Autonomous Robots (2018) 42:1749–1770
https://doi.org/10.1007/s10514-018-9726-5

Multi robot collision avoidance in a shared workspace

Daniel Claes[1] · **Karl Tuyls**[1]

Received: 15 February 2017 / Accepted: 29 March 2018 / Published online: 13 April 2018

Abstract
This paper presents a decentralised human-aware navigation algorithm for shared human–robot work-spaces based on the velocity obstacles paradigm. By extending our previous work on collision avoidance, we are able to include and avoid static and dynamic obstacles, no matter whether they are induced by other robots and humans passing through. Using various cost maps and Monte Carlo sampling with different cost factors accounting for humans and robots, the approach allows human workers to use the same navigation space as robots. It does not rely on any external positioning sensors and shows its feasibility even in densely packed environments.

Keywords Collision avoidance · Shared-workspace · Velocity obstacles

1 Introduction

Current research in mobile robotics focuses more and more on enabling robots and humans to share a common workspace. A well known research initiative in this direction is the *Factory of the Future*, which has the goal to develop smart factories with networked tools, devices, and mobile manipulation platforms (e.g. the KUKA youBot). It is also known as *Industry 4.0*, which was coined at the Hanover Fair in 2011 (Kagermann et al. 2011).

Nowadays, robots in manufacturing are typically not designed to be mobile and human-safe. They are placed inside cages and operation is interrupted as soon as a human enters the safety zones. Current solutions for mobile robots in manufacturing settings are restricted to predefined paths, e.g., tracks on the floor, or restricted to movement in a grid to ensure easy navigation. Humans are not allowed to enter the navigation zone of the robots in order to ensure safety.

Relying on predefined paths and grids for navigation is too restrictive and does not allow for a flexible and generally applicable setup of a mobile multi-robot system. Ideally, robots should be able to plan their paths through any open space and ensure safety without any external limitations such as restricted zones. Additionally, in an unstructured workspace there are no traffic rules that direct the navigation. To safely navigate in such a shared multi-robot and human setting, the robot system has to take into account that the surrounding moving 'obstacles' are essentially pro-active agents and thus might aim to avoid collisions.

Although robot localisation is a requirement for multi-robot collision avoidance, most approaches assume perfect sensing and positioning and avoid local methods by using global positioning via an overhead tracking camera (Alonso-Mora et al. 2015a) - or are purely simulation based (van den Berg et al. 2011). Nevertheless, to be able to correctly perform local collision avoidance in a realistic environment, a robot needs a reliable position estimation of itself and the other agents and humans without the help of external tools. Additionally, multi-robot systems in a real-world environment need methods to deal with the uncertainty in their own positions, and the positions and possible actions of the other agents.

1.1 Contributions

In this paper, we add the following three novelties to this field: First, we show a reliable estimation of the localisation uncer-

This is one of several papers published in *Autonomous Robots* comprising the "Special Issue on Distributed Robotics: From Fundamentals to Applications".

✉ Daniel Claes
daniel.claes@liverpool.ac.uk

Karl Tuyls
k.tuyls@liverpool.ac.uk

[1] Department of Computer Sciences, University of Liverpool, Ashton Building, Ashton Street, Liverpool L69 3BX, UK

tainty using the adaptive Monte Carlo localisation (AMCL), then we combine this with a sampling-based approach to incorporate human avoidance and lastly, by incorporating the commonly used Dynamic Window Approach (DWA) (Fox et al. 1997) with a global path planner, allows us to handle complex environments with multiple dynamic, i.e. humans and robots, and static obstacles.

In more detail, we show how the distribution of the particle cloud when using AMCL can be used as an estimator for the localisation uncertainty. This estimator can be used to enlarge the robots' footprints to ensure safe navigation within the neighbourhood of other robots. The robots share footprint and position information using limited local communication. This assumes that the robots share the same reference frame, and that the robots can communicate with each other in a limited range. These assumptions can be accommodated in many settings, i.e. (local) communication can be realised via radio or WiFi, and the common reference frame is realised by using the same map for all robots.

We introduce a sampling based approach that incorporates human avoidance. By using the sampling based approach together with a more complex evaluation function, more control over the behaviour of the robots is gained. For instance, it is straight forward to discourage robots to pass closely by humans by assigning a high cost, while closely passing by other robots has lower costs.

Lastly, we introduce the combination of the sampling based approach with the DWA method. The DWA approach is commonly used as control algorithm for local control. It is the standard method which is used on many platforms when using ROS (Quigley et al. 2009). It uses forward simulations of a set of velocity commands, known as trajectory rollouts. In our experiments, we show how the sampling based method can successfully be combined with the DWA approach to ensure good navigation within the proximity of other robots, static obstacles and humans.

The remainder of the paper is structured as follows. Section 2 summarises the related work; Sect. 3 provides the necessary background; Sect. 4 introduces the combination of on-robot localisation with the velocity obstacle paradigm. Section 5.2 extends the previously introduced algorithm with human detection and a sampling based approach and Sect. 5.3 combines the sampling based approach with the DWA planner. Section 6 presents the empirical results of the approaches. Section 7 concludes the paper and discusses future work.

2 Related work

Typically, path-planning methods for navigation are divided into global planning and local control. The global planner searches through the configuration space to find a path from the current location towards the goal location. The task of the local controller is then to steer free of collisions with any static or dynamic obstacle, while following the global plan to navigate towards a goal location.

2.1 Local control

Many approaches for local control make use of the "frozen world" assumption, i.e. that the world is static in each time-step. In Thrun et al. (2005) a number of probabilistic approaches are presented for a single robot environment. Potential fields are an approach that creates a virtual force-field in the map. Around obstacles it is pushing the robot away, and near the goal, it is pulling the robot towards it. In Koren and Borenstein (1991) the limitations of this approach are presented and described. Another approach is the dynamic window approach as described in Fox et al. (1997). However, all of these approaches lack the possibility to navigate safely within a dynamic multi-robot environment.

In multi-robot collision avoidance research, there is often a centralised controller. For instance, in Bruce and Veloso (2006) an approach for safe multi-robot navigation within dynamics constraints is presented. However, these approaches are not robust, since if the centralised controller fails, the whole system breaks. Another common approach is motion planning, which can take dynamic obstacles into account. The main assumption here is that the whole trajectory of the dynamic obstacles is known as in Ferrara and Rubagotti (2009).

In Althoff et al. (2012) a probabilistic threat assessment method for reasoning about the safety of robot trajectories is presented. Monte Carlo sampling is used to estimate collision probabilities. In this approach, the trajectories of other dynamic obstacles are sampled. This way, a global collision probability can be calculated. This work is closely related to the research done in this paper; however, that approach is probabilistic instead of the geometric representation used for the algorithms we propose.

Recently, in Bareiss and van den Berg (2015), a generalised reciprocal collision avoidance method was introduced. This method uses control obstacles, i.e. it looks which input controls may lead to a collision in the future. This enables planning for any kind of robot where the motion model is known. However, these control obstacles are non-linear making the calculations more complex. Additionally, the work does not consider static obstacles and the experiments rely on an external positioning system.

2.2 Collision avoidance in shared workspaces

This work introduces a local collision avoidance approach that deals a.o. with the problems of multiple robots sharing the same workspace with or without humans. An overview

of existing (global and local) approaches for human aware navigation (Kruse et al. 2013) shows that the main focus of current research is on the comfort, naturalness and sociability of robots in human environments. This usually entails only one robot acting in a group of humans, i.e. as a personal assistant. Our approach however, is aimed at a different distribution of agents, namely many robots navigating together with many humans in the same shared workspace.

An example of the single robot, multi-human navigation approach is the stochastic CAO approach (Rios-Martinez et al. 2012), which models the discomfort of humans and uses the prediction of human movement to navigate safely around people. Another similar approach is described in Lu (2014). It is based on layered costmaps in the configuration space and it also describes a user study where gaze-detection was used to determine the intended heading of the humans to update the costs. This layered costmaps idea is similar to the multiple evaluation functions in our approach. However, it is purely based on the configuration space, i.e. it assumes all obstacles to be static. Hence, this approach also does not cover the dynamic nature of moving obstacles as opposed to our presented approach, which uses the velocity space to explicitly model dynamic obstacles.

Similarly, the work in Linder et al. (2016) has the focus on a single robot acting in a multi-human environment. The focus is on tracking and predicting humans and classifying multiple humans into groups. This research is complementary to the work in this paper as it allows the robots to detect and track humans, which is necessary for collision avoidance.

In Alonso-Mora et al. (2015a), a collision avoidance algorithm for multiple unmanned aerial vehicles (UAVs) is introduced. In that research, a centralised and decentralised convex optimisation approach are explained and the system is integrated with two UAVs flying in close proximity of a human. However, they rely on external positioning in order localise the UAVs and the processing is performed off-board on an external machine.

Other approaches for multi-robot collision avoidance use auctions (Calliess et al. 2012) at a rather high communication overhead, or stigmergy (Theraulaz and Bonabeau 1999; Lemmens and Tuyls 2012; von der Osten et al. 2014), which relies on pheromones that are hard to apply in a real world setting. Additionally, these approaches do not implement robot–human avoidance.

3 Background

This article describes a decentralised multi-robot collision avoidance system based on the velocity obstacle paradigm as introduced in Fiorini and Shiller (1998) and per-agent localisation. This is in contrast to many other algorithms that utilise centralised planning or assume perfect knowledge about the other robots' positions, shapes and speeds.

We will present the construction of the various types of the velocity obstacles that have evolved over time to take reciprocity into account. Afterwards, three examples of how to select a new collision free velocity are explained and how dynamic and movement constraints for different type of robots can be taken into account.

3.1 Velocity obstacles (VO)

The Velocity Obstacle (VO) was introduced as an approach to deal with dynamic obstacles. The VO is a geometric representation of all velocities that will eventually result in a collision given that the dynamic obstacle maintains the observed velocity. To cover speed changes of the dynamic obstacles, it is necessary that the controller runs multiple times per second. This results in a piece-wise linear approximation of the problem.

Once we have calculated the area for each velocity obstacle in the velocity space, we have areas leading to collisions and collision free velocities. If the preferred velocity is leading to a collision, we want to find a velocity that is close to the preferred velocity but still collision free. There are several approaches to calculate this new velocity.

The subsequent definition of the VO assumes planar motions, though the concept extends to 3D motions in a straight forward manner as shown in Alonso-Mora et al. (2015b).

Let us assume a workspace configuration with two robots on a collision course as shown in Fig. 1a. 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 a 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:

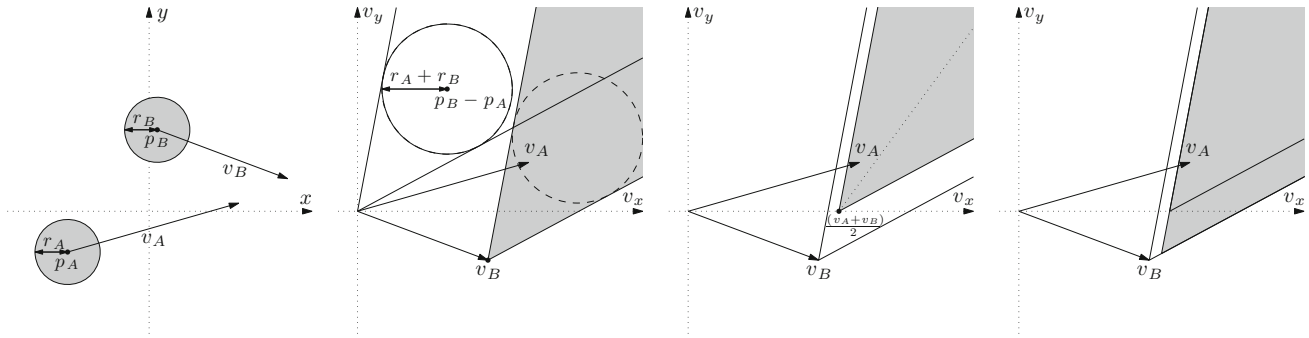
$$\mathcal{M}_{A,B} = \{a + b \mid a \in A, b \in B\} \tag{1}$$

For the remainder of this 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:

$$\theta_{left} = \max_{p_i \in \mathcal{F}_A \oplus \mathcal{F}_B} atan2((p_{rel} + p_i)^{\perp} \cdot p_{rel}, (p_{rel} + p_i) \cdot p_{rel}) \tag{2}$$

$$\theta_{right} = \min_{p_i \in \mathcal{F}_A \oplus \mathcal{F}_B} atan2((p_{rel} + p_i)^{\perp} \cdot p_{rel}, (p_{rel} + p_i) \cdot p_{rel}) \tag{3}$$



(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 R_A's current velocity, and the leg of the VO that corresponds to the leg that is furthest away from R_A's current 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)

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 *atan*2 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 centre p_{rel} as shown in Fig. 1b. The angle can then be calculated as:

$$\theta_{left} = -\theta_{right} = arcsin\left(\frac{r_A + r_B}{|p_{rel}|}\right) \tag{4}$$

In our example in Fig. 1b, it can be seen that robot R_A's current velocity vector v_A points into the VO, thus we know that R_A and R_B are on collision course. As a result, the robot should adapt its velocity in order to avoid collision.

Each agent computes a VO for each of the other agents, in our example R_B also calculates the VO induced by R_A. If all agents at any given time-step adapt their velocities such that they are outside of all VOs, the trajectories are guaranteed to be collision free.

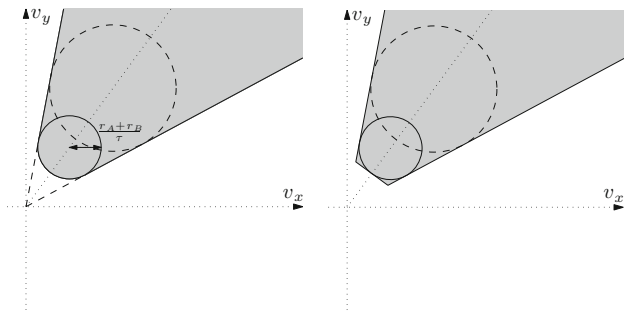
However, oscillations can still occur when the robots are on collision course. All robots select a new velocity outside of all VOs independently, hence, at the next time-step, the old velocities pointing towards the goal will become available again. Thus, all robots select their old velocities, which will be on collision course again for the next calculation, where each robot selects again a collision free velocity outside of all VOs.

To overcome these oscillations, the reciprocal velocity obstacle (RVO) was introduced in van den Berg et al. (2008). The surrounding moving obstacles are in fact 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 $\frac{v_A+v_B}{2}$ as shown in Fig. 1c. This leads to the property that if every robot chooses the velocity outside of the RVO closest to the current velocity, the robots will avoid to the same side. However, in some situations the robots will not avoid to the same side, since the selected velocity should make progress towards its goal location as well, and therefore, the closed velocity, which is collision free, is on the *wrong* side of the RVO.

To counter these situations, the hybrid reciprocal velocity obstacle (HRVO) was introduced in Snape et al. (2009, 2011). Figure 1d shows the construction of an HRVO. To encourage the selection of a velocity towards the preferred side, e.g. left in this example, the opposite 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 over-constraining the velocity space. The robot might still try to pass on the *wrong* side, e.g. another robot induces a HRVO that blocks the whole side, but then soon all other robots will adapt to the new side too.

Another problem occurs when the workspace is cluttered with many robots and these robots to not move or to only move slowly. As shown Fig. 1b, the VOs are translated by the velocity of the other agents. Thus, in these cases, the apexes of the VOs are close to the origin in velocity space. Additionally, if static obstacles such as walls are included, any velocity will lead to a collision eventually, thus rendering the robots immobile. This problem can be solved using truncation.

The idea of truncating a VO can best be explained by imagining a static obstacle. Driving with any velocity in the



(a) Truncated VO (VO^{τ}) **(b)** Approximated VO^{τ}

Fig. 2 Truncation. **a** Truncation of a VO of a static obstacle at $\tau = 2$. **b** Approximating the truncation by a line for easier calculation

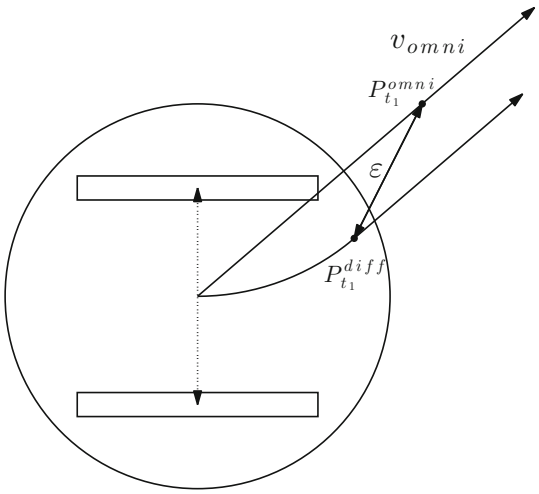


Fig. 3 The tracking error (ε) is defined as the difference between the position that a holonomic robot would be in after driving with v_{omni} for t_1 ($P_{t_1}^{omni}$) and the position of the differential drive robot at t_1 ($P_{t_1}^{diff}$)

direction of the obstacle will eventually lead to collision, but not directly. Hence, we can define an area in the velocity space, for which the selected velocities are safe for at least τ time-steps. The truncation has then 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 Fig. 2a. VO^{τ} denotes a truncated velocity obstacle. The truncation can be closely approximated by a line perpendicular to the relative position and tangential to the shrunken disk as shown in Fig. 2b. This enables easier calculations, since then each VO is defined by one line segments and two rays.

Applying the same method as creating a HRVO and RVO from a VO, we can create a truncated HRVO and truncated RVO ($HRVO^{\tau}$ and RVO^{τ}, respectively) from VO^{τ} by translating the apex accordingly.

3.2 Incorporating kinematic and dynamic constraints

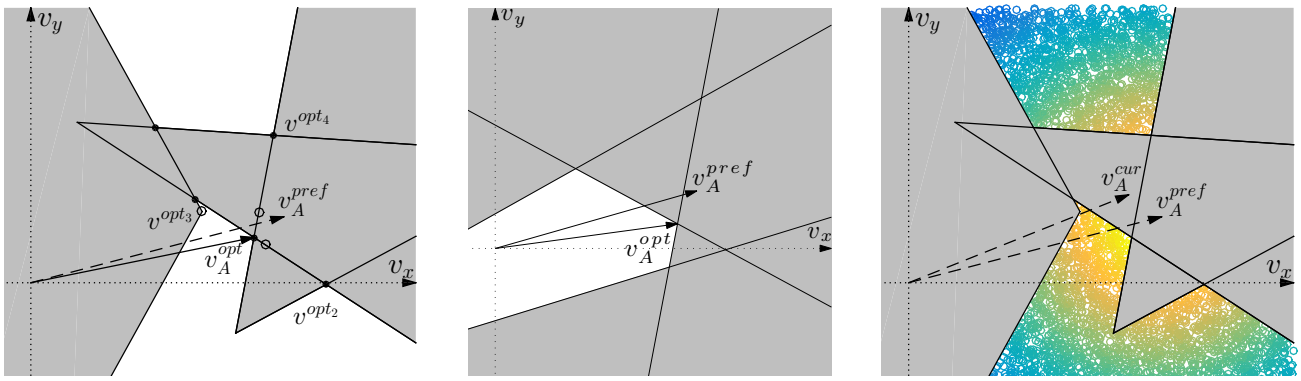
As previously mentioned, the VO paradigm assumes that the robots are able to instantaneously accelerate to any velocity in the two dimensional velocity space. This implies that the velocity obstacle approach requires a fully actuated *holonomic* platform able to accelerate into any direction from any state. However, differential drive robots with only two motorised wheels are much more common due to their lower cost. Additionally, all robots can only accelerate and decelerate within certain dynamic constraints. In this section, we will show how to incorporate these dynamic and kinematic constraints into the VO framework.

For a holonomic robot, when the acceleration limits and motion model of are known, the region of admissible velocities can be calculated and approximated by a convex polygon. This region defines the set of velocities that is currently achievable. In other words, we limit the allowed velocity space by calculating the maximal and minimally achievable velocities in both x and y direction, and only allow velocities inside this region.

A method to handle non-holonomic robot kinematics has been introduced in Alonso-Mora et al. (2010). The approach to handle dynamic and kinematic constraints can be applied to any VO-based approach. 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 an arc and then along a straight line which is parallel to a holonomic vector in that direction as shown in Fig. 3. The time needed to get parallel to the holonomic trajectory is defined as t_1. The tracking error (ε) is then defined as the difference between the position that a holonomic robot would be in after driving with a holonomic velocity (v_{omni}) for t_1, shown as ($P_{t_1}^{omni}$), and the position of the differential drive robot at that time ($P_{t_1}^{diff}$).

A set of allowed holonomic velocities is calculated based on the current speed and a maximum tracking error ε. To allow smooth and collision free navigation, the virtual robot footprints have to be increased by the tracking error, ε, since the robots only track the desired holonomic velocity with the defined error.

Using this approach, we can approximate the region of possible holomonic velocities using a polygon, and only allow the robots to choose a velocity within that region. In the next section, we will introduce three possible methods to select a new collision-free velocity.



(a) ClearPath **(b)** ORCA **(c)** Monte Carlo Sampling

Fig. 4 **a** 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). The next best points are shown for reference. **b** ORCA creates a convex representation of the *safe velocity space* and uses linear programming to find the closest point to the preferred velocity. **c** We can also use Monte Carlo sampling to select the best velocity. The distance for each sample to the preferred velocity (dashed line) is evaluated. If the sample falls within any VO, it is discarded. Yellow shows a high rating and blue is a low rating (Color figure online)

3.3 Selection of the best velocity

When all velocity obstacles are calculated, the union of these velocity obstacles depicts the set of velocities that will eventually lead to a collision. Vice versa, the complementary region is the region that holds all *safe velocities*, i.e. velocities that are collision free. If we are using truncation, the region is collision free for at least the defined τ time-steps. Additionally, we limit the velocity space according to the dynamic and kinematic constraints, as explained in the previous section.

The new velocity has to be selected Within the remaining region. In order to do this efficiently, there are several ways to calculate the new velocity. Usually, we are following a global plan, which gives us a general direction in which we want to move. This is our preferred velocity v^{pref}. Recently, some algorithms were introduced that aim to solve this problem efficiently, namely the ClearPath algorithm (Guy et al. 2009) and ORCA (van den Berg et al. 2011). ClearPath follows the general idea that the collision free velocity that is closest to the 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 4a shows the graphical interpretation of the algorithm.

With ORCA, the VOs are translated into half-planes which constrain the velocity space into a convex space. The optimal velocity is then in this space and linear programming is used to find the optimal solution for the current situation. An example is shown in Fig. 4b.

Another method is to generate possible sample velocities based on the motion model of the robots and test whether these velocities are collision free and how well they are suited. Each sample gets a score according to one or multiple cost functions, i.e. distance to current and preferred velocities and whether it is inside a velocity obstacle or not as shown in Fig. 4c. The velocity samples should be limited to the velocities that are achievable in the next timestep. If the velocity is not holonomic, the samples can be translated to approximate holonomic velocities as presented in the previous Sect. 3.2. We rollout a trajectory using the current velocity sample and then use the position to calculate the approximate holonomic velocity and the corresponding tracking error.

3.4 Adaptive Monte-Carlo localisation

The localisation 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 Localisation method, which dynamically adapts the number of particles (Fox 2003).

Monte-Carlo Localisation (also known as a particle filter), is a widely applied localisation method in the field of mobile robotics. It can be generalised in an initialisation phase and two iteratively repeated subsequent phases, the prediction and the update phase.

In the initialisation 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, w^i) has a x- and y-value and a rotation $s^i = (\hat{x}, \hat{y}, \hat{\theta})$ and a weight (w^i).

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

Fig. 5 Typical particle filter situations. **a** A well localised robot at the end of a corridor resulting in a particle cloud with small variance. **b** In an open ended corridor, 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

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. We refer to Fox (2003) for further details.

In our work, AMCL is not used for global localisation, but rather initialised 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 corridor; 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 Fig. 5.

3.5 Navigation using the dynamic window approach and a global plan

When a robot is able to successfully localise itself in an environment, (as, for instance, when using ACML with a pre-recorded map as explained in the previous section) to be autonomous, the robot has to be able navigate to a given goal location.

A commonly used approach for this navigation is the Dynamic Window Approach (DWA) (Fox et al. 1997) together with a global path planning algorithm. This global path planning algorithm is usually a Dijkstra or A* (Hart et al. 1968) type of search based on the known grid map which is, for instance, created using gmapping or HectorSLAM as described in the previous section. Detected obstacles are marked in this map when they are seen by one of the robot's sensors. In order to create an environment for fast and efficient search, the obstacles that are marked in the map are inflated by the robot's circumscribed radius. This simplifies the problem since the robot can now be seen as a point.

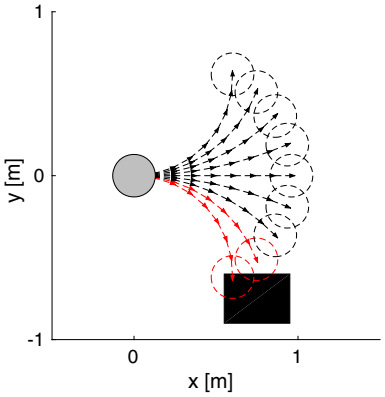


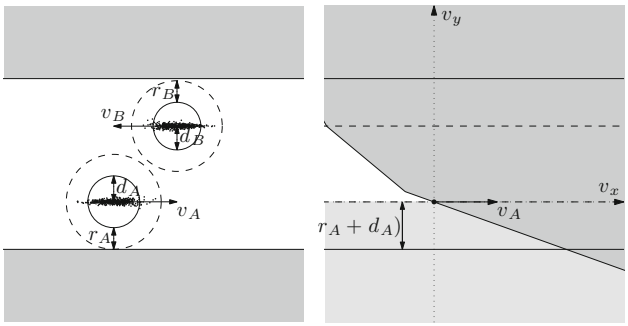
Fig. 6 DWA generates various sample control inputs and uses forward simulations in the configuration space to detect if the given combination of control inputs leads to a collision. In this example, the lower two trajectories lead to a collision and will be excluded

After the global path is found, a local control algorithm (such as the previously mentioned DWA) has the task to follow this path towards the goal while staying clear of obstacles. DWA creates samples in the control space of the robot. More specifically, it creates samples in every possible velocity dimension that the robot is actuated in. For example, a differential drive robot can be actuated in linear velocities in *x-direction (forward and backward)* and angular velocities, and, a holonomic robot, can additionally be actuated in linear velocities in *y-direction (left and right side-ways)*. These samples are created based on the current velocity and the dynamic constraints of the robots from which the name *dynamic window* is derived.

When the velocity samples have been created, the robot uses a forward simulation to predict the effect of the given velocity in the configuration space. In other words, the robot simulates the trajectory, if the given velocity would be commanded. Afterwards, this trajectory is scored based on various cost functions. For instance, the robots footprint is imposed on each point in the simulated trajectory and if the robot is in collision at any point, the trajectory is excluded. Other cost functions are, for instance, the distance to the goal location and the distance to the given path. Figure 6 shows a graphical interpretation of the approach for a differential drive robot.

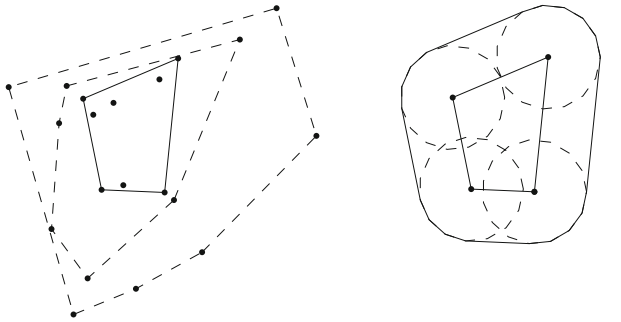
4 Convex outline collision avoidance with localisation uncertainty (COCALU)

In our previous work, *Collision avoidance under localisation uncertainty* (CALU) (Hennes et al. 2012) we successfully combined the velocity obstacle approach with on-board



(a) Configuration space **(b)** Velocity space

Fig. 7 The corridor problem: approximating the localisation 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



(a) Convex hull peeling **(b)** Minkowski sum

Fig. 8 **a** Three iterations of convex hull peeling. **b** Minkowski sum of the resulting convex polygon and a circular footprint

localisation. CALU provides a solution that is situated in-between centralised 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) (Alonso-Mora et al. 2010) to compute collision free velocities for disc-shaped robots with kinematic constraints. Uncertainty in localisation is addressed by inflating the robots' circumscribed radii according to the particle distribution of adaptive monte carlo localisation (AMCL) (Fox 2003).

While CALU effectively alleviates the need for global positioning by using decentralised localisation, some problems remain. Suboptimal behaviour 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 corridors). In both situations, the resulting VOs largely overestimate the unsafe velocity regions. Hence, this conservative approximation might lead to a suboptimal (or no) solution at all.

As an extension, we introduced *convex outline collision avoidance under localisation uncertainty* (COCALU) to address these shortcomings (Claes et al. 2012). COCALU uses the same approach based on decentralised computation, on-board localisation 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 localisation density distribution. Instead of NH-ORCA, ClearPath (Guy et al. 2009) is employed to efficiently compute new collision-free velocities in the closed-loop controller.

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 Fig. 7, 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* (Chazelle 1985), in combination with an error bound ε.

4.1 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 8a 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 Sect. 3.4.

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:

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) \approx \sum_{i=1}^{N} w_k^i \, \delta\left(\mathbf{x}_k - \mathbf{s}_k^i\right) \tag{5}$$

where $\delta(\cdot)$ is the Dirac delta measure. We recall that a particle state at time k is captured by $\mathbf{s}_k^i = \left(\hat{x}_k^i, \hat{y}_k^i, \hat{\theta}_k^i\right)$. In the limit $(N \to \infty)$, Eq. (5) approaches the real posterior density distribution. We can define the mean $\bar{} = \left(\mu_x, \mu_y, \mu_\theta\right)$ of the distribution accordingly:

$$\mu_x = \sum_i w_k^i \, \hat{x}_k^i \tag{6}$$

$$\mu_y = \sum_i w_k^i \, \hat{y}_k^i \tag{7}$$

$$\mu_\theta = atan2\left(\sum_i w_k^i \, sin\left(\hat{\theta}_k^i\right), \sum_i w_k^i \cos\left(\hat{\theta}_k^i\right)\right) \tag{8}$$

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:

$$p(\mathbf{x}_k \in \mathcal{A}|\mathbf{z_{1:k}}) = \int_{\mathcal{A}} p(\mathbf{x}|\mathbf{z_{1:k}}) d\mathbf{x} \tag{9}$$

We can rewrite (9) using (5) as follows:

$$p(\mathbf{x}_k \in \mathcal{A}|\mathbf{z_{1:k}}) \approx \sum_{\forall_i : \mathbf{s}_k^i \in \mathcal{A}} w_k^i \, \delta\left(\mathbf{x}_k - \mathbf{s}_k^i\right) \tag{10}$$

From (10) we see that for any given $\varepsilon \in [0, 1)$ there is an $\mathcal{A}$ such that:

$$p(\mathbf{x}_k \in \mathcal{A}|\mathbf{z_{1:k}}) \geq 1 - \varepsilon \tag{11}$$

Given sufficient samples, the localisation uncertainty is thus bounded and we can guarantee that the robot is located 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 Fig. 8b.

4.2 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 $\mathcal{O}(n \log h)$ (Chan 1996), 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 $\mathcal{O}(n \log n)$ (Chazelle 1985).

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.

ClearPath runs in $\mathcal{O}(N(N + M))$, where N is the number of neighboring robots and M the number of total intersection segments (Guy et al. 2009).

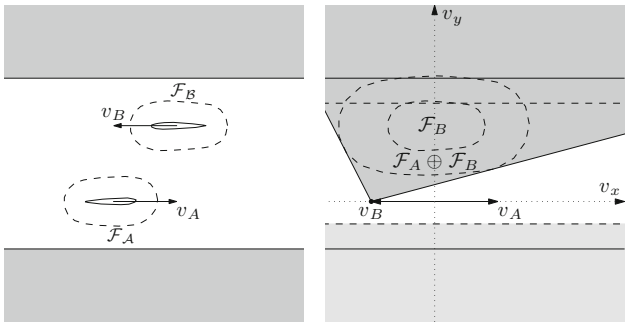
Using convex hull peeling for approximating localisation uncertainty and convex footprints solves the corridor problem. Comparing Figs. 7 and 9 shows the differences when using CALU and COCALU. In the latter figure, it can be seen that the robots can easily pass each other even without adapting their path.

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; ε: error bound; v^{pref}: preferred Velocity; τ: truncation timesteps

Output: v^{new}: New collision free velocity

$bound \leftarrow 0$;
while $bound \leq \varepsilon$ **do**
  Create convex hull $\mathcal{C}$ of $\mathcal{S}$;
  $bound \leftarrow bound + \sum_{\forall i : s^i \in \mathcal{C}} w_i$;
  $\mathcal{P} \leftarrow \mathcal{P} \setminus \{(s^i, w^i) \in \mathcal{P} | s^i \in \mathcal{C}\}$;
$\mathcal{M} \leftarrow \mathcal{F} \oplus \mathcal{C}$;
foreach $(\mathcal{F}_j, p_j, v_j) = A_j \in \mathcal{A}$ **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 $HRVO^{\tau}_{A_j}$ from $HRVO_{A_j}$ with τ;
Use ClearPath to calculate new velocity v^{new} from v^{pref} and all $HRVO^{\tau}_{A_j}$;



(a) Configuration space **(b)** Velocity space

Fig. 9 Using COCALU solves the corridor problem. Since the robots' footprints and localisation uncertainty are approximated with less overestimation, the robots can pass along the corridor without a problem

5 Towards human-safe pro-active collision avoidance

While the previous algorithms, CALU and COCALU, provide guaranteed safety and even optimality for the individual agents, there are still some limitations that remain. Specifically, the algorithms calculate the velocity that is closest to the preferred velocity and still safe. This implies that the robots always pass each other within only marginal distances. While this approach is feasible in simulation, in real world applications it is not always possible to exactly control the velocity of the robots. With only marginal distances between the robots that pass each other, there is an increased risk that the smallest error in control will lead to a collision. An additional limitation is that all agents, either human or robot, are treated in the same way, while it would be desirable to preserve more distance from humans than from other robots.

Furthermore, if a robot knows that another robot is running the same algorithm (e.g. by using communication), it can drive closer to that robot since it can assume that the other robot will partly take avoiding actions as well. While when driving towards other robots and, particularly in the presence of humans, more distance is recommended.

To tackle these problems, we introduce a pro-active local collision avoidance system for multi-robot systems in a shared workspace that aims to overcome the stated limitations. The robots use the velocity obstacle paradigm to choose their velocities in the input space; however, instead of choosing only the closest velocity to the preferred velocity, more cost features are introduced in order to evaluate which one is the best velocity to choose. This allows us to apply different weights or importance factors for passing humans, other robots, and static obstacles. Furthermore, we introduce a smart sampling technique that limits the need to sample throughout the whole velocity space. The resulting algorithm is decentralised with low computational complexity, such that the calculations can be performed online in real time, even on lower-end onboard computers.

As explained previously, some problems and limitations remain when using COCALU. In the previous approach, we have focused on the robot–robot collision avoidance, i.e. the robots head with a straight path to the goal, and only needed to deviate to avoid other robots. Thus, static obstacles have been ignored.

Additionally, VO-based methods tend to end up in deadlock situations. This means that they come to a situation in which the optimal velocity is zero since it is the only velocity not leading to a collision. This is especially problematic with many static obstacles since the environment does not change. Thus as soon as the robot is in a situation in which the best velocity is zero it will stay this way forever.

Unfortunately, optimality can be defined in many ways. In the case of COCALU, optimality means driving as close as possible to the desired speed without collisions. In many cases this implies that robots using this algorithm pass each other close to zero distances, i.e. there is no margin for error. In real life, where control of the robot is not instantaneous and perfectly accurate, it is likely this will lead to collisions. While COCALU implicitly provides safety by enlarging the robots' footprints by the localisation uncertainty, this is not an optimal solution. This is evident when the localisation accuracy is high, the point cloud converges to the actual robots position and the safety region decreases. Therefore, we need to explicitly take this into account.

Another limitation of the approach is that it is perceived as uncomfortable or unsafe by humans when the robots pass unnecessarily close by. An intrusion of ones personal space is usually not appreciated, especially when it concerns a robot.

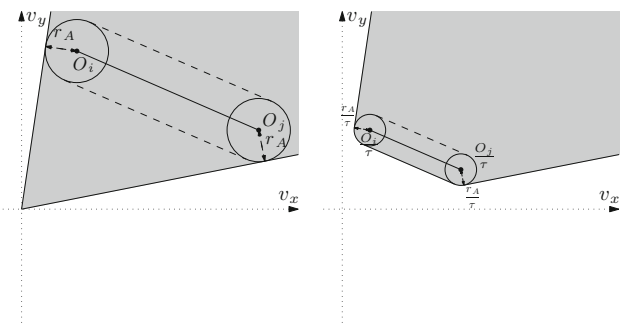
In the following subsection, we present additions and extensions to the previous COCALU approach in order to tackle the problems outlined above.

5.1 Static obstacles with VO-based methods

In order to avoid static obstacles in VO-based methods, they can be integrated as if they are static agents. Figure 10a shows the construction of a VO for a round robot with radius r_A and an obstacle line-segment defined by two points O_i and O_j. The construction follows the same rules as already presented in Sect. 3. Additionally, if we detect a complete outline of the obstacles, we can use the Minkowski sum of the robots footprint with the detected outline and compute the VO according to Eqs. (2) and (3).

Since static obstacles, by definition, do not move, we have to truncate the VO by τ since otherwise the apex of the VO is at the origin of the velocity space, and the robot is rendered immobile as soon as it is surrounded by obstacles as for instance in a room. The walls would induce a VO in any direction since all velocities will lead to a collision eventually.

Likewise, We cannot translate the VO, e.g. to create a RVO or HRVO, since these types of VO are based on the



(a) VO_{obst} for a robot R_A **(b)** VO^{τ}_{obst}

Fig. 10 **a** Constructing a *VO* out of a robots' footprint and an obstacle line-segment. **b** Truncating the VO by τ

assumption that the other robot takes part in the collision avoidance, which is not the case for static obstacles.

Finally, as explained above, VO-based methods tend to end up in dead-lock situations. This can be overcome by adding a global planning method on top of the local VO-based controller. The global planner computes a path to the goal and feeds waypoints to the controller. The direction of these waypoints than determines the preferred velocity for the VO-based algorithm. As soon as the controller does not find a valid non-zero velocity, the global planner is called again in order to recompute a new path.

5.2 COCALU with Monte Carlo sampling

Our proposed algorithm has the same assumptions as COCALU. The robots have to be able to sense velocity and shape of other robots and humans. The detection of other robots and humans is a whole research field in itself. For instance, the *Social situation-aware perception and action for cognitive robots (SPENCER)* project[1] is a European Union funded initiative of six universities with the goal to enable robots to work in human environments. Implementing this detection based on sensors is out of scope of this thesis, thus, we rely on communication between the robots. More specifically, the robots use the same global reference frame and constantly broadcast their positions via WiFi. For the human detection, we rely on the code that was made available for ROS in the SPENCER project (Linder et al. 2016).

There are multiple ways to ensure that the robots are passing each other with more distance between them. One straightforward idea is to virtually increase the size of the robots' footprints. This results in larger velocity obstacles and consequently the robots will have more distance between one another. However, this also drastically reduces the safe velocity space as shown in Fig. 11. This approach marks more regions in the velocity unsafe and therefore reduces the options to choose from. It can lead to problems in dense situations when many other robots are present and the entire velocity space is marked unsafe though it still would be possible to manoeuvre without collisions.

To overcome this problem, we use a Monte Carlo sampling based approach with multiple cost functions. This means that the chosen velocities get evaluated not only by their distances from the preferred goal velocity but by multiple other evaluation functions. Figure 12 shows the result of different evaluation functions in the example setting. The distances of the sampled velocity against the preferred velocity but also against the current velocity are shown. Likewise, it is shown how the closest distance to any velocity obstacle can be modelled as negative cost. The resulting distance can be limited, i.e. that points which are further away than a set distance do not get scored higher. This can effectively control the behaviour of the robot. Similarly, if we assume that a velocity obstacle is induced by a human, this can be weighted differently than the distances from the other velocity obstacles. The effect is shown in Fig. 12c, d where the right most velocity obstacle is weighted with double the cost than the other two velocity obstacles. Using this approach, we can model the personal space of a human by setting the cost for intrusion very high up to a certain distance. For personal space, a distance of 50 cm is usually regarded as applicable (Kruse et al. 2013).

[1] http://www.spencer.eu/.

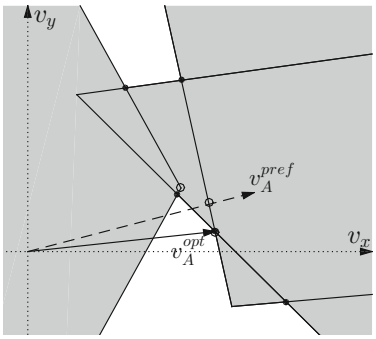
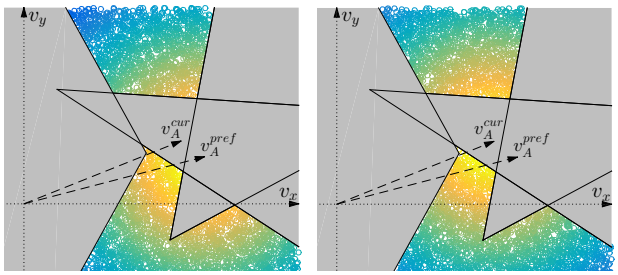
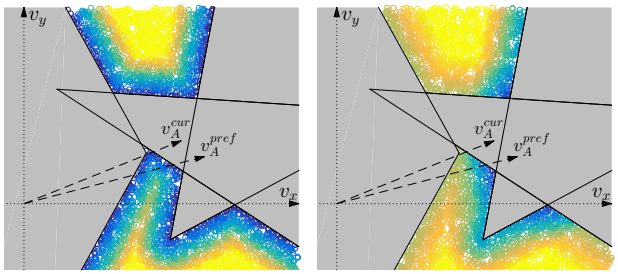


Fig. 11 Increasing the footprint of the other robots is one way to create more safety. However, this reduces the available safe velocities to choose from and could lead to potential problems in dense configurations where the whole velocity space becomes unavailable

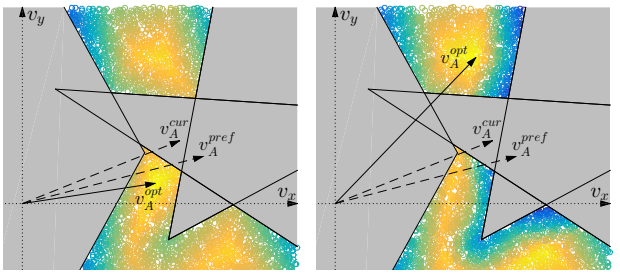


(a) Costmap v^{pref} **(b)** Costmap v^{cur}



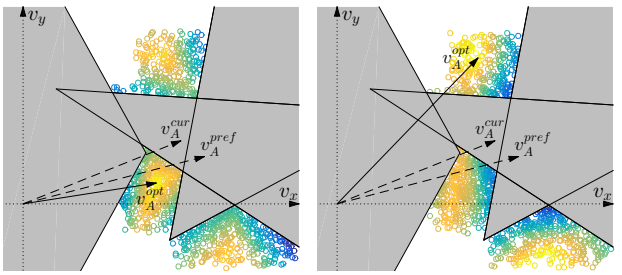
(c) Costmap $dist_{VO}$ with same weights **(d)** Costmap $dist_{VO}$ with different weights

Fig. 12 Different cost functions for evaluating a velocity. Parts in yellow depict lower, i.e. better costs, and parts in blue show higher costs. The distances to the preferred velocity (**a**) and current velocity (**b**) are shown as cost, where further away yields higher cost. **c** and **d** show the distances to the VOs as costmaps, where points closer to the VOs yield higher cost (Color figure online)



(a) v^{opt} with all VOs weighted equally **(b)** v^{opt} with different VO weights

Fig. 13 Selecting the optimal velocity based on different combinations of the costmaps and sampling throughout the full velocity space. **a** All VOs are weighted equally. **b** The VO on the right has additional weight



(a) v^{opt} with same VO weights and smart sampling **(b)** v^{opt} with different VO weights and smart sampling

Fig. 14 Applying smart sampling only around the best v_{cp}^{opt} points as calculated by ClearPath. **a** All VOs are weighted equally. **b** The VO on the right has additional weight

In order to select the optimal velocity, we sample inside the velocity space and translate the velocity to non-holonomic motions afterwards. A velocity sample that points inside a VO is disregarded since it is unsafe. Figure 13 shows the costmaps and the resulting optimal velocity. As can be seen in Fig. 13a, the resulting velocity is close to the originally calculated optimal velocity when using ClearPath. However, when the VOs are weighted differently, the optimal velocity is in a different region of the velocity space as shown in Fig. 13b.

We can combine the ClearPath algorithm with the above idea to incorporate a smarter sampling algorithm. The ranked velocities calculated by ClearPath are used as a seed (see Fig. 4a: points marked as v^{opt_i}), such that samples are only created in the vicinity of these velocities. Figure 14 shows the idea of this algorithm. The trade-off of this approach is that it might miss the global optimum in favor of being computationally faster.

Lastly, we can also adapt the truncation factor to improve the safety against other uncontrolled robots and humans. A higher truncation time results in safer velocities since, as stated in Sect. 3, it determines the time the chosen velocity is guaranteed to be collision-free in the current configuration of the system.

5.3 COCALU with DWA

In the previous section, we have shown on how to use Monte Carlo simulations to generate the samples for the velocity of the robots. As another solution, we can generate the samples according to the motion model of the robots and translate the velocities based on the motion model to an approximated holonomic speed. This idea of so-called trajectory rollouts is applied in the well know DWA-planner (Fox et al. 1997) which is commonly used in ROS. The controller generates velocities according to the dynamic motion constraints of the robots and predicts the position-based motion model of robots. Each trajectory is evaluated according to various cost functions as presented in Sect. 3.5. While these cost function are in configuration space and not in velocity space, the similarities to our COCALU with Monte Carlo sampling approach allows us to easily combine the two planners. We can use DWA to generate the velocity samples and trajectories and evaluate the configuration space based critics as the normal DWA-planner would use and then evaluate the trajectory based on our COCALU with Monte Carlo sampling cost functions. Since the trajectory is already available, the translation to velocity space is straight forward by computing the differences of the starting point and end points and dividing by the simulation time.

A major advantage is that, since the DWA-planner is already commonly used, the COCALU cost functions can easily be added to any robot that is using the DWA-planner.

As previously mentioned, a common problem with VO-based approaches is that the velocity space is too restricted when also including static obstacles, even when truncating the VO. Furthermore, as static obstacles are immobile, it is preferable to deal with them in the configuration space. When using a trajectory rollouts approach as with DWA, we can check collisions with static obstacles by imposing our footprint on the resulting trajectory. If the footprint collides with any static obstacle, the trajectory is invalidated and discarded for this iteration.

Thus we can combine the DWA cost functions in the configuration space for dealing with features that are well represented in that space (e.g. collisions with static obstacles, progress towards the goal), with the COCALU cost functions, which are well suited to avoid dynamic obstacles. As a result, we have a navigation approach that is highly flexibly and can be used in various environments.

5.4 Pro-active collision avoidance

An advantage when using any velocity obstacle based approach is that we can easily have pro-active collision avoidance, even when the robots are standing still. When not moving, the robots' preferred velocity is zero, which can be evaluated using the same approach as while driving. Thus,

when remaining at the same position would result in a collision, the robots using this approach will pro-actively take actions and avoid the incoming robot or human. This is of course only necessary when the incoming robot is not already taking care of the avoidance itself. In the latter case, the robots that are standing still will detect that their preferred velocity, i.e. zero, does not lead to collision again, thus they remain in-place.

5.5 Complexity analysis of the approach

The complexity of this approach is the time needed to generate and evaluate all samples S. For the evaluation, the distances to any approximated truncated VO, and the distances to the preferred and current velocity have to be calculated. These are all geometric operations with linear complexity. Thus the evaluation of the samples runs in $\mathcal{O}(S \times N))$, where N is the number of neighboring robots. The generation of the samples when sampling in the velocity space is only depending on the number of samples and the random generator used. When we use the motion model to generate our samples, we can pre-compute the motions and use these motion primitives in a lookup, so the sample generation is $\mathcal{O}(S)$. If we use the dynamic window approach, we need to recompute the samples according to the current state of the robot. This depends on the resolution of the trajectory-rollouts. As previously stated, ClearPath runs in $\mathcal{O}(N(N+M))$, where M is the number of total intersection segments (Guy et al. 2009).

6 Experiments and results

The presented algorithms are implemented in the framework of the open source *Robot Operating System (ROS)* (Quigley et al. 2009). The code for the implementation in ROS can be found on *GitHub*.[2] As described above we rely on communication between the robots to broadcast their positions in a common reference frame. The robots are controlled at 10 Hz, and at each timestep the robots evaluate the current position and independently choose their preferred velocity.

As baselines, we use the original *COCALU* approach and also the commonly used *DWA* method. These baselines are compared with the newly proposed *COCALU* with Monte Carlo Sampling using the smart sampling as explained in Sect. 5.2 referred to as $COCALU^{sampling}$, and with the approach that combines *COCALU* and *DWA* as explained in Subsection 5.3, to which we refer to as $COCALU^{dwa}$.

We evaluate several performance measures: (a) number of collisions and deadlocks, (b) time to complete a single run, (c) distance travelled and (d) jerk cost. The jerk cost measures the smoothness of a path. More specifically, the jerk is the change in acceleration over time. It is defined as:

$$Jerk_{lin} = \frac{1}{2} \int \dddot{\mathbf{x}}(t)\, dt \tag{12}$$

$$Jerk_{ang} = \frac{1}{2} \int \dddot{\theta}(t)\, dt \tag{13}$$

where $\mathbf{x}$ is the forward displacement of the robot, i.e. the linear speed is $\dot{\mathbf{x}}$ and θ the robot's heading, i.e. $\dot{\theta}$ is the angular speed. A deadlock is defined in this case when the goals are not reached within 60 s and there is no collision present.

For all algorithms we use truncation of the velocity obstacles induced by other robots with $\tau = 10$, while for VOs induced by static obstacles we used $\tau = 1$. As static obstacles do not move, the truncation factor can be much decreased. The localisation uncertainty is set to $\epsilon = 0.3$, thus we include 70% of the particles in our footprint enlargement. This was selected by comparing the increase in footprint size against the average localisation error, such that the enlargement was enough to cover the mean localisation error.

All costmaps are included for the sampling approaches and are weighed equally. In the cases where there is a velocity obstacle induced by an uncontrolled robot or a human, the minimum distance to these velocity obstacles is weighed double.

6.1 Simulation runs

We have evaluated our approach in simulation using *Stage* (Gerkey and Mataric 2003; Vaughan 2008) and in real-world settings. Simulation allows us to investigate the system performance using many repetitions and various extreme settings, i.e. a very dense settings with a lot of robots.

The robots are all controlled independently and running localisation with a simulated LIDAR that is updated at 10 Hz, and has a 180° field of view. Only the positions are shared via the ROS message passing system.

For the original *COCALU* and the $COCALU^{sampling}$ approach we used the simulated LIDAR to detect the outlines of static obstacles, using the approach as described in Sect. 5.1. The $COCALU^{dwa}$ approach uses the configuration space costmaps to check the distances to the static obstacles during the trajectory rollouts. Also the distance to the path and the goals are scored.

A common scenario to evaluate movement in dense environments is to place a number of robots on a circle (equally spaced). The goals are located on the antipodal positions, i.e. each robot's shortest path is through the centre of the circle (see van den Berg et al. 2011; Alonso-Mora et al. 2010). We use a circle with a radius of 1.7 m in simulation. The goal is assumed to be reached when the robots centre is within a 0.15 m radius of the true goal. We evaluate this scenario from

[2] https://github.com/daenny/collvoid.

Fig. 15 A sample configuration of the simulation environment for the antipodal circle setting with eight robots

2 up to 10 robots. A sample configuration of the simulation environment is shown in Fig. 15.

For another less symmetric scenario, we confined the robots in a 5 by 5 m square room and placed them using a uniform random distribution. Additionally, static obstacles with a square size of 0.4 by 0.4 m were placed in the same environment. The generated positions were constrained such that each robot was at least 0.9 m apart, to ensure that it is not in collision with another robot or a static obstacle.

The goals for the robots were also randomly generated, with the condition that they have to be at least 2 m away from the current position. We call this the *random with obstacles* setting. This setting is evaluated with 6 static obstacles and from 2 up to 10 robots, and with 10 static obstacles from 2 up to 6 robots.

All experiments in simulation are run on a single machine with a quad-core 3.4 GHz Intel i7 processor and 16 GB of memory. Each setting is repeated 50 times and the results are averaged. Runs in which collisions occurred or which had deadlocks are excluded from the averages. We calculate 90% confidence intervals using the student t-distribution. The simulations are run in real time since the message passing is an essential component of the described approach. As the ROS message passing uses real time serialisation and deserialisation, increasing the simulation speed would lead to inaccurate results.

The amount of collisions and deadlock are summarised in Table 1.

In the antipodal circle experiment (Table 1a), using *COCALU* only in two runs with ten robots a collision occurred, and for $COCALU^{sampling}$ no collision occurred at all. With $COCALU^{dwa}$ the amount of collision runs increases from three with five robots up to fourteen with 10 robots. The pure *DWA* method, does not have any way to avoid the incoming robots. As the shortest path is through the centre of the circle for every robot, the robots collide in every run.

The collisions that occur with the other approaches can have multiple reasons. As said before, the localisation uncertainty epsilon was set to 0.3, which means that there is a chance that collisions between the robots happen, when AMCL is unable to track the robots' positions sufficiently accurate. Additionally, the limited update rate of 10 Hz and the low fidelity of the simulator, which only approximates the kinematics of the robots, might lead to inaccurate trajectories and therefore collisions. Especially for *COCALU* and $COCALU^{sampling}$, collisions happened with the static

Table 1 Collisions (first number) and deadlocks (second number) for the different settings

	2	3	4	5	6	7	8	9	10
(a) Antipodal circle									
DWA	50/0	50/0	50/0	50/0	50/0	50/0	50/0	50/0	50/0
C	0/0	0/0	0/0	0/2	0/0	0/2	0/4	0/3	2/2
$C^{sampling}$	0/1	0/4	0/2	0/7	0/4	0/0	0/2	0/7	0/14
C^{dwa}	0/0	0/0	0/0	3/0	4/0	10/0	13/0	11/1	14/1
(b) Random with 6 obstacles									
DWA	9/0	14/0	25/1	28/0	37/0	46/1	49/0	50/0	50/0
C	0/1	2/2	2/7	4/6	3/4	3/10	7/10	4/15	12/16
$C^{sampling}$	0/0	0/7	0/5	2/6	1/9	2/10	5/7	1/13	4/15
C^{dwa}	0/0	2/0	2/1	0/1	3/2	8/3	5/2	5/4	17/5

	2	3	4	5	6
(c) Random with 10 obstacles					
DWA	3/1	13/3	30/2	37/3	46/2
C	1/6	2/6	2/9	5/14	4/26
$C^{sampling}$	1/5	2/9	4/7	7/19	5/15
C^{dwa}	2/2	3/1	11/1	6/6	7/6

For visual purposes, *COCALU* is abbreviated with *C*

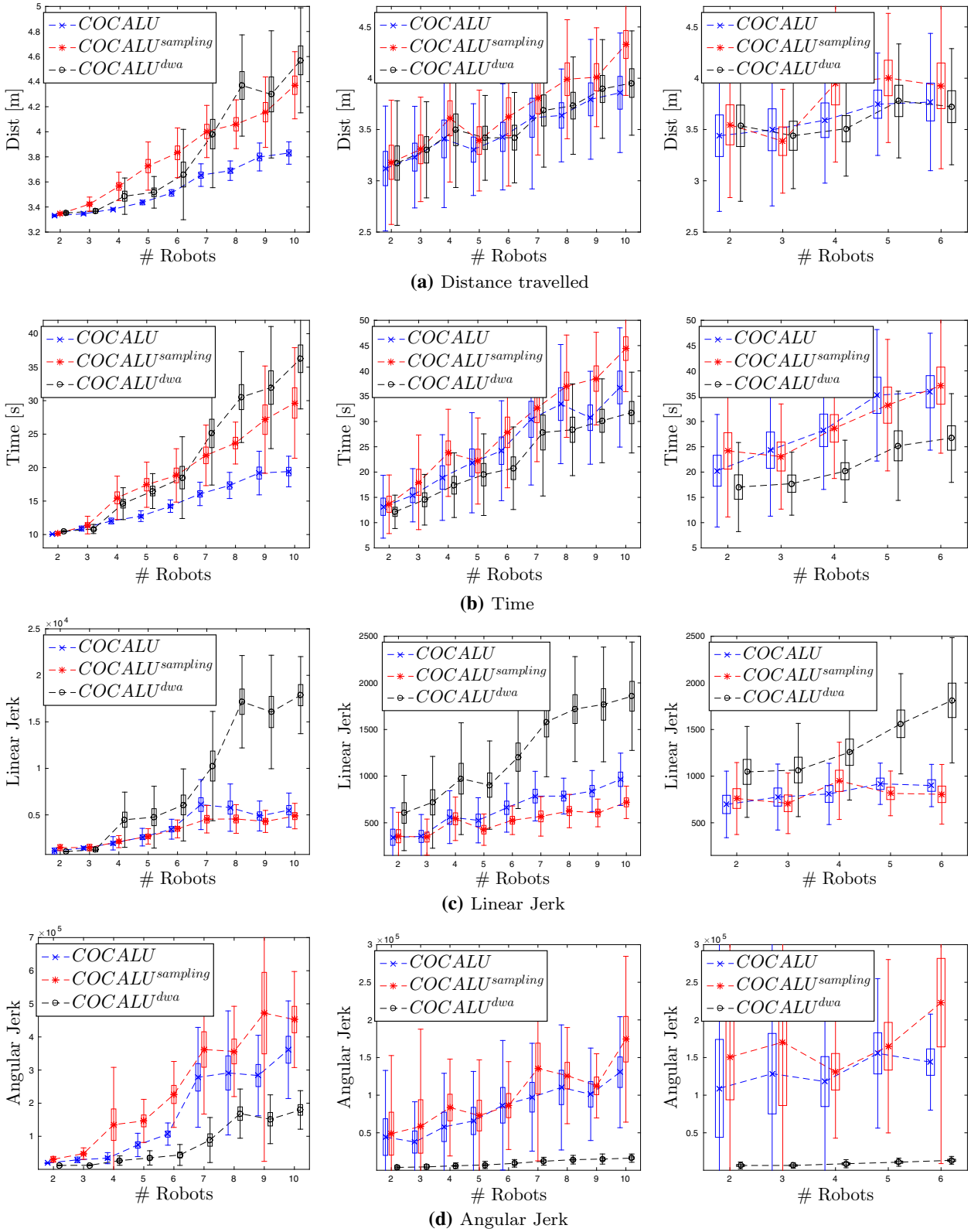


Fig. 16 Evaluation of 50 runs in simulation of the antipodal circle (left), random with 6 obstacles (center) and random with 10 obstacles (right). The (**a**) distance travelled, (**b**) time to complete, and linear (**c**) and angular (**d**) jerk are shown. The boxes show the 90% confidence intervals and the whiskers show the standard deviation

obstacles, since using the LIDAR for the detection based on the outlines and then using the VO approach can lead to inaccurate footprints due to noise in the measurements. *COCALU*^{dwa} has the advantage of dealing with the static obstacles in configuration space, which leads to a more accurate representation and less collisions with static obstacles. However, for *COCALU*^{dwa} the collisions usually occurred with other robots. If not configured correctly, the costs for the distance to the paths and goal can have too much weight such that the robot acts mostly according to those and does not take actions to avoid the collisions according to the *COCALU* cost functions. At some point the robot is in a inevitable collision state, i.e. it cannot prevent collision due its kinematic constraints and the other robots are too close.

With *COCALU*^{sampling} the amount of runs that exceeded the 60 s time limit increases with more and more robots. This is usually due to having a dead-lock situation. More specifically, this means that some robots already have reached their goals, while the others are trapped behind these robots and are not able to reach their goals anymore. With the sampling based method, this can happen due to the different cost-functions that incentivise safe paths, which is to stay away from the other robots.

For the random setting, the results are more mixed. The original *DWA* method was able to complete a couple of runs. Especially with few, i.e. two or three, robots, there is a chance that the paths of the robots do not even cross. On the other hand, we can see that if the number of robots is increased, and the environment becomes more complex, *DWA* is not able to deal with it at all, leading to failing almost every single run due to collisions.

When comparing the *COCALU* based methods, the original *COCALU* method performs worst for most of the scenarios. With 10 obstacles, *COCALU*^{dwa} yields the best performance, while with 6 obstacles the differences are less pronounced. This is probably due to the better handling of the obstacles when using the configuration space instead of translating the outlines of the static obstacles into VOs.

The results for the other metrics are summarised in Fig. 16. The results for *DWA* are excluded since the amount of runs with collisions was too high to build sensible statistics. For the antipodal circle experiment (left column), we can see that the *COCALU*^{dwa} approach performs comparable to the *COCALU*^{sampling} and *COCALU* approaches in terms of time and distances travelled. This holds for up to six robots. Afterwards, the performance deteriorates. The *COCALU*^{sampling} approach uses more time and travels farther than the original *COCALU* approach. This is to be expected since the robots deviate from the fastest path in order to improve safety and the costmaps are designed to give incentives to not choose the velocities which leave no margin for error.

When looking at the linear and angular jerk, it can be seen that *COCALU* and *COCALU*^{sampling} use significantly more angular jerk, while *COCALU*^{dwa} uses a lot more linear jerk. This is due to the differences in the sampling methods for the velocities. *COCALU* uses ClearPath for selecting the best velocity and *COCALU*^{sampling} uses smart sampling around the ClearPath points to find the best velocity. These are based on the (holonomic) velocity space and then translated into linear and angular commands. Thus when the ClearPath point switched, this leads to a large change in the angular velocity, while *COCALU*^{dwa} uses trajectory rollouts which are sampled based on the kinematic model of the robot, leading to less changes in direction, but more in linear acceleration and deceleration.

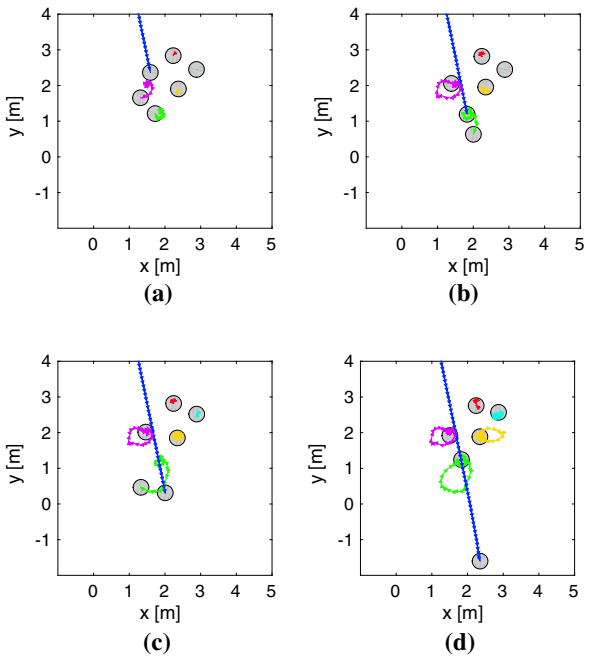


Fig. 17 Pro-active collision avoidance. The *uncontrolled robot* (blue trace) neglects the presence of the other robots and drives straight towards a crowd of other robots. **a** The robot shown with pink traces is the first to start avoiding to ensure safety. **b** The robots with green and yellow traces have to move out of the way, while the pink (**c**) returns to its original place. **d** Due to localisation errors, the yellow traced robot has to readjust its position to return to its original place (Color figure online)

For visual inspection, some sample trajectories for 7 up to 10 robots and 6 obstacles are shown in Figure 22. Generally, it can be observed that *COCALU*^{dwa} has smoother trajectories, which reflects the less usage of angular jerk. With the other two approaches, the robots manoeuvre more. Additionally, in the setting with 10 robots, it can be seen that the robot with red traces, starting in the lower right corner, with *COCALU* has a collision with a static obstacle, while with the other two approaches, the robot reaches its goal.

6.1.1 Pro-active collision avoidance

To show how the pro-active collision avoidance works, Fig. 17 shows the trajectories of one "uncontrolled" robot

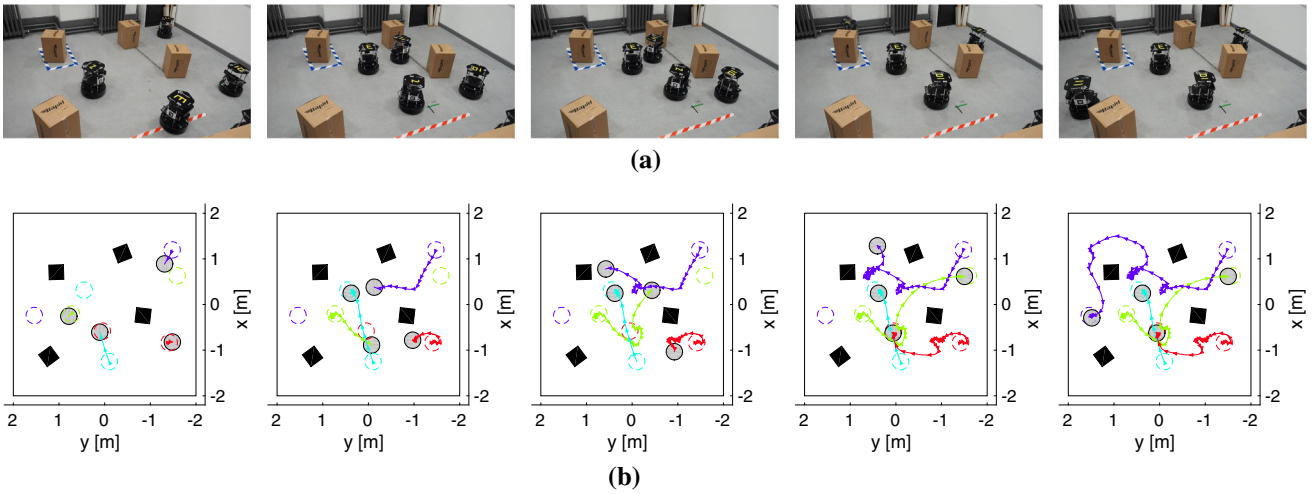


Fig. 18 A run in the real world setting. **a** Pictures of the actual run. **b** Trajectories of the robots. The initial and target positions are marked in dashed circles. The robot with purple traces (starting top right) has to readjust his path twice. The red robot waits until the green robot has passed (Color figure online)

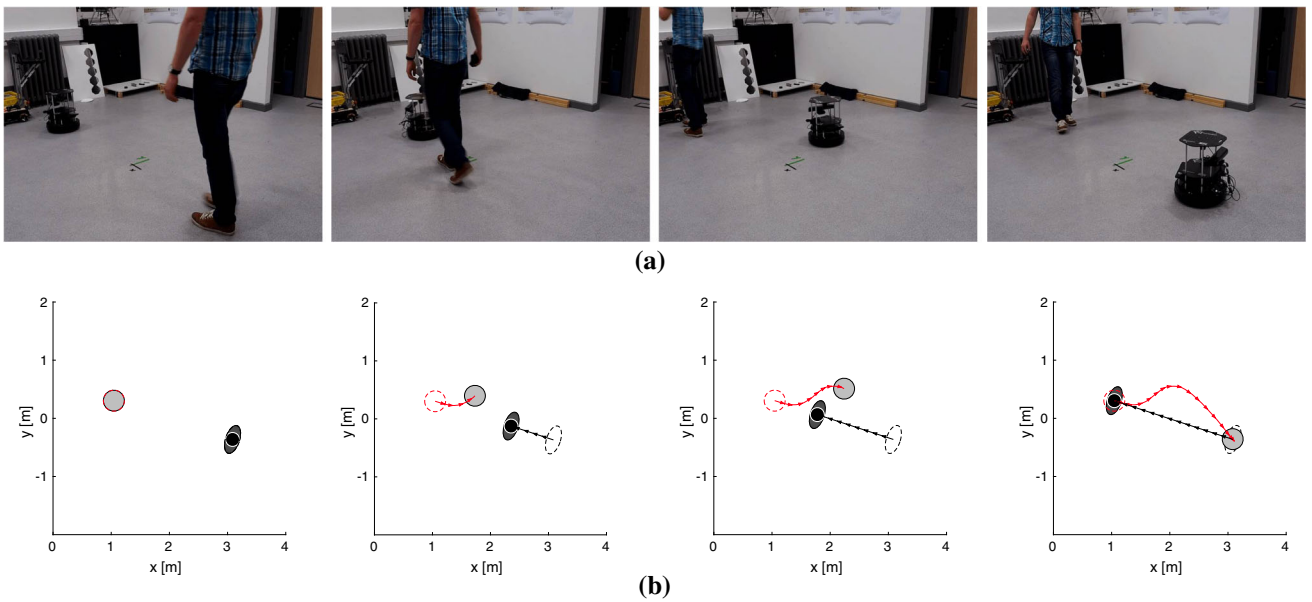


Fig. 19 Testing with a human with the antipodal circle experiment in the real world. A single robot switches place with a human. It detects the human and avoids him by adjusting its path. **a** Pictures of the actual run, **b** trajectories of the of the robot and the human. The human trajectory is approximated

passing through a crowd of robots. "Uncontrolled" in this experiment means that the robot disregards the existence of the other agents and just drives straight without taking any avoiding measures. Thus, the five robots in the centre have to pro-actively move out of the way in order to ensure safety. The robot with the blue trace is approaching, while the pink and green traced robots start moving out of the way (Fig. 17a). As soon as the uncontrolled robot has passed, pink returns to its position (Fig. 17b). The same happens with the green robot (Fig. 17c). The robot with the yellow traces just moves a little bit to clear the way, however due to localisation uncertainty, the position changes such that it becomes necessary for the robot to make a more elaborate manoeuvre to reach back to its original position. The final positions and complete trajectories can be seen in Fig. 17d.

6.2 Real world experiments

We evaluated the performance in a real-word setting using up to four differential drive Turtlebot 2's.[3] In addition to the usual sensors, they are equipped with a Hokuyo URG laser-range finder to enable better localisation in larger spaces. All computation is performed on-board on an Intel i3 380 UM 1.3 GHz dual core CPU notebook. Communication between

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

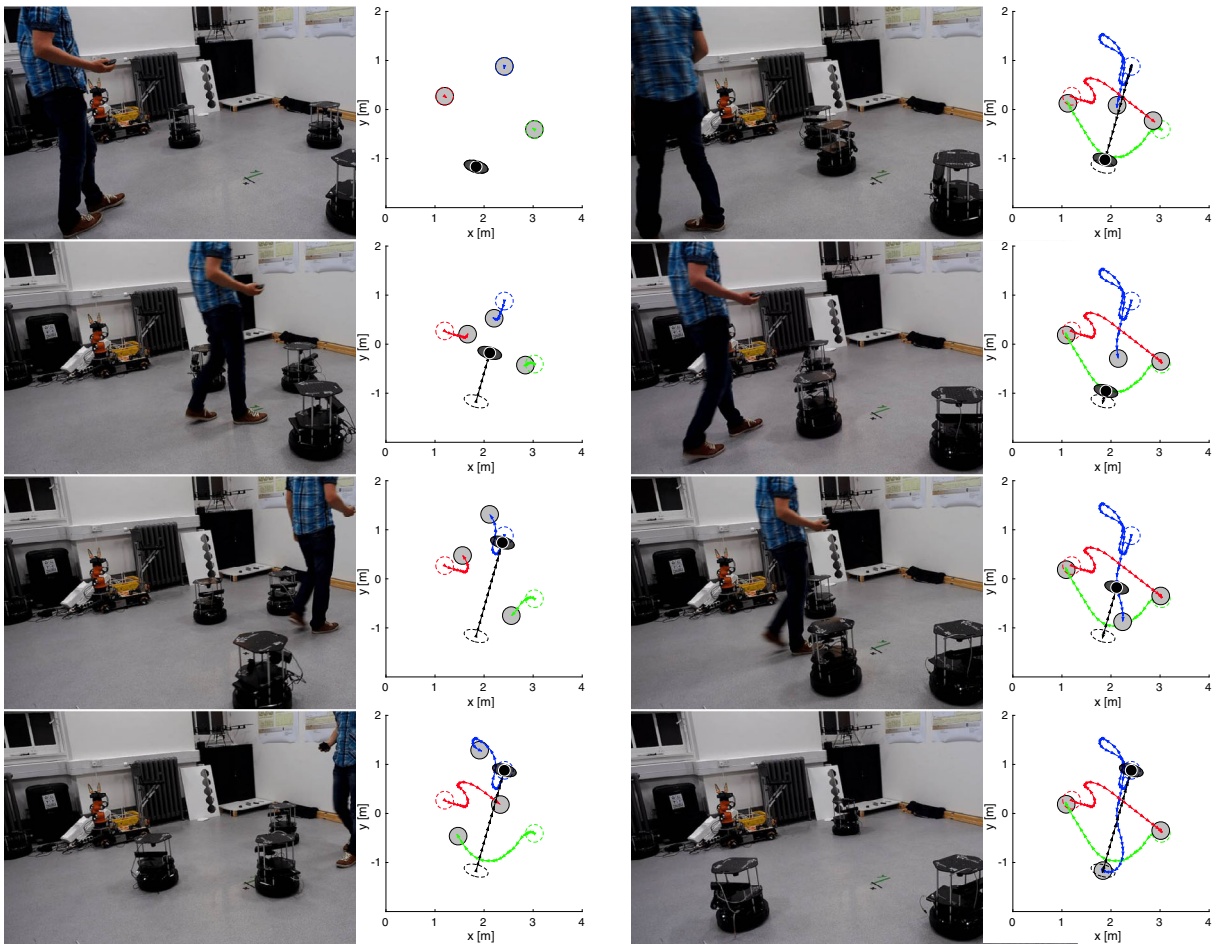


Fig. 20 Collision avoidance with three robots and a human. The human walks through the crowd of robots three times while the robots avoid him while driving to their target positions. **a** and **c** Pictures of the actual run. **b** and **d** Trajectories of the robots and the human. The human trajectory is approximated

the robots is realised via a 2.4 GHz WiFi link using a UDP connection and the LCM library (Huang et al. 2010). For human detection we use the SPENCER project code (Linder et al. 2016). It uses the laser range-finder to detect and match human legs and tracks the resulting people. We ran the random setting with four obstacles, and the antipodal circle experiment which included one human. The trajectories of the robots are recorded using the positions determined by AMCL and the human trajectories that are shown are approximated. The obstacles are shown at their estimated positions. The robots were running the $COCALU^{dwa}$ algorithm for navigation.

6.2.1 Random with obstacles

A sample trajectory of the random with obstacles setting is shown in Fig. 18. The left column (Fig. 18a) shows the photos of the run, while the right column shows the trajectory plots over time (Fig. 18b). It is tested with four Turtlebots and four obstacles. The light blue robot, starting on the bottom has a direct path to the goal. It blocks the green robot, which also reacts on the purple robot that is heading towards it. The purple robot re-plans and deviates from its original path, while the red robot waits until the green robot has passed. Afterwards the purple robot has to change its path again due to the previously unseen obstacle. Finally, all robots have reached their goals.

Some more sample trajectories of this setting are shown in Fig. 21. From visual inspection, it can be seen that the robots drive smoothly for most of the trajectories, while for sample trajectory shown in Fig. 21d, it shows the difficulties that $COCALU^{dwa}$ has in very dense configurations. The robot with the green traces, starting left, has to wait first for the robot with the red traces to move away. In the mean time, the other two robots (with purple and light blue traces) start moving towards their goal positions forcing the green robot to adjust its path multiple times to avoid them. Eventually, the robots are at their goal positions and the green robot can pass.

6.2.2 Antipodal circle

We tested our approach as well with a human switching place with a robot and passing through a crowd of robots. Pictures of the runs are shown in Figs. 19 and 20. In our first experiment (Fig. 19), we tested a robot exchanging the positions with a human. The pictures of the run are shown in the left column (Fig. 19a) and the trajectories are shown in the right column (Fig. 19b). The trajectory of the human is approximated. The human does not take care of avoiding the robot, he walks at a reasonable pace towards the robot. The robot detects the human and realises that it is on collision course. Thus, the robot avoids him by adjusting its path accordingly.

In a second experiment, we tested the antipodal circle experiment with one human and three robots. The human passes through the crowd of robots three times back and forth, while the robots have to reach the antipodal position (see Fig. 20). The pictures of the run are shown in Fig. 20a, c and the trajectories are shown in Fig. 20b, d.

The robots that are not on a head-on collision course with the human, marked with red and green traces, start detecting the human and slow down. The robot that has to exchange the place with the human (blue traces) backs away in order to avoid the collision. This results in the robot with red traces having to back away further, while the robot with green traces now has a free path towards his goal. After the human has passed for the first time, the robot with red traces has a free path to its goal position, while the robot with blue traces has avoided the human by driving towards the to right corner. Afterwards, the human returns to his starting point, thus the robot with blue traces has a free path towards its goal. At the third pass, the robot with blue traces has to avoid the human once more. A video showing the results can be found at: http://wordpress.csc.liv.ac.uk/smartlab/collvoid/.

7 Conclusions and future work

This paper introduces a navigation approach that can deal with non-mapped static obstacles and dynamic obstacles such as humans and other robots. We substantially extend our previous contributions, in which we introduced how the particle cloud of AMCL can be used as an estimator for the robots' localisation uncertainty. The robots' footprint can then be enlarged accordingly to ensure safety in multi-robot situations.

Our new approach introduces the use of a global planner to overcome deadlock situations and improve the performance in the presence of static obstacles.

Additionally, we have presented how we can use Monte Carlo sampling to select the velocities in the velocity space according to some cost functions. This allows us to easily incorporate humans in the system and tune the cost functions, such that the robots keep more distance from humans than from other robots in order to not intrude the humans' personal spaces.

Lastly, we have shown that this sampling based method can be readily added to the well known and commonly used DWA planner in the ROS framework. This allows us to use the configuration space for avoiding static obstacles and integrating the VO-based avoidance in the velocity space for dynamic obstacles.

The presented methods have been extensively evaluated in simulation and in real world settings, illustrating the feasibility of the proposed approach.

In future work, we will work on removing the communication assumption. For instance, in Tuyls et al. (2016) we have implemented robot–robot detection based on AR markers such that the robots do not need to broadcast their positions. Additionally, using control obstacles as described in Bareiss and van den Berg (2015) instead of the velocity obstacles could further improve the performance as well.

Appendix

See the Figs. 21 and 22.

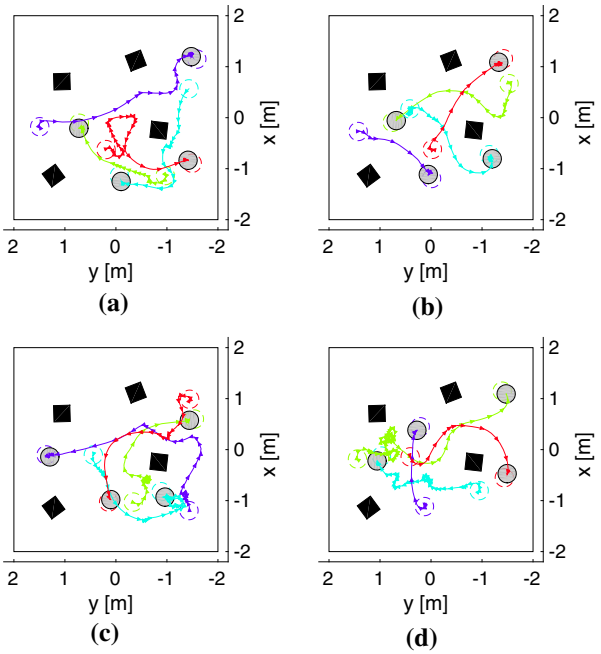
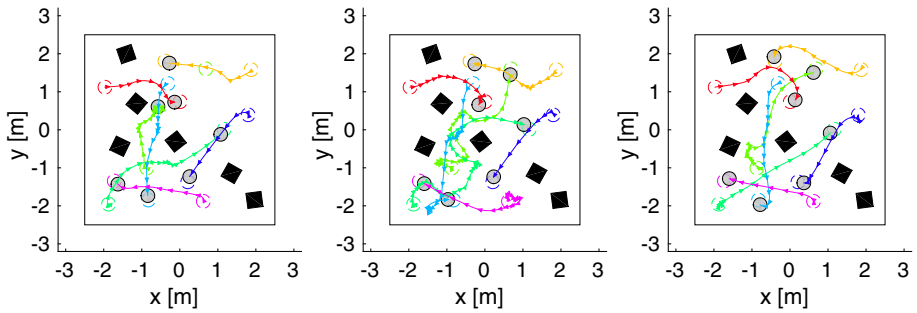
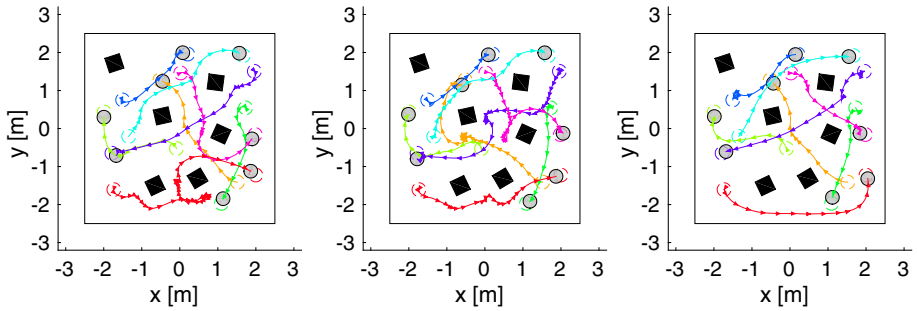


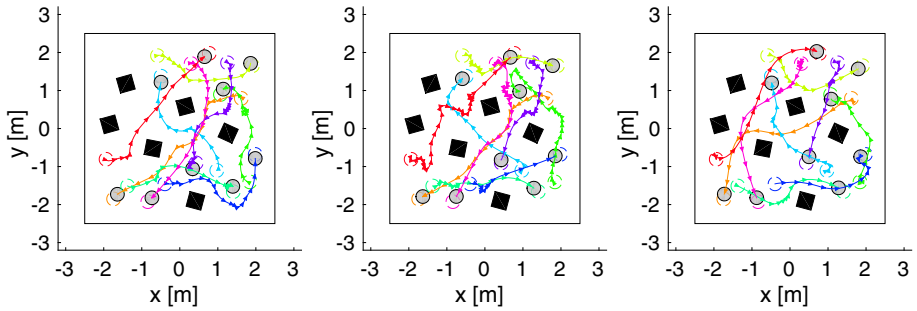
Fig. 21 Several trajectories in the real world setting. The initial and target positions are marked with dashed circles



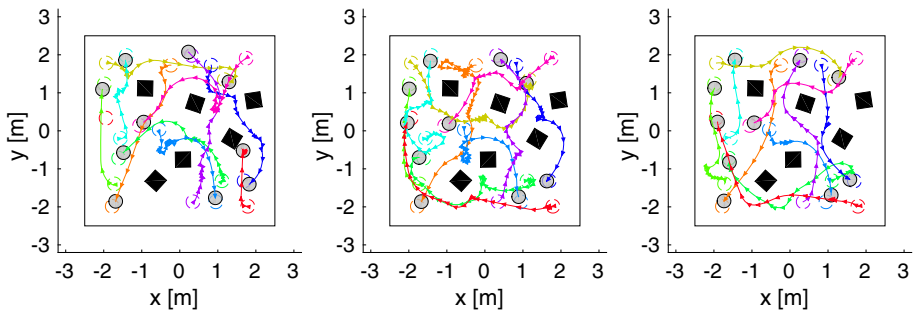
(a) 7 robots, 6 obstacles



(b) 8 robots, 6 obstacles



(c) 9 robots, 6 obstacles



(d) 10 robots, 6 obstacles

Fig. 22 Sample trajectories for the setting with random goals and 6 randomly placed obstacles. With 7–10 robots from top to bottom and the standard *COCALU* (left), $COCALU^{sampling}$ (center) and $COCALU^{dwa}$ (right). The initial and target positions are marked with dashed circles

References

Alonso-Mora, J., Baker, S., & Rus, D. (2015a). Multi-robot navigation in formation via sequential convex programming. In *Proceedings of the international conference on intelligent robots and systems* (pp. 4634–4641). IEEE.

Alonso-Mora, J., Breitenmoser, A., Rufli, M., Beardsley, P. A., & Siegwart, R. (2010). Optimal reciprocal collision avoidance for multiple non-holonomic robots. In *Proceedings of the Internationanl Symposium on Distributed Autonomous Robotic Systems, DARS 2010* (pp. 203–216). Lausanne, Switzerland, November 1–3, 2010.

Alonso-Mora, J., Naegeli, T., Siegwart, R., & Beardsley, P. (2015b). Collision avoidance for aerial vehicles in multi-agent scenarios. *Autonomous Robots*, *39*(1), 101–121.

Althoff, D., Kuffner, J., Wollherr, D., & Buss, M. (2012). Safety assessment of robot trajectories for navigation in uncertain and dynamic environments. *Autonomous Robots*, *32*, 285–302. https://doi.org/10.1007/s10514-011-9257-9.

Bareiss, D., & van den Berg, J. (2015). Generalized reciprocal collision avoidance. *The International Journal of Robotics Research*, *34*(12), 1501–1514.

Bruce, J., & Veloso, M. (2006). Safe multirobot navigation within dynamics constraints. *Proceedings of the IEEE*, *94*(7), 1398–1411.

Calliess, J. P., Lyons, D., & Hanebeck, U. D. (2012). Lazy auctions for multi-robot collision avoidance and motion control under uncertainty. In *Advanced agent technology* (pp. 295–312). Springer.

Chan, T. M. (1996). Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete & Computational Geometry*, *16*, 361–368.

Chazelle, B. (1985). On the convex layers of a planar set. *IEEE Transactions on Information Theory*, *31*(4), 509–517.

Claes, D., Hennes, D., Tuyls, K., & Meeussen, W. (2012). Collision avoidance under bounded localization uncertainty. In *Proceedings of the international conference on intelligent robots and systems* (pp. 1192–1198).

Ferrara, A., & Rubagotti, M. (2009). A dynamic obstacle avoidance strategy for a mobile robot based on sliding mode control. In *IEEE control applications (CCA) and intelligent control (ISIC)* (pp. 1535–1540).

Fiorini, P., & Shiller, Z. (1998). Motion planning in dynamic environments using velocity obstacles. *International Journal of Robotics Research*, *17*, 760–772.

Fox, D. (2003). Adapting the sample size in particle filters through KLD-sampling. *International Journal of Robotics Research*, *22*, 985–1003.

Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics Automation Magazine*, *4*(1), 23–33.

Gerkey, B. P., & Mataric, M. J. (2003). Multi-robot task allocation: Analyzing the complexity and optimality of key architectures. In *Proceedings of the IEEE international conference on robotics and automation* (Vol. 3, pp. 3862–3868). IEEE.

Guy, S. J., Chhugani, J., Kim, C., Satish, N., Lin, M. C., Manocha, D., & Dubey, P. (2009). Clearpath: Highly parallel collision avoidance for multi-agent simulation. In *Symposium on computer animation*.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107.

Hennes, D., Claes, D., Tuyls, K., & Meeussen, W. (2012). Multi-robot collision avoidance with localization uncertainty. In *Proceedings of the international joint conference on autonomous agents and multi agent systems*.

Huang, A. S., Olson, E., & Moore, D. C. (2010). LCM: Lightweight communications and marshalling. In *Proceedings of the international conference on intelligent robots and systems* (pp. 4057–4062). IEEE.

Kagermann, H., Lukas, W. D., & Wahlster, W. (2011). Industrie 4.0: Mit dem internet der dinge auf dem weg zur 4. industriellen revolution. VDI nachrichten 13:11.

Koren, Y., & Borenstein, J. (1991). Potential field methods and their inherent limitations for mobile robot navigation. In *Proceedings of the IEEE international conference on robotics and automation* (Vol. 2, pp. 1398–1404).

Kruse, T., Pandey, A. K., Alami, R., & Kirsch, A. (2013). Human-aware robot navigation: A survey. *Robotics and Autonomous Systems*, *61*(12), 1726–1743.

Lemmens, N., & Tuyls, K. (2012). Stigmergic landmark optimization. *Advances in Complex Systems*, *15*(8), 1150025.

Linder, T., Breuers, S., Leibe, B., & Arras, K. O. (2016). On multi-modal people tracking from mobile platforms in very crowded and dynamic environments. In *Proceedings of the IEEE international conference on robotics and automation* (pp. 5512–5519). IEEE.

Lu, D. V. (2014). *Contextualized robot navigation*. Ph.D. Thesis, Washington University.

Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., & Ng, A. Y. (2009). ROS: An open-source robot operating system. In *Proceedings of the open-source software workshop (ICRA)*.

Rios-Martinez, J., Renzaglia, A., Spalanzani, A., Martinelli, A., & Laugier, C. (2012). Navigating between people: A stochastic optimization approach. In *Proceedings of the IEEE international conference on robotics and automation* (pp. 2880–2885). IEEE.

Snape, J., van den Berg, J., Guy, S. J., & Manocha, D. (2009). Independent navigation of multiple mobile robots with hybrid reciprocal velocity obstacles. In *Proceedings of the international conference on intelligent robots and systems*.

Snape, J., van den Berg, J., Guy, S. J., & Manocha, D. (2011). The hybrid reciprocal velocity obstacle. *IEEE Transactions on Robotics*, *27*(4), 696–706.

Theraulaz, G., & Bonabeau, E. (1999). A brief history of stigmergy. *Artificial Life*, *5*(2), 97–116.

Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic robotics (intelligent robotics and autonomous agents series)*. Cambridge: The MIT Press.

Tuyls, K., Alers, S., Cucco, E., Claes, D., & Bloembergen, D. (2016). A telepresence-robot approach for efficient coordination of swarms. In *Proceedings of the international conference on the synthesis and simulation of living systems (Alife)*. MIT Press.

van den Berg, J., Guy, S., Lin, M., & Manocha, D. (2011). Reciprocal n-body collision avoidance. In *International journal of robotics research* (Vol 70, pp. 3–19).

van den Berg, J., Lin, M., & Manocha, D. (2008). Reciprocal velocity obstacles for real-time multi-agent navigation. In *Proceedings of the IEEE international conference on robotics and automation*.

Vaughan, R. (2008). Massively multi-robot simulation in stage. *Swarm Intelligence*, *2*(2), 189–208.

von der Osten, F. B., Kirley, M., & Miller, T. (2014). Anticipatory stigmergic collision avoidance under noise. In *Proceedings of the conference on genetic and evolutionary computation* (pp. 65–72). ACM.

Daniel Claes is currently finishing his Ph.D. research in multi robot systems at Liverpool University. During a research visit to Willow Garage in 2011, he found his passion for robotics. He participated in the RoboCup @Work and RoCKin@work competitions and was the team leader of the smARTLab@work team from 2013 until 2015. His team was the winner of the German Open robocup@work competitions in 2013 and 2014, world champion of the RoboCup@Work competitions in 2013 and 2014 and the winner of the RoCKIn@work competition in 2015. He received his B.Sc. degree in Knowledge Engineering in 2010 and his M.Sc. degree in Artificial Intelligence with summa cum laude in 2012. During his research he has received various rewards, under which: the student prize for the best thesis at Maastricht University and the best demo award at AAMAS'12 and BNAIC'13.

Karl Tuyls (FBCS) is professor of Computer Science at the University of Liverpool and is also part-time professor of BioInspired Robotics and Autonomous Systems at Delft University of Technology. Previously, he held positions at the Vrije Universiteit Brussel, Hasselt University, Eindhoven University of Technology, and Maastricht University. At the University of Liverpool he is director of research of the school of Electrical Engineering & Electronics and Computer Science. He founded and leads the smARTLab robotics laboratory since 2013 (http://wordpress.csc.liv.ac.uk/smartlab/). Prof. Tuyls has received several awards with his research, amongst which: the Information Technology prize 2000 in Belgium, best demo award at AAMAS'12, winner of the German Open robocup@work competitions in 2013 and 2014, world champion of the RoboCup@Work competitions in 2013 and 2014, winner of the RoCKIn@work competition in 2015. Furthermore, his research has received substantial attention from national and international press and media (http://karltuyls.net). He is a fellow of the British Computer Society (BCS), is on the editorial board of the Journal of Autonomous Agents and Multi-Agent Systems, and is editor-in-chief of the Springer briefs series on Intelligent Systems. Prof. Tuyls is also a member of the board of directors of the International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).