Safe Navigation in Dynamic, Unknown, Continuous, and Cluttered Environments

Mike D'Arcy, Pooyan Fazli, and Dan Simon

***Abstract*— We introduce PROBLP, a probabilistic local planner, for safe navigation of an autonomous robot in dynamic, unknown, continuous, and cluttered environments. We combine the proposed reactive planner with an existing global planner and evaluate the hybrid in challenging simulated environments. The experiments show that our method achieves a 77% reduction in collisions over the straight-line local planner we use as a benchmark.**

I. INTRODUCTION

Safe navigation through Dynamic, Unknown, Continuous, and Cluttered (DUCC) environments is a crucial ability for autonomous robots in search and rescue, self-driving cars, and servicing tasks. A robot traveling through such environments must make fast decisions to react to moving obstacles, while still finding an efficient path through the partially or fully unknown area.

Path planning approaches can be broadly divided into global and local/reactive methods. Global planners attempt to find a complete path from the robot to the goal. A common way to extend these methods to handle moving obstacles is to add a time dimension to the planning space and modeling obstacles as spacetime volumes. This has the advantage of making it possible to consider the long-term effects of actions but can also result in long planning times in large or complex environments. However, in dynamic environments, standing still for a long time while replanning could result in a collision. In addition, many global approaches expect that a full map of the target area is given *a priori* [1, 6, 15, 17], which is an unrealistic assumption in many scenarios.

The limitations of global planners are often addressed by combining the global planner with a local method. Local or reactive planners only attempt to plan one or more steps in the future, without necessarily finding a complete path to the goal. This results in much faster decision-making for autonomous robots in the vicinity of dynamic obstacles but typically has weak goal-directedness, making it hard to provide completeness or optimality guarantees. Many reactive planners assume deterministic knowledge of obstacles, such as most Velocity Obstacle based approaches [4, 5, 18].

In this paper, we define the safe navigation problem in DUCC environments, with no constraints on the shape or size of the environment or the shape or velocity of the moving obstacles. The proposed local planner, PROBLP, consolidates information about the target and the obstacles detected by the range sensor to identify a safe path in the environment. We run experiments in which we combine PROBLP with the Dynamic Rapidly-exploring Random Tree (DRRT) [3] algorithm to produce a hybrid capable of navigating through complex environments while still remaining safe around moving obstacles. We show empirically that this combined approach significantly outperforms the DRRT algorithm, which by default uses a straight-line local planner.

Mike D'Arcy, Pooyan Fazli, and Dan Simon are with the Electrical Engineering and Computer Science Department, Cleveland State University, Cleveland, OH 44115, USA {m.m.darcy,p.fazli,d.j.simon}@csuohio.edu

II. RELATED WORK

Many existing navigation approaches work only in discrete state spaces. This is the case for D* [16], AD* [11], and also for the SIPP based methods [14, 12]. While it is usually possible to discretize a continuous environment into an occupancy grid [2] or roadmap [9], doing so can require large amounts of memory and processing power, and it destroys some of the information about the environment.

Many of the methods that do work in DUCC environments are based on Rapidly-exploring Random Trees (RRT) [10]. RRT methods use random sampling to build a tree structure over the map. Given enough samples, the RRT will eventually find a path if one exists, so it is called a *probabilistically complete* algorithm. Due to the path being constructed randomly, there is no upper bound on the cost of the path produced in regular RRT, and it can be proven that the probability of the basic RRT algorithm producing an optimal path is zero [7]. This path optimality problem is addressed by RRT* [7], an extension of RRT that will converge to an optimal solution asymptotically as the number of random samples increases. While this improves the speed of path execution, the planning time to find an optimal path can still be quite large, making it unsafe for environments where obstacles may be moving during the planning process. Anytime RRT* [8] was developed to reduce this drawback by quickly finding a suboptimal solution and improving it throughout the execution of the path, but some challenges remain. For both RRT and RRT*, if the path to the goal requires going through a very narrow set of acceptable states, as is the case with a narrow tunnel, the planning time can increase significantly as the probability of sampling one of the acceptable states is low.

In dynamic environments where the initial solution may have to be replanned several times, the potentially high planning time of RRT methods is amplified. A variant of RRT that focuses on efficient replanning with a time dimension is the Multipartite RRT (MP-RRT) algorithm [19], which reuses parts of the previous tree to significantly reduce the required planning time. However, while MP-RRT is

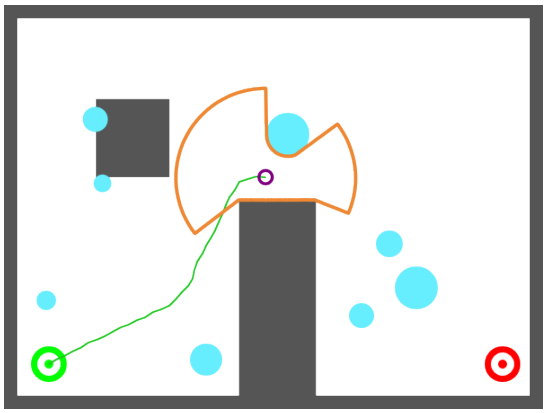


Fig. 1: A sample environment. Static obstacles are dark grey and moving obstacles are light blue. The start point is shown in green on the bottom left of the map, and the target region is shown in red on bottom right. The small purple circle represents the robot, the orange line depicts the robot's range sensor readings, and the green line shows the robot's path.

capable of planning in continuous, unknown environments with moving obstacles and has a low replanning time relative to other RRT methods, the planning time still increases with the size and complexity of the environment due to the nearest-neighbor search over all the nodes in the tree. This makes the approach unsafe for some real-world applications, because obstacles may collide with the robot while it is replanning.

A method that does not plan a complete path to the goal was proposed by Petti and Fraichard [13]. Called Partial Motion Planning (PMP), this approach still uses RRT as the exploration strategy but handles real-time constraints by stopping the growth of the tree when planning time runs out and provides provable safety conditions using the Inevitable Collision States (ICS) concept. PMP provides a significant improvement in real-time planning ability over the global RRT methods, but there remains the possibility that PMP will not find any collision-free trajectories, even when one exists. If there is only a narrow path to escape obstacles, PMP's uniform sampling strategy may not be as effective as an intelligently biased sampling approach.

III. Problem Statement

The problem takes place in a continuous two-dimensional configuration space C with static and dynamic obstacles. Our agent is a holonomic robot that can move at a fixed speed. It has no prior information about the size and shape of the map or the velocity of the moving obstacles in the environment. The robot has a limited-range 360° sensor, with which it can determine the distance to the nearest obstacle in any direction. We additionally assume the robot can perfectly determine its own location and the location of the goal in terms of (x, y) coordinates in the map. The objective of the robot is to navigate from an initial position to the goal, without colliding with any of the static or dynamic obstacles.

Figure 1 shows a sample environment and a robot navigating toward the target point.

IV. Probabilistic Local Planner (ProbLP)

We introduce ProbLP, a probabilistic local planner, to enable an autonomous robot to navigate safely in DUCC environments. The algorithm works by sampling a set of candidate trajectories from the robot's current position and then choosing the best by scoring each candidate on safety and on how much closer it brings the robot to the goal. We define a trajectory as a sequence of waypoints $p_0, p_1, \cdots p_n$, and we say the robot has *executed* a trajectory when it has visited all of the waypoints in order. Our approach does not sample trajectories entirely at random and instead biases the sampling using a probability distribution to increase the likelihood of choosing favorable candidate trajectories. We will first describe the construction of the distribution, and then how the trajectory sampling is performed.

A. Trajectory Sampling: Probability Distribution

The probability distribution f_Θ that the robot uses to sample trajectories is a distribution over direction angles $\theta \in \Theta = [0, 2\pi)$, so $f_\Theta(\theta)$ is the probability of θ being the best direction for the robot to travel next. We construct f_Θ using two other distributions: a *target distribution*, f^g_Θ, and an *obstacle distribution*, f^o_Θ. Each of these are also distributions over $\theta \in \Theta \in [0, 2\pi)$. Note that while f_Θ is a probability distribution, f^g_Θ and f^o_Θ are only pseudo probability distributions, because their integrals do not necessarily sum to 1.

1) Target Distribution: The target distribution f^g_Θ represents the extent to which moving at a direction angle θ would bring the robot closer to the goal. It is defined as a Gaussian distribution:

$$f^g_\Theta(\theta) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{\text{angleDiff}(\theta,\alpha)^2}{2\sigma^2}}, \tag{1}$$

where α is the direction angle between the current location of the robot and the goal, and σ is the standard deviation. The choice of σ will affect the behavior of the robot, with small values making it strongly prefer to go directly towards the goal, and large values making it more willing to take a roundabout path to stay safe around obstacles. `angleDiff(`θ_1`,` θ_2`)` returns the absolute value of the smallest difference between two angles. It can be formally written as:

$$\texttt{angleDiff}(\theta_1, \theta_2) = \min\big((\theta_1 - \theta_2) \bmod 2\pi, \\ (\theta_2 - \theta_1) \bmod 2\pi\big)$$

2) Obstacle Distribution: The obstacle distribution f^o_Θ represents the extent to which moving in a direction θ would move the robot into free space. It is defined as:

$$f^o_\Theta(\theta) = \frac{range(\theta)}{\lambda}, \tag{2}$$

where $range(\theta)$ is the distance between the robot and the nearest obstacle at direction θ, and λ is a scale factor. The

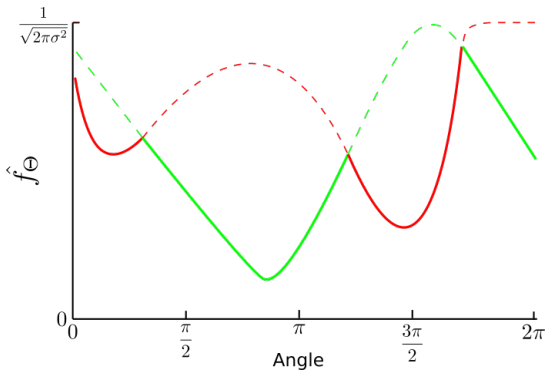


Fig. 2: The red and green lines show obstacle and target distributions respectively. The dashed part is cut away, and the solid line is the resulting final distribution.

specific λ chosen can vary by use case and affects the robot's sensitivity to obstacles, with higher values increasing the sensitivity. In our experiments, we set it to:

$$\lambda = maxRange \times \sqrt{2\pi\sigma^2} \tag{3}$$

where $maxRange$ is the maximum range of the robot's range sensor.

To combine the target and obstacle distributions into the final distribution f_Θ, we take the element-wise minimum and then normalize the area under the curve to be 1:

$$\hat{f_\Theta} = \min(f^g_\Theta, f^o_\Theta) \tag{4}$$

$$f_\Theta = \frac{\hat{f_\Theta}}{\int_{\theta=0}^{2\pi} \hat{f_\Theta}} \tag{5}$$

Figure 2 shows an example of the result of combining two distributions.

B. Trajectory Sampling: Selecting the Candidate Trajectories

We use f_Θ to sample a set of candidate trajectories, from which the robot will select its next movement. Each candidate trajectory is constructed as follows: The first waypoint p_0 is set to the current location of the robot. We then sample an angle θ at random from $[0, 2\pi)$, biased by f_Θ. The location of p_1 is determined by predicting the location the robot would have if it started at p_0 and moved in a direction of θ for Δt seconds. Δt is the computation time allowed for the robot. The remaining waypoints up to p_n are computed in a similar way, with the caveat that a new f_Θ must be constructed for each waypoint.

Recall that f_Θ is composed of the target and obstacle distributions. The target distribution is constructed using the direction angle between the current location of the robot and the goal. However, when picking p_2, we must consider that the robot will start from the previous waypoint p_1 and not from its current position p_0, so the target distribution should be constructed using the direction angle from p_1 to the goal. For the obstacle distribution, we must consider that not only the location of the robot will change, but also time will pass as it progresses along the trajectory. Therefore, we predict the future locations of the moving obstacles to construct an accurate distribution. To this end, we define a predictor function that returns the probability of an obstacle obs occupying position pos at time $t \geq t_0$:

$$P(obs|pos, t) = \min\left(4\frac{1 + (t - t_0)}{1 + d}, 1\right), \tag{6}$$

where t_0 is the current time, and $d = min_{c \in C_{obs}} \|c - pos\|$. C_{obs} is the set of all locations currently occupied by obstacles, which is computed by the robot's range sensor. If no information is available with which to predict future obstacle positions, it can simply be assumed that $\forall t\ P(obs|pos, t) = P(obs|pos, t_0)$, which is trivial to compute directly from the range sensor. However, better predictions improve safety and reduce the need to replan. Likewise, there is no specific reason to prefer the predictor function in Equation 6 to any other obstacle prediction method, but through empirical testing of many possible predictor functions in the experiments in Section VI we found that this function works well.

Using the obstacle predictor function, we can define a $range_i(\theta)$ function giving the predicted range sensor value for angle θ at time $t_i = t_0 + i\Delta t$, with the robot predicted to be located at p_i. This function can be used in Equation 2 in place of $range(\theta)$ to compute the predicted f^o_Θ for future times. Let C^i_{obs} be the set of all points pos such that $P(obs|pos, t_i) > \gamma$, where γ is some cutoff value in the range $[0, 1]$. Then let $C^{i,\theta}_{obs} \subseteq C^i_{obs}$ be the subset of points that lie on the line segment from p_i to $p_i + (maxRange)(\hat{\mathbf{u}})$, where $\hat{\mathbf{u}}$ is a unit vector with direction θ. Then:

$$range_i(\theta) = \min_{c \in C^{i,\theta}_{obs}} \|c - p_i\| \tag{7}$$

The selection of the cutoff value γ can be adjusted depending on the use case, but in general it is most important not to pick a value that is too low. For example, if $\gamma = 0.01$ and there is some uncertainty in future obstacle locations, even points relatively far from obstacles may exceed the small cutoff value, unnecessarily restricting the robot's options. A large value of γ will increase the likelihood of sampling waypoints with low safety scores, increasing the number of samples needed to find a good trajectory, but it is less likely to eliminate desirable trajectories. We found that a cutoff of $\gamma = 0.3$ worked well in our experiments.

In general, when picking waypoint p_i, f_Θ should be computed relative to the location of p_{i-1} and relative to time $t_{i-1} = t_0 + (i - 1)\Delta t$. After f_Θ is constructed for this location and time, p_i can be chosen as described previously, by picking a θ from f_Θ and projecting the location the robot would have after starting at p_{i-1} and traveling in direction θ for Δt seconds. The process is repeated up to waypoint p_n, at which point the generated trajectory is ranked by a combination of distance and safety scores.

C. Trajectory Sampling: Selecting the Final Trajectory

Our algorithm is modular with respect to the metrics used for the distance and safety scores, but we calculated them as follows:

$$Distance\ Score = f_\Theta(\beta) \times \frac{\|p_n - p_0\|}{\sum_{i=0}^{n-1} \|p_{i+1} - p_i\|}, \quad (8)$$

where β is the direction angle between p_0 and p_n, and f_Θ is constructed relative to the current location of the robot and to the current time. The second term in Equation 8 is used for smoothing, as the ratio of straight-line distance to total path length is larger for more straight trajectories.

$$Safety\ Score = \prod_{i=0}^{n} [1 - P(obs|p_i, t_0 + i\Delta t)]. \quad (9)$$

This is equivalent to the probability of the robot being able to follow the trajectory without having any collisions. We set a minimum safety threshold and eliminate all candidate trajectories with a safety score below this threshold. This prevents unsafe trajectories from being considered even if they score highly on distance.

The final score of each trajectory is computed by taking a weighted sum of the distance and safety scores. The safety score is weighted by w and the distance score by $(1 - w)$, where w can be adjusted on a per-use-case basis to make the robot more cautious (high w) or aggressive (low w). The robot picks the trajectory with the highest score and attempts to follow it to completion but continuously updates the estimate of the safety as new information is obtained. If the safety score goes below the minimum threshold, the robot immediately plans a new trajectory to avoid the danger. Otherwise, the robot will not replan until it reaches the terminal waypoint p_n. This makes the robot prefer to continue on its planned smooth path, instead of replanning and changing direction at each decision step.

V. Combining ProbLP with a Global Planner

ProbLP is weakly goal-directed, so the robot will attempt to move directly towards the goal when it is safe to do so. In an environment with complex arrangements of static obstacles, such as a maze or office building, the reactive planner alone may not be able to reach the goal due to the need to backtrack. Therefore, it is desirable to combine ProbLP with a global planner. The global planner plans a path to the goal and divides it into a series of configurations with straight lines between them. The goal for the reactive planner is then to navigate to the next configuration instead of to the target point. This allows the complex long-term path planning to be handled by the global planner, and the reactive planner can remain simple and fast for computing safe trajectories around moving obstacles.

To this end, we selected the Dynamic Rapidly-exploring Random Tree (DRRT) algorithm [3] as the global planner to combine with ProbLP. DRRT is an extension of the RRT algorithm that improves replanning speed by reusing parts of the old tree when regrowing it. DRRT generates configurations to guide our planner through the static map and only considers static obstacles in its plan. The avoidance of moving obstacles is entirely handled by ProbLP to avoid the relatively expensive DRRT replanning. Because ProbLP may deviate significantly from the DRRT path for safety reasons, it will ask DRRT to replan if it cannot reach the next configuration within 10 seconds.

DRRT initializes a tree with the root node at the goal, and then grows the tree from the goal to the robot by repeatedly (1) sampling a random point p, (2) finding the node n_{near} in the tree that is closest to the sampled node, and (3) adding a new node n_{new} to the tree at a distance of at most η along the line from p to n_{near}, as long as the line from p to n_{new} does not intersect any obstacles. This is done until one of the nodes added to the tree is within some distance ϵ of the robot's location. The list of intermediate configurations given to the local planner is the list of ancestors of the node closest to the robot.

To work under limited-vision constraints, the DRRT builds its own internal map for collision checking, which starts empty and adds obstacles as they are observed. When it replans, the tree is first checked for collisions, and any branches that intersect an obstacle are pruned. The pruned nodes are inserted into a fixed-size waypoint cache with random replacement. Then the tree is regrown, and the local planner is re-initialized with the resulting intermediate configurations.

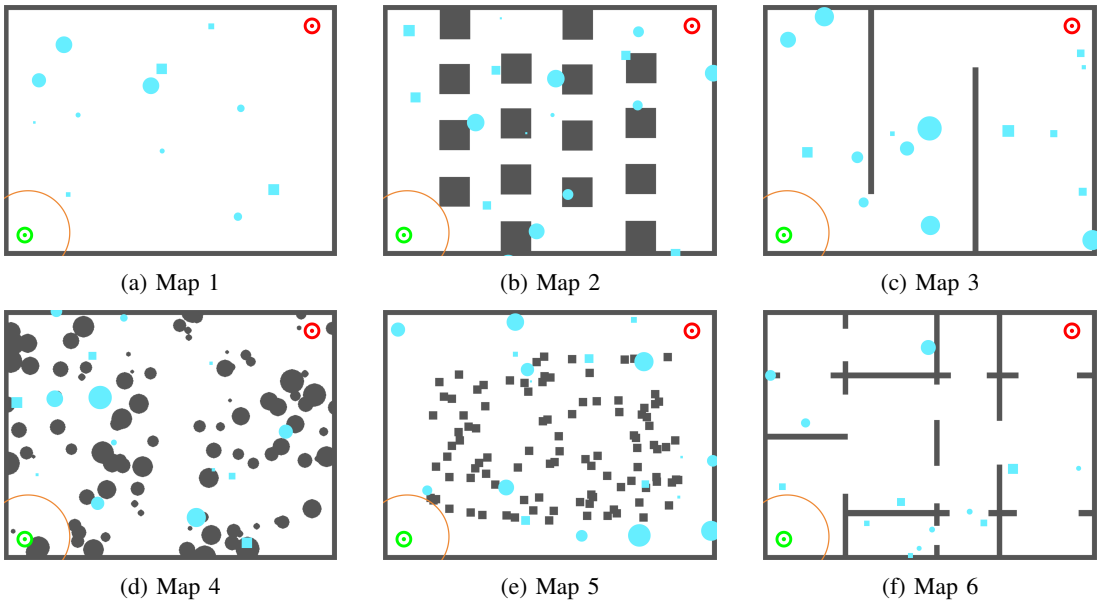
For our experiments, $\eta = 3$ m and $\epsilon = 0.7$ m. The size of the waypoint cache is 200 nodes and the random sampling for growing the tree is biased to pick the robot's location 10% of the time, a random point from the waypoint cache 40% of the time, and a point anywhere on the map the remaining 50% of the time. In addition, after the DRRT finds a path, we smooth it by searching for pairs of nodes along the existing path that could be connected by a straight line, skipping the nodes between them.

VI. Experiments

We evaluate the proposed algorithm in six simulated environments. Each environment has a different static layout, as shown in Figure 3. All maps are 80 m $\times$ 60 m. Twenty dynamic obstacles are also generated randomly for each trial, ten circular and ten square. The obstacles have random sizes ranging between 0.5 m (radius for circles, edge length for squares) and 3 m. Obstacles move randomly around the map, and collisions between obstacles are not considered (i.e., obstacles can overlap and move through each other).

The robot speed was set to a constant 1.0 m/s for all the experiments, but we tested a variety of obstacle settings. These settings can be divided into two *movement modes* and four *speed modes*. The movement modes are as follows:

- **MM-1**: Each obstacle picks a random point on the map and moves straight towards it. When it reaches the point, it picks a new point to move towards, and this continues indefinitely.



(a) Map 1 (b) Map 2 (c) Map 3

(d) Map 4 (e) Map 5 (f) Map 6

Fig. 3: Maps used for experiments. Static obstacles are dark grey and moving obstacles are light blue. The start point is shown in green on the bottom left of each map, and the target region is shown in red on top right. The orange circle represents the robot's sensor range.

- **MM-2**: Each obstacle moves back and forth between two random points picked at the start of the simulation.

The speed modes are as follows:

- **SP-1**: All obstacles move 0.5 m/s (slower than the robot)
- **SP-2**: All obstacles move 1.0 m/s (same as the robot)
- **SP-3**: All obstacles move 1.5 m/s (faster than the robot)
- **SP-4**: Each obstacle moves randomly between 0.5-1.5 m/s

For each trial, all obstacles use the same movement and speed modes (i.e., for a given trial, there cannot be some obstacles using movement mode 1 and others using mode 2), and we evaluate all combinations of maps, movement modes, and speed modes for a total of $6 \times 2 \times 4 = 48$ experimental setups.

The robot is equipped with a range sensor with an effective range of 10 m, and we assume the sensor can distinguish between static and dynamic obstacles. In practice, this could be achieved with an obstacle-tracking algorithm, and this allows the obstacle predictor to be set to $P(obs|p,t) = 1$ if p is an observed static obstacle, and otherwise the predictor remains the same as defined in Equation 6. The robot speed was set to a constant 1.0 m/s for all the experiments.

For scoring candidate trajectories, we set the safety threshold to 0.1 to immediately filter out any trajectories that the obstacle predictor determined had lower than a 10% chance of being collision-free. When taking the weighted sum to combine the distance and safety scores, we set the weights to 0.5 for both distance and safety. The standard deviation, σ, of the target distribution is set to 100, and trajectories are sampled with a fixed length of two waypoints (i.e., two timesteps into the future are considered). In our testing, we found that these settings produce a good balance of path optimality and safety.

We compare our hybrid approach (DRRT-ProbLP) with a hybrid of DRRT and a straight-line local planner (DRRT-SLLP). The straight-line local planner simply travels in a straight line towards its next configuration. Unlike the DRRT in DRRT-ProbLP, which only handles static obstacles and leaves dynamic obstacle avoidance to ProbLP, the DRRT in DRRT-SLLP does handle dynamic obstacles, because the straight-line planner does not have an obstacle avoidance strategy of its own. The straight-line planner asks DRRT to replan if there are any obstacles obstructing its current path within its range of vision.

Our experiments consist of 100 trials of each of the 48 combinations of speed mode and map. Both the DRRT-ProbLP robot and the DRRT-SLLP robot run simultaneously on the same map with the exactly same obstacles, to reduce error in the comparison. We allow a maximum of 5000 time steps to get to the goal before marking the trial as a failure, and also mark the trial as a failure if the DRRT is ever unable to find a path to the goal within 5000 nodes added to the tree (which could happen if a moving obstacle blocks the only path). Failed trials are not used when calculating the averages in the results.

VII. Results and Discussion

Of the 4800 trials, 4375 (91%) were successful and 425 (9%) failed. The results of the successful trials are shown in Table I. In all of the speed modes and map combinations, DRRT-ProbLP had fewer collisions than DRRT-SLLP, and the overall averages were 0.27 collisions for DRRT-ProbLP and 1.20 for DRRT-SLLP. Furthermore, even in SP-3, where obstacles are all faster than the robot, DRRT-ProbLP robot was able to receive less than one collision on average for all maps. The overall average path lengths were

TABLE I: Results of experiments. Numbers represent average of collisions

MAP	DRRT-PROBLP					DRRT-SLLP				
	SP-1	**SP-2**	**SP-3**	**SP-4**	**Average**	**SP-1**	**SP-2**	**SP-3**	**SP-4**	**Average**
1	0.01	0.07	0.43	0.13	**0.16**	0.60	0.91	1.22	0.91	**0.90**
2	0.09	0.19	0.62	0.19	**0.27**	0.50	1.19	1.67	1.28	**1.15**
3	0.05	0.18	0.85	0.25	**0.33**	1.12	1.71	2.46	1.74	**1.75**
4	0.03	0.10	0.65	0.24	**0.25**	0.54	1.06	1.73	1.13	**1.10**
5	0.13	0.14	0.65	0.26	**0.30**	0.39	0.90	1.47	1.02	**0.94**
6	0.07	0.17	0.78	0.31	**0.33**	0.69	1.48	1.89	1.26	**1.33**
Average	**0.05**	**0.13**	**0.68**	**0.24**		**0.65**	**1.15**	**1.72**	**1.22**	

similar for both robots, being 132.09 for DRRT-PROBLP and 128.85 for DRRT-SLLP. This indicates that DRRT-PROBLP robot did not need to deviate far from the global path to achieve the safety improvement.

We also tracked the number of individual trials in which DRRT-PROBLP received fewer collisions than DRRT-SLLP. DRRT-PROBLP had fewer collisions in 2470 trials, DRRT-SLLP had fewer in 285 trials, and in 1620 trials both had the same number. Interestingly, the DRRT-PROBLP robot reached the goal first in 2035 of the trials, with the DRRT-SLLP robot arriving first in 2240. This again indicates that the safety gains came at almost no detriment to path length.

In SP-1, with the obstacles being slower than the robot, DRRT-PROBLP averaged only 0.05 collisions per trial, and had no collisions in 94% of trials. DRRT-SLLP had a much higher average of 0.65, and only 56% of trials were collision-free. In SP-2, the obstacles moved with the same speed as the robot, and DRRT-PROBLP had 0.13 collisions on average, with 88% of trials being collision-free. In this speed mode, DRRT-SLLP had only 32% collision-free trials. Not surprisingly, SP-3 was the most challenging one, as having the obstacles moving faster than the robot meant there could be situations in which the robot simply did not have time to maneuver around the obstacle before being hit. This had a large effect on the performance of DRRT-SLLP, which averaged 1.72 collisions and had collisions in 80% of the trials. Despite the difficulty of this mode, however, DRRT-PROBLP averaged only 0.65 collisions and had zero collisions in 54% of the trials. Speed mode 4 averaged 0.24 collisions and 81% collision-free trials for DRRT-PROBLP, and 1.22 collsisions and 33% collision-free trials for DRRT-SLLP.

Unsurprisingly, both robots had the lowest average number of collisions on Map 1, which has no static obstacles. The averages were 0.16 and 0.90 for DRRT-PROBLP and DRRT-SLLP respectively. The most challenging map for both robots was Map 3, having 0.33 average collisions for DRRT-PROBLP and 1.75 for DRRT-SLLP. The second most challenging map was Map 6, having 0.33 collisions for DRRT-PROBLP and 1.33 for DRRT-SLLP. Maps 2, 4, and 5 performed slightly better, having 0.27, 0.25, and 0.30 average collisions for DRRT-PROBLP, and 1.15, 1.10, and 0.94 average collisions for DRRT-SLLP.

We conjecture that the reason some maps performed better is related to the way the static obstacles are arranged. Maps 2, 4, and 5 have a scattered obstacle layout, with many small static obstacles spread across the map. In contrast, Maps 3 and 6 are more structured, consisting of long, contiguous walls with large free spaces between them. In the scattered-obstacle maps, the static obstacles can be avoided with relatively small adjustments to the trajectory of the robot, making it possible to take a relatively direct path to the goal. In the environments with walls, the robot may have to make a long detour if it finds itself trapped in a dead end, increasing the time spent in the environment and therefore the number of chances to be hit by an obstacle.

We measured the planning times for our implementation of each algorithm and found that DRRT-PROBLP took an average of 117 ms to plan each action, whereas DRRT-SLLP took 149 ms. Intuitively, DRRT-PROBLP should have a higher planning time because it has a more complicated local planner, but this result suggests the opposite. We believe the reason for the superior performance of DRRT-PROBLP is due to the difference in the DRRT for each algorithm. The DRRT in DRRT-PROBLP only needs to consider static obstacles and therefore only needs to replan when new static obstacles are observed, but the DRRT in DRRT-SLLP has to handle dynamic obstacles as well as static obstacles, causing it to undergo the expensive replanning operation much more frequently.

VIII. CONCLUSION AND FUTURE WORK

We have presented PROBLP, a probabilistic local planner, for navigating safely in dynamic, unknown, continuous, and cluttered environments. We showed that this algorithm outperforms the straight-line local planner algorithm we used for comparison.

In future work, we would like to use informed hyperparameter optimization techniques, such as evolutionary algorithms, to tune the algorithm automatically. We would also like to extend our algorithm to multi-robot safe navigation and to non-holonomic robots, and to make it more robust with respect to uncertainties in the range sensor observations and in the positions of the robot and the goal. Finally, because we have so far only developed this approach for use

in 2D environments, we would be interested in extending the algorithm to work in higher-dimensional planning spaces.

IX. Acknowledgment

The authors thank Arash Roshanineshat and Utkarsh Patel for their assistance in developing the simulator.

References

[1] J.P. van den Berg and M.H. Overmars. "Roadmap-based motion planning in dynamic environments". In: *IEEE Transactions on Robotics* 21.5 (2005), pp. 885–897.

[2] A. Elfes. "Using Occupancy Grids for Mobile Robot Perception and Navigation". In: *Computer* 22.6 (1989), pp. 46–57.

[3] Dave Ferguson, Nidhi Kalra, and Anthony Stentz. "Replanning with RRTs". In: *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA*. 2006, pp. 1243–1248.

[4] P. Fiorini and Z. Shiller. "Motion Planning in Dynamic Environments Using Velocity Obstacles". In: *The International Journal of Robotics Research* 17.7 (1998), pp. 760–772.

[5] Oren Gal, Zvi Shiller, and Elon Rimon. "Efficient and safe on-line motion planning in dynamic environments". In: *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA*. 2009, pp. 88–93.

[6] M. Kallman and Maja Mataric. "Motion planning using dynamic roadmaps". In: *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA*. 2004, pp. 4399–4404.

[7] Sertac Karaman and Emilio Frazzoli. "Incremental sampling-based algorithms for optimal motion planning". In: *Proceedings of the Robotics Science and Systems, RSS*. 2010.

[8] Sertac Karaman et al. "Anytime motion planning using the RRT". In: *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA*. 2011, pp. 1478–1483.

[9] Lydia E. Kavraki et al. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *IEEE Transactions on Robotics and Automation* 12.4 (1996), pp. 566–580.

[10] Steven M. Lavalle. *Rapidly-Exploring Random Trees: A New Tool for Path Planning*. TR 98-11. Iowa State University, 1998.

[11] Maxim Likhachev et al. "Anytime Dynamic A*: An Anytime, Replanning Algorithm." In: *Proceedings of the International Conference on Automated Planning and Scheduling, ICAPS*. 2005, pp. 262–271.

[12] Venkatraman Narayanan, Mike Phillips, and Maxim Likhachev. "Anytime safe interval path planning for dynamic environments". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*. 2012, pp. 4708–4715.

[13] Stéphane Petti and Thierry Fraichard. "Safe motion planning in dynamic environments". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*. 2005, pp. 2210–2215.

[14] Mike Phillips and Maxim Likhachev. "SIPP: Safe interval path planning for dynamic environments". In: *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA*. 2011, pp. 5628–5635.

[15] Cyrill Stachniss and Wolfram Burgard. "An integrated approach to goal-directed obstacle avoidance under dynamic constraints for dynamic environments". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*. 2002, pp. 508–513.

[16] Anthony Stentz. "Optimal and efficient path planning for partially-known environments". In: *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA*. 1994, pp. 3310–3317.

[17] Jur Van Den Berg, Dave Ferguson, and James Kuffner. "Anytime path planning and replanning in dynamic environments". In: *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA*. 2006, pp. 2366–2371.

[18] David Wilkie, Jur Van Den Berg, and Dinesh Manocha. "Generalized velocity obstacles". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*. 2009, pp. 5573–5578.

[19] Matt Zucker, James Kuffner, and Michael Branicky. "Multipartite RRTs for rapid replanning in dynamic environments". In: *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA*. 2007, pp. 1603–1609.