



**Faculty of Electrical Engineering**  
**Department of Control Engineering**

Bachelor's thesis

# **Automatic Control of an Unmanned Aerial Vehicle in Robot Operating System**

**Jan Machálek**

**Prague, May 2018**  
**Supervisor, Ing. Tomáš Báča**





## Declaration

I declare that I have worked out my thesis separately and that I have listed all the information sources used in accordance with a Methodical Guideline on Ethical Principles in Preparation college final thesis.

Prague, date \_\_\_\_\_

\_\_\_\_\_

signature





## **Acknowledgement**

I would like to thank Ing. Tomáš Báča who has helped me with everything. I would also like to thank all my friends who helped me with my thesis, and my family, which has supported me to study a college and work on this thesis.



## I. Personal and study details

Student's name: **Machálek Jan** Personal ID number: **456902**  
Faculty / Institute: **Faculty of Electrical Engineering**  
Department / Institute: **Department of Control Engineering**  
Study program: **Cybernetics and Robotics**  
Branch of study: **Systems and Control**

## II. Bachelor's thesis details

Bachelor's thesis title in English:

**Automatic Control of an Unmanned Aerial Vehicle in Robot Operating System**

Bachelor's thesis title in Czech:

**Automatické řízení bezpilotní helikoptéry v systému ROS**

Guidelines:

The goal of the work is to design and implement an automatic controller for Unmanned Aerial Vehicles (UAVs) developed in the MRS lab at FEE CTU. The work aims to replace the currently used backstepping controller [1], which the lab obtained in the form of a black box. The thesis should cover the following points:

- \* Design a PID controller [2, 3] which will control the position of a multicopter UAV.
- \* Implement the controller using ROS (Robot Operating System) and integrate it within the software architecture used in the MRS lab.
- \* Design a modular system which will allow connecting the controller to various control reference generators, e.g., currently used MPC tracker.
- \* For testing of the control system, design and implement a simple reference generator, e.g., linear reference generator.
- \* Investigate the backstepping control approach [1] and simulate it using Matlab. Compared it to the PID controller.
- \* Conduct tests and experiments of the implemented control system in realistic ROS Gazebo simulator and compare it to the proprietary controller currently used on the MRS aerial platform.

Bibliography / sources:

- [1] Lee, Taeyoung, Melvin Leok, and N. Harris McClamroch. 'Nonlinear robust tracking control of a quadrotor UAV on SE (3).' *Asian Journal of Control* 15.2 (2013): 391-408.  
[2] T. Baca, 'Control of Relatively Localized Unmanned Helicopters', Bachelor's thesis, CVUT FEL, 2013  
[3] V. Endrych, 'Control and Stabilization of an Unmanned Helicopter Following a Dynamic Trajectory', Master's thesis, CVUT FEL, 2014

Name and workplace of bachelor's thesis supervisor:

**Ing. Tomáš Báča, Multi-robot Systems, FEE**

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

Date of bachelor's thesis assignment: **30.01.2018** Deadline for bachelor thesis submission: **25.05.2018**

Assignment valid until: **30.09.2019**

Ing. Tomáš Báča  
Supervisor's signature

prof. Ing. Michael Šebek, DrSc.  
Head of department's signature

prof. Ing. Pavel Ripka, CSc.  
Dean's signature





## Abstrakt

Tato práce se zabývá návrhem systému řízení polohy pro bezpilotní letecká vozidla (UAV). Vyšetřujeme dva přístupy, PID regulátor a backstepping regulátor. Regulátory byly integrovány do stávajícího prostředí. Oba regulátory byly testovány v simulátoru Gazebo. V případě PID byl také proveden skutečný letový test. V této práci se také popisuje architektura použité platformy. Výsledkem této práce je funkční systém řízení polohy. Je zde popsán proces návrhu nového systému. V pokusech z reálného světa jsme porovnali náš regulátor s původním regulátorem.

## Klíčová slova

ROS, Gazebo, PID, backstepping, MRS group, UAV, řízení, Matlab, Simulink

## Abstract

This thesis is about design of a position control system for Unmanned Aerial Vehicles (UAVs). We investigate two approaches. The first approach is the PID controller, and the backstepping controller. Regulators are then deployed into ROS environment. Regulators are integrated into an already existing pipeline. After the implementation of regulators they are tested in Gazebo simulator. In the case of PID, real flight tests are conducted. The architecture of a UAV platform used in this thesis is described. The result of this thesis is a functional position control system. The design process of the new system is described as well. Using results from the real-world tests, we compared our regulator with the one currently used in MRS group pipeline.

## Keywords

ROS, Gazebo, PID, backstepping, MRS group, UAV, control, Matlab, Simulink


# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1	State of the art . . . . .	1
2	Problem Statement . . . . .	2
3	Outline . . . . .	3
<b>2</b>	<b>Unmanned Aerial Vehicles</b>	<b>4</b>
1	Unmanned Aerial Vehicles in MRS group . . . . .	4
2	Model of Unmanned Aerial Vehicles . . . . .	5
2.1	Coordinates System . . . . .	6
3	Kinematics and Dynamics model of UAV . . . . .	6
3.1	Rotational Matrix . . . . .	6
3.2	Translational Equation of Motion . . . . .	7
3.2.1	Implementation of translation model . . . . .	8
3.2.2	Identification of translation model . . . . .	9
3.2.3	Rotational dynamics of UAV . . . . .	11
3.3	State space model of the UAV . . . . .	11
<b>3</b>	<b>Control</b>	<b>13</b>
1	Design and testing of PID controller . . . . .	13
1.1	Altitude control with PID . . . . .	13
1.2	Longitudinal control with PID for x-axis . . . . .	14
1.3	Longitudinal control with PID for y-axis . . . . .	14
1.4	Completion of PID . . . . .	14
1.4.1	Gain scheduling . . . . .	14
2	Design and testing of backstepping controller . . . . .	15
2.1	Roll control with backstepping . . . . .	15
2.2	Pitch control with backstepping . . . . .	17
2.3	Yaw control with backstepping . . . . .	19
2.3.1	Simulation . . . . .	19
2.4	Altitude control with backstepping . . . . .	20
2.5	Longitudinal Control in x-axis . . . . .	22
2.6	Longitudinal Control in y-axis . . . . .	23
<b>4</b>	<b>Simulation</b>	<b>24</b>
1	Matlab simulation . . . . .	24
1.1	Parameters of the simulation . . . . .	24
1.1.1	Dynamic model and Environment . . . . .	24
1.1.2	PID controller . . . . .	24
1.1.3	Backstepping . . . . .	25
1.2	Resolution of the simulation . . . . .	25

<b>5</b>	<b>Linear reference generator</b>	<b>27</b>
1	The proprietary (original) interface between controller and tracker	27
2	Implementation of the linear tracker . . . . .	28
2.1	Linear tracker mathematically . . . . .	28
2.2	Linear tracker in ROS . . . . .	28
2.3	Structural point of view on linear trucker . . . . .	29
2.3.1	How to make a plugin . . . . .	29
<b>6</b>	<b>Designing of a modular guidance system</b>	<b>31</b>
1	Towards complete modularity . . . . .	32
<b>7</b>	<b>Simulation in Gazebo Simulator</b>	<b>33</b>
1	Description of the simulated scenario . . . . .	33
2	Simulation . . . . .	33
<b>8</b>	<b>Real flight comparison</b>	<b>36</b>
1	Real flight measures . . . . .	36
2	The first real flight . . . . .	36
3	Path for comparison . . . . .	36
4	Control performance . . . . .	37
<b>9</b>	<b>Conclusion</b>	<b>41</b>
	<b>References</b>	<b>42</b>
	<b>CD content</b>	<b>44</b>

## List of Figures

1	Diagram of the control pipeline which includes the MPC tracker and the collision avoidance mechanism [1]. . . . .	2
2	MRS's group hexacopter in flight when equipped with minimal equipment. . . . .	4
3	MRS's group hexacopter onboard equipment [2]. . . . .	5
4	Coordinates system and position vector. . . . .	6
5	Translation model in <code>Matlab Simulink</code> and <code>Gazebo</code> . . . . .	9
6	Simuling model of Unmanned Aerial Vehicles with the embedded stabilization loop closed. . . . .	9
7	Speed in z-axis simulated in <code>Matlab Simulink</code> . . . . .	10
8	Speed in x-axis simulated in <code>Matlab Simulink</code> and <code>Gazebo</code> . . .	11
9	Block diagram of PID regulation. . . . .	13
10	Roll angle part of backstepping controller. . . . .	17
11	Roll and pitch angle part of backstepping controller. . . . .	18
12	Roll, pitch and yaw angle of backstepping controller. . . . .	19
13	Block scheme for simulation backstepping controller on second order system. . . . .	20
14	Output of simulation of backstepping controller. . . . .	20
15	Comparison of the PID and the BackStepping controllers. (a) shows the vertical axis, (b) shows the horizontal axis. . . . .	25
16	Both regulators following reference in 3D space. (a) shows the BackStepping controller, (b) shows the PID controller. . . . .	26
17	Block diagram of the modular system for connecting various control reference generators. . . . .	31
18	Block diagram of modular system for switching various control reference generators, and feedback controller. . . . .	32
19	Step responses comparison between the PID and the original controllers in horizontal axis. . . . .	33
20	Comparison of the PID and original controllers in horizontal plane when following sine reference. (a) shows comparison in x-axis, (b) shows comparison in y-axis. . . . .	34
21	Comparison of both controllers in z-axis and overall view. (a) shows comparison on the PID and original controllers, (b) shows overall view. . . . .	34
22	Comparison of both controllers in 3D space. (a) shows performance of the original controller, (b) shows comparison of both controllers. . . . .	35
23	Trajectory has been generated in <code>Matlab</code> . . . . .	36
24	UAVs estimated position in horizontal plane when controlled by the PID controller. (a) shows following reference in x-axis, (b) shows following reference in y-axis. . . . .	37
25	Estimated position of the UAV in vertical axis and overall view of the performance. (a) shows postion of UAV in z-axis, (b) shows overall view of the flight. . . . .	37



26	Comparison of both controllers in horizontal plane. (a) shows both controllers in x-axis, (b) shows both controllers in y-axis. . .	38
27	Comparison of both controllers in 3D space. . . . .	38
28	Snapshots of testing of the system in real-world conditions. Video from the experiment can be found at <a href="https://www.youtube.com/watch?v=tpn7tCahuIQ">https://www.youtube.com/watch?v=tpn7tCahuIQ</a> . . . . .	39
29	Unmanned aerial vehicle when controlled by the developed control system. . . . .	40



## List of Tables

1	UAVs onboard hardware [2]. . . . .	5
2	Altitude control description of parameters. . . . .	13
3	Longitudinal (x) control description of parameters. . . . .	14
4	Longitudinal (y) control description of parameters. . . . .	14
5	UAV model parameters for Matlab Simulation. . . . .	24
6	PID parameters for Matlab Simulation. . . . .	24
7	Backstepping controller parameters for Simulation, $k_5, k_6, k_7, k_8,$ $k_9, k_{10} \in \mathbb{R}_+$ . . . . .	25
8	CD content. . . . .	44



## Listings

1	Code for gain scheduling. . . . .	15
2	Interface between controller and tracker. . . . .	27
3	How to get node-handle. . . . .	27
4	Creating a subscriber. . . . .	27
5	Linear tracker core. . . . .	29
6	Plugin registration in code. . . . .	29
7	Plugin decription file. . . . .	30
8	Plugin makefile changes. . . . .	30
9	Package.xml adding export tag. . . . .	30
10	Package.xml adding build dependencies. . . . .	30

# Part 1

## Introduction

Flying objects have always had the great fascination of a man which has encouraged him to do all kinds of research and development. Recent technological progress of sensors, actuators, processors and power storage devices caused a notable improvement of unmanned aerial vehicles (UAVs) around us. Owing to those improvements, today's UAVs are both easy to build and control. UAVs are capable of performing complex tasks and staying in the air for more than 25 minutes, based on equipped hardware. UAVs can be used in various ways including aerial surveillance, package delivery, drone racing, search and rescue operations and more [3]. In this thesis, our primary focus is on hexacopter type UAV.

Hexacopters are a favourite subset of UAVs their name comes from the number of rotors they have. Hexacopters still face some challenges in the control field because of its highly nonlinear, multivariable system and its six Degrees of Freedom. These control problems are handled by onboard PixHawk, an industry standard autopilot [2].

In the first section of this thesis, we will be looking at the modeling and control of a hexacopter type UAV. The Second section will be about Matlab [4] Simulink model and we will have look at the outcome of the simulation. We will then move to ROS<sup>1</sup> (Robot Operating System) conceptual description of this problem in this framework. Lastly, we will make a conclusion of the control of the UAV and two different methods: PID [5] and backstepping [6]. The assignment of this thesis is to develop and implement a PID and backstepping controllers for the multi-copter platform used in the MRS group at FEE CTU in Prague. The proposed control system will be tested in realistic Gazebo simulator [7] and the real world.

The contributions of this work are: derivation of an accurate mathematical model of the hexacopter UAV with PixHawk controller, development of linear and non-linear control algorithms and application of those to the derived mathematical model in computer-based simulations and real flight. The thesis will be concluded with a comparison between the developed control algorithms with the focus on their dynamic performance and their ability to stabilize the system.

## 1 State of the art

Several concepts have been proposed to control UAV when the lower regulation is handled by PixHawk. In MRS group, the currently used original controller is based on paper [8] and it is optimal for MRS group tasks. The controller is only a part of regulation pipeline in the figure 1 which is not open source. Whole regulation cascade in the figure 1 has a source code, except for the element  $S0(3)$ .

Two main approaches are PID controller from paper [9] which is an inspiration for this thesis and also another backstepping approach in paper [10] that gives us the tools to construct our own solution allowing the backstepping controller to replace original controller. In [11] paper, there is presented stabilizing control laws synthesis by sliding mode based on backstepping approach. In paper [12] they presented a novel backstepping controller for velocity control of a UAV such as Yamaha R-Max. This controller design is part of an overall objective of landing the R-Max helicopter. In paper [13] a robust control strategy to solve the path

---

<sup>1</sup><http://wiki.ros.org/>



tracking problem for a quadrotor helicopter has been presented. The proposed strategy was designed in consideration of external disturbances like aerodynamic moments. The movement dynamic equations by Lagrange-Euler formalism have been developed. In paper [14] a new approach of the full state backstepping control using sliding mode observer has been applied to a quadrotor unmanned aerial vehicle. Although the behavior of the quadrotor, affected by aerodynamic forces and moments, is non linear and high coupled, the backstepping technique, applied to the helicopter, turns out to be a good starting point to avoid complex nonlinear control solutions. Next well know type of regulation is in work [9] this method is named Sliding Mode Controller which can be used for the same task as the others.

## 2 Problem Statement

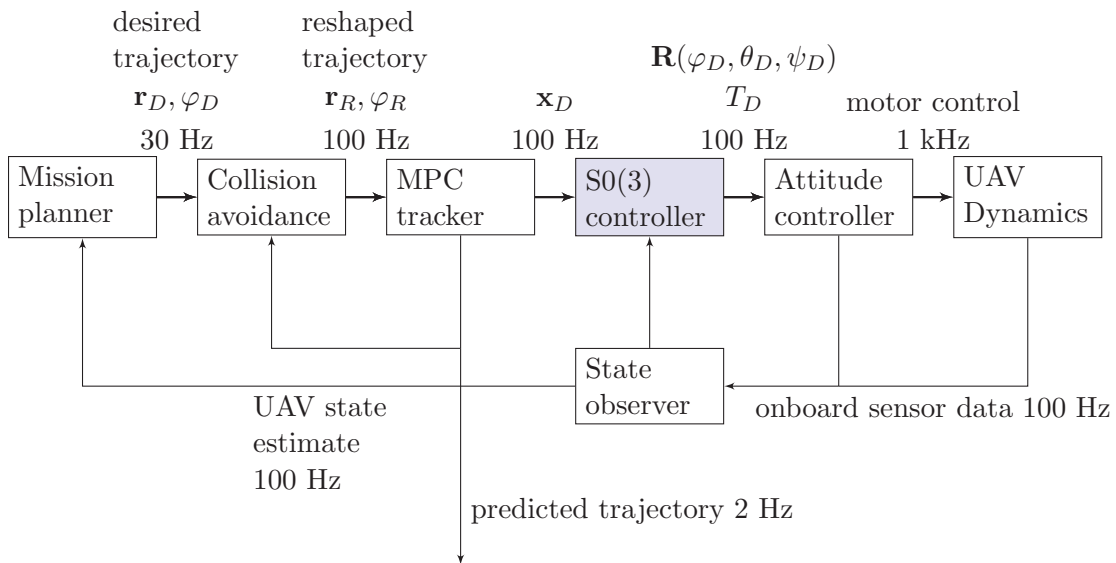


Figure 1: Diagram of the control pipeline which includes the MPC tracker and the collision avoidance mechanism [1].

Figure 1 shows the system from MRS group. Every element can be modified inside of the MRS group with the exception of S0(3) this element which is a binary file. Moreover, the main goal of this thesis is the replacement for original (S0(3)) controller with accessible and editable source code. The source code can be found in CD at the end of this thesis.

Our main goal is to replace a node in ROS that is programmed by the University of Pennsylvania. MRS group has a control pipeline which can be seen in figure 1. Our main goal is the replacement of a node in ROS that is programmed by the University of Pennsylvania so as to enable us to make a publication of the pipeline which would not be otherwise possible due to the original controller not being owned by the MRS group. This modular system can switch between multiple reference trackers.

We need to make core node that contains controller and system that can switch tracker for the controller. Moreover, implemented test linear tracker. The linear tracker should use current interface from the actual system. This tracker will be used for take off.

The last goal is to test the system. The system should be tested in Gazebo simulator and then in real flight. The results are presented by this thesis.

## ■ 3 Outline

The modular system was design and implemented with ability to connect various control references generators. The modular system can be seen in section 6. We also implemented linear tracker generator in section 5. The linear tracker generator was tested in Gazebo simulator. We investigate backstepping control approach and simulated in Matlab Simulink. The backstepping was investigated in section 3. We compared the PID and backstepping in Matlab. The comparison PID and backstepping controllers was conducted in section 4. The proposed system has been successfully implemented into the ROS pipeline. We conducted tests in Gazebo simulator in section 7. We tested PID in real world which can be seen in section 8.

## Part 2

# Unmanned Aerial Vehicles

Unmanned aerial vehicles (UAVs) in figure 2 can be remotely operated by a human, or they can be either operated by a person using a remote operated or fully autonomous. The latter case forms a crucial part of this thesis. For this thesis we will be designing a controller for position of a UAV with specifically prescribed characteristics. We describe the UAV as a mathematical model which takes form of differential equations or diagram in simulink which can be seen in figure 6.



Figure 2: MRS's group hexacopter in flight when equipped with minimal equipment.

## 1 Unmanned Aerial Vehicles in MRS group

Hexacopter have been developed by the MRS group of Czech Technical University in Prague Faculty of Electrical Engineering, with the help of University of Pennsylvania (original controller). Figure 3 shows the hardware equipment of an helicopter of the MRS group.

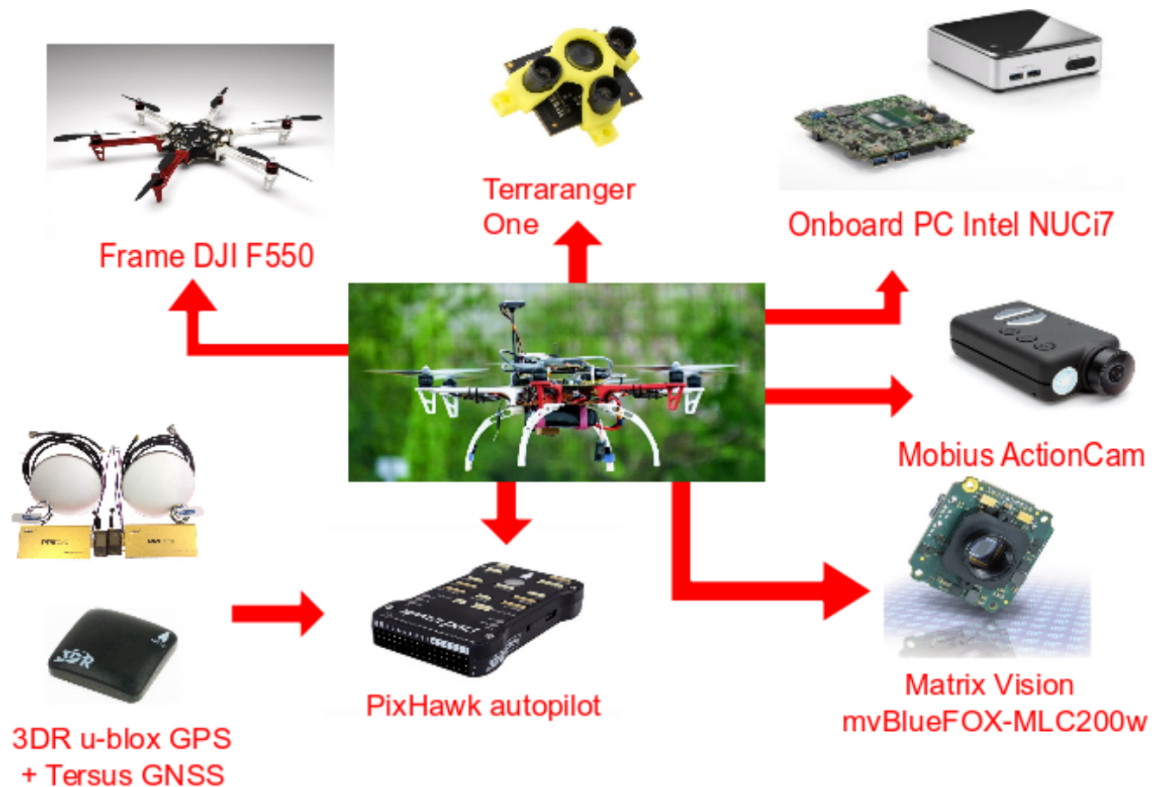


Figure 3: MRS's group hexacopter onboard equipment [2].

PixHawk autopilot	Pixhawk PX4 Flight Controller
Intel NUCi7	Powerful onboard PC running the ROS
Frame	Commercial DJI F550 hexacopter
3DR uBlox GPS, Tersus GNSS	Absolute localization system for outdoor environment.
Terraranger One	External altitude sensor for application, where a knowledge of the precise distance to the surface is required.
Mobius ActionCam	Horizontal camera
mvBlueFOX	Vertical camera

Table 1: UAVs onboard hardware [2].

From our point of view, we need to know dynamics of the whole frame with the already implemented PixHawk autopilot (controller).

## ■ 2 Model of Unmanned Aerial Vehicles

In this chapter, we construct a model of UAV. This model will not include the complete rotational model of UAV. However, we will associate the rotational model done by first-order systems because it already controlled by in build regulators on the UAV itself so this approximation is sufficient.

## 2.1 Coordinates System

We need two coordinate systems to describe our system. The first system is attached to Earth and the second one is connecting the to UAV with its origin in Center of Gravity of the UAV. Schematic in figure 4 of a hexacopter with the body frame of reference denoted by X, Y, Z axes. The four decrease of freedom of the UAV are the Euler angles yaw, pitch, roll and thrust force which is in the direction of  $Z'$ .

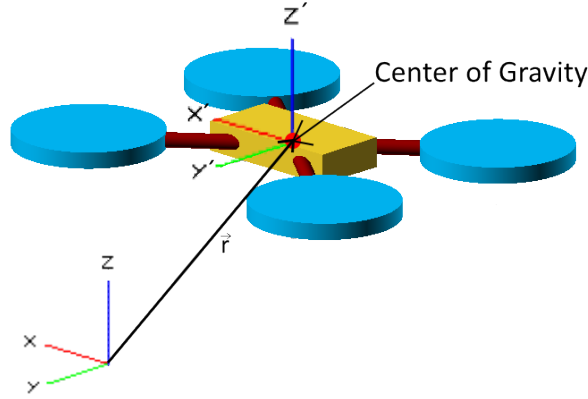


Figure 4: Coordinates system and position vector.

In order to derive the model of the UAV, we first need to define the coordinate frames that will be used. Figure 4 shows two different reference frames. One of them is  $S = [x \ y \ z]$  which is Earth reference system. The other one is the body-fixed frame  $S' = [x' \ y' \ z']$  with the origin in the center of gravity of UAV while  $x'$  points in the direction of flight or forward. The  $y'$  points to the left when UAV is flying forward, and  $z'$  axis points upwards when UAV is in hover mode. The translation between the Earth frame  $S$  and the body frame  $S'$  describes the absolute position of the center of mass of the UAV

$$\mathbf{r} = [x \ y \ z]^T. \quad (1)$$

The rotation  $\mathbf{R}$  from the body frame  $S'$  to the Earth frame  $S$  describes the orientation of the UAV. The orientation of the UAV is described using roll, pitch and yaw angles ( $\varphi$ ,  $\theta$  and  $\psi$ ) representing rotations around the X, Y, and Z axes respectively.

## 3 Kinematics and Dynamics model of UAV

UAV we use Newton-Euler method [9]. For this procedure we need to add some assumptions:

- UAV has rigid and symmetrical construction,
- Center of Gravity is in the origin of body-fixed frame coordinates system of UAV.

### 3.1 Rotational Matrix

For deriving complete Rotational Matrix  $\mathbf{R}$  we need to rotate around all three axes in order to get right result. The inertial frame  $S$  is rotated about its z-axis by the yaw angle  $\psi$  to get

the  $S_1$  frame. The first rotation is given as

$$\mathbf{R}_S^{S_1} = \begin{bmatrix} \cos \psi & \sin \psi & 0 \\ -\sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (2)$$

The resulting frame  $S_1$  is then rotated by the pitch angle  $\theta$  around its y-axis to result in the  $S_2$  frame. The transformation from the  $S_1$  frame to the  $S_2$  frame is given by

$$\mathbf{R}_{S_1}^{S_2} = \begin{bmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{bmatrix}. \quad (3)$$

The last rotation is the rotation of the  $S_2$  frame about its x-axis to result in the body frame. The transformation from the  $S_2$  frame to the body frame  $S'$  is given by

$$\mathbf{R}_{S_2}^{S'} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \varphi & \sin \varphi \\ 0 & -\sin \varphi & \cos \varphi \end{bmatrix}. \quad (4)$$

Finally the rotation matrix or transformation from the inertial frame  $S$  to the body frame  $S'$  is given by

$$\mathbf{R}_S^{S'} = \mathbf{R}_{S_2}^{S'} \mathbf{R}_{S_1}^{S_2} \mathbf{R}_S^{S_1} \quad (5)$$

$$\mathbf{R}_S^{S'} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \varphi & \sin \varphi \\ 0 & -\sin \varphi & \cos \varphi \end{bmatrix} \begin{bmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{bmatrix} \begin{bmatrix} \cos \psi & \sin \psi & 0 \\ -\sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (6)$$

After the multiplication we get complete rotational matrix from  $S$  frame to  $S'$  frame

$$\mathbf{R}_S^{S'} = \begin{bmatrix} \cos \theta \cos \psi & \cos \theta \sin \psi & -\sin \theta \\ \sin \psi \sin \theta \cos \psi - \cos \varphi \sin \psi & \sin \varphi \sin \theta \sin \psi + \cos \varphi \sin \psi & \sin \varphi \cos \theta \\ \cos \varphi \sin \theta \cos \psi + \sin \varphi \sin \psi & \cos \varphi \sin \theta \sin \psi - \sin \varphi \cos \psi & \cos \varphi \cos \theta \end{bmatrix}. \quad (7)$$

For the other direction we simply need to use

$$\mathbf{R}_S^{S'} = \mathbf{R}_{S'}^S{}^T = \mathbf{R}^T. \quad (8)$$

The rotation matrix  $\mathbf{R}$  will be used in formulating the dynamics model of the UAV. The rotation matrix is because some states are measured in the body frame  $S'$ , for example, the thrust forces produced by the propellers while some others are measured in the inertial frame  $S$ , for example, the gravitational forces  $\mathbf{g}$  and the UAV's position. Thus, to have a relation between both types of states, a transformation from one frame to the other is needed.

## ■ 3.2 Translational Equation of Motion

Action quantity is between 0 and 1 for this thesis we will call this quantity thrust  $t$  it makes a force in the direction of  $-z'$  in body frame and this force is described

$$\mathbf{F}_{\text{thrust}} = \begin{bmatrix} 0 \\ 0 \\ -T \end{bmatrix} \quad (9)$$

$T$  is given by

$$T = t \cdot Mt, \quad (10)$$

where  $Mt$  stand for Max thrust is part of the model and has to be identify. This parameter describe Max thrust when  $t = 1$  then  $T$  is equal to Max Thrust.

The translation equations of motion for the UAV are based on Newton's second law and they are derived in the Earth frame  $S$  as

$$m\ddot{\mathbf{r}} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + \mathbf{R}\mathbf{F}_{\text{thrust}}, \quad (11)$$

where  $m$  is UAV's mass,  $g$  is gravitational acceleration and its value is 9.81, m/s<sup>2</sup>,  $\mathbf{r}$  position vector from equation (1) and figure 4,  $\mathbf{R}$  rotation matrix,  $\mathbf{F}_{\text{thrust}}$  force thrust vector from equation (9).

According to Feynman from this moment we will use the Feynman's trigonometric notation to describe  $\sin()$  as  $s()$  and  $\cos()$  as  $c()$ . After that we can rewrite (11) as

$$m\ddot{\mathbf{r}} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + \begin{bmatrix} c(\theta)c(\psi) & s(\psi)s(\theta)c(\psi) - c(\varphi)s(\psi) & c(\varphi)s(\theta)c(\psi) + s(\varphi)s(\psi) \\ c(\theta)s(\psi) & s(\varphi)s(\theta)s(\psi) + c(\varphi)s(\psi) & c(\varphi)s(\theta)s(\psi) - s(\varphi)c(\psi) \\ -s(\theta) & s(\varphi)c(\theta) & c(\varphi)c(\theta) \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -T \end{bmatrix}. \quad (12)$$

When we decompose this vector equation 12 then it yields this result

$$\ddot{x} = -\frac{T}{m} (\cos \varphi \sin \theta \cos \psi + \sin \varphi \sin \psi) \quad (13)$$

$$\ddot{y} = -\frac{T}{m} (\cos \varphi \sin \theta \sin \psi - \sin \varphi \cos \psi) \quad (14)$$

$$\ddot{z} = \frac{T}{m} (\cos \varphi \cos \theta) - g. \quad (15)$$

### ■ 3.2.1 Implementation of translation model

With this set of equations follow with modelling the system using (Matlab Simulink) as shown in figure 5 below this equation are written in blocs named as fcn

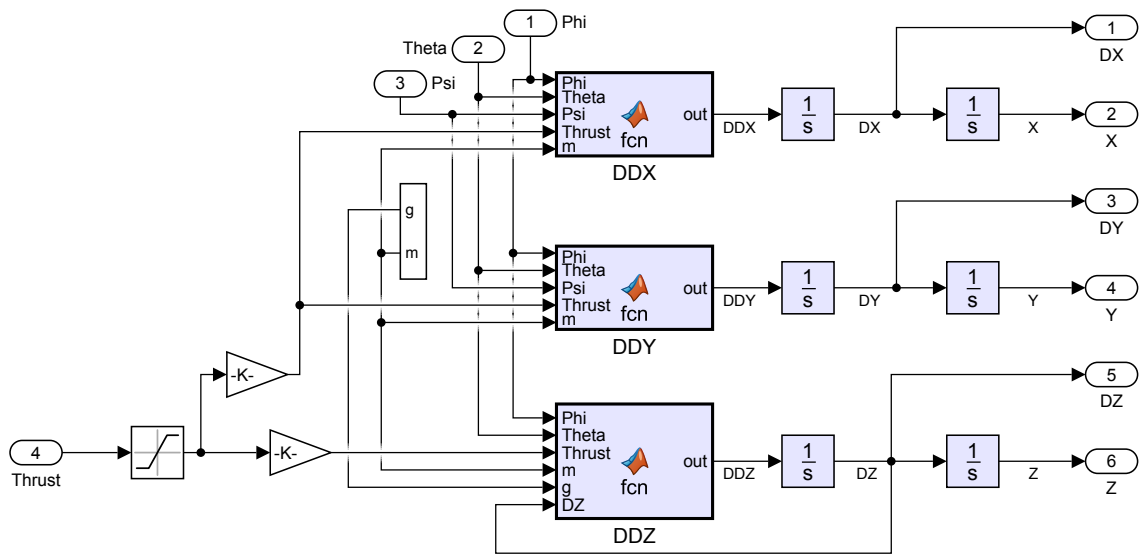


Figure 5: Translation model in Matlab Simulink and Gazebo.

### 3.2.2 Identification of translation model

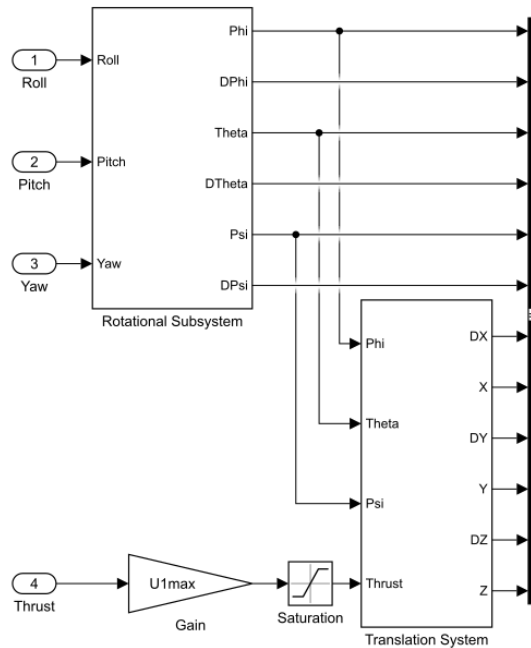


Figure 6: Simulating model of Unmanned Aerial Vehicles with the embeded stabilization loop closed.

- When we already have a model we need to adjust parameters of this model to agree with the model in the Gazebo simulator. The first part of identification consisted of extracting definitons from simulator, such as the UAV mass  $m = 3$  kg and gravitational



acceleration  $g = 9.81 \text{ m/s}^2$ . The rest of parameters (list) was obtained using experiments in the simulator.

- First Experiment was done by original controller which sets roll and pitch angles to zero which means that thrust vector has the direction opposite to gravitational force, then we measured the response with full thrust, gives us the step response of the first order system. After this measurements, we fit the responses of our Matlab model to measured responses.

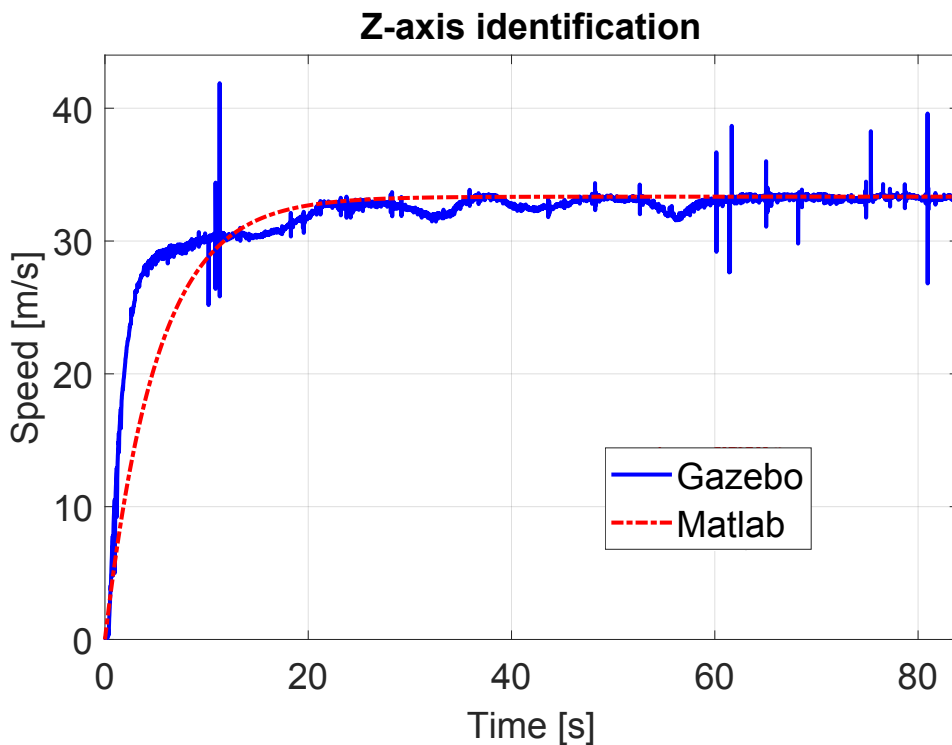


Figure 7: Speed in z-axis simulated in Matlab Simulink.

- When we have z-axis identified then we run another experiment for x or y axes to identify parameters in this axis. The experiment was done by transition UAV from work point to small angle  $\theta$  or  $\varphi$ . In this experiment, we changed to  $5^\circ$  from the operation point. Moreover, we measured the response in this experiment we neglect the rotational dynamics of the system because it matches faster then dynamics of x, y axes. Time constants are very different one of them is in range of 1 - 100 ms (Rotational dynamics) and the other one is in seconds (Translation dynamics). After we had the data from gazebo simulator, we to fit our model.

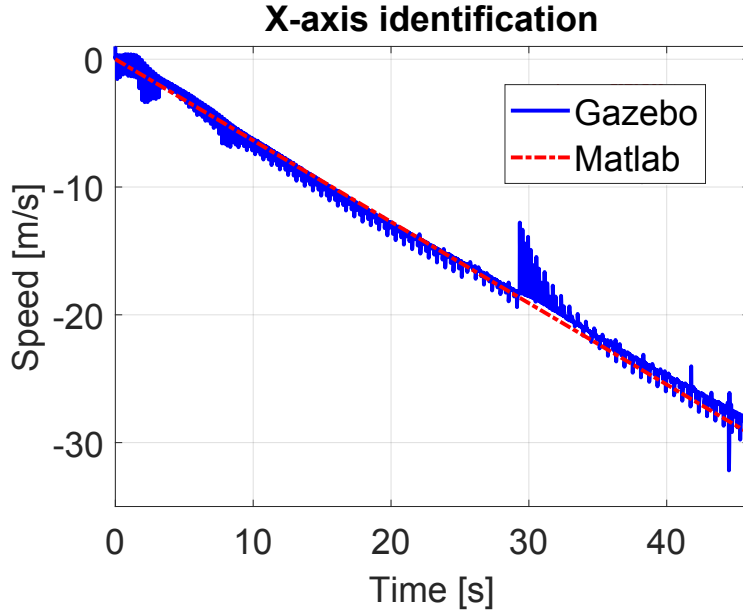


Figure 8: Speed in x-axis simulated in Matlab Simulink and Gazebo.

### ■ 3.2.3 Rotational dynamics of UAV

We came to conclusion that modelling the whole rotational dynamic of the UAV is not necessary because of low-level controllers which take care of this dynamics approximate with second order system between  $\theta_{ref}$  and  $\theta$ ,  $\psi_{ref}$  and  $\psi$ ,  $\varphi_{ref}$  and  $\varphi$ . When we have this Matlab model for experimenting, we start to try to implement backstepping regulation for this model.

### ■ 3.3 State space model of the UAV

If we model rotational system of UAV with low-level regulator on board with second order system then we can write:

$$H(s) = \frac{\omega_i^2}{s^2 + 2\zeta_i\omega_i s + \omega_i^2}, \quad (16)$$

where  $\omega_i$  is natural frequency of the system and  $\zeta_i$  is dumping ratio. This simplification is very convenient because it is simple to tune to identified and also help us with building our

backstepping algorithm:

$$\begin{aligned}
\dot{x}_1 &= x_2 = \dot{\varphi} \\
\dot{x}_2 &= \varphi_{in}\omega_\varphi - 2\zeta_\varphi\omega_\varphi\dot{\varphi} - x_1\omega_\varphi^2 \\
\dot{x}_3 &= x_4 = \dot{\theta} \\
\dot{x}_4 &= \theta_{in}\omega_\theta - 2\zeta_\theta\omega_\theta\dot{\theta} - x_3\omega_\theta^2 \\
\dot{x}_5 &= x_6 = \dot{\psi} \\
\dot{x}_6 &= \psi_{in}\omega_\psi - 2\zeta_\psi\omega_\psi\dot{\psi} - x_5\omega_\psi^2 \\
\dot{x}_7 &= \dot{z} = x_5 \\
\dot{x}_8 &= \ddot{z} = \frac{T}{m}(\cos(\varphi)\cos(\theta)) - g \\
\dot{x}_9 &= \dot{x} = x_7 \\
\dot{x}_{10} &= \ddot{x} = -\frac{T}{m}(\cos\varphi\sin\theta\cos\psi + \sin\varphi\sin\psi) \\
\dot{x}_{11} &= \dot{y} = x_9 \\
\dot{x}_{12} &= \ddot{y} = -\frac{T}{m}(\cos\varphi\sin\theta\sin\psi - \sin\varphi\cos\psi).
\end{aligned} \tag{17}$$

For the simplicity we suppose

$$\omega_\varphi = \omega_\theta = \omega_\psi \tag{18}$$

$$\zeta_\varphi = \zeta_\theta = \zeta_\psi. \tag{19}$$

This is consequence of assumption that our UAV with on board regulator is symmetrical. Thus we can write

$$\dot{x}_2 = \varphi_{in}\omega_0 - 2\zeta\omega_0\dot{\varphi} - x_1\omega_0^2 \tag{20}$$

$$\dot{x}_4 = \theta_{in}\omega_0 - 2\zeta\omega_0\dot{\theta} - x_3\omega_0^2 \tag{21}$$

$$\dot{x}_6 = \psi_{in}\omega_0 - 2\zeta\omega_0\dot{\psi} - x_5\omega_0^2. \tag{22}$$

## Part 3

# Control

In this section, we focus on a mathematical derivation of principles that make backstepping possible. Three controllers will be developed, PID controller, backstepping, and Gain Scheduling PID controllers. Every single controller will be made, Matlab/Simulink and will be used to assess the performance of the developed controllers. For PID with gain, scheduling will be implemented in C++ for ROS and then tested in gazebo simulation.

### 1 Design and testing of PID controller

PID regulator in the implementation to MRS environment we will add gain scheduling to PID and we will get better performance from it. After the state space model of the UAV we can test and develop PID controller on our own as it is shown in figure 9. This task is separated in three smaller tasks: altitude (z-axis) and two longitudinal for x, y axes.

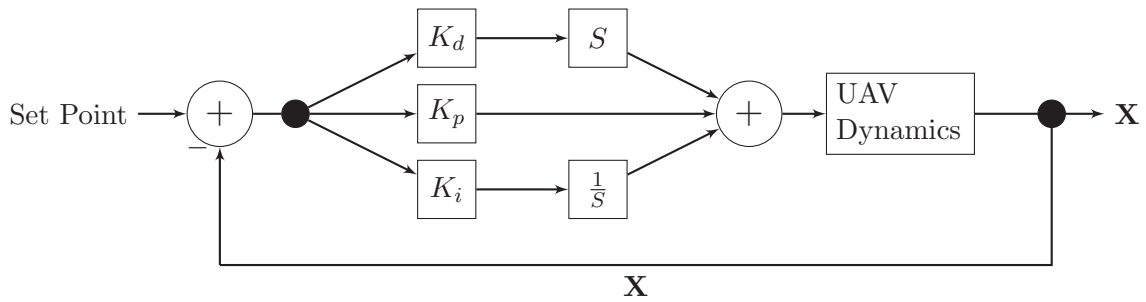


Figure 9: Block diagram of PID regulation.

#### 1.1 Altitude control with PID

A PID controller is developed to control the altitude  $z$  of the UAV. The controller generates the thrust input to controller one layer below (PixHawk) and its output normalized to be from 0 to 1. In this thesis  $T$  stands for thrust. From [5] book about control. The control law for the PID is defined as

$$T = k_p(z - z_{des}) + k_d \frac{\Delta(z - z_{des})}{\Delta t} + k_i \sum (z - z_{des}) \Delta t, \quad (23)$$

where:

$k_p$	proportional gain
$k_d$	derivative gain
$k_i$	integral gain
$z_{des}$	desired altitude
$z$	estimated altitude
$T$	Thrust produced by our PID to next layer (PixHawk autopilot) controller.
$\Delta t$	small change in time, in our case for 200 Hz loop it is five milliseconds.

Table 2: Altitude control description of parameters.

## 1.2 Longitudinal control with PID for x-axis

For x-axis we use this control law which can be written as

$$\theta_{des} = -asin\left(k_p(x - x_{des}) + k_d \frac{\Delta(x - x_{des})}{\Delta t} + k_i \sum (x - x_{des})\Delta t\right), \quad (24)$$

where:

$k_p$	proportional gain
$k_d$	derivative gain
$k_i$	integral gain
$x_{des}$	desired position in x-axis
$x$	estimated position in x-axis
$\theta_{des}$	desired angle for PixHawk controller
$\Delta t$	small change in time, in our case for 200 Hz loop it is five milliseconds.

Table 3: Longitudinal (x) control description of parameters.

## 1.3 Longitudinal control with PID for y-axis

For y-axis we use this control law which can be written as

$$\varphi_{des} = asin\left(k_p(y - y_{des}) + k_d \frac{\Delta(y - y_{des})}{\Delta t} + k_i \sum (y - y_{des})\Delta t\right), \quad (25)$$

where:

$k_p$	proportional gain
$k_d$	derivative gain
$k_i$	integral gain
$y_{des}$	desired position in y-axis
$y$	estimated position in y-axis
$\varphi_{des}$	desired angle for PixHawk controller
$\Delta t$	small change in time, in our case for 200 Hz loop it is five milliseconds.

Table 4: Longitudinal (y) control description of parameters.

## 1.4 Completion of PID

After we have all components of PID, we have to make upgrades. This upgrades consists of Gain scheduling, and in z-axis we take part of back stepping approach, and we decide output of the controller T by  $\cos(\theta) \cdot \cos(\varphi)$  in order to make our PID usable in real-world scenario or in Gazebo simulation. We need to set up operation point for the altitude subsystem and angle output limits on our longitudinal subsystem. For angle,  $\psi$  is not set and control low because we can send our reference to PixHawk autopilot.

### 1.4.1 Gain scheduling

We can improve PID with gain scheduling. We can make a look up table where we can set up  $(k_p, k_d, k_i)$  in order to produce more relevant input to UAV. This table has input difference between estimated position and reference position in space. This difference can be simplified

in to the distance between the reference and estimated position. We can set  $(k_p, k_d, k_i)$  as a function of distance. This addition to PID can improve PIDs properties. In implementation to C++ we come with gain scheduling that is in this case possible. We change  $k_p$  in our angle parts of PID. If UAV is closer then  $distance_{th}$  increase  $k_p$  and of course the other way around. We can obtain  $k_p$  as,

```

if(distance > distanceTh){
    Kp = kpOut;
}else{
    Kp = kpIn;
}

```

Listing 1: Code for gain scheduling.

Where kpOut and kpIn are manually set to some adequate numbers. In this instance we have the sphere which has origin in our set point and the UAV is point in space. This type of collision is easy and quick to check.

## ■ 2 Design and testing of backstepping controller

Backstepping is a recursive control algorithm that can be applied to both linear and nonlinear systems. Nowadays is proposed the use of backstepping and sliding-mode nonlinear control methods to control the UAV which should give better performance in the presence of disturbances. For example, Lee et al. [8] used a backstepping controller to control the position and attitude of a UAV; the proposed controller was tested in a noisy environment and gave an excellent performance. In our case, the backstepping controller is used for altitude and heading control. The backstepping controller is based on the state space model which can be written as 17 backstepping works by designing ordinary control laws (constraints) for some of the state variables. These state variables are called “virtual controls” for the system [15]. Unlike other control algorithms that try to linearize nonlinear systems such as the feedback linearization algorithm, backstepping does not work to cancel the nonlinearities in the system. This backstepping algorithm leads to more flexible designs since some of the nonlinear terms can contribute to the stability of the system.

### ■ 2.1 Roll control with backstepping

The first two states of the state space model in equation (17) is the roll angle  $\varphi$  and its rate of change as

$$\dot{x}_1 = x_2 = \dot{\varphi}, \quad (26)$$

$$\dot{x}_2 = \varphi_{in}\omega_\varphi - 2\zeta_\varphi\omega_\varphi\dot{\varphi} - x_1\omega_\varphi^2. \quad (27)$$

The roll angle  $\varphi$  subsystem is in the strict feedback form (only the last state is a function of the control input  $\varphi_{in}$ ) which makes it easy to pick a positive definite Lyapunov function. Lyapunov function, which can be used to prove the stability of an equilibrium of an ordinary differential equations we obtain

$$L_{ap1} = \frac{1}{2}e_1^2, \quad (28)$$

where  $e_1$  is error between the desired  $\varphi_{in}$  and actual  $\varphi$  roll angle defined as follows

$$e_1 = \varphi_{dis} - \varphi. \quad (29)$$

Now we need to make a time derivative of the Lyapunov function defined in equation (28) is,

$$\dot{L}_{ap1} = e_1 \dot{e}_1 \quad (30)$$

$$= e_1(\dot{\varphi}_{dis} - \dot{\varphi}) \quad (31)$$

System is guaranteed to be stable according to Krasovskii–LaSalle principle if the time derivative of a positive definite Lyapunov function is negative semi-definite [15]. If we want to achieve that choose a positive definite bounding function

$$f_1(e_1) = k_1 e_1^2, \quad (32)$$

to bound  $\dot{L}_{ap1}$  as

$$\dot{L}_{ap1} = e_1(\dot{\varphi}_{dis} - \dot{\varphi}) \leq f_1(e_1) \quad (33)$$

$$= e_1(\dot{\varphi}_{dis} - \dot{\varphi}) \leq k_1 e_1^2. \quad (34)$$

This choice of  $f_1(e)$  is also a common choice for a bounding function for strict feedback systems [15]. where  $k_1$  is a positive constant to preserve positive definite bounding function. To satisfy this inequality the virtual control input can be chosen to be

$$\dot{\varphi}_d = \dot{\varphi}_{dis} + k_1 e_1. \quad (35)$$

A new error variable  $e_2$  to be the deviation of the state  $\dot{\varphi}$  from its desired value

$$e_2 = \dot{\varphi} - \dot{\varphi}_{dis} - k_1 e_1. \quad (36)$$

Now we have opportunity to rewriting Lyapunov's function time derivative  $L_{ap1}$  in the new coordinate  $(e_1, e_2)$  and we get

$$\dot{L}_{ap1} = e_1(\dot{\varphi}_{dis} - (e_2 + \dot{\varphi}_{dis} + k_1 e_1)) \quad (37)$$

$$= -e_1 e_2 - k_1 e_1^2. \quad (38)$$

We have to be careful because the presence of the term  $e_1 e_2$  in  $\dot{L}_{ap1}$  may not be a negative semidefinite time derivative but this will be taken care of in the next iteration of the backstepping algorithm. The next step is to augment the first Lyapunov function  $L_{ap1}$  with a quadratic term in the second error variable  $e_2$  to get a positive definite  $L_{ap2}$ :

$$L_{ap2} = L_{ap1} + \frac{1}{2} e_2^2. \quad (39)$$

next step in the algorithm has to be the time derivation of equation (39) thus we get

$$\dot{L}_{ap2} = \dot{L}_{ap1} + e_2 \dot{e}_2 \quad (40)$$

$$= -e_1 e_2 - k_1 e_1^2 + e_2(\dot{\varphi} - \ddot{\varphi}_{des} - k_1 \dot{e}_1). \quad (41)$$

Choosing the positive definite bounding function to be  $f_2(e) = -k_1 e_1^2 - k_2 e_2^2$  where  $k_2$  is a positive definite and substituting by the value of  $\dot{\varphi}$  from equation (27) brings us to the following inequality:

$$\dot{L}_{ap2} = -e_1 e_2 - k_1 e_1^2 + e_2(\varphi_{in} \omega_0 - 2\zeta \omega_0 \dot{\varphi} - \varphi \omega_0^2 - \ddot{\varphi}_{des} - k_1 \dot{e}_1) \leq -k_1 e_1^2 - k_2 e_2^2. \quad (42)$$

Solving the last inequality, the control input  $\varphi_{ref}$  can be written as,

$$\varphi_{in} = 2\zeta \dot{\varphi} + \varphi \omega_0 + \frac{1}{\omega_0} (\ddot{\varphi}_{des} + k_1 \dot{e}_1 - k_2 e_2) \quad (43)$$

The end of this algorithm tells us how to control roll angle  $\varphi$ . Now we can continue to next step in our journey for complete backstepping regulator. If you look to next figure 10 you can see that a lot of controllable inputs to our UAV is not connected so we will make our regulator better when we make this exact procedure for pitch angel  $\theta$ . In order to make next part of our regulator we will repeat previous process once again.

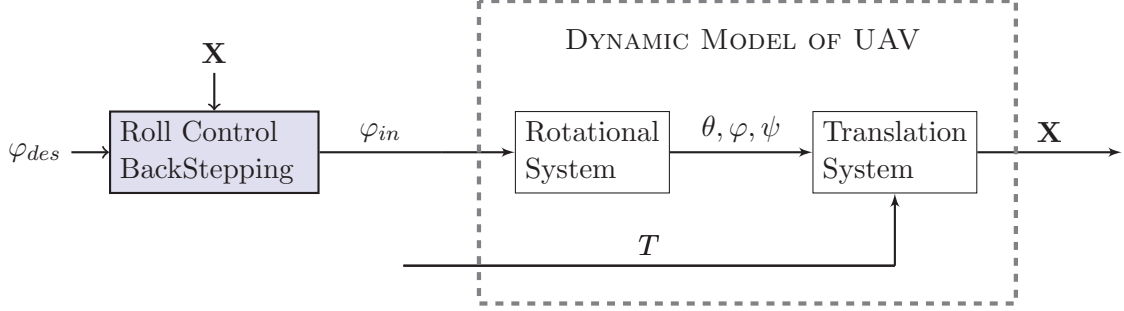


Figure 10: Roll angle part of backstepping controller.

## 2.2 Pitch control with backstepping

The pitch controller is derived with same procedure as the roll controller. The states used are angular speed and pitch angle are express as

$$\dot{x}_3 = x_4 = \dot{\theta}, \quad (44)$$

$$\dot{x}_4 = \theta_{in}\omega_0 - 2\zeta\omega_0\dot{\theta} - x_3\omega_0^2. \quad (45)$$

Now we need to choose Lyapunov function

$$L_{ap3} = \frac{1}{2}e_3^2, \quad (46)$$

where  $e_3$  is error between the desired  $\theta_{in}$  and actual  $\theta$  pitch angle defined as follows

$$e_3 = \theta_{dis} - \theta, \quad (47)$$

Now we need to make a time derivative of the Lyapunov function defined in Equation (46) as

$$\dot{L}_{ap3} = e_3\dot{e}_3 \quad (48)$$

$$= e_3(\dot{\theta}_{dis} - \dot{\theta}). \quad (49)$$

System is guaranteed to be stable according to Krasovskii–LaSalle principle if the time derivative of a positive definite Lyapunov function is negative semi-definite [15]. If we want to achieve that choose a positive definite bounding function

$$f_3(e_3) = k_3e_3^2 \quad (50)$$

to bound  $\dot{L}_{ap3}$  as

$$\dot{L}_{ap3} = e_3(\dot{\theta}_{dis} - \dot{\theta}) \leq f_3(e_3) \quad (51)$$

$$= e_3(\dot{\theta}_{dis} - \dot{\theta}) \leq k_3e_3^2. \quad (52)$$



This choice of  $f_3(e)$  is also a common choice. Where  $k_3$  is a positive constant to preserve positive definite bounding function. To satisfy this inequality the virtual control input can be chosen to be

$$\dot{\theta}_d = \dot{\theta}_{dis} + k_3 e_3. \quad (53)$$

A new error variable  $e_4$  to be the deviation of the state  $\dot{\theta}$  from its desired value

$$e_4 = \dot{\theta} - \dot{\theta}_{dis} - k_3 e_3. \quad (54)$$

Now we have opportunity to rewriting Lyapunov's function time derivative  $L_{ap3}$  in the new coordinate  $(e_3, e_4)$  and we get,

$$\dot{L}_{ap3} = e_3(\dot{\theta}_{dis} - (e_4 + \dot{\theta}_{dis} + k_3 e_3)) \quad (55)$$

$$= -e_3 e_4 - c_3 e_3^2. \quad (56)$$

We have to be careful because the presence of the term  $e_3 e_4$  in  $\dot{L}_{ap1}$  may not be a negative semidefinite but this will be taken care of in the next iteration of the backstepping algorithm. The next step is to augment the first Lyapunov function  $L_{ap3}$  with a quadratic term in the second error variable  $e_4$  to get a positive definite  $L_{ap3}$  as

$$L_{ap4} = L_{ap3} + \frac{1}{2} e_4^2. \quad (57)$$

The next step in the algorithm has to be the time derivation of equation (57) thus we get

$$\dot{L}_{ap4} = \dot{L}_{ap3} + e_3 \dot{e}_4 \quad (58)$$

$$= -e_3 e_4 - k_3 e_3^2 + e_4(\ddot{\theta} - \ddot{\theta}_{des} - k_3 \dot{e}_3). \quad (59)$$

Choosing the positive definite bounding function to be  $f_4(e) = -k_3 e_3^2 - k_4 e_4^2$  where  $k_4$  is a positive definite and substituting by the value of  $\ddot{\theta}$  from equation (45) brings us to the following inequality,

$$\dot{L}_{ap4} = -e_3 e_4 - k_3 e_3^2 + e_4(\theta_{in} \omega_0 - 2\zeta \omega_0 \dot{\theta} - \theta \omega_0^2 - \ddot{\theta}_{des} - k_3 \dot{e}_3) \leq -k_3 e_3^2 - k_4 e_4^2. \quad (60)$$

Solving the last inequality, the control input  $\theta_{ref}$  can be written as

$$\theta_{in} = 2\zeta \dot{\theta} + \theta \omega_0 + \frac{1}{\omega_0} (\ddot{\theta}_{des} + k_3 e_3^2 - k_4 e_4). \quad (61)$$

We concluded the same output as before but now we can do the same for yaw angle  $\psi$  we will repeat this procedure once again to come up with last angle controller for our complete angle control.

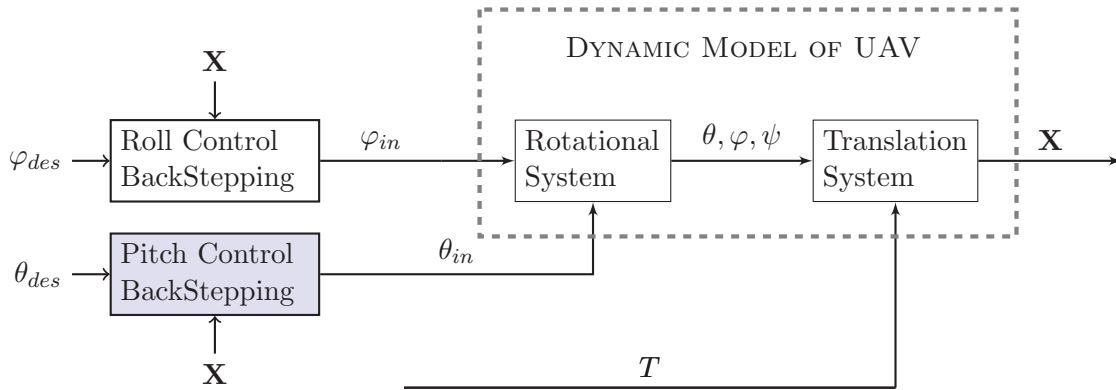


Figure 11: Roll and pitch angle part of backstepping controller.

## ■ 2.3 Yaw control with backstepping

Following exactly the same steps as the pitch and roll controllers, the control input for the yaw angle is derived to be

$$\psi_{in} = 2\zeta_{\psi}\dot{\psi} + \psi\omega_{\psi} + \frac{1}{\omega_{\psi}}(\ddot{\psi}_{des} + k_5e_5^2 - k_6e_6). \quad (62)$$

And the errors ( $e_5, e_6$ ) are

$$e_5 = \psi_{dis} - \psi \quad (63)$$

$$e_6 = \dot{\psi} - \dot{\psi}_{dis} - k_5e_5. \quad (64)$$

We would like to point out that  $k_5, k_6$  are positive constants.

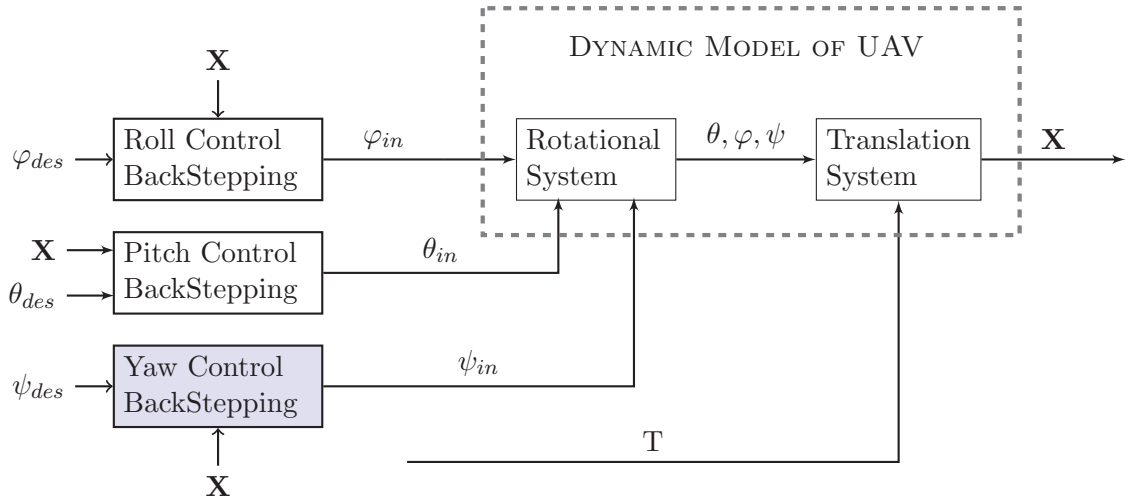


Figure 12: Roll, pitch and yaw angle of backstepping controller.

In this moment we have controllers for all three degree of freedom pitch roll and yaw. All of this is already on board of our UAV so it was not necessary but when we were performing this steps we come to conclusion that this exercise was irreplaceable when we applied this algorithm to our real problem which is making  $\psi_{des}, \theta_{des}$  and  $\psi_{des}$  but the last thing to do is thrust, which should be done in similar manner and it is useful in our real application.

### ■ 2.3.1 Simulation

After derivation of this controllers we simulate it in Matlab Simulink. The simulation was conducted with the following parameters:

$$\begin{aligned} \omega_0 &= 0.8 \\ \zeta &= 0.9 \\ k_1 &= 6 \\ k_2 &= 50 \end{aligned} \quad (65)$$

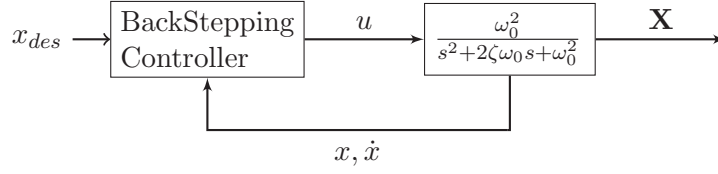


Figure 13: Block scheme for simulation backstepping controller on second order system.

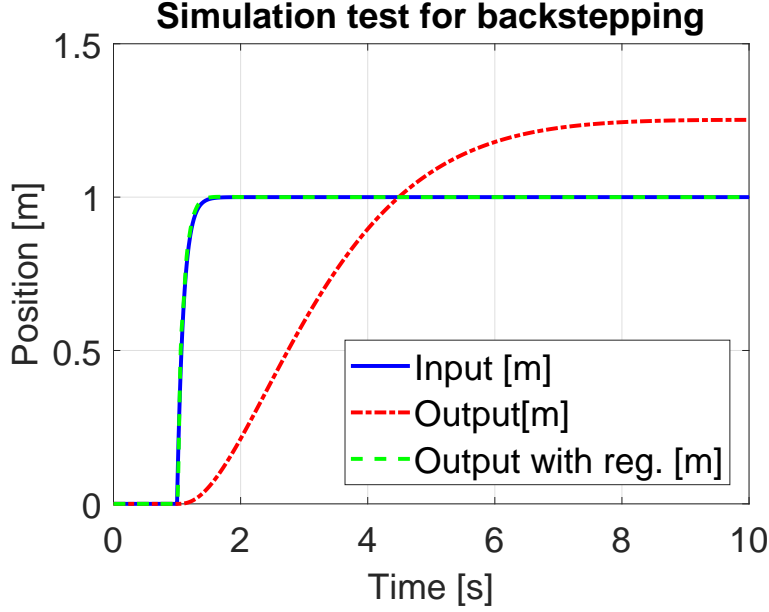


Figure 14: Output of simulation of backstepping controller.

In a figure 14 we can be seen how the controller improves characteristics of our plant and what was very interesting to us is that if we changed some parameter of the system we did not see any changes to our regulated output. After concluding that backstepping is so powerful, we continue our work towards our regulator. Next step is to invent altitude control for our UAV, and that is done in next section.

## 2.4 Altitude control with backstepping

In this subsection, we will try to construct part of our backstepping controller which is responsible for thrust. This thrust we call  $T$ , and it has to be derived. The first two states of the state space model in Equation (17) are related to the thrust as

$$\dot{x}_4 = \dot{z} = x_5 \quad (66)$$

$$\dot{x}_5 = \ddot{z} = \frac{T}{m} (\cos(\varphi) \cos(\theta)) - g. \quad (67)$$

The thrust  $T$  is in the strict feedback form (only the last state is a function of the control input  $T$ )

$$L_{ap5} = \frac{1}{2} e_5^2, \quad (68)$$

where  $e_5$  is an error between the desired  $z_{des}$  and actual  $z$  z-axis position defined as

$$e_5 = z_{des} - z. \quad (69)$$

Now we need to make a time derivative of the Lyapunov function defined in Equation (68) as

$$\dot{L}_{ap5} = e_5 \dot{e}_5 \quad (70)$$

$$= e_5(\dot{z}_{dis} - \dot{z}). \quad (71)$$

When we want to achieve a positive definite bounding function

$$f_5(e_5) = k_5 e_5^2, \quad (72)$$

to bound  $\dot{L}_{ap5}$  as

$$\dot{L}_{ap5} = e_5(\dot{z}_{dis} - \dot{z}) \leq f_5(e_5) \quad (73)$$

$$= e_5(\dot{z}_{dis} - \dot{z}) \leq k_5 e_5^2, \quad (74)$$

where  $k_5$  is a positive constant to preserve positive definite bounding function. To satisfy this inequality the virtual control input can be chosen as

$$\dot{z}_d = \dot{z}_{dis} + k_5 e_5. \quad (75)$$

A new error variable  $e_6$  to be the deviation of the state  $\dot{z}$  from its desired value is

$$e_6 = \dot{z} - \dot{z}_{dis} - k_5 e_5. \quad (76)$$

Now we have an opportunity to rewriting the Lyapunov's function time derivative  $L_{ap5}$  in the new coordinate  $(e_5, e_6)$  and we get

$$\dot{L}_{ap5} = e_5(\dot{z}_{dis} - (e_6 + \dot{z}_{dis} + k_5 e_5)) \quad (77)$$

$$= -e_5 e_6 - k_5 e_5^2. \quad (78)$$

Now we make second iteration of backstepping algorithm.

$$L_{ap6} = L_{ap5} + \frac{1}{2} e_6^2 \quad (79)$$

next step in the algorithm has to be the time derivation of equation (79) thus we got,

$$\dot{L}_{ap6} = \dot{L}_{ap5} + e_6 \dot{e}_6 \quad (80)$$

$$= -e_5 e_6 - k_5 e_5^2 + e_6(\ddot{z} - \ddot{z}_{des} - k_5 \dot{e}_5). \quad (81)$$

Choosing the positive definite bounding function to be  $f_6(e) = -k_5 e_5^2 - k_6 e_6^2$  where  $k_6$  is a positive definite and substituting by the value of  $\ddot{z}$  from equation (67) brings us to the following inequality

$$\dot{L}_{ap6} = -e_5 e_6 - k_5 e_5^2 + e_6 \left( \frac{T}{m} (\cos(\varphi) \cos(\theta)) - g - \ddot{z}_{des} - k_5 \dot{e}_5 \right) \leq -k_5 e_5^2 - k_6 e_6^2. \quad (82)$$

Solving the last inequality, the control input  $T$  can be written as

$$T = \frac{m}{\cos(\varphi) \cos(\theta)} (-e_5 - \ddot{z}_{des} + g - k_5 \dot{z}_{des} + k_6 \dot{z} + k_5 \dot{z}). \quad (83)$$

The end of this algorithm tells us how to control thrust (83) according to our altitude and our desired altitude. Simulation are shown in end of this section.

## ■ 2.5 Longitudinal Control in x-axis

According to [10] we can define error as

$$e_7 = x_{des} - x. \quad (84)$$

Considering a Lyapunov function

$$L_{ap7} = \frac{1}{2}e_7^2 > 0 \quad (85)$$

we can rewrite this as follows

$$\dot{L}_{ap7} = e_7\dot{e}_7 = e_7(\dot{x}_{des} - \dot{x}) \quad (86)$$

In equation (86) the virtual control  $\dot{x}_v$  can be chosen as

$$\dot{x}_v = \dot{x}_{des} + k_7 e_7. \quad (87)$$

After this step we can define another error  $e_8$  as

$$e_8 = \dot{x}_v - \dot{x} = \dot{x}_{des} + k_7 e_7 - \dot{x}, \quad (88)$$

and then we get

$$\dot{e}_8 = \ddot{x}_{des} + k_7 \dot{e}_7 - \frac{T}{m} \Lambda. \quad (89)$$

Next iteration brings new Lyapunov's function,

$$L_{ap8} = L_{ap7} + \frac{1}{2}e_8^2 > 0 \quad (90)$$

Then we substitute from equation (89) and get

$$\dot{L}_{ap8} = \dot{L}_{ap7} + e_8 \left( \ddot{x}_{des} + k_7 \dot{e}_7 - \frac{T}{m} \Lambda \right). \quad (91)$$

If we combine equation (86) and equation (88) we get

$$\dot{L}_{ap8} = -k_7 e_7^2 + e_7 e_8 + e_8 \left( \ddot{x}_{des} + k_7 \dot{e}_7 - z \frac{T}{m} \Lambda \right), \quad (92)$$

where  $\Lambda = 1/m + \cos(\varphi) \sin(\theta) \cos(\psi) + \sin(\varphi) \sin(\psi)$ . It is easy to see from (92) that the fictitious control  $\Lambda_x$  can be considered as the orientation responsible for the motion in  $x$  position, then we get

$$\Lambda_x = \frac{m}{T} (\ddot{x}_{des} + k_7 \dot{e}_7 + e_7 + k_8 e_8). \quad (93)$$

Then, it is possible to find the desired pitch angle  $\theta$  in order to track the reference of  $x$  position, this angle is given by

$$\theta_{des} = \sin^{-1} \frac{\Lambda_x - \sin \varphi \sin \psi}{\cos \varphi \cos \psi} \quad (94)$$

After we have derived desired angle  $\theta$  we will repeat same steps in order to derive desired angle  $\varphi$ .

## ■ 2.6 Longitudinal Control in $y$ -axis

Since the fictitious control  $F_y$  can be considered as the orientation responsible for the motion in  $y$  position, we obtain

$$F_y = \frac{m}{T} (\ddot{y}_{des} + k_9 \dot{e}_9 + e_9 + k_{10} e_{10}), \quad (95)$$

where  $F$  is

$$F = \frac{1}{m} + \cos(\varphi) \sin(\theta) \sin(\psi) - \sin(\varphi) \cos(\psi) \quad (96)$$

and the errors  $(e_9, e_{10})$  are

$$e_9 = y_{des} - y \quad (97)$$

$$e_{10} = \dot{y}_{des} + k_9 e_9 - \dot{y}. \quad (98)$$

Finally, replacing  $\theta$  in (95) by  $\theta_{des}$  from (94), it is possible to find the desired roll angle  $\varphi_{des}$  as

$$\varphi_{des} = \sin^{-1} (F_y \sin(\psi) - F_y \cos(\psi)). \quad (99)$$

After this last step we now have in our hands complete backstepping regulator which we will compare to PID controller design in next section.

# Part 4

## Simulation

### 1 Matlab simulation

In this section, we will focus on comparing backstepping regulator and PID in Matlab Simulink. We will test them on step inputs, and we will change an amplitude of our steps function, and we will observe how the output of our dynamic model will change in order to determine better controller. To be able to compare reasonably between the two implemented control techniques, the response graph of the system under the effect of each controllers was plotted overlap one to another.

#### 1.1 Parameters of the simulation

To test our developed controllers we need set up our dynamic model and also parameters and constants of our controllers.

##### 1.1.1 Dynamic model and Environment

For UAV model we will use this parameters:

Symbol	Value	Unit	Description
m	2.97	[kg]	mass of the UAV
g	9.81	[m/s <sup>2</sup> ]	gravitational constant of the environment
$t_1$	18.15	[-]	constant that realize normalization of thrust 1
$t_2$	48.57	[-]	constant that realize normalization of thrust 2

Table 5: UAV model parameters for Matlab Simulation.

##### 1.1.2 PID controller

For PID controller it has been used,

Symbol	Value	Unit	Description
$P_z$	1.1	[-]	proportional constant in altitude
$I_z$	0.9	[-]	integral constant in altitude
$D_z$	1.02	[-]	derivative constant in altitude
$P_x$	1.12	[-]	proportional constant in longitude (x-axis)
$I_x$	0.01	[-]	integral constant in longitude (x-axis)
$D_x$	2.1	[-]	derivative constant in longitude (x-axis)
$P_y$	1.12	[-]	proportional constant in longitude (y-axis)
$I_y$	0.01	[-]	integral constant in longitude (y-axis)
$D_y$	2.1	[-]	derivative constant in longitude (y-axis)

Table 6: PID parameters for Matlab Simulation.

As you can see in our simulation we presumed symmetry in x and y axes therefore the constants are same for x and y axes.

### 1.1.3 Backstepping

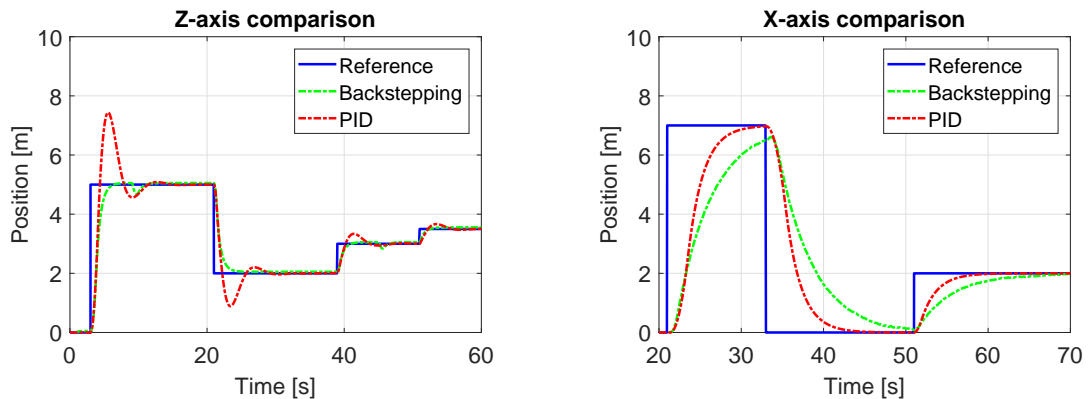
For backstepping controller we used this constants that were derived in section 3.3 and their values are,

Symbol	Value	Unit
$k_5$	18	[-]
$k_6$	1.85	[-]
$k_7$	17	[-]
$k_8$	0.18	[-]
$k_9$	8.5	[-]
$k_{10}$	0.385	[-]

Table 7: Backstepping controller parameters for Simulation,  $k_5, k_6, k_7, k_8, k_9, k_{10} \in \mathbb{R}_+$ .

### 1.2 Resolution of the simulation

The simulation was done in software called Matlab Simulink, which is provided by our university. A version of this software is 2017a, and all files necessary to run the Matlab simulation can be acquired by sending email to machaj45@fel.cvut.cz or on CD. The whole ROS node system form MRS group will be available in the near future on <http://mrs.felk.cvut.cz/platform>. From figure 15 (a) is visible that for PID without Gain Scheduling is difficult to not overshoot when the steps are bigger. On the other hand, the backstepping controller has no problem with that. In the next situation, PID is managing to give better results then backstepping controller, and it looks that backstepping controller is somehow under regulates int longitudinal region as can be seen in 15 (b). From 16 can be seen that the last change of reference any of controllers did not make it in time, so they start to reduce their altitudes in order to produce zero regulation error.



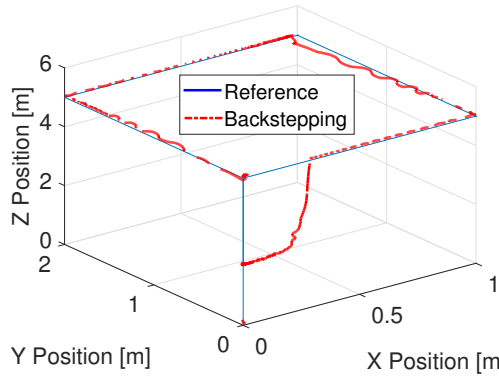
(a) Comparison of the PID and the BackStepping controllers in vertical axis.

(b) Comparison of the PID and the BackStepping controllers in horizontal axis.

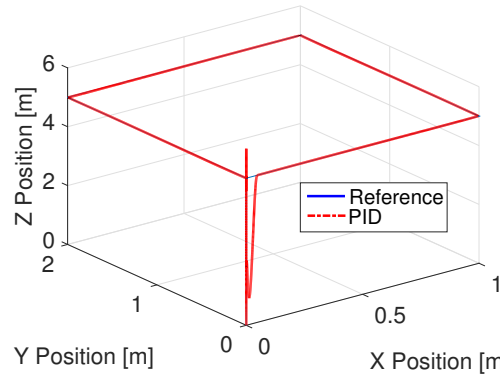
Figure 15: Comparison of the PID and the BackStepping controllers. (a) shows the vertical axis, (b) shows the horizontal axis.

From 16 can be seen that the last change of reference any of controllers did not make it in time so they starts to reduce their altitudes in order to make zero regulation error.





(a) Flight of simulate UAV when regulated by the BackStepping controller.



(b) Flight of simulate UAV when regulated by the PID controller.

Figure 16: Both regulators following reference in 3D space. (a) shows the BackStepping controller, (b) shows the PID controller.

In figure 16 a we can clearly see that we did not give enough time for our regulator to get to the point a the end of its trajectory. In conclusion we picked to implementation PID because of its simplicity and for not that poorly at all.

## Part 5

# Linear reference generator

## 1 The proprietary (original) interface between controller and tracker

In order to implement linear reference generator we need to consider that in the MRS pipeline we already had a tracker. We need to make sure that the old tracker is still able to connect to our regulator. Thus we decided to use their interface which defines the communication between the controller and reference generator. This interface can be seen in listings 2. The interface `trackers_manager::Tracker` consists of five methods and one constructor.

```
class <TrackerName> : public trackers_manager::Tracker {
public:
    <TrackerName>(void); // Constructor
    void Initialize(const ros::NodeHandle &nh,
        const ros::NodeHandle &parent_nh);
    bool Activate(const quadrotor_msgs::PositionCommand::ConstPtr &cmd);
    void Deactivate(void);
    const quadrotor_msgs::PositionCommand::ConstPtr
    update(const nav_msgs::Odometry::ConstPtr &msg);
    const quadrotor_msgs::TrackerStatus::Ptr status();
};
```

Listing 2: Interface between controller and tracker.

The methods that trackers need to implement are *Initialize*, *Activate*, *Deactivate* and *update*. The initialize method sets up the tracker. The initialize method is called once at the beginning. The initialize method should not contain never ending loops. The initialize method should not contain long term operations. We give the initialize method two pointers to node-handle one of them is for the node of the tracker and the second one is parent node which is in our case our PID controller. In *Initialize*, we fill the first node-handle with our node-handle pointer

```
ros::NodeHandle priv_nh(nh, "linear_tracker");
```

Listing 3: How to get node-handle.

Now we have access to the parent node-handle from the PID side. Next things to set up are publishers and subscribers.

```
desPosition = priv_nh.subscribe
("desired_position", 1, &LinearTracker::desPositionHandle,
this, ros::TransportHints().tcpNoDelay());
```

Listing 4: Creating a subscriber.

In this our node will create and subscribe to topic named “desired\_position” but this topic can be found `/uav#/linear_tracker/desired_position`, where you can give necessary commands to our tracker, where `#` represents the ID of UAV. Setting is loaded as a last part of the initialized method.

The activation method can be called after initialization at any time. The activation method activates the tracker. The activation method give the trucker information about the last command from the predecessor to ensure smooth transition from one tracker to another.

The deactivate method we disable the tracker.

The update method is called in a cycle after initialization method is called. The update method give back command to the controller and takes estimated state of UAV. The communication is not realized by topics. Topic communication is slow for our proposes. We used the update method (non-copy pointers) communication. The non-copy communication can be describe as a calling a function in C++.

## ■ 2 Implementation of the linear tracker

In this section, we will mathematically derive and implement the linear tracker.

### ■ 2.1 Linear tracker mathematically

Let us have a point in space  $\mathbf{U}$ , which is a position of our UAV, and second point  $\mathbf{S}$ , our setpoint position.

The direction from to s is defined algebraically as

$$\mathbf{D} = \mathbf{S} - \mathbf{U}, \quad (100)$$

where  $\mathbf{D}$  is a vector, pointing in a direction of the setpoint. We have normalized the direction vector as

$$\mathbf{D}_{\text{ir}} = \frac{\mathbf{D}}{\|\mathbf{D}\|}, \quad (101)$$

where symbol  $\|\mathbf{D}\|$  stand for Euclidean norm. The  $\mathbf{D}_{\text{ir}}$  is the desired direction of flight. Now we construct linear reference generator as

$$\mathbf{C} = \mathbf{U} + \mathbf{D}_{\text{ir}}s\Delta t, \quad (102)$$

where  $\mathbf{C}$  is our output command and s stands for speed and  $\Delta t$  for time elapsed in a loop.

### ■ 2.2 Linear tracker in ROS

In ROS the linear tracker is implemented using C++ as shown in listing 5.

```

// Get information about where to fly and where uav is
double vectorLength = sqrt((desX - x) * (desX - x) +
(desY - y) * (desY - y) + (desZ - z) * (desZ - z));
// set positions from odom
if (vectorLength > 5) {
    position_output.position.x = x +
    ((desX - x) / vectorLength) * speed;
    position_output.position.y = y +
    ((desY - y) / vectorLength) * speed;
    position_output.position.z = z +
    ((desZ - z) / vectorLength) * speed;
} else {
    position_output.position.x = desX;
    position_output.position.y = desY;
    position_output.position.z = desZ;
}

// set yaw based on current odom
if (useYaw) {
    position_output.yaw = desYaw;
    position_output.yaw_dot = 0;
}

//Send command to the controller

```

Listing 5: Linear tracker core.

As you can see in listing 5 the core of linear tracker. Furthermore, the use of yaw is optional and can be enabled on demand.

## ■ 2.3 Structural point of view on linear trucker

The linear trucker resembles a library. In order to make plugin we have to change package.xml. Next change is to call macro in source file (lineartracker.ccp). We use plugin for extending the PID controller [16].

### ■ 2.3.1 How to make a plugin

1. Registering the plugin
2. Make the plugin description file linear\_trackers.xml
3. Register the plugin with ROS package System

To satisfy the first point we need to add to our code pluginlib macro library and use the macro, as the linear tracker. Registering as a plugin can be seen in listing 6.

```

#include <pluginlib/class_list_macros.h>
PLUGINLIB_EXPORT_CLASS(LinearTracker, trackers_manager::Tracker)

```

Listing 6: Plugin registration in code.

Macro will function only when we give it the class that will be made as a plugin and the interface to communicate with ROS packages. Next, task we create the linear\_trackers.xml, which carries out description of the plugin. Example can be found in 7

```

<library path="lib/liblinear_tracker">
  <class name="linear_tracker/LinearTracker" type="LinearTracker"
    base_class_type="trackers_manager::Tracker">
    <description>This is Linear tracker.</description>
  </class>
</library>

```

Listing 7: Plugin decription file.

The file linear\_trackers.xml gives rest of the ROS information about the location of the plugin and which interface uses. The file linear\_tracker.xml gives information to user about the plugin. This file has to be in package folder. Acording to [16] “We need this file in addition to the code macro to allow the ROS system to automatically discover, load, and reason about plugins.” Before last step we have to make our plugin compatible with catkin build system. The makefile is updated according to listings 8.

```

add_library(linear_tracker src/linear_tracker.cpp)
install(FILES linear_trackers.xml DESTINATION
${CATKIN_PACKAGE_SHARE_DESTINATION})

```

Listing 8: Plugin makefile changes.

The last step is to change plugin.xml to enable the plugin. In order for pluginlib to query all available plugins on a system across all ROS packages, each package must explicitly specify the plugins it exports and which package libraries contain those plugins. A plugin provider must point to its plugin description file in its package.xml inside the export tag block [16].

```

<export>
  <trackers_manager plugin="${prefix}/linear_trackers.xml" />
</export>

```

Listing 9: Package.xml adding export tag.

Further more for the above export command to work properly, the providing package must depend directly on the package containing the plugin interface. That means we need to add two more lines to our package.xml file as in listing 10.

```

<build_depend>trackers_manager</build_depend>
<run_depend>trackers_manager</run_depend>

```

Listing 10: Package.xml adding build dependencies.

After all these steps, we now have fully functional plugin, which can be tested in the Gazebo simulator.

## Part 6

# Designing of a modular guidance system

In this section, we will focus on modular reference generator system. The system responsible for UAV navigation is based on the PID controller and integrates MPC tracker and Linear tracker.

First prototype of the modular system can be seen in figure 17. Connection from **Services** block is implemented either by ROS service or ROS topic.

Connections in **TRACKER\_INTERFACE** are implemented by *update*.

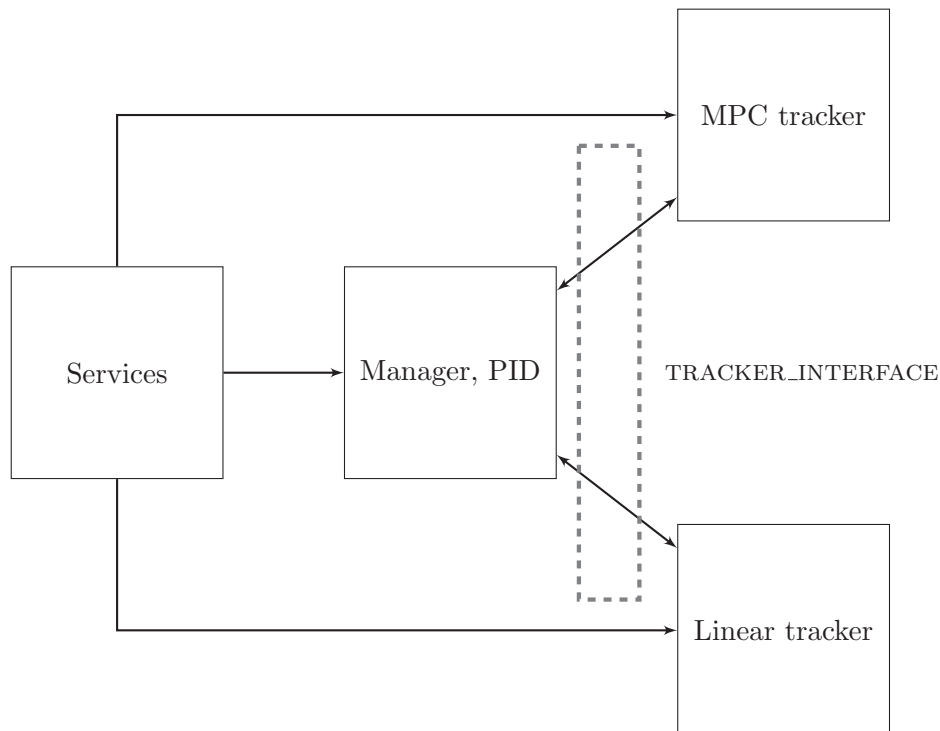


Figure 17: Block diagram of the modular system for connecting various control reference generators.

The switching between MPC and Linear tracker can be done using dynamic reconfigure. Dynamic reconfigure provides a standard way to expose a subset of a node's parameters to reconfiguration. Client programs, e.g., GUIs, can query the node for the set of reconfigurable parameters, including their names, types, and [17]. This tool is highly useful for real-time tuning a controller or setting its operation point. Figure 17, shows services that are connected to all nodes and the ability to switch controller is not possible because it is embedded in **manager** node. Experiment verification of the system can be found in section 6 subsection 4.

## ■ 1 Towards complete modularity

Based on the prototype system, we propose even more modular system which can be seen in figure 18.

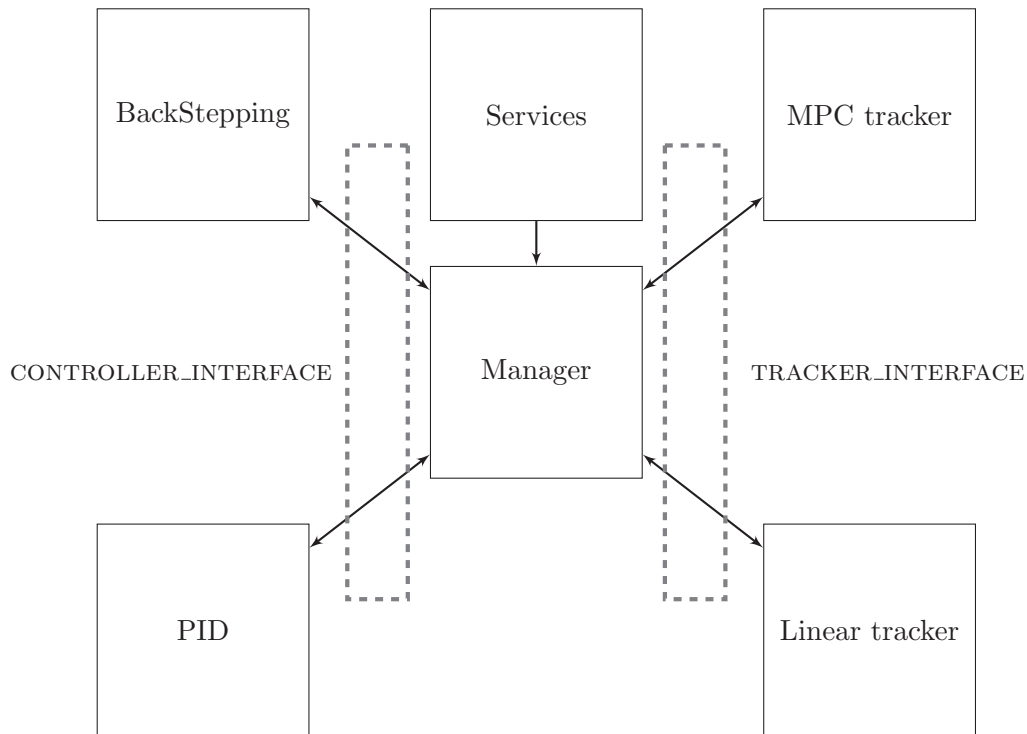


Figure 18: Block diagram of modular system for switching various control reference generators, and feedback controller.

Firstly, the service from figure 18 can be seen as a bundle of ROS services and ROS topics controlling the behaviour of the system. For example, we can tell to linear tracker to set desired position by publishing on special topic or we can call service on MPC tracker to goTo and set desired position again. Also we can directly communicate with the PID directly using dynamic reconfigure to select tracker or set PID constants in runtime. Another main block in figure 18 is the **Manager**. The **manager** block handles all services and enables switching of trackers and controllers. This state is only cosmetic but when we have another controller (**BackStepping**), we can switch them.

## Part 7

# Simulation in Gazebo Simulator

## 1 Description of the simulated scenario

In this section, we will test our regulator in Gazebo simulator, and we will test properties in comparison with the old controller, which is currently used in the MRS group. For testing in simulator, we used the same trajectory as in real flight as you can see in figure 23. This trajectory is followed after a series of relative go-to commands to MPC tracker. The series of goTo commands is composed from 8-meters steps and can be seen in figure 19

## 2 Simulation

The simulation was done in Gazebo simulator. In MRS group it is said that if a software is flown in this simulator, then it has chance that the same phenomenon will be seen in real life. That is shown in section 8 subsection 4 where we present result from a real flight. We conducted tests in Gazebo simulator for both of regulators, the PID and the originally used and proprietary Back-stepping regulator that is currently used by MRS group. Both regulators have the same trajectory to flight and results are shown in figures 19, 20, 21.

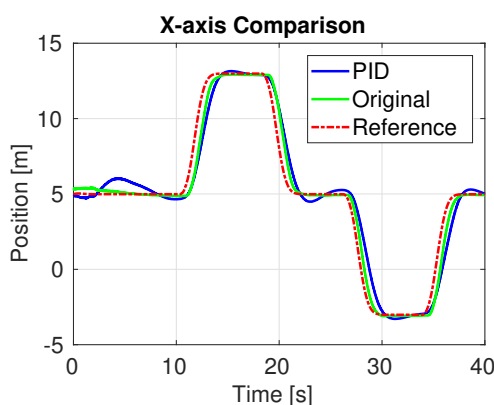
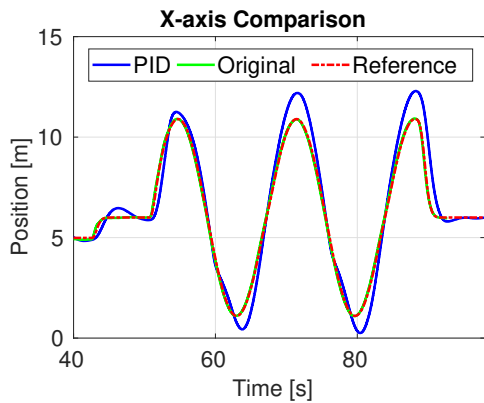


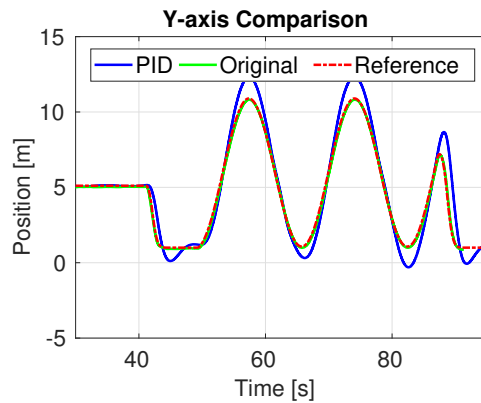
Figure 19: Step responses comparison between the PID and the original controllers in horizontal axis.

As you can see in figure 19 we compared both regulators on the same trajectory. We can see that the old regulator tracks reference perfectly, but our new regulator has a difficulty to track these changes despite the fact that the regulator is given a smooth reference produced by the tracker, which satisfies dynamical constrains of the UAV. The biggest overshoot is 0.6 m, which is acceptable, for autonomous flight with the vehicle localized using GPS. Precise control was not the primary criterion. The regulator was tuned for smooth flight and to minimize chances of high frequency oscillations, which may cause a crash of the real UAV.





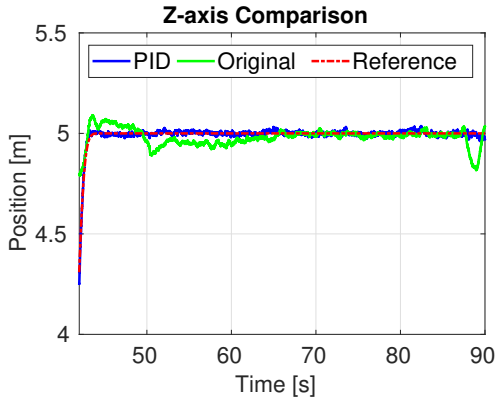
(a) Comparison of the PID and original controllers in horizontal (x) axis when following sine reference.



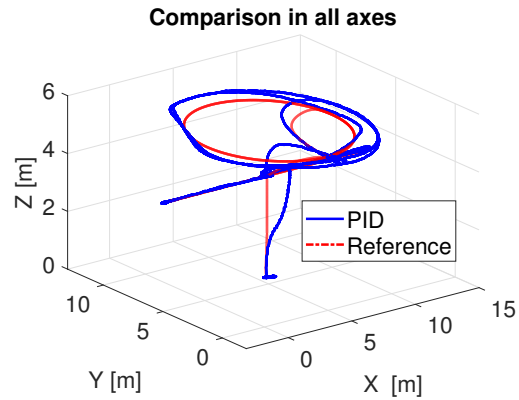
(b) Comparison of the PID and original controllers in horizontal (y) axis when following sine reference.

Figure 20: Comparison of the PID and original controllers in horizontal plane when following sine reference. (a) shows comparison in x-axis, (b) shows comparison in y-axis.

In this figure 20 we see the circling part of our preplanned trajectory.



(a) Comparison of the PID and original controllers in vertical (z) axis.

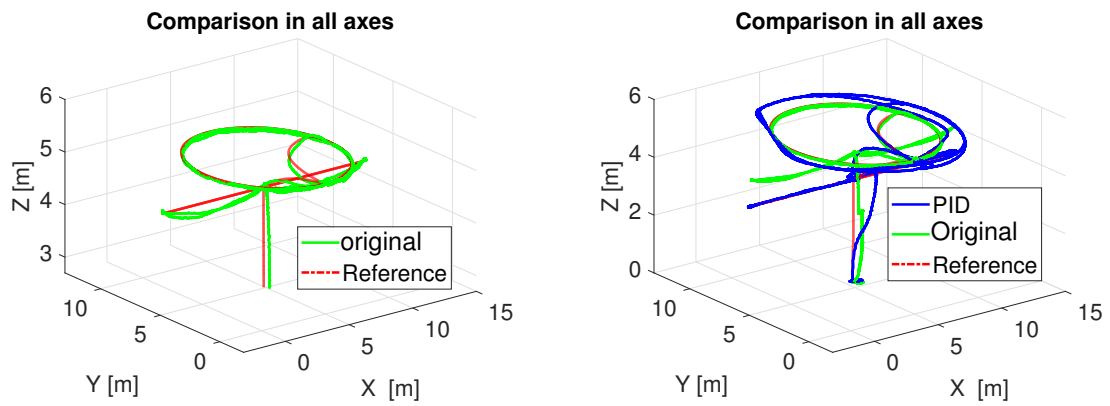


(b) Performance in 3D space of the PID controller.

Figure 21: Comparison of both controllers in z-axis and overall view. (a) shows comparison on the PID and original controllers, (b) shows overall view.

Figure 21 (a) shown the heigh control, which the new regulator can handle better than the old backstepping regulator.

Figure 21 (b) shown flight in 3D space, where blue line represents flight path of UAV and the red one reference. This figure gives reader full picture of the flight that was performed in simulation.



(a) Overall view of performance original controller in 3D space. (b) Comparison of both controllers in 3D space.

Figure 22: Comparison of both controllers in 3D space. (a) shows performance of the original controller, (b) shows comparison of both controllers.

Figure 22 (a) shown original backstepping regulator where green line represents flight path of UAV and the red one reference. Figure 22 (b) shown the PID controller where green line represents flight path of UAV when backstepping is used and the red one reference. The blue one represents the UAV position when controlled by PID. Figure 22 (b) shown comparison in 3D space where green line represents flight path of UAV when backstepping is used and the red one reference also the blue one represents UAV position when the PID was used. Again, full view to both regulators in one plot.

## Part 8

# Real flight comparison

### 1 Real flight measures

Because the UAV is valuable, we have to take measures with the start sequence. In order to perform a take off safely we implemented two new services to enable thrust  $T$  and the control in x and y axes ( $\varphi$  and  $\theta$ ) separately. When the UAV is on the ground, the safest way to take off and after the UAV is in the air, the regulating in x and y axes is enabled.

### 2 The first real flight

In the first flight, the PID was fine tuned for the particular UAV hardware. We used the dynamic reconfigure [17] for tuning our controller. The tuning was performed on step changes. The step changes were done by operator. The operator called `goToRelative` service provided by MPC tracker, which allows defining new setpoint based on the current position of the UAV.

### 3 Path for comparison

In order to compare our PID whit original controller, we made a path that both regulators have to follow. For that, we use Matlab to generate the path. This path is a circle that is loaded using a dedicated trajectory loader package. We had to make the path with dynamic parameters in mind. First, is acceleration that is set in the configuration file for MPC tracker to  $1 \text{ m/s}^2$  and the maximal speed that is also set in MPC tracker to  $2 \text{ m/s}$ . The path was made to meet requirements. Figure 13 shows the circle trajectory. We tested circle of  $5 \text{ m}$  radius with design velocities of  $1 \text{ m/s}$ .

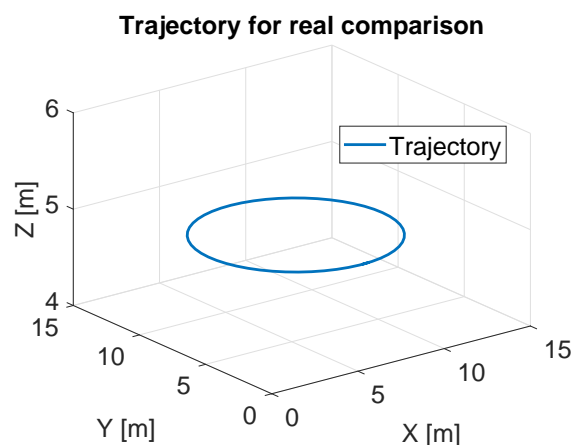
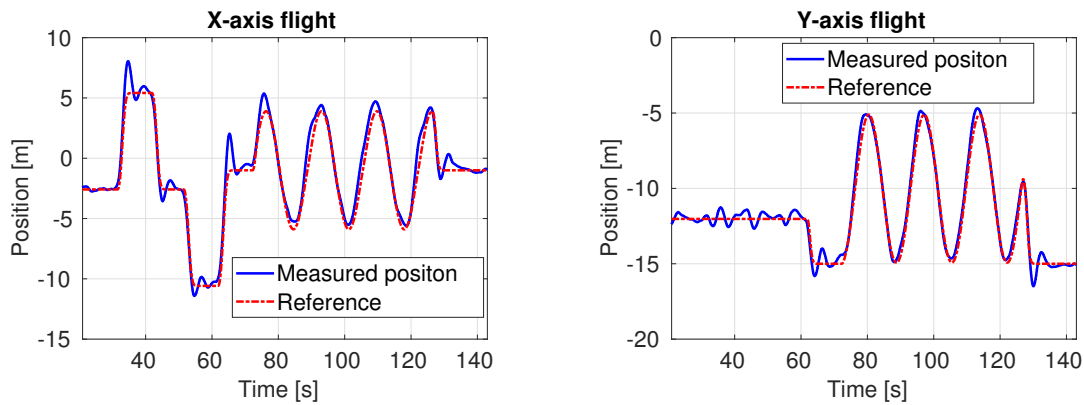


Figure 23: Trajectory has been generated in Matlab.

## 4 Control performance

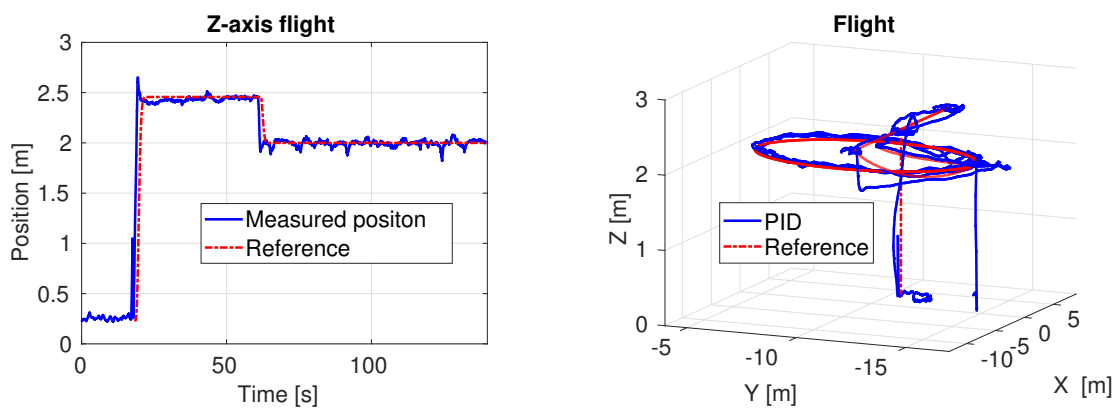
As you can see in figures 24 (a) and (b), the proposed PID regulator follows the reference sufficiently. In the next set of figures you can see a comparison with the old proprietary regulator.



(a) UAV following specific reference in horizontal (x) axis when controlled by the PID controller. (b) UAV following specific reference in horizontal (y) axis when controlled by the PID controller.

Figure 24: UAVs estimated position in horizontal plane when controlled by the PID controller. (a) shows following reference in x-axis, (b) shows following reference in y-axis.

The PID regulator follows the reference in both axes. The step responses can be seen in figure 24 (a). The responses in figure 24 (a) and b are sufficient in out door flight with GPS localization. The controller performed similarly even with steps as large as 50 m thanks to the tracker.

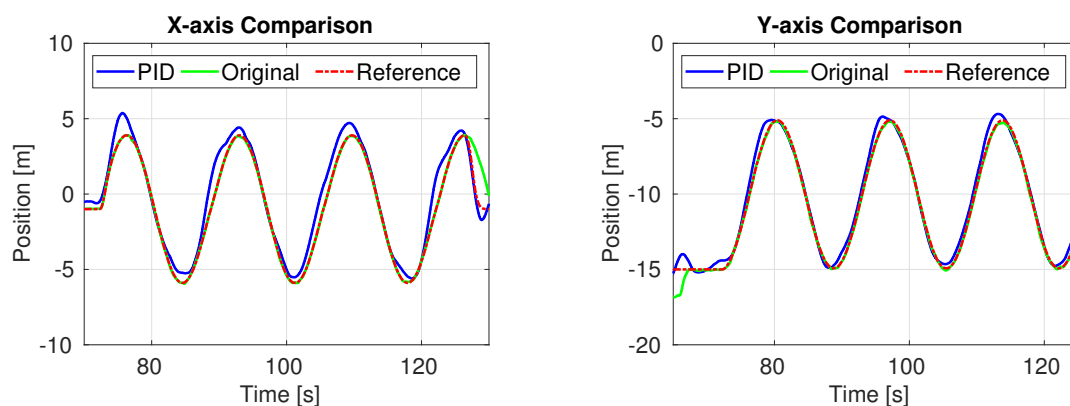


(a) Estimated position of the UAV in vertical (z) axis when controlled by the PID controller. (b) Flightpath in 3D space reference (red) estimated position (blue)

Figure 25: Estimated position of the UAV in vertical axis and overall view of the performance. (a) shows position of UAV in z-axis, (b) shows overall view of the flight.

Figure 25 (a) shows an initial step in height issued take off. Figure 25 (b) shows PID in control during a complex trajectory that consist of steps changes in position and the circle reference 1 m/s, describe in section 8 subsection 3. The PID performance is comparable to

old proprietary controller. Comparison between our new PID and old controller can be seen in figures 26, 27.



(a) Comparison of both controllers in horizontal (x) axis when following sine reference.

(b) Comparison of both controllers in horizontal (y) axis when following sine reference.

Figure 26: Comparison of both controllers in horizontal plane. (a) shows both controllers in x-axis, (b) shows both controllers in y-axis.

### Comparison in all axes

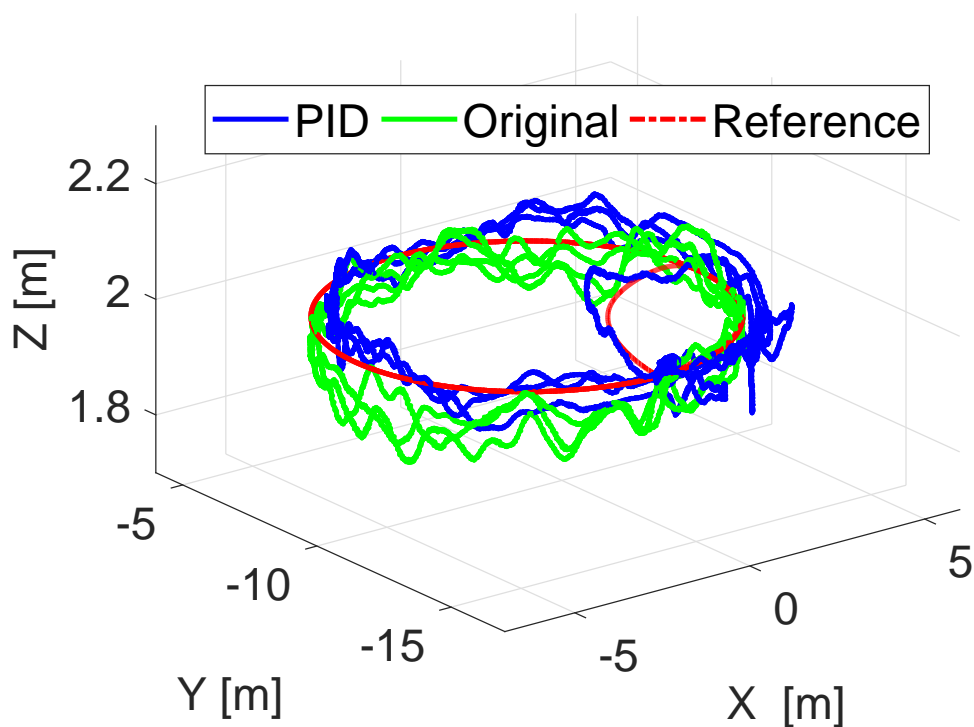


Figure 27: Comparison of both controllers in 3D space.

In figure 27 can be seen that old controller has different starting sequence the PID. Figure 28a shows UAV controlled by PID control system as well as figure 28b. Figures 28a, 28b

shows step from position in figure 28a to position in figure 28b. Both pictures are at hover mode before and after step change in reference.



(a) UAV 5 controlled by the PID controller at the beginning of trajectory




(b) UAV 5 controlled by the PID controller at the end of trajectory

Figure 28: Snapshots of testing of the system in real-world conditions. Video from the experiment can be found at <https://www.youtube.com/watch?v=tpn7tCahuIQ>.



Figure 29: Unmanned aerial vehicle when controlled by the developed control system.



## Part 9

# Conclusion

In this thesis, we have developed software solution that allows execution of the model position control of micro aerial vehicles. We designed and tested PID controller. The PID controller was implemented in C++. The implemented PID was integrated in to ROS pipeline. The dynamical model of the UAV has been derived and its parameters have been experimentally and numerically identified. The modular system was design and implemented with ability to connect various control references generators. We also implemented linear tracker generator. The linear tracker generator was tested in Gazebo simulator. We investigate backstepping control approach and simulated in Matlab Simulink. The backstepping was investigated and then simulated in Matlab Simulink. We compared the PID and backstepping in Matlab and conducted that follow reference as well as backstepping controller. The proposed system has been successfully implemented into the ROS pipeline. We conducted tests of PID in Gazebo simulator in order to confirm that the PID is able to fly in real world. We tested PID and fine tuned in real world. The PID sufficiently followed reference from tracker. We are now able to flight without the original controller. The entire assignment of this thesis has been fulfilled successfully. During the development of this thesis, several ideas emerged that specify our future work. The modular system will be overdo to proposed ideal state. We will separate main manage form controller. We will construct plugins controllers: PID, backstepping. And we will fine tuned system to usable state. We also add more tracker for better performed at take offs.



# References

## References

- [1] T. Báča, D. Hert, G. Loianno, M. Saska, and V. Kummer, “Model predictive trajectory tracking and collision avoidance for reliable outdoor deployment of unmanned aerial vehicles,” 2018.
- [2] P. Petráček, “Decentralized model of a swarm behavior boids in ros,” in *Bachelor’s Thesis*. CTU FEE, 2017.
- [3] P. Štibinger, “Localization of a radiation source by a formation of unmanned aerial vehicles,” in *Bachelor’s Thesis*. CTU FEE, 2017.
- [4] Y. Altman. (2018) Undocumented matlab. [Online]. Available: <https://undocumentedmatlab.com/> [Accessed: 2018-05-20]
- [5] G. F. Franklin, J. D. Powell, and M. L. Workman, *Digital control of dynamic systems*. Addison-wesley Menlo Park, CA, 1998, vol. 3.
- [6] P. V. Kokotovic, “The joy of feedback: nonlinear and adaptive,” *IEEE Control systems*, vol. 12, no. 3, pp. 7–17, 1992.
- [7] O. S. R. Foundation. (2018) Gazebo. [Online]. Available: <http://gazebosim.org/> [Accessed: 2018-05-12]
- [8] H. Lee, S. Kim, and T. Ryan, “Backstepping control,” in *Backstepping Control on SE(3) of a Micro Quadrotor for Stable Trajectory Tracking*, 2013.
- [9] H. talla Mohamed Nabil ElKholy, “Dynamic modeling and control of a quadrotor using linear and nonlinear approaches,” in *The degree of Master of Science in Robotics, Control and Smart Systems*, 2014, pp. 0–143.
- [10] C. A. Arellano-Muro, L. F. Luque-Vega, B. Castillo-Toledo, and A. G. Loukianov, “Backstepping control with sliding mode estimation for a hexacopter,” in *International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*, 2013.
- [11] H. Bouadi, M. Bouchoucha, and M. Tadjine, “Sliding mode control based on backstepping approach for an uav type-quadrotor,” *World Academy of Science, Engineering and Technology*, vol. 26, no. 5, pp. 22–27, 2007.
- [12] H. R. Pota, B. Ahmed, and M. Garratt, “Velocity control of a uav using backstepping control,” in *45th IEEE Conference on Decision and Control (CDC)*. IEEE, 2006, pp. 5894–5899.

- [13] G. V. Raffo, M. G. Ortega, and F. R. Rubio, “Backstepping/nonlinear h control for path tracking of a quadrotor unmanned aerial vehicle,” in *American Control Conference (ACC)*. IEEE, 2008, pp. 3356–3361.
- [14] T. Madani and A. Benallegue, “Sliding mode observer and backstepping control for a quadrotor unmanned aerial vehicles,” in *American Control Conference (ACC)*. IEEE, 2007, pp. 5887–5892.
- [15] M. Krstic, I. Kanellakopoulos, and P. Kokotovic, “Nonlinear design of adaptive controllers for linear systems,” in *Nonlinear and Adaptive Control Design*. John Wiley & Sons, Inc., 1994.
- [16] M. Arguedas. (2015) Pluginlib. [Online]. Available: <http://wiki.ros.org/pluginlib> [Accessed: 2018-05-19]
- [17] B. Gassend. (2015) Dynamic reconfigure. [Online]. Available: [http://wiki.ros.org/dynamic\\_reconfigure](http://wiki.ros.org/dynamic_reconfigure) [Accessed: 2018-05-19]

## CD content

Table 8 lists names of all root directories on CD together with their content.

Directory name	Description
/main.pdf	Bachelor thesis in pdf format.
/matlab	All Matlab files for simulation and making plots and measured data form gazebo simulation and real flight .
/PIDBackStepping	Location of all nodes in ROS.
/texfiles	Location of all texfiles for this thesis.
/source	Main node only source file pidcontroll.cpp and linear_tracker.cpp.

Table 8: CD content.