Gezamenlijke opleiding UHasselt en KU Leuven

Faculteit Industriële Ingenieurswetenschappen

master in de industriële wetenschappen: elektromechanica

Masterthesis

Affordances-based recognition of assembly activities through probabilistic modeling

Joren Colson
Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektromechanica

PROMOTOR :
Prof. dr. ir. Eric DEMEESTER

BEGELEIDER :
ing. Martijn CRAMER
Mevrouw Yanming WU

2022
2023

Masterthesis

Affordances-based recognition of assembly activities through probabilistic modeling

Joren Colson

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektromechanica

PROMOTOR :

Prof. dr. ir. Eric DEMEESTER

BEGELEIDER :

ing. Martijn CRAMER

Mevrouw Yanming WU

Preface

I am pleased to present my master's thesis titled “Affordances-based Recognition of Assembly Activities through Probabilistic Modelling”. This research represents the culmination of my study in the joint programme in Engineering Technology at UHasselt and KU Leuven. Conducted on behalf of the ACRO research group, this thesis explores the use of object affordances for recognising operator activities in an assembly task through probabilistic modelling.

I am deeply grateful to several individuals who have played significant roles in the realisation of this research and the completion of this thesis. Foremost, I would like to express my sincere appreciation to my esteemed promotor and supervisors, Prof. dr. ir. Demeester, ing. Martijn Camer, and ir. Yanming Wu, for their invaluable guidance, expertise, and continuous support throughout this journey. Their insightful feedback and exceptional mentorship have been instrumental in shaping this work.

To the reader, I hope this thesis provides you with an engaging exploration of my research and I invite you to delve into the pages that follow.

Table of contents

List of tables

List of figures

Nomenclature

6DOF	six degrees of freedom
ACRO	Automation, Computer Vision and RObotics
DC	Distributional Clauses
HAR	human activity recognition
HIM	Human Interface Mate
IDE	integrated development environment
HMM	hidden Markov model
SVM	support vector machine
k-NN	k-nearest neighbours
PGM	probabilistic graphical model
DGM	directed graphical model
UGM	undirected graphical model
BN	Bayesian network
CPD	conditional probability distribution table
DBN	dynamic Bayesian network
TFP	TensorFlow Probability
PLL	probabilistic logic languages
MCMC	Markov chain Monte Carlo
NUTS	No-U-Turn Sampler
PPF	Point Pair Feature
PCL	Point Cloud Library

Abstract

This master's thesis, conducted within the ACRO research group, focuses on enhancing flexibility and robustness of human-robot collaboration. For this collaboration to work fluently, the robot needs to recognise the activity performed by the operator. Specifically, this research explores the use of object affordances for recognising operator activities in an assembly task through probabilistic modelling. Related studies use mostly human information, which lacks in flexibility and robustness.

The objective of this research is threefold. Firstly, the operator needs to be free to position the different assembly parts as desired and assemble the product wherever (s)he likes. Secondly, the system needs to be usable by multiple operators. Finally, the success rate aims to be around 78 per cent, which is the average success rate of similar conducted research. Initially, different probabilistic modelling tools are compared. Next, a Dynamic Bayesian network is modelled that is able to infer the assembly activity performed based on the relative distance, motion direction, and velocity of the assembly parts. These input variables are deduced from the position of the objects by detecting ArUco markers in the video that captures assembly operations.

The final model works independent of the initial position of the assembly parts, is usable by multiple operators, and achieves a success rate of 75 per cent. This research also did a first attempt to extract 6D object pose from RGB-D data using Point Pair Feature matching method.

Abstract in het Nederlands

Deze masterproef, uitgevoerd binnen de onderzoeksgroep ACRO, richt zich op het verbeteren van flexibiliteit en robuustheid van mens-robot samenwerking. Voor een vloeiende samenwerking moet de robot de activiteit die door de operator wordt uitgevoerd herkennen. Specifiek bespreekt dit onderzoek het gebruik van *object affordances* om operatoractiviteiten bij assemblagetaken te herkennen via probabilistische modellering. Gerelateerd onderzoek maakt voornamelijk gebruik van menselijke informatie, wat flexibiliteit en robuustheid mist.

Dit onderzoek heeft drie doelstellingen. De operator moet vrij zijn om de assemblagedelen naar wens te positioneren en het product eender waar te monteren, het systeem moet bruikbaar zijn door meerdere operators en men streeft naar een succespercentage van ongeveer 78%, het gemiddelde succespercentage van soortgelijk onderzoek. Aanvankelijk worden verschillende probabilistische modelleringsmodellen met elkaar vergeleken. Vervolgens wordt een dynamisch Bayesiaans netwerk gemodelleerd voor het afleiden van de uitgevoerde assemblageactiviteit op basis van de relatieve afstand, bewegingsrichting en snelheid van de onderdelen. Deze invoerparameters worden afgeleid aan de hand van ArUco-markers.

Het finale model werkt onafhankelijk van de oorspronkelijke positie van de onderdelen, is bruikbaar door meerdere operators en behaalt een succespercentage van 75%. Dit onderzoek deed ook een eerste poging in het afleiden van de 6D poses uit RGB-D data door het gebruik van de *Point Pair Feature matching*-methode.

1 Introduction

1.1 Context

This master's thesis is conducted on behalf of the ACRO research group. The acronym ACRO stands for Automation, Computer vision and RObotics. ACRO therefore studies these fields in depth. Examples of the research topics are: human-robot interaction and collaboration, collision-free trajectory generation and navigation and functional programming for robotics in the cloud. This research group belongs to the Department of Mechanical Engineering of KU Leuven and is located in the Technology Centre at Diepenbeek Campus, see Figure 1 [1].

Figure 1: Technology Center in Diepenbeek [1]

This research belongs to ACRO's robotics branch, more specifically: human-robot interaction and collaboration. The branch consists of six professors and 13 researchers. In this topic, a cobot (i.e., collaborative robot) is used. This cobot makes human-robot collaboration more efficient because humans and cobots are capable of assisting each other while performing collaborative tasks. This is a major difference from the past where one had to safely lock the robot behind fences or in cages [2]. In this field, ing. Martijn Cramer is active with his doctoral research, "Towards intention-based human-robot collaborative manufacturing". Here, Mr. Cramer strives for a more flexible collaboration between humans and cobots. For example, in an assembly process, the robot will constantly try to detect and identify the operator's activities and intentions and then respond appropriately in order to provide optimal assistance to the operator. This makes it possible for the operator to deviate from his/her predefined task sequence. Recognising operator actions and intentions can be done by tracking the operator's movements (i.e., registering the skeleton data) [3].

Another ongoing doctoral research at ACRO is "6D object pose tracking", conducted by ir. Yanming Wu. The goal of her research is to estimate the 3D translation and rotation of an object with respect to the camera coordinate system. Applications include scene understanding, augmented reality, robot control and navigation, and human-machine interaction [4].

This master's thesis searches for an opportunity to link the two previously mentioned doctoral studies: the 6D pose tracking algorithms from Ms. Wu's research should be used as an input for recognising assembly activities in Mr. Cramer's research. This could give an improvement in flexibility and robustness in comparison with e.g. the usage of skeleton data. These improvements will be elaborated in sections 1.2 and 1.3.

1.2 Problem statement

Flexibility is a very important term in today's industry. By making the production process as flexible as possible, a manufacturing company can respond to individual requirements of different customers as quickly as possible. Focused on this research, flexibility can be seen as the ability to assemble a product in different ways, depending on the operator's preference; or the ability to assemble a product independently of the location and rotation of the different parts on the workbench.

Today, some technologies are already being used in the industry to guide operators in performing assembly tasks. An example of this is the Human Interface Mate (HIM), shown in Figure 2, which is developed by the Belgian tech company Arkite. HIM guides the operator during assembly by using a projector that projects instructions onto the workbench. A depth camera checks the presence of all components and tools, and whether they are in their expected locations. In addition, the HIM checks that all required actions in the assembly process are performed properly and in the required order. The system requires all products to be located in fixed, predefined 3D volumes in order to detect their corresponding assembly activities. This of course creates a limitation on the flexibility of the system [5].

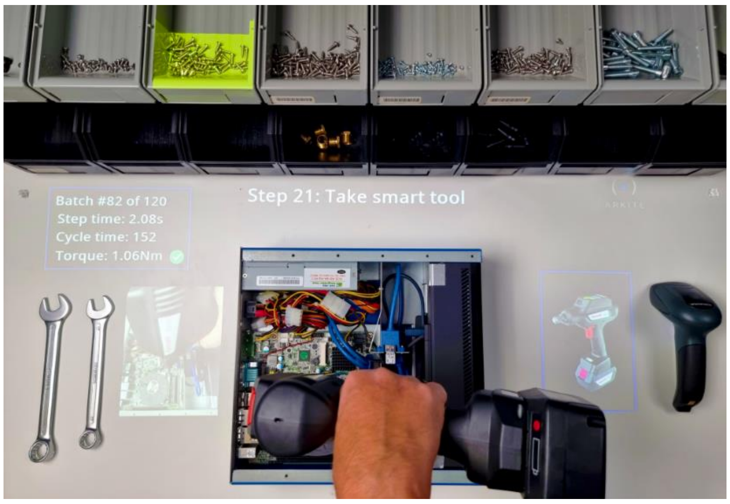


Figure 2: Projections provided by the HIM on the operator's workbench [5]

There is also a lot of research conducted into state-of-the-art methods for activity recognition, including skeleton-based activity recognition. This method focuses only on the operator's movements. Certain points on the body of the operator will be observed and based on the movements of these points, the operator's action is identified. In the scenario of assembling a product on a worktable, these points will mainly be placed on the hands so that the actions can be identified as accurately as possible. An example of this can be seen in Figure 3. This method is not ideal for the discussed application since a lot of information is lost by looking only at the movements of the operator and not at the assembly parts that are manipulated [6]. It is basically an indirect way of "sensing" the activity. Since different operators perform the same actions in different ways, just using skeleton data does not seem to be the most robust solution either [2].

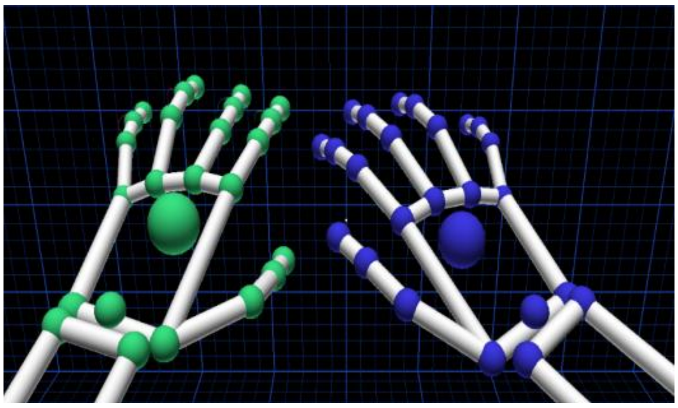


Figure 3: A representation of how skeleton data is used to register the movements of the hands by tracking specific points of the hand [6]

It is possible to recognise the activity that is performed by using the valuable information that the assembly parts offer during assembly. This is done by using object affordances. Affordance includes three variables: objects, actions, and effects. Using an affordance model, when two variables are known, the third can be predicted. For example, a robot observes assembly parts and their poses (objects) lying separately on a table. Next, the robot tracks the relative movements of the assembly parts (effect). From this, the robot can predict that the person is performing an assembly activity (action). This concept suggests the possibility of estimating assembly activities in real-time by looking at the relative displacements of the different parts during assembly [7].

1.3 Objectives

Continuing on what was discussed in section 1.2, the main goal is a more flexible and robust approach for recognising operator actions. Since the goal of this research is to improve activity recognition by using object and pose information as features instead of other types of data, the targeted success rate should be at least the same as (but preferably higher than) those of researchers who have utilised "indirect" ways of sensing the operator's activity. In Roitberg et al.'s research, skeleton data was used as features and the overall success rate was 78 per cent [8]. This research aims for a similar success rate.

One of the objectives of the system consist of enabling the operator to position the different assembly parts on the worktable as desired and assemble the final product wherever s/he likes.

This makes it clear that the HIM is not the most optimal solution for this application, since the assembly parts need to be in fixed predefined areas.

Another objective is that the system should not be biased towards a specific operator, but instead should perform equally accurate, independent of the operator that is using it. The disadvantage of employing skeleton data for activity recognition is that it only takes into account the operator's actions and does not use information from what happens to the assembly parts. Different operators can perform certain actions in different ways. If the action is thus performed differently from when the system is trained, the action may not be recognised by the system. This is where the method lacks robustness.

1.4 Methods

As stated earlier, this thesis tries to link the doctoral research of Mr. Cramer and Ms. Wu. Figure 4 gives a schematic overview of the process of human activity recognition (HAR), starting from acquiring data to classifying the executed action. This thesis will confine itself to part four 'Classification' and will explore methods to recognise which action is performed using 6DOF object poses as features instead of e.g., skeletal data. The latter is where Ms. Wu's research comes into play. The data will be captured by an RGB-D camera, the Realsense L515.

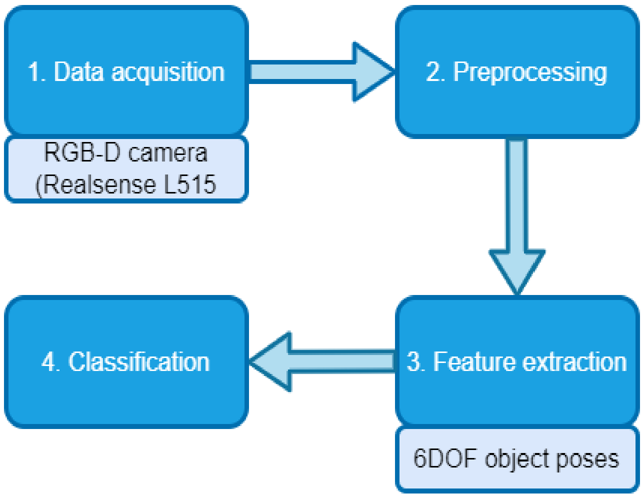


Figure 4: Schematic overview of the intended process for human activity recognition

The first step is to have a good notion of the two doctoral studies involved. This is self-evident since this master's project is very dependent on these studies. Next, a literature review is conducted. This literature review is used to gather as much information as possible on the topic of 'object affordances' and '6D pose estimation'. This study will mainly focus on leveraging the concept of object affordances for activity recognition in human-robot collaborative assembly. The literature study makes it possible to do a listing, study, and comparison between different methods to recognise the operator's action performed from object tracking. From this, the most appropriate method i.e., the method that meets the previously established requirements, is then chosen. Next, the selected method will be applied to infer the activity performed by an operator during the assembly of a product. The chosen product is the Bourjault ballpoint pen, visualised in Figure 5, and will be represented by ArUco markers.

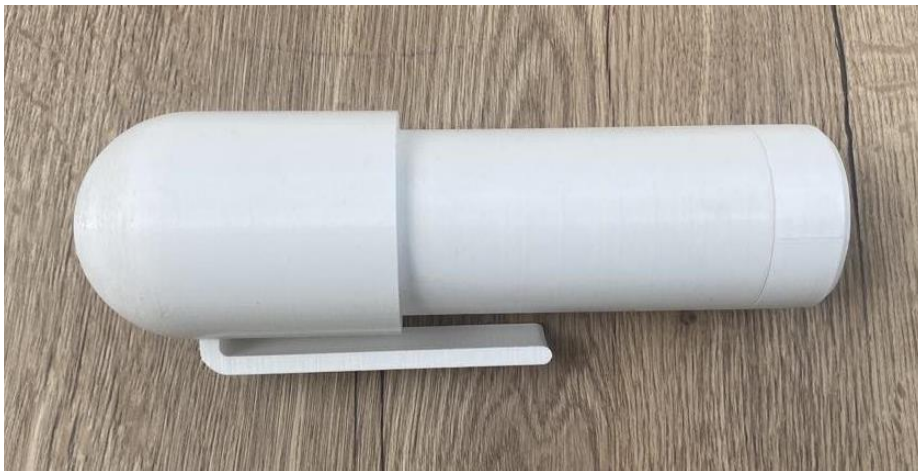


Figure 5: Upscaled model of an assembled Bourjault ballpoint pen

An ArUco marker is a specific type of marker used in computer vision and augmented reality applications. It is designed as a synthetic square-shaped marker that consists of two main components: a wide black border and an inner binary matrix. The black border helps with fast detection, while the binary matrix determines its unique identifier. This combination allows for quick and accurate marker detection and identification [9]. Figure 6 provides an example of an ArUco marker.

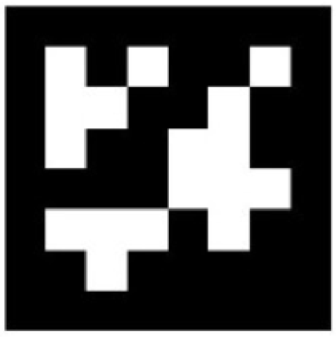


Figure 6: Example of an ArUco marker [9]

For the implementation, a software program will be written. The chosen coding language will be Python and it will be developed in the integrated development environment (IDE) of PyCharm. Finally, a test assembly is performed.

1.5 Structure of the thesis

The structure of this paper is as follows. The next chapter conducts a literature study, encompassing various topics such as human activity recognition, object affordances, probabilistic graphical models, modelling tools, and object detection and tracking methods. In the third section, an implementation of a probabilistic logic language called ProbLog, covered in the literature study, is discussed. Subsequently, the transition is made from a probabilistic logic language to a probabilistic graphical model known as dynamic Bayesian networks, which is also addressed in the literature study. The fourth chapter delves into the applied methods, specifically focusing on activity recognition in an assembly case by first addressing a subobjective of collision detection between two moving objects. Different dynamic Bayesian models are developed in this chapter, tailored to their respective applications. Chapter 5 encompasses a series of experiments conducted based on the models created in the previous chapter. The subsequent chapter evaluates the results obtained from these experiments. To validate the performance of the final model, a seven-fold cross-validation is employed to assess accuracy, precision, recall, and F1-score. Chapter 7 introduces an initial attempt at using six-dimensional pose as input data for activity recognition, utilising PPF matching for extracting the pose. Finally, chapter 8 concludes the thesis and provides an outlook on future work that can be pursued.

2 Literature study

To properly embark on research, it is necessary to conduct a literature study. This involves acquiring knowledge about the research topic and reviewing what research within this topic has already been conducted. This literature study is structured as follows. In the first section, it provides an explanation of what human activity recognition (HAR) entails and subsequently clarifies the different steps of HAR. Then, in the next section, the concept of object affordances is discussed. In section 3, different probabilistic graphical models are explained. Section 4 deals with possible modelling tools that can be used to implement the previously discussed probabilistic graphical models. Finally, section 5 offers an overview of possible methods that can be used for object detection and tracking.

2.1 Human activity recognition

The term central to this research is 'human activity recognition'. This kind of recognition is necessary to achieve a fluent cooperation between humans and robots e.g., when the robot is able to recognise the assembly activity that the operator is performing, it can aid by preparing the next task that is most likely going to be performed. HAR usually follows four steps: data acquisition, pre-processing, feature extraction and classification. Figure 7 pictures a schematic overview of the process involved in HAR.

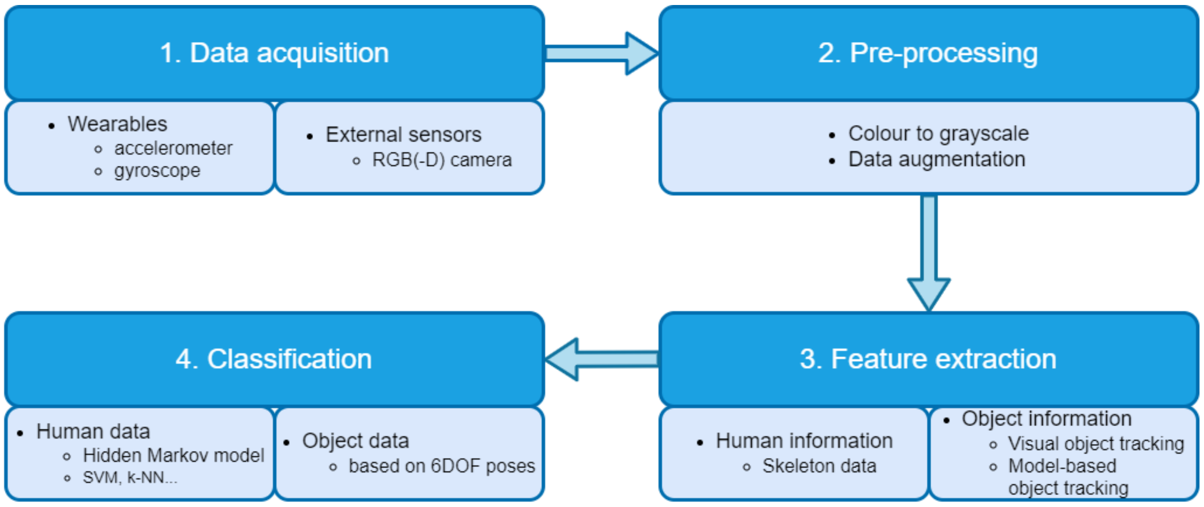


Figure 7: Overview of the human activity recognition process

2.1.1 Data acquisition

In the first step, the data necessary for HAR are acquired. Data acquisition involves recording data (or signals) using sensors. These data can then be used and analysed within a process. One distinguishes between two different types of sensors used for data acquisition within human activity recognition: wearable and external sensors.

The first type of sensors used for data acquisition are the wearable sensors, also known as wearables. Examples of wearables include accelerometers, gyroscopes, magnetometers, etc [10]. Another state-of-the-art wearable sensor used for data acquisition for HAR are head-mounted glasses that track the operator eye gaze to estimate his/her intension and the activity that is being performed [11]. Wearable sensors are able to monitor various human body signals, which makes it a fitting data acquisition method to recognise operator activities. The assembly of a product is an example in which wearable sensors can be used. By mounting a sensor on the operator's hand, it is possible to track the movement of the hand and therefore identify the assembly step that the operator is executing. For example, in an assembly, there is a significant difference in hand motion between tightening a bolt and performing a hammering operation. This is where the value of wearables that register the motion of the hands and fingers becomes evident. Figure 8 pictures the use of a wearable device that tracks the movements of a hand.

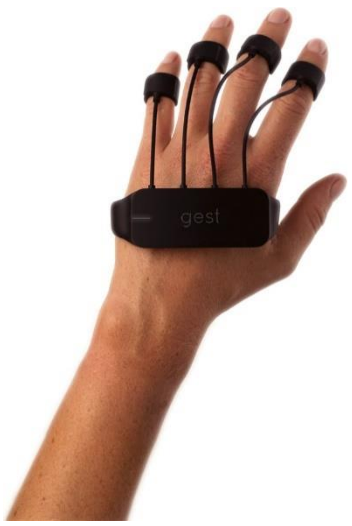


Figure 8: Wearable sensor for tracking hand activity [12]

One of the drawbacks of wearable sensors is that they can only capture data from the actions performed by the body parts to which the sensors are attached. To also capture the context in which the actions are performed, external sensors can be used. Continuing the example described earlier, by using cameras, not only the movements of the operator can be captured, but also the effects of these movements on the surroundings, such as the relative displacements of the assembly parts in relation to each other. Examples of these kind of sensors are a RGB [13] or RGB-D camera [8], [14], [15] (which captures depth information as well). Figure 9 provides an overview of different sensors available for human activity recognition.

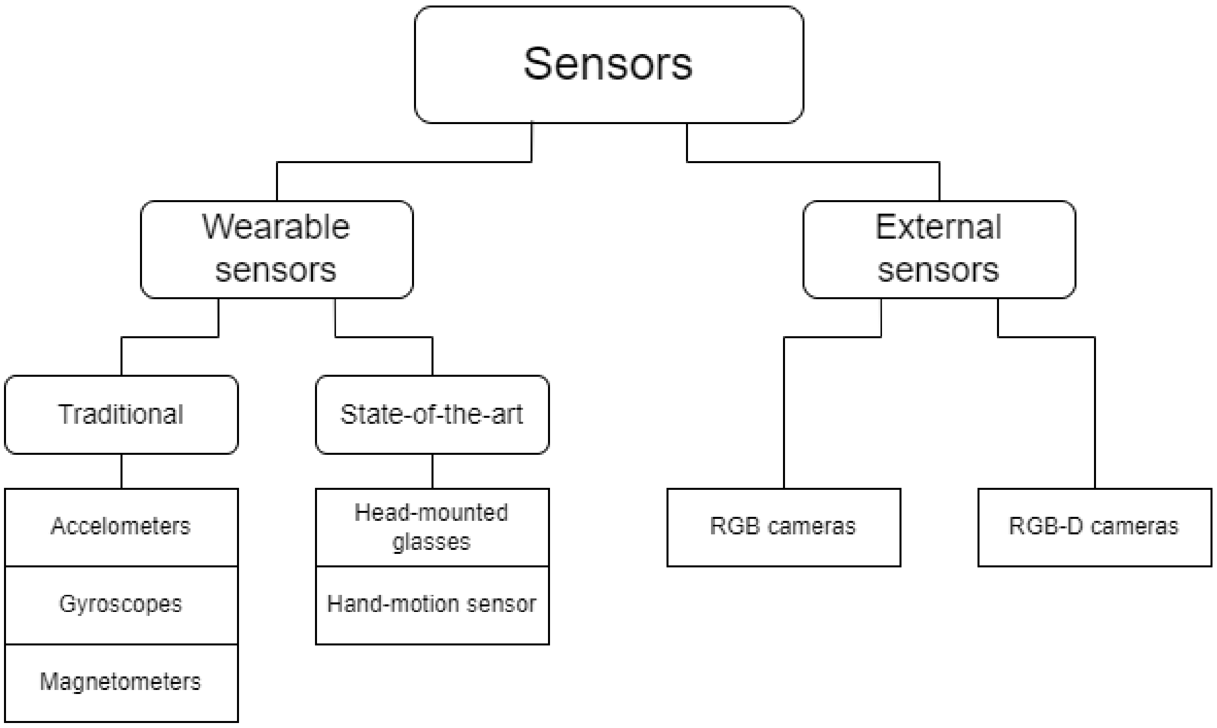


Figure 9: Overview of the different types of sensors that can be used for human activity recognition

2.1.2 Pre-processing

The data obtained from the sensors is usually noisy and not sufficiently clean to be used in the subsequent steps. To clean up this data, pre-processing is performed. What this step exactly involves, depends on the input data and the HAR algorithm. For example, three-channel RGB colour images may need to be converted to single-channel greyscale images. This way, the amount of storage the images take up can be reduced and the colours will not add extra unnecessary complexity. Data augmentation is also a common technique used in pre-processing. When using a dataset consisting of a number of images, this technique will start duplicating these images and slightly modify each copy (e.g., by enlarging/shrinking or rotating). By enlarging the dataset with images with slight deviations from the original images, the subsequent recognition algorithm is less biased towards the training dataset and more robust against variations in the input data during deployment [16]. Figure 10 gives an example of data augmentation applied on two images of a butterfly. Once the original data is pre-processed, the noisy-free input can be used in the next step.

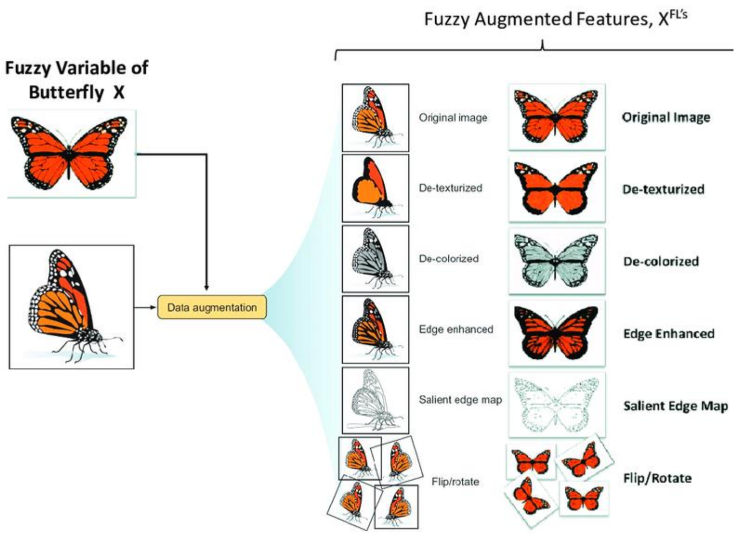


Figure 10: Example of data augmentation [16]

2.1.3 Feature extraction

Once the data are utilisable, feature extraction will be performed. In this stage, significant features that are unique for a specific activity are extracted. Feature extraction can be done using multiple methods. With relevance to this research, the different methods will be divided into two categories: methods based on human information and those based on object information.

Multiple methods have been developed for feature extraction based on human information. One of these methods is based on using skeleton data. Skeleton data are obtained by observing the position and movement of different points on the operator's hand during assembly such as for example in [17]. Depending on the HAR algorithm, the torso, arms and legs can also be tracked and used as skeleton data.

Feature extraction can also be done based on object information. Instead of using information from the operator performing the task, these methods use the info from the objects that are subject to the task. One of these methods is based on 2D appearance models, which uses visual object tracking [18]. A 2D appearance model is a representation of objects in a two-dimensional space based on their visual appearance. It can be utilised to track and monitor the movements or changes in the appearance of objects in a video sequence, and does not rely on the operator performing the task. Figure 11 provides a representation of how a 2D appearance model is applied to track a cartoon image through subsequent frames [19].

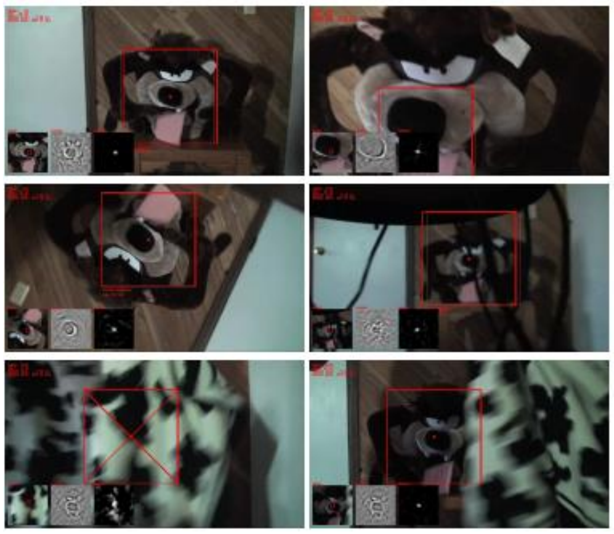


Figure 11: Tracking of a cartoon figure through subsequent frames based on its 2D appearance model [19]

Following the 2D appearance models, 3D models can also be used for feature extraction [20]. From these models, notable features get extracted, such as geometric features, surface properties, and point cloud features, in order to recognise these features in an RGB-D image and classify the objects that are represented by the models. Feature extraction can also be done by estimating 6DOF object poses. By detecting the 3D translation and rotation of an object, it is possible to use object information as features [21], [22].

2.1.4 Classification

The final phase in human activity recognition is classification. This involves classifying the activity based on the features extracted in the previous step. Multiple techniques are developed for this. These techniques can be categorised depending on whether they are model-based or learning-based, respectively called model-based and learning-based techniques.

Model-based techniques used for classification typically involve constructing a mathematical or probabilistic model that represents the relationship between input features and output labels [23]. A hidden Markov model (HMM) is a technique that can be utilised for model-based classification.

A hidden Markov model models a system that contains an unobservable process (the hidden layer). The states within this layer cannot be determined directly but have an influence on another process with variables that are observable [24]. In the concept of HAR, the hidden layer could represent different activities that need to be recognised. On the other hand, the visible layer represents the features or sensor data that are directly observable e.g., the movements of the hands registered by the wearable sensor depicted in Figure 8. Figure 12 shows the dynamic Bayesian network representation of this type of model. The hidden layer contains successive states q. These states succeed each other according to a transition matrix A. The state q_i is hidden but has an effect on the variable O_i in the observable layer according to the emission matrix B. By applying the Forward algorithm onto this HMM, it is possible to predict the probability of hidden state q being true given the sequence of previous states O in the observable layer as historical evidence [25].

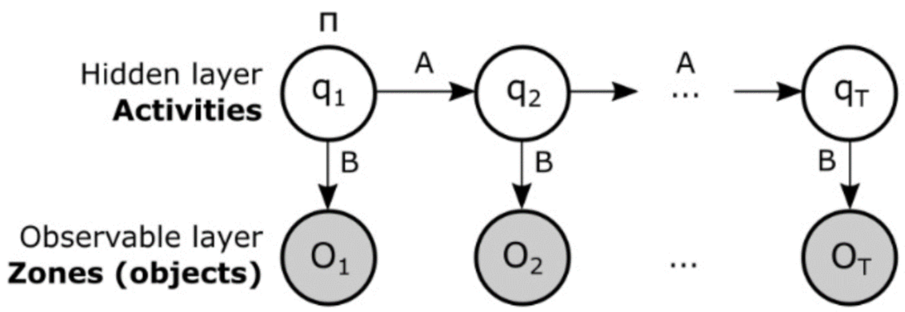


Figure 12: The dynamic Bayesian network representation of a HMM [25]

Another model-based technique for classification is the Support Vector Machine (SVM). The concept of SVMs was introduced by Vladimir Vapnik and Corinna Cortes in the 1990s. The basic idea behind SVMs is to find an optimal hyperplane that separates data points belonging to different classes in a high-dimensional feature space. The hyperplane serves as the decision boundary, maximising the separation between classes while maintaining the greatest possible distance between data points and the hyperplane. This distance is commonly known as the "maximum margin" [26]. In a simple example of HAR, where there are only two possible activities and two features, the training data could be labelled and plotted in a two-dimensional space. The SVM will then create a line in the two-dimensional space that maximally separates the data corresponding to the two activities. Each new activity that is registered will then be labelled as either one of the possible activities depending on its relative placement with respect to this line. The hyperplane in this example is represented by a line because only two variables/features are implemented. A SVM's ability to be applied in a higher dimensional space, makes it a fitting method to handle a broad range of features [27].

An example of a learning-based technique for classification is k-Nearest Neighbour (k-NN). In k-NN, k is a variable that states the number of neighbours a point can have. Such a point can represent a variety of data elements, e.g., in the context of this paper, a registered human activity in feature space. For instance, if k is equal to five, the five closest points are considered neighbours to the new data point. The classification of this new data point depends on the value or label of its neighbouring points. [28]

In [29], multiple machine learning techniques, such as k-NN, SVM and Random Forest are compared on their ability to classify human activities that are registered by accelerometers and velocity sensors attached to the subject's at the waist. The research concludes that the SVM method has the best precision and recall performance for that application, however, the Random Forest method has the best accuracy value and F1-score. [30] uses recurrent neural networks to classify assembly activities from different hand gestures recorded by a RGB camera during assembly. When only human data from the operator performing an activity is used for classification, important information that the external objects involved in the activity might provide, is neglected. In order to be able to use this kind of information, classification based on object data sounds promising. [22] is an example of using object data for classification of activities. In this research, a 3D object tracking algorithm for RGB-D data is employed. This algorithm tracks the 6DOF poses of the different objects in order to recognise which activity is performed. The final goal of that research was not to assess the performance of activity recognition but to compare a child's capability to perform an assembly alone and in collaboration with a robot.

Object data is not only useful in human-robot collaboration, but can also be adopted for operator guidance during assembly. Arkite's Human Interface Mate is a product that is already available on the market which uses object data to guide its user during assembly. Through a depth camera, the HIM checks the presence of all components and tools and whether or not they are in the correct place. In addition, the HIM also projects the assembly instructions on the workbench and checks if all the assembly steps are correctly performed [31]. Figure 13 provides an example of the usage of the HIM for assembly guidance. The HIM projects an image of how the pink cells should be placed in the container, what the next step is that the operator should perform, and where all the parts are placed [31].

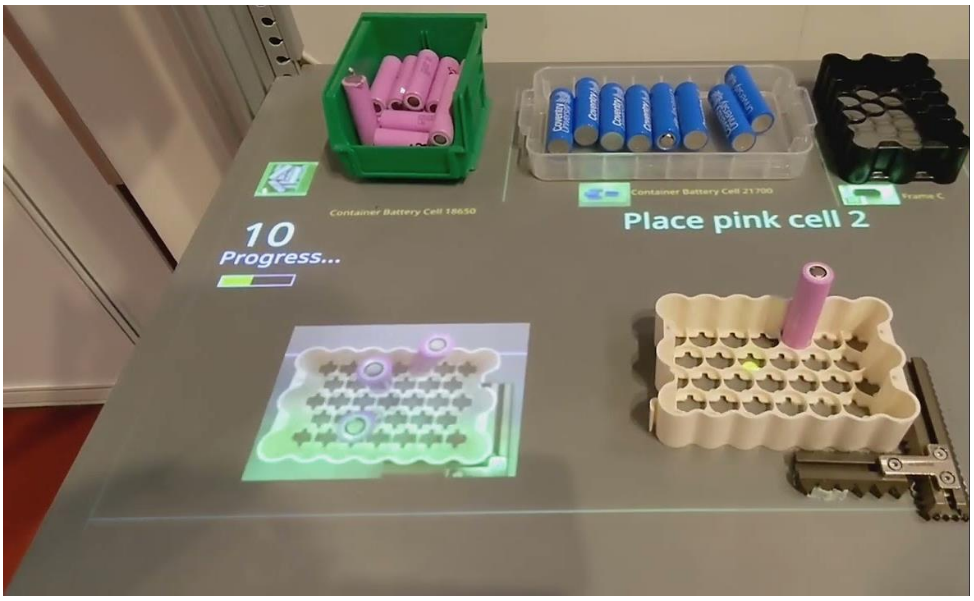


Figure 13: Assembly guidance by the HIM [31]

Human and object data can also be combined for classification. [32] uses both action recognition based on skeleton data and object tracking in order to learn to recognise human activities.

2.2 Object affordances

The concept of affordances was first introduced by the psychologist J.J. Gibson. It describes how “inherent ‘values’ and ‘meanings’ of things in the environment can be directly perceived and how this information can be linked to the action possibilities offered to the organism by the environment” [33, p. 1].

In robotics, the concept of affordances is used to represent the possible actions that a robot can perform on objects in the environment. It does so by identifying the different objects in the environment and stating the interdependencies between the properties of the object, the executed actions on these objects, and their respective effects. Consequently, affordances are neither properties of the objects nor the environment, but of the relations between the objects themselves or between an object and the environment. This is illustrated by the following example: *Left_of(nut, bolt)*. This example states that the nut is positioned to the left of the bolt—if the formula is defined so that the first element is in relation to the second—. The relation *Left_of* is inherent to neither the nut nor the bolt, but is a property of the relation between the two [34].

2.2.1 Propositional affordances

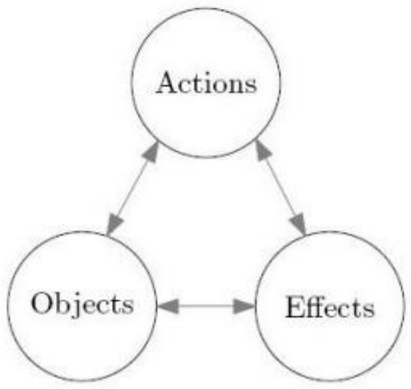
[7] models affordances as relations between three variables in a robotic environment:

$O = \{o_1, o_2, \dots, o_l\}$: the set of *objects* (and their properties) detected by the robot

$A = \{a_1, a_2, \dots, a_m\}$: the set of *actions* that are available to the robot

$E = \{e_1, e_2, \dots, e_n\}$: the set of *effects* after performing the actions on the objects

The affordance model allows to predict one of these variables when the other two are given. Figure 14 gives a schematic overview of the generic affordance model and the three different functions for which it can be used.



Inputs	Outputs	Function
(O, A)	E	Effect prediction
(O, E)	A	Action recognition/planning
(A, E)	O	Object recognition/selection

Figure 14: Example of an affordance model [7]

Consider the following example for action recognition: a robot observes a ball and a box (objects) lying separately on a table and being handled by a person. Next, the ball lies in the box (effect). From this, the robot can predict that the person has put the ball in the box (action).

If the affordance model is not provided, the robot usually performs an exploration phase, called motor babbling, in order to obtain such a model. During this phase the robot performs its set of actions on the different objects in the environment and perceives the effects of its actions. Next, the robot undergoes a learning phase in which it learns a model by using the collected samples during the exploration phase. Once this phase is completed, the robot developed an affordance model [7]. This affordance model uses a propositional representation for object properties, actions, and effects. However, this particular model faces challenges when applied to robots because of the uncertainty and complexity of the physical aspects involved. Robots interact with the real world through sensors, which may introduce noise, and actuators, which may be imprecise or noisy. These physical aspects can make it difficult to accurately interpret sensory data, process images from cameras, and control the robot's actuators effectively.

To address this problem, the concept of *relational affordances* is introduced. Relational affordances take into account the relationships and interactions between objects in the environment. Rather than relying solely on logical propositions, relational affordances consider the context, spatial relations, and dynamics of the environment to determine the potential actions and effects.

2.2.2 Relational affordances

Relational affordances represent an expansion of the previously mentioned affordance model by utilising a relational representation instead of a propositional one. In this case, a relational affordance refers to a probability distribution that captures the relationships among the variables O, A, and E. This distribution, illustrated in Figure 14, P(O, A, E), represents the likelihood of different states of the objects, actions, and environments occurring together. Affordance models based on propositional affordances are not able to cope with multiple objects interacting with each other. This is where relational affordances form the solution. The concept of relational affordances is unique because it takes into account the (spatial) relations between different objects. By incorporating relational affordances, the model can better handle the uncertainties and complexities of physical interactions. It allows the robot to understand how objects relate to each other and how their interactions can influence the robot's actions and the overall environment. This approach provides a more robust and adaptable framework for robots to perceive and interact with their surroundings [7].

2.3 Probabilistic graphical models

Probabilistic graphical models (PGMs) are a class of statistical models that use graphical structures to represent probabilistic relationships between random variables. Such a model contains two types of graphical structures: nodes and edges. A node is a representation of a random variable, while an edge represents the dependencies or conditional independencies between these variables. PGMs can be categorised into two main types: directed graphical models (DGMs) and undirected graphical models (UGMs), respectively known as Bayesian networks (BNs) and Markov random fields [35].

A PGM allows making predictions about the probability distribution of one or more variables given the values of other variables. Inference in PGMs involves calculating the marginal or conditional probabilities of the variables. These calculations can be performed using various algorithms, such as belief propagation, variational inference, or Markov chain Monte Carlo methods [35].

Learning in PGMs involves estimating the model parameters based on observed data. Depending on whether a DGM or UGM is used, this is done differently. Since a variable in a DGM solely depends on the value of its parent variables, learning typically involves estimating the conditional probabilities of each variable given its parents in the graph. In UGMs, it is necessary to estimate the potential functions that capture the compatibility between the states of neighbouring variables [35].

The most relevant PGMs to this research are three directed graphical models: standard and dynamic Bayesian networks, and hidden Markov models. These will be discussed in the following sections.

2.3.1 Bayesian networks

A Bayesian network, also known as a belief network or Bayes net, is a probabilistic graphical model that represents knowledge about an uncertain domain. As described earlier, the network is composed of nodes and edges, with each node representing a random variable and each edge representing the conditional probability for the corresponding random variables [36]. By utilising probabilities to represent uncertainties, BNs provide a framework for probabilistic reasoning, enabling decision makers to explicitly model and reason about uncertainty. This attribute makes BNs highly beneficial for decision making. Figure 15 illustrates an example of the structure of a Bayesian network modelling the chance of an alarm going off when a burglary or an earthquake occurs and John or Mary calling the authorities in response.

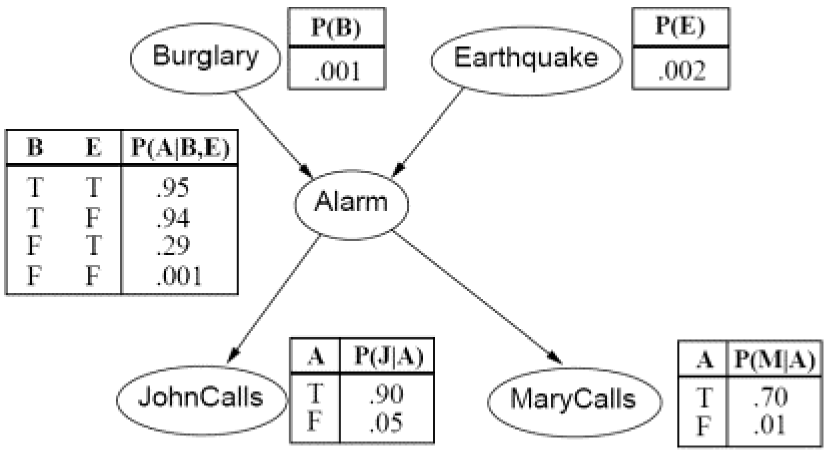


Figure 15: Bayesian network model for the burglary-earthquake example [37]

Each node (Burglary, Earthquake, Alarm, JohnCalls and MaryCalls) in the Bayesian network corresponds to a specific random variable. The state of each variable is represented by a probability distribution e.g., P(Burglary) = 0.001, which states that there is a marginal probability of 0.1 percent that a burglary will occur. The edges in the graph indicate the conditional dependencies between the variables. These edges are directed from one node to another, indicating the parent-child relationship between the two nodes. The child node, the node that is pointed towards, represents a random variable whose value depends on the values of its parent nodes e.g., P(Alarm|Burglary, Earthquake). A node can be a child to multiple parent nodes and parent to multiple child nodes. For example, in the figure above, 'Alarm' is a parent node to both 'JohnCalls' and 'MaryCalls'. The probability distribution of a node for that variable given the values of its parent nodes, are represented by conditional probability distribution tables (CPDs). The probabilities that the CPDs are composed of can be determined through either expert knowledge or data collection. In the former case, an expert provides their expertise and knowledge to estimate the probabilities. In the latter case, statistical techniques such as maximum likelihood estimation or Bayesian estimation are applied to compute the probabilities based on the input data provided.

By combining the CPDs of all nodes in the network, it is possible to calculate the joint probability distribution of all variables in the network. For example, to calculate the probability of the joint event Q: “Burglary=True”, “Earthquake=False”, “Alarm=True”, “JohnCalls=True”, and “MaryCalls=False”, the following probabilities are multiplied: P(Burglary=True), P(Earthquake=False), P(Alarm=True | Burglary=True, Earthquake=False), P(JohnCalls=True | Alarm=True), P(MaryCalls=False | Alarm=True). This gives the following result:

$$Q = 0.001 * 0.998 * 0.94 * 0.9 * 0.3 = 0.000253$$

This indicates that there is a 0.0253 per cent chance that there is a burglary but no earthquake, the alarm goes off, and that John calls the authorities but Mary does not.

2.3.2 Dynamic Bayesian networks

Dynamic Bayesian networks (DBNs) are an extension of Bayesian networks that are also capable of modelling systems that change over time. They are commonly used in engineering to model complex systems that involve time-dependent relationships between variables. [38] uses a dynamic Bayesian network to select landmarks for mobile robot navigation. Similar to a BN, in a DBN, the variables of the system are represented by nodes and the relationships between the nodes are represented by edges. However, unlike a static Bayesian network, the edges in a dynamic Bayesian network can represent both causal relationships and temporal dependencies between variables [39]. Figure 16 provides a general example of a dynamic Bayesian network. The causal relationships are represented by the intra-slice arcs and the temporal dependencies are presented by the inter-slice arcs.

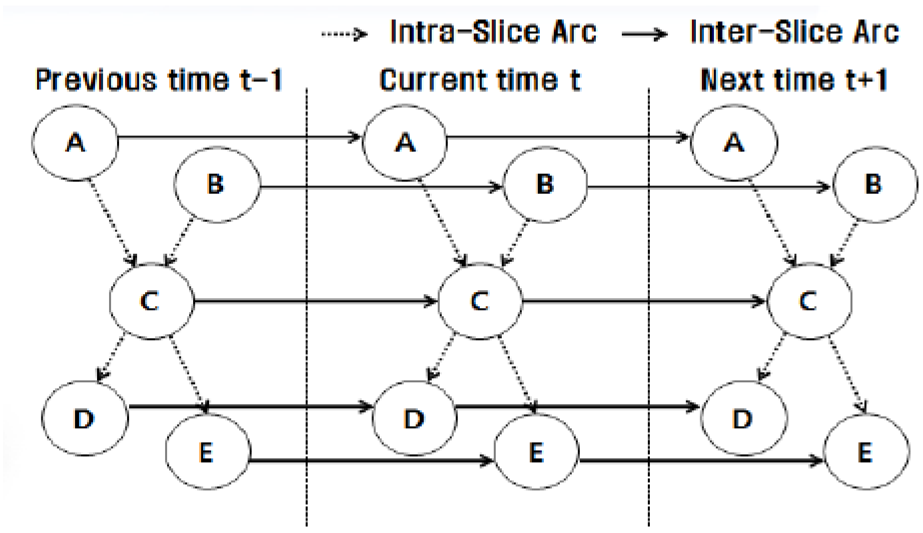


Figure 16: General example of a dynamic Bayesian network [40]

To learn a DBN from data, it is necessary to estimate the conditional probability distributions that describe the relationships between the variables at each time-step. This can be done by using various techniques, such as the Expectation-Maximization algorithm, which alternates between estimating the hidden variables and updating the parameters of the model, or particle filtering, which is a sequential Monte Carlo method that estimates the posterior distribution over the hidden variables. Once the DBN is learned, it can be used to make predictions about future states of the system, or to infer hidden states based on observed data, or to perform other tasks such as decision making.

Several extensions and variants of DBNs exist, such as dynamic Markov networks, time-varying Bayesian networks and continuous-time Bayesian networks. These models allow for more flexible and expressive representations of time-dependent processes [41].

2.3.3 Hidden Markov models

Hidden Markov models can be thought of as a special case of dynamic Bayesian networks. The structure of a HMM consists of a chain of hidden states. Each of the hidden states is connected to the next state in the chain through a directed edge. The visible layer, which contains the observed data, is then connected to each state through another set of directed edges. Figure 17 illustrates a general example of a hidden Markov model. In the visible layer, the possible observable states are presented as o, while the states in the hidden layer are presented as s. The probability that a state s_1 transitions into state s_2 is defined as the transition probability p_{12}, while the probability that o_1 is observed when s_1 occurs, is defined as φ_1.

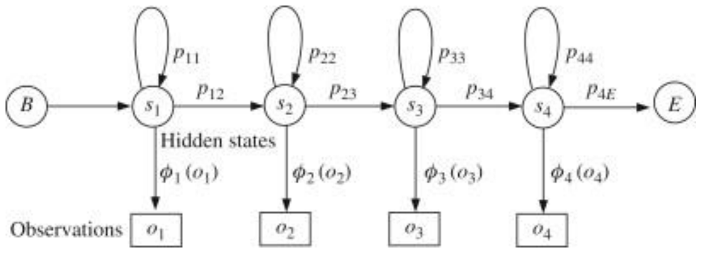


Figure 17: General example of a hidden Markov model [42]

In a HMM, the process of making inferences involves the computation of the posterior distribution of the hidden states given the observed data. For Figure 17, this computation can be denoted as:

$$P(s|o,p,\varphi) \tag{1}$$

This task can be accomplished by using the forward-backward algorithm. This algorithm is a form of dynamic programming that calculates the marginal probabilities of each hidden state at each time step. To obtain a model that accurately represents the underlying probabilistic structure of the data, the HMM undergoes a learning process. This process involves estimating the model's parameters from a given set of observed data. A way to do this is by utilising the Expectation-Maximization algorithm that was mentioned earlier in section 2.3.2 [43].

In order to determine the most likely sequence of hidden states that generated a given sequence of observations, a decoding process is performed. This is typically done through the Viterbi algorithm. The principle of the Viterbi algorithm is that it explores the space of possible state sequences and selects the one with the highest probability [42].

Given a particular HMM, the probability of observing a given sequence of observations can be calculated by applying the likelihood computation, also known as the forward algorithm. It computes the forward probabilities, which represent the probability of being in a specific hidden state at a specific time, taking into account the previous observations and transitions [42].

HMMs have a wide range of applications. They are particularly useful in situations where the underlying process is not directly observable, but can be inferred from the observed data. Various extensions of the basic hidden Markov model exist, such as the auto-regressive hidden Markov model [44] and the factorial hidden Markov model [45]. These variations allow for more complex and flexible modelling of sequential data.

2.4 Modelling tools

Probabilistic graphical models are powerful tools for representing and reasoning about uncertain knowledge in various fields. However, the actual implementation of these models requires specialised software that is capable of efficiently developing and implementing these probabilistic models. In this section, an overview of different types of modelling tools is provided.

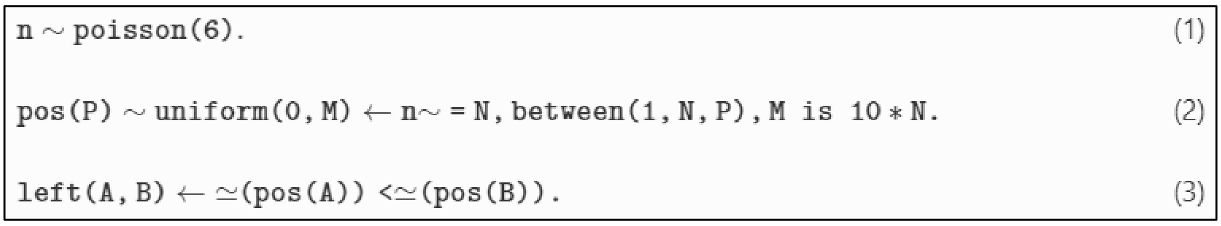
Probabilistic logic languages provide a rich and flexible way to specify and reason about probabilistic relationships using logical rules. They allow to represent complex dependencies between variables, and to perform efficient inference over large-scale models [46]. Python libraries, on the other hand, provide a powerful and flexible environment for building and analysing probabilistic models. In this section, some of the popular PLLs will be discussed such as Distributional Clauses and ProbLog. In addition, some of the popular Python libraries used for probabilistic modelling will be discussed including PyMC3, TensorFlow Probability (TFP) and pgmpy.

2.4.1 Probabilistic logic languages

Probabilistic logic languages (PLLs) are programming languages that unify probabilistic modelling and traditional general-purpose programming. They are used to create programs that can deal with uncertainty making it an important concept in this research. PLLs have the capability to integrate logic programming and probability theory, and they can be categorised into two groups: those that utilise distributional semantics and those that employ knowledge base model construction. The main difference is in their ability to represent uncertainty. In distributional semantics, uncertainty is typically represented using probability distributions over the possible values of variables. On the other hand, in knowledge base model construction, uncertainty is represented through the construction of multiple possible models of the knowledge base [47]. Distributional Clauses and ProbLog are both PLLs that follow distributional semantics.

Distributional Clauses

Most probabilistic models used for problems such as state estimation in robotics cannot easily represent relational information. This information includes the objects, properties and the relations that hold between them. Distributional Clauses is a probabilistic logic language that takes this information into account and can deal with hybrid relational domains. In the context of distributional clauses, a formula takes the form $h{\sim}D \leftarrow b_1, \dots, b_n$ where b_1 represents literals and $\sim$ is a binary predicate expressed in infix notation [48]. The term h refers to the head, while $b_1, \dots, b_n$ corresponds to the body [49]. Figure 18 represents an example of distributional clauses.



```
n ~ poisson(6).                                                    (1)

pos(P) ~ uniform(0,M) ← n~ = N, between(1,N,P), M is 10 * N.      (2)

left(A,B) ← ≃(pos(A)) <≃(pos(B)).                                 (3)
```

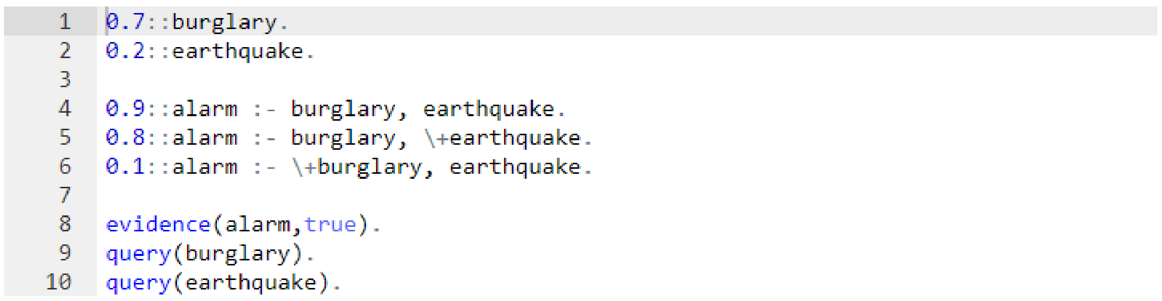
Figure 18: Example of a Distributional Clauses program [48]

In clause (1), variable n is defined as having a Poisson distribution with a mean of 6. Clause (2) specifies that the position of P is a continuous random variable uniformly distributed between 0 and $10 * n$, where P represents each person identifier ranging from 1 to n. For instance, if n takes the value of 2, there will be two distinct random variables $pos(1)$ and $pos(2)$, each uniformly distributed between 0 and 20. Finally, clause (3) models the binary relation 'left' between A and B, namely person A should be positioned left of person B [48].

ProbLog

ProbLog is a probabilistic extension of ProLog. ProLog, short for "Programmation en Logique," is a logical programming language where program statements are used to represent facts and rules related to various problems within a formal logic system. A ProbLog program typically consists of a set of rules in the form of clauses: deterministic rules and a set of probabilistic facts that each state the probability that it belongs to a sampled logic program. That way, the ProbLog program defines a distribution over the different logic programs. Finally, a ProbLog program includes one or multiple queries. The probability that such a query is successful in a randomly sampled program defines the semantics of ProbLog [50].

Consider the following example in order to get a better understanding of how a ProbLog program is structured. The example consists of a simplified version of the burglary-earthquake scenario. The program is visualised in Figure 19.



```
 1  0.7::burglary.
 2  0.2::earthquake.
 3
 4  0.9::alarm :- burglary, earthquake.
 5  0.8::alarm :- burglary, \+earthquake.
 6  0.1::alarm :- \+burglary, earthquake.
 7
 8  evidence(alarm,true).
 9  query(burglary).
10  query(earthquake).
```

Figure 19: Example of a ProbLog program for the simplified burglary-earthquake scenario [51]

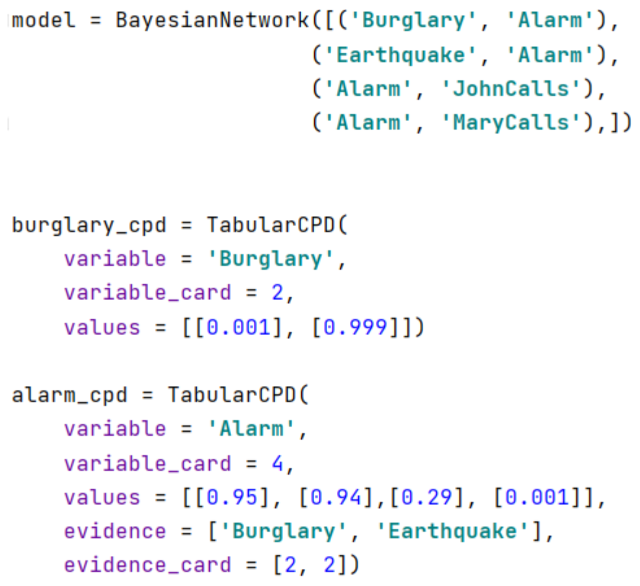
The first row states a probabilistic fact: there is 70 per cent marginal probability that a burglary occurs. An example of a rule being defined in the form of a clause can be seen in line 4. This rule defines that there is a probability of 90 per cent that the alarm goes off if a burglary and an earthquake occur together. Once these sets of facts and rules are defined, it is possible to use the program in order to solve queries. Line 9 gives an example of such a query that asks for the probability that a burglary occurs. The program can also be extended by adding evidences. In this example, line 8 states that the alarm goes off. This piece of evidence will influence the query in line 9.

2.4.2 Python libraries

PyMC3 is an open-source probabilistic programming framework designed for flexible specification of Bayesian statistical models in Python. It has a user-friendly and powerful syntax that makes it easy to describe complex models, and it uses advanced algorithms to efficiently sample from these models. In PyCM3, it is possible to use probabilistic programming to define, for instance, a Bayesian network by specifying the prior distributions and likelihood functions for each node in the network. Then, advanced Markov chain Monte Carlo (MCMC) sampling algorithms like the No-U-Turn Sampler (NUTS) will be used to generate samples from the posterior distribution of the network, allowing to estimate the probability distribution of the nodes given the observed data [52].

TensorFlow Probability is a Python library build on TensorFlow. It provides a range of tools for constructing and fitting probabilistic models including variational inference and MCMC algorithms and optimisers such as Nelder-Mead, BFGS and SGLD. In order to use TPF as a modelling tool, first, it is necessary to specify the probabilistic model in terms of its likelihood function and prior distributions. Next, TFP's inference tools are used to fit the model to the data. This can be done by using the tools mentioned earlier. Furthermore, TFP provides tools for model validation and selection such as Bayesian model averaging [53].

Another Python library for working with PGMs is pgmpy. In pgmpy, the model is first defined. This means that the structure of the PGM and the conditional probability distributions that define the relationships between the variables is specified. Figure 20 provides a snippet of the burglary-earthquake Bayesian network implemented in Python through pgmpy. First, the layout of the model is constructed by defining the relationships between the nodes. Next, the state probabilities for each node is defined.



```
model = BayesianNetwork([('Burglary', 'Alarm'),
                       ('Earthquake', 'Alarm'),
                       ('Alarm', 'JohnCalls'),
                       ('Alarm', 'MaryCalls'),])


burglary_cpd = TabularCPD(
    variable = 'Burglary',
    variable_card = 2,
    values = [[0.001], [0.999]])

alarm_cpd = TabularCPD(
    variable = 'Alarm',
    variable_card = 4,
    values = [[0.95], [0.94],[0.29], [0.001]],
    evidence = ['Burglary', 'Earthquake'],
    evidence_card = [2, 2])
```

Figure 20: Snippet of the model for the burglary-earthquake example constructed in Python through pgmpy

Once the model is created, pgmpy will perform inference on it. This involves computing marginal or conditional probabilities of variables in the model given the evidence. Finally, the performance of the model can be evaluated by comparing the predicted probabilities to the actual outcomes. By using pgmpy, it is easy to define and modify the structure of the models [54].

2.5 Object detection and tracking

Object tracking refers to the process of locating and tracking of a specific object in a sequence of video frames over time. It is a crucial component of many computer vision systems including those used for industrial automation and robotics. This section explores several methods for object tracking that can be used to accurately and reliably track the pose of objects in real-time.

Blob detection is an example of such a method that can be applied for object tracking. It relies on isolating an object based on its colour. By applying blob detection to detect an object in successive frames, the object can be tracked throughout the frames. This method is especially useful for tracking objects that have a distinctive colour. Objects can also be tracked based on a template of the object. By creating templates that represent the object and matching this template in video frames in which the objects occur, the position of the objects in that frame can be located. Another method for tracking objects is Point Pair Feature (PPF) matching. Instead of focusing on the colour of the object, this method identifies and tracks specific geometric features of the object by using 3D information from the model and scene. This makes this method more robust than colour filtering, as it can be used to track objects even when their colour or appearance changes. On the other hand, blob detection based on colour filtering will work better if the depth data needed for PPF matching is not available, the depth quality is not good, or if the geometry of the object is not distinctive. Finally, several commercially available software solutions for object tracking will be examined. These software packages and open source libraries can provide a comprehensive and reliable solution for object tracking in industrial automation and robotics applications.

2.5.1 Blob detection

Blob detection is a method widely used for object tracking because of its relatively easy implementation. [55] uses blob detection and colour filtering for video surveillance. Firstly, an image of the object of interest is captured and a colour histogram is created of the object that needs to be tracked. This colour histogram will then be used to identify pixels in subsequent frames that match the object's colour. This can be done by comparing the colour of each pixel in the image to the colour histogram and assigning a weight to each pixel based on how well it matches the histogram. The pixels with the highest weights are then isolated. Regions of the image that contain these isolated pixels are then defined as blobs. These blobs represent the objects that are detected. By repeating this process for every subsequent frame, the relevant object can be tracked throughout the complete video stream. Figure 21 illustrates the implementation of blob detection to detect the position of a red cup in a frame.

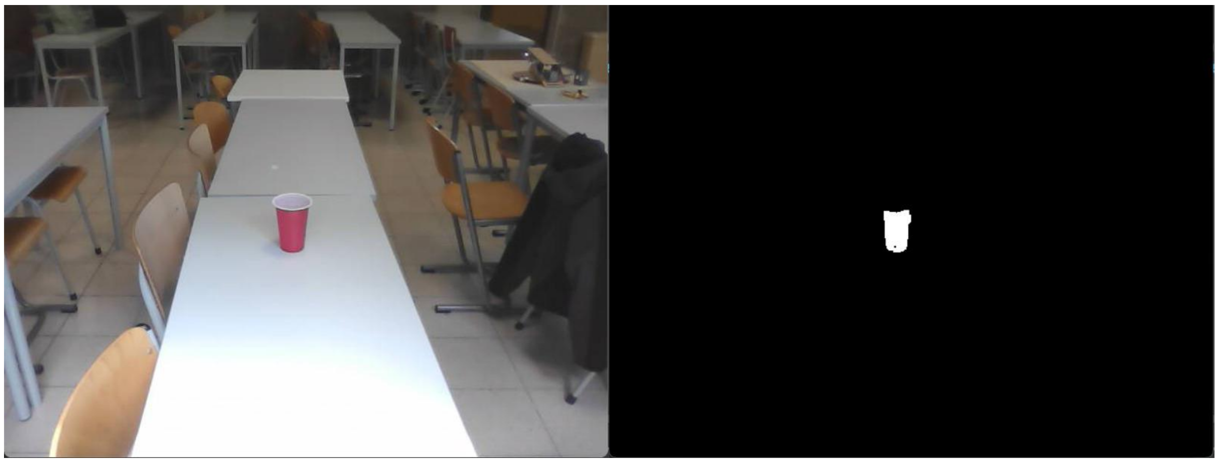


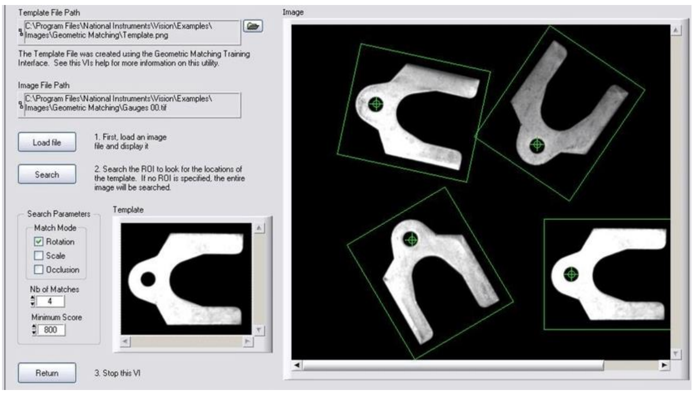
Figure 21: Image of a red cup (left) and the blob that represents the red cup after colour filtering (right)

Blob detection through colour filtering does not require complex algorithms or specialised equipment, such as an RGB-D camera. This makes it a fast process, which means it could be used for real-time object tracking and run on less powerful devices. Unfortunately, colour filtering also has its disadvantages. Since the method relies on isolating the object based on its colour, it is not an effective method if the object's colour is similar to the background. Besides, the colour of an object registered by the camera can vary based on factors, such as shadows, lighting and reflections. Additionally, it is not always possible to track and detect the pose of an object through blob detection. If the object has exceptionally distinct dimensions, one could consider constructing a bounding box around its blob for pose estimation.

2.5.2 Template matching

Template matching for object tracking is a technique used to find instances of a specific object or pattern within an image. It involves sliding a template, which is a smaller image representing the object of interest, across the entire larger image and comparing the template with each overlapping portion of the image. The goal is to identify areas in the image that closely resemble the template. To perform template matching, a two-dimensional convolution operation is used. Convolution is a mathematical operation that combines two matrices by multiplying corresponding elements and summing the results. In this case, one matrix represents the image, while the other matrix is the template, also known as a convolution kernel [56].

The process starts by placing the template on top of the image at a specific location. The overlapping region between the template and the image is multiplied element-wise, and the resulting values are summed. This sum represents the resemblance or similarity between the template and the corresponding portion of the image. The template is then shifted to a new position, typically by one or a few pixels, and the process is repeated. By sliding the template across the entire image, a similarity map is generated indicating how well the template matches each location in the image. To determine the location of the object being tracked, the highest similarity score or a predefined threshold can be used. The position with the highest score or the locations above the threshold are considered as potential matches [56]. An example of an industrial software provided by National Instruments that applies template matching to locate objects in an image is illustrated in Figure 22.



Template File Path: C:\Program Files\National Instruments\Vision\Examples\Images\Geometric Matching\Template.png

The Template File was created using the Geometric Matching Training Interface. See this VIs help for more information on this utility.

Image File Path: C:\Program Files\National Instruments\Vision\Examples\Images\Geometric Matching\Gauges 00.tif

1. First, load an image file and display it
2. Search the ROI to look for the locations of the template. If no ROI is specified, the entire image will be searched.
3. Stop this VI

Search Parameters
Match Mode: ☑ Rotation ☐ Scale ☐ Occlusion
Nb of Matches: 4
Minimum Score: 800

Figure 22: Example of an industrial software tool for template matching [57]

Template matching can be used for various tasks including object detection, recognition, and tracking. It is a simple yet an effective method for locating objects within images based on their visual similarity to a template. [57] uses template matching to recognise human activities based on model postures of the human. Template matching also has some drawbacks. It is a less robust method when the lighting alters significantly or the objects appearance changes (e.g., when the objects are partly occluded).

2.5.3 Point Pair Feature matching

Object detection and pose estimation are common challenges in computer vision. These tasks often involve locating a specific object within 2D or 3D scenes. Industrial objects are typically represented by CAD models or reconstructed in 3D. The goal is to detect instances of the object within scenes captured using one or multiple RGB-D cameras. However, in many scenarios of robotics and computer vision applications, this information is not sufficient, and additional information such as the six-dimensional pose is required. Point Pair Feature matching is a popular technique used in computer vision and robotics to obtain these data from a point cloud registered by a depth camera [58].

The idea behind PPF matching is to compute a set of unique features that capture the local geometry of an object in a scene. These features are based on the relative position and orientation of pairs of points on the surface of the object. The PPFs consist of three elements per point pair: the angles between the normal vectors at the two points, and the distance and angle between the projection of the second point onto the plane defined by the first point and its normal vector [58].

Once the PPFs are computed for a set of points on the object's surface, they can be used to match the 3D object to the scene. This process involves computing the PPF for each point in the scene and finding the closest match in the set of features computed for the object. Once a match is found, the relative pose between the object and the camera can be computed using the corresponding point pairs [58].

PPF matching is very suitable for environments with complex scenes because of its robustness to occlusion and clutter. This is illustrated in Figure 23. The objective is to detect the sculpture of a frog in a scene with multiple different sculptures. The green overlay shows the matched 3D model in the scene. Even when the frog is occluded, it can still be detected.

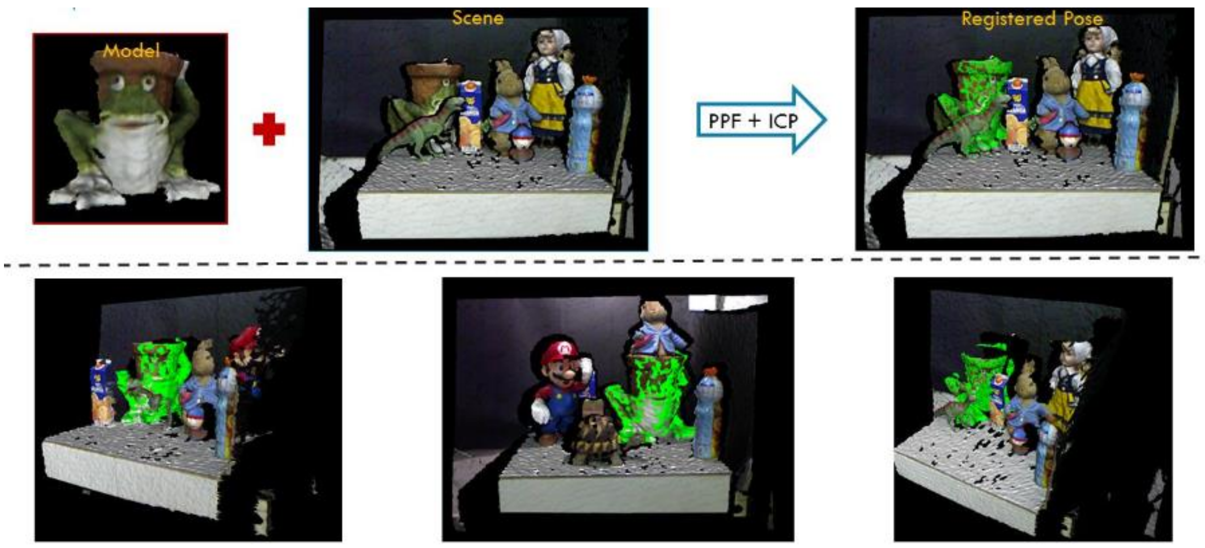


Figure 23: Implementation of PPF matching to detect the sculpture of a frog in a scene [59].

Finally, it is an accurate method and can handle large variations in object poses making it suitable for a wide range of applications in robotics and computer vision. A drawback of the method is that it is more complex and computationally expensive than the other previously mentioned, which results in significantly higher computation times.

2.5.4 Commercially-available software

Multiple commercially available software exist for object tracking. These software are often designed to be user-friendly and easy to use. Additionally, they typically come with technical support and documentation, which is very helpful getting information about the software or when running into problems while using it. Finally, these software are typically also thoroughly tested and validated by the developers.

An example of such software is Pickit 3D, a software and hardware package used for robot guidance and inspection using 3D vision technology. Pickit 3D makes it possible for a robot equipped with a 3D camera to quickly and accurately detect and locate objects. Its software provides a user-friendly interface for configuring and calibrating the system and for programming the robot to perform specific tasks. The software also includes advanced features, such as collision avoidance and path planning [60].

2.5.5 Open source libraries

Multiple computer vision libraries, such as Point Cloud Library (PCL) and OpenCV, exist as well that can be adopted for object tracking. These libraries provide a set of pre-defined algorithms and functions that simplify the implementation of computer vision applications.

Point Cloud Library is an open source software library designed for 3D point cloud processing. It provides a set of tools to work with these point clouds, such as algorithms for filtering, segmentation, feature extraction, and visualization. PCL also includes support for various sensors and data formats commonly used in robotics and computer vision applications. Additionally, it provides integration with other libraries, such as OpenCV or ROS, making it a popular choice for developing complex robotic and computer vision systems [61].

Another open source software library that can be adopted for object tracking is OpenCV. OpenCV has a wide range of tools and techniques for object tracking that can be used in a variety of applications. OpenCV is a widely used library in industry and academia, which means there are plenty of resources, tutorials and examples available. Additionally, OpenCV's broad range of tracking algorithms makes it suitable for almost every type of application. Finally, OpenCV can be easily integrated with other libraries such as PCL or TensorFlow, which gives the user the possibility to create an optimal system by combining the advantages of multiple computer vision libraries [62].

2.6 Conclusion

This literature study provides insights in which input data, models, concepts and languages are beneficial to this research. To make use of as much data as possible for classification of assembly activities, and to make sure no valuable information gets lost, classification based on object data is preferred over classification based on human data. Prior to classification, object information—preferably the objects' 6DOF poses as they contain a maximum amount of information—will be used for feature extraction. In order to retrieve the 6DOF poses, a RGB-D camera needs to be employed as it also provides depth data. The concept of object affordances, more specifically relational affordances, also plays an important role as it allows modelling the (spatial) relations between the different assembly objects. To eventually classify the operator activity, probabilistic graphical models could be applied. These models are able to effectively model the uncertainties and dependencies present in an assembly task. Bayesian networks are a popular type of probabilistic graphical models for activity classification in an assembly case because it represents the causal relationship between the variables. To implement these models, modelling tools such as probabilistic logic languages could come in handy. Multiple Python libraries exist for this purpose as well. Accurate and reliable object tracking is crucial for successful classification of assembly activities. Several methods are available for object tracking including colour filtering and Point Pair Feature matching. Additionally, there are multiple commercially available solutions for object tracking available.

3 Implementation probabilistic logic languages

In an assembly task, many factors could influence the assembly process, such as the sequence of steps in the assembly process, the behaviour of the operator during assembly, and the physical environment of the assembly area. These factors ensure that an assembly procedure will not always be performed in the same way, thus introducing a significant level of unpredictability. To deal with these uncertainties, Distributional Clauses and ProbLog seemed to be appropriate modelling tools because of their abilities to handle uncertainties and probabilistic relationships between variables.

The original idea was to implement Distributional Clauses as the probabilistic logic language for creating a model that is capable of classifying the performed assembly activity based on information provided by the assembly parts. Unfortunately, during its installation, it was discovered that Yap Prolog, which is necessary to run DC, was outdated and no longer maintained. ProbLog was selected as a promising alternative.

3.1 Methods

As a first step towards the goal of this thesis, a ProbLog script was created that could predict the probability of two moving objects colliding. This collision between objects is relatable to performing a task for the assembly of the Bourjault ballpoint pen, described in section 1.4, such as placing the ink cartridge inside the ballpoint pen's body in which a similar movement is performed by the operator.

3.2 Experiments

Utilising the ProbLog tool, a Python script to calculate the probability of collision between two objects was created.

During a recording of a random movement sequence of the two objects, represented by ArUco markers, the 2D coordinates of both objects were extracted for each camera frame. To calculate the probability of collision at time t, the 2D coordinates of both objects at time t and $t-1$ where used as input data. Based on these input data, three variables are calculated: the distance between the two objects, the angle difference between the direction of movement of object A and the orientation of object B relative to object A, and the angle difference between the direction of movement of object B and the orientation of object A relative to object B.

Figure 24 illustrates this angle difference α. β represents the direction of movement of object A and γ represents the orientation of object B relative to object A.

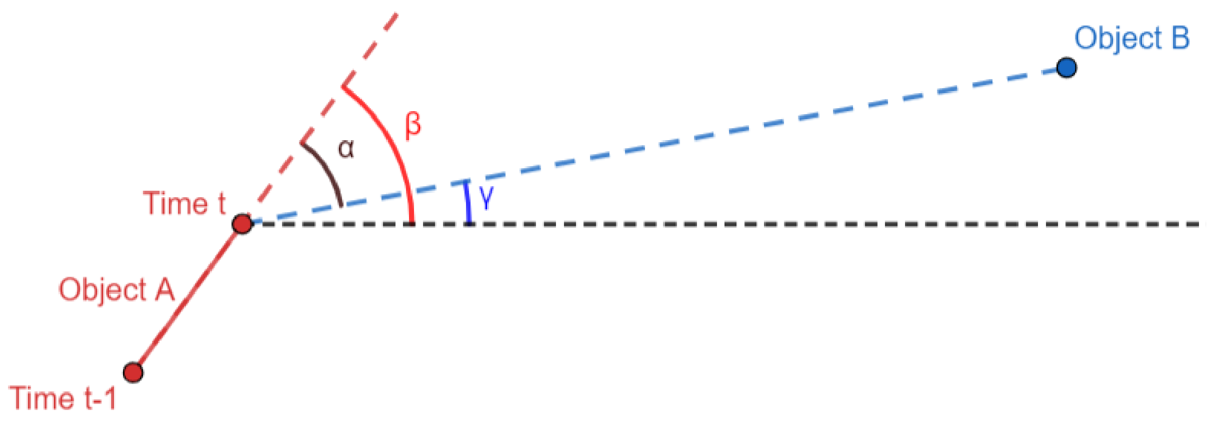


Figure 24: Angle difference between the direction of movement of object A and the orientation of object B relative to object A

The algorithm to calculate the angle difference is presented in Table 1. The input variables for this application are the previous position of object A, the current position of object A, and the position of object B.

Table 1: Algorithm to calculate the angle difference between the direction of an object and its goal

Algorithm 1: angle difference

```
1. FUNCTION calculate_angle_difference(previous_position, current_position, goal_position)
2.     $\text{direction} = \tan\left(\frac{\text{current\_position}[1] - \text{previous\_position}[1]}{\text{current\_position}[0] - \text{previous\_position}[0]}\right)$
3.     $\text{goal\_direction} = \tan\left(\frac{\text{current\_position}[1] - \text{goal\_position}[1]}{\text{current\_position}[0] - \text{goal\_position}[0]}\right)$
4.     $\text{angle\_difference} = \text{goal\_direction} - \text{direction}$
5.     RETURN angle_difference
6. ENDFUNCTION
```

By combining these three variables through ProbLog, it is possible to infer the probability of collision between the two objects through the weighted joint probability of collision based on each input variable. Figure 25 gives an overview of the implementation in ProbLog for the collision case.

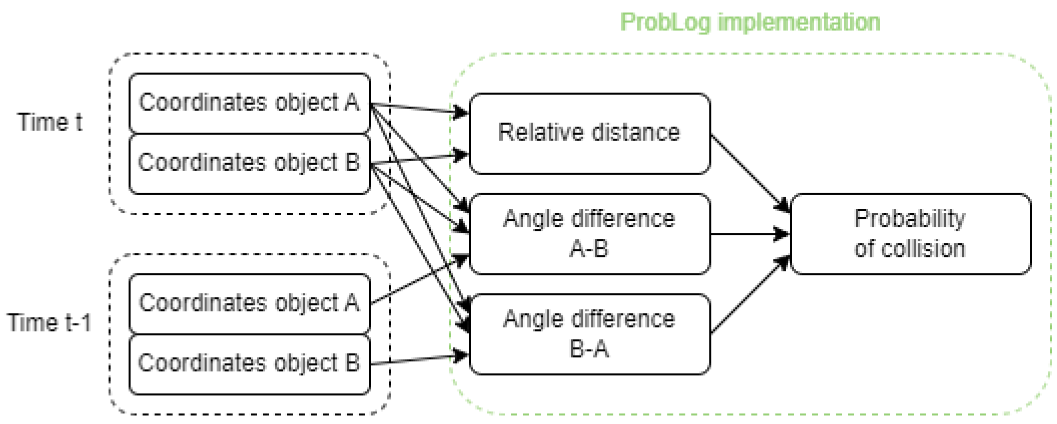


Figure 25: Schematic overview of the probabilistic model implemented in ProbLog

3.3 Results

The use of ProbLog to determine the likelihood of a collision between two objects did not produce satisfactory results. Although it suggested a potential collision, it failed to distinguish between an actual collision and just two objects crossing each other. This was due to the fact that the variables related to the direction of the objects were assigned low weight values in the calculations. Increasing the weight of these variables resulted in highly inconsistent and unreliable outcomes. Figure 26 and Figure 27 illustrate the results of the ProbLog scripts applied to two scenarios.[1]

[1] Videos of all sequences can be found in a Google Drive folder

In the first case, depicted by Figure 26, two objects crossed each other. In this figure it is visible that the probability increases when the two objects are moving towards each other and decreases once the objects have passed each other around frame 80, while there actually should be a probability close to 0 once the objects have crossed and are moving away from each other.

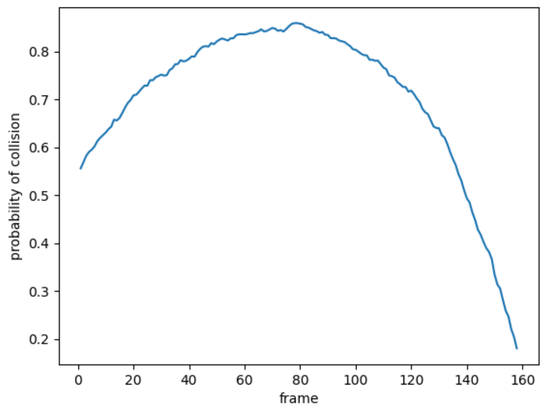


Figure 26: Result of the ProbLog script to calculate the probability of collision for two objects that cross each other

Figure 27 illustrates the results when two objects are moving alongside each other. Here, the probability remains more or less constant round 50 per cent, while the probability should be lower since the objects are not moving towards each other.

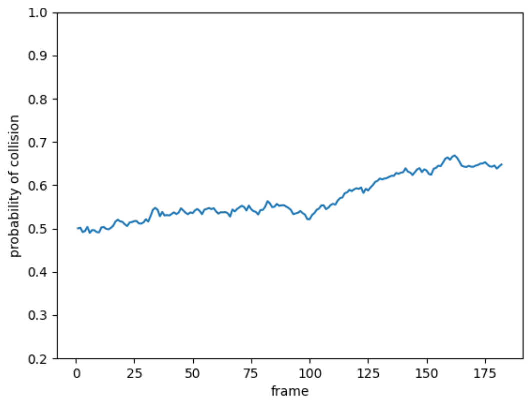


Figure 27: Result of the ProbLog script to calculate the probability of collision for two objects that move parallel to each other in the same direction

Ultimately, while ProbLog appeared to be a viable approach, its implementation in this application was unsuccessful. This can be attributed partly to a lack of relevant examples to draw from and a prevailing belief that alternative methods would yield better results.

4 Methods

With ProbLog turning out to be an unfitting modelling tool for the targeted application, the switch was made to one of the probabilistic graphical models discussed in the literature study, called a dynamic Bayesian network. As mentioned in the study, a dynamic Bayesian network is a type of probabilistic graphical model that is great at representing knowledge about an uncertain domain. Because of this reason, it is implemented in this application, first for the collision case, in which the goal is to calculate the probability of collision between two objects, and later for the assembly case, in which the goal is to classify the assembly activity that is being performed.

4.1 Collision detection

To first quickly validate the use of a dynamic Bayesian network, it is first applied on the collision case through one of the Python libraries discussed in 2.4.2, called pgmpy.

4.1.1 Dynamic Bayesian network based on absolute variables

Similar to the ProbLog script in section 3.2, the initial stage involved the development of a model. To construct a Bayesian network, an assessment of variables that impact the probability of collision is essential. Inspired by the presented ProbLog model, the positions and the relative motion directions of the objects were selected as parent nodes. Furthermore, the velocity of each object was incorporated as parent nodes as well. The model's schematic representation is illustrated by Figure 28. At any given time t, the probability of collision, represented by the child node, is determined by computing the values of the parent nodes, namely the positions, velocities, and motion directions of both objects.

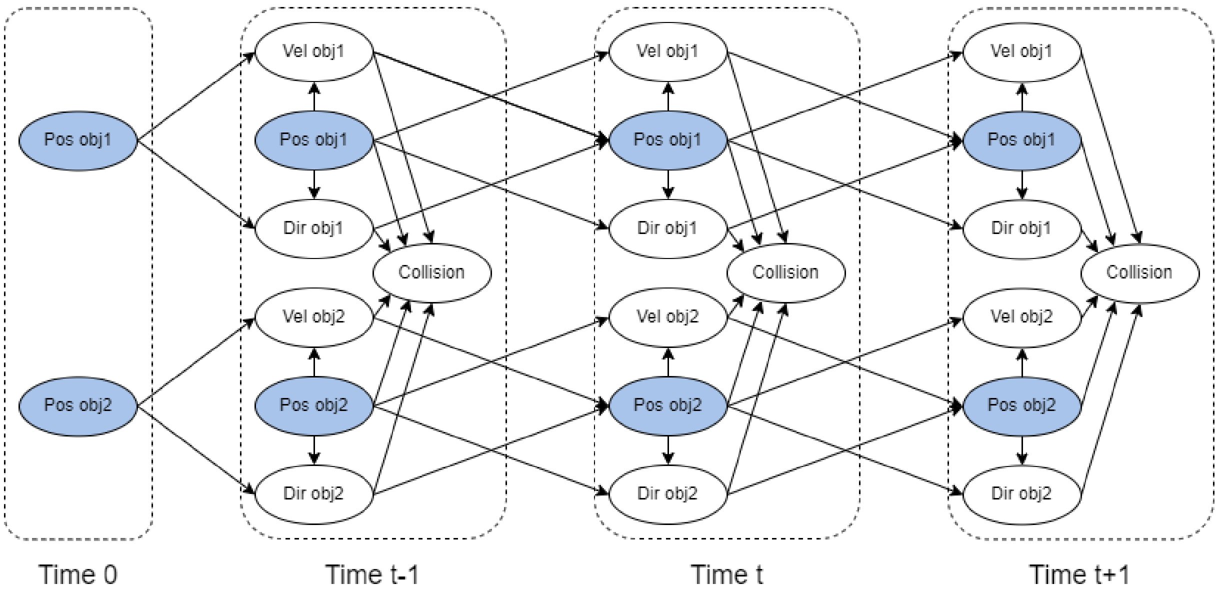


Figure 28: Initial model of the dynamic Bayesian network for the collision case

4.1.2 Dynamic Bayesian network based on relative variables

Instead of utilising the objects' absolute positions, velocities and directions, it is more appropriate to take into account the relative values of these variables, as they offer greater predictive capacity for detecting a collision between two objects. With this idea in mind, the model depicted in Figure 29 was conceived. By considering the positions of object 1 and 2 at time t and $t + 1$, the relative distance, orientation, velocity and motion direction of the two objects are determined. These variables serve as parent nodes for the prediction of the relative distance at time $t + 1$, positioned in an intermediate layer of the network. The probability of collision, represented by the child node, is then determined by assessing the predicted relative distance at the next timestep.

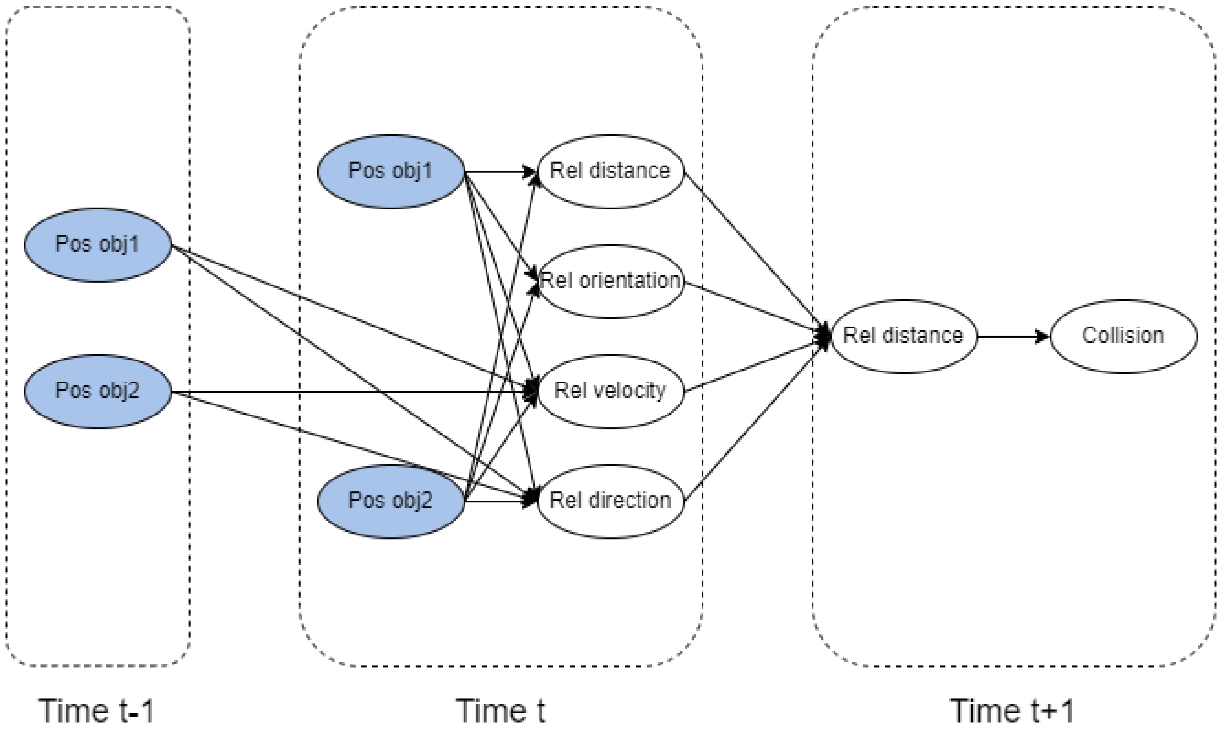


Figure 29: Dynamic Bayesian network model taking into account the relative instead of the absolute variable values for the collision case

Figure 30 illustrates an alternative model proposal for the dynamic Bayesian network. In this model, unlike the model of Figure 29, the velocities and motion directions of the two objects at time t are determined in addition to their relative distance. By taking into account the position, velocity and direction, it is possible to approximate the position of each object at time $t+1$. The difference in relative distance between time t and $t+1$, coupled with the relative distance at time $t+1$, serve as the parent nodes to ascertain the probability of collision. Because of the promising results yielded by the previous model and the fact that the collision case was only an intermediate step, this model was not pursued further.

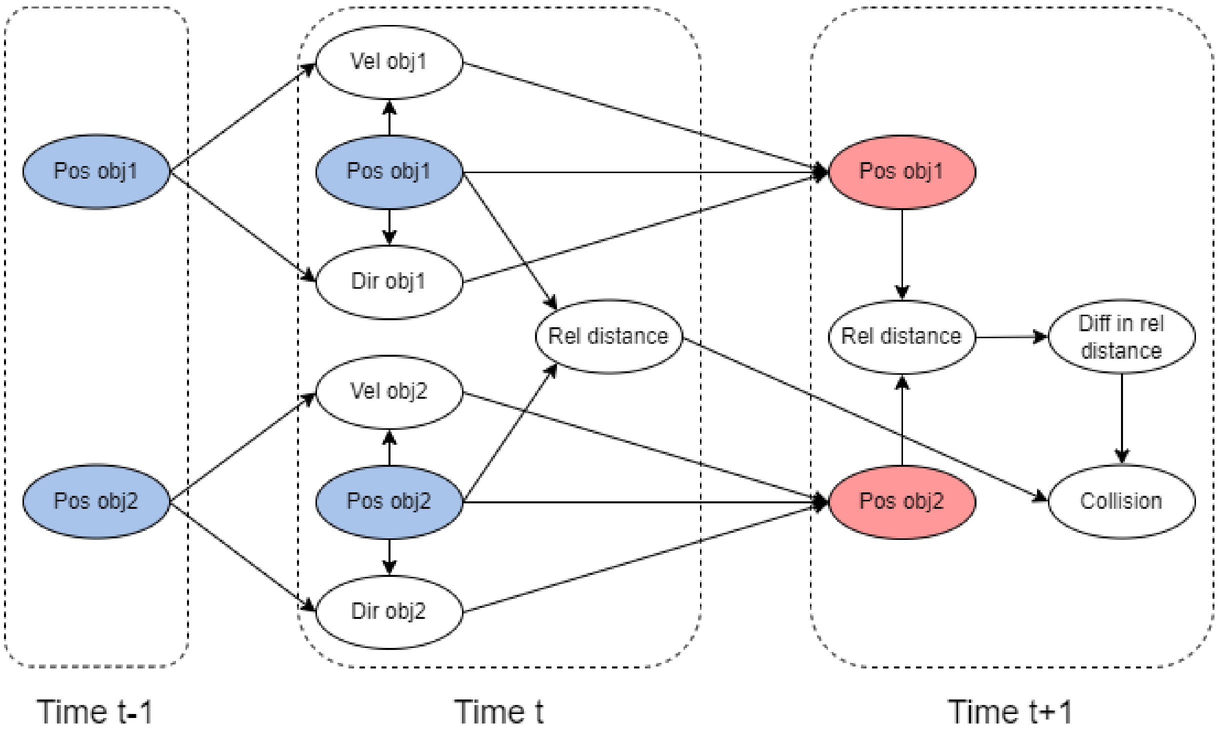


Figure 30: Dynamic Bayesian network model for the collision case based on the prediction of the positions of the two objects

To prevent having to manually define the probability distribution tables, a new approach was applied to compute the CPDs probability values. First, a sequence of two ArUco markers moving randomly and possibly colliding is recorded. Then, each frame of a set of frames that represent a specific movement (e.g. moving towards each other or moving away from each other), is labelled with whether or not the outcome of that movement is a collision. By finally dividing, for each set of inputs, the amount of frames with that input and that are labelled with a collision as outcome by the total amount of times that that set of input occurs, the CPDs probability values are computed. By repeating this process for multiple sequences, more training data will be generated and the CPD will become more reliable. This process is formulated by:

$$P(Collision|\ set\ of\ input\ variables) = \frac{\#frames|\ collision, set\ of\ input\ variables}{\#\ frames|\ set\ of\ input\ variables} \qquad (2)$$

4.2 Assembly activity recognition

After achieving promising results with the final model for collision detection, the focus was shifted towards the assembly case, which involves the recognition of the assembly activity being performed.

4.2.1 Three-part assembly

To this end, a model was developed specifically for the assembly of the simplified Bourjault ballpoint pen, which comprises three distinct parts: the bottom, body, and cap. The model takes inspiration from the collision detection case and employs the relative distance, relative velocity, and relative motion direction of the parts as input data.

To minimise the number of variables involved and thereby reduce the amount of training data required for reliable computation of the CPD, only the relative parameters between the primary component being manipulated and the remaining parts are considered. Figure 31 provides an illustration of the model.

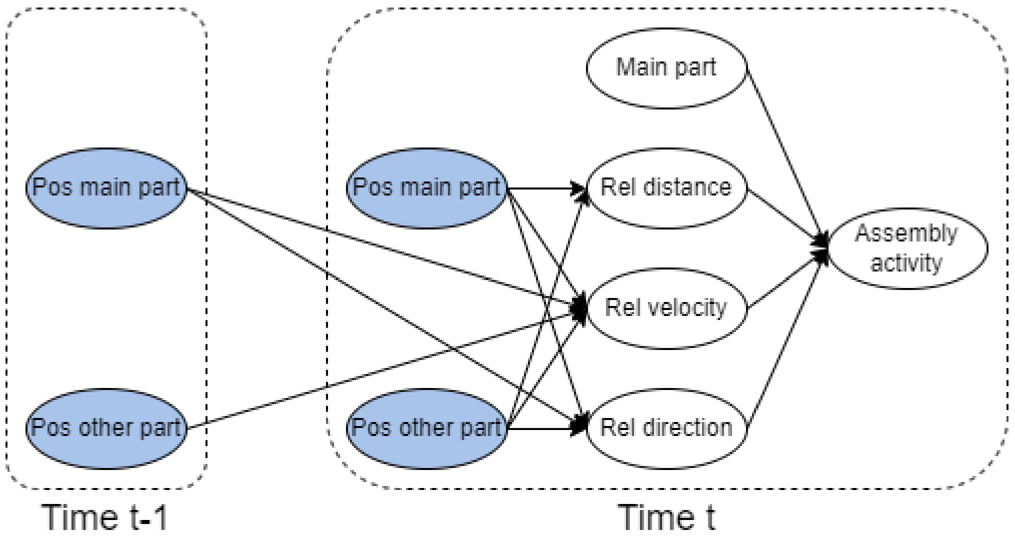


Figure 31: Model of the dynamic Bayesian network used for activity recognition in the simplified assembly case

4.2.2 Complete assembly

After successful validation of the simplified assembly model through test assemblies, it was expanded to encompass the full assembly of the Bourjault ballpoint pen. The full assembly comprises six parts: the bottom, ink, cartridge, body, head, and cap. These parts are visualised in Figure 32.

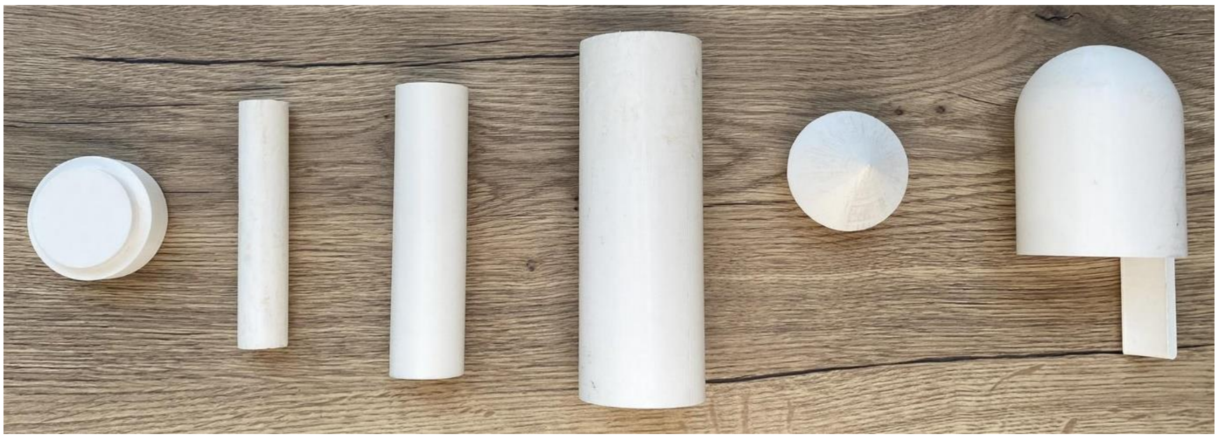


Figure 32: Assembly parts of the Bourjault ballpoint pen, from left to right: bottom, ink, cartridge, body, ballpoint, cap

The depicted model presented in Figure 31 remains unchanged. In an effort to minimise the number of variables involved, as opposed to the scenario of the three-part assembly where the relative parameters between the primary part and all other parts are considered, only the relative parameters between the primary part and parts that bear relevance to it are taken into consideration. Table 2 provides the relevant parts for each component of the Bourjault ballpoint pen. For instance, incorporating the relative distance, velocity, and direction between the bottom and body help to better estimate the assembly activity. However, incorporating the relative information between bottom and cap would solely augment the volume of training data required for a reliable computation of the CPD, without significantly contributing relevant information.

Table 2: The main parts with their respective relevant parts in the complete assembly case

Main part	Relevant parts
Bottom	Body
Ink	Cartridge
Cartridge	Body, ink
Body	Bottom, cartridge, head
Head	Body, cap
Cap	head

5 Experiments

With the introduction of the different models that were constructed in chapter 4, it was decided to pursue the model based on relative variable values further in an experimental phase for collision detection. In a first attempt, the CPD is manually computed, later this is done automatically through the labelling process. Later, the finalised model depicted by Figure 31, is implemented to classify operator activities in an assembly case. First, in an intermediate step, this is done for the simplified version of the Bourjault ballpoint pen. Next, the process is expanded for the ballpoint pen comprising all six parts.[1]

5.1 Collision detection

5.1.1 Manual computation of the CPD

In a first attempt for the collision case, the model from Figure 29 based on the relative variable values was implemented in Python using pgmpy. The first step entailed defining the model's structure, which comprised three layers of nodes. In this case, the first layer consists of the variables calculated from the input data: the relative distance, orientation, velocity and motion direction between the objects. These are the parent nodes of the variable in the intermediate layer: the prediction of the relative distance at the subsequent time step. Finally, this variable functions as the parent node for the node in the third layer, representing the probability of collision. In addition to nodes, a dynamic Bayesian network also consists of edges that represent the conditional probabilities between these nodes. These conditional probabilities are defined in conditional probability distributions for every child node, given their parent nodes as evidence. At this stage in the project, the values of the CPDs were manually established based on the author's experience. This entails defining the probability distribution of the child node for each combination of values of the parent nodes. In order to use the relative distance, orientation, velocity, and motion direction, the values of these variables need to be discretised. For this method of manually defining the CPD, the discretisation needed to be very broad. For the relative distance, orientation, velocity and motion direction, the nodes could take on three, four, two, and four possible values respectively. The possible values for these nodes are represented in Table 3.

Table 3: Discretisation of the variable values in the first layer for the collision case in which the CPD is manually defined

Relative distance	Relative orientation	Relative velocity	Relative direction
Far	Northern	Fast	North
Average	Eastern	Slow	East
Close	Southern		South
	Western		West

The CPD that represents the edge between the first an intermediate layer contains $(3 * 4 * 2 * 4 * 3)$ 288 elements. This illustrates that even when the discretisation is performed very broad, the CPD already contains a significant amount of elements. For instance, if one wants a finer discretisation of the relative orientation and relative motion direction that include the intercardinal directions, the CPDs size would increase tremendously and contain 1152 elements, and manually defining the CPD would take significantly more time. Table 4 represents a snippet of this CPD.

Table 4: Snippet of the CPD between the first and intermediate layer

	Far	**Average**	**Close**
Far, northern, fast, north	1	0	0
Far, northern, fast, east	0.7	0.2	0.1
Far, northern, fast, south	0.1	0.8	0.1
Far, northern, fast, west	0.7	0.2	0.1
Far, northern, slow, north	1	0	0
Far, northern, slow, east	0.95	0.05	0
Far, northern, slow, south	0.4	0.55	0.05
Far, northern, slow, west	0.95	0.05	0

5.1.2 Automatic computation of the CPD

In order to compute the CPD with the method of labelling the data, five random movement sequences were recorded. As explained in the previous chapter, each frame is labelled with the set of input values and whether or not the object collides at the end of the movement performed in that sequence. The set of input variables are slightly different than before: relative distance, relative velocity, relative motion direction of object 1 towards object 2 and relative motion direction of object 2 towards object 1. Now that the CPD is not defined manually anymore, the new automatic method allows for a finer discretisation of these variables. Table 5 represents the possible values of each variable.

Table 5: Discretisation of the variable values in the first layer for the collision case in which the CPD is automatically constructed

Relative distance	**Relative velocity**	**Relative direction 1**	**Relative direction 2**
Collided	Very fast	Straight towards	Straight towards
Very close	Fast	Mostly towards	Mostly towards
Close	Normal	Slightly towards	Slightly towards
Intermediate	Slow	Not towards	Not towards
Far	Very slow	Away	Away
Very far	Not moving		
Extremely far			

Next, for each frame, the set of input values are calculated based on the coordinates of the objects in that frame and in the previous frame. Once this is done for every frame in the sequence, every frame is also manually labelled with the outcome of the movement sequence in which that frame occurred. Table 6 includes a snippet of this operation performed on the first recording of the movement sequences.

Table 6: Snippet of manually labelling the training data for collision detection in the collision case

Frame	Set of input variables (rel. dis., rel. vel., rel. dir. 1-2, rel. dir. 2-1)	Outcome
621	Intermediate, very slow, away, not towards	No collision
622	Intermediate, very slow, slightly towards, not towards	No collision
623	Intermediate, very slow, slightly towards, not towards	No collision
624	Intermediate, very slow, slightly towards, not towards	No collision
625	Intermediate, slow, away, slightly towards	Collision
626	Far, slow, straight towards, slightly towards	Collision
627	Far, slow, straight towards, slightly towards	Collision
628	Far, very slow, slightly towards, not towards	Collision
629	Far, slow, straight towards, slightly towards	Collision
630	Far, slow, slightly towards, slightly towards	Collision
631	Far, slow, slightly towards, slightly towards	Collision

The CPD can now be constructed by dividing the number of times a set of input variables is labelled with a collision by the total number of times that set of input variables occurred in the training data, described by Formula 2. Once this is done for all five movement sequences, the accuracy of the model can be validated by recording a sixth movement sequence and predicting for each frame of that sequence whether or not a collision will occur.

5.2 Assembly activity recognition

5.2.1 Three-part assembly

The dynamic Bayesian network depicted in Figure 31, requires a computation of the CPD, which is carried out using the automatic methodology outlined in the preceding section. The input variables are discretised according to the specifications provided in Table 7.

Table 7: Discretisation of the input variables for assembly activity recognition

Main part	Relative distance	Relative velocity	Relative direction
Cap	Connected	Moving	Towards
Body	Very close	Not moving	Not towards
Bottom	Close		Away
	Intermediate		
	Far		

The ballpoint pen's assembly process involves two distinct activities: attaching the bottom to the body and placing the cap on the body, leading to six potential assembly states. These states include no assembly activity has yet been performed, one of the two activities being performed, one of the two activities having just been completed, or the operator moving the parts around.

To compute the CPD, each frame of a recorded assembly is assigned a label indicating the set of input variables and the current state of assembly. By determining the probability of each state of assembly for each set of input variables, the CPD can be calculated. Table 8 presents a snippet of this operation performed on the frames of a single recorded assembly. Thirty assemblies were recorded and labelled as training data for this purpose.

Table 8: Snippet of manually labelling the training data for activity recognition in the three-part assembly case

Frame	Set of input variables (main part, rel. dis. 1-2, rel. dis. 1-3, rel. vel. 1-2, rel. vel. 1-3, rel. dir. 1-2, rel. dir. 1-3)	State of assembly
160	Body, connected, intermediate, n. moving, n. moving, away, away	Bottom attached
161	Body, connected, intermediate, n. moving, n. moving, away, away	Bottom attached
162	Body, connected, intermediate, n. moving, n. moving, away, away	Bottom attached
163	Cap, close, intermediate, moving, moving, towards, n. towards	Placing cap
164	Cap, close, intermediate, moving, moving, n. towards, n. towards	Placing cap
165	Cap, close, intermediate, moving, moving, n. towards, n. towards	Placing cap
…	…	…
199	Cap, connected, very close, n. moving, n. moving, towards, towards	Placing cap
200	Cap, connected, very close, n. moving, n. moving, towards, towards	Completed
201	Cap, connected, very close, n. moving, n. moving, towards, towards	Completed
202	Cap, connected, very close, n. moving, n. moving, towards, towards	Completed

5.2.2 Complete assembly

For the complete assembly of the Bourjault ballpoint pen, the discretisation of the input variables remain identical to the discretisation of the input variables in the three-part assembly case, illustrated in Table 7. In the complete assembly case, there are five different assembly activities that can be performed: loading of the ink, inserting of the cartridge, attaching of the bottom, attaching of the head, and placing of the cap. This means that the amount of assembly states increases to eleven: no assembly activity has yet been performed, one of the five assembly activities is being performed, one of the five activities has just been completed, or parts are being moved around.

In total, 35 assembly sequences were recorded and manually labelled in order to compute the CPD and train the model. A snippet of this operation performed on one of the assembly sequences is presented in Table 9.

Table 9: Snippet of manually labelling the training data for activity recognition in the six-part assembly case

Frame	Set of input variables (main part, rel. dis. 1-2, rel. dis. 1-3, rel. dis. 1-4, rel. vel. 1-2, rel. vel. 1-3, rel. vel. 1-4, rel. dir. 1-2, rel. dir. 1-3, rel. dir. 1-4)	State of assembly
42	Body, intermediate, intermediate, far, n. moving, n. moving, n. moving, away, n.towards, away	No assembly activity
43	Body, intermediate, intermediate, far, n. moving, n. moving, n. moving, n. towards, away, n. towards	Attaching bottom
44	Body, intermediate, intermediate, far, n. moving, n. moving, n. moving, away, away, away	Attaching bottom
…	…	…
97	Body, connected, intermediate, intermediate, n. moving, n. moving, n. moving, n. towards, n. towards, n. towards	Attaching bottom
98	Body, connected, intermediate, intermediate, n. moving, n. moving, n. moving, away, away, away	Bottom attached
99	Body, connected, intermediate, intermediate, n. moving, n. moving, n. moving, away, away, away	Bottom attached
…	…	…
141	Body, connected, intermediate, intermediate, n. moving, n. moving, n. moving, away, away, away	Bottom attached
142	Body, connected, intermediate, intermediate, n. moving, n. moving, n. moving, away, away, away	Inserting cartridge
143	Cartridge, intermediate, intermediate, none, n. moving, n.moving, none, away, away, none	Inserting cartridge

Each part has a different amount of relevant parts. For example, the body has three relevant parts, the bottom, the cartridge, and the head, while the ink only has one relevant part, the cartridge. In the last row of Table 9, where the main part is the cartridge, there are only two relevant parts, this is why 'rel. dis. 1-4', 'rel. vel. 1-4', and 'rel. dir. 1-4' have a value 'none'.

6 Results and discussion

In this chapter, the results of the experiments explained in chapter 5 are presented and discussed. First, for the collision case, the model for collision detection based on the manual computation of the CPD is reviewed through five possible movement scenarios. Next, with the switch from manual to automatic computation, the accuracy of the model was also tested by comparing the prediction of collision with whether or not an actual collision occurred. This methodology is also followed for the assembly case with the simplified ballpoint pen. For the activity recognition in the assembly of the complete ballpoint pen, a seven-fold cross-validation is performed in order to assess the performance of the finalised model. At last, the fully trained finalised model is also applied for an additional assembly scenario.[1]

6.1 Collision detection

6.1.1 Manual computation of the CPD

For the first attempt at implementing Bayesian networks in Python through pgmpy, five scenarios were examined: both a head-to-head and a sideways collision between two objects, two scenarios in which the objects move away from each other, once head-to-head and once sideways, and finally, two objects crossing over each other. These possible scenarios are illustrated in Figure 33.

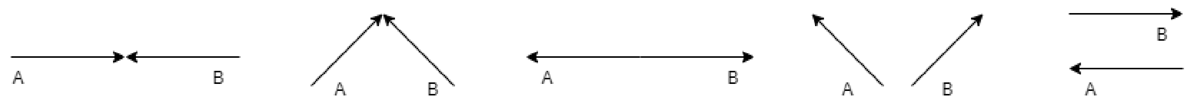


Figure 33: From left to right: a head-to-head and sideways collision between two objects A and B, a head-to-head and sideways separation of A and B, and a cross between A and B

The results of these scenarios implemented in the model are visible in Figure 34 to Figure 38.

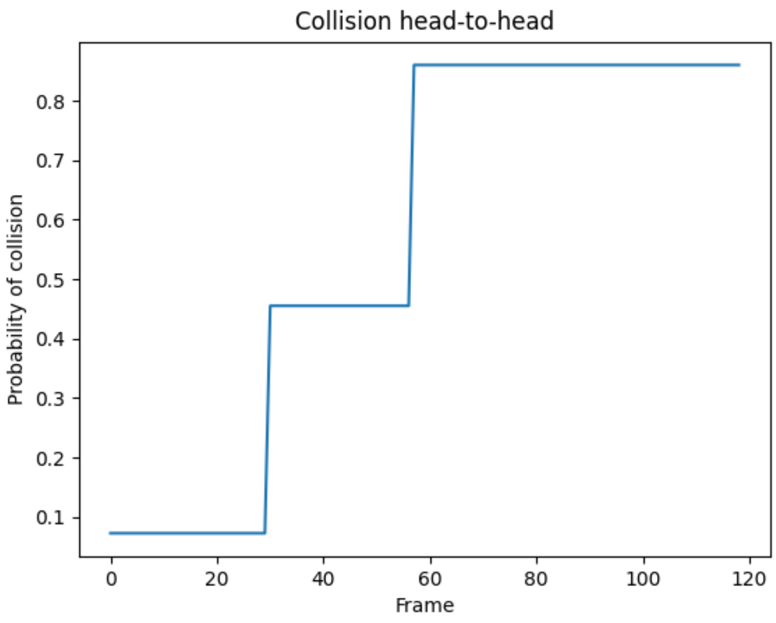


Figure 34: Probability of collision in the scenario where a head-to-head collision occurs

When a head-to-head collision occurred, the probability of collision increases over time. The increment happens in large ‘jumps’, this is because of the broad discretisation of the variables.

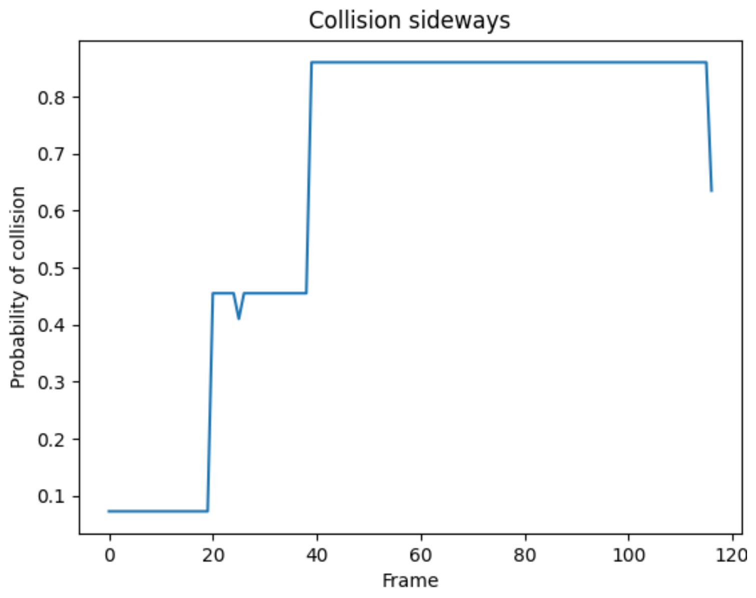


Figure 35: Probability of collision in the scenario where a sideways collision occurs

When a sideways collision occurs, the probability of collision increases as well. A small sudden spike is visible in the figure. This is due to a small change in the actual value of one of the variables that creates a change in the discrete value of that variable, which results in a sudden change of probability. The probability decrease at the end of the experiment is because of a small separation after collision.

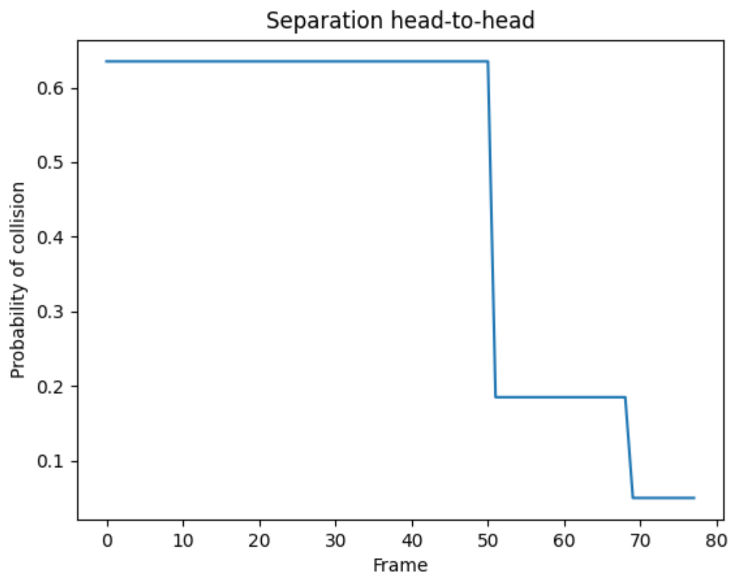


Figure 36: Probability of collision in the scenario where a head-to-head separation occurs

A head-to-head separation of the objects results in a decrease of probability, once again in significant jumps.

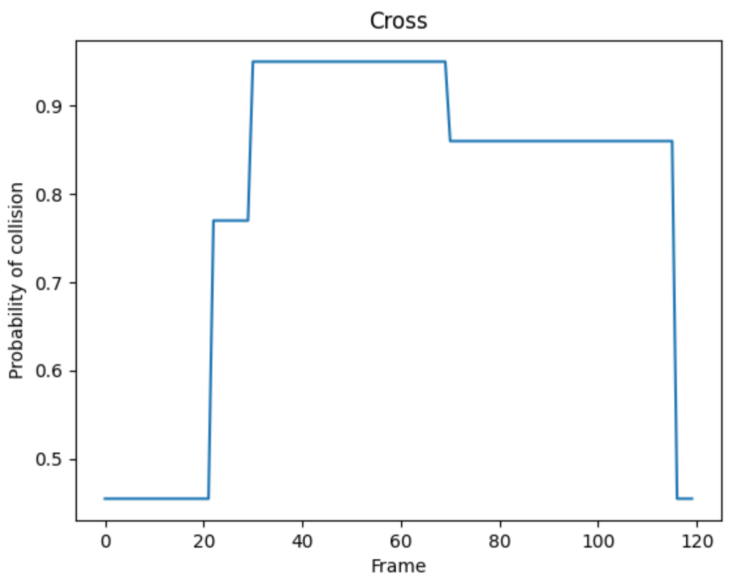


Figure 37: Probability of collision in the scenario where a cross occurs

If the two objects cross each other, it is visible that the probability of collision increases when the objects are moving towards each other, but decreases when it becomes apparent that they will cross instead of collide.

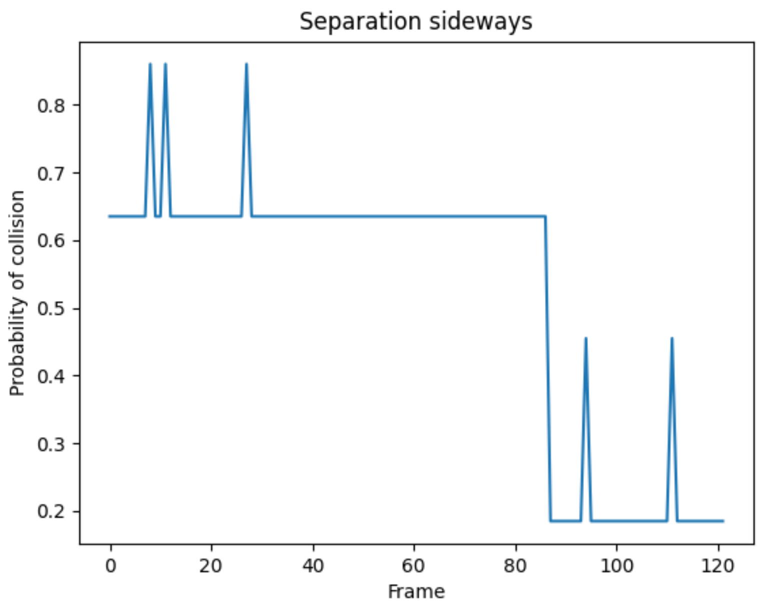


Figure 38: Probability of collision in the scenario where a sideways separation occurs

During a sideways separation of the objects, there is a general decrease in probability as well. Once again, there are some sudden spikes visible, this happens for the same reason as mentioned for the sideways collision experiment.

From these results, it is clear that the program is able to provide a clear notion whether or not the objects would collide. The main problem with this approach was that the variables needed a broad discretisation, indicated by the mentioned large ‘jumps’ in the results, because of the manual definition of the CPDs. In order to be able to use finer discretisation, a different approach was necessary.

6.1.2 Automatic computation of the CPD

In order to test the new model that allowed for finer discretisation of the variables, a new movement sequence was recorded and a prediction on whether or not a collision would occur was defined through the model. By counting the amount of frames in which the prediction was right, and dividing that by the total number of frames, the performance level of the model can be measured. This gave an accuracy of 68 per cent. Multiple other methods, such as precision, recall, and F1-score, exist for measuring the performance of a model, but since the collision case was only an intermediate step and the results were promising, these other methods where not applied.

6.2 Assembly activity recognition

6.2.1 Three-part assembly

In the simplified assembly case, to assess the model’s efficacy, an additional set of five assemblies were recorded for validation purposes. By utilising the trained model to determine the assembly activity carried out in each frame and comparing these results with the actual assembly activity performed in that frame, the accuracy of the model can be determined. Out of the 1556 frames present in the five test assemblies, the model accurately recognised 1141 frames, resulting in an overall accuracy rate of 73%.

6.2.2 Complete assembly

To assess the performance of the obtained model for the complete assembly scenario, a 7-fold cross-validation approach was employed. This involved utilising 30 assembly sequences as training data and the remaining five sequences as test data to calculate metrics such as accuracy, precision, recall, and F1-score. The final model yielded an accuracy of 75 per cent, a precision of 76 per cent, a recall of 75 per cent and an F1-score of 74 per cent.

Furthermore, when the final model trained on all 35 assembly sequences was applied to recognise operator activity in an additional assembly sequence, it achieved an accuracy of 77 percent.

Figure 39 illustrates the results using bar plots, where each bar plot represents the actual activity performed, and each bar in their respective plot represents the amount of times that each activity is recognised. For example, in the third plot corresponding to the 'loading ink' activity, six frames were labelled as 'no assembly activity', 92 frames were correctly labelled as 'loading ink', 19 frames were labelled as 'ink loaded', and three frames were labelled as 'cartridge inserted'.

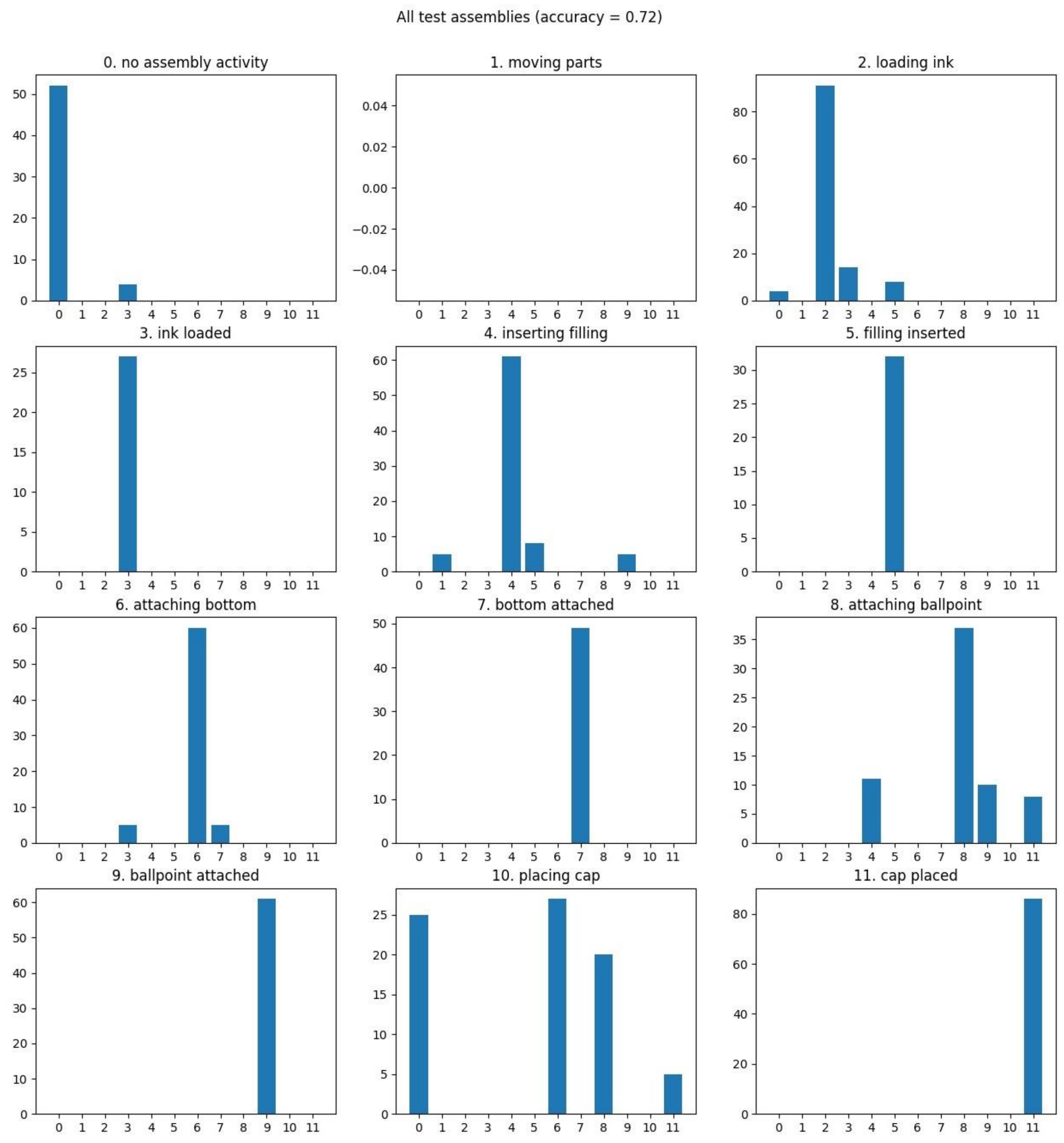


Figure 39: Bar plots for each actual activity in which every bar represents the amount of times that an activity is recognised

These results can also be presented in a normalised confusion matrix as in Figure 40. The diagonal of the confusion matrix indicates that the activity is correctly recognised in most cases. Most of the errors occur when the cap is being placed. This will be further discussed later.

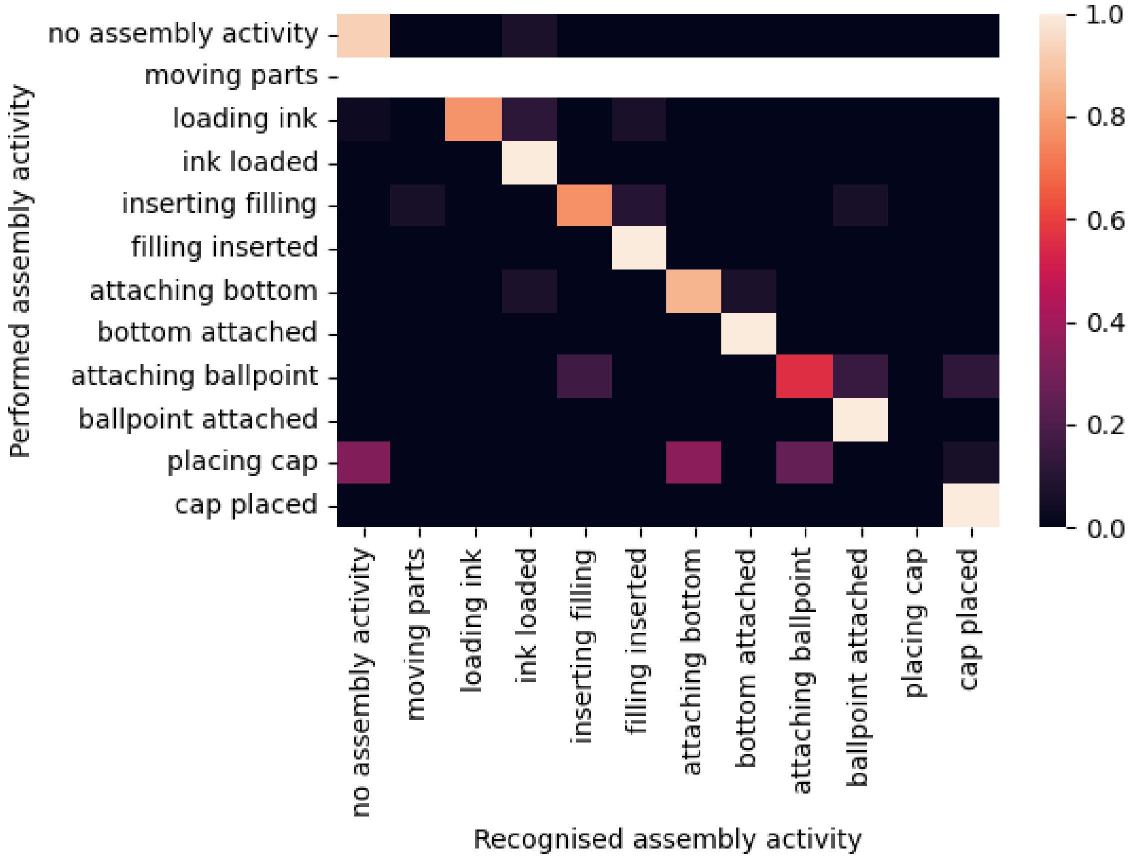


Figure 40: Row-normalised confusion matrix of the results of the additional assembly sequence

Figure 41 offers an alternative representation of the results from the final assembly sequence. In this depiction, the probability of each assembly activity occuring is plotted for each frame. The top horizontal line corresponds to the actual activity performed at that particular frame. This visualisation demonstrates that, on the whole, the actual performed activity alligns well with the activity that has the highest probability according to the model. The most significant errors tend to occur during transitions between different activities.

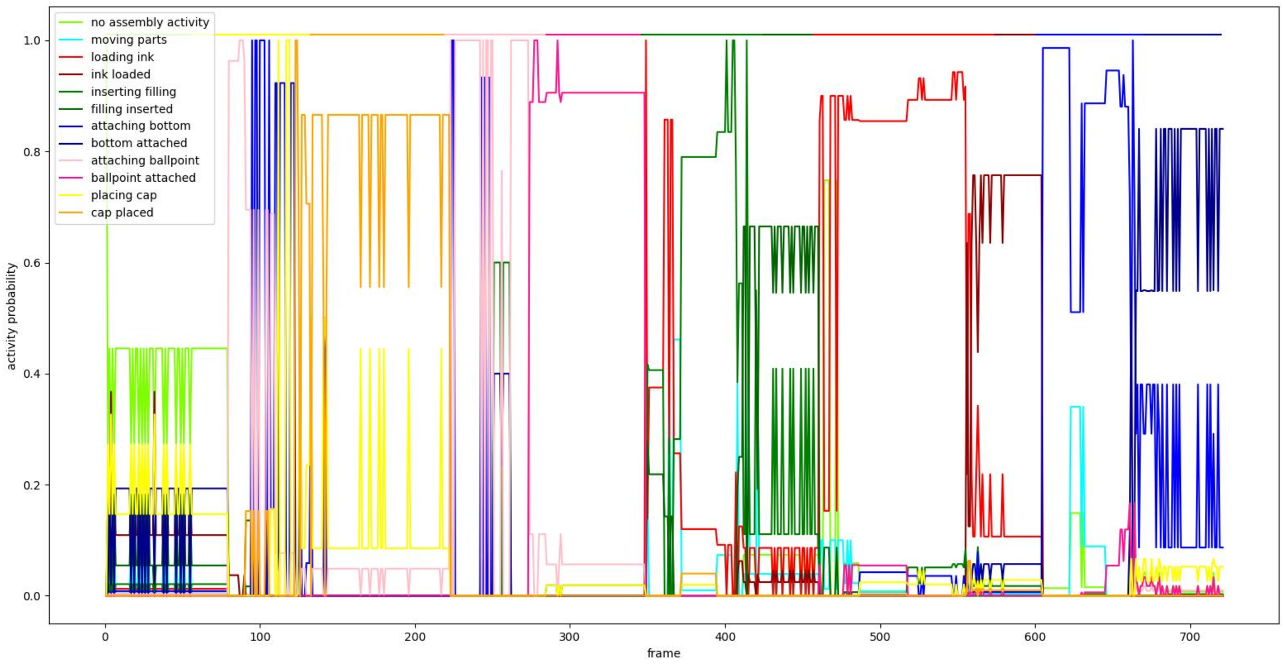


Figure 41: Representation of the probability that each assembly activity occurs at each frame

An interesting observation from the combination of Figure 39 and Figure 41 is that the accuracy of the system improves as the assembly progresses. Even though the current state of assembly is not directly taken into account, the relative positioning of the main object in relation to its relevant objects indirectly accounts for this.

Figure 42 displays a portion of the preceding figure, illustrating the cap placement during assembly. This specific step witnessed a significant number of errors. Subsequent investigation determined that the cause of these errors stemmed from a notable disparity between the way that this activity is performed in this assembly sequence and the ones performed in any of the training assembly sequences.

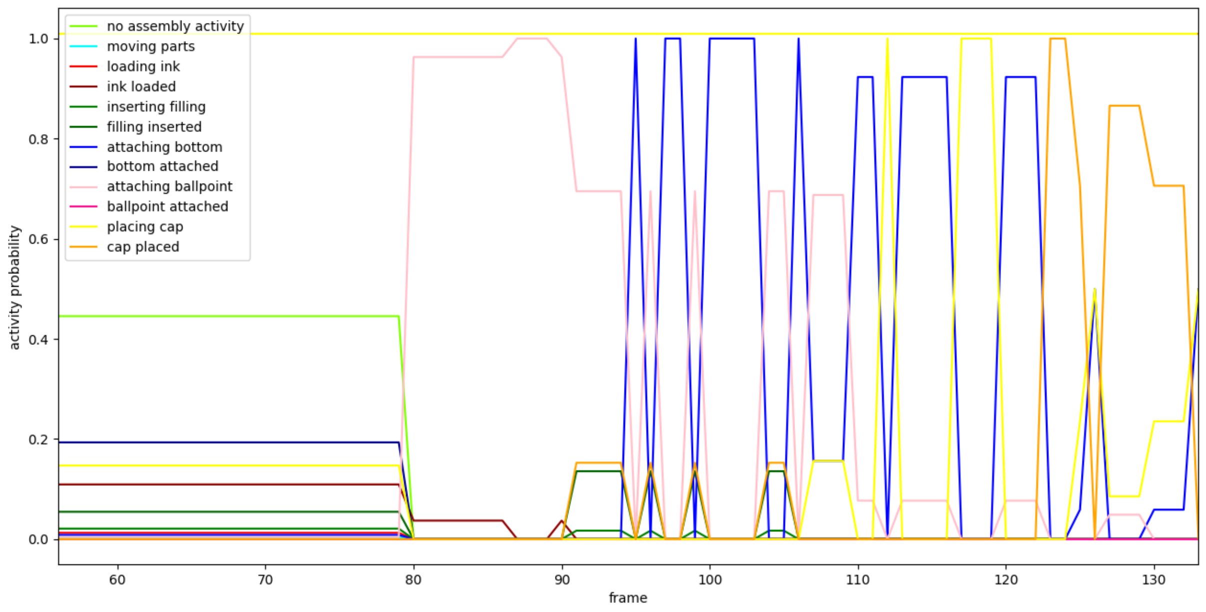


Figure 42: Snippet of the probability representation in which the actual performed activity was the placing of the cap

Figure 43 presents another portion of the assembly sequence, in which the bottom is being attached. In this figure, it is noticeable that in the beginning and in the end, the model inferred that, respectively, the ink is loaded and the bottom is attached. Interestingly, these activities correspond respectively to the preceding and subsequent steps in the assembly sequence. Thus, although the validation suggests that these frames are incorrectly classified, they are not entirely inaccurate. This occurs in each transition between assembly activities and is one of the main reasons for the limited performance scores of the model.

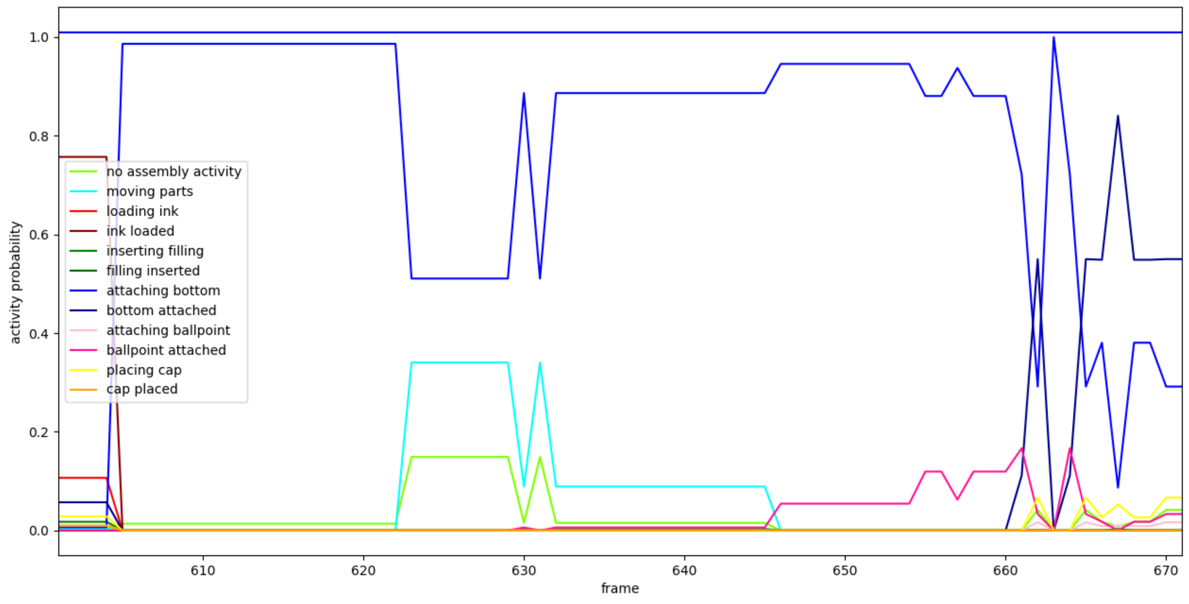


Figure 43: Snippet of the probability representation in which the actual performed activity was the attaching of the bottom

7 6D object pose estimation using PPF matching

The initial aim of the research was to utilise the six-dimensional pose of the assembly parts as input data for activity recognition, instead of relying on the ArUco markers used throughout the thesis. To extract the six-dimensional poses of the objects during the assembly process, PPF matching was employed to obtain the initial pose of each part.

7.1 PPF Data preparation

This method necessitates a point cloud of the assembly part's model and a point cloud of the scene, which represents the initial frame of the assembly sequence. The point cloud of the model was acquired from a CAD model using MeshLab, while the scene was obtained from RGB-D data captured by the Realsense L515 depth camera. Initially, the Realsense D345 depth camera was employed to capture the RGB-D data, but it was later discovered that the camera was malfunctioning.

In an initial attempt to implement PPF matching, the cap's CAD model was converted into a point cloud in PLY format using Blender. Figure 44 depicts the original model and the point cloud of the cap. However, the points within the point cloud were sparse and irregularly distributed, posing challenges for the effective functioning of PPF matching.

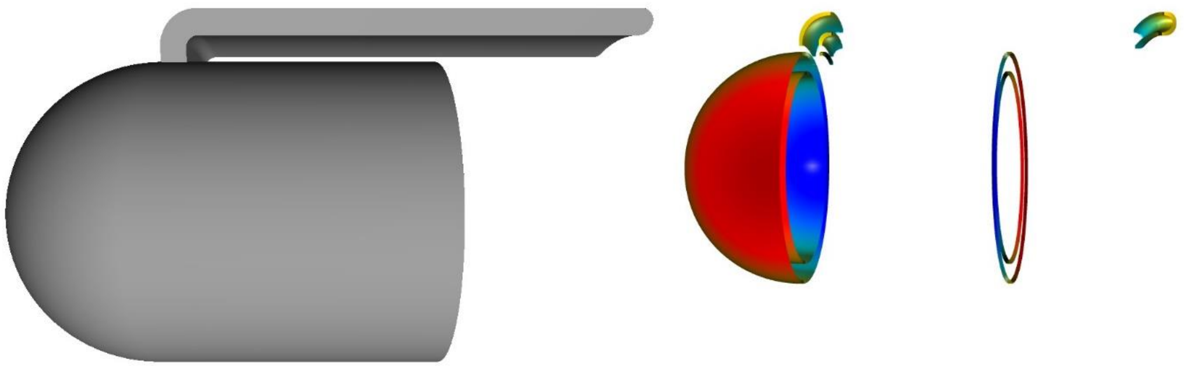


Figure 44: Original model of the cap (left) and the first point cloud of the cap (right)

Initially, the Realsense D345 depth camera was used to record a scene featuring the cap and the body of the Bourjault ballpoint pen, as shown in Figure 45. In this image, it is evident that the depth camera experienced malfunctions, as indicated by the black areas in the depth image. Figure 46 displays the point cloud generated from this image, where the dark dots in the depth image are not detected and therefore not included as points in the point cloud.

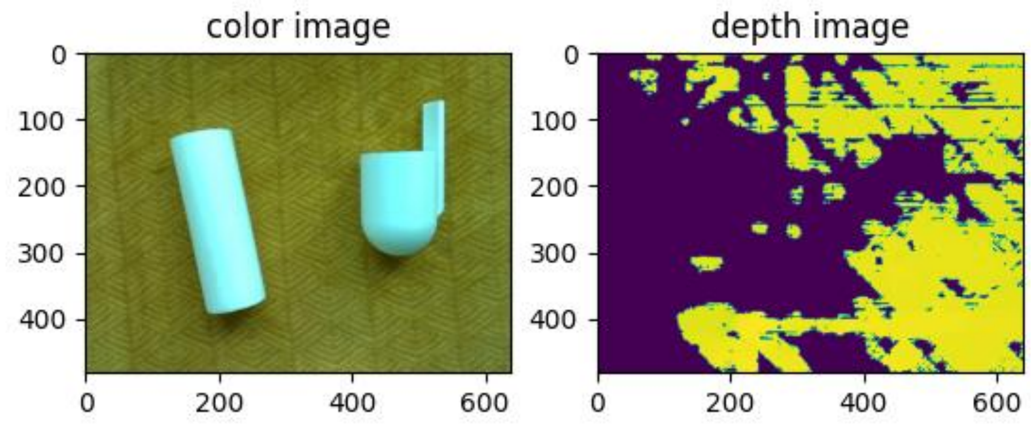


Figure 45: Color (left) and depth (right) image of the scene captured by the Realsense D435 depth camera

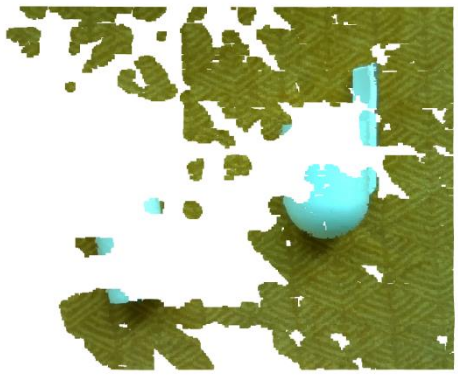


Figure 46: Point cloud of the scene captured by the Realsense D345 depth camera

During the second attempt, two primary issues required resolution: the need to increase the number of points in the model's point cloud, and finding a solution for the dark areas in the scene. To address the first issue, the 'octree depth' value in Blender was raised before converting the mesh into a point cloud. By doing so, more points were sampled from the model's mesh. The outcome of this operation on the cap model is depicted in Figure 47.

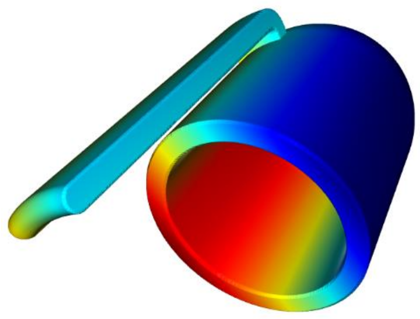


Figure 47: Point cloud of the cap after increasing the point density

To address the problems with the depth camera, a switch was made from using the Realsense D435 to the Realsense L515 depth camera. The L515 camera offered improved resolution and no longer exhibited the black areas in the depth image. As a result, the point cloud of the scene, which showcased the cap and body of the pen placed on a table, is presented in Figure 48.

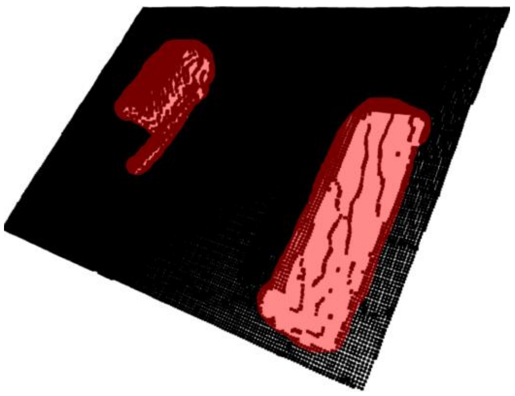


Figure 48: Point cloud of the scene containing the cap and body

Regrettably, even after switching to the new model and scene, the PPF matching algorithm continued to be ineffective. Upon investigation, it was found that the conversion process from mesh to point cloud using Blender was unsuccessful. To address this issue, a different software program called FreeCAD was utilised for the conversion. Figure 49 demonstrates the resulting point cloud of the cap achieved through this new approach.

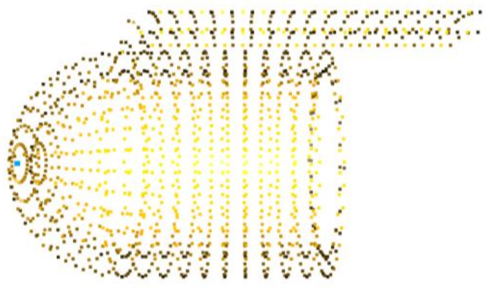


Figure 49: Point cloud of the cap through FreeCAD

7.2 PPF performance

Upon incorporating the new model into the PPF matching algorithm along with the scene captured by the Realsense L515 depth camera, a match was found between the model and scene. This match is depicted in Figure 50.

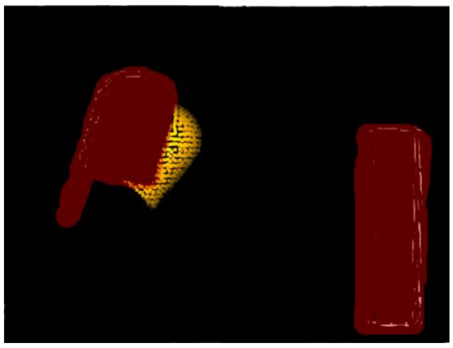


Figure 50: Model and scene matching through the PPF matching algorithm

The model of the cap and the cap in the scene still exhibit slight discrepancies. Since PPF matching seeks pairs of matches between the model and the entire scene, a potential solution was to eliminate irrelevant points in the scene, such as those representing the table. To accomplish this, plane segmentation was employed. Plane segmentation involves fitting a plane through the point cloud in a manner that minimises the sum of distances between each point and the plane. Given the substantial number of points representing the table in comparison to those representing the assembly parts, the fitted plane effectively represents the table within the scene. By removing each point that intersects and lies in close proximity to the plane, only the points representing the assembly parts remain. Figure 51 visually demonstrates this operation implemented on a scene featuring the cap resting on a table.

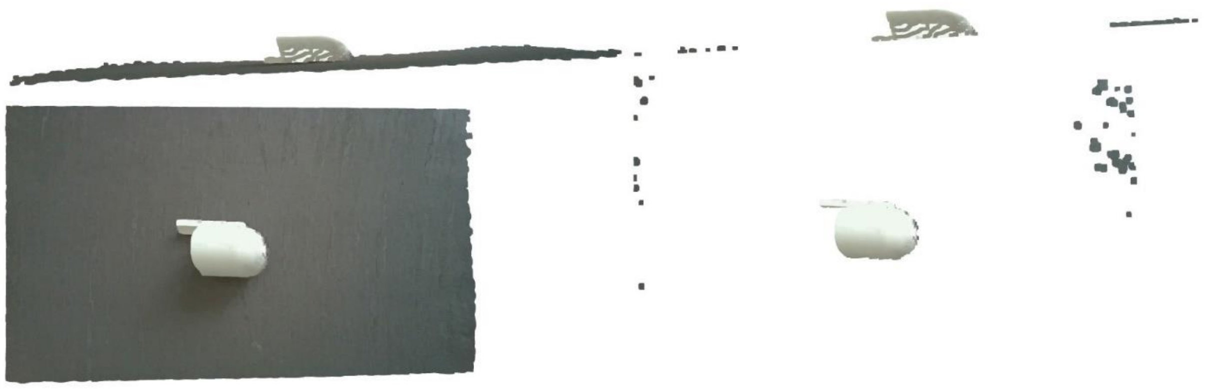


Figure 51: Original scene (left) and scene after plane segmentation (right)

Following the plane segmentation process, there are still some remaining points in the point cloud that are not associated with the cap. These extraneous points can be eliminated by employing a clustering operation. This operation defines clusters of neighbouring points for each point in the point cloud. Two points are considered neighbours if the Euclidean distance between them falls below a specified threshold. Any point that does not have a minimum number of neighbours is removed from the point cloud. The outcome of this operation, conducted on the scene after plane segmentation, is illustrated in Figure 52. Unfortunately, the clustering operation proved to be time-consuming, and therefore it was not further applied.

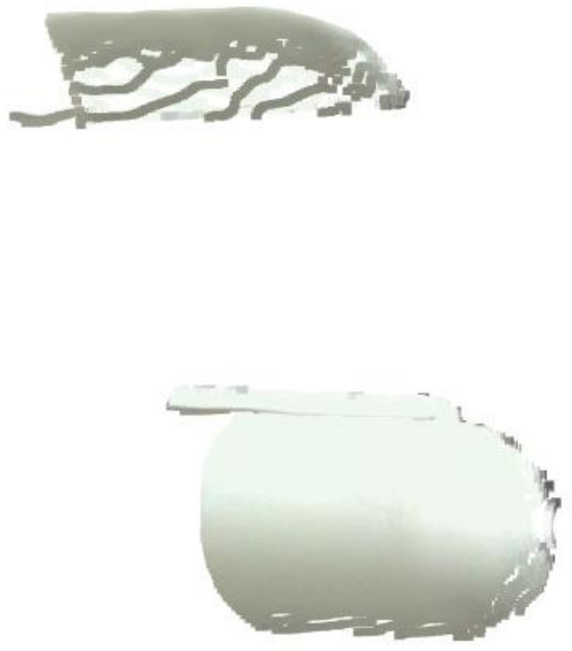


Figure 52: Scene after clustering operation

Upon utilising PPF matching on the model and scene following the plane segmentation process, there was no improvement in pose estimation, as depicted in Figure 53.

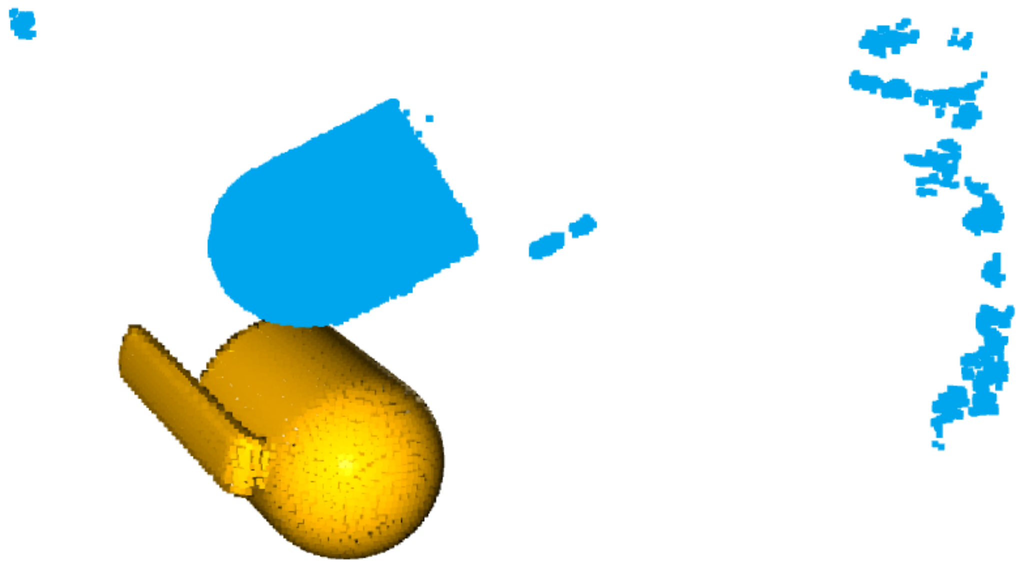


Figure 53: PPF matching after performing plane segmentation on the scene

Despite making modifications to various variables and properties of both the model and the scene, including the size of the model, point density, and conversion method to point cloud, the results did not yield further improvement. The main potential reason for this could be the simple shapes of the parts. This means that the extracted PPF will not be distinctive enough since PPF matching relies on distinctive geometric features.

7.3 Manual 6D pose annotation

An alternative approach to obtain the initial pose of the assembly parts is by employing a 6D annotator. This method involves manually aligning the models of the assembly parts with their corresponding parts in the scene. The result with the initial pose of the cap, body, and bottom in the initial frame through this operation is illustrated in Figure 54. For example, in the right part of the figure, the point cloud of the model of the cap is represented by the white dots and is manually aligned with the cap in the scene.

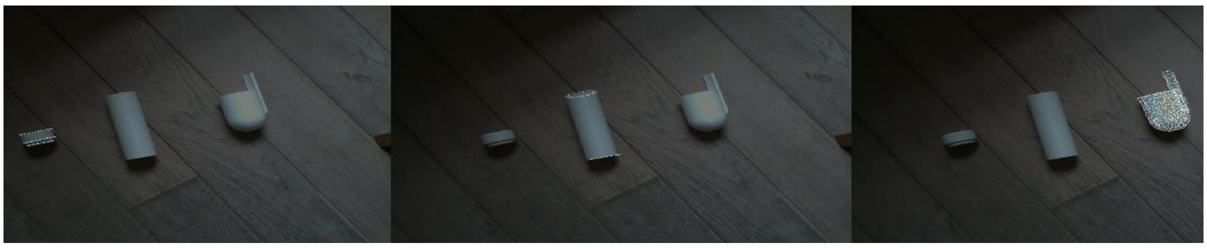


Figure 54: 6D annotator applied to extract the initial pose for the bottom (left), body (middle), and cap (right)

8 Conclusion and future outlook

This master's thesis develops a model that is able to recognise operator activities in the assembly of the Bourjault ballpoint pen through probabilistic modelling and object affordances, in which each part of the ballpoint pen is represented by ArUco markers.

Starting from the literature study, in which first human activity recognition and its components, data acquisition, pre-processing, feature extraction, and classification are explained, different probabilistic graphical models and probabilistic logic languages are compared. Additionally, multiple object detection and tracking methods are discussed and compared.

After an unsuccessful attempt at implementing two probabilistic logic languages, Distributional Clauses and ProbLog, to create a model that can be used to recognise operator activities in an assembly case, the transition was made to dynamic Bayesian networks, one of the probabilistic graphical models discussed in the literature study, through the pgmpy library.

The first step involved constructing a DBN model to predict collisions between two ArUco markers. Through several iterations, the finalised model incorporated factors such as relative distance, orientation, velocity, and motion direction of the objects, providing the probability of collision as output. Initially, the conditional probability distribution of the model was manually computed with broad variable discretisation, resulting in a general indication of collision likelihood but inaccurate probabilities. In a second attempt, the CPD was automatically computed using training data, enabling finer variable discretisation and improved results. In this case, the model, trained by five movement sequences and validated by an additional movement sequence, yielded an accuracy of 68 per cent.

Encouraged by the promising collision case results, the focus shifted to the assembly case, aiming to recognise activities performed during assembly. Initially, a DBN model was developed for the assembly of a simplified Bourjault ballpoint pen, consisting of three parts: the bottom, body, and cap. Inspired by the collision case, the model considered relative distance, velocity, and motion direction between the manipulated main part and other parts. To compute the CPD of the model, 30 assembly sequences were used a training data. Five additional assembly sequences were used to validate the model. Through these five sequences, the model achieved an accuracy of 73 per cent.

Finally, the model was extended to encompass the complete assembly of the Bourjault ballpoint pen, comprising six parts. In this scenario, the input variables were the relative parameters between the main part and its relevant counterparts instead of all other parts. The model's performance was evaluated through a 7-fold cross-validation, utilising 30 assembly sequences for training and five sequences for testing. The final model achieved an accuracy of 75 per cent, a precision of 76 per cent, a recall of 75 per cent, and an F1-score of 74 per cent. Moreover, when the final model trained on all 35 assembly sequences was applied to recognise operator activity in an additional assembly sequence, it achieved an accuracy of 77 per cent.

For future research, the model can be expanded to utilise the six-dimensional pose of assembly parts as input data, replacing the reliance on ArUco markers. The pose of the parts can be extracted through algorithms such as PPF matching, which was, unfortunately unsuccessfully, attempted in this thesis. Implementing the six-dimensional pose of Bourjault's ballpoint pen parts would enable the model's application in a realistic assembly setting for a real product.

References

[1] “Contact ACRO - Research group ,” *KU Leuven*, Mar. 07, 2022.

[2] E. Demeester, “Voorstel Masterproefonderwerp IIW.” Apr. 21, 2022. Accessed: Oct. 09, 2023. [Online]. Available: https://iiw.kuleuven.be/onderzoek/acro

[3] ACRO, “Naar intentie-gebaseerde mens-robotsamenwerking in de maakindustrie,” Oct. 09, 2022. https://www.kuleuven.be/onderzoek/portaal/#/projecten/3E170567 (accessed Oct. 09, 2023).

[4] ACRO, “6D-tracking van objectposities.” https://www.kuleuven.be/onderzoek/portaal/#/projecten/3E170567 (accessed Oct. 09, 2023).

[5] Arkite, “Arkite.” https://arkite.com (accessed Oct. 09, 2023).

[6] X. Wang, Y. Dai, L. Gao, and J. Song, “Skeleton-based Action Recognition via Adaptive Cross-Form Learning,” Jun. 2022, [Online]. Available: http://arxiv.org/abs/2206.15085

[7] B. Moldovan, “Relational Affordances and their Applications,” 2015.

[8] A. Roitberg, N. Somani, A. Perzylo, M. Rickert, and A. Knoll, “Multimodal human activity recognition for industrial manufacturing processes in robotic workcells,” in *ICMI 2015 - Proceedings of the 2015 ACM International Conference on Multimodal Interaction*, Association for Computing Machinery, Inc, Nov. 2015, pp. 259–266. doi: 10.1145/2818346.2820738.

[9] OpenCV, “Detection of ArUco markers.” https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html (accessed May 24, 2023).

[10] G. De Leonardis *et al.*, “Human Activity Recognition by Wearable Sensors : Comparison of different classifiers for real-time applications,” in *MeMeA 2018 - 2018 IEEE International Symposium on Medical Measurements and Applications, Proceedings*, Institute of Electrical and Electronics Engineers Inc., Aug. 2018. doi: 10.1109/MeMeA.2018.8438750.

[11] J. de Lope and M. Graña, “Deep transfer learning-based gaze tracking for behavioral activity recognition,” *Neurocomputing*, vol. 500, pp. 518–527, Aug. 2022, doi: 10.1016/j.neucom.2021.06.100.

[12] “Gest,” *apotact labs*, 2022.

[13] M. J. Ibrahim, J. Kainat, H. Alsalman, S. S. Ullah, S. Al-Hadhrami, and S. Hussain, “An Effective Approach for Human Activity Classification Using Feature Fusion and Machine Learning Methods,” *Appl Bionics Biomech*, vol. 2022, 2022, doi: 10.1155/2022/7931729.

[14] J. Hadfield, P. Koutras, N. Efthymiou, G. Potamianos, C. S. Tzafestas, and P. Maragos, *Object Assembly Guidance in Child-Robot Interaction using RGB-D based 3D Tracking*. 2018. doi: 10.0/Linux-x86_64.

[15] M. G. A. Komang, M. N. Surya, and A. N. Ratna, "Human activity recognition using skeleton data and support vector machine," in *Journal of Physics: Conference Series*, Institute of Physics Publishing, May 2019. doi: 10.1088/1742-6596/1192/1/012044.

[16] E. Buah, L. Linnanen, H. Wu, and M. A. Kesse, "Can artificial intelligence assist project developers in long-term management of energy projects? The case of CO2 capture and storage," *Energies (Basel)*, vol. 13, no. 23, Dec. 2020, doi: 10.3390/en13236259.

[17] M. G. A. Komang, M. N. Surya, and A. N. Ratna, "Human activity recognition using skeleton data and support vector machine," in *Journal of Physics: Conference Series*, Institute of Physics Publishing, May 2019. doi: 10.1088/1742-6596/1192/1/012044.

[18] G. Bhat, "Visual Object Tracking," *Linköping University*.

[19] D. Bolme, Beveridge J, B. Draper, and Y. Lui, *Visual Object Trackingusing AdaptiveCorrelation Filters*.

[20] A. D. Worrall, R. F. Marslin, G. D. Sullivan, and K. D. Baker, "Model-based Tracking."

[21] Y. Wu, "6D-tracking van objectposities," *KULeuven*.

[22] J. Hadfield, P. Koutras, N. Efthymiou, G. Potamianos, C. S. Tzafestas, and P. Maragos, *Object Assembly Guidance in Child-Robot Interaction using RGB-D based 3D Tracking*. 2018. doi: 10.0/Linux-x86_64.

[23] D. M. Blei, "Model-Based Classification," 2012.

[24] S. R. Eddy, "P R I M E R What is a hidden Markov model?," 2004. [Online]. Available: http://www.nature.com/naturebiotechnology

[25] M. Cramer, J. Cramer, K. Kellens, and E. Demeester, "Towards robust intention estimation based on object affordance enabling natural human-robot collaboration in assembly tasks," in *Procedia CIRP*, Elsevier B.V., 2018, pp. 255–260. doi: 10.1016/j.procir.2018.09.069.

[26] C. Cortes, V. Vapnik, and L. Saitta, "Support-Vector Networks," Kluwer Academic Publishers, 1995.

[27] M. G. A. Komang, M. N. Surya, and A. N. Ratna, "Human activity recognition using skeleton data and support vector machine," in *Journal of Physics: Conference Series*, Institute of Physics Publishing, May 2019. doi: 10.1088/1742-6596/1192/1/012044.

[28] P. Cunningham and S. J. Delany, "K-Nearest Neighbour Classifiers," *ACM Computing Surveys*, vol. 54, no. 6. Association for Computing Machinery, Jul. 01, 2021. doi: 10.1145/3459665.

[29] I. A. Bustoni, I. Hidayatulloh, A. M. Ningtyas, A. Purwaningsih, and S. N. Azhari, "Classification methods performance on human activity recognition," in *Journal of Physics: Conference Series*, Institute of Physics Publishing, Feb. 2020. doi: 10.1088/1742-6596/1456/1/012027.

[30] P. Rückert, B. Papenberg, and K. Tracht, "Classification of assembly operations using machine learning algorithms based on visual sensor data," in *Procedia CIRP*, Elsevier B.V., 2020, pp. 110–116. doi: 10.1016/j.procir.2020.05.211.

[31] Arkite, "Arkite - Operator guidance."

[32] S. C. Akkaladevi, M. Plasch, M. Hofmann, and A. Pichler, "Semantic knowledge based reasoning framework for human robot collaboration," 2020.

[33] E. Şahin, M. Çakmak, M. R. Doğar, E. Uğur, and G. Üçoluk, "To afford or not to afford: A new formalization of affordances toward affordance-based robot control," *Adaptive Behavior*, vol. 15, no. 4, pp. 447–472, Dec. 2007, doi: 10.1177/1059712307084689.

[34] A. Chemero, "An Outline of a Theory of Affordances," 2003.

[35] D. Koller and N. Friedman, "Probabilistic Graphical Models: Principles and Techniques."

[36] X.-S. Yang, "Mathematical foundations," in *Introduction to Algorithms for Data Mining and Machine Learning*, Elsevier, 2019, pp. 19–43. doi: 10.1016/b978-0-12-817216-2.00009-0.

[37] F. De la Rosa and A. Gómez, "Robot Motion Imitation Based on Bayes Networks and Learning," *IADIS International Journal on Computer Science and Information Systems*, vol. 6, no. 3, pp. 87–103, 2011, [Online]. Available: https://www.researchgate.net/publication/236013145

[38] M. Beinhofer, J. Müller, and W. Burgard, "Near-optimal landmark selection for mobile robot navigation," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2011, pp. 4744–4749. doi: 10.1109/ICRA.2011.5979871.

[39] R. Neapolitan and X. Jiang, "Decision Analysis Fundamentals," 2007.

[40] J. W. Hwang, Y. S. Lee, and S. B. Cho, "Structure evolution of dynamic Bayesian network for traffic accident detection," in *2011 IEEE Congress of Evolutionary Computation, CEC 2011*, IEEE Computer Society, 2011, pp. 1655–1671. doi: 10.1109/CEC.2011.5949815.

[41] Z. Ghahramani, "Learning Dynamic Bayesian Networks," 1997. [Online]. Available: http:[/www.cs.utoronto.ca/˜zoubin/

[42] O. C. Ibe, "Hidden Markov Models," in *Markov Processes for Stochastic Modeling*, Elsevier, 2013, pp. 417–451. doi: 10.1016/b978-0-12-407795-9.00014-1.

[43] K. P. Murphy, "Dynamic Bayesian Networks *," 2002. [Online]. Available: www.ai.mit.edu/~murphyk

[44] T. Xuan, "Autoregressive Hidden Markov Model with Application in an El Niño Study," 2004.

[45] Z. Ghahramani and M. I. Jordan, "Factorial Hidden Markov Models."

[46] L. De Raedt and A. Kimmig, "Probabilistic (Logic) Programming Concepts."

[47] F. Riguzzi, “Probabilistic Logic Programming Languages.”

[48] D. Nitti, T. De Laet, and L. De Raedt, “Probabilistic logic programming for hybrid relational domains,” *Mach Learn*, vol. 103, no. 3, pp. 407–449, Jun. 2016, doi: 10.1007/s10994-016-5558-8.

[49] B. Gutmann, I. Thon, A. Kimmig, M. Bruynooghe, and L. De Raedt, “The Magic of Logical Inference in Probabilistic Programming.”

[50] L. De Raedt, A. Kimmig, and H. Toivonen, “ProbLog: A Probabilistic Prolog and its Application in Link Discovery.” [Online]. Available: www.ncbi.nlm.nih.gov/Entrez/

[51] L. De Raedt, “ProbLog,” *DTAI Research Group*, 2020.

[52] J. Salvatier, T. Wiecki, and C. Fonnesbeck, “Probabilistic Programming in Python using PyMC,” Jul. 2015, [Online]. Available: http://arxiv.org/abs/1507.08050

[53] TensorFlow, “TensorFlow Probability.” https://www.tensorflow.org/probability (accessed Apr. 29, 2023).

[54] A. Ankan and A. Panda, “pgmpy: Probabilistic Graphical Models using Python,” 2015. [Online]. Available: https://github.com/pgmpy/pgmpy

[55] T. D. Grove, K. D. Baker, and T. N. Tan, “Colour based object tracking,” Institute of Electrical and Electronics Engineers (IEEE), Nov. 2002, pp. 1442–1444. doi: 10.1109/icpr.1998.711975.

[56] N. S. Hashemi, R. B. Aghdam, A. S. B. Ghiasi, and P. Fatemi, “Template Matching Advances and Applications in Image Analysis,” Oct. 2016, [Online]. Available: http://arxiv.org/abs/1610.07231

[57] C. Li and T. Hua, “Human action recognition based on template matching,” in *Procedia Engineering*, 2011, pp. 2824–2830. doi: 10.1016/j.proeng.2011.08.532.

[58] T. Birdal and S. Ilic, “Point Pair Features Based Object Detection and Pose Estimation Revisited.”

[59] “OpenCV: Surface Matching.” https://docs.opencv.org/4.x/d9/d25/group__surface__matching.html (accessed May 22, 2023).

[60] “Pickit - robot vision made easy.” https://www.pickit3d.com/en/ (accessed May 03, 2023).

[61] R. B. Rusu and S. Cousins, “3D is here: Point Cloud Library (PCL),” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2011. doi: 10.1109/ICRA.2011.5980567.

[62] I. Culjak, D. Abram, T. Pribanic, H. Dzapo, and M. Cifrek, “A brief introduction to OpenCV,” 2012.