Safety Hierarchy for Planning With Time Constraints in Unknown Dynamic Environments

Bruno L'Espérance and Kamal Gupta

***Abstract*—We present a planning algorithm for a mobile robot in unknown indoor dynamic environments. The algorithm takes into account the robot's own dynamics and the dynamic obstacles' future behavior in a safety hierarchy. It also uses an appropriate time horizon for planning and respects the timing constraints on various modules of the planner that are imposed by interleaving of planning and execution. We provide a formulation of this safety hierarchy in a single step via a composite costmap. The planning algorithm has been implemented in Player/Stage. We have run extensive simulations and compared the performance of the full safety hierarchy with various permutations of safety levels, as well as with some baseline algorithms over three different performance measures. We show that our safety hierarchy performs statistically significantly better across most performance measures and most simulation scenarios.**

***Index Terms*—Dynamic obstacles, mobile robot, motion planning, safety.**

I. Introduction

A. Background and Motivation

MOTION planning for dynamic environments has been studied for the past 25 years with significant algorithmic contributions in known dynamic environments (see [1]–[6]). However, a satisfactory planning and execution system for unknown dynamic environments is still elusive. The key issue is how and what kind of safety guarantee in terms of avoiding collisions between the robot and its surrounding objects can be provided for motion planning in uncertain dynamic environments with occlusion.

Because of occlusions in the robot's field of view (FOV) and the uncertainty related to the obstacles' trajectory estimation, a safe planning scheme will need to account for multiple possibilities. For instance, dynamic obstacles can emerge from the unseen region beyond the boundary of the FOV of the robot. On the other hand, the trajectory information of the seen dynamic obstacles needs to be taken into account, although it may be imprecise or uncertain.

Manuscript received December 23, 2013; revised June 6, 2014; accepted August 29, 2014. Date of publication September 25, 2014; date of current version December 3, 2014. This paper was recommended for publication by Associate Editor D. Hsu and Editor D. Fox upon evaluation of the reviewers' comments. This work was supported by the Natural Sciences and Engineering Research Council of Canada through a Postgraduate Scholarship to B. L'Espérance and Discovery Grant to K. Gupta.

The authors are with the School of Engineering Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada (e-mail: blespera@sfu.ca; kamal@sfu.ca).

This paper has supplementary downloadable material avaialable at http://ieeexplore.ieee.org.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TRO.2014.2354933

As originally mentioned in [7] and more recently refined in [8], a robotic system should, at least, consider three criteria when it is planning its future motion: 1) *robot's dynamics*; 2) *dynamic obstacles' future behavior*, which we will refer to as the *model of the future*; and 3) ideally, it should reason over *appropriate time horizons*, both for the duration of the planned trajectory (plan horizon) and for the extent of time for which the model of the future (future horizon) is valid. Most recent contributions to planning in dynamic environments satisfy the first criterion, except for the nearness diagram of [9]. The highly popular dynamic window approach of [10] only satisfies the first criterion since it considers all obstacles in the environment to be static. The extension to the dynamic window approach, proposed in [6], satisfies both the first and the second criteria but does not use appropriate time horizons. Most works, such as [2]–[4] and [6], do not use appropriate time horizons. In all these cases, the plan horizon is chosen arbitrarily without giving any basis for the chosen horizon. Our planning algorithm considers the above-mentioned three criteria and does not chose the time horizons arbitrarily. Note also that most purely reactive methods (with the exception of [11]) do not satisfy the third criterion since their time horizons are typically extremely short, which does not guarantee that they will avoid states for which a collision is certain to occur (for a more complete discussion, see [5] and [7]). The important difference in [11] is that while the plan horizon is short, the future horizon is infinite (see [11] for details) .

Furthermore, in few cases where the planning algorithms do satisfy these three criteria (see [12] and [13]), they arbitrarily choose one particular model of the future, or some other aspects are not considered. For example, the time horizons in [12] are appropriate, but the algorithm only considers a single dynamic obstacle which is not sufficient. In [13], the time horizons are appropriate, but the algorithm does not take into account the occlusion in the robot's FOV or the uncertainty associated with the trajectories of the dynamic obstacles. In fact, many planning algorithms also assume that the robot has precise knowledge about the obstacles' trajectories (see [2], [3], and [6]). Although this might be the case in engineered environments (e.g., if moving objects are other mobile robots), it is not realistic for environments with humans moving around. In such cases, the trajectories of the obstacles need to be estimated (based on sensor data, see [14] and [15]). We refer to this as the *estimated trajectory (ET)* model of the future. The estimated model can also be probabilistic, e.g., [16] use probabilistic velocity obstacles; see [17] and [18] for other probabilistic approaches.

From a safety point of view, the ET model is not satisfactory, at least in its standard deterministic form, because it ignores several other possibilities of the motion of dynamic obstacles.[1] A safer and a more conservative model, which is used in [19], is to use a worst-case scenario where sensed obstacles in the robot's FOV are assumed to move in all possible directions with maximum velocity, i.e., the sensed obstacle expands (in space time) with maximum velocity, starting from the current time. We refer to this as the *sensed worst-case scenario (SW)* model of the future. The down side is, as pointed by the authors themselves, that often, there is no trajectory that does not collide with obstacles in this SW model of the future.

An even more conservative model is the enhancement of the SW model to allow for the possibility that unseen dynamic obstacles too can emerge from the boundary of the robot's FOV (e.g., out of a blind corner), as in [5], where both the dynamic obstacles and every point on the boundary of the robot's FOV are grown (in space time) at a maximum velocity. We will refer to this as *boundary worst-case scenario (BW)* model of the future. As in SW model, it is often the case that there is no robot trajectory that does not collide with the obstacles. The model has also been used more recently inside a reactive planner in [11]. However, an important drawback of their method is, as mentioned by the authors themselves, that their algorithm is not concerned with reaching a particular goal but only with avoiding obstacles reactively.

The above discussion shows that using conservative models of the future is safer; however, often, no feasible trajectories exist, while using less conservative models leads to a solution, but the safety may be compromised. We propose to keep the three models and use them in what we call a *safety hierarchy (SH)*. A key property of this hierarchy is that it attempts to first find a trajectory that does not collide with obstacles under the most conservative model of the future, and if there is no such trajectory, it tries to find one which does not collide with obstacles under the next less-conservative model. We provide a formulation of this SH in a single step via a composite costmap that essentially represents the three models of future. Note that while it is impossible to provide complete guarantee of safety for dynamic unknown environments with occlusion in the FOV and incomplete information (see [13]), our approach presents a useful compromise between safety and the ability of the robot to eventually reach the goal.

Another important point that is often omitted (with some exceptions: [13], [20]–[22]) when dealing with motion planning in unknown dynamic environments is the interleaving of planning and execution and its consequences on different timing parameters. Due to finite plan and future horizons, the robot needs to be able to execute the current trajectory while planning the next trajectory that will be executed at the end of the current one. The planning system should also ensure that the next trajectory to be executed will be computed before the robot finishes the execution of the current one. This consideration has been studied in [13] and [20] in a framework called *partial motion planning (PMP)*. Our planning algorithm also satisfies this condition with a key addition: We respect constraints on the various timing parameters by using a deterministic planner. Both studies [13] and [20] use the PMP framework together with a randomized planning algorithm, which can possibly create some problems. For instance, the randomized planner returns trajectories of different duration. Generally speaking, on an average case basis, a trajectory of longer duration would require more time to compute,[2] and this can possibly lead to a downward spiraling effect where the successively found trajectories are of increasingly shorter durations. This occurs because a shorter duration trajectory implies that there is less time to compute the next trajectory. Without careful considerations, this downward spiraling effect could potentially lead an algorithm to become, more or less, a reactive algorithm. While this has not been reported in the literature, it is clearly a possibility. Our planning algorithm avoids this potential problem by using a deterministic core planner generating trajectories of a fixed duration. This allows us to respect the conditions imposed by interleaving of planning and execution.

B. Contributions

This paper has several contributions. The first one is an SH with respect to multiple models of the future instead of picking a particular model for planning in unknown dynamic environments. Furthermore, this SH is expressed via a composite costmap instead of multiple planners running in parallel. The second contribution is determining constraints (imposed by interleaving of planning and execution) on several time parameters that must be respected, including appropriate times to plan and execute a trajectory by directly linking them to the SH and, therefore, not leaving any time horizons to be chosen arbitrarily. We also combine all these components together in a planning system providing a useful compromise between safety and the ability of the robot to eventually reach the goal. Finally, we have run extensive simulations and compared the performance of the full SH with various permutations of safety levels, as well as with some baseline algorithms over three different performance measures, showing that our SH performs statistically significantly better across most performance measures and most simulation scenarios.

C. Paper Outline

Section II presents the assumptions and an overview of the problem. Section III presents all the details of our SH approach. Section IV presents the timing calculus necessary for the interleaving of planning and execution. Section V presents our planning system architecture. Section VI defines different performance measures and presents the simulation results including the comparison of our full SH with different safety levels and different algorithms. Section VII presents a discussion related to the convergence to the goal of our system. Sections VIII and IX present the conclusion and the directions of our future work, respectively.

[1]Even typical probabilistic models such the ones mentioned above may not account for all possibilities of obstacle motions- for example, a child walking and suddenly darting in a random direction.

[2]This is the case for typical core planners such as RRT (used by [13] and [20]) and the exhaustive core planner used by us.

II. Assumptions and Problem Overview

A. Assumptions

The environment is composed of static obstacles (walls and furniture, for example) and dynamic obstacles (humans moving around). The robot has a finite FOV, and in addition, occlusion often occurs due to obstacles (both static and dynamic) obstructing the line of sight of the robot. We assume that the robot can distinguish between static and dynamic obstacles. We also assume that the dynamic obstacles are non-adversarial and do not avoid the robot.

In the following notation, subscript r denotes a robot parameter, subscript o denotes an obstacle parameter, subscript s denotes a static environment parameter, and superscript c denotes a computation or runtime. The robot is a general wheeled robot (differential drive or car-like, etc.) with a maximum speed of $v_r^{\max}$ and maximum sensing range s_r. In our implementation, we assume that the robot is circular.[3] The robot requires $\Delta_{r,p}^c$ seconds to plan its next trajectory of duration Δ_p (plan horizon). We assume that the sensing apparatus comprises 1) a "static map" module that provides the sensed static map of the environment at time t, denoted by $W_s(t)$; and 2) a "trajectory estimator" module that provides estimates of the trajectories of obstacles in the robot's FOV at time t, denoted by $tr_o(t, \Delta_o)$, for a duration of Δ_o seconds (future horizon). We do not assume any extrapolation of trajectories after Δ_o seconds. This means that the robot has no information about the trajectories of obstacles beyond Δ_o. The trajectory estimator requires Δ_o^c seconds to compute these obstacle trajectory estimates. As mentioned earlier, we assume that this module can distinguish between static and dynamic obstacles in the environment. Note that there is no guarantee that the trajectories given by the estimator module are accurate, i.e., they are only estimates.

These are realistic assumptions. In general, a mobile robot operating in an indoor environment does not have infinite computation power, nor it is able to perfectly forecast the dynamic obstacles trajectories for an infinite period of time. If the dynamic obstacles are assumed to be humans, we have to take into consideration that they can be inattentive. This is why we are assuming that the dynamic obstacles do not avoid the robot.

B. Overview of the Problem

The motion planning problem in unknown dynamic environments with occlusion in the FOV is to find a sequence of trajectories, from the initial configuration at time $t = 0$, that will eventually lead to the final configuration at an unspecified finite time T, that do not collide with the static or the dynamic obstacles and respect the kinematic and dynamic constraints of the robot. Note that the solution cannot simply be a single planned trajectory between the initial configuration and the final configuration because we only have the dynamic obstacles' trajectory information for Δ_o. There are certain constraints that the duration of each trajectory, and the time it takes to compute the trajectory, needs to satisfy. We will discuss these constraints in more detail later in this paper. First, we present our SH approach. In a second step, we will incorporate the timing constraints in a plan execute cycle.

[3]We present a brief description of how our algorithm can be modified to handle the noncircular cases in the future work section.

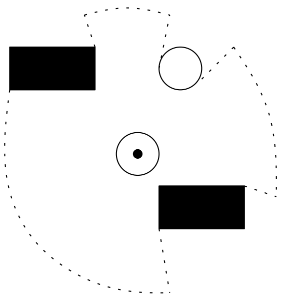


Fig. 1. Robot is represented by the white disk with the black dot. The black rectangles represent static obstacles, and the white disk represents a dynamic obstacle. The dotted lines and curves represent the boundary of the FOV of the robot that takes occlusion into account.

III. Safety Hierarchy Approach

A. Multiple Models of the Future

As mentioned in the introduction, because of occlusions in the robot's FOV and the uncertainty related to the obstacles' trajectory estimation, a safe planning scheme will need to account for multiple possibilities. We outline three models of the future that our SH scheme considers, starting from the most conservative to the less conservative ones.

The first model of the future, referred to as BW, assumes that a dynamic obstacle can emerge from any point of the boundary of the FOV at anytime (see Fig. 1). This model treats all points that are reachable (with velocity $v_o^{\max}$) from the current boundary of the FOV in time Δ_p as obstacles. It also considers the points on the boundary of sensed dynamic obstacles. Computationally, this is achieved by simply growing the boundary of the FOV and the region occupied by the dynamic obstacles in space time[4] at velocity $v_o^{\max}$. Hence, the BW model of the future is in fact represented by a 3-D region $\mathrm{BW} \in \mathbb{R}^2 \times \mathbb{R}^+$. It completely ignores the future trajectory information of dynamic obstacles provided by the trajectory estimator module. Note that this model of the future is the safest possible (most conservative).

The second model, called SW, considers the sensed dynamic obstacles (i.e., it assumes that no unseen obstacles will emerge from the boundary of the FOV) and grows them in space time at velocity $v_o^{\max}$. The SW model is also represented by a 3-D region $\mathrm{SW} \in \mathbb{R}^2 \times \mathbb{R}^+$. This model is less conservative than the BW model. Therefore, we expect the number of situations for which we cannot find a feasible robot trajectory to be smaller than for the BW model of the future.

The third and last model, called ET, simply considers the trajectories given by the trajectory estimator module. It is also represented by a 3-D region $\mathrm{ET} \in \mathbb{R}^2 \times \mathbb{R}^+$.

[4]Note that this expansion is not allowed through the static obstacles. This comment also applies to the SW model presented below.

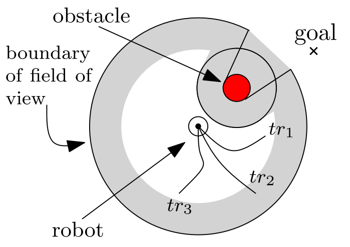


Fig. 2. Light gray represents the region occupied by the BW model of the future after Δ_p seconds. The robot has the choice between three trajectories: tr_1, tr_2, and tr_3; tr_2 collides with BW; hence, it is not chosen, and tr_1 reaches a configuration closer to the goal than tr_3; hence, the robot chooses tr_1.

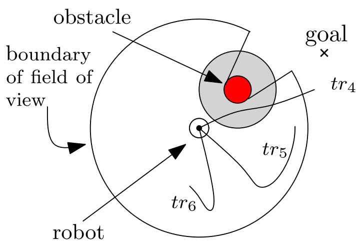


Fig. 3. Light gray disk represents the region occupied by the SW model of the future after Δ_p seconds. The robot has the choice between three trajectories: tr_4, tr_5, and tr_6; tr_4 collides with the SW model of the future; hence, it is not chosen, and tr_6 collides less (the portion of the trajectory which collides with BW is smaller) with BW than tr_5; hence, the robot chooses tr_6. Note that tr_4, tr_5, and tr_6 all collide with BW.

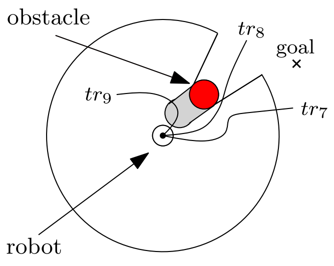


Fig. 4. Light gray region represents the region occupied by the ET model of the future as a function of time. The robot has the choice between three trajectories: tr_7, tr_8, and tr_9; tr_9 collides with ET; hence, it is not chosen, and tr_7 collides less with SW than trajectory tr_8; hence, the robot chooses tr_7. Note that tr_7, tr_8, and tr_9 all collide with SW.

B. Safety Hierarchy

It should be clear that none of these models of the future is perfect. Essentially, the conservative ones often fail to yield feasible robot trajectories, and the less conservative ones can increase the chance of collision. Our SH combines the three, using the less conservative model, only if there is no feasible trajectory under the more conservative one.

For the moment, we will assume that there is a trajectory generator module that takes $\Delta^c_{r,p}$ seconds to return n different trajectories of identical duration Δ_p. Each robot trajectory tr_i can be represented by a 3-D curve in space time $tr_i \in \mathbb{R}^2 \times [t_c, t_c + \Delta_p]$, where t_c is the current time. Using this notation, a collision occurs between the robot's trajectory and one of the models of the future if

$$\exists t \in [t_c, t_c + \Delta_p], \quad \text{s.t.} \quad tr_i(t) \cap \mathrm{BW}(t) \neq \emptyset \tag{1}$$

$$\exists t \in [t_c, t_c + \Delta_p], \quad \text{s.t.} \quad tr_i(t) \cap \mathrm{SW}(t) \neq \emptyset \tag{2}$$

$$\exists t \in [t_c, t_c + \Delta_p], \quad \text{s.t.} \quad tr_i(t) \cap \mathrm{ET}(t) \neq \emptyset. \tag{3}$$

The SH uses the three models of the future as follows (see Figs. 2–4).

Safety hierarchy:

1) Among the n trajectories, find the ones that do not collide with the BW model of the future. If there is no such trajectory, go to step 2. Otherwise, pick the one that reaches the "closest" point towards the goal configuration (see Fig. 2).[5]
2) Among the n trajectories, find the ones that do not collide with the SW model of the future. If there is no such trajectory, go to step 3. Otherwise, pick the one that minimizes the collision cost (collides the least) with the BW model (see Fig. 3).
3) Among the n trajectories, find the ones that do not collide with the ET model of the future. If there is no such trajectory, pick the one that minimizes the collision cost (collides the least) with the ET, SW, and BW models of the future in this order. Otherwise, if there are trajectories that do not collide with the ET model of the future, pick the one that minimizes the collision cost (collides the least) with the SW and BW model of the future in this order (see Fig. 4).

Once again, referring to Figs. 2–4, we can order the different trajectories from the most desirable to the least desirable as follows: $(tr_1, tr_3, tr_6, tr_2, tr_5, tr_7, tr_4, tr_8, tr_9)$.

This particular SH means that the robot will try to reach the goal only when it is safe to do so with respect to the most conservative model of the future (this is achieved in step 1) and will try to minimize collisions with the obstacles otherwise (this is achieved in steps 2 and 3).[6] Note that there might be a case where the best trajectory obtained in step 2 or 3 is still making progress toward the goal if this trajectory minimizes that collision cost. Our approach provides a compromise between safety and the ability for the robot to eventually reach the goal.

C. Safety Hierarchy via a Costmap

The SH, as described above, could be implemented in different ways. For example, one could compute the validity of the n trajectories in each step, one after the other, in $3n$ time cost. In this section, we present an efficient costmap approach to implement the above SH. It allows us to validate n trajectories, determine their cost, and pick one satisfying the SH in a single step.

We maintain a discretized 3-D dynamic costmap array $\mathrm{DCM}(x, y, t)$ in space time, i.e., $x \times y \times t$. It essentially represents the evolution of the environment according to all three models of the future. The bound for the time component is

[5]The precise definition of the closest point will be given later in Section III-C.
[6]Some of the implications of this will be examined in Section VII.

simply Δ_p. The bounds for the space components are the maximum of either the robot's sensing range s_r or $\Delta_p * v_r^{\max}$ and are centered at the robot's location at the current time t_c. Let δ_r be spatial resolution of the costmap. The time resolution δ_t is then chosen to be $\min\{\delta_r / v_r^{\max}, \delta_r / v_o^{\max}\}$. This ensures that neither the robot nor the obstacles travel more than one discrete cell in one-time step. The costmap DCM is then built as follows.

First, we create a global static costmap (SCM), a 2-D array of the environment, which captures the topology of the space. This is accomplished by running the numerical navigation function NF1, in space only (see [23]) with respect to current static map $W_s(t_c)$ of the environment, where t_c denotes the current time. $W_s(t_c)$ is a discretized bitmap representation of the environment in which the unknown part of the environment is considered free. This allows our algorithm to use the NF1 method even if we do not have the complete static map at the current time t_c. This technique has been used by Brock and Khatib [24], where they experimentally showed that the robot is able to reach the goal without getting stuck in U-shaped obstacles. Here is a brief description of the resulting NF1 navigation function computation. All the obstacles in the static map are inflated by the radius of the robot. Then, a special value (typically -1) is assigned to the cell occupied by the known static obstacles, a large positive value is assigned to all the free or unknown cells, and a numerical value of 0 is assigned to the goal position. Using a wavefront expansion (starting from the goal), the neighboring cells of a cell with already assigned value are assigned a value incremented by one (L_1 metric). The wavefront is forbidden from expanding in cells occupied by static obstacles but is allowed to expand in the free or unknown cells. A lower value at a cell in the SCM implies that the cell is closer to the goal (note that this is not the usual Euclidean distance). For each time slice of the DCM array, we copy the values of the 2-D SCM array (computed for $W_s(t_c)$) centered at the robot and extending up to the limits of the DCM array. Now, the DCM array is identical at each time slice. The reason for doing this is that the SCM is used to determine the spatial distance of a cell from the goal.

Second, we inflate the BW and SW models of the future by the robot's radius for the first time slice. Then, at each time slice, we add a cost value of c_{bw} to the cells that are occupied by the BW model of the future. We determine these cells by using a wavefront expansion, in space time, from the boundary of the FOV, which is essentially given by the sensor scan, and the currently sensed dynamic obstacles starting at the first time slice and propagating at velocity $v_o^{\max}$. Third, we add a cost of c_{sw} to the cells that will be occupied by the SW model of the future. We determine these cells by using a wavefront expansion, in space time, from the currently sensed dynamic obstacles at the first time slice and propagating at velocity $v_o^{\max}$. Finally, we add a cost of c_{et} to the cells that will be occupied by the ET model of the future inflated by the robot's radius. We determine these cells by using the estimated trajectories tr_o given by the trajectory estimator module. The pseudocode for the SH Costmap algorithm, i.e., SHCostmap(), is presented in Algorithm 1.

Algorithm 1: Safety Hierarchy Costmap

```
$SHCostmap(W_s(t_c), tr_o(t_c, \Delta_p), goal)$ {
$SCM = \text{ConstructSCM}(W_s(t_c), goal)$
**for** ($t = t_c$ *to* $t = t_c + \Delta_p$) **do**
    $DCM(x_{low} : x_{high}, y_{low} : y_{high}, t) = SCM$
**end**
**for** ($t = t_c$ *to* $t = t_c + \Delta_p$) **do**
    **for** ($x = x_{low}$ *to* $x_{high}$) **do**
        **for** ($y = y_{low}$ *to* $y = y_{high}$) **do**
            **if** $(x, y, t) \in BW$ **then**
                $DCM(x, y, t) += c_{\text{bw}}$
            **end**
            **if** $(x, y, t) \in SW$ **then**
                $DCM(x, y, t) += c_{sw}$
            **end**
            **if** $(x, y, t) \in ET$ **then**
                $DCM(x, y, t) += c_{\text{et}}$
            **end**
        **end**
    **end**
**end**
} return DCM;
```

Assume that a discretized trajectory has m time steps, i.e., $\Delta_p = m * \delta_t$. Then, the cost of a trajectory tr is given by

$$c(tr) = \begin{cases} \text{DCM}(tr(m * \delta_t)), & \text{if } tr \in \mathcal{F} \\ \sum_{t=t_c}^{t_c + m*\delta_t} \text{DCM}(tr(t)), & \text{otherwise} \end{cases} \quad (4)$$

where $\mathcal{F}$ is the subset of $\mathbb{R}^2 \times \mathbb{R}^+$ which does not overlap with the BW, SW, and ET models of the future.

The trajectory with the lowest cost is chosen. If a trajectory does not collide with any model of the future, its cost is the NF1 value of the cell where the trajectory ends. Recall that the NF1 value of a cell essentially dictates the closeness of the cell to the goal. Hence, choosing the lowest cost implies that the trajectory that leads closer to the goal is preferred. Therefore, the notion of *closest point* to the goal is defined with respect to the value of the NF1. As we will show below, this takes care of step 1 of our SH. In the event that all trajectories collide with a particular model of the future, we must still choose one. For this, we need an ordering with respect to "collision cost." We choose this as the number of cells along the trajectory that are in collision as our ordering metric. Hence, the definition of a trajectory *colliding less* with a particular model of the future than another trajectory is then expressed with respect to the number of (x, y, t) cells which are colliding with this model.

We show below that with an appropriate choice of c_{bw}, c_{sw}, and c_{et}, the lowest cost trajectory exactly satisfies the SH described above. We obtain the following results.

Proposition: Let $\underline{nf1}$ and $\overline{nf1}$ be the minimum and maximum value, respectively, of the NF1 navigation function over the static map, and let

$$c_{\text{bw}} = m * (\overline{nf1} - \underline{nf1}) + 1 \quad (5)$$

$$c_{\text{sw}} = m * (\overline{nf1} + c_{\text{bw}} - \underline{nf1}) + 1 \quad (6)$$

$$c_{\text{et}} = m * (\overline{nf1} + c_{\text{sw}} - \underline{nf1}) + 1. \quad (7)$$

Then, the minimum cost trajectory satisfies the SH.

Proof: The proof involves ten different cases to verify. These different cases are directly coming from the SH. For example, there are four cases embedded in step one of the SH. We have to verify that a trajectory, which is free of collision, will always be chosen over a trajectory that collides with any model of the future (three cases). We also have to verify that among all the trajectories, which are free of collision, the one that reaches the closest point towards the goal will be chosen. Due to lack of space, we are only presenting the proof for three of the ten cases.

Case 1: Assume that we have two trajectories tr_1 and tr_2, such that tr_1 reaches a point closer to the goal configuration than tr_2 and both trajectories do not collide with BW. In this case, according to step one of the SH, we should pick trajectory tr_1. Hence, we need to show that $c(tr_1) < c(tr_2)$. Since $tr_1, tr_2 \in \mathcal{F}$, we are using the upper branch of the cost function, $\text{DCM}(tr_i(m * \delta_t))$. Once again, since $tr_1, tr_2 \in \mathcal{F}$, the costmap evaluated at $m * \delta_t$ only consists of NF1 values. By assumption, tr_1 reaches a point closer to the goal than tr_2, and by definition of *closest point*, it means that tr_1 reaches a lower NF1 value, hence $c(tr_1) < c(tr_2)$. This completes the proof of case 1.

Case 2: Assume that we have two trajectories tr_1 and tr_2, such that tr_1 does not collide with the BW model of the future and tr_2 does. We then need to show that $c(tr_1) < c(tr_2)$. Toward this end, we compare the highest cost possible for tr_1 with the lowest cost possible for tr_2. The highest cost for tr_1 is $\overline{nf1}$. Hence

$$\begin{aligned} c(tr_1) &\leq \overline{nf1} < m * \overline{nf1} + 1 \\ &= m * (\overline{nf1} + \underline{nf1} - \underline{nf1}) + 1 \\ &= m * \underline{nf1} + m * (\overline{nf1} - \underline{nf1}) + 1 \\ &= m * \underline{nf1} + c_{\text{bw}} \\ &\leq c(tr_2). \end{aligned}$$

The last line is true since $m * \underline{nf1} + c_{\text{bw}}$ is the lowest possible cost for a trajectory that collides with the BW model of the future. This completes the proof of case 2.

Case 3: If both tr_1 and tr_2 collide with the BW model of the future but both do not collide with the SW model of the future and tr_1 collides a smaller number of time with BW than tr_2, then we should have $c(tr_1) < c(tr_2)$. Once again, we will compare the highest cost possible for tr_1 with the lowest cost possible for tr_2. Let p_1 and p_2 be the number of cells along tr_1 and tr_2 that are in collision with BW, respectively. The maximum cost for tr_1 is $m * \overline{nf1} + p_1 c_{\text{bw}}$, and the minimum cost for tr_2 is $m * \underline{nf1} + p_2 c_{\text{bw}}$. We need to verify that

$$\begin{aligned} & m * \overline{nf1} + p_1 c_{\text{bw}} < m * \underline{nf1} + p_2 c_{\text{bw}} \\ \Leftrightarrow\ & m * (\overline{nf1} - \underline{nf1}) < (p_2 - p_1) c_{\text{bw}} \\ \Leftrightarrow\ & c_{\text{bw}} - 1 < (p_2 - p_1) c_{\text{bw}} \\ \Leftrightarrow\ & -1 < (p_2 - p_1 - 1) c_{\text{bw}}. \end{aligned}$$

The last line is true because the right-hand side is greater than or equal to zero since $p_2 \geq p_1 + 1$. This completes the proof of case 3.

The other cases are similar in nature and, so are their proofs. ■

IV. Timing Calculus With Timing Constraints

As pointed out in Section II, the solution to the motion planning problem in unknown dynamic environments is complicated by the fact that it is neither safe nor realistic to expect the robot to plan the entire trajectory between the initial configuration at time $t = 0$ and the final configuration at an unspecified time $T < +\infty$. Instead, the robot must plan a sequence of shorter duration trajectories and execute them, in an interleaved sequence of planning and execution that will eventually lead to the final configuration. In particular, the robot must plan for the future trajectory as it is executing the current one (which, of course, was planned at an earlier time). In this section, we explain the interleaving of planning and execution and derive important bounds imposed by the nature of the motion planning problem.

Assume that at time t_c (current time), the first trajectory over the interval $[t_c, t_c + \Delta_p]$ has already been planned and is available to the robot. At time t_c, the robot starts executing this trajectory and, simultaneously, starts planning the next trajectory. Although the first trajectory was planned for a duration of Δ_p, the robot will only execute it for a duration of

$$\Delta_e \leq \Delta_p \quad (8)$$

while replanning the next trajectory. The inequality embodies the fact that we do not want to fully execute the planned trajectory and would like to replan well before Δ_p. A key reason is that the planned trajectory (planned during t_c and $t_c + \Delta_e$) is based on estimates (available at time t_c) of obstacle trajectories and the estimates can differ from the actual trajectories. Since the robot cannot change the planned trajectory until the next plan is available, Δ_e should be kept small. This also gives the robot a safety buffer of $\Delta_p - \Delta_e$, in case there is an unforeseen collision occurring just after Δ_p. The subsequent trajectories have to respect the same interleaving of execution and planning until the robot reaches the goal; as the robot executes the current trajectory of duration Δ_e, it simultaneously plans the next trajectory of duration Δ_p (see Fig. 5). For example, the first planned trajectory over $[t_c, t_c + \Delta_p]$ is only executed for over $[t_c, t_c + \Delta_e]$. This trajectory was based on the ET of the obstacle (obtained at time $t_c - \Delta_e$) given by the dotted blue line over $[t_c, t_c + \Delta_p]$. Now at time t_c, the robot gets a new estimation for the obstacle's trajectory over $[t_c + \Delta_e, t_c + \Delta_e + \Delta_p]$. The robot will plan its next trajectory based on this information. The process continues until the robot reaches the goal. Note that

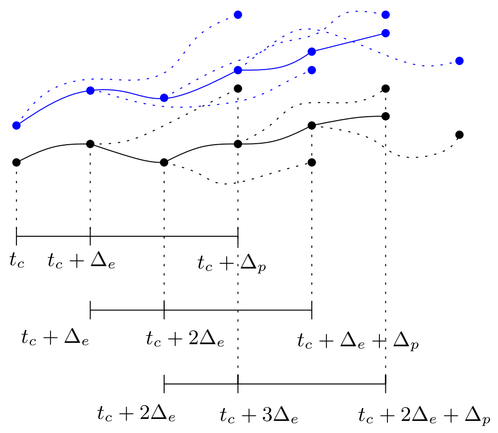


Fig. 5. Interleaving of planning and execution. The robot's trajectory is represented by the black lines. The solid line segments are the executed part of the robot's trajectory. The dotted line segments are the planned part of the robot's trajectory that is not executed. The obstacle's trajectory is represented by blue lines (gray in the black and white version). The solid line segments are the actual obstacle's trajectory, while the dotted line segments represent the ET of the obstacle, the estimate having been made at some previous time.

obstacle's actual trajectory (solid blue line) might differ from the estimated one (dotted blue line). This does not mean that the robot will collide with the obstacle because the SH is taking care of this issue.

The next condition that needs to be satisfied is that the next trajectory to be executed by the robot must have been planned before the end of the current trajectory being executed. Since it takes Δ_o^c seconds for the trajectory estimator module to compute the obstacles' trajectories and it takes $\Delta_{r,p}^c$ seconds for the robot to compute the trajectories of duration Δ_p, the duration of the current executable portion of the trajectory, which is denoted Δ_e, should satisfy

$$\Delta_e \geq \Delta_o^c + \Delta_{r,p}^c. \tag{9}$$

Otherwise, the robot will have no trajectory to execute at the end of the current one. In a purely static environment, the robot could simply apply a braking maneuver and take whatever time is necessary to compute the next trajectory, but it cannot do this in dynamic environment. Now, since we only have estimates of the obstacles trajectories and that once the robot starts executing a trajectory it cannot change it, it will be safer to have Δ_e as small as possible; hence

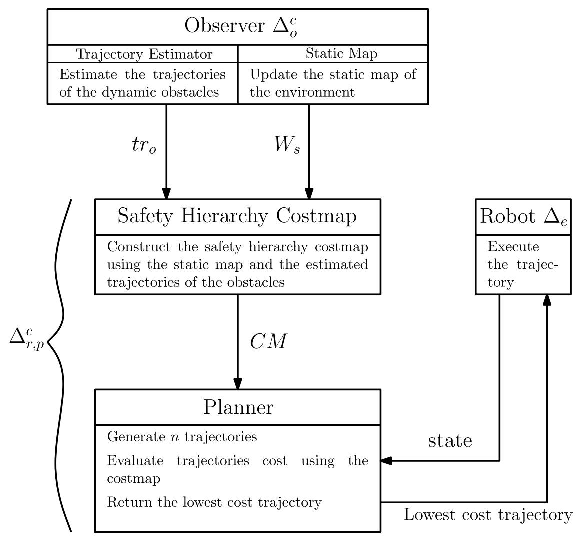
$$\Delta_e = \Delta_o^c + \Delta_{r,p}^c. \tag{10}$$

On the other hand, it would make sense to use as much of the information we have about the dynamic obstacles as possible, and hence, we should plan the trajectory for

$$\Delta_p = \Delta_o - \Delta_e \tag{11}$$

even if we expect to execute it only for Δ_e seconds. Recall that Δ_o is the duration of the estimated trajectories given by the trajectory estimator module and that the robot has no information about these trajectories beyond $t_c + \Delta_o$.

Another condition that needs to be satisfied by Δ_p is directly related to the SH. Let s_r be the range of the robot' sensor, and



Observer Δ_o^c
Trajectory Estimator
Estimate the trajectories of the dynamic obstacles
Static Map
Update the static map of the environment
tr_o
W_s
Safety Hierarchy Costmap
Construct the safety hierarchy costmap using the static map and the estimated trajectories of the obstacles
Robot Δ_e
Execute the trajectory
$\Delta_{r,p}^c$
CM
Planner
Generate n trajectories
Evaluate trajectories cost using the costmap
Return the lowest cost trajectory
state
Lowest cost trajectory

Fig. 6. Planning system architecture.

$\Delta_s = s_r / v_o^{\max}$; then, Δ_s represents the time necessary for the relevant space around the robot to be filled by the BW model of the future. This means that, in general, any trajectory of duration longer than Δ_s will collide with the BW model of the future. Hence, if we expect to make any use of the BW model of the future, we need to have $\Delta_p \leq \Delta_s$. In fact, we need to have $\Delta_p < \Delta_s$, because, for the equality condition, the least cost trajectory for the robot is simply to stay stationary. Using this new condition together with (11), we obtain the following general condition:

$$\Delta_p < \min\{\Delta_o - \Delta_e, \Delta_s\}. \tag{12}$$

In summary, the conditions that all the timing parameters have to respect are

$$\Delta_o^c + \Delta_{r,p}^c = \Delta_e \leq \Delta_p < \min\{\Delta_o - \Delta_e, \Delta_s\}. \tag{13}$$

Now at each planning cycle, while the robot is executing the current trajectory, the planner will generate n trajectories all of duration Δ_p satisfying the above conditions. It will then pick the lowest cost trajectory based on the SH presented earlier.

V. Planning System Architecture

The overall planning system architecture comprises four basic modules: the observer, the SH costmap, the planner, and the robot (see Fig. 6). The SH costmap module has already been extensively explained earlier in this paper. The robot modules simply executes the current trajectory and returns the robot's state to the planner module. We now explain the other modules and the system architecture flow.

A. Observer

The observer module comprises two submodules: the trajectory estimator and the static map. The trajectory estimator module simply estimates the trajectories of the dynamic obstacles for a duration of Δ_o and passes this information to the

SH costmap module. The static map module simply updates the static map of the environment and passes it to the SH costmap module. In the future, the observer module will be implemented using real sensors information; for the simulation, this information is simply given to the SH costmap module. For example, trajectory estimation has been successfully done using a 2-D laser scan in [25] and [26].

B. Planner

Two key characteristics of the core planning module needed by our scheme are that it should generate trajectories of a specified duration Δ_p, and this should be accomplished within a fixed duration of runtime. There is no guarantee that a standard randomized planner like RRT would return any trajectory of required duration even if we impose the runtime restriction. While one could design variations of the standard RRT, we would also like to emphasize that sampling-based techniques such as RRT are primarily designed for high-dimensional space, which is not the case here. As a consequence, we have experimented with an exhaustive planner that tries out all possible trajectories with a given control resolution as follows.

We discretize the time duration Δ_p in k equal time steps. This is the number of times that we will allow the planner to modify the control given to the robot. Note that we can have $k = m$, but it is generally smaller for computational reasons (see below). We discretize the space of controls in p different controls. At each time step, we select a control from the discretized control space. A trajectory is then a sequences of k controls. The number of possible trajectories $n = p^k$ (p is itself exponential in the dimension of the control space). At each planning cycle, we calculate the cost of these p^k trajectories using the costmap and take the one with the lowest cost. This technique has the advantage that we can find an upper bound on the computation time of these p^k trajectories by expressing them in terms of number of collision checks. Of course, the drawback of this method is the exponential growth of the number of possible control sequences. This could be a problem in high-dimensional spaces but with modern CPU's and realistic scenarios for Δ_p (say less than 4 s), the exhaustive planner can be run quite fast (see Section VI for details).

C. System Architecture Flow

At the beginning of each planning cycle, the robot provides its state to the planner module. At the same time, the observer module provides the ET of the obstacles tr_o and the updated static map W_s to the SH costmap module. This is accomplished in Δ_o^c seconds. Using this information, the SH costmap module constructs the costmap DCM and passes it to the planner module. Using the robot's state, the planner module generates n trajectories, each of duration Δ_p, and using the costmap DCM [via (4)], it evaluates their respective cost. The lowest cost trajectory is passed to the robot to execute, and a new planning cycle can begin (once again, see Fig. 6).

VI. Simulation Results

We have implemented the entire planning system using C++ and Stage simulator [27]. In order to test the effectiveness of our approach, we ran a total of more than 800 simulations. We explain and present the results of these simulations in this section.

A. Obstacles Motion Model, Sensor Model, and Estimator Module

We have created three different obstacles' motion models for our simulation. For all models, the obstacles do not try to avoid the robot, but they are not chasing the robot either. For the first two models, if there are no obstacle in their way, the dynamic obstacles are moving along linear trajectories for 2 s, after which, they change their direction by choosing a random angle α and move along this new linear trajectory for another 2 s. If there are obstacles in their way, they execute an heuristic avoiding manoeuvre. In the first motion model, we have $\alpha \in [-120°, 120°]$. This creates dynamic obstacles with large abrupt changes in their direction. We call this motion model *highly erratic model* (see VIDEO 1). In the second motion model, we have $\alpha \in [-30°, 30°]$. This creates dynamic obstacles trajectories closer to linear trajectories. We call this motion model *less erratic model*. In the third model, some dynamic obstacles have periodic motions and emerge from behind different static obstacles in the map. We call this model *periodic model*. VIDEO 2 contains obstacles with the less-erratic and the periodic models.

The robot' sensor is a line scan range sensor with a sensing range of 6 m and an FOV of $360°$. It does not sense obstacles that are occluded or are beyond its FOV.

Finally, the trajectory estimator module assumes that the dynamic obstacles in the environment will move with linear trajectories with the current velocity until the next estimation cycle. This means that the trajectory estimator module does not exactly predict the obstacle trajectories (realistic scenario) since the obstacles do not move along linear trajectories.

B. Different Algorithms

To show the effectiveness of the SH, we need to compare its performance with other alternative algorithms. The first point to verify is whether it is really necessary to use all three models of the future (ET, SW, and BW) or whether we could use some combination of either only one or two of them. To this end, we compare the full SH (using the three models of the future) with the six different other combinations of the models of the future (algorithms 4–9 in the list below). All these different variations of the SH algorithm are using $\Delta_e = 0.8\,\text{s}$.

The second point to verify is whether a simpler algorithm which is faster to compute and, hence, updates its planned trajectory at a much faster rate would nullify the need for a SH. To this end, we have created a fast algorithm, called Fast ET (F-ET, algorithm 2 in the list below), which uses the same exhaustive planner without the costmap (the most intensive part of the computation in the SH algorithm) and only with the ET model of the future. This allows us to use $\Delta_e = 0.2$ s, which is four times faster than the Δ_e needed for the SH algorithm and its different variations. The rest of the parameters are exactly the same for all algorithms.

Finally, we also include a variant of F-ET, i.e., F-ET with perfect sensor and estimator; sensor can sense even occluded obstacles (within its sensing range) and estimator provides exact trajectories of obstacles. We call this algorithm perfect fast ET (PF-ET). This is a benchmark algorithm that allows to verify whether the collisions of the other algorithms occur because the environments are extremely complicated (in which case the PF-ET algorithm would also collide) or whether the collisions occur because of imperfect sensing and estimation of obstacle trajectories.

In summary, we ran the simulations for two other algorithms, the full SH algorithm, and six different variations of the SH, as listed below:

1) perfect Fast ET, PF-ET with ($\Delta_e = 0.2$ s);
2) fast ET, F-ET with ($\Delta_e = 0.2$ s);
3) full SH with ($\Delta_e = 0.8$ s);
4) only ET model, O-ET with ($\Delta_e = 0.8$ s);
5) only SW model, O-SW with ($\Delta_e = 0.8$ s);
6) only BW model, O-BW with ($\Delta_e = 0.8$ s);
7) ET and SW models, ET + SW with ($\Delta_e = 0.8$ s);
8) ET and BW models, ET + BW with ($\Delta_e = 0.8$ s);
9) SW and BW models, SW + BW with ($\Delta_e = 0.8$ s).

C. Performance Measures

We consider three different performance measures to compare the different algorithms outlined in the previous section.

The first performance measure is the percentage of simulations for which the robot reaches the goal without colliding with any obstacles, i.e., percentage of collision free runs (PCFR). As already mentioned, it is impossible to guarantee safety in unknown dynamic environments, and hence, this measure provides a useful safety comparison across different algorithms.

The second performance measure counts the average number of collisions per run (ANCR). Instead of simply stopping a simulation when the robot collides with a dynamic obstacle, we allow the simulation to continue until the robot reaches the goal and keep track of whether the robot collides again with another obstacle or not. This performance measure allows us to make the distinction between two different algorithms with similar PCFR.

The third performance measure is the average minimum distance (AMD) between the robot and the dynamic obstacles. For each simulation, we measure the distance between the robot and any dynamic obstacle along the robot's trajectory and record the minimum distance. If the robot collides with an obstacle, the minimum distance is simply zero. We then take the average of all these minimum distances over all the simulations. This gives us the AMD measure. The idea is that it is safer for the robot to stay farther from the dynamic obstacles.

D. Results

We have tested the three different algorithms (PF-ET, F-ET, and SH) and the six variations of SH algorithm in five different

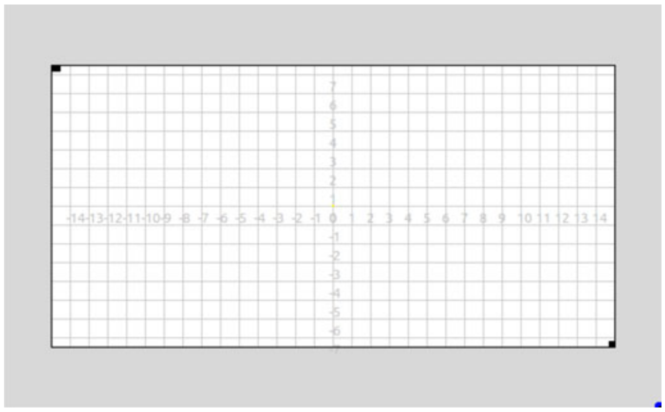


Fig. 7. Map 1: Empty space with no obstacles.

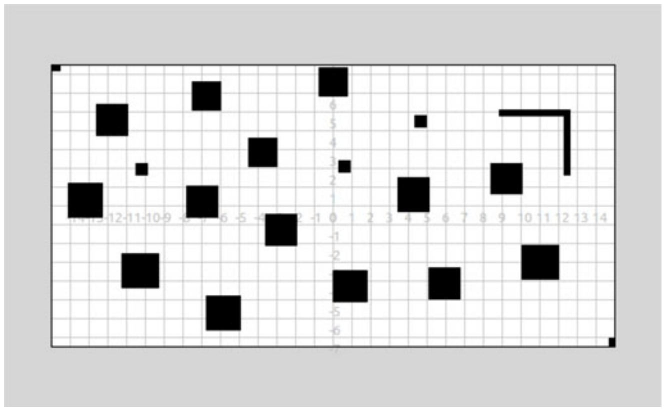


Fig. 8. Map 2: Cluttered with obstacles.

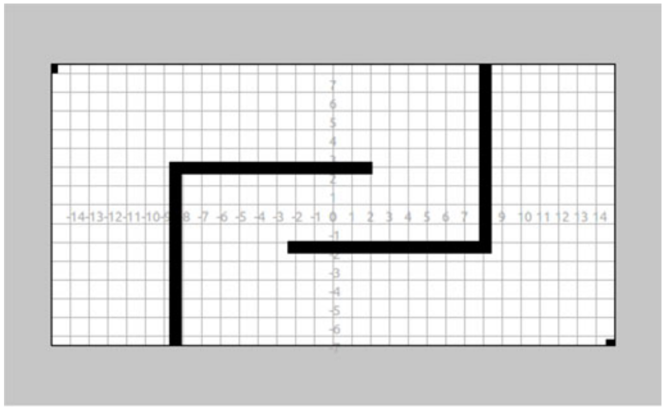


Fig. 9. Map 3: Contains u-shaped obstacles.

maps (see Figs. 7–11) and across three different sets of simulations (varying parameters for the robot and the motion models of the dynamic obstacles). For each set, each algorithm is run six times per map. Each run corresponds to a new initial and goal configuration pair for the robot and a new set of trajectories for the dynamic obstacles. For example, for the first set of simulations, the SH algorithm has been run 30 times (six times in each map). Therefore, we have run $9 \times 30 \times 3 = 810$ simulations in

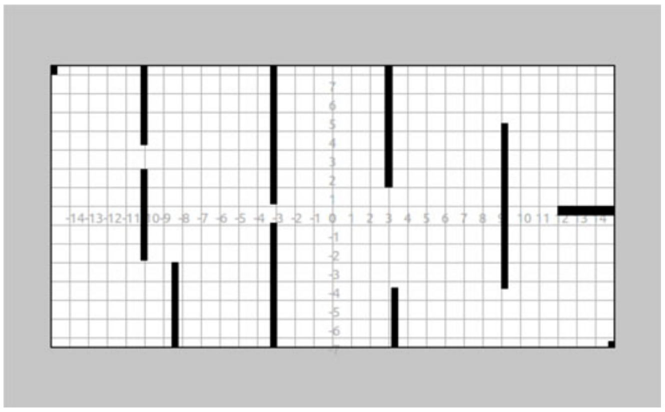


Fig. 10. Map 4: Several narrow passages.

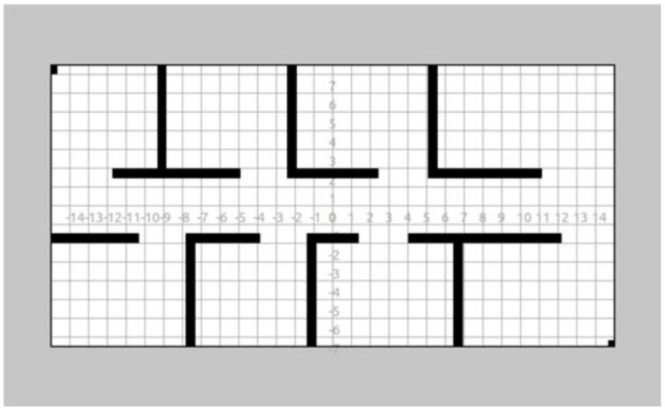


Fig. 11. Map 5: Hallway situation.

total. The most important parameters for all these simulations are given in Table I.

For each set of simulations and each algorithm, we report the three different performance measures in Tables II–IV. Below each performance measure, we report the *p*-value of the hypothesis test indicating whether the performance of the SH is statistically significantly better than the result obtained for the particular algorithm. A small *p*-value signifies that there is enough evidence to support the claim that the performance of the SH is better than the performance of the particular algorithm. We have also included the average time to reach the goal in the last row of each table.

As expected, the PF-ET algorithm performs better than any other algorithm across most performance measures (except the AMD). This is normal since the PF-ET algorithm is the only algorithm with the perfect knowledge about the trajectories of the dynamic obstacles. For the AMD measure, the PF-ET algorithm performs poorly because it frequently grazes the obstacles. It is also interesting to note that even with the perfect knowledge, there still were simulations where the PF-ET algorithm was not able to reach the goal without colliding with a dynamic obstacle. This means that for these particular simulations, $\Delta_p = 3$ s is too short; the robot puts itself in a ICS where the collision with an obstacle will occur later than 3 s from now.

We analyze in more detail the results for the three different sets of simulations.

1) First Set of Simulations: For the first set of simulations, all the dynamic obstacles in the environments are governed by the highly erratic model. The maximum velocity of the robot is 1 m/s and maximum turning rate is 0.8 rad/s. The complete results for the first set of simulations are presented in Table II.

The key result is that the SH performs significantly better than the F-ET algorithm for all three performance measures. This is even if the F-ET algorithm uses $\Delta_e = 0.2$ s which is four times faster than the $\Delta_e = 0.8$ s used by the full hierarchy. We show a typical run with a collision of the F-ET algorithm in VIDEO 3. We show the same run with the full hierarchy algorithm in VIDEO 4. With respect to the different safety variations, the SH performs significantly better than the six variations on at least one performance measure. Finally, note that it takes, on average, about twice as long for the SH algorithm to reach the goal than for the F-ET algorithm.

2) Second Set of Simulations: For the second set of simulations, most dynamic obstacles (21 of them) are using the less erratic model and a few (four of them) are using the periodic model. The robot parameters are the same as in the first set of simulations. The complete results for the second set of simulations are presented in Table III.

The key result is, once again, that the SH performs significantly better than the F-ET algorithm for all three performance measures. With respect to the different safety variations, the SH performs significantly better than the six safety variations on at least one performance measure. Note that the values of the three performance measures are better for the SH than the variation ET+SW but this difference is less statistically significant since the lowest *p*-value is 0.1652. Finally, note that it takes, on average, about twice as long for the SH algorithm to reach the goal than for the F-ET algorithm.

3) Third Set of Simulations: For the third set of simulations, all the obstacles in the environments are using the highly erratic model. The robot is also slower; the maximum velocity of the robot is 0.5 m/s and maximum turning rate is 0.4 rad/s. Under this new set of parameters for the robot, the environment becomes extremely challenging since the robot is slower and less maneuverable; in fact, the maximum velocity of the robot is lower than that of the obstacles. The complete results for the third set of simulations are presented in Table IV.

The key result is that the SH performs better than the F-ET algorithm for all three performance measures. However, the difference is statistically significant only for the AMD measure. We believe this can be explained by the combined facts that the obstacles move faster than the robot and that on average it takes about twice as long for the SH to reach the goal than for the F-ET algorithm. This may lead to more situations where the robot cannot avoid a collision with an obstacle simply because they are faster than the robot. We show one of the simulations with such a collision of the SH algorithm in VIDEO 5. With respect to the different safety variations, the SH performs significantly better than the six variations on at least one performance measure.

TABLE I
VALUE OF THE IMPORTANT PARAMETERS

Component	Parameters	Description	Value
Planner	Δ_o^c	observer computation time	$\approx$ 0.10 s
	$\Delta_{r,p}^c$	planner computation time	$\approx$ 0.60 s
	Δ_o	duration of obstacles trajectories	6 s
	Δ_p	duration of planned trajectory	3 s
	m	number of time steps	30
	δ_t	duration of time step	0.1 s
	n	number of possible robot trajectories in each planning cycle	2025
Robot	$v_r^{\max}$	maximum velocity of the robot	1 m/s or 0.5 m/s
	$\omega_r^{\max}$	maximum turning rate of the robot	0.8 rad/s or 0.4 rad/s
	s_r	maximum sensor range	6 m
Obstacles	$v_o^{\max}$	maximum velocity of dynamic obstacles	0.75 m/s
	α	angle of direction change	$[-120°, 120°]$ or $[-30°, 30°]$

TABLE II
RESULTS: FIRST SET OF SIMULATIONS

Algorithms	PF-ET	F-ET	**SH**	O-ET	O-SW	O-BW	ET + SW	ET + BW	SW + BW
PCFR	93.3 %	33.3 %	63.3 %	23.3 %	50.0 %	26.7%	50.0 %	46.7 %	50%
p-value	0.9988	0.0074	N/A	0.0003	0.1465	0.0011	0.1465	0.0941	0.1465
ANCR	0.07	1.10	0.47	1.59	2.48	4.14	1.29	1.62	2.45
p-value	0.9962	0.0090	N/A	0.0001	0.0237	0.0000	0.0150	0.0088	0.0348
AMD (m)	0.09	0.04	0.15	0.05	0.11	0.09	0.11	0.17	0.14
p-value	0.0707	0.0025	N/A	0.0070	0.2454	0.1400	0.2458	0.3915	0.4095
Time (s)	44	55	103	78	119	120	88	94	121

TABLE III
RESULTS: SECOND SET OF SIMULATIONS

Algorithms	PF-ET	F-ET	**SH**	O-ET	O-SW	O-BW	ET + SW	ET + BW	SW + BW
PCFR	96.7 %	40.0 %	66.7 %	33.3 %	50.0 %	20.0 %	60.0 %	43.3 %	40.0 %
p-value	0.9994	0.0158	N/A	0.0031	0.0920	0.0000	0.2956	0.0308	0.0158
ANCR	0.03	1.07	0.37	1.38	1.12	2.5	0.6	1.14	0.71
p-value	0.9903	0.0055	N/A	0.0023	0.0177	0.0076	0.2106	0.0097	0.1540
AMD (m)	0.08	0.04	0.14	0.04	0.09	0.03	0.11	0.06	0.11
p-value	0.0186	0.0000	N/A	0.0001	0.0712	0.0002	0.1652	0.0354	0.1978
Time (s)	44	45	104	63	120	152	130	141	137

TABLE IV
RESULTS: THIRD SET OF SIMULATIONS

Algorithms	PF-ET	F-ET	**SH**	O-ET	O-SW	O-BW	ET + SW	ET + BW	SW + BW
PCFR	93.3 %	30.0 %	33.3 %	10.0 %	20.0 %	10 %	23.3 %	20.0 %	20.0 %
p-value	0.9999	0.3906	N/A	0.0111	0.1188	0.0111	0.1936	0.1188	0.1188
ANCR	0.07	1.63	1.43	2.70	4.47	7.80	2.86	2.67	3.83
p-value	0.9999	0.3161	N/A	0.0152	0.0049	0.0000	0.0167	0.0101	0.0139
AMD (m)	0.06	0.02	0.05	0.05	0.06	0.02	0.07	0.06	0.06
p-value	0.2856	0.0613	N/A	0.1256	0.3241	0.1736	0.2535	0.3354	0.3701
Time (s)	94	79	182	104	223	242	193	158	226

4) Results Summary Conclusions: The key result is that the SH performs better than the F-ET algorithm across all performance measures for most of the simulations. Based on our simulations, this safety advantage is less pronounced when the robot is less maneuverable. On the whole, this shows that it is better to use the SH even if it takes more time to compute than using the simpler and faster F-ET algorithm.

With respect to the different safety variations, the SH performs significantly better than most of the six safety variations across at least one performance measure for most of the simulations. This shows that, in most cases, it is better to use all three models of the future than using only a combination of either only one or two of them. We note that there are some cases where the ET+SW variation might be sufficient and performs similar to the full SH. These two points show the relevance of the SH algorithm.

VII. Convergence Issues

In unknown dynamic environments, one cannot normally provide guarantees of absolute safety and convergence. Nevertheless, in order to properly place this work in the appropriate context, we discuss these issues vis a vis our algorithm. It is possible that the robot may get stuck in a deadlock simply because the SH implies that if there are two paths: one which is slightly risky and makes forward progress or one which is not risky but actually degrades progress (say, moves backwards); then, the path that moves backward will be chosen. This can lead to deadlocks where no progress toward the goal is made and the robot is stuck. These deadlocks can occur purely due to static obstacles (we call them static deadlocks) or due to dynamic obstacles (we call them dynamic deadlocks). We discuss examples of each case below, as well as provide some solutions.

A. Static Deadlocks

Recall that in the SH algorithm the robot attempts to reach the goal only when it can find a trajectory that does not collide with the BW model of the future. This model treats all points that are reachable (with velocity $v_o^{\max}$) from current boundary of the FOV in time Δ_p as obstacles. If there are points on the boundary of the FOV which are relatively close to the current robot position, the robot might not make any progress towards the goal since the entire workspace in vicinity of the robot is swamped with "expanded obstacles." A simple example of such a scenario is when the robot tries to pass through a "narrow" passage (precise width will depend on $v_o^{\max}$ and Δ_p). In this case, it is possible that there is never any trajectory that exits the passage and does not collide with the expanded boundary of the FOV of the BW model of the future. If this happens, the robot either stays stationary or moves back and forth in the passage without ever exiting it unless it is forced to exit by some dynamic obstacles behind it (see Fig. 12). We suggest the following way to deal with static deadlocks.

A somewhat ad hoc solution is quite simple and has been implemented in the current version of the algorithm. One can easily detect if the robot stays either at rest or moves within a "small region" for a certain number of planning cycles (a circle

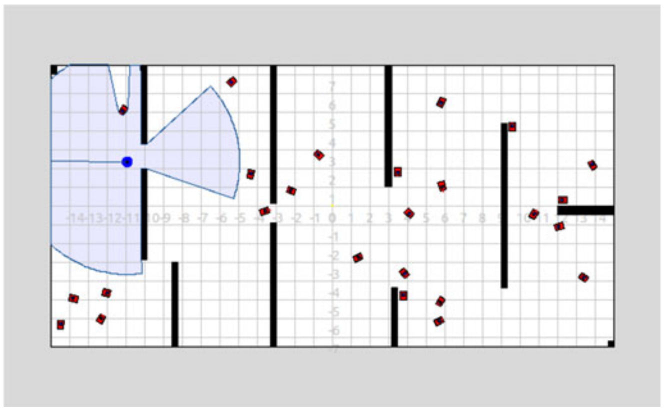


Fig. 12. Passage is too narrow for the robot to pass through it because of the BW model. The robot gets stuck in a static deadlock.

of radius 1 m for ten iterations in our implementation) of the static map, and if it happens, the robot is deemed to be in a deadlock situation. The planner then drops the safety level BW by temporarily setting $c_{\text{bw}} = 0$ for a certain number of planning cycles (five in our implementation). Once this number of cycles is attained, the planner returns to the full SH by restoring the original c_{bw} value. Unfortunately, there will be cases where this solution might set $c_{\text{bw}} = 0$, and this was not necessary. The complete simulation of the situation shown in Fig. 12 is shown in VIDEO 6 attached to this paper. Note that although this is an ad hoc solution, it does actually work. Among the 90 runs of the full hierarchy algorithm, the robot got stuck 12 times, and the ad hoc fix always managed to resolve the deadlock; in every case, the robot eventually reached the goal.

B. Dynamic Deadlocks

Dynamic deadlocks can occur in a varied number of different situations. For example, if a dynamic obstacle is temporarily stationary or moving very slowly just next to the goal, the robot will not reach the goal because of the BW and the SW model of the future.

Some of the situations are more difficult to characterize. For example, if at any time, the robot has to move from a region where the density of dynamic obstacles is sparse to a region where it is high, to reach the goal, a dynamic deadlock can occur. A concrete example is when a robot needs to go across a room full of people to reach its goal (see Fig. 13 and VIDEO 7). The SH deems it safer (and it is safer) to stay outside of the room than to enter it; hence, the robot will not enter the room and will not be able to reach the goal. Note that this is simply because it is not safe to enter the room and our algorithm is not willing to compromise the safety of the robot and obstacles to reach the goal at all cost. To show this, we have repeated the exact same simulation with the F-ET algorithm (see VIDEO 8). As can be seen in the video, the robot takes a chance by entering the room and ends up colliding with two dynamic obstacles before it reaches the goal.

One solution for the dynamic deadlocks, similar to the first fix for the static deadlocks, is to add a condition that if the robot is not making enough progress toward the goal, we can shut down

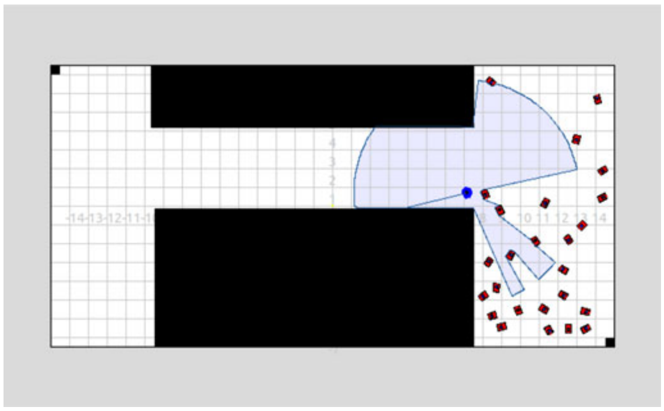


Fig. 13. Dynamic obstacles are artificially forced to stay in the room on the right for the entire simulation. There are 25 dynamic obstacles in the room. The robot's goal is in the bottom right corner, and it is not possible for the robot to reach the goal safely.

the BW and the SW model of the future by setting $c_{\rm bw} = 0$ and $c_{\rm sw} = 0$. However, this compromises the safety of the robot and, as expected, leads to more frequent collisions with dynamic obstacles (as verified by our simulations). One problem with the dynamic deadlocks is that they are not as easy to identify as the static ones.

VIII. Conclusion

We have presented a planning algorithm for a mobile robot in unknown indoor dynamic environments. The algorithm takes into account the robot's own dynamics and the dynamic obstacles' future behavior in a SH. It also uses an appropriate time horizon for planning and respects the timing constraints on various modules of the planner that arise due to the interleaving of planning and execution by using a planner for which an upper bound exists for the computation time. We provide a formulation of the SH in a single step via a composite costmap. We believe that our SH provides a useful compromise between the safety of the robot and its ability to reach the goal. The planner has been implemented in Player and Stage, and we have illustrated its performance on several simulations.

IX. Future Work

First and foremost, we would like to implement our SH approach on the Powerbot mobile robot in our lab. Next, there are four aspects that we would like to address in our future work. The first is a more systematic way to deal with deadlocks. For static deadlocks, we believe a better approach is to identify the regions in the static map that create these deadlocks. Note that such regions will depend on 1) direction of robot travel, 2) $v_o^{\max}$, and 3) Δ_p. Once these regions have been identified, we can temporarily set $c_{\rm bw} = 0$ when the robot enters these regions. As opposed to the approach mentioned earlier in this paper, here, the planner will only set $c_{\rm bw} = 0$ when such a region has to be crossed to reach the goal. This idea could be incorporated in our system to avoid static deadlocks.

The second aspect is to adapt our algorithm to deal with the cases where the robot has an arbitrary shape. Here is a brief explanation of how this could be done. The costmap is constructed exactly the same way as for a point robot (as described in Section III-C). The change occurs when we evaluate the different robot's trajectories in the costmap. A robot's trajectory is now represented by the quadruple (x, y, θ, t). For a given time t_0, using the geometry of the robot, we determine the costmap cells occupied by the robot. The cost for this particular point of the trajectory (x, y, θ, t_0) is the maximum of the costmap cells values which are occupied by the robot. Taking the maximum maintains the SH. For example, if the robot is colliding with both the SW and the ET model of the futures, the costmap cell related to the ET model has a higher cost, which would be the cost of that piece of the trajectory.

The third aspect is that we would like to change the ET model of the future with a probabilistic model of the future in the SH. We believe that this can be done in a straightforward way by modifying $c_{\rm et}$ and incorporating the probabilities in the costmap.

Finally, we assumed that Δ_e and Δ_p are fixed, i.e., the duration of each trajectory to be executed is always the same. We will relax this restriction in a subsequent version of this work by allowing Δ_e and Δ_p to be adaptive parameters. This could be accomplished, for example, by combining our approach with the $\tau-$safety concept presented in [21]. We anticipate that these changes could eliminate the static deadlock problems.

References

[1] K. Kant and S. W. Zucker, "Toward efficient trajectory planning: The path-velocity decomposition," *Int. J. Robot. Res.*, vol. 5, no. 3, pp. 72–89, Sep. 1986.

[2] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *Int. J. Robot. Res.*, vol. 17, no. 7, pp. 760–772, Jul. 1998.

[3] F. Large, S. Sckhavat, Z. Shiller, and C. Laugier, "Using non-linear velocity obstacles to plan motions in a dynamic environment," in *Proc. Int. Conf. Control, Autom., Robot. Vis.*, Singapore, Mar. 2002, pp. 734–739.

[4] D. Hsu, R. Kindel, J. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *Int. J. Robot. Res.*, vol. 21, no. 3, pp. 233–256, Mar. 2002.

[5] T. Fraichard and H. Asama, "Inevitable collision states—A step towards safer robots?" *Adv. Robot.*, vol. 18, no. 10, pp. 1001–1024, Jul. 2004.

[6] M. Seder and I. Petrovic, "Dynamic window based approach to mobile robot motion control in the presence of moving obstacles," in *Proc. Int. Conf. Robot. Autom.*, Rome, Italy, Jun. 2007, pp. 1986–1991.

[7] T. Fraichard, "A short paper about motion safety," in *Proc. IEEE Int. Conf. Robot. Autom.*, Rome, Italy, Apr. 2007, pp. 1140–1145.

[8] T. Fraichard and T. Howard, " Iterative motion planning and safety issue," in *Handbook of Intelligent Vehicles*, A. Eskandarian, Ed. London, U.K.: Springer, 2012, pp. 1433–1458.

[9] J. Minguez and L. Montano, "Nearness diagram (ND) navigation: Collision avoidance in troublesome scenarios," *IEEE Trans. Robot.*, vol. 20, no. 1, pp. 45–59, Mar. 2004.

[10] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robot. Autom. Mag.*, vol. 4, no. 1, pp. 23–33, Mar. 1997.

[11] S. Bouraine, T. Fraichard, and H. Salhi, "Provably safe navigation for mobile robots with limited field-of-views in dynamic environments," *Auton. Robots*, vol. 32, no. 3, pp. 267–283, 2012.

[12] Z. Shiller, O. Gal, and T. Fraichard, "The nonlinear velocity obstacle revisited: The optimal time horizon," presented at the Guaranteeing Safe Navig. Dyn. Environ. Workshop, Anchorage, AK, USA, May 2010.

[13] K. Macek, D. Vasquez, T. Fraichard, and R. Siegwart. Towards safe vehicle navigation in dynamic urban scenarios. *Automatika*, vol. 50, no. 3–4, pp. 184–194, Nov. 2009.

[14] B. Kluge, C. Kohler, and E. Prassler, "Fast and robust tracking of multiple moving objects with a laser range finder." in *Proc. IEEE Int. Conf. Robot. Autom.*, 2001, pp. 1683–1688.

[15] A. Ess, K. Schindler, B. Leibe, and L. Van Gool, "Object detection and tracking for autonomous navigation in dynamic environments," *Int. J. Robot. Res.*, vol. 29, pp. 1707–1725, Dec. 2010.

[16] C. Fulgenzi, A. Spalanzani, and C. Laugier, "Dynamic obstacle avoidance in uncertain environment combining PVOs and occupancy grid," in *Proc. IEEE Int. Conf. Robot. Autom.*, Rome, France, Apr. 2007, pp. 1610–1616.

[17] A. Bautin, L. Martinez-Gomez, and T. Fraichard, "Inevitable Collision States: A Probabilistic Perspective," in *Proc. IEEE Int. Conf. Robot. Autom.*, Anchorage, AK, USA, May 2010, pp. 4022–4027.

[18] D. Althoff, M. Althoff, D. Wollherr, and M. Buss, "Probabilistic collision state checker for crowded environments," in *Proc. IEEE Int. Conf. Robot. Autom.*, Anchorage, AK, USA, May 2010, pp. 1492–1498.

[19] J. Van Den Berg, D. Ferguson, and J. Kuffner, "Anytime path planning and replanning in dynamic environments," in *Proc. IEEE Int. Conf. Robot. Autom.*, Orlando, FL, USA, May 2006, pp. 2366–2371.

[20] S. Petti and T. Fraichard, "Partial motion planning framework for reactive planning within dynamic environments," presented at the IFAC/AAAI Int. Conf. Informat. Control, Autom. Robot., Barcelona, Spain, Sep. 2005.

[21] E. Frazzoli, M. A. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *AIAA J. Guid., Control Dyn.*, vol. 25, no. 1, pp. 116–129, 2002.

[22] R. Vatcha and J. Xiao, "Perceived CT-space for motion planning in unknown and unpredictable environments," in *Proc. Workshop Algorithmic Found. Robot.*, Guanajuato, Mexico, Dec. 2008, pp. 183–198.

[23] J.-C. Latombe, *Robot Motion Planning.* Norwell, MA, USA: Kluwer, 1991, ch. 7.

[24] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach." in *Proc. Int. Conf. Robot. Autom.*, 1999, pp. 341–346.

[25] K. Arras, S. Grzonka, M. Luber, and W. Burgard, "Efficient people tracking in laser range data using a multi-hypothesis leg-tracker with adaptive occlusion probabilities," in *Proc. IEEE Int. Conf. Robot. Autom.*, Pasadena, CA, USA, 2008, pp. 1710–1715.

[26] K. O. Arras, O. M. Mozos, and W. Burgard, "Using boosted features for the detection of people in 2d range data," presented at the IEEE Int. Conf. Robot. Autom., Rome, Italy, 2007.

[27] R. T. Vaughan, "Massively multi-robot simulations in stage," *Swarm Intell.*, vol. 2, nos. 2–4, pp. 189–208, 2008.

Bruno L'Espérance received the B.Sc. degree in mathematics and economics from the University of Montreal, Montreal, QC, Canada, and the M.S. degree in applied mathematics from the University of British Columbia, Vancouver, BC, Canada, in 2005 and 2007, respectively. He is currently working toward the Ph.D. degree with the School of Engineering Science, Simon Fraser University, Burnaby, BC, since September 2009.

His research interests include motion planning algorithms to guarantee safety of mobile robotic systems in dynamic environments.

Kamal Gupta received the Ph.D. degree in electrical engineering from McGill University, Montreal, QC, Canada, in 1987.

He is a Professor with the School of Engineering Science, Simon Fraser University, Burnaby, BC, Canada. His research interests include motion planning and sensor-based planning and safety.