Development of a UAV Trajectory Control System

MASTER'S THESIS

By the 2nd year Master's student, Group 273, Faculty of Computer Science and Information Technologies
Field of Study: 02.04.03 – Mathematical Support and Administration of Information Systems
Faculty of Computer Science and Information Technologies
Ekaterina Yuryevna Voronina

Academic Supervisor:
Associate Professor of the Department of Computer Science and Legal Informatics
_____   E. V. Kudrina

Head of the Department:
Ph.D. in Physics and Mathematics, Associate Professor   _____   M. V. Ogneva

Saratov
2025

Contents

Notations and Abbreviations

UAV	unmanned aerial vehicle
BA	unmanned aviation
APC	hardware and software complex
ACS	automatic control system
GNSS	global navigation satellite system
MC	microcontroller
TCS	trajectory control system
CS	coordinate system
NPU	ground control station
BAU	onboard control equipment
NAU	ground control equipment
APIC	analytical software-invariant design
SVS	air signal system
BTS	unmanned transport system
IMU	inertial measurement unit
RPM	(Revolutions Per Minute) a measure of rotational speed

Introduction

The development of unmanned aviation (UA) is a priority area for state industrial policy. Following the Russian government's approval of a strategy for the development of unmanned aviation until 2030 and beyond to 2035 [1], the UA sector is currently receiving significant financial support from the state.

The growing interest in unmanned aviation is a global trend. This is happening for a number of reasons. Unmanned aerial vehicles (UAVs) are generally much cheaper than manned aircraft and helicopters. Operator training is cheaper than pilot training. The absence of a human on board eliminates the need for life support systems, reduces the weight and dimensions of the aircraft, and allows for an increase in the range of permissible overloads. In addition, the loss of unmanned aircraft does not result in human casualties, which increases the overall level of operational safety.

Unmanned aerial vehicles can be divided into two types: operator-controlled and autonomous, where the piloting functions are completely entrusted to the control system. The human brain effectively processes visual and other sensory data, allowing it to recognize objects and situations. However, due to limited reaction speed, attention, and memory, humans do not always provide reliable control.

In recent years, autonomous UAVs have significantly increased their reliability and are performing an increasingly wide range of tasks in various industries. However, they still do not use their full physical potential. Most of these aircraft fly at low speeds, close to hovering, to ensure stable operation of perception and obstacle avoidance algorithms [2, 3].

Improvements in autonomous control systems in terms of speed and maneuverability allow for expanding the capabilities of UAVs: increasing flight range and speed, improving the ability to avoid fast-moving objects, and ensuring control in complex, confined spaces. Despite progress, autonomous UAVs still lag behind humans in terms of versatility and reliability, which requires further research to bridge this gap.

Rotary-wing UAVs (quadcopters) are among the most dynamic platforms in unmanned aviation. Thanks to their maneuverability, they are used in a wide variety of scenarios: from flights in dense forests and urban areas to monitoring fire-hazardous areas, infrastructure inspection, logistics, and agriculture [2, 3].

One of the most difficult and demonstrative maneuvers for such systems is an acrobatic flip — a 360° rotation around one of the axes. Such a maneuver is in demand not only in demonstration flights, but also in applied tasks — for example, when sharply evading obstacles, changing trajectory in confined spaces, when calibrating navigation systems (e.g., when inverting the orientation of an antenna or sensor), and when controlled recovery from instability caused by external disturbances or aggressive maneuvers. Successful execution of a flip requires precise control of angular velocities and thrust, taking into account the current state of the drone and its dynamic limitations.

While popular open-source solutions, such as the Clover platform [4], implement flips

using a rigidly defined sequence of commands—for example, by setting a fixed angular velocity for a limited time—such approaches effectively ignore the dynamics of the system and do not use feedback. They do not adapt to different hardware configurations, which reduces the accuracy and reliability of the maneuver.

The goal of this work is to develop a trajectory control system for a quadcopter, in which a flip is considered as a controlled motion taking into account the full dynamic model of the quadcopter. This goal determined the following objectives:

1. To review the history of UAV development.

2. To describe the architecture of the onboard hardware and control system of a UAV.

3. To present the mathematical model of a rotorcraft-type UAV.

4. To study the algorithmic foundations of UAV motion control.

5. To provide an overview of the tools used for the development of UAV trajectory control systems.

6. To learn how to obtain and process data from quadcopter sensors.

7. To develop a dynamic model of the quadcopter and implement a flip controller based on it.

8. To verify the controller in a simulation environment.

Chapter 1

Theoretical Part

1.1 History of UAV Development

UAVs are one of the fastest growing and most revolutionary technologies in the modern aerospace industry. Their history dates back to the early 20th century, with the first attempts to create devices for specific tasks. UAV prototypes appeared during World War I, when the British developed the Kite Balloons device for pilot training. However, the real breakthrough in the development of unmanned technology occurred in the 1920s in the United States with the creation of the Radioplane OQ-2 under the leadership of Howard Hughes, which became the first radio-controlled unmanned aircraft used to train artillerymen.

Evolution of management systems

The development of UAVs is closely linked to advances in control systems. Initially, control was carried out manually via radio channel or implemented in the form of a rigidly defined trajectory, while stabilization was achieved using mechanical gyroscopes. In the 1950s–1970s, analog autopilots appeared, ensuring course and altitude maintenance. With the development of microprocessors and GPS in the 1980s and 1990s, digital autopilots using PID controllers, inertial measurement systems, and sensor fusion algorithms became widespread. Starting in the 2000s, more complex control methods began to be introduced: adaptive, robust, and predictive. Algorithms for planning trajectories taking into account dynamic constraints and obstacles were also developed. Since the early 2010s, the focus has shifted to autonomous navigation: the integration of SLAM, visual odometry, machine learning methods, and distributed control for swarm coordination. Modern UAVs are capable of adapting to dynamic environments, performing localization tasks, avoiding obstacles, and interacting with other agents in real time.

Quadcopters are classified according to their design and levels of autonomy [3]:

- Multirotor quadcopters, with four rotors, are the most popular configuration;

- Hybrid quadcopters, combining the properties of multirotor and winged aircraft, offer flexibility in operation;

In terms of autonomy, quadcopters can be:

- manually controlled, with remote control;

- semi-autonomous, capable of performing certain tasks without constant operator intervention;

- autonomous, equipped with navigation systems and sensors for independent performance of complex tasks.

Thus, the evolution of UAVs from simple radio-controlled models to high-tech autonomous systems with artificial intelligence and complex sensor technologies demonstrates rapid growth and broad prospects for development in a wide variety of fields.

1.2 Architecture of Onboard Hardware and UAV Control System

in Figure 1.1 shows the architecture of the onboard hardware device (OHD) of the UAV.
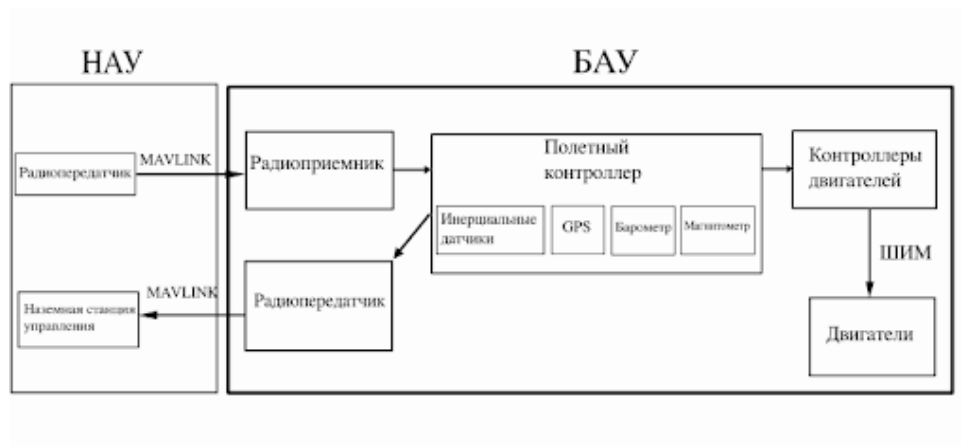

Figure 1.1: UAV on-board hardware device [3, 5]

The flight controller is the central control element in the system. It receives commands from the ground station or control panel and processes data from inertial and navigation sensors. The main task of the flight controller is to stabilize the position and implement autonomous control in real time.

Sensor modules

The inertial measurement system (IMS) includes an accelerometer and a gyroscope.

- An accelerometer provides relative acceleration, i.e. the difference between true acceleration and gravity. It can be one-, two- or three-component depending on the number of measurement axes. The principle of operation is based on the displacement of the inertial mass of the sensor and the conversion of this displacement into an electrical signal.

- Gyroscope registers angular velocity. The basis of operation of the mechanical is a rotor, the axis of rotation of which is free in one or more planes. Microelectromechanical (MEMS) gyroscopes with a size of several millimeters have a similar device [5]. Gyroscopic sensors allow tracking the UAV rotation in space, and their sensitive elements provide an output signal proportional to the angular velocity.

The IIS provides the data needed to calculate flight dynamics. However, due to gyroscope drift and accelerometer noise, the data requires filtering. For this purpose, Kalman filters or PID control are used [6-8].

Navigation systems

Standard GNSS is vulnerable to data spoofing, so modern navigation is often based on visual odometry or SLAM approaches that utilize data from cameras and IIS. In [9], the combination of GNSS and IIS improved the localization accuracy of UAVs. And [10, 11] describe navigation approaches based on the fusion of ANN and optical sensors.

A barometer measures atmospheric pressure and is used to determine altitude. There are several systems of altitude measurement:

- True - from the surface of the earth;

- relative - from an arbitrary level (e.g. runway);

- echelon - from a standard level of 760 mm Hg;

- absolute - from sea level.

Barometer measurements are affected by weather conditions, so ultrasonic, optical, laser and radio wave rangefinders are used to compensate for errors.

Sensor data processing: filtering and error compensation To ensure stable and accurate trajectory following by a quadrocopter in real conditions, it is necessary to take into account the non-idealities of the environment and dynamic disturbances such as side wind, turbulence, uneven load, and sensor errors. A quadrocopter is an inherently nonlinear, multi-input and unstable system with high sensitivity to external disturbances. Under such conditions, simple open control systems are insufficient.

The use of PI and PID controllers is necessitated by the need:

- suppression of oscillations and fast compensation of deviations due to maneuvers or external disturbances;

- stabilization of Euler angles and heights with minimum overshoot and transient time;

- accounting for accumulated systematic errors, e.g. due to center-of-mass drift or slow drift of sensors;

- stable control in the presence of actuator lags and inertial oscillations;

- maintaining stability in the presence of active feedback, especially in closed-loop applications.

Due to their simplicity, predictability and adaptability to specific flight conditions, PI/PID controllers remain the standard in UAV control systems.

For more complex tasks, such as correcting deviations to account for dynamic changes, the use of filters to process the data is an important element. One such filter is the extended Kalman filter (EKF), which is actively used for localization and stabilization of UAV motion. The EKF allows to integrate data from different sensors and obtain more accurate information about the position and orientation of the vehicle, which is critical for the correct functioning of the controllers.

EKF can be used to efficiently process noisy data and obtain approximated values for a more accurate assessment of the UAV's state, which in turn improves the performance of PI and PID controllers, increasing the overall stability of the control system.

Thus, the combination of filters, such as EKF, and controllers (PI, PID) allows for high accuracy, stability and adaptability of the quadrocopter control system, which is especially important in dynamically changing external conditions.

The BAU also includes:

- radio modules (for telemetry and control);

- electric motors and their controllers;

- lithium-polymer batteries;

- DC/DC converters and power systems.

Flight controller The automatic control system (ACS) implemented in the flight controller is a software and hardware complex designed to ensure stable, controlled and autonomous flight of the UAV. It performs the following key functions [12-15]:

- Stabilization of position in space - based on data from inertial sensors, barometer, and, if available, data from external navigation (GPS/GLONASS/SLAM). The control system realizes cascade control loops by Euler angles or quaternion using PID controllers or more complex control algorithms.

- Autonomous flight by specified GPS points - the route is specified in advance as a set of coordinates in the world system. The flight controller interprets the route, calculates navigation errors and generates a control action to follow the trajectory, including turns, heading hold, wind correction and altitude correction.

- Return-to-start mode - activated when communication with the operator is lost, when the mission time expires, when the battery level drops below the threshold, or on command. In this mode, the controller stabilizes the vehicle, raises it to a specified safe altitude and returns on a reverse trajectory, after which it performs a landing or hover.

- Hover and landing modes - provided by closed-loop control loops for altitude, attitude, and velocity. The controller holds the vehicle at a fixed point in space or smoothly descends it vertically. If a camera or rangefinder is available, landing with visual or radiometric marker guidance is possible.

- Radio control - the control system converts operator commands (e.g. from a transmitter with SBUS, PPM, iBUS protocols) into control actions directly affecting PWM signals of motors and servos. This makes it possible to realize manual, semi-autonomous and fully autonomous control.

Thus, the flight controller does not just stabilize the UAV, but also performs low-level actuator control, high-level mission planning and control, and serves as an interface between the hardware and the operator.

Most current flight controller solutions such as INAV[12], Betaflight[13], ArduPilot[14] utilize microcontrollers (MCs) of the STM32 family from STMicroelectronics. ARM Cortex-M based MCUs provide an optimal balance between performance, power efficiency and peripheral support. With I2C, SPI, UART, PWM, CAN and USB interfaces, as well as a wide range of available models in terms of frequency and memory capacity, the STM32 remains the de facto standard in the small UAV industry. At the same

time, more resource-intensive platforms, such as PX4[15] or autonomous VTOL systems, use more powerful processors, including ARM Cortex-A, as well as, on Linux-based systems - Raspberry Pi, NVIDIA Jetson, BeagleBone or hybrid solutions with coprocessors. Such architectures enable the implementation of visual navigation algorithms, SLAM and onboard AI, which goes beyond the typical capabilities of flight controllers.

Modern drone control systems require the integration of multiple components such as sensors, navigation systems and actuators. A unified data communication interface is required to ensure efficient communication between them. One such interface that is widely used in the industry is UAVCAN.

UAVCAN (Uncomplicated Application-level Vehicular Communication And Networking) is a data communication protocol adapted for use in UAV control systems and other robotic systems. It is built on principles similar to the standard CAN (Controller Area Network), but is designed with specific aerospace and robotics applications in mind [16]. Importantly, flight controllers such as the PX4 and Pixhawk actively support the UAVCAN interface, allowing the integration of various UAV components into a single, scalable and robust system.

Key features of UAVCAN include:

- Modularity, allowing new devices to be easily added to the system, especially important for flexibility in the design and operation of complex systems.

- Reliable data transmission, critical for operation in environments with limited connectivity or potential interference.

- Broad compatibility, supporting a variety of devices, from sensors to actuators, making the system scalable and adaptable.

- Efficiency in minimizing latency and data loss, which is especially important for real-time UAV operations.

- Ease of integration, thanks to open specifications and documentation.

The UAVCAN protocol uses a publish-subscribe architecture, allowing devices to publish data or subscribe to messages from other components, which facilitates dynamic communication in the network without strictly binding to specific addresses [16]. This makes it a convenient and powerful tool for developing state-of-the-art highly reliable UAV control systems. With PX4 and Pixhawk systems actively supporting UAVCAN, the implementation of this protocol in future UAV designs is becoming increasingly relevant, providing high flexibility, reliability and performance.

1.3   Mathematical Model of Rotary-Wing UAV

A quadrocopter is a highly maneuverable aircraft, but its dynamics is characterized by low stability because it is subject to significant external disturbances. In this regard, the quadrocopter control system should solve the problems of angular and spatial stabilization, ensuring the specified altitude, as well as maintaining the state of hovering, landing and flight along a predetermined trajectory [6-8]. The control laws are worked out by methods of mathematical, simulation and computer modeling. To solve this problem, it is necessary to understand the principles of quadrocopter operation, flight mechanics and the availability of a mathematical model describing its dynamics. Kinematic and

dynamic models serve as a basis for analyzing the UAV behavior in different conditions, which allows optimizing control algorithms and increasing the accuracy of the specified trajectories. The kinematic model of the quadrocopter does not take into account external forces acting on its motion. Changes in the drone's position, velocity and orientation in space are considered. A detailed description of the generalized kinematic scheme of a quadrocopter is given in [6]. In this paper, two main types of motion are distinguished:

- Progressive motion is the movement of the quadrocopter's center of mass in an inertial reference frame, in this case relative to the Earth. Such motion is described by the coordinates and velocity of the quadrocopter in space.

- Rotational motion of the quadrocopter characterizes its orientation relative to the inertial coordinate system and can be represented using Euler angles, quaternions, or the Directed Cosine Matrix (DCM).
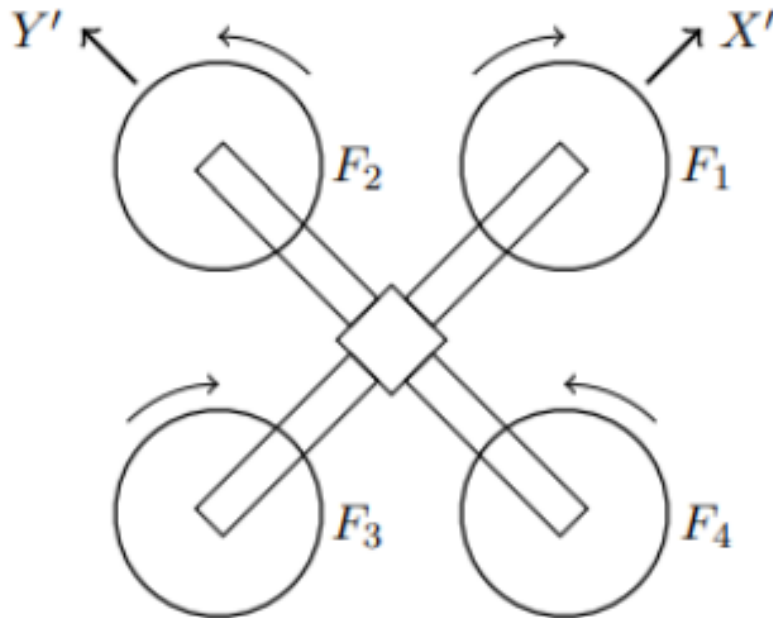

Figure 1.2: Schematic representation of the quadrocopter [?, ?]

In this paper, the design of the quadrocopter, as shown in Figure 1.2, which has a symmetric configuration, and its kinematic diagram, shown in Figure 1.3, will be considered.

In the $OXYZ$ coordinate space, the flight direction of the quadrocopter is defined by the OX axis. Two pairs of rotating motors, located at equal distances L_1 and L_2 from the origin O, are oriented along the axes OL_1 and OL_2, respectively. The tilt angles of the quadrocopter's arms relative to the positive direction of the OX axis are α_1 and α_2, with $\alpha_1 > \alpha_2$. The unit vectors of these axes OL_1 and OL_2 are denoted by $\boldsymbol{l}_1$ and $\boldsymbol{l}_2$, where

$$\boldsymbol{l}_1 = \begin{pmatrix} \sin \alpha_1 \\ \cos \alpha_1 \end{pmatrix}, \quad \boldsymbol{l}_2 = \begin{pmatrix} \sin \alpha_2 \\ \cos \alpha_2 \end{pmatrix}. \tag{1.1}$$
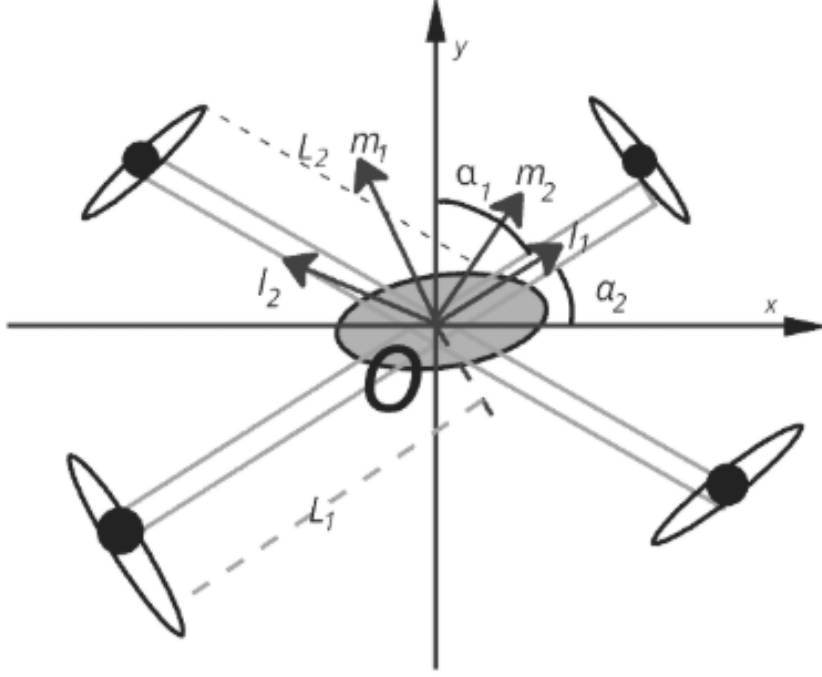
Figure 1.3: Kinematic diagram of the quadrocopter

The transformation matrix R_{BL} from the OXY axes to the OL_1L_2 axes, associated with the propeller placement, has the form:

$$R_{BL} = \begin{pmatrix} \sin\alpha_1 & \sin\alpha_2 \\ \cos\alpha_1 & \cos\alpha_2 \end{pmatrix}. \tag{1.2}$$

The determinant of the transformation matrix is defined as:

$$\det(R_{BL}) = \sin(\alpha_2 - \alpha_1). \tag{1.3}$$

For the scheme shown in Figure 1.3, we assume $L_1 = L_2$, $\alpha_1 = 45°$, and $\alpha_2 = 135°$.

Since the propellers are mounted directly on the motor shafts, it can be approximately assumed that the torque M_i of the i-th motor, generated by thrust T_i, is determined by the relation:

$$M_i = kT_i, \quad i = 1, 2, 3, 4, \tag{1.4}$$

where k is a design parameter.

The total lift force u_z along the vertical axis OZ of the quadrocopter is given by:

$$u_z = T_1 + T_2 + T_3 + T_4, \tag{1.5}$$

The yaw moment about the OZ axis is defined by:

$$u_\psi = -(T_1 - T_2 + T_3 - T_4) = -k(T_1 - T_2 + T_3 - T_4), \tag{1.6}$$

The moments M_1 and M_2 about the OX and OY axes are generated by the difference in thrusts of oppositely acting motors, where the moments of each motor pair lie in the OXY plane and are given by:

$$M_1 = -(T_1 - T_3)L_1, \quad M_2 = -(T_2 - T_4)L_2. \tag{1.7}$$

The vectors $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$, directed perpendicular to the motor axes, are defined as:

$$\boldsymbol{m}_1 = \begin{pmatrix} \cos\alpha_1 \\ -\sin\alpha_1 \end{pmatrix}, \quad \boldsymbol{m}_2 = \begin{pmatrix} -\cos\alpha_2 \\ \sin\alpha_2 \end{pmatrix}. \tag{1.8}$$

The control moments $\boldsymbol{M}_{XY}$ in the coordinate plane OXY are defined as:

$$\boldsymbol{M}_{XY} = u_x \boldsymbol{i} + u_y \boldsymbol{j} = \boldsymbol{M}_1 + \boldsymbol{M}_2 = -(T_1 - T_3)L_1 \boldsymbol{m}_1 + (T_2 - T_4)L_2 \boldsymbol{m}_2. \tag{1.9}$$

Substituting the expressions for $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$, the components of the control moments are:

$$u_x = (T_1 - T_3)L_1 \sin\alpha_1 + (T_2 - T_4)L_2 \sin\alpha_2, \tag{1.10}$$
$$u_y = (T_1 - T_3)L_1 \cos\alpha_1 - (T_2 - T_4)L_2 \cos\alpha_2. \tag{1.11}$$

Thus, the analytical expressions for the lift force u_z (1.5) and the control moments: yaw u_ψ (1.6), roll u_x (1.10), and pitch u_y (1.11) are presented. They depend on the thrusts T_1, T_2, T_3, T_4 and the geometry of the motor installation characterized by the angles α_1, α_2, as well as the distances L_1, L_2.

The relation between the control moment vectors

$$\boldsymbol{U} = \begin{pmatrix} u_z \\ u_\psi \\ u_x \\ u_y \end{pmatrix}$$

and the motor thrusts

$$\boldsymbol{T} = \begin{pmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{pmatrix}$$

can be written in matrix form as:

$$\boldsymbol{U} = D_M \boldsymbol{T}, \tag{1.12}$$

where the matrix D_M has the following form:

$$D_M = k \begin{pmatrix} -1 & -1 & -1 & -1 \\ L_2\cos\alpha_2 & -L_1\cos\alpha_1 & L_2\cos\alpha_2 & -L_1\cos\alpha_1 \\ L_2\sin\alpha_2 & L_2\sin\alpha_2 & L_1\sin\alpha_1 & L_1\sin\alpha_1 \end{pmatrix}, \tag{1.13}$$

where the matrix size and elements should be confirmed according to your model. The determinant of this matrix is:

$$\det(D_M) = -8kL_1L_2 \sin(\alpha_1 - \alpha_2). \tag{1.14}$$

Thus, the inverse matrix $K_D = D_M^{-1}$, which relates the motor thrusts to the required control moments, has the form:

$$K_D = \frac{1}{4kL_1L_2\sin(\alpha_1-\alpha_2)}\begin{pmatrix} 1 & 1 & 1 \\ \cos\alpha_1\cdot 2L_2\sin(\alpha_1-\alpha_2) & -\cos\alpha_2\cdot 2L_1\sin(\alpha_1-\alpha_2) & -\cos\alpha_2\cdot 2L_2\sin(\alpha_1 \\ \sin\alpha_1\cdot 2L_2\sin(\alpha_1-\alpha_2) & -\sin\alpha_2\cdot 2L_1\sin(\alpha_1-\alpha_2) & -\sin\alpha_1\cdot 2L_2\sin(\alpha_1 \\ \frac{1}{4k} & -\frac{1}{4k} & \frac{1}{4k} \end{pmatrix}$$

$$(1.15)$$

If we take $\alpha_1 = 45°$, $\alpha_2 = 135°$, and $L_1 = L_2 = L$, then the matrices D_M and K_D become:

$$D_M = k\begin{pmatrix} -1 & -1 & -1 & -1 \\ -2L & 2L & 2L & -2L \\ 1 & -2L & 2L & 1 \\ k & -2L & -2L & 1 \end{pmatrix}, \tag{1.16}$$

$$K_D = \frac{1}{4kL}\begin{pmatrix} 1 & 1 & 1 & 1 \\ -2L & -2L & 2L & 2L \\ -2L & 2L & 2L & -2L \\ \frac{1}{4k} & -\frac{1}{4k} & \frac{1}{4k} & -\frac{1}{4k} \end{pmatrix}. \tag{1.17}$$

These mathematical expressions were used to develop the quadrocopter model in the software environment and are intended for calculating its kinematic characteristics, such as motor thrust values and their influence on the quadrocopter body around the yaw axis.

The dynamic model of the quadrocopter describes the evolution of its state under the influence of internal and external forces, including thrust from the propellers, gravity, aerodynamic drag, and moments caused by thrust asymmetry. Unlike a purely kinematic approach, it accounts for inertial effects and the impact of control inputs on flight dynamics.

The quadrocopter has six degrees of freedom: its motion consists of translational displacement of the center of mass and rotation about the center of mass. To describe orientation in space, a parameterization using quaternions is employed, which avoids singularities and ensures numerically stable integration of rotational motion.

The state of the vehicle is described by the vector [**?**]:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{p}, & \boldsymbol{v}, & \boldsymbol{q}, & \boldsymbol{\omega} \end{bmatrix}, \tag{1.18}$$

where $\boldsymbol{p} \in \mathbb{R}^3$ is the position of the quadrocopter in the world coordinate system, $\boldsymbol{v} \in \mathbb{R}^3$ is the linear velocity, $\boldsymbol{q} \in \mathbb{R}^4$ is a unit quaternion describing orientation in space, and $\boldsymbol{\omega} \in \mathbb{R}^3$ is the angular velocity in the body frame.

Control is applied via the vector of propeller angular velocities:

$$\boldsymbol{u} = \begin{bmatrix} \omega_1 & \omega_2 & \omega_3 & \omega_4 \end{bmatrix}^\top \in \mathbb{R}^4. \tag{1.19}$$

—

1.3.1 Linear dynamics

The total thrust force generated by the propellers in the body coordinate system is expressed as:

$$\boldsymbol{F}_{\text{body}} = \begin{bmatrix} 0 \\ 0 \\ \sum_{i=1}^{4} k_f \omega_i^2 \end{bmatrix}, \tag{1.20}$$

where k_f is the thrust coefficient.

This force is transformed into the world coordinate system using the rotation matrix $R(\boldsymbol{q})$ corresponding to the current quaternion:

$$\boldsymbol{F}_{\text{world}} = R(\boldsymbol{q})\boldsymbol{F}_{\text{body}} - m\boldsymbol{g} - c_d\boldsymbol{v}, \tag{1.21}$$

where m is the mass of the quadrocopter, $\boldsymbol{g} = \begin{bmatrix} 0 \\ 0 \\ 9.81 \end{bmatrix}$ m/s^2 is the gravity vector, and c_d is the aerodynamic drag coefficient proportional to the velocity.

From Newton's second law, the acceleration is obtained as:

$$\boldsymbol{a} = \frac{\boldsymbol{F}_{\text{world}}}{m}. \tag{1.22}$$

By integrating the acceleration, the linear velocity and position are updated as:

$$\boldsymbol{v}(t + \Delta t) = \boldsymbol{v}(t) + \boldsymbol{a}\,\Delta t, \tag{1.23}$$

$$\boldsymbol{p}(t + \Delta t) = \boldsymbol{p}(t) + \boldsymbol{v}(t)\,\Delta t + \frac{1}{2}\boldsymbol{a}\,(\Delta t)^2, \tag{1.24}$$

—

1.3.2   Rotational dynamics

The torque moments generated by the propellers are modeled as:

$$\boldsymbol{\tau} = \begin{bmatrix} lk_f(\omega_2^2 - \omega_4^2) \\ lk_f(\omega_3^2 - \omega_1^2) \\ k_m(\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2) \end{bmatrix}, \tag{1.25}$$

where l is the arm length, k_f is the thrust coefficient, and k_m is the moment coefficient. The angular acceleration is computed from Euler's equations:

$$\dot{\boldsymbol{\omega}} = I^{-1}\left(\boldsymbol{\tau} - \boldsymbol{\omega} \times (I\boldsymbol{\omega})\right), \tag{1.26}$$

where $I \in \mathbb{R}^{3\times 3}$ is the inertia tensor.

The angular velocity is updated as:

$$\boldsymbol{\omega}(t + \Delta t) = \boldsymbol{\omega}(t) + \dot{\boldsymbol{\omega}}\,\Delta t. \tag{1.27}$$

—

The body orientation is represented by the quaternion $\boldsymbol{q}$. From quaternion kinematics:

$$\dot{\boldsymbol{q}} = \frac{1}{2}\boldsymbol{q} \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix}, \tag{1.28}$$

where $\otimes$ denotes quaternion multiplication.

The orientation is updated via numerical integration:

$$\boldsymbol{q}(t + \Delta t) = \boldsymbol{q}(t) + \dot{\boldsymbol{q}} \, \Delta t, \tag{1.29}$$

followed by normalization to unit length, with a small constant 10^{-8} added to avoid division by zero.

—

The translational acceleration of the center of mass is given by Newton's second law:

$$m\boldsymbol{a} = \boldsymbol{F}_{\text{thrust}} - m\boldsymbol{g} - \boldsymbol{F}_{\text{drag}}, \tag{1.30}$$

while the rotational motion is defined by Euler's angular momentum equation:

$$I\dot{\boldsymbol{\omega}} = \boldsymbol{\tau} - \boldsymbol{\omega} \times (I\boldsymbol{\omega}), \tag{1.31}$$

where $\boldsymbol{\tau}$ is the resultant torque caused by unbalanced rotor thrusts and reactive moments, $\boldsymbol{\omega}$ is the angular velocity, and I is the inertia tensor of the drone.

The moments include:

- overturning moments (roll/yaw): generated by the difference in thrust of opposite rotors;

- reaction torque (yaw): arises from the difference in reaction torques of all four rotors, accounted for through the torque coefficient.

Thus, the model covers both translational and rotational motion of the quadrocopter, including the effects of gravity, aerodynamic drag, propeller constraints, and nonlinear moments. The choice of a quaternionic representation provides a numerically stable description of the orientation without features inherent in, for example, Euler angles.

It should be noted that hybrid stochastic models are now widely used for modeling and estimation of motion parameters of various objects.

Hybrid systems are mathematical models of dynamics in which continuous behavior described by one of the predetermined sets of continuous dynamic equations alternates with discrete events. These events can cause an instantaneous switch between different modes of dynamics, an instantaneous change in the system state (coordinates), or both at the same time [17,18].

The scientific literature presents many approaches to the construction of hybrid models. Thus, [17] describes a hybrid model of vehicle motion in which the acceleration and braking phases are modeled using differential equations, while lane change is treated as a discrete event. In [18], an original neuro-stochastic hybrid model combining artificial neural network and stochastic process methods is proposed. In [18], we consider an algorithm for route construction in a navigation problem using piecewise linear approximation of the trajectory when moving through several geophysical fields with restrictions on the length of linear sections.

In [19], a hybrid stochastic model is described as a multimode system consisting of a set of discrete linear stochastic models, each of which describes a separate mode of motion of an object - a trajectory segment approximated by a linear model. This approach allows a complex, generally nonlinear trajectory to be approximated by a sequence of simple linear segments.

Hybrid models are widely used to describe and control the motion of quadrocopters, where they serve as an effective tool for implementing adaptive and accurate behavior in different environmental conditions. Combining physical models and machine learning

methods allows achieving high accuracy in control and navigation tasks. The development of such models opens prospects for improving the efficiency of drones and expanding their practical applications.

In particular, the control of a drone while performing a maneuver such as a flip can be described as a hybrid system. The flight is divided into different modes - takeoff, rotation, stabilization, landing - each of which has its own continuous dynamics described by a system of differential equations. Transitions between modes occur discretely - by events, signals, time conditions, or reaching a certain state.

In this paper, a hybrid approach is used, combining kinematic and dynamic models within a stochastic process. This approach allows taking into account the limitations on the computational resources of the system, as well as combining different computational methods to achieve an acceptable quality of flight control. The use of the Kalman filter for state estimation (position, linear and angular velocity, orientation) allows to ensure stable and adaptive operation of the model even under conditions of uncertainty and disturbances.

At the same time, the modeled system has the characteristics of a hybrid structure:

- discrete events, such as switching control strategies;

- different dynamics models for individual phases of flight;

- discrete logic for switching between these phases, based on time, reaching set angles, or other criteria.

1.4 Algorithmic Foundations of UAV Motion Control

Modern UAVs solve problems requiring high accuracy and reliability of flight task fulfillment, which is achieved by using efficient algorithms for real-time trajectory planning, prediction and correction. Within the framework of this work, the following main tasks related to UAV trajectory motion are solved:

- construction of a mathematical model of motion dynamics taking into account physical parameters of the quadrocopter, control constraints and aerodynamic effects;

- formation of a three-dimensional representation of the surrounding space based on inertial navigation system (IMU) data processed using the Kalman filter to estimate position, velocity, orientation and angular velocity;

- search for the optimal route for the UAV trajectory.

Let us consider in more detail modern approaches to solving each of the above tasks.

Building a mathematical model of motion dynamics When planning a path, it is important to consider the kinematic and dynamic constraints of the quadcopter, including limitations on speed, acceleration, momentum, and the ability to avoid obstacles. Such constraints are critical for realizing safe and realistic trajectories and are discussed in detail in [21-25].

A dynamic model of the quadrocopter is built based on the equations of motion of a solid body considering thrust, aerodynamic forces and moments, controllability constraints, and actuator parameters. This model serves as a basis for predicting the system behavior within the control strategy with prediction allows to take into account the

physical limitations of the UAV, trajectory deviations and feedback from on-board measurements when planning control actions. Various approximation methods, such as those used in [23,26], are used to obtain a smooth and feasible trajectory over given control points.

The dynamic model is necessary to implement optimal control algorithms that minimize the target functional while respecting the dynamics of the system. Also, the model is necessary for flight simulation and testing algorithms in a virtual environment, serves as a basis for state estimation filters and is used for back calculation of control when performing a given trajectory. Thus, the dynamic model is the foundation for a wide range of UAV planning, control and localization algorithms.

The formation of a three-dimensional representation of the surrounding space is one of the key tasks of autonomous navigation, which is necessary for the realization of feedback control - obstacle avoidance, trajectory construction and solution of applied tasks. In modern UAVs, mapping tasks are often solved simultaneously with the localization task, which allows reducing the dependence on GPS signal and increasing the stability of the system in its absence or interference [2, 27-29]. This approach is implemented in the framework of SLAM (Simultaneous Localization and Mapping) algorithms based on simultaneous map construction and UAV position determination. Depending on the sensors used, visual-inertial and lidar-inertial implementations are distinguished. Lidar-inertial methods have an important advantage of high robustness to illumination changes, which is especially valuable when working in complex environments where visual methods lose accuracy.

One of the advanced SLAM algorithms is LIO-SAM [30-31], developed as a development of the popular LOAM [32]. The main improvement of LIO-SAM is the use of a factor graph for odometry optimization. This allows flexibility to incorporate both relative and absolute measurements, including contour closures from different sources.

Compared to lidars, cameras are cheaper, consume less power, and provide higher frame rates (typically 30-120 Hz or higher), making them particularly attractive for use on small UAVs. A detailed review of visual-inertial odometry (VIO) techniques is presented in [33]. Visual sensors provide rich texture information, which enables the use of computer vision algorithms. However, lidars still outperform cameras in terms of accuracy in measuring distances and object geometry. One of the main limitations of VIO methods is the decrease in accuracy at high flight speeds due to error accumulation. To reduce this effect, various filters such as MSCKF and ROVIO are used, as well as the VINS-Mono algorithm, which implements localization based on data from a single camera and inertial module.

In addition to VIO, there are also visual odometry methods that do not use inertial data. Algorithms such as ORB-SLAM and OpenVSLAM build a world map based on images only, applying graph optimization and loop closure mechanisms to correct accumulated errors. These approaches are particularly effective when the motion is slow and there are no sudden accelerations, such as in mapping tasks with ground robots. A number of studies [31-33] have compared the RTAB-Map, ORB-SLAM, LSD-SLAM, and OpenVSLAM algorithms. ORB-SLAM3 showed the highest localization accuracy, whereas OpenVSLAM demonstrated excellent contour closure and map updating ability. Pattern recognition systems play an additional role in navigation, especially in uncertain environments such as indoor or urban environments. Such systems analyze visual information to identify objects, determine their position and speed, and avoid collisions. One of the common techniques is the contrast method, which highlights characteristic features

of images - object boundaries, sharp brightness differences and other significant features. This allows to perceive and interpret the surrounding space more accurately, improving the construction of trajectories and stability of navigation. Various classes of algorithms are used to solve the problem of finding the optimal path in three-dimensional space:

Graph-based pathfinding algorithms Classical pathfinding algorithms include Dijkstra, Floyd [34-36] and their various optimizations. The A* heuristic is also widely used. These algorithms efficiently solve pathfinding problems in discrete spaces with a known map. Other heuristic algorithms such as Particle Swarm Optimization, Artificial Bee Colony and Harmony Search are optimization oriented and are used for planning smooth trajectories and tuning drone control parameters.

Random Search Algorithms In [3], the RRT algorithm was successfully applied to plan the high-speed movement of a drone in a dense forest environment. In addition to RRT, Probabilistic Roadmap and Rapidly-exploring Random Graph methods are also widely used for navigation and path planning tasks, which efficiently explore complex state spaces and provide robust pathfinding in high-dimensional and constrained environments.

Visual Perception Methods In situations where 3D point cloud data is not available, optical flow analysis algorithms are often used. One of the simplest and most common methods is the Horn-Schunk method, but requires significant computational resources. The Lucas-Canade method operates locally on key points and is effective for small displacements, and its pyramidal version extends the range of motion.

In modern scientific research on the topic of optimal control, methods of the family of linear quadratic regulators (LQR) are increasingly used. Special attention is paid to the iterative variant, iLQR, which effectively copes with control problems of nonlinear systems. In [37,38], a hybrid iLQR-MPC approach combining the advantages of an iterative linear quadratic regulator and model-predictive control is proposed and investigated. The papers show that such a combined controller outperforms both a stand-alone iLQR and a classical PID controller, both in terms of trajectory following accuracy and transient stability, especially under fast maneuvers and controllability constraints.

In [38], a constrained iterative LQR, CILQR, an extension of iLQR subject to constraints and uncertainties, is presented to improve the safety and adaptability of the system. The authors proposed a two-stage prediction strategy and a scenario-dependent cost function, which achieved stable behavior without the need to manually adjust parameters for each scenario.

In addition, [39] noted the use of a linear version of the LQR algorithm to control the speed of permanent magnet synchronous motors instead of a traditional PID controller. This approach allowed to significantly reduce speed and torque fluctuations, which had a positive effect on the overall performance of the control system.

The cost function plays a central role in optimal control methods. It not only estimates the errors in various system parameters such as position, orientation, and control actions, but also guides the system toward the final goal by minimizing these errors. Effective use of the cost function ensures that the system moves in the right direction and reaches the goal with minimal deviations. The major components of the cost function typically include:

- position errors - deviations of the current position from the target position;

- orientation errors - deviations of the current orientation from the target orientation;

- control errors - deviations between the current control forces and those required to follow the trajectory.

In addition, the cost function may include a terminal term that plays an important role in ensuring the convergence of the system to the final goal. This ensures that, regardless of the path the system takes, it will tend to a given end state.

Optimal Control Methods A class of algorithms aimed at constructing control actions that minimize a given objective function, taking into account system dynamics and state and control constraints. Unlike discrete graph methods and reinforcement learning approaches, optimal control operates in continuous spaces and allows one to take into account the physical characteristics of the system, such as mass, moment of inertia, velocity and acceleration limits.

In modern scientific research on the topic of optimal control, methods of the family of linear quadratic regulators (LQR) are increasingly used. Special attention is paid to the iterative variant, iLQR, which effectively copes with control problems of nonlinear systems. In [37,38], a hybrid iLQR-MPC approach combining the advantages of an iterative linear quadratic regulator and model-predictive control is proposed and investigated. The papers show that such a combined controller outperforms both a stand-alone iLQR and a classical PID controller, both in terms of trajectory following accuracy and transient stability, especially under fast maneuvers and controllability constraints.

In [38], a constrained iterative LQR, CILQR, an extension of iLQR subject to constraints and uncertainties, is presented to improve the safety and adaptability of the system. The authors proposed a two-stage prediction strategy and a scenario-dependent cost function, which achieved stable behavior without the need to manually adjust parameters for each scenario.

In addition, [39] noted the use of a linear version of the LQR algorithm to control the speed of permanent magnet synchronous motors instead of a traditional PID controller. This approach allowed to significantly reduce speed and torque fluctuations, which had a positive effect on the overall performance of the control system.

The cost function plays a central role in optimal control methods. It not only estimates the errors in various system parameters such as position, orientation, and control actions, but also guides the system toward the final goal by minimizing these errors. Effective use of the cost function ensures that the system moves in the right direction and reaches the goal with minimal deviations.

The major components of the cost function typically include:

- position errors - deviations of the current position from the target position;

- orientation errors - deviations of the current orientation from the target orientation;

- control errors - deviations between the current control forces and those required to follow the trajectory.

In addition, the cost function may include a terminal term that plays an important role in ensuring the convergence of the system to the final goal. This ensures that, regardless of the path the system takes, it will tend to a given end state.

The cost function for the trajectory planning problem used in this work is defined as follows:

$$X_k = (x_k - x_{\text{target},k})^\top Q (x_k - x_{\text{target},k}), \tag{1.32}$$

$$U_k = (u_k - u_{\text{target},k})^\top R (u_k - u_{\text{target},k}), \tag{1.33}$$

$$J = \sum_{k=0}^{N-1} (X_k + U_k) + (x_N - x_{\text{target},N})^\top Q_f (x_N - x_{\text{target},N}). \tag{29}$$

where:

- x_k and u_k are the state and control input vectors at time step k,

- $x_{\text{target},k}$ and $u_{\text{target},k}$ are the target state and control input vectors at step k,

- Q and R are weighting matrices for the state and control input, which determine the importance of minimizing state and control deviations,

- Q_f is the terminal weighting matrix that defines the cost of the final state deviation at step N.

Smooth curve approximation by trajectory control points is an important step in UAV motion planning. In [41], a numerical gradient optimization method was proposed for the classical B-spline approach to shift the trajectory to a safe distance from obstacles. This optimization ensures not only smoothing but also safety of the planned path. A comparative analysis of various trajectory approximation methods is also presented, including:

- B-splines;

- radial basis functions (RBFs);

- quintic polynomials;

- approximation based on Chebyshev polynomials;

- collocation methods;

- parametric models with constraints (Bezier and the like);

- parametric curves with smooth curvature (clotoids, cubic spirals, G^2-splines, intrinsic-splines);

Each of these approaches has its own advantages in terms of smoothness, computational complexity, ease of constraint consideration, and robustness to noise. However, once the target trajectory is constructed, the next important task is to ensure its reliable reproduction under the conditions of real drone dynamics and external disturbances. At this stage, the key is not only to follow the predefined trajectory, but also to effectively control the behavior of the system to ensure stability and accuracy in real conditions. An important part of solving this problem is the use of a cost function that allows us to evaluate the quality of task performance and manage deviations from the target trajectory.

It should be noted that in the iLQR method, the approximation takes place at the level of local linear and quadratic approximations of the dynamics and cost function, which allows the integration of planning and control to take into account the real physical constraints of the system directly in the optimization process. While classical trajectory approximation methods such as splines and polynomials are mainly focused on geometric smoothing and can account for constraints through additional terms, iLQR works with dynamics and control criteria to provide more accurate modeling of velocity, acceleration, and deviations. This allows iLQR to converge to a locally optimal solution faster and adapt to changing conditions and perturbations in real time.

1.5 Development Tools for Control Systems

UAV control systems are typically implemented in C and C++ because they provide direct access to hardware registers, low response times, and resource control, which are critical for real-time tasks. These languages are widely used in low-level software, including HAL layers, RTOS, sensor drivers, and stabilization loop implementations. The Python language is predominantly used in high-level components - for example, for developing and testing scheduling algorithms, visual navigation, interacting with simulators, and analyzing logs, due to its wide ecosystem environment (NumPy, SciPy, ROS 2, OpenCV, etc.). MATLAB/Simulink is used in engineering practice for dynamics modeling, stability analysis and automatic generation of C code for implementation in microcontrollers.

One of the leading platforms for developing robotic systems is ROS 2, a modular and scalable system focused on distributed architecture. The use of the DDS protocol provides reliable and fast message passing, which is critical for real-time control. ROS 2 supports C++ and Python programming languages and provides tools for debugging and visualization. Moreover, ROS 2 is tightly integrated with the PX4 autopilot [42,43].

When choosing Python as a language for the implementation of algorithms for trajectory planning and dynamics control of UAVs, libraries with support for automatic differentiation, critical for the accurate calculation of derivatives, are used. Popular solutions include JAX, Pytorch, and Pydrake. JAX is higher performance due to its JIT compilation using XLA and efficient GPU/TPU operation, providing accurate derivatives and flexibility in differentiating complex functions. Pytorch also supports automatic differentiation, but is inferior to JAX in optimizing for gas pedals.

In Pydrake, you cannot directly pass an arbitrary function as the dynamics of the system. Instead, the complete system structure must be built through classes inherited from LeafSystem, defining inputs, outputs, continuous states. This approach requires more effort and does not allow for rapid changes in the shape of the dynamics. In addition, automatic differentiation in Pydrake is done via AutoDiffXd types, which requires explicit handling of system contexts and objects containing derivatives. This makes computing gradients and jacobians cumbersome compared to JAX, where you can simply apply jax.jacobian() or jax.grad() to any function written in Python.

Chapter 2

Practical Part

2.1 Problem Statement

The objective is to develop a trajectory control system for a rotorcraft-type UAV capable of effectively managing the motion parameters of a quadrotor during a flip maneuver. This includes stabilization of the vehicle's orientation and position under external disturbances and physical constraints. The control system should be adaptive, taking into account varying flight conditions and the dynamic characteristics of the quadrotor, such as rotor speed variation, inertial effects, and other factors influencing maneuver execution.

In this study, a quadrotor with a symmetric X-shaped frame was used. The main parameters of the drone are listed below:

- Mass of the UAV: $0.82\,\text{kg}$

- Motor time constant: $0.02\,\text{s}$

- Rotating parts moment of inertia: $6.56 \times 10^{-6}\,\text{kg}\,\text{m}^2$

- Thrust coefficient: $1.48 \times 10^{-6}\,\text{N}/(\text{rad/s})^2$

- Torque coefficient: $9.4 \times 10^{-8}\,\text{Nm}/(\text{rad/s})^2$

- Aerodynamic drag coefficient: $0.1\,\text{N}/(\text{m/s})$

- Aerodynamic moment coefficients (for all axes X, Y, Z): $0.003\,\text{Nm}/(\text{rad/s})^2$

- Moments of inertia (for all axes X, Y, Z): $0.045\,\text{kg}\,\text{m}^2$

- Maximum angular velocity of propeller rotation: $2100\,\text{rad}\,\text{s}^{-1}$

- Moment arm (distance from center of mass to thrust application point): $0.15\,\text{m}$

- Process noise used in force and moment filtering: $1.25 \times 10^{-7}\,\text{N}\,\text{m}^2\,\text{s}$ and $0.0005\,\text{N}^2\,\text{s}$, respectively

- Sensor noise and initial covariance parameters:

    - Initial bias variance: 1×10^{-5}
    - Accelerometer white noise: $1 \times 10^{-4}\,\text{m}^2/\text{s}^4$
    - Gyroscope white noise: $1 \times 10^{-5}\,\text{rad}^2/\text{s}^2$

2.2 Acquisition and Processing of Quadcopter Sensor Data

To obtain sensor data, the following telemetry messages from PX4-Autopilot were used:

- `VehicleAttitude` — drone orientation as a quaternion and angular velocities;

- `VehicleImu` — accelerometer and gyroscope data in the drone's body frame;

- `ActuatorOutputs` — normalized motor control signals in the range $[-1; 1]$;

- `VehicleLocalPosition` — position, velocity, and acceleration estimates in the NED frame; used for testing and validation during simulation;

- `VehicleAngularVelocity` — angular velocities;

- `VehicleAngularAccelerationSetpoint` — angular acceleration setpoints;

- `VehicleMagnetometer` — magnetic field vector, used in the EKF to correct yaw;

- `SensorBaro` — barometric pressure for correction of the vertical position component;

- `EscStatus` — motor rotation speeds (RPM).

An inertial navigation system was implemented based on IMU, barometer, and magnetometer data. The primary task of the module is to estimate the drone's current velocity and position in the world coordinate system. Navigation is performed through step-by-step integration of the velocity change measured by the accelerometers, taking into account the current orientation of the vehicle.

To transform local accelerations into the global frame, a quaternion representing the drone's orientation is used. The transformation is performed using the `scipy.spatial.transform.Rota` library. After transforming accelerations into the world frame, the gravitational component is added, and velocity and position are integrated.

Python code for IMU integration:

Listing 2.1: Inertial integration based on PX4 IMU data

```python
def vehicle_imu_callback(self, msg: VehicleImu):
    delta_velocity = np.array(msg.delta_velocity, dtype=np.float32)
    delta_velocity_dt = msg.delta_velocity_dt * 1e-6  # convert from  s  to

    if delta_velocity_dt > 0.0:
        # Convert orientation quaternion to rotation matrix
        rotation = Rot.from_quat(self.vehicleAttitude_q)
        # Transform acceleration to world coordinates
        delta_velocity_world = rotation.apply(delta_velocity)
        gravity = np.array([0.0, 0.0, -9.80665], dtype=np.float32)
        # Add gravity
        delta_velocity_world += gravity * delta_velocity_dt
        # Update linear velocity
        self.vehicleImu_velocity_w += delta_velocity_world
        # Update position
        self.position += self.vehicleImu_velocity_w * delta_velocity_dt
```

Altitude estimation from barometric pressure:

Listing 2.2: Altitude calculation using barometer

```
SEA_LEVEL_PRESSURE = 101325.0
self.baro_altitude = 44330.0 * (1.0 - (msg.pressure / SEA_LEVEL_PRESSURE) **
```

The implemented approach enables estimating the drone's position in space in the absence of external navigation systems. However, over time this leads to error accumulation. For short-term maneuvers, such as a flip, such drift can be acceptable. Nevertheless, to improve the robustness and accuracy of state estimation even during brief, dynamically intense flight segments, an *Extended Kalman Filter (EKF)* is applied.

The EKF aligns predictions from the dynamic model with sensor measurements, compensating for sensor noise and inaccuracies. It is especially useful in conditions of non-linear drone dynamics, helping stabilize the navigation estimate during rapid changes in orientation and acceleration.

The state prediction (position, velocity, orientation) is based on the physical equations of motion (see Appendix A) and sensor inputs. Filtering is implemented using the `filterpy.kalman.ExtendedKalmanFilter` library.

Noise covariance matrices used:

Listing 2.3: EKF process and measurement noise covariance

```
self.ekf.Q = np.diag([
    0.01, 0.01, 0.001,                  # x, y, z
    0.001, 0.001, 0.001,                # vx, vy, vz
    0.01, 0.01, 0.01, 0.01,             # qw, qx, qy, qz
    0.01, 0.01, 0.01                    # wx, wy, wz
])

self.ekf.R = np.diag([
    0.05, 0.05, 0.005,                  # position x, y, z
    0.005, 0.005, 0.005,                # velocity vx, vy, vz
    0.05, 0.05, 0.05, 0.05,             # orientation qw, qx, qy, qz
    0.05, 0.05, 0.05,                   # angular velocity wx, wy, wz
    0.005                               # barometric altitude
])
```

The values in the process noise covariance matrix Q were selected considering the properties of the inertial navigation system operating without external signals. For linear parameters, values in the range of $10^{-3} - 10^{-2}$ were set, reflecting the integration error accumulation. Higher values (on the order of 10^{-2}) were used for orientation (quaternions) and angular rates due to gyroscopic integration drift.

In the measurement noise covariance matrix R, increased values were assigned to the gyroscope-related components to account for IMU noise and drift. In contrast, lower noise values were used for barometric altitude and linear velocities, as these are generally more accurate and stable.

2.3 Development of a Dynamic Model of the Quadcopter

The drone model takes into account the following key physical parameters:

- Mass: $0.82\,\mathrm{kg}$

- Moments of inertia (for all axes): $0.045\,\mathrm{kg\,m^2}$

- Arm length (distance from center of mass to motor): $0.15\,\mathrm{m}$

- Thrust and torque coefficients: $1.48 \times 10^{-6}\,\mathrm{N/(rad/s)^2}$ and $9.4 \times 10^{-8}\,\mathrm{Nm/(rad/s)^2}$, respectively

- Maximum rotor angular velocity: $2100\,\mathrm{rad\,s^{-1}}$, approximately $20\,054$

- Aerodynamic drag is considered, as well as a maximum body angular rate limit: $25\,\mathrm{rad\,s^{-1}}$

Additional model features include:

- Gravitational acceleration is included;

- Quaternion-based orientation with transformation between the body and world frames;

- Thrust and control moments from each motor are calculated based on the square of rotor speeds;

- Quaternion normalization at each integration step to prevent numerical drift and avoid gimbal lock during flip maneuvers;

- Gyroscopic moments caused by interaction between angular velocity and rotor inertia are accounted for, which is especially important during aggressive maneuvers like flips;

- Numerical integration using the Euler method is applied for both translational and rotational dynamics;

- Forces acting on the UAV in the world frame are considered, including total thrust, aerodynamic drag, and gravitational weight.

To simulate quadrotor dynamics, a function $f(x, u, \Delta t)$ was implemented (see Appendix A), which computes the next UAV state over integration step Δt, based on the current state x and control input u. The state vector components are updated according to equations (1)–(28).

Quaternion normalization is applied to prevent gimbal lock during flips. A rotation matrix from body to world coordinates is formed as follows:

Listing 2.4: Conversion from quaternion to rotation matrix

```
R_bw = jnp.array(R.from_quat(quat).as_matrix())
```

Control Signal Processing and State Update

The control input u contains motor RPMs. For physical modeling, they are clipped to a maximum value and squared to obtain thrust:

<center>Listing 2.5: RPM to thrust conversion</center>

```
rpm = jnp.clip(u, 0.0, max_speed)
w_squared = rpm ** 2
thrusts = kf * w_squared  # Thrust produced by each motor
```

The thrust from each motor is summed to compute the total force in the drone's body frame:

$$F_z^{\text{body}} = \begin{bmatrix} 0 \\ 0 \\ \sum_{i=1}^{4} T_i \end{bmatrix}$$

This force is transformed into the world frame and combined with gravity and air drag:

```
F_world = R_bw @ Fz_body - m * g - drag * vel
acc = F_world / m
```

Using the resulting acceleration, velocity and position are integrated using the Euler method:

```
new_vel = vel + acc * dt
new_pos = pos + vel * dt + 0.5 * acc * dt ** 2
```

Torque Calculation

Control torques $\boldsymbol{\tau}$ are derived from the difference in thrusts, assuming a symmetric X-configuration:

$$\tau_{\text{roll}} = l \cdot (T_2 - T_4)$$
$$\tau_{\text{pitch}} = l \cdot (T_3 - T_1)$$
$$\tau_{\text{yaw}} = k_m \cdot (\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2)$$

```
tau = jnp.array([
    arm * (thrusts[1] - thrusts[3]),  # Roll
    arm * (thrusts[2] - thrusts[0]),  # Pitch
    km * (w_squared[0] - w_squared[1] + w_squared[2] - w_squared[3])
# Yaw
])
```

Gyroscopic (Coriolis) effects are added:

```
omega_cross = jnp.cross(omega, I @ omega)
```

Angular acceleration is computed via:

$$\dot{\omega} = I^{-1}(\tau - \omega \times (I\omega))$$

```
omega_dot = jnp.linalg.solve(I, tau - omega_cross)
new_omega = omega + omega_dot * dt
```

Quaternion Update

The quaternion derivative is computed based on angular velocity:

```
omega_quat = jnp.concatenate([jnp.array([0.0]), omega])
dq = 0.5 * self.quat_multiply(quat, omega_quat)
new_quat = quat + dq * dt
new_quat /= jnp.linalg.norm(new_quat + 1e-8)
```

Final State Vector

The updated state vector includes:

- Position and linear velocity in the world frame,

- Quaternion orientation,

- Angular velocity in the body frame.

```
x_next = jnp.concatenate([new_pos, new_vel, new_quat, new_omega])
```

Demonstration of drone parameter estimation

During testing of the navigation system, logging of the drone's state while climbing to a height of 5 meters was performed. The simulation environment provides the reference coordinates of the drone, which are used for comparison with the subsystem estimates. Figure 2.1, the state estimation successfully smooths the actual trajectory, demonstrating the accurate and stable behavior of the algorithm. Figure 2.2 shows that the high error estimate on X in the process peaked at about 0.5 meters and then decreased to about 0.1 meters. There was a sharp spike in error on Y, which was also successfully smoothed out. The most stable and accurate estimate is provided by Z due to the use of barometer data. Overall, the state estimation effectively smooths the actual trajectory, demonstrating the accuracy and stability of the algorithm.
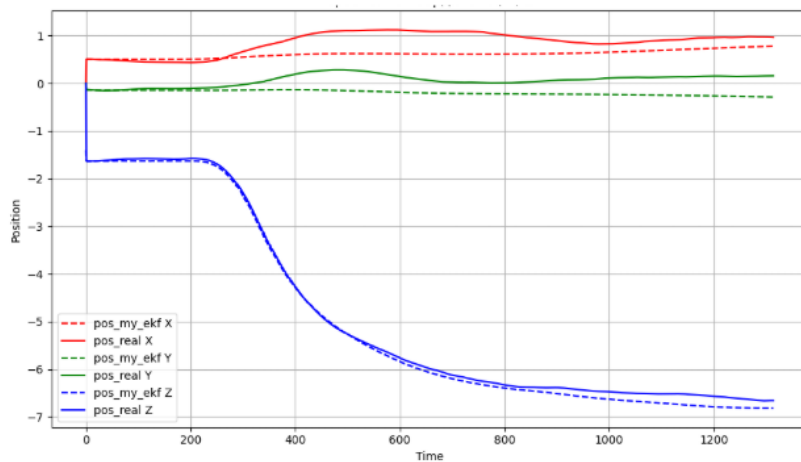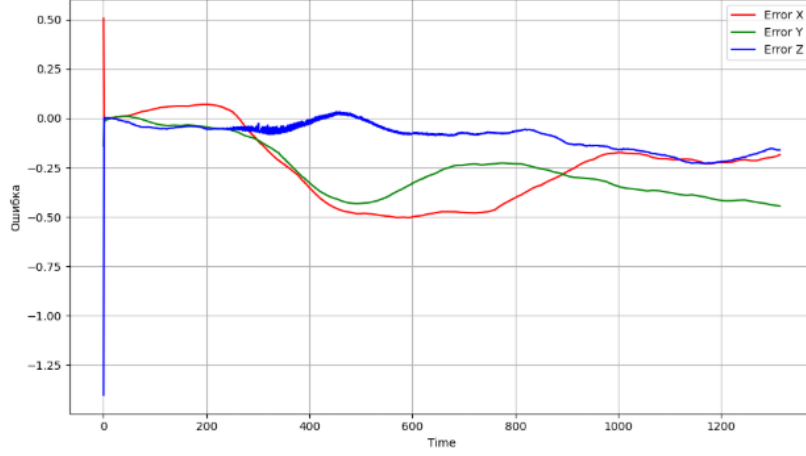

Figure 2.1: Measurements X,Y,Z

Figure 2.2: X,Y,Z coordinate errors

Figure 2.3 show that during a smooth ascent to the point (0.0, 0.0, 5.0) and subsequent hovering, the orientation parameters exhibit slight oscillations occurring during the stabilization process. These oscillations gradually fade after the drone's controllers (PID loops) are actuated, ensuring stable behavior in the target position.
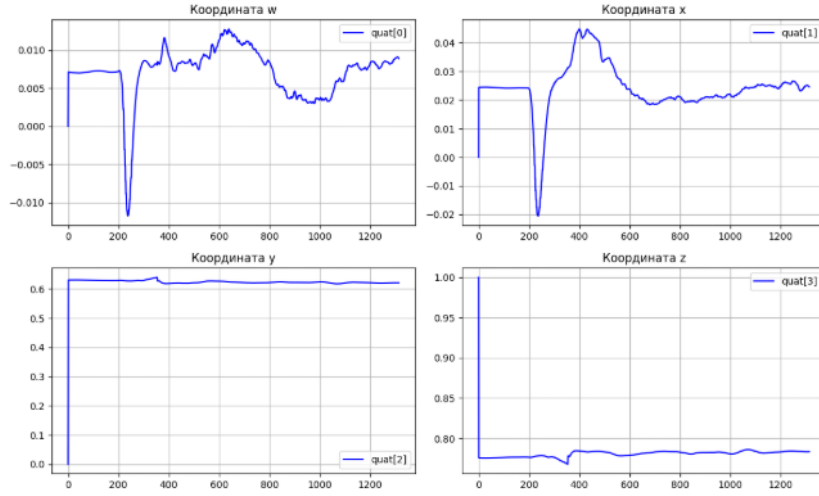

Figure 2.3: Orientation changes in quaternionic form

Figure 2.4 shows the dynamics of velocity along Z. It can be seen that during ascent there is an increase in vertical velocity, which gradually decreases as we approach the target altitude. This behavior corresponds to the phase of active altitude gain followed by the transition to stabilized hovering.

Figure 2.5 shows the behavior of angular velocity at the moment of takeoff. At the beginning, fluctuations are observed due to the active work of the regulators in the ascent phase. After the drone stabilizes and reaches a stationary position, the angular velocity decreases and remains almost constant, which corresponds to the immobility of the vehicle in all axes.

Thus, a realistic model suitable for use in simulations and controllers including MPC control and EKF state estimation is realized.
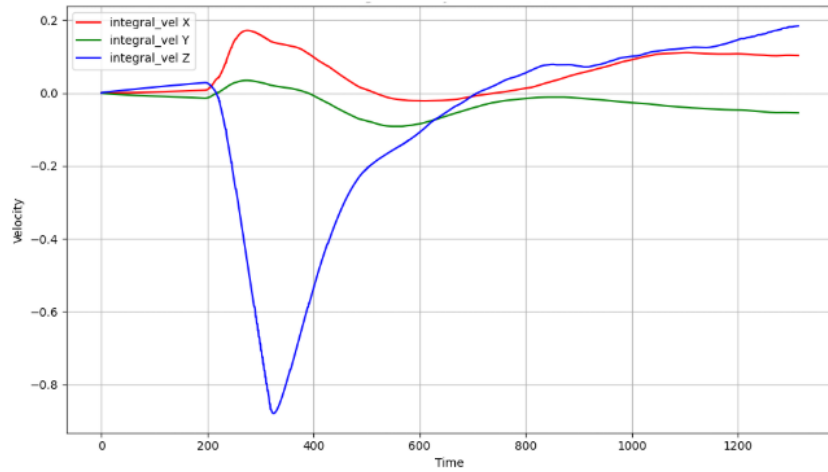
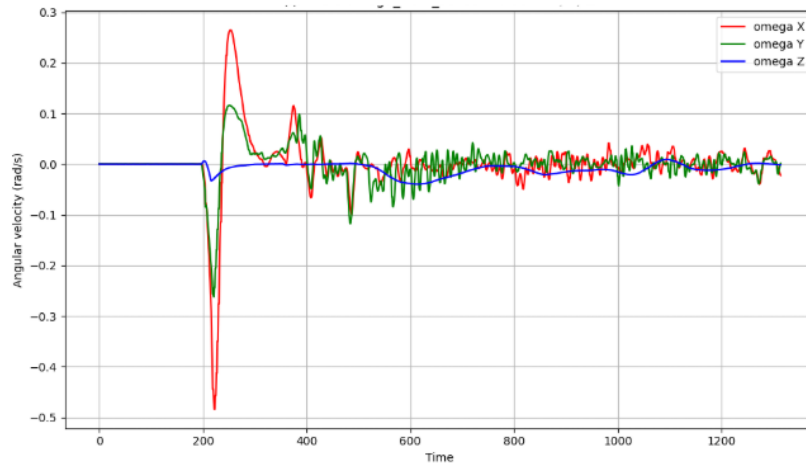Figure 2.4: velocity change along x,y,z axes

Figure 2.5: changes in angular velocity along the x,y,z axes

2.4 Controlled Execution of a Quadcopter Flip

An optimal control system was implemented using the iterative Linear Quadratic Regulator (iLQR) method with automatic differentiation provided by the `JAX` library.

Trajectory Simulation

The function `simulate_trajectory` computes the full system trajectory for a batch of control inputs using JAX's `vmap`, enabling parallel computation of the next state for each control input:

Listing 2.6: Trajectory simulation via JAX vmap

```
@jit
def simulate_trajectory(x0, U):
    X = [x0]
    f_batch = vmap(f, in_axes=(None, 0, None))
    U = jnp.array(U)
    X_new = f_batch(x0, U, dt)
    X.extend(X_new)
```

```
        return jnp.stack(X)
```

Dynamics Linearization

Linearization of the system dynamics is performed by computing the Jacobians with respect to the state and control:

Listing 2.7: Linearization via Jacobians

```
@jit
def linearize_dynamics(x, u):
    A = jacobian(f, argnums=0)(x, u, dt)
    B = jacobian(f, argnums=1)(x, u, dt)
    return A, B
```

Cost Function Quadratization

To quadratize the cost, gradients and Hessians are calculated with respect to both state and control variables:

Listing 2.8: Quadratization of the cost function

```
@jit
def quadratize_cost(x, u, x_target, u_target, Q, R):
    def scalar_cost(x_, u_):
        dx = x_ - x_target
        du = u_ - u_target
        return dx @ Q @ dx + du @ R @ du

    lx = grad(scalar_cost, argnums=0)(x, u)
    lu = grad(scalar_cost, argnums=1)(x, u)
    lxx = hessian(scalar_cost, argnums=0)(x, u)
    luu = hessian(scalar_cost, argnums=1)(x, u)
    lux = jacobian(grad(scalar_cost, argnums=1), argnums=0)(x, u)
    return lx, lu, lxx, luu, lux
```

Backward Pass with Riccati Recursion

The backward pass solves the discrete Riccati equations with regularization of Q_{uu} to ensure invertibility and numerical stability:

Listing 2.9: Backward pass (Riccati recursion)

```
@jit
def backward_pass(X, U, x_target_traj, u_target_traj, Q, R, Qf):
    ...
    Quu_reg = Quu + 1e-6 * jnp.eye(m)
    Quu_inv = jnp.linalg.inv(Quu_reg)
    K = -Quu_inv @ Qux
    kff = -Quu_inv @ Qu
    ...
    return K_list, k_list
```

Forward Pass with Control Update

In the forward pass, the state and control trajectories are updated based on the feedback and feedforward terms computed in the backward pass. A step size parameter α is used to scale the control correction:

Listing 2.10: Forward pass with trajectory update

```
@jit
def forward_pass(X, U, k_list, K_list, alpha):
    ...
    for k in range(horizon):
        dx = x - X[k]
        du = k_list[k] + K_list[k] @ dx
        u_new = U[k] + alpha * du
        U_new.append(u_new)
        x = f(x, u_new, dt)
        ...
```

Cost Function

The cost function (see Appendix B) implements quadratic optimization using weight matrices Q, R, and Q_f. The weights were carefully tuned to reflect the drone's physical characteristics and the control requirements for fast maneuvers.

State and Control Weight Matrices

The state weight matrix Q defines the relative importance of individual state components during the flip maneuver. The chosen matrix is:

$$
Q = \mathrm{diag}\left(\begin{bmatrix} 1.0 & 1.0 & 10.0, & \\ 1.0 & 1.0 & 1.0, & \\ 0.0 & 50.0 & 50.0 & 0.0, \\ 5.0 & 5.0 & 1.0 & \end{bmatrix}\right)
$$

The matrix encodes the following design considerations:

- **Z-position (10.0)** is weighted heavily to ensure the drone maintains altitude during the flip. This helps reduce vertical drop, particularly when thrust is partially misaligned during inversion.

- **X and Y positions (1.0)** have lower importance, as moderate horizontal displacements are acceptable during the maneuver. Overconstraining lateral motion may degrade control robustness, especially given that horizontal position is estimated solely via inertial sensors prone to drift.

- **Orientation weights (50.0 for q_x, q_y)** emphasize roll and pitch alignment, which are critical for flip symmetry. Components q_w and q_z are not penalized directly, as they are indirectly constrained through quaternion normalization.

- **Angular velocities (5.0 for ω_x, ω_y)** are given significant weight to maintain consistent rotational behavior and prevent excessive spinning. The weight for ω_z is lower (1.0), since yaw dynamics are less critical during a roll-based flip.

The control effort is penalized by the input weight matrix R, which applies a small cost uniformly to all motors:

$$R = \text{diag}\left(\begin{bmatrix} 0.001 & 0.001 & 0.001 & 0.001 \end{bmatrix}\right)$$

This setting allows for aggressive maneuvers without overly restricting actuator output, while still encouraging energy-efficient behavior.

Terminal Cost and Flip Maneuver Planning

Terminal Weight Matrix

Small values in the control cost matrix R are used deliberately to avoid limiting actuator power. This enables the system to fully exploit available control authority, which is particularly important for short, high-intensity maneuvers like flips.

The terminal cost matrix Q_f enforces stricter requirements on the final state of the drone:

$$Q_f = \text{diag}\left(\begin{bmatrix} 1.0 & 1.0 & 10.0, \\ 0.1 & 0.1 & 0.1, \\ 0.0 & 100.0 & 100.0 & 0.0, \\ 10.0 & 10.0 & 1.0 \end{bmatrix}\right)$$

- The **Z-position (10.0)** remains heavily weighted to ensure the drone returns precisely to its desired altitude at the end of the maneuver.

- The orientation components q_x and q_y are even more emphasized (**100.0**), enforcing accurate alignment and recovery from inversion.

- **Angular velocities** along the X and Y axes are also weighted more heavily (**10.0**), ensuring that the drone stops residual rotation after completing the flip.

iLQR Implementation and Flip Execution

A complete iLQR optimization routine was implemented with automatic differentiation provided by JAX. This enabled efficient computation of the system's nonlinear dynamics derivatives and solution of the optimal control problem during a flip maneuver.

The **iLQR planner** is responsible for generating a trajectory of control inputs that execute the flip. It is structured as a finite state machine (FSM) with four main phases:

1. **Take-off**: the drone ascends to a safe altitude.

2. **Flip**: a rotational maneuver is executed about the desired axis.

3. **Recovery**: the drone stabilizes and returns to upright orientation.

4. **Landing**: descent is performed under controlled conditions.

Transitions between states are triggered by reaching specific conditions on altitude, roll angle, or quaternion orientation. This logic ensures robust phase switching and safe recovery.

The complete implementation of the iLQR controller is provided in Appendix B, and the finite state machine is detailed in Appendix C.

Adaptive Flip Control and PX4 Integration

A key feature of the developed system is the **adaptive feedback-based flip control**. During the flip phase, the current roll angle is continuously compared against a theoretically expected roll trajectory. The expected angle is computed as a linear function of time within the flip duration.

Expected roll angle computation:

```
t_local = np.clip(current_time - self.flip_started_time, 0.0, self.flip_durat
roll_expected = 2 * np.pi * t_local / self.flip_duration
roll_current, _, _ = self.euler_from_quaternion(q_current)
roll_error = roll_expected - roll_current
gain_adaptive = gain_base + 0.3 * np.tanh(roll_error)
```

The adaptive gain `gain_adaptive` increases when the roll error grows, compensating for under-rotation by amplifying the target roll and angular velocity. The hyperbolic tangent ensures smooth saturation, avoiding abrupt changes.

State-Driven Phase Transitions

The flip control logic is implemented as a closed-loop system using a finite-state machine. Transitions between phases—takeoff, flip, recovery, and landing—occur exclusively based on real-time feedback, including:

- reaching the required altitude,

- approaching a target roll angle near 2π,

- re-establishing horizontal orientation.

This feedback-based transition mechanism increases robustness and adaptability to drone-specific dynamic properties.

PX4 Integration and Actuation

To transmit control commands to the PX4 autopilot, the computed RPM values are converted into normalized control efforts of the form: [roll, pitch, yaw, thrust]. The transformation from motor RPM to torque and thrust is performed using:

Listing 2.11: RPM to control effort conversion

```
def rpm_to_control(rpm, arm, kf, km):
    w_squared = rpm ** 2
    thrusts = kf * w_squared
    thrust = jnp.sum(thrusts)
```

```
    roll_torque  = arm * (thrusts[1] - thrusts[3])
    pitch_torque = arm * (thrusts[2] - thrusts[0])
    yaw_torque   = km * (w_squared[0] - w_squared[1] + w_squared[2] - w_squar
    return jnp.array([roll_torque, pitch_torque, yaw_torque, thrust])
```

The outputs are normalized as follows:

- roll, pitch, yaw: scaled to the range $[-1, 1]$,

- thrust: scaled to the range $[0, 1]$.

These values are published to the PX4 Fast RTPS Bridge via the `/fmu/in/actuator_controls_0` topic using the `ActuatorControls` message type.

To accept these commands, PX4 must be configured as follows:

- `SYS_CTRL_ALLOC` parameter set to `1`,

- `OFFBOARD` control mode enabled.

The complete implementation of this module is provided in Appendix D.

Adaptive Roll Feedback During Flip and PX4 Command Transmission

The key feature of this implementation is the **adaptive flip control based on real-time feedback**. During the flip phase, the current roll angle is continuously compared with the theoretically expected value, computed based on elapsed time since the start of the flip.

Listing 2.12: Adaptive gain based on roll deviation
```
# Compute local time within flip phase [0, flip_duration]
t_local = np.clip(current_time - self.flip_started_time, 0.0, self.flip_dur

# Linearly expected roll angle from 0 to 2   over flip duration
roll_expected = 2 * np.pi * t_local / self.flip_duration

# Convert current quaternion to roll angle
roll_current, _, _ = self.euler_from_quaternion(q_current)

# Roll deviation
roll_error = roll_expected - roll_current

# Adaptive gain increases if roll deviates from expected trajectory
gain_adaptive = gain_base + 0.3 * np.tanh(roll_error)
```

The adaptive gain modulates the control signal by amplifying the target roll and angular velocity when the real flip lags behind the expected trajectory. This dynamic correction improves trajectory tracking and ensures successful execution even under uncertain drone dynamics.

State-Based Phase Transitions

Transitions between control phases—takeoff, flip, recovery, and landing—are based solely on system feedback:

- reaching a minimum takeoff altitude,

- achieving a roll angle close to 2π,

- restoring horizontal orientation.

This **closed-loop phase transition logic** increases robustness and allows the system to handle varying quadrotor dynamics, especially during aggressive maneuvers.

Control Signal Conversion and PX4 Integration

The controller outputs desired RPM values, which are converted into control torques and thrust in the form $[\text{roll}, \text{pitch}, \text{yaw}, \text{thrust}]$. These values are normalized before being sent to PX4.

Listing 2.13: Conversion from motor RPM to PX4 control format

```
def rpm_to_control(rpm, arm, kf, km):
    w_squared = rpm ** 2
    thrusts = kf * w_squared
    thrust = jnp.sum(thrusts)
    roll_torque  = arm * (thrusts[1] - thrusts[3])    # M2 - M4
    pitch_torque = arm * (thrusts[2] - thrusts[0])    # M3 - M1
    yaw_torque   = km * (w_squared[0] - w_squared[1]
                         + w_squared[2] - w_squared[3])
    return jnp.array([roll_torque, pitch_torque, yaw_torque, thrust])
```

The resulting torques and thrust are normalized as:

- **Roll, pitch, yaw:** scaled to the range $[-1, 1]$,

- **Thrust:** scaled to the range $[0, 1]$.

These normalized commands are published as a `ActuatorControls` message to the topic `/fmu/in/actuator_controls_0` via the PX4 Fast RTPS Bridge.

To receive these commands, PX4 must be configured accordingly:

- `SYS_CTRL_ALLOC = 1` to enable control allocation,

- `OFFBOARD` mode activated for external control.

The full software implementation of this module is provided in Appendix D.

Adaptive Flip Control via Feedback and PX4 Command Interface

The main feature of the implementation is **adaptive flip control through feedback**. During the flip, the current roll angle is compared to the theoretically expected value, which is calculated based on the elapsed time within the flip phase.

Listing 2.14: Adaptive roll control algorithm

```
# Calculate local time from flip start, limited to [0, flip_duration]
t_local = np.clip(current_time - self.flip_started_time, 0.0, self.flip_dur

# Expected roll angle at current time,
# assumed to linearly increase from 0 to 2    during the maneuver
roll_expected = 2 * np.pi * t_local / self.flip_duration

# Obtain current roll angle from the current orientation quaternion
roll_current, _, _ = self.euler_from_quaternion(q_current)

# Calculate roll error between expected and current roll
roll_error = roll_expected - roll_current

# Adaptive gain increases with roll error deviation
# tanh function is used to smooth and limit the gain value
gain_adaptive = gain_base + 0.3 * np.tanh(roll_error)
```

Based on the orientation error and current phase time, the adaptive increment to the target roll is computed. If the actual rotation lags behind the expected trajectory, the node amplifies the target roll angle and angular velocity, compensating for deviations.

Phase transitions occur solely based on feedback — moving to the next phase only happens after meeting the required conditions: reaching the target altitude, roll angle close to 2π, and returning the orientation to horizontal. This approach enhances control stability and allows the system to remain robust against non-standard quadcopter dynamics.

Thus, this node implements a closed-loop feedback control system generating control commands that reliably perform the complex flip maneuver using adaptive parameters and the drone's current state.

For sending control commands to the PX4 autopilot in the format [roll, pitch, yaw, thrust], a system converting RPM values into control torques and thrust is implemented. The converted values are normalized as follows: roll, pitch, and yaw components lie in the range $[-1, 1]$, thrust ranges from $[0, 1]$.

Listing 2.15: Conversion from motor RPM to control torques and thrust

```
def rpm_to_control(rpm, arm, kf, km):
    w_squared = rpm ** 2
    thrusts = kf * w_squared
    thrust = jnp.sum(thrusts)
    roll_torque  = arm * (thrusts[1] - thrusts[3])    # M2 - M4
    pitch_torque = arm * (thrusts[2] - thrusts[0])    # M3 - M1
    yaw_torque   = km * (w_squared[0] - w_squared[1] + w_squared[2] - w_squ
    return jnp.array([roll_torque, pitch_torque, yaw_torque, thrust])
```

The normalized control commands are published as `ActuatorControls` messages to the topic `/fmu/in/actuator_controls_0` via the PX4 Fast RTPS Bridge interface.

To receive these commands, the PX4 autopilot is configured by enabling `OFFBOARD` mode and activating the control allocation system with the parameter `SYS_CTRL_ALLOC = 1`.

The full software implementation of this module is provided in Appendix G.

Setting Execution Frequencies of Functions

To select the control time quantum, we limit possible values from above and below. A typical value for drones during normal flight is 100 ms or more, which is sufficient to perform iterations during navigation in steady trajectory flight.

Since performing a flip requires real sensor data, acceptable characteristics correspond to a 20 ms period. For smaller values, high-speed specialized sensors for pressure, accelerometer, gyroscope, and magnetometer are required.

Data filtering also demands approximately 20 ms on the microcontroller of the flight controller.

Therefore, for this system, the time interval was chosen as 20 ms.

2.5   Demonstration of Operation

For testing and demonstrating the developed control system, the Gazebo simulation environment was used, where a virtual world and a quadcopter model were created (see Appendix D). The model accounted for mass, inertia, aerodynamics, and motor dynamics, allowing tests to be conducted under conditions close to real flight.

To evaluate the performance of the algorithm during the flip maneuver, timestamps marking the start and end of control input computation at each optimization iteration were collected (see Appendix H).

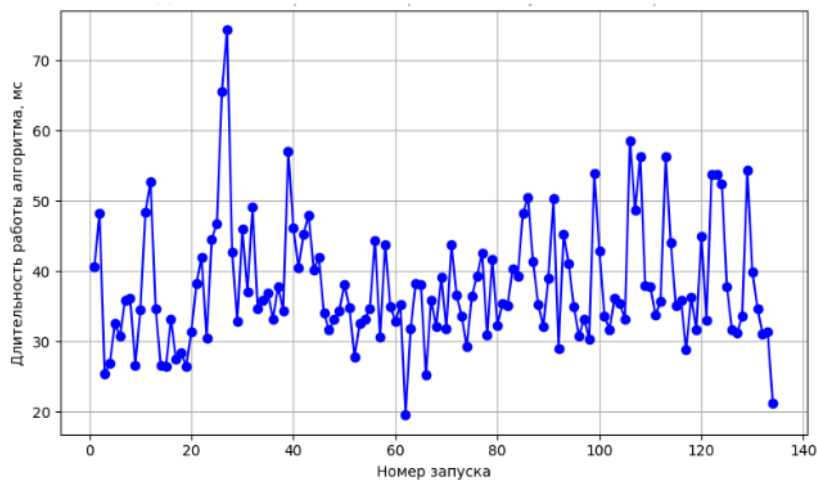The measurement results are shown in Figures 2.6 and 2.7.

Figure 2.6: Running time of the optimal trajectory search algorithm

The dynamics of the drone's position and orientation at the moment of the flip are shown in Figures 2.8 and 2.12 respectively.

As can be seen from Figures 2.8 and 2.12, during the flip, the drone experiences slight deviations in the XY axes - about 0.27 m, which is comparable to the size of the drone. A moderate decrease in the OZ axis - about 2 m - is also highlighted. Drone trajectory in height

Figure 2.10 shows that the drone successfully performed a flip along the roll axis, compensating for a slight displacement that occurred during the maneuver. After rotating approximately 20 degrees beyond the inverted position, the drone stopped rotating and regained stable orientation within 0.5 seconds.
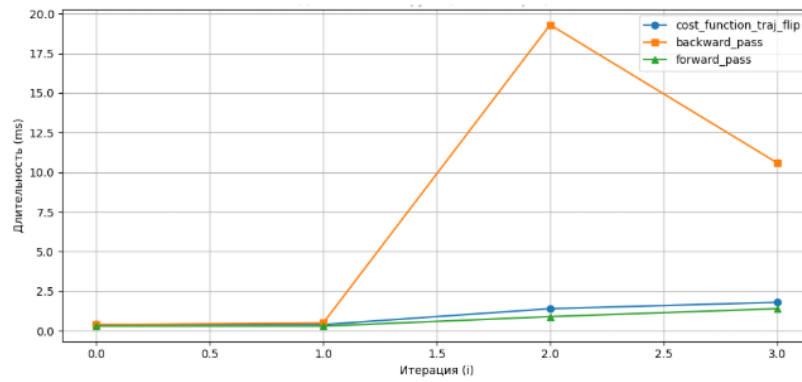
Figure 2.7: Duration of the steps of the algorithm for searching the optimal trajectory

/src/Drone_trajectory.png

Figure 2.8:   Drone trajectory during a flip

For comparative analysis, a simplified control approach was also developed (see Appendix I), based on an open-source code snippet from the Clover platform [?]:

```
void controlFlip(){
    if (t - lastTime > 0.1) {
        flipStage = WAIT;
        stageTime = t;
    }
    lastTime = t;
    if (flipStage == ACCELERATE) {
```

Figure 2.9: Drone trajectory by altitude

```
    thrustTarget = 0.1;
    if (t - stageTime > ACCELERATE_TIME) {
        flipStage = ROTATE;
        stageTime = t;
    }
} else if (flipStage == ROTATE) {
    thrustTarget = 0.2;
// ETC ...
```

The above implementation, based on fixed time intervals and pre-defined angular velocities, does not take into account important dynamic characteristics of the quadcopter — such as thrust interactions, moments of inertia, aerodynamic drag, and gyroscopic effects. This simplified approach limits the accuracy and stability of control during the maneuver.

In practice, the implementation of this method proved ineffective: tuning time intervals did not yield consistent results, and the hard-coded phase durations prevented the maneuver from completing properly. As a result, the approach was modified — transitions between maneuver stages were made dependent on feedback from the drone, particularly the current roll angle.

/src/orientation_flip.png

Figure 2.10: Changing the orientation of the drone during a flip

/src/Drone_trajectory_ptimitive.png

Figure 2.11: Trajectory of drone position at the moment of flip realization (primitive algorithm)

/src/Drone_trajectory_by_alt.png

Figure 2.12: Drone trajectory by altitude

Conclusion

Thus, all the set tasks were accomplished. And the set goal has been achieved. The key contribution is the development of an adaptive control system based on the dynamic model of quadrocopter using iLQR optimizer. For testing, a specialized simulation environment based on Gazebo was created to verify the algorithms in conditions close to the real one. The implemented control system is structured and modular: the algorithms are divided into phases and provided with full negative feedback, which makes it possible to design them as library modules and integrate them into other ROS/ROS 2-systems. A comparative analysis has shown notable advantages of the proposed approach over primitive control. As demonstrated by the simulation results, the iLQR-based system can significantly reduce the altitude sag and provide a more stable and predictable drone behavior when performing complex maneuvers such as flip. The OZ-axis sag with the model was more than halved compared to the simple solution. At the same time, the deviations in the XY axes (about 0.27 m) are comparable to the size of the drone and are considered acceptable, and the recovery from the flip occurred in 0.5 seconds. Although there are scientific publications that discuss the application of iLQR for quadrocopter control, there are currently no open source projects that implement this approach on the ROS2, PX4 stack. Thus, the development represents a unique example of integration of modern robotics tools with soft real-time optimization control, which makes the project relevant and practically significant.

References

Bibliography

[1] Government of the Russian Federation. Decree No. 1630-r of June 21, 2023: On the Approval of the Strategy for the Development of Unmanned Aviation in the Russian Federation until 2030 and for the Future until 2035, and the Plan for Its Implementation. *Collection of Legislation of the Russian Federation*, No. 27, 2023.

[2] Terentyev, V. M. Development of Theoretical Foundations and New Methodological Approaches for the Design of Trajectory Control Systems for Advanced Unmanned Aerial Vehicles. Moscow: *Innovative Mechanical Engineering*, 2023.

[3] Loquercio, A., et al. Learning high-speed flight in the wild. *Science Robotics*, vol. 6, eabg5810, 2021. DOI: 10.1126/scirobotics.abg5810.

[4] Clover Project. Drone Flip Control Documentation. *Clover COEX*, 2024. Available online: https://clover.coex.tech/ru/snippets.html#flip (accessed: Feb. 20, 2025).

[5] Fetisov, V. S., Neugodnikova, L. M., Adamovsky, V. V., Krasnoperov, R. A. Unmanned Aviation: Terminology, Classification, and Current State. Ufa: *Foton*, 2014.

[6] Zelensky, V. A., Kovalev, M. A., Ovakimyan, D. N., Kirillov, V. S. Mathematical model of a generalized kinematic scheme of a quadcopter and its software implementation. *Bulletin of Samara University. Aerospace Engineering, Technologies and Mechanical Engineering*, vol. 23, no. 1, 2024. DOI: 10.18287/2541-7533-2024-23-1-7-20.

[7] Kalyagin, M. Yu., Voloshin, D. A., Mazaev, A. S. Simulation of a quadcopter flight control system in Simulink and Simscape Multibody. *Proceedings of MAI*, no. 112, 2020. DOI: 10.34759/trd-2020-112-20.

[8] Lysukho, G. V., Maslennikov, A. L. Quadcopter: Dynamics and Control. *Polytechnic Youth Journal*, no. 5, 2020. DOI: 10.18698/2541-8009-2020-04-604.

[9] Sun, R.; Zhang, W.; Zheng, J.; Ochieng, W. Y. GNSS/INS Integration with Integrity Monitoring for UAV No-fly Zone Management. *Remote Sensing*, 2020, vol. 12, article 524. DOI: 10.3390/rs12030524.

[10] Hiba, A.; Gáti, A.; Manecy, A. Optical Navigation Sensor for Runway Relative Positioning of Aircraft during Final Approach. *Sensors*, 2021, vol. 21, article 2203. DOI: 10.3390/s21062203.

[11] Yan Gan, Meng Wei-Da, Dong Hou-Go, Feng Ning-Ning. Use of Keypoints and Image Lines in Real-Time Visual-Inertial Odometry. *Gyroscopy and Navigation*, 2023, vol. 31, no. 4, pp. 96–113.

[12] INAV Development Team. INAV: Navigation-enabled flight control software. GitHub repository. Available online: https://github.com/iNavFlight/inav/tree/master/src/main/target (accessed: May 1, 2025).

[13] Betaflight Development Team. Betaflight: Navigation-enabled flight control software. GitHub repository. Available online: https://github.com/betaflight/betaflight/tree/master/src/main/target (accessed: May 1, 2025).

[14] ArduPilot Development Team. ArduPilot: Open Source Autopilot Software. Available online: https://ardupilot.org/ (accessed: Feb. 20, 2025).

[15] Stepanov, O.A., Nosov, A.S. An Informative Route Planning Algorithm for Navigation Using a Map. In: *Proc. of the XXVIII St. Petersburg International Conference on Integrated Navigation Systems*, St. Petersburg, Russia, May 31 – June 2, 2021. St. Petersburg: Concern "Central Research Institute Electropribor", 2021. (in Russian)

[16] Pan, Y., Lin, Q., Shah, H., Dolan, J.M. Safe Planning for Autonomous Driving via Adaptive Constrained ILQR. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Las Vegas, NV, USA, 2020, pp. 2377–2383. DOI: 10.1109/IROS45743.2020.9340886.

[17] Pan, J., Zhang, L., Manocha, D. Collision-Free and Smooth Trajectory Computation in Cluttered Environments. *The International Journal of Robotics Research*, 2012, Vol. 31.

[18] Zhou, B., Gao, F., Wang, L., Liu, C., Shen, S. Robust and Efficient Quadrotor Trajectory Generation for Fast Autonomous Flight. *IEEE Robotics and Automation Letters*, 2019. DOI: 10.1109/LRA.2019.2927938.

[19] Kulathunga, G., Devitt, D., Fedorenko, R., Klimchik, A.S. Path Planning Followed by Kinodynamic Smoothing for Multirotor Aerial Vehicles (MAVs). *Russian Journal of Nonlinear Dynamics*, 2021, Vol. 17, No. 4.

[20] Bestaoui, Y. 3D Flyable Curves for an Autonomous Aircraft. In: *9th International Conference on Mathematical Problems in Engineering, Aerospace and Sciences*, 2012, Vol. 1493.

[21] Barros, A.M., Michel, M., Moline, Y., Corre, G., Carrel, F. A Comprehensive Survey of Visual SLAM Algorithms. *Robotics*, 2022, No. 11.

[22] Yu, H., Wang, Q., Yan, C., Feng, Y., Sun, Y., Li, L. DLD-SLAM: RGB-D Visual Simultaneous Localisation and Mapping in Indoor Dynamic Environments Based on Deep Learning. *Remote Sensing*, 2024, Vol. 16.

[23] Vedadi, A., Yousefi-Koma, A., Yazdankhah, P., Mozayyan, A. Comparative Evaluation of RGB-D SLAM Methods for Humanoid Robot Localization and Mapping. In: *Proceedings of the 2023 International Conference on Robotics and Mechatronics (ICRoM)*, 2023. DOI: 10.1109/ICRoM60803.2023.10412425.

[24] Shan, T., Englot, B., Meyers, D., Wang, W., Ratti, C., Rus, D. LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 5135–5142.

[25] Shan, T., Englot, B., Meyers, D., Wang, W., Ratti, C., Rus, D. LIO-SAM: Tightly-Coupled Lidar Inertial Odometry via Smoothing and Mapping. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Las Vegas, NV, USA, 2020. DOI: 10.1109/IROS45743.2020.9341176.

[26] Zhang, J., Singh, S. LOAM: Lidar Odometry and Mapping in Real-Time. In: *Robotics: Science and Systems Conference (RSS)*, 2014.

[27] Lee, D., Shim, D. RRT-based Path Planning for Fixed-Wing UAVs with Arrival Time and Approach Direction Constraints. In: *International Conference on Unmanned Aircraft Systems (ICUAS)*, 2014.

[28] Bestaoui, Y. 3D Flyable Curves for an Autonomous Aircraft. In: *9th International Conference on Mathematical Problems in Engineering, Aerospace and Sciences*, 2012, Vol. 1493.

[29] Open Source Robotics Foundation. Robot Operating System (ROS) Official Website. URL: https://www.ros.org/, accessed on March 28, 2024.

[30] PX4 Development Team. ROS 2 User Guide — PX4 Documentation. URL: https://docs.px4.io/main/en/ros2/user_guide.html, accessed on March 28, 2024.

Appendix A

Dynamic Model of the Quadcopter

Appendix B

iLQR Controller

Appendix C

Adaptive Goal Planner

Appendix D

Control Signal Generation Module

Appendix E

Gazebo Virtual Environment

Appendix F

Data Collection During Trajectory Optimization

Appendix G

Simplified Flip Task Solution