps := \frac{(x_{max} - x_{min} - d_{offset})}{f}$
    if $|\vec{x_d}[0] - left| < |\vec{x_d}[0] - right|$ then
        $f := -f$
    end
    for $i := 0; i < steps; i := i + 1$ do
        $\vec{n} \leftarrow \begin{bmatrix} \vec{c}[0] \\ top \\ H \end{bmatrix}$
        $traj.append(goToPoint(\vec{c}, \vec{n}, v_s, getHeading(\vec{c}, \vec{n})))$
        $\vec{c}[1] := top$
        $\vec{n}[0] := f + \vec{n}[0]$
        $traj.append(goToPoint(\vec{c}, \vec{n}, v_s/c_f, getHeading(\vec{c}, \begin{bmatrix} \vec{n}[0] \\ bot \end{bmatrix})))$
        $\vec{c}[0] := f + \vec{c}[0]$
        $\vec{n}[1] := bot$
        $traj.append(goToPoint(\vec{c}, \vec{n}, v_s, getHeading(\vec{c}, \vec{n})))$
        $\vec{c}[1] := bot$
        $\vec{n}[0] := f + \vec{n}[0]$
        $traj.append(goToPoint(\vec{c}, \vec{n}, v_s/c_f, getHeading(\vec{c}, \begin{bmatrix} \vec{n}[0] \\ bot \end{bmatrix})))$
        $\vec{c}[0] := f + \vec{c}[0]$
    end
    return $traj$
end
```

Algorithm 3: Scanning trajectory planning algorithm.

Chapter 6

Evaluation

This chapter summarizes the experimental evaluation of the presented algorithms.

6.1 Simulation

An essential part of the evaluation and testing of the solution was done using the Gazebo [1] robotic simulator. Gazebo can simulate multiple robots in a 3D environment, with extensive dynamic interaction between them and the objects. It has an integration with a Pixhawk autopilot simulator which makes the setup very similar to the real-world platform. There is a minimal difference in software that is running on the UAV and in the simulation. This fact enables a good understanding of what is going to happen in real-life experiments.

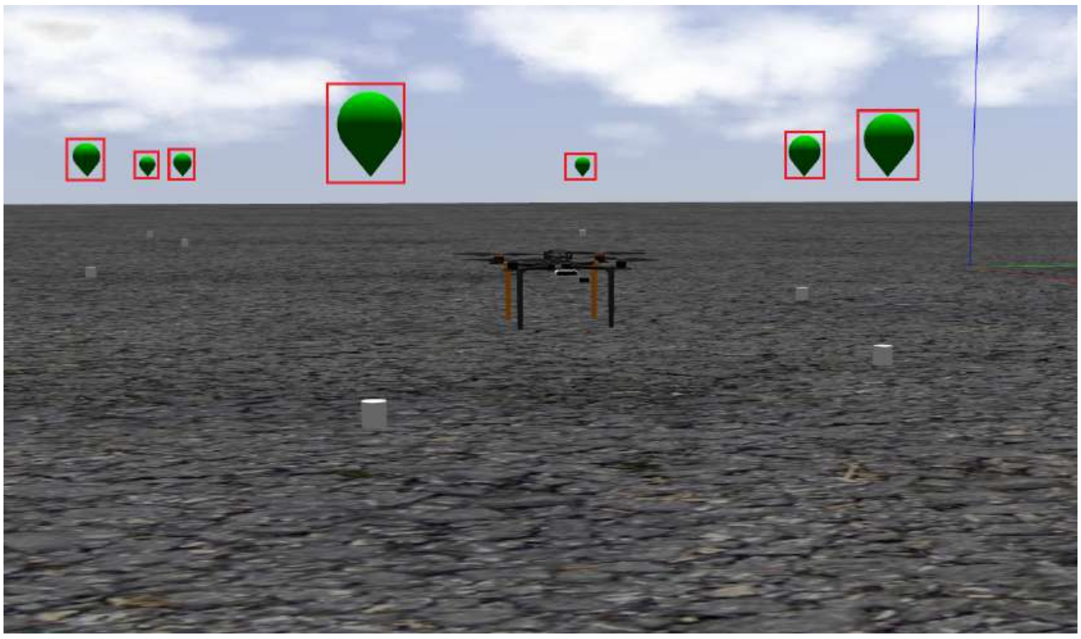


FIGURE 6.1: An UAV in the Gazebo robotic simulator. The targets(balloons) are highlighted using red rectangles.

The simulated UAV has the same set of sensors as in the real-world. Their parameters are similar to the real ones in the real world, which is very useful for evaluation and testing of different approaches in simulation and verifying them before flying in real-world experiments. The platform is also designed to be physically the same as in real-world (compare the real-world appearance Figure 2.1 and the simulation equivalent Figure 6.1).

The UAV balloon targets are also simulated using the Gazebo engine (see Figure 6.1). Using the Gazebo's interface, physical interaction is tracked, and once a collision is detected between the object and an UAV, the target balloon is deleted. The physical engine also enables wind simulation via applying a force to the object, so the targets behaved similarly to the real-world.

The biggest problem with simulation is the vision part. In the simulation, vision worked perfectly, and there were no problems with height offsets or wrong distance estimation. However, in real-world scenarios with different light conditions problems occurred, distance jumped, sudden height offset appeared. This will be discussed further.

6.2 Color Picker

The balloon target that was specified by organizers has been initially red-colored. But before the competition, the rules have been changed, and the color was changed to green. Using the Color Picker, the Object Detect was robust to change of color. The resulting color segmentation is shown in Figure 6.2. It just required calibration before use, and that's all. The color picking took around 40 seconds on the competition and could be done from long distance.

One of the main features of the developed plugin is its distribution - GUI is separate from the computation part. Using this user can run the computation on the UAV and see the output on the computer without any wired connection using the ROS Wi-Fi connection. Though, the communication has limits in transporting images through the network, so only one drone could be used to pick colors at the same time.

The resulting plugin can be used with any ROS compatible camera plugin or with recorded data via RosBag[1] (ROS plugin that records topic data).

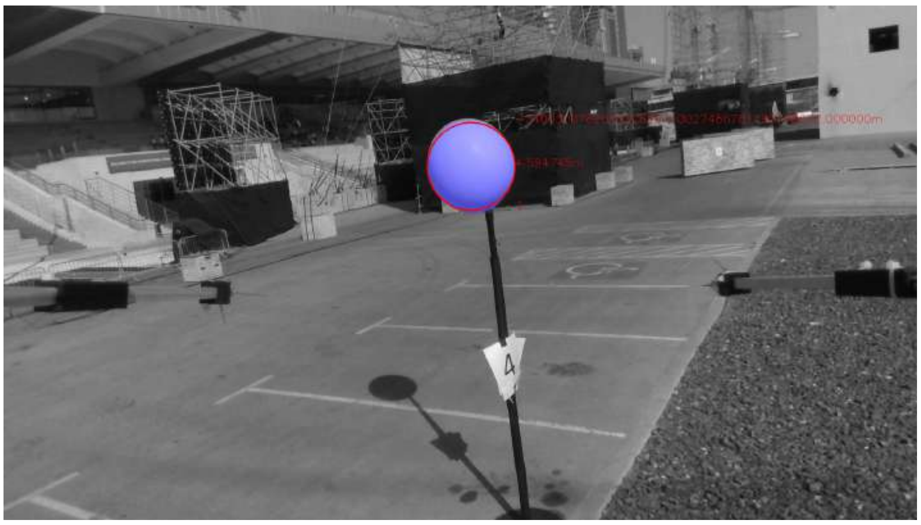
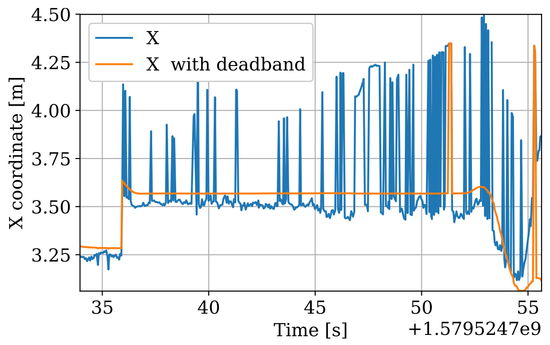


FIGURE 6.2: A debug image from onboard camera processed by Object Detect. The tracked object is highlighted by red circle.

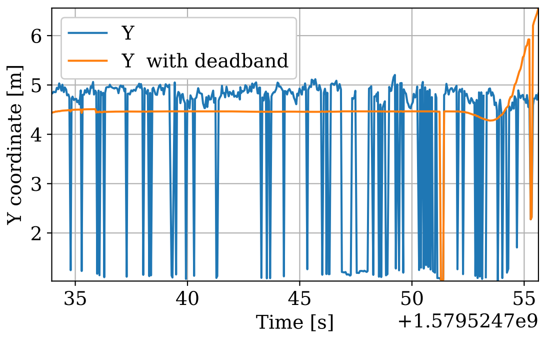
[1]http://wiki.ros.org/rosbag

6.3 Deadband

As stated in section 6.1, the targets are simulated in Gazebo, so the approaching and attacking trajectory was first tested there. Because of ideal conditions for the vision solution, the simulated wind was used to oscillate the target to test the deadband. However, even with oscillation, there were no issues. The real problem occurred when testing the target approach without attacking. The oscillation was significant, and the drone behaved aggressively. This was due to a bad calibrated vision. That is when the deadband solution was introduced. In Figure 6.3, the position of the target jumped, and how drastically deadband stabilized the reference. In this situation, the deadband control zone was set to 3 meters, since the target position jumped so much. This was only at a certain angle but was very risky for the drone.



(A) Stabilization of X coordinate using deadband with D_f 3 m.



(B) Stabilization of Y coordinate using deadband with D_f 3 m.

FIGURE 6.3: Stabilization of reference obtained from bad calibrated **Object Detect**.

6.4 Target approach and attack

Before the competition, multiple experiments were concieved in different and challenging environments, see Figure 6.4. In this section, a comparison of the simulation data and results from the competition will be provided.

(A) First preliminary tests, executed with an old UAV platform.

(B) First preliminary tests with a new MAV platform, dedicated for solving this task.

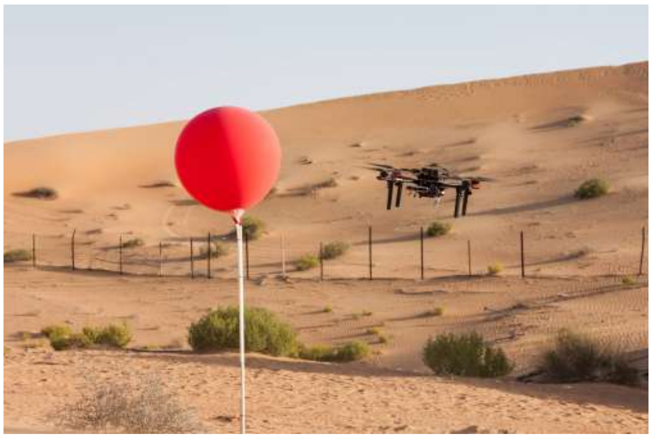
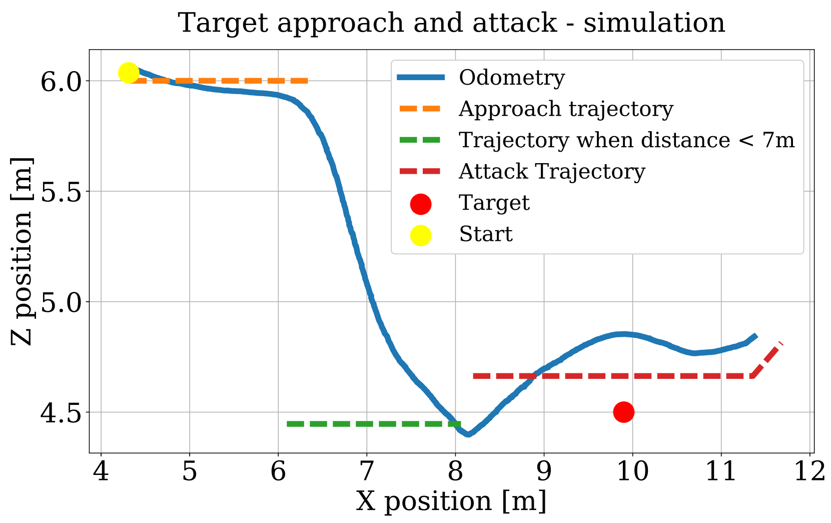

(C) Deadband test in desert.

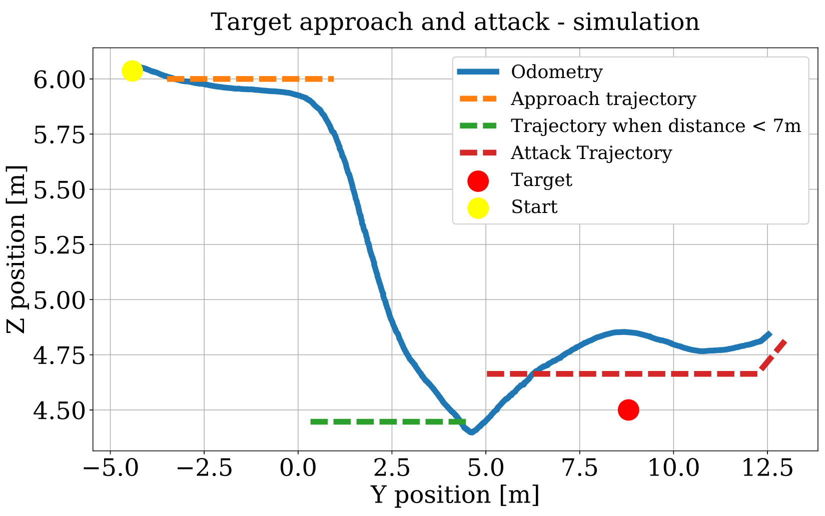
FIGURE 6.4: Photos of the tests done throughout summer 2019 - winter 2020.

On the Figure 6.5 are shown target approach and attack trajectory executed by the drone in simulation. The trajectories are shown in XZ, YZ coordinate planes to visualize the approach scheme as here Figure 5.2 and the attack scheme in Figure 5.3. The MPC reshaped the trajectory and made it more smooth, though the transition was fast, and the drone went through the target with height offset 0.3 m.

In comparison to simulation the Figure 6.6 shows target approach and destroying performed during Grand Challenge of the MBZIRC 2020. The trajectory in real-world performed even better and smoother then in simulation. Execution of this trajectory is shown in Figure 6.7.

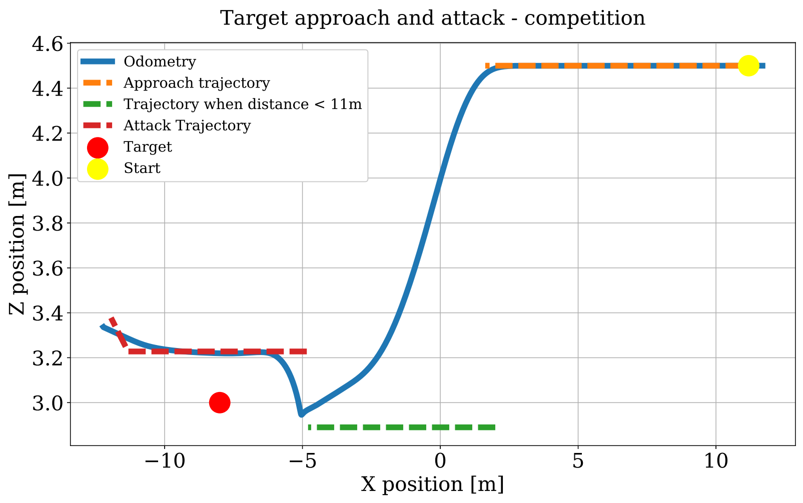


(A) Approach and attack plot in XZ plane in simulation

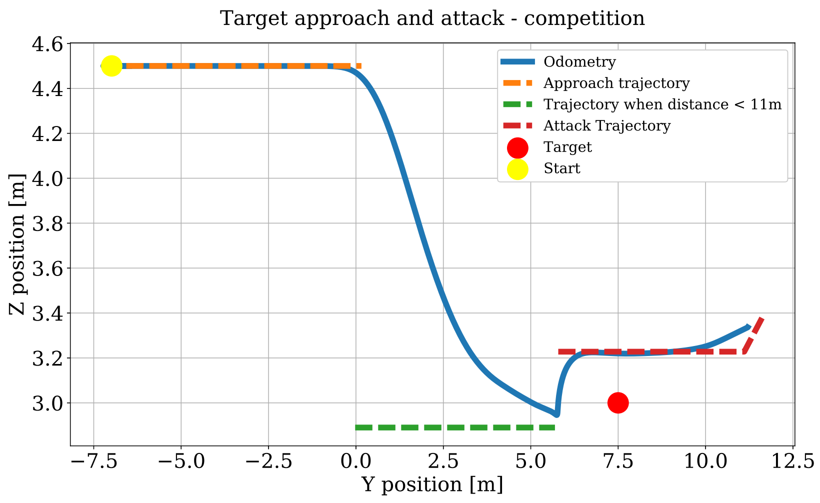


(B) Approach and attack plot in YZ plane in simulation

FIGURE 6.5: Approaching and attacking the target in simulation.



(A) Approach and attack plot in XZ plane



(B) Approach and attack plot in YZ plane

FIGURE 6.6: Approaching and attacking the target on Grand Challenge.

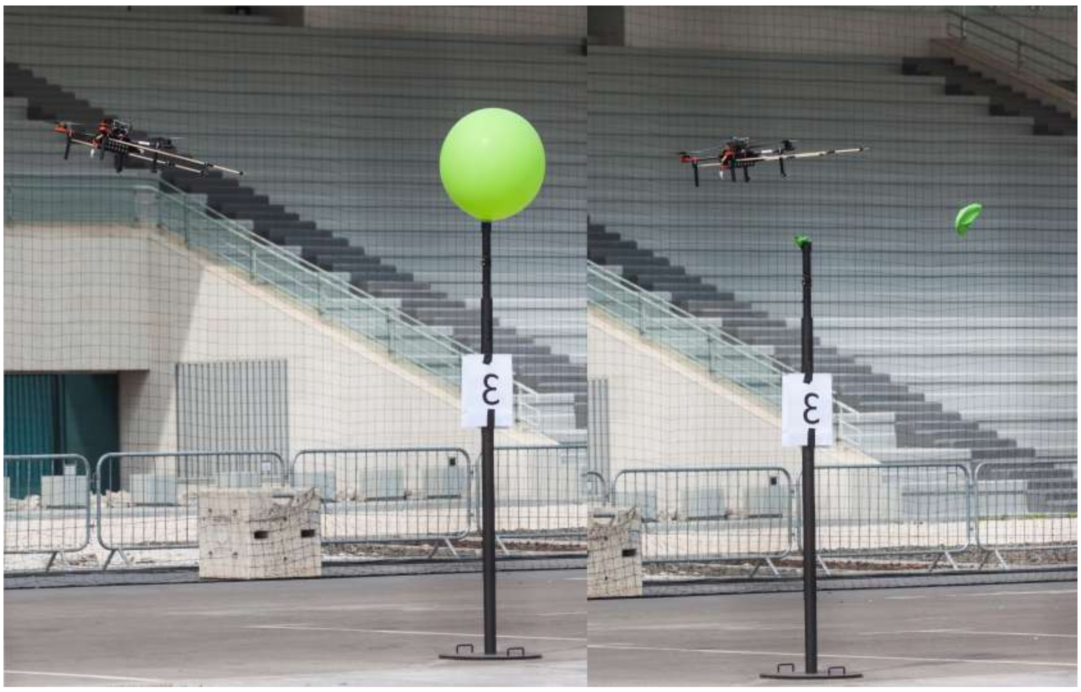


FIGURE 6.7: A series of pictures showing drone approaching the balloon (from the left side) and popping it (on the right).

6.5 Arena scanning

The arena layout was given only on the rehearsal days and the solution proved itself robust by executing the whole pipeline smoothly and without problems. In Figure 6.8 is shown scan performed by UAV during grand challenge. The scanning area is shown as green rectangle, and working area as red non-convex polygon (safe zone). Though the trajectory itself is rectangular, the MPC reshaped the trajectory and made it smooth as it is shown in the figure.

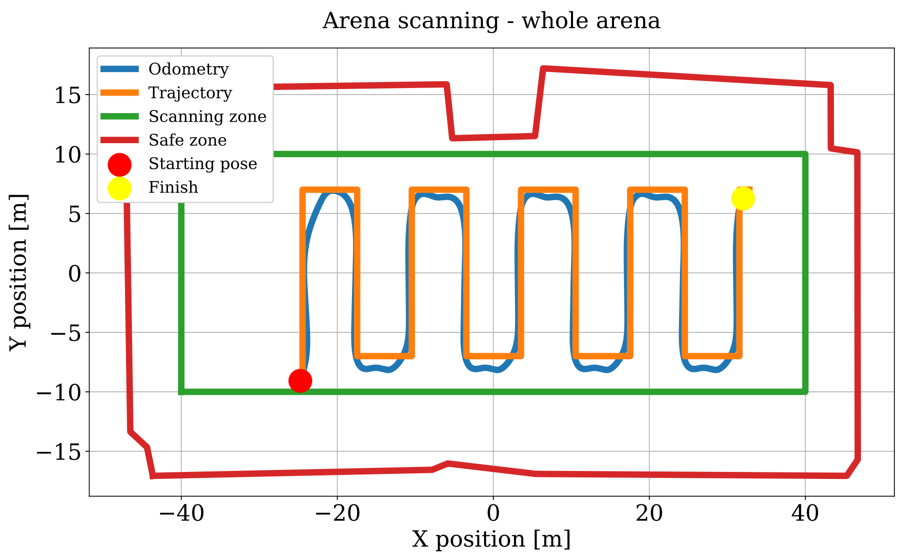


FIGURE 6.8: Scanning of the MBZIRC 2020 arena during the Grand Challenge.

In Figure 6.9 is shown scan performed by a team of two UAVs during second trial of the first challenge. It is clearly seen that the arenas are split so there won't be any possible collisions.

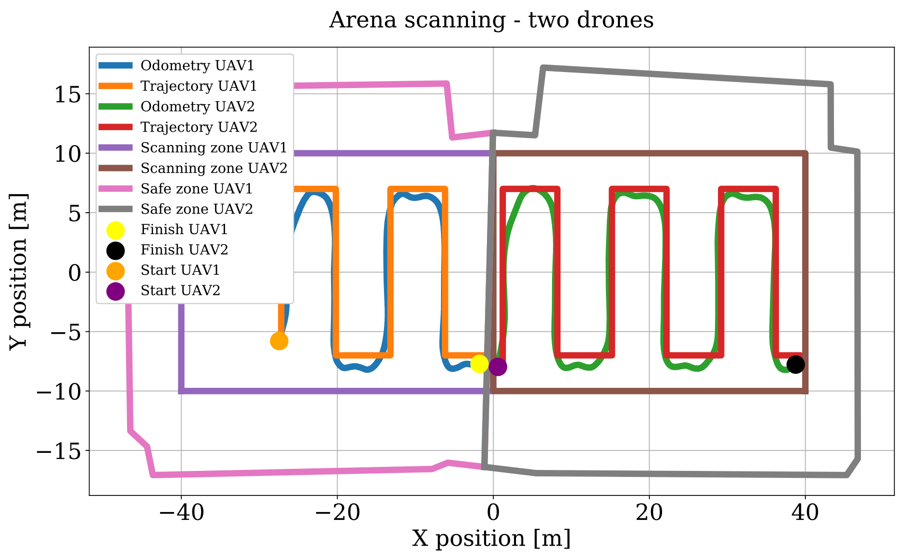


FIGURE 6.9: Scanning of the MBZIRC 2020 arena with two UAVs during First Challenge.

6.6 State machine performance

One of the major issues that can happen with state machine is that it can get stuck in one state or get focused only on one target. During the competition there was no such case, and the resulting algorithm has worked as planned state by state thanks to the robust design and builtin failsafe mechanisms.

Video of the state machine performance is published on YouTube[2].

An example of full cycle, going through **IDLE** state to state **DESTROYING** through state **GOING AROUND** is shown in Figure 6.10 for one UAV during the Grand Challenge of MBZIRC 2020. The plot shows the process of scanning the area, then spotting the target, executing approaching trajectory and then destroying the target.

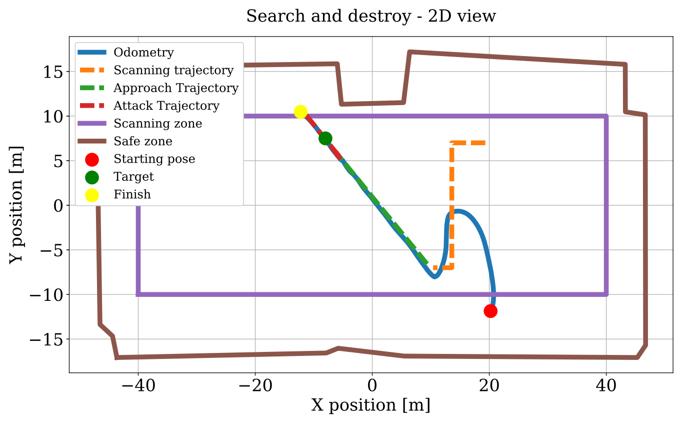


FIGURE 6.10: Full cycle of state machine during the Grand Challenge of MBZIRC 2020.

It is hard to calculate the resulting speed of the proposed algorithm because of resets during the trial. During the trial each team may call for a reset, after it they can enter the arena and change the drone's position, change batteries, reboot the onboard computer.

Trial	Drones	Time	Note
Rehearsal 3	1(2)	154 s	One UAV crashed due to disarm + reset
Trial 1	1(2)	255 s	One UAV crashed due to disarm + reset
Trial 2	2	315 s	Two UAVs, one drone with mechanical issue caused a height offset + reset
Grand Challenge	1	181 s	One reset due to emergency landing of another UAV
Desert	2	109 s	Test on self-made arena with 4 balloons.

TABLE 6.1: State machine timings. Time is calculated only when drone was in the air and until the last balloon was popped.

Video from the first run can be seen here: `https://www.youtube.com/watch?v=39q0irwiYoE`

In Table 6.1 time[3] is shown that state machine needed to destroy all the balloons in the designated arena. During the rehearsals, one drone had a hardware issue that resulted in its crash. The crash also happened during the first trial, so only one drone

[2]https://www.youtube.com/watch?v=2-cLSjRCKDg
[3]The time data was obtained by reviewing videos from the competition.

was used. The data clearly shows that one drone was more efficient in destroying the balloons in the competition, but not in the desert.

Although during the second trial (see **??**), the two drones destroyed the first four balloons (out of 5) just in 72 s, the last one took 183 s. This was because of the height offset, which we could not investigate, whether it was due to hardware or software issues. The drone performed four tries to pop the balloon, and the resulting fifth was a success. This shows that the state machine was able to find a place where it could destroy the target and got it, which is a success.

Since the positions of the targets changed after each trial we could not test how consistent the state machine is. Although we can state that proposed algorithm fulfilled fulfilled its task successfully.

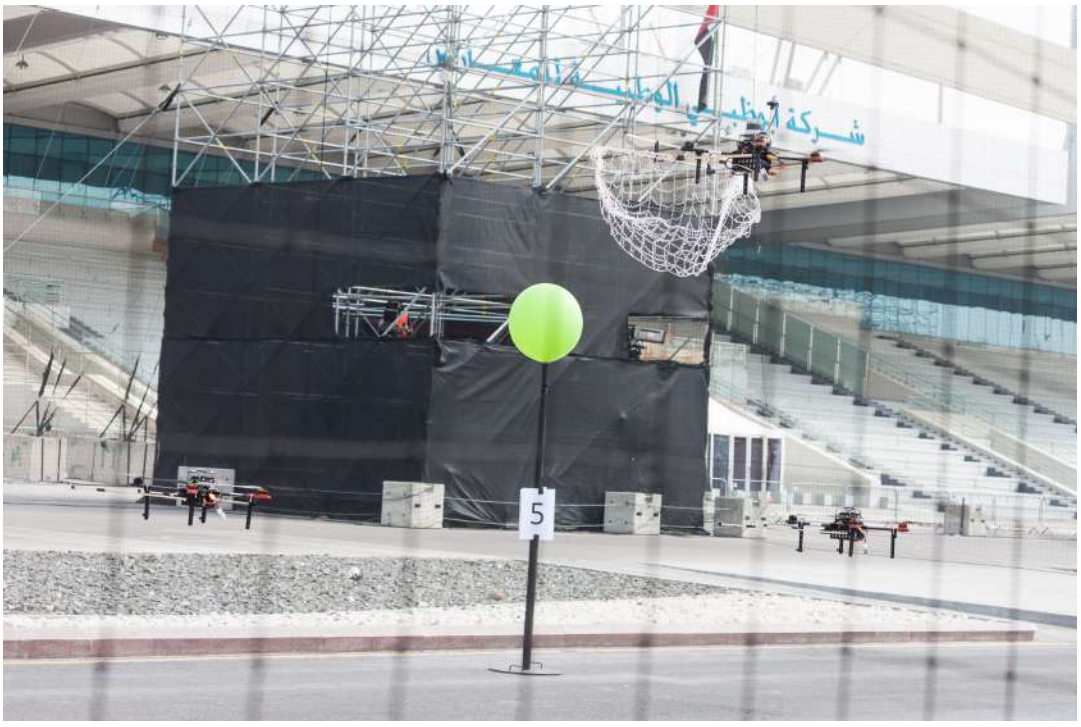


FIGURE 6.11: Team of UAVs during second trial of the MBZIRC 2020 First Challenge.

Chapter 7

Conclusion

In this work, a system for autonomous balloon popping by a team of cooperating UAVs for solving Challenge 1 of the MBZIRC 2020 competition was presented. The developed system consists of a high-level state machine which controls the target searching and destroying the targets, a semi-automatic tool for efficient and fast color picking, a trajectory generation scheme for approaching, and attacking the target and the multi robotic scenario for multiple UAVs. Every part of the system was extensively tested in Gazebo robotic simulator as well as real-world outdoor experiments in different environments. Furthermore, as the resulting test was conducted at the MBZIRC 2020 competition, the results are described in section 7.1.

The resulting system consists of:

- High-level state machine that controlled the search and destroy mission described in chapter 3.
- Object Detection color based computer vision system was integrated into the system for target localization purposes in chapter 4.
- Color picking plugin was designed and implemented in chapter 4. Furthermore, the plugin is published and available at `https://github.com/ctu-mrs`.
- An approaching and attacking strategy, as well as strategy for searching the targets, were designed and implemented in chapter 5.
- The system was tested in simulation and in the real world in different conditions. The results of the tests are described in chapter 6. The whole solution was used on the MBZIRC 2020 challenge, and the results are in section 7.1.

7.1 Competition result

The final scores of the MBZIRC 2020 are published on the official page[1] of the competition. List of participants is full of top rated technical universities from all the world[2], although not everybody scored the highest rank. Top teams are shown in Table 7.1.

[1]`https://www.mbzirc.com/winning-teams/2020`

[2]`https://www.mbzirc.com/qualified-teams/2020`

Team	Challenge 1	Grand challenge
Beijing Institute of Technology	100	-
CTU in Prague, UPENN and NYU	72	72
University of Tokyo	42	0
University of Bonn	30	30
UPM, UPO, PUT and CNRS	18	40.5

TABLE 7.1: MBZIRC resulting scores.

The maximum points that the team could obtain for balloon popping were 30 points from 100 overall for the whole challenge.

As part of team of CTU in Prague, UPENN and NYU we scored second in First challenge and first place in Grand challenge (see Figure 7.1). For the first and the grand challenges we scored equal amount of points 72 , where 30 points are for the balloon hunting.

(A) (B)

FIGURE 7.1: Team of CTU in Prague, UPENN and NYU secured the first place in the Grand and Second Challenges, and second place in the First challenge.

Bibliography

[1] Gazebo - a dynamic multi-robot simulator. Available: https://github.com/osrf/gazebo.

[2] Tomas Baca, Daniel Hert, Giuseppe Loianno, Martin Saska, and Vijay Kumar. Model predictive trajectory tracking and collision avoidance for reliable outdoor deployment of unmanned aerial vehicles. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[3] Tomas Baca., G. Loianno, and M. Saska. Embedded model predictive control of unmanned micro aerial vehicles. In *21st International Conference on Methods and Models in Automation and Robotics (MMAR)*, 2016.

[4] Marius Beul, Matthias Nieuwenhuisen, Jan Quenzel, Radu Alexandru Rosu, Jannis Horn, Dmytro Pavlichenko, Sebastian Houben, and Sven Behnke. Team nimbro at mbzirc 2017: Fast landing on a moving target and treasure hunting with a team of micro aerial vehicles. *Journal of Field Robotics*, 36(1):204–229, 2019.

[5] Bill Canis. Unmanned aircraft systems UAS: Commercial outlook for a new industry, 2015.

[6] Oscar Chang, Patricia Constante, Andrés Gordon, and Marco Singana. A novel deep neural network that uses space-time features for tracking and recognizing a moving object. *Journal of Artificial Intelligence and Soft Computing Research*, 7(2):125–136, 2017.

[7] International Color Consortium et al. Image technology colour management-architecture, profile format, and data structure. *Specification ICC. 1: 2004-10 (Profile version 4.2. 0.0)*, 2004.

[8] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. Centernet: Keypoint triplets for object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6569–6578, 2019.

[9] Jan Faigl, Petr Váňa, Robert Pěnička, and Martin Saska. Unsupervised learning-based flexible framework for surveillance planning with aerial vehicles. *Journal of Field Robotics*, 36(1):270–301, 2019.

[10] M. Flint, M. Polycarpou, and E. Fernandez-Gaucherand. Cooperative control for multiple autonomous uav's searching for targets. In *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, volume 3, pages 2823–2828 vol.3, 2002.

[11] George H Joblove and Donald Greenberg. Color spaces for computer graphics. In *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, pages 20–25, 1978.

[12] Curtis D Johnson. *Process control instrumentation technology*. Prentice Hall PTR, 1999.

[13] R. E. Kalman. A new approach to linear filtering and prediction problems" transaction of the asme journal of basic. 1960.

[14] Taeyoung Lee, Melvin Leok, and N Harris McClamroch. Geometric tracking control of a quadrotor UAV on se (3). In *49th IEEE conference on decision and control (CDC)*, pages 5420–5425. IEEE, 2010.

[15] Lorenz Meier, Petri Tanskanen, Friedrich Fraundorfer, and Marc Pollefeys. Pixhawk: A system for autonomous flight using onboard computer vision. In *2011 IEEE International Conference on Robotics and Automation*, pages 2992–2997. IEEE, 2011.

[16] M. Petrlík, T. Báča, D. Heřt, M. Vrba, T. Krajník, and M. Saska. A robust uav system for operations in a constrained environment. *IEEE Robotics and Automation Letters (RAL)*, 2020.

[17] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[18] Max Schwarz. nimbro_network - ROS transport for high-latency, low-quality networks, Robot Operating System (ROS), 2015. Available: `https://github.com/AIS-Bonn/nimbro_network`.

[19] CIE Color Space. Gernot hoffmann.

[20] V. Spurny, Tomas Baca, M. Saska, R. Penicka, T. Krajnik, J. Thomas, D. Thakur, G. Loianno, and V. Kumar. Cooperative Autonomous Search, Grasping and Delivering in a Treasure Hunt Scenario by a Team of UAVs. *Journal of Field Robotics*, 36(1):125–148, 2019.

[21] M. Vrba, D. Heřt, and M. Saska. Onboard marker-less detection and localization of non-cooperating drones for their safe interception by an autonomous aerial system. *RA-L*, 4(4):3402–3409, Oct 2019.