Robot Task Classification and Local Manipulation Controllers

Klassifizierung von Roboteraufgaben und lokale Manipulationssteuerungen
Master thesis by Chen Xue
Date of submission: 16. Februar 2022

1. Review: M.Sc João Carvalho
2. Review: M.Sc Suman Pal
3. Review: Prof. Dr. Jan Peters
4. Review: Prof. Dr.-Ing. Marius Pesavento
Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 und §23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Chen Xue, die vorliegende Masterarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß §23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 16. Februar 2022

C.Xue

Abstract

Learning from videos is a novel issue in the field of human-machine interactions, of which the robotic manipulators are expected to reproduce the movement of human demonstrators in video clips. Learning primitive movements in this case leads to a mapping problem that transforms the high-dimensional RGB-data sequences into low-dimensional information, like the label of a particular task or the moving trajectories. We use the time series classifier to dedicated solve the recognition problem, which is fast to build and the final accuracy is up to 88% for our time series trajectories.

In assembly scenarios, tasks like pick-and-place, peg-in-hole, and screw are seen as the most representative and frequently used operations. On one hand, it is challenging to use the conventional controller to handle the task that frequently interacts with the environment, because modeling of the environment is time-consuming and observations of states are often noised, thus the systems often cannot be controlled efficiently. Reinforcement learning provides a possibility that the manipulator can adapt to uncertain circumstances through training. On the other hand, learning RL-based solutions is data inefficient and cumbersome. Thus, we use the sub-optimal classical controller and the RL policy to work on their best part in two sub-spaces, so that their advantages are taken and the shortcomings can be minimized. The classical controller is used to bring the object close enough to the target position and let the RL-based policy handle the rest part. To reproduce these tasks on the robotic platform, we use the Cartesian impedance controller, which is not only agile controlled in a position-loop, but also compliant so that the manipulator keeps from the severe crash and jamming in the environment.

Contents

1 Introduction

Today's robotics technology has gradually integrated into the field of industrial automation, such as widely used in manufacturing, processing and assembly. Due to high reliability, precision and efficiency, industrial robots are often used to precisely control repetitive tasks, which rely on fine pre-programmed trajectories. However, when the robots work in a dynamic environment, or the planted object is in an uncertain location, these sophisticated properties may no longer exist, in other words, deterministic strategies might not suitable to dynamic environments anymore. By contrast, humans can easily learn a specific action by observing alone, instead of painstakingly planning a path for each action. The motivation is just like human that let robots reproduce specific primitive tasks by just watching videos of human demonstrators.

Thus, we divided the problem into two main subsections: perception and reproduction. The target tasks we focus on here are among the most common assembly tasks: pick-and-place, peg-in-hole, and screw. The problem can be reformulated as with a given video clip, identifying the type of action contained in the video and sending it to the robot, the robot recognizes the corresponding behavior to complete this action. In our work, the collaborative robot Panda from Franka Emika is mainly used to achieve these tasks while manipulating in the environment.

First of all, understanding primitive actions require the perception of the movements from videos. The machine learning community has plenty of methods for recognizing actions from videos or image sequences, some of them require encoding high-dimensional images into low-dimensional space, like trajectories of the moving object, other methods build a mapping direct from image to the categories of a certain movement. In our work, we adopt the former means that encoding the vision information into the trajectory sets, because we need for manipulation not only categorizing the type of task but also the position information that is helpful for assembly tasks. We extracted the trajectories from the video by using *Mediapipe* and consider these trajectory sets as our data set. The trajectories are considered as time series data and we use various classification tools to classify the tasks

and reproduce the peg-in-hole task in the simulation environment. We assume that the robot can achieve the task by grasping the object and finding itself a way to insert the peg into the hole. The classical controller in general is reliable because of its excellent dynamical features and robustness, which relies on the fine-grained model of the controller and the system. But the classical policy is insufficient to an unknown environment or a dynamic environment, while reinforcement learning can solve the uncertainty by learning a policy under different conditions. However, reinforcement learning from scratch usually results in unexpected policy and is inefficient to converge. Our solution takes advantage of both methods by combining the strength of both and disabling the weak parts as needed. We use the conventional controller to roughly bring the object close to the target position, then the robot should do a "blind search" by interacting with the environment and in the end insert into the hole, this behavior is just similar to our human beings because humans sometimes sensing the object without seeing with their eyes. The "blind search" part is conducted by reinforcement learning(RL) that lets the robot obtain experience by learning in the task. We call the strategy of the conventional controller in our work as nominal policy. The RL policy will improve the nominal policy to do insertion tasks, both policies work together so that the robot learns faster and more efficiently, which is well-known as the residual policy. The robot during the entire task should be compliant hence it doesn't break the object or cause other damages, we address this requirement by using Cartesian impedance control so that the robot tends to be compliant to the current accelerations caused by total force while in contact.

The distribution of the remaining part of the content is mainly as follows, relevant work done by other researchers in chapter 2, which compared the advantages and limitations of the methods they used and whether these methods are suitable in our case. The methodology and preliminaries in the chapter 3, the details in the task that we achieved in chapter 4. As for the environment settings and the experiment details, like training process, hyperparameters, and so on, are introduced in chapter 6. The rests are the outcomings and our comprehension from this work.

2 Related Work

In this chapter, we present some prior work on task classification and robot manipulation, followed by a brief background on residual policy. Task classification in our work is the prerequisite of learning from the video. When the human action is recognized, the robot can do manipulation to reproduce the action. Section 2.1 is about various ML-based algorithms that classify human motions. Section 2.2 discusses the approaches that the robot manipulates in low-level tasks like grasp, peg-in-hole, etc. The last section 2.3 related to the up-to-date residual policy for agents adaptive to the various and indeterministic environments.

2.1 Task Classification

Many ML algorithms are being used to solve classification problems. For example, that support vector machine(SVM) is seen to be interpretable and had been mostly used for classification in the learning community. However, the no free lunch theorem tells that no universal algorithm outperforms every other classification algorithm in different domains. Bagnall et al. [3] shown in their work the various categories that are dedicated for the time series classification problem, like distance-based, interval-based, and shapelet-based methods. Abanda et al. [1] partitioned these methods into two sorts that according to the extracted features, one is a global method that takes the entire time series into account, the other pay attention to local features of intervals, sliding windows, and so on. Among the distance-based methods, k-nearest neighbors(kNN) is the most common algorithm that measures the similarity from one data instance to another. The Euclidean distance and dynamic time warping(DTW) are usually adopted as the metric for the distance-based method, where the DTW(Berndt and Clifford [4]) was originally used for pattern detection. More studies now use DTW for human movement recognition, like Lee [20] simply use three-dimensional trajectories to measure the gait similarities among

people with different gender, ages, height, etc. However, the global-based distance usually underperforms when the noise in the series occurs and is brittle to subtle differences(Ding et al. [13]). The interval-based algorithms otherwise, like the time series forest(TSF) algorithm that is more robust to outliers, uses local features that transforms the time series into primitive features, such as mean, standard deviation, slope, etc that represent the local features(Deng et al. [11]), the extracted local features are then trained for decision trees. The new instance to be classified will be voted on by all trees in the forest to decide the class that instance belongs to. The shapelet-based method is also popular for time series classification proposed by Ye and Keogh [38], which uses the shapelet as the splitting criterion of the decision tree. The advanced variations of shapelet-based method are compared by [2] that the rotation forest outperforms the other shapelet-based methods for continuous features, but the drawback is the forest is slow to build if the attributes for each tree are too many since it is a tricky job and not the main point to our robot manipulation, we do not consider the shapelet transform method and its variations. The neural-networks-based method has not been considered either for our work due to the following limitations: hyperparameter tuning, limited data instances, and slow to train.

2.2 Robot Manipulation

The common challenge in intelligent robotics is to let robot achieve their goals by directly interacting with the environment(Billard and Kragic [5]). Robots today are capable of doing plenty of tasks like pick-and-place, stack, push, pour, and so on, but most robots still use a parallel jaw gripper in many applications that lack dexterity and good sensing like human hands. It is well known that robots can perceive environment information without involving in it, like recognizing and localizing the object by receiving camera images, like the works of Burchfiel and Konidaris [8], Song et al. [34], wherein the manipulation tasks are mainly vision-based, which is known as the passive perception method. Different from passive perception, the interactive perception (Bohg et al. [7]) lets the robot physically interacts with the environment so that latent states can be acquired while doing interactions with surroundings. The interactive perception-based assembly problem: peg-in-hole, is widely investigated and researched under various conditions.

The peg-in-hole assembly task according to Xu et al. [37] can be divided into contact model-based and contact model-free strategies. The former strategy relies on the accurate modeling of the contact point where both mating objects contact. Kim et al. [18] proposed

precision hole detection algorithm that only using position control and F/T sensor. In their work, the vision system with 4 to 5mm estimation error helps to roughly determine the distance between the peg and the hole. They adopted admittance control that the force is controlled in the outsider control-loop so that stable contact is ensured. However, the detection of contact edge is intricate and hard to disambiguate to other shapes and dimensions. Park et al. [26] proposed a compliance-based peg-in-hole method without force feedback. They did not control the force in the feedback loop but applied scaled force under three different conditions: pushing when no contact detected, rubbing when contact established, wiggling when hole detected and the same time screwing, this approach is quite similar to human beings but difficult to sense and recognize the state. At present, robots are more expected to recognize the environment interactively, similar to humans. For this reason, reinforcement learning(RL) based methods emerging rapidly and are looking forward to generalizing to unseen environments. Oikawa et al. [24] solved the contact-rich insertion task, as well as a gear-insertion task by using RL algorithm to online, generate stiffness matrices to improve the trajectory tracking ability. The RL-based stiffness matrices output can be seen as another kind of force shaping that has limited perception of the environment because the robotic manipulation is still done by a given trajectory. Inoue et al. [16] provided a promising RL-based method that without having a predetermined local trajectory, they trained a recurrent neural network by using reinforcement learning to achieve high precision assembly tasks. RL with LSTM is an intelligent substitute to the contact model-based approaches that use short-term memory layers to approximate the Q-function. They observed the states that are composed of forces and torques as well as relative joint angles by the robot encoders, which is much more generalized than the trajectory-generation methods. In the fine-tuning phase, only selected force and torques are measured. Compare to their work, Kulkarni et al. [19] used the relative distance in the Cartesian space to replace the angle distance, i.e. from the grasped object to the goal position in the goal frame The forces that act on the robot end-effector are also measured as the states for RL policy. The differently applied residual policy that they combined position-based controller and the LSTM-based RL policy. A decay factor is used to temporally scale the importance that position-based controllers and RL controllers have. They obtained the results surprisingly fast on the real robot without implementing in simulation. However, the force sensor is expensive and not available for every robot. The decay factor is still a hyperparameter that needs to be considered empirically.

In our work, we proposed a more generalized method only by perception of the position via interaction with the environment but based on the relative distance, which is investigated in the chapter 4.

2.3 Residual Polciy Learning

Initially, the residual policy was proposed to solve the data-inefficient problem in the reinforcement learning approach. Saveriano et al. [31] investigated this issue and applied it in the dynamic system, they learned residual difference via the Gaussian process and combined it with the simplified, parameterized dynamic model. The differences between the real system and the simplified system are assumed to be the Gaussian distribution, which is less strict than assuming the entire real system is Gaussian distributed. The PI-REM algorithm they put forward reduces the iterations and execution time in the Cart-pole experiment, which greatly outperforms the PILCO algorithm. Johannink et al. [17] used residual RL policy for robot control in the real world with noisy oriented blocks, which is difficult for conventional feedback control to address. The residual part was not explicitly defined but needs to contribute to the maximization of the reward that the surrounding blocks keep stable and not fall. Many other attempts tried to improve the low-level control performance are discussed by Hynes et al. [15], Staessens et al. [35], which are proven to be more efficient than the fine-grained original control scheme and significantly improved the performance on slider-crank and quarter-car suspension control. Silver et al. [33] elaborated the shortcomings of Rl from scratch, wherein the nondifferentiable circumstances are difficult for conventional approaches to deal with. They study residual policy learning in six challenging MuJoCo tasks involving partially observable, sensor noise and control miscalibration, and so on. The learned policy exhibits interesting behavior that is not expected but qualitatively complete the assignment, which proved the residual policy is well suited for complex manipulation tasks that improve a good but imperfect initial policy.

3 Foundations

3.1 Task Classification

A common task for motion recognition is a classification problem, which assigns data to some specific classification model to get relevant type labels. Since motion and action have some temporal correlation, the algorithms of our interest require the data to be time-series samples with features, such as image sequences or trajectory coordinates.

Differing from tabular classifiers that treat each time point as a separate feature and learn a model via common ML algorithms, time series classifiers also use the information that correlates data in order of time. Since the results obtained by time series classification algorithms already meet the demand of our usage, methods like CNNs and LSTMs are deliberately avoided due to the carefully selected architecture, limited data samples, massive computation, and hyper-parameter tuning. To compare different algorithms, we introduce the following methods that we applied.

3.1.1 Support Vector Machine

Support Vector Machine (SVM) is a supervised machine learning algorithm that was originally designed for binary classification but can be extended to both regression and classification(Murphy [23]). In the SVM algorithm, each data point is defined in an N-dimensional space, where N is the number of features. The goal of SVM is to separate points into binary point sets by an optimal N-dimensional hyperplane. This hyperplane is selected by finding the maximum margin among the infinite hyperplanes that pass through data points. The best hyperplane locates where the distance to the two classes is maximum. Support vectors are the points closest to the hyperplane, and the distance between the closest points and the hyperplane is called margin.

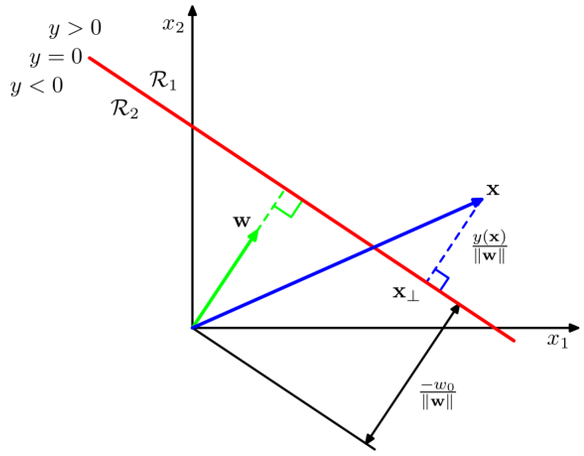


Figure 3.1: The geometry of decision surface in 2D case, based on Figure 4.1 of [6]

The 2D data is linearly separable by using a single straight line, the SVM in this case called linear SVM. The N-dimensional data is not linearly separable when $N > 2$, then we can use Non-Linear SVM, which is the most common case in the real world. The data points can be projected to higher dimensions utilizing kernel function. The discriminant function is defined as: $f(\mathbf{x}) = \sum_{j=1}^{d} w_j x_j = \mathbf{w}^T\mathbf{x} + w_0$. For $\mathbf{x}$ on the hyperplane holds $f(\mathbf{x}) = 0$, where $\mathbf{w}$ is a vector that is perpendicular to the decision boundary and w_0 defines the bias parameter that controls the distance of the decision boundary from the origin.
The negative label y = -1 stands for points $\mathbf{x}$ at the left area of the hyperplane and y = 1 for right cases, i.e. $y = \text{sign}(f(\mathbf{x}))$.

3.1.2 Dynamic Time Warping

Most time-series research in the field of data mining has focused on distance metrics that can be used for clustering, query, and classification. Euclidean distance(ED) and dynamic time warping(DTW) is often considered as the standard benchmark for distance measures(Bagnall et al. [3]).

The vector in our case is denoted in bold, and matrix is denoted in capital bold, so that a time sequence can be denoted as $\mathbf{X} = [\mathbf{x}^1, \mathbf{x}^2, ..., \mathbf{x}^l]^T \in \mathbb{R}^{l \times d}$, where $\mathbf{x}^i \in \mathbb{R}^{d \times 1}$ for i^{th}

time point of d-dimensional time series in total length l. A label y is always associated with every time sequence. The classification problem is now formulated as a mapping from the space of inputs into the probability distribution over the class categories.

The Euclidean distance between two equal-length time series $\mathbf{X_1}, \mathbf{X_2}$ can be easily computed as

$$\text{ED}(\mathbf{X_1}, \mathbf{X_2}) = \sqrt{\sum_{i=1}^{l}\sum_{j=1}^{d}(x_1^{i,j} - x_2^{i,j})^2},$$

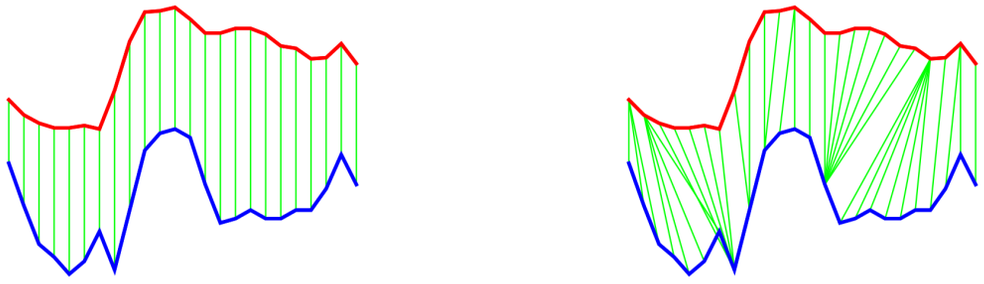
where the distances are derived from the same time points of two sequences. The limitation of ED is obvious that the signals with different rates or scaled differently can not be directly aligned. DTW is different from ED in that the optimal alignment of two sequences is found to compute the minimum distance(Tan et al. [36]). To align two sequences, the DTW algorithm builds an $l_1 \times l_2$ matrix, where the $(i,j)^{th}$ element of the matrix is the associated squared distance $d(x_1^i, x_2^j) = (x_1^i - x_2^j)^2$ that denote how good the alignment between i^{th} time point of sequence X_1 and j^{th} time point of sequence X_2. The optimal warping path is the path walking through the matrix that has the minimum accumulated sum of all distances along the path. The summation of distances of the path in length K

$$\text{DTW}(\mathbf{X_1}, \mathbf{X_2}) = \min\left\{\sqrt{\sum_{k=1}^{K} w_k}\right\},$$

is the warping cost of DTW algorithm, where the w_k is the k^{th} element on path that also denotes $(i,j)^{th}$ element of the distance matrix, therefore $w_k = d(x_1^i, x_2^j)$. The warping path W always starts from the first time point and ends with the last time point of interesting sequences. The element of warping path w_k(for $k = 1, ...K$) is selected by dynamic programming

$$\text{DTW}(\mathbf{X_1}^{1:i}, \mathbf{X_2}^{1:j}) = d(x_1^i, x_2^j) + \min\left\{\begin{array}{l}\text{DTW}(\mathbf{X_1}^{1:i-1}, \mathbf{X_2}^{1:j-1}) \\ \text{DTW}(\mathbf{X_1}^{1:i-1}, \mathbf{X_2}^{1:j}) \\ \text{DTW}(\mathbf{X_1}^{1:i}, \mathbf{X_2}^{1:j-1})\end{array}\right\},$$

where $d(x_1^i, x_2^j)$ indicates the distance between i^{th} element of $\mathbf{X_1}$ and j^{th} element of $\mathbf{X_2}$, and notation $1:i$ and $1:j$ describes the time series from the beginning 1 till the time position i or j, because the cumulative distance metrics here is considered. The algorithm kNN is originally using euclidean distance, can be now rewritten by replacing



(a) Euclidean Distance (b) Dynamic Time Warping

Figure 3.2: The example of two time series aligned by ED in 3.2a and DTW distance in 3.2b of [36]

the ED metric with DTW metric. kNN with DTW is commonly applied for time series classification due to its simple and robust and highly interpretable, however kNN-DTW needs to calculate each sample to all the other samples which lead to a plenty of space and time to compute.

In a naive implementation, the time complexity of the Euclidean distance algorithm is O(N), while DTW during the dynamic programming has a quadratic time complexity O(NM). The space consumption according to Manacher and Hirschberg [22] can be reduced to O(min(N,M)). Although the DTW algorithm consumes much more time than ED, it still acquires more reliable results and is not restricted to having the same length of time sequences. To speed up the computation of ED and/or DTW, the algorithms can be optimized by using the squared distance, lower bounding, early abandoning, etc. More details to optimization can be found by Rakthanmanon et al. [28].

3.1.3 Time Series Forest

Although the kNN-DTW is robust to distortion of the time and has quite satisfactory results, it involves limited views of the temporal features that are beneficial for distinguishing one-time series from the other classes(Deng et al. [11]). Temporal features also referred to as interval features, as an interval-based tree-ensemble classifier, time series forest(TSF) adopts the entrance gain to evaluate splits, where the entrance means the combination of distance measure and entropy gain.

Interval features are some particularly selected features from a time series interval. The

simple and representative features are considered, such as mean and standard deviation. Given K feature types and $f_k(\cdot)(k = 1, 2, ...K)$ is the operation to have k^{th} type of feature. In time interval $[t_1, t_]$, the k^{th} type of feature $f_k(t_1, t_2)$ for the typical three features are computed as $f_1(t_1, t_2) =$ Mean$(\mathbf{X^{t_1:t_2}})$, $f_2(t_1, t_2) =$ Standard Variation$(\mathbf{X^{t_1:t_2}})$, $f_3(t_1, t_2) =$ Slope$(\mathbf{X^{t_1:t_2}})$ where $\mathbf{X}^{t_1:t_2}$ denotes the series interval of total time series $\mathbf{X}$ between time t_1 and t_2, we use the same symbol as in 3.1.2 to keep the notation consistent. The feature space of naive implementation of TSF is $O(N^2)$, and can be reduced to $O(N)$ per tree node by using random sampling strategy(Deng et al. [11]).

The best way to split a node in a tree is to consider a splitting criterion in a time series tree of a time series forest, this criterion lets the original series split into random intervals with random start locations and random length. The interval instances are satisfied and split to the left child node if the condition $f_k((t_1, t_2) \leq \tau$ holds to the right child node if the condition is not satisfied, where τ denotes the threshold. The threshold τ for specific feature f_k is selected by equally dividing κ times from the interval $[minf_k^n(t_1, t_2), maxf_k^n(t_1, t_2)], n \in 1, 2, ...N$, where N is the number of samples in the data set. Since the entropy gain is considered in the splitting, let $E = -\sum_{c=1}^{C} p_c \log p_c$ defines the entropy at the node, where p_c is the proportion of the particular class c out of this node. The entropy gain ΔE then defined as the entropy deviation from the child node to the parent node. For the case that the multiple splits have the same ΔE, the extra metrics called margin is imported to build the entire criterion of splitting, the margin for $n \in 1, 2, ...N : M = \min|f_k^n(t_1, t_2) - \tau|$. The combination of entropy gain and margin is referred to as Entrance Entrance $= \Delta E + \alpha \cdot M$. Figure 3.3 shows the way that entrance split six instances in one dimension, where the instances are represented as three class in colors of blue, red and green. Due to the same distribution of each class, ΔE of each case are all the same so that the selected splits for entropy gain are shown as S_1, S_2 and S_3, and S_3 is the best candidate for taking margin into account.

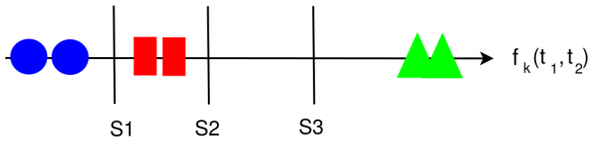


Figure 3.3: The illustration of six samples that are associated with three splits S_1, S_2, S_3 that represents the same ΔE, the entrance, however, splits the green instances best.

3.2 Cartesian Impedance Control

Due to enhanced sensory capabilities, the research on robot force control has thrived over the past few years. In practice, the contact force and moment are rapidly grown if the deviation between the reference trajectory and the end-effector can not be ignored. The control system will also enhance the output to reduce this deviation, one possible result is that the environment or object components are eventually damaged. This downside can be overcome if the manipulator can behave compliant movement. The compliant behavior in contact tasks can be implemented by passive and active interaction control(Siciliano and Khatib [32]). Unlike the passive interaction control, the active interaction control considers the interaction forces as well as exchanged mechanical work(Peters [27]).

There are many perspectives to classify the control type. We adopt the classification of Calanca et al. [9], which can be found in figure 3.4. As we can see there that the impedance control belongs to the type indirect control of active interaction control, which execute force control via motion control without explicit closure loop of force feedback, while the direct control requires a feedback loop so that the output force to stay in a set value or trajectory(Siciliano and Khatib [32]).

Our objective here is to use the compliant control without any measure of the force, according to the discussion above we select here impedance control in the Cartesian space.

The impedance control is usually considered as a virtual equilibrium problem if a set point is taken into account. Just like a spring-damper system, the end-effector of the robot only reaches the set-point if there is no external forces acting on the robot. A general dynamic model of the robot can be considered as:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau} + \boldsymbol{\tau}_{ext}, \tag{3.1}$$

where $\mathbf{M}(\mathbf{q})$ is the inertia matrix, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ is the Coriolis and centrifugal matrix, $\mathbf{g}(\mathbf{q})$ is the gravity torques vector, and $\boldsymbol{\tau}_{ext}$ is the external torques. It is essential that the transformation between configuration space and the task space because we care about the performance in Cartesian space, this transformation is described as $\mathbf{f} : \mathbb{Q}^n \to \mathbb{R}^m$, i.e. $\mathbf{x} = \mathbf{f}(\mathbf{q})$ for $m = 6$ for Cartesian space and n is the degrees of freedom of the robot. By Jacobian matrix $J(q) \in \mathbb{R}^{m \times n}$ for end-effector, we know the relationship between the configuration and Cartesian space $\dot{\mathbf{x}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}$, correspondingly we have:

$$\ddot{\mathbf{x}} = \dot{\mathbf{J}}(\mathbf{q})\dot{\mathbf{q}} + \mathbf{J}(\mathbf{q})\ddot{\mathbf{q}} \tag{3.2}$$

$$\mathbf{u} = \mathbf{J}^{\mathbf{T}}(\mathbf{q})\mathbf{F}_x, \tag{3.3}$$

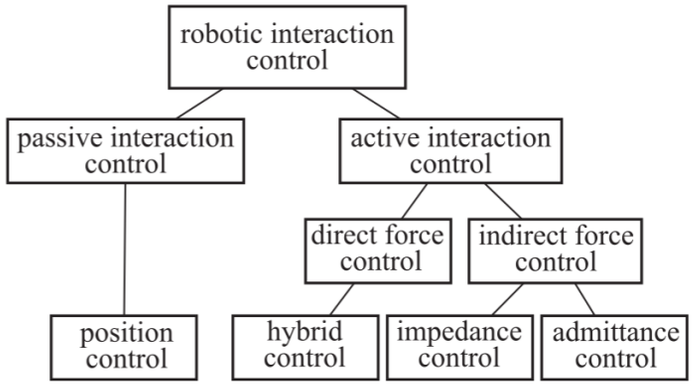


Figure 3.4: The overview of interaction control of [9]

which is also applies to the relationship of the forces in the two spaces, where $\mathbf{F}_x$ denotes the total forces applied on the end-effector, i.e. in Cartesian space $\mathbf{u} = \boldsymbol{\tau} + \boldsymbol{\tau}_{ext}$. The force of equation 3.3 can also represented with inertia matrix in operational space $\mathbf{M}_x$ as:

$$\mathbf{F}_x = \mathbf{M}_x(\mathbf{q})\ddot{\mathbf{x}}_{des}. \tag{3.4}$$

The compliant behaviour can directly revealed by equation 3.4 that the $\ddot{\mathbf{x}}_{des}$ describes the desired acceleration under the influence of the force $\mathbf{F}_x$. We substitute equation 3.4 into equation 3.3 and yield: $\mathbf{u} = \mathbf{J^T}(\mathbf{q})\mathbf{M_x}(\mathbf{q})\ddot{\mathbf{x}}_{des}$. From equation 3.1 we have the acceleration in configuration space $\ddot{\mathbf{q}} = \mathbf{M^{-1}}(\mathbf{q})[\mathbf{u} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{g}(\mathbf{q})]$.
After substituting and rearranging it into equation 3.2, correspondingly we get the acceleration in task space

$$\ddot{\mathbf{x}} = \dot{\mathbf{J}}(\mathbf{q})\dot{\mathbf{q}} + \mathbf{J}(\mathbf{q})\mathbf{M^{-1}}(\mathbf{q})[\mathbf{J^T}(\mathbf{q})\mathbf{M_x}(\mathbf{q})\ddot{\mathbf{x}}_{\mathbf{des}} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{g}(\mathbf{q})].$$

The nonlinear term $\dot{\mathbf{J}}(\mathbf{q})\dot{\mathbf{q}} - \mathbf{J}(\mathbf{q})\mathbf{M^{-1}}(\mathbf{q})\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) - \mathbf{g}(\mathbf{q})$ is ignored and eventually we have:

$$\ddot{\mathbf{x}} = \mathbf{J}(\mathbf{q})\mathbf{M^{-1}}(\mathbf{q})[\mathbf{J^T}(\mathbf{q})\mathbf{M_x}(\mathbf{q})\ddot{\mathbf{x}}_{\mathbf{des}}.$$

Our objective is to let the end-effector in the end behave the same as the outer force tends to, so let $\ddot{\mathbf{x}} = \ddot{\mathbf{x}}_{des}$ to achieve compliant behavior, we have the inertia matrix in Cartesian space:

$$\mathbf{M_x}(\mathbf{q}) = [\mathbf{J}(\mathbf{q})\mathbf{M^{-1}}(\mathbf{q})\mathbf{J^T}(\mathbf{q})]^{-\mathbf{1}}.$$

Now we consider the output torque with gravity compensation:

$$\tilde{\mathbf{u}} = \mathbf{J}^{\mathbf{T}}(\mathbf{q})\mathbf{M}_{\mathbf{x}}(\mathbf{q})\ddot{\mathbf{x}}_{\mathbf{des}} + \mathbf{g}(\mathbf{q}).$$

We use PD controller to track the desired behaviour $\ddot{\mathbf{x}}_{\mathbf{des}} = \mathbf{S}(\mathbf{x}_{\mathbf{des}} - \mathbf{x}) + \mathbf{D}(\dot{\mathbf{x}}_{\mathbf{des}} - \dot{\mathbf{x}}))$, where $S \in \mathbb{R}^{m\times m}$ is the desired stiffness matrix and $D \in \mathbb{R}^{m\times m}$ stands for damping matrix, in Cartesian space holds m=6. In the end, the total torque applied on joint space for robot control is:

$$\tilde{\mathbf{u}} = \mathbf{J}^{\mathbf{T}}(\mathbf{q})\mathbf{M}_{\mathbf{x}}(\mathbf{q})[\mathbf{S}(\mathbf{x}_{\mathbf{des}} - \mathbf{x}) + \mathbf{D}(\dot{\mathbf{x}}_{\mathbf{des}} - \dot{\mathbf{x}})] + \mathbf{g}(\mathbf{q}). \tag{3.5}$$

So far we have introduced how to derive the output torque of impedance control in the Cartesian coordinate system, this result can be used in the simulation environment and real robot. According to proposition 3.4 of [25] that the globally asymptotical stability is ensured when the $\mathbf{F}_{ext} = 0$ and the stiffness matrix $\mathbf{S}$ and damping matrix $\mathbf{D}$ are symmetric and positive definite matrices. We have not discussed yet how to design meaningful stiffness matrix $\mathbf{S}$ and damping matrix $\mathbf{D}$, this does not mean that we will neglect to consider the content of this part the selection of them both is even tricky, but due to space constraints of this thesis, we will not expand too much here and only introduce the relevant part to this application. The stiffness matrix is usually constant and defined depending on the application. Since the inertia matrix is non-diagonal and time-varying, it is better to use a positive definite position-dependent damping matrix instead of the constant damping matrix $\mathbf{D}$, although the stability of the system still holds[25]. Very extensive literature like [25][29][30] can be referenced to confirm this damping matrix, and it is normally formulated as:

$$D_i = 2\xi_i\lambda_{S,i} \tag{3.6}$$

where $\lambda_{S,i}$ is the i^{th} eigenvalue of the stiffness matrix $\mathbf{S}$ and ξ is a damping factor that to be chosen in the range of 0 and 1.

3.3 Residual Policy Learning

3.3.1 Reinforcement Learning

In RL algorithms, an agent observes state $s_t \in \mathbb{R}^m$ at a discrete-time t, and takes an action $a_t \in \mathbb{R}^n$ that generated according to a policy μ which maps a_t from a state s_t.

For each action is taken, the agent then observes a immediate reward $r_t(s_t, a_t)$ and the corresponding state s_{t+1} of the next discrete time. The cumulative future reward from the particular state s_t with the discount factor $\gamma \in [0, 1]$ is given as $R_t = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$, the discount factor defines how much we value rewards over the time, i.e. a smaller γ gives the short-term more attention and the larger γ tend to wait longer for larger amounts of rewards in the future. The agent during this time aims to learn a policy that maximizes the expected future reward if the start distribution is given. An episode defines the process that the task starts and terminates after a certain time length T or the termination condition fulfills. The agent learns the policy after each episode terminates by evaluating the total reward of the entire episode. The main challenge for the agent to learn is to set up an appropriate reward function that motivates the agent to learn correctly. The total RL process described above is assumed as a Markov decision process(MDP) that is fully observable for each state.

In MDP, the value function is applied to evaluate how good or bad a given state could be, and the action-value function can indicate an anticipated future reward if an action is being performed. In discrete MDPs, the value function can be represented by a lookup table, however, for the case that the states and actions are in continuous space, it is impossible to store every individual case in the table. The common method to solve this problem in continuous MDP is using function approximation for generalization of the state features to indicate value function as well as policy function.

3.3.2 Deep Reinforcement Learning

As mentioned in 3.3.1, it is possible to represent value functions with tabular method. And it is more convenient for the neural network to deal with the approximation of value functions in continuous space without caring about the feature expression. With the help of the neural network, the value functions can be optimized in a larger space with only small storage compare to tabular methods. Although the RL algorithm has the overestimation problem due to the radical learning process, there are algorithms that prevent this from happening, like double DQN, actor-critic based methods like DDPG, TD3, etc. We are not going into detail about the RL algorithm we applied, but only discuss how the RL can be combined with nominal policy in chapter 4 section 4.1.3.

3.3.3 Residual Policy

The residual policy is a method that heightens the initial policy with the model-free deep RL algorithms. The different roles of the RL term in residual policy are discussed by Silver et al. [33] that depends on whether the initial policy is ideal or not. If the initial policy is nearly perfect, the additional part to the initial policy is exclusively considered as a slight correction. But when the initial policy is near to ideal, the additional term is then dominant and the initial part gives small hints to the RL for exploration, thus the learned policy part here plays a more significant role than the former case. These two expressions represent both ends of the residual algorithm spectrum. We prefer the first case that take the initial policy in a leading place so that the learning turns to more data-efficient than learning from scratch.

We define an initial policy μ that μ: $\mathcal{S} \rightarrow \mathcal{A}$ with states $s \in \mathcal{S}$ and actions $a \in \mathcal{A}$, and the residual function to be learned is denoted as: $\pi_\theta(s) : \mathcal{S} \rightarrow \mathcal{A}$, the entire residual policy thus denoted as:

$$\mu_\theta(s) = \mu(s) + \pi_\theta(s). \tag{3.7}$$

Since that the parameters θ only decide the residual function π_θ in the entire policy, we have

$$\nabla_\theta \mu_\theta(s) = \nabla_\theta \pi_\theta(s), \tag{3.8}$$

which means that the gradient of the residual policy is only dependent on the residual function $\pi_\theta(s)$ but not affected by the initial policy $\mu(s)$. Hence, the residual policy can be learned by gradient-based methods regardless of the nondifferentiable property of the initial term $\mu(s)$. The initial term is also called nominal policy in this work, the nominal policy we applied in our experiments is position-based control which will be fully discussed in chapter 4.

It is worth mentioning that the residual part should not have output initially since we assume the $\mu(s)$ works perfectly at the very beginning. We then initialize the residual part $\pi_\theta(s) = \mathbf{0}$ accordingly by setting the weights of the last layer to zero(Silver et al. [33]). The reason for only setting the last layer is to let the neurons hold the different weights so that the weights are still changeable, because the changes of weights will be absorbed by the next hidden layer if there are more than one hidden layers.

4 Methodology

4.1 System Overview

In this work, we intend to present the solution to assembly task problem in the Cartesian space by employing the robotic manipulator Franka-Emika panda with a redundant 7-DOF robotic arm and its 2-finger robotic gripper. The primary features of our system are the ability to distinguish the primitive tasks pick-and-place, peg-in-hole, and screw, then achieve the peg-in-hole task with low-level control using residual policy learning, the structure of the system is shown in figure 4.1, which simply describes the system pipeline that starts from the left block, that shows an input video that filmed by Azure Kinect camera as well as by Opti-Track that recorded the trajectories of the object. The video is encoded from raw image sequences into the 3D trajectories of the human hands by using *Mediapipe*(Lugaresi et al. [21]), and the object trajectories are obtained by Opti-Track system. These trajectories are fed into classifier to output the task category, and the category label is sent to the right block so that the manipulation task can work correspondingly. The approach we adopt is present in this chapter, section 4.1.2 introduces the main components of the environment and the way they interact with each other. Section 4.1.3 presents the nominal method that the control without any uncertainty involving, section 4.1.4 describes the combination of the nominal controller and the learned RL controller.

4.1.1 Feature Selection for Classification

The purpose of feature selection is to reduce the dimension of input variables for a predictive model. The feature we used here is the trajectories of hand landmarks defined in *Mediapipe*(Lugaresi et al. [21]), plus the trajectories captured from the Opti-Track in our lab. The brief illustration of the landmarks is demonstrated in figure 4.2. To

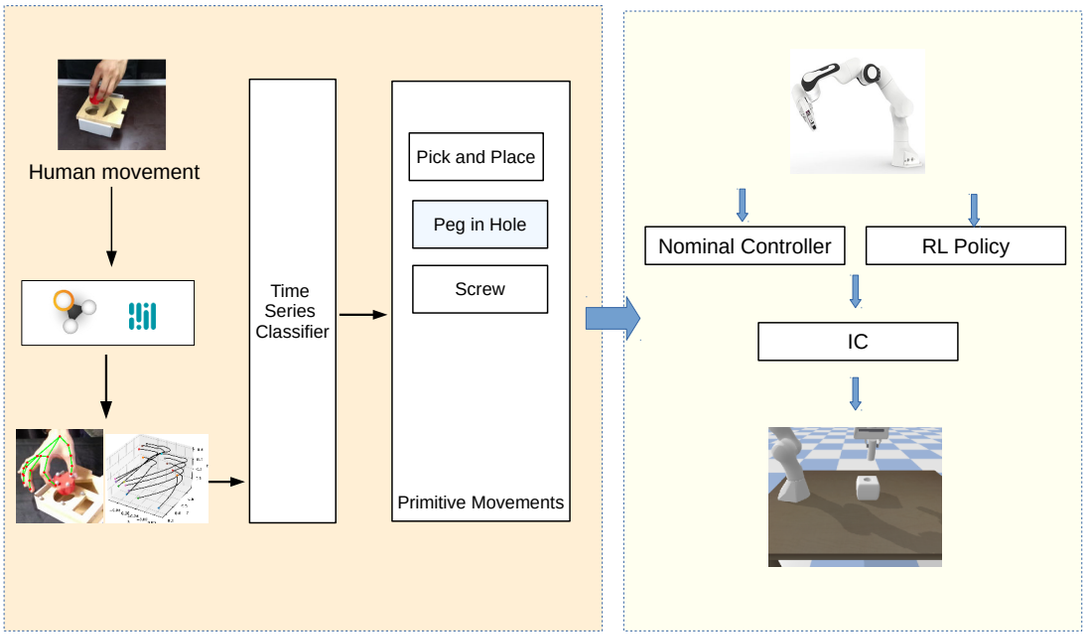
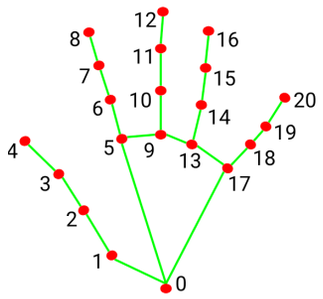


Figure 4.1: The framework of the manipulation system. The left block shows the time series classification, the right block shows the control structure in simulation.

remain less but enough features we only adopt nine landmarks from index 0 to index 8. Empirically, using all these nine landmarks has a higher accuracy rate than only using some landmarks within 0∼8 landmarks. In time length T, the position of all nine landmarks in three-dimension x, y, z of the hand is $\mathbf{H_p^T} \in \mathbb{R}^{27}$, the position of the object are $\mathbf{O_p^T} \in \mathbb{R}^3$ and $\mathbf{O_o^T}$ is the set of quaternions that made a 4-dimensional vector space. Thus, the states of the trajectories can be denoted as $(\mathbf{H_p^T}, \mathbf{O_p^T}, \mathbf{O_o^T}) \in \mathbb{R}^{34}$. The bold character here exactly implies that the state is not an individual time stamp, but a time series formulated trajectory. Since the data dimension is too large, we also used less feature to train by three algorithms,in which the DTW-kNN was coded by ourselves, SVM and TSF use the packages sklearn and sktime respectively.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Figure 4.2: Illustration of landmarks representation. Each red dot stands for a key point of the human hand that can be recognized by *Mediapipe* algorithms. The indices on the landmarks are associated with the joint name of the human hand.

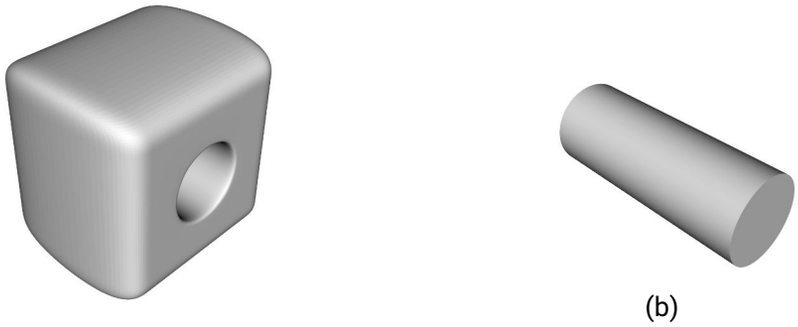
4.1.2 Environment Settings

To demonstrate the effectiveness of our robot manipulation system, we build an environment that specific for the peg-in-hole task which consists of the robotic manipulator Franka-Emika Panda and the objective workpieces including a cylindrical peg, and a hollow cuboid with a cylinder cut out in it, the models of these objects are present in Figure 4.3. To design these workpieces we use *Blender 2.80*(Community [10]), a free and open-source 3D modeling and rendering package.

To ensure the modules can correctly interact with each other, the geometry of both objects should be made carefully. We did subdivision to the model, and loop the edge of the hole so that the surface curved with clean wireframes. The insertion allowance between to objects is 1cm, which can be also inferred refer to figure 4.3.

In this work, the objects to be assembled are set to peg and hole, where the hole is cut out from the cuboid and cuboid is fixed on the table. The peg is assumed firmly grasped by the robotic gripper at the end-effector, we assume that the interaction force during contact does not let the grasped object move relatively from the gripper, which is exactly the meaning of "firmly grasped". Due to this feature, the external reaction force applied to the peg will cause similar effects to the end-effector. Therefore, the transformation between the end-effector and the peg is assumed to be not time-varying but keeps the constant relative transformation, so we say there is a rigid coupling between the peg and the robotic gripper(Kulkarni et al. [19]).

This assembly task can be reformulated as the representations of the frames. We define



(a) (b)

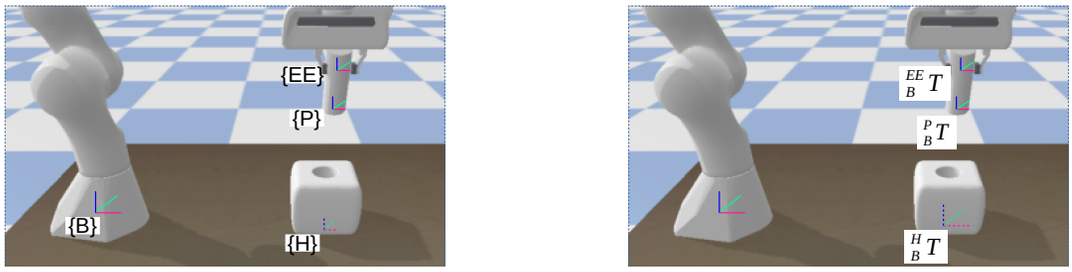
Figure 4.3: Designed 3D models specific for this work. Left figure 4.3a: the cuboid with dimension 16 $\times$ 16 $\times$ 13 mm; the hole inside the cuboid has radius 30mm, right figure 4.3b shows the model of peg with radius 25mm and height 16mm.

the frames of the components in the environment as $\{B\}, \{EE\}, \{P\}, \{H\}$ that represent the frames of robot base, end-effector, peg, and hole respectively. Figure 4.4a shows the settings of the environment and the relations between the frames. The assembly task is being done by driving the frame $\{P\}$ closer to the frame $\{H\}$, note that the $\{H\}$ is the frame that attaches to the bottom of the cuboid and $\{P\}$ attaches the bottom of the peg as well. Since the position of the peg can not be directly measured but must be calculated concerning the position of the end-effector. The transformation matrix from the end-effector frame to the peg frame is denoted as ${}^{P}_{EE}T$. Then, the transformation matrix to the peg frame with respect to robot base frame can be described as ${}^{P}_{B}T = {}^{P}_{EE}T\ {}^{EE}_{B}T$, which in detail is:

$$ {}^{P}_{EE}T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta d \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1} $$

where the Δd is explicitly depicted in figure 4.5. The problem then becomes that through driving the end-effector and the peg correspondingly, the frame of the peg has to eventually coincide with the frame of the goal.

In our work, the goal frame is designed as the same as the $\{H\}$, peg in this case, can reach



(a) (b)

Figure 4.4: The representation of frames and transformation matrices in the environment. The objective of this work is to move the $\{P\}$ so that the $\{P\}$ is close to $\{H\}$ as much as possible. We simplify the task by introducing a nominal controller that is capable of bring the end-effector near to the above opening of the hole.

the very bottom of the cuboid, because of figure 4.3 we know the peg length is a bit longer than the hole height, such that a complete insertion is guaranteed. The most difficult part for the insertion task always happens at the beginning that two objects start matching to each other using the orientation adjustment. We do not let the gripper actively release peg over running the simulation, because our focus is on the insertion and regardless of the high-level policy, like releasing peg halfway to let it fall freely. By doing this, the experiment and calculation can remain straightforward. The transformation that represent the peg position in the goal frame is ${}^{P}_{H}T = {}^{B}_{H}T\,{}^{EE}_{B}T\,{}^{P}_{EE}T$, which is found that related to figure 4.4b.

It must be clear that to keep the generality we express the peg position as $\mathbf{p} = (x_p, y_p, z_p)$ in the frame $\{H\}$, the (x_p, y_p, z_p) can then be calculated in the base(world) frame and subsequently transformed in $\{H\}$, i.e. as in figure 4.5 illustrates, the representation ${}^{H}\mathbf{p} = {}^{H}_{B}T^{B}\mathbf{p}$ is the the position of the peg in the hole frame.

4.1.3 Nominal Control Scheme

As we mentioned before, in the peg-in-hole part, we aim to solve the assembly task with a low-level control policy. We assume that the experiment starts from the gripper already grasped the peg. To keep it general, the peg is randomly placed on the table where near to the cuboid. The nominal control is the policy to move the peg from the table to the place close to the hole opening so that the insertion adjustment is mainly executed by the RL

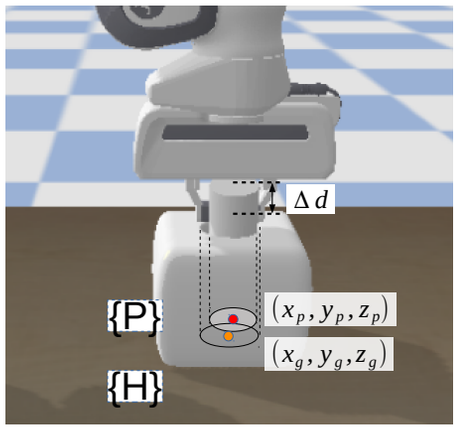
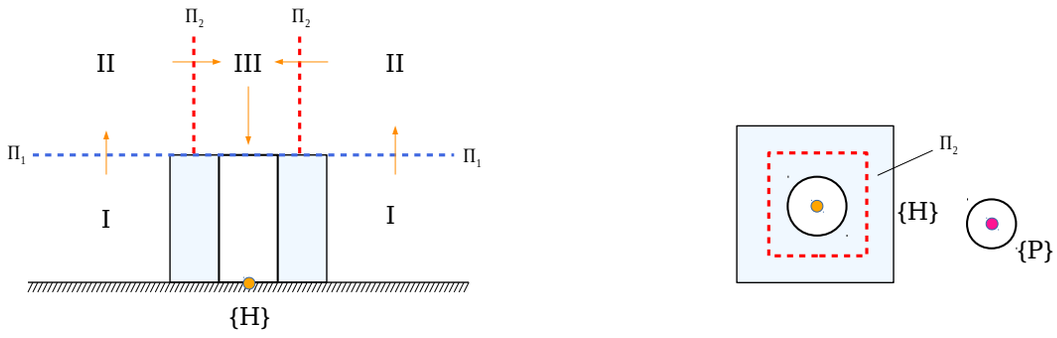


Figure 4.5: The indication of the process that the frame $\{P\}$ getting closer to $\{H\}$, where the red dot denotes the frame $\{P\}$ and the orange dot denotes the frame $\{H\}$ as well as the goal position (x_g, y_g, z_g) in the world frame. In the actual calculation, we use the (x_p, y_p, z_p) that is depicted in the frame $\{H\}$.

policy. The nominal control policy is applied under three conditions, which corresponds to three regions where the peg currently locates, the three relevant regions are denoted as I, II, III that divided into three parts by two infinite plane Π_1 and Π_2:

- I: move the peg above the plane that is a bit higher than the height of the hole opening.
- II: move the peg in horizontal direction so that the peg appears around above the hole opening.
- III: move downwards continually with a constant slight displacement.

To have perspective of the nominal policy, we have the simple schematic diagram that makes this policy clearer in figure 4.6. The process that brings peg from the initial position into the region III is mainly achieved by the nominal controller, as the distance between the $\{P\}$ and $\{H\}$ getting closer, the weights of the nominal control decay correspondingly, in our current work, this decay factor λ is not linearly changing, but as constant value with the amount depending on the region, we can see clearly in figure 4.7. When the frame $\{P\}$ falls into region III, the nominal controller keeps moving the peg down to encourage more contacts. As the peg starts inserting, the decay factor is equal to the distance between frames $\{P\}$ and $\{H\}$, which means the closer from $\{P\}$ to $\{H\}$, the more that RL-term dominant in the residual policy. Further explanation that combining



(a) (b)

Figure 4.6: A brief diagram of the nominal control scheme, the figure 4.6a indicates the environment with a cross-section of the front view, while the figure 4.6b is the environment view from above. The orange dot denotes the $\{H\}$ frame that attaches to the bottom of the cuboid and the red dot simply denotes $\{P\}$ frame. The blue dotted line denotes the border plane Π_1 between region I and II, the red dotted line denotes the sum of four sub-planes that above the cuboid and extend to the infinite point.

nominal controller and RL controller as well as the residual policy and impedance control law is illustrated in the 4.1.4 section.

4.1.4 Combination of Impedance Control and Residual Policy Learning

Nowadays there is a growing number of literature Johannink et al. [17], Kulkarni et al. [19], Silver et al. [33], Staessens et al. [35] that refer to residual policy, which provides a wide range of ideas and design details. As mentioned in section 3.3.3, the residual policy(RP) can be trained using the gradient-based method, even the initial policy is non-differentiable. The purpose of applying residual policy is to enhance the data-efficiency of learning and take advantages of both policies. To maintain flexibility as well as the robustness of insertion task, we assume that there are no accurate sensors but only the noised and biased input information, like some set points with rough location, the nominal controller is required to accomplish the most of the displacements. The rest part that result from system uncertainties is expected to be solved by RL-policy. In addition, to better decouple the roles of the two policies in RP, we use a decay constant that scale

the policy output that depends on the distance between the current position and the goal position, overall, as the peg gets closer to the target, RL will play a bigger role, and vice versa. The output of both policies and their combined part is expressed in action space, which is the displacement formulated in the Cartesian coordinates of translation and rotation. Due to the previously mentioned compliance characteristics that needed for interactive task, the combination of two policies, i.e. the total actions, are fed into the impedance controller for the sake of compliant manipulation.

Figure 4.7 shows the combination of RP controller and impedance controller, the RP

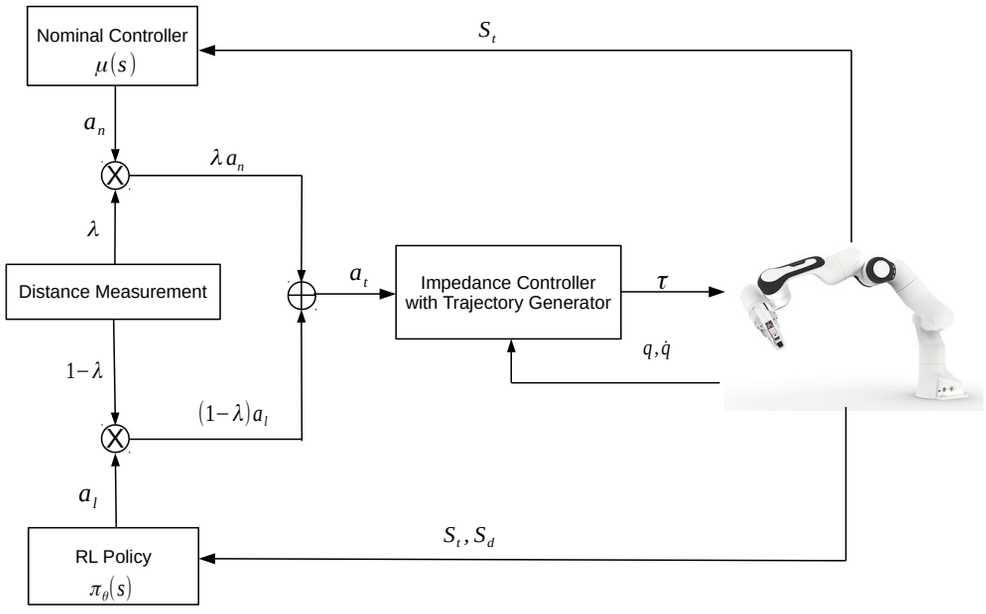


Figure 4.7: Overall control scheme that combined residual policy with the Cartesian impedance control.

controller composed of nominal policy and RL policy that both parts scaled by the distance factor λ that depend on the distance between peg position to the target position. The impedance controller receives the total desired displacement a_t and outputs the corresponding torque τ in joint space, same as in section 3.2 introduced. At the same time, the impedance controller receives the joint information from the robot, such as q and dq. The nominal controller uses the current states of peg S_t that are transferred from the robot to compute the difference between the peg to the goal. The RL controller receives not only

the current states of the peg but also the desired position S_d, which denotes the position that the peg is expected to reach. For the output from learned policy a_l and output from nominal controller a_n, consider the notation in section 4.1.3, the combination out of both policy is shown as:

$$a_t = \lambda a_n + (1 - \lambda)a_l,\ \lambda \in (0, 1). \tag{4.2}$$

Since the a_t is the Cartesian displacement from the current states to the desired states, the part of angle difference of a_t is firstly calculated in quaternion space, the minimum angle distance is transformed through rotation matrix into axis error, in the end, impedance controller is using this angle error as well as the translation error for controlling in "mass-spring-damper" wise. Since the mass-spring-damper system can be regarded as controlled by a PD-controller, although the robot is compliant to the external force, a large distance deviation still causes sudden and instantaneous movements and potential destruction. To keep the movement of the end-effector stable and converge into a reasonable static error, the Stiffness matrix described in equation 3.5 is tuned and a trajectory generator is employed to generate a trajectory that composed of multiple sequential sub-displacement, the start point and the endpoint of the trajectory are exactly the current position and the desired final position of the trajectory. For each sub-displacement, the control sub-step must be considered and must be limited to a value that is small enough to ensure the static error. The minimal executing time for each control step is also considered and tested. In the simulation of the experiment, the control frequency is set up to 1kHz, the setting time of each sub-displacement is tested and shown in chapter 5.

4.2 Learning the Residual Policy with TD3

In this section, we discuss the implementation of TD3(Twin Delayed Deep Deterministic Policy Gradient) in our experiment. TD3 is an off-policy algorithm based on the actor-critic method and derived from DDPG, where DDPG use actor and critic network to avoid overestimating. TD3 algorithm extends DDPG that uses double critic network to evaluate action value at the same time, and update the network with the smaller action value, in this way, the overestimation of the action value can be alleviated. In addition, the target network in TD3 does not update immediately but delayed updates after a few steps. As a result of this temporal difference update, the value function is built according to an estimation of a subsequent state (Fujimoto et al. [14]), so that the accumulating estimation error can be reduced. The elements of RL in our experiment are defined as follows:

- States. We assume that the robot does not have any force sensors to maintain a force control loop. To perceive the information from the environment that reveals where the obstacles are and where is free to pass, we use deviation between the desired posture of the peg and the actual posture of the peg after executing a specific action, that is position error at time t: $P_t = (\Delta x_t, \Delta y_t, \Delta z_t)$ and quaternion error $Q_t = (\Delta w_t, \Delta x_t, \Delta y_t, \Delta z_t)$. In this case, each learning step is not equal to the simulation step, but a sub displacement that made of multiple control steps. Moreover, the states contains the position of peg in the goal frame $\{H\}$: $P_g = (x_p, y_p, z_p)$. In total, we have the states as (P_t, Q_t, P_g).
- Rewards. Since the learning step is composed of a great number of simulation steps, let the negative position distance as the reward after each learning step: $r_t = -||P_g||$, if the peg is slipped away from the gripper, the episode not terminates and get a reward -2. If early terminate the episode when peg slips way, the agent may keep do weird movements to shorten the executing steps.
- Actions. The actions are defined as a short displacement in 6-DOF Cartesian space: $a_t = (\Delta x, \Delta y, \Delta z, \Delta\varphi, \Delta\theta, \Delta\psi)$, the last three elements are the rotation displacement of roll, pitch, and yaw, we also clip these angle error into values with a small amount($\ll 20°$).

We believe that both RL policy and impedance control have a necessary role for the mating of peg and hole. The whole RL only output a clipped displacement, because the allowance of the hole opening size and peg diameter should $< 1cm$), hence, it is helpful for robot to do local adjustment(only the surroundings of the hole opening), and impedance control is necessary in the other side that provides robot compliant behavior that also contribute to the jamming problem while experimenting.

5 Experiments

5.1 Data Collection and Preprocessing

The data is collected by the Azure Kinect camera and the Opti-Track system. The camera is set up around 40cm above the table where experiments are made. There are four markers attached to the top of each object so that the position and orientation can be captured. When detecting the hand landmarks by using *Mediapipe* framework, it is a certain probability that the landmarks are misidentified or overlapped to each other while running, the misidentified landmarks lead to faulty data set and wrong classification results. To improve the quality of the data set, human movements in the video should not move too fast and all fingers need to stretch and be exposed to the camera so that the algorithm built in the *Mediapipe* can distinct each finger and the landmarks correctly, although only a few fingers are used(In our experiment, the landmarks 0∼8 means only the thumb and index finger are detected to grasp objects, refer to figure 4.2). The figure 5.1 shows the comparison between the originally captured image and the image overlapped with landmarks.

There are in total 93 trajectories recorded for three primitive movements, pick-and-place, peg-in-hole, and screw, thus there are for each movement 31 samples respectively. The process starts with the hand moving towards the object, grasping it, moving to the target, and doing whether insertion or screwing. The pick-and-place task is terminated when the object is stably placed at the table. To not lose generality, the trajectory starts from the random place and random height(stack on another irrelevant object) with random direction on the table, the same case to where the object ends up with.

As mentioned in section 4.1 the data to be classified is presented as the trajectories of the human hands and the grasped object. First of all, the videos and trajectories are recorded

(a) (b)

Figure 5.1: Example of the insertion task. Figure 5.1a is the original picture and the figure and 5.1b is the recognition of the hand by adding landmarks on the figure 5.1a.

at the same time. However, the trajectories have more dense data in the entire process than the image sequences because they are not recorded with the same frequency. To align the data from them both, the transformation message of the object is only sampled when the image message in the time stamp also exists. As in section 4.1.1 formulated that each state is a trajectory that represents in time series, therefore, the time length of all trajectories can not be automatically the same. As for classification with DTW, the algorithm doesn't restrict to having the same length of data, while the SVM and TSF algorithms need each data sample to keep the same size. We have in that way two sets of data prepared for different classification algorithms. That is, for DTW we keep the trajectories as what it is and for the SVM and TSF algorithm, the trajectories are clipped to be the same length as 40 timestamps from the end time point backward to the beginning. Then the data is processed by standard scaling which performs better than the minimax scaling because the standardization centered the value around the mean so it is less affected by outliers.

5.2 Dynamic Settings for Robot System

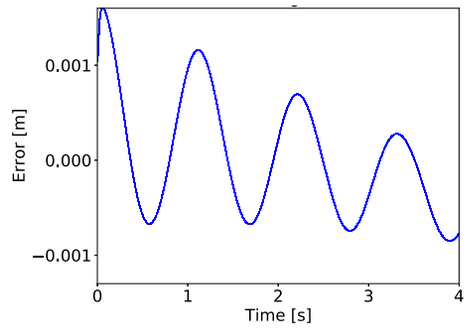
While doing manipulation tasks, it is usually important for the robot to have good dynamic characteristics, such as small overshooting, fast response speed, few steady-state errors, etc. However, there is another property we need from the robot manipulator in this work, which is compliant behavior. The compliant behavior requires the robot to act as the external force tends to, so the compliance is kind opposite to the requirements of

high dynamic(fast response) that demands huge gain and eliminate error or external interference as soon as possible. In addition, our set-point based impedance control is behaving like a mass-spring-damper system that does not converge immediately if external force suddenly withdraws. These trade-offs motivate us to do experiments to tune the parameter that makes control more stable and efficient.

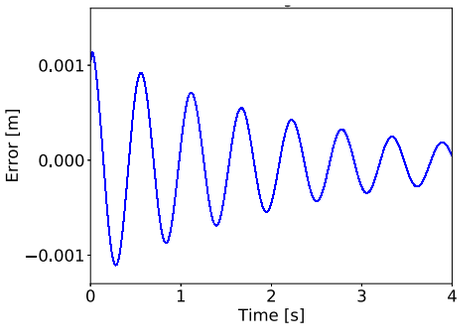
In the experiment, the dynamic characteristics of the Panda robot have been finely tuned in the simulation. We set up the parameters and limits in simulation so that the output torques of the robot are clipped in a reasonable range, which is also beneficial to sim2real in the future. In the simulation environment, the dynamics of objects are adjusted in the first place. Since the work is about the insertion task with a 1cm tolerance, we set the peg and hole both have high friction coefficients to prevent peg from sliding into the hole but encourage RL policy to explore actions that adjust the insertion orientations and translations.

Since the impedance control we used is the same as controlling at a virtual equilibrium point, large displacement causes for sure huge torque output that we do not anticipate. Hence, we set a maximum sub-displacement that is not allowed to be violated in the specific setting time of control. Although the robot does the manipulation task in a contact-rich scenario, we need to define the appropriate sub-displacement and corresponding setting time first in the case without any contact happens. Refer to Proposition 3.4 of [25] that for the twice differentiable desired trajectory $\mathbf{x}_d(t)$, the well-known closed-loop PD controller in configuration space is uniformly globally asymptotically stable can be ensured, if the cartesian coordinates are valid globally and their stiffness, damping matrices are symmetric positive definite, and there are no external forces applied. When the contact exists, the stability can be ensured but not asymptotic stability.

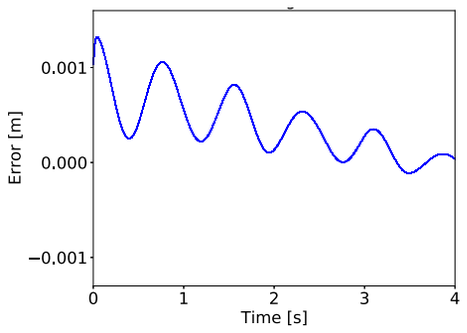
As defined before, the tolerance between peg and hole is 1cm, so the translational sub-displacements should require $|\Delta d| \ll$ 1cm. The length of the generated translational displacement for each learning step is an integer multiple of the length of 1mm. Let the sub displacement be 1mm, different gain and sub-steps are tested for sub displacement, and the results shown in figure 5.2, show the control for a 1mm displacement with multiple segments and execute each segment sequentially. It is quite clear from figure 5.2 that the output property is closely dependent on the gain K_p and the number of steps. Figure 5.2a, 5.2c, 5.2e represent the set-point control with various gains and a fixed sub-step. It is easy to determine that lower gain has less oscillation frequency



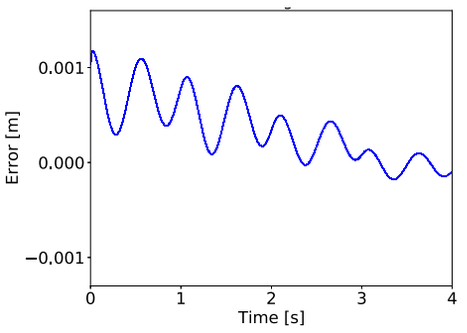
(a) $K_p = 1000, \quad step = 10$



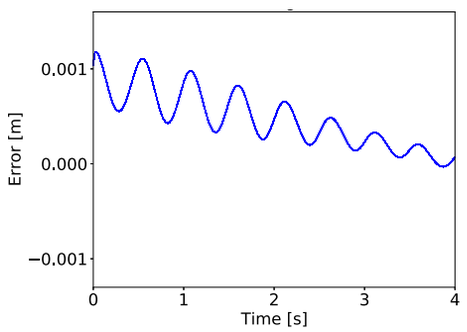
(b) $K_p = 3900, \quad step = 1$



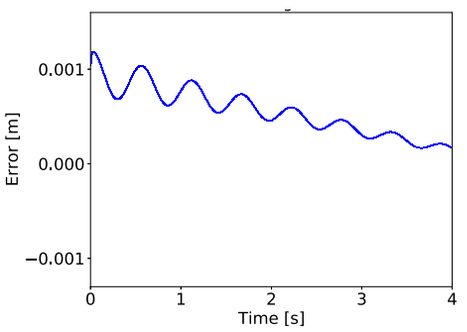
(c) $K_p = 2000, \quad step = 10$



(d) $K_p = 3900, \quad step = 5$



(e) $K_p = 3900, \quad step = 10$

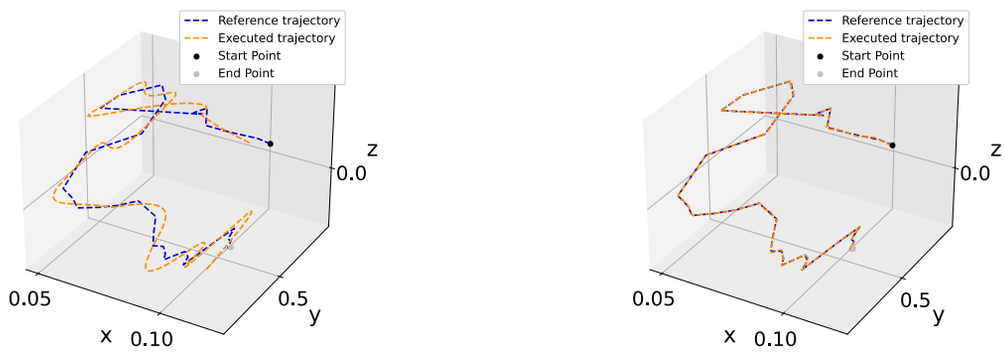


(f) $K_p = 3900, \quad step = 50$

Figure 5.2: Output features of maximum displacements in a toy experiment for trajectory generation. Although the stability of IC control is ensured, less oscillation and faster response is still the important element. Figure 5.2a, 5.2c,5.2e shows 10 sub control steps by applying different gain within 1mm, while figure 5.2b, 5.2d,5.2f shows the case with same gain but different sub step within 1mm. Although the case 5.2f performs best of all, we select 5.2e as the optimal candidate by considering the physical limitation of Panda robot.

but takes longer to converge. Figure 5.2b, 5.2d, 5.2f illustrate the different outputs that sub-steps bring. A smaller sub-step distance reduces the overshoot of the output position. We finally adopted the case with $K_p = 3900$ and $step = 10$ because the case with the same gain but $step = 50$ has too small a sub-step that even breaks the real robot's repeatability, so we discard this option that is not appropriate for further sim2real problem.

There are two trade-offs here that have a direct impact on the results. One is the compliant behavior and the precision of the control, which are already discussed as three cases on the left of figure 5.2. The other is the number of sub-steps and the time required for control, which should be considered if the running time is required to be short. To make the previous conclusions more intuitive, another toy experiment is conducted by using the off-the-shelf trajectory from our data set.



(a) $K_p = 3900, \quad step = 1$

(b) $K_p = 3900, \quad step = 10$

Figure 5.3: The comparison of trajectories by using different sub-steps. The slight difference in the figure between 5.3a and 5.3b shows the tracking ability of the trajectory when using different sub-steps. In the case of 5.3a, robot needs complete 1mm with only 1 control steps, while 5.3b do 1mm within 10 steps.

We process the trajectory derived from the real world by scaling the dimension into the simulation environments. Then the robot manipulates and traces the trajectory. The results shown in the figure 5.3 illustrate that applying dense sub displacements indeed precisely tracks the trajectory. If there is no strict requirement for running time, the tracking requirement is fulfilled by using 1mm displacement more than 5 control steps. In our case, we adopt $K_p = 3900$ and $step = 10$.

5.3 Policy Learned by RL algorithm

The environment is established by ourselves in *Pybullet* and the policy is learned by using Mushroom (D'Eramo et al. [12]). The experiments we conducted of the residual policy learning that has a robust and stable impedance controller with trajectory generation in the Cartesian space. The policy is given by the combination of the nominal controller and the learned RL policy, like in section 4.1.3, chapter 4 illustrated. Both policies are weighted by a factor that is dependent on the location of the grasped object. This factor λ is set to 0.99 for grasped object in the region I and II. The region III is broad so that the peg is not directly fallen into the hole but interacts with the surface of the cuboid by means of contact. The states in RL reveal the current situation of the end-effector whether it gets stopped by the obstacles or stopped in which specific direction. In the region III, we set the λ depending on the distance from the frame $\{P\}$ to $\{H\}$, so that the RL weights more as the peg closer to the target.

The actor-critic based RL algorithms are implemented to not only avoid overestimating but also to have good converging properties. To let the nominal policy work at the beginning without having the RL policy get involved, we initialize the weights of the last layer in the actor-network to zero without gradient tracing.

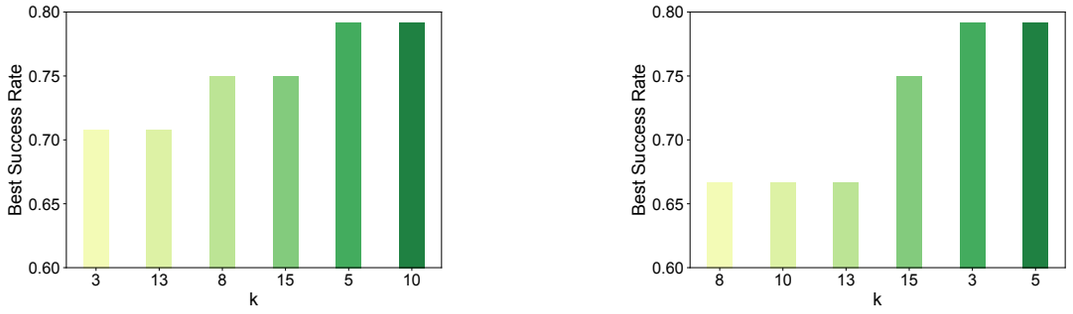
The best hyperparameters are selected from the results obtained from the multiple learning trials. We initially selected 180 features for the hidden layer, the output layer is to scale the values into -1 to 1 by using the $tanh$ function, the output from RL policy a_l is scaled the again into our requirements of the dynamic. In each learning step, the output a_l is scaled for translational displacement $\leq 0.01m$ and for rotational displacement $\leq \pi/50$ rad, since the rotational error while controlling is quite small, we didn't explicit illustrated in the part of trajectory generation. After several attempts, we set the features to be 40 because of the overfitting problem. Since the immediate reward is the negative distance from the frame $\{P\}$ to $\{H\}$, the agent tends to make the way with minimum steps to have a higher total reward. Originally, the episode is terminated if insertion is complete or the grasped object is slipped away from the robot gripper. But we found that the agent keeps behaving weirdly so that the peg frequently slipped away and get a higher total reward. Thus, the experiment terminates only when the goal is reached, and a very low reward is used when the peg is out of the gripper. The hyperparameter like the horizon is set to around 60 that slightly more than the steps of a perfect insertion, the batch size is originally set to 128.

6 Results

6.1 Classification

As mentioned in the last chapter that multiple algorithms are been employed to distinct primitive movements. The algorithm SVM is due to its poor results that already mentioned in chapter 3, so there are no further extensions here, instead, we only discuss the results that DTW+kNN and TSF outcome. There are balanced data sets been used for each primitive movement, that is 31 samples for each movement thus in total 93 samples. Therein are two kinds of post-processed data sets for DTW applied and compared, which is about whether the data is cropped into the same size or not, because the DTW-based method is not very dependent on the same data size. Since k is the value of the nearest neighbor that can be considered as a hyperparameter. We use hold-out cross-validation for all samples and get the best results for each k, the results are shown in figure 6.1. The test set is made of 24 samples out of all 93 samples, in that way 8 samples for each movement. As for the DTW+kNN algorithm, since k-NN is a well known lazy learner because it doesn't learn a discriminative function from the training data, i.e. a model with weights parameters, it calculates the distance from the training data directly.From the results, we can see that the best attempt occurs with $k = 5$, which is typical for kNN based method. In general, too small k leads to a large estimation error and the prediction result is sensitive to the nearby points. A larger k can reduce the estimation error, but the training instances far from the test instance will also play a role in the prediction. That is the reason we take a relatively small k and the cross-validation method helps select the optimal k value.

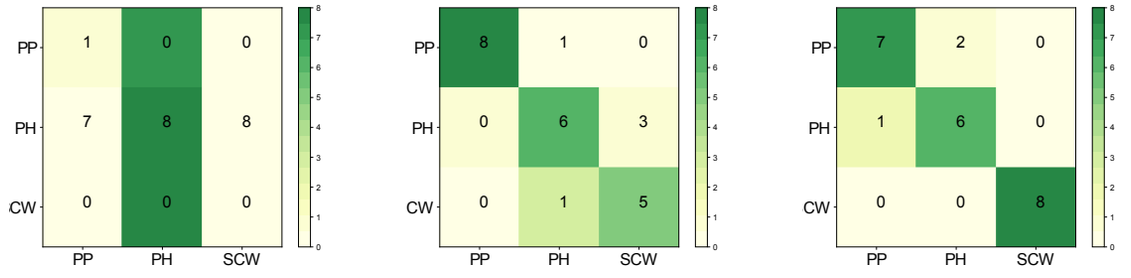
From the best results of all algorithms, we have the confusion matrix that presents in figure 6.2. It is quite obvious that the TSF has the highest accuracy rate and best positive prediction performance among all methods. The prediction performance is various by the type of class, and TSF outperforms in this field in all classes. Although DTW also has a considerable accurate rate, the results are highly dependent on the way that the training



(a) Original Data Set (b) Cropped Data Set

Figure 6.1: The best success rate from all test samples with different k values. Figure 6.1a uses the data set that trajectories have different time lengths and the figure 6.1b is the same data set as TSF, where all the time series are cropped into the same length.

set is selected and on the appropriate hyperparameter. The trade-off about selecting k is also tricky, larger k can improve the robustness of the algorithm, but the method itself is really time and compute costing compared to the decision-tree based method. The detailed evaluation can be found in the chart 6.1, which includes results that use



(a) SVM (b) DTW+kNN (c) TSF

Figure 6.2: Confusion matrix of best results from three algorithms. The abbreviation "PP", "PH" and "SCW" represent the primitive movement pick-and-place, peg-in-hole and screw respectively. The algorithms here are displayed from left to right: SVM, DTW+kNN, and TSF respectively.

fewer hand features and also the results without information of the grasped object. This chart records the performance of all used methods and all applied training features. The accuracy row illustrates directly that the TSF with the entire features that include not

Evaluation					
	TSF	TSF Less Features	TSF Without Object Info	DTW+kNN	SVM
Accuracy	0.88	0.83	0.75	0.79	0.38
Precision(0)	0.78	0.75	0.67	0.70	1.00
Precision(1)	0.86	0.75	0.75	0.75	0.35
Precision(2)	1.00	1.00	0.88	1.00	0.00
Recall(0)	0.88	0.75	1.00	0.88	0.12
Recall(1)	0.75	0.75	0.38	0.75	1.00
Recall(2)	1.00	1.00	0.88	0.75	0.00
f1-score(0)	0.82	0.75	0.80	0.78	0.22
f1-score(1)	0.80	0.75	0.50	0.75	0.52
f1-score(2)	1.00	1.00	0.88	0.86	0.00

Table 6.1: The evaluation of all use cases. The number behind precision, recall, and f1-score is the rate that gives to the ability to classify individual tasks, i.e. "0" for "pick-and-place" task, "1" for "peg-in-hole" task, and "2" for "screw" task.

only 9 landmarks of hands but also the grasped object outperform other counterparts. The precision rows show that TSF most correctly predicted the positive observations of all positive observations. Similarly, it can be seen from the recall columns and the f1-score columns that TSF with complete features has the highest classification discrimination ability. Compare to the second column under "TSF Less Features", it is interpretable that this case contains the most complete information that necessary for classification, the missing information from fewer features is the object traced by the Opti-Track that contains the orientation information in the quaternion space. The landmarks data from *Mediapipe* can not supply orientation information, and Opti-Track for now is a significant supplement. To some extent, the results that only come from an object are even more important than the hand landmarks since the second column has a higher accuracy rate than the third column. As discussed before, regardless of the running consumption, the DTW-based method still has a not bad performance, which can be considered as a baseline classifier for future work, and SVM should by no means be used to tackle time series classification problems.

6.2 Learning

In the residual policy learning, we would like to verify the plausibility and feasibility of this new environment, thus we tended to use the off-the-shelf RL algorithm to learn a residual policy. We opted TD3 as our RL algorithm, and initialized the experiments with different hyperparameters, and adopt the values that refer to best trials to train the model, wherein the batch size is 128, the horizon is 60 steps(nominal policy takes around 37 steps), the feature size is 40.

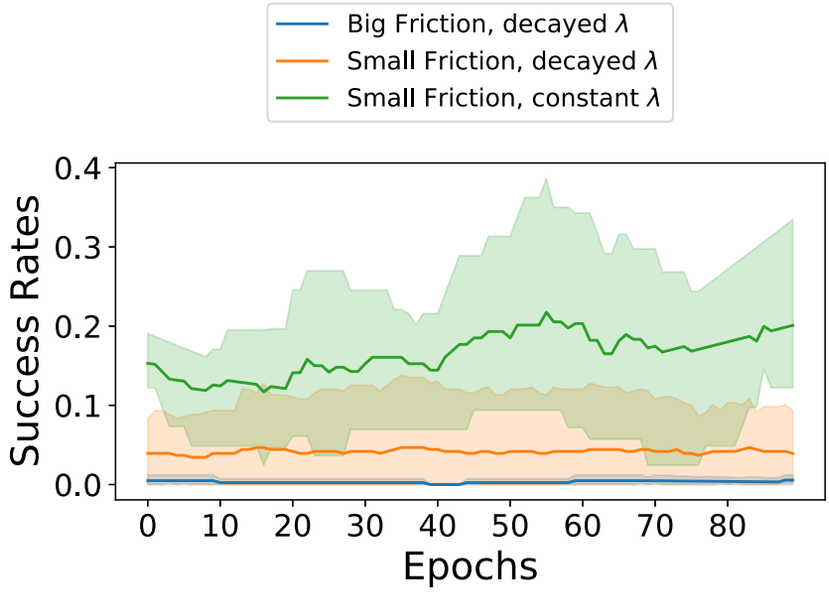


Figure 6.3: The comparison of success rate among three cases for each over 10 seeds: big friction coefficient with constant λ, big friction coefficient with decayed λ and small friction coefficient with decayed λ .

During the experiment we found that the success insertion highly depends on the relations between nominal policy and RL policy, in other words, for total policy output $a_t = \lambda a_n + (1 - \lambda)a_l$, the selection of λ is essential to the results, where a_n, a_l stands for nominal and learned policy respectively. We also set the high friction coefficient to encourage the policy to do more local manipulation, but high friction also leads to the jamming problem. Thus, we verify our assumptions by comparing the performances under

these three scenarios, the results can be found in figure 6.3.

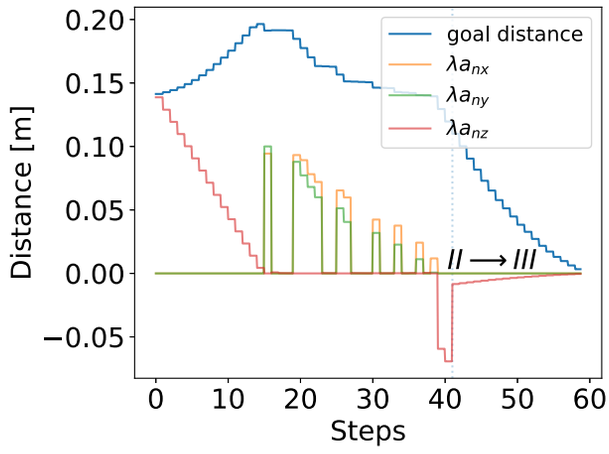
It is easy to draw from the figure that the learning with the decayed λ has the worst outcoming, in this case, we have $\lambda = ||P_g|| \leq 0.13$, where $||P_g||$ is the goal distance in the hole frame, and 0.13cm is exactly the height of the hole opening. Since the nominal policy is less weighted, the total policy is then dominated by the RL policy. When the peg goes deeper and closer to the bottom, the RL policy gradually loses the guidance from the nominal policy. The most fatal factor is, in this low allowance contact-rich scenario, the decayed λ starts drastically varying and hardly converge to a value, the same case happens at the output of the $(1 - \lambda)a_l$, this is data-inefficient and there is no big difference with pure RL.

The case of the small friction applied is also explainable, the interaction between the peg and the hole can be uncertainly affected by the relative sliding of the hole, on one hand, the peg is compliant to slide away instead of jamming somewhere, on the other hand, this uncertainty also causes the huge standard variation of the reward. But in total, we still prefer the constant λ with small friction as our optimal policy, because this option has the most success rate and leads to more possibilities to explore. In our final test, the best agents trained by constant λ with small friction are conducted in 20 insertion trials, and there were 6 successful insertions. Normally, it took 5 minutes to finish one episode, thus, due to the time limits, we didn't adopt more epochs for training, but more epochs bring different results because the nominal policy takes too many steps!

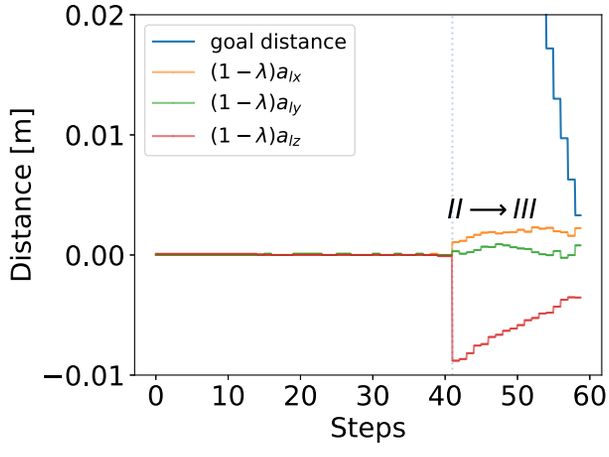
To illustrate the non-clipped output of both policies in the residual policy, we have an example of a success insertion shown in figure 6.4, wherein two subfigures shown the same process of insertion that in one episode, both figures are differently scaled because a_n and a_l work in different dimensions. Figure 6.4a is the output of the nominal policy, we can see that the policy worked at the beginning till step 15 to move the peg from region I into II(refer to system overview in chapter 4), so only movement in the vertical direction a_{nz} worked. Then the peg moved horizontally to the surrounding of the hole opening. From step 39 the a_n output downwards displacement to force the peg contact with the surface of the cuboid, and then the influence is decayed when peg get closer to the goal, the sign of II$\rightarrow$ III means the time that the λ switches and RL policy is in dominant. The case of 6.4b shows the RL policy is more weighted when the peg gets closer to the goal. Actually, this example is a very rare success of all trails, because as mentioned, when the factor λ decayed, the output of a_l is hardly to converge. Note that, the actual total output fed to

impedance controller is not equal to $a_t = \lambda a_n + (1 - \lambda) a_l$, but the clipped action of this a_t, because we must ensure the output values are valid and reality-based. The example of the success episode and the example of the failure episode are shown in figure 6.5, wherein the figure 6.5a shows an often encountered case that the peg slipped away from the gripper.

At the very end of this work, we realized that the methods we used still has a lot to improve, and the RL policy was not well learned, because the nominal policy took too many steps in the total episode, although the equation 3.8 holds for residual policy, it still needs a lot of computation and training to converge. One possible optimization for the future is to reduce the steps done by the nominal policy.

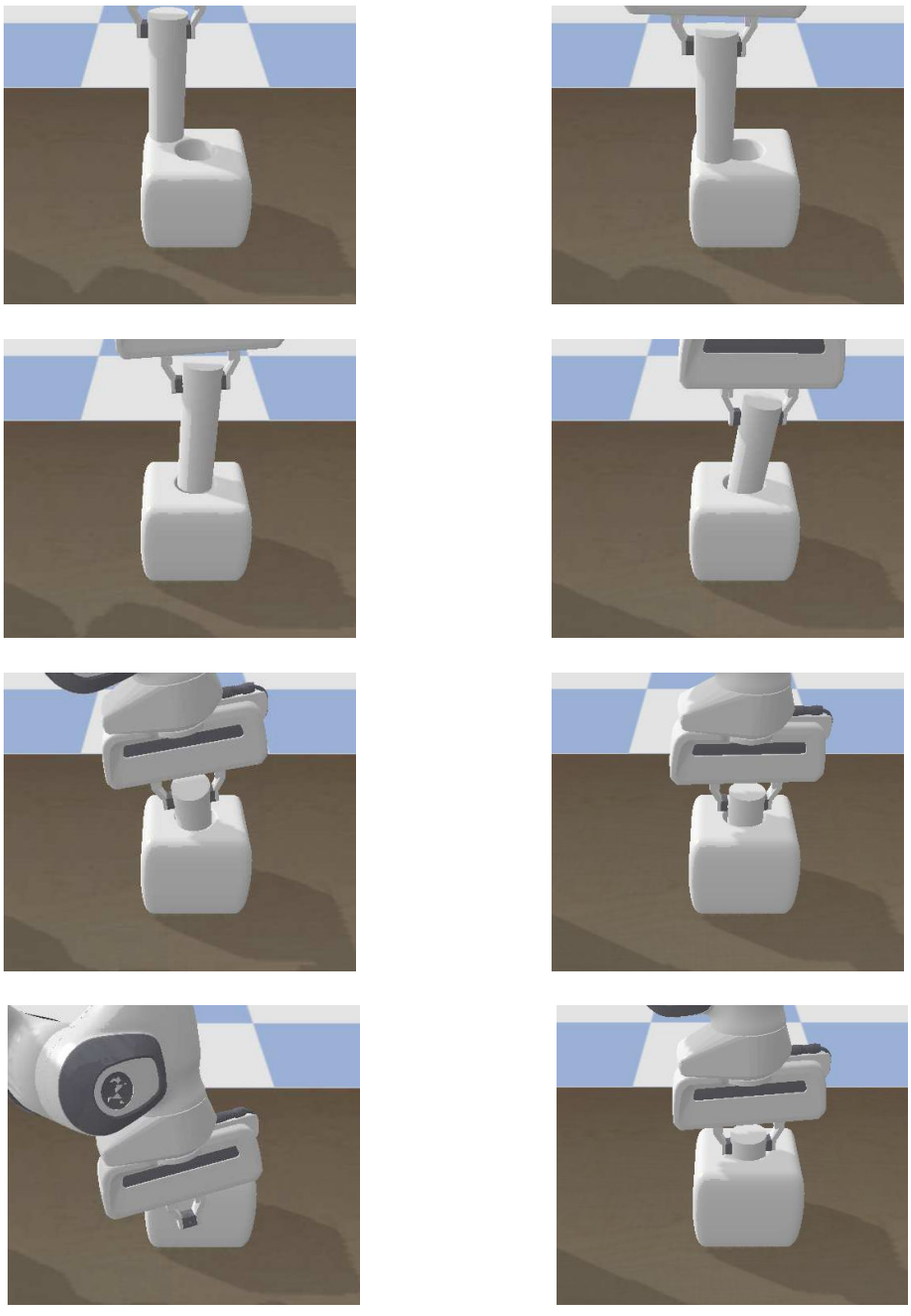


(a)



(b)

Figure 6.4: Example of the residual policy by using decayed factor λ that equal to the goal distance. The dotted shaded line means the step that the peg goes from region II into III refer to chapter 4. Note that the values only represent the original output of nominal and RL policy that not be clipped.



(a) (b)

Figure 6.5: Comparison of a failure and a successful example from the best-trained agents. Both cases are sampled at the same step. The left figures 6.5a show the case that the peg slipped away from the gripper.

7 Conclusion

Our first contribution in this work is, we built up a pipeline that automatically maps from a video to the robot manipulation task, and we can accomplish the primitive task if the category is peg-in-hole. The classification results meet our requirement, it is fast to have the outcomes and the accuracy rate is quite satisfied.

Our second contribution is the development of a stable and robust simulation environment for robot manipulation, where the robot contains position-based compliant behavior without any force sensing. Our work is reality-oriented so it aimed very closely at dynamic tuning, so that is more likely to transplant the policy on the real robot in the future. But there are more difficulties than expected when tuning parameters for impedance control of the system because there are several trade-offs in the field of Cartesian space impedance control. For example, the trade-off between stability(convergence) and steady manipulation(no crash). The trajectory generation solves the problem by steadily control in sub displacement, which in addition needs fine-tuning of the stiffness matrix and damping matrix for our specific task. The other trade-off is the steady, compliant movement with the huge running time. The long-term running limited our attempts for more algorithms and hyperparameter tuning.

As you can see, the peg accidentally released from the gripper often happens. A possible solution in the future is to fix the peg as the last link of the robot, which is easier to train and avoid releasing as well. Due to the software limits, it is currently difficult to simulate the bolt-nut pairs for the screw task in the simulation, which is a small regret of our work. And the screw task needs a more complicated nominal policy for the robot because the end-effector can not continually rotate due to joint limitations. Instead, in the future, we can expand the object from cylinder to polygons and other shapes. And the task becomes to do a generalized insertion task without considering the shape of the hole.

In fact, from chapter 6 we know that when a peg is partially inserted into the hole, it is better to use a fixed weight factor as a dynamic one, the combination of both policies is always worth exploring and investigating. One another worthy discussing topic is, to better generalize the insertion problem, insert not only the polygon shape peg as mentioned but also with a random insertion angle and let the RL policy adjust the insertion posture.

Using memory-based approaches to learn past observations is seen to be promising in some similar works. For example, the recurrent TD3 has now been used and has significant improvement over approaches without RNN.

Acknowledgments

First of all, I would like to express my sincere gratitude to my advisors João Carvalho and Suman Pal, who helped me the most throughout this work. They gave me great suggestions as well as insightful feedback on the thesis, which taught me a lot. They are the most responsible, resourceful people, I've been so lucky to work with them.

Second, I would like to appreciate Prof. Dr. Jan Peters who gave me the chance to do master thesis at the IAS group, and Prof. Dr.-Ing. Marius Pesavento agreed to be my Co-supervisor at FB18.

My appreciation also extends to PhD student Puze Liu from IAS Group, as well as my colleagues Zhiyuan Hu and Haoyi Yang, they gave me some good advice and encouraged me a lot.

And I would like to thank Min Zhang and Lifu Zhou, thank you for your companionship, support, and encouragement all along.

Last but not least, I want to thank me, thank me for believing in me, thank me for never quitting.

Bibliography

[1] Amaia Abanda, Usue Mori, and Jose A. Lozano. A review on distance based time series classification, 2018.

[2] A. Bagnall, M. Flynn, J. Large, J. Line, A. Bostrom, and G. Cawley. Is rotation forest the best classifier for problems with continuous features?, 2020.

[3] Anthony Bagnall, Aaron Bostrom, James Large, and Jason Lines. The great time series classification bake off: An experimental evaluation of recently proposed algorithms. extended version. 02 2016.

[4] Donald J. Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD Workshop*, 1994.

[5] Aude Billard and Danica Kragic. Trends and challenges in robot manipulation. *Science*, 364, 2019.

[6] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[7] Jeannette Bohg, Karol Hausman, Bharath Sankaran, Oliver Brock, Danica Kragic, Stefan Schaal, and Gaurav S. Sukhatme. Interactive perception: Leveraging action in perception and perception in action. *IEEE Transactions on Robotics*, 33(6):1273–1291, Dec 2017. ISSN 1941-0468. doi: 10.1109/tro.2017.2721939. URL `http://dx.doi.org/10.1109/TRO.2017.2721939`.

[8] Benjamin Burchfiel and George Konidaris. Hybrid bayesian eigenobjects: Combining linear subspace and deep network methods for 3d robot vision. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2018. doi: 10.1109/iros.2018.8593795. URL `http://dx.doi.org/10.1109/IROS.2018.8593795`.

[9] Andrea Calanca, Riccardo Muradore, and Paolo Fiorini. A review of algorithms for compliant control of stiff and fixed-compliance robots. *IEEE/ASME Transactions on Mechatronics*, 21:613–624, 2016.

[10] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. URL `http://www.blender.org`.

[11] Houtao Deng, George C. Runger, Eugene Tuv, and Vladimir Martyanov. A time series forest for classification and feature extraction. *CoRR*, abs/1302.2277, 2013. URL `http://arxiv.org/abs/1302.2277`.

[12] Carlo D'Eramo, Davide Tateo, Andrea Bonarini, Marcello Restelli, and Jan Peters. Mushroomrl: Simplifying reinforcement learning research, 2020.

[13] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. Querying and mining of time series data: Experimental comparison of representations and distance measures. *PVLDB*, 1:1542–1552, 08 2008.

[14] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018.

[15] Andrew Hynes, Elena Sapozhnikova, and Ivana Dusparic. Optimising pid control with residual policy reinforcement learning. 12 2020.

[16] Tadanobu Inoue, Giovanni De Magistris, Asim Munawar, Tsuyoshi Yokoya, and Ryuki Tachibana. Deep reinforcement learning for high precision assembly tasks, 2017.

[17] Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control, 2018.

[18] Young Loul Kim, Hee Chan Song, and Jae Bok Song. Hole detection algorithm for chamferless square peg-in-hole based on shape recognition using f/t sensor. *International Journal of Precision Engineering and Manufacturing*, 15(3):425–432, March 2014. ISSN 1229-8557. doi: 10.1007/s12541-014-0353-6. Funding Information: This work was supported by the MOTIE under the Industrial Foundation Technology Development Program supervised by the KEIT (No. 10038660) and by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2007-0056650). Copyright: Copyright 2021 Elsevier B.V., All rights reserved.

[19] Padmaja Kulkarni, Jens Kober, Robert Babuška, and Cosimo Della Santina. Learning assembly tasks in a few minutes by combining impedance control and residual recurrent reinforcement learning. *Advanced Intelligent Systems*, 2021. doi: 10.1002/aisy.202100095.

[20] Hyun-Seob Lee. Application of dynamic time warping algorithm for pattern similarity of gait. *Journal of Exercise Rehabilitation*, 15:526 – 530, 2019.

[21] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, Wan-Teh Chang, Wei Hua, Manfred Georg, and Matthias Grundmann. Mediapipe: A framework for building perception pipelines, 2019.

[22] G. Manacher and D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences.

[23] Kevin P. Murphy. *Machine learning : a probabilistic perspective*. MIT Press, Cambridge, Mass. [u.a.], 2013. ISBN 9780262018029 0262018020.

[24] Masahide Oikawa, Kyo Kutsuzawa, Sho Sakaino, and Toshiaki Tsuji. Assembly robots with optimized control stiffness through reinforcement learning, 2020.

[25] Christian Ott. *Cartesian Impedance Control: The Rigid Body Case*, pages 29–44. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-69255-3. doi: 10.1007/978-3-540-69255-3_3. URL https://doi.org/10.1007/978-3-540-69255-3_3.

[26] Hyeonjun Park, Jaeheung Park, Dong-Hyuk Lee, Jae-Han Park, Moon-Hong Baeg, and Ji-Hun Bae. Compliance-based robotic peg-in-hole assembly strategy without force feedback. *IEEE Transactions on Industrial Electronics*, 64(8):6299–6309, 2017. doi: 10.1109/TIE.2017.2682002.

[27] Jessica Peters. Implementation of compliant motion control schemes on the stewart platform hybrid position/force control and position-based impedance control. 2019.

[28] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, M Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. volume 2012, 08 2012. doi: 10.1145/2339530.2339576.

[29] Carlos Saldarriaga, Nilanjan Chakraborty, and Imin Kao. *Design of Damping Matrices for Cartesian Impedance Control of Robotic Manipulators*, pages 665–674. 01 2020. ISBN 978-3-030-33949-4. doi: 10.1007/978-3-030-33950-0_57.

[30] Carlos Saldarriaga, Nilanjan Chakraborty, and Imin Kao. Damping selection for cartesian impedance control with dynamic response modulation. *IEEE Transactions on Robotics*, pages 1–10, 2021. doi: 10.1109/TRO.2021.3116855.

[31] Matteo Saveriano, Yuchao Yin, Pietro Falco, and Dongheui Lee. Data-efficient control policy search using residual dynamics learning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4709–4715, 2017. doi: 10.1109/IROS.2017.8206343.

[32] Bruno Siciliano and Oussama Khatib. *Springer Handbook of Robotics*. Springer-Verlag, Berlin, Heidelberg, 2007. ISBN 354023957X.

[33] Tom Silver, Kelsey Allen, Josh Tenenbaum, and Leslie Kaelbling. Residual policy learning, 2019.

[34] Shuran Song, Andy Zeng, Johnny Lee, and Thomas Funkhouser. Grasping in the wild:learning 6dof closed-loop grasping from low-cost demonstrations, 2020.

[35] Tom Staessens, Tom Lefebvre, and Guillaume Crevecoeur. Adaptive control of a mechatronic system using constrained residual reinforcement learning, 2021.

[36] Chang Wei Tan, Christoph Bergmeir, Francois Petitjean, and Geoffrey I. Webb. Time series extrinsic regression, 2021.

[37] Jing Xu, Zhimin Hou, Zhi Liu, and Hong Qiao. Compare contact model-based control and contact model-free learning: A survey of robotic peg-in-hole assembly strategies, 2019.

[38] Lexiang Ye and Eamonn Keogh. Time series shapelets: A novel technique that allows accurate, interpretable and fast classification. *Data Min. Knowl. Discov.*, 22:149–182, 01 2011. doi: 10.1007/s10618-010-0179-5.