repulsive fields of obstacles that are far away (for potential fields), sampling far away areas (for PRM) or creating Voronoi diagrams. The RRT method differs from that but is largely affected by the sampler function. Proper biasing on an RRT planner can lead to fast convergence in some cases and very slow convergence in other cases, therefore there is no general do-it-all method that can be used for efficiently computing a global path in a large map for robots with or without holonomic constraints.

A different approach was introduced in [8] where an initial spline connecting the start and goal positions was iteratively altered by adding control points by a collision detection algorithm. The additional control points introduced by the collision, where initially added on the midpoint of the collision entry and exit points. In a step by step manner, the control point is moved perpendicular to the collision until the path is free of collisions. This approach presents a few drawbacks although a smooth trajectory is generated and the computational complexity is low. First of all, the control points (and therefore the collision coordinates) must be sorted in an ascending manner. Secondly, this method may be trapped in complex workspaces where narrow passages might exist or rooms which have only one exit. More importantly, the creation of the configuration space is needed for the entire map (inflated obstacles), which is a computationally intensive and, as it will be shown, an unneeded task in numerous cases.

Advancing that approach, a hybrid method combining Voronoi diagrams and Bézier curves was introduced in [9]. The method proposed advances the work of [10] by resolving the drawbacks of his real-time obstacle avoiding algorithm that partitions the space using Voronoi diagrams of the obstacles and the start and goal positions. The generation of Bézier curves using control points is a rather complicated procedure, where many different cases about the obstacle shapes and the curve are studied so as to avoid collision with the obstacles. The convex hull of the control points is checked for collision and if found not suitable, is divided into neighboring Bézier curves with continuous tangents at the junctions. The complexity of the method depends on the Voronoi construction algorithm and the convex hull algorithm, and is therefore a function of the number of the total obstacles in the map and the complexity of the workspace.

The proposed Collision planner was inspired by the works of [8] for the Overbot autonomous vehicle. The Collision planner however uses the penetration calculation to sample the nearby region to add a control point in order to modify the initial path. The sample is checked for collision and is inserted in the correct sequence so as to create a smooth path that is collision free in a probabilistic manner. By using this directed local sampling approach when obstacles are intersecting, the preprocessing phase of the PRM is omitted, and by utilizing a spline curve the postprocessing is omitted. This allows the Collision planner to be very effective in large environments with cluttered obstacles regardless of their shape, especially when the path connecting the start and goal configuration does not intersect with all of them. At the same time, the algorithm is probabilistically complete with very high convergence rate to the solution due to its' bias towards the goal, and the results indicate that it does not suffer from local minima due to the probabilistic nature of the penetration sampling process.

The paper is organized as follows: Section II formulates the path planning problem and Section III describes the proposed method. In Section IV the simulation experiments are shown, followed by a brief discussion and the conclusions in Section V.

II. PATH PLANNING

The path planning problem in a 2D environment is formulated as [11]

Definition 1: Given a workspace $\mathbb{W} = \Re^2$, an obstacle region $\mathbb{O} \subset \mathbb{W}$ and a robot $\mathbb{R}$ with a given mapping of the configuration space to the workspace $\mathbb{C} \rightarrow \mathbb{W}$, compute a continuous collision free path from q_s to q_g, where $q_s, q_g \in \mathbb{C}_{free}$.

$\mathbb{C}$, $\mathbb{C}_{free}$ and $\mathbb{C}_{obs}$ are defined by the mapping of the configuration space of the robot $\mathbb{R}$ to $\mathbb{W}$, $\mathbb{W} \setminus \mathbb{O}$ and $\mathbb{O}$ respectively.

When mobile robots are concerned, the continuous path must also be subject to various kinodynamic constraints, depending on the structure of the robot wheels. Most mobile robots are differential driven, and the kinematic equations are expressed by the non-holonomic constraints described in eq. 1.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \upsilon \\ \omega \end{bmatrix} \tag{1}$$

where υ and ω are the linear and angular velocity of the mobile robot. These non-holonomic constraints define that the mobile robot must always move tangent to its' orientation and cannot have perpendicular velocity. That constraint is reflected on the continuous path of the planner by G^1 continuity, which implies that the orientation of the robot is also a continuous function along that generated path. A differential robot can follow any path in the configuration space, however, a smooth path causes less odometry drift and therefore minimizes the uncertainty of localization algorithms.

A two stage process is used in most cases where the non-holonomic constraints cannot be directly integrated in the path planner [12].

1) Create a path connecting q_s and q_g using the path planner without any kinodynamic constraints.
2) Perform subdivisions on the path and smoothing until all points of the path can be linked by an admissible collision-free path

Of course this approach is neither optimal, nor complete. A continuous path can be generated by the 1st step which cannot be accessed when the constraints are applied. In such cases, either the algorithm will fail, or new paths must be provided by the planner that will be re-evaluated.

The RRT approaches and the Bump-Surfaces concept on the other hand are able to create a path that directly fulfills the constraints, therefore the 2nd step is unnecessary. The Collision Planner falls into that category, since the kinodynamic constraints can be integrated into the path generation algorithm by the appropriate selection of piecewise splines, so no smoothing or subdivision would be needed to create a path that can be followed by a non-holonomic car-like robot. However, for the rest of the paper, the method described does

not apply the constraints specific to car-like robots (such as curvature criteria), so the paths generated can only be applied to differential driven robots. The application of such kinematic constraints is considered future work and is associated to the sampling process as well as the curve model used to describe the paths.

III. COLLISION PLANNER

The proposed Collision Planner calculates a feasible path connecting the start and goal configuration in a given map with known shape and location of obstacles. No assumptions on the shape of the obstacles is taken, so the obstacles and the robot can be of any shape, convex or nonconvex, polygonal or curved. A partially known map can be used, however the planner must recalculate the trajectory every time the map is altered by the SLAM algorithm.

The Collision Planner utilizes the latest advancements in collision detection and penetration calculation algorithms to compute a feasible path by sampling the neighborhood of the obstacles that obstruct the robot from moving from $q_s = (x_s, y_s, \theta_s)$ to $q_g = (x_g, y_g, \theta_g)$. Initially, a curve with control points (x_s, y_s) and (x_g, y_g) and first derivative boundary conditions used to satisfy the orientation of the robot θ_s and θ_g is used as the candidate solution as it is shown in fig. 1a.

That candidate solution is then checked for collision against the obstacles. If a collision is found, a penetration calculation algorithm is used to calculate the characteristics of that collision. Using that information, N (which is a parameter tuned by the user) different candidate solutions are forked, each using a different sample as a control point generated by that collision. In fig. 1b the first collision wave shows 4 of the N candidate curves (for simplicity) that were forked from the initial path. The process repeats for new collisions of those N candidate solutions without however forking (adding) but by editing the N existing ones. The process re-evaluates the candidate curves by a piecewise interpolation on the new control points, splitting the colliding piece into two different pieces connected by the control point. Since the candidate curves are piecewise splines, the additional control points only alter the generated path locally, so no collision checking is needed on pieces that were already evaluated.

Once a collision wave of N candidates is evaluated and some paths are found collision free, the iteration terminates. The feasible paths are then sorted according to a metric (e.g. length) and the best one is returned. Up to N feasible paths can be generated in each collision wave, so the best path is only optimal with regard to that collision wave. The algorithm is illustrated in Algorithm 1. In fig. 1c the proposed algorithm after two collision waves for $N = 16$ candidate solutions exits with four feasible paths generated by the control points shown by the tangent to the path vectors. The collision-free paths differ largely, however the designer can utilize any metric to sort them and use the best solution applicable to his criteria (length, curvature, distance to the obstacles, etc).

Let us define the initial curve $\mathscr{S}$ to be evaluated for collision detection and penetration calculation as shown in fig. 1a.

$$\mathscr{S} = \{S_1, \ldots, S_i, \ldots, S_k\}, \quad i \in \{1, \ldots, k\} \tag{2}$$

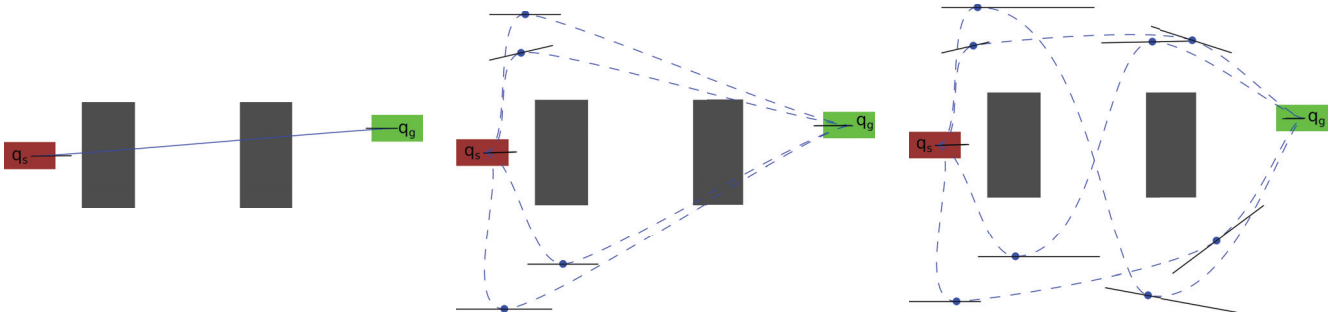
Algorithm 1 Collision Planner

```
S[1] ← ADDCP(q_s, q_g)
if (ISCOLLIDING(S[1],𝕆)) then
    penet ← PENETRATION(S[1],𝕆)
    for (i = 1 to N) in parallel do
        sample ← CREATE_SAMPLE(penet)
        candidates[i] ← SPLIT(S, 1, sample)
        unchecked[i] ← 1
    end for
    abort ← false
    while (abort = false) do
        for (i = 1 to N) in parallel do
            S ← candidates[i]
            for (j =unchecked[i] to SIZEOF(S)) do
                if (ISCOLLIDING(S[j],𝕆)) then
                    penet ← PENETRATION(S[j],𝕆)
                    sample ← CREATE_SAMPLE(penet)
                    candidates[i] ← SPLIT(S, j, sample)
                    unchecked[i] ← j
                    break
                end if
            end for
            if (j =SIZEOF(S)) then
                abort ← true
                solutions ← ATOMIC_PUSH_BACK(S)
            end if
        end for
    end while
else
    solutions[1] ← S
end if
for (i = 1 to SIZEOF(solutions)) in parallel do
    lengths[i] ← CALC_LENGTH(solutions[i])
end for
solutions ← SORT(lengths,solutions)
return solutions[1]
```

Where $\mathscr{S}$ is a piecewise curve consisting of k different Hermite cubic splines $S_i = (x_i, y_i)$.

$$\begin{aligned} S_i &= \begin{bmatrix} x_i(u_i) \\ y_i(u_i) \end{bmatrix}, \quad u_i \in [0,1] \\ x_i(u_i) &= a_{x_i} + b_{x_i} u_i + c_{x_i} u_i^2 + d_{x_i} u_i^3 \\ y_i(u_i) &= a_{y_i} + b_{y_i} u_i + c_{y_i} u_i^2 + d_{y_i} u_i^3 \end{aligned} \tag{3}$$

We chose Hermite cubic splines for the piecewise interpolating curve in order to obtain local shape control and control over the orientation of the robot at the control points. Hermite cubic splines have G^0 and G^1 (slope) continuity, but not curvature G^2 where the spline segments join. Initially (zero collision waves) the curve is comprised of one Hermite cubic spline ($k = 1$).



(a) The initial path (in blue) generated by q_s and q_g control points and their orientation (tangent to the curve), causing collision to both obstacles.

(b) 4 out of the 16 different candidate paths are illustrated for simplicity after the first collision wave. The initial path has been forked to 16 independent piecewise curves.

(c) The 4 feasible paths and their control points (vectors to show the orientation). Two collision waves were necessary to terminate the iterative process.

Fig. 1: The proposed collision planner, where the robot start configuration is illustrated in red, the goal configuration in green and the obstacles in gray color.

A. INITIAL PATH

The initial curve is formulated by the following initial boundary constraints

$$\begin{aligned}\left(x_s, y_s, \theta_s\right) &\rightarrow \left(x_1(0), y_1(0), x_1'(0), y_1'(0)\right)\\ \left(x_g, y_g, \theta_g\right) &\rightarrow \left(x_1(1), y_1(1), x_1'(1), y_1'(1)\right)\end{aligned} \tag{4}$$

The magnitude of the tangent vectors, ${}^{dx_i}/_{du_i}$ and ${}^{dy_i}/_{du_i}$ needed to calculate the Hermite cubic splines coefficients, is chosen to be the distance of the control points of each piece of the curve according to eq. 5.

$$\begin{bmatrix} {}^{dx_i}/_{du_i} \\ {}^{dy_i}/_{du_i} \end{bmatrix} = \left| \begin{bmatrix} x_g \\ y_g \end{bmatrix} - \begin{bmatrix} x_s \\ y_s \end{bmatrix} \right| \cdot \begin{bmatrix} \cos\theta_i \\ \sin\theta_i \end{bmatrix} \tag{5}$$

where θ_i is the orientation of the tangent vector of the control point of the Hermite cubic spline. The slope (orientation) can be calculated using the chain rule

$$\begin{aligned} x_i'(u_i) &= \frac{dx_i}{du_i} = \frac{dx_i}{dy_i}\frac{dy_i}{du_i} \\ y_i'(u_i) &= \frac{dy_i}{du_i} = \frac{dy_i}{dx_i}\frac{dx_i}{du_i} \end{aligned} \tag{6}$$

Eqs. 4 and 5 can be easily converted to the 8 equations needed to calculate the spline coefficients of the initial curve.

$$\begin{aligned} x_1(0) &= a_{x_1} \\ y_1(0) &= a_{y_1} \\ x_1(1) &= a_{x_1} + b_{x_1} + c_{x_1} + d_{x_1} \\ y_1(1) &= a_{y_1} + b_{y_1} + c_{y_1} + d_{y_1} \\ x_1'(0) &= b_{x_1} \\ y_1'(0) &= b_{y_1} \\ x_1'(1) &= b_{x_1} + 2c_{x_1} + 3d_{x_1} \\ y_1'(1) &= b_{y_1} + 2c_{y_1} + 3d_{y_1} \end{aligned} \tag{7}$$

B. SAMPLE GENERATION

The most important part of the Collision Planner that differentiates it from other probabilistic planners is the sampling process. Compared to other sampling algorithms where a uniform sampling over $\mathbb{C}$ or $\mathbb{W}$ is performed, the Collision Planner only samples the areas near the obstacles that are blocking the path connecting q_s and q_g. In order to identify the sampling area, collision checking is performed iteratively that identifies the obstacles blocking the start and goal poses and samples the free space surrounding each of those obstacles. The random point generation is chosen over a deterministic one to assist in complex environments and generate multiple feasible paths that can be evaluated at a latter stage.

The process iterates through collision waves amongst all N candidate paths until at least one candidate is found collision free. The iteration terminates and all feasible paths that belong to that collision wave are evaluated so as to evaluate the best.

1) PENETRATION CALCULATION: In the case of sampling based algorithms, collision checking is an essential part that impacts performance largely. A 'point collision checking' step is necessary for all the samples generated prior to the 'path collision checking' step that is used when trying to connect the nearest neighbors to create the edges of the graph. In [13] the path planner utilizes a proximity algorithm to reduce the collision queries necessary for sampling based path planners. A collision checked sample stores the distance to the nearest obstacle, so that all the samples within that distance generated are marked collision free without having to be checked exclusively for collisions.

Lately, the use of multi-core GPUs in collision detection and penetration calculation (or estimation) have improved the performance of collision queries in 2D and 3D environments significantly [14][15]. Although the penetration calculation algorithm is essential to the proposed path planning algorithm, as a proof of concept the collision detection algorithm uses the R*-tree data structure with each obstacle bounded by 2D rectangular boxes[16]. The penetration calculation algorithm uses a brute-force method for all the points of the colliding obstacle, to simply evaluate the outcome and feasibility of the proposed method. Although the complexity of this approach is large compared to state of the art methods utilizing Bounded Volume Hierarchy such as Bounding Volume Test Tree, k-Discrete Orientation Polytopes, Oriented Bounding Boxes and Axis-Aligned Bounding Boxes, the total number of collision

queries compared to other probabilistic methods are fewer resulting overall in very fast and low complexity convergence. Also, by using a brute force method, there are no limitations or assumptions on the shape of the obstacles, so the Collision Planner can be applied to any obstacle shape, either convex, non-convex or curved.

As with all sampling based algorithms, the Collision Planner utilizes 'point collision checking' as well as 'path collision checking'. However, at the 'path collision checking' step, the penetration characteristics shown in eq. 8 are extracted which are essential to the sample generation process.

In fig. 2 the blue continuous line shows the initial path which intersects with the gray-colored obstacle when the robot is at the configuration q_a shown with the red colored rectangular robot. The collision with that obstacle happens at the point P_{in} (entry point) shown with the white square. The initial path continues to collide with the gray obstacle up to the configuration q_b (exit point) shown by the other red colored rectangular robot. The final collision happens at the point P_{out} shown with the other white square. Those two collision points, P_{in} and P_{out} form the dashed black line (collision line) that separates the workspace in two planes, $Plane_s$ and $Plane_b$. The two penetration depths, h_s and h_b are calculated as the distances of the distal points of the colliding obstacle, shown by the two black vectors perpendicular to the collision line. The yellow vector $\overrightarrow{n}$ is shown to illustrate the perpendicular bisector of the collision line.

$$\begin{matrix} h_s, h_b & h_s \leq h_b \\ P_{in} = \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix} & P_{out} = \begin{bmatrix} x_{out} \\ y_{out} \end{bmatrix} \end{matrix} \tag{8}$$

2) SAMPLING FUNCTION: Utilizing the information extracted by the collision detection algorithm, a sample is generated near the obstacle. In order to identify that area and sample appropriately we use the geometry of the obstacles as shown in fig. 2.

The collision line created by P_{in} and P_{out} defines two collision semi-planes $Plane_s, Plane_b$ and two different penetration depths h_s, h_b the small and big accordingly. Candidate splines must be generated on both sides of the obstacles so we use a Bernoulli distribution with probability density function P_B to decide on which plane each sample lies.

$$P_B = \begin{cases} \frac{h_s}{h_s+h_b} & \text{for } Plane_b \\ \frac{h_b}{h_s+h_b} & \text{for } Plane_s \end{cases} \tag{9}$$

We then use Gaussian distributions to sample the two semi-planes. The Gaussian centers x_c, y_c, θ_c used to generate the boundary conditions of the inserted control point and standard deviation $\sigma_{x_c}, \sigma_{y_c}, \sigma_{\theta_c}$ are different for each plane that the sample lies on according to

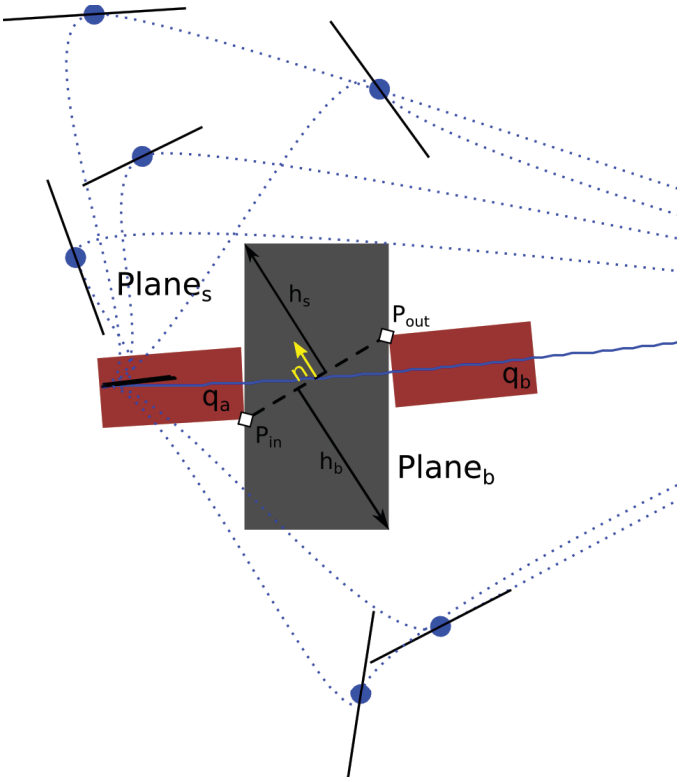


Fig. 2: The collision generated by the grey colored obstacle is used to extract the penetration characteristics. P_{in} and P_{out} are used to create the collision line and h_s, h_b are the distance of the most distal points of the obstacle from that line. The blue large dots shown in both planes are the generated control points while the black coinciding line shows the orientation.

$$\begin{aligned} h_p &= \begin{cases} h_b & \text{if } Plane_b \\ h_s & \text{if } Plane_s \end{cases} \\ \begin{bmatrix} x_c \\ y_c \end{bmatrix} &= \overrightarrow{p}_{in} + \frac{\overrightarrow{p}_{out} - \overrightarrow{p}_{in}}{2} + \overrightarrow{n}(h_p + \frac{w}{2}) \\ \theta_c &= angle\left(\overrightarrow{p}_{out} - \overrightarrow{p}_{in}\right) \\ \sigma_{x_c} &= \max(|x_{out} - x_{in}|, h_p) \\ \sigma_{y_c} &= \max(|y_{out} - y_{in}|, h_p) \\ \sigma_{\theta_c} &= \frac{\pi}{8} \end{aligned} \tag{10}$$

where $\overrightarrow{n}$ is the normal vector to $(\overrightarrow{P}_{out} - \overrightarrow{P}_{in})$ at the midpoint of the segment, w is the robot width. θ_c is needed to properly define the boundary conditions of the Hermite cubic spline we are using to generate the feasible path, and the Gaussian center is defined as the angle of the vector $(\overrightarrow{P}_{out} - \overrightarrow{P}_{in})$ while the standard deviation was experimentally chosen to be $\pi/8$.

The Gaussian centers were chosen in order to minimize the probability of generating samples that are colliding with the same or nearby obstacles. The centers x_c, y_c, θ_c form a configuration that places the robot directly outside of the obstacle in a parallel configuration to the collision line as shown in fig. 2. The deviations however provide the ability

to differentiate and create alternate paths that might be shorter or smoother.

Each samples' boundary conditions $[x_r, y_r, \theta_r]$ is therefore calculated according to

$$\begin{bmatrix} x_r \\ y_r \\ \theta_r \end{bmatrix} \sim \begin{bmatrix} \mathcal{N}(x_c, \sigma_{x_c}) \\ \mathcal{N}(y_c, \sigma_{y_c}) \\ \mathcal{N}(\theta_c, \sigma_{\theta_c}) \end{bmatrix} \tag{11}$$

Each sample is checked for collision and if colliding is recalculated. The algorithm is shown in pseudocode in Algorithm 2. Six different samples are illustrated with the blue dots and their orientation by the black tangent line in fig. 2. Three of those generated control points, although are collision free, generate paths (shown by the dotted blue curves) that continue to collide with the obstacle. This is one of the main reasons why the sampling mechanism is used instead of a deterministic approach. Also, this points out the necessity of performing collision checking on both generated splines and not assuming that collision free samples generate collision free paths.

Algorithm 2 Sample Generation

```
function CREATE_SAMPLE(penetration)
    p_b ← COMP_BERN_DENS(penetration)
    g_c ← COMP_GAUS_CEN(penetration)
    g_s ← COMP_GAUS_STD(penetration)
    repeat
        plane ← COMP_PLANE(p_b)
        sample ← COMP_SAMPLE(plane,g_c,g_s)
    until (ISCOLLIDING(sample) = 0)
    return sample
end function
```

C. SAMPLE INSERTION

After the sample generation step, the generated control point for the detected collision must be used to edit the colliding piecewise curve $\mathscr{S}$. The current candidate path is populated by k Hermite cubic splines as described in eq. 2. Let us examine the insertion of a Hermite spline when the piece of $\mathscr{S}$ that collided is S_i where $1 \leq i \leq k$. The Hermite spline S_i is defined by

$$S_i = \begin{cases} x_i(0), y_i(0), \theta_i(0) \\ x_i(1), y_i(1), \theta_i(1) \end{cases} \tag{12}$$

That spline is removed from $\mathscr{S}$ and two new Hermite splines S_i and S_{i+1} replace it using the generated sample (x_r, y_r, θ_r) incrementing k by one unit.

$$S_i = \begin{cases} x_i(0), y_i(0), \theta_i(0) \\ x_r, y_r, \theta_r \end{cases} \tag{13}$$

and

$$S_{i+1} = \begin{cases} x_r, y_r, \theta_r \\ x_i(1), y_i(1), \theta_i(1) \end{cases} \tag{14}$$

The magnitude of the tangent vectors is calculated using eq. 5 following the same principle. In order to apply kinodynamic constraints to the sample, the magnitude must be tuned so that both pieces generated by the sample insertion fulfill them. The insertion of the Hermite spline is a trivial task and has constant complexity for linked lists as shown in Algorithm 3.

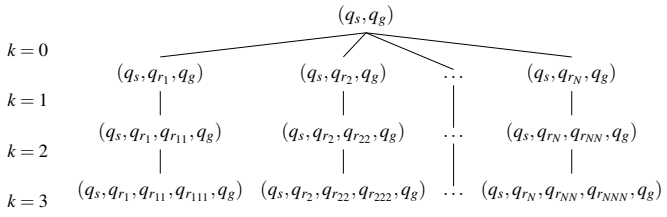


Fig. 3: The N candidate paths generated when $k = 3$.

Algorithm 3 Sample Insertion

```
function SPLIT(S, i, sample)
    [start,end] ← GETCP(S[i])
    INSERT(S,i) ← ADDCP(start,sample)
    INSERT(S,i) ← ADDCP(sample,end)
    REMOVE(S,i)
    return S
end function
```

D. COLLISION WAVE SEARCHING

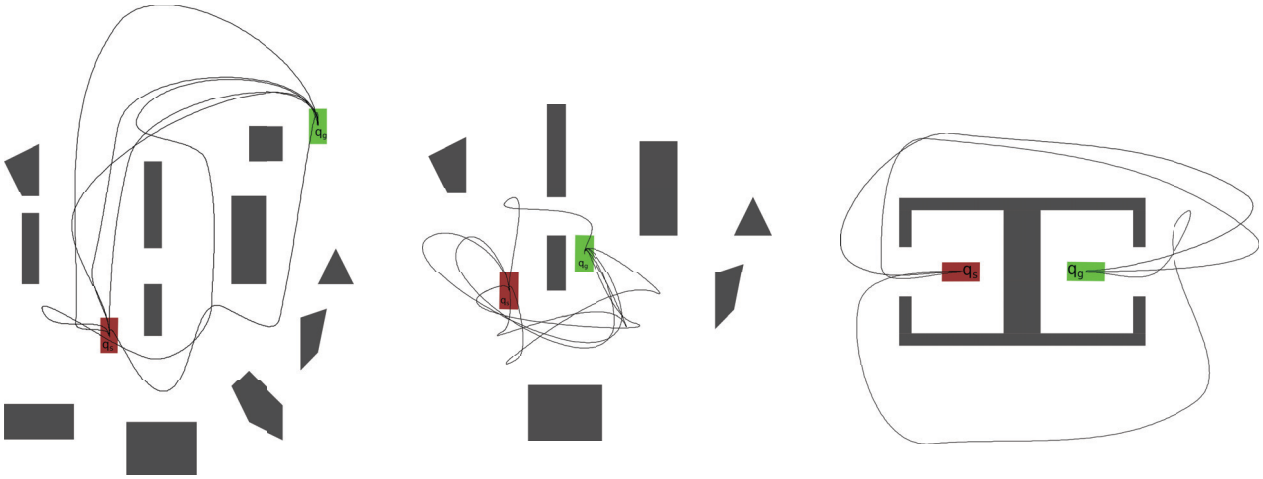
The Collision Planner searches the workspace by collision waves (or collision query generations). Initially, a single candidate solution is evaluated for feasibility. If that candidate collides, N different candidate solutions fork (shown in fig. 3) from that collision and each one of them iterates independently and, if possible, simultaneously via parallelism. The candidate solutions are evaluated according to their generation, meaning that after the 1st collision wave, all candidates are evaluated and if none is found feasible, the second collision wave is evaluated for all of them etc. That way, the path that must dodge the fewer obstacles is preferred over one that must dodge many. At the same time, if many paths are found feasible for the same amount of collision waves, then the best one according to a chosen metric (e.g. the path length) is preferred. This leads to optimal generation, but not to the globally optimum path.

The computational complexity of the proposed algorithm is $\mathrm{O}(kN)$, where k is the minimum number of colliding obstacles (and thus the minimum number of collision waves) and N are the number of candidate solutions the planner has to evaluate.

IV. SIMULATION EXPERIMENTS

In order to evaluate the proposed algorithm, various workspaces have been tested, in serial and parallel implementations. In this paper, three sample maps, shown in fig. 4 are shown with different characteristics and their performance is evaluated by the mean execution time (over 20 repetitions), as shown in table I. In all the maps, the start and goal configurations are illustrated by the q_s and q_g red and green colored rectangular robots respectively, while the obstacles are shown in gray color.

In fig. 4a a complex environment with cluttered obstacles is tested using $N = 16$ candidate solutions. Although the map



(a) A complex map with many obstacles.

(b) A simple map with few obstacles.

(c) Two rooms connected via narrow doorways.

Fig. 4: The start configuration is illustrated in red and the goal configuration in green. The obstacles are shown in gray color while the generated feasible paths are shown in black.

contains 12 obstacles of various shapes and sizes, only four obstacles intersect with the start and goal configurations. The generated feasible paths (shown by the continuous black line) are generated either by only one collision wave or a maximum of five (in another repetition), pointing the fact that the total number of obstacles does not increase the complexity of the proposed algorithm. In fig. 4b a simpler case is used where the start and goal configurations are very close and only one of the seven obstacles intersects with the initial path. In this case, $N = 32$ candidate solutions are used and as a result more feasible solutions are calculated. In the third case, shown in fig. 4c, the map used requires the crossing of two rooms with two narrow passages (the doorways). The obstacles in that workspace are non-convex, without however affecting the outcome of the proposed algorithm. In all the cases the execution times in both the parallel and serial implementations are kept bellow one second on an Intel i7 920 2.66GHz quad-core processor. The algorithms were implemented in C++11, using the Boost Generic Image Library [17] and parallelization with OpenMP [18].

As it is expected, the number of candidate solutions does not always degrade the performance of the algorithm, as the creation of more samples might lead to faster convergence. This is shown in fig. 5, where using the workspace of fig. 4a, the average, minimum and maximum execution time (over 10 repetitions) of 32 candidate solutions is found to be the most appropriate. This result however should not be generalized as it is a function of the complexity of the workspace and further research is needed to fine tune its' value or perform adaptation using learning techniques during the path generation.

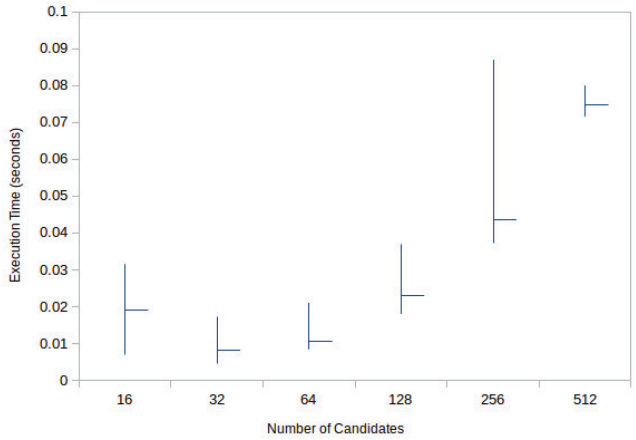


Fig. 5: The average, minimum and maximum execution time against the number of candidate solutions for Map 1 in the parallel implementation.

V. DISCUSSION AND CONCLUSIONS

The proposed Collision planner utilizes collision detection algorithms to generate feasible smooth paths for mobile robots by sampling appropriately near the obstacles obstructing the start and goal configurations. The method computes feasible paths which are not globally optimum, however due to the low computational complexity, it can be used in real time applications with dynamic environments. The family of splines used to model the generated curves can be altered in order to apply different kinodynamic constraints. The results presented verify the feasibility of the generated paths as well as the fast

	Map 1 ($N = 16$)	Map 2 ($N = 32$)	Map 3 ($N = 128$)
Serial	0.0444sec	0.0478sec	0.1543sec
Parallel	0.0175sec	0.0156sec	0.0395sec
Gain (times)	2.533	3.060	3.906

TABLE I: Mean execution time over 20 repetitions in the serial and parallel collision planner.

execution times in the magnitude of milliseconds.

The Collision planner has some limitations that are considered for future research. The Gaussian and Bernoulli sampling distributions used must be evaluated against other probability functions. The collision wave searching algorithm cannot be used if optimality is necessary, so other searching algorithms must be used. Also, a restart mechanism can be used to prune the branches that have been trapped. Finally, the generalization of this method for manipulators and other multi-degree of freedom path planning problems is a topic that must be researched and the authors are currently focused on. All experiments will eventually be tested on real mobile robots using ROS [19], and released under the GPLv3 open source licence.

ACKNOWLEDGMENT

This research was not supported by any research program or grant. Aris Synodinos would like to thank his family for supporting him.

REFERENCES

[1] J.-C. Latombe, *Robot Motion Planning*. Kluwer Academic Publishers, Mar. 1991.

[2] L. Kavraki and P. Svestka, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, 1996. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=508439

[3] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011. [Online]. Available: http://ijr.sagepub.com/content/30/7/846.short

[4] S. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," *Algorithmic and Computational Robotics: New Directions*, pp. 293—-308, 2000. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.38.1387

[5] M. Zucker, J. Kuffner, and M. Branicky, "Multipartite RRTs for Rapid Replanning in Dynamic Environments," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, Apr. 2007, pp. 1603–1609. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4209317http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4209317

[6] E. K. Xidias, P. N. Azariadis, and N. A. Aspragathos, "Two-dimensional motion-planning for nonholonomic robots using the bump-surfaces concept," *Computing*, vol. 79, no. 2-4, pp. 109–118, Mar. 2007. [Online]. Available: http://www.springerlink.com/index/10.1007/s00607-006-0190-2

[7] J. Denny and N. M. Amato, "Toggle PRM: Simultaneous mapping of C-free and C-obstacle - A study in 2D -," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Sep. 2011, pp. 2632–2639. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6095102http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6095102

[8] J. Connors and G. Elkaim, "Analysis of a Spline Based, Obstacle Avoiding Path Planning Algorithm," *2007 IEEE 65th Vehicular Technology Conference - VTC2007-Spring*, pp. 2565–2569, Apr. 2007. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4212956

[9] J.-W. Choi, R. Curry, and G. Elkaim, "Real-Time Obstacle-Avoiding Path Planning for Mobile Robots," *AIAA Guidance, Navigation, and Control Conference*, pp. 1–15, Aug. 2010. [Online]. Available: http://arc.aiaa.org/doi/abs/10.2514/6.2010-8411

[10] S. Sahraei, Alireza and Manzuri, MohammadTaghi and Razvan, MohammadReza and Tajfard, Masoud and Khoshbakht, "Real-time trajectory generation for mobile robots," *AI*IA 2007: Artificial Intelligence and Human-Oriented Computing*, vol. 4733, pp. 459–470, 2007. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74782-6_40

[11] S. M. LaValle, *Planning Algorithms*. Cambridge: Cambridge University Press, 2006. [Online]. Available: http://www.google.com/books?hl=en&lr=&id=Clg8SWNMSRAC&oi=fnd&pg=PR11&dq=planning+algorithms&ots=gYcl9mCrDJ&sig=bsOEFNWzRsnBC8f7Z6zaxxY2xsUhttp://ebooks.cambridge.org/ref/id/CBO9780511546877

[12] A. Sanchez, J. Abraham Arenas, and R. Zapata, "Nonholonomic motion planning for car-like robots," in *Iberoamerican Conference on Artificial Intelligence*, 2002, pp. 1–10. [Online]. Available: http://www.lsi.us.es/iberamia2002/confman/SUBMISSIONS/66-bropezmsan.pdf

[13] J. Bialkowski, S. Karaman, M. Otte, and E. Frazzoli, "Efficient Collision Checking in Sampling-Based Motion Planning," *Algorithmic Foundations of Robotics X*, vol. 86, pp. 365–380, 2013. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-36279-8_22

[14] J. Pan and D. Manocha, "GPU-based parallel collision detection for fast motion planning," *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 187–200, 2012. [Online]. Available: http://ijr.sagepub.com/content/31/2/187.short

[15] J. Yoon, J. Park, and M. Baeg, "GPU-based collision detection for sampling-based motion planning," in *2013 10th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*. IEEE, Oct. 2013, pp. 215–218. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6677345http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6677345

[16] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: an efficient and robust access method for points and rectangles," in *Proceedings of the 1990 ACM SIGMOD international conference on Management of data - SIGMOD '90*. New York, New York, USA: ACM Press, 1990, pp. 322–331. [Online]. Available: http://dl.acm.org/citation.cfm?id=98741http://portal.acm.org/citation.cfm?doid=93597.98741

[17] L. Bourdev, H. Jin, and C. Henning, "Boost Generic Image Library," 2006. [Online]. Available: http://www.boost.org/libs/gil

[18] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," *IEEE Computational Science and Engineering*, vol. 5, 1998.

[19] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, vol. 32, 2009, pp. 151–170.