Integrated Perception and Control at High Speed: Evaluating Collision Avoidance Maneuvers Without Maps

Pete Florence[1], John Carter[1], and Russ Tedrake[1]

MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA
{peteflo,jcarter,russt}@csail.mit.edu

Abstract. We present a method for robust high-speed quadrotor flight through unknown cluttered environments using a fast approximation of collision probabilities. Motivated by experiments in which the difficulty of accurate state estimation was a primary limitation on speed, our method forgoes maintaining a map in favor of using only instantaneous depth information in the local frame. This provides robustness in the presence of significant state estimate uncertainty. Additionally, we present approximation methods augmented with spatial partitioning data structures that enable low-latency, real-time reactive control. The probabilistic formulation provides a natural way to integrate reactive obstacle avoidance with arbitrary navigation objectives. We validate the method using a simulated quadrotor race through a forest at high speeds in the presence of increasing state estimate noise. We pair our method with an MPC-based motion primitive library and compare with a global path-generation and path-following approach.

1 Related Work

A closely related line of work is the work in "funnel" computation and sequential composition approach that has come out of our research lab. The method we present here may be considered a variant that treats unbounded probability distributions rather than bounded uncertainty. An advantage of the funnel approach is that it rigorously handles non-linear systems, whereas here we present only a full formulation for a linear model with Gaussian noise.

Matthies et al. considers low-latency planning and collision checking in the depth image with considerations of occlusions, and so is closely aligned with our goals of low-latency obstacle avoidance with a focus on robustness.

As our method is memoryless in that it does not consider any history of previous states or measurements, it may be considered in part a reactive control method. To date, some of the most impressive obstacle avoidance results have been achieved with reactive methods. There are four primary categories of approaches that may be considered reactive and have been implemented with success on UAV platforms for obstacle avoidance: (i) optic flow methods, (ii) artificial potential fields, (iii) imitation learning, and (iv) deterministic collision

checking of offline-computed libraries. In some cases, some robustness considerations have been explored with these approaches, although not as the primary item of study, as they are for this paper.

Another closely related area of work is that of chance constrained optimization. In the chance-constrained literature, the probability of collision at any time is upper-bounded as a constraint in an optimization, but the total probability of collision for a time-varying distribution of configurations is not evaluated over a discrete library of maneuvers.

A noteworthy and complementary approach to the one presented here aims to learn collision probabilities outside of the conservative field of view approximations we make here (1).

2 Contributions

Our contributions are as follows:

- Efficient evaluation of the distribution of future robot configurations in the local frame which allows for fast online computation of collision probabilities
- Probabilistic formulation which enables seamless integration of "reactive" control with arbitrary navigation objectives
- Implementation of the above for robust, fast obstacle avoidance for a quadrotor UAV
- Simulation experiments demonstrating the efficacy of a probabilistic mapless approach compared to a deterministic mapless approach, and global map-based planning

3 Generalized Formulation for Collision Avoidance

First, we consider the problem of estimating the probability of collision over a time-varying distribution of configurations using only instantaneous depth information. We then present approximation methods that enable fast computation for collision avoidance at high speeds. Additionally, we discuss the use of spatial partitioning data structures and the incorporation of global navigation objectives. This section is generalized to allow for application to an arbitrary robot. In the next section, a particular implementation for a quadrotor is presented.

3.1 Evaluating Collision Probabilities from Instantaneous Depth Information

Consider the problem of evaluating the probability of collision of a time-varying distribution of configuration using only local instantaneous depth information. We wish to evaluate

$$P\big(\text{Collision during } t \in [0, t_f] \mid \mathbf{D}, p_t(\mathbf{q})\big) \tag{1}$$

where $p_t(\mathbf{q})$ is the time-varying distribution of configuration, t_f is the final time, and $\mathbf{D}$ is a vector of depth sensor returns $[\mathbf{d}_0, ..., \mathbf{d}_n]$. This probability cannot be calculated with certainty, due to the large amount of unknown space $\mathcal{U} \subset \mathbb{R}^3$ caused by occlusions and the finite FOV (field of view) of the depth sensor.

Each depth return corresponds to an occupied frustum $\mathcal{F}_{\mathbf{d}_j} \subset \mathbb{R}^3$ whose volume is defined by the image resolution and depth return distance, which together compose the known occupied subset of space, $\mathcal{O} = \bigcup_j \mathcal{F}_{\mathbf{d}_j}, \mathcal{O} \subset \mathbb{R}^3$. The conservative route is to make the assumption that all unknown space $\mathcal{U}$ is occupied, which provides a mapping from $\mathbf{D} \to \{\mathcal{U}, \mathcal{O}\}$ that is strictly conservative. With this assumption, each depth return also creates a portion of unknown space $\mathcal{F}_{(\text{occluded by } \mathbf{d}_j)} \subset \mathcal{U}$ corresponding to an "unbounded frustum" occluded by that depth return.

Fig. 1: Depictions of (a) depth measurements (black) and conservative assumption of occupied space (blue) and (b) time-varying distribution of configuration (purple).

At any given point in time and given the distribution $p(\cdot)$ over robot configuration $\mathbf{q}$, the probability of collision is obtained by the probability that the robot is in collision with any of the possible sensor returns or occupies any unknown space.

$$P\big(\text{Collision for } p_t(\mathbf{q}) |\ \mathcal{O}, \mathcal{U}\big) = P\big(\mathbf{q}(t_i) \in \{\mathcal{O} \text{ or } \mathcal{U}\}\big) \tag{2}$$

Note that the probabilities are not disjoint, since for any non-zero-volume robot, a given configuration can be in collision with multiple frustums, occupied or unknown. To evaluate this probability, an integral over all possible configurations must be integrated, where the integrand is the probability distribution of configurations $p(\mathbf{q})$ times the indicator function of whether the volume of the robot $\mathcal{R}(\mathbf{q}) \subset \mathbb{R}^3$ in configuration $\mathbf{q}$ intersects any occupied or unknown space:

$$P\big(\text{Collision for } p(\mathbf{q}) \mid \mathcal{O}, \mathcal{U}\big) = \int_{SE(3)} p(\mathbf{q}) \times \mathbb{1}\Big(\big(\mathcal{R}(\mathbf{q}) \cap \mathcal{O}\big) \bigcup \big(\mathcal{R}(\mathbf{q}) \cap \mathcal{U}\big)\Big) d\psi \tag{3}$$

where ψ represents a differential over SE(3) and

$$\mathcal{R}(\mathbf{q}) \cap \mathcal{O} = \bigcup_j \mathcal{R}(\mathbf{q}) \cap \mathcal{F}_{\mathbf{d}_j} \tag{4}$$

$$\mathcal{R}(\mathbf{q}) \cap \mathcal{U} = \big(\bigcup_j \mathcal{R}(\mathbf{q}) \cap \mathcal{F}_{(\text{occluded by } \mathbf{d}_j)}\big) \bigcup \mathcal{R}(\mathbf{q}) \cap \mathcal{U}_{(\text{outside FOV})} \tag{5}$$

Even given a solution to this integral in Equation 3, this only provides an evaluation of one possible distribution of configuration at some future time, and hence the probability of collision for the time-varying distribution of configuration is still difficult to evaluate, given that all future positions in time are dependent on previous positions in time. One route to estimating this probability is through Monte Carlo simulation, in which random trajectories are simulated through the Markov chain, and collision-checked deterministically against the conservative depth image. In the limit of infinite samples, the proportion that are in collision of the sampled trajectories is an exact evaluation of the time-varying configuration distribution's collision probability.

In the next section, we present a fast approximation to the above which is tractable for online calculation.

3.2 Fast Approximation of Collision Probabilities

Given the goal to evaluate collision probabilities in real-time for the purpose of collision avoidance, some approximations are in order. In the current approach, two important simplifying assumptions are made, each of which lose conservatism but enable fast computation speed. Although these are significantly simplifying assumptions, the simulation results presented in this paper suggest that even these approximations offer a significant improvement over deterministically collision checking trajectories.

First, we consider a Markov chain with no process noise so that the computation of the probability distribution $p_t(\mathbf{q})$ is a deterministic function of time. For implementation on a computer, future positions are sampled in time, and with this independence approximation the time-varying configuration distribution's total probability of collision is approximated as the subtraction from unity of the product of the no-collision probabilities:

$$P\big(\text{Collision}, p_t(\mathbf{q})\big) \approx 1 - \prod_i \big[1 - P(\text{Collision}, p_t(\mathbf{q}), \text{time } t_i)\big] \tag{6}$$

Second, we make an additional independence approximation: for any t_i, the probability of collision with any depth return $\mathbf{d}_j$ is independent of the probability of collision with all other depth returns $\mathbf{d}_0, ..., \mathbf{d}_n$. Since each distribution of configurations over time is assumed independent of all other times, the computation may be done all in one product:

$$P\big(\text{Collision}, p_t(\mathbf{q})\big) \approx 1 - \prod_{i,j} \big[1 - P(\text{Collision}, p_t(\mathbf{q}), \text{time } t_i, \text{point } \mathbf{d}_j)\big] \tag{7}$$

We note that this makes it so that each $P(\text{Collision, time } t_i, \text{point } \mathbf{d}_j)$, for each maneuver, is a 100% parallelizable computation. Accordingly, in future work, we plan to investigate GPU implementations.

With the full independence approximation (Equation 7), naively, one evaluation of $P(\text{Collision, time } t_i, \text{point } \mathbf{d}_j)$ is required for each maneuver evaluated, for each point in time t_i, for each point cloud point $\mathbf{d}_j$, i.e. naive complexity of $O(n_{\mathcal{T}} \times n_t \times n_{\mathbf{d}})$, where $n_{\mathcal{T}}$ is the number of maneuvers, n_t is the number of time steps considered, and $n_{\mathbf{d}}$ is the number of depth points considered in the point cloud for each robot position. In practice for our quadrotor, 10-100 is sufficient for $n_{\mathcal{T}}$ and n_t, whereas a dense point cloud may result in naive $n_{\mathbf{d}}$ being orders of magnitude higher. Even for a "low-resolution" 160x120 depth image, this is $n_{\mathbf{d},\text{Total}} = 19,200$ points. We next consider spatial partitioning data structures to reduce $n_{\mathbf{d}}$ to an amount amenable to real-time computation.

3.3 Spatial Partitioning of Depth Image Data

In practice, a spatial partitioning data structure such as a *k-d*-tree proves to be extremely useful for collision probability approximation in a depth image, due to the ability to efficiently query nearest neighbors even for low-probability collisions. For any future robot positions in time, a specified number $n_{\mathbf{d}}$ of closest depth sensor returns can be efficiently queried, where $n_{\mathbf{d}} \in [1, n_{\mathbf{d},\text{Total}}]$ can be varied to trade off computational cost and approximation accuracy.

In contrast to deterministic collision checking, collision probability approximation significantly benefits from spatial partitioning as opposed to operating directly on the depth image. In deterministic collision checking, there is no faster way to collision-check than using the raw depth image – a future robot position $\mathbf{q}_{x,y,z}$ in the Cartesian depth image frame may be projected into depth image space via $(u, v) = \pi(K\mathbf{q}_{x,y,z})$ to find the corresponding pixel (u, v) location (where K is the camera intrinsic properties), and that pixel can be deterministically collision checked. With this method, it is possible to finish collision checking with lower computational cost than it even takes to build a *k-d*-tree, let alone query it for nearest neighbors.

With collision probability approximation, however, since we care about "long-tails" of the robot position distribution, there is great advantage in the nearest neighbor search. In order to consider long-tail positions in the direct depth image method, a large block of pixels around the center pixels (u, v) needs to be considered. We also note, however, that since the direct depth image method requires no building of a new data structure, highly parallelized implementations may tip in its favor (as opposed to needing to sequentially build a *k-d*-tree, then searching it). Computational latencies of these different approaches for our quadrotor implementation are analyzed in the next section.

3.4 Integrating Reactive and Navigation Objectives

A benefit of the probabilistic maneuver evaluation approach is that it naturally offers a mathematical formulation that integrates "reactive"-type obstacle avoidance with arbitrary navigation objectives. Whereas other "layered" formulations might involve designed weightings of reactive and planning objectives, the probabilistic formulation composes the expectation of the reward, $\mathbb{E}\big[R\big]$. Given some

global navigation function that is capable of evaluating a reward $R_{nav}(\mathcal{T}_i)$ for a given maneuver, the expected reward is computed from the global navigation function, and the probability of collision:

$$\mathbb{E}\big[R(\mathcal{T}_i)\big] = P(\text{No Collision}, \mathcal{T}_i)R_{nav}(\mathcal{T}_i) + P(\text{Collision}, \mathcal{T}_i)R_{collision} \quad (8)$$

As we show in the simulation experiments, $R_{nav}(\cdot)$ may not even need to consider obstacles, and collision avoidance can still be achieved. The global navigation function can be, for example, just Euclidean progress to the global goal for environments with only convex obstacles, or for environments with dead-ends could for example be a cost-to-go using Dijkstra's algorithm. A key point is that with the instantaneous mapless approach handling collision avoidance, $R_{nav}()$ can be naive, and/or slow, although a good $R_{nav}(\cdot)$ is only a benefit. One parameter that must be chosen, and can be tuned up/down for less/more aggressive movement around obstacles, is the cost (negative reward) of collision, $R_{collision}$.

Given a library of maneuvers, the optimal maneuver $\mathcal{T}^*$ is then chosen as:

$$\mathcal{T}^* = \underset{i}{\text{argmax}} \;\; \mathbb{E}\big[R(\mathcal{T}_i)\big] \quad (9)$$

4 Implementation for High Speed Quadrotor Flight

The formulation presented above is generalizable for different robot models and choices of maneuver library with probability distributions. In this section we present a specific implementation for high-speed quadrotor control without a map.

4.1 Feedback Control with an MPC-Library

We use an approach similar to a traditional trajectory library, except our library is generated online based on a simplified dynamical model. In the sense that a model is used for real-time control, and we use no "trajectory-tracking" controller, this is MPC (Model Predictive Control), but since we perform no continuous optimization but rather just select from a discrete library, we refer to this as an MPC-Library method. Our plan is to expand our analysis of this method in future work. For now, it plays a supporting role for the probabilistic formulation.

To build intuition of our simple model, we first describe the basic version of a constant-input double-integrator (constant-acceleration point-mass) modeled around the attitude controller. Later, we add a small adjustment, to extend to a triple integrator. The constant-acceleration version approximates the quadrotor in feedback with the inner-loop attitude and thrust controller as a point-mass capable of instantaneously producing an acceleration vector of magnitude $||\mathbf{a}|| \leq a_{max}$ in any direction. Together with gravitational acceleration, this defines the achievable linear accelerations. We use a simple constant-input double-integrator model in the local frame:

$$\mathbf{p}_i(t) = \frac{1}{2}\mathbf{a}_i t^2 + \mathbf{v}_0 t \quad \text{for maneuver } \mathcal{T}_i, t \in [0, t_f] \quad (10)$$

whe $\mathbf{v}_0$ is initial velocity and $\mathbf{a}_i$ is the chosen acceleration that defines the maneuver $\mathcal{T}_i$. By definition in the local frame the initial position $\mathbf{p}(0) = 0$. This model is applied with the inner-loop attitude and thrust controller in the loop, as depicted in Figure 2. Given a desired acceleration $\mathbf{a}_i$, geometry defines the mapping to {roll, pitch thrust} required to produce such an acceleration, given any yaw.

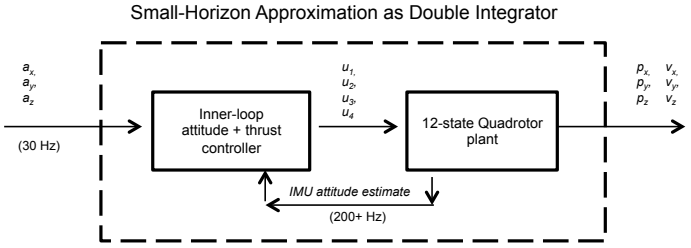


Fig. 2: Dynamics approximation considered: the quadrotor is modeled in feedback with the inner loop attitude and thrust controller.

A motivating factor for this model is that the overwhelmingly ubiquitous implementation for quadrotor control involves a high-rate (∼200+ Hz) inner-loop attitude and thrust controller which achieves desired roll, pitch, yaw (or angular rates) and thrust commands. The desirability of quickly closing an error-minimizing PID or similar loop around the IMU makes this an attractive control design choice. The model additionally has the properties of being closed-form for any future $t \in [0, t_f]$, of being linear, and cheap to evaluate.

Extension to Piecewise Triple-Double Integrator Model The limitations of the constant-acceleration model are clear, however: it does not consider attitude dynamics, even though they are fast (∼100-200 ms to switch between extremes of roll/pitch) compared to linear dynamics. It is preferable to have a model that includes attitude dynamics: as an example, the initial attitude of the quadrotor may significantly affect "turn-left-or-right" obstacle avoidance decisions. (I.e., if the vehicle is already rolled to the left, even with zero left/right velocity, it is easier to move to the left than it is to the right.)

Accordingly, the model we use is a piecewise constant-jerk for $t \in [0, t_{jf}]$, followed by constant-acceleration model for $t \in [t_{jf}, t_f]$. Although the actual attitude dynamics are nonlinear, a linear approximation of the acceleration dynamics via a constant-jerk "acceleration transition period" is an improved model over the constant-acceleration-only model. By approximating the "jerk period" t_j as ∼200 ms for our quadrotor, this piecewise constant-jerk constant-acceleration model is:

$$\mathbf{p}_i(t) = \frac{1}{6}\mathbf{j}_i t^3 + \frac{1}{2}\mathbf{a}_0 t^2 + \mathbf{v}_0 t + \mathbf{p}_0 \quad \text{for } [0, t_{jf}], \mathcal{T}_i \tag{11}$$

$$\mathbf{v}_i(t) = \frac{1}{2}\mathbf{j}_i t^2 + \mathbf{a}_0 t + \mathbf{v}_0 \quad \text{for } [0, t_j] \tag{12}$$

$$\text{where } \mathbf{j}_i = \frac{\mathbf{a}_i - \mathbf{a}_0}{t_j} \tag{13}$$

$$\mathbf{p}_i(t) = \frac{1}{2}\mathbf{a}_i t^2 + \mathbf{v}_j t + \mathbf{p}_j \quad \text{for } (t_j, t_f] \tag{14}$$

$$\text{where } \mathbf{p}_j = \mathbf{p}_i(t_j), \mathbf{v}_j = \mathbf{v}_i(t_j) \tag{15}$$

The trajectories produced by this piecewise triple-double integrator retain the properties of being closed-form for any future $t \in [0, t_f]$, of being linear, and cheap to evaluate. There is certainly more work to be done on this class of models, as we plan to do in future work.

Due to closed-form property, we do not need to sequentially forward-simulate a maneuver by integration, instead we can calculate directly the position $\mathbf{p}(t)$ of the robot for any future time $t \in [0, t_f]$. This also means that calculating each position of each action at each time is 100% parallelizable, which is especially attractive for our probabilistic approximations.

Gaussian Propagation Through Linear Model With our linear model, we use Gaussian initial state estimate noise propagated with no process noise. With only sensor noise, and not process noise, the linear velocity estimates $\mathbf{v}_0$ coming from our state estimation system are the only source of uncertainty. For each $\mathbb{A}_i$ at each t, we assume that the robot's future position is a Gaussian distribution:

$$\mathbf{p}(t) \sim N(\hat{\mathbf{p}}(t), \Sigma_p(t))$$

Where in the local frame the covariance of the position is due to the initial velocity covariance:

$$\Sigma_{p_i(t)} = t^2 \Sigma_{v_0}$$

Note that for the constant-jerk portion, an initial acceleration estimate, $\mathbf{a}_0$ is required. For now, this is assumed to be a deterministic estimate – since roll, pitch, and thrust are more easily estimated than linear velocities, this is a reasonable assumption.

4.2 Trajectory Library and Attitude-Thrust Setpoint Control

We use a finite 3D trajectory library constrained to a single altitude 2D plane, where the trajectories are determined by achieving different acceleration vectors as inputs to the piecewise triple-double integrator approximation. To create the bins for a_x and a_y each, we approximate the maximum horizontal acceleration and sample over possible horizontal accelerations around a circle in the horizontal plane. The max horizontal acceleration is approximated as the maximum thrust vector (T_{max}) angled just enough to compensate for gravity: $a_{max} = \frac{\sqrt{T_{max}^2 + (mg)^2}}{m}$. By sampling both over the horizontal acceleration with just a few discretizations (for example, $[a_{max}, 0.6a_{max}, 0.3 * a_{max}]$) and just 8 evenly spaced θ over $[0, 2\pi]$, this yields a useful discretization in the horizontal plane. We also add a $[0, 0, 0]$ acceleration option, for 25 trajectories total in the

plane. The figure below is a depiction of forming the horizontal $[a_x, a_y]$ bins. For this 2D-plane implementation, a PID loop on z-position maintains desired altitude by regulating thrust. We allow for slow yawing at 90 degrees per second towards the direction $\mathbf{p}(t_f) - \mathbf{p}_0$, which in practice has little effect on the linear model and allows for slow yawing around trees.

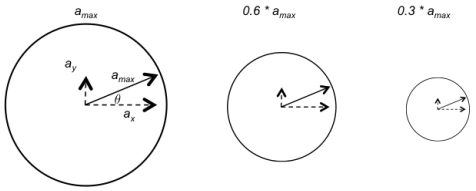


Fig. 3: Formation of horizontal actions from sampling over thrust-scaled circles in the horizontal plane.

Evaluation of Collision Probability and Global Navigation Given a robot position at future time t, we first check if the projection of the position is within the bound of the depth image. To be conservative, we assign collision probability of 1 to positions outside the depth image bounds. To allow for speeds past 10 m/s, given $t_f = 1.0$ s, we do not consider positions beyond our simulated depth image horizon of 10 meters to be in collision, and evaluate them against the $k - d - tree$. An exception is made to positions that are within a small radius $r = 0.5$ of the robot, which even if they are outside of the depth image FOV are not assigned collision probability of 1, to allow the robot to stabilize one position, for example on takeoff.

If the future robot position is not occluded, we can approximate the probability that the depth return point and the robot are in collision, by multiplying the point Gaussian probability density by the volume V_r of the robot's sphere (see Toit 2011):

$$P(\text{Collision, time } t) \approx V_r \times \frac{1}{\sqrt{\det(2\pi\Sigma_r)}} \exp\left[-\frac{1}{2}(x_r - x_p)^T \Sigma_r^{-1}(x_r - x_p)\right] \quad (16)$$

where Σ_r is the covariance of the robot position. This approximation has been used in the chance-constrained programming literature, where the probability of collision at any time is upper bounded as a constraint in an optimization. Here, we instead use this equation to form an approximation of the collision probability for a whole trajectory, and only perform evaluation-and-selection rather than optimization. The above equation is used together with Equation 6 to evaluate probabilities of collision for all the maneuvers in the library.

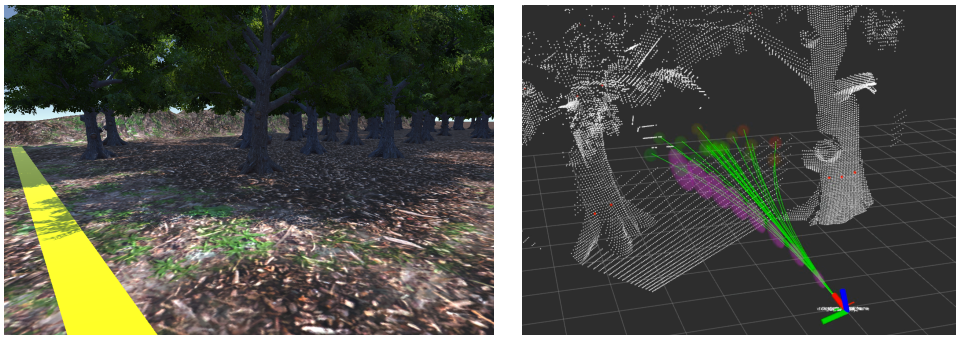
For our quadrotor race through the forest, since the obstacles are all convex and so navigating out of dead-ends is not a concern, we use a simple Euclidean progress metric as our navigation function R_{nav}, plus a cost on terminal velocity:

$$R_{nav}(\mathcal{M}_i) = ||\mathbf{p}_0 - \mathbf{p}_{goal}|| - ||\mathbf{p}_i(t_f) - \mathbf{p}_{goal}|| + k||\mathbf{v}_i(t_f)|| \quad (17)$$

Where we used $k = 10$, and $R_{collision} = -10,000$.

5 Simulation Experimental Setup

5.1 Simulator Description



(a) (b)

Fig. 4: Screenshots from our race-through-forest simulation environment. (a) Screenshot from Unity of our forest simulation environment. (b) Screenshot from Rviz which shows the evaluation of the 25-trajectory real-time-generated MPC-library. The chosen trajectory and the 1-σ of the Gaussian distribution over time are visualized for the chosen trajectory. The small sphere at the end of each trajectory indicates approximated collision probabilities from low to high (green to red), evaluated against the 160x120 depth image point cloud (white) and 100-laser 2-dimensional LIDAR point cloud (red).

To facilitate the comparison study, simulation software was developed to closely mimic the capabilities of our hardware platform for the Draper-MIT DARPA FLA (Fast Lightweight Autonomy) research team. The sensor configuration includes a depth sensor that provides dense depth information at 160x120 resolution out to a range of 10 meters, with a FOV (field of view) limited to 58 degrees horizontally, 45 degrees vertically. To complement the range and horizontal field of view limitations of the camera system, a simulated 2D scanning lidar is incorporated to provide range measurements reaching out to 30 meters. Both depth sensors are simulated at 30 Hz. Pixhawk software is used to control the attitude and thrust of the quadrotor.

Drake was used to simulate vehicle dynamics using a common quadrotor 12-state nonlinear quadrotor model while Unity (game engine) provides high fidelity simulated perceptual data that includes GPU-based depth images and raycasted 2D laser scans. The flight controller uses a version of the Pixhawk firmware running in the loop (SITL) that utilizes an EKF over noisy simulated inertial measurements to estimate attitude and attitude rates of the vehicle. We use ROS as the underlying communication and logging layer to tie the various subsystems together and MAVROS is used to send attitude and thrust setpoints to the Pixhawk flight controller software.

5.2 Experimental setup

The experiments were carried out in a virtual environment that consists of an artificial forest valley that is 50 meters wide and 160 meters long. The corridor is filled with 53 randomly placed trees whose trunks are roughly 1 meter in diameter and is surrounded by steep, impassable terrain. SpeedTree was used for the tree geometry and includes simulated wind that articulates the branches and leaves to increase the realism of the perceptual data.

For each experimental trial, the vehicle is started on one side of the forest valley and is commanded to fly at 1.8 meters altitude to a goal on the far end, forcing it to traverse a 150 meter stretch of the obstacle-laden environment. A timer is started when the vehicle crosses the 5 meter mark and stopped either when a collision occurs or when the 155 meter mark is reached. If the vehicle is able to navigate the forest without colliding with any of the trees or terrain, the trial is considered a success. If a collision occurs or if the finish line has not been reached in a predetermined amount of time, the trial is considered a failure.

The experiments were repeated for each algorithm at various target velocities $\{$ 3, 5, 8, 12$\}$ meters per second and with increasing levels of state estimate noise for $x, \dot{x}, y, \dot{y}$. We do not simulate noise in the altitude or in the orientation or angular velocities since these are more easily measurable quantities. To simulate noise that causes position to drift over time, we take the true difference in x, y over a timestep, $\Delta\mathbf{p}_{x,y}(t)$, and add zero-mean Gaussian noise which is scaled linearly with the velocity vector. The three noise levels we use are $\sigma = \{0, 0.1, 1\}$ which is scaled by $\frac{\sigma}{10}\mathbf{v_{true}}$. We also add true-mean Gaussian noise to $\dot{x}$ and $\dot{y}$, with standard deviations that are the same as for position noise. Accordingly we have:

$$\mathbf{p}_{noisy}[i+1] \sim \mathcal{N}(\mathbf{p}_{true}[i+1] - \mathbf{p}_{true}[i], \frac{\sigma}{10}\mathbf{v_{true}})$$

$$\mathbf{v}_{noisy}[i] \sim \mathcal{N}(\mathbf{v}_{true}[i], \frac{\sigma}{10}\mathbf{v_{true}})$$

The total time taken and the trial outcome was recorded for 10 trials at each noise and speed setting. The results are tabulated in grid form and indicate the path taken through the forest on each trial as well as the overall success rate of each method.

5.3 Pure pursuit algorithm description

We compare our method to a typical map-based robotics navigation solution that consists of a global path planner that is paired with a path following algorithm. The particular implementation we chose functions by maintaining a global probabilistic occupancy grid (Octomap) with a 0.2 meter voxel size into which all incoming depth information is inserted and registered with the current vehicle pose. At a specified rate, a horizontal slice of the map is extracted and a

globally optimal path from the vehicle's position to the goal is computed using Dijkstra's algorithm. The path planning includes a soft cost on proximity to obstacles to find paths that maintain sufficient clearance. We then use a pure pursuit algorithm to command a vehicle velocity along the resulting path to the goal.

This approach has been validated in hardware during experiments in which the quadrotor was commanded to navigate a previously unknown cluttered warehouse environment at speeds ranging from 2.0 m/s to 5.5 m/s to a goal approximately 65 meters from the starting location. Using this method, we have been able to successfully navigate through 24 randomly configured obstacle courses using a state estimator that fuses measurements from a laser scan matcher with accelerometer and gyroscope data from the Pixhawk's IMU.

6 Simulation Results and Discussion

The key metric for our comparison of the three methods is the no-collison success rate of reaching the finish line, and is presented in Figure 5.

The results for the global path planning and following approach show both the limitations on handling higher speed, and on handling higher state estimate noise. The approach was not able to handle any of the severe noise ($\sigma = 1$) for any of the speeds and was only able to reliably reach the goal at 5 m/s and below, with zero or little state estimate noise. These limits on speed and state estimate noise match well our experimental results in hardware. Primary inhibiting factors for this approach's success are (i) dependence on a global position estimate, (ii) latency incurred by processing sensor data into a global map, (iii) latency incurred by path planning on the local map, and (iv) neglect of vehicle dynamics, which are increasingly important for obstacle avoidance at higher speeds.

For the deterministic method, the average time to goal on a successful run was faster than the probabilistic method by approximately 14%. However, the deterministic nature of the collision checking causes the method to leave little margin for error while navigating around obstacles. Thus, small inaccuracies in the model or the state estimate can lead to fatal collisions.

The results for the probabilistic method demonstrate a marked increase in robustness at higher speeds and with the highest of state estimate noise. The sacrifice in average time to goal is outweighed by the gains in robustness. Since the methods differ only in the way they consider collisions, the results suggest that the probabilistic method is superior.

Additionally, an important practical consideration is that, given our fast collision probability approximations as presented, the computational costs of both methods are nearly identical, as is displayed in Table 1. This is a strong argument for replacing deterministic collision checking with fast collision probability approximation in a wide number of scenarios.

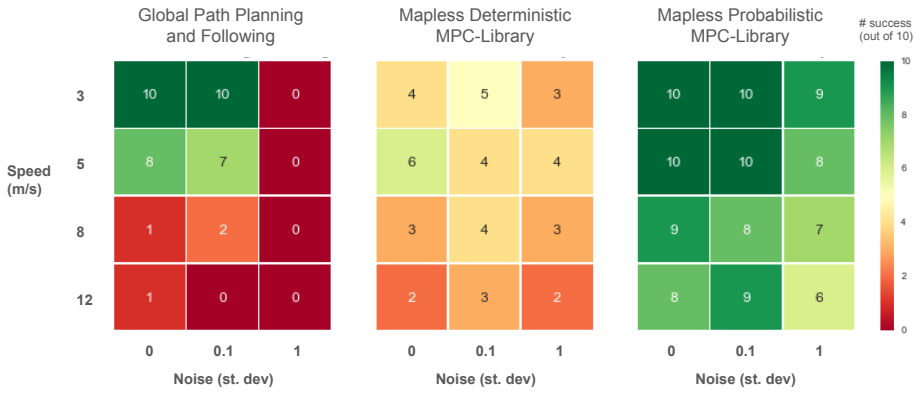


Fig. 5: Comparison summary of number of successful collision-free trials for the different approaches tested in in our simulated quadrotor race through the forest. Ten trials were run for each of the three approaches, for four different speeds {3, 5, 8, 12} meters per seconds, and for three different levels of 2-dimensional state estimate noise (x-y only, no noise on z), σ={0,1,10} for $\mathbf{p}_{noisy}[i{+}1] \sim \mathcal{N}(\mathbf{p}_{true}[i{+}1] - \mathbf{p}_{true}[i], \frac{\sigma}{10}\mathbf{v_{true}})$ and $\mathbf{v}_{noisy}[i] \sim \mathcal{N}(\mathbf{v}_{true}[i], \frac{\sigma}{10}\mathbf{v_{true}})$ as described.

	Deterministic, N=1		**Probabilistic, N=1**		**Probabilistic, N=10**	
Subprocess	Average time (μs)	Percentage time (%)	Average time (μs)	Percentage time (%)	Average time (μs)	Percentage time (%)
Building kd-tree	1900 +/- 700	51.0	2000 +/- 500	58.4	1900 +/- 400	43.0
Evaluating future positions from real-time generated 25-trajectory MPC-library	4 +/- 1	0.1	4 +/- 1	0.1	4 +/- 1	0.1
Evaluating collision probabilities with N-nearest neighbor search on kd-tree	1800 +/- 800	48.3	1400 +/- 600	40.9	2500 +/- 1000	56.5
Evaluating expected reward, given R_{nav}	2 +/- 1	0.1	2 +/- 1	0.1	2 +/- 1	0.0
Calculating attitude setpoint for attitude controller	17 +/- 5	0.5	17 +/- 5	0.5	17 +/- 5	0.4

Table 1: Measured averages and standard deviations of subprocess latencies for our implementations.

7 Future Work

There are several extensions to this line of work that we plan to work on in the near future. For one, we plan to validate the method in hardware. Additionally, the highly parallel nature of the fast collision probability approximation algorithm is amenable to data-parallel implementations on a GPU. We also plan to expand on the MPC-library approach, including true 3D flight, increased variety of maneuvers, and analysis of the accuracy of the model approximation. We also plan to characterize the performance of the collision probability approximation with more elaborate global navigation functions.

Bibliography

[1] Richter, C., Vega-Brown, W., Roy, N.: Bayesian learning for safe high-speed navigation in unknown environments. In: Proceedings of the International Symposium on Robotics Research (ISRR 2015), Sestri Levante, Italy (2015)