# Recursive Probabilistic Velocity Obstacles for Reflective Navigation

Boris Kluge
Research Institute for Applied Knowledge Processing
Helmholtzstr. 16
89081 Ulm, Germany
kluge@faw.uni-ulm.de

## Abstract

*An approach to motion planning among moving obstacles is presented, whereby obstacles are modeled as intelligent decision-making agents. The decision-making processes of the obstacles are assumed to be similar to that of the mobile robot. A probabilistic extension to the velocity obstacle approach is used as a means for navigation and modeling uncertainty about the moving obstacles' decisions.*

## 1 Introduction

Motion planning for a robot in an environment containing obstacles is an old and basic problem in robotics. For the task of navigating a mobile robot among moving obstacles, numerous approaches have been proposed. However, moving obstacles are most commonly assumed to be traveling without having any perception or motion goals (i.e. collision avoidance or goal positions) of their own.

In the expanding domain of mobile service robots deployed in natural, everyday environments, this assumption does not hold, since humans (which are the moving obstacles in this context) do perceive the robot and its motion and adapt their own motion accordingly. Therefore, reflective navigation approaches which include reasoning about other agents' navigational decision processes become increasingly interesting.

In this paper an approach to reflective navigation is presented which extends the velocity obstacle navigation scheme to incorporate reasoning about other objects' perception and motion goals.

### 1.1 Related Work

Predictive navigation is a domain where a prediction of the future motion of the obstacles is used to yield more successful motion (with respect to travel time or collision avoidance), see for example [3, 6]. However, reflective navigation approaches are an extension of this concept, since they include further reasoning about perception and navigational processes of moving obstacles.

The velocity obstacle paradigm, which belongs to the class of predictive navigation schemes, has been presented by Fiorini and Shiller [2] for obstacles moving on straight lines, and has been extended by Shiller et al. [7] for obstacles moving on arbitrary (but known) trajectories.

Modeling other agents' decision making similar to the own agent's decision making is used by the recursive agent modeling approach [4], where the own agent bases its decisions not only on its models of other agents' decision making processes, but also on its models of the other agents' models of its own decision making, and so on (hence the label *recursive*).

### 1.2 Overview

The remainder of this paper is organized as follows: The basic velocity obstacle approach is briefly introduced in Section 2, and its probabilistic extension is presented in Section 3. Now being able to cope with uncertain obstacle velocities, Section 4 describes how to recursively apply the velocity obstacle scheme in order to create a reflective navigation behavior. An implementation of the approach and an experiment are given in Section 5. After discussing the presented work in Section 6, Section 7 concludes the paper.

## 2 Review: Velocity Obstacles

This section gives a brief introduction to the original velocity obstacle approach [2].

Let $B_i$ and $B_j$ be circular objects with centers $c_i$ and $c_j$ and radii $r_i$ and $r_j$, moving with constant velocities $v_i = \dot{c}_i$ and $v_j = \dot{c}_j$. To decide if these two objects are on a collision course, it is sufficient to con-

sider their current positions together with their relative velocity $v_{ij} = v_i - v_j$, see Fig. 1. Let

$$\hat{B}_{ij} = \{c_j + r \,|\, r \in \mathbb{R}^2, |r| \leq r_i + r_j\}, \quad (1)$$

$$\lambda_{ij} = \{c_i + \mu v_{ij} \,|\, \mu \geq 0\}. \quad (2)$$

Then $B_i$ and $B_j$ are on a collision course, if and only if $\hat{B}_{ij} \cap \lambda_{ij} \neq \emptyset$.

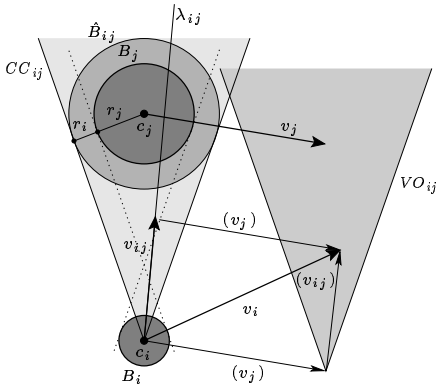


Figure 1: Collision cone and velocity obstacle

Therefore we can define a set of colliding relative velocities, which is called the *collision cone* $CC_{ij}$, as

$$CC_{ij} = \{v_{ij} \,|\, \hat{B}_{ij} \cap \lambda_{ij} \neq \emptyset\}. \quad (3)$$

In order to be able to decide if an absolute velocity $v_i$ of $B_i$ leads to a collision with $B_j$, we define the *velocity obstacle* of $B_j$ for $B_i$ to be the set

$$VO_{ij} = \{v_i \,|\, (v_i - v_j) \in CC_{ij}\}, \quad (4)$$

which is, in other words,

$$VO_{ij} = v_j + CC_{ij}. \quad (5)$$

Now for $B_i$, any velocity $v_i \in VO_{ij}$ will lead to a collision with $B_j$, and any velocity $v_i \notin VO_{ij}$ for $B_i$ will avoid any collision with $B_j$.

In general, object $B_i$ is confronted with more than one other moving object. Let $\mathcal{B} = \{B_1, \ldots, B_n\}$ the set of moving objects under consideration. The velocity obstacle of $\mathcal{B}$ for $B_i$ is defined as the set

$$VO_i = \cup_{j \neq i} VO_{ij}. \quad (6)$$

For any velocity $v_i \notin VO_i$, object $B_i$ will not collide with any other object.

Finally, a simple navigation scheme based on velocity obstacles can be constructed as following. The moving and non-moving obstacles in the environment are continuously tracked, and the corresponding velocity obstacles are repeatedly computed. In each cycle, a velocity is chosen which avoids collisions and approaches a motion goal, for example a maximum velocity towards a goal position, or a velocity minimizing the difference to a target velocity in order to coordinate the motion with another object [5].

## 3 Probabilistic Velocity Obstacles

Let $B_i$ and $B_j$ be circular objects with centers $c_i$ and $c_j$ and radii $r_i$ and $r_j$ as in the preceeding section. However, now we will consider uncertainty in shape and velocity of the objects. This allows to reflect the limitations of real sensors and object tracking techniques.

### 3.1 Shape Uncertainty

We define the probabilistic collision cone of object $B_j$ relative to object $B_i$ to be a function

$$PCC_{ij} : \mathbb{R}^2 \to [0, 1] \quad (7)$$

where $PCC_{ij}(v_{ij})$ is the probability of $B_i$ to collide with $B_j$ if $B_i$ moves with velocity $v_{ij}$ relative to $B_j$ (see Fig. 2).

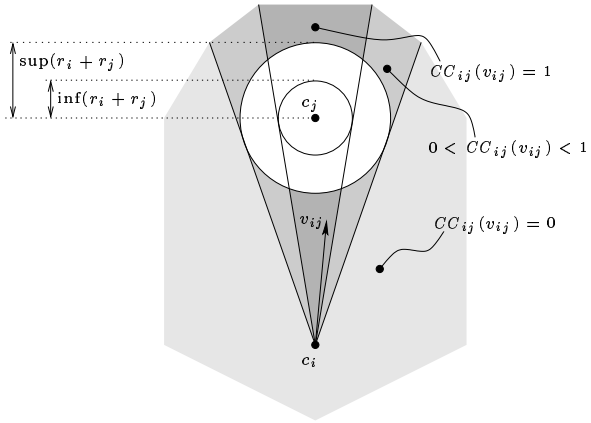


Figure 2: Probabilistic collision cone

Uncertainty of the collision cone is induced by uncertainty in the exact shape of objects. For example if the radii of the objects are only known by their probability distributions, we have

$$PCC_{ij}(v_{ij}) = \text{Prob}\left[r_i + r_j > \min_{\mu > 0} |c_i - c_j + \mu v_{ij}|\right]. \quad (8)$$

### 3.2 Velocity Uncertainty

Furthermore, there may be uncertainty about object velocities. Therefore we represent the velocity of object $B_j$ by a probability density function

$$V_j : \mathbb{R}^2 \to \mathbb{R}_0^+. \quad (9)$$

Now we define the probabilistic velocity obstacle of object $B_j$ relative to object $B_i$ as a function

$$PVO_{ij} : \mathbb{R}^2 \to [0,1] \tag{10}$$

which maps absolute velocities $v_i$ of $B_i$ to the according probability of colliding with $B_j$. It is

$$PVO_{ij}(v_i) = \int_{\mathbb{R}^2} V_j(v) PCC_{ij}(v_i - v) d^2v \tag{11}$$

which is equivalent to

$$PVO_{ij} = V_j * PCC_{ij} \tag{12}$$

where $*$ denotes the convolution of two function. Notice the structural similarity between Equations 5 and 12.

### 3.3 Probabilistic Velocity Obstacle

The probability of $B_i$ colliding with any other obstacle when traveling with velocity $v_i$ is the probability of not avoiding collisions with any other moving obstacle. Therefore we may define the probabilistic velocity obstacle for $B_i$ as the function

$$PVO_i = 1 - \prod_{j \neq i} (1 - PVO_{ij}) \,. \tag{13}$$

### 3.4 Navigating with Probabilistic VO

In the deterministic case, navigating is rather easy since we consider only collision free velocities and can choose a velocity which is optimal for reaching the goal. But now, we are confronted with two objectives: reaching a goal and minimizing the probability of a collision.

Let $U_i : \mathbb{R}^2 \to [0,1]$ a function representing the utility of velocities $v_i$ for the motion goal of $B_i$. However, the full utility of a velocity $v_i$ is only attained if (a) $v_i$ is dynamically reachable, and (b) $v_i$ is collision free. Therefore we define the relative utility function

$$RU_i = U_i \cdot D_i \cdot (1 - PVO_i), \tag{14}$$

where $D_i : \mathbb{R}^2 \to [0,1]$ describes the reachability of a new velocity.

Now a simple navigation scheme for object $B_i$ based on probabilistic velocity obstacles is obtained by repeatedly choosing a velocity $v_i$ which maximizes the relative utility $RU_i$.

### 3.5 Implementation Problems

Objects like probabilistic velocity obstacles as presented above are continuous functions over $\mathbb{R}^2$. Therefore any implementation has to deal with a discretization of the used objects and a restriction to a finite number of cells per object.

#### 3.5.1 Discretization

Step functions $s : \mathbb{R}^2 \to \mathbb{R}$ with $s(x,y) = s(x',y')$ for $i\kappa \leq x, x' < (i+1)\kappa$ and $j\kappa \leq y, y' < (j+1)\kappa$ are used for discretization in this paper. In other words we use functions which are piecewise constant on squares of size $\kappa^2$, where $\kappa$ is a predefined constant.

For a point $p = (x,y) \in \mathbb{R}^2$, its discretization is

$$p^\square = \left( \left\lfloor \frac{x}{\kappa} \right\rfloor , \left\lfloor \frac{y}{\kappa} \right\rfloor \right) \in \mathbb{Z}^2. \tag{15}$$

Accordingly, for a discretized point $p^\square = (z_1, z_2) \in \mathbb{Z}^2$ we define its cell as

$$cell(z_1, z_2) = [z_1\kappa, (z_1+1)\kappa[ \times [z_2\kappa, (z_2+1)\kappa[ \,. \tag{16}$$

For any function $F : \mathbb{R}^2 \to [0,1]$ we define the discretization of $F$ to be the function $F^\square : \mathbb{Z}^2 \to [0,1]$ with

$$F^\square(z_1, z_2) = \frac{1}{\kappa^2} \int_{cell(z_1,z_2)} F(x,y)\, dx\, dy, \tag{17}$$

i.e. $F^\square(z_1, z_2)$ is the average of $F$ on $cell(z_1, z_2)$.

Finally, for a discretized function $F^\square$,

$$\sigma(F^\square) = \{(x,y) \in \mathcal{Z}^2 \mid F^\square(x,y) > 0\} \tag{18}$$

denotes the supporting set of $F^\square$, which is the set of cells on which the discretized function does not vanish. The property

$$\sigma(F^\square G^\square) = \sigma(F^\square) \cap \sigma(G^\square) \tag{19}$$

is easily shown. Furthermore,

$$\sum_{p^\square \in \mathbb{Z}^2} F^\square(p^\square)\kappa^2 = \int_{\mathbb{R}^2} F(p) d^2p \tag{20}$$

holds.

#### 3.5.2 Restriction

Now we discuss the restriction problem in the context of navigating object $B_i$. Assuming the velocity of any other object $B_j$ to be bounded or to be known with bounded error, the supporting set $\sigma(V_j^\square)$ is finite. Therefore, $PVO_{ij}^\square(v_i^\square)$ can be computed for any $v_i$ by using

$$PVO_{ij}^\square(v_i^\square) = \sum_{v^\square \in \sigma(V_j^\square)} V_j^\square(v^\square) PCC_{ij}^\square(v_i^\square - v^\square)\kappa^2, \tag{21}$$

which is the discrete version of Equation 11.

Furthermore, for any real object $B_i$ the set $\sigma(D_i^\square)$ is finite, as any bounded acceleration applied to a body

of non-zero mass for a bounded period of time results in a bounded change of velocity. Since only velocities from $\sigma(RU_i^{\square})$ will be considered for navigating $B_i$, and since $\sigma(RU_i^{\square}) \subseteq \sigma(D_i^{\square})$, we can restrict velocities to the finite domain $\sigma(RU_i^{\square})$.

#### 3.5.3 Algorithm

Combining the results from the previous subsections, we get

$$PVO_i^{\square} = 1 - \prod_{j \neq i} (1 - PVO_{ij}^{\square}) \tag{22}$$

and further

$$\begin{aligned} RU_i^{\square}(v_i^{\square}) &= D_i^{\square}(v_i^{\square}) U_i^{\square}(v_i^{\square})(1 - PVO_i^{\square}(v_i^{\square})) \\ &= D_i^{\square}(v_i^{\square}) U_i^{\square}(v_i^{\square}) \prod_{j \neq i} (1 - PVO_{ij}^{\square}(v_i^{\square})) \end{aligned} \tag{23}$$

for any $v_i^{\square} \in \sigma(RU_i^{\square})$.

Assuming that $V_j^{\square}$, $PCC_{ij}^{\square}$, $D_i^{\square}$, and $U_i^{\square}$ can be computed in time $\mathcal{O}(1)$ (per function call), we can compute $PVO_{ij}^{\square}(v_i^{\square})$ in time $\mathcal{O}(|\sigma(V_j^{\square})|)$ (see Eq. 21) and $RU_i^{\square}(v_i^{\square})$ in time $\mathcal{O}(\sum_{j \neq i} |\sigma(V_j^{\square})|)$ (see Eq. 23). Finally, a discrete velocity $v_i^{\square}$ maximizing $RU_i^{\square}$ can be found in time

$$\mathcal{O}\left( |\sigma(D_i^{\square})| \cdot \sum_{j \neq i} |\sigma(V_j^{\square})| \right). \tag{24}$$

That is to say the dependence of the running time on the number of obstacles is only linear, but the dependence on the discretization is $\mathcal{O}(1/\kappa^4)$.

## 4 Recursive Probabilistic Velocity Obstacles

Traditionally, when navigating a mobile robot among moving obstacles (like humans), these obstacles' abilities to avoid collisions and their resulting motion behaviors are not taken into account. In contrast to this plain obstacle modeling, recursive modeling techniques presume the opponents (or more generally, the interaction partners) to deploy decision making processes for navigation similar or equal to the own approach.

### 4.1 Obstacle Modeling

Obstacles are assumed to perceive their environment and deduce according reactions, the reasoning process being similar to that of the robot. That is, any obstacle $B_j$ is assumed to take actions maximizing its relative utility function $RU_j$ (see Equation 14). Therefore, in order to predict the action of obstacle $B_j$, we need to know its current utility function $U_j$, dynamic capabilities $D_j$, and velocity obstacle $PVO_j$.

The utility of velocities can be inferred by recognition of the current motion goal of the moving obstacle. For example, Bennewitz et al. [1] learn and recognize typical motion patterns of humans. If no global motion goal is available through recognition, one can still assume that there exists such a goal which the obstacle strives to approach, expecting it to be willing to keep its current speed and heading.

By continuous observation of a moving obstacle it is also possible to deduce a model of its dynamics, which describes feasible accelerations depending on its current speed and heading.

Finally, the velocity obstacle $PVO_j$ for obstacle $B_j$ is simply computed using the world and self models of robot $B_i$, assuming similar perception among the objects.

Using this information, the future motion of an obstacle can be predicted by taking its relative utility function as a probabilistic description of its future velocity.

### 4.2 Recursive Modeling

Having such obstacle models as described above, the predicted obstacle velocities can be used to replace the observed velocities in the own obstacle avoidance scheme. Of course this reflective modeling is not restricted to recursion of depth one by a matter of principle. However, computational demands will increase heavily with the depth of the recursion, and intuitively, one does not expect recursion depths of more than two or three to be of broad practical value, since such deeper modeling is not observed when we are walking as human beings among other humans.

Note that such deeper models of moving obstacles are prerequisite for more sophisticated reflective navigation approaches in order to be able to deceive and feint particularly malevolent obstacles like deliberate obstructors. However being dreams of the future, such potential abilities indicate the importance of reflective navigation approaches and their investigation.

### 4.3 Formalization

This section presents a formal approach to recursive models in the given context, which serves as a basis for an implementation.

#### 4.3.1 Models of Functions

Let $F, G : \mathbb{R}^2 \to [0,1]$ functions, $i, i_1, i_2, \cdots \in \mathbb{N}$. Then $\mu_i F : \mathbb{R}^2 \to [0,1]$ is called the model by agent $B_i$ of $F$, and $\mu_{i_2}\mu_{i_1} F$ is called the model by agent $B_{i_2}$ of the model by agent $B_{i_1}$ of $F$ and so forth.

We want to be able to accept another argument in our notation and will write $\mu_i^d F$ to denote a depth-

$d$ recursive model of $F$ by agent $B_i$, which is called recursive since it may depend on further models of at most depth $d-1$.

The following two rules are introduced for models of functions. Firstly, agents are modeled as computing correctly:

$$\mu_i(F \circ G) = \mu_i F \circ \mu_i G \quad \text{for } \circ \in \{+, \cdot, *\}. \tag{25}$$

Secondly, an agent's model of an own model is the own model itself:

$$\mu_i \mu_i^d F = \mu_i^d F \quad \text{and} \quad \mu_i^d \mu_i F = \mu_i F. \tag{26}$$

#### 4.3.2 Recursion for Reflective Navigation

With the above definitions, the central proposition of reflective navigation using probabilistic velocity obstacles can be expressed as

$$\mu_i^d RU_j = \begin{cases} \mu_i U_j \mu_i D_j (1 - \mu_i \mu_j^{d-1} PVO_j) & \text{if } d > 0, \\ \mu_i U_j \mu_i D_j & \text{else,} \end{cases} \tag{27}$$

together with

$$\mu_i^d V_j = \begin{cases} 1/w\, \mu_i^d RU_j & \text{if } d > 0,\ w = \int_{\mathbb{R}^2} \mu_i^d RU_j(v) d^2 v \\ & \text{exists, and } w > 0, \\ \mu_i V_j & \text{else.} \end{cases} \tag{28}$$

Equation 27 expresses that each object is assumed to deduce its relative utility from recursive probabilistic velocity obstacle considerations, while Equation 28 expresses the assumption that objects will move according to their relative utility function.

To navigate a mobile robot $B_i$ using depth-$d$ recursive probabilistic velocity obstacles, we repeatedly choose a velocity $v_i$ maximizing $\mu_i^d RU_i$. For $d = 0$, we get a behavior that only obeys the robot's utility function $U_i$ and its dynamic capabilities $D_i$, but completely ignores other obstacles. For $d = 1$, we get the plain probabilistic velocity obstacle behavior as described in Section 3. Something new happens for $d > 1$, when the robot starts modeling the obstacles as perceptive and decision making.

Figure 3 illustrates a simple depth-2 evaluation example with three agents, whereby the tree is evaluated in postorder. To find its depth-2 relative utility, object 0 models the depth-1 relative utilities of the other objects and deploys them as a prediction of these other objects' velocities.

For $\mathcal{M} = \mu_{i_1} \dots \mu_{i_n}$, Figure 4 illustrates the general recursion step with more details. One can see that for a recursive velocity model (the root node), according

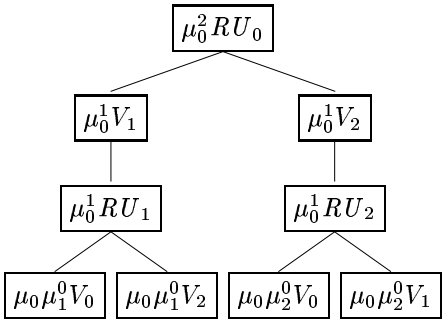


Figure 3: Recursive PVO Evaluation

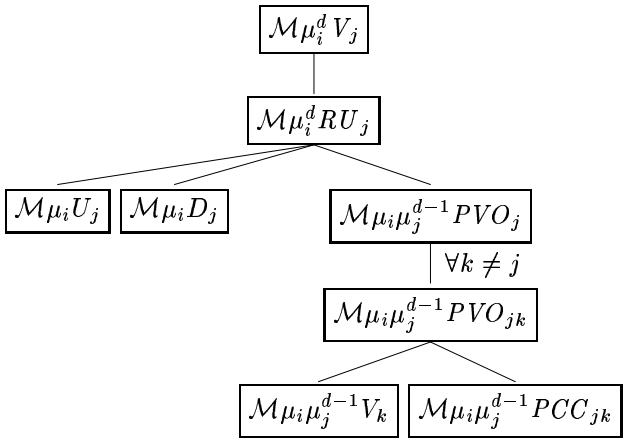


Figure 4: Recursive PVO Evaluation

models of utility and dynamic capabilities, as well as lower depth recursive models of velocities and collision cones for any other object are required (the leaf nodes). Leaf nodes may need to be expanded further, until only plain, non-recursive models remain. Therefore, the size of a fully expanded evaluation tree will grow exponentially with increasing recursion depth.

## 5 Implementation

For the implementation we drop the differentiation between models by different agents, and write simply $\mu F$ for a model of $F$ and $\mu^d F$ for a depth-$d$ recursive model of $F$. With this simplification, the dependence of the complexity on the recursion depth is reduced to linear, since the number of models to be computed is equal on each recursion level. Algorithm 1 gives the details of the implementation. However not indicated for simplicity, functions are discretized according to the approach presented in Section 3.5.1.

### 5.1 Complexity

We begin the complexity assessment by measuring the sizes of the supporting sets of the discretized func-

**Algorithm 1** RPVO(DEPTH $r$, $n$ OBJECTS)

1: **Input:** for $i, j = 1, \ldots, n$, $j \neq i$
   - object descriptions for $PCC_{ij}$
   - velocities $V_i$
   - dynamic capabilities $D_i$
   - utilities $U_i$

2: **for** $i = 1, \ldots, n$ **do**
3: $\mu^0 V_i \leftarrow V_i$
4: $\mu^0 RU_i \leftarrow D_i U_i$
5: **end for**
6: **for** $d = 1, \ldots, r$ **do**
7: **for** $i = 1, \ldots, n$ **do**
8: $\mu^d RU_i \leftarrow D_i U_i \prod_{j \neq i} (1 - \mu^{d-1} V_j * PCC_{ij})$
9: $w \leftarrow \int_{\mathbb{R}^2} \mu^d RU_i(v) d^2 v$
10: **if** $w > 0$ **then**
11: $\mu^d V_i \leftarrow (1/w) \mu^d RU_i$
12: **else**
13: $\mu^d V_i \leftarrow \mu^0 V_i$
14: **end if**
15: **end for**
16: **end for**
17: **Output:** recursive models $\mu^d V_i$ and $\mu^d RU_i$ for $i = 1, \ldots, n$ and $0 \leq d \leq r$

tions used in Algorithm 1. Line 4 implies

$$\sigma(\mu^0 RU_i^{\square}) \subseteq \sigma(D_i^{\square}), \tag{29}$$

and from line 8 follows

$$\sigma(\mu^d RU_i^{\square}) \subseteq \sigma(D_i^{\square}) \tag{30}$$

for $d > 0$. Line 3 implies

$$\sigma(\mu^0 V_i^{\square}) = \sigma(V_i^{\square}), \tag{31}$$

and from lines 11 and 13 follows

$$\sigma(\mu^d V_i^{\square}) \subseteq \sigma(D_i^{\square}) \cup \sigma(V_i^{\square}) \tag{32}$$

for $d > 0$, using the three preceding Equations.

Now we count the numbers of operations used in the algorithm, which we write down using $N_i = |\sigma(D_i^{\square}) \cup \sigma(V_i^{\square})|$. Line 8 requires $\mathcal{O}(N_i \cdot \sum_{j \neq i} N_j)$ operation (Equation 24). Lines 9, 11, and 13 each require $\mathcal{O}(N_i)$ operations, and are thus dominated by line 8. Therefore the loop starting in line 7 requires

$$\mathcal{O}(\sum_{i=1}^{n} (N_i \sum_{j \neq i} N_j)) \tag{33}$$

operations, and the loop starting in line 6 requires

$$\mathcal{O}(r \sum_{i=1}^{n} (N_i \sum_{j \neq i} N_j)) \tag{34}$$

operations. The complexity of the loop starting in line 6 clearly dominates the complexity of the initialization loop starting in line 2. Therefore Equation 34 gives an upper bound of the overall time complexity of our implementation. That is to say the dependence on the recursion depth is linear, the dependence on the number of objects is $\mathcal{O}(n^2)$, and the dependence on the discretization remains $\mathcal{O}(1/\kappa^4)$.

## 5.2 Experiments

Figure 5 shows the situation of an experiment, where object 0 and object 1 are moving towards each other with a slight offset. The positions of object 0 and 1 are $(-1, 0.05)$ and $(1, 0)$, respectively, and their mean velocities are $(0.5, 0)$ and $(-0.5, 0)$ with a bounded conical distribution (maximum difference to mean is 0.05). Both objects are assumed to be able to change their velocity in one step by at most 0.15 in any direction. The utilities are conical distributions, too, with their maximums at $(0.7, 0)$ and $(-0.7, 0)$.

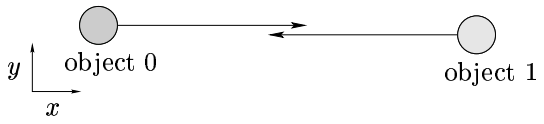


Figure 5: Experiment Situation

Figure 6 shows the computed models of relative utility for this situation. Illustrations for the models of velocity are omitted since they are only scaled versions of the presented figures. The left (right) column shows models for object 0 (object 1). The recursive depth increases from depth 0 at the top line to depth 2 at the bottom line.

The top line show the plain relative utility, obeying only the capabilities and the goals of the objects. Both objects aim at accelerating their current motion.

The second line shows the results of a single step probabilistic velocity obstacle evaluation. Clearly, moving straight ahead is of low utility, since this will lead to a collision. Since object 0 is a little bit offset in the direction of the positive $y$-axis, we get higher relative utility for velocities (see the isolines in the figures) which are offset in the same direction, as velocities with lower $y$-components which still evade the collision have lower utility to object 0. The situation for object 1 is more or less analogous.

Finally, line three shows depth-2 recursive models of relative utility. Each object assumes the other object to be moving according to its depth-1 relative utility.

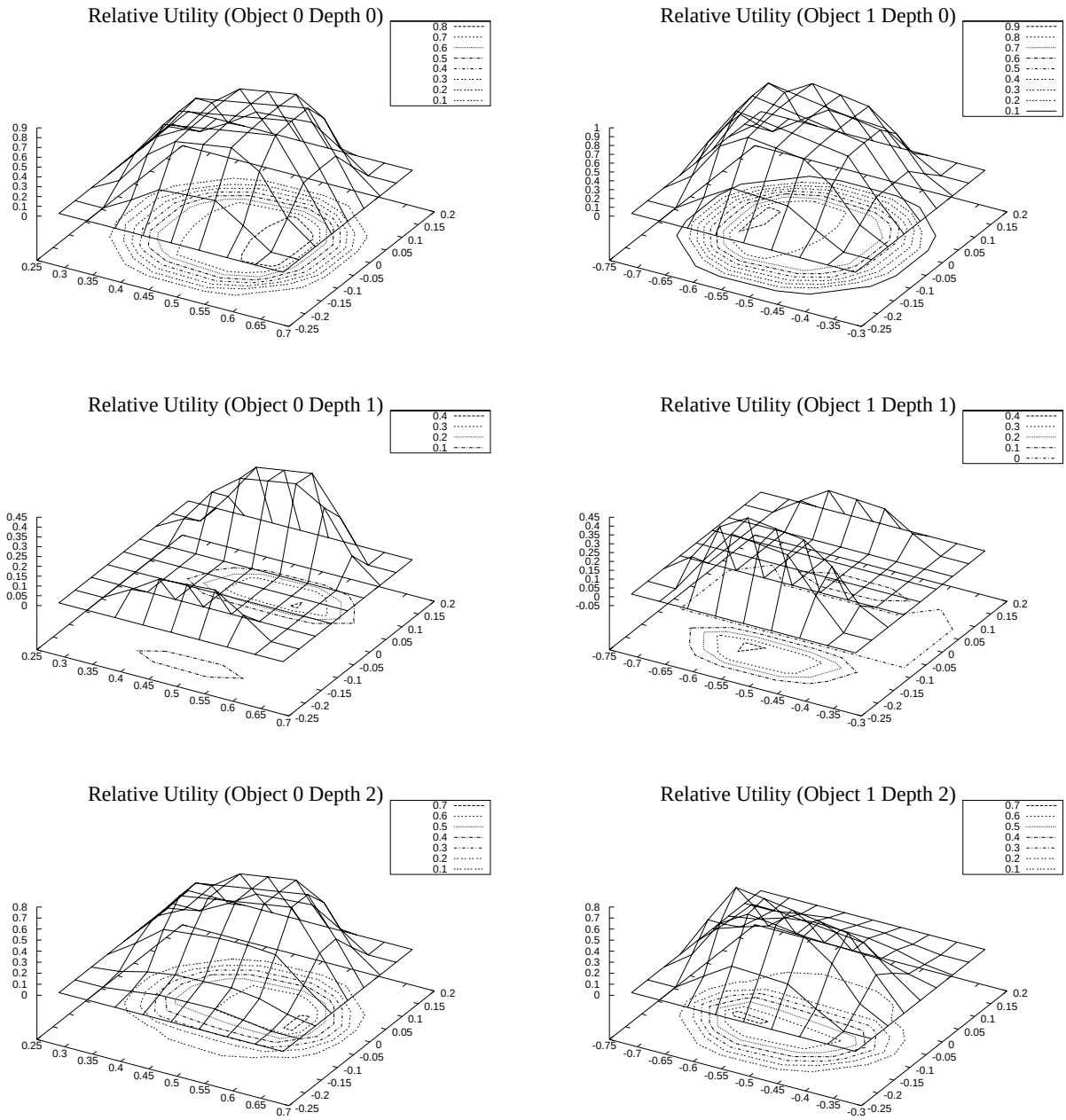


Figure 6: Models of Relative Utility

## 6 Discussion

Determining the correct evaluation depth is not obvious. If the time to collision is still large enough, we can use deeper models to possibly achieve better motion coordination. Otherwise, if obstacles and collisions are imminent, we better give in and use a smaller depth of recursion. (Imagine a "trial of courage" scenario where two vehicles are driving towards each other, and the first to perform a collision avoidance maneuver is the "loser".) An alternative approach might try to combine different depth relative utility models of an object.

Time horizons can be included in the presented approach in order not to react to heavily on still distant objects. See the original velocity obstacle works [2] for details on time horizons.

Oscillations might appear in models for successive depths. Reconsider the example from Section 5.2 with both objects facing each other. Assume at depth $d$, both objects avoid a collision by deviating to the left or to the right. Then in depth $d+1$, none of the objects will perform an avoidance maneuver, since each object's depth-$d$ model of the other object predicts that other object to avoid the collision. Subsequently, in depth $d+2$, both objects will perform collision avoidance maneuvers again, an so on.

A rather different aspect of the presented recursive modeling scheme is that it can serve as a basis for an approach to reasoning about the objects in the environment. That is to say, one could compare the observed motion of the objects to the motion that was predicted by recursive modeling, possibly discovering relationships among the objects. An example for such a relationship is deliberate obstruction, when one object obtrusively refrains from collision avoidance.

Finally, deeper models of the interaction partners are required for effectively generating unexpected actions. If $\mu_j U_i$ "differs notably" from $U_i$, but $\mu_i \mu_j U_i$ is "rather close" to $\mu_j U_i$, agent $i$ can detect the difference between $\mu_i \mu_j U_i$ and $U_i$ and exploit this situation by doing something that is unexpected and therefore unobstructed by agent $j$.

## 7 Conclusion

An approach to motion coordination in dynamic environments has been presented, which reflects the peculiarities of natural, populated environments: obstacles are not only moving, but also perceiving and making decisions based on their perception.

The approach can be seen as a twofold extension of the velocity obstacle framework. Firstly, object velocities and shapes may be known and processed with respect to some uncertainty. Secondly, the perception and decision making of other objects is modeled and included in the own decision making process.

Experimental evaluation has been done on a small number of static situations only. Implementation on the robot "MAid" (autonomous wheelchair) for real world tests as well as investigation of the approach in a dynamic simulation environment are ongoing.

### Acknowledgments

This work was supported by the German Department for Education and Research (BMB+F) under grant no. 01 IL 902 F6 as part of the project MORPHA.

## References

[1] M. Bennewitz, W. Burgard, and S. Thrun. Learning motion patterns of persons for mobile service robots. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2002.

[2] P. Fiorini and Z. Shiller. Motion planning in dynamic environments using velocity obstacles. *International Journal of Robotics Research*, 17(7): 760–772, July 1998.

[3] A. F. Foka and P. E. Trahanias. Predictive autonomous robot navigation. In *Proceedings of the 2002 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, pages 490–495, EPFL, Lausanne, Switzerland, Oct. 2002.

[4] P. J. Gmytrasiewicz. *A Decision-Theoretic Model of Coordination and Communication in Autonomous Systems (Reasoning Systems)*. PhD thesis, University of Michigan, 1992.

[5] B. Kluge, D. Bank, and E. Prassler. Motion coordination in dynamic environments: Reaching a moving goal while avoiding moving obstacles. In *Proc. of 11th IEEE International Workshop on Robot and Human Interactive Communication*, 2002.

[6] J. Miura and Y. Shirai. Modeling motion uncertainity of moving obstacles for robot motion planning. In *Proc. of Int. Conf. on Robotics and Automation (ICRA)*, 2000.

[7] Z. Shiller, F. Large, and S. Sekhavat. Motion planning in dynamic environments: Obstacles moving along arbitrary trajectories. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, pages 3716–3721, Seoul, Korea, May 2001.