Collision-Free Mixed-Integer Planning for Quadrotors Using Convex Safe Regions

6.832 Underactuated Robotics - Final Project

Bernhard Paus Graesdal
Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

***Abstract*—In this project, a complete collision-free planning scheme for quadrotors has been devised. Because of the differentially flat nature of quadrotors, trajectories are constructed from polynomial segments. The planner first generates safe, convex regions, to which the polynomial trajectory segments are later confined by a mixed-integer optimization program. By enforcing SOS constraints on the polynomial, this approach is further able to guarantee that the trajectory stays collision free for its entire duration, and not just at a finite set of sample points. Finally, the trajectory is stabilized using a Time-Varying-LQR design.**

I. Introduction

This project report is structured as follows: In II, the entire trajectory generator is described, implementing the trajectory generation problem as a Mixed-Integer optimization problem. In III, the quadrotor model and its differential flatness property is described. Then, in IV, a TVLQR controller stabilizing the quadrotor around the generated trajectory is constructed. Finally, in V, the system is implemented in simulation and experiments are performed. A video of the results can be seen at [1].

II. Trajectory Generation

First, a trajectory is to be generated for the quadrotor to follow. This part is based on the results in [2].

A. Polynomial Trajectories

As will be detailed in III the quadrotor model is *differentially flat* [3], meaning that any trajectory designed in the flat output space will be dynamically feasible, as long as it is differentiable up to a certain degree. This effectively lets us disregard the dynamics of the plant in the trajectory optimization process. Therefore, the trajectories will be generated in the flat output space $\sigma = [x,\, y,\, z]^\top$.

More generally, the trajectories will be represented as piecewise polynomials in one variable t, of dimension n and of degree $d+1$. In the case where trajectories are generated for the 3 dimensional flat output space, $n = 3$ is used. Each polynomial segment $j = 1, \ldots, N$ will be parametrized by its coefficients of a set of basis functions. Further, the basis functions are chosen as the set all monomials of t up to degree d: $m(t) = [1,\, t,\, t^2,\, \ldots\, , t^d]^\top$. Let $C_{jk} \in \mathbb{R}^n$ denote the vector of coefficients for segment j corresponding the monomial with degree k. Then, for each segment j, the trajectory can be evaluated as

$$P_j(t) = \sum_{k=1}^{d+1} C_{j,k} m_k(t) \quad t \in [0,1] \tag{1}$$

Enforcing continuity of the entire trajectory requires linear constraints on the values of the polynomial segments at times $t = 0$ and $t = 1$. In this project, the polynomial trajectories will be required to be continous up to order $d-1$. The following linear constraints are implemented for $j = 1, \ldots, N-1$

$$P_j(1) = P_{j+1}(0), \tag{2}$$
$$\dot{P}_j(1) = \dot{P}_{j+1}(0), \tag{3}$$
$$\vdots$$
$$P_j{}^{(d-1)}(1) = P_{j+1}{}^{(d-1)}(0)$$

In addition to this, linear constraints for the initial and final position, velocity and acceleration of the quadrotor will be added:

$$P_1(0) = X_0, \quad \dot{P}_1(0) = 0, \quad \ddot{P}_1(0) = 0 \tag{4}$$
$$P_1(N) = X_f, \quad \dot{P}_N(1) = 0, \quad \ddot{P}_N(1) = 0 \tag{5}$$

Similar to what is done in [2] and [3], we will attempt to minimize the jerk, that is, the second derivative of acceleration. As argued in [3], this can be directly as this can be related to the inputs, thus minimizing the required actuator efforts.

B. Generating Safe Convex Regions

In contrast to many similar approaches for obstacle-avoidance, this project does not implement constraints on each of the half-spaces of the convex obstacles [4]. Instead, like done in [2], the trajectory will instead be confined to lie within safe convex regions generated from the obstacle filled space.

In order to generate convex safe regions, an open-source library for computing large convex regions of obstacle-free space through Semi-Definite Programming (SDP) called IRIS [5] is used. IRIS takes in a set of obstacles given by their vertices, and grows the largest possible convex region from an initial seed point. Thus, given the right seed points, the obstacle filled space can be sub-divided into smaller, convex, safe regions.

Generating the best seed points for IRIS is in itself a hard problem. For this project, a heurustic based on dividing the entire workspace into a coarse grid, and then finding the next non-colliding point that maximizes the distance to all obstacles and previously computed regions. In this context, non-colliding means that the point is neither inside an obstacle, nor inside a previously computed convex region.

C. Mixed-Integer Optimization: Assigning each segment to a region

To keep the entire trajectory collision-free, each polynomial segment is restriced to lie within a single convex region. To enfore this, binary integer variables are used. For every combination of region r and segment j, there will be a binary decision variable added to the optimization problem, indicating whether that segment is confined to lie within that region. For a total of R regions and N segments, this gives a matrix of binary decision variables $H \in \{0,1\}^{R\times N}$, where

$$H_{r,j} \implies P_j(t) \in G_r \quad \forall t \in [0,1] \tag{6}$$

for the polynomial segment $P_j(t)$ and the convex region G_r. Implementation of this constraint will be detailed in the following sections.

In order to guarantee that the path is indeed collision-free, each polynomial segment must only be confined to one convex region. This means that any polynomial segment is always guaranteed to be entirely within one convex region, even though it might also happen to lie entirely or partly within other regions. This is implemented as a linear constraint on the decision variables $H_{r,j}$:

$$\sum_{r=1}^{R} H_{r,j} = 1 \tag{7}$$

D. SOS Constraints: Restricting a polynomial to a polytope

To implement the constraint in (6), sums-of-squares constraints will be used.

In general, a polytope can be expressed as a set of linear inequality constraints $Ax \leq b$. For this project, we will constrain the convex regions to be polytopes, such that the same representation can be used. Ensuring that the entire polynomial segment is within region G_r can therefore be expressed as:

$$A_r P_j(t) \leq b_r \quad t \in [0,1] \tag{8}$$

where $A_r \in \mathbb{R}^{m\times n}$ and $b_r \in \mathbb{R}^m$ describes the polytope corresponding to the convex region r.

By using the evaluation of each polynomial segment in (1), and by noting that (8) can be divided up into m independent constraints, one can write:

$$a_{r,l}^\top \sum_{k=1}^{d+1} C_{j,k} m_k(t) \leq b_{r,l} \quad \forall t \in [0,1] \quad l = 1,...,m \tag{9}$$

Redistribution and moving all terms to the right side yields:

$$q_l(t) := b_{r,l} - \sum_{k=1}^{d+1} (a_{r,l}^\top C_{j,k}) m_k(t) \geq 0 \quad \forall t \in [0,1] \tag{10}$$

for $l = 1,...,m$. This can be can be enforced as a SOS constraint on $q_l(t)$. In the case where d is odd, $q(t) \geq 0 \quad \forall t \in [0,1]$ will hold if and only if $q(t)$ can be written as:

$$\begin{gathered} q(t) = t\sigma_1(t) + (1-t)\sigma_2(t) \\ \sigma_1(t),\ \sigma_2(t) \text{ are sums of squares} \end{gathered} \tag{11}$$

where $\sigma_1(t)$ and $\sigma_2(t)$ are polynomials in t of degree $d-1$ [6]. In this project, either $d = 3$ or $d = 5$ will be chosen, hende d will always be odd.

Therefore, (10) is enforced by ensuring that the coefficients of the polynomial $q(t)$ are equal to the coefficients of the resulting polynomial expressed in (11), and by adding the constraint that $\sigma_1(t)$, $\sigma_2(t)$ are both sums-of-squares.

E. Reducing the SOS constraint to a SOCP constraint

When $d = 3$, the SOS constraint from the previous section can actually be reduced to a Second-Order-Cone-Program (SOCP) constraint. This will be important when implementing the optimization problem.

Consider the case when $d = 3$. This means that $\sigma_i(t)$ in (11) will be a quadratic polynomial:

$$\sigma_i(t) = \beta_1 + \beta_2 t + \beta_3 t^2 \ \geq 0 \tag{12}$$

Using the standard way to reformulate sums-of-squares into a SDP constraint, one can equivalently write

$$\sigma_i(t) = \begin{bmatrix}1 & t\end{bmatrix} \begin{bmatrix}\beta_1 & \frac{\beta_2}{2} \\ \frac{\beta_2}{2} & \beta_3\end{bmatrix} \begin{bmatrix}1 \\ t\end{bmatrix}, \quad \begin{bmatrix}\beta_1 & \frac{\beta_2}{2} \\ \frac{\beta_2}{2} & \beta_3\end{bmatrix} \succeq 0 \tag{13}$$

As the matrix is symmetric, this is equivalent requiring that all the leading principal minors are all non-negative:

$$\begin{gathered} \beta_1 \geq 0 \\ \beta_1\beta_3 - \frac{1}{4}\beta_2^2 \geq 0 \end{gathered} \tag{14}$$

In general, a large class of convex problems can be cast as a Second-Order-Cone-Program (SOCP) using hyperbolic constraints [7]:

$$w^2 \leq xy,\ x \geq 0,\ y \geq 0 \iff \left\| \begin{bmatrix}2w \\ x-y\end{bmatrix} \right\| \leq x+y \tag{15}$$

Using this fact, (14) can easily be reformulated as a SOCP constraint:

$$\left\| \begin{bmatrix}\beta_2 \\ \beta_1 - \beta_3\end{bmatrix} \right\| \leq \beta_1 + \beta_3 \tag{16}$$

Therefore, for degree $d = 3$, the MISDP problem reduces to a MISOCP problem, which can be solved efficiently by many commercially available solvers, or by a Branch-and-Bound technique.

F. Solving the optimization problem

In short, implementation of a sums-of-squares constraint is done by implementing a constraint on positive-semi-definiteness of the matrix of the coefficients of the SOS polynomial [6]. Hence, the optimization problem in II-D will be of the problem class Mixed-Integer-Semi-Definite-Program (MISDP). This class of problem is not readily solvable by most commercially available solvers. While solving these problems is still possible, according to [2], this quickly leads to numerical instability for polynomials of degree 7 or higher. For implementation, we will therefore look to reduce the problem from the original MISDP class.

As will be shown in III, the quadrotor model will require the polynomial trajectory to be of degree $d = 5$. As seen in II-E, by choosing $d = 3$ the problem reduces to a MISOCP problem, which is a significantly easier class of optimization problems. Therefore, we first solve the problem for $d = 3$, minimizing the norm of the second order derivative of the position. Then, to obtain the required trajectory of $d = 5$, the problem is solved again, but this time with the binary decision variables $H_{r,j}$ fixed to their assigned values from solving the problem with $d = 3$. For the 5-th order trajectory, the norm of the jerk is minimized. This means that we first solve a MISOCP problem for $d = 3$, then a regular SDP problem for $d = 5$, allowing us to obtain a full fifth-order polynomial trajectory without explicitly solving the MISDP problem from II-D.

G. Complete problem formulation

First, the following problem generating a third-degree polynomial is solved:

$$\min_{P_j, H_{r,j}, \sigma_i} \sum_{j=1}^{N} \left\| \frac{d^2}{dt^2} P_j(t) \right\|^2 \text{ subject to} \tag{17}$$

$$\begin{aligned} P_j(1) &= P_{j+1}(0), \\ \dot{P}_j(1) &= \dot{P}_{j+1}(0), \\ \ddot{P}_j(1) &= \ddot{P}_{j+1}(0) \end{aligned} \tag{18}$$

$$\begin{aligned} P_1(0) = X_0, \quad \dot{P}_1(0) = 0, \quad \ddot{P}_1(0) = 0 \\ P_1(N) = X_f, \quad \dot{P}_N(1) = 0, \quad \ddot{P}_N(1) = 0 \end{aligned} \tag{19}$$

$$H_{r,j} \in \{0, 1\} \tag{20}$$

$$H_{r,j} \implies b_{r,l} - a_{r,l}^\top P_j(t) = t\sigma_1(t) + (1-t)\sigma_2(t) \tag{21}$$

$$\sigma_i(t) = \beta_1 + \beta_2 t + \beta_3 t^2 \tag{22}$$

$$\left\| \begin{bmatrix} \beta_2 \\ \beta_1 - \beta_3 \end{bmatrix} \right\| \leq \beta_1 + \beta_3 \tag{23}$$

$$\sum_{r=1}^{R} H_{r,j} = 1, \quad \forall j \in \{1, \ldots, N\} \tag{24}$$

Then, with $H_{r,j}$ fixed at the values from the previous problem and no longer being a decision variable, the following problem is solved to obtain the 5-th order trajectory that will be followed by the quadrotor:

$$\min_{P_j, \sigma_i} \sum_{j=1}^{N} \left\| \frac{d^4}{dt^4} P_j(t) \right\|^2 \text{ subject to} \tag{25}$$

$$\begin{aligned} P_j(1) &= P_{j+1}(0), \\ \dot{P}_j(1) &= \dot{P}_{j+1}(0), \\ &\vdots \\ {P_j}^{(4)}(1) &= {P_{j+1}}^{(4)}(0) \end{aligned} \tag{26}$$

$$\begin{aligned} P_1(0) = X_0, \quad \dot{P}_1(0) = 0, \quad \ddot{P}_1(0) = 0 \\ P_1(N) = X_f, \quad \dot{P}_N(1) = 0, \quad \ddot{P}_N(1) = 0 \end{aligned} \tag{27}$$

$$H_{r,j} \implies b_{r,l} - a_{r,l}^\top P_j(t) = t\sigma_1(t) + (1-t)\sigma_2(t) \tag{28}$$

$$\sigma_1(t),\ \sigma_2(t) \text{ are sums of squares} \tag{29}$$

For both problems, the safe convex regions are computed automatically, while the number of polynomial segments are chosen beforehand.

III. Modelling and Differential Flatness

The quadrotor model that is used can be shown to be differentially flat, meaning that any trajectory designed in the output space will be dynamically feasible. In the case of the quadrotor, the chosen flat output space is $\boldsymbol{\sigma} = [x,\ y,\ z,\ \psi]^\top$, where ψ denotes the yaw angle of the quadrotor.

The attentive reader might notice that ψ is disregarded in II; for the trajectory optimization part of this project, $\psi = 0$ has been chosen, as ψ does not matter for pure obstacle-avoidance. In general, ψ can be chosen to be any two times differentiable trajectory, as is shown in [3].

A. Quadrotor Model

For the modelling of the quadrotor, two coordinate frames are considered: Let $\mathcal{B}$ denote the body frame, and let $\mathcal{W}$ denote the world frame.

From using Newton-Eulers equation of motion, the angular acceleration of the quadrotor can be given as:

$$\dot{\boldsymbol{\omega}}_{\mathcal{BW}} = \boldsymbol{J}^{-1}[-\boldsymbol{\omega}_{\mathcal{BW}} \times \boldsymbol{J}\boldsymbol{\omega}_{\mathcal{BW}} + \boldsymbol{\tau}_c] \tag{30}$$

where $\boldsymbol{\omega}_{\mathcal{BW}} = [p,\ q,\ r]^\top$ denotes the angular velocity of the body frame relative to the world frame, $\boldsymbol{J}$ denotes the inertia matrix, and $\boldsymbol{\tau}_c$ are the commanded torques.

The acceleration of the center of mass is given as:

$$m\ddot{\boldsymbol{r}} = -mg\boldsymbol{z}_w + F_{th}\boldsymbol{z}_b \tag{31}$$

where m denotes the mass of the quadrotor, g denotes gravitational acceleration and F_{th} denotes the total commanded thrust. By denoting $\boldsymbol{R}_{\mathcal{WB}}$ as the rotation matrix from the body frame to the world frame, (31) can be reformulated as:

$$m \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = -mg \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + F_{th}\boldsymbol{R}_{\mathcal{WB}} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{32}$$

Parametrizing the rotations by Euler angles gives the full state vector:

$$\boldsymbol{x} = [x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, p, q, r]^\top \tag{33}$$

The real world quadrotor and the quadrotor used in simulation cannot directly control the input torques, however. Assuming that the squared propeller speeds of the quadrotor can be instantly controlled, we define

$$\boldsymbol{u} = [\omega_1^2, \omega_2^2, \omega_3^2, \omega_4^2]^\top \tag{34}$$

The virtual inputs F_{th} and $\boldsymbol{\tau}_c$ are transformed to squared propeller speeds by using the following relationship:

$$\begin{bmatrix} F_{th} \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} k_F & k_F & k_F & k_F \\ 0 & k_F L & 0 & -k_F L \\ -k_F L & 0 & k_F L & 0 \\ k_m & k_m & k_m & k_m \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \tag{35}$$

which can be derived from the quadrotor geometry. k_F and k_m are constants relating the propeller speed to the generated force and torque, respectively, and L is the length of the quadrotor arms.

Finally, the entire system can be described as:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \tag{36}$$

B. Differential Flatness of The Quadrotor

The differential flatness property gives that, given a trajectory in the flat output space, the resulting full-state trajectory can be explicitly calculated. The differential flatness property exploited in this project are the exact same as described in [3]. For completeness, the equations used in this project are included here:

First, introduce a third coordinate frame, the $\mathcal{C}$ frame, located at the center of mass of the quadrotor, and rotated only by ψ around the z-axis. Then, the required attitude of the quadrotor can be calculated from:

$$\boldsymbol{z}_B = \frac{\boldsymbol{t}}{\|\boldsymbol{t}\|}, \quad \boldsymbol{t} = [\ddot{x}, \ddot{y}, \ddot{z} + g]^T \tag{37}$$

$$\boldsymbol{x}_C = [\cos\psi, \sin\psi, 0]^T \tag{38}$$

$$\boldsymbol{y}_B = \frac{\mathbf{z}_B \times \mathbf{x}_C}{\|\mathbf{z}_B \times \mathbf{x}_C\|}, \quad \mathbf{x}_B = \mathbf{y}_B \times \mathbf{z}_B \tag{39}$$

$$\boldsymbol{R}_{\mathcal{WB}} = [\boldsymbol{x}_B\, \boldsymbol{y}_B\, \boldsymbol{z}_B] \tag{40}$$

Next, we look towards calculating the angular velocities $\boldsymbol{\omega}_{\mathcal{BW}}$. First, differentiating (31) once gives:

$$m\dot{\boldsymbol{a}} = \dot{F}_{th}\boldsymbol{z}_B + \boldsymbol{\omega}_{\mathcal{BW}} \times F_{th}\boldsymbol{z}_B \tag{41}$$

Taking the dot product of this expression along $\boldsymbol{z}_b$ gives $\dot{F}_{th} = \boldsymbol{z}_b \cdot m\dot{\boldsymbol{a}}$. Inserting this back into (41) and diving everything by F_{th} gives:

$$\boldsymbol{h}_\omega := \boldsymbol{\omega}_{\mathcal{BW}} \times \boldsymbol{z}_B = \frac{m}{F_{th}}(\dot{\boldsymbol{a}} - (\boldsymbol{z}_B \cdot \dot{\boldsymbol{a}})\boldsymbol{z}_B) \tag{42}$$

From this, the angular velocities can be calculated as:

$$p = -\boldsymbol{h}_\omega \cdot \boldsymbol{y}_B \tag{43}$$

$$q = \boldsymbol{h}_\omega \cdot \boldsymbol{x}_B \tag{44}$$

By using these relationships, the full state trajectory $\boldsymbol{x}(t)$ can be determined from the trajectory $\boldsymbol{\sigma}(t)$ given in the flat output space.

The differential flatness property of the quadrotor also lets us explicitly calculate the inputs $\boldsymbol{u}$. This is achieved by using the attitude dynamics of the quadrotor given in (30), together with the control input mapping in (35). For this, the angular accelerations $\boldsymbol{\alpha}_{\mathcal{BW}} = \dot{\boldsymbol{\omega}}_{\mathcal{BW}}$ are needed.

Differentiating (41) once more gives the slightly more complicated expression:

$$\begin{aligned} m\ddot{\boldsymbol{a}} = \ddot{F}_{th}\boldsymbol{z}_B + \alpha_{\mathcal{BC}} \times F_{th}\boldsymbol{z}_B \\ +2\boldsymbol{\omega}_{\mathcal{BW}} \times \dot{F}_{th}\boldsymbol{z}_B + \boldsymbol{\omega}_{\mathcal{BW}} \times \boldsymbol{\omega}_{\mathcal{BW}} \times F_{th}\boldsymbol{z}_B \end{aligned} \tag{45}$$

Proceeding as before, we take the dot product of the entire expression along $\boldsymbol{z}_B$, yielding an expression for $\ddot{F}_{th}$. Inserting this back into (45), and by dividing everything by F_{th}, one gets:

$$\begin{aligned} \boldsymbol{h}_\alpha :&= \boldsymbol{\alpha}_{\mathcal{BW}} \times \boldsymbol{z}_B \\ &= \frac{m}{F_{th}}(\ddot{\boldsymbol{a}} - (\boldsymbol{z}_B \cdot \ddot{\boldsymbol{a}})\boldsymbol{z}_B) \\ &- \boldsymbol{\omega}_{\mathcal{BW}} \times \boldsymbol{\omega}_{\mathcal{BW}} \times \boldsymbol{z}_B \\ &+ (\boldsymbol{z}_B \cdot (\boldsymbol{\omega}_{\mathcal{BW}} \times \boldsymbol{\omega}_{\mathcal{BW}} \times \boldsymbol{z}_B))\boldsymbol{z}_B \\ &- \frac{2}{F_{th}}\boldsymbol{\omega}_{\mathcal{BW}} \times (\boldsymbol{z}_B \cdot m\dot{\boldsymbol{a}})\boldsymbol{z}_B \end{aligned} \tag{46}$$

And similarly to before, the angular accelerations can be calculated as:

$$\dot{p} = -\boldsymbol{h}_\alpha \cdot \boldsymbol{y}_B \tag{47}$$

$$\dot{q} = \boldsymbol{h}_\alpha \cdot \boldsymbol{x}_B \tag{48}$$

From these angular accelerations, the corresponding input trajectory $\boldsymbol{u}(t)$ can be uniquely determined.

As we are only dealing with trajectories where $\psi = 0$, we get $r = 0, \dot{r} = 0$. However, for a non-zero yaw trajectory, the resulting angular velocity r and acceleration $\dot{r}$ can be calculated as done in [3].

IV. Control

The generated trajectory is to be followed by the quadrotor. To achieve this, a Time-Varying Linear Quadratic Regulator (TVLQR) stabilizing the generated trajectory is designed.

A. Linearizaton

First, the three-dimensional trajectory generated in the flat output space needs to be converted into a full twelve-dimensional nominal state trajectory $\boldsymbol{x}_0(t)$, as well as a four-dimensional nominal input trajectory $\boldsymbol{u}_0(t)$: this is achieved by using the relationships derived in III.

Second, define the deviation from the nominal trajectory as $\bar{\boldsymbol{x}}(t) = \boldsymbol{x}(t) - \boldsymbol{x}_0(t)$ and $\bar{\boldsymbol{u}}(t) = \boldsymbol{u}(t) - \boldsymbol{u}_0(t)$. Then the dynamics given in the general form (36) can be linearized around the nominal trajectory to achieve the following linear, time-varying system:

$$\dot{\bar{\boldsymbol{x}}} = \boldsymbol{A}(t)\bar{\boldsymbol{x}} + \boldsymbol{B}(t)\bar{\boldsymbol{u}} \tag{49}$$

where

$$\boldsymbol{A}(t) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\Big|_{\substack{\boldsymbol{x}=\boldsymbol{x}_0(t),\\ \boldsymbol{u}=\boldsymbol{u}_0(t)}}, \quad \boldsymbol{B}(t) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\Big|_{\substack{\boldsymbol{x}=\boldsymbol{x}_0(t)\\ \boldsymbol{u}=\boldsymbol{u}_0(t)}} \tag{50}$$

B. Finite Horizon LQR

The system in (49) can be stabilized using a finite-horizon LQR design. Consider a cost function of the form:

$$\min_{u(\cdot)} \int_0^{t_f} \bar{\boldsymbol{x}}^\top \boldsymbol{Q}\bar{\boldsymbol{x}} + \bar{\boldsymbol{u}}^\top \boldsymbol{R}\bar{\boldsymbol{u}}\, dt \tag{51}$$

where $\boldsymbol{Q} \succeq 0$, $\boldsymbol{R} \succ 0$. An optimal cost-to-go function J^* for this cost function takes the form $J^*(\bar{\boldsymbol{x}}, t) = \bar{\boldsymbol{x}}^\top \boldsymbol{S}(t)\bar{\boldsymbol{x}}$. It can be shown that the optimal controller satisfying the Hamilton-Jacobi-Bellman equation for this J^* is:

$$\bar{\boldsymbol{u}}^* = -\boldsymbol{R}^{-1}\boldsymbol{B}(t)^\top \boldsymbol{S}(t)\bar{\boldsymbol{x}} := -\boldsymbol{K}(t)\bar{\boldsymbol{x}} \tag{52}$$

where $S(t)$ satisfies the Differential Riccati Equation:

$$-\dot{\boldsymbol{S}} = \boldsymbol{S}\boldsymbol{A}(t) + \boldsymbol{A}(t)^\top \boldsymbol{S} - \boldsymbol{S}\boldsymbol{B}(t)\boldsymbol{R}^{-1}\boldsymbol{B}(t)^\top \boldsymbol{S} + \boldsymbol{Q} \tag{53}$$

as is done in the chapter on Linear Quadratic Regulators in [8]. The Differential Riccati equation is integrated backwards in time, starting from the desired final value of $\boldsymbol{S}$, as indicated by the sign on $\dot{\boldsymbol{S}}$. The choice of the final value $\boldsymbol{S}(t_f)$ is described in the next section.

From this, our complete TVLQR controller stabilizing the entire nominal trajectory given by $\boldsymbol{x}_0(t)$ and $\boldsymbol{u}_0(t)$ is given by:

$$\boldsymbol{u}(t) = -\boldsymbol{K}(t)\bar{\boldsymbol{x}} + \boldsymbol{u}_0(t) \tag{54}$$

where $\boldsymbol{K}(t)$ is computed as in (52) with $\boldsymbol{S}(t)$ satisfying the Differential Riccati equation in (55), and where $\boldsymbol{u}_0(t)$ essentially serves as a feed-forward of the nominal input trajectory.

C. Infinite Horizon LQR

Once the quadrotor reaches the final point on the calculated trajectory, it is desirable to stabilize the quadrotor around the final position. Therefore, at time $t = t_f$ the TVLQR devised in the previous section is replaced by an Infinite-Horizon LQR stabilizing the final position. This motivates the choice of $\boldsymbol{S}(t_f) = \boldsymbol{S}_{\text{inf}}$, where $\boldsymbol{S}_{\text{inf}}$ satisfies the Continuous Algebraic Riccati Equation

$$\boldsymbol{0} = \boldsymbol{S}\boldsymbol{A} + \boldsymbol{A}^\top \boldsymbol{S} - \boldsymbol{S}\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^\top \boldsymbol{S} + \boldsymbol{Q} \tag{55}$$

with $\boldsymbol{A} = \boldsymbol{A}(t_f)$ and $\boldsymbol{B} = \boldsymbol{B}(t_f)$.

V. Experimental Results

A. Implementation

The project is implemented in Drake [9] using C++ and the open-source library Eigen [10]. The trajectory optimization problem is implemented from scratch using Drake's wrapper for the commercial solver Mosek [11]. The TVLQR controller is also implemented from scratch, symbolically linearizing the equations of motion using Drake's symbolic library. The entire source-code can be found at github.com/bernhardpg/collision-free-mixed-integer-planning-for-uavs. A video of the results can be seen at [1].

The entire system is built and simulated in Drake, on a Macbook Pro 15" running a 2.8 GHz Quad-Core Intel Core i7. In general, it was found that choosing each trajectory to consist of around 5-8 polynomial segments and using 5-10 convex regions usually resulted in a computation time in the order of a few minutes.

B. Flight test 1 - Simple environment

First, the system is tested for a simple obstacle environment shown in Fig. 1. From the figure, it can be seen how the safe convex regions are generated iteratively by automatically choosing the next point furthest away from any obstacles or previously generated regions. Calculating the safe regions and the collision-free trajectory takes less than 30 seconds in this case. In Fig 2, the resulting calculated trajectory can be seen. It is clear that the trajectory stays clear of any obstacles for the entirety of the trajectory.

C. Flight test 2 - Challenging obstacle environment

In the second flight test, a much more challenging obstacle environment is constructed. The environment is constrained in such a way that the quadrotor has no other choice than to fly through a narrow opening very close to the ground. Still, the generated trajectory stays collision-free, and the quadrotor is able to follow it accurately. Generating the convex regions and the trajectory for this flight test took considerably longer than for the first flight task, and the time required for calculation was approximately 2 minutes. Images from the flight test can be seen in Fig. 3.

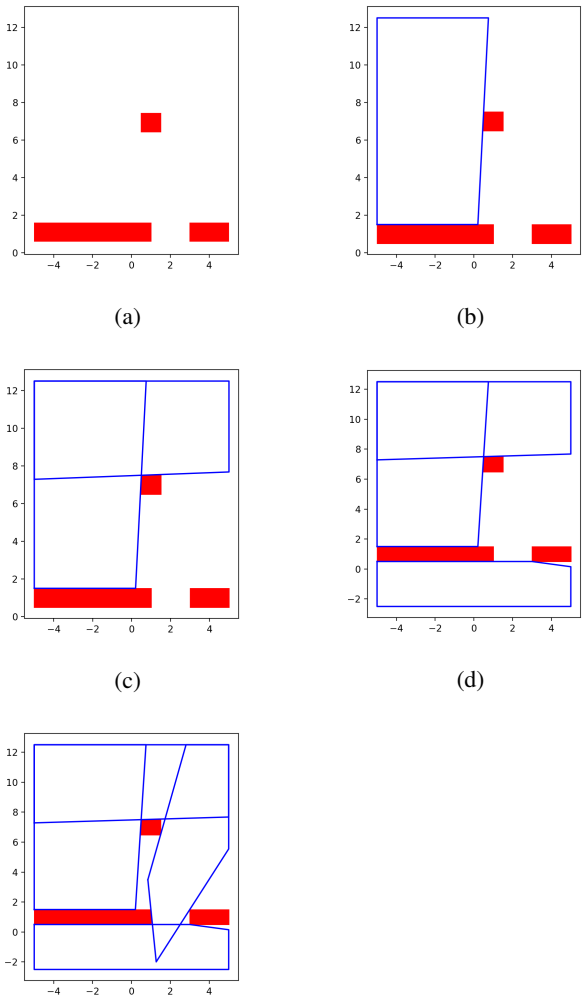
D. Flight test 3 - Varying the number of convex regions

In the last flight test, the effect of varying the number of convex regions is tested. As the number of convex regions increases, more and more of the original space will be covered by the safe regions, allowing the quadrotor to consider bigger and bigger portions of the space. However, increasing the number of convex regions also comes with a big increase in the required computation time.

First, the planner is allowed to calculate 8 convex regions, as shown in Fig. 4a. Then, the planner is allowed to add one more convex region, as shown in Fig. 4b. Notice that the region indicated by the green circle is not covered by any regions before the 9th convex region is added.

The resulting trajectories can be seen in Fig. 5. In the case where the planner is only working with 8 regions, it misses out on a better trajectory, as this path is not entirely covered by safe regions and thus is infeasible. However, when increasing the number of convex regions by one, the more optimal trajectory is discovered. Note that both solutions are globally optimal in terms of the cost function in (25), but differ from each other because they are subject to different region constraints.

This test highlights the importance of using enough convex regions to cover the essential parts of the environment, and



(a) (b) (c) (d) (e)

Fig. 1: Iteratively adding convex regions

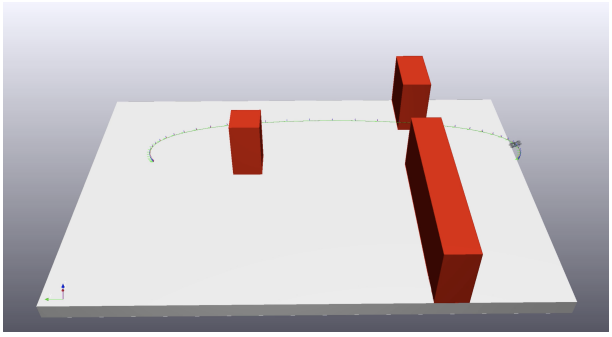
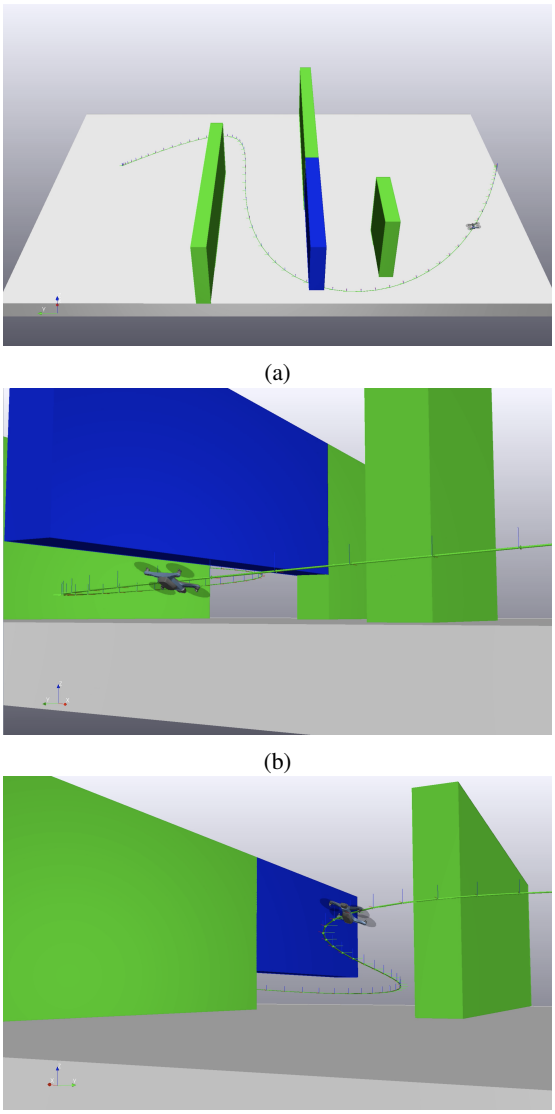


Fig. 2: The collision-free path found for the obstacle environment



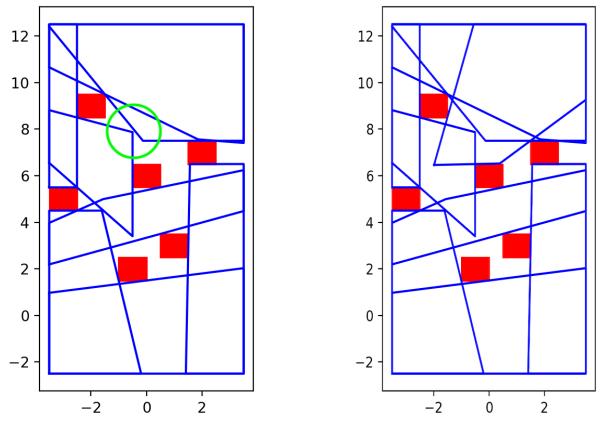
(a) (b) (c)

Fig. 3: Images of the quadrotor flying very close to the ground to stay clear of the obstacles

illustrates how much the choice of convex regions may impact the resulting trajectory.

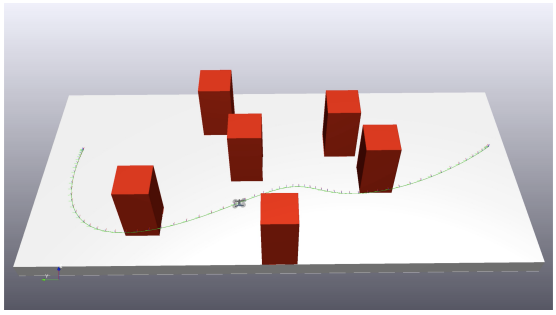
E. Conclusion

In this project, a complete approach to planning and stabilizing collision-free trajectories in obstacle-filled environments for the differentially flat quadrotor is constructed. Several experiments are performed in simulation, showing the strength of the method. The importance of the choice of convex regions has been illustrated, underlining the need for a sufficient number of convex regions in order to find a good trajectory. However, when increasing the number of convex regions or the number of polynomial segments for the trajectory, the required computation time increases quickly, making it unlikely to implement this design as-is for a real-time application.

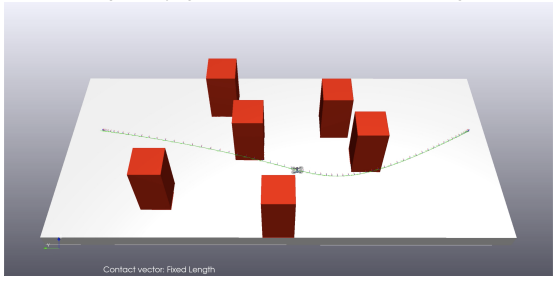


(a) 8 convex regions. (b) 9 convex regions.

Fig. 4: The effect of varying the number of convex regions can make a big difference for the resulting trajectory. Notice that the unfilled area marked by a green circle in (a) is covered in (b).



(a) Trajectory generated from 8 convex regions.



(b) Trajectory generated from 9 convex regions.

Fig. 5: Images showing the results from Flight Test 3, which illustrates the importance of the number of convex regions.

F. Future Work

In the future, it would be interesting explore alternative strategies for seeding IRIS when generating the convex regions. For example, one could utilize sample-based planning algorithms like RRT* to generate waypoints which could be used as seedpoints.

During this project, I did not have time to look into how time-scaling the trajectories would work. Combining this with an approach for limiting the actuator inputs, for example by iteratively making the trajectory more and more aggressive until the limits are reached as done in [12] is something I would really like to explore.

Finally, exploring different ways of avoiding the Mixed-Integer formulation, and thus removing a huge computational bottleneck, would be very interesting. For example by looking at utilizing the convex hull property of Bézier-curves as done in [13].j

References

[1] B. P. Graesdal, "Collision free mixed integer planning for quadrotors using convex safe regions." https://tinyurl.com/yawd6wv4, 2020.
[2] R. Deits and R. Tedrake, "Efficient mixed-integer planning for uavs in cluttered environments," vol. 2015, pp. 42–49, 06 2015.
[3] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," pp. 2520 – 2525, 06 2011.
[4] D. Mellinger, A. Kushleyev, and V. Kumar, "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams," in *2012 IEEE International Conference on Robotics and Automation*, pp. 477–483, 2012.
[5] R. Deits and R. Tedrake, "Computing large convex regions of obstacle-free space through semidefinite programming," vol. 107, pp. 109–124, 04 2015.
[6] P. Parrilo, "6.972 algebraic techniques and semidefinite optimization." https://ocw.mit.edu, Spring 2006.
[7] M. S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret, "Applications of second-order cone programming," *Linear Algebra and its Applications*, vol. 284, no. 1, pp. 193 – 228, 1998. International Linear Algebra Society (ILAS) Symposium on Fast Algorithms for Control, Signals and Image Processing.
[8] R. Tedrake, "Russ tedrake. underactuated robotics: Algorithms for walking, running, swimming, flying, and manipulation (course notes for mit 6.832),"
[9] R. Tedrake and the Drake Development Team, "Drake: Model-based design and verification for robotics," 2019.
[10] G. Guennebaud, B. Jacob, *et al.*, "Eigen v3." http://eigen.tuxfamily.org, 2010.
[11] "The mosek optimization software." http://www.mosek.com/, 2020.
[12] C. Richter and A. Bry, *Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments*, pp. 649–666. 04 2016.
[13] F. A. Koolen, "Balance control and locomotion planning for humanoid robots using nonlinear centroidal models," 2019.