A UAV SIMULATION ENGINE SUPPORTING MULTIPLE DEPOTS AND EASY EXCHANGE OF PATH-PLANNING ALGORITHM FOR THE SMART CITY

Jan Gerald Huenges

Distributed Artificial Intelligence Laboratory
Technical University Berlin
Str. des 17. Juni 135
Berlin
jan.g.huenges@campus.tu-berlin.de

Santiago Masip Gimeno

Distributed Artificial Intelligence Laboratory
Technical University Berlin
Str. des 17. Juni 135
Berlin
dominik.schroeck@campus.tu-berlin.de

Aleksandra Pranskaityte

Distributed Artificial Intelligence Laboratory
Technical University Berlin
Str. des 17. Juni 135
Berlin
dominik.schroeck@campus.tu-berlin.de

Dominik Schroeck

Distributed Artificial Intelligence Laboratory
Technical University Berlin
Str. des 17. Juni 135
Berlin
dominik.schroeck@campus.tu-berlin.de

Christopher-Eyk Hrabia, M.Sc.

Distributed Artificial Intelligence Laboratory
Technical University Berlin
Str. des 17. Juni 135
christopher-eyk.hrabia@dai-labor.de

Prof. Dr.-Ing. habil. Dr. h.c. Sahin Albayrak

Distributed Artificial Intelligence Laboratory
Technical University Berlin
Str. des 17. Juni 135
Berlin
sahin.albayrak@dai-labor.de

ABSTRACT

Unmanned Aerial Vehicles (UAVs) are flexible, versatile, easily installed and are in hold of relatively small operating expenses. Because of these advantages, the number of different UAV deployment scenarios is growing rapidly. In this paper, we address UAV-based parcel delivery systems. Implementing and testing such delivery systems is a complex and challenging task. To facilitate validation in the early stage of development we build up a simulation engine. This simulation engine is based on a decentralized approach in controlling multiple UAV delivery in a 3D environment. The aim of the simulation engine is to test an evaluate the spatial distribution of depots of drones as well as the suitability of a path-planning algorithm for the delivery use-case.

Keywords: UAV simulation, agent-based modelling, Smart City, Swarm optimization

1 INTRODUCTION

Once developed for the military sector, Unmanned Aerial Vehicles (UAV) are gaining popularity in private as well as commercial sector. One often-mentioned use-case is the parcel delivery. A number of big companies such as Amazon have already announced plans to implement UAV-based home delivery systems (Stolaroff 2014). It looks like in the near future UAVs might deliver parcels to your doorstep. The logistics industry hopes for positive impacts on their costs and competitiveness. The deployment of such a delivery concept is quite complex. It requires efficient path finding algorithms and an inter-vehicle communication, which is very challenging (Bekmezci, Sahingoz, and Temel 2013). To allow early evaluation and reduce the number of required costly flight-testing, a simulation engine is required. This paper proposes a simulation engine for multiple UAV delivery scenario. It supports a number of depots and UAVs as well as simple change of path-planning algorithm. UAVs operate in a 3D environment with realistic building shapes. The engine does not simulate the behavior of the UAV in the aerial space itself, but allows the evaluation of spatial distribution of depots or the suitability of a path-planning algorithm. It was developed considering the use of swarm algorithms in the delivery use-case and allows the avoidance of a central ground control due to inter-drone communication and autonomous organization.

2 RELATED WORK

The literature on simulation engines for a UAV parcel delivery scenario is still relatively scarce. In general the researchers concentrate more on specific components or fight behavior rather than the whole delivery use case. Eric Johnson and Sebastien Fointaine developed a simulation engine for the behavior of specific hardware components of a UAV (Johnson and Fontaine 2001). Their research focuses mainly on low-cost UAVs with limited capacities. The simulation provides high benefit for minimal effort and enables simulations with reasonable costs, taking into account the limited resources of the UAVs without accessing the UAVs actual hardware. Lu and Geng (2011) simulated the flight dynamic behavior of a UAV using Matlab/Simulink focusing on the control law of UAVs (Lu and Geng 2011). They propose a Hardware-In-a-Loop (HIL) simulation based on a mathematical UAV model, that includes the characteristics of the UAV. The simulation shows very low deviations between the system and real flights and is therefore suitable to test and validate the control law designs of a UAV.
MultiUAV (Rasmussen, Mitchell, Schulz, Schumacher, and Chandler 2003), a very general simulation engine has been developed to simulate cooperative algorithms for finding targets. The software includes not only vehicle dynamics but also flight dynamics, that is based on cooperative control algorithms. The cooperative control of UAVs requires an implementation of a inter-vehicle communication which is also addressed in our research. Furthermore, an agent-based simulation engine implementing a surveillance scenario was introduced in 2005(Jang, Reddy, Tosic, Chen, and Agha 2005). The main goal of this research was to develop efficient coordination methods for large-scale multi-agent systems. The simulation shares some similar concepts with the simulation proposed in our paper. For example, in their scenario the UAVs try to find targets without prior knowledge of the targets' locations. The UAVs are implemented as autonomous agents that interact with each other via messages. Moreover, the notion of obstacles and bases is being used. The described sensor concept is also similar to the one used in this paper.

3 SCENARIO

Before embarking the software model, the delivery scenario shall be explained. As mentioned in the introduction, the UAVs operate in a 3D environment representing a city. Every UAV has an assigned depot, called base-station in the engine, from which it receives items to be delivered. Every base-station has a limited range in the area and contains only items with the destination within its range. UAVs are assigned to one specific base-station and only deliver items to its area. After receiving an item, the UAV starts moving to the item's target destination. It scans the area and stores information on all fields it has visited on its route. If a UAV meets another UAV, they exchange information on the already explored parts of the area. There is no central unit, coordinating the UAVs. The system is based on autonomous logic of the UAVs and the inter-drone message exchange. If a field on the route contains an obstacle, the UAV will deviate from its planed route and try to find a way around the obstacle. Obstacles can have different heights and UAVs are able to fly different heights excluding different slopes. After delivering the item, the UAVs fly back to the base-station to receive a new item. The limited battery power of a UAV was also taken into consideration in the simulation. If the battery reaches a certain predefined threshold, the UAV will target the next nearest base-station (not necessarily its home base) and recharge its battery.

4 DESCRIPTION OF THE MODEL

The simulation has been written in Python, using the MESA framework for agent-based modelling. Figure 4 represents the software architecture. There are three major components: the simulation model itself, implementing the simulation environment and logic, an analytics interface for evaluating a simulation's performance and the GUI realized using a web server and allowing for easy access with a standard web-browser. The UAV as part of the simulation model, shall be described in particular in section 4.3.
The simulation's aim is to support comparing of different decentralized path-planning algorithms by making them easily exchangeable. Predefined KPI's and an output method are used to support analyzing the performance of one's implementation. Section 4.4 discusses the way KPI's are defined.
Figure 4 shows the architecture of the simulation model. In general, the model contains of a UI component

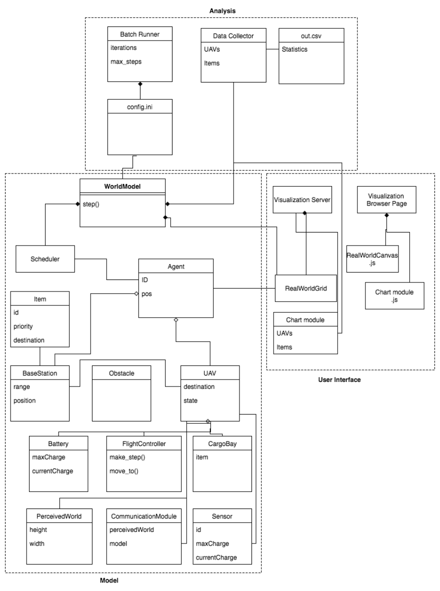


Figure 1: Architecture of simulation model

including configurationa and GUI, the model containing the UAV, the world model and the programming logic and the data analysis component which writes out the simulation results to allow for further evaluation.

4.1 MESA

MESA (Masad and Kazil 2015) is an open source agent-based modeling framework (ABM) developed specifically for the Python programming language. Agents are objects that have rules and states that act accordingly on every step of the simulation (Axtell 2000). ABM allows to capture the path of the simulation

along with the solution and allows for analysing the dynamic history (Axtell 2000). ABM also allows to dynamically pause and resume the simulation at any given step and analyse the current results.
MESA comes with implementations of important components, such as a grid for implementing a simple 2D environment, a web-browser-based UI, a data analysis tool and an agent scheduler. Our simulation makes use of some of these components while modifying others to suit our needs. We use the data collector, the scheduler and the agent base class without modifications. The scheduler activates the agents and triggers an action. We use the RandomActivation which means that in every step the agents are activated in a random order. The DataCollector is used for collecting relevant quantitative data to support the evaluation of an algorithm.
To better fit our needs, we modified the MESA grid, visualization elements and the visualization server. In the following section we describe what we changed and explain the reasons behind the modifications. The MESA grid is "a simple list-of lists" [1]. Each entry in such a grid has to be an agent. Since we wanted to perform our simulation on a larger scale, i.e. on a 500 x 500 grid, we had to place a lot of obstacles to come up with a realistic simulation environment. With such a huge amount of obstacles we experienced severe performance issues. The reasons behind this is the standard visualization element that is used to render the grid. The visualization element iterates over all registered agents and renders them in the GUI. As a result we decided to not consider obstacles as agents. Instead, we introduced our own grid, which is a matrix, that only contains numeric values. Each value represents either an obstacle, a base station or "nothing".

4.2 User Interface: Configuration and GUI

The user interface consists of two parts: The file *config.ini* to change the simulation scenario and a web server responsible for rendering the GUI and delivering it through a web browser.
The configuration file is being parsed at the startup of the simulation engine and contains all the required settings for the simulation to work properly. It contains crucial information that have a high impact on an algorithm's performance and makes it easy to set up different scenarios, e.g. one might consider testing the results using different number of UAVs per depot or more or less depots itself. Furthermore the impact of different battery capacity (in steps) can easily be simulated. A higher sensor range can be beneficial for an algorithm's performance but should be chosen realistically according to the model.
As mentioned earlier, the GUI is implemented using a web server. For the GUI to properly function, two image files are required. One contains the map as the user is supposed to see it. The other one stores information on obstacles and their respective heights using color codes. White means that there is no obstacle and other colors represent different altitude levels. An obstacle is any static object (such as trees or buildings etc.) that obstructs the movement of the UAV. The file will be parsed on start-up and the information stored in the model. Depots are placed on obstacles. Compare figure 4.2 for an overview of the GUI. The GUI allows for starting, pausing and resetting the simulation and shows the movement of UAVs in the space. It is implemented in a 2D fashion although internally the model calculates with 3-dimensional coordinates. The image's color codes for obstacles are used to represent the different altitudes. For now, the simulation tool supports 5 different height levels but one can easily modify the parser and define more colors for more altitude levels. To be able to see more detailed information on each agent (UAV, base station and item), one can click on the visual representation of the agent. The click activates a detail view. The detail view for UAV can be seen in figure 4.2.

[1]\urlttp://mesa.readthedocs.io/en/latest/apis/space.html

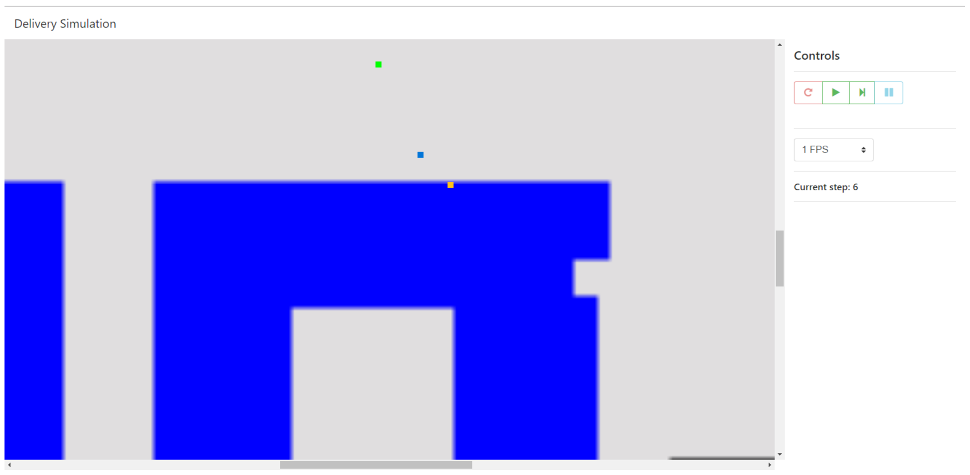
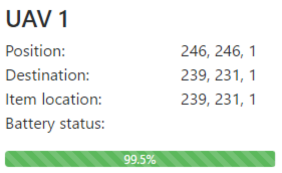


Figure 2: The GUI of the simulation engine



UAV 1

Position:	246, 246, 1
Destination:	239, 231, 1
Item location:	239, 231, 1
Battery status:	99.5%

Figure 3: Detailed view of UAV

4.3 Agents

The most important agent of a UAV simulation is the UAV itself. We developed a realistic simulation representing the required components and behavior and omitted not required components to cover the delivery scenario. The simulated UAV has the following components:

- Battery: The battery has a limited battery life that decreases on each step. When the battery reaches a certain threshold (configurable), the UAV will fly to the nearest depot to recharge.
- CargoBay: The cargoBay represents the UAV's storage unit which can contain exactly one item.
- FlightController: This component contains the path-finding/route-planning algoritm. It represents a navigation unit. Multiple algorithms could be implemented, compare section 5 for an example. The algorithm uses the sensor to identify its neighboring fields at any step. Additionally it stores a perceived world which holds all fields that are already explored. The component can access all information from the other component and the UAV itself. On every step, the UAV calls the method "make_step()" which has to be implemented in case of changing the algorithm.
- CommunicationModule: If the sensor found another UAV in the sensor's radius, this component will communicate with the other UAV and exchange grid information. Exchange of information on previously-unknown terrain helps when a UAV has to deliver an item to a new area of the map. This allows for precomputing optimal routes before exploring the actual area.
- Sensor: Scanning the grid for obstacles and other agents. This component is the interface to the model

UAVs do not communicate with a central entity such as a control server organizing the flight traffic. They communicate with each other if they are in sensor range and only rely on their own sensor. To optimize performance for the exchange of grids, the UAVs only store a dictionary containing their perceived world rather than a full matrix implemented in MESA's grid. There is one dictionary for each height level, mapping coordinates to information whether there is an obstacle on that field or not.
The second agent in the simulation is the BaseStation. In the real world, this is the depot of a logistics company. It assigns orders to UAVs and hands items over to them. Depots are responsible for all items in a part of the map and for a subset of all drones. Also, they contain chargers for UAVs whose battery reached a low level. In the configuration, one can set how many depots are supposed to be on the map by specifying the number of fields the depot is responsible for (its delivery area). Depots are located as close to the center of their area as possible, but only on obstacles (on buildings). Items that are picked up from a depot can have different priorities in order to make the simulation more realistic. For example, items with high priority can be important and immediately required medical equipment or parcels for which a customer paid a higher fee to receive it faster.

4.4 Analytics

During a run, data and results are being generated. To allow for the evaluation of a path finding algorithm For evaluating the delivery scenario, we identified the following KPI's:

- Average walk length: Describes the average number of steps taken for an item to be delivered.
- Average walk length divided by initial distance: This data point is a ratio calculated by dividing the average number of steps taken by the initial distance from base station to an item's destination. The initial distances are calculated as euclidean distances. Example: A value of 2 means that the average walk length is 2 times the initial distances.
- Standard deviation of average walk lengths: Represents the standard deviation of all walk lengths to get a better insight into the data additional to the average walk length.
- Items delivered per UAV: Items delivered divided by number of UAV's.
- Average lifetime of item: Calculation of the average lifetime of items by aggregating all item's lifetimes, divided by number of items. Lifetime is defined as the time from being available for delivery at the depot to the actual delivery at the item's destination.

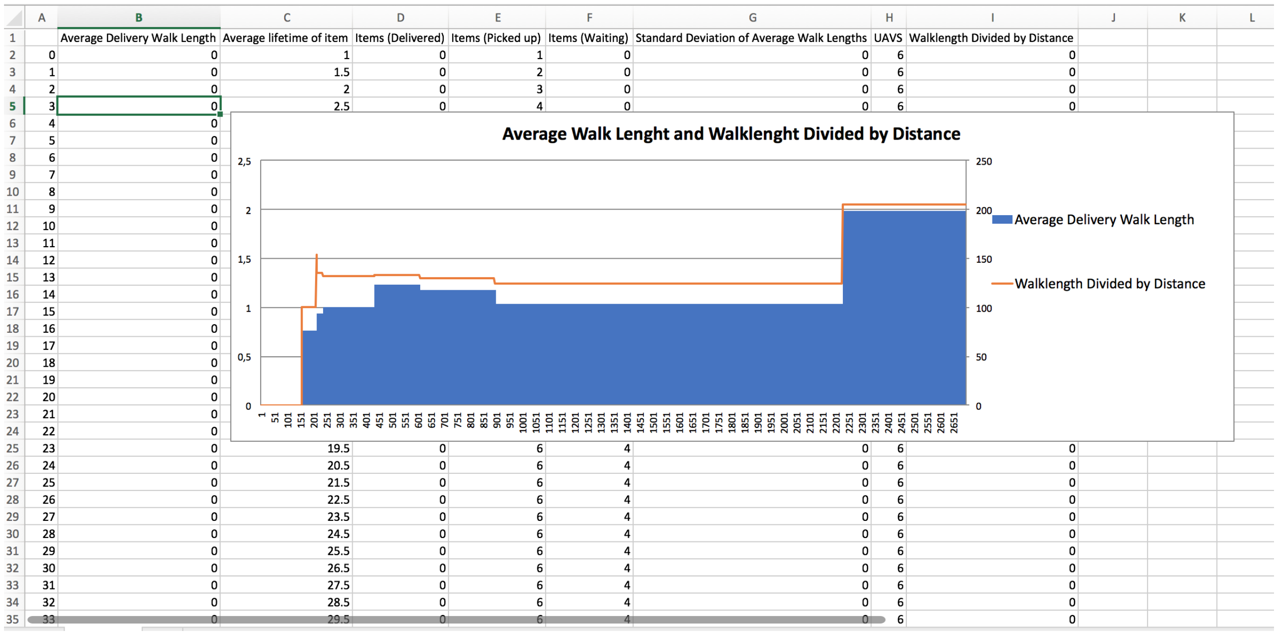
The results are step-wise written out into a CSV file during the simulation and can for example be imported into a spreadsheet software or more sophisticated tools like R or any other application to run statistical analysis of the KPI's. Technically, this is realized by using MESA's DataCollector class. This allows for easy definition of values to be calculated every step and export them to a CSV file. The KPI's can easily be extended or modified in the source code.

5 IMPLEMENTATION OF A SAMPLE ALGORITHM: A STAR

To provide an example simulation, the A* algorithm has been used SOURCES!

6 EVALUATION

The simulation engine presented in this paper allows for a simple evaluation of a path planning algorithm in a 3-dimensional space. Also it helps to implement traditional algorithms into a swarm optimization problem as UAVs exchange their grid whenever they come close enough to each other. A bottleneck is the calculation performance. On a mid-2015 Apple Macbook Pro with a i5 Dual core processor, the simulation



	A	B	C	D	E	F	G	H	I	J	K	L
1		Average Delivery Walk Length	Average lifetime of item	Items (Delivered)	Items (Picked up)	Items (Waiting)	Standard Deviation of Average Walk Lengths	UAVS	Walklength Divided by Distance			
2	0	0	1	0	1	0	0	6	0			
3	1	0	1.5	0	2	0	0	6	0			
4	2	0	2	0	3	0	0	6	0			
5	3	0	2.5	0	4	0	0	6	0			
6	4	0										
7	5	0										
8	6	0										
9	7	0										
10	8	0										
11	9	0										
12	10	0										
13	11	0										
14	12	0										
15	13	0										
16	14	0										
17	15	0										
18	16	0										
19	17	0										
20	18	0										
21	19	0										
22	20	0										
23	21	0										
24	22	0										
25	23	0	19.5	0	6	4	0	6	0			
26	24	0	20.5	0	6	4	0	6	0			
27	25	0	21.5	0	6	4	0	6	0			
28	26	0	22.5	0	6	4	0	6	0			
29	27	0	23.5	0	6	4	0	6	0			
30	28	0	24.5	0	6	4	0	6	0			
31	29	0	25.5	0	6	4	0	6	0			
32	30	0	26.5	0	6	4	0	6	0			
33	31	0	27.5	0	6	4	0	6	0			
34	32	0	28.5	0	6	4	0	6	0			
35	33	0	29.5	0	6	4	0	6	0			

Figure 4: Example of analyzing KPI's in a spreadsheet software

of 5000 steps (run without GUI) still takes a very long time. This is due to the fact that the implementation of the MESA grid used in the model is not performance-optimized. Also, the exchange of the grid is a complex task. It was tried to optimize performance by using dictionaries for the UAV's perceived world rather than storing a complete matrix. The model's internal grid also has been modified to just store numerical values for obstacles (1+ altitude), baseStations/depots (-1) or 0 for no content.

7 CONCLUSION

This paper presented a new simulation engine specifically developed for the delivery use-case with autonomous UAV not being controlled by a central control server. Delivery of items will be relevant in the near future and a simulation engine is required to test for different scenarios, such as different numbers of depots or rural against urban regions. The engine supports multiple depots and UAVs, communication between UAV and different altitude levels for testing. It is based on the MESA modeling framework whereas many adjustments for performance have been implemented. The GUI allows for visual evaluation of the simulation's progress and the CSV output helps analyzing the final scenario results. The simulation of a battery helps to understand how far a destination should be from a depot and thus how many depots are required.

The engine does not support UAVs carrying more than one item nor changing the input file for the map. Currently it comes with an example map but in the future this should be exchangeable. An example algorithm was implemented

REFERENCES

Axtell, R. 2000. "Why agents?: on the varied motivations for agent computing in the social sciences".

Bekmezci, I., O. K. Sahingoz, and Ş. Temel. 2013. "Flying ad-hoc networks (FANETs): A survey". *Ad Hoc Networks* vol. 11 (3), pp. 1254–1270.

Jang, M.-W., S. Reddy, P. Tosic, L. Chen, and G. Agha. 2005. "An actor-based simulation for studying UAV coordination". In *15th European Simulation Symposium*, pp. 323.

Johnson, E., and S. Fontaine. 2001. "Use of flight simulation to complement flight testing of low-cost UAVs". In *AIAA Modeling and Simulation Technologies Conference*.

Lu, P., and Q. Geng. 2011. "Real-time simulation system for UAV based on Matlab/Simulink". In *Computing, Control and Industrial Engineering (CCIE), 2011 IEEE 2nd International Conference on*, Volume 1, pp. 399–404. IEEE.

Masad, D., and J. Kazil. 2015. "Mesa: An Agent-Based Modeling Framework".

Rasmussen, S., J. Mitchell, C. Schulz, C. Schumacher, and P. Chandler. 2003. "A multiple UAV simulation for researchers". In *AIAA Modeling and Simulation Technologies Conference and Exhibit*, pp. 5684.

Stolaroff, J. 2014. "The Need for a Life Cycle Assessment of Drone-Based Commercial Package Delivery". *Livermore, CA: Lawrence Livermore National Laboratory*.

AUTHOR BIOGRAPHIES

THERESA M. K. ROEDER is an Associate Professor of Decision Sciences at San Francisco State University. She holds a PhD in Industrial Engineering and Operations Research from UC Berkeley. Her research interests lie in O.R. education and simulation modeling, especially in healthcare and higher education. Her email address is wsc15roeder@gmail.com.

PETER I. FRAZIER is an Associate Professor in the School of Operations Research and Information Engineering at Cornell University. He holds a Ph.D. in Operations Research and Financial Engineering from Princeton University. His research interests include optimal learning, sequential decision-making under uncertainty, and machine learning, focusing on applications in simulation optimization, e-commerce, and the physical sciences. His email address is wsc16frazier@gmail.com.

ROBERTO SZECHTMAN received his Ph.D. from Stanford University, and currently is Associate Professor in the Operations Research Department at the Naval Postgraduate School. His research interests include applied probability and military operations research. His email address is wsc16szechtman@gmail.com.

ENLU ZHOU is an Assistant Professor in the H. Milton Stewart School of Industrial & Systems Engineering at Georgia Institute of Technology. She holds a Ph.D. in Electrical Engineering from the University of Maryland, College Park. Her research interests include stochastic control and simulation optimization, with applications in financial engineering and revenue management. Her email address is wsc16zhou@gmail.com.