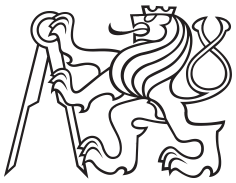
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Optimal Trajectory Planning for a Quadrotor UAV for Autonomous Drone Race

Bc. František Nekovář

Supervisor: Ing. Milan Rollo, Ph.D.
Field of study: Cybernetics and Robotics
Subfield: Systems and Control
January 2019

Acknowledgements

I would like to express thanks to the CTU for beautiful 5 years of study, and to my supervisor for giving me a chance to work on this topic.

Declaration

I declare that I have produced the submitted work independently and that I have provided all the information sources used in accordance with the Methodological Guideline on Ethical Principles in the Preparation of University Graduate theses.

In Prague, 7. January 2019

Abstract

In this work, existing state of quadcopter drone racing competitions is surveyed, both human-driven and autonomous ones, and then several optimal trajectory planning algorhitms are compared to use in constrains of these competitions. A mathematical model of a quadcopter is created, with proposed nonlinear closed loop attitude controler and open loop trajectory following controler. This system is then tested on trajectories generated by the proposed optimal planning algorhitm in set of several different competitions. Implementation of the control and trajectory planning for ROS is shown, implementation is then simulated in SITL using ROS-Gazebo.

Keywords: trajectory planning, UAV, MAV, quadcopter, drone racing, ROS, Gazebo, Matlab, control

Supervisor: Ing. Milan Rollo, Ph.D.
Artificial Intelligence Center - FEL
Karlovo nám. 13,
Praha 2

Abstrakt

Práce se zabývá průzkumem existujících soutěží závodů dronů, vč. lidmi řízených a autonomních. Je porovnáno několik algoritmů pro plánování trajektorie ve vhodnosti pro tyto závody. Je vytvořen matematický moter quadrokoptéry, vč. nelineárního closed loop regulátoru pro řízení náklonů a open loop regulátoru pro sledování trajektorie. Tento systém je testován na trajektoriích vygenerovaných navrhovaným plánovacím algoritmem v rámci různých soutěží. Je ukázána implementace řízení a plánování trajektorie pro ROS, implementace je poté simulována SITL pomocí ROS-Gazebo.

Klíčová slova: plánování trajektorie, UAV, MAV, čtyř-rotorová helikoptéra, závody dronů, ROS, Gazebo, Matlab, řízení

Překlad názvu: Plánování optimální trajektorie pro čtyř-rotorovou helikoptéru pro závody dronů

Contents

Figures

Tables

Chapter 1

Study of drone racing and rules of existing competitions

1.1 Introduction

Drone racing, an e-sport which originated from Australia around 2014, has been steadily gaining on popularity in recent years and resulted in establishment of the Drone Racing League, MultiGP and plenty more lesser leagues, which aim to compete for audience with other "high velocity" sport leagues, such as Formula F1. This was made possible by research and development of compact battery technologies in recent years, which made the devices cheaper and therefore available to general public. During these human controlled races, pilots make use of head mounted displays, which provide them with point of view from drone on-board camera. Due to their both speed and agility, small quadcopters are used for the racing, which even though being limited in battery life, reach speeds exceeding 100 kph during the races. Races are typically taken in indoor environments on previously known courses. The goal is to navigate a course, which we can assume as a set of gates, through which the drone has to pass, and sometimes also a set of obstacles, which the drone has to avoid, in shortest time possible. So far, competitive drone racing has been human domain, there is however great interest in developing algorithms and flight AIs which would surpass human pilots using the same visual data, just because of tremendous potential for application in all aspects of our life. In recent years, we have seen huge steps in autonomous vehicle development, as self-driving cars or military drones. Having reliable means to control UAVs, for example in urban environments or disaster sites would be invaluable. And because drone racing places great stresses on both speed

and reliability (a drone which crashes during the race does not score very well), it is an interesting benchmark for development of said technologies. In late 2018, the DRL has announced host an Autonomous Drone Competition in cooperation with Nvidia and Lockheed Martin during the 2019 season. This only supports the assumption that autonomous drone racing is gaining popularity not just for hobbyists but also mainstream audience, since races are now commonly present on sports TV broadcasts where other robot competitions already have a tradition. Aim of this work is to study the drone racing from engineering point of view, formulate it in mathematical terms and to propose and implement trajectory planning and control algorhitms which would be suitable for autonomous drone racing.

1.2 Autonomous challenges

1.2.1 IROS Autonomous Drone Racing Competition 2016

The first official public race competition for autonomous drones was held during IROS 2016 conference. Indoor racing track contained several different scenarios, from straight path for high speed flight to upward spiral or sharp turns. To make the identification of gates easier, all of the 26 gates were assigned printed QR codes to them. Two kinds of gates were present, an open gate which only limited drone from 3 sides and a rectangular closed gate. Both gates were of side of 1.2 m. The racing track also contained one dynamic moving gate. There was additionaly a size limit for the drone, being a cube with side of 1 m. It was allowed and encouraged to use any onboard sensor data which make no physical contact with surroundings, the drone however had to be fully autonomous. CAD model of the track, subjected to some minor differences, was available for download and testing before the competition took place. It should be noted that only 3 teams managed to compete in the challenge, none of which have finished the whole track.

1.2.2 IROS 2017

Second installment of IROS Autonomous Drone Racing competition had similar rules as the previous, with some notable differences in the track design. The track contained three kind of gates: closed gates, same as in previous year, were squares with side of 1.4 m, open gates were replaced by tree like gates for straight flight and rectangular dynamic section with continuously

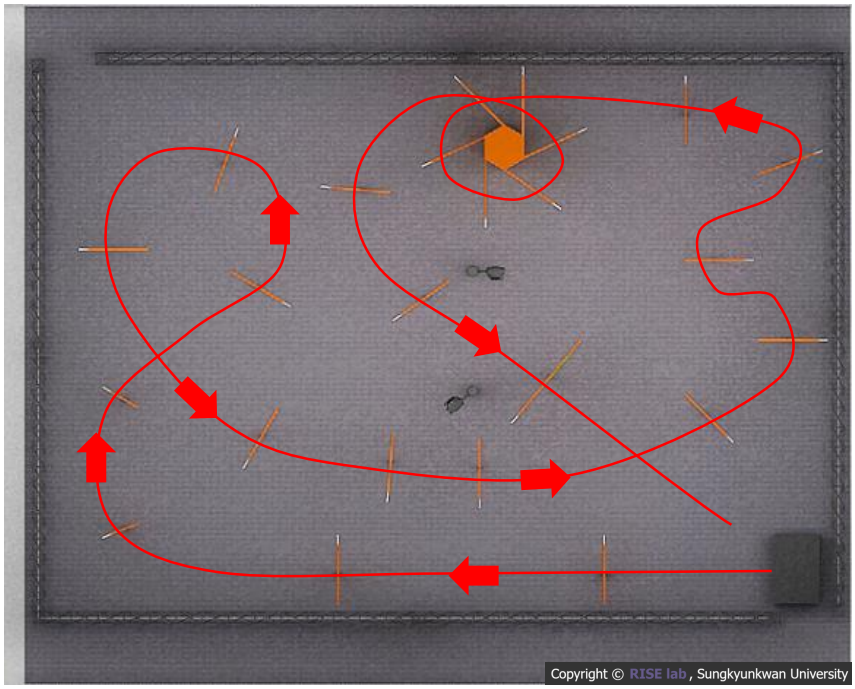


Figure 1.1: IROS 2016 track

rotating barrier. Although the competing teams were more successful this time, the issues with the track itself called for its redesign. Most notably, the track was still too cluttered with gates for successful recognition, and the recognition of gates itself was a significant problem.

1.2.3 IROS 2018

The racing track saw significant changes. It was notably shorter, with lesser amount of gates, but still featuring the dynamic section with rotating obstacle and furthemore cubic "jungle gate". Implication is, as cited from ADR website, that: „autonomous drone navigation is possible if the environmental information is perfectly known and optic flow works properly. However, such cases are not realistic in many useful practical applications such as first responding to disaster sites or autonomous drone taxi in cities.". Gates were, after discussing the issue with participating teams, modified with clean white borderline to make recognition easier. Their position was also randomized, so there would be more emphasis on recognition and flight planning. This is a significant shift from previous years, where tracks resembled usual racing, while this challenge was more focused on environment navigation.

1.3 Major drone racing leagues

1.3.1 MultiGP

First major drone racing league, founded in 2015. It is the usual entry point league for beginners, since it operates worldwide using local chapters and enables pilots to use their own equipment during the races. It provides track gate and drone hardware specifications in multiple size categories, to which custom drones can be designed. Tracks can be created freely, although some standard specifications are available like "Universal Time Trial" tracks, which allow pilots to compete on international leaderboards. Racing tracks of MultiGP are generally simple, mostly on 2D plane but some variations in height of gates might appear.

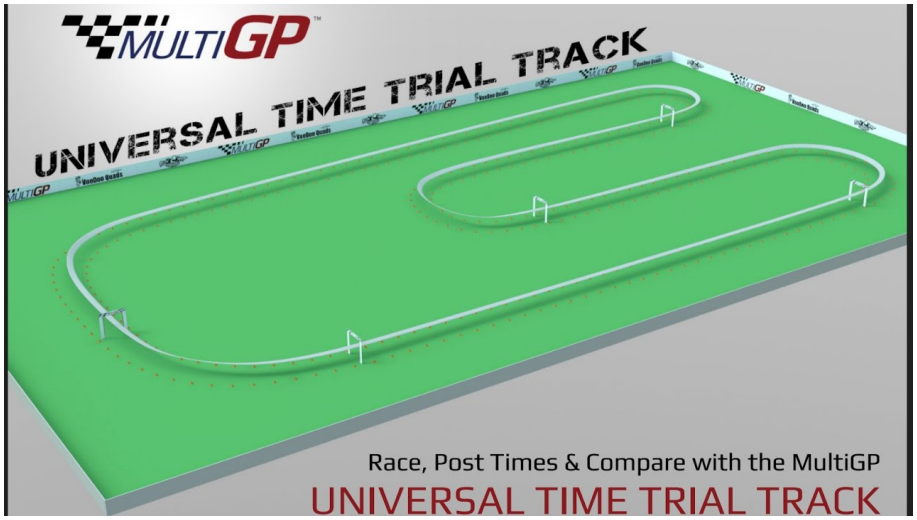


Figure 1.2: MultiGP track example

1.3.2 Drone Racing League

DRL was also founded in early 2015, but is in comparison to MultiGP, is purely e-sport oriented. It provides its contracted pilots with custom built drones and equipment, which are not for sale on the market. Aspiring pilots can qualify through DRL Simulator, a PC game developed by the DRL to broaden its audience. DRL has, soon after it was launched, started broadcasting on international sports channels. Tracks in DRL are not generally specified, they can be laid out with anything including poles, trees or hula hoops and are of complicated 3D character. Racing however is mostly held indoors.

1.3.3 DR1 Racing

Another e-sport league, DR1 races in comparison typically take place in outdoor environments. Tracks are therefore very large, featuring triangular gates. While using a standardized drone for official races, it introduces its own class of racing drones, which doesn't limit size or performance.

1.3.4 Drone Championship League

The current biggest live broadcasted drone racing league. In comparison to other leagues, it provides almost complete freedom of drone design, only limiting weight. Tracks are also outdoor, however gate specification varies.

1.4 Hardware requirements and limitations in drone racing

Most obvious limit on the drone platform is its physical size. If we take IROS competitions for example, the drone has to be able to freely pass through gates without collision, but there should be some wiggle room kept in case of altitude or attitude disturbances. IROS specifies maximum of 1m on each axis, which is by a rule of thumb maximum size which can reliably pass through the gate safely. The racing leagues are sometimes stricter than that, requiring machine to be even small and sometimes non-restrictive at all. The DCL featuring only limits weight between 0.7 to $1kg$. Racing drones are usually limited in performance of motors, power to weight ratio is then dependent on making the drone as light as possible. The battery life of racing drones is very short then, but racing courses mostly require below a minute to finish so that is not of an issue. The biggest difference between racing drone and autonomous racing drone is the on board equipment. IMU and a gyroscope are usually present in racing drones, since the drone pilot doesn't control the motors directly, but rather provides reference rotation rates. Outdoor flight localization can be improved with GPS, which is a low-cost solution but might not be enough assuming the high speeds racing drones typically allow. With the POV camera already on board of every racing drone, some sort of simple SLAM (orbslam2 for example) to enhance the localization could be possible. The supplied cameras are however of low resolution, to reduce the latency of transmission to pilots FPV goggles. Drone should therefore be equipped with

at least a HD capable camera with additional hardware. Because of the fully autonomous operation, significant computational performance is required, in form of some embedded solution connected to the autopilot hardware. For example, IROS 2016 winning drone mounted Nvidia TX2 module on board, running attitude Kalman Filter and a gate detection for optical flow path panning. The IROS 2018 winning team mounted Intel Up board on their drone. It is to be noted that both teams used neural networks for gate localization [JHSS18]. There are many out of the box solutions on the market for drone racing, but autonomous racing requires a custom build depending on specific race and computational load required by control algorhitms. There are many ROS capable autopilots on the market today, for example Pixhawk or NAVIO.

1.5 Quadcopter

Quadcopters or quadrotors are VTOL aircraft equiped with four propellers, each spun by a motor at variable speed. Propeller rotation is opposite in pairs, to equalize generated torque and rotor blades are typically fixed (although some variable pitch models exist on the market). Rotor blades are optimized to give best performance in only one direction of rotation, which makes reverse spin mode inefficient although not impossible. Quadcopters today find many uses, due to their both outstanding stability and maneuverability. Since every rotor is controlled independently, there is a wide variety of possible maneuvers which would be impossible for conventional helicopter, let alone a fixed wing aircraft. This makes quadcopters ideal for operation in dense environments, assuming a skilled pilot is in control or a suitable control algorhitms are provided, which I will adress later.

1.5.1 Brief history

First experiments with manned quadcopters began as early as in 1907, and continued for several years, but quadcopters were surprassed by helicopters, since they were easier to pilot. A helicopter naturally stabilizes itself in hover while a quadcopter does not. Large clumsy quadcopters were therefore put aside, and as technology advanced, small remote controlled quadcopters appeared. With the aid of electronic stabilization and miniaturalization of electronics, quadcopters came back into favor during first decade of 21st century and are now subject of innovation in many fields.

Figure 1.3: QAV-R FPV Racing Quadcopter

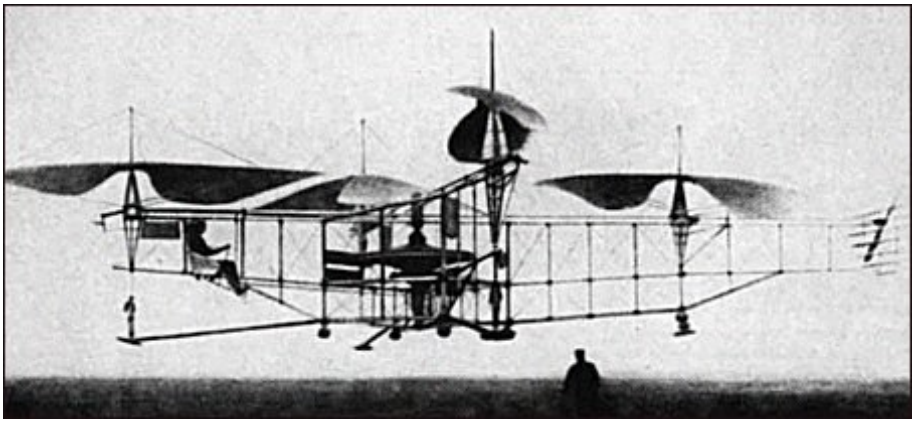


Figure 1.4: Oemichen 2 quadcopter from 1920

1.5.2 Mathematical model

Formulation of a quadcopter in mathematical terms is fairly straightforward. From kinematics standpoint, it is a 6 degrees of freedom underactuated nonlinear system, which is controlled by 4 inputs, which are angular velocities of motors. Since hover state approximation is not practical in drone racing, we have to account for nonlinear dynamics. The system is further complicated by moments of inertia of both frame and rotors, and there is some drag involved during high speeds, which is however difficult to model since precise aerodynamics of frame and rotors are unknown. The motor dynamics are neglected in the general moment, but are included with specific optimization

methods. We follow with definition of the nonlinear system:

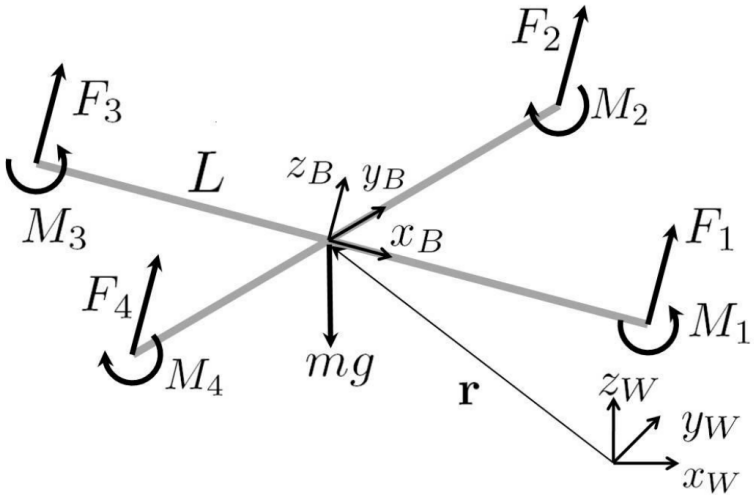


Figure 1.5: Forces and moments

$$\begin{aligned}
\dot{\mathbf{p}} &= \mathbf{v} \\
\dot{\mathbf{v}} &= \frac{F_t}{m} R(\mathbf{\Phi})\mathbf{z}_W - g\mathbf{z}_W \\
\dot{\mathbf{\Phi}} &= J(\mathbf{\Phi})\omega \\
\dot{\omega} &= I^{-1}(-\omega \times I\omega + \mathbf{M})
\end{aligned} \tag{1.1}$$

where $\mathbf{p} = [x, y, z]^T$ denotes a position vector of the centre of mass, $\mathbf{v} = [v_1, v_2, v_3]^T$ a velocity vector of the centre of mass, g denotes a gravitational constant, m denotes a mass of quadcopter, F_t denotes a total force generated by propellers, $\mathbf{\Phi} = [\theta, \varphi, \psi]^T$ denotes orientation in Euler angles - roll, pitch and yaw angles respectively, and $\omega = [p, q, r]^T$ denote angular rotation rates in the body frame, and $\mathbf{M} = [M_x, M_y, M_z]^T$ are the moments of force (or torques) about respective body axes.

Let us further define matrices:

$$R(\mathbf{\Phi}) = \begin{bmatrix} \cos\varphi\cos\psi & \sin\theta\sin\varphi\cos\psi - \cos\theta\sin\psi & \cos\theta\sin\varphi\cos\psi + \sin\theta\sin\psi \\ \cos\varphi\sin\psi & \sin\theta\sin\varphi\sin\psi + \cos\theta\cos\psi & \cos\theta\sin\varphi\sin\psi - \sin\theta\cos\psi \\ -\sin\varphi & \sin\theta\cos\varphi & \cos\theta\cos\varphi \end{bmatrix} \tag{1.2}$$

as a rotational matrix from body to world frame, then:

$$J(\mathbf{\Phi}) = \begin{bmatrix} 1 & \sin\theta\tan\varphi & \cos\theta\tan\varphi \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta\sec\varphi & \cos\theta\sec\varphi \end{bmatrix} \tag{1.3}$$

as the mapping from body frame $\dot{\omega}$ to world frame $\dot{\mathbf{\Phi}}$, and:

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{1.4}$$

as an inertial matrix along three main axes, where components are moments of inertia.

Not only is the system underactuated, but it is limited by other factors, most notably the inputs. Since the rotor output thrusts are limited by the maximum velocities of motors, the maximum drone motion and attitude dynamics are also limited. Taking these into account and assuming a suitable angular velocity controller is present to compensate for inertial forces and tracking of the angular velocity references, we can simplify the quaddcopter dynamics into a vector-thrust model:

$$\begin{aligned} \dot{\mathbf{p}} &= \mathbf{v} \\ \dot{\mathbf{v}} &= -g\mathbf{z}_W + \frac{F_t}{m} R(\mathbf{\Phi})\mathbf{z}_W \\ \dot{\mathbf{\Phi}} &= J(\Phi)\omega \end{aligned} \tag{1.5}$$

This simplified model becomes is used in some optimization approaches, since it compromises between system nonlinear dynamics, which happen to be unstable, and the load put on optimization algorhitms.

1.5.3 Simulation model

The mathematical model of quadcopter in form of differential equations will be used in optimization process and simple validation of controller, for software in the loop simulation however we need a different kind of model. For the purpose of this work, I have chosen to use a simulation model of AscTec Hummingbird Drone. The reason is that this model has already been used for research in drone control [WBH+11] and aerodynamics [FBAS16] and serves as a research platform for aggressive flight research, so the precise values of dynamic, aerodynamics and even motor parametres are known. Measurement of those would require extensive experimentation with a real model [CFCH14] and is out of scope of this work. Therefore, I perform the optimization on general mathematical model of quadcopter dynamics (where only mass, inertia and constrains on the states matter) and then evaluate performance of

trajectory tracking controller, which should reliably compensate for missing dynamics.

Figure 1.6: AscTec Hummingbird drone

Another significant difference of simulation model is presence of simulated sensors, most notably simulated IMU and position estimation. The motors of the model are modeled by first order system with the time constant τ_{DC}. The most important model parameters used both during optimization and during simulation are taken from [WBH+11], and are:

Parameter	Symbol	Value	Unit
Mass	m	0.68	kg
Inertia XX	I_{xx}	0.007	kgm^2
Inertia YY	I_{yy}	0.007	kgm^2
Inertia ZZ	I_{zz}	0.012	kgm^2
Gravitational acceleration	g	9.81	ms^{-2}
Rotor velocity squared to thrust ratio	k_F	5.710^{-8}	$N\,rpm^{-2}$
Rotor velocity squared to moment ratio	k_M	9.1210^{-10}	$N\,rpm^{-2}m$
Rotor arm length	L	0.17	m
Rotor drag constant	$-$	$8 \cdot 10^{-5}$	$-$
Motor time constant	τ_{DC}	0.125	s

The model itself is represented in an URDF format.

1.6 Mathematical formulation of drone racing

1.6.1 Race course

Depending on the competition, we can expect very specific constraints on the form of track. We start with a set of waypoints along the track, like gates, which should be passed through. Some gates however might be of logical nature, like passing around an obstacle in specified direction. Gates themselves too form an obstacle in trajectory formulation case, since they cannot be passed in arbitrary direction and to be collided with. The tracks are not usually limited in maneuvering space, but might feature obstacles, which have to be accounted for. It is practical to formulate a race course as a set of waypoints, however not ideal to fix a waypoint just to single point in space. Given that gates are usually very small compared to the size of the race course, it is a plausible assumption. This becomes versatile during polynomial formulation of trajectory. Another, approach, is to specify a waypoint as a constraint in path space. Exact formulation is dependent on the optimalization method. I presume the whole track profile and obstacle positions are available before the race. The race course can then be formulated as a constrained path space, on which some optimal trajectory has to be planned. The benchmark race course for this work is the MultiGP Universal Time Trial track.

The waypoints of the track are:

Number	X	Y
1	0	0
2	56	0
3	28	14
4	56	28
5	-14	14
6	0	0

The flight level is set at $1.5\,m$.

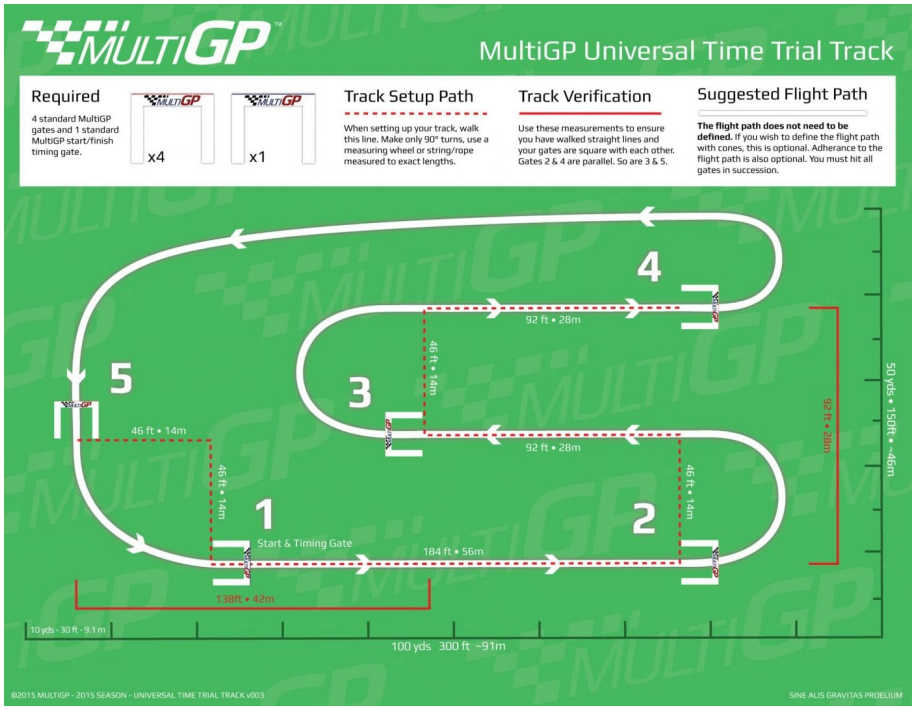


Figure 1.7: Specifications of race track

1.6.2 The race

Having established both vehicle and environment, we can formulate the drone race as a trajectory planning problem. The suitable trajectory $(\mathbf{x}(t), \mathbf{u}(t))$ (note that $\mathbf{x}$ here represents states of the system), which takes a drone along the race track, is constrained by the underactuated nature of quadcopter system ie. is subject to the quadcopter dynamics, should take the drone along the track in shortest time possible, therfore the cost $J = T$ is the total time taken. I will not consider presence of other vehicles on the track for the purpose of this work, since it obviously cannot predict their trajectories. The race can then be formulated as a generation of time optimal trajectory which satisfies the drone and environmental constraints:

$$\begin{aligned} &\min_{x,u,T} T \\ \text{subject to} \quad &\dot{\mathbf{x}} = f(\mathbf{x}(t), \mathbf{u}(t)) \\ &\mathbf{x}(0) = \mathbf{x_0} \\ &lbx_i <= \mathbf{x}_i <= ubx_i \\ &lbu_j <= \mathbf{u}_i <= ubu_j \\ &\mathbf{x}(T) \in X_T \end{aligned} \tag{1.6}$$

where $\mathbf{x_0}$ is the initial state, followed by constrains on trajectory states and inputs and a final constrained.

Chapter 2

Trajectory optimalization

There have been many approaches proposed for trajectory optimization of quadcopters in recent years, each suited for different application. In drone racing, we require the trajectory to pass through multiple waypoints in continuous fashion, which removes a wide array of point-to-point planning algorithms. We additionaly need to satisfy the system dynamics and constrains and environmental constrains, so some sort of feasible path search is in order. Some trajectory planning approaches also come with a quadcopter controller in mind. I will now follow with most practical trajectory generation algorhitms for drone racing, and compare their performance later.

2.1 Polynomial trajectory

Proposed in [MK11], generation of snap minimizing polynomials $\mathbf{p}$ to represent drone trajectories for aggressive maneuvering takes advantage of important propery of quadcopter nonlinear system, which is its differential flatness [vNM96]. In this approach, the states of the nonlinear system and its outputs can be written as an algebraic function of four *flat outputs*, which are $\sigma = [\sigma_1, \sigma_2, \sigma_3, \sigma_4] = [x, y, z, \psi]$ and finite number of their derivatives. Trajectory is then defined as a smooth curve $\sigma(t)$ being a function of time [MMR03]. System states can be easiy computed from σ, as is explained in [MK11]. Important fact to be noted however is than the computation does not depent on the derivatives of σ, which means it can be chosen arbitrarily. This becomes useful for example if we want to follow the trajectory with the camera facing forward.

2.1.1 Differential flatness

Lets first consider the equation of motion of the center of mass with use of body coordinates:

$$m\ddot{\mathbf{p}} = m\dot{\mathbf{v}} = m\mathbf{a} = F_t\mathbf{z}_B - mg\mathbf{y}_3 \tag{2.1}$$

where $\mathbf{a} = [a_1, a_2, a_3]^T$ denotes vector of acceleretion of the mass point in world frame. From there we can define first body axis vector $\mathbf{z}_B$ dependence on trajectory:

$$\mathbf{z}_B = R(\mathbf{\Phi})\mathbf{z}_W = \frac{m\mathbf{a} + mg\mathbf{z}_W}{F_t} = \frac{\mathbf{t}}{\|\mathbf{t}\|} \tag{2.2}$$

where $\mathbf{t} = m\mathbf{a} + mg\mathbf{z}_W = [\ddot{\sigma}_1, \ddot{\sigma}_2, \ddot{\sigma}_3 + g]^T$ denotes total acceleration. Also note that $m\|\mathbf{t}\| = F_t$. Given the yaw angle ψ, we can define a vector $\mathbf{xy}_B = [\cos\psi, \sin\psi, 0]$ on the world frame plane $[1, 1, 0]$. Then by vector multiplication, we receive remaining body frame vectors:

$$\mathbf{y}_B = \frac{\mathbf{z}_B \times x\mathbf{y}_B}{\|\mathbf{z}_B \times x\mathbf{y}_B\|}, \tag{2.3}$$

$$\mathbf{x}_B = \mathbf{y}_B \times \mathbf{z}_B. \tag{2.4}$$

We can also specify rotation matrix R from the flat outputs:

$$R = [\mathbf{x}_B, \mathbf{y}_B, \mathbf{z}_B]\,. \tag{2.5}$$

We follow the same way with angular velocity, but now using the first derivative of motion:

$$m\dot{\mathbf{a}} = \dot{F}_t\mathbf{z}_B + F_t\dot{\mathbf{z}}_B = \dot{F}_t\mathbf{z}_B + \omega \times F_t\mathbf{z}_B, \tag{2.6}$$

and since $\dot{F}_t = \mathbf{z}_B \cdot m\dot{\mathbf{a}}$ we can write:

$$\omega \times \mathbf{z}_B = \frac{m}{F_t}(\dot{\mathbf{a}} - (\mathbf{z}_B \cdot \dot{\mathbf{a}})\mathbf{z}_B). \tag{2.7}$$

where $\omega \times \mathbf{z}_B$ is once again projection. Components of ω are then computed as:

$$p = -(\omega \times \mathbf{z}_B)\mathbf{y}_B, \tag{2.8}$$

$$q = (\omega \times \mathbf{z}_B)\mathbf{x}_B, \tag{2.9}$$

$$r = \dot{\psi}\mathbf{z}_W \cdot \mathbf{z}_B. \tag{2.10}$$

Last remaining are the angular acceleration rates $\dot{\omega}$. We start with:

$$m\ddot{\mathbf{a}} = \ddot{F}_t\mathbf{z}_B + \dot{F}_t\dot{\mathbf{z}}_B + \dot{F}_t\dot{\mathbf{z}}_B + F_t\ddot{\mathbf{z}}_B, \tag{2.11}$$

and since $\ddot{F}_t = \mathbf{z}_B \cdot m\ddot{\mathbf{a}}$ we can write:

$$\begin{aligned} m\ddot{\mathbf{a}} =& (\mathbf{z}_B \cdot m\ddot{\mathbf{a}})\mathbf{z}_B + (\mathbf{z}_B \cdot m\dot{\mathbf{a}})(\omega \times \mathbf{z}_B) + (\mathbf{z}_B \cdot m\dot{\mathbf{a}})(\omega \times \mathbf{z}_B) + \\ &+ F_t((\dot{\omega} \times \mathbf{z}_B) + (\omega \times (\omega \times \mathbf{z}_B)), \end{aligned} \tag{2.12}$$

$$\dot{p} = -(\dot{\omega} \times \mathbf{z}_B)\mathbf{y}_B, \tag{2.13}$$

$$\dot{q} = -(\dot{\omega} \times \mathbf{z}_B)\mathbf{x}_B, \tag{2.14}$$

$$\dot{r} = \ddot{\psi}\mathbf{z}_W \cdot \mathbf{z}_B. \tag{2.15}$$

The moments of force can then be computed from Eulers equation. The important conclusion here is, that all states and inputs of the system can be expressed through 4th derivative of the position $\ddot{\mathbf{a}} = \dot{\mathbf{j}} = \mathbf{s}$, which we call snap (which is the derivative of jerk), and the 2nd derivative of the yaw.

2.1.2 Controller from polynomial trajectory

The proposed nonlinear controller [LLM10] for quadcopter using differential flatness is defined by quations:

$$F_t = (-k_x \mathbf{er}_x - k_v \mathbf{er}_v + mg\mathbf{z}_W + m\mathbf{a_{ref}}) \cdot \mathbf{z}_b \tag{2.16}$$

$$\mathbf{M} = -k_R \mathbf{er}_R - k_\omega \mathbf{er}_\omega + \omega \times I\omega - I(\hat{\omega} R^T R_{ref} \omega_{ref} - R^T R_{ref} \dot{\omega}_{ref}), \tag{2.17}$$

where k_x, k_v, k_R, k_ω are the control gains and $R = R(\mathbf{\Phi})$ known rotation matrix. The $\hat{\omega}$ is the hat map angular velocity vector mapping the vector to skew symmetric matrix:

$$\hat{\omega} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}. \tag{2.18}$$

While errors of position $\mathbf{er}_p = \mathbf{p} - \mathbf{p}_{ref}$ and velocity $\mathbf{er}_v = \mathbf{v} - \mathbf{v}_{ref}$ are easily found, errors of rotational matrix $\mathbf{er}_R$ and angular velocity are more complicated:

$$\mathbf{er}_R = \frac{1}{2}(R_{ref}^T R - R^T R_{ref})^\vee, \tag{2.19}$$

whrere $^\vee$ denotes the vee map which is an inverse operation to the hat map, and:

$$\mathbf{er}_\omega = \omega - R^T R_r e f \omega_{ref} = \omega - R^T R_{ref}(R_{ref}^T \dot{R}_{ref})^\vee. \tag{2.20}$$

All remaining terms can be computed using differential flatness. Last problem to solve is to map the control force and moments to true quadcopter inputs, which are the motor conrols. I assume existence of direct mapping of required angular velocity (or produced thrust) to the voltage of the motor, so we don't have to account for motor input $u_{DC}(\omega_{DC})$. The relations between motor angular velocities and the reference inputs then are:

$$\begin{bmatrix} F_t \\ M_x \\ M_y \\ M_z \end{bmatrix} = \begin{bmatrix} k_F & k_F & k_F & k_F \\ 0 & k_F L & 0 & -k_F L \\ -k_F L & 0 & k_F L & 0 \\ k_M & -k_M & k_M & -k_M \end{bmatrix} \begin{bmatrix} \omega_{DC1}^2 \\ \omega_{DC2}^2 \\ \omega_{DC3}^2 \\ \omega_{DC4}^2 \end{bmatrix}, \tag{2.21}$$

where k_F is ratio of produced force produced by motor depending on angular velocity, k_M the ratio of moment and L is the distance of motor from the center of mass. We assume non negative motor velocities $\omega_{DC} >= 0$. Our model does not operate with motor velocities but with rotor thrust, the relation of force and moments to thrust however stays simmilar, just with different constants. It is worth mentioning that there has been effort put into incorporating quadcopter aerodynamics into control using differential flatness. In [FFS18] for example, it is shown how this controller can directly compensate for rotor drag. I have tested the performance of implemented controller on quadcopter model in Matlab before implementing it for ROS. The position tracking is nearly perfect except for slight vertical disturbances caused by trajectory sampling rate of 100 Hz. The time of trajectory was 19.49 s.

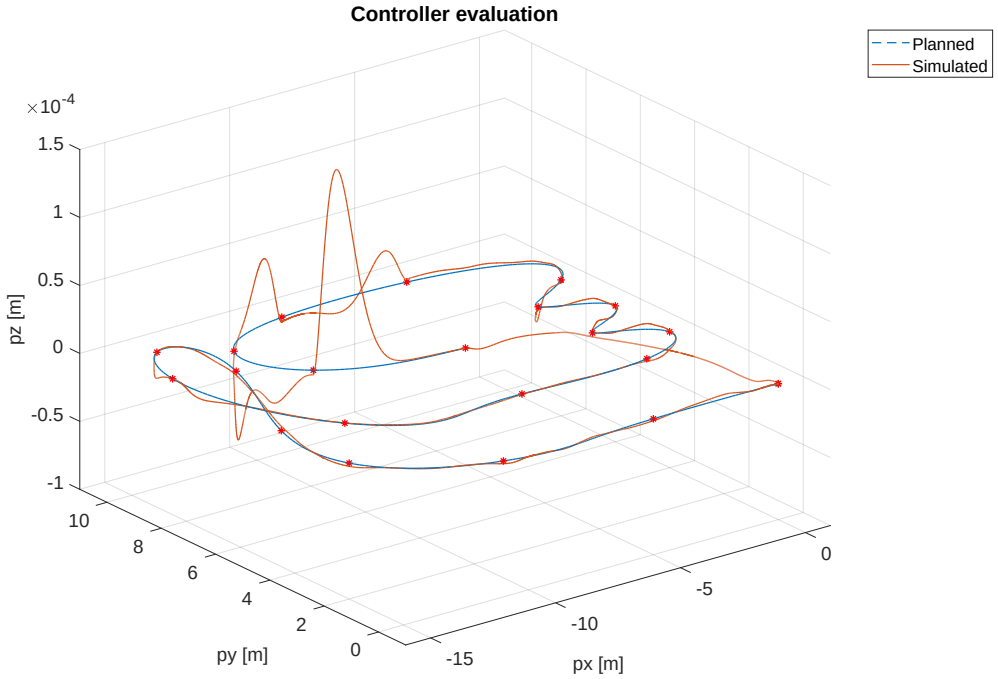


Figure 2.1: Evaluation of controller performance

2.1.3 Generation of polynomial trajectory

Since the system dynamics is dependent on jerk of the trajectory, it is reasonable to assume that by minimizing the jerk, we will minimize the energy spent by the quadcopter. This will then enable us to perform very agressive maneuvers. From the differential flatness property, we can make use

a fact that trajectory in each dimension (x, y, z) can be optimized separately. We will make use of Quadratic Programming. We seek to minimize the cost function of position x (on a single axis):

$$J = \frac{1}{2}\int_0^T (x^{[4]}(t))^2 dt = \mathbf{p}^T Q(T)\mathbf{p}, \tag{2.22}$$

where the notation $x^{[i]}$ means ith derivative of the position with respect to time. Final time T has to be specified beforehand. We also assume the initial state $x_0 = \left[x_0^{[1]}, x_0^{[2]}, x_0^{[3]}\right]$ and final state $x_T = \left[x_T^{[1]}, x_T^{[2]}, x_T^{[3]}\right]$ to be known. Let us now adopt a different notation, as the position $x(t) = \mathbf{p}(t)$ is represented by a polynomial of order N in time:

$$\mathbf{p}(t) = p_0 + p_1 t + p_2 t^2 + ... + c_N t^N. \tag{2.23}$$

The Hessian matrix $Q(T)$ comes from differenetiating the squared polynomial by its respective coefficients, as is closer explained in [RBR12]. Since we will be planning trajectory over multiple waypoints, more general description is in order since it will be contstructed from M polynomial segments:

$$J_{total} = \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix}^T \begin{bmatrix} Q_1(T_1) & & \\ & \ddots & \\ & & Q_M(T_M) \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_M \end{bmatrix}, \tag{2.24}$$

where $Q_1 ... Q_M$ are Hessian matrices of polynomial cost with their respective durations $T_1 ... T_M$. What remains is to determine the order of the polynomial. We solve this using Euler-Langrange function:

$$J = \frac{1}{2}\int_0^T (x^{[4]}(t))^2 dt = \mathcal{L}(t, x^{[4]}(t)), \tag{2.25}$$

and from the condition of optimality:

$$\frac{\partial \mathcal{L}(t, x^{[4]}(t))}{\partial x^{[4]}(t)} = 0, \tag{2.26}$$

we can determine:

$$x^{[8]}(t) = 0, \tag{2.27}$$

therefore, we require a polynomial of 7th order ($N = 7$) for minimizing snap. The joint optimization of multiple segments in addition has to be constrained in order to enforce continuity of polynomials and their derivatives by equality constrains. We enforce general equality constrains in form of:

$$A_{T_m,m}\mathbf{p}_m = \mathbf{b}_m, \tag{2.28}$$

where m is the polynomial segment number. The initial derivatives for the first segment and final derivatives for final segments are known, as are initial and final positions for all the segments, so only required continuity constrains

are on the 1st, 2nd and 3rd derivative of a segment to match the same derivatives of the next segment:

$$A_{T_m,m}\mathbf{p}_m = A_{0,m+1}\mathbf{p}_{m+1}. \tag{2.29}$$

This is the solved using a suitable QP solver. I myself have used the OPTI Toolbox for Matlab [CW12], which provides several solvers under BSD license. I have found it to be a very practical free alternative to Mathworks Optimization Toolbox.

2.1.4 Reformulation to unconstrained optimization

Before following with time optimality, we need to adress an important issue of this approach, which is its numerical instability. With 7th degree polynomials and multiple waypoints, we approach inversions of matrices with values to so small they are nearly singular. The issue is further discussed in [RBR12], and I have reached simmiliar results. Practical way to circumvent this issue is to reformulate the constrained optimization into unconstrained one and solve a mean square problem instead of quadratic programming one. Sadly, we have to give up the corridor constrains proposed in [MK11] with this approach. We do this by solving for endpoint derivatives of polynomial segments rather than for polynomial coefficients. We reformulate the polynomial cost function as:

$$J = \begin{bmatrix}\mathbf{b}_1\\ \vdots \\ \mathbf{b}_M\end{bmatrix}^T \begin{bmatrix}A_1 & & \\ & \ddots & \\ & & A_M\end{bmatrix}^{-T} \begin{bmatrix}Q_1 & & \\ & \ddots & \\ & & Q_M\end{bmatrix} \begin{bmatrix}A_1 & & \\ & \ddots & \\ & & A_M\end{bmatrix}^{-1} \begin{bmatrix}\mathbf{b}_1\\ \vdots \\ \mathbf{b}_M\end{bmatrix}, \tag{2.30}$$

and then reorder the derivatives into group of fixed $\mathbf{b}_F$ and unspecified $\mathbf{b}_p$ variables with use of permutation matrix C:

$$\begin{bmatrix}\mathbf{b}_1\\ \vdots \\ \mathbf{b}_M\end{bmatrix} = C^T \begin{bmatrix}\mathbf{b}_F\\ \mathbf{b}_P\end{bmatrix}, \tag{2.31}$$

and then substitute to previous equation to obtain:

$$J = \begin{bmatrix}\mathbf{b}_F\\ \mathbf{b}_P\end{bmatrix}^T CA^{-T}QA^{-1}C^T \begin{bmatrix}\mathbf{b}_F\\ \mathbf{b}_P\end{bmatrix} = \begin{bmatrix}\mathbf{b}_F\\ \mathbf{b}_P\end{bmatrix}^T \begin{bmatrix}R_{FF} & R_{FP}\\ R_{PF} & R_{PP}\end{bmatrix}^T \begin{bmatrix}\mathbf{b}_F\\ \mathbf{b}_P\end{bmatrix}. \tag{2.32}$$

Important fact to be noted for successful implementation is that the matrices A_m for each polynomial segments do not contain continuity constrains in unconstrained optimization, but only the equality constrains as if the values

of derivatives in matrix b were known. The final cost of unconstraned optimization is the determined as:

$$J = \mathbf{b}_F^T R_{FF} \mathbf{b}_F + \mathbf{b}_F^T R_{FP} \mathbf{b}_P + \mathbf{b}_P^T R_{PF} \mathbf{b}_F + \mathbf{b}_P^T R_{PP} \mathbf{b}_P. \tag{2.33}$$

The minimum of J can be found by equating derivative to zero and we receive well known mean square solution:

$$\mathbf{b}_P^* = -R_{PP}^{-1} R_{FP}^T \mathbf{b}_F. \tag{2.34}$$

The values of polynomials are then computed using matrix C for each segment.

2.1.5 Time optimality and input constrains

Optimization of polynomial trajectory in time is straightfoward. Since time allocated for each segment is specified before the optimization, we can first optimize with initial guess (square of distance between waypoints works well), and then proceed with simple gradient descent. The [vNM96] proposes to include the time cost into general cost function, which is the sum of costs of all trajectories (as each dimension is optimized separately):

$$J = J_x + J_y + J_z + k_T \sum_m T_m, \tag{2.35}$$

with use of some weight constant k_T. Note that when using cost 2.33 in optimization, the magnitude of cost is different from orignal 2.24, so in contrary to values proposed in [vNM96] I have found $k_T \in [0.1, 0.7]$ to work well for generating trajectories with reference motor thrust of magnitude of 10^1 for a quadcopter with weight with magnitude of 10^0. Important fact to point out here is that while total time will converge to some value depending on the k_T, the ratio between final segment times will reach the same optimum for every value of k_T. It therefore makes sense to start with some small value of k_t, find optimal segment ratios and optimize further by decreasing total time. We can also keep normalizing the total time during optimization to keep it fixed until we find the optimal ratio and then decrease it. I have reached the same results with both. Since the values of control inputs can be directly computed for trajectory at any moment, we can lower the total time until we hit some constrain on the inputs. This will however still not be optimal, since required thrust tends to peak out during longest segments, so care has to be taken to keep the segment waypoint distance simmilar for best performance.

2.1.6 Results

By using the unconstrained formulation and gradient descent, I have achieved results simmilar to those in [RBR12]. Even a complex trajectory with 20 waypoints can be reliably optimized using 7th degree polynomial. Gates were represented as waypoints. The yaw was set to 0 to further reduce controller effort. The controller effort was limited by maximum thrust of the motors.

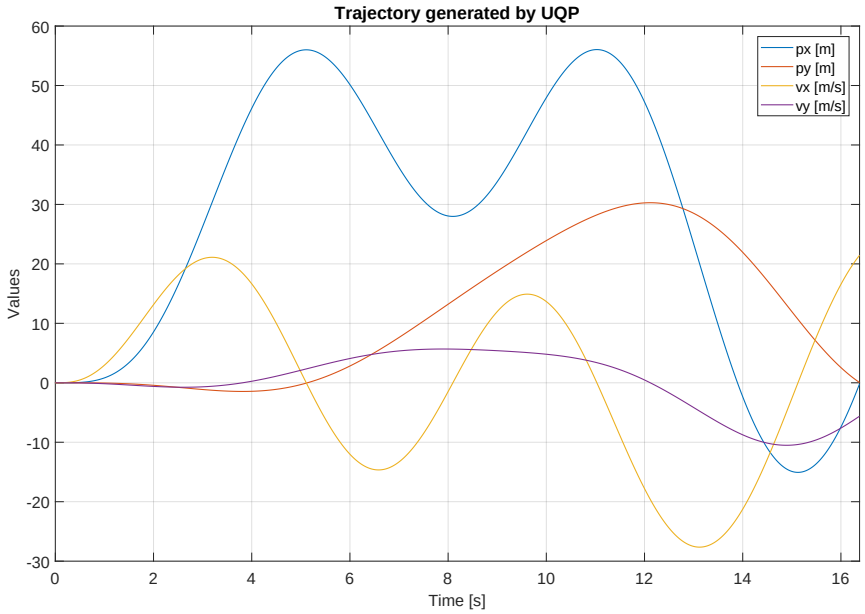


Figure 2.2: Results of UQP 1

The manufacturer specifies maximum thrust of $5\,N$ per rotor, so the value was capped at $4\,N$ to leave some margin for error correction.

The final time of polynomial trajectory was $16.38\,s$. This, while far from the record of $8.82\,s$ by a human pilot, is still quite competitive. Even more so when we take into account that we are by not using a quadcopter fully optimized for drone racing and are starting from initial hover at $1.5\,m$. The values of Z axis are not specified, since the flight is assumed to be level, there is nothing to optimize for.

One extra waypoint was added, since there are no contrains on the final velocity, to get better results. This way quadcopter does not come to a full stop at the finish line. In conclusion, this trajectory planning method proved to be the most reliable, requiring less tuning by hand compared to

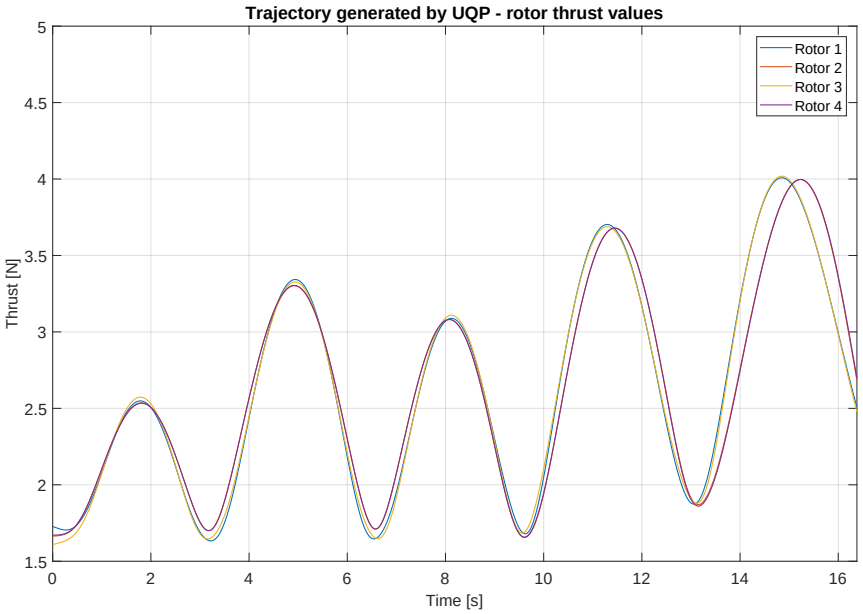


Figure 2.3: Results of UQP 2

the other methods, and since it optimizes for control effort directly, produces easiest to follow trajectories. It is not ideal for drone racing though, since the unconstrained nature and optimization for every dimension separately does not allow to specify constrains.

2.2 Frame path trajectory

Another approach to trajectory optimization proposes use of so called framed path [SNBF16][SN18]. The main idea is to represent the race track as an arc-length parametrized locally non intersecting curve in space $\mathbf{p}_f(s), \forall s \in [0, L]$ where L is the total length of curve, and the trajectory along the track as states in a Serret-Frenet frame. By deriving the position vector $\mathbf{p}_f(s)$ according to arc-length s, we receive:

$$\begin{aligned} \mathbf{t}(s) &= \frac{d\mathbf{p}_f(s)}{ds} \\ \mathbf{n}(s) &= \frac{1}{k(s)} \frac{d^2\mathbf{p}_f(s)}{ds^2}, \\ \mathbf{b}(s) &= \mathbf{t}(s) \times \mathbf{n}(s) \end{aligned} \tag{2.36}$$

which are tangent, normal and binormal vectors of the path respectively and $k(s)$ is the curvature. We denote $\tau(s) = \mathbf{n}(s)\frac{d\mathbf{b}(s)}{ds}$ as torsion. By mapping

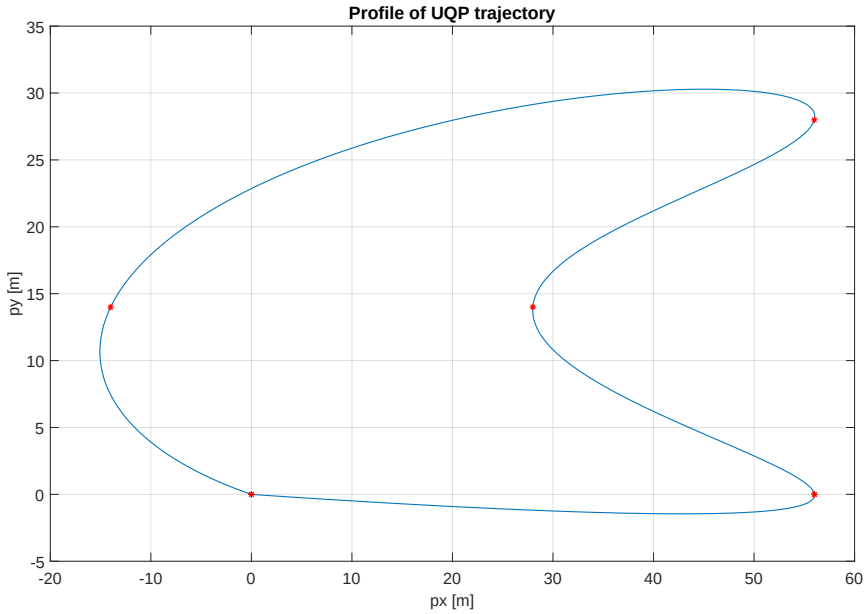


Figure 2.4: Profile of UQP trajectory

the position vector to the closest position on path by orthogonal projection:

$$s_f(t) = \arg\min_s \|\mathbf{p}(t) - \mathbf{p}_f(s)\|^2 . \tag{2.37}$$

2.2.1 Transverse dynamics

We can describe position along the path using transverse coordinates:

$$\begin{aligned} w1 &:= \mathbf{n}(s_f)(\mathbf{p}(t) - \mathbf{p}_f(s_f)) \\ w2 &:= \mathbf{b}(s_f)(\mathbf{p}(t) - \mathbf{p}_f(s_f)). \end{aligned} \tag{2.38}$$

Using approach described in detail in [SNBF16], we can derive formulation of previously shown simplified thrust vector model dynamics in transverse coordinates, if we state that:

$$t(s_f) = \frac{1 - k(s_f)w1(s_f)}{\mathbf{t}(s_f) \cdot \mathbf{v}(s_f)} = \frac{1}{\dot{s}_f}, \tag{2.39}$$

ie. all variables which are functions of time only depend on it through s_f in the transverse coordinates. The transverse dynamics are then:

$$\begin{aligned}
\dot{w}_1 &= \mathbf{n} \cdot \mathbf{v} \frac{1}{\dot{s}_f} + \tau w_2 \\
\dot{w}_2 &= \mathbf{b} \cdot \mathbf{v} \frac{1}{\dot{s}_f} - \tau w_1 \\
\dot{\mathbf{v}} &= (-g\mathbf{z}_W + \frac{F_t}{m} R(\mathbf{\Phi})\mathbf{z}_W) \frac{1}{\dot{s}_f} \\
\dot{\mathbf{\Phi}} &= J(\Phi)\omega \frac{1}{\dot{s}_f},
\end{aligned} \tag{2.40}$$

since derivative of any function $\alpha(t(s_f))$ by s_f will be $\dot{\alpha}(t(s_f))\dot{s}_f$. The advantage of this approach to trajectory formulation is that we can specify some arbitrary path along the race track, and then limit the space of possible trajectories by putting contrains on states $w1$ and $w2$. We can enforce limits on gate and obstacle safety in the same way. The dynamic constrains are satisfied by constrains of system inputs F_t and ω.

2.2.2 Cost and constrains

Since we seek to minimize time, we define the funtional as:

$$J_t = T = \int_0^T 1dt = \int_{s_f(0)}^{s_f(T)} t(s)ds = \int_0^L \frac{1}{\dot{s}_f} ds \tag{2.41}$$

, ie. we minimize the total length of the trajectory by maximizing the velocity along path. To enforce path and dynamic constrains, we will apply the barrier fuction approach introduced in [HS06]. This consists on relaxing the constrains by including them in the cost function to be minimized. General constraint function $c_f(x(s_f))$ of some state or input $x(s_f)$ is:

$$c_f(x(s_f)) = (\frac{2x(s_f) - (x_{max} + x_{min})}{x_{max} - x_{min}})^2 - 1 \leq 0, \tag{2.42}$$

and in cases where $x_{max} = x_{min}$ (angle and angular rate limits):

$$c_f(x(s_f)) = (\frac{x(s_f)}{x_{max}})^2 - 1 \leq 0. \tag{2.43}$$

The cost of constraint would then be computed using logarhitm. However, this implies that the trajectory has to be feasible. In order to optimize on both feasible and infeasible trajectories (initial guess for example might not be feasible), we make use of the barrier function in form:

$$\beta_l(c_f) := \begin{cases} -\log(c_f), \text{ for } x > l \\ -\log(l) + \frac{1}{2}((\frac{x-2l}{l})^2 - 1), \text{ for } x \leq l \end{cases}. \tag{2.44}$$

The final cost functional and optimization problem is then:

$$\begin{aligned} & \min_{x,u} \int_0^L \frac{1-kw1}{\mathbf{t}\cdot\mathbf{v}} + \epsilon \sum_j \beta_\nu(c_{f,j}(\mathbf{x}(s_f),\mathbf{u}(s_f)))ds + \epsilon_{fin} \sum_i \beta_{\nu_{fin}}(-c_{f,i}(\mathbf{x}(L))) \\ \text{subject to} \quad & \dot{\mathbf{x}} = f(\mathbf{x}(s),\mathbf{u}(s)) \\ & \mathbf{x}(0) = \mathbf{x}_0 \\ & c_f(\mathbf{x}(L)) \leq 0 \\ & c_f(\mathbf{x}(s_f),\mathbf{u}(s_f)) \leq 0, \forall s_f \in [0,L]\,, \end{aligned} \tag{2.45}$$

where $\epsilon, \epsilon_{fin}, \nu, \nu_{fin}$ are the weights of constrains and will be explained later. Also note that cost functional is strongly convex. Having estabilished transverse dynamics, cost function and contrains, we can proceed to the optimization.

2.2.3 PRONTO optimization

Projection Operator Newton Descent Optimization [Hau02] approach to trajectory optimization has been known for nearly two decades, yet has lately seen revival in trajectory planning for autonomous vehicles[KLM+17]. I refer interested reader to the textbook [FPN18], and will only outline basic principles of the algorhitm. The Matlab implementation developed for the purpose of this work is available too. Let us define the projection operator $\mathcal{P} : \xi \to \eta$ as a mapping from some bounded state control curve $\xi = (\alpha(s_f), \mu(s_f))$ to a feasible (from the standpoint of system dynamics) system trajectory $\eta = (\mathbf{x}(s_f), \mathbf{u}(s_f))$ for $s \geq 0$. This is done by continuously integrating nonlinear feedback system:

$$\begin{aligned} \dot{\mathbf{x}} &= f(\mathbf{x},\mathbf{u}) \\ \mathbf{u} &= \mu(s_f) + K(s_f)(\alpha(s_f) - \mathbf{x}), \end{aligned} \tag{2.46}$$

where $\mathbf{x}(0) = \mathbf{x}_0$ and $f(\mathbf{x},\mathbf{u})$ are the system dynamics. State feedback gain $K(s)$ can be computed in fashion of finite horizon LQR controller for time varying system using linearized dynamics (gain scheduled LQR) along the control trajectory by solving the Riccati diff. equation backwards in arc-length. Using the projection operator, we can perform unconstrained optimization, since the cost of trajectory is $g(\eta) = h(\mathcal{P}(\eta))$ where $h(\eta) = \int_0^L \frac{1-kw1}{\mathbf{t}\cdot\mathbf{v}} + \epsilon \sum_j \beta_\nu(c_{f,j}(\mathbf{x}(s_f),\mathbf{u}(s_f)))ds + \epsilon_{fin} \sum_i \beta_{\nu_{fin}}(-c_{f,i}(\mathbf{x}(L)))$. Newton descent direction ζ_i for current trajectory iterate ξ_i is obtained from first and second Fréchet differentials $\mathcal{DP}(\xi)$ and $\mathcal{D}^2\mathcal{P}(\xi)$:

$$\zeta_i = \arg\min_\zeta \mathcal{D}h(\xi_i) + \frac{1}{2}\mathcal{D}^2 g(\xi_i), \tag{2.47}$$

and the trajectory is updated:

$$\xi_{i+1} = \mathcal{P}(\xi_i + \gamma\zeta_i), \tag{2.48}$$

where γ is some positive constant obtained for example by line search. We repeat this iterative algorhitm until solution converges to some local minimum. That is bound to happen, since the cost functional, which is also part of Frechét derivatives is strongly convex and has strictly positive second derivative. The search direction can be neatly found by backward and forward integration (solving LQ problem), if analytical second derivatives of dynamics and cost functional are known. We also recompute feedback gain $K(s)$. As a first step, we obtain some initial trajectory ξ_0. Lets repeat that trajectory is defined as states and controls of the system $\xi = (\mathbf{x}(.), \mathbf{u}(.))$. This can easily be done by polynomial optimization. Then we proceed with PRONTO algorhitm. After solution converges to some local minimum, we reduce the tuning parameters $\epsilon, \epsilon_{fin}, \nu, \nu_{fin}$ (for example by half), and repeat the PRONTO. This will place lower weight on constrains and more on original cost J_t, and the trajectory will converge to the optimum while respecting constrains. Strength of this approach is that each subsequent iteration is performed on tangent trajectory manifold.

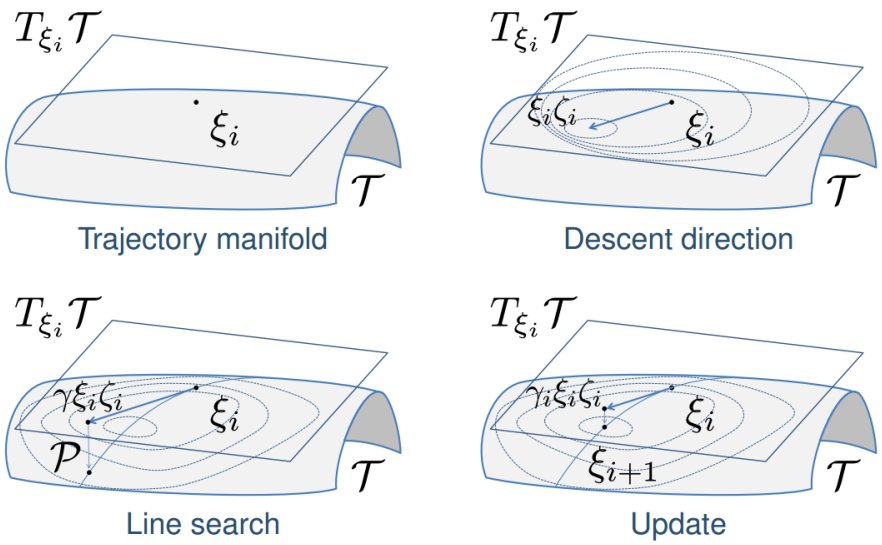


Figure 2.5: PRONTO optimization algorhitm from [Hau02]

2.2.4 Results

Although I have implemented frame path optimization with PRONTO in Matlab, I have not been successful in running it. There are issues not

adressed in the [SNBF16], which is very vague about some parts of the implementation, for example of the feedback gain K which fails to reliably stabilize the transverse dynamics for $\|\mathbf{v}\| \neq 1$, and discretization of trajectory (although the PRONTO algorhitm is continuous, trajectory ξ is saved in form of discrete states and control inputs in memory). One significant issue is that the second derivative of the system dynamics is not positive definite along the trajectory, so the search direction can not be found - the Riccati diff. equation is not numericaly stable. There has been significant effor put into understanding the approach and implementing it, and the reader is invited to inspect the implementation.

2.3 Nonlinear programming trajectory

Approach through nonlinear optimization is fundamentally different. We no longer make use of the differential flatness, but of the entire the system model, since performing black-box optimization through shooting approach would be nearly impossible. There is variety of tools for nonlinear optimization, I myself have used the freely available FALCON.m [RBG$^+$] toolbox for Matlab, which provides interface to the IPOPT [WB06] solver. Although this aproach has been described as unreliable and unstable in various literature, it appears to be working just fine and yields truly time optimal trajectory.

2.3.1 Description of algorhitm

As before, we seek to minimize the total time of trajectory, or rather sum of segment times $J = \sum T_m$. We keep the term of segments, however this time the segments are not represented by polynomials connecting waypoints in time, but discrete state and control grids. The segments are connected by constrains simmilar to those used in polynomial optimization, to enforce state continuity. We begin with implementation of the mathematical model 1.1 in Matlab. We include the motor dynamics in the model with the time constant $\tau_{DC} = 0.2\,s$ for safety. The model knowledge is important, since IPOPT uses the knowledge of the model analytical derivatives in optimization. While this can be done by hand as is shown in frame path optimization [SN18], we can make use of FALCON.m implementation of Symbolic Toolbox and Coder Toolbox which compiles the derivatives and discretization into a mex file, which makes the optimization process significantly faster. The segments between waypoints are described as phases during the optimization proces. We begin with initial guesses on the segment times (squared waypoint distances

as before). We can specify initial guess for system states, although the framework can construct it by itself either through interpolation or simulation of dynamics.

2.3.2 IPOPT solver

I would like to make a quick remark here about limits of the IPOPT algorhitm. Since the algorhitm is gradient based, it makes use of system derivatives and in theory only provides solution in form of some local minimum, which is dependent on the initial trajectory. I have however found the resulting optimal trajectory to be the same for both generated initial guess and initial guess from polynomial optimization.

2.3.3 Results

Although computationaly quite heavy, the algorhitm converges to optimum (or in this case, 10^{-5} constrain error) in about 50 seconds on conventional PC. The resulting trajectory is very interesting, and emulates flight trajectories of professional pilots. Since the NLP optimization can make use of full dynamics,

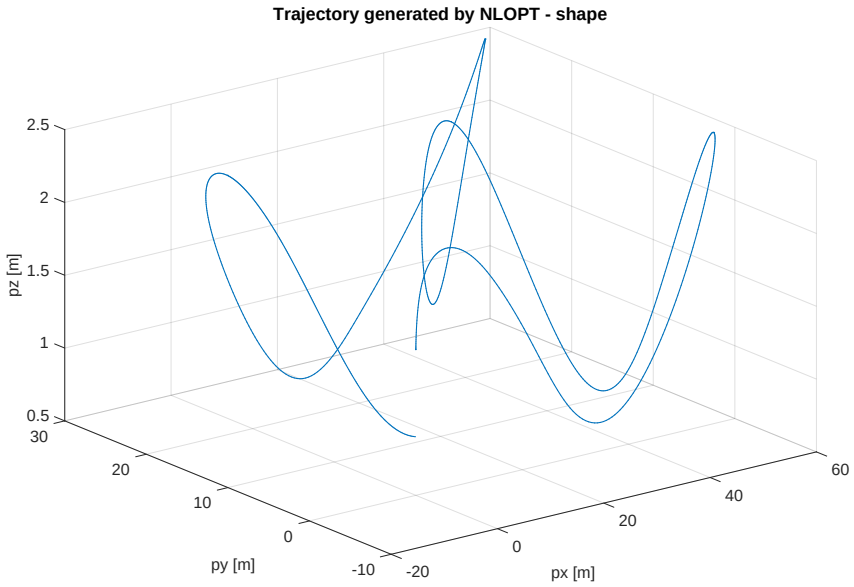


Figure 2.6: NLP trajectory in space

it inherently tries to maximize thrust over time. Energy, which does not transfer directly to motion between gates is saved in height. The limits on height are $[0.5, 2.5]$ which complies with size of the gates. For the sake of comparison with other methods, I have kept the waypoint representation of race track, but it is possible to add any constrains. The importance

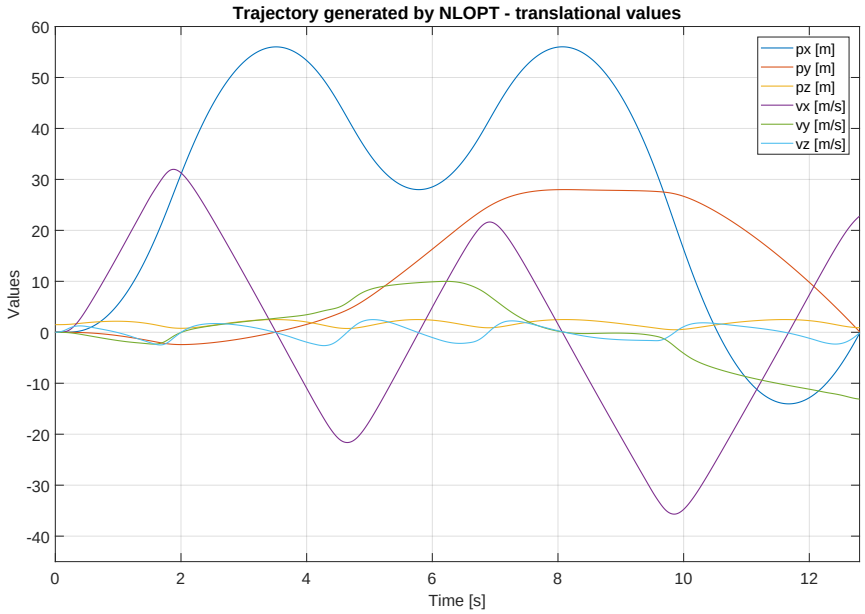


Figure 2.7: Results of NLP 1

of including motor dynamics is obvious from rotor thrust graph. Since the optimization tries to keep the total thrust at maximal value, the control signal would not be continuous. The thrust was limited at $4\,N$ to give controller some safety margin. Final time of the trajectory is $16.83\,s$, which is quite good. Of course, the flight time could further be improved by removing safety margins, but I have found the controller unable to cope with such agressive trajectories.

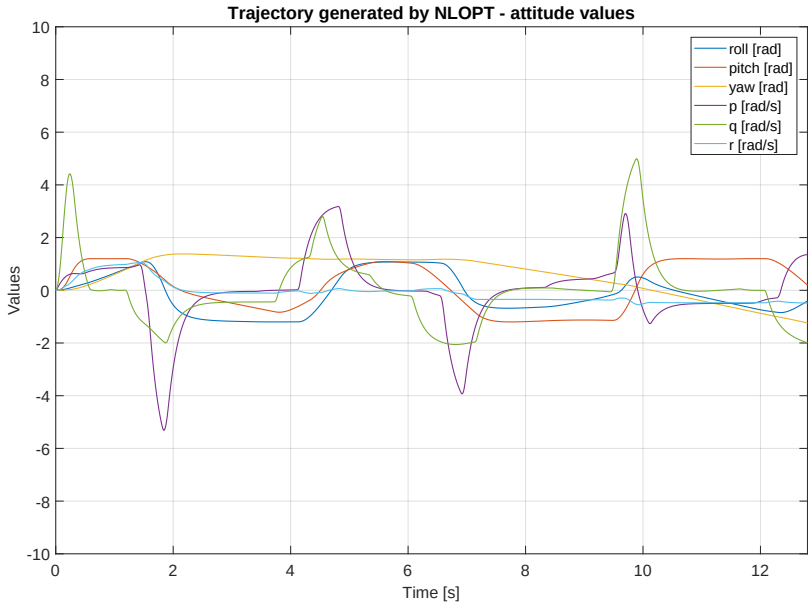


Figure 2.8: Results of NLP 2

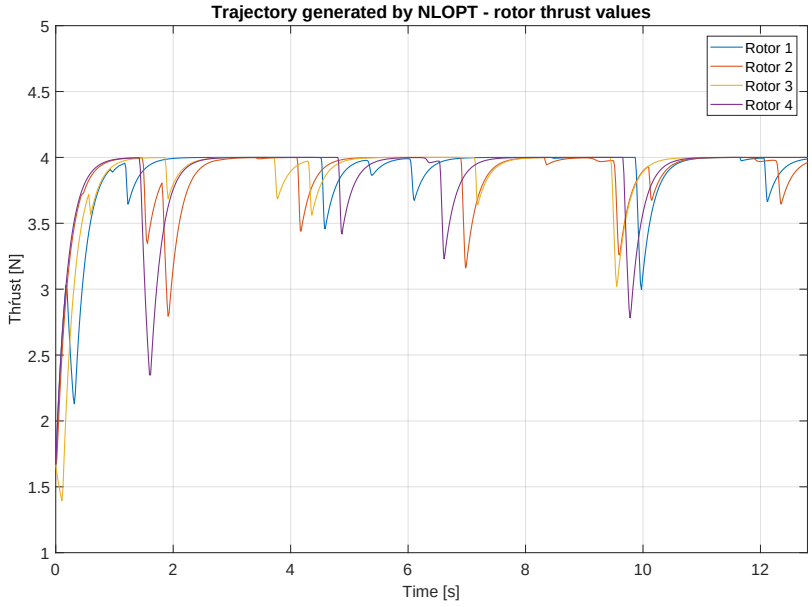


Figure 2.9: Results of NLP 3

Chapter 3

Simulation

To test the performace of proposed algorhitms and trajectories, I have decided to simulate races using Gazebo simulation environment [KH] and implement control using Robot Operating System [QCG+09] (Gazebo 7 with ROS Kinetic running at Ubuntu 16.04 VM).

3.1 ROS enviroment

Creating an autopilot software from a scratch would be a huge undertaking, so I decided to implement the controller using a suitable existing framework, which there are several to choose from. I have chosen the RotorS [FBAS16] MAV simulator, which provides compact interface from ROS to Gazebo simulator and supports Hardware In the Loop, if we later decided to test the performance on real hardware. ROS provides means of communication between computers, sensors and acutators in (most notably) the field of robotics. Separate parts of the system (nodes) communicate over some unspecified physical, datalink and transport network layer, while negotiation of specific protocol and communication details between them like bandwidth is handled by the ROS core node. A notable ROS package (extension, implementation of some function) used in the simulation is MavRos, which implements the MavLink protocol. This is a protocol designed for communication of ground-station and MAV, and is supported by majority of commercial autopilots. This enables direct communication of our quadcopter with ROS.

3.1.1 Quadcopter controller

There already is an existing implementation of cotroler in RotorS, which is based on the controler proposed in [LLM10]. It is however not suitable for the fully specified discrete trajectory represented as state and control grids or a polynomial trajectory. I have therefore extended it with capability to follow complete reference. Note that the reference used by the controller does not have to come from differential flatness, but can be provided directly. The overall structure is a quadcopter controller node subscribing to odometry and waypoint messages and publishing motor inputs. The odometry is provided from the simulation or state estimator, and the waypoint messages are generated by a publisher from provided trajectory file, which contains the discretized trajectory at $100\,Hz$. These are published by groundstation node, which controls switching between flight modes (hover, trajectory following) and publishes motor inputs at the frequency of about (but is not limited to) $50\,Hz$. The attitude controller is initiated not by current position reference, but by odometry which is published at the rate of $100\,Hz$, however due to computational load (caused by either controller, Gazebo or ROS delay itself), is slowed to $50\,Hz$. This is however reliable rate at which odometry and controls can be computed.

3.1.2 Sensor model

There are two possibilities how to obtain sensory data in Gazebo environment. We can either extract precise odometry straight from the simulation, the ground truth data, or we can implement a simulated sensor through the URDF model of the quadcopter and define its properties. I have used the RotorS an implementation of IMU with added uncertanities, coupled with a state estimator. The controller has however shown large sensitivity and I have failed to finish a race using it.

3.2 Simulated races

3.2.1 Polynomial trajectory

The simulation of fully modeled quadcopter with ROS controller on the polynomial trajectory produced acceptable results. The final time of the race was 16.75 s. The delay is caused mainly by the ROS control system, as the state reference output and control algorhitm output have to travel by ROS messages. This issue would be circumvented on real hardware by implementing the attitude control loop directly on the drone autopilot. Here

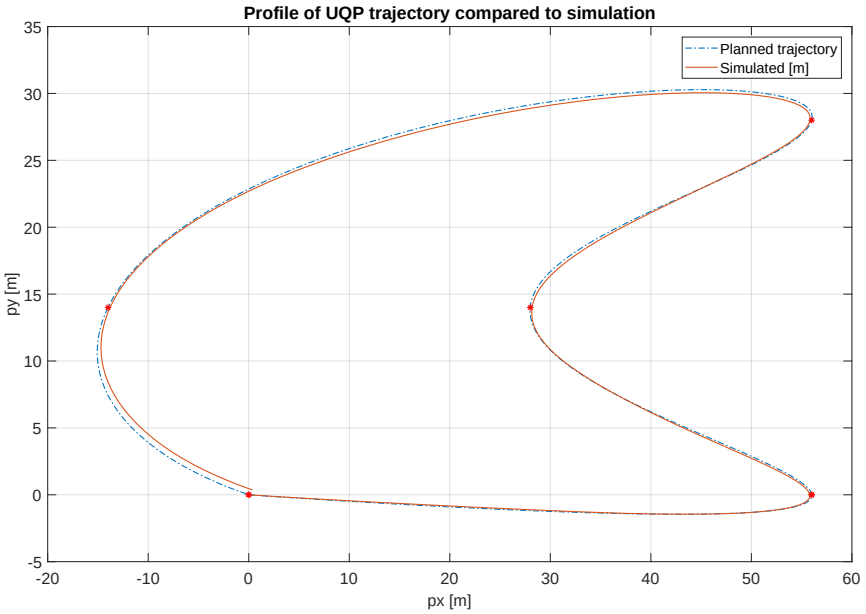


Figure 3.1: Simulation of UQP trajectory - profile

we can compare the position values for reference and simulation. Perfect tracking was not expected due to incomplete model, however excluding the delay, the position profile matches the reference profile quite well.

3.2.2 NLP trajectory

Just as with polynomial trajectory, the trajectory control was delayed and the final time was 13.15 s. From general shape of simulated trajectory, we can see that we have reached limits of the controller, which struggled to follow

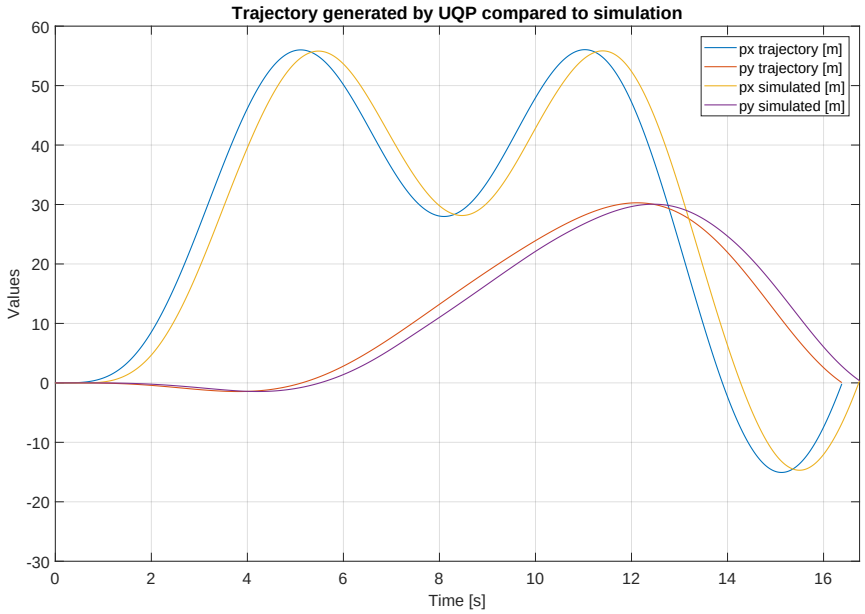


Figure 3.2: Simulation of UQP trajectory - position

the Z axis reference. The other references however were followed quite well, albeit with constant disturbance. It is also useful to inspect the profile of simulated trajectory. Here, the deviation is more rampant, yet still under $2\,m$ at the waypoints, which is the safety margin.

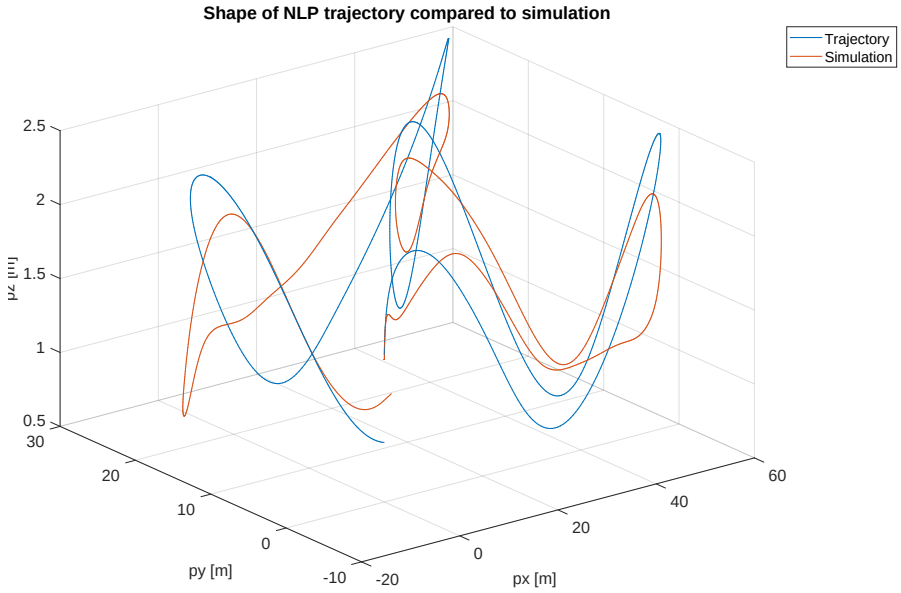


Figure 3.3: Simulation of NLP trajectory - shape

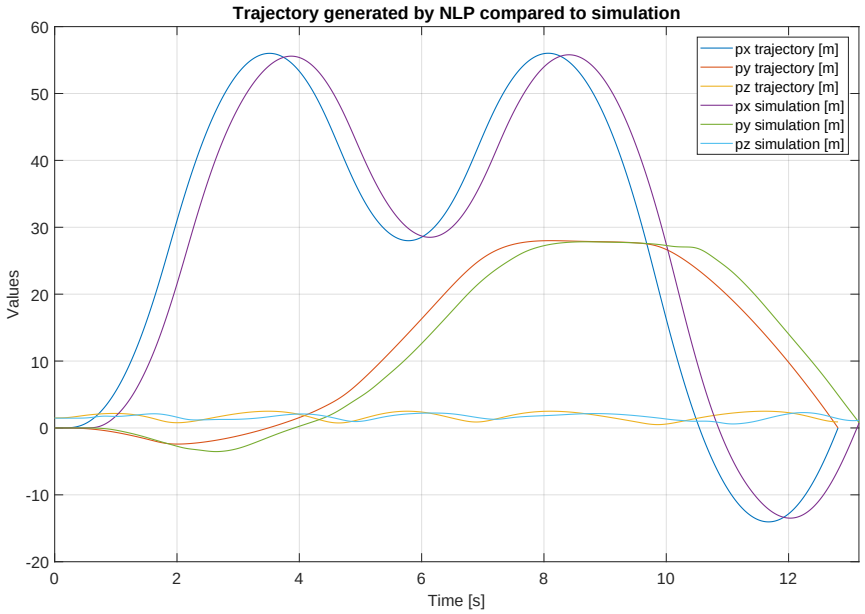


Figure 3.4: Simulation of NLP trajectory - position

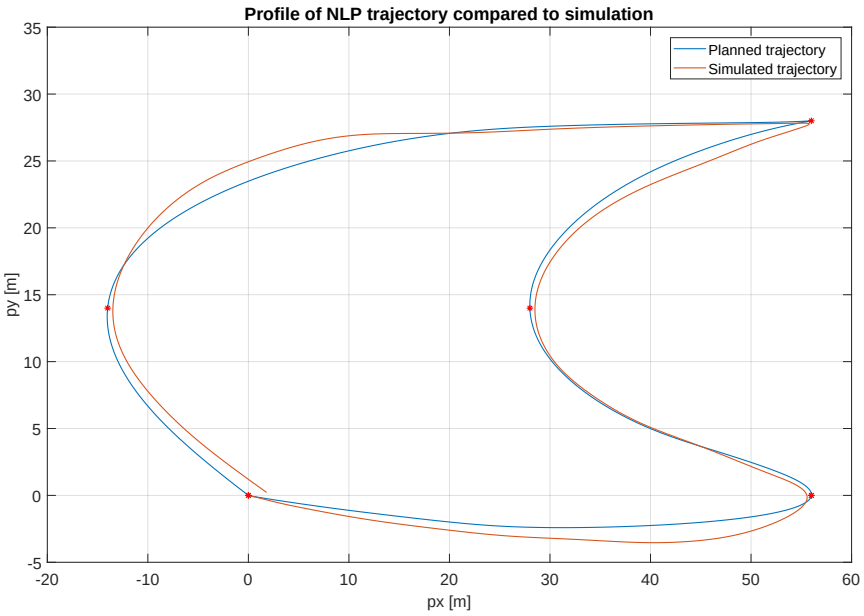


Figure 3.5: Simulation of NLP trajectory - profile

Chapter 4

Conclusion

In this work, I have presented the idea of quadcopter drone racing, how it can be approached from the engineering point of view and how can autonomous drone race trajectory planning control be tackled. I have presented simulation using in the industry estabilished control methods for drone operation. The span of topics discussed in this work was quite broad, and all of them are further discussed and proven in the cited literature. I have shown that it can be entirely possible to develop complex algorhitms and simulations purely in simulation and test their performance without requiring real hardware. Most of the tools required to reproduce this work can be obtained for free under GPL or simmilar license, although Matlab, which was used for trajectory planning and development of quadcopter controller requires academical or commerical license. I have not been able to simulate the races simulated sensors coupled with a state estimation, which will require more development. The ROS RotorS framework can be further improved upon, as the nonlinear trajectory planning algorhitm is only limited in quality of trajectories by our constrains, and those are imposed to keep the quadcopter with controller stable. Reimplementation of the controller on the UAV and general refinement of the communicaiton structure is in order.

Bibliography

[CFCH14] Anežka Chovancová, Tomáš Fico, Ľuboš Chovanec, and Peter Hubinsk, *Mathematical modelling and parameter identification of quadrotor (a survey)*, Procedia Engineering **96** (2014), 172–181.

[CW12] Jonathan Currie and David I. Wilson, *OPTI: Lowering the Barrier Between Open Source Optimizers and the Industrial MATLAB User*, Foundations of Computer-Aided Process Operations (Savannah, Georgia, USA) (Nick Sahinidis and Jose Pinto, eds.), 8–11 January 2012.

[FBAS16] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart, *Robot operating system (ros): The complete reference (volume 1)*, ch. RotorS—A Modular Gazebo MAV Simulator Framework, pp. 595–625, Springer International Publishing, Cham, 2016.

[FFS18] Matthias Faessler, Antonio Franchi, and Davide Scaramuzza, *Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories*, IEEE Robot. Autom. Lett. **3** (2018), no. 2, 620–626.

[FPN18] Thor I. Fossen, Kristin Y. Pettersen, and Henk Nijmeijer, *Sensing and control for autonomous vehicles applications to land, water and air vehicles*, Springer International Publishing, 2018.

[Hau02] John Hauser, *A projection operator approach to the optimization of trajectory functionals*, IFAC Proceedings Volumes **35** (2002), no. 1, 377–382.

[HS06] John Hauser and Alessandro Saccon, *A barrier function method for the optimization of trajectory functionals with constraints*, Proceedings of the 45th IEEE Conference on Decision and Control (2006).

[JHSS18] Sunggoo Jung, Sunyou Hwang, Heemin Shin, and David Hyunchul Shim, *Perception, guidance, and navigation for indoor autonomous drone racing using deep learning*, IEEE Robotics and Automation Letters **3** (2018), no. 3, 2539–2544.

[KH] N. Koenig and A. Howard, *Design and use paradigms for gazebo, an open-source multi-robot simulator*, 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566).

[KLM+17] N. Kingry, Y. Liu, M. Martinez, B. Simon, Y. Bang, and R. Dai, *Mission planning for a multi-robot team with a solar-powered charging station*, 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Sep. 2017, pp. 5233–5238.

[LLM10] T. Lee, M. Leok, and N. H. McClamroch, *Geometric tracking control of a quadrotor uav on se(3)*, 49th IEEE Conference on Decision and Control (CDC), Dec 2010, pp. 5420–5425.

[MK11] D. Mellinger and V. Kumar, *Minimum snap trajectory generation and control for quadrotors*, 2011 IEEE International Conference on Robotics and Automation, May 2011, pp. 2520–2525.

[MMR03] Philippe Martin, Richard Murray, and Pierre Rouchon, *Flat systems, equivalence and trajectory generation.*

[QCG+09] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng, *Ros: an open-source robot operating system*, ICRA Workshop on Open Source Software, 2009.

[RBG+] Matthias Rieck, Matthias Bittner, Benedikt Grüter, Johannes Diepolder, and Patrick Piprek, *Falcon.m user guide.*

[RBR12] Charles Richter, Adam Bry, and N. Roy, *Polynomial trajectory planning for quadrotor flight*, 2012.

[SN18] Sara Spedicato and Giuseppe Notarstefano, *Minimum-time trajectory generation for quadrotors in constrained environments*, IEEE Transactions on Control Systems Technology **26** (2018), no. 4, 1335–1344.

[SNBF16] Sara Spedicato, Giuseppe Notarstefano, Heinrich H. Bülthoff, and Antonio Franchi, *Aggressive maneuver regulation of a quadrotor uav*, Springer Tracts in Advanced Robotics Robotics Research (2016), 95–112.

[vNM96] Michiel van Nieuwstadt and Richard M. Murray, *Real time trajectory generation for differentially flat systems*, IFAC Proceedings Volumes **29** (1996), no. 1, 2301 – 2306, 13th World Congress of IFAC, 1996, San Francisco USA, 30 June - 5 July.

[WB06] Andreas Wächter and Lorenz T. Biegler, *On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming*, Math. Program. **106** (2006), 25–57.

[WBH+11] Jian Wang, Thomas Bierling, Leonhard Höcht, Florian Holzapfel, Sebastian Klose, and Alois Knoll, *Novel dynamic inversion architecture design for quadrocopter control*, Advances in Aerospace Guidance, Navigation and Control (2011), 261–272.

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Nekovář František** Personal ID number: **420212**

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

Department / Institute: **Department of Control Engineering**

Study program: **Cybernetics and Robotics**

Branch of study: **Systems and Control**

II. Master's thesis details

Master's thesis title in English:

Optimal Trajectory Planning for a Quadrotor UAV for Autonomous Drone Race

Master's thesis title in Czech:

Plánování optimální trajektorie pro čtyř-rotorovou helikoptéru pro závody autonomních dronů

Guidelines:

1. Study the problem of autonomous drone racing, describe rules of existing competitions, environment, drone equipage and physical limitations.
2. Survey existing optimal trajectory planning methods.
3. Create mathematical model of quadrotor UAV.
4. Design and implement control algorithms for Robot Operating System (ROS).
5. Create model of UAV including localization sensors and simulation environment for virtual drone race competition.
6. Experimentally evaluate features of proposed algorithm in a set of simulated competitions.

Bibliography / sources:

[1] K. Bipin, V. Duggal and K. M. Krishna, 'Autonomous navigation of generic Quadrocopter with minimum time trajectory planning and control,' 2014 IEEE International Conference on Vehicular Electronics and Safety, Hyderabad, 2014, pp. 66-71.
[2] Y. Liu, S. Longo and E. C. Kerrigan, 'Nonlinear predictive control of autonomous soaring UAVs using 3DOF models,' 2013 European Control Conference (ECC), Zurich, 2013, pp. 1365-1370.
[3] B. Penin, R. Spica, P. R. Giordano and F. Chaumette, 'Vision-based minimum-time trajectory generation for a quadrotor UAV,' 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 2017, pp. 6199-6206.

Name and workplace of master's thesis supervisor:

Ing. Milan Rollo, Ph.D., Artificial Intelligence Center, FEE

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

Date of master's thesis assignment: **16.01.2018** Deadline for master's thesis submission: **08.01.2019**

Assignment valid until: **30.09.2019**

Ing. Milan Rollo, Ph.D.	prof. Ing. Michael Šebek, DrSc.	prof. Ing. Pavel Ripka, CSc.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

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

. _____
Date of assignment receipt

_____
Student's signature