

Master Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Probabilistic Gesture Control for a Robotic Arm

Petr Vanc

Supervisor: M.Sc. Jan Kristof Behrens
Supervisor–specialist: Mgr. Karla Štěpánová, Ph.D.
May 2021

Acknowledgements

I would like to thank my thesis supervisors M.Sc. Jan Kristof Behrens and Mgr. Karla Štěpánová, Ph.D. of the Czech Institute of Informatics, Robotics and Cybernetics at Czech Technical University for their willingness and rare advices.

Declaration

I declare that I have written this work independently and I have listed all literature I used.

In Prague, 21. May 2021

Abstract

Human-Robot-Interaction and Learning from Demonstration require intuitive and ergonomic means of communication. Guiding a robot through the task of simple interaction with its environment usually requires, additionally to a domain expert, also a robotics expert. Human hand gestures are a valuable communication modality, since *(i)* humans are very skilled in producing them, and *(ii)* relevant geometrical quantities can easily be demonstrated.

In this thesis, we develop a connection between the human hand and a robot manipulator arm. Different methods for gesture-based robot control, including Cartesian position, are explored, such as live mapping with the manipulator and its action-based control. Definition and implementation of several methods to perform various robotic manipulation use-cases (e.g., opening cabinet drawers, pushing a button, and manipulating objects).

The approach was implemented using a selection of static and dynamic gestures and verified in isolated experiments as well as in a user study with seven participants. Two gesture detection mechanisms were implemented and compared: an engineered approach based on selected features and a probabilistic learning-based method.

The whole system idea is to forge together different parts of the system, where independently they are widely used, but together, they are unused. The thesis originated for a purpose of implementation Human-Robot-Interaction into the everyday world with a possibility to facilitate the people life. That is where controlling the robot manipulator with a hand can be beneficial.

Keywords: Human-Robot-Interaction, gesture detection, probabilistic gesture detection, Probabilistic neural networks,

Dynamic Time Warping, robot manipulator

Supervisor: M.Sc. Jan Kristof Behrens
Jugoslávských partyzánů 1580/3,
160 00 Praha 6

Abstrakt

Interakce člověka s robotem a učení se z demonstrace vyžadují intuitivní a ergonomické komunikační prostředky. Vedení robota úkolem jednoduché interakce s jeho prostředím obvykle vyžaduje kromě odborníka na doménu také odborníka na robotiku. Gesta lidské ruky jsou cennou komunikační modalitou, protože *(i)* lidé jsou velmi zruční v jejich výrobě a *(ii)* relevantní geometrické veličiny lze snadno demonstrovat.

V této práci rozvíjíme spojení mezi lidskou rukou a robotickým ramenem manipulátoru. Jsou zkoumány různé metody pro ovládání robotického manipulátoru na základě gest, včetně kartézské polohy, jako je živé mapování pomocí robotického manipulátoru a jeho ovládání založené na řízení. Definice a implementace několika metod k provádění různých případů použití robotické manipulace (např. Otevírání zásuvek skřínky, stisknutí tlačítka a manipulace s objekty).

Přístup byl implementován pomocí výběru statických a dynamických gest a ověřen v izolovaných experimentech i v uživatelské studii se sedmi účastníky. Byly implementovány a porovnány dva mechanismy detekce gest: inženýrským přístupem založeným na vybraných funkcích a pravděpodobnostní metodou založenou na učení neuronové sítě.

Myšlenkou celého systému je spojit dohromady různé části systému, kde jsou nezávisle široce používány, ale společně využívány nejsou. Práce vznikla za účelem implementace interakce člověk-robot do každodenního světa s možností usnadnit lidem život. Jelikož se jedná o obor, kde může být ovládání robotického manipulátoru rukou výhodné.

Klíčová slova: Člověk-robot interakce, detekce gest, pravděpodobnostní detekce gest, pravděpodobnostní neuronové sítě, Dynamic Time Warping, robot manipulator

Překlad názvu: Pravděpodobnostní přístup k ovládání robotického ramene pomocí gest

Contents

1 Introduction	1	5.6 Dataset	36
1.1 Thesis Motivation	1	6 Gesture recognition	39
1.2 Brief summary of used approach	2	6.1 Defined gestures	39
1.3 Goals of the thesis	3	6.2 Using value measures to evaluate gestures	41
2 Related work	5	6.3 Deterministic approach	41
2.1 Gesture recognition industry	5	6.3.1 Static gesture recognition	43
2.2 Robot control via gestures	5	6.3.2 Dynamic gesture recognition	47
2.2.1 Human-Machine-Interaction via gestures	6	6.4 Probabilistic approach	50
2.2.2 Thesis acquisition	7	6.4.1 Static gestures recognition	51
3 Theoretical background	9	6.4.2 Modelling dynamic gestures via ProMP	55
3.1 Brief robot control	9	6.4.3 Dynamic Gesture recognition via Dynamic Time Warping	60
3.1.1 Motion planning via <i>Relaxed IK</i>	11	7 Robot control via gestures	65
3.2 Introduction about gestures	12	7.1 Controlling the end-effector via gestures	65
3.2.1 Hand detection methods	13	7.1.1 Correspondence mapping from perceived hand pose to robot workspace	66
3.3 Hand bone structure	13	7.1.2 Live mapping	68
3.4 Probability theory introduction	14	7.1.3 Interactive mapping	69
3.4.1 Probabilistic Machine learning and Programming	15	7.1.4 Memory path execution with hand control	70
3.4.2 Bayesian inference processes	16	7.1.5 Gesture based robot control use-cases	70
3.4.3 ADVI	16	7.1.6 Iterative gesture control	73
3.4.4 MCMC sampling	17	7.2 Testing accuracy of inverse kinematics controller	73
3.5 Dynamic Time warping introduction	17	7.3 User study	74
3.6 ProMP background	18	8 Conclusion and Future work	77
4 Material and methods	21	Bibliography	79
4.1 Experimental setup	21	A List of Abbreviations	85
4.1.1 <i>Leap Motion</i> Controller	21	B ROS messages definitions	87
4.1.2 Kuka iiwa LBR manipulator	22	C Project Specification	89
4.2 Modeling of probabilistic network	23		
4.2.1 Probabilistic package <i>PyMC3</i>	23		
4.2.2 ProMP	23		
4.3 How to read Confusion matrix	23		
5 System overview	25		
5.1 Setup overview and dataflow	26		
5.1.1 Program configuration and communication between nodes	27		
5.2 Processing Leap Controller data	29		
5.2.1 Further process of specific variables	30		
5.3 Robot setup	32		
5.4 User Interface	32		
5.5 Defined manipulation workspaces	34		
5.5.1 Leap workspace definition	35		

Figures

3.1 Robot control diagrams. Description what blocks perform one to another.	10	7.1 Transformations between workspaces.	66
3.2 Bone structure. [1]	14	7.2 <i>Above</i> and <i>Wall</i> workspaces visualized.	68
3.3 A comparison of Euclidean distance a) with Dynamic Time Warping b) for time series. [2] . . .	18	7.3 Live mapping example with drawing a trajectory.	69
4.1 Experimental setup workspace. . .	21	7.4 Path play trajectory bar.	70
4.2 Example confusion matrix.	24	7.5 Picking and placing of two boxes. Two boxes scene.	71
5.1 Dataflow diagram.	25	7.6 Opening and closing drawer sockets.	71
5.2 Main page of the application. . .	33	7.7 Clicking a button.	72
5.3 Configuration page of the application.	34	7.8 Plane test accuracy of Relaxed IK controller.	73
5.4 Workspace of <i>KUKA iiwa</i>	35		
5.5 Interaction box of Leap Motion Controller [3].	36		
5.6 Example of dataset recording. . .	37		
6.1 Gesture processing diagram. . . .	39		
6.2 Pinch measure example.	41		
6.3 The feature rm example demonstration.	42		
6.4 Applied Boolean hysteresis. . . .	43		
6.5 Five closed fingers, five features.	44		
6.6 Static gesture recognition via Deterministic approach.	47		
6.7 Layers of neural network model.	52		
6.8 Example of dataset.	54		
6.9 n hidden layers vs. Accuracy. . .	54		
6.10 Static gesture recognition via Probabilistic approach.	56		
6.11 Static gesture recognition via Probabilistic approach.	57		
6.12 Dataset samples of gesture <i>Swipe left</i>	58		
6.13 Dynamic gesture recognition via Euclidean distance.	60		
6.14 Dynamic gesture recognition via Euclidean distance.	61		
6.15 Dynamic gesture recognition via Dynamic Time Warping.	63		
6.16 Dynamic gesture recognition via Dynamic Time Warping.	64		

Tables

3.1 Denavit-Hartenberg parameters	11
5.1 <i>Frame</i> \mathcal{F} definition.	27
5.2 <i>Processed frame</i> \mathcal{P} definition.	28
5.3 Deterministic gesture recognition output.	28
5.4 Probabilistic gesture recognition output.	28
5.5 <i>Inverse kinematics</i> node input.	29
5.6 <i>Inverse kinematics</i> node output.	29
6.1 User study, dynamic time gestures deterministically.	50
6.2 Probabilistic experiment, 2-layer model.	55
6.3 Probabilistic experiment, 3-layer model.	55
6.4 Probabilistic experiment, 2-layer model, var. 2.	56
6.5 Probabilistic experiment, 2-layer model, var. 3.	57
6.6 Dynamic Time Warping method.	59
6.7 Dynamic Time Warping method.	62
7.1 User study evaluation of Deterministic approach.	74
7.2 User study evaluation of Probabilistic approach.	75
7.3 User study, drawer manipulation method.	75
7.4 User study, pick/place manipulation method.	76
7.5 User study, button manipulation method.	76
B.1 Pose definition.	87
B.2 Timestamp definition.	87
B.3 Joint Trajectory definition.	87
B.4 Joint Trajectory definition.	87

Chapter 1

Introduction

1.1 Thesis Motivation

Programming and testing a new robot manipulator operation task requires a lot of work. Most of the robot movements are deterministic and do not offer any kind of variation. In the future, there will be a substantial demand for robots that can operate on its own in various situations. Robots will need to be capable of handling more common situations than only the programmed ones. For example, robots could be in principle good work bench companions, handing over tools and work pieces or aid in manipulating heavy objects. New robot interaction methods are needed to make the human-robot interaction more fluid and robust. In this thesis is introduced a system that facilitates human gestures to control a robot performing a set of (manufacturing related) tasks.

What if there was a more natural way to manipulate the robot operations than with the standard approaches (e.g., keyboard, mouse). One option is to use the most essential asset we have - a hand. Human hands are capable of producing a large variety of nuanced gestures, which are much used in human-human conversations (adding accentuation, geometric analogies, etc.).

To hand gestures might be assigned various signaling meanings, such as direction, specific action or more advanced meanings. We can even extend that list and define via gestures, signaling quantities (e.g., length of motion, velocity conditions, etc...), geometric relations (e.g., orientation), time points (e.g., pulse quantities in a pattern), etc. Hand position has very good manipulation control with very good accuracy. That is why, still to this day, the computer mouse is one of the most accurate controlling devices for computers. That could be a good match to couple the robot manipulation end-effector to the hand position.

How far can we go with hand control? To what extend can actually the hand control be used for operating a robot? Should this approach be expanded into daily use within programmers and even real users? Which steps towards its wide spread are needed? What accuracy of the gesture recognition is needed to make this method useful?

There is an option to combine the coupling of positions while performing a

unique static gesture with a hand, for example, grabbing with a hand while maintaining a specific position. That is the most elementary but simple example that could perform a decent function for defined control operations. For example, opening a drawer or grabbing a box.

The tasks could be split into different levels of autonomy (triggering, controlling process, and fine-grained motion control). For example, disaster response remote control. There is a need to trigger some response in a form of a robot manipulation execution. Another example could be a crane manipulator control. With the position of a hand, the operator can easily operate the rotation of the crane, its manipulation arm, and its holder with high accuracy. Some security features would need to be introduced while achieving this situation. Speaking of security, robustness and safety are a major concern in human-robot interaction.

Quantifying the uncertainty in human-robot interaction has many potential benefits, especially the increase of robustness. One option to model uncertainty is to utilize probabilistic methods for this purpose. The framework of probabilistic programming enables (i) to learn from data and (ii) classify observations, and (iii) evaluate the certainty with which was the observation classified within the given category.

In the context of this thesis, we use probabilistic programming to learn new gestures from demonstration data and to classify gestures during the robot operation. Briefly speaking, the thesis aims to develop a connection between the human hand and the robot arm. We evaluate the accuracy of detection of various types of gestures. To answer the above mentioned question if the system might be useful for real users, we evaluate the quality of the proposed solution on a set of users of various age.

1.2 Brief summary of used approach

The hands of the user are recorded using a proper device (specifically Leap Motion 4.1.1). Hands are detected and bone structure is constructed from hands. The bone structure with respect to time is passed into the gesture detection application where the output is matched to a previously learned system of gestures. The detected gesture is then mapped to robot manipulator actions. This action can be a direct execution of a motion or a condition for another more complex action which consists of multiple motions. The actions can be of various types and are linked to the robot configuration (e.g., a dynamic gesture *Swipe left* is performed relatively to the current robot configuration).

Gesture detection is done with a deterministic method and probabilistic method. The deterministic one is programmed and tuned by hand with the intention of achieving the best possible values for the detection of each gesture. The probabilistic gesture detection is automatic detection where the system itself finds the best parameters to classify between individual gestures. The Bayesian machine learning methods are trained on an input dataset of hand movements. Specifically, we use a process known as Bayesian inference to

detect individual gestures, which are previously defined. Gesture detection via Bayesian inference can be a difficult task, usually demanding a lot of computational power. Often it is needed to select the right approach to utilize the input dataset effectively and satisfy user demands. Then there needs to be verified if, indeed, the statistical model is valid (i.e., if it is still true to original idea or it just behaves like that), checking validity by going back to the roots even if the model is working correctly and analyze the system again.

1.3 Goals of the thesis

The main goal of the thesis is to create a mapping between a hand and a robot's end-effector to enable direct control of the robot by gestures. A visual feedback of the detected pose and robot movement should be provided to the operator.

With that being said, there is a need to define a set of manipulation tasks which the robot manipulator should perform. These tasks might include, for example, picking an object, pushing a button, or opening and closing a socket of a drawer. Afterwards, a suitable gesture from the custom set of gestures is attached to every action.

Furthermore, a set of gestures and corresponding robot actions should be specified. Categorize gestures and recognize them by the system. Use not only deterministic approaches but also probabilistic ones.

A custom gesture-based user interface enables to display the results of gesture detection as well as to control the high-level robot actions and properties manually. Switching between individual modes in this interface is enabled also via specific gestures.

User study displaying the accuracy and error rate of manipulation and gesture recognition was performed.

Chapter 2

Related work

In this chapter are presented the studies and experiments on which this thesis builds up. Various topics are presented on each category of research. The focus is on works from the following three categories: gesture recognition, robot control via gestures, and human-machine interaction. Finally, the contribution of the thesis compared to the presented studies is summarized.

2.1 Gesture recognition industry

Gesture recognition has been with us for a few decades already. Either if we are talking about face gesture detection interpretation [4] or hand dynamic gesture detection [5]. Although these were successful attempts to detect gestures from an image sequence, they were limited by the computational power of the machines at that time.

Newer studies processed with very high quality a considerable variance of detection itself. For example, gesture recognition is based on surface Electromyographic signals [6] or Inertial measurement units such as 3 Axis Accelerometers and 3 Axis Gyroscopes.

There is a lot of academic work specific to Leap Motion controller concentrated on gesture detection. Some are concentrating on the recognition of hand gestures themselves using Leap Motion utilities [7]. Or there is a possibility of controlling remote robots, for instance, Quadrotor Drones, by gestures in the form of a game [8]. Where the position of the drone is mapped to the hand of a user using a PID controller.

2.2 Robot control via gestures

There are relevant studies on the utilization of gesture recognition, for example, controlling a wheelchair [9] using a combination of EMG signal sensors and IMU sensors. Methods for gesture recognition in this work are processes that are mainly using the root mean square of input signals with further Deep Support Vector Machine classifier.

There are also studies of Gesture Control with Robotic Arms equipped with a remote transceiver [10] where the output system contains a specific

human hand with five fingers. The author concentrates on execution on a microcontroller and making the control wireless.

Other works use a remote robot agent which has a computer camera mounted on the robot agent [11]. Gesture recognition is performed by utilizing RGB filters for a hand search. Ongoing with the longest distance calculation from the center of gravity of the hand. The output is a direction for the robot to go .

■ 2.2.1 Human-Machine-Interaction via gestures

Now, let us concentrate on related work using the representation of hand movements via Dynamic and Probabilistic motion primitives specifically applied in Human-Robot Interaction.

Before Probabilistic Movement Primitives, the Deterministic Movement Primitive systems have been used with fulfillment for a mixture of robotic tasks along with grasping or locomotion [12].

The advantage of probabilistic approaches is that, they can naturally work with variability using a probability distribution. A few of probabilistic representations of movement primitives target on learning a distribution by demonstrated states using Hidden Markov models. Afterwards, using the log likelihood as cost function to reproduce the learned movement using an optimal control method [13].

The use of a hierarchic HMM to learn and perform responsive robot behaviors. In their approach, high-level HMM identifies the current state of the interaction and triggers low-level HMMs which correspond to the robot's motor primitives. In order to ensure that the robot adapts to the movement of the human partner, virtual springs are attached between markers on the human body and corresponding positions on the robot [14].

Dynamic Movement Primitives allows for a low-dimensional, adaptive representation of a trajectory. The general idea is to encode a recorded trajectory as dynamical systems, which can be used to generate different variations of the original movement. In the context of interaction [15].

The proposed solution combines a linear attractor system which is modulated by a time-dependent forcing function and normalized Gaussian basis functions. Dynamic movement primitives introduce a concept of a phase of movement which can elongate or shorten the desired path, enable learning from a single demonstration [16], or generalization [17]. There are also presented normalized Gaussian basis functions. Dynamic movement primitives introduce temporal scaling of the movement, enable learning from a single demonstration or generalization to new final positions.

Related studies on Probabilistic Motion primitives include Jan Peters team specializing in physical human-robot interaction using imitation learning. In [18] they evaluated the quality of imitation learning on a classical n degree of freedom robot arm.

■ 2.2.2 Thesis acquisition

Most of the previous studies have focused on either gesture detection or robot control. This thesis emphasizes the integration of gesture detection on the robot arm while using deterministic and probabilistic methods and comparing their performance.

Chapter 3

Theoretical background

In this section are introduced all things needed to understand the theory and the proposed system. Robot control is briefly introduced, various hand and gesture detection techniques are presented. Introduction to probability theory and basics when performing probabilistic learning are shown.

3.1 Brief robot control

The control of the robot should achieve flawless following a constantly moving target with the end-effector. It should deal with the singularities and avoid collisions. *ROS* libraries with MoveIt! interface are used (see S. 4.1.2) on *Capek* workspace (see S. 4.1.2). The process of robot control is visualized in Fig. 3.1a.

In the Fig. 3.1a can be seen the pipeline from the application to the robot control block. Arrows represent the data which is propagated through the pipeline and the blocks are described below. The Fig. 3.1b represents, what is needed to be done until the joint states are published.

In live trajectory execution, the robot path is constantly updated to follow the moving target. There is a request for execution move towards the defined path pose \mathbf{p} . Pose is made up of position and orientation (see Tab. B.1). This is output from *Path Pose* block.

These poses are converted to joint states using an inverse kinematics (see Eq. 3.1) in *Inverse kinematics* block. Joint states represent the angle orientations of each robot joint. Operation can be written as $\mathbf{j} = IK(\mathbf{p})$ (3.1), where $\mathbf{j} = (j)_{n=1}^N$ are joint states with N depending on the number of robot joints. In the case of the *KUKA iiwa* $N = 7$, *IK* function is represented by *Relaxed IK* module [19], it deals with disjointed end-effector jumps and self-collisions, for more see S. 3.1.1.

Trajectory action client [20] is used to send trajectory segments to the robot and update them on demand. The robot interpolates the trajectories using quintic splines and executes them using the position control interface of the robot. The *iiwa Fast Robot Interface* is then used on the real robot. The input of this block is a list of joint states with timing. This array can be filled

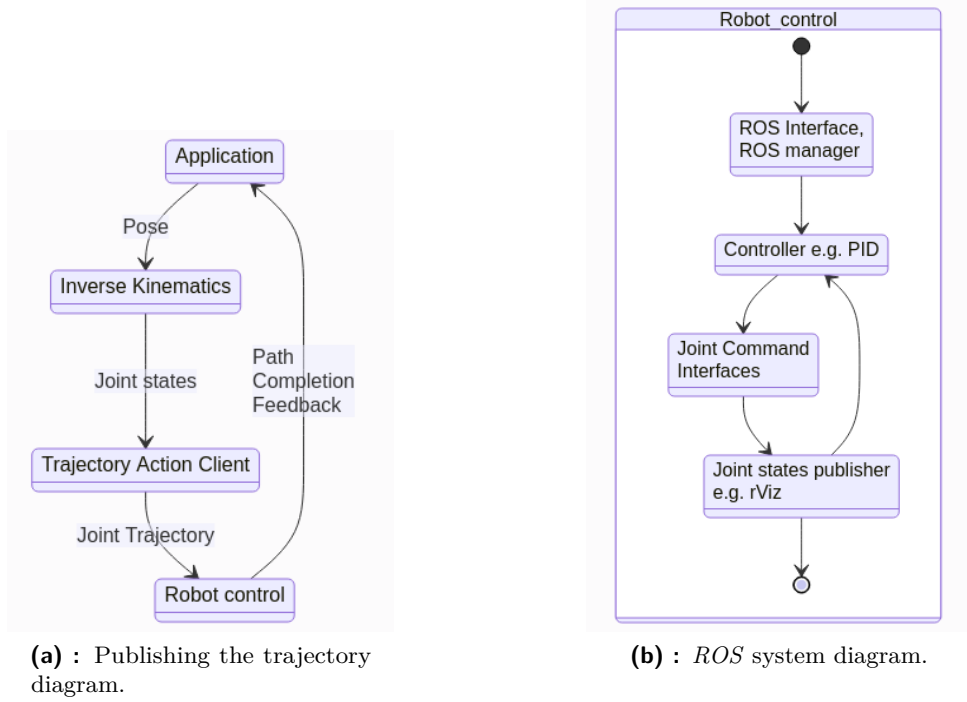


Figure 3.1: Robot control diagrams. Description what blocks perform one to another.

with joint states from the current state towards the goal state as follows:

$$\mathbf{j}_a = \left(0.1 \cdot i \cdot (\mathbf{j}_g - \mathbf{j}_0) + \mathbf{j}_0 \right)_{i=1}^N, \quad (3.2)$$

where \mathbf{j}_g is the set of goal joint states, \mathbf{j}_0 is the array of current joint states. \mathbf{j}_a is two-dimensional array of joint states across time and N is the tens of percent of execution between the stated joint values and the goal ones. N can be assigned to value $N = 5$, then every trajectory update will update *Joint state trajectory* only to half of a goal, which in live tracking results in finer movements.

Trajectory is made of ROS Joint Trajectory message type (see Tab. B.3). It contains an array of trajectory points. These trajectory points hold the joint states of the manipulator from the present to goal joint state positions.

Path completion feedback includes the received robot joint states. To get the pose of end-effector again, the forward kinematics task is performed.

Exemplary computation of Forward Kinematics task for *Kuka iiwa* manipulator can be written in terms of Denavit-Hartenberg (DH) notation, which is represented by homogeneous transformations by propagating from the first to the last joint [21]. The homogeneous transformations is computed as Eq. 3.3, where *Rot* is function for rotation matrix and *Tran* is function for translation matrix, the parameters for the function are DH parameters: θ_i , d_i , a_{i-1} , α_{i-1} , where θ_i is the angle of i th joint, $i \in \{1, \dots, 7\}$. d_i , a_{i-1} represents the translation motion and α_{i-1} represents the rotational motion.

The homogeneous matrix is created in Eq. 3.4 and Eq. 3.5. The propagation from the first angle to the last by multiplying the homogeneous matrices is given by Eq. 3.6. Lastly, the real translation and rotation values of a specific robot are marked into Tab. 3.1.

$$T_i = Rot(x, \theta_i) \cdot Tran(z_i, d_i) \cdot Tran(x, a_i) \cdot Rot(x, \alpha_i) \quad (3.3)$$

$$T_i = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} & 0 & 0 \\ s_{\theta_i} & c_{\theta_i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_{\alpha_i} & -s_{\alpha_i} & 0 \\ 0 & s_{\alpha_i} & c_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.4)$$

$$T_i = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.5)$$

$$T_7^0 = T_1^0 \cdot T_2^1 \cdot T_3^2 \cdot T_4^3 \cdot T_5^4 \cdot T_6^5 \cdot T_7^6 \quad (3.6)$$

θ_i	d_i [mm]	a_{i-1}	α_{i-1} [°]
q1	0.34	0	-90
q2	0.00	0	90
q3	0.40	0	90
q4	0.00	0	-90
q5	0.40	0	-90
q6	0.00	0	90
q7	0.126	0	0

Table 3.1: Denavit-Hartenberg parameters [21].

3.1.1 Motion planning via Relaxed IK

Relaxed inverse kinematics is a real-time motion synthesis method for robot manipulators that cannot only accurately match end-effector pose goals as done by traditional IK solvers, but also create smooth, feasible motions that avoid joint space discontinuities, self-collisions, and kinematic singularities [19].

Inverse kinematics is solved as a weighted-sum nonlinear optimization problem. Motion goals in addition to end-effector pose matching can be encoded as terms in the sum. There is a normalization procedure such that the method can effectively make trade-offs to simultaneously reconcile many and potentially competing objectives. Using these trade-offs, the formulation allows features to be relaxed when in conflict with other features deemed more important at a given time [19].

Relaxed IK uses neural networks to speed up the computation. Effectively, it caches the collision checks. There is a need to set up the current robot manipulator configuration. Learning of the neural network takes about 10 minutes based on a given CPU. Once the processing is done, it is saved into a file and can be loaded whenever needed.

Planning contains its own collision detection. Different objects can be added to the workspace. The neural network is then learned with that configuration in mind. This can be used, for example, when there are some static objects in the workspace or static objects with respect to each link of the manipulator.

3.2 Introduction about gestures

The human hand is capable of performing very diverse motions and enables many different configurations. Some of the configurations and some motions are used as a mean of communication, i.e., gestures. An important observation is that gestures are not precise points in the configuration space of the hand, but rather regions (e.g., for the gesture signaling *Fingers crossed* gesture can be defined as the position of the two fingers to each other is crucial, but the absolute location of the hand does not matter that much).

Before the gesture definitions will be presented, let us divide the gestures by time series into two categories:

- Static gestures
- Dynamic gestures

Static gestures are computed from only as a single time sample (frame) at a time. Additionally, the mean values of multiple time frames can be used as well. Dynamic gestures are computed from time multiple samples (frames). Therefore, all time samples from the recording can be used.

Gesture definition

The main two gesture type definitions used in the thesis are *Manipulative* and *Semaphore* gestures. The manipulative type of gestures and semaphore ones. Definition of a *manipulative gestures* can be interpreted as:

“Manipulative gesture is one whose intended purpose is to control some entity by applying a tight relationship between the actual movements of the gesturing hand/arm with the entity being manipulated [22].”

Manipulation can be in 2D, e.g., a computer mouse, or in 3D, which involves the hand to mimic manipulation in the space of some virtual object, e.g., the Leap Motion sensor.

Three dimensional space can serve a purpose as a 3D mouse to manipulate 2D objects, where there is a defined picking and placing zone on the third-dimensional axis. Or the third dimension can be utilized to modulate pressure.

Because the robot manipulator operates in 3D, the robot end-effector can be directly mapped to the 3D gesture space.

The definition of *Semaphore gestures* is stated:

“Semaphores are systems of signaling using flags, lights, or arms. We define semaphoric gestures to be any gesturing system that employs a stylized dictionary of static or dynamic hand or arm gestures. Semaphoric approaches may be referred to as ”communicative” in that gestures serve as a universe of symbols to be communicated to the machine [22].”

■ 3.2.1 Hand detection methods

The basis for gesture detection is to detect a hand. In the world, there exist a lot of hand detection methods. They depend on what data are on input. If the data on the input is a point cloud, which is a set of Cartesian points in space, the bone structure can be obtained with RANSAC. *Random Sample Consensus* is an iterative algorithm for the robust estimation of parameters from a subset of inliers from the incomplete data set. RANSAC type of regression returns a lot of lines with some processing can be constructed to the bone structure [23].

Another option is when we are working with one RGB image, the Convolutional Neural network can be used to predict 2D heatmaps and novel 3D location maps for all joints. The 2D keypoints can be retrieved after filtering and read off the 3D pose. Per-frame estimates are combined with a stable global pose by skeleton fitting [24]. Another approach is to model and detect the hand bone structure.

■ 3.3 Hand bone structure

From Osteologist point of view, the hand can be extracted into bones as in Fig. 3.2.

Another approach is to model and detect the hand bone structure using two infrared cameras as it might be with Leap Motion. Leap Motion detects the bone structure of the hand. Specifically, it detects the following points on the the hand. Note that every point has Cartesian coordinates $P^* = (x, y, z)$, the index will be omitted for simplicity.

- Bone structure of points $\mathbf{S} \in \{P_w, P_p, P_f^b\}$
 - Wrist P_w
 - Palm P_p
 - Thumb, Index, Middle, Ring, Pinkie fingers P_f^b , ($f = 1, \dots, 5$), ($b=1, \dots, 4$)
 1. Tip point of bone Metacarpal P_f^0

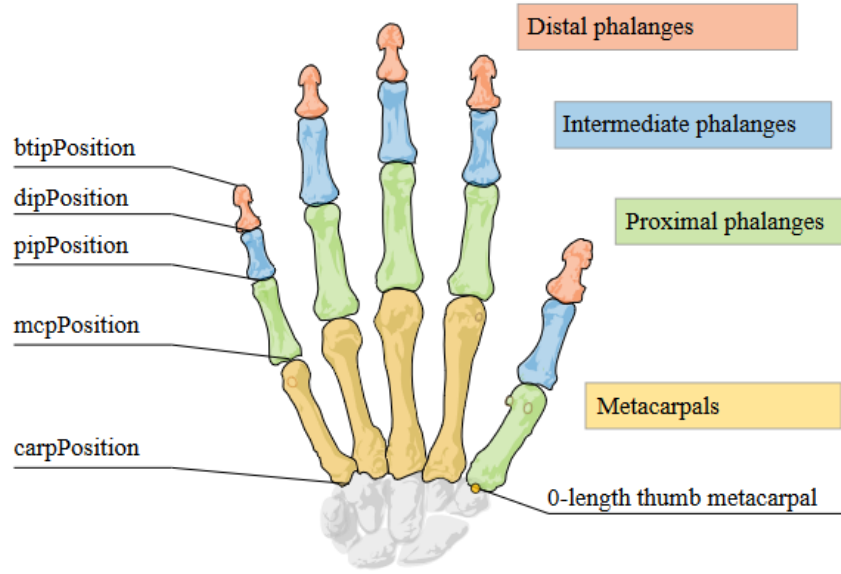


Figure 3.2: Bone structure. [1]

2. Tip point of bone Proximal P_f^1
3. Tip point of bone Intermediate P_f^2
4. Tip point of bone Distal P_f^3

3.4 Probability theory introduction

Let's define a *Random variable* A . The realization of this variable is not known and can have multiple values. In general, set of one or more values can be defined as *event*. Probability that an event is occurring is a scalar value in the closed interval 0 to 1. For the number 0 that event is certainly not happening and for probability 1 of the event is certainly happening.

The probability of an event X with outcome x can be written by Eq. 3.7.

$$P(X = x) \quad (3.7)$$

Conditional probability is defined as the probability of an event A given that another event B is occurring:

$$P(A|B). \quad (3.8)$$

Mathematical rules of probability can be outlined by *product rule* and *sum rule*.

Sum rule. The probability that either A or B occurs is shown in Eq. 3.9.

$$P(A \cup B) = P(A) + P(B) - P(A \cap B). \quad (3.9)$$

Product rule (also called the chain rule). The probability of two events A and B occurring in parallel $P(A, B)$ is the multiplication of the probability of

one of the events and the conditional probability of the other event happening given the first event happened:

$$P(A \cap B) = P(A|B) \cdot P(B) = P(B|A) \cdot P(A). \quad (3.10)$$

Bayes' Theorem describes inverted conditional probability. Probability of an event A given event B can be expressed via the probability of event B given an event A . See Eq. 3.11.

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}. \quad (3.11)$$

Probability density function can be defined as the relative probability of areas of the space of outcomes. Probability density computed using an integral which is shown in Eq. 3.12.

$$P(X \in A) = \int_{x \in A} p_X(x) dx, \quad (3.12)$$

$X \in A$ means that event X is in A . Cumulative distribution function $P(X \leq x)$ is defined as the probability X is less or equal to the outcome x (see Eq. 3.13).

$$P(X \leq x) = \int_{-\infty}^x p_X(u) du. \quad (3.13)$$

Expected value $\mathbb{E}[X]$ of a variable X is the average value of the variable if an infinite number of independent samples are made. Definition of the probability density function is described in Eq. 3.14.

$$\mathbb{E}[X] = \int xp(x) dx \quad (3.14)$$

Finding the best volume by integration not with optimization.

3.4.1 Probabilistic Machine learning and Programming

The probabilistic programming is applying the above-mentioned probabilistic rules as some generative model. The generative model can use the prior distribution of the values as an input, the likelihood works similarly as the conditioning of statements. The output of these methods is a posterior distribution. In Bayesian framework, probabilities are seen as a degree of belief based on a prior knowledge. The output of the neural network model will grant us data as fixed and the parameters as random variables. This means that the parameters of our model will be represented by distributions.

The posterior $P(\theta|y)$ can be evaluated using numerical methods. The posterior is proportional to the likelihood $P(y|\theta)$ times the prior $P(\theta)$:

$$P(\theta|y) \propto P(y|\theta)P(\theta) \quad (3.15)$$

When speaking about machine learning, there would be useful to define the data structure of such networks.

■ Neural network architecture

The Neural Network architecture is chosen accordingly to the type of data. Probabilistic machine learning is based on Bayes' theorem (see Eq. 3.11), which is computed in separate units. The unit nodes are called perceptrons, resp., neurons. All units in parallel then form a specific layer. The length of the layer is then denoted as *hidden layer size*. More layers in serial then form a multilevel neural network model.

Now the used methods of probabilistic programming will be presented.

■ 3.4.2 Bayesian inference processes

Inference problem requires statements about the value of an unobserved (latent) variable x based on observations y which are related to x , but may not be sufficient to fully determine x . This requires a notion of uncertainty. In the real world, there does not exist a fully certain event. We can specify priors to inform and constrain our model and get the uncertainty estimation in the form of a posterior distribution.

Firstly, the proposed method (see Ch.6.4.1) is called variational inference that is used to quantify the uncertainty in the weights of deep neural networks. It makes informed predictions for the gesture recognition dataset. More in S. 3.4.3. Secondly, Markov Chain Monte Carlo is used for generating result samples from the posterior distribution. More in S. 3.4.4

■ 3.4.3 ADVI

Automatic Differentiation Variational Inference. Probabilistic modeling is iterative. A scientist deposits a simple model, fits it to the data, refines it according to the analysis, and repeats. It is a Gradient-Based method. Some systems defined without inference models would be hard to implement using conventional frameworks.

Variational Inference minimizes the *Kullback-Leibler divergence* which is shown in Eq. 3.16.

$$\mathbb{KL}(q(\theta)||p(\theta, y)) = \int q(\theta, \phi) \frac{q(\theta, \phi)}{p(\theta, y)} d\theta \Rightarrow \mathbb{E}_q \left(\log \left(\frac{q(\theta)}{p(\theta|y)} \right) \right) \quad (3.16)$$

from the approximate distributions, but we cannot calculate the true posterior distribution [25].

Evidence lower bound (ELBO) is calculated by minimizing the *Kullback Leibler* divergence (see Eq. 3.17, Eq. 3.18).

$$\mathbb{KL}(q(\theta)||p(\theta, \mathcal{D})) = -(\mathbb{E}_q(\log p(\mathcal{D}, \theta)) - \mathbb{E}_q(\log q(\theta)) + \log p(\mathcal{D})), \quad (3.17)$$

$$ELBO = (\mathbb{E}_q(\log p(\mathcal{D}, \theta)) - \mathbb{E}_q(\log q(\theta))). \quad (3.18)$$

Optimization of the model includes tuning the parameters by examining the posterior distribution, over the weights.

3.4.4 MCMC sampling

A Markov Chain is a mathematical process that undergoes transitions from one state to another. Key properties of a Markov process are that it is random and that each step in the process is “memoryless” in other words, the future state depends only on the current state of the process and not the past. ??

MCMC is particularly useful in Bayesian inference because of the focus on posterior distributions which are often difficult to work with via analytic examination. In these cases, MCMC allows the user to approximate aspects of posterior distributions that cannot be directly calculated (e.g., random samples from the posterior, posterior means, etc.). [26]

In modeled neural network system, this process is used to make samples from posterior distributions.

3.5 Dynamic Time warping introduction

Dynamic Time Warping is used to compare the similarity or calculate the distance between two arrays or time series with different lengths and speed [27]. Time warping can be applied to video, audio, or any linear sequence of data, meaning that it can be interpreted as n-dimensional series.

The time warping method has its own rules. Every index from the first sequence must be matched with one or more indices from the other sequence. The first index from the first sequence must be matched with the first index from the other sequence and the last index from the first sequence must be matched with the last index from the other sequence. The mapping of the indices from the first sequence to indices from the other sequence must be monotonically increasing. If $j > i$ are indices from the first sequence, then there must not be two indices $l > k$ in the other sequence, such that index i is matched with index l and index j is matched with index k [27]. All these rules have to be satisfied, the minimal cost is computed as the sum of the absolute differences, for each matched pair of indices, between their values. In the Lst. 3.1 is example code of Dynamic Time Warping code written in Python language.

```

1 def dtw(s, t):
2     n, m = len(s), len(t)
3     dtw_matrix = np.zeros((n+1, m+1))
4     for i in range(n+1):
5         for j in range(m+1):
6             dtw_matrix[i, j] = np.inf
7     dtw_matrix[0, 0] = 0
8
9     for i in range(1, n+1):
10        for j in range(1, m+1):
11            cost = abs(s[i-1] - t[j-1])
12            # take last min from a square box
13            last_min = np.min([dtw_matrix[i-1, j], dtw_matrix[i,
14            j-1], dtw_matrix[i-1, j-1]])
15            dtw_matrix[i, j] = cost + last_min

```

```
15 return dtw_matrix
```

Listing 3.1: Time-warping sample code in Python. The most important is line 13, where the last minimum from square box is obtained [28].

In the dynamic gesture detection is not only used the Dynamic Time Warping method, but also the Euclidean method, which gives Euclidean distance between two points in the time order of the trajectory. Differences between these two methods in an organization of points can be seen in Fig. 3.3.

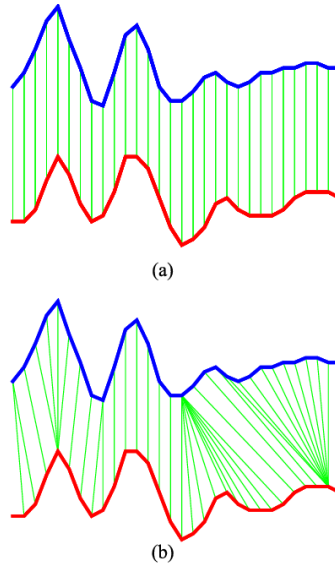


Figure 3.3: A comparison of Euclidean distance a) with Dynamic Time Warping b) for time series. [2]

3.6 ProMP background

Probabilistic movement primitives is well-established approach for representing robot motion skills and blending between motions [17]. It is the right tool to provide generalization to new situations and temporal modulation of the movements.

Furthermore, there is an ability to sequence more shorter motion primitives to create longer ones. Probabilistic movement primitives represent a probability distribution over trajectories. Trajectory distribution can be defined in various spaces such as joint-space, task-space, etc. However, the most reasonable approach for our situation is to use the joint-space.

ProMP encodes the optimal behavior in the case that the system operates with linear dynamics. In contrast to the deterministic approaches, it does not encode only the mean solution, but introduces the variability of the movement. The important key is to model the coupling between the individual DOF (degree of freedom) of the robot by estimating the covariance between different DOFs.

Pseudocode of the implementation of *ProMPs* used in S. ?? can be seen in the Lst. 3.2.

```

1 Data: A set of N trajectories with position observations  $Y_i$ ,
       $i = 1, \dots, N$  at time  $t_i$ 
2 Input: Number of basis functions K, Basis function width h,
      Regression parameter  $\lambda$ 
3 Output: The mean  $\mu_w$  and covariance  $\Sigma_w$  of  $p(\mathbf{w}) \sim \mathcal{N}(\mathbf{w}|\mu_w, \Sigma_w)$ 
4
5 foreach trajectory i do
6
7   Compute phase:  $z_i = \frac{t_i}{t_i^*nd}$ 
8   Generate basis:  $\Psi_t = f(z_i, K, b)$ 
9   Compute the weight vector  $\mathbf{w}_i$  for trajectory i
10
11   $\mathbf{w}_i = (\Psi_t^T \Psi_t + \lambda \mathbf{I})^{-1} \Psi_t^T \mathbf{Y}_i$ .
12 end
13 Fit gaussian over the weights vector  $w_i$ 
14
15  $\mu_w = \frac{1}{N} \sum_{i=1}^N \mathbf{w}_i$ 
16  $\Sigma_w = \frac{1}{N} \sum_{i=1}^N (\mathbf{w}_i \mu_w)(\mathbf{w}_i \mu_w)^T$ 

```

Algorithm 3.1: : Probabilistic Movement Primitives, Learning trajectory movements by Alexandros Paraschos [17].

Chapter 4

Material and methods

In this chapter are presented the used materials and methods to build the solution. Used devices for detection, the type of used robot manipulator, probabilistic package, and more are described in this chapter.

4.1 Experimental setup

The components of the setup are mainly composed of two units. The sensing device *Leap Motion Controller* and the robot manipulator *KUKA iiwa LBR*. The setup can be visualized as Fig. 4.1. The overview of the setup is then presented in the next chapter Ch. 5.

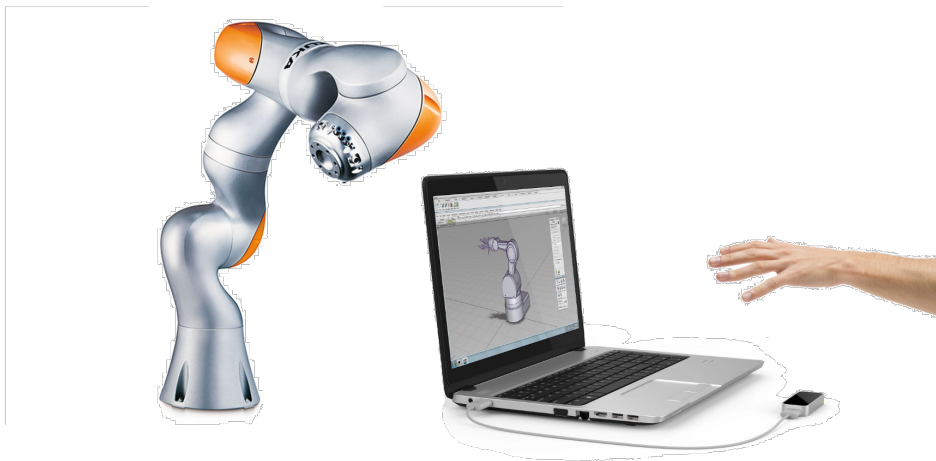


Figure 4.1: Leap Motion controller with Kuka iiwa LBR manipulator, Experimental setup [29] (edited)

4.1.1 *Leap Motion Controller*

In this experimental setup, Leap Motion Controller is used to track hand bone structure.

Leap Motion Controller is a small device approximately the size of a thumb (see Fig. 4.1). It connects to a computer by USB port. Inside of it, there are two infrared cameras and multiple LEDs. The LEDs illuminate your hands with infrared light invisible to the human eye [30]. Because of a narrow range of near-infrared spectra at a wavelength of 850 nanometers, they can stand a high range of environments and lighting conditions. LEDs are pulsing to lower the power and increase intensity. The interaction zone is extending from a device in a $140^\circ \times 120^\circ$ and extends from 10 cm to 60 cm. It takes the form of an inverted pyramid for this kind of model [31]. The device has its own local memory and computes the most necessary operations such as resolution adjustments. The real tracking software is then computed on the computer. Detection accuracy is about $200 \mu m$ [32].

Leap Motion Controller software for bone recognition provides better hand tracking with every version. The current detection version Orion does not provide gesture detection itself. Respective gesture recognition is described in the Ch. 5.

Leap Motion Controller does not use point clouds to compute bone structure. Point clouds are substituted for multilevel algorithms instead.

■ 4.1.2 Kuka iiwa LBR manipulator

The IIWA LBR is an industrial seven degree-of-freedom (DOF) serial manipulator (see Fig. 4.1) intended for human-robot collaboration [33]. Internal joint torque sensing is provided, the robot is constructed with safety features in mind to be able to operate with humans in the robot reach area.

■ *Capek* workspace

The robot configuration and environment is using the *Capek* workspace. The workspace is available on GitLab. It manages the communication with the robot and tools for its control. The whole system presented in Ch. 5 is therefore connected and ready to execute on the real manipulator version.

■ Modelling the scene objects

The objects for all scenes were modelled on *Sketchup* software [34]. The models include a drawer with the shell and three inner sockets and the button inner part and its outer shell.

■ ROS

Robot operating system (ROS) are open source libraries and tools for working with a robot manipulator and are used in the implementation of the whole system. ROS incorporates drivers, state-of-the-art algorithms, and powerful developer tools. For communicating between nodes, ROS messages are used (some major definitions of ROS messages are in Ch. B).

ROS toolkit MoveIt! is used to help with some robot operations and functions. Some scene interface functions such as attach item and detach item are used from the toolkit.

4.2 Modeling of probabilistic network

4.2.1 Probabilistic package *PyMC3*

PyMC3 is a probabilistic programming package for Python that allows users to fit Bayesian models using a variety of numerical methods, most notably Markov chain Monte Carlo (MCMC) (see Sec. 3.4.4) and variational inference (VI) (see Sec. 3.4.3). Its flexibility and extensibility make it applicable to a large suite of problems. Along with the core model specification and fitting functionality, *PyMC3* includes functionality for summarizing output and for model diagnostics [35].

The main competitor to *PyMC3* package is *Stan* [36] which offers modelling of probabilistic networks as well. *PyMC3* and *Stan* belongs to the most modern tools to construct and estimate probabilistic models. It is based on *Theano* which optimizes the performance of computation and can utilize computer GPU. The main objective of these methods is to generate the inference from specified probability models.

4.2.2 ProMP

Probabilistic movement primitives have a lot of interpretations with different functions and approaches. For our situation, it is best to stick to a more verified and valued interpretation by Alexandros Paraschos [17], implemented by Michael Mathew [37]. The Probabilistic model for representing the trajectory distribution that is based on a basis function representation is defined in S. 3.5.

4.3 How to read Confusion matrix

To evaluate the quality of the gesture detection, we need a measure which will tell us how accurate the classification is. For this purpose were created confusion matrices.

Confusion matrix is created from an array of predicted gestures and an array of real gestures. These arrays are converted to a matrix with the length of the number of gestures, where one axis denotes the predicted gestures and the other real gestures. Example confusion matrix is shown in Fig. 4.2, where accuracy of classifying individual animals is shown. For example, the cell (1,3) says that in one case (6.67%) a actual bird is recognized as a dog. The last column says in how many cases were individual animals predicted, the last row says in how many cases were the actual animals appearing (e.g., a dog was actually appearing in 6 cases, but detected only in 5 cases). The

overall accuracy of the detection is shown in the bottom right corner, in this case it is 60%, and is computed as a sum of the accuracies on the diagonal.

Confusion tables are generated with tool from `pretty print` [38].

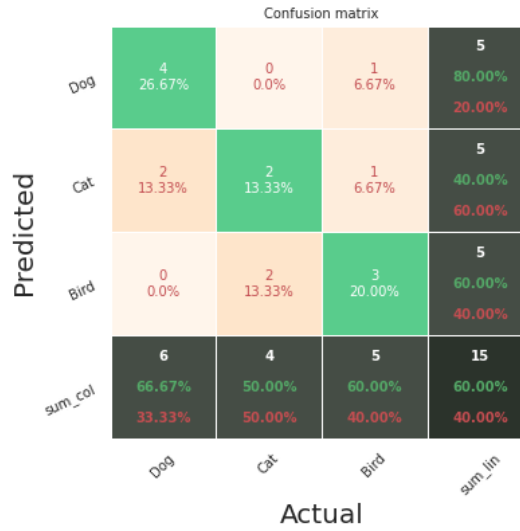


Figure 4.2: Example confusion matrix. Read from left to right. Predicted two samples of cat, when the actual object is dog, wrongly predicted. Predicted four samples of dog when actual object is also dog, correctly predicted.

Chapter 5

System overview

This chapter introduces the whole system top-down. It gradually analyzes the data flow and the processing steps. Later chapters will then describe the individual processing steps. The system processes can be displayed in Fig. 5.1.

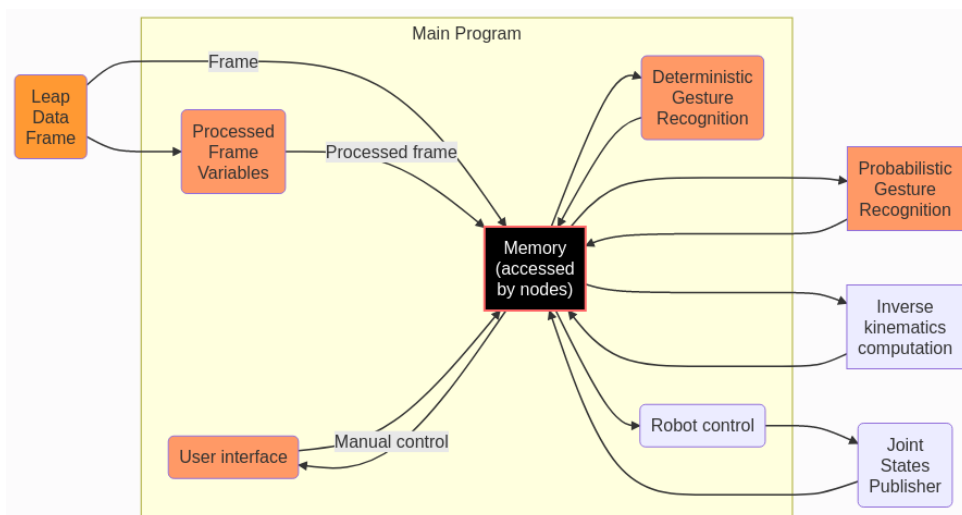


Figure 5.1: Dataflow Diagram, individual processes are described in S. 5.1. Input modules are in orange. Developed modules are in red and used processing and output modules are in blue color. *Memory* block in black color is not a thread. It is a shared space memory that the nodes are interacting with, all data are stored here. Hand data are received from *Leap Data Frame* block which illustrates *Leap Motion API* and serves as an input. Then data are processed in *Processed Frame Variables* block. Each *Deterministic and Probabilistic gesture recognition* has its own computational thread. *Main program* nodes are individual threads inside the program. Nodes outside *Main program* block are *ROS nodes* and communication is done via *ROS messages*. The main ones are noted on the connections, the connection names are the passed data names.

5.1 Setup overview and dataflow

Fig. 5.1 shows the individual components of the system and the data passed between the modules. There is no main component in the system. Every module has its own thread which serves its own purpose. The system can be summarized as broadcasting the Leap data (*Leap Motion Frame*) through processing (*Processed Frame Variables*), saving the data in storage database (*Memory*), computing the inverse kinematics (*Inverse kinematics computation*) and passing the data to robot (*Robot control* and *Joint States Publisher*).

All these components are described in detail below in S. 5.1.

Leap Data Frame node

This node takes the data from *Leap Motion API*. API is installed on computer and takes data from Leap Motion hardware device and computes the bone structure as the frame. The *frame* data (see Tab. 5.1) is the output of this block. These data are outputted in the block with a frequency between 90 to 110 frames per second. It depends on a user CPU power.

Processed Frame Variables node

This block takes the *frame* data (see Tab. 5.1) as an input and processes it as shown in S. 5.2. Function prepares the data for gesture recognition. The output of this block is *processed frame* (see Tab. 5.2) and it is saved in share *Memory* block. This computation step is executed at the same frequency as *Leap Data Frame* node (90 – 110 Hz) and takes only few μ seconds.

Robot control node

The *Capek* workspace (see Ch. 4.1.2) is used in this node. The input are joint space variables (see Tab. 5.6) of robot configuration. The function of this node is to compute the control values that are sent to robot (*Joint States Publisher*) as Joint Trajectory message (see Tab. B.3).

Deterministic and Probabilistic gesture detection modules

These modules perform the calculation of the detection. The input is *Processed Frame Variables* (see Tab. 5.2). The outputs are Tab. 5.3 for deterministic approach and Tab. 5.4 for probabilistic approach.

Inverse Kinematics module

The input for this module is the goal pose (see Tab. 5.5). Output is joint space variables (see Tab. 5.6). This module performs the inverse kinematics task on data that are placed into shared memory and returns the results back to the shared memory. The task is performed with *Relaxed IK* (see Ch. 3.1.1) in realization.

■ Memory

It is the only node which is not a thread but only a shared memory. All data are saved in this block. Saving and loading of variables to the shared space are *atomic* processes, therefore threads are safe and cannot disturb each other. Data of frames and processed frames are saved as a ring buffer to keep the memory usage limited.

It has determined **Buffer length**, which can be changed depending on the situation. The default capacity is 300 variable units, which is approximately equivalent to 4 seconds of *Frame* data depending on CPU configuration.

Based on the use-case, the **Buffer length** can be adapted. For example, extend the buffer when longer periods of data are required to store. One example can be the detection of gestures with long duration. The **Buffer length** can be extended based on the computer RAM volume.

■ 5.1.1 Program configuration and communication between nodes

Capek workspace (see Ch. 4.1.2) is using the language *Python 2.7*, therefore all nodes except probabilistic learning are using Python version 2. Probabilistic learning with *PyMC3* is using *Python 3*, therefore one node is using the newer version. This node is communicating with others via *ROS* messages.

Data contents are the names defined in Fig. 5.1 as the connection names. Now follows the description of each of the data content.

■ Frame

This data flows from *Leap Data Frame* node and *Processed Frame Variables* node. Its contents are shown in Tab. 5.1. They are received values from *Leap Motion API*.

Name	Type
Hand Bone structures $(\mathbf{S})_{n=1}^H$	Float[] (see S. 3.3)
Hand id $\mathbf{h}_{id} = (h_{id})_{n=1}^H$	Integer[]
Hand confidence $\mathbf{h}_{conf} = (h_{conf})_{n=1}^H$	Float[]
Time of recording t_{rec}	Timestamp (see Tab. B.2)

Table 5.1: *Frame* \mathcal{F} definition. Note that *Bone structure* is extracted from frame, other variables (e.g. infrared picture omitted). H denotes number of hands visible by Leap Motion and [] denotes a vector specification.

■ Processed Frame

This data is the output of *Processed Frame Variables* node and are saved to the shared *Memory* block. The *deterministic and probabilistic gesture detection* modules takes this data from the shared memory as an input.

5.2

Name	Type
Hand visible h_{vis}	Boolean
Fingertips distances \mathbf{d}_{norm} (see Eq. 5.11)	Float[], len. 30
Fingers, wrist angles \mathbf{ed} (see Eq. 5.7, Eq. 5.7)	Float[], len. 42
Cartesian hand velocities \mathbf{vp}_{max} (see Eq. 6.26)	Float[], len. 3
Palm pose $P_{pose} = \{P_p, P_o\}$	Pose (B.1)
Time of last stop t_{stop} [s]	Float
Fingertip Cartesian positions $P_{f \in \{1,2,3,4,5\}}^3$	Float[], len. 15
Time of recording t_{rec}	Timestamp (description, B.2)

Table 5.2: *Processed frame* \mathcal{P} definition, saved twice for left and right hand, other hands are discarded, [] denotes a vector specification.

Recognition outputs

Outputs from *deterministic and probabilistic gesture recognition* can be seen in tables Tab. 5.3 and Tab. 5.4. The output from Deterministic approach is not a *one-hot encoding*, because two or more gestures can be on at the same time. Deterministic gesture recognition can be performed with a frequency higher than 30 Hz. It is less computationally demanding than the probabilistic one which is performed strictly with frequency 10 Hz. The same for static gestures as for the dynamic ones.

Name	Type
Gesture toggle $\mathbf{g} = (g)_{n=1}^G$	Boolean[], len. G
Index of highest gesture probability	Integer

Table 5.3: Deterministic gesture recognition output, where G is number of gestures, [] denotes a vector specification.

Name	Type
Gesture probability $\mathbf{g} = (g)_{n=1}^G$	Float[], len. G
Index of highest gesture probability	Integer

Table 5.4: Probabilistic gesture recognition output, [] denotes a vector specification.

Inverse kinematics communication

The *inverse kinematics* block is communicating between the main memory block. Inverse kinematics topic operates with strict frequency of 10 Hz. Input property of the inverse kinematics block is given in Tab.5.5 and output properties are given by 5.6.

Name	Type
Goal pose for robot end-effector	Pose (see Tab. B.1)
Time of goal pose	Timestamp (see Tab. B.2)
Sequence number s	Integer

Table 5.5: *Inverse kinematics* node input. One sample of stream of close end-effector poses.

Name	Type
Joint states (7 DOF) $\mathbf{j} = (j)_{n=1}^7$	Float[], len. 7

Table 5.6: *Inverse kinematics* node output. One sample of stream of Joint states for given robot configuration, [] denotes a vector specification.

■ Robot control communication

The output of *Joint States Publisher* block are real, resp. simulated, variables such as joint state, velocity, and effort for each joint, see Tab. B.4. Robot control publishing operates with strict frequency of 10 Hz.

■ 5.2 Processing Leap Controller data

Data from *Leap Data Frame* node (see. Tab 5.1), the Leap Controller API, are received by performing the inner Leap motion algorithm of advanced detection, similar to the algorithms from Ch. 3.2.

The data received includes sequence of *Frame* data structure (see Tab. 5.1) that contains *Bone Structure* (see Ch. 3.3). To summarize, array of hands is received with frequency around 100 Hz. Each hand detection consists of its own bone structure \mathbf{S} , which includes position for every finger's bone joint P_f^b , and positions of wrist P_w and palm P_p .

Following processes in this section takes place in *Processed Frame Variables* block, note again that every point has Euler coordinates $P^* = (x, y, z)$, the index will be omitted for simplicity.

Firstly, from bone positions, direction vectors $\mathbf{dir} = dir_f^b = (dir_f)_{n=1}^4$ are computed:

- Wrist to Palm $dir_w = |P_w - P_p|$ (5.1)

- Palm to Metacarpal bones $dir_f^1 = |P_f^1 - P_p|$ (5.2)

- Metacarpals to Proximal bones $dir_f^2 = |P_f^2 - P_f^1|$ (5.3)

- Proximal to Intermediate bones $dir_f^3 = |P_f^3 - P_f^2|$ (5.4)

- Intermediate bones to Distal bones $dir_f^4 = |P_f^4 - P_f^3|$ (5.5)

Lengths of the vectors are: $len(dir_w) = 3$, $len(dir_f^b) = 60$.

The transform of direction vectors to Tait–Bryan angles can be seen in Eq. 5.6, Eq. 5.7 and Eq. 5.8.

The angle calculations are needed, because gesture detection operates on the angle differences.

Whole vector of angles can be written as $\mathbf{e} = e_f^b, f = (1, 2, 3, 4, 5), b = (1, 2, 3, 4)$.

$$e_{f1}^b = 0, \quad (5.6)$$

$$e_{f2}^b = \arcsin(-dir_{fy}^b), \quad (5.7)$$

$$e_{f3}^b = \arctan\left(\frac{dir_{fx}^b}{dir_{fz}^b}\right). \quad (5.8)$$

Angles are labelled as Roll, Pitch, Yaw. These names are based on mnemonics to remember the angle name and its properties. Roll angle is omitted in gesture detection.

5.2.1 Further process of specific variables

The processing still takes place in *Processed Frame Variables* block. The next values are computed from previously passed values. Additional values are computed to serve the purpose of input for gesture recognition. Passed values are direction vectors $\mathbf{dir} = dir_f^b, b = (1, 2, 3, 4), f = (1, 2, 3, 4, 5)$, and Euler angles $e = e_f^b$.

Variables that will be the output of processing are the distances between fingers \mathbf{d} , plus normalized ones \mathbf{d}_{norm} and angle differences between bone fingers \mathbf{ed} .

Distance between finger tips

Distance between finger tips is computed using standard Euler distances in Eq. 5.9.

$$\mathbf{d} = d_f = \sum_{f \in \{0,1,2\}} (dir_f^4)^2. \quad (5.9)$$

where dir_f^4 are the direction vectors of the last two finger bones as described in Eq. 5.5.

Normalize distance with normalization values K_1, K_2 . The parameters K_1, K_2 were obtained from experimental tuning with method *grid search* in order to have normalized values as in Eq. 5.11, \mathbf{d}_{norm} (5.10) is vector of normalized distances got from $\mathbf{d}_{norm} = f(\mathbf{d})$.

$$f : d_{norm} = d \cdot K_1 + K_2. \quad (5.11)$$

Number of normalized distances is the length of the combination between fingers, where F is number of fingers and k is length of subset, which is assigned as $k = 2$, see Eq. 5.12.

$$C = C_k^F = \binom{F}{k} = \frac{F!}{k! \cdot (F-k)!} = \frac{5!}{2! \cdot (5-2)!} = 10, \quad (5.12)$$

therefore, the list of distances is $len(\mathbf{d}) = C = 10$. Every two-finger combination has its own index on which the value is accessed.

■ Angle differences between finger bones

First interpretation is using a single subtraction for each bone angle from each previous bone. Metacarpal bone is subtracting angle between the metacarpal bone and palm position as in Eq. 5.13.

$$\mathbf{ed} = ed_f^b = |e_f^b - e_p^{(b-1)}|. \quad (5.13)$$

The second interpretation converts the angles to quaternion form and performs a multiplication between each other, resulting in a constant angle difference, which is a useful input for gesture detection, as done in Eq. 5.14.

$$\mathbf{edq} = edq_f^b = q(e_f^b) \cdot q(e_p^{(b-1)}), \quad (5.14)$$

where the function q transforms Euler angles to quaternion form. Function `quaternion_from_euler` from the transformation package for ROS [39] is used. Output is a single value describing the difference between two angles.

■ Final list of values as input for gesture recognition

Recapitulation as a list of variables processed in this section as an input for gesture recognition, part of *Processed frame* content (Tab. 5.2).

- Normalized position distances \mathbf{d}_{norm} (Eq. 5.10)
- Angle differences \mathbf{ed} , \mathbf{edq} (Eq. 5.13, Eq. 5.14)
- Palm pose $P_{palm} = \{P_{palm}, P_o\}$ (B.1)
- Fingertip positions $P_{f \in \{1,2,3,4,5\}}^3$ (chap. 3.3)

Additional checks that must be performed:

- The hand must be the same (same \mathbf{h}_{id}). When a new hand gets visible, it receives a new id, this id is the same as long the hand stays visible.
- The hand detection must have greater confidence than the set confidence threshold. $h_{conf} > T_{conf}$, where T_{conf} is a confidence threshold. Confidence is received from Leap Motion *frame* (see Tab. 5.1) for each hand and signals how well were bones constructed from the images. h_{conf} is a scalar number within an interval 0 to 1.

There is a correlation between confidence and the orientation of a hand. When a hand is turned down to the Leap sensor, the Leap controller will still construct a bone position structure. However, the accuracy will be much lower. Although the controller can predict very well the finger position if they are not seen, it is advised to keep a hand in a direction down to the sensor.

5.3 Robot setup

This chapter describes what procedures were used when setting up the robot configuration. For Robot control see Ch. 3.1.

Description of the robot manipulator can be defined by *urdf* file format which stands for Universal Robotic Description format. It contains descriptions about each joint, its types, axis, inertia constants. With attached 3D bitmaps it can be a complete description of the manipulator. One possible version of *urdf* file for *iiwa* configuration can be used from `lbr_iiwa7_r800-urdf`-package [40].

This description, however, does not include a gripper, therefore there is a need to include it manually. It can be done by adding this snippet to the end of *urdf* file.

```

1 <joint name="eef_joint" type="fixed">
2   <parent link="link7"/>
3   <child link="eef_link"/>
4   <origin xyz="0 0 0"/>
5 </joint>
6 <link name="eef_link">
7 </link>

```

In this configuration, the joint names are therefore:

```
joint_names = ['joint1', ..., 'joint7', 'eef_joint'].
```

The configuration joint name has the identical named ordering. The starting configuration is zero for all joints.

5.4 User Interface

For easy communication of the user with the system, the interface was built (see Fig. 5.2) as a main window application. The app was built using *PyQt5* library, which is Python binding for comprehensive *Qt* library written in cross platform *C++* language and therefore it is compatible on all platforms including *Windows*, *Linux* and even *Android* and *iOS*.

The purpose for making an application was firstly to visualize real-time responses from gesture recognition systems. Secondly, to manually operate the robot control and configuration values, which is difficult to do from the command line.

Application is constructed separately from *rViz* interface, however it is possible to be used as an *rViz* panel plugin in the future.

The description of the developed application can be written with bullet points:

■ Features of the application

- Application can have a custom size, the side panels will adjust to it
- Toggling view - view of the gesture outcomes can be turned off, if the user wants to see only the hand visualization or turned on if the user want to see gesture detection results
- Responsiveness - Application is always connected (if possible) to both *Leap Motion* and robot workspaces and plots live system behaviour

■ Pages of the application

The application consists of the following pages:

- Main page (see Fig. 5.2)
- Configuration page (see Fig. 5.3)

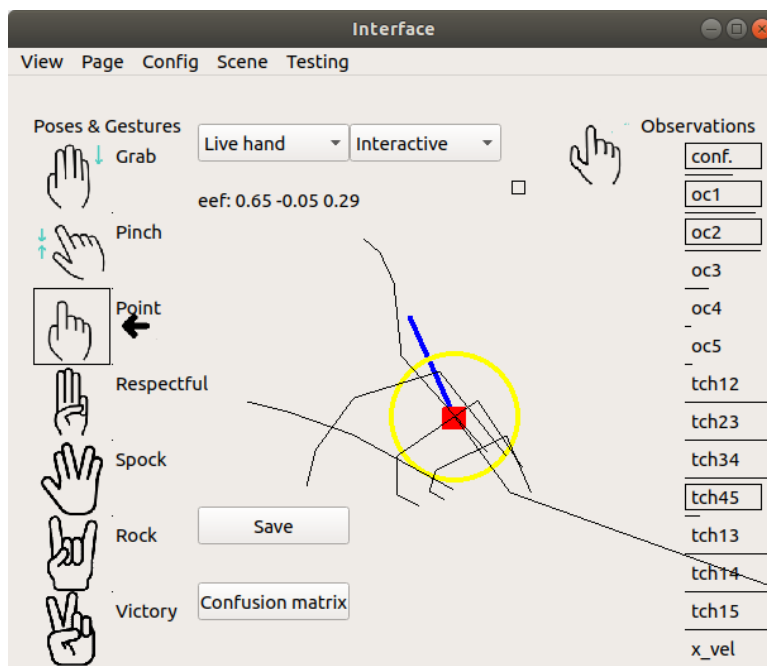


Figure 5.2: Main page of the application.

■ List of functions of the application

- Visualisation of the observations
- Visualisation of the gesture on/off value
- Visualisation of the bone structure

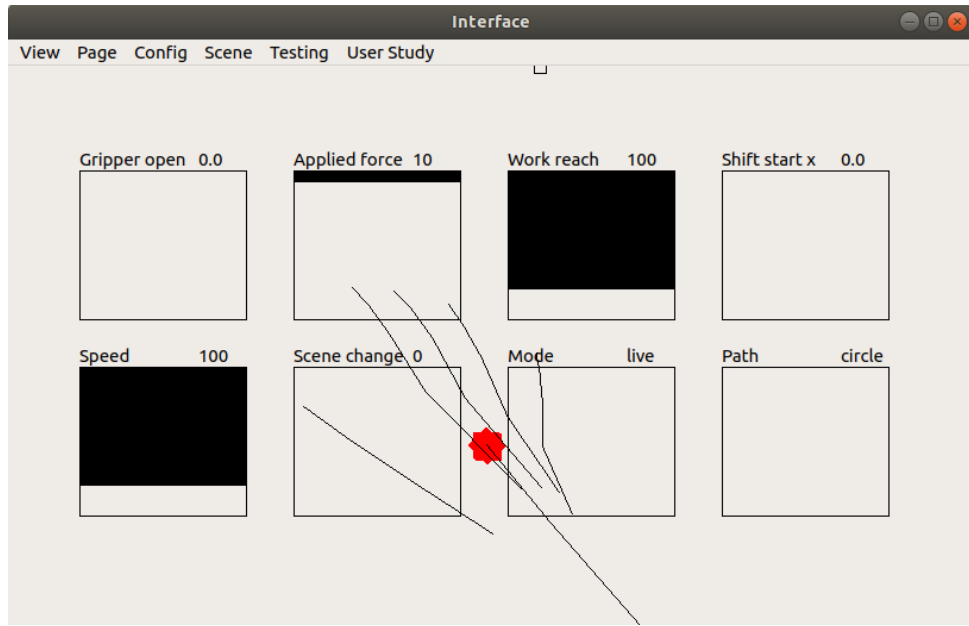


Figure 5.3: Configuration page of the application, you can see adjustable sliders, which are operated by hovering over slider. Sliders can be adjusted to hold value of any property.

- Change between Main Mode, Live Mode and Path
- Play/Stop path execution
- Toggle fixed orientation of the end-effector
- Change scenes
- Change work-spaces
- Record confusion matrix
- Record new datasets
- Inverse kinematics precision test

5.5 Defined manipulation workspaces

The n-DOF robot has a given workspace where it can operate (see Fig. 5.4). Additionally, for our use case, we can define several custom work spaces. In our case, we decided to define three work spaces. They are *above*, *wall* and *table*, see Fig. 5.4. These work spaces are self-defined by maximizing the reach of the manipulator and have the following properties.

Manipulator workspace called *Above* is defined by a rectangle with a starting point $rect_s$ and length $rect_l$ as in Eq. 5.15.

$$rect_s = (-0.35, -0.35, 0.6), rect_l = (0.7, 0.7, 0.55). \quad (5.15)$$

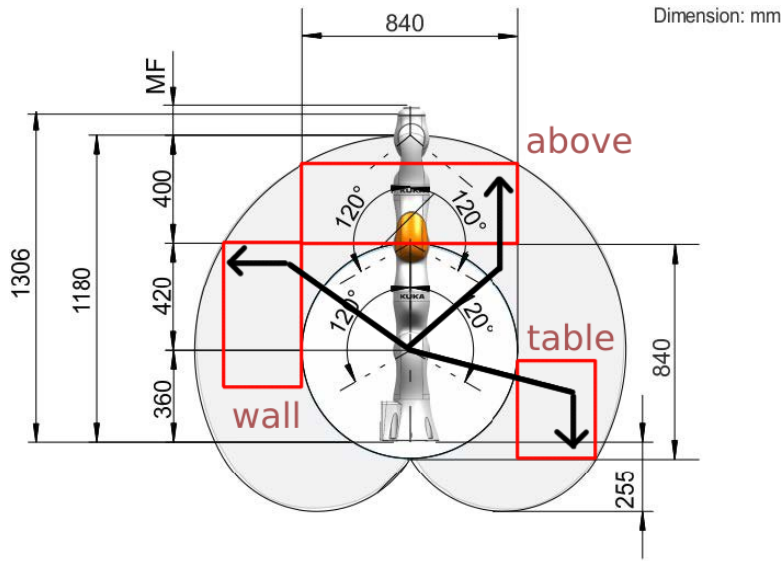


Figure 5.4: Workspace of *KUKA iiwa* [41]. The additional workspaces *wall*, *above*, and *table* are visualised by red rectangles, note that *wall* and *table* workspaces can be on any pose side.

Default orientation of an end-effector for this workspace is defined in Eq. 5.16.

$$x = 0.0, y = 0.0, z = 0.0, w = 1.0. \quad (5.16)$$

Manipulator workspace called *Wall* is defined by a rectangle with a starting point $rect_s$ and length $rect_l$ as in Eq. 5.17.

$$rect_s = (0.4, -0.2, 0.2), rect_l = (0.3, 0.4, 0.55). \quad (5.17)$$

Default orientation of end-effector for this workspace is defined in Eq. 5.18.

$$x = 0.5, y = 0.5, z = 0.5, w = 0.5 \quad (5.18)$$

Manipulator workspace called *Table* is defined by a rectangle with starting point $rect_s$ and length $rect_l$ as in Eq. 5.19.

$$rect_s = (0.4, -0.3, 0.0), rect_l = (0.3, 0.6, 0.6). \quad (5.19)$$

Default orientation of end-effector for this workspace is defined in Eq. 5.20.

$$x = \sqrt{2}/2, y = \sqrt{2}/2, z = 0.0, w = 0.0. \quad (5.20)$$

5.5.1 Leap workspace definition

This is the workspace in which hands can operate, resp. The Leap Motion Controller can see them and it is guaranteed that they will stay in view. This area is called the *Interaction Box* (see Fig. 5.5). Its center is 20 cm from the controller upwards. And its lengths are 23.5 cm, 15 cm, 23.5 cm.

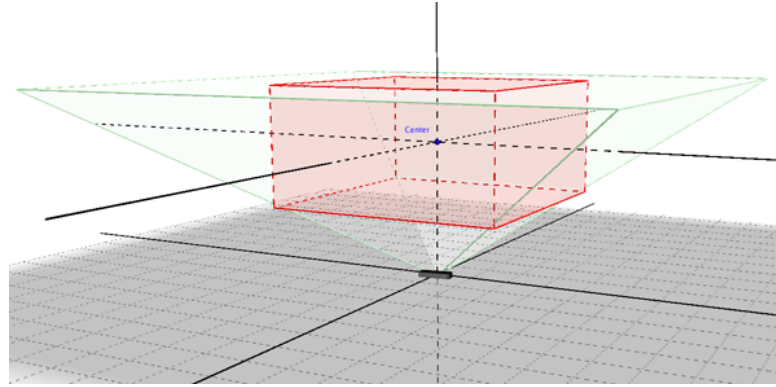


Figure 5.5: Interaction box of Leap Motion Controller [3].

5.6 Dataset

To enable the evaluation of our system, a dataset was created. It was created with emphasis on variability and with no false categorized samples that would deceive the learning process. It serves a purpose only for probabilistic types of recognition.

The main dataset contains around $S \approx 50$ recording samples for every type of $G = 14$ gestures, resulting in more than 600 record samples in the learning dataset. Every sample has a length set to 300 time samples, which is always longer than one second. Every time sample contains the processed time frame of observations.

One recorded a data sample for a previously determined recorded gesture. Therefore, one sample recording is recorded for one specific gesture from the list, where $G_s = [\text{Grab, Pinch, Point, Respectful, Spock, Rock, Victory, Swipe Up, Swipe Down, Swipe Left, Swipe Right, Pin, Touch, Rotate}]$. Recordings are stacked, this is also the order of gestures in the dataset. There is no field where the type of gesture is saved. The type of gesture is determined by the computer directory. with time sample length T consists of one *Processed frame* \mathcal{P} data (see Tab 5.2), therefore: $D_r = (\mathcal{P})_{t=1}^T$.

All recordings for one gesture are $D_g = (D_r)_{s=1}^S$, then the whole learning dataset is: $\mathbf{D} = (D_g)_{g=1}^G$. More about gestures written on Ch. 6.

Gesture recognition module then has an option what values from Processed frame \mathcal{P} will use. Two variations of the \mathcal{P} utilization are created:

1. *User defined* - contains **12 values**, which are deterministically computed: open/close of fingers \mathbf{oc}_f and normalized distances between fingers \mathbf{d}_{norm} (some distances are omitted for simplicity in this option)
2. *All defined* - contains **87 values**, which includes: Wrist to hand angle differences \mathbf{ed}_w , finger angle differences propagated from palm node to the end of a fingers \mathbf{ed} , fingertip differences combinations \mathbf{d}_{norm}
3. *Palm points* P_p - contains **300 values**, trajectory of Cartesian points by time with length 100

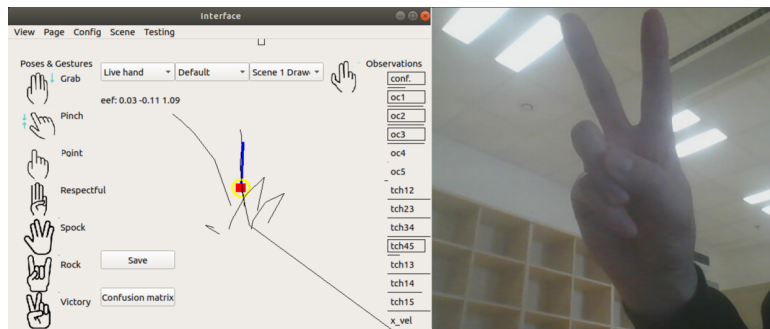


Figure 5.6: Example of dataset recording, note that the enclosed RGB picture is only for illustration from dataset recording process, it is not included in the dataset.

Each sample recording of n seconds is saved in a separate file with the extension *.pkl*, where Python classes can be saved in a raw form. The saving and loading of the file is made with a data serialization package called *Pickle*. Further work would involve saving data to some general form, however, this method can be superior in a way for its simplicity of saving and loading. The possibility to delete any chosen recorded sample is also welcomed.

Chapter 6

Gesture recognition

In this chapter, deterministic and probabilistic approaches to gesture recognition and their processes are described in detail. The experiment processes and their results are also depicted in this chapter.

This chapter concentrates on implementing *Deterministic and Probabilistic gesture recognition* block seen on Fig. 5.1. *Processed frame \mathcal{P}* (see Tab. 5.2) is the input of this block and it can be reduced based on data utilization choice 5.6. The main output of these blocks can be indices of the highest gesture probability. Complete outputs are in Tab. 5.3 for deterministic output and Tab. 5.4 for probabilistic one.

Gesture detection from a processing point of view is represented in Fig. 6.1, every gesture processing has some input configuration. With this configuration, the specific gesture inner function is performed, the output of a function is the output of a given gesture.

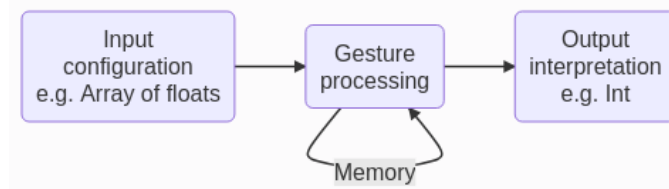


Figure 6.1: Gesture processing diagram.

6.1 Defined gestures









Static gestures were picked as the most (most of the gestures from [42] and few from [43]) recognized gestures throughout the world. The list of chosen static gestures is (see Sec.6.3.1 for more detailed description).

When these gestures are detected, they can trigger any action which would be assigned to them. For example *Grab* gesture picks an object, *Point* gesture switches the view mode or the *Victory* gesture can turn on motor. Gestures are easy to demonstrate and they are perfect for testing detection methods.



From the perspective of robot control, they serve a purpose as semaphoric gestures. They trigger the assigned action of choice and need. There is many

functions of semaphoric functions such as: Pushing the button only when *Point* gesture is turned on. Grabbing the drawer socket only when *Grab* gesture is on.

There need to be clarified that further types of gestures are intended for gesture detection, testing, and variability. Furthermore, there is a possibility to control the robot with them.

- | | |
|--|---|
| ■ Grab  | ■ Spock  |
| ■ Pitch  | ■ Rock  |
| ■ Point  | ■ Victory  |
| ■ Respectful  | ■ Italian  |

Defined dynamic gestures are chosen based on reliability and comfort in performing them. They are based on preliminary testing. See S. 6.3.2 for the detailed description.

- | | |
|---|--|
| ■ Swipe  | ■ Pin  |
| <ul style="list-style-type: none"> ■ Left ■ Right ■ Up ■ Down | ■ Touch  |
| | ■ Rotate  |

In next S. 6.3, we introduce a deterministic approach to detect these gestures from sensor data. Later, in S. 6.4.1, we develop a probabilistic gesture classification pipeline.

■ Detection assumptions

Basics and definitions of gestures (see Ch. 3.2) are used in this chapter. On top of that, some assumptions were made. Difference between data between static and dynamic gestures:

- Static gestures are computed from only one sample (frame) at a time. They capture the pose of a hand in every time sample. There is a need for only one *Processed frame* \mathcal{P} (from Tab.5.2).

- Dynamic gestures are computed within the time horizon. Therefore, all *Processed frames* \mathcal{P} are used in the dataset, $\sim 50-100$ samples per second, more on the dataset in Ch. 5.6.

6.2 Using value measures to evaluate gestures

Because the later interpretations are detecting the only *Semaphoric* type of gesture, in this chapter is presented a short jump to the value type of measures, e.g., the output of the gesture is a real number.

There is an option to detect the gestures using other metrics than the logical values presented in the previous S. 6.3.1. One option is to use value measures, which can provide real numbers as output. They fall into a group of *Manipulative gestures* with one dimension (definition on Ch. 3.2).

An example of implementation is using distances between fingers to set values. In the proposed system, it can be demonstrated by using the distance from the thumb to the index finger $d_{norm,1-2}$ called Pinch (see Fig. 6.2). Values can be for specific purposes normalized, namely, from zero to one. The reason for doing that is the ability to apply the measure to any scale in the future and secondly to make the measure independent of hand size. It is done with Eq. 6.1. The position of the end of the fingers must be precise, therefore for this pose, a rotation downwards of the hand is advised.

$$p^* = \frac{p - p_{min}}{p_{max} - p_{min}}, \quad (6.1)$$

the formula transforms the sensed values to a normalized output, where p can be assigned to the distance from the thumb to the index finger, previously defined in Eq. 5.11, so $p = d_{norm,1-2}$, p_{min} is the minimum distance measured with Leap Motion set by default to $p_{min} = 0[mm]$ and p_{max} is maximum distance measured with Leap Motion set by default to $p_{max} = 100[mm]$ and is updated if bigger number is observed.

As it has been said, for applying the measurement, a certain mode must be active to proceed the user to execute the measurement. This is done to avoid the measurement to be active all time.



Figure 6.2: Pinch measure example.

6.3 Deterministic approach

In this section, we present an engineering approach to gesture recognition. For each gesture, a set of conditions on the sensor data is identified via analysis

of the gesture. If all conditions are met, the gesture is reported as detected. This approach depends on tuning the conditions manually, which is one of its greatest drawbacks. We use the results achieved with the method described in this section as a baseline.

The deterministic approach involves developing a processing and detection intuitively. Weights are tuned with live testing. Weights are tuned by performing a gesture and then marking every observation for a different configuration and then updating the values.

In the next sections (S. 6.3), the values from *Processed frame* are extracted. Features such as *open/close* a finger and finger *touches* are computed. These values will further be used in the detection.

■ Feature extraction and processing

As an input, the *Processed frame* \mathcal{P} is used as defined in Ch. 5.1. To distinguish individual gestures, we perform some operations on top of the data received from *Processed frame*. The basic operation as the output feature is determining the orientation of individual fingers. Orientation meaning of each finger is up or down, as it can be seen in Eq. 6.2, computed from the angle difference between fingers.

$$\mathbf{oc} = oc_f = q(e_f^1) \cdot q(e_f^4), \quad (6.2)$$

where oc_f is number for each finger f , for each finger is kept a number between 0 and 1, where the value of 0 is for the closed finger and the value of 1 is for the open finger. e_f^1 are Euler angles for first bones for each finger and e_f^4 are Euler angles for the last bones for each finger, both computed from the direction vectors in Fig. 6.5a. q is Euler angles to quaternion function described before.

Boolean value for open/closed fingers ocb is expressed as the condition with tuned threshold values and are debounced using a hysteresis (see Fig. 6.4), therefore there is a separated *turn on* threshold and a *turn off* threshold (both *turn on* and *turn off* are used in \mathbf{ocb} and \mathbf{db} features), as it can be seen in Eq. 6.3 and Eq. 6.4. The hysteresis is only considering the value and not the time, because the fingers can move very quickly and there is no intention to slow the process down by the possible time hysteresis.

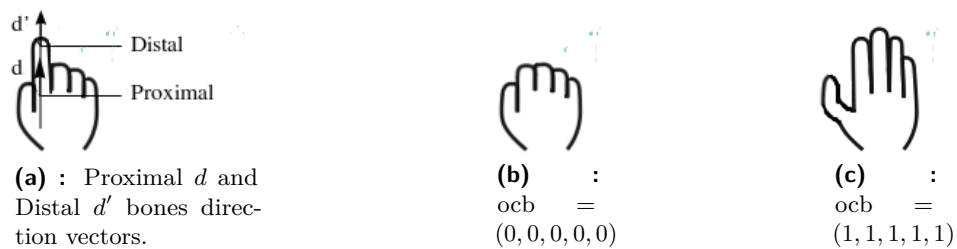


Figure 6.3: The feature rm example demonstration.

$$\mathbf{ocb}_{on} = \text{ocb}_{f,on} = \text{oc}_f \geq \text{OCB}_{f,on}, \quad (6.3)$$

$$\mathbf{ocb}_{off} = \text{ocb}_{f,off} = \text{oc}_f < \text{OCB}_{f,off}, \quad (6.4)$$

where ocb_f is Boolean and OCB are tuned thresholds for every finger.

From the fingertip, the distances of fingertips are computed *touches* between individual fingers. If the touch Boolean db is *True*, then the selected two fingers are touching each other, see Eq. 6.5 and Eq. 6.6.

$$\mathbf{db}_{on} = \text{db}_{f,on} = d_{norm,f} > \text{TCB}_{f,on}, \quad (6.5)$$

$$\mathbf{db}_{off} = \text{db}_{f,off} = d_{norm,f} < \text{TCB}_{f,off}, \quad (6.6)$$

where db is distance Boolean for every finger and TCB_f are tuned thresholds for every distance combination.

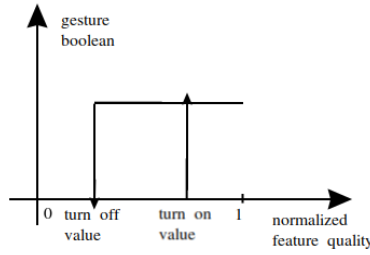


Figure 6.4: Applied Boolean hysteresis.

6.3.1 Static gesture recognition

Here is presented how static gestures are determined using a deterministic approach with the following logical equations. Note that they are *Semaphoric* gestures, which means that in these representations, they are either present or not present, they do not output any other value. In terms of a deterministic approach, the formulation does not need any other measurement.

After performing these operations, we obtained a list of Boolean values for every gesture. If there is a need to determine and pick only one gesture with the most likelihood of happening, the system of priorities can be applied.

All gestures follow the same pattern to switch on and off. $\text{gest}_{on} = \bigwedge_{f \in F_{gest}} f$ and $\text{gest}_{off} = \bigvee_{f \in F_{gest}} \neg f$, where F_{gest} is the set of features selected for gesture detection.

For all static gestures, the orientation of the hand is irrelevant, while it is advised to point towards the sensor due to better accuracy.

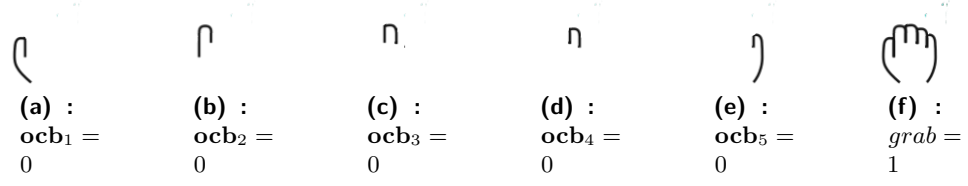


Figure 6.5: Five closed fingers correspond to five features $\mathbf{ocb} = (0, 0, 0, 0, 0)$ resulting in *Grab* gesture.

■ Grab

Logic values for the turn on/off gesture are defined in Eq. 6.7 and Eq. 6.8. For *Grab* gesture \mathbf{ocb} feature (see Fig. 6.5b) containing a logical value for each finger is used, describing the orientation of a finger (see more on S. 6.3). Gesture is turned on if every finger is closed, it is being turned off if one of the five fingers is opened.

$$grab_{on} = ocb_1 \wedge ocb_2 \wedge ocb_3 \wedge ocb_4 \wedge ocb_5, \quad (6.7)$$

$$grab_{off} = \neg ocb_1 \vee \neg ocb_2 \vee \neg ocb_3 \vee \neg ocb_4 \vee \neg ocb_5. \quad (6.8)$$

■ Pinch

Logic values for the turn on/off gesture are defined in Eq. 6.9 and Eq. 6.10. The touch logical values from S. 6.3 are utilized as requirement that the thumb must touch the index finger. The last three fingers must be open for this gesture to be turned on. Thumb orientation is irrelevant for this gesture.

$$pinch_{on} = db_{12} \wedge ocb_3 \wedge ocb_4 \wedge ocb_5, \quad (6.9)$$

$$pinch_{off} = \neg db_{12} \vee \neg ocb_3 \vee \neg ocb_4 \vee \neg ocb_5. \quad (6.10)$$

■ Point

Logic values for the turn on/off gesture are defined in Eq. 6.11 and Eq. 6.12. Intuitively, the index finger must be opened and the last three fingers are closed. Thumb orientation is irrelevant.

$$point_{on} = ocb_2 \wedge \neg ocb_3 \wedge \neg ocb_4 \wedge \neg ocb_5, \quad (6.11)$$

$$point_{off} = \neg ocb_2 \vee ocb_3 \vee ocb_4 \vee ocb_5. \quad (6.12)$$

■ Respectful

Logic values for the turn on/off gesture are defined in Eq. 6.13 and Eq. 6.14. The index, middle, and ring fingers are opened and the thumb with the pinky finger are strictly closed. Orientation towards the sensor is advised for this gesture.

$$resp_{on} = db_{23} \wedge db_{34} \wedge \neg ocb_1 \wedge ocb_2 \wedge ocb_3 \wedge ocb_4 \wedge \neg ocb_5, \quad (6.13)$$

$$resp_{off} = \neg db_{23} \vee \neg db_{34} \vee ocb_1 \vee \neg ocb_2 \vee \neg ocb_3 \vee \neg ocb_4 \vee ocb_5. \quad (6.14)$$

■ Spock

Logic values for the turn on/off gesture are defined in Eq. 6.15 and Eq. 6.16. Thumb is not conditioned in this gesture, the rest of the fingers needs to be opened, while the touches between the index and middle finger as well as the ring and pinky finger must occur and the most important note is that the middle finger and the ring finger cannot touch. This gesture is difficult for some people to execute.

$$spock_{on} = db_{23} \wedge \neg db_{34} \wedge db_{45} \wedge ocb_2 \wedge ocb_3 \wedge ocb_4 \wedge ocb_5, \quad (6.15)$$

$$spock_{off} = \neg db_{23} \vee db_{34} \vee \neg db_{45} \vee \neg ocb_2 \vee \neg ocb_3 \vee \neg ocb_4 \vee \neg ocb_5. \quad (6.16)$$

■ Rock

Logic values for the turn on/off gesture are defined in Eq. 6.17 and Eq. 6.18. The index and pinky fingers must be opened, while they cannot touch. The middle and ring finger must be closed.

$$rock_{on} = \neg db_{25} \wedge ocb_2 \wedge \neg ocb_3 \wedge \neg ocb_4 \wedge ocb_5, \quad (6.17)$$

$$rock_{off} = db_{25} \vee \neg ocb_2 \vee ocb_3 \vee ocb_4 \vee \neg ocb_5. \quad (6.18)$$

■ Victory

Logic values for the turn on/off gesture are defined in Eq. 6.19 and Eq. 6.20. The index and middle finger are opened and cannot touch each other. The last two fingers need to be closed. Thumb position is irrelevant.

$$vic_{on} = \neg db_{23} \wedge \neg ocb_5 \wedge ocb_2 \wedge ocb_3 \wedge \neg ocb_4 \wedge \neg ocb_5, \quad (6.19)$$

$$vic_{off} = ocb_5 \vee \neg ocb_2 \vee \neg ocb_3 \vee ocb_4 \vee ocb_5. \quad (6.20)$$

■ Italian

Logic values for the turn on/off gesture are defined in Eq. 6.21 and Eq. 6.22. This gesture has only conditions and that is, all fingers need to touch each other. This gesture problems are depicted in conclusion 6.3.1.

$$vic_{on} = db_{12} \wedge db_{23} \wedge db_{34} \wedge db_{45}, \quad (6.21)$$

$$vic_{off} = ocb_5 \vee \neg ocb_2 \vee \neg ocb_3 \vee ocb_4 \vee ocb_5. \quad (6.22)$$

Note that in this representation of gesture evaluation, more than one gesture can be *True*.

■ Experimental results and Discussion

The implemented deterministic method is tested on the recorded dataset (see Ch. 5.6), specifically on 7 gestures. The total accuracy of all classified gestures is 84%, as can be seen from Fig. 6.6. The proposed system has some flaws, especially within *Grab* and *Pinch* gesture. More tuning would bring the percentage number up by a few percent, but it is quite unlikely to reach the 100% accuracy mark on the whole dataset.

In this section, we present a hand-crafted gesture detection pipeline based on the deterministic evaluation of the features observed by the Leap Motion sensor. The results show that although we can detect many gestures in general, the system is not very reliable. For example, the *Grab* gesture is often confused with the *Pinch* gesture and the *Point* gesture is also confused with *Victory* gesture. Especially, when the data returned by the sensor is incomplete, the gesture detection precision quickly deteriorates. To seek better accuracy, probabilistic approaches are introduced in Ch. 6.4.1.

In the process, we stumbled across a problem with Italian gestures. The problem is that the gesture is performed with fingers upwards, and because the sensor is mounted on the table, the gesture cannot be used, the gesture was discarded in later applications, but it will eventually comeback if the mounting position of the sensor would change to ceiling.

The Leap Motion detection of bones and hands relies heavily on the ability to recognize the hand. When recognition is wrong, the bones of the fingers are not detected properly, therefore detection of most of the gestures will fail. This is completely understandable. Two cameras are only a few centimeters apart and cannot detect every angle. If the cameras were longer from each other, the detection would and could be better in some circumstances.

When the hand has a downward direction, the position of the bones is proper. When the position is different, the position of bones can differ with more than 4 cm apart. The values will not converge. The hand must "shake" to regain a proper bone position.

Originally, it was intended that the *Pinch* gesture is realized as a contact between the thumb and the index finger. As this was colliding with *Grab*

Confusion matrix

Predicted	Grab	41 14.29%	1 0.35%	1 0.35%	10 3.48%	8 2.79%	0 0.0%	0 0.0%	61 67.21% 32.79%
	Pinch	0 0.0%	40 13.94%	0 0.0%	14 4.88%	0 0.0%	2 0.70%	8 2.79%	64 62.50% 37.50%
	Point	0 0.0%	0 0.0%	40 13.94%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	40 100% 0.00%
	Respectful	0 0.0%	0 0.0%	0 0.0%	16 5.57%	0 0.0%	0 0.0%	0 0.0%	16 100% 0.00%
	Spock	0 0.0%	0 0.0%	0 0.0%	0 0.0%	33 11.50%	0 0.0%	0 0.0%	33 100% 0.00%
	Rock	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	39 13.59%	0 0.0%	39 100% 0.00%
	Victory	0 0.0%	0 0.0%	0 0.0%	1 0.35%	0 0.0%	0 0.0%	33 11.50%	34 97.06% 2.94%
	sum_col	41 100% 0.00%	41 97.56% 2.44%	41 97.56% 2.44%	41 39.02% 60.98%	41 80.49% 19.51%	41 95.12% 4.88%	41 80.49% 19.51%	287 84.32% 15.68%
		Grab	Pinch	Point	Respectful	Spock	Rock	Victory	sum_lin
		Actual							

Figure 6.6: Static gesture recognition via Deterministic approach. Confusion matrix shows the results for each of 7 compared static gestures from main dataset (5.6). On the results, total accuracy of properly classified gestures is 84%.

gesture and others, the configuration was edited to more similar as *OK gesture*.

For maximum accuracy, the orientation of the gesture needs to be defined. The reason for this can be observed in humans. When a person is showing a gesture to another person, the orientation of the hand needs to be determined, so the other person knows that the gesture concerns him, similarly, the Leap Controller can act a similar way and respond only to gestures that are oriented towards it.

6.3.2 Dynamic gesture recognition

The dynamic gesture introduces a time variable in detection. Recorded path trajectory, or multiple trajectories are evaluated to detect a given gesture from a determined set of gestures. The time variable extends the list of possible outcomes of the gesture.

Dynamic gesture evaluation uses reduced feature space to show simpler examples. To be more specific, for this thesis, only the *Palm point* is used. P_p (the center of the palm position) and index finger position P_2 , the data are from *Processed frame* in Tab. 5.2.

Dynamic gestures can be various types, for example, the execution gestures can be *Stroke* movement, *Circle* rotating movement, *Touch* with fingers movement, etc.

Direction vectors can serve the function of assigning the direction to execution of *Stroke/Swipe* gesture. The size of the velocity can be conditioned, for example, if the velocity of a human palm is greater than a given threshold, a different action is triggered than when the velocity is lower. Moreover, the rotation of the hand in time can be used to trigger some actions, for example, rotating the end-effector towards the rotation (see implementation in Ch. 7.1.6).

To simplify the explanation of individual gestures, in this section, we introduce new variables included in *Processed frame* \mathcal{P} (5.2).

- $pp = P_p = (x, y, z)$ - position of palm (from \mathcal{P})
- $pf = P_2 = (x, y, z)$ - position of pointing finger (from \mathcal{P})
- $vp = \mathbf{vp} = (x, y, z)$ - velocity of palm (from \mathcal{P})
- $vf = (x, y, z)$ - velocity of pointing finger (computed in *Deterministic Gesture recognition block* in Fig. 5.1)

■ Swipe in direction

Swipe gesture can be described as a stroke with the hand across space in any direction when a certain velocity is needed. In this section, only the on/off value for this gesture is written. This gesture can be coupled with a given velocity and orientation to incorporate a more complex gesture.

We define the direction of a movement as a main axis. We define side axis the other two axes of direction in Eq. 6.23.

$$main = (x, y, z), side = ((y, z), (x, y), (x, y)). \quad (6.23)$$

Very simple implementation is to evaluate the velocity of a main axis compared to the side axis. We start with the palm position trajectory, which has T time samples, defined in Eq. 6.24.

$$\mathbf{pp} = (pp)_{t=1}^T, pp = (x, y, z). \quad (6.24)$$

The velocity is computed by the derivation of position with respect to time. Time series is received from the *Timestamps*. Finite differences are used to approximate $\frac{d\mathbf{pp}}{dt}$. *Numpy* function `diff` can be used.

$$\mathbf{vp} = \frac{d\mathbf{pp}}{dt}. \quad (6.25)$$

Get the maximum velocity in each axis as in Eq. 6.26.

$$vp_{max} = \max(\mathbf{vp}), \quad (6.26)$$

then it is compared the maximum of each axis with the tuned threshold, as in Eq. 6.27. There is the rule that comparing the main axis needs to have a velocity bigger than a certain threshold value and the side axis needs to have a threshold lower than a certain value.

$$swipe_{main} = vp_{max,main} > T_{main} \wedge vp_{max,side1} < T_{side} \wedge vp_{max,side2} < T_{side}, \quad (6.27)$$

where *swipe* is a Boolean value and *T* are tuned thresholds.

This same method is applied to rotating in multiple directions, only distances are substituted with angles.

■ Finger touches

This gesture interprets touching the virtual screen ahead of a user. It is similar to the swipe gesture with few exceptions. Firstly, the gesture is quickly compared to the swipe. This is represented with different thresholds of multiple velocities for gesture activation. Secondly, the static gesture *Point* must be enabled when proceeding with this dynamic gesture, and thirdly, the gesture has a stop point in the middle, therefore the velocity vector includes one value near zero in Eq. 6.28.

$$stopped = \min(|\mathbf{vp}|) < sst, \quad (6.28)$$

where *stopped* is Boolean and *sst* is steady state threshold is determined by observing the velocity of the static hand pose. Euclidean norm of velocity is computed in Eq. 6.29. Then touch Boolean is computed in Eq. 6.30.

$$vp_{max,*} = \max_{i \in [0..T]} \|\mathbf{vp}_i\|_2, \quad (6.29)$$

$$touch = point \wedge stopped \wedge vp_{max,*} > T_{main}. \quad (6.30)$$

■ Finger pin

Finger pin represents the movement of the pointing finger as if the finger would play on the piano. Note that the palm position is not moving and only the point finger does the movement downwards. L2 norm of velocity is computed in Eq. 6.31 and then Boolean of gesture in Eq. 6.32.

$$vf_{max,*} = \max_{i \in [0..T]} \|\mathbf{vf}_i\|_2, \quad (6.31)$$

$$pin = \prod_{n=1}^3 vp_{max,n} < sst \wedge vf_{max,*} > T, \quad (6.32)$$

where *sst* is steady state threshold, *T* is the tuned threshold.

Circle rotation

Circle rotation of the palm position would be described by the palm trajectory. However, there is no need for path extraction, because there is an option to extract data about rotation movements conveniently from Leap Controller. Rotation information of the data *frame* F contains the radius of the performed circle and its progress.

The rotation direction of the circle is calculated by comparing the circle normal angle n with circle direction angles d with condition $\arccos \frac{n \cdot d}{|n| \cdot |d|} \leq \frac{\pi}{2}$. The condition calculates the difference between two angle orientations.

Experimental results and Discussion

Results of dynamic gestures are provided in User Study in Ch. 7.3.

The experiments were performed by 7 people, dynamic gestures 4 times. They were asked to perform 4 types of gestures. The names are: *Rotate*, *Swipe*, *Touch* and *Pin* gestures. The most accurate was the *Rotate* gesture and the least accurate was the *Pin* gesture. The accuracy is also linked to user ability in operating these actions. The users which were experienced with computers were having higher accuracy. Even though the average accuracy for all performed gestures were only 71%, the model can be used in real situations when the *Pin* and *Touch* gestures are discarded from the set.

Person no.	Accuracy	Gestures detected	
1	61%	Rotate	Pin
2	82%	Rotate, Swipe, Touch	Pin
3	72%	Rotate, Swipe	Pin
4	55%	Rotate,	Touch
5	84%	Rotate, Swipe	Pin
6	69%	Rotate, Swipe	Pin
7	74%	Rotate, Swipe	Pin
mean	~ 71%		

Table 6.1: User study, dynamic time gestures deterministically, 7 people were performing 4 gestures at least 4 times. Average total accuracy of model is ~ 71%

6.4 Probabilistic approach

In this chapter, there are presented approaches to solve the gesture detection probabilistically. Firstly, we propose a probabilistic neural network model for detecting static gestures, and then the Probabilistic Movement Primitives are used to help with dynamic gesture detection which is solved using Euclidean distance and Dynamic Time Warping. Finally, in the chapter is presented conclusion about the results.

6.4.1 Static gestures recognition

The results from the deterministic approach were decent, with a total accuracy of 85% on whole dataset for static gestures, but we seek for a higher robustness and learnability of gestures.

For a toolbox, *PyMC3* is used to model probabilistic neural networks as described in Ch. 4.2.1. The proposed model is based on Bayes' theorem (see Eq. 3.11). We intended to use 2D neural networks only, therefore the problem must be simplified to fit the 2D space, that is the reason to restrict gestures only to static ones in this section.

Because only static gestures are recognized and all recordings in dataset (5.6) are *Processed frames* to time-series. The ways how to treat the time series are represented in the next S. 6.4.1. After that, the proposed network layout is proposed and its specifications described. Next, the experiments and results are presented.

Dealing with time-series

When computing static gesture recognition, there is no need for time series. What time value for the static configuration will be used? There are few options.

1. Pick the middle sample in the time-series in presumption to have the best gesture configuration possible.
2. Average all the configurations. Even more advanced method would be discarding the start and the end of the time-series by volume of 10 %.
3. Use multiple configurations of time-series by adding them as new samples. This can radically expand the dataset. Drawback of adding every configuration in time-series is, that data one sample to each other are not radically different. Therefore was decided to use every 10th time-sample as the new configuration.

Network layout

The model is composed of multiple layers. Each layer has a certain number of random variables. The model is fully connected. The model is using function `tanh` and `sigmoid` as an activation function. Drop-out regularization is not presented in this model.

Input embedding

Network input variables X, y . Variable X is a 2D array, where the first dimension denotes the number of samples and the second dimension is the number of observations, which describes one observation. The number of observations is the input of the input layer. They have to have the same dimension. Variable y is a 1D integer array with the length of the number of samples and corresponds to the true gesture ID in each sample.

■ Output embedding

Classification block is used for output embedding. It is integer-based. Therefore, using a discrete distribution. Categorization block is using Eq. 6.33.

$$f(x|p) = p_x, \quad (6.33)$$

where $p \in \{0, 1, \dots, |p| - 1\}$, $p > 0$ and the elements of p must sum to 1. Otherwise, they need to be rescaled.

■ Two layer model

Diagram of two-layer model is shown in Fig. 6.7 with further description of the layer weights and their probabilistic types in Eq. 6.34, Eq. 6.35 and Eq. 6.36.

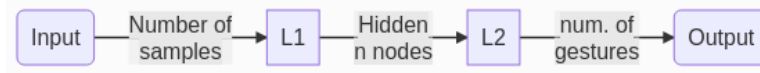


Figure 6.7: Layers of neural network model. The dataflow through nodes demonstrates how big the flow through nodes is. *Number of samples* is number of total samples in the dataset. *Hidden n nodes* is variable describing number of random variables between layers, *num. of gestures* is number on output, how many categorized gestures are we detecting.

$$weights_1 \sim Normal(\mu = 0, \sigma = 1.0), \quad (6.34)$$

$$weights_2 \sim Normal(\mu = 0, \sigma = 1.0), \quad (6.35)$$

$$out \sim Categorical(p = f(f(nn_input, weights_1), weights_2)) \quad (6.36)$$

where *Normal* indicates the self-titled type of distribution with initial parameters, which are μ as mean value and σ as standard deviation. f is *Categorization* block function (defined above 6.33) it is computed as $f(x|p)$, where $p \in \{0, 1, \dots, |p| - 1\}$.

■ More layer models

Models with more layers using the above-mentioned configuration were built using the previous two-layer model and a determined number of weight variables and an activation function was added.

■ Example code of two layer model

For closer examination and seeing the model how it really is, it is ideal to include the neural network model written in Python language. An example of model creation for two-layer neural network can be seen in Lst. 6.1. Implementation of the network learning is shown in Lst. 6.2.

```

1 l1_prior = np.random.randn(X.shape[1],
2                             n_hidden[0]).astype(floatX)
3 l2_prior = np.random.randn(n_hidden[0],
4                             out_n).astype(floatX)
5
6 with pm.Model() as neural_network:
7     nn_input = pm.Data("nn_input", X_train)
8     nn_output = pm.Data("nn_output", Y_train)
9
10    weights_1 = pm.Normal("w_in_1", 0, sigma=1,
11                          shape=(X.shape[1], n_hidden[0]), testval=l1_prior)
12    weights_2 = pm.Normal("w_2_out", 0, sigma=1,
13                          shape=(n_hidden[0], out_n), testval=l2_prior)
14
15    act_1 = pm.math.tanh(pm.math.dot(nn_input, weights_1))
16    act_2 = pm.math.sigmoid(pm.math.dot(act_1, weights_2))
17
18    out = pm.Categorical("out", act_2,
19                        observed=nn_output, total_size=Y_train.shape[0])

```

Listing 6.1: Two-layer layout of probabilistic neural network written in Python using *PyMC3* package. Firstly, the prior distribution values are created as arrays of random variables. Network, the *PyMC3* model includes data for training *X_train* with their true value indices *Y_train*. Layer weights are created as a array of Normal distributions. Activation functions are created on top of each weight with *tanh* and *sigmoid* types. Output is *Categorical* block with the size of the total number of gestures. Notice that the variables *act_1*, *act_2*, *out* are connected to each other in series.

```

1 with neural_network:
2     inference = pm.ADVI()
3     approx = pm.fit(n=30000, method=inference)

```

Listing 6.2: Learning the framework as fitting with *fit* function from *PyMC3* package. Chosen method Variational Inference *ADVI* with iterations *n*.

■ Experimental results and Discussion

The probabilistic method was evaluated on 7 static gestures from the main dataset (Sec. 5.6). Configuration of dataset (see Sec. 5.6) varies between *User defined* and *All defined*. To be specific, the used dataset has around 300 recorded samples (~ 50 samples for each gesture). Note that all experiments have 30000 iterations, if not told otherwise. Example view of the dataset values can be seen on Fig. 6.8, note two different features from the dataset were chosen.

In this subsection, it is shown how to get from the low accuracy (Tab. 6.2) to the high one (Tab. 6.5). By evaluating the results for different number of hidden nodes, we can see (see Fig. 6.9) that there is always some optimal point. If the number of nodes is lower, the network cannot fit the detection in such a small flow width. If the number of nodes is bigger than optimal, the network starts with overfitting, the number of input samples is too low. We found out that for our dataset with 7 gestures, the optimal number of nodes is 50.

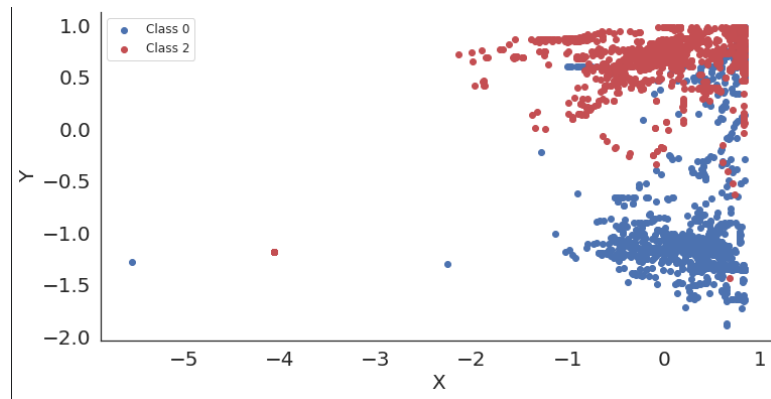


Figure 6.8: Example of dataset. Two features were chosen randomly. Note that values are not normalized.

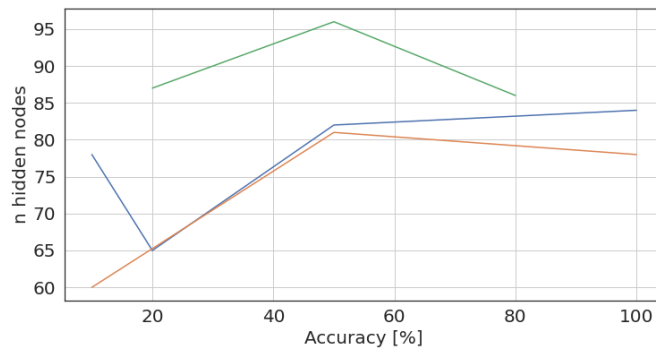


Figure 6.9: n hidden layers vs. Accuracy. As you can see in the picture, the number of layers has the optimal value for the maximum accuracy, somewhere in the middle.

Insufficient learning of the network can be seen in Tab. 6.3. The network has three layers, while the number of samples is only 287. Using data from more frames resulted in much higher accuracy, see Tab. 6.4. As it can be seen from Fig. 6.10, every gesture is learned properly only with one exception, the *Grab-Pinch* connection. That can indicate that the network has still insufficient number of samples for 3-layer network.

The most suitable option and the best result is to use the *All defined* dataset configuration (5.6), that is using the possibility of all data from the sensor. It seems that from this point, the network can finally understand the gestures properly, the total accuracy is 96% (see Fig. 6.11). Now let's compare the **Probabilistic** with **Deterministic** type of gesture detection. The total accuracy on a whole dataset for *Deterministic* option is only 84%, the *Probabilistic* approach is about 12% better with the total accuracy of 96%.

Hidden nodes	Accuracy of data		Train time	Note
	Test	Train		
10	78%	79%	<10sec.	
20	65%	66%	<10sec.	
50	82%	80%	<10sec.	
100	84%	81%	~1min.	
200	77%	85%	~5min.	

Table 6.2: Two layer model. Data picked as middle time-sample. Learning for 7 types of static gestures, 41 sample recordings of each gesture, 12 observations per recording. Input dataset $X = \text{shape}(287 \times 12)$, $Y = \text{shape}(41)$.

Hidden nodes	Accuracy of data		Train time	Note
	Test	Train		
5	57%	56%	<10sec.	
10	79%	75%	<10sec.	
20	76%	78%	<10sec.	
50	76%	78%	<1min.	
100	80%	80%	~3min.	

Table 6.3: Three layer model. Data picked as middle time-sample. Learning for 7 types of static gestures, 41 sample recordings of each gesture, 12 observations per recording. input dataset $X = \text{shape}(287 \times 12)$, $Y = \text{shape}(287)$.

6.4.2 Modelling dynamic gestures via ProMP

Probabilistic movement primitives are a great tool for incorporating the trajectory in compressed form by weights. The weight representation of Radial Basis Functions. Therefore, it is a different expression than the trajectory position. Every trajectory in the joint space can be represented as a linear combination of basis functions.

The representation of weights is suitable for learning representation with uncertainty from several demonstrations. General framework for state trajectories.

Processing data before training

For ProMP training, there is a specific need for further data processing. The reason is that the gestures can, unlike a robot trajectory, be performed at another place and should still be recognized by the system.

Normalization to zero can be done with Eq. 6.37. $\mathbf{p} = (p)_{n=1}^P$ variables are the old point values and \mathbf{p}^* is the array of new values. Values P is number of points and p_{ref} is picked automatically the reference point to which the path will be normalized to, often it is the first point of path \mathbf{p} or the middle point.

$$\mathbf{p}^* = (p - p_{ref})_{n=1}^P. \quad (6.37)$$

Second normalization is the scaling one, which sets the position values

Hidden nodes	Accuracy of data		Train time	Note
	Test	Train		
10	60%	60%	~2min.	Fig. 6.10
50	81%	82%	~10min.	
100	78%	77%	~40min.	
50	89.7%	90.2%	~10min.	

Table 6.4: Two layer model. Data picked as every 10th time-sample. Learning for 7 types of static gestures, 1230 sample recordings of each gesture, 12 observations per recording. input dataset $X = \text{shape}(8610 \times 12)$, $Y = \text{shape}(8610)$.

Predicted	Confusion matrix							sum_col
	Grab	Pinch	Point	Respectful	Spock	Rock	Victory	
Grab	347 13.43%	280 10.84%	4 0.15%	28 1.08%	18 0.70%	30 1.16%	36 1.39%	743 46.70% 53.30%
Pinch	1 0.04%	77 2.98%	0 0.0%	0 0.0%	2 0.08%	0 0.0%	0 0.0%	80 96.25% 3.75%
Point	0 0.0%	2 0.08%	356 13.78%	1 0.04%	2 0.08%	1 0.04%	0 0.0%	362 98.34% 1.66%
Respectful	0 0.0%	0 0.0%	0 0.0%	327 12.66%	0 0.0%	0 0.0%	8 0.31%	335 97.61% 2.39%
Spock	30 1.16%	11 0.43%	2 0.08%	6 0.23%	332 12.85%	1 0.04%	3 0.12%	385 86.23% 13.77%
Rock	0 0.0%	0 0.0%	0 0.0%	0 0.0%	2 0.08%	323 12.50%	0 0.0%	325 99.38% 0.62%
Victory	0 0.0%	0 0.0%	9 0.35%	5 0.19%	0 0.0%	0 0.0%	339 13.12%	353 96.03% 3.97%
sum_col	378 91.80% 8.20%	370 20.81% 79.19%	371 95.96% 4.04%	367 89.10% 10.90%	356 93.26% 6.74%	355 90.99% 9.01%	386 87.82% 12.18%	2583 81.34% 18.66%
	Grab	Pinch	Point	Respectful	Spock	Rock	Victory	sum_lim

Figure 6.10: Static gesture recognition via Probabilistic approach. Confusion matrix shows the results for each of the 7 compared static gestures from main dataset (5.6), *User defined* type configuration (5.6). Network with 2 layers and 50 hidden nodes was used. As it can be seen the network is learned properly (over 90% predicted accuracy, column right) except the *Grab* gesture (only 46% predicted accuracy, top cell, right column), especially when the actual gesture is *Pinch* (only 20% actual accuracy, when predicted gesture is *Grab*). Experiment Tab. 6.4.

Hidden nodes	Accuracy of data		Train time	Note
	Test	Train		
20	87%	87.2%	~12min.	50000 iter., Fig. 6.11
50	96%	96%	~40min.	
80	86%	88%	~45min.	

Table 6.5: Two layer model. Data picked as every 10th time-sample. Learning for 7 types of static gestures, 1230 sample recordings of each gesture, 87 observations per recording. input dataset $X = shape(8610 \times 87)$, $Y = shape(8610)$.

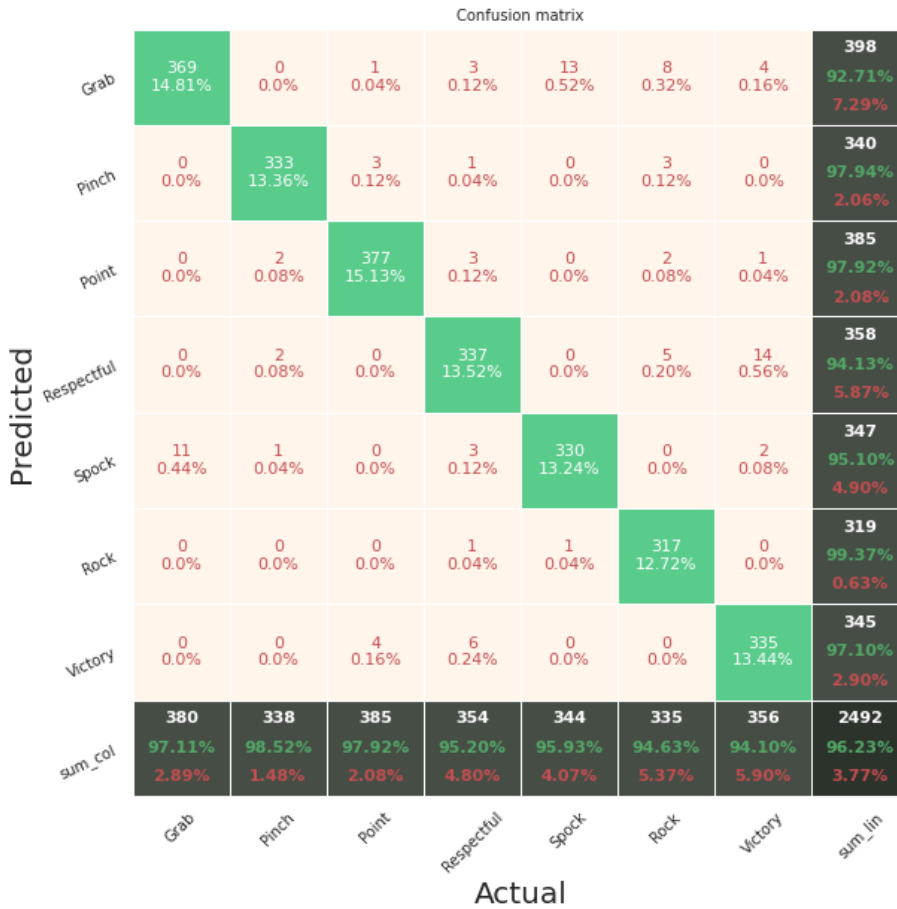


Figure 6.11: Static gesture recognition via Probabilistic approach. Confusion matrix shows the results for each of the 7 compared static gestures from main dataset (5.6), *All defined* type configuration (5.6). Network with 2 layers and 50 hidden nodes was used. As it can be seen from total accuracy over 96%. The network is learned well, the only difficulties obtained from *Grab/Spock* combinations, however stil over ~ 92%. Experiment Tab. 6.5.

between 0 and 1. This can be done with Eq. 6.38, where the *max* function returns the maximum number of array and *min* function returns the minimum

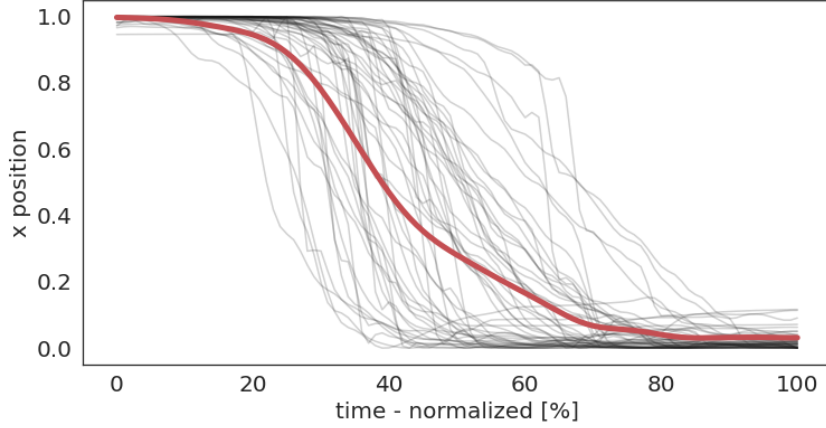


Figure 6.12: Dataset samples of gesture *Swipe left* with the ones trained with *ProMP* sample. Visualized the x axis.

number of an array. Then for whole array it is $\mathbf{p}^* = f(\mathbf{p})$.

$$f(p) : p^* = \frac{p - \min(p)}{\max(p) - \min(p)}. \quad (6.38)$$

The input for *ProMP* training are also the velocities. They can be computed with derivation of a trajectories with respect to a time series for example with `diff` function from *Numpy* package [44].

■ Training and sampling trajectories

Input data for ProMP is a set of trajectories in three dimensions, in particular *Palm points* P_p and *Palm velocities* P_v , i.e., several demonstrations of a gesture.

The network is trained. Properties of network includes Exponential basis function. Processes of training are described in Ch. 3.5.

The outputs of the trained network from the input trajectories representation of one trajectory are the movement weights \mathbf{w} and from them are computed mean values vector μ_W and variances matrix Σ_W .

Trained paths with all gestures of type *Swipe left* can be visualized as in Fig. 6.12.

■ Gesture Detection via Euclidean distance

The method for dynamic gesture detection in this section is using Euclidean distance comparison of two trajectories. The comparing process can be visualized as in Fig. 3.3. As it can be seen, the comparison points are picked consistently by the time series.

The main gesture dataset is used (see Ch. 5.6) for detection of 4 types of *Swipe* gestures (*Swipe Left*, *Swipe Right*, *Swipe Up*, *Swipe Down*). Around 50 sample recordings are included for every of these gestures.

Firstly, the *ProMP* network is trained on that dataset and generates one sample trajectory for every type of gesture. Therefore, we have four paths in every of the three dimensions. These paths are used for comparing the trajectories with the given testing samples. Secondly, intentionally in the first two methods, the *ProMP* samples are not presented. The difference between applying the *ProMP* was observed.

Methods:

- Method without *ProMP*. Compare the new trajectory to every sample in the dataset and then compute the mean value of all gestures. Computer demanding, not suitable.
- Method without *ProMP*. Compare a new trajectory to a random sample from every dataset
- Method using *ProMP*: Dataset is separated to trajectories, for each chosen set of trajectories the *ProMP* learning is applied. For each gesture, a sample trajectory is created. Then these trajectories are compared to the test trajectory

■ Experimental results and Discussion

The results were marked into Tab. 6.7. The method No. 1 is presented as Euclidean distance calculation between two trajectories, in a configuration of each sample with each sample. On Fig. 6.15 can be seen that the total accuracy is 94%, which is impressive, on the other hand, the computational time is about 25 minutes. Which would result in 48 seconds to single sample. This method cannot be used in live detection. Second Method is weaker in terms of total accuracy, to only $\sim 70\%$, the compared trajectory is chosen randomly, this method is not the right approach, and it is here only for comparison. The third method did finally involve the *ProMP* trained trajectory paths. The third method is the most useful when evaluating accuracy and performing time (Total accuracy 89%), while 50 times lower execution time than method No. 1.

Method	Accuracy	Time	Note
No.1	94%	~ 25 min.	Fig. 6.15
No.2	$\sim 70\%$	~ 15 sec.	
No.3	89%	~ 15 sec.	Fig. 6.14

Table 6.6: Dynamic Time Warping method. Experiments performed with swipe directional gestures. Three methods presented.

In this section, different sample trajectories based on *ProMP* were created. Three methods with Euclidean distance evaluation were performed. was performed, see Fig. 6.14.

Confusion matrix

Predicted	Swipe_Left	48 23.76%	0 0.0%	2 0.99%	0 0.0%	50 96.00% 4.00%
	Swipe_Right	0 0.0%	54 26.73%	0 0.0%	7 3.47%	61 88.52% 11.48%
	Swipe_Down	3 1.49%	0 0.0%	57 28.22%	0 0.0%	60 95.00% 5.00%
	Swipe_Up	0 0.0%	0 0.0%	0 0.0%	31 15.35%	31 100% 0.00%
	sum_col	51 94.12% 5.88%	54 100% 0.00%	59 96.61% 3.39%	38 81.58% 18.42%	202 94.06% 5.94%
		Swipe_Left	Swipe_Right	Swipe_Down	Swipe_Up	sum_lin
		Actual				

Figure 6.13: Dynamic gesture recognition via Euclidean distance. Confusion matrix shows the results for method No. 1, each with each, swipe directional gestures are compared from main dataset (5.6), *Palm point* P_p path configuration

6.4.3 Dynamic Gesture recognition via Dynamic Time Warping

Time warping as described in Ch. 3.5 was used to help with dynamic gestures, respectively, to help with path evaluation. The function that was used is from *fastdtw* package [45]. The task is to classify the category of an observed trajectory. We propose and compare three approaches using dynamic time warping to match an observation to a set of labeled trajectories. The methods vary in computational complexity and accuracy.

The task is to determine to which category is classified some selected (or newly recorded) sample, knowing that the time warping function returns the Euclidean distance between two paths as a scalar quantity. Input trajectory is extracted as the trajectory of palm *Poses* P_{pose} (see Tab. 5.2). Length of trajectory is $S = 100$ samples as 1 second recording, therefore input is $\mathbf{P}_{pose} = (P_{pose})_{s=1}^S$. Methods:

1. Compare new trajectory to every sample in dataset and then compute mean value of all gestures. Computer demanding, not suitable.
2. Compare a new trajectory to random sample from every dataset
3. Create 'mean' trajectories from *ProMP* sampler and then compare the

Confusion matrix

Predicted	Swipe_Left	41 20.30%	0 0.0%	9 4.46%	0 0.0%	50 82.00% 18.00%
	Swipe_Right	0 0.0%	54 26.73%	0 0.0%	3 1.49%	57 94.74% 5.26%
	Swipe_Down	9 4.46%	0 0.0%	50 24.75%	0 0.0%	59 84.75% 15.25%
	Swipe_Up	1 0.50%	0 0.0%	0 0.0%	35 17.33%	36 97.22% 2.78%
	sum_col	51 80.39% 19.61%	54 100% 0.00%	59 84.75% 15.25%	38 92.11% 7.89%	202 89.11% 10.89%
		Swipe_Left	Swipe_Right	Swipe_Down	Swipe_Up	sum_lin
		Actual				

Figure 6.14: Dynamic gesture recognition via Euclidean distance. Confusion matrix shows the results for method No. 3, *ProMP* sampled paths are used, for 4 directional *Swipe* gestures.

new trajectory to generated one

■ Input data processing

The input for this approach execution is *Processed frame* \mathcal{P} with two types configuration (see 5.6). The *User defined* and *All defined*.

The reason for data processing is that the recorded data are not consistent, frames per second are erratic, and there is even no guarantee that it will stay over 50 fps.

Firstly, there is a need to pick exactly one-second long intervals. As it has been said, we have an array of *Processed frames* \mathcal{P} . The program analyzes the *Timestamp* of every frame and checks if it not older than the last *Timestamp*. The buffer length is always bigger than one second. Note that this is needed to be done for every sample recording.

Secondly, all samples have different time lengths and there is a need for every sample to have the same length. One option to do so is to create the remaining samples with interpolation. For every trajectory in the dataset is performed interpolation. This can be done with `interp1d` Python function from *SciPy* package [46]. The series can and is interpolated with a cubic function. The length of time can be changed in settings, but it is set to 100

samples.

■ Experimental results and Discussion

Method	Accuracy	Time	Note
No.1	96%	~10min.	Fig. 6.15
No.2	~75%	~0.5sec.	
No.3	92%	~0.5sec.	Fig. 6.16

Table 6.7: Dynamic Time Warping method. Experiments performed with swipe directional gestures. Three methods presented.

The method is Dynamic Time Warping is not computationally demanding at all. While evaluating *each with each* (Method No. 1 by Euclidean Distance, Tab. 6.7) took about 20 minutes, the evaluating *each with each* with DTW, it took only 8 sec. of computation. The results can be seen from results in Fig. 6.15, the accuracy across all recorded samples is the best with a total value of 96%. It can be said that this model is the most useful of all the presented dynamic gesture detection models and it is advised to use it for this kind of detection.

Let us talk about the second method. Test trajectory is compared with the random trajectory from the dataset. From the given results, we can see the decrease of accuracy from 96% (Method No.1) to only ~75% (Method No.2), the method is arbitrary, but the percentage number can be used later in comparison. This method is quick to compute, but it is not sufficient to be useful.

Method number three uses *ProMP* network. The method divides the dataset by gesture and computes the *ProMP* training on all divided sub-datasets. By training the network, the same number of sampled trajectories is created. The computation time is half a second for all samples in the dataset, which is 0.025s for a single test sample. The accuracy is ~ 92% as seen in Fig. 6.16. This could be the best solution for live detection of gestures, because the method brings the highest accuracy for the lowest time.

Confusion matrix

Predicted	Swipe_Left	47 23.27%	0 0.0%	0 0.0%	0 0.0%	47 100% 0.00%
	Swipe_Right	0 0.0%	54 26.73%	0 0.0%	4 1.98%	58 93.10% 6.90%
	Swipe_Down	4 1.98%	0 0.0%	59 29.21%	0 0.0%	63 93.65% 6.35%
	Swipe_Up	0 0.0%	0 0.0%	0 0.0%	34 16.83%	34 100% 0.00%
	sum_col	51 92.16% 7.84%	54 100% 0.00%	59 100% 0.00%	38 89.47% 10.53%	202 96.04% 3.96%
		Swipe_Left	Swipe_Right	Swipe_Down	Swipe_Up	sum_lin
		Actual				

Figure 6.15: Dynamic gesture recognition via Dynamic Time Warping. Confusion matrix shows the results for Method No.1, each with each, swipe directional gestures are compared from main dataset (5.6), *Palm point* P_p path configuration. As it can be seen, total accuracy is around 96% is rated as useful model. Experiment Tab. 6.7.

Confusion matrix

Predicted	Swipe_Left	39 19.31%	0 0.0%	1 0.50%	0 0.0%	40 97.50% 2.50%
	Swipe_Right	0 0.0%	54 26.73%	0 0.0%	3 1.49%	57 94.74% 5.26%
	Swipe_Down	12 5.94%	0 0.0%	58 28.71%	0 0.0%	70 82.86% 17.14%
	Swipe_Up	0 0.0%	0 0.0%	0 0.0%	35 17.33%	35 100% 0.00%
	sum_col	51 76.47% 23.53%	54 100% 0.00%	59 98.31% 1.69%	38 92.11% 7.89%	202 92.08% 7.92%
		Swipe_Left	Swipe_Right	Swipe_Down	Swipe_Up	sum_lin
		Actual				

Figure 6.16: Dynamic gesture recognition via Dynamic Time Warping. Confusion matrix shows the results for Method No.3, *ProMP* generated base trajectory as reference, swipe directional gestures are compared from main dataset (5.6), *Palm point* P_p path configuration. As it can be seen from total accuracy around 92%. Experiment Tab. 6.7.

Chapter 7

Robot control via gestures

In this chapter are presented different methods of coupling the hand tracking to robot end-effector to accomplish the defined policies. The defined ones are picking and placing a button, opening and closing a drawer, and pushing a button. These are the most basic tasks a robot manipulator can do and they can be extended in further work.

Four modes of interaction are presented. *Live* mode maps the hand to end-effector in real time. *Interactive* does the same as *Live* mode but adds the Scene object interaction. *Gesture based* mode is moving the end-effector incrementally based on gesture detection ticks. Finally, the memory path execution with hand control is executing any robot path that is saved in memory as directed by the gestures. This is useful when motions have to be executed with high positional precision, but the process shall be controlled via gestures.

Possible output interpretations

What type of control does a person need or what type of interpretation can be output from gestures.

- Boolean value, trigger something
- Use distance as a value, set some properties with given value
- Using pose (e.g., palm position) for x,y coordinates
- Directional cross, move in the direction (left, right, up, down) one step at a time when the gesture is triggered
- Wait/Hold (For confirmation), remain in given pose

Then for a more robust operation trigger, there can be used a combination of these types. Every action and their reaction is tied to some context.

7.1 Controlling the end-effector via gestures

In this section is a description of mapping from *Leap Motion Controller* to *Base link* motion planning workspace and to *User Interface*. All workspaces

are tied with Kuka *iiwa* described in Ch. 4.1.2. The transformations can be visualized by diagram in Fig. 7.1. The workspaces are described in Ch. 5.5.

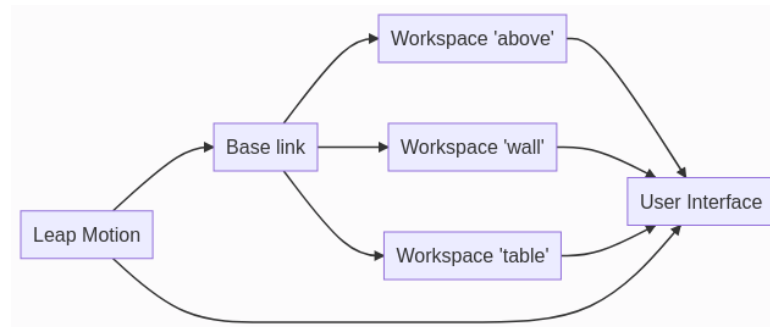


Figure 7.1: Transformations between workspaces.

7.1.1 Correspondence mapping from perceived hand pose to robot workspace

Every workspace represented in Fig. 7.1 are completely different environments, and they have different defined axis ordering, different scaling, and translations. In order to achieve harmony between workspaces, different transformations are presented.

The person can place Leap Motion sensor virtually in the robot workspace and scale the observations to let the operator comfortably control the robot in its workspace. The scaling might be adjusted for tasks requiring high precision or large motion.

Fixed orientation mode

Fixed orientation mode is defined with orientation specifications:

- Fixed: End-effector orientations are assigned as the default values defined for each workspace (see Eq. 7.5)
- Not Fixed: End-effector takes real orientation from the palm position

Leap Motion to *Base link* transformation

First transformation is needed to be done, because the *Leap Motion* device has different axes and scaling. Following equation (Eq. 7.1) take point from Leap workspace and transforms it into the *Base link* workspace. As it can be seen first two axes are switched because Leap Motion is intended to have axes x, y as the computer screen (see Fig. 7.2c), on the other hand *Base link* workspace has x, y as a ground axis. Scaling parameter S is in this case

$S = 0.001$, because millimeters from Leap Motion space are converted to meters in *Base link* workspace.

$$\mathbf{P}_{baselink} = S \cdot \begin{bmatrix} p_x \\ -p_z \\ p_y \end{bmatrix}_{leap} \quad (7.1)$$

■ Base link to Workspace transformations

Scenes are composed of three workspaces, which are defined in Ch. 5.5. These workspaces are defined with different end-effector translations and orientations. They are named as the *above* workspace (see Fig. 7.2a) for working in space above the robot, the *wall* workspace (see Fig.7.2b) so the robot can write to the wall and the *table* workspace so the robot can write on the table.

Transformations from *Base link* to these workspaces can be written as Eq. 7.2, Eq. 7.3 and Eq. 7.4.

$$\mathbf{P}_{above} = S \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}_{Baselink} + \begin{bmatrix} 0.0 \\ 0.0 \\ 0.85 \end{bmatrix}_{start}, \quad (7.2)$$

$$\mathbf{P}_{wall} = S \cdot \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}_{Baselink} + \begin{bmatrix} 0.7 \\ 0.0 \\ 0.1 \end{bmatrix}_{start}, \quad (7.3)$$

$$\mathbf{P}_{table} = S \cdot \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}_{Baselink} + \begin{bmatrix} 0.4 \\ 0.0 \\ 0.0 \end{bmatrix}_{start}, \quad (7.4)$$

where S is the scalar scaling factor of the workspace. In practise, $S = 1$.

End-effector default orientations for each workspace can be defined. This definition is strictly for the fixed end-effector orientation mode. Upwards for *above* workspace, direction to x-axis for *wall* option and direction downwards for *table* option. Orientations can be written with quaternions in Eq. 7.5.

$$Q_{above} = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}, Q_{wall} = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}, Q_{table} = \begin{bmatrix} 0.0 \\ 0.0 \\ \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix} \quad (7.5)$$

End-effector live orientation transformation, which is when end-effector fixed position mode is off. This transformation is needed, because different workspaces have different axis ordering. Performing the axis switch is done by a sequence of Eq. 7.6 and Eq. 7.7.

$$M = rot(e_1, axis_1) \cdot rot(e_2, axis_2) \cdot rot(e_3, axis_3), \quad (7.6)$$

$$e^* = euler_from_matrix(M), \quad (7.7)$$

where $\text{axis} = \begin{bmatrix} \text{axis}_1 \\ \text{axis}_2 \\ \text{axis}_3 \end{bmatrix}$ rotational matrix, $e = [e_1, e_2, e_3]$ are input Euler

angles and e^* is the output Euler angles. `rot` function returns the rotation matrix of the angle in the given axis and `euler_from_matrix` returns Euler angles from the rotation matrix. Note that `euler_from_matrix`, and `rot` functions are used from *ROS* transformation package [39].

Leap Motion to User Interface transformation

Leap Motion palm position is transformed to *UI* screen x, y coordinates (2D projection) using Eq. 7.8, received values $\mathbf{P}_{ui,x'}$ and $\mathbf{P}_{ui,y'}$ represents pixel values on the screen and $\mathbf{P}_{ui,z'}$ is displayed as volume of hand marker, typically between volume 5 – 100.

$$\mathbf{P}_{ui} = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 2 \cdot x \\ -2 \cdot y \\ -(z - 200)/10 \end{bmatrix} + \begin{bmatrix} w/2 \\ h \\ 0 \end{bmatrix} \quad (7.8)$$

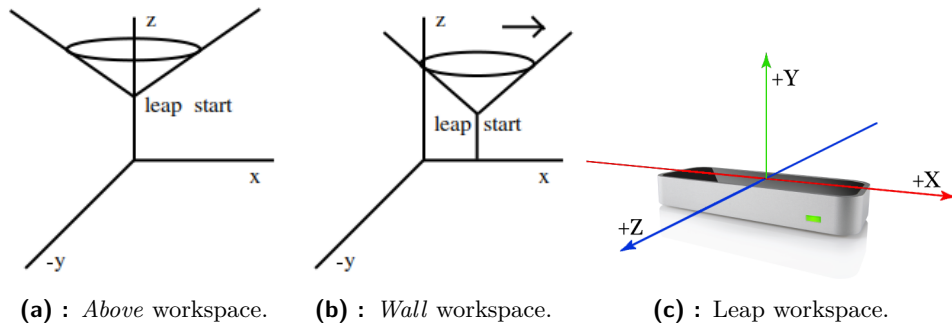


Figure 7.2: *Above* and *Wall* workspaces visualized in coordinate frame in respect to *Base link*.

7.1.2 Live mapping

Live mapping process that invokes the *Manipulation* type of gesture (3.2). The main goal is to track the robot manipulator end-effector with *Leap Motion* as with 3D mouse. In this mode, the chosen workspace must be set up properly, otherwise, the end-effector will not reach the right position. Specific orientation even shrinks the reachable space, then the fixed mode can be used, the manipulation is more comfortable.

First, we choose what position from Leap Motion will be mapped. In all cases, it is best to use the center of the user's palm, but it is possible to map any position of the hand's *Bone structure* \mathbf{S} (3.3). The tip of the pointing finger is also worth mentioning as the second best candidate for mapping. Transformations from *Leap to Workspace* are used to convert poses to the

robot workspace. This can be done with a custom frequency, in realization it has been set to 10Hz. Users choose from defined workspaces to operate in. The process involves writing the transformed pose of end-effector to memory where it can be picked by the inverse kinematics task which then returns joint values and then these joint values are used for robot control.

The transformations can have different properties. For example, the scaling factor of the workspace can be changed according to the situation. As well as the starting position of workspace and more. The changes are then projected as an updated *Interaction Box* (Ch. 5.5.1) and its corners which are visualized by *RVIZ* markers, see Fig. 7.3.

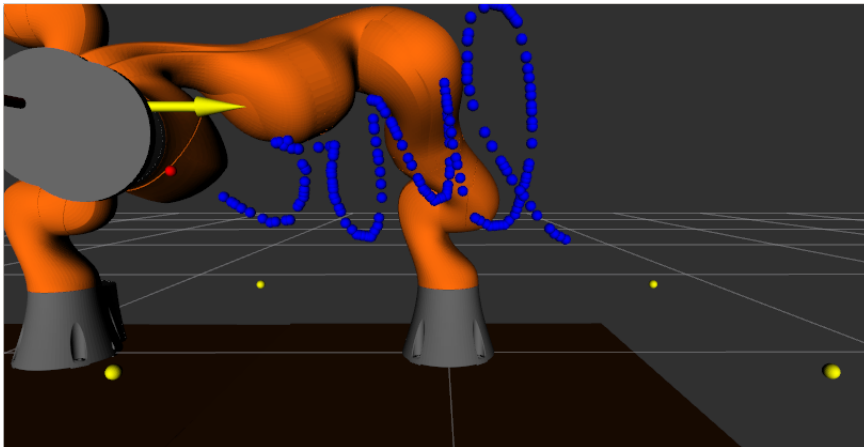


Figure 7.3: Live mapping example with drawing a trajectory.

7.1.3 Interactive mapping

Interactive mapping was created to perform defined scenarios and policies. The idea was to develop a system that could easily interact with the environment. Use Leap Motion 3D mouse to bring the end-effector flawlessly to the closeness of the object, but not interact with it unless the user does the specific action.

The main key is that scenarios, objects, and actions can be variously created according to specific needs. Interactive mapping mode is defined as a live mapping option but with objects in mind. This mode interacts with objects and should have provided collision detection. Every object defined in Scene has its *grasping position* defined from its origin. If end-effector comes closer to this point, the grasping action can be executed. This action is different based on the type of object.

Collision detection checks if end-effector is inside any object. If it is, it will not propagate the end-effector further and find the nearest pose, that is, without collision. That can be done by stopping the live mapping, if the end-effector is near the object and then computing the closest distance to the user desired point, while still maintaining collision-free status. The grid search method can be used.

7.1.4 Memory path execution with hand control

The process involves a user choosing an arbitrary path from memory by choosing from existing paths or from files. Selected time trajectory will pop up on top of the user interface, see Fig. 7.4.

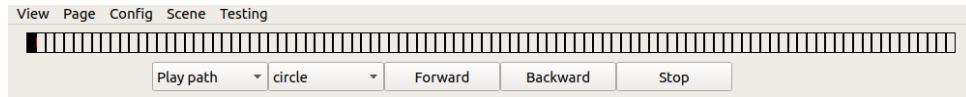


Figure 7.4: Path play trajectory bar.

The chosen path can be played, stopped, or rewinded by a button. While the path is executing, the black mark signals its position based on the path request and the red mark signals the progress of the received end-effector position.

And it would not be gesture controlled if the path execution would not be able to control with gestures. A user can *grab* the black path index marker with the left hand and move it left or right on the timeline. The robot manipulator then goes to the corresponding position of the path.

7.1.5 Gesture based robot control use-cases

Three robotic manipulation tasks were selected to demonstrate the capabilities of the proposed system. The three use cases are box picking, drawer opening, and button pushing. Attaching and detaching of objects is done via MoveIt! script (see Ch. 4.1.2).

Pick/place method

In this method, the user is told to move the box to the goal region. The user can decide where he is putting the box, also can decide when he wants to attach or detach the box. The most comfortable and easy is use the *Grab* gesture for attaching the box and detaching the box when the *Grab* gesture is released. Another option can be grabbing with three fingers or with two fingers, which corresponds to the *Pinch* gesture. When picking new gestures for this method, the picked gesture must be intuitive, for example, the *Point* gesture is easier to perform, but it unsuitable for this method. This gesture operates with the *table* workspace to reach the surface with end-effector. For better moving accuracy, the scaling factor can be lowered. Brave goal would be to increase the level of accuracy to the point that the user could fill the toothbrush with the toothpaste, but that would require testing on a real robot. This method is visualized on Fig. 7.5.

Additional informations about the method of realization: The grasping tool is in this example represented by suction cups mounted on end-effector joints. If the end-effector comes close to any object, it is able to attach it and detach it on demand. Action zone is anytime when the end-effector finds itself near to the box.

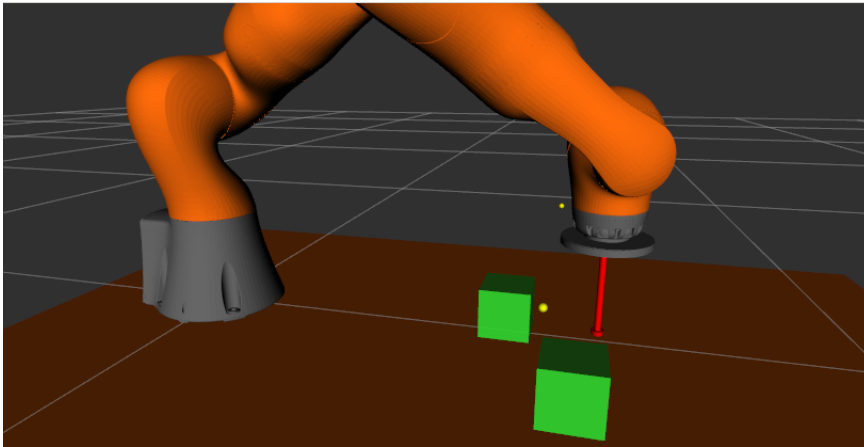


Figure 7.5: Picking and placing of two boxes. Two boxes scene.

■ Drawer manipulation method

The drawer is made up by its immovable shell and three movable sockets. The user has the ability to interact with any of them how he wishes. Again, some gesture is used for attaching the particular socket. The most reliable gesture is again the *Grab* gesture, it interpreters the real life most of all gestures. The interaction point with the drawer socket is in the middle of each drawer. This method is visualized on Fig. 7.6.

When the socket is attached, the program is turned to *Strict mode* where only movement in one axis is allowed. The Strict mode is turned on until the user detaches the drawer. This simplification is useful, because the drawer is restricted by all means to move only in one dimension at all times, the drawer movements are defined and do not offer moves in other dimensions. Most users do not have the best accuracy in hand control and it would result in hinting the drawer shell.

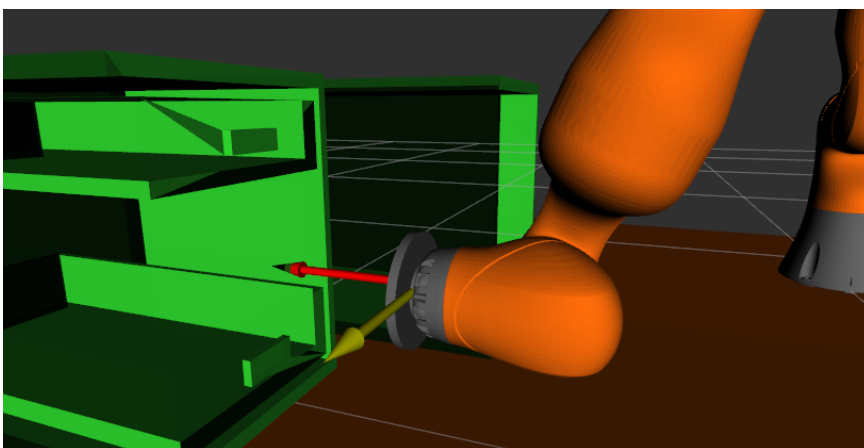


Figure 7.6: Opening and closing drawer sockets. Two drawers scene. Note that *RVIZ* is not visualizing the front sides of the sockets (for better view), but they are still there.

■ Pushing button method

The user is able to move around in live mode, when he comes near the button, the action will pop up. Interaction point is on the top of the button. The button is clicked with the assigned gesture. Every time the button is clicked, the movement of the clicking button is performed. The clearest candidate for clicking is *Point* gesture. Another potential gesture could be the *Touch* gesture towards the button. Performed as touching the screen with the index finger. The drawback is that when the user is performing the *Touch* gesture, the center of the palm is moved and the the *Touch* gesture is recognized, it deflects from the original position, and the new position does not have to be in the action zone. The *Pin* gesture is more suitable to use (*Pin* movement performed as playing the piano key with the index finger). The center of the palm is not changed when performing this gesture. This method is visualized in Fig. 7.7.

Additional informations about the scene: The button has its immovable housing part which is not moving and its inner part which does the vertical movement (click) during activation.

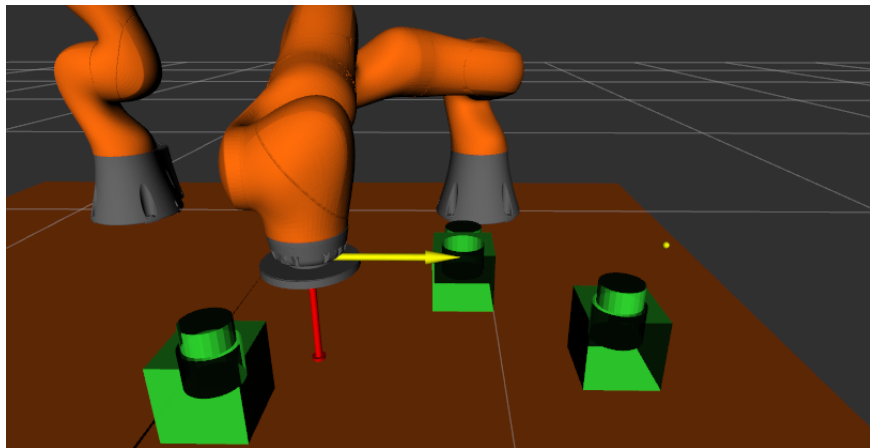


Figure 7.7: Clicking a buttons. Three buttons scene.

■ List of capabilities

Note that the gestures picked in the previous S. 7.1.5 can be changed by the situation to reach the highest accuracy and user comfort. The *Grab*, *Pinch* and *Point* static gestures were the most natural way to control the robot actions. The dynamic gestures were also tested for action launching, but they were worse for comfort when performing them. For example the detaching an object can be performed by *Rotate* gesture, but releasing the static one is much easier. Sometimes the most efficient and useful realizations are also the most simplest ones.

7.1.6 Iterative gesture control

The last mode of end-effector manipulation in this chapter is using the iterative control. This method is presented as the contrast to live method control. Possible representation of the control realization is made using the *Swipe* gesture. Condition to every *Swipe* gesture is to condition the gesture by orientation. There is defined 6 orientation directions (Left, Right, Up, Down, Front, Back). Every gesture swipe detected in a given dimension is performed as a move of end-effector towards the given position. The movement iteration has defined distance of movement. It is set to 10 cm by default.

7.2 Testing accuracy of inverse kinematics controller

RelaxedIK formulates the *path-wise inverse kinematics problem* as multi-objective optimization problem [19], where motion goals in addition to end-effector pose matching can be encoded as terms in the weighted sum. The solver can decide to avoid threats (self-collisions, singularities, etc.) at the expense of accuracy.

First, we tested the accuracy of the inverse kinematics controller from package *Relaxed IK*. The test is very simple. First, we assign the goal pose of end-effector and then see to what point it will converge.

Test on a single plane when axis $z = 0.1m$ and varying x and y coordinates on one side of a robot. The results are in Fig. 7.8. There can be seen that with a proper configuration the error of inverse kinematics is under 1 cm. On the left and right sides of the plot, the robot starts to get out of range and that is the reason for the higher errors.

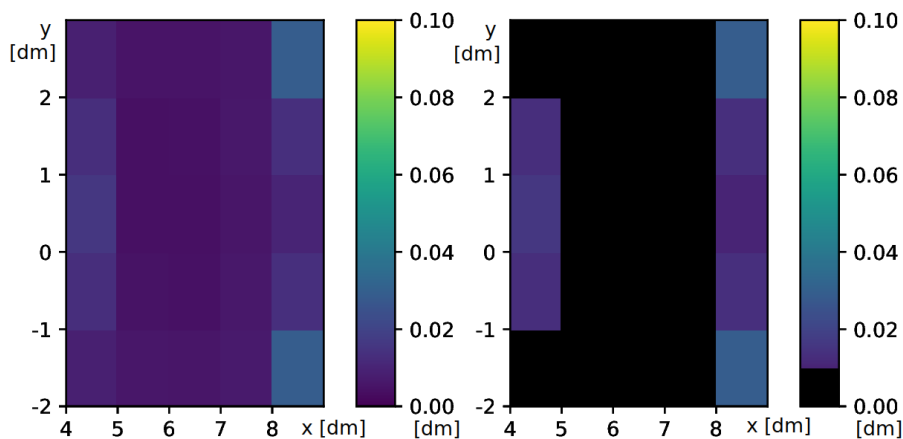


Figure 7.8: Plane test accuracy of Relaxed IK controller. x and y are axis of the robot grid. Values of axis as well as scale units are in decimeters.

7.3 User study

User study was done on 7 people. There were three categories. The deterministic gesture recognition, the probabilistic one, and manipulation methods. For gesture recognition, two confusion matrices for each category were created. They includes the sum of all participants. For the manipulation methods, three tables determining the time and success were created. Manipulation method action gestures were detected with deterministic type of detection.

Deterministic gesture results

Users were asked to perform 7 gestures from list. Every gesture recordings were repeated from 3 to 5 times. The results can be seen on Tab. 7.1. From the table, it can be seen that the first three gestures were 100% successful in all tries and therefore can be adopted in future applications. The worst gesture was *Spock*, because it is hard to perform this gesture on the first try properly.

Person no.	Accuracy	Best Gesture	Worst Gesture
1	77.2%	Grab, Pinch, Point, Respectful, Rock	Spock (0/4)
2	80.6%	Grab, Pinch, Point, Rock	Victory (0/4)
3	64.3%	Grab, Pinch, Point, Respectful	Spock (0/4)
4	70.0%	Grab, Pinch, Point, Respectful, Rock	Spock (0/5)
5	82.7%	Grab, Pinch, Point, Respectful	Spock (0/4)
6	64.3%	Grab, Pinch, Point, Rock	Spock (0/4)
7	100.0%	All	-
mean	~ 77%		

Table 7.1: User study evaluation of Deterministic approach. 7 static gestures were tested. Best Gesture states that all tries were successful. Worst gesture states that no tries were successful.

Probabilistic gesture results

The different method was applied to the same dataset. From the results on Fig. 7.2, it can be seen that the method learned better the *Spock* gesture than the *Deterministic* version, however the *Rock* and *Victory* gestures were a problem for it. The reason can be found in the recording of the dataset. The probabilistic network was not learned from the slight variances in the gesture execution, which the users were performing. If the network would be learned by the dataset from more people, the accuracy would be higher.

Manipulation method results

The results were divided into three tables (Tab. 7.3, Tab. 7.4 and 7.5) for each manipulation task. Gesture detected column represents which gestures

Person no.	Accuracy	Best Gesture	Worst Gesture
1	79.7%	Grab, Pinch, Point, Spock	Rock (1/3)
2	87.4%	Grab, Pinch, Point, Spock	Rock (0/4)
3	86.6%	All except Victory	Victory (0/4)
4	72.4%	Grab, Pinch, Point, Spock	Victory (2/4)
5	66.0%	Grab, Point, Spock	Rock (1/4)
6	84.5%	Grab, Pinch, Point, Rock	Rock (2/4)
7	78.3%	Grab, Pinch, Point, Spock	Victory (0/4)
mean	~ 79%		

Table 7.2: User study evaluation of Probabilistic approach. 7 static gestures were tested. Best Gesture states that all tries were successful. Worst gesture states that no tries were successful.

were detected throughout the process, where the red one are wrongly detected gestures. Time denotes the duration from the start to completion of the manipulation task and *Success* column indicates if the process were successful on the first try. In the tables, there can be seen a substantial variance in time completion, which were also influenced by the fact that they have seen the manipulation for the first time and most of them were told only the minimum informations on how to operate the manipulator. The second fact is that some users did not want to use the manipulation as quickly as possible, but they were executing the task at a slow pace, which they liked. All attempts succeed on the first try, even though every user has seen the task on the first try.

Person no.	Gestures detected	Time	Success
1	Grab	28 s	True
2	Grab	21 s	True
3	Grab, Pinch	40 s	True
4	Grab	23 s	True
5	Grab	16 s	True
6	Grab	12 s	True
7	Grab, Pinch	24 s	True

Table 7.3: User study, manipulation method. Users were asked to open and close one socket of the drawer with *Grab* gesture. Average time of completion is 23 s, all attempts were successful on first try.

Person no.	Gestures detected	Time	Success
1	Grab	16 s	True
2	Grab	24 s	True
3	Grab, Point, Pinch	40 s	True
4	Grab, Pinch	36 s	True
5	Grab	8 s	True
6	Grab	8 s	True
7	Grab, Pinch	20 s	True

Table 7.4: User study, manipulation method. Users were asked to pickup a box and put it into goal region with *Grab* gesture. Average time of completion is 21 s, all attempts were successful on first try.

Person no.	Gestures detected	Time	Success
1	Point	9 s	True
2	Point	4 s	True
3	Point	5 s	True
4	Point	8 s	True
5	Point	2.5 s	True
6	Point	3 s	True
7	Point, Grab	12 s	True

Table 7.5: User study, manipulation method. Users were asked to push a button with *Point* gesture, one-time click. Average time of completion is ~ 6.2 s, all attempts were successful on first try.

Chapter 8

Conclusion and Future work

The summarization of the whole thesis can be done in three points. Firstly, the gesture detections were defined and developed in various ways. The results are presented below. Secondly, the manipulation methods were developed and it was tested their accuracy with user study. Thirdly, the User Interface were developed.

Let us compare the static gesture representation results, 7 static gestures were compared with values from the main dataset (see S. 5.6). *Deterministic* approach with total accuracy of 84% (see Fig. 6.6) with the *Probabilistic* approach with 96% (see Fig. 6.5). The *Probabilistic* method is superior to the *Deterministic* one, which confirms the hypothesis of the thesis, that the *Deterministic* defined gesture detection will never be as accurate as the *Probabilistic* one, learned with the proper way. The best *Probabilistic* method has only 2-layers and 50 inner nodes.

Evaluating the *Dynamic gestures*, which method is performing best, both the accuracy and computational cost were taken into account. The highest accuracy was reached by the Dynamic Time Warping method, (Ch. 6.7), where each-to-each type of comparison was used. For this method, the achieved accuracy was up to 96%. The best method for compromise is the *ProMP* trajectory sampled method. The accuracy is only a little lower than with the first method, resp. 92%, but the time is reduced by more than fifty times. This option is the same as for the lowest time computed.

The main thing which was not depicted in discussions is the dependence of all detection tests on the proper tracking of Leap Motion bone detection software. When the user study was performed, it happened that the subject did not know that the sensor is inactive and performed a given gesture with a completely wrongly detected bone position, resulting in a worse accuracy score.

The user study enabled the evaluation of static gesture detection and ease of use of the proposed system. The deterministic method did not do well on the user study. The overall accuracy was 79% (see Fig. 7.2), which is worse than the result of the initial tests, and this might be caused by the tuning the parameter for a single user. Despite the bad results, the *Probabilistic* type still wins with 79% (see Fig. 7.2), even if only with a small difference.

The robot manipulator can be extended into two arms. Each arm corre-

sponds to the left and right hand. The *Relaxed IK* tool does also support a multiarm setup, so the realization would not be difficult. Two-hand gestures would enable a variety of new interaction methods, which might be worth exploring. More complex interactions would be possible to operate. One manipulation policy could be filling the toothbrush with a toothpaste that could help an elderly person when is needed. Other manipulations can be swiping the floor or washing the dishes.

The next thing would be implementing even better User Interfaces. The motivation is the Leap Motion provides already some partially built packages for computers. Therefore, that I should not have to implement my own interface from the bottom like I was doing in Ch.5.4.



Bibliography

- [1] v ercart, “Why do Leap Hands have intermediate phalanges on the thumbs instead of metacarpals?” 2018. [Online]. Available: <https://community.leapmotion.com/t/why-do-leap-hands-have-intermediate-phalanges-on-the-thumbs-instead-of-metacarpals/7097>
- [2] Chang Wei Tan Geoffrey I Webb François Petitjean Paul Reichl, “Figure 2: A comparison of Euclidean distance (a) with dynamic time...” 2017. [Online]. Available: https://www.researchgate.net/figure/A-comparison-of-Euclidean-distance-a-with-dynamic-time-warping-b-for-time-series_fig1_317662829
- [3] Leap Motion, “Coordinate Systems — Leap Motion Python SDK v2.3 documentation,” 2021. [Online]. Available: https://developer-archive.leapmotion.com/documentation/v2/python/devguide/Leap_Coordinate_Mapping.html?highlight=interaction%20box
- [4] G. Edwards, C. Taylor, and T. Cootes, “Interpreting face images using active appearance models,” in *Proceedings Third IEEE International Conference on Automatic Face and Gesture Recognition*. Nara, Japan: IEEE Comput. Soc, 1998, pp. 300–305. [Online]. Available: <http://ieeexplore.ieee.org/document/670965/>
- [5] M. Yeasin and S. Chaudhuri, “Visual understanding of dynamic hand gestures,” *Pattern Recognition*, vol. 33, no. 11, pp. 1805–1817, Nov. 2000. [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S0031320399001752>
- [6] W.-T. Shi, Z.-J. Lyu, S.-T. Tang, T.-L. Chia, and C.-Y. Yang, “A bionic hand controlled by hand gesture recognition based on surface EMG signals: A preliminary study,” *Biocybernetics and Biomedical Engineering*, vol. 38, no. 1, pp. 126–135, 2018. [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S0208521617300323>
- [7] G. Marin, F. Dominio, and P. Zanuttigh, “Hand gesture recognition with leap motion and kinect devices,” in *2014 IEEE International Conference*

- on *Image Processing (ICIP)*. Paris, France: IEEE, Oct. 2014, pp. 1565–1569. [Online]. Available: <http://ieeexplore.ieee.org/document/7025313/>
- [8] A. A. Bandala, J. M. Z. Maningo, E. Sybingco, R. R. P. Vicerra, E. P. Dadios, J. D. D. Guillarte, J. O. P. Salting, M. J. A. S. Santos, and B. A. E. Sarmiento, “Development of Leap Motion Capture Based - Hand Gesture Controlled Interactive Quadrotor Drone Game,” in *2019 7th International Conference on Robot Intelligence Technology and Applications (RiTA)*. Daejeon, Korea (South): IEEE, Nov. 2019, pp. 174–179. [Online]. Available: <https://ieeexplore.ieee.org/document/8932800/>
- [9] A. S. Kundu, O. Mazumder, P. K. Lenka, and S. Bhaumik, “Hand Gesture Recognition Based Omnidirectional Wheelchair Control Using IMU and EMG Sensors,” *Journal of Intelligent & Robotic Systems*, vol. 91, no. 3-4, pp. 529–541, Sep. 2018. [Online]. Available: <http://link.springer.com/10.1007/s10846-017-0725-0>
- [10] A. B. Bakri, R. Adnan, and F. A. Ruslan, “Wireless Hand Gesture Controlled Robotic Arm Via NRF24L01 Transceiver,” in *2019 IEEE 9th Symposium on Computer Applications & Industrial Electronics (ISCAIE)*. Malaysia: IEEE, Apr. 2019, pp. 16–22. [Online]. Available: <https://ieeexplore.ieee.org/document/8743772/>
- [11] A. M. Faudzi, M. H. K. Ali, M. A. Azman, and Z. H. Ismail, “Real-time Hand Gestures System for Mobile Robots Control,” *Procedia Engineering*, vol. 41, pp. 798–804, 2012. [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S187770581202646X>
- [12] S. Aoi and K. Tsuchiya, “Locomotion Control of a Biped Robot Using Nonlinear Oscillators,” *Autonomous Robots*, vol. 19, no. 3, pp. 219–232, Dec. 2005. [Online]. Available: <https://doi.org/10.1007/s10514-005-4051-1>
- [13] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, “Robot Programming by Demonstration,” in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Berlin, Heidelberg: Springer, 2008, pp. 1371–1394. [Online]. Available: https://doi.org/10.1007/978-3-540-30301-5_60
- [14] Ajay Kumar, “Small-variance asymptotics for non-parametric online robot learning - Ajay Kumar Tanwani, Sylvain Calinon, 2019,” 2019. [Online]. Available: <https://journals.sagepub.com/doi/full/10.1177/0278364918816374>
- [15] A. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, “Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors,” *Neural computation*, vol. 25, Nov. 2012.
- [16] S. Gomez-Gonzalez, G. Neumann, B. Scholkopf, and J. Peters, “Using probabilistic movement primitives for striking movements,” in

- 2016 *IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*. Cancun, Mexico: IEEE, Nov. 2016, pp. 502–508. [Online]. Available: <http://ieeexplore.ieee.org/document/7803322/>
- [17] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, “Using probabilistic movement primitives in robotics,” *Autonomous Robots*, vol. 42, no. 3, pp. 529–551, Mar. 2018. [Online]. Available: <https://doi.org/10.1007/s10514-017-9648-7>
- [18] G. J. Maeda, G. Neumann, M. Ewerton, R. Lioutikov, O. Kroemer, and J. Peters, “Probabilistic movement primitives for coordination of multiple human–robot collaborative tasks,” *Autonomous Robots*, vol. 41, no. 3, pp. 593–612, Mar. 2017. [Online]. Available: <http://link.springer.com/10.1007/s10514-016-9556-2>
- [19] D. Rakita, B. Mutlu, and M. Gleicher, “RelaxedIK: Real-time Synthesis of Accurate and Feasible Robot Arm Motion,” in *Robotics: Science and Systems XIV*. Robotics: Science and Systems Foundation, Jun. 2018. [Online]. Available: <http://www.roboticsproceedings.org/rss14/p43.pdf>
- [20] Ian McMahon, “RethinkRobotics/baxter_examples,” 2015. [Online]. Available: https://github.com/RethinkRobotics/baxter_examples
- [21] Yong Ann Vouern, “yongan007/inverse_foward_kinematics_kuka_iiwa,” 2019. [Online]. Available: https://github.com/yongan007/inverse_foward_kinematics_Kuka_iiwa
- [22] M. Karam and m. c. schraefel, “A Taxonomy of Gestures in Human Computer Interactions,” 2005, ISBN: 9780854328338 Publisher: s.n. [Online]. Available: <https://eprints.soton.ac.uk/261149/#:~:text=This%20work%20presents%20a%20unique,output%20technologies%20used%20for%20implementation.>
- [23] T. P. Truong, M. Yamaguchi, S. Mori, V. Nozick, and H. Saito, “Registration of RGB and Thermal Point Clouds Generated by Structure From Motion,” in *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*. Venice: IEEE, Oct. 2017, pp. 419–427. [Online]. Available: <http://ieeexplore.ieee.org/document/8265267/>
- [24] D. Mehta, S. Sridhar, O. Sotnychenko, H. Rhodin, M. Shafiei, H.-P. Seidel, W. Xu, D. Casas, and C. Theobalt, “VNect: Real-time 3D Human Pose Estimation with a Single RGB Camera,” *ACM Transactions on Graphics*, vol. 36, no. 4, pp. 1–14, Jul. 2017, arXiv: 1705.01583. [Online]. Available: <http://arxiv.org/abs/1705.01583>
- [25] Christopher Fonnesbeck, “Christopher Fonnesbeck - Probabilistic Programming with PyMC3,” 2021. [Online]. Available: <https://speakerdeck.com/pycon2017/christopher-fonnesbeck-probabilistic-programming-with-pymc3?slide=47>

- [26] D. van Ravenzwaaij, P. Cassey, and S. D. Brown, “A simple introduction to Markov Chain Monte–Carlo sampling,” *Psychonomic Bulletin & Review*, vol. 25, no. 1, pp. 143–154, Feb. 2018. [Online]. Available: <https://doi.org/10.3758/s13423-016-1015-8>
- [27] Jeremy Zhang, “Dynamic Time Warping. Explanation and Code Implementation | by Jeremy Zhang | Towards Data Science,” 2020. [Online]. Available: <https://towardsdatascience.com/dynamic-time-warping-3933f25fcdd>
- [28] 262588213843476, “dtw.py,” 2020. [Online]. Available: <https://gist.github.com/MJeremy2017/8ea397091a62cbaba7cd5eeb314e169a>
- [29] Two Reality, “tworeality-realidad-virtual-inmersiva-oculus-rift-leap-motion.jpg (JPEG Image, 1191 × 510 pixels),” 2015. [Online]. Available: <https://www.tworeality.com/wp-content/uploads/2015/02/tworeality-realidad-virtual-inmersiva-oculus-rift-leap-motion.jpg>
- [30] Leap Motion, “Hand Tracking Overview,” 2020. [Online]. Available: <https://docs.ultraleap.com/hand-tracking/>
- [31] L. Motion, “How Hand Tracking Works | Ultraleap,” 2020. [Online]. Available: <https://www.ultraleap.com/company/news/blog/how-hand-tracking-works/>
- [32] F. Weichert, D. Bachmann, B. Rudak, and D. Fisseler, “Analysis of the Accuracy and Robustness of the Leap Motion Controller,” *Sensors (Basel, Switzerland)*, vol. 13, no. 5, pp. 6380–6393, May 2013. [Online]. Available: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3690061/>
- [33] V. Chawda and G. Niemeyer, “Toward controlling a KUKA LBR IIWA for interactive tracking,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. Singapore, Singapore: IEEE, May 2017, pp. 1808–1814. [Online]. Available: <http://ieeexplore.ieee.org/document/7989213/>
- [34] SketchUp, “3D Design Software | 3D Modeling on the Web,” 2021. [Online]. Available: <https://www.sketchup.com/page/homepage>
- [35] J. Salvatier, T. V. Wiecki, and C. Fonnesbeck, “Probabilistic programming in Python using PyMC3,” *PeerJ Computer Science*, vol. 2, p. e55, Apr. 2016. [Online]. Available: <https://peerj.com/articles/cs-55>
- [36] Stan, “stan-dev/pystan,” May 2021, original-date: 2017-09-17T14:13:04Z. [Online]. Available: <https://github.com/stan-dev/pystan>
- [37] M. J. Mathew, “mjm522/promps_python,” Feb. 2021, original-date: 2018-02-02T20:33:34Z. [Online]. Available: https://github.com/mjm522/promps_python

- [38] W. Cipriano, “wqipriano/pretty-print-confusion-matrix,” May 2021, original-date: 2018-07-03T15:47:33Z. [Online]. Available: <https://github.com/wqipriano/pretty-print-confusion-matrix>
- [39] D. Held, “davheld/tf,” Jan. 2021, original-date: 2014-04-16T22:55:42Z. [Online]. Available: <https://github.com/davheld/tf>
- [40] Rot_Tianers, “Tianers666/lbr_iiwa7_r800-urdf-package,” Mar. 2020, original-date: 2020-03-16T03:17:56Z. [Online]. Available: https://github.com/Tianers666/lbr_iiwa7_r800-urdf-package
- [41] KUKA, “LBR iiwa,” 2021. [Online]. Available: <https://www.kuka.com/en-de/products/robot-systems/industrial-robots/lbr-iiwa>
- [42] C. Miller, “The Meaning of Hand Gestures Around the World | English Live Blog,” 2020, section: English In The Real World. [Online]. Available: <https://englishlive.ef.com/blog/english-in-the-real-world/hand-gestures/>
- [43] Vanessa Van Edwards, “60 Hand Gestures You Should Be Using and Their Meaning,” 2015. [Online]. Available: <https://www.scienceofpeople.com/hand-gestures/>
- [44] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, “Array programming with NumPy,” *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: <http://www.nature.com/articles/s41586-020-2649-2>
- [45] S. Salvador and P. Chan, “FastDTW: Toward Accurate Dynamic Time Warping in Linear Time and Space,” p. 11, 2020.
- [46] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Goullart, and T. Yu, “scikit-image: image processing in Python,” *PeerJ*, vol. 2, p. e453, Jun. 2014. [Online]. Available: <https://peerj.com/articles/453>



Appendix A

List of Abbreviations

Shortcut	Meaning
LMS	Leap Motion Sensor
GDS	Gesture detection system, such as Leap Motion sensor
DOF	Degree of freedom
CPU	Central processing unit
ProMP	Probabilistic Movement Primitives
ROS	Robot operating system
HRI	Human-Robot Interaction
RAM	Random access memory
RGB	Red Green Blue (Image)

Appendix B

ROS messages definitions

Additional variable definitions based on *ROS* messages.

Name	Type
Cartesian position $p = [x, y, z]$	Float[] length 3
Quaternion orientation $o = [x, y, z, w]$	Float[] length 4

Table B.1: Pose definition, [] denotes a vector specification.

Name	Type
seconds s	Integer
nanoseconds ns	Integer

Table B.2: Timestamp definition.

Name	Type
Header	ROS header
Joint names	Strings[]
Joint trajectory points	Joint Trajectory Points[] (see Tab. B.4)

Table B.3: Joint Trajectory definition, [] denotes a vector specification, ROS header serves purpose as unique identifier of message, timestamp and sequence number is included in there.

Name	Type
Positions	Float[]
Velocities	Float[]
Accelerations	Float[]
Efforts	Float[]

Table B.4: Joint Trajectory Point definition, [] denotes a vector specification.

I. Personal and study details

Student's name: **Vanc Petr** Personal ID number: **465926**
Faculty / Institute: **Faculty of Electrical Engineering**
Department / Institute: **Department of Control Engineering**
Study program: **Cybernetics and Robotics**
Branch of study: **Cybernetics and Robotics**

II. Master's thesis details

Master's thesis title in English:

Probabilistic gesture control of a robotic arm

Master's thesis title in Czech:

Pravděpodobnostní přístup k ovládání robotického ramene pomocí gest

Guidelines:

The goal of this thesis is to propose, implement, and test a user-interface which enables online control of the robot via a set of gestures. The possibility to use ProMP (probabilistic movement primitives) [1] for this type of control should be explored in detail and the benefits and weaknesses of this approach evaluated. The gestures might be of different type – either they trigger some action independently on where the gesture is performed (e.g., opening a gripper), or they might be coupled with the robot and the position of the hand during performing the gesture is a parameter used for the robot control. The gestures might be associated with different constraints on the end-effector pose (e.g., keep distance to table, face down), robot control mode (e.g., apply force, impedance, etc.), or set parameters of these movements (e.g., number of shown fingers signals the speed of the end-effector). Hand gestures, especially finger gestures (detected via Leap motion) will be used.

The following steps should be carried out in order to complete the thesis:

1. Defining set of simple manipulation tasks the robot should do (e.g., picking up object and placing it to a goal region, pushing a button, closing a drawer, etc.) and a set of gestures which will be used for robot control (e.g., push gesture, number of fingers showing the force, rotation of the hand to rotate end-effector, opening hand to open a gripper, etc.)
2. Implementation of individual robot motions corresponding to individual gestures (e.g., EEF facing up/down, gripper closing/opening, rotation of the the EEF, etc.).
3. Enabling direct control of the robot - i.e., mapping of finger position to robot end-effector position. Finger pose detected by Leap motion is tracked, and visualised. In parallel, it is used as the set point for the robot's position controller, i.e. the robot follows the path corresponding to the demonstration.
4. Representation of individual gestures via ProMP (or another suitable representation [3]). These representations will be taught via multiple demonstrations of the given gesture. Evaluating the quality of the result based on individual parameters (e.g., input parameters of the model, amount of training data, variability of data, type of representation of the gesture, etc.). A confusion matrix for gesture classification might be created.
5. Explore how to couple the robot movement with the human gesture (e.g., degree of open hand corresponds to the opening of the gripper). This can be done for example via interaction ProMP ([2]).
6. Design of a gesture-based user interface, that allows to control the robot safely (prevent accidental and harmful robot motions) and effectively (all required robot functions can be controlled).
7. (optional) User study evaluating the effectiveness and errorness of the control approach.

Bibliography / sources:

- [1] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Using probabilistic movement primitives in robotics," *Auton Robot*, vol. 42, no. 3, pp. 529–551, Mar. 2018, doi: 10.1007/s10514-017-9648-7.
- [2] G. Maeda, G. Neumann, M. Ewerton, R. Lioutikov, O. Kroemer, and J. Peters, "Probabilistic movement primitives for coordination of multiple human–robot collaborative tasks," *Autonomous Robots*, vol. 41, Mar. 2017, doi: 10.1007/s10514-016-9556-2.
- [3] S. Ameer, A. B. Khalifa, and M. S. Bouhlel, "A comprehensive leap motion database for hand gesture recognition," in *2016 7th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT)*, Dec. 2016, pp. 514–519, doi: 10.1109/SETIT.2016.7939924.
- [4] E. Coronado, J. Villalobos, B. Bruno, and F. Mastrogiovanni, "Gesture-based robot control: Design challenges and evaluation with humans," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 2761–2767, doi: 10.1109/ICRA.2017.7989321.

Name and workplace of master's thesis supervisor:

Jan Kristof Behrens, MSc., Robotic Perception, CIIRC

Name and workplace of second master's thesis supervisor or consultant:

Mgr. Karla Štěpánová, Ph.D., Robotic Perception, CIIRC

Date of master's thesis assignment: **29.01.2021** Deadline for master's thesis submission: **21.05.2021**

Assignment valid until:

by the end of summer semester 2021/2022

Jan Kristof Behrens, MSc.
Supervisor's signature

prof. Ing. Michael Šebek, DrSc.
Head of department's signature

prof. Mgr. Petr Páta, Ph.D.
Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt

Student's signature