

A Framework for Autonomous UAV Swarm Behavior Simulation

Piotr Cybulski

Military University of Technology
ul. Kaliskiego 2,
00-908 Warszawa, Poland
Email: piotr.cybulski@wat.edu.pl

Abstract—In the last several years a large interest in the unmanned aerial vehicles (UAVs) has been seen. This is mostly due to an increase of computational power and decreasing cost of the UAVs itself. One of an intensively researched area is an application of a swarm behavior within team of such UAVs. Simulation tools are one of the means with which quality of solutions in this matter can be measured. In this paper such simulation framework is proposed. The proposed framework is capable of taking under consideration interferences between communicating UAVs, as well as interaction between UAV and surrounding environment. Mathematical models based on which simulation is performed were described, definition of simulation scenario and results of exemplary simulation were also presented.

I. INTRODUCTION

THE unmanned aerial vehicles, commonly called as drones, are gaining more interest by both civilian and military organizations. From an academic perspective drones are specially interesting because of the swarm intelligence, that can be implemented into them. Combining the artificial intelligence (AI) with the UAV swarms can significantly change the way of providing services such as traffic monitoring, area patrolling or, in the military area of interests, creating a situational awareness. There are many propositions how to approach the swarm intelligence subject so that results will meet the requirements [1], but there is a lack of a common framework to compare results. One of the reasons is large variety of task types for the swarms. To name a few, search and attack on a target[2][3][4], area patrolling[5][6], disaster operations[7] or transportation services[8]. Currently, there are tools for simulating specific algorithms[9] or certain scenarios [7]. There is, however, lack of a general purpose tool for UAV swarm simulation. In this paper I presented the simulation framework to fill this gap.

II. RELATED WORK

Due to large interest in the area of drone swarms, there have been developed few tools for simulating them. These are mainly for prototyping solutions and studying how does single UAV, or swarm of them, act in a particular scenario. Majority of them is designed to simulate a predetermined type of scenarios such as search and attack on target[4][9] or disaster operations[7]. Some of currently working simulators are based

on commercial software such as Matlab[10] or X-Plane[11]. Finally, there are programming languages, the Proto[12] for example, designed specifically for testing certain paradigms (amorphous computing in this case) application in the UAV swarms area.

III. PRELIMINARIES AND BASIC DEFINITIONS

Using the definition from [1] let's define the UAV swarm as a team of an autonomous unmanned aerial vehicles, where behavior of a single UAV emerges from its inner state and from surrounding environment, including neighbor UAVs. Because of wide range of tasks for UAV swarms and scale of solutions for these tasks, the following assumptions were made regarding algorithms which are compatible with proposed framework:

- 1) The UAV swarm is controllable in decentralized manner;
- 2) No common knowledge database is being used. Each UAV has to either have necessary information, or it has to be able to obtain it during a mission;
- 3) An environment in which swarm is operating is continuous three dimensional space, or it must be convertible to such.

Each simulation object “inside” framework is treated in the same way in terms of controlling its behavior. It is possible to combine the fully autonomous object, such as swarm's members, with manually controlled units (e.g. proxies between an operator and the swarm). The later described framework is not restricted to any particular types of a mission. However, it is important to point out that mission objectives supported by the framework must be evaluable solely on the situation in the moment at which evaluation is performed. We will define mission objectives as a set of states simulation objects must be in. Formally a mission objective will be described in IV-C

IV. THE BASIC SWARM MODEL

The main inspiration for basic swarm model was taken from “the asynchronous event-driven robotic network” presented in [13]. Following are main differences between basic swarm model and the one mentioned above:

- 1) A sender of a message is not known explicitly to a receiver, they can be inferred (if there is a need for it) from the message itself;

- 2) In each simulation moment there can be generated multiple messages by all objects.

A. The goal

The basic swarm model shall be capable of simulating communication between objects. Communication, if occurs, is not altered in any way between a sender and a receiver. In other words, interference is not taken under consideration.

B. The description of basic model's components

A core element of the model is a *simulation object*. Every entity modeled within simulation has to be considered as a simulation object. There are two types of simulation objects: passive and active. Passive simulation objects are aimed to represent entities such as:

- 1) On-the-ground beacons;
- 2) The GPS;
- 3) An environment.

These objects play important role in the behavior of the swarm (and not only that), but their key trait is that they are not changing its behavior due to an interaction with "outside world" (other simulation objects). Active simulation objects on the other hand are designed to represent real world's entities like:

- 1) Ground and aerial vehicles (drones for example);
- 2) Humans;
- 3) Anti-aircraft systems.

Definition 1. The Simulation Object (SO), as the core element of the simulation, consists of following components:

- 1) A logical state;
- 2) A physical state;
- 3) An identifier.

A logical state shall contain all data required to control behavior of a simulation object. For example, a logical state can be composed of information about whether the simulation object is still alive, or what is its destination.

A physical state shall contain all data required to visualize simulation object in simulation's world, example of such data might be its position and rotations about each of axes.

A SO's identifier shall be unique name of this object during simulation. Each pair of identifiers shall be comparable (on whether they are equal or not), and for the set of all identifiers the relation of order shall be established on.

The main reason for distinguishing logical state from physical state is to emphasize that they may differ even if they represent the same phenomenon. As an example let us consider a position of an object. In this case the physical state would represent actual values, one may say the values that are correct. The logical state in this case could represent data received from devices such as the GPS or some on-the-ground localization systems. It may happen that these two states will be very different from one another, specially if simulation object will not be able to establish connection with positioning system.

Definition 2. The Passive Simulation Object (PSO) is an extension of the simulation object. It can send messages,

but it cannot receive any. Passive simulation object is the simplest element that can take part in the simulation. Each PSO contains, despite what it has inherited from SO, following elements :

- 1) A physical state update function; (PSUF)
- 2) A physical state control function; It produces an input for the PSUF; (PSCF)
- 3) A messages generation function; (MGF)
- 4) A messages generation function trigger; (MGFT)
- 5) A (logical) state transition function (STF) and its trigger (STFT);

Definition 3. The Active Simulation Object (ASO) is an extension of the passive simulation object. Its capabilities extends PSO in a way that it is can receive messages. Each ASO contains, above what it has inherited from PSO, message receiving function.

C. Formal definitions of the basic model's components

Let I^{SO} denote a set of all (passive) simulation objects' identifiers. By $I^{ASO} \subset I^{SO}$ we denote set of active simulation objects' identifiers, this is a subset of the set of all the identifiers. Sets $LS_i, i \in I^{SO}$ and $PS_i, i \in I^{SO}$ denotes accordingly a set of all possible logical states of simulation object, and a set of all physical states of the same object.

For all simulation objects there is a common set of basic physical state attributes PS_0 defined as follows:

$$\forall i \in I^{SO} : PS_i = X_i^{PS} \times PS_0$$

X_i^{PS} - a set of secondary physical state attributes used by the i^{th} simulation object.

Using the above definitions we can define following behavior controlling functions.

Definition 4. The Physical State Update Function

$PSUF_i : PS_i \times U_i \rightarrow PS_i, i \in I^{SO}$ - a physical state update function used by the i^{th} object.

$U_i, i \in I^{SO}$ - a set of vectors to control the i^{th} physical state.

Definition 5. The Physical State Control Function

$PSCF_i : PS_i \times LS_i \rightarrow U_i, i \in I^{SO}$ - a physical state control function used by the i^{th} object.

Definition 6. The State Transition Function Trigger

$STFT_{i,j} : LS_i \times PS_i \rightarrow \{true, false\}, j \in I_i^{STF}, i \in I^{SO}$ - a trigger of the j^{th} state transition function used by the i^{th} object.

Definition 7. The State Transition Function

$STF_{i,j} : PS_i \times LS_i \rightarrow LS_i, j \in I_i^{STF}, i \in I^{SO}$ - the j^{th} state transition function used by the i^{th} simulation object.

Definition 8. The Messages Generation Function Trigger

$MGFT_i : PS_i \times LS_i \rightarrow \{true, false\}$ - a messages generation function trigger used by the i^{th} simulation object.

In order to define last two functions we need to define one additional set:

M^∞ - set of all messages.

Definition 9. The Messages Generation Function

$MGF_i : PS_i \times LS_i \times I^{SO} \rightarrow 2^{M^\infty}, i \in I^{SO}$ - a message generation function used by the i^{th} object.

Definition 10. The Communication Capability Function

$E_{comm} : \prod_{i \in I^{SO}} PS_i \rightarrow 2^{I^{SO} \times I^{SO}}$ – a communication capability function.

Interpretation: $\forall i, j \in I^{SO} : \langle i, j \rangle \in 2^{I^{SO} \times I^{SO}} \iff i^{th}$ and j^{th} simulation objects can communicate with each other (they can exchange messages with one another).

Definition 11. The mission objective of UAV swarm

Let $I_i^T \subset \mathbb{N}, i \in I^{SO}$ denote a set of tasks for the i^{th} object identifier.

Additionally let

$$M_i = \{T_{i,j} : PS_i \times LS_i \times \mathbb{Z} \rightarrow \{true, false\}\}_{j \in I_i^T}$$

denote a set of tasks for the i^{th} object. Interpretation of the function $T_{i,j}$ is as follows. If physical and logical state, at the moment of function execution, meet the criteria of j^{th} task then the function returns *true*, otherwise it returns *false*. Having defined tasks for the i^{th} simulation object, a mission objective is the set of all tasks for every object, that is:

$$M = \bigcup_{i \in I^{SO}} M_i$$

A mission objective is considered as completed if all tasks for every object are accomplished.

$$\forall i \in I^{SO} \forall j \in I_i^T : T_{i,j}(ps_i(t), ls_i(t), t)$$

where:

$ps_i(t) : \mathbb{Z}_+ \rightarrow PS_i$ – a function returning physical state of the i^{th} object for a given simulation moment.

$ls_i : \mathbb{Z}_+ \rightarrow LS_i$ – a function returning logical state of the i^{th} object for a given simulation moment.

$t \in \mathbb{Z}_+$ – the simulation moment at which task condition is being checked.

D. The simulation

In the following section we will use defined herein notation for description of simulation steps. Notation:

$$\forall x \in X : func_1(x), i = x; X \subset \mathbb{Z}$$

is equal, in C# programming language, to:

```
foreach (int x in X)
{
    func1(x);
    i=x;
}
```

Let's define the basic swarm model simulation as follows. For subsequent $t \in \mathbb{Z}_+$, where \mathbb{Z}_+ is the set of positive integer numbers:

- 1) $\forall i \in I^{SO}$ update physical state of the i^{th} object:

$$u_i(t) = PSCF_i(ps_i(t), ls_i(t))$$

$$ps_i(t) = PSUF(ps_i(t), u_i(t))$$
- 2) $\forall i \in I^{SO}$ check if the i^{th} object generates messages:

$$d_i(t) = MGFT_i(ps_i(t), ls_i(t))$$
- 3) Based on previous check, generate messages:

$$\forall i \in I^{SO} : d_i(t) == true \Rightarrow$$

$$M_i^\infty(t) = MGF_i(ps_i(t), ls_i(t), i)$$

- 4) Generate communication graph:

$$c(t) = E_{comm}(\prod_{i \in I^{SO}} ps_i(t))$$

- 5) Each active simulation object receives all messages from all objects it can communicate with:

$$\forall i \in I^{ASO} \forall \langle i, j \rangle \in c(t) \forall m \in M_j^\infty(t) : ls_i(t) = MRF_i(ps_i(t), ls_i(t), m)$$

- 6) Every simulation object updates its logical state:

$$\forall i \in I^{SO} \forall j \in I_i^{STF} : v_{i,j} = STFT_{i,j}(ls_i(t), ps_i(t))$$

$$\forall i \in I^{SO} \forall j \in I_i^{STF} \forall v_{i,j} == true : ls_i(t) = STF_{i,j}(ls_i(t), ps_i(t))$$

- 7) The logical and physical state of all objects at the end of each iteration becomes their initial state in the next iteration:

$$ls_i(t+1) = ls(t)$$

$$ps_i(t+1) = ps_i(t)$$

V. THE EXTENDED MODEL WITH STIMULI

The basic model was extended by adding sets of stimuli. The main reason for this was to allow the simulation objects to interact with fragments of the environment rather than directly interact with each other. All definitions from the basic model remains unchanged.

A. The goal

Main goals of the extended model were:

- 1) To allow an interference in communication between the objects to occur;
- 2) To add an influence of observer's position on content of received message. For example we can consider a loudness of a sound, that will be perceived differently by objects located in different places.

B. The description of extended model's components

All the definitions from IV-C stays unchanged, below are listed only new elements of the model.

Definition 12. The stimulus

A stimulus is a carrier of messages. By analogy we can exemplify it as radio waves.

Definition 13. The Passive Simulation Object using Stimuli (PSOuS)

A passive simulation object using stimuli is an extension of the PSO, by allowing it to emit stimuli based on previously generated messages.

It keeps limitations from the PSO, so it cannot receive any messages. Due to the above mentioned reason, it cannot receive any stimuli as well.

Each PSOuS consist of (regardless of what it has inherited from the PSO):

- 1) Stimulus emitting functions;
- 2) Selector of stimulus emitting function.

Definition 14. The Active Simulation Object using Stimuli (ASOuS)

A ASOuS is an extension of both the PSOuS and the ASO. It can generate messages, emit stimuli, receive them and receive messages.

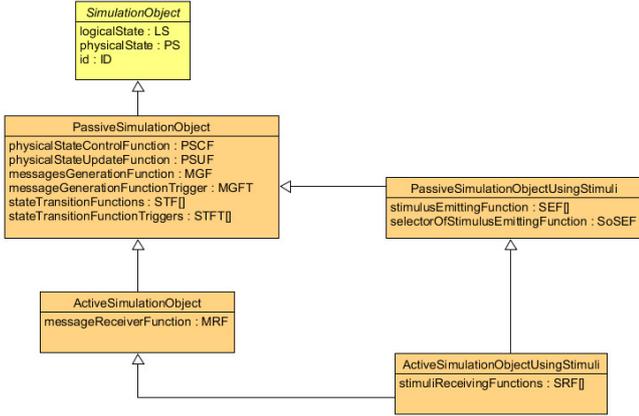


Fig. 1. Class diagram with the hierarchy of simulation object types.

Each ASOuS despite of what it has inherited from the PSOuS and the ASO, has stimuli receiving function.

A class diagram presenting the simulation object types hierarchy is presented on the figure 1.

C. Formal definitions of the extended model's components

Let I^S denote a set of stimuli types identifiers, and by $S_i, i \in I^S$ we denote a set of i^{th} type stimuli. Each set of stimuli must define following operations:

$+$: $S_i \times S_i \rightarrow S_i$ – the addition of the i^{th} type stimuli;
 P_i : $PS_0 \times S_i \rightarrow S_i$ – the perception of the i^{th} type stimulus;
 To clarify, each set of stimuli defines above operations only for its own type. Additionally, each set of stimuli must define a neutral element of itself.

$$\forall i \in I^S \exists! e_i \in S_i \forall s \in S_i : +_i(s, e_i) = s$$

Finally, there must be a definition of perception operation on a subset of stimuli:

$$P_i^2 : PS_0 \times 2^{S_i} \rightarrow S_i, i \in I^S$$

Definition 15. Stimulus Emitting Function

Let's denote a set of all stimulus emitting functions kept by i^{th} simulation object as SEM_i^∞ . Formal definition of the SEM_i^∞ set is as follows:

$\{SEM_{i,j,l} : PS_i \times LS_i \times M^\infty \rightarrow S_j\}_{l \in \mathbb{Z}_+, j \in I^S, i \in I^{SO}}$
 SEM_i^∞ – a set of all stimulus emitting functions kept by the i^{th} simulation object.

$SEM_{i,j,l}$ – a l^{th} stimulus emitting function being used by the i^{th} simulation object. The stimulus emitted by this function is of the j^{th} type.

Definition 16. Selector of Stimulus Emitting Function

$SoSEM_i : PS_i \times LS_i \times M^\infty \times 2^{SEM_i^\infty} \rightarrow 2^{SEM_i^\infty}, i \in I^{SO}, j \in I^S$ – a selector of stimulus emitting function used by the i^{th} object.

Definition 17. Stimuli receiving function

$SRF_i^\infty = \{SRF_{i,j,l} : PS_i \times LS_i \times 2^{S_j} \rightarrow M^\infty\}_{l \in \mathbb{Z}, i \in I^{SO}, j \in I^S}$

SRF_i^∞ – a set of all stimuli receiving functions used by the

i^{th} object.

$SRF_{i,j,l}$ – a l^{th} stimuli receiving function kept by the i^{th} object. This function accepts stimuli of the type j .

Definition 18. Functional assigning corresponding stimuli type to emitters and receivers

$S^T : SRF_{i,j} \rightarrow I^S, i \in I^{SO}, j \in I^S$ – a functional assigning a stimulus type to a stimuli receiving function.
 $S^T : SEM_{i,j} \rightarrow I^S, i \in I^{SO}, j \in I^S$ – a function assigning stimulus type to a stimulus emitting function.

D. The simulation

All the definitions from IV-D remains unchanged. In order to describe the simulation process of the extended model, first we need to define the following function:

Definition 19. The stimuli set at simulation moment function

$s_i : \mathbb{Z}_+ \rightarrow 2^{S_i}, i \in I^S$ – a function assigning each subsequent simulation moment its corresponding set of i^{th} type stimuli.

The simulation is performed according to the following steps.

For each subsequent $t \in \mathbb{Z}_+$:

- 1) See the step 1 of IV-D;
- 2) See the step 2 of IV-D;
- 3) See the step 3 of IV-D;
- 4) For every message generated by each simulation object select a set of emitters:
 $\forall i \in I^{SO} \forall m \in M_i^\infty(t) : S_i^{SEL}(t) = S_i^{SEL}(t) \cup \{ \langle SoSEM_i(ls_i(t), ps_i(t), m, SEM_i^\infty), m \rangle \}$
- 5) Using each selection made in previous step generate messages:
 $\forall i \in I^{SO} \forall \langle S^{SEM_s}, m \rangle \in S_i^{SEL}(t) \forall sem \in S^{SEM_s} : j = S^T(sem), s_j(t) = s_j(t) \cup \{ sem(ps_i(t), ls_i(t), m) \}$
- 6) With every stimuli receiver of each object perform perception operation on the appropriate set of stimuli:
 $\forall j \in I^S \forall i \in I^{SO} \forall srf \in SRF_i^\infty : S^T(srf) == j \Rightarrow m_i^r(t) = m_i^r(t) \cup srf(ps_i(t), ls_i(t), s_j(t))$
- 7) Each simulation object receives all messages generated by its receivers in the previous step:
 $\forall i \in I^{SO} \forall m \in M_i^\infty(t) : ls_i(t) = MRF_i(ps_i(t), ls_i(t), m)$
- 8) See the step 6 of IV-D;
- 9) See the step 7 of IV-D.

VI. THE FRAMEWORK ARCHITECTURE

The framework architecture will be described according to [14]. The entire tool for the simulation of the UAV swarms can be divided into two web services and a client application. The client application is responsible for performing the simulation. Two web services provided by the framework, the first one to register every simulation moment, and the second one to visualize results of the simulation in 3D, are designed to reconstruct the course of the simulation later on (with vastly lower computational cost).

In the following architecture description we will primarily focus on the client application, because it is the only element that require implementation from a framework's user.

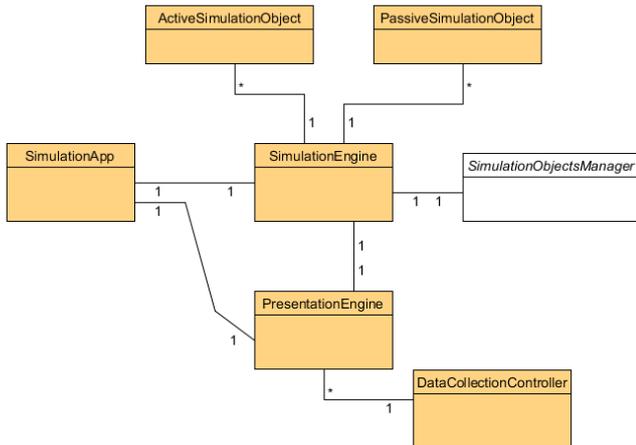


Fig. 2. The diagram of the main conceptual classes

A. The logical view

The Figure 2 shows main conceptual classes that allows performing and saving simulation of the basic model results. The classes mentioned above are:

- **SimulationApp** – is the facade aimed to control the simulation. It is also responsible for management of the threads required to perform the simulation;
- **PresentationEngine** – is responsible for communication with the web service and saving simulation state on a server;
- **SimulationEngine** – supervises the course of the simulation and communicates with the presentation engine;
- **SimulationObjectsManager** – is the main element responsible for calculations needed by the simulation. It is also the element of which implementation quality impacts the performance of the simulation process the most;
- **DataCollectionController** – is the class responsible for registering the subsequent simulation moments;
- **ActiveSimulationObject** – is the class that represents the active simulation object. It contains all the elements depicted on the Figure 1
- **PassiveSimulationObject** – is the class that represents the passive simulation object. It contains all the elements depicted on the Figure 1.

The **ActiveSimulationObject** (usingStimuli) and **PassiveSimulationObject** (usingStimuli) classes are the only the components that requires further implementation.

B. The process view

In the client module we can distinguish 2 main processes, as is depicted in Figure 3. The first process calculates subsequent states of the simulation, the second one communicates with the web services so the results from the first one can be saved.

It is worth mentioning that the above perspective is simplified, it doesn't involve any optimization (such as parallelization of calculation for every simulation object).

C. The physical view

From the physical point of view, the client module instances can be located on the same device as the server is. Although, it is suggested to not share location of both modules.

Multiple client modules can be instantiated on one computer. It is also highly recommended for the computer that the database will be on to have sufficiently fast storage drive. It should be remembered that accessing the drive will occur much more often when reconstructing (visualizing) simulation, than when it is being performed (registered). The exemplary configuration is presented in the Figure 4.

D. The developer view

The framework can be divided into 2 main modules, the client module that is doing the calculation, and the web module that registers the simulations, and it allows to access them later on.

The framework user needs to implement components such as functions to control physical state and messages generation/receiving function, they are all located in client module.

The client module consists of a set of dynamically loaded libraries (DLLs), while the web module is made of a WAR file that can be deployed on webservers like Payara[15] or Glassfish[16].

E. The scenarios

The simplified scenario of the basic model's simulation was presented in the Figure 5. It is important to notice that steps 1-5 include, omitted on the picture, returning values needed for presentation layer from a manager.

Similarly the extended model's simulation scenario will only differ in number of steps it requires.

VII. THE FRAMEWORK IMPLEMENTATION

The framework for the UAV swarms simulation was implemented in two technologies. The client module was implemented using C# (.NET Framework 4.7.2), and the web module was developed using JEE with frontend PrimeFaces framework in version 6.2. A 3D visualization of the simulation results is being performer with ThreeJS library.

The main reason why the client module was developed in the different language than the web module was the type erasure mechanism in the Java.

The following components are delivered as part of the framework:

- A basic implementation of physical state;
- A basic implementations of physical state control and update functions;
- A set of basic stimuli types;
- A tool for image-to-environment generation.

VIII. AN EXEMPLARY SCENARIOS SIMULATION

The scenario of a UAV swarm mission consist of three elements:

- 1) Generating a map of a mission environment;
- 2) Configuring the simulation objects;

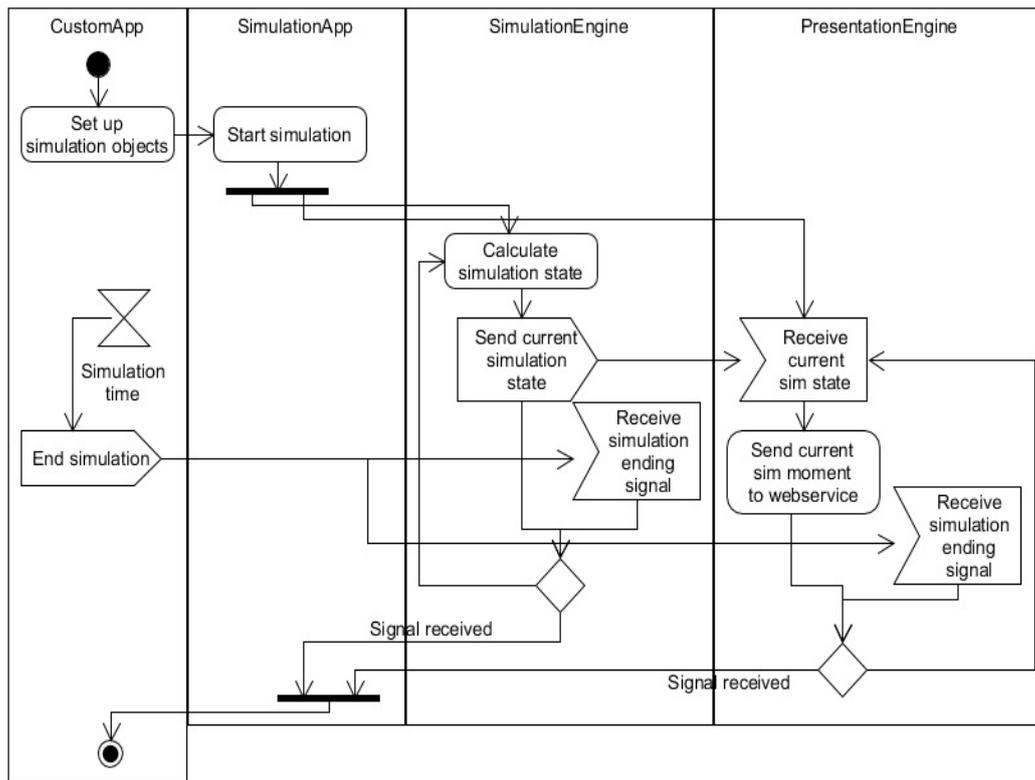


Fig. 3. The diagram of processes in the client module

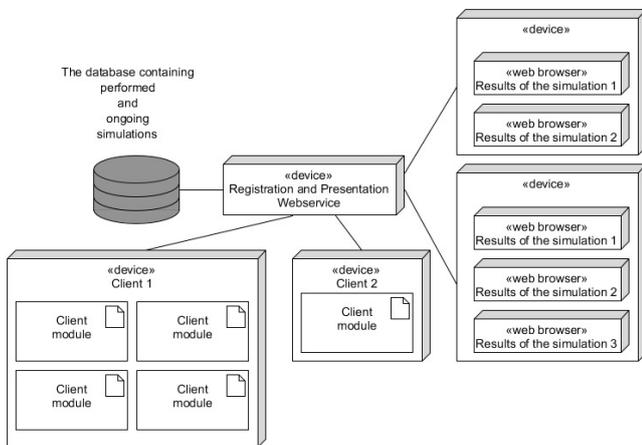


Fig. 4. The exemplary physical configuration

3) Setting up the mission objective for the simulation objects.

The mission environment map generation is a matter of two things. The first thing is to set parameters of the map, its width, depth, maximum heightens etc. The second step is to select an image that represents the map. All these can be done with the provided tool (see Figure 7).

The simulation objects configuration is essentially about

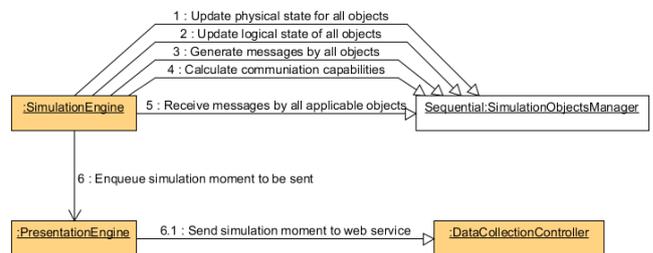


Fig. 5. The simulation of the basic model scenario

implementing functions described in IV or/and V.

The exemplary simulation scenario aimed to check whether a homogenous set of an unmanned aerial vehicles will be capable to reach a certain destination while avoiding environmental obstacles.

The environmental map was generated using bitmap depicted in the Figure 6.

Interpretation of the Figure 6 in the context of the map of environment is as follows:

- Height along the X axis will depend on the brightness of pixels along width of the picture;
- Height along the Z axis will depend on the brightness of pixels along height of the picture;

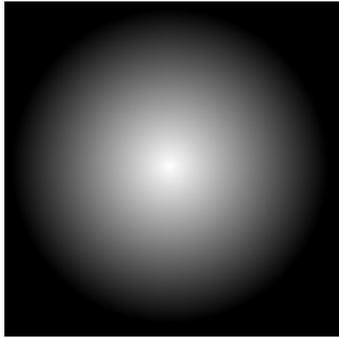


Fig. 6. The map of the mission environment used in the exemplary simulation scenario

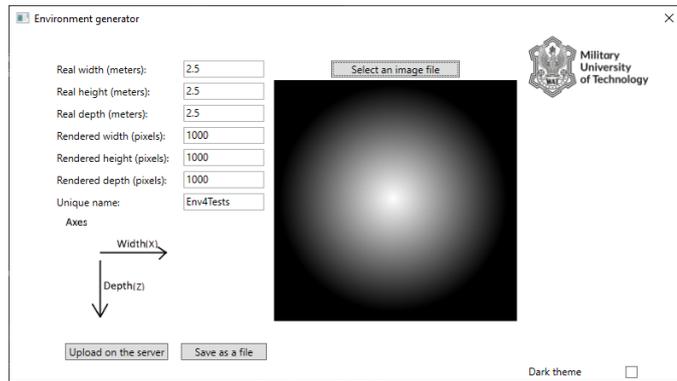


Fig. 7. The tool for mission environment generation

- The origin of the coordinate system (on the map) corresponds to the upper left corner in the picture.

The real world dimensions of the generated map were 2.5 meters by 2.5 meters. Maximum height (corresponding the white area in the Figure 6) was also set to 2.5 meters.

Four homogenous aerial vehicles have been placed on the map. Each of them had a goal to reach position (assuming xyz coordinate order) at point (2.5,0.1,2.3).

The simulation was performed using the extended model with one type of stimuli, let's call it collision type stimuli. A structure of the collision type stimulus is as follows:

$$S_1 = \{x : x \in R^3 \times \{true, false\}\}$$

The results confirmed possibility of performing the simulation with scenario described earlier. Three out of four objects reached their destination, the one that did not make it failed because of a flaw in an algorithm and not because of the framework. Below are listed figures showing particular simulation moments. The Figure 8 shows the beginning of the simulation, while Figure 9 shows middle of the simulation and the Figure 10 shows the simulation state at the end of the simulation process.

It is worth to mention that a map of the environment can be generated from any bitmap file. For example, more “advanced” map was generated using the file from Figure 11

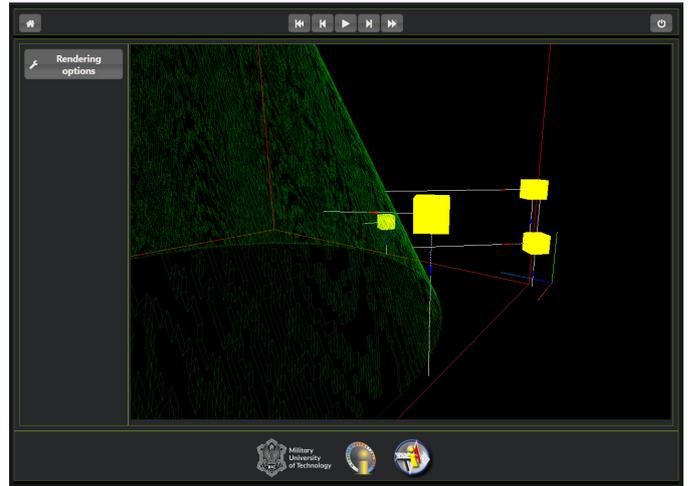


Fig. 8. The screenshot with results of a simulation in 14th simulation moment

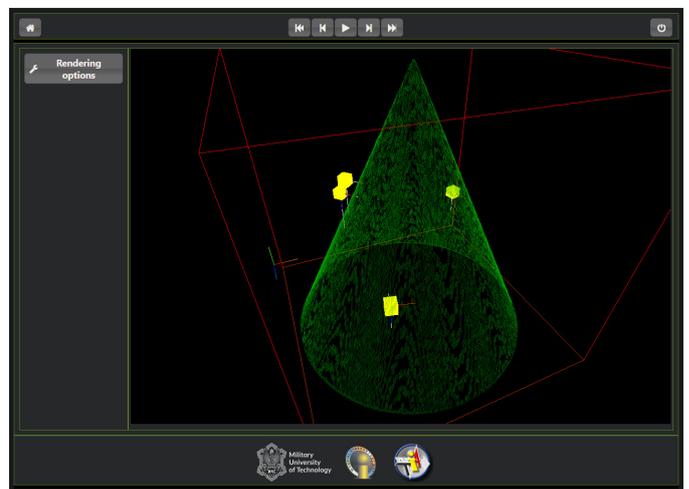


Fig. 9. The screenshot with results of a simulation in 109th simulation moment

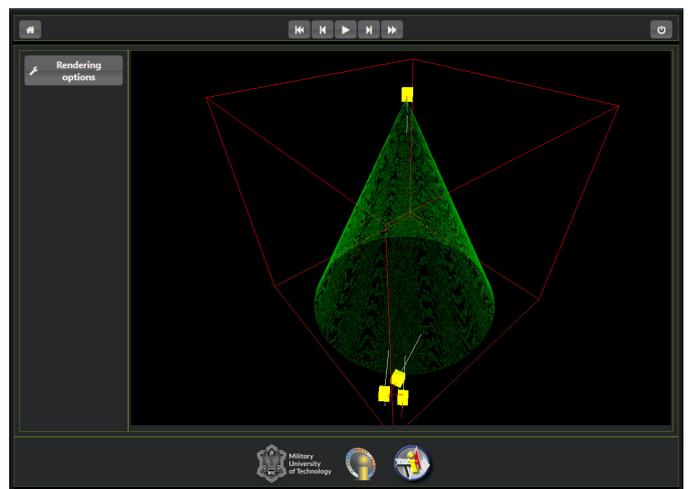


Fig. 10. The screenshot with results of a simulation in 431th simulation moment

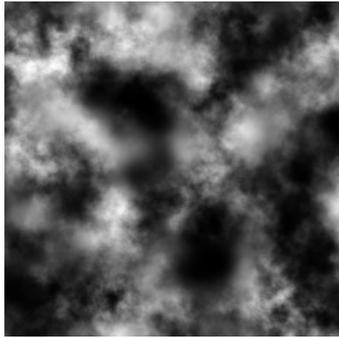


Fig. 11. An example of a mission area

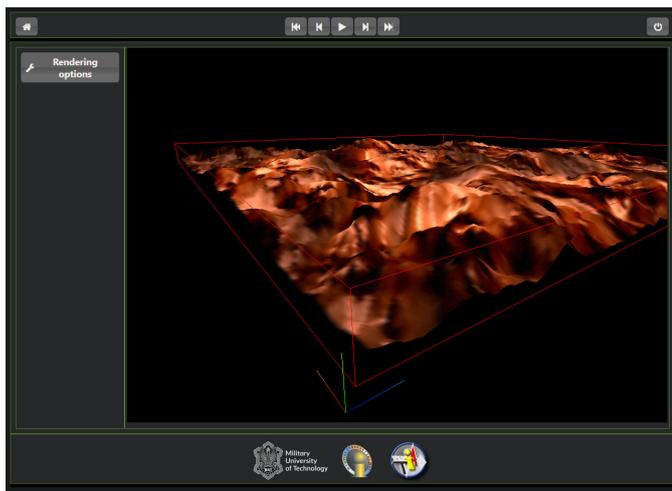


Fig. 12. A more advanced mission area – first view from the simulator

The mission environment generated from the Figure 11 was depicted on Figure 12 and Figure 13.

Each map can be rendered in two modes : solid and mesh. The map from Figure 12 shows the solid rendering mode, while Figure 13 depicts mesh rendering mode.

IX. CONCLUSIONS

The goal of this work was to present the framework for the UAV swarm simulation with any scale and any type of tasks that may be set for it. Two simulation models, that are acceptable by the simulator, were presented, as well as their goals and limitations. The framework allows to perform simulation with user-defined map of environment, this is particularly useful when combined with the second model.

Results of an exemplary simulation scenarios were also presented.

There are two direction for future work on this framework. The first one is to design a script-like language so that a simulation objects configuration can be done without need of programmatic implementation. The second direction is to allow the simulation objects to be configured with languages other than C#, this would probably end up in designing a meta-framework for the tool presented here.

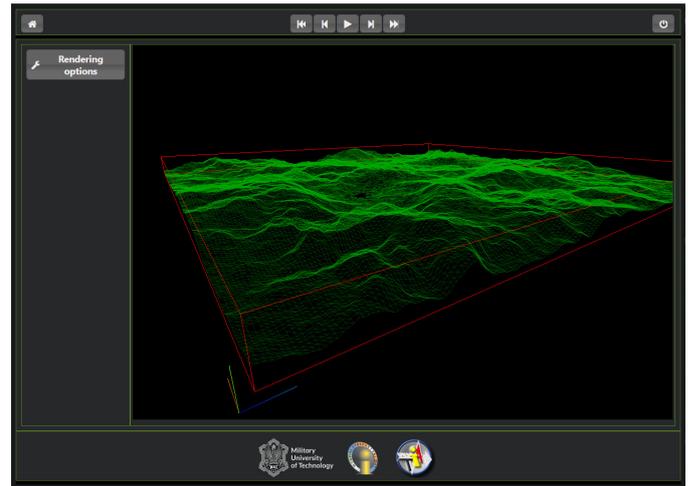


Fig. 13. A more advanced mission area – second view from the simulator

REFERENCES

- [1] A. Kolling, P. M. Walker, N. Chakraborty, K. P. Sycara, and M. Lewis, "Human interaction with robot swarms: A survey," *IEEE Transactions on Human-Machine Systems*, vol. 46, pp. 9–26, 2016.
- [2] C. Gao, Z. Zhen, and H. Gong, "A self-organized search and attack algorithm for multiple unmanned aerial vehicles," *Aerospace Science and Technology*, 2016.
- [3] M. Kim, H. Baik, and S. Lee, "Response threshold model based uav search planning and task allocation," *Journal of Intelligent & Robotic Systems*, 2014.
- [4] H. Cheng, J. Page, and J. Olsen, "Dynamic mission control for uav swarm via task stimulus approach," *American Journal of Intelligent System*, vol. 2, pp. 177–183, 01 2013.
- [5] G. Wang, Q. Li, and L. Guo, "Multiple uavs routes planning based on particle swarm optimization algorithm," *International Symposium on Information Engineering and Electronic Commerce*, 07 2010.
- [6] W. Zhenhua, Z. Weiguo, S. Jingping, and H. Ying, "Uav route planning using multiobjective ant colony system," in *Conference on Cybernetics and Intelligent Systems*, pp. 797 – 800, 10 2008.
- [7] T. Ahmed, D. Feil-Seifer, T. Jiang, S. Jose, S. Liu, and S. Louis, "Development of a swarm uav simulator integrating realistic motion control models for disaster operations," 10 2017.
- [8] M. Brust, G. Danoy, P. Bouvry, D. Gashi, H. Pathak, and M. P. Goncalves, "Defending against intrusion of malicious uavs with networked uav defense swarms," pp. 103–111, 10 2017.
- [9] J. Wang, Y. Tang, J. Kavalen, A. Abdelzaher, and S. P. Pandit, "Autonomous uav swarm: Behavior generation and simulation," in *Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 1–8, 06 2018.
- [10] S. Rasmussen, J. Mitchell, P. Chandler, C. Schumacher, and A. Smith, "Introduction to the multiuav2 simulation and its application to cooperative control research," pp. 4490 – 4501 vol. 7, 07 2005.
- [11] R. Garcia and L. Barnes, "Multi-uav simulator utilizing x-plane," *Journal of Intelligent and Robotic Systems*, vol. 57, pp. 393–406, 01 2010.
- [12] J. Bachrach, J. Mclurkin, and A. Grue, "Protoswarm: A language for programming multi-robot systems using the amorphous medium abstraction," vol. 2, pp. 1175–1178, 01 2008.
- [13] F. Bullo, J. Cortés, and S. Martínez, *Distributed Control of Robotic Networks*. Applied Mathematics Series, Princeton University Press, 2009. Electronically available at <http://coordinationbook.info>.
- [14] P. Kruchten, "Architectural blueprints – the 4+1 view model of software architecture," *IEEE Software* 12, 1995.
- [15] "<https://www.payara.fish/>"
- [16] "<https://javaee.github.io/glassfish/>"