Context-Aware and Perceptually Realistic Synthetic Wrapping for Military Training and Exercises

Olaf Visker, Annemarie Burger, Remco van der Meer, Anne Merel Sternheim, Ruben Smelik
Modelling, Simulation & Gaming Department
TNO Defense, Safety & Security
The Hague, The Netherlands
{olaf.visker, annemarie.burger, remco.vandermeer, anne_merel.sternheim, ruben.smelik}@tno.nl

ABSTRACT

Armed forces must constantly be ready for deployment in both national and international operations. Large-scale exercises ensure that different forces learn to co-operate, work together as a team, and maintain a level of preparedness. There are however limitations to the level on which exercises can be performed. Lack of available personnel, material, specific capacities and insufficient space and budgets result in difficulties with performing exercises with higher echelons involved. Even for small-scale training events similar difficulties can occur, especially when team collaboration is required. To be cost-effective and simultaneously meet training goals, training often requires a mix of live, virtual, and constructive simulations (LVC). Synthetic wrapping is a form of LVC simulation that enriches training through virtual and constructive simulation. Simulated units that participate autonomously in live exercises and training modules need to be able to communicate with the primary training audience, behave perceptually realistic, and contribute to the learning goal.

This research explores a variety of Artificial Intelligence techniques in the context of creating perceptually realistic synthetically wrapped units. It focuses on three different domains: (i) communication via natural language, (ii) understanding the environment, and (iii) adaptive behavior. We present a synthetic-wrapping framework and a proof of concept on a representative use-case: Joint Terminal Attack Controller (JTAC) training. In the use-case, the JTAC requests Close Air Support (CAS) from a simulated fighter pilot and guides the pilot towards a target. The proposed framework is successfully implemented for the use-case and supports adaptive behavior based on the environment and natural interaction with a simulated unit. Our results show that modern Artificial Intelligence techniques can aid in the creation of perceptually realistic simulated units that allow for believable interactions and behaviors during military training and exercises.

ABOUT THE AUTHORS

Olaf Visker is a research scientist at TNO Defence, Safety & Security. He graduated from the Rijksuniversiteit Groningen with a Master's Degree in Artificial Intelligence, specializing in Computational Intelligence and Robotics. At TNO, he focuses artificial intelligence, data driven decision support and innovation within the military and national security domain.

Annemarie Burger is a research scientist at TNO Defence, Safety & Security. She holds a bachelor's degree in Artificial Intelligence from the University of Amsterdam, and a master's degree in Big Data Management & Analytics from the Technical University of Berlin, specializing in distributed graph stream processing. At TNO, she focuses on innovations on topics related to AI, data and computer software.

Remco van der Meer is a research scientist at TNO Defence, Safety & Security. He received a MSc degree in Applied Mathematics from Delft University of Technology, specializing in Computational Science and Engineering. At TNO, he focuses on innovations for simulation applications and automatic synthetic environment modelling.

Ruben Smelik is a research scientist at TNO Defence, Safety & Security. He holds a MSc degree in Computer Science from the University of Twente and earned his PhD degree from Delft University of Technology based on his thesis on the automatic creation of 3D virtual worlds. At TNO, he focuses on innovations in the field of automated synthetic environment modelling for military simulation applications.

Anne Merel Sternheim is a research scientist at TNO Defence, Safety & Security. She received a bachelor's degree in Linguistics from Radboud University Nijmegen, and a master's degree in Artificial Intelligence at the University of Utrecht. At TNO, she specializes in the extraction of information from textual data sources.

Context-Aware and Perceptually Realistic Synthetic Wrapping for Military Training and Exercises

Olaf Visker, Annemarie Burger, Remco van der Meer, Anne Merel Sternheim, Ruben Smelik
Modelling, Simulation & Gaming Department
TNO Defense, Safety & Security
The Hague, The Netherlands
{olaf.visker, annemarie.burger, remco.vandermeer, anne_merel.sternheim, ruben.smelik}@tno.nl

INTRODUCTION

The Netherlands Armed Forces must constantly be prepared for deployment in both national and international operations. Maintaining this level of preparedness and the preparatory process is one of the Army's core tasks. Units can be deployed in all manners of configuration. Large-scale exercises ensure that different forces learn to co-operate and work as a team on various levels. Furthermore, they allow the Army to test and apply new developments and innovations in the field.

There are however limitations to the level on which exercises can be performed. Lack of deployable personnel, material, specific capacities, insufficient space, and financial restrictions result in difficulties with performing exercises with higher echelons. Compromises must be made as live training is no longer sufficient as the sole training method. To be cost-effective and simultaneously meet training standards, training requires a mix of live, virtual, and constructive simulations (LVC) (Muller, 2011).

Live, Virtual, and Constructive simulation, as defined by the American Department of Defense, consists of three distinct categories (Department of Defense [DOD], 2016):

- Live – Simulations in which real people use real systems in a real environment, such as soldiers on field training missions. This is the traditional method of training.
- Virtual – Simulations in which real people operate in a simulated environment (virtual reality), such as a pilot in a flight simulator.
- Constructive – Simulations involving simulated (i.e., Computer Generated Forces (CGF) or Artificial Intelligence (AI)) units operating in a simulated environment, such as a war game for mission planning or a real-time strategy game.

Live simulation is an immersive form of training as real operational systems interact with one another to augment scenario complexity. Combining live simulation with virtual simulations provides new opportunities for training, the main advantages being the possibility to simulate costly and scarce assets, not being limited to physical restrictions, the amount of control over a scenario, and the efficient delivery of training.

Synthetic wrapping is a form of LVC simulation that combines the aforementioned categories. It involves enriching live instrumented training through both virtual and constructive simulation. Synthetic wrapping allows users to inject exercises with synthetic or simulated units and systems. For example, a Joint Terminal Attack Controller (JTAC) might practice calling in close-air-support (CAS) in a real environment and watch an airstrike being performed virtually through a tablet.

This research explores a variety of AI techniques in the context of creating perceptually realistic synthetically wrapped units. Perceptual realism in this instance means that we focus on observable interactions important for conveying realistic behavior. For example, in the case of a CAS training where the trainee performs the role of a JTAC, we are not interested in realistically modelling the physical forces acting on the wings of a jet fighter, as this interaction will not be observed by the JTAC in the field. However, the communication between the constructive fighter pilot and the JTAC is important, as unrealistic or unnatural behavior in their communication may result in a degraded training.

In order to achieve this perceptual realism, we focus on three different domains:

i. Communication via natural language: constructive units need to be able to communicate via natural language and understand intention and goals that are communicated.
ii. Understanding the environment: constructive units need to be able to make sense of the environment in which they operate and be able to deal with uncertainty in information.
iii. Adaptive behavior: constructive units need to be able to correctly adapt to different scenarios and unexpected interactions.

We present a synthetic-wrapping framework for creating perceptually realistic units that implement the three domains mentioned above. We show the implementation of the framework in a representative use-case, namely a training scenario where a JTAC requests close-air-support. For practical purposes, we have replaced the live, in the field, part of the synthetically wrapped training with a man-in-the-loop simulator.

JTAC personnel are responsible for calling in close-air-support. They are part of a Tactical Air Control Party (TACP) and help guide a pilot to its target. The protocol for CAS can be divided into three different steps:

i. The check-in where both the JTAC and the pilot communicate their call signs, location and other relevant information.
ii. Once a check-in has been completed, a JTAC proceeds with communicating a nine-line. The nine-line consists of a protocolized message with information regarding the details of the air support and target.
iii. After a nine-line is completed and final remarks are given, a talk-on is performed. A talk-on consists of verbally guiding a pilot towards its target. Although Digitally Aided Close Air Support (DACAS) is an increasingly important operational capability and provide significant benefits, voice transmissions currently remain the principal means of communication during CAS operations (Reitz et al., 2018; Seavey et al., 2019).

Our implementation of the above use-case features a CAS scenario where a JTAC, standing on an observation point, guides our synthetically wrapped fighter jet towards a target vehicle in a urban setting, using a manual talk-on in natural language.

Related work on the topic of synthetic wrapping includes that of van den Bosch and Boonekamp (2014). In their research they focus on developing technology that aims to solve the problems that come with the reliance on the presence of Subject Matter Experts (SMEs) in training programs. Specifically they aimed to develop virtual humans (agents) to play the supporting roles autonomously in a realistic and intelligent manner. They used a Believe-Desire-Intentions model (BDI) along with a speech recognition system to create a virtual pilot to assist in the training of Helicopter Directing Officers (HDOs).

Other work by Muller et al (2011) focused on creating a flexible and contextually rich environment for training in Urban Short Range Interaction (USRI). Constructive models allowed for virtual characters to act and respond intelligently to trainees and events. They achieved this by tracking trainee behavior, modeling behavior of virtual characters and rendering behavior of virtual players in the training environment.

This paper is structured as follows: the next section presents a high-level overview of the agent architecture developed for synthetically wrapped units. In the subsequent sections, a technical discussion on the three main modules of the architecture is given: communication, terrain reasoning, and agent behavior. After that, the implementation of the demonstrator is discussed and the results are presented. The paper is concluded with a short recap and a discussion of features and limitations of our approach and the way ahead for synthetic wrapping.

FRAMEWORK ARCHITECTURE

Our framework is made up of several different modules. Each module is responsible for a different task. This separation of concerns aids in the reusability of each component and allowed for parallel development during our research. We define three main modules in our framework:

- The communication module is responsible for interpreting natural language and extracting the intention and relevant elements from sentences.
- The terrain module is responsible for querying and extracting different features from the landscape.
- The decision module is responsible for acting on the current state of the world and determining the best course of action.

Figure 1 gives an overview of our framework architecture. Each of the modules is integrated into our constructive agent. This agent, in turn, acts upon a simulated world. This simulated environment has minimal dependencies on our agent, allowing different simulation software packages to be coupled with our framework. For our use-case, we opted to go with Virtual BattleSpace (VBS) for our simulation environment (Bohemia Interactive Simulations, 2022).

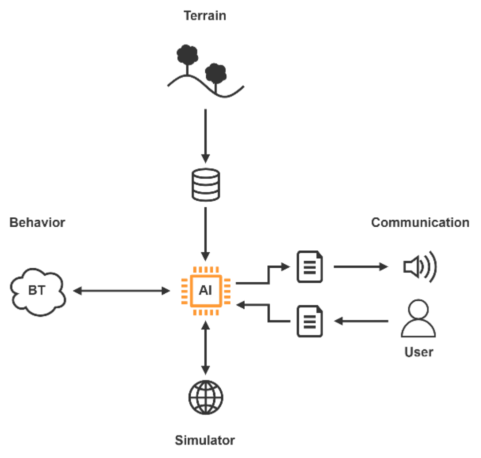


Figure 1. Framework Architecture Overview.

COMMUNICATION MODULE

In order to extract intent and relevant elements from natural sentences, the communication module includes two types of text parsers, which are used for different purposes. Firstly the template parser, which is used when communication follows a strict protocol. Secondly the free-format parser, which is used when communication may follow a free(r) format. The aim of both parsers is to identify and label relevant pieces of text for further processing in the interpretation part of the agent behavior module. The input for both parsers is a textual representation of the spoken input. Due to voice variability and background noise, current speech recognition technology is unable to transcribe input speech perfectly (Edmon et al., 2018). However, as this research does not focus on speech-to-text capabilities but on identifying and labeling sentences, we have assumed a perfect speech-to-text conversion.

Template Parser

The template parser makes use of templates, which specify the expected format of a piece of text. In order to identify the part of the input sentence that matches the template best, the template is compared against different subsets of the entire utterance. Multiple templates can be defined and matched against the same piece of text in order to determine the most likely label for a piece of text.

For example, a nine-line is a protocolized utterance, in which it is known exactly which elements to expect (e.g. '*Initial point*' and '*Target elevation*'), and in which order. Moreover, the appearance of the elements is known (e.g., the initial point is a location, and target elevation is expressed in feet above mean sea level). In this small example, the most likely label for "*two two zero feet*" is therefore '*Target elevation*'.

A template expresses the generic format of a particular type of text (what a labeled element within the utterance 'looks like'). Similar to (Tanaka, et al., 2019), we have created a repository of words which is sorted by conceptual word categories, such as locations, cardinal or relative directions, numbers, and measurement units. The template checks whether the expected word categories occur in the expected order in the input text. Sequences of words with variable length (n -grams) extracted from the input texts are compared against these word categories using a score which is based on the Levenshtein distance (SeatGeek Inc., 2014). This is a string edit distance that expresses the similarity of two input texts based on the insertions, deletions and substitutions needed to turn one input into the other. It is assumed that the input word (sequence) is part of the word category to which the highest similarity score (i.e., shortest Levenshtein distance) was found.

For example, the template for target elevation specifies that target elevation is expressed with at least 3 words, of which the last express a measurement unit (e.g. feet), and the first express a sequence of numbers (e.g. "*two two zero*"). This template will return a perfect score (1) for "*two two zero feet*", but a score close to or equal to zero for e.g. "*CHLOE*", which expresses a named location.

However, even in a strictly protocolled conversation, there may be pieces of the input sentence which you wish to label, but are not suitable for templating. This may for example be the case when the conversation protocol includes a “comments” part, where any phrasing is allowed. In order to be able to label these pieces of text as well, the template was left empty and a positional heatmap was created. A positional heatmap reflects the expected applicability of a template to a piece of text, based only on the relative position of this piece of text. For each template, for each position (i.e. word) in a text, a probability for that piece of text for that template is returned, which reflects the likelihood that that template is centered around that sentence position.

The application of templates and heatmaps to a text may initially result in multiple labels for one piece of text and multiple pieces of texts ascribed to one label. Based on two assumptions, the results are trimmed until each label is ascribed to exactly one piece of text. In the first iteration of this solving process, the following assumption is made: a word (as part of a sentence piece) cannot be labeled by two different templates. Concretely, if exactly one piece of text is ascribed to a label, all sentence parts that contain at least one word from this particular piece of text are removed from the result set. This may result in another label which has been ascribed exactly one label. The logic is repeated until no changes can be made to the result set any more. In the next iteration, if no solution is found yet, the following assumption is made: the text contains as little redundancy as possible, therefore a superset of words that also matches the template should be labeled by that template rather than the subset. Concretely, if two pieces of text match a template perfectly and one encompasses the other, the longest piece of text is chosen. Still, it may occur that the problem was not solved after these two iterations. Based on the use-case, more specific assumptions and solutions may be implemented.

The template parser has proved to work well for our use-case. However, the positional heatmap more often decreased than increased the performance of the parser. Analysis showed that this is due to the positional heatmap depending on the relative lengths and positions of the sentence elements. We expect that depending on the relative order of the elements instead of their relative length and position might generate more reliable results.

Free-format Parser

The free-format parser extracts different types of objects and (related) characteristics from a text. It also classifies the text with a ‘context’, which expresses the conversational goal of a given sentence.

Similar to the template parser, the free-format parser makes use of a score based on the Levenshtein distance to determine how close a word from the input text is to a category of words (e.g. objects, directions, colors or sizes). If the similarity score for a word and a category of words is higher than a given threshold, the word is labelled as belonging to that category of words. All words are lemmatized, stripped of interpunction and lowercased before being compared against the different word categories. Stop words, from Looper and Bird (2002), are removed as well.

Once all words have been labelled, extracted characteristics (such as color and size) are related to objects (such as houses and bridges). This is primarily done by applying the simple assumption that (in English) all characteristics occur before the objects that they relate to in the sentence, and as close as possible. Depending on the use-case, additions and modifications may be made. For example, when the sentence structure “<object A> with the <characteristic> <object B>” (i.e.: ‘the house with the red roof’) occurs, where grammatically <characteristic> relates to <object B>, we want to relate <characteristic> to <object A>. The dependencies determined by the spacy dependency parser (Honnibal & Ines, 2017) are exploited in order to find these kinds of sentence structures.

Finally, the ‘context’ of the text can be determined with DeepPavlov (Burtsey, et al., 2018), a deep learning model for intent classification that was fine-tuned on our use-case. However, this implementation is computationally heavy. Given that our use-case contained a limited number of different contexts we opted for a simpler implementation and instead looked for key-words that determine the context of a given sentence.

For example, “the house with the red roof” is labeled as a reference to a point of interest. ‘House’ is recognized as an object, to which the characteristic ‘red’ is applicable.

TERRAIN REASONING MODULE

The main purpose of the terrain reasoning module is to provide geospatial awareness for the synthetically wrapped unit, e.g., a virtual fighter pilot. To this end, the module has two components: an underlying database that contains points of interest that would be visible to the synthetic unit, and a set of filters with which this database can be queried in a convenient way.

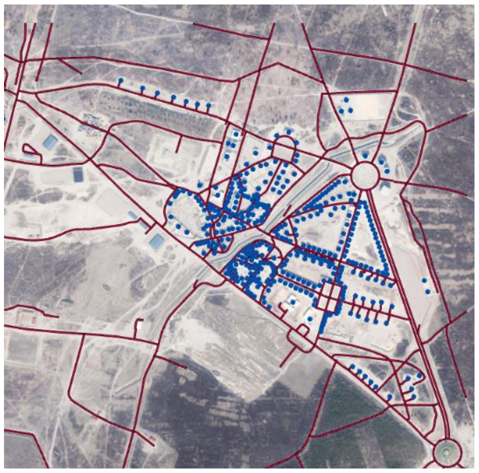


Figure 2. Part of the Terrain Reasoning Database Used in the Case Study Visualized.

Terrain Database

Geospatial data are traditionally stored in either raster data, e.g. satellite imagery or elevation maps, or vector data, in which attributes are stored alongside geo-referenced primitive geometric shapes, such as polygons, linestrings and points. For the database used for terrain reasoning, the latter form is the most convenient option.

To be able to provide geospatial awareness, a sufficiently detailed and accurate terrain database is needed (see Figure 2). Most importantly, there has to be a strong correlation between the terrain database and the visual representation provided to the trainee (either live outdoors or a virtual representation in a simulator). Objects that can be seen by the trainee are likely to be described by their visual properties, and such a description should be understood by the agent. Specifically, this means that visual information should be available in queryable form within the terrain database. For instance not only do we need the type, height and footprint of a building, but also any distinct visual properties, such as a red roof or a large chimney. Table 1 provides an overview of the features and attributes used within our use-case.

Table 1. Features and Attributes in the Terrain Database.

Feature	Shape	Attributes
Buildings	Points	Height of the building in m
		Footprint area in m^2
		Type, e.g., residential, facility
		Roof color, e.g., red, gray
		Footprint shape, e.g., simple, complex
Fields	Points	Type, e.g., soccer field, barren area
Intersections	Points	Type, e.g., a roundabout, intersection, crossing
Roads	Lines	Type, e.g., primary, tertiary, trail
		Surface type, e.g., gravel, concrete, asphalt

A consideration is the availability of source geodata; a terrain database can hypothetically be made as large and detailed as necessary. However, realistically, there is always a limit on the amount, quality and accuracy of source data such as satellite imagery, elevation data, or vector data, that is available (due to time, budget and availability of sensors and analysts). For virtual trainings, it is essential to (automatically) derive both the terrain reasoning database and the 3D representation in the simulator(s) from the same processed dataset. For live, outdoor training, some manual effort might be required to fill in missing information or to correct recently changed features from the real-world training site.A final requirement is that use-case / scenario specific data should be present within the terrain reasoning database. For our use-case, this includes several named locations that are known beforehand by the pilot and JTAC.

Terrain Reasoning

Various queries made up of filters and selection methods are available to be used by the agent. The basis of the terrain reasoning module are simple, straightforward queries that are common to terrain databases, e.g., give me all two-story residential buildings in this area or direction. To support the more complex and fuzzy reasoning that is needed for a talk-on procedure in the CAS use-case, we extended this with the two following principles:

1. Queries are generally considered to build on the result of the previous query. In other words, queries transfer the attention of the virtual unit from one terrain feature to the next. The starting feature will be referred to as the *origin*, and the feature of interest is referred to as the *target*.
2. Since we are dealing with natural language descriptions of target features instead of precise coordinates, there is not always a 100% match for distinct target feature. Instead, the terrain reasoning module returns a *heatmap* with potential target features and their likelihoods. To ensure consistency, heatmaps are normalized such that the likelihoods of all features in the heatmap sum to 1.

These heatmaps are generated based on the input query by multiplying the likelihoods belonging to the individual properties. For example, consider the query 'the building north of the soccer field within a distance of 1km'. The origin is given by the soccer field, and three properties describe the likelihoods of the potential targets: the type must be *building*, the distance vector between the origin and target must point towards the north, and the distance must be less than 1km. The resulting heatmap is the multiplication of an indicator function for the type *building*, a function that decreases linearly with the angle of the distance vector, and a function that decreases linearly with the distance. The heatmap of this example is displayed in Figure 3.

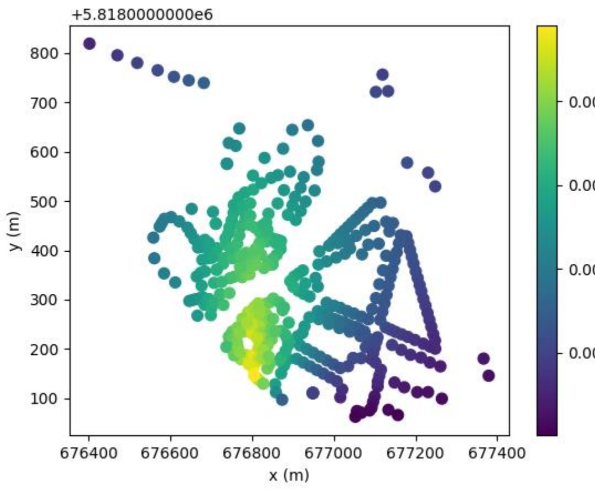


Figure 3. Heatmap Generated by Querying the Database for All Buildings North of the Soccer Field.

Uncertainty may also be present in the origin of a query. When that is the case, the origin is represented by a heatmap instead of a single feature. To still be able to generate a single output heatmap, heatmaps are generated using all features in the origin heatmap individually, and summing the results using the likelihoods in the origin heatmap as weights.

Some examples of queries that can be performed by the agent:

- All features within a certain distance of the origin.
- All buildings with red roofs close to the origin.
- All flat-roof buildings about 6m high north-west of the origin within 50m of the origin.
- The middle bridge of all bridges within 100m north of the origin road intersection. Middle is a specialized selector where the target feature with lowest cumulative distance to other target features has the highest probability.
- The second roundabout following the origin road north. Note that it is not always clear which road segments are a continuation of the origin road and there might be several semi-parallel roads nearby leading north. A specialized algorithm computes the probabilities of target roundabouts in this case, following all road segments that match distance and direction thresholds.
- The closest building next to the bend in the origin road. This uses a specialized bend-detection algorithm to determine whether a line segment is straight or bendy, and in the latter case, how many bends are present and what their properties are, i.e. in what direction they turn and how much. It is implemented as a greedy algorithm that starts at any point with a local maximum in the curvature, and expands the bend in both directions until either the curvature drops below some threshold, or the curvature switches direction.

AGENT BEHAVIOR MODULE

In order to reason and act on the current state of the world, the agent needs to be able to make decisions given a certain state. We opted to employ the use of behavior trees, as they have been proven successful in modeling the behavior of non-player characters (NPCs) (Agis et al., 2020; Lim & Colton, 2010). Behavior trees have further shown to be well suited for developing behavior models for semi-automated forces (SAF) in constructive simulations (Evensen et al., 2019). Besides game AI, they have also shown useful in other control dependent systems (Bagnell et al., 2012; Klöckner, 2013; Ogren, 2012).

A behavior tree is a directed rooted tree (Colledanchise & Ogren, 2018). For each pair of connected nodes, one is called the parent and the node it connects toward is called the child. The root of the tree starts the execution of the tree by sending a tick to its children. These ticks are released with a given frequency, usually many per second. A tick activates a node, and a node is only executed when it received a tick. When a node receives a tick, it immediately returns a status to its parent. This status can either be running, success, or failure. The children are ticked in a certain order, depending on your visualization this could either be from left to right, or from top to bottom. An example of a behavior tree as used in our use-case is shown in Figure 4.

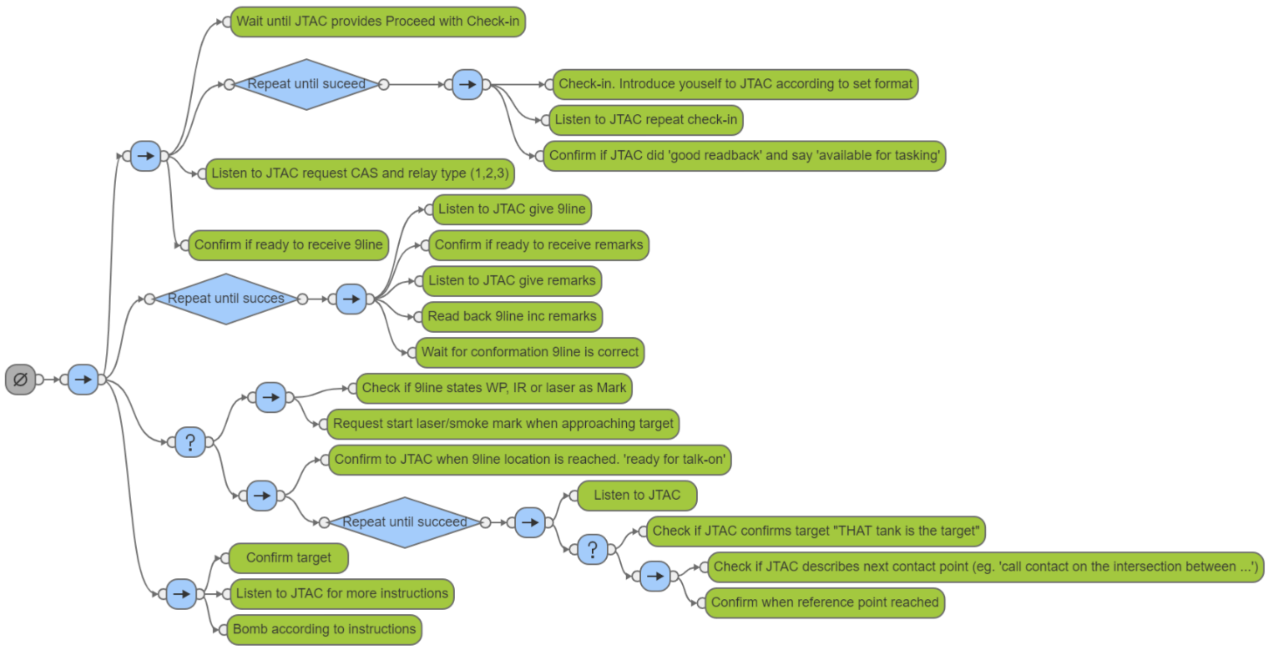

Wait until JTAC provides Proceed with Check-in
Repeat until suceed
Check-in. Introduce youself to JTAC according to set format
Listen to JTAC repeat check-in
Confirm if JTAC did 'good readback' and say 'available for tasking'
Listen to JTAC request CAS and relay type (1,2,3)
Listen to JTAC give 9line
Confirm if ready to receive 9line
Confirm if ready to receive remarks
Repeat until succes
Listen to JTAC give remarks
Read back 9line inc remarks
Wait for conformation 9line is correct
Check if 9line states WP, IR or laser as Mark
Request start laser/smoke mark when approaching target
?
Confirm to JTAC when 9line location is reached. 'ready for talk-on'
Listen to JTAC
Repeat until succeed
Check if JTAC confirms target "THAT tank is the target"
?
Check if JTAC describes next contact point (eg. 'call contact on the intersection between ...')
Confirm target
Confirm when reference point reached
Listen to JTAC for more instructions
Bomb according to instructions

Figure 4: The Behavior Tree Describing the Behavior of a F16 in Our Use-Case.

Behavior trees have internal nodes and leaf nodes. These leaf nodes are called execution nodes and describe actions or conditions, in Figure 4 these are green. The internal nodes are called control flow nodes and determine how to execute their child nodes. In our use-case we use Sequence nodes, Selector nodes, and Decorator nodes as control flow nodes. A Sequence node sends ticks to its children, and is visualized in Figure 4 as an blue node with an arrow in it. Once it finds a child that returns either the status *failure* or *running* it relays this status to its parent and waits until it receives the next tick from its parent. A Sequence node returns the status *success* if and only if all its children return the status *success*. The Selector node is visualized in Figure 4 in blue with a 'question mark' symbol. It also ticks its children, and once it finds a child that returns either the status *success* or *running*, it relays this status to its parent. A Selector node returns the status *failure* if and only if all its children return the status *failure*. Decorator nodes are control flow nodes that have a single child. They are visualized in Figure 4 as blue diamond shaped nodes. The Decorator node manipulates how the child is ticked, or how the child's status is returned, or both. In our use-case we use the Until node, which keeps ticking its child until a certain condition is met.

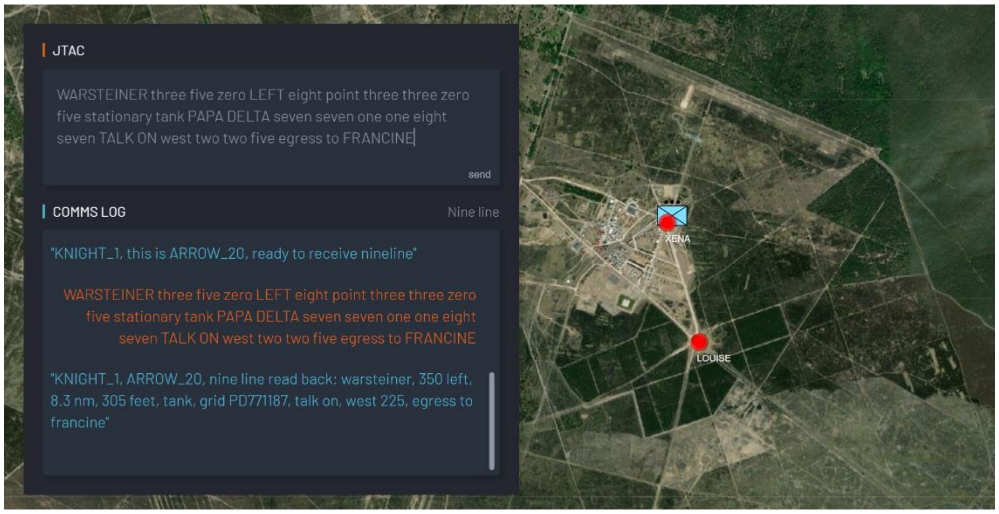
Internally, the behavior tree checks certain conditions that affect which nodes gets ticked next. These conditions are mostly about understanding events and environment. Before being able to check if a certain condition is met, it is necessary to correctly interpret events and environment. In general, the interpretation is very simple and mostly consists of interpreting text into other data formats. The communication module does most of this, for example by returning a dictionary with key-value pairs after getting a nine-line as input. The interpreter then makes sure that the text "PAPA DELTA seven six zero one eight seven" gets interpreted as the MGRS coordinate '32U**PD760**00**187**00'. In bold we highlighted the information given in the spoken communication, whereas the other information in this MGRS location is implied.

A specific case that requires interpretation is the part of our system that handles free-form text, and is used for the talk-on. The communication module extracts different types of objects and (related) characteristics from a text as well as a 'context', which expresses the conversational goal of a given sentence. These can then serve as input for the heatmap queries in the terrain module. This requires a smart rule-based combination of input text into the actual terrain queries, that consists of filters and selectors. The interpreter takes care of the translation necessary between communication module output and terrain module input.

The behavior tree is also in charge of constructing proper output communications. Most of the time there are limited options, but the sentences usually require some case-specific info, which can be extracted from the internal knowledge representation of the behavior tree.

DEMONSTRATOR

As part of our a research a demonstrator was build that implements our framework. The demonstrator walks the user through a full CAS operation use-case. It is also able to demonstrate our technology in a more isolated sandbox environment. The demonstrator use our framework written in Python 3.7 as a backend with a web interface written in Javascript with HTML5 and CSS3 markup as a frontend. Leaflet was used as a mapping library. Communication between the backend and the frontend was done via websockets.
Figure 5 gives an overview of the demonstrator displaying the CAS use-case. The input field below *JTAC* provides an input console for the user. Note that input sentences are formatted as a speech-to-text processor would translate human speech. For example, numbers are typed out. Our system does not infer any information on capitalization or punctuation. As this research does not focus on speech-to-text transcription, this step was omitted from the



WARSTEINER three five zero LEFT eight point three three zero five stationary tank PAPA DELTA seven seven one one eight seven TALK ON west two two five egress to FRANCINE

"KNIGHT_1, this is ARROW_20, ready to receive nineline"

WARSTEINER three five zero LEFT eight point three three zero five stationary tank PAPA DELTA seven seven one one eight seven TALK ON west two two five egress to FRANCINE

"KNIGHT_1, ARROW_20, nine line read back: warsteiner, 350 left, 8.3 nm, 305 feet, tank, grid PD771187, talk on, west 225, egress to francine"

Figure 5. A Screen Capture of the Demonstrator Showing Input for a JTAC and Replies From the Constructive Fighter Pilot.

demonstrator. However, the sentences generated by the synthetically wrapped unit are emitted as both text on screen as well as text-to-speech audio clips. The *COMMS LOG* shows the input from the JTAC (in red) and the replies of our constructive fighter pilot (in blue) as well as the current context (*nine-line* in the figure). The background shows a map with registered named locations and the location of the JTAC. For our use-case, this is the area of Altmark (Germany), which contains a large military training site.

The demonstrator goes through a full scenario of a JTAC calling in close-air-support for a target vehicle in a village. The scenario goes through the check-in, nine-line and talk-on steps as described in the introduction. During the nine-line the system denotes details about the attack and checks which area is relevant for the following talk-on. During the talk-on phase the system queries the terrain module for possible points of interests (POI) described by the JTAC. Figure 7 shows and example of the heatmap of possible POI based on the input from the JTAC. When the specific POI has been determined by the system this then becomes a relative POI serving as an anchor for the next POI. This process repeats until a final target has been determined. Next to terrain features, manual named locations can also be given. The demonstrator defines two of these named locations; *XENA* and *LOUISE* (see Figure 5).

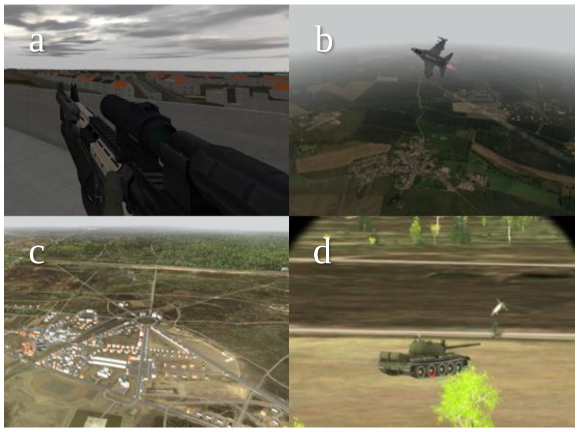
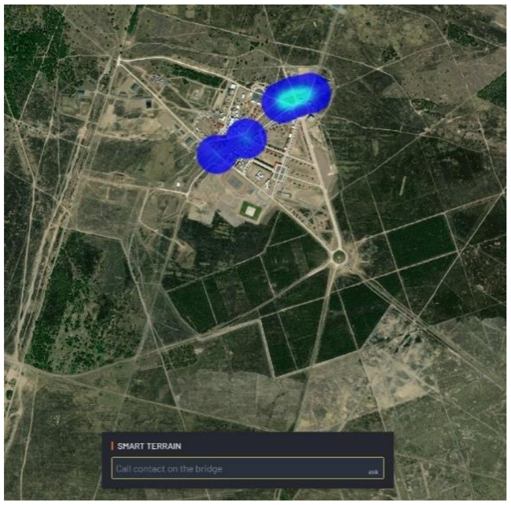


Figure 6. VBS Coupled With the Framework Showing the CAS Scenario in Action: (a) JTAC, (b) Fighter Jet, (c) Location Altmark, (d) Target.

During this process a connection with VBS is made and the state of the world is updated. Actions selected by the agent's behavior tree are translated to VBS parameters and SQF scripts, which are communicated and executed in the running VBS simulation using an API gateway connection (e.g., a bombing run).

In order to test and demonstrate the capabilities of our framework without having to run through the full CAS scenario we also added an option to run the demonstrator without the CAS use-case. This option works in an isolated setting and does not implement a constructive unit using the behavior tree module. Instead it focusses on sentence parsing and terrain querying. Figure 7 gives an overview of this option in action where a request using natural language can be made, which results in a heatmap displaying the probability space of the requested target.



Call contact on the bridge

Figure 7. A Screen Capture of the Demonstrator Showing the Heatmap Generated for All Possible Requested Bridges.

CONCLUSION & FUTURE WORK

The goal of this research was to explore different artificial intelligence techniques in order to create perceptually realistic synthetically wrapped units. Three domains were considered, namely communication via natural language, understanding the environment, and adaptive behavior. A framework was developed to implement a representative use-case: a JTAC training for CAS.

Behavior trees provide a modular and composable way of crafting behavior models. This makes them well suited for generating behavior for constructive units. For our use-case, behavior trees allowed us to quickly and iteratively refine and add on behaviors to improve the quality of the constructive unit's behavior. However, to implement behavior trees for different types of units, a database of basic behavioral models (e.g., drills) would be needed. Once such a database has been established, different types of constructive units could quickly be realized by composing more complex behavioral models using basic models. One should also allocate some time into defining the correct implementation of the actions included in the behavior tree, since those handle the actual execution of the intended behavior.

The two different parsers implemented in the communication module, allowed for the extraction of information from both protocolized and (more) freely formatted texts. For our use-case, the implementation generated reliable results that could be used for further processing in the behavior tree. However, analysis of unexpected results showed that most extraction mistakes were made due to the heatmap relying too heavily on the expected relative lengths and positions of the elements. An alternative approach should be considered in order to overcome this problem, while at the same time allowing for non-protocolized sentence elements within a text. Furthermore, when a bigger range of contexts is required, intent classifiers such as DeepPavlov could be used in order to improve the free format parser further.

To provide the synthetically wrapped unit with spatial awareness, a terrain reasoning module was developed. This module consists of an underlying terrain database which contains a list of objects and their visual properties, and several filters with which this database can be queried. Which filters should be used is decided based on the information extracted by the communication module. Combining these filters with the results of previous queries results in a synthetic unit that appears to understand its environment. To further improve the perceptual realism, uncertainty was incorporated into the module by making queries return heatmaps instead of single features.

Currently a subset of possible filters to query the terrain database were developed. These were fitted to support our use-case. However, to further bridge the gap between real and synthetic units, more complex filters could be developed. In particular, filters that recognize patterns in the distributions of objects are a prime candidate for future research. For example, being able to recognize clusters of buildings, trees that form a circle, or a triangular-shaped city, would further expand the capabilities of constructive units.

Future research might focus onto these specific modules as each can be finetuned to the military domain. Furthermore other use-cases might be considered. For example, infantry that have to move towards a specific target. These units have a more direct connection and interaction with their environment as opposed to a fighter jet.

Overall our results give us confidence that modern AI techniques can aid in the creation of perceptually realistic simulated units that allow for believable interactions and behaviors during military training and exercises. The proposed framework worked well for our use-case and provided perceptually realistic interaction with the user. As the interaction between a JTAC and the constructive pilot is mainly done via speech over radio, achieving perceptually realistic communication was the main aspect we focused on. Since requesting and aiding CAS is heavily protocolized and depends on clear and distinct communication, the scope our models needed to work in was clearly defined. This helped narrow down the possible types of interactions a constructive agent could have. Similarly, possible relevant entities and contexts from parsed sentences is limited to the operational environment and therefore known beforehand. This simplified the task of extracting entities from sentences. Depending on the use-case and the type of interactions that need to be supported, the complexity of each of the three modules may vary quite a bit.

Synthetic wrapping offers a promising way of incorporating virtual and constructive units into large-scale military exercises. Combined with modern artificial intelligence techniques, these allow constructive units to behave more perceptually realistic resulting in possible improvements to learning goals and maintain a level of preparedness.

REFERENCES

Muller, T. J., van den Bosch, K., Kerbusch, P., & Freulings, J. (2011). *LVC training in urban operation skills* (pp. 115-120). Simulation Interoperability Standards Organizations (SISO).

DEPARTAMENT, O. D. (2016). *Modeling and Simulation (M&S) Glossary*. Tech. Rep. 1, DOD, http://goo.gl/dSZeeW (oct. 2011).

Reitz, E. A., Seavey, K., & Mullins, M. (2018). Bridging the Joint Close Air Support Training Gap. *Proceedings of I/ITSEC 2018. Orlando, FL*.

Seavey, K., Reitz, E. A., Hanne, C. F., & Mont de Marsan, F. (2019) Digitally-Aided Close Air Support Capabilities in Simulation: Lessons Learned from a France-US Effort.

Van den Bosch, K., & Boonekamp, R. (2014). Virtual Pilot: Agent-Based Simulations for Effective Training. *Proceedings of the NATO MSG Symposium on Advanced Technologies for Military Training, Exploiting Commercial Technologies and Games for Use in NATO (in the Maritime Domain), Italy*. https://doi.org/10.14339/sto-mp-msg-130-16-pdf

Bohemia Interactive Simulations. (2022). VBS4 | Retrieved May 9, 2022, from https://bisimulations.com/products/vbs4/

Emond, B., Kondratova, I., Durand, G., & Valdés, J. J. A multi-role reconfigurable trainer for naval combat information operators. In *Interservice/Industry Training, Simulation and Education Conference (I/ITSEC)* (p. 14).

Tanaka, A., Stensrud, B., Welch, G., Guido-Sanz, F., Guido, F., Sciarini, L. L., & Phillips, C. H. (2019). The development and implementation of speech understanding for medical handoff training. In *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC), Orlando, FL*.

SeatGeek Inc. (2014). fuzzywuzzy: Fuzzy String Matching in Python. Retrieved from https://github.com/seatgeek/fuzzywuzzy

Loper, E., & Bird, S. (2002). Nltk: The natural language toolkit. *arXiv preprint cs/0205028*.

Honnibal, M., & Montani, I. (2017). spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. *To appear*, *7*(1), 411-420.

Burtsev, M. S., Seliverstov, A. V., Airapetyan, R., Arkhipov, M., Baymurzina, D., Bushkov, N., ... & Zaynutdinov, M. (2018, July). DeepPavlov: Open-Source Library for Dialogue Systems. In *ACL (4)* (pp. 122-127).

Agis, R. A., Gottifredi, S., & García, A. J. (2020). An event-driven behavior trees extension to facilitate non-player multi-agent coordination in video games. *Expert Systems with Applications*, *155*, 113457.

Lim, C. U., Baumgarten, R., & Colton, S. (2010, April). Evolving behaviour trees for the commercial game DEFCON. In *European conference on the applications of evolutionary computation* (pp. 100-110). Springer, Berlin, Heidelberg.

Evensen, P. I., Stien, H., & Bentsen, D. H. (2019). Using behaviour trees to model battle drills for computer-generated forces. *NATO Science and Technology Organization STO-MP-MSG*, *171*, 01.

Bagnell, J. A., Cavalcanti, F., Cui, L., Galluzzo, T., Hebert, M., Kazemi, M., ... & Zhu, R. (2012, October). An integrated system for autonomous robotics manipulation. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 2955-2962). IEEE.

Klöckner, A. (2013). Behavior trees for UAV mission management. *INFORMATIK 2013: informatik angepasst an Mensch, Organisation und Umwelt*, 57-68.

Ogren, P. (2012, August). Increasing modularity of UAV control systems using computer game behavior trees. In *Aiaa guidance, navigation, and control conference* (p. 4458).

Colledanchise, M., & Ögren, P. (2018). *Behavior trees in robotics and AI: An introduction*. CRC Press.