Lucas Monteiro Fioravanço

Licenciado em Ciências da Engenharia Eletrotécnica e de Computadores

Human-aware Collaborative Manipulation with Reaching Motion Prediction

Dissertação para obtenção do Grau de Mestre em

Engenharia Eletrotécnica e de Computadores

Orientador: José António Barata de Oliveira, Associate Professor, NOVA University of Lisbon

Co-orientador: Francisco Marques, Research Engineer, UNINOVA-CTS

Júri

Presidente: Doutor Luís Filipe Lourenço Bernardo - FCT/UNL
Arguente: Doutora Ana Inês da Silva Oliveira - FCT/UNL
Vogal: Doutor José António Barata de Oliveira - FCT/UNL

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

March, 2021

Human-aware Collaborative Manipulation with Reaching Motion Prediction

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Acknowledgements

I would like express my gratitude to my dissertation supervisor Prof. José Barata for his support and, through is classes, encouraging me to pursue the robotics field. I would also like to acknowledge my co-advisor, Francisco Marques, for his help and complete availability during the development of my dissertation.

Furthermore, I want to thank all my friends and colleagues, in particular to Francisco Matos, Clarisse Feio, David Leonardo, Alina Vasilciuc, Diogo Pereira and Miguel Calado, for their support and friendship and to whom I wish nothing but the best and that they achieve whatever goals they might pursue.

Finally, my family for their support and motivation, in particular to my parents and sister for their motivation and unconditional support during, not only during my academic journey, but during my life, without whom would have been impossible to be writing this dissertation today.

Abstract

This dissertations presents a possible approach to improve human-robot interaction in an industrial collaborative situation, where the human operator and a collaborative industrial robot work within a shared work-space. The approach presented in this dissertation focuses on a situation where part of the assembly process needs to be carried out by a human operator, whose assembly station is located on a work-bench, and a robot is used to pick and place products in specific locations on the operator's work station. Because those locations can be accessed both by the robot or the human operator at any time, collisions can occur and should be avoided in order to make the process more natural for the human operator as well as to avoid the emergency stop of the collaborative robot which has to be restarted and thus decreases productivity.

In order to prevent those collisions the proposed system defines key-areas in each of the locations as well as other relevant positions for the collaborative task. The system uses a Kinect Sensor and a neural network to track the user's hand over time and Gaussian Mixture Models to make predictions regarding the possible destination key-area given the observed trajectory until that moment. If a collision is predicted the robot pauses the task being executed at the moment in order to prevent it and, once the conflict has been resolved, resumes operation.

Resumo

Esta dissertação apresenta uma possível aproximação para melhorar a interação humano-robot em situações industrias colaborativas, onde um operador humano e um robot industrial colaborativo trabalham num espaço partilhado. A aproximação apresentada nesta dissertação foca situações onde parte do processo de produção deve ser realizado por um operador humano cuja área de trabalho se localiza numa mesa. É utilizado um robot de forma a colocar e retirar produtos de locais especificos da mesa de trabalho do operador. Uma vez que estes locais podem ser acedidos pelo utilizador e pelo robot a qualquer momento é possivel que ocorram colisões que devem ser evitadas, de forma a tornar a interação mais natural para o humano e evitar paragens de emergencia, que requerem que o robot colaborativo seja reiniciado manualmente e, portanto, diminuem a produtividade.

De forma a prevenir essas colisões, o sistema proposto define áreas-chave nos locais onde podem ocorrer colisões e em outras localisões relevantes para a tarefa colaborativa a ser executada. A solução proposta utiliza um sensor Kinect, juntamente com uma rede neuronal para seguir a mão do operador ao longo do tempo e usa *Gaussian Mixture Models* para fazer previsões relativas à área de destino dada a trajetoria observada até ao momento. Se for prevista uma colisão o robot interrompe a execução da tarefa programada de forma a evitar a colisão. Uma vez o conflito resolvido, o robot retoma a tarefa do ponto onde parou.

Contents

List of Figures

List of Tables

Acronyms

ATCRF	Anticipatory Temporal Conditional Random Fields.
CNN	Convolutional Neural Network.
CRF	Conditional Random Fields.
DBOW	Dynamic Bag of Words.
GMM	Gaussian Mixture Model.
GMR	Gaussian Mixture Regression.
HMM	Hidden Markov Model.
HSMM	Hidden Semi-Markov Model.
IBOW	Integral Bag of Words.
LSTM	Long Short-Term Memory.
MEI	Motion Energy Image.
MHI	Motion History Image.
PCA	Principal Component Analysis.
ROS	Robot Operating System.
RTG	Random Trajectory Generation.
S-HSMM	Switching Hidden Semi-Markov Model.
SVM	Support Vector Machine.
TP	Teach Pendant.

Chapter 1

Introduction

One fundamental aspect of human interaction is the ability to recognise actions being performed by others. As an observer, a human can predict, not only possible actions that others will perform in real time, but also, given one more or less complex action, decompose it into simpler actions and predict the possible steps the actor is likely to take.

This ability comes innately to humans so, naturally, humans expect others to understand their actions in the same way. As such, in situations where robots need to interact with humans either to navigate among them, in the case of mobile robots, or in collaborative assembly lines robots should be able to *understand* the human motions and/or actions in order to emulate this innate human ability.

However, the meaning of *understanding an action* is somewhat subjective and is dependent on the level of analysis being used. For the purposes of this dissertation, when referring to *motions*, the author means the actual motion in space performed by the actor, regardless of the purpose. On the other hand when referring to the actual meaning, or purpose of those motions, this dissertation will use the word *action*.

Action recognition and prediction alongside with collaborative robotics are very active research topics and contain a set of complex challenges yet to be overcome in order to achieve a *quasi* human level of interaction.

Given the vastness of this research field, the first logical step is to limit the problem to a specific sub-set of situations. Although the knowledge gained by recognising what action is being performed by a human actor could undoubtedly be useful when interacting with humans, one could argue that in specific situations, such as in industrial collaborative assembly lines, where humans and robots work in a shared work-place, the knowledge gained by recognising the action being performed by the user may not be relevant, or even necessary, in order to improve human-robot interaction.

One can describe (although quite bluntly) some assembly line work as repetitive

where the actions being performed are not likely to change drastically, thus diminishing the requirements for high adaptability from a system designed to improve human-robot interaction. Such a system could, therefore, be action agnostic. Focusing on the motions which constitute the action, rather than on the meaning (and/or consequences) of the action itself. This is the approach taken in this dissertation, where only the motions are relevant to the system instead of the action itself, and use that information to make prediction about those motions.

When dealing with human-robot interaction and action recognition many approaches have been proposed over time using different methodologies for the different modules of the system. For example some approaches may opt to aim for action recognition using either deep learning methods, or even shallow approaches to represent the problem. Whilst others, may take action-agnostic approaches to the problem, focusing on the user's motions by themselves, without trying to extract meaning from them.

The data used can also vary as different approaches can use different sensors, such as single or multiple RGB-cameras, depth sensors or a combination of different sensors, each with advantages and disadvantages. For example, RGB-based methods might be more susceptible to background noise and lighting changes, than depth-base methods.

Other problems arise when dealing with sensors such as the ones referred above, namely the different views of the same scene due to the position of the sensor relatively to the actor. Particularly, but not exclusively, for supervised action recognition approaches, where the learn spatial features may not represent the actions correctly when viewed from a different angle from the one used in the data set during training. One possible solution to this problem is the use of skeletal key-points locations, for example [1], where a convolutional neural network is used to detect different body parts. If every key-point is represented successfully then the problem of view variance could be mitigated. However, acquiring the location of skeletal key-points is subject to problems regarding position relative to the actor, as some body parts can be occluded either by the actor's body itself, or by the environment.

Even if one ignores all the problems described so far and assume an unrealistic situation where the acquired data is perfect, there are still two main problems to solve regarding the data, which depending on the nature of the algorithm used for prediction or recognition, may influence on different degrees the performance of the system. The first is inter-class variation, i.e. the different possible actions. This problem is clearly more prominent on action recognition algorithms, whereas in action agnostic approaches the system does not try to classify an action. The second problem, on the other hand, intra-class variation is relevant even in action agnostic systems, as each actor may perform the same type of motion, i.e. the same action, albeit slightly differently, or even the same actor may perform the same motion but not in the exact same way.

There are several possible uses for action recognition and prediction. As pointed out in [2], these methods might be useful for visual surveillance, using action recognition for criminal activity detection, video retrieval, using text based description for videos

Figure 1.1: Examples of the detected joint positions in different scenarios and with different number of people, originally showed in [1].

might return unreliable results for search engines for example, simply because those descriptions might be wrong, features using action recognition might wield better results in this regard. Entertainment is another possible use for this technology, in fact already in use today in motion-controlled video games using Microsoft's Kinect. [2] also refers the importance this technology has in two other relevant areas, human-robot interaction and self-driving cars, where *understanding* human actions and intentions is key for, not only effective operation, but also for safe operations of such systems.

1.1 Goals for the Proposed Approach

The proposed approach on this dissertation focuses on a situation where a human operator and robot need to work on a shared place, in particular the human operator is working on a workbench and the robot can access certain areas of that workbench, where the possibility of contact between the robot and human may happen. This is, obviously, undesirable and should be avoided. In order to avoid this situation and make the interaction more efficient and natural for the human, this system will allow the user to perform the trajectories before the robot is running between different areas, and once the robot starts performing its task the system makes predictions about the destination of the current motion. This can then be used to adapt the behaviour of the robot to the predicted human motion. In this case the robot stops if a collision is predicted, and once the conflict is resolved the robot resumes the task.

In order to acquire the data the system will use the kinect sensor to acquire RGB and

depth data, the anatomical key-points will be detected using a neural network model, and Gaussian Mixture Models will be used to make destination predictions. Finally, this dissertation will also integrate the predictions with a real industrial robot using the moveit framework.

1.2 Dissertation Outline

This dissertation is divided into five chapters, including this one, where the author presents some relevant work already done in this field, presents some essential concepts in order to fully understand the proposed approach. Later on this document describes the proposed approach and finally the results obtained from with the proposed approach and conclusion of the dissertation.

Chapter 3 presents the work already done in the field of action recognition and prediction, showing some of the different approaches taken by the respective authors. From deep neural network approaches to shallow representations for both action recognition and prediction and action specific and action agnostic approaches.

Chapter 2 introduces relevant concepts, which, although not the main focus of this dissertation, play important roles in the proposed approach. This chapter gives a brief overview on fully connected and convolutional neural networks and the concept of convolution. Furthermore this chapter also provides a brief presentation of the inner workings of the method of structured light used by the Kinect sensor in order to acquire depth information and about Gaussian Mixture Models. This chapter does not intend to provide an in-depth explanation for each of these concepts, instead it aims to provide the reader with an introduction to them in order to allow a better understanding of the proposed method.

On chapter 4 is described the proposed approach on this dissertation. There, the author provides an explanation inner workings of the approach, first in a general sense explaining the architecture of the system, as well as the data necessary to use this system. Later on a detailed description of the inner workings of each software module is provided to the reader, as well as the hardware components used, namely the data acquisition sensor and the robot.

After the description of the proposed approach, chapter 5 presents the tests and respective results performed with the system. This chapter intends to present the results in a cumulative fashion, by performing tests with each component added gradually, until the system is tested as a whole.

Finally, chapter 6 is the conclusion of this dissertation. This chapter presents the final considerations regarding the performance of the system, as well as downsides associated with the chosen approach and possible future improvements that could be made to the system.

CHAPTER 2

Supporting Concepts

The system proposed in this dissertation uses several different modules that accomplish different goals using different techniques which, although not the main topic of this dissertation, play a crucial role in the proposed approach. Therefore this chapter aims at providing the reader with an overview of some of the techniques and algorithms used in this system, namely Convolutional Neural Networks (CNN), Gaussian Mixture Models (GMM) and the structured light process as depth information acquisition method.

2.1 Convolutional Neural Networks

The building block of every neural network is the artificial neuron. Based on the inner workings of the biological neuron, the artificial neuron is a mathematical model that intends to simulate the neuron's biological behaviour through a numerical method. The artificial neuron has a set of inputs and a set of weights. Each input if multiplied by it's correspondent weight and all these values are added together. Finally this result is fed into an activation function which output is the actual neuron output (O). The artificial neuron can be modelled as follows:

$$O = \phi(\sum_{n=0}^{N} w_n x_n) \tag{2.1}$$

Where x_n is the nth input, w_n is it's correspondent weight and ϕ is the activation function. This structure is able to encode hidden features present in data, to a certain extent given the simplicity of a single neuron, through the adjustments of the weights based on the error between the output of the neuron and the ground truth.

A single neuron is a fairly simple algorithm with a rather limited learning capacity on it's own. It is possible, however, to group several of these single neurons in different layers in such a way that the outputs of the previous layers are the inputs of the next

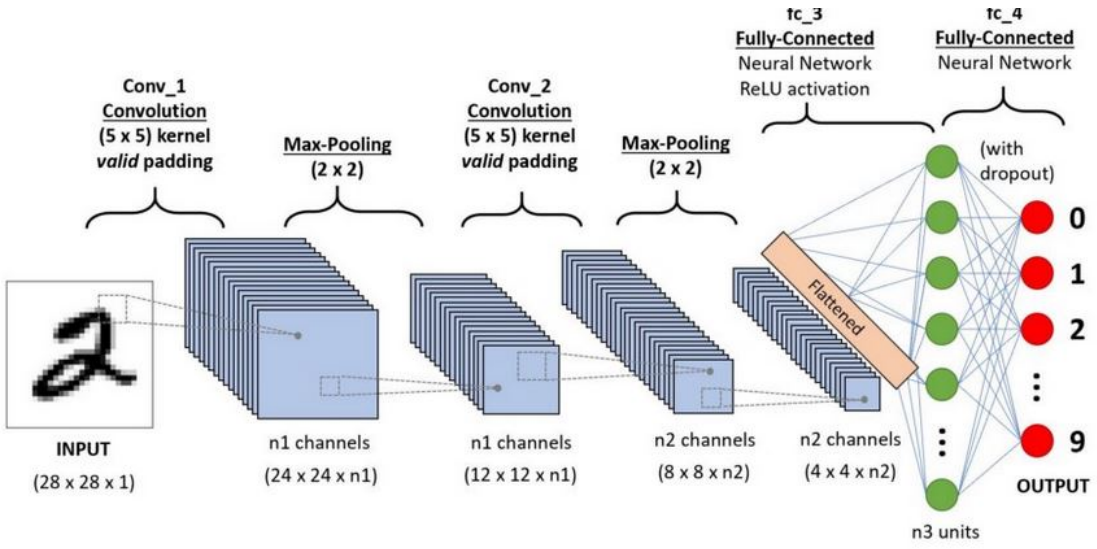


Figure 2.1: Example Arquitecture of a Convolutional Neural Network for hand-written digits recognition. Source: [3]

layer, fully connected or otherwise, with different activation functions, linear or not, which allow the network to encode different non-linear features within it's weights. The adjustment of weights is also made from the error between the ground truth and the output of the network, however that error can only be computed for the output neurons because a ground truth for every single neuron is not available, instead a method called back propagation is used to propagate the error from the output layer through the entire network.

Another important landmark in machine learning was the development of Convolutional Neural Networks (CNN), in particular, but not limited, to image classification. This algorithm makes use of two main operations, convolution and pooling. In the case of images, a convolution is a mathematical operation which can be defined as:

$$h * g[x,y] = \sum_{j=-a}^{a} \sum_{i=-b}^{b} h[i,j]g[x+i,y+j] \tag{2.2}$$

Where g is an image of dimensions $W \times H$, h is the filter being applied to the image of size $W_f \times H_f$ and $h * g[x,y]$ is the result of the convolution at (x,y), $a = floor(H_f/2)$ and $b = floor(H_f/2)$. This operations is widely use in image processing with fixed values for the kernel weights. The same principle is used in CNN's, however the weights in the kernel are the actual trainable parameters. The resulting "images" are called feature maps and during the training process the kernel will adjust the weights in order to acquire the relevant features for the problem at hand.

The pooling operation is responsible for reducing the dimensions of the feature maps by combining clusters of spatial features. An example Convolutional Neural Network is represented in Figure 2.1, where one can observe the sequential convolutional layers,

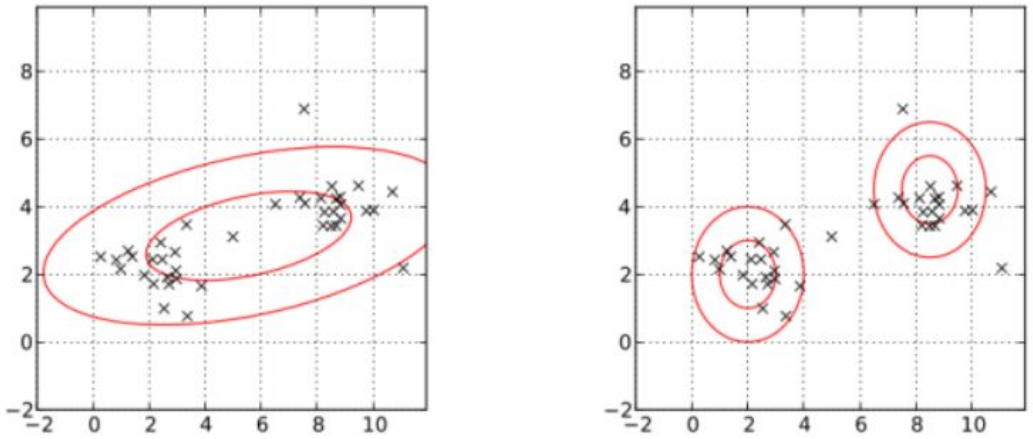


Figure 2.2: To the left is an example of using a single Normal distribution to model a multi-modal data set. On the right is the same data being modelled using a two component Gaussian mixture model. Adapted from [4].

followed by pooling operations, in this case max-pooling, to reduce the feature maps dimension. The network ends with fully connected layers to output the classification result.

2.2 Gaussian Mixture Model

The Gaussian or Normal distribution is one of the most used probability density functions to describe real world data. However, this distribution fails to accurately represent multi-modal data. In this case a better fit to the data might be achieved by using several of these Normal distributions, where each distribution represents part of the data set, using the best fitting parameters for each part. An example of this phenomenon can be observed in Figure 2.2, where a multi-modal data set is modelled with a single Normal distribution (left), obtaining a relatively "loose" fit to the data. Whereas the use of multiple distributions, in this case in particular, two distributions (right), results in a much better fit to the data.

A multi-dimensional Gaussian Mixture Model is defined as:

$$p(X) = \sum_{i=1}^{N} \phi_i \mathcal{N}(X|\mu_i, \Sigma_i) \tag{2.3}$$

$$\mathcal{N}(X|\mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^N |\Sigma|}} \exp\left(-\frac{1}{2}(X-\mu)^T \Sigma^{-1}(X-\mu)\right) \tag{2.4}$$

Where μ and Σ are the means and covariance matrices, respectively, ϕ_i are the weights attributed to each of the components ($\sum_{i=1}^{N} \phi_i = 1$) and N is the number of components. The Gaussian Mixture Model can use an iterative algorithm (Expectation-Maximisation) in order to fit the means, covariances and weights of the different Gaussian components to the data set.

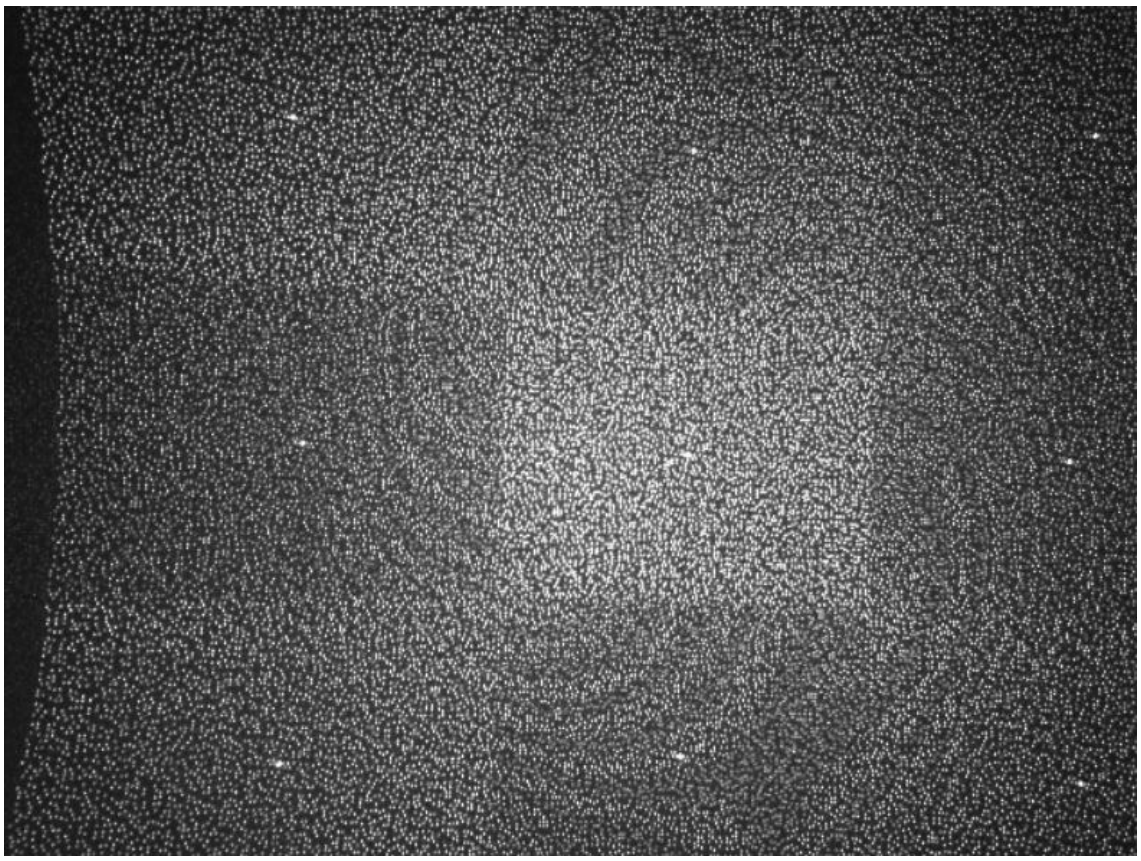


Figure 2.3: Pattern projected by the Kinect sensor for the structured light method. Source: [5].

2.3 Structured Light

Currently exist several depth data acquisition methods, purely with RGB data generated by stereo cameras, or using other techniques such as by projecting a known pattern or by computing the distance based on time of flight. The proposed approach presented in this dissertation uses a Kinect sensor as a source of depth data. Specifically, the proposed approach uses the first version of the Kinect sensor, which uses the structured light as a depth data acquisition method. However a more recent verion of the sensor (Kinect ONE) uses time-of-flight technology to acquire depth data.

The structured light method uses the projection of a known pattern to determine the depth information of a particular scene. According to [5] in the case of Kinect sensor the IR projector paints the scene with a pattern of bright and dark spots. This pattern is saved at a known distance during calibration and by comparing regions of the new image with the images from the calibrated pattern it is possible to determine the distance to the object.

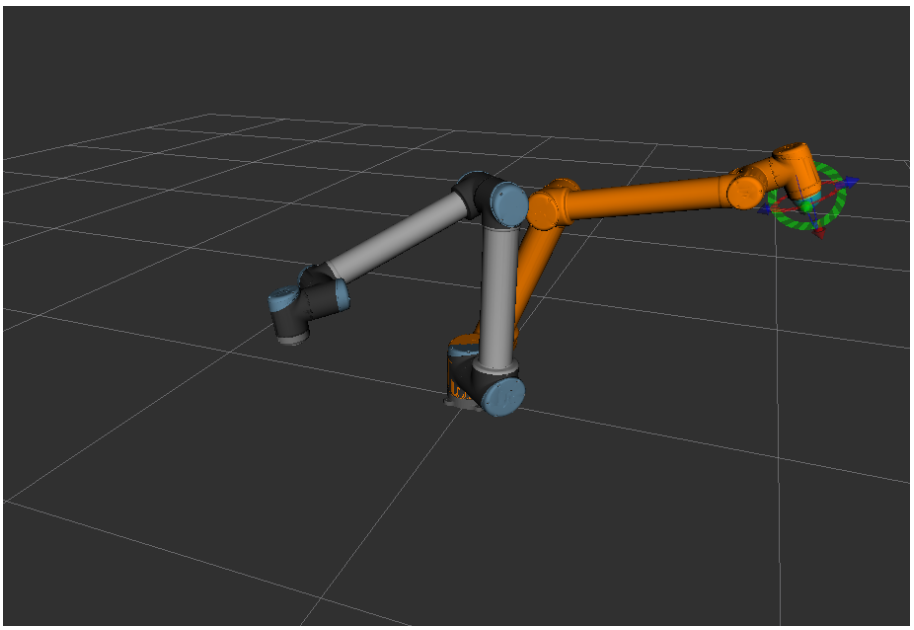


Figure 2.4: Rviz representation of a UR-10 robot in its current state, as well as the current target pose (orange).

2.4 Robot Operating System

The Robot Operating System (ROS) is an open-source middle-ware built for robotics applications. ROS provides a various tools and libraries useful for robotic software development, such as libraries for perception, localisation and mapping, navigation, coordinate frame representation and simulation frameworks. Among other tools, ROS also provides software to visualise several components of the system being developed. One of the software that allows this type of functionality is Rviz, and an example of the images generated by Rviz can be seen in Figure 2.4, where an UR-10 robot is represented in its current state and, in orange, the current target pose.

The Robot Operating System allows the software development to be modular, by providing developers with several inter-process communication mechanisms. Each of these processes are referred to as *Nodes* and ROS provides interface mechanisms between them. There are two main communication methods used in the ROS environment: *Services* and *Topics*.

A Service represents a functionality that a node (server) can advertise to other nodes and usually receives input data and outputs the result of the operation performed by the implemented service logic. A service can be logically compared to a function call where a set of input parameters is passed, the function is executed and the output data is returned to the caller. In the context of a Service, the caller is referred to as a client, which awaits the service response once it has been called, suspending the thread of execution where the service call occurred.

A Topic, on the other hand, is a way of sending messages between nodes. Similarly

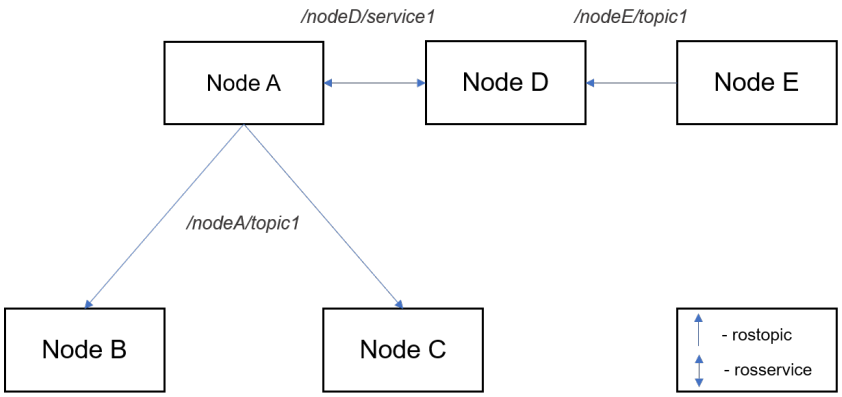


Figure 2.5: Example use of ROS Services and Topics between different ROS nodes.

Nodes	Relation with Topics and/or Services
A	Service Client */nodeD/service1* Advertise *nodeA/topic1*
B and C	Subscribe *nodeA/topic1*
D	Service Server */nodeD/service1* Subscribe */nodeE/topic1*
E	Advertise */nodeE/topic1*

Table 2.1: Relation between the different nodes represented in Figure 2.5 and each Service or Topic.

to the service, one node can advertise a topic (Publisher) and any other node can receive the messages sent by the Publisher and are referred to as Subscribers. Every subscriber that subscribes to a specific topic implements a callback function to handle the received message. An example of the use of these communication mechanisms (ROS Services and Topics) is depicted in Figure 2.5, where Nodes A through E are individual ROS nodes and communicate with each other using Topics and Services. The action taken by each node regarding the communication mechanisms are listed in Table 2.1.

The data transmitted in Services and Topics can be standard messages included with ROS, or the messages can be custom made to include the relevant information. This mechanism creates an abstraction layer from the implementation details of each node by allowing communication between two or more nodes without requiring any special attention to integrating different technologies from the developer. Every Topic and Service are identified by a string such as */node_name/topic1*. Furthermore, the implementation of these mechanisms allows for communication between nodes even if they are running on different machines on the same network.

The final relevant aspect, regarding process communication, for this dissertation is the implementation of *Actions*. Services already provide a way for nodes to perform tasks. However, some tasks might take a long period of time to complete and the developed may need the ability to cancel the execution of said task. In the ROS environment, one can use an action server to perform a long lasting task, by passing to the action server a goal and

the action sever will handle the execution of that goal without halting the caller thread. Action servers also provide relevant data regarding the task execution and different functionalities that may be useful to the developer, the ability to cancel the pursuit of a goal is among those functionalities. These feedback data and functionalities are implemented using the communication methods presented above, Topics and Services.

Furthermore, ROS also provides tools to record the messages published on a set of topics chosen by the developer and save them to a file. That file can then be accessed and replay the messages for the recorded topics. This files are referred to as ROS bags and beside replaying the recorded information this files can also be analysed with some tools included within ROS, in order to determine what topics were recorded, the type of each message as well as the total number of messages published in each topic.

2.5 MoveIt

MoveIt is a ROS framework for manipulation using different types of robots. This framework provides developers with solutions for motion planning, inverse kinematics, control, integration with 3D perception and collision checking.

In particular for this dissertation, the main functionalities of this node are control and motion planning. The control part is specific to the low-level control aspect of the motion execution, which allows the framework to make the desired trajectories at the speeds and time given by the developer. The motion planning aspect, on the other hand, includes the creation of the motion that should be executed in order to achieve a target pose. With the inverse kinematics, one could also set a target position for the end-effector of the robot and the framework would compute the pose that the robot should adopt so that the end-effector is on the set destination. In Figure 2.4 the robot is represented in its current position, using the Rviz GUI it is possible to choose the angles for each joint manually in order to set a target pose for the robot, represented in orange in Figure 2.4. The target state is then passed to a planner to compute the necessary trajectory to place the robot in the target state and the computed movement is executed.

MoveIt provides a C++ API to be used with the necessary nodes, which provides a layer of abstraction from the low-level aspect of the robot control. This allows the developer to use high level commands such as setting a target state for every joint of the robot, which is a particular important command for this dissertation. Given the target state, the framework computes the motion plan and then executes it. Furthermore, MoveIt uses the Action mechanism presented in the previous section to execute the motion. This also allows the developer to get feedback from the task execution, such if the task was successful or if it was cancelled and the goal could not be achieved.

2.6 Collaborative Robots

Industrial robots have been used to replace humans in repetitive, menial and dangerous tasks, or even in situations that require precision that humans are simply not able to achieve, or, at least, not in a time frame where the use of a robot would not increase performance. However, humans remain far superior when dealing with situations that require flexibility and the ability to deal with unexpected situations, or problems that require any kind of higher cognitive ability to solve. Traditionally, industrial robots were kept separate from humans, and only authorised people could enter the robot areas after the robot had been shutdown and then restarted after the operator left[6] that specific area, and human workers were limited to the designated "safe" areas. This, however, might represent a sub-optimal solution to the problem, when the qualities of both human workers and their robotic counterparts are needed. The combination of the accuracy, speed and up-time of robots with the brain power of human operators may result in better, more efficient production lines.

According to the International Federation of Robotics (IFR) collaborative industrial robots are a class of robots designed to perform task in collaboration with other workers in an industrial environment which may or may not comply with ISO defined standards to be used in different situations. Robots designed for this purpose many times implement inherently safe designs and use hardware and software features to avoid collisions or minimise risk to human operators working in close proximity.

Chapter 3

Related Work

3.1 Action Recognition vs. Action Prediction

Human action recognition and prediction are important problems within the computer vision community with vasts amounts of research being made in this field over the years. Although these topics appear to be of similar nature, they are fundamentally different: Action recognition uses a video of a complete action and the model outputs a label for that action depicted in the video. Action prediction on the other hand, uses video with an incomplete action and outputs the action which is most likely to be performed by the person.

One possible approach to this problem is to train a model for action recognition, i.e., the model is trained using videos containing complete action, and used to predict actions from incomplete videos. Evidence suggest, as presented by [7], that the performance of such a model, although dependent on the data set, is overall lower than expected.

The work on [7] presents a way to transfer knowledge acquired from full videos for action prediction. The process is divided into two main steps. The first step is the learning phase, where visual features of the full videos are mapped into an embedding space and the result of this operation is used to train a classifier which outputs a label for the input embedded features. The second step is the knowledge transfer to partial videos. Similarly to the strategy described above, the visual features from a partial video is mapped into a embedded features space, then the result is compared to the embedded features of the full video, this process is repeated for several parts of the same full video, and also repeated for all the full videos. This comparisons allow for the computation of the projection parameters, W_E. The classifier used in the first step also has relevant information for action prediction, and through the minimisation of a cross-entropy loss function a second set of projection parameters is computed, W_D. Finally, during the

inference phase both, W_E and W_D, are used to compute the embedded features for the given set of visual features. This work achieved better results than other state-of-the-art methods, particularly for lower observation-ratios, that don't use knowledge transfer methods.

3.2 Action Recognition

Action recognition is the ability of a system to, given a set of images (video frames) portraying a complete action, output a label identifying the action being performed by the human on the images.

Over the years several different approaches for action recognition were developed, varying not only on data type, there is work done using RGB, depth data or a combination of both, but also on feature extraction methods. Some of those techniques rely on more traditional shallow feature representation, while other rely on deep learning models for automatic feature gathering. Although the work done using shallow feature representation methods have shown significant results, as will be discussed later, still rely on expert knowledge to be modelled and don't generalise very well. To solve this problem a lot of research has been done with deep learning models for action recognition.

3.2.1 Approaches Using Deep Learning

Because every action has a spatial component and temporal one, there is relevant information in both of those domains that must be taken into account during training. Spatial features can be captured by standard convolutional layers, however that is not the case for temporal features, therefore a way to gather spatio-temporal features is required. [2], presents three main techniques for temporal modelling. The extension of convolutional neural networks to three dimensions (two spatial and one temporal dimension), multi-stream networks, which use separate convolutional neural networks to encode spatial and temporal features. And a combination of convolutional and recurrent neural networks to encode spatial and temporal featers, respectively.

The concept of 3D convolutional neural networks (CNN) consists of an extension of 2D CNN using 3D convolution. This allows the model to gather features from the time-domain as well as from the spatial one. One of the first methods [8] using this technique directly applied a 3D CNN to a set of contiguous frames in order to classify actions. The input of this network is a set of seven contiguous frames. The first layer consists of five hardwired kernels used to create five information channels, grey, gradient-x, gradient-y, optical flow-x and optical flow-y, each of these kernels is used in every frame. The second layer is a convolutional layer and it is applied to each channel individually. In order to increase the number of feature maps, two distinct kernels are uses, resulting in two sets of twenty three feature maps, next follows a sub-sampling layer to decrease spatial resolution, this layer does not change the number of feature maps. Next follow

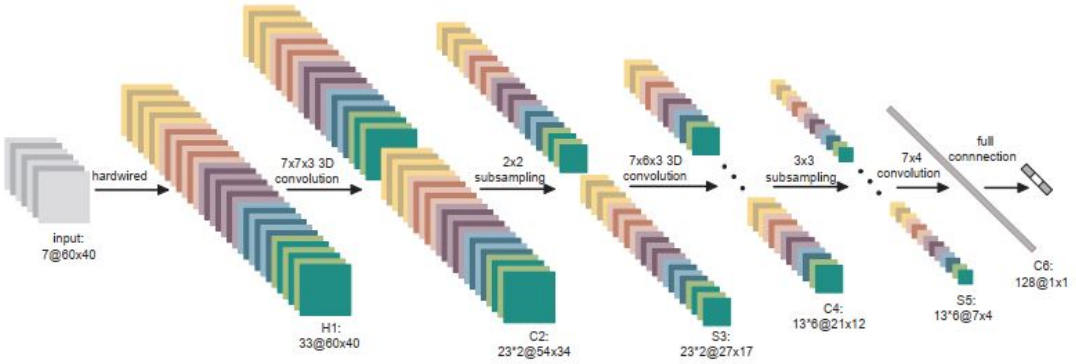


Figure 3.1: 3D Convolutional Neural Network architecture presented in [8].

two more convolution layers with a sub-sampling layer in-between. The result of the last convolutional layer is a 128 feature vector. Each of those feature is fed to a fully connected neuron. The number of output neurons is the same as the number of actions the model should classify. Essentially, the fully interconnected layers of this model act as classifier give the input features gathered from the seven input frames. One thing to note in this work is the fact that the seven input frames weren't consecutive, only every two frames were used as input.

The model proposed by the work presented above is a relatively "shallow" one, with only 3 convolutional layers and 2 pooling layers, other works have extended this approach using "deeper models" such as the presented in [9], referred to as C3D, which has 8 convolutional layers and 5 max-pooling layers, not including the final fully connected layers for feature classification.

The problem with architectures such as the aforementioned C3D is the extensive number of parameters to be trained. The work in [10] presents one method to decrease the number of parameters to be trained, by replacing the 3D convolution kernel used in C3D (3x3x3) by two kernels one for spatial dimensions (3x3x1) and one for the temporal dimension (1x1x3). The results from [10] show that the implemented architecture outperforms the C3D model while resulting in significant smaller models.

The second approach, as described by [2], uses multi-stream networks. The main idea behind this approach is to use multiple convolutional neural networks to model different features. For human action recognition two-stream networks have been used with significant results such as the work in [11]. This work divides videos into two components, spatial and temporal, and uses one convolutional network for each of those components. The spatial network uses individual frames for action recognition, while the temporal network uses optical flow computed from several consecutive frames and both outputs are fused together. In this case, two methods were used, the output average and through an SVM. This work achieved competitive results on the UCF-101 and HMDB-51 data sets and suggests that training a model on optical flow instead of stacked frames

might achieve better results. One of the problmes with this type of approach is the fact that there is no interaction between the information contained in the spatial domain and temporal domain, such interaction encode important information for action recognition since actions happen in both time and space simultaneously.

The third and final approach described in [2] is the combined use of recurrent neural network, specifically LSTMs, and convolutional networks. This type of approach has been explored in [12], where the model LRCN (Long-term Recurrent Convolutional Network) is presented. This model uses convolutional networks as a feature extractor and a LSTM to capture the time dynamics. Every frame that make up an action is fed to the CNN which outputs a fixed-frame feature vector. That vector is then used as the input for the LSTM modules as well as the output from the previous LSTM output. A linear prediction layer is then used to predict the class of that particular frame, this layer outputs a set of classification scores corresponding to the set of possible labels. This process is repeated for every frame of a given action and the final classification scores are given by averaging the classification scores of every individual frame.

3.2.2 Approaches using shallow representations

Although deep learning models have been explored extensively in action recognition, shallow models still show some advantages over deep models. Deep learning methods have a vast number of parameters to be trained, making the training a difficult and computationally heavy process and tends to need large data sets for a good performance to be achieved.

Shallow approaches don't have the ability to generalise, making them susceptible to noise in the data set and intra and inter-class variation, i.e. the slightly different ways people can perform the same action and the, possible, small differences between different actions, respectively. There are two primary concerns when using a shallow method, action representation (feature gathering) and action classification.

Several different approaches have been taken for action representation using shallow features, meaning features that rely on expert's knowledge to develop a method of modelling a given action as opposed to the automatic development of such method as it happens with deep learning models.

One of this approaches was first proposed in [13], where the concept of Motion Energy Images (MEI) and Motion History Image (MHI) were presented. Motion Energy Images, represented in Figure 3.2, are black-and-white images that represent where there is motion in an image. Motion History Images are grey scale images where the value of the image is a function of the time elapsed since the first frame was captured. An example of this type of data can be observed in Figure 3.3.

This paper also states that image differentiation is adequate to compute the MEI for many applications. In the context of a mobile robot such a method might be inadequate if the camera motion is not taken into account and compensated for. Other problems

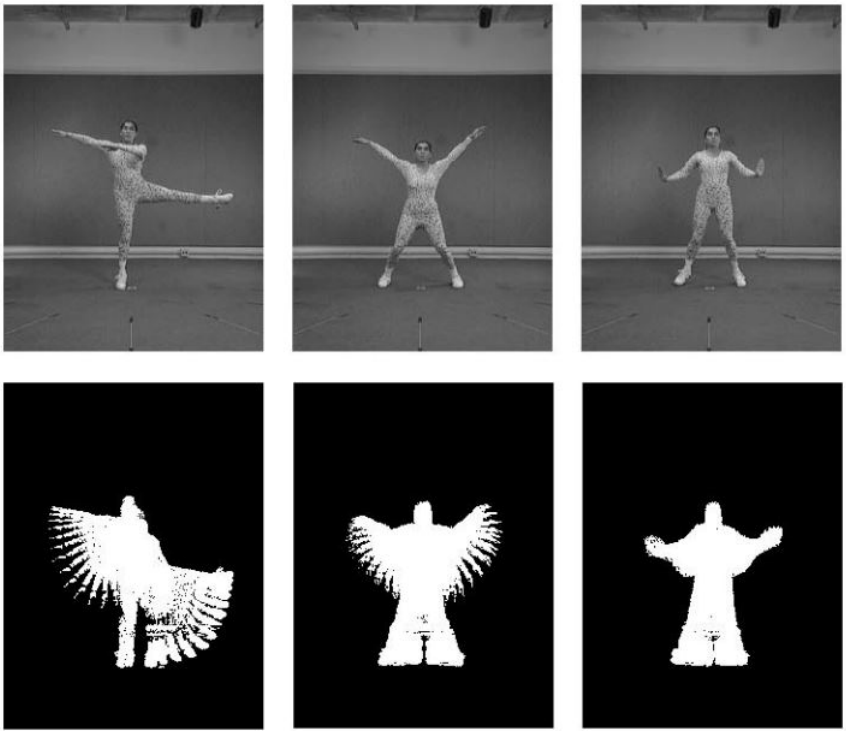


Figure 3.2: Example of Motion Energy Images presented in [13], computed from different action clips.

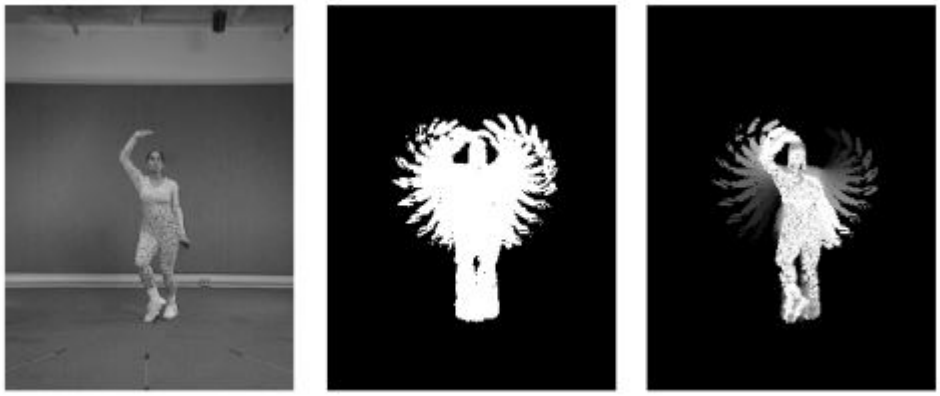


Figure 3.3: One of the examples presented by [13]. To the left a raw frame of the action, in the middle the MEI and to the right the MHI.

identified in this paper are the fact that motion of some body parts might not be specified, which increases the intra-class variation, there might be cases where the whole occlusion of some body parts and the fact that this representation is view specific. The latter problem was solved, or mitigated, by using classification techniques based on image moments which are known for being translation and scale invariant. The classification method in it self is based on minimising the Mahalanobis distance between the input moment description and the known movements.

A generalisation of [13] is presented in [14], extending the motion templates to 3D. According to [14], 3D allows for a natural way of fusing information from multiple images and it is also a more robust way of representing the action regarding the relative camera position. One key difference in this approach is the fact that a Fourier transform is used as a rotation invariant descriptor. It also benefits of being robust to noise. The features gathered are then compared with known actions in order to classify them. Euclidean and Mahalanobis distances were used for classification of the input features.

The methods presented so far follow a holistic philosophy, extracting features from raw video frames. Although these methods appear to encode more information by maintaining spatio-temporal relations between the gathered features, they are susceptible to background noise and partial occlusion. In order to lessen some of these problems extra steps are often taken along side them, such as tracking, segmentation or background subtraction. The need of such operations might render some holistic representation unusable in some situations where high processing power is not available.

Other methods of feature representation have also been broadly studied. One line of thought is the fact that many interesting events in videos are related to strong variations of the data, in both the spatial and temporal dimensions. One of the first methods developed that subscribed to this line of thought is the work presented in [15], where the main idea was to identify regions in images where strong variations occurred in both directions. In [16] this idea was generalised to encompass the temporal dimension as well.

Other methods of creating action descriptors, such as 3D Histograms of Oriented Gradients [17], optical-flow histograms [18], and a combination of both [19], have also been proposed.

The creation of action descriptions has been studied extensively through out the years and numerous methods have been developed with promising results. There is also a lot of research done in action classifiers, according to [2], there are several different types of approaches that can be taken in action classification: direct classification, sequential, space-time, part-based, manifold learning, mid-level features and feature fusion approaches. Although each of these approaches have been heavily explored, because the focus of this works intends to build on the prediction aspect of the problem, only the direct classification and sequential approaches will be explored as these are some of the most active research areas.

Direct classification relies on using already well established classifiers directly on the gathered features. Some of those methods use support vector machines(SVM), k-nearest

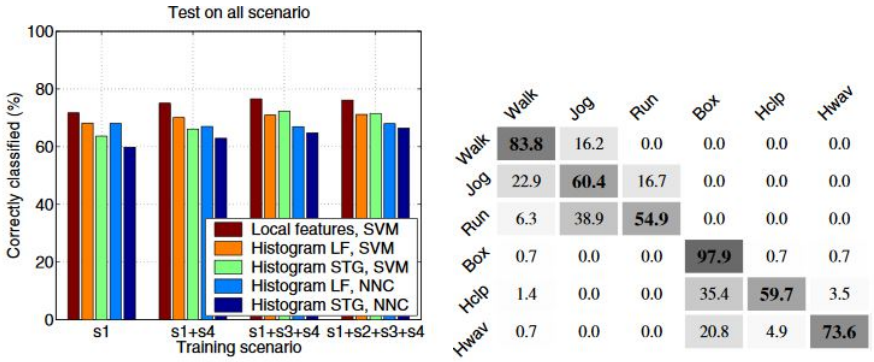


Figure 3.4: To the left the results of testing using the different combinations of features and classifiers across different scenarios. To the right the confusion matrix regarding local features and SVM for all scenarios. Source: [20]

neighbour or the bag-of-words model. In [20] a SVM was used to classify local features computed using the method presented in [16]. A k-means clustering algorithm was also used on top of these feature sets in order to divide them into separate primitive events. As described in [20] the primitive events were then used to build feature histograms of image sequences. This paper also presented different combinations of features (local features, histograms of local features and histograms of spatio-temporal gradients) and classifiers (SVM and Nearest Neighbour). However, the combination that showed the best results is the local features with the SVM classifier.

The data set used for testing contains several different actions, such as walking, jogging, hand waving and clapping, as well as different scenarios, namely, outdoors, outdoors with scale variation, outdoors with different clothes and indoors. The results show that local features with SVM had the best results across all scenarios, although a clear separation between similar actions wasn't achieved, e.g. walk and run, a differentiation between arms and legs was established as pointed out in [20] and can be observed in Figure 3.4.

Another example of direct feature classification is the work in [21]. In this paper features are encoded in two optical flow channels, one for the horizontal and one for the vertical directions, and a silhouette image. A histogram is computed from each of these three channels and are concatenated together, resulting in a 216-dimension frame descriptor. In order to capture motion context a set of 15 frames is used, each set of 5 frames represent the past, present and future. The same local feature capture algorithm is applied to every frame, resulting in a 1080-dimension feature vector for each of these blocks. Each of the resulting 1080-dimension block descriptors are fed into a PCA (Principal Component Analysis) algorithm. The PCA algorithm outputs a set of principal components for each block, the first 50 are kept for representing the current block of frames and the first 10 are used for the past and future blocks of frames. Finally the 70-dimension context descriptor is appended to the current frame to create the final descriptor.

One approach that has received some attention from researchers is the bag-of-words

model. The idea behind bag-of-words is to represent an action with a histogram of sets of features (or "visual words") which can be discovered through clustering algorithms, and the resulting histogram is then used to classify an action with some classification algorithm. One work using this method is the aforementioned [20].

The methods presented so far relative to direct classification don't take into account time directly, that information might be encoded in the features fed to the classifier such as MEI and MHI, however methods using histograms don't have time explicitly encoded in the input given to the classifier. To solve this problem some approaches have been suggested using Hidden Markov models (HMM) and Structured Support Vector Machines (SSVM).

In [22] an approach using a hierarchical Hidden Semi-Markov Model with two layers, referred to as Switching Hidden Semi-Markov Model (S-HSMM), which intends to model actions in different time frames simultaneously. The lower-level HSMM is used to classify atomic actions being performed by the actor using as input the location of the actor in a house and the time spent in each location. Whilst the higher-level HSMM is used to classify the action that is composed by the sequence of atomic actions determined in the bottom layer.

A similar approach was taken in [23]. In the previous work ([22]), the model worked primarily on the trajectories the actor would take to perform a certain action. In this paper however, the model is based on the position of different body parts in the 3D space. Hidden Markov Models were used to model motion of each of those body parts. A second HMM is then used for the higher-level action classification.

In [24] another approach was proposed and showed competitive results, representing actions as set of different *poselets*, the main advantages of these type of approaches, as stated in the paper, is the fact that the model focuses only on relevant information.

The use of depth data and joints position has also been used for action recognition. An example of such an approach is presented in [25], where depth images gathered from a Microsoft Kinect were used to compute the position of 12 joints of the human body in order to create a compact representation of a posture. The relevant joints in this work are left and right elbows, hands, knees, feet and hips, as well as head and hip centre. Due to the computed information regarding joint location it is possible to infer the direction the person is facing, this achieves one of the main goals presented in this [25], making this method view invariant.

The methodology described in [25] intends to encode actions in a histogram of joint positions over time. The joint positions are represented in a spherical coordinate system where α represents the reference vector parallel to the ground and θ is the vector perpendicular to α. In order to compute the histogram space is divided into bins. Each bin is defined by a particular pair of angle intervals, according to α and θ, as is is represented in figure 3.6 The remainder 9 joints are used to compute the histogram, given a joint at (μ_α, μ_θ) will contribute to the histogram according to a Gaussian weight function in order to increase robustness against small variations.

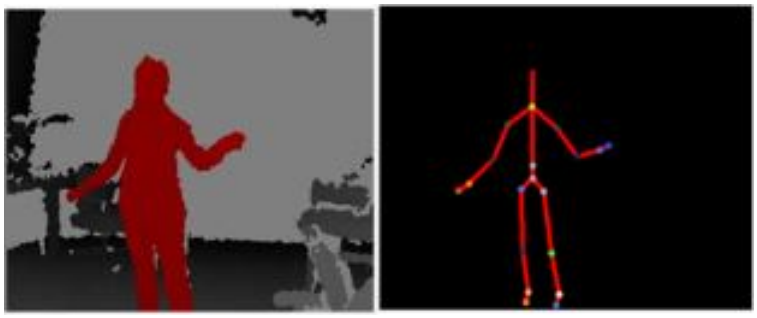


Figure 3.5: To the left is a presentation of a depth image provided by a depth sensor such as the Microsoft Kinect, the red region in the image represents a person. To the right are the joints position computed from the depth image. Originally presented in [25].

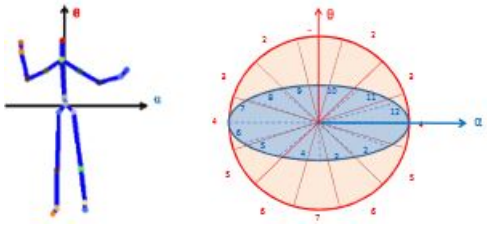


Figure 3.6: To the left a joint representation and to the right the used spherical coordinate system, as stated in the original paper, [25], both α and θ are placed according to the direction the person is facing. Source: [25]

Once the histogram has been computed, Linear Discriminant Analysis is used for feature extraction. Since actions are collections of frames this procedure is applied to each frame for every action video, in turn the resulting features are then clustered using k-means forming visual words. Finally, with this data it is possible to represent actions as time series of visual words. The recognition in itself is performed using a Hidden Markov Model.

3.3 Action Prediction

The approaches presented so far rely on actions being fully represented in a video clip, this is not possible, however, in real time operation. If a system is to be able to understand human action, it should be able to understand the implications of the actor's actions before the action in it self has ended. Therefore the system should be able to recognise what actions a person is likely to take given the motion performed until the moment of the observation. This operation, besides the problems that action recognition already presents, faces other problems caused by the inherent uncertainty of future prediction.

As previously stated, action prediction has several distinct problems. The first problem is classifying an incomplete action whilst being performed. The second problem, on the other hand, is related with intention prediction, which usually entails inferring the

actions a person is likely to take in order to achieve a certain goal, given a current observation. Finally, the third type of problem is related with trajectory prediction, meaning the system should be able to estimate where a person might be some time in the future given the current motion status. All of these problems are also affected by how far into the future the system intends to make predictions. Obviously, the furthest away in the future the system makes predictions the most useful they are. However, at the same time, the more uncertain those predictions are.

In [26] two approaches for early action recognition were proposed, introducing the concepts of Integral Bag of Words (IBoW) and Dynamic Bag of Words (DBoW). The base functionality is the same as the discussed in the previous section, features are gathered according to some method, in this work in particular space-time interest points, those features are then divided into a certain number of clusters which constitute "visual words". In this work the action is represented by a histogram of which each bin is associated with one of the visual words. To compute the histogram, given some video sequence, to each bin is associated the number of features belonging to the corresponding visual word until some time instant. The integral histogram,$H(O)$, of a certain video sequence, O, is defined as:

$$H(O) = [h_1(O), h_2(O), ..., h_N(O)] \tag{3.1}$$

Where h_i represents the feature histogram for the i-th frame of the video sequence O. This process is repeated for all the complete action clips, each class is represented by the average integral histogram of all action clips of that particular class. The prediction is represented by

$$Action = \arg\max_p \sum_d P(A_p, d|O, t) \tag{3.2}$$

The main problem with the first approach is the fact that it does not take into account the temporal localisation of features, which is, for obvious reasons, one fundamental aspect of action recognition and prediction. To solve this problem in this paper a second approach is also presented, the dynamic bag of words. This model intends to divide both the action model and the observed video into different length segments and find similarities between segments of the activity model and the observed video.

One of the most significant problems in action prediction is the fact that a label should be output with a relatively low number of observed frames, the reasons for this are quite clear, since the feature that best represent actions are found in the middle of the action instead of the beginning of the action. To solve this problem, architectures using LSTM modules were proposed such as the work presented in [27]. Which initially uses two channels for feature gathering, one being the raw frames and the second optical flow frames. Convolutional neural networks are used in each frame, RGB and optical flow, in order to produce a feature vector. Those features are then used as input to LSTM layers responsible for creating temporal relations between features. Finally, the resulting feature from the RGB and Optical Flow frames are concatenated.

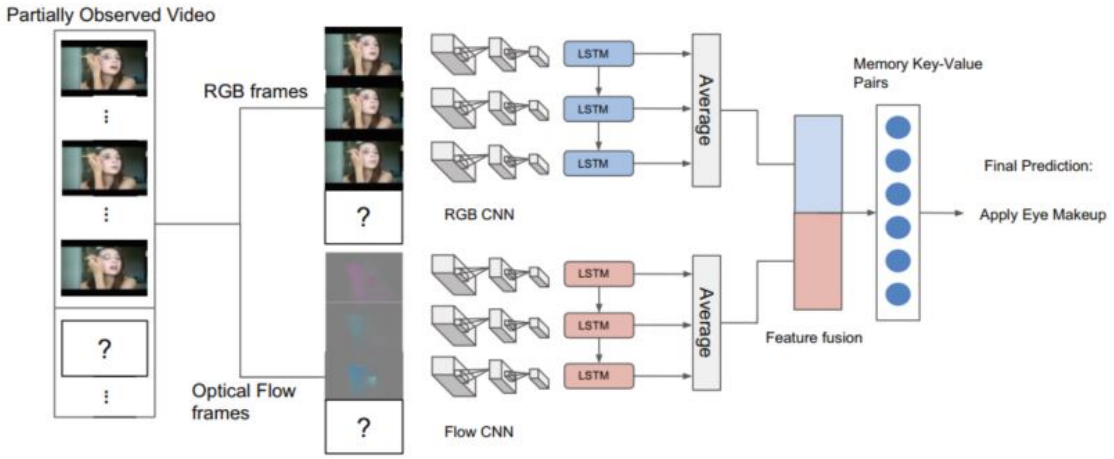


Figure 3.7: The proposed system architecture presented as presented in [27].

Besides the model architecture, one of the main distinguishing features of this work is the memory module used to memorise hard-to-predict sample. The memory follows a key-value paradigm where keys are the features computed by the neural network and the value is both the action label and the progress level. During training, different length observations of the same action video are used as input. During operation, the frames are used as input for the neural network and it outputs the feature vector, which will be then used to query the memory. The result of the query is the value which corresponds to the most similar value to the query. In the results showed in [27], this method achieved better results then some of the aforementioned approaches, C3D (with SVM) and both integral and dynamic bag-of-words, with tests performed with the UCF-101 and Sports-1M with an observation ratio of 0.1, i.e. 10% observed frames.

A different approach using LSTM modules was presented in [28]. However, this work focuses on activity detection and early detection regarding a longer time frame, that is, instead of trying to predict the action in the near future, it intends to classify an activity that might take several minutes to finish, as early as possible. The main concern of this work is not the model architecture in itself, rather, the main contribution rests in a different method of training recurrent neural networks, in particular a way to compute classification error. The main idea behind this method is the fact that "classic" action classifiers attribute the same error regardless of how much of the action was watched until a particular time step. To solve this problem [28] proposes two solutions. The first solution is based on the fact that the system should output higher prediction scores with the increasing observation ratio of a given action. The second solution follows the same philosophy, but states that the system should become more confident in the difference between correct and incorrect action labels the longer the activity is watched. An example of the detection score over time is represented in Figure 3.8. Similarly to the previous presented work, this method relies on a CNN to extract features from video frames and uses LSTM to establish temporal relations between features.

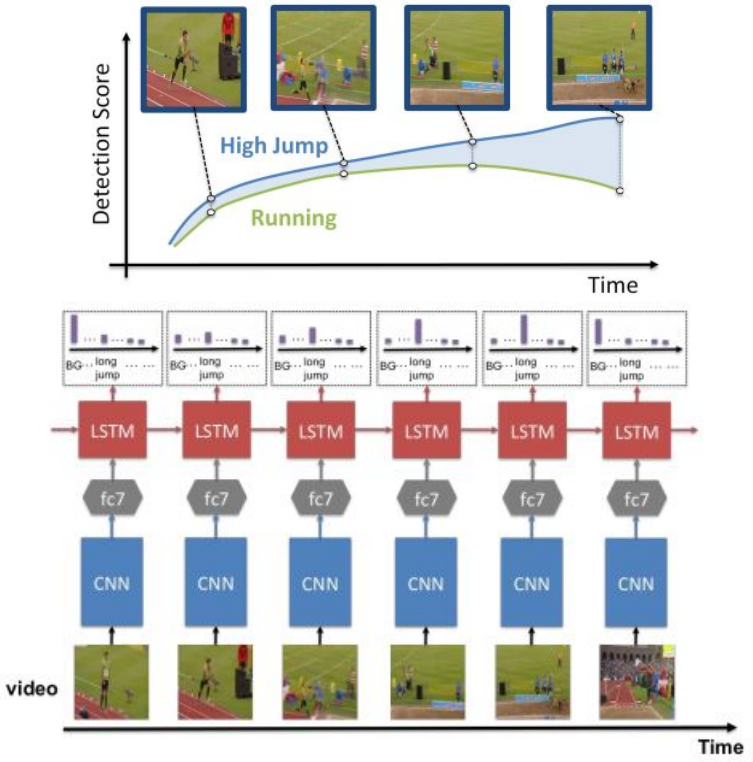


Figure 3.8: On the top is a graphical representation of the main concept of this work. In the bottom is the representation of the used model. Source: [28].

.

In [29] a different idea is presented, stating that the paper's main intuition is that learning to predict the activity progress would implicitly learn features which would allow for temporal localisation and, in turn, that information could be used for activity prediction. One of the differentiating factors in this work is the fact that progression is not thought of as a scalar, rather the duration is discretized into a number of intervals and the model would output the interval to which the observed activity progression corresponded. The other main characteristic of this paper is the proposed loss function.

Since the action progress is divided into intervals it allows one to use standard classification loss functions. However, this leads to every incorrect classification to be penalised in the same way, such a policy might result in sub-optimal training. Therefore, a new loss function was proposed, where the similarity to the ground truth is taken into consideration.

This paper ([29]) uses a two stream residual neural network to find relevant features. One stream is the raw frames, whilst the other is the optical flow. The resulting features are passed on to a LSTM network. One relevant aspect of this architecture is the length of each interval, as stated in the paper, an interval that encompasses the entirety of the action would allow for a continuous progress estimation. However, it would also make the model difficult to train. On the other hand, if the interval is too short there might not be enough relevant motion information. The other problem is the different duration

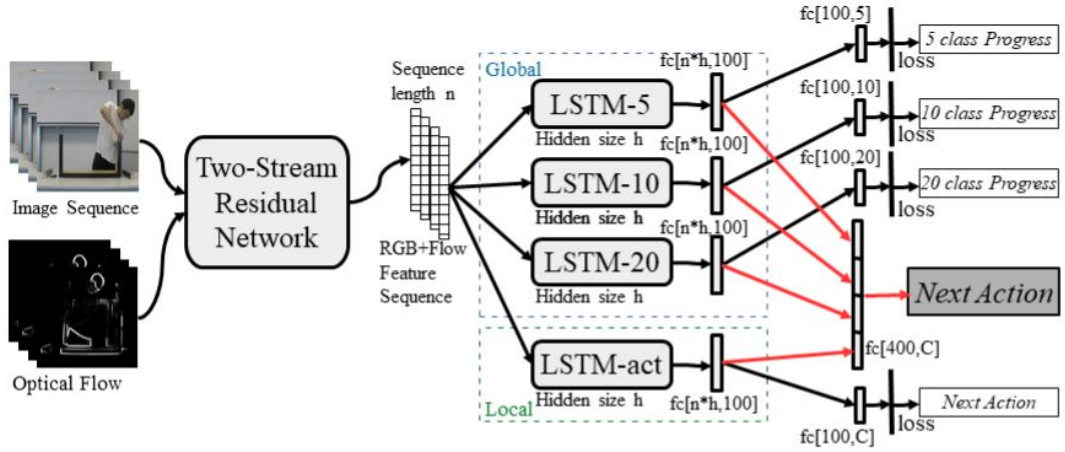


Figure 3.9: The model architecture as presented in [29].

of each action during a given activity. Therefore, to mitigate this problem the solution proposed in this paper is to use three parallel LSTM layers considering the action with different length intervals. Finally, the output of the LSTM layers is passed to a fully connected network to output the predicted progression interval.

Other methods have also been proposed which do not use deep learning models, one of those works is described in [30], where a camera is thought of representing a robot's "first person" perspective, in order to understand action that might affect it. Actions such as *shaking hands* or *throwing* that might be directed at the robot. The core idea of this work is to use subtle moves people might perform to predict the main action that might follow. These small actions are defined as *onset activities* by [30], and are characterised by having a short duration and the fact that do not have a direct effect on the robot, e.g. a person picks up an object and throws it, in this case the *onset activity* would be the person picking the object up, as it is something that happens in a short period of time and does not directly affect the robot.

Given the very subtle nature of the *onset activities*, some of them, since they are very short in duration, might not encode enough motion information to be used directly. Furthermore, these type of actions are stochastic in nature, that is, the *onset activities* may or may not be present before a certain action. As a solution to these problems the paper describes a method to use this information in a more efficient way, called *onset signatures*. This signatures are time series representing the similarity of each video segment and the *onset activities* over time. This information, alongside features extracted from the video segments are used to predict the actor's next action.

The motivation in [31] is to enable robots to anticipate what a human will do next given the current observation. The main difference between this approach and the others presented so far is the fact that it uses other environment information for action prediction, namely objects and its position relative to the actor. This method relies, similarly to the work presented previously also uses probabilistic methods for solving the problem

Figure 3.10: Example of the affordance heatmap presented in [31], to the left the heatmap of the reachibility of an object, i.e. how likely it is for the object to be reached by the actor. To the right is the heatmap of the drinkability of an object, i.e. if an object is in this area is likely to be drinkable.

Figure 3.11: The possible trajectories heat maps and their temporal evolution, as presented in [31].

instead of using deep learning models, specifically, Conditional Random Fields (CRF).

The first step in this approach is to perform temporal segmentation, that is, divide the activities into sub-activities and represent them as a CRF, taking into account the human pose, object affordance (which action can be performed to a given object) and object location. Then a set of future possibilities, with different sub-activities, objects and their spatio-temporal trajectories, this concept is introduced in this paper and is referred to as Anticipatory Temporal Conditional Random Field (ATCRF). Two key aspects of this approach are the aforementioned object's affordance, which evaluates the likelihood of an object to be used for a certain action in space. And the trajectory generation which will allow for the robot to create plans to react accordingly to the human's action, this is done by generating a set of possible trajectories a given object might take to get to the predicted target.

Other approaches make use of Gaussian Mixture Models (GMM) to make prediction about reaching motions. Examples of such work include [32], which proposed an unsupervised online framework for reaching motion prediction. This paper proposes a two-layer system, a reaching motion classification layer and reaching motion prediction layer. The first layer consists of a set of Gaussian Mixture Models, each of which represents a class of human palm reaching motion. The second layer, on the other hand, is composed of a set of libraries of arm motions, and each of those libraries is linked to a specific GMM in the palm reaching motion library. With the observed trajectory or portion of the trajectory the first layer classifies the palm motion and the second layer specifies the type of motion with the arm motion data, finally the system predicts the remainder of the trajectory through regression.

The work presented in [32] uses an algorithm which allows the system to decide whether to add a new GMM to the motion library or use the newly observed trajectory to

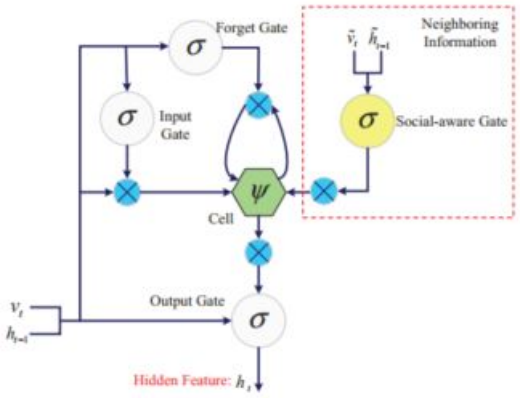


Figure 3.12: The social-aware LSTM module presented in [34]. In the red square is represented the social-aware part of the model where the inputs correspond to neighbour data.

update the parameters of the GMM which classified the motion.

Another approach presented in [33] also uses GMM's to make predictions about the trajectory. This work uses a two-stage process to do recognition and prediction of human motion and use that information to create robot motion plans. The first stage of the process is an offline phase where a Gaussian Mixture Model is fitted to each class of motion performed by a human and then used to extract a motion which that best fits that particular motion class through GMR. This trajectories are then used to compute the occupancy grid for every motion class. During the online phase, the GMM's for every class of motion is queried with the observed trajectory which results in a set of likelihoods. This data, together with the 3D occupancy information is used to calculate the probability of each voxel in the 3D grid being occupied. Finally, and also during this online phase, the system uses the predicted information to create a robot motion plan.

Another area related to action prediction is trajectory of motion forecasting, this field is particularly useful for tasks such as autonomous driving. However, in the case of mobile robots that might operate in spaces alongside humans, the ability to predict the trajectory each person is most likely to take is paramount for the robot to move more fluently and effectively through possible crowded environments. One of the works exploring this field is [34], in which future paths are generated by a deep model. As stated in the paper velocity tends to remain constant or to suffer slight changes over time, unless there are obstacles or significant interactions. The paper argues that this fact makes it easier to make accurate predictions. The trajectory can then be computed by integrating the velocity. The approach presented in [34] relies on LSTM modules, however, as pointed out in the paper, the modules by themselves have a deterministic nature, thus not representing the inherit uncertainties associated with trajectory prediction. To solve this problem [34] presents an approach combining Gaussian Processes, to attribute a probability to the predicted path, forming a Deep Gaussian Process (DGP).

A)

B)

Figure 3.13: Results presented in [34]. A) Results obtained in a structured environment. B) Results presented for unstructured scenes.

Chapter 4

Proposed Approach

This dissertation proposes a method of predicting the target of a specific reaching motion performed by a human operator working in close proximity with a robot. Also in this dissertation an action-agnostic approach was taken, meaning that the action the operator is performing is not the focus of the system, rather the system uses a set of positions acquired over time of the human operator's right. Furthermore, this approach uses a set of well-defined areas which can either be *critical*, where both the robot and operator can reach, but collisions should be avoided. Or *neutral* if only the human operator can enter and no collisions will happen.

The system proposed in this dissertation intends to make predictions regarding the area of destination, whether critical or neutral, of the motion being performed by the human operator, given the observed set of positions using Gaussian Mixture Models. The proposed approach follows the approach taken in [32] with a different data acquisition method, as well as a different system architecture regarding the Gaussian Mixture Models and use the predictions produced by the system to change the behaviour of a collaborative robot. This chapter will describe the overall system architecture, as well as, the individual components, the communication between each component and how the integration with the robot.

4.1 System Overview

This system can be logically divided into two major parts: Data acquisition which encompasses all the logic regarding the perception portion of the system. And the data management part, which is responsible for handling the prediction logic. Finally, the ***Robot Dispatcher*** component is responsible for the integration of the predictions with robot's motion. The architecture of the system, as well as the logical division of the

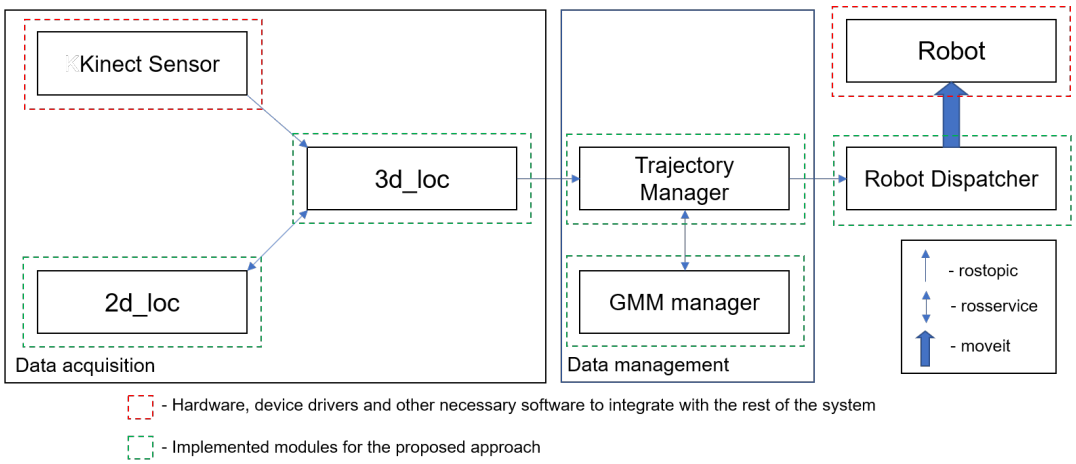


Figure 4.1: Overview of the proposed approach architecture divided into its two logical components: Data acquisition and data management.

components into data acquisition and management is represented in Figure 4.1.

The data acquisition part of the system is composed by the kinect sensor, along with the ***3d_loc*** and ***2d_loc*** modules. Which are used to get the operator's right hand in 3D space. When a new RGB and depth image are received the ***3d_loc*** node sends a ROS service request to ***2d_loc*** to retrieve the joint positions from the RGB image. With that information the system is able to compute the 3D coordinates of the body joints using the depth image, also acquired by the sensor. The resulting information is published in a ROS topic in order to be received by the ***Trajectory Manager*** node.

The data management encompasses the ***Trajectory Manager*** and ***GMM Manager*** nodes. The ***Trajectory Manager*** handles keeps a record of the detected hand positions as well as the locations and dimensions of the critical and neutral areas. With this information the ***Trajectory Manager*** node can detect when a new type of motion has been performed, described by its origin and destination areas and when it should make a prediction.

The ***GMM Manager*** keeps a library of Gaussian Mixture Models where each of them encode a specific type of motion detected by the ***Trajectory Manager*** and provides to services. One to add a new GMM to the library when a new type of motion is detected and another to make a prediction. The decision of which of these services should be called falls to the ***Trajectory Manager*** node.

Finally, the prediction is published into a ROS topic so that the ***Robot Dispatcher*** can integrate that information with the status of the task currently being executed.

Every block in figure 4.1 represents an independent software module that use the ROS framework for inter-process communication. Therefore, each of these processes are referred to as ROS node or, simply, node, hereinafter. The following sections of this chapter will go over the inner working of each module.

4.2 Data Acquisition

The proposed approach uses a Microsoft Kinect sensor (Figure 4.2) for both RGB and depth data acquisition. This sensor is equipped with an RGB camera and an infrared projector and camera, which are used for 3D data acquisition using a structured light method. The sensor is capable of streaming RGB and depth video at 30 frames per second in 640 x 480 and 320 x 240 pixels, respectively, with a field of view of 57 degrees horizontally and 43 degrees vertically. The ROS node freenect_camera [36], which bridges

Figure 4.2: Microsoft Kinect sensor and sensor overview. Adapted from [35].

the libfreenect driver for Linux and the ROS framework, allowing the developer to access the data being streamed by the sensor through ROS topics. The libfreenect driver [37] also solves the registration problem caused by the relative location of both the RGB and depth cameras, publishing in a ROS topic the corrected depth image.

4.2.1 2D data acquisition

As previously stated the acquisition of 2D data is performed by a ROS node, in figure 4.1 is referred to as ***2d_loc***. All the different body parts are detected using an implementation [38] of the OpenPose algorithm [39] [40] proposed by researchers at Carnegie Mellon University. This algorithm utilises a convolutional neural network to infer the locations of the several relevant anatomical points using two concepts referred to, in [39], as Confidence Maps and Part Affinity Fields, the former intends to locate different body parts whilst the latter creates a vector field which encodes the level of association between to given body parts.

In the method described in [39], the image is first analysed by the first 10 layers of the VGG-19[41] neural network, subsequently the algorithm uses a two branch iterative architecture, one of the branches creates the confidence maps while the other creates the part affinity fields, the network's architecture is represented in Figure 4.3. This architecture is said to be iterative because the network can have several similar steps which refine the results of the previous step. The system undergoes supervision after each step as it mitigates the problem of the vanishing gradient, as stated in [39].

In 2019 this algorithm was updated, namely the network architecture, which moves on from the two branch architecture to a simpler single branch, two stage network where the part affinity fields are computed first and the confidence maps are computed based on the part affinity fields in the second stage, as represented in Figure 4.4. Both stages still follow the same iterative nature as in [39]. [40] states that the refinement of the part affinity fields contribute for better confidence maps, the reverse is not true, however. Given this fact, the new algorithm effectively decreases the necessary computations by half in each stage, as stated in [40]. Further contributing for the decrease in the number of operations is the replacement of the 7x7 convolutional layers by three 3x3 layers.

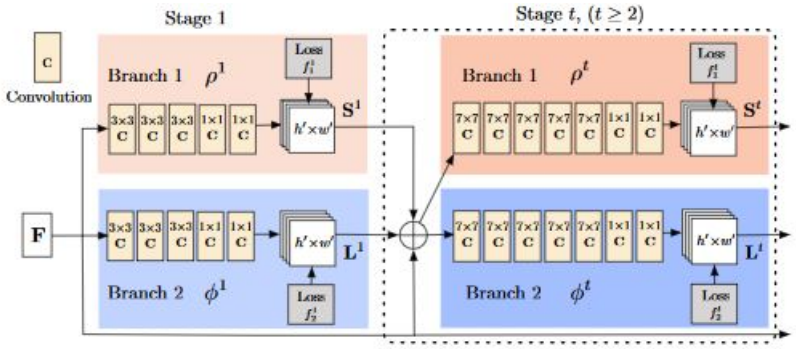


Figure 4.3: Network architecture proposed in [39]. The confidence map branch is represented in beige and in blue the part affinity field branch. Source: [39].

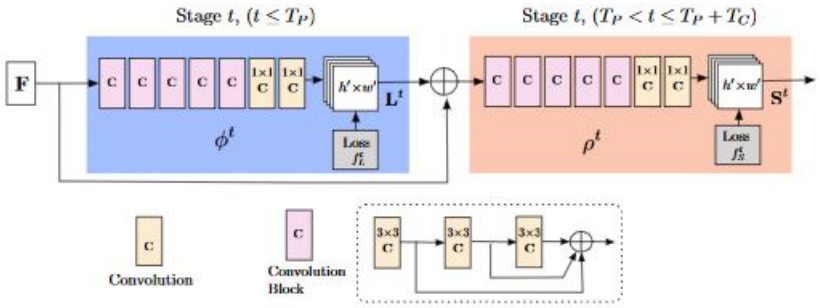


Figure 4.4: Updated network architecture. Source: [40].

This dissertation uses the tensorflow implementation of this algorithm available at [38], which provides an implementation of the network described above, as well as other network architectures referred to as mobilenet_thin, mobilenet_V2_large and mobilenet_V2_small. Each of these algorithms will be further compared in chapter 5.

To maintain the modularity of the system, this node implements a ROS service in order to maintain the rest of the nodes agnostic to the implementation of the algorithm. A node (in this case the ***3d_loc*** node) can make a request to this node with an RGB image and the index of the anatomical key points of interest, and the node responds with the set of coordinates of the requested key points in the RGB image. Before running the inference of the network on the input image, the system resizes the image in order to decrease the

computation time. The coordinates output by the network are normalised by the width and height of the input image, which are then converted back to the pixel coordinates of the original input image. The block diagram of this process if represented in Figure 4.5.

Figure 4.5: Block diagram representation of the 2d_node.

4.2.2 3D data acquisition

By combining the 2D information with the depth information provided by the kinect sensor it is possible to establish a relationship between the position of a given object in the RGB image with the physical location in 3D space relative to the kinect sensor. Because the depth image is correctly registered to the RGB image it is possible to retrieve the depth information for any pixel in the RGB image by accessing the pixel at the same position in the depth image, provided that a valid depth could be acquired for that particular location. Let θ_v and θ_h be the vertical and horizontal fields of view of the kinect sensor, W and H are the width and height of the image, respectively.

Let $\hat{\mathbf{s}}$ be a vector perpendicular to the sensor plane aligned with the centre of the sensor, A the interest point in space and O the origin of the Kinect reference frame, i.e. the centre of the sensor.

$$\vec{\mathbf{OA}} = A - O \tag{4.1}$$

The position relative to the kinect sensor can be determined through the angles formed by $\hat{\mathbf{s}}$ and the projections of $\vec{\mathbf{OA}}$ on xy and xz, according to the defined frame of reference described in figure 4.6. Those angles shall be, respectively, referred to as ϕ_h and ϕ_v henceforth and are represented in Figure 4.7. Those angles can be computed, using the pixel position, (P_x, P_y), of the point of interest in the RGB image and the fields of view, as follows:

$$\phi_h = -\frac{\theta_h}{2} + \hat{\theta}_h P_x \tag{4.2}$$

$$\phi_v = -\frac{\theta_v}{2} + \hat{\theta}_v P_y \tag{4.3}$$

Where $\hat{\theta_h}$ and $\hat{\theta_v}$ represent the angle coverage of each pixel of the image, and is simply computed as:

$$\hat{\theta}_h = \frac{\theta_h}{W} \tag{4.4}$$

$$\hat{\theta}_v = \frac{\theta_v}{H} \tag{4.5}$$

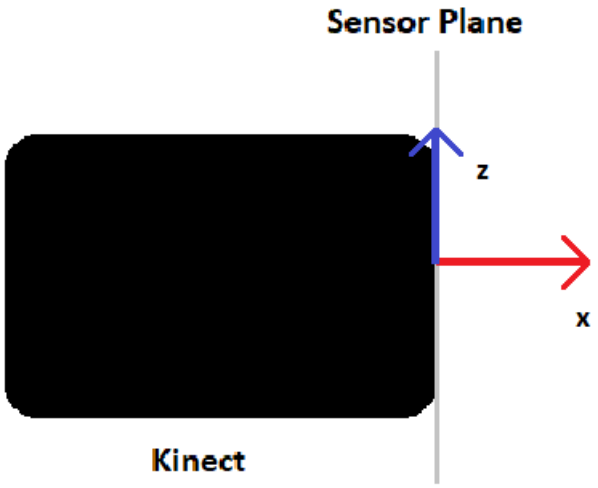


Figure 4.6: Kinect side view and the defined reference frame.

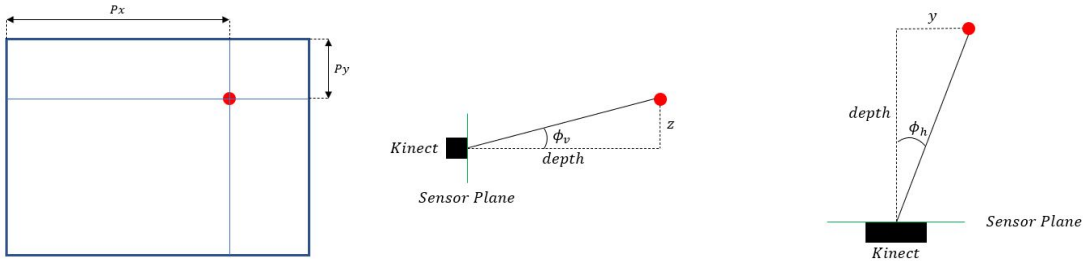


Figure 4.7: Left - Pixel coordinates of the point of interest on the RGB image. Centre and Right - 3D scene representation from side and top views.

Given the depth value, d, for that particular point, which can be accessed in the registered depth image at (P_x, P_y), the coordinates can be computed as follow:

$$x = d \tag{4.6}$$

$$y = d \tan(-\phi_h) \tag{4.7}$$

$$z = d \tan(-\phi_v) \tag{4.8}$$

This algorithm is implemented in the ***3d_loc*** node which subscribes to the rostopics advertised by the freenect_camera node, where the RGB and registered depth images are published. This node uses an approximate time filter in order to synchronise the images, once both the images have been published the system calls the ***2d_loc*** node service in order to locate the relevant anatomical keypoints, in the case of the proposed method, the operator's right wrist. Once the service responds this algorithm computes the position in 3D from the 2D information found by ***2d_loc*** and sends the computed 3D coordinates via a ROS topic. This process is schematically representend in Figure 4.8.

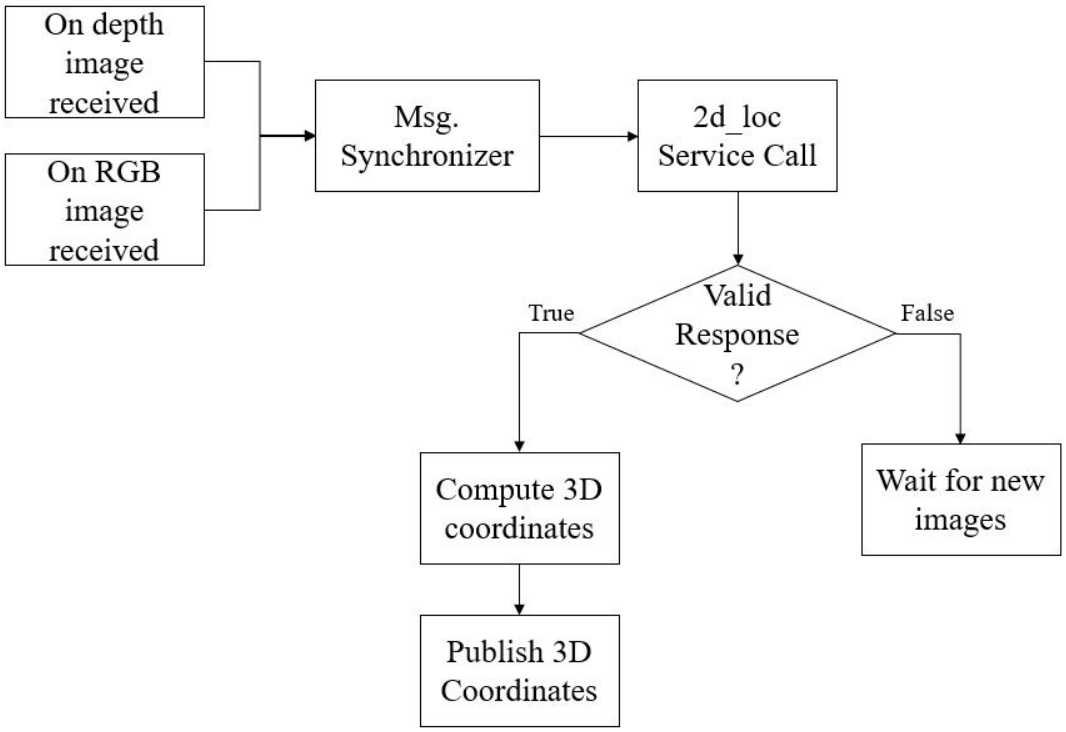


Figure 4.8: Schematic representation of the 3d_node.

4.3 Data Management

4.3.1 GMM Manager

In an assembly situation, particularly in a collaborative situation, the movements will be repetitive with some variation in respect with the physical location and time duration, but essentially the same. This system intends to provide a way to increase efficiency in collaborative assembly situations by allowing the robot to anticipate the operator's moves without the need for a lengthy offline training phase with a new data set. Therefore, this system implements a mechanism similar to the approach presented in [32] where the system learns a library of human motions over time. As mentioned in chapter 3 the system proposed in [32] does not require any offline training and the system can also adapt to changes in the operator's movement.

The approach taken in this dissertation does not focus on the latter, but the ability to learn movements over time without the need of offline training would allow for easier setup in the use of such a system. This dissertation implements a single-layer rather than a two-layer GMM library to make predictions. Furthermore, the proposed system does not make use of regression to predict the future trajectory instead, it uses the GMM's in the motion library to predict the type of motions being performed and thus the destination of the motion. This node implements two services, the first is called when the system detected a new motion and requests this node to create a new GMM given the trajectory.

Alternatively, the system might call the second service, and request a prediction based on the stored trajectory at the moment.

As stated in [32], the creation of GMM require some amount of training data in order to fit the parameters of each component to the data, if the number of trajectories used during training is small or the variance of said trajectories is very low the generality of the GMM will be affected. In order to avoid this problem, or, to a certain extent, mitigate it, [32] proposed an algorithm that allows the creation of a GMM using a single trajectory, which is used in this dissertation proposed approach as well.

Intuitively, the idea behind this algorithm is to create several similar trajectories based on the real trajectory observed by the system and use them to train the GMM. The algorithm is represented in pseudo-code in Algorithm 1.

Algorithm 1: Random Trajectory Generation algorithm as presented in [32]. L is the length of the trajectory.

Input: Trajectory: $X \in \mathbb{R}^{T \times D}$
Δ: Maximum distance to X
Precompute: A = finite difference matrix
$R^{-1} = (A^T A)^{-1}$
$Q = R^{-1}$ with each column scaled such that the maximum element is $1/L$
begin
  Generate difference matrix δX where each column vector $\theta_i \sim \mathcal{N}(0, R^{-1})$ for $i = 1, 2, 3, ..., D$
  while $DTW(X, X + \delta X) > \Delta$ **do**
    $\delta X = Q\delta X$
  end
end

The finite difference matrix referenced in algorithm 1 is a $(L+2) \times L$ matrix defined as:

$$A = \begin{bmatrix} 1 & 0 & 0 & & 0 & 0 & 0 \\ -2 & 1 & 0 & \dots & 0 & 0 & 0 \\ 1 & -2 & 1 & & 0 & 0 & 0 \\ & \vdots & & \ddots & & \vdots & \\ 0 & 0 & 0 & & 1 & 0 & 0 \\ 0 & 0 & 0 & & -2 & 1 & 0 \\ 0 & 0 & 0 & \dots & 1 & -2 & 1 \\ 0 & 0 & 0 & & 0 & 1 & -2 \\ 0 & 0 & 0 & & 0 & 0 & 1 \end{bmatrix} \tag{4.9}$$

Thus, when the system requests the creation of a new GMM when a new motion has been observed, the systems uses the set of observed positions to generate the several random trajectories, using the RTG algorithm described in algorithm 1. To each of the positions that form the trajectories generated by the RTG algorithm is appended a time index and new GMM is then fitted using these trajectories and added to the GMM library.

Finally, when the system requests a prediction for a given set of positions, the system first appends a time index to each of the positions and gets the probability of the input trajectory for every GMM in the library and the predicted target for the input trajectory is the target whose probability is highest.

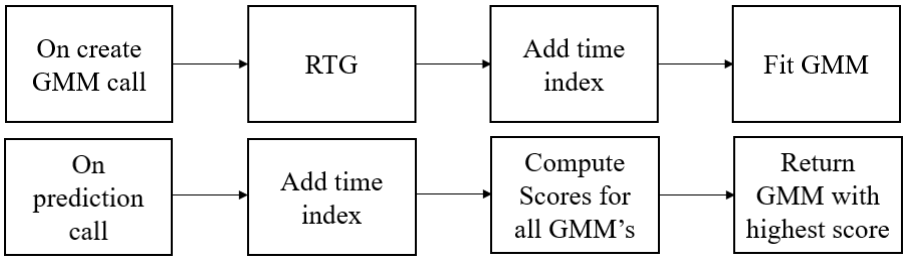


Figure 4.9: Schematic representation of the GMM manager services. Top - creation of a new GMM. Bottom - Prediction service.

4.3.2 Trajectory Manager

This node has three main responsibilities. The first is to maintain a record of the past positions in space, for that this node subscribes to the ROS topic advertised by the ***3d_loc*** node where the 3D positions are published and add each of them to a list. However, it is not desirable to keep those positions stored for prolonged intervals of time because it might affect the quality of the predictions as, depending on how long the positions are considered valid, might contain residual positions from the previous motion or even contain multiple motions. In order to keep only the most recent positions, the system attaches a timestamp to each of them when they are received. Alongside with the process' main thread, this node also has a separate thread of execution that deletes the positions older than a predefined time-to-live. The rate at which the thread is run can also be adjusted to accommodate different time-to-live values.

As stated at the beginning of this chapter, the system is intended to be action-agnostic, focusing instead on key areas and motions between them. In order to accomplish that this node maintains a record of the areas defined by the user. Each of those areas is characterised by a position, relative to the operator, a radius and a height offset, effectively creating cylinders where the operator's hand will eventually enter. At each new hand position the system determines whether or not that position is inside one of the cylinder. The current position P is inside one of the cylinders C_i, $i \in \{0, 1, ..., N_c\}$ if:

$$\sqrt{(P_x - C_{ix})^2 + (P_y - C_{iy})^2} \leq C_{iR} \tag{4.10}$$

and

$$|P_z - C_{iz}| \leq C_{izoff} \tag{4.11}$$

Where N_c is the number of areas defined by the user, C_R is the radius of the cylinder and C_{zoff} is the height offset.

The final major role this node plays in the proposed method is keeping a record of all the motions that have been performed by the user, which will allow the system to decide when to make a prediction or add a new motion to the library. In order to accomplish that goal the system keeps a record of both the previous and current key-area. However, this information alone is not enough to ensure that a valid motion has been performed, as the time-span over which the motion occurs is also relevant. In order to limit the duration of said motions, a timer is started when the previous position is known, i.e. when the user's hand leaves a key-area, and is cancelled only if when the hand moves into a new key-area. If that does not happen within the predefined time interval the timer expires and the previous position is set to unknown, effectively preventing the system from creating new motions until the operator's hand enter a new key-area.

All the information acquired by this node allows it to make decision on whether to request a prediction from the GMM_manager node, given the current saved positions, or the creation of a new GMM because a new motion has been observed. Figure 4.10 contains the schematic representation of the Trajecory manager node, where *areas* refer to the previous presented key-areas, TD is the distance travelled by the operator's hand and MIN_DIST is a variable parameter which can be adjusted to allow for earlier or later predictions based on the distance.

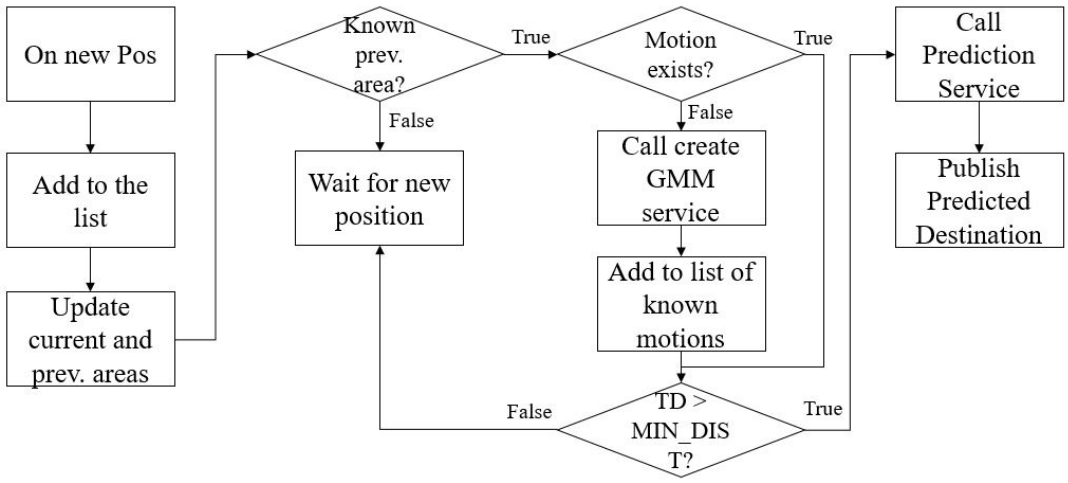


Figure 4.10: Schematic representation of the trajectory_manager node.

4.4 Robot Dispatcher

The main function of this node is the high-level control of the robot, namely this node uses joint states to command the robot to move to the different positions necessary during the execution of the task. To achieve this, the node keeps a set of joint state vectors, i.e. every joint angle of the robot, for each position. This node also keeps a set of positions which are

the aforementioned critical positions, where collisions between the robot and the human operator might occur, the node compares the critical area in which the operator's hand is predicted to be with the current target position for the robot. If the prediction matches the robot's target then the robot is halted, if not, the robot continues undisturbed.

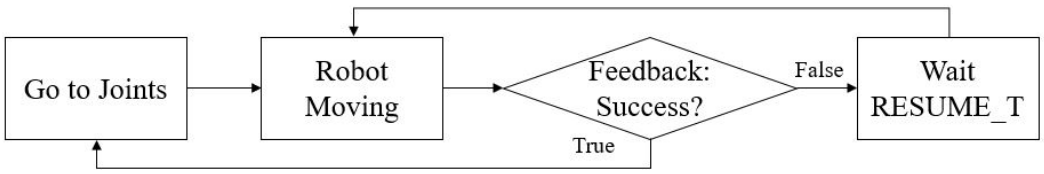
In case the robot stops due to a predicted collision a timer is started (or restarted in case a new prediction to the same critical area is received) and the robot will only resume operation once that timer has expired, the duration is a parameter that can be tuned for each particular task. Figure 4.11 - A represents the interaction between the move commands (*Go to Joints* represents a move command being issued by the Robot Dispatcher node), and the feedback sent by the MoveIt action server. A prediction may be received at any time (Figure 4.11 - B) and when a conflict is predicted, the node issues a stop command, which will cause the MoveIt action server to send a feedback message. Once a feedback message is received the node either waits for RESUME_T or issues a new move command. It is important to note that 4.11 A and B are asynchronous, therefore a prediction can be received at any time, regardless of A.

The interaction with the robot is made through the ROS-Industrial Universal Robot meta-package[42] and Moveit [43] which allows the programmer to use high-level commands to the robot such as go to specific coordinates or go to specific joint angles, and moveit uses a planner to accomplish that goal. As for the movement control of the robot, this node uses destination joint angle states and the topics provided by the action server implemented by Moveit, which provide the node with information regarding the status of a particular goal. This particular node uses the topic */move_group/result* in which is published a message reporting the result of the plan execution. This information is used to manage the robot's motion, if the result indicate the plan execution was cancelled, because of a prediction then the system executes the logic to avoid the collision, otherwise, if the result indicates the successful execution of the plan the system goes to the next position as defined by the task.

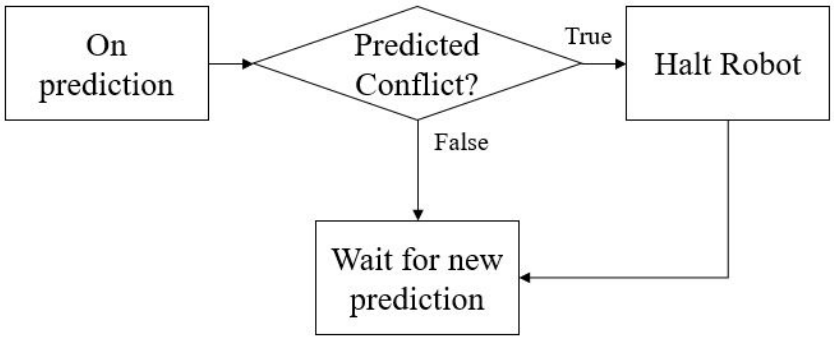
Regarding the robot used in the proposed approach, it is noteworthy to reiterate what was stated in Chapter 1, this system intends to improve the interaction between human workers and industrial collaborative robots by proposing a system which would allow the robot to emulate the ability that humans possess to predict and react to each other's motions. However, it does not, nor was it designed to, make additional guarantees regarding safety. The safety features used are the ones implemented by the collaborative robot.

With these considerations in mind the robot used in the experiments, described in the next chapter, was the Universal Robot UR10 (Figure 4.12). The UR10 is a lightweight industrial manipulator capable of carrying a payload up to 10Kg and implements the necessary safety features to be used in a collaborative situation.

Finally, the package ur_robot _driver [44] is a driver for the Universal robots, namely UR3, UR5 and UR10, which provide an interface between the robot and the ROS environment. This node is used alongside the externalcontrol-1.0.4.urcap software which needs



A)



B)

Figure 4.11: Schematic representation of the Robot Dispatcher node. A) integration of movement commands with task feedback from MoveIt. B) Handling of predictions.

to be started from the robot's teach pendent to allow the external control of the robot. The robot is connected to the machine running the system over Ethernet.

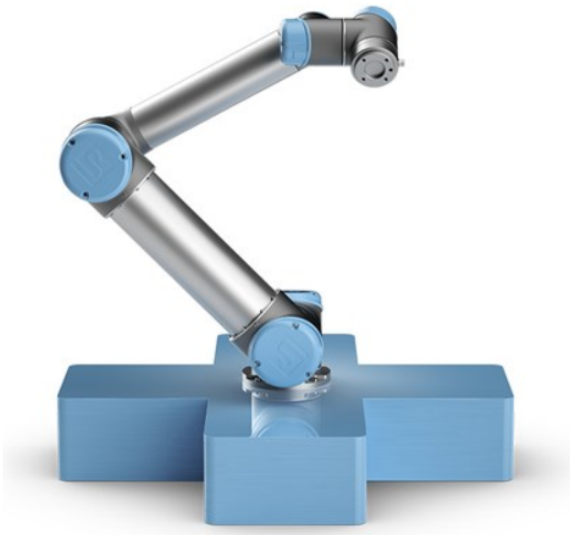


Figure 4.12: Universal Robots UR10. Image from[45].

Chapter 5

Experimental Results

This chapter describes the experimental results obtained with the different components of the proposed system, as well as the method to perform the different tests.

Similarly to the way the system was described during the previous chapter, this chapter also intends to show the results obtained by the different components in a cumulative way, by testing each component individually and progressively joining them with the other components until the system is complete.

To this effect, this chapter will present the results obtained by the method described in the previous chapter used to compute the 3D coordinates for each pixel in the depth image, showing a reconstruction of 3D scenes in a point cloud representation. Compare some characteristic of different models for body joints detection in order to choose the model which allows for the best real time performance, given the hardware used during the testing phase. In addition, some captured trajectories are also shown here.

Finally, this chapter presents the testing results for the whole system as well as a simulated collaborative task.

5.1 Implementation Details

As stated in the previous chapter each of the blocks described in the Proposed Approach chapter are implemented as ROS nodes and use the ROS mechanisms for inter-process communication, namely ROS Services and Topics. The main function of the ***2d_loc*** node is to run the neural network, in order to do so the system uses the tensorflow-gpu backend. This node was implemented in Python and also makes use of Numpy library for matrix operations. The ***3d_loc*** node was implemented in C++ and makes use of OpenCV to pre-process the images published by the freenect_camera node, specifically to convert the RGB frames, which are directly published as Bayer pattern encoded, to RGB.

The ***gmm_manager*** node was implemented in Python and uses the scikit-learn implementaion of the Gaussian Mixture Model. Finally, the ***trajectory_manager*** and the ***robot_dispatcher*** nodes were implemented in C++. The latter uses the MoveIt framework to interact with the robot.

All the tests described in this chapter were performed on a single machine with an Intel(R) Core i7-8750H @ 2.20 GHz with 16GB of RAM and Nvidia GTX 1050 Ti, running Xubuntu 18.04 and ROS Melodic.

5.2 Data Acquisition Tests

This section will go over the results obtained on the data acquisition component of the system, namely the RGB and corresponding depth image, results from the joint position estimation. As stated in the previous chapter the package which implements the neural network model [38] includes, not only an implementation of the CMU's original model but also three other models which are intended for less powerful machines, this section will also cover the differences between the models in key aspects for this project such as variation in the detection and processing time.

5.2.1 2D and 3D data

The images presented in this section intend to show the data acquired by the Kinect sensor, both RGB and depth, as well as the results obtained by the method used to compute the coordinates in 3D space in the form of a point cloud. As such, the acquired data by the sensor was acquired in three scenes with three objects on a table-top in different positions. The images were captured by the Kinect sensor at approximately one meter above ground with a tilt of roughly six degrees.

Figures 5.1, 5.3 and 5.4 represent the depth data acquired by the Kinect sensor. For better visualisation the depth values were normalised and colour coded, where red tones indicate a shorter distance to the object and green tones represent longer distances, i.e. the objects get greener, as they get further away from the sensor.

As for the point cloud representation of each scene, the coordinates of each point were computed using the method presented in the previous chapter and the colour of each point is tied to the z coordinate. The point cloud reconstruction of the scenes are represented in Figures 5.2, 5.5 and 5.6 viewed from the front (top-left), top (centre) and side (bottom) as well as in perspective (top-right). The first scene (Figure 5.1) contains the objects placed at different distances from the sensor, in the second scene (Figure 5.3) two of the objects were placed side by side with the third object being placed in front of the other. In the third scene (Figure 5.4) the objects were placed closer to the sensor in the same plane.

In the different images presented so far it is possible to observe a section in the largest object where the depth data is not available. By changing the position of the sensor one

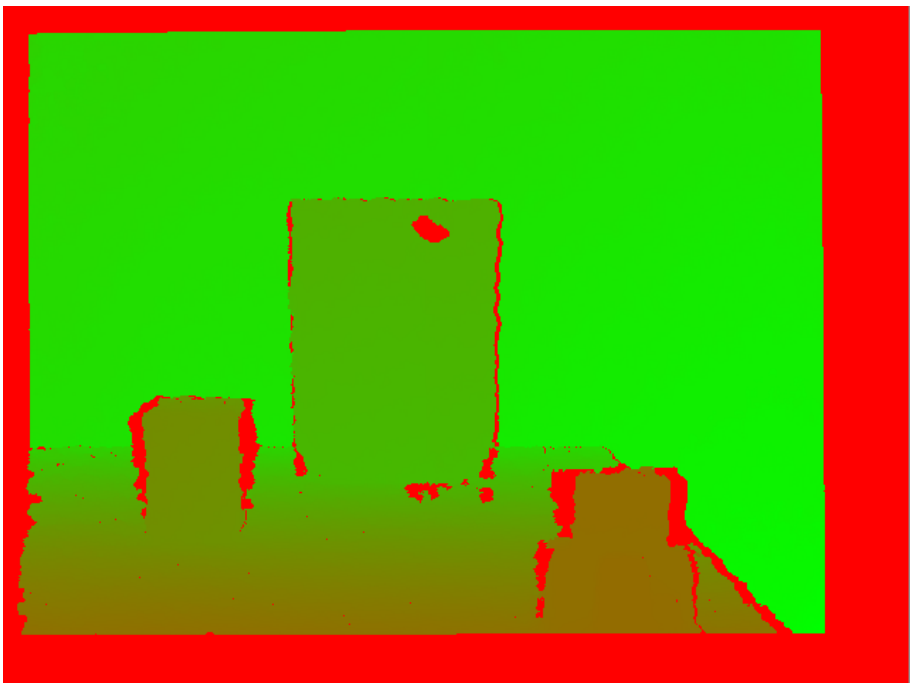


Figure 5.1: Coloured depth image from the first scene.

can see that the spot position also changes, as can be seen in 5.7 and 5.8. This is one of the consequences of the structured light method of depth data acquisition, which uses an IR projection as a method for depth calculation. The object in question has a highly reflective surface which reflects the light emitted by the projector, thus preventing the determination of the distance. This can be further confirmed by examining the IR images for the same situations in Figure 5.9.

Furthermore, whilst using this sensor the shadows cast by objects closer to the sensor prevents the system to determine the distance to those areas where the pattern could not be projected. This phenomenon can be observed in the colour depth images as a red outline surrounding the objects (Figure 5.10 - Left) and as shadows in the images captured by the IR sensor (Figure 5.10 - Right).

5.2.2 Human Joint Detection

The next step is the detection of human joints from the RGB image and computation of the correspondent 3D position with the depth data. The detection of joints is performed by the neural network model from [38]. This repository provides four models which detect the desired joints, which present variation of both processing time and accuracy of the resulting joint locations. This section will analyse the differences between the different models, namely the processing time, the number of processed frames and variation of location between consecutive frames.

In order to analyse the proposed factors the measurements were performed on five Rosbags which contain the RGB and depth images of person performing different motions

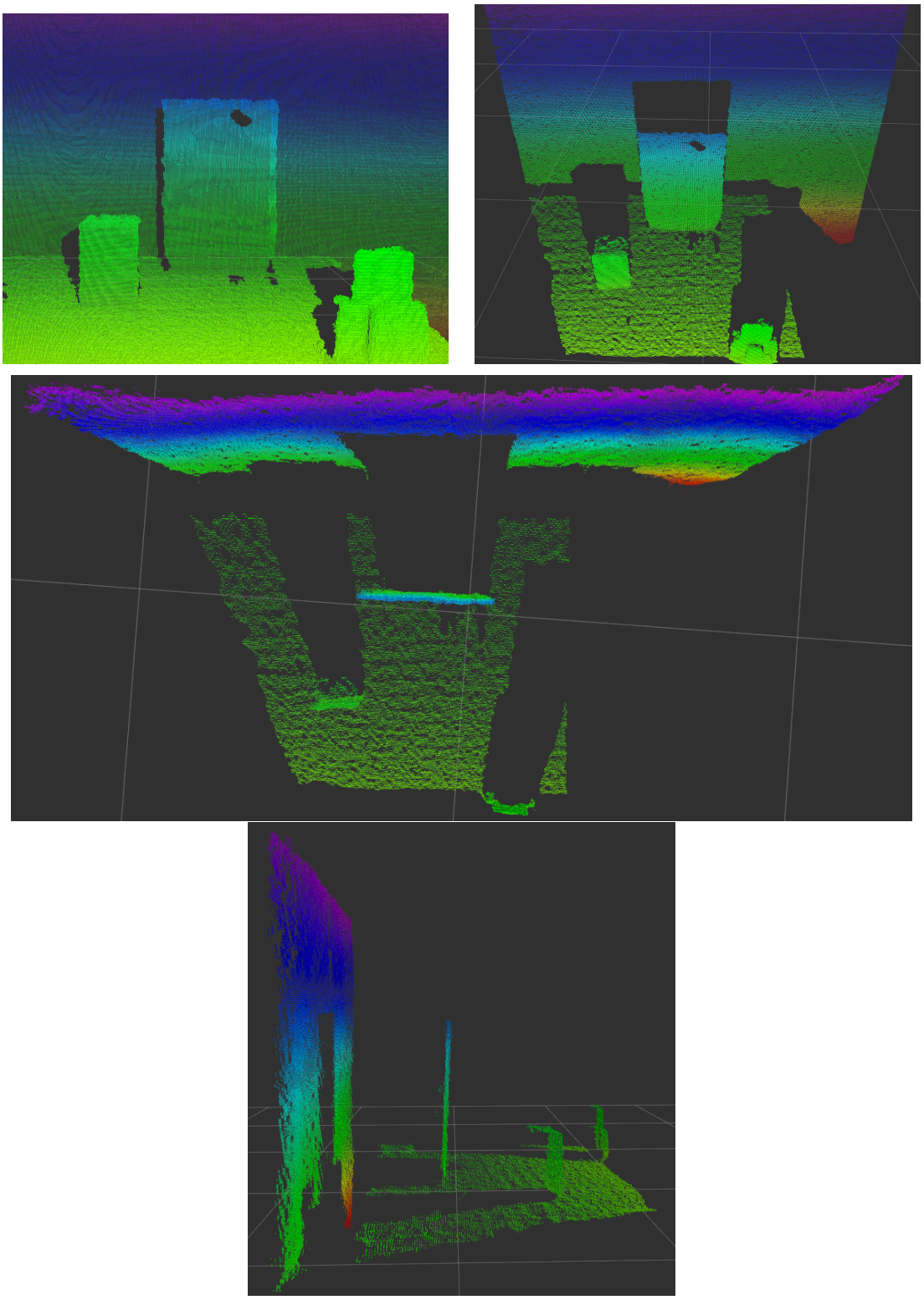


Figure 5.2: Point cloud representation of the depth data from the first scene.

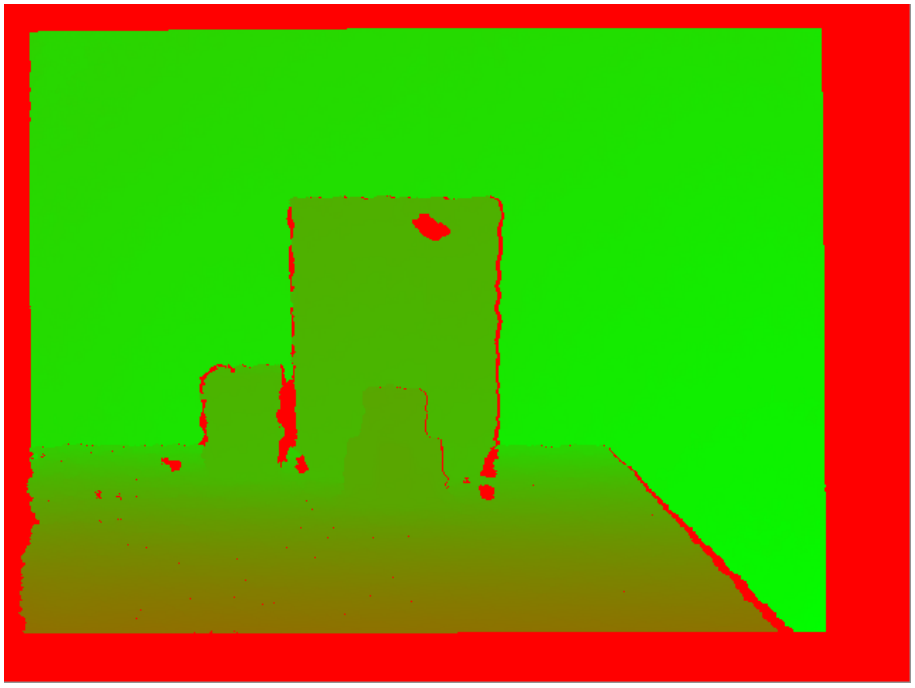


Figure 5.3: Coloured depth image from the second scene.

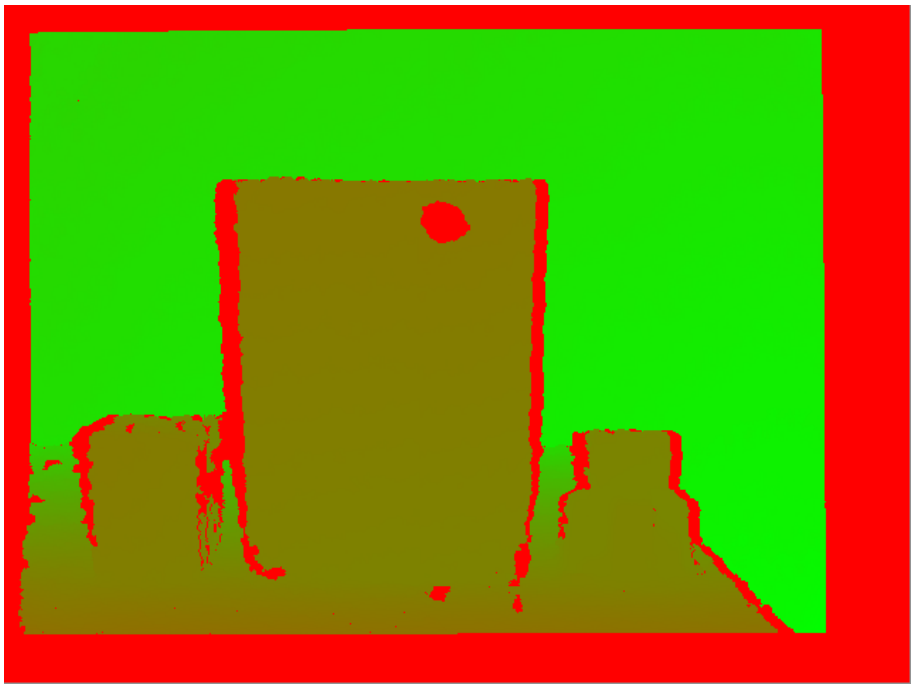


Figure 5.4: Coloured depth image from the third scene.

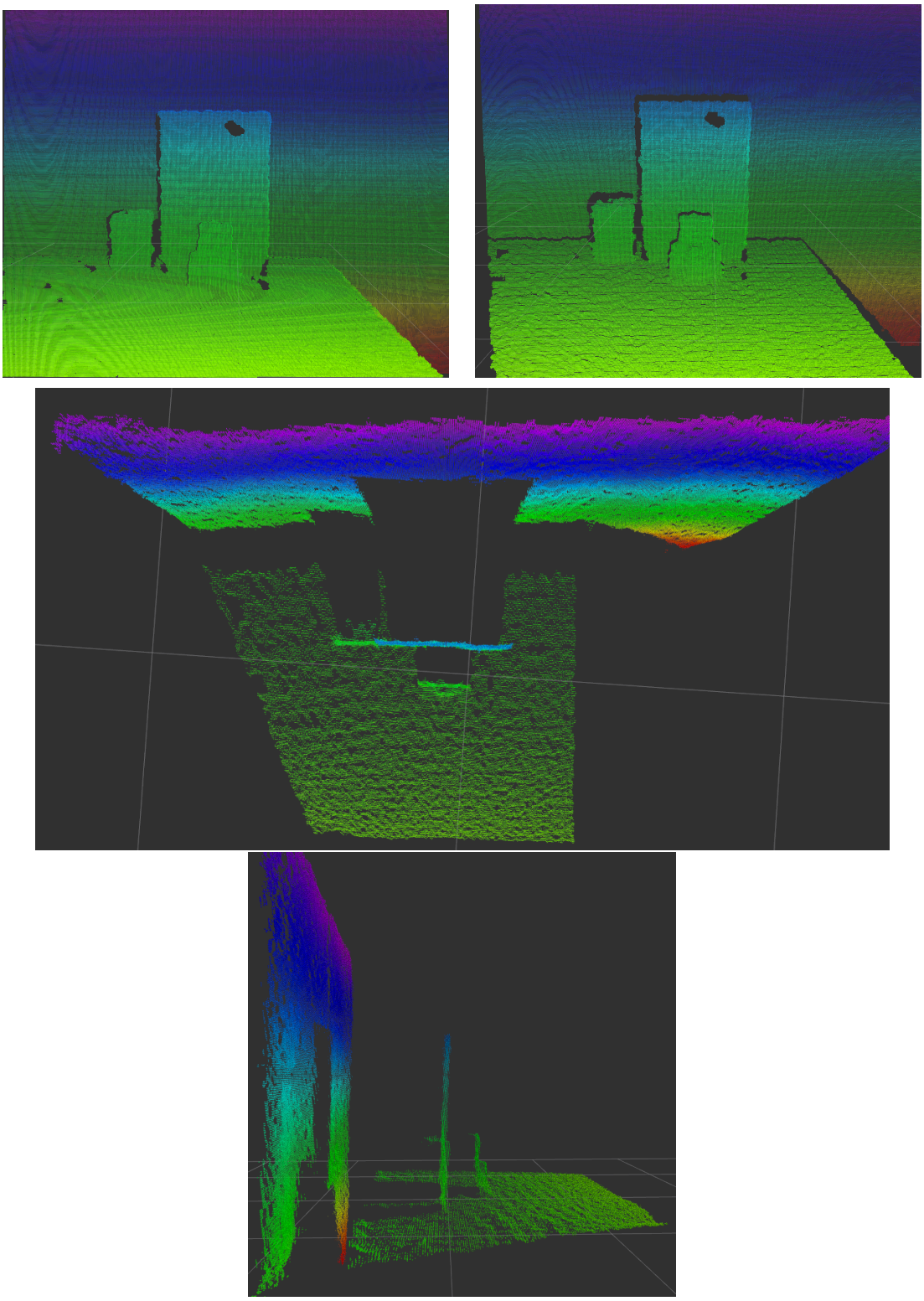


Figure 5.5: Point cloud formed from the data captured for the second scene.

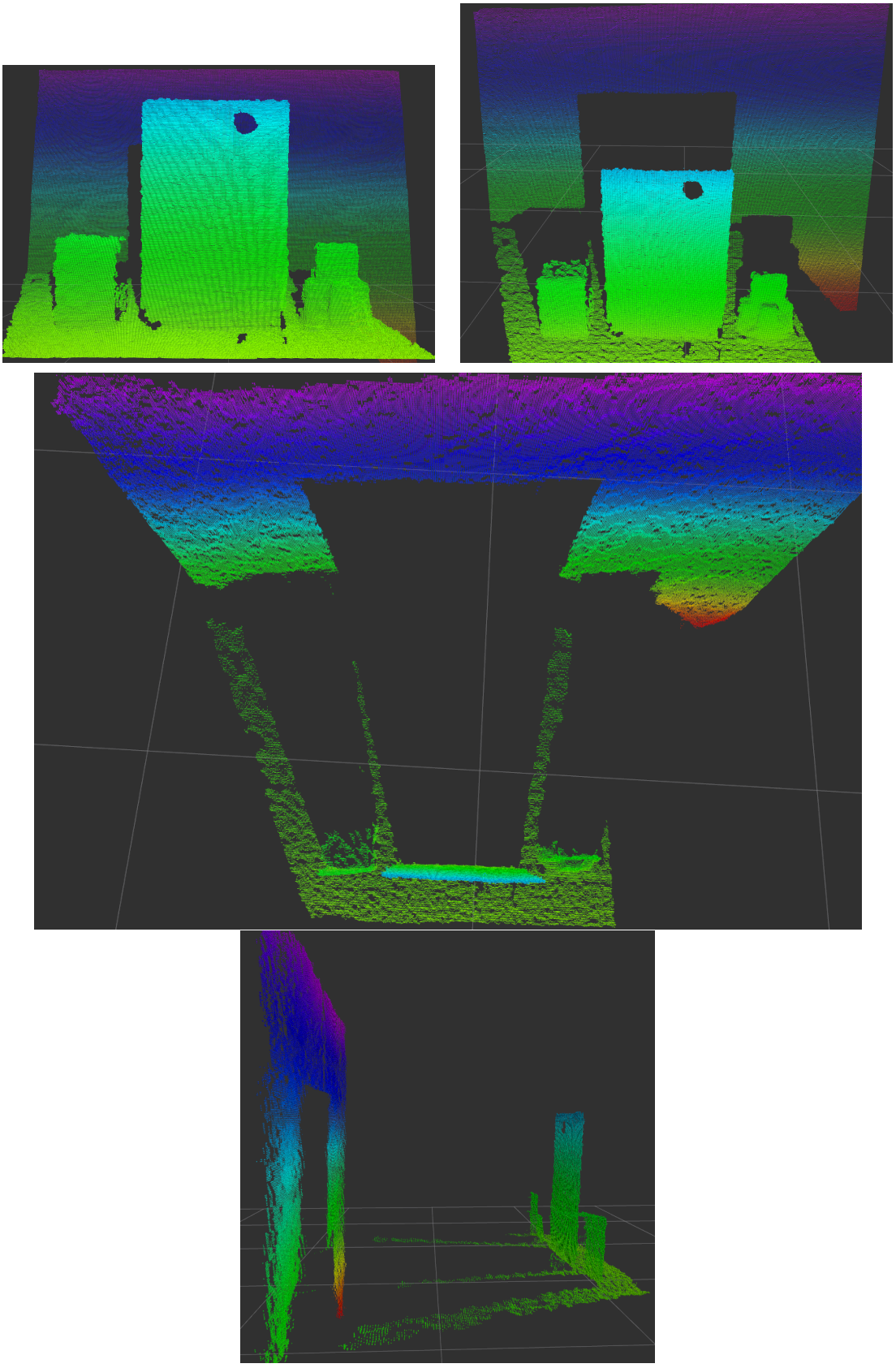


Figure 5.6: Point cloud formed from the data captured for the third scene.

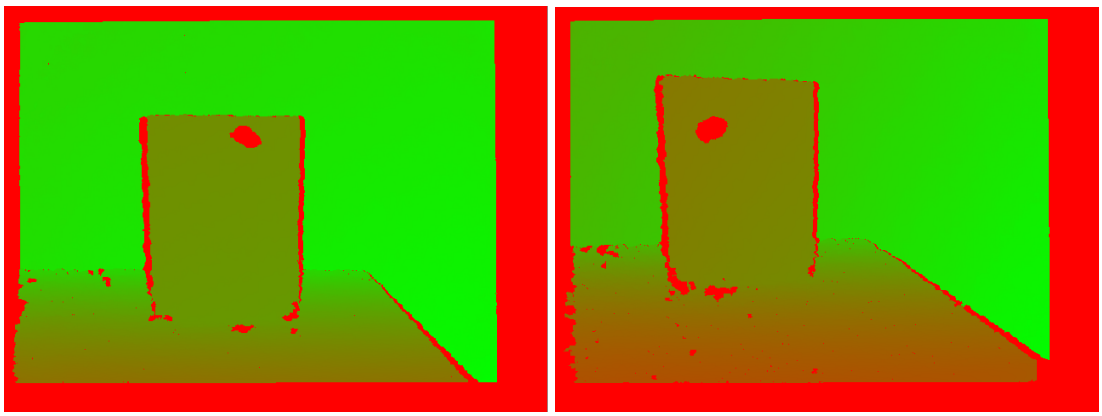


Figure 5.7: Position of the spot with unavailable depth data. Left - the Kinect sensor is on the same position as the previously shown examples. Right - the image captured after moving the Kinect sensor.

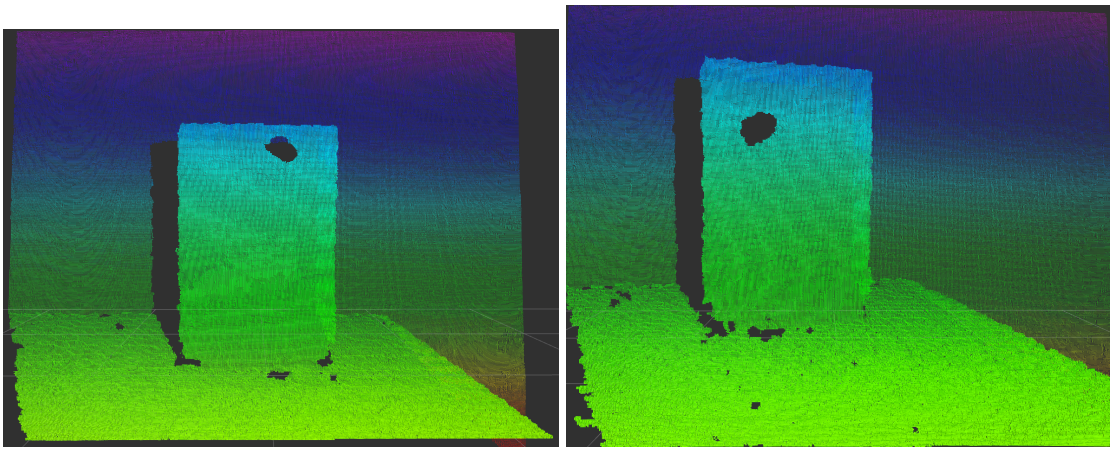


Figure 5.8: Point cloud representation of the depth images shown in 5.7. Each point in the point cloud is coloured according to the value of the z coordinate.

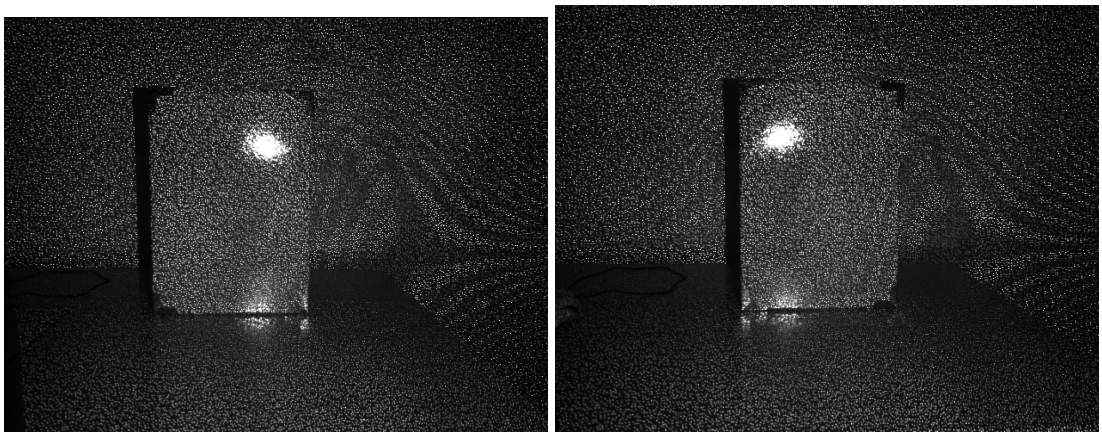


Figure 5.9: Images captured by the IR sensor with the visible IR projection reflection.

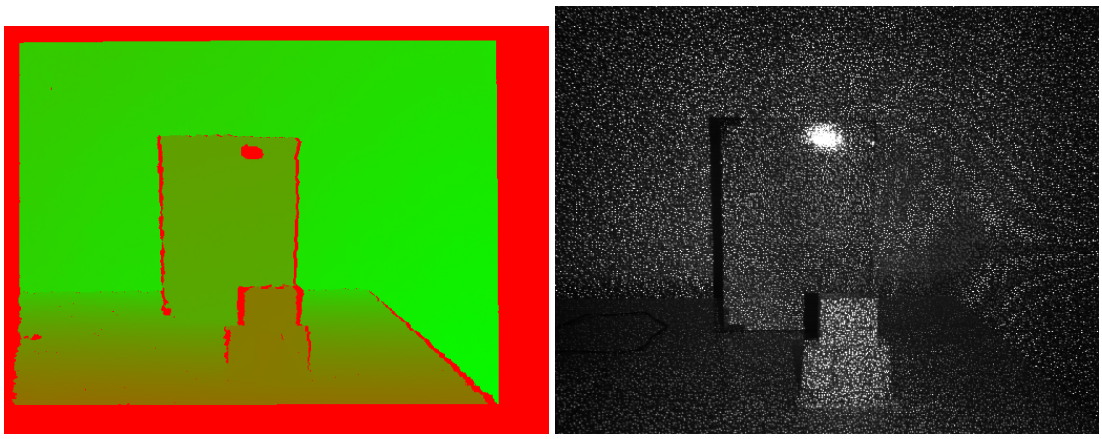


Figure 5.10: On the coloured depth image one can see the red outline on both objects. On the right is the correspondent IR image.

such as walking forwards and backwards, walking sideways and different arm motions whilst standing in the same location. Those bags have different duration and different number of messages.

Bag	rosbag info msgs	callback calls
1	195	194
2*	160	106
3	195	192
4*	159	103
5*	678	613

Table 5.1: Comparison of the number of messages reported by the rosbag info program and the number of times the callback function of the topic is called. The bags marked with * displayed a visible "jump" to a future frame.

According to the analysis of the rostopic hz program RGB and depth images are published at 30Hz. However, since the system uses a message filter to synchronise the RGB and depth messages it may influence the number of frames that are actually received. To determine the impact of the synchronisation method in the number of calls to the callback function associated with the message filter the system incremented a counter each time the callback function was executed. This process was repeated for every recorded bag file and the results are displayed in table 5.1. By analysing the data one can observe the differences between the number of messages shown by the rosbag info analysis and the number of callbacks calls, particularly, it is possible to notice the relatively large difference between the two, for the bags marked with a (*), where it was also possible to observe a noticeable "jump" to a future frame during the bag playback. By disabling the synchronisation mechanism this does not appear to happen, allowing one to hypothesise that a delay longer than the filter's threshold occurs between messages and, consequently, results in the system waiting for the corresponding message. Such events are relevant as

Model	Average Processing Time (ms)
mobilenet_v2_large	79.71
mobilenet_v2_small	72.55
mobilenet_thin	77.33
cmu	165.30

Table 5.2: Comparison of the average computation time for the different Neural Networks for bag file 1.

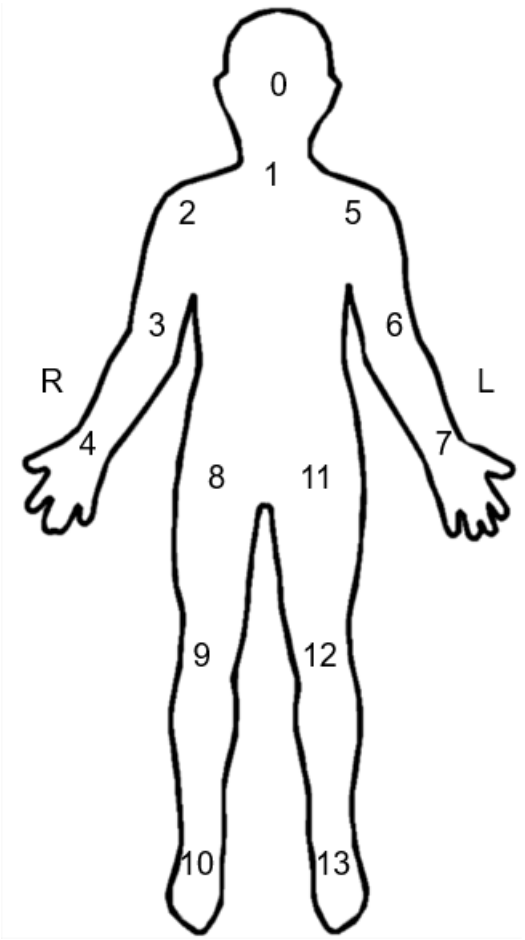


Figure 5.11: Numerical identifier for the different detected anatomical key-points. Adapted from [0].

they may happen during normal operation which obviously impacts the detected trajectories.

Because the reaching motions which are the targets of the proposed system are very short the number of positions for the trajectories is heavily influenced by the processing time of each frame and the average time elapsed between the service call and the arrival of the response for the different models can be seen in table 5.2 The data produced by the neural network model is a set of coordinates which describe the position of each of the body parts in the image. Although the model is also capable of detecting the anatomical key-points in images with more than one human in the RGB image, this functionality was

Figure 5.12: Detected joint locations on the RGB image. Each green circle is centred on the pixel pinpointed by the neural network.

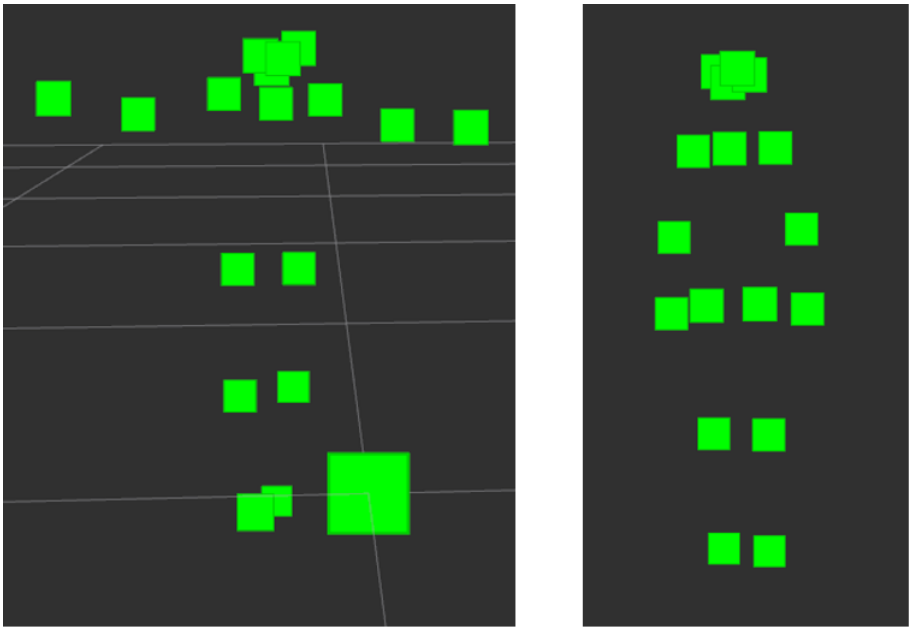


Figure 5.13: Joint position in 3D space. On the right the human adopted a neutral stand. On the left the subject kept the arms raised.

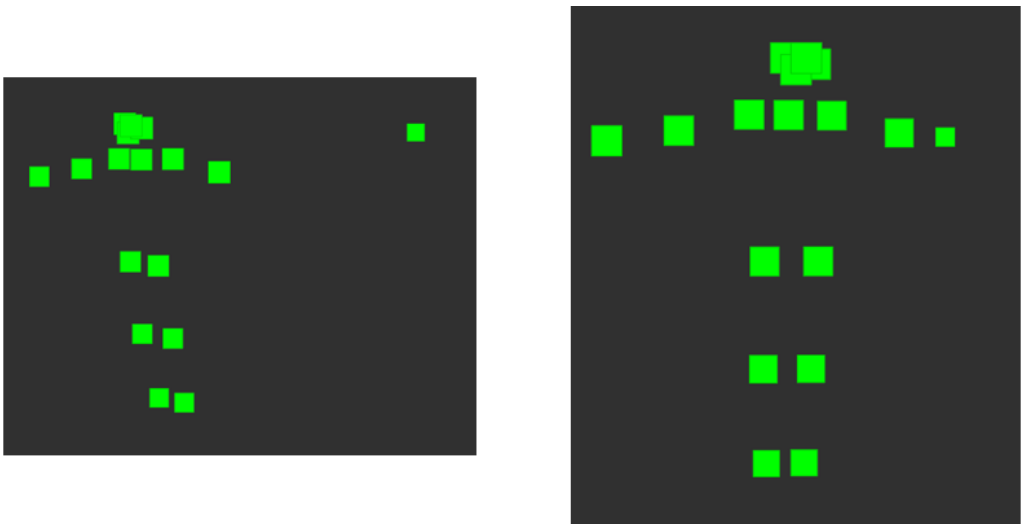


Figure 5.14: Bad depth information on the pixel identified by the neural network on the RGB image.

not used in the proposed approach. Each body part is identified with an index which are represented in figure 5.11 and an example representation of the detection result on an RGB image can be seen in 5.12. In figure 5.12 it is also possible to see extra points detected on the subject's face which are not represented in 5.11, those points were disregarded as they constitute irrelevant information for the proposed approach. Furthermore, upon reviewing the acquired data for the 3D position, they resulted in the most unreliable data. Therefore, this data will not be included in further analysis.

By merging the information output by the neural network and the depth information the location of each joint can be computed as described in the previous chapter, two examples of the this method can be observed in figure 5.13. The image on the right represents the correct position of the anatomical key-points. Conversely on the left image, one can observe the misplacement of one point. This happens when the kinect sensor could not retrieve depth information for that particular pixel. Another problem that might influence the performance of the trajectory detection is represented in image 5.14, where the neural network correctly identified the body part and its location, but the depth data present in that pixel on the depth image is not compatible with the rest of the joint locations. There are two situations which may happen that cause this phenomenon, the first is the fact that the neural network outputs the normalised coordinates, and in order to get the pixel position on the image it may be required to make approximations, therefore choosing the wrong pixel on the depth image, or it is also possible that the depth data was not correctly acquired because of the dimension of the target.

During testing one can notice, by visually examining the location of the joints on the RGB image, that each of the output locations have significant variations between consecutive frames, even if the subject remains stationary. The variability of those positions is different for each model and individual body part. In order to choose the most efficient

model for the task at hand, one should take into consideration the processing time, as it will impact the number of positions in each trajectory and the variation of the detected positions. If the key-point detection is very precise but takes too long the trajectories will contain a low number of positions, or, if the opposite happens, each trajectory will have a high number of positions but each of those positions might not even correspond to the true location of the user's right hand. Therefore a balance between the two must be achieved in favour of creating the best possible trajectories. The processing time has already been analysed in this chapter and the data can be seen in table 5.2. As for the variation, two rosbags were recorded where a subject stood in front of the Kinect sensor and remained stationary for a few seconds. In one of the rosbags the subject assumed a neutral stance (standing with arms resting alongside the body) and, in the second rosbag, the subject's arms were kept elevated at shoulder height. In both cases the entirety of the subject's body is inside the field of view of the kinect sensor for the entirety of the rosbag duration. For each frame of each rosbag the system outputs the positions of the fourteen anatomical key points and saves it to a csv file and is then processed by a Python script which removes every row which is incomplete, i.e. one or more anatomical key-points were not detected, and, for each body part, computes the euclidean distance between consecutive frames. Finally the average distance is computed for each body part and the results can be seen in tables 5.3 and 5.4, for the raised arms and neutral stance, respectively.

Given the results obtained from the average euclidean distance between two consecutive frames, in particular the user's right hand (index four in Figure 5.11), and the average processing time the author chose to use the mobilenet_v2_large, which appear to achieve the best balance between 3D position stability and processing time.

This is the case for the system running on the machine described in the beginning of this chapter, by running the system on a more powerful machine this may not hold true and other models may be more adequate.

5.3 Trajectory Detection

By combining the position detected by the ***3d_loc*** node and the location of each key-area, the system can define trajectories between key-areas. In Figure 5.15 is represented the top-view of a hypothetical table-top where three key-areas were defined, in red the critical areas one (CA1) and two (CA2) and, in green, a non-critical area (N). The possible trajectories, that the system would be able to recognise are N→ CA1, CA1→ N, N→ CA2, CA2→ N, CA1→ CA2 and CA2→ CA1.

5.3.1 Example Trajectories

In Figure 5.16 two trajectories captured by the Kinect sensor are shown. These trajectories are later used by the RTG algorithm in order to create similar trajectories which allow for

Body Part Index	CMU	MV2-large	MV2-small	M-thin
0	0.005396	0.014719	0.015266	0.012109
1	0.020475	0.015929	0.020584	0.008476
2	0.018777	0.031514	0.018083	0.009781
3	0.009657	0.040611	0.139048	0.017701
4	0.194208	**0.363209**	0.655696	0.837775
5	0.015347	0.019609	0.021108	0.013489
6	0.018174	0.021538	0.109469	0.016169
7	0.280342	0.635834	0.651210	0.405433
8	0.005806	0.012463	0.251815	0.010976
9	0.017301	0.005677	0.281122	0.017567
10	0.010478	0.018554	0.442596	0.011291
11	0.009741	0.015403	0.270841	0.009759
12	0.024351	0.016226	0.351491	0.016080
13	0.018932	0.019957	0.769529	0.020657

Table 5.3: Euclidean distance between consecutive frames for each anatomical key points, whilst the subject in-frame stands stationary assuming an "open arms" position. The body part indices are the indices depicted in Figure 5.11.

Body Part Index	CMU	MV2-large	MV2-small	M-thin
0	0.005271	0.014963	0.011613	0.009381
1	0.004438	0.013758	0.014039	0.000776
2	0.010252	0.020026	0.020310	0.012636
3	0.011355	0.019964	0.148550	0.016346
4	0.014283	**0.012686**	0.092152	0.007245
5	0.016073	0.011494	0.017579	0.015773
6	0.006888	0.019282	0.022118	0.016140
7	0.109226	0.028715	0.180674	0.065957
8	0.003694	0.010910	0.020524	0.008082
9	0.006188	0.004936	0.025086	0.007330
10	0.010051	0.017682	0.042138	0.019559
11	0.012790	0.011467	0.018613	0.009636
12	0.025856	0.016633	0.037017	0.006536
13	0.017273	0.018104	0.053920	0.028114

Table 5.4: Euclidean distance between consecutive frames for each anatomical key points, whilst the subject in-frame stands stationary assuming a neutral position. The body part indices are the indices depicted in Figure 5.11.

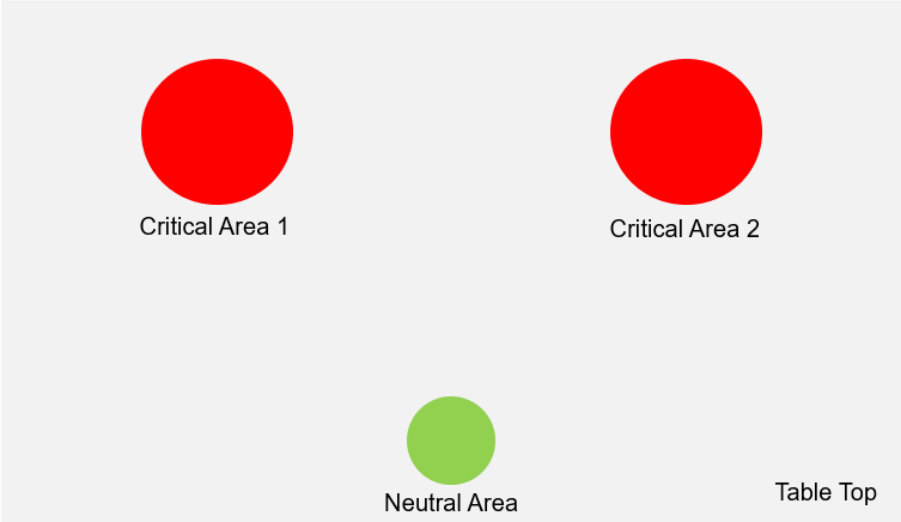


Figure 5.15: Example of a situation with three critical areas (red) and one non-critical key-area (green).

small variations which will likely occur in real life, because the user's arm motion will not be the exact same every time it is performed. The RTG algorithm has a mechanism which allows the tuning of how "similar" the generated trajectories are to the original, that parameter is Δ in Algorithm 1. Figures 5.17 and 5.18 represent the impact of the Δ value. On both figures, on the left the use of a smaller Δ value resulted in the generated trajectories being much more similar to the original trajectory, when compared to the example on the right on both figures, which uses a higher Δ value.

This value also had to be tuned in order to achieve the best possible predictions, as if the value is too low than the confidence of motions for the same target will be too low due to small variations of the trajectory, or if it is too large than the system will not be able to distinguish between the different types of trajectories.

Given the flaws discussed with this system previously, the same effects would be expected in the detected trajectories, and do, in fact, happen and have a prominent effect on trajectory detection. Two examples of such situations are represented in figures 5.19 and 5.20. The first figure is a trajectory where many positions could not be detected either because the neural network could not detect the user's hand or due to the synchronisation mechanism, as referred above. The second, on the other hand, although it contains the expected number of points, there is some erratic variation in some of the points, which may be caused by variation on the detected position on the RGB image or incorrect depth information caused by the relative position of the subject's body and the sensor.

5.4 Reach Motion Prediction and Robot Interaction

The results shown in this chapter so far have been obtained by gradually adding more functionality to each test up to the point where the whole system is running. The

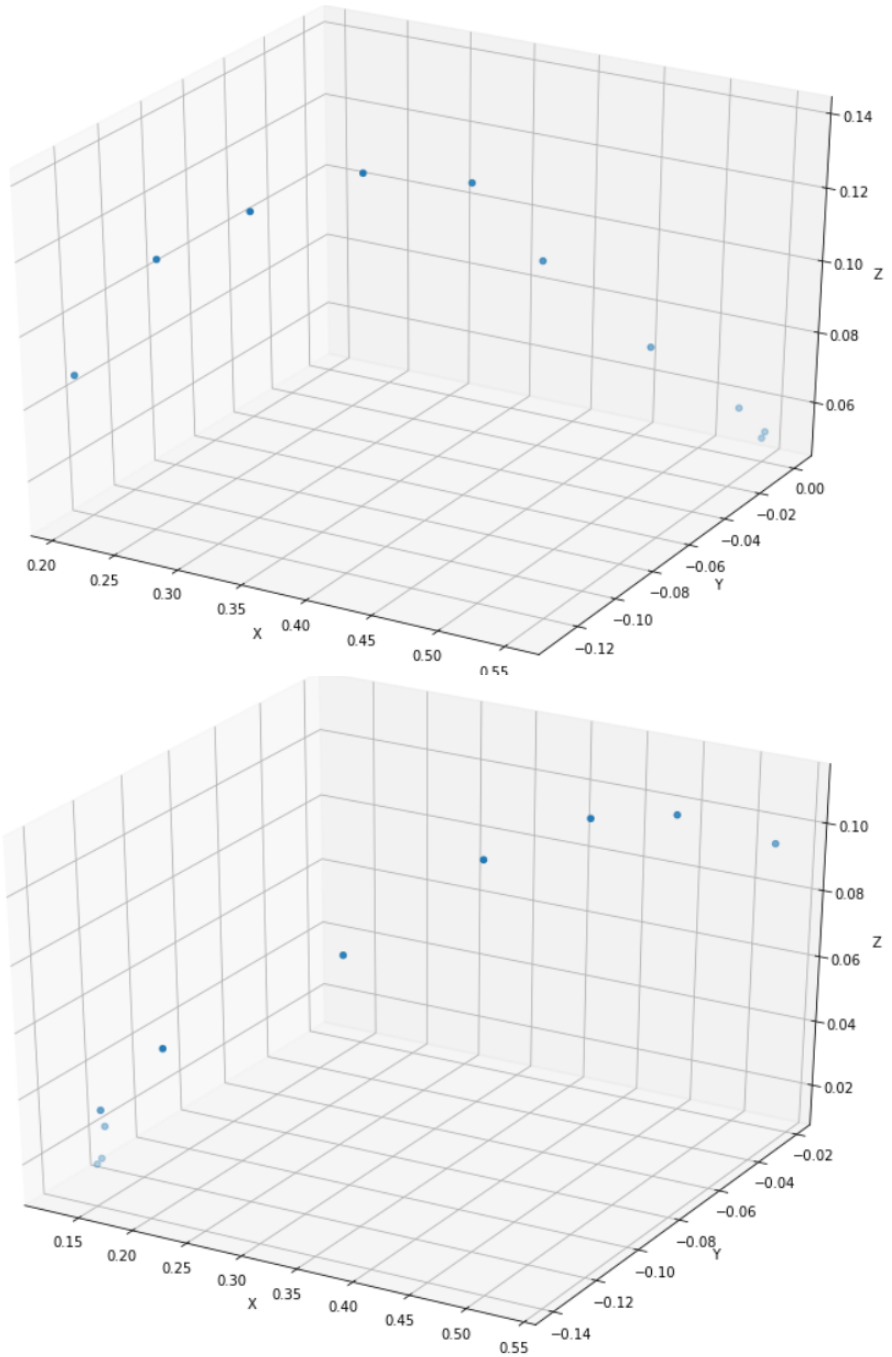


Figure 5.16: Two examples of real trajectories between two key-areas captured by the Kinect sensor. Each blue data point represents a position published by the ***3d_loc*** node.

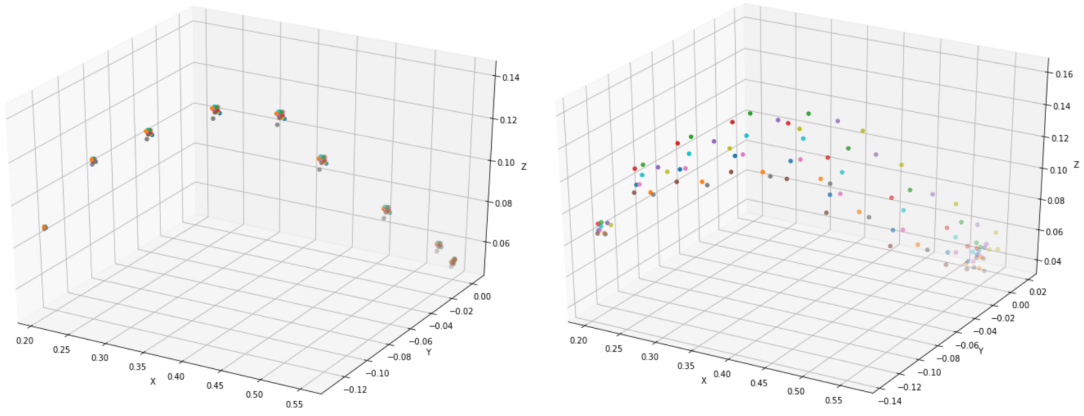


Figure 5.17: Trajectories generated by the RTG algorithm with different values for Δ, which uses the trajectory on the top image of figure 5.16. Each colour in this image represents one of the generated trajectories.

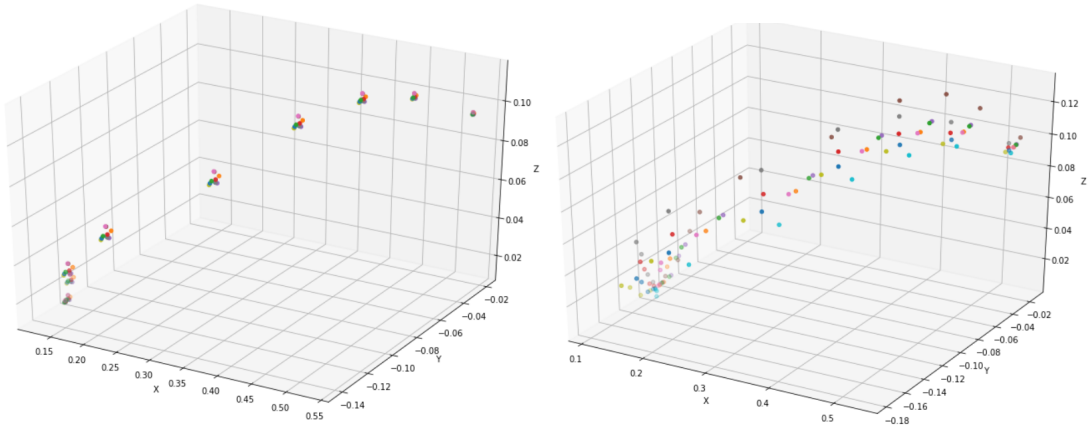


Figure 5.18: Trajectories generated by the RTG algorithm with different values for Δ, which uses the trajectory on the bottom image of figure 5.16.

robot_dispatcher node, however, was not tested separately as the results were trivial, given the nature of the developed node for this purpose. This section will go over the tests performed with the whole system. Specifically, five tests were performed with the proposed approach, varying two parameters which play significant roles in the performance of the system as a whole, namely the robot movement speed and the number of key-areas. As a prelude to the obtained results, one could speculate on the role of both parameters in the performance of the system, as they appear to be quite clear. The robot speed directly influences the time available for a correct prediction from the system and a larger number of key-areas will result in larger uncertainty on the destination, particularly if the critical areas are placed in close proximity of each other, whereas a smaller number of possible targets will decrease that uncertainty, thus allowing for earlier predictions.

The first test has only two key-areas where only one of them is a critical area. This test was repeated three times, varying the robot speed. The second and third tests were

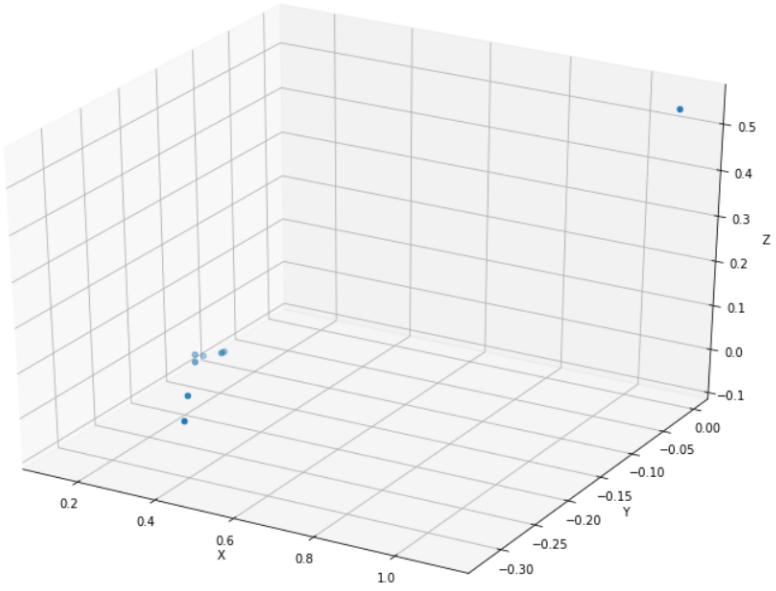


Figure 5.19: Failed to detect hand positions in a significant part of the trajectory between two key-areas.

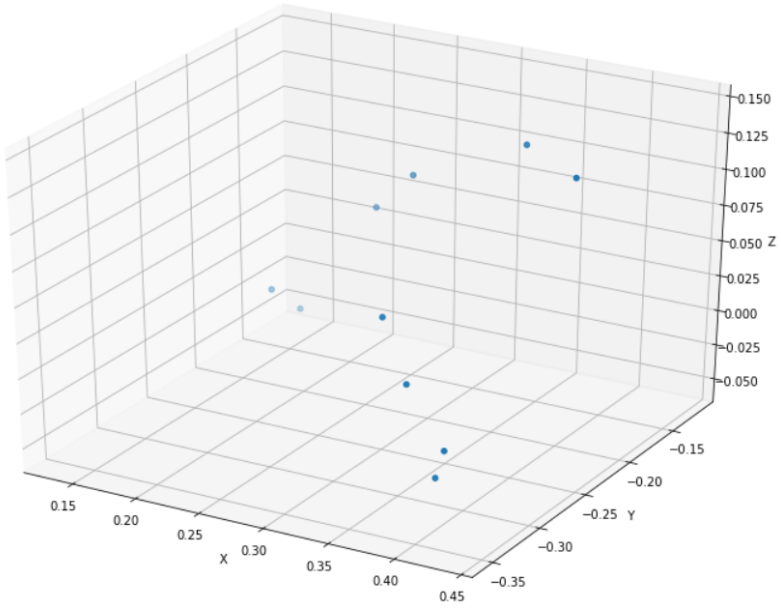


Figure 5.20: Trajectory with erratic position variation. Each data point represents the human operator's hand at a given moment in time.

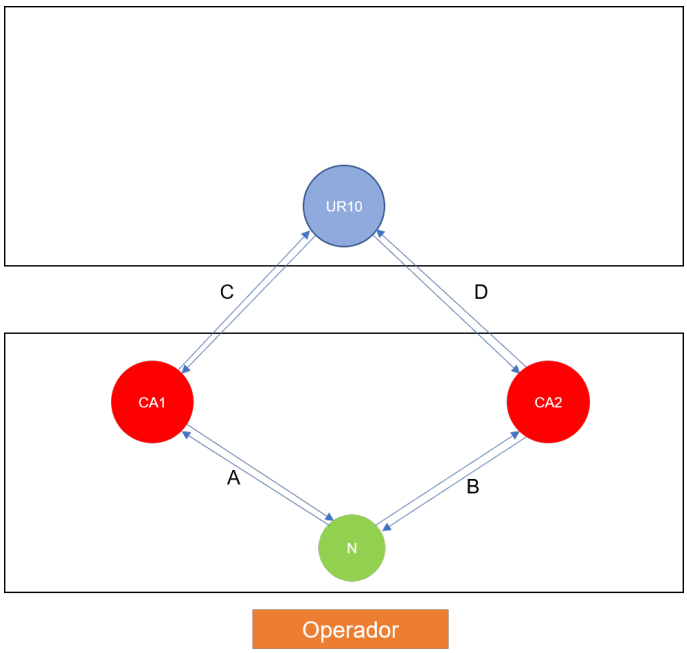


Figure 5.21: During the testing process the robot performs trajectories C and D repeatedly, whilst the human operator may either perform trajectory A or B. This image portraits a situation with two critical areas but the process is analogous for one or three critical areas.

performed with three key-areas where two of them were critical, the difference between them is the motion speed of the robot. Each of them were repeated three times because it became apparent the impact of the quality of the detected trajectories, during the initialisation phase, on the prediction performance.

Finally, the final fourth and fifth tests were performed with four key-areas, three of them critical. Analogously to the tests two and three, the only different factor between these tests is the robot speed. Each of them was repeated three times. Figure 5.21 shows the type of motions performed by the robot and the human operator.

5.4.1 Test 1 - One Critical Area

For this test, as for the rest of them, the user first performs the necessary trajectories thereby enabling the system to make predictions for future reaching motions based on the similarity with the trajectory used during the initialisation phase. After the initialisation process the robot initiates its task. The robot's task is a simple set of motions designed to simulate a situation where the robot needs, to access the critical area and other non-critical areas during operation. The operator, on the other hand, performs motions which both use the critical area, in order to force the situation where a collision could occur, and other areas where the robot would not need to stop. The temporal evolution of the tests is present in figure 5.23, which contains in blue the key-area which the user's hand is in, at that specific moment in time, or -1 if the hand is not inside any of the areas. The

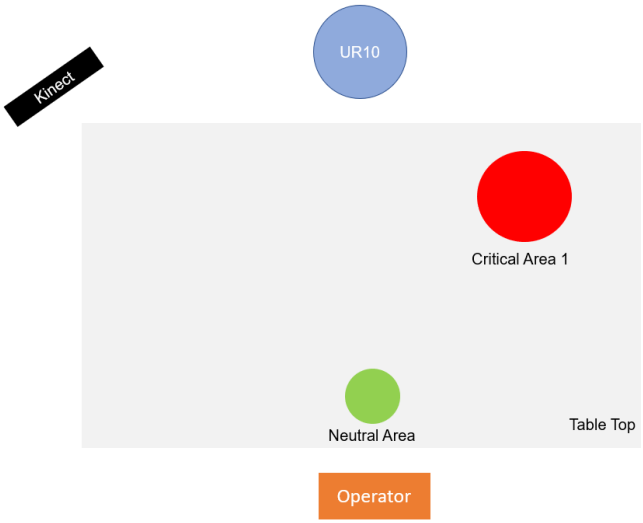


Figure 5.22: Setup used during the first test. This image is not to scale.

system's predictions are represented in orange and in grey the moveit action server result topic, which takes the value of 1 if the plan was successfully achieved, or -4 if the robot was stopped, i.e. the prediction caused the robot to abort the plan. Although it is possible that other factors would cause a -4 code to be published in this topic, during this test that did not happen.

Even though there is only one critical area, and consequently only two possible trajectories, it is possible to observe in figure 5.23 the importance of the initialisation process, as the number of predictions in 5.23 - C is clearly inferior to the number in 5.23 - A. Furthermore, during the test some predictions arrived later than ideal. An example of this phenomenon is represented in figure 5.23 - A (A).

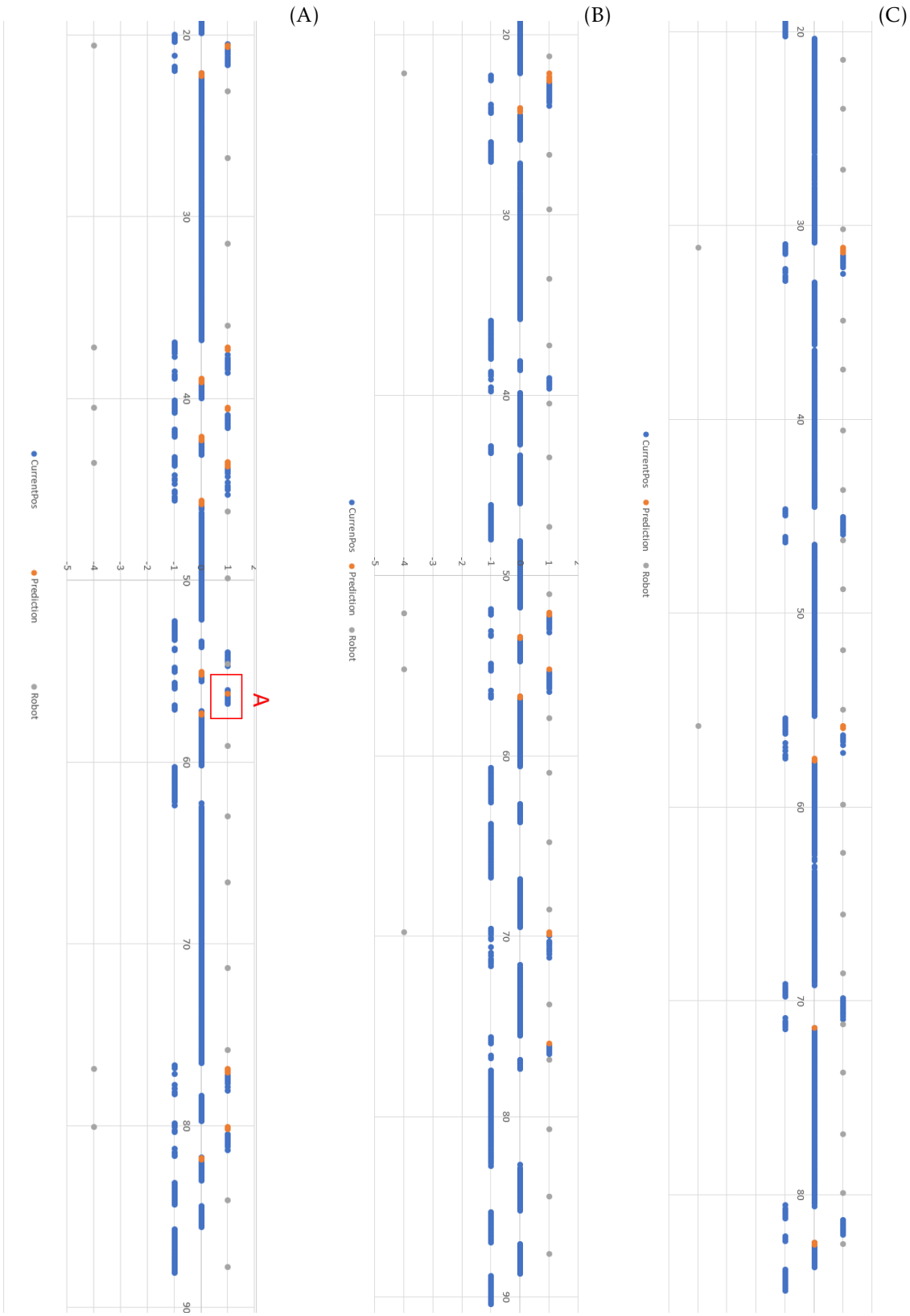


Figure 5.23: Temporal evolution of the current hand key-area (blue), prediction (orange) and robot feedback (grey) for test 1. The robot speed for each test is as follows: A - 50%, B - 60%, C - 75%.

5.4.2 Test 2 - Two Critical Areas and robot speed at 50%

Figure 5.24: Setup used in tests 2 and 3. This image is not to scale.

Contrary to test 1, where there was only one possible target from either one of the key-areas, on tests two and three the system must establish the difference between trajectories with the same origin, specifically N → CA1 and N → CA2, where N represents the neutral key-area and CA1 and CA2 represent the two different critical areas. During the tests with more than one critical areas, the trajectories between critical (CA1 ↔ CA2) areas were not used as the spatial separation was too small thus impeding the system from making any useful predictions due to time constraints.

In addition to the problems present in test 1, in this test further errors occurred, as expected, due to the system not being able to distinguish between the two targets until a later stage of the trajectory. This can be seen particularly in 5.25 - B and 5.25 - C. As an example of late and wrong predictions refer to figure 5.25 - B (A) and (B) respectively. The variation observed between tests within the same layout reinforce the claim made in the beginning of this chapter on the dependence of the quality of the trajectories used during the initialisation phase.

In figure 5.25 - B it is possible to observe that the system produces better predictions for CA2 that CA1, in fact most of the predictions for CA1 arrived after the user has entered the area, whereas CA2 predictions actually arrive at a useful time. This resulted in the robot stopping when collisions were predicted at CA2, but either not stop at all at CA1 or stop too late, given that the robot does not stop instantaneously upon receiving the stop

signal.

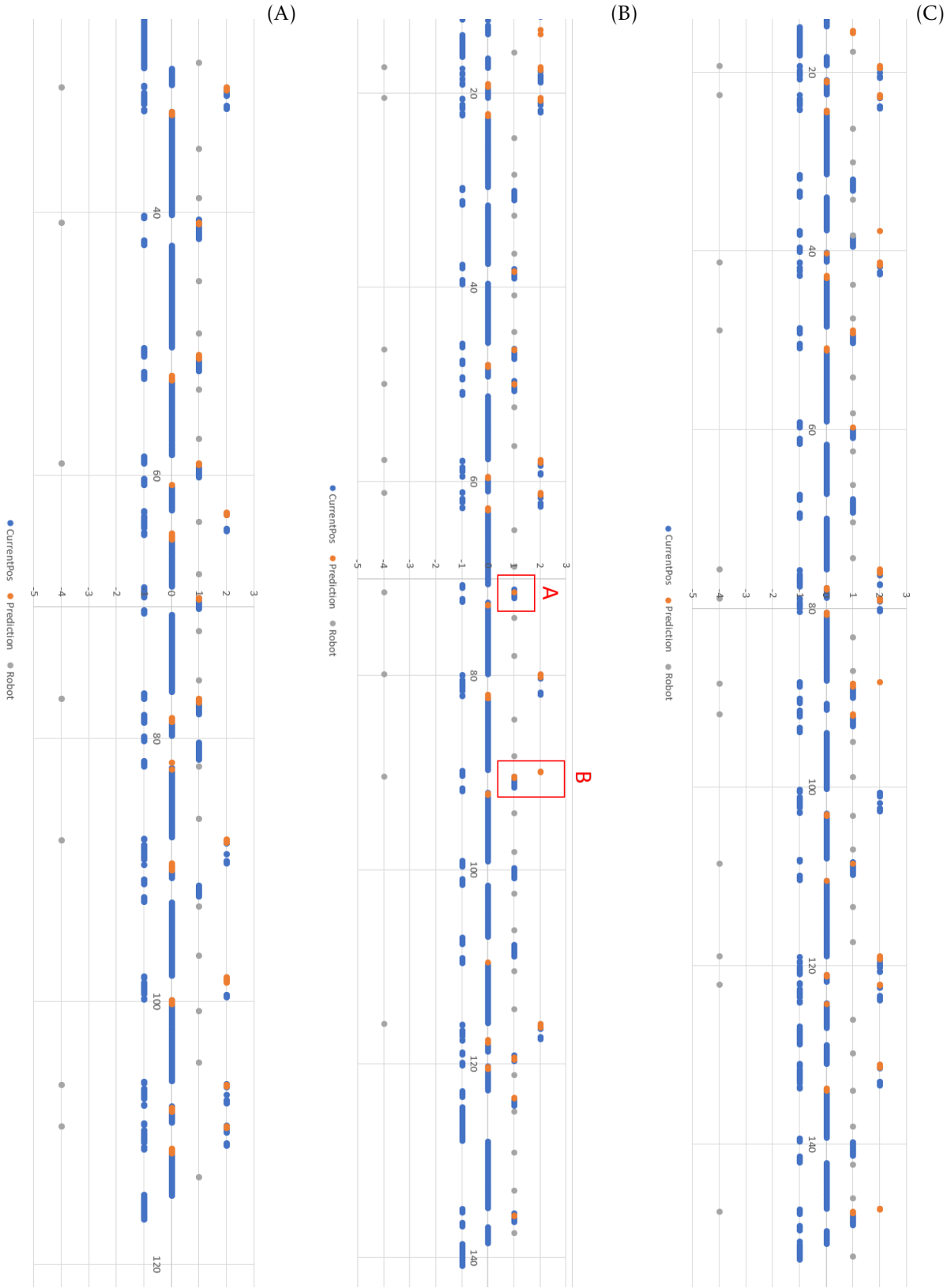


Figure 5.25: Temporal evolution of the current hand key-area (blue), prediction (orange) and robot feedback (grey) for test 2. During these three tests the robot speed was kept at 50%.

5.4.3 Test 3 - Two Critical Areas and robot speed at 60%

As the topology of the key-areas did not change from test 2, the problems present in this test are the same as in that test. However, all the tests were kept in order to cover the most possibilities regarding the initialisation trajectories. The use of higher movement speed for the robot, on the other hand, does aggravate the late prediction problem for two main reasons, the first is the reduced time interval available to get a valid prediction. The second problem relates to the time the robot takes to stop after receiving the stop command.

Another prominent problem, which can be observed, for example, in 5.26 - B (A), where a wrong prediction caused the robot to stop unnecessarily. This phenomenon can be observed several times, particularly in 5.26 - B and 5.26 - C. Because the system only the signals the robot to stop if the predicted critical area corresponds to the robot current destination, a wrong prediction does not necessary stops the robot, this can be observed in 5.26 - B (B).

5.4.4 Tests 4 and 5 - Three Critical Areas

These two final tests use a similar setup to the one used in test 1, with three different critical areas. Similarly to what happened when the second critical area was introduced in tests 2 and 3, the addition of the third critical area increased the incidence of the problems already discussed up to this point. In figures 5.28 and 5.29, it is possible to observe that that the number of missed predictions increases significantly with addition of the third critical area and, once again, the importance of the initialisation process. The difference between both tests is the speed of the robot, where the robot's speed is set to 50% for test 4 and 60% in test 5.

By examining the data presented in the plots for each test, one can observe the situations when the system makes a correct or incorrect prediction, when the prediction is published after the user's hand has entered a certain key-area, or when it fails to produce a prediction.

For simplicity, examples of these three types of events have been labelled on all three tests represented in both figure 5.28 and 5.29. On these two figures, events labelled as A represent a wrong prediction, in events marked as B the system did not make a prediction and C marks situations where the prediction was published after the user's hand entered the key-area. It is noteworthy to state that the label used for each type of event on figures 5.28 and 5.29 is not necessarily the same as the one used to describe analogous events on previous plots. Furthermore, these figures illustrate some examples of the situations described throughout this chapter and not every event was labelled, as it would jeopardise the readability of the plots.

Finally, although not possible to directly determine the distance from the target when the robot stops, one can estimate the distance based on the time difference between a stop signal (-4) and a target reached signal (1). Since during these tests the robot did not make

a emergency stop, the only sources of stop signals result from the stop command being issued.

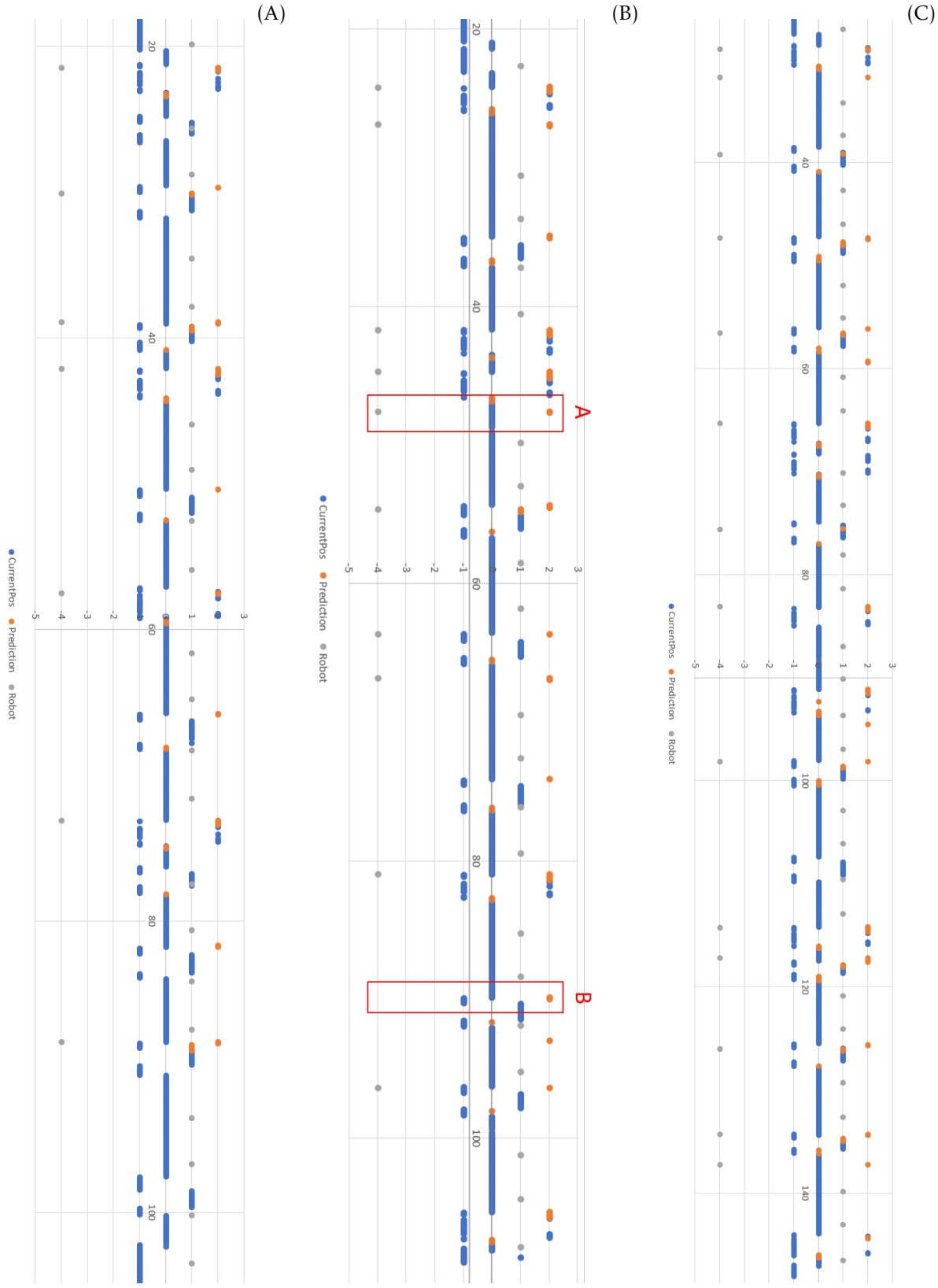


Figure 5.26: Temporal evolution of the current hand key-area (blue), prediction (orange) and robot feedback (grey) for test 3. The robot speed was kept at 60%.

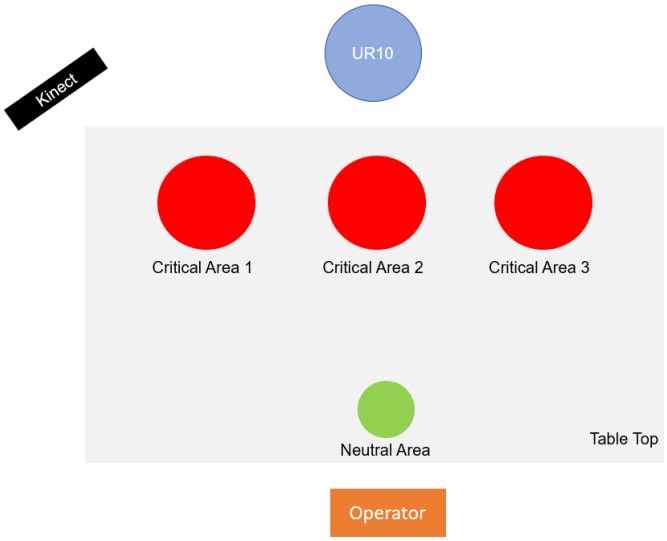


Figure 5.27: Setup used in tests 4 and 5. This image is not to scale.

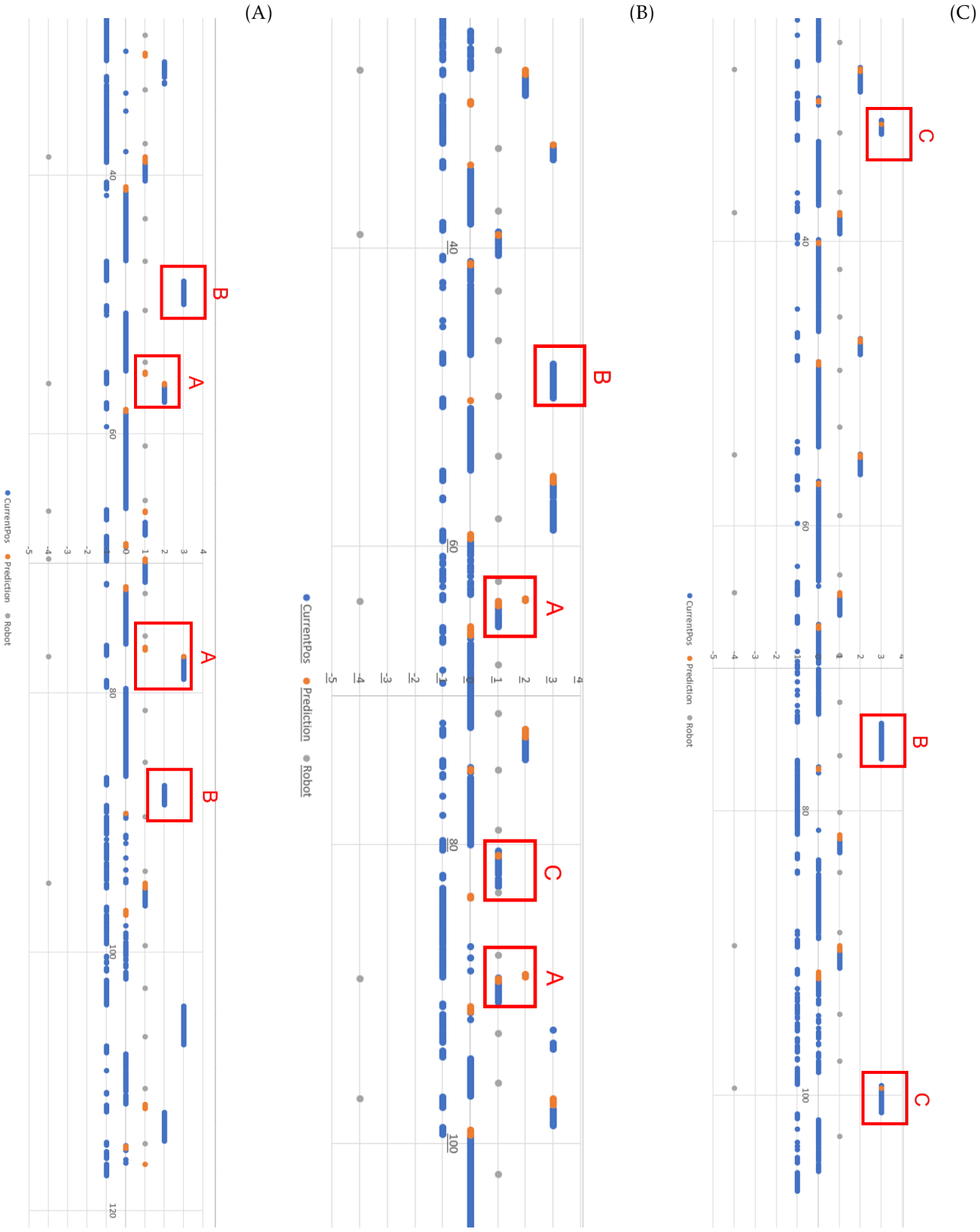


Figure 5.28: Temporal evolution of the current hand key-area (blue), prediction (orange) and robot feedback (grey) for test 4, with the robot speed set to 50%

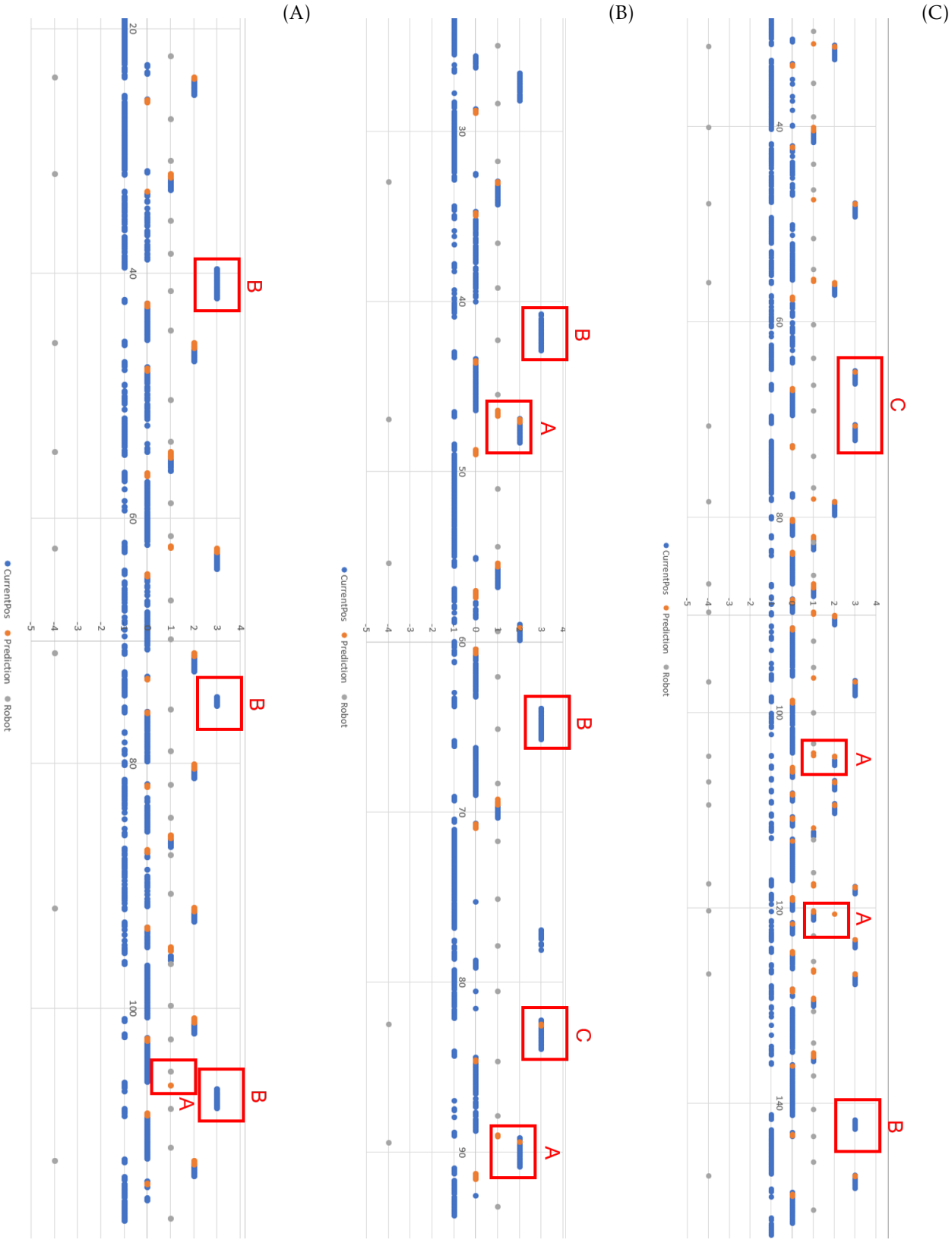


Figure 5.29: Temporal evolution of the current hand key-area (blue), prediction (orange) and robot feedback (grey). for test 5, with the robot speed set to 60%

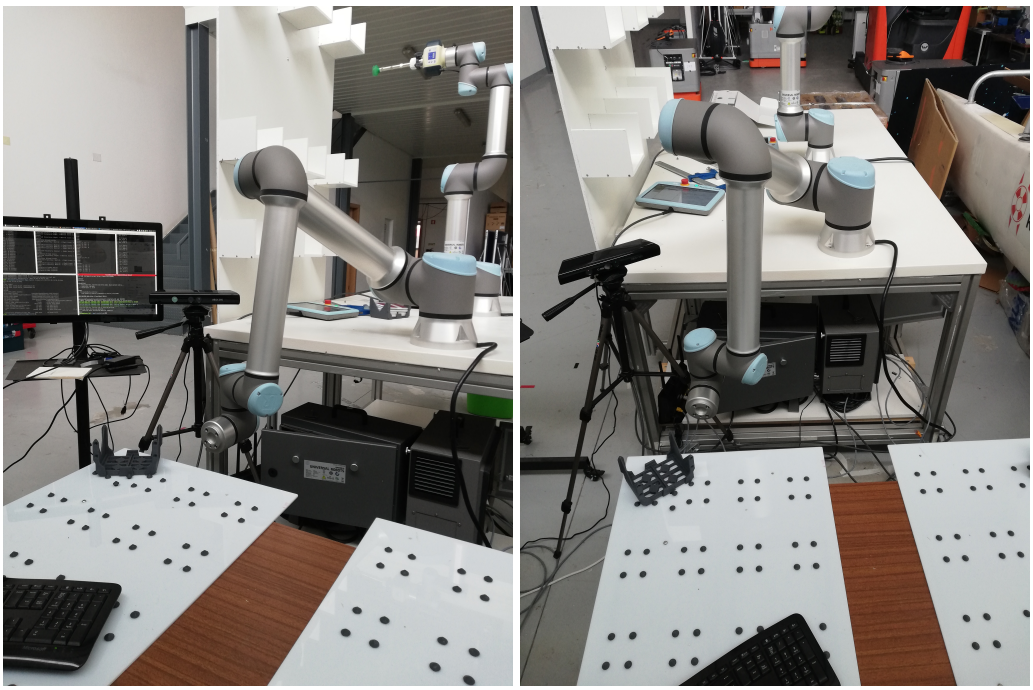


Figure 5.30: Setup used during the final tests presented in this section. The kinect sensor and robot are visible on both images.

5.4.5 Simulated Collaborative Task

This section will present the results obtained by simulating an industrial setting where a human operator and a robot need to share the work-space. This section uses a more realistic robot task, instead of only making motions intended to force the robot into situations where collisions could occur.

This simulated task consists on a collaborative industrial process where a part of said process needs to be performed by a human operator. In this case human operator's part of the process is performing a quality control evaluation of some products and the robot's main task is the transportation of each product in the environment. The layout of this simulated task is schematically represented in Figure 5.31. Each product is assumed to enter in the *Product In* zone, present in Figure 5.31, and exits in the *Product Out* represented in the same image. Furthermore, this collaborative task also defines two zones referred to as *Quality Control 1* and *Quality Control 2* which are divided in two sub-areas *In* and *Out*.

A UR10 robot was used to transport the products between the areas. The robot may pickup a product from the entry point and place it in the exit point or, alternatively, the robot may place the product in one of the quality control entry areas. The human operator removes the product from the entry area, performs the quality control process and places the product in one of the quality control exit areas. The robot then picks the product up from the quality control area and places it on the exit point. The layout of the operator's work space in the real setup is represented in Figure 5.32, where the two Quality Control

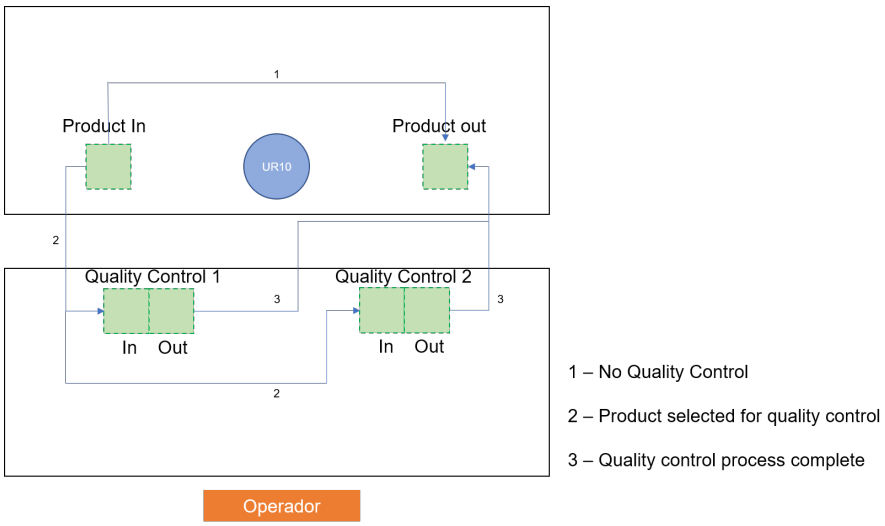


Figure 5.31: Layout of the areas of interest for the simulated collaborative task and the possible ways the products can be transported depending on the state of the task.

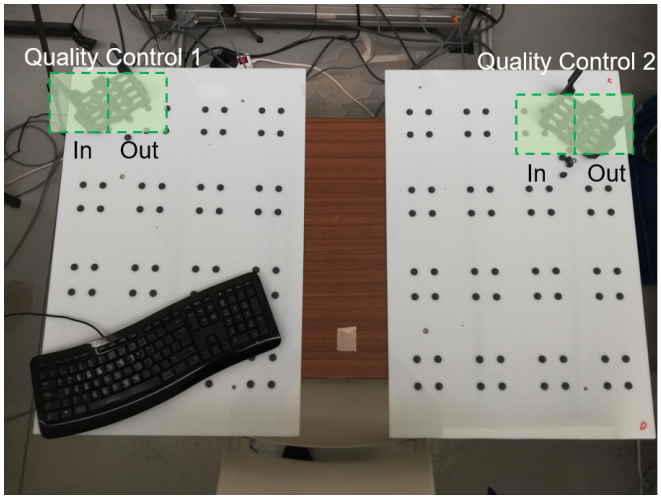


Figure 5.32: Real setup used in the simulated collaborative task. The locations highlighted in green represent the quality control input/output areas.

areas are overlaid over the real setup.

Finally, it is noteworthy to state that, since this task intends to be simulated no products were actually moved during the task's execution. Therefore, the entry and exit points do not change overtime due to product accumulation, as it would add complexity to the process which would not affect the behaviour of the proposed system.

A task supervisor node, used by the research group for this type of operation, had to be added to the system which deals with the high level details of the task. This node assigns a numerical value to each of the relevant positions, more specifically the value zero specifies the product entry point, one represents the product exit area whilst two, three, four and five represent the quality control input and output points for both quality control areas, respectively.

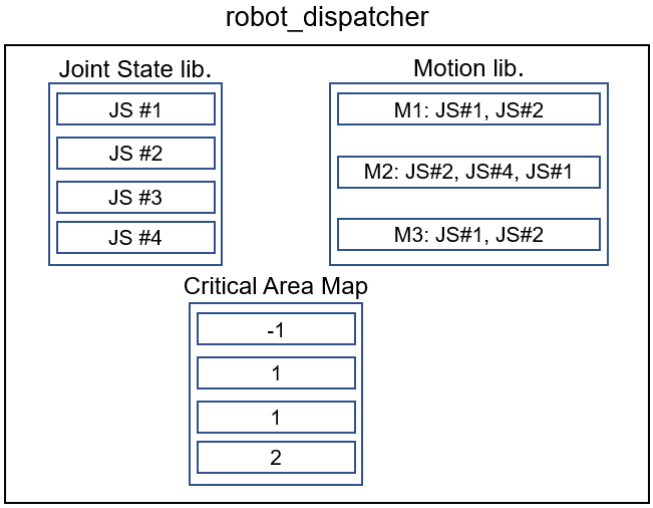


Figure 5.33: Information stored in the ***robot_dispatcher*** node used to control the robot movement and integrate the prediction information.

This node also maintains a record of the occupancy status of each of the quality control input and output points and decides when to move a product directly from product entry to product exit (without going through the quality control process), or when a product should be subjected to the quality control procedure. In order to keep the occupancy information of the supervisor node, the human operator signals the system when a product has been placed or removed from one of the quality control sub-areas. The opportunity for the proposed system to intervene arises in the quality control points, where the human and the robot might try to access those areas simultaneously.

It is noteworthy to state that in order to integrate the supervisor node with the rest of the proposed system, some changes had to be made to the ***robot_dispatcher*** node. The interaction with the prediction information was not altered, specifically, instead of having a task programmed directly in the ***robot_dispatcher*** node, it now keeps a library of key joint-states and a library of motions. Each joint-state is an array representation of the state of each robot joint. The joint-state library is a collection of joint states which places the robot in a particular pose relevant for the task being developed. Each motion is a sequence of joint-states which the robot should follow to perform a specific motion. Finally, the ***robot_dispatcher*** also keeps a critical area map, which relates each joint-state in the joint-state library with a specific critical area of the system. The value -1 is used to signal that the correspondent joint-state does not access the critical area. The latter uses the positions stored in the joint-state library to define motions.

The supervisor issues movement commands with a source and destination location index, which is used to select the motions to be executed. An illustrative example is represented in figure 5.33.

The execution of these tests was performed similarly to the previous tests. An initialisation phase occurs when the user performs the necessary trajectories in order to populate

Tests	N→ Right (cm)	N→ Left (cm)	Right↔ Left (cm)
1 and 2	68	68	89
3 and 4	49	68	61
5 and 6	49	54	38

Table 5.5: Distances between the neutral area (N) and the right and left critical areas.

the GMM library. During the initialisation process and the robot does not execute any motions. Once the system has been initialised the robot supervisor node is launched and starts issuing commands to the robot.

In order to record each joint state, the freedrive mode was used to place the robot in the desired position by hand and the joint state of the robot was saved to a file and the robot speed was set to 60% of the maximum speed on the Teach Pendant (TP). Once all the relevant joint states were recorded, the joint information was used in the ***robot_dispatcher*** node.

In total, the system was tested with three different critical area locations. For each critical area layout the experiment was performed twice and the speed setting on the TP was kept constant throughout the whole testing process.

Although, in the previous section, the tests were represented as the raw information from each topic, the longer duration of the final tests poses a constraint in the graphical representation of the final tests using the same method. Therefore, some processing of the data took place in order to accommodate the relevant information whilst improving the readability of the data representation. Namely, the plots only represent the portion of the test where there are events regarding predictions, each current position data point is only represented when there is a change in the current position, rather than continuously representing that information. Each robot feedback data point now represents that the robot stopped.

5.4.6 Experiments

5.4.6.1 Experimental Setup

The first step in performing these tests was the creation of the trajectories. Once all the trajectories were created the robot was started and kept repeating the same task until the end of the test, the user would try to either force the robot to stop by attempting to reach for the same area as the robot at the same time, or go to a different area in order to evaluate the capacity to distinguish between trajectories to close key-areas.

The robot's task for each test was created using the freedrive feature of the UR10 robot which allows the operator to place the robot in the desired position by hand, and using a custom auxiliary program used to save each position of the task to a file. That information was then used in the ***robot_dispatcher*** node to execute the task. As for the robot program, the only instruction in the script was the one provided by the external control urcap.

Finally, the data was processed using the rostopic echo tool in order to transcribe the messages from the rosbag file into csv files, one for each type of message. A Python script then processed every set of files in order to join all the data in a single file, and use the first message time-stamp as reference for the following messages time-stamps. Therefore the time axis values are, in fact, the time offsets from the first message. The plots were then created using Microsoft Excel. It is also important to note that due to the large duration disparity between each test and single trajectories, the plots for each test had to be scaled in such a way which allowed the correct representation of both of those time scales, thus it was necessary to crop the plots so that they could fit in a single page. For this reason, the section of the parts of the plots regarding the initialisation phase have been cropped out of the figures, and the beginning of the plot refers to the point where the robot started the task execution.

5.4.6.2 Large Horizontal Separation - Final Test 1 and 2

Following the results presented in the previous section, the first test performed on the system represents a situation with a large horizontal separation between the critical areas. The layout used for these tests is illustrated in Figures 5.34 and 5.35. In figure 5.36 is the temporal representation of the main events that occurred during both tests in the first situation. As can be observed in this figure, in particular in 5.36 - A, the system predicted the destination for most motions. However, in some instances the prediction was published later than would be desirable. The points in Figure 5.36 - A (A) are two examples of such situation. It is also noteworthy to mention the data point in Figure 5.36 - A (B), which represents the user entering area 2 without a prediction. As for test 2 (Figure 5.36 - B), contrary to the first test, where the number of data points in the "Current Pos" data series reflects the logic used to decrease the number of data points on the plot. In test 2, there is a considerable larger collection of points, exclusively for position 0. As stated before the data regarding the area in which the user hand is in was filtered by keeping only the data points where the position changes. By examining the original raw data it was possible to observe the data alternating between 0 (neutral area) and -1 (unknown). This is most likely due to the hand being placed near the border of the neutral area. Aside from this fact, it is possible to observe that during this second test the system made some late prediction similarly to what happened in the previous test. However, there were not instances where the robot had stopped incorrectly, nor wrong predictions. This results are inline with the previous presented results, the difference in performance between these two tests can, once again, be explained by the different initialisation phases.

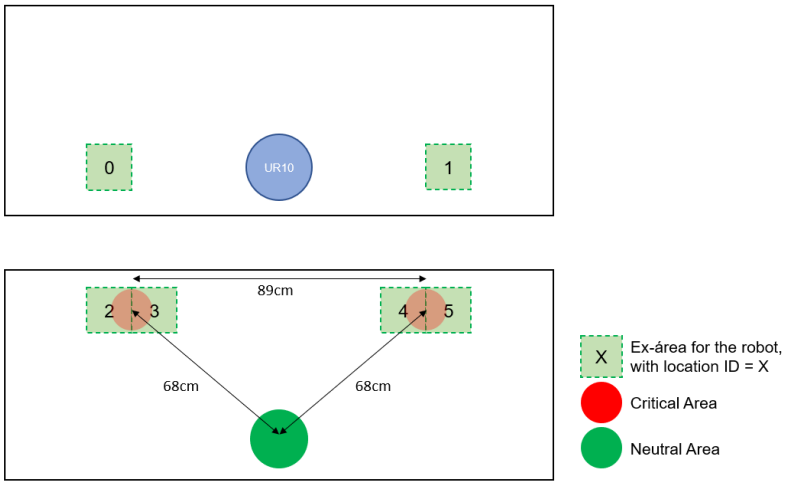


Figure 5.34: Layout of the several system components used during tests 1 and 2. This image is not to scale.

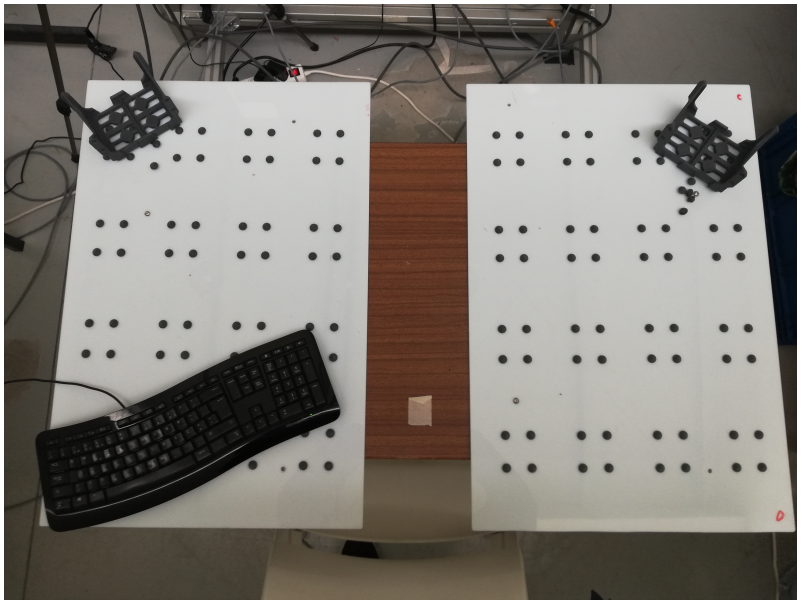


Figure 5.35: Actual layout used during tests 1 and 2. Each plastic support represents a quality control in/out.

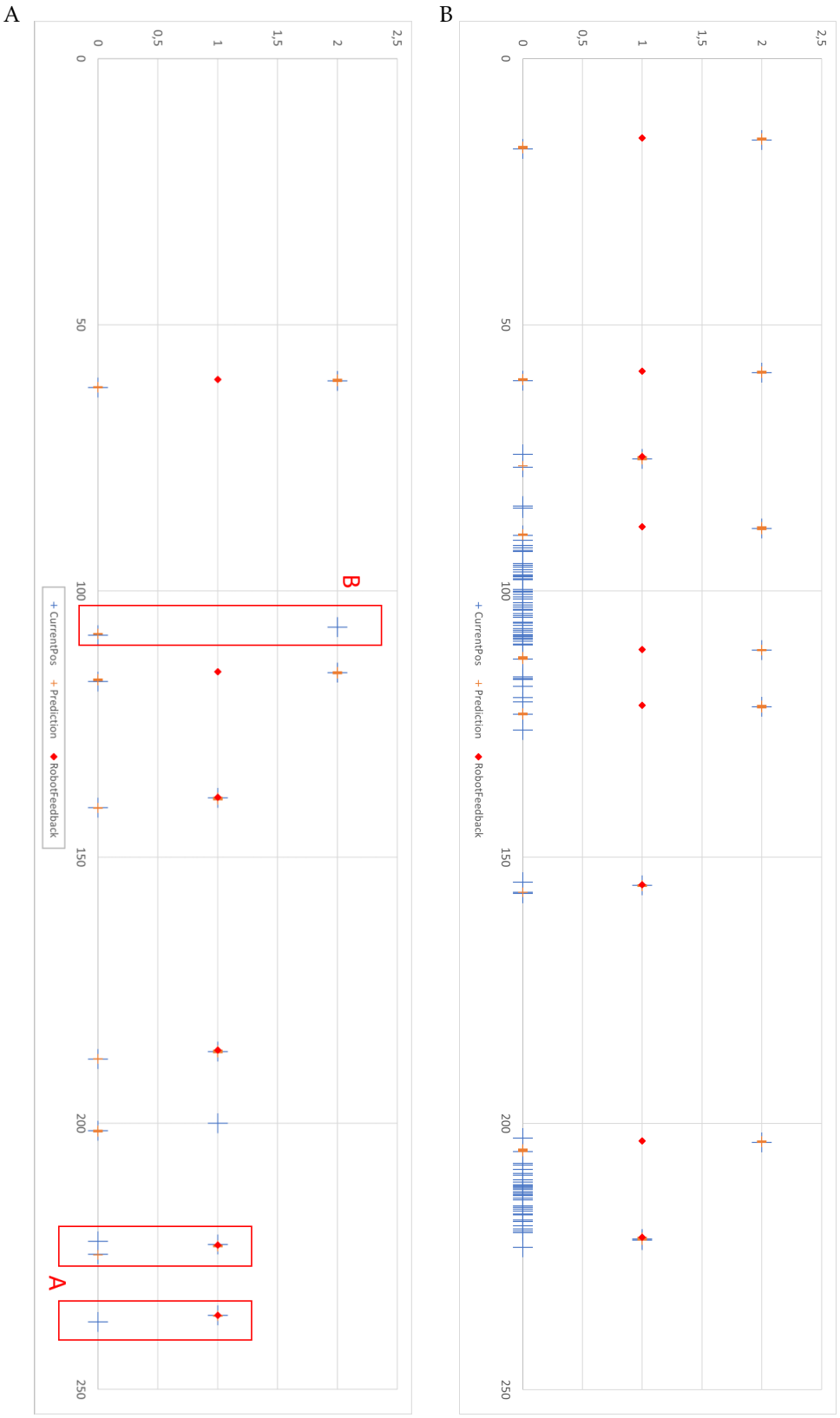


Figure 5.36: Temporal representation of the current key-area occupied by the user (blue), predictions (orange) and robot stop signals (red) for test 1 - A and 2 - B.

5.4.6.3 Medium Horizontal Separation - Tests 3 and 4

For these two tests the horizontal separation between the two critical areas was decreased, as can be observed in figure 5.37 and 5.38, and this change had clear effects in the performance of the system. Namely, the number of wrong predictions. One can observe in Figure 5.39 - A several occurrences of predictions where the actual position did not end up in the critical area, rather a new prediction was made later to the correct position, concretely the predictions made in Figure 5.39 - A (C) are examples of such events.

Although the system does make wrong predictions and, as consequence, signals the robot to stop unnecessarily, the system can make the correct prediction by observing more of the trajectory being performed by the human operator, those cases can also be observed in Figure 5.39 - A (D), where the system does actually makes the correct prediction, albeit later than ideal. Specifically, for test 4 (Figure 5.39 - B), the performance was similar to test 3, with a significant event occurring in Figure 5.39 - B (E), where three stop signals were sent. The first of these three events actually represents a correct stop signal. As for the second, the user had already removed the hand from critical area 1, this was likely due to the fact that the hand locations from the previous motion had not yet been cleared, because the time-to-live had not yet expired. As for the third, it represents an incorrect prediction, the system observed part of the trajectory and predicted the wrong critical area, this caused the robot to stop once again.

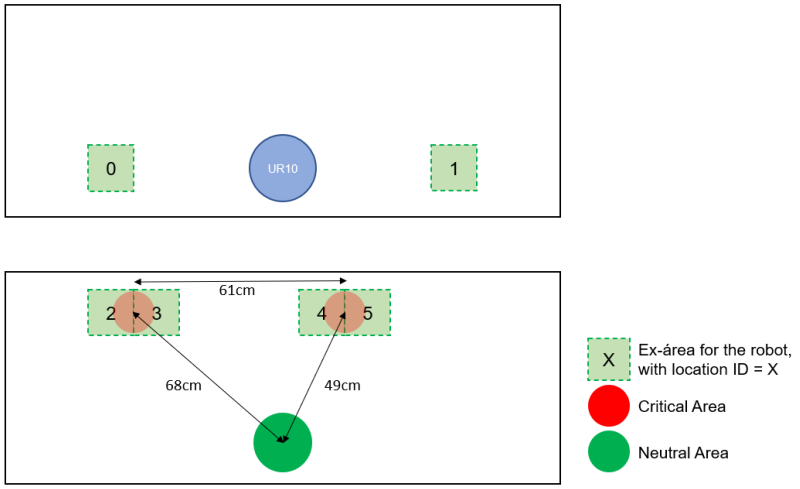


Figure 5.37: Layout of the several system components used during tests 3 and 4. This image is not to scale.

5.4.6.4 Short Horizontal Separation - Tests 5 and 6

Finally, regarding the final situation, the horizontal separation between the two critical areas was further decreased resulting in the layout represented in figure 5.40 and 5.41. Two tests were performed following the same procedure as the the one used in previous presented tests. The temporal evolution of test 5 and 6 are represented in figure 5.42

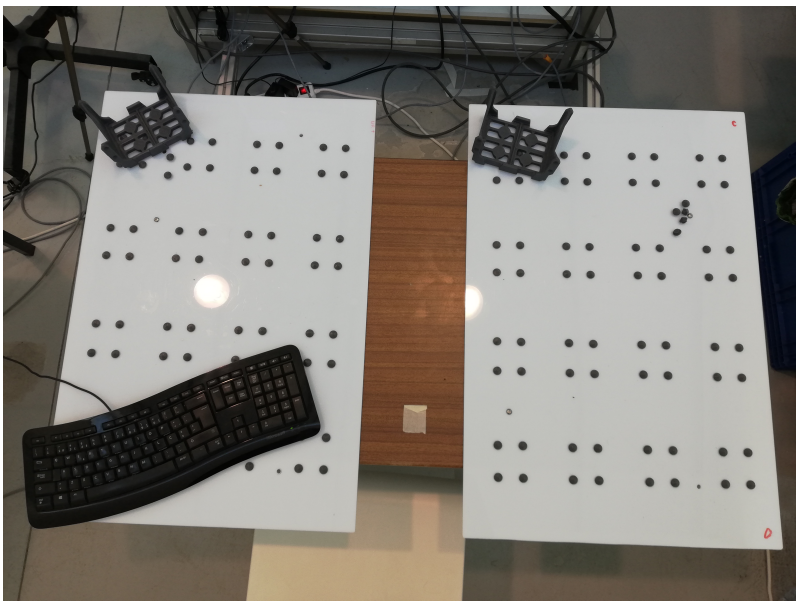


Figure 5.38: Actual layout used during tests 3 and 4. Each plastic support represents a quality control in/out.

A and B, respectively. The results obtained in these tests matches the expected result, following the relocation of the critical areas to be much closer than in tests 1 and 2, namely the number of wrong and late predictions.

5.5 Result Discussion

The results shown in this chapter, specifically for the large horizontal separation reflect the ability of system to make predictions about the destination of a certain motion given part of that motion. It is also possible to observe from the tests that the predictions are made, in most situations, before the user's hand enter the destination key-area, at the speeds used during testing.

However, during the testing phase the the system also showed some factors which can impact the performance of the system. Namely, the initialisation phase, the speed of the robot and the distances between critical areas.

The impact of the initialisation phase can be observed by the different results obtained within the same testing setup. As stated throughout this chapter, some factors might influence the quality of the position that constitute the trajectory for a particular motion between two key-areas, when this happens during the initialisation phase the GMM for that particular motion is fitted to faulty data. This, in turn, results in sub-optimal prediction performance, not only for that particular motion but for the overall prediction process as the GMM scores for each motion are compared in order to choose the best prediction for the data.

Even though this approach has this limitation it also provides a way to use the system without the need for an offline training phase. With this approach, ideally the human

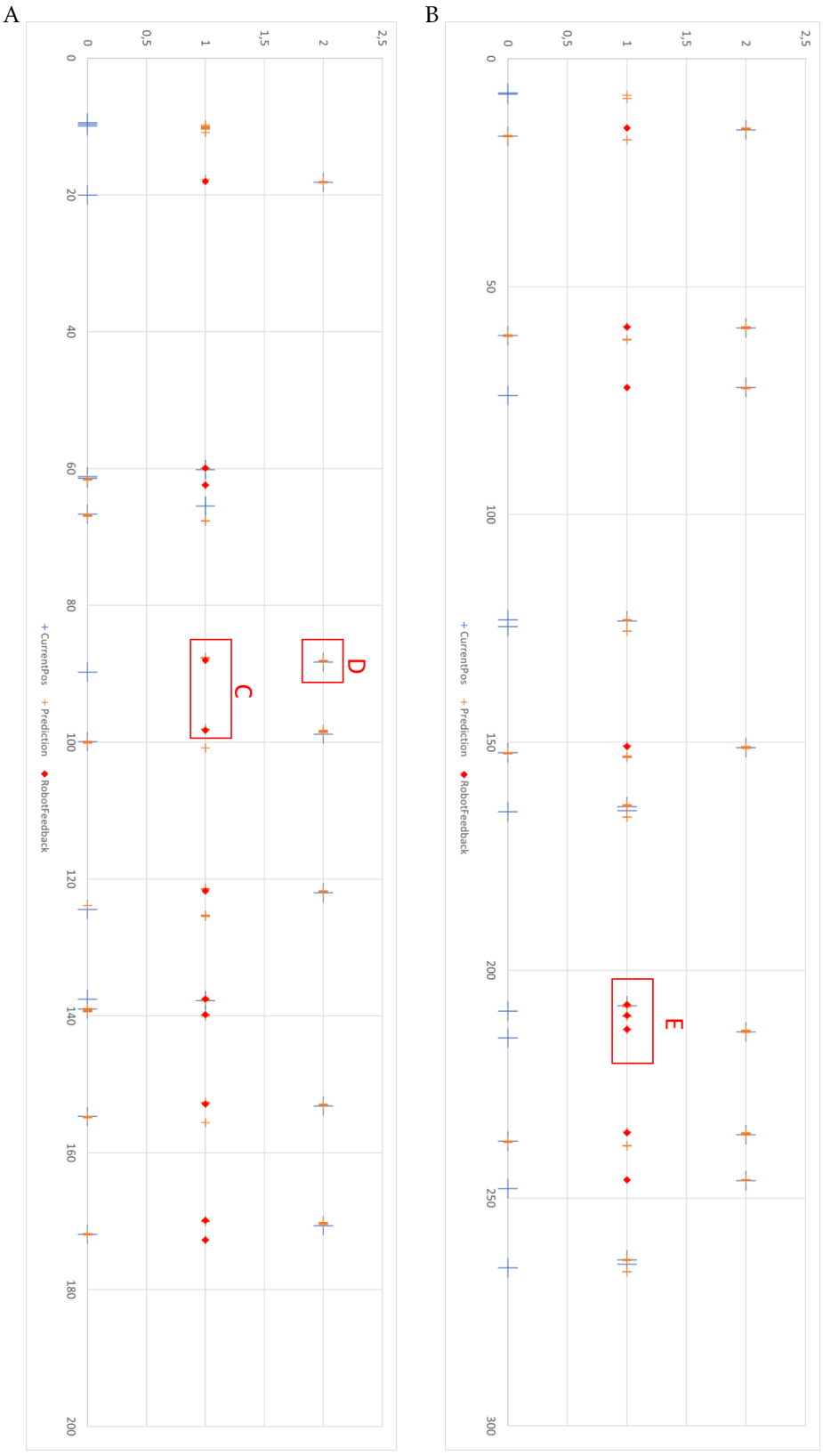


Figure 5.39: Temporal representation of the current key-area occupied by the user (blue), predictions (orange) and robot stop signals (red) for test 3 - A and 4 - B.

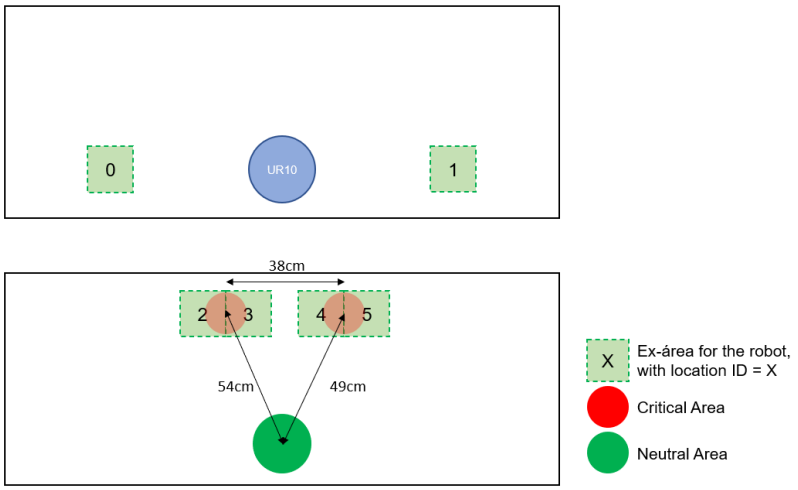


Figure 5.40: Layout of the several system components used during tests 3 and 4. This image is not to scale.

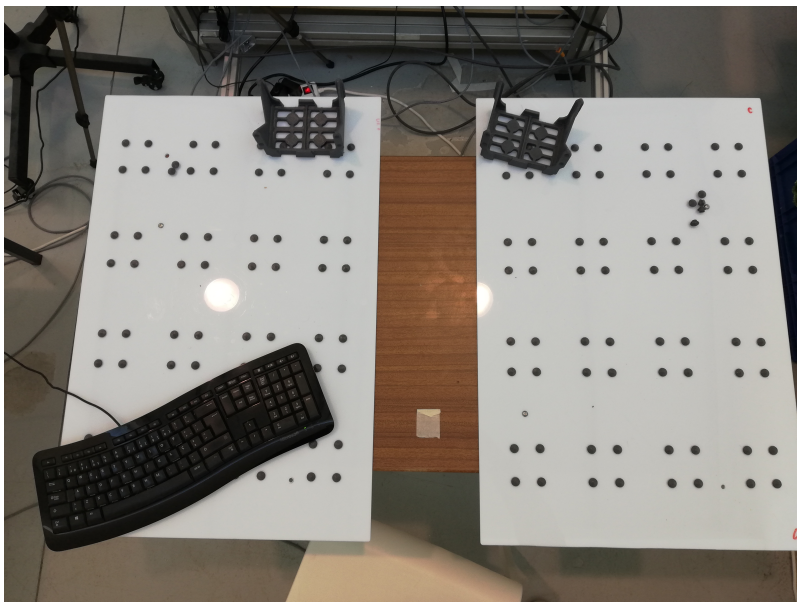


Figure 5.41: Actual layout used during tests 5 and 6. Each plastic support represents a quality control in/out.

operator would be able to perform the motions once and the system would be able to recognise that particular motion, or new motions could be added on the fly, if a specific key-area had been previously defined for that purpose.

Additionally, the distance between the different key-areas is also an important factor in the system's performance as it is possible to observe on both the tests with three critical areas and the simulated collaborative task. Which show the decrease in performance with the decrease of the horizontal separation between key-areas.

Although not referred during the tests, relative positioning of the key-areas also plays an important role on the system's ability to make correct predictions. For example, the

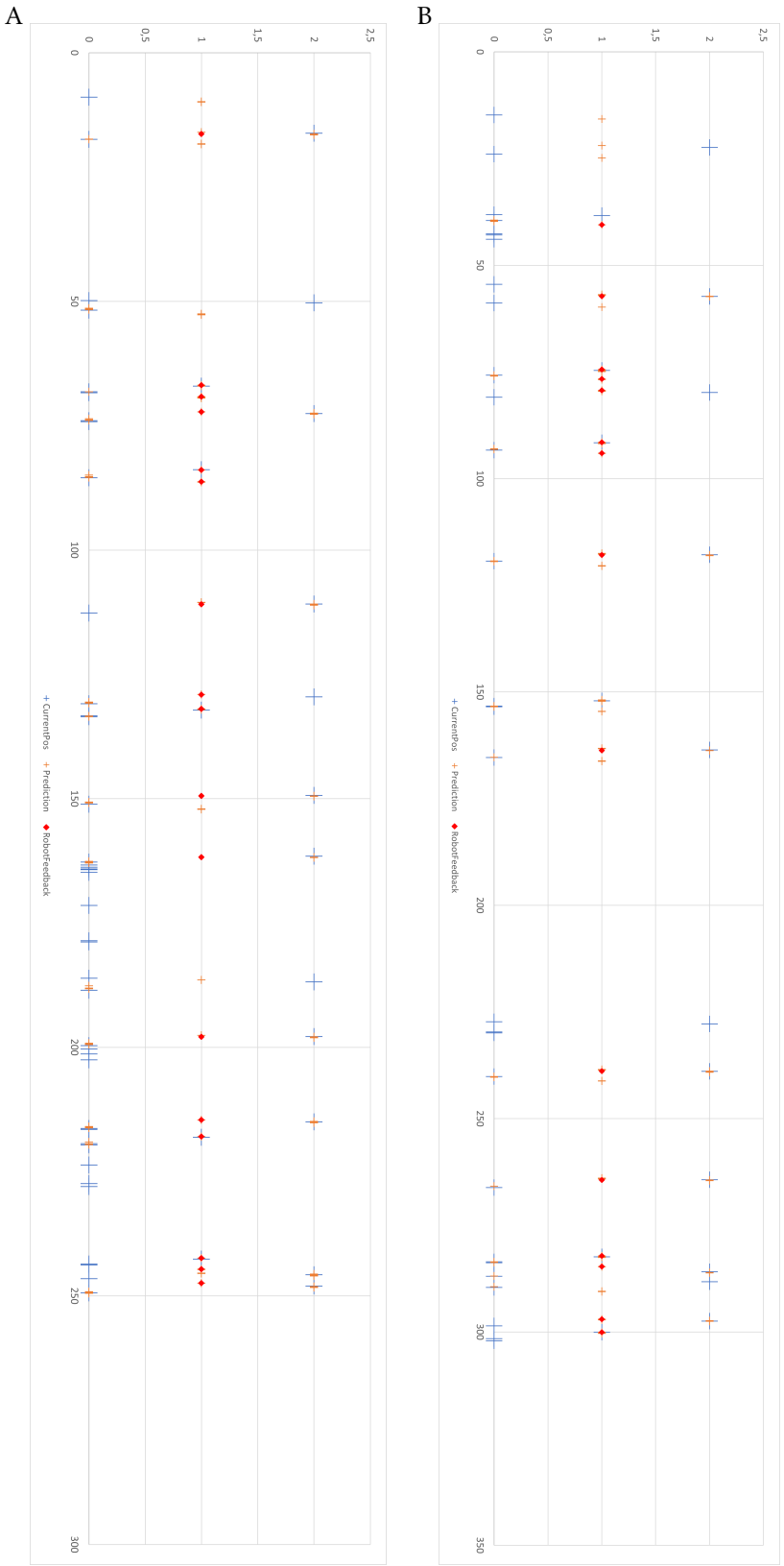


Figure 5.42: Temporal representation of the current key-area occupied by the user (blue), predictions (orange) and robot stop signals (red) for test 5 - A and 6 - B.

system might not perform well if three or more key-areas are placed co-linearly, or if two or more key-areas intersect each other.

The final obstacle to this approach is the speed at which the motions are performed, either by the robot or by the human operator. In the robot's case the speed of motion influences both the time it takes to stop as well as the time available to the system to make a correct prediction.

In the operator's case on the other hand, if the motion is performed too quickly the system will not be able to acquire enough data points in order to accurately represent the trajectory performed by the user's hand. This problem could, perhaps, be mitigated by running the system on either more than one machine, or running it on a single more powerful machine, which would be able to process more frames. This problem also aggravates the problems caused by a high robot speed.

Chapter 6

Conclusion

Interaction between humans and robots is a widely researched topic, with several unsolved problems. Although current collaborative robots, such as the one used in this thesis, are capable of stopping after hitting an object or an operator, the interaction with the robot could be further improved if the robot were able to stop without requiring physical contact.

This dissertation proposed a possible system to improve the interaction in an industrial environment, which can be used as an add-on to the current capabilities of collaborative robots in order to stop the robot when a possible collision is detected.

The method proposed in this dissertation shows the ability to make prediction regarding the destination of a particular motion, within certain parameters and to use that information to change the behaviour of the robot. It is worthwhile remembering what was stated in the Proposed Approach chapter regarding safety issues in collaborative situations. Collaborative robots, such as the one used in this dissertation, are designed with the intent of being used in close proximity of humans, and, in order to do it safely, implement several safety features to minimise the risk to the human operators, such as the automatic stop when the robot detects a collision. One more important factor to keep in mind regarding the safety of collaborative robots is the end-effector attached to it, as it may also pose a threat to human workers and should, therefore, be taken into consideration in collaborative situations.

With these facts in mind, although the robot can stop due to a collision, it would, arguably, be more comfortable, and perhaps natural, to the human operators if the robot could prevent the collisions from happening. This is where this dissertation intends to build upon. By giving the robot the ability to make predictions regarding the motion of the operator. Which is accomplished by the proposed approach, within the expected level of performance.

However, as it can be observed in the results presented in the previous chapter, this system does present itself with some limitations. Namely, the fact that the system is dependent on the detected trajectories during the initialisation phase, therefore if the initialisation phase does not get good quality trajectories, either because the system could not get the correct hand location, both in 2D or 3D, or because the system lost frames due to the synchronisation mechanism, it will not perform well during the entire time the system is running, the opposite being true as well. Therefore, an addition to this approach could be a method similar to the one presented in [32] which allows the system to adapt over time to less favourable initial conditions. This problem could be further mitigated by using different sensors which allows for better modelling of the trajectories.

The speed of the robot also plays a major role in the performance of the system. In a situation where the trajectories got correctly initialised, the system cannot make correct predictions in time to stop the robot before it reaches the target in time if the speed is too high, given the extremely narrow prediction window that high speeds impose on the system. This is the reason why the full tests in the previous chapter were made at a maximum of 75% speed for the single critical area test and 60% for the rest. These speeds allowed the system to respond at a reasonable time to stop the robot before collisions. In addition, the fact that the speed of the robot was kept lower is also necessary due to safety concerns as the robot should operate at lower speeds when working in close proximity to a human worker. It is, however, important to note that the time duration of each reaching motion is around one to two seconds, if the robot executes at full speed, besides the fact that the amount of time to make predictions decreases, it also takes longer to stop, reducing the reaction time even further.

Another fact that showed to have a significant impact on performance is the speed at which the reaching motion is performed by the operator. It is possible to observe in the last chapters that due to the processing time of the neural network many of the frames captured by the Kinect sensor end up being discarded as the system is still processing a previous frame. This problem could eventually be mitigated by using a more powerful machine than the one used during the tests.

The tests showed that, with a single critical area, the system manages to predict when the user is performing a trajectory that will likely end up at the critical area, whilst not stopping the robot while the user performs other motions and can, for the most part, distinguish between the different trajectories, even though during the tests presented in the previous chapter, does seem to get better performance for one direction than the other, on situations with two critical areas. As for the situation with three critical areas, the system needs some extra time to distinguish the between the trajectories, as the user needs to move the hand further in order to decide where the destination will be.

With these considerations in mind, the system could benefit from a different way of data acquisition. Replacing the proposed data acquisition method which does rely mainly on RGB frame processing, other methods could be used for detecting the user's hand in 3D space, that could provide more precise hand location as well stream that data faster,

which would allow for more accurate representation of the trajectory performed by the user's hand.

The robot is controlled using the MoveIt framework, however it is also possible to control the robot without it, using the robot's program to control the motion of the robot through external controls directly to the the robot's controller, as if they were issued by the teach pendent. This could prove to be faster than the MoveIt commands, however this hypothesis was not tested.

One final consideration on the proposed approach, which deals more with logistic operations necessary to the operations of this method, which would benefit from a method of setting up the necessary data, namely the position of the critical areas, which at the time of writing, needed to be setup manually by placing the hand on the desired positions on the table top, setting those positions manually on the code and then rebuilding the program. Also, a more flexible way of setting the poses of the robot. Although an auxiliary program was created in order to save the key-locations to a file, that information had to be input to the ***robot_dispatcher*** node by hand, and then rebuilding the program. The existence of methods which could expedite this processes, which at the time of writing were not implemented, would make this system more easily deployed in different situations.

6.1 Future Work

One of the drawbacks of the proposed approach is the perception system, which relies on a single depth sensor (Microsoft Kinect in this case) to retrieve the required data. This makes the system susceptible to problems such as occlusion. One possible solution to this problem could be the use of multiple depth sensors and fuse the data gathered by each one in order to improve the reliability of the data.

However, this approach would necessarily increase the processing power needed to run the system. Another approach would be to use different skeletal information retrieval techniques.

Regarding the robot control, this dissertation relied on the Moveit framework to interface with the robot alongside the driver ROS node. However, the UR10 robot allows the robot to communicate with other machines through other mechanisms which issue commands as if they were issued by the robot's TP, this way the robot task could be programmed in the TP, as it is usually done and the system issues pause and resume commands to the robot. This method of control could be more responsive than the one described in this document.

Furthermore, in this dissertation the robot was only halted when a collision occurred. It could also be beneficial to change the plan of the robot when a collision is forecast. For example, if the system predicts that the user will enter a particular critical area, the robot may adjust its trajectory on the fly and move to the other critical area, thus avoiding the conflict without the need to stop the robot.

Finally, as stated above, at the current state this system is not easily deployed as the programs need to be recompiled in order to accommodate changes, either in critical area location, joint states for the robot and joint-state-critical area map. A tool which allowed the operator to describe all these parameters in an XML file, for example, would allow for an easier deployment of the system.

Bibliography

[1] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh. "OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields." In: *arXiv preprint arXiv:1812.08008* (2018).

[2] Y. Kong and Y. Fu. "Human action recognition and prediction: A survey." In: *arXiv preprint arXiv:1806.11230* (2018).

[3] *A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way*. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. Accessed: 2020-10-29.

[4] *Gaussian Mixture Model. Brilliant.org*. https://brilliant.org/wiki/gaussian-mixture-model/#references. Accessed: 2020-10-30.

[5] *Technical description of Kinect calibration*. https://wiki.ros.org/kinect_calibration/technical.

[6] *IFR publishes collaborative industrial robot definition and estimates supply*. https://ifr.org/post/international-federation-of-robotics-publishes-collaborative-industrial-rob. Accessed: 2020.

[7] Y. Cai, H. Li, J.-F. Hu, and W.-S. Zheng. "Action Knowledge Transfer for Action Prediction with Partial Videos." In: ().

[8] S. Ji, W. Xu, M. Yang, and K. Yu. "3D convolutional neural networks for human action recognition." In: *IEEE transactions on pattern analysis and machine intelligence* 35.1 (2012), pp. 221–231.

[9] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. "Learning spatiotemporal features with 3d convolutional networks." In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 4489–4497.

[10] Z. Qiu, T. Yao, and T. Mei. "Learning spatio-temporal representation with pseudo-3d residual networks." In: *proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 5533–5541.

[11] K. Simonyan and A. Zisserman. "Two-stream convolutional networks for action recognition in videos." In: *Advances in neural information processing systems*. 2014, pp. 568–576.

[12] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. "Long-term recurrent convolutional networks for visual recognition and description." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 2625–2634.

[13] A. F. Bobick and J. W. Davis. "The recognition of human movement using temporal templates." In: *IEEE Transactions on Pattern Analysis & Machine Intelligence* 3 (2001), pp. 257–267.

[14] D. Weinland, R. Ronfard, and E. Boyer. "Motion history volumes for free viewpoint action recognition." In:

[15] W. Förstner and E. Gülch. "A fast operator for detection and precise location of distinct points, corners and centres of circular features." In: *Proc. ISPRS intercommission conference on fast processing of photogrammetric data*. Interlaken. 1987, pp. 281–305.

[16] I. Laptev. "On space-time interest points." In: *International journal of computer vision* 64.2-3 (2005), pp. 107–123.

[17] A. Klaser, M. Marszałek, and C. Schmid. "A spatio-temporal descriptor based on 3d-gradients." In:

[18] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld. "Learning realistic human actions from movies." In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*.

[19] N. Dalal and B. Triggs. "Histograms of oriented gradients for human detection." In: *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*. Vol. 1. IEEE. 2005, pp. 886–893.

[20] C. Schuldt, I. Laptev, and B. Caputo. "Recognizing human actions: a local SVM approach." In: *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.* Vol. 3. IEEE. 2004, pp. 32–36.

[21] D. Tran and A. Sorokin. "Human activity recognition with metric learning." In: *European conference on computer vision*. Springer. 2008, pp. 548–561.

[22] T. V. Duong, H. H. Bui, D. Q. Phung, and S. Venkatesh. "Activity recognition and abnormality detection with the switching hidden semi-markov model." In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. IEEE. 2005, pp. 838–845.

[23] N. Ikizler and D. Forsyth. "Searching video for complex activities with finite state models." In: *2007 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2007, pp. 1–8.

[24] M. Raptis and L. Sigal. "Poselet key-framing: A model for human activity recognition." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 2650–2657.

[25] L. Xia, C.-C. Chen, and J. K. Aggarwal. "View invariant human action recognition using histograms of 3d joints." In: *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. IEEE. 2012, pp. 20–27.

[26] M. S. Ryoo. "Human activity prediction: Early recognition of ongoing activities from streaming videos." In: *2011 International Conference on Computer Vision*. IEEE. 2011, pp. 1036–1043.

[27] Y. Kong, S. Gao, B. Sun, and Y. Fu. "Action prediction from videos via memorizing hard-to-predict samples." In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[28] S. Ma, L. Sigal, and S. Sclaroff. "Learning activity progression in lstms for activity detection and early detection." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 1942–1950.

[29] T. Han, J. Wang, A. Cherian, and S. Gould. "Human action forecasting by learning task grammars." In: *arXiv preprint arXiv:1709.06391* (2017).

[30] M. Ryoo, T. J. Fuchs, L. Xia, J. K. Aggarwal, and L. Matthies. "Robot-centric activity prediction from first-person videos: What will they do to me?" In: *2015 10th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE. 2015, pp. 295–302.

[31] H. S. Koppula and A. Saxena. "Anticipating human activities using object affordances for reactive robotic response." In: *IEEE transactions on pattern analysis and machine intelligence* 38.1 (2015), pp. 14–29.

[32] R. Luo and D. Berenson. "A framework for unsupervised online human reaching motion recognition and early prediction." In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, pp. 2426–2433.

[33] J. Mainprice and D. Berenson. "Human-robot collaborative manipulation planning using early prediction of human motion." In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, pp. 299–306.

[34] H. Su, J. Zhu, Y. Dong, and B. Zhang. "Forecast the Plausible Paths in Crowd Scenes." In: *IJCAI*. Vol. 1. 2017, p. 2.

[35] Wikipedia contributors. *Kinect — Wikipedia, The Free Encyclopedia*. 2020. URL: https://en.wikipedia.org/w/index.php?title=Kinect&oldid=991440017.

[36] *freenect_camera - ROS Wiki*. https://wiki.ros.org/freenect_camera. Accessed: 2020.

[37] *libfreenect*. https://github.com/OpenKinect/libfreenect. Accessed: 2020.

[38] I. Kim. *tf-pose-estimation*. https://github.com/ildoonet/tf-pose-estimation.

[39] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh. "Realtime multi-person 2d pose estimation using part affinity fields." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7291–7299.

[40] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh. "OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields." In: *arXiv preprint arXiv:1812.08008* (2018).

[41] K. Simonyan and A. Zisserman. "Very deep convolutional networks for large-scale image recognition." In: *arXiv preprint arXiv:1409.1556* (2014).

[42] ROS-Industrial. *universal_robot*. https://github.com/ros-industrial/universal_robot. Accessed: 2020.

[43] I. A. Sucan and S. Chitta. *MoveIt*. moveit.ros.org. Accessed: 2020.

[44] U. R. A/S. *Universal_Robots_ROS _Driver*. https://github.com/UniversalRobots/Universal_Robots_ROS_Driver. Accessed: 2020.

[45] *Universal Robot UR10e*. https://www.universal-robots.com/products/ur10-robot/. Accessed: 2020.

[0] *File:Outline-body.png*. https://en.wikipedia.org/wiki/File:Outline-body.png. Accessed: 2020.