Institute of Parallel and Distributed Systems
Department of Machine Learning and Robotics

Master Thesis

Hierarchical Manipulation Learning From Demonstration

Hamid Reza Zaheri

Course of Study:	Master of Science Information Technology
Examiner:	Prof. Dr. Marc Toussaint
Supervisor:	Prof. Dr. Marc Toussaint
Commenced:	May 31, 2016
Completed:	Feb 2, 2017
CR-Classification:	I.2.6, I.2.9, I.2.10

To my parents. None of these would have been possible without their supports.

ACKNOWLEDGMENTS

Just before starting this project at the University of Stuttgart, I moved to London to start my entrepreneurship adventure. During which It was simply not possible to find any free time to spend on this work. If it wasn't because of my advisor, Marc Toussaint, and his supports, I wouldn't be able to finish this work.

I am grateful to all my friends and fellow researchers who inspired and helped me during this journey.

Abstract

Despite many efforts in the field of Robotics, human-robot interface still suffers from unnecessary complexities that in turn restricts the application of robots. A plausible solution is an intuitive interface to teach robots variety of tasks via demonstration. In this work, we attempt to design such a framework that allows the robot to learn multi step object manipulation tasks from only a handful of demonstration. In particular, we are interested in the problem of pick and placing.

To make the learning process tractable and allow the skills to be generalized, we break down the monolithic policies to hierarchical skills. In other words, Robot acquires different set of skills such as grasping different object and variety of trajectories from different learning sessions and is then able to combine those skills to achieve the objective of a novel task the might require a combination of the previously learned skills by the robot. To accomplish this, we define a notion of action types and use them as labels to train our segmentation model. The process of features extraction from the segmentation process will then be merged with the training process of our controller model where it helps the controller to more carefully learn the relevant features that are associated with the preconditions and postconditions of each segment.

To efficiently extract visual features from the cameras installed on the robot, we propose a novel method of training an encoder to transform RGB stereo images into their corresponding depth map, and while doing that, extracting the most important visual features that are relevant to object positions and their corresponding distance from the camera. This is implemented using convolutional neural networks with a bottleneck (feature vector) in the middle to reduce the dimensionality of the data. Next, the extracted visual features are combined with the kinematic features of the demonstrations i.e. pose of the robot’s end-effector as an input to our segmentation and controller modules.

Moreover, we introduce a novel methods of grasp prediction using a two-step prediction model. Our algorithm uses a coarse to fine approach to predict the position and orientation of the free grasping points on different object. Also grasp predictions are based on the user predefined position on each object, which is one of the practical requirements in robot grasping tasks in industrial applications.

Only the simulated scenes has been used in this work. This might raise the concern over the transferability of the learned skills to the real world applications. We address this issue by suggesting different ways to mitigate this problem, especially for the grasp prediction skill.

TABLE OF CONTENTS

	Page
ACKNOWLEDGMENTS	**ii**
ABSTRACT	**iii**
LIST OF TABLES	**v**
LIST OF FIGURES	**vi**
CHAPTER	
1. Introduction	**1**
1.1. Motivation	2
1.2. Contributions	2
1.3. Chapters outline	3
2. Background and Literature Review	**4**
2.1. Learning from Demonstration	4
2.2. Trajectory Segmentation and Controller design	5
2.3. Grasping	7
2.4. Convolutional Neural Networks	8
2.5. Recurrent Neural Networks	9
3. Simulation and Demonstrations	**10**
3.1. Simulated scene	10
3.2. Demonstrations and Objectives	11
3.2. Task demonstration	13
4. Grasp	**18**
4.1. Grasp point detection	18
4.2. Training data generation	21
4.3. GPN (grasp proposal network)	22
4.4. Transfer learning	24
5. Segmentation and trajectory generation	**25**
5.1. Segmentation	25
5.1.1. Statistical approach	25
5.1.2. Stereo to depth encoder	26
5.1.3. Actions types	29
5.2. Motion controller	32
5.3. Combined segmentation and trajectory generation	34
5.4. Model execution phase	36
6. Conclusion and Future work	**37**
7. Bibliography	**39**

LIST OF TABLES

Table	Page
3.1 summary of the recorded data during the task demonstrations (per time step)	17
5.1 summary of the architecture of our Stereo to depth encoder model	27

LIST OF FIGURES

Figure	Page
2.1 Process of learning by demonstration. From a set of demonstrations by the teacher a policy is derived which is then used to select robot actions based on the observation from the world state.	4
2.2 Standard hidden markov model (HMM) where at each state, only the the latent variables (observations) are accessible	6
2.3 multiple robots are attempting to grasp diverse set of objects to gather training data for the task of success prediction [6]	8
2.4 multi step approach of generating grasp proposals and ranking [20]	8
3.1 A screenshot of the V-REP simulation software used in this work [14]	10
3.2 Randomly generate pile of cups	11
3.3 Randomly generate pile of nuts	12
3.4 Designed scene for the demonstration of task 2	13
3.5 Overview of the demonstration procedure	14
3.6 Sample demonstration of the task 2, trajectory shows the evolution of the robot's end-effector Cartesian coordinates	15
3.7 Another sample demonstration of the task 2	15
3.8 Robot's end-effector pose values duration the task demonstration	16
3.9 Robot joint values duration the task demonstration	16
3.10 The simulation scene while the demonstrator program is running and the top-right corner of the image shows the stereo RGB camera images	17
4.1 User defined grasp point on the nut (left) and the cup(right)	18
4.2 Left image show the color-coded collision free grasp points and the right image is the visualization of the ground truth label on the corresponding depth image	19
4.3 visualization of the 2D grid to estimate the approximate location of the available grasps in the first layer of our prediction algorithm	19
4.4 Cropped areas of the original image with high probability of finding a free grasping point	20
4.5 Possible predicted location of the grasp point center and orientation of the grasp in the second layer of our model	20
4.6 Sample randomly generated piles	21
4.7 Architecture of the first layer of our predictor	22
4.8 Architecture of the second layer of our predictor	23
4.9 sample predicted grasp point using the GPN model	24
5.1 BP-AR-HMM segmentation of the joint value, including the gripper status	26
5.2 BP-AR-HMM segmentation of the pose value	26

5.3 Architecture of our Stereo to depth encoder 27

5.4 Sample depth image reconstruction from the stereo RGB images. The image on the left is the ground truth depth image and on the right is the reconstruction. Note that the network learned to associate visibility of part of the gripper to the high probability of an object existing between the fingers, hence a shadow of a nut or half a cup can be seen in the reconstruction 28

5.5 Another example of the stereo RGB to depth reconstruction 28

5.6 Gripper joint values from the recording of a sample demonstration 29

5.7 Running moving average over the gripper values to reduce the noise 30

5.8 Grasp action (left), Release action (middle), Non-gripper action (right) 30

5.9 Derivative of the gripper values correspond to the defined action types 31

5.10 RNN trained to predict the action types based on the features extracted from the stereo images and pose values of the end-effector 31

5.11 Prediction of the action type by RNN, Green line is the ground truth and the blue line represents the prediction 32

5.12 Architecture of the trajectory generator using a single LSTM cell 33

5.13 Comparison between the predicted and demonstrated joint values 33

5.14 Combination of segmentation and controller RNNs 34

5.15 Training the controller 35

5.16 Execution phase 36

Chapter 1
Introduction

Over the past decade, the problem of learning from demonstration [2] has become one of the main streamlines in the robotics research. As many have stated, once solved, it can revolutionize the relation between human and robot. It would help to bring out the robot from laboratories to our daily lives, allowing people without extensive robotics knowledge to interact with and program robots. And most importantly, it will help to automate many laboring tasks.

Every year we observe the decreasing price of the robotic hardware and evidence suggest that the software and control systems are the bottleneck on having robots in the human environment. Robots have been used in industry for many years, but for only specific tasks where the coordinates of the robot's end effector, joint configurations or the desired path have been precisely calculated and hardcoded on the robot’s controller to achieve the desired objective. This approach is not scalable to dynamic environments where the conditions which can affect the robot’s behavior are subjected to random changes.

We summarize the main problems with the robotic software in three major categories:

1. Robots should be able to cope with the randomness in the dynamic environment and still be able to follow their trajectory to achieve the predefined goals.
2. There needs to be a revolution in the way human interacts and programs robots. The Traditional way of spending hours of coding and testing even to achieve simple robot behaviors should be replaced by an efficient and more natural approach.
3. Robots program should be abstracted from the specific hardware. There needs to be a method to teach/program robots while the same program can efficiently be transferred to other robots and an underlying process can translate the abstracted information to the robot’s specific hardware.

Attacking the first problem requires gathering many observations from the human environment. If an adaptive behavior is to be achieved, patterns of the random events in the environment needs to be predictable and appropriate responses to those events should be practices and learned by the system. For the second problem, One good way would be to program robots via demonstration. This approach allows non-expert people to interact with the robots via natural movement and the system ideally would be able to interpret the objective and constraints of the task and generate its own program or sequence of steps that will lead to the desired behavior meant by the demonstrator. And a possible solution to the third problem would be to use the task space as the only source of information while training a robot and adaptation to individual hardware can be accomplished by the robots interfaces.

Learning from demonstration has been a popular research topic in the robotics community for many years. Different approaches have been proposed to solve variety of problems such as teaching helicopter complex maneuvers [3], pendulum swing up by robotic arm [32] etc. However many of those approaches disregard the importance of encoding the visual features in the learning process and simply treat the task as learning pure trajectory. Whereas in the real world applications tasks such as object manipulation. E.g. Pick and placing, are directly related to the environment status and visual features.

This leads to the question that is it possible to efficiently combine both the trajectory and visual features and use them to learn a robot controller that is capable of predicting the robot's configuration based on both environment and robot's status?

1.1. Motivation

We are concerned with the task of robot manipulation learning from the demonstration and in particular grasping problem. To narrow it down, we would like to implement a framework for the robot to learn multi-step tasks with the focus on the pick and placing. There's enormous need in the industry, and in particular, manufacturing for this application. Despite all the advances in the in the field of Robotics and AI, and efforts such as [4][5][6], There's is still no concrete solution to this problem with the accuracy and precision needed for the manufacturing sector.

Although we do not claim that the method offered in this work will solve this problem, however, it's an attempt to design a simple framework which can be expanded and improved incrementally over time to reach the accuracy required for the industrial applications.

1.2. Contributions

Our contribution in this work can be summarized as follows:

- A simplified framework to combine visual and trajectory features of the demonstrations and efficiently incorporate them to learn the task of segmentation and trajectory generation.
- A novel hierarchical method in grasping from homogeneous piles.
- A novel method for generating training data for the robot grasping task.
- Design and implementation of hierarchical manipulation learning where new grasping skills can be added to the system and robot's skills in manipulation can be expanded over time.
- A novel way to extract features from the images using convolutional neural networks (2.4) by converting stereo images to depth maps. This method allows the algorithm to encode the features of the scene that are more relevant to the object positions and their distance from the camera. And as we will see, these features can be used to generate robot commands that are affected by the status of the environment.

1.3. Chapters outline

Chapter 2: Background and literature review

Background on learning by demonstration, manipulation skills, Convolutional and Recurrent Neural Networks and their application in robot control. However, materials provided in this chapter are restricted to their relevance to this work.

Chapter 3: Simulation and demonstrations

In this chapter, we introduce our approach to demonstrate a multi-step pick and placing task to the robot. Next, we design a simulated scene and perform few demonstration and visualize the acquired data that later will be used to teach our algorithms the desired behaviors.

Chapter 4: Grasp

This chapter demonstrates our solution to solve the grasping problem with the focus on grasping from a pile of cluttered but homogeneous objects. We present our algorithm which attempts to predict collision free grasp points on an object via multi-step *grasp proposal network*.

Chapter 5: Segmentation and trajectory generation

This chapter introduces our approach to defining phases in the demonstration and train a controller for our simulated robot via the combination of visual and trajectory features in the task space. Throughout this chapter, we attempt to segment the demonstrated trajectory via defining the relevant “Action types” to the task of pick and placing. Moreover, we propose our controller design using recurrent neural networks to generate robot configurations based on kinematics and visual features of the input.

Chapter 6: Conclusion and future work

A brief discussion over the shortcomings of the proposed method, future plans to improve our framework and suggestions for possible directions in the research of learning manipulation skills from demonstration.

Chapter 2

Background and Literature Review

This chapter will provide an overview of the topics relevant to our work. First, we define the problem of learning by demonstration and provide some insights over different categories of problems relevant to this topic. Next, we introduce the problem of trajectory segmentation which will lead to our discussion over the task phase definition and a brief introduction to Markov processes. Next, we discuss different approaches in designing the robot controller and trajectory generation. Then we briefly discuss the advances in the application of convolutional neural networks. Finally, we provide an overview of the recurrent neural nets and their application in processing sequences.

2.1. Learning from Demonstration

One of the main objective in robots programming is to find an appropriate map between the world state and robot actions, which is also called *policy* [2]. It enables the robot to select actions based on the observations on the current (and possibility the history) world state.

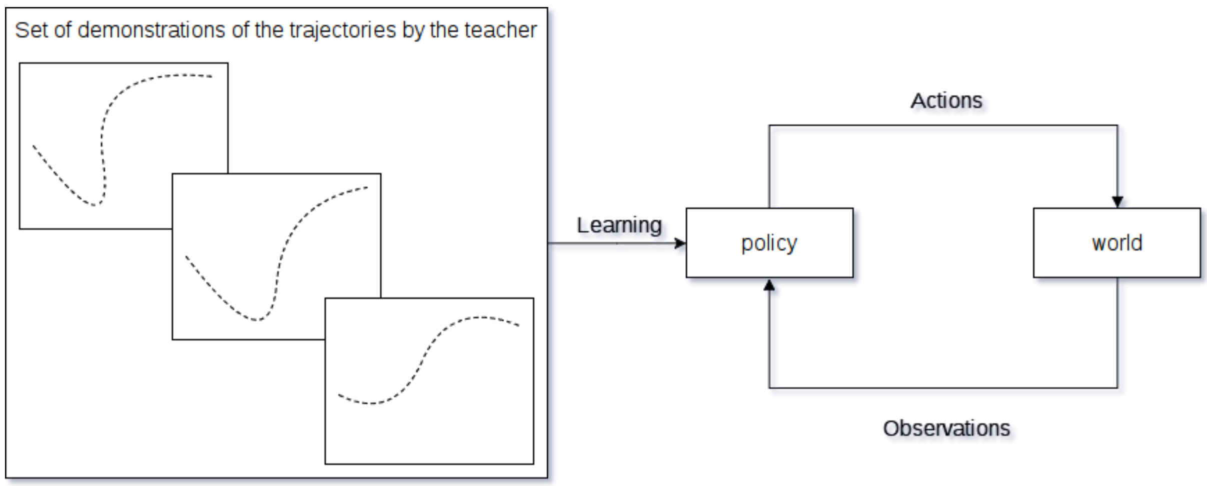

Figure 2.1. Process of learning by demonstration. From a set of demonstrations by the teacher a policy is derived which is then used to select robot actions based on the observation from the world state.

Policies can be acquired in different ways. A trivial way would be to design them by hand. This has been the most common approach sofar in the industry, Where a team of programmers will study the problem carefully, determine all the constraints and program the robot controller for every step of the process. The process is time consuming and requires many hours of testing. Another approach is learning through examples provided by the teacher's demonstration, or as it's called, *Learning from Demonstration*.

Examples are defined as sequences of world states and actions. LfD then utilizes this information to derive the policy for the robot to accomplish the same task demonstrated by the teacher (figure 2.1). However, to provide the demonstration different design choices are available. Basically, the designer has to choose:

1. How to demonstrate the data.
2. How the data would be presented to the learning algorithm.

The former is referred to as the choice of the demonstrator and the latter as the demonstration technique (the way designer choose to feed the data to the learner algorithm) [2].

Alternative choices for the demonstrator includes a human demonstration in front of a camera, teleoperating the robot, kinesthetic teaching [1], hand-written code etc. In this work, we choose the hand-written code as our mean to demonstrate the task to the robot. This can later be replaced by teleoperation if needed. One main advantage of using the robot's body is that it automatically solves the *correspondence* problem. As no intermediate mapping from the teacher's body to the robot is necessary.

As for the demonstration technique, we will feed the recorded trajectories, images, and the robot's configuration directly to our algorithm. In brief, our technique will learn a function to map raw images and the pose of the end-effector directly to the policy which is defined as the joint values. In other words:

$$D(z', a') \rightarrow \pi(z) = a \tag{2.1}$$

Where π is the learned mapping function, *z'* is the collection of the camera images and pose values, *a'* is the joint values, *z* is the observations during the execution of the task by the robot, and *a* is the action taken by the robot.

2.2. Trajectory Segmentation and Controller design

While many attempts in LfD in the past has been focused on learning monolithic policy, more recent approaches have incorporated different methods to segment the demonstrations and learn a policy for each segment [5][8][1], which will lead to more flexibility in learning of complex, multi-step tasks. This approach also offers generalization of the learned policy to novel situations where the system will be able to pick any of the learned skills and connects them in such a way that can solve the problem at hand. However, the power of these methods differs in term of their generalization flexibility as we discuss later in the next chapter.

Traditionally, trajectory segmentation has been considered as a statistical problem where methods such as BP-AR-HMM [9], Change point [10] and ST-AR-HMM [5] are used to split the demonstrated trajectory (usually based on kinesthetic demonstration) into separate segments in order the learn the underlying motion controller.

To discuss some of the mentioned approaches we need to first provide a brief introduction to Markov processes. *Markov process* is defined as a sequence of random states, where (assuming the Markov condition), Given the current state, the probability of the future states are conditionally independent of the past.

$$p(x_n|x_{n-1}, x_{n-2}, \dots, x_1) = p(x_n|x_{n-1}) \tag{2.2}$$

In many cases, the actual state of the system are not accessible, but only latent variables which represent some features of the underlying state can be seen. This process is called *Hidden Markov Model*, and latent variable is called the observation of the state. In the standard HMM, same transition matrix describes the evolution of the system, and observations, given the corresponding state (figure 2.2) are independent of each other.

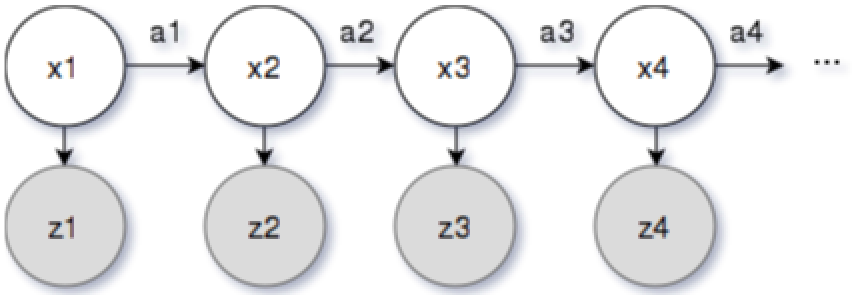

Figure 2.2. Standard hidden markov model (HMM) where at each state, only the the latent variables (observations) are accessible

Now we're ready to explain some of the above mentioned approaches in the trajectory segmentation. BR-AR-HMM (Beta Process Autoregressive Hidden Markov Model) is first proposed by fox et. al. [9]. The main advantage of this method over the standard HMM is two fold. First, the Beta process allows each segment of the demonstration to be represented by a set of primitive motions or features that are discovered by the system earlier. Second, Autoregressive can relax the independence assumption of the observations and allow temporal connections between the latent values of the HMM. Niekum et. al. [1] then used this method and introduced semantic segmentation by clustering the segments to their specific coordinate frame which could be placed on any objects or the robot depending on the subgoal of that segment to construct a library of skills which can later be used. However, in their experiment, objects were represented by AR tags. All the relevant object to the task had to be present and their pose was extracted prior to the execution. Kroemer et. al. [5] used

ST-AR-HMM (State-based Transition Autoregressive Hidden Markov Model) as the starting point to capture the phase transitions. They then defined an extension to this approach to encode entry and exit conditions of each *phase* of the demonstration.

Konidaris et. al. [10] have used the Change point detection as the segmentation tool. The algorithm works by detecting points on the trajectory where either the most relevant abstraction changes or where the trajectory on both sides of the point is too complex to represent as a single segment. Where abstractions are the most relevant motor and perceptual features to the task. Their goal was to segment demonstration trajectories into skill chains and merge skill chain from multiple demonstrations into a skill tree. However, they approach fails to detect repeated skills and an efficient design of the abstraction set for each task is not practical for real world scenarios.

Other Similar methods have been used to segment the demonstration in an unsupervised way. However, they usually totally ignore, use AR tags or tailor the visual features of a natural scene to the specific objects and predefined tasks.

More recently Garg et. al. [8] has shown the potential of using Convolutional neural Networks (2.4) in the task of segmentation and more specific Transition State Clustering with Deep Learning. Where they use the features extracted from a CNN together with the kinematics features of the task to discover the transition points in the demonstration. Our approach is similar to this method as we'll see in Chapter 5.

Traditionally, the task of segmentation and learning the low-level controller for each segment has been separated and control algorithm has been more focused on learning monolithic policies. Works such as [28][8][1] has shown that the trajectories for more complex tasks can be segmented automatically into *sub-skill* and learned independently with separate controllers specialized for each skill to command the robot. One of the main choices among the controllers, has been DMP (Dynamic Movement Primitive) [29]. This method uses a set of nonlinear differential equations to control dynamical systems. It worth noting that this is a powerful method in which by introducing an additional canonical system, the stability and convergence are guaranteed. Moreover, this controller can easily facilitate the trajectory scaling in both time and space.

However, in more recent approaches, CNNs have also been used to generate motor commands. [6][11][12], In our opinion, this is a more robust approach as visual features of the scene are directly used to infer the state and adjust the robot's configuration accordingly to achieve the desired objective.

2.3. Grasping

Grasping has been a research topic in the robotics community for many years. More recent works have adopted the CNNs as the main tool to extract relevant information for the grasping task. Levin et. al. [6] uses the CNN to predict the probability of success for the sample motor commands before the grasp attempts.

Figure 2.3. multiple robots are attempting to grasp diverse set of objects to gather training data for the task of success prediction [6]

Davison et. al [19] uses the pre-processed depth images to predict the probability of the grasp in a long vector which represents the location and 2D orientation of the grasping points along with the probability of success. Lenz et. al. [20] uses a multi-step approach of generating many grasp proposals and then ranking them to find the most suitable one.

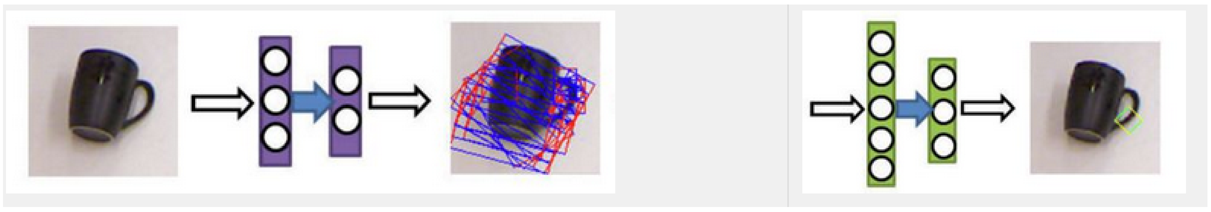

Figure 2.4. multi step approach of generating grasp proposals and ranking [20]

Another class of methods is focused on visual servoing where a closed loop between the image feed and motor commands is designed to guide the robot continuously toward the suitable grasping point [11][20]. Our approach is similar to [19], more details will be provided in chapter 4.

2.4. Convolutional Neural Networks

Over the past few years, CNNs [17] have revolutionized the image processing problem. They have been successfully used to detecting objects, semantic segmentation [22] and more recently they have been applied to the field of reinforcement learning, or as it is called, Deep Reinforcement Learning, such as Playing Atari games [13], continuous control [23] and visual servoing of robots [11].

As the trend shows, CNNs are becoming more mature, and they have been increasing used to extract visual features in the robotics research as a vital source of information when interacting with the environment. For the above reason, we will be using this model in our framework in multiple places, where generally there is a need to encode features or reduce the dimensionality of the image data in order for us to efficiently combine the visual features with robot kinematics features to achieve better stability and reliability in our segmentation and control modules.

2.5. Recurrent Neural Networks

In general, Neural network is a powerful class of computational models that can be applied to many different fields. Depending on their architecture, they can internally represent linear and nonlinear maps. However, one main disadvantage of these models, including the CNNs, is that they cannot hold the state, or basically they're not designed to process sequences. Luckily, recurrent neural networks and in particular, LSTM [24] solves this problem. Moreover, these models have shown better performance and more generalization capabilities comparing to the traditional Markov models [25]. Another recent application of the RNNs is the visual attention [26], where these model can help to extract and process only the relevant parts of the image to acquire the necessary information versus in traditional models where every pixel in the images are processed which would increase the learning time.

Since the task of robot control can naturally be represented as time series, using RNNs are one of the natural choices to process the sequences of kinematics and even visual features that are extracted from the robot's behavior, or in our case the demonstrator's trajectory in order to predict the motor commands. As we will see in chapter 5, this model will be the at the center of our segmentation and trajectory generation module.

Chapter 3

Simulation and Demonstrations

We start this chapter by introducing our approach in demonstration of a multi-step pick and placing task to the robot. All the demonstrations have been done in a simulated environment. (A clear concern regarding the use of simulation would be transferring to the real-world application which will be addressed in section 4.4). We then step through the process of data collection and preparing datasets to train our models.

3.1. Simulated scene

In this work, we've used the V-REP (Virtual Robot Experimentation Platform[1]) [14] to simulate the learning process. Our robot is the Universal Robots UR10 with 6 DOF arm available in the simulation software. A two finger gripper (RG2) is attached to the robot as shown in figure 3.4. A stereo RGB camera captures the images on both sides of the gripper and a depth camera is placed in the middle of the gripper. The reason behind this design will be explained in the next chapter.

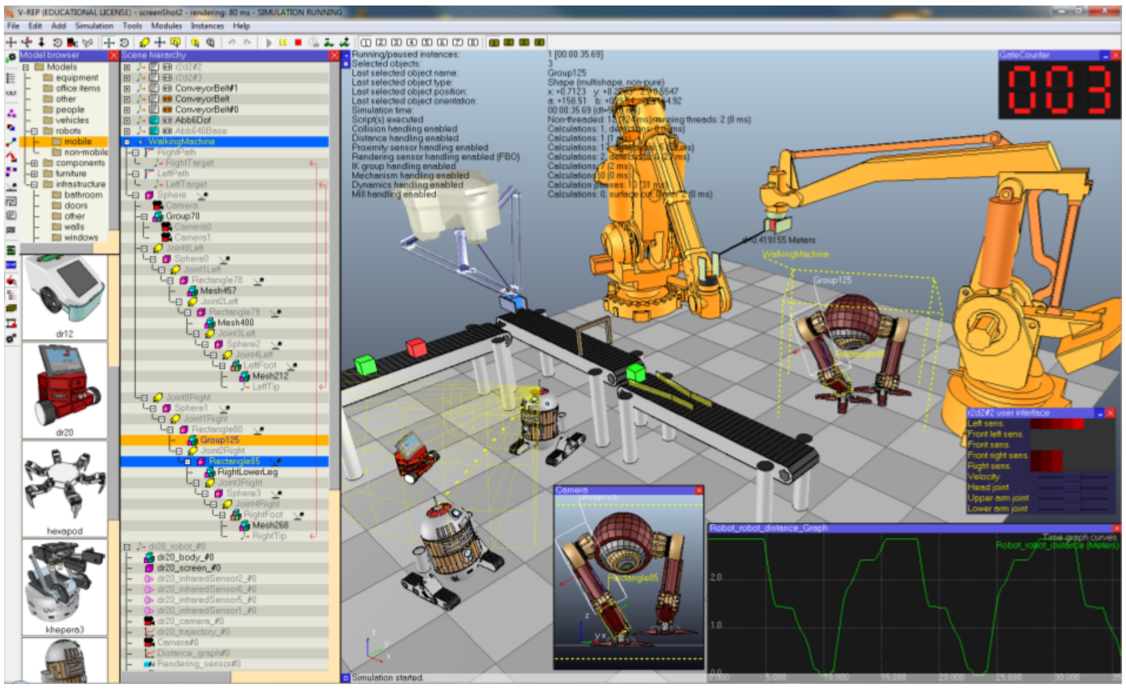

Figure 3.1. A screenshot of the V-REP simulation software used in this work [14]

[1] The educational edition of V_REP is available for free download

Using the remote API designed to communicate with the simulated scene and ROS (Robot Operating System from Willow Garage) [15] the robot has been programmed to demonstrate predefined multi-step tasks as explained in the next section.

3.2. Demonstrations and Objectives

To evaluate the performance of our framework, two tasks were designed with the goal that the skills learned from the first task would be transferrable to the second task.

Task 1: In the first task, the model should learn to find the free grasping points. To make the predictions more robust, (and to tackle one of the old problems in the robot grasping, namely, *Bin Picking*) instead of individual items, the model will learn how to pick an object from a homogeneous pile. The goal here is to encode the user-defined grasping points on each specific object and later use this skill when required in a novel situation. Figure 3.2 and Figure 3.3 show a randomly generate piles of the two object used in our experiment (cup and nut). Two cameras (1 RGB + 1 Depth) are placed in the same location above the pile. The image on the top-right corner shows the depth map. The details of our grasp point detection will be elaborated in the next chapter.

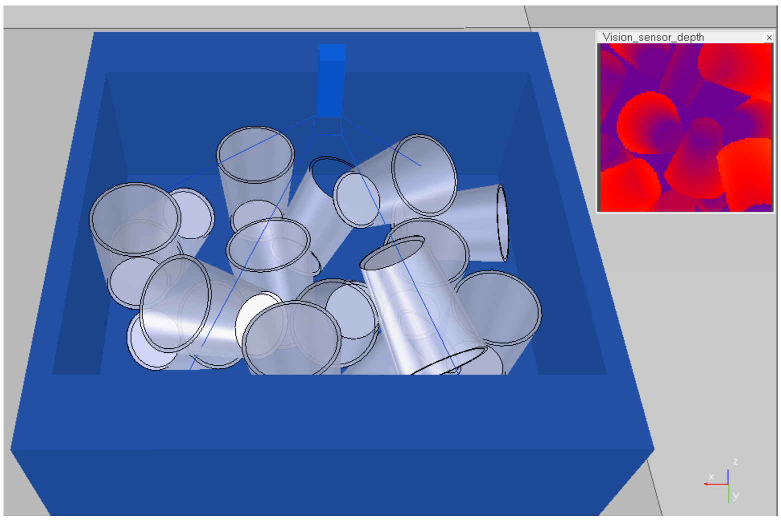

Figure 3.2. Randomly generate pile of cups.

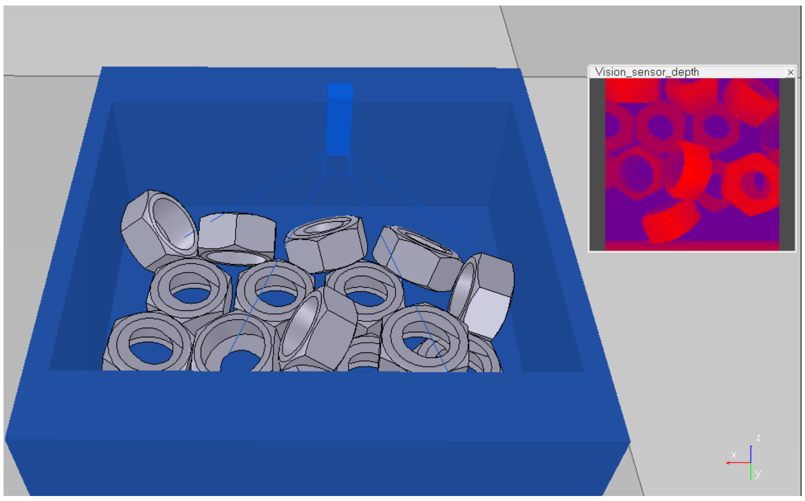

Figure 3.3. Randomly generate pile of nuts.

Task 2: In order to demonstrate the trajectory of a multi-step manipulation task which involves picking and placing sub-tasks, the following scenario is defined. (Figure 3.4 shows the designed scene for task 2)

Step 1: Pick up the cup from its initial position

Step 2: Place it vertically in the empty box on the right hand side of the robot

Step 3: Pick up the nut from its initial position

Step 4: Drop it into the cup

Step 5: Pick the cup containing the nut

Step 6: Place it on the table

In the next section, we explain in more details how the demonstrations are provided. But it worth noting that injected randomness into the initial environment's state lets the learning process to be more robust. The task might seem simple, however the goal here is to evaluate the proposed solution and verify whether the previously learned skills can efficiently be used to imitate the demonstration in a multi-step pick and placing task.

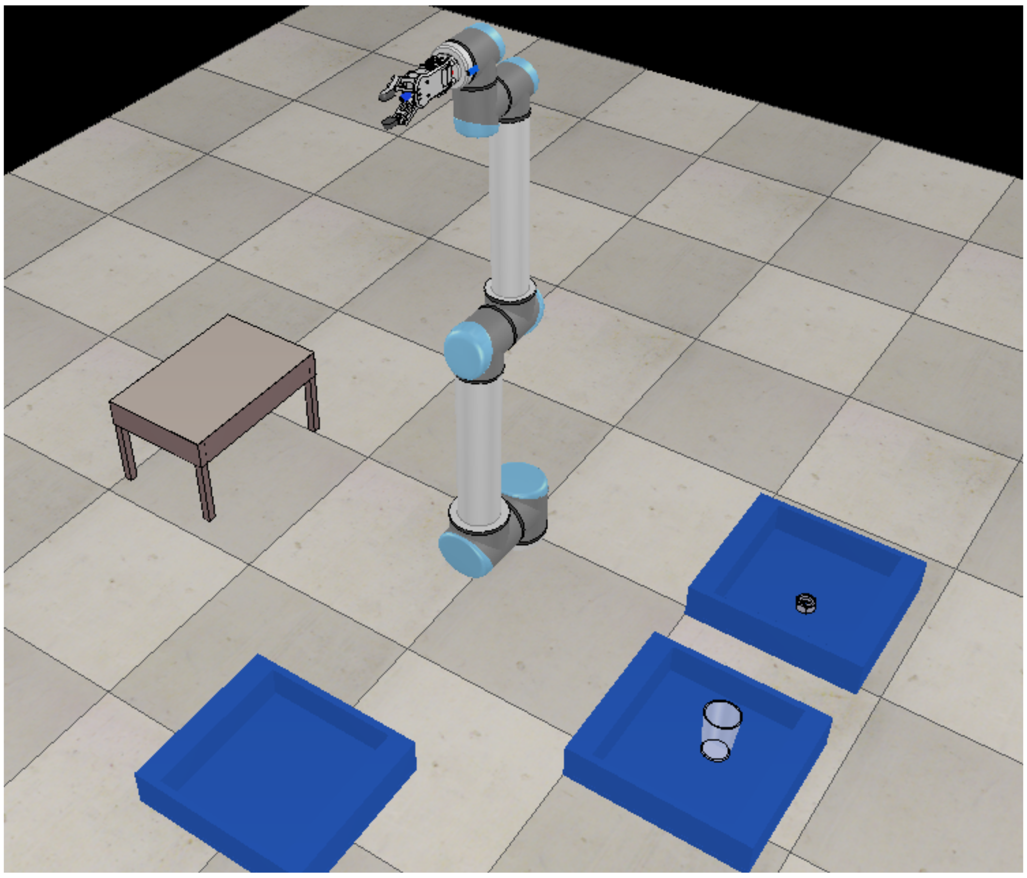
Figure 3.4. Designed scene for the demonstration of task 2.

3.2. Task demonstration

As previously discussed, there are different choices of the demonstrators [2] (By Human teacher in front of the camera, explicit coding, kinesthetic, teleoperation, etc.). In our experiment, after randomly initializing the object positions in the scene we used an explicit program with injected randomness in the solution of the IK solver to demonstrate the task. For this purpose, we use trac-IK [16], a generic Inverse kinematics solver available as a ROS package. We implement an interface for the IK solver to make it accessible to the client program as a service. Figure 3.5 shows an overview of the demonstration procedure.

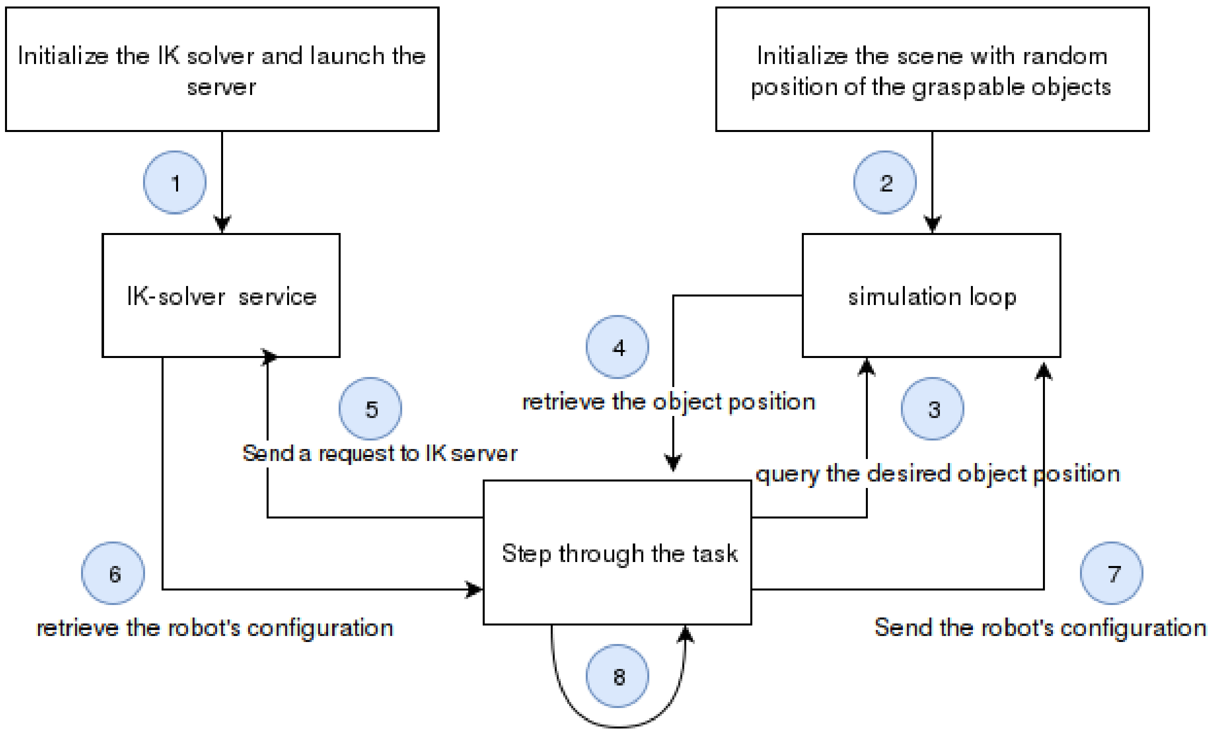
Initialize the IK solver and launch the server
Initialize the scene with random position of the graspable objects
1
2
IK-solver service
simulation loop
4
retrieve the object position
5
3
Send a request to IK server
query the desired object position
Step through the task
6
7
retrieve the robot's configuration
Send the robot's configuration
8

Figure 3.5. Overview of the demonstration procedure

Also, the algorithm one shows the pseudo-code of the demonstrations:

Algorithm 1: Task demonstration procedure

```
demonstrate(Task-name):
1:    O: collection of the object names from the configuration file of the task
2:    JL: Robot's Joint limits retrieved randomly from a set of predefined valid joint values for the task
3:    Initialize_IK_server(JL): Using the robot's definition and joint limits initialize th IK solver service
4:    Initialize_the_scene(O): Initialize the scene and set a new position for the graspable objects within their containers boundary
5:    Repeat:
6:        query the relevant object pose for the current step from the simulation
7:        retrieve the position
8:        send a request to IK server to find the desired robot's configuration
9:        retrieve the robot's configuration (joint values)
10:       send the joint values to the robot
```

Figure 3.6 and 3.7 show two sample demonstrations of the task 2 generated using the above procedure. As shown, even though generated sequences have obvious differences, they share the similar set of constraints or keypoint, which identifies the phase transitions (more about this in chapter 5), by which the higher purpose of the demonstration can be extracted by the algorithm.

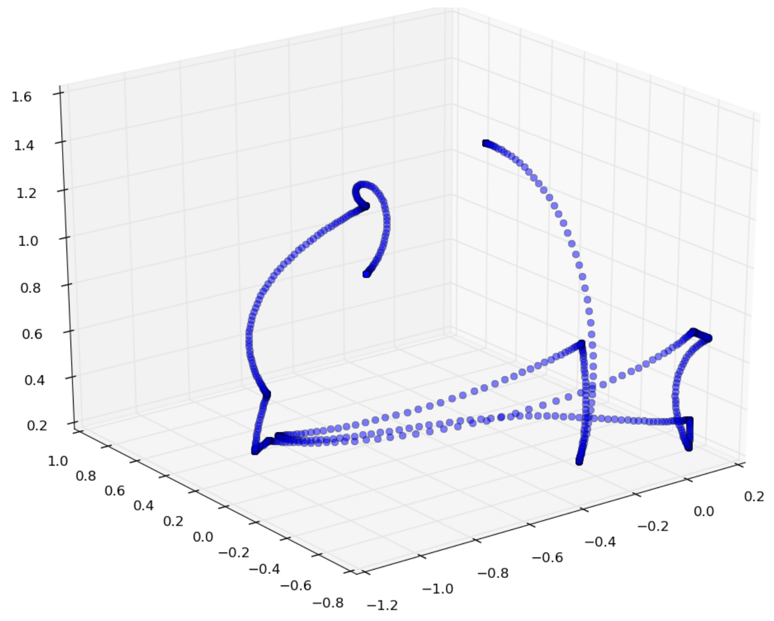

Figure 3.6. Sample demonstration of the task 2, trajectory shows the evolution of the robot's end-effector Cartesian coordinates

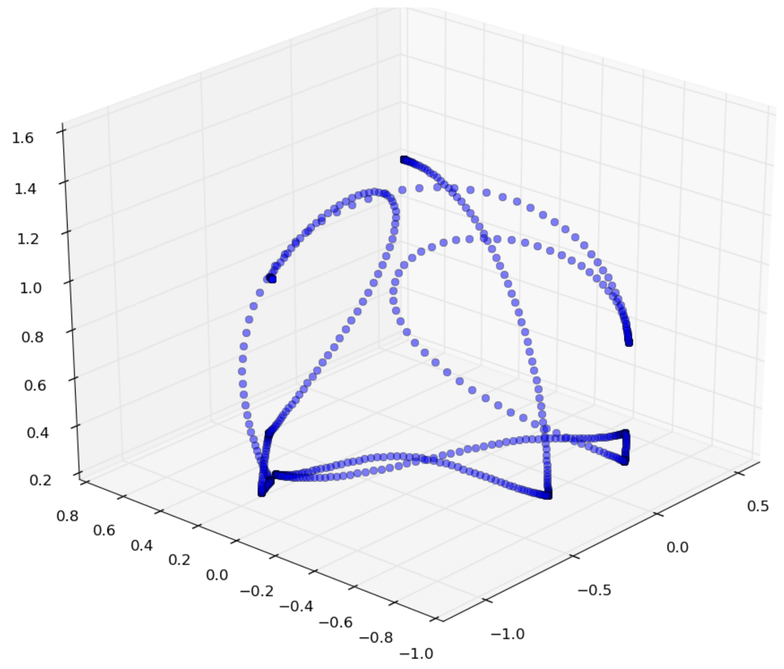

Figure 3.7. Another sample demonstration of the task 2

Also, Figure 3.8 shows the full pose information (3D Cartesian coordinates and 4D quaternion) of the end-effector as recorded during a sample demonstration.

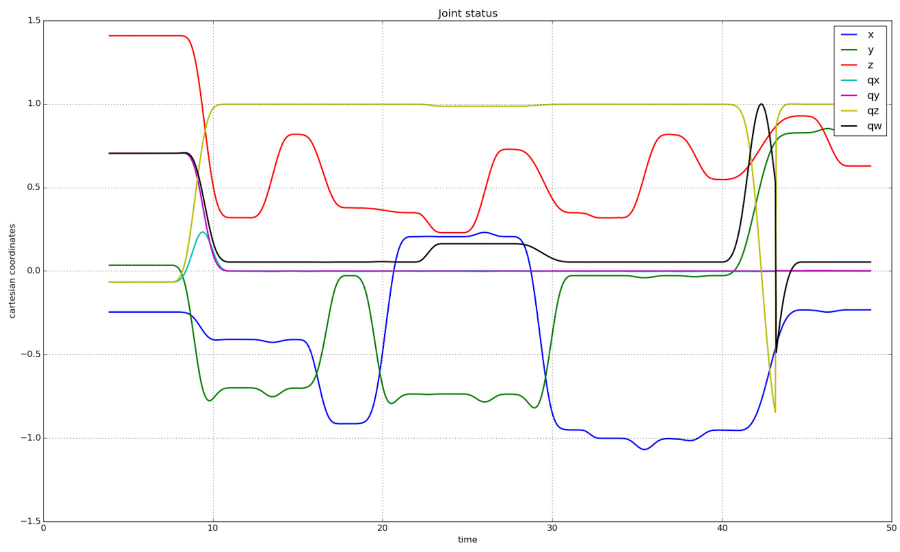

Figure 3.8. Robot's end-effector pose values duration the task demonstration

While demonstrating the task, beside recording the pose of the robot's end-effector, the images from both RGB and the depth camera and the robot's configuration has also been stored for further processing. Figure 3.9 shows a sample recording of the robot's joint values corresponding to a trial in the task 2.

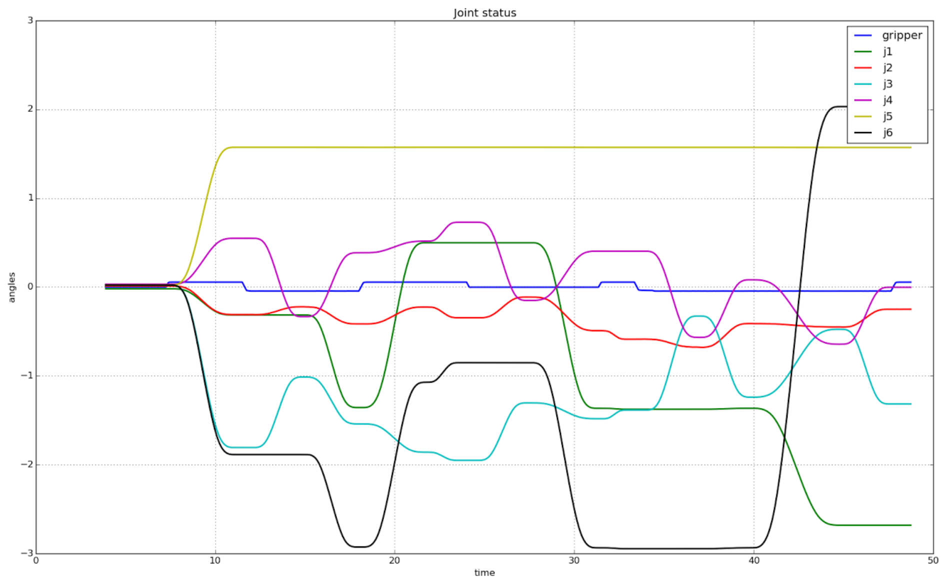

Figure 3.9. Robot joint values duration the task demonstration

Also, figure 3.10 shows a sample of the recorded images from the cameras and the scene status while capturing those images.

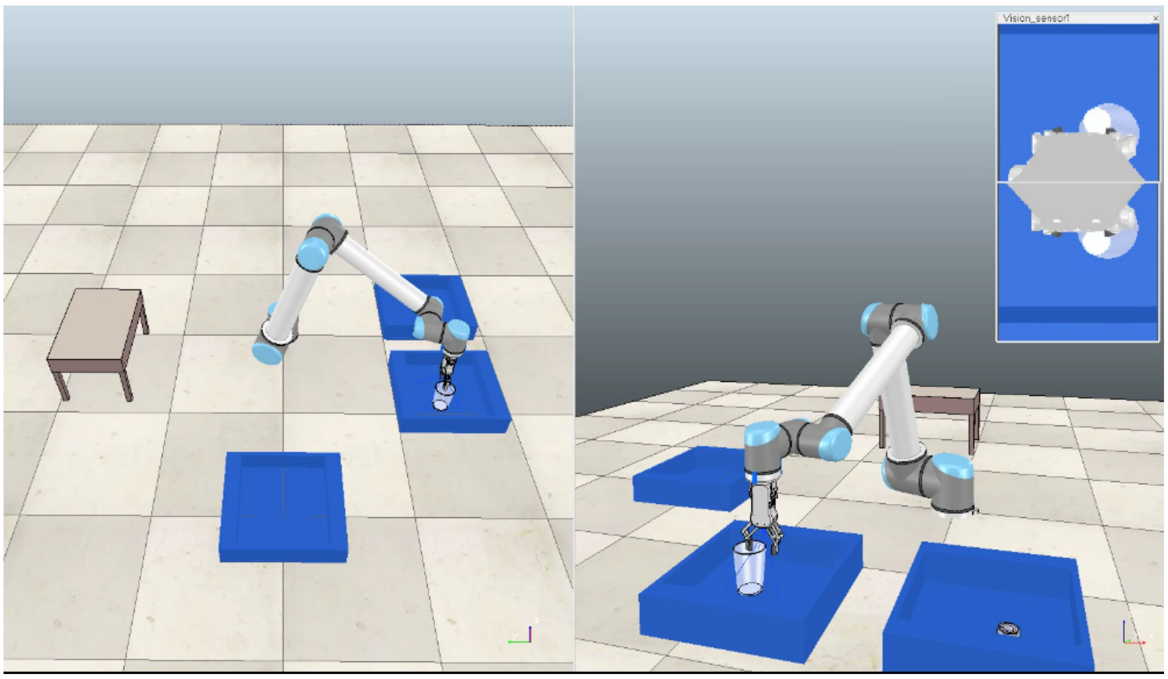

Figure 3.10. The simulation scene while the demonstrator program is running and the top-right corner of the image shows the stereo RGB camera images

To summarize, table 3.1 shows the data recorded during the demonstrations in each time step of the simulation:

Table 3.1. summary of the recorded data during the task demonstrations (per time step)

Data Type	Size	Description
Pose	7	Cartesian coordinates: 3 Orientation (quaternion): 4
Joint values	6	Robot’s DOF: 6
Gripper	1	Openness value of the gripper: 1
Images	7 x 128 x128	Stereo RGB images: 2 Depth Image: 1

Chapter 4
Grasp

In this chapter, we propose our approach to grasping objects, particularly from a homogeneous pile. We introduce a new approach to define grasping points on different objects and use camera filters to extract the ground truth vectors and use them to train our CNN model. Next step we demonstrate an efficient way to generate training data for the model training. Then we move on to our model architecture and we close this chapter by discussing the issue of transfer learning. Note that our goal is to separately learn the grasping skill of different objects and add them to the skill library. Later, based on the object similarities, the robot will automatically fetch the relevant grasping skill from the library when it faces a grasping segment in the demonstration.

4.1. Grasp point detection

We train our model to predict the grasp points in a pile. Why training on a pile instead of separate objects? The reasoning behind using the pile is to make the prediction more robust. Learning in presence of more limitation makes the prediction more reliable. Also grasping from a pile, or *Bin Picking* has been a problem in the industrial application for more than 30 years, we hope this would be a step toward solving that problem.

However, we try to simplify the problem. Instead of considering the general grasping problem, we want the robot to learn a predefined grasping locations on different objects due to the practicality as required by the nature of the bin-picking problem.

To define the grasping points, we use a layer of visualization in the simulated scene that is invisible to the robot's hand camera. Grasping points are defined as a small pair of cubes around the object. (Figure 4.1)

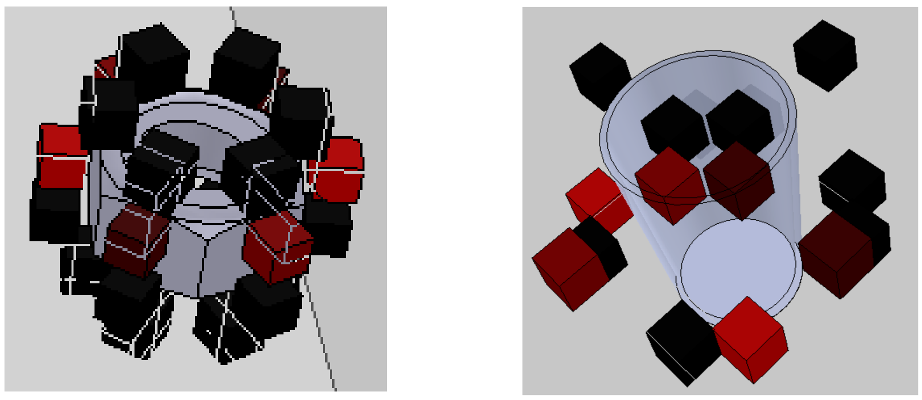

Figure 4.1. User defined grasp point on the nut (left) and the cup(right)

The way it works is by enabling collision detection between each pair of these cubes with all other objects in the scene and hiding them when the collision is true. This way, true labels will only correspond to collision free grasp points. While the input to the predictor is a raw depth image. Figure 4.2 shows a sample of the training data generated to train the model.

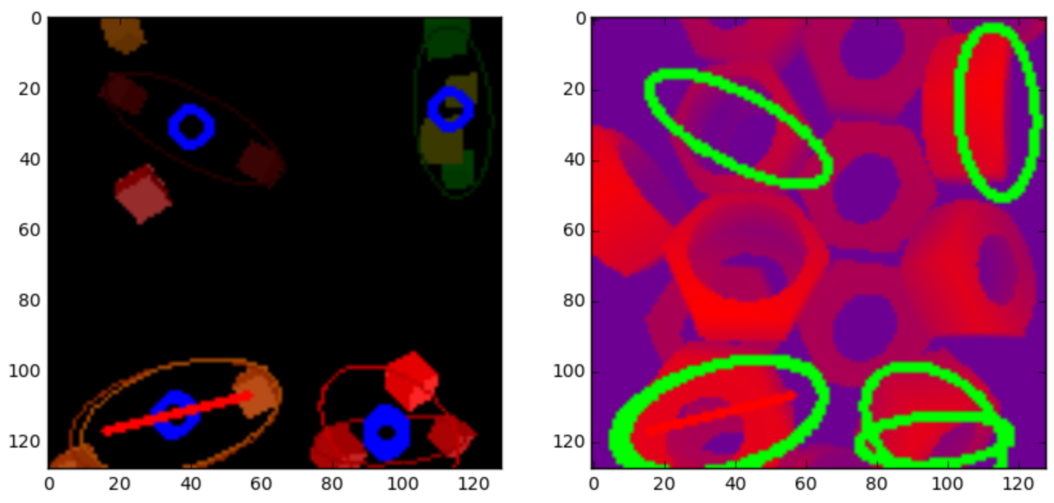

Figure 4.2. left image show the color-coded collision free grasp points and the right image is the visualization of the ground truth label on the corresponding depth image.

We would like to design this problem as a classification to make the learning more tractable. Our approach in detecting the grasping points is based on the work of Davison et. al. [19]. But we extend the idea by introducing a two-step approach to extract the location and 2D orientation of the grasping point. (An extension to this work will further add efficient 3D pose extraction, however, to simplify, we restrict this paper to two dimensions).

In the first step, we lay a 2D grid on a horizontal plane above the pile, as shown in figure 4.3:

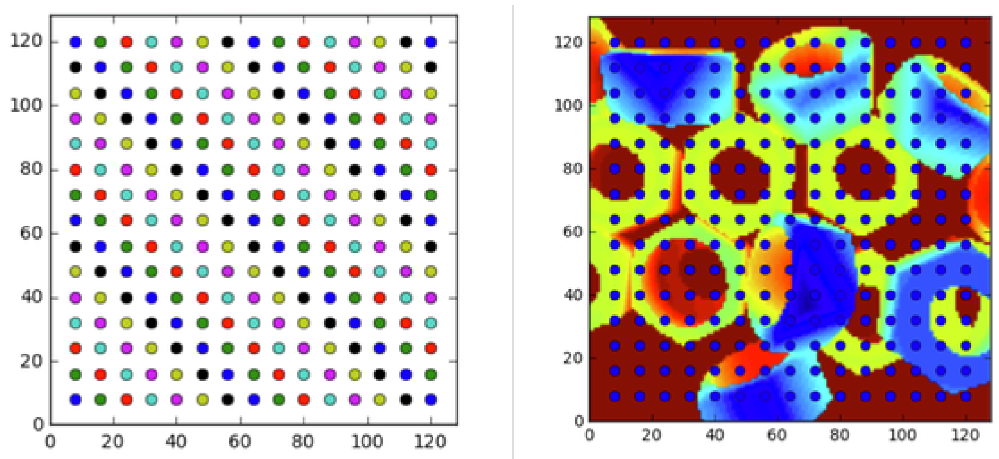

Figure 4.3. visualization of the 2D grid to estimate the approximate location of the available grasps in the first layer of our prediction algorithm

Our model will learn to predict the probability of finding a grasp on each point in the grid. We treat this problem as *multi-class,multi-label* case where the multi-class refers to the fact that each point in the grid is treated as a separate class and multi-label means, there can be more than one true label in the prediction. In simple worlds, there are multiple possible grasping points available in a single image of the pile. Next step will then post-process these areas to find the highest probability.

To further process the areas with predicted probability of finding a grasp point more than a defined threshold, we crop those areas of the image. Figure 4.4 shows a sample collection of such cropped images

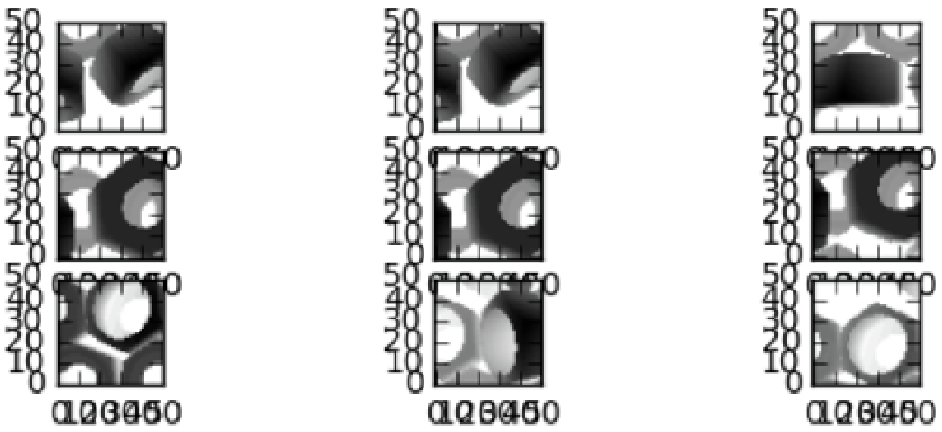

Figure 4.4. Cropped areas of the original image with high probability of finding a free grasping point

The next layer of our prediction model has three objectives. First, to further analyze the predictions and discard them if they don't pass a certain threshold of probability of success, second, to improve the predicted position of the grasping point center, and third, to predict the orientation of the grasp (2D in this work). Figure 4.5 demonstrates possible labels for the second layer of the prediction model.

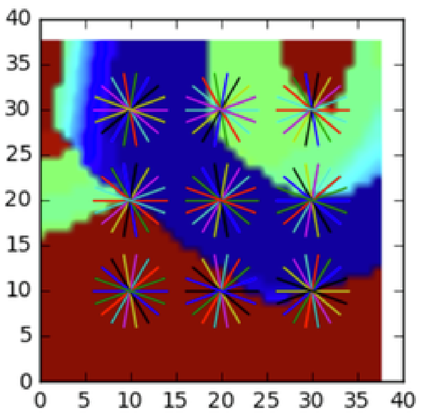

Figure 4.5. Possible predicted location of the grasp point center and orientation of the grasp in the second layer of our model.

This problem is considered as *multi-class, single-label*, where maximum one true label can exist in this layer of prediction.

4.2. Training data generation

To generate training data, a 3D grid above the bins are defined to randomly place each instance of the object and let them fall into the bin to generate random images which then used to train our two-layer model. Figure 4.6 shows a sample random pile generate for the objects used in this work. Worth noting, there's no restriction for this method to be used for other objects.

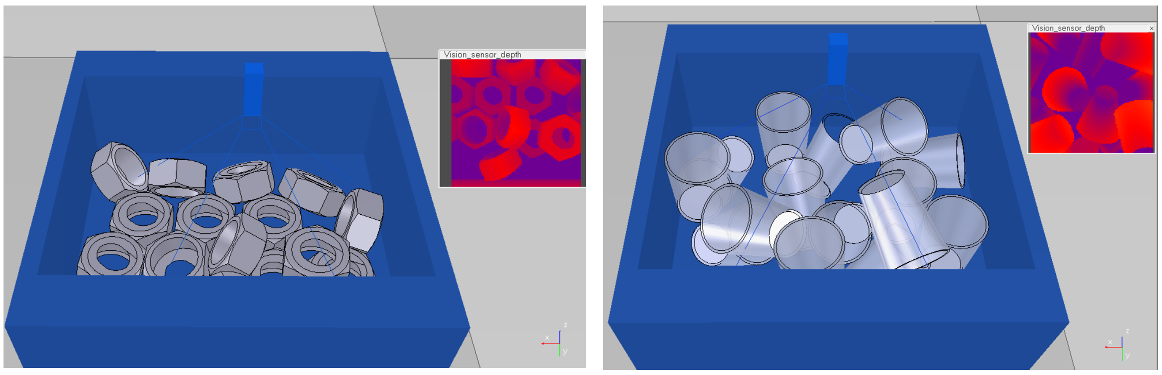

Figure 4.6. Sample randomly generated piles

Algorithm 2 provides the pseudocode to generate the training data.

Algorithm 2: Generate training data

```
generate_training_data (m, n):
        m: 3D model of the object
        n: number of required training data
        i = 0: number of generated data points, initialized to zero
1:      while i < n:
2:           initialze_pile(): randomly place the objects above the bin and release them fall into the bin
2:           Capture_images(): capture images from both depth camera and the maske camera to obtain the ground truth labels
3:           Extract_labels(): Use the mask images to extract the labels
4:           Export_data(): export depth images and their corresponding label vector
5:           i = i + 1
```

4.3. GPN (grasp proposal network)

We call our two-layer grasp prediction model, GPN. It's consist of two CNNs. The first layer predicts the probability of grasp at each point on the grid. It takes an image as input and outputs 1D vector of values that can be interpreted as grasp proposals. Figure 4.7 shows our architecture of the CNN for the first layer:

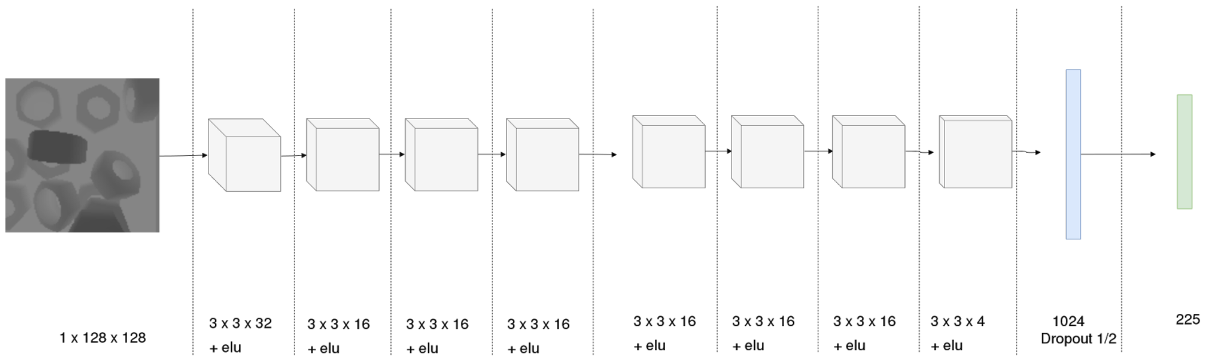

Figure 4.7. Architecture of the first layer of our predictor

In the above CNN, traditional Relu activations are replaced by ELU (Exponential Linear Units) [30] which is claimed to increase the learning speed.

The output layer is a vector with the length of 225. Each scalar in the vector represents the probability of finding a grasp point in the 2D location on the grid corresponding to its index in this vector. As stated before, this is a multi-class, multi-label problem. To train our CNN, we used the *sigmoid_cross_entropy* cost function. This cost function measures the probability error in discrete classification tasks in which each class is independent and not mutually exclusive.

$$sigmoid\_cross\_entropy = -(log\ Sigmoid(t) * z + log(1 - Sigmoid(t)) * (1 - z)) \tag{4.1}$$

$$Sigmoid(t) = \frac{1}{(1+e^{-t})} \tag{4.2}$$

$$logit = log(\tfrac{p}{1-p}) = log(p) - log(1-p) \tag{4.3}$$

Where x = logits of the predictions and z = ground truth labels. As explained, predictions represent probabilities for a successful grasp at each location on the grid. Next, areas around the high probability grid points are cropped (figure 4.4) and passed to the second

CNN which will further analyze those proposals. Figure 4.8 shows our architecture of the CNN for the second layer.

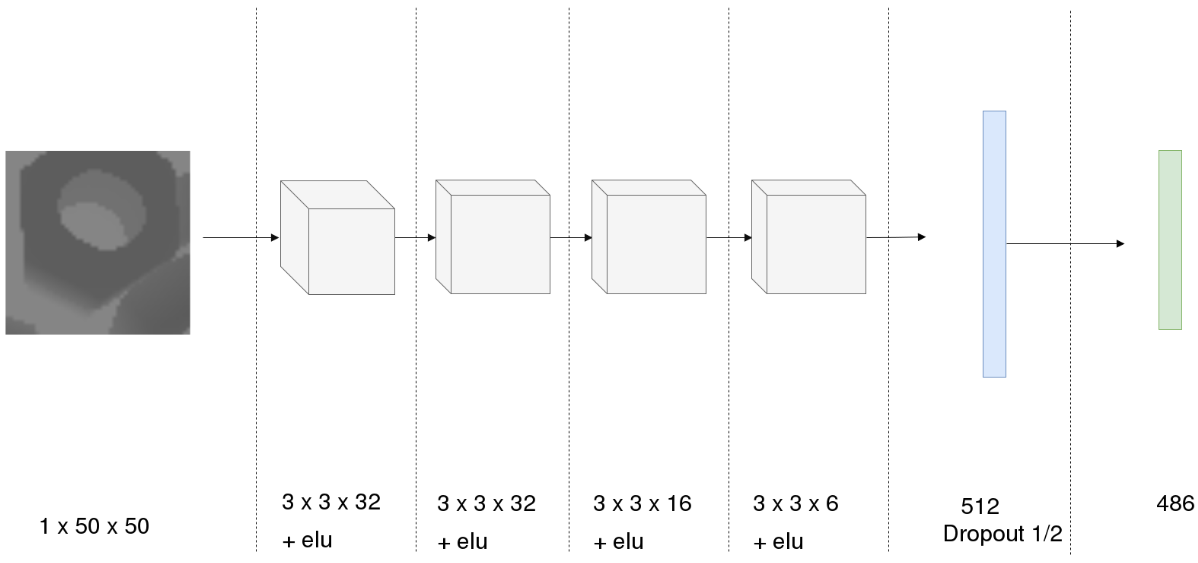

Figure 4.8. Architecture of the second layer of our predictor

Note that the proposed architectures are not claimed to be the most efficient. They've simply performed better during our experiments.

The output layer is a vector with the length of 486. This vector included both the corrected location of the grasp point center and its predicted 2D orientation. This is a multi-class, single-label classification where at most one true label can exist in each prediction. To train this model, we used softmax_cross_entropy cost function. It measures the probability error in discrete classification tasks in which the classes are mutually exclusive (each entry is in exactly one class).

$$softmax\_cross\_entropy = -(log\ Softmax(t) * z + log(1 - Softmax(t)) * (1 - z)) \tag{4.4}$$

$$Softmax(y = j|img) = \frac{e^{x_j}}{\sum_{k=1}^{K}(1+e^{-x_k})} \tag{4.5}$$

Figure 4.9 demonstrates a sample predicted grasp position using the above method. Note that the model is now aware of a possible collision between neighboring objects and the highlighted prediction has enough free space around the object to place the gripper.

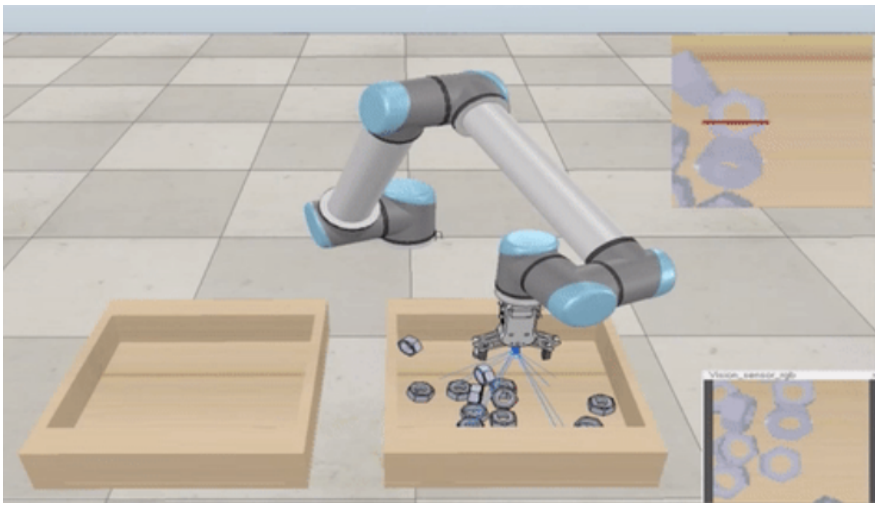

Figure 4.9. sample predicted grasp point using the GPN model

4.4. Transfer learning

One obvious concern about using the simulation to train model is the issue of transferring the knowledge to the real-world applications. First, we count some of the reasons that make the simulator a very efficient starting point for the learning process:

1. Camera filters to see only specific parts of the scene e.g. one camera captures raw images, while the other captures the masked grasp points from the same viewpoint.
2. Keep track of objects in the environment to train the collision avoidance algorithms.
3. Generate over million training data points in very short time which are required to train CNN models.

And many other reasons convinced us that the simulator, can be a good place to pre-train models such as the ones used in this work.
To mitigate the problem of transferring, we propose the following solutions that are valid in our case:

1. Better rendering engines with realistic texturing can help partially to solve this problem.
2. Depth images are used in the grasping and since there's no interference of the colors in the depth images it can easily be translated to the real world applications
3. Using Generative Adversarial Networks [31] to make the simulated images more realistic

Chapter 5
Segmentation and trajectory generation

We start this chapter by explaining our approach to the segmentation of the demonstration, through this section, we will discuss the notion of the phases and transition points and the trick of using the gripper status as the labeling mechanism for our segmentation. This will then facilitate the supervised learning of this task using a Recurrent Neural Network. Next, we will discuss our choice of the motion controller and again we'll introduce yet another RNN which can efficiently generate the corresponding joint values for us. Using these results we will end up combining these network to a simple and yet powerful RNN model that can learn both of these tasks concurrently and use the mutual information to improve the efficiency of both components.

5.1. Segmentation

5.1.1. Statistical approach

As explained in the previous chapter, the reason behind the segmentation is to separate different phases of the demonstration and make it possible to learn a complex multi-step task. As in this work we are dealing with the task of pick and placing, clear phases in each demonstration can be easily defined by the teacher. These segments, in our case, can simply be the same as the steps define the task demonstrations. i.e. section 3.2, task 2.

If examined carefully, it's clear that each task can be separately learned without affecting the learning process of the neighboring phases. E.g. Grasping skill of different object can be acquired independently of the joint trajectories that are necessary for the robot in each task to transform the gripper pose in the pre-grasp and post-grasping phases.

To examine how the Markov processes defined in Chapter 2, such as BP-AR-HMM can help us to segment the trajectories, we ran an experiment with the code from [1], on a recorded data from 3 different demonstrations. Once on the robot's configuration, including the gripper values (figure 5.1), once without the gripper values and once only on the pose of the end-effector (figure 5.2).

As seen in the figure, many segments are detected based on only statistical analysis of the trajectory. To include visual features of the scene, Niekum et. al. [1] added the position of all the relevant objects in the demonstration into the segmentation procedure by tracking the AR tags on each of them. However, since our final goal is to learn the process of object picking from a pile in the industrial setting, this approach won't be helpful for us.

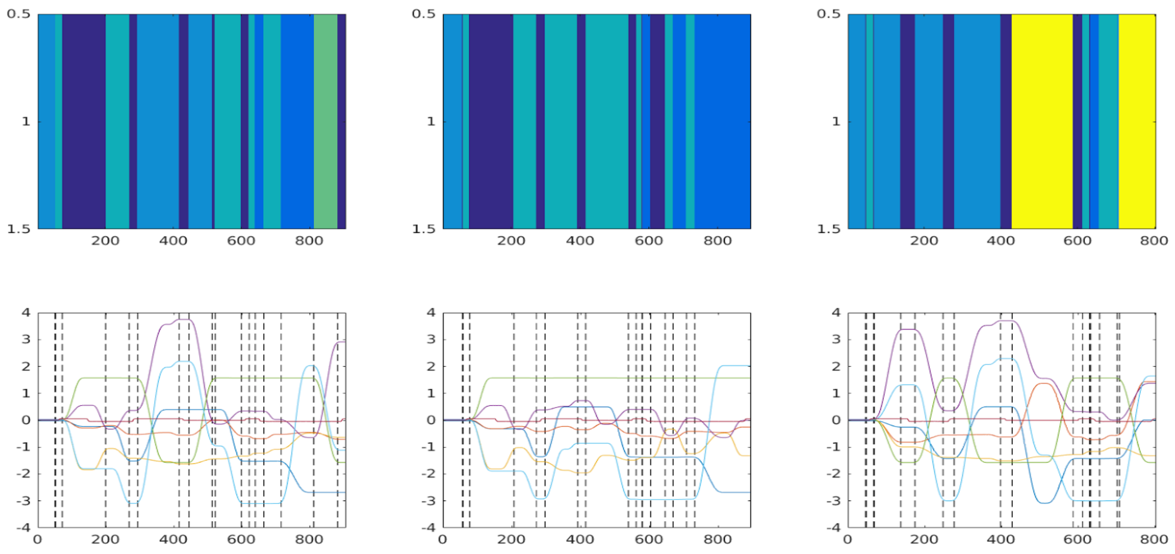

Figure 5.1. BP-AR-HMM segmentation of the joint value, including the gripper status

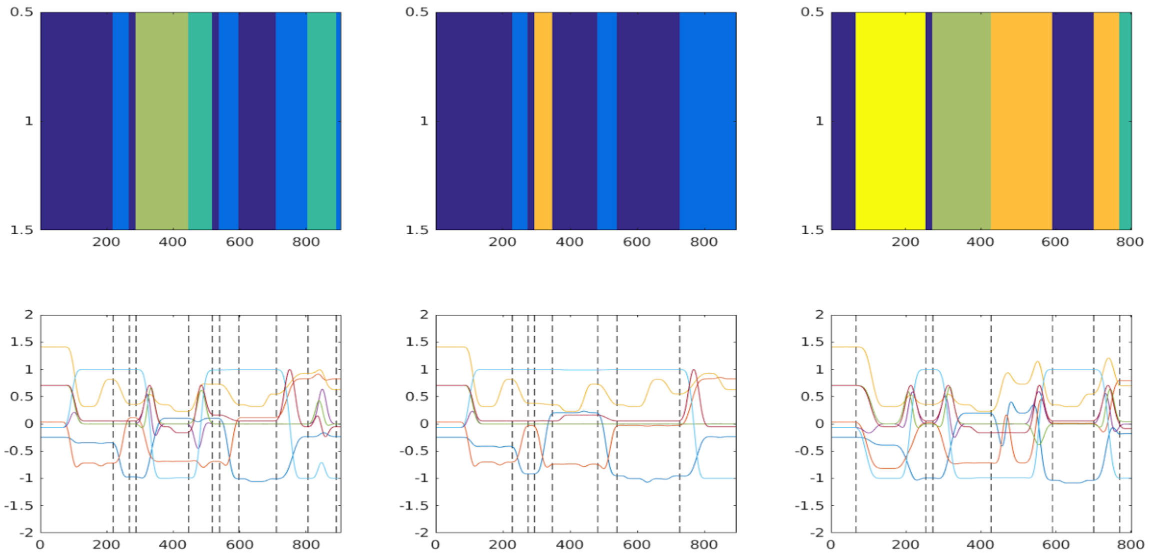

Figure 5.2. BP-AR-HMM segmentation of the pose values

5.1.2. Stereo to depth encoder

Our goal is to find a robust solution with ideally fewer hyperparameters to tune. While efficiently incorporate the visual information in our segmentation method.

As suggested by [8], we decided to use convolutional neural nets to extract visual feature that would help us in both the segmentation and trajectory generation steps. We needed a set of features that would contain information about the position of the objects that are visible to the camera and their distance from the depth camera. To solve this problem we introduce a convolutional encoder of the stereo to depth images. (figure 5.3).

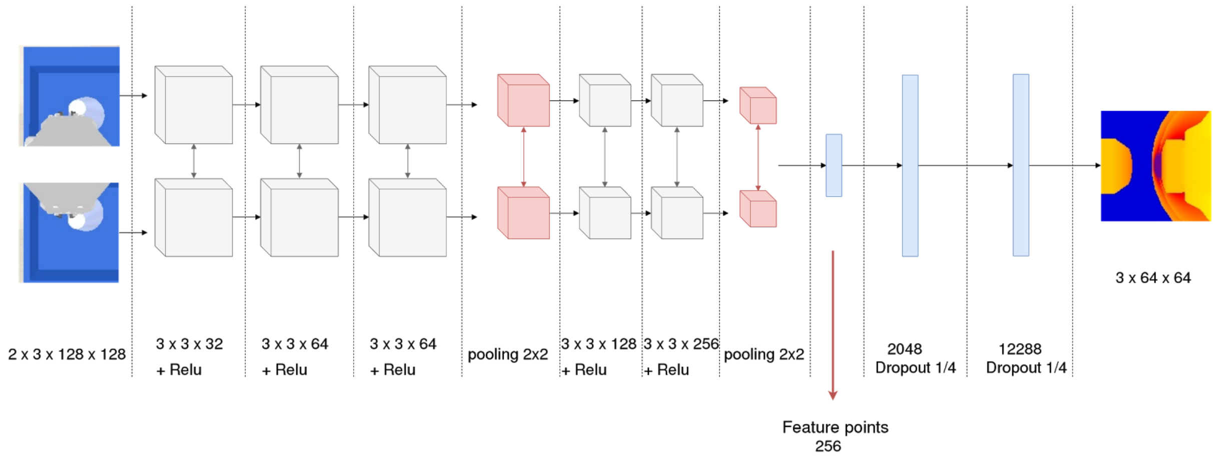

Figure 5.3. Architecture of our Stereo to depth encoder

The two-way arrows between the layers indicated the parameter sharing. Since both layers are processing similar images, there's no need to separate the parameters and this will drastically reduce the training time and unnecessary complexity of the model.

Our architecture of the convolutional layers is inspired by the VGG model [18]. However with much less convolutional layers as our datasets are not comparable to the ones, these models has been designed for. Table 5.1 shows a summary of the architecture.

Also, note that here depth images are constructed out of the combination of 3 channels, hence the size of the last layer.

To train the model, we used a simple RMS error of the difference between the ground truth depth images and the reconstructions:

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2} \tag{5.1}$$

Table 5.1. summary of the architecture of our Stereo to depth encoder model

Layer	type	size
1	input	2x3x128x128
2	conv	3x3x32
3	conv	3x3x64
4	conv	3x3x64
5	max pool	2x2
6	conv	3x3x128
7	conv	3x3x128
8	max pool	2x2

9	fc	256
10	fc with dropout	2048
11	fc with dropout	12288
12	output	3x64x64

Figure 5.4 and 5.5 Show sample of reconstructions and their corresponding ground truth images.

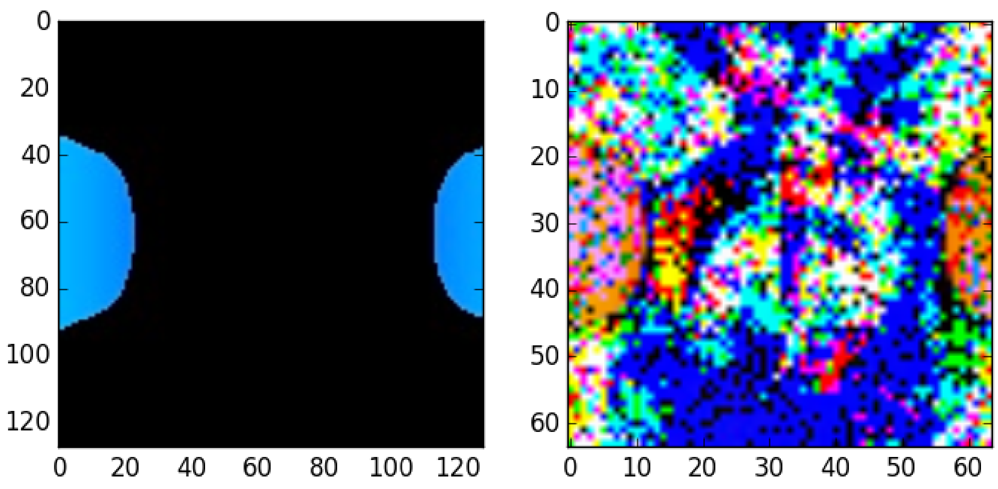

Figure 5.4. Sample depth image reconstruction from the stereo RGB images. The image on the left is the ground truth depth image and on the right is the reconstruction. Note that the network learned to associate visibility of part of the gripper to the high probability of an object existing between the fingers, hence a shadow of a nut or half a cup can be seen in the reconstruction.

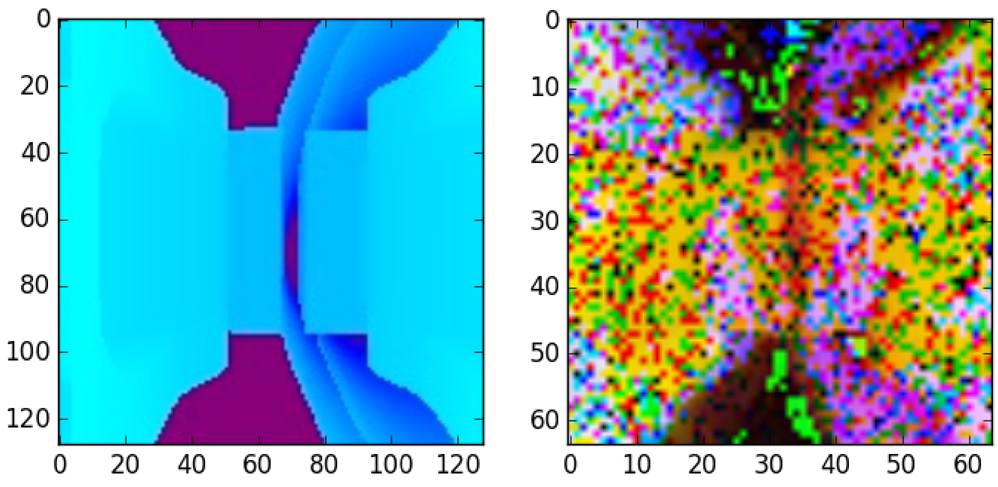

Figure 5.5. Another example of the stereo RGB to depth reconstruction.

5.1.3. Actions types

In the next step, we use the intuition that the gripper status can be directly used to segment the demonstration as the main objective of our demonstrations are pick and placing. To make this more clear, we've shown a sample gripper values from a demonstration of the task 2 in figure 5.6:

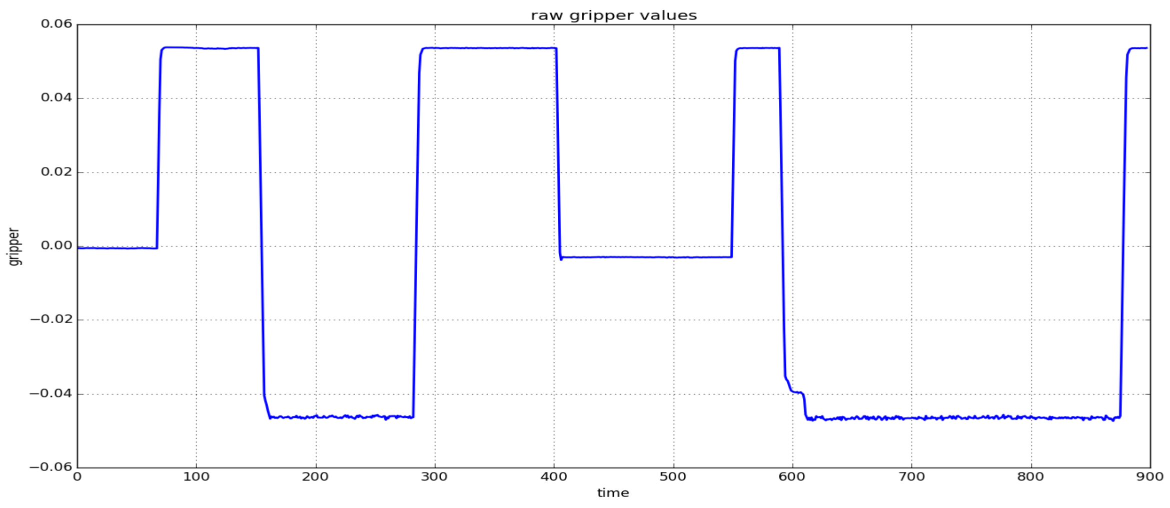

Figure 5.6. Gripper joint values from the recording of a sample demonstration

As it can be seen in the Figure 5.6, the gripper values are highly correlated to the task steps that we've defined earlier and they can be used in our segmentation process. However, to efficiently use this information we defined a concept of *Action Types*.

Similar to approach in [5], we would like to distinguish between the gripper and non-gripper actions. The reasoning behind this, is that, primarily for the task of pick and placing, actions are centered around the gripper joint values, before grasping, gripper should be opened up to an appropriate value required to grasp a certain object, then it closes to hold the object, then there's a pure trajectory for the end-effector of the robot without any change in the status of the gripper and finally, to place the object, gripper should be oriented accordingly and opens up to release the object.

To better visualize this concept, we'll use a moving average with a window size of 30 data point on the raw gripper values from the figure 5.6:

$$moving\ average = \frac{x_m + x_m - 1 + .. + x_{m-(n-1)}}{n} = \frac{1}{n}\sum_{i=0}^{n-1} x_{m-i} \tag{5.2}$$

Where n represents the window size.

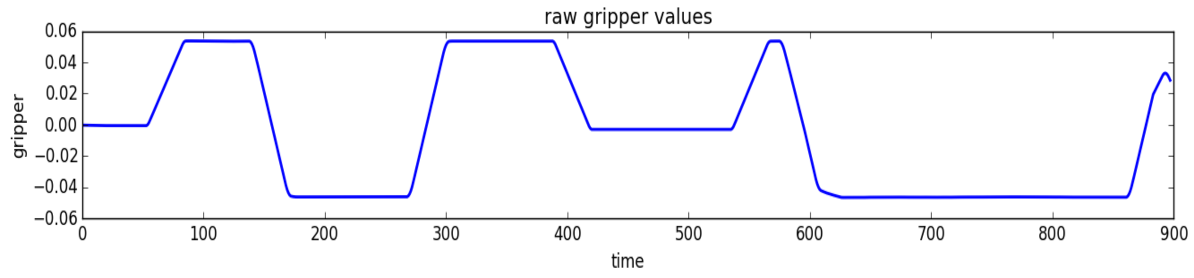

Figure 5.7. Running moving average over the gripper values to reduce the noise

As it can be observed in figure 5.7, 3 types of actions are distinguishable from this time series:

Type 1: Grasp action. It starts with some offset before the change in the gripper width and finishes when the gripper is closed.

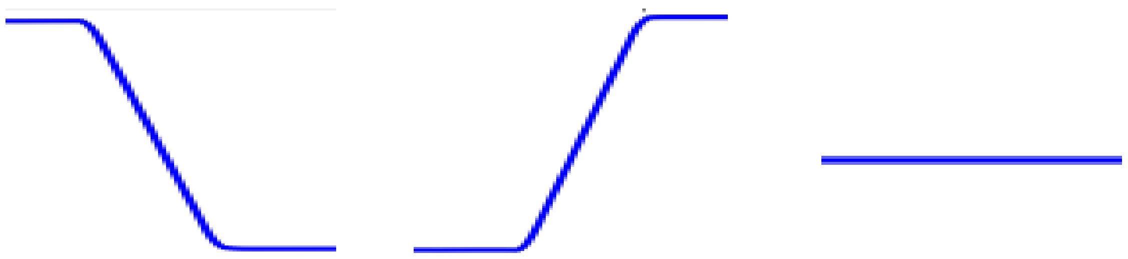

Figure 5.8. Grasp action (left), Release action (middle), Non-gripper action (right)

Type 2: Release action. It starts with some offset before the change in the gripper width and finishes when the gripper is opened to place the object in the target location.

Type 3: None gripper action. Where the width of the gripper doesn't change. This is the period where gripper is either holding an object or is simply empty.

Now that we defined the action types, we can use the first derivative of the gripper values to detect the places where the transition between different action types happens.

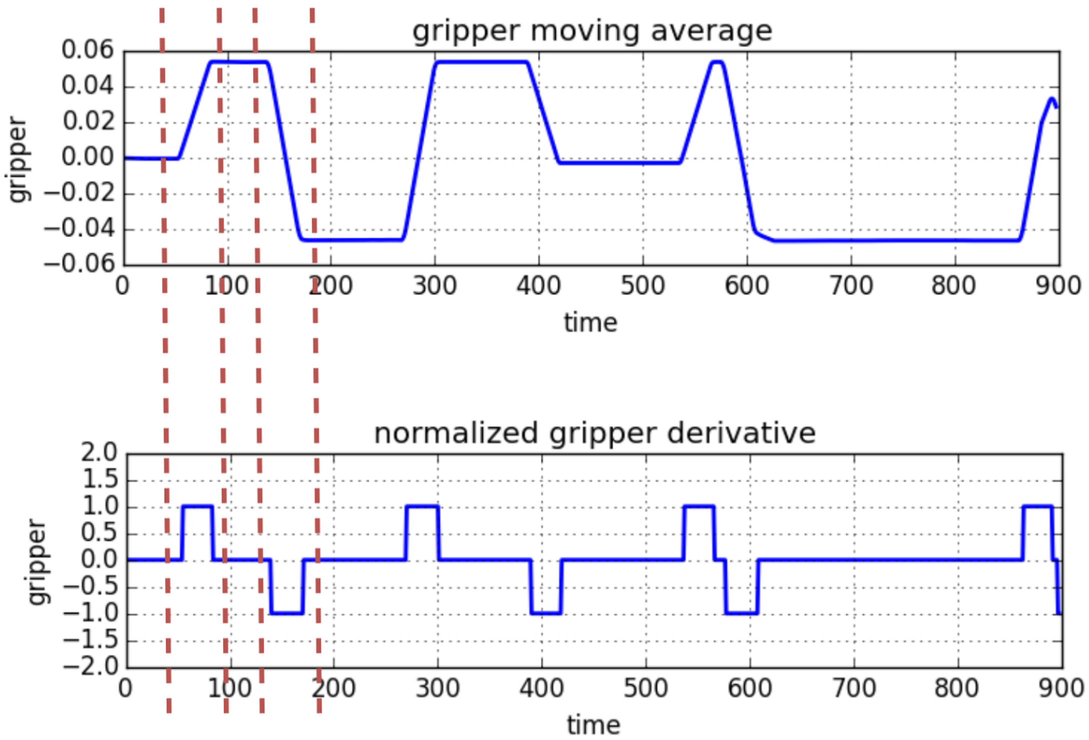

Figure 5.9. Derivative of the gripper values correspond to the defined action types

At this point, it seems that we can hand-code the segmentation process, however, we would like to train a Recurrent Neural Network with an LSTM cell to learn this segmentation. The reason will become clear in section 5.3.

The input to our network will be the combination of the extracted features from the stereo images using the encoder (5.1.2) and the raw pose values of the robot's end-effector.

Our network has the following architecture:

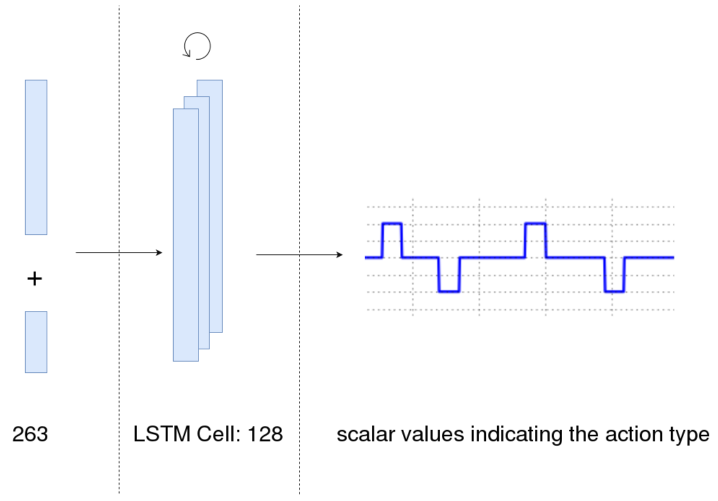

Figure 5.10. RNN trained to predict the action types based on the features extracted from the stereo images and pose values of the end-effector

A sample prediction of the action types is shown in the figure 5.11:

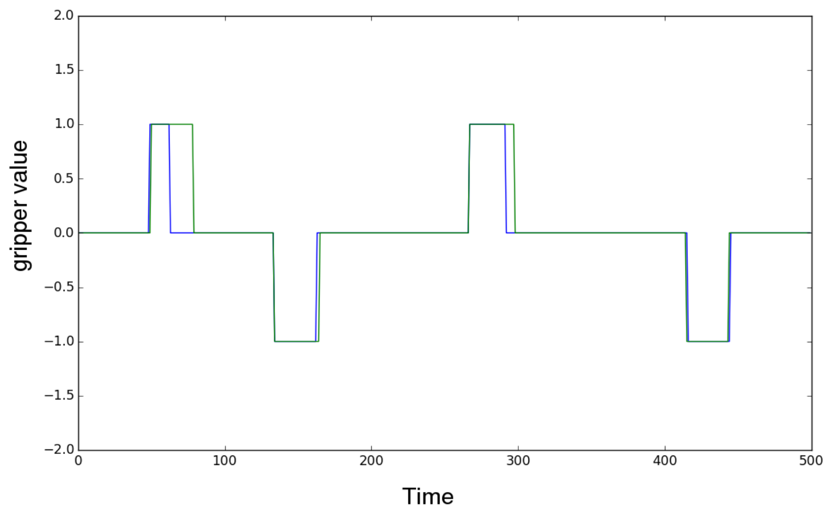

Figure 5.11. Prediction of the action type by RNN, Green line is the ground truth and the blue line represents the prediction

To train the network, predictions are passed through a *Softmax* (4.3). The cost function for training the RNN is the *categorical cross-entropy* as it's designed for the problem of multi-class classification. Concretely, it calculates the entropy difference between a given distribution *q(x)* and the target distribution *p(x)*.

5.2. Motion controller

Following the same intuition as the previous section, we would like to replace the traditional use of controllers such DMP with a more generic approach that also considers visual features in generating the motor command.

Previously we trained an encoder to extract a vector of features from the images, now using the similar approach as for the segmentation, we'll train another RNN that accepts the combination of the visual and kinematics features as the input and produces the robot's configuration as the output. Figure 5.12 shows the architecture of our RNN with one LSTM cell:

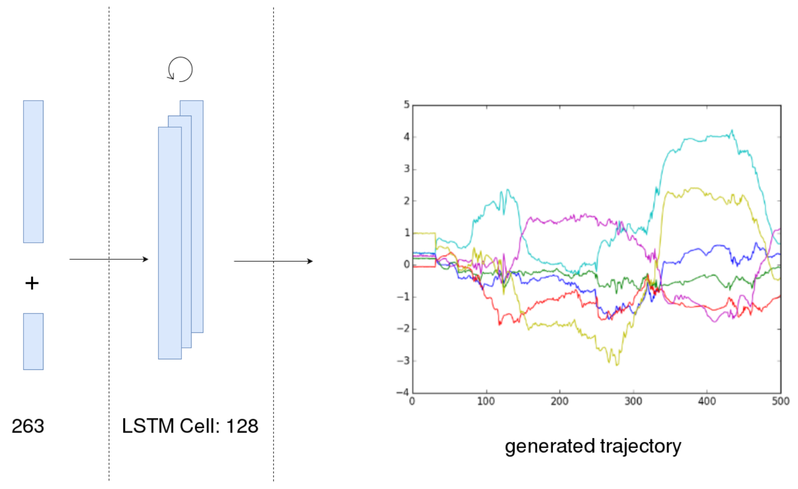

Figure 5.12. Architecture of the trajectory generator using a single LSTM cell

Figure 5.13 shows a comparison between the predicted values for a sample demonstration and the ground truth values recorded during the demonstration.

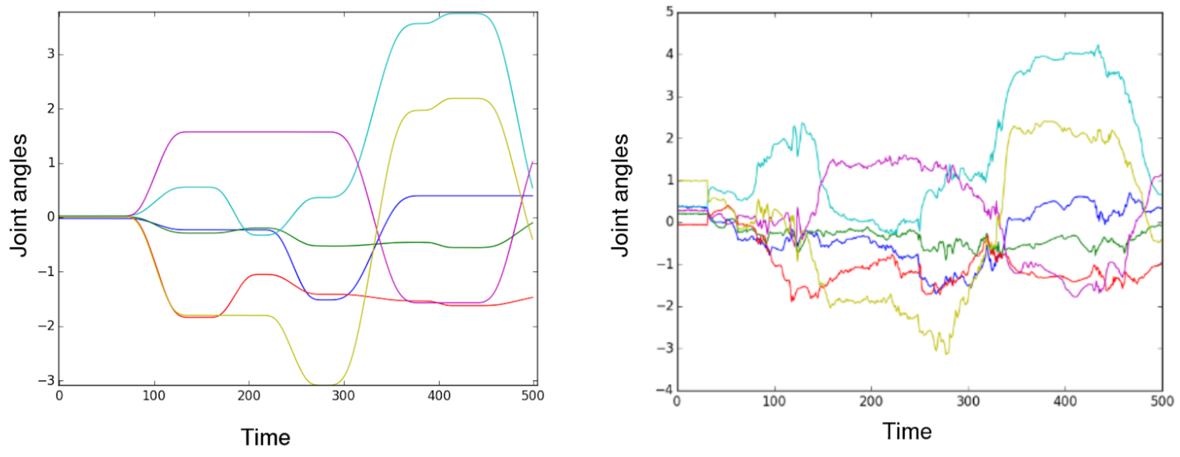

Figure 5.13. Comparison between the predicted and demonstrated joint values

We use a simple mean_squared cost function to train our RNN.

5.3. Combined segmentation and trajectory generation

Considering the similarities between the segmentation and controller models, we decided to combine the two networks together. As suggested in the figure 5.14:

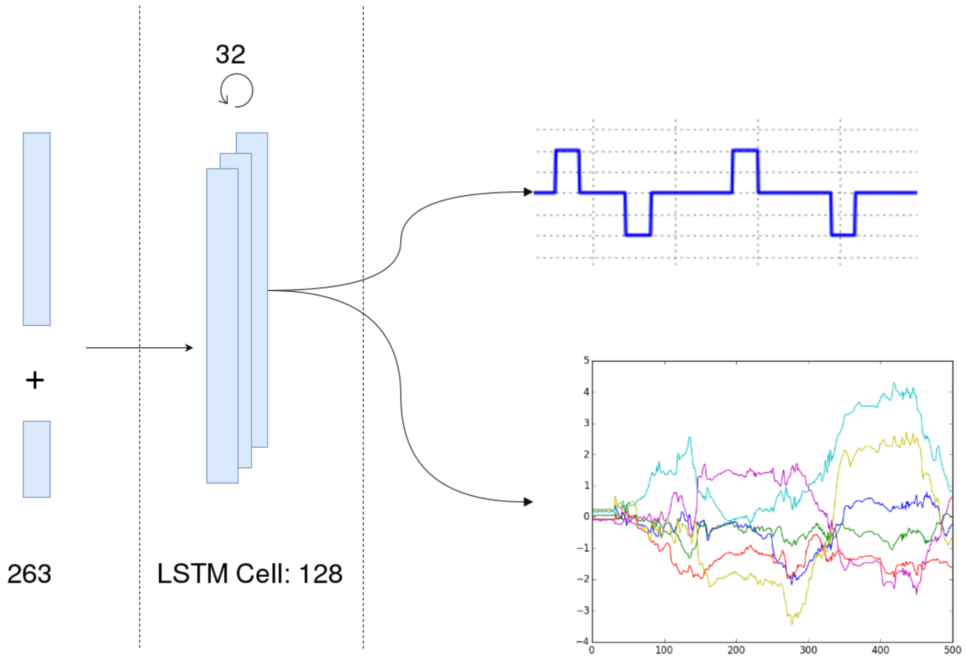

Figure 5.14. Combination of segmentation and controller RNNs

As discussed in chapter 2, the task of segmentation and learning the low-level controller for each segment has been traditionally separated. However, in the above model, the controller's accuracy improved by combination with the segmentation. The intuition behind this would be that the network will try to learn the features that are associated with the transition point and the trajectories at the same time which will results in the better prediction of the robot's configuration especially around the phase transitions.

To replay the demonstrated task, we pass a moving average filter with the window size of 10 over the predicted joint values to remove the high frequency noises.

In the algorithm 3, a summary of all the post processing steps to train the models is presented.

Algorithm 3: Segmentation and Trajectory generation

```
Post_Process (P, J, G, I):
        P: sequence of pose values of the robot's end-effector
        J: sequence of robot's joint values
        G: sequence of gripper joint values
        I: 2 x RGB + 1x D Images
1:      Update_encoder (I): update the stereo to depth encoder with a combination c
old and new images
2:      Extract_feature (I): pass all the images through the encoder to obtain their
corresponding feature vector E
3:      Calculate_normalized_gripper_derivative (G): transform the gripper values to
their corresponding action types and retrieve the transformed vector A
4:      Train_RNN (J, E, A, P): train the combined RNN model and return the controller
corresponding to the demonstrated task
```

Figure 5.15 shows the process of training the controller.

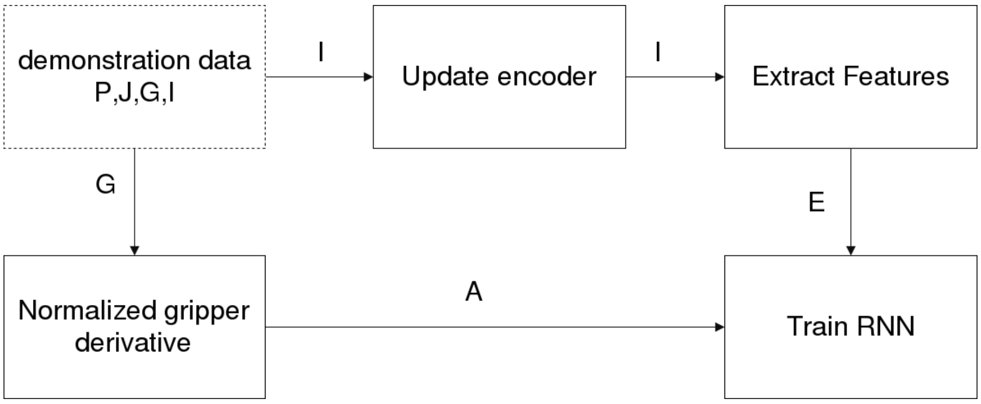

Figure 5.15. training the controller

5.4. Model execution phase

Here we demonstrate the steps taken by the model while attempting to replay the demonstrated task.

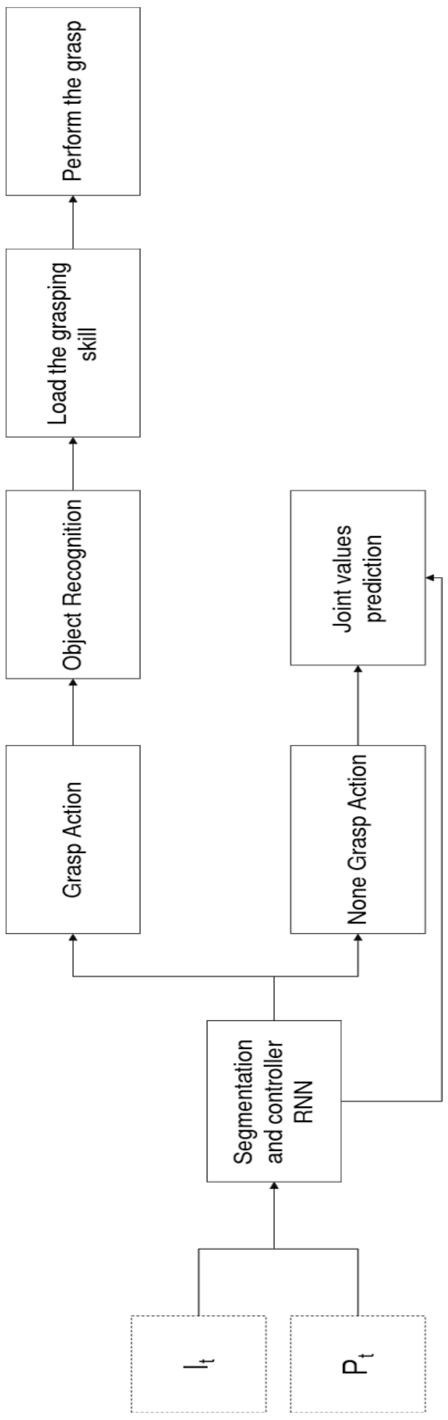

Figure 5.16. Execution phase

Chapter 6

Conclusion and Future work

In this work, we proposed a new framework for learning multi step object manipulation and in particular, pick and placing task from a handful of demonstration. To facilitate learning process we introduced the notion of action types and used them to separate the learning process of sub-skills. Our framework can extract visual features from the images that are recorded during the demonstrations and use them in combination with kinematics features to train the robot controller.

However, we targeted a simplified version of the problem. In particular, out of different action types, we only focused on the grasping, and grasp point prediction happened in 2D. Moreover, Trajectory segments could be separated and collection of robot maneuver skills could be learned independently of each other, which was not addressed in this work.

Above statement suggests that there are many ways in which this work can be continued and expanded. Here we mention few of them:

- To simultaneously benefit from capabilities of the simulation as a starting point to pre-train many robotic skills and enable transferring the knowledge to real world applications, there needs to be more research focused on the problem of transfer learning, or “Domain adaptation”. As discussed, in the image domain, one possible solution would be to use GAN models, where by feeding images from both simulation and real world, adversarial part tries to discriminate between these two domains while the generator part will learn to generate more realistic images from the simulated ones to fool the discriminator.
- Also, the performance of the GAN models in trajectory generation could be evaluated. In this scenario, by feeding both the generated robot configurations and the real data from the demonstration, the discriminator will try to separate these two domains while the generator will try to generate better IK solutions. But it is important to include the sequence analysis or recurrent connections into this structure considering the nature of trajectory as a time series.
- Trajectories learned by our RNN model could be splitted using their segment labels and learned independently. Later, concatenation of the learned robot maneuvers for different manipulation skills could suggest new solutions to novel scenarios.
- Reinforcement learning could be utilized to incrementally improve the performance of each module in our framework, by learning a policy on top of the model predictions.
- In the pile grasping, approaches such as [33] could be utilized to extract the pose of the object, in order to generalize the proposed solution in this work to 3D domain.
- Other gripper actions such as placing object could also be independently learned and used in the robot’s task execution.

- Approaches such as [7] could be used to incrementally improve the performance of object grasping from a pile.

As above suggests, there are still many problems to be solved before the results from this work can be used in the real world applications. We hope our suggested research directions will motivate more efforts in this field.

Chapter 7
Bibliography

[1] Niekum, Scott, Sachin Chitta, Andrew Barto, Bhaskara Marthi, and Sarah Osentoski. "Incremental Semantically Grounded Learning from Demonstration." *Robotics: Science and Systems IX*, 2013.

[2] Argall, Brenna D., Sonia Chernova, Manuela Veloso, and Brett Browning. "A Survey of Robot Learning from Demonstration." *Robotics and Autonomous Systems* 57.5, 469-83. 2009.

[3] Abbeel, Pieter, and Andrew Y. Ng. "Apprenticeship Learning via Inverse Reinforcement Learning." *Twenty-first International Conference on Machine Learning - ICML '04*, 2004.

[4] Holz, Dirk, Angeliki Topalidou-Kyniazopoulou, Jorg Stuckler, and Sven Behnke. "Real-time Object Detection, Localization and Verification for Fast Robotic Depalletizing." *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[5] Kroemer, Oliver, Christian Daniel, Gerhard Neumann, Herke Van Hoof, and Jan Peters. "Towards Learning Hierarchical Skills for Multi-phase Manipulation Tasks." *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[6] Sergey Levine, Peter Pastor, Alex Krizhevsky, Deirdre Quillen. "Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection". ArXiv:1603.02199v4, 2016.

[7] Abdeslam Boularias, J. Andrew (Drew) Bagnell, and Anthony (Tony) Stentz , "Learning to Manipulate Unknown Objects in Clutter by Reinforcement," Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI), 2015.

[8] Animesh Garg, Sanjay Krishnan, Adithyavairavan Murali, Florian T. Pokorny, Pieter Abbeel, Trevor Darrell, Ken Goldberg. "On Visual Feature Representations for Transition State Learning in Robotic Task Demonstrations", *NIPS Workshop on Feature Extraction, 2015.*

[9] Fox, Emily B., Michael C. Hughes, Erik B. Sudderth, and Michael I. Jordan. "Joint Modeling of Multiple Time Series via the Beta Process with Application to Motion Capture Segmentation." *The Annals of Applied Statistics* 8.3, 1281-313, 2014.

[10] Konidaris, George, Scott Kuindersma, Roderic Grupen, and Andrew Barto. "Robot Learning from Demonstration by Constructing Skill Trees." *The International Journal of Robotics Research* 31.3, 360-75, 2012.

[11] Finn, Chelsea, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. "Deep Spatial Autoencoders for Visuomotor Learning." *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[12] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. "End-to-end training of deep visuomotor policies". *J. Mach. Learn. Res.* 17, 1 , 1334-1373. 2016.

[13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller. "Playing Atari With Deep Reinforcement Learning." *NIPS Deep Learning Workshop*, 2013.

[14] E. Rohmer, S. P. N. Singh, M. Freese, ``V-REP: a Versatile and Scalable Robot Simulation Framework,'' IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 2013

[15] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. "ROS: an open-source Robot Operating System". ICRA workshop on open source software. Vol. 3. No. 3.2. 2009.

[16] Beeson, Patrick, and Barrett Ames. "TRAC-IK: An Open-source Library for Improved Solving of Generic Inverse Kinematics." *IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, 2015.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks". In *Proceedings of the 25th International Conference on Neural Information Processing Systems* (NIPS'12), 2012.

[18] Karen Simonyan, Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". ArXiv:1409.1556v6, 2015.

[19] Johns, Edward, Stefan Leutenegger, and Andrew J. Davison. "Deep Learning a Grasp Function for Grasping under Gripper Pose Uncertainty." *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[20] Lenz, Ian, Honglak Lee, and Ashutosh Saxena. "Deep Learning for Detecting Robotic Grasps." *Robotics: Science and Systems IX*, 2013.

[21] Lampe, Thomas, and Martin Riedmiller. "Acquiring Visual Servoing Reaching and Grasping Skills Using Neural Reinforcement Learning." *The 2013 International Joint Conference on Neural Networks (IJCNN)*, 2013.

[22] Girshick, Ross, Jeff Donahue, Trevor Darrell, and Jitendra Malik. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation." *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[23] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra*, "Continuous Control with Deep Reinforcement Learning"*, 2016.

[24] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long Short-Term Memory." *Neural Computation* 9.8: 1735-780, 1997.

[25] Zachary C. Lipton, John Berkowitz, Charles Elkan, *"A Critical Review of Recurrent Neural Networks for Sequence Learning"*. ArXiv:1506.00019., 2015.

[26] Volodymyr Mnih, Nicolas Heess, Alex Graves, Koray Kavukcuoglu, "*Recurrent Models of Visual Attention*" ArXiv:1406.6247, . 2014.

[27] Pastor, Peter, Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, and Stefan Schaal. "Skill Learning and Task Outcome Prediction for Manipulation." *IEEE International Conference on Robotics and Automation, 2011.*

[28] Jenkins, Odest Chadwicke, and Maja J. Matarić. "Performance-Derived Behavior Vocabularies: Data-Driven Acquisition Of Skills From Motion." *International Journal of Humanoid Robotics*, 2004.

[29] Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. "Learning attractor landscapes for learning motor primitives". In *Proceedings of the 15th International Conference on Neural Information Processing Systems* (NIPS'02), 2002.

[30] Djork-Arné Clevert, Thomas Unterthiner, Sepp Hochreiter, "*Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)"*. ArXiv:1511.07289. 2016.

[31] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio"*Generative Adversarial Networks"*. ArXiv:1406.2661. 2014.

[32] Atkeson, C. G., and, Schaal, S. "Robot Learning From Demonstration." Machine Learning: Proceedings of the Fourteenth International Conference. 1997.

[33] Changhyun Choi, Henrik I. Christensen, “3D Textureless Object Detection and Tracking: An Edge-based Approach,” in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.

Declaration:

I certify that this work contains no material which has been accepted for the award of any other degree or diploma in my name, in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text.
In addition, I certify that no part of this work will, in the future, be used in a submission in my name for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Stuttgart
.
Signature:
Stuttgart,