Institut für Visualisierung und Interaktive Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 269

# Visuelle Unterstützung einer Simulation zur Optimierung von Arbeitswegen in der Fertigung

Sebastian Grund

| | |
|---|---|
| **Studiengang:** | Informatik |
| **Prüfer/in:** | Prof. Dr. Thomas Ertl |
| **Betreuer/in:** | Dipl.-Inf. Dominik Herr,<br>Dr. Steffen Koch |
| **Beginn am:** | 17. November 2015 |
| **Beendet am:** | 13. Mai 2016 |
| **CR-Nummer:** | H.5.2, I.6.3, D.4.7 |

# Kurzfassung

Bei der Fabrikplanung gibt es den Bedarf der Reduzierung von späteren Kosten. Dies kann durch eine Optimierung des Fabriklayouts geschehen. Einer der Aspekte dieser Layout-Optimierung sind die Werkerwege. Werkerwege sind die Wege die ein Arbeiter während einer Fertigung zurücklegen muss, um beispielsweise andere Arbeitsflächen oder Materialien zu erreichen. Die Optimierung dieser Wege gehört in den Bereich der Betriebsoptimierung und bedeutet Kosteneinsparungen in der Produktion.

Man kann hierbei zwei Hauptoptimierungsziele ausmachen. So ist die Wegstrecke für den Arbeiter möglichst gering zu halten, damit weniger Arbeitszeit dafür aufgewendet werden muss. Außerdem ist darauf zu achten, dass sich nicht zu viele Werker gleichzeitig in einem Bereich befinden, damit sie sich nicht gegenseitig behindern.

Diese Arbeit umschreibt die Entwicklung eines Konzepts und die Implementierung dessen in einen Prototyp, mit welchem eine Visualisierung und interaktive Optimierung von Fabriklayouts, unter dem Gesichtspunkt der Werkerwege, stattfinden soll. Dazu werden die erwarteten Werkerpfade in ein 2D-Layout eingezeichnet. Der Benutzer kann mit Interaktion das Layout umstellen. Die veränderten Wege werden dem Benutzer zur Bewertung visuell präsentiert. Der Prototyp wird anhand einiger Use-Cases evaluiert.

# Inhaltsverzeichnis

# Abbildungsverzeichnis



# Tabellenverzeichnis

# Verzeichnis der Algorithmen

# 1 Einleitung

In der industriellen Fertigung gibt es, bei den meist sehr komplexen Verarbeitung, viele Freiheitsgrade. Es herrscht das Bedürfnis nach Optimierung der Produktion. Der Komplexität und hinzukommenden Veränderungen an Produkt und Produktion geschuldet, kommt man mit einer diskreten Optimierung oft nicht mehr nach und bedient sich immer öfter einer Simulation des Produktionsablaufes um dessen Effektivität zu bewerten. Ein Teilaspekt einer solchen Simulation sind die Werkerwege, welche auch Möglichkeiten zur Optimierung bieten. Es können Mitarbeiterwege verkürzt werden, damit die Arbeiterzeit des Arbeiters effektiver genutzt werden kann. Zudem kann darauf geachtet werden, dass nicht zu viele Mitarbeiter auf einmal in einem gewissen Bereich der Fabrik arbeiten, damit sie sich nicht gegenseitig behindern.

Werkerwege werden vor allem durch das Fabriklayout bedingt, dessen Entwurf und Änderung Teil jeder Fabrikplanung sind.

Diese Arbeit soll das Konzept einer visuellen Analyse für Arbeiterwege in Simulationsergebnissen hervorbringen. Diese Visualisierung es einem Benutzer ermöglicht die besagten Wege zu betrachten, sowie per Interaktion und damit Variationen des Fabriklayouts, zu verändern.

Der endgültige Nutzen wäre eine Optimierung des Produktionsvorganges für Mitarbeiter und Fabrikbesitzer.

## Gliederung

Die Arbeit ist in folgender Weise gegliedert:

**Kapitel 2 – Grundlagen** Hier werden Grundlagen dieser Arbeit beschrieben.

**Kapitel 3 – Aufgabenstellung und Lösungsansatz** Hier wird die Aufgabenstellung an diese Arbeit vorgestellt.

**Kapitel 4 – Verwandte Arbeiten** Hier werden Projekte vorgestellt, die dieser Arbeit ähneln.

**Kapitel 5 – Konzept** Hier wird der begangene Arbeitsweg aufgezeigt der zum Konzept führte.

**Kapitel 6 – Implementierung** Die Implementierung des Konzepts, so wie die Benutzeroberfläche werden hier erklärt.

**Kapitel 7 – Evaluation** Hier wird das erarbeitete Konzept anhand von Usecases evaluiert

**Kapitel 8 – Zusammenfassung und Ausblick** fasst die Ergebnisse der Arbeit zusammen und stellt Anknüpfungspunkte vor.

# 2 Grundlagen

In diesem Kapitel werden Grundlagen erklärt, die dem Leser nicht unbedingt vorliegen, aber zum Verständnis der Arbeit unter Umständen hilfreich ist.

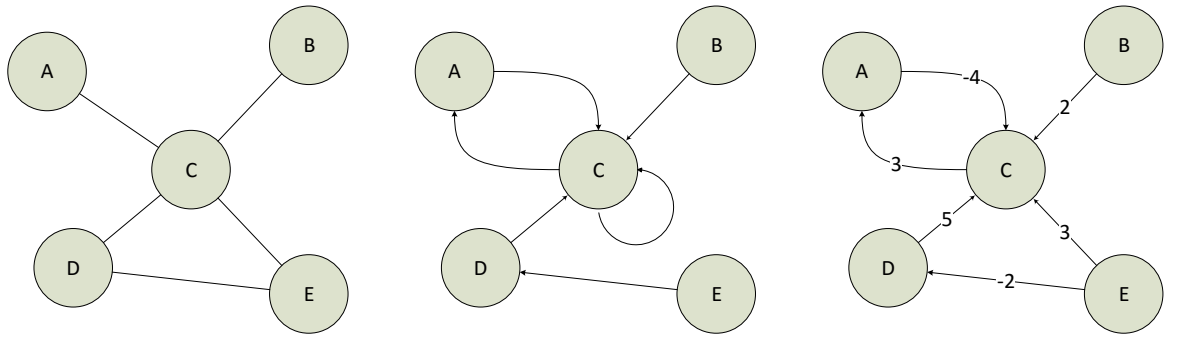
## 2.1 Graphen

Graphen eigenen sich besonders gut Zweidimensionale Wegenetze zu modellieren.

Sie sind die Grundlage der Graphentheorie, was eine Disziplin der Mathematik ist. [Die12]

Graphen bestehen aus Knotenpunkten und (un-)gerichteten, (un-)gewichteten Kanten die zwischen je zwei Knotenpunkten verlaufen.

Zur Erklärung siehe Abbildung 2.1.




**(a)** Ein ungerichteter und ungewichteter Graph, hierbei sind Kanten „nur “ Verbindungen zwischen zwei Knotenpunkten

**(b)** Ein gerichteter Graph. ( mit einer Kante deren Ziel und Startknoten identisch sind)

**(c)** Ein gerichteter und gewichteter Graph. Hierbei bekommt jede Kante noch ein Gewicht zugeteilt, welches Beispielsweise die Wegkosten dieser Kante angibt.

**Abbildung 2.1:** Abbildung jeweils eines ungerichteten, eines gerichteten und eines gerichteten sowie gewichteten Graphen

### 2.1.1 Dynamische Graphen

Bei einem fest definierten Graphen, wie im letzten Kapitel beschrieben, spricht man von einem statischen Graphen, da er sich, laut Definition, nicht verändern kann. Ein dynamischer Graph dagegen, lässt sich, sehr gut, durch eine Folge an statischen Graphen verstehen. Die einzelnen Teilschritte, können durch Algorithmen, Benutzerinteraktion oder andere Aktivitäten verändert werden.

Diese Änderungen kann man in drei Arten unterteilen: (Unterteilung entnommen aus [Lia14] S.13)

1. **Knotenänderungen** Das heißt es kommen Knoten zum Graphen hinzu, oder werden vom Graphen (mitsamt ihren Kanten) entfernt.
2. **Kantenänderungen** Hier werden Kanten aus dem Graph genommen, oder neue Kanten in den Graph eingefügt.
3. **Änderung der Kantengewichte** Hier werden nur von schon existierenden Kanten die Kantengewichte geändert.

War der dynamische Graph vorher der Graph $G$, so repräsentiert ihn nun der Graph $G_1$, der aus Änderung von $G$ resultiert.

Die Herangehensweise der Graphen-Folge, hilft auch darauf zu achten während Berechnungen und Visualisierungen immer einen statischen Graphen zu nutzen, beziehungsweise anzuzeigen. Damit beugt man ungewollten Anzeige Artefakten und falschen Ergebnissen vor. Wenn man diese strikte Trennung zwischen den Graph-Zuständen nicht vollzieht, dann kann es, zum Beispiel, zu falschen Wegberechnungen kommen (ein Beispiel für eine Wegberechnung auf einem Graph kann man in Kapitel 2.2 sehen).

Werden, während der Berechnung des Weges, Kantengewichte verändert, so kann es sein, dass der gefundene Weg kein Idealer mehr ist, da zur Berechnung noch die vorherigen Kantengewichte hergenommen wurden.

Beim Graph der zur Wegberechnung innerhalb des hier vorgestellten Ansatzes benutzt wird, handelt es sich um einen dynamischen, der durch Interaktion des Benutzers in allen drei Arten verändert wird. (siehe Kapitel 5.2.1)

Ein großes Problem von umfangreichen, dynamischen Graphen ist ihre Rechenintensivität, sprich der Aufwand für das Ausführen von Operationen auf/mit einem solchen Graphen, die vor allem bei Echtzeitanwendungen, negativ, zum Vorschein kommt. Beispiel Kantengewichtsänderung:

Es existiert durch einen Graphen ein eingezeichneter Idealen Weg zwischen zwei Knoten. Hat nun der Benutzer die Möglichkeit, beispielsweise mit der Maus, durch verschieben von Knotenpunkten die Kantengewichte zu ändern, so müsste ja in jedem Zustand, die der Bildschirm anzeigt, der neue Ideale Weg berechnet werden.

Um dem Benutzer eine Animation zu zeigen kann man zum Beispiel mal 24 Hz ansetzen, das ist die normale Bildrate eines Films.

Wenn wir nun von einem A* Algorithmus (Kapitel 2.2) für die Wegsuche hernehmen, so müssen wir 24 mal in der Sekunde eine Rechnung der Komplexität $O(|K|^2)$ ausführen, was bei entsprechender Kontenanzahl zu sehr viel Rechenaufwand führen kann. ($K$ ist hier die Menge der Knoten im Graphen) Allein schon das häufige Zeichnen, sprich anzeigen, von vielen Elementen kann schon, je nach Methode, zu Problemen führen.

## 2.2 Wegsuchalgorithmus A-Stern

Ganz Allgemein ist der Suchalgorithmus A-Stern, oder auch A*, ein deterministischer Wegsuchalgorithmus auf einem Graphen mit positiven Kantengewichten. Um ihn anzuwenden muss eine Abschätzung, bzw. Heuristik, der Wegkosten für jeden Knoten hin zum Zielknoten gegeben sein. Beschrieben wird er unter anderem von Hart et al. [HNR68] Eingabe sind, die oben beschriebene Kostenschätzung, der Graph auf dem die Suche stattfinden soll, sowie Start- und Endknoten des gesuchten Pfades.

Knoten des Graphen: $K = \{A, B...\}$
Kanten des Graphen: $L = \{(C, D), ...\}$ wobei $C, D \in K$
Startknoten: $S \in K$
Zielknoten: $E \in K$
Heuristik-Funktion: $h(A) \forall A \in K$ (geschätzte Kosten vom Knotenpunkt A zu E)
Real-Abstandskosten-Funktion: $c(A, B) \forall A, B \in K$ (reale Kosten des Weges zwischen A und B)
Realkosten-Funktion: $r(A) \forall A \in K (= c(A, E))$
Pfadkosten-Funktion: $g_p(A) \forall A \in K$ (Kosten im aktuellen Pfad von S nach A)
Prioritätsliste-Kosten: $f(A) = g_p(A) + h(A) \forall A \in K$

Um eine Lösung zu finden muss die Heuristik **zulässig** sein, das heißt sie darf niemals größer als die realen Kosten des Weges sein. (Siehe Gleichung 2.1)

(2.1) $h(A) \leq r(A) \forall A \in K$

### 2.2.1 Funktionsweise

Im Pseudo-Code 2.1 wird die Funktionsweise des A* beschrieben.

Das Herzstück des Algorithmus ist eine „Priority-Queue“ $P$, also eine sortierte Liste aus Knotenpunkten, welche nach dem Wert der Funktion $f$ sortiert werden. Wenn ein Knoten eingefügt wird, der schon in der Liste ist, so wird nur der mit dem kleineren $f$-Wert behalten.

**Algorithmus 2.1** Pseudo-Code zum A*-Algorithmus

**procedure** A-Stern(Graph $G$, Heuristik-Funktion $h$, Startknoten $S$, Endknoten $E$)
    Priority-Queue $P$
    Visited-Queue $V$
    $P.Enqueue(S, 0, h(S))$ // Knotenpunkt, Kosten bisher, erwartete Kosten
    **while** $P.HasItems()$ **do**
        Knoten $A$, Kosten $S$ bis $A$ $k$ = $P.DequeueMinPathCostVertex()$
        **if** $A$ == $E$ **then**
            **return** $A$ => Pfad gefunden
        **end if**
        $V.EnList(A)$
        **for all** Nachbarn $B$ von $A$ **do**
            **if** $V.Contains(B)$ **then**
                $Continue$
            **end if**
            $k_{new} = k +$ Kosten von A nach B
            **if** $P.HasSmalerValueFor(B, k_{new})$ **then**
                $Continue$
            **end if**
            $P.Enqueue(B, k_{new}, k_{new} + h(B))$
            $B.Predecessor = A$
        **end for**
    **end while**
    **return** NULL => Pfad nicht existent
**end procedure**

Des weiteren gibt es mit der „Visited-Queue“ $V$ eine Liste der bereits untersuchten Knotenpunkte. Verwendet wird sie um zu verhindern das Knotenpunkte öfters besucht werden. Diese Liste ist nicht unbedingt nötig, kann aber die Laufzeit des Algorithmus erheblich verringern. (Siehe Abbildung 2.2)

Wie im Pseudo-Code 2.1 beschrieben, wird zu Beginn der Startknoten in $P$ eingefügt.

Jetzt wird immer der Knoten mit dem kleinsten $f$-Wert aus $P$ entnommen. Wenn dieser der Zielknoten ist, so hat man den Weg gefunden. Ansonsten wird der Knoten in $V$ und Seine Nachbarn in $P$ eingefügt. (Sofern sie nicht schon in $V$ sind). Jeder Knoten in $P$ speichert seinen Vorgänger, sprich den Knoten, durch den sie in $P$ gesteckt wurden. Wenn $P$ nun leer ist, das heißt, dass alle von Startknoten aus erreichbaren Knoten besucht wurden, so existiert kein Weg.

### 2.2.2 Eigenschaften

Der Algorithmus A* ist vollständig. Das heißt, dass immer eine existente Lösung gefunden wird.

Gilt über die Zulässigkeit noch das die Heuristik **monoton** ist, so gilt auch das A-Stern einen optimalen Weg findet.

Monoton ist eine Heuristik, wenn die geschätzten Kosten von einem Knoten A zum Endknoten, immer kleiner oder gleich sind wie die geschätzten Kosten eines Nachbarn B von A, addiert mit den tatsächlichen Kosten für den Übergang von A nach B. (Siehe Gleichung 2.2),

(2.2) $h(A) \leq c(A, B) + h(B) \forall A, B \in K :$ B ist Nachbar von A

Das kann auch, im Austausch gegen eine langsamere Performance durch leichte Umstellung des Algorithmus (u.a. Weglassen der Visited-Liste) mit einer nur zulässigen, aber nicht monotonen, Heuristik erreichte werden.

Eine monotonen Heuristik ist A-Stern auch optimal effizient, das heißt, dass kein anderer Algorithmus mit den Informationen der Heuristik schneller eine optimale Lösung findet.

### 2.2.3 Beispiel

Als ein Beispiel ist in Abbildung 2.5, ein einfacher Durchlauf einer Wegsuche in einem Weggraph mithilfe des A-Stern Algorithmus aufgezeigt.

## 2.3 Heatmap

Eine Heatmap ist eine Methode um Daten innerhalb eines Datensatzes zu visualisieren. Hier werden bei einer (1D-, 2D-, oder 3D-) Visualisierung des Datensatzes, Bereiche, nach Wertigkeit einer betrachteten Eigenschaft, (Häufigkeit eines Elements, Temperatur), verschieden eingefärbt.

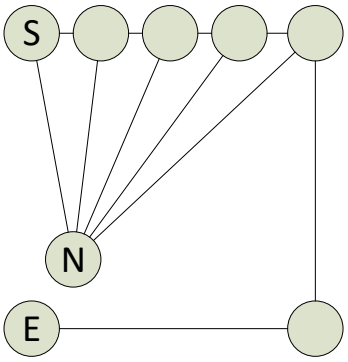
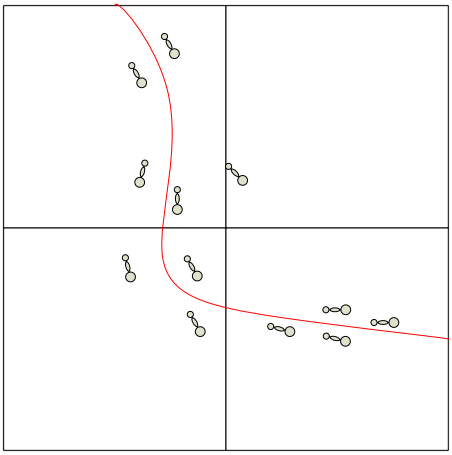
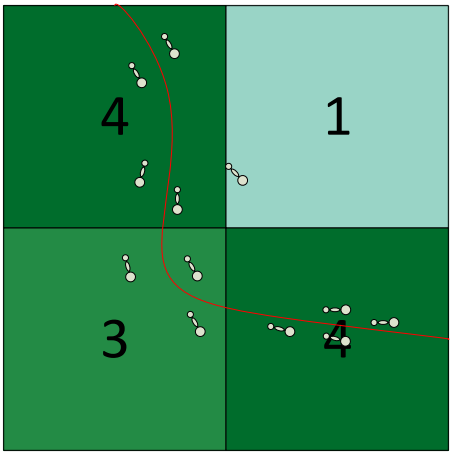



**Abbildung 2.2:** Ein Beispiel für einen Suchgraph bei dem der Einsatz einer Visited-Liste zu einer erheblichen Verringerung der Laufzeit führt.
Der gesuchte Pfad geht von S nach E, mit einer Heuristik, die den geometrischen Abstand der Knoten zum Endknoten als Schätzung nimmt.
Ohne Visited-Liste wird der Knoten N immer wieder untersucht, da er einen kleinen Wert der Funktion f(x) hat.



**(a)** Heatmap-Gitter über einer 2D Visualisierung von Ameisen-Positionen




**(b)** Gitter aus 2.3a nach dem Einzeichnen der Werte

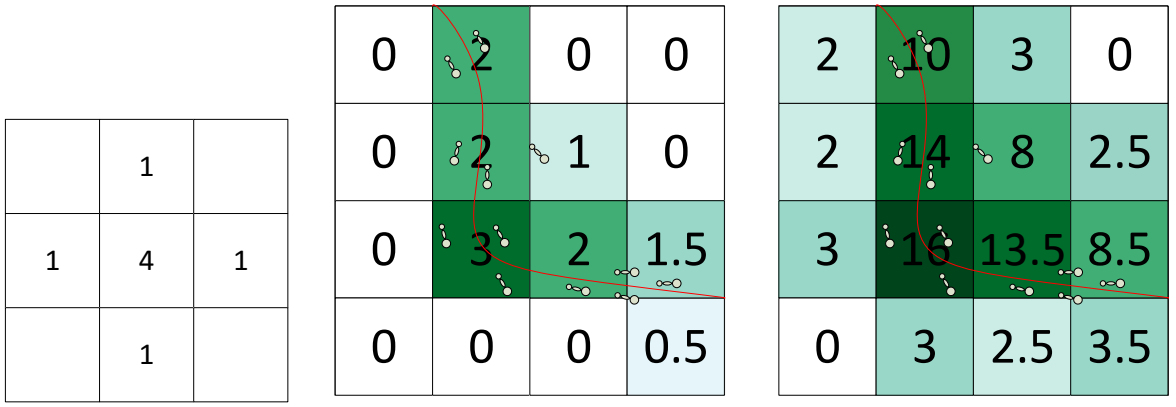
**Abbildung 2.3:** Einfache Heatmap mir grüner Färbung (Farben von [BH+])

### 2.3.1 Vorgehensweise

Die Vorgehensweise soll im Folgenden anhand einer 2D Visualisierung von Ameisen, entlang einer Ameisenspur, gezeigt werden.

Man legt über den betrachteten Visualisierungs-Raum ein Gitter. Für jede der Gitterzellen wird nun ein Wert festgestellt, z.B Wie viele Ameisen sich zum Zeitpunkt a in ihr befunden haben. Anhand dieses Wertes wird die Zelle dann eingefärbt (siehe Figur 2.3).

Durch die Verfeinerung des Gitters kann man nun auch genauere Verteilungen visualisieren. Eine intuitivere Visualisierung der Verteilung lässt sich in einem weiteren Schritt zum Beispiel durch Verwendung eines Gauß-Kernels erstellen.
Ein Kernel ist eine Matrix aus Faktoren, welche für die Berechnung der Gitterwerte herangekommenen werden.
Im Beispiel wird für jede Ameise der eigentliche Heatmap-Zellenwert um vier und die der Nachbarn links, rechts, über und unter der eigenen Zelle um jeweils eins erhöht. (Siehe Abbildung 2.4)

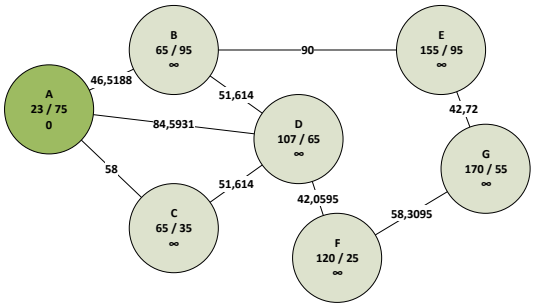



**(a)** Beispiel eines Gaußkernels

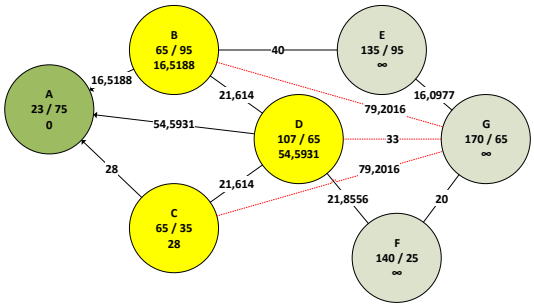
**(b)** Heatmap ohne Gaußkernel

**(c)** Heatmap mit Gaußkernel aus 2.4a

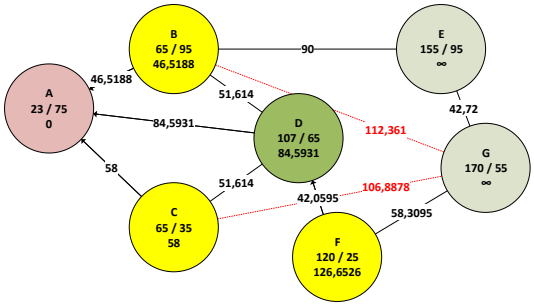
**Abbildung 2.4:** Heatmap unter Verwendung eines Gaußkernels (Farben von [BH+])



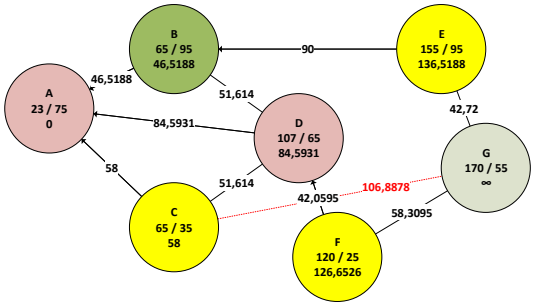

**(a)** Wir starten mit dem Knoten A



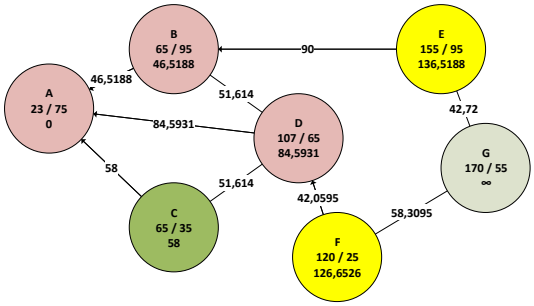

**(b)** Es werden alle Nachbarn von A mit ihren $f(x)$ Kosten in $P$ eingefügt. Falls geschätzten Kosten nicht als reale Kante vorliegen, werden sie als gedachte, rot-gestrichelte Linie eingezeichnet.



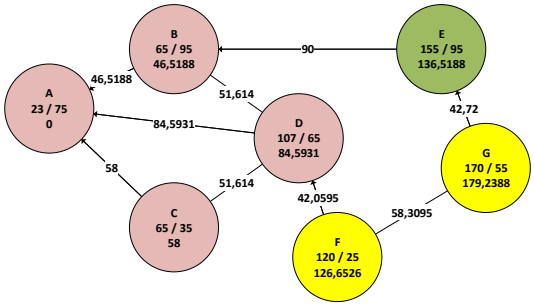

**(c)** Von D aus kann weder der Weg über B noch der über C, verkürzt werden. Der Knoten F wird in $P$ eingefügt, aber die zu erwartenden Kosten sind höher als die von B und C




**(d)** Vom Knoten B aus, wird der Knoten E eingefügt, der aber größere Kosten erwarten lässt als C




**(e)** Vom Knoten C aus kann keinerlei weitere Erkenntnis über den kürzesten Weg gewonnen werden




**(f)** Der Knoten E, den wir als nächstes aus $P$ bekommen, hat nun eine direkte Verbindung zu G, weshalb wir nun den kürzesten Weg gefunden haben

**Abbildung 2.5:** Ablauf einer A-Stern Suche für den günstigsten Pfad von A nach G.
Auf den Knoten steht von oben nach unten der Name, die Koordinaten und die Kosten von A bis zum jeweiligen Knoten
Dunkelgrün: Der gerade aktuell betrachtete Knoten
Gelb: Knoten in der Priority Queue
Rot: Knoten in der Visited Liste

# 3 Aufgabenstellung und Lösungsansatz

## 3.1 Hintergrund

Bei der Fertigung in einer Fabrik, gibt es das Bedürfnis der Optimierung. Da die Realität zu komplex für eine diskrete Optimierung wäre, bedient man sich in der Praxis öfters einer Simulation des Produktionsablaufes um dessen Effektivität zu bewerten.

Ein Teil eines solchen simulationsgetrieben Optimierungsprozesses sind die Optimierungen der Werkerwege.

Werkerwege werden vor allem durch das Fabriklayout bedingt, dessen Entwurf und Änderung Teil jeder Fabrikplanung sind.

## 3.2 Aufgabenstellung

Als Ziel dieser Arbeit wurde die Erstellung eines Konzepts zur Visualisierung von Arbeitswegen innerhalb eines Fabriklayouts, festgelegt. Dieses soll in einem Prototyp umgesetzt werden, der es einem Benutzer ermöglicht durch Änderungen am simulierten Fabriklayout die Produktion zu optimieren.

Die verlangten Leistungen lassen sich wie folgt zusammenfassen:

Es soll zu existierenden Ansätzen recherchiert werden. Des Weiteren soll ein Analysekonzept für die visuelle Darstellung und implizite Veränderungen des Parameterraums einer Simulation, erarbeitet werden. Dieses Konzept soll darauf in einen selbst implementierten Prototyp einfließen, welcher durch das Beschreiben von Anwendungsfällen evaluiert werden soll.

# 4 Verwandte Arbeiten

Im folgenden Kapitel werden einige Arbeiten vorgestellt, welche in Zusammenhang mit meiner Arbeit stehen, oder die als Grundlage für meine Arbeit geholfen haben diese zu erstellen.

## 4.1 IPO.Plan

Die Firma IPO.Plan vertreibt eine kommerzielle Software, „IPO.Log“, welche ihren Usern beim Austakten und Logistikplanung, helfen soll. [Kel12] IPO.Plan hat sich hier auf die Fließbandfertigung spezialisiert.

Es wird hier eine 3D-Visualisierung verwendet, in welche die User verschiedenste Schritte einer Fabrikplanung einbringen.

Nun werden Gant-Auslastungs der Arbeiter und Taktungs-Edition zum Zwecke einer Simulations-Optimierung zur Verfügung gestellt.

Die Ladungsträger werden initial anhand eines Algorithmus, am vom User geplanten Fließband verteilt, können aber vom User verschoben werden, falls er dies wünscht. Wege welche die Fabrikarbeiter während ihrer Tätigkeiten zurücklegen müssen, werden berechnet und auf dem Boden eingezeichnet. Nun kann der Benutzer hingehen und die Arbeiter in einer Animation, zum Leben erwecken und kann das Szenario in seinem Fabrikentwurf simulieren. Auch Einzelarbeitsplätze können so Simuliert werden. Wenn er nun Änderungen hinsichtlich der Ladungsträger- und/oder Station(s)-Positionen wünscht, dann kann er sich errechnen lassen, welche Kosteneinsparungen sich hierdurch ergeben. Des weiteren können Werker-Aufgaben und Tacktung verändert werden.

## 4.2 SmoothScroll

Ist ein Ansatz, welcher eine Lösung für das Problem des genauen Navigieren durch große eindimensionale Datensätzen erlaubt. [WE13]

Wenn man einen großen Datensatz hat, so wird es schwer durch eine Scrollbar innerhalb des Datensatzes zu navigieren, da schon wenig Verschiebung der „Scroll-Box"(auch „Thumb"genannt) große Parameteränderung innerhalb des Datensatzes bedeutet.(Siehe Abbildung 4.2a)

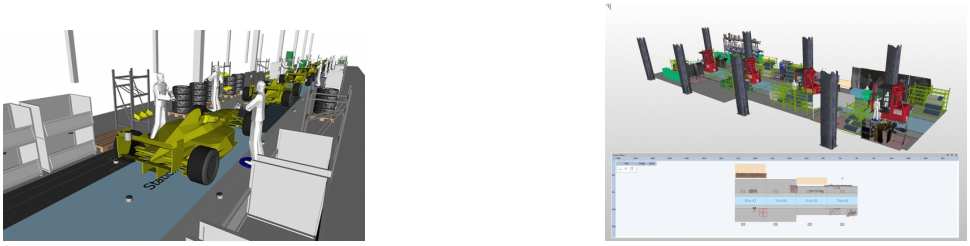
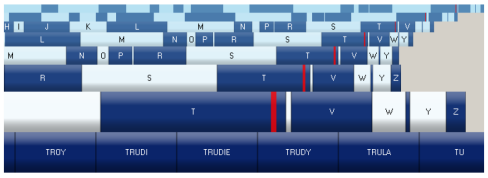


**Abbildung 4.1:** Screenshots aus dem Tool IPO.Log von der IPO.Plan Website [IPO]

Um diesem Problem zu begegnen wurde in dieser Arbeit ein Controll entwickelt, das unter Zuhilfenahme vieler Scrollbars eine detailliertere Filterung von Daten zulässt.

Hier haben viele übereinander angeordnete Scrollbars verschiedene Scrollbereiche. Um nun im Detail zu scrollen, so hat man nun die Möglichkeit seinen Bereich, den man betrachten will, durch die verschiedenen Bars auszuwählen. Die weiter oben, hinten liegenden Scrollbars haben hierbei einen größeren Scrollbereich, und die weiter unten, im Vordergrund, liegenden Scrollbars erlauben eine genauere Auswahl. (Siehe Abbildung 4.2b)

**(a)** Scrollbar mit den dazugehörigen Elementen. Buttons für das Scrollen nach links und rechts. Dazwischen eine Scrollbox, welche die Auswahl repräsentiert, auf einer Fläche, die den ganzen Scrollraum darstellt.




**(b)** Screenshot aus aus [WE13]. Hier kann ein Namensverzeichnis mit dieser speziellen Scrollbar durchforstet werden.

**Abbildung 4.2:** Erklärende Bilder zu [WE13]

## 4.3 Die Visualisierung dynamischer Graphen als Small Multiples

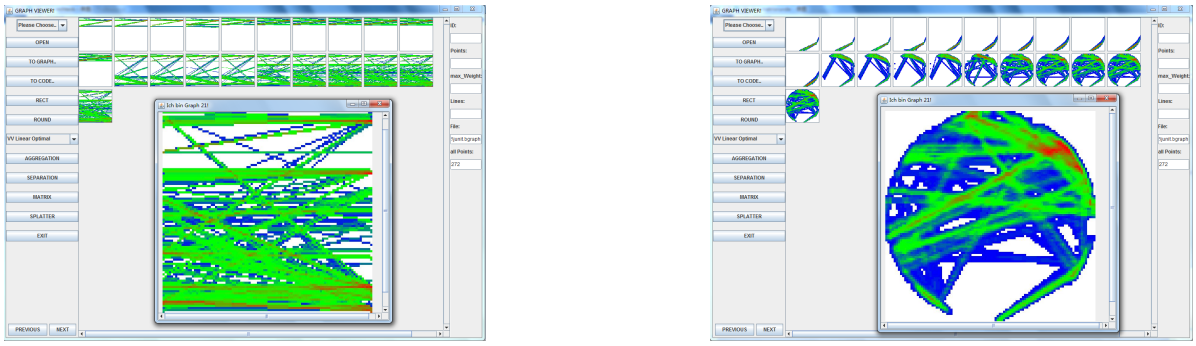
Liang stellt in seiner Diplomarbeit Möglichkeiten vor und implementiert diese auch, um einem User User, Daten eines dynamischen Graphen, graphisch anzuzeigen.[Lia14]

Es wird Edge Splatting [BVB+11] verwendet um für den Benutzer Bilder zu erzeugen, die ihm einen Überblick über die Datenverbindungen im dynamischen Graph zu geben.

Zwei Arten der Anzeige werden vorgestellt.

Darstellungsform „Rechteck": Hier werden die Knotenpunkte des Graphen jeweils entlang der linken und der rechten Kante eines Rechtecks gemappt. Für jede Datenverbindung von Knoten A nach B wird nun einen Linie von der linken Repräsentation des Knoten A zum Rechten Repräsentanten des Knoten B gezogen. Das Resultierende Bild wird per Edge Splatting [BVB+11] für den User zu einer Art Heatmap zusammengefasst.

Darstellungsform „Kreis": Hier werden die Knotenpunkte durch Punkte entlang eines Kreises modelliert. Die Verbindung A-B wird dann als Gerade die den Kreis schneidet visualisiert. Mit dem resultierenden Bild wird gleich verfahren wie bei der Rechteckigen Variante.



**(a)** Darstellungsform ”Rechteck“ **(b)** Darstellungsform ”Kreis“

**Abbildung 4.3:** Screenshots aus der Implementierung von [Lia14]

Mit vielen solcher Plots in Serie kann der Benutzer gut nachverfolgen was sich am dynamischen Graphen getan hat.

# 5 Konzept

In diesem Kapitel soll erklärt werden, welche Wege, für die Umsetzung des Prototyps, begangen wurden, und warum.

Es werden Methoden vorgestellt und argumentiert, weshalb bei der Implementierung ein bestimmter Weg beschritten wurde.

Die aufgetretenen zu lösenden konzeptionellen Aufgaben lassen sich wie folgt gruppieren/zusammenfassen:

1. Anzeigen des Layouts
2. Visualisierung der Werkerpfade innerhalb dieser Anzeige
3. User-Interaktion

## 5.1 Anzeigen des Layouts

Die erste Frage, die sich stellt, ist die Wahl einer Visualisierungsmethode für das Fabriklayout und der Werkerwege.

### 5.1.1 Die Wahl der Dimensionalität

Die intuitiven Möglichkeiten, wären eine 1D Liste mit angegebenen Koordinaten, eine 2D-Darstellung und eine 3D View. Die Wahl fiel am Ende auf die zweidimensionale Möglichkeit, die Argumente für diese Entscheidung, sollen im folgenden Erwähnung finden.

#### Eindimensionale Listen Darstellung

Die 1D Liste, mag für ein wirklich kleines Expertenteam, deren Mitglieder das tatsächliche Layout im Kopf haben, übersichtlicher und auch effizienter zu benutzen sein. Für den normalen Planer eines Layouts ist sie aber in den meisten Fällen eher ungeeignet, da das Layout auch anderen erklärt werden muss. Dagegen steht, dass die Fehleranfälligkeit sehr hoch ist, da Elemente, oder deren Eigenschaften bei großer Anzahl an Einträgen gerne übersehen werden.

Des Weiteren kann das ganze Layout, bei entsprechender Komplexität nicht mehr kognitiv erfasst werden.

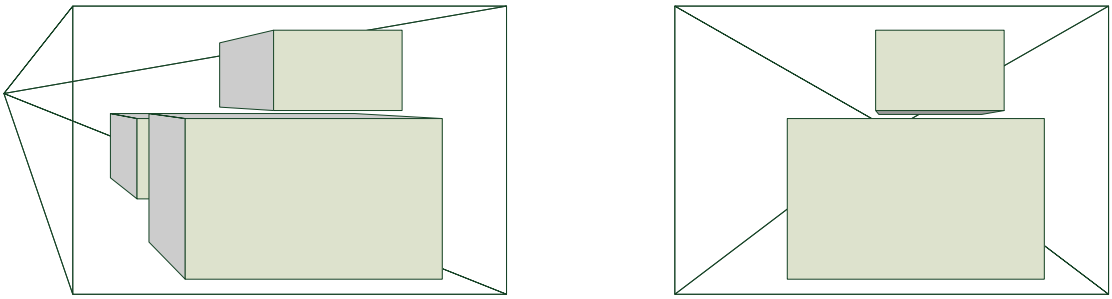
### Dreidimensionale Visualisierung

Die größte Immersion und einen höheren, möglichen Detailgrad kann man mit einer dreidimensionalen Darstellung der Szene erreichen.
Allerdings sprechen auch hiergegen einige Punkte:

**Technologieprobleme** Zum jetzigen Zeitpunkt gibt es noch Probleme mit der Anzeigetechnologie. Es gibt zwar einige Ansätze, ein solches Anzeigegerät zu entwickeln, diese sind aber zum heutigen Zeitpunkt entweder Qualitativ noch nicht ausgereift: z.B. Bei Head-Mounted Displays mit zu geringer Auflösung und Bildwiederholungsrate. Oder sie sind von ihrem Platz und Preisaufwand noch nicht mit ihrem Nutzen vereinbar z.B bei Projektoren mit extrem hoher Bildwiederholungsrate zum Beispiel in Kombination mit Shutter-, oder Polarisationsbrillen.

Falls man sich allerdings auf ein zweidimensionales Anzeigegerät, das dafür technisch ausgereift und kostengünstig ist, beschränkt, so geht die Immersion verloren und auch das Gefühl wo etwas im Raum steht. Auch kommt dann der zweite Punkt umso stärker zum tragen.

**Überdeckung** Das zweite und größere Problem ist, das es bei einer dreidimensionalen Darstellung immer wieder zu Überdeckungen kommt.
Das heißt, man übersieht Details, Objekte oder sogar ganze Komplexe, weil sie sich in der aktuellen Ansicht hinter einem anderen Objekt befinden.



**(a)** Szene mit drei sichtbaren Elementen in einer 3D Visualisierung

**(b)** Die selbe Szene aus einer anderen Perspektive, mit nur 2 sichtbaren Elementen

**Abbildung 5.1:** Beispiel für das Phänomen der Verdeckung bei einer 3D-Visualisierung

### 5.1.2 Farbwahl

Die Farbwahl für die Fabriklayout-Darstellung ist wichtig. Sie muss neutral sein, das möglichst viel andere Informationen mit Farbe in die Darstellung eingefügt werden und zum anderen muss sie für den Nutzer als angenehm empfunden werden, das er auch längere Zeit mit dem Tool verbringen kann. In Anlehnung, an ein Schema des Tools Visual-Studio, welches ähnliche Kriterien bei der Farbwahl erfüllen muss, fiel die initiale Wahl auf ein Farbschema von weißem Vordergrund und Schwarzem Hintergrund, in einem Zukünftigen Produkt, sollte die Farbwahl aber beim Benutzer liegen.

## 5.2 Visualisierung der Wege innerhalb dieser Anzeige

Die Wege der Mitarbeiter sollen innerhalb des Layouts visualisiert werden. Doch zuerst müssen die Wege festgelegt werden.

### 5.2.1 Wegefindung

Als Grundlage für die Werker-Simulation existiert deren Umgebung (das Fabriklayout) und eine Liste ihrer Arbeitsschritte (Siehe auch Kapitel 6.3). Man kann annehmen, dass der Mensch, während der Arbeit, immer versuchen wird den kürzesten Weg zu nehmen, es wird deshalb ein optimaler Weg zwischen den Arbeitsstationen, welche wir aus der Simulation erhalten, gesucht.

Benötigt wird hierfür ein Weggraph.

#### Weggraph

Eine Möglichkeit den Weggraph zu definieren, ist auf Grundlage der Ecken der Elemente des Layouts. (Mit Elementen, sind Objekte gemeint, die in der Lage sind Wege zu versperren, wie Maschinen, Arbeitstische, Säulen etc.) Sie sind die Knotenpunkte des Graphen.

In einem ersten Schritt wird, für alle Ecken der eingelesenen Objekte des Layouts, geprüft welche anderen Ecken für sie erreichbar, sprich nicht durch andere Elemente versperrt, sind. Anhand dieser Information werden als Folge Kanten in den Graphen eingefügt.

Das Kantengewicht entsprecht der Kantenlänge.

Wie schon im Grundlagenkapitel (siehe Kapitel 2) erwähnt handelt es sich beim Weggraphen um einen dynamischen. Der User kann später Elemente des Fabriklayouts mit der Maus beliebig verschieben und drehen, was dazu führt, dass der Wegegraph sich verändert.

Wie schon Kapitel 2 beschrieben, kann bei dynamischen Graphen oft die Laufzeit der Operationen problematisch sein.

Im Folgenden wird davon ausgegangen, dass die Anzahl an Ecken pro Layout-Element nach oben begrenzt ist. Beispielsweise könnte man ein Objekt in einem Layout mit einer Rechteckigen Boundingbox approximieren. Schon beim initialen Einlesen der Daten muss für $n$-Knotenpunkte eine Operation in $\Omega(n^3)$ ausgeführt werden:
Für jeden Knoten muss geprüft wird, ob er eine Kante zu den anderen Knotenpunkten hat. Um die Existenz der Kante zu verifizieren muss sie mit allen Elementen des Layouts geschnitten werden.

Wenn man dem Benutzer eines Echtzeittools nun die Möglichkeit einräumt den Graph zu verändern, so muss nach jeder relevanten User-Interaktion diese Operation (von nun an auch Methode I genannt) von neuem ausgeführt werden.

Um dieses Problem zu umgehen speichert jedes Element, schon beim initialen Erstellen des Graphen, welche Kanten es verhindert hat. Hierbei wird jede Kante nur in einem Element eingetragen (dessen Blockade beim Errechnen zuerst auftritt). Wird es nun durch eine User-Interaktion verschoben, oder gedreht, so kann man auf meist weniger aufwendige Rechenoperationen zurückgreifen:

1. Alle denkbaren Kanten die am verschobenen Element anliegen müssen neu überprüft werden. ($O(n^2)$)
2. Alle bisher existierenden Kanten, müssen mit dem verschobenen Element geschnitten werden um ihren Fortbestand zu überprüfen und sie eventuell als blockiert im Element zu speichern. (worst Case $O(n^2)$)
3. Die gesammelten Kanten, welche durch das Element verhindert wurden, müssen neu überprüft werden. (worst Case $O(n^3)$)

Das alternative Verfahren (von nun an auch Methode II genannt) ist insofern besser, als das es nur im schlimmsten Fall eine Komplexität in $O(n^3)$ bei der Berechnung braucht. Meist blockiert ein Element nicht alle möglichen Verbindungen, außerdem wird die erneute Überprüfung einer blockierten Kante immer nur von **einem** Element ausgelöst, was es noch unwahrscheinlicher macht, dass die Berechnung ähnlich viel Zeit benötigt wie eine neue, komplette Überprüfung aller Kanten.

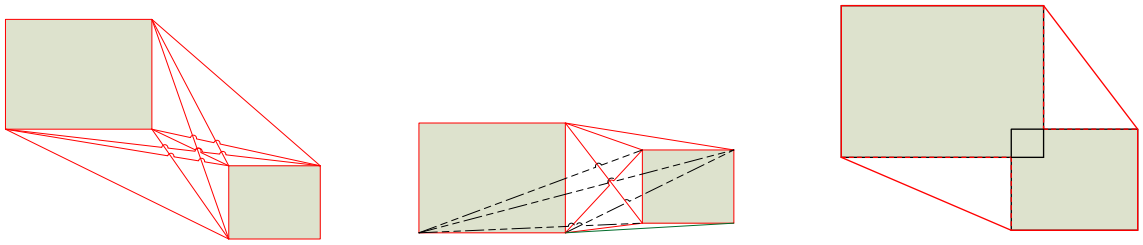
Wenn nach der Veränderung Ecken von Elementen innerhalb anderer Elemente liegen, so sind diese Knotenpunkte dem Weggraphen entnommen, da sie für einen Arbeiter nicht mehr erreicht werden können.

In der konkreten Fabrik kann dies auch vorkommen, wenn zum Beispiel ein Fließband durch eine Maschine verläuft und beide als einzelne Elemente im Layout verzeichnet sind.

Falls nach der Veränderung ein Element einen bisher möglichen Weg versperrt, und/oder einen anderen im Vorhinein nicht möglichen Weg freigemacht hat, so werden Kanten entfernt und andere hinzugefügt.

Auch werden durch Verschiebung von Elementen auf neue Positionen die Kantenlängen geändert, was in diesem Fall einer Kantengewichtsänderung entspricht.

Beispiele solcher Änderungen kann man in Abbildung 5.2 sehen.



**(a)** Grundstellung der beiden Elemente, mit allen möglichen Kanten

**(b)** Durch Verschiebung des rechten Elements nach oben kommt eine neue Kante hinzu (grün) und 4 werden aus dem Wegegraph entfernt (schwarz unterbrochen)

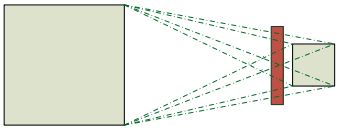
**(c)** Bei dieser Stellung der Elemente fallen zwei Knotenpunkte aus dem Graphen. (Und es entstehen zwei neue Knotenpunkte die Kanten zu diesen „Ecken“ sind gestrichelt eingezeichnet)

**Abbildung 5.2:** Verschiebung eines Elements und die resultierenden Graphänderungen

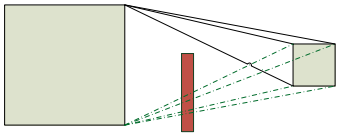
### Spontaner Weggraph

Eine Idee um die komplizierten Berechnungen einzusparen könnte die „spontane"Wegberechnung sein, demnach den Graphen nirgends vollständig zu speichern, sondern immer nur für jeden Schritt der Wegfindung alle möglichen Nachbarn zu errechnen. Das Problem ist, dass ein so konzipiertes Tool nicht skalierbar wäre für viele Datensätze. Für wenige zu findende Pfade mag diese Herangehensweise Vorteile bringen, jedoch können Simulationen manchmal mehrere Monate/Jahre abdecken und sehr viele Arbeiter enthalten, hier wäre die spontane Berechnungïn der Masse viel teurer als die statische, einmalige, komplette Generierung des Weggraphen.

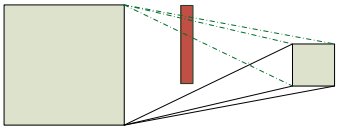
Ein Mittelweg wäre die Ergebnisse der spontanen Nachbarknotensuche zu speichern, und damit den relevanten Teilgraphen komplett zu errechnen und jede Suche maximal einmal durchzuführen. Dadurch könnte man eine Reduktion der Laufzeit erreichen.



**(a)** Grundstellung der Elemente. Die grün gestrichelt dargestellten Kanten werden vom zu verschiebenden Element, rot, als blockiert gespeichert.



**(b)** Während nach Methode I beim Übergang von (a) nach (b) alle Kanten neu geprüft werden: Fakultät aus 8 Kanten ($8! = 40320$) + die Kanten des verschobenen Elements, müssen nach Methode II nur die in (a) grün, gestichelt markierten Kanten neu geprüft werden: 8 + die Kanten des verschobenen Elements.



**(c)** Beim Übergang von (b) nach (c) müssen nach Methode I wieder 40320 Kanten + die Kanten des verschobenen Elements überprüft werden, während nach Methode II nur noch 4 + die Kanten des verschobenen Elements geprüft werden müssen. Man sieht auch, das es nun weniger Grüne, gedachte Kanten gibt, da einige Kanten nun in einem anderen Element, als geblockt gespeichert sind.

**Abbildung 5.3:** Vergleich der beiden Herangehensweisen I und II. Es wurde angenommen, dass die Kollisionsprüfung bei Kanten immer von Links nach Rechts geschieht. Die Kanten des zu verschiebenden Elements wurden aus Gründen der Übersichtlichkeit nicht eingezeichnet.

## 5.3 User-Interaktion

Der Benutzer soll mit dem Anzeigegerät interagieren können. Auf diese Weise sollen die visualisierten Layout-Daten manipuliert werden. Die veränderten Daten müssen in einer ähnlichen Form wie sie eingelesen wurden Speicherbar sein.

Die hervorgerufenen Änderungen der Manipulation sollen dem Benutzer in der Visualisierung und durch Kennzahlen rückgemeldet werden. Dadurch hat er die Möglichkeit die Veränderungen zu bewerten und kann erkennen ob eine Optimierung seines Produktionslayouts stattgefunden hat.

# 6 Implementierung

Ein großer Teil dieser Arbeit ist die Umsetzung des Konzepts in einem Prototyp.

## 6.1 Verwendete Programme und Tools

Das Konzept wurde in der Programmiersprache **C#** implementiert.
Als Programmierumgebung diente hierbei **Visual Studio**, in den Versionen 2013 und 2015 mit dem auf dem **.NET Framework 4.5** aufbauenden Framework WPF (**W**indows **P**resentation **F**oundation [Mic06])).

Die komplette Arbeit wurde auf Computern entwickelt die unter Windows in den Versionen 8.1 bzw. 10 liefen. Der Prototyp wurde auch auf diesen Plattformen getestet, er sollte allerdings auf allen Windows Versionen, die das .Net Framework 4.5 unterstützen, laufen.

Für die Erstellung der Grafiken dieser Ausarbeitung wurde, falls nicht anders angegeben, das Microsoft Programm **Microsoft Visio** benutzt.

Zum editieren, sowie öffnen der CSV-Dateien, wurde **Notepad++** [Ho+08] verwendet.

Bei den genutzten Eingangsdaten für den Prototyp handelt es sich um Simulationsexport-Daten des kommerziellen Tools **IMV** (**I**ntegrated **M**anufacturing **V**alidation) der Firma **iFAKT GmbH** [iGmbb], die für die Arbeit zur Verfügung gestellt wurden.

## 6.2 Umsetzung des Konzepts

Folgend soll aufgezeigt werden, welche Aspekte des Konzepts umgesetzt wurden, und in welcher Art.

### 6.2.1 Suchalgorithmus

Beim Algorithmus der die Wegsuche innerhalb des Weggraphen vornimmt, fiel die Wahl auf den A* Algorithmus (Abschnitt 2.2). Hauptargument für diese Wahl, war dessen Eigenschaft, in kürzester Zeit, den garantiert schnellsten Weg zu finden.

Bei der Wegsuche verwendet der A* Algorithmus eine Heuristik um den Weg zu suchen. Wenn diese auf dem euklidischen Abstand der Knotenpunkte basiert, so erscheint mit die Herangehensweise der eines Menschen sehr ähnlich, was auch zur Wahl beigetragen hat.

Ein großer Nachteil, der durch einen deterministischen Algorithmus entsteht, sind extrem lange Berechnungszeiten, falls der gesuchte Weg in einem komplexen Layout nicht existiert und erst alle erreichbaren Wege erforscht werden müssen.

Sinnvolle Abbruchkriterien, lassen sich, ohne erheblichen Mehraufwand bei der Grapherstellung, welche, wie oben beschrieben, eh schon lange dauern kann, nicht formulieren. Allerdings ist hier wohl noch Verbesserungspotential vorhanden.

### 6.2.2 Heatmap

Wegen der schlechten Skalierbarkeit beim Rendern von Elementen in WPF (**W**indows **P**resentation **F**oundation [Mic06])) Probleme mit der Heatmap.

Leider kommen viele Renderprozesse nur mit einer begrenzten Menge (Halb-)Transparenter Elemente zurecht. Diese künstliche Schranke hat durchaus ihre Berechtigung, da die Laufzeit des Renderprozesses sonst für deren Anwendungsgebiet nicht mehr verhältnismäßig wäre. Es wurden, für diese Arbeit, einige Ansätze für eine CPU basierte Renderung der Heatmap überprüft, von denen allerdings keine den Anforderungen genügt, und so wurde letztendlich ein auf einem Shader basierter Lösungsansatz nach [Dar10] eingeführt.

Der Vorteil eines Shaders ist, das die Heatmap auf der GPU berechnet wird, was zu einer deutlich schnellerem Renderprozess führt.

Die gerenderte Heatmap wird der Oberfläche als statische Bitmap zugefügt.

### 6.2.3 Oberfläche und Funktionen

Die graphische Benutzeroberfläche teilt sich in drei Bereiche (siehe Abbildung 6.1), die im folgenden genauer beschrieben werden:

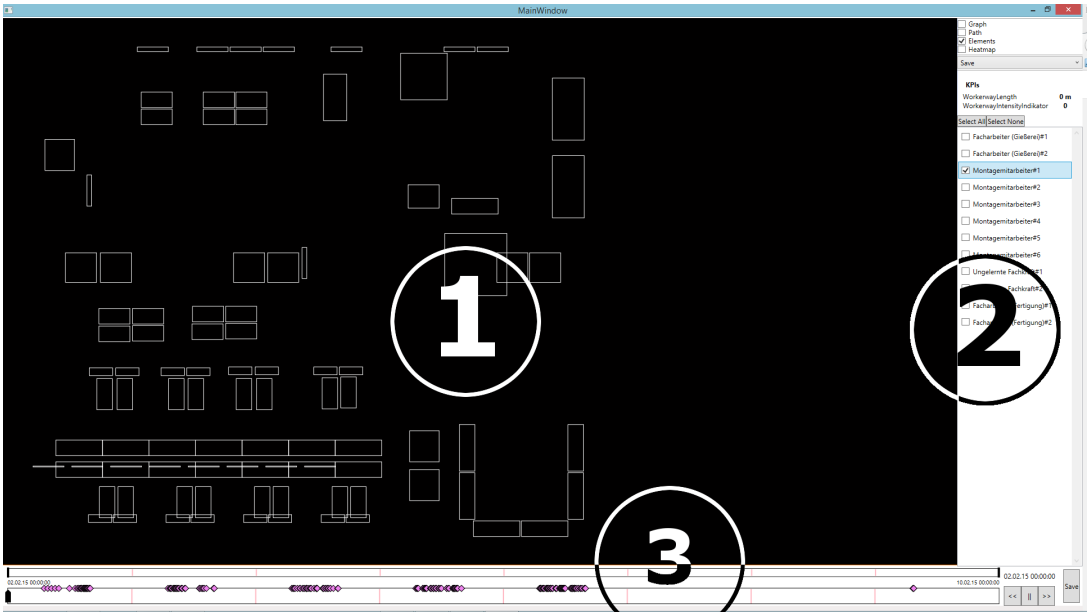



**Abbildung 6.1:** Die drei Bereiche der Benutzeroberfläche:
1. Das **Canvas-Element** mit der 2D-Repräsentation des Fabriklayouts
2. Der **Side-Bereich** mit Einstellungsmöglichkeiten für den User
3. Die **Timeline** oder auch Zeitstrahl

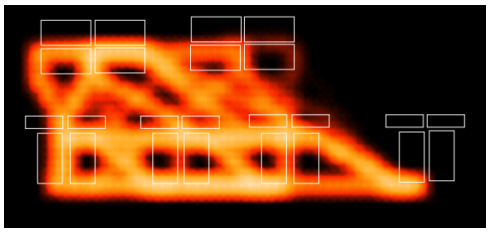
### Canvas-Element

Der erste Bereich, des Canvas-Elements, zeigt auf schwarzem Hintergrund in weißen Polygonen die Konturen der Elemente des 2D-Layouts an.

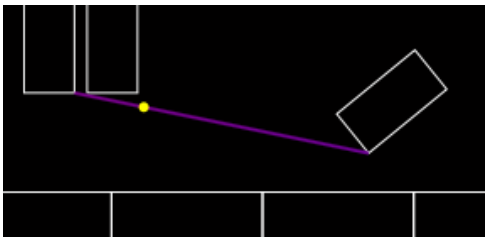
Die einzelnen Layout-Elemente lassen sich mit der linken Maustaste per Drag and Drop verschieben und mit der rechten drehen. Die graphische Rückmeldung des Programmes ist Abbildung 6.3 zu sehen.

Neben den Layout Elementen werden in diesem Bereich der Oberfläche auch die Werkerwege als Linien, eine Heatmap, und Mitarbeiter als kleine Punkte eingezeichnet (siehe Abbildung 6.2).

Durch die oben beschriebenen Interaktionsmöglichkeiten kann der User sein Layout verändern. Die Änderungen veranlassen eine Neuberechnung der Werkerwege, der Heatmap und der Werkerposition. Diese Visualisierungen, so wie die Kennziffern im Side-Bereich (siehe weiter unten) sollen dem Benutzer Rückmeldung über die Auswirkungen der Änderung geben.

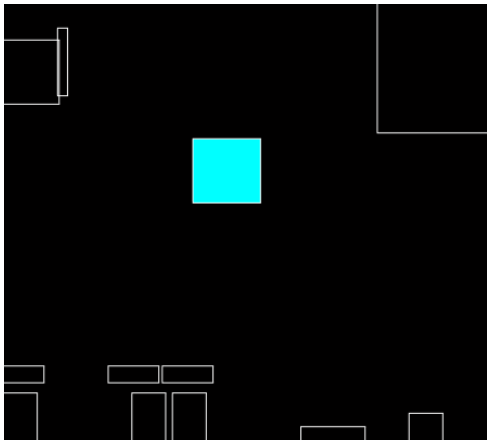


**(a)** Screenshot der Heatmapvisualisierung des Prototyps

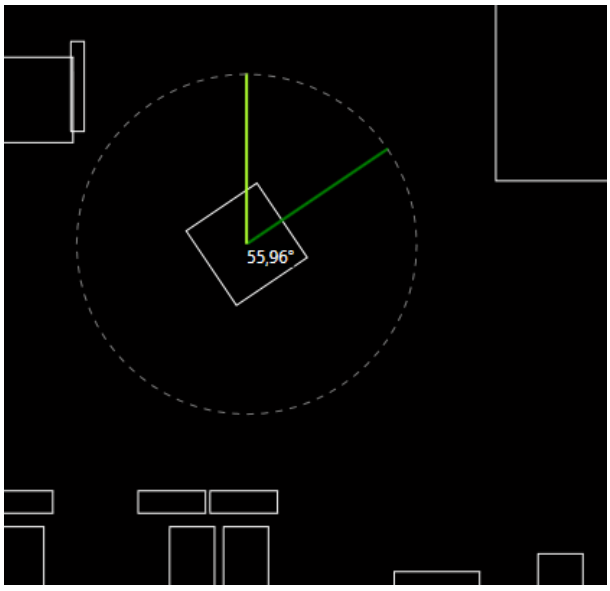


**(b)** Screenshot aus dem Prototyp der einen Werkerpfad mit eingezeichneter Werkerposition zeigt

**Abbildung 6.2:** Beispielhafte Screenshots aus dem Prototyp



**(a)** Ein Element wird durch Drag and Drop mit der linken Maustaste verschoben, und färbt sich während des Verschiebe-Vorgangs Blau.




**(b)** Durch Ziehen mit gedrückter rechter Maustaste wird ein Element um seinen Mittelpunkt gedreht. Die gelbe Linie zeigt die Ausgangsstellung (0°) während die grüne Linie den gerade ausgewählten Drehwinkel angibt, welcher auch als Zahl im Element angezeigt wird. Durch drücken von Alt bzw. Leertaste kann der User eine Rasterisierung auf ganze Gradzahlen, bzw. auf ein 45 Grad Raster, erwirken.

**Abbildung 6.3:** Interaktionsmöglichkeiten mit dem Canvas-Bereich der Oberfläche

### Side-Bereich

Im Side-Bereich hat der Nutzer die Möglichkeit die Informationen die ihm im Canvas-Element angezeigt werden sollen zu variieren. Von oben nach unten sind das:

**Graph** Diese Einstellung ermöglicht es den kompletten Wegegraphen einzeichnen zu lassen. Diese Option ist vor allem dann von Vorteil, wenn die "Wege-Wahl"der Werker dem User nicht schlüssig erscheint, da man in dieser Ansicht erkennen kann, welcher Weggraph für die Berechnung benutzt wurde.

**Path** Diese Option blendet die errechneten Werkerwege ein und aus.

**Elements** Mit dieser Option lassen sich die Layout-Elemente aus und einblenden.

**Heatmap** Mit dieser Option kann der User sich eine Heatmap der ausgewählen Werkerwege in den Canvas-Bereich einzeichnen lassen um so Bereiche zu erkennen an denen besonders oft Werker laufen.

**DatenInput** Mit der Dropdown-Liste lässt sich aus den verschiedenen Eingangsdatensätzen wählen.

**KPIs** Mit den **K**ey **P**erformance **I**ndicators (auch Leistungskennzahlen) kann der User schnell erkennen, welche Effekte seine Änderungen haben. Ihm werden hier zwei Kennzahlen angezeigt:

a) Die Wegstrecke der ausgewählten Werker, welche er zum Optimieren möglichst verkleinern will.

b) Eine Bewertung darüber, wie dicht die Werker aufeinander stehen. Auch diesen Wert sollte für eine Optimierung, klein gehalten werden.

Während der erste Wert noch intuitiv verständlich ist, so ergibt sich der zweite Wert aus den Berechnungen der Heatmap und ist das Maximum der Werkerverteilung auf einer engmaschige Rasterisierung, unter Zuhilfenahme eines Gaußkernels. (siehe Kapitel 2.3)

**Workerliste** In dieser Liste werden alle Werker die im Datensatz vorliegen aufgelistet und der Benutzer kann sich, auch unter Zuhilfenahme der beiden Buttons, auswählen, welche Werker für die Anzeige im Canvas, bzw. für die Errechnung der Werkerwege-KPI relevant sein sollen.

### Timeline

Im dritten Bereich befindet sich von links nach rechts, von oben nach unten:

**Time-Span Picker** Hier kann der Benutzer die angezeigten Daten filtern. Dies geschieht durch Auswahl eines Zeitraums auf dem Zeitstrahl, der den kompletten Simulationszeitraum abdeckt, mit den beiden schwarzen Datepickern. Die Rosa eingezeichneten Linien verdeutlichen den Beginn eines neuen Tages.

**Zeitstempel** Hier wird der visualisierte Zeitpunkt angezeigt.

**Timeline** Das namensgebende Element ermöglicht es dem User per verschieben des Reglers einen Zeitpunkt auszuwählen. Der Bereich, welcher zur Auswahl steht, wird durch den Time-Span Picker bestimmt. Die Positionen, der Arbeiter zum ausgewählten Zeitpunkt, werden im Canvas-Bereich, durch Punkte visualisiert. Je nachdem wie groß der Zeitraum ist, der gerade durch Timeline auswählbar ist, sind die Tage, Stunden und Minuten, durch rosa, silberne und graue Linien, zur besseren Orientierung des Users eingezeichnet.

Auf dem oberen Rand der Timeline sind, falls vorhanden, die Arbeitsschritte der ausgewählten Werker, als kleine Rauten eingezeichnet. Diese Rauten geben bei überfahren mit dem Cursor Auskunft über den Zeitpunkt des Events und den Werker, den es betrifft.

**Play-Break Button** Für eine animierte Simulation kann der Benutzer hier ein Fortlaufen des visualisierten Zeitpunktes erreichen. Dies ist in beide Richtungen möglich, sprich in die Vergangenheit und in die Zukunft. Durch mehrfaches Klicken auf einen der Pfeile beschleunigt sich die Animation. Der Button mit den beiden parallelen Balken, hält die Animation an.

**Save** Diese Operation dient dazu das, durch Interaktion veränderte, Fabriklayout als CSV-Datei zu exportieren. Diese Exportdateien können später als neue Eingangsdaten verwendet werden (siehe Kapitel 6.3).

## 6.3 Struktureller Aufbau der Eingangsdaten

Die Eingangsdaten für das Tool, entsprechen den Exportdaten eines Simulationsdurchlaufes durch das Tool IMV der Firma IFakt GmbH [iGmbb] und liegen in mehreren CSV-Dateien vor. In diesem Kapitel soll erklärt werden, was CSV-Dateien sind und welche Daten, in welcher Struktur, nötig sind um den Prototyp zu verwenden.

### 6.3.1 CSV-Dateien

CSV (**C**omma **S**eparated **V**alue) Dateien, sind eine effiziente Möglichkeiten Datentabellen zu speichern.

Es sind Text-Dateien, welche in der ersten Zeile die Spaltenheader und in jeder weiteren Zeile die Daten der Tabelle, jeweils per Semikolon getrennt, beinhalten.

Der Vorteil gegenüber den, meist benutzen, Excel-Dateien ist, das CSV-Dateien sich schneller einlesen lassen, weniger Speicher verbrauchen und man keine kommerzielle Software erwerben muss um sie zu editieren.

### 6.3.2 Struktur

Damit der Prototyp auch ohne die Simulationssoftware verwendet werden kann, wird hier die Datenstruktur spezifiziert, welche von der Software erzeugt wird. Durch die Offenlegung des Formats, ließen sich verwendbare Daten auch von einem anderen Tool generieren.

Das Tool benötigt nicht den vollen Umfang der ursprünglich bereitgestellten Simulationsdaten. Die Anforderungen beschränken sich auf folgende Tabellen und Tabellenwerte.

**LayoutElements.csv** Enthält die Layout Elemente mit Positionsdaten (siehe Tabelle 6.1).

**WorkerItems.csv** Liste der in der Simulation vorhandenen Werker (siehe Tabelle 6.2).

**SimComponents.csv** Enthält Verbindungsdaten zwischen, in der Simulation benutzten Objekten und deren geometrischen Eigenschaften in der LayoutElements Tabelle (siehe Tabelle 6.3).

**WorkerActivities.csv** Hier sind die Simulationsevents, mit Start und Endzeitpunkt, aufgelistet, sowie an welchem Simulationsobject diese stattfinden (siehe Tabelle 6.4).

**WorkerActivityWorkerLinks.csv** Enthält die Information welcher Werker welchem Simulationevent zugeordnet ist (siehe Tabelle 6.5).

Die Verbindungen der Datentabellen kann man noch einmal in Abbildung 6.4 sehen.

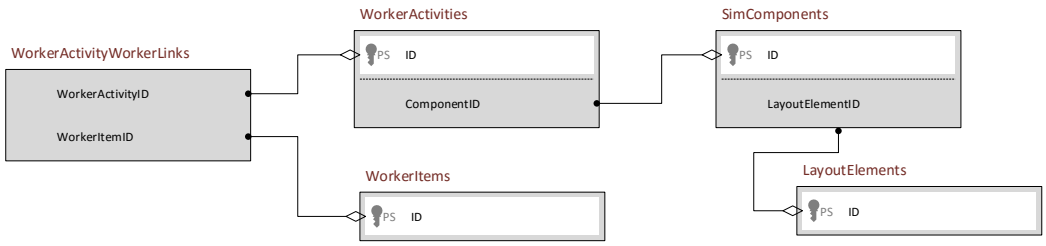



**Abbildung 6.4:** Diagramm der Datenverbindungen

| Header | Typ | Beschreibung |
|---|---|---|
| **ID** | *guid* | ElementID-Wert, zum Binden an andere Tabellen |
| **Name** | *string* | Der Name des Elements, kann auch leer sein |
| **LocationX** | *double* | X Wert der Position des Elements innerhalb des Fabriklayouts |
| **LocationY** | *double* | Y Wert der Position des Elements innerhalb des Fabriklayouts |
| **BoundsX** | *double* | Ausdehnung des Elements in X Richtung |
| **BoundsY** | *double* | Ausdehnung des Elements in Y Richtung |
| **RotationZ** | *double* | Der Winke der Rotation (im Uhrzeigersinn) des Elements |

**Tabelle 6.1:** LayoutElements.csv Spalten

| Header | Typ | Beschreibung |
|---|---|---|
| **ID** | *guid* | WorkerID-Wert, zum Binden an andere Tabellen |
| **WorkerItem** | *string* | Bezeichnung des Werkers, dient zur Identifikation |

**Tabelle 6.2:** WorkerItems.csv Spalten

| Header | Typ | Beschreibung |
|---|---|---|
| **ID** | *guid* | ComponentID-Wert zum Binden an andere Tabellen |
| **LayoutElementID** | *guid* | LayoutElementID-Wert zum Binden (Tabelle 6.1) |

**Tabelle 6.3:** SimComponents.csv Spalten

| Header | Typ | Beschreibung |
|---|---|---|
| **ID** | *guid* | WorkerActivityID-Wert zum Binden an andere Tabellen |
| **ComponentID** | *guid* | ComponentID-Wert zum Binden (Tabelle 6.3) |
| **StartDate** | *datetime* | StartDate of the Activity |
| **endDate** | *datetime* | EndDate of the Activity |

**Tabelle 6.4:** WorkerActivities.csv Spalten

| Header | Typ | Beschreibung |
|---|---|---|
| **WorkerItemID** | *guid* | WorkerItemID-Wert zum Binden (Tabelle 6.2) |
| **WorkerActivityID** | *guid* | WorkerActivityID-Wert zum Binden (Tabelle 6.4) |

**Tabelle 6.5:** WorkerActivityWorkerLinks.csv Spalten

# 7 Evaluation

In diesem Kapitel werden Anwendungsfälle für das im Rahmen der Arbeit erstellte Tool vorgestellt. Dazu ist erst einmal festzustellen, welche Anwendungsfälle vorliegen können. (Abbildung 7.1)

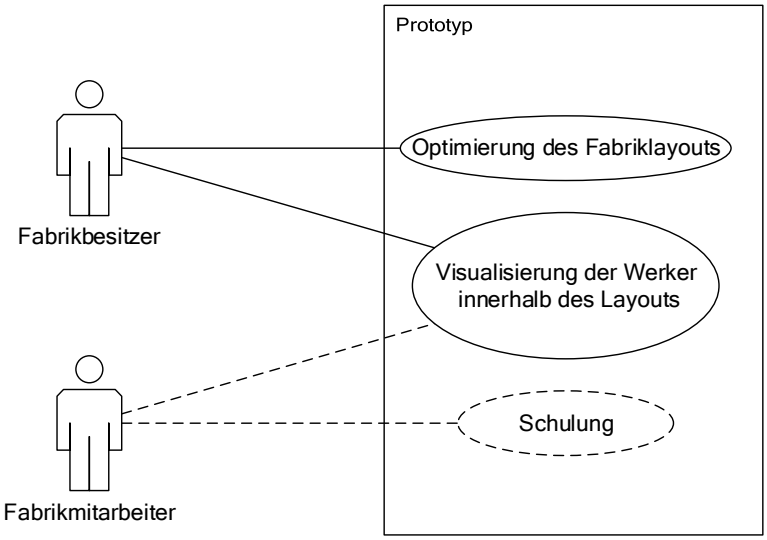
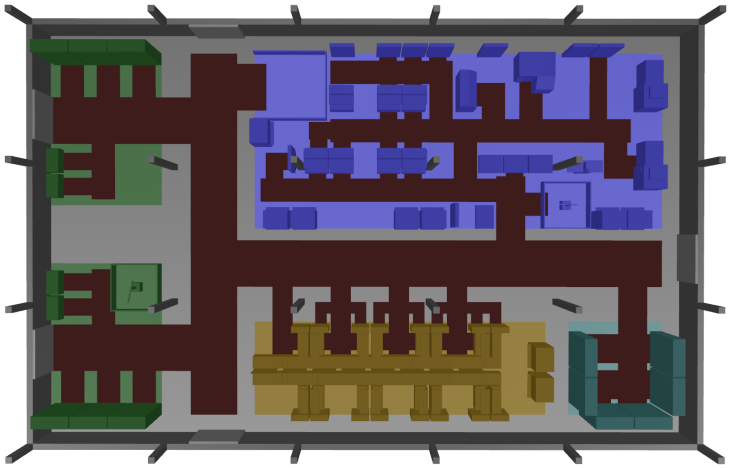



**Abbildung 7.1:** UseCase-Diagramm

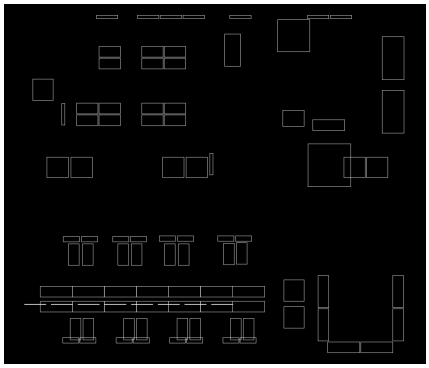
## 7.1 Beispielhaftes Layout

Die folgenden Anwendungsfälle verwenden dasselbe Ausgangslayout, welches von der Firma iFAKT zur Verfügung gestellt wurde.
Das Layout leitet sich aus einem beispielhaften 3D Modell für den, öffentlich verfügbaren, Demo-Simulationsdatensatz [iGmba] ab. Es wurden allerdings Änderungen am Datensatz vorgenommen, so sind Elemente weggelassen, oder verschoben worden.
Wichtig für die Use-cases ist, das es sich um ein Layout handelt, welches in der realen Produktion genau so vorkommen könnte.

Das 3D Modell ist in Abbildung 7.2 zu sehen.



**(a)** 3D-Modell zum Demo-Produktionsdatensatz [iGmba]



**(b)** 2D-Layout, welches für die Use-Cases verwendet wurde

**Abbildung 7.2:** 3D-Layout und das daraus abgeleitete 2D Layout

## 7.2 Optimierung des Fabriklayouts

Dieser Anwendungsfall ergibt sich vor allem für Fabrikplaner, die Unterstützung bei der Planung neuer Werkhallen bzw. ganzer Fabrikkomplexe benötigen.

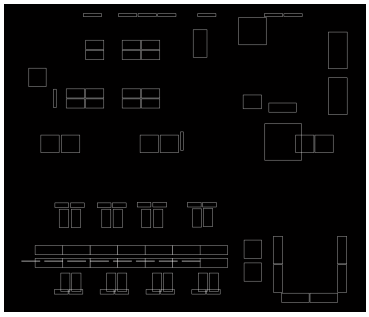
Bevor eine Fabrikhalle gebaut/eingerichtet wird, erfolgt eine Simulation der späteren Fertigung simuliert. Mit den Ergebnissen der Simulation arbeitet nun der Prototyp und es besteht die Möglichkeit das anfänglich erstellte, für die Simulation verwendete, Layout zu verändern. Ziel ist eine Optimierung der Arbeiterwege.

Der weitere Arbeitsablauf wird anhand der Abbildung 7.3 und 7.4 beschrieben.

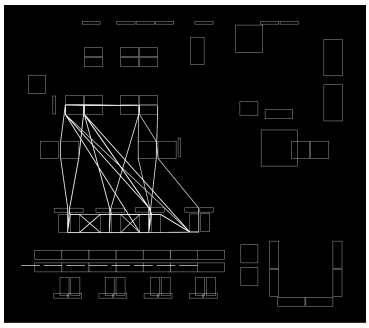
Da ein Fabrikplaner die real-Räumlichen Gegebenheiten kennt, achtet er darauf das die Elemente nur an Stellen verschoben werden, welche sinnvoll sind. Dies ist wichtig, da Beschränkungen wie, zum Beispiel, Kabelanbindungen u.Ä. im Prototyp nicht abgebildet werden.

Durch die visuelle Darstellung und die Änderungen der KPIs kann der Benutzer abschätzen ob seine Änderungen eine Optimierung für sein Layout darstellen.

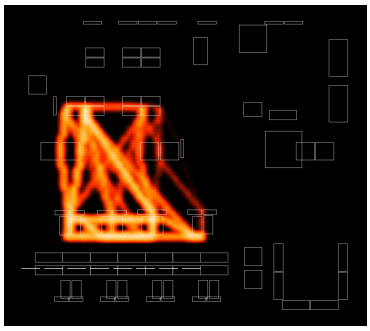
Als Fazit kann man sehen, dass durch das Verwenden des Tool, eine Änderung am Planungslayout vorgenommen wurde, welche dazu führt, dass die Arbeit in der realen Produktion später effizienter, da weniger Zeit auf der Strecke bleibt, und für die Mitarbeiter angenehmer sein wird.



**(a)** Anfängliches Layout, der Simulationsdaten

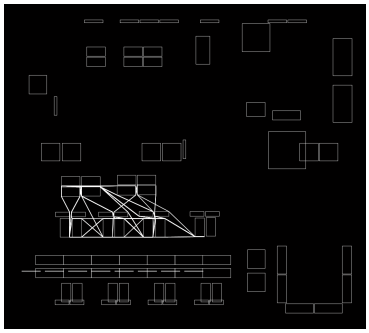


**(b)** Das selbe Layout mit eingezeichneten Werkerwegen. Man sieht das die Werkerwege sich für die simulierten Produktionsschritte auf einen Teilbereich des Layouts beschränken, und das sie unnötig weit sind.

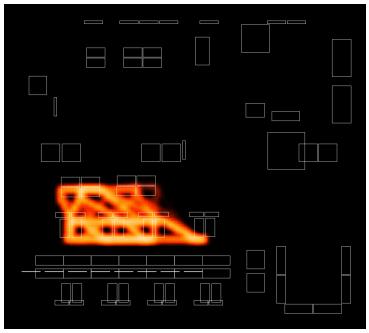


**(c)** Ein Blick auf die Heatmap verrät dem Betrachter, dass die langen Werkerwege sogar relativ häufig benutzt werden, weshalb sich eine Änderung des Layouts höchstwahrscheinlich lohnen würde.

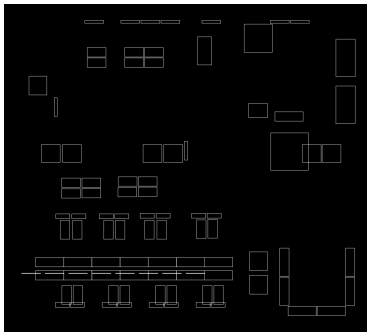
**Abbildung 7.3:** Layout des ersten Use-Cases, vor der Änderung



**(a)** Durch Verschiebung der, für die betrachteten Produktionsschritte, nötigen Layout-Elemente konnte eine Verringerung der Mitarbeiterwege erreicht werden.



**(b)** Auch die Heatmap zeigt eine Verkleinerung des durch die Wege betroffenen Bereichs innerhalb des Layouts



**(c)** Das veränderte Layout. Durch den Speicherbutton lässt sich es sich im bekannten CSV-Format exportieren (Siehe Abschnitt 6.3.1)

**Abbildung 7.4:** Verändertes Layout des ersten Anwendungsfalles

## 7.3 Visualisierung der Werker innerhalb des Layouts

Angenommen, der Fabrikplaner möchte einen neuen Roboter in seine Fabrik integrieren, der einen qualifizierten Mitarbeiter in der Umgebung braucht, der dessen Fehlerprotokoll einmal am Tag auf Einträge überprüft. Als Erstes muss er darauf achten, wo der Roboter am besten im Fabriklayout untergebracht werden kann, möglichst ohne Werkerwege zu versperren, oder zu verlängern. Hierbei kann das Tool, durch die Visualisierung dieser hilfreich sein.
Nun allerdings, muss er darauf achten, dass ein dafür ausgebildeter Mitarbeiter einmal am Tag die Wartung vornehmen kann. Dazu kann er sich nun mit dem Tool, und der Timeline, anzeigen lassen, welcher Mitarbeiter sich in der Fertigung zum fraglichen Zeitraum in der Nähe des im ersten Schritt gefundenen Platzes aufhält und kann daraufhin mit dieser Zusatzinformation entscheiden, wer die Fortbildung zur Wartung bekommt. (Siehe Abbildung 7.5)

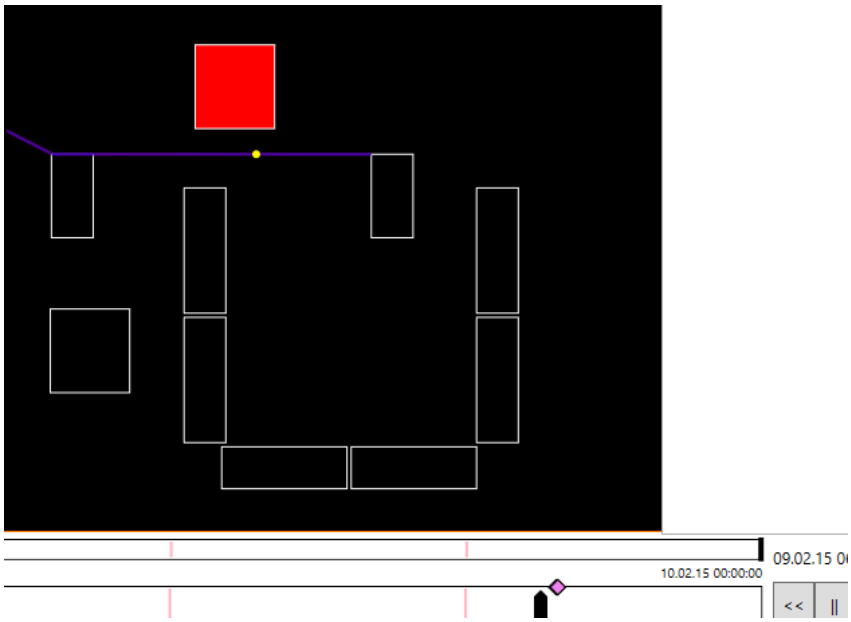



**Abbildung 7.5:** Position eines Werkers, zum ausgewählten Zeitpunkt, in der Nähe des Roboters, hier rot markiert. (Abbildung entspricht nicht der genauen Darstellung des Tools)

# 8 Zusammenfassung und Ausblick

## 8.1 Zusammenfassung

In der Arbeit wurde Konzept entwickelt und in einem Prototyp umgesetzt, welches in implementierter Weise einem Anwender helfen soll, die Werkerwege innerhalb seines Fabriklayouts zu analysieren und zu optimieren.

## 8.2 Ausblick

Bei dem in dieser Arbeit entwickelten Konzept, und dessen Umsetzung im Prototyp, gibt es viele Aspekte die verbessert werden können, oder Ansätze die besser geeignet sind.

Die Art der Visualisierung könnte sich mit der Weiterentwicklung der dreidimensionalen Darstellungsmöglichkeiten verändern, so sind begehbare Plots des Layouts in eine leere Fabrikhalle für den Planer, mit Augmented-Reality-Technologien denkbar.

Auch bisherigen Rechenleistungsprobleme könnten durch andere/optimierte Algorithmen, vor allem in der Grapherstellung und er Wegfindung, besser begegnet werden.

Auch eine bessere Rückmeldung an den Benutzer über Optimierungen durch seine Aktionen wären wünschenswert.

Das ganze Konzept sollte auch im Planungsprozess eine Fabrik immer in Verbindung mit einer Simulation der restlichen Fertigung ablaufen, um zu validieren, dass die Optimierungen der Werkerwege nicht größere negative Folgen für den restlichen Produktionsablauf hat.

# Literaturverzeichnis


[BH+] C. A. Brewer, M. Harrower et al. *ColorBrewer 2.0*. http://www.ColorBrewer2.org/ (Zitiert auf S. 16, 17).

[BVB+11] M. Burch, C. Vehlow, F. Beck, S. Diehl und D. Weiskopf. „Parallel edge splatting for scalable dynamic graph visualization“. In: *Visualization and Computer Graphics, IEEE Transactions on* 17.12 (2011), S. 2344–2353 (Zitiert auf S. 22, 23).

[Dar10] N. Darnell. „WPF: Heat Maps“. In: *Nick Darnell's Blog*, *http://nickdarnell.com/wpf-heat-maps/* (2010) (Zitiert auf S. 32).

[Die12] R. Diestel. *Graph theory*. 4. Aufl. Springer, 2012 (Zitiert auf S. 11).

[HNR68] P. E. Hart, N. J. Nilsson und B. Raphael. „A formal basis for the heuristic determination of minimum cost paths“. In: *Systems Science and Cybernetics, IEEE Transactions on* 4.2 (1968), S. 100–107 (Zitiert auf S. 13).

[Ho+08] D. Ho et al. *Notepad Plus Plus*. https://notepad-plus-plus.org/. 2008 (Zitiert auf S. 31).

[iGmba] iFAKT GmbH. *Demodatensatz für IMS*. https://demo.ifakt-ims.de/ims-portal5/help (Zitiert auf S. 39, 40).

[iGmbb] iFAKT GmbH. *Produktvorstellung IMS (IMV)*. https://www.ifakt.de/software/ims/ (Zitiert auf S. 31, 36).

[IPO] IPO.Plan. *IPO.Log DemoVersion*. http://www.ipoplan.de/en/inhalt/ipolog-demo-version.199 bzw. http://www.ipoplan.de/de/inhalt/ipolog-demoversion.194 (Zitiert auf S. 22).

[Kel12] M. Kellermann. „Interaktive Prozessplanung und Logistikoptimierung in 4D mit IPO.Log v3“. In: *Vernetzt Planen und Produzieren TU Chemnitz* (2012) (Zitiert auf S. 21).

[Lia14] S. Liang. *Die Visualisierung dynamischer Graphen als Small Multiples*. Dipl. Arbeit. 2014 (Zitiert auf S. 12, 22, 23).

[Mic06] Microsoft. *Introducing Windows Presentation Foundation*. MSDN-Portal. Sep. 2006. URL: https://msdn.microsoft.com/en-us/library/aa663364.aspx (Zitiert auf S. 31, 32).

[WE13] M. Wörner und T. Ertl. „Smoothscroll: A multi-scale, multi-layer slider“. In: *Computer Vision, Imaging and Computer Graphics-Theory and Applications* 274 (2013), S. 142–154 (Zitiert auf S. 21, 22).

Alle URLs wurden zuletzt am 12. 05. 2016 geprüft.

**Erklärung**

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

---

Ort, Datum, Unterschrift