MŰEGYETEM 1782

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Trajectory Tracking Control Methods for an Autonomous Quadcopter

SZAKDOLGOZAT

Készítette
Török Ferenc

Konzulensek
Péter Stumpf /BME AUT/
Tamás Péni /MTA SZTAKI/

December 18, 2018

Table of contents

List of Figures

HALLGATÓI NYILATKOZAT

Alulírott *Török Ferenc*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhetõ elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, December 18, 2018

Török Ferenc
hallgató

Abstract

The problem of designing a trajectory tracking controller for a quadcopter UAV is addressed in this document. The motivation is to use the quadrotors in an industrial environment, where reliable, fast and precise execution of the flight tasks is mandatory. The goal therefore is to develop a control architecture that is able to satisfy these strict requirements. The control architecture consists of a trajectory designer and a controller part.

Two trajectory designing algorithms are going to be introduced, both of them presents a solution for fitting a trajectory on desired points in the 3D space. The methods have to give a feasible trajectory (for the quadcopter) for which the requirements originate in the dynamical and also in the motor and rotor properties of the UAV.

The first solution for the trajectory design fits a third order spline on the desired trajectory points. The quadcopter accelerates to a constant speed, with which the trajectory is feasible and is safe to complete. At the end of the path, the quadcopter decelerates to zero speed and remains in a hovering position. The velocity profile is designed so, that the path will be carried out under the shortest possible time subject to the actuator and plant constraints.

The second solution connects the desired points with straight lines. The smoothness and hence the feasibility of the path is achieved with a carefully chosen velocity profile which is also is also designed so that the path will be carried out under a priori given completion time.

For the control, three different solutions are going to be introduced, the performance of which was analyzed via numerical simulations on a high fidelity nonlinear model of an experimental quadcopter aircraft. The three methods are briefly introduced as follows:

Successive input/output linearization: This method introduces a two level control architecture based on successive linearization and LQR regulation. The method has been proposed in [1].

Jacobi linearization and LQG control: Based on the Jacobi linearized model of the quadcopter around a trim-point, an LQG control can be developed. It can be then further improved in order to achieve trajectory following capability.

Feedback linearization: Through a change in the input variables, a system can be obtained which than can be feedback linearized. For the feedback linearized system, an LQR controller can be applied, making the controller capable of trajectory following in such way.

The performance of the controllers was analyzed via simulations in the MATLAB-Simulink environment. In these simulations, the controllers were tested on the different types of

trajectories with different speeds. Simulations with changed quadcopter properties, such as mass and inertia, were also carried out in order to check the robustness of the controllers. 3D visualization is solved in V-REP and in MATLAB-figure as well.

Kivonat

A szakdolgozat egy autonóm quadcopter trajektória követő szabályzásának tervezését mutatja be. A motiváció a quadcopter használata ipari környezetben, ahol a repülési feladatok megbízható, gyors és pontos kivitelezése elengedhetetlen. A cél ebből kifolyólag egy olyan szabályzó architektúra tervezése, ami kielégíti ezen szigorú követelményeket. A control architektúra magába foglalja a trajektória tervező és a control software-t is.

Kétféle trajektória tervező algoritmus kerül bemutatásra, melyek a 3D térben előre megadott pontokra illesztenek pályát. Az elkészült trajektóriáknak teljesíthetőeknek kell lenniük a quadcopter által, ennek feltételeit a quadcopter dinamikája valamint a rotor és motor tulajdonságai határozzák meg.

Az első trajektória tervező megoldás egy harmadfokú spline-t illeszt a meghatározott pontokra. A quadcopter a pálya elején felgyorsul egy konstans sebességre, amivel biztonságosan kivitelezhető a teljes repülés. Az utolsó pont előtt pedig lelassul nulla sebességre és egy helyben lebeg. A sebességprofil úgy kerül megtervezésre, hogy a pálya teljesítési ideje a lehető legrövidebb legyen miközben az aktuátor és a rendszer által felállított korlátok nem sérülnek.

A második megoldás egyenes vonalakkal köti össze a meghatározott pontokat a térben. A trajektória kellő simaságát és így teljesíthetőségét a megfelelően választott sebességprofillal lehet elérni. A sebességprofil megtervezésénél ebben az esetben a szempont az előbb említetteken kívül, hogy a pálya egy előre meghatározott idő alatt legyen teljesítve.

Ebben a dokumentumban háromféle szabályzó van bemutatva, amelyek működését és teljesítményét numerikus szimulációval elemeztem a quadcopter nagy részletességű modelljén. A három szabályozót a következőképpen lehet röviden jellemezni:

Több lépéses input/output linearizáció: Ez a módszer egy két szintű szabályzási stratégiát mutat be amelynek alapja egy több lépéses linearizálás és az így kapott rendszerre alkalmazott két szintű LQR típusú szabályzó. A módszer [1] cikken alapul.

Jacobi linearizálás és LQG szabályzás: A quadcopter egy trim-pont körüli linearizált modelljére LQG szabályzó alkalmazható. Az így kapott szabályzó tovább fejleszthető a trajektória követő képesség elérése érdekében.

Feedback linearizáció: A bemeneti változók megfelelő megváltoztatásával elérhető egy rendszer amit feedback linearizálni lehet. Az ilyenképpen linearizált rendszerre LQR szabályzót lehet alkalmazni, így elérve a kívánt trajektória követő képességet.

A szabályzók teljesítményét numerikus szimulációval elemeztem a MATLAB-Simulink környezetben. Ezekben a szimulációkban a szabályzókat teszteltem a korábban említett

trajektória változatokon, különböző sebességekkel is. Ezen kívül a szabályzók robusztussági tulajdonságait vizsgáltam megváltoztatott tömegű és inerciájú quadcopter modelleken. A 3D vizualizációt V-REP illetve MATLAB-figure segítségével oldottam meg.

Introduction

There has already been a considerable interest in the use of small unmanned aerial vehicles (UAVs) for several applications including security, search and rescue, agricultural monitoring and transportation. Furthermore, quadrotor aircrafts are also quite popular among control researchers due to the simplicity of their dynamics and mechanical design. However, high cost and technical limitations kept the use of quadcopters relatively limited until the recent years.

The new trends in the industry, in connection with *Industry 4.0* [2, 3], can easily serve with new application fields for UAVs. The expression *Industry 4.0* denotes the current trend of automation and data exchange in the manufacturing industry. According to [3], *Industry 4.0* has three essential dimensions: ”(1) horizontal integration across the entire value creation network, (2) end-to-end engineering across the entire product life cycle, as well as (3) vertical integration and networked manufacturing systems”. These together allow for faster response to costumer needs and it also improves the speed, flexibility and quality of the production process [2]. Next to the previously mentioned advantages, [3] also sees the possibility of achieving more sustainable manufacturing processes with the help of this new industrial revolution. This would be possible with a more efficient allocation of resources on the basis of intelligent cross-linked value creation modules.

Autonomous manufacturing serving quadcopters can easily fit in the ideology of *Industry 4.0*. Of course, the range of application fields of the quadcopters is strongly limited by the short flight time that can be achieved. In this respect, designing a manufacturing process that relies on transportation carried out by quadcopters would be surrealistic. However, the flexibility and speed of these UAVs can be appealing for several other uses. Autonomous robots running on the ground are already used for transportation within manufacturing processes. Quadcopters could be used for example as a backup in the event of a sudden failure of one of these robots. The robot is disabled in the middle of the transportation process and hence the transported part can not reach its final destination, a following step in the manufacturing. In this case, the continuity of the whole manufacturing process is endangered. If such a situation appears, an autonomous quadcopter could be alarmed to pick up the part from the disabled robot and transport it to its destination. An other application possibility is to use autonomous UAVs for monitoring the manufacturing site from the air in certain time periods. Since both previously mentioned application fields take place in industrial environment, accurately and reliably executed flights are mandatory. The motivation for the controller design is to fulfill these strict requirements.

The controllers that have been designed for quadcopter UAVs spread in the range from simple PID controllers [4] through LQG [5] to even Learning Based Model Predictive Controller (LBMPC) solutions [6]. The simplest controls try only to stabilize the aircraft around a hovering position meanwhile other solutions, such as receding horizon algorithms, such as linear and nonlinear MPC [7] or LBMPC controls have spectacular capabilities for wide ranges of applications (Such as trajectory following or catching objects thrown to them.).

Trajectory generation for UAVs is also of major interest among researchers. Optimization based solutions for different requirements are introduced for example in [8]. In contrast to classical two level architectures, which consist of a trajectory designer and a trajectory follower control part, an other interesting approach is Model Predictive Contouring Control [9] where trajectory design and control is executed in one optimization step. This for example could be used in a field, where the UAV has a safe corridor in the air where it is allowed to fly. An exact trajectory generation would not be necessary in this case.

Although MPC based controls are very appealing due to their high performance, the application of them on quadcopters is not very common because of the very high computational requirements. State feedback control techniques are relatively easy to use and are much less computationally expensive. Hence, these techniques appear to be a good choice for the application field mentioned before.

I am going to introduce three different control solutions. All of these methods are based on linearizing the nonlinear plant of the quadcopter with different methods and than applying a state feedback controller for the linearized system. The paper also presents two different trajectory generation methods based on desired points in the 3D space. The performance of the controllers is than tested in simulation using a high fidelity model of the quadcopter. The simulations are carried out in the MATLAB-Simulink environment and 3D visualization is solved in V-REP.

In the first chapter, the dynamical model of the quadcopter and the actuators is introduced. The second chapter details the control methods. (1) First a successive input-output linearization method is shown, based on the article [1]. This utilizes a two level control architecture with Linear Quadratic Regulator (LQR) Control. (2) The second method is based on the Jacobi linearization of the model. The plant and the simulation is charged with noises as well and hence an LQG control is used. (3) The third approach applies exact feedback linearization on the model after some changes in the system variables. The linearized system is than controlled with LQR control. The third chapter details the trajectory design methods. Both of the solutions generate trajectories for desired points in the 3D space. The first one fits a spline curve and a sufficient velocity profile on the points. The velocity profile is designed so that the path would be carried out under the shortest time that is possible subject to plant constraints. The second trajectory type is a piecewise linear trajectory between the points with an additional requirement that is, that the quadcopter must finish the path under a predefined time. In the fourth chapter, the simulation results for the different setups are shown. These include simulations on different trajectories with different speeds also with changed weight and inertia in order to check the robustness properties of the controllers. In the fifth chapter, the 3D visualization is detailed briefly and then I make the conclusion. The Appendix details the simulation files.

Chapter 1

The dynamical model of the quadcopter

1.1 The equations of motion

The nonlinear dynamical model of the quadcopter is introduced in this section, based mostly on [1]. First of all we have to define the parent (global, external) and body (vehicle, quadcopter, attached) coordinate system. The definition of these frames, the Euler-angles, the individual rotor forces and the main thrust force can be seen on Figure 1.1, which is taken from [1].

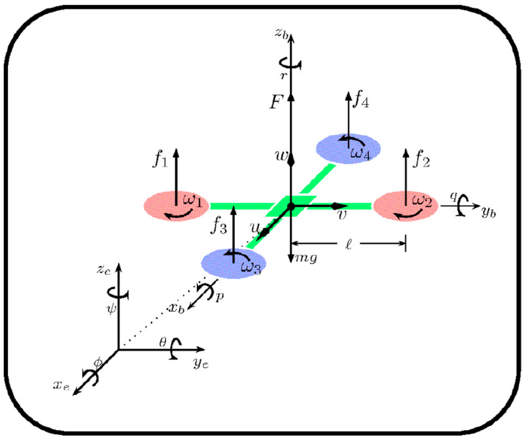


Figure 1.1: *Quadcopter configuration scheme, taken from [1]*

According to Figure 1.1, the position and orientation of the body coordinate system in the fixed (central) coordinate frame is given by ξ and η respectively.

$$\xi = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad \eta = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}$$

The dynamical model of the quadcopter can be obtained as it is in equations (1.1) and (1.2), from which the former one describes the translational and the latter one the angular dynamics.

$$m\ddot{\xi} = R_b^e \begin{bmatrix} 0 \\ 0 \\ F \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} \tag{1.1}$$

$$\tilde{J}\ddot{\eta} = \tau - C(\eta, \dot{\eta})\dot{\eta} \tag{1.2}$$

Where m is the mass of the drone, R_b^e is the rotation matrix between the body and the central frame, F is the thrust force generated by the four rotors together, $C(\eta, \dot{\eta}) = J\dot{W}_n$ is the Coriolis term, $\tau = [\tau_\phi, \tau_\theta, \tau_\psi]^T$ is the torque vector generated by the rotors in the body frame and $\tilde{J} = JW_n$ where J is the inetrial matrix. The previously mentioned R_b^e and W_n are as follows:

$$R_b^e = R_x R_y R_z = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.3}$$

$$W_n = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \cos\theta\sin\phi \\ 0 & -\sin\phi & \cos\theta\cos\phi \end{bmatrix} \tag{1.4}$$

Where W_n is the matrix that makes the transformation between Euler-angles and the angular velocity vector: $W_n\dot{\eta} = \Omega$. Here Ω is the angular velocity vector defined in the body coordinate system. This transformation is necessary since $\dot{\eta}$ is just the time derivative of the Euler angle representation in the fixed frame, meanwhile Ω is a vector defined in the body frame. The derivation of W_n can be found in [10]. Equation (1.2) can be deduced keeping in mind the previously defined connection between the angular velocity Ω and the Euler-angle derivative $\dot{\eta}$ and that the angular acceleration comes analytically from Euler's equation: $\tau = J\dot{\Omega} + \Omega \times J\Omega$.

An important property can be noticed by the expansion of (1.1) into (1.5: The yaw angle, ψ, does not appear in the equation of the translational motion. This allows us to generate a trajectory for ψ independently from the desired translational trajectory ξ_d.

$$\ddot{\xi} = \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \frac{1}{m} \begin{bmatrix} F\sin\theta \\ -F\cos\theta\sin\phi \\ F\cos\theta\cos\phi - mg \end{bmatrix} \tag{1.5}$$

Suppose each rotor produces $f_i \quad i = \{1, ..., 4\}$ thrust force. As it can be seen on figure (1.1), rotors 1 and 2 rotate clockwise while 3 and 4 rotate counterclockwise. This makes the quadcopter capable of keeping and changing its orientation (The role of the tail rotor of a helicopter). We assume all the 4 rotors to be identical. Hence, the torques and the thrust force produced by the rotors can be expressed as follows:

$$F = \sum_{1}^{4} f_i \tag{1.6}$$

$$\tau_\phi = (f_2 - f_1)l \tag{1.7}$$

$$\tau_\theta = (f_4 - f_3)l \tag{1.8}$$

$$\tau_\psi = \frac{b}{k}(-f_1 - f_2 + f_3 + f_4) \tag{1.9}$$

where l is the length of the arms of the quadcopter (figure 1.1), b is the drag force coefficient and k is the thrust force constant. (b and k are rotor parameters, and can be found in most of the catalogs.) The thrust forces produced by each rotors are proportional to the square of the rotor velocity with the proportionality factor k. By substituting $f_i = k\omega_i^2$, where ω_i is the rotor velocity of the i^{th} rotor, the following equations are obtained:

$$\begin{bmatrix} F \\ \tau \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -l & l & 0 & 0 \\ 0 & 0 & -l & l \\ -\frac{b}{k} & -\frac{b}{k} & \frac{b}{k} & \frac{b}{k} \end{bmatrix} \begin{bmatrix} k\omega_1^2 \\ k\omega_2^2 \\ k\omega_3^2 \\ k\omega_4^2 \end{bmatrix} \tag{1.10}$$

We can summarize the dynamical model of the quadcopter now. We can consider the rotor velocities (ω_i) as the inputs of the system. These inputs produce directly the thrust force and torques which act on the quadcopter in the manner introduced in equations (1.1) and (1.2). As the thrust force and the torques are just simply linear combinations of the squares of the rotor velocities, we can consider the $[F, \tau]^T$ vector as the input of the system, and we do so in the followings. (From this input vector the angular velocities of the rotors can be calculated easily based on equation (1.10).)

Through equations (1.1) and (1.2), the translational and angular accelerations of the quadcopter can be calculated and from them the remaining state variables of the quadcopter can be obtained by integration. These state variables are the translational velocity ($\dot{\xi}$), position (ξ), angular velocity ($\dot{\eta}$) and orientation (η).

In order to use this model, and hence to build a controller for it, the following parameters of the quadcopter need to be known: mass (m), inertia, (J), length of an arm (l) and the rotor constants (b and k).

1.2 Actuator model

The control algorithms in this document are derived by considering the thrust force and the torques ($[F, \tau]$), produced by the rotors, to be the inputs of the system. After the desired inputs for the quadcopter are calculated, the desired rotor velocities for them can be obtained based on equation (1.10). However, it has to be kept in mind that the prescribed rotor velocities generated by the controller can not be directly applied to the drone. The velocity of the rotor is generated by an electric motor which has a non zero moment of inertia and can not produce infinitely large torques. Therefore the maximal angular acceleration of the rotor is limited. The dynamical equation of the rotor is obtained as follows:

$$\dot{\omega} = \frac{1}{J_{act}}(T_{mot} - T_{drag}) \tag{1.11}$$

where:

$\dot{\omega}$:	The angular acceleration of the rotor.
$J_{act} = J_{motor} + J_{rotor}$:	The moment of inertia of the motor and the rotor together.
$T_{mot} \in [-T_{max}; T_{max}]$:	The motor torque
T_{drag} :	The so called drag torque, that is proportional with the square of the rotor velocity as follows:

$$T_{drag} = \frac{1}{2}\rho_{air} A v^2 = b\omega^2 \tag{1.12}$$

Where: ρ_{air} is the density of air, A is the frontal area of the rotor and $b = \frac{1}{2}\rho_{air} r^4 \pi$ is the drag constant. This is the same drag constant that is used in equations (1.9) and (1.10). (And it makes sense, because the torque that makes the drone rotate around the yawn angle ψ has to act on the motor counter-wise.) The derivation of the rotor dynamics and the origin of these rotor parameters can be found in details in [11].

The simplest way to make the rotor track a given ω_r velocity reference is to choose the motor torque as follows:

$$T_{mot} = J_{act}\dot{\omega}_r + b\omega_r^2 + J_{act}k_c(\omega_r - \omega)$$

Substituting it in (1.11) results in the following error dynamics:

$$\dot{\omega} - \dot{\omega}_r = -\frac{b}{J}(\omega + \omega_r)(\omega - \omega_r) + k_c(\omega_r - \omega)$$

It can be easily checked that this dynamics are asymptotically stable even if k=0, since $\omega + \omega_r > 0$ (The rotor can only rotate in one direction, and that is chosen to be the positive direction.). The gain k_c can be tuned to speed up the control loop.

It is assumed that the dynamics of the controlled actuator is much faster than the quadcopter's dynamics (The same assumption is taken for example in [[8]]). In this way, the actuator model is not taken into account during the design of the trajectory following controller. On the other hand, the actuator model is included in the simulation as a saturation in the quadcopter model:

$$T_{mot} = sat(J_{act}\dot{\omega}_r + b\omega_r^2 + J_{act}k_c(\omega_r - \omega)) \tag{1.13}$$

where *sat* stands for saturation ($T_{mot} \in [-T_{max}; T_{max}]$). Since the saturation of the actuator is included in the model on which the controllers are tested, the simulation results are more realistic.

As it was mentioned above, the actuator dynamics is not considered during the controller design. On the other hand, the finite maximal value of T_{mot} implies constraints on the maximal thrust force and the torques as well. The maximal velocity of a single rotor and hence the maximal thrust force it can produce can be obtained based on equation

(1.11),(1.12) and (1.13) as follows:

$$\omega_{max} = \sqrt{\frac{T_{max}}{b}} \qquad f_{max} = \frac{kT_{max}}{b}$$

This clearly means a limitation on the inputs, which should be taken into account during trajectory design. This is further detailed in section 4.

For the actuator model one should know the following properties of the motor and the rotor: The moment of inertia of the rotor and the motor ($J_{act} = J_{mot}+J_{rot}$)), the maximum torque of the motor (T_{max}) and the drag coefficient of the rotor (b).

Chapter 2

Control methods for trajectory tracking

2.1 Successive input/output linearization

The control design method has been taken from [1]. The algorithms is based on a successive linearization of the quadrotor dynamics, which results in a simple linear time invariant tracking error model. Then the error dynamics is stabilized by standard LQR feedback (detailed in for example [12, 13]). The synthesis procedure in [1] generates a two level control strategy, where the external loop controls the translational dynamics of the quadcopter and produces the input trajectory of orientation angles for the inner loop. The inner loop controls the orientation of the drone and produces desired rotor velocities for the Electronical Speed Control (ESC) that is responsible for the operation of the rotors. A very similar two level control strategy is used in [8] with differences in the orientation and hence in orientation error definitions.

The control design starts from the nonlinear model derived in section 2. Please recall that the control inputs are $[F, \tau]$ and in this design method we assume that state variables $\xi, \dot{\xi}, \eta$ and $e\dot{t}a$ are available for measurement. Based on [1], in the next two subsections, the design of the external and internal control loops are summarized. The block scheme of the two level control system can be seen on Figure 2.1.

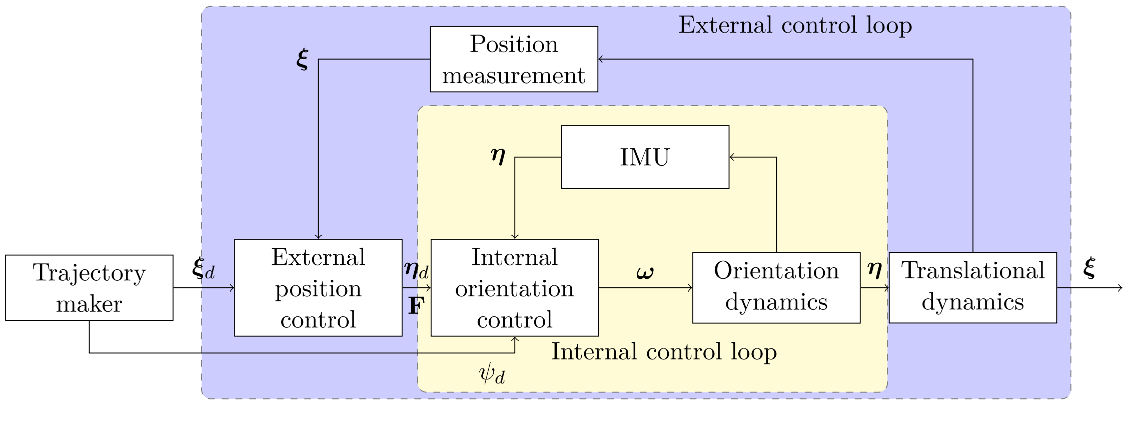


Figure 2.1: *The block scheme of the two level control system*

2.1.1 The external control loop

The altitude control is related to the dynamics of the coordinate z as it is expressed in equation (1.5):

$$m\ddot{z} = \cos\theta\cos\phi F - mg \tag{2.1}$$

In order to linearize (2.1), let F to be chosen as follows:

$$F = \frac{m(r_z + g)}{\cos\theta\cos\phi} \tag{2.2}$$

Where r_z is an auxiliary control input. By substituting F in equation (2.2) back into (2.1), the equation of motion in the z direction reduces to: $\ddot{z} = r_z$.

The motion in the direction y is governed by the following equation:

$$m\ddot{y} = -F\cos\theta\sin\phi \tag{2.3}$$

By substituting the previously expressed F into (2.3) the following equation is obtained:

$$m\ddot{y} = -(r_z + g)\tan\phi \tag{2.4}$$

Similarly to the previous case if ϕ is chosen as follows:

$$\phi = \arctan\left(-\frac{r_y}{r_z + g}\right) \tag{2.5}$$

where r_y is an auxiliary control input, then the nonlinear dynamical equation reduces to $\ddot{y} = r_y$. Finally, in the direction x, the governing equation is:

$$m\ddot{x} = F\sin\theta \tag{2.6}$$

Again, by substituting the expression for F in (2.2) and choosing θ to be as follows:

$$\theta = \arctan\left(\frac{r_x\cos\phi}{r_z + g}\right) \tag{2.7}$$

where r_x is an auxiliary input variable, the governing equation in the direction of x reduces to: $\ddot{x} = r_x$.

For the optimal state feedback control of the outer loop, it is necessary to define the position errors of the quadcopter:

$$\begin{aligned} e_x &= x - x_d \\ e_y &= y - y_d \\ e_z &= z - z_d \end{aligned} \tag{2.8}$$

where x, y and z are the actual position coordinates in the parent frame and x_d, y_d and z_d are the (at least) twice differentiable desired trajectory coordinates. By substituting the previously introduced auxiliary controls (r_x, r_y and r_z), the acceleration error (the second time derivative of the position error) is given by:

$$\begin{aligned} \ddot{e}_x &= r_x - \ddot{x}_d \\ \ddot{e}_y &= r_y - \ddot{y}_d \\ \ddot{e}_z &= r_z - \ddot{z}_d \end{aligned} \tag{2.9}$$

By defining new auxiliary inputs as: $\hat{r}_x = \ddot{e}_x = r_x - \ddot{x}_d$, $\hat{r}_y = \ddot{e}_x = r_y - \ddot{y}_d$ and $\hat{r}_z = \ddot{e}_x = r_z - \ddot{z}_d$, the state space equation of the errors becomes as follows:

$$\dot{e} = Ae + B\hat{r} \tag{2.10}$$

where: $e = [e_x, \dot{e}_x, e_y, \dot{e}_y, e_z, \dot{e}_z]^T$, $\hat{r} = [\hat{r}_x, \hat{r}_y, \hat{r}_z]^T$ and

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A system with these state space matrices is obviously controllable. We have also assumed at the beginning of the chapter, that ξ and $\dot{\xi}$ are available for measurement, which means that the states of the error system (2.10) are also measurable. Hence, it is possible to design an optimal state feedback control with LQR method. By this, the input of the error system of (2.10) is chosen to be as follows:

$$\hat{r} = -Ke \tag{2.11}$$

where K is the matrix that minimizes the following quadratic cost function:

$$J = \int_0^\infty (e^T Q e + \hat{r}^T R \hat{r}) dt \tag{2.12}$$

where Q and R weight matrices define what the purpose of the control is. If the values in Q are high compared to R, then the feedback gain matrix K will provide precise trajectory tracking (small tracking error) even for the price of large control inputs. In the other way around, the trajectory following will be less accurate but it will be less energy consuming. Q and R are:

$$Q = \begin{bmatrix} \xi_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \xi_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \xi_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & \xi_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & \xi_5 & 0 \\ 0 & 0 & 0 & 0 & 0 & \xi_5 \end{bmatrix} R = \begin{bmatrix} \mu_1 & 0 & 0 \\ 0 & \mu_2 & 0 \\ 0 & 0 & \mu_3 \end{bmatrix}$$

(ξ_1, ξ_3, ξ_5) are the costs of the position accuracy, (ξ_2, ξ_4, ξ_6) are the costs of the velocity accuracy and (μ_1, μ_2, μ_3) are the costs of the auxiliary inputs.

From (2.9) and (2.10) the auxiliary controls $r = [r_x, r_y, r_z]$ can be calculated as follows:

$$r = \ddot{\xi}_d - Ke \tag{2.13}$$

Once the auxiliary control vector r is calculated, equations (2.5), (2.7) and (2.8) can be solved in this particular order. And in this way the desired orientation and thrust force for the internal loop can be determined. As it is was previously discussed, and also detailed in [1], it is a matter of choice, how the third Euler-angle, ψ_d, is defined. [1] suggests for example to choose it so that a dedicated point of the quadcopter points in the direction of the tangent of the trajectory at every time. It is possible, however the method that the article suggests is not applicable in it self due to singularities in the definition. An individual and smooth enough trajectory for ψ_d can also be prescribed.

2.1.2 The internal control loop

The external control loop, introduced in the previous subsection, produces a desired thrust force (F_d) and a twice differentiable trajectory for the orientation angles ϕ_d and θ_d. (equations (2.5) (2.7 and (2.8)) As ψ_d does not appear in the governing equations of the external loop it can be defined independently. The desired trajectory ψ_d should be an at least twice continuously differentiable trajectory. This is a requirement, since the torque τ is the function of the angular acceleration as it can be seen from (1.2). The internal loop determines the input τ with that the prescribed η_d trajectory can be followed optimally.

Equation (1.2) can be reduced to $\ddot{\eta} = \tilde{\tau}$ by the following change of variable:

$$\tau = \tilde{J}\tilde{\tau} + C(\eta, \dot{\eta})\dot{\eta} \tag{2.14}$$

The structure of the internal control loop is quite similar to the external one. The angular error vector is defined (analogously to (2.8)), and an optimal state feedback is designed for the error state space (analogously to (2.9),(2.10 and (2.11)). The main point is, that the internal control loop has to be faster than the external one. To achieve this, [1] suggests the following method to define the internal loops feedback gain: (K is the external loop's and K_η is the internal loops feedback gain.)

$$K = \begin{bmatrix} k_{1x} & k_{2x} & 0 & 0 & 0 & 0 \\ 0 & 0 & k_{1y} & k_{2y} & 0 & 0 \\ 0 & 0 & 0 & 0 & k_{1z} & k_{2z} \end{bmatrix}$$

The settling times are defined after [[1]] as follows:

$$t_{ssx} = \frac{10}{k_{2x}}, \quad t_{ssy} = \frac{10}{k_{2y}}, \quad t_{ssz} = \frac{10}{k_{2z}}$$

With these, the components of K_η become:

$$\begin{gathered} k_{2\phi} = k_{2\theta} = k_{2\psi} = \frac{10}{\rho \min\{t_{ssx}, t_{ssy}, t_{ssz}\}} \\ k_{1\phi} = \sqrt{\frac{k_{2\phi}}{2}}, \quad k_{1\theta} = \sqrt{\frac{k_{2\theta}}{2}}, \quad k_{1\psi} = \sqrt{\frac{k_{2\psi}}{2}} \end{gathered} \tag{2.15}$$

And hence the feedback gain of the internal loop becomes as follows:

$$K = \begin{bmatrix} k_{1\phi} & k_{2\phi} & 0 & 0 & 0 & 0 \\ 0 & 0 & k_{1\theta} & k_{2\theta} & 0 & 0 \\ 0 & 0 & 0 & 0 & k_{1\psi} & k_{2\psi} \end{bmatrix} \tag{2.16}$$

Once the feedback gain is determined, the auxiliary control of the internal loop can be obtained:

$$\tilde{\tau} = \ddot{\eta}_d - K_\eta e_\eta \tag{2.17}$$

Where $\ddot{\eta}_d = [\ddot{\phi}_d, \ddot{\theta}_d, \ddot{\psi}_d]^T$ and $e_\eta = [e_\phi, \dot{e}_\phi, e_\theta, \dot{e}_\theta, e_\psi, \dot{e}_\psi]^T$. The auxiliary control $\tilde{\tau}$ of the internal control is in the role of r in the external control. The error system of the Euler angles is defined similarly to the error system of the position in the previous subsection. With $\tilde{\tau}$ The desired torques can be calculated based on (2.14). As F has already been calculated by the external control loop (equation(2.2)), the input vector $[F, \tau]$ can be produced.

2.1.3 The summary of the control method

1. Based on the position and velocity error, compared to the desired trajectory, the external loop produces the desired thrust force control input F and the desired orientation trajectory η_d for the internal loop.
   (a) composition of the position error vector: equation (2.8)
   (b) calculation of the auxiliary control ($\hat{r}$): equation (2.13)
   (c) calculation of ϕ_d, θ_d and F_d: equations (2.5), (2.7), (2.2)
   (d) definition of the third orientation angle trajectory ψ_d on demand.
2. The internal loop calculates the desired τ torques based on the orientation error.
   (a) composition of the orientation error vector: $e_\eta = [e_\phi, e_\theta, e_\psi, \dot{e}_\phi, \dot{e}_\theta, \dot{e}_\psi]^T$
   (b) calculation of the auxiliary control ($\tilde{\tau}$): equation (2.17)
   (c) calculation of the input torques (τ): equation(2.14)

   After the external and internal control loop, the desired thrust force and torque is given, (As it was detailed in the second section, these are the basic inputs of the quadrotor) and from these the rotor velocities that are producing these inputs can be calculated based on equation (2.18) which is derived from (1.10). These rotor velocities can be given to the ESC (Electronic Speed Control) of the motor.

$$\begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} = \frac{1}{k} \begin{bmatrix} 1 & 1 & 1 & 1 \\ -l & l & 0 & 0 \\ 0 & 0 & -l & l \\ -\frac{b}{k} & -\frac{b}{k} & \frac{b}{k} & \frac{b}{k} \end{bmatrix}^{-1} \begin{bmatrix} F \\ \tau \end{bmatrix} \tag{2.18}$$

2.2 Jacobi linearization and LQG control

The LQG (Linear Quadratic Gaussian) control theory is one of the most fundamental optimal control problems. It deals with linear systems with additive white Gaussian noise having incomplete state information. LQG control can be applied both to LTI and LTV systems. An LQG system basically consists of an LQE (Linear Quadratic Estimator, Kálmán-filter) and an LQR (Linear Quadratic Regulator). The separation principle ensures, that these two can be designed separately [12].

Jacobi linearization around an equilibrium point is introduced in [14]. LQG control is detailed in [12] and [13]. The discrete time Kálmán-filter and Extended Kálmán-filter is detailed in [15] and the continuous time derivation from the discrete time can be found in [16]. A solution for stabilizing a quadcopter in hovering position with LQG control is proposed in [5]. However, it is not further improved for trajectory tracking, I will introduced later.

The block scheme of the controller can be seen on Figure 2.2.

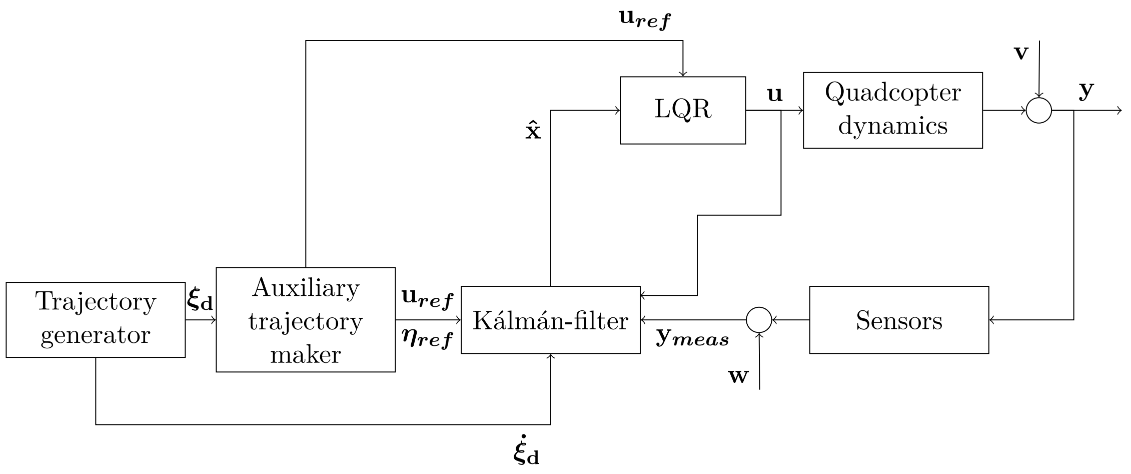

$\mathbf{u}_{\boldsymbol{ref}}$
$\mathbf{v}$
LQR
$\mathbf{u}$
Quadcopter dynamics
$\mathbf{y}$
$\hat{\mathbf{x}}$
Trajectory generator
$\boldsymbol{\xi}_\mathbf{d}$
Auxiliary trajectory maker
$\mathbf{u}_{\boldsymbol{ref}}$
$\boldsymbol{\eta}_{\boldsymbol{ref}}$
Kálmán-filter
$\mathbf{y}_{\boldsymbol{meas}}$
Sensors
$\mathbf{w}$
$\dot{\boldsymbol{\xi}}_\mathbf{d}$

Figure 2.2: *The block scheme of the LQG control*

2.2.1 Jacobi linearization

LQG control can be applied only to linear systems. The dynamical model of the quadcopter is a nonlinear one, which can be seen easily from the equations of the dynamics (1.1) and (1.2). So first of all, the system has to be linearized in the neighborhood of a trim point. Consider the following nonlinear differential equation:

$$\dot{x}(t) = F\left(x(t), u(t)\right) \tag{2.19}$$

Where $F(x, u)$ is the non linear function of the state vector x and the input vector u, and $\dot{x}$ is the time derivative of the state vector. x_0 and u_0 are called trim point values if they form an equilibrium of F, i.e.:

$$F(x_0, u_0) = 0 \tag{2.20}$$

Equation (2.20) implies that the trim point means, that state x_0 with an input u_0 is a stable state of the system. The small deviation variables from the trim point can be defined as follows:

$$\delta_x(t) = x(t) - x_0$$
$$\delta_u(t) = u(t) - u_0$$

Substituting these back in (2.19), the following equation is obtained:

$$\dot{\delta}_x(t) = F(x_0 + \delta_x(t), u_0 + \delta_u(t)) \tag{2.21}$$

This is exact and still nonlinear, but it can be expanded into Taylor-series:

$$\dot{\delta}_x \approx F(x_0, u_0) + \left.\frac{\partial F}{\partial x}\right|_{\substack{x(t)=x_0\\u(t)=u_0}} \delta_x + \left.\frac{\partial F}{\partial u}\right|_{\substack{x(t)=x_0\\u(t)=u_0}} \delta_u + H.O.T. \tag{2.22}$$

Where $H.O.T$ denotes the higher than 1st order terms, which will be neglected and due to (2.20), the following can be written:

$$\dot{\delta}_x \approx \left.\frac{\partial F}{\partial x}\right|_{\substack{x(t)=x_0\\u(t)=u_0}} \delta_x + \left.\frac{\partial F}{\partial u}\right|_{\substack{x(t)=x_0\\u(t)=u_0}} \delta_u \tag{2.23}$$

By using the notations

$$\left.\frac{\partial F}{\partial x}\right|_{\substack{x(t)=x_0\\u(t)=u_0}} =: A \qquad \left.\frac{\partial F}{\partial u}\right|_{\substack{x(t)=x_0\\u(t)=u_0}} =: B \tag{2.24}$$

the following linear state space system comes:

$$\begin{aligned}\dot{\delta}_x &= A\delta_x + B\delta_u\\ \delta_y &= C\delta_x + D\delta_u\end{aligned} \tag{2.25}$$

where $D = 0$ and $\delta_y = y(t) - y_0 = y(t) - Cx_0$.

This system is already linear and hence an LQG controller can be designed for it. To be able to complete the previously introduced linearization, the function $F(x(t), u(t))$ and the trim point values x_0 and u_0 have to be defined in the case of the quadcopter. The state and the control variables will be the followings:

$$x(t) = [\dot{\xi}, \eta, \dot{\eta}]^T = [\dot{x}, \dot{y}, \dot{z}, \phi, \theta, \psi, \dot{\phi}, \dot{\theta}, \dot{\psi}]^T \quad u(t) = [F, \tau]^T = [F, \tau_\phi, \tau_\theta, \tau_\psi]^T$$

With these state variables and based on equations (1.1) and (1.2) the nonlinear differential equation that describes the quadcopter is the following:

$$\begin{bmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{z}\\ \dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\\ \ddot{\phi}\\ \ddot{\theta}\\ \ddot{\psi}\end{bmatrix} = \begin{bmatrix}\frac{1}{m}F\sin\theta\\ -\frac{1}{m}F\cos\theta\sin\phi\\ \frac{1}{m}F\cos\phi\cos\theta - mg\\ \dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\\ a_1\\ a_2\\ a_3\end{bmatrix} \tag{2.26}$$

where:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \tilde{J}^{-1}(\tau - C(\eta, \dot{\eta})\dot{\eta})$$

A reasonable trim point choice for the quadcopter would be the stable hovering point, in which case the drone has no velocity, its orientation angles are zeros (horizontal position) and it does not have any angular velocities. For this hovering, the quadcopter does not need any torque input but it needs a thrust force that keeps the balance with the gravitational force. In this way the trim point state and input vectors become:

$$x_0 = [0,0,0,0,0,0,0,0,0]^T \qquad u_0 = [mg, 0, 0, 0]^T \tag{2.27}$$

With these trim point values and with differential equation (2.26) that describes the system, the Jacobian-matrices defined in (2.24) become as follows:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & g & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -g & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{1}{I_{xx}} & 0 & 0 \\ 0 & 0 & \frac{1}{I_{yy}} & 0 \\ 0 & 0 & 0 & \frac{1}{I_{zz}} \end{bmatrix} \tag{2.28}$$

Suppose we can measure the velocity and the orientation of the quadcopter, but not the angular velocity of it. This way the C matrix of the linear state space becomes:

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{2.29}$$

With equations (2.28) and (2.29) all the information is given to design an optimal feedback control system with LQR to the linear system (2.25).

2.2.2 Controllability and observability

In order to be able to design an LQG controller, the LTI system (2.25) has to be controllable and detectable. These two properties can be checked through the controllability (Co) and observability (O) matrices. If these matrices have the rank equal to the dimension of the state vector the terms are satisfied. These matrices for an n dimension state space have the following forms [12, 17]:

$$\begin{aligned}Co &= [B, AB, A^2B, ..., A^{n-1}B] \\ O &= [C, CA, CA^2, ..., CA^{n-1}]^T\end{aligned}$$

These matrices can be constructed in MATLAB with specified functions, and the result is that both of them have the rank 9, which is the dimension of the state vector, so the system is controllable and observable.

2.2.3 Kálmán-filter design (LQE)

In reality every system has noises, both in the process and in the sensors as well. In this case, a state estimator ([17]) is needed. There are several types of state estimators but the most commonly used type for linear systems is the well known Kálmán-filter [12, 15, 16]. The system with additional noises can be described with the following stochastic equations:

$$\begin{aligned}\dot{x}(t) &= Ax(t) + Bu(t) + v(t) \\ y(t) &= Cx(t) + w(t)\end{aligned} \tag{2.30}$$

where $v(t)$ and $w(t)$ are the process and measurement noises respectively. Since there is noise in the system, and we don't even know all the states from the measurements, a state observer has to be designed. The estimation process in continuous time is the following:

$$\dot{\hat{x}}(t) = A\hat{x}(t) + Bu(t) + L(y(t) - \hat{y}(t)) \tag{2.31}$$

where $\hat{x}$ is the estimated state, $\hat{y} = C\hat{x}$ is the estimated output, and L is the observer gain. The following variables for the error of the state and output estimations have to be introduced:

$$\begin{aligned}\Delta x &= x - \hat{x} \\ \Delta y &= y - \hat{y} = C\Delta x\end{aligned} \tag{2.32}$$

By substituting these back into (2.31) and also taking into account the noises from (2.30), the following dynamics for the errors can be obtained:

$$\Delta\dot{x} = (A - LC)\Delta x + v + Lw \tag{2.33}$$

For the use of the Kálmán-filter, the following assumptions need to be taken:

- stationary (the noise does not evolve with time)
- zero-mean
- Gaussian distribution
- white noise (the value of the noise is independent at any time from the values at any previous times)
- x_0, v and w are uncorrelated. (x_0 is the initial state.)

The process and the measurement noises have the following properties:

$$\begin{aligned}E\left\{v(t)v(\tau)^T\right\} &= V\delta(t-\tau) \\ E\left\{w(t)w(\tau)^T\right\} &= W\delta(t-\tau)\end{aligned} \tag{2.34}$$

Where V and W are the variance matrices of the noises, and $\delta(t-\tau)$ is the Kronecker-delta function, which is 1 if $t = \tau$ and 0 anywhere else. (In this respect (2.34) also implies that the noises are white noises.) We assume, that the noises on the different variables of the states and the outputs are not correlated to each other, in other words, V and W matrices are diagonal. V and W matrices are also positive definite, and we can assume that W is *strictly* positive definite, since there is no sensor without noise. We also use the following initial statements:

$$\begin{aligned} \bar{x}_0 &= E\left\{x(t_0)\right\} \\ P_0 &= E\left\{(x(t_0) - \bar{x}_0)\,(x(t_0) - \bar{x}_0)^T\right\} \end{aligned} \tag{2.35}$$

The following expressions formalize the assumption that was mentioned before, that the initial state uncertainty, the process noise and the measurement noise are independent from each other.

$$E\left\{v(t)w(t)^T\right\} = 0 \quad E\left\{w(t)x_0(t)^T\right\} = 0 \quad E\left\{x_0(t)v(t)^T\right\} = 0 \tag{2.36}$$

The LQE problem is to choose L to minimize the following expected (scalar) value:

$$E\left\{\Delta x(t)^T \Delta x(t)\right\} \tag{2.37}$$

For the solution of this, there is two more things that have to be fulfilled:

- (A, C) has to be observable. This was detailed in the previous section, that the system is observable after the linearization.
- (A, V) has to be stabilizable. Since (A, B) is controllable, and stabilizable in a not too big environment of the trim point, and V is bounded, it can be said, that (A, V) is stabilizable.

The derivation of the continuous Kálmán-filter can be found in details in [16]. The result of the derivation is the following for L:

$$L = \bar{P}C^T W^{-1} \tag{2.38}$$

where $\bar{P}$ is the solution of the following algebraic Ricatti-equation:

$$0 = \bar{P}A^T + A\bar{P} + V - \bar{P}C^T W^{-1} C\bar{P} \tag{2.39}$$

Luckily MATLAB has an *lqe* function, that calculates this Kálmán-filter gain, based on matrices A, C, V and W. It also waits for a transfer matrix that defines how the process noise influences the state, but we can assume it to be an identity matrix. (In this case the noises are simply added to the states.)

2.2.4 Trajectory tracking with the Jacoby-linearized model

Once ξ_d desired trajectory is defined, η_{ref}, $\dot{\eta}_{ref}$, $\ddot{\eta}_{ref}$ and F_{ref} trajectories can be defined instantly based on equation (1.5). (ψ_d trajectory can be defined independently). After η_{ref}, $\dot{\eta}_{ref}$ and $\ddot{\eta}_{ref}$ are given, τ_{ref} can be calculated based on (1.2). In this way, we can define reference angular and input trajectories for given ξ_d and ψ_d desired trajectories. This property of the dynamics is called the differential flatness and is further detailed in [8]. The meaning of differential flatness is that all the states and the inputs can be calculated from some carefully selected outputs, ξ and ψ in our case. This means, that based on the desired trajectories for these outputs (ξ_d, ψ_d), reference trajectories for all the states and the inputs can be obtained. I will not deduce this property analytically since I think it can be seen easily from equations (1.5) and (1.2). The calculation of the reference trajectories can be found in details in (3.10).

The model, introduced in subsection 3.2.1., in it self is only capable of stabilizing the quadcopter in a hovering position. To achieve the trajectory following feature, the states and inputs of the state space model defined in (2.25) have to be changed as follows:

$$\hat{\delta}_x = \begin{bmatrix} \dot{\xi} - v_{ref} \\ \eta - \eta_{ref} \\ \dot{\eta} - \dot{\eta}_{ref} \end{bmatrix} \quad \hat{\delta}_u = \begin{bmatrix} F - F_{ref} \\ \tau - \tau_{ref} \end{bmatrix} \tag{2.40}$$

Where:
$v_{ref} = \dot{\xi}_d + k(\xi_d - \xi)$: Reference velocity. The second part of the expression of v_{ref} has to be added to the first time derivative of the trajectory, since the position is not represented in the linear model, and without this term the position accuracy would not be guaranteed. k is a constant and can be set based on demand.

η_{ref}: The desired orientation trajectory that is exactly defined by ξ_d trajectory. ψ_d is independent again, and can be defined separately. For the definition of ϕ_{ref} and θ_{ref} see equation (3.10) which was obtained based on (1.5) and (1.2).

$\dot{\eta}_{ref}$: The first time derivative of the desired orientation trajectory η_{ref}.

F_{ref}: The reference thrust force trajectory which is exactly defined by ξ_d and η_{ref}. For the definition of F_{ref} see equation (3.10) which was obtained based on (1.5) and (1.2).

τ_{ref}: The reference torque trajectory that is exactly defined by η_{ref}. For the definition of τ_{ref} see equation (3.10) which was obtained based on (1.5) and (1.2).

We assume, that the quadcopter will not leave the close surrounding of the hovering state. In this way, we can use the previously introduced A and B state and K feedback gain matrices of the system. The important difference is that instead of δ_x and δ_u, we control the motion of the quadcopter based on the bias from the desired reference trajectories ($\hat{\delta}_x$ and $\hat{\delta}_u$). If the quadcopter does not leave the close neighborhood of the hovering point, the system is assumed to be linear. And if it is a linear system, then theoretically the correction input for a small deviation from the reference state trajectories will be the same as if the deviation would be compared to the hovering point. In this way, the control input $u = [F, \tau]$ to the quadcopter becomes:

$$u = u_{ref} - K\hat{\delta}_x \tag{2.41}$$

Where K is the LQR feedback gain matrix.

Equation (2.41) also means, that if we had an ideal system, with absolutely no state and measurement noises then δ_x would always be 0, and hence the controller would always give the quadcopter u_{ref} that naturally would drive it exactly on the desired trajectory.

That assumption we have made by letting the state space matrices to be the exact same as they are around the hovering trim point, lets us to design the control easily. On the other hand it means a limitation to its operation area. The quadcopter controlled in this way would not be able to follow a trajectory that requires big angular velocities or angles compared to the hovering position. These limitations have to be considered during the trajectory design.

The ξ position of the quadcopter can not be taken into the linear state space system, while a fixed position and a fixed velocity as trim point value would be impossible to satisfy. However, the position can be involved in the Kálmán-filter's state space system, it only means an extra integrator in the system. In this way, it is possible to estimate the position of the quadcopter as well which is necessary for the generation of v_{ref}. It is also important that due to the definition of v_{ref}:

$$\int \dot{\xi} - v_{ref} dt \neq \xi - \xi_d$$

Due to this fact, the Kálmán-filter's estimating equation in (2.31) can be obtained for this application with the states and state space matrices as follows:

$$\dot{\tilde{\delta}}_x = \tilde{A}\tilde{\delta}_x + \tilde{B}\tilde{\delta}_u + \tilde{L}(\tilde{\delta}_y - \tilde{\delta}_y) \tag{2.42}$$

Where

$$\tilde{\delta}_x = \begin{bmatrix} \xi_{est} - \xi_d \\ \dot{\xi}_{est} - \dot{\xi}_d \\ \eta_{est} - \eta_d \\ \dot{\eta}_{est} - \dot{\eta}_d \end{bmatrix} \quad \tilde{\delta}_y = \begin{bmatrix} \xi_{meas} - \xi_d \\ \dot{\xi}_{meas} - \dot{\xi}_d \\ \eta_{meas} - \eta_d \end{bmatrix} \quad \tilde{\delta}_y = \begin{bmatrix} \xi_{est} - \xi_d \\ \dot{\xi}_{est} - \dot{\xi}_d \\ \eta_{est} - \eta_d \end{bmatrix} \quad \tilde{\delta}_u = \begin{bmatrix} F - F_{ref} \\ \tau - \tau_{ref} \end{bmatrix}$$

$\tilde{\delta}_x$ is the state vector of the filter, which is not identical to the state vector of the controlled system $\hat{\delta}_x$. (It contains $\dot{\xi}_d$ instead of v_{ref})
$\tilde{A}$ contains an extra line of integrators compared to A in (2.28)
$\tilde{B}$ contains an extra line of zeros compared to B in (2.28)
$\tilde{L}$ is the estimating feedback gain of the filter designed for this system.

It can be seen, that in order to be able to include an extra integrator in the system and hence to measure the position as well we have to use $\dot{\xi}_d$ instead of v_{ref}. So the states of the Kálmán-Filter differ from the states of the controlled system. To get back the states of the controlled system as they are in equation (2.40), the following equation can be obtained:

$$\hat{\delta}_x = T\tilde{\delta}_x$$

Where:

$$T = \begin{bmatrix} -k \cdot I_{3\times3} & I_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & I_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & I_{3\times3} \end{bmatrix}$$

Where $I_{3\times3}$ and $0_{3\times3}$ denote the 3×3 dimension identity and zero matrix respectively.

2.2.5 The summary of the control method

Based on the desired trajectories, ξ_d and ψ_d, reference trajectories for the angles and the control inputs are generated (η_{ref}, u_{ref}). In a not too big neighborhood of the hovering trim point, the quadcopter is assumed to be a linear system, for which an LQR control can be obtained. The difference of the state variables from the reference values through the feedback gain generates the difference input. By adding this difference input to the reference input, we get the optimal control input for the quadcopter. During the process, the states are estimated by a Kálmán-filter which works with slightly different states in order to be able to estimate the position of the quadcopter as well.

2.3 Feedback linearization

In this section, the exact feedback linearization control method will be introduced. At first the control will be summarized for SISO systems, then the extension of it to trajectory following MIMO systems and the adoption of it for the quadcopter will be detailed.

2.3.1 Feedback linearization of a SISO system

In this subsection, I am only going to introduce the feedback linearization method briefly and going to concentrate on its implementation method for trajectory tracking application. The detailed theory and deduction of feedback linearization can be found in [18] and [19].

For the sake of simplicity, I am going to introduce the feedback linearization of a nonlinear SISO system first. The method described in this way can be extended to MIMO systems easily with small changes. Suppose we have the following nonlinear SISO system which is affine in the input, that is, a system of the form:

$$\begin{aligned} \dot{x} &= f(x) + g(x)u \\ y &= h(x) \end{aligned} \tag{2.43}$$

where $x \in \mathbb{R}^n, u \in \mathbb{R}$ and $y \in \mathbb{R}$. The vector fields $f(x) : \mathcal{D} \longmapsto \mathbb{R}^n$ and $g(x) : \mathcal{D} \longmapsto \mathbb{R}^n$ and the function $h(x) : \mathcal{D} \longmapsto \mathbb{R}$ are assumed to be smooth in the domain $\mathcal{D} \subset \mathbb{R}^n$, that is, their partial derivatives respect to x of any order exist and are continuous on $\mathcal{D}$.

It is also assumed that all states are available for measurement.

The objective of feedback linearization is to find a smooth control law in the form:

$$u = \alpha(x) + \beta(x)v \tag{2.44}$$

that transforms the map between the new input v and output y into a linear time invariant one.

In order to construct the linearizing feedback, the notion of Lie-derivative and relative degree have to be introduced:

- $f(x)$ vector field is a smooth mapping on $\mathbb{R}^n$. It consists of smooth functions of $x \in \mathbb{R}^n$:

$$f(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{bmatrix} \tag{2.45}$$

- given a vector field $f(x) : \mathbb{R}^n \longmapsto \mathbb{R}^n$ in the form of (2.45) and a function $\lambda : \mathbb{R}^n \longmapsto \mathbb{R}$. The derivative of $\lambda(x)$ along the vector field $f(x)$ is called the Lie-derivative of $\lambda(x)$ and is denoted by:

$$L_f\lambda(x) = \frac{\partial \lambda}{\partial x} f(x) \tag{2.46}$$

- The higher order derivatives can be calculated recursively:

$$L_f^k\lambda(x) = \frac{\partial L_f^{k-1}\lambda(x)}{\partial x} f(x)$$

An other important definition that has to be made is the notion of **relative degree** of a system. The system of the form of (2.43) has a relative degree of r if:

$$\begin{aligned}
&1.: \quad L_g h(x) = 0 \\
&2.: \quad L_g L_f^i h(x) = 0 \qquad \text{for all} \quad i < r-1 \\
&3.: \quad L_g L_f^{r-1} h(x) \neq 0
\end{aligned} \tag{2.47}$$

This definition practically means that the output of the system has to be differentiated r times, before the input first appears explicitly.

In this way, suppose, the relative degree of a system of the form (2.43) is $r > 1$. Then:

$$\begin{aligned}
\dot{y} &= \frac{dy}{dt} = \frac{dh(x)}{dt} = \frac{\partial h}{\partial x}\dot{x} = \frac{\partial h}{\partial x} f(x) + \frac{\partial h}{\partial x} g(x)u = L_f h(x) + L_g h(x) = L_f h(x) \\
\ddot{y} &= \frac{d^2y}{d^2t} = \frac{dL_f h(x)}{dt} = \frac{\partial L_f h(x) h}{\partial x}\dot{x} = L_f^2 h(x) + L_g L_f h(x) = L_f^2 h(x) \\
&\vdots \\
y^{(r)} &= \frac{d^r y}{d^r t} = \frac{dL_f^{r-1}h(x)}{dt} = \frac{\partial L_f^{r-1} h(x) h}{\partial x}\dot{x} = L_f^r h(x) + L_g L_f^{r-1} h(x)
\end{aligned} \tag{2.48}$$

Suppose, a system has a relative degree of n. Then by differentiating the output n times, the following set of equations is obtained:

$$\begin{aligned}
y &= h(x) = z_1 \\
\dot{z}_1 &= \dot{y} = L_f h(x) = z_2 \\
\dot{z}_2 &= \ddot{y} = L_f^2 h(x) = z_3 \\
&\vdots \\
\dot{z}_n &= y^{(n)} = L_f^n h(x) + L_g L_f^{n-1} h(x) u
\end{aligned} \tag{2.49}$$

By choosing the input u to be as follows:

$$u = \frac{1}{L_g L_f^{n-1} h(x)} \left[v - L_f^n h(x) \right] \tag{2.50}$$

equation (2.49) can be transformed into a chain of integrators:

$$\begin{aligned}
y &= z_1 \\
\dot{z}_1 &= z_2 \\
\dot{z}_2 &= z_3 \\
&\vdots \\
\dot{z}_n &= v
\end{aligned} \tag{2.51}$$

Due to equation (2.51), the realization of the system would become:

$$A = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ & & & \ddots & \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \qquad C = [1 \quad 0 \quad \dots 0] \tag{2.52}$$

A system with these state space matrices is obviously controllable.

Since $L_g L_f^{n-1} h(x) \neq 0$ because of the definition of the relative degree of a system and that we assumed it to be equal to n, equation (2.50) always has a solution. In this way the exact feedback linearization is complete and the functions of (2.44) can be obtained as:

$$\begin{aligned} \alpha(x) &= \frac{-L_f^n h(x)}{L_g L_f^{n-1} h(x)} \\ \beta(x) &= \frac{1}{L_g L_f^{n-1} h(x)} \end{aligned} \tag{2.53}$$

2.3.2 Extension of the control method to trajectory tracking MIMO systems

Suppose we have a system of the following form:

$$\begin{aligned} \dot{x} &= f(x) + g(x)u \\ y &= h(x) \end{aligned} \tag{2.54}$$

where $x \in \mathbb{R}^n, u \in \mathbb{R}^p$ and $y =\in \mathbb{R}^p$. The vector fields are $f(x) : \mathcal{D} \longmapsto \mathbb{R}^n$ and $h(x) : \mathcal{D} \longmapsto \mathbb{R}^p$ and the matrix $g(x) : \mathcal{D} \longmapsto \mathbb{R}^{n \times p}$.

For MIMO systems, the **vector relative degree** of a system has to be defined. Based on the definition of relative degree for SISO systems: The system has a vector relative degree of $\{r_1, r_2, ..., r_p\}$ if the i$^{\text{th}}$ output of the system has to be differentiated r_i times for a control input to appear explicitly.

If the system has a vector relative degree $\{r_1, r_2, \dots, r_p\}$ then the following equation can be obtained:

$$\begin{bmatrix} y_1^{(r_1)} \\ y_2^{(r_2)} \\ \vdots \\ y_p^{(r_p)} \end{bmatrix} = F(x) + G(x)u \tag{2.55}$$

If $G(x)$ is nonsingular then u can be chosen as follows:

$$u = G^{-1}(x)(v - F(x)) \tag{2.56}$$

Necessary conditions for the non-singularity of $G(x)$:

- $\sum r_i = n$
- $G(x)$ is a $p \times p$ square matrix. This is why the dimensions of y and u in the system (2.54) both have to have the dimension p.

Similarly to the SISO case, equation (2.56) transforms the nonlinear system of equations (MIMO version of (2.50)) into a chain of integrators as follows (MIMO version of (2.51):

$$\begin{array}{ccccc} y_1 = z_{11} & y_2 = z_{21} & & y_p = z_{p1} \\ \dot{z}_{11} = z_{12} & \dot{z}_{21} = z_{22} & & \dot{z}_{p1} = z_{p2} \\ \vdots & \vdots & \dots & \vdots \\ \dot{z}_{1r_1} = v_1 & \dot{z}_{2r_2} = v_2 & & \dot{z}_{pr_p} = v_p \end{array} \tag{2.57}$$

For details and derivation of the exact feedback linearization see [18] and [19].

2.3.3 Trajectory tracking with the feedback linearized model

Since the nonlinear dynamics of the quadcopter has been transformed into a chain of integrators, and hence it became a linear system, the trajectory tracking capability can be achieved easily. For this, the following choice of the v auxiliary input variable is necessary:

$$v = \begin{bmatrix} y_{1d}^{(r_1)} \\ y_{2d}^{(r_2)} \\ \vdots \\ y_{pd}^{(r_p)} \end{bmatrix} + Ke \tag{2.58}$$

Where $y_{id}^{(r_i)}$ is the r_ith derivative of the ith desired output variable and e is the error vector which has the form of:

$$e = [e_1^{(1)}, \dots, e_p^{(1)}, e_1^{(2)}, \dots, e_p^{(2)}, \quad \dots \quad, e_1^{(r_1-1)}, \dots, e_p^{(r_p-1)}]^T \tag{2.59}$$

where $e_i^{(j)} = y_{id}^{(j)} - y_i^{(j)}$.

Substituting the input choice of equation (2.58) into the system (2.55) the following dynamics for the errors comes:

$$\begin{bmatrix} y_{1d}^{(r_1)} - y_1^{(r_1)} \\ y_{2d}^{(r_2)} - y_2^{(r_2)} \\ \vdots \\ y_{pd}^{(r_p)} - y_p^{(r_p)} \end{bmatrix} = \begin{bmatrix} e_1^{(r_1)} \\ e_2^{(r_2)} \\ \vdots \\ e_p^{(r_p)} \end{bmatrix} = -Ke \tag{2.60}$$

Which can be stabilized with an appropriate choice of the feedback gain K. (In the case of the quadcopter control, K is determined with LQR method for the feedback linearized error system of (2.63).)

With this choice of auxiliary input v, the real inputs of the system become as follows:

$$u = G^{-1}(x) \left[\begin{bmatrix} y_{1d}^{(r_1)} \\ y_{2d}^{(r_2)} \\ \vdots \\ y_{pd}^{(r_p)} \end{bmatrix} - F(x) + Ke \right] \tag{2.61}$$

2.3.4 Implementation of the method on the quadcopter

I choose the output variables to be the position and the yaw angle. $y = [\xi, \psi]^T$ (This is a reasonable choice, since trajectory can be prescribed for these states.) The input variables are the ones, that were previously: $u = [F, \tau]^T$. In this way, the assumption, that is that the dimensions of the input and the output have to match is fulfilled.

The state space ($[\xi, \dot{\xi}, \eta, \dot{\eta}]^T$) has the dimension of 12. According to equations (1.1) and (1.2), both the position variables and the yaw angle have to be differentiated 2-2 times until an input variable appears explicitly. In this way, the (vector) relative degree of the system in this form is 8, which is less then the dimension of the state space. We have two choice: we either find 4 more functions for the coordinate transformation (This is a common solution when the sum of the vector relative degree is less than the dimension of the state space. It is detailed in [18]), or we make some variable changes, with which $\sum r_i = n$ holds.

The second option turns out to be the easier solution. The dynamics of the quadcopter in the x, y, z direction can be obtained after (1.1) as follows:

$$\begin{aligned} \ddot{x} &= \frac{1}{m} F \sin\theta \\ \ddot{y} &= -\frac{1}{m} F \cos\theta \sin\phi \\ \ddot{z} &= \frac{1}{m} F \cos\theta \cos\phi - g \end{aligned} \tag{2.62}$$

It can be seen that if instead of F we choose $u_f = \ddot{F}$ to be the input variable of the system then the vector relative degree of the system becomes 14. (As we have to differentiate the 3 translational dynamics equations of (2.62) four-four times to reach the new control variable u_f, and ψ_d has to be differentiated 2 times to reach τ.) Since the new control variable is the second derivative of the thrust force, two more integrators appear in the system, and hence the number of the states also becomes $n = 14$. In this way,the necessary criteria of exact feedback linearization is reached. ($\sum r_i = n$)

In addition, this solution implies that the prescribed trajectory has to be at least 4 times differentiable in time. Luckily, as it is detailed at the beginning of section 4, the four times differentiable trajectory prescription is a requirement for all controls anyways (because of the dynamical properties of the quadcopter), so this change in the variables does not mean an extra boundary for the trajectory design.

To make the functions that describe the system simpler and hence the differentiations easier, it is also reasonable to choose $\ddot{\eta}$ to be the new input instead of τ. This does not change the dimension of the state space nor the vector relative degree of the system, it just makes calculations easier. After all these changes in the variables, the new input vector of the system is: $u_{new} = [u_f, \ddot{\eta}]^T$

With these changes in the input and with the method described in subsection 3.3.2., the error system for trajectory tracking can be obtained as follows:

$$\dot{e} = Ae + Bv \tag{2.63}$$

Where $e = [e_\xi, e_\xi^{(1)}, e_\xi^{(2)}, e_\psi, e_\xi^{(3)}, e_\psi^{(1)}]^T$ is the error vector, where $e_\xi^{(i)} = \xi_d^{(i)} - \xi^{(i)}$ and

$e_{\psi}^{(i)} = \psi_d^{(i)} - \psi^{(i)}$. The matrices of the system are:

$$A = \begin{bmatrix} 0_{3\times3} & I_{3\times3} & 0_{3\times3} & 0_{3\times1} & 0_{3\times3} & 0_{3\times1} \\ 0_{3\times3} & 0_{3\times3} & I_{3\times3} & 0_{3\times1} & 0_{3\times3} & 0_{3\times1} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times1} & I_{3\times3} & 0_{3\times1} \\ 0_{1\times3} & 0_{1\times3} & 0_{1\times3} & 0_{1\times1} & 0_{1\times3} & 1_{1\times1} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times1} & 0_{3\times3} & 0_{3\times1} \\ 0_{1\times3} & 0_{1\times3} & 0_{1\times3} & 0_{1\times1} & 0_{1\times3} & 0_{1\times1} \end{bmatrix} \qquad B = \begin{bmatrix} 0_{10\times4} \\ I_{4\times4} \end{bmatrix} \tag{2.64}$$

Where $0_{n\times m}$ denotes an $n \times m$ dimension zero matrix and $I_{n\times n}$ means an $n \times n$ dimension identity matrix.

For the error system defined in (2.63), an LQR control can be designed, and hence the auxiliary control variable v can be calculated through static state feedback. The new, changed control variables ($u_{new} = [u_f, \ddot{\eta}]^T$) can be calculated based on equation (2.56) as follows:

$$u_{new} = \begin{bmatrix} u_f \\ \ddot{\eta} \end{bmatrix} = G^{-1}(x) \left[\begin{bmatrix} \xi_d^{(4)} \\ \psi_d^{(2)} \end{bmatrix} - F(x) + Ke \right] \tag{2.65}$$

$F(x)$ and $G(x)$ can be reached through differentiating the expressions for the translational accelerations in (2.62) two more times. (Since in the new control variable choice $\ddot{\psi}$ is a control variable it self, the last line of $F(x)$ consists of zeros, and the last line of $G(x)$ becomes easily $[0, 0, 0, 1]$.) After u_{new} is calculated, the real inputs for the system, $u = [F, \tau]^T$, can be found with the help of the definition of u_f ($F = \iint u_f dt$) and equation (1.2). ($\tau = \tilde{J}\ddot{\eta} - C(\eta, \dot{\eta})\dot{\eta}$)

One more thing that needs to be checked is that at which occasions becomes $G(x)$ singular. After looking at $G(x)$ (which can be done by running **symbolic_differentiation.m**) script file), it can be seen, that singularity occurs when the orientation angles ϕ and θ and also the thrust force of the motor F become zero at the same time. If we make sure, that this scenario never happens then exact feedback linearization can be used successfully for the control of the quadcopter.

However, the control method brings major difficulties with itself, which origins in the multiple times differentiation. First of all, it is not realistic that we can measure all the states and their derivatives, that would be required for the control. In this case we would probably need a state estimator, for example an Extended Kálmán-filter [15]. Second of all, Producing the trajectory and their higher order derivatives turns out to be not very easy for trajectories of the kind that will be introduced in the next section.

2.3.5 The summary of the control method

The e error vector can be built based on the desired and the real trajectories. After that, the auxiliary input v can be calculated through the feedback gain: (equation (2.58)). Based on equation (2.61) and v the new input variables can be found, and from them, based on the definition of u_f and equation (1.2) the real inputs of the system can be calculated.

Chapter 3

Trajectory design for the quadcopter

The quadcopter gets predefined points in the 3D space, which it should interlace with a 4 times differentiable trajectory. (The order of differentiability depends on the prescribed velocity profile which will be detailed later in subsections 3.1.1 and 3.2)

The four times differentiable trajectory is necessary because of the followings: Based on equation (1.5), the orientation angles (η) can be expressed as functions of the second derivative of the trajectory ($\ddot{\xi}$). Moreover, equation (1.2) implies that the torque is the function of the orientation, its first and its second derivatives. Finally the rotor velocities are functions of the torque. From these it can be seen, that the rotor velocities are functions of the fourth derivative of the desired translational trajectory,ξ_d. In this way a four times differentiable trajectory prescription is essential in order to achieve a continuous rotor velocity function.

This paper introduces two solutions for the trajectory prescribing task. In the first case a natural spline is fitted on the points. The second is a piecewise linear trajectory, where the points are simply connected with straight lines. The higher order continuity is achieved by the definition of the velocity profile in both cases.

3.1 Spline trajectory

The curve trajectory definition is produced in two steps. First, a spline curve is fitted on the desired points in the space. Second, a velocity profile is defined, which respects the constraints of the quadcopter.

Given k points $\xi_i = [x_i, y_i, z_i]^T$ in the 3D space. We fit a third order spline (a two times differentiable spline), $sp(\rho)$, on these points. The fitting is done so, that we define $k-1$ third order polynomials, one for each $[\xi_i; \xi_{i+1}]$ interval, which satisfy 3 boundary conditions at their endpoints(zeroth, first and second derivative continuity). These polynomials are denoted as $q_i(\rho)$, where $\rho \in [\rho_i; \rho_{i+1}] \quad i = 1, 2...(k-1)$. Here ρ_i and ρ_{i+1} are the boundaries of the parameter on the i^{th} polynomial. The previously mentioned properties and the boundary conditions can be expressed as follows:

$$\begin{aligned} q_i(\rho) =& A_i(\rho-\rho_i)^3 + B_i(\rho-\rho_i)^2 + C_i(\rho-\rho_i) + D_i && (3.1)\\ q_i(\rho_{i+1}) =& q_{i+1}(\rho_{i+1}) = \xi_{i+1} && (3.2)\\ \frac{d}{d\rho}q_i(\rho_{i+1}) =& \frac{d}{d\rho}q_{i+1}(\rho_{i+1}) && (3.3)\\ \frac{d^2}{d^2\rho}q_i(\rho_{i+1}) =& \frac{d^2}{d^2\rho}q_{i+1}(\rho_{i+1}) && (3.4) \end{aligned}$$

Please note that equation (3.2) is basically 2 equations, so we have 4 equations ((3.2),(3.3) and (3.4)) for the 4 unknown constant coefficients A_i, B_i, C_i and D_i that determines the i^{th} component of the spline.

The problem of the trajectory definition with a spline is that the curve is not the function of time nor distance, but the function of a parameter ρ. In order to evaluate the spline in time, the function between time and ρ has to be defined. It is solved in the following way: With a fine grid, the spline is evaluated in many points between ρ_{min} and ρ_{max}. The arc length between the neighboring points is approximated with a straight line. It is an acceptable approximation if the grid is fine enough. In this way, a function between the arc length $\tilde{\rho}$ and the parameter ρ can be achieved. The function between $\tilde{\rho}$ (arc length) and the parameter ρ is constructed by an other spline. Because of the very fine grid, the nonlinearity between the new parameter of the new spline and $\tilde{\rho}$ is negligible. in this way the function $f(\tilde{\rho}_p) = \rho$ can be obtained. So, if p is a point on the spline and the arc length $\tilde{\rho}_p$ between p and the start point of the spline is known, the parameter of this point, ρ_p, can be calculated. In this way, if we know the covered distance on the spline, the parameter (ρ) of that point can be calculated. In this way, suppose the quadcopter's velocity follows $v(t)$ function in time. After all this, the point of the desired trajectory spline at a time instance t_i can be determined as follows:

$$\begin{aligned} \tilde{\rho}(t_i) =& \int_{t_0}^{t_i} v(t)dt && (3.5)\\ \rho(t_i) =& f(\tilde{\rho}(t_i)) && (3.6)\\ \xi_d(t_i) =& sp(\rho_i) = sp(f(\tilde{\rho}(t_i))) && (3.7) \end{aligned}$$

The time derivatives of the trajectory (with the notation $sp(t) = \xi_d(t)$):

$$\begin{aligned} \dot{\xi}_d =& \frac{d\xi_{ref}}{dt} = \frac{\partial\xi_{ref}}{\partial\rho}\frac{\partial\rho}{\partial\tilde{\rho}}\frac{\partial\tilde{\rho}}{\partial t} = \frac{\partial\xi_{ref}}{\partial\rho}\frac{\partial\rho}{\partial\tilde{\rho}}v(t) && (3.8)\\ \ddot{\xi}_d =& \frac{d^2\xi_{ref}}{d^2t} = \frac{\partial^2\xi_{ref}}{\partial^2\rho}\left(\frac{\partial\rho}{\partial\tilde{\rho}}v(t)\right)^2 + \frac{\partial\xi_d}{\partial\rho}\left(\frac{\partial^2\rho}{\partial^2\tilde{\rho}}v(t)^2 + \frac{\partial\rho}{\partial\tilde{\rho}}a(t)\right) && (3.9) \end{aligned}$$

($v(t)$ and $a(t)$ are scalar functions and are the velocity and acceleration on the trajectory)

3.1.1 Constructing the velocity profile for the spline

Assume the spatial spline trajectory, i.e. (3.1) and (3.8) is given. To construct the velocity profile $v(t)$ it has to be kept in mind that the motor and rotor properties imply maximal values for the $[F, \tau]$ inputs (as it was introduced in subsection 2.2.1.). It is also reasonable to set constraints on the orientation angles (θ_{max}, ϕ_{max}).

The quadcopter will accelerate to a constant velocity at the beginning of the path, keep this constant speed and then decelerate at the end of the path to zero. This constant velocity should be set such that the previously mentioned input and orientation angle constraints would not be violated. For this, a maximal constant velocity v_{max} is computed which defines a feasible trajectory. This maximal constant velocity can be calculated with a non-linear constraint optimization problem.

For this, suppose that the desired trajectory ξ_d be given. If ξ_d is given, the corresponding reference angular trajectory η_{ref} and input trajectory u_{ref} can be determined from the state space model (1.5) and (1.2) as follows:

$$\begin{aligned}
\phi_{ref} &= f_\phi(\ddot{\xi}_d) = arctan\left(-\frac{\ddot{y}}{\ddot{z}+g}\right)\\
\theta_{ref} &= f_\theta(\ddot{\xi}_{ref}, \phi_{ref}) = arctan\left(-\frac{\ddot{x}\cos\phi_{ref}}{\ddot{z}+g}\right)\\
F_{ref} &= f_F(\ddot{\xi}_d, \phi_{ref}, \theta_{ref}) = \frac{m(\ddot{z}+g)}{\cos\theta\cos\phi}\\
\eta_{ref} &= \begin{bmatrix} f_\phi(\ddot{\xi}_d)\\ f_\theta(\ddot{\xi}_d, \phi_{ref})\\ \psi_{ref}\end{bmatrix} \quad \dot{\eta}_{ref} = \begin{bmatrix} \dot{f}_\phi(\ddot{\xi}_d)\\ \dot{f}_\theta(\ddot{\xi}_d, \phi_{ref})\\ \dot{\psi}_{ref}\end{bmatrix} \quad \ddot{\eta}_{ref} = \begin{bmatrix} \ddot{f}_\phi(\ddot{\xi}_d)\\ \ddot{f}_\theta(\ddot{\xi}_d, \phi_{ref})\\ \ddot{\psi}_{ref}\end{bmatrix}\\
\tau_{ref} &= \tilde{J}\ddot{\eta}_{ref} + C(\eta_{ref}, \dot{\eta}_{ref})\dot{\eta}_{ref}\\
u_{ref} &= [F_{ref}, \tau_{ref}]^T
\end{aligned} \tag{3.10}$$

The aim is to maximize v such that the state and input constraints are not violated. Formally this can be formulated as follows:

$$v_{max} = max\left\{v\right\} \quad \text{s.t.:} \quad u_{ref} \in [u_{min}; u_{max}] \quad \text{and} \quad \eta_{ref} = [\eta_{min}; \eta_{max}]$$

If v velocity and $sp(\rho)$ spline are given then the ξ_d trajectory can be calculated based on equations (3.5),(3.6) and (3.7). As it can be seen from (3.10), this desired trajectory (ξ_d) defines exactly the orientation and the input trajectories. Hence η_{ref} and $u_{ref}[F_{ref}, \tau_{ref}]$ can also be obtained for a given v and $sp(\rho)$. In this respect, the iterative optimization of v goes so, that v is risen in small steps from 0, and for each v, the maximal reference orientation angle and input is calculated. This process goes on until one of the constraints are violated. The last feasible v is kept and is chosen to be the constant velocity along the spiral trajectory v_{max}.

The other thing that needs to be set is the function that defines the acceleration. The quadcopter based on its dynamics has several requirements for such a function. Since we know that the velocity in the middle of the path is constant ($v(t) = v$), the velocity function at acceleration and deceleration has to satisfy the following boundary conditions. (t_s is the duration of the acceleration. in other words, the acceleration function is valid only if $t \in [t_0; t_s]$. After that the quadcopter flies with the previously defined constant velocity v_{max}.)

$$\begin{aligned}
v(t_0) &= 0 & v(t_s) &= v_{max}\\
\dot{v}(t_0) &= 0 & \dot{v}(t_s) &= 0\\
\ddot{v}(t_0) &= 0 & \ddot{v}(t_s) &= 0\\
\dddot{v}(t_0) &= 0 & \dddot{v}(t_s) &= 0
\end{aligned}$$

For this 8 boundary conditions a seventh order polynomial is chosen. This function has the following form:

$$v(t) = A(t-t_0)^7 + B(t-t_0)^6 + C(t-t_0)^5 + D(t-t_0)^4 + E(t-t_0)^3 + F(t-t_0)^2 + \\ + G(t-t_0) + H \tag{3.11}$$

The coefficients of the polynomial can be expressed with t_s (the duration of the acceleration) and v_{max} (constant velocity of on the path, which the acceleration function reaches at $t = t_s$) as follows:

$$\begin{aligned} A &= -\frac{20v_{max}}{t_s^7} \\ B &= \frac{70v_{max}}{t_s^6} & E &= 0 \\ C &= -\frac{84v_{max}}{t_s^5} & F &= 0 \\ & & G &= 0 \\ & & H &= 0 \\ D &= \frac{35v_{max}}{t_s^4} \end{aligned} \tag{3.12}$$

In the expressions of these coefficients, the only variable that can be changed is t_s. (v_{max} is the constant velocity, which is provided by the previously introduced optimization process.) In this way, the task is to determine the shortest t_s acceleration time, during which none of the constraints (input and orientation angles) are harmed. This leads to an other non-linear constraint optimization which should be solved.

This process is solved again in an iterative manner. The aim of the optimization is to minimize t_s (and in this way to define the shortest acceleration function) such that the constraints for the orientation angles and the inputs would not be violated.

$$t_{smin} = min\left\{t_s\right\} \quad \text{so that:} \quad u_{ref} \in [u_{min}; u_{max}] \quad \text{and} \quad \eta_{ref} = [\eta_{min}; \eta_{max}]$$

where: u_{ref} is the reference input and η_{ref} are the reference angular trajectories which are exactly defined by t_s and the spline $sp(\rho)$ during the acceleration in the following way: equations (3.12) and (3.11) determine the velocity profile $v(t)$ from t_s; with the help of (3.5), (3.6) and (3.7) the desired trajectory ξ_d can be obtained; η_{ref} and u_{ref} comes from (3.10).

The iterative optimization process works similarly to the optimization of the constant maximal velocity v_{max}. t_s is reduced by small steps until one of the constraints is violated. The last feasible t_s is kept and then the coefficients of the acceleration function are calculated (3.12). (Since the acceleration and deceleration function are the same, the violation of the constraints is checked both at acceleration and deceleration, so both at the beginning and at the end of the spline.)

The deceleration function can be given easily based on the acceleration function. It can be set to be the mirror image of the acceleration function. It is achieved with the following change in the function: (t_t is the total time duration of the path. Or in other words, at t_t the quadcopter arrives at the endpoint of the curve with 0 velocity.)

$$v(t) = A(t_t-t)^7 + B(t_t-t)^6 + C(t_t-t)^5 + D(t_t-t)^4 + E(t_t-t)^3 + F(t_t-t)^2 + \\ + G(t_t-t) + H \\ t \in [t_t - t_s; t_t]$$

3.1.2 Summary of the spline trajectory design algorithm

Once the spline is fitted on the points, the mapping between the spline parameter (ρ) and the arc length measured from the beginning of the spline ($\tilde{\rho}$) is constructed. The maximal constant velocity and the shortest acceleration time is determined with iterative optimization processes. In this way, the scalar velocity profile $v(t)$ can be determined for the trajectory (optimization precesses and then equations (3.12) and (3.11)). After this, the desired trajectory ξ_d is defined based on equations (3.5),(3.6) and (3.7).

3.2 Piecewise linear trajectory

In case the desired trajectory consists of straight lines between the points, the only problem to be solved is the prescription of a sufficient velocity profile. A proper velocity profile is necessary in order to achieve a 4 times differentiable trajectory. Moreover, for the piecewise linear trajectory, a strict completion time for the whole path is also prescribed. (So in this case the aim is not to complete the trajectory under the shortest possible time, as it was at the spline trajectory, but to complete it under a prescribed time.) The velocity profile to be designed is again divided into an acceleration and a deceleration function on each single line and both of them are sixth order functions of time. (the deceleration function is just the mirror image of the acceleration function on each straight line/ section.) Sixth order polynomials are necessary in order to fulfill the boundary conditions in (3.14), which guarantee the four times differentiability in time. Note, that $\ddot{v}(t_0 + t_a) = \dot{a}(t_0 + t_a) = 0$ is not prescribed in (3.14). It is not necessary, since in the middle of each single line the acceleration of the quadcopter does not have to have a local extrema. This is just the point, where the acceleration changes its sign, before that it accelerated and after that it decelerates. But it can have a constant change in its acceleration. (It is something like, it can go through the horizontal position with a constant angular velocity and it does not have to stop there for a moment.) In this way there are only 7 boundary conditions for each single velocity function. (A single velocity function is only an acceleration or a deceleration function! In other words, the complete velocity function of a single straight line consists of two single velocity functions: one for the acceleration part and one for the deceleration part.) Note that as the velocity profile is symmetric, the acceleration function has the same shape as the deceleration function. Because of this fact, the acceleration takes exactly the same time as the deceleration. Hence, the time instant when the acceleration turns into deceleration is at the middle of the total duration of a single straight line. Because of this symmetry in time and the similarity of the accelerating and decelerating functions, I will only focus on the acceleration part. As we are talking about the first half of the velocity function, (the acceleration part) the following notations will be used:

s_i :	The length of the i^{th} straight line.
t_{total} :	The completion time of the total path.
t_{oi} :	The starting time of the i^{th} section.
t_i :	The completion time of the i^{th} section.
$t_{ai} = \frac{t_i}{2}$:	The duration of the acceleration or the deceleration.
$t_{hi} = \frac{t_{ai}}{2}$:	The acceleration has a maximum value at the middle of the acceleration, so at this instance.
$a_{maxi} = \dot{v}(t = t_{0i} + t_{hi})$:	The maximum acceleration of the i^{th} section.

From the previously announced variables s_i and t_{total} are fixed variables. All the others are free ones, and in the latter discussion we would like to determine them in order to define the acceleration velocity profile. However, it needs to be realized that the remaining time variables are not independent from each other.

The velocity profile on the i^{th} section during acceleration can be obtained with the following polynomial:

$$v_i(t) = A_i(t - t_{0i})^6 + B_i(t - t_{0i})^5 + C_i(t - t_{0i})^4 + D_i(t - t_{0i})^3 + E_i(t - t_{0i})^2 + F_i(t - t_{0i}) + G_i \quad t \in [t_{0i}; t_{0i} + t_{ai}] \tag{3.13}$$

And the boundary conditions are:

$$\begin{aligned} v_i(t_{0i}) &= 0 \\ \dot{v}_i(t_{0i}) &= 0 \\ \ddot{v}_i(t_{0i}) &= 0 \\ \dddot{v}_i(t_{0i}) &= 0 \end{aligned} \qquad \begin{aligned} \dot{v}_i(t_{0i} + t_{hi}) &= a_{maxi} \\ \dot{v}_i(t_{0i} + t_{ai}) &= 0 \\ \dddot{v}_i(t_{0i} + t_{ai}) &= 0 \end{aligned} \tag{3.14}$$

If t_{ai}, t_{0i} and a_{maxi} would be known, it could be solved ($t_{hi} = t_{ai}/2$). But in this case only these conditions are known:

$$\sum_i t_i = t_{total} \tag{3.15}$$

$$\int_{t_{0i}}^{t_{0i}+t_{ai}} v_i(t)dt = \frac{s_i}{2} \tag{3.16}$$

The third condition is, that the *maximum* acceleration is the same on every single path. ($a_{maxi} = a_{maxj}$ for all (i, j)) Due to this condition, from now on I will denote the maximal acceleration simply with a_{max}. This condition would not be necessary but it makes the calculation easier as it can be seen in the followings. The fact that the maximal acceleration on each line is the same practically means that the velocity profile is just magnified proportionally with the length of a line. On a longer path, the drone will have a higher max velocity, but it will be reached with the same maximal acceleration. (The maximal acceleration is also constrained by the maximal thrust force and orientation angles constraints. If the prescribed time is too short for the completion then this acceleration constraint is violated. This violation is checked within the program.) If a_{max} is fixed, the time duration t_i of a single line with the length s_i can be deduced based on the form of the velocity profile of (3.13) and the boundary conditions (3.14).

$$t_i = \frac{5}{8}\sqrt{\frac{21 s_i}{a_{max}}} \tag{3.17}$$

Since we have said that the maximal acceleration a_{max} is the same on each lines and the total time is the sum of the individual path durations this can be said:

$$t_{total} = \frac{5}{8}\sqrt{\frac{21}{a_{max}}}\sum_{i}^{n}\sqrt{s_i} \tag{3.18}$$

Where n is the number of lines. In our case however, not the maximal acceleration but instead the total duration of the path completion (t_{total}) is given and from that should we determine a_{max}. In this respect, a_{max} should be obtained as the function of the individual straight line lengths s_i and the total duration t_{total}, which now can be done easily based on equation (3.18) as follows:

$$a_{max} = \frac{525}{64 t_{total}^2}\left[\sum_{i}^{n}\sqrt{s_i}\right]^2 \tag{3.19}$$

After this, in the knowledge of a_{max} and the length of each straight line s_i, the time variables of each line ($t_i, t_{0i}, t_{ai}, t_{hi}$) can be calculated after substituting back in (3.17) and the variable definitions. Now in the knowledge of a_{max} and t_{hi}, the coefficients of the fifth order polynomial (3.13) for each straight line can be obtained (These can be deduced based on the form of the acceleration function in (3.13), the constraints in (3.14 and the assumption that the maximal acceleration is the same on every single line.):

$$\begin{aligned} A_i &= \frac{a_{max}}{10 t_{h_i}^5} & D_i &= 0 \\ B_i &= -\frac{14 a_{max}}{25 t_{h_i}^4} & E_i &= 0 \\ & & F_i &= 0 \\ C_i &= \frac{4 a_{max}}{t_{h_i}^3} & G_i &= 0 \end{aligned} \tag{3.20}$$

The deceleration function is just the mirror image of the acceleration function, so with the exact same coefficients the function for slowing down can be written as follows:

$$v_i(t) = A_i(t_i - t)^6 + B_i(t_i - t)^5 + C_i(t_i - t)^4 + D_i(t_i - t)^3 + E_i(t_i - t)^2 + F_i(t_i - t) + G_i \qquad t \in [t_{0i} + t_{ai}; t_i] \tag{3.21}$$

Once the function of the scalar velocity function for the straight lines are given the trajectory definition can be achieved relatively easily. Let p_i denote the i$^{\text{th}}$ prescribed point of the path and $r_i = p_{i+1} - p_i$ the vector that points from that point to the next one. Then the trajectory prescription and its time derivatives for the i$^{\text{th}}$ straight line can be obtained as follows:

$$\begin{gathered} \boldsymbol{\xi_d} = s_i(t)\frac{\mathbf{r_i}}{|\mathbf{r_i}|} \qquad \dot{\boldsymbol{\xi}}_{\mathbf{d}} = v_i(t)\frac{\mathbf{r_i}}{|\mathbf{r_i}|} \qquad \ddot{\boldsymbol{\xi}}_{\mathbf{d}} = a_i(t)\frac{\mathbf{r_i}}{|\mathbf{r_i}|} \\ \boldsymbol{\xi_d^{(3)}} = \dot{a}_i(t)\frac{\mathbf{r_i}}{|\mathbf{r_i}|} \qquad \boldsymbol{\xi_d^{(4)}} = \ddot{a}_i(t)\frac{\mathbf{r_i}}{|\mathbf{r_i}|} \end{gathered} \tag{3.22}$$

Where boldface characters denote vectors and the simple ones are scalar quantities.

3.2.1 Summary of the piecewise linear trajectory design algorithm

The maximal acceleration for each single line will be the same. In this way, based on the given total duration t_{total}, the length of each straight line s_i and equation (3.19), (a_{max}) maximal acceleration can be calculated. After this, the duration of each individual line can be calculated (equation (3.17)). From t_i, t_{hi} can be calculated, and hence the scalar coefficients of the velocity function can be obtained (equation (3.20)).

Chapter 4

Simulation results

All three of the methods are simulated in the MATLAB Simulink environment. Initializer MATLAB scripts are written, where the variable initialization, path planning and the necessary calculations for the controllers take place. The control architectures are then built and the simulations run in the Simulink environment. These scripts and the Simulink files are detailed in the Appendix.

The simulations run with the continuous time model of the quadcopter and with continuous time controllers. The time delay of realistic systems is neglected in the simulations. This delay however could be included in the controllers in the following manner: The system and the controllers have top be discretized supposing zeroth order hold inputs. When the calculation of the k^{th} input takes place, measurements of the $(k-1)^{th}$ states are available only. In this way, a popular solution is to simulate the model between the $(k-1)^{th}$ and k^{th} time-steps, in this way to get an approximation of the states in the k^{th} time-step. Than the inputs for the k^{th} time-step can be calculated based on these approximated states. This calculation takes place between the $(k-1)^{th}$ and k^{th} time-steps and the control input is applied to the plant at the k^{th} instant. This method for solving time delays is used for example in [20] where they use it in the application of a discrete MPCC (Model Predictive Contouring Control) controller. This compensation was not the aim of the simulations and hence this solution is not applied during the simulations.

The simulated quadcopter has the following properties, which are typical values. These anyway could be set on demand.

$$m = 0.468[kg] \quad l = 0.275[m] \quad b = 1.14 \cdot 10^{-6}[kgm] \quad k = 2.98 \cdot 10^{-5}\left[\frac{N}{rad^2}\right]$$

$$T_{mot} = 100[mNm] \quad J_{act} = 1.25 \cdot 10^{-7}[kgm^2]$$

$$J = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} = \begin{bmatrix} 4.856 \cdot 10^{-3} & 0 & 0 \\ 0 & 4.856 \cdot 10^{-3} & 0 \\ 0 & 0 & 8.801 \cdot 10^{-3} \end{bmatrix}$$

4.1 Results and experiences

4.1.1 Piecewise linear trajectory

The accuracy of the trajectory following gets better if the quadcopter has more time to complete it. This causes smaller accelerations and hence errors decrease. The results of 2 simulations will be introduced in the followings in which the desired time for the quadrotor to complete the path were 25 and 40 seconds. The total length of the trajectory was $63.41[m]$ which means that the average velocities on the path were $2.53\left[\frac{m}{s}\right]$ and $1.58\left[\frac{m}{s}\right]$ respectively. The highest velocities that the drone has reached were $8.29\left[\frac{m}{s}\right]$ and $5.18\left[\frac{m}{s}\right]$ respectively.

The piecewise linear trajectory on which the quadcopter was tested and the trajectories of the quadcopters with the different controls can be seen on Figure (4.1).

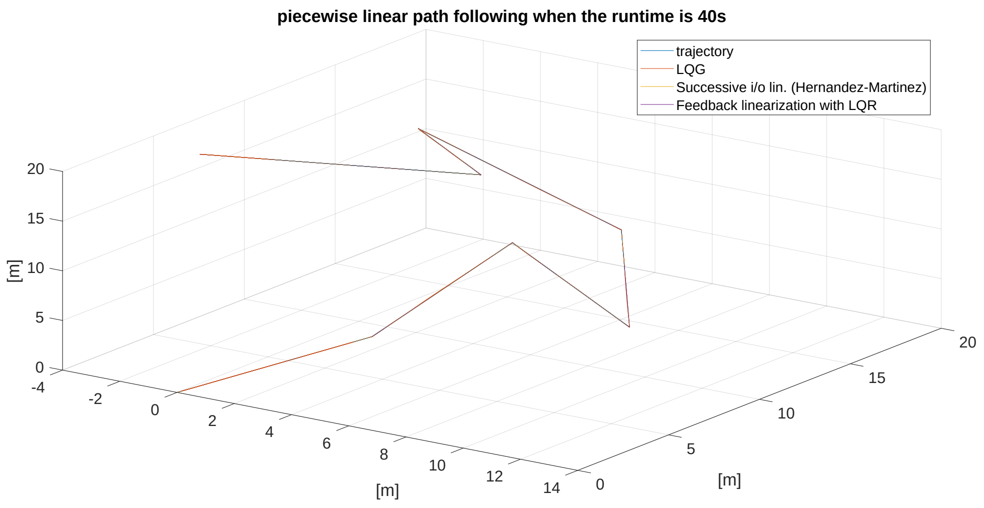


Figure 4.1: *The piecewise linear test trajectory with* $25[s]$ *runtime*

The errors of the runs can be seen on Figure (4.2) and (4.3).

Both the successive i/o linearization and feedback linearization controllers have quite good trajectory following capabilities. These controllers perform better in this simulation than the LQG. First of all, it is due to the fact, that during the design of these controllers we took into account the exact nonlinear system (with some substitutions and transformations to reach a system that is controllable with linear control methods.). This is not true for the LQG controller, where the Jacobi-linearization of the system is not exact. (We neglect the higher order terms from the Taylor-series.) Second of all, the simulation of the LQG controller includes noises as well and we did not assume that we can measure all the states required for the state feedback (Instead a Kálmán-filter is used.). By the other two controllers, we assumed that all the required states are available for measurement. This is however not very realistic, and in reality probably an Extended Kálmán-filter (EKF) [15] would produce these variables for the controllers. Again, due to the fact, that the LQG simulation contains noises, the quadcopter would not be able to reach a stable settling point with 0 error. However, the magnitude of this error remains in the range of the noises.

The rotor velocities that the quadcopter produced in order to follow the path can be seen on figure (4.4), (4.5) and (4.6). It can be examined on figure (4.5) that the rotor velocities

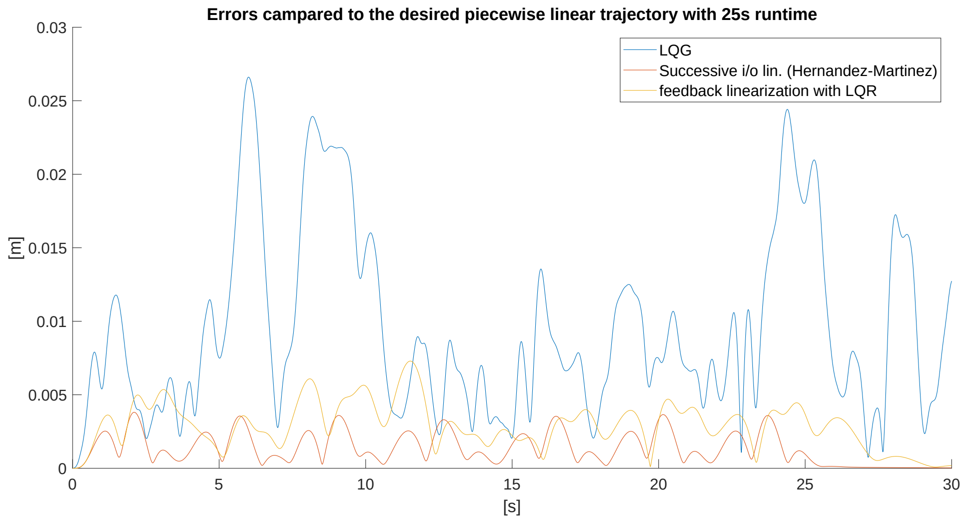


Figure 4.2: *The errors on piecewise linear test trajectory with* $25[s]$ *runtime*

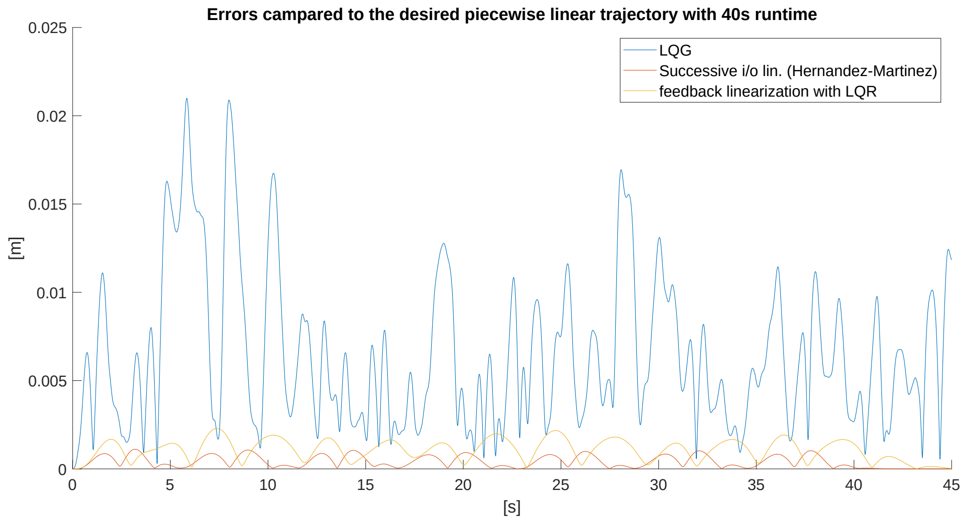


Figure 4.3: *The errors on piecewise linear test trajectory with* $40[s]$ *runtime*

of the LQG controller are noisy compared to the others. This is again caused by the noises in the simulated system. If these noises are tuned to 0 then the rotor velocities are just as smooth as they are in the case of the other two control methods.

4.1.2 Spline trajectory

I used a spiral trajectory to test the controllers on spline paths. The parameters of the trajectory are: $R = 4[m], h = 1[m]$ and it has 8 turns. (h is the rise of each turn.) This simple trajectory choice is reasonable since the LQG controller is unable to follow accurately trajectories which require high angular velocities and angular accelerations because it would be outside of the small neighborhood of the trim-point. The results of two simulations will be introduced with different, $4\left[\frac{m}{s}\right]$ and $6\left[\frac{m}{s}\right]$, velocities.

The spiral trajectory and the paths of the two quadrotors can be seen on Figure (4.7).

The errors of the control methods with the two different speeds can be seen on Figures (4.8) and (4.9).

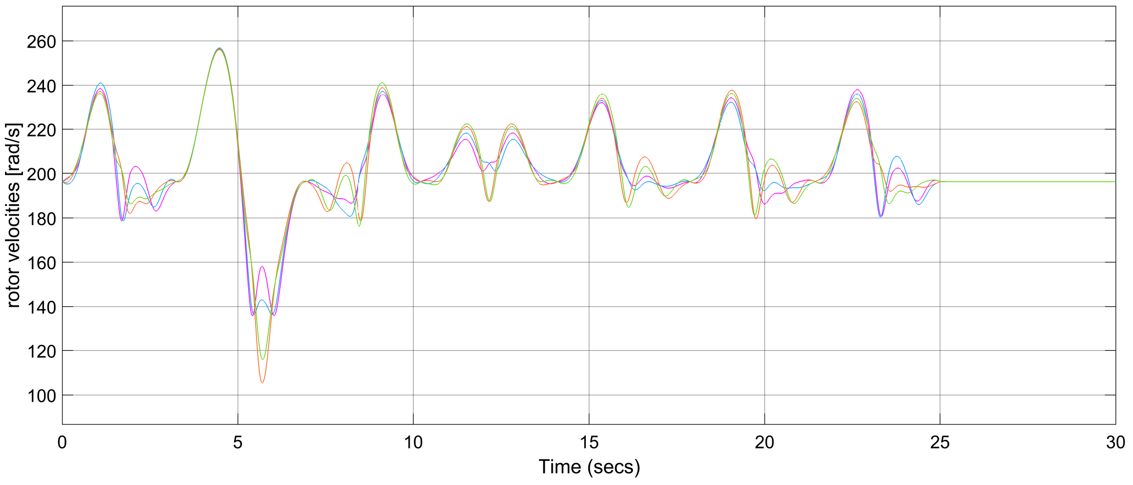


Figure 4.4: *Rotor velocities on piecewise linear trajectory with successive i/o control*

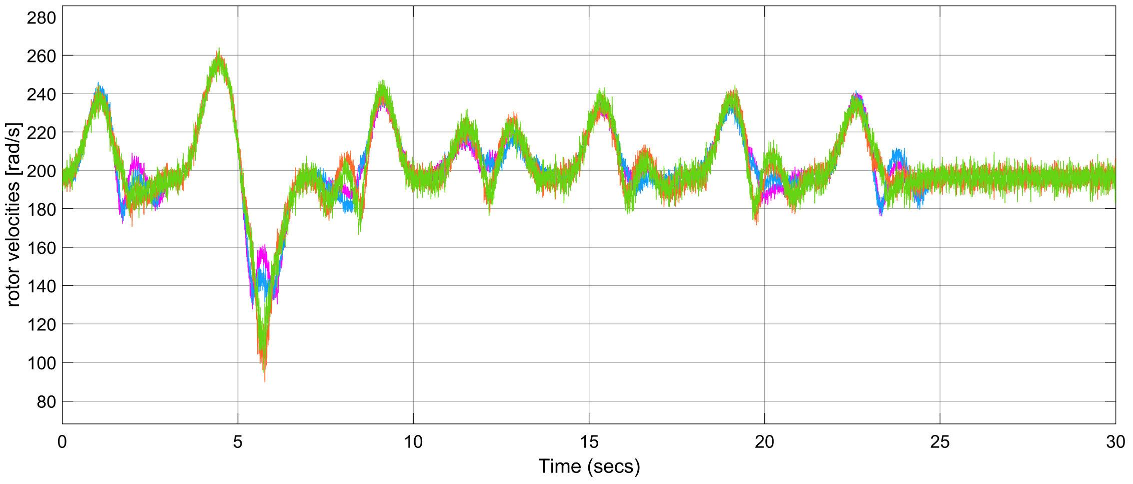


Figure 4.5: *Rotor velocities on piecewise linear trajectory with LQG control*

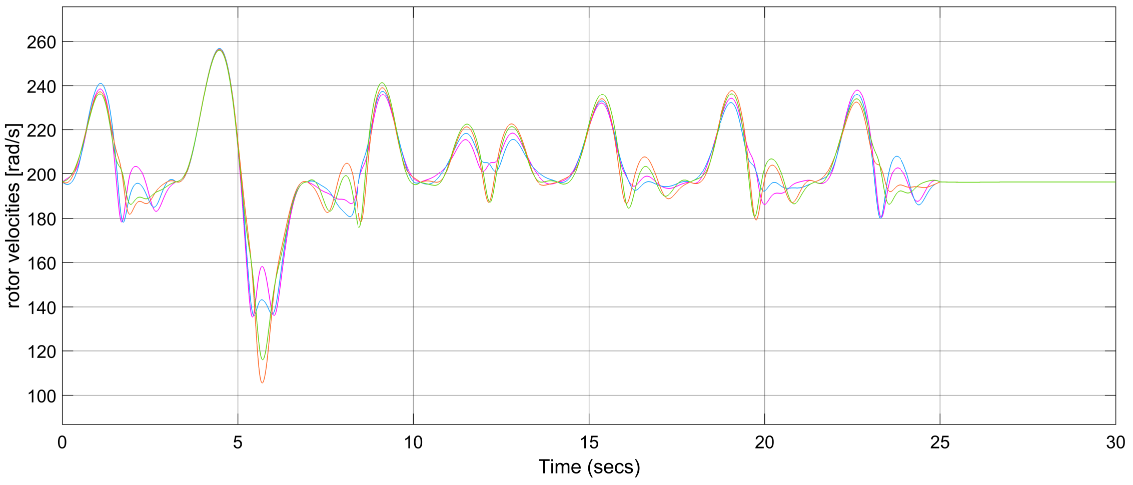


Figure 4.6: *Rotor velocities on piecewise linear trajectory with feedback linearized system with LQR control*

The successive i/o linearization controller works quite stable in this case as well. The error is almost constant and has a low value in the middle of the path, which is the

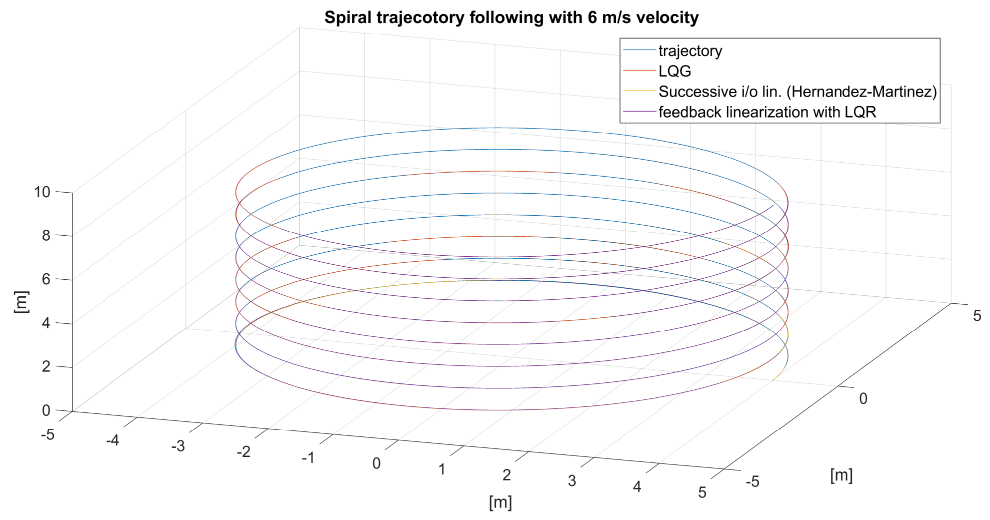


Figure 4.7: *The spiral test trajectory with* 6 $\left[\frac{m}{s}\right]$ *desired velocity*

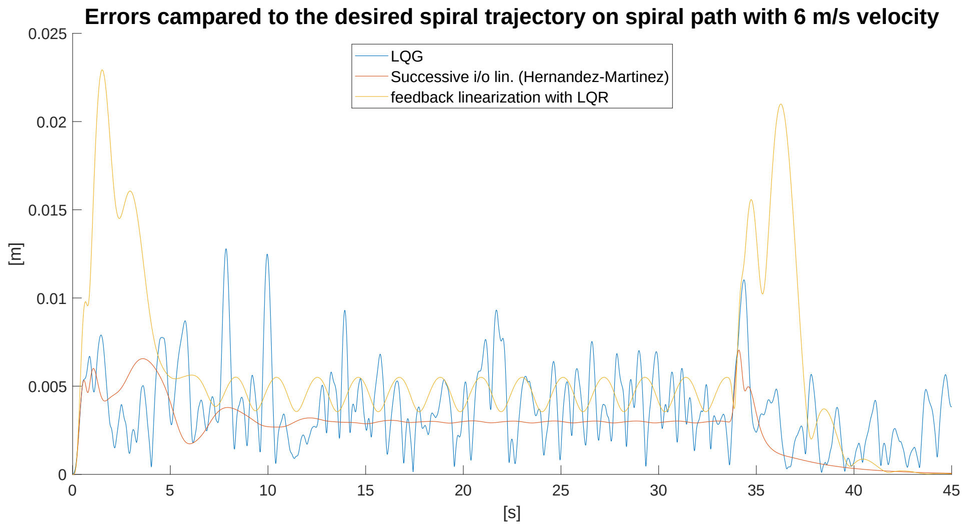


Figure 4.8: *The errors of the spiral trajectory following with* 6 $\left[\frac{m}{s}\right]$ *desired velocity*

period of constant path parallel velocity. The error fairly grows during acceleration and deceleration. It is also the effect of the actuator model, since it saturates occasionally during these periods.

The LQG controller's error is again in connection with the noise. Due to these disturbances, the error can not be near zero permanently at the end of the path. On the other hand, a that big difference in accuracy compared to the other controllers, as it was by the piece wise linear trajectory simulation, is not present in this case.

The error of the feedback linearization controller grows much higher at the beginning and at the end of the path. This is probably caused by the effect of the new control variable choice. Since the real thrust force is the second integrative of this control variable, the quadcopter can not react to big accelerations that well. (It slows the response of the controller.) The same effect can be examined in sharp turns as well. This integrating effect causes also that once a bigger error has been present, it will only vanish after some fluctuations. (This is the same phenomena that occurs if a PID controlled system is thrown out of its stable position. It can only settle after at least one swing to the other direction.)

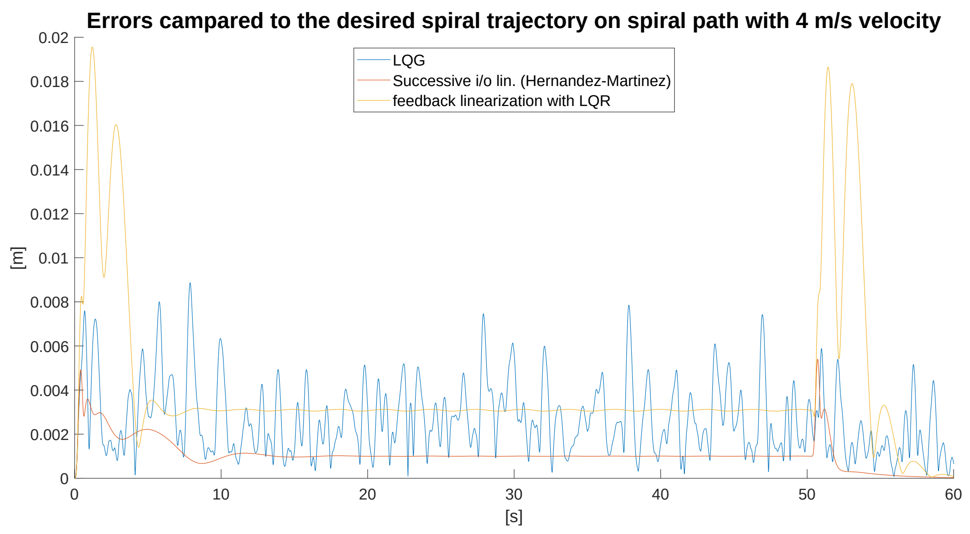


Figure 4.9: *The errors of the spiral trajectory following with* $4\left[\frac{m}{s}\right]$ *desired velocity*

This problem did not appear in the linear path case probably because there were smaller accelerations.

4.1.3 Robustness with different weights

The robustness capabilities of the controllers were also examined. I ran 10-10 simulations with each of them. In the first five runs, only the mass of the quadcopter was modified in 5 steps between 10% and 50%. In the second 5 runs, the mass and also the inertia of the quadcopter was increased (again in 5 steps between 10% and 50%). The controllers were tested on the previously introduced spiral trajectory with $4\left[\frac{m}{s}\right]$ velocity.

The successive i/o linearized controller has quite good robustness properties. It follows the trajectory with a promising accuracy in x and y directions and it only has a more significant inaccuracy in the z direction. At the end of the path, it hovers in a stable position with a remaining error in the z direction. This inaccuracy in the hovering elevation is a property that comes from the nature of the controller: If there were no position inaccuracy then the thrust force would be just as big that it would be able to hover the quadcopter with the exact weight. If the weight is bigger, a bigger control input is required and this can only be reached if there is an inaccuracy in the position.(Some sort of an integrating property is missing from the controller.) The errors of the successive i/o controller in this case can be seen on Figure (4.10).

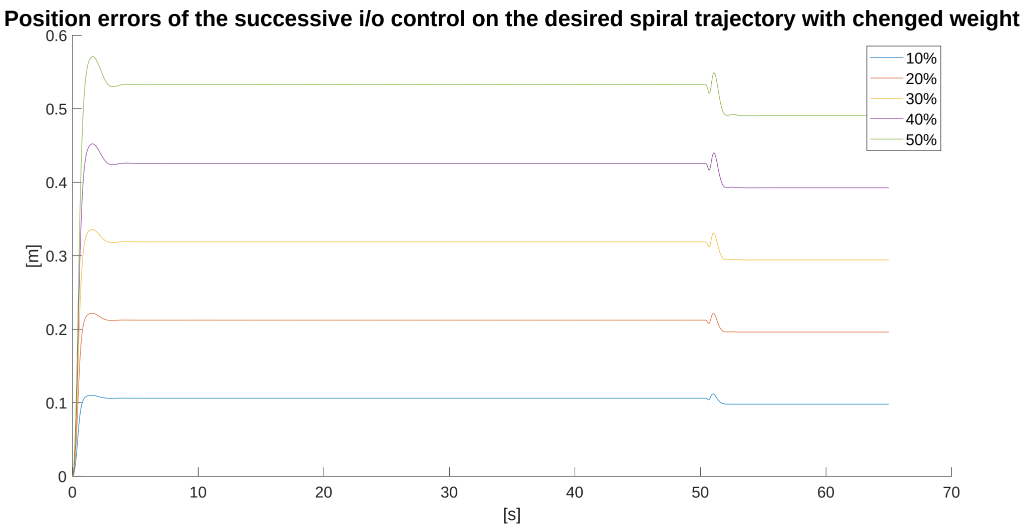


Figure 4.10: *The errors of the successive i/o controller on spiral trajectory when the weight of the quadcopter is increased*

The LQG control acts quite similarly to the previously shown successive i/o controller. Again, its x and y direction accuracy is good and it has a vital error in the z direction. Similarly to the previous case, the position in the z direction can not be accurate completely. This is due to the fact, that the reference velocity is defined in the following way: $v_{ref} = \dot{\xi}_d + k(\xi_d - \xi)$. It can be seen from equation (2.41) that if the first row of $\hat{\delta}_x$ is zero, then the input will become exactly the reference input. Which is not enough for the increased weight quadcopter to levitate in a stable position. In this way, there must be a difference between the prescribed and the produced velocity. Since $\dot{\xi}_d = 0$ and we want $\dot{\xi}$ to be zero vector, the only possibility is that $v_{ref} \neq 0$. And that can only be if $\xi \neq \xi_d$. The accuracy can be improved by a higher value choice for k in v_{ref}. However, if k is too big then the controller will react to a small position error very aggressively and hence the stability of the system can be endangered. The errors of the LQG controller in this case can be seen on Figure (4.11).

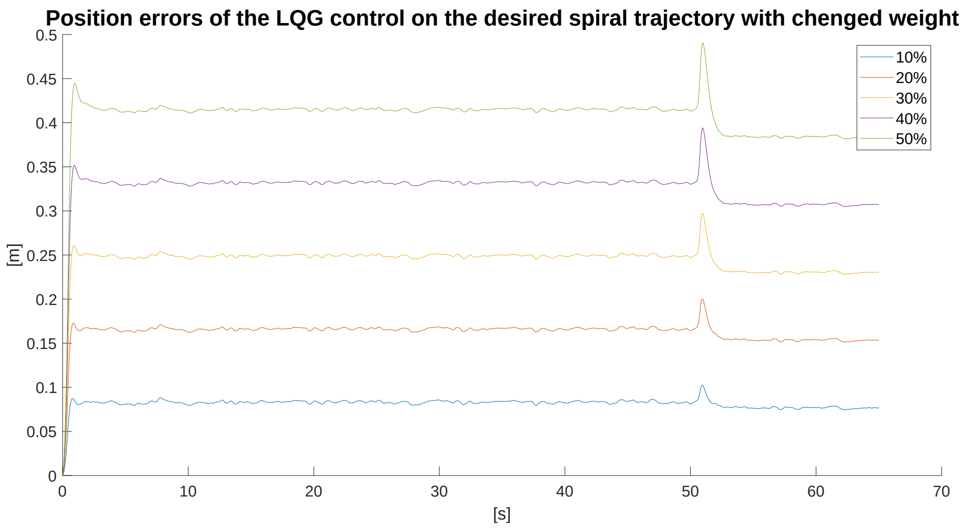


Figure 4.11: *The errors of the LQG controller on spiral trajectory when the weight of the quadcopter is increased*

From a certain aspect the feedback linearized system with LQR controller has the best robust properties from the 3 controllers. It has a very small position error when the path parallel velocity is constant on the spiral and is capable of a perfectly accurate position keeping at the end. On the other hand, this is the control which is the most sensitive for parameter varying when it has to accelerate (Especially when the inertia of the quadcopter is not accurate, see in the next part.). The reason for this is that due to the new auxiliary control choice, the desired thrust force is calculated through two integrators. This is exactly what was missing from the previous two controls, and this is why it is capable of hovering in the exact position. The downside of this integrating manner in the thrust force is that it reacts slower, and this is why it produces bigger errors and longer transients while accelerating. The errors of the feedback linearized system with LQR controller in this case can be seen on Figure (4.12).

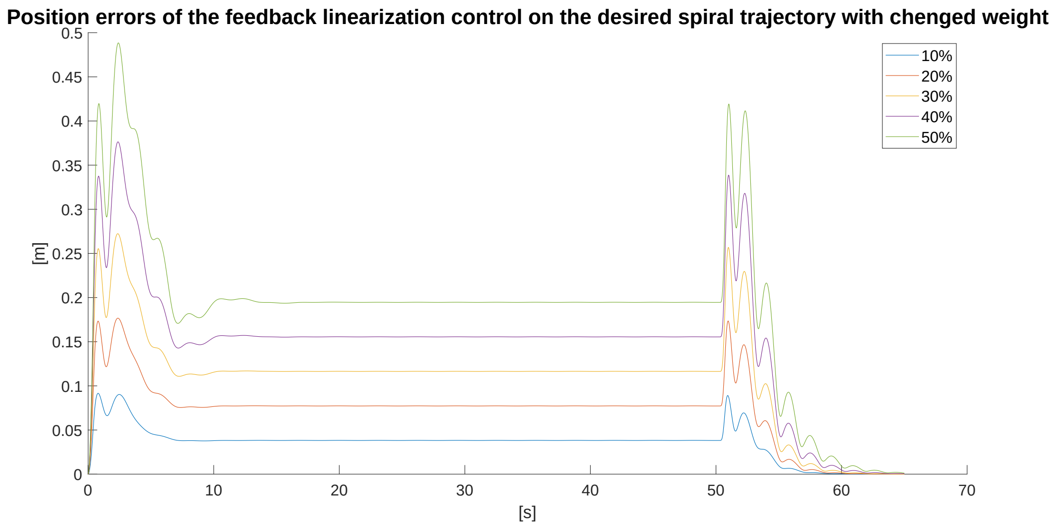


Figure 4.12: *The errors of the feedback linearized system with LQR control on the spiral trajectory when the weight of the quadcopter is increased.*

4.1.4 Robustness with different weights and inertia

In these simulations the inertia of the quadcopter was also changed. In many cases this is a more reasonable examination, since if for example the quadcopter has to carry something it can heavily effect the inertia of the whole system, not just the weight of it.

The change in the inertia of the quadcopter did not really effect the trajectory following accuracy of the successive i/o and the LQG controller. Generally it can be said, that the growth of the inertia has an impact on the performance when high angular accelerations occur, so in other words, when the fourth derivative of the trajectory is high. In the case of the spiral trajectory this scenario happens mainly during acceleration and deceleration. According to this, the main difference between this simulation and the previously introduced one appears at the beginning and the end of the path. During these periods the errors are fairly larger compared to the previous case. The errors of the successive i/o linearization and the LQG controllers when the weight and the inertia of the quadcopter is changed can be seen on Figures (4.13) and (4.14) respectively.

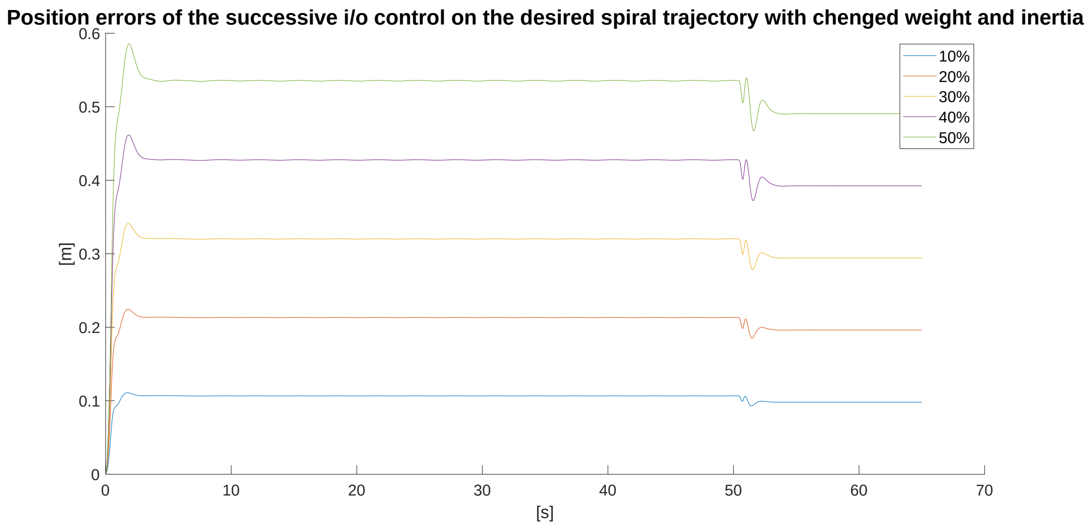


Figure 4.13: *The errors of the successive i/o controller on the spiral trajectory when the weight and inertia of the quadcopter is increased.*

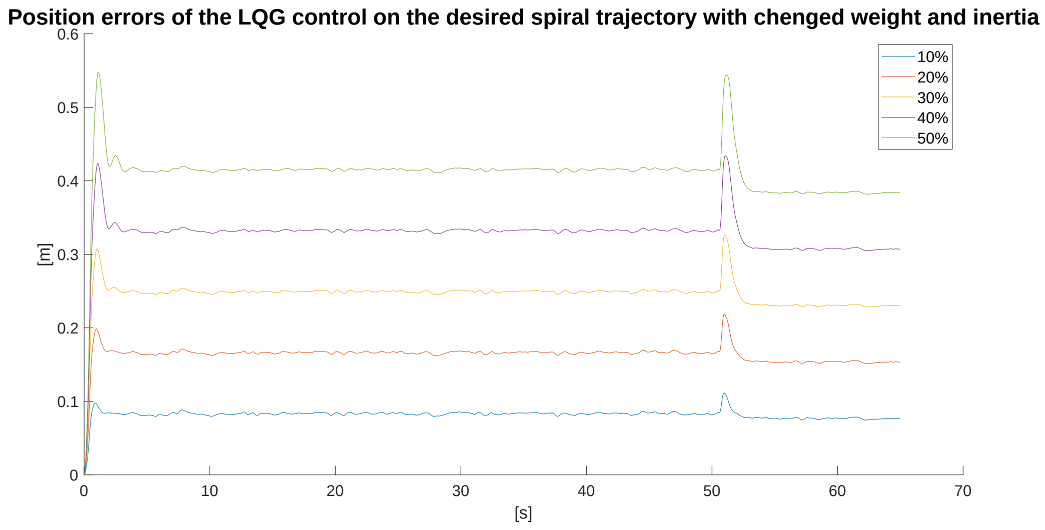


Figure 4.14: *The errors of the LQG controller on the spiral trajectory when the weight and inertia of the quadcopter is increased.*

The feedback linearized system with LQR controller on the other hand is heavily effected by the change in the inertia of the quadrotor. Again, in situations when the inertia of the quadcopter does not play an important role, so when the angular accelerations are not high, the controller is capable of following the path with relatively small errors, even when the weight and inertia parameters are changed heavily. These errors nevertheless are much bigger than they were in the previous case. During acceleration and deceleration the controller reacts slowly and hence the errors grow huge and only disappear after a longer oscillation. The errors of the feedback linearized system with LQR control in this case can be seen on Figure (4.15).

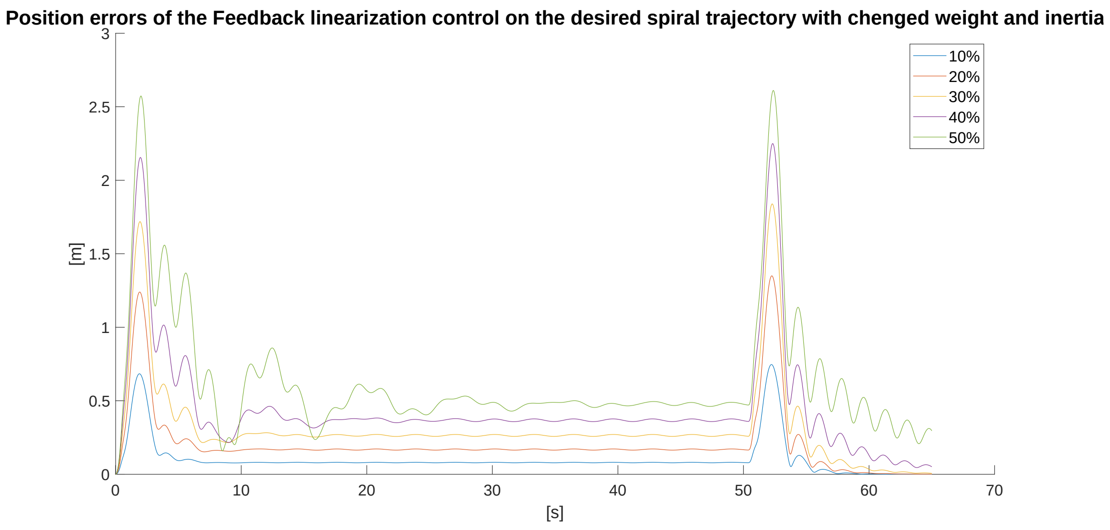


Figure 4.15: *The errors of the feedback linearized system with LQR control on the spiral trajectory when the weight and inertia of the quadcopter is increased.*

Chapter 5

3D visualization in V-REP

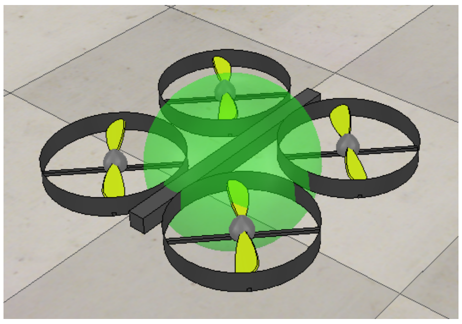


Figure 5.1: *quadcopter V-REP model*

V-REP is a robot simulator software with integrated development environment. It is based on a distributed control architecture: each object/model can be individually controlled via an embedded script, a plugin, a ROS or BlueZero node, a remote API client, or a custom solution. What makes V-REP interesting for us is that the control can also be written in a MATLAB script through the remote API framework. Furthermore, V-REP luckily contains a default quadcopter model which basically consists of the body and the 4 rotors.

The working of V-REP is based on the parent child relationships between the parts. In the quadcopter's case the body is the parent of the rotors among others. This means, that there is a matrix defined between them that defines the children's positions related to the parent. So if one object is the parent of an other, they can be moved together in V-REP. In the software, the parts usually have (threaded or non threaded) child scripts attached to them, from where they are controlled. These scripts are written in LUA language. But V-REP can be controlled through MATLAB as well, and I have used this option for the sake of simplicity.

The Visualization goes as follows: The motion of the controlled quadcopter is simulated in a MATLAB script. From this simulation, the actual simulated position and orientation of the drone is sent to V-REP through the Remote API framework. The controller which is simulated in this way is the successive i/o linearization control. The detailed description of the method and the simulation files can be found in the appendix in the A.4 section.

A video of the 3D simulations can be seen on this Youtube link:
https://youtu.be/LH8xv1qjG5c

Conclusion

Trajectory tracking control solutions with different kinds of trajectory design methods have been introduced in this paper. The controllers have been tested via simulations on the different trajectories with different speeds and with changed weight and inertia. The tracking errors have been plotted in these cases and explanation to their possible origins have been made. Based on these experiments, it can be said, that the presented controls are suitable for tasks such as the ones proposed in the introduction.

The **successive input-output linearization controller** uses an exact linearized model of the quadcopter through clever variable changes, and applies a two level control architecture on it. Based on the transversal errors, the external loop produces the thrust force and the angular trajectories for the internal loop. The internal loop then calculates the desired torque inputs based on the angular errors. The method has been proposed in [1]. The control algorithm has quite good trajectory following capabilities if it is well tuned. Its robust tracking properties are also sufficient, however, position error will always remain due to the lack of integrating property in the controller. In real life operation an EKF state estimator would be needed to provide the necessary state information.

The **Jacobi linearized model with LQG controller** uses the trimmed out model model of the quadcopter around the hovering state, it is also charged with noises in the states and in the measurements as well. An LQG controller for this linear plant is then investigated. This type of linearization is not exact and hence the trajectory following capabilities are a little poorer than in the case of the other two controllers. On the other hand, designing a KF is much easier than doing it for an EKF. Furthermore, due to the totally linear controller, computational requirements are smaller than in the other two cases. The robust behavior of the control is quite similar to the successive i/o's, Position error remains due to the lack of integrating property.

The **feedback linearized model with LQR controller** uses the exact linearized model of the plant via feedback. An LQR controller is then applied to the linearized system. The sufficient conditions for exact feedback linearization are achieved through changes in the input variables. Thanks to these changes, integrating manner also appears in the control which was missing from the previous two solutions. This integrating property makes the control to react slightly slower than the other two solutions, but it has an appealing behavior when it comes to robustness. The controller reacts to accelerations slower, and hence fluctuations can appear in these cases, however no position error remains, which could be an important aspect during design.

Next to the control methods, two trajectory designing solutions have been proposed for solving the problem of generating sufficiently smooth trajectories through desired points in the 3D space. The requirements for the trajectories such as sufficient smoothness based on the dynamics of the quadrotor and the limitations that originate in the actuator properties are introduced. Trajectory designing methods have been then shown, which fulfill these requirements partially by analytical and partially by optimization methods.

Although these trajectory design methods are sufficient for the tasks, better solutions could also be carried out. The piecewise linear trajectory solves the problem of keeping the quadcopter in a safe area defined by the properly positioned desired points. On the other hand, the fact that the quadcopter has to stop at every single key-point makes the trajectory not very time and consumption efficient. In this respect, a possible future development area can be to solve this problem based purely on optimization. Nice solutions are proposed for example in [8], where solutions for designing smooth trajectories though desired points within safe corridors are introduced.

Acknowledgement

This work has been supported by the GINOP-2.3.2-15-2016-00002 grant of the Ministry of National Economy of Hungary.

I would also like to thank *Tamás Péni* for the huge amount of help during this work!

Appendix

A.1 File description of the Successive i/o linearization simulation

loadvar.m The weighting matrices Q and R from equation (2.12) can be set here. After this the script calculates the LQR feedback gain of the external loop of the control, defined in (2.11) and (2.12). After this the feedback gain of the internal loop is calculated based on equations (2.15) and (2.16).

spline_points.m

(a) several constants, such as the quadcopter parameters, gravitation, actuator parameters etc. is set here.

(b) The desired points, to which the trajectory should be fitted, can be given.

(c) The path type can be set. path_type=1: spline on the points, path_type=2: piecewise linear trajectory on the points.

(d) fits the desired curve to the points

(e) The desired time of completion for the piecewise linear trajectory is set here.

(f) calculates the maximum velocities, accelerations with that the drone can complete the path safely

(g) calculates the total time necessary for the path, and some other values that are useful during the simulation

(h) calculates many important values for running. (such as time breakpoints of acceleration/deceleration, spline parameters etc.)

quad_LQR.slx This is the file, where the simulation of the controller runs. The mechanism of the controller simulation can be summarized as the following:

The next point and the first four time derivatives of the desired trajectory is calculated in each time-step (MATLAB function name: **trajectory_generation**). From the desired trajectory and the real position of the drone, the position error vector, specified in equation (2.10), is calculated (MATLAB function name: **posture_error**). From the error, the desired orientation angles η_d and the thrust force (F_d) is calculated based on equations (2.13), (2.5), (2.7) and (2.2) (MATLAB function name: **Calculate_angles**). (the third orientation angle ψ_d is also calculated, but not based on these equations. For the yaw angle, ψ, any reasonable trajectory can be defined. From the desired orientation angles and the real actual orientation of the quadcopter the angular error vector (e_η) is calculated (MATLAB function name: **Angle_error**). From the angular error and the desired thrust force the ω input for the motors are calculated (MATLAB function name: **Omega_for_ESC**). If the motor can not produce that high acceleration on the rotors, as it would be necessary, saturation occurs ((MATLAB function name: **actuator_dynamics**). From the rotor velocities the torques, thrust force and the angular accelerations are calculated base on equations (1.10) and (1.2) (MATLAB function name: **Quadcopter plant**). From these the real position can be found based on the translational dynamics of the quadcopter described in (1.1) (MATLAB function name: **Translational dynamics**).

First **loadvar.m** and **spline_points.m** should be loaded in order to calculate the parameters for the simulation. After that the **quad_LQR.slx** can be run. In the run parameters the following settings are recommended: Fixed step, ODE1 (Euler) solver, 0.001 step size. The simulation time should be set according to the length of the path. (e.g. the total_time variable should be checked, to get the necessary time for the drone to complete the path)

In the followings the MATLAB Function blocks of the Simulink file are detailed.

A.1.1 trajectory_generator

Inputs:

v	The desired velocity in case the path is a spline. This is the velocity which is safe for the quadcopter, even in the sharpest turn. The drone accelerates to this, keeps it for a while and then decelerates to zero at the and.
t_sattle	The time that is needed to accelerate to ”v”.
t	The actual time of the simulation.
total_time	The total time that is needed to complete the path in both cases.
time_breakpoints	This vector is needed only for the linear path type. This helps to decide between which 2 points is the quadcopter at any time instant.
a_max	The maximal acceleration the drone can handle.
path_type	This declares whether the trajectory is a spline (1) or a piecewise linear one (2).
Coefs_sp	The coefficients of the spline that is fitted on the desired points. (**sp.coefs**)
Coefs_ro_rotild	The coefficients of the spline that represents the $\rho(\tilde{\rho})$ function. (**ro_rotilde.coefs**)
Breaks_sp	The breakpoints of the ρ spline parameter of the spline that is fitted on the desired points (**sp.breaks**)
Breaks_ro_rotild	The breakpoints of the spline parameter of the spline that represents the $\rho(\tilde{\rho})$ function. (**ro_rotilde.coefs**)
spline_length	The total length of the spline.
last_point	The endpoint of the trajectory where the drone should arrive and levitate in at the end.
points	The vector of the desired points.

Outputs:

xi	the point of the desired trajectory at the time instant t
xi1	The first time derivative of the desired trajectory at the time instant t
xi2	he second time derivative of the desired trajectory at the time instant t
xi3	he third time derivative of the desired trajectory at the time instant t
xi4	he fourth time derivative of the desired trajectory at the time instant t

This function contains a case structure for the two different path type. In case of a spline, the quadcopter should accelerate to the specified "v" and at the end slow down to zero. In case of straight lines path, the drone accelerates and decelerates on every line, as it was detailed previously. The function block contains a case structure for each types of trajectories.

Spline

A spline is fitted on the desired points of the trajectory. After that, as it was detailed before in section 4.1, the function between arc length on the path $\tilde{\rho}$ and spline parameter ρ is reached by fitting a spline on many sample points. (we evaluate the spline at many ρ spline parameter points and always calculate the arc length of the spline from the beginning of the spline till this point. In this way we produce two vectors containing many ρ and $\tilde{\rho}$ pairs. A spline (**ro_rotilde**) is fitted on these points to get $\rho(\tilde{\rho})$ function.)

The problem is, that a spline structure can not be given to a MATLAB Function Block as an input variable. The solution is to use only the parameters of these splines and evaluate them manually within the block. These splines have the following general form:

$$r(p) = A_i(p - p_{0i})^3 + B_i(p - p_{0i})^2 + C_i(p - p_{0i}) + D_i \qquad p \in [p_{0i}; p_{0i+1}]$$

where p is the parameter of the spline, A_i, B_i, C_i and D_i are the coefficients of the i$^{\text{th}}$ part and $p_0 i$ is the value of the parameter at the beginning of the i$^{\text{th}}$ part. The coefficients and the beginning parameters of the parts of a spline are held in the **.coeffs** and **.breaks** fields of a spline struct. Since a spline can not be given as an input variable to a MATLAB Function Block, these variables (coefficients and breaks) are given to it. From these variables the splines and their derivatives can be reconstructed and evaluated easily.

The trajectory generation goes as follows: $\tilde{\rho}(t) = s(t), v(t), a(t), \dot{a}(t)$ and $\ddot{a}(t)$ are calculated in each time-step with the help of equations (3.11) and (3.12). With the help of the breaks and coefficients of the spline **ro_ro_tilde** (which represents the $\rho(\tilde{\rho})$ function), the actual $\rho(\tilde{\rho})$ spline parameter and its derivatives respect to $\tilde{\rho}$ are calculated. With ρ, the spline fitted on the desired points ($\xi(\rho)$) and its derivatives respect to ρ can be calculated. In this way the next value of the trajectory and its first 4 time derivatives are calculated.

Straight lines path

Based on the variable **time_breakpoints**, the program determines on which straight line the desired point in the next time instant should be. $s(t), v(t), a(t), \dot{a}(t)$ and $\ddot{a}(t)$ values are calculated based on equations (3.20) and (3.13). After it, based on equation (3.22) the next point of the trajectory and its first 4 time derivatives can be calculated.

A.1.2 posture_error

Inputs:

xd	desired trajectory (ξ_d)
xd_p	the desired velocity (first time derivative of the trajectory) $\dot{\xi}_d$
xi	the real position of the quadcopter (ξ)
xi_p	the real velocity of the quadcopter ($\dot{\xi}$)

Outputs:

e_vect	the position error vector, introduced in (2.10): $e = [e_x, \dot{e}_x, e_y, \dot{e}_y, e_z, \dot{e}_z]^T$

A.1.3 Calculate_angles

Inputs:

K	feedback gain matrix, calculated in **loadvar.m** based on equation (2.12).
e	position error vector
xd_pp	second time derivative of the desired trajectory
psi_final	the desired final orientation of the quadcopter, at the endpoint.
total_time	The time, the quadcopter needs to complete the path.
t	actual time.

Outputs:

etaD	The desired orientation of the quadcopter
ThrustD	The desired thrust force of the rotors, the quadcopter has to produce.

Based on equation (2.11) the auxiliary control vector r can be calculated. From r the desired angles ϕ, θ and the desired thrust force can be computed (equations (2.5), (2.7) and (2.2)). Since the third Euler-angle of the orientation, ψ, does not appear in these equations (this is because of the choice of the transformation matrix R), a trajectory for this can be prescribed for it separately from the ξ_d desired translational trajectory. This $\psi(t)$ trajectory has to be differentiable continuously 2 times in time. (The rotor velocities are the functions of the second derivatives of the angular trajectories.)

A.1.4 angle_error

Inputs:

etaD	desired orientation (η_d)
etaD_p	the desired angular velocity (first time derivative of the desired orientation $\dot{\eta}_d$
eta	the real orientation of the quadcopter (η)
eta_p	the real angular velocity of the quadcopter ($\dot{\eta}$)

Outputs:

e_vect	the orientation error vector: $e = [e_\phi, \dot{e}_\phi, e_\theta, \dot{e}_\theta, e_\psi, \dot{e}_\psi]^T$

$$e_{\phi,\theta,\psi} = \eta_{d\phi,\theta,\psi} - \eta_{\phi,\theta,\psi}$$
$$\dot{e}_{\phi,\theta,\psi} = \dot{\eta}_{d\phi,\theta,\psi} - \dot{\eta}_{\phi,\theta,\psi}$$

A.1.5 Calculate_omega

Inputs:

K_eta	feedback gain matrix of the orientation error system, calculated based on equations (2.15) and (2.16). It is calculated in **loadvar.m**.
e_eta	orientation error vector
etaD_pp	second time derivative of the desired orientation trajectory
ThrustD	the desired thrust force to be produced by the rotors.
etaD	The desired orientation of the quadcopter.
etaD_p	The desired angular velocity, the first time derivative of the desired orientation.

Outputs:

omega_square	the square of the desired angular velocities of the rotors.

Based on the mechanical properties and equations (2.14) and (2.17) and (2.18) the desired angular velocities of the rotors can be calculated.

A.1.6 Actuator dynamics

Inputs:

t	The actual time
omega_square_d	the square of the desired rotor velocity
omega_actual	The actual rotor velocity

Outputs:

omega_p	the acceleration of the rotors.
T	Thrust force, produced after the saturation

Based on equations (1.11) and (1.13) the actuator checks whether the motor is capable of the rotor acceleration that the desired rotor velocity prescribes or not. If it is capable of it, the **omega_p** would be so, that the rotor velocity will be the exact same as **omega_square_d**. If the motor can not produce that high acceleration, **omega_p** will be calculated with the biggest torque the rotor can produce.

A.1.7 Quadcopter plant

Inputs:

omega_square	the square of the desired rotor velocities
eta	the real orientation of the quadcopter (η)
eta_p	the real angular velocity of the quadcopter ($\dot{\eta}$)

Outputs:

Thrust	The thrust force generated by the rotors.
eta_pp	The angular acceleration, the second time derivative of the orientation. ($\ddot{\eta}$)

From the actual orientation and angular velocity of the quadcopter and the angular velocities of the rotors, the generated thrust force and angular acceleration can be calculated based on equations (1.10) and (1.2). The angular velocity and orientation is calculated from the integration of $\ddot{\eta}$.

A.1.8 Translational Dynamics

Inputs:

Thrust	The thrust force generated by the rotors.
eta	The orientation of the quadcopter

Outputs:

xi_pp	the acceleration of the quadcopter. $\left(\ddot{\xi}\right)$

From the orientation and the thrust force of the rotors the acceleration of the quadcopter can be calculated based on equation (1.1). The velocity and the position is calculated from the integration of $\ddot{\xi}$.

A.2 File description of the LQG control simulation

setup.m This script is almost completely the same as **spline__points.m** in the successive i/o linearization simulation.

LQG__main.m

(a) Calls the function linear_LQR(), which gives back the(A, B) matrices of the state space model (2.25) that are defined in (2.24) and the K LQR feedback gain of the linearized system.

(b) Calculates the observability and controllability matrices.

(c) Sets the trim point values of the states, outputs and control inputs.

(d) Sets the initial values of the states, outputs and control inputs.

(e) Sets the state disturbance and measurement noise covariance matrices

(f) Calculates the Feedback gain of the Kálmán-filter.

LQG__simulink.slx This is the file, where the simulation of the controller runs. The mechanism of the controller simulation can be summarized as follows:

The next point and the first four time derivatives of the desired trajectory is calculated in each time-step (MATLAB function name: **trajectory__generation**). The reference trajectories of the thrust force and the orientation angles are calculated based on equation (1.1). The desired trajectory of the (ψ) yaw angle could be defined here as well (MATLAB function name: **auxiliary__trajectory**). Based on the Inertia of the quadcopter and the angular trajectories, the auxiliary trajectory of the torques is calculated based on equation (1.2) (MATLAB function name: **tau__ref**). The Kálmán-filter calculates the estimated state based on equation (2.42).(MATLAB function name: **kalman**). Based on the estimated state and the auxiliary reference trajectories the input variable is produced with the use of equation (2.41) (MATLAB function name: **input__calculator**). The necessary rotor velocities for these inputs are calculated based on equation (2.18) (MATLAB function name: **calculate__omega**). From the rotor velocities the torques, thrust force and the angular accelerations are calculated base on equations (1.10) and (1.2) (MATLAB function name: **Quadcopter plant**). From these the real position can be found based on the translational dynamics of the quadcopter described in (1.1) (MATLAB function name: **Translational dynamics**).

The detailed description of MATLAB function blocks **trajectory__generator**, **omega__calculator**, **Quadcopter plant** and **translational Dynamics** can be found in section A.1.

A.2.1 auxiliary__trajectory

Inputs:

xi__d__pp	The second time derivative of the trajectory
m	The weight of the quadcopter.

Outputs:

F_ref	The reference trajectory of the thrust force. (F_{ref})
eta_ref	The reference trajectory of the orientation angles. (η_{ref})

It calculates the reference trajectory of the thrust force and the orientation angles based on equation (1.1).

A.2.2 tau_ref

Inputs:

eta_ref	The reference trajectory of the orientation angles. (η_{ref})
eta_ref_p	The first derivative of the reference trajectory of the orientation angles. ($\dot{\eta}_{ref}$)
eta_ref_pp	The second derivative of the reference trajectory of the orientation angles. ($\ddot{\eta}_{ref}$)
J	The inertia of the quadcopter.

Outputs:

tau_ref	The reference trajectory of the thrust force. (τ_{ref})

The reference trajectory of the torques is calculated based on equation (1.2).

A.2.3 kalman_filter

Inputs:

u	The inputs that were produced by the motors. $[F, \tau]^T$
xi	The measured (and hence noisy) position of the quadcopter. ($\dot{\xi}$)
xi_p	The measured (and hence noisy) velocity of the quadcopter. ($\dot{\xi}$)
eta	The measured (and hence noisy) orientation of the quadcopter
delta_y_est	The estimated output of the system by the Kálmán-filter. ($\hat{y} = C\hat{x}$)
delta_x_est	The estimated state by the Kálmán-filter. ($\hat{x}$)
A_f	The matrix of the linearized system's state space equation.
B_f	The matrix of the linearized system's state space equation.
K_f	Kálmán-filter gain.
xd_p	the first time derivative of the desired trajectory
xd	the desired trajectory
F_ref	The reference trajectory of the thrust force (F_{ref}).
tau_ref	The reference trajectory of the torques (τ_{ref}).
eta_ref	The reference trajectory of the orientation angles (η_{ref}).

Outputs:

delta_x_est_p	The derivative of the estimated state. ($\dot{\tilde{\delta}}_{xest}$)
xi_act_est_p	The first 3 element of the derivative of the estimated state vector.

The Kálmán-filter estimates the states with the use of equation (2.42).

A.2.4 input_calculator

Inputs:

F_ref	The reference trajectory of the thrust force. (F_{ref})
tau_ref	The reference trajectory of the torques. (τ_{ref})
delta_x	The estimated states. ($\hat{x}$)
K	The LQR feedback matrix.

Outputs:

ud	The desired inputs for the quadcopter.

It produces the real state vector $\hat{\delta}_x$ from the Kálmán-filter's state vector $\tilde{\delta}_x$. After it it calculates the desired inputs for the quadcopter based on equation (2.41).

A.3 File description of the feedback linearized system with LQR control simulation

main.m This script is almost completely the same as **spline_points.m** in the successive i/o linearization simulation. It designs the trajectories and the quadcopter parameters are set here. It calls the function **lqr_matrices.m** which returns with the A, B matrices (2.64) of the error system in equation (2.63) and the LQR feedback gain.

symbolic_diff.m This script calculates the $F(x)$ and $G(x)$ functions of equation (2.55) through symbolic differentiation of the dynamic model of the quadcopter. These functions that are calculated here are used in the Simulink file.

Feedback_lin.slx This is the Simulink file in which the simulation of the feedback linearized control runs. The mechanism of the controller simulation can be summarized as follows:

The next values of the trajectory and its derivatives are calculated ((MATLAB function name: **trajectory_generator**)). The error vector of equation (2.63) is built. With the help of the error vector, the auxiliary control v is calculated ((MATLAB function name: **aux_input**)). The u_{new} input variable and τ is calculated from v in (MATLAB function name: **input_calculator**). Through two integrator blocks F is reached from u_f. From the real inputs ($[u_f, \tau]$) the desired rotor velocities are obtained ((MATLAB function name: **omega__calculator**)). The rotor velocities are fed to the dynamical model of the quadcopter (MATLAB functions names: **Quadcopter Plant** and **Translational dynamics**). The states are fed back for the error vector composition.

The detailed description of MATLAB function blocks **trajectory__generator**, **omega__calculator**, **Quadcopter plant** and **translational Dynamics** can be found in section A.1.

A.3.1 aux__input

Inputs:

desired__states	The vector of the desired states of the feedback linearized system. $x_d = [\xi_d, \xi_d^{(1)}, \xi_d^{(2)}, \psi_d, \xi_d^{(3)}, \psi_d^{(1)}]$
real__states	The vector of the real states of the feedback linearized system. $x = [\xi, \xi^{(1)}, \xi^{(2)}, \psi, \xi^{(3)}, \psi^{(1)}]$
K	LQR feedback gain

Outputs:

v	auxiliary input variable v

First the error vector in the form of (2.59) is produced, then the v auxiliary input variable is calculated based on equation (2.58).

A.3.2 input__calculator

Inputs:

v	The auxiliary input variable v.
eta	The real orientation of the quadcopter (η)
eta1	The real angular velocities of the quadcopter ($\dot{\eta}$)
eta2	The real angular acceleration of the quadcopter ($\ddot{\eta}$)
FF	The thrust force
FF1	The first time derivative of the thrust force
xd4	The fourth time derivative of the desired trajectory
psi__d__2	The second time derivative of the desired trajectory of ψ yaw angle.

Outputs:

u__f	An input of the feedback linearized system, that is the second derivative of the thrust force. (u_f)
tau	the torque input of the system (τ)

This block calculates of the new input variables of the system ($u_{new} = [u_f, \ddot{\eta}]^T$) based on equation (2.65). The real input variable ($u = [F, \tau]^T$) is then obtained in two steps. τ is calculated within the block based on equation (1.2). F is reached through two times integration outside of the block. After it u real input is generated with a **Mux** block.

A.4 V-REP

A.4.1 Connecting MATLAB to V-REP

The connection of MATLAB to V-REP can be established with the use of the V-REP Remote API framework. To be able to complete the connection, several files have to be included in our working library. These files can be found in the */programming/remoteApiBindings/matlab/matlab* folder and have the following names: *simpleTest.m*, *simpleSynchronousTest.m* and *complexCommandTest.m*. There is one more file that is needed in the folder: */programming/remoteApiBindings/lib/lib/64Bit* (or *32Bit*) folder, and the file name is: *remoteAPI.dll* for Windows,*remoteAPI.so* for Linux or *remoteApi.dylib* for MAC. We have to copy these files into the working library.

I used synchronous communication mode between the two softwares. The connection can be reached with the functions that are in *simpleSynchronousTest.m* example code. This code should be copied to the MATLAB script and then the control can be built in a "while loop" as it can be seen in the *quadcopter_vrep.m* MATLAB script file.

A smaller complication occurs when the control from MATLAB is built up. Concretely, the previously mentioned parent-child relationships disappear as MATLAB takes over control. This is because of the fact that these relationships are prescribed in the tree structure of the elements but they also have to be declared within the code. Since they were defined in the LUA child scripts, which are disabled, these connections have to be declared again from the MATLAB code. (The LUA-child script of an object has to be disabled in order to be able to control an object from MATLAB through the remote API framework.) The syntax that is used for redefine the parent-child relationships is:

[number returnCode]=simxSetObjectParent(number clientID,number objectHandle,number parentObject,boolean keepInPlace,number operationMode)

A.4.2 Quadcopter with arm attached to it

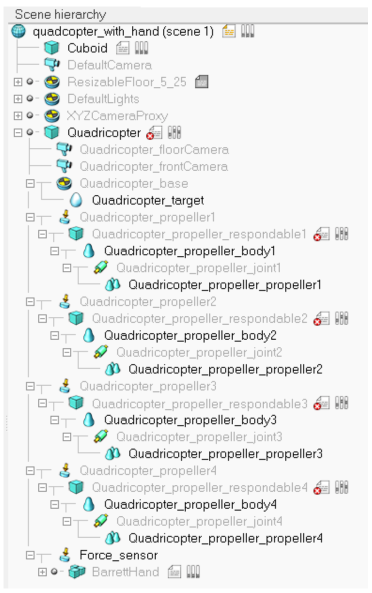
In the V-REP simulation an arm is attached to the quadcopter, symbolizing that it will be used for some sort of transportation. The hand that is used is also a basic element of V-REP, it is called *BarretHand*. The connection between the hand and the quadcopter body is achieved with a force sensor. These sensors, next to their evident function, act as a rigid connection between bodies. After the force sensor is positioned properly in/on the body of the quadcopter, the scene object properties of it should be opened. Under the *Common* tab, one should click on the *Assembling* button. In the jump up window the only check-box should be checked. This will imply that once the force sensor is assembled with an other object, a transformation matrix will be built that specifies the position, orientation etc. between it self and the other object.

Once it is done, first the quadcopter, then the force sensor should be selected and then the *Assemble/Disassemble* button should be clicked. This will make the force sensor the child of the quadcopter. (This can also be observed in the scene hierarchy tree on the right.) After positioning the *BarretHand* correctly, the same steps should be done, except that the parent of the *BarretHand* should be the force sensor.

Unfortunately, in this case the previously mentioned parent-child relationship declaration from the MATLAB code does not work for some reasons. Instead, the LUA code of the *BarretHand* has to be kept and the relationship should be declared there. The code for this is:

effectorHandle=sim.getObjectHandle('BarrettHand')
targetHandle=sim.getObjectHandle('Quadricopter')
sim.setObjectParent(targetHandle,effectorHandle,true)

If it is made correctly, the scene hierarchy tree should look something like on figure (A.4.1). The drone with the hand can be seen o figure (A.4.2):



```
Scene hierarchy
quadcopter_with_hand (scene 1)
    Cuboid
    DefaultCamera
    ResizableFloor_5_25
    DefaultLights
    XYZCameraProxy
    Quadricopter
        Quadricopter_floorCamera
        Quadricopter_frontCamera
        Quadricopter_base
            Quadricopter_target
        Quadricopter_propeller1
            Quadricopter_propeller_respondable1
                Quadricopter_propeller_body1
                    Quadricopter_propeller_joint1
                        Quadricopter_propeller_propeller1
        Quadricopter_propeller2
            Quadricopter_propeller_respondable2
                Quadricopter_propeller_body2
                    Quadricopter_propeller_joint2
                        Quadricopter_propeller_propeller2
        Quadricopter_propeller3
            Quadricopter_propeller_respondable3
                Quadricopter_propeller_body3
                    Quadricopter_propeller_joint3
                        Quadricopter_propeller_propeller3
        Quadricopter_propeller4
            Quadricopter_propeller_respondable4
                Quadricopter_propeller_body4
                    Quadricopter_propeller_joint4
                        Quadricopter_propeller_propeller4
        Force_sensor
            BarrettHand
```

Figure A.4.1: *scene hierarchy tree*

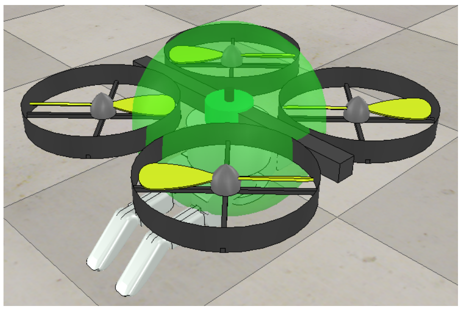


Figure A.4.2: *quadcopter with hand*

A.4.3 MATLAB script

The V-REP simulation/visualization is implemented with the successive i/o linearization controller.

The file **quadcopter_vrep.m** contains the program code of the V-REP simulation. It contains all the connection initializing functions, such as the parent-child relationships and the control program it self. The program part, that is responsible for the connection with V-REP is in the very first part. If the trial for connection was not successful, the rest of the program does not run.

After this comes the initializing part where the necessary handles of the objects are asked from V-REP; the parent-child relationships are declared; all the initial values are set and almost the same code runs as in the MATLAB simulation in **spline_points.m** and **loadvar.m** (Which can be found in the Appendix).

The controller runs in a ”while loop”. The structure is exactly the same as it was in the successive i/o linearization Simulink simulation. The functions are also almost the same and are in function scripts. The only difference can be between these functions and the ones in the Simulink simulation is that in some cases much more practical function calls replace some parts in the program. This appears because many functions are inaccessible from the so called MATLAB Function blocks in Simulink. (Or for example splines can not be given to them as input variables.)

Bibliography

[1] E. Hernandez-Martinez, G. Fernandez-Anaya, E. Ferreira, J. Flores-Godoy, and A. Lopez-Gonzalez, “Trajectory tracking of a quadcopter uav with optimal translational control**the authors acknowledgments the financial support from cona-cyt and universidad iberomaericana, mexico city.” *IFAC-PapersOnLine*, vol. 48, no. 19, pp. 226 – 231, 2015, 11th IFAC Symposium on Robot Control SYROCO 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2405896315026622

[2] M. Rüßmann, M. Lorenz, P. Gerbert, M. Waldner, J. Justus, P. Engel, and M. Harnisch, “Industry 4.0: The future of productivity and growth in manufacturing industries,” *Boston Consulting Group*, vol. 9, 2015.

[3] T. Stock and G. Seliger, “Opportunities of sustainable manufacturing in industry 4.0,” *Procedia Cirp*, vol. 240, pp. 2536–541, 2016.

[4] T. Luukkonen, “Modelling and control of quadcopter,” *Independent research project in applied mathematics, Espoo*, vol. 22, 2011.

[5] R. Fessi and S. Bouallegue, “Modeling and optimal lqg controller design for a quadrotor uav,” *Proceedings of the 3rd International Conference on Automation, Control, Engineering and Computer Science (ACECS’16), Hammamet, Tunisia*, pp. 264 – 270, 2016.

[6] P. Bouffard, A. Aswani, and C. Tomlin, “Learning-based model predictive control on a quadrotor: Onboard implementation and experimental results,” in *Robotics and Automation (ICRA), 2012 IEEE International Conference on.* IEEE, 2012, pp. 279–284.

[7] T. S. Lourenço, A. M. de Almeida Pinto, and R. V. Lopes, “Model predictive control applied to a quadrotor uav,” *Revista Interdisciplinar de Pesquisa em Engenharia-RIPE*, vol. 2, no. 20, pp. 164–178, 2016.

[8] D. Mellinger and V. Kumar, “Minimum snap trajectory generation and control for quadrotors.” pp. 2520–2525, 2011. [Online]. Available: http://www-personal.acfr.usyd.edu.au/spns/cdm/papers/Mellinger.pdf

[9] D. Lam, C. Manzie, and M. Good, “Model predictive contouring control,” in *Decision and Control (CDC), 2010 49th IEEE Conference on.* IEEE, 2010, pp. 6137–6142.

[10] R. Beard and T. McLain, *Small Unmanned Aircraft: Theory and Practice.* Princeton University Press, 2012. [Online]. Available: https://books.google.hu/books?id=YqQtjhPUaNEC

[11] L. Carrillo, A. López, R. Lozano, and C. Pégard, *Quad Rotorcraft Control: Vision-Based Hovering and Navigation*, ser. Advances in Industrial Control. Springer London, 2012. [Online]. Available: https://books.google.hu/books?id=sqniW-H39pIC

[12] H. Kwakernaak and R. Sivan, *Linear optimal control systems*, ser. Wiley-Interscience publication. Wiley Interscience, 1972. [Online]. Available: https://books.google.hu/books?id=mf0pAQAAMAAJ

[13] L. Béla, *Irányítási rendszerek elmélete és tervezése II.* Akadémiai Kiadó, 2016. [Online]. Available: https://mersz.hu?kdid=136

[14] H. Packard, Poola, "Jacobian linearizations, equilibrium points," *Dynamic Systems and Feedback*, 2002.

[15] G. Welch and G. Bishop, "An introduction to the kalman filter," *Technical Report TR 95-041 of Department of Computer Science, University of North Carolina at Chapel Hill*, 2004.

[16] P. N. Shimkin, "The continuous-time kalman filter," *Technion - Israel Institute of Technology, Department of Electrical Engineering - Lecture Notes*, 2009.

[17] Lecturers and doctoral students of BME-MOGI harmonized by Dr. Péter Korondi, *Irányítástechnika jegyzet.* BME-MOGI, 2016.

[18] A. Isidori, *Nonlinear Control Systems*, ser. Communications and Control Engineering. Springer London, 2013. [Online]. Available: https://books.google.hu/books?id=N9h5BgAAQBAJ

[19] K. Groves and A. Serrani, "Modeling and nonlinear control of a single-link flexible joint manipulator," *Department of Electrical and Computer Engineering The Ohio State University Columbus, OH 43210 USA*, 2004. [Online]. Available: http://www2.ece.ohio-state.edu/~passino/lab5prelabnlc.pdf

[20] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1: 43 scale rc cars," *Optimal Control Applications and Methods*, vol. 36, no. 5, pp. 628–647, 2015.