Laser-based Collision Avoidance and Reactive Navigation using RRT* and Signed Distance Field for Multirotor UAVs

Liang Lu, Carlos Sampedro, Javier Rodriguez-Vazquez and Pascual Campoy

Abstract— **Collision avoidance plays a crucial role for autonomous navigation in unknown dynamic environments and still remains an ongoing research problem. In this paper, we present a new collision avoidance algorithm by combining an RRT* path planner with a Signed Distance Field (SDF) based collision checking algorithm, in which the trajectory is optimized by a short cut and Optimal Polynomial Trajectory algorithms. The proposed algorithm is integrated to work in combination with a Model Predictive Control (MPC) based trajectory controller in order to provide a complete system for reactive navigation purposes. A thorough evaluation of the proposed algorithm has been conducted in several simulating scenarios using RotorS Gazebo simulator, showing fast collision checking capabilities in the presence of static and dynamic obstacles. The results show that the proposed algorithm outperforms in 76.93% considering the processing time when tested in a 1000 $\times$ 1000 pixels map. The results also demonstrate that the proposed navigation algorithm allows the safe navigation of a multirotor Unmanned Aerial Vehicle (UAV).**

I. INTRODUCTION

Collision avoidance is a key challenge in the area of autonomous navigation. Nowadays, a lot of researches are focus on this problem [1,8-17]. This challenge gets increased when the robotic platform has to operate in scenarios with dynamic obstacles. Operation in such situations, requires fast perception, control, and planning algorithms. Perception and control modules are responsible for the localization and motion of the robotic platform. In order to achieve collision avoidance capabilities, especially in dynamic environments, fast and flexible planning modules need to be implemented.

Sampling-based planners, such as Rapidly-exploring Random Trees (RRTs) [2], Probablistic Roadmaps (PRMs) [3] and Expansive Space Trees (ESTs) [4], are usually used inside the planning module for safe navigation featuring collision avoidance capabilities. As mentioned by Michal Kleinbort *et al* [5], collision checking plays a very important part for reducing the time of collision-free path finding of sampling-based planners. There are two main computations in the normal operation of sampling-based path planners: 1) determine whether a configuration is collision-free or not; 2) test if the straight line segment connecting two configurations lies in the free configuration space or not. Thus, in this paper, we focus our efforts on improving the aforementioned components for collision checking, which would lead to speed up the collision-free path finding procedure.

In this work, we use SDF to represent the obstacles and a new collision checking algorithm is designed. Subsequently, the proposed collision checking algorithm is integrated within an RRT* planner. Then we implement RRT* planner, which uses a random shot cut [20] and Optimal Polynomial Trajectory algorithms [21]. At last, we build a reactive navigation system, which combines a MPC based trajectory controller which is similar to [23] from our aerial robotic framework Aerostack[1] [6] and the RRT* planner. There are two key innovations in our work: a) We integrate a novel SDF based fast collision checking algorithm within an RRT* planner, which does not depend on the geometry of the obstacles. b) The proposed algorithm has been thoroughly evaluated in a wide range of simulation scenarios using RotorS Gazebo simulator [7], which provides realistic dynamics for the UAVs.

The remainder of the paper is organized as follows. Section II presents related works. In Section III, we introduce the problem formulation. The proposed collision checking algorithm and reactive navigation system is described in Section IV and V. We show the experiment results and discussion in Section VI, and finally, Section VII concludes the paper and summarizes future research directions.

II. RELATED WORK

Several methods have been proposed during the last decades addressing the problem of collision avoidance. Among these methods, the followings have gained a lot of interest within the research community, Artificial Potential Field (APF) [8], geometry-based approach [9], Velocity Obstacle (VO) [10-11], Partially Observable Markov Decision Process (POMDP) [12], learning based method [1,13] and sampling-based strategy [14-17].

The APF introduced in [8] uses gradient descent planning to find the minimum artificial potential energy. It calculates the attractive potential from the goal and repulsive potential from the obstacles, and find a trajectory which is collision-free. However, local minima is a common problem in these strategies.

The geometry-based approach presented in [9] uses the geometric relationship between the robot and the obstacles and calculates an updated path for the robot. However, these methods always use a simple UAV model and are applied in simple environments.

All authors are with the Computer Vision and Aerial Robotics (CVAR) Group. Centre for Automation and Robotics (CAR), UPM-CSIC (Technical University of Madrid). Calle José Gutiérrez Abascal 2, Madrid (Spain). liang.lu@upm.es

[1] www.aerostack.org

The VO like [10] uses a Probabilistic Velocity Obstacle (PVO) to predict the probability of the uncertainty in obstacles' position, shape and velocity. but it cannot guide the robot to the goal or avoid some emergency conditions. The work presented in [11] uses Acceleration Velocity Obstacle (AVO) to help the robot avoid obstacles in a dynamic environment while obeying acceleration constraints. However, VO based approaches highly rely on the robot's model and require accurate measurements of the obstacle's velocity.

The POMDP presented in [12] consists of a general framework for modeling and planning under uncertainty. The POMDP approach is applied to collision avoidance system development involves building a model that has a specified goal and operating environment.

Learning-based methods, such as the one presented in [13] uses Sample Consensus Initial Alignment (SAC-IA) in combination with a learning based approach to adapt obstacles boundary patterns confronted in prior environments to the present scenario followed by corresponding adaptations in the obstacle-free path. However, this approach needs to train the path planner offline and the obstacles in the environment should have great similarities in their boundary patterns respect to the trained missions. The work presented in [1] uses a deep reinforcement learning approach to build a reactive navigation algorithm by adopting an APF formulation in the reward function. The algorithm provides appropriate reactive navigation behaviors in simulated and real indoor scenarios with static and dynamic obstacles.

Sampling-based strategies usually have important constraints regarding the computational cost derived from computing an appropriate path. In order to reduce the time consuming, Kris Hauser [14] uses a lazy collision check procedure which checks the collision only if a better path is found. Yucong Lin *et al* [15] develop a sampling-based method to provide reactive navigation capabilities and build a closed-loop system in order to simulate the trajectory of the UAV during the collision checking procedure. Jia Pan and Dinesh Manocha [16] present a collision checking for a sampling-based approach, which determines new query samples collision status guess according to the collision checking results of these previous query samples, based on the reality that neighbouring configurations are probable to have the similar collision status. Joshua Bialkowski, Sertac Karaman and *et al* [17] develop a strategy to perform collision checking for sampling-based methods, this method reduces the computational cost by not calling the collision checking function if new samples are closer to a previously collision-checked point than the latter is to an obstacle. The proposed work presented in this paper also uses a sampling-based strategy but enhanced by using an SDF based fast collision checking algorithm for the navigation of multirotor UAVs.

III. PROBLEM FORMULATION

This section describes in detail the formulation followed for representing the UAV and the obstacles within the planning problem. The model of multirotor UAV is described first and then the obstacles representation is given.

A. Model Assumption

The obstacle avoidance and reactive navigation simulations are based on the following assumptions:

The multirotor UAV has 6 DOF pose encompassing the translation,$(x, y, z)^T$, and rotational, $(\phi, \theta, \psi)^T$, where ϕ, θ and ψ represent the roll, pitch and yaw of the UAV. Thus the UAV can fly freely in the 3D environment. In practice, the UAV's body can be approximated by a set of geometric primitives which enclose it, such as spheres, capsules, and polyhedral. In this paper, The body of the UAV is approximated as a sphere which has a risk radius of R_{risk}. Risk radius means the minimum safe distance from obstacle to robot. The main on-board sensor are a Hokuyo-laser which can detect the environment within the sensor's Field Of View (FOV) and an Inertial Measurement Unit (IMU) sensor which can estimate the attitude and heading of the UAV.

B. Obstacles Representation

Obstacles are represented by the function $Dist(x)$ which is used to compute the distance from any point in the environment to the surface of the nearest obstacles. Assuming all obstacles are closed objects with finite volume, if the point x in the environment is inside an obstacle, $Dist(x)$ is negative, whereas if the point x is outside all obstacles, $Dist(x)$ is positive, and $Dist(x)$ is zero if the point x lies on the frontier of an obstacle.

In general, there are two ways to compute the distance function, one is using geometric obstacle primitives (such as boxes, spheres and cylinders) and another approach consists of using the Euclidean Distance Transform (EDT) [18]. However the obstacles (complicate, non-convex shapes) cannot be easily represented by geometric obstacle primitives. For these reasons, an efficient EDT algorithm is used to compute the function by using a boolean obstacle grid. The boolean obstacle grid can be built from raw laser information or hector SLAM mapping [19]. The SDF which is $Dist_{img}(x)$, as shown in Fig.1c, is computed by taking the difference of distance function $d(x_{img})$ which is Distance Field (DF) (Fig.1a) and its complement $\overline{d}(x_{img})$ (Fig.1b). Fig.1a and Fig.1b are calculated by using the EDT from the laser scan of the multirotor UAV in the Gazebo environment (Fig.1d). $d(x_{img})$ is the distance in pixels from any point in the image to the nearest obstacle and $\overline{d}(x_{img})$ is the complement of $d(x_{img})$, and $Dist_{img}(x)$ is the SDF and given by the difference of $d(x_{img})$ and $\overline{d}(x_{img})$.

The process of calculating $Dist_{img}(x)$ is shown in (1).

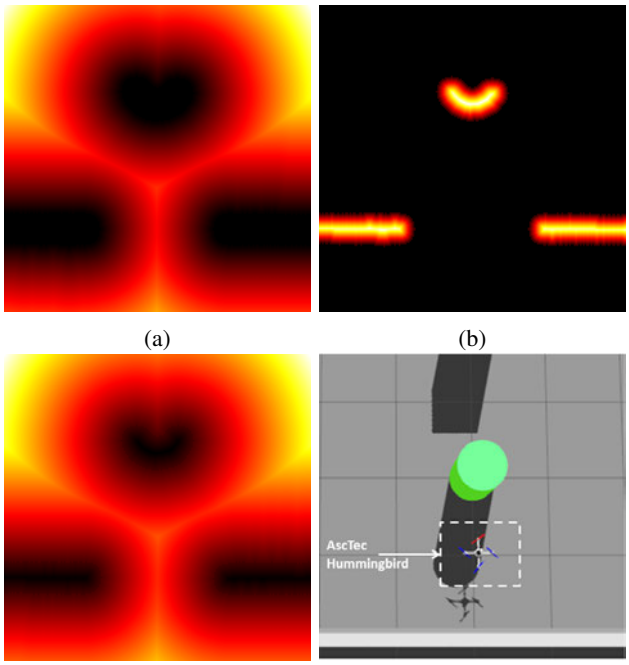
$$Dist_{img}(x) = d(x_{img}) - \overline{d}(x_{img}) \tag{1}$$

After computing $Dist_{img}(x_{img})$, a transformation from world frame to image frame is performed, then the distance function $Dist(x)$ can be computed by multiplying $Dist_{img}(x_{img})$ by the resolution of the SDF which is res in the equation.

The process of computing $Dist(x)$ is described in (2).

$$Dist(x) = Dist_{img}(T_{F_W}^{F_{IMG}}(x)) \times res \tag{2}$$

Where T is the transformation from world frame F_W to image frame F_{IMG}.



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

Fig. 1: SDF transformation. (a) is the DF, (b) is the DF Complement, (c) is the SDF and (d) is the Gazebo environment for illustrating the process.

IV. SIGNED DISTANCE FIELD BASED COLLISION CHECKING ALGORITHM

With the distance function $Dist(x)$, a new strategy is designed to check the collision between the line segment and the obstacles.

Assuming there are finite obstacles and a line segment in the environment. The start point of the line segment is x_i and the end point is x_j. The distance from one point in the line segment to the surface of the nearest obstacles is $Dist(x_i)$, $i \in 0, 1, 2, \cdots, j-1, j$. R_{risk} is the safety radius of robot when it moves within the environment.

Lemma 4.1: For x_i, $Dist(x_i) > R_{risk}$, we can find a point x_{i+1} in the line segment, for which the distance between x_i and x_{i+1} is $Dist(x_i) - R_{risk}$. For all the segment points between x_i and x_{i+1} in the line segment, there is no obstacle near to them in the distance of R_{risk}.

Proof.

As can be seen from Fig.2, C_{risk} is a circle whose radius is R_{risk}(d_3 in Fig.2) and its center can move between x_i and x_{i+1}. $C1$ and $C2$ are the obstacles, C_i represents the circles whose radius d_1 is the distance to the nearest obstacle $C1$ from x_i. It means there is no obstacle in the area of C_i, so proof *Lemma 4.1* is equal to proof all the area of C_{risk} is in the area of C_i. For the point $p \in (x_i, x_{i+1})$ which is the center of C_{risk}, (3) and (4) can be used to judge if C_{risk} is in the area of C_i or not.

$$F_{IsCircleIn}(p) = Dist(x_i) - R_{risk} - ||p - x_i|| \tag{3}$$

$$\begin{cases} C_{risk} \not\subset C_i, & F_{IsCircleIn}(p) < 0 \\ C_{risk} \subset C_i, & F_{IsCircleIn}(p) \geq 0 \end{cases} \tag{4}$$

As $Dist(x_i)$ and R_{risk} is static, so if $F_{IsCircleIn}$ is no less than 0 when $||p - x_i||$ comes to the biggest value, C_{risk} will be in the area of C_i. Because the distance between x_i and x_{i+1} is $Dist(x_i) - R_{risk}$, the biggest value of $||p - x_i||$ will be $Dist(x_i) - R_{risk}$(when p reaches x_{i+1}) and the value of $F_{IsCircleIn}$ will be 0, so all the area of C_{risk} will be in the area of C_i and *Lemma 4.1* is proved.

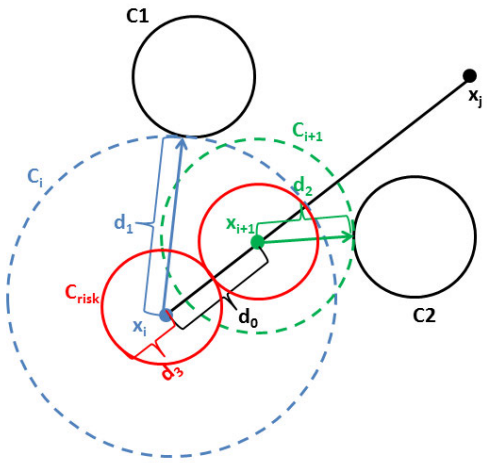


Fig. 2: Collision checking method, d_0, d_1, d_2 and d_3 are equal to $Dist(x_i) - R_{risk}$, $Dist(x_i)$, $Dist(x_{i+1})$ and R_{risk} respectively.

Equation (5) shows the strategy to find x_{i+1} and (6) shows the strategy of collision checking.

$$x_{i+1} = x_i + \frac{Dist(x_i) - R_{risk}}{||x_i x_j||} \cdot \overrightarrow{x_i x_j} \tag{5}$$

$$CheckCollision(x_i) = \begin{cases} 1, & Dist(x_i) \leq R_{risk} \\ 0, & Dist(x_i) > R_{risk} \end{cases} \tag{6}$$

We assume there is no obstacle in C_{risk} when the center of it is x_i, In order to check if the line $\overline{x_i x_j}$ is collision free, the collision checking algorithm will keep finding x_{i+1} using (5) and check collision of this point using (6). This procedure will continue until the value of $CheckCollision(x_{i+1})$ is 1, which means $\overline{x_i x_j}$ is in collision, or until $Dist(x_{i+1})$ is no less than $||x_{i+1} x_j|| + R_{risk}$, which means $\overline{x_i x_j}$ is collision free.

The pseudocode of the proposed collision checking algorithm is shown in Algorithm 1.

Algorithm 1 Proposed Collision Checking Algorithm

Input: $x_i, x_j, R_{risk}, Dist(x)$
Output: $Flag1$

$x_{init} \leftarrow x_i, x_{goal} \leftarrow x_j, Flag1 \leftarrow False$;
$eta \leftarrow 0$;
$eta = Norm(x_{init}, x_{goal})$;
if $Dist(x_{init}) < R_{risk}$ **then**
  $Flag1 = True$;
  $break$;
else
  $x_{new} \leftarrow x_{init}$;
  while $Dist(x_{new}) < eta + R_{risk}$ **do**
    $x_{new} \leftarrow x_{new} + \frac{Dist(x_{new}) - R_{risk}}{||\overrightarrow{x_{int}x_{goal}}||} \cdot \overrightarrow{x_{init}x_{goal}}$;
    $eta = Norm(x_{new}, x_{goal})$;
    if $Dist(x_{new}) <= R_{risk}$ **then**
      $Flag1 = True$;
      $break$;
    end if
  end while
end if
return $Flag1$;

V. OPTIMAL RRT* BASED REACTIVE NAVIGATION SYSTEM

By using the SDF based collision checking algorithm, a reactive navigation system is developed in this paper. This reactive navigation system is based on an RRT* framework. Four main improvements have been done to properly integrate the proposed system onboard an aerial robotic platform to perform reactive navigation tasks. 1) using the proposed SDF based collision checking algorithm to check collisions, which is described in section II. 2) a random short cut algorithm [20] is applied to shorten the path generated by the RRT* algorithm. 3) an Optimal Polynomial Trajectory algorithm [21] is applied to smooth the path after shortening. 4) building the reactive navigation system on top of our software architecture named Aerostack.

The whole system is shown in Fig.3. First, a raw path is computed using the RRT* planner with the information of initial point $X_{initial}$, goal point X_{goal} and the distance function $Dist(X)$. Then the raw path is shortened by a path shortening module and smoothed by a polynomial trajectory optimization module. Lastly, a control system is used to fly the multirotor UAV. During the flight, if there is a collision between the trajectory and newly detected obstacles, the planner will plan a new raw path. This process will not stop until the multirotor UAV reach the goal.

A. RRT*

For a better comprehension of the entire document, we describe the main features of the RRT* algorithm [22], which is an extension of the standard RRT algorithm. The pseudocode is shown in algorithm 2. Compared with standard RRT algorithm, there are two main improvements. 1) RRT* algorithm will choose near nodes around a new

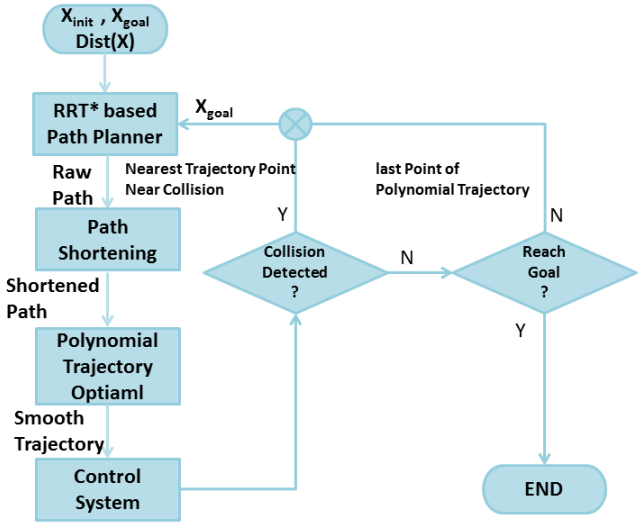


Fig. 3: The process of the proposed reactive navigation system.

extended node. These near nodes will be in a sphere region whose center is the new extended node. the radius of the sphere region is $\gamma(\frac{logn}{n})^{1/d}$, d is the dimension and γ is a constant. The algorithm will choose the best parent of the new extended node from these near nodes. 2) A rewire function which will change the parent of the nodes in the sphere region if the cost from new extended node to them is less than their previous cost. This behavior is shown in line 10 of the pseudocode presented in Algorithm 2.

A detail description of the functions in the pseudocode can be found in [22].

Algorithm 2 RRT* Path Planner

Input: $P_{initial}$, P_{goal}
Output: $Path$

1: $T \leftarrow P_{initial}, Iteration \leftarrow N$;
2: **while** $true$ **do**
3: $P_{rand} \leftarrow SampleFree()$;
4: $P_{nearest} \leftarrow Nearest(T, P_{rand})$;
5: $(P_{new}, T_{new}) \leftarrow Steer(P_{nearest}, P_{rand})$;
6: **if** $ObstacleFree(P_{new})$ **then**
7: $P_{near} \leftarrow Near(T, P_{new}, R)$;
8: $P_{min} \leftarrow BestParent(P_{near}, P_{nearest}, P_{new})$;
9: $T \leftarrow (P_{new}, T)$;
10: $T \leftarrow Rewire(T, P_{min}, P_{near}, P_{new})$;
11: **end if**
12: **if** $Reach(P_{goal})$ **or** $Iteration \geq N$ **then**
13: $Pub(Path\{P_{init}, \cdots, P_{goal}\})$;
14: $break$;
15: **end if**
16: **end while**

B. Path Shortening

In order to shorten the raw path generated from the RRT* planner, a random short cut algorithm is ap-

plied. The pseudocode is shown in Algorithm 3. N is the number of the segment points in the raw path. $NewPathSegment(P(x_i), P(x_j))$ is a new path segment. The start point and end point of the path segment are $P(x_i)$ and $P(x_j)$ respectively. $NewLine(P(x_i), P(x_j))$ means generating a new path segment from $P(x_i)$ to $P(x_j)$ and $OldLine(P(x_i), P(x_j))$ means using the raw path segment from $P(x_i)$ to $P(x_j)$.

Algorithm 3 Random Short Cut Algorithm

Input: P_{raw}
Output: $P_{shorten}$

$NumFail \leftarrow 0$;
while $NumFail < MaxNumFail$ **do**
  $failure \leftarrow true$;
  $N \leftarrow Number\ of\ Segment\ Points\ in\ Path$;
  $x_1 = Rand(0, N-1)$;
  $L_{01} = NewPathSegment(P_{raw}(0), P_{raw}(x_1))$;
  $L_{12} = NewPathSegment(P_{raw}(x_1), P_{raw}(2))$;
  if $ObsFree(L_{01})$ **then**
    $P_{new} \leftarrow L_{01}$, $failture \leftarrow false$;
  else
    $P_{new} \leftarrow OldLine(P_{raw}(0), P_{raw}(x))$;
  end if
  if $ObsFree(L_{12})$ **then**
    $P_{new} \leftarrow NewLine(P_{new}, L_{12})$;
    $failture \leftarrow false$;
  else
    $P_{new} \leftarrow OldLine(P_{new}, P_{raw}(2))$;
  end if
  $P_{shorten} = P_{new}$;
  if $failture$ **then**
    $NumFail = NumFail + 1$;
  end if
end while

C. Optimal Polynomial Trajectory

After shortening, it is also very important to smooth the path to make it possible for a multirotor UAV to fly it. In order to smooth the shortened path, the Optimal Polynomial Trajectory approach is used. This approach considers a differential flat representation of the multirotor UAV model and transforms the shortened path into a smooth path through an optimization between the time taken for traversal and the snap of the path.

D. Trajectory Control System for Multirotor UAV

A MPC based trajectory controller which is similar to [23] is used to fly the multirotor UAV. The smooth trajectory will be interpolated into several points and UAV will fly these trajectory points.

As can be seen in Fig.4, the red trajectory is the current smooth trajectory. A few trajectory points in the fly way of the multirotor UAV will be checked. If there is one trajectory point within an obstacle, the minimum distance from the point to the obstacle is less than R_{risk}. For example, P is the trajectory point, when D_{min} is less than risk radius, a new trajectory will be found (the orange trajectory in Fig.4). The UAV will follow the new trajectory until it reaches the goal or a new collision is detected.

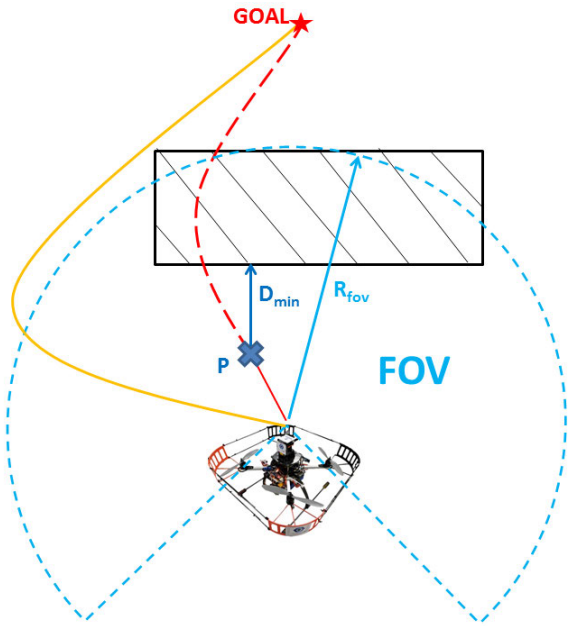


Fig. 4: Collision checking and trajectory re-plan.

TABLE I: Comparison of the proposed algorithm and baseline algorithm. The scenarios used in the test are shown in Fig.5.

Scenario		*Proposed Algorithm*	*Baseline Algorithm*
Scenario A1	*PT(ms)*	***70.78±12.33***	*306.8±85.7*
	SR(%)	***91.04***	*90.40*
Scenario A2	*PT(ms)*	***10.28±1.64***	*16.9±5.35*
	SR(%)	***92.09***	*88.55*
Scenario B	*PT(ms)*	***10.46±1.65***	*26.8±8.15*
	SR(%)	***81.89***	*79.15*
Scenario C	*PT(ms)*	***10.53±1.77***	*33.65±7.45*
	SR(%)	***88.44***	*86.60*

VI. EXPERIMENTS AND RESULTS

A. Experimental Setup

RotorS Gazebo simulation environment and Robot Operating System (ROS) [24] are used to run on the top of Ubuntu 18.04. The Rviz/Gazebo environment can use real physical parameters of robot and environment model. All the experiments run on a laptop with Intel Core i7-

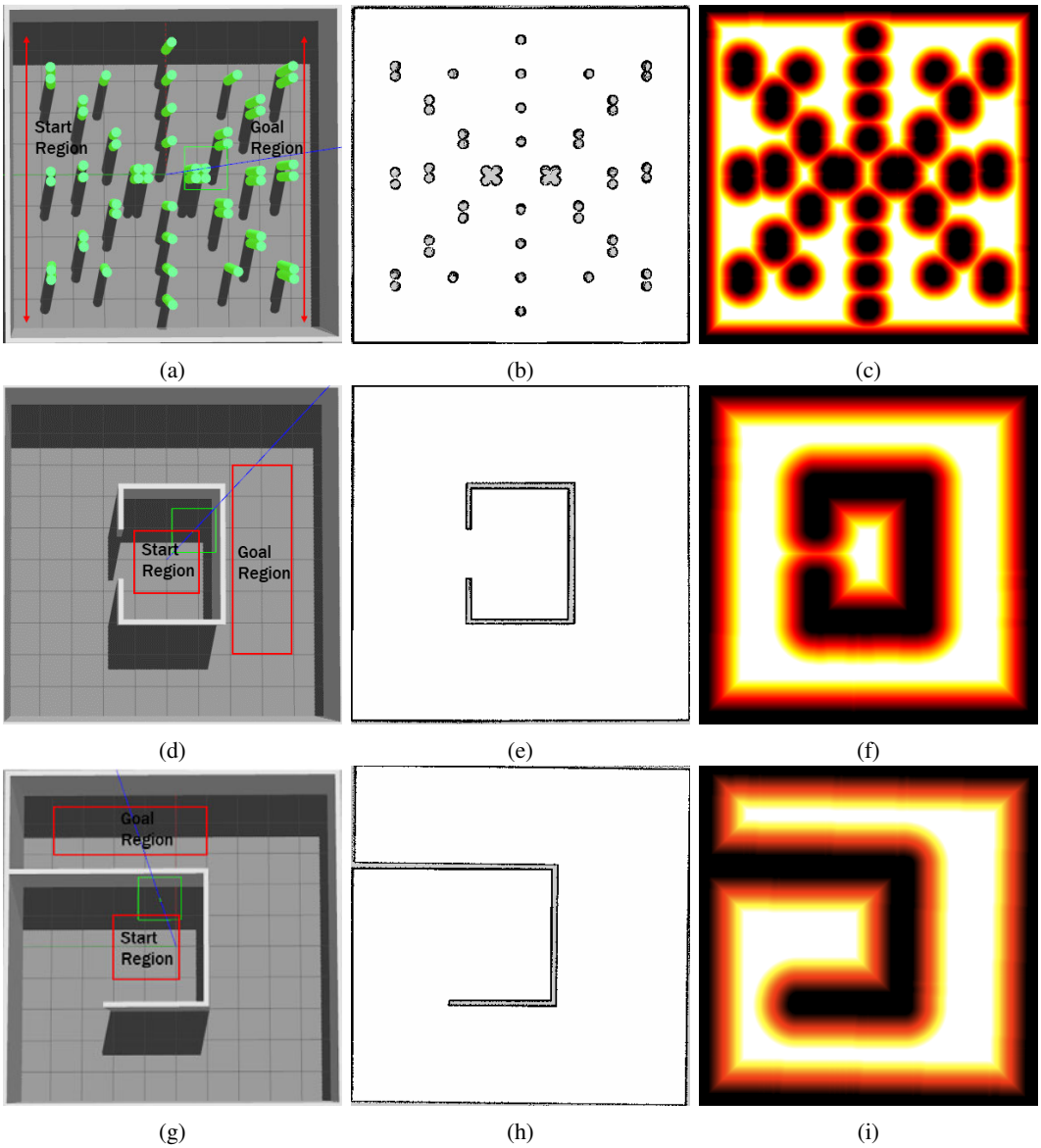
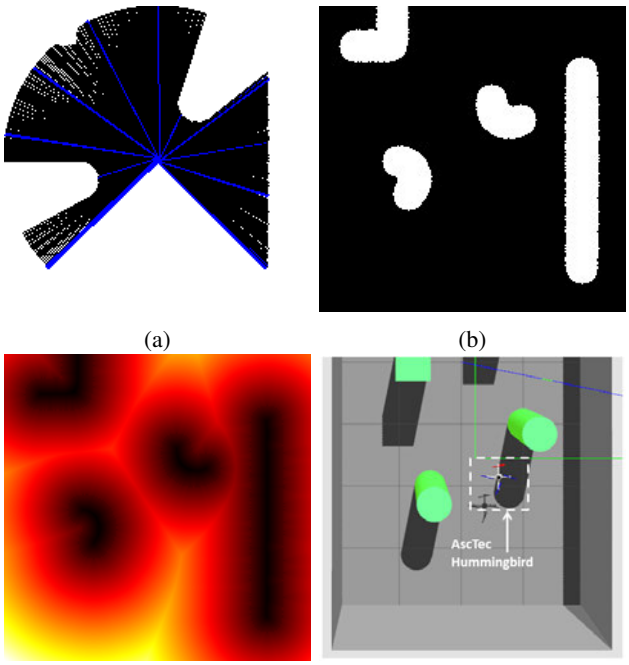


Fig. 5: Three different scenarios and their SDF transformation, the size of the scenario is 10 × 10 m. (a), (d) and (g) represent the gazebo environment model of scenario A, B and C respectively. (b), (e) and (h) are the occupancy grid map of scenario A, B and C respectively. (c), (f) and (i) are the SDF of scenario A, B and C. The map of scenario A1 and A2 in table I and II are both built from scenario A but with different resolution.

TABLE II: Results obtained in the evaluation of the proposed algorithm in the scenarios of Fig.5.

Scenario	*Mapping Time(ms)*	*Path Finding Time(ms)*
Scenario A1	*70.7±12.3*	*0.0875±0.0308*
Scenario A2	*10.2±1.6*	*0.0882±0.0404*
Scenario B	*10.3±1.6*	*0.161±0.0577*
Scenario C	*10.3±1.7*	*0.235±0.0764*

8750H at 2.2GHz, 16GB memory. The simulation of the proposed navigation system is integrated into our open source framework Aerostack. The used environments are 3D indoor environments. The UAV in the simulation is the AscTec Hummingbird, which is equipped with a Hokuyo laser rangefinder UTM-30LX with a FOV of 270°, maximum range 30 m and minimum range 10 cm. the laser takes charge of receiving information from the outside environment. The UAV can fly in the 3D simulation environment, but the information used to perform obstacle avoidance is 2D.



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

Fig. 6: SDF computation procedure using the laser scan information. (a) is the laser scan, (b) is the obstacle field built from the laser scan, (c) is the SDF and (d) is a sample Gazebo environment.

B. Comparison of The Proposed Collision Avoidance Algorithm and Baseline Algorithm

Baseline Algorithm: We use an RRT* with a standard way to perform collision checking [5] as the baseline algorithm. In order to detect if a line segment in the environment is in collision or not, the line is split into a finite number of points. If one of the points is in collision, the line will be defined as in collision.

In order to test the performance of the collision checking algorithm, we integrate the proposed and the baseline algorithms into an RRT* planner. In order to evaluate and compare the proposed algorithm and the baseline algorithm, we build three different scenarios in the Gazebo simulation environment. Then, we use Hector-SLAM ROS package to build the occupancy grid map.

For comparison, we use 4 different scenarios A1, A2, B and C, which are shown in the Fig.5. The occupancy grid map of scenario A has two different resolutions, one is 0.01 m/pixel and the size is 1000 $\times$ 1000 pixels (scenario A1 in Tables I and II), the other one is 0.025 m/pixel and the size is 400 $\times$ 400 pixels (scenario A2 in Tables I and II). The main difference between scenario A1 and scenario A2, the number of pixels in scenario A2, which is only 16 $\%$ the number in scenario A1. The resolution of the occupancy grid map of scenario B and C is 0.025 m/pixel and the size is 400 $\times$ 400 pixels. For every scenario, the coordinate of center is $(0, 0)$, the coordinate of lower left corner is $(-5, 5)$, the coordinate of lower right corner is $(-5, -5)$, the coordinate of upper left corner is $(5, 5)$ and the coordinate of upper right corner is $(5, -5)$.

In every scenario, the RRT* based planner with the proposed collision checking algorithm and the RRT* baseline algorithm are executed 2000 times. In scenario A1 and A2, coordinate x of initial point and goal point are randomly generated between $[-4.8, 4.8]$, coordinate y will be -4.5 for the initial point and 4.5 for the goal point. In scenario B, both coordinates (x and y) of initial point are randomly generate between $[-1, 1]$, coordinate x and coordinate y of goal point are randomly generated between $[-3, 3]$ and $[-2, -4]$ respectively. In scenario C, the coordinates corresponding to the initial takeoff point of the UAV are set randomly between $[-1, 1]$ and $[0, 2]$, on the other hand, the coordinates of the goal vary between $[3, 3.5]$ and $[-1, 4]$. Then we measure the average time of planning a feasible path, as well as the successful rate to find a feasible path. In order to achieve the best performance, we select the number of iterations for the RRT* algorithm to be 200, 400, and 600 in the scenarios A1, A2, and B respectively. The results are shown in Tables I and II.

In Table I, PT is the mean time to achieve a feasible path and SR is the performance for finding the feasible path in the 2000 times running. In Table II, the mean mapping time for building SDF from occupancy grid map and the path finding time of the proposed algorithm are given.

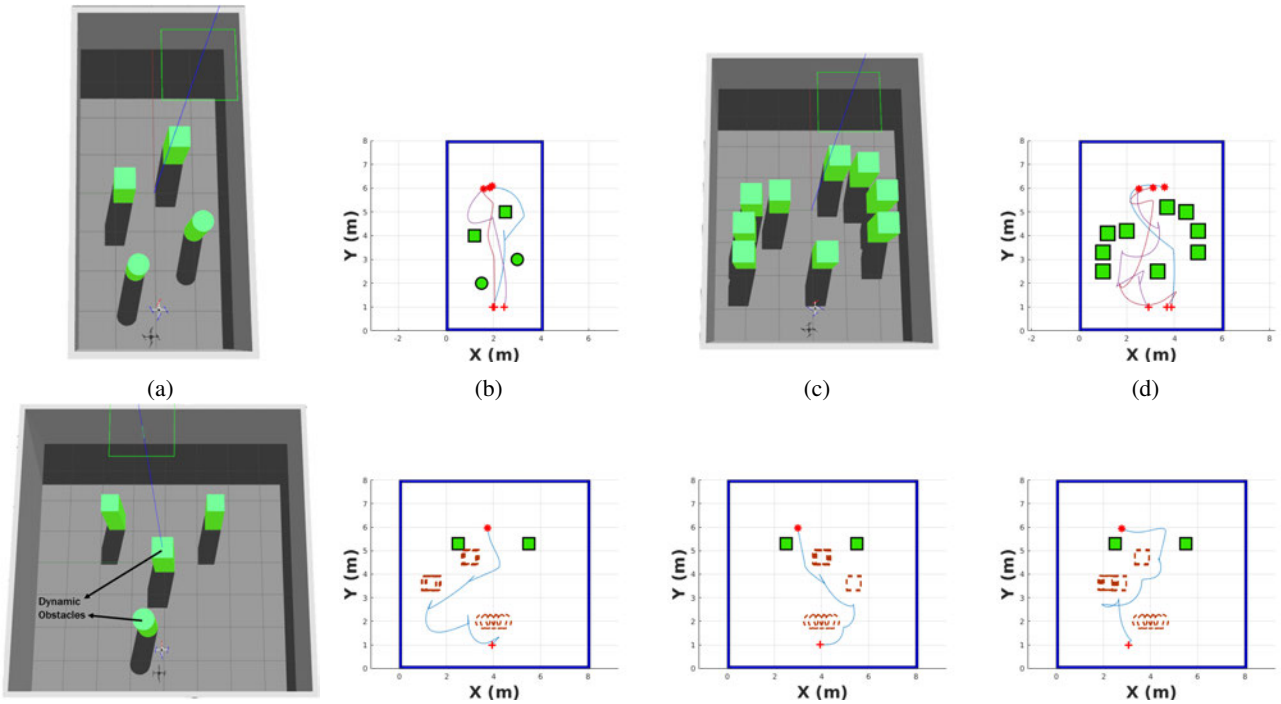
C. Simulation Results of the Reactive Navigation System

As shown in Fig.6, from an specific time step during the flight in a Gazebo simulation scenario (see Fig.6d), we use the current laser scan (see Fig.6a) to build the obstacle field (see Fig.6b), from which the SDF is computed (see Fig.6c). In the simulation, the area of the sensors FOV is a arc with an angle of 270°, and its radius is 2 m. The obstacle field is a 4 $\times$ 4 m squared area. the size of the SDF built from the obstacle field is 200 $\times$ 200 pixels and its resolution is 0.02 m/pixel.

Regarding the simulation experiments conducted in order to test the proposed reactive navigation algorithm, the main reasons of restricting the range of the laser scan are: First, in order to reduce the SDF mapping time to improve the real-time performance of our algorithm, which is important for reactive navigation. Second, the planner used in the reactive navigation system performs as a local planner.

We build three different scenarios in the Gazebo simulator to test the reactive navigation system. These scenarios can be seen in Fig.7a, Fig.7c and Fig.7e. The first two scenarios contain static obstacles (Fig.7a and Fig.7c), while the last one is made up of dynamic obstacles (Fig.7e). In the last scenario, there are obstacles which can exhibit different movements (a quadrilateral obstacle which can suddenly appear in front of the robot and a cylindrical obstacle which has a sinusoidal trajectory).

A video showing the results achieved by the proposed algorithm in the aforementioned experiments has been made available in: `https://vimeo.com/318632365`.



(a) (b) (c) (d) (e) (f) (g) (h)

Fig. 7: Three different scenarios and three flying trajectories for every scenario. (a), (c) and (e) are the Gazebo environments. (b) and (d) are three flying trajectories for (a) and (c) while (f), (g) and (h) are three flying trajectories for (e).

TABLE III: Evaluation of different trajectories in the Gazebo environments which are shown in Fig.7.

Scenario		*PL(m)*	*TG(s)*	*MD(m)*
Scenario 1	*Trajectory 1(Brown)*	*6.09*	*17.15*	*0.6223*
	Trajectory 2(Purple)	*6.67*	*22.36*	*0.5849*
	Trajectory 3(Blue)	*6.90*	*19.87*	*0.6881*
Scenario 2	*Trajectory 1(Brown)*	*7.82*	*25.79*	*0.7744*
	Trajectory 2(Purple)	*9.11*	*32.72*	*0.6067*
	Trajectory 3(Blue)	*6.23*	*16.97*	*0.8249*
Scenario 3	*Trajectory 1(Left)*	*9.38*	*40.18*	*0.7567*
	Trajectory 2(Midden)	*6.94*	*27.50*	*0.6123*
	Trajectory 3(Right)	*7.26*	*34.62*	*0.7071*

D. Discussion

As can be seen from Table I, the RRT* algorithm can plan a path much faster with the proposed collision checking algorithm. The planner saves 76.93 %, 39.17 %, 60.97 % and 68.71 % the time used by the baseline algorithm in scenario A1, A2, B and C, respectively. Table I also shows that it will not lose any success rate when implementing the proposed collision checking algorithm within the RRT* planner. Table II shows that the required time for building the map takes a large part of the whole planning time when using RRT* planner with our collision checking algorithm. It will take much less time when using our algorithm with a pre-built map. Results presented in Tables I and II also reveal the appropriate behavior of the proposed algorithm in scenarios with U-shaped obstacles, as shown in Fig.5.

Table III is the evaluation of the results in Fig.7, scenario 1, 2 and 3 are the Gazebo environments in Fig.7a, Fig.7c and Fig.7e respectively. PL is the length of the trajectory, TG is the time for the robot to reach the goal and MD is the minimum distance to the obstacle when robot is moving in the trajectory from the starting point to the goal point.

In this paper we have conducted an extensive set of simulation experiments for evaluating the proposed algorithm and compare it with a state-of-the-art path planner. This evaluation and comparison have been conducted using the RotorS Gazebo simulator, which is widely adopted in the robotics research community [25-26] as the dynamics included in the aerial robot models provide realistic behaviors which can provide an appropriate feeling of the behavior of the robot in real flight conditions.

VII. CONCLUSION AND FUTURE WORK

In this work, for achieving collision avoidance in a 2D dynamic environment, we propose a SDF based fast collision checking algorithm and combine it with an optimal RRT* planner. Furthermore, we integrate our proposed algorithm within Aerostack framework for aerial robotics. By

comparing our collision checking algorithm with an RRT* baseline, results show that our algorithm uses much less time to find a collision-free path than baseline in the simulation scenarios (as shown in TABLE I). The results of simulations of the navigation system can also reveal that our navigation system achieves reactive navigation in an unknown dynamic environment.

As a continuation of this work, we will integrate the proposed collision checking algorithm within a global sampling-based planner for obtaining a complete system which can provide navigation capabilities in a faster and optimal way. The real flight experiments will also be conducted in order to test the capabilities of the proposed planning algorithm running on-board a multirotor aerial robot.

ACKNOWLEDGMENT

The work reported in this paper is sponsored by the Chinese Scholarship Council (CSC). The MONCLOA Campus of International Excellence is also acknowledged for funding the predoctoral contract of the second author.

REFERENCES

[1] C. Sampedro, H. Bavle, A. Rodriguez-Ramos, P. de la Puente and P. Campoy, "Laser-Based Reactive Navigation for Multirotor Aerial Robots using Deep Reinforcement Learning," in Intelligent Robots and Systems (IROS), *2018 IEEE/RSJ International Conference on.* IEEE, 2018, pp.1024-1031.

[2] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *International Journal of Robotics Research*, 20(5): 378-400, 2001.

[3] L. E. Kavraki, P. vestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, 12(4): 566-580, 1996.

[4] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles", *International Journal of Robotics Research*, 21(3): 233-255, 2002.

[5] M. Kleinbort, O. Salzman, and D. Halperin, "Collision detection or nearest-neighbor search? On the computational bottleneck in sampling based motion planning", *arXiv preprint arXiv*: 1607.04800v3, 2016.

[6] J. L. Sanchez-Lopez, J. Pestana, P. de la Puente and P. Campoy, "A Reliable Open-Source System Architecture for the Fast Designing and Prototyping of Autonomous Multi-UAV Systems: Simulation and Experimentation," *J Intell Robot Syst*, 84: 779-797, 2016.

[7] F. Furrer, M. Burri, M. Achtelik and R. Siegwart "RotorS - A Modular Gazebo MAV Simulator Framework," *In: Koubaa A. (eds) Robot Operating System (ROS). Studies in Computational Intelligence*, vol 625. Springer, Cham, 2016.

[8] K. Sigurd and J. How, "UAV trajectory design using total field collision avoidance," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2003, p. 5728.

[9] Z. Lin, L. Castano, and H. Xu,"A Fast Obstacle Collision Avoidance Algorithm for Fixed Wing UAS," in Unmanned Aircraft Systems (ICUAS), *2018 IEEE International Conference on.*, IEEE, 2018, pp. 559-568.

[10] C. Fulgenzi, A. Spalanzani, and C. Laugier, "Dynamic Obstacle Avoidance in uncertain environment combining PVOs and Occupancy Grid," in Robotics and Automation (ICRA), *2007 IEEE International Conference on.*, IEEE, 2007, pp. 1610-1616.

[11] J. van den Berg, J. Snape, S. J. Guy and D. Manocha, "Reciprocal Collision Avoidance with Acceleration-Velocity Obstacles," in Robotics and Automation (ICRA), *2011 IEEE International Conference on.*, IEEE, 2011, 3475-3482.

[12] H. Bai, D. Hsu, M. J. Kochenderfer, and W. S. Lee, "Unmanned aircraft collision avoidance using continuous-state POMDPs," *Robotics: Science and Systems (RSS) VII*, 2012, p. 1.

[13] O. Saha and P. Dasgupta, "Experience Learning From Basic Patterns for Efficient Robot Navigation in Indoor Environments," *J Intell Robot Syst*, 92: 545-564, 2018.

[14] K. Hauser, "Lazy Collision Checking in Asymptotically Optimal Motion Planning," in Robotics and Automation (ICRA), *2015 International Conference on.*, IEEE, 2015, pp. 2951-2957.

[15] Y. Lin, S. Saripalli, "Sampling based collision avoidance for UAVs," in *2016 American Control Conference (ACC)*, IEEE, 2016, pp. 1353-1358.

[16] J. Pan, and D. Manocha, "Fast probabilistic collision checking for sampling-based motion planning using locality-sensitive hashing", *International Journal of Robotics Research*, 35(12): 1477-1496, 2016.

[17] J. Bialkowski, S. Karaman, M. Otte, and E. Frazzoli, "Efficient Collision Checking in Sampling-based Motion Planning," *International Journal of Robotics Research*, 35(7): 769-796, 2016.

[18] P. F. Felzenszwalb and D. P. Huttenlocher, "Distance transforms of sampled functions." *Technical Report TR2004-1963*, Cornell University, 2004.

[19] S. Kohlbrecher, J. Meyer, O. von Stryk and U. Klingauf, "A Flexible and Scalable SLAM System with Full 3D Motion Estimation," In Safety, Security and Rescue Robotics (SSRR), *2011 International Symposium on.*, IEEE, 2011, pp. 155-160.

[20] S. Sekhavat, P. Svestka, J. P. Laumond, and M. Overmars, "Multilevel path planning for nonholonomic robots using semi-holonomic subsystems," *International Journal of Robotics Research*, 17(8): 840-857, 1998.

[21] R. Charles, B. Adam and R. Nicholas, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," *In: Inaba M., Corke P. (eds) Robotics Research. Springer Tracts in Advanced Robotics*, vol 114, Springer, 2016.

[22] S. Karaman and E. Frazzli, "Sampling based algorithms for optimal motion planning," *International Journal of Robotics Research*, 30(7): 846-894, 2011.

[23] M. Kamel, M. Burri and R. Siegwart, "Linear vs Nonlinear MPC for Trajectory Tracking Applied to Rotary Wing Micro Aerial Vehicles," *arXiv preprint arXiv: 1611.09240*, 2016.

[24] Q. Morgan, C. Ken, G. Brian P, F. Josh, F. Tully, L. Jeremy, W. Rob, and Ng. Andrew Y, "ROS: an open-source Robot Operating System," *Proc. Open-source Software Workshop of Robotics and Automation (ICRA), 2009 International Conference on.*, IEEE, 2009.

[25] T. Cieslewski, E. Kaufmann and D. Scaramuzza, "Rapid exploration with multi-rotors: A frontier selection method for high speed flight," in Intelligent Robots and Systems (IROS), *2017 IEEE/RSJ International Conference on.*, IEEE, 2017, pp. 2135-2142.

[26] H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart and E. Galceran, "Continuous-time trajectory optimization for online UAV replanning," in Intelligent Robots and Systems (IROS), *2016 IEEE/RSJ International Conference on.*, IEEE, 2016, pp. 5332-5339.