Generalized Velocity Obstacles

David Wilkie Jur van den Berg Dinesh Manocha

***Abstract*— We address the problem of real-time navigation in dynamic environments for car-like robots. We present an approach to identify controls that will lead to a collision with a moving obstacle at some point in the future. Our approach generalizes the concept of *velocity obstacles*, which have been used for navigation among dynamic obstacles, and takes into account the constraints of a car-like robot. We use this formulation to find controls that will allow collision free navigation in dynamic environments. Finally, we demonstrate the performance of our algorithm on a simulated car-like robot among moving obstacles.**

I. Introduction

The problem of computing a collision-free path for a robot moving among dynamic obstacles is important in many robotics applications, including automated transportation systems, automated factories, and applications involving robot-human interactions, such as robotic wheelchairs [18]. In many of these applications, the robot of interest is a car-like robot and subject to non-holonomic kinematic constraints.

In this paper, we extend the velocity obstacle concept to handle such robots. This concept was introduced by Fiorini and Shiller [2] for navigating robots among arbitrarily moving obstacles and has been extended to multi-agent navigation and real-world applications. The velocity obstacle formulation works well for robots that can move in any specified direction, but it may not capture the movement of car-like robots well, which can only move, at any instant, with a velocity parallel to the rear wheels.

In this paper, we present an extension of the velocity obstacle concept to handle robots with kinematic constraints. We present a new algebraic formulation of the velocity obstacle in terms of the set of controls that can, at some point in the future, result in a collision. This generalized velocity obstacle approach is used to safely navigate a car-like robot among dynamic obstacles.

Our approach focuses on iteratively sensing and avoiding obstacles. The intended use is within *sense-plan-act* iterations that could be incorporated in a global planner or a *partial motion planner*, a concept introduced in [16] to compute global collision-free paths incrementally.

The rest of this paper is organized as follows. Section II discusses the related work and focuses on techniques for navigating car-like robots. Section III provides background information on velocity obstacles and the issues in using them for car-like robots. Section IV introduces the idea of generalized velocity obstacles. In Section V, we apply generalized velocity obstacles to a car-like robot moving among dynamic obstacles, and highlight the algorithm's performance. Section VI presents concluding remarks on future work and some deficiencies of the approach.

Department of Computer Science, College of Arts and Sciences, University of North Carolina, 201 South Columbia Street, Chapel Hill, NC 27599, USA. E-mail: {wilkie, berg, dm}@cs.unc.edu

For further project details, code, and videos, refer to http://gamma.cs.unc.edu/NHRVO.

This work was supported in part by ARO contracts DAAD19-02-1-0390 and W911NF-04-1-0088, NSF awards 0400134, 0429583, and 0404088, DARPA/RDECOM contract N61339-04-C-0043, Intel, and the UNC Merit Assistantship.

II. Prior Work

In this section, we give a brief overview of related work on navigating agents and robots in dynamic environments.

Numerous motion planning algorithms have been developed for car-like robots in static environments. In [11], the notion of a *car-like robot* was formalized, and the fact that a path for a holonomic robot lying fully in open regions of the configuration space can always be transformed into a feasible path for a nonholonomic robot was proven. Laumond *et al.* [11] also provided an algorithm to generate a feasible path for a nonholonomic robot from a path found for a holonomic robot. Smooth planning for car-like robots among obstacles was first proposed by Scheuer *et al.* [20], and this idea was extended by Lamiraux and Laumond [9], which uses a steering function rather than computing clothoid curves.

Approaches applicable to car-like robots have been developed for complete trajectory planning among moving obstacles, such as [4], [6], [26]. Some of these approaches decompose the problem of planning in a dynamic environment into generating a feasible path and planning a velocity profile to safely traverse the path, [7], [15], [25].

An alternative to complete planning is to plan for the robot as it acts, taking new sensor inputs as they arrive and planning locally. Some of the prominent work in this area is based on *velocity obstacles* (VO) [1], [2]. The idea is to define a set of velocities that would, if used as a control for the agent, lead to a collision with an obstacle at some time in the future. In its original formulation, VO was applied to agents moving along piecewise linear velocities and assumed the obstacles would be moving at constant velocities over a time interval. The idea was extended in [21], which used the idea of a *non-linear velocity obstacle* to define a VO for an obstacle moving along an arbitrary trajectory, which needs to be known in advance. Large *et al.* [10] addressed the problem of predicting the motion of obstacles and applied the result to a car-like robot, but they still model the agent velocity as a piecewise linear function. An extension of VO by van den Berg *et al.*, the *Reciprocal Velocity Obstacle*, appeared in [24], which addressed the oscillation issue that occurs in the VO formulation. This work was expanded to address the

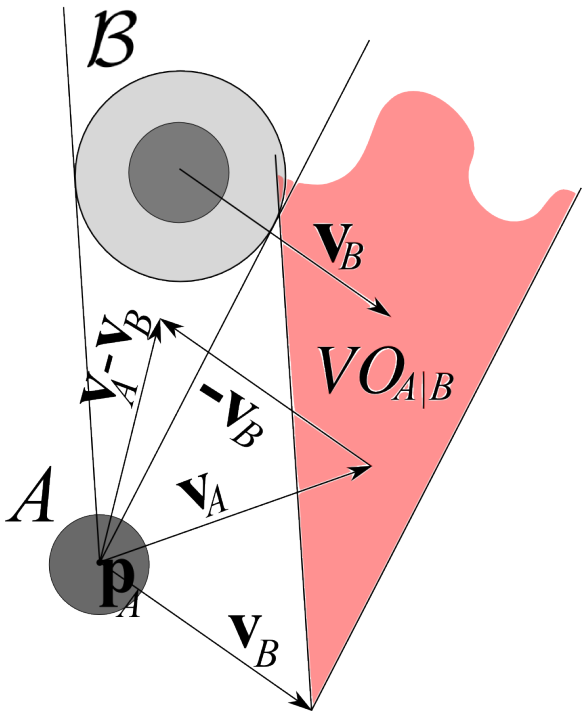

Fig. 1. The velocity obstacle $VO_{A|B}$ for robot A relative to dynamic obstacle B. The cone of velocities that would lead to a collision in the static case is translated by $\mathbf{v_B}$. Note that as the velocity $\mathbf{v_A}$ is inside the velocity obstacle, the relative velocity, $\mathbf{v_A} - \mathbf{v_B}$, leads to a collision.

n-body collision avoidance problem [23]. The VO concept has been applied to control real robots, such as in [17], [18].

An analytical approach to compute a trajectory for a non-holonomic robot moving among dynamic obstacles is given in [19], and this approach was combined with rapidly-exploring random trees (RRTs) in [13]. Another approach to navigating non-holonomic agents in dynamic environments is presented in [14].

Some iterative approaches to planning in dynamic environments exist, such as [3], [6], [10], that seek to balance this local planning strategy with reaching a goal.

Finally, another approach to navigation among dynamic obstacles is to create potential fields for obstacles [8], [22].

III. Velocity Obstacles

In this section, we review the concept of velocity obstacles and discuss its application to car-like robots.

A. Definition

For a disc-shaped agent A and a disc-shaped moving obstacle B with radii r_A and r_B, respectively, the velocity obstacle for A induced by B, denoted $VO_{A|B}$, is the set of velocities for A that would, at some point in the future, result in a collision with B. This set is defined geometrically. First, let $\mathbf{p}_A$ and $\mathbf{p}_B$ be the center points of A and B, respectively, and let $\mathcal{B}$ be a disc centered at $\mathbf{p}_B$ with a radius equal to the sum of A's and B's. This is generalized as the Minkowski sum. If B is static (i.e. not moving), we could define a cone, $\mathcal{C}$, of velocities for A that would lead to a collision with B as the set of rays shot from $\mathbf{p}_A$ that intersect the boundary of $\mathcal{B}$. To derive a velocity obstacle from this, we simply translate the cone $\mathcal{C}$ by the velocity $\mathbf{v}_B$ of B, as shown in Figure 1. More formally:

$$VO_{A|B} = \{\mathbf{v} \,|\, \exists t > 0 :: \mathbf{p}_A + t(\mathbf{v} - \mathbf{v}_B) \in \mathcal{B}\}. \quad (1)$$

B. Robots with Kinematic Constraints

For many kinematically constrained robots, such as car-like robots, the set of feasible velocities at any instant is a single velocity – the specific velocity in the direction of the rear wheels. One way to workaround this constraint and use velocity obstacles is to use the set of velocities that can be achieved over some time interval τ [2]. However, this approach does not guarantee collision-free navigation: consider the set of velocities, V, that a car-like robot can reach after τ seconds. VOs can be constructed for the robot, and a velocity from V outside the VOs can be selected to navigate the robot. In this case, a car-like robot will actually need to follow an arc to achieve the selected velocity, and the VOs provide no guarantee that this arc will be collision-free. Additionally, the robot will be at a different position when it achieves the selected velocity, but the VOs that were computed using the robot's initial position, and thus the velocity is no longer guaranteed to be collision-free. To alleviate these issues, we can select a small value for τ, but this has some implications for navigation: as τ decreases, the set of velocities that are being considered by the robot becomes smaller, and the robot can miss feasible controls.

IV. Generalized Velocity Obstacles

In this section, we define a new concept called the *Generalized Velocity Obstacle*. This is a generalization of the velocity obstacle concept and seeks to address the difficulty of using velocity obstacles with kinematically constrained agents (*e.g.* car-like robots).

A. Definition

The generalized obstacle can be defined as follows. Given an obstacle B, let us denote its position at time t by $B(t)$. Similarly, given the position of agent A at time $t = 0$ and a control u, let us denote the position agent A will have after undertaking u for time interval t by $A(t, u)$. Here, a control u is a set of inputs to the kinodynamic model that results in a change in the robot's configuration. If we continue to restrict our attention to agents and obstacles that are circularly shaped, we can define the obstacle in the control space as

$$\{u \,|\, \exists t > 0 :: \|A(t, u) - B(t)\| < r_A + r_B\}. \quad (2)$$

Given a specific set of kinematic constraints for some system, we can apply the $VO_{A|B}$ formulation as follows. Let $t_{\min}(u)$ be the time at which the distance between the centers of A and B is minimal for a given control u for A:

$$t_{\min}(u) = \arg\min_{t>0} \|A(t, u) - B(t)\|. \quad (3)$$

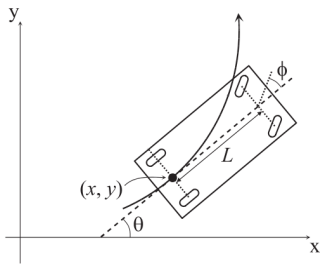

Fig. 2. The kinematic model of a simple car.

If a closed expression for $t_{\min}(u)$ is obtained, for example by solving $\frac{\partial \|A(t,u)-B(t)\|}{\partial t} = 0$ for t, then it may also be possible that an explicit equation for the velocity obstacle can be found by solving $\|A(t_{\min}(u), u) - B(t_{\min}(u))\| < r_A + r_B$ for u.

For the cases when a closed form solution is not obtainable, this approach can be used in a sampling scheme. In this case, controls in U are sampled. For each control u, the minimum distance that agent A and obstacle B would achieve were A to use control u is numerically calculated. This distance determines whether the control would be collision free.

B. Example: an Unconstrained Robot

To illustrate this approach, let us take as an example an agent without kinematic constraints – the type of agent for which the original velocity obstacle formulation was defined by Fiorini and Shiller [1]. In this case, a control u for A directly corresponds to a velocity $\mathbf{v}$ for A. Let us assume, without loss of generality, that the initial position of A is at the origin. Let an obstacle B be at the initial position $\mathbf{p}_B$ and be moving with the velocity $\mathbf{v}_B$. Then we have

$$A(t, \mathbf{v}) = t\mathbf{v}, \quad (4)$$

$$B(t) = \mathbf{p}_B + t\mathbf{v}_B. \quad (5)$$

Solving for t the equation $\frac{\partial \|A(t,u)-B(t)\|}{\partial t} = 0$ gives the equation for $t_{\min}(\mathbf{v})$:

$$t_{\min}(\mathbf{v}) = \frac{\mathbf{p}_B \cdot (\mathbf{v} - \mathbf{v}_B)}{\|\mathbf{v} - \mathbf{v}_B\|^2}. \quad (6)$$

Reducing $\|A(t_{\min}(\mathbf{v}), \mathbf{v}) - B(t_{\min}(\mathbf{v}))\| < r_A + r_B$ yields an expression that is true for all velocities in the velocity obstacle.

This velocity obstacle is identical to those derived in [2] and can be used for navigation in a fast sampling algorithm. A preferred control u^* can be computed based on a function of the current configuration and the goal configuration, and the control closest to the preferred control but outside all velocity obstacles can be used to navigate the robot. The difference between the two definitions of VOs in (1) and (2) is that we can generalize the latter to incorporate kinematic constraints, as shown in Section V.

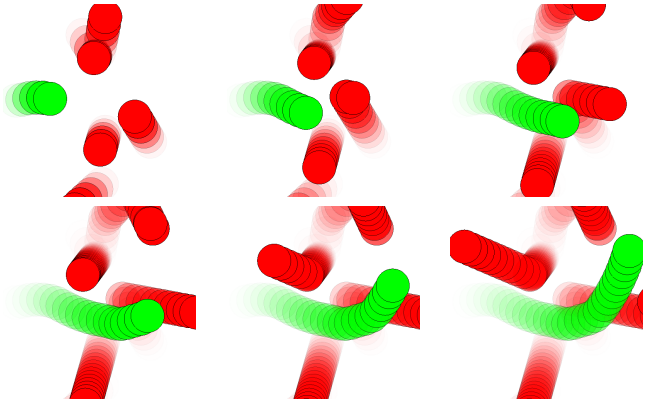

Fig. 3. The agent, green, navigates among numerous obstacles, in red, while heading toward a goal in the upper right corner.

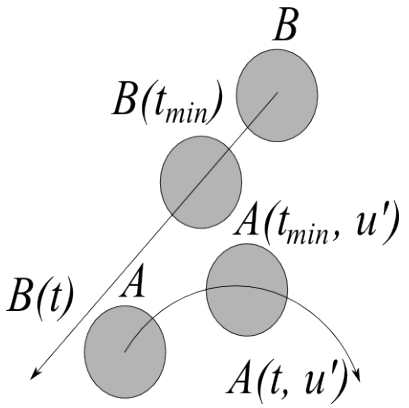

Fig. 4. Obstacle B is moving along trajectory $B(t)$, and agent A is trying to evade. A tests control u', which generates trajectory $A(t, u')$. The minimum distance between B and A occurs at t_{min}, and this distance is greater than the sum of the radii. Therefore, control u' is not in the velocity obstacle.

C. Finite Time Horizon

In practice it is useful to limit the velocity obstacle to address only those collisions that will happen before a time horizon, $t_{\lim}$. Beyond the time horizon the collisions are assumed to be too unlikely to consider. For the traditional velocity obstacle formulation, this time horizon changes the velocity obstacle from a cone to a truncated cone with a rounded end, which can be approximated as in [5]. The time horizon can easily be incorporated in the above definitions: the clause "$t > 0$" must be replaced by "$t \in [0, t_{\lim}]$" in (1), (2) and (3). The rest of the paper assumes the use of a time horizon.

V. Navigating a Simple Car

A. Velocity Obstacles for Car Kinematics

We address navigating a simple car robot following the framework laid out above. Let (x, y) be the position of the robot and θ its orientation. Following [12], its kinematic

Algorithm 1 Find best feasible control

```
for i = 0 to n do
  u ← sample controls from the set of all controls U
  t_lim ← sample time limit ∈ (0, max]
  free ← true
  min ← ∞
  for all Moving Obstacles B do
    let D(t) = the distance between A(t, u) and B(t)
    t_min ← solve min(D(t)) for t ∈ [0, t_lim].
    d ← D(t_min).
    if d < r_A + r_B then
      free ← false
    end if
  end for
  if ‖u − u*‖ < min then
    min ← ‖u − u*‖
    argmin ← u
  end if
end for
return argmin
```

constraints are given as

$$\begin{aligned} x'(t) &= u_s \cos\theta(t), \\ y'(t) &= u_s \sin\theta(t), \\ \theta'(t) &= u_s \frac{\tan u_\phi}{L}, \end{aligned} \tag{7}$$

where u_s and u_ϕ are the controls of the car, i.e. speed and steering angle, respectively, and L is the wheelbase of the car. This is shown in Figure 2.

Integrating the above equations (7) yields an expression for the position of a car at time t under the assumption that the controls remain constant:

$$A(t,u) = \begin{pmatrix} \frac{1}{\tan(u_\phi)} \sin(u_s \tan(u_\phi) t) \\ -\frac{1}{\tan(u_\phi)} \cos(u_s \tan(u_\phi) t) + \frac{1}{\tan(u_\phi)} \end{pmatrix}. \tag{8}$$

This derivation assumes the car has a wheelbase $L = 1$. For the full derivation of these equations, please see the appendix. These positions are derived relative to the robot's frame of reference, in which the robot is at the origin with its orientation along the positive x-axis. We will assume that there are an arbitrary number of obstacles B_i and that we can view them as moving linearly over a short time interval,

$$B_i(t) = \mathbf{p}_{B_i} + \mathbf{v}_{B_i} t \tag{9}$$

We proceed by finding an expression for the minimum distance between the robot A and a dynamic obstacle B given a control u for A. The derivative of the distance can be calculated, but this does not produce a simple analytical expression for $t_{\min}$. Therefore, we will solve for $t_{\min}$ numerically for specific control and check whether the control is inside the velocity obstacle. In Figure 4, we see an example of a control u' that is outside the velocity obstacle.

B. Approach

We use an optimization procedure to navigate the robot among multiple moving obstacles, as shown in Figure 3. Let u^* be the control the robot would select if no moving obstacles were around. We refer to u^* as the *preferred* control. In this paper, the preferred control is simply the control that would bring the agent closest to its goal, ignoring local minima. However, the preferred control could be the result of another algorithm. In this case, the algorithm used should have the ability to quickly replan from an unforeseen starting point. This would allow the agent to move in a way that does not exactly conform to the path, but that avoids the obstacles.

The actual control u to give the robot is given by the solution to the problem

$$u = \underset{u' \notin \bigcup_{B_i} VO_{A|B_i}}{\arg\min} \|u^* - u'\|. \tag{10}$$

That is, the problem of navigation among multiple dynamic obstacles can be formulated as the minimization of the distance between the optimal control, u^*, and a sampled control, u', where u' is subject to a velocity obstacle constraint for each moving obstacle.

An approach to solving this optimization procedure is given in Algorithm 1. At each discrete time step, t, we sample n controls, where n is a parameter. For each sampled control, we check, for each moving obstacle B_i, whether this control is inside the velocity obstacle induced by B_i. If the control is outside all of the velocity obstacles, then the control is feasible and is tested against the current best feasible control in terms of distance from the preferred control, u^*.

Collision free controls. The controls chosen are guaranteed to be collision free only so long as the obstacles continue along their paths and only up to the time horizon used in the computation, t_{lim}. Otherwise, the robot is guaranteed not to collide with the obstacles.

C. Experiments

Empirical tests of some of the algorithm's parameters are summarized in Figure 5. Each experiment takes place in an open environment in which the agent starts at (-5,0) and must move to a goal located at (10,10). The obstacles are distributed at random within a square region bounded by (-12.5, -12.5) and (12.5, 12.5). The obstacles are given random velocities, and both the obstacles and the agent are subject to the same upper limit on their speed, $\pm$ $1.5 \frac{unit}{sec}$. The agent and obstacles both have radius of size 1. For all the experiments, the probability that an obstacle will change direction within 1 second is 0.2. This could occur at any timestep, however, so the timestep of the simulation influences when an obstacle can change direction. The time horizon for each experiment is 3.5 seconds. Each experimental value presented is the mean of 10 trials. These experiments were done on a 3.2 GHz computer with 1 GB of memory.

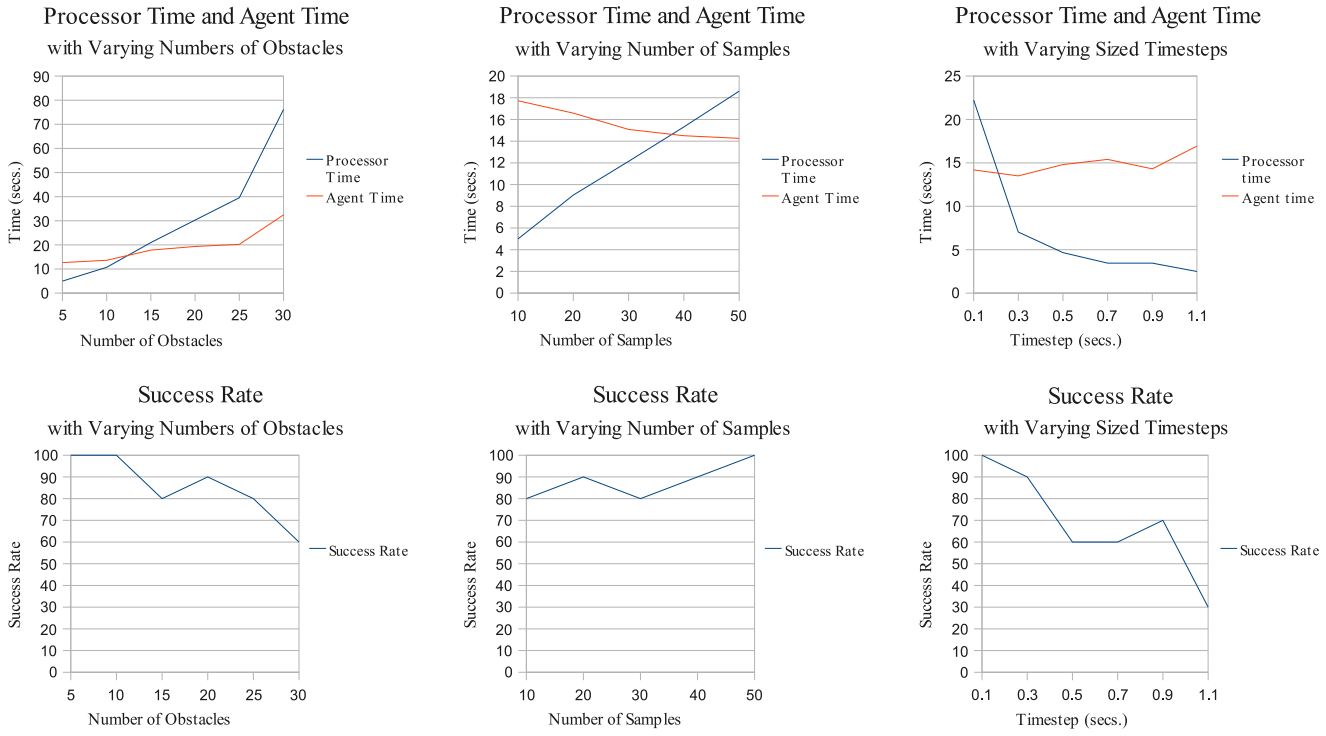

Fig. 5. These graphs present the experiments done for three parameters, the number of obstacles, the number of samples drawn, and the length of the timestep. The effect on processing time and success rate are investigated. The processor time plots are the time taken to process the algorithm, and the agent time plots are the time that elapses in the simulation, *i.e.* the time that passes from the agent's perspective. A successful run is one in which the agent reaches the goal without being hit by an obstacle, which occurs when an obstacle changes direction and traps the agent. (1) On the left, the number of obstacles present is varied. The top image shows that between ten and fifteen obstacles, the processing can no longer be done in real time. As more obstacles are added, the required route becomes more circuitous and takes longer in agent time. On the bottom, the percentage of trials that were successful drops as more obstacles are added. In both of these experiments, the timestep is set to 0.2 seconds and the number of samples is 30. (2) In the center, the number of samples is varied. As shown in the top graph, real time processing is possible until 40 samples per timestep. As the number of samples increases, the agent finds better controls and takes a more direct path to the goal, reducing the time required by the agent (red line). On the bottom, we see the success rate rise as the number of samples increases. In these scenarios, there are 10 obstacles and the timestep is set to 0.2 seconds. (3) On the right, the experiments vary the timestep length, the time between which new samples are taken and a control is selected. The top graph shows that between 0.1 and 0.3 seconds, real time processing becomes possible. However, as shown in the bottom graph, the success rate drops as the timestep increases. For these experiments, the number of obstacles was 10 and 30 samples were taken at each step.

VI. Conclusion

In this paper, we have generalized the velocity obstacle concept to formulate an approach to navigating a car-like robot among dynamic obstacles. The algorithm presented allows for fast navigation for car-like robots among dozens of arbitrarily moving obstacles and allows the robots to test their entire control space for a feasible control, rather than a constrained subset.

While effective, this approach has some limitations. First, the method uses a numerical means to calculate the minimum distance. The inaccuracy in the computation of the minimum time implies that a small safety buffer is required around each obstacle. Second, as this is a probabilistic algorithm, a feasible solution may not be found even if one exists. Third, this algorithm relies on the robot measuring the position and velocity of the obstacles: in a real world scenario, such measurements would surely be noisy. The noise in the reading of the obstacle's position can be overcome by enlarging the radius of the obstacle in the formulation, however, noisy velocity readings are harder to incorporate. And finally, as the current implementation uses a greedy preferred control, it is not suitable for complex environments with local minima. To handle scenarios such as this, the preferred control would need to come from a complete planner.

In our future work, we will attempt to incorporate additional constraints into the approach in order to handle fast moving agents and to generate smooth paths, rather than piece-wise smooth paths. We will also investigate using this approach for multi-agent navigation.

References

[1] P. Fiorini and Z. Shiller. Motion planning in dynamic environments using the relative velocity paradigm. In *1993 IEEE International Conference on Robotics and Automation, 1993. Proceedings.*, pages 560–565, 1993.

[2] P. Fiorini and Z. Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760, 1998.

[3] T. Fraichard. Trajectory planning in a dynamic workspace: a 'state-time space' approach. *Advanced Robotics*, 13(1):75–94, 1998.

[4] E. Frazzoli, M.A. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. *Journal of Guidance Control and Dynamics*, 25(1):116–129, 2002.

[5] S. J. Guy, J. Chhugani, C. Kim, N. Satish, P. Dubey, M. Lin, and D. Manocha. Highly Parallel Collision Aviodance for Multi-Agent

Simulation. Technical report, Department of Computer Science, University of North Carolina, 2009.
[6] D. Hsu, R. Kindel, J.C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, 21(3):233, 2002.
[7] K. Kant and S.W. Zucker. Toward efficient trajectory planning: The path-velocity decomposition. *The International Journal of Robotics Research*, 5(3):72, 1986.
[8] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90, 1986.
[9] F. Lamiraux and J.P. Lammond. Smooth motion planning for car-like vehicles. *IEEE Transactions on Robotics and Automation*, 17(4):498–501, 2001.
[10] F. Large, D. Vasquez, T. Fraichard, and C. Laugier. Avoiding cars and pedestrians using velocity obstacles and motion prediction. In *IEEE Intelligent Vehicle Symposium*, pages 375–379, 2004.
[11] J.P. Laumond, PE Jacobs, M. Taix, and RM Murray. A motion planner for nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation*, 10(5):577–593, 1994.
[12] S.M. LaValle. *Planning algorithms*. Cambridge University Press, 2006.
[13] K. Macek, M. Becker, and R. Siegwart. Motion planning for car-like vehicles in dynamic urban scenarios. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2006.
[14] E. Owen and L. Montano. Motion planning in dynamic environments using the velocity space. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005.(IROS 2005)*, pages 2833–2838, 2005.
[15] J. Peng and S. Akella. Coordinating multiple robots with kinodynamic constraints along specified paths. *The International Journal of Robotics Research*, 24(4):295, 2005.
[16] S. Petti and T. Fraichard. Safe motion planning in dynamic environments. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2210–2215, 2005.
[17] E. Prassler, J. Scholz, and P. Fiorini. Navigating a robotic wheelchair in a railway station during rush hour. *The international journal of robotics research*, 18(7):711, 1999.
[18] E. Prassler, J. Scholz, and P. Fiorini. A robotics wheelchair for crowded public environment. *IEEE Robotics & Automation Magazine*, 8(1):38–45, 2001.
[19] Z. Qu, J. Wang, and C.E. Plaisted. A New Analytical Solution to Mobile Robot Trajectory Generation in the Presence of Moving Obstacles. *IEEE Transactions on Robotics*, 20(6):978–993, 2004.
[20] A. Scheuer, T. Fraichard, and M. INRIA. Continuous-curvature path planning for car-like vehicles. In *Intelligent Robots and Systems, 1997. IROS'97., Proceedings of the 1997 IEEE/RSJ International Conference on*, volume 2, 1997.
[21] Z. Shiller, F. Large, and S. Sekhavat. Motion planning in dynamic environments: Obstacles moving along arbitrary trajectories. In *IEEE International Conference on Robotics and Automation, 2001. Proceedings 2001 ICRA*, volume 4, 2001.
[22] RB Tilove. Local obstacle avoidance for mobile robots based on the method of artificial potentials. In *1990 IEEE International Conference on Robotics and Automation, 1990. Proceedings.*, pages 566–571, 1990.
[23] J. van den Berg, S. J. Guy, M. Lin, and D. Manocha. Reciprocal n-body Collision Avoidance. In *International Symposium on Robotics Research (ISRR)*, 2009.
[24] J. Van den Berg, M. Lin, and D. Manocha. Reciprocal Velocity Obstacles for real-time multi-agent navigation. In *IEEE International Conference on Robotics and Automation, 2008. ICRA 2008*, pages 1928–1935, 2008.
[25] J. van den Berg and M. Overmars. Kinodynamic motion planning on roadmaps in dynamic environments. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems-IROS*, pages 4253–4258, 2007.
[26] M. Zucker, J. Kuffner, and M. Branicky. Multipartite RRTs for rapid replanning in dynamic environments. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 1603–1609, 2007.

APPENDIX

In this section, we derive equations used in our approach. The car is modeled as a rigid body moving along a plane, and thus its configuration is (x, y, θ). In the car's frame of reference, the origin is at the midpoint of its rear axle and it is pointed in the positive x direction. In all of our derivations, we assume the wheelbase of the car is 1. To restate (7), the kinematic equations in this reference frame are given as

$$\begin{aligned} \frac{dx}{dt} &= u_s \cos\theta, \\ \frac{dy}{dt} &= u_s \sin\theta, \\ \frac{d\theta}{dt} &= u_s \tan u_\phi, \end{aligned} \tag{11}$$

where u_s and u_ϕ are possible controls in the control space U. These are the speed and turning angle, respectively.

We can integrate as follows to derive our equation for $A(u,t)$. First, we integrate $\frac{d\theta}{dt}$,

$$\theta = u_s \tan(u_\phi) t. \tag{12}$$

Next, we integrate $\frac{dx}{dt}$ by substituting $u_s \tan(u_\phi)t = \theta$,

$$\frac{dx}{dt} = u_s \cos(\theta(t)), \tag{13}$$

$$= u_s \cos(u_s \tan(u_\phi) t), \tag{14}$$

$$\int dx = \int \frac{1}{\tan u_\phi} \cos(\theta) d\theta, \tag{15}$$

$$\text{thus, } x = \frac{1}{\tan u_\phi} \sin(u_s \tan(u_\phi) t). \tag{16}$$

We can make a similar argument for $\frac{dy}{dt}$, and we can determine from inspection that the integral adds the constant 1 to avoid a division by zero. This leads to the equation $y = \frac{1}{\tan u_\phi}(-\cos(u_s \tan(u_\phi)t) + 1)$. The velocity obstacle for the simple car and linearly moving obstacle uses the distance function,

$$D(u) = \sqrt{\left(\frac{\sin(u_s \tan(u_s)t)}{\tan u_\phi} - p_{B_x} - v_{B_x} t\right)^2 + \left(-\frac{\cos(u_s \tan(u_\phi)t)}{\tan u_\phi} + \frac{1}{\tan u_\phi} - p_{B_y} - v_{B_y} t\right)^2}, \tag{17}$$

and seeks to find its minimum by searching for a zero in the derivative of $D(u)^2$,

$$\begin{aligned} \frac{dD(u)^2}{dt} &= 2\left(\frac{\sin(u_s \tan(u_\phi)t)}{\tan u_\phi} - p_{B_x} - v_{B_x} t\right) \\ &\quad * \left(\frac{\cos(u_s \tan(u_\phi)t) u_s \tan u_\phi}{\tan u_\phi} - v_{B_x}\right) \\ &+ 2\left(-\frac{\cos(u_s \tan(u_\phi)t)}{\tan u_\phi} + \frac{1}{\tan u_\phi} - p_{B_y} - v_{B_y} t\right) \\ &\quad * \left(\frac{\sin(u_s \tan(u_\phi)t) u_s \tan u_\phi}{\tan u_\phi} - v_{B_y}\right). \end{aligned} \tag{18}$$

With these defined, we can proceed along the lines of Algorithm 1.