Aggressive 3-D collision avoidance for high-speed navigation

Conference Paper · May 2017

DOI: 10.1109/ICRA.2017.7989677

2 authors:

Brett T Lopez
NASA Jet Propulsion Laboratory
38 PUBLICATIONS **240** CITATIONS
SEE PROFILE

Jonathan How
Massachusetts Institute of Technology
829 PUBLICATIONS **22,585** CITATIONS
SEE PROFILE

Aggressive 3-D Collision Avoidance for High-Speed Navigation

Brett T. Lopez and Jonathan P. How

***Abstract*— Autonomous robot navigation through unknown, cluttered environments at high-speeds is still an open problem. Quadrotor platforms with this capability have only begun to emerge with the advancements in light-weight, small form factor sensing and computing. Many of the existing platforms, however, require excessive computation time to perform collision avoidance, which ultimately limits the vehicle's top speed. This work presents an efficient perception and planning approach that significantly reduces the computation time by using instantaneous perception data for collision avoidance. Minimum-time, state and input constrained motion primitives are generated by sampling terminal states until a collision-free path is found. The worst case performance of the Triple Integrator Planner (TIP) is nearly an order of magnitude faster than the state-of-the-art. Experimental results demonstrate the algorithm's ability to plan and execute aggressive collision avoidance maneuvers in highly cluttered environments.**

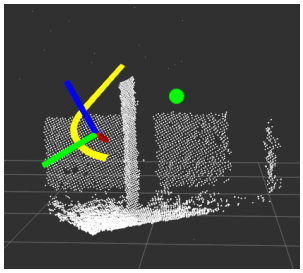
I. INTRODUCTION

Navigating in unknown environments is essential for many robot applications, and it is a problem that encompasses the key issues of real-time perception, path planning, and state estimation. The standard perception and planning pipeline used in navigation has two fundamental functions: 1) transform sensor data into a usable representation of the world and 2) use the transformed data to find a path to a known goal. But the planning strategy is often selected with little consideration of the computational effort required to transform sensor data into the required representation. To achieve the best performance, the planning approach, the required representation, and the time needed to convert the sensor data into this form should be chosen at the same time. And, of course, this selection will be mission/task dependent. For example, for high-speed navigation there is very little time available to react to newly observed obstacles, so there is a strong emphasis on fast, simple representations. To address this point, this work presents an efficient perception and planning approach that is demonstrated to enable aggressive quadrotor flight through unknown environments.

Advancements in lightweight sensing and computing have led to quadrotor platforms capable of navigating in unknown environments. Many of these platforms [1]–[4] use laser scans or point clouds [5] to populate an occupancy grid [6], [7]. The occupancy grid is then used to evaluate motion primitives [4] or is further processed [1], [3] to be used in an optimization. Although an occupancy grid, or some other type of map, is necessary for global planning, its utility for local collision avoidance is ultimately a consequence of the planning strategy. Furthermore, the computational burden of building and processing a map for local planning can be alleviated by choosing a different planning strategy.

Aerospace Controls Laboratory, Massachusetts Institute of Technology, 77 Massachusetts Ave., Cambridge, MA, USA btlopez, jhow@mit.edu

(a) Quadrotor navigating in unknown environment.



(b) Instantaneous point cloud and trajectory visualization.

Fig. 1. A quadrotor navigating in an unknown environment. (a): The quadrotor is unaware of oncoming obstacles because of its limited sensing range, shown in red. (b): The vehicle observes the first set of obstacles and generates a 3-D motion primitive, in yellow, to go around the first pillar and over the set of boxes towards the global goal, in green. The RGB coordinate system is representative of the vehicle's position and orientation.

The main contribution of this work is the development and experimental verification of a collision avoidance algorithm, known as the **T**riple **I**ntegrator **P**lanner (**TIP**), that is nearly an order of magnitude faster than the state-of-the-art and can operate at a level of agility near the physical limits of the vehicle. The key property of the algorithm is the ability to generate collision-free, dynamically feasible paths within 5ms (worst case) of receiving a point cloud. The short computation time is achieved as a result of three key developments. First, the algorithm uses a world representation easy to obtain from instantaneous perception data in the form of point clouds (Section IV-B). Second, a closed-form solution for a minimum-time, state and input constrained optimal control problem is used to generate high-performance motion primitives (Section IV-C). Third, the motion primitives are intelligently sampled [8] for possible collisions (Section IV-D). The planner's ability to aggressively navigate though unknown environments is a direct consequence of selecting the planning strategy, required representation, and sensor process time simultaneously.

A subsequent contribution of this work is the flight demonstration of the Intel® RealSense™ R200. It is also shown that the 34g RGB-D camera's indoor range can be increased from the standard 4m to 6-8m with an external infrared emitter. In addition, a 15-25m outdoor range is demonstrated.

II. RELATED WORK

Path planning algorithms can be classified as optimization-based or as motion primitives. Optimization-based trajectory planners for quadrotor became computationally possible

when they were proved to be differentially flat [9]. Adaptations of the minimum-snap formulation original presented in [9] and modified in [10] have recently appeared in the literature for receding horizon planning. For instance, [1] built a local occupancy grid with the most recent perception data and generated a minimum-jerk trajectory through waypoints from an A* search over the local occupancy grid. An approximate method [12] was used for waypoint time allocation instead of gradient descent as in [9], [10] to achieve real-time performance. In addition, the free-space in the occupancy grid was convexified to prevent sampling the output trajectory for collisions. [3] presented a novel optimization formulation that generated minimum-snap trajectories that satisfied higher-order state constraints. This approach relied on identifying free-space flight corridors within an occupancy grid to find collision-free trajectories. [2] also used the minimum-snap formulation but with a soft cost for state constraints to improve computation time. While these adaptations are much faster than the original minimum-snap algorithm, they require a very specific world representation that is computationally intensive to obtain, as evident by the reported long computation times. Further, no experimental results demonstrating aggressive collision avoidance have been presented.

Motion primitives can be generated by either sampling the vehicle's state space or control space [13]. State-based primitives easily satisfy state constraints but are not always dynamically feasible and can be difficult to compute because a boundary-value problem needs to be solved online [13]. Control-based primitives are more common because they are guaranteed to be dynamically feasible and are easy to generate offline for systems with complex dynamics [13]. In either formulation, the primitives are assigned a cost at each planning epoch and the primitive with the lowest cost is selected for execution. [14] was one of the first to successfully navigate a UAV in simulation using the control-based approach. [4] used precomputed motion primitives using the maximum dispersion algorithm [15] to navigate a quadrotor through a forest. The primitives were evaluated on an a map that contained a short history of the observed world. [16] presented a state-based method that generated minimum-jerk primitives online given the vehicle's current state and desired final state. To capture state and input constraints, however, the method required iterating through possible final times until the constraints were satisfied. Motion primitives are an appealing planning strategy because they are not computationally complex to evaluate online and provide flexibility in how the world is represented. However, few solutions exist in the literature that are dynamically feasible, satisfy state constraints, and are computationally efficient to compute online [13].

III. Problem Formulation

Quadrotors are differentially flat [9] so their entire state can be represented by the flat variables $\boldsymbol{x} = [x\ y\ z\ \psi]^T$, where ψ is the vehicle's yaw, and their derivatives. We use the approach taken by [11] to model the quadrotor as a state and input constrained triple integrator system. This model does not capture the full nonlinear dynamics of quadrotors but is accurate enough to achieve good attitude and angular rate tracking even during an aggressive maneuver (Section VI). Furthermore, planning with a kinematic model is less computationally intensive and has been well studied in the optimal control community [17].

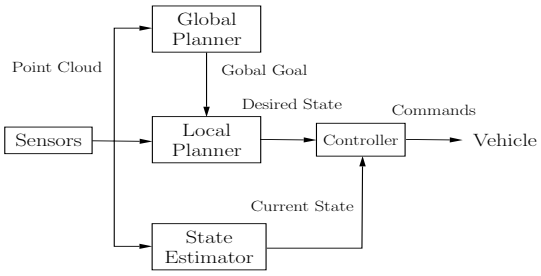


Fig. 2. System architecture used in this work. A point cloud is used for collision avoidance by the local planner. Position and velocity reference commands are sent from the local planner to a position controller. Acceleration and jerk are used for feedforward. It is assumed that a global planner is sending a global goal to the local planner.

Let the quadrotor's state $\mathbf{x}$, assuming yaw is constant, and control input $\mathbf{u}$ be

$$\mathbf{x} = \begin{bmatrix} \boldsymbol{x}^T & \dot{\boldsymbol{x}}^T & \ddot{\boldsymbol{x}}^T \end{bmatrix}^T = \begin{bmatrix} \boldsymbol{x}^T & \boldsymbol{v}^T & \boldsymbol{a}^T \end{bmatrix}^T \tag{1}$$

$$\mathbf{u} = \dddot{\boldsymbol{x}} = \boldsymbol{j}, \tag{2}$$

where $\boldsymbol{v}$, $\boldsymbol{a}$, and $\boldsymbol{j}$ are the vehicle's velocity, acceleration, and jerk, respectively.

The goal is to navigate the vehicle from it current location to a known goal location $\boldsymbol{G} \in \mathbb{R}^3$ in an unknown environment as fast as possible subject to the constraints

$$0 \leq \ ||\boldsymbol{v}||_2 \ \leq v_{\max} \tag{3}$$

$$a_{\min} \leq \ ||\boldsymbol{a}||_2 \ \leq a_{\max} \tag{4}$$

$$||\boldsymbol{j}||_2 \ \leq j_{\max}. \tag{5}$$

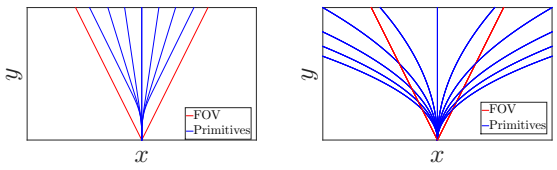
Note that any solution to this problem will not be globally optimal because the environment is only partially known, hence the ambiguity in "as fast as possible".

IV. Algorithm Overview

A. Overview

The system architecture is shown in Fig. 2. A point cloud is provided to the local planner, which performs high-rate collision avoidance. The local planner sends time indexed position and velocity reference commands to a position controller [18] that also has access to a state estimate. Acceleration and jerk are used for feedforward. It is assumed a global planner is providing a goal for the local planner. It is important to acknowledge that a global planner is necessary for environments with dead-ends as planning with instantaneous data, a key component of this work, cannot safely backtrack without some global information.

Collision avoidance with instantaneous perception data requires the entire trajectory remain in the sensor's FOV to guarantee safety [1] (Section IV-E). Relaxing the final position constraint in the minimum-time formulation



(a) Dynamically feasible, state-based primitives. (b) Control-based primitives.

Fig. 3. Primitives comparison. (a): The primitives used in this work are dynamically feasible and easily satisfy state constraints, such as remaining within the sensor's field of view (FOV). (b): Control-based primitives, generated by sampling the allowable control inputs, are always dynamically feasible but many do not satisfy state constraints.

in [11], state and input constrained motion primitives can be generated in an online fashion without a binary-search on the final time [18]. As a result, the FOV constraint can be easily included when the state-based primitives are generated. Fig. 3 shows the benefits of this formulation by comparing it to control-based primitives. All of the state-based primitives in Fig. 3a satisfy the FOV constraint as compared to only a few of the control-based primitives in Fig. 3b.

The desired final state of a primitive is specified as a desired speed and heading. In this work, speed is kept constant while possible headings are uniformly sampled from within the FOV. Each possible heading is assigned a cost that ensures the vehicle makes progress to the goal. The cost of heading i consists of a stage cost and a terminal cost:

$$J_i = \underbrace{k_1 \ \phi^2_{\text{last},i}}_{\text{Stage Cost}} + \underbrace{k_2 \ \theta^2_{\text{goal},i}}_{\text{Terminal Cost}} \ , \tag{6}$$

where the stage cost is the angle difference $\phi_{\text{last},i}$ between heading i and the previously selected heading. This heuristic prioritizes headings similar to the one last selected and dictates the planner's reluctance to large direction changes. The terminal cost is the angle difference between heading i and the heading to the global goal $\theta_{\text{goal},i}$ and ensures the vehicle makes progress to the goal. The ratio k_1/k_2 dictates the behavior of the planner. A small ratio leads to more aggressive flights while a large ratio leads to more conservative flights. In practice $k_1/k_2 = 2.0$ worked best. It should be noted that this cost function is independent of the world model by design.

The primitives are generated for each possible heading and checked for collisions. This procedure is repeated every time the planner receives a point cloud. In practice, a 7×5 heading grid was found to work well for a $56\text{deg} \times 40\text{deg}$ FOV.

B. World Model

The world model serves as a mechanism for evaluating potential paths and must be easy to obtain from sensor data to keep computation time low. At the lowest level, the model is used to determine the likelihood of collision. Fundamentally, collision checking (Section IV-D) entails finding the nearest obstacle along the intended path. Point clouds are a versatile

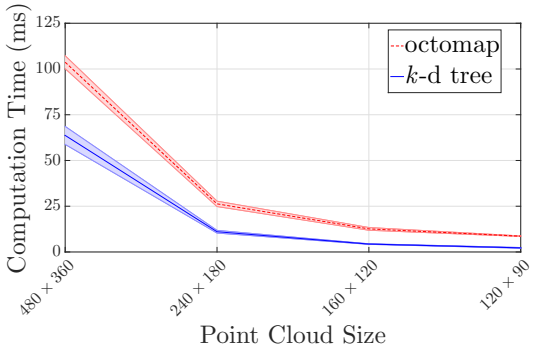


Fig. 4. Computation times for building a k-d tree and octomap with a single point cloud of different sizes. The k-d tree requires less time to generate than an octomap. The times are averaged over 10,000 iterations each.

data structure so an easy-to-obtain model capable of nearest neighbor search with a point cloud is desired. A k-d tree and octree both fit this description. For comparison purposes, octomap was used because its fundamental data structure is an octree [7]. Fig. 4 shows the computation time required to build a k-d tree from PCL's library [5] and an octomap with a single point cloud of different sizes. The octomap's resolution was set to 5cm and the elements of the point cloud were randomly generated with a maximum 3m distance from the origin. The k-d tree consistently requires less time to populate with a point cloud than an octomap. As such, a k-d tree is used to model the world to achieve the lowest computation time possible. In addition, sufficient resolution is required for accurate collision checking. 120×90 is the best choice because it provides enough resolution for collision checking in practice and almost no speed-up is obtained for smaller resolutions.

C. Minimum-Time Motion Primitives

A minimum-time, state and input constrained 3-D motion primitive is generated to point the velocity vector in a new direction. Minimum-time is an appropriate cost function given aggressive collision avoidance is needed for complex environments. The maximum allowable jerk is equally allocated to each axis to decouple the jerk constraint [11], [18], similarly for acceleration.

Using a third order kinematic model, it can be shown that the control input switches at most twice when the final position constraint is relaxed [18]. In the case of inactive state constraints, a "bang-bang" control strategy is employed. When the constraint is active, however, the control input is zero for a finite time leading to a "bang-off-bang" solution. Hence, by planning in velocity space, the problem reduces to finding the times at which the control input switches. The switching time for the inactive constraint case entails solving a quadratic equations whereas the active constraint case involves solving a linear system of equations [18].

The validity of decoupling the state and input constraints is quantified by comparing the final cost of the coupled and decoupled problems. The coupled solution was found using the MATLAB optimization toolbox YALMIP [20]. The final time was iteratively increased until a solution was

TABLE I

COUPLED AND DECOUPLED COST COMPARISON.

Heading Change (deg)	Coupled Cost	Decoupled Cost	% Difference
15	0.132	0.156	16.6
30	0.185	0.216	15.4
45	0.230	0.258	11.4
60	0.262	0.284	8.04
75	0.288	0.300	4.26
90	0.306	0.306	0

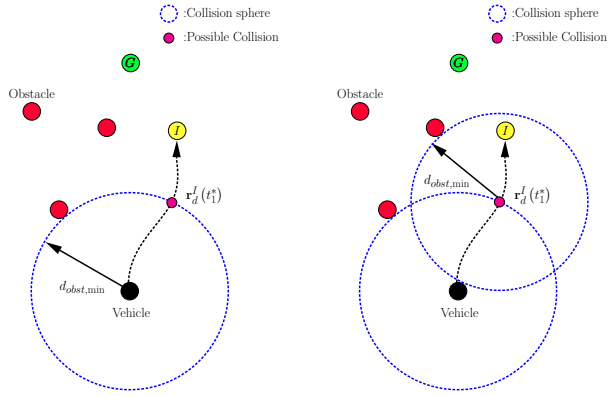
found. Table I shows the final cost for each formulation with $a_{\max} = 20^{\mathrm{m}}/\mathrm{s}^2$ and $j_{\max} = 60^{\mathrm{m}}/\mathrm{s}^3$. The percent difference is largest for small heading changes and decreases to 0 for the $90\mathrm{deg}$ case. The decoupled formulation is two to three orders of magnitude faster to solve with only a slight reduction in performance. This justifies decoupling the constraints and solving the optimal control problem for each axis independently. It is important to note that, although the final cost changes for different optimization parameters, the percent difference does not.

D. Collision Checking

Collision checking is generally the most computationally intensive component of path planning. This work adapts an approach used by the sampling-based planning community [8] that intelligently samples the path for potential collisions. This method is more computationally efficient than finely sampling the path to check for collisions [8] and avoids excessive processing of the world model. An illustration of the method is shown in Fig. 5. When a new plan is generated, the distance to the closest obstacle is calculated, as in Fig. 5a. Since this is the closest obstacle, the entire trajectory within the sphere of radius $d_{obst,\min}$ is guaranteed to be collision free. The next time-instance that a collision is possible is when $||\boldsymbol{r}_d^I(t_1^*)|| = d_{obst,\min}$. Since a desired top-speed $v_{\max}$ is known, t_1^* can be approximated to be $t_1^* \approx \frac{d_{obst,\min}}{v_{\max}}$. Even if the vehicle is not traveling at top speed, it is guaranteed to be within the collision-free sphere for $t \leq t_1^*$. The process of finding the closest obstacle and evaluating the trajectory at the approximate boundary of the collision-free sphere is repeated, Fig. 5b, until either the closest point is an intermediate goal $\boldsymbol{I}$ (typically the edge of the sensing horizon) or an obstacle is closer than a predefined safe distance. If a collision is predicted, the next best primitive is evaluated.

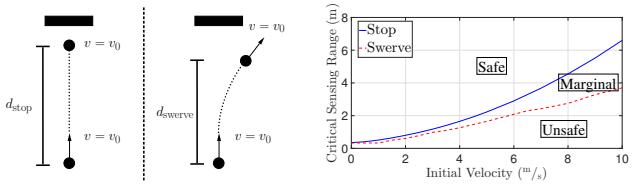
E. Safety

An important role of the planner is to ensure the vehicle never enters an inevitable collision state (ICS). Safety can be guaranteed by restricting primitives to lie within the sensor's FOV in addition to ensuring a stop maneuver is possible if no viable path is found [1]. Another approach is to ensure the vehicle can at least swerve to avoid an obstacle. While this requires an upper bound on obstacle width, the vehicle can travel faster through the environment because collision avoidance entails only perturbing the velocity vector instead



(a) First collision check. (b) Second collision check.

Fig. 5. Collision checking through intelligent sampling of the path. (a): The closest obstacle is calculated when a new trajectory is generated. (b): The trajectory is then re-evaluated at the next possible collision time, which occurs at the edge of the collision-free sphere. This process is repeated until an intermediate goal is reached or a collision is detected. The next best primitive is evaluated if a collision is detected.



(a) Stop/swerve distance definitions (b) Stop/swerve curve for planner.

Fig. 6. Stop/swerve definition and curves. (a): Stopping distance is always longer than the swerving distance since stopping requires complete dissipation of the vehicle's kinetic energy. (b): Critical sensing range is the minimum sensing range required to stop/swerve at a given speed. There are three distinct regions: safe, marginally safe, and unsafe (inevitable collision state or ICS). A faster speed is possible during marginally safe flight because collision avoidance entails only perturbing the velocity vector to swerve around obstacles.

of dissipating all of its kinetic energy to stop. To analyze this property, consider Fig. 6, which defines the stop/swerve distance and shows the stop/swerve curve as a function of initial velocity. The critical sensing range is defined to be the minimum sensing range to execute a stop/swerve maneuver. There are three distinct regions in Fig. 6b – safe, marginally safe, and unsafe (ICS). The stop maneuver is always available when in the safe region while only the swerve maneuver is possible in the marginally safe region. If the vehicle is in the ICS region a collision is unavoidable. It is clear that the sensing range required to stop is always longer than that to swerve. As a result, for a given perception range, a faster speed is achievable if the vehicle swerves instead of stops to prevent collisions. Operating in the marginally safe regime places more emphasis on the planner's ability to make quick and intelligent decisions. This work was not able to test marginally safe flight because of limited flight space but future hardware and simulation experiments are planned to evaluate the planner's performance in this regime.

F. Algorithm Benchmarking

The performance of the **T**riple **I**ntegrator **P**lanner (**TIP**) was quantified by its total computation time for different test

TABLE II

PLANNING ALGORITHM BENCHMARKING NO OBSTACLES.

Component	Time (ms)	Percentage
Transform point cloud	0.260	12.3
Build k-d tree	1.81	84.7
Sort headings	0.0144	0.674
Collision Check	0.0483	2.26
Total	2.13	100

TABLE III

PLANNING ALGORITHM BENCHMARKING NO PATH.

Component	Time (ms)	Percentage
Transform point cloud	0.268	5.31
Build k-d tree	2.01	39.9
Sort headings	0.0147	0.290
Collision Check	2.76	54.5
Total	5.06	100

TABLE IV

COMPARISON WITH STATE-OF-THE-ART ALGORITHMS.

Authors	This paper	Chen et al. [3]	Burri et al. [2]	Liu et al. [1]
Comp. Time (ms)	5.06	> 34	> 40	160

cases. The algorithm has four main components: 1) transform point cloud from camera frame to world frame, 2) build the k-d tree, 3) assign a cost and sort the possible headings, and 4) generate and check each primitive for possible collisions. The average time for each component where no obstacles are in the FOV is shown in Table II. This is the best case scenario because only a single primitive is evaluated. The k-d tree constitutes the largest percentage, 84.7%, of the computation time. The primitive were evaluated up to a distance of 3m away from the vehicle. In total, the planner only takes 2.13ms to generate a solution.

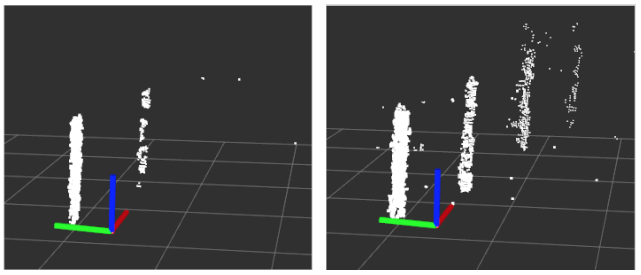
The worst case scenario is when no path is feasible and every primitive has to be checked for collision. Table III shows the computation time of each component for this case. The collision check time increases to 54.5% of the total computation time because each primitive is evaluated. For this worst case scenario, the planner still only takes 5.06ms to compute a solution. All times were recorded on a 2.70GHz Intel Core i7-2620M laptop and are an average of 48,000 samples.

Table IV compares the computation time of **TIP** to the state-of-the-art. Note that the other algorithms are highly dependent on the number of waypoints used to generate a trajectory so the reported best-case performance is used here. In addition, the time to acquire and update the world model is included in the reported times. The worst case performance of **TIP** is almost an order of magnitude faster than the next closest algorithm. Therefore, **TIP** is capable of faster flight through unknown environments because of its ability to quickly respond to new obstacle.

V. HARDWARE

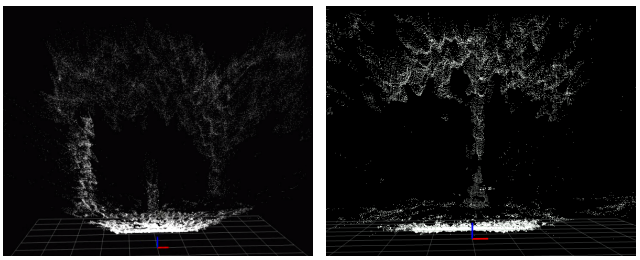
The quadrotor used in this work is shown in Fig. 7. The frame is composed of arms from a DJI F330 quadrotor and a custom Delrin base plate. The vehicle is equipped with a Jetson TX1 to enable onboard perception, planning, and control. A Vicon motion capture system is used, but only for vehicle state estimation – all obstacle knowledge is inferred

Fig. 7. Flight vehicle used in this work equipped with a Jetson TX1 for onboard perception, planning, and control. The Intel® RealSense™ R200 RGB-D camera, front, is used for onboard perception.



(a) Regular R200 point cloud. (b) R200 point cloud with external IR emitter.

Fig. 8. Indoor point cloud. (a): Point cloud using the R200's onboard IR emitter, a max range of 4m is achieved indoors. (b): Adding an external IR emitter increases the range to 6-8m for indoors. Pillars are placed every 2m. The coordinate system is representative of the camera's position. Each grid cell is 1mx1m.



(a) Outdoor scene with 3 obstacles. (b) Outdoor scene with 1 obstacle.

Fig. 9. Outdoor point cloud. (a): The point cloud for an environment with 3 obstacles that are 12m (left), 16m (middle), and 20m (right) away. (b): A 25m sensing range is possible when the sensor is in the shade.

from the onboard processing of the vehicle-mounted sensors. The vehicle can hover at approximately 42% throttle with a 4S battery, leaving enough control authority for aggressive flight experiments. The total vehicle weight is 1.2kg. The max acceleration and jerk were experimentally found to be $20^{\mathrm{m}}/_{\mathrm{s}^2}$ and $60^{\mathrm{m}}/_{\mathrm{s}^3}$, respectively.

The 34g Intel® RealSense™ R200 RGB-D camera was used for perception. This lightweight sensor uses an IR stereo camera pair to provide 3-D depth information at a maximum rate of 60fps with a 480×360 resolution. Fig. 8a shows the point cloud using the sensor's onboard IR emitter with pillars spaced every 2m. The onboard emitter projects a static infrared pattern to increase scene texture [21]. Only the second pillar is observable so the max indoor range is 4m. If an external IR source is used, such as 10 IR 850nm high-powered LED's, the max range can be increased to approximately 6-8m, as shown in Fig. 8b. The LED array is lightweight and does not consume a lot of power, making it possible to add onboard a quadrotor.

TABLE V
INTEL® REALSENSE™ R200 BENCHMARKING.

Environment	Max Range (m)
Indoor	4
Indoor w/ external IR source	6-8
Outdoor	15-25

The R200's max range is significantly longer in outdoor environments during the day. Fig. 9a shows a point cloud with three obstacles: the left is 12m away, the middle is 16m away, and the right is 20m away. Depending on the lighting, a max range of 25m is possible Fig. 9b. The sensor works best in the shade but a 15m sensing range is achieved when in direct sunlight. The sensor's max range for the indoor and outdoor tests is summarized in Table V.

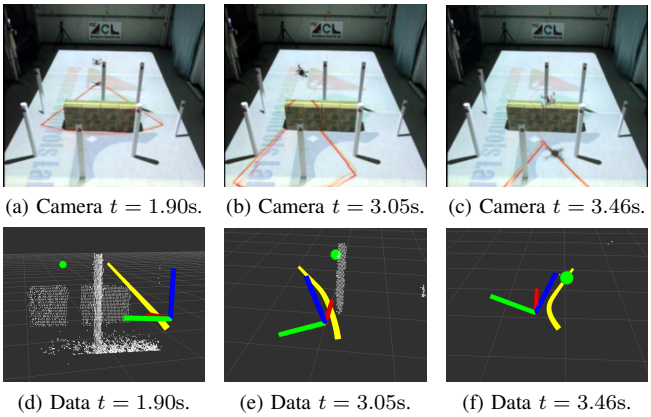
VI. EXPERIMENTAL RESULTS

TIP was experimental tested in environments of varying difficulty. Difficulty was quantified by the number of large attitude and fast angular rate maneuvers utilized by the planner during collision avoidance. Three flight tests of increasing difficulty are presented here. In each case, the vehicle traverses a 9m distance at a constant speed of $3^{\mathrm{m}}/_{\mathrm{s}}$ with different obstacle configurations requiring 3-D flight. The obstacles are not known *a priori* and are unobservable at takeoff. The Measurable Augmented Reality Cyber-Physical Systems [22] at MIT's Aerospace Control Laboratory was used to display the vehicle's sensing FOV in real-time, shown in red. Note the planner was allowed to violate the FOV constraint during start up to facilitate reaching top speed in a short distance.

a) Test 1: served as a baseline of **TIP**'s capabilities and is the easiest test case presented here. Fig. 10 shows a series of camera and data visualization stills. The red outline on the floor is representative the sensor's field FOV, Figs. 10a to 10c. The vehicle executes the planned trajectory, yellow, around the pillars and over the row of boxes, white, to the global goal, green Figs. 10d to 10f. The roll angle and roll rate are shown in Figs. 11a and 11b, respectively. Good tracking is achieved throughout and validates the triple integrator model assumption. A max roll angle of $39.5\mathrm{deg}$ and roll rate of $335.7^{\mathrm{deg}}/_{\mathrm{s}}$ was achieved. These values are far from the physical limits of the vehicle but are a good baseline.

b) Test 2: had a more complex obstacle configuration. Fig. 12 shows a series of camera Figs. 12a to 12c and data visualization Figs. 12d to 12f stills from the experiment. The roll angle and roll rate are shown in Figs. 13a and 13b, respectively. Good tracking is again achieved throughout. A max roll angle of $60.7\mathrm{deg}$ and roll rate of $515.5^{\mathrm{deg}}/_{\mathrm{s}}$ was achieved. The obstacles were placed to impede the planner's desired path, as inferred from *Test 1*. While these values are higher than those in *Test 1*, they are still not at the physical limits of the vehicle.

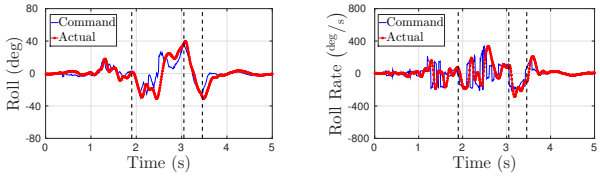
c) Test 3: was deemed the most difficult test environment because the obstacle configuration forced the vehicle to execute a number of aggressive maneuvers. Fig. 14 shows a series of camera Figs. 14a to 14c and data visualization Figs. 14d to 14f stills from the experiment. The roll angle and roll rate are shown in Figs. 15a and 15b, respectively. Again good tracking is achieved throughout. A max roll angle of $75.4\mathrm{deg}$ and roll rate of $695.7^{\mathrm{deg}}/_{\mathrm{s}}$ was achieved. These values are very close to the physical limitation of the vehicle and indicate the difficulty of the test environment. The number of aggressive maneuvers increased because the vehicle observed obstacles during a large attitude or fast angular rate maneuver. As such, maneuvers countering the current maneuver were required to avoid the newly seen obstacles.



(a) Camera $t = 1.90$s. (b) Camera $t = 3.05$s. (c) Camera $t = 3.46$s.

(d) Data $t = 1.90$s. (e) Data $t = 3.05$s. (f) Data $t = 3.46$s.

Fig. 10. *Test 1*: $3^{\mathrm{m}}/_{\mathrm{s}}$ flight in which the vehicle must swerve the pillars and fly over a row of boxes to reach the goal. The red projection represents the sensor's range and FOV. The green sphere is the global goal, the trajectory is shown in yellow, and the RBG coordinate system is representative of the vehicle's position and orientation.



(a) Roll angle. (b) Roll rate.

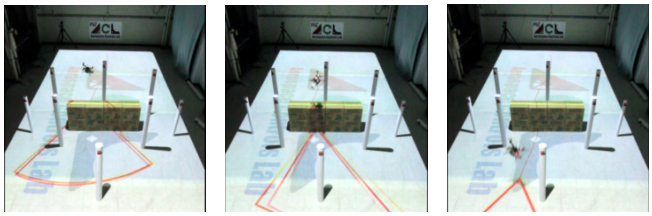
Fig. 11. *Test 1*: roll and roll rate tracking. (a): A peak roll angle of $39.5\mathrm{deg}$ was achieved during flight. (b): A peak roll rate of $335.7^{\mathrm{deg}}/_{\mathrm{s}}$ was achieved during flight.

VII. CONCLUSION AND FUTURE WORK

This work presented the development and experimental verification of a collision avoidance algorithm that is nearly an order of magnitude faster than the state-of-the-art and can execute aggressive collision avoidance maneuvers. The key property of the **T**riple **I**ntegrator **P**lanner is its ability to generate collision-free, dynamically feasible paths within 5ms (at worst) of receiving a point cloud. **TIP** successfully navigated complex environments requiring 3-D flight with Intel® RealSense™ R200. Future work includes incorporating "map-less memory" into the planner and determining its impact on performance.

VIII. ACKNOWLEDGMENTS

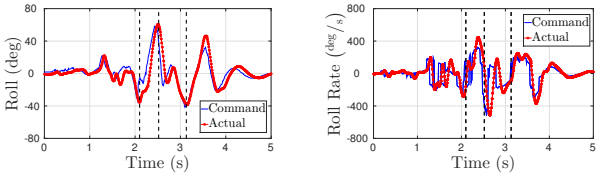
This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. 1122374 and by the DARPA Fast Lightweight



(a) Camera $t = 2.10$s. (b) Camera $t = 2.52$s. (c) Camera $t = 3.13$s.

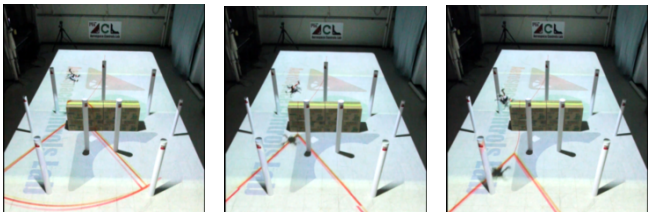
(d) Data $t = 2.10$s. (e) Data $t = 2.52$s. (f) Data $t = 3.13$s.

Fig. 12. *Test 2*: $3^{\mathrm{m}}/_{\mathrm{s}}$ through a more complicated environment. The obstacles were placed to impede the planner's desired path, as observed from *Test 1*.



(a) Roll angle. (b) Roll rate.

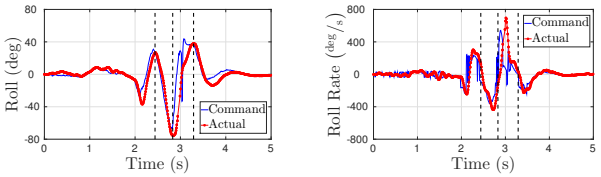
Fig. 13. *Test 2*: roll and roll rate tracking. (a): A peak roll angle of 60.7deg was achieved during flight. (b): A peak roll rate of $515.5^{\mathrm{deg}}/_{\mathrm{s}}$ was achieved during flight.



(a) Camera $t = 2.44$s. (b) Camera $t = 2.83$s. (c) Camera $t = 3.29$s.

(d) Data $t = 2.44$s. (e) Data $t = 2.83$s. (f) Data $t = 3.29$s.

Fig. 14. *Test 3*: $3^{\mathrm{m}}/_{\mathrm{s}}$ flight through the most difficult obstacle configuration. A number of aggressive maneuvers were executed because the vehicle observed obstacles during a large attitude or fast angular rate maneuver.



(a) Roll angle. (b) Roll rate.

Fig. 15. *Test 3*: roll and roll rate tracking. (a): A peak roll angle of 75.4deg was achieved during flight. (b): A peak roll rate of $695.7^{\mathrm{deg}}/_{\mathrm{s}}$ was achieved during flight.

Autonomy (FLA) program. The author would also like to acknowledge Boeing Research & Technology for support of the indoor flight facility and hardware.

References

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

[2] M. Burri, H. Oleynikova, M. W. Achtelik, and R. Siegwart, "Real-time visual-inertial mapping, re-localization and planning onboard mavs in unknown environments," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 1872–1878.

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

[4] D. Dey, K. S. Shankar, S. Zeng, R. Mehta, M. T. Agcayazi, C. Eriksen, S. Daftry, M. Hebert, and J. A. Bagnell, "Vision and learning for deliberative monocular cluttered flight," in *Field and Service Robotics*. Springer, 2016, pp. 391–409.

[5] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–4.

[6] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[7] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[8] 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*, p. 0278364915625345, 2016.

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

[10] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for quadrotor flight," in *International Conference on Robotics and Automation*, 2013.

[11] M. Hehn and R. DAndrea, "Quadrocopter trajectory generation and control," in *World Congress*, vol. 18, no. 1, 2011, pp. 1485–1491.

[12] M. Watterson and V. Kumar, "Safe receding horizon control for aggressive mav flight with limited range sensing," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 3235–3240.

[13] T. M. Howard, C. J. Green, A. Kelly, and D. Ferguson, "State space sampling of feasible motions for high-performance mobile robot navigation in complex environments," *Journal of Field Robotics*, vol. 25, no. 6-7, pp. 325–345, 2008.

[14] M. Pivtoraiko, D. Mellinger, and V. Kumar, "Incremental micro-uav motion replanning for exploring unknown environments," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 2452–2458.

[15] C. J. Green and A. Kelly, "Toward optimal sampling in the space of paths," in *In Proceedings of the International Symposium of Robotics Research*. Citeseer, 2007.

[16] 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.

[17] D. E. Kirk, *Optimal control theory: an introduction*. Courier Corporation, 2012.

[18] B. T. Lopez, "Low-latency trajectory planning for high-speed navigation in unknown environments," Master's thesis, Massachusetts Institute of Technology, 2016.

[19] R. F. Hartl, S. P. Sethi, and R. G. Vickson, "A survey of the maximum principles for optimal control problems with state constraints," *SIAM review*, vol. 37, no. 2, pp. 181–218, 1995.

[20] J. Lofberg, "Yalmip: A toolbox for modeling and optimization in matlab," in *Computer Aided Control Systems Design, 2004 IEEE International Symposium on*. IEEE, 2004, pp. 284–289.

[21] "Intel® RealSense™ Camera R200," https://software.intel.com/sites/default/files/managed/d7/a9/realsense-camera-r200-product-datasheet.pdf, 2016, online; accessed 27 August 2016.

[22] S. Omidshafiei, A. akbar Agha-mohammadi, Y. F. Chen, N. K. Ure, S.-Y. Liu, B. Lopez, J. How, J. Vian, and R. Surati, "MAR-CPS: Measurable Augmented Reality for Prototyping Cyber-Physical Systems," in *IEEE Control Systems Magazine*, 2016.