Improved Deadlock Detection and Detours

An extension for MIRAN

November 28, 2018

Marjolein Zwerver
ICA-5595363

Game and Media Technology
Utrect University, The Netherlands
Supervisors: dr. R. J. Geraerts and W. G. van Toll

Universiteit Utrecht

Abstract

Crowd simulation is an important area of study since it is a broadly used subject: from the simulation of crowds in games to increase immersion, to simulations to improve the flow of people during an evacuation. As technology advances it is possible to simulate more and more agents in real time. This also increases interest in the simulation of high-density crowds. Even though there are many methods that are able to simulate high-density crowds there are still some unsolved problems. Examples are the forming of deadlocks for a variety of reasons such as the lack of lane forming or underutilization of available space.

The aim of this thesis is to create a method that solves some of these known issues to create more time-efficient paths for agents. For this purpose, this thesis introduces the Improved Deadlock Detection and Detours (ID^3) algorithm as an extension of the MIRAN algorithm used for navigation. ID^3 improves on MIRAN in two ways: by introducing deadlock detection and by planning detours using a density-based method based on the detours planned by the MIRANDA algorithm. The deadlock detection is vision-based and accounts for the flow of surrounding agents. The detours are improved by determining the detour goal by sampling density values and detecting what type of detour is planned. For global detours, paths are repaired using gates and memory is added to prevent continuous switching between different paths.

The experiments show, in most cases, that there was an improvement regarding the time it takes for an agent to reach their goal. In cases where there was no improvement on the time, there was often an improvement on either the realism or length of the path.

Contents

1 Introduction

In Section 1.1, we will start with discussing the importance of crowd simulation. In Section 1.2, we will discuss density-based crowd simulation and issues with current methods. In Section 1.3, we will discuss our research goals and contributions and in Section 1.4 we explain the structure of this thesis.

1.1 Importance of crowd simulation

Crowd simulation is a broad topic of research that has many different applications. From a business aspect, crowd simulation research can be used to improve the flow of people in busy places such as shopping malls or airports [22] which can improve customer satisfaction and income.

Crowd simulation research also has societal importance: research into evacuation dynamics and crowd disasters can save lives. The goal of the analysis of evacuation dynamics is to simulate an evacuation situation as realistically as possible and to try to find what measures can be taken to ensure people's safety [14, 25, 13]. It is even used for virtual reality training systems for urban emergencies [36]. Analysis of crowd disasters is done by Helbing et al. [16]. In their research, the Love parade disaster is analyzed and recommendations to prevent future crowd disasters are given.

In terms of entertainment, crowd simulation also plays a small role: algorithms used in games and movies improve the realism and immersion [1].

In addition, ongoing research on the topic of crowd simulation is important as improvements on hardware and research into working on GPU hardware [27] allow for more and more realistic simulations to be performed in real-time.

1.2 Density-based crowd simulation

As the number of agents in simulations increases, agents who plan without any knowledge of others will no longer produce realistic looking paths, moreover, the computational load might get too high for interactive applications. One possibility is to use similar computations for groups of agents to reduce computation times [4, 34, 35]. The downside of algorithms utilizing this method is the lack of individual behavior they produce, making the simulation less realistic. Even algorithms that do simulate individual behavior in high-density scenarios do not always provide realistic behavior: agents may get stuck in deadlocks or collide with each other. We will use the same definition of a dense crowd as is used in [30] by Stüvel who uses an average of at least 3 humans per square meter to define dense maneuvering.

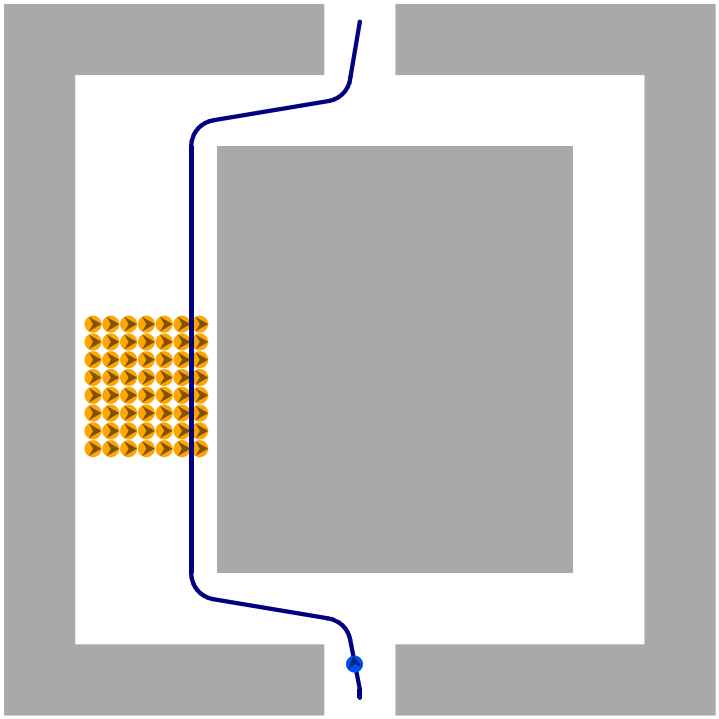


Figure 1: An agent takes the shortest path where a slightly longer path would be traversed faster

1.2.1 Issues with current methods

A lot of issues with current crowd simulation methods can be attributed to the fact that agents plan without any knowledge of other agents. In high-density crowds, this leads to different types of problems. In general, these can be divided into global and local issues. Global issues have to do with the fact that a general path can lead an agent through locations that contain a lot of stationary agents blocking a path, while there is a slightly longer path without agents that would result in a route that takes less time to traverse (Fig. 1).

Local issues arise because an agent ignores the flow of agents around it. When many agents move in the same general direction, they might benefit from aligning their velocities in order to avoid collisions. Another problem with flow arises in a corridor with two opposing flows: if agents don't form lanes, they will get stuck in the middle and get into a deadlock as is shown in Fig. 2. The second local issue is the underutilization of available space, when for example agents turn around a corner. As can be seen in Fig. 3, all agents take the shortest path resulting in overlapping routes where the better option would be to utilize the space around the corner to avoid this.

Another issue is that an agent sometimes will have no other choice than to move through a crowded space. This can happen when the only path to the goal is blocked by a deadlock. Even though these deadlocks should not

occur in realistic situations, there are other situations such as a crowded bar where moving through a dense crowd is behavior that should be implemented. In only a few papers, such as the one by Stüvel et al. [32], these kinds of behaviors are mentioned and handled.

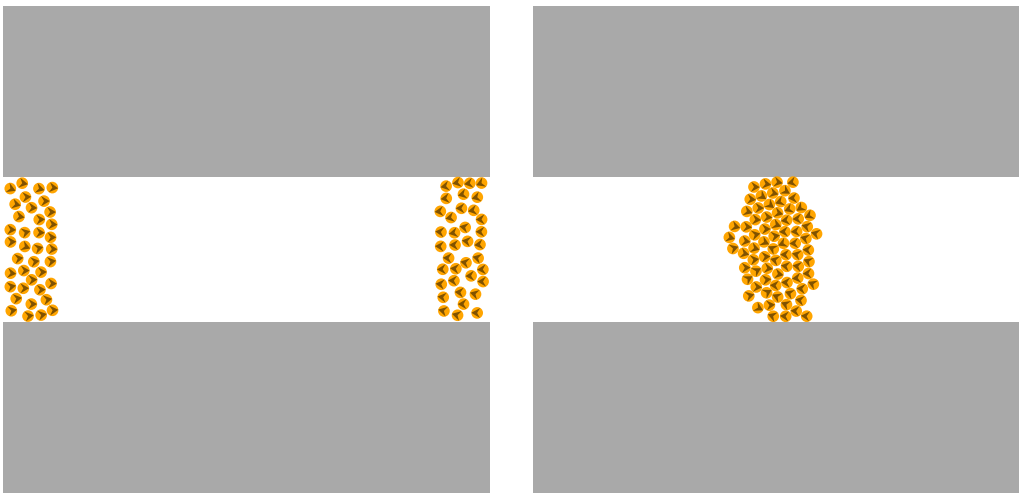


Figure 2: Two opposing groups may form a deadlock in the center of the corridor

1.3 Research goals and contributions

In this thesis, we seek to improve on paths created by current methods when simulating high-density crowds. To do this, our aim is to solve some of the global and local issues mentioned above, and, in doing so, create paths that avoid congestions and deadlocks. We will try to create better and more realistic paths and use the time it takes for an agent to traverse a path as a measure of this. The main reason for this is that when there is a short path blocked by a slow or stationary group of agents we want the algorithm to prefer longer paths if they can be traversed faster. In conclusion, the main focus is to improve the arrival time of an agent by avoiding places with many agents, which avoids slowing the agent down due to collisions. This leads us to the first research question:

How can we improve current methods to create time-efficient paths in high-density crowds?

For our attempt to solve the aforementioned issues, we limit the scope of situations for which we try to solve the problem. The problem scope is specified with the following assumptions:

- A solution exists such that it is possible for all agents to reach their goal without global coordination

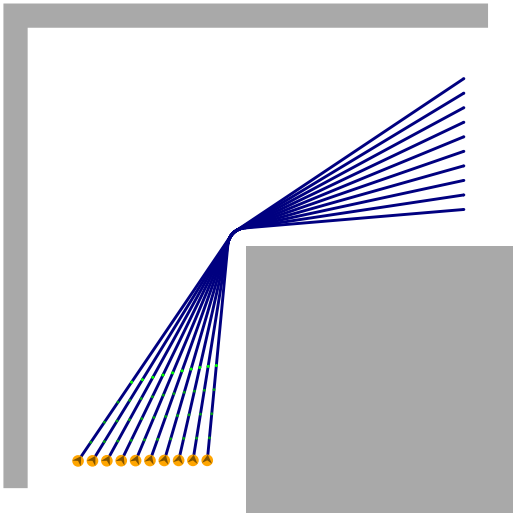


Figure 3: Underutilization of space around a corner

- Agents have full knowledge of the geometry of the environment
- The environment is static and two-dimensional
- A start and goal position for an agent's path are given
- Agents have global knowledge of all other agents' positions and velocities

To answer the main question we formulated the following subquestions:

- What factors are important when calculating the cost of a path?
- What method can best be used to determine when a path needs to be replanned?
- What methods can best be used to replan a path?

This thesis seeks to create a method improving on current problems that similar methods have and in doing so improve behavior for high-density scenarios.

1.4 Document structure

The rest of this document is organized as follows. In Section 2, we start summarizing related work regarding path planning and crowd simulation. Then we will go into more detail by discussing flow-based methods, density-based methods and methods used for corner turning. In Section 3, we will discuss concepts and algorithms necessary for understanding our own algorithm such

as the medial axis and the MIRAN algorithm. In Section 4, we will explain our own algorithm: ID^3. In Section 5, we show the experiments we have conducted and the results. Then, in Section 6, we will conclude that our algorithm meets our research goals, and, in Section 7, we will discuss our ideas regarding possible future research.

2 Related work

In this section, we will give an overview of the current research on high-density crowd simulation. We will start with a broad overview of path planning and crowd simulation methods and then shift to more specific density-based methods. For a more general overview on crowd simulation we would like to refer the reader to the books by Pelechano et al. [26] and Thalmann and Musse [33].

2.1 Path planning

One of the first algorithms for path planning is Dijkstra's algorithm [5]. This algorithm finds the shortest path between two nodes in a graph. An algorithm very similar to Dijkstra's algorithm was developed later: A* [10]. This algorithm uses a given heuristic function to decrease computation times on the shortest path. Using A* on a grid is often used for path planning since the shortest path on the grid is guaranteed to be found if it exists, but it also has some serious drawbacks [8]. The resulting paths need smoothing, resulting in an increased computational load. Also, dynamic obstacles or other agents are not taken into account.

A method that handles single query path planning is the RRT-connect method [23] by Kuffner et al. This method builds rapidly-exploring random trees from the start and goal and attempts to connect them. The sampling of new nodes for each tree is based on the area of its Voronoi region, which biases exploration towards unexplored paths.

These methods can work well for certain application such as planning motion for robots with high degrees of freedom, but, to produce realistic behavior when simulating crowds, there is a lack of flexibility when using these methods as is. Examples are a lack of smoothness of the paths or the inability to avoid other agents or dynamic obstacles.

2.2 Crowd simulation

To fix the aforementioned issues with current methods, van Toll et al. suggests to use a hierarchy for agent navigation in virtual crowds [39] that allows for the needed flexibility. The five levels are high-level planning, global route planning, route following, local movement, and animation.

2.2.1 High-level planning

High-level planning plans which goals to visit by translating a semantic action (e.g. go to the store) to one or more paths from a start position to a goal position.

2.2.2 Global planning

The second level, *global path planning*, creates a route between the start and goal passed from the *high-level planning* level. For some applications, such as motion planning for robotics, these paths can be used as is. For other applications such as video games or evacuation studies, these paths are unrealistic and are used as a route to be followed roughly, also known as an indicative route. Examples are the probabilistic roadmap method and the RRT-connect method.

The probabilistic roadmap method [21] attempts to construct a roadmap of the environment to use for queries. The learning phase that constructs the roadmap is divided into two steps: a construction step and an expansion step. The construction step is used to add nodes to the roadmap and the expansion step is used to improve connectivity of the roadmap by selecting nodes in difficult regions and attempts to connect them to existing connected components.

2.2.3 Route following

The *route following* level calculates a preferred velocity that follows the global path. Examples are the Indicative Route Method and the Modified Indicative Routes and Navigation method.

The Indicative Route Method (IRM) [20] by Karamouzas et al. uses a corridor around the global path or indicative route to solve flexibility issues. A backbone path is created by retracting the indicative route onto the medial axis of the environment. The corridor that represents the free space around this path is then calculated. The indicative route is then followed using four different forces:

- A boundary force that is a repulsive force away from the corridor's boundary
- A steering force towards an attraction point along the indicative route
- A noise force to generate slightly different paths for each agent
- An obstacle-avoidance force that is a repulsive force from obstacles within a region of influence

A more recent method based on the Indicative Route Method is the Modified Indicative Routes and Navigation [18] or MIRAN for short by Jaklin et al. This method adds the possibility to take an agent's region preference into account. For example, bicyclists may prefer the bike lane over other lanes and pedestrians may dislike walking over muddy terrain. This problem is also described as an agent traversing a heterogeneous environment and is solved here by assigning weights to the environment polygons that represent each type of terrain. The steering force that the Indicative Route Method

uses is complemented with weights assigned to attraction points based on the weights of the types of terrain that are crossed.

2.2.4 Local behavior

The *local behavior* or collision-avoidance takes the preferred velocity calculated by the route-following method and calculates the final velocity by taking a velocity close to the preferred velocity that avoids other agents and obstacles.

One of the earlier methods for local behavior is the social-force model by Helbing and Molnár [15]. It is stated that pedestrian movement can be described as if they would be subject to "social forces". Instead of these forces being physical, external forces, they originate from an agent's internal motivation to move. This internal motivation depends on the agent's field of view. The social force is comprised of three different components. The first component is a force that causes an attraction towards the goal. The second component is a collision-avoidance force and the last component is an attractive environment force. This last force is used to attract the agent towards people (friends, street performers etc.) or objects (stores, shop displays, etc.).

A more recent method by van den Berg et al. [37] presents the Optimal Reciprocal Collision-Avoidance or ORCA method based on the concept of velocity obstacles [6]. To determine the best velocity for an agent, a half-plane of permittable velocities that are guaranteed to be collision-free with respect to each other agent is determined. The intersection of these half-planes then produces the set of permitted velocities for that agent compared to all other agents.

2.2.5 Animation

And last, the *animation* level adds an animation to the constructed path.

2.3 Flow-based methods

Another category of crowd simulation methods are flow-based methods. As opposed to agent-based methods, flow-based methods focus on the crowd as a whole and often have a decrease in computation times as a result.

One of the earlier flow-based methods is the flow tiles method [3] by Chenney. This method provides the user with an interface to construct a velocity field using flow tiles. These velocity fields can be used for different purposes besides crowd simulation such as the simulation of rivers or fog. Opposite to its broad use is the lack of implementation options for specific behavior needed for crowd simulation such as varying goal points or speeds between different agents. This makes its use for crowd simulation limited since there is a lack of individual behavior which reduces the amount of realism that can be achieved.

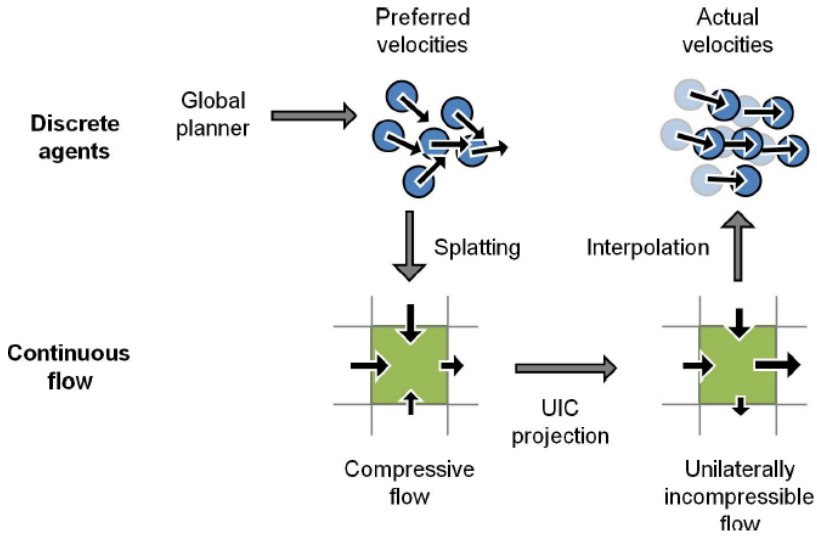


Figure 4: Overview of the algorithm by Narain et al. (Figure courtesy of Narain et al. [24])

Treuille et al. presented a method that combines global planning with collision-avoidance [34]. The proposed method works by combining different potential fields to accomplish different goals. Examples are a static goal field and a dynamic field that models the other agents for collision-avoidance. These potential fields are shared among agents with roughly the same goal, which saves on computation time. The downside of this methods is the lack of individual behavior. Due to the increase of computation time, as few groups as possible are preferred. Even though it is stated that interesting crowd phenomena can be attained with few groups, the small number of groups limits the realism of simulations. Another downside for crowd simulation of high-density crowds is the lack of coordination between groups, which will lead to suboptimal paths for the crowd as a whole.

There are also methods that try to combine the possibility for individual behavior of agent-based methods with the efficiency of flow-based methods. An example of such a hybrid method is proposed by Narain et al. [24]. Fig. 4 shows an overview of this method. The method needs a global planner to determine an agent's preferred velocity. Then, the speed and density field are calculated over all agents and this information is combined with the preferred velocity to get an initial speed. Then, the unilateral incompressibility constraint is solved, which enforces volumetric constraints on the crowd and serves as a counterpart to collision-avoidance. Obstacles are avoided by increasing the density of the cells in which the obstacle is contained accordingly. Even though this method is able to simulate a large number of agents in real-time there are some drawbacks. One of the downsides of this method is that it does not anticipate collisions, which leads to suboptimal unrealistic paths.

2.4 Density-based crowd simulation

The aforementioned methods will perform well for a single agent, but as the number of agents increases, the quality of the paths as a whole will decrease since global paths might overlap causing agents to collide when this might not be necessary. The issue is that each agent plans a path without any knowledge of the positions or paths of the other agents. To solve this issue, some methods have been presented that use global density information.

A relatively simple method that uses density information for the calculation of global paths is presented by Karamouzas et al. [19]. A density map is created by adding density values to a grid and then A* is used on this grid to produce indicative routes for all agents. After the indicative routes are calculated, the Indicative Route Method is used to follow these routes. A nice feature of this method is that it is able to steer agents away from undesired regions by increasing the density of the relevant cells. Computation times are reduced because of the way the density values are updated. Some issues could still surface because the global paths are planned once based on the density information available at that time. There could be areas on this global path that were not crowded at the time the global path was created, but are crowded the moment the agent passes it.

A paper that solves this issue and uses density information for global planning and partial replanning is presented by van Toll et al. [40]. This method uses a navigation mesh that is based on the Explicit Corridor Map [7]. This navigation mesh divides the space into a set of nonoverlapping polygons. The density value for each cell is updated for each movement step by dividing the total area of all agents contained in the cell by the total cell area. These density values can be used for global planning by assigning a density value for each edge of the medial axis and using A* to find the shortest weighted path. This algorithm also allows for replanning the path to the goal. This is useful since density values change as the crowd moves and previously empty corridors can get blocked up as time passes. A downside to this replanning is that it is possible for an agent to switch between two corridors if the situation is really unfortunate. This could be solved by implementing some form of memory.

While both above methods only influence global behavior, the method by van Goethem et al. [38] focuses on local behavior. It combines individual behavior with flow behavior. Each agent's velocity is computed by interpolating between the agent's preferred velocity and the perceived stream velocity using the agent's incentive. The agent's local perceived density influences the stream velocity in a way that switches between following and aligning behavior. The incentive is influenced by, among other things, the deviation from the preferred velocity, the local density and the time spent so far to reach the goal. The presented method reduces the number of collisions between agents at high densities, resulting in more time-efficient paths. The only downside is that the method is local and will not influence an agent's

global path.

Another method that combines global and local behavior is described in the master thesis by Bloemheuvel [2]. This method uses a dynamic global path to increase flexibility. This path is then followed using an alternative Indicative Route Method which uses local density. If the density is high, the lane will be followed strictly. If the density is low, the shortest path is taken which is in accordance with real-life observations.

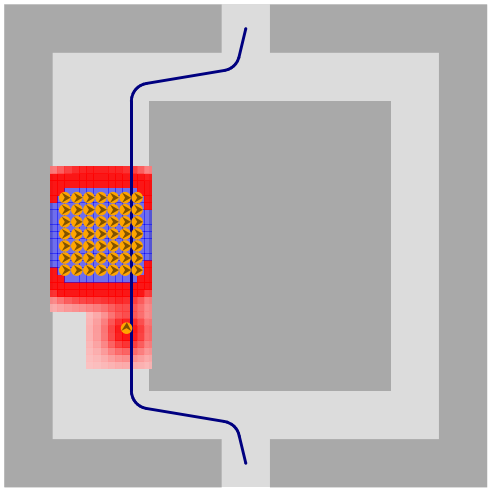
Another method focused specifically on the simulation of high-density crowds is the High-Density Autonomous Crowds method or HiDAC [25] by Pelechano et al. This method is based on the social forces model discussed in Section 2.2.4 and is combined with psychological and geometrical rules to simulate emergent behaviors. This algorithm performs very well for evacuation situations: fallen agents are modeled by dynamic obstacles and impatient and panicking agents can be modeled as well. A shortcoming of this method is that it appears to be made specifically for evacuation simulations. Collision with other agents is only checked within the same room or near the room's doorways. This aspect of the algorithm significantly reduces computation time, but it is unclear if the algorithm could be extended to larger open spaces without a significant increase in computation time.

To conclude: there are methods handling the global issues and ones handling the local issues seen in high-density planning, but none manage to solve both issues at the same time. Also few of these methods solve the issue of underutilized space when for example turning corners.

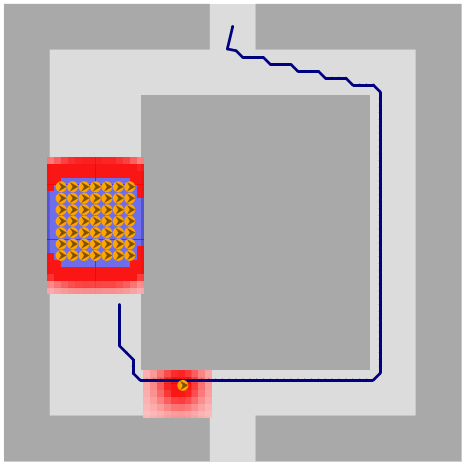
The small project by Seń [29] offers an interesting approach that does improve on local and global issues. It improves upon MIRAN's candidate attraction point selection by adding an extra weight factoring in the local density information. Another important aspect of this method is when high-density areas are detected, a detour is planned towards a point further along the indicative route using global density information. A further in-depth explanation can be found in Section 3.3.

Regarding the issues mentioned in Section 1.2, these are only partially solved or improved upon. It does, however, improve upon both local and global issues. The first local issue regarding flows is not handled, as all other agents are stationary. The issue of underutilized space around corners is improved upon by the improved candidate attraction point selection by using the density grid.

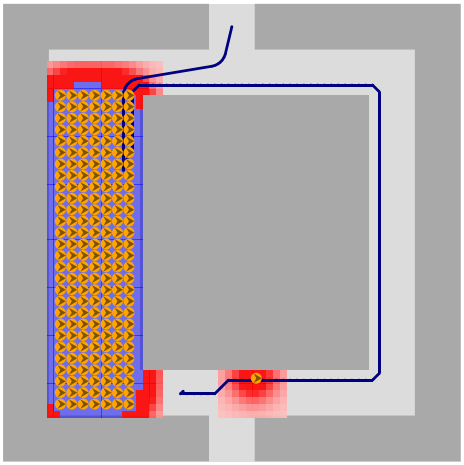
Regarding the global issues: this method will not avoid crowded paths when calculating the initial indicative route, but when they are detected as the indicative route is traversed, a detour will be planned. This detour can plan around deadlocks as can be seen in Fig. 5., but it can also make an unnecessary detour when the agent is already going with the flow of agents. Another issue is that the quality of the detour that is planned depends on the local-area-radius parameter. If this parameter puts the goal for replanning inside of the deadlock, the last part of the detour will go back into the deadlock, see Fig. 5c.



(a) The initial indicative route



(b) Indicative route after planning a detour



(c) Indicative route after planning a detour when the local-area-radius is too low

Figure 5: Global problem solving by MIRANDA

2.5 Corner turning

A situation where problems at high densities often are apparent is when agents turn a corner. The methods mentioned in this section try to improve agent behavior at corners.

A relatively simple method for corner turning is presented by Rojas et al. [28]. This paper utilizes an invisible group agent comprised of five fixed agent-sized slots in a row. An agent is assigned to each of the slots. Each agent uses its corresponding slot as a waypoint agent to be followed. The speed at which the agent follows the group agent is based on a finite-state machine depending on the distance from the group agent. For effective corner turning behavior, path wide triggers are placed right before the corner. In addition, user-defined waypoint graphs are placed to guide agents along the corner. Once the first agent corresponding to a group agent steps on the trigger, the best waypoint graph is chosen. The group agent then follows this waypoint graph to efficiently guide agents along the corner. The collision-avoidance is done separately to avoid other agents and dynamic obstacles. The existence of parallel waypoint graphs balances wide corner turns and corner hugging increasing the flow of people. The downside to this approach is that these waypoint graphs have to be created manually which can take up a significant amount of time for large environments such as city blocks.

He et al. [12] observes that people tend to use safer routes rather than short ones to move around corners [11]. To guide agents along these safer routes, shadow obstacles are placed on each corner. Once the area of the shadow obstacle has been entered, the agent will be steered such that the viewing range is maximized. Once the agent can sufficiently see beyond the corner, the shortest route to the goal is chosen again. The improved visibility reduces the chances of collision. At the same time, little use is made of additional space on the outside of the corner and there is also no coordination between agents, which will still provide issues at higher densities.

The paper by Tsai et al. [35] proposes to use a similar approach for navigation fields by placing crowd monitors at the corners of objects. These crowd monitors collect data on crowd flow and density around their assigned corner. A guidance path is then created around the corner, which is converted to a guidance field as can be seen in Fig. 6. This approach reduces congestion because it is specifically tailored to the local density and flow, and guides agents away from congested areas. One disadvantage of this method is that it does not handle dynamic obstacles. Also, the underlying navigation field is created using only a single goal cell, guiding all agents to the same point.

The paper by Dias et al. [4] uses a Cellular Automata model to move agents around corners. To produce realistic behavior, their model is calibrated using experimental data collected by Zhang et al. [42]. Both discrete and continuous floor representations were used and it was verified that both could accurately represent pedestrian behavior around corners. One of the advantages of Cellular Automata models is that they are easy to implement,

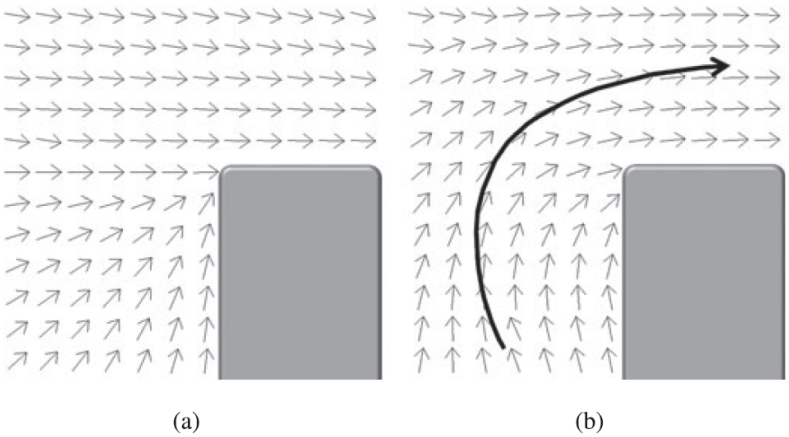


Figure 6: (a) The initial navigation field. (b) The guidance path and resulting navigation field. (Figure courtesy of Tsai et al. [35])

in return more complex individual behavior cannot be modeled. With this model, the used trajectory data was of a single person turning a corner and so more complex inter-pedestrian interactions cannot be modeled.

To conclude: the methods mentioned in this section are specialized in improving behavior around corners, but it is often unclear if and how the other local and global density-related issues are being dealt with. This shows that there is a need for a method that either switches between existing methods or an extension of current methods to solve multiple issues at the same time.

3 Preliminaries

This section provides an in-depth explanation of the geometric concepts and algorithms used and extended within this thesis. In Section 3.1, we discuss basic geometric concepts that underlie the Modified Indicative Routes And Navigation algorithm discussed in Section 3.2. In Section 3.3, we discuss an extension: the Modified Indicative Routes And Navigation with a Detouring Algorithm method. In Section 3.4, we will discuss the velocity field and memory components, which are components of our own extension.

3.1 Geometric Concepts

3.1.1 Voronoi diagram

The Voronoi diagram of a set of points P is the subdivision of the plane into regions. For each point p in P called a *site*, its corresponding region consists of all points in the plane closest to p. Points that are equidistant from two sites form the *Voronoi edges* and points equidistant from three or more sites are the *Voronoi vertices*. An example of a Voronoi Diagram of a set of points can be found in Fig. 7.

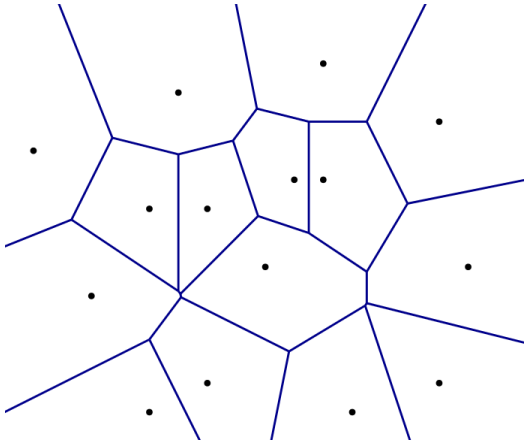
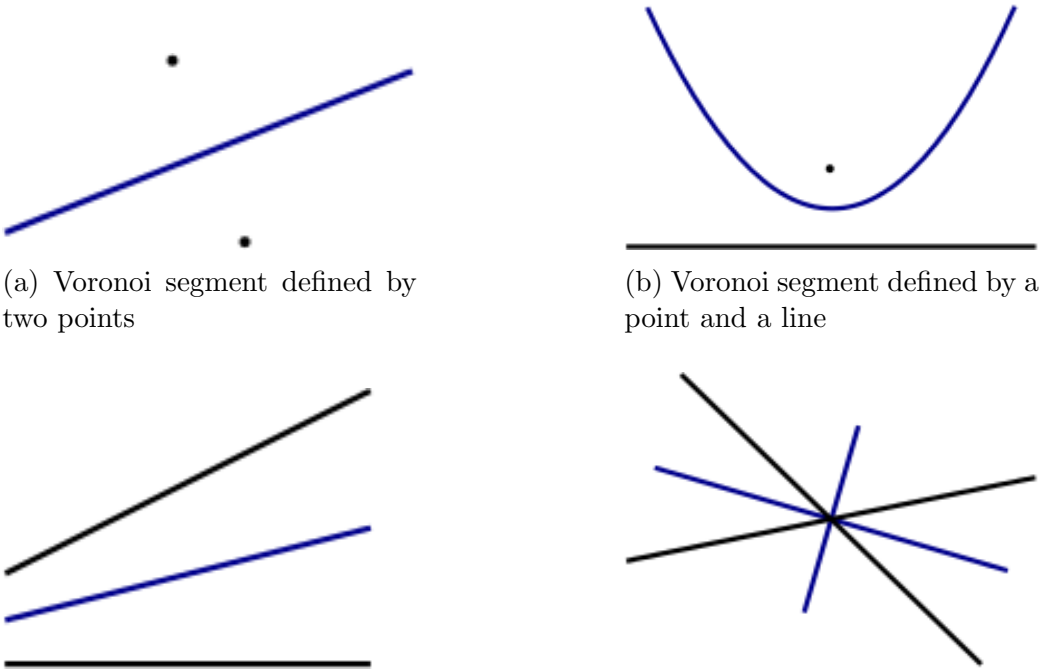


Figure 7: A Voronoi diagram of a set of points

There are various generalizations of the Voronoi diagram called *Generalized Voronoi Diagrams* (GVD) including Voronoi diagrams for higher-dimensional spaces, Voronoi diagrams with different metrics and distance functions, and Voronoi diagrams with different types of inputs. For applications in crowd simulation, we are interested in Voronoi diagrams of straight-line segments. This diagram consists of two types of edges: a straight-line segment and a parabolic arc. Figure 8 shows the different types of Voronoi edges depending on the type of sites.

Using the definition of a GVD for line segments, we can extend the concept to polygons by treating each polygon as a finite chain of straight-line segments closing in a loop.



(a) Voronoi segment defined by two points

(b) Voronoi segment defined by a point and a line

(c) Voronoi segment defined by two lines

(d) Two Voronoi segments defined by two intersecting lines

Figure 8: Different Voronoi segments based on the type of sites. The blue segments are the Voronoi segments defined by the black sites

3.1.2 Medial axis

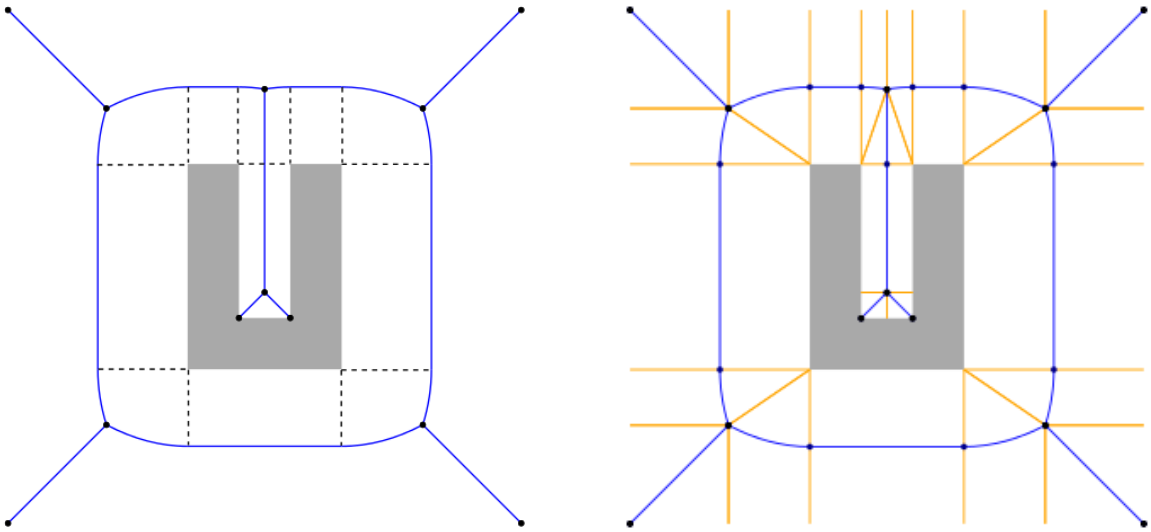
Related to the concept of the GVD is the *medial axis*. The medial axis is obtained from the GVD by deleting edges that connect to obstacle's convex corners. The medial axis of a U-shaped environment is showed in Fig. 9a. Note that the edges that are removed are not necessary for path planning since they do not give any access to new parts of the environment. These edges are merely connections between the medial axis and the obstacle.

3.2 Modified Indicative Routes And Navigation

In this Section we discuss the Modified Indicative Routes And Navigation or MIRAN method and its underlying concepts. Algorithm 1 gives an overview of the method.

3.2.1 Explicit Corridor Map

The *Explicit Corridor Map* (ECM) is the medial axis of the environment annotated with *event points* together with their closest points to obstacles. Every edge is formed by a concatenation of the types of segments shown in Fig. 8. The points where these types of segments join are called the *event points*. For every vertex and event point, the closest obstacle point is stored.



(a) (b)

Figure 9: (a) The medial axis of a U-shaped environment in blue. The dotted lines are the edges removed from the GVD to obtain the medial axis. (b) The Explicit Corridor Map of a U-shaped environment. The orange lines connect the vertices to their closest obstacle points.

Algorithm 1 The MIRAN method

Input. Start s, goal g, indicative route from s to g
Output. Smooth terrain-dependent path from s to g

1: $i \leftarrow 0$
2: $x_0 \leftarrow s$
3: **while** $x_i \neq g$ **do**
4: $r_i \leftarrow$ ComputeReferencePoint(x_i)
5: $\mathcal{A}_i \leftarrow$ ComputeCandidateAttractionPoints(r_i, x_i)
6: $\alpha_i \leftarrow$ PickBestCandidate$(\mathcal{A}_i, x_i)$
7: $x_{i+1} \leftarrow$ MoveAgentTowardsAttractionPoint(x_i, α_i)
8: $i \leftarrow i + 1$

Since an edge is defined by two obstacles, the closest obstacle points are the closest points on each of these obstacles. This is the same for the vertices, except these are defined by three or more obstacles and will have an equal number of closest obstacle points. Fig. 9b shows the ECM of a U-shaped environment.

3.2.2 Explicit Corridors

The ECM can be used to plan a path for each agent. The start point s and goal point g are retracted onto the medial axis resulting in s' and g'. A graph search is then used to find the shortest path between s' and g'. This path is called the *Indicative Route* or IR. In order to plan a path, an *Explicit Corridor* is extracted from the ECM corresponding to the indicative route. This explicit corridor is a description of the free space around the path. It is defined as a sequence of maximum clearance disks with their center on the medial axis. This corridor is used to be able to plan around dynamic obstacles and agents, since the corridor indicates the free space around the path.

3.2.3 Moving the agent

After the path is calculated and the corresponding corridor is extracted, MIRAN uses a force-based steering method very similar to the IRM to follow the path. Three forces are calculated to steer the agent towards its goal: A steering force $F_s(x)$ towards the goal, a boundary force $F_b(x)$ away from the corridor's boundary and an avoidance force $F_o(x)$ to avoid dynamic obstacles and other agents.

3.2.4 Computing and Choosing Attraction Points

The steering force $F_s(x)$ is implemented as an attractive force towards an *attraction point*. This attraction point is a point on the medial axis ahead of the agent, pulling the agent towards the goal. First, the set of possible attraction points $\mathcal{A}_i$ is calculated for each timestep i after which the best attraction point is selected.

To compute $\mathcal{A}_i$, first, the reference point is computed. If the agent's current position is x_i, the reference point r_i is defined as the first closest point from x_i to the part of the indicative route π_{ind} that lies between the previous reference point r_{i-1} and the previous attraction point α_{i-1} for $i \leq 1$ (Fig. 10). For the initial step $i = 0$, $r_0 = x_0$ is used.

To calculate the set of attraction points $\mathcal{A}_i$, two parameters are used: the sampling distance d and the shortcut parameter σ. The *sampling distance* defines the maximum curve length distance between the candidate attraction points. The *shortcut parameter* defines the maximum curve length distance from the reference point to the farthest attraction point. If the value of this parameter is increased, points closer to the agent but further along π_{ind} can

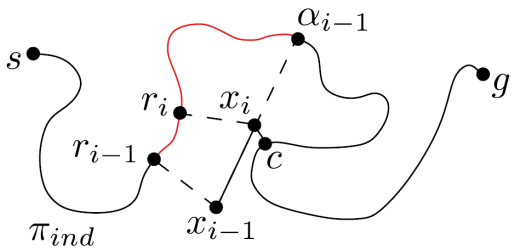


Figure 10: The reference point is the closest point on the part of π_{ind} that lies between r_{i-1} and α_{i-1}. Choosing the closest point c would lead to an undesired shortcut. (Figure courtesy of Jaklin et al. [18])

be used as an attraction point, increasing the length of path that is skipped and increasing smoothing.

After calculating the reference point, the part of π_{ind} that is visible from x_i: $\mathcal{V}_i$ is computed. This leads to the division of π_{ind} into invisible and visible intervals $V_j = [a_j, b_j]$ such that for each $t \in V_j$, $\pi_{ind}(t)$ is visible. The endpoints of each visible interval $\pi_{ind}(a_j)$ and $\pi_{ind}(b_j)$ are the first points added to $\mathcal{A}_i$. Next, each visible interval is sampled using sampling distance d as the curve length distance between two consecutive attraction points. The sampling is repeated until the total curve length distance exceeds shortcut distance σ. The final set $\mathcal{A}_i$ (Fig. 11) contains all attraction points sorted from smallest to largest curve length distance from the reference point.

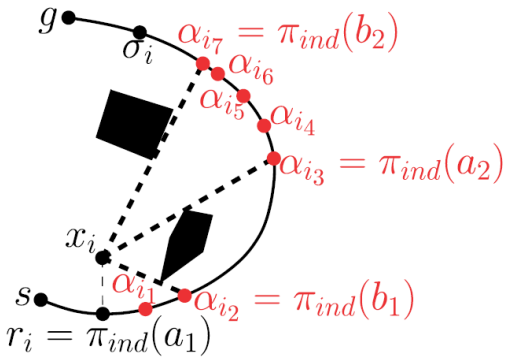


Figure 11: Set of attraction points generated by the MIRAN method (Figure courtesy of Jaklin et al. [18])

After computing $\mathcal{A}_i$, the best point needs to be selected to serve as the agent's attraction point. This is done using weighting function ω. For each attraction point α_{i_j} we consider the straight-line segment between the agent's current position x_i and α_{i_j}. The weight ω of this line segment $l(a_{i_j}, x_i)$ is computed using the Euclidean distance of the line segment, the different types of terrain it crossed and the agent's terrain preferences by using the following formula:

$$\omega(l(a_{i_j}, x_i)) = \sum_{T \in \mathcal{T}_{i_j}} w(T) * l_{i_j}^T / d_{i_j},$$

where $\mathcal{T}_{i_j}$ is the set of terrain types that $l(a_{i_j}, x_i)$ crosses, $w(T)$ is the weight for each terrain type T, $l_{i_j}^T$ is the length of the line segment that intersect terrain type T and d_{i_j} is the curve length distance along π_{ind} from the reference point r_i to the candidate attraction point α_{i_j}. After calculating the weights for each attraction point, the attraction point corresponding to the lowest weight ω is selected as the final attraction point.

The steering force $F_s(x)$ is now in the direction of the chosen attraction point and with some speed close to the agent's preferred speed. The total force $F(x) = F_s(x) + F_b(x) + F_o(x)$ is the force exerted on the agent.

3.3 Modified Indicative Routes And Navigation with a Detouring algorithm

The Modified Indicative Routes And Navigation with a Detouring algorithm or MIRANDA is an extension for MIRAN with two key components: the density-based attraction point selection and the detouring algorithm which both rely on the density field.

3.3.1 The Density Field

The density field $\mathcal{DF}$ is a function that provides a density value ρ for every point in the free space. The function used here is the *Gaussian Density Distribution* function with $\sigma = 4$, applied to a grid. Each agent interacts with this density field using two parameters: ρ_t is the maximum density an agent is willing to navigate through and ρ_s is the scaling factor that describes the weight of the density values when planning a path.

3.3.2 Density-based Candidate Attraction Point Selection

This method uses the information from the density field to improve on the candidate attraction point selection from the MIRAN method. This is done by adding a density weight to the weight for each attraction point. The value of this weight is the weighted average value of density over the line segment $l(\alpha_{i_j}, x_i)$. There is also a density threshold ρ_t. When the density for any point on the line segment $l(\alpha_{i_j}, x_i)$ is larger than ρ_t, the value of the density weight is set to $infinite$. This represents that the path from x_i to α_{i_j} is not traversable. This results in the following weight function:

$$\delta(l(a_{i_j}, x_i)) = \begin{cases} infinite & \text{if } \mathcal{DF}(p) > \rho_t \text{ at any point } p \text{ on } l(\alpha_{i_j}, x_i) \\ \frac{\int_{\alpha_{i_j}}^{x_i} \mathcal{DF}(p)\mathrm{d}p}{d_{i_j}} & \text{otherwise} \end{cases}$$

After calculating the density weight, the final weight is calculated by adding it to the original weight:

$$w(\alpha_{i_j}) = \omega(l(\alpha_{i_j}, x_i)) + \rho_s * \delta(l(\alpha_{i_j}, x_i))$$

3.3.3 The Detouring Algorithm

In addition to the density-based candidate attraction point selection, a detouring algorithm is used to plan around blocked routes. Algorithm 2 gives an overview of the algorithm. The first step in constructing the detour is the calculate the *local detour goal*. The local detour goal g_{detour} is considered to be the limit of the local area for detouring purposes (Fig. 12). g_{detour} is taken to be the first intersection of the unvisited part of the agent's IR: π_{free} and the *local area boundary*. The local area boundary is a circle with the agent's position x_i as its center point, and a radius d_{local}. The first intersection is taken to prevent undesired detours in for example a maze-like environment.

Algorithm 2 FindLocalDetour

Input. Current position x_i, local-area-radius d_{local}, indicative route π_{ind} and its unsampled subcurve π_{free}, goal position g, density field $\mathcal{DF}$
Output. A local detour $\pi_{modified}$ from x_i to a point on π_{free} free from high-density areas.

$c \leftarrow$ the circle centred on x_i with radius d_{local}
$\mathcal{P} \leftarrow$ the set of intersections between c and π_{free}
if $\mathcal{P}$ is empty **then**
    $g_{detour} \leftarrow g$
else
    $g_{detour} \leftarrow$ the first point along π_{free} in $\mathcal{P}$
$\pi_{detour} \leftarrow$ A*(x_i, g_{detour})
$j \leftarrow 0$
$p_0 \leftarrow g_{detour}$
$\pi_0 \leftarrow$ A*(x_i, p_0) + the subcurve of π_{ind} from p_0 to g_{detour}
while $path\_cost(\pi_j) \leq \beta * path\_cost(\pi_{detour})$ **and** $p_j \in \pi_{free}$ **do**
    $p_{j+1} \leftarrow p_j$ slid back d along π_{free}
    $\pi_{j+1} \leftarrow$ A*(x_i, p_{j+1}) + the subcurve of π_{ind} from p_{j+1} to g_{detour}
    $j \leftarrow j + 1$
$\pi_{modified} \leftarrow \pi_{j-1}$

After g_{detour} is determined, a path from x_i to g_{detour} is found using A*. The cost function of A* is altered to take density information into account. The step cost now consists of the line segment length, the region weight and the density weight, resulting in the following formula:

$$step_{cost}(n_a, n_b) = l(n_a, n_b) + \omega(l(n_a, n_b) + \delta(l(n_a, n_b)) * \rho_s$$

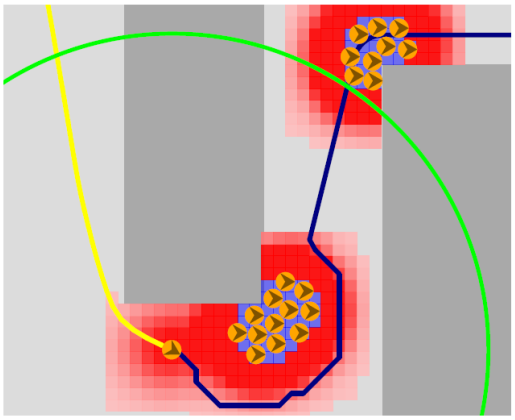


Figure 12: The local area boundary in green, the intersection with the IR is the detour goal (Figure courtesy of Seń et al. [29])

3.3.4 Anchoring Bias

After finding the detour, the method states that there is an *anchoring bias*: a desire to return to the original path. The goal is to find the path that reconnects to the original path the fastest while the difference in path cost lies under a certain percentage of the cost of the original path. This is implemented by sampling π_{free} in the backwards direction starting at g_{detour}. For each sample π_j, a path from the agent to π_j is calculated using A*. The new path cost is compared to the increased original cost:

$$path\_cost(\pi_j) \leq \beta * path\_cost(\pi_{detour}),$$

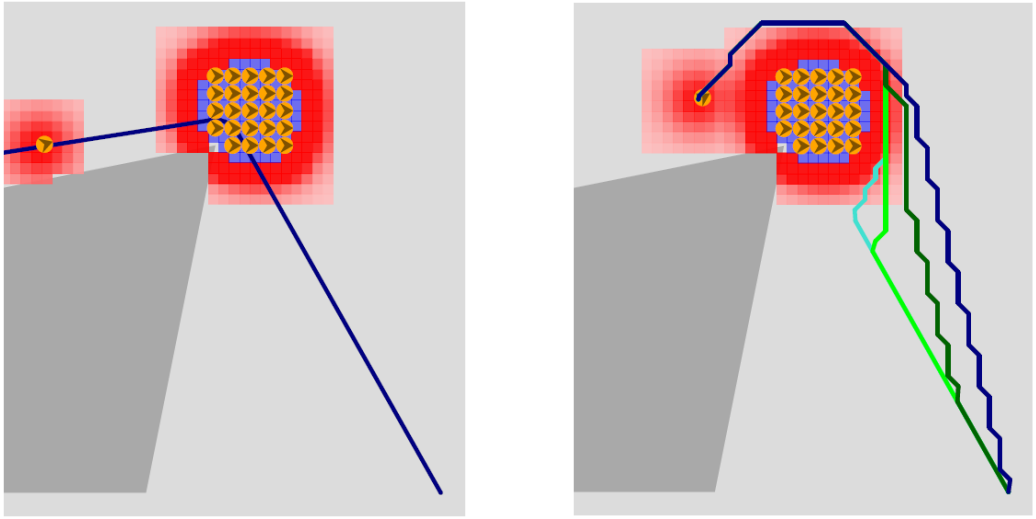
where $\beta > 1$ is the factor by which the path cost is allowed to increase. When π_j does not meet this requirement anymore, π_{j-1} is chosen as the final detour goal $\pi_{modified}$. Detour paths with different values for β can be found in Fig. 13.

3.4 Improved Deadlock Detection and Detours

In this Section we discuss the velocity field and memory components which are components of our own algorithm.

3.4.1 The Velocity Field

One of the things we want to accomplish with our extension is that when a group of agents blocks a path for another agent, but their velocity is very similar to this agent's velocity, no deadlock is detected and the agent will continue its original path. To accomplish this, we will use a *velocity field* to access the velocity of agents. The velocity field $\mathcal{VF}$ is a function that provides a velocity value v_p for every point p in the free space. Any desired



(a) (b)

Figure 13: (a) Original path. (b) Detour path using different values for β. Dark green $\beta = 1.05$, light green $\beta = 1.075$, turquoise $\beta = 1.1$, dark blue is the detour before modification. (Figure courtesy of Seń et al. [29])

function can be used here, but we propose the following formula as used in Hillbrand et al. [17] :

$$\vec{V}(l,t) = \frac{\sum_{p \in P(t)} \vec{v}_p f(l,p)}{\sum_{p \in P(t)} f(l,p)},$$

where l is a location, t is the current time, $P(t)$ is the set of locations of the agents, $\vec{v_p}$ is the velocity of agent p and $f(l,p)$ is a weighting factor. Because we use the Gaussian-based density function, the corresponding weighting function is as follows:

$$f_{gaussian}(l,p) = \frac{1}{\pi R^2} e^{-\frac{d(l,p)}{R^2}},$$

where $d(l,p)$ is the Euclidean distance between l and p and R is a parameter that influences the agent's contribution to the perceived density. This function is applied to a grid over the free space, where the cell width needs to be equal to the cell width used for the density field.

3.4.2 Memory

A disadvantage of previous methods is, in worst case, that an agent could switch between two paths indefinitely. In this case, using some form of memory can be beneficial. Memory could also be useful with stationary agents if, for example, we would limit the use of density values only to the cells that are visible to the agents. This could prevent the same situation where the best path is alternating between two corridors, and would add more realism

at the cost of longer total path lengths. In these cases, we will refer to the agents blocking the path as a *block*.

To solve the issue, we add a list of memories $\mathcal{M}$ to the agent. A *memory* M consists of the following components:

- $p_{blocked_1}$: the first point on π_{ind} that was blocked
- $p_{blocked_2}$: the last point on π_{ind} that was blocked
- $t_{blocked}$: the time the block was encountered
- $c_{blocked}$: a list of the cells that are affected
- v_{block}: the block's average velocity

When a deadlock is detected, $c_{blocked}$ is determined by using a flood-fill algorithm starting at $p_{blocked_1}$ to find all grid cells c for which $\mathcal{DF}(c) > \rho_t$. v_{block} is calculated by taking the average velocity of these cells:

$$v_{block} = \frac{\sum_{c \in c_{blocked}} \mathcal{VF}(c)}{|c_{blocked}|}$$

4 Improved Deadlock Detection and Detours

In this Section we will discuss our own contribution: the Improved Deadlock Detection and Detours or ID^3 algorithm. The ID^3 algorithm consists of several components discussed in the following sections. The deadlock detection is improved by introducing vision-based deadlock detection and accounting for flow. The detours are improved by determining the detour goal by sampling density values and detecting what type of detour is planned. For global detours, paths are repaired using gates and memory is added to prevent continuous switching between two paths.

4.1 Vision-based deadlock detection

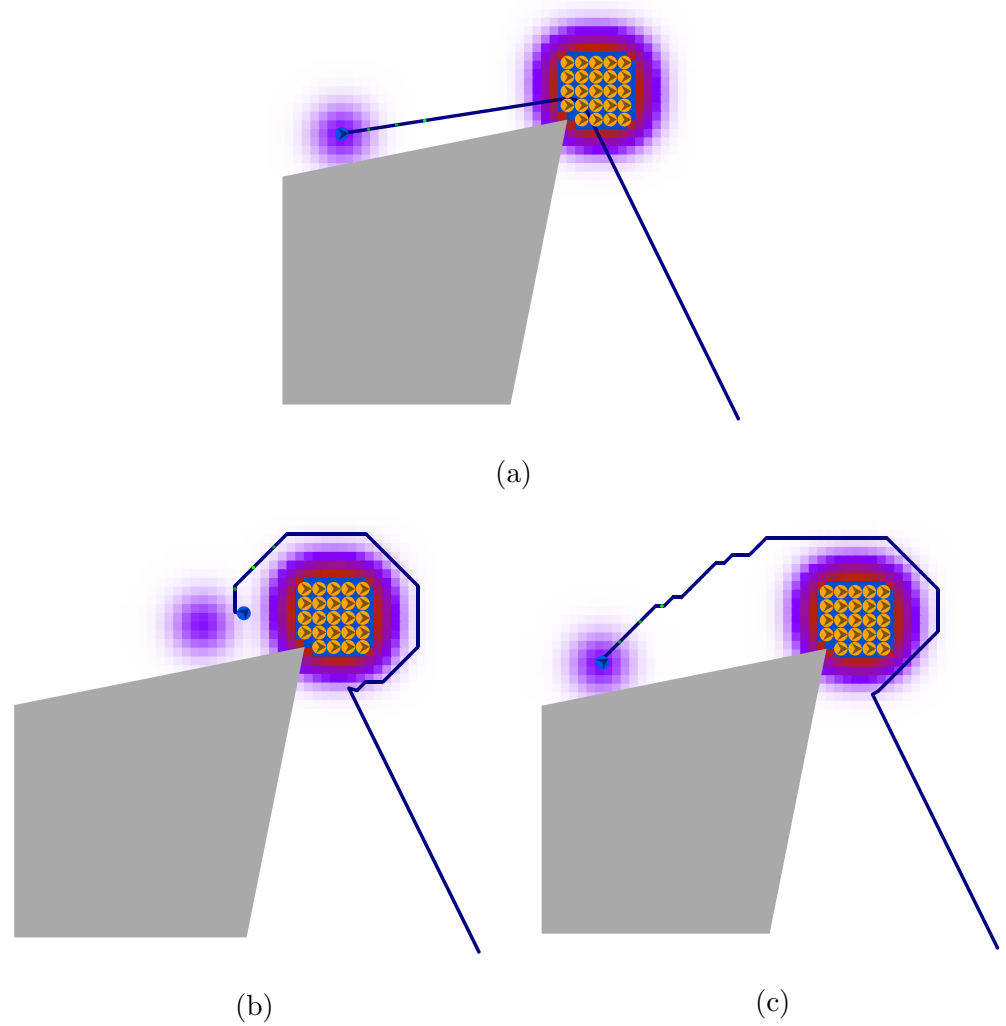
The detection of a deadlock when using the MIRANDA method uses the sampling and look-ahead distance used by MIRAN. The advantage of using the existing attraction points is that no extra computation time is needed to compute these points. Unfortunately, there are several disadvantages:

When using higher sampling distances similar to the ones in the paper introducing the MIRAN method ($d = 10$ or 20), small deadlocks can easily go undetected, unless the cap rejection limit is really low. Changing the parameters involved with the deadlock detection also influences the MIRAN algorithm, while, ideally, these components would be separate. Lastly, unless a high look-ahead distance is used with a relatively low rejection cap limit, deadlocks will only be detected two meters ahead (Fig. 14b).

To ensure deadlocks are detected in time, we propose a vision-based method (Fig. 14c). This method uses a deadlock length parameter l_d that indicates the size of the deadlocks the agent needs to plan around. The visible part of the agent's indicative route is sampled using sampling distance d_v which is calculated as follows:

$$d_v = l_d * precision$$

Where the *precision* determines the number of sample points. Making the number of sample points dependent on the deadlock length ensures we do not get more sample points than necessary for large deadlock sizes. For now, we use $l_d = l_{cell} = 0.3$ so a detour is always planned if a path is blocked, independent of the number of agents blocking the path.



(a)

(b)

(c)

Figure 14: (a) Initial configuration. (b) Without vision-based detection, the deadlock is detected very late. (c) With vision-based detection, the deadlock is detected at the start. This way, the agent starts evasion at an earlier stage.

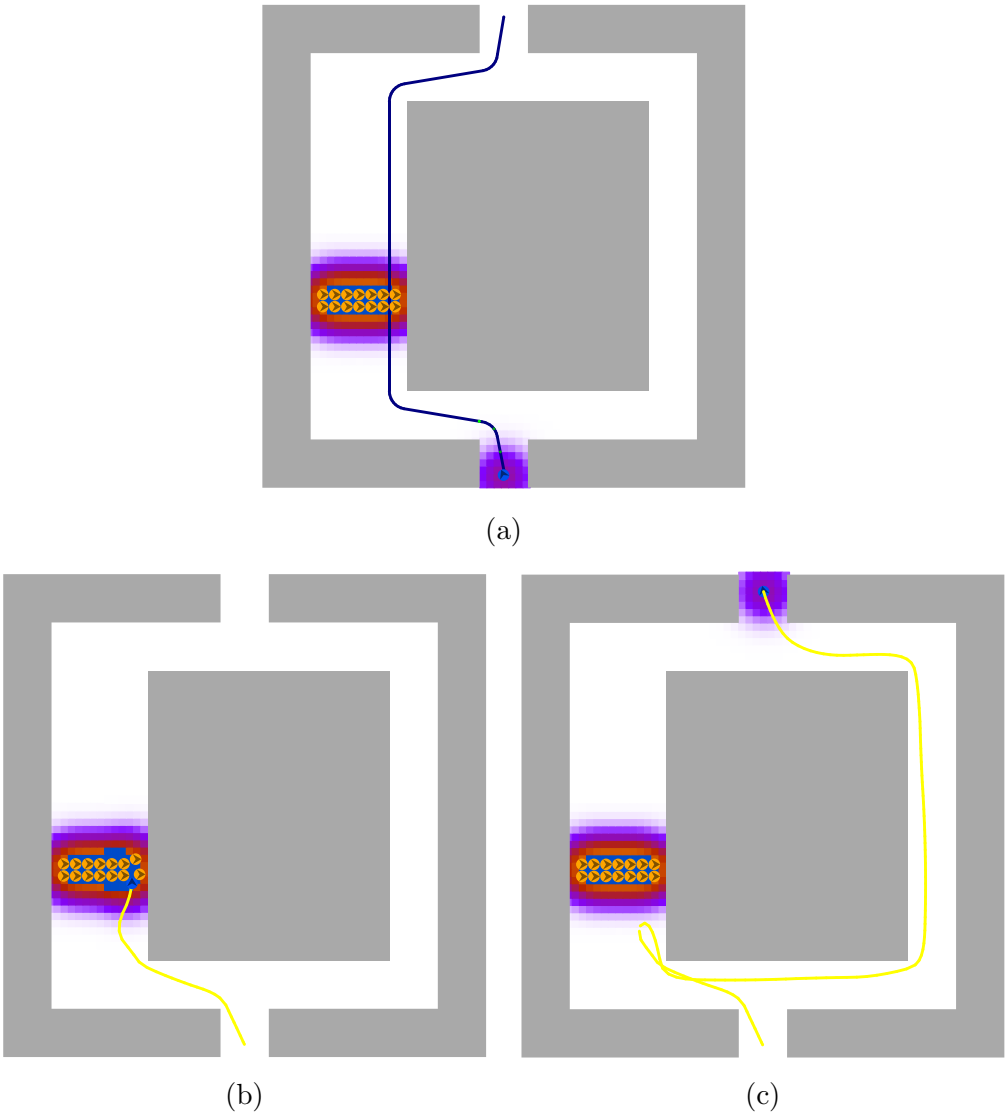
For each sample point p_j, if the density value for the corresponding grid cell is higher than the density threshold, the sample point is blocked:

$$\mathcal{DF}(p_j) > \rho_t$$

The distance between consecutive blocked points is measured and if this distance exceeds l_d, it is classified as a deadlock. We chose $precision = 0.2$ to ensure that in the worst case, the measured deadlock size is only 20 percent lower than the actual size. The precision can be increased, but this will come at the cost of higher computation times.

Fig. 15 shows the influence of l_d on the agent's behavior. While we set our deadlock size to the grid cell size, it might be beneficial to increase the size so detours are not planned around small deadlocks. When there are only two rows of agents as in Fig. 15a instead of replanning around the block,

navigating through the block of agents might be more desirable behavior. We will extend on possible methods to accomplish this in Section 7. Fig. 15b shows an agent with $l_d = 2.0$ which prevents the detour being taken. Fig. 15c shows an agent with $l_d = 0.3$ which results in the agent planning a detour around the block.



(a) (b) (c)

Figure 15: (a) The initial scenario. (b) No detour is taken because the part of the IR that is blocked is smaller than the deadlock size parameter. (c) The deadlock size parameter is set to grid cell width and a detour is planned.

In contrast to MIRANDA which checks for deadlocks every simulation step using information that is mostly already present in the framework, our method computes additional sample points. Because of this, we decided to add a cooldown t_{vision} to our vision-based method. The cooldown is the number of seconds between our vision-based deadlock checks. The addition

of the cooldown ensures that the user can make their own trade-off between precision and computation time that is most suitable for the application.

4.2 Flow

The MIRANDA method plans a detour when a dense enough group of agents is detected, regardless of the movement of this group (Fig. 16b). When the movement of the group of agents is similar to the agent's desired movement, the agent can move with the flow and prevent an unnecessary detour (Fig. 16c). To accomplish this, we used the concept of an incentive from van Goethem et al. [38]. This concept uses several factors to determine an agent's incentive to coordinate with the crowd; the one we adopt is the deviation factor Φ, which makes an agent leave a stream if its individual velocity deviates too much from the stream's velocity.

The MIRANDA method checks if the density value of an attraction point exceeds threshold ρ_t to determine whether or not it should be blocked. In addition, we add two requirements regarding the difference between the agent's velocity v_{indiv} and the stream's velocity v_{stream}. For an attraction point to be blocked, the first additional requirement is that the difference between the angle of the agent's velocity v_{indiv} and the flow's velocity v_{stream} must exceed threshold Φ_{dev}. Where v_{stream} is determined by looking up the corresponding value in the velocity field $\mathcal{VF}$ as is described in Section 3.4.1. The second additional requirement is that $l(v_{stream})$ must be lower than the agent's speed $l(v_{indiv})$ multiplied by the speed deviation $speed_{dev}$. The speed deviation is the factor by which the agent is willing to reduce its speed, so $speed_{dev} = 0.5$ signifies that the agent is willing to lower its speed by half.

The added two requirements result in the improved density weight $\delta^+(l(a_{i_j}, x_i))$:

$$\delta^+(l(a_{i_j}, x_i)) = \begin{cases} infinite & \text{if } \mathcal{DF}(p) > \rho_t \text{ at any point } p \text{ on } l(\alpha_{i_j}, x_i) \\ & \text{and } \angle(v_{indiv}, v_{stream}) > \Phi_{dev} \\ & \text{and } l(v_{stream}) < l(v_{indiv}) * speed_{dev} \\ \dfrac{\int_{\alpha_{i_j}}^{x_i} \mathcal{DF}(p)\mathrm{d}p}{d_{i_j}} & \text{otherwise} \end{cases}$$

When using the vision-based deadlock detection, we do not use the agent's current velocity v_{indiv}. Instead, for each sampled point we use the expected direction of the agent at that point by taking the local angle of the indicative route v_{expect}. We do this to prevent cases where the agent's current velocity does not match the velocity at the sampled points. This for example happens when an agent is turning at a corner. Not using v_{expect} in such a case would result in the agent planning a detour while it is not necessary.

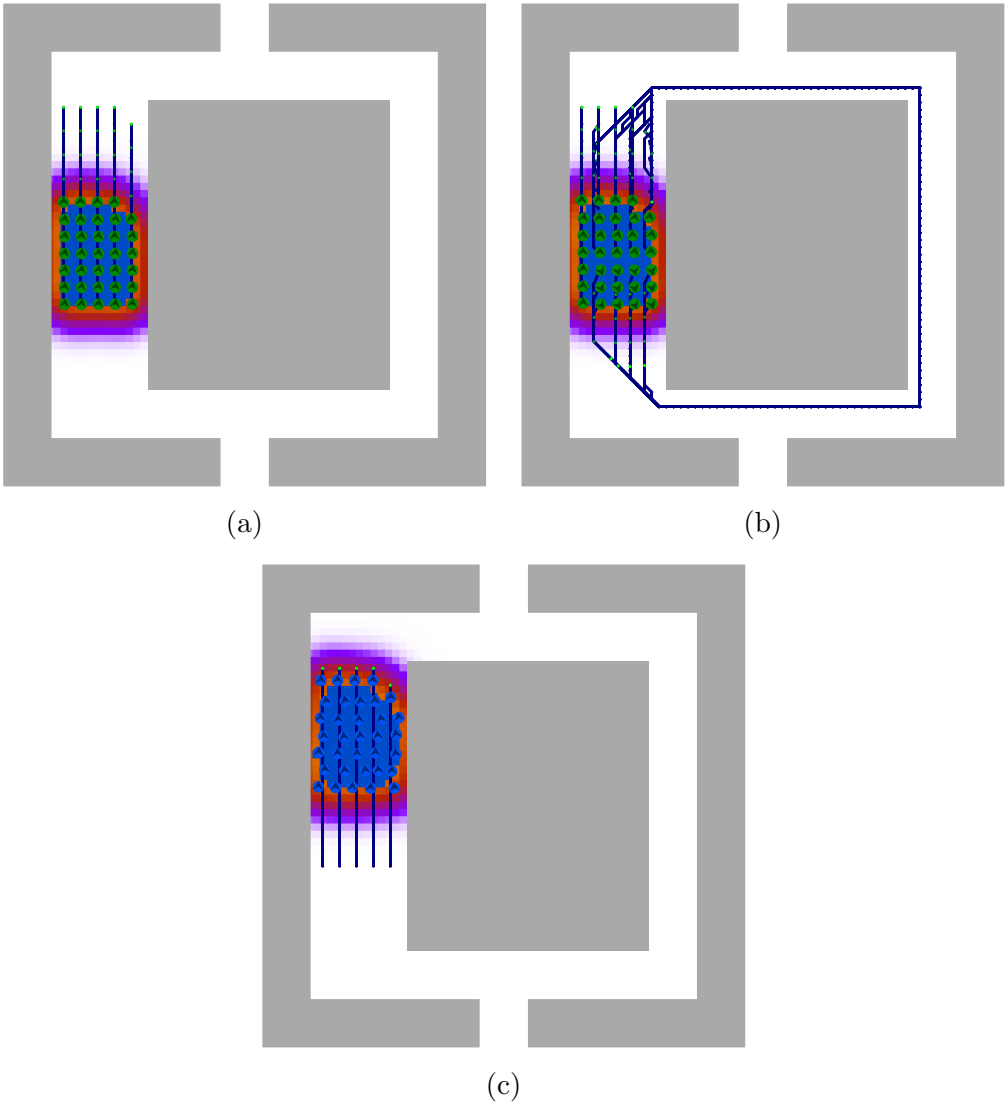
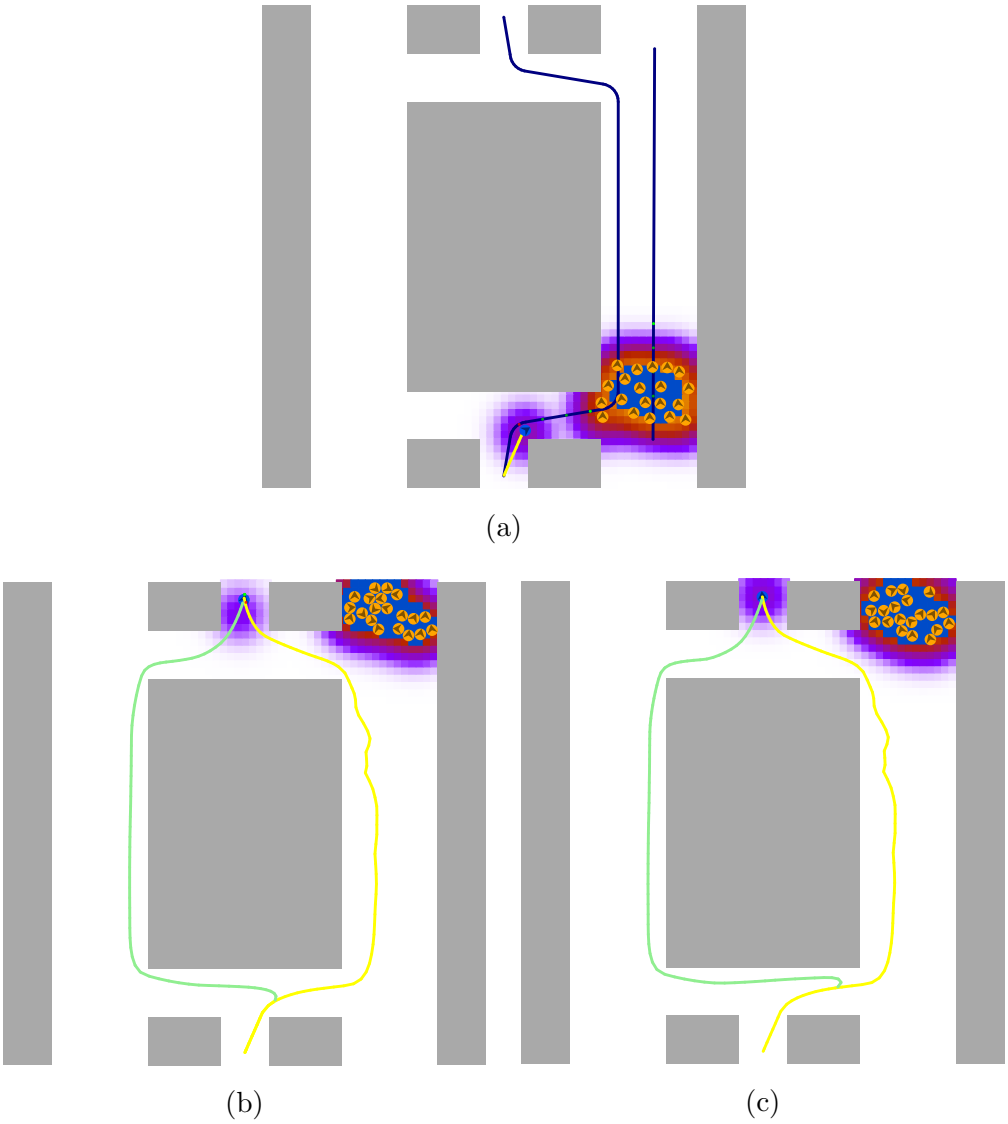


Figure 16: (a) Initial configuration. (b) With MIRANDA, deadlocks are detected and detours are planned for a large portion of the group. (c) With our method, no deadlocks are detected and the agents proceed as normal.

The influence of $speed_{dev}$ and $angle_{dev}$ is show in Fig. 17. The group has a preferred speed of 1.0 m/s and the single agent has a preferred speed of 1.4 m/s. In Fig. 17b two paths were traced corresponding to an agent with different values for $speed_{dev}$ while $angle_{dev} = \frac{1}{2}\pi$. Here the formula $l(v_{stream}) < l(v_{indiv}) * speed_{dev}$ is used to determine if a detour will be planned. The light green path is traced for an agent with $speed_{dev} = 1.0$. Here $1.0 < 1.4 * 1.0$, which results in the path being blocked and the agent planning a detour. The yellow path was traced for an agent with $speed_{dev} = 0.5$. Here $1.0 \not< 1.4 * 0.5$, which results in the cells not being blocked and the agent continuing its path behind the group of agents.

In Fig. 17c two paths were traced corresponding to an agent with different

values for $angle_{dev}$ while $speed_{dev} = 0.5$. Here the formula $\angle(v_{indiv}, v_{stream}) > \Phi_{dev}$ is used to determine if a detour will be planned. The light green path is traced for an agent with $angle_{dev} = \frac{1}{4}\pi$. Here $\angle(v_{indiv}, v_{stream}) > \frac{1}{4}\pi$, which results in the path being blocked and the agent planning a detour. The yellow path was traced for an agent with $angle_{dev} = \frac{1}{2}\pi$. Here $\angle(v_{indiv}, v_{stream}) \not> \frac{1}{2}\pi$, which results in the cells not being blocked and the agent continuing its path behind the group of agents.



(a)

(b)

(c)

Figure 17: (a) The initial scenario. (b) Two possible paths depending on the speed deviation value: the light green path for $speed_{dev} = 1.0$ and the yellow path for $speed_{dev} = 0.5$. (c) Two possible paths depending on the angle deviation value: the light green path for $angle_{dev} = \frac{1}{4}\pi$ and the yellow path for $angle_{dev} = \frac{1}{2}\pi$.

4.3 Density-based detour goal

When a deadlock is detected by using MIRANDA, the current detour goal is set to be the intersection of the local area boundary and the unvisited subcurve of the indicative route π_{free}. This point is only influenced by the radius of the local area boundary. The main disadvantage of this approach is that if the area blocked with agents is large enough, the detour goal will lie within this blocked area as can be seen in Fig. 5c. We suggest a density-based approach where π_{free} is sampled until a point with sufficiently low density is found, to use as the new detour goal. To determine this point, we sample π_{free} starting at the first blocked point $p_{blocked}$ encountered by either MIRANDA or our vision-based approach. The sample distance d is the same as the sample distance used for MIRAN. We continue sampling until we find a point for which the density lies below the density threshold ρ_d. In our experiments, we found $\rho_d = \rho_t * 0.5$ to produce good results.

$$\mathcal{DF}(p_j) < \rho_d$$

The first point p_j that satisfies this formula will serve as our detour goal g_{detour}.

4.4 Detecting the type of detour

One downside of the density-based detour goal selection is that the new detour goal is right behind the deadlock. The problem this causes is that in some scenarios, like the one seen in Fig. 21a, the section of the indicative route behind the deadlock will have to be traversed twice for the agent to reach its original goal. Since this only happens for certain detours, we would like to detect the type of detour. If the detour planned by the algorithm is not in the same homotopic class as the original path we consider it to be a *global detour* (Fig. 19c, 19d). This situation occurs when a path is blocked entirely by other agents. If there is a group of stationary agents but the detour can be planned around these agents locally, so the homotopic class is the same as the original path, we consider this to be a *local detour* (Fig. 19a, 19b).

To solve the problem of determining which type of detour is planned we use a set of waypoints or gates. We define these gates as 2D areas, part of the walkable area. We have decided to implement these gates in the form of disks using information already present in the Explicit Corridor Map: the ECM vertices and their clearance. With each vertex as a center, we use the clearance as a radius to form a disk around the center. An example of this can be seen in Fig. 18.

To determine the type of detour, first, we determine which gates are crossed if the original path would be continued and which gates would be crossed if the detour path would be taken. To determine which gate is crossed for a given point p_j, we check for intersection with all gates $\mathcal{G}$. Then, if p_j

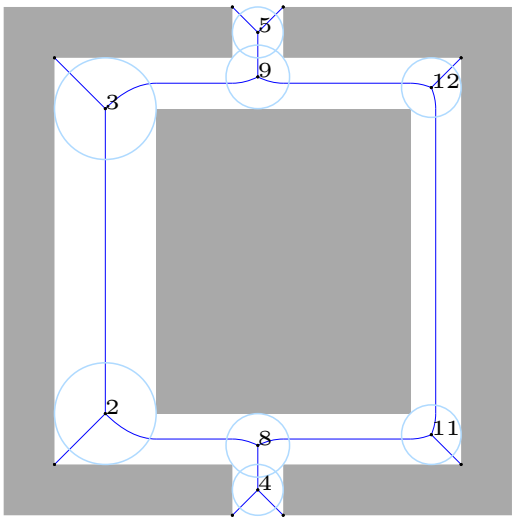


Figure 18: A scenario where the vertices of the ECM are used as gates. Each gate is defined by a vertex and its clearance, which is used as the radius of the disk.

lies in multiple gates, we take the gate which center lies closest to p_j to be the corresponding gate. The next step is to remove the loops in both sets of gates. A loop consists of a set of gates $\mathcal{G}_{first}$ followed by any gate g followed by $\mathcal{G}_{second}$ which equals $\mathcal{G}_{first}$ in reverse order. An example would be {5, 8, 7, 4, 7, 8, 5}, where $\mathcal{G}_{first} = \{5, 8, 7\}$ and $g = 4$. These loops are removed because the removal of a loop does not change the homotopic class of the path. But to be able to compare sets of gates to determine the type of detour, the presence of a loop can result in the incorrect classification of a detour. Consider a set of gates for the original path $\mathcal{G}_o$ = {2, 3, 4, 6} and a set of gates for the detour path $\mathcal{G}_d$ = {2, 3, 4, 5, 8, 7, 4, 7, 8, 5, 6}. When comparing these sets as is, they are not equal and thus the detour path will be incorrectly classified as a global detour. After removing the loop {5, 8, 7, 4, 7, 8, 5}, $\mathcal{G}_d = \mathcal{G}_o$ = {2, 3, 4, 6} and the path will be correctly classified as a local detour. There is another possible scenario that forms a loop: when only one other gate is crossed either at the start or end of the path. These loops will not be filtered out using our algorithm. Instead, when comparing the two sets of gates for equality, we also check whether they are the equal except for one added gate either at the start or end. Pseudocode for the algorithm can be found in Algorithm 3.

Unfortunately there still are cases where the original path and the detour path are in the same homotopic class but still cross a different set of gates. An example can be found in Fig. 20. In this scenario, the original path crosses gates {51, 50} and the detour path crosses {51, 53, 50}. This results in the classification of the detour path as a global detour, while it is in fact a local detour. Fortunately, the downsides of misclassification are minimal.

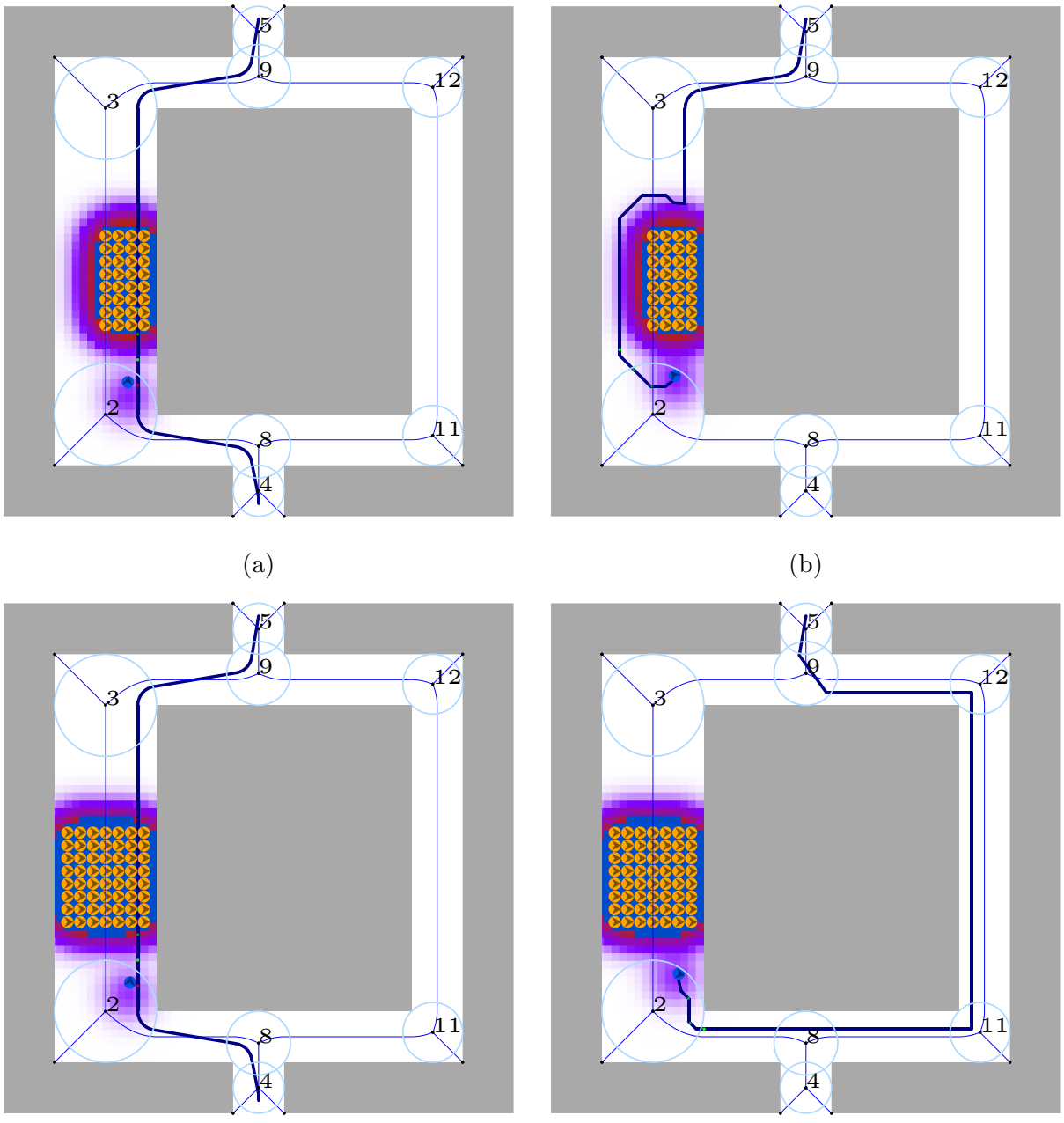


Figure 19: (a) Initial configuration. Gates crossed are: 2, 3, 9, 5. (b) Planned detour is local. Gates crossed are: 2, 3, 9, 5. (c) Initial configuration. Gates crossed are: 2, 3, 9, 5. (d) Planned detour is global. Gates crossed are: 2, 8, 11, 12, 9, 5

Algorithm 3 isGlobalDetour

Input. A set of points $\mathcal{P}_{original} = \{p_0, ..., p_n\}$ that forms the original path and a set of points $\mathcal{P}_{detour} = \{p_0, ..., p_n\}$ that forms the detour path
Output. If the detour path is a global detour

1: $G_{original} \leftarrow$ getGates($\mathcal{P}_{original}$)
2: $G_{detour} \leftarrow$ getGates($\mathcal{P}_{detour}$)
3: $G_{original} \leftarrow$ removeLoops($G_{original}$)
4: $G_{detour} \leftarrow$ removeLoops(G_{detour})
5: $globalDetour \leftarrow$ isEqual($G_{detour}, G_{original}$) or isOneOff($G_{detour}, G_{original}$)
6: return $globalDetour$

First, situations like this rarely occur, especially when trying to find the shortest path from A to B. Second, we only use this classification to repair the path and to add memory. We will discuss why the downsides of the misclassification are minimal in the respective sections.

4.5 Repairing global detours

After classifying the detour, global detours might need repairing in situations like in Figure 21a, which we will use as the example here. We will use the same gates used for classification of the detour to repair the path. First, we find the last gate G that is entered by π_{detour} twice. This is the gate corresponding to ECM vertex 9: G_9. Next, We find the first and last point on the indicative route that are within the area of this gate: π_{first} and π_{last}. Lastly, we connect π_{first} and π_{last}, removing all points that lie between them resulting in the repaired path $\pi_{repaired}$. The repaired path can be found in Figure 21b.

Regarding the misclassification of the type of detour: if a local detour is classified as a global detour, no gate G will be found and no repairs will be made.

4.6 Memory

For a global detour, in addition to repairing the path, we add a memory M to the agent to indicate the original path was blocked as explained in Section 3.4.2. Next, for each one of the agents memories $M \in \mathcal{M}$, we check if the detour intersects any of its cells $c_{blocked}$. If no intersections are found, the agent proceeds as normal. Otherwise, the agent is making a detour through a previously crowded section of the environment. When this scenario occurs, we want the agent to stick to either the current path or the planned detour.

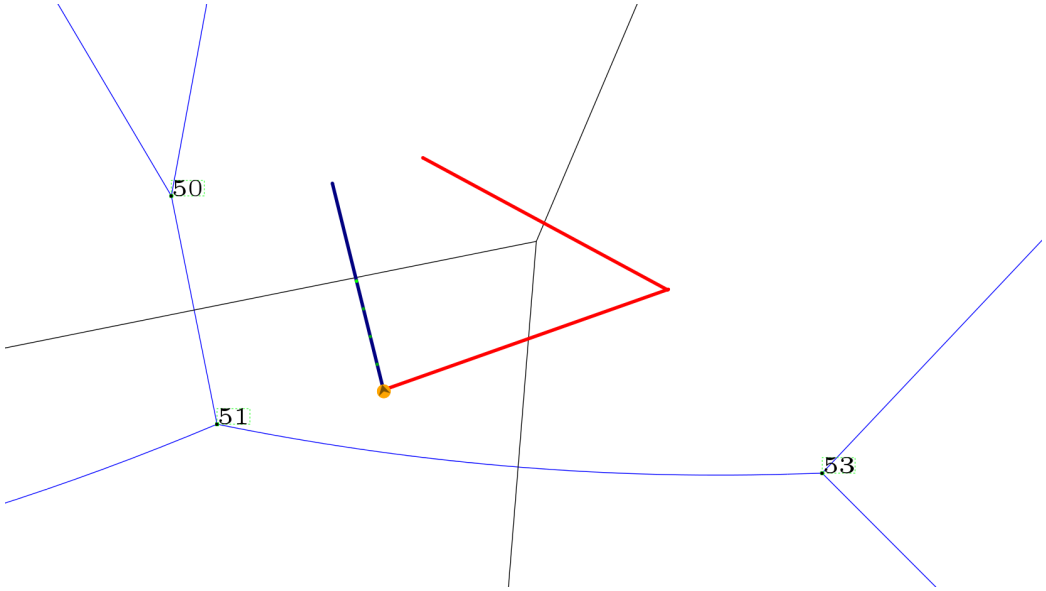
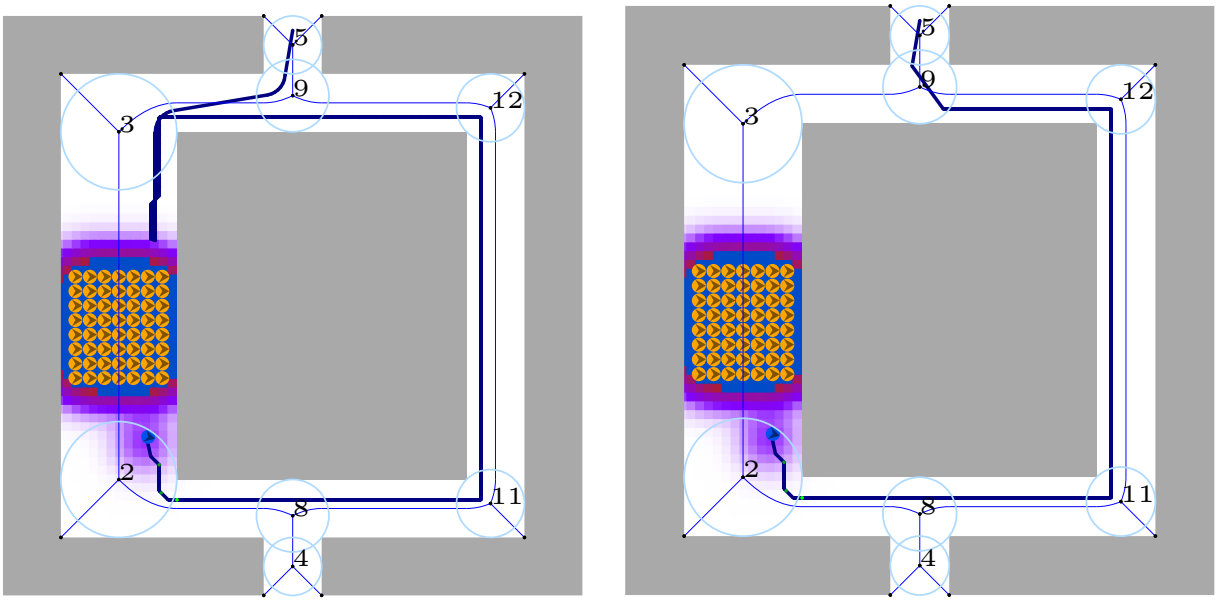


Figure 20: Scenario with ECM edges in blue and the Voronoi diagram of the ECM vertices in black. In this scenario, the detour (red path) will be incorrectly classified as a global detour. Note that the gates for all vertices are drawn but not visible due to the scale of the image.



(a) (b)

Figure 21: (a) The path after planning a detour to the density-based detour goal. (b) The repaired path by connecting the first and last point in the top gate.

4.6.1 Committing to a path

To choose between the current path and the planned detour path, we estimate the time it would take the agent to traverse both paths. The path that would take the least time to traverse is chosen as the path the agent has to commit to. After that, the agent is not allowed to make global detours until the decision gate is reached. Pseudocode of the algorithm can found in Algorithm 4. We define the *decision gate* in the following way: When a global detour is planned, it is planned to g_{detour} after which it connects to the original path. From that point on, the original path and the detour path are the same. This point from which both paths are the same is marked by a gate. This is the decision gate. The decision gate usually lies on a fork in the path, hence its name.

There are some scenarios in which this method cannot be used to find the decision gate, for example when there are no common gates between the original path and the detour path. This can occur when g_{detour} is the original goal position and the original path and the detour path both reach the goal through an entirely different path. In this case, we take the last gate of the detour path to be the decision gate.

To estimate the time to traverse a path, we start by estimating the current location of the block p_{block} if the block was not encountered in the current simulation step. We start with the average velocity of the block v_{block} and determine if the block is heading in the same or opposite direction as the agent. This is done by comparing v_{block} to the direction of π_{ind} at the location of the block as shown in the formula below. Note that if $l(v_{block}) = 0$ the cost is set to infinite. This signifies that the path is blocked and ensures the other path under consideration is chosen.

$$\Phi_{block} = \angle(v_{block}, p_{blocked_2} - p_{blocked_1})$$

We take $p_{blocked_2} - p_{blocked_1}$ here for the direction of the agent because it is an approximation of the average velocity of the agent while moving through the block. If the block lies around a sharp corner, the direction of the indicative route at $p_{blocked_1}$ and $p_{blocked_2}$ can differ a lot. In cases like these taking an average velocity is needed to produce the correct results. If $\Phi_{block} > \frac{\pi}{2}$, the block moves in the opposite direction and the cost is set to infinite.

Next we calculate the estimated distance the block has traveled:

$$l_{block} = v_{block} * (t - t_{blocked})$$

Lastly, we move $p_{blocked}$ either forwards along π_{ind} until we have moved it a distance equal to l_{block}. This results in the point p_{block}.

The next step is to calculate the time it would take for the agent to reach p_{block}, t_{block}. We calculate this by multiplying the agent's preferred velocity with the length of the indicative route between the agent and the estimated block location:

$$t_{block} = v_{pref} * \pi_{ind}(p_{agent}, p_{block}),$$

where $\pi_{ind}(p_{agent}, p_{block})$ is the subpath of π_{ind} from p_{agent} to p_{block}. The last step is to calculate the time it would take to move from the block to the decision gate, $t_{decision}$. We calculate this by multiplying the average block speed with the distance on the indicative route between the estimated location of the block and the first point in the decision gate:

$$t_{decision} = v_{block} * \pi_{ind}(p_{block}, p_{decision})$$

The final estimated time is calculated as follows:

$$t_{estimate} = t_{block} + t_{decision}$$

The path with the lowest value for $t_{estimate}$ is selected as the final path. The next step is to prevent the agent from planning a global detour until the decision gate is reached. We accomplish this by checking, for each attempted global detour, if the decision gate has been reached by the agent. If it has, the global detour is allowed. Otherwise, the agent continues its original path.

Algorithm 4 commitToPath

Input. A set of points $\mathcal{P}_{current} = \{p_0, ..., p_n\}$ that forms the current path, a set of points $\mathcal{P}_{detour} = \{p_0, ..., p_n\}$ that forms the detour path, the set of all memories $\mathcal{M}$ for the agent and the memory corresponding to the crossed block M_{detour}
Output.

1: $M_{current} \leftarrow$ crossedMemory$(\mathcal{P}_{detour}, \mathcal{M})$
2: $\mathcal{G}_{current} \leftarrow$ getGates$(\mathcal{P}_{current})$
3: $\mathcal{G}_{detour} \leftarrow$ getGates$(\mathcal{P}_{detour})$
4: $G_{decision} \leftarrow$ findDecisionGate$(\mathcal{G}_{current}, \mathcal{G}_{detour})$
5: $t_{current} \leftarrow$ EstimatePathTime$(\mathcal{P}_{current}, G_{decision}, M_{current})$
6: $t_{detour} \leftarrow$ EstimatePathTime$(\mathcal{P}_{detour}, G_{decision}, M_{detour})$
7: **if** $t_{detour} > t_{current}$ **then**
8: setPath$(\mathcal{P}_{detour})$
9: **else**
10: setPath$(\mathcal{P}_{current})$
11: stickToPath$(G_{decision})$

5 Experiments and results

In this section we show our implementation details, experimental setup and experiment results.

5.1 Implementation details

Our experiments were performed on an Intel Core i5-3470 @ 3.2 GHz with 8 GB of RAM memory. We used the existing UUCS framework[1] created by Kremyzas, van Toll and Geraerts. We implemented the MIRANDA algorithm within this framework for comparison. All of the parameters used in our experiments can be found in Table 1. Each block corresponds to one of the algorithms and each algorithm uses the parameters from its own block and the ones above it, with the exception of the d_{local} and β which are not utilized by ID^3.

	Parameter	Symbol	Value
MIRAN	Agent radius	r	0.24 m
	Agent's preferred velocity	v_{pref}	1.4 m/s
	Look-ahead distance	σ	5 m
	Sampling distance	d	1 m
MIRANDA	Cell size	l_{cell}	0.3 m
	Density threshold	ρ_t	1.4 agents/m^2
	Density weight scaling factor	ρ_s	5
	Local-area-radius	d_{local}	25 m
	Anchoring bias	β	1.05
	Candidate attraction point rejection limit	cap_{reject}	0.6
	Density grid update time	t_ρ	2 s
ID^3	Deadlock length	l_d	0.3 m
	Velocity grid update time	t_{vel}	2 s
	Angle deviation	Φ_{dev}	$\frac{1}{4}\pi$
	Speed deviation	$speed_{dev}$	0.5
	Vision replan cooldown	t_{vision}	2 s
	Precision	$precision$	0.2
	detour goal density threshold	ρ_d	0.7 agents/m^2

Table 1: Parameters used in our experiments

5.2 Scenarios

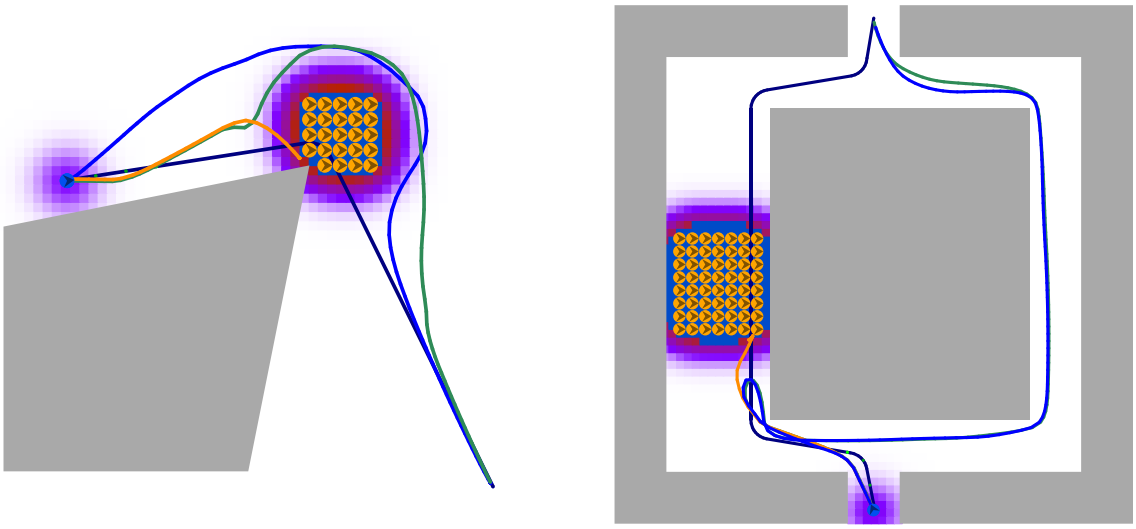
To test and compare our algorithm we created sets of scenarios as shown in the Figures below. Agents are colored differently depending on which algorithm is used: orange agents use MIRAN, green agents use MIRANDA and blue agents use our own extension ID^3. The indicative route for each

[1] https://ucrowds.com/documentation/core/

agent or group of agents is represented by a dark blue line. Paths for each agent were traced and are displayed in the same color as the agent.

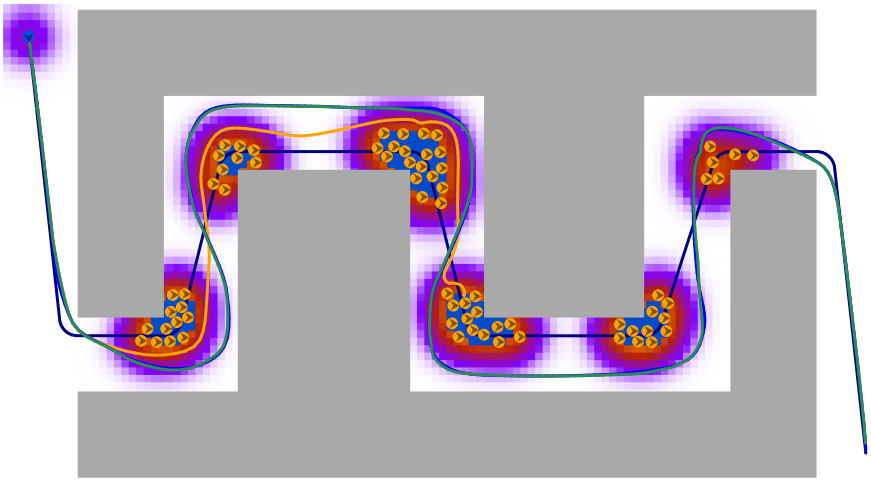
Our scenarios are divided into 3 sets and one single scenario: The first set is shown in Figure 22. This set contains scenarios where one agent moves (blue) and the other agents remain stationary (orange). The second set is shown in Figure 23 and consists of scenarios that were made to test the memory aspect of ID^3. The third set is shown in Figure 24 and consists of one environment with different configurations. The top row contains experiments where all agents have a preferred obstacle clearance of 0.5. For the experiments on the bottom row, the preferred obstacle clearance depends on the initial distance to the obstacle. For the left column, each agent goal has the same obstacle clearance as the distance to the obstacle corresponding to the starting position. For the right column, the goals are switched: The agent that starts with the most clearance to the obstacle has the goal position with the least amount of clearance to the obstacle. The goal of this is to force to agents to cross paths at some point to reach their goal and thus make it more difficult. We conducted these experiments for two cases: in the first case, all agents use MIRAN, for the other case we randomly made 20% of the agents use the ID^3 algorithm which resulted in 4 out of 19 agents using ID^3.

The last scenario, as shown in Figure 25 is a large city environment used for two purposes: to compare CPU times for each component of the ID^3 algorithm and to compare CPU times between MIRAN, MIRANDA and ID^3. Four regions are defined in this environment. Every simulation second, we spawn 15 agents with their start and goal position assigned randomly to these regions. To create a more realistic scenario, an agent's preferred obstacle clearance is randomly set to a value between 0.5 and 2.5 m. We also let the simulation run for 3 minutes before measuring the CPU time taken by each substep.

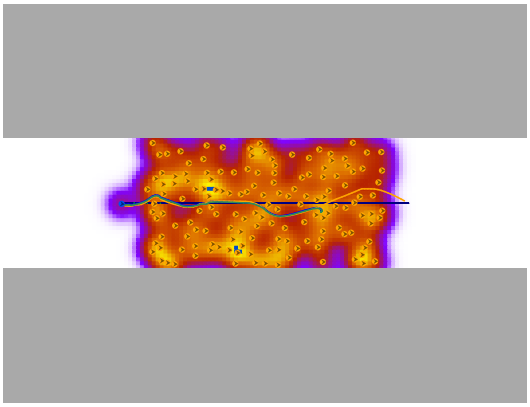


(a) Busy corner

(b) Blocked path

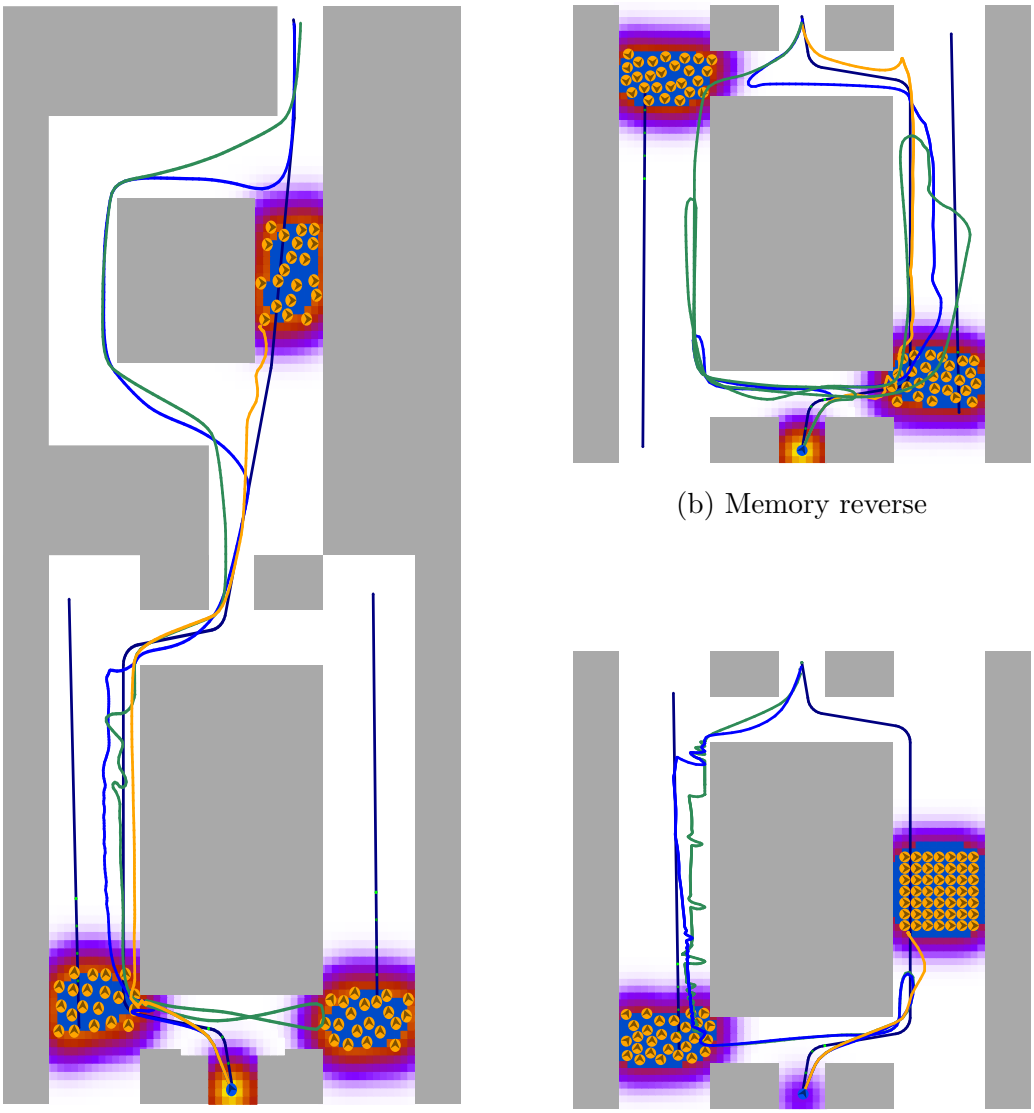


(c) Many corners



(d) Random

Figure 22: Single agent experiments. The indicative route is shown in dark blue. Traced paths are shown with the color corresponding to the agent's algorithm: orange = MIRAN, green = MIRANDA and blue = ID^3.

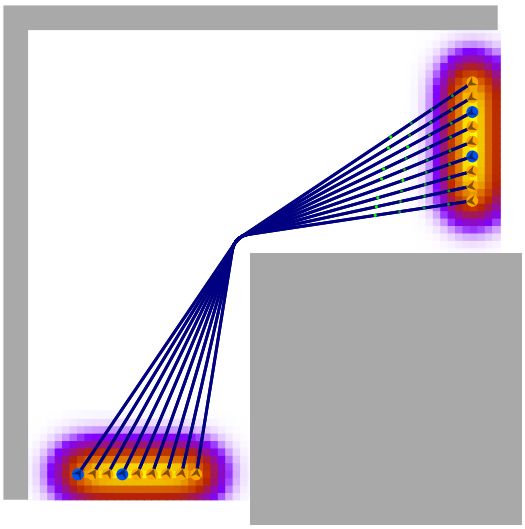


(a) Memory extended

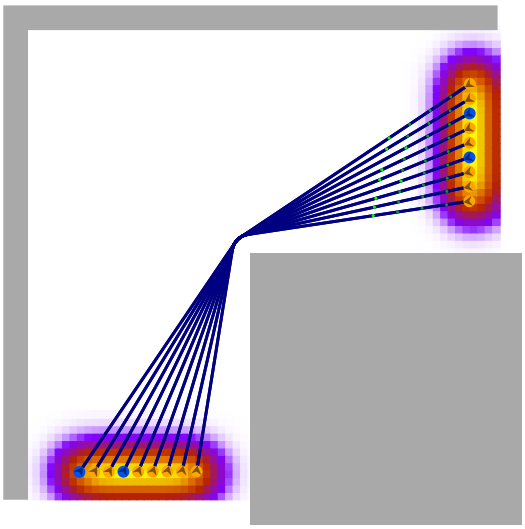
(b) Memory reverse

(c) Memory block

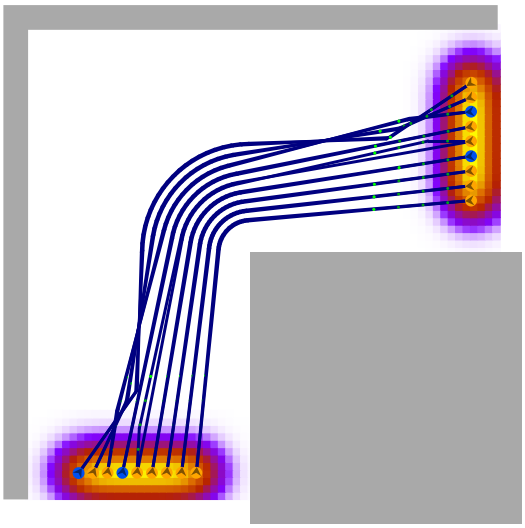
Figure 23: Memory experiments. The indicative route is shown in dark blue. Traced paths are shown with the color corresponding to the agent's algorithm: orange = MIRAN, green = MIRANDA and blue = ID^3.



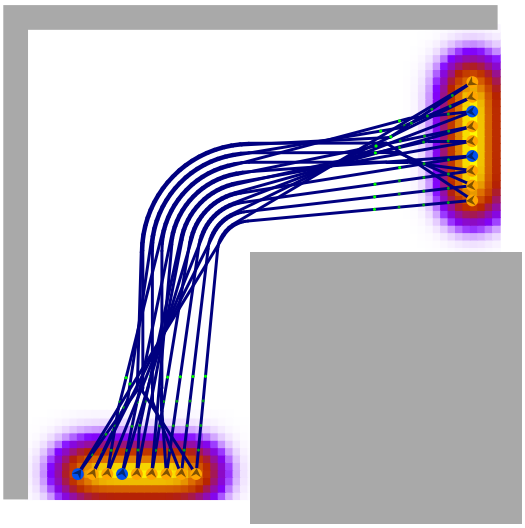
(a) Same obstacle clearance, normal goals



(b) Same obstacle clearance, reversed goals



(c) Dependent obstacle clearance, normal goals



(d) Dependent obstacle clearance, reversed goals

Figure 24: Corner experiments. For the top row, all agents have the same preferred obstacle clearance. For the bottom row, the preferred obstacle clearance depends on the initial distance to the obstacle. For the left column the goal position depends on the initial distance to the obstacle. For the right column, the goals are reversed.

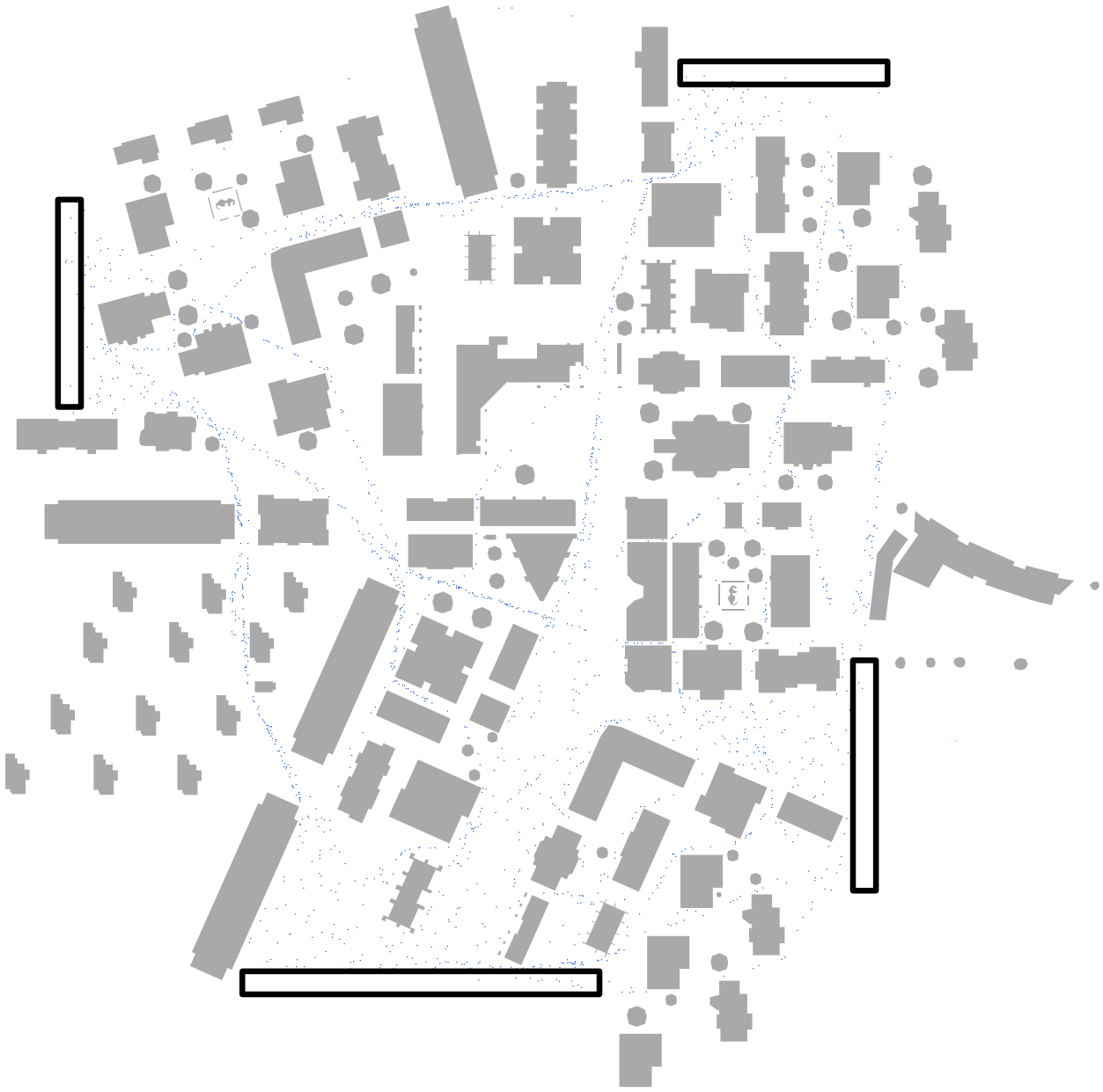


Figure 25: City experiment after 3 minutes. The black rectangles represent the start and goal regions.

5.3 Results

All results were averaged over 10 runs of the experiment. For the single agent and memory scenarios we measured the following statistics for the agent:

- Goal reached: Whether or not the agent reached their goal.
- Simulation time: The simulation time in seconds it takes for the agent to reach their goal.
- Path length: The path length of the traversed path in meters.
- Detours: The number of detours the agent has taken.

Results can be found in Tables 2 and 3.

For the corner scenarios we measured the simulation time shown in Table 4 and the combined path length for all agents shown in Table 5. Traversed paths for regular goals with and without dependent preferred obstacle clearance are shown in Fig. 26.

For the city environment CPU times were measured for 20 seconds or 200 simulation steps for MIRAN, MIRANDA and ID^3. Results can be found in Table 6. The following are the simulation substeps with a short description:

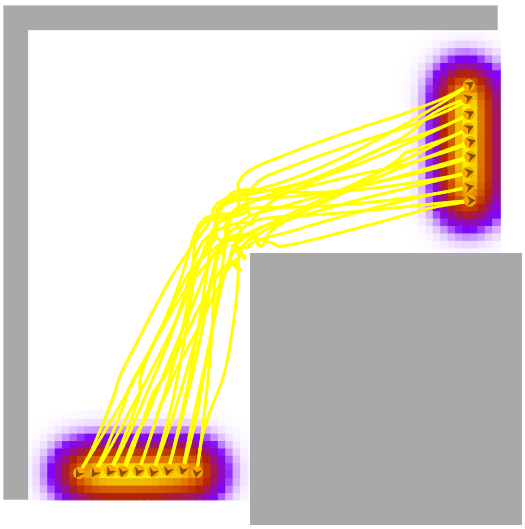
1. **Compute retractions**: Computes the retraction of agents onto the medial axis, used for further calculations.
2. **Update CQS**: Updates the Character Query Structure, a data structure that can answer nearest-neighbor queries for agents.
3. **Replanning**: For MIRAN, replans the path if none of the attraction points are visible.
4. **Periodic Replanning**: Periodically checks if replanning is necessary based on the agent's vision.
5. **Local replanning**: If needed, replans a path for an agent using MIRANDA or ID^3.
6. **Compute prefered velocities**: Computes preferred velocities for all agents.
7. **Compute actual velocities**: Computes all agents' new velocity, potentially with collision-avoidance
8. **Apply forces**: Applies the calculated forces to all agents.
9. **Update Positions**: Updates positions of all the agents.
10. **Execute scenario**: Spawns agents and calculates their paths, updates density and velocity sensors.

Scene	Method	Goal reached	Simulation time (s)	Path length (m)	Detours
Busy corner	MIRAN	No	-	-	-
	MIRANDA	Yes	20.2	26.7	1
	ID^3	Yes	20.1	26.9	1
Blocked path	MIRAN	No	-	-	-
	MIRANDA	Yes	30.8	40.5	1
	ID^3	Yes	30.9	40.7	1
Many corners	MIRAN	No	-	-	-
	MIRANDA	Yes	64.2	86.1	1
	ID^3	Yes	64.0	86.2	1
Random	MIRAN	Yes	20.1	23.1	0
	MIRANDA	No	-	-	-
	ID^3	No	-	-	-

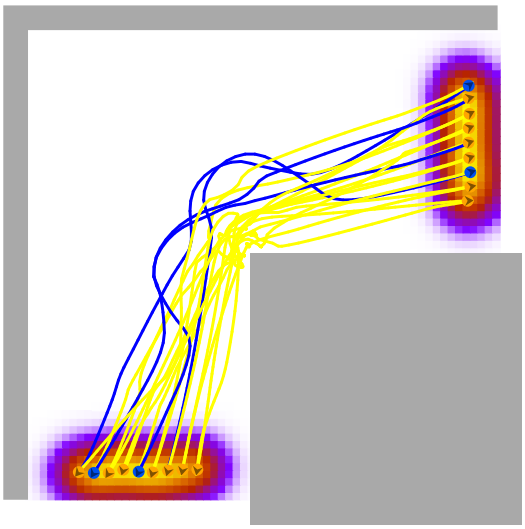
Table 2: Single agent experiment results

Scene	Method	Goal reached	Simulation time (s)	Path length (m)	Detours
Memory reverse	MIRAN	Yes	61.3	26.4	0
	MIRANDA	Yes	72.8	86.4	10
	ID^3	Yes	60.4	45.3	2

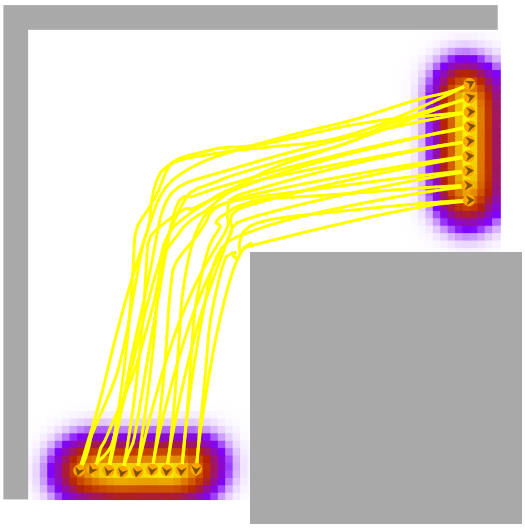
Table 3: Memory experiment results



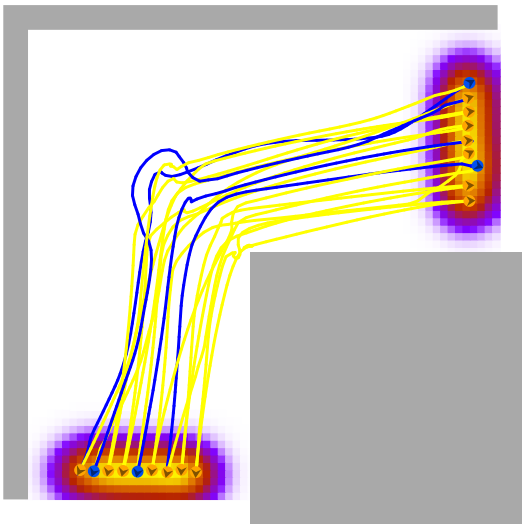
(a) MIRAN, same obstacle clearance



(b) ID^3, same obstacle clearance



(c) MIRAN, dependent obstacle clearance



(d) ID^3, dependent obstacle clearance

Figure 26: Traversed path for the corner experiments with normal goals. The first row contains results for the experiments with the same obstacle clearance, the second row contains experiments with dependent obstacle clearance. The first column contains results for MIRAN, the second column contains results for ID^3.

		goal			
		normal		reverse	
		MIRAN (ms)	ID^3 (ms)	MIRAN (ms)	ID^3 (ms)
obstacle clearance	normal	30.1	24.8	31.1	25.5
	dependent	19.4	22.0	22.5	21.6

Table 4: Corner experiment simulation time

		goal			
		normal		reverse	
		MIRAN (m)	ID^3 (m)	MIRAN (m)	ID^3 (m)
obstacle clearance	normal	394.4	397.4	403.2	406.8
	dependent	395.3	409.8	414.8	415.3

Table 5: Corner experiment combined path lengths

	MIRAN (ms)	MIRANDA (ms)	ID^3 (ms)
Compute retractions	0.13	0.08	0.08
Update CQS	0.04	0.04	0.03
Replanning	0.02	0.02	0.01
Periodic replanning	0.02	0.02	88.68
Local replanning	0.02	468.16	9.26
Compute preferred velocities	3.47	4.45	3.15
Compute actual velocities	1.20	1.17	1.14
Apply forces	0.02	0.2	0.02
Update positions	37.67	36.32	37.51
Execute scenario	0.09	4.61	9.37
total	42.68	515.07	149.25

Table 6: City experiment results

5.4 Experimental conclusions

From these experiments we can draw a number of conclusions:

- The single agent tests show no significant improvements regarding time-efficiency compared to MIRANDA. We do see some improvements regarding realism when looking at the traversed paths, especially for the busy corner scenario. This stems primarily from the addition of the vision-based deadlock detection, which was created with human-like obstacle avoidance in mind.

- The random experiment shows that there are situations in which an improved algorithm like ID^3 not only is unnecessary, but will not reach the goal where algorithms like MIRAN do. For both MIRANDA and

ID^3, a detour is taken that leads the agent into a few stationary agents. The density of the corresponding cells does not exceed the threshold and therefore no additional detours are planned and the agent gets stuck.

- For most of the memory experiments, we see a significant reduction of the time it takes for an agent to reach its goal. We also see that the switching of paths as is seen for MIRANDA is not present for ID^3.

- For the memory extended scenario, we see a slight increase in simulation time. This is most likely caused by the detour planned at the end, where, for the last section of the path, we can see that the agent using ID^3 first heads towards a point right behind the deadlock, and the agent with MIRANDA goes straight towards the original goal. Even though this is a minor downside of the density-based detour goals, we see that using the density-based detour goals will reduce the CPU time used since the detour goal is closer to the agent. This difference only increases for larger, more realistic scenarios.

- For the corner experiments, we see a reduction in time taken to reach the goal for a scenario with some agents using ID^3 in comparison for the scenario with only agents using MIRAN for the experiments where the obstacle clearance was the same for each agent. For the experiments where the obstacle clearance was dependent on the initial distance from the obstacle, we see an increase in simulation time for the normal goals and a slight decrease in time for the reverse goals. This might indicate that the use of ID^3 will provide a greater decrease in simulation time for more complex scenarios.

- For the city experiment, we see an increase in total CPU time for ID^3 compared to MIRAN but a decrease compared to MIRANDA. This decrease is mostly due to the elimination of the anchoring bias, which calculates an A* path for multiple sample points which is expensive.

- We also see that the simulation substep responsible for the vision-based deadlock detection is responsible for over 50% of the total CPU time of the algorithm and would be an important target for optimizations to simulate more agents in real-time.

6 Conclusions

This section provides a summary of findings for this thesis. We start with a summary of observations done throughout this thesis. We then show the conclusions drawn from the experiments and finish with our contribution.

6.1 Summary

In this thesis, we have shown that there are still some issues that are not solved by current algorithms regarding high-density crowd simulation. Examples of still occurring problems are the underutilization of available space and the absence of memory. We have tried to improve on some of these problems and made the following observations:

- Vision-based deadlock detection gives an agent more time to evade a deadlock and makes paths look more realistic.
- There is a need for accounting for flow when planning detours to prevent unnecessary detours.
- The method MIRANDA uses for determining the detour goal is inconvenient and should be improved.
- Determining the type of detour has multiple uses: it can help determine if paths need repairs and if there is a need for the addition of memory to prevent continuous switching between two paths.

Our algorithm was created with these observations in mind. We did some experiments to look at the time-efficiency of the paths created by our algorithm.

In conclusion, the experiments show that, in most cases, there was an improvement regarding the time which an agent needs to reach their goal. In cases where there was no improvement on the time, there was often an improvement on the realism of the path.

6.2 Contribution

Our contribution is the algorithm Improved Deadlock Detection and Detours or ID^3 as an extension for MIRAN. This extension improves the deadlock detection by adding vision-based deadlock detection that will detection detours earlier than previous methods. The algorithm also accounts for the flow of agents, so agents which move in the same direction will not plan a detour around each other. The detours are based on the MIRANDA algorithm, which uses a density grid to capture crowded points on a map. These detours are improved by determining the detour goal by sampling density values and detecting what type of detour is planned. Detours where an agent can go around a block locally are called local detours and detours that take an

entirely different path are called global detours. For global detours, double sections in paths emerge because the detour goal will be right behind the detour. We repair these paths by removing these double sections using gates. We also use memory to prevent continuous switching between different paths.

7 Future work

Our contribution provides a method that is more suitable for simulation of high-density crowds. There still are many possibilities for future work. In this section, we discuss some possible improvements and extensions to our work.

7.1 Perfect detection of detour type

Since our gate-based method does not always produce the correct results regarding the classification of the type of detour, a possible improvement would be to replace this method with a robust method that can determine whether or not paths belong to the same homotopic class. One possible method would be to project all points of a path onto the medial axis and determine if both projected paths belong to the same homotopic class. It is important, though, to carefully weigh the advantages of a robust method against the probable increase in computation cost.

7.2 Estimated density grid

One issue that is very apparent in scenarios such as the ones part of our corner scenario where there are two opposing streams is that the agent plans its detour using the current information on density that is available. The problem with this approach is that the agents forming the dense areas move away, therefore the area can be traversed again but the agent will still go around it. One possible solution for this problem is to use a density grid which estimates density values for the future. Another possible solution would be to check regularly whether or not a planned detour is still necessary, and replan if needed. We do predict that this method will come with a significant increase in computation time depending on the implementation.

7.3 Navigation through deadlocks

Another element that is missing from our method is the ability for agents to navigate through a dense crowd. This can be useful in scenarios where there is no other option than to navigate between agents, in for example a bar or concert like scenario. We recommend using a method like the one created by Stüvel et al. [32].

Another option would be to use the composite agent concept by Yeh et al. [41]. In a scenario where an agent needs to pass through a line of agents, a proxy repulsing other agents could be placed in the line to create room for the agent to pass through. It could also be used to improve behavior in scenarios like our corner scenario (Fig. 24). A proxy could be used to grant one of the two groups access.

7.4 Lane forming

Lane forming behavior could also be added to the algorithm. The method by Goethem et al. [38] that is already implemented into the framework might provide a solution here. It would require some testing to determine if the different components of the method cause conflicting behavior.

References

[1] Azahar, M. A. B. M., Sunar, M. S., Daman, D., and Bade, A. Survey on real-time crowds simulation. In *International Conference on Technologies for E-Learning and Digital Entertainment* (2008), Springer, pp. 573–580.

[2] Bloemheuvel, M. Creating dynamic and density dependent indicative routes for crowd simulation, 2014. (Master's thesis). Utrecht University.

[3] Chenney, S. Flow tiles. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2004), Eurographics Association, pp. 233–242.

[4] Dias, C., and Lovreglio, R. Calibrating cellular automaton models for pedestrians walking through corners. *Physics Letters A* (2018).

[5] Dijkstra, E. W. A note on two problems in connexion with graphs. *Numerische mathematik 1*, 1 (1959), 269–271.

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

[7] Geraerts, R. Planning short paths with clearance using explicit corridors. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on* (2010), IEEE, pp. 1997–2004.

[8] Geraerts, R., and Overmars, M. H. The corridor map method: A general framework for real-time high-quality path planning: Research articles. *Comput. Animat. Virtual Worlds 18*, 2 (may 2007), 107–119.

[9] Guy, S. J., Chhugani, J., Curtis, S., Dubey, P., Lin, M., and Manocha, D. Pledestrians: a least-effort approach to crowd simulation. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics symposium on computer animation* (2010), Eurographics Association, pp. 119–128.

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

[11] Hashimoto, K., Yoshimi, T., Mizukawa, M., Ando, Y., and Takeuchi, K. A study of collision avoidance between service robot and human at corner—analysis of human behavior at corner. In *Ubiquitous Robots and Ambient Intelligence (URAI), 2013 10th International Conference on* (2013), IEEE, pp. 383–384.

[12] He, G.-q., Jin, Y., Chen, Q., Liu, Z., Yue, W.-h., and Lu, X.-j. Shadow obstacle model for realistic corner-turning behavior in crowd simulation. *Frontiers of Information Technology & Electronic Engineering 17*, 3 (2016), 200–211.

[13] Helbing, D., Farkas, I. J., Molnár, P., and Vicsek, T. Simulation of pedestrian crowds in normal and evacuation situations. *Pedestrian and evacuation dynamics 21*, 2 (2002), 21–58.

[14] Helbing, D., and Johansson, A. Pedestrian, crowd and evacuation dynamics. In *Encyclopedia of complexity and systems science*. Springer, 2009, pp. 6476–6495.

[15] Helbing, D., and Molnár, P. Social force model for pedestrian dynamics. *Physical review E 51*, 5 (1995), 4282.

[16] Helbing, D., and Mukerji, P. Crowd disasters as systemic failures: analysis of the love parade disaster. *EPJ Data Science 1*, 1 (2012), 7.

[17] Hillebrand, A., Hoogeveen, H., and Geraerts, R. Comparing different metrics quantifying pedestrian safety.

[18] Jaklin, N., Cook, A., and Geraerts, R. Real-time path planning in heterogeneous environments. *Computer Animation and Virtual Worlds 24*, 3-4 (2013), 285–295.

[19] Karamouzas, I., Bakker, J., and Overmars, M. H. Density constraints for crowd simulation. In *Games Innovations Conference, 2009. ICE-GIC 2009. International IEEE Consumer Electronics Society's* (2009), IEEE, pp. 160–168.

[20] Karamouzas, I., Geraerts, R., and Overmars, M. Indicative routes for path planning and crowd simulation. In *Proceedings of the 4th International Conference on Foundations of Digital Games* (2009), ACM, pp. 113–120.

[21] Kavraki, L. E., Svestka, P., Latombe, J.-C., and Overmars, M. H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation 12*, 4 (1996), 566–580.

[22] Krijnen, T., Beetz, J., and de Vries, B. Airport schiphol.

[23] Kuffner, J. J., and LaValle, S. M. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on* (2000), vol. 2, IEEE, pp. 995–1001.

[24] Narain, R., Golas, A., Curtis, S., and Lin, M. C. Aggregate dynamics for dense crowd simulation. In *ACM Transactions on Graphics (TOG)* (2009), vol. 28, ACM, p. 122.

[25] Pelechano, N., Allbeck, J. M., and Badler, N. I. Controlling individual agents in high-density crowd simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2007), Eurographics Association, pp. 99–108.

[26] Pelechano, N., Allbeck, J. M., and Badler, N. I. Virtual crowds: Methods, simulation, and control. *Synthesis Lectures on Computer Graphics and Animation 3*, 1 (2008), 1–176.

[27] Richmond, P., and Romano, D. M. A high performance framework for agent based pedestrian dynamics on gpu hardware. *Proceedings of EUROSIS ESM 20* (2008), 27–29.

[28] Rojas, F. A., Park, J. H., and Yang, H. S. Group agent-based steering for the realistic corner turning and group movement of pedestrians in a crowd simulation. *Proceedings of Computer Animation and Social Agents (CASA 2013)* (2013).

[29] Seń, R. The detouring algorithm, 2017. (Small project). Utrecht University.

[30] Stüvel, S. A. *Dense Crowds of Virtual Humans*. PhD thesis, Utrecht University, 2016.

[31] Stüvel, S. A., de Goeij, M., van der Stappen, A. F., and Egges, A. An analysis of manoeuvring in dense crowds. In *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games* (2015), ACM, pp. 85–90.

[32] Stüvel, S. A., Magnenat-Thalmann, N., Thalmann, D., van der Stappen, A. F., and Egges, A. Torso crowds. *IEEE transactions on visualization and computer graphics 23*, 7 (2017), 1823–1837.

[33] Thalmann, D., and Musse, S. R. *Crowd simulation*. Wiley Online Library, 2007.

[34] Treuille, A., Cooper, S., and Popović, Z. Continuum crowds. In *ACM Transactions on Graphics (TOG)* (2006), vol. 25, ACM, pp. 1160–1168.

[35] Tsai, T.-Y., Wong, S.-K., Chou, Y.-H., and Lin, G.-W. Directing virtual crowds based on dynamic adjustment of navigation fields. *Computer Animation and Virtual Worlds 29*, 1 (2018).

[36] Ulicny, B., and Thalmann, D. Crowd simulation for interactive virtual environments and vr training systems. In *Computer Animation and Simulation 2001*. Springer, 2001, pp. 163–170.

[37] Van Den Berg, J., Guy, S. J., Lin, M., and Manocha, D. Reciprocal n-body collision avoidance. In *Robotics research*. Springer, 2011, pp. 3–19.

[38] Van Goethem, A., Jaklin, N., Cook, I., Geraerts, R., et al. On streams and incentives: A synthesis of individual and collective crowd motion, 2015.

[39] van Toll, W., Jaklin, N., and Geraerts, R. Towards believable crowds: A generic multi-level framework for agent navigation.

[40] Van Toll, W. G., Cook, A. F., and Geraerts, R. Real-time density-based crowd simulation. *Computer Animation and Virtual Worlds 23*, 1 (2012), 59–69.

[41] Yeh, H., Curtis, S., Patil, S., van den Berg, J., Manocha, D., and Lin, M. Composite agents. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2008), Eurographics Association, pp. 39–47.

[42] Zhang, J., Klingsch, W., Rupprecht, T., Schadschneider, A., and Seyfried, A. Empirical study of turning and merging of pedestrian streams in t-junction. *arXiv preprint arXiv:1112.5299* (2011).