Chapter from the book Mobile Robots - Control Architectures, Bio-Interfacing, Navigation, Multi Robot Motion Planning and Operator Training


Interested in publishing with IntechOpen?
Contact us at book.department@intechopen.com
Cooperative Path Planning for Multi-Robot Systems in Dynamic Domains

Stephan Opfer, Hendrik Skubch and Kurt Geihs

University of Kassel
Germany

1. Introduction

A stationary robot in an assembly line expects to execute all its hard-coded movements without any collision. Its recognition is limited to its tasks and the corresponding area of operation. In case of a failure an alarm is given and it stops working until a human being corrects the fault. The assembly robot does not react independently, which is the exact opposite to autonomous mobile robots. An autonomous mobile robot has to solve many problems an assembly robot is never confronted with. While the latter moves according to its hard-coded patterns, a mobile robot on the contrary moves in a potentially unknown and dynamic environment. Operating in such environments requires the mobile robot to localise its position relative to the environment, sense its surroundings continuously, and move accordingly. In order to achieve its goals in a changing environment, a mobile robot also has to adapt its actions according to the changes. Hard-coded action sequences are too limited and error-prone to reliably achieve a specific goal. Finally, a secure execution demands the robot to plan its movements and their consequences. Collision-avoidance is an important topic in this context.

The requirements for an autonomous mobile robot which is part of a team of robots are even more challenging. A robot has to recognise its team members separately, include their positions and abilities in its own planning and consider their paths for its own path planning. Nevertheless, many reasons for accepting this overhead exist. One robot handling complicated tasks is possibly more expensive and complex than several single robots. Furthermore, several robots are able to solve a given task faster than a single robot, by dividing the task into subtasks and assigning them to different robots. Moreover, a team of robots exhibits reliability, as malfunctioning robots can be replaced by team members at runtime. Finally, a large spectrum of tasks can be covered by a group of robots at once, while a single robot has to be adapted for each specific task.

A central requirement to benefit from these advantages is the ability to coordinate multiple autonomous mobile robots. For this chapter we will confine ourselves to the problem of path planning in a team of autonomous mobile robots. Refining the requirements, we assume the following conditions:

**Highly Dynamic Environment** Assigned tasks are very critical with regard to time, because of a highly dynamic environment. Therefore robots must move at speeds in the order of 2-6 meters per second and smooth paths are crucial to complete tasks in time and without accidents such as rollovers.
Neutral or even antagonistic entities  It is possible that robots and other moving obstacles are crossing or blocking the intended path. Handling these situations demands reactive behaviour.

Limited Sensor Range  The sensor range of the robots is limited, so that the environment is only partially observable. Hence, destinations are often beyond the local sensor range.

Unreliable Communication  Team members are able to communicate their sensor information between each other by means of unreliable network communication, such as W-LAN. Therefore, sensor information of team members are only available with a certain delay or not at all. Moreover, due to the dynamic and time critical nature of the domain, robots cannot wait for communication to take place before acting.

Noisy Sensors  All sensor information is subject to a certain amount of noise.

The RoboCup Middle Size League\(^1\) is a domain that features these conditions. Figure 1 shows robots competing in that league. Among other scenarios, we use this domain as a test bed.

![RoboCup Middle Size League 2009](image)

These conditions require careful crafting of the whole process, ranging from sensor fusion, validation of sensed information, and tracking of objects over time, to information abstraction, planning and conflict resolution. In this chapter we will address all of these issues.

In Section 2, we will discuss the foundational techniques as well as the prior work on which our approach relies. Section 3 explains our approach in detail. It is, same as the approach itself, divided into two parts, namely a data-driven part, in which sensor fusion, tracking and information abstraction occurs, and a command driven part, in which planning and reactive conflict resolution occurs. The data-driven part is explained in Section 3.1, while the command-driven computations are described in Section 3.2. Afterwards, in Section 4, we discuss the presented framework and conclude in Section 5.

2. Foundations

This chapter explains the foundations of cooperative path planning. Examples and scenarios are drawn from the robotic soccer domain. Section 2.1 gives an overview of existing path planning algorithms and techniques in the multi-robot context. In the following, in Section 2.2 to 2.3, the algorithms used in our approach are explained in more detail.

\(^1\) [www.robocup.org](http://www.robocup.org)
2.1 Path planning in multi-robot systems

There are various approaches to path planning in multi-robot system, however, finding the optimal solution is NP-hard. Hopcraft et al. (1984) simplify the planning problem to the problem of moving rectangles in a rectangular container. They proved the NP-hardness of finding a plan from a given configuration to a goal configuration with the least amount of steps. Hence, all feasible approaches to path planning are a compromise between efficiency and accuracy of the result.

Parker (2009) classifies path planners in three groups: central, or coupled, approaches, decoupled approaches and coordinated approaches. In the following, each of them is described.

2.1.1 Central approaches

This form of planning considers the individual robots as part of a single, coupled system, and uses a single, central planning unit for calculating the plans. The employed algorithms are then directly drawn from single-robot versions:

**Potential Fields** Using derivable functions, such as gaussians, each point in space is mapped onto a real value. These values can be interpreted as a force pushing a robot away or drawing it in. In the most simple case, a robot just needs to follow the gradient to arrive at its goal. Figure 2(a) shows an example, depicting a robot with a blue dot and its goal with a cross. (Bruijnen et al., 2007; Topor, 1999)

**Space Decomposition** Here, the space between obstacles is divided into small areas. A path is represented as a set of neighbouring areas. Should multiple robots try to enter the same area, the area is sub-divided. The subdivisions are then assigned to the different robots. (Parsons & Canny, 1990)

**Combinatorial Approaches** In order to simplify the search space, it is abstracted to road maps. A road map consists of a set of globally available paths and intersections. Each robot has to follow one of the paths and exit the road map close to its goals (Peasgood et al., 2008). Voronoi graphs, such as the example in Figure 2(b) are a road map variant. We will discuss advantages of Voronoi graphs in Section 2.5.

2.1.2 Decoupled approaches

In contrast to coupled approaches, decoupled approaches consider each robot on its own. Thus, no central planning unit is necessary and every robot can calculate its own path. There are two possibilities to resolve conflicts:

- Each robot is assigned a priority. Robots with lower priority have to wait for the planning result of robots with higher priority and consider the received paths when planning.
- All paths are planned without regard for each other. Only afterwards potential conflicts are solved to avoid collision.

In contrast to central approaches, the complexity of decoupled approaches is only linear in the number of participating robots. (Parker, 2009)

2.1.3 Coordinated approaches

Coordinated approaches differ strongly from the two previously discussed groups. Favouring time efficiency over completeness, a complete path planning is omitted. Parker (2009) mentions three different kinds of coordinated approaches:
Traffic Control  Collisions and coordination problems are resolved by globally known traffic rules (i.e., right-before-left or vice versa). Possible applications include storage facilities, where autonomous vehicles transport goods on tracks. Cooperation is only necessary when multiple vehicles meet at an intersection. Usually simple rules such as the aforementioned right-before-left suffice. (Asama et al., 1991)

Reactive Behaviour  Here, the focus is on collision avoidance. Using highly reactive behaviours, long-term planning is simplified and collisions are avoided shortly before they would occur. (Mataric, 1992)

Swarm Behaviour  This group of approaches does not value that each robot reaches its goal. Instead, the swarm should exhibit a certain behaviour. For instance, one robot leads while the others follow in a certain formation. Hence, the following robots only need to consider their relative positions (Mourikis & Roumeliotis, 2006). There are various adaptations of animal swarm behaviours, such as ant swarming, buffalo migration or the following of geese.

2.2 A-star
The A-Star search algorithm is probably the most well-known informed search. It was firstly described by Peter Hart, Nils Nilsson, and Bertram Raphael (Hart et al., 1968) in 1968. It is employed to find the optimal path between a start and a goal node in a road map, given a consistent heuristic function. Since it is essential for the following chapter, we will briefly describe it.

Assuming positive costs for travelling along each edge in a graph, A-Star finds the cheapest path from start node $a$ to goal node $b$. An arbitrary node $x$ is evaluated using both the costs of arriving at $x$ from $a$ and a heuristic estimate of the costs from $x$ to $b$. If the heuristic function is consistent, then A-Star is complete and optimal (Dechter & Pearl, 1985). A heuristic $h$ is consistent if and only if

$$h(x, x') \leq c(x, x')$$
and
\[ h(x, b) \leq c(x, x') + h(x', b) \]
where \( c \) is the cost functions, and \( x \) and \( x' \) arbitrary nodes.

### 2.3 Clustering

Cooperative path planning requires consistency at some level of abstraction. That is, at some step of the calculations, resulting data has to be fused throughout the participating robots. This can be at the level of paths or sooner, at the map level. Here we employ a clustering algorithm to fuse obstacle information of all robots. On the basis of the resulting clusters, a Voronoi graph is constructed. Each robot detects some obstacles and cannot perceive others. Moreover, sensor noise is present, so obstacles are not perceived at their true locations. Thus, data representing the same obstacle should be merged, ideally yielding a more precise location than any single robot’s data alone. However, data representing different obstacles should not be merged, in order to prevent the disappearance of perceived obstacles. Network latencies is one of the issues that make this process difficult, as data received from remote robots is generally older than local data and represents past situations. The problem of merging these obstacles is also called data-association problem.

Clustering algorithms merge data in groups or clusters, such that objects in each cluster are as similar as possible and objects of different clusters are as different as possible. (Ester & Sander, 2000). We limit our discussion to the two-dimensional case, where obstacles can be abstracted to points. Every clustering algorithm needs a similarity, or distance measurement \( \text{dist}(p, q) \). For obstacles on a plane, the euclidean distance suffices.

A cluster is a set of points and is represented by a single point, the centroid. A measurement, which describes the divergence of a set of points from its centroid is given by:
\[
TD^2(C) = \sum_{p \in C} \text{dist}(p, \mu_C)^2
\]

\( TD^2(C) \) is also a measurement for the compactness of a cluster. The smaller \( TD^2(C) \), the closer the individual points are to each other, and the smaller the variance within the cluster (Ester & Sander, 2000). \textit{K-MEANS} (MacQueen, 1967) is a common algorithm for minimising the variance of a set of clusters. \textit{K-MEANS} divides the given points into \( K \) clusters, such that \( TD^2 = \sum_{i=1}^{K} TD^2(C_i) \) is minimised. However, it requires prior knowledge of the number of clusters, \( K \).

This disadvantage is lifted by hierarchical clustering algorithms, which represent data in a hierarchical manner to derive clusters (Ester & Sander, 2000). This requires the distance measurement to be extended to set of points. The following three variants of distance measurements are derived from the smallest, largest and average individual distance between points in \( P \) and \( Q \), respectively:

- **Single-Link**: \( dist_{sl}(P, Q) = \min_{p \in P, q \in Q} d(p, q) \)
- **Complete-Link**: \( dist_{cl}(P, Q) = \max_{p \in P, q \in Q} \{d(p, q)\} \)
- **Average-Link**: \( dist_{al}(P, Q) = \frac{1}{|P||Q|} \sum_{p \in P, q \in Q} d(p, q) \)
Another method, based on the work of Ward (1963), uses the difference between the variance before and after fusing two clusters \( P \) and \( Q \).

\[
\text{Increase in variance: } \Delta \sigma^2(P, Q) = TD^2(P \cup Q) - (TD^2(P) + TD^2(Q))
\]

Independent of the distance function, hierarchical clusterings can be described as either dividing (top-down) or collecting (bottom-up) algorithms. Top-down algorithms start with a single cluster containing all points and divide it into subsets until the resulting clusters fulfil a termination criterion. Bottom-up techniques start with single points and fuse them to clusters, again, until a termination criterion is met. The resulting graph is called a dendrogram (see Figure 3).

Figure 3 shows the distances within each cluster versus the number of clusters. Starting at zero, the threshold (dotted line) is increased step-by-step. As soon as the threshold becomes larger than the distance between two clusters, the clusters have to be merged. All clusters which have a larger distance remain separate. In Figure 3, the threshold is set such that three clusters remain.

Other approaches, such as the work by Schubert & Sidenbladh (2005), based on the work of Schubert (1993) and Bengtsson & Schubert (2003), use a Bayesian filter for clustering. Here, points are processed sequentially and no hierarchy is built, instead a set of hypotheses is maintained, out of which the maximum a posteriori estimate is seen as the clustering result. The advantage here is that, due to the sequential nature of the algorithm, observations can be added over time. Thus, it lends itself to algorithms tracking moving obstacles over time (see Section 3.1.2).

### 2.4 Delaunay triangulation

A triangulation is a triangle mesh without overlaps. Figure 5(a) shows an example for a specific triangulation, the Delaunay triangulation. Since this triangulation can be used to derive a Voronoi graph (see Section 2.6), we present some details here.

The Delaunay triangulation of a set of points \( S \) yields the Delaunay graph \( DG(S) \). \( DG(S) \) fulfills the *circumcircle* property: The circumcircle of every triangle does not contain any point of \( S \) which is not also one of the three points belonging to the triangle. Figure 4 shows on the left an enlarged part of the Delaunay graph. The circumcircle does not contain any other point of \( P \). On the right side of Figure 4, the circumcircle property is violated, thus it does not show a Delaunay graph. A triangulation fulfilling this property also maximises the minimal internal angle of all triangles.
2.5 Voronoi graph

The Voronoi graph by Voronoi (1908) is a subdivision of space into so-called Voronoi regions. Figure 5(b) shows an example. We will use Voronoi graphs as road maps (see Section 2.1.1).

In the following, we are limiting ourselves to the two-dimensional case, as we assume robots to be moving on a plane or sufficiently planar surface. Hence, every point is a member of $\mathbb{R}^2$. Every point $p$ in the set $S$ defines a Voronoi-region $V_{orR}(p, S)$:

$$V_{orR}(p, S) = \bigcap_{q \in S \setminus \{p\}} D(p, q)$$

where $D(p, q)$ is a open half-plane:

$$D(p, q) \overset{\text{def}}{=} \{ x \in \mathbb{R}^2 : |p - x| < |q - x| \}$$

The Voronoi graph $\text{Vor}(S)$ is then defined by the set of all regions $V_{orR}(S)$:

$$\text{Vor}(S) \overset{\text{def}}{=} \bigcap_{p \in S} V_{orR}(p, S)$$

$$\text{Vor}(S) \overset{\text{def}}{=} \mathbb{R}^2 \setminus \text{Vor}(S)$$

One of the main properties of the Voronoi graph is that its edges have the maximal distance to the neighbouring points in $S$. Thus following the edges yields itself to obstacle avoidance, given that the obstacles are represented by the points in $S$.

2.6 Duality between Voronoi and Delaunay graph

The Voronoi graph of $S$ is a connected graph $V_{G}(S)$. We will call the set of edges of $V_{G}(S)$ $E_{V_{G}}$, the set of nodes or vertices $V_{V_{G}}$, and the set of regions $R_{V_{G}}$. Analogously, the Delaunay graph $D_{G}(S)$ has the edges $E_{D_{G}}$, vertices $V_{D_{G}}$, and the set of regions $R_{D_{G}}$. Figure 5(b) shows that some edges lead outside of the picture. Assuming the existence of another point in infinite distance, one can connect all these edges to said point, without losing any Voronoi property (de Berg et al., 2000). Figure 6 illustrates this.

Voronoi graph and Delaunay graph are dual to each other, assuming the existence of the point in infinite distance. Thus, there is an isomorphism between the vertices of one graph and the regions of the other. In other words, every point in $S$ yields a vertex in $V_{D_{G}}$ and a region in $R_{V_{G}}$.

$$V_{D_{G}} \leftrightarrow F_{V_{G}}$$
Moreover, every circumcentre of a region in $R_{DG}$ is also a vertex in $V_{VG}$. The infinitely large area encompassing the triangulation represents the node in infinite distance.

\[ F_{DG} \leftrightarrow V_{VG} \]

Finally, there is an isomorphism between the edges of both graphs. For every edge in one graph, there is one in the other at a right angle. However, they do not necessarily overlap.

\[ E_{DG} \leftrightarrow E_{VG} \]

Figure 5(c) illustrates the dual relationship between the graphs. This duality can be exploited to construct a Voronoi graph by Delaunay triangulation, as there exist efficient algorithms to construct Delaunay graphs (Bhattacharya & Gavrilova, 2007; Guibas et al., 1990; Leach, 1992).

### 3. Approach

Our approach is divided into two main parts, data fusion and planning. Figure 7 depicts an overview of the architecture. It is meant to be used in conjunction with other frameworks or engines, which manage other parts of the robotic architecture. For instance, we assume that a behaviour issues a command to move to a certain point. What a behaviour constitutes and
how it comes to a decision is intentionally left open. A treatment of multi-robot behaviour control and cooperation is given in Skubch et al. (2011). Furthermore, we assume that the cooperative robots are able to communicate, for instance following an approach as described in Baer (2008).

During the data fusion step, observations from the sensory subsystem, such as image processing, is integrated with observations received from other cooperative robots. This includes the truthfully emitted positions and velocities of these cooperative robots. The data is validated and clustered, yielding a set of adversarial and a set of cooperative robots. Positions and velocities of adversarial robots are tracked using a Bayesian filter, while positions and velocities of cooperative robots are assumed to be correct. This is due to the fact that each robot’s own localisation and velocity estimate is in most cases more accurate than observations of the same information from other sources. Finally, both sets are used to generate a Voronoi Graph of the current scene.

Based on the Voronoi Graph and the current intended destination given by a behaviour, a path is planned by the robot in question. Hence, each robot calculates its preferred path individually. Thus, path planning is done decoupled. This planning step is both efficient and optimal with respect to the roadmap. Since the path planning does not account for moving obstacles, these are reacted to in a final step. Tackling this problem, we use the cooperative and reactive method for collision avoidance introduced by Fujimori et al. (2000). If collisions between cooperative robots become imminent, the current driving directions are adapted in speed and angle according to globally defined rules. In order to avoid other moving obstacles similar rules are applied, taking their potential adverseness into account. The resulting driving command is given to the actuator subsystem. The whole process is repeated every 30ms.

The next section explains the data fusion step in detail, while Section 3.2 describes both path planning and reactive behaviour.
3.1 Data fusion

Sensor fusion is a wide topic, of which we only discuss parts that are relevant for our path-planning approach. A more in-depth treatment can be found in Reichle (2010). In our setting we assume that a set of known robots \( C \) cooperate. Each robot \( r \in C \) emits its known position \( p \), velocity \( v \) and a set of perceived obstacles \( o \) every 100ms. The obstacle set is noisy, contains false positives and does not necessarily contain all obstacles in line-of-sight. Furthermore, there exists an unknown, but bounded, number of moving adversarial robots, \( A \), which do not communicate at all.

3.1.1 Clustering

During the clustering step each robot fuses its own position and its perceived obstacles with the positions and obstacles of all cooperative robots. The deployed clustering is based on the method according to Ward (1963), described in Section 2.3. Furthermore the following adaptations take the age, type and source of the information into account.

The precision of perceived obstacle positions reduces with increasing age and distance. Due to potential network latencies and the lower frequency of 10Hz, the information given by cooperative robots is older than the local information. Therefore, near obstacle positions perceived by the local robot should be treated preferential to the obstacle positions of the cooperative robots. In our approach the clustering is omitted in a 1.5m radius around the local robot. Inside this radius only locally perceived obstacles are taken into account.

Another important adaptation is to distinguish between positions of cooperative robots and adversarial robots. If a cooperative robot is merged into a cluster, the cluster itself represents the cooperative robot. Therefore, it is important to avoid inconsistent clusters, which can be created by merging two different cooperative robots into one cluster. Another source of inconsistency is the possibility of merging an obstacle’s position into the cluster of the cooperative robot which sent the obstacle position itself, because cooperative robots cannot see themselves as obstacles.

At the end of the clustering step, we are able to distinguish the clusters into the two groups of cooperative robots and adversarial robots. As mentioned before the received velocities of the cooperative robots are considered quite precise, therefore only the adversarial robot clusters are given to the object tracker, described in Section 3.1.2.

3.1.2 Tracking

Tracking is the process of relating observations over time. This process not only smooths the data, removes infrequent false positives and compensates for missing observations, but it also estimates velocities. In Section 2, we briefly mentioned the applicability of Bayesian filters to the tracking and clustering problem. Recently, van de Molengraft (2009) introduced a dynamic tracking algorithm for an unknown number of objects based on the work of Schubert & Sidenbladh (2005). Schubert’s approach maintains a set of hypotheses of cluster centres. For each new observation, every hypothesis \( h \) is expanded to \( n + 2 \) to new ones, where \( n \) is the number of cluster centres maintained in \( h \). That is, given \( h \), the new observation is considered to be faulty data, belonging to a new cluster, or belonging to one of the clusters maintained in \( h \). Schubert then estimates the a posteriori probability of all newly generated hypotheses, resamples them using Monte Carlo sampling and merges highly similar hypotheses. Due to the resampling with a constant number of particles, the amount of hypotheses does not grow beyond a constant threshold.

Schubert’s method was originally only applicable to non-moving targets. Van de Molengraft (2009) extended this approach by maintaining a Kalman Filter for each cluster, using the
constant velocity assumption. Furthermore, they introduced a lower threshold, independent of the Monte Carlo resampling to the number of maintained hypotheses.

Our approach differs from the aforementioned in that we exchanged the Kalman Filters for a linear ridge regression, which proved in practice to require less precise parameter optimisation. Kalman Filters with a constant velocity assumption tend to produce estimations which are either too susceptible to noise or too inert to changing conditions if the probability parameters do not match the scenario (Lauer, 2007). Furthermore, we limited the maximum amount of tracked objects, such that the runtime complexity is now in \( O(n) \), where \( n \) is the number of observations, instead of \( O(nk) \), where \( k \) is the number of objects tracked. Thus, we assume a constraint number of objects instead of an unknown one. Finally, we omit the Monte Carlo sampling and directly use the likelihood as an estimation for the a posteriori probability, further reducing the computation time.

The downside of using a regression as opposed to a Kalman Filter is the lack of a probability estimate, which the Kalman Filter produces directly. As Schubert, we assume Gaussian observation noise and estimate the probability of an observation \( z \) at time \( t \) caused by object \( o \) as:

\[
P(z_t | o) \propto e^{-\frac{(z - m_o(t))^2}{2\sigma^2}}
\]

where \( m_o(t) \) is the estimated position of object \( o \) at time \( t \).

The Monte Carlo sampling served the purpose of avoiding too fast convergence. However, since in dynamic environments the input data changes rapidly over time and the fact that similar hypotheses are collapsed into a single one, early convergence is not a problem in our approach, thus the Monte Carlo sampling can be omitted, saving computation time. This yields an algorithm that integrates 10 observations into 10 tracked objects using 30 hypotheses in under 5ms on current PCs (Intel i7 2.8GHz, single core).

### 3.1.3 Voronoi construction

For generating a Voronoi graph, we use the same method as Bhattacharya & Gavrilova (2007). At first, the Delaunay triangulation is created by the randomised incremental algorithm introduced by Guibas et al. (1990) and afterwards the Voronoi graph is derived from the Delaunay triangulation. Obviously, the fused data serves as input for this process, but we also insert some artificial obstacles along the side lines of the football field. The football field, being the containing rectangle of the whole scenario, is sampled similar to the sampling method of Bhattacharya & Gavrilova (2007). This yields Voronoi edges between obstacles inside the containing rectangle and the borders of that rectangle. Otherwise, the roadmap would miss edges for driving around obstacles at the sideline of the football field. In the resulting Voronoi graph each robot, including the local one, represents one Voronoi cell. In other scenarios, similar bounding boxes can be sampled this way. Alternatively, in case the bounds are unknown or unlimited, a sufficiently large bounding circle can be used, containing the current focus of interest for the robot.

### 3.2 Path planning

As mentioned before the path planning is divided into two clearly separated steps. The first step is described in Section 3.2.1, it searches for the best path according to the given destination and Voronoi graph. The second step comprises the adjustment of the initial driving direction. The initial driving direction is derived from the path found in the first step. To adjust the driving direction globally defined rules are used. These details are described in Section 3.2.2.

#### 3.2.1 Path generation

The Voronoi graph and the destination, given by a behaviour, form the input of the path search algorithm. We utilize the A-Star search algorithm, described in Section 2.2. Considering the
common prominence of the A-Star search algorithm, here we only describe the used heuristics and the special cases at the start and end of the path. Besides the start and the end of a path, each path is a sequence of Voronoi edges. Therefore, the applied heuristics have to estimate the costs of a sequence of Voronoi edges.

Let $p$ denote a path with edges $E = e_0, \ldots, e_n$, and for every edge $e$, let $e.s$ denote its starting node, $e.e$ its ending node, such that $e_i.e = e_{i+1}.s$, $e_0.s$ the robot’s current position, and $e_n.e$ the destination $D$ if $p$ is complete. Further, let $l(e)$ be the length of edge $e$, $w(e)$ its width, $\Delta\alpha(a, b)$ the angle between edges or vectors $a$ and $b$, and $\vec{g}_i$ be the calculated driving direction in the robot’s $i$-th cycle.

Given additionally the robot’s radius $r$, the cost function $c(p)$ and the heuristic function $h(p)$ for a path $p$ calculated in cycle $i$ are then computed in the following way:

$$c(p) = c_{\text{dist}}(p) + w_w c_w(p) + w_\alpha c_\alpha(p) + w_s c_s(p)$$

with

$$c_{\text{dist}}(p) = \sum_{i=0}^{n} l(e_i)$$

$$c_w(p) = \sum_{i=0}^{n} \frac{1\text{mm}}{\max(1\text{mm}, w(e_i) - r)}$$

$$c_\alpha(p) = \sum_{i=1}^{n} (\Delta\alpha(e_i, e_{i-1})^2$$

$$c_s(p) = \Delta\alpha(e_0, \vec{g}_{i-1})$$

where all $w_x$ are constant weights. Thereby, the cost function is a weighted sum of four factors. Firstly, the length of the path, the longer a path, the less valuable it is. Secondly, a function depending on the width of the occurring edges, as small gaps should be penalised as opposed to wide openings. The third factor sums up the square angle difference between adjacent edges, as sharp turns are less preferably than wide turns, which can be driven faster. Finally, the last summand depends on the previously calculated driving direction. It stabilises the decision of the robot under sensor noise, as it penalises larger changes in the driving direction over time.

The weights depend on the specific scenario and capabilities of the robot. For different manoeuvres the weights might even be set differently. For instance in RoboCup, a robot is less agile if it dribbles, as it has to maintain control of the ball, thus it would greatly prefer paths without sharp turns.

Each summand gives rise to a summand in the heuristic function for path, $h(p)$. The heuristic function we use is defined by:

$$h(p) = \begin{cases} 
0 & \text{if } e_n.e = D \\
\text{dist}(e_n, D) + w_\alpha \Delta\alpha(e_n, eD)^2 + w_w \max_{e \in E_{VG}} \frac{1\text{mm}}{\max(1\text{mm}, w(e) - r)} & \text{otherwise}
\end{cases}$$

The path length to the destination is approximated by the linear distance between the last node in the path and the destination. The costs due to angle differences are approximated by the angle between the last edge and the line connecting the last node and the destination, as this is the very least the robot will need to turn to arrive at the destination. The stabilisation summand only factors in the first node and thus it’s heuristic is 0. The width costs are
approximated by the maximal width of any edge in the graph. This is a weak heuristic and could be improved by limiting the set of considered edges, however this introduces more overhead than it would save expansion steps in the search, unless the width related costs are dominating.

The width of a Voronoi edge is determined by its distance to an adjacent cell defining object. The width of a starting edge, leading from the robot’s position to a Voronoi node equals the minimum distance to any object spanning a neighbouring Voronoi cell. The width of a final edge, leading to the destination, depends only on the position of the object defining the cell which contains the destination. If the destination is on a Voronoi edge, the final edge is part of this Voronoi edge, thus has equal width.

Figure 8 shows a robot planning to drive to the right goal. The resulting path is shown, with start and end edges emphasised. As each robot is inside its own convex Voronoi cell the path search can be initialized with one path for each Voronoi node of the Voronoi cell. If no initial Voronoi node is reachable, the robot is surrounded by obstacles and cannot drive anywhere. The drawback of this initializing method is that it creates unnecessary complicated paths to destinations which could be reached directly, without approaching a Voronoi node. Comparing the sequences of Voronoi edges with the direct line to the destination prevents unnecessary complicated paths.

The destination is either inside a Voronoi cell, or on the edge of a cell. If a path is expanded during the search to one of the Voronoi nodes of the cell, which includes the destination, we also expand the path to the destination if possible. This is done in a similar way as the path search was initialized. If the destination is blocked by an obstacle the path stops as near as the robot can get to the destination. It is plausible, that an obstacle overlaps with the destination,
so that it is impossible to reach the destination. In this case, the path which ends closest to the
destination is returned by the search.
The driving direction according to the resulting path is not necessarily the direction of the first
edge. Just as we test to drive directly to the given destination, we also test if it is possible to
reach any of the Voronoi nodes on the path directly. Starting at the second to the last Voronoi
node on the path, each point is considered as short cut on the path. Only if no Voronoi node
serves as short cut, the driving direction corresponds to the first path edge.

### 3.2.2 Reactive collision avoidance behaviour

The collision avoidance behaviour is based on the work of Fujimori et al. (2000) and uses
globally announced rules to make additional communication between the robots unnecessary.
The needed data and its computation is described in Section 3.1.
A collision between two robots is imminent if they are near and further approaching each
other. In this case, the expected minimal distance between the two robots is computed.
Furthermore, if the driving directions intersect, the angle between them, \( \delta \) is noted. As
described by Fujimori et al., this intersection angle \( \delta \) is very important. In Figure 9(a) the
collision can be avoided only if both robots change their directions. The robots in Figure 9(c)
would have to change their directions much more than the in Figure 9(a). Therefore, changing
their velocities would be much more effective. Figure 9(e) shows a situation requiring both,
changing velocity and direction. For example, Robot 1 could pass Robot 2 on its right side by
changing the direction and accelerating appropriately. Meanwhile Robot 2 should reduce its
velocity to finish the procedure faster. So there are different kinds of collisions and each of
them needs an adjustment of directions, velocities, or both.

Given the current situation, each robot is assigned a priority to coordinate their collision
avoidance behaviour. For example, the decision which robot should accelerate and which
should decelerate is made according to the robot’s priorities. A high prioritized robot should
arrive quickly at its destination and has the right of way. Lower prioritized robots should
clear the way for higher prioritized robots. Therefore, in Figure 9(c) the robot with the higher
priority will accelerate and the other one will decelerate. The priority \( \omega \) of robot \( i \) towards
robot \( j \) is defined as follows:

\[
\omega_{ij} = \begin{cases} 
\frac{v_i}{k, \|v_i\|}, & \text{if intersection point } P_{ij} \text{ exists,} \\
 v_i, & \text{if no intersection point exists and } v_i \neq v_j 
\end{cases}
\]  

In the first case the higher priority is assigned to the robot which first arrives at the intersection
point. The latter case is applied if the driving directions are parallel to each other and a
collision as shown in Figure 9(a) is imminent. Here the higher priority is assigned to the
faster robot. If the priorities \( \omega_{ij} \) and \( \omega_{ji} \) are the same, the robot’s ID decides.

According to the priorities and the angle of collision the velocities and directions can be
calculated. The velocity adjustment \( \Delta v \) of robot \( i \) to avoid a collision with robot \( j \) is defined as follows:

\[
\Delta v_{ij} = \begin{cases} 
sat(|\delta|, \frac{\pi}{2}, \pi, v_{acc}, 0), & \omega_{ij} > \omega_{ji}, \\
sat(|\delta|, \frac{\pi}{2}, \pi, v_{dec}, 0), & \omega_{ij} < \omega_{ji} 
\end{cases}
\]  

The corresponding direction adjustment \( \Delta \gamma \) is defined by:

\[
\Delta \gamma_{ij} = \begin{cases} 
k_\gamma \text{sgn}(\delta) \frac{1 - \frac{\pi}{2} |\delta|}{|\delta|}, & \omega_{ij} > \omega_{ji}, \\
\text{sgn}(\delta) \text{sat}(|\delta|, \frac{\pi}{2}, \pi, 0, k_\gamma) & \omega_{ij} < \omega_{ji} 
\end{cases}
\]
Fig. 9. Different Angles of Collision

where $\text{sgn}(x)$ and $\text{sat}(x, a, b, c, d)$ are defined as:

$$\text{sgn}(x) = \begin{cases} 
1 & x \geq 0 \\
-1 & x < 0
\end{cases}$$

$$\text{sat}(x, a, b, c, d) = \begin{cases} 
\frac{d - c}{b - a} (x - a) + c & x > a, \\
\frac{c - d}{b - a} (x - a) + d & a \leq x < b, \\
d & x \geq b
\end{cases}$$

Figure 10 shows the functions to calculate the adjustment of velocity and driving direction. Index H means high and L means low priority. The amount of adjustment to the driving direction can be defined through the parameter $k_\gamma$. In the opposite of Fujimori et al., we do not assign a fix value to $k_\gamma$. Instead, $k_\gamma$ depends on the remaining time to collision (ttc):

$$k_\gamma = \min(\eta, \frac{0.3 \text{ [sec]}}{\text{ttc [sec]}})$$

If ttc is less than 0.3 seconds, which corresponds to 10 iterations of the robots calculation loop, the adjustment to the driving angle is set to $\eta$.

The amount of velocity change can be set by the parameters $v_{\text{acc}}$ and $v_{\text{dec}}$. At most a higher prioritized robot accelerates about $v_{\text{acc}}$, while a lower prioritized robot decelerates about $v_{\text{dec}}$. Instead of the differential velocity, Fujimori et al. calculates the absolute velocity of the robots, which is less reactive concerning the situation. If the angle of collision $\delta$ is exactly $\pi$, only
the driving directions are affected and the velocities stay unchanged. If the angle of collision \( \delta \) is exactly \( \frac{\pi}{2} \), only the velocities are affected and the driving directions stay unchanged. At the range of \( \frac{\pi}{2} \leq \delta \leq \pi \) the change to the velocities and driving directions are linearly interpolated. At the range of \( 0 \leq \delta \leq \frac{\pi}{2} \) the velocity changes of both robots stay at their maximum, but the change of the driving direction of the higher prioritized robot increases to its maximum again. This corresponds to the situation shown in Figure 9(e).

![Fig. 10. Change of direction and velocity](image)

In case of collisions with more than two robots, the overall change of direction and velocity of robot \( i \) are given by the weighted sum of changes of each collision:

\[
\Delta \gamma_i = \frac{\sum_{j=1}^{n} \frac{1}{t_{tcij}} \Delta \gamma_{ij}}{\sum_{j=1}^{n} \frac{1}{t_{tcij}}} \\
\Delta v_i = \frac{\sum_{j=1}^{n} \frac{1}{t_{tcij}} \Delta v_{ij}}{\sum_{j=1}^{n} \frac{1}{t_{tcij}}}
\]

Due to the fact that each change is weighted by the corresponding inverse of ttc, earlier collisions have more influence than later collisions.

Adapting the introduced collision avoidance technique to a specific team of robots, the values of the mentioned parameters have to be chosen appropriately. These parameters include the maximum distance between two robots for collision avoidance to apply, as well as a minimum velocity, below which no collision avoidance is considered. Altogether, there are two activation parameters: the minimum velocity two robots approach each other and the distance between them. How much the collision avoidance behaviour changes the calculated driving direction and velocity of a robot depends on the parameters \( \eta \), \( v_{acc} \) and \( v_{dec} \). These three parameters should be chosen as large as necessary and as small as possible to avoid collisions with as little effort as possible. The evaluation of the approach of Fujimori et al. includes simulator tests as well as tests on real robots. According to the results a collision is avoided in most cases and problems arise if the robots have to move in confined spaces. Our evaluation follows in the next section.

4. Discussion

Although our architecture may appear to be overly complex at first glance, each part, shown in Figure 7 of Section 3, has clearly defined interfaces and makes an essential contribution to the overall performance. Furthermore, the modular design enables exchangeability of each
module and increases the reusability of our architecture. Each module comes with its own set of parameters, hence adapting our architecture to other types of scenarios might raise suspicions that tuning these parameters is a lot of work. However, the modules described are very robust against small parameter changes, such that rough adjustment of the parameters already produces good results. Although a centralised approach can produce better results, due to such an approach avoiding additional coordination problems, it also has a single point of failure, which cannot be tolerated in many domains. Our combination of different algorithms with efficiency in mind yields a real-time path planning architecture, which even provides the possibility to plan paths of other cooperative robots. In the following sections, we discuss the main advantages of each module.

4.1 Clustering
We evaluated the clustering algorithm using two cooperative robots. One is located at the centre of the football field recording its own data together with the data received from the other robot. The other robot is controlled and sends its own data to the robot in the middle. The evaluation is based on the recorded data of the robot located at the centre. The data includes 5 minutes with a recording frequency of 30Hz for the local data and 10Hz for the data received from the remote controlled robot. The recorded data is subject to the conditions mentioned in Section 1.

Clustering is subject to two kinds of errors. Either data describing the same obstacle is not fused (first case), or data describing different obstacles is fused nevertheless (second case). The frequency of occurrence of both cases depends directly on the variance threshold. The variance threshold defines the maximum variance of a cluster. The bigger the variance threshold is the bigger the distance between fused data can be. In other words, the occurrences of the first case errors increase and the occurrences of the second case errors decrease. Reducing the variance threshold inversely influences the two cases of errors. Determining the occurrences of errors near the controlled robot the second case of errors is negligible, because from the recording robot’s point of view there is only the controlled robot on the football field. Furthermore, obstacle positions received from another robot will not be fused with the position of this robot, because the robot cannot see itself. The frequency of occurrence against the variance threshold is shown in Figure 11.

![Figure 11. Relation of error frequency against variance threshold](image-url)
In Figure 11 a small part of the graph is shown magnified. The emphasised point (40000, 0.12) indicates that an obstacle position must not exceed the standard deviation of $\sqrt{40000\text{mm}^2} = 20\text{cm}$ to its cluster centre. In this case, 12 percent of the clustered situations contain a first case error. Assuming a robot radius of 25cm a variance threshold of $40000\text{mm}^2$ means that two robots have to be perceived overlapping by 10cm to be erroneously fused. Moreover, a local observation of a robot, moving with a relative speed of over 9m/s with respect to the local robot, might not be fused with its broadcast position, given the average age of 50ms of any remotely received information. A relative speed of 9m/s is reached if two robots move in opposite direction with 4.5m/s, which is common in RoboCup Middle Size League games. Both statements hold only in case of noiseless sensor information and without network latencies. Factoring in sensor noise, erroneous fusing occurs earlier. However, in most domains robots hardly move faster than 4.5m/s, thus high velocities are usually not a cause for badly clustered data. Therefore, we can state that the clustering with a variance threshold of $40000\text{mm}^2$ is fairly robust against high robot velocities. Figure 12(a) further underlines this, as the error is independent of the relative velocity for small values of said velocity.

![Graph](image)

(a) velocity relation

![Graph](image)

(b) distance relation

Fig. 12. Relation of error frequency against velocity and distance

Imprecision introduced by sensors noise is much more influential. Figure 12(b) shows a correlation between distance and error frequency. This correlation can be explained by the fact, that the feature extraction algorithm of the robot is much more imprecise in estimating the position of robots, which are further away than of robots which are closer by. The small number of error occurrences at a distance above 5m arose, because the feature extraction algorithm often did not recognize the remote controlled robot at these distances. Summarized, the clustering is robust against the dynamics of a RoboCup Middle Size League game, hence it unifies the data of all cooperative robots to the same single point of view on all cooperative robots. The simple clustering algorithm used here is much more time efficient than the Bayesian filter used for tracking over time. Thus, using the clustered data as input for the tracking algorithm instead of using the raw observations reduces the latency of the whole computation.

4.2 Tracking

In this Section, we briefly demonstrate the performance of the tracking algorithm discussed in Section 3.1.2. In the following simulated experiment, 10 objects moved with a randomly
chosen constant velocity between 1 and 4m/s on a field of $20m \times 20m$ size. In case they collided with the boundaries, they bounced back perfectly. The tracker was set up with a maximum hypotheses number of 30, easily allowing it to process 10 observations in less than 5ms. Each test was performed for 500 iterations.

In each iteration, 10 observations were given to the tracker, each with a probability $p = 0.1$ uniformly distributed over the field, else normally distributed around an object. Figure 13 shows the resulting tracking errors, against an increasing amount of observation noise. Note that the tracking algorithm assumes a constant deviation of 0.25m in all experiments.

One can see that position and velocity errors increase linearly with increasing deviation, while the error in the number of objects increases rapidly for deviations higher than the assumed 0.25m. At the expected deviation of 0.25m, an average of 0.5 objects are wrongly tracked. Note that this also includes the start up phase, in which the tracker gradually increases the amount of objects tracked. The average position error is 0.3m and the average velocity error is 1.2m/s. Large velocity errors mostly occur after an object has bounced off the boundary of the field, as this clearly violates the constant velocity assumption.

Figure 14 shows the same setup, where the objects additionally have a rotational speed between $10^\circ /s$ and $90^\circ /s$, and thus move on circles. As expected, the velocity error is in general higher than in the case of a constant velocity. Moreover, the number of tracking errors increases earlier, as observations not fitting the model are more common. Interestingly, the position error is only marginally affected.

### 4.3 Path generation

Like every path planning approach using some kind of abstraction, such as a roadmap, the paths generated by our architecture are not optimal. Approaches trying to generate optimal paths are faced with a PSPACE-hard problem (see Section 2.1) and as a result these approaches are not feasible in domains similar to the one addressed by our architecture.

Avoiding stationary obstacles can easily be done using a Voronoi graph as roadmap. A robot following the edges of the Voronoi graph is as far away from the surrounding obstacles as possible. A path planning algorithm using this graph can easily adapt to the current
tracking performance with objects moving on circles situation, using differently weighted cost functions, thus allowing robots to switch between an aggressive behaviour, using short, narrow paths and a safe behaviour using longer and wider paths, for instance. Moreover, a Voronoi graph can be used to answer various spatial queries other than path planning, such as finding free areas or the robot closest to a given point. Bhattacharya & Gavrilova (2007) demonstrated that it is possible to calculate Voronoi graphs to approximate arbitrary environments.

4.4 Path correction
The reactive nature of our collision avoidance behaviour forms a contrast to the forward planning of the path planning algorithm and it might be more intuitive to avoid collisions along the whole path. However, in order to avoid collisions along the whole path obstacle positions, including adversarial robots, need to be predicted for time periods of up to $5 - 10s$. Predicting the dynamic system that is formed by multiple cooperative and adversarial robots for more than 200ms is completely infeasible. Thus, collisions need to be handled reactively without relying on comparatively long-term predictions. Due to the globally known avoidance rules, there is no need for an additional communication effort. Furthermore, the discussed rules scale linearly with the number of robots involved. Similar to the adaptivity of the planning algorithm, the parameters of the collision avoidance allows to switch between different behaviours during runtime, using different grades of sensitivity. Summarized, the collision avoidance technique is applicable to any number and any kind of autonomously driving robots.

5. Conclusion
The described architecture for cooperative real-time path planning and collision avoidance, consisting of various adapted algorithms, was evaluated on robots of the Carpe Noctem RoboCup team of the University of Kassel and used in tournaments since 2011. The architecture demonstrates that it is possible to use a decentralised path-planning approach combined with a conflict resolution inspired by the reactive behaviour approach. The only
requirement is to fuse and unify the information among the group of cooperative robots, so that they come to similar decisions. Fused and unified information enable path planning beyond the sensor range of single robots up to the sensor range of the team, without any additional communication for conflict resolution. Furthermore, the complete calculation loop is real-time capable. More precisely, the complete recognition, fusion and path planning cycle can be done at a frequency of 30Hz. Furthermore, the architecture meets the requirements of the highly dynamic RoboCup domain of the Middle Size League described in Section 1. The stability and similarity of fused information and calculated paths among the group of cooperative robots can be analysed over time and compared with other path planning and sensor fusion approaches in future.

6. References


