

UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Data Science Curriculum

Karl-Kristjan Kõverik

NATO standard mission task symbols pose detection based on symbol requirements

Master's Thesis (15 ECTS)

Supervisor(s): Ardi Tampuu, PhD

Tartu 2023

NATO standard mission task symbols pose detection based on symbol requirements

Abstract:

The objective of this master's thesis is to develop a series of neural network models that can accurately detect the pose of NATO mission task symbols. This research is significant as NATO mission task symbols play a critical role in planning military operations and automatically identifying their identity, location and pose can save valuable time and resources when digitizing military plans.

The first step of research involved determining the essential components required to recreate mission task symbols into the KOLT digital planning system. This required a thorough analysis of the mission task symbols and their underlying characteristics. Through this analysis, we were able to identify the crucial features that the models must detect to sufficiently characterize each mission task symbol.

The second task of research involves developing models that can detect the rotations of the mission task symbols accurately. Additionally, for trajectory symbols, it is essential to develop models that can detect and describe the symbols' trajectory accurately. However, detecting poses of hand-drawn symbols is challenging, as there is often significant variability among symbols with the same meaning. Therefore, the model must be designed to account for this variability.

The success of research will have significant implications for military operations. Accurate and efficient recognition of the poses of mission task symbols complements YOLO-based localization that is concurrently implemented in the research group, enabling AI-based digitalization of military plans. Fast digitalization enhances situational awareness and decision-making capabilities.

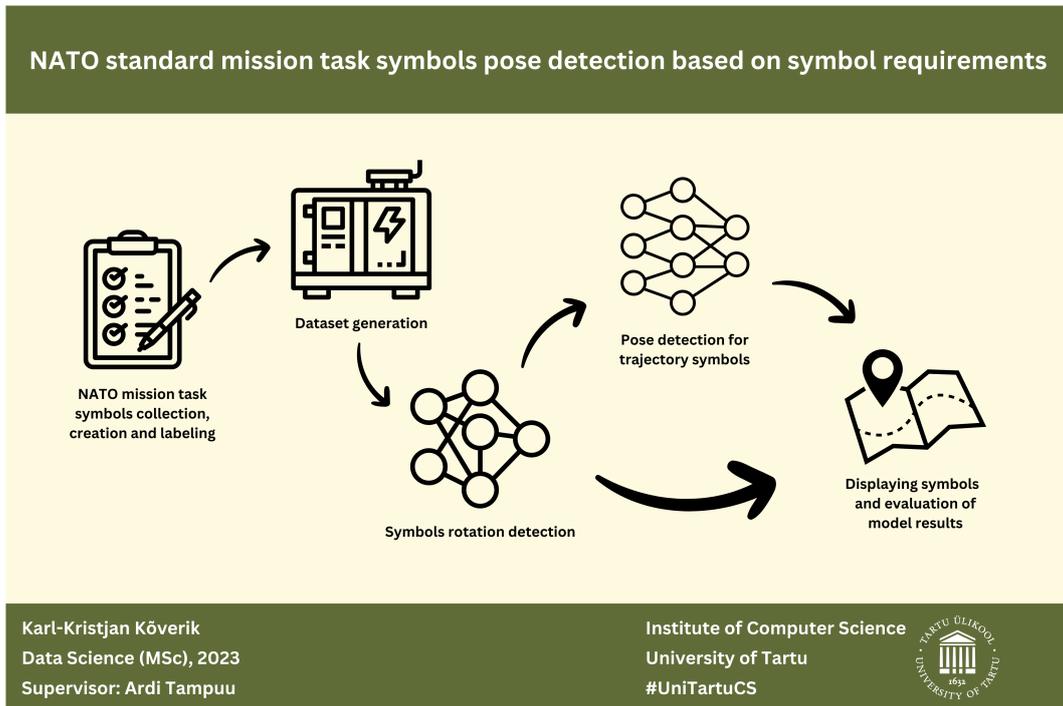
Keywords:

Convolutional neural network, machine learning, artificial neural network

CERCS:

P176 Artificial intelligence

Visual abstract:



NATO standardsete lahingutoimingu sümbolite asendi tuvastus sümboli nõuete alusel

Lühikokkuvõte:

Magistritöö eesmärk on arendada närvivõrgu mudeleid, mis suudavad täpselt tuvastada NATO lahingutoimingu sümbolite asendit. See uurimistöö on oluline, kuna NATO lahingutoimingu sümbolid mängivad olulist rolli sõjaliste operatsioonide planeerimisel ja nende identiteedi, asukoha ja asendi automaatne tuvastamine säästab digitaalsete sõjaliste plaanide loomisel väärtuslikku aega ja ressursse.

Uurimistöö esimene samm hõlmas oluliste komponentide määramist, mis on vajalikud lahingutoimingu sümbolite rekonstrueerimiseks KOLT digitaalsesse planeerimissüsteemi. Selleks oli vaja põhjalikku analüüsi lahingutoimingu sümbolitest ja nende omadustest. Selle analüüsi abil suutsime tuvastada olulised omadused, mida mudelid peavad tuvastama, et iga lahingutoimingu sümbolit piisavalt iseloomustada.

Uurimistöö teine ülesanne hõlmab mudelite arendamist, mis suudavad täpselt tuvastada lahingutoimingu sümbolite rotatsiooni. Lisaks on trajektoori sümbolite puhul oluline arendada mudeleid, mis suudavad trajektoori sümbolid täpselt tuvastada ja kirjeldada. Siiski on käsitsi joonistatud sümbolite asendi tuvastamine keeruline, kuna sama tähendusega sümbolitel on sageli märkimisväärne varieeruvus. Seetõttu peab mudel olema kavandatud selle varieeruvuse arvestamiseks.

Uurimistöö edul on sõjalistele operatsioonidele märkimisväärne mõju. Lahingutoimingu sümbolite asendi täpne ja tõhus tuvastamine täiendab YOLO-põhist lokaliseerimist, mida uurimisrühmas samaaegselt rakendatakse, võimaldades tehisintellektil põhinevat sõjaliste plaanide digitaliseerimist. Kiire digitaalseerimine suurendab oluliselt olukorra teadlikkust ja otsustusvõimekust.

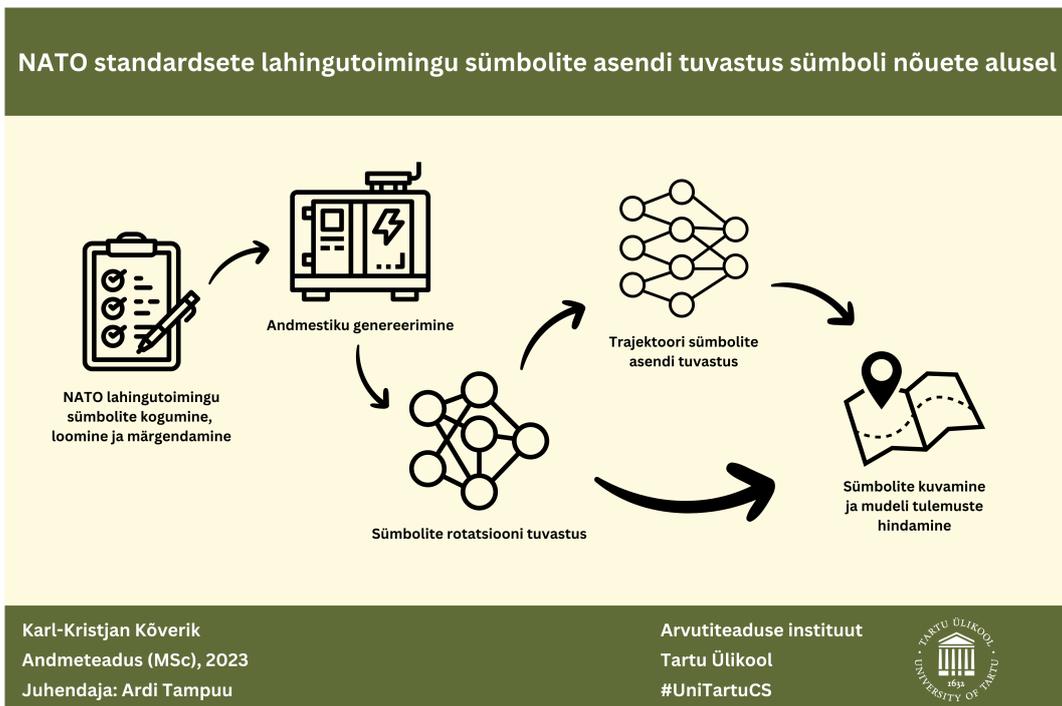
Võtmesõnad:

Konvolutsiooniline närvivõrk, masinõpe, tehisnärvivõrk

CERCS:

P176 Tehisintellekt

Visuaalne kokkuvõte:



Contents

1	Introduction	8
2	Background	12
2.1	KOLT	12
2.2	Computer vision	13
2.2.1	Symbol detection	13
2.2.2	Dealing with handwritten symbols	14
2.2.3	Using YOLO to locate and classify symbols	14
2.2.4	Rotation detection	15
2.2.5	Pose detection	16
2.3	Overview of the collaboration project	18
3	Methodology	20
3.1	Determining requirements	20
3.2	Data used	21
3.2.1	Rotation models data	24
3.2.2	Pose models data	26
3.2.3	Pose base model dataset	26
3.2.4	Pose autoencoder dataset	27
3.3	Methods used	28
3.3.1	Rotation detection method	28
3.3.2	Pose detection methods	31
3.3.3	Pose base model method	31
3.3.4	Pose autoencoder method	33
3.4	Language model to improve readability	37
4	Results	38
4.1	Set of requirements for each symbol type	38
4.2	Rotation detection results	40
4.3	Pose detection results	44
4.3.1	Base model results	44
4.3.2	Beta variational autoencoder model results	48
5	Discussion	54
6	Conclusions	56
	References	57

Appendix **60**
I. Requirements table 60
II. Licence 66

1 Introduction

Over the years, computer vision has made significant strides in its capabilities. Its wide range of applications in various fields [KST⁺21, DKCA21, GYLP18, ZZK23] make it a powerful technology with enormous potential for real-world impact. The topic of the master's thesis emerged from a collaborative project between the University of Tartu and the Estonian National Defence College, demonstrating the diverse and interdisciplinary nature of this exciting field. At present, deployment plans for the Estonian Defence Forces are manually added to KOLT, their situational and combat awareness system (in Estonian Kaitseväe olukorra ja lahinguteadlikkuse süsteem). However, incorporating computer vision technology to automatically locate and characterize mission task symbols could greatly improve the efficiency and accuracy of the digitization process. Automating and digitizing these activities is crucial for the Estonian Defence Forces and the command chain as a whole, making their procedures faster, more convenient, and better suited to the demands of modern warfare. Quick decisions are essential for effective communication between units and the success of military operations, making the adoption of computer vision technology a high priority for enhancing the command and control capabilities of the Estonian Defense Forces.

Currently, military personnel at different levels create military plans on transparent plastic films and then send them by military personnel to the headquarters. The plans are manually aligned with the map as the films have markings that allow to realign them with the map. Only then can the mission task symbols be drawn into KOLT. However, this process is extremely time-consuming (up to few minutes per symbol), taking up valuable resources that could be better used elsewhere. KOLT is a program that allows military personnel to select mission task symbols from a menu and place them on a map with different sizes and poses, providing a comprehensive overview of the battlefield for higher-ups to understand. Once these symbols are digitized in KOLT, they are permanently saved, eliminating the need to repeat the alignment and entering process. To improve the efficiency of this process, the Estonian National Defence College and Institute of Computer Science of University of Tartu are exploring computer vision technologies to automate some aspects of the digitization process. By leveraging the power of computer vision, it is possible to significantly reduce the time and effort required to create and update deployment plans, allowing military personnel to focus on other critical tasks.

It is crucial to note that when it comes to mission task symbols, knowing their location is not enough - their orientation is just as critical. Rotational information is key to accurately interpreting mission task symbols, indicating the direction of actions or expected threats. Additionally, there are symbols that illustrate a trajectory for which it is important to determine their shape in order to properly understand their meaning.

If computer vision algorithms can accurately estimate the rotation, poses, as well as locations and identities of mission task symbols, this enables the automated creation and updating of deployment plans with greater efficiency and accuracy. To complete the task at hand the following steps needed to be done are:

- Digitizing the films
- Relating every pixel of the image to digital map coordinates
- Detecting symbol locations and identities
- Detecting symbol rotations and poses

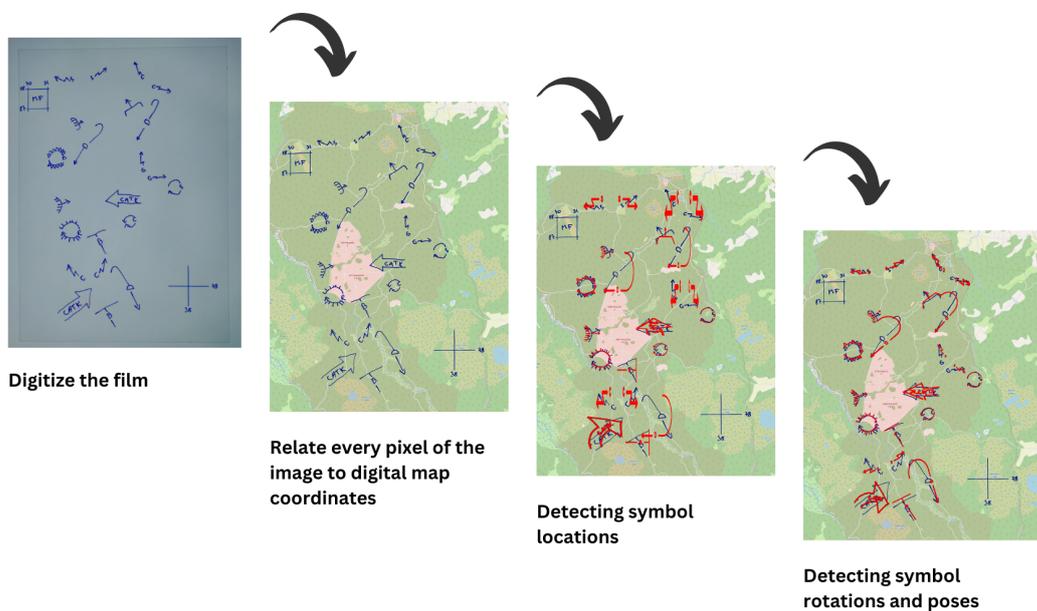


Figure 1. The steps to complete the task.

This thesis aims to make several key contributions to the field of computer vision for detecting mission task symbols. Firstly, in collaboration with the Estonian National Defence College, the precise requirements for the automatic mission task symbol detection were determined. Essentially, we need to determine the information that needs to be extracted from each symbol to ensure it is displayed accurately in KOLT. Interpreting mission

task symbols in a military context is a crucial task that requires careful consideration of various factors. One important factor is determining whether the symbol's meaning is affected by rotation or trajectory. Rotation refers to the direction the symbol is facing, while trajectory refers to the path it is moving along. However, there may be other factors that also affect the symbol's meaning, making interpretation challenging at times. It is important to keep in mind that interpreting mission task symbols may require additional context or information. By taking into account these various factors, we can gain a better understanding of the military situation and recreate mission task symbols in KOLT with sufficient accuracy i.e. without losing their military meaning.

Secondly, once the requirements have been established, symbols can be grouped according to their specific needs. This will help streamline the symbol recognition process. For example, the requirement analysis may identify symbols for which pose does not matter at all, for which only rotation matters and for which both rotation and trajectory matter. The amount of information we need to extract and hence the number of processing steps will differ for these three groups. Knowing these symbol groups allows us to choose and to prioritize simpler symbol types, exclude some symbols and in general be more aware of the challenges.

Thirdly, based on the documented requirements, prototypes for predicting symbol rotations and trajectories will be developed. Statistical analysis can then be used to evaluate the accuracy of the models, with any inaccuracies being identified and addressed. For symbols that cannot be accurately described by the prototypes, alternative approaches will be listed for future development.

To begin this project, the Estonian National Defence College provided an initial set of 15 NATO mission task symbols, which were used to develop initial prototype solutions for the computer vision system. Through ongoing communication with the college personnel, it became apparent that a broader set of symbols was needed to fully capture the range of combat operations used in practice. As a result, this work will ultimately include 31 different mission task symbols, covering a wider range of military operations. Choosing the symbols to include in this second phase was led by the author and was done in communication with the Estonian National Defence College, to include symbols that are frequently used at the company-sized and smaller military units. By developing solutions for each of these symbols, the computer vision system can provide comprehensive and accurate deployment planning support for a broad range of military scenarios.

The thesis will be structured as follows:

- Background chapter will provide a background on KOLT and its use in the military, as well as an overview of different machine learning technologies.

- The methodology chapter presents the methodology used for collecting and processing data, as well as a detailed description of the methods used to develop the prototype.
- Results chapter present the results obtained, including accuracy and validation metrics, and will provide an analysis of how well the prototype models perform.
- Discussion chapter will be dedicated to a discussion of the research findings, including a critical evaluation of the methods used, and an explanation of how the findings contribute to the field of study.
- Conclusion chapter will summarize the main contributions of the thesis and provide some directions for future research.

2 Background

This chapter aims to provide the reader with the necessary terminology and practical knowledge related to detecting mission task symbols, which will facilitate their comprehension of the subsequent chapters of this thesis. We begin with an overview of KOLT, the defense force situational and combat awareness system. Next, we provide an introduction to computer vision methodologies, which will be used to develop the prototype for detecting the poses of mission task symbols. Followed by a description of the collaboration project and its objectives.

2.1 KOLT

KOLT (Kaitseväe olukorra ja lahinguteadlikkuse süsteem in Estonian) is the Estonian Defence Force's situational and combat awareness system. KOLT is one of the most important tangible achievements born from the cooperation of conscripts, reserve and active military personnel. It is a comprehensive IT system designed to provide commanders with up-to-date and accurate information about the battlefield in a digital form, enabling them to make informed decisions. With KOLT, commanders can quickly assess the situation on a digital map that can span large areas while allowing to zoom in for details, allowing them to plan and execute operations with greater efficiency and effectiveness. The system also enables better coordination between different units as the digital plan can be shared and enhances communication between lower rank officers in the field and their commanders. Overall, KOLT is a critical tool for the Estonian Defence Force in ensuring situational awareness and effective decision-making on the battlefield.

Currently, military plans are created by hand on transparent film by military personnel on the battlefield. Once the plan is complete, someone has to physically transport the film to the headquarters, which can take a long time if the plan was created far away from headquarters. When the film arrives, someone has to manually redraw the symbols into the KOLT system, which is a time-consuming and laborious process. The person doing this has to align the film with a real paper map to determine the geographical locations of symbols and then manually enter each symbol on the digital map as accurately as possible, which can result in location errors.

By automating and digitizing the process, the entire process can be made faster and more spatially precise. Instead of physically transporting the film, a picture can be taken and automatically sent to headquarters. With the help of computer vision, it will hopefully be possible to take those photos of films and automatically detect the symbols and their information, including their rotation and pose. After all the information is gathered about the content of the films it would be possible to add that information automatically into

KOLT. As the process becomes more and more automated, it would greatly reduce the chance of human error. This means that the entire process would be quicker and more accurate, and would ultimately lead to better and more timely decision-making on the battlefield.

2.2 Computer vision

Computer vision is a subset of artificial intelligence (AI) that aims to mimic human visual perception and processing. Computer vision is a field of study focused on developing algorithms and technologies that enable computers to interpret, analyze, and understand visual data. Computer vision is using various hardware and software technologies, such as cameras, sensors, image processing algorithms and machine learning models. Computer vision has gained wider adoption over the years, finding application in various fields including industrial manufacturing, where it is used for quality control and inspection[ZZK23]; healthcare, where it is used for diagnosing symptoms[GYP18]; and transportation, where it is used in autonomous vehicles and drones for navigation and obstacle avoidance[KST⁺21, DKCA21]. Overall, Computer vision has numerous applications in diverse industries, and its adoption is rapidly expanding as technology advances and new use cases emerge.

2.2.1 Symbol detection

Object detection is a computer vision task that involves identifying objects. The main goal in object detection is to accurately detect and locate objects in an image or a video.

There are several popular datasets that have been used for training and evaluation of object detection models. To name a few are COCO[LMB⁺14] and PASCAL VOC[EEVG⁺15]. COCO (Common Objects in Context) is a large-scale dataset that contains over 330,000 images and more than 2.5 million object instances across 80 different object categories. It is widely used for training state-of-the-art object detection models, and there were annual COCO challenges, that evaluated the performance of different models on the dataset. PASCAL VOC (Visual Object Classes) is another popular object detection dataset, containing images with 20 object categories, such as person, car, and airplane. The dataset was used in an annual competition called the PASCAL VOC Challenge, which evaluated object detection performance on the dataset. Both mentioned datasets contain natural images. This data type is visually very different from the military plans we work with. Natural images are photographs or digital images that depict real-world scenes or objects, such as landscapes, people, animals, or buildings. These images are usually captured by cameras or generated by computer graphics. We deal with binary

images of symbols, not natural images. Binary images of symbols are digital images that consist of only two colors, typically black and white, and represent graphical symbols or characters, such as letters, numbers, or shapes. These images are also widely studied in computer vision, optical character recognition, or machine learning applications, where the goal is to recognize or classify the symbols based on their visual features[CWF⁺15].

There have been similar use cases to detect and classify symbols. Examples of similar tasks are detecting and classifying engineering drawing symbols[EJAG20], automatic floor plan analyzing and recognition[PHSS22] and analyzing the engineering and production drawings[PHSS22]. However in our case we deal with the handwritten symbols, which makes the task more difficult.

2.2.2 Dealing with handwritten symbols

Detecting handwritten symbols is a complex task that poses several challenges when using computer vision. The variability in handwriting makes detecting handwritten symbols much harder than printed symbols, as handwriting varies widely between individuals, and even the same person may write the same symbol differently each time. This variability makes it challenging to create a model that is robust enough to accurately recognize symbols of various styles. Handwritten symbols can also be ambiguous, making it difficult to distinguish one handwritten symbol from another, especially without using or without understanding the context. While models trained on large MNIST handwritten datasets have shown significant improvements in handwritten digit detection accuracy, they still struggle to achieve perfect accuracy on even slightly different data. For instance, in a recent study [GGSS19], the authors trained a deep learning model on the MNIST dataset and achieved a validation accuracy of 98.45% on the testing set. The training dataset consisted of 40000 images and the test dataset consisted of 2000 images. However, when tested on 300 similar random images from the internet, the same model achieved an accuracy of only 68.57%. The only route to achieve better generalization and deployment performance seems to expose the model to more and more handwriting variability during the training.

2.2.3 Using YOLO to locate and classify symbols

You only look once (YOLO)[AC11] is the most popular object detection algorithm that is used to detect and recognize objects. YOLO uses a convolutional neural network (CNN) architecture. The mentioned architecture has several layers to analyze an image and to detect the objects. One key innovation that sets YOLO apart from other object detection algorithms is its ability to perform localization and identification in a single step, rather

than separating the two tasks. This approach, made possible by YOLO's unique per-class processing, enhances both the speed and accuracy of the algorithm. YOLO's efficiency stems from its use of a single convolutional neural network (CNN) to detect objects in an image. The per-class processing feature of YOLO allows the algorithm to process each class of object separately, which leads to higher accuracy and a reduction in false positives.

In the collaboration project with the Estonian National Defence College for automatic digitization of task symbols on military plans, YOLO is used for extracting identity and location of symbols.

2.2.4 Rotation detection

In computer vision, detecting symbol rotation involves defining a zero rotation and then creating an algorithm that can determine how much the symbol in a given image is rotated with respect to this zero angle. This can be a challenging task since symbols are hand-drawn and can be rotated in 360 degrees, resulting in a large variability.

One method to detect symbol rotation would be to modify YOLO and add a rotation detection output in addition to localizing and detecting symbols[YPS⁺18]. However modifying YOLO can be time-consuming since it is already a complex algorithm. The YOLO v5 code base our team is using in the project has rigidly built in assumptions about the data it is using, adding an extra label would mean significant changes throughout the entire codebase, from data reading up to metrics calculations.

Alternatively, you can use a separate CNN model to detect rotation[AGM21, FDB15, MB20]. This approach can make implementation much easier and clearer. By using a separate model, you can avoid the complexities of modifying YOLO and focus on developing your dataset and a simpler-to-grasp algorithm that still fairly accurately detects symbol rotation.

In addition, it is important to consider the potential limitations of these approaches. For example, the accuracy of symbol rotation detection may depend on factors such as the quality of the image and the type of symbol being analyzed. It is also important to test and validate the algorithm on a diverse set of images to ensure that it performs well in various scenarios.

2.2.5 Pose detection

In here we use the term pose to refer to the shape of mission task symbols that illustrate a trajectory. While pose could also mean rotation, we separate rotation and shape (trajectory) detections. There are at least three approaches to detect a symbol's pose.

The baseline approach to pose detection involves training a deep learning model to classify different poses into distinct classes. This requires a large labeled dataset of images, where each image is annotated with the corresponding pose label. The model then learns to identify the features and patterns that distinguish one pose from another. During inference, the model takes an input image and outputs the corresponding pose label. All more detailed information will be lost, the model only reports the class. The pose information forwarded to KOLT will correspond to a template or average of this class. This approach can be effective for differentiating between poses that have clear visual differences, but will by definition not capture subtle variations between poses.

Another approach to pose detection involves using keypoint detection[SJMS17, ZCT21]. This involves identifying specific points on a symbol and using their relative positions to infer the pose. For example, in human pose detection, keypoints might be the locations of the joints like the elbows, knees, and shoulders. In our case the keypoints would be locations of the turning points, that are shown as PT2 and PT3 in figure 2, that KOLT expects to receive as input when drawing the symbol.

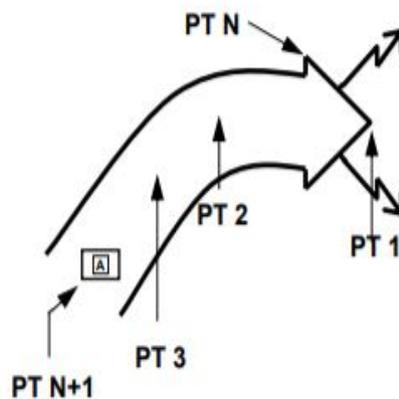


Figure 2. PT2 and PT3 show the turning points of the symbol.[AC11]

Such keypoint model would learn to identify the position of each keypoint in an image. This approach can be more precise than the baseline model since it focuses on identifying

specific landmarks which can be placed in various ways even for similar symbols (same class). Ideally, the output of this approach captures as much variability as KOLT allows to visualize. However, this approach assumes defining landmark points for different symbols and labeling the locations of these points for many images to form a training set. Labeling landmark locations is significantly more time-consuming than labeling an image with a class name in the baseline approach.

The third approach to pose detection involves using an autoencoder[WHX⁺13, Ros20]. An autoencoder is a type of neural network that learns to encode an input image into a compressed representation and then decode it back into the original image. During training, the autoencoder learns to minimize the difference between the original and reconstructed images. Once trained, the model can take an input image and generate a reconstructed pose image. More importantly for us, the compressed representation between encoder and decoder can be used, with the help of some distance measure, to find images that are similar to each other. There are several distance measures such as euclidean distance, cosine similarity and mahalanobis distance. In particular, for any input image, we can find the nearest neighbor within a set of known images for which we have manually annotated all the necessary information to recreate them in KOLT. In such an approach, the pose information KOLT will receive about the input image is the pose of the nearest annotated neighbor. If the set of known images covers the necessary diversity, this approach can be effective for detecting poses that have high variability.

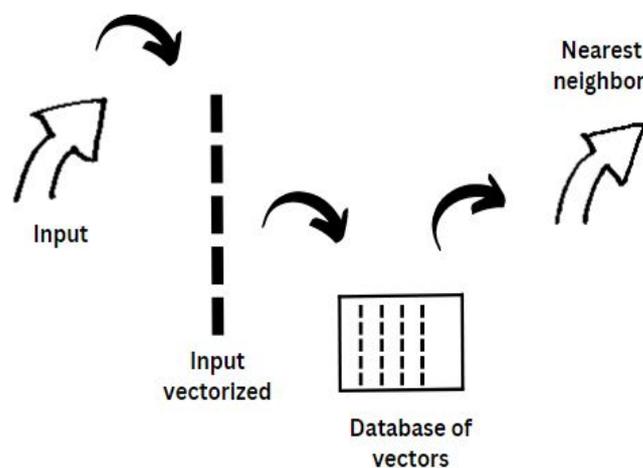


Figure 3. Steps to find the nearest neighbor.

Notice that the creation of autoencoders does not require labeled data, we only need to label the known set among which we find the nearest neighbor.

As labeling the data is a time consuming work. Using keypoint detection is not feasible within the time limitations of this thesis, as we would be unable to label sufficient number of images with necessary keypoints. The baseline approach needs a label per image, but the keypoint approach needs at least 4-5 labels per symbol. Furthermore, the class label is immediately visible when looking at an image, in a fraction of a second, whereas keypoints must be labeled by noting down coordinates. Autoencoder does not need any labeling per class for training the encoder and eventually only needs a limited number of labeled examples, not an entire training dataset. So this thesis will not use the keypoint detection approach and focus on the baseline and autoencoder approach.

2.3 Overview of the collaboration project

In this section, I will provide an overview of the entire collaborative project between the University of Tartu and the Estonian National Defence College, including its main aspects and objectives.

The project is based on detecting mission task symbols[1]. Mission task symbols are standardized graphical symbols that are used to depict various military tasks and activities on operational maps. These symbols are used in military maps and other military planning documents to pass on information about tasks that need to be accomplished by military units. The use of standardized mission task symbols ensure that everyone who is involved has a common understanding of the tasks to be accomplished. Each mission task symbol is different and represents a specific task or activity. For example, an arrowhead symbol known as an attack symbol(fig. 4) represents an attack mission. Another example would be three arrows and a line at the end of arrows(fig. 4) represents a clear symbol, which means to remove enemy forces and eliminate organized resistance in the area.



Figure 4. Attack and clear symbol types.

Overall, mission task symbols are an essential tool in military planning and operations. By using standardized symbols, military personnel can communicate quickly and effectively, which is critical in time-sensitive and rapidly changing situations on the battlefield.

The main objective of the project is to improve the process of how military plans are handled by automating and digitizing the process. This involves solving several key challenges, including obtaining digital images of the films, the alignment of the digital images with the map, localization detection of the mission task symbols, detection of the symbol class and lastly detection of the symbol's rotation and pose. The final objective is to display the detected symbols and information about them in the KOLT system.

The alignment of the films with the map is critical to the success of the project, as it ensures that the mission task symbols are placed accurately on the map. For this end, the films are equipped with location markers that can be detected via computer vision methods on the photos taken of these films. A transformation matrix between image pixels and geographical locations can be defined. Once the films are aligned with the map, the next task is to localize the symbols on the film and detect their class. This is important, as mission task symbols can appear anywhere on the map. After the symbols are localized and detected, it is possible to determine their rotation. This information is essential because it shows in which direction the symbol is pointing, for example it matters to which direction fire support must be directed (support by fire symbol).

Some symbols that depict trajectories do not have a specific shape, so it is important to detect the entire pose of the symbol. One such symbol is the "attack symbol" shown in figure 4. Trajectory symbols can have a wide variety of shapes, which makes their detection and pose estimation particularly challenging.

3 Methodology

3.1 Determining requirements

The first task of this thesis was to determine the essential information required to correctly enter and display each mission task symbol in KOLT. To achieve this, we conducted an in-depth analysis of NATO symbology and considered the various attributes of different mission task symbols. The analysis involved examining the contents of each symbol, as well as any special features or details that may affect their meaning.

In the second phase, we collaborated with the Estonian National Defence College to validate our understanding of NATO symbology and identify the most commonly used and important symbols in military operations. We created a reference document that outlines the key information required to properly understand and interpret each mission task symbol. The document (in appendix I. Requirements table) and a part of it shown in table 1, provide examples of different symbols and the specific details that need to be taken into consideration when recreating mission task symbols and validating their accuracy.

We worked closely with the experts at the Estonian National Defence College to ensure that our understanding of the symbols was thorough and accurate. The validation process was critical as it helped to ensure that our work was aligned with the needs and expectations of the military community. By collaborating with experts in the field, we were able to gain a deeper understanding of NATO symbology.

After a thorough analysis and validation process, we selected 31 most used symbols for company level units, taking into consideration their complexity and likelihood of being successfully identified.

Thirdly, in April 2023, we had a meeting with the KOLT development team to discuss the details of how mission task symbols are inserted into KOLT. During the meeting, we learned that mission task symbols are created by taking into consideration the same keypoint that are used in defining them in the NATO joint symbology document [AC11]. These keypoints define the placement of symbols in KOLT, using the Web Mercator coordinate system. As explained, mission task symbols can have varying numbers of keypoints. For instance, the advance to contact mission task symbol, which is depicted in table 1, can have a large number of keypoints. Those keypoints are marked as PT 1, PT 2, etc. The number of keypoints depends on how long the attack symbol is and what is the shape of the symbol. On the other hand, the control mission task symbol, shown in table 1 contains a fixed number, two keypoints. One in the middle of the symbol and second one at the peak point of the upper arrow.

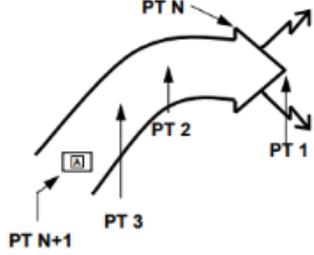
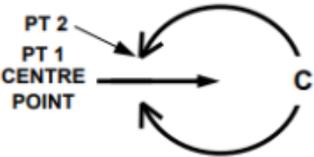
Symbol	Requirements needed to recreate	Metrics to validate
<p>Advance to contact</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Shape • Direction 	<ul style="list-style-type: none"> • The Euclidean distance of PT1 from the human-marked point. • The degree of direction difference from the direction marked by a person. • Qualitative - Similarity of the trajectory
<p>Control</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction 	<ul style="list-style-type: none"> • Intersection over union(IoU) accuracy • The degree of direction difference from the direction marked by a person.

Table 1. Mission task symbols and their requirements example.[AC11]

Overall, our analysis provided us with a comprehensive understanding of the different mission task symbols used in NATO symbology and the key information that is required to correctly enter and display these symbols in KOLT. By applying this knowledge, we can ensure that our team can output to KOLT information in a format that results in correct symbol placement into KOLT and supports effective decision-making in the field.

3.2 Data used

The data used in the thesis consists of symbols representing military mission tasks. The symbols come from NATO's official symbol set [AC11]. Initially, the Estonian National Defence College provided a list of 15 initial mission task symbols to study. Subsequently, in collaboration with the Estonian National Defence College, the 31 most commonly used symbols(fig. 5) were selected and the study was expanded.

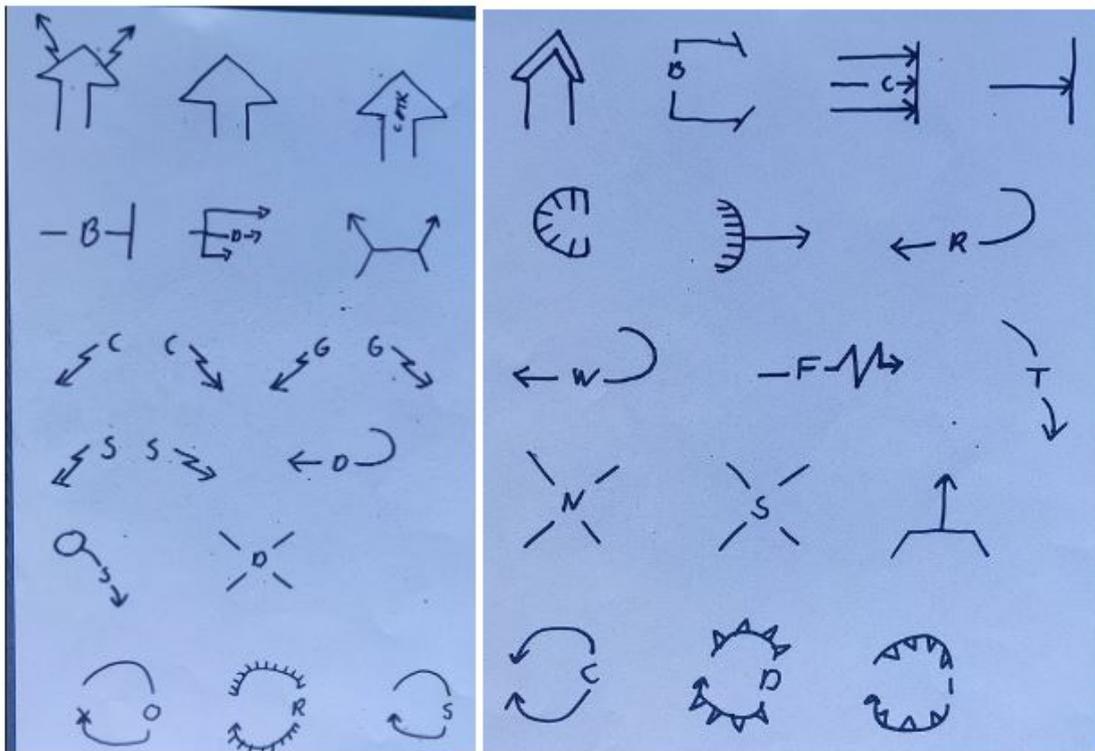


Figure 5. 31 most commonly used symbols. Left paper sheet shows the 15 initial mission task symbols - advance to contact, attack, counterattack, block, disrupt, support by fire, cover, guard, screen, delay, seize, destroy, occupy, retain and secure. Right paper sheet shows 16 added mission task symbols - main attack, breach, clear, penetrate, contain, ambush, retire, withdraw, fix, turn, neutralize, suppress, attack by fire, control, deny and isolate.

Since there was no existing dataset available for this project, a dataset had to be created. As a result, many of the symbols had to be drawn manually by myself and other collaboration project members to create a dataset. Those drawn symbols had to be manually labeled into different classes. Deep learning models require a significant amount of data to be trained effectively. At the beginning of the project, the team had zero real military plans. Throughout the project, we were able to obtain a few hundred, but the quality of these was judged to be 80% unacceptable by military personnel. Even if putting in significant effort to label military films using labeling tools, there simply wasn't enough data, not even on film and even less digitized. For example, YOLO recommends training a neural network with 10000 instances (labeled objects) per class, which it uses to learn to detect symbol localization and class[JCS⁺22].

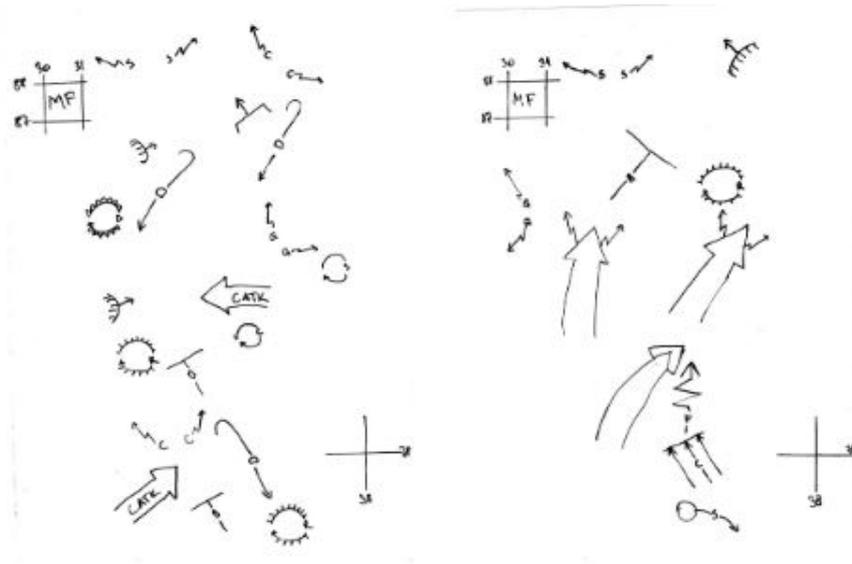


Figure 7. Real clean images. The symbols form a meaningful plan and have some kind of military sense.

mission task symbols. The second data would be real data from the wild, as drawn by cadets in military practices. These datasets were used to evaluate the performance of the models in more realistic scenarios. Even if the trained models do not generalize directly to real data, unless real data is not just more diverse, but actually in essence different, the same methods should work if given enough real data, which we simply do not have yet.

3.2.1 Rotation models data

Training a neural network model requires a vast amount of data. In this case, a dataset was generated using 31 different template images for each of 31 symbols classes for the training set. To ensure a diverse dataset of handwriting styles, the images of symbols were drawn by nine different individuals. This approach captured natural variation in handwriting present in real-world data, which is crucial for models used in applications with varying handwriting styles. Involving multiple individuals reduced potential bias in the dataset that could have been introduced if only one person had drawn all symbols. Including a variety of individuals with different handwriting styles created a more representative dataset and reduced bias towards a particular handwriting style. In total 31 x 31 template images were created, on the basis of which generation process was launched. It is also important to note that each data point in the generated dataset consisted of an image and a rotation value. All template images were drawn to be in the same zero

rotation position, and the generator was aware of the rotation value it applied to each template image when placed on a canvas. The resulting rotated symbol was extracted using bounding box information, and the rotation value was set as the corresponding label. This ensured that the generated dataset included a diverse range of rotated symbols.

The total number of generated images produced was 150000, with each image containing between 3 to 6 symbols. The generation process took about 2-3 hours of computing power. As a result, the dataset consisted of approximately 675000 symbols, with just over 20000 symbols per class. It is important to note that the dataset was generated using a dataset generator. This tool enabled the generation of large amounts of synthetic data with a high level of control over the properties of the data. The dataset generator utilized a set of rules and parameters to create the images of symbols. For example, the generator could vary the size, shape, and orientation of the symbols, as well as the background and foreground elements in the images. This allowed for a wide range of variation in the dataset, which can help to improve the model's ability to generalize to new, unseen data. By using a dataset generator, it was possible to generate a dataset with a sufficient number of samples to train a neural network model effectively. This approach is particularly useful in this case as obtaining large amounts of real-world data is difficult or expensive. The ability to generate synthetic data is helping to reduce the need for costly data collection and annotation. Furthermore, it should be emphasized that there are essentially 31 separate datasets from one large dataset, each for a different symbol class, which will be treated separately during the model training process. This approach allows for greater control over the training process and enables the model to learn the specific features of each symbol class more effectively.

To assess the effectiveness of the trained neural network model, a separate test dataset was generated. The test dataset consisted of 7000 images. The test dataset was generated based on three template symbols per class. It is important to note that, the test templates were not used during training data generation, so test data can be considered to represent "new handwriting" for the models created. This resulted in a total of 31000 symbols in the test dataset, with roughly 1000 symbols per class. The test dataset was also generated by the generator and each image had again 3 to 6 symbols per image. By using a separate test dataset, the accuracy of the model's predictions on new, unseen data could be evaluated. This is important as it can help to determine whether the model has learned to generalize effectively to new data, or if it is simply memorizing patterns in the training set. The use of a test dataset can also help to identify any potential issues with the model, such as overfitting or underfitting. Overfitting occurs when the model becomes too specialized to the training data, resulting in poor performance on new data. Underfitting occurs when the model is too simple and cannot effectively capture the patterns in the data, resulting in poor performance on both the training and test data. In summary, the test dataset

provides an important benchmark for evaluating the performance of the neural network model, and can help to identify any potential issues that may arise.

3.2.2 Pose models data

There are four symbols which are more complicated to describe than others as there is a higher variability that can occur for these symbol types. This is because these symbols illustrate movement trajectories on the map. These symbols are advance to contact, attack, counterattack and main attack (fig. 8). For these models, rotation is defined as the direction the arrow starts from, with zero rotation meaning arrows starting from bottom center. However, further information needs to be extracted to characterize the exact pose of the arrow. We propose two types of models for retrieving this information.

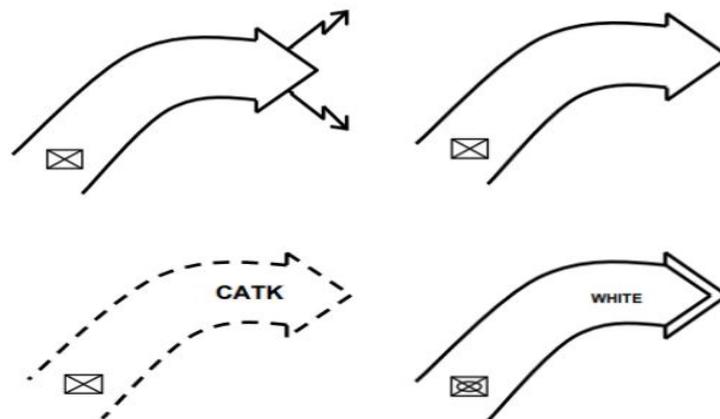


Figure 8. Symbols that represent a trajectory. Top right - advance to contact mission task symbol, top left - attack mission task symbol, bottom right - counterattack mission task symbol and bottom left - main attack mission task symbol.

3.2.3 Pose base model dataset

For trajectory symbols a new dataset was generated which only consisted of trajectory symbols and the needed labels for the trajectory baseline model.

For every symbol type 39 examples were collected which then were labeled into 6 classes: narrow straight, bold straight, 45 degrees right, 90 degrees right, 45 degrees left and 90 degrees left.

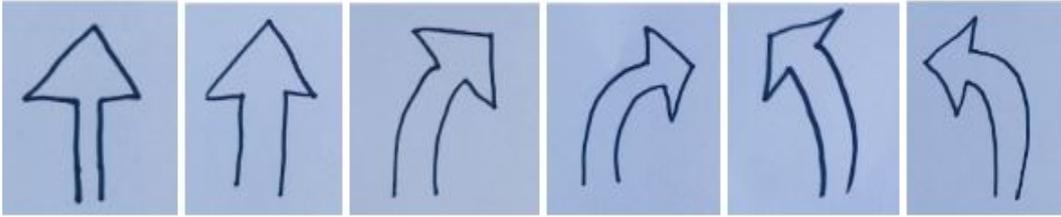


Figure 9. Examples of subclasses. From left to right: narrow straight, bold straight, 45 degrees right, 90 degrees right, 45 degrees left and 90 degrees left.

The next step was using a generator to generate a dataset. However the generator was changed for generating this type of dataset as usually there is only a class we need to know. Here we need to know two, what type of the symbol we are dealing with, e.g. advance to contact, attack, counterattack or main attack and also how this symbol is interpreted, e.g. narrow straight, bold straight, 45 degrees right, 90 degrees right, 45 degrees left or 90 degrees left. We will call the first mentioned class as class and second class as subclass. 6500 canvas images were generated which included randomly rotated and distorted by augmentations images based on these 4 x 39 examples. Each image had 3 to 6 symbols, resulting in around 30000 symbol images or around 7500 symbols per class and around 1250 symbols per subclass.

To generate the test set new examples were again used. For every symbol type 18 new examples were drawn and labeled into 6 classes. The same method was used as in the process of generating the training dataset. Test dataset contained 1350 canvas images. Which makes 1500 symbols per class and 250 symbols per subclass.

3.2.4 Pose autoencoder dataset

A dataset containing 21600 canvas images was generated for the autoencoder. This dataset includes 4 different types of trajectory symbols: advance to contact, attack, counterattack, and main attack. Similar to previous dataset generations, there were between 3 to 6 symbols present per canvas image, resulting in close to 24000 symbols per symbol type. In addition, it is important to note that an autoencoder is a type of neural network that learns to encode and then decode data, with the goal of reducing the amount of data needed to represent the input while retaining important information. So in this case there is no needed corresponding labels data.

For testing, the same dataset was used as in the base model, which contains 1350 canvas images and which makes 1500 symbols per symbol type. This allows for a fair comparison between the performance of the autoencoder and the base model on the same dataset.

3.3 Methods used

There are several ways to detect and classify symbols of combat operations. For the classification and localization of mission task symbols, in the collaboration project we use YOLO[RDGF16]. Since it is a powerful solution that already includes symbol detection and then recognition. But as far as rotation detection and pose detection, the solutions were developed in this thesis and they will be explained in the following chapters. All the methods and data generation codes used in this study are available in collaboration project public GitHub repository at <https://github.com/aralacikalin/NatoSymbols>.

3.3.1 Rotation detection method

Rotational prediction is quite difficult, as the number of possible rotations of a symbol can be quite large, which can make it difficult to accurately predict all possible rotations. One key issue is that rotations of an image can result in significant changes to the pixel values and overall appearance of the image. This can make it difficult for the model to identify the symbol and classify it correctly.

Modifications to YOLO architecture exist that allow detecting rotations of objects[YPS⁺18]. Initially, we planned to use this approach as I was familiar with YOLO architecture as in the collaboration project we are using it for symbol localization and detection. However, upon closer examination, it became clear that although YOLO can be used for rotation detection, it would require a very large amount of modification to YOLO source code, and the result would probably not be better compared to the convolutional neural network(CNN) [Sae17, RDD19] created by oneself.

Therefore, the decision was made to create new convolutional neural network models specifically designed for the detection of rotation. Datasets that contain 150000 images were used to train the CNN models for each symbol type. 31 specialized models were made, each dedicated to detecting the rotation of a specific symbol type. Since zero rotations of symbols are arbitrarily defined, combining all rotation detections into one large model is unlikely to provide any benefit. Given that the task is visually distinct for each symbol, having multiple specialized models is likely to yield better results.

For each symbol type, we used the same CNN architecture for training. We based

the model architecture on previous works that were created for similar tasks, such as document orientation detection and image orientation detection[AGM21, FDB15, MB20]. However, we modified the model to suit our specific use case. The model begins with an input layer that takes in grayscale images of size 80x80 pixels. The middle of the model architecture consists of 5 convolutional layers with max pooling and batch normalization, followed by a flatten layer and 2 fully connected layers. Finally, the model architecture ends with a softmax activation function that produces the classification output into 360 classes, i.e. one degree precision prediction of rotation angle.

The model architecture:

- Input shape: (80, 80, 1)
- First layer: 32 filters, kernel of size (3,3), with stride of (1,1), using 'same' padding, and ReLU activation.
- Batch normalization.
- MaxPooling2D layer is applied with pool size of (2,2) and 'same' padding.
- Second layer: 32 filters, kernel of size (3,3), with stride of (1,1), using 'same' padding, and ReLU activation.
- Batch normalization.
- MaxPooling2D layer is applied with pool size of (2,2) and 'same' padding.
- Third layer: 64 filters, kernel of size (3,3), with stride of (1,1), using 'same' padding, and ReLU activation.
- Batch normalization.
- MaxPooling2D layer is applied with pool size of (2,2) and 'same' padding.
- Fourth layer: 64 filters, kernel of size (3,3), with stride of (1,1), using 'same' padding, and ReLU activation.
- Batch normalization.
- MaxPooling2D layer is applied with pool size of (2,2) and 'same' padding.
- Fifth layer: 64 filters, kernel of size (3,3), with stride of (1,1), using 'same' padding, and ReLU activation.
- Batch normalization.

- Flatten layer.
- Two fully connected layers with 512 neurons and ReLU activation.
- An output layer with 360 neurons and softmax activation.

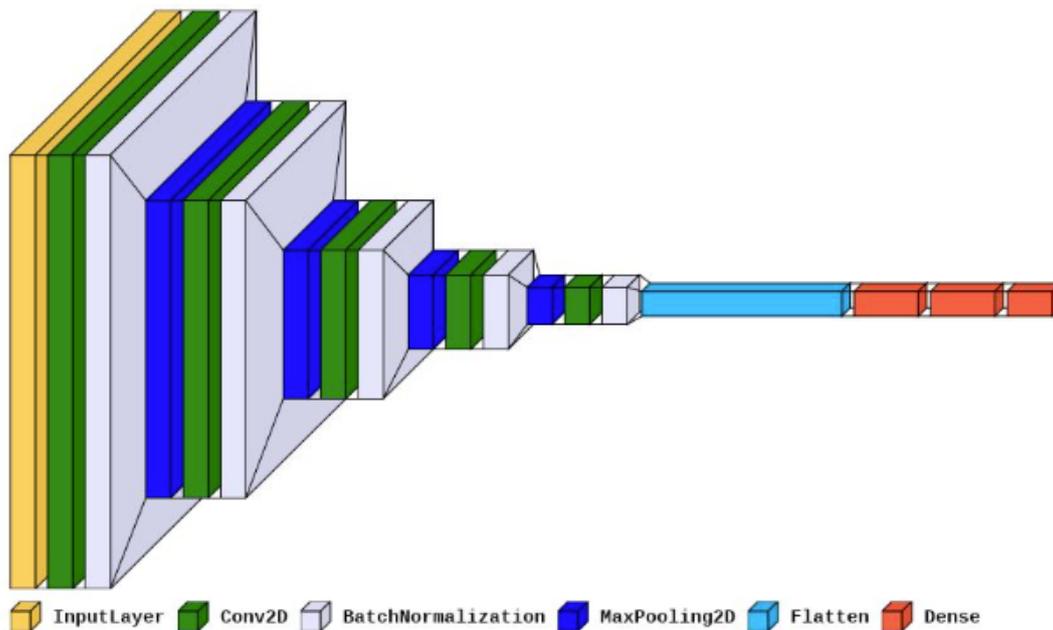


Figure 10. Rotation model architecture.

The model is compiled using categorical cross-entropy for the loss function, adam optimizer with default parameters, and categorical accuracy and mean absolute angle error for evaluation metrics. Angle error metrics calculates the absolute difference between the true value and predicted value. The difference can be between 0 to 180 degrees. The model is trained using 64 batch size and training is going on for 100 epochs. The training dataset is split 80% for training and 20% for validation. Early stopping with patience 10 and improvement threshold 0 is applied to stop training the model before overfitting.

After the model is trained, its performance is evaluated using a separate test dataset. This dataset consists of 7000 images, which were not represented in the training or validation and which are based on a separate set of handwritten template images.

3.3.2 Pose detection methods

Detecting a symbol's pose is a difficult task as trajectory symbols can have a large variety between themselves. To address this difficulty, two models were developed: a baseline model and an autoencoder model. At first, an easier baseline model is tried, which classifies all possible trajectories into 6 classes, and secondly a more difficult autoencoder model is trained, which should be able to capture trajectories with more detail.

3.3.3 Pose base model method

To train the baseline model, firstly the data preparation is handled. The train dataset consists of 6500 canvas images. The CNN model is made to train the model. The model is made of an input layer which is followed by four convolutional layers with max pooling and batch normalization, followed by a flatten layer and 2 fully connected layers. The last layer is the output layer with 6 classes, which are wide straight, narrow straight, 45 degrees right, 90 degrees right, 45 degrees left or 90 degrees left. For each class an approximate number of examples are 4875 examples.

The model full architecture:

- Input shape: (80, 80, 1)
- First layer: 64 filters, kernel of size (3,3), with stride of (1,1), using 'same' padding, and ReLU activation.
- Batch normalization.
- MaxPooling2D layer is applied with pool size of (2,2) and 'same' padding.
- Second layer: 64 filters, kernel of size (3,3), with stride of (1,1), using 'same' padding, and ReLU activation.
- Batch normalization.
- MaxPooling2D layer is applied with pool size of (2,2) and 'same' padding.
- Third layer: 128 filters, kernel of size (3,3), with stride of (1,1), using 'same' padding, and ReLU activation.
- Batch normalization.
- MaxPooling2D layer is applied with pool size of (2,2) and 'same' padding.

- Fourth layer: 128 filters, kernel of size (3,3), with stride of (1,1), using 'same' padding, and ReLU activation.
- Batch normalization.
- MaxPooling2D layer is applied with pool size of (2,2) and 'same' padding.
- Flatten layer.
- Two fully connected layers with 256 neurons and ReLU activation.
- An output layer with 6 neurons and softmax activation.

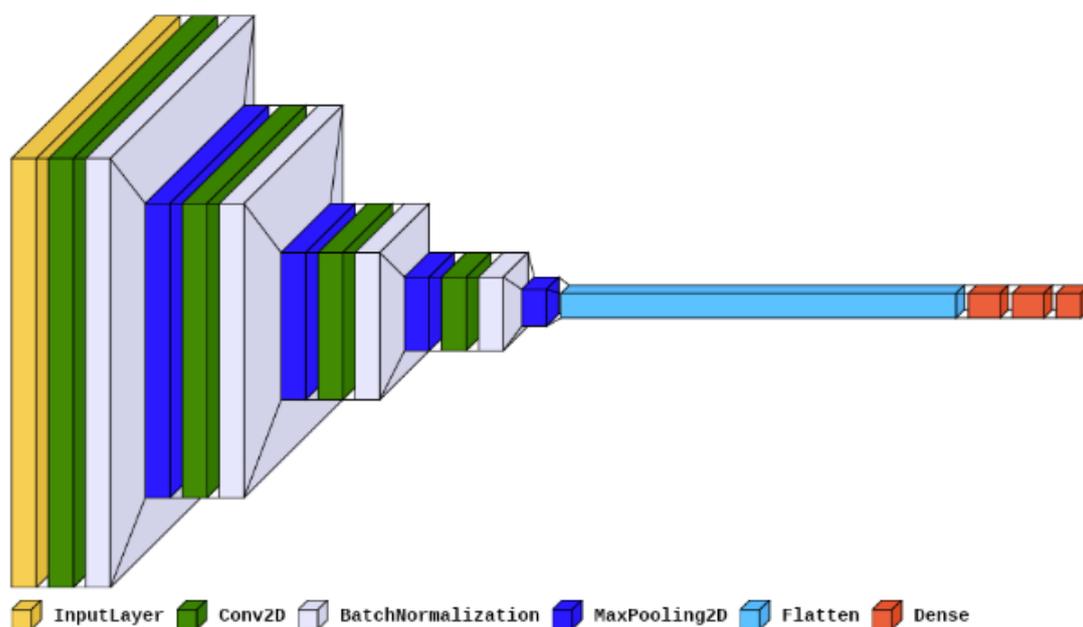


Figure 11. Base Pose model architecture.

The model is compiled using categorical cross-entropy loss function and Adam optimizer with default parameters. The training evolution is tracked using categorical accuracy metrics. The model is trained for 100 epochs using batch size 64. Training data is split 80% of the data to training and 20% of the data to validation. No early stopping was employed.

During the evaluation step, a separate test dataset is utilized to assess the model's performance. This test dataset consists of symbols that were not present in the training dataset and consists of approximately 6000 symbols extracted from 1350 canvas images.

The mentioned dataset is used to evaluate the model accuracy.

3.3.4 Pose autoencoder method

An autoencoder is a type of neural network that learns to encode and decode data in an unsupervised manner. It is composed of two main parts: an encoder and a decoder[PGH⁺16]. The encoder takes the input data and transforms it into a compressed representation, which is called the latent space. The decoder then takes this compressed representation and tries to reconstruct the original input data. The goal of an autoencoder is to learn a compressed representation of the input data that captures the most important features of the data. This compressed representation can then be used for a variety of purposes. One of the key advantages of autoencoders is that they can learn to extract meaningful features from the input data without the need for explicit supervision or labels. This makes them useful for tasks where labeled data may be difficult or expensive to obtain.

In our specific case, a variational autoencoder (VAE) was chosen to interpret images[Doe21, KW19]. A VAE is a type of generative model that learns a low-dimensional latent representation of high-dimensional data. It is called "autoencoder" because it is composed of two parts, an encoder and a decoder, that work together to encode and decode the input data. VAEs introduce a probabilistic element to the model, which means that they model the data distribution using a probability distribution over the latent variables. This allows the model to learn the distribution of the data in the latent space, rather than just learning a fixed latent vector representation for each input. This probabilistic approach to modeling the data distribution is one of the key features that sets VAEs apart from traditional autoencoders. To achieve this, VAEs use a reparameterization technique that makes gradient estimation tractable. This technique involves sampling from a Gaussian distribution in the latent space and using a reparameterization trick to enable backpropagation through the sampling operation. This allows the model to learn the distribution of the latent space while still being able to use gradient-based optimization to update the model parameters. In mathematical term VAE can be written as follows[KW19]:

$$L(\theta, \phi) = E[\log p(x|z, \theta)] - KL(q(z|x, \phi)||p(z)), \quad (1)$$

where θ and ϕ are the parameters of the decoder and encoder networks, respectively.

VAE mathematical term consists of two terms:

- **Reconstruction term:** The first term $E[\log p(x|z, \theta)]$ is the expected log-likelihood of the data x given the latent code z and the decoder parameters θ . This term

encourages the model to generate data that is similar to the observed data. The reconstruction term measures how well the decoder can reconstruct the input data x from the latent code z .

- KL divergence term: The second term $KL(q(z|x, \phi)||p(z))$ is the Kullback-Leibler divergence between the approximate posterior distribution $q(z|x, \phi)$ and the prior distribution $p(z)$ over the latent code z . This term measures the difference between the approximate posterior distribution and the prior distribution and encourages the model to learn a latent code that is close to the prior distribution. The KL divergence term encourages the model to learn a disentangled representation of the input data.

To further improve the performance of the model, an extension called Beta-VAE was utilized[FMMW21]. This approach adds an adjustable hyperparameter (beta) that balances the trade-off between latent channel capacity and independence constraints with reconstruction accuracy. Beta-VAE has been shown to outperform standard VAEs, particularly in terms of disentangling the underlying factors of variation in the data[BHP⁺18]. In mathematical term beta-VAE can be written as:

$$L(\theta, \phi) = E[\log p(x|z, \theta)] - \beta KL(q(z|x, \phi)||p(z)), \quad (2)$$

where β is a beta hyperparameter.

The VAE encoder part takes as input an image size of 80x80 pixels and 1 channel. After the input layer there are 4 convolutional layers. Each layer applies a 3x3 filter with stride 2 and also a ReLU activation function. The following layer is a flatten layer and then comes two fully-connected layers with 256 neurons. Then it will be split into two branches, each with 10 neurons.

Encoder architecture:

- Input shape: (80, 80, 1)
- First convolutional layer: 32 filters, kernel of size (3,3), with stride of (2,2), using 'same' padding, and ReLU activation.
- Second convolutional layer: 32 filters, kernel of size (3,3), with stride of (2,2), using 'same' padding, and ReLU activation.
- Third convolutional layer: 32 filters, kernel of size (3,3), with stride of (2,2), using 'same' padding, and ReLU activation.
- Fourth convolutional layer: 32 filters, kernel of size (3,3), with stride of (2,2), using 'same' padding, and ReLU activation.

- Flatten layer.
- Fully connected layers with 256 neurons and ReLU activation.
- Fully connected layers with 256 neurons and ReLU activation.
- Split into two branches, with each branch consisting of a fully connected layer with 10 neurons. These branches are used to calculate the mean and log-variance of the latent space using the reparameterization trick.

The second part of VAE is the latent space layer. It is a bottleneck layer that represents the distribution over the latent variables. The latent space is a lower-dimensional representation of the input that captures the important features of the data. The latent space gets from the encoder the mean and log of the variance of a multivariate Gaussian distribution. The mean vector represents the center of the distribution in the latent space, while the log variance provides the shape of the distribution. Then the mean and log variance are used to sample a point from the distribution using the reparameterization trick. The reparameterization trick involves multiplying the standard deviation with a random sample drawn from a unit Gaussian distribution, adding the mean, and obtaining a sample from the distribution.

The third part of VAE is the decoder. Decoder is taking in the latent space point. There are at first 3 fully connected layers. Which is followed by four transposed convolutional layers with 32 filters, using a kernel size of 3x3 and stride of 2. The final layer outputs an image.

Decoder architecture:

- Fully connected layers with 256 neurons and ReLU activation.
- Fully connected layers with 256 neurons and ReLU activation.
- Fully connected layers with 512 neurons and ReLU activation.
- Reshape into 32x5x5.
- First transposed convolutional layer: 32 filters, kernel of size (3,3), with stride of (2,2), using 'same' padding, and ReLU activation.
- Second transposed convolutional layer: 32 filters, kernel of size (3,3), with stride of (2,2), using 'same' padding, and ReLU activation.
- Third transposed convolutional layer: 32 filters, kernel of size (3,3), with stride of (2,2), using 'same' padding, and ReLU activation.

- Fourth transposed convolutional layer: 32 filters, kernel of size (3,3), with stride of (2,2), using 'same' padding, and ReLU activation.
- Output layer as transposed convolutional layer: 1 filter, kernel of size (3,3), with stride of (1,1), using 'same' padding, and sigmoid activation.

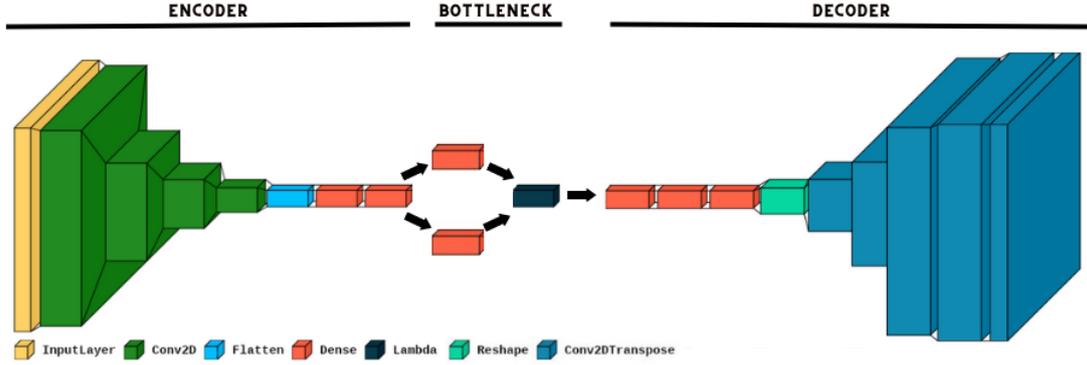


Figure 12. Beta VAE model architecture.

Before training the model additional loss function is added to the model. The VAE loss is a combination of reconstruction loss and KL divergence loss. The VAE loss is the mean value from the sum of the reconstruction loss and the KL divergence loss.

$$VAELoss = reconstructionloss + KLloss \quad (3)$$

The reconstruction loss in VAE is computed as the mean squared error between the input image and the output. The reconstruction loss measures how well the model is able to reconstruct the image.

$$reconstructionloss = mse(input, output) \quad (4)$$

The KL divergence loss in VAE is computed by utilizing the mean and log-variance of the latent space obtained from the encoder, along with a hyperparameter known as beta for beta VAE. This loss is used to measure the difference between the distribution of the latent space and a known prior distribution. A multivariate normal distribution is employed in the calculation of KL divergence in VAE.

$$KLloss = -0.5 * beta * sum(1 + logvar - square(mean) - exp(logvar)) \quad (5)$$

A model is trained for each trajectory symbol type. For each model training dataset size is around 20000 images of symbols. The validation dataset size is around 4000 images

of symbols. The loss function for the model is previously mentioned VAE loss function. The Adam optimizer with default parameters is used for the model. The models are trained for 200 epochs with a batch size of 128.

To evaluate the performance of the autoencoder model, the same test dataset used in the base model chapter was utilized, consisting of 1350 canvas images and around 6000 images of symbols. However, unlike the base model, the unsupervised autoencoder does not have any labeled data to train on. Therefore, to make predictions on new data, the latent space values of the new data are computed using the encoder network of the autoencoder.

To find the predicted class of the new data, cosine similarity is utilized to compare the latent space values of the new data to the latent space values of the symbols in the comparison dataset. The symbol with the closest latent space value to the new data is then predicted as the class of the new data. It is important to note that a labeled database is necessary to implement this approach, as the nearest neighbors method requires labeled data to find the closest match. This method allows for the unsupervised autoencoder to make predictions on new, unseen data without the need for manual labeling.

3.4 Language model to improve readability

For my Master's thesis, I employed ChatGPT, a large language model, to improve the readability of the text. ChatGPT's advanced natural language processing capabilities allowed me to refine the language and structure of my writing, resulting in a more coherent and engaging final product. The use of ChatGPT helped to ensure that the research was communicated effectively, and that my findings were accessible to a wider audience. ChatGPT was not used to create new paragraphs, it was only used for editing existing content.

4 Results

4.1 Set of requirements for each symbol type

The first outcome of this thesis is a table containing for each symbol type what information is needed to recreate this symbol at the correct location, with correct size and shape on a digital map in the KOLT system. In the scope of the present project we mainly needed to segregate symbols for which only location matters, those for which location and rotation are needed and those for which a trajectory as well needs to be defined. However, the full requirements table also contains a mapping of which metrics one should use when evaluating the detection results of each symbol.

To simplify the project, the symbols were segregated into three categories based on their requirements. The first category includes three symbol types - destroy, neutralize, and suppress - for which only location is necessary. The center point location is enough to correctly interpret them into the KOLT system.

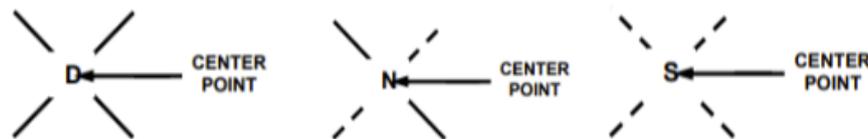


Figure 13. First symbol types category. From left to right: Destroy, Neutralize and Suppress.

The second category comprises the largest group of symbol types - 24 out of 31 - which require both location and rotation. For these symbols, 2 to 4 points are necessary to interpret them into the KOLT system. The symbols in this group have varying requirements. Some have two, three or four points defined, while others have a center point and one or two additional points. The following symbol types in this group are: breach, block, clear, penetrate, contain, disrupt, support by fire, cover, guard, screen, ambush, attack by fire, delay, retire, withdraw, fix, seize, turn, control, deny, isolate, occupy, retain and secure.

Explanation of how different these symbols are:

- Breach, block, clear and penetrate symbols type are similar, as they all have 3 points defined. Points 1 and 2 define the front of the symbol and point 3 defines the rear of the graphic.

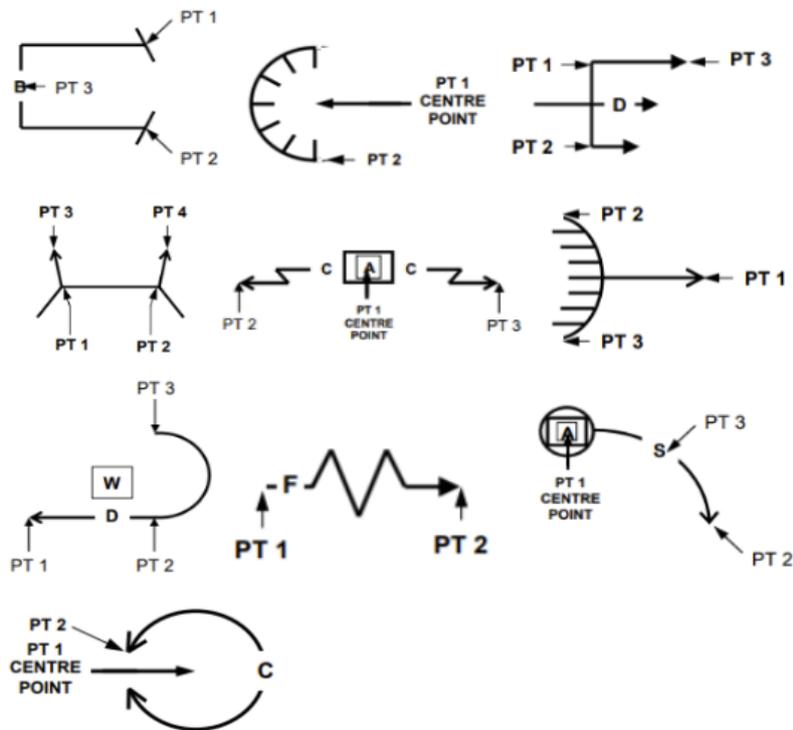


Figure 14. Several symbols from second symbol types category. Symbols from up to down and left to right: breach, contain, disrupt, support by fire, cover, ambush, delay, fix, seize, control

- Contain symbol has a center point, which defines the center of the symbol and point 2, which defines the graphic start point.
- Disrupt symbol has 3 points. Point 1 and 2 define the end points of the graphic and point 3 defines the tip of the symbol.
- Support by fire symbol has 4 points. Point 1 and 2 define back line size and point 3 and 4 define the tip of the graphic.
- Cover, guard and screen symbol types are similar. All of them have center point and point 2 and 3, which define the tips of the graphics.
- Ambush and attack by fire symbols types are similar. Both of them have point 1, which defines the tip of the graphic and points 2 and 3 define the endpoints.
- Delay, retire, withdraw symbol types are similar. Point 1 defines the tip of the symbol, point 2 defines the endpoint of the straight line and start of the arc. Point

3 defines the orientation and size of the circular arc.

- Fix symbol has 2 points. Point 1 defines the rear of the graphic and point 2 defines the front of the graphic.
- Seize and turn symbol types are similar. They both have point 1, which defines the rear of the graphic, point 2 defines the tip of the graphic and point 3 defines 90 degree arc.
- Control, deny, isolate, occupy, retain and secure are similar symbol types. They all have a center point, which defines the center point of the graphic and point 2 defines the graphic's start point and radius.

The third and final category includes symbols that require trajectory as well. These symbols are advance to contact, attack, counterattack, and main attack(fig. 15). For these symbols, several points are needed to define to get the arrowhead and anchor points.

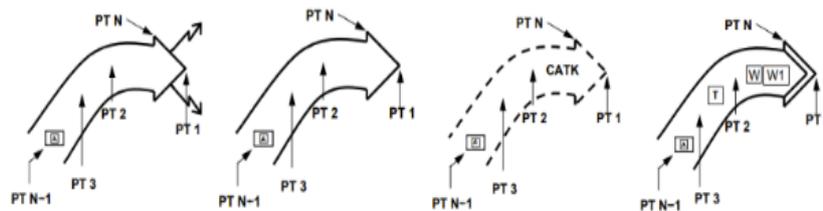


Figure 15. Trajectory symbols: advance to contact, attack, counterattack and main attack.

For all these symbols point 1 defines the tip of the arrowhead, point N defines the end of the arrowhead and the width of the arrow. Points 2 to N-1 define anchor points of the symbol. Point N-1 also defines the rear of the graphic.

Overall, this table provides a detailed breakdown of the requirements for each symbol type in the KOLT system. It includes information on the number of points required and which metrics to use when evaluating the detection results of each symbol.

4.2 Rotation detection results

The first goal for the rotation detection, agreed within the team and communicated to collaboration partners, was to have at least 80% of the symbols detected with a rotation angle lower than 30 degrees from the real value. Although this seemed challenging at

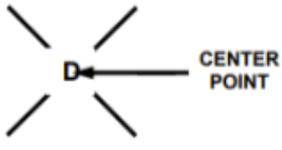
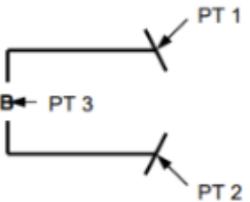
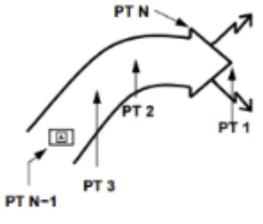
Symbol group and example	What is need to know
Location group: Destroy 	<ul style="list-style-type: none"> • Symbol ID • Location
Location and rotation group: Breach 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction
Location, rotation and trajectory group: 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction • Shape

Table 2. Information of what KOLT needs to know about symbols to be inserted into KOLT.[AC11]

first, the initial model yielded promising results for 15 symbols. For training purposes, 31 different template symbols were used for each class, while 3 different symbols were used for generating test dataset for each class. The first iteration of results showed an average error of 18.5 degrees, with 88.7% of the detection errors being lower than 30 degrees.

After refining the model, the overall results for all 31 symbol types showed an impressive mean degree error of 11.65 degrees. Moreover, the mean percentage of symbols detected with a rotation angle error lower than 30 degrees across all symbol types was 95.6%. These results demonstrate the effectiveness of the approach taken in achieving the goal of accurate rotation detection. The results for each symbol type can be seen in table 3.

Symbol name	Count	Mean degree error	Count of "< 30"	% of "< 30"
advance_to_contact	995	10.15	968	97.29%
ambush	1053	9.47	1034	98.20%
attack	987	7.42	972	98.48%
attack_by_fire	1042	9.95	1018	97.70%
block	1048	12.39	1005	95.90%
breach	990	9.82	965	97.47%
clear	1099	8.82	1086	98.82%
contain	1027	10	999	97.27%
control	1043	13.75	999	95.78%
counterattack	979	9.22	937	95.71%
cover	986	11.6	944	95.74%
delay	1007	9.13	993	98.61%
deny	983	9.71	968	98.47%
destroy	1036	12.83	968	93.44%
disrupt	983	9.61	966	98.27%
fix	1000	10.42	978	97.80%
guard	1016	9.5	997	98.13%
isolate	979	14.25	939	95.91%
main_attack	980	14.29	910	92.86%
neutralize	1010	13.78	926	91.68%
occupy	958	10.72	932	97.29%
penetrate	1061	9.86	1036	97.64%
retain	994	9.89	977	98.29%
retire	1008	10.1	980	97.22%
screen	1018	11.75	979	96.17%
secure	975	9.29	956	98.05%
seize	1075	21.52	875	81.40%
support_by_fire	1016	9.25	1002	98.62%
suppress	1044	26.7	777	74.43%
tum	1057	15.57	978	92.53%
withdraw	935	10.38	912	97.54%
		11.65		95.57%

Table 3. Rotation models results.

The models were also evaluated on “clean real data” images obtained from the Estonian National Defence College. It is important to note that the dataset only contained 17 different symbol types, and therefore the overall accuracy of all 31 models cannot be fully assessed. The mean degree error achieved was 31.80 degrees, and 72.41% of the symbols were classified with an error under 30 degrees. More precisely values can be seen in table 4.

Across the two evaluation datasets we see that some symbols types are more challenging for the rotation detection models. This can be due to their symmetric nature (so rotation

Symbol name	Count	Mean degree error	Count of "< 30"	% of "< 30"
advance_to_contact	20	9.6	20	100.00%
ambush	30	24.2	25	83.33%
attack_by_fire	10	20.1	9	90.00%
block	30	8.8	30	100.00%
clear	10	4.4	10	100.00%
control	10	71.2	4	40.00%
counterattack	20	24.75	12	60.00%
cover	20	31.75	14	70.00%
delay	30	23.13	24	80.00%
deny	10	31.4	8	80.00%
fix	10	17	9	90.00%
guard	20	23.7	18	90.00%
main_attack	10	51.7	4	40.00%
retain	28	40.5	17	60.71%
screen	19	50.05	7	36.84%
secure	10	55.2	6	60.00%
seize	10	53.2	5	50.00%
		31.80		72.41%

Table 4. Rotation models result on real clean data.

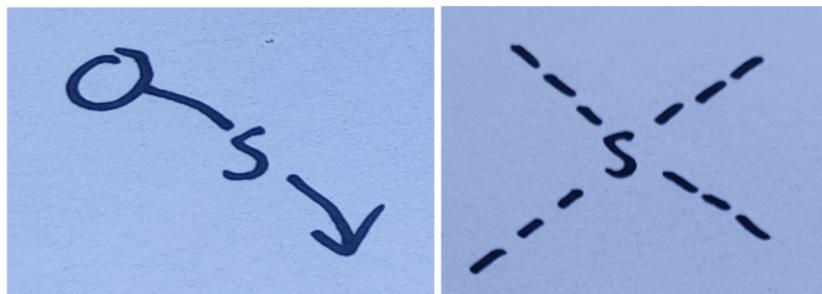


Figure 16. Left side is a seize symbol and the right side is a suppress symbol.

causes minimal visual difference at certain angles) or large variability in their appearance due to handwriting. For the generated test data, the only symbols detected with above 20 degree mean error were seize and suppress, which are illustrated on figure 16.

The clean real data represents example plans with actual military meaning, drawn in a variety of handwritings. The rotation detection struggled the most with a different set

of symbols compared to the generated test set. Likely, the handwriting in this dataset differed and the models failed to generalize in certain cases. Due to handwriting the difficulty of detecting orientation of highly symmetric symbols can be amplified (high error in control, secure, retain). Additionally, the “screen” and “main attack” symbols are often drawn in non-standard ways in practice, making the rotation detection challenging. As “screen” symbols in real clean data, do not have unit symbols in them, however in the training dataset there were unit symbols inside “screen” symbols(fig. 17). Also in the real data there are a lot of symbols very close, which can make the prediction harder compared to the synthetic data.

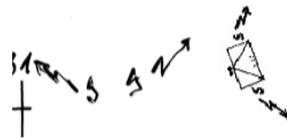


Figure 17. Left screen symbol is from real clean data and right screen symbol is from the training dataset.

In conclusion, the results of our study indicate that machine learning models show great potential in recognizing and interpreting mission task symbol rotation. Our models, trained on synthetic data, achieved the desired results for all symbols. When tested on real clean data, the models still performed well enough on most symbol types. These promising results suggest that with a larger dataset that includes more real-world data, the methods can learn to generalize to all cases.

4.3 Pose detection results

4.3.1 Base model results

The base model performed well with an accuracy of 87% over the 4 symbol types in predicting the six subclasses, which include wide straight, narrow straight, 45 degrees right, 90 degrees right, 45 degrees left, and 90 degrees left. However, it struggled with distinguishing between symbols from neighboring subclasses, particularly those with angles between 45 and 90 degrees and wide/narrow straight symbols. Additionally, the model occasionally misclassified straight symbols as 45 degrees left or right, and vice versa.

Upon closer inspection of the different symbol types and their accuracies, it was found that the attack symbol type had the highest accuracy of 94%. The symbol types that

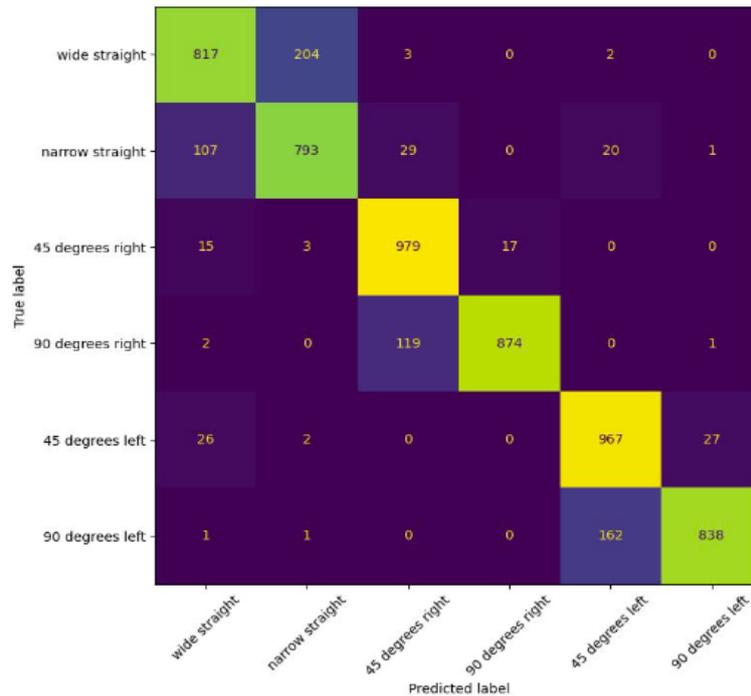


Figure 18. Base model confusion matrix over all 4 symbol types.

followed in accuracy were main attack, counterattack, and advance to contact, with accuracies of 90%, 84%, and 79%, respectively. More precisely shown the results in figure 19. These results demonstrate the varying degrees of accuracy achieved across different symbol types and highlight the importance of considering these differences when designing and implementing machine learning models for symbol classification. It also indicates that the model may be better suited for certain symbol types over others.

Figure 20 illustrates the difference between 45 degrees right and 90 degrees right symbols. While these symbols have distinct angles, they can be challenging to differentiate even for the human eye. However, the model is pretty adept at distinguishing between them.

The model performs well in predicting the classification of the six classes. However, as shown in figure 18, distinguishing between symbols that are oriented 90 degrees left and 45 degrees left can be challenging. This is because many symbols fall between the range of 45 to 90 degrees, making it difficult to accurately classify them within these two categories. The same issue is observed for symbols oriented at 0 and 45 degrees. The same issue arises for the symbols that are oriented to the right side.

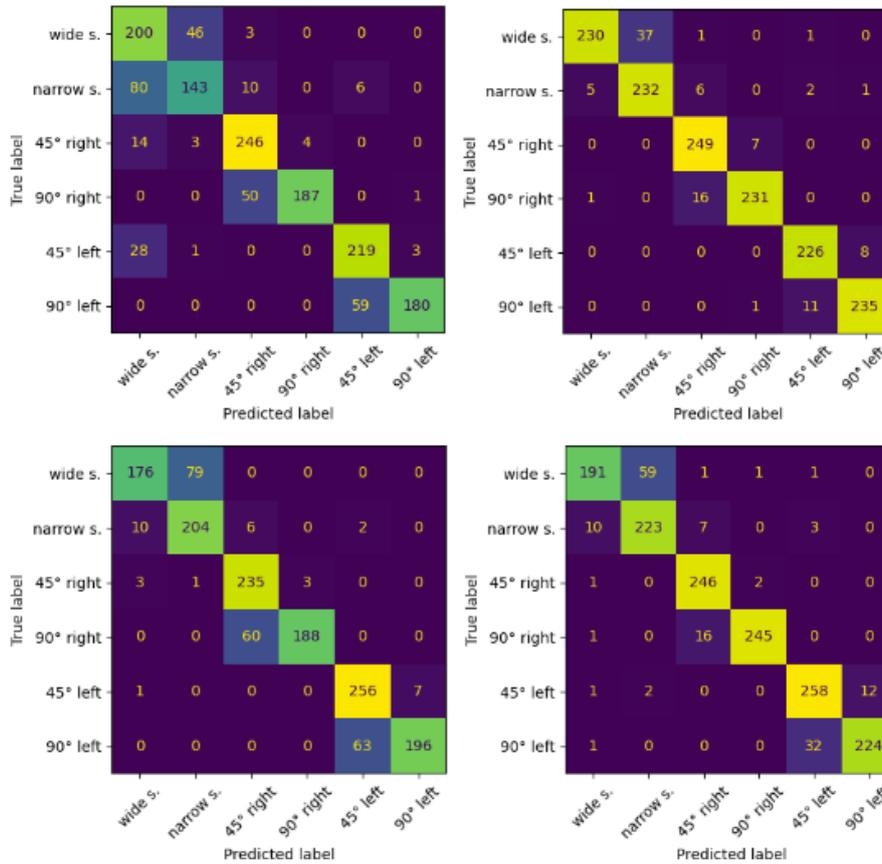


Figure 19. Base model confusion matrix by symbol type. Top right advance to contact, top left attack, bottom right counterattack and bottom left main attack - base model results confusion matrices.

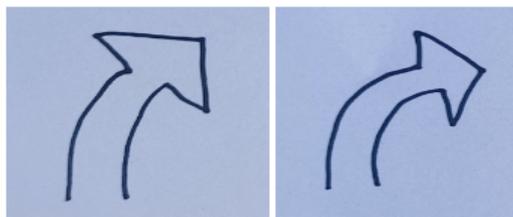


Figure 20. The left image shows 45 degrees right and the right image shows 90 degrees right.

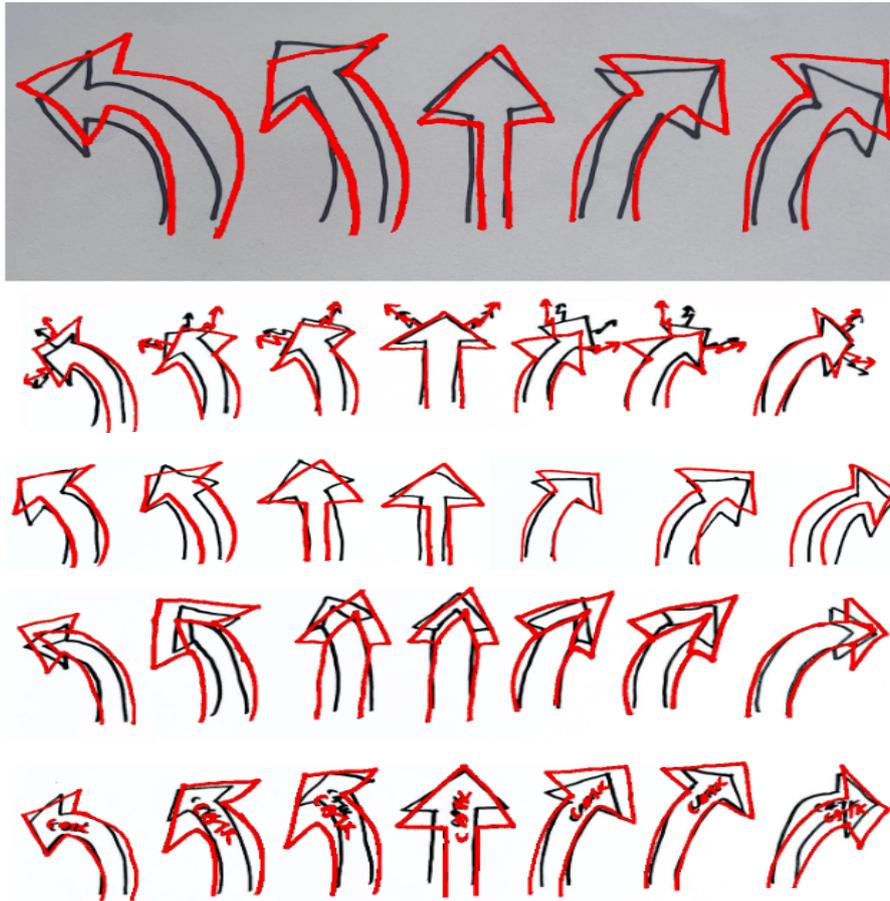


Figure 21. Base model results. Original input image (black) is overlaid by a template for the detected class (red).

Distinguishing between symbols with different shapes poses a challenge to their accurate classification. These symbols have been grouped into six classes based on their orientation, including 45 and 90 degrees left and right, as well as narrow and wide straight symbols. The model demonstrates good precision in predicting the classification of the six symbol subclasses, as evidenced by the high accuracy rates reported for each class.

Figure 21 shows that the model can capture the direction of the symbols quite well, with the overlaid class template providing a good indication of their orientation. It is worth noting that in many cases, it is not even clear to which subclass an arrow should belong, even for human labelers. While the model captures the direction of the arrows very well, the overlaid class template can sometimes be slightly off, particularly at the

tip of the arrow. This is an important consideration as the tip of the arrow is a critical element for meaning.



Figure 22. Base model detection on real clean data.

While the model's accuracy on the real clean data was 68%, which is lower compared to the 87% accuracy achieved on the synthetic generated data, it still demonstrates some degree of accuracy. As in the real clean data there are only straight arrows. The misclassification came mainly from wide and narrow straight symbols mainly. A few symbols were classified as 45 or 90 degrees left or right. Moreover, the fact that there is a significant gap between the two datasets highlights the challenges of working with real-world data, particularly in terms of its quality and diversity. Despite this limitation, the results show that there is potential for improving the accuracy of the model on real data through further refinement and optimization. Therefore, it is crucial to continue exploring strategies to improve the accuracy of the model, particularly when working with real-world scenarios.

4.3.2 Beta variational autoencoder model results

Evaluating autoencoder models using the confusion-matrix and class accuracy method is not a natural way of assessing their performance. This is because autoencoders are not trained to perform classification tasks, but rather to output compressed representations of input data. To compare the performance of autoencoder models with baseline models, we construct a way to turn these compressed representations into labels. However, these labels are not actually used when using the variational Autoencoder (VAE) approach.

What really matters is how well the VAE approach captures the shapes of the symbols. Ideally, one would like to have quantitative measures of how close the keypoints, particularly the arrowhead, are to the original symbols, but this would require labeling the test dataset with keypoints, which there is no time for at the moment. Instead, we rely on qualitative observations and visual comparisons of black and red symbols overlaid, for all four symbols.

The process of evaluating and comparing the autoencoder model with the baseline model involves inserting an image, extracting its compressed representation, and then measuring the distance between this representation and the labeled symbols in the database. This is done using a cosine similarity measure, which allows us to identify the most similar labeled symbol to the inserted image.

The VAE developed in this study achieved an overall accuracy of 61% in classifying the six different subclasses. However, the model exhibited some challenges in correctly distinguishing between certain subclasses. Specifically, the most common errors were between wide and narrow straight, 45 degrees and 90 degrees left, and 45 degrees and 90 degrees right. Additionally, the models often predicted 45 and 90-degree left and right symbols as straight symbols, leading to misclassifications.

To address these issues, the impact of introducing beta hyperparameters to the VAE model was explored. Several beta hyperparameter values were experimented with, including 3, 10, and 50, to evaluate their effects on the model's accuracy.

Results showed that the inclusion of beta hyperparameters led to a modest improvement in the model's accuracy. When the beta hyperparameter was set to 3, the VAE model achieved an accuracy of 64%. However, the accuracy slightly decreased to 63% when the beta hyperparameter was set to 10. The most significant improvement in accuracy was achieved when the beta hyperparameter was set to 50, resulting in an accuracy of 66%. These findings indicate that beta hyperparameters can enhance the performance of VAE models in classifying hand-drawn symbols accurately. Moreover, the significant improvement in accuracy achieved with a beta hyperparameter value of 50 suggests that further exploration of hyperparameters could lead to even better performance.

Upon closer examination of the trajectory symbol classification results, it becomes clear that the accuracy varies significantly between different symbol types. Results indicate that advance to contact and counterattack symbols achieved significantly better accuracy, with 75% and 70%, respectively, compared to attack and main attack symbols, which had accuracy levels of 62% and 58%, respectively.

Upon analyzing the accuracy of advance to contact symbols in study, what was found is that the main problem lies in misclassifying wide and narrow straight symbols, as well as 45 and 90 degrees left and right symbols. Although these misclassifications are noteworthy, they are relatively less problematic than those observed in other symbol types. In particular, analysis revealed that other symbol types experienced more significant issues with classification accuracy, with a higher number of 45 and 90 degree symbols being

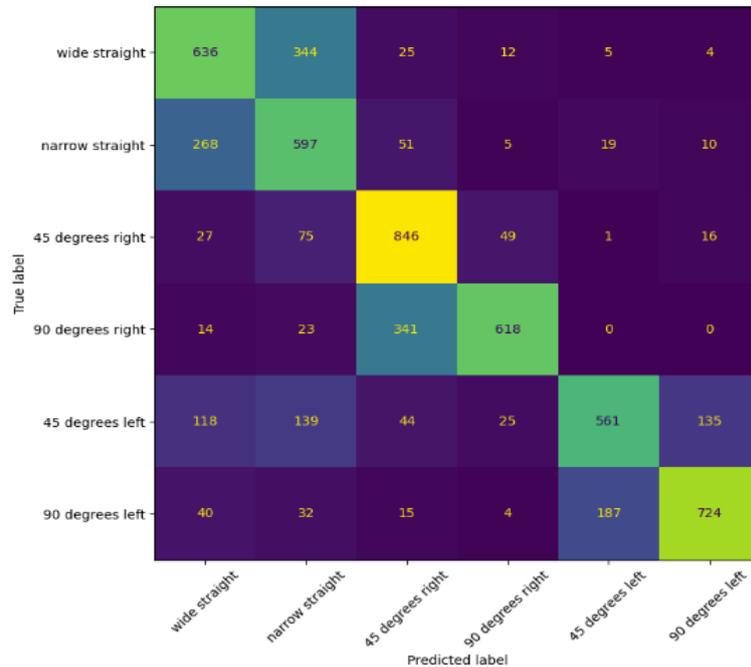


Figure 23. Confusion matrix of beta variational autoencoder with beta hyperparameter set to 50.

classified as straight symbols. This suggests that the accuracy of symbol classification is dependent on a variety of factors, including the complexity and variety of symbols within each symbol type.

Although beta VAE is not perfect, figure 25 highlights the effectiveness of beta VAE in identifying and retrieving similar images from its database. A comparison of figure 21 to figure 25 indicates that beta VAE is able to identify a significantly greater number of similar symbols when compared to the base classification model. This is because while the base classification model maps each input to one of 6 classes each represented by one representative template, beta VAE finds the most similar symbol within an arbitrarily large and diverse dataset of labeled templates. Not all diversity can be summarized in the 6 images the base model can return, but by extending the labeled templates set for VAE approach, wider variability of symbols can captured. While there is not numerically evaluation on this, it seems that arrowheads point to a more correct location in case of VAE outputs compared to the baseline model (red and black tips of arrows in fig. 25 vs 21). Arrowheads of these symbols point to specific locations on the map toward which the action should be conducted and have high significance.

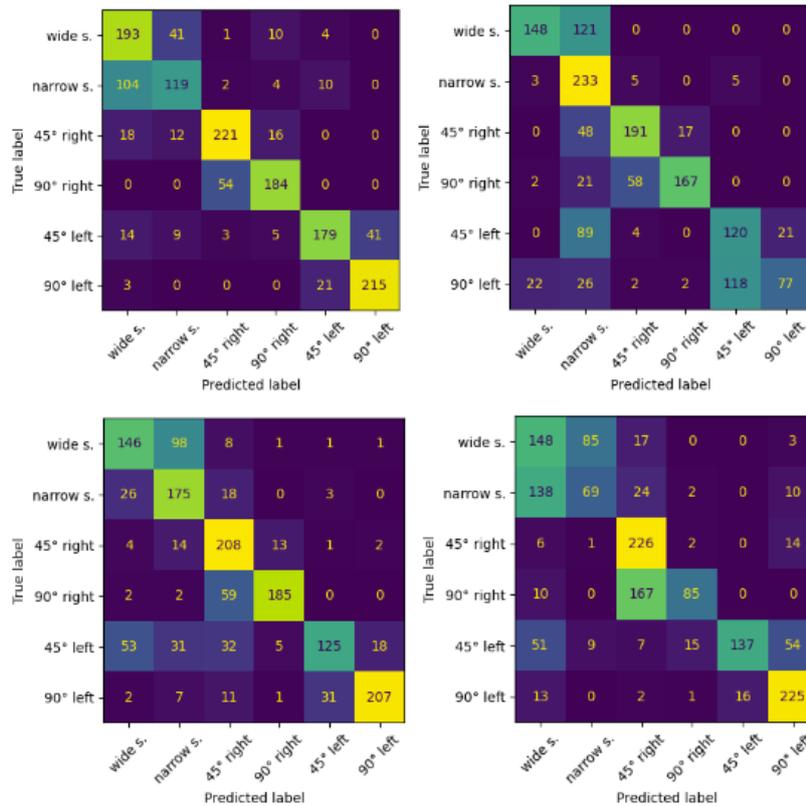


Figure 24. Beta VAE confusion matrix by symbol type. Top right advance to contact, top left attack, bottom right counterattack and bottom left main attack - VAE model results confusion matrix.

The effectiveness of beta VAE models in detecting NATO mission task symbols trajectory was also evaluated using real data from the Estonian National Defence College. The dataset included advance to contact, counterattack, and main attack symbols, but did not include any attack symbols. Additionally, all symbols in the dataset were either narrow or wide straight.

Evaluating beta VAE models on the clean real clean data from Estonian National Defence College, which contained advance to contact, counterattack and main attack. It did not contain attack symbols. Also all the symbols were either narrow or wide straight. The beta VAE models were able to achieve an overall accuracy of 64% on this clean real-world dataset. This is a promising result, indicating that the models are able to effectively identify mission task symbols in near real-world scenarios. It is also important to note that the dataset used in this evaluation was more complex than the synthetic datasets

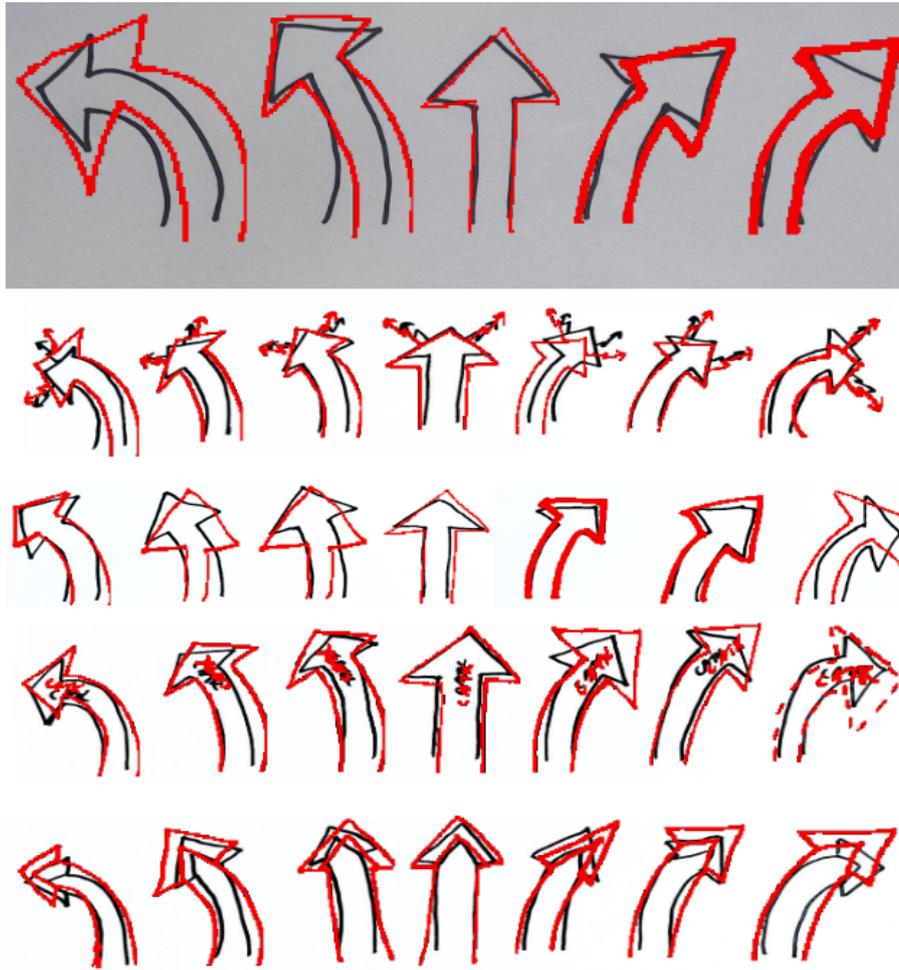


Figure 25. Beta variational autoencoder models results. Original input image (black) is overlaid by the nearest neighbor in the reference set for the detected class (red).



Figure 26. Beta variational autoencoder model detection on real clean data.

used in previous evaluations, further demonstrating the effectiveness and versatility of beta VAE models. It is important to keep in mind that the dataset used for evaluating the beta VAE models only contained narrow and wide straight symbols, and did not include any 45 or 90 degrees left or right symbols. Therefore, while the achieved 68% overall accuracy is promising, it is limited in its ability to provide a complete picture of the model's performance.

5 Discussion

Better results were achieved in detecting rotations than initially anticipated in synthetic generated data. The primary objective was to develop models capable of detecting rotation errors of under 30 degrees in 80% of the examples. However, the models surpassed this goal by successfully detecting under 30 degree errors in 95.57% of the examples. On clean real data, the findings were not as satisfactory, with 72.41% of the symbols being classified with an error under 30 degrees. The values on real data did not meet the initial expectations.

Additionally, it is worth mentioning that during evaluation on synthetic generated data, the mean degree error across all 31 mission task symbol types was only 11.65 degrees. This result suggests that our models were consistently accurate across a diverse range of symbol types, which demonstrates their robustness and effectiveness. However, our evaluation on clean real data yielded a mean degree error on rotations of 31.80 degrees.

It is important to note that for 9 out of 17 symbol types had mean degree error lower than 30 and also 9 out of 17 symbol types, the 80% accuracy threshold was achieved, demonstrating the potential of the models to accurately classify symbols. Generalization issues were only observed for a few symbol types, which could be attributed to factors such as handwriting variability and symmetry, where a 90-degree rotation yields essentially the same picture. In fact, rotation barely matters for circular (e.g. control, deny, isolate, occupy, retain and secure) and X-shaped symbols (e.g. destroy, neutralize and suppress). The interpretation of data plays a significant role in achieving accurate results. Specifically, when examining the advance to contact, block, and clear symbol types, it is evident that the models can perform better on real clean data as compared to generated data. For instance, the mean degree error for advance to contact on generated data is 10.15 degrees, whereas on real clean data, it is 9.6 degrees. Similarly, for the block symbol type, the mean degree error on generated data is 12.39 degrees, whereas on real clean data, it is 8.8 degrees. The same can be said for clear as, mean degree error for clear on generated data is 8.82 degrees, whereas on real clean data, it is 4.4 degrees. These results demonstrate the potential to achieve better accuracy on real data. By further refining and optimizing our models, we can improve their performance in identifying and analyzing mission task symbols in real-world scenarios.

For the trajectory detection the base model achieved a better result than the VAE models when summarizing results at the level of six subcategories, however the VAE model is much more versatile than the base model and has the promise to match keypoints with higher accuracy in the long run.

There are big differences between accuracies between symbol types. These differ-

ences suggest that certain symbol types may be more challenging to classify accurately than others, and that developing specialized models or training approaches may be necessary to achieve high accuracy in symbol classification. Furthermore, the variability in accuracy between different symbol types highlights the importance of considering the specific context and characteristics of the symbols being classified when designing and training deep learning models. Additionally, identifying the factors contributing to the differences in accuracy between symbol types can provide insights into the underlying patterns and features that are most important for accurate classification.

It is crucial to recognize that, in practice, a human will still need to post-process the results and make adjustments to the rotation or keypoints to some degree. With that being said, the achieved results are still very promising and can be extremely helpful and time-saving. The fact that the baseline model can place the attack close enough is already a significant advantage, but the more versatile autoencoder model offers even more potential benefits. By providing a more accurate initial estimate, it can greatly reduce the amount of time and effort required for manual post-processing, making it a valuable tool.

The results demonstrated that the methods are applicable and promising and the main limitation of our study lies in the availability of labeled real-world data with good quality. Obtaining high-quality real-world data can be challenging, as it often requires significant resources and can be subject to various constraints, such as privacy or security concerns. Additionally, the variability and complexity of real-world data can make it more difficult to develop accurate models that generalize well. Therefore, it is important to continue to explore ways to obtain and use high-quality real-world data in order to further improve the accuracy and applicability of our models.

These results are a significant achievement in our research, indicating that our models can become highly accurate in detecting rotations of the NATO mission task symbols if a representative training dataset exists. The model's accuracy level demonstrates the effectiveness of our approach in developing deep learning models to identify and analyze hand-drawn symbols accurately. The high level of accuracy has important practical implications as it enhances the ability to detect and recognize the mission task symbols, which can be critical in military operations. Moreover, the success of our models in detecting rotation errors can have broader implications for the development of deep learning models and their application in various domains where accurate identification of symbols or objects is crucial.

6 Conclusions

In conclusion, our study aimed to explore the potential of machine learning models in recognizing and interpreting mission task symbols rotation and poses. We investigated several methods for symbol pose detection, including the use of a CNN-based approach for rotation, as well as baseline CNN-based and autoencoder-based approaches for pose detection. These approaches were evaluated based on their performance on both synthetic and real clean data.

Experiments revealed promising results for rotation models using synthetic and real clean data. What was demonstrated is that both baseline and autoencoder-based approaches can accurately capture the shapes of these symbols and identify subclasses. Furthermore, our study highlights the potential of VAE models in symbol classification tasks, while also acknowledging their limitations. The results showed that the accuracy of the VAE model can be improved by tuning the beta hyperparameters. However, our findings suggest that the accuracy of trajectory symbol classification can vary significantly between different symbol types, indicating the importance of considering unique characteristics and contexts when developing classification models and training approaches. It is also important to emphasize that the close enough baseline model is a valuable tool in saving time and that the more versatile autoencoder can provide even more help in symbol recognition tasks.

As for future work, the study suggests that exploring the potential of autoencoders in reconstructing cleaned versions of symbols from real messy data could be a promising avenue. As real-world data can be with lots of background garbage. Generate canvases with and without background garbage and making autoencoder reproduce cleans from noisy. So there is a potential it can work on real messy data.

References

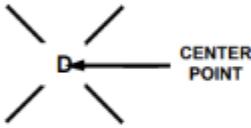
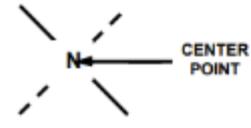
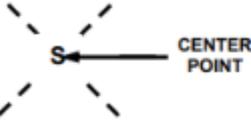
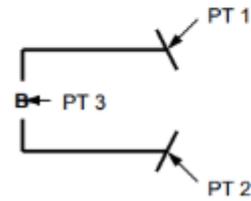
- [AC11] APP-6(C). Nato joint military symbology, 2011.
- [AGM21] Shivam Aggarwal, Safal Gaur, and Dr Manju. *Text Document Orientation Detection Using Convolutional Neural Networks*, pages 153–164. 05 2021.
- [BHP⁺18] Christopher P. Burgess, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in β -vae, 2018.
- [CWF⁺15] Li Chen, Song Wang, Wei Fan, Jun Sun, and Satoshi Naoi. Beyond human recognition: A cnn-based framework for handwritten character recognition. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 695–699, 2015.
- [DKCA21] Aditya Dixit, Ramesh Kumar Chidambaram, and Zaheer Allam. Safety and risk analysis of autonomous vehicles using computer vision and neural networks. *Vehicles*, 3(3):595–617, 2021.
- [Doe21] Carl Doersch. Tutorial on variational autoencoders, 2021.
- [EEVG⁺15] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, January 2015.
- [EJAG20] Eyad Elyan, Laura Jamieson, and Adamu Ali-Gombe. Deep learning for symbols detection and classification in engineering drawings. *Neural Networks*, 129:91–102, 2020.
- [FDB15] Philipp Fischer, Alexey Dosovitskiy, and Thomas Brox. Image orientation estimation with convolutional networks. In Juergen Gall, Peter Gehler, and Bastian Leibe, editors, *Pattern Recognition*, pages 368–378, Cham, 2015. Springer International Publishing.
- [FMMW21] Miroslav Fil, Munib Mesinovic, Matthew Morris, and Jonas Wildberger. Beta-vae reproducibility: Challenges and extensions, 2021.
- [GGSS19] Adhesh Garg, Diwanshi Gupta, Sanjay Saxena, and Parimi Praveen Sahadev. Validation of random dataset using an efficient cnn model trained on mnist handwritten dataset. In *2019 6th International Conference on Signal Processing and Integrated Networks (SPIN)*, pages 602–606, 2019.

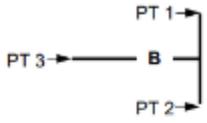
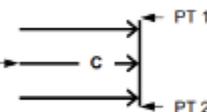
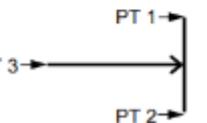
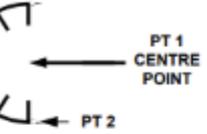
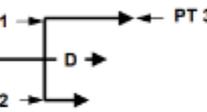
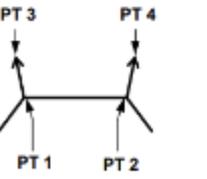
- [GYLP18] Junfeng Gao, Yong Yang, Pan Lin, and Dong Sun Park. Computer vision in healthcare applications. *Journal of Healthcare Engineering*, 2018:5157020, Mar 2018.
- [JCS⁺22] Glenn Jocher, Ayush Chaurasia, Alex Stoken, Jirka Borovec, NanoCode012, Yonghye Kwon, Kalen Michael, TaoXie, Jiacong Fang, imyhxy, Lorna, Zeng Yifu, Colin Wong, Abhiram V, Diego Montes, Zhiqiang Wang, Cristi Fati, Jebastin Nadar, Laughing, UnglvKitDe, Victor Sonck, tkianai, yxNONG, Piotr Skalski, Adam Hogan, Dhruv Nair, Max Strobel, and Mrinal Jain. Tips for Best Training Results, May 2022.
- [KST⁺21] Efstratios Kakaletsis, Charalampos Symeonidis, Maria Tzelepi, Ioannis Mademlis, Anastasios Tefas, Nikos Nikolaidis, and Ioannis Pitas. Computer vision for autonomous uav flight safety: An overview and a vision-based safe landing pipeline example. *ACM Comput. Surv.*, 54(9), oct 2021.
- [KW19] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.
- [LMB⁺14] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham, 2014. Springer International Publishing.
- [MB20] Subhadip Maji and Smarajit Bose. Deep image orientation angle detection, 2020.
- [PGH⁺16] Yunchen Pu, Zhe Gan, Ricardo Henao, Xin Yuan, Chunyuan Li, Andrew Stevens, and Lawrence Carin. Variational autoencoder for deep learning of images, labels and captions. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.
- [PHSS22] Pablo N. Pizarro, Nancy Hitschfeld, Ivan Sipiran, and Jose M. Saavedra. Automatic floor plan analysis and recognition. *Automation in Construction*, 140:104348, 2022.
- [RDD19] Rosemberg Rodriguez, Eva Dokladalova, and Petr Dokladal. Rotation invariant cnn using scattering transform for image classification. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 654–658, 2019.

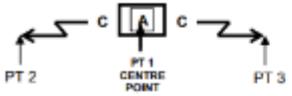
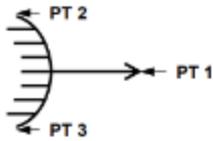
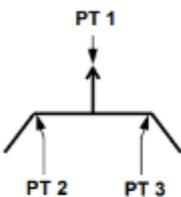
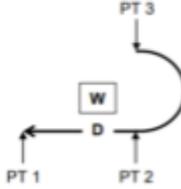
- [RDGF16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.
- [Ros20] Adrian Rosebrock. Autoencoders for content-based image retrieval with keras and tensorflow, Mar 2020.
- [Sae17] Daniel Saez. Correcting image orientation using convolutional neural networks, Jan 2017.
- [SJMS17] Tomas Simon, Hanbyul Joo, Iain Matthews, and Yaser Sheikh. Hand keypoint detection in single images using multiview bootstrapping. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.
- [WHX⁺13] Pengcheng Wu, Steven C.H. Hoi, Hao Xia, Peilin Zhao, Dayong Wang, and Chunyan Miao. Online multimodal deep similarity learning with application to image retrieval. In *Proceedings of the 21st ACM International Conference on Multimedia, MM '13*, page 153–162, New York, NY, USA, 2013. Association for Computing Machinery.
- [YPS⁺18] Yin, Jun, Pan, Huadong, Su, Hui, Liu, Zhonggeng, and Peng, Zhirong. A fast orientation invariant detector based on the one-stage method. *MATEC Web Conf.*, 232:04036, 2018.
- [ZCT21] Jing Zhang, Zhe Chen, and Dacheng Tao. Towards high performance human keypoint detection. *International Journal of Computer Vision*, 129(9):2639–2662, Sep 2021.
- [ZZK23] Longfei Zhou, Lin Zhang, and Nicholas Konz. Computer vision techniques in manufacturing. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 53(1):105–117, 2023.

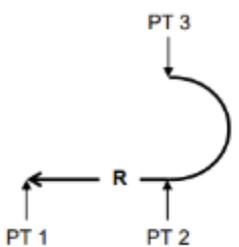
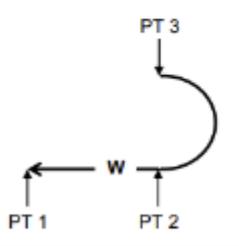
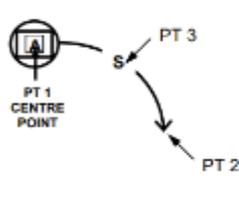
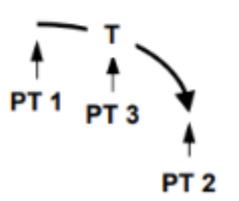
Appendix

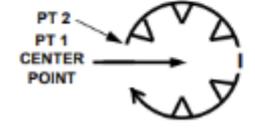
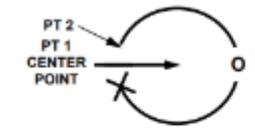
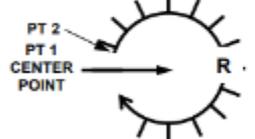
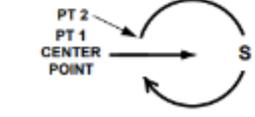
I. Requirements table

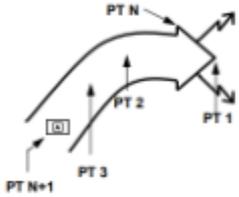
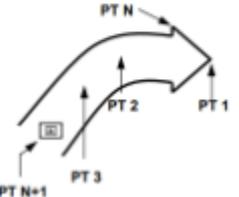
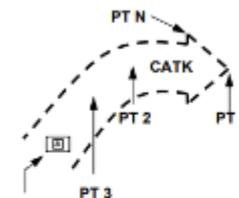
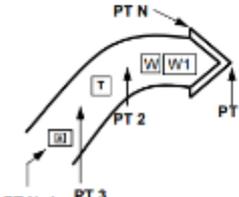
Symbol	Requirements needed to recreate	Metrics to validate
Grouping 1	Location only needed	
Destroy 	<ul style="list-style-type: none"> • Symbol ID • Location 	<ul style="list-style-type: none"> • Intersection over union(IoU) accuracy
Neutralize 	<ul style="list-style-type: none"> • Symbol ID • Location 	<ul style="list-style-type: none"> • Intersection over union(IoU) accuracy
Suppress 	<ul style="list-style-type: none"> • Symbol ID • Location 	<ul style="list-style-type: none"> • Intersection over union(IoU) accuracy
Grouping 2	Location and rotation needed	
Breach 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction 	<ul style="list-style-type: none"> • Intersection over union(IoU) accuracy • The degree of direction difference from the direction marked by a person.

<p>Block</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction 	<ul style="list-style-type: none"> • Intersection over union(IoU) accuracy • The degree of direction difference from the direction marked by a person.
<p>Clear</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction 	<ul style="list-style-type: none"> • Intersection over union(IoU) accuracy • The degree of direction difference from the direction marked by a person.
<p>Penetrate</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction 	<ul style="list-style-type: none"> • Intersection over union(IoU) accuracy • The degree of direction difference from the direction marked by a person.
<p>Contain</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction 	<ul style="list-style-type: none"> • Intersection over union(IoU) accuracy • The degree of direction difference from the direction marked by a person.
<p>Disrupt</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction 	<ul style="list-style-type: none"> • Intersection over union(IoU) accuracy • The degree of direction difference from the direction marked by a person.
<p>Support by fire</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction 	<ul style="list-style-type: none"> • Intersection over union(IoU) accuracy • The degree of direction difference from the direction marked by a person.

<p>Cover</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction 	<ul style="list-style-type: none"> • Intersection over union(IoU) accuracy • The degree of direction difference from the direction marked by a person.
<p>Guard</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction 	<ul style="list-style-type: none"> • Intersection over union(IoU) accuracy • The degree of direction difference from the direction marked by a person.
<p>Screen</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction 	<ul style="list-style-type: none"> • union(IoU) accuracy • The degree of direction difference from the direction marked by a person.
<p>Ambush</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction 	<ul style="list-style-type: none"> • union(IoU) accuracy • The degree of direction difference from the direction marked by a person.
<p>Attack by fire</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction 	<ul style="list-style-type: none"> • union(IoU) accuracy • The degree of direction difference from the direction marked by a person.
<p>Delay</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction 	<ul style="list-style-type: none"> • union(IoU) accuracy • The degree of direction difference from the direction marked by a person.

<p>Retire</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction 	<ul style="list-style-type: none"> • union(IoU) accuracy • The degree of direction difference from the direction marked by a person.
<p>Withdraw</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction 	<ul style="list-style-type: none"> • union(IoU) accuracy • The degree of direction difference from the direction marked by a person.
<p>Fix</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction 	<ul style="list-style-type: none"> • union(IoU) accuracy • The degree of direction difference from the direction marked by a person.
<p>Seize</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction 	<ul style="list-style-type: none"> • union(IoU) accuracy • The degree of direction difference from the direction marked by a person.
<p>Turn</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction 	<ul style="list-style-type: none"> • union(IoU) accuracy • The degree of direction difference from the direction marked by a person.

<p>Control</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction 	<ul style="list-style-type: none"> • union(iou) accuracy • The degree of direction difference from the direction marked by a person.
<p>Deny</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction 	<ul style="list-style-type: none"> • union(iou) accuracy • The degree of direction difference from the direction marked by a person.
<p>Isolate</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction 	<ul style="list-style-type: none"> • union(iou) accuracy • The degree of direction difference from the direction marked by a person.
<p>Occupy</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction 	<ul style="list-style-type: none"> • union(iou) accuracy • The degree of direction difference from the direction marked by a person.
<p>Retain</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction 	<ul style="list-style-type: none"> • union(iou) accuracy • The degree of direction difference from the direction marked by a person.
<p>Secure</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Direction 	<ul style="list-style-type: none"> • union(iou) accuracy • The degree of direction difference from the direction marked by a person.

Grouping 3	Location and pose needed	
<p>Advance to contact</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Shape • Direction 	<ul style="list-style-type: none"> • The Euclidean distance of PT1 from the human-marked point. • The degree of direction difference from the direction marked by a person. • Qualitative - Similarity of the trajectory
<p>Attack</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Shape • Direction 	<ul style="list-style-type: none"> • The Euclidean distance of PT1 from the human-marked point. • The degree of direction difference from the direction marked by a person. • Qualitative - Similarity of the trajectory
<p>Counterattack</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Shape • Direction 	<ul style="list-style-type: none"> • The Euclidean distance of PT1 from the human-marked point. • The degree of direction difference from the direction marked by a person. • Qualitative - Similarity of the trajectory
<p>Main Attack</p> 	<ul style="list-style-type: none"> • Symbol ID • Location • Size • Shape • Direction 	<ul style="list-style-type: none"> • The Euclidean distance of PT1 from the human-marked point. • The degree of direction difference from the direction marked by a person. • Qualitative - Similarity of the trajectory

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Karl-Kristjan Kõverik**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,
NATO standard mission task symbols pose detection based on symbol requirements,
supervised by Ardi Tampuu.
2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.
3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.
4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Karl-Kristjan Kõverik

09/05/2023