Rapid Collision Detection for Multicopter Trajectories

Nathan Bucki and Mark W. Mueller

Abstract— **We present a continuous-time collision detection algorithm for quickly detecting whether certain polynomial trajectories in time intersect with convex obstacles. The algorithm is used in conjunction with an existing multicopter trajectory generation method to achieve rapid, obstacle-aware motion planning in environments with both static convex obstacles and dynamic convex obstacles whose boundaries do not rotate. In general, this problem is difficult because the presence of convex obstacles makes the feasible space of trajectories nonconvex. The performance of the algorithm is benchmarked using Monte Carlo simulations, and experimental results are presented that demonstrate the use of the method to plan collision-free multicopter trajectories in milliseconds in environments with both static and dynamic obstacles.**

I. Introduction

A key enabler of the use of autonomous systems in real-world situations is a fast method for generating state and input feasible trajectories between desired states. This problem is know as the motion planning problem, and is a well researched area that includes numerous methods for the generation of such trajectories. In particular, sampling-based methods such as rapidly exploring random trees (RRT) [1], probabilistic roadmaps (PRM) [2], and fast marching trees (FMT) [3], have been used with great success to construct collision-free paths between desired states. Such methods are typically performed by sampling the state space of the system and attempting to connect feasible sampled nodes with simple trajectories that do not collide with obstacles. Performing collision detection on both the sampled nodes and the trajectories that connect them is often considered the most computationally expensive step in the motion planning process, and will be the focus of this paper.

Previous work focusing on the reduction of collision detection time includes [4], which presents an algorithm that involves using distance information from previously sampled nodes to avoid performing explicit node-obstacle collision detection when possible. In [5] this idea is adapted to reduce collision detection time for multicopter trajectories by computing overlapping collision-free spheres around the trajectory based on the maximum velocity of the vehicle and distance to the nearest obstacle at each sample point.

Rather than generating a number of multicopter trajectories and then checking each one for collisions, the authors of [6] first compute a series of overlapping, obstacle-free polyhedrons and then generate a series of trajectory segments that remain inside the polyhedrons. In [7], the authors take a similar approach by using an octree-based representation of the environment in order to enforce corridor constraints on each trajectory segment generated. A third approach to collision avoidance for aggressive flight is explored in [8], where a dense set of alternative trajectories to some desired trajectory are precomputed, allowing for one of the alternative trajectories to be chosen if a collision is predicted along the desired trajectory. In this case, a collision is determined by comparing the distance of each point along the discretized candidate trajectory to the points in a point cloud generated by a laser scanner.

In contrast to methods concerned only with planning in static environments, the authors of [9] leverage sequential convex programming to compute trajectories for multiple quadcopters that do not collide, allowing for dynamic formation changes. In [10] a method for dynamic obstacle collision avoidance is presented that models the obstacles as ellipsoids and incorporates them as nonconvex constraints in a model predictive controller.

In this paper we are interested in reducing the computational time required to find feasible trajectories for multicopters in order to enable high-speed flight in cluttered, unknown environments (e.g. when navigating a forest). Fast trajectory generation is a requirement in these scenarios due to the limited range of onboard sensors and limited onboard computational power. For example, obstacles can often be occluded or unexpectedly change position, requiring an immediate, agile response to avoid a collision if flying at high speeds. Furthermore, due to the constrained onboard computational power of aerial vehicles, efficient algorithms are often required in order to achieve acceptable performance.

To this end, we propose a computationally efficient method for quickly evaluating whether a candidate trajectory collides with obstacles in the environment. We limit ourselves to evaluating multicopter trajectories similar to those described in [11], which describe the vehicle's position as a fifth order polynomial in time. These trajectories result in the minimum average jerk over the duration of the trajectory, and are particularly useful because they can be generated and checked for input feasibility with little computation. Unlike other collision detection methods that discretize the trajectory in time and perform a number of static collision checks at each sample point (e.g. as detailed in [12]), our method leverages a continuous-time representation of the trajectory to rapidly perform continuous-time collision detection.

II. System model

We follow [11] in modeling the multicopter as a six degree of freedom rigid body with acceleration $\ddot{\boldsymbol{x}} \in \mathbb{R}^3$ (written in

The authors are with the High Performance Robotics Lab, University of California, Berkeley, CA 94703, USA. {nathan_bucki, mwm}@berkeley.edu

the inertial coordinate frame) and orientation $\boldsymbol{R}$, where $\boldsymbol{R}$ represents the rotation matrix that rotates vectors in the body-fixed frame to the inertial frame. Gravity is denoted $\boldsymbol{g} \in \mathbb{R}^3$ (written in the inertial frame), and the mass-normalized total thrust force $f \in \mathbb{R}$ acts in the $\boldsymbol{e}_3$ direction, where $\boldsymbol{e}_3$ is the body-fixed thrust direction. We assume that the angular velocity of the vehicle $\boldsymbol{\omega}$ is controlled by a high-bandwidth low-level controller such that the angular velocity converges very rapidly to a desired value, and may thus be treated as an input to the system. The translational dynamics and attitude dynamics of the multicopter are then

$$\ddot{\boldsymbol{x}} = \boldsymbol{R}\boldsymbol{e}_3 f + \boldsymbol{g}, \qquad \dot{\boldsymbol{R}} = \boldsymbol{R}\boldsymbol{S}(\boldsymbol{\omega}) \tag{1}$$

where $\boldsymbol{S}(\boldsymbol{\omega})$ is the skew-symmetric form of the angular velocity vector $\boldsymbol{\omega}$ so that $\boldsymbol{S}(\boldsymbol{\omega})\,\boldsymbol{v} = \boldsymbol{\omega} \times \boldsymbol{v}$ for any vector $\boldsymbol{v}$.

Using this model, it can be shown that kinematically feasible polynomial trajectories in time can be generated using the differential flatness property of multicopter dynamics [13]. Specifically, we plan trajectories by defining the components of the jerk $\dddot{\boldsymbol{x}}(t)$ as second order polynomials in time between time $t = 0$ and $t = T$. As described in [11], this results in trajectories that minimize the average jerk over the trajectory. The thrust f and angular velocity $\boldsymbol{\omega}$ are then written as a function of $\ddot{\boldsymbol{x}}$ and $\dddot{\boldsymbol{x}}$ as follows.

$$f = ||\ddot{\boldsymbol{x}} - \boldsymbol{g}||_2, \qquad \begin{bmatrix}\omega_2\\ \omega_1\\ 0\end{bmatrix} = \frac{1}{f}\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 0\end{bmatrix}\boldsymbol{R}^{-1}\dddot{\boldsymbol{x}} \tag{2}$$

where ω_1 and ω_2 are the components of angular velocity perpendicular to the thrust direction (i.e. the roll and pitch rates). Note that the angular velocity in the $\boldsymbol{e}_3$ direction does not affect the translational motion of the vehicle, and is taken to be $\omega_3 = 0$ for the rest of the paper.

The position and velocity of the multicopter are defined as $\boldsymbol{x}$ and $\dot{\boldsymbol{x}}$ respectively, and are both in $\mathbb{R}^3$ and written in the inertial frame. Let $\boldsymbol{x}(0)$, $\dot{\boldsymbol{x}}(0)$, and $\ddot{\boldsymbol{x}}(0)$ be the position, velocity, and acceleration of the vehicle at the start of the trajectory. Because the minimum average jerk trajectory is achieved when each component of the jerk is a second order polynomial in time, the trajectories of the states of the system follow as

$$\begin{bmatrix}\boldsymbol{x}(t)\\ \dot{\boldsymbol{x}}(t)\\ \ddot{\boldsymbol{x}}(t)\end{bmatrix} = \begin{bmatrix}\frac{\boldsymbol{\alpha}}{120}t^5 + \frac{\boldsymbol{\beta}}{24}t^4 + \frac{\boldsymbol{\gamma}}{6}t^3 + \frac{\ddot{\boldsymbol{x}}(0)}{2}t^2 + \dot{\boldsymbol{x}}(0)t + \boldsymbol{x}(0)\\ \frac{\boldsymbol{\alpha}}{24}t^4 + \frac{\boldsymbol{\beta}}{6}t^3 + \frac{\boldsymbol{\gamma}}{2}t^2 + \ddot{\boldsymbol{x}}(0)t + \dot{\boldsymbol{x}}(0)\\ \frac{\boldsymbol{\alpha}}{6}t^3 + \frac{\boldsymbol{\beta}}{2}t^2 + \boldsymbol{\gamma}t + \ddot{\boldsymbol{x}}(0)\end{bmatrix} \tag{3}$$

where $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma} \in \mathbb{R}^3$ are linear functions of $\boldsymbol{x}(T)$, $\dot{\boldsymbol{x}}(T)$, and $\ddot{\boldsymbol{x}}(T)$.

A method for quickly checking whether a given trajectory satisfies bounds on the minimum and maximum thrust f and the magnitude of the angular velocity $\boldsymbol{\omega}$ is given in [11], to which we refer the reader for further discussion.

III. ALGORITHM FOR STATIC OBSTACLE COLLISION DETECTION

In this section we describe the collision detection algorithm. All obstacles are assumed to be convex; nonconvex obstacles may be approximated by defining them as a union of convex obstacles. In general, the presence of convex obstacles results in the feasible space being nonconvex, making the trajectory generation and collision detection problem difficult to perform using traditional optimization methods.

We first review a method used to check whether a polynomial trajectory lies on one side of a plane, which is defined by a point $\boldsymbol{p}$ and unit normal $\boldsymbol{n}$ (both written in the inertial frame). The distance of the trajectory from the plane can be computed as

$$d(t) = \boldsymbol{n}^T(\boldsymbol{x}(t) - \boldsymbol{p}) \tag{4}$$

Furthermore, the critical points of $d(t)$ can be computed by differentiating with respect to t and finding the roots of the resulting equation:

$$\dot{d}(t) = \boldsymbol{n}^T\dot{\boldsymbol{x}}(t) = c_4t^4 + c_3t^3 + c_2t^2 + c_1t + c_0 \tag{5}$$

where

$$\begin{gathered} c_4 = \tfrac{1}{24}\boldsymbol{n}^T\boldsymbol{\alpha}, \qquad c_3 = \tfrac{1}{6}\boldsymbol{n}^T\boldsymbol{\beta}, \qquad c_2 = \tfrac{1}{2}\boldsymbol{n}^T\boldsymbol{\gamma}\\ c_1 = \boldsymbol{n}^T\ddot{\boldsymbol{x}}(0), \qquad c_0 = \boldsymbol{n}^T\dot{\boldsymbol{x}}(0)\end{gathered} \tag{6}$$

The trajectory $\boldsymbol{x}(t)$ is defined only between $t = 0$ and $t = T$, so the critical points of $d(t)$ occur between and include the start and end of the candidate trajectory. The set of critical points $\mathcal{T}_{\text{crit}}$ is then defined as

$$\mathcal{T}_{\text{crit}} = \{t_i : t_i \in [0, T],\ \dot{d}(t_i) = 0\} \cup \{0, T\} \tag{7}$$

If the distance $d(t)$ at each critical point is positive, this indicates that $\boldsymbol{x}(t)$ does not cross the plane. Because $\dot{d}(t)$ is a fourth order polynomial in time, its roots can be found in closed form, meaning that $\mathcal{T}_{\text{crit}}$ can be found with very little computation.

We now extend this method to detect collisions with convex obstacles. The given convex obstacle $\mathcal{O}$ and polynomial trajectory $\boldsymbol{x}(t)$ are required to have the following two properties. First, it must be possible to check whether a specific point $\boldsymbol{x}(t_0)$ is inside $\mathcal{O}$. Second, assuming $\boldsymbol{x}(t_0) \notin \mathcal{O}$, it must be possible to define a separating plane between $\boldsymbol{x}(t_0)$ and $\mathcal{O}$ (defined with point $\boldsymbol{p}$ unit normal $\boldsymbol{n}$). Thus, if $\boldsymbol{x}(t)$ is found to not cross the separating plane, it is guaranteed to not collide with $\mathcal{O}$.

Algorithm 1 leverages this property to verify whether an individual segment of a given trajectory is in collision with a given obstacle. The algorithm begins by checking whether the end points of the trajectory $\boldsymbol{x}(0)$ and $\boldsymbol{x}(T)$ are inside the obstacle (lines 4-5), followed by a call to CHECKSECTION, which returns whether the given section is feasible, infeasible, or whether the feasibility of the section cannot be determined (line 6). For each call to CHECKSECTION(t_s, t_f), a time t_{split} between t_s and t_f is chosen which divides the trajectory in two. We choose t_{split} to be the average of t_s and t_f, as it will evenly divide the trajectory into two sub-trajectories in time (line 8).

For each section checked recursively by CHECKSECTION, $\boldsymbol{x}(t_{\text{split}})$ is first checked for feasibility (line 9), and then the minimum resolution of the section $t_{\min}$ is checked (line 11). The parameter $t_{\min}$ serves to terminate the algorithm early in order to prevent excessive time being spent checking any particular candidate trajectory, and limits the recursive depth of the algorithm. This end condition can be reached in the case where the candidate trajectory passes sufficiently close to the obstacle, requiring the trajectory to be split into a large number of sub-trajectories to be checked.

Next, the unit normal $\boldsymbol{n}$ and location $\boldsymbol{p}$ of a separating plane are found (line 13). Although there are many possible ways to find a separating plane, in our implementation we choose $\boldsymbol{p}$ such that it is the minimum distance point to $\boldsymbol{x}(t_{\text{split}})$ located in $\mathcal{O}$. The unit normal of the plane $\boldsymbol{n}$ is then chosen such that the resulting plane lies on the obstacle boundary at $\boldsymbol{p}$ and points from $\boldsymbol{p}$ to $\boldsymbol{x}(t_{\text{split}})$. The times $\mathcal{T}_{\text{crit}}$ at which the critical points of the distance of the trajectory from the resulting separating plane occur are then computed by solving the corresponding fourth order polynomial given by (5) (line 15). Once $\mathcal{T}_{\text{crit}}$ is computed, the two sections of the trajectory occurring before and after t_{split} are each checked for feasibility. Let $\mathcal{T}_{\text{crit}}^{(\downarrow)}$ be the elements of $\mathcal{T}_{\text{crit}}$ in (t_s, t_{split}) and $\mathcal{T}_{\text{crit}}^{(\uparrow)}$ be the elements of $\mathcal{T}_{\text{crit}}$ in (t_{split}, t_f).

The section of the trajectory between t_{split} and t_f is first checked for feasibility by iterating forward in time over $\mathcal{T}_{\text{crit}}^{(\uparrow)}$ and checking whether each critical point of $d(t)$ lies on the feasible side of the separating plane (lines 17-18). If a critical point is found to lie on the obstacle side of the plane, the section between the previous critical point (already determined to be on the feasible side of the plane) and t_f cannot be guaranteed to be feasible and is recursively checked with CheckSection (line 19). Finally, the section of the trajectory between t_s and t_{split} is checked for feasibility in a similar manner by iterating backwards in time over $\mathcal{T}_{\text{crit}}^{(\downarrow)}$ (lines 26-28). A graphical representation of Algorithm 1 is shown in Figure 1.

Note that Algorithm 1 treats $\boldsymbol{x}(t)$ as the trajectory of a point. In order to detect collisions between $\mathcal{O}$ and a real multicopter, we define a sphere of radius r_q that contains the vehicle, and enlarge $\mathcal{O}$ by r_q in each direction. Additionally, because polynomials of order greater than four do not have closed form solutions except in special cases, greater computation time would be required to find the critical points of any higher order position trajectories (e.g. as used in [14]).

IV. Performance Measures

In this section we provide two simulations used to benchmark the performance of the proposed algorithm.[1] The algorithm was implemented in C++ and compiled with GCC version 5.4.0 with the highest speed optimization settings. All simulations were ran as a single thread on a laptop with a 1.80GHz Intel i7-8550U processor.

A. Monte Carlo simulation with random obstacles

First, a Monte Carlo simulation was conducted in order to characterize the computational time required to perform collision detection on a single candidate trajectory. The methods of [11] are used to generate the candidate trajectories and

[1]An implementation can be found at `https://github.com/nlbucki/RapidQuadcopterCollisionDetection`

Algorithm 1 Trajectory Collision Detection

```
 1: input: Candidate trajectory parameters $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$, initial
    conditions $\boldsymbol{x}(0)$, $\dot{\boldsymbol{x}}(0)$ $\ddot{\boldsymbol{x}}(0)$, minimum checking time
    $t_{min}$, convex obstacle $\mathcal{O}$
 2: output: feasible, infeasible, or indeterminable
 3: function CollisionCheck
 4:     if $\boldsymbol{x}(0)$ or $\boldsymbol{x}(T)$ inside obstacle then
 5:         return infeasible
 6:     return CheckSection(0, $T$)
 7: function CheckSection($t_s$, $t_f$)
 8:     $t_{\text{split}} \leftarrow \frac{t_s+t_f}{2}$
 9:     if $\boldsymbol{x}(t_{\text{split}})$ inside obstacle then
10:         return infeasible
11:     else if $t_f - t_s < t_{\text{min}}$ then
12:         return indeterminable
13:     Find plane separating $\boldsymbol{x}(t_{\text{split}})$ and obstacle
14:     $d(t) \leftarrow$ distance of $\boldsymbol{x}(t)$ from separating plane
15:     $\mathcal{T}_{\text{crit}}^{(\uparrow)} \leftarrow$ critical points of $d(t)$ from $t_{\text{split}}$ to $t_f$
16:     Sort $\mathcal{T}_{\text{crit}}^{(\uparrow)}$ ascending
17:     for $t_i$ in $\mathcal{T}_{\text{crit}}^{(\uparrow)}$, skipping $t_{\text{split}}$ do
18:         if $\boldsymbol{x}(t_i)$ is on obstacle side of plane then
19:             result $\leftarrow$ CheckSection($t_{i-1}$, $t_f$)
20:             if result is feasible then
21:                 break
22:             else
23:                 return result
24:     $\mathcal{T}_{\text{crit}}^{(\downarrow)} \leftarrow$ critical points of $d(t)$ from $t_{\text{split}}$ to $t_s$
25:     Sort $\mathcal{T}_{\text{crit}}^{(\downarrow)}$ descending
26:     for $t_i$ in $\mathcal{T}_{\text{crit}}^{(\downarrow)}$, skipping $t_{\text{split}}$ do
27:         if $\boldsymbol{x}(t_i)$ is on obstacle side of plane then
28:             return CheckSection($t_s$, $t_{i-1}$)
29:     return feasible
```

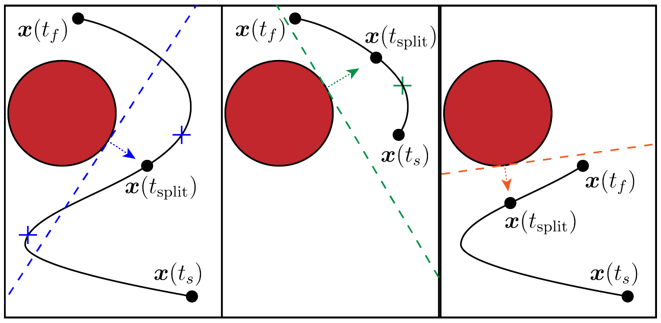

Fig. 1. A graphical depiction of Algorithm 1. Three sequential calls to CheckSection (as defined in Algorithm 1) are shown. The red circle represents the convex obstacle and the solid black line represents the trajectory in time. In the first call to CheckSection (shown in the left panel), two critical points (drawn as crosses) of the distance to the separating plane (drawn as a dashed line) are found. The trajectory is found to cross the separating plane between t_f and the critical points occurring after t_{split}, prompting a recursive call to CheckSection. As shown in the middle panel, this sub-trajectory is found to not collide with the obstacle because it lies entirely on the opposite side of the newly computed separating plane. Next, the original trajectory (left) is again found to cross the separating plane between t_s and t_{split}, leading to a second recursive call to CheckSection. As shown in the right panel, this sub-trajectory is also found to lie entirely on the opposite side of the newly computed separating plane, proving that the entire trajectory does not collide with the obstacle.

TABLE I

AVERAGE COLLISION DETECTION TIME.

	Feasible	Infeasible	Indeterminable
Fraction of trajectories	95.99%	4.01%	$< 0.01\%$
Collision detection time	$1.44\,\mu\mathrm{s}$	$1.36\,\mu\mathrm{s}$	$11.59\,\mu\mathrm{s}$

check whether the generated trajectory satisfies some given input bounds. Any trajectory requiring a mass-normalized thrust that is not between $5\,\mathrm{m\,s^{-2}}$ and $30\,\mathrm{m\,s^{-2}}$ or that requires an angular velocity of greater than $20\,\mathrm{rad\,s^{-1}}$ is discarded.

Candidate trajectories are generated with a fixed initial position of $\boldsymbol{x}(0) = (0, 0, 0)$. The final position, initial and final velocity, and initial and final acceleration along each axis are generated from uniform distributions over the intervals ($-4\,\mathrm{m}$, $4\,\mathrm{m}$), ($-4\,\mathrm{m\,s^{-1}}$, $4\,\mathrm{m\,s^{-1}}$), and ($-4\,\mathrm{m\,s^{-2}}$, $4\,\mathrm{m\,s^{-2}}$) respectively. The length of time of the trajectory is sampled uniformly at random between $0.2\,\mathrm{s}$ and $4\,\mathrm{s}$. A sphere with radius sampled uniformly at random on ($0.1\,\mathrm{m}$, $1.5\,\mathrm{m}$) and positions sampled uniformly at random on ($-4\,\mathrm{m}$, $4\,\mathrm{m}$) along each axis is used as an obstacle. The minimum collision detection time per section $t_{\min}$ is chosen to be $2\,\mathrm{ms}$.

For 10^9 such trials, the average time required to detect collisions was $1.44\,\mu\mathrm{s}$, and of the candidate trajectories, 95.99% did not collide with the obstacle. Table I shows the computation time required depending on whether the trajectory was found to be feasible, infeasible, or of indeterminable feasibility.

B. Monte Carlo simulation with constant obstacles

A second Monte Carlo simulation involving generating collision free trajectories that bring the vehicle to rest was also conducted. This scenario is of interest in the case where, for example, the vehicle must perform an emergency stopping maneuver (e.g. when an unexpected obstacle appears in the path of the vehicle while flying at high speed). The simulation is run in batches of 100 candidate trajectories, where each candidate trajectory starts from the same initial state and ends at rest at a position sampled uniformly at random along each axis on ($-2.5\,\mathrm{m}$, $2.5\,\mathrm{m}$). For each batch, the initial position of the vehicle is constrained to be ($-2.5\,\mathrm{m}$, $0\,\mathrm{m}$, $0\,\mathrm{m}$), the initial velocity and acceleration in the x-direction are sampled uniformly at random on ($2\,\mathrm{m\,s^{-1}}$, $8\,\mathrm{m\,s^{-1}}$) and $4\,\mathrm{m\,s^{-2}}$, $10\,\mathrm{m\,s^{-2}}$) respectively, and the initial velocity and acceleration in the y- and z-directions are sampled uniformly at random on ($-2\,\mathrm{m\,s^{-1}}$, $2\,\mathrm{m\,s^{-1}}$) and ($-2\,\mathrm{m\,s^{-2}}$, $2\,\mathrm{m\,s^{-2}}$) respectively. The length of time of the candidate trajectories is sampled uniformly at random between $0.5\,\mathrm{s}$ and $2\,\mathrm{s}$. The positions and orientations of the obstacles, represented as five long rectangular prisms, are fixed as shown in Figure 2, which additionally shows the candidate trajectories of a single batch.

One million batches were simulated. The average time required to find the first collision free trajectory was $14.6\,\mu\mathrm{s}$. On average, each trajectory required $0.1\,\mu\mathrm{s}$ to generate, $0.5\,\mu\mathrm{s}$ to check for satisfaction of constraints on the total thrust and angular velocity, and $7.7\,\mu\mathrm{s}$ to detect any collisions with the five obstacles. For each batch, an average of 60.2% of generated trajectories were collision free.

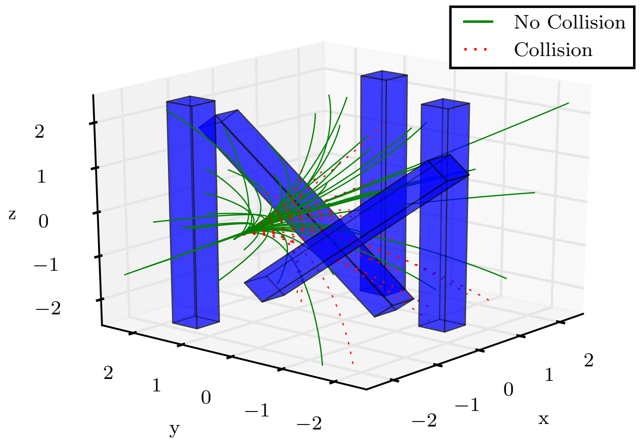

Fig. 2. Visualization of obstacle distribution and stopping trajectories. Obstacles are represented by blue rectangular prisms. Solid green lines represent the collision free trajectories from a single batch of 100 trajectories, and dotted red lines represent the trajectories that would collide with an obstacle. On average, the first feasible stopping trajectory was found in $14.6\,\mu\mathrm{s}$.

V. DYNAMIC OBSTACLE COLLISION DETECTION

In the previous section we showed that our algorithm is capable of detecting collisions between given quadcopter trajectories and static convex obstacles in an environment. This method is easily extended to detect collisions with dynamic obstacles whose boundaries do not rotate, and whose position trajectories are described by fifth order or below polynomial in time. Example applications of this method include detecting collisions between two quadcopters with different trajectories or between a projectile and a moving quadcopter.

Let $\boldsymbol{x}_{\mathcal{O}}(t)$ be the predicted trajectory of the center of a given obstacle. The relative position of the obstacle and the quadcopter at any time t is then

$$\tilde{\boldsymbol{x}}(t) = \boldsymbol{x}(t) - \boldsymbol{x}_{\mathcal{O}}(t) \tag{8}$$

where each component of $\tilde{\boldsymbol{x}}(t)$ will a polynomial in time if each component of both $\boldsymbol{x}(t)$ and $\boldsymbol{x}_{\mathcal{O}}(t)$ are also polynomials in time.

Recall that we model the quadcopter as a sphere with radius r_q. A collision between the quadcopter and the dynamic obstacle occurs if $\tilde{\boldsymbol{x}}(t)$ intersects with an obstacle centered at the origin of the same size as the dynamic obstacle but enlarged by r_q in each direction. Because the boundary of the obstacle is required to not rotate, the same methods described in Section III may be used to detect collisions between $\tilde{\boldsymbol{x}}(t)$ and the enlarged obstacle centered at the origin. Obstacles with boundaries that do rotate may be straight-forwardly encoded by enclosing them convex shapes with boundaries that do not rotate (e.g. spheres) at the penalty of introducing conservatism to the collision detection.

VI. Experimental results

This section presents experimental results where the proposed algorithm is used to enable a quadcopter to avoid unexpected static and dynamic obstacles. For the static obstacle case, we interrupt the motion of the quadcopter while following a trajectory by placing a surface in the path of the vehicle, forcing it to rapidly plan a new trajectory to avoid the collision and bring the vehicle to rest. For the dynamic obstacle case, we throw a projectile at the vehicle while it is following a trajectory to a goal position, again forcing it to rapidly plan a new trajectory to avoid the projectile and then continue to the original goal position. All experiments can be viewed in the attached video.

The quadcopter has a mass of $685\,\mathrm{g}$, and receives thrust and angular velocity commands at $50\,\mathrm{Hz}$ via radio from an offboard laptop. Collision detection and trajectory generation is performed using the same laptop as used for benchmarking in Section IV. The position and attitude of the quadcopter and obstacles are measured using a motion capture system at $200\,\mathrm{Hz}$.

During each controller time step, we check whether any collisions are predicted to occur between the quadcopter and the obstacle used for each experiment. If a collision is detected, a new trajectory is generated that is not predicted to collide with the obstacle and ends at rest with zero velocity and zero acceleration. We sample candidate end positions for the avoidance trajectory uniformly at random in a $3.2\,\mathrm{m} \times 5.2\,\mathrm{m} \times 1\,\mathrm{m}$ rectangular space and sample durations of the avoidance trajectory uniformly at random from $0.5\,\mathrm{s}$ to $2\,\mathrm{s}$. While tracking the avoidance trajectory, we continue to check for predicted collisions and generate a new avoidance trajectory if necessary.

When searching for feasible trajectories during both experiments, we generate and evaluate candidate trajectories for $15\,\mathrm{ms}$, and at the end of the allocated time choose the trajectory with the minimum average jerk that satisfies all state and input constraints. We choose the trajectory with the minimum average jerk in order to favor less aggressive trajectories. When evaluating each candidate trajectory, we first compute the average jerk of the trajectory and reject it if it has a higher average jerk than any previously found state and input feasible trajectory. Next, we use the methods of [11] to check that the total mass-normalized thrust f remains between $5\,\mathrm{m\,s^{-2}}$ and $30\,\mathrm{m\,s^{-2}}$ and that the maximum angular velocity remains bellow $20\,\mathrm{rad\,s^{-1}}$, as these are the physical limits of the experimental vehicle. We then check that the candidate trajectory stays within a box of $3.4\,\mathrm{m} \times 5.4\,\mathrm{m} \times 3.1\,\mathrm{m}$ to prevent the vehicle from flying into the ceiling, floor, or walls. Finally, we check that the candidate trajectory does not collide with any obstacles using Algorithm 1 with $t_{\min} = 0.002\,\mathrm{s}$.

A. Static obstacle avoidance

For the static obstacle avoidance experiment, we define the static obstacle as a rectangular prism measuring $1.64\,\mathrm{m} \times 1.43\,\mathrm{m} \times 0.78\,\mathrm{m}$, which includes both the increase in size in each direction necessary to account for the true size of the quadcopter and a small buffer to account for trajectory tracking and estimation errors. The quadcopter tracks two predefined trajectories that result in a roughly circular motion with an average speed of $5\,\mathrm{m\,s^{-1}}$, and checks these trajectories for collisions with the rectangular prism at each controller time step. The obstacle is then moved by hand in front of the vehicle about $0.5\,\mathrm{s}$ before the vehicle would pass, causing a collision to be predicted and an avoidance trajectory to be generated that brings the vehicle to rest without colliding with the obstacle. Figure 3 shows a sequence of images from the experiment. For the experiment shown in Figure 3, 14,610 candidate avoidance trajectories were evaluated in the allocated $15\,\mathrm{ms}$ after first predicting a collision with the obstacle (recall that the controller runs with a $20\,\mathrm{ms}$ period).

B. Dynamic obstacle avoidance

For the dynamic obstacle avoidance experiment, we throw a projectile at the vehicle while it is performing a rest to rest maneuver from some initial position to some final position $\boldsymbol{p}_f$. If a collision between the quadcopter and the projectile is predicted, the quadcopter rapidly plans a trajectory to avoid the projectile. The projectile is thrown by hand, meaning that its trajectory can only be predicted by the system after it has been thrown. The position $\boldsymbol{x}_p(0)$ and velocity $\dot{\boldsymbol{x}}_p(0)$ of the projectile at the current controller time step are estimated using position measurements received from the motion capture system and a Kalman filter. The position of the projectile $\boldsymbol{x}_p(t)$ is then predicted over a five second time horizon as a quadratic function of time that depends on $\boldsymbol{x}_p(0)$ and $\dot{\boldsymbol{x}}_p(0)$:

$$\boldsymbol{x}_p(t) = \boldsymbol{x}_p(0) + \dot{\boldsymbol{x}}_p(0)t + \frac{1}{2}\boldsymbol{g}t^2 \tag{9}$$

The minimum allowable distance between the center of mass of the projectile and the center of the quadcopter is chosen to be $40\,\mathrm{cm}$, which is chosen such that there is at least $10\,\mathrm{cm}$ separation between the quadcopter and projectile to account for any trajectory tracking and estimation errors. During each controller time step, we check whether the projectile is predicted to collide with the quadcopter by checking whether their relative position $\tilde{\boldsymbol{x}}(t)$ ever enters a sphere of radius $40\,\mathrm{cm}$ centered at the origin, and begin generating an avoidance trajectories if a collision is detected. While evaluating candidate avoidance trajectories, we not only check that the avoidance trajectory will not collide with the projectile during the maneuver, but also check that the projectile will not collide with the quadcopter after the quadcopter has reached the end position of the avoidance trajectory.

While tracking the avoidance trajectory, we try to generate sample return trajectories at each controller time step that

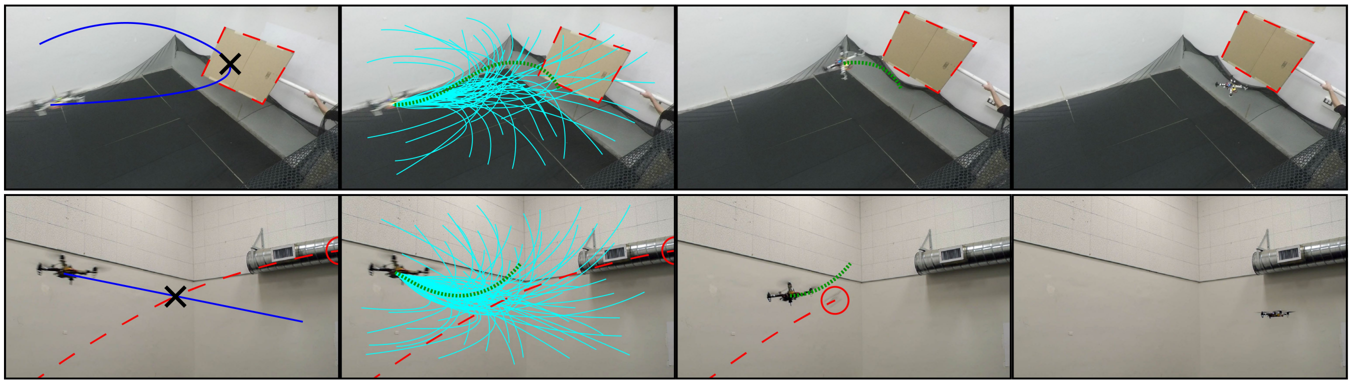

Fig. 3. A quadcopter avoiding an unexpected surface (top) and a thrown projectile (bottom). The images are from the attached video, and move forward in time from left to right. The original desired trajectory of the vehicle is shown with a solid blue line, and the position of the surface and predicted trajectory of the projectile are both shown by a dashed red line. In the first frame a collision is predicted to occur if the quadcopter remains on its current trajectory. In the second frame, a large number of alternative trajectories are generated (shown as solid cyan lines). Alternative trajectories that do not satisfy state and input constraints are discarded, and the minimum average jerk trajectory that satisfies all constraints is chosen (shown as a dotted green line). In the experiments shown, 14,610 and 2,371 candidate trajectories were generated and evaluated in 15 ms to avoid the surface and projectile respectively. In the third frame, the avoidance trajectory is tracked while continuing to detect predicted collisions with the obstacle and replanned if necessary. Finally, the fourth frame shows the vehicle successfully coming to rest without colliding with the surface (top), and generating and tracking a trajectory that brings the vehicle to the original desired end position (bottom).

bring the quadcopter from its current state to rest at the originally desired end position $\boldsymbol{p}_f$. Durations of the candidate return trajectories are sampled between $0.5\,\mathrm{s}$ and $4\,\mathrm{s}$. Once a feasible trajectory is found that does not collide with the projectile and ends at $\boldsymbol{p}_f$, the avoidance trajectory is interrupted and the vehicle begins tracking the return trajectory. Figure 3 shows a sequence of images from the experiment. For the experiment shown in Figure 3, 2,371 candidate avoidance trajectories were evaluated in the allocated $15\,\mathrm{ms}$ after first predicting a collision with the projectile.

VII. Conclusion

In this paper we presented a method for quickly detecting whether a polynomial trajectory collides with a convex obstacle. This method can be applied to both static convex obstacles and dynamic obstacles whose boundaries do not rotate. We used the proposed algorithm to perform rapid collision detection of multicopter trajectories, which can be modeled by fifth order polynomials in time. The ability to rapidly assess whether a given trajectory will collide with obstacles allows for a collision-free trajectory to be found in a short period of time by generating and checking many candidate trajectories for collisions. Because such a large number of the candidate trajectories can be generated and evaluated in such a short period of time, the vehicle is able to plan collision-free trajectories within milliseconds. This enables the vehicle to avoid obstacles that suddenly appear while the vehicle is flying at high speeds and to avoid projectiles thrown at high speeds.

Acknowledgement

This material is based upon work supported by the Berkeley Fellowship for Graduate Study. The experimental testbed at the HiPeRLab is the result of contributions of many people, a full list of which can be found at `hiperlab.berkeley.edu/members/`.

References

[1] S. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” 1998.

[2] L. Kavraki, P. Svestka, J. Latombe, and M. H. Overmars, *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*. International Transactions on Robotics and Automation, 1994, vol. 12.

[3] L. Janson, E. Schmerling, A. Clark, and M. Pavone, “Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions,” *The International Journal of Robotics Research*, vol. 34, no. 7, pp. 883–921, 2015.

[4] J. Bialkowski, M. Otte, S. Karaman, and E. Frazzoli, “Efficient collision checking in sampling-based motion planning via safety certificates,” *The International Journal of Robotics Research*, vol. 35, no. 7, pp. 767–796, 2016.

[5] B. T. Lopez and J. P. How, “Aggressive 3-D collision avoidance for high-speed navigation,” in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 5759–5765.

[6] S. Liu, M. Watterson, S. Tang, and V. Kumar, “High speed navigation for quadrotors with limited onboard sensing,” in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1484–1491.

[7] J. Chen, T. Liu, and S. Shen, “Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments,” in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1476–1483.

[8] J. Zhang, R. G. Chadha, V. Velivela, and S. Singh, “P-cap: Pre-computed alternative paths to enable aggressive aerial maneuvers in cluttered environments,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 8456–8463.

[9] F. Augugliaro, A. P. Schoellig, and R. D’Andrea, “Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2012, pp. 1917–1922.

[10] T. Nägeli, J. Alonso-Mora, A. Domahidi, D. Rus, and O. Hilliges, “Real-time motion planning for aerial videography with dynamic obstacle avoidance and viewpoint optimization,” *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1696–1703, 2017.

[11] M. W. Mueller, M. Hehn, and R. D’Andrea, “A computationally efficient motion primitive for quadrocopter trajectory generation,” *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1294–1310, 2015.

[12] S. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[13] D. Mellinger and V. Kumar, “Minimum snap trajectory generation and control for quadrotors,” in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2011, pp. 2520–2525.

[14] C. Richter, A. Bry, and N. Roy, “Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments,” in *Robotics Research*. Springer, 2016, pp. 649–666.