

TDK-dolgozat
Ficsor Attila

Flow-field network-based analysis of manufacturing layout

Ficsor Attila, Pannon Egyetem

HUN-REN-PE Komplex Rendszerek Figyelemmel Kísérése Kutatócsoport

Témavezető: Dr. Abonyi János egyetemi tanár

Contents

1	Introduction	5
2	Flow-field network-based analysis algorithm methodology	9
2.1	Digitalization and k-NN graph-based description of the layout	9
2.2	Calculation of geodesic distances based on the graph of the layout	12
2.3	Flow-field calculation and visualization	12
2.4	Simulation-based path finding	13
2.5	Proper orthogonal decomposition analysis	14
2.6	Visualizing bottlenecks in the layout	16
3	Algorithm implementation and real-world applications	17
3.1	Implementation	17
3.1.1	Python libraries and packages utilized in the algorithm	17
3.1.2	Digitalization of the layout	18
3.1.3	k-NN graph-based description of the layout	19
3.1.4	Calculation of geodesic distances based on the graph of the layout .	20
3.1.5	Flow-field calculation	20
3.1.6	Simulation-based path finding	20
3.1.7	Proper orthogonal decomposition analysis of the modes	21
3.1.8	Application of bottleneck analysis	22
3.2	Real-world application of the pathfinding and analysis algorithm	23
3.2.1	Operator 4.0 lab	23
3.2.2	Mini automated guided vehicles	24
3.3	Runtime test of the algorithm on different grid sizes	24
4	Conclusion	26

Flow-field network-based analysis of manufacturing layout

Author: Ficsor Attila

ELKH-PE Complex Systems Monitoring Research Group

Supervisor: Dr. János Abonyi

Optimizing material flow is a crucial task in manufacturing processes, where efficiency can have a significant impact on production rate and quality. Identification of the optimal path between supply and delivery points is a non-trivial problem. To address this issue, I propose a flow-field-based optimization technique that can be used to find the optimal path in manufacturing processes. The fundamental concept behind the flow-field-based path-finding technique is to find optimal paths to every potential point and aggregate the direction vectors in a flow-field. The flow-field-based approach is often used in situations where there are a huge amount of agents required to be moved. Agents follow a vector based on the calculated cost field, and as a result, a quasi-optimal path is discovered. In this work, a novel approach is used, where image processing is used, in order to dynamically load layouts as images, and construct a mesh grid based on the content of the input image. The points are removed that were too close to the not-allowed regions using a nearest-neighbor approach. The k-NN graph is constructed, in order to traverse it more efficiently, and the geodesic distances are calculated as shortest paths between all pairs of vertices. Furthermore, Proper Orthogonal Decomposition (POD) analysis is applied to examine the modes, enabling the identification of bottlenecks through the evaluation of energy distributions. This innovative method offers a comprehensive understanding of the layout, effectively highlighting critical regions and streamlining optimization efforts. To demonstrate the effectiveness of the proposed technique, it is applied to a real-world example connected to a manufacturing environment of the Operator 4.0 laboratory at the University of Pannonia. The results demonstrate that the proposed technique can efficiently explore critical paths and regions of the shop floor. The technique can be used in lean projects to optimize material flow, workstation layouts, and supply and delivery directions.

Keywords: Flow-Field pathfinding, k-NN graph, POD analysis, SVD, bottleneck analysis

1 Introduction

In manufacturing processes, optimizing material flow is an important task that directly affects the efficiency of production rates and the quality of products [1]. The simultaneous optimization of the manufacturing process, the layout, and the material flow is a complex problem that requires a more comprehensive approach.

Analysis of the common paths of the layout can help us understand the bottlenecks and problems in the interior design of the manufacturing facility. By combining the insights gained from path planning algorithms and layout analysis, we can find a solution that balances these different factors and leads to considerable improvements in manufacturing processes.

The search for the optimal path involves complex decision-making processes that require careful consideration of various factors, including time, cost, and quality. In this context, finding a solution that balances these factors can lead to significant improvements in manufacturing processes. This work highlights the importance of optimizing material flow and the challenges involved in identifying the optimal path in manufacturing processes [2].

Path planning algorithms are an essential component of material flow optimization in manufacturing processes. Traditional algorithms, such as brute force search [3], Dijkstra's algorithm, A* [4], and genetic algorithms [5], have been widely used in this field. However, these algorithms have limitations in their ability to handle the complexities of modern manufacturing processes and layouts. Here are some of the downsides of each algorithm:

- Brute force search involves considering every possible path, which can become computationally expensive and infeasible for larger manufacturing environments. The brute force algorithm can be effective for smaller environments or when dealing with a limited number of paths, but it quickly becomes impractical for larger manufacturing facilities where the number of paths increases exponentially [6].
- Dijkstra's algorithm and A* algorithm are graph-based algorithms. Dijkstra's algorithm is a single-source shortest path algorithm that finds the shortest path between a source node and all other nodes in a graph, while Dijkstra's algorithm gives an optimal path, it can be really expensive computationally. A* algorithm is a heuristic search algorithm that estimates the total cost of a path from the start node to the end node, but in some cases, the found path isn't optimal. Although these algorithms are effective in many cases, these algorithms can be sub-optimal for larger graphs [4].
- Genetic algorithms are a powerful optimization technique that mimics the process

of natural selection. However, they may require a significant amount of computational resources and may not be effective in handling the complexities of modern manufacturing processes. Additionally, the results of genetic algorithms may not always be consistent or reproducible, making it difficult to apply this approach in real-world manufacturing processes [5].

Researchers have proposed innovative approaches such as the flow-field-based optimization approach [7], which has shown to be more effective in identifying optimal paths in complex manufacturing environments. The flow-field pathfinding algorithm is often used in video games, mostly real-time strategy ones, where we have to move many agents at the same time [8]. In this context, traditional path-finding algorithms can be too computationally expensive, as calculating the path for every single agent scales the cost exponentially. Flow-Field pathfinding solves this problem, by computing the vectors for the full map. While this can be computationally expensive too, over a number of agents this approach becomes favorable to classic pathfinding methods, as the calculation's cost stays the same for any number of agents. The flow-field-based optimization approach offers an alternative to traditional optimization techniques and provides a more efficient way of optimizing material flow in modern manufacturing processes.

Flow-field pathfinding is a pathfinding technique that utilizes a grid-based approach to navigate through a map, where each cell is assigned a vector representing the direction of the shortest path to a goal. This vector is computed by first creating a cost field, which assigns each cell a cost value representing the distance to the goal. This cost field is then used to generate a gradient field, where each cell is assigned a vector pointing towards the neighboring cell with the lowest cost. The algorithm used to compute these fields is typically a graph traversal algorithm like Dijkstra's or A*.

Previous research has explored various techniques that I could utilize in my work. One such technique is the pre-computed flow-field map approach, which has shown promising results in improving pathfinding efficiency [7]. Another relevant approach is the combination of flow-fields and the A* algorithm to generate optimal paths for entities in complex environments [9]. This approach uses Voronoi diagrams to identify the free space for movement and then assign flow vectors to each cell of the diagram to capture the direction and magnitude of the movement. The A* algorithm is then used to search for the optimal path through the flow-field.

By building upon the strengths of previous research, the proposed technique offers a unique solution to the challenges of optimizing material flow in modern manufacturing processes. To address the challenges of material flow optimization in modern manufacturing processes, a novel approach has been developed based on flow-field pathfinding.

The algorithm that has been developed involves several steps to analyze the flow-field

in a given layout. The first step is to digitalize the layout in binary and construct a grid of points based on the initial layout, as shown in Fig. 1a and b. The next step is to remove nodes that are too close to objects in not allowed regions, as depicted in Fig. 1c. The resulting points are then used to construct a graph using the k-NN algorithm, illustrated in Fig. 1d. The distances between every node in the graph are calculated. The flow-field is then calculated, which is shown in Fig. 1e. Simulation is run in order to showcase the algorithm as shown in Fig. 1g. Finally, the flow patterns and their underlying dynamics are analyzed, as depicted in Fig. 1f.

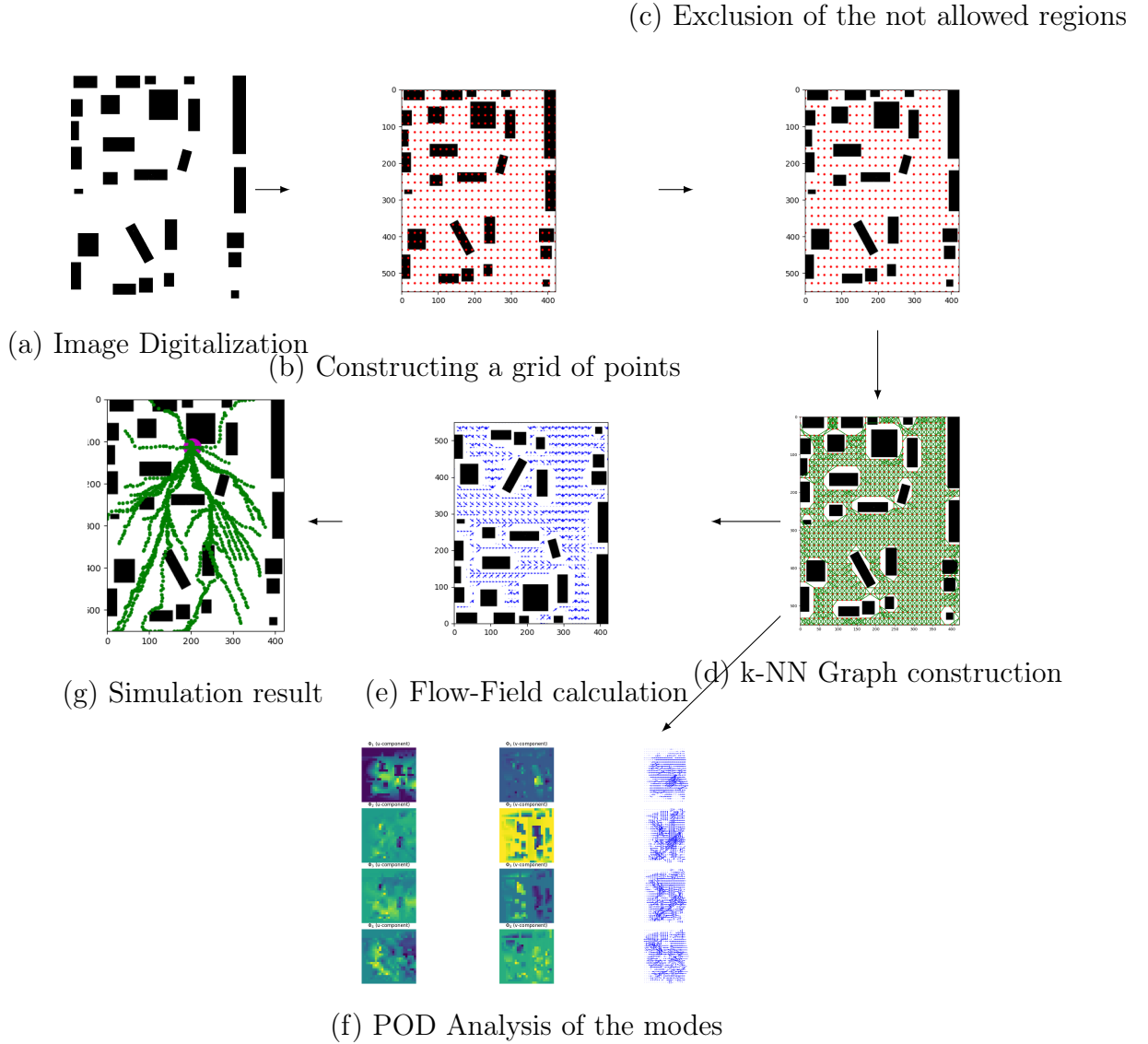


Figure 1: Roadmap of the analysis pipeline.

By analyzing the layout using this approach, potential bottlenecks and obstacles that may obstruct material flow can be identified. After identifying the flow directions, the layout of the system can be designed in a way to minimize the distance traveled by the materials and ensure that the materials are flowing in the right direction. The layout

analysis helps to identify the optimal paths between workstations or supply and delivery points and to optimize the material handling system to ensure that it is as efficient as possible, leading to improved production efficiency and product quality. By using this approach, the aim of this algorithm is to improve the efficiency and speed of material flow optimization in modern manufacturing processes.

The structure of the work is as follows: The theoretical background for the flow-field network-based analysis algorithm is introduced in Section 2. Section 2.1 describes the digitalization of the layout and grid construction, while Section 2.2 covers the K-NN graph-based description of the layout. The calculation of geodesic distances in the graph is presented in Section 2.3, followed by the flow-field calculation in Section 2.4. The simulation-based pathfinding is explained in Section 2.5, and the analysis of flow patterns and their underlying dynamics are discussed in Section 2.6 and Section 2.7. In Section 3, I present the implementation of the algorithm, real-world application, and runtime tests. Finally in Section 4 concluding remarks are given

2 Flow-field network-based analysis algorithm methodology

In this section, the methodology used in the approach to optimize material flow in manufacturing processes is discussed. The process begins with a description of digitalizing the layout using various techniques, followed by generating a grid of nodes representing potential paths for the agents. Subsequently, the removal of nodes in close proximity to not-allowed regions is discussed, which prevents collisions between agents and obstacles. The generation of the graph, calculation of flow-fields, and analysis of the modes are also covered in this section.

2.1 Digitalization and k-NN graph-based description of the layout

Optimizing material flow in manufacturing processes requires a thorough understanding of the layout and obstacles within the facility.

Digitalization of the layout is a crucial first step in achieving this, and it can be accomplished using various techniques, such as blueprint analysis from CAD files [10], LiDAR [11], and so on. Accurate digitalization of the layout is paramount in achieving effective optimization, as it enables the analysis of the shop floor to identify potential bottlenecks and other issues. In this context, the analysis of the layout forms the basis of the algorithm.

Using the digitalized layout a grid of points is generated based on the digitalized image using a sampling method to represent the nodes in the graph. The size and density of the grid are determined based on the input image size and the desired node density. The grid represents a set of potential paths that the agents can follow to navigate through the allowed regions in the image. However, some of the nodes in the grid are located too close to the not-allowed region in the binary mask, which can lead to collisions between the agent and the obstacles. To prevent this, a node removal process is implemented. This process involves the application of the nearest neighbors algorithm to identify nodes within a certain radius of any pixel in the not-allowed region. These nodes are then removed from the node-set, leaving a clear margin around the not-allowed region.

To demonstrate the effect of the node removal process, two images of the grid points are generated. The first image shows the grid before node removal: Figure 2a, while the second image shows the grid after node removal: Figure 2b. The resulting image after node removal shows a clear margin around the not-allowed region, indicating that the nodes near it had been successfully removed.

The above-mentioned methodology ensures that the generated path does not come too close to the not-allowed region, thus preventing collisions between the agent and the obstacles.

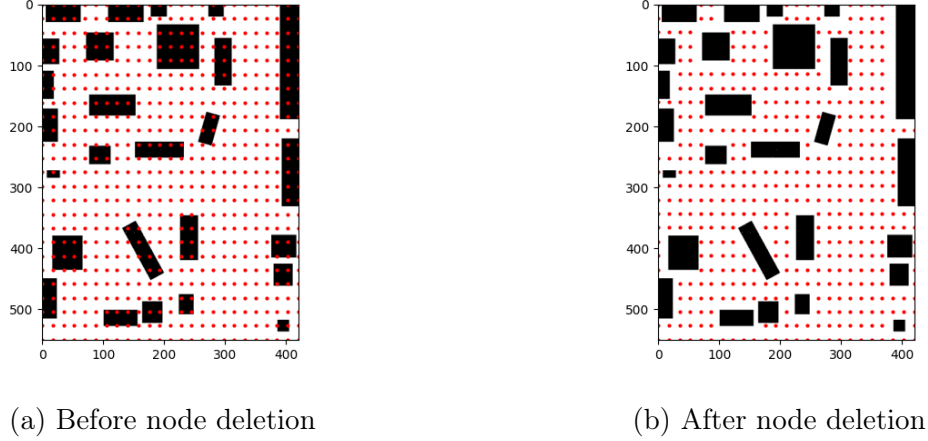


Figure 2: Sampling and deletion of the nodes

The following step in the algorithm is constructing a graph, based on the sampled and filtered points. To achieve this, a well-established machine learning technique for finding the k -nearest neighbors of a given data point is utilized. The choice of using k -NN is motivated by its ability to effectively capture the local structure of the data, while also providing a flexible and scalable approach for graph construction. Using the capabilities of the k -NN algorithm, the method is able to analyze the layout and identify critical regions that can be further optimized [12].

To construct a graph representation of the grid, the points generated by the sampling method employed in the last section are used, and each point in the grid is assigned k -nearest neighbors. These neighbors are then used to create nodes and edges in the graph. The resulting k -NN graph visually displays each point and its connections to its k -nearest neighbors. The resulting k -NN graph is a useful representation of the grid that can be used for further analysis and manipulation of the data. The construction of the k -NN graph can be explained by the following algorithm steps:

Let $G = (P, E)$ be an undirected graph, where P is a set of n points representing the nodes in the graph, and E is a set of edges representing the connections between nodes. For each point, $\mathbf{p}_i \in P$, the NearestNeighbors algorithm is used to finding its k nearest neighbors N_i in the grid. The set of neighbors N_i is defined as:

$$N_i = \{\mathbf{p}_j \in P \setminus \{\mathbf{p}_i\} : \mathbf{p}_j \text{ is among the } k \text{ nearest neighbors of } \mathbf{p}_i\}$$

Using N_i , we construct the set of edges E_i for each point p_i as follows:

$$E_i = \{(\mathbf{p}_i, \mathbf{p}_j) : \mathbf{p}_j \in N_i\}$$

The resulting set of edges E is the union of all sets E_i for $i = 1, 2, \dots, |N_i|$. The resulting graph $G = (P, E)$ is the k-NN graph of the grid, where each node represents a point in the grid and each edge represents a connection between a point and its k nearest neighbors.

The resulting k-NN graph is visualized in Figure 3, where each point is connected to its $k = 8$ nearest neighbors.

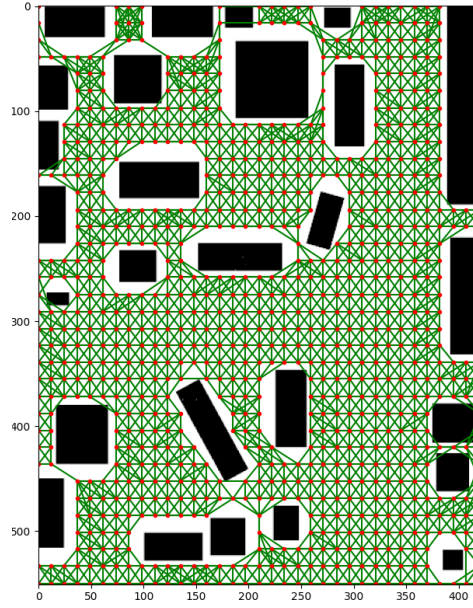


Figure 3: k-NN graph constructed from the grid

2.2 Calculation of geodesic distances based on the graph of the layout

The approach used in this project involves precomputing the distances to improve the performance and scalability of the pathfinding. This technique is similar to previous work on precomputing flow-fields, which has shown to be effective in improving efficiency for real-time applications [7].

After constructing the k-NN graph, the distances between vertices are calculated using Dijkstra's algorithm. These distances are represented by a distance matrix $\mathbf{D}$, defined as $\mathbf{D} = [d_{l,k}]_{NxN}$, where $d_{l,k}$ is the distance between vertices l and k in the graph, and k is the goal node that the agents have to reach. The result of this procedure is the geodesic distance of all points to every other sample point on the grid. This approach allows efficient and scalable multi-agent pathfinding by avoiding the need to compute the flow-field in real-time. The distances calculated using the k-NN graph enable effective navigation and path planning for multiple agents in a given environment.

2.3 Flow-field calculation and visualization

After calculating the distances between points in the P array, the flow-field is calculated with respect to a single target node denoted by k , which is the goal:

1. A weighted graph is generated based on the edge lengths. A new graph object is created and vertices are added to correspond to the points in P . Edges are then stored in a matrix, with edge weights set to the calculated distances.
2. Using the edge weights, the shortest paths between each pair of vertices in the graph are calculated and stored in a matrix.
3. The flow-field is calculated based on the graph. First, the goal position is defined and the index of the closest point to the goal is found. The distances from the goal node are then computed, and a heatmap is created based on these distances. The gradient of the distances along the edges is then calculated using the following equation:

Let $\mathbf{D} = [d_{l,k}]_{NxN}$ be the distance matrix, where $d_{l,k}$ is the distance between vertices l and k in the graph and k is the goal node which the agents have to reach.

The gradient of the distances along each edge in E can be computed as follows:

$$\begin{aligned}\Delta d_e &= \Delta d_{l,k} = d_{l,k} - d_{k,k} \quad e = (l, k) \in E \\ \Delta \mathbf{p}_e &= \Delta \mathbf{p}_{l,k} = \mathbf{p}_k - \mathbf{p}_l, \quad e = (l, k) \in E \\ \mathbf{g}_e &= \mathbf{g}_{l,k} = \frac{\Delta d_e}{\Delta \mathbf{p}_e} \quad e = (l, k) \in E\end{aligned}$$

Here, Δd_e is the difference in distances between vertices l and k for each edge $e = (l, k)$ in E , and $\Delta \mathbf{p}_e$ is the difference in coordinates between the endpoints of e . $\mathbf{g}_e$ is the gradient of the distances along e . The gradient $\mathbf{g}_e$ is a vector that points in the direction of increasing distances along e , with its magnitude proportional to the rate of change of distance.

2.4 Simulation-based path finding

In the simulation, the movement of particles in the flow-field can be studied, where the particles are influenced by the flow's components on a grid P . The particles represent multiple objects or agents that we want to navigate simultaneously through the environment, which could include robots, vehicles, or other entities that need to follow the calculated paths. We have a total of M particles, each denoted by $\mathbf{x}_m$ for $m = 1, 2, \dots, M$. The position of each particle at the t -th iteration is represented by $\mathbf{x}_m(t)$.

To simulate the movement of particles, their positions are updated at each iteration using the following equation:

$$\mathbf{x}_m(t+1) = \mathbf{x}_m(t) - \mathbf{g}_{i,j} \Delta t \mid \arg \max_i d(\mathbf{x}_m(t), \mathbf{p}_i)$$

Here, $\mathbf{g}_{i,j}$ represents the influence of the flow-field components (U and V) on the particles at grid point P_i , Δt is the time step between iterations, and $d(\mathbf{x}_m(t), \mathbf{p}_i)$ is the distance between the particle's position at iteration t and grid point $\mathbf{p}_i$. The term $\arg \max_i d(\mathbf{x}_m(t), \mathbf{p}_i)$ selects the grid point with the maximum distance from the particle's current position.

By applying this equation iteratively, we can simulate the movement of particles in the flow-field over time. This allows us to analyze their behavior and understand the impact of the flow-field on their trajectories.

The simulation continues until the specified time period is reached, and the final positions of the particles are plotted. An example of the result of the simulation and the heatmap generated from the distances can be seen in Figure 4

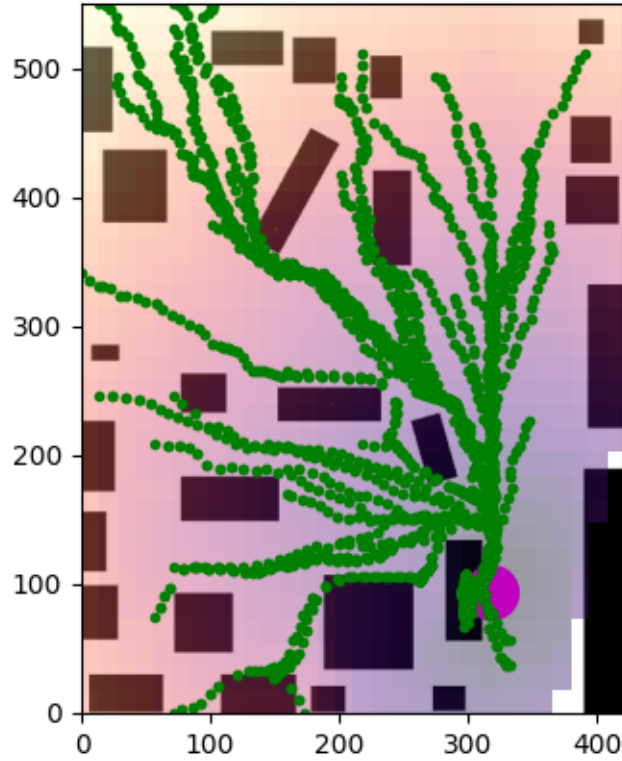


Figure 4: Particle trajectories and heatmap of many agents

2.5 Proper orthogonal decomposition analysis

To gain insight into the dominant modes of flow, POD analysis can be used.

The POD analysis involves computing the eigenvectors of the flow-fields and projecting them onto a reduced set of basis functions. The resulting basis functions, or modes, represent the dominant patterns in the flow-fields [13].

To perform the POD analysis, the flow-fields generated by the algorithm are randomly sampled, and the corresponding gradients along each edge are calculated. The gradients are then interpolated onto a regular grid using grid interpolation techniques. The resulting dense representation of the flow-fields is stacked into a single matrix.

Singular Value Decomposition (SVD) is used to compute the eigenvectors and eigenvalues of the stacked flow-fields matrix. The SVD decomposition of the matrix results in three matrices, ϕ , s , and C , Where ϕ is an orthogonal matrix representing the left singular vectors of the original matrix, s is a diagonal matrix containing the singular values in decreasing order, and C is another orthogonal matrix representing the right singular vectors of the original matrix, which are used to extract the dominant modes of variation in the data. The eigenvectors, or POD modes, capture the spatial structure of the flow-fields,

while the eigenvalues represent the amount of variance explained by each mode.

By analyzing the eigenvalues, we can determine the number of modes required to accurately represent the flow-fields. Additionally, the POD modes can be used for anomaly detection, pattern recognition, and other data analysis tasks.

The application of POD to the flow-fields generated by our algorithm provides a more compact and meaningful representation of the data, allowing for a deeper understanding of the flow patterns and their underlying dynamics. It also enables us to perform more efficient computations and optimizations, such as path planning and trajectory optimization, that require lower-dimensional representations of the flow-fields.

We can use the eigenvectors to project the flow-fields onto a reduced set of basis functions. The resulting modes represent the dominant patterns in the flow-fields, and can be used to identify different features of the layout.

Finally, we visualize the results of the POD analysis and discuss their implications for understanding the flow-fields and designing more efficient layouts, or paths for agents in obstacle-rich environments.

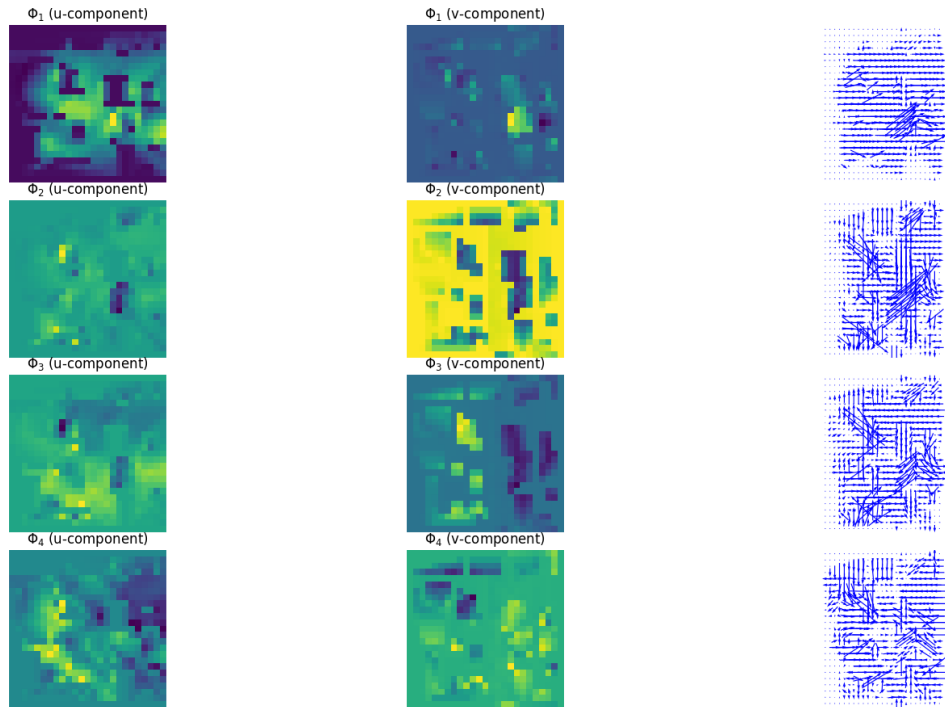


Figure 5: Analysis of the modes

2.6 Visualizing bottlenecks in the layout

After analyzing the modes using POD through Singular Value Decomposition (SVD), the bottlenecks can be identified in the layout by computing the cumulative energy map. In this context, N represents the total number of modes considered for the calculation. Utilizing the singular values obtained from the SVD analysis, represented by $\mathbf{s}$, we proceed to calculate the normalized energy, q_i , for each mode i as follows:

$$q_i = \frac{\mathbf{s}_i^2}{\sum_{j=1}^N \mathbf{s}_j^2}$$

Before calculating the cumulative energy map $\mathbf{R}$, we compute the magnitudes $\mathbf{F}_i$ for each mode. These magnitudes represent the contributions of each mode to the overall structure. Let $\mathbf{U}_i$ and $\mathbf{V}_i$ be the matrices obtained from ϕ for each mode i . Then, the magnitude $\mathbf{F}_i$ for each mode i can be calculated as:

$$\mathbf{F}_i = \sqrt{\mathbf{U}_i^2 + \mathbf{V}_i^2} \cdot q_i$$

The cumulative energy map $\mathbf{R}$ can then be calculated by summing the magnitudes $\mathbf{F}_i$ of the modes as:

$$\mathbf{R} = \sum_{i=1}^N \mathbf{F}_i$$

The cumulative energy map is then normalized to a range between 0 and 1 as follows:

$$\mathbf{O} = \frac{\mathbf{R} - \mathbf{R}_{\min}}{\mathbf{R}_{\max} - \mathbf{R}_{\min}}$$

Where $\mathbf{O}$ can be interpreted as a measure of the importance of different modes in the overall structure of the layout.

3 Algorithm implementation and real-world applications

In this section, the implementation of the method in Python is discussed, and a real-world application in the Operator 4.0 laboratory of the University of Pannonia is reviewed.

3.1 Implementation

In this section the specific implementation of the algorithm will be discussed, mentioning the packages and libraries used in the steps discussed in Section 2.

3.1.1 Python libraries and packages utilized in the algorithm

The approach is built using several technologies that are well-suited for generating graph-based flow-fields for path-planning applications. Scikit-learn is used to construct the k-NN graph, which is then processed using the Igraph library. Numpy is used for efficient data processing, while OpenCV is used for image processing tasks such as conversion and masking. Finally, Matplotlib is used for generating visualizations of the flow-field. These technologies were selected for their compatibility and effectiveness in achieving the goals of the proposed approach.

- OpenCV (Open Source Computer Vision) is an open-source library of computer vision and machine learning algorithms. It includes a variety of image processing and computer vision algorithms, such as image filtering, feature detection, object recognition, and more. OpenCV is widely used due to its versatility and ease of use. In the proposed approach, OpenCV is used for generating a binary mask of the image that is used to construct the graph-based flow-field for path planning.
- NumPy is a powerful Python library that provides support for large, multi-dimensional arrays and matrices, as well as a wide range of mathematical operations that can be performed on them. NumPy is also used extensively in the implementation of the graph-based flow-field generation in the proposed approach, as it provides useful tools for working with large arrays of data, computing distances between data points, and performing other mathematical operations necessary for constructing the graph and generating the flow-field very efficiently.
- Scikit-learn is a popular open-source machine-learning library for Python. It provides a wide range of tools for data analysis and modeling, including clustering, regression, classification, and more. In the proposed approach, I use the Nearest-

Neighbors module from Scikit-learn to construct the k-NN graph, which is the basis for the graph-based flow-field.

- Igraph is a Python library that provides tools for creating and manipulating graphs. It offers a variety of algorithms for graph analysis. In the proposed approach, I use Igraph to construct the graph and run various algorithms on it.
- Matplotlib is a data visualization library in Python that is used for creating static plots, charts, and graphs. In the proposed approach, I use Matplotlib to generate visualizations of every step in the process, which helps to provide a better understanding of path planning.

3.1.2 Digitalization of the layout

An appropriate image of the layout is needed for this approach to work. I processed the digitalized layout using OpenCV, so that the image is in the correct format, and can be used in later steps, regardless of the initial format. The image is turned into greyscale and turned binary by using a threshold on the values and turning them 1 or 0. The mesh grid is generated based on the preprocessed image, using NumPy's mesh grid procedure, so that a graph can be constructed later in the pipeline. This mesh grid is constructed using sampled points, which can be implemented in a number of different ways. My approach uses NumPy's linspace method because it gives a fairly reliable, and consistent grid as seen in Figure 6a. Another approach could be using Latin Hypercube Sampling, because it's efficient, and can handle more complex environments, although it's less reliable and consistent, as we can see in Figure 6b

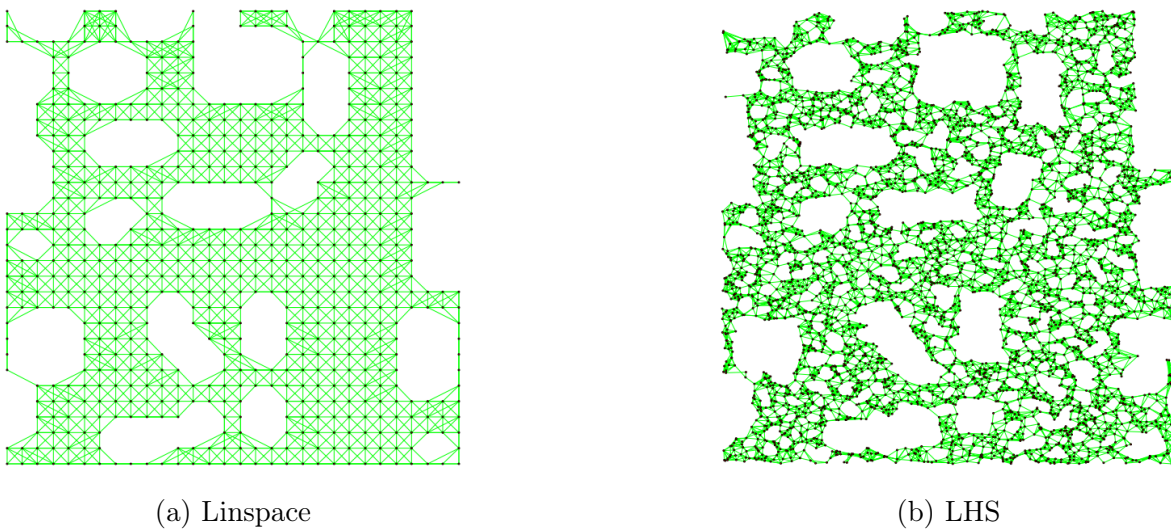


Figure 6: Different sampling methods

The layout of the manufacturing environment has to be analyzed to identify obstacles

that need to be avoided. This can be accomplished by masking the layout image, which allows for the identification of regions where objects are located. By setting the pixel values to 0 in these regions, the corresponding nodes in the mesh grid can be removed, effectively creating a pathfinding map that accounts for the obstacles.

In order to further optimize the pathfinding behavior and avoid the common issue of wall-hugging that is found in Flow-field pathfinding, [9] nodes within a certain radius of the obstacles has to be removed. This ensures that the pathfinding algorithm does not get stuck following the contours of the obstacles, but instead finds a smoother and more efficient path. This procedure is done by first creating a set of points XP that correspond to the nodes in the mesh grid, excluding those that were previously removed due to their proximity to the obstacles. The NearestNeighbors function is then used to identify the nearest neighbors of each point in XP within a specified radius. The resulting distances and indices arrays contain the distances and indices of the neighboring points for each point in XP.

To remove nodes close to the not-allowed region, the indices of the neighboring points are flattened and their unique values are obtained. These indices correspond to the nodes that are within the specified radius of the not-allowed region. These nodes are then removed from the original set of points P using a NumPy function, resulting in a new set of points with fewer nodes.

3.1.3 k-NN graph-based description of the layout

To construct the graph-based flow-field, the k-NN algorithm is used to build the graph from the remaining data points. This is achieved using the NearestNeighbors module from scikit-learn, which calculates the distances between each pair of points and identifies the k-nearest neighbors for each one. The value of k is set to 9 in this case.

The resulting graph is represented as a set of edges connecting pairs of nodes. To construct the edge list, the indices of the nearest neighbors are organized into pairs and filtered to remove duplicates and self-loops.

To filter out edges that are considered to be too long or unimportant for the flow-field, a distance threshold "DIST_THR" is specified. Edges with weights greater than this threshold are removed from the graph.

The final edge list is represented as a 2D array, where each row contains the indices of the nodes connected by an edge, along with the corresponding weight. This edge list is then used to generate the graph-based flow-field. The final graph can be seen in Figure 6a.

3.1.4 Calculation of geodesic distances based on the graph of the layout

The Igraph library is used to generate a weighted graph based on the edge lengths and calculate the shortest paths between all pairs of vertices.

The Geodesic distance formula is used to calculate the distances of all pairs of nodes. These distances are then used in order to calculate the Flow-Field needed for pathfinding, and later in the POD analysis of the modes. Each dimension of the matrix corresponds to a single goal node and the distances of all the other nodes toward it.

3.1.5 Flow-field calculation

After obtaining all of the distances, we can calculate the flow-field by determining the differences in distances from the goal node for each pair of connected nodes. The differences are divided by the corresponding edge lengths to obtain the flow-field's gradient components U and V. We can see the generated flow-field in Figure 7

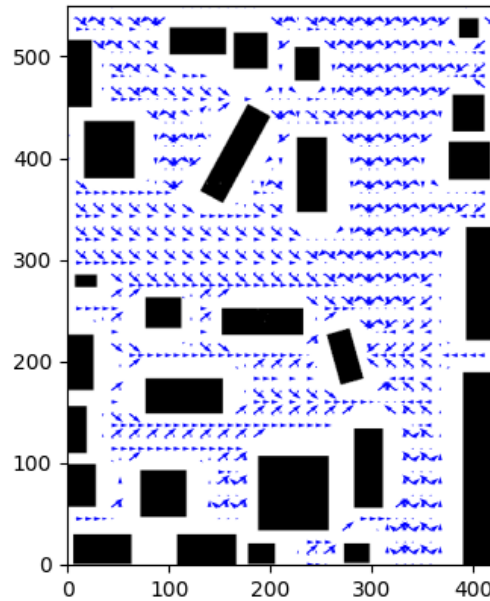


Figure 7: Flow-field visualized

3.1.6 Simulation-based path finding

To better understand the performance of this flow-field navigation approach, I constructed a simulation using the calculated flow-field. I implemented a simulation that takes the

flow-field components (U , V), an array of initial positions (x_0 , y_0), and simulation parameters (t_{sim} , dt) as input. The simulation function uses nearest neighbors interpolation to estimate the flow-field at each agent's position, updating their positions accordingly.

During the simulation, the movement of the agents is visualized by plotting their positions at each time step. This allows the agents' navigation to be observed in the environment, following the flow-field toward the goal. The simulation results provide an additional level of insight into the effectiveness of the proposed path-planning approach. In the following illustrations, we can observe the path the entities used in order to reach the goal:

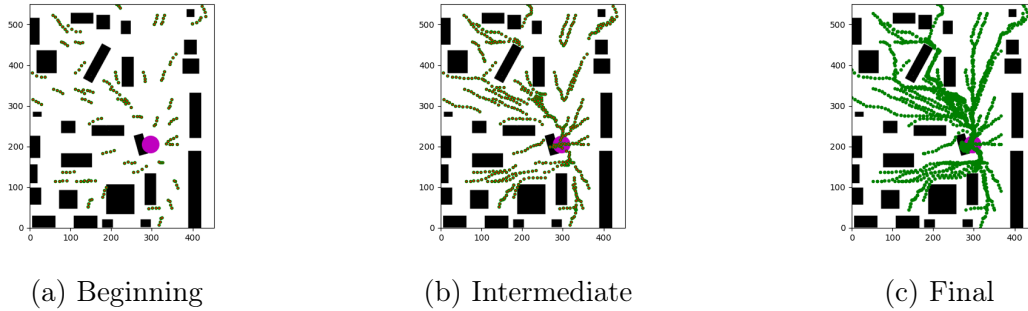


Figure 8: Simulation results at different stages: (a) beginning, (b) intermediate, and (c) final.

The simulation results in Figure 8 show that the proposed path planning approach successfully guides the agents towards the goal while avoiding typical wall-hugging behavior. Each agent was able to find a quasi-optimal path towards the goal point, as demonstrated in the following illustrations. These results show the effectiveness of the implemented flow-field calculation and visualization techniques described in previous sections.

3.1.7 Proper orthogonal decomposition analysis of the modes

In this section, I detail the steps taken to implement the POD analysis for the given problem.

First, a set of random permutations is generated to shuffle the data points. Then, for each data point, the gradients of the distance field are computed using the given edge information. The gradients are then interpolated onto a regular grid, resulting in separate arrays for the U and V components of the flow-field. These arrays are filled with interpolated values, and any NaN values are replaced with zeros.

Next, the data is prepared for the POD analysis by subtracting the mean from each row. The Singular Value Decomposition (SVD) is then performed on the resulting data matrix, yielding the orthogonal basis vectors (modes) represented by Φ , the singular values s , and the right singular vectors C .

Finally, the modes are visualized by plotting the U and V components separately, as shown on Figure 5 as well as displaying a quiver plot that combines both components. This visualization allows for a better understanding of the spatial structures and flow patterns represented by each mode. The process is repeated for a specified number of modes, showcasing their contribution to the flow-field.

3.1.8 Application of bottleneck analysis

The visualization of bottlenecks in the layout is particularly useful for understanding the regions of high energy concentration, which could indicate areas with significant flow variations or greater influence on the overall flow-field. Identifying these regions allows for further investigation and optimization to improve the efficiency and performance of the system.

The following Figure 9 displays the result of the bottleneck analysis applied to a sample layout. The grayscale layout image is overlaid with the cumulative energy map, showing the distribution of energy across the layout.

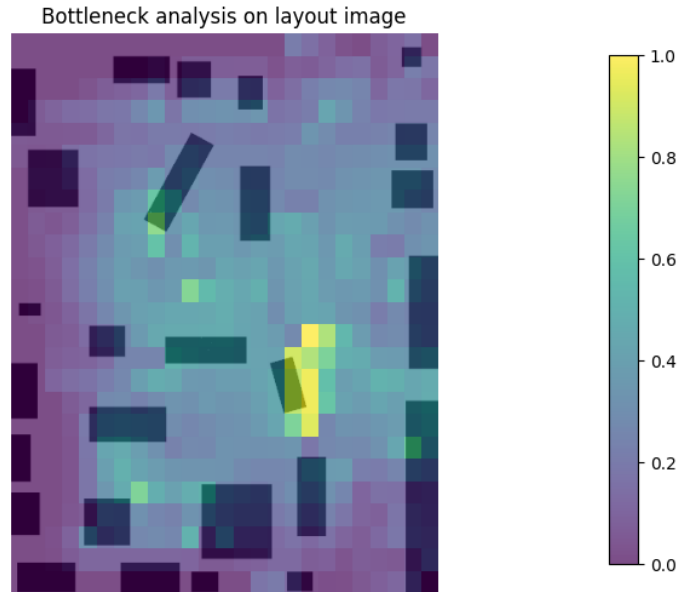


Figure 9: Bottleneck analysis on a sample layout image

In the visualization, the presence of a single high-energy blob indicates that there is a specific region with a high concentration of energy. This could mean that this area experiences more significant flow variations or has a greater influence on the overall flow-field. Consequently, this region may require further investigation or optimization in order to improve the efficiency or performance of the system. Additionally, focusing on this

particular region could help identify the primary sources of flow instabilities or energy losses, which can provide valuable insights for the design and operation of the system.

3.2 Real-world application of the pathfinding and analysis algorithm

In this section, the real-world application of the algorithm is discussed in two different scenarios

3.2.1 Operator 4.0 lab

The pathfinding algorithm developed in this work has potential applications in a wide range of industrial settings, including logistics, manufacturing, and healthcare. One notable application is in the Operator 4.0 laboratory at the University of Pannonia, which focuses on the integration of advanced sensors and sensing technologies in manufacturing machines and processes [14].

The Operator 4.0 laboratory serves as an educational and research platform that prepares students for the development and application of Industry 5.0 technologies. At the core of the laboratory is the concept of "intelligent space," which addresses the challenges of monitoring the stochastic nature of operators in production processes.

A game has been developed to showcase the cooperation between the operator and robot which involves players who must find hidden tags in the lab by using the visual cues handed by the system. The pathfinding algorithm calculates the shortest and most efficient route for the player to take to reach the tag, taking into account any obstacles or walls in the lab.

The game demonstrates the algorithm's capability to guide operators and robots toward specific parts or equipment quickly and accurately, even in complex environments. By using the algorithm, businesses can enhance the efficiency of their industrial systems, reduce the time needed to complete a task, and ultimately increase their productivity.

The algorithm is a powerful tool for enhancing transport systems and reducing transportation time. It can help businesses to streamline their operations and improve the flow of goods and equipment within their facilities. Overall, the pathfinding algorithm is a valuable tool for improving the efficiency of industrial systems. The hot and cold game demonstrates the algorithm's potential applications in an industrial setting and highlights its ability to guide operators and robots toward specific parts or equipment quickly and accurately.

3.2.2 Mini automated guided vehicles

The algorithm presented in this work can be used in mini-AGVs for navigation and pathfinding. The AGVs would need to be equipped with sensors that can detect their position within a grid of points. Once the AGV's position is known, the algorithm can be used to determine the direction and speed at which the AGV should move to reach its destination.

To use the algorithm in mini-AGVs, the AGVs would need to be equipped with a positioning system that can detect their position within the grid of points. Once the AGV's position is known, the flow-field can be used to determine the direction in which the AGV should move to reach its destination.

The algorithm has real-world applications in various industries where the efficient transport of goods or equipment is essential. In an automated warehouse system, mini-AGVs equipped with a positioning system and the algorithm can transport goods quickly and accurately without the need for human intervention. In manufacturing or healthcare, mini-AGVs equipped with the algorithm can navigate through complex environments and transport goods or equipment with greater speed and efficiency, ultimately improving the efficiency of the system.

3.3 Runtime test of the algorithm on different grid sizes

Grid dimension (n x n)	25	50	100	150
Node count	625	2500	10000	22500
Distance calculation time (ms)	950	1425	8847	51956
Flow-field calculation Time (ms)	37	101	378	2512

Figure 10: Algorithm runtimes for varying grid sizes

The algorithm developed has been subjected to several runtime tests to evaluate its performance. The tests were conducted on grids of varying dimensions, ranging from 25x25 to 150x150 nodes.

The algorithm was tested on a notebook with an Intel Core i5-8300H processor, 16GB of RAM, and a GTX-1050 graphics card. Despite the hardware specifications not being at the highest end of the spectrum, the algorithm was able to handle grids of varying sizes and complexities.

As expected, the runtime of the algorithm increases as the number of nodes in the grid increases. For example, the distance calculation time for a grid of 25x25 nodes was 950ms, while the distance calculation time for a grid of 150x150 nodes was 51,956ms. Similarly, the goal calculation time for a grid of 25x25 nodes was 37ms, while the goal calculation time for a grid of 150x150 nodes was 2,512ms.

It can be observed that the runtime calculation, „flow-field calculation" is considerably faster than the pre-computing part in „Distance calculation", making this algorithm really efficient in systems where the layout doesn't change in every single iteration.

4 Conclusion

In this work, I presented a novel algorithm and methodology for efficient flow-field calculation and pathfinding in grayscale images. The main contributions of this work include:

- I developed a k-NN graph-based algorithm to create a graph on a grid of points, considering the not-allowed regions in the image. This results in a more accurate representation of the image for pathfinding purposes.
- I calculated the shortest paths between all pairs of vertices in the graph using the igraph library, which speeds up the flow-field calculation process.
- I constructed flow-fields based on the calculated distances and gradients, enabling the simulation of particle movement in the flow-field.
- I utilized Singular Value Decomposition (SVD) to perform bottleneck analysis on the flow-field, identifying critical regions in the image that influence particle movement.

The proposed methodology successfully demonstrates the ability to calculate flow-fields and perform pathfinding on and analyze the digitalized layouts. The approach can be applied in various applications such as robotics, computer vision, and network analysis. The algorithm can also be applied to environmental monitoring, where it can help optimize the movement of drones monitoring air quality, wildlife habitats, or detecting natural disasters. Furthermore, in the realm of virtual reality and video games, the method can be employed to create more efficient navigation systems for characters and agents in complex virtual environments.

Future plans for this method includes:

- Integrating machine learning techniques to improve the bottleneck analysis, potentially enabling the system to predict critical regions and make proactive decisions to optimize particle movement.
- Exploring alternative graph construction and shortest path algorithms to further optimize the flow-field calculation process and reduce computational complexity.
- Expanding the algorithm to handle dynamic environments, where obstacles or the goal position may change over time, enabling real-time adaptation and pathfinding.

References

- [1] Georg Hoellthaler, Martin Schreiber, Kilian Vernickel, Johannes Be Isa, Jan Fischer, Nils Weinert, Roland Rosen, and Stefan Braunreuther. Reconfiguration of production systems using optimization and material flow simulation. *Procedia CIRP*, 81:133–138, 2019.
- [2] Abderraouf Maoudj and Anders Lyhne Christensen. Decentralized multi-agent path finding in warehouse environments for fleets of mobile robots with limited communication range. pages 104–116, 2022.
- [3] Richard E Korf. Optimal path-finding algorithms. *Search in artificial intelligence*, pages 223–267, 1988.
- [4] Amir R Soltani, Hissam Tawfik, John Yannis Goulermas, and Terrence Fernando. Path planning in construction sites: performance evaluation of the dijkstra, a, and ga search algorithms. *Advanced engineering informatics*, 16(4):291–303, 2002.
- [5] Sarah Uttendorf and Ludger Overmeyer. Fuzzy-enhanced path-finding algorithm for agv roadmaps. In *2015 Conference of the International Fuzzy Systems Association and the European Society for Fuzzy Logic and Technology (IFSA-EUSFLAT-15)*, pages 675–681. Atlantis Press, 2015.
- [6] P Grant. Local path planning: a brute force approach. pages 689–693, 1990.
- [7] Alexander Bruce and Jonathan Helsing. A scalability and performanceevaluation of precomputed flow field maps for multi-agent pathfinding. 2022.
- [8] Elijah Emerson. Crowd pathfinding and steering using flow field tiles. *Game AI Pro: Collected Wisdom of Game AI Professionals*, pages 307–316, 2013.
- [9] Mark Sabbagh, M Hassan Tanveer, Antony Thomas, Jacob Faile, and Muhammad Salman. Real time voronoi-like path planning using flow field and a. pages 103–107, 2020.
- [10] Hesham M Osman, Maged E Georgy, and Moheeb E Ibrahim. A hybrid cad-based construction site layout planning system using genetic algorithms. *Automation in construction*, 12(6):749–764, 2003.
- [11] Amanda D Roberts Thompson and Jonathan Finch. Reconstructing the layout of a coastal georgia plantation: Applications of lidar. *Journal of Archaeological Science: Reports*, 47:103798, 2023.

- [12] Jing Wang, Jingdong Wang, Gang Zeng, Zhuowen Tu, Rui Gan, and Shipeng Li. Scalable k-nn graph construction for visual descriptors. pages 1106–1113, 2012.
- [13] J Pedersen and K Meyer. Pod analysis of flow structures in a scale model of a ventilated room. *Experiments in fluids*, 33(6):940–949, 2002.
- [14] Tamás Ruppert, András Darányi, Tibor Medvegy, Dániel Csereklei, and János Abonyi. Demonstration laboratory of industry 4.0 retrofitting and operator 4.0 solutions: Education towards industry 5.0. *Sensors*, 23(1):283, 2022.