Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Irányítástechnika és Informatika Tanszék

Számítástechnikai és Automatizálási Kutatóintézet
Rendszer- és Irányításelméleti Kutatólaboratórium
Repülésirányítási és Navigációs Kutatócsoport

Kvadkopter trajektóriakövető prediktív szabályozásának tervezése és implementációja

Diplomaterv

Készítette
Gyulai László

Konzulens
dr. Kiss Bálint
dr. Luspay Tamás Gábor

2020. december 21.

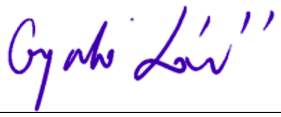
Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Gyulai László*, szigorló hallgató kijelentem, hogy ezt a diplomatervet 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 hozzáférhető 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, 2020. december 21.



Gyulai László
hallgató

Kivonat

Diplomatervemben bemutatom egy kvadkopter szabályozási rendszerének tervezésének és repülési tesztjeinek eredményeit. A SZTAKI-ban kiépített, UWB alapú beltéri pozícionáló rendszerrel felszerelt demonstrációs célú drónarénában történő autonóm működést tűztük ki célul. Ehhez trajektóriatervezési és szabályozási funkciókat kellett fejlesztettem és implementáltam.

A felhasznált modellezési technikák áttekintése után a nemlineáris rendszer trajektória menti linearizálását mutatom be, illetve bemutatok egy B-Spline alapú trajektóriatervező algoritmust. A pozíciószabályozást PID, LQ és MPC szabályozóval is megvalósítom és összevetem a kapott eredményeket.

A kvadkopter repülési tesztjei során Raspberry Pi alapú beágyazott környezetben futtatjuk az irányítást, az orientációszabályozást pedig Pixhawk-on futó Arducopter szoftverrel végezzük. Az implementáció során az LQ és az MPC szabályozókat integráló tulajdonsággal látjuk el a jobb zavarelnyomás érdekében. A repülések rávilágítottak a behangolt szabályozók alkalmazhatóságára a drónaréna környezetben. Befejezésképpen javaslatot teszek néhány megfontolandó továbbfejlesztési lehetőségre.

Abstract

In this thesis, I will give an introduction to the development and flight testing of a quadcopter trajectory tracking controller. In the Institue of Computer Sciences and Control (SZTAKI), a UWB indoor positioning system based quadcopter arena was established to perform autonomous flights. This task involves the development of trajectory planning and control with implementation on an embedded system.

After introducing the nonlinear model of the quadcopter we perform trajectory linearization to have a linear error model of the system. A B-Spline based trajectory generation method was utilized to the system. PID, LQ, and MPC position control were designed and their performance was compared.

During the flight tests, a Raspberry Pi based embedded flight control computer was used as a companion computer for Pixhawk. Ther attitude control and the MAVLink interface was provided by Arducopter. Considering the implementation of the controllers we introduced an integral effect to the LQ and MPC controller to achieve better disturbance rejection. Flight tests has confirmed the applicability of the aforementioned controllers in the quadcopter arena. Possible improvements have been also summarized for further work.

Feladatkiírás

Az autonóm kvadkopterek (drónok) alkalmazási lehetőségei folyamatosan bővülnek, működtetésükhöz ugyanakkor számos alrendszer együttes és összehangolt irányítására van szükség. A Számítástechnikai és Automatizálási Kutató Intézetben (SZTAKI) egy demonstrációs célú drónaréna kerül kiépítésre, melyben az drónok autonóm működése valós körülmények között is kipróbálható. A hallgató faladata a releváns szakirodalom alapján egy teljes szabályozási rendszer megtervezése, mely magában foglalja a feladatspecifikus trajektória generálását és ennek mentén a kvadkopter szabályozását. A feladat része a már elkészült LTI MPC szabályozás továbbfejlesztése, és implementációja.

A hallgató a feladat megoldása során az alábbiak szerint halad:

1. Az irányítási architektúra elvi felépítésének áttekintése.
2. Spline alapú trajektóriatervezési módszerek alkalmazása: algoritmusok a pálya megtervezéséhez, a sebességprofil meghatározása, a trajektóriatervező modul integrációja az irányítási algoritmusba.
3. LTV irányítás vizsgálata: diszkretizálás, mintavételi frekvenciák választása, különböző típusú MPC algoritmusok alkalmazhatóságának vizsgálata (explicit ill. adaptív MPC).
4. EKF alkalmazása az irányítási feladatban.
5. Nemlineáris MPC tervezése és szimulációja.
6. A szabályozás implementációja, tesztelése valódi drónokon.

1. fejezet

Bevezetés

1.1. Motiváció

Napjainkban az autonóm járműveket egyre több helyen megtalálhatjuk, amint kórházakban, autógyárakban, raktárakban önjáróan végzik feladataikat. Felügyeleti, szállítási, célokra kifejlesztett pilóta nélküli repülőgépeket (drónokat) is több helyen használnak külés beltérben egyaránt[1]. A drónok a robotikai kutatások standard platformjai lettek, mivel irányítástechnikai szempontból is nagyon érdekesek: az egyik legnépszerűbb kutatóhelye a beltéri autonóm járműveknek a Zürichi Műszaki Egyetemen található, ahol demonstrációs célú drónarénát építettek [15]. A platform fő eredményeit felhasználva alakultak olyan (úgynevezett spin-off, azaz az egyetemről, kutatóhelyről elindított) cégek, amelyek ma már a piacon elérhető termékekkel rendelkeznek[2]. Ezek miatt relevánsnak tartom a drónok irányításának vizsgálatát, mellyel az önvezető technológiákba nyerhetünk bepillantást.

A közvetkezőkben először bemutatom a kvadkopter által elérni kívánt funkcionalitást, majd áttekintem, hogy milyen eszközpark állt rendelkezésre ennek megvalósítására. Bemutatom a témában eddig született főbb eredményeket, majd adok egy rövid áttekintést a dolgozatban felhasznált esetleg ismeretlen, a témához köthető szakkifejezésekről.

A következő két alfejezet célja az, hogy betekintést nyújtson az elvégzett munkába. A hardverekkel részletesen az implementációnál foglalkozom majd, most csak az irányított szakasz szempontjából közelítem meg a feladatot.

1.1.1. Autopilóta megvalósítása

Egy drón elengedhetetlen kelléke egy távirányító, ami a manuális repülések során a referenciajeleket biztosítja. Különböző repülési módokban a távirányítón alacsonyabb vagy magas szintű parancsok is kiadásra kerülhetnek. Az ismertetett repülési tesztek humán pilóta felügyeletével történtek, hiszen a folyamatosan fejlesztett új funkciók és algoritmusok hibalehetőségeket is magukban rejtenek. A tesztek során a távirányítóval végezzük a drón átkapcsolását manuálisból automata üzemmódba és vissza, illetve vész esetén a motorok kikapcsolását. A tesztek nagy többségét úgy végeztük, hogy a drónt kikötve reptettük, hogy a környezetében ne tehessen kárt rossz beállítások, instabil szabályozó vagy egyéb hiba esetén sem. A tesztek végére a drón szabadon repülésre is képessé vált.

[1] A Flyability nevű cég beltérben is biztonságosan használható drónokat gyárt: `https://www.flyability.com/`

[2] Egy ilyen vállalkozás például a Verity (`https://verity.ch/`), amely automatizált raktározást technológiát kínál.

A cél az, hogy a drónt élesítve, és automata üzemmódba kapcsolva adott feladatokat el tudjunk végezni, úgy mint a fel- és leszállás, a drón egy helyben tartása, trajektóriakövetés, identifikációs célú gerjesztések kiadása. Munkánk során többféle szabályozást kipróbálva kerestük azt a szabályozót és paraméterezést, amely a rendszerünk korlátain belül a legjobb eredményt tudja elérni.

1.1.2. A felhasznált eszközök

A munka során egy pilóta nélküli repülőgépet, azaz (az angol drone kifejezést magyarosítva) drónt fogunk használni. A drónok közé sorolható az általunk használt kvadkopter, aminek 4 propellere (légcsavarja) van, de léteznek még hexakopter és optokopter drónok is 6- illetve 8 propellerrel. Léteznek merevszárnyú drónok is, viszont azok beltéri használatra nem alkalmasak, ugyanis nem tudnak egy helyben lebegni ("hover") [15].

Számos cég árul készen kapható rendszereket, amelyek a dobozból kivéve repülésre alkalmasak. Többnyire ezeknek a manuálison kívül *félautomata* funkciói is vannak, amelyek jellemzően GPS alapon biztosítják pl. az egy helyben lebegést szeles időben.

A mi rendszerünk egy beltéri rendszer, ahol nem lehetséges a GPS használata [18]. Számos megoldás elérhető beltéri pozicionáló rendszerek körében is, ám ezek többnyire rendkívül drágák.

Mi a SZTAKI-ban egy egyedi fejlesztésű rendszert használtunk. Korábbi kollégák állítottak össze egy UWB alapú pozícionáló rendszert, melynek nagy előnye, hogy jelentősen olcsóbb a fent említett kész megoldásoknál, amelynek pontossága természetesen elmarad a professzionális termékektől. Ellenben egy saját fejlesztésű rendszer használatával jobban fogjuk ismerni a működő rendszert, mivel az esetleges hibákat is magunknak kell elhárítani.

A drón összerakását is hasonló elvek vezérelték. A piacon elérhető komponenseket felhasználva állítottunk össze egy kvadkoptert, amelyen a saját algoritmusainkat szerettük volna futtatni. Az általunk fejlesztett állapotbecslő és szabályozó rendszereket ki tudtuk próbálni, a szabályozókat össze tudtuk hasonlítani.

1.2. A szakirodalomban található módszerek áttekintése

- A nemlineáris modell felépítése során jellemző megközelítés [17] [1] [11] a pozíció- és orientáció dinamika együttes felhasználása és a motorokra leképezhető alacsony szintű beavatkozó jelek használata. Mivel az orientációkövetés esetünkben egy külön szinten van megvalósítva szimulációban és az implementációban is, ezért a pozíció dinamikára fogunk szabályozásokat tervezni [12], trajektória menti linearizálás módszerével [21][23].
- Szabályozásra használatosak a PID szabályozó, optimális irányításon alapuló módszerek[5], modell-prediktív irányítás [4] [3] [24] [13], illetve egzakt linearizálás[12].
- Számos tesztkörnyezet létezik, például az ETH Zürich drónaréna [15], de több helyen is találhatóak repülési eredmények [3].
- Trajektóriatervezési módszerek közül a különböző spline alapúak [6] [28], [9], illetve a gyorsulás harmadik, negyedik deriváltját minimalizáló trajektóriatervezés [17] gyakori, utóbbi a differenciális simaságát használja ki a rendszernek.
- Az implementáció során alkalmazott kiegészítésekre, hardveres megvalósításra bővebben kitér [1] [13].

- Kitekintés, továbbfejlesztési lehetőségek [4] [26] [28]

- Implementációs megfontolások (MATLAB keretein belül) [10], [8] és ROS-sal[3] [28] [13]

1.3. Szószedet

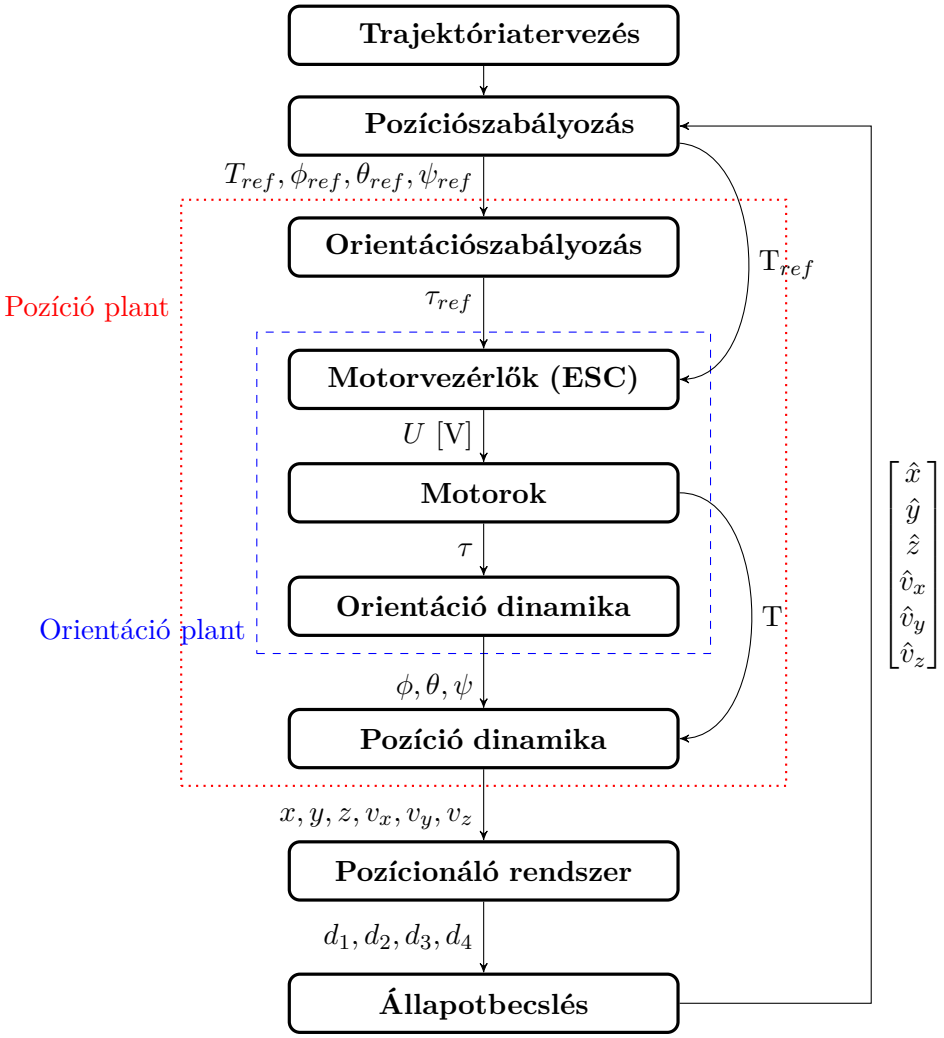
- orientáció (attitude / orientation): a kvadkopter Euler szögekkel (RPY, azaz ϕ, θ, ψ) kifejezett szöghelyzete
- hover: bedöntésmentes állapot (a ϕ és a θ szögek ekkor nullák), ilyenkor a kvadkopter egy helyben lebeg.
- autopilot – a kvadkopter autonóm repülését biztosító szabályozó
- UAV – pilóta nélküli légijármű (unmanned aerial vehicle)
- UWB – Ultra Wide Band, azaz széles frekvenciatartományú impulzusokat felhasználó pozícionáló rendszer
- LQ – lineáris kvadratikus optimális szabályozási feladat (lineáris, időinvariáns rendszer kvadratikus költségfüggvénnyel)
- B-Spline – bázis spline: polinom bázisok szorzata az azokat súlyozó kontrol pontokkal
- MPC – Modell-prediktív szabályozás
- LTV – lineáris, időben változó (variáns) rendszer
- Pixhawk – nyílt forráskódú hardver, többféle UAV irányítására használható fedélzeti egység.
- ardupilot – a pixhawk-on futó szoftver, ami pl. az orientációszabályozást, és egyes állapotbecslési feladatokat végzi
- Raspberry pi – egy egykártyás fedélzeti számítógép, ami alkalmas generált kód fordítására és futtatására, és a repülések során a fő kommuikációs interface-t biztosítja
- Simulink coder – simulink modellekből kifordítható kódot készít, ami alkalmas a raspberry pi-n történő futtatásra

[3] Usenko [28] munkájához rendelkezésre áll kódbázis, amivel ROS begyázott operációs rendszerben valósítható meg az akadályelkerülő on-line trajektóriatervezés

2. fejezet

Kvadkopter modellezése és irányítása

Mielőtt a modellalkotás és a szabályozótervezés feladatait sorra vennénk, áttekintem a teljes rendszer felépítését. Így a bemutatott eredményeket is könnyebben el tudjuk majd helyezni és össze tudjuk vetni a szakirodalomban szereplő többi munkával.



2.1. ábra. A kvadkopter tesztkörnyezet hatásvázlata

A modellezendő rendszer és az ahhoz tartozó irányítás 2.1. ábrán látható komponensei jelen diplomatervhez a következőkben ismertetettek szerint kapcsolódnak. A szabályozótervezés tárgya főként a pozíciószabályozás lesz, ezt a 3. fejezetben ismertetem részletesen. Az implementáció során kitérünk majd orientációszabályozáshoz kapcsolódó feladatokra is (5.2. alfejezet). A trajektóriatervező algoritmus (lásd 4. fejezet) a kvadkopter által bejárni kívánt pontok, az ahhoz tartozó sebességek és orientációk időfüggvényeit határozza meg.

A rendszer nem minden ábrán szereplő komponensének fogjuk figyelembe venni a viselkedését, így a motorok és az aerodinamikai hatások modellezésétől eltekintünk, csak a merev test dinamikával foglalkozunk. A motorok dinamikája ugyanis kellően gyors ahhoz, hogy feltételezzük, hogy azok beállási idői elhanyagolhatók az orientáció dinamikához képest [17][12]. Az orientációszabályozást pontos modellparaméterek ismeretének hiányában nem tudjuk szimulációban megtervezni, az orientáció modellt viszont röviden ismertetjük.

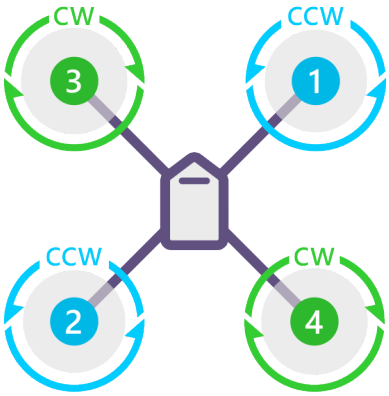
Részletesen foglalkozunk a pozíció dinamika modellezésével. Ennek állapotait egy állapotbecslő határozza meg, amely a kvadkopteren található IMU és egy beltéri pozícionáló rendszer méréseit használja. A pozícionáló rendszer egy munkatérben ismeretlen helyen lévő úgynevezett tag és 4 fixen elhelyezett bázispont (anchor) közötti távolságot méri. A szimuláció és a tervezés során feltételezzük, hogy a méréseink, és az ebből becsült pozíció és sebesség kellően pontosak.

2.1. A kvadkopter dinamikai modellezése

A pozíció és az orientáció dinamika együttesen egy 12 állapotú nemlineáris rendszerként írható le a 2.1 szerinti differenciálegyenletekkel. A modell szétbontható pozíció és orientáció dinamikára: a felső hat sor a pozíció dinamika $\begin{bmatrix} T & \phi & \theta & \psi \end{bmatrix}^T$ bemenetekkel, az alsó hat sor pedig az orientáció dinamika $\begin{bmatrix} \tau_x & \tau_y & \tau_z \end{bmatrix}^T$ bemenetekkel, amelyek a motorok kvadkopterre vonatkozó forgatónyomatékai.

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{p}_z \\ \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \\ -\left(\sin\phi \cdot \sin\psi + \cos\phi \cdot \cos\psi \cdot \sin\theta\right) \cdot \frac{T}{m} \\ -\left(\cos\phi \cdot \sin\psi \cdot \sin\theta - \cos\psi \cdot \sin\phi\right) \cdot \frac{T}{m} \\ -\cos\phi \cdot \cos\theta \cdot \frac{T}{m} + g \\ p + \sin\phi \cdot \tan\theta \cdot q + \cos\phi \cdot \tan\theta \cdot r \\ \cos\phi \cdot q - \sin\phi \cdot r \\ \frac{\sin\phi}{\cos\theta} \cdot q + \frac{\cos\phi}{\cos\theta} \cdot r \\ \frac{J_y - J_z}{J_x} \cdot q \cdot r + \frac{\tau_x}{J_x} \\ \frac{J_z - J_x}{J_y} \cdot p \cdot r + \frac{\tau_y}{J_y} \\ \frac{J_x - J_y}{J_z} \cdot p \cdot q + \frac{\tau_z}{J_z} \end{bmatrix} \tag{2.1}$$

Aszerint, hogy a világkoordináta-rendszert milyen irányokban vesszük fel, a (2.1) egyenletben más előjelekkel szerepelnek a gyorsulások és az orientációk közti együtthatók. Ahol a világkoordináta-rendszer z tengelye felfelé mutat (például [6], illetve [11] esetében), ott a gravitáció járuléka negatív lesz. Az általunk használt beltéri pozícionáló rendszer



2.2. ábra. A kvadkopter váz elrendezése és légcsavarjainak forgásiránya

koordináta-rendszeréhez igazodva viszont előnyösebb a [3] és [29] által alkalmazott alak, amilyen a fenti (2.1) egyenletben is szerepel.

Esetünkben a világkoordináta-rendszert NED alakban tekintjük, amellyel az x, y, z koordináták úgy alkotnak jobbsodrású rendszert, hogy a z tengely lefelé mutat[1]. Ha a munkatér alsó sarkához, a talaj magasságába rögzítjük a koordináta-rendszert, akkor a talajszint feletti pozícióknak negatív lesz a z koordinátája (például [3, 4. ábra]).

Az orientáció modell bemenetei a kvadkopterre ható τ nyomatékok. Ezekhez hozzárendelhetők a 2.2. ábra szerinti négy motorhoz tartozó ω szögsebességek, amelyeket a motorvezérlők (ESC, azaz Electronic Speed Controller) számára referenciaként kiadhatunk. Az ω szögsebességgel forgó motorok által kifejtett nyomatékot a (2.2) egyenlet adja meg, ahol k_F az erőkkel, k_M a kifejtett nyomatékokkal arányos együtthatók, L pedig a motorok távolsága a tömegközépponttól [17]. A modellben szereplő $J_{x,y,z}$ paraméterek a kvadkopter fő tehetetlenségi nyomatékai.

$$\begin{bmatrix} T \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} k_F & k_F & k_F & k_F \\ 0 & k_F L & 0 & -k_F L \\ -k_F L & k_F L & 0 & 0 \\ k_M & -k_M & k_M & -k_M \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \tag{2.2}$$

A fentebb ismertetett 12 állapotú modell helyett az implementációhoz egy egyszerűsített modellt alkalmaztunk, ugyanis az orientációszabályozást egy külön fedélzeti egység végezte. Így a paramétereink száma is jelentősen lecsökkent, mivel így a kvadkopter inerciáját nem kell külön megmérni, illetve kiszámítani. A trajektóriakövető szabályozás tervezéséhez is csupán a pozíció dinamika modelljét használtuk, amit a következőkben ismertetek.

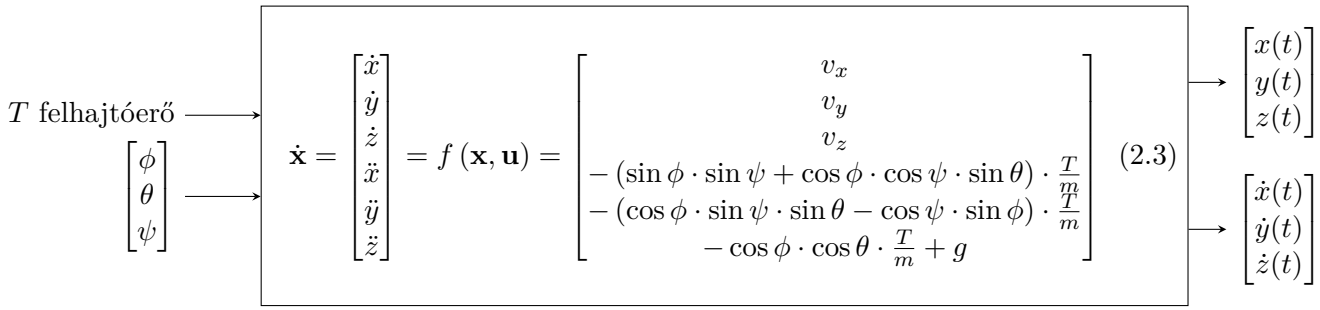
2.2. Pozíció dinamika

A modellezés során az egyszerűsítési lehetőségeket a következő megfontolás adja. A 2.1. ábrán látható irányítási rendszer egy kaszkád szabályozásként valósítható meg. Ekkor a trajektóriatervező algoritmus kimenete szolgál referenciaként a pozíciószabályozáshoz.

[1] A másik konvenció az ENU, azaz East North Up. Bővebben a két rendszer közötti különbségekről: `https://en.wikipedia.org/wiki/Axes_conventions` (angolul).

A pozíciószabályozás egy orientációt ad ki beavatkozó jelként. Az orientációszabályozás pedig a motorok forgási sebességét változtatja. Így látható egy hierarchia, miszerint az alacsonyabb szintű, gyorsabb dinamikájú rendszerekre külön-külön tervezhető szabályozás.

Ha egy kaszkád szabályozási rendszerben a belső rendszerek dinamikája jelentősen gyorsabb a külsőknél, akkor a belső rendszerek átvitelét el lehet hanyagolni. Mivel a motorok beállási ideje kellően gyors, az orientációszabályozó kimenetén kiadott τ nyomatékokra úgy tekinthetjük, hogy azokat tökéletesen előállítják a motorok. Ezzel a motorok dinamikáját elhanyagoltuk, a modellezés során ezzel már nem kell foglalkoznunk. Hasonlóan feltételezzük azt is, hogy a pozíciószabályozás beavatkozó jeleként számított orientációkat képes a kvadkopter kellő pontossággal lekövetni, amihez a kvadkopter fedélzeti egységén egy orientációbecslő és egy hangolható szabályozó is rendelkezésre áll.



2.3. ábra. A pozíció modell be- és kimenetei

A felhasznált pozíció dinamika egy mindössze hat állapotú modellt eredményez. Ez viszonylag könnyen kezelhető, és megérthető.

Ez a hat állapotú modell még mindig nemlineáris, lineáris szabályozás tervezéséhez célszerű linearizálni. A következőkben [23] alapján bevezetjük a hibadinamikát, ami [6]-ban felhasznált módszer. Ennek segítségével nem csak egy pontbeli, hanem trajektória menti modelleket is elő tudunk majd állítani. Először általános jelöléseket vezetünk be, majd a (2.3) egyenlettel adott kvadkopter modellre is alkalmazzuk az ismertetett eszköztárat.

2.2.1. Egyensúlyi pont

Legyen egy nemlineáris rendszer alakja következő:

$$\dot{x}(t) = f\left(x(t), u(t)\right) \tag{2.4}$$

ahol f leképezés $\mathbf{R}^n \times \mathbf{R}^m \mapsto \mathbf{R}^n$, azaz a rendszernek n bemenete és m állapota van. Egy $\overline{x} \in \mathbf{R}^n$ pontot egyensúlyi pontnak nevezünk, ha létezik hozzá egy $\overline{u} \in \mathbf{R}^m$ egyensúlyi bemenet, és teljesül, hogy

$$f\left(\overline{x}, \overline{u}\right) = 0_n. \tag{2.5}$$

Legyen $\overline{x}$ egyensúlyi pont a hozzá tartozó $\overline{u}$ egyensúlyi bemenettel, és tekintsünk a 2.4 szerinti rendszert $x(t_0) = \overline{x}$ állapotából indítva, $u(t) \equiv \overline{u}$ bemenettel. Ekkor minden $t \geq t_0$-ra teljesülni fog, hogy

$$x(t) = \overline{x}$$

Ezért nevezzük $\overline{x}$-t egyensúlyi pontnak.

2.2.2. Hibadinamika előállítása Jacobi-linearizálással

Látható, hogy ha a (2.4) rendszert az egyensúlyi pontjából indítjuk, akkor az onnan nem fog eltávolodni. Viszont vizsgáljuk azt az esetet, amikor valamilyen eltéréssel indulunk $\overline{x}$ állapotól, és az $\overline{u}$ bemenettől eltérően gerjesztjük a rendszert! Tulajdonképpen az történik, hogy az állapotokat és a bemeneteket eltoljuk az egyensúlyi pontba.

$$\begin{aligned} \delta_x &\coloneqq x(t) - \overline{x} \\ \delta_u &\coloneqq u(t) - \overline{u} \end{aligned} \tag{2.6}$$

A (2.4) egyenletbe beírva a (2.6) átrendezett alakját a következő összefüggést kapjuk:

$$\dot{\delta}_x(t) = f(\overline{x} + \delta_x(t), \overline{u} + \delta_u(t))$$

A fenti egyenletet első rendű Taylor-sorával közelítjük.

$$\dot{\delta}_x(t) \approx f(\overline{x}, \overline{u}) \quad + \quad \left.\frac{\partial f}{\partial x}\right|_{\substack{x=\overline{x} \\ u=\overline{u}}} \delta_x(t) \quad + \quad \left.\frac{\partial f}{\partial u}\right|_{\substack{x=\overline{x} \\ u=\overline{u}}} \delta_u(t)) \tag{2.7}$$

Felhasználva (2.5) egyenletet, látható, hogy $f(\overline{x}, \overline{u})$ kiesik, így az alábbi alakú lesz:

$$\dot{\delta}_x(t) \approx \left.\frac{\partial f}{\partial x}\right|_{\substack{x=\overline{x} \\ u=\overline{u}}} \delta_x(t) + \left.\frac{\partial f}{\partial u}\right|_{\substack{x=\overline{x} \\ u=\overline{u}}} \delta_u(t))$$

Bevezethetjük a következő jelöléseket:

$$A = \left.\frac{\partial f}{\partial x}\right|_{\substack{x=\overline{x} \\ u=\overline{u}}} \in \mathbf{R}^{n \times n} \qquad\qquad B = \left.\frac{\partial f}{\partial u}\right|_{\substack{x=\overline{x} \\ u=\overline{u}}} \in \mathbf{R}^{n \times m} \tag{2.8}$$

Amivel a hibadinamika egy lineáris rendszer lesz, ha a 2.7 közelítés igaz. Emiatt $\overline{x}$ és $\overline{u}$-tól való kis eltérések esetén érvényes a lineáris modell:

$$\dot{\delta}_x(t) \approx A\delta_x(t) + B\delta_u(t))$$

2.3. A kvadkopter hibadinamikája

A kvadkopter (2.3.) szerinti nemlineáris modelljére vonatkozó egyensúlyi pont a lebegés (hover). Ennek során $\overline{u} = \begin{bmatrix} T_0 & 0 & 0 & \psi_0 \end{bmatrix}^T$ bemenet és $\overline{x} = \begin{bmatrix} x_0 & y_0 & z_0 & 0 & 0 & 0 \end{bmatrix}^T$ állapot tartozik a hover-hez. Ezeket felhasználva (2.8) alapján a lineáris hibamodell a következő:

$$A = \left.\frac{\partial f(x,u)}{\partial x}\right|_{\substack{x=\overline{x}\\u=\overline{u}}} = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial z} & \frac{\partial f_1}{\partial v_x} & \frac{\partial f_1}{\partial v_y} & \frac{\partial f_1}{\partial v_z} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} & \frac{\partial f_2}{\partial z} & \frac{\partial f_2}{\partial v_x} & \frac{\partial f_2}{\partial v_y} & \frac{\partial f_2}{\partial v_z} \\ \frac{\partial f_3}{\partial x} & \frac{\partial f_3}{\partial y} & \frac{\partial f_3}{\partial z} & \frac{\partial f_3}{\partial v_x} & \frac{\partial f_3}{\partial v_y} & \frac{\partial f_3}{\partial v_z} \\ \frac{\partial f_4}{\partial x} & \frac{\partial f_4}{\partial y} & \frac{\partial f_4}{\partial z} & \frac{\partial f_4}{\partial v_x} & \frac{\partial f_4}{\partial v_y} & \frac{\partial f_4}{\partial v_z} \\ \frac{\partial f_5}{\partial x} & \frac{\partial f_5}{\partial y} & \frac{\partial f_5}{\partial z} & \frac{\partial f_5}{\partial v_x} & \frac{\partial f_5}{\partial v_y} & \frac{\partial f_5}{\partial v_z} \\ \frac{\partial f_6}{\partial x} & \frac{\partial f_6}{\partial y} & \frac{\partial f_6}{\partial z} & \frac{\partial f_6}{\partial v_x} & \frac{\partial f_6}{\partial v_y} & \frac{\partial f_6}{\partial v_z} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.9a}$$

$$B = \left.\frac{\partial f(x,u)}{\partial u}\right|_{\substack{x=\overline{x}\\u=\overline{u}}} = \begin{bmatrix} \frac{\partial f_1}{\partial T} & \frac{\partial f_1}{\partial \phi} & \frac{\partial f_1}{\partial \theta} & \frac{\partial f_1}{\partial \psi} \\ \frac{\partial f_2}{\partial T} & \frac{\partial f_2}{\partial \phi} & \frac{\partial f_2}{\partial \theta} & \frac{\partial f_2}{\partial \psi} \\ \frac{\partial f_3}{\partial T} & \frac{\partial f_3}{\partial \phi} & \frac{\partial f_3}{\partial \theta} & \frac{\partial f_3}{\partial \psi} \\ \frac{\partial f_4}{\partial T} & \frac{\partial f_4}{\partial \phi} & \frac{\partial f_4}{\partial \theta} & \frac{\partial f_4}{\partial \psi} \\ \frac{\partial f_5}{\partial T} & \frac{\partial f_5}{\partial \phi} & \frac{\partial f_5}{\partial \theta} & \frac{\partial f_5}{\partial \psi} \\ \frac{\partial f_6}{\partial T} & \frac{\partial f_6}{\partial \phi} & \frac{\partial f_6}{\partial \theta} & \frac{\partial f_6}{\partial \psi} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -sin\psi_0 \cdot \frac{T}{m} & -cos\psi_0 \cdot \frac{T}{m} & 0 \\ 0 & cos\psi_0 \cdot \frac{T}{m} & -sin\psi_0 \cdot \frac{T}{m} & 0 \\ -\frac{1}{m} & 0 & 0 & 0 \end{bmatrix} \tag{2.9b}$$

A hibadinamika mátrixai az x_0, y_0, z_0-tól nem függnek. A B mátrix viszont függ a yaw szögtől. A parciális deriváltakban az f_i-k az f sorait jelentik. A modell bemenetei $\delta_u = \begin{bmatrix} \delta T & \delta\phi & \delta\theta & \delta\psi \end{bmatrix}^T$, a kimenetei $\delta x = \begin{bmatrix} \delta x & \delta y & \delta z & \delta\dot{x} & \delta\dot{y} & \delta\dot{z} \end{bmatrix}^T$.

Feltesszük, hogy a kopter yaw szöge (heading) konstans nullában van. Ez csak egy *ideiglenes* megszorítás, a későbbiekben tárgyalni fogjuk azt az esetet is, amikor a heading is változhat.

Ha a kvadkopter pozícióban tartása a cél, adja magát a hover állapot, mint munkapont felvétele, és ebben a pontban történő linearizálása a 2.3. egyenlet szerinti dinamikának. A 4. fejezetben megtervezett trajektória kielégíti ennek a rendszernek egyenleteit (ez következik a 2.10. egyenletből). Ezért itt egyensúlyi pont helyett egy trajektóriát fogunk jelölni a korábbi $\overline{x}$ és $\overline{u}$ jelölésekkel.

Egy t_0 időpillanatban az $x_0 = \overline{x}(t_0)$ állapotból indulva, időben előrehaladva $u(t) = \overline{u}(t)$ beavatkozójelet alkalmazva minden $\tau \leq t$-re igaz, hogy

$$\dot{\overline{x}}(\tau) = f(\overline{x}(\tau), \overline{u}(\tau)) \tag{2.10}$$

azaz a rendszer $\overline{u}$ trajektóriához tartozó bemenőjel esetén végighalad $\overline{x}$ állapotokon. A (2.9) szerinti linearizálást ezért elvégezhetjük a trajektória mentén is [4] [21], így a hibamodellt megkaphatjuk a trajektória pontjaiban.

Arra számítunk, hogy trajektória mentén előállított modellsereget alkalmazva egy LTV szabályozás a zavarásokat jobban kompenzálni tudja, mint egy olyan szabályozó, amely csak a lebegés mellett linearizált modellt használja.

2.4. Névleges irányítás

Felmerülhet a kérdés, hogy milyen $\overline{u}(t)$ beavatkozójel szükséges ahhoz, hogy a kvadkopter a $\overline{x}(t)$ trajektória mentén végighaladjon. Ezt számíthatjuk ki a névleges irányítással, és ez a névleges irányítás használható fel a trajektória menti linearizáláshoz.

$$\begin{aligned}
\overline{\phi} &= arcsin\left(\frac{\ddot{x}\sin\psi - \ddot{y}\cos\psi}{\sqrt{\ddot{x}^2 + \ddot{y}^2 + (g-\ddot{z})^2}}\right) \\
\overline{\theta} &= arctan\left(-\frac{\ddot{x}\ \cos\psi + \ddot{y}\sin\psi}{g-\ddot{z}}\right) \\
\overline{T} &= m\cdot\frac{g-\ddot{z}}{\cos\overline{\phi}\cos\overline{\theta}}
\end{aligned} \tag{2.11}$$

Ha a yaw szöget nullának választjuk meg ($\overline{\psi} = 0$), számos egyszerűsítést tehetünk, a névleges irányítás ekkor a következőképp számolható:

$$\begin{aligned}
\overline{\phi} &= arctan\left(-\frac{\ddot{y}}{g-\ddot{z}}\right) \\
\overline{\theta} &= arctan\left(-\frac{\ddot{x}\ cos\overline{\phi}}{g-\ddot{z}}\right) \\
\overline{T} &= m\cdot\frac{g-\ddot{z}}{\cos\overline{\phi}\cos\overline{\theta}}
\end{aligned} \tag{2.12}$$

A (2.6) szerinti jelölést alkalmazva a névlegestől való eltérések értelmezhetők a kvadkopter rendszerre is, a bemeneten $\delta_u = \begin{bmatrix}\delta T & \delta\phi & \delta\theta & \delta\psi\end{bmatrix}^T$, a kimeneten pedig $\delta x = \begin{bmatrix}\delta x & \delta y & \delta z & \delta\dot{x} & \delta\dot{y} & \delta\dot{z}\end{bmatrix}^T$ jelölésekkel:

$$\begin{bmatrix}T\\ \phi\\ \theta\\ \psi\end{bmatrix} = \begin{bmatrix}\overline{T}\\ \overline{\phi}\\ \overline{\theta}\\ \overline{\psi}\end{bmatrix} + \begin{bmatrix}\delta T\\ \delta\theta\\ \delta\phi\\ \delta\psi\end{bmatrix} \tag{2.13a}$$

Ideális esetben a (2.11) egyenletek szerint kiszámolt névleges beavatkozó jelek visszacsatolás nélkül (open loop) végigvinnék azt a trajektória mentén. Ez csak akkor történhetne meg, ha semmilyen modellbizonytalansággal nem kellene számolni, illetve zavarás, vagy

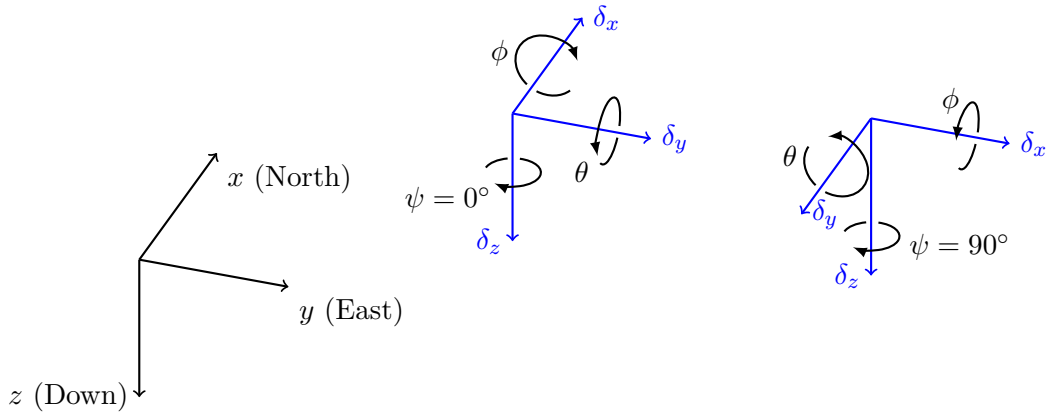
offsethiba nem terhelné a ki- és bemeneteket. Mivel ilyen helyzet igen valószínűtlen (különösen a gyakorlatban), a pozíciószabályozók feladata lesz az, hogy a δ_u beavatkozójel-korrekciókat kiszámítsák, biztosítva ezzel a trajektóriakövetést: ide értve a zavarelnyomást, beavatkozó jelekre ható offsethibák kompenzálását, és az állapotbecslés zajos mérései melletti működést.

2.5. Koordináta-rendszerek

A hibamodell bevezetése után fontos kitérni arra, hogy milyen koordináta-rendszereket tudunk a kvadkopter környezetben megkülönböztetni.

- a trajektóriatervező algoritmus gyorsulásokat számol ki a megadott pontok mentén, amelyek mind NED-ben, azaz világkoordinátákban értelmezettek. Az aktuális ψ szög ismeretében számíthatóak a névleges orientációk és T thrust (felhajtóerő)
- a világkoordinátákhoz képesti forgatásokat a kvadkopter test koordináta rendszerébe egy R rotációs mátrix adja meg [6].
- a beltéri pozícionáló rendszer mérései NED-ben adnak meg pozíciót, sebességet.
- Fontos különbség, hogy mivel a gravitáció a NED szerinti koordináta-rendszerben függőlegesen lefelé hat, a test koordináta-rendszerben függőlegesen felfelé ható felhajtóerő (thrust) $\phi \neq 0$ vagy $\theta \neq 0$ esetben szöget zár be a gravitációval, így azt (2.11)-ben látható módon meg kell növelni.

2.6. Modell SISO alakja



2.4. ábra. Kvadkopter viselkedése $\psi = 0°$ és $\psi = 90°$ mellett

A kvadkopter hover helyzetét munkapontnak választva linearizálhatjuk a rendszert. Ennek során egy olyan rendszert kapunk, amelyben egy kimenetre csak egy bemenet hat: az x irányú mozgását a θ pitch szöggel, az y iránybeli mozgást a ϕ roll szöggel, a magasság z koordinátáját pedig a thrust bemenőjellel lehet befolyásolni. Ez egy szétcsatolt rendszer és a három pozíció kimenettel felfoghatjuk 3 egybemenetű és egykimenetű rendszerként. Ilyen rendszerekre lehetséges PID szabályozót tervezni.

Nem szabad viszont megfeledkezni arról, hogy a rendszer szétcsatoltsága csak $\overline{u} = \begin{bmatrix} T_0 & 0 & 0 & \psi_0 \end{bmatrix}^T$ szerinti hover-ben (lebegés) és annak egy kis környezetében érvényes. A yaw szög hatása könnyedén észrevehető: ha a z tengely mentén elforgatjuk a kvadkoptert, egy roll gerjesztés már nem csak a hover-ben látott y tengely menti elmozdulást eredményez. Ha 90°-kal forgatnánk el a rendszert, akkor a ϕ szög a $-x$, a θ pedig $-y$ irányban hatna, ahogyan az a 2.4. ábrán látható.

A ψ szöget állandónak, ez esetben $\psi = 0$-nak feltételezzük, így lehet 3 db lineáris, SISO modellünk, a következő alakban:

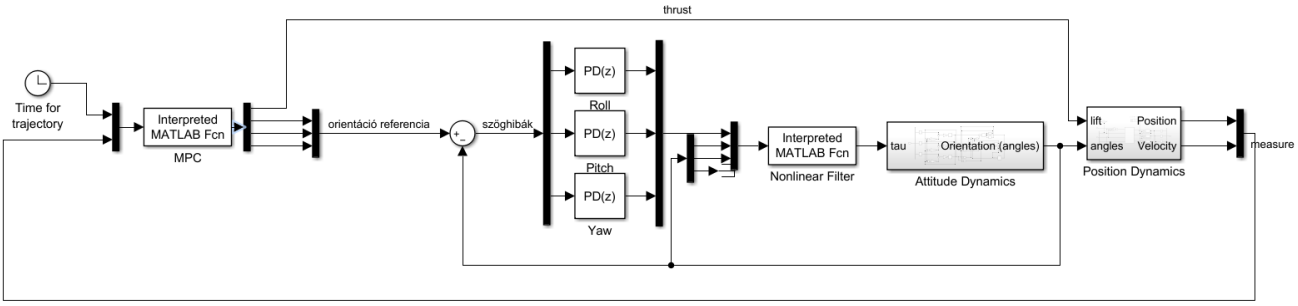
$$\begin{aligned} \begin{bmatrix} \dot{\delta}_x \\ \ddot{\delta}_x \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \delta_x \\ \dot{\delta}_x \end{bmatrix} + \begin{bmatrix} 0 \\ g \end{bmatrix} \cdot \delta_\theta \\ \delta_x &= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \delta_x \\ \dot{\delta}_x \end{bmatrix} \end{aligned} \tag{2.14}$$

$$\begin{aligned} \begin{bmatrix} \dot{\delta}_y \\ \ddot{\delta}_y \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \delta_x \\ \dot{\delta}_y \end{bmatrix} + \begin{bmatrix} 0 \\ g \end{bmatrix} \cdot \delta_\psi \\ \delta_y &= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \delta_y \\ \dot{\delta}_y \end{bmatrix} \end{aligned} \tag{2.15}$$

$$\begin{aligned} \begin{bmatrix} \dot{\delta}_y \\ \ddot{\delta}_y \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \delta_x \\ \dot{\delta}_y \end{bmatrix} + \begin{bmatrix} 0 \\ -\frac{1}{m} \end{bmatrix} \cdot \delta_\psi \\ \delta_y &= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \delta_y \\ \dot{\delta}_y \end{bmatrix} \end{aligned} \tag{2.16}$$

2.7. Szimuláció

A kvadkopter (2.1) szerinti 12 állapotú modellje Simulinkben adott, az orentáció és pozíció alrendszerei külön blokkban találhatók meg. A szabályozáshoz kaszkád struktúrát használunk. A belső szabályozási körben az orientációszabályozás található, a külső hurokban pedig a trajektóriakövető szabályozás a 2.5. ábrán látható formában.



2.5. ábra. A pozíció- és orientációszabályozás kaszkád alakja

Az orientációkövető szabályozás szimulációs környezetben

A belső szabályozási kör nyomaték beavatkozó jeleit PD szabályozók állítják elő. Ez a kör biztosítja a pozíció alrendszerünk számára, hogy a kvadkopter a magasabb szintű szabályozó által kiadott orientációba álljon, és megvalósulhasson a trajektóriakövetés.

Az orientáció dinamika egy nemlineáris szétcsatoló szűrővel kettős integrátorokra bontható, az egyes csatornák PD szabályozóval szabályozhatók.

A szabályozási kör ezen modelljét nem használjuk ki a tervezés során, de szimulációban hasznos lehet annak validálására, hogy például egy nyomaték jellegű zavarást hogyan tud kompenzálni a szabályozónk.

A referencia trajektória tervezése a szabályozásra is nagy hatással van, ugyanis például sima trajektória követéséhez ugyanakkora beavatkozó jelek mellett jobb zavarelnyomás biztosítható. Megfelelően megtervezett trajektóriával jól ki tudjuk használni az eszközeink képességeit, és jobban elláthatók az elvégzendő feladatok, például időoptimális trajektóriák használatával.

3. fejezet

Irányítások tervezése szimulációban

Az alábbi irányításokat mind a hibadinamikára tervezzük.

Regulator probléma

A cél az, hogy a kvadkopter egy helyben lebegjen, egy előírt pontban. Ezt hover állapotnak fogjuk nevezni. A szabályozó a kopterre ható zavaró hatásokat igyekszik ellensúlyozni, igykekszik egy helyben tartani azt. Ha a kopter hover állapotban van, akkor sebessége nulla. De ez nem elég, nullának is kell maradnia, ezért hover állapotban a névleges roll és pitch szögek nullák (a yaw bármilyen értéket felvehet).

Ideális esetben a hover állapot tehát azt jelenti, hogy a kiadott roll / pitch szög nulla.

Alapjel követése

A felszállás és leszállás manővereit szűrt alapjellel valósítottuk meg az implementációhoz.

Trajektóriakövetés

Előírt trajektória lekövetéséhez a kvadkopter bemeneteihez hozzáadtuk a névlegesen a trajektória bejárásához szükséges beavatkozó jeleket, a mért bemenetekből pedig kivontuk a névleges pozíciókat és sebességeket.

3.1. SISO irányítás: PID

A PID szabályozások rendkívül széles körben használatosak. Előnyük, hogy kipróbált, biztosan működő megoldásokat adnak. Tervezése és a hangolása nagyon könnyű, lépésről lépésre elvégezhetőek.

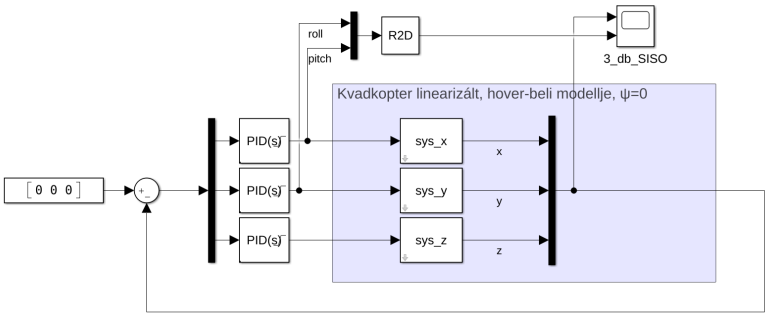
Performanciájuk jó viszonyítási alapot ad a többi szabályozó teljesítményéről ezzel összevetve [29]. Nem szükséges modell sem a tervezéshez, ha kísérleti alapon hangoljuk.

Egy behangolt szabályozás megmutatja, hogy egy alkalmazás során a kopter milyen beavatkozójel-tartományban működik. Ez pedig kiinduló alapot jelenthet más szabályo-

zók hangolásánál, egyszerűsége miatt fényt deríthet a szimuláció vagy az implementációs környezet egyéb hibáira.

3.1.1. Tervezés

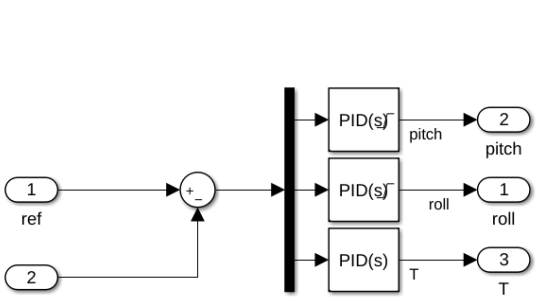
A tesztelés első lépésben csak a (2.14) – (2.16) hibadinamika felhasználásával történik, az egyes alrendszerekre beállított nem nulla kezdőállapotokkal: azt vizsgáljuk, hogy a hibadinamikát a szabályozó milyen gyorsan képes nullába irányítani. A beavatkozó jelekre 5°-os szaturációt adtunk meg.



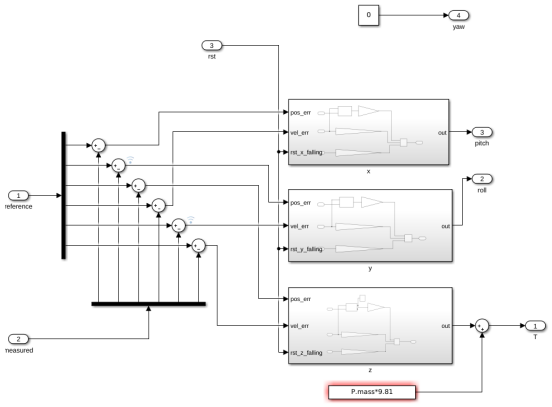
3.1. ábra. PID szabályozók tesztelése a lineáris hibadinamikán

3.1.2. Performancia

Ezeket a SISO modellekre tervezzük. Két lehetséges struktúrája (realizációja) az alábbi:



(a) PID blokk



(b) PID a becsült sebesség felhasználásával

3.2. ábra. PID realizációk csatornánként

A 3.2b. ábrán látható alak a mért sebességeket használja fel.

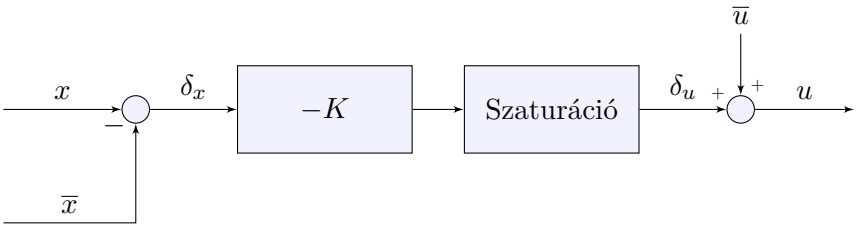
3.1.2.1. Kompenzáció a ψ szög eltérésére

Habár a kvadkoptert a $\psi_0 = 0°$-os referencia szöghelyzetet (heading-ként, azaz tájolásként is hivatkoznak rá) mellett linearizáltuk, mégis elképzelhető, hogy a kvadkopter elfordul a

repülés során, pl. zavarások miatt. Ekkor a roll és pitch szögek kompenzáció nélküli kiadása jelentősen ronthatja a szabályozó teljesítményét. Ezért [1, (56) egyenlet] a (3.1) szerinti transzformációt javasolja, biztosítva így azt, hogy a rendszer szétcsatoltsága megmaradjon.

$$\begin{bmatrix} \delta_\phi^\psi \\ \delta_\theta^\psi \end{bmatrix} = \begin{bmatrix} cos(\psi_{mért}) & -sin(\psi_{mért}) \\ sin(\psi_{mért}) & cos(\psi_{mért}) \end{bmatrix} \begin{bmatrix} \delta_\phi \\ \delta_\theta \end{bmatrix} \tag{3.1}$$

3.2. MIMO irányítás: LQ



3.3. ábra. LQ szabályozó a hibadinamikára tervezve

Az LQ szabályozó tervezéséhez MIMO modellt használtunk, amit hover-ben linearizáltunk, majd diszkretizáltunk.

Az optimális irányítások elméletének [2] [16] felhasználásával egy állapotvisszacsatolást tervezünk, így a szabályozó szimulációja és implementációja egyszerű.

A költségfüggvényben szereplő súlyok fizikai tartalommal bírnak, így azt várjuk, hogy a hangolások eredményei az adatsorokban szemmel láthatóan megjelennek. A megengedhető beavatkozójel és hibajelek abszolút értékéhez kiindulási alapul vehetjük a PID szabályozó szimulációjakor kapottakat. (A behangolt szabályozó súlyai pedig jó alapot jelentenek az MPC számára.)

3.2.0.1. Tervezés

Cél az alábbi költségfüggvény minimalizálása

$$J = \sum_{k=0}^{N-1} \left(x_k^T Q x_k + u_k^T R u_k + 2 x_k^T N u_k \right) \tag{3.2}$$

Amihez a MATLAB `lqr` parancsa kiszámít egy statikus K mátrixot, ami által ez egy állapotvisszacsatolás, és az optimális irányítást adja.

$$u^{opt} = -Kx \tag{3.3}$$

Ahol K a következőből adódott a (2.9) MIMO rendszer diszkretizált alakjának felhasználásával:

```
[K,S,CLP]=dlqr(sys.A,sys.B,Q,R);
```

A súlyok felvétele [19] alapján az állandósult állapotban megengedhető hibák és beavatkozójelek alapján történt:

$$e_{x,y}^{max} = 0.025m \qquad e_{v_x,v_y}^{max} = 0.4\frac{m}{s} \tag{3.4a}$$

$$q_1 = q_2 = \frac{1}{(e_x^{max})^2} \qquad q_4 = q_5 = \frac{1}{(e_{v_x,v_y}^{max})^2} \tag{3.4b}$$

$$e_z^{max} = 0.05m \qquad e_{v_z}^{max} = 0.15\frac{m}{s} \tag{3.4c}$$

$$q_3 = \frac{1}{(e_z^{max})^2} \qquad q_6 = \frac{1}{(e_z^{max})^2} \tag{3.4d}$$

$$Q = \begin{bmatrix} q_1 & & \\ & \ddots & \\ & & q_6 \end{bmatrix} \tag{3.4e}$$

$$\delta T^{max} = 1 \qquad \delta\phi^{max} = \delta\theta^{max} = \delta\psi^{max} = 0.5^\circ \tag{3.4f}$$

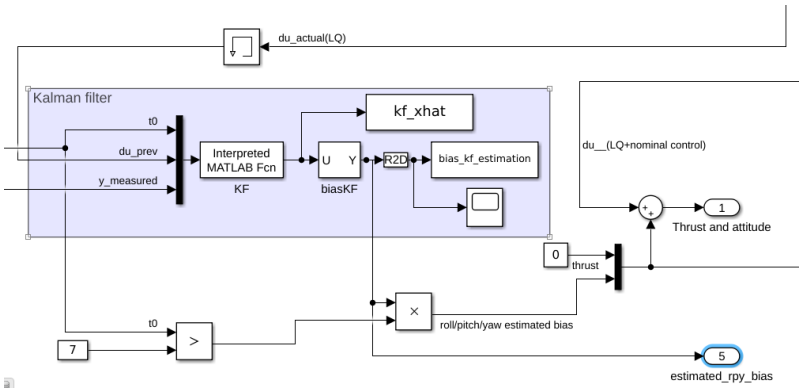
$$r_1 = \frac{1}{(\delta T^{max})^2} \qquad r_2 = r_3 = r_4 = \frac{1}{(\delta\phi/\delta\theta/\delta\psi^{max})^2} \tag{3.4g}$$

$$R = \begin{bmatrix} q_1 & & \\ & \ddots & \\ & & q_4 \end{bmatrix} \tag{3.4h}$$

Ahhoz, hogy tranziensek során ne adjunk ki túl nagy beavatkozó jeleket, a szabályozó kimenetét szaturálni kellett. A beavatkozójeleket összevetettük a behangolt és letesztelt PID-del, hogy nagyságrendileg hasonlóak legyenek.

Performancia

Szimulációban a kimenetekre ugrás zavarást adva a szabályozás nem volt képes maradó hiba nélküli referenciakövetésre. Valamiféleképpen szükséges volt ennek a kiküszöbölése, ezért egy Kalman-szűrőt terveztem a zavarás becslésére. A szimulációs eredmények a 3.5. ábrán láthatóak.

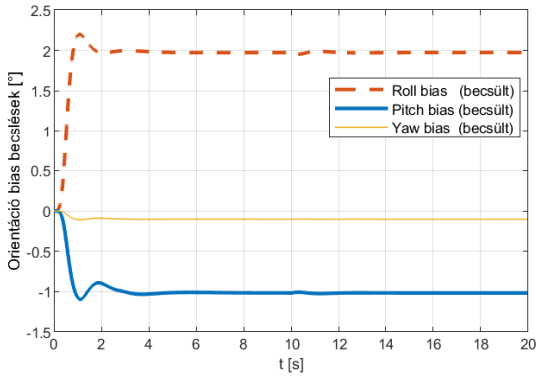


3.4. ábra. Kalman szűrővel kiegészített LQ szabályozás szimulációja

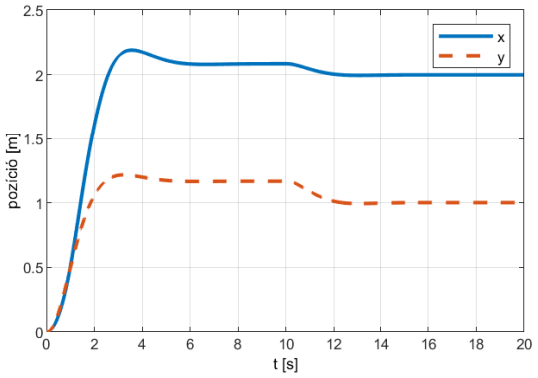
Kalman-szűrő az offsethibák becslésére

Az orientáció bias becslésére a következő kiterjesztett rendszert vettem fel. Mivel hover állapotban szeretnénk tartani a koptert, ezt vettem fel munkapontként, és itt linearizáltam.

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \\ \dot{b}_\phi \\ \dot{b}_\theta \\ \dot{b}_\psi \end{bmatrix} = \left[\begin{array}{cccccc|ccc} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -g & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & g & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ b_\phi \\ b_\theta \\ b_\psi \end{bmatrix} + \left[\begin{array}{cccc} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -g & 0 \\ 0 & g & 0 & 0 \\ -\frac{1}{m} & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right] \begin{bmatrix} T \\ \phi \\ \theta \\ \psi \end{bmatrix} \tag{3.5}$$



(a) Kalman szűrő által becsült orientáció offsetek



(b) Ha az LQ szabályozó kimenetén zavarás jelenik meg, az maradó pozícióhibát okoz.

3.5. ábra. Kalman szűrővel kiegészített LQ szabályozás működése szimulációban. A 3.4. ábrán látható módon a bal oldalon látható becsült offsetekkel 10 másodperc után kompenzáltuk a kimeneten ható zavarást, így a maradó pozícióhiba eltűnt.

3.3. MPC

Legyen egy általános LTI MIMO rendszer, melynek diszkrét idejű alakja a következő:

$$x_{k+1} = A \cdot x_k + B \cdot u_k \tag{3.6}$$
$$y_k = C \cdot x_k \tag{3.7}$$

A beavatkozó jel $u \in \mathbb{R}^n$, az állapotok száma $x \in \mathbb{R}^m$, így $A \in \mathbb{R}^{m \times m}$ négyzetes mátrix, illetve $B \in \mathbb{R}^{m \times n}$ (m sora és n oszlopa van).

Szeretnénk az aktuális x_k kezdeti állapot és egy $[u_k \quad u_{k+1} \quad \dots \quad u_{k+h-1}]^T$ beavatkozójel-sorozat mellett prediktálni a rendszer kimenetét, h hosszú horizonton. Ehhez kiszámítjuk az $\mathbf{x}_k = [x_k \quad x_{k+1} \quad \dots \quad x_{k+h}]^T$ állapotokat.

A fenti $\mathbf{u}$ beavatkozójel-sorozat mellett a prediktált állapotok a következőképp írhatók:

$$\mathbf{x}_k = \mathcal{M} x_k + \mathcal{C} \mathbf{u}_k \tag{3.8}$$

Ahol a mátrixok előállíthatók a következőképpen:

$$\mathcal{M} = \begin{bmatrix} I \\ A \\ A^2 \\ \vdots \\ A^h \end{bmatrix}, \qquad \mathcal{C} = \left[\begin{array}{cccc} 0 & 0 & \dots & 0 \\ \hline B & & & \\ AB & B & & \\ \vdots & \vdots & \ddots & \\ A^{h-1}B & A^{h-2}B & \dots & B \end{array}\right] \tag{3.9}$$

Az alábbi költséggel (x_0-ból x_k-ba):

$$J(\mathbf{u}, x_0) = \sum_{k=0}^{N-1} \left(x_k^T Q x_k + u_k^T R u_k \right) \tag{3.10}$$

QP alakban felírható az optimalizálási feladat: (ez csak $\mathbf{u}$-tól függ)

$$J(\mathbf{u}) = \mathbf{u}_k^T H \mathbf{u}_k + 2 x_k^T F^T \mathbf{u}_k + x_k^T G x_k \tag{3.11}$$

ahol az egyes tagok [7] alapján:

$$\begin{aligned} H &= \mathcal{C}^T \tilde{Q} \mathcal{C} + \tilde{R} \\ F &= \mathcal{C}^T \tilde{Q} \mathcal{M} \\ G &= \mathcal{M}^T \tilde{Q} \mathcal{M} \end{aligned} \tag{3.12}$$

Amivel a `quadprog` MATLAB utasítás segítségével megoldható alakra hoztuk a feladatot $f = 2x_k^T F^T$ helyettesítéssel:

$$\begin{array}{ll} \min\limits_{\mathbf{u}} & \mathbf{u}^T H \mathbf{u} + 2 f^T \mathbf{u} \\ \text{s.t.} & A_c \mathbf{du} \quad \leq \quad b_c \end{array} \tag{3.13}$$

Az optimalizálás során kapott $\mathbf{u}$ beavatkozójel-sorozat első elemét adjuk ki az irányított szakaszra, majd a szabályozó mintavételi idejének megfelelően minden mintavételkor újra elvégezzük az optimalizációt. (A szakaszmodell mintavételi ideje lehet azonos a szabályozó mintavételi idejével, de a kettő nem feltétlenül kell, hogy megegyezzen [4, 88. dia].)

4. fejezet

Trajektóriatervezés

4.1. Célja

Adott pontokon való áthaladás esetére a minimális eljutási idő és az ehhez tartozó optimális mozgásprofil keresése.

A generált B-Spline trajektóriánál a megadott pontokhoz az időbélyegek automatikusan generálódnak. Két dolog miatt fontos ez: az előre megadott pontok közti szakaszokat különböző idők alatt futhassuk be, illetve a teljes trajektória megtételéhez szükséges időt ne kelljen előre megmondani. A korlátozásainkat[1] a maximális beavatkozó jelekre írjuk elő[2]: a kvadkopter névleges roll és pitch szögét limitáljuk egy kitérés érték alá.

A névleges értékeket zavarásmentes esetben lehet elérni, amikor is egyéb szabályozás nélkül is végig lehet vezetni a kvadkoptert a trajektória mentén, az előírt trajektória menti gyorsulások ismeretében.

A korábbi trajektóriatervező algoritmus nem felelt meg ezen elvárásoknak, így annak alapjaira készült el a B-Spline trajektóriatervezést használó modul.

A trajektóriatervezés lépései

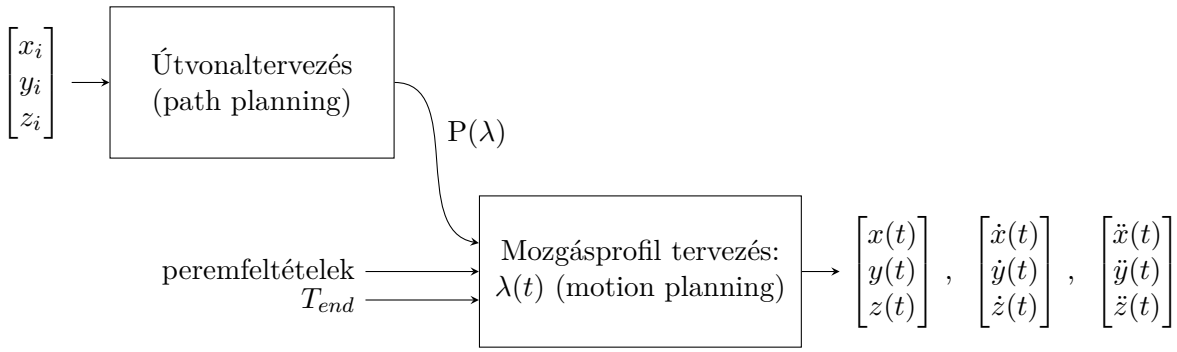
1. Pozíció (útvonaltervezés)
2. Mozgásprofil (= sebességprofil, időskálázás)
3. Kiértékelés
4. Trajektória menti gyorsulások számítása a névelges irányításhoz
5. Trajektória menti linearizálás az LTV MPC szabályozáshoz

[1] Mivel több optimalizálási fázisa is van a trajektóriatervezésnek, ezért a különböző lépésekben egyszerre csak egy paraméterrel foglalkozunk. Beavatkozó jelek korlátozása a T_{max} megválasztása esetén merül fel. (A sebességprofil tervezésekor szükséges a hover végállapot, de ez módosítható igény szerint.) A pozíció profil tervezésekor az algoritmus bemeneteként kapott pontokat egy ε toleranciával vesszük figyelembe, és itt csak a megengedhető munkatér jelenti a korlátozást.

[2] Mivel a beavatkozó jelre vonatkozó korlátokat tudjuk előre megválasztani / specifikálni, ill ez az, ami időben nem változik. Később át lehetne térni a fizikailag kiadott beavatkozó jelek paraméterekre, ami pl. a rotort is figyelembe veszi, esetleg az orientáció változására előírni korlátozást az elérhető forgatónyomatékok ismeretében.

4.2. Áttekintés

Fontos leszögezni, hogy jelenleg a trajektória tervezése a mozgás előtt megtörténik, majd végrehajtódik.[3] Ezért pl. a gyorsulás adatokból csak a névleges értékek szükségesek a kvadkopter számára. Zavarásmentes esetben a névleges trajektórián végighalad a kvadkopter. A valós rendszeren elég a pozíciókat és sebességeket visszacsatolni az MPC szabályozáshoz.



4.1. ábra. A trajektóriatervező modul felépítése

A trajektóriatervező algoritmus bemenő paraméterei azon pontok, amelyen a kvadkoptert szeretnénk, hogy áthaladjon[4] (ezeket a 4.1. ábrán x_i, y_i, z_i-vel jelöltem, ahol esetünkben $i = 1, 2, 3$; szerepel egy kezdeti, egy köztes és egy végállapot). Az pontokból és a teljes mozgás idejéből generált trajektóriát az egyes időpontokban kiértékelve megkapjuk a kvadkopter számára előírt referencia pozíciókat és sebességeket és gyorsulásokat (a 4.1. ábrán ezt időfüggvénnyel jelöltem: $t \in [0 \;\; T_{end}]$, ami a 4.2b. ábrán látható $\lambda(t)$ értelmezési tartománya).

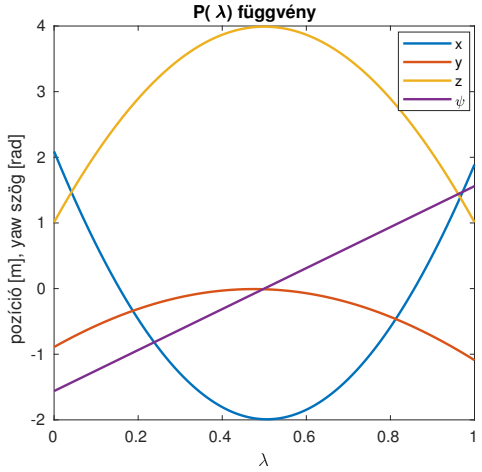
Az eljutáshoz szükséges időt iteratívan minimalizáljuk[5]. Minden iteráció során T_{end} idő ismeretében elkészül a sebességprofil, melynek a végpontokban a harmadik deriváltig zérusnak kell lennie. Ezzel biztosítjuk azt, hogy az orientáció referenciajelek simák legyenek. A pozíció trajektóriát úgy írjuk fel, hogy minden előírt pontot érintsenek[6].

[3] A végrehajtás közben már nem finomítjuk.

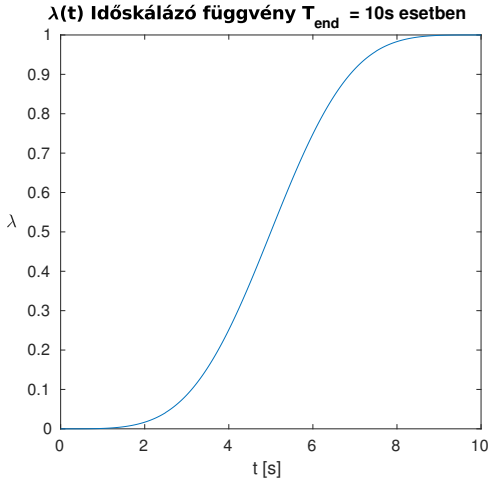
[4] Egyelőre a pontokban orientáció megadására nincsen lehetőség. A trajektória elején és végén hover állapot van, ha a köztes pontokhoz orientációt rendelünk, ahhoz nemlineáris `fmincon` constraintet kellene használnunk (az ehhez tartozó mátrixot `Ceq(X)` jelöli az `fmincon` súgójában). Továbbá egyelőre egy köztes pontban vett pozíciót tudunk előírni, ami a 4.2a. ábrán a λ= 0.5-nél látható.

[5] (A T_{end} értékét gamma algoritmussal csökkentjük egy kiinduló értékről: egy adott értékhez kiszámítjuk a névleges orientációkat a trajektória mentén, és ezt tartjuk egy előírt érték alatt.)

[6] Nagy kérdés még a szegmensek száma. Minimális hossz kritérium esetén a szegmensek számának növelésével a pályát egyre egyenesebb elemekből állítja össze a trajektóriatervező algortmus. Az ilyen *sarkosabb* pályához viszont nagyobb beavatkozó jelek fognak tartozni. Egy szegmens hozzáadása egy extra szabadsági fok az optimalizációban. [14] beszél a szegmensekről, érdemes lehet megnézni

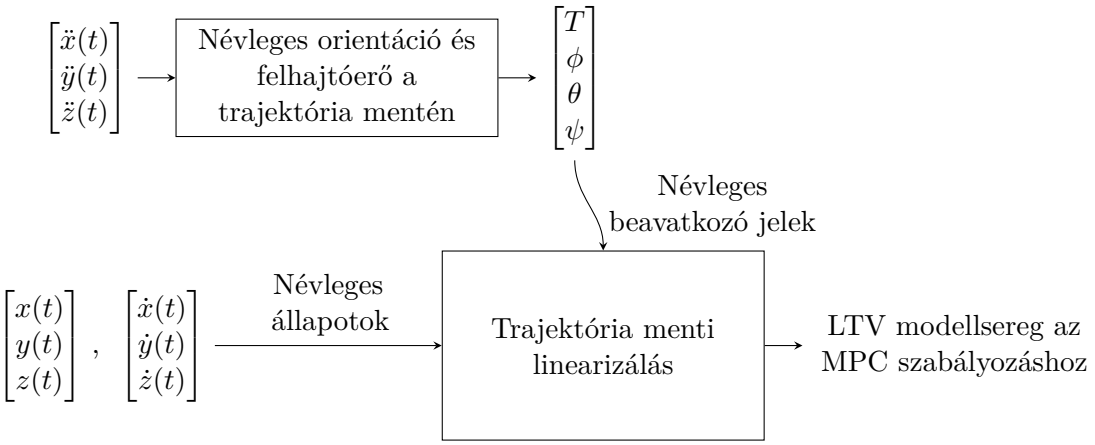


(a) A kvadkopter számára előírt útvonal, még hozzárendelt időbényegek és sebességek nélkül: a P(λ) függvény



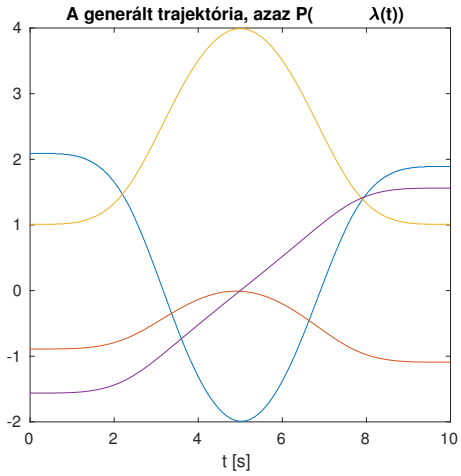
(b) A $\lambda(t)$ időskálázó függvény időpontokat rendel az útvonal pontjaihoz.

4.2. ábra. A trajektória az időskálázás behelyettesítésével kapható: P(λ(t)) alakban áll elő, így már csak az időtől függ (ld. 4.4a. ábra)

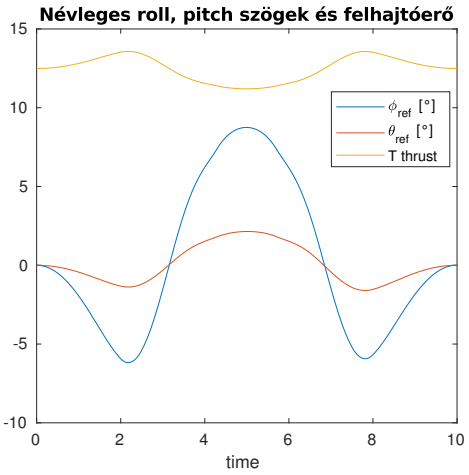


4.3. ábra. A trajektória felhasználása az irányításhoz

A 4.3. ábra mutatja azt, hogy a 4.1. ábrán feltüntetett trajektória menti gyorsulásokból hogyan kaphatók meg a névleges beavatkozó jelek (felhajtóerő (thrust, vagy lift), ϕ roll és θ pitch szög: számítása a 2.12. egyenlet szerint). A kvadkopter hibadinamikáját pedig a dinamikai modell névleges állapotainak és bemeneteinek eltéréseiből kapjuk, Jacobi linearizálással.



(a) P(λ(t)) trajektória, amiről minden időfüggvény leolvasható



(b) Névleges beavatkozó jelek

4.4. ábra. Trajektória (a referencia pozíciók időbeli alakulása) és az általa meghatározott névleges beavatkozó jelek

4.3. Specifikáció felállítása

A B-Spline trajektóriatervező modul fejlesztése előtt irodalomkutatást végeztem, és sorra vettem a korábbi trajektóriatervező algoritmus hiányosságait. A B-Spline módszereket számos területen használják, [14] és [25] tartalmaz egy igen részletes elméleti összefoglalót. A kvadkopterekre történő trajektóriatervezési módszerek alapjait [11] és [6] fogalja össze.

Az eddigi tapasztalatok alapján egy jobban konfigurálható, feladathoz szabható trajektóriatervezési módszert szerettem volna implementálni, először Simulinkben. Mindenképpen dinamikusan generált trajektóriát szerettem volna használni, hogy az előírt útvonalat a kvadkopter a számára előírt korlátozások betartásával automatikusan hajtsa végre: így az eljutási időt is úgy határozza meg, hogy a képességeit maximálisan kihasználja.

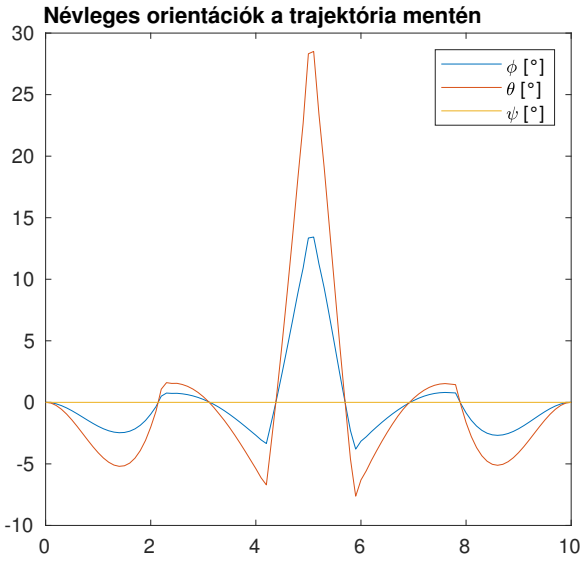
A régi trajektóriatervező hibái

A régi trajektóriához tartozó sebességprofil konstans volt. Ha a megadott trajektória pl. éles kanyarokat tartalmazott, az orientációnak nagyon gyorsan kellett megváltoznia (ld. a ld. 2.12. egyenlet szerint számolt névleges beavatkozó jeleket a 4.5. ábrán), és ezt a belső szabályozási kör nem tudta követni.

Ha törés van a trajektóriában, akkor módosítani kell a sebességprofilt, és le kell lassítani. A korábbi programkódban fixen szereplő mozgásprofil helyett egy dinamikusan változtatható függvény kell, ami a hirtelen ugrásokat kisimítja, és az orientáció referencia jelben a nagy ugrásokat kiszűri.

A cél az volt, hogy a megadott útvonal alakjától függetlenül képesek legyünk olyan trajektóriát illeszteni, ami betartja a kvadkopternek előírt korlátozásokat. Az eddig alkalmazott `piecewise polynomial`[7] formáról a B-Spline alakra tértem át [6] alapján, és a cikkben található gondolatmenetet követtem.

[7] A MATLAB-ban a két spline reprezetáció közti különbségeket a `https://www.mathworks.com/help/curvefit/types-of-splines-ppform-and-b-form.html` foglalja össze.



4.5. ábra. Névleges orientáció a trajektória mentén a régi algoritmussal, ami a Curve Fitting Toolbox harmadfokú spline illesztő függvényeivel dolgozott.

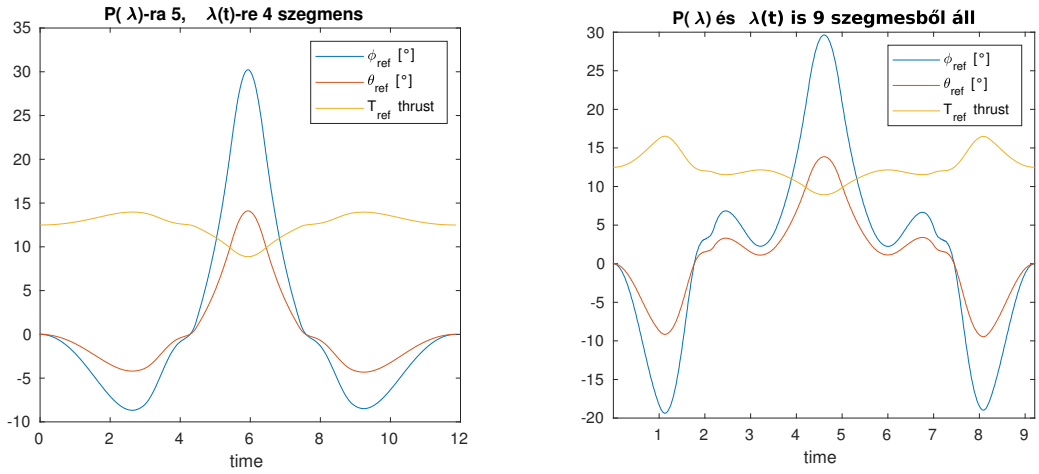
4.4. A B-Spline trajektóriatervező algoritmus

A következőkben bemutatom az eddig felvázolt igényekhez illeszkedő trajektóriatervező alrendszert, és értékelem a működését. Néhány javasat is szerepel majd a fejlesztési lehetőségekkel, hiányosságokkal kapcsolatban.

4.4.1. Eddigi eredmények

A B-Spline-ok miatt a gyorsulás második deriváltjai is folytonosak. A mozgások így simák, a trajektória köztes pontjaiban sincsen ugrás a referencia orientációkban (ez az ötödfokú, azaz quintic spline-nak köszönhető). A korlátozásokra dinamikusan optimalizált trajektóriát lehet generálni. Egyelőre egy optimalizálási keretrendszert sikerült kialakítani, és szükség esetén a kritérimfüggvények és a korlátozások módosíthatók. A valóban időoptimális trajektóriához ezeket pontosítani kell még. Jelenleg az optimalizálás célfüggvényei minimum hosszra vannak megadva.

A 4.5 ábrát a 4.6. és 4.7. ábrával összehasonlítva látható, hogy a B-Spline módszer sokkal jobban konfigurálható, a trajektóriát úgy futja be az időben, hogy a szükséges beavatkozó jelek egyenletesebben oszlanak el. Továbbá 30° maximális megengedett $\phi(t), \theta(t)$ kitérések mellett 10s helyett kevesebb, mint 7.5s alatt képes a kezdőpontból a végpontba eljutni, félúton az előírt pont érintése mellett. Az eredmény eléréséhez manuálisan állítottam be a B-Spline függvények szegmenseinek számát. A $\mathrm{P}(\lambda)$ 3 szegmensből áll, a $\lambda(t)$ 9 szegmensből. A jövőben az optimalizálási célfüggvényeket úgy kell majd módosítani, hogy ilyen manuális beállításra ne legyen szükség.



(a) (b)

4.6. ábra. Többféle szegmens választás hatása a névleges beavatkozó jelekre. Minden esetben a 4.5. ábrán láthatóhoz képest simább jeleket kaptunk. A szegmensek számának növelése extra szabadsági fokot ad az algoritmusnak [27].

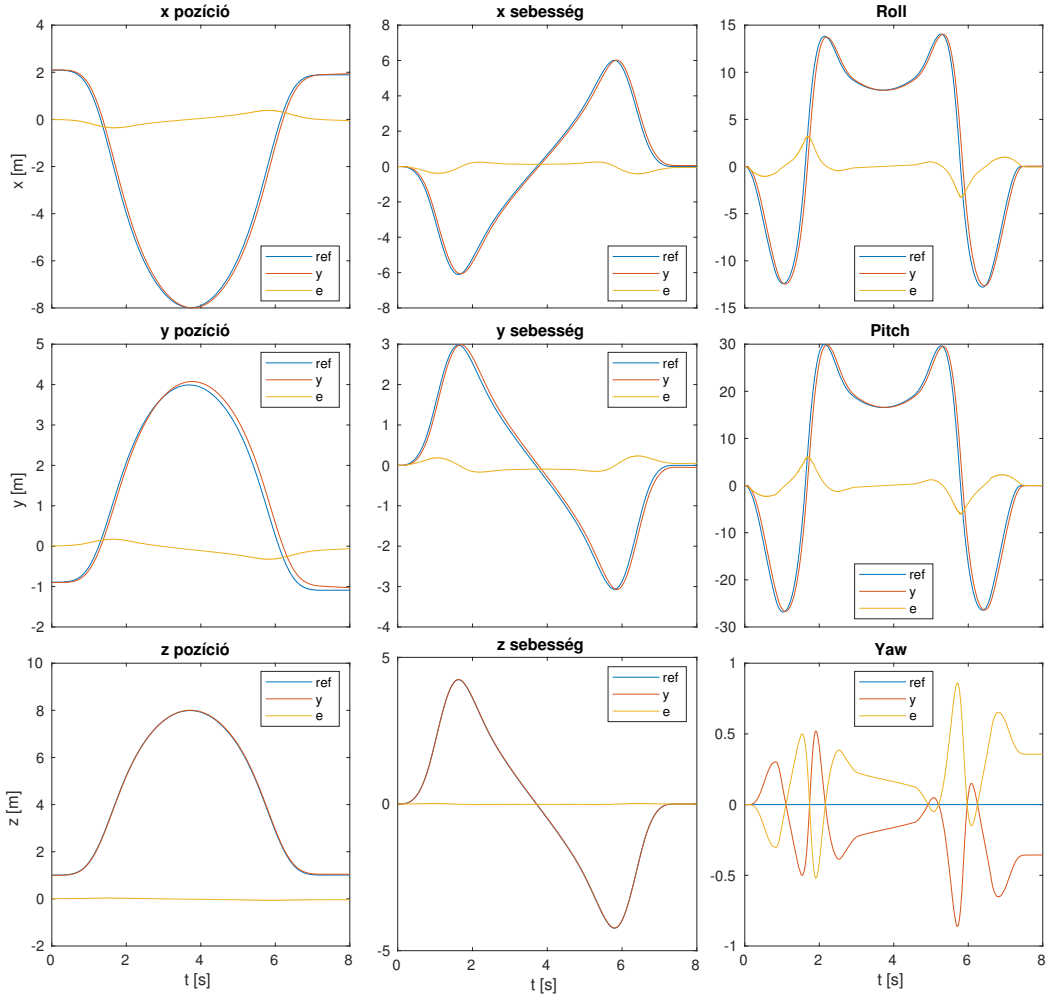
4.4.2. A B-Spline trajektóriatervező működése

Az optimalizálási változók P(λ) és λ(t) esetében is a spline control pontjai. A control pontok számára a spline fokszáma is hatással van. Ahány control pontunk van, annyi bázisfüggvényt fogunk generálni, így annál több komponens lineáris kombinációja alkotja a B-Spline-t (a bázisfüggvények, és a belőlük generált spline látható a 4.8 . ábrán). Továbbá a végpontokban a spline érétkét minél magasabb rendben szeretnénk meghatározni, annál nagyobb multiplicitással kell, hogy a végpontok szerepeljenek[8].

A szegmensek száma a control pontok számánál öttel kisebb, ezért egy egyszerű, egy szegmenses B-Spline hat control pontból áll, azaz hat bázisfüggvénye van. A bázisfüggvények generálásához knotok szükségesek, ezekből a control pontokhoz képest öttel több kell. Az elkészült spline-nak 11 knot-ja lesz: 5-5 a végpontokhoz tartozik, és egy lesz az intervallum közepén. A hat bázisfüggvényhez tartozó hat control pont ekkor 0 és 1 között egyenletesen elosztva helyezkedik el.

Több szegmensből álló trajektóriának tehát több szabad paramétere van, amit ki lehet használni az optimalizáció során. Bouktir cikke [6] említi, hogy a legrövidebb időt eredményező trajektória nem mindig a legrövidebb úton vezet. Ha a P(λ) és a λ(t) spline-ok optimalizálását összekötjük, és nem fix P(λ) értékekhez keresünk optimális sebességprofilt, akkor egy még általánosabb probléma megoldását kapnánk.

[8]Látható, hogy a 4.8b. ábra szerinti piros színű spline control pontjai mind különböznek. Ha az első két control pont nulla, akkor a kezdőpontban a derivált is nulla, ahogy az a kék spline-nál látható. Magasabb rendű deriváltakhoz, ill. köztes pontokban értelmezett constraintekhez növelni kell a control pontok számát.



4.7. ábra. Referenciakövetés open-loop irányítással (csak a névleges beavatkozó jelekkel), zajmentes esetben. A maximális távolság a referencia trajektóriától fél méter.

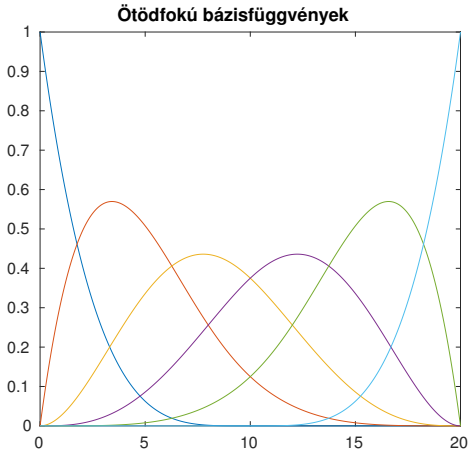
4.4.3. Implementáció

A bázisfüggvények generálása a Cox De Boor algoritmussal lehetséges a t_i knotokból. Ez egy iteratív eljárás, melynek utolsó lépése látható a 4.8a. ábrán, ahol $x \in [0 \quad 20]$ eset látható, a knotok pedig $\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 10 & 20 & 20 & 20 & 20 & 20 \end{bmatrix}$ értékűek.

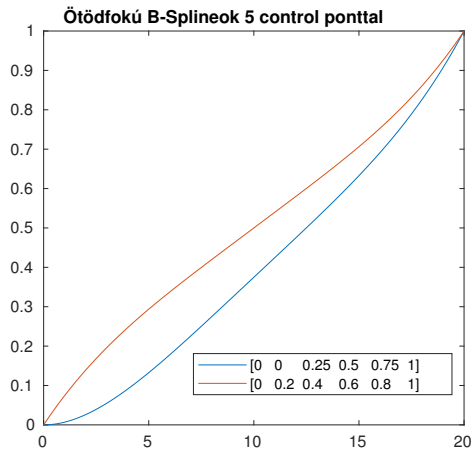
A bázisfüggvények generálása a következőképpen történik:

$$B_{i,0}(x) := \begin{cases} 1 & \text{ha} \quad t_i \leq x < t_{i+1} \\ 0 & \text{egyébként} \end{cases}$$

$$B_{i,k}(x) := \frac{x - t_i}{t_{i+k} - t_i} B_{i,k-1}(x) + \frac{t_{i+k+1} - x}{t_{i+k+1} - t_{i+1}} B_{i+1,k-1}(x).$$



(a) Bázisfüggvények ötödfokú spline-hoz: az optimalizálás tárgya, hogy a 6 bázisfüggvényt mekkora súllyal vesszük figyelembe, azaz 6 db control pont határozza meg a spline alakját.



(b) Control pontok értékei: ezek súlyozzák a bázisfüggvényeket. A piros B-Spline-ra csak a végpontbeli érték korlátozást írtuk elő, a kékre a nullában vett deriváltat is kikötöttük.

4.8. ábra. Egyszerű ötödfokú B-Spline konstrukciója egy dimenzióban

Ebből az n-edfokú B-Spline felírható:

$$S_{n,\mathbf{t}}(x) = \sum_i \alpha_i B_{i,n}(x). \tag{4.1}$$

Több dimenzióban, P(λ) esetében minden egyes csatornára külön írjuk fel a bázisfüggvényeket, és ezeket összefűzzük.

A bázisfüggvények annak megfelelően alakultak, ahogy a MATLAB is generálja őket, ám a saját implementációval a folyamatot mélyebben is meg tudtam érteni, és a további feldolgozásra könnyebben testre tudom szabni azt. (Az ábrán `spmak([0 0 0 0 0 0.5 1 1 1 1 1],cf)` függvény eredménye látható, `cf` pedig olyan hat elemű control pont vektor, melyben csak egy elemet állítotam 1-esnek, ez ad egy bázist a hatból.)

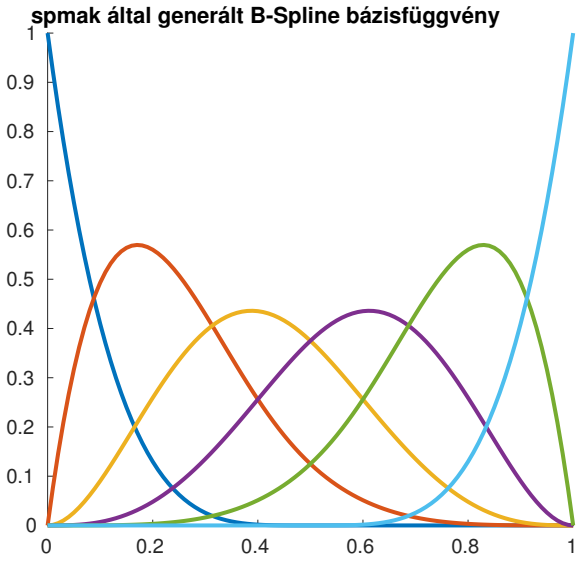
Trajektóriák egymásba ágyazása

A trajektóriatervező algoritmus a P(λ) és λ(t) függvényekhez külön bázisfüggvényeket generál, és külön-külön optimalizációval kapja meg a control pontjainak értékét.

Az idővel paraméterezett referenciajel előállításához a P(λ) függvényeket nem ekvidisztánsan értékeljük, ki, hanem azokban a 0 és 1 közé eső pontokban, amit a λ(t) függvény értéke határoz meg. Ezzel kapjuk meg a trajektória P(λ(t)) alakját.

Diszkretizálás, trajektória menti deriváltak és gyorsulások

Ezek az összefüggések numerkusan lettek megvalósítva, a t valamilyen finomságú felosztásával generáltam a bázisfüggvényeket. A korábbi trajektróiatervező algoritmus a spline-t annak analitikus formájában szolgáltatta, így az `fnval`, `fnplt`, `fnder` függvényekkel lehetett csak kezelni őket, és a mintavételezésük problémás volt.



4.9. ábra. Az algoritmus ellenőrzése a MATLAB `spmak`-kal kapott eredményhez képest

A megvalósított algoritmussal a spline-okat numerikusan megadjuk minden mintavételi időponthoz. Viszont ez jár egy hátránnyal, ugyanis a 4.4a. ábrán látható trajektória deriváltjai igen kicsik a végpontokban. Az implementációban a Matlab `diff` függvényét a trajektória menti gyorsulásokra nagyon pontatlan eredményeket kaptam.

A numerikus pontatlanságok orvosolásához a 4.1. egyenletet használtam ki. A diszkretizálás miatt minden bázisfüggvényhez figyelembe kell venni a felosztás finomságát, hogy a deriváltak numerikus értéke megfelelő legyen.

Mivel a B-Spline a bázisfüggvények control pontokkal súlyozott lineáris kombinációja, ezért a deriváltakat úgy is előállíthatjuk, hogy a bázisfüggvényeket deriváljuk, majd a control pont értékekkel súlyozzuk ezeket. Így sokkal pontosabb deriváltakat kapunk, amik felhasználhatók a szabályozáshoz.

A trajektória felírása mátrixos alakban a c_P control pontokkal[9] a P-hez és c_λ control pontokkal a sebességprofilhoz; illetve a hozzájuk tartozó ötödfokú bázisfüggvényekkel:

$$\mathrm{P}(\lambda(t)) = c_P^T \cdot B_P(c_\lambda^T \cdot B_\lambda(t)) \tag{4.2}$$

A derivált láncszabály alkalmazásával:

$$P(\lambda(t))' = P'(\lambda(t)) \ \cdot \ \lambda'(t)$$

A második derivált:

$$P(\lambda(t))'' = \Big(P'(\lambda(t))\Big)' \cdot \lambda'(t) + P'(\lambda(t)) \cdot \lambda''(t)$$

Amit tovább rendezve:

[9]Mivel $\mathrm{P}(\lambda)$ 4 dimenziós, ezért minden csatornának külön vannak control pontjai.

$$P(\lambda(t))'' = P''(\lambda(t)) \cdot (\lambda'(t))^2 + P'(\lambda(t)) \cdot \lambda''(t)$$

Az itt szereplő deriváltak felírásához az eredeti control pontokat használom fel, és a bázisfüggvényeket deriválom numerikusan. Így a deriváltakat sokkal pontosabban kapom meg. Ez a 4.1. ábra kimeneteinek felel meg:

$$P(\lambda(t))' = \begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{z}(t) \end{bmatrix} \tag{4.3}$$

$$P(\lambda(t))'' = \begin{bmatrix} \ddot{x}(t) \\ \ddot{y}(t) \\ \ddot{z}(t) \end{bmatrix} \tag{4.4}$$

Ezen értékek már felhasználhatók a névleges beavatkozó jelek számításához, illetve a kvadkopter modell trajektória menti linearizálásához.

Optimalizálási feladat

Az optimalizálási feladat változói a control pontok, amik a bázisfüggvények súlytényezői.

$$\mathrm{P}(\lambda) = c_p^T \cdot B_P(\lambda) \tag{4.5}$$

Azokat a control pontokat keressük, amellyel a kezdő- a köztes és a végpontokban az adott pozícióba kerülünk. Itt még nem vesszük figyelembe a trajektória időbeli lefolyását.

$$\begin{aligned} \min_c \quad & \|P(\lambda, c)\| \\ \text{úgy, hogy} \quad & P(0,c) = \begin{bmatrix} x_0 & y_0 & z_0 \end{bmatrix}^T \\ & P(\lambda = 0.5, c) = \begin{bmatrix} x_1 & y_1 & z_1 \end{bmatrix}^T \\ & P(1,c) = \begin{bmatrix} x_2 & y_2 & z_2 \end{bmatrix}^T \end{aligned} \tag{4.6}$$

Ahol λ=0 a kezdőpontra, λ=0.5 a köztes pontra, λ=1 a végpontra vonatkozó constraint. Az optimalizálást MATLAB-ban az `fmincon` végzi, a kiértékelt bázisfüggvények kerültek az `Aeq` mátrixba soronként, az előírt értékek pedig a `Beq` mátrixba szintén soronként.

A sebességprofil felépítése a következő:

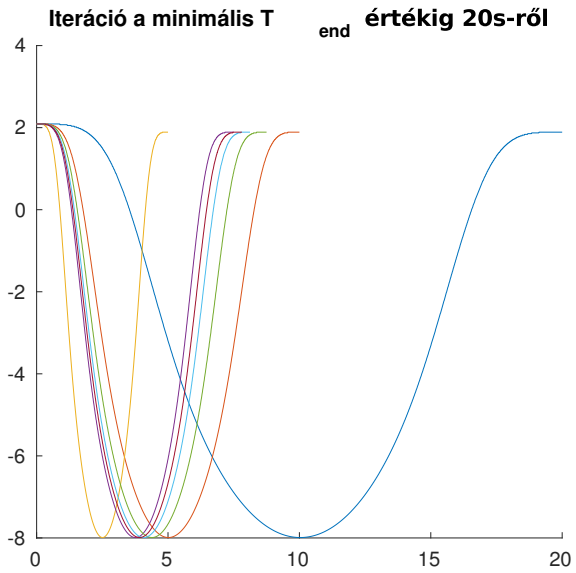
$$\begin{aligned} \min_c \quad & \|\lambda(t, c)\| \\ \text{úgy, hogy} \quad & \lambda(0,c) = 0 \quad \lambda(1,c) = 0 \\ & \dot{\lambda}(0,c) = 0 \quad \dot{\lambda}(1,c) = 0 \\ & \ddot{\lambda}(0,c) = 0 \quad \ddot{\lambda}(1,c) = 0 \\ & \dddot{\lambda}(0,c) = 0 \quad \dddot{\lambda}(1,c) = 0 \end{aligned} \tag{4.7}$$

A végpontokban a harmadik deriváltaknak folytonosnak kell lennie, illetve a control pontok szekvenciájának monoton növőnek kell lennie. A fenti választással

A végrehajtási idő optimalizálása az alábiak szerint történik: először beállítunk egy maximális végrehajtási időt (ez a 4.10. ábrán 20 másodperc. Iteratívan csökkentjük ezt az időt, minden lépésben ellenőrizve, hogy az kielégíti-e a maximum kitérés kritériumot (ehhez kiértékeljük a végleges irányítást a trajektória mentén), ha igen, akkor csökkentjük a végrehajtáshoz szükséges időt.

A próbálkozásokat a gamma-iteráció algoritmusával végezzük, ezt az alábbi képen látható iteráció követésével magyarázom el: először a [10 20] intervallumon vizsgálódunk. T_{end}=10-re a korlátozások teljesülnek. Ezért az új intervallumunk a [0 10], 10-nél szintén teljesül a korlátozás, [0 5]-öt vizsgáljuk. Ez már nem teljesül, ezért már 5-nél nem csökkentünk tovább, az [5 7.5] intervallumot vizsgáljuk, és így tovább.

Minden lépésben az intervallumunk hosszát a felére csökkentjük, így biztosítva a gyors konvergenciát.



4.10. ábra. Iteráció a végrehajtás idejének meghatározásához. A leállási kritérium az előre megadott maximális θ_{ref}, ϕ_{ref} 1°-os környezete

Yaw szög tervezése

A trajektóriát [17] szerint a pontok, és a hozzájuk rendelt yaw szögek alkotják, ezek együtt az ott használt nevezéktan szerinti ún. keyframe-ek, vagy waypointok, amiket úgy lehet elképzelni, mint egy lyukas képkeret, amelyen csak egy bizonyos irányból lehet áthaladni.

Gyakran használnak a szabályozók validálásához kör, spirál, vagy nyolcas alakú trajektóriát. Ilyenkor pedig jellemzően azt várjuk el a kvadkoptertől, hogy ezeket az alakzatokat úgy járja be, hogy mindig a pályára érintőirányba nézzen, ne konstans északnak. Az előírt yaw szögnek ebben az esetben ugrása lehet, ha azt $-180°$ és $+180°$ között mérjük (ez főként implementáció során érdekes).

Mint már láthattuk, a spline alapú trajektóriatervezésnél az illesztett görbék deriválhatóak, így ha a bemenő adatokban ugrás van, az az algoritmus helytelen működését okozza: hogy az ugrást kisimítsa, egy Gibbs-oszcillációhoz hasonló jelenség lép fel, és nem valósul meg az előírt ψ lekövetése.

Ahhoz, hogy a trajektóriatervezés és végrehajtás zökkenőmentes legyen, a yaw szöget az ugrásmentes, így akár a $-180°$ és $+180°$-os tartományon kívüli esetre kell megtervezni (a MATLAB-ban az `unwrap` függvény használatával), a referenciajelet pedig az implementációs környezetnek megfelelően kell kiadni, ami esetünkben $-180°$ és $+180°$ közti értéktartományban (a MATLAB-ban a `wrap` függvény használatával).

5. fejezet

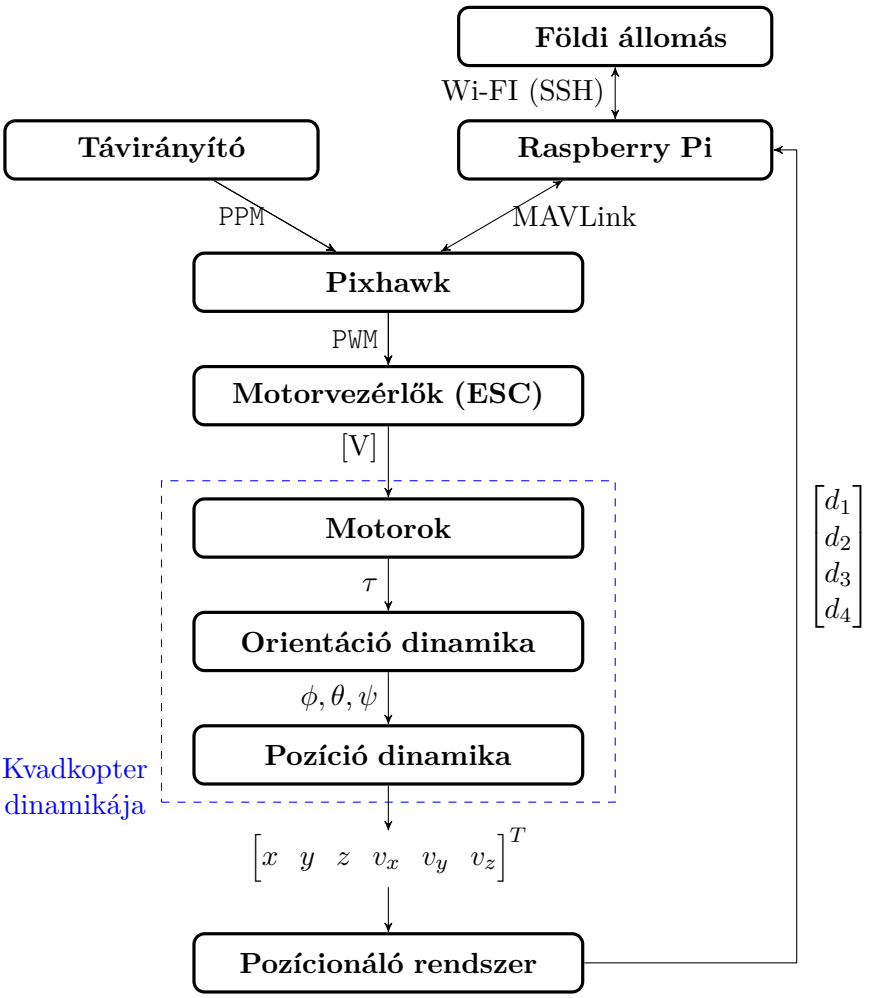
Implementáció

Az implementáció célja az volt, hogy az eddig bemutatott algoritmusok működését a drónaréna környezetben is ki tudjuk próbálni. Ezzel sokkal jobban körüljártuk a bemutatott módszereket, megtapasztalhattuk, hogy azok mennyire könnyen (vagy mennyire nehezen) ültethetők át a gyakorlatba, és milyen eredményekkel kecsegtetnek.

5.1. ábra. A repülési tesztek során használt kvadkopter, a 2.2. ábra szerinti elrendezésű, Hobbyking S500-as típusú vázzal

A munkánk során számos esetben megtapasztaltuk, hogy a szimulációban működő funkciókat mennyi további felügyeleti réteggel kell kiegészíteni, hogy azok biztonságosan használhatóak legyenek.

A bemutatott szabályozókat mind implementálni tudtuk, ám a korábban ismertetett formában azok nem voltak alkalmazhatók. A fejezetben bemutatjuk, milyen lépésekre volt szükség ahhoz, hogy ezek alkalmazhatóak legyenek és jó eredményeket adjanak.



5.2. ábra. A kvadkopter tesztkörnyezet felépítése

5.1. MATLAB

A cél az, hogy a drónt élesítve, és automata üzemmódba kapcsolva adott feladatokat el tudjunk végezni, úgy mint a fel- és leszállás, a drón egy helyben tartása, trajektóriakövetés, identifikációs célú gerjesztések kiadása.

A Simulinkben implementált algoritmusokat kód generálással tettük beágyazott környezetben is futtathatóvá. A Raspberry Pi 4 magos processzorából egyet a pozíciószabályozó és állapotbecslő kap (utóbbi szintén Simulinkben lett megvalósítva), egyet a Pixhawk és a Raspberry Pi közti kommunikációt biztosító MAVLink kommunikáció, egyet pedig a mérések memóriába mentését biztosító log-olási funkció kapott.

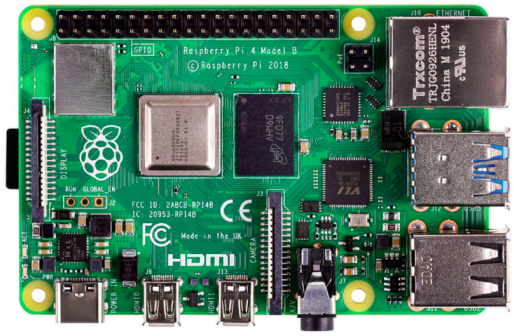
5.1.1. Autopilóta feladatok - *a szabályozó környezete*

- Bekapcsoláskor az integrátorok, belső állapotok nullázása
- Előre beállított trim értékek, az offsetek kompenzálására
- Hover célpozíció beállítása
- Névleges thrust jel kiadása

- Trajektória bejárásához a névleges állapotok (ebből lesznek a referenciajelek a hibadinamika szabályozónak) – 6.1. ábra és névleges beavatkozó jelek kiadása
- Doublet kiadása a roll / pitch csatornákra, ezalatt a thrust szabályozás bekapcsolva tartása, a másik kettő kiiktatása
- Hover idejének, leszállás időpontjának paraméterezhetősége

Ezen funkciók nagy része a 6.1. ábrán látható módon került megvalósításra. A paraméterek számára létrehozott Matlab Function blokkokra azért van szükség, hogy a kifordított kódba ezek ún. tunable parameter-ként kerüljenek bele, azaz értékük a generált C kódból változtatható legyen[1].

(a) Pixhawk fedélzeti kontroller



(b) Raspberry Pi, ami a Simulinkből fordított kód futtatását végzi

5.3. ábra. A fedélzeti elektronika főbb komponensei

5.2. Hardveres környezet

A Pixhawk fedélzeti kontroller felelős a kvadkopteren az orientációszabályozásért, illetve a távirányító is ehhez csatlakozik.

A Pixhawk rendszeren[2] (5.3a. ábra) futó Ardupilot szoftvernek számos repülési módja van, amelyek a különböző jellemző felhasználási módokhoz igazodnak (összesen 23 ilyen mód van, melyből 10 mód használata a legjellemzőbb[3]). Ezek között a távirányító segítségével, vagy standard protokollt használó üzenetekkel lehet váltani.

[1] Kiemelten fontos szempont a paraméterezhetőség és konfigurálhatóság, hogy ne kelljen a teljes toolchainen keresztül Matlab Simulinkből fordítani az egész kódbázist, majd a teljes programot a Raspberry Pi-n újra fordítani, hanem lokálisan a Raspberry Pi-n is módosíthatók legyenek ezek az értékek, ami után egy gyors build-eléssel futtatható a szoftver.

[2] A Pixhawk kontrollerhez csatlakoztatható perifériák listája a `https://ardupilot.org/copter/docs/advanced-pixhawk-quadcopter-wiring-chart.html` oldalon található (angolul).

[3] A repülési üzemmódokról bővebben (angolul): `ardupilot.org/copter/docs/flight-modes.html`

Table 1. Sampling frequency and response time for the different sublevels of a quadrotor.

System	Frequency and Response Time
Motor/ESC	1000Hz, 0.05s
Attitude	200Hz, 0.5s
Position	50Hz, $\geq$1s

5.4. ábra. A különböző alrendszerek dinamikája [3]

Gyakran használt üzemmód pl. a `Stabilize`, amely engedélyezi a pilótának a manuális repülést, de ha a pilóta elengedi a távirányítón a roll / pitch szögek megadására használt botokat, akkor a koptert stabilizálja hover állapotba. Ez a funkció kiváltja az ún trim-elés műveletét, ami arra szolgál, hogy megtaláljuk azokat az offset értékeket, amiket kiadva a kopter ténylegesen hover állapotban marad. Ekkor, ha nem hatnak rá jelentős zavarások (pl. nyílt térben, szélcsendes időben), akkor nem kúszik el jelentősen a pozíciójából sem. Ebben az üzemmódban tehát az történik, hogy ha a kopter nem kap orientáció referenciajelet, akkor saját magának találja ki azt annak érdekében, hogy hover-ben maradjon.

Az `Acro` üzemmód ennél direktebb irányítást tesz lehetővé: amennyiben egy orientáció referenciajelet kap (akár távirányítóról, akár üzenetcsomagban), akkor arra az értékre (setpointra) áll be, és azt tartja. Mivel a kvadkopter egy instabil rendszer, ezt az üzemmódot igen veszélyes szabályozás nélkül, manuális üzemmódban használni. Például bedöntés esetén szükség van a thrust növelésére, hogy a kopter ne veszítsen a magasságából: ezt a `Stabilize` mód automatikusan elvégzi, az `Acro` módban nincsen ilyen kompenzáció.

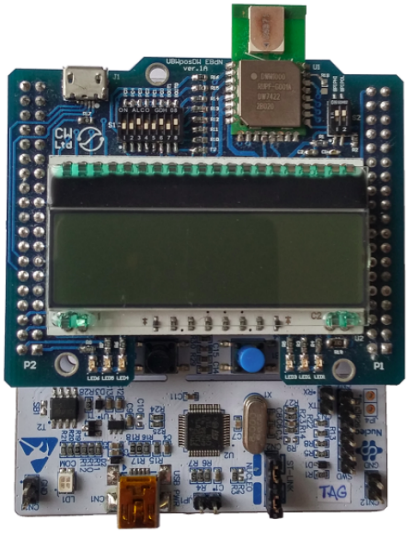
A többi repülési üzemmód között vannak olyanok, amelyek GPS-szel működnek, vagy pl. barométer segítségével magasságtartásra alkalmasak. Létezik `Autotune` mód is, ahol az orientációszabályozó kört adaptívan hangolja a szoftver. Ezek a módok többnyire kültérben használhatóak, beltérben nem használhatók ki a képességeik. Ezért pl. az orientációszabályozás finomhangolásához más módszereket kellett használnunk.

A repülési üzemmódokon felül hozzáférhető egy teljes paraméterlista, ahol a legkülönbözőbb konfigurációs beállításokat lehet elvégezni. Itt szükséges beállítani az Ardupilot számára, hogy milyen típusú vázat, hány motort, mekkora légcsavarokat használ az irányítani kívánt légijármű[4] (UAV). A repülési tesztek során az a célunk, hogy a kiadott orientáció referenciajeleket a kopter a lehető legjobban lekövesse, így például az orientáció referencia szűrő paramétereit (`ATC_INPUT_TC`) alacsonyabb időállandójúra kellett beállítani. Az orientációszabályozási körre a *belső loop* elnevezést fogom használni: ennek hangolása, a többi paraméterek beállítása a QGrondControl szoftverben történt (ld. 5.6. ábra).

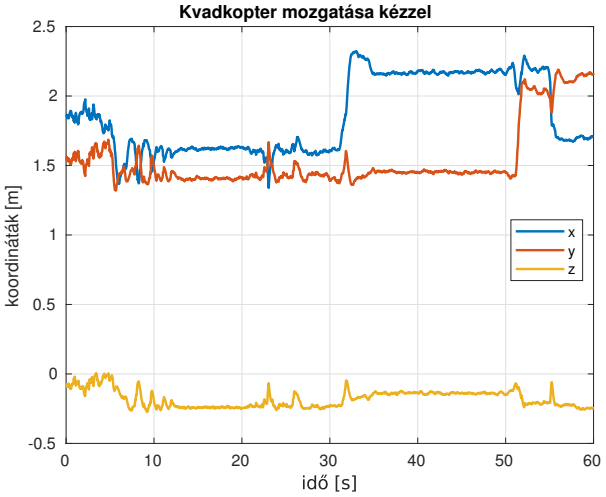
Pozícionáló rendszer pontossága

A beltéri pozícionáló rendszer pontosságának becsléséhez végeztünk néhány olyan tesztet, amikor a kvadkoptert a talajon mozgattuk egy-egy koordinátatengely mentén. A 5.5b. és 5.5c. ábrákon látható, hogy ha a koptert egy tengely mentén mozdítottuk el, az áthallást okozott a többi csatornák becsléseire. Megállapítható, hogy alacsony sebességek esetén a pozícionálás kellően pontos, a becsült értékek kb. 2-3cm-en belül vannak.

[4] A lehetséges konfigurációk listája és a légcsavarok forgásirányai megtalálhatók a `https://ardupilot.org/copter/docs/connect-escs-and-motors.html` oldalon (angolul), illetve a 2.2. ábrán.



(a) A pozícionáló rendszer áramköre [18]



(b) Pozíció becslés pontosságának vizsgálata

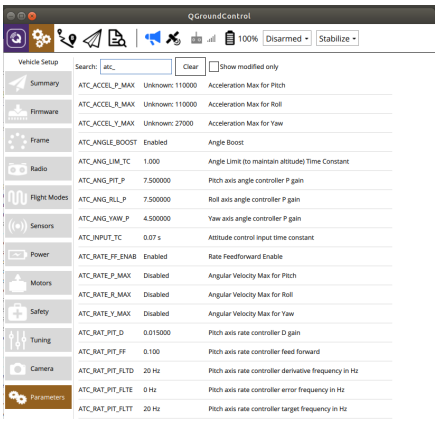
(c) Sebességbecslés pontosságának vizsgálata

5.5. ábra. Beltéri pozícionáló rendszer zajjal terhelt mérései

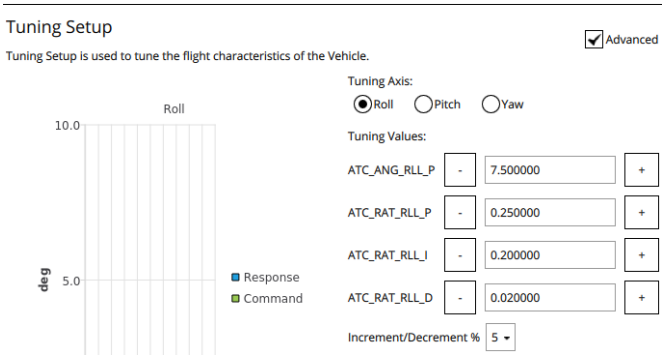
Orientációk lekövetése

A pozíciószabályozás tervezése során feltételeztünk, hogy az orientáció dinamika gyorsasága miatt annak átvitele egységnyinek vehető, a beállási ideje elhanyagolható. A Pixhawkon el kellett végezni néhány beállítást annak érdekében, hogy az orientációkövetés kellően gyors legyen, ez paraméterek[5] és szabályozók finomhangolását jelentette, illetve elengedhetetlen volt a gyorsulásmérő és magnetométer szenzorok kalibrációja is.

Alapbeállítás szerint a Pixhawk egy igencsak robusztus orientációkövető szabályozót futtat. Ez azért szükséges, hogy többféle tömegű illetve inerciájú kvadkopteren működjön, emiatt viszont a performanciája elmarad az általunk elvárttól. A szükséges beállításokat a QGroundcontrol szoftver használatával végeztük el.



(a) Pixhawk paraméterek az orientációszabályozáshoz



(b) PID paraméterek

5.6. ábra. Orientációkövetéshez tartozó paraméterek

Doubletek

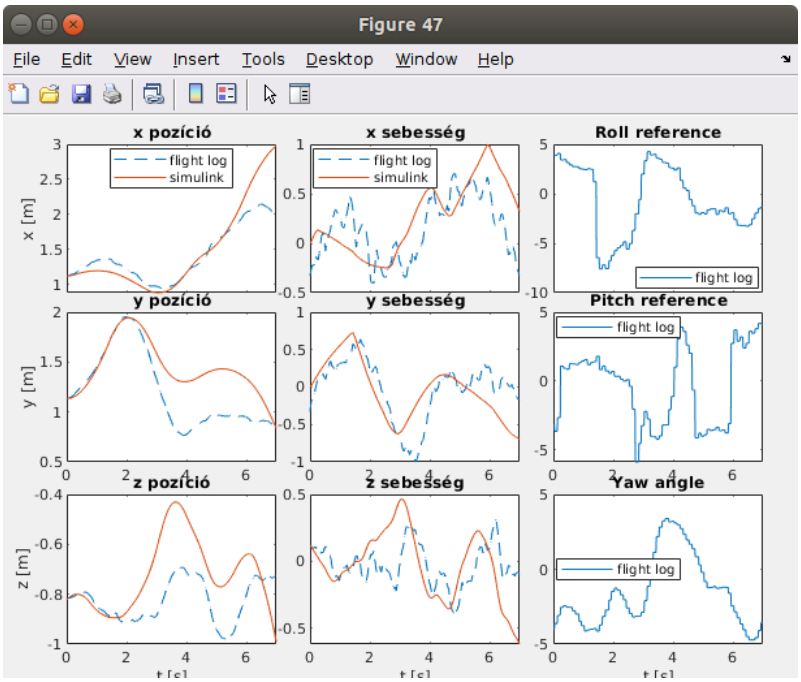
A finomhangolás validálása úgy történt, hogy autopilóta módban felszálltunk, bevittük egy kiindulási pontba a koptert, majd egy ún. doubletet adtunk rá.

A doublet időtartamára a roll/pitch szabályozókat kikapcsoltuk: az MPC-nél ez könnyen megvalósítható volt azzal, hogy a constraintjeit a roll/pitchre kinulláztuk, így a thrust loop aktív maradt, a kopter nem zuhant be.[6]

A doubletet külön a roll és külön a pitch csatornára adtuk, amíg a másikra nullát adtunk ki. Ám ekkor a szenzor rögzítése miatt a kopter teste (frame) rollban kb -1 fokban tért ki. Ezért ahhoz, hogy a roll-hoz tartozó y koordinátában ne mozduljon el (az x koordinátához tartozó) pitch doublet alatt, a roll-ra 1°-ot kellett adni.

[5]Mivel a Pixhawkon futó Ardupilot keretrendszer paraméterlistája igen terjedelmes (lásd 5.6a. ábra), a lehetséges beállítások feltréképezése, és a helyes beállítások megtalálása igen hosszadalmas lehet. Külön nehézséget okoz, hogy a paraméterek fel- és letöltése nem túl felhasználóbarát, könnyen előfordulhat, hogy egy-egy konfiguráció lementése elmarad.

[6]Ha a roll/pitch nem nulla, akkor a thrustnak egy korrekciót kell hozzáadni, hogy a magasságát tartani tudja.



5.7. ábra. A mérések során kiadott orientációk összevetése a Simulink modell pozíció dinamikájával

5.3. A nemlineáris modell validálása

A modellalapú szabályozótervezés során kíséretet tettünk arra, hogy a repülési tesztek során kiadott doubletek hatását reprodukáljuk a szimulációs környezetben is. Ehhez a Pixhawkon a megfelelő orientációbecslőt kellett beállítani [20] és annak a méréseit MAVLink üzenetekkel lekérdezni, hogy azok a Raspberry Pi számára hozzáférhetőek legyenek.

Így a repülések során logolt orientációkkal meg tudjuk gerjeszteni a nemlineáris modellt, és (a névleges beavatkozójeleket kivonva ezekből) a linearizált hibadinamikát, hogy ellenőrizzük, hogy a kopterünk pozíció dinamikája a modellnek megfelelően viselkedik-e.

Ehhez a logolt orientációkra és thrustra volt szükség. Viszont a pontos szimulációhoz a logolt értékek offset- és erősítéshibájának becslése is elengedhetetlen volt. Ideális esetben a hover-hez tartozó orientációk zérusok, az ehhez tartozó thrust pedig konstans kellene hogy legyen. Viszont az akkumulátor merülésével a kopternak egyre nagyobb thrust értéket kell kiadnia a hoverben tartáshoz, ezt pedig a névleges irányítás nem tudja kezelni. A szabályozónk ezért az idő elteltével egyre nagyobb és nagyobb thrust referenciát ad ki, ami a szimulációban ahhoz vezet, hogy a kopter egyre csak emelkedni fog.

A pontos szimulációhoz azt is figyelembe kell venni, hogy a yaw szöget a kopternek nem sikerül teljesen nullában tartania. Ezért a nemlineáris modellre adott roll és pitch szögeket *visszaforgatjuk* a yaw szögnek megfelelően. Szemléletesen erre azért van szükség, mert nem nulla yaw szög esetén egy roll doublet hatása x-ben is megfigyelhető. További hibát okozhatnak a rossz kezdeti sebesség értékek, amik néhány másodperc alatt a szimulált pozíció driftelését okozzák.

6. fejezet

Irányítás és repülési tesztek

6.1. Névleges irányítás blokkja

Tartalmazza felszálláshoz szükséges simított z koordináta referencia, a hover xy koordinátái (a trajektória kiindulási pontja), a leszállást megvalósító z referencia szűrését, és az ehhez szükséges paramétereket. Ezeket mind a C kódban lehet állítani anélkül, hogy Simulinkből ismételten kódot kellene generálni, gyorsítva így a tesztelést és a hibakeresést.

A trajektória az autopilótába kapcsolástól paraméterezhetően késleltethető, végrehajtásnak kezdetét a t_0 paraméter növekedése jelzi. A névleges irányítás a yaw szög függvényében transzformálva kerül kiadásra a 6.1. ábrán látható `nominal_control` blokkban megvalósított 2.11. egyenlet alapján.

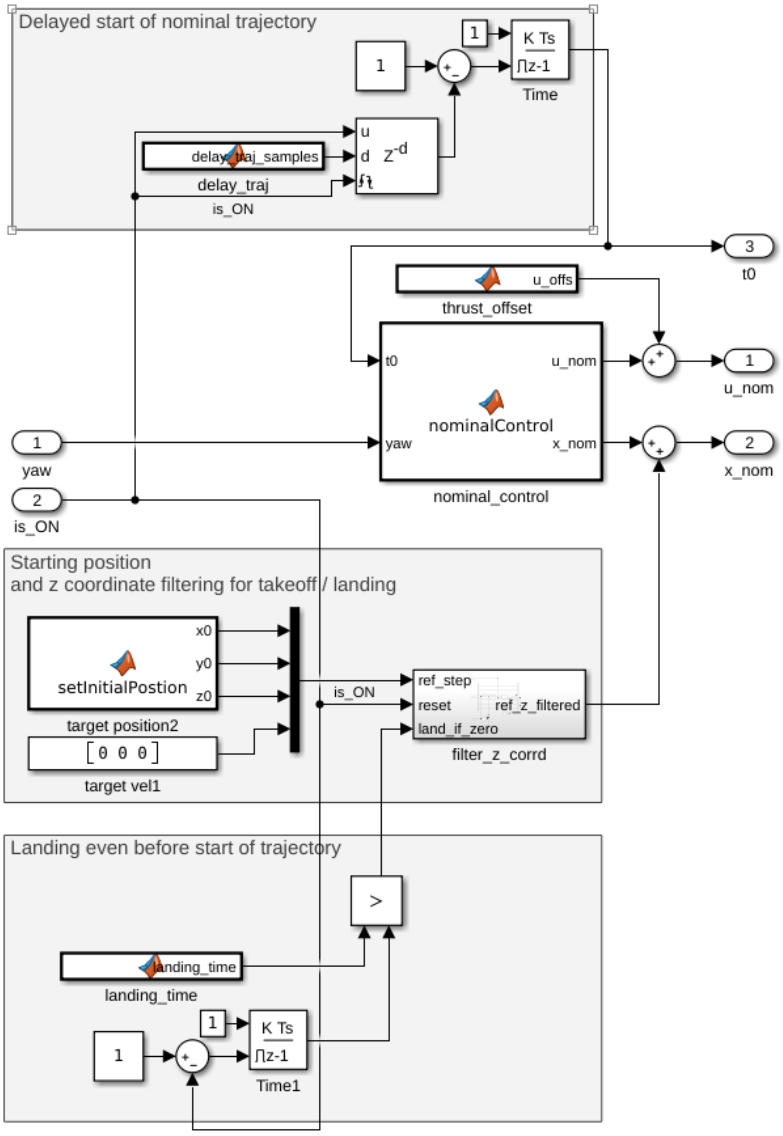
6.2. PID

A PID szabályozást a deriváló tag helyett a becsült sebesség visszacsatolásával valósítottuk meg. A behangolás után a következő paramétereket kaptuk:

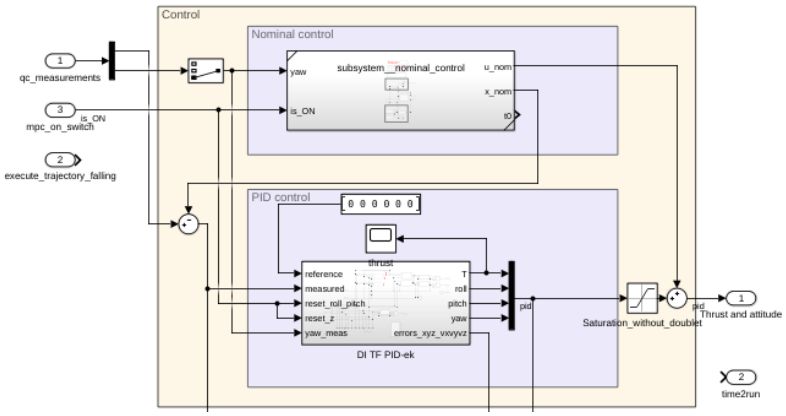
6.1. táblázat. PID szabályozó paraméterek

	P	I	D
thrust	-3	0.05	-3
roll	0.1	0.01	0.15
pitch	-0.1	-0.01	-0.15

A kvadkopter hover állapotban tartását a szabályozó a 6.4c. ábra szerint volt képes elvégezni. A szabályozó a másik két algoritmussal összehasonlítva igen jó eredményt ért el. A referencia pozíciótól vett abszolút eltérésben a középső volt, az LQI szabályozót megelőzve, de az MPC szabályozás performanciájához is közel került. Beavatkozó jelei viszont relatíve nagyok voltak, és elérték az 5°-os szaturációt.



6.1. ábra. Névleges irányítást megvalósító subsystem blokkvázlata

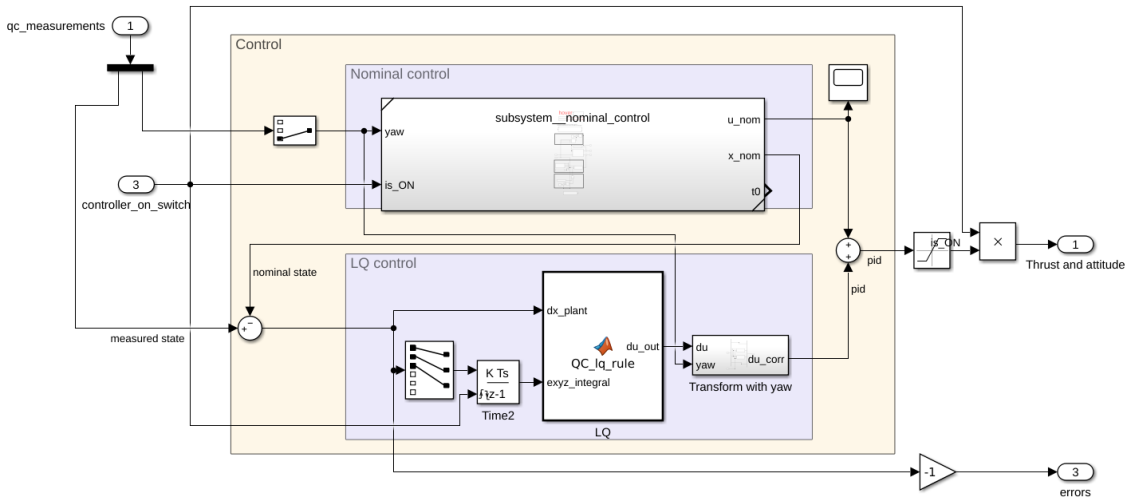


6.2. ábra

6.3. LQI

Az LQ szabályozás implementáció során megmutatkozó előnye, hogy mivel a beavatkozó jel egy K mátrixszal való szorzással áll elő, a számítási kapacitás igénye kicsi. Ha egy C kódban állítható paraméterben tudjuk tárolni a K mátrixot és a kimeneti szaturáció limitjét, akkor gyorsan tesztelhető és szükség esetén újrahangolható a szabályozó.

Ideális esetben a hoverhez tartozó roll/pitch orientációk zérus értékűek. Viszont ha az orientáció mérés offsettel terhelt, akkor a hover állapothoz nem nulla beavatkozó jelek szükségesek. Lineáris szabályozó nulla hibára nem képes nem nulla beavatkozójelet kiadni, az LQ szabályozó így bár megtartotta hover-ben a koptert (így sebessége és roll / pitch szöge nulla lett), de az előírt ponthoz képest maradó hibával. Ezért a szimulációban megtervezett LQ szabályozót módosítani kellett.



6.3. ábra. LQI szabályozás implementációja

- Ennek egyik oka, hogy a szenzort (az IMU a Pixhawk-on belül található) nem lehet pontosan vízszintesen rögzíteni. Ez azt eredményezi, hogy a hover állapothoz nem nulla beavatkozó jelek szükségesek.

- Tegyük fel, hogy a kvadkoptert hover állapotban tartjuk, pl. manuálisan. Átkapcsolva autopilótára, egy lineáris szabályozó nulla pozíció és nulla sebességhibához nulla értékű beavatkozó jelet adna ki. Viszont a fenti offset miatt ez azt eredményezné, hogy a kvadkopter el fog távolodni az előírt pontból, a hiba megnő.
- Annak, hogy az LQ szabályozás gyengébben működött, több oka is lehetett: például a zajos állapotbecslés miatt a beavatkozójel is zajosabb lett. Ennek egy lehetséges továbbfejlesztése a frekvenciafüggő Q és R mátrixok használata.
- A szimulációban a kimeneti offsethibák becslését egy Kalman-szűrő végezte. Ezt jól behangolni hosszadalmas folyamat lett volna, így inkább az alább megvalósított integráló szabályozást terveztem meg.

6.3.1. Tervezés

A maradó hiba megszüntetésére felvettük három új állapotnak a pozícióhibák integrálját. A szabályozó által használt kibővített szakasz a következő alakban írható:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \\ \dot{e}_x \\ \dot{e}_y \\ \dot{e}_z \end{bmatrix} = f(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} v_x \\ v_y \\ v_z \\ -(\sin\phi \cdot \sin\psi + \cos\phi \cdot \cos\psi \cdot \sin\theta) \cdot \frac{T}{m} \\ -(\cos\phi \cdot \sin\psi \cdot \sin\theta - \cos\psi \cdot \sin\phi) \cdot \frac{T}{m} \\ -\cos\phi \cdot \cos\theta \cdot \frac{T}{m} + g \\ x_{ref} - x \\ y_{ref} - y \\ z_{ref} - z \end{bmatrix} \tag{6.1}$$

Így az erre a szakaszra felírt hibadinamika tartalmazza a trajektóriakövetési hiba integrálját is, így ezt is bevehetjük az LQ-szabályozó költségfüggvényébe:

```
J = dx'*Q*dx + du'*R*du;
```

Az integrátor miatt a Q mátrix az alábbi alakban használatos:

$$Q = \begin{bmatrix} q_1 & & \\ & \ddots & \\ & & q_9 \end{bmatrix} \tag{6.2}$$

Ahol $q_1 = q_2 = \frac{1}{(0.4)^2}$, $q_3 = \frac{1}{(0.05)^2}$ érték a megengedhető pozíció hibaintegrálokra, $q_4 = q_5 = \frac{1}{(1.5)^2}$ a pozíciókra, a többi érték pedig (3.4) szerinti.

amely Q és R mátrixokra meghívhatjuk a diszkrét idejű LQ tervező függvényt:

```
[K,S,CLP]=dlqr(sys.A,sys.B,Q,R);
```

És így az LQI szabályozó a kiterjesztett szakaszra egy állapotvisszacsatolásként adódik. Ez az alak az implementációhoz azért előnyösebb, mert zajos pozíció- és sebességmérések nem *rontják el* a beavatkozójelet, a követési hiba kiintegrálása pedig egy zajmentesebb állapotot eredményez.

A szabályozó performanciája a 6.5. ábrán hover-ben, a 6.7 ábrán trajektória mentén látható, hasonló pontossággal.

6.4. MPC

A tervezés során felhasznált modell a kibővített szakasz [13] volt. Az állapot szót a hibadinamika 9 állapotára kell érteni, a bemenet szót pedig a szabályozó predikciós horizontján optimalizált beavatkozó jelre (amiben nincs benne a névleges trajektóriához tartozó névleges beavatkozó jel. A tervezéshez felhasználható volt az LQ hangolása során adódó súlyozás, mivel az MPC szabályozáshoz is a (6.1) egyenlet szerinti modellt használtuk fel.

$$\begin{array}{ll} \min\limits_{\mathbf{du}} & \mathbf{du}^T H \mathbf{du} + 2 f^T \mathbf{du} \\ \text{s.t.} & A_c \mathbf{du} \leq b_c \end{array} \tag{6.3}$$

A behangolás után a súlyozás a következőképp alakult:

$$\int e_{xy}^{max} = 0.4 \qquad \int e_z^{max} = 0.12 \tag{6.4a}$$

$$q_1 = q_2 = \frac{1}{(e_z^{max})^2} \qquad q_2 = \frac{1}{(e_z^{max})^2} \tag{6.4b}$$

$$e_{x,y}^{max} = 0.2m \qquad e_{v_x,v_y}^{max} = 0.12 \tag{6.4c}$$

$$q_4 = q_5 = \frac{1}{(e_x^{max})^2} \qquad q_6 = q_7 = \frac{1}{(e_{v_x,v_y}^{max})^2} \tag{6.4d}$$

$$e_z^{max} = 0.25 \qquad e_{v_z}^{max} = 0.4 \tag{6.4e}$$

$$q_6 = \frac{1}{(e_z^{max})^2} \qquad q_9 = \frac{1}{(e_z^{max})^2} \tag{6.4f}$$

$$Q = \begin{bmatrix} q_1 & & \\ & \ddots & \\ & & q_9 \end{bmatrix} \tag{6.4g}$$

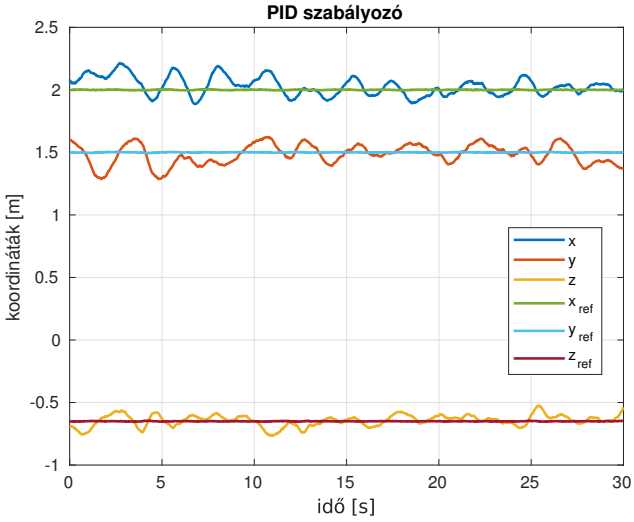
$$\delta T^{max} = 0.8 \qquad \delta\phi^{max} = \delta\theta^{max} = \delta\psi^{max} = 1.5^\circ \tag{6.4h}$$

$$r_1 = \frac{1}{(\delta T^{max})^2} \qquad r_2 = r_3 = r_4 = \frac{1}{(\delta\phi/\delta\theta/\delta\psi^{max})^2} \tag{6.4i}$$

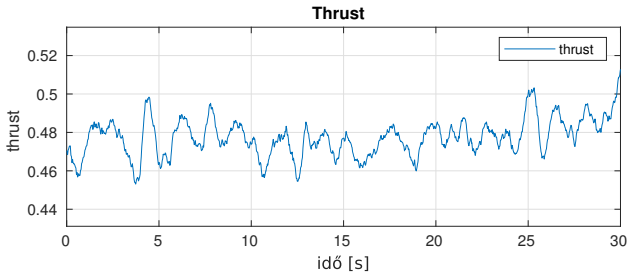
$$R = \begin{bmatrix} q_1 & & \\ & \ddots & \\ & & q_4 \end{bmatrix} \tag{6.4j}$$

6.5. Eredmények

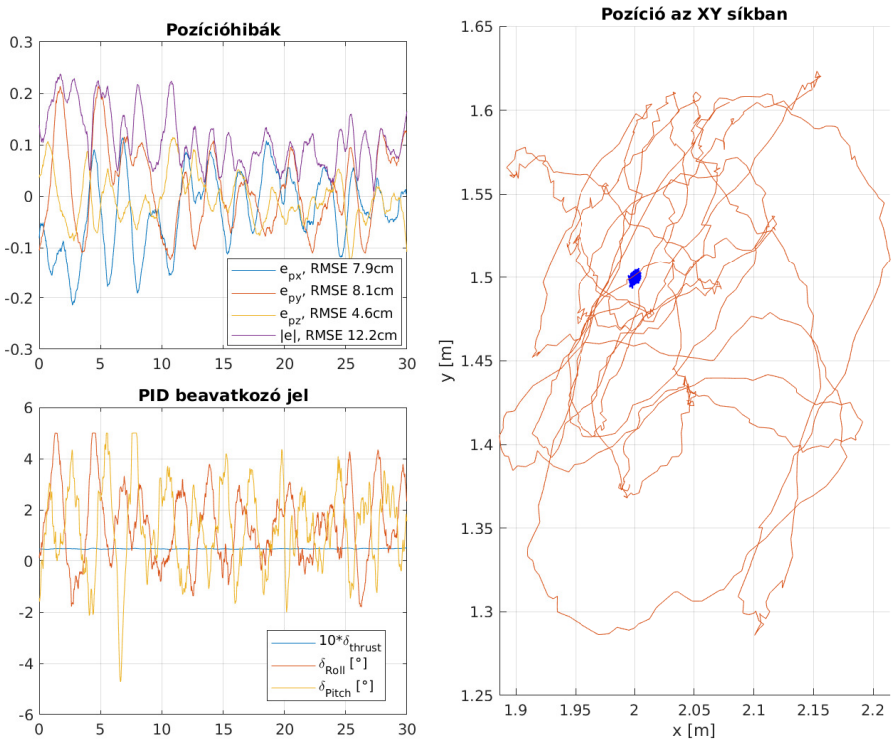
Mindhárom szabályozót teszteltük hover-ben, és egy egyszerű négyzet alakú trajektóriával.



(a) Pozíciók koordinátánként

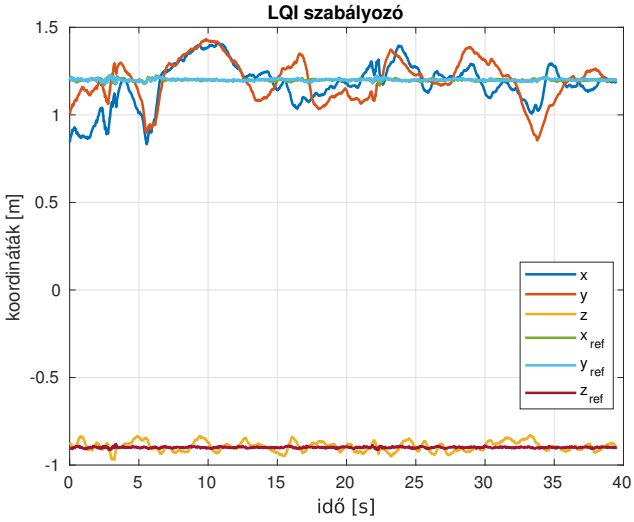


(b) Kiadott thrust

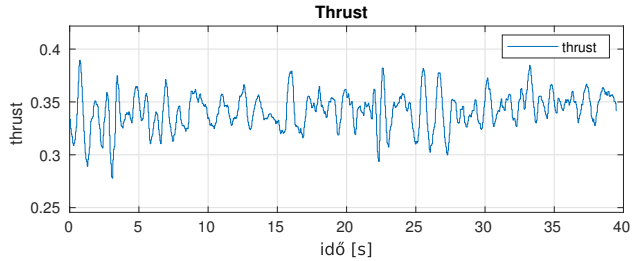


(c) PID szabályozó pozíció hibája és beavatkozó jelei

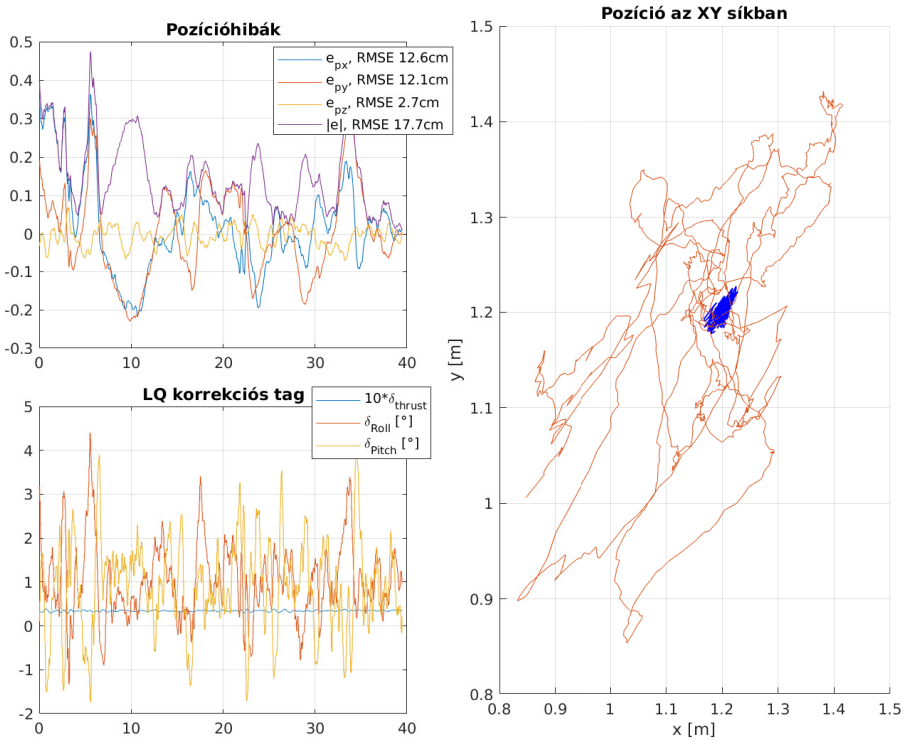
6.4. ábra. PID szabályozó performanciája



(a) Pozíciók koordinátánként

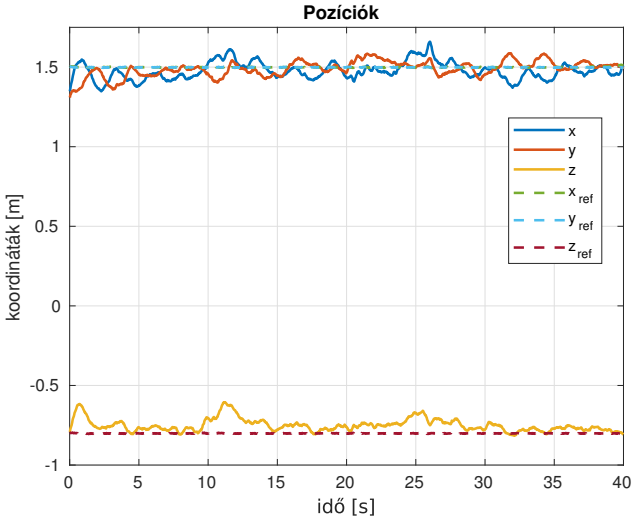


(b) Kiadott thrust

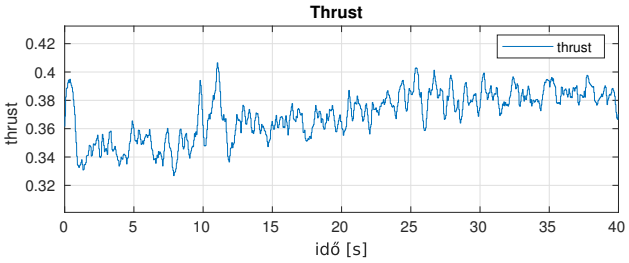


(c) LQI szabályozó pozíció hibája és beavatkozó jelei

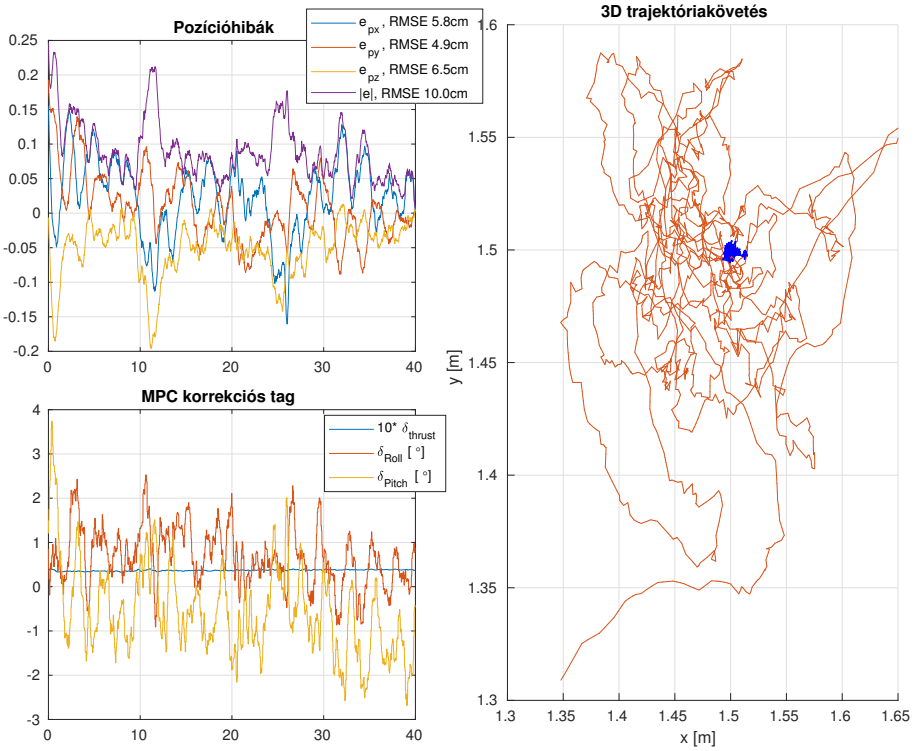
6.5. ábra. LQI szabályozó performanciája



(a) Pozíciók koordinátánként



(b) Kiadott thrust

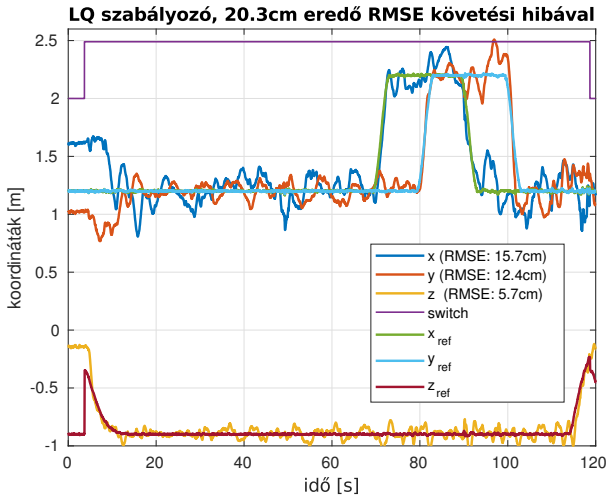


(c) MPC szabályozó hover-re (lebegésre)

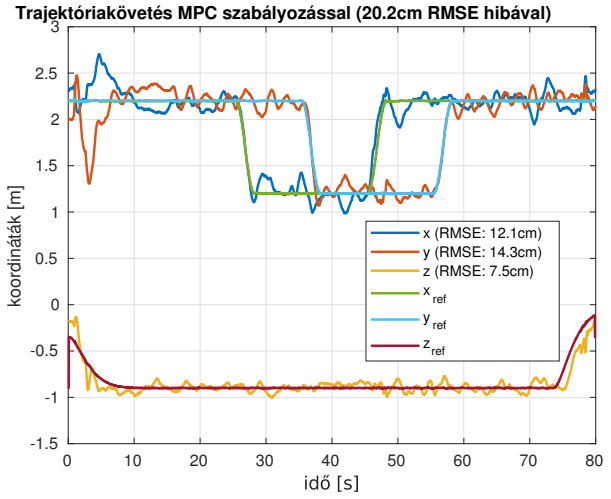
6.6. ábra. MPC szabályozó performanciája

6.6. Trajektóriakövetés tesztelése

A trajektóriakövetés tesztelésekor az LQ szabályozó esetében ugyanolyan performanciát kaptunk. A 6.7.3. fejezetben ismertetett algoritmusok tesztelésekor az MPC-t és az orientációszabályozást is finomhangoltuk, ezzel kaptuk a 6.6c. ábra szerinti 10 centiméteres pontosságot teljesítő szabályozást. A 6.8. ábra szerinti trajektóriát a (6.4) egyenlek szerinti súlyozással végrehajtva minden bizonnyal jobb eredmény született volna.



6.7. ábra. Trajektóriakövetés négyzet alakú trajektória mentén LQ szabályozóval



6.8. ábra. Trajektóriakövetés négyzet alakú trajektória mentén MPC szabályozóval.
(Az ábrán látotthoz képest újrahangolt 6.6c. ábrán látható MPC-vel jobb eredményt is elérhettünk volna megismételt repülési tesztekkel.)

6.7. Nemlineáris MPC keretrendszer

A kvadkopter hover állaptban tartásához megfelelőnek bizonyult a lineáris, időinvariáns modellt megvalósító MPC.

Az LTI modell-prediktív szabályozó továbbfejlesztéseként a `quadprog` numerikus megoldó helyett az `fmincon` használható, amely egy Matlab scriptben megírt költségfüggvény szerint képes az optimalizálást végrehajtani, így annak alakjára nincsenek a kvadratikus programozás során látott megkötések. A fejezet célja a nemlineáris MPC futtatására alkalmas keretrendszer implementálhatóságának vizsgálata, illetve abban egy LTV modellt használó szimuláció végrehajtása.

- A nemlineáris MPC egy nemlineáris programozási feladatra vezet (NLP)
- az NLP megoldható a Sequential Quadratic Programming (SQP) algoritmussal, amint a Matlab-ban az `fmincon` valósít meg [4]

6.7.1. Trajektória mentén előírt yaw szög lekövetése

Amennyiben a trajektória mentén a yaw szög változik, a 2.4. ábra alapján a referencia ϕ és θ szögek más-más irányba billentik a kvadkoptert ψ függvényében. A (2.9) szerinti trajektória menti linearizálást ezért elvégezhetjük a trajektóriageneráló algoritmus számára megadott értékekre. A trajektória végrehajtása során ekkor mindig aszerint a ψ szög szerint linearizált modellt használ az MPC szabályozás, ami megfelel az aktuálisnak. Ekkor nincsen szükség 3.1 transzformációra sem.

A 6.9. ábrán egy felszállás, egy kör megtétele változó yaw szöggel, és leszállás látható.

6.7.2. Kód generálás a nemlineáris MPC-hez

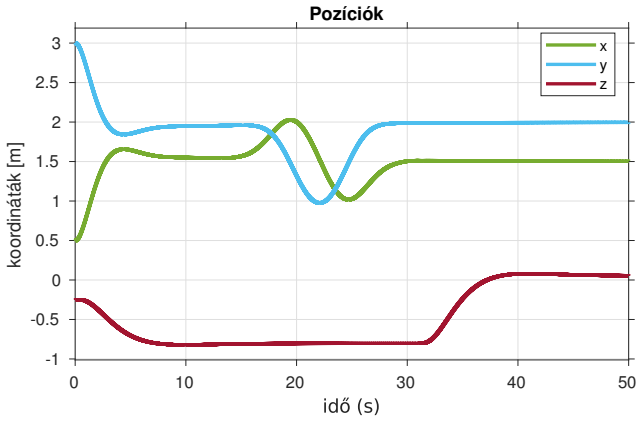
MATLAB-ban function handle-t kell használni, és a kiértékelendő költségfüggvényt egy külön m-fileban megírni. Az adatszerkezetek összeállításánál (például ha a hibadinamika linearizálását futásidőben szeretnénk elvégezni) figyelni kell arra, hogy azok előre nem ismert méretű elemeket, cell array-eket ne tartalmazzanak.

6.7.3. A futásidő csökkentésének lehetséges módjai

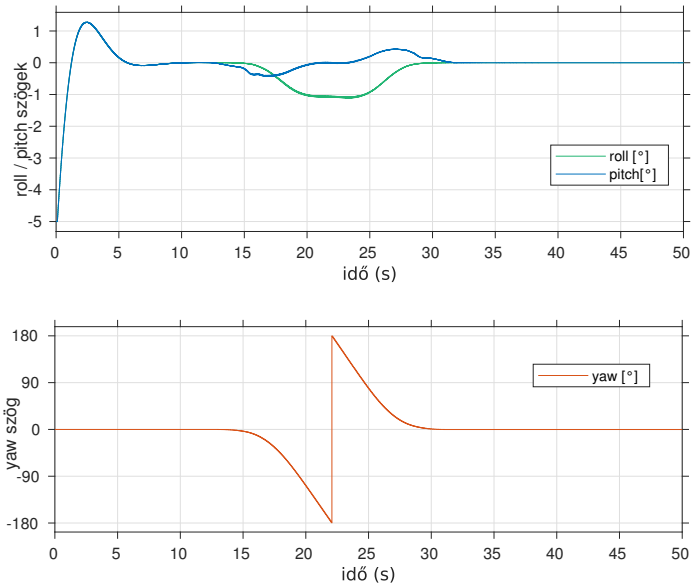
A nemlineáris MPC költségfüggvényének optimalizálását `fmincon`-nal végeztem. Ennek a futásideje az alapbeállítással túl nagy ahhoz, hogy valós időben futtatható legyen. Lehetséges viszont az `fmincon` számára lazább toleranciát, illetve maximális iterációszámot megadni. Ezekkel a beállításokkal valamennyivel ugyan csökken a predikció és így a szabályozás pontossága, de ha nem férne bele az optimalizáció a real-time környezet mintavételi idejébe, akkor instabillá is válhatna a zárt szabályozási körünk.

Többféle toleranciával elvégeztem a méréseket, és kis mértékben finomhangoltam az így kapott szabályozókat. Azt találtam, hogy 2.5 – 5 százalék körüli pontosság elegendő eredményt ad.

A nemlineáris optimalizálást a kvadkopter beágyazott számítógépén futtatva 60-75ms futásidő adódott, ami egy 0.1s mintavételi idejű szabályozás futtatását teszi lehetővé, ha biztonsági tartalékkal is számolunk.



(a) Kör alakú trajektória bejárása kezdeti pozíció hibával

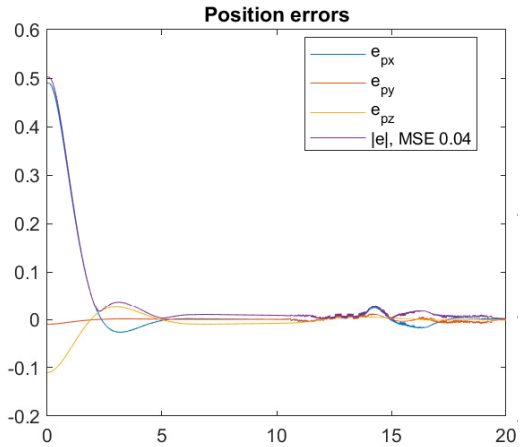


(b) A kör alakú trajektóriát a kopter $-1°$-os roll bedöntéssel járta be, yaw szöge a körrel érintőirányú.

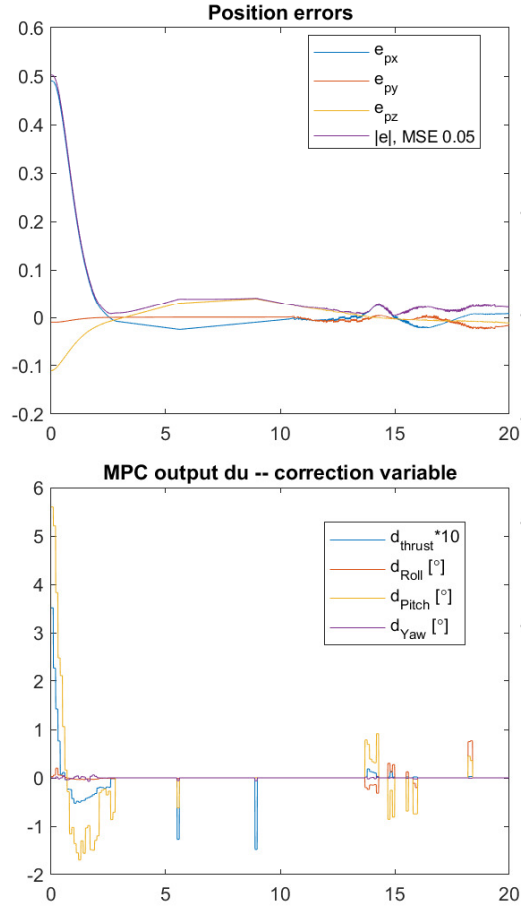
6.9. ábra. Kör alakú trajektória bejárásának az eredményei, trajektória mentén linearizált modell használatával

A beágyazott környezetben a log, telemetria szálak mellett fut a control szál, ami az állapotbecslést is végzi. Az ideális az lett volna, ha ki tudunk alakítani egy olyan szoftveres környezetet, hogy amíg tízszer lefut a névleges irányítást, állapotbecslést, logolást, és a Pixhawk-kal való kommunikációt futtató szál, addig egy külön magon legyen lehetőség a nemlineáris optimalizáció futtatására. Ám külön szálak indítása során `segmentation fault` üzeneteket kaptunk, ami vélhetően memóriafoglalással, vagy az indított szálak közötti kommunikáció hibájával magyarázható.

A legújabb, 2020b verziójú MATLAB támogatja az `lsqnonlin` optimalizálóhoz tartozó kód generálást, ami elviekben szintén megelelt volna az elvárásoknak, viszont 35ms körüli futásidővel sajnos itt sem lehetett volna megkerülni külön szál indítását és a szálkezelés körültekintőbb megvalósítását. A jelenlegi rendszerben felemelhettük volna a rendszer mintavételi idejét 50ms-re, viszont a csupán 20Hz-es állapotbecslés és névleges irányítás már jelentősen rosszabb lenne, mint a Flying Machine Arena [11] 5ms-es (200Hz) állapot-



(a) fmincon 0.5 százalékos tolerancia mellett
A beavatkozó jelek folytonosak, nincs nagy ugrásuk. Ezek a beavatkozójelek könnyen lekövethetők az orientációszabályozás számára. Nagyobb tolerancia esetén a szabályozás minősége számottevően nem romlik (szimulációban), viszont a beavatkozó jeleket a valós rendszeren nehezebben lehetne lekövetni.



(b) fmincon 10 százalékos toleranciával.
Tranziens alatt az iterációk száma 20-30, állandósult állapot elérésekor (szimulációban) akár 0 is lehet, ami azt jelenti, hogy az előző iteráció során számolt beavatkozójelsorozat megfelelő. Zajos mérések esetén valószínűleg a kezdeti 25 körüli iterációszám 10 körülre szorítható.

6.10. ábra. Különböző tolerancia beállítások jelentősen csökkenthetik a végrehajtáshoz szükséges időt. A MATLAB sajnos csak úgy képes kódot fordítani, hogy ez a tolerancia konstans; érdemes lenne viszont egy olyan esetet tesztelni, hogy a nagy kezdeti hiba esetén kis toleranciával dolgozunk, a trajektória közelében viszont nagyobb pontosságot követelünk meg. Valószínűleg a generált C kódban való módosításokkal elérhető lett volna többféle tolerancia használata, viszonta generált kód módosítása a fejlesztést nagyon lelassította volna.

becslési frekvenciájával[1], illetve a [3] szerinti 50Hz-es frekcenciától is jócskén elmaradna.

[1] A FMA-ban a pozíció-, sebesség- és orientációbecslés egyaránt Vicon kamerarendszer segítségével történik.

Ezért ezt a megoldási lehetőséget elvetettük, és arra a következtetésre jutottunk, hogy a nemlineáris MPC repülési tesztelését lehetővé tevő solverek használata a jelenlegi szoftver architektúrán nem lehetséges, annak jelentős átdolgozására van szükség.

7. fejezet

Továbbfejlesztés lehetőségei

7.1. Modellezés

A modellezés során elhanyagoltuk az aerodinamikai hatásokat. Ezek beltérben felszálláskor, vagy akadályok közelében okozhatnak problémát. Hasonlóan nem modelleztünk olyan fizikai tényezőket, mint a tápellátást biztosító akkumulátorok merülése, és az emiatt lecsökkenő felhajtóerő. A megfelelő skálafaktorokat a Pixhawk rendszeren belül lehetne finomhangolni, ugyanis az eszköz képes mérni a motorok tápellátását adó Li-Po (lítium-polimer) akkumulátorok töltöttségi szintjét.

Az orientációkövetést modellezhetjük [13] szerint egytárolós taggal, amennyiben a Pixhawk-on már nem lehetséges az orientációkövetés további finomhangolása.

7.2. Trajektóriatervezés

A jelenlegi keretrendszer lehetővé teszi azt, hogy a különböző költségfüggvényeket használjunk, amelyek az adott trajektórián különböző szempontból optimális sebességprofilt eredményeznek [9].

7.3. LQI szabályozó

A beavatkozó jel sávkorlátosságát lehetne figyelembe venni a [2] által javasolt loopshaping megoldással. A szabályozó tervezése során a költségfüggvény mátrixait választhatjuk frekvenciafüggőnek is.

7.4. MPC szabályozás

A trajektóriatervezés során a változó yaw szögeket tartalmazó trajektóriákat (például kör alakzat bejárása érintőirányú ψ szöggel) csak szimulációban sikerült működésre bírni, a valódi rendszeren még tesztelés alatt állnak. Ez a teszt validálhatná az LTV modell használatának előnyeit, ugyanis a rendszerről kapott plusz információval pontosabb szabályozást tehetne lehetővé.

7.4.1. EKF és MPC együttes használata

A repülési tesztek során az a szabályozó már az állapotbecslőtől kapott pozíciókat, sebességeket és orientációkat kapta, az állapotbecslésen hallgatótársam dolgozott. Az állapotbecslés [18] szerinti pozicionáló rendszerrel történt, egy paraméter nélküli EKF használatával. Ez azt jelenti, hogy az állapotbecsléshez nem lett felhasználva a kvadkopter dinamikus modellje. A becsült pozíciókat és a sebességeket az UWB és IMU méréseiből, illetve az orientációk segítségével kapjuk. Az MPC az állapotbecsléstől függetlenül működött.

Amennyiben az EKF a kvadkopter paramétereit felhasználó modellel dolgozik, minden lépésben el kell, hogy végezze a nemlineáris dinamika linearizálását. Az MPC ugyanezt a linearizált modellt fel tudja használni a mozgó horizontú becslésekhez, így számítási kapacitást lehet megtakarítani [22, 15-18. o.]. Mivel a dinamikus modellt használó EKF sajnos nem működött a kvadkopteren, az MPC-vel való integrációt egyelőre nem sikerült megvalósítani.

7.4.2. Numerikus optimalizálás gyorsítása

A kvadratikus programozási feladat struktúrájának kihasználásával olyan numerikus megoldók használatára nyílik lehetőség, amelyekkel az optimalizálási probléma sokkal rövidebb idő alatt megoldható, így komplexebb feladatok valós idejű végrehajtása is lehetséges lenne.

A MATLAB legjabb, 2021a verziójának megjelenés előtti összefoglalójában szerepel az `lsqnonlin` és `fmincon` függvények végrehajtási idejének gyorsítása. Érdemes mindig a legfrissebb verziót használni, hiszen az R2019b verzió például még nem támogatta az `lsqnonlin` függvény számára a kód generálás lehetőségét.

7.5. Pozícionáló rendszer

A szabályozást érdemes lenne egy [3], illetve [15] által használt Vicon vagy egy Optitrack[1] pozícionáló rendszerrel kipróbálni. Egy ilyen kísérlettel meg lehetne határozni azt, hogy a szabályozás mennyire érzékeny a jelenlegi pozícionáló rendszer zajosabb méréseire, és mi az a legjobb eredmény, amit teljesen pontos mérések[2] esetén el tud érni.

[1] Az Optitrack rendszer (`https://optitrack.com/applications/robotics/`) képes együttműködni a Pixhawk-kal is: `https://ardupilot.org/copter/docs/common-optitrack.html`

[2] Az Optitrack rendszer oldalán 0.3mm-es pozíció és 0.05°-os orientációhibát említ, összehasonlításképp a mi beltéri rendszerünk 3-4cm-es pontosságú [18].

8. fejezet

Összefoglalás

A kifejlesztett algoritmusok kipróbálása egy kihívásokkal teli folyamat volt. A repülési tesztek során lehetőség volt a szabályozások közelebbi megismerésére, és lépésről lépésre történő fejlesztésére. Az egyre növekvő komplexitású algoritmusokkal egyre több feladatot képes a kvadkopter ellátni. Az alkalmazott fedélzeti rendszerek számos autonóm járművön megtalálhatóak, így a megszerzett tudásunkat az egész szakterületen kamatoztatni lehet. A trajektóriatervező algoritmus implementálásával megismertem a vonatkozó szakirodalmat, és optimalizálási problémákat, a trajektória menti linearizálással időben változó modellt használtam a szabályozótervezéshez. Már jól ismert szabályozókat próbáltam ki és hasonlítottam össze a teljesítményüket. A felhasznált forrásokból megismerhettem a feladathoz kapcsolódó elméleti alapokat és további fejlesztési ötleteket kaptam belőlük.

A drónaréna rendszer az elvégzett repülési tesztekkel már egy bejáratott platform, ahol a bemutatott alapokra építve lehetőség van más komponensekből felépülő rendszert tesztelni, és a teljesítményét az elvégzett mérésekkel összehasonlítani. Bízom benne, hogy a platformon tanulva mások is elmélyíthetik tudásukat és kézzel fogható tapasztalatot gyűjthetnek.

Köszönetnyilvánítás

Szeretnék segítségéért köszönetet mondani konzulensemnek, Luspay Tamásnak a szakmai segítségért, feleségemnek, Vivinek a támogatásáért és Marcinak a repülési tesztekben való segítségért.

Irodalomjegyzék

[1] K. Alexis – C. Papachristos – R. Siegwart – A. Tzes: Robust explicit model predictive flight control of unmanned rotorcrafts: Design and experimental evaluation. In *2014 European Control Conference (ECC)* (konferenciaanyag). 2014, 498–503. p. URL https://ieeexplore.ieee.org/document/6862269.

[2] Brian D. O. Anderson – John B. Moore: *Optimal Control: Linear Quadratic Methods*. USA, 1990, Prentice-Hall, Inc. ISBN 0136385605.

[3] Moses Bangura – Robert Mahony: Real-time Model Predictive Control for Quadrotors. *IFAC Proceedings Volumes*, 47. évf. (2014) 3. sz., 11773 – 11780. p. ISSN 1474-6670. URL http://www.sciencedirect.com/science/article/pii/S1474667016434890. 19th IFAC World Congress.

[4] Alberto Bemporad: Linear time-varying and nonlinear MPC. http://cse.lab.imtlucca.it/~bemporad/teaching/mpc/imt/2-ltv_nl_mpc.pdf. Hozzáférés dátuma: 2020-12-15.

[5] S. Bouabdallah – A. Noth – R. Siegwart: Pid vs lq control techniques applied to an indoor micro quadrotor. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)* (konferenciaanyag), 3. köt. 2004, 2451–2456 vol.3. p.

[6] Yasser Bouktir – Moussa Haddad – Taha Chettibi: Trajectory planning for a quadrotor helicopter. *2008 16th Mediterranean Conference on Control and Automation*, 2008., 1258–1263. p. URL https://ieeexplore.ieee.org/document/4602025.

[7] Mark Cannon: C21 Model Predictive Control. https://markcannon.github.io/assets/downloads/teaching/C21_Model_Predictive_Control/mpc_notes.pdf. Hozzáférés dátuma: 2020-12-15.

[8] Choose Among Types of Model Components, MATLAB Simulink. https://www.mathworks.com/help/simulink/ug/types-of-model-components.html. Hozzáférés dátuma: 2020-12-15.

[9] Karam Eliker – Guoqing Zhang – Said Grouni – Weidong Zhang: An Optimization Problem for Quadcopter Reference Flight Trajectory Generation. *Journal of Advanced Transportation*, 2018. évf. (2018. 07), 1–15. p.

[10] Generate Code for quadprog, MATLAB, Optimization Toolbox. https://www.mathworks.com/help/optim/ug/code-generation-for-quadprog-example.html. Hozzáférés dátuma: 2020-12-15.

[11] Markus Hehn – Raffaello D'Andrea: Quadrocopter trajectory generation and control. *IFAC Proceedings Volumes*, 44. évf. (2011) 1. sz., 1485 – 1491. p.

ISSN 1474-6670. URL http://www.sciencedirect.com/science/article/pii/S147466701643819X. 18th IFAC World Congress.

[12] E.G. Hernandez-Martinez – G. Fernandez-Anaya – E.D. Ferreira – J.J. Flores-Godoy – A. Lopez-Gonzalez: Trajectory tracking of a quadcopter uav with optimal translational control. *IFAC-PapersOnLine*, 48. évf. (2015) 19. sz., 226 – 231. p. ISSN 2405-8963. URL https://doi.org/10.1016/j.ifacol.2015.12.038. 11th IFAC Symposium on Robot Control SYROCO 2015.

[13] Mina Kamel – Thomas Stastny – Kostas Alexis – Roland Siegwart: *Model Predictive Control for Trajectory Tracking of Unmanned Aerial Vehicles Using Robot Operating System*. Cham, 2017, Springer International Publishing, 3–39. p. ISBN 978-3-319-54927-9. URL https://doi.org/10.1007/978-3-319-54927-9_1.

[14] Tetsuzo Kuragano – Kazuhiro Kasono: B-spline curve generation and modification based on specified radius of curvature. *WSEAS Transactions on Information Science and Applications*, 5. évf. (2008. 01).

[15] Sergei Lupashin – Markus Hehn – Mark W. Mueller – Angela P. Schoellig – Michael Sherback – Raffaello D'Andrea: A platform for aerial robotics research and demonstration: The flying machine arena. *Mechatronics*, 24. évf. (2014) 1. sz., 41 – 54. p. ISSN 0957-4158. URL http://www.sciencedirect.com/science/article/pii/S0957415813002262.

[16] ME233 Advanced Control II, Lecture 1, Dynamic Programming & Optimal Linear Quadratic Regulators (LQR). https://berkeley-me233.github.io/static/ME233_Sp16_L1_DP_Optimal_LQR.pdf. Hozzáférés dátuma: 2020-12-15.

[17] D. Mellinger – V. Kumar: Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE International Conference on Robotics and Automation* (konferenciaanyag). 2011, 2520–2525. p.
URL https://ieeexplore.ieee.org/document/5980409.

[18] M. Molnár – T. Luspay: Development of an UWB based Indoor Positioning System. In *2020 28th Mediterranean Conference on Control and Automation (MED)* (konferenciaanyag). 2020, 820–825. p.

[19] Richard M. Murray: CDS 110b Control and Dynamical Systems, Lecture 2 – LQR Control. https://www.cds.caltech.edu/~murray/courses/cds110/wi06/lqr.pdf. Hozzáférés dátuma: 2020-12-15.

[20] Pixhawk Ardupilot Extended Kalman Filter (EKF) for orientation estimation. https://ardupilot.org/copter/docs/common-apm-navigation-extended-kalman-filter-overview.html. Hozzáférés dátuma: 2020-12-15.

[21] Anders Rantzer: Linearization around a trajectory FRTN05 - Nonlinear Control and Servo Systems. http://www.control.lth.se/fileadmin/control/Education/EngineeringProgram/FRTN05/2019/lec02eight.pdf. Hozzáférés dátuma: 2020-12-15.

[22] C.V. Rao: *Moving Horizon Strategies for the Constrained Monitoring and Control of Nonlinear Discrete-time Systems*. 2000, University of Wisconsin–Madison. URL https://books.google.hu/books?id=b67SAAAAMAAJ.

[23] Kameshwar Poolla Roberto Horowitz, Andrew Packard: ME132 Dynamic Systems and Feedback, Class Notes. https://www.cds.caltech.edu/~murray/courses/cds101/fa02/caltech/pph02-ch19-23.pdf. Hozzáférés dátuma: 2020-12-15.

[24] Pengkai Ru – Kamesh Subbarao: Nonlinear model predictive control for unmanned aerial vehicles. *Aerospace*, 4. évf. (2017. 06), 31. p.

[25] David Salomon: *Curves and Surfaces for Computer Graphics*. 2005. 01.

[26] G. Stein – M. Athans: The LQG/LTR procedure for multivariable feedback control design. *IEEE Transactions on Automatic Control*, 32. évf. (1987) 2. sz., 105–114. p. URL https://dspace.mit.edu/handle/1721.1/1109.

[27] F. Stoican – I. Prodan – D. Popescu: Flat trajectory generation for way-points relaxations and obstacle avoidance. In *2015 23rd Mediterranean Conference on Control and Automation (MED)* (konferenciaanyag). 2015, 695–700. p.

[28] V. Usenko – L. von Stumberg – A. Pangercic – D. Cremers: Real-time trajectory replanning for MAVs using uniform B-splines and a 3D circular buffer. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (konferenciaanyag). 2017, 215–222. p. URL https://arxiv.org/abs/1703.01416.

[29] Thomas T.R. van de Wiel – Roland Tóth – Vsevolod I. Kiriouchine: Comparison of parameter-varying decoupling based control schemes for a quadrotor. *IFAC-PapersOnLine*, 51. évf. (2018) 26. sz., 55 – 61. p. ISSN 2405-8963. URL http://www.sciencedirect.com/science/article/pii/S2405896318328313. 2nd IFAC Workshop on Linear Parameter Varying Systems LPVS 2018.