Collision Avoidance under Bounded Localization Uncertainty

Daniel Claes, Daniel Hennes, Karl Tuyls and Wim Meeussen

***Abstract*— We present a multi-mobile robot collision avoidance system based on the velocity obstacle paradigm. Current positions and velocities of surrounding robots are translated to an efficient geometric representation to determine safe motions. Each robot uses on-board localization and local communication to build the velocity obstacle representation of its surrounding. Our close and error-bounded convex approximation of the localization density distribution results in collision-free paths under uncertainty. While in many algorithms the robots are approximated by circumscribed radii, we use the convex hull to minimize the overestimation in the footprint. Results show that our approach allows for safe navigation even in densely packed environments.**

I. Introduction

In this paper we address the problem of real-time local collision avoidance in multi-mobile robot systems. 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. Static obstacles can be avoided using traditional planning algorithms whereas dynamic obstacles pose a tough challenge. The naive approach is to observe consecutive obstacle positions in order to extrapolate the future trajectory. The velocity obstacle (VO) [1] is a geometric representation of all velocities that will eventually result in a collision given the dynamic obstacle maintains the observed velocity.

In a multi-mobile robot scenario, robots can not merely be regarded as dynamic obstacles. Each robot is a pro-active agent, taking actions to avoid collisions. Neglecting this fact might lead to oscillations and thus highly inefficient trajectories or even collisions. Van den Berg et al. [2] introduced the reciprocal velocity obstacle to address reciprocity: for each pair of robots, one takes half of the responsibility to avoid the other. RVO has since been extended to the hybrid reciprocal velocity obstacle (HRVO) [3], [4] to overcome ”reciprocal dances” that occur when robots can not reach independent agreement on which side to pass each other.

The aforementioned approaches avoid local sensing methods by using global positioning via an overhead tracking camera or assume perfect sensing (i.e. purely simulation based). This limits the possibilities for application greatly. *Collision avoidance under localization uncertainty* (CALU) [5] combines the velocity obstacle approach with on-board localization. CALU provides a solution that is situated in-between centralized motion planning and communication-free individual navigation. While actions are 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. CALU uses non-holonomic optimal reciprocal collision avoidance (NH-ORCA) [6] to compute collision free velocities for disc-shaped robots with kinematic constraints. Uncertainty in localization is addressed by inflating the robots' radii according to the particle distribution of adaptive monte carlo localization (AMCL) [7].

While CALU effectively alleviates the need for global positioning by using decentralized localization, some problems prevail. Suboptimal behavior is encountered when (a) the footprint of the robot is not efficiently approximated by a disk; and (b) the pose belief distribution of AMCL is not circular but elongated along one axis (typically observed in long hallways). In both situations, the resulting VOs vastly overestimate the unsafe velocity regions. Hence, this conservative approximation might lead to a suboptimal - or no solution at all.

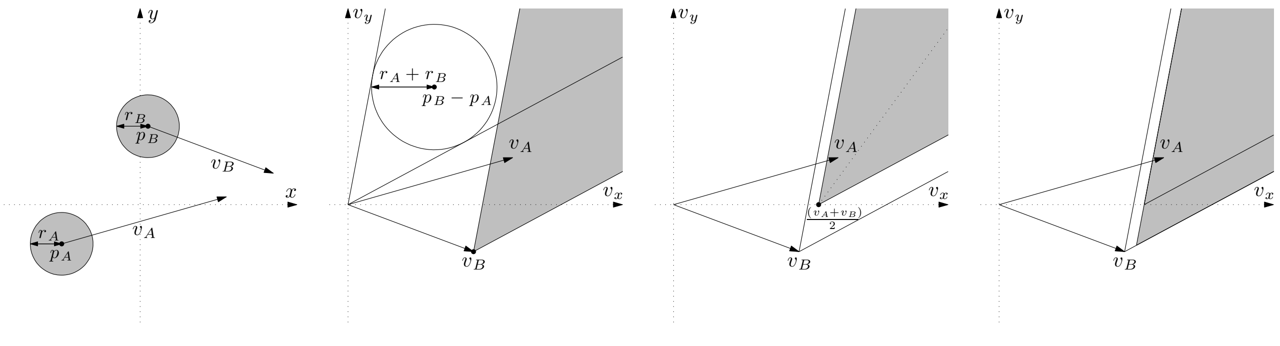
In this paper, we introduce *convex outline collision avoidance under localization uncertainty* (COCALU) as an extension of CALU to address these shortcomings. COCALU uses the same approach based on decentralized computation, on-board localization and local communication to share relevant shape, position and velocity data between robots. This data is used to build the velocity obstacle representation using HRVOs for convex footprints in combination with a close and error-bounded convex approximation of the localization density distribution. Instead of NH-ORCA, ClearPath [8] is employed to efficiently compute new collision-free velocities in the closed-loop controller.

The remainder of the paper is structured as follows. Section II provides background information about VOs, ClearPath and AMCL. Section III discusses key challenges in applying velocity-based collision avoidance to real-world robotic scenarios and illustrates the CALU approach. In Section IV, we introduce the new algorithm COCALU, using a close and error-bounded convex approximation of the localization density distribution. Experimental results are presented in Section V. The paper concludes with a brief discussion in Section VI.

II. Background

In this section, we will concisely describe the underlying principles of velocity obstacles, ClearPath and adaptive monte carlo localization.

D. Claes, D. Hennes and K. Tuyls are with the Department of Knowledge Engineering, Maastricht University, 6200MD Maastricht, The Netherlands. W. Meeussen is with Willow Garage Inc., Menlo Park, CA, USA. danielclaes@me.com, daniel.hennes@gmail.com, k.tuyls@maastrichtuniversity.nl, meeussen@willowgarage.com



(a) Workspace configuration

(b) VO

(c) RVO

(d) HRVO

Fig. 1. Creating the different velocity obstacles out of a workspace configuration. (a) A workspace configuration with two robots R_A and R_B. (b) Translating the situation into the velocity space and the resulting velocity obstacle (VO) for R_A. (c) Translating the VO by $\frac{v_A+v_B}{2}$ results in the reciprocal velocity obstacle (RVO), i.e. each robot has to take care of half of the collision avoidance. (d) Translating the apex of the RVO to the intersection of the closest leg of the RVO to the own velocity, and the leg of the VO that corresponds to the leg that is furthest away from the own velocity. This encourages passing the robot on a preferred side, i.e. in this example passing on the left. The resulting cone is the hybrid velocity obstacle (HRVO).

A. Velocity Obstacles

The velocity obstacle (VO) was first introduced by Fiorini et al. [1] for local collision avoidance and navigation in dynamic environments with multiple moving objects. The subsequent definition of the VO assumes planar motions, though the concept extends to 3-D motions in a straight forward manner.

Let us assume a workspace configuration with two robots on a collision course as shown in Figure 1(a). If the position and speed of the moving object (robot R_B) is known to R_A, we can mark a region in the robot's velocity space which leads to collision under current velocities and is thus unsafe. This region resembles a cone with the apex at R_B's velocity v_B, and two rays that are tangential to the convex hull of the Minkowski sum of the footprints of the two robots. The Minkowski sum for two sets of points A and B is defined as:

$$A \oplus B = \{a + b | a \in A, b \in B\} \tag{1}$$

For the remainder of the paper, we define the $\oplus$ operator to denote the convex hull of the Minkowski sum such that $A \oplus B$ results in the points on the convex hull of the Minkowski sum of A and B.

The direction of the left and right ray is then defined as:

$$\theta_{left} = \max_{p_i \in \mathcal{F}_A \oplus \mathcal{F}_B} atan2((p_{rel} + p_i)^{\perp} \cdot p_{rel}, (p_{rel} + p_i) \cdot p_{rel})$$

$$\theta_{right} = \min_{p_i \in \mathcal{F}_A \oplus \mathcal{F}_B} atan2((p_{rel} + p_i)^{\perp} \cdot p_{rel}, (p_{rel} + p_i) \cdot p_{rel})$$

where p_{rel} is the relative position of the two robots and $\mathcal{F}_A \oplus \mathcal{F}_B$ is the convex hull of the Minkowski sum of the footprints of the two robots. The $atan2$ expression computes the signed angle between two vectors. The resulting angles θ_{left} and θ_{right} are left and right of p_{rel}. If the robots are disc-shaped, the rays are the tangents to the disc with the radius $r_A + r_B$ at center p_{rel} as shown in Figure 1(b). The angle can then be calculated as:

$$\theta_{left} = -\theta_{right} = arcsin(\frac{r_A + r_B}{|p_{rel}|}) \tag{2}$$

In the example in Figure 1, robot R_A's velocity vector v_A points into the VO, thus we know that R_A and R_B 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 when the robots are on collision course. Since all robots select a new velocity outside of all velocity obstacles independently, at the next time step, the old velocities pointing towards the goal will become available again. Hence, all robots select their old velocities, which will be on collision course again after the next time step.

To overcome these oscillations, the reciprocal velocity obstacle (RVO) was introduced by Berg et al. [2]. The surrounding moving obstacles are in fact also pro-active agents and thus aim to avoid collisions too. Assuming that each robot takes care of half of the collision avoidance, the apex of the VO can be translated to $\frac{v_A+v_B}{2}$. Furthermore, this leads to the property, that if every robot chooses a velocity outside of the RVO closest to the current velocity, the robots will pass on the same side. However, each robot optimizes its commanded velocity in respect to a preferred velocity in order to make progress towards its goal location. This can lead to reciprocal dances, i.e. that both robots first try to avoid to the same side and then to the other side. In a situation with perfect symmetry and sensing, this behavior continues infinitely.

To counter these situations, the hybrid velocity obstacle (HRVO) was introduced by Snape et al. [3]. Figure 1(d) shows the construction of the HRVO. To encourage the selection of a velocity towards the preferred side, e.g. left in this example, the other leg of the RVO is substituted with the corresponding leg of the VO. The new apex is the intersection of the line of the one leg from RVO and the line of the other leg from the VO. This reduces the chance of selecting a velocity on the ”wrong” side of the velocity obstacle and thus the chance of a reciprocal dance, while not overconstraining the velocity space. The robot might still try to pass on the ”wrong” side, e.g. another robot induces a

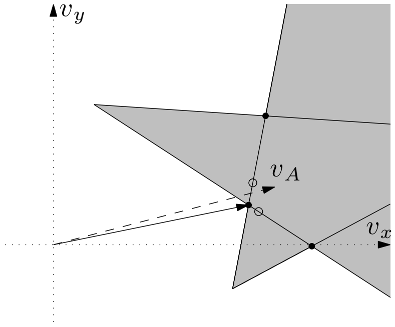


Fig. 2. ClearPath enumerates intersection points for all pairs of VOs (solid dots). In addition the preferred velocity v_A is projected on the closest leg of each VO (open dots). The point closest to the preferred velocity (dashed line) and outside of all VOs is selected as new velocity (solid line).

HRVO that blocks the whole side, but then soon all other robots will adapt to the new side, too.

B. ClearPath

To efficiently compute collision free velocities, we employ the ClearPath algorithm introduced by Guy et al. [8]. The algorithm is applicable to many variations of velocity obstacles (VO, RVO or HRVO) represented by line segments or rays. ClearPath follows the general idea that the collision free velocity that is closest to preferred velocity is: (a) on the intersection of two line segments of any two velocity obstacle, or (b) the projection of the preferred velocity onto the closest leg of each velocity obstacle. All points that are within another obstacle are discarded and from the remaining set the one closest to the preferred velocity is selected. Figure 2 shows the graphical interpretation of the algorithm.

C. Kinematic and dynamic constraints

In order to execute the computed collision free velocity, the robot has to be able to instantaneously accelerate to any velocity in the two dimensional velocity space. This implies that the velocity obstacle approach requires a fully actuated holonomic platform, able to accelerate into any direction from any state. However, differential drive robots with only two motorized wheels are much more common due to their lower price point. Additionally, robots can only accelerate and decelerate within certain dynamic constraints.

If the acceleration limits and motion model of the robot is known, a region of admissible velocities can be calculate and approximated by a convex polygon. These constraints can easily be added to the ClearPath formulation to be incorporated in the calculation of the new velocity. For differential drive robots, there are several possibilities, e.g. using the effective center of a robot (Kluge et al. [9]) or using a holonomic tracking error (Alonso-Mora et al. [6]).

Any robot can track a holonomic speed vector with a certain tracking error ε. This error depends on the direction and magnitude 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. Additionally, in dense configurations 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).

D. 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 [7].

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.

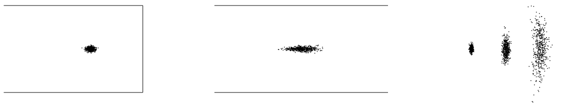
The first iterative step is the prediction phase, in which the particles of the previous population are moved based on the motion model of the robot, i.e. the odometry. Afterwards, in the update phase, the particles are weighted according to the likelihood of the robot's measurement for each particle. 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.

1) 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:

$$\begin{bmatrix} \hat{x}_{k+1} \\ \hat{y}_{k+1} \\ \hat{\theta}_{k+1} \end{bmatrix} = \begin{bmatrix} \hat{x}_k + \rho cos(\hat{\theta}_k + \Delta\theta) \\ \hat{y}_k + \rho sin(\hat{\theta}_k + \Delta\theta) \\ \hat{\theta}_k + \Delta\theta \end{bmatrix} \quad (3)$$

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 [10].

2) 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



(a) End of corridor (b) Long corridor (c) Open space

Fig. 3. Typical particle filter situations. (a) A well localized robot at the end of a corridor resulting in a particle cloud with small variance. (b) In an open ended hallway the sensor only provides valid readings to the sides, resulting in an particle cloud elongated in the direction of the corridor. (c) In an open space no sensor readings result in a particle cloud driven purely by the motion model.

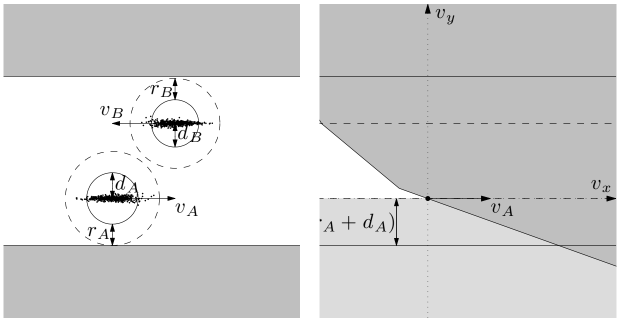


Fig. 4. The corridor problem: Approximating the localization uncertainty (and the footprint) with circumscribed circles, vastly overestimates the true sizes, such that the robots do not fit next to each other. Thus, the HRVO together with the VO of the walls invalidates all forward movements.

the probability of the actual sensor measurement (z_k) given the particles position (s_k^i) at time k as shown below:

$$w_{k+1}^i = p(z_k | s_k^i) \tag{4}$$

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

$$w_k^i = \frac{w_k^i}{\sum_i w_k^i} \tag{5}$$

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. For further details we refer to [11].

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.

However, a common problem occurs if the environment looks very similar along the trajectory of the robot, e.g. a long hallway; or a big open space with only very few valid sensor readings. In these cases, particles are mainly updated and resampled according to the motion model leading to the situations shown in Figure 3.

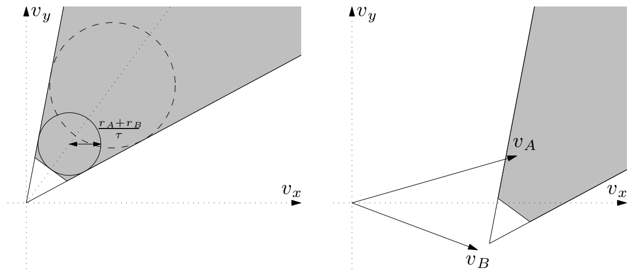
III. Collision Avoidance under Localization Uncertainty

As mentioned in the introduction, Collision avoidance under localization uncertainty (CALU) [5] combines the velocity obstacle approach with on-board localization using AMCL. The approach of CALU is to calculate the actions for each robot independently, while 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 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. Thus, the method is situated in-between centralized motion planning for multi robot systems and communication-free individual navigation.

CALU uses non-holonomic optimal reciprocal collision avoidance (NH-ORCA) [6], [12] to compute collision free velocities for disc-shaped robots with kinematic constraints. Instead of velocity obstacles, agents independently compute half-planes of collision-free velocities for each other agent. 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. The remaining collision free region is a convex polygon, where linear programming can be used to calculate the optimal new velocity.

As described before, some problems arise. Creating the half planes almost always overestimates the size of the VO, and therefore, overconstrains the velocity-space. In a densely packed configuration with many robots, this might invalidate the whole possible velocity space. Additionally, suboptimal behavior is encountered if the footprint of the robots are vastly overestimated by the circumscribed radius, e.g. in the case of a rectangular robot. Furthermore, if the AMCL pose distribution is not shaped circular, but elongated around one axis, or solely based on the motion model of the robot as presented in Figure 3, the approximation with a circle is problematic. Figure 4 shows an example configuration in which CALU would not be able to move forward, since the two enlarged disks do not fit anymore next to each other in the corridor. In this case, it is due to the elongated position uncertainty, which is largely overestimated by the circle. The same problem would occur when having a square or even rectangular robots.

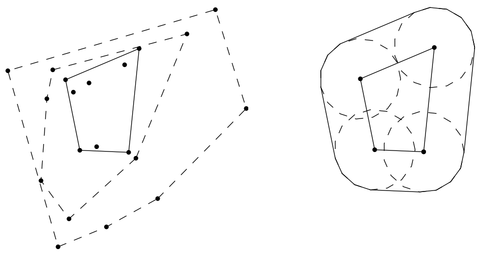
Other solutions use a Kalman Filter to track the robots' positions [3]. The covariances of the bivariate Gaussian distribution are used to increase the combined radii and to enlarge the HRVO. However, this approach has the same shortcomings as the approach proposed in CALU.



(a) Truncated VO

(b) Truncated HRVO

Fig. 5. (a) Truncation of a VO of a static obstacle at $\tau = 2$ and approximating the truncation by a line. (b) Translating the truncated cone according to the HRVO method to get a tHRVO.



(a) Convex hull peeling

(b) Minkowski sum

Fig. 6. (a) Three iterations of convex hull peeling. (b) Minkowski sum of the resulting convex polygon and a circular footprint.

IV. Convex Outline Collision Avoidance under Localization Uncertainty

The key difference between CALU and COCALU is to use the shape of the particle cloud instead of using a circumscribed circle. The corridor example, as presented in Figure 4, shows the shortcomings of the previous approach. In this approach, we approximate the shape of the particle filter by a convex hull. However, using the convex hull of all particles can results in large overestimations, since outliers in the particles' positions inflate the resulting convex hull immensely. As a solution to this problem, we use *convex hull peeling*, which is also known as *onion peeling* [13], in combination with an error bound ε.

A further difference between the two algorithms is that COCALU uses ClearPath as described in Section II-B to calculate the new velocity. Additionally, we introduce truncated HRVOs to overcome the problems that can occur with NH-ORCA in densely packed configurations.

A. Truncated hybrid velocity obstacle

When the workspace is cluttered with many robots that do not move or only slowly, the apices of the HRVOs are close to the origin in velocity space; thus rendering robots immobile. This problem can be solved using truncation. The idea of a truncated hybrid velocity obstacle (tHRVO) can be best explained by imagining a static obstacle. A velocity in the direction of the obstacle will eventually lead into collision, but not directly. Hence, we can define an area in which the selected velocities are safe for at least τ time steps. The truncation is then in the shape of the Minkowski sum of the two footprints, shrunk by the factor τ. If the footprints are discs, the shrunken disc that still fits in the truncated cone has a radius of $\frac{r_A+r_B}{\tau}$, see Figure 5(a). The truncation can be closely approximated by a line perpendicular to the relative position and tangential to the shrunken disk. Applying the same method to create a HRVO from a VO, we can create a tHRVO out of the truncated VO by translating the apex accordingly (Figure 5(b)).

B. Convex hull peeling with an error bound

The idea behind the onion peeling is to create layers of convex hulls. This can be intuitively explained by removing the points on the outer convex hull, and to calculate a new convex hull of the remaining points. This process can be repeated iteratively until the remaining points are less than two. Figure 6(a) shows three iterations of the method on an example point cloud.

COCALU finds the convex hull layer in which the probability of the robot being located in is greater than $1-\varepsilon$. To derive this bound, we revisit the particle filter described in Section II-D.

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:

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

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:

$$\mu_x = \sum_i w_k^i \; \hat{x}_k^i \qquad (7)$$

$$\mu_y = \sum_i w_k^i \; \hat{y}_k^i \qquad (8)$$

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

The mean gives the current position estimate of the robot. The probability of the robot actually residing within a certain area $\mathcal{A}$ at time k is:

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

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

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

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

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

Given sufficient samples, the localization uncertainty is thus bounded and we can guarantee that the robot is located

Algorithm 1 COCALU

```
Input: $(\mathcal{F}, p, v)$ : Robot footprint, position and velocity,
    $(s^i, w^i) \in \mathcal{P} = \mathcal{S} \times \mathcal{W}$ : AMCL weighted particle set,
    $(\mathcal{F}_j, p_j, v_j) \in \mathcal{A}$: List of neighboring Agents,
    $\varepsilon$ : error bound, $v^{pref}$ : preferred Velocity,
    $\tau$ : truncation timesteps

  $bound \leftarrow 0$
  while $bound \leq \varepsilon$ do
    Create convex hull $\mathcal{C}$ of $\mathcal{S}$
    $bound \leftarrow bound + \sum_{\forall i: s^i \in \mathcal{C}} w_i$
    $\mathcal{P} \leftarrow \mathcal{P} \setminus \{(s^i, w^i) \in \mathcal{P} | s^i \in \mathcal{C}\}$
  end while
  $\mathcal{M} \leftarrow \mathcal{F} \oplus \mathcal{C}$
  for all $(\mathcal{F}_j, p_j, v_j) = A_j \in \mathcal{A}$ do
    $\mathcal{M}_{A_j} \leftarrow \mathcal{F}_j \oplus \mathcal{M}$
    Construct $VO_{A_j}$ from $\mathcal{M}_{A_j}$ at $p_j - p$
    Construct $HRVO_{A_j}$ from $VO_{A_j}$ with $v_j$ and $v$
    Construct $tHRVO_{A_j}$ from $HRVO_{A_j}$ with $\tau$
  end for
  Use clearpath to calculate new velocity $v^{new}$ from $v^{pref}$
  and all $tHRVO_{A_j}$
```

within area $\mathcal{A}$ with probability $1 - \varepsilon$. Thus, the weights of excluded samples sum up to at most ε.

In order to find this specific convex hull enclosing area $\mathcal{A}$, we propose an iterative process as described in the first part in Algorithm 1. As long as the sum of the weights of the removed samples does not exceed the error bound, we create the convex hull of all (remaining) particle samples. Afterwards, we sum up all the weights of the particles located on the convex hull and add this weight to the previously computed sum. If the total sum does not exceed the error bound, all the particles that define the current convex hull will be removed from the particle set and the process is repeated.

When the convex hull is found, we calculate the Minkowski sum of the robot's footprint and the convex hull. The convex hull of the Minkowski sum is then used as new footprint of the robot as shown in Figure 6(b).

C. Complexity

The first part of COCALU (see Algorithm 1) computes the convex hull according to the bound ϵ. One iteration of the convex hull can be computed in $\mathcal{O}(n \log h)$ [14], where n is the number of particles and h is the number of points on the hull (our experiments show $h \leq 50$ for $200 \leq n \leq 5000$ particles). The worst case (bound reached in the last iteration) results in complete convex hull peeling, which can be achieved in $\mathcal{O}(n \log n)$ [13].

The convex hull of the Minkowski sum of two convex polygons (operator $\oplus$) can be computed in $\mathcal{O}(l + k)$, where l and k are the number of vertices (edges). If the input vertices lists are in order, the edges are sorted by the angle to the x-axis and simply merging the lists results in the convex hull.

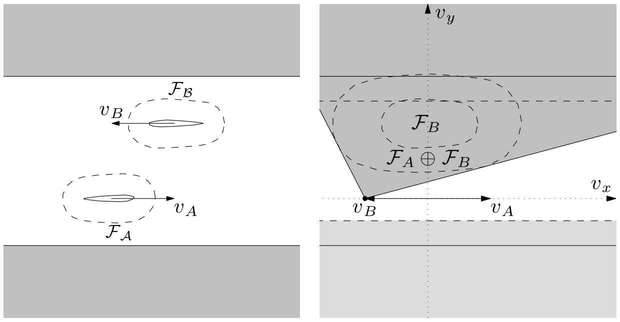


Fig. 7. Using COCALU solves the corridor problem. Since the robots' footprints and localization uncertainty are approximated with less overestimation, the robots can pass along the corridor without a problem.

ClearPath runs in $\mathcal{O}(N(N+M))$, where N is the number of neighboring robots and M the number of total intersection segments [8].

In summary, using convex hull peeling for approximating localization uncertainty and convex footprints solves the corridor problem. Comparing Figure 4 and 7 shows the differences when using CALU and COCALU. In the latter figure, it can be seen that the robots can easily pass each other even without adapting their path.

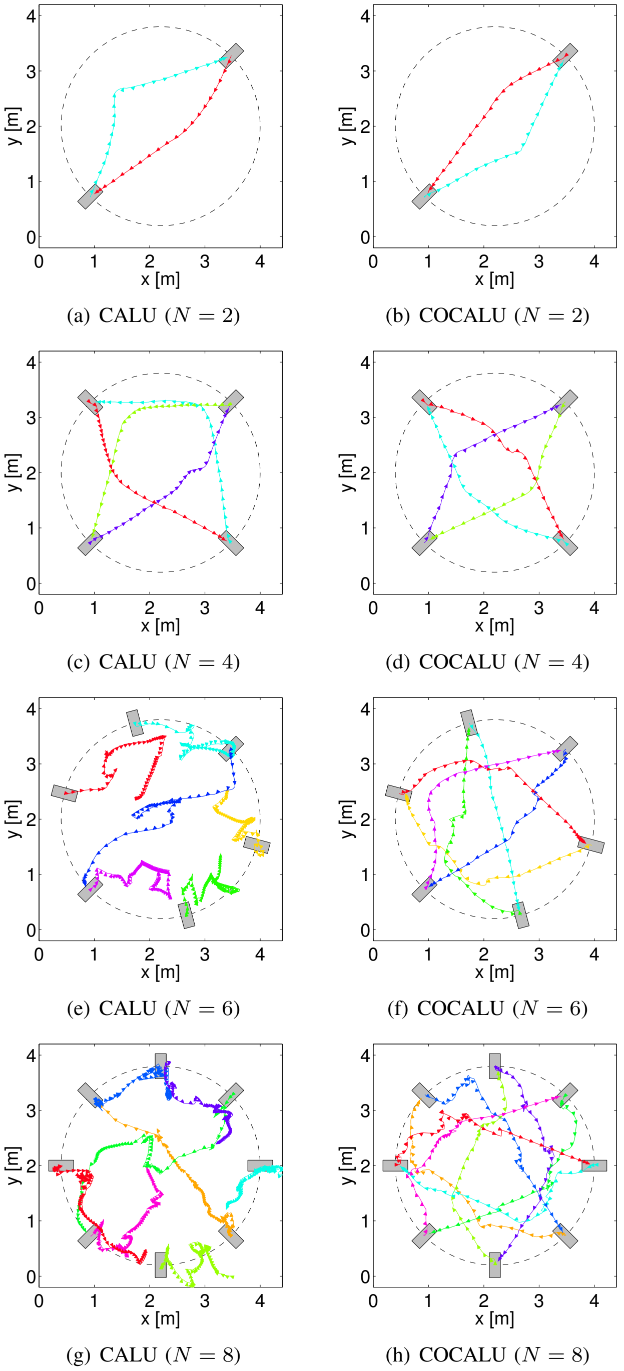
V. Experiments and results

This section presents experiments and results of the proposed system. We have evaluated our approach in simulation using *Stage* [15] and in a real-world setting.

A. Simulation experiments

Simulation allows us to investigate the system performance when using localization in various extreme settings. For evaluation we have chosen four different scenarios, using two to eight rectangular shaped robots with a length of 45cm and a width of 20cm. In each setting the robots were 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. 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. All three settings were tested using CALU and COCALU for collision avoidance. Experiments in simulation were run on a single machine with a quad core 2.8 GHz Intel i7 processor and 4GB of memory.

Some typical trajectories when comparing CALU and COCALU are presented in Figure 8. The problem of approximating the footprint and localization error by circumscribed radii become visible when using more than four robots. Already with two and four robots, we can see that the trails of CALU are much further apart than using COCALU. With six robots and using CALU, only one robot reaches the goal, after some maneuvering. Using COCALU, some small jumps can be seen in the trajectories, but the paths are still smooth. In the last setting with eight robots, the approximation using



(a) CALU ($N = 2$)

(b) COCALU ($N = 2$)

(c) CALU ($N = 4$)

(d) COCALU ($N = 4$)

(e) CALU ($N = 6$)

(f) COCALU ($N = 6$)

(g) CALU ($N = 8$)

(h) COCALU ($N = 8$)

Fig. 8. Typical robot trajectories observed for different numbers of stick-shaped robots when comparing CALU to COCALU. The starting positions are shown in grey. The goal positions are located on antipodal side of the circle.

CALU clutters the workspace so much that most robots are not able to reach their goal locations. However using COCALU, the robots are still moving along rather smooth paths towards the goals.

The small jumps in the trails of COCALU are probably due to the rotational sensitivity of the stick shaped robots. A small rotation in the robot's orientation results already in a large change in the Minkowski sum of the footprint. Hence, the robots need to react quickly in order to be on a collision free path again.

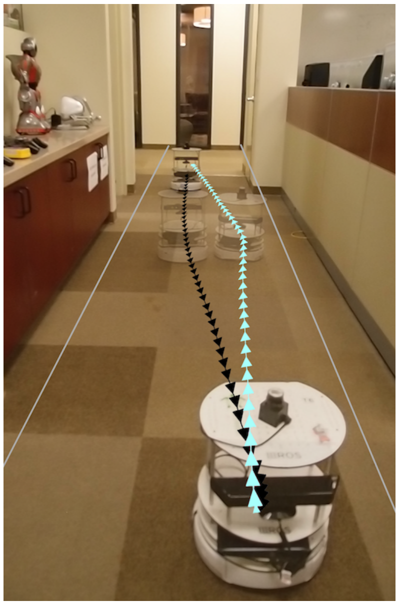


Fig. 9. Real-world collision avoidance with two differential drive robots using COCALU in a small hallway. To stay clear from the walls two extra constraints where added to both robot.

B. *Real-world experiment*

In addition to simulation runs, we have investigated the performance of COCALU in real-word settings using differential drive Turtlebots[1]. 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 i3 380UM 1.3GHz dual core CPU notebook. 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.

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 9 shows the setup and the resulting paths using COCALU. To overcome that the robots drive into the walls, two additional constraints were added by calculating the closest obstacle point and adding a line at the truncated VO with $\tau = 1$. The resulting paths are very close, but still collision free.

VI. Conclusions

While previous approaches as HRVO and CALU work well in many situations, both have various limitations. In HRVO an overhead camera is used for tracking the robots; CALU combines AMCL and NH-ORCA to accomplish localization on a per-agent level. However, both approaches vastly overestimate the localization uncertainty and robots' footprints by using circumscribed circles. In free space, this

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

is usually not a problem, since the robots just avoid each other further apart, but for instance in close corridors a better approximation is necessary. COCALU introduces a method to bound the localization error in a similar way as CALU does, but using convex hull peeling, the resulting region is a lot smaller. Additionally, convex polygons are used as robot footprints, thus the Minkowski sum of the localization uncertainty area and the footprint results in a much better approximation of the real state of the situation without overly constraining the velocity space. Furthermore in highly packed situations, using truncation allows the robots to use velocities that are collision free for a number of time steps but will result in a collision eventually.

The proposed approach works well in many cases in simulation and real life. However, there are some cases that we need to address. If the workspace gets more and more crowded with multiple robots, the resulting paths are not always smooth. COCALU 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.

Additionally, COCALU is very sensitive to the rotation of the robots. When two rectangular shaped robots are first parallel to each other and then rotate in opposite directions, the Minkowski sum increases by a large amount. To overcome this, the changes can be filtered to smooth the transitions.

In future work, we will extend our experiments to different scenarios, i.e. larger map, various start and goal location configurations and uncontrolled moving obstacles like humans. Furthermore, more test cases involving real robots will be investigated.

References

[1] P. Fiorini and Z. Shiller, “Motion planning in dynamic environments using velocity obstacles,” *International Journal of Robotics Research*, vol. 17, pp. 760–772, July 1998.

[2] J. van den Berg, M. Lin, and D. Manocha, “Reciprocal velocity obstacles for real-time multi-agent navigation,” in *ICRA 2008*, 2008, pp. 1928 –1935.

[3] J. Snape, J. van den Berg, S. J. Guy, and D. 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*, 2009, pp. 5917–5922.

[4] J. Snape, J. P. van den Berg, S. J. Guy, and D. Manocha, “The hybrid reciprocal velocity obstacle,” *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 696–706, 2011.

[5] D. Hennes, D. Claes, K. Tuyls, and W. Meeussen, “Multi-robot collision avoidance with localization uncertainty,” in *Proceedings of the Intl. Conf. on Autonomous Agents and Multiagent Systems*, Valencia, Spain, June 2012.

[6] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P. Beardsley, and R. Siegwart, “Optimal reciprocal collision avoidance for multiple non-holonomic robots,” in *Proceedings of the 10th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, 2010.

[7] D. Fox, “Kld-sampling: Adaptive particle filters,” in *Advances in Neural Information Processing Systems 14*. MIT Press, 2001.

[8] S. J. Guy, J. Chhugani, C. Kim, N. Satish, M. C. Lin, D. Manocha, and P. Dubey, “Clearpath: Highly parallel collision avoidance for multi-agent simulation,” in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*. ACM, 2009, pp. 177–187.

[9] B. Kluge, D. Bank, E. Prassler, and M. Strobel, “Coordinating the motion of a human and a robot in a crowded, natural environment,” in *Advances in Human-Robot Interaction*, ser. Springer Tracts in Advanced Robotics. Springer Berlin / Heidelberg, 2005, vol. 14, pp. 231–234.

[10] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series)*. The MIT Press, 2005.

[11] D. Fox, “Adapting the sample size in particle filters through kld-sampling,” *International Journal of Robotics Research*, vol. 22, 2003.

[12] J. van den Berg, S. Guy, M. Lin, and D. Manocha, “Reciprocal n-body collision avoidance,” in *Robotics Research*, vol. 70, 2011, pp. 3–19.

[13] B. Chazelle, “On the convex layers of a planar set,” *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 509–517, 1985.

[14] T. M. Chan, “Optimal output-sensitive convex hull algorithms in two and three dimensions,” *Discrete & Computational Geometry*, vol. 16, pp. 361–368, 1996.

[15] R. Vaughan, “Massively multi-robot simulation in stage,” *Swarm Intelligence*, vol. 2, no. 2, pp. 189–208, 2008.