DOI: 10.22032/dbt.57807

Simulation in Produktion und Logistik 2023
Bergmann, Feldkamp, Souren und Straßburger (Hrsg.)
Universitätsverlag Ilmenau, Ilmenau 2023
DOI (Tagungsband): 10.22032/dbt.57476

Ereignisdiskrete Modellierung autonomer Transportfahrzeuge mittels Open-Source Software

Discrete-event modelling of autonomous transport vehicles using open-source software

Viktor Artiushenko, Marcel Müller, Tobias Reggelin, Otto-von-Guericke-Universität Magdeburg, Magdeburg (Germany), viktor.artiushenko@ovgu.de, marcel1.mueller@ovgu.de, tobias.reggelin@ovgu.de

Sebastian Lang, Fraunhofer-Institut für Fabrikbetrieb und Automatisierung (IFF), Magdeburg (Germany), sebastian.lang@iff.fraunhofer.de

Abstract: This contribution addresses the need for improved methods in modelling and simulating transportation vehicles in discrete event simulation (DES), since current commercial software solutions suffer from a limited adaptability, high costs, and slow performance. We introduce an object class that enables the addition of freely moving transport resources to open-source DES libraries, including collision-free motion modelling. Applicable to all object-oriented programming languages, this class design extends the functionality of existing open-source software. A component for an intralogistics transport vehicle was developed for the Python library Salabim, and its functionality was successfully verified in a two-vehicle simulation environment.

1 Einleitung

Simulationsstudien zum Materialfluss zielen im operativen Betrieb gewöhnlich darauf ab, Transport- und Lagerkosten zu reduzieren und letztendlich die Produktionszeit zu verkürzen, was zu einem wichtigen Wettbewerbsvorteil werden kann. Für die Analyse von Materialflüssen eignen sich ereignisdiskrete Simulationsmodelle (DES – vom engl. Begriff „Discrete Event Simulation) (Alves et al. 2009). Mittels DES können Systeme und Prozesse unterschiedlicher Komplexität und in einem beliebigen Detaillierungsgrad abgebildet werden. Durch DES werden in der Produktion primär die Produktionsplanung und -steuerung, Leistungsbewertung von Produktionssystemen, Supply-Chain-Optimierung, sowie das Bestandsmanagement unterstützt (Tako und Robinson 2012).

Die Transportfahrzeuge und deren Management spielen bei der Materialflussplanung eine wichtige Rolle, um sicherzustellen, dass das richtige Material zur richtigen Zeit

am richtigen Ort zur Verfügung steht. Mit Transportfahrzeugen sind in diesem Beitrag sowohl konventionelle Gabelstapler als auch moderne autonome Mittel wie fahrerlose Transportsysteme (FTS) oder autonome mobile Roboter (AMR) gemeint.

In den letzten Jahrzehnten wurden Probleme der Materialflussplanung in Produktionssystemen mit Transportfahrzeugen in mehreren Publikationen, wie beispielsweise von Herazo-Padilla et al. (2013), Sun et al. (2018), Fransen et al. (2020), Müller et al. (2022), Kousi et al. (2019) mittels Simulationen analysiert. Die durchgeführte Literaturrecherche zeigt, dass in den meisten wissenschaftlichen Arbeiten kommerzielle Softwarelösungen zur Optimierung des Materialflusses verwendet wurden. In einigen dieser Arbeiten wurden jedoch nur sehr abstrakte Modellbeschreibungen für die Abbildung der Transportvorgänge verwendet. Dies liegt daran, dass die Integration von Intralogistikfahrzeugen mit komplexem Verhalten in ereignisdiskrete Simulationsumgebungen mehrere Einschränkungen aufweist.

Moderne kommerzielle Softwarelösungen ermöglichen die Modellierung und Simulation von Materialflüssen. Sie verfügen in der Regel über fertige Drag-and-Drop-Elemente zur Modellierung von Transportsystemen mit vordefinierten Eigenschaften. Für eine detaillierte und realitätsnahe Abbildung des Fahrzeugverhaltens (z.B. dynamische Bewegungssteuerung) sollen sie jedoch oft mit externen Algorithmen integriert werden. Da die Kommunikation über zusätzliche Schnittstellen und Protokolle erfolgt und nicht alles in einem integrierten Programm abläuft, arbeiten diese kommerziellen Werkzeuge langsamer als üblich. Auch in diesem Fall ist die Anpassbarkeit eingeschränkt, da kein Zugriff auf den Quellcode dieser Software möglich ist (Peyman et al. 2021).

Dieses Problem wurde zuvor auch von Peyman et al. (2021) identifiziert. Sie haben Python mit verschiedenen kommerziellen DES-Werkzeugen kombiniert, um ein realitätsgetreueres Fahrzeugverhalten zu erzielen. Aufgrund der eingeschränkten Funktionalität der Python-APIs konnten sie jedoch nicht die Flexibilität einer reinen Open-Source-Implementierung erreichen.

Hinzu kommen die Lizenzpreise für kommerzielle Simulationswerkzeuge. Sie sind sowohl für Unternehmen als auch für Forschungseinrichtungen mit begrenztem Budget nicht immer vernachlässigbar. Das beeinträchtigt ihre Wettbewerbsfähigkeit und verlangsamt den technischen Fortschritt in diesem Bereich.

Eine potentielle Lösung versprechen Open-Source Bibliotheken für DES. Neben der hohen Anpassbarkeit bieten sie Interoperabilität, d.h. Benutzer*innen sind nicht von einem einzigen Anbieter*innen abhängig. Die existierenden Open-Source Lösungen bieten jedoch meist keine fertigen Referenzmodelle für die Abbildung von Fahrzeugen und deren Verhalten an, welche direkt in ein Simulationsmodell eingebettet werden können. Sie müssen deswegen meist von Grund auf programmiert werden, was zeitaufwendig ist und gute Programmier- und Domänenkenntnisse erfordert.

Im Beitrag von Reith et al. (2021) wurde eine allgemeine Struktur des möglichen Open-Source Simulationsframeworks zur Modellierung der Transportprozesse dargestellt. Zudem gibt es in der Literatur bisher kein detailliertes Vorgehen für die Erstellung und Integration von Transportfahrzeugen in Open-Source DES-Simulationsumgebungen. Dieser Beitrag soll helfen, diese Lücke zu schließen.

2 Zugrundeliegende Annahmen

Um die komplexen Zusammenhänge, die bei der Bewegung von Fahrzeugen auftreten, zu vereinfachen, wurden in diesem Beitrag diese Annahmen bei der Entwicklung des Objektklasse und der nachfolgenden Berechnungsmethoden getroffen:

1. Die Fahrzeuggeometrie wird zu einer rechteckigen Form vereinfacht
2. Brems- und Beschleunigungswerte der Fahrzeuge werden zunächst nicht betrachtet, da diese Eigenschaften nicht direkt aus den technischen Unterlagen entnommen, sondern nur empirisch ermittelt werden können. Außerdem sind sie stark von anderen Faktoren wie Fahrzeugbeladung, Radzustand usw. abhängig.
3. Der Sicherheitsabstand ε wird eingeführt, um die vorgenommenen Vereinfachungen auszugleichen. Je höher ε ist, desto sicherer ist die generierte Trajektorie. Jedoch gehen mit zunehmenden ε womöglich Optimierungspotentiale für eine weg- oder zeitminimale Trajektorie abhanden.

3 Entwurf der Objektklasse für Transportressourcen

In Rahmen dieses Beitrags wurde eine Klasse erstellt, welche direkt in bestehende Strukturen der Open-Source DES-Bibliotheken eingebunden werden kann. Eine Klasse ist ein grundlegendes Konzept in objektorientierten Programmiersprachen und dient der Definition von Objekten. Sie besteht aus Attributen, welche die Eigenschaften oder Merkmale der Objekte repräsentieren, und Methoden, die das Verhalten der Objekte abbilden.

Zuerst wurden die Attribute festgelegt. Dies sind die Merkmale, die allen Fahrzeugen gemeinsam sind und mit denen sie dargestellt werden können. Dazu gehören sowohl physikalische (Abmessungen, Radkonstruktion usw.) und dynamische Merkmale (z. B. Höchstgeschwindigkeit) als auch Parameter, die für die Interaktion mit der DES-Umgebung und für die Bewegungssteuerung erforderlich sind. Die vollständige Liste der Attribute ist in Tabelle 1 enthalten.

Zusätzlich zu den Attributen sind bestimmte Methoden erforderlich, um die grundlegenden Funktionalitäten der Fahrzeugklasse sicherzustellen. Dazu gehören unter anderem die Überprüfung einer möglichen Kollision zwischen Fahrzeugen und die Auswahl von geeigneten Kollisionsvermeidungsmaßnahmen. Wie diese umgesetzt werden, wird in den folgenden Abschnitten erläutert. Die als grundsätzlich notwendig erachteten Methoden sind in Tabelle 1 aufgelistet. Diese Liste kann anders gegliedert und bei Bedarf um weitere Methoden ergänzt werden.

Von dieser können weitere Klassen abgeleitet werden, die von der Basisklasse „Transportfahrzeug“ erben und diese um weitere spezifische Attribute und Methoden ergänzen. So können verschiedene Fahrzeugtypen implementiert werden (z.B. Subklasse „Gabelstapler“ oder „AMR“). Die Elemente werden dann als Klassenobjekte in die Simulationsumgebung eingefügt.

Nach diesem Klasseentwurf können Fahrzeuge in Open-Source DES-Softwarelösungen in allen objektorientierten Programmiersprachen implementiert werden.

Tabelle 1: Entwurf der Objektklasse „Transportfahrzeug“

Klasse „Transportfahrzeug“	
Attribute (verändern sich nach der Initialisierung nicht):	
fahrzeug_id:	Eindeutiger Identifikator für das Fahrzeug
rad_typ:	Typ der Räder (definiert, welche Ausweichmanöver ausgeführt werden können)
l und b:	Abmessungen (Länge und Breite) des Fahrzeugs
max_geschw:	maximale Geschwindigkeit des Fahrzeugs
max_zuladung:	maximale Zuladung, die das Fahrzeug tragen kann
Attribute (verändern sich dynamisch nach der Initialisierung):	
geschw:	aktuelle Geschwindigkeit des Fahrzeugs
ladung:	Menge der Ladung, die das Fahrzeug derzeit trägt
position:	Aktuelle Position des Fahrzeugs in einem Koordinatensystem (z.B. (x, y))
richtung:	aktuelle Fahrtrichtung des Fahrzeugs in Grad (0 bis 359) oder Radiant (diese Information kann verwendet werden, um zu ermitteln, ob Fahrzeuge auf Kollisionskurs sind und um bei Bedarf Ausweichmanöver zu planen)
status:	aktueller Status des Fahrzeugs (z. B. frei, in Betrieb usw.).
koll_vermeiden:	boolescher Wert zur Anzeige des Kollisionsvermeidungszustands (ermöglicht Koordination mit Fahrzeugen oder Verkehrsmanagementsystemen)
traj:	Der zugewiesene vordefinierte Pfad, bestehend aus einer Liste von Punkten
Methoden:	
process():	Basismethode für die Interaktion des Fahrzeugs mit der DES-Simulationsumgebung
animate():	Visualisiert das Fahrzeug und dessen Trajektorie auf einer gegebenen Zeichenfläche
setze_traj ():	Legt eine neue Trajektorie für das Fahrzeug fest
kann_koll ():	Prüft, ob dieses Fahrzeug mit anderem in ereignisdiskreter Simulationsumgebung potenziell kollidieren kann
koll_maßnahme():	Methode zur Auswahl einer Kollisionsvermeidungsmaßnahme

4 Dynamische Bewegungssteuerung

Die Bewegungsplanung erfolgt in diesem Beitrag nach Reith et al. (2021) in zwei Phasen. In der ersten Phase werden die optimalen Routen für alle Fahrzeuge zentral ermittelt. Dies wird in der Regel durch A*- oder Dijkstra-Algorithmen realisiert. Da der Fokus dieser Arbeit auf der Abbildung der Transportressourcen und deren Verhalten in der Open Source Simulationssoftware DES liegt, wurde das Problem der Pfadfindung nicht betrachtet und dementsprechend diese Algorithmen nicht

implementiert. In der zweiten Phase werden die Bewegungen der einzelnen Agenten (Transportmittel) dezentral an die Umgebungsbedingungen angepasst.

Eine Fahrzeugbewegung ist ein zeitlich kontinuierlicher Prozess, der in DES-Modellen über endlich viele Ereignisse approximiert werden muss. Zu diesem Zweck werden künstliche Ereignisse in vordefinierten Zeitabständen eingeführt. Die Zeitabstände resultieren aus der Unterteilung des zurückzulegenden Wegs in mehrere Wegpunkte. An jedem Wegpunkt findet ein Ereignis statt, durch welches die Positions- und Bewegungsattribute der Fahrzeuge aktualisiert werden. Hierdurch kann überprüft werden, ob zum Zeitpunkt des Ereignisses neue Kollisions- und Deadlockgefährdungen auftreten. Wenn dies der Fall ist, werden vorbeugende Maßnahmen (z. B. Geschwindigkeitsanpassung, Ausweichmanöver) ergriffen.

4.1 Erkennen von Gefahrensituationen

Die frühzeitige Erkennung möglicher Kollisionen erfolgt in zwei Schritten. Zunächst wird festgestellt, ob sich die beiden Fahrzeuge in unmittelbarer Nähe zueinander befinden. Dazu werden Kreisflächen mit einem bestimmten Durchmesser um die Fahrzeuge gezogen. Der Durchmesser hängt von der Geschwindigkeit der Fahrzeuge ab (Je höher die Geschwindigkeit, desto größer der Durchmesser). Überschneiden sie sich, werden im Folgenden die Positionen (x_i; y_i) der Fahrzeuge im aktuellen (x_{iT}; y_{iT}) und im vorherigen ($x_{i(T-1)}$; $y_{i(T-1)}$) Ereignis herangezogen. Anhand dieser Informationen kann festgestellt werden, ob sich die Fahrzeuge einander nähern und wenn ja, welche Art von Kollision möglich ist: Frontal-, Heck- oder Seitenaufprall.

Dazu berechnet man zunächst die Winkel zwischen zwei Richtungsvektoren:

$$\theta = \cos^{-1} \frac{\overrightarrow{r_1} \cdot \overrightarrow{r_2}}{|\overrightarrow{r_1}| \cdot |\overrightarrow{r_2}|} \tag{1}$$

$\overrightarrow{r_i}$ bezeichnet den Richtungsvektor, der folgenderweise ermittelt wird:

$$\overrightarrow{r_i} = \{x_{iT} - x_{i(T-1)}; y_{iT} - y_{i(T-1)}\} \tag{2}$$

Anschließend wird ermittelt, wie sich der Abstand der Fahrzeuge zwischen zwei benachbarten Ereignissen verändert hat. Die Entfernung zwischen Fahrzeugen vor dem Zeitschritt in der DES-Umgebung wird wie folgt berechnet:

$$d_{T-1} = \sqrt{(x_{2(T-1)} - x_{1(T-1)})^2 + (y_{2(T-1)} - y_{1(T-1)})^2} \tag{3}$$

Analog wird die Entfernung nach dem Zeitschritt (d_T) bestimmt. Die möglichen Kollisionsarten und die Kriterien, nach denen sie klassifiziert werden, sind in Tabelle 2 aufgeführt.

Tabelle 2: Kriterien für die Kollisionsklassifizierung

Szenario	Kollisionskriterien
Frontalaufprall	$(\theta = 180°) \wedge (d_T < d_{T-1}) \wedge ((x_{1T} = x_{2T}) \wedge (y_{1T} = y_{2T}))$
Heckaufprall	$(\theta = 0°) \wedge (d_T < d_{T-1}) \wedge ((x_{1T} = x_{2T}) \vee (y_{1T} = y_{2T}))$
Seitenaufprall	$(0° < \theta < 180°) \wedge (d_T < d_{T-1})$

Durch diese Voranalyse kann frühzeitig erkannt werden, ob ein Kollisionsrisiko besteht, und wenn nicht, kann auf die Einführung zusätzlicher Ereignisse und fortgeschrittener Berechnungen verzichtet werden, was im Hinblick auf eine effiziente DES von Bedeutung ist.

4.2 Eingriffsentscheidung

Wenn im ersten Schritt festgestellt wurde, dass ein Kollisionsrisiko besteht, wird eine detaillierte Analyse durchgeführt. Diese basiert auf der erweiterten Time-to-Collision (TTC)-Berechnungsmethode nach Jiménez et al. (2013) unter Berücksichtigung der in diesem Beitrag getroffenen Annahmen. TTC ist die Zeit, die vergeht, bevor es zu einem Zusammenstoß zwischen den beteiligten Fahrzeugen kommt, sofern sich ihre Geschwindigkeiten nicht ändern und sie ihre Fahrwege beibehalten.

Auf Grundlage der Berechnungsergebnisse wird entschieden, ob Eingriffe zur Kollisionsvermeidung erforderlich sind. Dabei werden die Abmessungen, die Geschwindigkeit und die Ausrichtung der Fahrzeuge in Betracht gezogen. Abhängig von der Art der Kollision gelten unterschiedliche Berechnungsformeln.

Für das Szenario mit möglichen Heckaufprall (siehe Abbildung 1(a)) gilt die folgende Bedingung:

$$T < \frac{x_2 - x_1 - \frac{l_2}{2} - \frac{l_1}{2} - \varepsilon}{v_1 - v_2} \tag{4}$$

Der Parameter T ist dabei die Zeit bis zum nächsten Ereignis. Fahren die Fahrzeuge entlang Y-Achse, soll x in der Formel durch y ersetzt werden. Solange diese Bedingung erfüllt ist, besteht kein Handlungsbedarf. Andernfalls muss die Geschwindigkeit des auffahrenden Fahrzeugs reduziert werden, um eine Kollision zu vermeiden:

$$v_1 = v_2 \tag{5}$$

Wenn sich die Fahrzeuge in einem Winkel (θ) zueinander bewegen (siehe Abbildung 1 (b) und (d)) und folgende Bedingungen erfüllt ist, ist kein Eingreifen erforderlich:

$$\frac{(l_1 + d_{C1} + \varepsilon)\cos\alpha_1 + \frac{w_2}{2\sin\theta} + \frac{w_1 \cot\theta}{2}}{v_1} < \frac{(d_{C2} + \frac{l_2}{2})\cos\alpha_2 - \frac{\frac{w_1}{2} + \varepsilon}{\sin\theta}}{v_2} \tag{6}$$

In dieser Formel sind d_{Ci} die jeweiligen Abstände der beiden Fahrzeuge vom Schnittpunkt ihrer Trajektorien und α_i die jeweiligen Orientierungen bezüglich der X-Achse. Bei Nichterfüllung der Bedingung ist eine Reduzierung der Geschwindigkeit des Fahrzeugs erforderlich, das sich dem Kollisionspunkt als Letztes nähert. Die neue Geschwindigkeit ist der Höchstwert, bei dem die Bedingung erfüllt ist.

Wenn die Gefahr eines Frontalzusammenstoßes zwischen Fahrzeugen besteht (siehe Abbildung 1 (c)), sind mehrere Szenarien zu betrachten. Im ersten Fall ändert eines der Fahrzeuge die Fahrtrichtung und es wird folgende Bedingung für die Entscheidungsfindung verwendet:

$$\frac{d_{A1}}{v_1} < \frac{d_{A2} - R + \frac{\pi R}{2} + \frac{l_2}{2} + \varepsilon}{v_2} \tag{7}$$

Dabei sind d_{A1} und d_{A2} die jeweiligen Abstände der beiden Fahrzeuge zum Abbiegepunkt, R ist der Abbiegeradius und Fahrzeug 2 wird abbiegen.

Ist diese Ungleichheit erfüllt, muss Fahrzeug, das geradeaus fährt, abbremsen. Die neue Geschwindigkeit entspricht der maximalen Geschwindigkeit, für die diese Bedingung noch erfüllt wird.

Im nächsten Szenario werden beide Fahrzeuge in eine Richtung abbiegen. Dabei wird folgende Ungleichung verwendet:

$$\frac{d_{A1} - R}{v_1} < \frac{d_{A2} - R + \frac{\pi R}{2} + \frac{l_2}{2} + \varepsilon}{v_2} \tag{8}$$

Tritt diese Bedingung ein, muss Fahrzeug, das später Abbiegepunkt erreicht, seine Geschwindigkeit verringern.

Bei der letzten Variante fahren die beiden Fahrzeuge weiter aufeinander zu, ohne die Richtung zu ändern. In diesem Fall besteht sofortiger Handlungsbedarf. Die Transportfahrzeuge können Ausweichmanöver durchführen. Es wird geprüft, ob während des Manövers Kollisionen mit statischen Objekten in der Umgebung ausgeschlossen sind. Die Zeit, die das erste Fahrzeug für das Manöver benötigt, wird nach folgender Gleichung berechnet:

$$T_M = \frac{\pi(x_2 + \frac{w_1}{2} + \frac{w_2}{2} + \varepsilon)}{2v_1} \tag{9}$$

Fahren die Fahrzeuge entlang Y-Achse, soll x in der Gleichung durch y ersetzt werden.

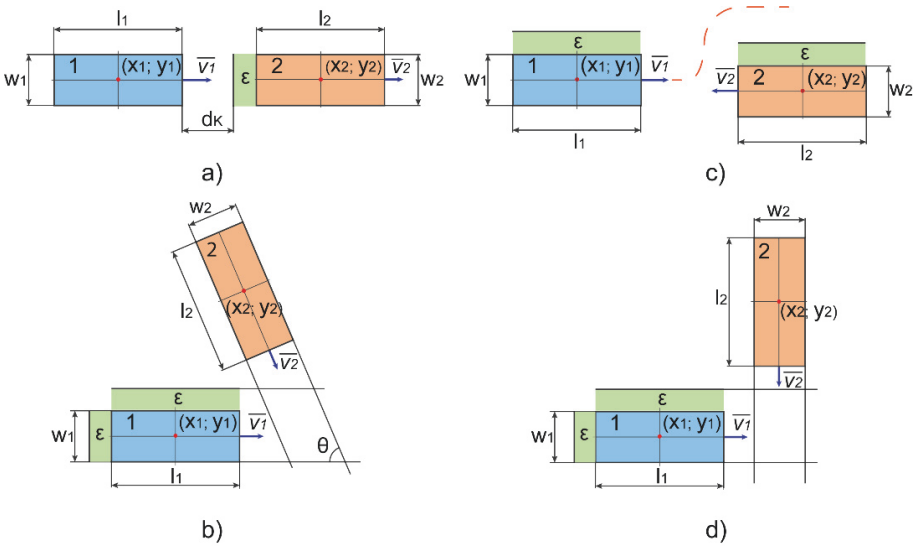

Abbildung 1: Kollisionsszenarien

In allen Szenarien wird der Zeitverlust durch Kollisionsvermeidungsmaßnahmen berechnet. Die dabei reduzierte Geschwindigkeit wird beim nächsten Ereignis, wenn

keine Gefahr mehr besteht, zurückgesetzt. Auf diese Weise werden Kollisionen dynamisch vermieden und dieses realitätsnahe Bewegungsverhalten der Fahrzeuge bei den klassischen Aufgaben des Ressourcenmanagements in DES berücksichtigt.

5 Implementierung

Python wurde als Programmiersprache für diesen Beitrag gewählt, da es sowohl in der Wissenschaft als auch in der Industrie weit verbreitet ist (Tambad et al. 2020). Des Weiteren ermöglicht Python, Simulationsframeworks mit z.B. KI- und anderen Optimierungsalgorithmen nahtlos zu kombinieren (Lang et al. 2021). In Python gibt es zwei populäre Simulationsframeworks für DES und zwar SimPy (Matloff, 2008) und Salabim (van der Ham, 2018). SimPy und Salabim sind in ähnlicher Weise aufgebaut. Alle Komponenten sind als Python-Klassen implementiert. Die Interaktion zwischen den Komponenten und der Simulationssteuerung erfolgt über Prozessmethoden, die als Generatorfunktionen implementiert sind. In dieser Arbeit wird Salabim verwendet, da es über eine integrierte Animationsengine verfügt. Dies vereinfacht die Visualisierung von Simulationen und die Implementierung von Layouts erheblich.

Die Funktionalität der erstellten Komponente wurde in einer Simulationsumgebung mit manuell definierten Hindernissen und Trajektorien getestet (Abbildung 2). Die Experimente wurden mit zwei Fahrzeugen durchgeführt, die mit ihrer jeweiligen Höchstgeschwindigkeit auf vordefinierten Routen fahren. Für die Simulationsumgebung wurde ein Maßstab (1 m = 100 px) angenommen. Für jedes Kollisionsszenario wurden insgesamt 100 Simulationen mit 4 verschiedenen Werten (jeweils 20 Simulationen) für den Sicherheitsabstand ($\varepsilon = \{0{,}1;\ 0{,}2;\ 0{,}3;\ 4\}$ m) durchgeführt. Alle Fahrzeugabmessungen wurden während der Versuche variiert ($w_i \pm 0{,}5$ m und $l_i \pm 0{,}5$ m) und jedes Mal zufällig zugewiesen. Unter diesen Bedingungen konnten die Kollisionen mit den vorgeschlagenen Berechnungsmethoden in allen simulierten Fällen erfolgreich vermieden werden. Unter "Kollision" wurde verstanden, dass die um die Sicherheitsabstände erweiterten Begrenzungsrahmen von Fahrzeugen überlappen. Das Video mit den Simulationen ist online unter dem Link https://youtu.be/YfH5DGkaVzA verfügbar.

6 Zusammenfassung

Die in diesem Beitrag entwickelte Objektklasse ermöglicht das Hinzufügen von frei beweglichen Transportressourcen zu Open-Source DES-Bibliotheken. Es unterstützt ebenfalls die Modellierung kollisionsfreier Bewegungen. Das vorgeschlagene Klassendesign ist in allen objektorientierten Programmiersprachen anwendbar. Dadurch wird die Funktionalität bestehender Open-Source- und kommerzieller Softwarelösungen erweitert und der Zugang zu komplexen Materialflusssimulationen für KMUs und Forschungseinrichtungen verbessert. Obwohl der Fokus dieser Arbeit auf Intralogistikfahrzeugen liegt, sind die Ergebnisse auch auf andere Fahrzeuge übertragbar.

Basierend auf dem vorgeschlagenen Klassendesign wurde eine Komponente für ein intralogistisches Transportfahrzeug für die Python-Bibliothek Salabim entwickelt. Die Funktionalität der generierten Komponente wurde in der Simulationsumgebung mit zwei Fahrzeugen unter den oben genannten Bedingungen erfolgreich verifiziert.

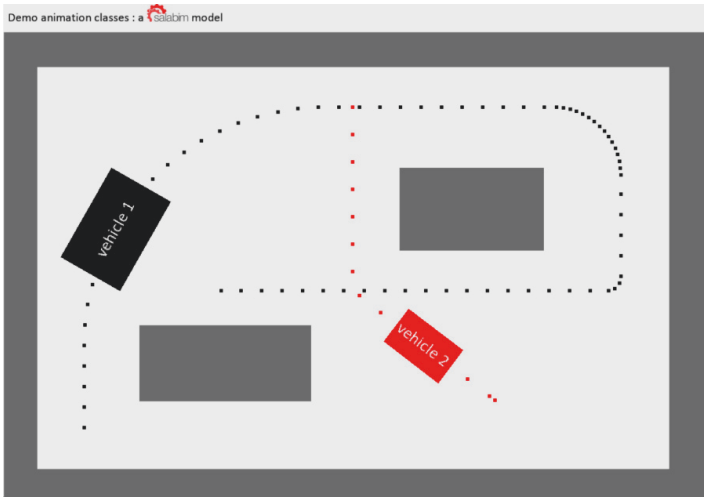

Abbildung 2: Simulationsumgebung

Der Programmieraufwand für die Verwendung der entwickelten Objektklasse ist geringer als bei einer Neuentwicklung. In der Regel bedarf es nur einer Anpassung der Attributwerte.

Es wurden auch Vergleiche mit der kommerziellen Softwarelösung Plant Simulation durchgeführt. In Plant Simulation gibt es beispielsweise keine vorprogrammierten Methoden zur Implementierung der Kollisionsvermeidung. Daher sind Anwender*innen dazu angehalten, diese Funktion mittels manuell erstelltem Programmcode zu realisieren. Dies impliziert, dass die kommerzielle Software in dieser Hinsicht keine Reduzierung des Programmieraufwandes bewirkt. Darüber hinaus nutzt "Plant Simulation" eine proprietäre Programmiersprache, welche nicht denselben Verbreitungsgrad wie Python hat. Der Zeitaufwand könnte folglich sogar erhöht sein, da für Python aufgrund der umfangreichen Python-Community vorgefertigte Code-Segmente für eine Vielzahl von Anwendungsfällen zugänglich sind.

Eine zusätzliche Evaluation der Komponente in komplexeren Szenarien, involvierend mehrere Fahrzeuge sowie zufällig generierte Hindernisse, ist jedoch erforderlich. Darüber hinaus steht eine detaillierte Gegenüberstellung mit den Funktionen kommerzieller Softwarelösungen noch aus. Dieses Vorhaben ist für künftige Untersuchungen geplant. Vor diesem Hintergrund sollte der vorliegende Beitrag als eine initiale Phase innerhalb des umfassenden Forschungsprojekts betrachtet werden.

Literatur

Alves, G.; Jürgen Roßmann; Roland Wischnewski: A Discrete-Event-Simulation Approach For Logistic Systems With Real Time Resource Routing And Vr Integration. International Journal of Computer and Information Engineering (2009), S. 821–826.

Fransen, K.; van Eekelen, J.; Pogromsky, A.; Boon, M.; Adan, I.: A dynamic path planning approach for dense, large, grid-based automated guided vehicle systems. Computers & Operations Research 123 (2020), S. 105046.

Herazo-Padilla, N.; Montoya-Torres, J.R.; Munoz-Villamizar, A.; Isaza, S.N.; Polo, L.R.: Coupling ant colony optimization and discrete-event simulation to solve a stochastic location-routing problem. In: 2013 Winter Simulations Conference (WSC), Washington, DC, USA, 08.12.2013 - 11.12.2013, 2013, S. 3352–3362.

Jiménez, F.; Naranjo, J.E.; García, F.: An Improved Method to Calculate the Time-to-Collision of Two Vehicles. International Journal of Intelligent Transportation Systems Research 11 (2013) 1, S. 34–42.

Kousi, N.; Koukas, S.; Michalos, G.; Makris, S.: Scheduling of smart intra – factory material supply operations using mobile robots. International Journal of Production Research 57 (2019) 3, S. 801–814.

Lang, S.; Reggelin, T.; Müller, M.; Nahhas, A.: Open-source discrete-event simulation software for applications in production and logistics: An alternative to commercial tools? Procedia Computer Science 180 (2021), S. 978–987.

Matloff, N.: Introduction to discrete-event simulation and the simpy language. Davis, CA. Dept of Computer Science 2 (2008), S. 1–33.

Müller, M.; Reggelin, T.; Kutsenko, I.; Zadek, H.; Reyes-Rubiano, L.S.: Towards Deadlock Handling with Machine Learning in a Simulation-Based Learning Environment. In: 2022 Winter Simulation Conference (WSC), Singapore, 11.12.2022-14.12.2022, 2022, S. 1485–1496.

Peyman, M.; Copado, P.; Panadero, J.; Juan, A.A.; Dehghanimohammadabadi, M.: A Tutorial on how to Connect Python with Different Simulation Software to Develop Rich Simheuristics. In: 2021 Winter Simulation Conference (WSC), Phoenix, AZ, USA, 12.12.2021 - 15.12.2021, 2021, S. 1–12.

Reith, K.-B.; Rank, S.; Schmidt, T.: A modular, discrete-event simulation framework for modelling free ranging transportation vehicles in intralogistics. In: Franke, J.; Schuderer, P. (Hrsg.): Simulation in Produktion und Logistik 2021, 15.09.2021-17.09.2021, 2021, S. 113–122.

Sun, X.; Wu, C.-C.; Chen, L.-R.: An Automated Warehouse Sorting System for Small Manufacturing Enterprise Applying Discrete Event Simulation. In: 2018 2nd IEEE Advanced Information Management,Communicates,Electronic and Automation Control Conference (IMCEC), Xi'an, 25.05.2018 - 27.05.2018, 2018, S. 1597–1601.

Tako, A.A.; Robinson, S.: The application of discrete event simulation and system dynamics in the logistics and supply chain context. Decision Support Systems 52 (2012) 4, S. 802–815.

Tambad, S.; Nandwani, R.; McIntosh, S.: Analyzing programming languages by community characteristics on Github and StackOverflow. arXiv preprint arXiv:2006.01351, 2020, zuletzt geprüft am 18.02.2022.

van der Ham, R.: salabim: discrete event simulation and animation in Python. Journal of Open Source Software 3 (2018) 27, S. 767.