Bachelor Project

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Measurement

Ryze Tello drone tracking

Bekhzod Masharipov

Supervisor: RNDr. Petr Štěpán, Ph.D.
May 2022

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Masharipov Bekhzod** Personal ID number: **483818**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Measurement**

Study program: **Open Informatics**

Specialisation: **Internet things**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Ryze Tello drone tracking

Bachelor's thesis title in Czech:

Sledování dronu Ryze Tello

Guidelines:

1) Learn about the Ryze Tello drone and how to control it from your computer using the ROS.
2) Test existing Ryze Tello drone simulators and modify them for your work if necessary.
3) Design a marker for the drone that would be detectable and allow this drone to be tracked by another drone. Test the accuracy of the marker detection.
4) Design a drone tracking algorithm and test it in a simulator and in a real experiment.

Bibliography / sources:

[1] Ryze Tech. TELLO SDK 2.0 User Guide. Online: https://dbl-cdn.ryzerobotics.com/downloads/Tello/Tello%20SDK%202.0%20User%20Guide.pdf 2018, verze 1.0 .
[2] David, Pařil. Autonomní řízení dronu Ryze Tello. BS thesis. České vysoké učení technické v Praze. Vypočetní a informační centrum., 2021
[3] Dyachenko, R. A., et al. "On the approach of synchronous control of robotic systems." Journal of Physics: Conference Series. Vol. 2032. No. 1. IOP Publishing, 2021

Name and workplace of bachelor's thesis supervisor:

RNDr. Petr Štěpán, Ph.D. Multi-robot Systems FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **18.01.2022** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until:
by the end of summer semester 2022/2023

RNDr. Petr Štěpán, Ph.D. Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

I would like to thank my supervisor for his guidance and consistent support during this thesis. Furthermore, I want to thank my family and girlfriend for their support and care.

Declaration

I declare that I have prepared the submitted work independently and that I have indicated all information sources used in accordance with the Methodical guideline for adhering to ethical principles when elaborating an academic final thesis.

Prague, May 20, 2022

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 20. května 2022

Abstract

This thesis aims to build a visual tracking system with a leader-follower formation for a pair of inexpensive DJI Ryze Tello drones. To keep the formation, only a video stream from the follower camera was used. For algorithm development and testing, a simulation of Ryze Tello drones was set up in Gazebo simulator. A Kalman filter was used to suppress the noise in the leader pose estimations and allowed for predicting the leader pose when the visual tracking algorithm temporarily failed to work.

The complete algorithm has been tested both in simulation and with real drones and has proven to work satisfactorily. The program code is delivered in the form of ROS packages.

Keywords: Autonomous UAV, Ryze Tello, Visual Tracking, Leader-Follower Formation, Kalman Filter, ROS, Gazebo

Supervisor: RNDr. Petr Štěpán, Ph.D.

Abstrakt

Cílem této práce je vytvořit vizuální sledovací systém s formací vedoucího a následovníka pro dvojici levných dronů DJI Ryze Tello. Pro zachování formace byl použit pouze přenos videosignálu z kamery následovníka. Pro vývoj a testování algoritmu byla v simulátoru Gazebo vytvořena simulace dronů Ryze Tello. Kalmanův filtr byl použit k potlačení šumu v odhadech polohy vedoucího a umožnil předpovídat polohu, když vizuální sledovací systém dočasně nefungoval.

Celý algoritmus byl testován jak v simulaci, tak se skutečnými drony a ukázalo se, že funguje uspokojivě. Programový kód je dodáván ve formě balíčků ROS.

Klíčová slova: Autonomní UAV, Ryze Tello, Vizuální sledování, Formace leader-follower, Kalmanův filtr, ROS, Gazebo

Contents

Figures

Tables

Chapter 1

Introduction

The popularity of unmanned aerial vehicles (UAVs) has been increasing rapidly. UAV can be defined as an aircraft without a pilot on board, which is either controlled remotely or able to fly autonomously to some extent [40]. Micro aerial vehicles (MAVs) are a class of UAVs appealing because of their smaller size and weight, making them more suitable for indoor flying, lower production costs, and more natural for forming a swarm of multiple aircraft. Since MAVs are very popular in the consumer market, they are better known as drones, and this term will be used in the following chapters.
The price and size advantages of drones come at the cost of low payload capacity, short operating time, and a smaller set of sensors to equip the drone with. A swarm of drones allows for broader coverage when exploring the environment and increases the amount of cargo carried. Use cases of a drone swarm are expanding into many areas such as surveillance, transportation, land inspection, data collection, and many more.

1.1 Scope

This thesis focuses on the use of inexpensive and commercially available DJI Ryze Tello drones to implement a leader and follower formation. This configuration can be advantageous, since only a leader drone must perform computationally intensive tasks, such as localization, obstacle avoidance, while a follower drone has to track the leader drone. Implementing autonomous navigation of the leader is beyond the scope of this thesis, so only tracking the leader is the main interest of future chapters.
The leader is expected to be manually controlled at slow speeds of up to 0.5 m/s, although there is no limitation that the drone leader can operate autonomously if the speed requirement is met.
Most inexpensive drones, including the Ryze Tello, have a minimal number of sensors, such as a monocular camera, IMU, and altitude sensor. Therefore, it is essential to rely only on this set of sensors when developing an algorithm for leader tracking.

1.2 Goals

Goals of this thesis can be described as:

1. Explore capabilities of the Ryze Tello drone. Use Robot operating system (ROS) to control the drone.
2. Create a model of the Ryze Tello drone in Gazebo simulator [7].
3. Detect and estimate a pose of the leader drone using the follower's camera. Choose the appropriate solution to achieve this and test its accuracy.
4. Design a Kalman filter to track the leader's pose, reduce measurement noise and predict a future pose.
5. Control the follower drone such that the distance to the leader is at the desired value.
6. Provide all solutions as ROS packages. Conduct experiments in the simulator and verify with real drones.

1.3 Terminology

The term leader in this paper refers to the leader drone that is assumed to be manually or autonomously controlled. The term follower refers to the follower drone whose task is to follow the leader.
Tello refers to the basic edition of Ryze Tello drone [24].
ROS is an abbreviation for Robotic operating system [16].

Chapter 2

Ryze Tello drone

Ryze Tello is a low cost drone developed by a tech startup Ryze Technology [23]. Priced at $99, it features DJI flight control system and an Intel image processor.

Figure 2.1: Source: [24]. Ryze Tello drone

It is important to note that Ryze Tello is available on the market in several editions, which can be confusing. In addition to the original and basic version of the Tello drone, there is a Ryze Tello EDU edition. The main difference is the ability to use the latest SDK, which has the following features:

- control multiple drones in a swarm mode. A major drawback is the lack of video output in a swarm mode.
- access the downward facing camera. Only one video stream can be transmitted at a time.
- shut down the motors preventing a drone from overheating in stationary mode
- plan missions using special patterns called mission pads.

More advanced and the latest drone edition is Robomaster Tello Talent (TT) [14]. This edition is based on the Tello EDU and extended using Robomaster TT expansion kit (RMTT) [15], which contains an ESP32 module, programmable 8x8 LED dot matrix screen and LED indicator, a single-point ToF sensor, 5.8-GHz WiFi module and special extension adapter to connect additional sensors. It is possible to upgrade an existing Tello EDU drone with RMTT expansion kit.

Figure 2.2: Source: [14]. Robomaster Tello Talent

Remaining editions, such as a Ryze Tello Boost Combo [26] and Iron Man edition [27] are Ryze Tello drones, which include an extra set of batteries and other accessories.

Since the availability of Tello EDU and Robomaster TT models is limited at the time of writing, this thesis uses a more affordable Ryze Tello drone in the basic version.

2.1 Tello characteristics and features

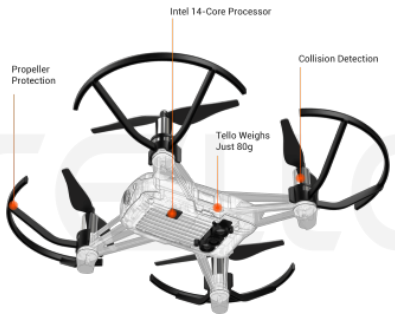

Figure 2.3: Source: [24]. Bottom part labeling of the Ryze Tello

Tello uses a visual positioning system (VPS) consisting of a pair of infrared sensors (IR) and a pointing-down camera to ensure a stable flight and prevent drift when hovering. However, it can be affected if Tello flies over monochrome, highly reflective, transparent surfaces without patterns or when flying in dark or poorly lit environments. If the VPS fails, Tello goes into Attitude mode and is no longer stable, which may be unsafe when flying indoors. Despite the VPS, Tello has a collision detection system that shuts down the engines in the event of a collision with an obstacle. Tello has no built-in cooling system and is only cooled by the airflow during flight. When taking measurements, this can be a problem and is solved by using external cooling, e.g., a fan. There is also a built-in IMU sensor and barometer.

Aircraft	
Weight	80g
Dimensions	98×92.5×41 mm
Camera	720p 30 FPS
Photo	5MP (2592x1936)
Field of view	82.6°
Battery	1.1 Ah/3.8 V
Flight characteristics	
Max flight distance	100 m
Max speed	8 m/s
Max flight time	13 min
Max flight height	30 min

Table 2.1: Source: [29]. Physical and flight characteristics of the Ryze Tello drone.

It is necessary to be connected to an access point hosted by Tello to communicate with it. This creates a limitation of being able to connect to only one drone at a time.
Tello uses UDP packets to receive and send information.

2.2 Measuring camera latency

The Tello's camera stream is encoded in H264 format. The Tello ROS driver described later does not decode a video stream by default. In order to enable stream decoding, it is necessary to modify the launch file by setting the parameter **stream_h264_video** to false.
The final decoded stream has a relatively high latency, which has a significant impact on the performance of the visual tracking algorithm. Therefore, it was necessary to measure and determine the camera latency.
For this purpose, Tello's camera recorded a running stopwatch while an external camera with 240 fps recorded the stopwatch and the decoded stream received from Tello.

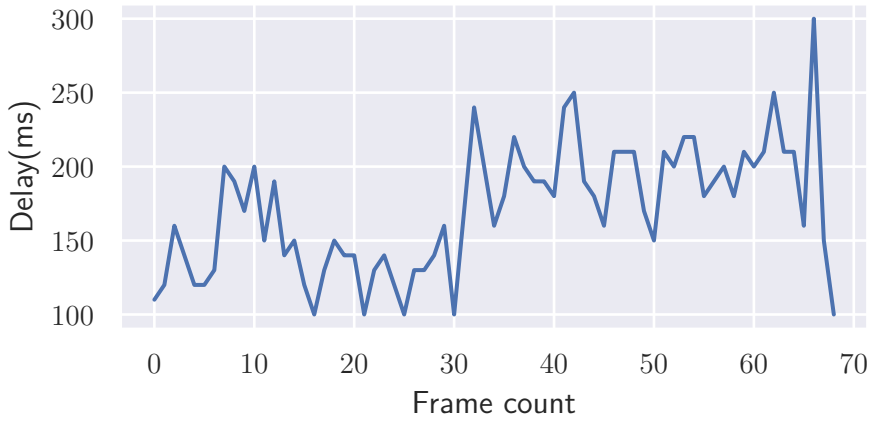

Figure 2.4: Ryze Tello camera latency

Average latency of 171.30 milliseconds was calculated from the measured data in Figure 2.4.
It is worth mentioning that the delay between frames may become significantly bigger than stated before if WiFi connection is weak or the machine controlling the drone overloads. Also, right after the ROS driver is connected to the drone, approximately 3 seconds of delay is present in the video stream. This can be explained by filling the video buffer while the driver is being initialized. To reduce the aforementioned issue, the first 270 frames are skipped to minimize the lag of the video stream.

2.3 Tellopilots

One of the significant advantages of the Ryze Tello drone is the vast community of users who share ideas and projects with Tello. There is a community forum called Tellopilots[31], where one can find solutions to problems or find helpful information related to the Tello drone when official documentation is not enough. Another resource is a Thingverse website [35], where a large number of 3D print-ready prototypes are available that extend Tello's capabilities.

Chapter 3

Robot operating system

"ROS (Robot operating system) is an open source software development kit for robotics applications. ROS offers a standard software platform to developers across industries that will carry them from research and prototyping all the way through to deployment and production" [16].

ROS is not an operating system but a software platform that significantly simplifies the development of algorithms to control robots. One of the main benefits of ROS is a large number of ready-made packages for most tasks created by a large community.
There are multiple ROS distributions available on different OS versions. The choice of a particular distribution depends on the operating system as well as on the desired packages to be used as the basis. In order to solve this problem and provide an opportunity to switch between different distributions quickly, it is possible to use system containers, which will be described in the following sections.

3.1 ROS architecture

A source code in ROS is divided into packages. Each package addresses a particular concern and may include nodes, launch files, scripts, and configuration metadata. For instance, it is reasonable to have a package to control a robot and a distinct package to process images obtained from an onboard camera. These various parts of a system may be implemented and distributed independently. Consequently, packages are placed in a workspace, which is a single folder designated for a specific project.
Basic working units in ROS are nodes. A node performs computations and communicates with other nodes via topics and services. Each topic has its subscriber and publisher, where the former receives information and the latter transmits it, with no explicit link between them. There are other types of communication - services - through which a node can request information from another node and receive it in a response.
A launch file is widely used for starting multiple nodes all at once. Bag files can be used to record and later replay selected topics. More information

about ROS and its applications can be found on [43], [44].
It is worth mentioning a few tools that can come in handy when developing ROS applications:

- Rviz [22] is a 3D visualization tool that can be used to visualize robot position and path, map of an environment, sensor output such as 3D point cloud from a LiDAR, depth map from a stereo camera, etc.
- Rqt [20] is a software platform consisting of many plugins for visualizing and debugging applications in the ROS environment.
- PlotJugger [12] is a lesser-known but nevertheless powerful tool. It allows to visualize topics in real-time in the form of time series, plus apply built-in functions such as converting angles in the form of quaternions to Euler angles.

3.2 Containers

"A container is a lightweight form of operating system virtualization. A single container might be used to run anything from a small microservice or software process to a larger application. Inside a container are all the necessary executables, binary code, libraries, and configuration files. Compared to server or machine virtualization approaches, however, containers do not contain operating system images. This makes them more lightweight and portable, with significantly less overhead. In larger application deployments, multiple containers may be deployed as one or more container clusters" [38].

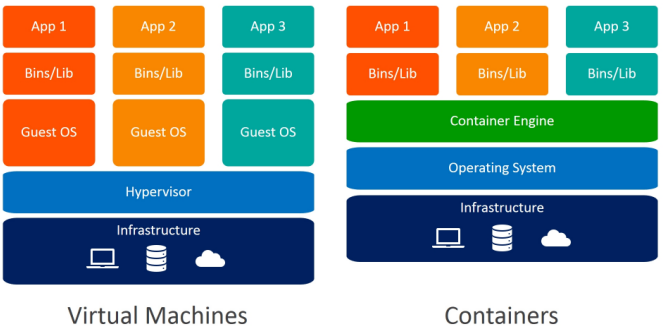

Figure 3.1: Source:[6]. Difference between virtual machines and containers.

3.2.1 LXD Containers

"LXD is a next generation system container and virtual machine manager. It offers a unified user experience around full Linux systems running inside containers or virtual machines" [11].

LXD was designed as an extension for LXC - well-known Linux container runtime. It comes with a back-end daemon and CLI, which make usage of

containers straightforward. LXD is image-based and it provides a wide range of Linux-based distributions through the remote repository.

It is worth noting a difference among other Container systems like Docker or Rkt, which considered to be application containers whereas LXD provides support for system containers.
"Application containers package a single process or application. System containers, on the other hand, simulate a full operating system and allow to run multiple processes at the same time." [1]

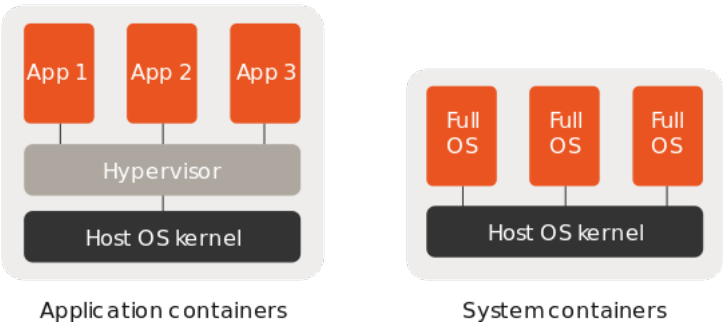

Figure 3.2: Source: [1]. Application and System containers.

The latest LXD version can be installed on different Linux distributions as a snap package. There are few commands needed to set up a new container for ROS development.

Command	Description
lxc ls	List all available container instances
lxc launch imageserver:imagename instancename	Create a new container instance based on image
lxc start instancename	Start an instance
lxc stop instancename	Stop an instance
lxc exec containername – /bin/bash	Run a shell inside a container
lxc profile create profilename	Create a new profile
lxc profile assign containername p1,p2	Assign profiles p1, p2 to a container

Table 3.1: Basic commands to set up a new container

Before going to set up a new container, it is necessary to understand what LXD profile is. Each container instance has its own configuration file in the form of namespaced key values. Container configuration includes properties like the number of CPU cores exposed to a container, memory limits, and pass-through devices. Profiles are named collections of configurations that can be applied to more than one container. Whenever a new container instance is created LXD applies `default` profile, which contains basic configuration including network interface **eth0**.

3.2.2 Setting up a new container for ROS development

First, start by pulling an Ubuntu 18.04 image and creating an empty container instance. Give it a name to associate with a specific ROS version, in this example - ROS Melodic distribution [18].

```
$ lxc launch ubuntu:18.04 rosmelodic
```

Then create two profiles:

```
$ lxc profile create gui
$ lxc profile create ros
```

- gui - enables GUI and GPU support
- ros - adds ros repositories

Use following configuration for **gui**:

```
config:
environment.DISPLAY: :0
environment.PULSE_SERVER: unix:/home/ubuntu/pulse-native
nvidia.driver.capabilities: all
nvidia.runtime: "true"
user.user-data: |
#cloud-config
runcmd:
- 'sed -i "s/; enable-shm = yes/enable-shm = no/g" /etc/pulse/client.
    conf'
packages:
- x11-apps
- mesa-utils
- pulseaudio
description: GUI LXD profile
devices:
 PASocket1:
       bind: container
       connect: unix:/run/user/1000/pulse/native
       listen: unix:/home/ubuntu/pulse-native
       security.gid: "1000"
       security.uid: "1000"
       uid: "1000"
       gid: "1000"
       mode: "0777"
       type: proxy
 X0:
       bind: container
       connect: unix:@/tmp/.X11-unix/X1
       listen: unix:@/tmp/.X11-unix/X0
       security.gid: "1000"
       security.uid: "1000"
       type: proxy
 mygpu:
```

```
      type: gpu
      name: x11
used_by: []
```

Listing 3.1: Source: [21]. GUI profile, which enables graphical user output in containers

```
# use along with GUI profile for rviz and other graphic programs
config:
raw.idmap: both 1000 1000 # needed for container to have write
    permissions in shared disk
user.user-data: |
#cloud-config
runcmd:
- "apt-key adv --fetch-keys 'https://raw.githubusercontent.com/ros/
    rosdistro/master/ros.asc'"
- "apt-add-repository 'http://packages.ros.org/ros/ubuntu'"
- "apt-add-repository 'http://packages.ros.org/ros2/ubuntu'"
description: ROS
# if not set with: lxc config device add [<remote>:]instance1 <device
    -name> disk source=/share/c1 path=opt
# devices:
#   share-dir:
#     path: /home/ubuntu/
#     source: /home/nick/Projects/ros/melodic-moveit
#     type: disk
name: ros
```

Listing 3.2: Source: [9]. ROS profile, which adds the repositories necessary to install ROS distributions

Now that the profiles have been created, add them to the existing container:

```
$ lxc add profile rosmelodic gui, ros
```

To test the newly created container, run the following commands:

```
[host]$ lxc exec rosmelodic -- /bin/bash
[rosmelodic]$ glxgears
[rosmelodic]$ xclock
```

If everything went correctly, there should be both **glxgears** and **xclock** windows.

At this point, it is possible to create containers and install any ROS distribution, including GUI tools such as Rviz or RQT.

3.2.3 Port forwarding

Ports are often used when communicating with commercial robots; e.g., a stream with camera images of a Ryze Tello drone is published on a particular port. Since containers are in their own local subnet by default, there is no access to ports within the container. Port forwarding can be used to solve this problem. LXD provides a simple way to forward ports and supports

several types of protocols, including UDP and TCP.
To forward UDP port 5555 to the same port number in a container, one can use the **lxc device add** command along with a name of the target container, name of a device(forwarding rule), source port and target port.

```
lxc config device add container port5555 proxy listen=
    udp:0.0.0.0:5555 connect=udp:127.0.0.1:5555
```

3.3 Ryze Tello ROS driver

One of the goals of the thesis was to find a suitable ROS driver to control a Ryze Tello drone. The driver refers to a middle software layer that allows control and communication with a drone within the ROS. Upon research, two of the most popular drivers were identified:

1. Tello_ros [34] based on ROS2 and uses the official SDK [28]. The distinguishing advantage of this driver is it comes with a ready-to-use model of Ryze Tello drone for simulation.

2. Tello_driver [33] is a wrapper over the unofficial library [32] and is based on ROS 1. It supports ROS Noetic distribution [19].

Since neither driver has complete documentation, it was necessary to try each of them and eventually choose the most suitable. For this purpose, containers were used, which allowed to test the drivers with two different versions of ROS thereby isolating the host operating system and eliminating possible conflicts.
According to the tests conducted with the first driver, the flight with the simulation model was too perfect and different from the real drone flight experience.
The second driver was chosen for more features such as the availability of odometry data and the ability to set the camera bitrate. Also, the amount of resources and documentation on ROS 1 is larger and more accessible.

Chapter 4
UAV Simulation

4.1 Introduction

Testing algorithms on a real UAV during the continuous development phase can be quite tricky, as it often requires access to expensive equipment and a dedicated workspace. When it comes to working with robots, a simulator is often used. The simulator must meet specific requirements. First, it must model the imperfect dynamics of a robot and a real-world environment as closely as possible. Further, it must enable an expansion of the robot using standard onboard sensors like LiDAR, Inertial measurement unit(IMU), IR or sonar sensor, and a camera. Lastly, it should be relatively easy to use and set up, so a user can focus on tackling higher-level tasks such as autonomous navigation, collision avoidance, path planning, visual tracking, etc.

In this thesis Gazebo simulator [7] was chosen as it is open source and well documented.

4.2 Gazebo

Gazebo [7] is a open source 3D simulator developed by Open Robotics. With Gazebo it is possible to accurately simulate complex robots and access physics engines including ODE, Bullet, DART and Simbody Simbody. It provides SDF [30] - declarative format language to describe robots and environments. With the use of Gazebo plugins, it is possible to mount a wide range of sensors. It offers great integration with ROS, which means very few changes have to be done to the main program to switch from a simulation to real hardware.

4.3 URDF

URDF [36] is a standard format language defined by ROS with the same purpose as SDF. This format will be used to describe a drone model and launch it with Gazebo, which internally will convert it to SDF.

The building blocks of a URDF file are links and joints.

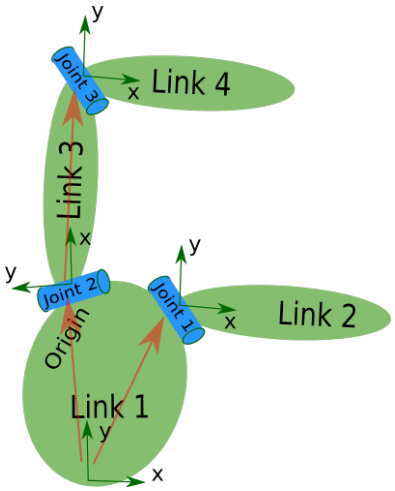

Figure 4.1: Source: [10]. Structure of a URDF file.

A link describes a rigid body by its physical properties (dimensions, position of its origin, color, etc.). The link tag in URDF may contain:

- Visual - the visible part of a rigid body. This could be a geometric primitive like a sphere, box, or external mesh file. A link may contain many visuals.
- Collision - boundaries of a body used for collision checking. A link may contain many collision elements
- Inertia - dynamic properties such as 3x3 rotational inertia matrix and mass.

Links are connected together by joint components. A joint describes the kinematic and dynamic properties of the connection (type of joint, axis of rotation, friction and damping, etc.). URDF code below describes the simplest model of a quadcopter using primitive geometric shapes.

```
<?xml version="1.0"?>
<robot name="uav">
    <material name="red">
        <color rgba="1.0 0 0 1"/>
    </material>
    <link name="base_link">
        <visual>
            <origin xyz="0 0 0.01"/>
            <geometry>
                <box size="0.07 0.02 0.02"/>
            </geometry>
            <material name="red"/>
        </visual>
        <visual>
            <origin xyz="0.05 -0.05 0"/>
            <geometry>First
                <cylinder length="0.01" radius="0.045"/>
            </geometry>
        </visual>
        <visual>
            <origin xyz="-0.05 0.05 0"/>
            <geometry>
                <cylinder length="0.01" radius="0.045"/>
            </geometry>
        </visual>
        <visual>
            <origin xyz="-0.05 -0.05 0"/>
            <geometry>
                <cylinder length="0.01" radius="0.045"/>
            </geometry>
        </visual>
        <visual>
            <origin xyz="0.05 0.05 0"/>
            <geometry>
                <cylinder length="0.01" radius="0.045"/>
            </geometry>
        </visual>
        <inertial>
            <mass value="0.1"/>
            <inertia ixx="0.000290833" ixy="0" ixz="0" iyy="0.00054"
                iyz="0" izz="0.000290833"/>
        </inertial>
        <collision name="collision">
            <geometry>
                <box size="0.18 0.18 0.05"/>
            </geometry>
        </collision>
    </link>
</robot>
```

Listing 4.1: Primitive 3D model of a drone

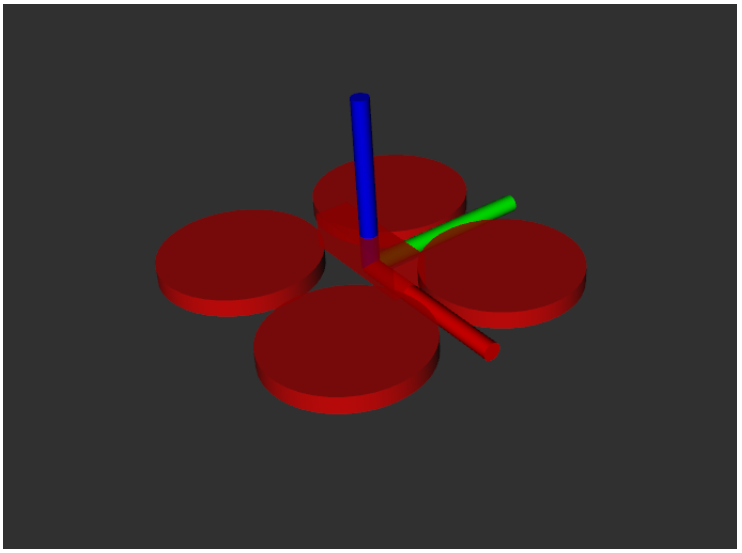

Figure 4.2: Rendered model in RViz

To make the drone fly, it is required to create a model plugin that will apply forces on the model based on the rotors speed. As this would be a time-consuming task, it was decided to use an off-the-shelf solution.

4.4 hector_quadrotor package

Hector_quadrotor [46] is a comprehensive UAV simulation system developed by a team from TU Darmstadt. It comes as a set of ROS packages related to modeling, controlling, and simulating quadrotor systems in ROS environment using Gazebo simulator [8]. The following packages are included:

- hector_quadrotor_description contains URDF files with a description of Hector quadrotor UAV with various sensors
- hector_quadrotor_gazebo contains launch files for running a complete simulation of an environment and spawning an UAV.
- hector_quadrotor_controllers provides velocity, position and altitude controllers.
- hector_quadrotor_gazebo_plugins provides plugins that are specific to the simulation of quadrotor UAVs in gazebo simulation.
- hector_quadrotor_demo contains launch files for executing indoor and outdoor simulations with hector quadrotor.

Executing the outdoor scenario from the hector_quadrotor_demo will launch RViz and Gazebo. Gazebo will display a hector quadrotor equipped with the Hokuyo UTM-30LX sensor in hilly terrain.

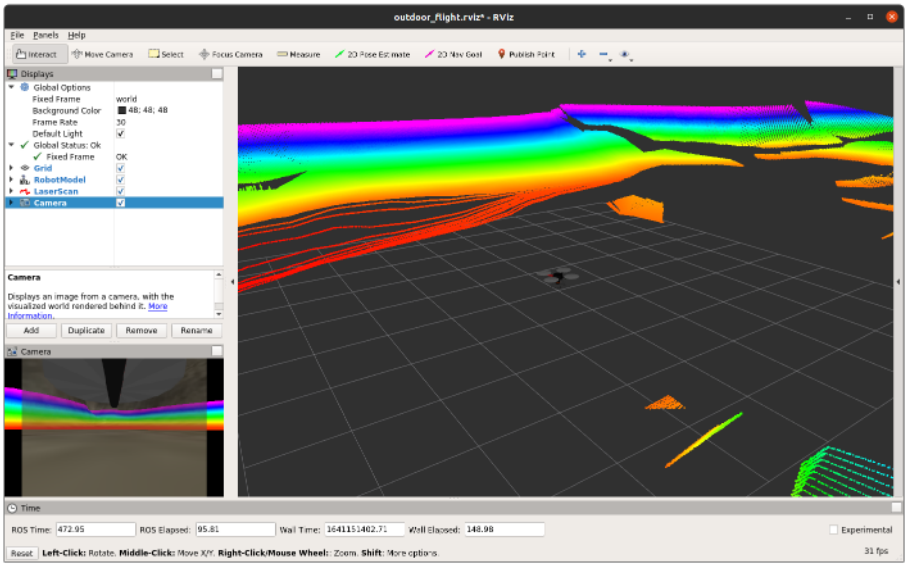

Figure 4.3: Outdoor simulation with Hector quadrotor in RViz

At the moment of writing the package is available for the Kinetic [17] and Melodic [18] distributions, whereas the newer ROS Noetic [19] has been chosen as the main distribution to communicate with the real Tello drone. This can easily be solved by using the containers presented earlier, thereby having two distributions on the same physical machine and quickly switching between them to the desired environment.

4.5 Simulating Tello UAV using hector_quadrotor package

With the hector_quadrotor package, a custom UAV model can be built with the desired set of sensors. For this purpose, a URDF file of a Ryze Tello drone is provided.
The URDF file contains a basic reference with visual, collision, and inertia tags. An approximate 3D model of the Tello drone provided in [25] was extended with propellers and used for better visualization in the simulation. The simulated drone was then augmented with the necessary sensors from the gazebo and hector plugin packages. A sonar sensor for altitude measurement was used instead of the IR sensor built into the real Tello, as it is it is not available for a simulation. A camera with the same resolution, frame rate, and viewing angle was attached to the front side of the simulated drone.

Figure 4.4: Rendered 3D model of Ryze Tello from [25], which was extended with propellers and the "Tello" label in Blender [39].

A simulation can be started by executing the launch file in the description package, which will create an instance of Gazebo simulator and RViz. The simulation will spawn a Tello drone in an indoor environment. To start, the drone /enable_motors service has to be called with a value set to True. Moving the drone is possible via /cmd_vel topic by specifying velocities along the x,y,z axes and yaw velocity. The value of the current altitude can be read from the topic /sonar_height.

4.6 Extending the simulation with ArUco marker

To complete the picture, it is necessary to simulate the ArUco marker described in the following chapter. First, a 3D cube covered with the ArUco marker was created in Blender [39]. Then the URDF file of the Tello drone was extended with a link containing a reference to the 3D marker mesh model. Finally, a fixed joint is used to attach the marker's link to the drone body.

Figure 4.5: Complete simulation with two drones and ArUco marker attached to the leader.

Chapter 5

Pose estimation using ArUco markers

In order to keep the leader-follower formation, it is necessary to identify the leader in the first place, then determine the follower's relative pose to the leader and, on the basis of this, direct the follower.
One approach would be to use convolutional neural networks for drone recognition. Although this solution does not require the use of unique markers to indicate the leader, it is computationally intensive and time-consuming.
Therefore, it was decided to use a special pattern to tag the leader, which can be accurately localized in an image captured by the follower.
First attempts were made using colored geometric objects but were unsuccessful due to many false positives, time-consuming operations for more accurate recognition, and being too dependent on good lighting.
A common approach is to use binary square fiducial markers. They have four edges to make camera pose estimation available, and each marker is associated with a unique identifier, which can be used in multiple leaders and followers scenarios.

5.1 ArUco markers

In this work, ArUco markers presented in [41] are used to detect the leader and extract its pose. As per the [2], ArUco markers are fiducial markers consisting of an inner binary matrix with black boundaries. The inner matrix pattern is what makes each marker unique and black background facilitates fast localization of a marker.
ArUco markers come in different sizes, which determine number of blocks in inner matrix, e.g. 4×4, 6×6 and with different dictionary sizes composed by markers. There is also an important parameter, which is physical size of a marker that should be measured accurately to improve pose estimation. Bigger sizes will result a better detection.

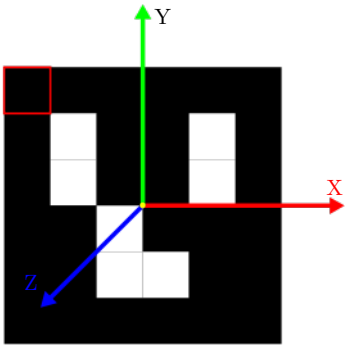

Figure 5.1: 4x4 ArUco marker with id = 3. The first corner of the marker is highlighted in red.

5.2 Pose Estimation

After a marker is detected, the transformation from the marker frame to the camera frame is computed. As shown on figure 5.1 the marker coordinate frame origin is located at the center of the marker. Complete description of how the transformation is computed can be found in [3]. To briefly summarize, it can be presented as:
Given a 3D point P_w in the world coordinate frame and 2D point p in the image plane, the task is to find the rotation and translation vectors transforming coordinates from the world to the camera frame. For this task a pinhole camera model is most commonly used.

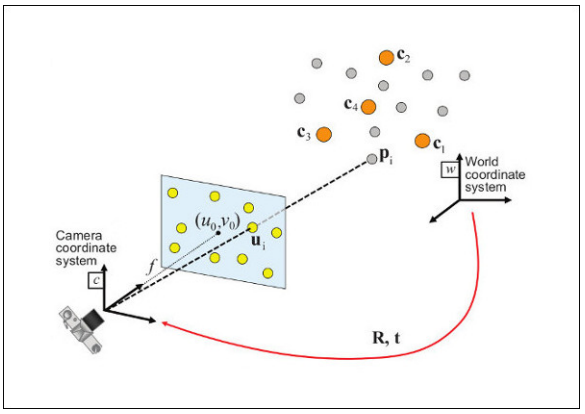

Figure 5.2: Source: [13]. Projection of 3D points in world coordinate system into the 2D image plane. R, t are the rotation matrix and the translation that describe transformation from world to the camera coordinate system.

$$s \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix} = \mathrm{K} \begin{bmatrix} \mathrm{R} \mid t \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{5.1}$$

where s is the projection scaling, $\begin{bmatrix} p_x & p_y & 1 \end{bmatrix}^T$ is 2D point in the image plane represented in homogeneous coordinates. $\mathrm{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$ is the camera intrinsic matrix that projects a 3D point in the camera frame into the image plane, f_x and f_y are focal lengths, (c_x, c_y) is the principal point. The intrinsic matrix along with distortion coefficients are obtained by calibrating the camera [5]. The tello driver [33] used in this work provides required camera parameters. R and t are the rotation matrix and the traslation vector, which transforms homogeneous vector $\begin{bmatrix} X_w & Y_w & Z_w & 1 \end{bmatrix}^T$ from the world to the camera coordinate system.
Thus the estimated pose is represented by the rotation matrix R and the traslation vector t:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{5.2}$$

Given the detected marker, R and t are those that transform all points from the marker coordinate system to the camera coordinate system.

5.3 Detection of ArUco markers in ROS using aruco_detect package

Aruco_detect [4] is an open source ROS package based on OpenCV aruco module [2] and developed by Ubiquity Robotics. It provides a node, which detects ArUco markers from an image stream, publishes their vertices and estimates the detected marker pose with respect to the camera frame.
There are two categories of parameters that can be used to configure the node - global and detection. Global parameters specify dictionary and marker size, marker's physical length in meters, enable pose estimation, one or range of marker IDs to be ignored and whether the node should publish images with detected markers. Detection parameters allow to adjust underlying detection algorithm.
The node subscribes to two topics - image stream and camera information. Camera information is a an intrinsic parameter of the camera. As already mentioned, the tello driver provides the camera intrinsic matrix along with distortion coefficients and publishes on the topic /tello/camera/camera_info. To use the node it has to be added in a launch file and provided with parameters. Listing 5.1 depicts how the aruco_detect node configured in the launch file.

```
<node pkg="aruco_detect" name="aruco_detect"
type="aruco_detect" output="screen" respawn="false">
  <param name="image_transport" value="raw"/>
  <param name="publish_images" value="true" />
  <param name="fiducial_len" value="0.048"/>
  <param name="dictionary" value="0"/>
  <param name="do_pose_estimation" value="true"/>
  <param name="ignore_fiducials" value="1-2,4-50"/>
  <param name="fiducial_len_override" value=""/>
  <remap from="/camera" to="/tello/camera/image_raw"/>
  <remap from="/camera_info"
         to="/tello/camera/camera_info"/>
</node>
```

Listing 5.1: Configuration of aruco_detect [4] node in launch file

With this configuration, only the marker with id = 3 is not ignored. An important part is remapping the node topics so that it is possible to use it directly with the tello driver.
Due to physical limits of the Tello drone, 4×4 ArUco marker with a side length of 48 mm was taped to a hard cardboard 5.3, which is in turn attached to the back of the leader drone so as not to interfere with propellers. White border

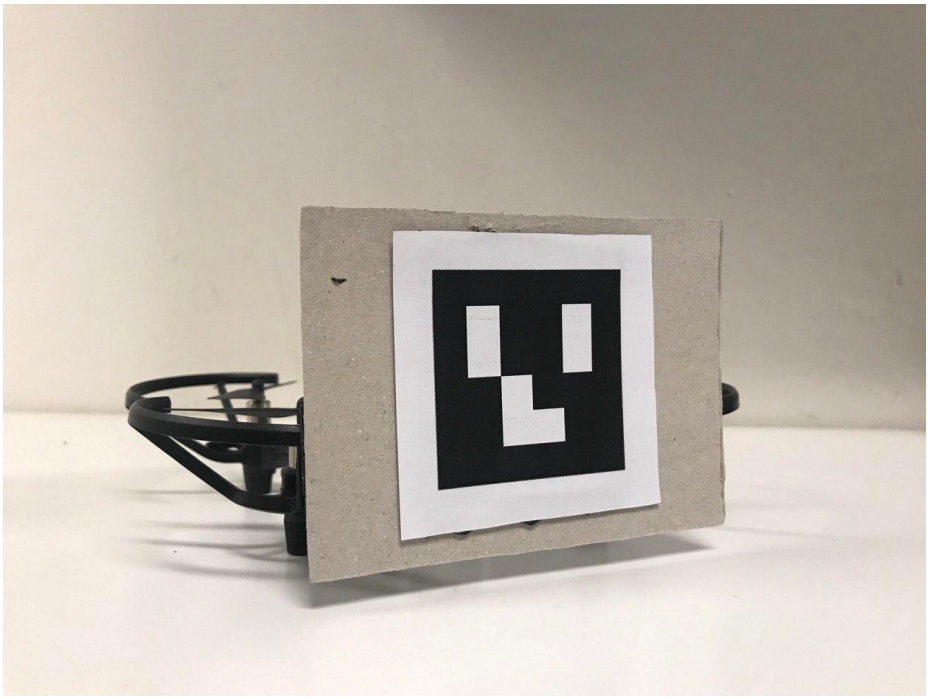

Figure 5.3: The leader drone with attached ArUco marker

around the marker increases number of detections, especially in low-light environments.

5.4 Pose estimation accuracy

The aruco_detect node publishes transformations between the detected marker and the camera frame into the **fiducial_transforms** topic. A transform message contains translation vector and rotation in quaternion form.

Several tests were conducted to measure an accuracy of pose estimation.

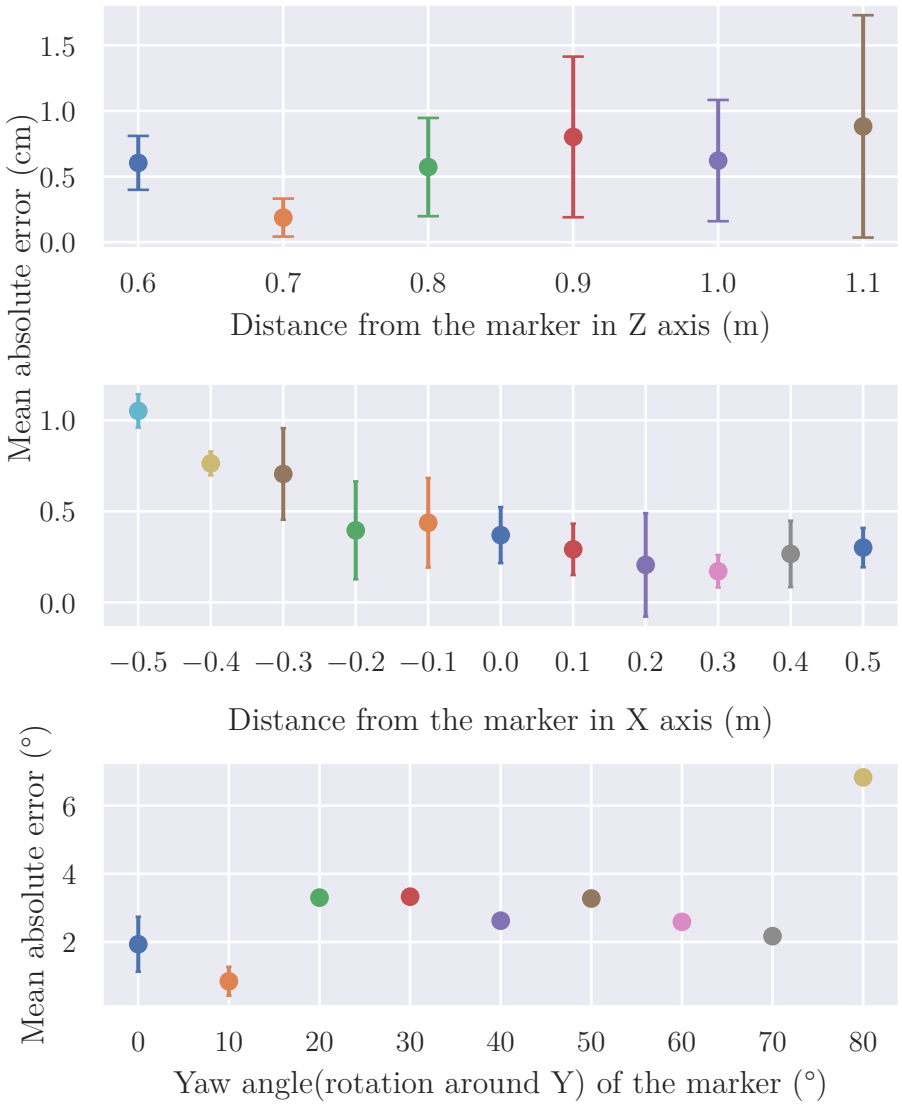

Figure 5.4: ArUco marker pose estimation accuracy. Z, X axes and Yaw angle estimation errors are shown.

The first test was organized in such a way that the follower was fixed on a certain point, and the stationary leader with the attached ArUco marker was

moved along the Z-axis from the marker with steps of 60 cm to 1.1 meters in 10 cm increments. In addition, at each step, the leader was moved along the X-axis from 0 cm to 0.5 cm in both directions. Results obtained from this test are shown in Figure 5.4:
The first graph shows the mean absolute errors of measured and true marker Z-axis distances, where each measurement was taken at different X-axis positions and then the average value was calculated. The same applies to the second graph, where the X-axis distances are taken at different Z-axis positions, the average value is calculated, and then average absolute errors are plotted. On the last graph the leader was rotated from 0 to 80 degrees towards the leader in 10 degree increments. During the test the follower's camera captured the marker for 5 seconds.

Figure 5.5 shows measured values along the Y axis of the marker from the second test. These measured values depend on the distance of the marker from the camera. In order to describe this dependence a straight line was fitted using the least squares method. The height between the center of the marker and the camera was 8 cm.

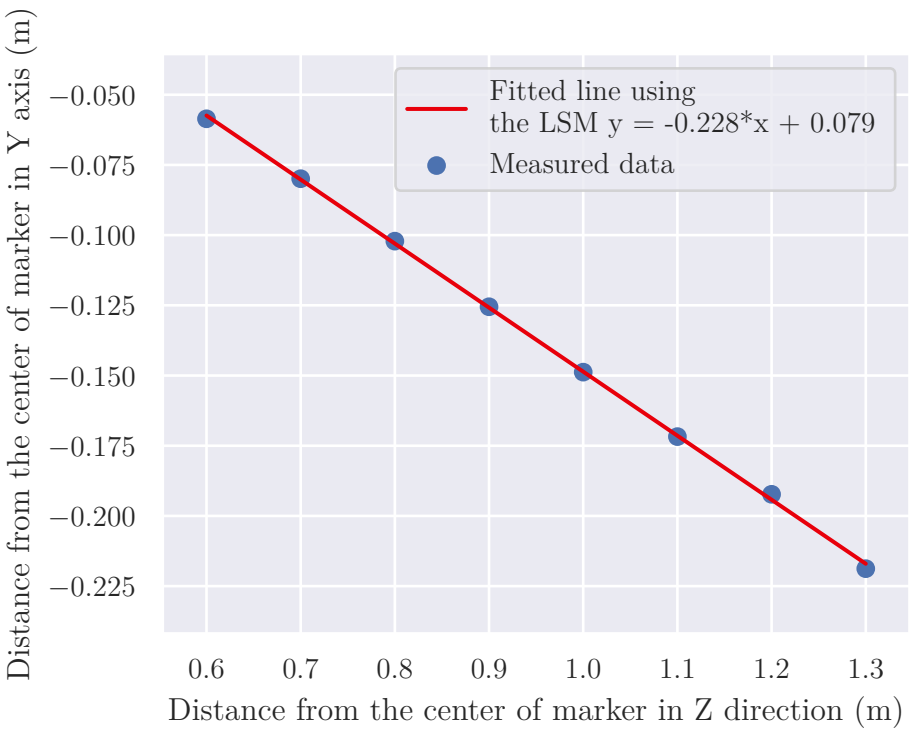

Figure 5.5: Height estimates between the center of ArUco marker and the camera of Tello drone.

Chapter 6
Kalman filter

"The problem of state estimation concerns the task of estimating the state of a process while only having access to noisy and/or inaccurate measurements from that process. It is a very ubiquitous problem setting, encountered in almost every discipline within science and engineering" [47].
A wide variety of filters such as g-h, Bayes, Particle, Kalman, and H-infinity filters can be used to tackle this problem. The most common one is the Kalman filter and it is the central concern for this chapter.

6.1 Objective

Although the marker pose estimation seems to be precise, it has a few issues. First, the marker could be easily occluded and thus the pose estimation will fail. Second, the camera output of the drone has noise and some frames could be corrupted, which also results in losing the target. Based on the measurements from the previous chapter, it is essential to emphasize that the marker pose estimation provides relatively accurate results, that is the main cause of the noise in pose measurements is the drones themselves, whose flights are accompanied by not negligible oscillations. This factor is amplified, given that both drones may contribute to the noise, depending on the correct operation of the VPS used to stabilize the drone.
The primary objective is being able to predict the marker pose when it is partly occluded or the camera stream becomes unavailable, and to reduce a noise in the marker's pose estimates.

6.2 Choice of Kalman filter type

The basic Kalman filter can be used only for linear state-space models. However, most real systems are nonlinear and modified versions of the Kalman filter were developed, such as an extended Kalman filter(EKF) and unscented Kalman filter(UKF). EKF allows using nonlinear models by computing a linear approximation at the point of the current estimate. UKF, on the other hand, samples a set of points with specific properties and passes it through a nonlinear function.

The basic Kalman filter with Newtonian motion model has been selected to track the marker due to its simplicity and computational ease.

6.3 Design Kalman filter

There are 2 phases involved in Kalman filter:

1. **Prediction**

   Predict next state

   $$\hat{\mathbf{x}} = \mathbf{F}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{6.1}$$

   where $\hat{\mathbf{x}}$ is predicted state at the next time step. $\mathbf{F}$ is a state transition function. $\mathbf{x}$ is a state estimate from the update phase. $\mathbf{B}$ is control function, which incorporates a control input $\mathbf{u}$ into the predicted state.

   Predict state covariance

   $$\hat{\mathbf{P}} = \mathbf{F}\mathbf{P}\mathbf{F}^{\mathbf{T}} + \mathbf{Q} \tag{6.2}$$

   where predicted state covariance $\hat{\mathbf{P}}$ shows how much uncertainty there is in the prediced state $\hat{\mathbf{x}}$. $\mathbf{Q}$ is a covariance of the process noise. Greater values would mean for the filter to favor measurements more rather than the dynamic model.

2. **Update**

   Compute Kalman gain

   $$\mathbf{K} = \hat{\mathbf{P}}\mathbf{H}^{\mathbf{T}}(\mathbf{H}\hat{\mathbf{P}}\mathbf{H}^{\mathbf{T}} + \mathbf{R})^{-1} \tag{6.3}$$

   where the Kalman gain $\mathbf{K}$ is a number between 0 and 1, which represents a ratio between the measurement and prediction. $\mathbf{H}$ is state to measurement space mapping. $\mathbf{S} = (\mathbf{H}\hat{\mathbf{P}}\mathbf{H}^{\mathbf{T}} + \mathbf{R})^{-1}$ is the innovation covariance. $\mathbf{R}$ is the measurement covariance.

   Compute state estimate

   $$\mathbf{x} = \hat{\mathbf{x}} + \mathbf{K}(\mathbf{z} - \mathbf{H}\hat{\mathbf{x}}). \tag{6.4}$$

   where $\mathbf{z}$ is measurement. $\mathbf{z} - \mathbf{H}\hat{\mathbf{x}}$ is a residual between the predicted value and measurement.

   Compute estimated state covariance

   $$\mathbf{P} = (\mathbf{I} - \mathbf{K}\mathbf{H})\hat{\mathbf{P}} \tag{6.5}$$

   where $\mathbf{I}$ is an identity matrix

A state is represented as 12 dimensional vector with x, y, z, yaw angle θ and its corresponding first and second derivatives:

$$\mathbf{x} = \begin{pmatrix} x & y & z & \theta & \dot{x} & \dot{y} & \dot{z} & \dot{\theta} & \ddot{x} & \ddot{y} & \ddot{z} & \ddot{\theta} \end{pmatrix}$$

Since the leader is not expected to fly at high speeds, roll and pitch angles are not monitored. From Newton's laws of motion, a state transition function is defined:

$$\mathbf{F} = \begin{bmatrix} 1. & 0. & 0. & 0. & dt & 0. & 0. & 0. & 0.5dt^2 & 0. & 0. & 0. \\ 0. & 1. & 0. & 0. & 0. & dt & 0. & 0. & 0. & 0.5dt^2 & 0. & 0. \\ 0. & 0. & 1. & 0. & 0. & 0. & dt & 0. & 0. & 0. & 0.5dt^2 & 0. \\ 0. & 0. & 0. & 1. & 0. & 0. & 0. & dt & 0. & 0. & 0. & 0.5dt^2 \\ 0. & 0. & 0. & 0. & 1. & 0. & 0. & 0. & dt & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 1. & 0. & 0. & 0. & dt & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. & 1. & 0. & 0. & 0. & dt & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. & 1. & 0. & 0. & 0. & dt \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 1. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 1. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 1. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 1. \end{bmatrix}$$

State to measurement space mapping is defined as:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Discrete white noise with a variance of 0.01^2 was used to model the process noise $\mathbf{Q}$. Based on results obtained from ArUco pose detection measurements and imperfect flight dynamics of Tello drones affecting pose estimation, the measurement noise $\mathbf{R}$ has been set to a variance of 0.02^2

After initialization, the filter is updated with the latest marker pose. Since there is no fixed rate at which poses are received the time step dt is calculated and the state transition matrix is updated accordingly. The whole pipeline is depicted in the figure 6.1

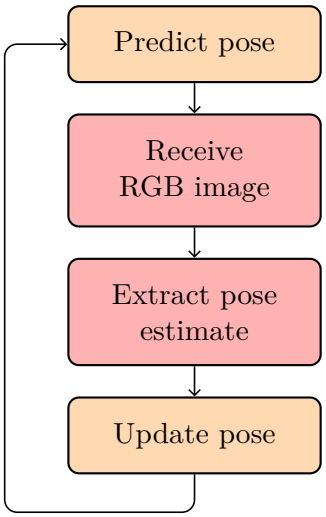

Figure 6.1: Kalman filter predict and update pipeline with pose measurements.

6.4 Adaptive filter

After evaluating the Kalman filter initialized according to the previous section, the major problem was a significant noise in the filter's output when the leader was flying with a constant velocity or hovering. This was consistent with the simulation, where no speed limit was set for safety reasons, and noise in the leader poses contributed to the acceleration of the follower, resulting in the drone starting to oscillate. The reason is the second-order system used to model the motion of the leader with attached marker, which performs well tracking maneuvers and a sharp increase in the velocity of the target. However, it falsely interprets the noise in measurements as an acceleration of the target. By contrast, the first-order motion model reduced noisy measurements but had a lag when the target started accelerating. It should be mentioned the second-order Kalman filter performs well with the real drones if there are no oscillations in a flight of both drones and the video stream from the follower's camera is not delayed or corrupted, which is hard to achieve due to many conditions — sufficient lightning, strong WiFi signal and minimal interference by other wireless networks, a flooring with non-regular patterns. The last condition is especially crucial since the VPS system uses the downward camera to stabilize the drone, and if it fails, the drone starts drifting.

One solution would be to use first- and second-order filters together by switching to a first-order filter when the target is flying at a constant speed and a last-order filter when maneuvering. There are multiple approaches based on using multiple filters, one of which is Interacting Multiple Models(IMM) [45]. Briefly, it allows defining a set of Kalman filters with different configurations for each mode of the target. Then the output of a filter producing better estimates is used to adjust the one which performed worse at the current step and the other way around. Finally, a weighted estimate of both filters' output is made.

This study uses a more straightforward solution called an adaptive filter with adjustable process noise presented in [45]. Instead of multiple filters, a single filter with a constant velocity model is used. The basic principle is based on the continuous adjustment of the process noise depending on the size of a residual defined as a difference between a measurement and prediction.

The residual is squared to avoid negative values and normalized by the covariance matrix such that significant changes are distinguishable from the measurement noise.

$$\epsilon = \mathbf{y}^{\mathbf{T}}\mathbf{S}^{-\mathbf{1}}\mathbf{y} \tag{6.6}$$

where

$\mathbf{y} = \mathbf{z} - \mathbf{H}\hat{\mathbf{x}}$ is a residual,

$\mathbf{S} = \mathbf{H}\mathbf{P}\mathbf{H}^{\mathbf{T}} + \mathbf{R}$ is covariance matrix representing the system uncertainty.

ϵ is the key value based on which the process noise is adjusted. Whenever ϵ exceeds some predefined limit caused by rapid increase of measurements

the process noise $\mathbf{P}$ of Kalman filter is gradually increased. After, when ϵ becomes smaller than the limit the process noise is again decreased. The code below depicts how the process noise is adjusted.

```
epsilon = square_and_normalize(residual)
if epsilon > epsilon_max:
       count += 1
       Process_noise *= noise_multiplier
else if count > 0:
       count -= 1
       Process_noise /= noise_multiplier
```

Listing 6.1: Source: [45]. Adjusting the process noise of the Kalman filter.

Constants $\epsilon\_max = 2$ and $noise\_multiplier = 100$ have been chosen empirically to compensate for noise and lag at the filter output.
The equation 6.6 has been modified to adjust the noise of the individual state variables subjected to a significant change:

$$\boldsymbol{\epsilon} = \mathbf{y} \odot \mathbf{S^{-1}}\mathbf{y} \tag{6.7}$$

where
$\boldsymbol{\epsilon}$ is a vector of squared and normalized residuals for each state variable

$\odot$ means elementwise vector multiplication
This modification was necessary because a drone can change position in different axes independently.

Finally, the state vector was reduced to position and velocity for each dimension, and the state transition with dimension space mapping was updated accordingly.

6.5 Results

To test the impact of the Kalman filter a test flight was recorded using **rosbag**. The main result is that the marker poses could be predicted when it is occluded and reduced noise in the output of the filter.

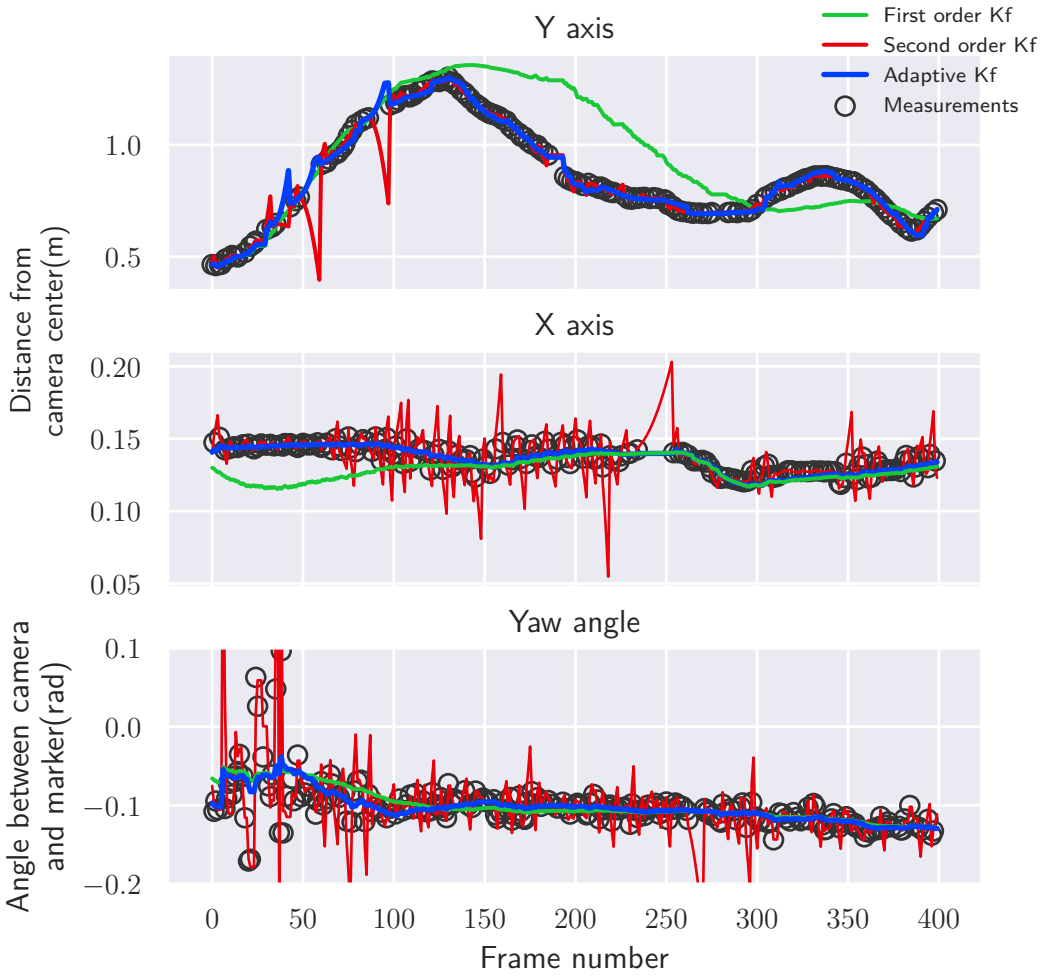

Figure 6.2: The stationary follower tracks the ArUco marker attached to the leader using 3 different Kalman filters. The leader was manually controlled with smooth movements in the y direction and short maneuvers in the x direction to simulate the bounces that occur for the reasons described at the beginning of the chapter.

According to the graph in Figure 6.2, the second-order Kalman filter performed well in tracking the marker in the y direction. This can be explained by the low amount of noise in the marker pose measurements along the y direction. However, when the marker was obscured, predicted second-order filter poses diverged too quickly from the actual pose. Measurements of yaw angles are very noisy due to imperfect flight and the second order filter treats noise as acceleration thus increasing noise even more. As a result, the control algorithm issues inefficient commands with frequent interruptions to rotate the drone to adjust the heading.
The first-order filter does an excellent job of handling the noise in measurements, but at the cost of lagging from the actual value. Both the first order and adaptive filters reduced noise of measured yaw angles.
The adaptive filter seems to combine both filters and produces the best estimate, although it will produce noise when the target accelerates rapidly.

To summarize, the first-order Kalman filter is best at reducing noise, but its output is delayed when the target accelerates, the second-order Kalman filter can track maneuvers but is heavily affected by measurement noise, and finally, the first-order Kalman filter with adjustable process noise attempts to combine the best of both filters - reducing measurement noise and minimizing the filter output delay when the target accelerates.

Chapter 7

Controlling the drone

Once the target's pose is estimated the drone has to smoothly navigate itself towards it preventing overshooting and undershooting. To achieve this a control loop system can be used.
According to [42] Control systems are classified into two general categories: open-loop and closed-loop systems. The dependence of the control action on the feedback is the distinction between these two categories.

1. **Open control loop** system is one in which the control action is independent of the output.
2. **Closed control loop** system is one in which the control action is somehow dependent on the output.

"Closed-loop control systems are more commonly called feedback control systems"[42]. They are well suited for the processes, where measurements are feasible, process disturbances are not rare and human intervention is not required allowing high degree of autonomy.

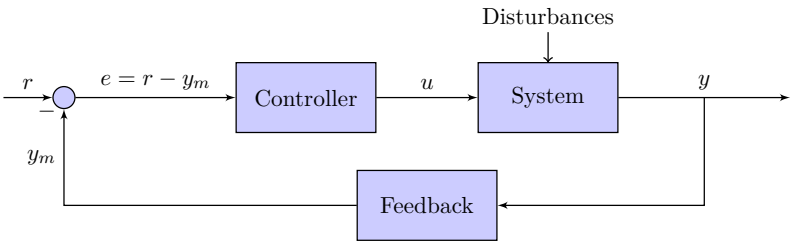

Figure 7.1: Closed loop system overview. e is the error between desired point r and the feedback y_m. u is the control output and y is the system output.

7.1 PID Controller

Since drone control is best suited to a closed-loop system, that is what is used in this project. Common example of the closed loop algorithms is a combination of **P,I,D controllers, further PID controllers**.
"The term controller in a feedback control system is often associated with

elements of the forward path, between the actuating(error) signal e and the control output variable u" [42].

P controller is proportional to the error:

$$u(t) = K_P e(t) \tag{7.1}$$

I controller is proportional to the integral of the error:

$$u(t) = K_I \int e(t)\,dt \tag{7.2}$$

D controller is proportional to the derivative of the error:

$$u(t) = K_D \frac{\mathrm{d}e}{\mathrm{d}t} \tag{7.3}$$

PID controller combines all three terms:

$$u(t) = K_P e(t) + K_D \frac{\mathrm{d}e}{\mathrm{d}t} + K_I \int e(t)\,dt \tag{7.4}$$

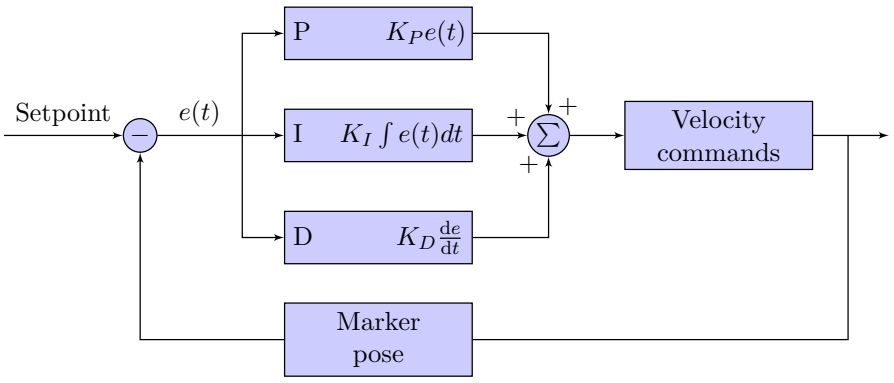

Figure 7.2: Overview of the PID Controller used to track the leader drone. The controller output is sent to the follower drone in the form of velocity command along an axis.

The PID controller is initialized for each of the control axes - x, y, z, θ. In order to achieve desired performance and stability of the PID controller, it has to be tuned first. Tuning the controller means selecting optimal gain constants used in P,I,D terms - K_P, K_I, K_D. There are several methods to tune the controller. In this work the trial and error method used, which involves manual calibration and testing the impact on the real model. Other methods propose less manual work and possible more optimal results, but require additional information of the drone's configuration model.

Control axis	K_P	K_I	K_D
x	1.5	0.001	0.5
y	1.5	0.001	0.5
z	1.5	0.001	0.5
θ(Yaw angle)	1	0.0001	0.5

Table 7.1: Empirically derived PID controller constants for each control axis of real Tello drone.

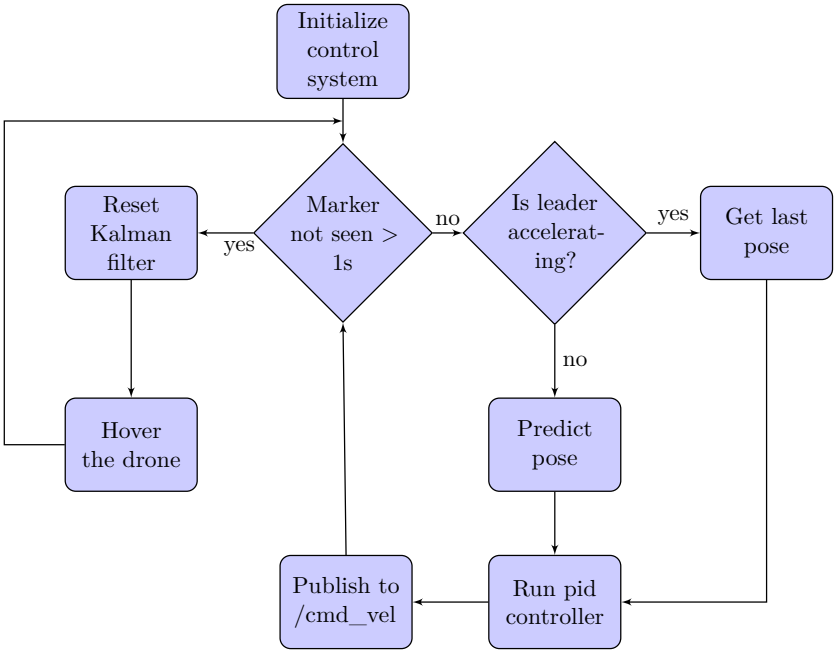

Figure 7.3: High level view of the main control loop. The loop is set to run every 100 ms.

7.2 Overview

To control the follower, a loop with a period of 100 ms was created. According to the figure 7.3, if the marker is not observed for more than 1 second, it is considered lost, and the follower must stop moving in either direction and hover waiting for the leader to reappear. Whenever the leader is marked as lost, the Kalman filter and the PID controllers will be reset. To compensate for the delay from camera stream and marker detection, the pose at the next time interval is predicted using the Kalman filter. This also solves the problem when the marker is temporarily occluded. The prediction step is canceled if the leader accelerates and the last detected pose is used instead to prevent collisions, since drones mostly operate indoors. Once the final marker pose is obtained, it is passed to the PID controllers to calculate velocities to achieve the desired pose relative to the leader. It is required for the follower to be 70 cm away from the leader in the y direction, 0 cm in the x direction, and at a 0 radian angle between the camera and the marker. The

height between the follower's camera and the marker center is set according to the fitted line in the figure 5.5. This configuration allows to keep drones at a safe distance, avoid obstacles by following the leader's trajectory, and add more follower drones using different markers.

Tello accepts velocity commands in a range from $[-1, 1]$. To convert the PID controller output to the acceptable velocity command the hyperbolic tangent function is used.

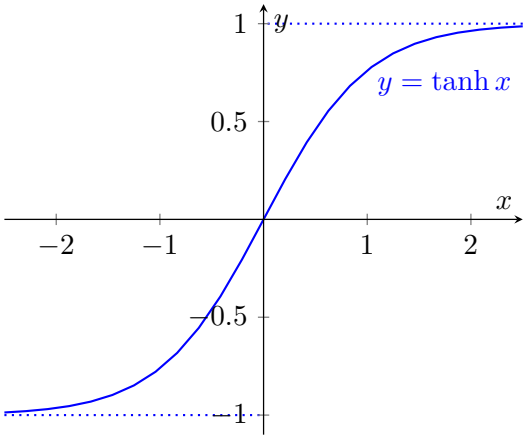

Figure 7.4: Hyperbolic tangent function

For safety reasons, the function output is limited to an interval of -0.5 to 0.5 to prevent the follower from accelerating too fast and colliding with obstacles. Once the correct velocities are calculated, they are applied all at once, since a drone is able to fly in all directions independently of each other.

Chapter 8

Implementation

Since the simulation and the ROS driver of Tello require separate distributions of ROS - Melodic and Noetic, respectively, they are separated into two distinct workspace folders. All software implemented in the thesis is provided in a ROS package named "follower". Each workspace has its own copy of the package due to differences in Python version and control interface of the simulation and real Tello.
The structure of the follower package is shown below. Some parts may vary in the package used for Tello drone simulation, but the underlying logic is the same:

- launch
  - tello.launch used to start a simulation or connect the Tello ROS driver.
  - follower.launch used to launch nodes for detection and following the leader with ArUco marker by the follower.
  - single.launch is a single launch file used to start both tello and follower launch files. To visualize the leader and follower the tf2_odom_broadcaster.py and tf2_aruco_broadcaster.py are started to define transformations between map, odometry and leader frames. Finally, rviz node will show position and orientation of drones. In a simulation ground truth values are used to visualize poses of drones and additional node is started, which takeoffs and generates a circular movement for the leader.
- msg
  - Pose3D.msg is a message definition for the filtered poses from Kalman filter used to send via a ROS topic to the control node.
- rviz contains a configuration file for rviz with plugins used for the visualization.
- src contains Python scripts defining ROS nodes:
  - flight_control.py defines a node for controlling the drone

- aruco_filter.py defines a node for filtering poses of the ArUco marker using the Kalman filter from previous chapter.
- tf2_odom_broadcaster.py defines a node for publishing corrected odometry transformations. This node is borrowed from David Pařil's bachelor thesis [48]. Not presented in the simulation package.
- tf2_aruco_broadcaster.py defines a node for publishing transformations between the odometry and the ArUco marker poses. Not presented in the simulation package.

- ROS package specific files used to define dependencies and declarations.

`https://youtu.be/n8WVKc7n2FE` contains recorded screen with complete simulation of Tello drones.

8.1 Assesment

To assess the performance of the algorithm, a test flight was conducted in the simulator. Ground truth positions of both the leader and follower drones were collected.
The leader launched from position (-5, 0, 0) and the follower from (-5,-1,0). Desired distances of the follower from the leader were 0 cm in X axis, 0 cm in Z axis(Y axis in the marker frame) and 70 cm in Y axis(Z axis in the marker frame). Figure 8.1 depicts a 3D flight path in the simulator, and Figure 8.2 shows a more detailed view with individual axes.

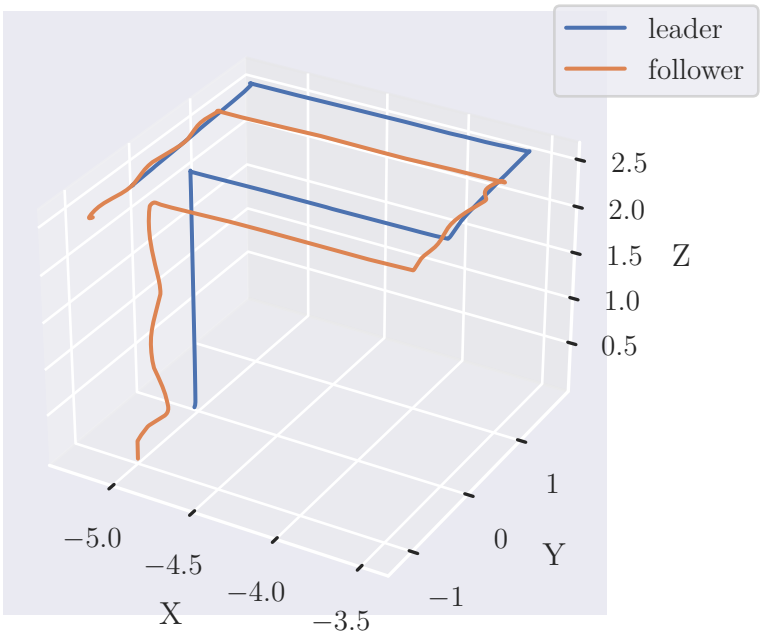

Figure 8.1: Recorded route of the leader and follower in the simulator.

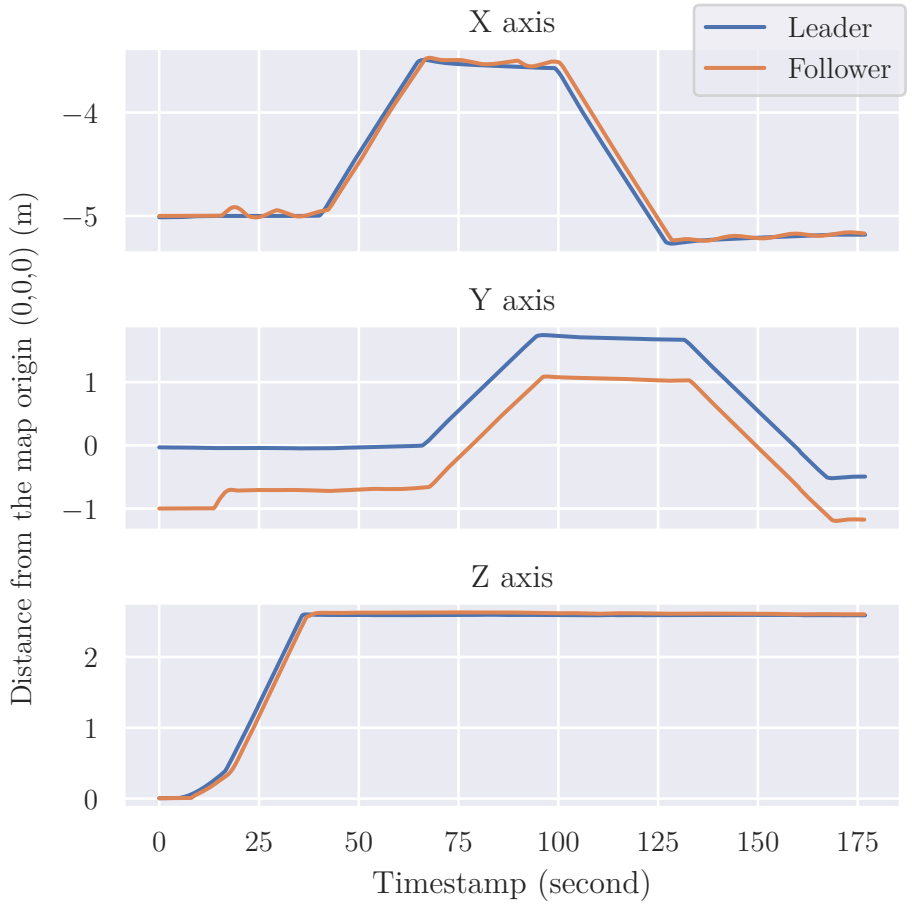

Figure 8.2: Detailed view of the flight from the previous graph.

Based on the ground truth values of both drones from the described test flight and desired distance in each axis mean absolute errors(MAEs) were calculated:

Axis	MAE (cm)
X	4.4
Z	7
Y	4

Multiple flights with real drones were carried out, however, it was not possible to quantitatively assess the performance of the algorithm because a global positioning system such as Vicon [37] was not available and the odometry data are highly inaccurate. To demonstrate the algorithm, a flight with real drones was recorded, available at `https://www.youtube.com/watch?v=V-9W5hL1Efo`.

Chapter 9

Conclusion

To summarize, the Tello drone was presented with its main technical characteristics. Tello ROS driver [33] was used to communicate with the drone in the ROS [16] ecosystem.
System containers were used to isolate the main operating system from the development and to quickly switch between different ROS distributions. This is a lightweight solution that allows the graphical output of tools such as Rviz [22], Gazebo Simulator [7], and RQT [20].

The only way to locate the leader is by detecting from the camera image. In the beginning, colored markers were used to achieve this by attaching them to the leader and allowing it to be identified. However, this was unsuccessful because of the many incorrect identifications, which required perfect illumination in the environment to solve. Instead, it was decided to use ArUco markers [2], which are relatively easy to identify and have fewer false positives compared to colored markers. These markers also provide a pose estimation with 6 degrees of freedom, which allows for determining the position of the leader relative to the follower.
Measurements were taken to determine the accuracy of the marker pose estimations, which were subsequently used to set the measurement noise in the Kalman filter. In most measurements, the estimation error did not exceed 1 cm.

A simulation model of the Tello drone with similar characteristics as the real version was created. The simulation was based on the hector_quadrotor [8] package, which contains many modules, including a set of sensors to extend the drone, flight controllers, and plugins that use physical engines to simulate the drone. Next, the Tello model was augmented with an ArUco marker, which acts as the leader drone. Although the flight in the simulator is not entirely identical to the real one, it allows the vast majority of development to take place in the simulator, which is a safer and more convenient solution when working with drones.

A Kalman filter was used to smooth out the noise in the marker pose estimations caused by most non-ideal drone movements, as well as the prediction

of the pose in the case of marker occlusions.
Initially, a Kalman filter was proposed with a second-order state transition that considers the acceleration of the following target. This made it possible for the follower to react quickly to abrupt changes in the leader's pose. A significant drawback was that the filter interpreted the noise as acceleration, which could lead to oscillations and collisions. To solve this problem, an adaptive first order filter with adjustable process noise presented in [45] was used. The outputs of the filters with different configurations were compared at the end of chapter 7, where the adaptive filter showed the best results.

A flight in the simulator was carried out to evaluate the performance of the algorithm, and ground truth values were used to obtain a quantitative assessment. The algorithm was also verified in a real drone flight, and its behavior was robust and predictable.

Appendix A

Bibliography

[1] Application containers vs system containers. https://linuxcontainers.org/lxd/introduction/#application-containers-vs-system-containers. Accessed: 2022-14-04.

[2] Aruco markers. https://docs.opencv.org/3.1.0/d5/dae/tutorial_aruco_detection.html. Accessed: 2022-03-03.

[3] Aruco pose. https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html. Accessed: 2022-05-03.

[4] aruco_detect. http://wiki.ros.org/aruco_detect?distro=melodic. Accessed: 2022-01-05.

[5] Camera calibartion. https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html. Accessed: 2022-05-03.

[6] Docker vs virtual machines (vms) : A practical guide to docker containers and vms. https://www.weave.works/blog/a-practical-guide-to-choosing-between-docker-containers-and-vms. Accessed: 2022-14-04.

[7] Gazebo simulator. https://gazebosim.org/home. Accessed: 2022-15-04.

[8] Hector_quadrotor package. http://wiki.ros.org/hector_quadrotor. Accessed: 2022-01-02.

[9] Installing ros in lxd containers. https://ubuntu.com/blog/installing-ros-in-lxd. Accessed: 2022-09-05.

[10] Links and joints. http://library.isr.ist.utl.pt/docs/roswiki/urdf(2f)Tutorials(2f)Create(20)your(20)own(20)urdf(20)file.html. Accessed: 2022-01-01.

[11] Lxd. https://linuxcontainers.org/#LXD. Accessed: 2022-14-04.

[12] Plotjuggler. https://www.plotjuggler.io/. Accessed: 2022-16-05.

[13] Projection. https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html. Accessed: 2022-04-05.

[14] Robomaster tt. https://www.dji.com/cz/robomaster-tt. Accessed: 2022-10-05.

[15] Robomaster tt expansion kit. https://store.dji.com/cz/product/robomaster-tt-expansion-kit. Accessed: 2022-04-05.

[16] Ros. https://www.ros.org/blog/why-ros/. Accessed: 2022-01-05.

[17] Ros kinetic kame. http://wiki.ros.org/kinetic. Accessed: 2022-16-05.

[18] Ros melodic morenia. http://wiki.ros.org/melodic. Accessed: 2022-16-05.

[19] Ros noetic ninjemys. http://wiki.ros.org/noetic. Accessed: 2022-16-05.

[20] Rqt. http://wiki.ros.org/rqt. Accessed: 2022-16-05.

[21] Running x11 software in lxd containers. https://blog.simos.info/running-x11-software-in-lxd-containers/. Accessed: 2022-01-05.

[22] Rviz. http://wiki.ros.org/rviz. Accessed: 2022-11-05.

[23] Ryze technology. https://www.ryzerobotics.com/about. Accessed: 2022-02-03.

[24] Ryze tello. https://www.dronekenner.nl/images/ab__webp/detailed/7/DJI-drones-kopen-online-dronekenner-Tello-Ryze-fun-camera-VR-headset-10306-99_f1uu-j3_449d-89_jpg.webp. Accessed: 2022-02-03.

[25] Ryze tello 3d model. https://www.halfchrome.com/downloads/dji-tello-cad-model/. Accessed: 2022-01-03.

[26] Ryze tello boost combo. https://www.robotworld.cz/dji-ryze-tello-boost-combo. Accessed: 2022-04-05.

[27] Ryze tello iron man edition. https://www.ryzerobotics.com/ironman. Accessed: 2022-04-05.

[28] Ryze tello sdk 1.3.0. https://terra-1-g.djicdn.com/2d4dce68897a46b19fc717f3576b7c6a/Tello%20%E7%BC%96%E7%A8%8B%E7%9B%B8%E5%85%B3/For%20Tello/Tello%20SDK%20Documentation%20EN_1.3_1122.pdf. Accessed: 2022-04-03.

[29] Ryze tello specs. https://www.ryzerobotics.com/tello/specs. Accessed: 2022-01-01.

[30] Simulation description format (sdf). http://sdformat.org/. Accessed: 2022-14-04.

[31] Tellopilots comunity forum. https://tellopilots.com/. Accessed: 2022-12-05.

[32] Tellopy. https://github.com/hanyazou/TelloPy. Accessed: 2022-02-03.

[33] tello_ros1. https://github.com/appie-17/tello_driver. Accessed: 2022-15-05.

[34] tello_ros2. https://github.com/clydemcqueen/tello_ros. Accessed: 2022-15-05.

[35] Thingverse ryze tello. https://www.thingiverse.com/tag:tello. Accessed: 2022-10-05.

[36] Unified robot description format (urdf). http://wiki.ros.org/urdf. Accessed: 2022-14-04.

[37] Vicon - award winning motion capture systems. https://www.vicon.com/. Accessed: 2022-16-05.

[38] What are containers. https://www.netapp.com/devops-solutions/what-are-containers/. Accessed: 2022-14-04.

[39] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018.

[40] Bart Custers. *Drones Here, There and Everywhere Introduction and Overview*, volume 27, pages 10–11. 10 2016.

[41] Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco Madrid-Cuevas, and Manuel Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47:2280–2292, 06 2014.

[42] I. J. Williams J. J. DiStefano, A. R. Stubberud. *Feedback and control systems*. McGraw-Hill, 1967.

[43] Lentin Joseph. *Robot Operating System (ROS) for Absolute Beginners*. Apress, Berkeley, CA, 2018.

[44] Anis Koubaa. *Robot Operating System (ROS)*. Springer, Cham, 2017.

[45] Roger Labbe. Kalman and bayesian filters in python, chapter 14 - adaptive filtering. https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python/blob/master/14-Adaptive-Filtering.ipynb, 2020.

[46] Johannes Meyer, Alexander Sendobry, Stefan Kohlbrecher, Uwe Klingauf, and Oskar von Stryk. Comprehensive simulation of quadrotor uavs using ros and gazebo. In *3rd Int. Conf. on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR)*, 2012.

[47] Fredrik Orderud and Sem Sælands. Comparison of kalman filter estimation approaches for state space models with nonlinear measurements. 2005.

[48] David Paril. *Autonomous Control of Drone Ryze Tello*. Czech Technical University in Prague. Computing and Information Centre., 2021.

Appendix B

Attachment

Below is the contents of the attached source code.

- tello_real - ROS workspace folder with packages for real Tello drone
  - readme.md - description of how to build and run the source code in src folder
  - src - ROS packages with simulation and implemented algorithms for tracking and following.
- tello_simulation - ROS workspace folder with ready to use simulation of Tello drones.
  - readme.md - description of how to build and run the source code in src folder
  - src - ROS packages with Tello ROS driver and implemented algorithms for tracking and following.