2014 International Conference on Unmanned Aircraft Systems (ICUAS)
May 27-30, 2014. Orlando, FL, USA

Path Planning Using 3D Dubins Curve for Unmanned Aerial Vehicles

Yucong Lin and Srikanth Saripalli

Abstract— **We present a path planning algorithm based on 3D Dubins Curves [1] for Unmanned Aerial Vehicles (UAVs) to avoid both static and moving obstacles. A variation of Rapidly-exploring Random Tree (RRT) [2] is used as the planner. In tree expansion, branches of the tree are generated by propagating along 3D Dubins Curves. The node sequence of shortest length together with Dubins curves connecting them is selected as the path. When the UAV executes the path, the path is checked for collision with updated obstacles' states. A new path is generated if the previous one is predicted to collide with obstacles. Such checking and replanning loop repeats until the UAV reaches the goal. The algorithm was validated through flight experiments using a small quadrotor UAV. In total, 6 flights to avoid static obstacles, 24 flights to avoid virtual moving obstacles and 20 flights to avoid real moving obstacles were performed. The efficacy of the algorithm was tested in office conditions and a parking structure. In all the situations our algorithm was able to reliably plan paths in real time and command the UAV to avoid obstacles.**

I. Introduction and Related Work

In the recently released Unmanned Aerial System (UAS) Integration Roadmap by Federal Aviation Association (FAA) [3], UAS is expected to expand its application to several civil domains such as commercial photography, cargo transport, critical infrastructure monitoring and disaster response. According to the document, "sense and avoid" capacity is one of the technological challenges to integrate UAS into National Airspace Systems (NAS). In "sense and avoid", an UAV should be able to maintain a self-separation and avoid collision with other obstacles.

One approach for obstacle avoidance was to control the UAV's heading to turn away from obstacles([4], [5], [6]). The desired yaw was calculated based on the relative geometry between the UAV and obstacles. Such methods required less computation power but could not guarantee optimality of the path. Other works tried to tackle aircraft collision avoidance by solving optimization problems. Control forces, path length and execution time were optimized while collision-free constraints needed to be observed. Different optimization methods were used, to name a few, Mixed Integer Linear Programming [7], Interior-Point Based Nonlinear Programming [8], and better-initial-guess plus gradient based optimization [9]. In the works, a simplified aircraft dynamic model was used and the algorithms were not verified in real flight tests. Optimal control was another approach for obstacle avoidance. Linear Quadratic Gaussian Control was used in [10] and Model Predictive Control in [11]. In them, a relatively accurate dynamic model of the vehicle was needed. The models were complex, not easily accessible and specific to a single type of vehicle. Several works [12], [13], [14] used Velocity Obstacle and its extensions for moving obstacle avoidance. They calculated velocities or control inputs of the robot that would induce collision and avoid such velocities or control inputs in action to prevent collision. However, they relied on a relatively accurate velocity measure of the obstacle. Partially Observable Markov Decision Process (POMDP) was used in [15], [16] to generate avoidance action. The action options were limited by the search space of large dimension.

A more fundamental problem—path planning in dynamic environment was studied in [17], [18], [19], [20] by sampling-based planning. Sampling-based planning had the advantages [21]: 1) able to find a feasible motion plan in relatively short time and guaranteeing probabilistic completeness. 2) applicable to very general dynamic models. 3) allowing trajectory-wise checking and therefore requiring no explicit enumeration of constraints. Probabilistic Roadmap was applied to path planning in state-time space in [17], [18]. A dimension of time was included to deal with moving obstacle. This resulted in a large sampling space. RRT handled systems with differential constraints effectively and was applied to autonomous driving in [20]. Different extensions to RRT were used in [22], [23] to avoid moving obstacles. The obstacles motion pattern was assumed to be represented by Gaussian Process and could be pre-learned. Multiple methods were proposed for UAV static obstacle avoidance, for example, Potential Field [24], Cross Entropy based on sampling in trajectory space [25] and alternate route via an expanding elliptical search [26]. It was yet to verify that they can be extended to moving obstacle avoidance.

This paper applies RRT-based path planning method to moving obstacle avoidance by UAV. The method works in real time, requires no accurate UAV dynamic model and is also applicable to static obstacle avoiance. In the method, nodes are connected by propagating along a 3D Dubins curve in tree expansion. Collision checking considers the evolution of the UAV and obstacles' states with time. At the start of path execution, the path is checked for collision with updated obstacle states. If it is predicted to result in collision, a new path will be generated. The rest of the paper is composed of problem formulation, algorithm description, experiment validation and extension to real scale problems.

II. Problem Formulation

The UAV's kinematic model is

$$\dot{\mathbf{y}}(t) = \mathbf{f}(\mathbf{y}(t), \mathbf{u}(t)),\ \mathbf{u}(t) \in U. \tag{1}$$

Yucong Lin and Srikanth Saripalli are with School of Earth and Space Exploration, Arizona State University, Tempe, AZ 85281, USA Yucong.Lin@asu.edu, Srikanth.Saripalli@asu.edu

978-1-4799-2376-2/14/$31.00 ©2014 IEEE

where $\mathbf{y} \in \mathbb{R}^{n_y}$ is the state of the system. $\mathbf{u} \in \mathbb{R}^{n_u}$ is the control input to it. U sets the control limit. The exact model used in this paper is described in experiment part (section IV).

To avoid obstacles, the UAV must observe the constraint:

$$\begin{aligned} &\text{for } i = 1, 2, ..N_o, \forall t \in [0, t_f], \\ &\|\mathbf{x}(t) - \mathbf{x}_{O_i}(t)\| \geqslant L_i \end{aligned} \tag{2}$$

where $\mathbf{x}_{O_i}(t)$ is the estimated position of obstacle O_i at t. It estimated by extrapolating from O_i's current position and current velocity:

$$\mathbf{x}_{O_i}(t) = \mathbf{x}_{O_i}(t_0) + \mathbf{v}(t_0) * (t - t_0) \tag{3}$$

For static obstacles $\mathbf{v}(t_0) = 0$. This indicates that the UAV must be able to maintain a distance above L_i to each obstacle O_i along the whole path. Other physical constraints include height limit and geofence. They are represented by

$$S(\mathbf{x}(t)) < 0 \tag{4}$$

Among all feasible paths, the one of the shortest length is selected. The optimization problem is

$$f^* = \underset{f}{\operatorname{argmin}} \int_f d\mathbf{s} \tag{5}$$

Where the path is represented by a function $f(x, y, z) = 0$. $\mathbf{x} \in \mathbb{R}^3$ is the 3D coordination along the path. The path starts from the UAV's initial position $\mathbf{x}_0$ and ends in the goal $\mathbf{x}_g$. $d\mathbf{s} = \sqrt{dx^2 + dy^2 + dz^2}$ is the curve length.

In summation, the path planning problem is defined as: Given the initial UAV state $\mathbf{y}(0) = \mathbf{y}_0$ and the control limit U, generate an optimal path $f^*(x, y, z) = 0$ defined by (5) that satisfies constraints in equations 1, 2 and 4.

III. Method Description

This section describes 1) RRT tree expansion; 2) the loop of execution.

A. Tree Expansion

A tree node is defined as

$$\text{node} = \{\mathbf{y}, \textit{cost}, \textit{cost-to-go}, \textit{goal\_reach}\} \tag{6}$$

$\mathbf{y}$ is the UAV's state. *Cost* is the length of trajectory from the tree root to the node and *cost-to-goal* from the node to the goal. *Goal_reach* is the flag indicating if the node is able to reach the goal. The state is defined as:

$$\mathbf{y} = (x, y, z, \psi, v_x, v_y, v_z) \tag{7}$$

It contains the UAV's position, velocity, yaw (ψ). For convenience, the paper represents a node q with the first four components (x_q, y_q, z_q, ψ_q) when the rest components need not be mentioned. Node connection is the process to generate a trajectory connecting two nodes or return that not one is available. 3D Dubins curve is used in node connection because: 1) it allows to assign initial and final heading of the UAV as well as position. 2) it is the shortest curve that connects two points with a constraint on the curvature (determined by UAV's turning radius) of the path and with prescribed initial and terminal headings. To connect node (x_0, y_0, z_0, ψ_0) to (x_1, y_1, z_1, ψ_1), a 2D Dubins curve C is first created from (x_0, y_0, ψ_0) to (x_1, y_1, ψ_1) as in [27]. It is then extended to 3D by assigning $z = z_0 + l(x, y)/l(x_1, y_1) * (z_1 - z_0)$ to each (x, y) in C, where $l(x, y)$ stands for the length along C from (x_0, y_0) to (x, y). The next step is to propagate the UAV's kinematic model along the 3D Dubins curve until reaching the end or being blocked by obstacles. If the end is reached, the propagated trajectory is the connection between the two nodes. Otherwise a connection does not exist due to collision with obstacles. Figure 1 shows the propagated trajectory (black dots) along the Dubins curve (blue) connecting $n_0 = (0, 0, 0.5, 0)$ and $n_1 = (4, 2, 2, 0)$. An approximated kinematic model of Parrot ARDrone was used. The model is described in section IV.

Here the trajectory from the propagation instead of the original Dubins curve is used as the node connection because the model for propagation considers the UAV's kinematics property and the propagated trajectory is able to approximate the UAV's real trajectory.

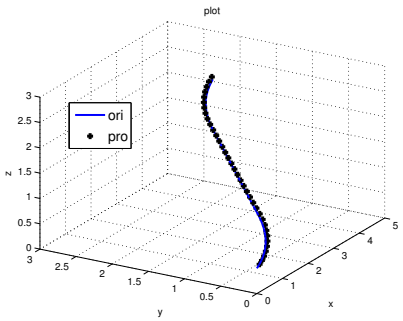


Fig. 1: To propagate along a 3D Dubins curve using an approximated kinematic model of an ARDrone. Blue line is the Dubins curve and black dots are propagated trajectory. The Dubins curve starts at $(0, 0, 0.5)$ and ends at $(4, 2, 2)$. Both the starting and ending yaw are 0.

In each loop of tree expansion, a sample is taken as in [28] and nodes in the tree are sorted ascendantly according to heuristics. The heuristics is based on the length of 3D Dubins curve between each node and the sample. In sorted order each node is connected to the sample. If the connection is blocked by obstacles, the trajectory is not abandoned if its length exceeds half length of the 3D Dubins curve. Intermediate nodes[1] of the trajectory are added to the tree. This strategy increases the tree density to make sure the probability of finding a feasible path. When the trajectory is fully collision-free, end and intermediate nodes are both added to the tree. When a node is added to the tree, its cost is set as the trajectory length from the root to it. A connection between the node and the goal is attempted to build. If a connection exists, cost-to-go of the node is set as the trajectory length from the node to the goal. To generate the path, the goal-reachable node $q*$ with the smallest sum of cost and cost-to-go is selected. The node sequence connecting the root to the goal via $q*$ along with Dubins curves between the nodes is the generated path. Figure 2 shows the grown tree (yellow branches) with only 2 samples and the generated path (red). A static obstacle is used.

[1] An intermediate is acquired node every N_{int} propagating steps. In this work, N_{int}= 5.

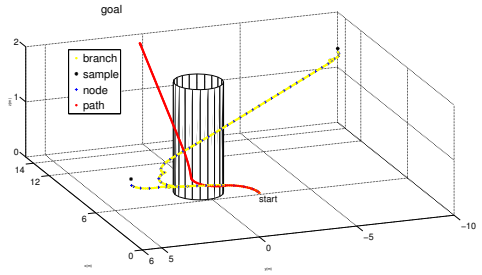


Fig. 2: Tree expansion with only 2 samples in a single static obstacle situation. Tree branches are trajectories propagated along Dubins curve (yellow). End and intermediate nodes are also added as nodes (blue cross). The red curve shows the generated path

B. Loop of Execution

The loop of execution uses a repetitive path replanning strategy to compensate the inaccuracy of the UAV's kinematic model and that of the prediction of obstacles' motion. It is summarized in algorithm 1.

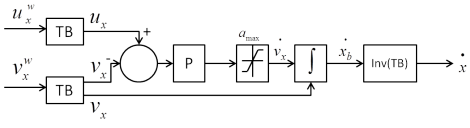
Algorithm 1 Execution Loop()

```
1: repeat
2:     Update the UAV's state and obstacles' state
3:     Estimate UAV's state after Δt and use it as the tree root.
4:     for Δt do
5:         TreeExpand()
6:     end for
7:     PathGeneration()
8:     if No path then
9:         if Obstacles are close then
10:            Use local path planner to avoid
11:        else
12:            Goto the line 2
13:        end if
14:    end if
15:    Update the UAV's state and obstacles' state
16:    Check if the generated path is still collision-free
17:    if not then
18:        Remove the blocked nodes and their children from the tree
19:        Goto line 7
20:    end if
21:    Start from t_1, execute the path for Δt;
22:    At t_1, update the UAV's state and obstacles' state
23:    In parallel, predict the UAV's state after Δt, obtaining y(t_1+Δt). Check
       if the path is still collision free if starting from y(t_1 + Δt)
24:    if not then
25:        Goto line 2.
26:    else
27:        Goto line 21.
28:    end if
29: until Reach the goal
```

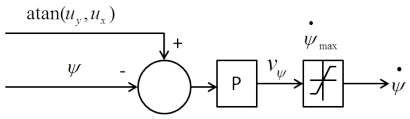
In an execution loop, updated states of the UAV and obstacles are first acquired (line 2). The UAV's state after Δt is estimated by propagating the kinematic model along previous planned path. It is used as the tree root (line 3). The tree is expanded for Δt and the path is generated (line 4 to 7). If a path is not available (line 8 to 14), a tree is rebuilt with updated UAV's and obstacles' states and path is re-generated if the obstacles are far away. Otherwise, the UAV needs to avoid the obstacles reactively. The path from line 7 won't be executed immediately but will be checked for collision with updated UAV's and obstacles' states (line 15 and 16). If there is collision, the blocked nodes with their children will be removed from the tree. After that, a new path is generated (line 17 to 20). These steps correct the path based on updated UAV's and obstacles' states to compensate inaccurate prediction of UAV's and obstacles' states in planning. Next, the adjusted path is executed by the UAV for Δt. At the start of execution, the states of the UAV and obstacles are updated. The UAV's state is used to obtain the predicted state after Δt. The predicted state is used together with obstacles' states to check if the path is still collision-free after Δt. If not, a new path needs to be created (line 25). Otherwise, the same path is followed (line 27).

IV. Experiments and Results

Multiple flight experiments were conducted to validate our algorithm. An off-the-shelf Parrot ARDrone was used as the UAV platform. An optical flow sensor is used for velocity estimation and odometry is used as the position system. An ARDrone receives commands from a computer and transfers its onboard sensor readings to it. The communication uses wifi. Technical details of ARDrones can be found in [29].

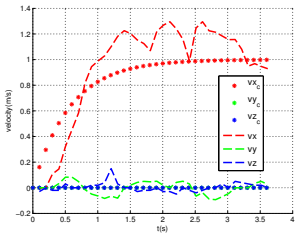


(a)

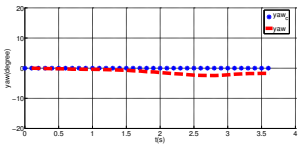


(b)

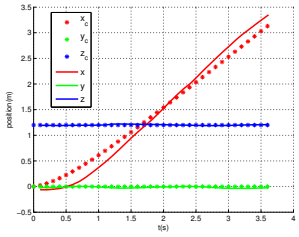
Fig. 3: Approximated kinematic model of the ARDrone. (a) describes the model for x and the same for y and z. As in the figure, velocity v_x and velocity command u_x are transformed into the body reference frame (TB). The error between them is used as the input to a P-controller. The output of the controller is thresholded by ARDrone's acceleration of saturation a_{max} to obtain the desired acceleration $\dot{v_x}$. The velocity output of the model $\dot{x}$ is calculated by integrating $\dot{v_x}$ and transforming back to the world reference frame (Inv(TB)). As in (b), the desired yaw speed v_Ψ is obtained with another P-controller. The input is the error between $atan(u_y, u_x)$ and the current yaw Ψ. v_Ψ thresholded by the saturation value $\dot{\Psi}_{max}$ gives the yaw rate $\dot{\Psi}$.



(a)



(b)



(c)

Fig. 4: Comparison of kinematic model and a real ARDrone. (a) compares the velocity, (b) compares the yaw, and (c) compares the position. Variables with the subscript "c" mean data from the model. Variables without subscripts come from a real ARDrone.

Figure 3 illustrates the approximated kinematic model of ARDrones used for tree expansion. Figure 3(a) describes the model for x and the same for y and z. As in the

figure, velocity v_x and velocity command u_x are transformed into the body reference frame (TB). The error between them is used as the input to a P-controller. The output of the controller is thresholded by ARDrone's acceleration of saturation a_{max} to obtain the desired acceleration $\dot{v_x}$. The velocity output of the model $\dot{x}$ is calculated by integrating $\dot{v_x}$ and transforming back to the world reference frame (Inv(TB)). As in figure 3(b), the desired yaw speed v_ψ is obtained with another P-controller. The input is the error between $atan(u_y, u_x)$ and the current yaw ψ. v_ψ thresholded by the saturation value $\dot{\psi}_{max}$ gives the yaw rate $\dot{\psi}$. In the model, the yaw of the UAV is controlled to align with that of the velocity command. The ARDrone was controlled in the same way in flight experiments. This way the ARDrone flew like a fixed-wing aircraft and therefore it could be considered as a proxy of a fixed-wing UAV. To compare the model with ARDrone's actual behavior, both a real ARDrone and the model were commanded to fly under the command $\mathbf{u} = (1, 0, 0)$ for 3.5 seconds and their states v.s. time are plotted in figure 4.

It is observed in figure 4(a) that the velocity v_x of the model and a real UAV has accordance in evolving trend and the maximal error is around $0.2m/s$. A similar pattern is observed for x position in figure 4(c). Yaw of the model and a real UAV almost overlap in figure 4(b).

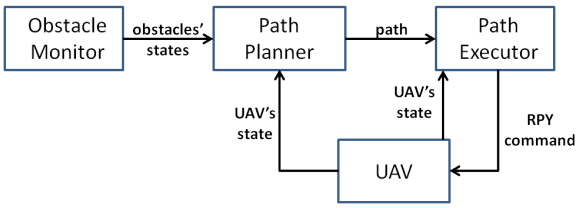


Fig. 5: ROS nodes to control the ARDrone in the experiments.

(a) (b)

Fig. 6: Two experiment situations: (a) avoid a static obstacle in an office; (b) avoid another ARDrone in a parking structure.

Three situations were designed for experiments: 1) avoid static obstacles in office condition; 2) avoid virtual moving obstacles; 3) avoid actual moving obstacles. 2) and 3) were carried out in a parking structure. The UAV was controlled using *Robot Operating System* (ROS). Figure 5 depicts the ROS nodes. The path planner node generates the path and sends to the path executor node. The latter creates the velocity command based on the UAV's current position and the path using vector field method [1]. The velocity command is transfered to roll-pitch-yaw command and sent to the UAV. The planner node also receives the UAV's state for path planning. It holds an obstacle monitor to receive updated obstacles' states. In office experiments, the position of static obstacles were set initially known to the UAV and didn't need to update. For virtual obstacles, the monitor read states from the logged trajectory of virtual obstacles at each time instant. In actual moving obstacle experiments, the monitor received states broadcasted by the obstacle.

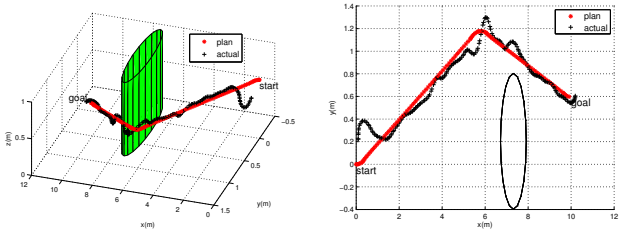
In all three situations, the origin of the coordinate system was set to the position of take-off location. The direction of x was set to the UAVs yaw direction the moment it finished take-off. This defined the global reference frame. Meter was used as the length unit. Δt in algorithm 1 was set to 1s.

A. *Flight Experiments in Office*

The office had a pillar in the center as the obstacle and table cubes around as the geofence. The UAV need avoid all of them. The size and locations of the pillar and cubes were known to the path planner. The start was $(0, 0, 0.8)$ and the goal was $(10, 0.6, 0.5)$. The pillar located at $(7.31, 0.2)$ and its safety radius was set to 0.6 meter according to the pillar's size. The four corners of the geofence were set to $(-1.5, 0)$, $(1.2, 0)$, $(-1.5, 11.58)$ and $(1.2, 11.58)$ based on the office's area.

flight#	1	2	3	4	5	6	avg
min_dis(m)	0.83	0.74	0.94	0.92	0.95	0.83	0.87

TABLE I: The closest distance between the UAV and the pillar in six flights in the office. The average of them is also presented.



(a) (b)

Fig. 7: The logged trajectory of one flight in the office. The red is the planned path and the black is the executed trajectory. (a) is from a 3D perspective and (b) from a top view. The green cylinder represents the pillar and its radius is 0.6m.

6 successful flights and 5 failing ones were performed. Failure was caused by the inaccuracy of position system. The logged data showed that the UAV's trajectory were clear from the pillar and cubes but actually the UAV collided with them. The magnetometer for yaw measurement was noisy and made the odometry drift from the real position. We didn't make efforts to improve the position system because localization was not the focus of this paper and a less accurate position system suffice to demonstrate the planning algorithm's efficacy. The minimum distance between the UAV and the pillar in the 6 successful flights are displayed in table I. They are all above the 0.6m safety threshold, showing successful avoidance. Logged trajectory of one flight of them is plotted in figure 7. It is observed in the figure that the UAV followed the path even though there were drifts. The difference of the UAV's start from the path's start was due to the drift when taking off.

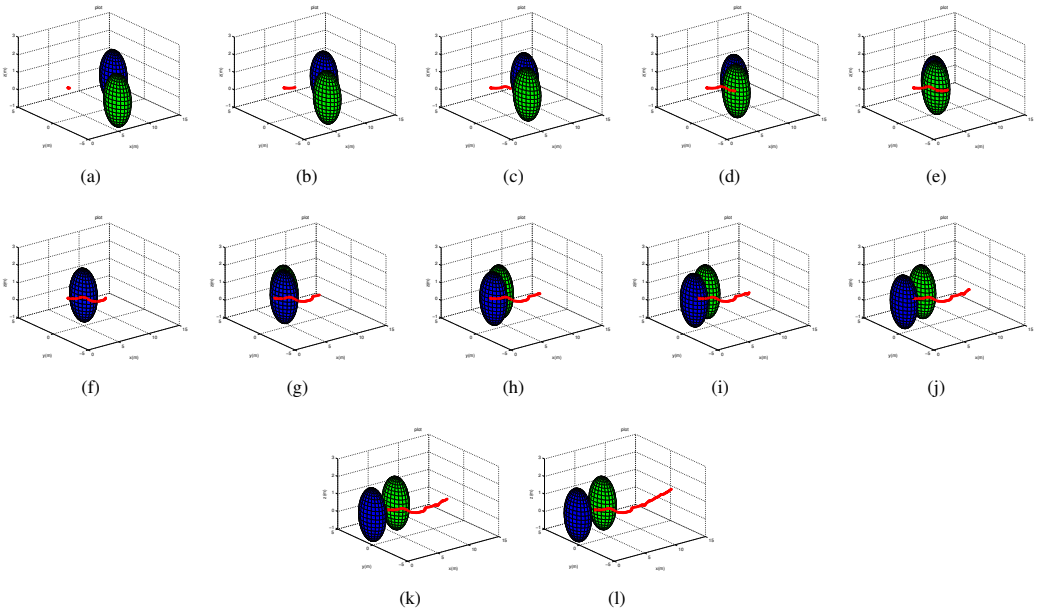


Fig. 8: The logged trajectory of the ARDrone (red) at different time instants to avoid two virtual obstacles. The green obstacle flew perpendicular to the original route of the ARDrone and the blue obstacle flew opposite to the original route. Both obstacles have a safety radius of 1.5m.

B. Avoiding Virtual Moving Obstacles

Virtual Obstacles were logged trajectories of a quadrotor flying in simulation. *hector_quadrotor*[2] ROS package was used to create them. For example, a virtual obstacle flying towards the UAV can be created by flying the quadrotor in simulation from $(0, 0, 0.75)$ to $(10, 0, 0.75)$ by velocity command $v = (-1, 0, 0)$. The command was constant but the velocity of the obstacle was not. In all experiments, the UAV took off from $(0, 0, 0)$ and tried to fly to $(15, 0, 0.8)$. The length unit is meter.

Figure 8 shows the trajectory of the ARDrone (red) avoiding two virtual obstacles (blue and green) at different time instants. Their radii stand for safety radii. The blue obstacle flew towards the ARDrone from $(10, 0, 0.75)$ to $(0, 0, 0.75)$ under the command $v = (-1, 0, 0)$ and the green one from $(5, -5, 0.75)$ to $(5, 1, 0.75)$ under $v = (0, 1, 0)$. In the whole procedure, the UAV stayed outside the two spheres, indicating successful avoidance. More flight experiments were carried out with different virtual obstacles and combinations of them. The speed of the UAV was set to 1m/s. The closest distances between the ARDrone and the obstacles were showed in table II. In the table, vkx10 means that the obstacle was created to move from $(10, 0, 0.75)$ to $(0, 0, 0.75)$ under command $v = (-k, 0, 0)$. It flew to the ARDrone head-to-head. v1y-k means that the obstacle started from $(k, -k, 0.75)$ and stopped at $(k, 1, 0.75)$, representing an obstacle approaching from the lateral direction. In row 1 to 4, the UAV avoided obstacles flying towards the UAV at different speeds. It is observed that the UAV was able to avoid obstacles of speeds up to 3m/s under the given take-off and goal position. Row 5 shows the results to avoid only a perpendicularly moving obstacle. The UAV was able to stay well clear of it during the whole trajectory in all 3 flights. In the next 3 rows, the UAV tried to avoid two obstacles. One moved head-to-head and another perpendicularly. The latter started from different locations. For each combination, the first row shows the smallest distance between the UAV and the first obstacle. The second row shows that between the UAV and the second. For all 3 combinations, the UAV maintained separations to two obstacles larger than safety threshold 1.5m in every flight. Therefore, the algorithm was validated in two-obstacle situations.

[2]http://wiki.ros.org/hector_quadrotor

row#	virtual obs	1	2	3	avg
1	v1x10	2.97	2.78	1.89	2.55
2	v2x10	2.12	1.69	1.81	1.87
3	v3x10	1.97	1.86	2.21	2.01
4	v4x10	1.34	1.39	1.44	1.39
5	v1y-7	3.71	3.74	3.52	3.66
6	v1x10&v1y-5	1.95 1.71	2.62 2.42	2.89 2.04	2.49 2.06
7	v1x10&v1y-6	2.71 1.54	1.80 1.61	2.65 1.72	2.39 1.62
8	v1x10&v1y-7	2.97 1.63	2.76 1.68	2.47 1.58	2.73 1.63

TABLE II: The minimum distances (m) between the UAV and virtual obstacles in multiple experiments. vkx10 means that the obstacle was created to move from $(10, 0, 0.75)$ to $(0, 0, 0.75)$ under command $v = (-k, 0, 0)$. It flew to the ARDrone head-to-head. v1y-k means that the obstacle started from $(k, -k, 0.75)$ and stopped at $(k, 1, 0.75)$, representing an obstacle approaching from the side. The unit is meter. Rows 1-5 are situations involving only one obstacle and rows 6-8 involve two obstacles. There flights were conducted for each type of obstacles. The average minimum distances are also presented.

The efficacy of replanning was demonstrated in the procedure to avoid the obstacle v1y-7 as in figure 9. The obstacle's speed was close to zero when the UAV took off. The path generated by the planner was a straight line to the goal (figure 9(a)). Later the obstacle speeded up and the straight line path would result in collision. Therefore, a new path was created (figure 9(b)) with the UAV's current position as the start. The UAV was able to avoid the obstacle by following the new path until reaching the goal (figure 9(c) and 9(d)).

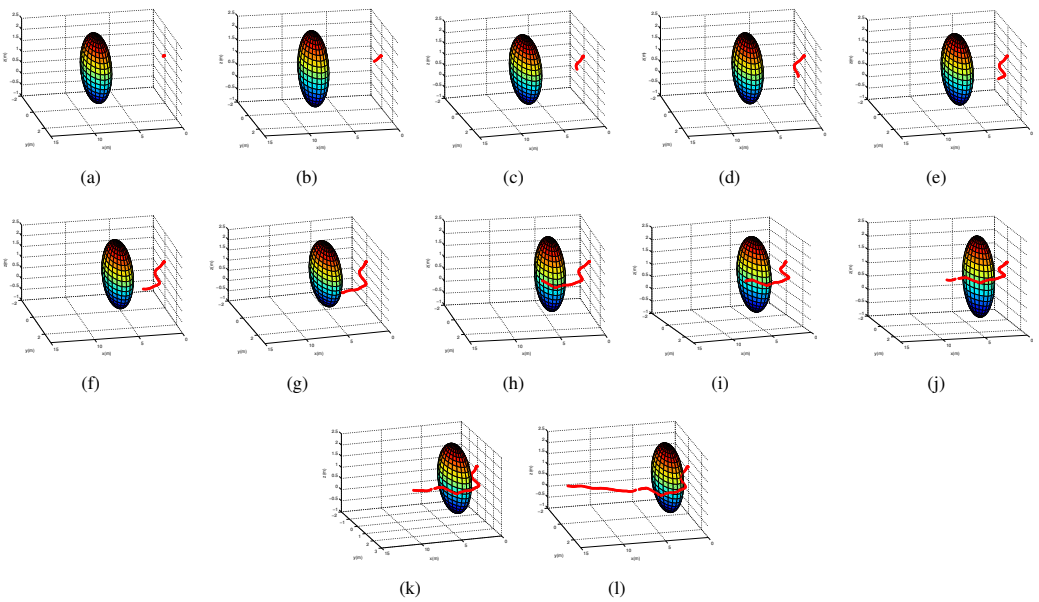


Fig. 10: The logged trajectory of the UAV (the red dot curve) at different time instants to avoid a real moving obstacle. The sphere represents the logged obstacle's position at each instant. Its radius represents the obstacle's safety radius and is 1.5m. In all the figures, we observe that the UAV (left end of the red curve) always stays outside the sphere, indicating the success of avoidance.

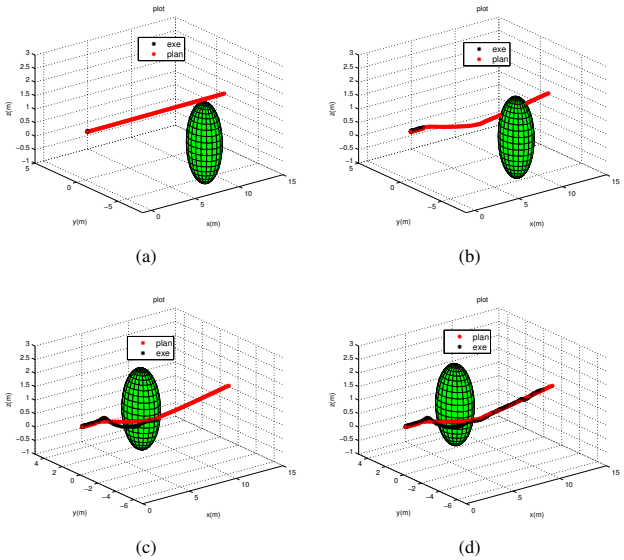


Fig. 9: A replan example: in (a) a straight path (red) would not cause collision due to the small initial speed of the obstacle (green sphere of 1.5m radius). The obstacle however speeded up in (b) and the planner abandoned the previous straight path and created a new path to avoid the obstacle. The black dots mean the executed trajectory when following the path.

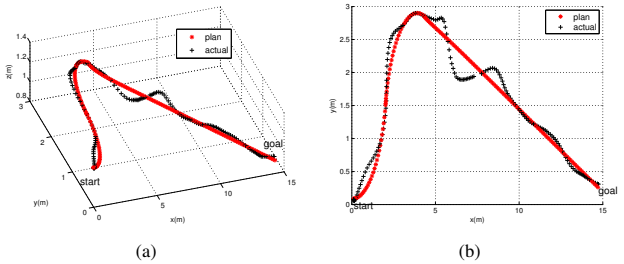


Fig. 11: The logged trajectory (black) and planned path (red) of the flight in figure 10.

C. Avoiding an Actual Moving Obstacle

Another ARDrone was used as the moving obstacle (figure 6(b)). It was controlled by a separate laptop. Its states were transmitted to the laptop through Wifi. They were next transmitted to the laptop controlling the host ARDrone through ethernet. This way the planner knew the states of the moving obstacle. The obstacle ARDrone flew towards the host UAV's taking-off position by following a straight line. The host ARDrone needed to avoid it and reach the goal. In the experiment, both ARDrones first took off and started to hover. When both of them were in stable hover, the obstacle ARDrone started to fly towards the host one and the host one started the execution loop in algorithm 1.

Figure 10 demonstrated a full avoidance procedure. Each figure shows a different time instant. The red is the logged trajectory of the UAV and the sphere is the obstacle. The end of the trajectory stands for the position of the UAV at the same moment as the obstacle. The obstacle started at $(7.753, -0.145, 0.735)$ and stopped at $(2.43, -0.04, 0.71)$. The UAV started from $(0.11, 0.08, 1.11)$ and stopped at $(14.74, 0.3, 0.85)$. The obstacle was commanded to fly at 0.5m/s and the UAV at 1m/s. In the figures, the UAV never enter the sphere whose radius is the safety distance (1.5m in our case). This proves that UAV successfully avoided the obstacle. The full trajectory for the same flight is plotted with the planned path in figure 11.

flight#	1	2	3	4	5	6	7	8	9	10	avg
v:1.0	2.58	1.74	2.65	2.29	2.28	2.11	3.45	2.33	2.41	1.63	2.35
v:0.5	2.55	1.99	2.22	1.98	1.58	1.44	1.90	1.99	2.37	3.83	2.19

TABLE III: The closest distance (in meter) between the UAV and the moving obstacle. The speed of the obstacle is 0.5m/s and that of the UAV is 0.5m/s or 1.0m/s. We carried out 10 flights for each speed. Only the minimum distance at 6th flight of 0.5m/s is slightly below the 1.5m safe separation threshold. All the rest flights successfully prove the efficacy of collision avoidance. The last column is the average minimum distance.

Multiple flights were conducted with the obstacle's speed at 0.5m/s and the UAV's speed at 0.5m/s or 1.0m/s. In

all of them, the obstacle ARDrone took off at $(7.5, 0, 0)$ and fly along a straight line until reaching $(-2, 0, 0.75)$. Table III lists the smallest distance between the obstacle and the UAV in all flights. From the table, the smallest distance of only one flight was slightly below the safety threshold (1.5m). This proves the efficacy of the algorithm to avoid a real moving obstacle. Flight videos are available at http://www.youtube.com/watch?v=ggS5-x0rjNM. Efficacy of the algorithm in avoiding virtual and actual moving obstacles were examined using logged position data based on optical flow. The logged position had similar drifts as in office condition although no physical collision occurred. In the absense of more accurate position systems such as motion capturing system, optical flow was used for positioning both the UAV and obstacles. The experiments were made as proxy to obstacle avoidance with positioning system of higher accuracy such as GPS.

D. Insights from Experiments

In the experiments, moving obstacles followed straight lines. It is fitted by equation 3. However, equation 3 will fail to predict the motion of obstacles that moves along a curve. In this case, the obstacle's heading needs to be considered in prediction. It is found in the virtual obstacles experiment that the algorithm was unable to deal with obstacles faster than a certain speed given a fixed initial distance between the UAV and the obstacle. A local planner to generate reactive avoidance motion was needed. Implementing and integrating the local planner is one of future works. From the experiment in the office, we observed that drifts of the odometry caused collision with obstacles. This implied that the algorithm will fail with low accuracy of position systems.

V. Extension to Real Scale Aircrafts

In our experiments, a small quadrotor was used and therefore the scale for distances ($\sim$ 20m) and velocities ($\sim$ 1m/s) were relatively small. Such scales are far away from that in National Airspace System. Therefore, this section discusses the possibility to apply the algorithm to problems of increased scale.

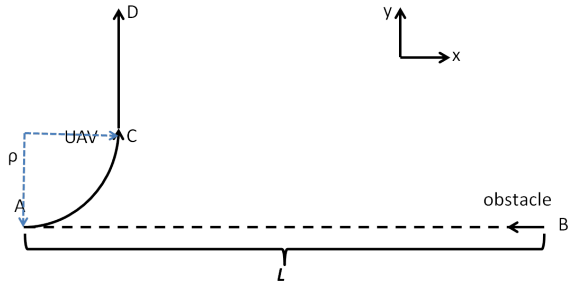


Fig. 12: The figure to demonstrate a simplified 2D moving obstacle avoidance case.

First analyze the relation between the minimum distance to detect the moving obstacle L, the speed of the UAV v_1 and the speed of the obstacle v_{obs}. Refer to figure 12, a simplified 2D case is studied: a UAV starts from $A(0, 0)$ with velocity $(v_1, 0)$ and detects an obstacle at $B(L, 0)$ of the velocity $(-v_{obs}, 0)$ at time $t = 0$. We need to calculate the minimum allowable value of L with which the UAV is able to avoid the obstacle. Assuming that the UAV has a full yaw rate ω and therefore the smallest turning radius $\rho = \omega v_1$. A full avoidance maneuver is first turning left for 90 degrees and fly forward and then turn back to the goal. According to the figure, the UAV should first move along the quarter circle AC then along the straight line CD. The position of the obstacle as a function of t can be expressed as

$$(x_{obs}, y_{obs}) = (L - v_{obs}t, 0) \tag{8}$$

When the UAV is on curve AC, its position is

$$(x_u, y_u) = (\rho sin(\omega t), \rho(1 - cos(\omega t))) \tag{9}$$

Therefore the square of distance between the UAV and the obstacle is

$$f(t) = (L - v_{obs}t - \rho sin(\omega t))^2 + \rho^2(1 - cos(\omega t))^2, \quad \text{where } t \in [0, \frac{\pi}{2\omega}) \tag{10}$$

Suppose that the safety radius is s, a successful avoidance requires

$$f(t) > s^2, \ \forall t \in [0, \tfrac{\pi}{2\omega}) \tag{11}$$

this is equivalent to

$$\min f(t) > s^2, \ t \in [0, \tfrac{\pi}{2\omega}) \tag{12}$$

To calculate the minimum value of $f(t)$ via its first order derivative requires solving a transcendental equation. Therefore, we turn to solve a less strict problem by requiring that

$$(L - v_{obs}t - \rho sin(\omega t))^2 > s^2, \ \forall t \in [0, \tfrac{\pi}{2\omega}) \tag{13}$$

This an sufficient but unnecessary condition of (11). Solving it, we have

$$\forall t \in [0, \frac{\pi}{2\omega}), \ L > s + v_{obs}t + \rho sin(\omega t) \tag{14}$$

$$\text{or } L < -s + v_{obs}t + \rho sin(\omega t) \tag{15}$$

Since $v_{obs}t + \rho sin(\omega t)$ increases monotonically in $[0, \frac{\pi}{2\omega})$, we have

$$L > s + \rho + \frac{v_{obs}\pi}{2\omega} \tag{16}$$

$$\text{or } L < -s \tag{17}$$

(17) obviously does not hold. Therefore, a sufficient but unnecessary condition for the UAV avoid the obstacle when it follows curve AC is

$$L > s + \rho + \frac{v_{obs}\pi}{2\omega} \tag{18}$$

There is another condition when the UAV follows the straight line DC, it is not listed here because (18) is sufficient to analyze how scale affects the safe-avoidance condition. In 18, the inequity still holds when multiplying L, s, v_{obs} by the same scale, assuming a fixed ω. This means, for an UAV and obstacles with larger speeds, the scale of the distance for obstacle detection and the safety radius must also be increased based on the speed.

Specifications of a MH-6/AH-6 little bird helicopter is used to test the path planning algorithm for increased scale problem. Its cruise speed is 69.45m/s, maximum speed is 78.19m/s and climb is 10.5m/s. In the path planning problem, the helicopter needs fly from $(0, 0, 55.2)$ to $(966, 41.4, 55.2)$

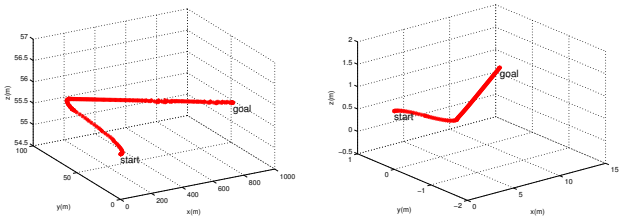
and avoid an obstacle moving with speed 69.45m/s from $(486.15, 0, 48.62)$ to $(0, 0, 48.62)$ with a safety radius 69.45. Meter is the unit. To compare, we assume a small UAV whose specifications differ from that of little bird up to a scale of 69.45. That means it has a cruise speed of 1m/s, the maximum speed 1.13m/s and climb 0.15m/s. It will fly from $(0, 0, 0.8)$ to $(14, 0.6, 0.8)$ and avoid an obstacle moving with speed 1m/s from $(7, 0, 0.7)$ to $(0, 0, 0.7)$ with a safety radius 1. In both cases, the time for tree expansion $\Delta t = 1.0s$. The kinematic model in experiment section is used but the gains for the real scale helicopter are lowered to avoid overshooting. The nodes number of the expanded tree and the length of paths generated from the tree are shown in table IV. The algorithm was run for three times. For comparison, the same data for the small UAV case are listed in table V. Column 4 of the table shows the lengths multiplied by the scale. They are close to column 3 of table IV. This is one of the evidences that the path planning algorithm works for different scale problems. Comparing column 2 of two tables, it is observed that the real scale problem generates less nodes in the tree. This is because a lower control gain is used and the step size to propagate along a Dubins curve changes less than the scale. Given that the length of a Dubins curve change in scale, propagating it costs more time. It does provide more intermediate nodes for each branch but the allowed number of samples and therefore number of Dubin curves to propagate is reduced. The net effect was the reduced number of nodes. Figure 13 shows the path generated for the two problem of different scales. This gives another evidence of our algorithm's adaptability to scales.

run#	nodes	length(m)
1	692	1076.4
2	570	1017.22
3	589	1011.91

TABLE IV: Path planning for a real scale helicopter. Column 2 lists number of tree nodes after tree expansion after 1s and the lengths of generated paths are shown in column 3. We ran path planning algorithm for 3 times.

run#	nodes	length(m)	length compare(m)
1	928	14.27	990.94
2	936	14.233	988.48
3	898	14.233	988.48

TABLE V: Path planning for a small UAV. Number of tree nodes after tree expansion after 1s (column 2) and the length of generated path (column 3). Column 4 lists the length in column 3 multiplied by the scale. We ran path planning algorithm for 3 times.



(a) (b)

Fig. 13: Generated path for a real scale problem (a) and a small scale problem (b).

VI. Conclusion and Discussions

We developed a path planning method for UAVs to avoid both static and moving obstacles. 3D Dubins curve is used to initialized node connection in the RRT tree. The tree is expanded by propagating the UAV's kinematic model along the Dubins curve while checking collisions. The node sequence from the root to the goal with the smallest length as well as Dubins curves connecting them are the generated path. The path is checked with updated states of the UAV and obstacles while being executed by the UAVs. A new path is created if the previous one is estimated to end in collision. Such checking and replanning loop makes sure that UAV's inaccurate kinematics model and approximated prediction of obstacles motion can be compensated. The algorithm is validated by multiple flight experiments to avoid static obstacles, virtual and real moving obstacles.

We only validated the algorithm in indoor and less-open outdoor environment. Optical flow based odometry was used as the position system and a quadrotor was used as the proxy of a fix-wing UAV. In the next step, we are going to use a fixed-wing UAV flying in open outdoor environment to further validate the algorithm. GPS will be used as the position system. This paper only considers neutrual moving obstacles. A more difficult case is to avoid obstacles that are also able to manoeurve to avoid collision. To deal with such reciprocal avoidance problem, common rules of aircraft behavior need to be introduced into path planning. For example, both aircraft manoeurve to the right when avoidance. This paper assumes that obstacles' states are available to the UAV and active sensing is not needed. To deal with general obstacles, our algorithm needs to be combined with onboard obstacle detection to achieve full "sense and avoid".

References

[1] R. W. Beard and T. W. McLain, "Implementing dubins airplane paths on fixed-wing uavs," *Contributed Chapter to the Springer Handbook for Unmanned Aerial Vehicles*, 2013.

[2] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[3] FAA, "Integration of civil unmanned aircraft systems (uas) in the national airspace system(nas) roadmap," http://www.faa.gov/about/initiatives/uas/media/UAS_Roadmap_2013.pdf, 2013.

[4] X. Yang, L. M. Alvarez, and T. Bruggemann, "A 3d collision avoidance strategy for uavs in a non-cooperative environment," *Journal of Intelligent & Robotic Systems*, vol. 70, no. 1-4, pp. 315–327, 2013.

[5] L. Mejias and I. F. M. P. Campoy, "Omnidirectional bearing-only see-and-avoid for small aerial robots," in *Automation, Robotics and Applications (ICARA), 2011 5th International Conference on*. IEEE, 2011, pp. 23–28.

[6] M. Melega, S. Lazarus, and A. Savvaris, "Autonomous collision avoidance based on aircraft performances estimation," in *Digital Avionics Systems Conference (DASC), 2011 IEEE/AIAA 30th*. IEEE, 2011, pp. 5B4–1.

[7] A. Richards and J. P. How, "Aircraft trajectory planning with collision avoidance using mixed integer linear programming," in *American Control Conference, 2002. Proceedings of the 2002*, vol. 3. IEEE, 2002, pp. 1936–1941.

[8] R. B. Patel, P. J. Goulart, and V. Serghides, "Real-time trajectory generation for aircraft avoidance maneuvers," in *AIAA Guidance, Navigation and Control Conference, no. AIAA*, vol. 5623, 2009.

[9] C.-K. Lai, M. Lone, P. Thomas, J. Whidborne, and A. Cooke, "On-board trajectory generation for collision avoidance in unmanned aerial vehicles," in *Aerospace Conference, 2011 IEEE*. IEEE, 2011, pp. 1–14.

[10] D. Bareiss and J. van den Berg, "Reciprocal collision avoidance for robots with linear dynamics using lqr-obstacles," in *IEEE Int. Conf. Robotics and Automation*, 2013.

[11] D. H. Shim and S. Sastry, "An evasive maneuvering algorithm for uavs in see-and-avoid situations," in *American Control Conference, 2007. ACC'07*. IEEE, 2007, pp. 3886–3891.

[12] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *The International Journal of Robotics Research*, vol. 17, no. 7, pp. 760–772, 1998.

[13] F. Large, C. Laugier, and Z. Shiller, "Navigation among moving obstacles using the nlvo: Principles and applications to intelligent vehicles," *Autonomous Robots*, vol. 19, no. 2, pp. 159–171, 2005.

[14] J. van den Berg, D. Wilkie, S. J. Guy, M. Niethammer, and D. Manocha, "Lqg-obstacles: Feedback control with collision avoidance for mobile robots with motion and sensing uncertainty," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 346–353.

[15] T. B. Wolf and M. J. Kochenderfer, "Aircraft collision avoidance using monte carlo real-time belief space search," *Journal of Intelligent & Robotic Systems*, vol. 64, no. 2, pp. 277–298, 2011.

[16] S. Temizer, M. J. Kochenderfer, L. P. Kaelbling, T. Lozano-Pérez, and J. K. Kuchar, "Collision avoidance for unmanned aircraft using markov decision processes," in *Proc. AIAA Guidance, Navigation, and Control Conference*, 2010.

[17] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *The International Journal of Robotics Research*, vol. 21, no. 3, pp. 233–255, 2002.

[18] J. Van Den Berg, D. Ferguson, and J. Kuffner, "Anytime path planning and replanning in dynamic environments," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*. IEEE, 2006, pp. 2366–2371.

[19] E. Frazzoli, M. A. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 1, pp. 116–129, 2002.

[20] Y. Kuwata, S. Karaman, J. Teo, E. Frazzoli, J. How, and G. Fiore, "Real-time motion planning with applications to autonomous urban driving," *Control Systems Technology, IEEE Transactions on*, vol. 17, no. 5, pp. 1105–1118, 2009.

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

[22] C. Fulgenzi, C. Tay, A. Spalanzani, and C. Laugier, "Probabilistic navigation in dynamic environment using rapidly-exploring random trees and gaussian processes," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. IEEE, 2008, pp. 1056–1062.

[23] B. D. Luders, G. S. Aoude, J. M. Joseph, N. Roy, and J. P. How, "Probabilistically safe avoidance of dynamic obstacles with uncertain motion patterns," 2011.

[24] S. Scherer, S. Singh, L. Chamberlain, and M. Elgersma, "Flying fast and low among obstacles: Methodology and experiments," *The International Journal of Robotics Research*, vol. 27, no. 5, pp. 549–574, 2008.

[25] M. Kobilarov, "Cross-entropy motion planning," *The International Journal of Robotics Research*, vol. 31, no. 7, pp. 855–871, 2012.

[26] S. Hrabar, "Reactive obstacle avoidance for rotorcraft uavs," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 4967–4974.

[27] A. M. Shkel and V. Lumelsky, "Classification of the dubins set," *Robotics and Autonomous Systems*, vol. 34, no. 4, pp. 179–202, 2001.

[28] Y. Lin and S. Saripalli, "Real time path planning and obstacle avoidance for unmanned aerial vehicles," in *Intelligent Robots and Systems (IROS) (submitted), 2014 IEEE/RSJ International Conference on*.

[29] P.-J. Bristeau, F. Callou, D. Vissière, N. Petit, *et al.*, "The navigation and control technology inside the ar. drone micro uav," in *18th IFAC World Congress*, 2011, pp. 1477–1484.