Swarm-Sim: A 2D & 3D Simulation Core for Swarm Agents

Ahmad Reza Cheraghi, Karol Actun, Sahdia Shahzad, Kalman Graffi
Technology of Social Networks, Heinrich Heine University Düsseldorf, Germany
Honda Research Institute Europe GmbH, Offenbach am Main, Germany
ahmad.cheraghi@hhu.de, karol.actun@hhu.de, sahdia.shahzad@hhu.de, kalman.graffi@honda-ri.de
https://tsn.hhu.de, https://www.honda-ri.de

***Abstract*—(Robot) swarm networks are consisting of magnitudes of individual agents that are capable to move in a world and interact with each other or with passive items. Through local, individual algorithms applied in the agents desired properties of the swarm can emerge. In this paper, we present swarm-sim, a round-based simulator that supports the evaluation of such large scale swarms in a 2D and 3D world. Agents can move in the simulated world, perceive their surrounding, carry other agents and items, as well as communicate with the position their stand on through pheromones or markings as well as with other agents and items in this world. Through this, a large number of swarm-related use cases are supported. Implementing swarm algorithms is easy as the simulator is completely written in python and its code is open-source. This allows also to simulate large swarms due to the good performance of the simulator. Thorough analysis of the swarm behavior is supported through plots and visualizations.**

I. Introduction

Swarm intelligence is the ability of a swarm of individual agents with limited information about their environment and restricted or even no communication among themselves to perform a given task. There are many examples in the nature of such intelligence: the flocking behavior of fishes and birds, the ability of ants to build their hills and find food sources or the adaptive behavior of microbial life to its environment.

These processes have been studied for many years and since the beginning of the information age people wanted to develop algorithms, which mimic and even improve these behaviors, so that they can be used to our advantage. A swarm of drones or robots, that scan and explore an entire cave without risking human lives, nanobots in human bodies as a new kind of targeted therapy or artificial bees for the pollination of our crops, are just few examples of the many possible use cases of artificial swarm intelligence.

The development of such algorithms is a difficult task, due to the fundamentally different approach than the classical computational algorithms. These algorithms are built up of a minimalistic set of rules, which specify the behavior of the swarm's individual agents on a given environmental situation, the agent is aware of, which is almost always just a small part of the whole picture. The other aspect is, that some of these algorithms rely on randomness of the environment or agent behavior, which makes the testing of them a difficult task. A Simulator is a solution for this problem.

In this paper, we present swarm-sim, a scalable swarm simulator. This simulator provides simple interface for programming the behaviors of individual agents, an environment for them to move through and allows to manage the randomness, such that a random scenarios can be easily reproduced. It is also equipped with a two- and three-dimensional visualizations, which show the operation of the algorithm. With a visualization it is easier to spot a mistake or possible improvements of the behavior of individual agents or the swarm in general, which makes it a very important tool in the development. Through the scalability of the simulator, even complex and large swarms behavior can be investigated.

The paper is structured as follows. First we give in Section II an overview on potential use cases for the simulator to shape out the vision. Then, in Section III we describe based on mathematical abstraction the requirements for an ideal round based simulator. Section IV introduces then swarm-sim's design and Section V specifies swarm-sim's architecture. The paper closes with an overview on related work in Section VI and conclusions in Section VII.

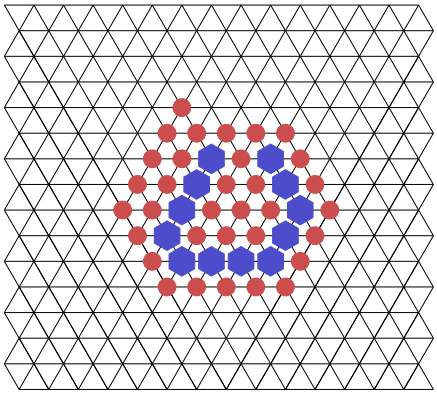
II. Use-Case Examples

Swarm-sim provides a core for implementation and testing new algorithm in any swarm agents research field. It allows to produce scenarios and solutions, real and nature inspired phenomena can be implemented and evaluated. In this section, we showcase some example in which we personally used swarm-sim to demonstrate its features and application range. For example, we used the simulator to evaluate the coating of arbitrary shaped objects [1]. We assume a given shape of passive objects, that is to be detected and optimally coated by a swarm of agents. An example for that is given in Figure 1a. Given are several agents that should coat an object that have any shape as shown in Fig. 1a. The aim of the agents is to reduce the distance of any agents to the object so that maximum distance of any agent to the object shape is smaller the minimum distance of any free location to the object shape. The agents compute their distances based on the first agent that hits the object and assumes its distance is one. It then shares its distance to the other agents in its neighborhood, which have then the distance of two (assuming that they themselves are not "touching" the object shape. Based on that, the agents can identify the agents with maximum distance as well as the distance of free position in their surrounding. Through communication they share they knowledge on smallest distances of free spots and the maximum distances of each agent, thus to learn whether and how to move to make space for those who have a higher distance until the condition is fulfilled.

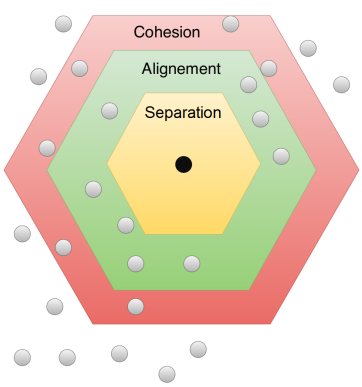
As a second example we use swarm-sim to investigate the flocking movement of swarm agents [2]. The agents are moving as a flock with each agent considering three zones in its surrounding (see Fig. 1b). First is the separation zone in that the agents are to near to each other and needs to increase their distance. Second is the alignment zone which optimal for the flocking movement. Last is the cohesion zone that is critical for the agents to leave, as then they are separated from the flock. We investigate in [2] the impact of these zone sizes.

Another example is the simulation of lawn mower swarm agents [3]. Here the aim of the agents is to mark/mow all the positions in any given terrain. In Fig. 1d, we can see the two red agents that already marked some of the terrain. The light blue marked locations are the cut grass, the dark blue ones are the uncut grass, non-marked locations are unknown locations for the agents. With swarm-sim, we investigate in [3] how the collaboration of several active agents introduces efficiency gains compared to non-collaborative agents.

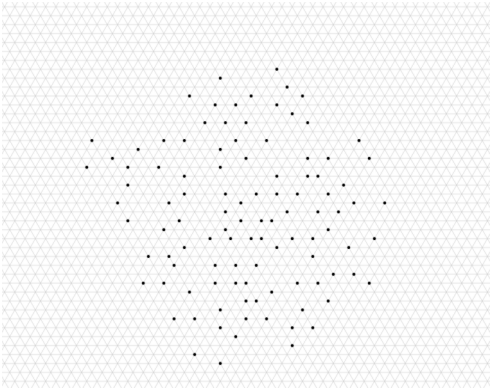
In [4] we use swarm-sim to evaluate the phototactic movement of agents in a swarm . In this example a light source is given and the agents do not know from where this light is coming from and the agents should move away from the light (Fig. 1e). We investigate approaches that allow the agents will move away from the light



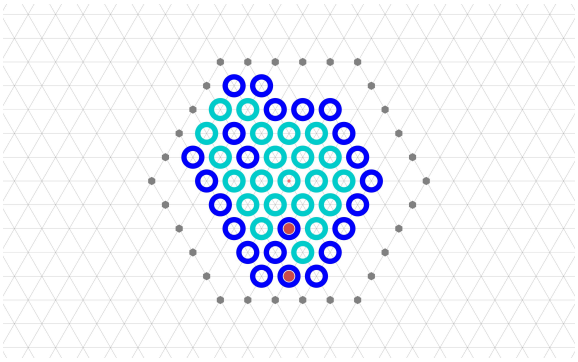
(a) Coating of an object [1]: Agents (red) aim to tightly pack around an object shape (blue)



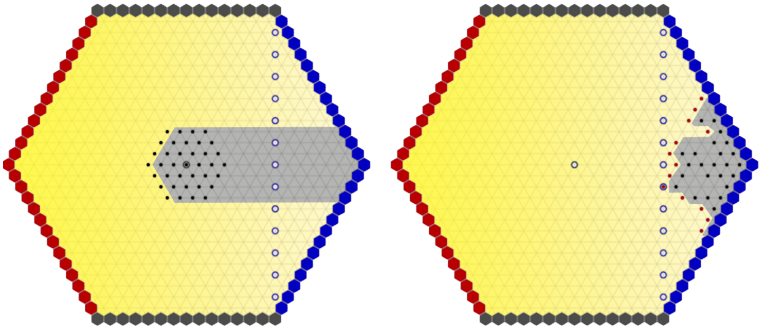
(b) Swarm flocking [2] using three zones - separation, alignment and cohesion



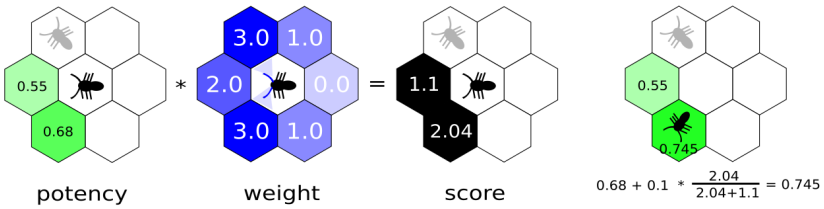
(c) Swarm flock [2] using a two hops wide separation ring



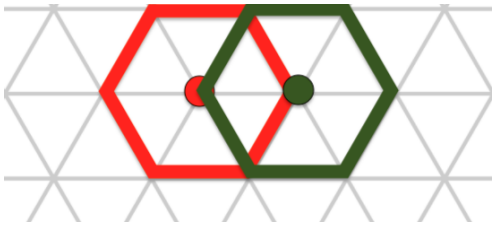
(d) Collaborative marking of an arbitrary terrain [3]: agents (red) aim to visit every marked location and turn it from unvisited (dark blue) to visited (light blue)



(e) Phototactic movement [4]: a swarm (of bugs) sensing light from the left and trying to move until they evade into darkness, i.e. cross the border to the right



(f) Ant swarms [5]: Ants sensing the pheromone potency in the surrounding fields, weighting it with their current direction to derive the (weighted) location score



(g) Opportunistic networking [6]: Two agents (red and green dot) in close proximity and their communication range (red and green hexagon respectively).

Fig. 1: Swarm-Sim use case examples with figures from our corresponding papers cited

through reacting on their perception of light as well as the availability of other agents in their near.

The problem of ant mills generated through sub-optimal pheromone setting, we explore in [5]. An example for this is presented in Fig. 1f. Finally, in [6] we explore opportunistic networking in the swarm. In this, as depicted in Fig. 1g, agents can send messages within a specific hop range, so that wireless multi-hop networking in mobile swarms can be simulated. Many other bio-inspired algorithms, such as mimicking the behavior of ants, bees or termites using pheromones as markers can be implemented. Thus, the uses cases are versatile and support a large range of swarm-related problem cases. Next, we introduce the underlying swarm model.

III. Mathematical Model of a Round based Simulator

In this section, the mathematical model of a round based simulator is introduced. Before this is going to happen, the necessary sets and functions of this model is described.

A. Time

In a round based simulator, time is handled as individual, ordered events, rather than a continuous dimension. At each point in time a set of actions and results is happening. Thus, time is a main factor for a round based simulator. Let T be a totally ordered set of points in time t.

$$T = (\mathbb{N}_0, <)$$

Time is a totally ordered set that is monotonically increasing. This is used to implement event-based simulators or round based simulators. In event-based simulators time steps are created and processed according to the simulation, e.g. $t_1 = 23.7s$, $t_1 = 35.2s$ and $t_1 = 96.1s$, and there is no activity in between. In round based simulators, actions are handled in blocks round for round, without the option to have activities in between. As a result, the next time step of a round is calculated by a monotonic function. NT gives the next time step function: $NT(t) = t+1$. Next, we focus on round-based simulations.

B. Matters

For simulating an algorithm, or anything, active and passive actors are essential. These actors are called **Matters**. Let M be a set of elements. Each $m \in M$ is a unique positive natural number that represents the ID of a *Matter*.

$$M = \{m \mid m \in \mathbb{N}_+\}$$

Active *Matters* have the capability of doing activities, e.g moving on their own and interacting with the other *Matters*, while passive *Matters* can be manipulated, but do not initiate actions themselves. Let $AM \subseteq M$ be the set of active *Matters* and let $PM \subseteq M$ be the set of passive *Matters*, then they form a partition of M: $AM \cup PM = M$ and $AM \cap PM = \varnothing$.

C. Locations

A simulation area is needed in which *Matters* are placed and interactions can happen. In our case, a physical two or three dimensional world is modeled, represented by an area in which discrete **Location** points exist. As any space can be linearized, we model a *Location* as a number. Let L be a set of *Locations* that build up together the simulation area.

$$L = \{l \mid l \in \mathbb{N}\}$$

Each *Matter* is mapped to a *Location*. The function is defined as $\lambda : M \to L$ and $\lambda(m) = l$.

D. Simulation World

The union of *Matters* and *Locations* is called simulation **World** W.

W^t be the set of the *Matters* and *Locations* at time $t \in T$ joined.

$$W^t = L^t \cup M^t$$

Thus, let W be the set of the *Matters* and *Locations* joined for all t's.

$$W = \{W^t \mid \forall t \in T\}$$

E. States

Each element of the simulation *World* has a state that can change while simulating, this is how interactions are modeled. Let ST_L be the set of possible states for a set of *Locations* L, ST_{PM} the set of possible states for a set of passive *Matters* PM and ST_{AM} the set of possible states for a set of active *Matters* AM, then let ST be the set of all possible states of *World* W.

$$ST = ST_L \times ST_{AM} \times ST_{PM}$$

We define ST^t as the vector of all the states of all *Matters* and *Locations* in a *World* W at time $t \in T$.

$$ST^t = ST_L^t \times ST_{AM}^t \times ST_{PM}^t, t \in T$$

As at each given time $t \in T$, we have a specific state for each *Location* and *Matter* in the *World* W (st^t), we can specify the state of the simulation at this time t as a tuple ST^t of the states of each *Location* and *Matter* in the *World* W:

$$ST^t = \{st^t \mid st \in ST, t \in T\}$$

F. Transition-State Function

A function is obligatory for changing the states of ST^t during each time step $t \to t+1$ to a new set of states ST^{t+1} and a new *World* W^{t+1} as the number of elements in W^t is subject to change. This function is

$$\delta : ST \to ST \times W \text{ and } \delta(ST^t) = (ST^{t+1}, W^{t+1})$$

G. Simulator Model

To simulate an algorithm, first a starting state has to be defined. Let ST^0 be the set of states when starting the simulation. Now that a starting state is defined, we also set a final goal. The simulator shall either terminate if a defined maximum time or a certain state is reached. Let $\hat{t}$ be the maximum round number that a simulation should run and FST be the set of final states. If one of the criteria is reached the simulator terminates. However, the simulator should give a result at each point of time t. Thus, the function $\mu : T \to ST$ and $\mu(t) = ST^t$ is defined. With all the previous defined elements a discrete simulator is a tuple as follows

$$(W, ST, T, \delta, \mu, \hat{t}, FST).$$

The simulator consists of the set of all *Worlds* W with the set of states ST and its set of monotonic increasing time T. It starts with a given state ST^0 and *World* W^0. The function $\delta(ST^t)$ is called for each time step $t \to t+1$ and returns a new state ST^{t+1} and its new *World* W^{t+1}. The function $\mu(t)$ gives the current state at time t. The simulation terminates when either the condition $t = \hat{t}$ or $ST^t \in FST$ is fulfilled.

IV. Swarm-Sim Mathematical Model

Now that some of the mathematical elements of a discrete simulator are described, in this section we elaborate swarm-sim with the use of those models. Some of the previous models are reused again, adapted for swarm-sim and some new ones are introduced.

A. Rounds

In swarm-sim the time T is modeled by the set of ordered rounds $\overline{R}$. Let R be a set of rounds. Each $r \in R$ is a natural number than can go to the maximum of defined round number called $\hat{r}$.

$$R = \{r \mid r \in \mathbb{N}_0, r \leq \hat{r}\}$$

Rounds is a totally ordered set of numbers which are monotonically increasing. Let $\overline{R}$ be the set of totally ordered rounds.

$$\overline{R} = (R, <)$$

B. Swarm-Sim Grid

Swarm-sim provides a 2D and a 3D Euclidean Space grid and each *Location* l is a vector with either two or three coordinates as follow:

$$l = \begin{cases} (x, y) \in \mathbb{Z} & \text{for} \quad \mathbb{R}^2 \\ (x, y, z) \in \mathbb{Z} & \text{for} \quad \mathbb{R}^3 \end{cases}$$

Eventually, the exact coordinates do not play a role as they are used mainly for visualization. Rather than the coordinates of a *Location*, the topology of the *Location*sis relevant.

C. Neighbor

Each *Location* is connected to adjacent *Locations*, termed **Neighbors**. Through this, the graph of *Locations* is spanning the whole *World*. Let N_l be the set of *Neighbors* for a *Location* $l \in L$. How many *Neighbors* a *Location* can have at maximum depends on the number of *Locations* that can fit around another and might further be limited by the border of the simulation *World*. Thus, $\hat{n_l}$ is the maximum amount of neighbors that a *Location* l can have.

$$N_l = \{n_j \mid n_j \in L \setminus l, 1 \leq j \leq \hat{n_l}\}$$

Also note that *Neighbors* are symmetric so $l_a \in N_l \Leftrightarrow l \in N_{l_a}$

D. Links and Directions

We say that two neighboring *Locations*, $l_a, l_b \in L$, are connected through a **Link** (l_a, l_b). The set of *Links* is given as

$$LI = \{(l_a, l_b) \mid l_a \in L \; and \; l_b \in N_{l_a}\}$$

In order to structure the set of Links relative to a specific *Location*, we number them and term them **Direction**:

$$DIR_l = \{dir | 1 \leq dir \leq \hat{n_l}\}$$

Let $\wp : LI \rightarrow DIR$ be a function that assigns a *Direction* $dir \in DIR$ to each *Link* from $l_a \in L$ to its adjacent *Location* $l_b \in N_{l_a}$ so that:

$$\{\wp(l_a, l_b) \mid l_b \in N_{l_a}\} = DIR_{l_a}$$

E. Paths, shortest Paths and Distance

Any two *Locations* in the *World* are connected over a chain of *Links* between two neighboring *Locations* forming a path. There are multiple possible paths between *Locations*, which is summarized in the set of **Paths**. The set of $Paths(l_a, l_b)$ between two *Locations* $l_a, l_b \in L$ is a set of *Links*, $\{(l_a, l_1), (l_1, l_2), (l_2, l_3), ..., (l_{p-1}, l_p), (l_p, l_b)\} \subseteq LI$, so that through several *Links* l_a and l_b are connected. Please, note that several possible paths exist. Typically the shortest paths are interesting. For that we define the set of **Shortest Paths** as

$$PathsMin : L \times L \rightarrow \mathcal{P}(LI)$$

$$p \in PathsMin(l_a, l_b) \subseteq Paths(l_a, l_b)$$

$$when \; p \in Paths(l_a, l_b) \wedge \nexists \; q \in Paths(l_a, l_b) : |q| < |p|$$

In this context, we also define the **Distance** between two *Locations*. Generally speaking, the *Distance* between two *Locations* is the length of a *shortest path* between the two *Locations* in the *Locations*' graph. Let $l_a, l_b \in L$ then the *Distance* $D : L \times L \rightarrow \mathbb{N}_+$ is defined as the minimum number of *Links* between the two *Locations*:

$$D(l_a, l_b) = |p| \; with \; p \in PathsMin(l_a, l_b)$$

The *Distance* between adjacent *Locations* in swarm-sim is 1 and in swarm-sim the distance unit is called **hop**.

F. Information and Memory

Each *Matter* and *Location* in swarm-sim has a memory in which information can be stored and shared. These memories are used by the active *Matters* to store state and to communicate with each other. Let MS be a set of numbers that represent memory slots.

$$MS = \{ms \mid ms \in \mathbb{N}_+\}$$

let I be a set of information.

$$I = \{i \mid i \in \Sigma^*\}$$

Σ^* is the set of possible finite strings over the alphabet Σ. A memory MM is a set of tuples of memory slots ms mapped to information i.

$$MM = \{(ms_j, i_j) \in MS \times I \mid 0 \leq j \leq |MS|\}$$

A memory can have the states either free $MMFree$ or full $MMFull$. It is full if its maximum capacity $\hat{MM}$ of possible information has been reached or free if not.

G. Locations

In swarm-sim *Matters* are associated with *Locations*. *Locations* define a topology and through that the action range of *Matters* in the *World*. *Locations* also offer a memory and can be used for writing and reading information, thus a *Location* in swarm-sim has a state.

Let ST_L be the set of states that all the *Locations* can have. First of all, each *Location* l has a set of neighboring *Locations* N_l as well as a memory MM_l and therefore a state of that memory. Additionally, *Locations* can be occupied by *Matter*, thus a counter MC_l for the number of *Matter* elements, that can stack, is introduced. *Locations* are considered $Free$ when there is no *Matter* on it or $Occupied$ if at least one *Matter* is on it.

$$ST_L(l) = \{N_l, MM_l, MC_l\}$$

H. Matters

For simulating an algorithm, or anything, subjects and objects are essential. These actors are called **Matters**. Let M be a set of elements. Each $m \in M$ is a unique positive natural number that represents the ID of a *Matter*.

$$M = \{m \mid m \in \mathbb{N}_+\}$$

I. Object: Item

In swarm-sim, we model objects to build obstacles or playing grounds called **Items**. A *Item* is the passive *Matter*; i.e. $T = t \mid t \in M$ in the swarm-sim *World*.

A *Item* can have different states. Let ST_T be the set of states that all the "*Items*" can have. The state consists of the *Item's* memory status MM_{item}, its *Location* $\lambda(item) \in L$ and whether it is immobile or currently carried by (at maximum) one active *Matter* $(TakenBy(item) \in \{\varnothing\} \cup AM)$.

$$ST_T(item) = \{MM_{item}, \lambda(item), TakenBy(item)\}$$

A transition-state in round t for a *Item* o that is not yet taken $(TakenBy(o) = \varnothing)$, located at the same*Location* $l = \lambda(o)$ as an active *Matter* $p \in AM$, would look like this when the *Item* is taken

$$TakenBy_t(o) = \{\varnothing\} \rightarrow TakenBy_{t+1}(o) = \{p\}$$

When placed down by that active *Matter* $p \in AM$ in round t:

$$TakenBy_t(o) = \{p\} \rightarrow TakenBy_{t+1}(o) = \{\varnothing\}$$

While being taken, a *Item* can change its *Location*, as it is carried around, as long as it is not placed down again. While placed down, it is immobile. In any case, its memory can be modified and read.

J. Subject: Agent

A *World* would not be alive without active *Matters* that can do actions on their own. Examples of *Matters* are robots, animals, software programs or even humans. Thus, the last *Matter* type in swarm-sim is called **agent** that represents active *Matters*, $P = p \mid p \in M$.

A *Agent* can have different states. Let ST_P be the set of states that all the *Agents* can have. A *Agent* $p \in P$, has a memory MM_p like *Items* and *Locations*, a *Location* $\lambda(p) \in L$ and a stack of *Items* $Taken(p) \subseteq P$ it carries.

$$ST_P(p) = \{MM_p, \lambda(p), Taken(p)\}$$

Please note that in most cases the number of *Items* to be carried at once is limited to 1.

State transitions for *Agents* occur, when they modify these entries, either through writing information to themselves or to adjacent

Agents, i.e. communication, through movement or by taking or dropping a *Item*.

Writing in memory:
Agents are the only elements capable of writing in their own, other *Agents*', *Items*' or *Locations*' memory. This information offloading often acts for the purpose of state manifestation, i.e. marking whether this *Matter*/*Location* has been seen before, or as means of communication. One *Agent* can leave a message at a *Location*, *Item* or even directly in another neighboring *Agent*, to inform other *Agents* about its actions or state, and thus to enable cooperative working on a task.

Movement:
One of the main abilities of a *Agent* p is to move at round t from its current *Location* l_a with $\lambda(p)_t = l_a$ to another adjacent *Location* $l_b = \lambda(p)_{t+1}$ for a valid *Link* (l_a, l_b). In swarm-sim, the *Agent* moves by specifying a *Direction* to move to relative to its position. One *Agent* move is denoted by the term *step* in the simulator.

Taking and dropping *Agents* or *Items*:
When a *Agent* p stands on a *Location* that is occupied either by one ore more *Agents* or *Items* t_i it can pick them up. For easiness a *Agent* or *Item* that can be taken or dropped is defined as an *Matter*. A *Agent* can also pick up *Matter* from adjacent *Locations*. Each *Agent* p maintains a list of items $Taken_t(p)$ it carries (at a time t). When an empty handed *Agent* picks up a matter m:

$$Taken_t(p) = \{\varnothing\} \rightarrow Taken_{t+1}(p) = \{m\}$$

When this *Agent* picks up a further item m:

$$Taken_t(p) \rightarrow Taken_{t+1}(p) = Taken_t(p) \cup \{m\}$$

When p places down an item $m \in M$ in round t:

$$Taken_t(p) \rightarrow Taken_{t+1}(p) = Taken_t(p) \setminus \{m\}$$

In most practical cases the items are set to not stack, however, the feature of stacking is supported. Also it is often assumed, that two *Items* cannot occupy the same *Location* and that *Agents* can only carry one *Item* at a time.

Scanning for *Matters* and written *Locations*: In order to gain motivation for all these transitions, an active *Matter* must have the ability to sense its surroundings, to be able to reason on it and to decide on appropriate reactions. An *Agent* can only sense its world up to a specific range, denoted in a number of *hops* h. Within this range, a *Agent* can sense the presence (i.e. state) of active and passive *Matters* as well as the state of the *Locations*. This means that every round a *Agent* gets the information of available *Locations*, *Agents* and *Items* in its surroundings (up to a distance of h *hops*). The *Location*, memory and further state information, such as Taken or TakenBy information, is made available to the *Agent*. In terms of *Locations*, only the *Locations* with non-empty memory are of interest. Based on this information, the *Agent* can decide which actions to initiate.

K. States

Each element of the simulation *World* has a state that can change while simulating, this is how interactions are modeled. Let ST_L be the set of possible states for a set of *Locations* L, ST_L the set of possible states for a set of passive *Matters* P and ST_P the set of possible states for a set of active *Matters* AM, then let ST be the set of all possible states of *World* W.

$$ST = ST_L \times ST_O \times ST_O$$

We define ST^t as the vector of all the states of all *Matters* and *Locations* in a *World* W at time $t \in T$.

$$ST^t = ST_L^t \times ST_P^t \times ST_P^t, t \in T$$

As at each given time $t \in T$, we have a specific state for each *Location* and *Matter* in the *World* W (st^t), we can specify the state of the simulation at this time t as a tuple ST^t of the states of each *Location* and *Matter* in the *World* W:

$$ST^t = \{st^t \mid st \in ST, t \in T\}$$

L. Swarm-Sim World

The union of *Matters* and *Locations* is called simulation **World** W.

W^t be the set of the *Matters* and *Locations* at time $t \in T$ joined.

$$W^t = L^t \cup M^t$$

Thus, let W be the set of the *Matters* and *Locations* joined for all t's.

$$W = \{W^t \mid \forall t \in T\}$$

$$SW = (W, ST)$$

The swarm-sim-World SW consist of all the *Locations*, *Agents* as active *Matters* and *Items* as passive *Matters*. The states of the *Locations* include their *Links* that span the *Location* graph and thus the *World*. The state of an active or passive *Matter* describes its status.

M. Swarm-Sim Scenario

A starting set of all the states at the beginning of the simulation, $round = 0$, is termed ST^0 and describes the **Scenario**. The *Scenario* describes a preset *World* W^0, the number and *Locations* of the *Matters* and their initial memory as well as the initial memory of the *Locations*. Typically, all memory is empty and the *Matters* are placed in a way that a given use case can be supported.

$$SCE = \{(W^0, ST^0) \mid ST^0 \in ST, W^0 \in W\}$$

N. Swarm-Sim Solution

A **Solution** SOL in swarm-sim describes a transition-state function that denotes how the agents react to their surroundings. This *Solution* is called once during each round and typically executes the same set of actions for each *Agent*, which require the *Agent* to sense its surroundings, consider its current algorithm state and induce a reaction in terms of movement, communication (through deploying a message in the memory of nearby other *Agents*), *Item* manipulation or interaction with the *Location* it is standing on.

O. Swarm-Sim Termination

The simulation terminates, when either the maximum round number $\hat{t}$ is reached or use-case specific conditions are met. Such a condition could be that the *Agents* found and marked all *Items* in the *World*. This, the simulation goal FST, is a subset of all potential states the simulation can take:

$$FST \subset ST$$

P. Swarm-Sim Model

The simulator swarm-sim has thus several components and is thus a tuple. First, it consists of the Swarm-Sim-World SW that has all the *Locations*, *Matters* and states (including the *Location* grid). It also contains a starting *Scenario* SCE that defines all the states of the *Locations* and *Matters* at the beginning of the simulation and the maximum round count $\hat{r}$, which acts as the upper limit for the rounds. A valid set of final states FST describes a goal for the simulation that ideally should be reached by the implemented *Solution*. A *Solution*

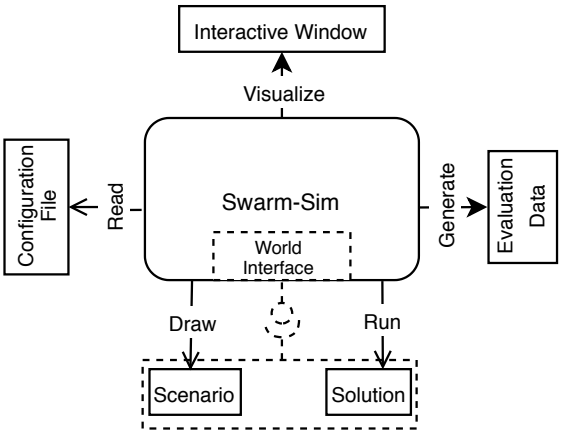


Fig. 2: Swarm-Sim General Architecture

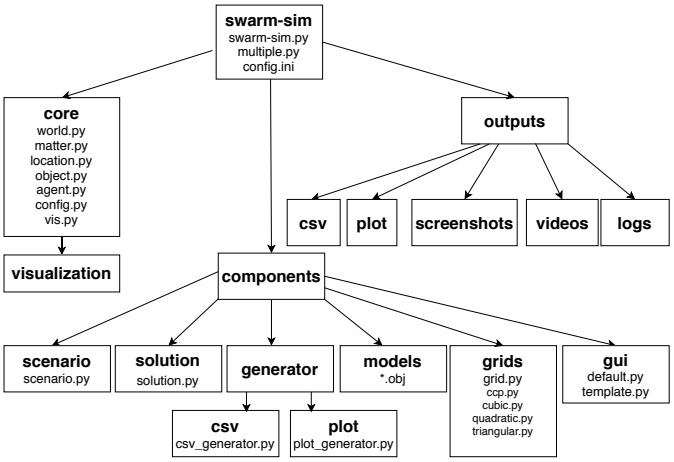


Fig. 3: Swarm-Sim Overview

SOL describes the transitions and interactions for the agents during each round:

$$(SW, SCE, SOL, \hat{r}, FST)$$

These swarm algorithms specify the behavior of the agents and implement the solutions for the given problem statements.

V. Swarm-Sim Architecture

As we mentioned before swarm-sim consist of the models World, Scenario, Solution, maximal rounds, and the finale state. Now, we describe how these models are developed and implemented. A general architecture overview of swarm-sim is shown in Figure 2. This swarm-sim core take care of reading the scenario to draw the first state at round 0, reads the necessary configuration data, runs the solution for $\hat{n}$ rounds and gives finally generate evaluation date. Nerveless, it also provides the world interface for the scenario and solution and makes swarm-sim to a user-friendly simulator. A deeper overview of swarm-sim is shown in figure 3. This we describe in the next section.

Swarm-sim is written in python 3.6. The structure of the folders is shown in Figure 3 and the code can be examined and downloaded from [7]. The main folder is the swarm-sim folder and within this folder three other necessary folders are included. In each folder the necessary files with their endings, e.g *.py, are written . In this section we describe each folders and files that are necessary to run and understand swarm-sim.

A. Swarm-Sim Elements

The swarm-sim folder is the main folder. It includes the running swarm-sim.py and multiple.py as well the config.ini and README.MD. First, in the README.MD is written how to install the necessary python libraries to run swarm-sim. Next, the config.ini is the configuration file. Users can change different parameters outside the actual code via this file. The different parameters are e.g. the scenario or solution. As well the maximum round number. Additional, if it should be in 2D or 3D. They are many parameters and allows to customize swarm-sim. Third, to run swarm-sim the python file swarm-sim.py has to be called. This can be done easy calling in terminal "python3.6 swarm-sim,py". The swarm-sim.py can be called with command line arguments. These can be e.g. a different seed value, scenario or solution. The command line arguments will overwrite the configuration value. Last, is the multiple.py. With multiple.py the user can run swarm-sim in different threads simultaneously to check e.g. how the finished state will be for different seeds. It has to be mentioned that the multiple can be run, like swarm-sim.py, from the terminal and it has command line arguments. However, it does not read the config.ini. This will be read from the called swarm-sim.py. Now that we know how to call install and run swarm-sim. It is important to know how to customize it. The next section "component" is about that.

B. Components

The component folder includes all the folders in that python codes can be built to customize swarm-sim. These folders and files we describe now.

1) Scenario: As mentioned earlier the scenario is necessary to build the states of swarm-sim at the beginning. In this folder a python file should be added and the name of the file should be inserted in the config.ini so swarm-sim can call it. An example of a simple scenario that adds with the interface world one agent, obstacle, and location is shown in 1.

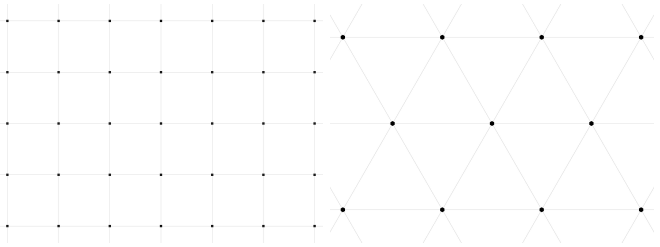
Listing 1: Example of a simple scenario

```
def scenario(world):
    world.add_agent((0.0,0.0,0.0))
    world.add_object((2.0,0.0,0.0))
    world.add_location((4.0,0.0,0.0))
```

2) Solution - Set of Swarm Algorithms: In the folder solution the use can add its own solution for running swarm-sim for a give scenario. In 2 we can see a simple solution. Again here we use the world interface that provides the agents list that were created with the scenario. Thus, the first loop is for running through the agents list. Next, the agent scans its neighborhood (i.e. the distance of hop = 1) for an object. If it found one, it will take it and move a step to a random direction and drop the object. If it does not find any it will move to any random choice direct. Like the scenario the solution python file can be customized named and should be inserted in the config.ini to run it.

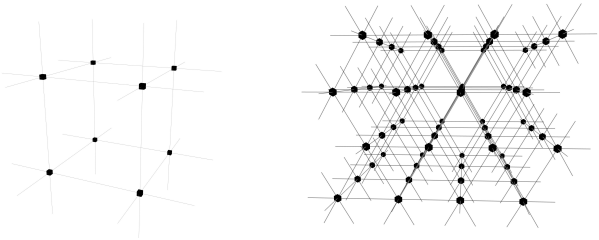
Listing 2: Example of a simple solution

```
def solution(world):
    for agent in world.agents:
        objects = agent.scan_for_object_in(hop=1)
        if len(objects) > 0:
            agent.take_object_with(objects[0].get_id())
        dire=random.choice(world.grid.get_directions_list())
        agent.move_to(dire)
        dire=random.choice(world.grid.get_directions_list())
        agent.drop_object_in(dire)
```



(a) Quadratic Coordinate (b) Triangular Coordinate

Fig. 4: Two-dimensional grids



(a) Cubic Coordinate (b) Cubic Close-Packing Grid

Fig. 5: Three-dimensional grids

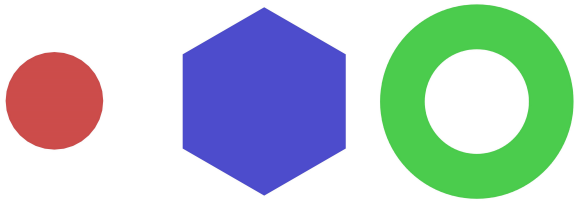
3) Generator: Hence, swarm-sim is for simulating, it is necessary to generate data. The generating of data is handled with a comma-separated value (CSV) format generator and a plot generator. Each of the generator is stored in the its own folder. Swarm-sim comes with a default generator for both formats. However, to let the user choose which value he likes to store and to plot, swarm-sim provides the possibility to customize each generator. Each customized generator must be stored in its own folder and be inserted in the config.ini so swarm-sim will use it while it is simulating.

4) Grids: In swarm-sim mathematical model we described that connection of the location via links are building a grid. Swarm-sim provided up to now four different grids. Two of them are in 2D and the two others in 3D. The visualization of the grids are shown in 4 and 5. However, swarm-sim is user friendly and can be customized. It has the abstract Grid class, which is the basis for all the possible coordinate systems. If a developer wand to insert a new grid, this class forces him to implement at least the necessary methods for the simulation and visualization.

5) Models: For the visualization of the agents, objects, and locations swarm-sim uses models with Wavefront File Format. The model is deconstructed into vertices, normal vectors, possible but not mandatory texture coordinates and the faces (triangles). The model is stored as plain text, so it is human-readable and the default file extension is '.obj'. In the configuration file model files can be assigned to simulator's objects. This allows fast modification of the visualization's appearance. Since the file is human-readable, it can be created manually, but it is not recommended, especially for larger objects. Instead it can be created using suitable software. There are multiple programs for creating Wavefront files, the most popular and free-to-use is blender [8]. As a default swarm-sim provides for the above-mentioned grids its own models for the agent, location, and object. Example for the triangular grid is shown in Figure 6.

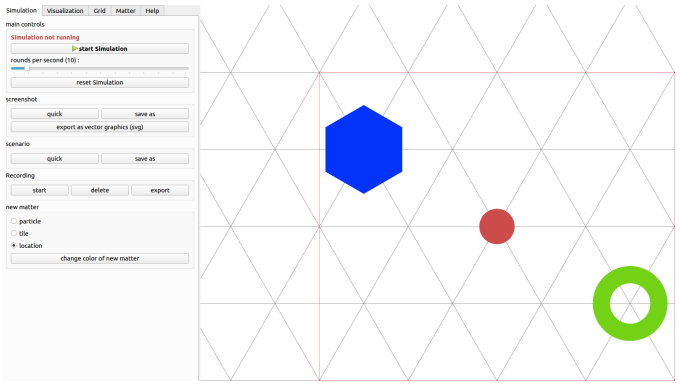
6) Graphical User Interface GUI: Swarm-sim provides a GUI shown in Figure 7, in that the user can start, pause and even reset the simulation, also add new matters and change their color.

In the GUI folder the default.py and template.py is located. The default.py is called by default. However, the user can build its own GUI in swarm-sim by using the template.py. The customized GUI can be inserted in the config.in to let swarm-sim to use it.

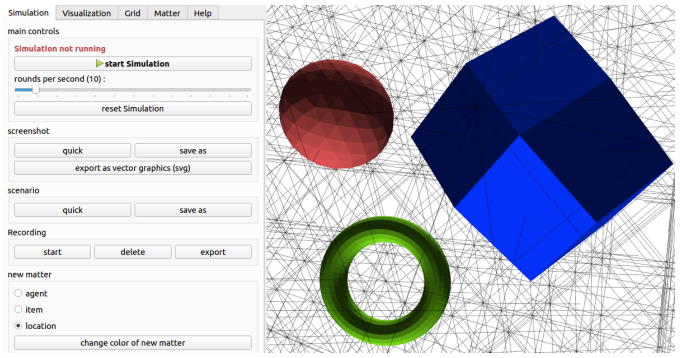


(a) Agent (b) Object (c) Location

Fig. 6: Visualization models of matters for a triangular grid



(a) 2D visualization of Swarm-Sim with agents (red), passive items (blue) and labeled locations (green) on a 2D hexagonal grid



(b) 3D visualization of Swarm-Sim

Fig. 7: Swarm-sim GUI

C. Core

The core folder includes the core python files that should not be customized and therefore touch. It includes the interface world.py, the parent class matters.py and its inheritance classes agent.py, location.py, and object.py. It also includes config.py and vis.py. The folder visualization is for the rendering and drawing of the models. This folder includes the necessary open-GL in general all the necessary files for producing the interactive window. We do not present this fold in detail, because it will be out of the scope and it is not necessary for knowing and running swarm-sim.

1) world.py: The world.py is the main API of swarm-sim. The reason why it is called world, because like a real world it has contains all the information about the scenario and the state. Hence, the world in that we live is basically a storage and can decide on its own what to created and to remove. The same ability is given to world.py and that is the reason why it is called so. Thus, world.py has many information and methods in order to build a scenario and solution for simulating swarms. The object of the world must always be included in the scenario and solutions methods. We describe briefly

some of the features of the world.py here and for more information you can read the code. World.py provides all the information about the created matters. E.g. world.agents is a list that includes all the objects of all the created agents in swarm-sim. Another example is world.agent_map_coordinates that is a python dictionary that maps the agent object with its actual location like the mathematical model $\lambda : M \rightarrow L$ and $\lambda(m) = l$. described earlier. Additionally, it gives information about the actual round number and it is possible to manipulate it. Plus, the world.py knows about the dimension of the grid, i.e. the size of the world and the amount of directions. It has many other methods and variables that can be seen in this file. So, world.py is one of the important objects that is first to set up the scenario and second to build up a solution.

2) matter.py: Matter is the parent class of the location, item, and agent. It includes the methods for read and write into the matter's memory and additional the color of the matters.

3) location.py: As mentioned earlier location is used in swarm-sim to mark a location to give it a state and to provide its being in the world.

4) item.py: Items are presenting the passive matters in swarm-sim. It can be taken or dropped by the agents. In this python file the class item is written and compared to its parent class matter it includes methods for changing the states of the item from taken to drop and vice versa.

5) agent.py: The agent.py is the most important class. It includes all the methods that are necessary for the agent actions. That are the taking and dropping of items. Creating or deleting items or locations. The moving method to any neighbor's location and many more. It also can read from and write in the memories of the neighbors. Additionally, it can scan for matters in n hops radius. All the capabilities of the agent are inside of the agent.py.

6) config.py: The config.py is for parsing the config.ini file. This file should only be edited if the user likes to add some more configuration options otherwise it should not be touched.

7) vis.py: Swarm-sim's visualization is implemented using the OpenGL API. This API provides an abstraction layer for hardware accelerated rendering, which substantially improves the time needed for the drawing of the scene. The graphical user interface is implemented using the PyQt5 library, which together with the OpenGL library make the whole visualization platform independent.

The visualization consists of a collection of modules, which allows for many customizations of the simulator's appearance. If the visualization is enabled, it controls the speed of the simulation and gives an overlook over the matter's memory and enables the interaction with the currently running solution, which helps to debug and inspect the developed algorithm.

The vis.py module contains the Visualization class, the main controller for the whole Visualization. This class is the connection between the Simulator, the OpenGL API and the GUI. Its main task is to ensure the correct execution time of the algorithm, whilst managing the rendering, data preparation and the GUI's event handling.

D. Outputs

Swarm-sim is an evaluation tool, that should output data for examination. Therefore, the output folder is used for all the possible outputs that swarm-sim can generate.

1) CSV: For the evaluation swarm-sim uses csv files. Three types of csv files are generated after each simulation. First, a csv file is created for the rounds in that for each round all the data is stored. Next, for the agents a csv file is generated to show for each individual agent what actions were done. Last, if the simulator were run with different seeds an aggregation happens over each seed and a all_aggregate csv file is generated, so to have a summary of all the simulation runs with the different seeds.

2) Plot: For the above csv files swarm-sim creates plot png images that are stored in the plot folder.

3) Screenshots: Swarm-sim provides the option to take screenshots while the simulator is running in the visualization mode. All the shots are stored in the screenshots folder.

4) Videos: Beside screenshots it also allows to record the simulation visualization. The recorded video can be stored either in mp4 or avi formats.

5) Logs: In the logs folder the system.log is stored. This log includes all the internal calls of swarm-sim.

With this range of outputs a thorough evaluation of the emerging swarm behavior can be conducted. Next, we discuss related work and show where their shortcomings are in addressing the initially sketched use cases.

VI. Related Work

Having discussed the usability and the details of swarm-sim, in this section we discuss related simulators. In 1986 Craig W. Reynolds created the *Boids* program [47] for simulating bird flocking behavior in three dimensions. He created a model, which mimics realistic movement of bird flocks with additional features like obstacle avoidance and goal seeking. The goal of this model was to create a realistic simulation of flocking behavior for the creation of videos and games.

In later years Craig W. Reynolds worked for the OpenSteer project [48], which is a open source framework for designing movement and steering behaviors for autonomous objects, mainly used in games and animations. This focus on games and animations sets it apart from swarm-sim, which aims to support the analysis of novel swarm algorithms. While a sophisticated game engine will be also able to simulate swarms, it requires expert knowledge. Swarm-sim in contrast aims to support easy access to scalable swarm simulations.

The Player Project is an open source software developed for the research of robotics and sensor systems. It provides a wide range of pre-defined controllers and sensors and a simple interface to communicate with the robots over the IP network. There are two simulators/visualizations build on top of the Player software. "Stage" is a 2D simulator optimized for the simulation of a large population of robots, but with the cost of low accuracy. Gazebo is a 3D simulator optimized for the simulation of smaller populations of robots, but with high accuracy, e.g. to simulate joints and actuators. These simulators focus on actual robot elements

Robot Virtual Worlds or RVW [49] is another project for developing and programming robotic behaviors, which is mainly targeted at students for teaching purposes. It provides an universal programming language (ROBOTC), which allows to design the behavior of virtual robots within an included 3D simulator as well as real world robot systems, provided by popular manufacturers like LEGO, VEX or TETRIX. The focus lies less on complex swarm algorithms and more on individual robot programming. Another very capable tool is the CoppeliaSim [50] project, formerly known as V-REP. Its designed for simulating high fidelity robotics systems, such as inverse kinematics in robot arms.

Simulating Collaborative Robots in Massive Multi-Agent Game Execution (SCRIMMAGE) [46], [51] is a 3-dimensional, open source simulation environment for the testing and comparing of mobile robotics algorithms and behaviors. SCRIMMAGE takes its inspiration from the properties that the stage project [14] offers, namely the simulation of multiple robots, the plugin interface for robotic control, models for locomotion, sensors, and the autonomous communication among robots. SCRIMMAGE not only simulates aerial robots but

Simulator	Objective	Developers	Open source	supported languages	Supported OS	2D or 3D	Status	Researches
Player and stage	offers free software for robots, sensors and actuators research	Brian Gerkey [9], Richard Vaughan, Andrew Howard, and Nathan Koenig	Yes	any language	Linux, Solaris, BSD and MacOSX	2D	Last update 2010	[10], [11], [12], [13], [14]
Gazebo	offers opportunity to simulate robotic swarms accurately and efficiently in various indoor and outdoor environments	Open Source Robotics Foundation (OSRF) [15]	Yes	mostly ROS (Robot Operating System)[16]	MacOS, Linux and Windows (a binary package available only for Linux)	3D	Active	[17], [18]
Robot Virtual Worlds	High-end simulation environment for students to learn programming	Robomatter Incorporated [19]	No	ROBOTC [20]	Windows and Mac	3D	Active	[21]
Teambots	Offers java classes and APIs to support research in mobile multi-agent systems	Georgia Tech's Mobile Robot Laboratory [22]	Yes	Java	Windows, NT, Solaris, SunOS, MacOS, OS X, Linux and IRIX	2D	Last update 2000	[23], [24], [25]
V-REP	A universal simulator with integrated development environment, where each object can be controlled individually	Coppelia Robotics [26]	Yes	C, Python, C++, Java, Lua, Matlab, Octave and Urbi	MacOSX, windows and linux,	3D	Active	[27], [28], [29], [30]
ARGoS	Aims to simulate heterogeneous robotic swarms in real-time	Developed within the swarmanoid project [31]	Yes	ASEBA scripting language (others are under study) [32]	Linux and MacOSX	2D and 3D	Active	[33], [34]
Webots	high-quality professional mobile robot simulator used for educational purposes	Cyberbotics Ltd. [35]	Yes	C, C++, Java and from third party software (via TCP/IP)	Windows, Linux and MacOSX	3D	Active	[36], [37], [38]
Workspace	An Offline Simulation and programming platform. Offers simulation solutions for industrial and educational purposes	Watson Automation Technical Solutions Ltd. [39]	No	Many robotic languages e.g. AB G-Code and Adept V-Plus	Windows	3D	Active	[40]
OpenHRP	A virtual platform to investigate humanoid robotics	AIST [41]	Yes	C, Python, C++, Java	Linux, Windows	3D	Last update 2012	[42], [43], [44], [45]
SCRIMMAGE	Used for the testing and comparing of mobile robotic algorithms and behaviors	Georgia Institute of Technology	Yes	Python, C++	Linux, MacOS	3D	Active	[46]
Swarm-Sim	A round based simulator developed for modeling swarm robotic systems in a 2D/3D environment.	Heinrich Heine University of Düsseldorf	Yes	Python	Linux, MacOS, and Windows	2D and 3D	Active	

TABLE I: Comparison and differences between several swarm robotic simulators.

also ground and water robots. SCRIMMAGE has a batch processing and it can analyze simulation results. Multiple plugins are provided by SCRIMMAGE, including autonomy, motion model, controller, sensor, entity interaction and metrics plugins. Swarm-sim addresses similar goals with a close focus on swarm algorithms that specifically aim to solve given problems.

The works of Gmyr and Scheideler [52] address similar questions as we do in the use cases. While our simulator resembles similar elements as swarm-sim, swarm-sim has been implemented as open source in the last years and is available online.

The WeBots project [53], just like CoppeliaSim, is a high-fidelity simulator of robotic systems. It provides the ability to develop and simulate algorithms for wheeled, legged and flying robots which can later be transferred to real mobile robots from various manufacturers. This focus on real mobile robots limits its capabilities and scalability. In swarm-sim, in contrast, we focus on the swarm algorithms and less on the potential implementation in real robots.

At last, the Workspace - robot simulation [54] is a program for simulating industrial machines. It offers an efficient and accurate simulation of various industrial machines and provides dynamic and static analysis of detected collisions. It does not support well the evaluation of mobile swarm algorithms.

Most of these simulators main interest lies in designing very realistic models and visualization for robots, which lowers the capabilities of simulating and visualizing large amounts of individual agents for simulations and algorithms not concerning robotics. The Boids, OpenSteer and Player/Stage projects are designed, such that they could simulate larger amounts of agents, but the Boids and OpenSteer projects are specifically implemented to simulate movement behaviors and not general swarm intelligence algorithms and the Player/Stage project offers only a 2D visualization and simulation. Furthermore all of these programs simulate their agents and objects in a continuous space, which can lead to small errors and inconsistencies as a result of floating-point rounding errors. Swarm-sim's quantized and modular space and its simple and performant visualization makes real time 3D simulations of large amounts agents possible. A comparison of all the mentioned simulators included our swarm-sim is shown in table I.

VII. Conclusion

Swarm-sim simulator is an round-based evaluation tool completely built with Python. It allows to simulate agents and items in a 2D and 3D world to explore various swarm algorithms. Agents can move in the simulated world, perceive their surrounding, carry other agents and items, as well as communicate with the position their stand on as well as with other agents and items. Through the thorough modeling given, the problem space that can be explore with this tool is wide and versatile. Implementing swarm algorithms is easy, flexible and allows for a rich evaluation using plots and visualizations. Swarm-sim is open-source [7] so a community can be built to continuously update it and make it day for day better.

Acknowledgments

We would like to thank the Arab-German Young Academy of Sciences and Humanities for funding this research as well as our collaboration partner Dr. Ahmed Khalil for the valuable discussions.

REFERENCES

[1] A. R. Cheraghi, G. Wunderlich, and K. Graffi, " Water Flow Coating of Arbitrary Objects Using Swarm Robots ," in *Proc. of Asia-Pacific Conference on Intelligent Robot Systems (ACIRS 2020)*, 2020.

[2] A. R. Cheraghi, A. Ben Janete, and K. Graffi, "Flocking on a 2-D Triangular Graph," in *Proc. of Asia-Pacific Conference on Intelligent Robot Systems (ACIRS 2020)*, 2020.

[3] A. R. Cheraghi, A. Abdelgalil, and K. Graffi, "Universal 2-Dimensional Arbitrarily Shaped Terrain Marking," in *Proc. of Asia-Pacific Conference on Intelligent Robot Systems (ACIRS 2020)*, 2020.

[4] A. R. Cheraghi, F. Schloesser Vila, and K. Graffi, " Phototactic Movement of Self-charging Swarms ," in *Proc. of Asia-Pacific Conference on Intelligent Robot Systems (ACIRS 2020)*, 2020.

[5] A. R. Cheraghi, J. Peters, and K. Graffi, "Prevention of Ant Mills in Pheromone-based Swarm Algorithms," in *Submitted to 3rd Int. Conf. of Intelligent Robotic and Control Engineering (IRCE'20)*, 2020.

[6] A. R. Cheraghi, J. Zenz, and K. Graffi, "Opportunistic Network Behavior: Passing Messages to Destination," in *Proc. of Asia-Pacific Conference on Intelligent Robot Systems (ACIRS 2020)*, 2020.

[7] A. R. Cheraghi. (2020) Swarm simulator. [Online]. Available: https://gitlab.cs.uni-duesseldorf.de/cheraghi/swarm-sim

[8] R. Hess, *The essential Blender: guide to 3D creation with the open source suite Blender*. No Starch Press, 2007.

[9] "Brian gerkey's website," https://brian.gerkey.org.

[10] R. T. Vaughan and B. P. Gerkey, "Reusable robot software and the player/stage project," in *Software Engineering for Experimental Robotics*. Springer, 2007, pp. 267–289.

[11] B. P. Gerkey, R. T. Vaughan, K. Stoy, A. Howard, G. S. Sukhatme, and M. J. Mataric, "Most valuable player: A robot device server for distributed control," in *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180)*, vol. 3. IEEE, 2001, pp. 1226–1231.

[12] T. H. Collett, B. A. MacDonald, and B. P. Gerkey, "Player 2.0: Toward a practical robot programming framework," in *Proceedings of the Australasian conference on robotics and automation (ACRA 2005)*, 2005, p. 145.

[13] B. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *Proc. of the Int. Conf. on Advanced Robotics*, vol. 1, 2003, pp. 317–323.

[14] R. Vaughan, "Massively multi-robot simulation in stage," *Swarm intelligence*, vol. 2, no. 2-4, pp. 189–208, 2008.

[15] "Open robotics," https://www.openrobotics.org.

[16] "Robot operating system," http://www.ros.org.

[17] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. Von Stryk, "Comprehensive Simulation of Quadrotor UAVS using ROS and Gazebo," in *Int. Conf. on Sim., Modeling, and Programming for Autonomous Robots*. Springer, 2012.

[18] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3. IEEE, 2004, pp. 2149–2154.

[19] "Robomatter incorporated," http://www.robomatter.com.

[20] "Robotc," http://www.robotc.net.

[21] "Robot virtual worlds," http://www.robotvirtualworlds.com.

[22] "Georgia Tech's Mobile Robot Laboratory," www.cc.gatech.edu/ai/robot-lab.

[23] T. Balch, "Teambots software and documentation," *Available through the World-Wide Web at http://www. teambots. org*, 2001.

[24] ——, "The teambots environment for multi-robot systems development," *Working notes of Tutorial on Mobile Robot Programming Paradigms, ICRA*, 2002.

[25] ——, "Behavioral diversity in learning robot teams," Georgia Institute of Technology, Tech. Rep., 1998.

[26] "V-rep," http://www.coppeliarobotics.com.

[27] M. Freese, S. Singh, F. Ozaki, and N. Matsuhira, "Virtual robot experimentation platform v-rep: A versatile 3d robot simulator," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2010, pp. 51–62.

[28] M. A. Olivares-Mendez, S. Kannan, and H. Voos, "Vision based fuzzy control autonomous landing with uavs: From v-rep to real experiments," in *23rd Mediterranean Conference on Control and Automation (MED)*. IEEE, 2015, pp. 14–21.

[29] E. Peralta, E. Fabregas, G. Farias, H. Vargas, and S. Dormido, "Development of a khepera iv library for the v-rep simulator," *IFAC-PapersOnLine*, vol. 49, no. 6, pp. 81–86, 2016.

[30] E. Rohmer, S. P. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ*. IEEE, 2013, pp. 1321–1326.

[31] C. Pinciroli, "The swarmanoid simulator," *Bruxelles: UniversitéLibre de Bruxelles*, 2007.

[32] S. Magnenat, P. Rétornaz, M. Bonani, V. Longchamp, and F. Mondada, "Aseba: A modular architecture for event-based control of complex robots," *IEEE/ASME Transactions on Mechatronics*, vol. 16, no. 2, pp. 321–329, 2011.

[33] M. Allwright, N. Bhalla, C. Pinciroli, and M. Dorigo, "Argos plug-ins for experiments in autonomous construction," Technical report TR/IRIDIA/2018-007, IRIDIA, Université Libre de Bruxelles . . . , Tech. Rep., 2018.

[34] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle *et al.*, "Argos: a modular, multi-engine simulator for heterogeneous swarm robotics," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 5027–5034.

[35] "Webots open source robot simulator," https://cyberbotics.com.

[36] O. Michel, "Webots: Symbiosis between virtual and real mobile robots," in *Int. Conf. on Virtual Worlds*. Springer, 1998, pp. 254–263.

[37] M. Olivier, "Cyberbotics LTD-webotstm: Professional Mobile Robot Simulation," *Int. Journal of Advanced Robotic Systems*, vol. 1, 2004.

[38] L. Wang, K. Tan, and V. Prahlad, "Developing khepera robot applications in a webots environment," in *MHS2000. Proceedings of 2000 International Symposium on Micromechatronics and Human Science (Cat. No. 00TH8530)*. IEEE, 2000, pp. 71–76.

[39] "Watsolutions," http://www.watsolutions.com.

[40] "Work space robot simulation," http://www.workspace5.com.

[41] "Aist," https://www.aist.go.jp/index_en.html.

[42] F. Kanehiro, H. Hirukawa, and S. Kajita, "Openhrp: Open architecture humanoid robotics platform," *The International Journal of Robotics Research*, vol. 23, no. 2, pp. 155–165, 2004.

[43] H. Hirukawa, F. Kanehiro, and S. Kajita, "Openhrp: Open architecture humanoid robotics platform," in *Robotics Research*. Springer, 2003.

[44] R. Mittal, A. Konno, and S. Komizunai, "Implementation of hoap-2 humanoid walking motion in openhrp simulation," in *2015 International Conference on Computing Communication Control and Automation*. IEEE, 2015, pp. 29–34.

[45] H. Hirukawa, F. Kanehiro, K. Kaneko, S. Kajita, K. Fujiwara, Y. Kawai, F. Tomita, S. Hirai, K. Tanie, T. Isozumi *et al.*, "Humanoid robotics platforms developed in hrp," *Robotics and Autonomous Systems*, vol. 48, no. 4, pp. 165–175, 2004.

[46] K. DeMarco, E. Squires, M. Day, and C. Pippin, "Simulating collaborative robots in a massive multi-agent game environment (scrimmage)," in *Distributed Autonomous Robotic Systems*. Springer, 2019.

[47] C. W. Reynolds, "Flocks, Herds and Schools: A Distributed Behavioral Model," in *ACM SIGGRAPH Computer Graphics*, July 1987, vol. 21, no. 4, pp. 25–34.

[48] C. Reynolds *et al.*, "Opensteer: Steering behaviors for autonomous characters," *Website. [Available: http://opensteer.sourceforge.net]*, 2004.

[49] "Robot virtual worlds," http://www.robotvirtualworlds.com.

[50] E. Rohmer, S. P. Singh, and M. Freese, "Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework."

[51] "Scrimmage," https://www.scrimmagesim.org, 2018.

[52] R. Gmyr, "Distributed algorithms for overlay networks and programmable matter," Ph.D. dissertation, University of Paderborn, Germany, 2018.

[53] O. Michel, "Cyberbotics ltd. webots™: professional mobile robot simulation," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, p. 5, 2004.

[54] "Workspace - robot simulation," http://www.workspace5.com/.