d³ FACT insight: A motion planning algorithm for material flow simulations in virtual environments

Kiran R Mahajan, Christoph Laroque, Wilhelm Dangelmaier ?
Michael Kortenjan, Christian Soltenborn, Daniel Kuntze †
University of Paderborn

Abstract

Visualization has always played an important role in the analysis of material flow simulations. These days, commercial software is available to visualize such systems. Using these software, the user has to model and parameterize the simulation and finally view the simulation in a virtual environment. After analyzing the system, typically he might wish to carry out changes in the layout, parameters, etc of the simulation model which also includes determining new motion paths for objects like forklifts, automated guided vehicles, etc. This paper presents a motion planning algorithm which automatically determines the paths for such objects depending on the new model layout without colliding with other objects of the virtual factory. First the motivation is presented in a case study form to emphasize drawbacks of existing software. Then the algorithm is described on the highest level followed by details of the methodology. The paper concludes with future research and conclusions.

Keywords: Discrete event material flow simulations, virtual reality, motion planning

1 Introduction

Simulation has been one of the most promising areas for the analysis of complex manufacturing systems. In the recent years visualization has been increasingly used to assist the less experienced user in analyzing the system. In recognition of this trend, we, at our institute have set up a project - d³ FACT insight which aims for advancement in research for simulation and visualization techniques of material flow systems. The main goals of this project include the development of an integrated software tool for all applications of the digital factory, especially the modeling and simulation of complex production processes. Specifically, this includes the ability to simulate manufacturing systems in multi-resolutions [Da04, DaMu04] with a modular approach, to interactively analyze the behavior of the underlying manufacturing system in 3D visualization [MuFi04] and the ability to provide multi-user support. Figure 1 shows the overall architecture of d³ FACT insight. One of the areas of research in the 3D visualization area is the ability to develop an algorithm for motion planning of elements like automated

? Heinz Nixdorf Institute, Business Computing, esp. CIM, Fürstenallee 11, 33102 Paderborn, Germany
† Heinz Nixdorf Institute, Algorithms and Complexity, Fürstenallee 11, 33102 Paderborn, Germany
guided vehicles, forklifts, etc to address specific problems faced in present day virtual environments. Section 2 addresses the need for developing such an algorithm with the help of short case studies.

2 Problem definition

TaylorED from \textit{F\&H simulations BV} is used to demonstrate the drawbacks of existing software in relation to planning motion paths for motion elements like forklifts, automated guided vehicles, etc. in virtual environments. Consider Figure 2 for a small system modeled used TaylorED. Parts enter the system at point A and are carried by the forklift to point B where they are loaded and processed in the following machines. In this case, because of the modeling connections between point A and B in Taylor ED, the forklift automatically finds the shortest path to travel between these two points.
In other software, for e.g., in AutoMOD from Brooks Automation Inc, the motion of an element between two points is to be planned by the modeler using “non-flexible” paths. If the model of the factory consists of many machines and many alternative paths for several moving elements, the approach used by Taylor ED would certainly not work (to automatically determine the motion path), whilst (both TaylorED and AutoMOD) also failing to automatically change the motion path of the moving object based on a new layout by the modeler to avoid collisions with other elements. Figure 3 shows the forklift passing through the wall as a result of changed layout of the manufacturing facility. The above discussion clearly shows that research is required to automatically find motion paths for moving elements based on the layout changes done by the modeler. Moreover, this should happen without colliding with other “obstacles” of the factory. Section 3 depicts the motion planning algorithm developed by our group.

3 Algorithm

Our goal is to compute paths and perform collision detection online. This should not need too much computing resources. For this purpose, our concept has a pre processing part in which we compute data that enables us to perform the online tasks in a very efficient way. We need small assumptions for our algorithms to work as desired:

1. Moving objects travelling in opposite directions can pass each other at most places.
2. It is not necessary to go to every point in the plane with the objects. Only specific points of interest are considered.

Moving an object along a certain path, obstacles (like machines, and other movable objects) must not be touched. The motion planning algorithm consists of three distinct steps as shown in Figure 4 (the pre processing consists of steps 1 and 2). First an object specific phase is performed to create a 2-dimensional outline from a 3-dimensional model. The second phase also creates a scene specific graph representing possible paths for the moving objects. Finally this graph is used for an algorithm to search online for a
path from the current position of an object to its destination in the third step. Each of the phases is described next in sections 3.1, 3.2 and 3.3.

Create a 2D outline from a 3D model of all objects and obstacles in the factory

Create a scene specific graph to represent possible motion paths for the movable object

Online search for a path and collision prevention

Figure 4: overall idea of the algorithm

### 3.1 Step 1: Creating 2D outline from a 3D model for all objects

Models of machines and moving objects are given as 3-dimensional objects. Since we show later the results of applying this step on real 3D objects, Figure 5 has been taken as an example. It basically consists of three machine cells and a forklift. Motion planning takes place on a plane as all vehicles move on the factory floor. So we need to obtain a 2-dimensional representation of our models.

Figure 5: an example of a manufacturing facility [MuFi04]
Given a 3D model we need the outline of the model’s orthogonal projection on the XZ-plane. However, simply projecting the 3D model does not consider different heights of objects. E.g. in Figure 6, the object is on a feasible position but it overlaps the obstacles outline. To circumvent this we discretise the model’s height. For the different height steps the outline is constructed and stored in a database. If we are given an object we use the outline of the height next larger than the object’s size. Since we always want to have a secure distance from the object to the obstacle we are not interested in the outline of obstacles clipped immediately above the objects height. Therefore, we use the smallest discretisation step which is high enough to maintain a safe distance. As shown in Figure 6, distance A (the distance between the height of the object and the next discretisation step) is bigger than the safe distance D. As a result, the outline for the obstacle then, actually, is the one shown with a bold line c. If coordinates of 3D models correspond to real world coordinates, e.g. one unit is one meter, reasonable values are easy to determine. Figure 7 shows the result of generating a 2D outline from 3D models for the example.
3.2 Step 2: Creation of a scene specific graph to represent possible paths

For each type of object\(^1\) we create a graph which has two purposes: enable the use of shortest path algorithms and store information on its edges and nodes, namely the position and direction of view of the object (used for displaying it by visualization) and the geometric object describing the current location of the object (used by collision prevention). Graph construction consists out of 3 parts:

1. Construction of subgraphs describing a directed path around each of the obstacles, identifying possible connection points.
2. Compute connections between the subgraphs of the obstacles using the connection points found in 1.
3. Map docking points to graph nodes.

A scene is specified by the obstacles (in Figure 7, the machine cell is an obstacle) it contains and their position. Figure 8 shows a scene with several obstacles.

![Figure 8: scene with obstacles](image)

The union of the obstacles’ outlines determines impassable regions for moving objects (refer Figure 8). For each kind of object we construct a graph representing potential paths in this scene. First we choose the outlines of the obstacles depending on the object’s height. The object’s dimensions characterise a minimal distance to the obstacles which is shown in Figure 9. For every object there is an orientation axis. If the object moves it has to be rotated such that the axis points along its moving direction. This axis can be given as metadata or implicitly. If there is no direction specified we assume that the object is

\(^{1}\)Objects can be AGVs, forklifts etc. The type of an object depends on its dimensions and turn radius.
modelled such that its moving direction is along the X-axis of its coordinate system. Each object has a reference point. We move this point along every obstacle’s outline, keeping a distance such that the object never collides with the obstacle if its orientation axis is aligned in the direction of the edge. This results in a new enlarged outline, shown in Figure 10.

Figure 9: obstacles and their extended outlines indicating the minimum distance

Figure 10: obstacles, extended outlines and nodes
The outlines define the graph. As seen in Figure 10 and 11, on the corner there would be many such nodes representing a circular path. The endpoints of the outlines result in nodes and for every line an edge is inserted between its endpoints. These edges represent a path around an obstacle. Every node of the graph stores its position, the orientation of the object and the geometry representing the actual object in the scene. This object will be used for collision prevention.
Our database additionally contains so called docking points which can be defined as positions where objects can interact with obstacles. E.g. a forklift must not drop his load anywhere near his target machine, but at a certain position. These docking points belong to obstacles. The simulation kernel will order an object to move to a specific docking point. On the outlines as seen in Figure 9, additional nodes are inserted, which represent these docking points. They are special target nodes for objects (refer Node 1 and Node 2 in Figure 12). The goal points of our algorithm are always these docking points.

In the last section of step 2 of the algorithm, we compute connections between adjacent obstacles and insert edges to the graph. These edges are used to cross free space between the obstacles. Figure 13 shows the resulting graph.

Since we are interested in finding short paths the edges are weighted with their length. If the direction of an edge is clockwise with respect to the obstacle, the edge is weighted with the Euclidean distance of the positions stored in the two nodes of the edge. If it is oriented counter clockwise, the edge is weighted higher: we multiply the distance with a constant factor. In this way we make sure that objects are overall going forward but we still allow moving backwards. Figure 14 shows the final graph, including the geometry objects associated with it which are used for collision prevention.
3.3 Step 3: Online search for a path from current position and collision prevention

At runtime we use this graph to find a path from an object’s current position to a given goal point. Since we move objects always to target points inside the graph the object’s position cannot be outside the graph. Using Dijkstra’s shortest path algorithm [DIJ59], we find the sought after path. After running the shortest path algorithm, the obtained paths are stored to be used for collision prevention. We have an optimistic approach: Since our graph implies a right handed traffic system, we expect collisions to rarely happen. We only predict the future for small time window in which collision is solved by waiting. At this point we use the stored paths to compute the positions of the moving objects in the plane. If simulation time has moved beyond the end of our time window, we compute the next time window. Combined with our visualization system another kind of collision can occur than two objects crossing paths. Since we want as much immersion of the user as possible, the environment should react to him. If the user blocks the way of an object, the object has to stop. We cannot control the user or estimate for how long he will stay there, so we just wait until the user leaves the path instead of trying to get around him. As paths of other objects can be affected we have to throw away all our future predictions and to recalculate them.
An example of this situation occurs in Figure 15 where forklift C has to wait until B has passed. A question that comes to mind when thinking about this discussion is what about priorities of the moving objects. Such priorities will be set by the simulation engine depending on what needs to be done in the simulation for the system under analysis. Another way to increase immersion is smoothing the path of the objects. Instead of exactly following the lines of the graph we use curves to turn to the next edge. The radius of these curves depends on the turn radius of the object. Notice that these curves can be obtained in the pre-processing step. If no curve exists for the given turn radius we know that this curve is to sharp for this object.

4 Conclusions and future research

In this paper, we have presented an algorithm for motion planning of moving objects in a virtual environment. To summarize, the automatic generation of a 2D outline of the 3D obstacles and moving objects has been implemented (section 3.1). The creation of a scene specific graph to represent possible paths, and associated work of connection between nodes and obstacles has also been implemented (section 3.2). The online search for a motion path from the current position and the automatic collision prevention is currently under implementation (section 3.3). As indicated in section 1 of this paper, our work has demonstrated that automatic motion planning for moving objects in a virtual environment can be done in an efficient way, including automatic collision avoidance of such objects.
To completely realize these goals, much further research and testing needs to be carried out in addition to the work already being carried out in our group [DaMu04], [MuFi04]. Issues addressed and results achieved form an important advancement in modern research for simulation and visualization of manufacturing systems.

Literature


