Multi-robot collision avoidance with localization uncertainty

Daniel Hennes
Maastricht University
P.O. Box 616, 6200MD
Maastricht, The Netherlands
daniel.hennes@gmail.com

Daniel Claes
Maastricht University
P.O. Box 616, 6200MD
Maastricht, The Netherlands
danielclaes@me.com

Wim Meeussen
Willow Garage
68 Willow Rd., Menlo Park
CA 94025, USA
meeussen@willowgarage.com

Karl Tuyls
Maastricht University
P.O. Box 616, 6200MD
Maastricht, The Netherlands
k.tuyls@maastrichtuniversity.nl

ABSTRACT

This paper describes a multi-robot collision avoidance system based on the velocity obstacle paradigm. In contrast to previous approaches, we alleviate the strong requirement for perfect sensing (i.e. global positioning) using Adaptive Monte-Carlo Localization on a per-agent level. While such methods as Optimal Reciprocal Collision Avoidance guarantee local collision-free motion for a large number of robots, given perfect knowledge of positions and speeds, a realistic implementation requires further extensions to deal with inaccurate localization and message passing delays. The presented algorithm bounds the error introduced by localization and combines the computation for collision-free motion with localization uncertainty. We provide an open source implementation using the Robot Operating System (ROS). The system is tested and evaluated with up to eight robots in simulation and on four differential drive robots in a real-world situation.

Categories and Subject Descriptors

I.2.9 [**Artificial Intelligence**]: Robotics

General Terms

Algorithms, Experimentation

Keywords

multi-robot systems, optimal reciprocal collision avoidance, adaptive monte-carlo localization, robot operating system

1. INTRODUCTION

Local collision avoidance is the task of steering free of collisions with static and dynamic obstacles, while following a global plan to navigate towards a goal location. Figure 1 shows a configuration of two robots on collision course. The task of the algorithm is to avoid collision with a minimal deviation of the preferred path. Thus, local collision avoidance differs from motion planning, global path planning and local path planning. In motion planning the environment of the robot is assumed to be deterministic and known in advance, thus allowing to plan a complete path to the goal. Global path planners usually operate on a static map and find either the minimum cost plan (e.g. using A* or Dijkstra's algorithm) or any valid plan (e.g. sample based planners). Local path planners, such as Trajectory Rollout and Dynamic Window Approaches (DWA), perform forward simulations for a set of velocity commands; each resulting trajectory is scored based on proximity to the goal location and a cost map built from current sensor data. In principle this allows to stay clear of dynamical obstacles; however, in multi-robot settings two problems arise:

1. Robots are not merely dynamic obstacles; each robot itself is a pro-active agent taking actions to avoid collisions. Neglecting this might lead to *oscillations* and thus highly inefficient trajectories or even collisions.

2. The sensor source (e.g. laser range finder) is usually mounted on top of the robot's base to allow for a maximal unoccluded viewing angle. In a system with homogenous robots this implies that there is very little surface area that can be picked up by the sensors of other robots and thus prevents the robots from observing each other.

Local collision avoidance addresses these challenges and is an important building block in any robot navigation system targeted at multi-robot systems. Although robot localization is a requirement for collision avoidance, most approaches assume perfect sensing and positioning and avoid local methods by using global positioning via an overhead tracking camera - or are purely simulation based. Nevertheless, to be able to correctly perform local collision avoidance in a realistic environment, a robot needs a reliable position estimation without the help of external tools.

Appears in: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), June, 4–8, 2012, Valencia, Spain.

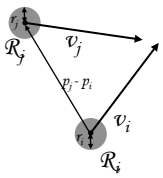


Figure 1: A workspace configuration with two robots R_i and R_j on collision course.

Our approach uses Optimal Reciprocal Collision Avoidance (ORCA) and the extension to non-holonomic robots (NH-ORCA) [1] in combination with Adaptive Monte-Carlo Localization (AMCL) [3].This effectively alleviates the need for global positioning by decentralized localization on a per-agent level. We provide a solution that is situated in between centralized motion planning for multi-robot systems and communication-free individual navigation. While actions will remain to be computed independently for each robot, information about position and velocity is shared using local inter-robot communication. This keeps the communication overhead limited while avoiding problems like robot-robot detection. The resulting algorithm is implemented in the open source Robot Operating System (ROS), which provides hardware abstraction and message-passing. Our experiments in simulation and on a physical system illustrate the feasibility and efficiency of the approach.

The remainder of the paper is structured as follows. Section 2 provides background information about ORCA, NH-ORCA, AMCL and ROS. Section 3 discusses key challenges in applying velocity-based collision avoidance to real-world robotic scenarios, leading to the proposed approach. In Section 4, we introduce our novel method to incorporate localization uncertainty. Experimental results are presented in Section 5. The paper concludes with a brief discussion and highlights future directions of this work in Section 6.

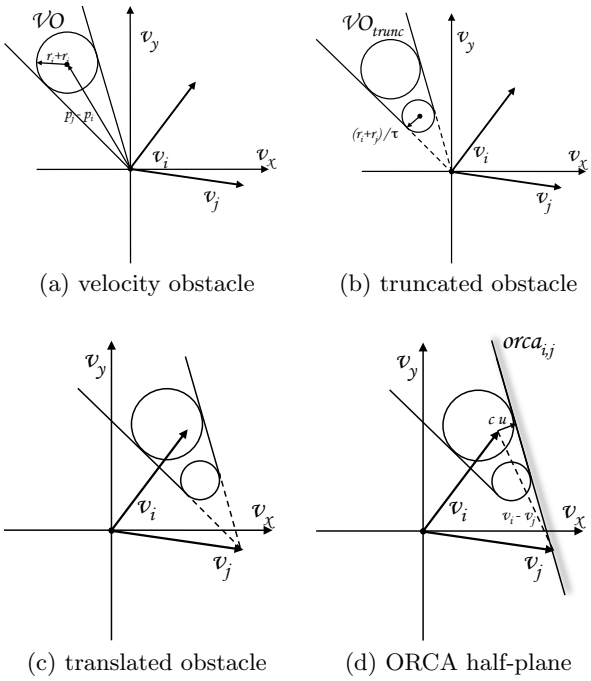
2. BACKGROUND

In this section, we will concisely describe the collision avoidance algorithms ORCA and NH-ORCA and the Adaptive Monte Carlo Localization (AMCL) method. Additionally, the Robot Operating System (ROS) will be introduced.

2.1 Optimal Reciprocal Collision Avoidance

Our work is based on the principle of *Optimal Reciprocal Collision Avoidance (ORCA)* introduced by van den Berg et al. [10], an extension of Reciprocal Velocity Obstacles (RVO) [11]. ORCA is a velocity-based approach [2] to achieve collision avoidance in multi-agent systems taking into account the motions of other agents. For simplicity, the two dimensional case is assumed, but the formulations can be extended into the third dimension.

ORCA describes a control policy were each agent selects a collision-free velocity from the two dimensional *velocity space* in x and y direction. This implies that a holonomic robot is assumed in the original formulation, since the robot has to be able to accelerate into every direction regardless of its current state. However, the movement model of a non-holonomic robot can be incorporated by limiting the velocity space accordingly, see Section 2.2.



(a) velocity obstacle (b) truncated obstacle

(c) translated obstacle (d) ORCA half-plane

Figure 2: Creating velocity obstacles and ORCA half-planes based on the workspace configuration shown in Figure 1. (a) Translating the situation into velocity space and the resulting velocity obstacle for R_i, assuming a static obstacle. (b) Truncating the velocity obstacle (VO) for time frame τ. Every velocity that is allowed will be collision-free for at least time τ. (c) Translating the VO according to the other robot's velocity v_j. Velocity v_i points into the translated VO, hence R_i is on collision course. (d) Creating an ORCA half plane for R_i. Vector u is the minimal vector to add to v_i to be outside of the VO. The perpendicular line to u at v_i+cu is the set of velocities that is closest to v_i and avoiding collisions for $c \geq 1$. For $c = \frac{1}{2}$ the robot R_i assumes that R_j will take care of the other half of the collision avoidance.

Let us assume a workspace configuration with two robots on a collision course as shown in Figure 1. If the position of agent R_j is known to R_i, a region in the robot's velocity space can be calculated which is leading to a collision under current velocities and is thus unsafe. If we assume disc-shaped robots, which are not moving, the region of non-allowed velocities is bounded by the half-lines emanating from the origin, tangent to a disk at the relative position of the two agents with the combined radius of the two robots as in Figure 2(a). These unsafe regions are called velocity obstacles (VO). Taking a velocity in direction of the other agent will not immediately result in a collision, hence the VO can be bounded by a given time frame τ, leading to a truncated cone. Taking a velocity which is now available again will be collision free for at least the given time frame, see Figure 2(b). If we now assume that both agents are moving, the VO has to be translated into the direction the speed of R_j as shown in Figure 2(c). Now, robot R_i's velocity vector v_i points into the VO, thus we know that R_i and R_j are on collision course. Each agent computes a VO for each of the other agents. If all agents at any given time step select

velocities outside of the VOs, the trajectories are guaranteed to be collision free. However, oscillations can still occur [11].

To overcome the problem of oscillations and to enable efficient calculation for safe velocities, *Optimal Reciprocal Collision Avoidance (ORCA)* was introduced by van den Berg et al. [10]. Instead of velocity obstacles, agents independently compute half-planes of collision-free velocities for each other agent as shown in Figure 2(d). The half planes are selected to be as close to the desired goal velocity as possible. Thus, they are parallel lines to either one of the two legs of the VO. If we assume reciprocal collision avoidance, the line can be slightly insight the VO, assuming that the other robot will take care of the other half of the collision avoidance. The intersection of all half planes is the set of collision free velocities. The optimal velocity from this set can be calculated by solving a linear program minimizing the distance to the desired goal velocity.

Though each agent selects a new velocity independently, a distributed implementation of ORCA on a physical system of mobile robots requires *perfect sensing* of the shape, position and velocities of other robots. A variant of the original RVO called *Hybrid Reciprocal Velocity Obstacles (HRVO)* is presented in [7]. This extension takes uncertainty of movement and sensing into account; however, it uses global positioning via an overhead camera and does not incorporate the ORCA formulation.

2.2 Kinematic constraints

As mentioned above, the original formulation of ORCA is based on holonomic robots, which can accelerate into any direction from every state. However, differential drive robots with only two motorized wheels are much more common due to their lower price point. To incorporate the differential drive constraints, Kluge et al. introduced a method to calculate the effective center of a differential drive robot [5]. The effective center represents a translation of the center of rotation to a point that can virtually move into all directions. It can be incorporated in the ORCA formulation by virtually enlarging the robots' radii prior to the calculations. These adaptations provide additional maneuverability to handle the differential drive constraints. ORCA-DD [8] extends this idea and enlarges the robot to twice the radius of the original size to ensure collision free and smooth paths for robots under differential constraints. The effective center is then located on the circumference of the robot at the center of the extended radius. However, this quadruples the virtual size of the robot, which can result in problems in narrow corridors or unstructured environments.

Another method to handle non-holonomic robot kinematics has been introduced by Alonso-Mora et al [1]: NH-ORCA is the generalized version of ORCA for any non-holonomic robot. The underlying idea is that any robot can track a holonomic speed vector with a certain tracking error ε. This error depends on the direction and length of the holonomic velocity, i.e. a differential drive robot can drive along an arc and then along a straight line which is close to a holonomic vector in that direction. A set of allowed holonomic velocities is calculated based on the current speed and a maximum tracking error ε. Resulting constraints are added to the linear program in the ORCA formulation. To allow smooth and collision free navigation, the virtual robot radii have to be increased by the tracking error ε, since the robots do not track the desired holonomic velocity exactly. Additionally, in dense configuration with many robots, turn in-place can be included by adapting the allowed tracking error ε dynamically depending on the current state (i.e. proximitiy to other robots and current velocities). The set of allowed holonomic velocities can be calculated for any possible angle and error. However, any further constraint in the linear program slows down the computation, thus the feasible set can be approximated by a polygon.

NH-ORCA is preferred over ORCA-DD, since the virtual increase of the robots' radii is only by a size of ε instead of doubling the radii.

2.3 Adaptive Monte-Carlo Localization

The localization method employed in our work is based on sampling and importance based resampling of particles, in which each particle represents a possible pose and orientation of the robot. More specifically, we use the adaptive monte-carlo localization method, which dynamically adapts the number of particles [3]. Monte-Carlo Localization (also known as a particle filter), is a widely applied localization method in the field of mobile robotics. It can be generalized in an initialization phase and two iteratively repeated subsequent phases, the prediction and the update phase.

In the initialization phase, a particle filter generates a number of samples N, which are uniformly distributed over the whole map of possible positions. In the 2.5D case, every particle s^i has a x- and y-value and a rotation $s^i = (\hat{x}, \hat{y}, \hat{\theta})$. The particles are usually initialized in such a way, that only valid positions are taken into account, i.e. they cannot be outside of the map or within walls.

The first iterative step is the prediction phase, in which the particles of the previous population are moved based on the motion model of the robot, i.e. the odometry. Afterwards, in the update phase, the particles are weighted according to the likelihood of the robot's measurement for each particle. Given this weighted set of particles the new population is resampled in such a way that the new samples are selected according to the weighted distribution of particles in the old population. In the following, the two phases are explained in further detail.

Prediction phase: After each movement, the position of each particle is updated according to the belief of the agent. More specifically, if the robot has moved forward 10 cm, each particle is moved 10 cm into the direction of its rotation. If the robot rotates, the particles are rotated accordingly. Thus, if a holonomic robot moves from state $\mathbf{x_k} = (x_k, y_k, \theta_k)$ to $\mathbf{x_{k+1}} = (x_{k+1}, y_{k+1}, \theta_{k+1})$, the particles are translated by:

$$\begin{bmatrix} \hat{x}_{k+1} \\ \hat{y}_{k+1} \\ \hat{\theta}_{k+1} \end{bmatrix} = \begin{bmatrix} \hat{x}_k + \rho cos(\hat{\theta}_k + \Delta\theta) \\ \hat{y}_k + \rho sin(\hat{\theta}_k + \Delta\theta) \\ \hat{\theta}_k + \Delta\theta \end{bmatrix} \quad (1)$$

Where $\rho = \sqrt{\Delta x^2 + \Delta y^2}$ and $\Delta\theta = \theta_k - \theta_{k+1}$. However, both ρ and $\Delta\theta$ are corrupted by noise due to errors in actuators and odometry. Hence, the more accurate the robot's motion model is, the better the performance of the prediction phase. For non-holonomic robots, the update equations can be changed accordingly [9].

Update phase: After a sensor update, the expected measurement for each particle is calculated. This means, measured sensor values are compared with the world view that

is expected if the robot would be at the position of the particle (i.e. by a laser scan matcher). The new weight (w_k^i) is the probability of the actual sensor measurement (z_k) given the particles position (s_k^i) at time k as shown below:

$$w_{k+1}^i = p(z_k|s_k^i) \tag{2}$$

As w is a probability distribution, the weight for each particle is re-normalized after each update:

$$w_k^i = \frac{w_k^i}{\sum_i w_k^i} \tag{3}$$

The resampling can be done in linear time as described in [9]. After resampling, all the weights are reset to the uniform weight of $1/N$.

Resampling and variable sample set size: Particle filters only need a large N to correctly identify the position when the initial state is unknown. However, when the present localization is quite accurate already, less particles are needed to keep track of the position changes. Hence, the number of samples can be changed adaptively depending on the position uncertainty. We use the approach of KLD-sampling, which determines the minimum number of samples needed, such that with probability $1 - \delta$ the error between the true posterior and the sample-based approximation is less than ε. The number of samples can be calculated as:

$$n = \frac{1}{2\varepsilon}\chi^2_{k-1,1-\delta} \tag{4}$$

This can be approximated using the Wilson-Hilferty transformation as:

$$n = \frac{k-1}{2\varepsilon}\left\{1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}}z_{1-\delta}\right\} \tag{5}$$

where k is the number of bins of the discrete distribution from which the particles are sampled. For further details we refer to [4].

Kidnapped robot and false localization: A common problem occurs if there are several locations which are represented similar according to the sensor values. For example two hallways, which have only one door on the right. In these cases, it happens that the robot localizes itself at the wrong position. Furthermore, the robot can be moved by an external force, like a human. To incorporate sudden changes or wrong localizations, a fraction of particles can be moved to a random location. This increases the robustness of the system.

In our work, AMCL is not used for global localization, but rather initialized with a location guess that is within the vicinity of the true position. This enables us to use AMCL for an accurate position tracking without having multiple possible clusters in ambiguous cases.

2.4 Robot Operating System (ROS)

The NH-ORCA and the AMCL algorithms are implemented in the framework of the open source *Robot Operating System (ROS)* [6]. ROS provides many useful tools, hardware abstraction and a message passing system between nodes. Nodes are self contained modules that run independently and communicate which each other over so called *topics* using a one-to-many subscriber model and the TCP/IP protocol. Naturally this is of great importance when working with distributed systems. In addition, the modularity enables to easily create various configurations for different settings; to run our system on ROS-enabled robots, only the parameters need to be adapted according to the robot's motion and sensor model.[1].

3. PROBLEM DESCRIPTION AND APPROACH

We propose a system that builds upon the two main components introduced in Section 2, i.e. NH-ORCA and AMCL, to provide collision free motion in a real-world system of robots. In this section we will revisit the assumptions commonly made by all velocity-based collision avoidance algorithms and motivate our choice for per agent-based localization in combination with position and velocity information sharing using inter-robot communication. Furthermore, we will point out the necessary addition of sensor uncertainty, leading to our proposed algorithm *Collision Avoidance with Localization Uncertainty (CALU)* explained in more detail in Section 4.

3.1 Problem description

ORCA (and all its variants) does not require any inter-robot negotiation to find optimal collision free motion trajectories and is hence in principal fully distributed. However, all methods require perfect information about the positions, velocities and shapes of all other robots. In order to preserve the distributed nature of this approach, robots need to be able to accurately identify other robots using on-board sensors; furthermore, positions and velocities have to be deduced from the same data. The list of typical sensors for mobile robots includes stereo cameras, laser range finders and lately 3D image sensors (e.g. Microsoft Kinect). These sensors deliver large data-streams that require considerable computational power to process even for the detection and classification of static obstacles.

The computational requirement is not the only problem when considering robot-robot detection. As low-end laser range finders (e.g. Hokuyo URG-04LX) become widely available even for mobile robotic projects on a small budget, they are the preferred sensor choice due to their high accuracy, resolution and field of view. However, the laser range finder is usually mounted on top of the robot's base to allow for a maximal unoccluded viewing angle. In a system with homogenous robots that means that there is very little surface area that can be picked up by the sensors of other robots and thus prevents the robots from observing each other.

Even though the laser range finder provides a high accuracy in the readings, the localization and tracking of the robot using AMCL will in general have the tendency to differ to some extent from the true position of the robot. If the size of the localization and tracking error is in the order of magnitude of the robots radius, collisions are bound to happen.

Previous approaches have worked around these problems by providing global positioning to all robots based on an overhead tracking camera. Such a system is not distributed since a host computer connected to the camera needs to process the sensor data and communicate with all robots to

[1]For more information see: `http://www.ros.org/`.

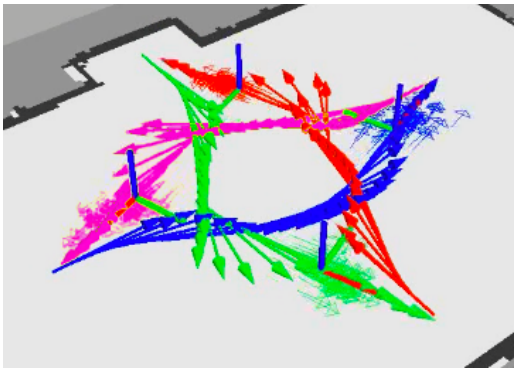


Figure 3: CALU with four robots. ROS visualization tool RVIZ is used to show the trajectories and localization particles of four robots.

provide position and velocity data. If this machine fails the system breaks.

3.2 Approach

We propose to utilize agent-based localization and inter-robot communication to provide a system that is more realistic in real-world scenarios (i.e. without the need for external positioning data) and also more robust (i.e. single component failure does not lead to system failure). Our approach, called Collision Avoidance with Localization Uncertainty (CALU), results in a fully decentralized system that uses local communication to share robot state information in order to ensure smooth collision free motion; an example for 4 robots is shown in Figure 3. Below we describe the four key components of this approach.

Platform: The robots are assumed to be differential drive robots. Required sensors are a laser range finder and wheel odometry. For simplicity we assume a circular footprint; other shapes can be approximated by the circumscribed radius. In order to connect the different subsystems, including device drivers and software modules, we use ROS (see Section 2.4).

Sensor processing and localization: Each robot integrates wheel odometry data which is in turn used to drive the motion model of AMCL (see Section 2.3), hence tracking the pose of the robot. Laser range finder scans are used in the update phase of AMCL. The uncertainty of the current localization, i.e. the spread and weight of the particles, is taken into account for the calculation of collision free velocities as will be explained in further detail in Section 4. We assume a prior static map that is used for localization and available to all robots, thus providing a consistent global coordinate frame.

Inter-robot communication: Each robot broadcasts its position and velocity information in the global coordinate frame on a common ROS topic. Each robot also subscribes to the same topic and caches position and velocity data of all other robots. Message delays are taken into account and positions are forward integrated in time according to the motion model of robots using the last known position and velocity information.

Collision avoidance: NH-ORCA (see Section 2.2) is used to compute optimal collision free velocities according to the aggregated position and velocity data of all surrounding robots. As a last step we incorporate localization uncertainty in the NH-ORCA computation as detailed in Section 4. The allowed tracking error is scaled depending on current speed of the robot.

4. LOCALIZATION UNCERTAINTY

The key idea of CALU is to bound the error introduced by localization. To derive this bound, we revisit the particle filter described in Section 2.3.

Let $\mathbf{x}_k = (x, y, \theta)$ be the state of the system. The posterior filtered density distribution $p(\mathbf{x}_k|\mathbf{z}_{1:k})$ can be approximated as:

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) \approx \sum_{i=1}^{N} w_k^i \; \delta\left(\mathbf{x}_k - \mathbf{s}_k^i\right) \tag{6}$$

where $\delta(\cdot)$ is the Dirac delta measure. We recall that a particle state at time k is captured by $\mathbf{s}_k^i = (\hat{x}_k^i, \hat{y}_k^i, \hat{\theta}_k^i)$. In the limit $N \to \infty$, Equation 6 approaches the real posterior density distribution. We can define the mean $\mu = (\mu_x, \mu_y, \mu_\theta)$ of the distribution accordingly:

$$\mu_x = \sum_i w_k^i \; \hat{x}_k^i \tag{7}$$

$$\mu_y = \sum_i w_k^i \; \hat{y}_k^i \tag{8}$$

$$\mu_\theta = atan2\left(\sum_i w_k^i \; sin(\hat{\theta}_k^i), \sum_i w_k^i \; cos(\hat{\theta}_k^i)\right) \tag{9}$$

The mean gives the current position estimate of the robot. However, the estimate is likely to be noisy and we have to take this uncertainty into account in order to ensure collision free motion. The probability of the robot residing within a certain area $\mathcal{A}$ at time k is:

$$p(\mathbf{x}_k \in \mathcal{A}|\mathbf{z_{1:k}}) = \int_{\mathcal{A}} p(\mathbf{x}|\mathbf{z_{1:k}}) d\mathbf{x} \tag{10}$$

We can rewrite (10) using (6) as follows:

$$p(\mathbf{x}_k \in \mathcal{A}|\mathbf{z_{1:k}}) \approx \sum_{\forall_i : \mathbf{s}_k^i \in \mathcal{A}} w_k^i \; \delta\left(\mathbf{x}_k - \mathbf{s}_k^i\right) \tag{11}$$

From (11) we see that for any given $\varepsilon \in [0, 1)$ there is an $\mathcal{A}$ such that:

$$p(\mathbf{x}_k \in \mathcal{A}|\mathbf{z_{1:k}}) \geq 1 - \varepsilon \tag{12}$$

Given sufficient samples, the localization uncertainty is thus bounded and we can guarantee that the robot is located within area $\mathcal{A}$ with probability $1 - \varepsilon$.

ORCA as well as NH-ORCA assume disc-shaped robots to make calculations tractable. If a robot radius is inflated by d, the center point of the robot can in turn be translated by a maximum distance of d from its original position while the resulting disc still circumscribes the entire robot. We next derive d such that (12) holds.

We define a subset $\mathbf{S} \subset \{\mathbf{s}^1, \ldots, \mathbf{s}^N\}$ with

$$d_{\mathbf{S}} = \max_{(x,y,\theta) \in \mathbf{S}} \left((x - \mu_x)^2 + (y - \mu_y)^2\right) \tag{13}$$

the maximal distance to the mean. Furthermore, we define:

$$\mathcal{S} : \mathbf{S} \in \mathcal{S} \text{ iff } p(\mathbf{x}_k \in \mathbf{S}|\mathbf{z_{1:k}}) \geq 1 - \varepsilon$$

# of robots	GT	NH-ORCA $\sigma = 0.0m$	NH-ORCA $\sigma = 0.2m$	CALU $\sigma = 0.0m$	CALU $\sigma = 0.2m$
2	0	0	0	0	0
4	0	0	62 (11)	0	0
6	0	7 (4)	85 (11)	0	0
8	0	17 (5)	391 (35)	0	1 (1)
loc. error	–	0.064 $\pm$ 0.009	0.127 $\pm$ 0.028	0.061 $\pm$ 0.011	0.117 $\pm$ 0.043

Table 1: Resulting collisions with various settings summed over 50 runs. The number in brackets shows the number of runs in which the collisions occurred. AMCL is either initialized with a perfect guess or initial guesses sampled from a two dimensional normal distribution ($\sigma_x = \sigma_y = 0.2m$) centered around the ground truth position.

There is a minimal subset $\mathbf{S}^* \in \mathcal{S}$ such that (12) holds and the maximal distance to the mean is minimized:

$$\mathbf{S}^* = \arg\min_{\mathbf{S}\in\mathcal{S}} d_{\mathbf{S}} \tag{14}$$

Thus, if the robot radius is inflated by $d = d_{\mathbf{S}^*}$ the resulting disc circumscribes the entire robot with a probability of $1-\varepsilon$.

The implementation of this computation is straightforward and efficient. An implementation of AMCL as explained in Section 2.3 commonly tracks particles in a k-d tree structure. The algorithm localizes the node closest to the mean μ and subsequently increases the radius d while adding particles that fall into the radius to the set $\mathbf{S}^*$ and accumulating the weight sum until the threshold $1 - \varepsilon$ is reached.

5. EXPERIMENTS AND RESULTS

This section presents experiments and results of the proposed system. We have evaluated our approach in simulation using *Stage* [12] and in a real-world setting.

5.1 Simulation experiments

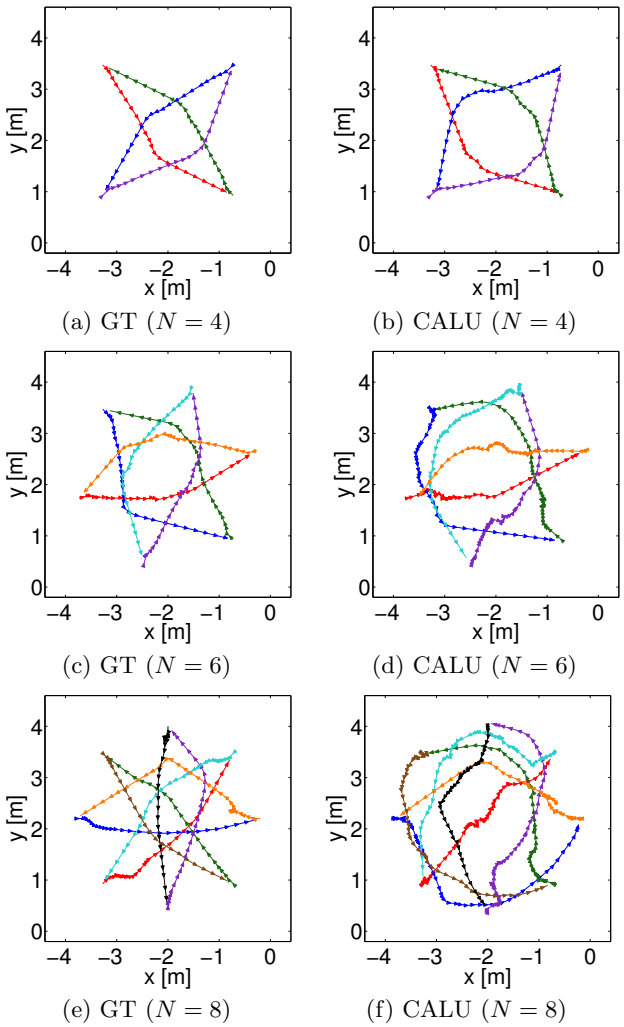
Simulation allows us to investigate the system performance when using localization in comparison to ground truth positioning with perfect information. For evaluation we have chosen seven different scenarios, using two to eight robots. In each setting, the robots where located on a circle (equally spaced) with a radius of 1.8 meter and the goals located on the antipodal positions, i.e. each robot's shortest path is through the center of the circle. The goal is assumed to be reached, when the robots center is within a 0.1 meter radius of the true goal.

Configurations: Each scenario is tested with three different configurations for localization:

Ground Truth (GT): Each robot gets perfect position and velocity information through the simulation environment.

AMCL with $\sigma = 0.0m$: Each robot starts AMCL initialized with the exact pose. The pose cloud is initialized with gaussian noise in x and y direction with $\sigma = 0.0m$.

AMCL with $\sigma = 0.2m$: Each robot starts AMCL initialized with initial guesses sampled from a 2-dimensional normal distribution ($\sigma_x = \sigma_y = 0.2m$) centered around the ground truth position.



(a) GT ($N = 4$) (b) CALU ($N = 4$)
(c) GT ($N = 6$) (d) CALU ($N = 6$)
(e) GT ($N = 8$) (f) CALU ($N = 8$)

Figure 4: Typical robot trajectories observed for different numbers of robots when comparing ground truth (GT) to CALU.

All three settings were tested using NH-ORCA and CALU for collision avoidance. When using ground truth both algorithms are essentially the same leading to a total of five different configurations.

Performance indices: We measure several performance indices: a) number of collisions, b) time to complete run, c) distance travelled, d) localization error and e) jerk cost. The jerk cost measures the smoothness of a path and is defined as:

$$Jerk_{lin} = \frac{1}{2}\int \dddot{\mathbf{x}}(t)\ dt\ , \qquad Jerk_{ang} = \frac{1}{2}\int \dddot{\theta}(t)\ dt\ ,$$

where $\mathbf{x}$ is the two dimensional position vector and θ the robot's heading.

System: Experiments were run on a single machine with a quad core 3.07 GHz Intel i7 processor and 6GB of memory. Each setting was repeated 50 times and results were averaged.

The results of the simulation experiments are summarized in Table 1. We can observe that the number of collisions using the original NH-ORCA rises immensely depending on

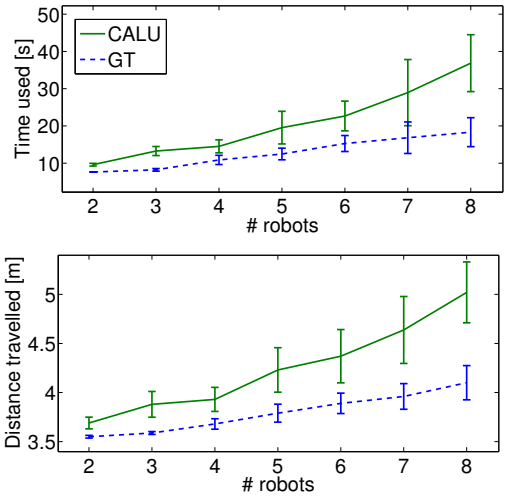


Figure 5: Total time (top) and distance travelled (bottom) metric for CALU and ground truth (GT) averaged over 50 simulations runs.

the localization error and the number of robots. Up to a total of 391 collisions in 35 runs. CALU stayed collision free; except for a single run, in which only one collision occurred. Therefore, for the further discussion NH-ORCA is excluded since it can not be seen as a realistic obstacle avoidance method without our adaptations. This leads us to further compare CALU and GT. To stay as realistic as possible, we focus on runs with CALU and an initial guess corrupted by gaussian noise. In reality, this will also be the case, since it is almost impossible to determine the real position of a robot on a map. Thus, when speaking of CALU in the coming paragraphs, it is referring to CALU with a noisy initial guess.

Some typical trajectories with ground truth (GT) and CALU that we observed during the simulation runs are presented in Figure 4. For up to seven robots the resulting trajectories are usually smooth. In the setting with eight robots, the relatively small area gets very crowded and there is hardly any space to maneuver, see Figure 4(f). Likewise, we can observe that using CALU generally results in larger arcs that are farther away than when using GT. This can be explained by the inflated radius when using CALU due to the sensor uncertainty.

As expected, the runtime and distance travelled increased for more robots as presented in Figure 5. CALU generally uses more time and travels longer than GT. This is also reflected in the average jerk costs, see Figure 6. Interestingly, CALU uses a lot more angular jerk than GT already for two and three robots, while GT increases a lot at first and then stabilizes below the CALU amounts. We assume that the large jerk costs already for only a few robots are due to the inflated radii based on the localization uncertainty. Especially right after initialization, the localization uncertainty is very large, since the particles are scattered more widely thus inflating the radius by a larger factor. This only stabilizes after a couple of update steps using the feedback from the odometry and the laser measurements.

5.2 Real-world experiments

In addition to simulation runs, we have investigated the performance of CALU in real-word settings up to four differential drive Turtlebots[2]. The robots are based on the iRobots Create platform and have a diameter of 33.5 cm. In addition to the usual sensors, they are equipped with a Hokuyo URG laser-range finder to enable better localization in large spaces. All computation is performed on-board on a Intel Atom D525 1.8GHz dual core CPU netbook. Communication between the robots is realized via a 2.4 GHz WiFi link. Before set up the robots are driven remotely to their initial positions and AMCL is initialized with an approximated initial guess.

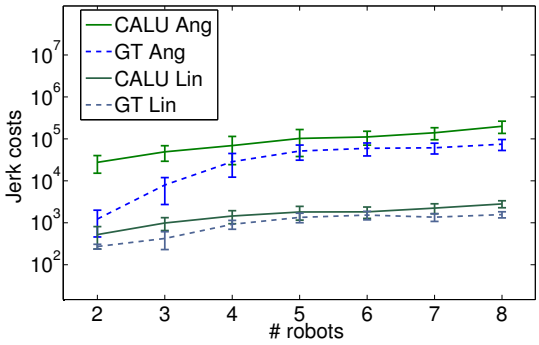


Figure 6: Linear and angular jerk costs for ground truth (GT) and CALU averaged over 50 simulation runs.

Figure 7 shows the trajectories of an example run of the four robots using CALU. The initial positions are approximately 3.5 meters apart; the goal location are set to the diagonally opposing start locations. The system successfully avoids collision and produces smooth paths; except for a small jump in the localization that can be observed in the path of robot starting in the upper right corner.

Additionally, we tested a realistic setting of two robots in a narrow hallway. Each robot wants to get to the other side of the hallway; thus having to pass the other robot. Figure 8 shows the setup and the resulting paths using CALU. To overcome that the robots drive into the walls, two ORCA lines where added to the robots. Adding these additional lines automatically, based on the map and sensor data, is topic of future work as described in the next section. The resulting paths are very close, but still collision free.

6. CONCLUSIONS

While the proposed approach works well in many cases, there are some limitations that we need to address. If the workspace gets more and more crowded with multiple robots, the resulting paths are not always smooth. (NH-)ORCA computes an optimal velocity that is collision free and closest to the desired velocity. However, in our experiments the desired velocity points always straight to the goal. Without a planning algorithm that plans multiple waypoints around fixed obstacles (and motionless robots) there can occur situations where the resulting velocity would be zero.

Differential drive robots can not follow the ORCA velocities directly and always have to turn on the spot or track an arc to accomplish the desired change of direction. In crowded situations, robots do have to turn on the spot in order to avoid collisions, while in open space collision avoid-

[2]For more information see: `http://turtlebot.com`.

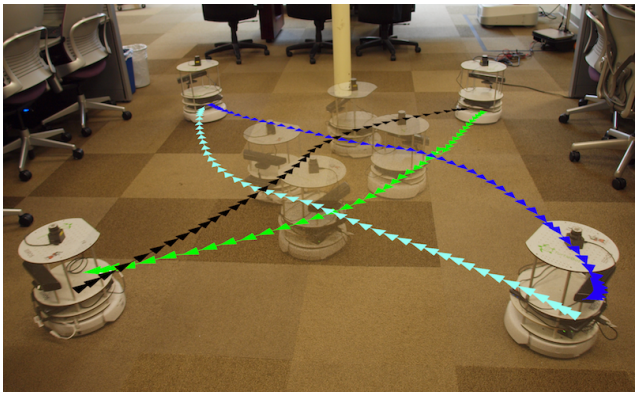


Figure 7: Real-world collision avoidance with four differential drive robots using CALU.

ance between only a few robots smoother arc tracking is desired. This is accomplished by the dynamic adaptation of the error bound depending on the speed.

If the size of the localization error is in the magnitude of the robot radius, collisions are bound to happen and this is shown in the results with NH-ORCA. This is resolved by introducing CALU and adapting the robots radii according to the localization uncertainty reported by AMCL. However, the combination of the dynamically adapting error bound and the inflation due to uncertainty might lead to oscillations, since previously free path become blocked and free again depending on the other agents' radii. To overcome this the radius scaling can be filtered. Additionally, the AMCL localization can jump and combined with delays in communication this can lead to collisions. An Extended Kalman Filter could be a possible solution to this problem.

In future work we will investigate these idea and furthermore extend our experiments to different scenarios, i.e. larger map, various start and goal location configurations and uncontrolled moving obstacles like humans. Additionally, we will examine the possibility of how to implement the presented algorithm as part of a global and local planner that will take the map and static obstacles obtained from the sensor data into account.

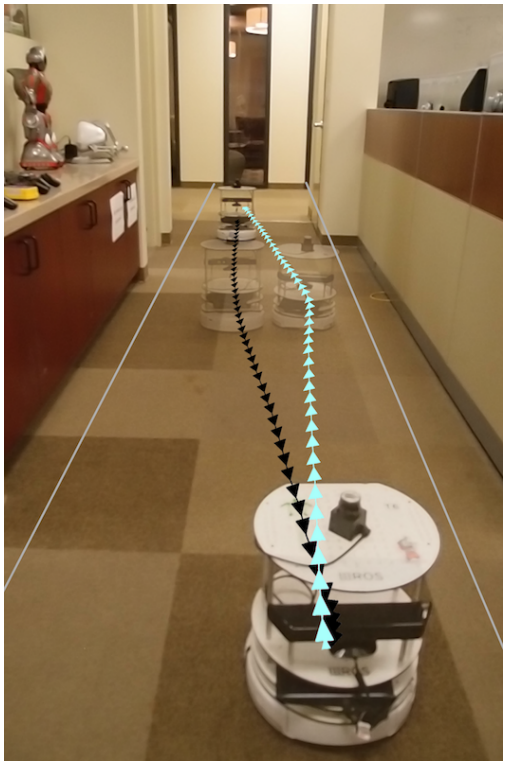


Figure 8: Real-world collision avoidance with two differential drive robots using CALU in a small hallway. To stay clear from the walls two extra ORCA lines where added to both robot.

7. REFERENCES

[1] Javier Alonso-Mora, Andreas Breitenmoser, Martin Rufli, Paul Beardsley, and Roland Siegwart. Optimal reciprocal collision avoidance for multiple non-holonomic robots. In *Proceedings of the 10th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, 2010.

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

[3] Dieter Fox. Kld-sampling: Adaptive particle filters. In *Advances in Neural Information Processing Systems 14*. MIT Press, 2001.

[4] Dieter Fox. Adapting the sample size in particle filters through kld-sampling. *International Journal of Robotics Research*, 22, 2003.

[5] Boris Kluge, Dirk Bank, Erwin Prassler, and Matthias Strobel. Coordinating the motion of a human and a robot in a crowded, natural environment. In *Advances in Human-Robot Interaction*, volume 14 of *Springer Tracts in Advanced Robotics*, pages 231–234. Springer Berlin / Heidelberg, 2005.

[6] Morgan Quigley et al. ROS: An open-source Robot Operating System. In *Proceedings of the Open-Source Software workshop (ICRA)*, 2009.

[7] Jamie Snape, Jur van den Berg, Stephen J. Guy, and Dinesh Manocha. Independent navigation of multiple mobile robots with hybrid reciprocal velocity obstacles. In *Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5917–5922, 2009.

[8] Jamie Snape, Jur van den Berg, Stephen J. Guy, and Dinesh Manocha. Smooth and collision-free navigation for multiple robots under differential-drive constraints. In *Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[9] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series)*. The MIT Press, 2005.

[10] Jur van den Berg, Stephen Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics Research*, volume 70, pages 3–19, 2011.

[11] Jur van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *ICRA 2008*, pages 1928 –1935, 2008.

[12] Richard Vaughan. Massively multi-robot simulation in stage. *Swarm Intelligence*, 2(2):189–208, 2008.