Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

R&D Project

Manipulating handles in domestic environments

Anirudh Narasimamurthy Jayasimha

Submitted to Hochschule Bonn-Rhein-Sieg,
Department of Computer Science
in partial fullfilment of the requirements for the degree
of Master of Science in Autonomous Systems

Supervised by

Prof. Dr. Paul G. Plöger
Alex Mitrevski

Jan 2020

I, the undersigned below, declare that this work has not previously been submitted to this or any other university and that it is entirely my own work, except for the following aspects, which were done jointly with the advisor.

Task	**Authors**
Perception dataset collection	Anirudh and Alex
Policy training in DoorGym environment	Anirudh and Alex
Collecting rollouts for MLE training on Toyota HSR	Anirudh and Alex
Collecting rollouts for testing MLE model on Toyota HSR	Anirudh and Alex

Date

Anirudh Narasimamurthy Jayasimha

Abstract

Robots that can learn to manipulate articulated objects has been a long-standing goal in the field of robotics. The robot must be able to perceive the contextual cues from its surroundings to accomplish different types of manipulation tasks. Handles are a commonly found articulated object found in domestic environments which can help the robot to access new areas in the environment. Early approaches for manipulating handles used hand-crafted features and reasoning which was purely based on the human insights of the environment. These methods are not generalizable to new environments or tasks and often work for a very constrained set of parameters. On the other hand, modern learning-based methods extract the required features and pattern from the data collected without human intervention and use them for generalizing over a wider range of environmental conditions.

The goal of this report is multifold: Firstly we have summarized the vast literature present for the task of "Manipulation of handles" and tabulated the details about the perception subtask, manipulation subtask and human-machine interface. The second major contribution concerns the detection of handles in domestic environments(pull/fixed handle, lever handle and round handle) which includes collection and annotation of a small custom dataset as well as training and comparing three deep learning models, FR-CNN, SSD and YOLO.

The third major contribution is prototyping two learning-based methods that can be used for solving the manipulation subtask. The first method involved porting the Toyota HSR robot to the 'DoorGym' simulation environment and using Reinforcement learning for obtaining a handle manipulation policy. The RL policy was trained for opening lever and fixed handles. It was observed that the simulated robot handles was able to grasp the handle but was not able to pull the door open correctly. The second manipulation method involves training a Maximum Likelihood Estimation model for maximizing the grasping success of different types of handles. The trained MLE model was run on a Toyota HSR and was observed to give a grasping success of 60% on a horizontal drawer handle and 40% on a vertical fridge handle.

Acknowledgements

I would like to thank my adviser Prof. Paul G. Plöger for sharing his valuable insights and reviews about my research topic and providing a wonderful intuition about the various mathematical concepts. I would specially like to thank my adviser Alex Mitrevski who played a big role in the successful completion of the report. His support in performing the experiments and the knowledge shared by him made the rigorous parts of the project go smoothly.

To my family, I would like to thank you for encouraging me to study hard and pursue my interests. It wouldn't have been possible without all your support in every aspect of life. To my friends, thank you for all the wonderful discussions we have had over the years and for bolstering my interests and hobbies.

Finally, I would like to extend my sincere gratitude to the university for providing the GPU Cluster and Toyota HSR, which gave us freedom to train and test many algorithms.

Contents

[1] Higher quality graphs in https://github.com/njanirudh/Research-Development-HBRS/

List of Figures

List of Tables

List of Abbreviations

NN	Neural Network
CV	Computer Vision
DL	Deep Learning
CNN	Convolution Neural Network
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
YOLO	You Only Look Once
SSD	Single Shot MultiBox Detector
FR-CNN	Faster Region-Convolution Neural Network
RPN	Region Proposal Network
PPO	Proximal Policy Optimization
A2C	Advantage Actor-Critic
PG	Policy Gradient
HSR	Toyota Human Support Robot
ROI	Region of Interest
RANSAC	RAndom SAmple Consensus
mAP	Mean Average Precision
LfD	Learning from Demonstration
HRI	Human Robot Interaction
VFH	Viewpoint Feature Histogram
NAF	Normalized Advantage Function

DMP	Dynamic Movement Primitives
RBF	Radial Basis Function
OID	Open Image Dataset
COCO	Common Objects in COntext dataset
mAP	Mean Average precision
mAR	Mean Average Recall
DoF	Degree of Freedom
MLE	Maximum Likelihood Estimation
W-MLE	Weighted - Maximum Likelihood Estimation
URDF	Unified Robot Description Format
MJCF	MuJoCo Format

List of Symbols

Symbol	Meaning
$\mathbb{Z}$	Set of all Integers
θ	Policy parameters
s *or* x	States
u *or* a	Actions
τ	Trajectory
$\mathbb{E}[X]$	Expectation of X
$p(X)$	Probability density of X
∇f	Gradient of a function (f)
$\{x_1, x_2..\}$	Set with elements $x_1, x_2...x_n$
$\pi(u\|x)$	Control policy
$Q^\pi(x, u)$	Q-Values of state,action pair
$V^\pi(x)$	Value function
$A(s, u)$	Advantage function
$\mathcal{Y}$	Sample array (MLE)
$\overline{\mathcal{Y}}$	Weighted sample array (MLE)
$\mathcal{L}$	Likelihood function (MLE)
$\mathcal{N}$	Normal distribution
μ, σ^2	Mean and variance of a normal distribution

1

Introduction

In robotics, the easy problems are hard and the hard problems are easy.

Moravec's Paradox

1.1 Motivation

The goal of robotics is to create artificial machines that can reach human-level performance for solving various tasks. There are many tasks that humans perform without much effort like perceiving the objects around us and extracting relevant and detailed features about them and using the observed object geometry and details to manipulate it. For robots each of these tasks requires large amount of prerequisite knowledge, human feedback and computational power. Inspite of the large amount of effort put into providing the prerequisite structure to the task, the end results are nowhere close to the satisfactory.

Manipulation of articulated objects is an important skill for robots that assist humans. Handles are a type of articulated objects that are found in most kinds of real-world domestic and industrial environments. Many of the common handles found in domestic environments are used to manipulate a connected object, which are usually doors or cabinets. The force that is applied to the handle is dependent on the kinetic model of the object like sliding door, prismatic cabinets, revolute doors and vertically opening cabinets. Ideally the robot must be able to manipulate the handle based on the best possible model that defines the underlying constrained object. Compared to manipulation of small objects, manipulation of handles poses many extra challenges from an engineering perspective. Compared to statically stable robots like four legged robots the forces during the interaction of the robot

with the door has far less influence when compared to the interaction forces when the robot has only two legs or wheels like humanoid robots[7]. Robots like Toyota HSR also have the unique problems of keeping the body of the robot stable when a pull or push force is applied during the handle manipulation.

Despite the extensive study of the task of manipulation of handles, it still remains an open problem with no single method that can work under all conditions. The vast variation in the type, size and orientation of model joints makes it infeasible to provide the models of all the possible handles. Apart from this, there are also cases where it might not be possible to estimate the best possible kinematic model by simply observing the handle. This may happen due to non-standard design of the handle and failure of the handle detection to classify correctly. Other major problem is that it is hard to compute a coordinated movement of both the base and the arm to manipulate the handle to open the door wide enough for the robot to navigate through it.

Humans have an unprecedented dexterity to perform manipulation tasks. We have a complex body structure that has a large DoF (twenty one) and the complex cognitive ability provided by the brain. Unlike robots, humans can also take advantage of the prior experiences to work in new and uncertain environments. A small child may be able to learn a complex manipulation skill by simply observing someone performing the relevant task once. For a robot to work autonomously in different conditions, similar to humans, it must be able to learn to solve the task at hand by exploring the different possibilities and exploiting its prior knowledge.

1.2 Problem Statement

This report is centered around the different types of methods that have been used for the task of detection and manipulation of handles, confined to those found in domestic environment. We also categorize the handle opening task along two subtasks [44] : the perception subtask and manipulation subtask. For each of the subtask, the methods that have been previously been used are collected and a brief summary of the advantages and disadvantages of using the method are given.

We use a dedicated dataset [5] for training multiple deep learning architectures and then use them on a Toyota HSR robot for detection of handles and doors in the robot camera frame. For the manipulation task we particularly look into learning based methods like Reinforcement learning and Maximum Likelihood Estimation. This is because learning based methods have better generalizability and can be more robust against small failures. Learning based methods can also be considered as a precursor for completely dexterous human like manipulation, which is the end goal of the study of AI.

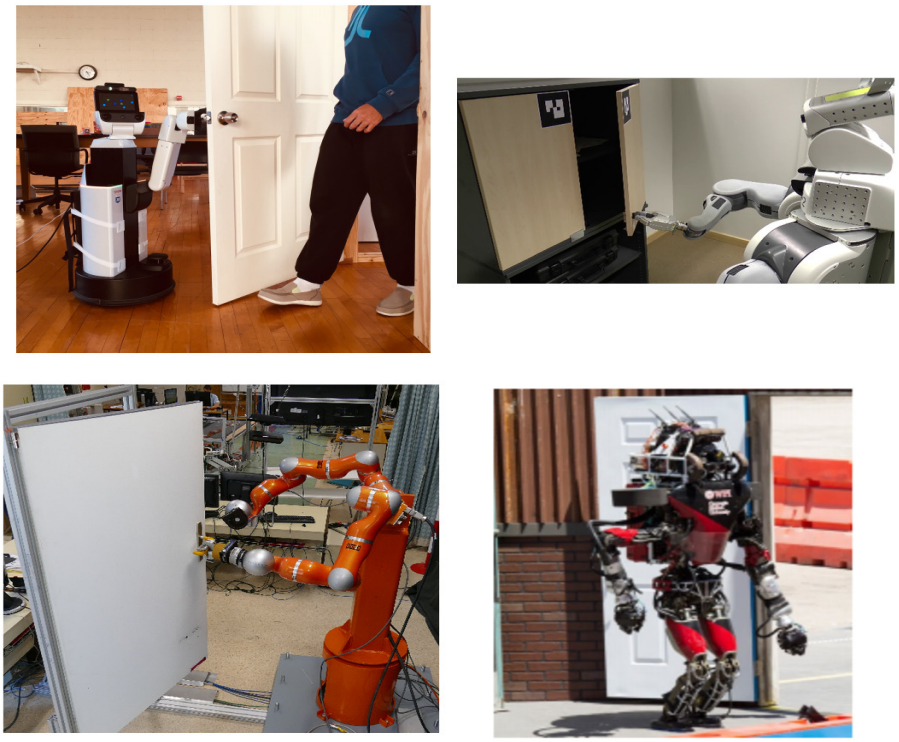


Figure 1.1: *top-left)* Toyota-HSR opening a door with round handle using deep learning for perception and Bayesian methods for manipulation[5] *top-right)* PR2 robot learning to open-doors using a single demonstration [17] *bottom-left)* Twin KUKA arms used to learn reinforcement learning policy for manipulating handles and open the door[46] *bottom-right)* ATLAS robot used for the task of opening doors in the DARPA Robotics Challenge using a gui based input from the user. [7]

Based on the study and the collected data about different methods, this report hopes to serve as a brief introduction for solving the task of manipulation of handles in domestic environment.

1.3 Report Organization

[chapter 2] of this report will present the general background description of the algorithms that have been used in the report. This includes a brief description of Deep learning and Convolution Neural Networks and how they are used for the task of object detection. The chapter further gives a brief introduction about Reinforcement Learning and MLE along with the associated terminologies and algorithms.

In [chapter 3] important state of the art methods are described are used for the specific task of manipulating handles according to previous literature. Along with the state of the

Figure 1.2: Different types of handles found in domestic environments. The image shows a fixed handle, a round handle and a lever handle [61]

art, the report will also broadly categorized the methods dealing with perception[section 3.1] and manipulation[section 3.2] task and state the advantages and disadvantages of the methods. The perception methods are sub-classified based on whether 2D, 3D image or point-cloud data is being used and the type of algorithm(learning or classical)used. Similarly the manipulation methods are divided into control based, reinforcement learning based, compliance based or few novel methods that use graph optimization and fuzzy logic.

The next chapter, [chapter 4], describes the general pipeline of the handle manipulation task by providing a state flow diagram of each of the subtasks involved. Introduction about the advantage of learning based methods for object detection is given along with a brief description of the architectures of perception models used by us for detecting handles in the camera frame. In manipulation, the reason for selecting reinforcement learning based methods over conventional control based methods are given along with the steps involved in training a policy in the simulation environment is also provided. We also give the experimental setup and description for collecting data for MLE.

[chapter 5] expands on the steps performed by us for training the deep learning models and provides the metrics for analysing them. Similarly for the policy training method, the steps to train a policy in the DoorGym simulation environment is given along with the relevant training metrics. The result obtained by the MLE experiment is also provided.

[chapter 6] will enumerate the main contributions of this report along with the possible future works to expand and improve the pipeline.

2

Background

A science is any discipline in which the fool of this generation can go beyond the point reached by the genius of the last generation

Max Gluckman
Politics, Law and Ritual, 1965

The theory behind perception and manipulating have a vast array of literature of both learning and non-learning methods. This chapter of the report will be giving a very brief background about the various techniques used in the pipeline given by this report for the task of "Manipulation of Handles".

2.1 Neural Network

Numerical tasks like multiplying two 6-digit numbers are very difficult for humans to perform but are very easy for computers to perform. On the other hand many of the intuitive tasks performed by the humans require lot of knowledge about the world and are very difficult or nearly impossible for the machines to perform. The most important steps for many real world problems involves representation of the knowledge of the domain into simpler or abstract information (called as *features*) which computers can perform computations. This task is called as *feature representation*[22].

Classical computer vision methods involved the use of hardcoded features for tasks such as object detection and recognition. These methods were not scalable for complex task due to the difficulty of mathematically formulating many of the complex shapes and features

that are found in the nature. Neural networks solved the problem of feature representation by using a hierarchy of simple features to denote complex hierarchy of features. The neural nets can be understood as complex universal function approximators that are able automatically learn new features from the data provided to it, and use them to solve the task. In CV tasks the neural networks are able to learn the simple edges, corners, lines etc and use them to build complex concepts like contours.

2.1.1 Convolution networks

Convolution networks or CNN are special kinds of neural network architecture that are use to process grid-like topological data [22], like 2D image or 1D time series data. Convolution operation on images are linear mathematical operations that involves the multiplication of a pixel and its neighboring pixel of an image by a special matrix called as the *kernel*. The outputs of the kernel operation on the full image is called as *feature map*.

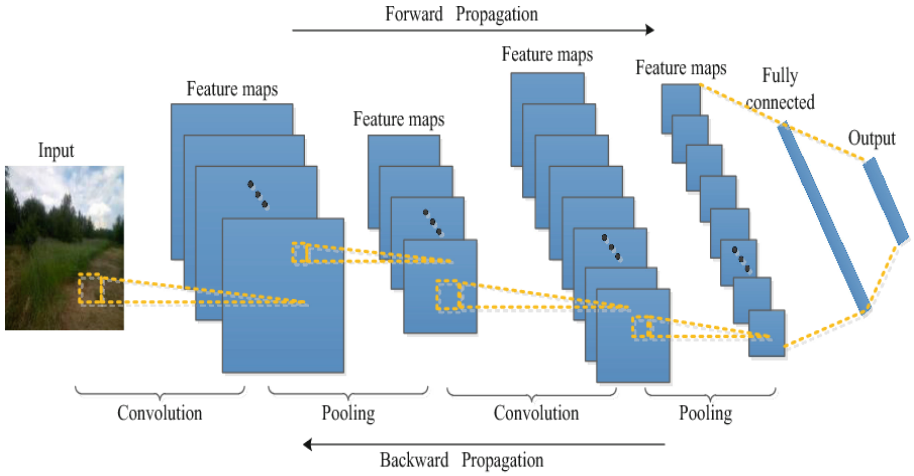


Fig. 1. The CNN structure

Figure 2.1: A basic Convolution Neural Network model showing the convolution, pooling, fully connected and output layers. The forward propagation is used during inference and to know the error whereas backpropogation is used to propagate the errors for training [65]

.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.1}$$

The above equation denotes a 2D convolution operation applied on an input image 'I' and and 2D kernel 'K'. CNN use many such kernels of different sizes and values to

extract different types of edges and corners from the images to generate complex hierarchy features. The layers that performs the convolution operation on the input data is called as *convolution layer*. In some models there are special types of convolution that are used called as Atrous convolution or Dilated convolution. Artous convolutions uses zeros between the non-zero points in the filter to increase the overall receptive field without increasing the computational complexity by maintaining the spatial dimension. But due to the extra zeros in the filter, Atrous filters require more memory.

Another layer that is present in CNN are called as the *pooling layer*. The pooling operation involves modifying the individual outputs obtained from the previous layers based on the values of the neighboring output points. There are many types of pooling layers based on how the input is modified. *max pooling* returns the maximum values based on the kernel size. Other common types of pooling layers are *average pooling*, L^2pooling. The main use of the pooling operation is that it helps in making the features invariant to small local translations in the input. Pooling with a non-single stride can also help in down-sampling the input image to reduce the overall number of computational steps in the subsequent layers.

Final layer of CNNs in the task of multi-class object recognition is called as *softmax layer*. Softmax layer assigns a probability to the possible classes based on the weights that are activated due to the input. Object detection networks like YOLO, SSD , FR-CNN also give the bounding box of the object along with the class probabilities in the last layer.

2.2 Point-cloud processing

Point-cloud data consists of a point data(x,y,z) in a 3D coordinate space. The stereo cameras, RGB-D and Lidar gives the output of the perceived environment in the form of a point-cloud or in the form of an 4 channel(RGB-D) image which can be converted into a point-cloud. Since point-cloud data is usually sparse and noisy due to hardware(lens distortion, sensor noise, etc.) and environment constraints, there are many preprocessing steps that are required to perform object detection tasks on it. Due to the many types of preprocessing and object detection algorithms that are used for point-cloud data, only a brief description of the common methods will be be given. An extended review of the various methods are beyond the scope of this report.

Neighborhood-methods of noise detection involves the use of similarity metrices between each point and its neighbors to remove unwanted noise from the point cloud. The most common filter called as the bilateral filter.

2.2.1 Finding planes in point-cloud data

Finding a plane in the point-cloud is very for a wide variety of tasks like object surface construction, segmentation and object detection. RANdom SAmple Consensus (RANSAC) algorithm is one of the most common methods of model fitting in computer vision [68] that are used due to the simplicity of algorithm and its robustness in the presence of a lot of outliers. Unlike the least squares methods for parameter estimation where all the data is considered for fitting the best model[18], RANSAC methods remove the erroneous data before fitting the model.

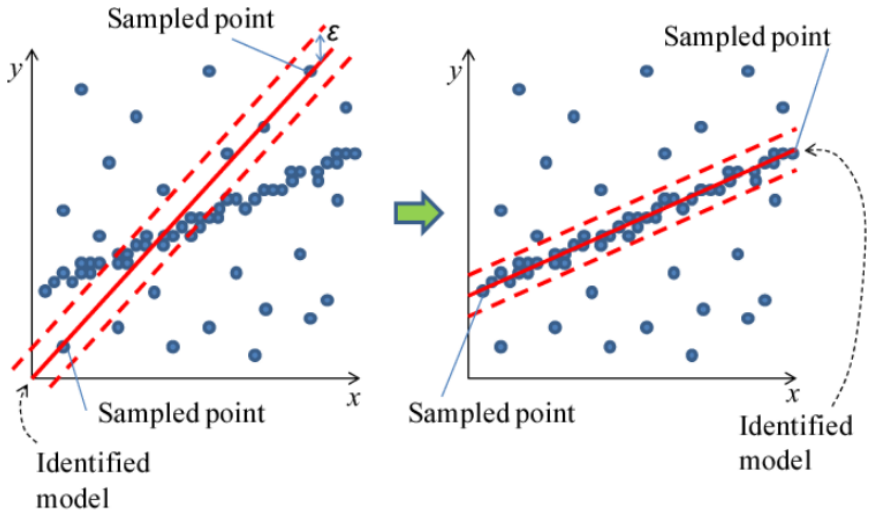


Figure 2.2: Image showing the conceptual steps of finding the best fitting model in the data[62]

.

The general steps in the RANSAC algorithm for plane detection involves the selection of a small number of points (Usually 4 for plane detection) from the obtained point-clouds and use that to fit a 2D-plane. For the model obtained we count the number of outliers and the inliers that lie within a given threshold distance. We continue the iteration until the fraction of the number of inliers exceeds an approximate threshold. There are many variants of the RANSAC algorithm based on the how the points are sampled and the verification of the model. For further details about the different RANSAC techniques [50][13] can be referred.

For the task given in this report the main use of RANSAC is to find doors in the environment of the robot.

Algorithm 1 RANSAC algorithm for finding planes in point-clouds[19]

```
// PointCloud is the set of all input 3D points
// MaxIter is the maximum number of iterations to run
// Threshold distance within which the point is considered an inlier
// K max points to assert best fit model
```

Input: *PointCloud*, *MaxIter*, *Threshold*,*K*

Output: *Inliers*,*Outliers*

Data: PlaneModel

while *not MaxIter* **do**
  Draw *S* from *PointCloud*;
  Fit *S* to *PlaneModel*;
  if *other point lies within Threshold* **then**
    Add to *Inliers*;
    if *Inliers* $\geq K$ **then**
      return *Best fit Inliers*
    else
      Continue with new sample S
    end
  else
end

2.3 Reinforcement learning

Reinforcement learning is considered as the third paradigm of machine learning apart from unsupervised and supervised learning methods. Reinforcement learning makes use of trials and error methods to interact with the environment and provide the most optimal solution. Sutton and Barto [59] defines Reinforcement Learning as "learning to map situations to actions to maximize a numerical signal". Reinforcement learning is a goal driven approach to solving a problem.

2.3.1 Markov Decision Process (MDP)

Reinforcement learning can be mathematically formulated as a markov decision process. A general markov decision process can be given as a set of states, action and transition operator that gives the probability of going to s_{t+1} from state s_t. The reward function gives a single scalar value as feedback from the environment.

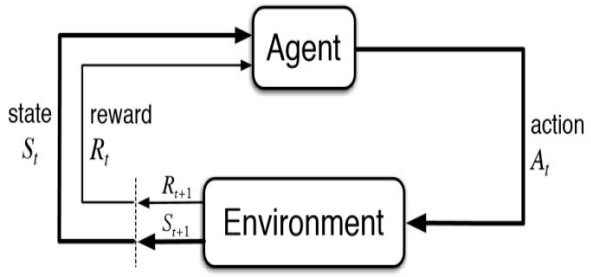


Figure 2.3: State flow diagram of a single iteration of a reinforcement learning cycle [59]
.

$$M = \{S, A, T, R\}$$
$$S \rightarrow \text{State space}$$
$$A \rightarrow \text{Action space}$$
$$T \rightarrow \text{Transition operator}$$
$$R \rightarrow \text{Reward function}$$

2.3.2 Important terminology

In robotics the states space of the RL problem comprises of internal states of the robot and the external states of the robot. Internal states of the robot includes the joint-angle, joint-velocity, pose of the end effector, the orientation/position of the body of the robot. The action space on the other hand denotes all the possible motor controls that the robot can perform.

A Reinforcement learning problem may be defined in terms of four major sub-elements as explained by Sutton and Barto [59] :

- *Policy* : RL uses a policy π to select the best possible action to select for a given state. The policy can be either a deterministic policy where there exists one particular action for each state $u = \pi(s)$ or a stochastic policy where the action is a probability over the state $\pi(u|s)$.

- *Reward signal* : The *reward* is a single scalar value that we get at each time-step from the environment. The reward signal determines the final goal of the task. The

reward may be positive or negative depending on the desirability of the action taken or the states reached.

- *Value function* : Unlike a reward, which indicates the desirability of a state at an immediate time step, value specifies the desirability of the state in the long run. In general terms value of a state can be defined as the accumulated reward over a long term starting from the present state. Although rewards are the primary reason for selecting a state, values are the one that are seen by
- *Model of the environment* : The formulation of the behaviour of the environment of the agent is called as the model. Given a set of state and action the model can help in predicting which is the next state.

Many major problems in robot manipulation can be naturally formulated as an RL problem. Problems in robotics are often very high dimensional (5 dimensional for simple robots to over 30 dimensional for humanoid robots) and deals with continuous actions and states . Since real world in often noisy due to the effects of environment and the sensors it is very difficult to precisely know the exact state the robot is in.

Many of the tasks in robotics can be considered to be *episodic* in nature. ie. The tasks start at an initial state and have a terminal state called as a trajectory. The trajectory is referred as a single trial containing a set of actions and states and is denoted by τ. A single trajectory or path is also referred to as a *rollout*.

$$\tau = (s_0, u_0, s_1, u_1 ...) \tag{2.2}$$

For a single rollout the accumulated reward over the states and actions is given as $R(\tau)$:

$$R(\tau) = r_T(x_T) \tag{2.3}$$

where r_t is called as the instantaneous reward , r_T is the final reward that might depend on the final expected goal position. The t in the equation indicates the time step and the time-dependence of the policy can be given by $\pi(u_t|s_t, t)$ for stochastic policy or $u_t = \pi(s_t, t)$ for deterministic policy. The reward functions also has a term to take into consideration the history upto which the instantaneous reward is considered called as *Discount factor* (γ) where $\gamma \in [0, 1)$.

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r(s_t, u_t) \tag{2.4}$$

A discount factor closer to 0 indicates that the reward considers the more recent state.

As the discount factor goes towards zero, the previous states are considered. Due to the formulation of the problems in robotics, the task in RL can be considered as selecting as locally optimal policy (π^*) that maximizes the expected reward accumulated over the trajectory.

$$J_\pi = [R(\tau)|\pi] = \int R(\tau)p_\pi(\tau)d\tau \tag{2.5}$$

where the reward ($R(\tau)$) is the objective of a given task and $p_\pi(\tau)$ defines the probability distribution over the trajectories.

Function-approximator in RL should support realtime learning as the robot interacts with the environment. This means the function approximator should be able to learn incrementally using the obtained data.

Value Function RL methods that are based on using the Value Function $V_t^\pi(s)$ estimate the reward associated with each state (s) at a given time step (t). This value function extended to consider both the current state and the action is called as the *action-value* function or the *Q-function*. This score is denoted by $Q^\pi(s, u)$.

$$V^\pi(s) = \mathop{\mathbb{E}}_{\tau}[R(\tau)|s_0 = s] \tag{2.6}$$

$$Q^\pi(s, u) = \mathop{\mathbb{E}}_{\tau}[R(\tau)|s_0 = s, u_0 = u] \tag{2.7}$$

Usually, the policy which returns the highest expected value is considered. The locally optimal policy $\pi^*(a|s)$ can be considered to be the optimal policy [59].

$$V^*(s) = \max_\pi V^\pi(s) \tag{2.8}$$

$$Q^*(s, u) = \max_\pi Q^\pi(s, u) \tag{2.9}$$

In some of the algorithms there is no need to get precise score representing how good an action is, only a relative score with respect to another action is enough. For this reason we get a scalar score called as the *advantage function*. The advantage function acts an variance reduction technique for the policy gradient.

$$A^\pi(s, u) = Q^\pi(s, u) - V^\pi(s) \tag{2.10}$$

Value functions are used to assess how good is it to perform an action u in a given state s. Value function based method require that a reward be associated with each set of

state and action (s, u) for the entire state-action space. This is practically impossible due to the high dimensional nature of the problems found in robotics these methods are not not commonly used. Also the discontinuity in the manifold of the value function leads to the non-optimal policies in continuous search-action space problems.

Unlike classical RL problems where no prior information is given about the agent environment and the agent is expected to explore its environment to learn the most rewarding states. In robotics, the RL task cannot be formalized without a prior as it can lead to a potentially dangerous situation when the robot is exploring its environment.

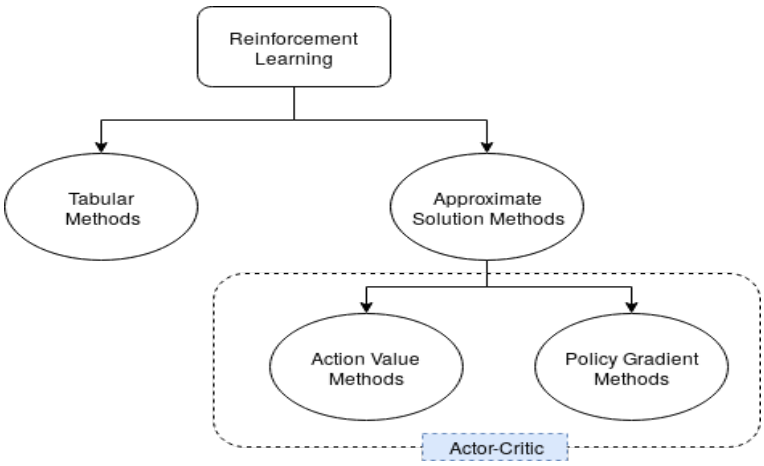


Figure 2.4: Broad classification of Reinforcement Learning methods [59]

.

2.3.3 Policy Search

Policy Search Policy are task appropriate pre-structures required to solve a specific problem. Policy Search (PS) methods help in simplifying many of the drawbacks associated with the higher dimensional problems found in robotics[33] by using parametrized policies $(\pi(\theta))$. It does this by reducing the effective search space for finding the optimal policy. PS method deals with finding the best parameters θ in the parameter space Θ ie. $\theta \in \Theta$. The parameter space of the policy can be further reduced by selecting a good initial estimate of the parameters of a policy by using imitation learning or learning from demonstration. Mathematically speaking, these methods learn the values of policy parameters based on maximizing a performance measure $J(\theta)$ with respect to the policy parameters θ.

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t) \tag{2.11}$$

Methods that are used to learn both the policy and the value functions using optimisation come under the wide group of algorithms called as Actor-Critic methods [59].

Policy search methods can be broadly classified into two different types based on how the policy is learnt. Model-free policy search methods sample trajectories τ from a real robot and use that to train the best policy. Since real robot hardware is involved to collect the trajectories, there are many major disadvantages of model-free policy search. The time to perform a rollout is longer than using a simulator. The physical wear and tear of the robot may lead to unexpected noise in the captured trajectory. Model-based policy search methods uses a few robot trajectories to learn the forward model[1][14] and the environment of the robot first. The newly learnt models are then used in the simulator to sample the trajectories efficiently. Although model based methods are helpful in addressing the problem of sample inefficiency associated with model-free methods, the task of learning a precise model of the robot dynamics is a very difficult task.

One of the main challenges in policy search methods is the selection of the step-size for changing the policy. A very small step size prevents the policy from learning the required behaviour whereas a large step-size may cause the major change in the policy and can lead to a catastrophic loss in the performance of the policy.

Policy representation

The representation of the policy is an important part of policy search methods as it determines whether the policy will converge without problem and whether it can be used for a given task. Few of the major requirement of when selecting the policy representation when using RL[34] in robotics are : smoothness of the policy to encode a continuous and smooth trajectory, enable safe exploration of the parameters, scalablity of the policy to very high DoF task eg. 30-50 DoF are common in modern humanoid robots, robust against small errors, enable incorporation of regularization parameters, should be invariant towards rotation and scaling.

Representation of policy can either be time-independent policy $\pi_\theta(s)$ or time-dependent policy $\pi_\theta(s,t)$. Time-independent policy uses only a single policy to represent the trajectory at all time steps. This means the policy requires a more complex parametrization compared to time-dependent policies where the policy is different for each time step so simpler policies can be used. Policy may also be either deterministic $\pi_\theta(s,t)$ or stochastic $\pi_\theta(s,t) + \epsilon_t$ where ϵ_t represents a noise vector[29].

In Robotics the policy may be represented in a few different ways depending on the

[1]Forward model is the influence of an action performed by a robot on its state and its environment

complexity of the task at hand.

Linear Policies Linear policy is a time independent representation that is a linear function of all the parameters $\theta_1, \theta_2, \theta_3 ... \theta_n$[33]. Since a purely linear model is very constraining in the functions it can model, it is extended by using "linear combinations of fixed nonlinear functions of the input variables" [8]. The nonlinear functions are called as the basis functions $\phi(s)$.

$$\pi_\theta(s) = \Theta^T \phi(s) = \sum_{j=0}^{N-1} \theta_j \phi_j(s) \tag{2.12}$$

Although the basis function $\phi(s)$ add non-linearity, the function is linear in θ, so are called as linear policy. These policies are used in very limited problems as the basis function have to be manually given.

Radial Basis Functions (RBF) Radial basis function use basis to get an unknown function from the given set of data points. The approximated function ($f(s)$) is expressed as a linear combination of many basis functions where each is centered on a data point. [8]

$$f(s) = \sum_{n=1}^{N} w_n h(\|s - s_n\|) \tag{2.13}$$

Dynamic Movement Primitives (DMP) DMP is a non-linear, second order differential equations that are used to represent a motion trajectory. Representing the trajectory in the form of a differential equation has the advantage of being model free and has self-correcting property against small perturbations[47]. The complete set of properties of DMPs can be found in [26].

$$\ddot{y} = \alpha_y(\beta_y(g - y) - \dot{y}) \tag{2.14}$$

Where y = System state α_y and β_y are the gain terms g is the final goal position

A non-linear forcing term (f) is added to the equation so that the trajectory can be modified as required and allow generation of complex movements. This non-linear term is also called as *canonical dynamic system.*

$$\ddot{y} = \alpha_y(\beta_y(g - y) - \dot{y}) + f \tag{2.15}$$

The non-linear function is given as a normalized linear combination of basis functions (ψ).

$$f(s) = \frac{\sum_i w_i \psi_i(t)}{\sum_i \psi_i(t)} \tag{2.16}$$

To remove the time dependence of the above function, the equation is replaced by a first order dynamics in terms of the state(x) such that.

$$f(s) = \frac{\sum_i w_i \psi_i(s)}{\sum_i \psi_i(s)} x(g - y_0) \tag{2.17}$$

Where g denotes the goal, ψ represents the Gaussian basis function and the w is the weight of each basis function. The x is a continuously diminishing term which makes sure that the equation converges to a point attractor form of the equation as the goal is reached [26]. Equation for a Normal distribution $\mathcal{N}$ is given as following

$$\mathcal{N}(x : \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(\frac{-(x-\mu)^2}{2\sigma^2}) \tag{2.18}$$

Comparing the equation of the normal distribution(2.18) with the basis function (**??**) we can consider 'c_i' to be the mean or the centre of the basis function and 'h' to be the variance of the basis function.

The DMPs also have the properties of being *spatially scalable* and *temporally scalable*.Spatially scalable means that the once a trajectory is learnt by a DMP it can be scaled to reach goals that are closer or further away. The term Temporally scalable means that the speed of the trajectory can be changed by modifying the equation by applying a timescaling variable τ such that at $\tau = 1$ the equation motion primitive is learnt at the same speed as the imitation :

$$\ddot{y} = \tau^2(\alpha_y(\beta_y(g - y) - \dot{y}) + f) \tag{2.19}$$

The formulation of the DMP equation includes the 'g' or the goal term that help it to generalize for different final positions. DMPs can also be used to encode complex motions by sequencing multiple DMPs. For learning very high performance DMPs for a trajectories, both supervised learning and reinforcement learning can be used[56]. The parameters to be optimized for the DMP is given by the weights (w_i) of the basis function. Supervised learning involves starting with am initial correct trajectory to reach the goal and finding the w_i of DMPs in each dimension by using locally weighted regression. In case of the RL approach rewards are used for following the best trajectory and random explorations are

used to find the best possible parameters are used for finding the task [32].

Artificial Neural Network (ANN) Neural Networks are universal function approximators that can be used for denoting various complex functions. The inputs can be used to non-linearly map the input state and action to next state to go to or the next action to take. The ANNs can also be used to learn the value or the Q function from the input states and actions.

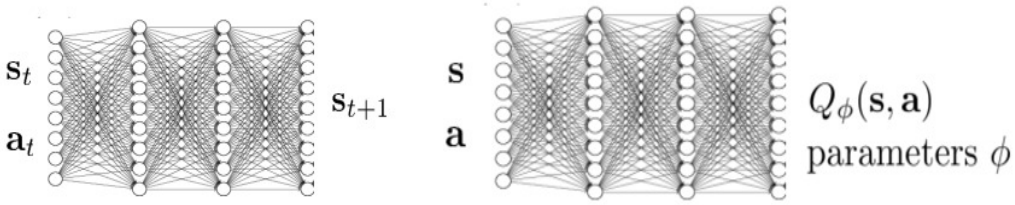


Figure 2.5: 1) First image shows the mapping of a pair of action and state (s_t, a_t) to the next best state (s_{t+1}) 2) The second images show the mapping of a pair of state and action (s, a) by a neural network to the Q value.[1]

Many methods use CNNs to map an image to a particular action. The image can be considered as a given state or observation.

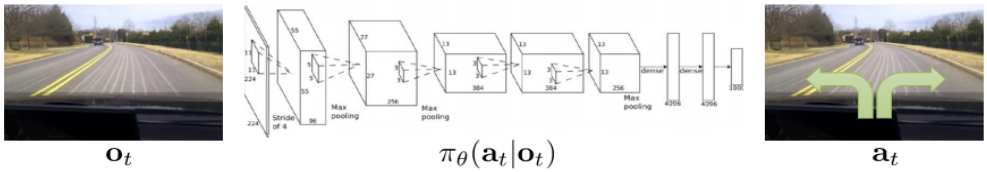


Figure 2.6: Image shows the mapping of an observation (o_t) which is in the form an image at time step(t) to an action (a_t) using the CNN as the policy $(\pi_\theta(a_t|o_t))$ and the weights of the network are the parameters that are learnt.[1]

Policy Gradient Methods

REINFORCE

REINFORCE or the Monte Carlo policy gradient is an extension to the policy gradient algorithm that makes use of gradient ascend to find the best parameters for the policy.

From the policy gradient theorem the gradient of the objective function (J) with respect to the parameters (θ) is given by the equation

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \log P(\tau|\theta) R(\tau)] \tag{2.20}$$

Trust region policy optimization (TRPO)

Compared to methods that use a linear search for finding the best direction of improvement, TRPO uses a trust region for finding the best improvement in the region. The main idea behind trust region method is to optimize a surrogate objective function instead of the total objective function. The surrogate objective function is an approximation of the objective function within a small region called as the trust region.

The optimization of parameters in the parameter-space might cause a drastic change in the learnt policy. TRPO solves the problem of selecting the best step size by selecting parameters constrained by the KL divergence between the old policy and the new policy during each iteration of the update.

$$\mathbb{E}[D_{KL}(\pi_{\theta_{old}}(.|s)||\pi_\theta(.|s))] \leq \delta \tag{2.21}$$

Where δ is a threshold distance parameter.

Proximal Policy Optimization (PPO)

PPO is an on-policy method which can be used for both discrete and continuous action spaces. PPO tries to solve the same issues as TRPO in finding the best step selection for the parameters without causing a major change in the performance of the policy. Unlike the KL divergence used in TRPO for comparing the new and old policy, PPO makes use of a clipping term to make sure that the new policy doesn't cause a major performance collapse.

Algorithm 1 PPO-Clip

Input: initial policy parameters θ_0, initial value function parameters ϕ_0
for $k = 0, 1, 2, ...$ **do**
  Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
  Compute rewards-to-go $\hat{R}_t$.
  Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
  Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

  typically via stochastic gradient ascent with Adam.
  Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

  typically via some gradient descent algorithm.
end for

Figure 2.7: PPO pseudo-code [4]

2.4 Maximum Likelihood Estimation

Maximum Likelihood Estimation is a statistical method for approximately estimating the parameters of a probability distribution to increase the probability of the observed data in the model.

Given a set of samples $\mathcal{Y} = [y_1, y_2, y_3, ...]$ and the parameter vector of a distribution $\theta = [\theta_1, \theta_2...]$, $(\mathcal{Y})$ can be used to optimize the likelihood function $(\mathcal{L})$ over the parameter space.

$$\hat{\theta} = \arg\max \mathcal{L}(\theta; \mathcal{Y}) \tag{2.22}$$

The specific parameters that are obtained that models the observed data most precisely is called as the maximum likelihood estimate $(\hat{\theta})$.

Considering that the underlying distribution is a normal distribution, $(\mathcal{N})$ the parameters of the gaussian distribution are given by $\theta = \{\mu, \sigma^2\}$. The likelihood function can be given as

$$\mathcal{L} = p(Y|\theta) = \mathcal{N}(\mathcal{Y}|\theta) \tag{2.23}$$

$$\mathcal{L} = \mathcal{N}(\mathcal{Y}|\mu, \sigma^2) \tag{2.24}$$

3

State of the Art

The science of today is the technology of tomorrow

Edward Teller
Theoretical Physicist

This chapter aims to expand on the vast literature present and the various types of algorithms that have been used previously to solve the task of manipulating handles. We will systematically review the methods used for perception, manipulation and the various types of human guidance involved for successfully completing the task addressed by this report. This chapter will also contribute by providing a broad perspective of all the possible ways of approaching each of the subtask while also providing their strengths and weaknesses.

The section 3.1 provides a general overview of what is meant by perception and the different subtasks that are involved in perception with respect to the task in the report. The subsections subsection 3.1.1 and subsection 3.1.2 will provide the previous works about the object detection and pose estimation subtasks respectively. The next section section 3.2 expands on the various types of methods used to manipulate the handle. These methods range from control based methods to learning based methods. The last section section 3.3 will briefly provide the general ways in which human guidance has be used for helping the robot to improve its perception skills or use imitation learning to teach the robot to manipulate different objects.

3.1 Perception

For a robot to work in uncertain environments and generalize itself to a wide variety of tasks, it must be able to sense and understand its environment. The task of processing the sensory data obtained from the various sensors into usable information is called as 'Perception'. Reliable perception is very important in robotics as most of the further operations in the pipeline will be dependent on the success and quality of the perception task.

Modern robots have a wide array of perception sensors which can be used to understand its environment. The sensors help the robot to have situational awareness so that it can perform autonomously in a dynamically changing real world environment. Some of the common sensors, hardware found in robots include 2D camera, stereo camera, RGB-D camera and lidar. In robotics, perception also plays an important role in localization of the object with respect to the robot and understand the properties (axis of the object, best grasp position) of the object so that it can be appropriately manipulated.

Perception for the task of handle manipulation mainly involves object detection and the object pose estimation. The object for this task refers to a small subset of items like door, window, and handle. The robot must be able to locate the handle and classify it (left turning or right turning or fixed) so that appropriate manipulation can be performed on it. The perception tasks in such manipulation also require very high accuracy to prevent the failure of the manipulation task or damage the robot and it surroundings. Handles have very specific physical properties like they are rigid, non-transparent and relatively Lambertian [1]. This specificity of the objects helps in constraining the complexity of the methods that are required for detection and finding the pose and grasp position.

[1] Lambertian surfaces reflects light equally in all directions

	Sensors				*Algorithm*		*Measured Value*		
	2D	*RGB-D*	*Lidar*	*Stereo*	*Learning*	*Classical*	*Classification*	*Localization*	*Pose*
Arduengo et al. [5]	✓	✓			✓	✓	✓		✓
Urakami et al. [61]	✓				✓		✓		✓
Nagahama et al. [45]		✓						✓	✓
Su and Chen [58]	✓				✓	✓	✓	✓	✓
Welschehold et al. [64]		✓				✓		✓	✓
Llopart et al. [43]	✓	✓			✓	✓	✓	✓	✓
Haynes et al. [24]			✓			✓	✓	✓	
Englert and Toussaint [16]	✓					✓			✓
Li et al. [39]		✓				✓		✓	✓
Banerjee et al. [7]	✓	✓				✓	✓	✓	✓
Axelrod and Huang [6]	✓					✓		✓	✓
Peleka et al. [48]	✓	✓				✓	✓	✓	✓
Chen et al. [12]	✓				✓		✓		
Endres et al. [15]		✓				✓			✓
Li et al. [40]		✓				✓	✓	✓	✓
Rühr et al. [53]		✓				✓	✓	✓	✓
Rusu et al. [54]			✓			✓		✓	✓
Stückler et al. [57]			✓			✓	✓		
Chen et al. [11]			✓			✓			✓
Klingbeil et al. [31]	✓		✓		✓	✓	✓	✓	✓
Meeussen et al. [44]	✓		✓	✓		✓	✓	✓	✓
Jain and Kemp [28]			✓			✓		✓	✓
Rusu et al. [54]			✓			✓		✓	✓

Table 3.1: Perception for the task of "Manipulation of handles". The classification of the various techniques used by the paper is done on the basis of the type of sensor used, whether the algorithm uses learning or class/non-learning method and the values extracted from the data.

3.1.1 Object detection for handle manipulation

Previous works

For the robot to grasp the handle, it must first be able to find the handle present in the camera frame. Object detection is considered as one of the fundamental problems in the field of computer vision and there has been a vast literature dedicated to this study. The field of object detection can be roughly divided into two periods :*"traditional object detection period (before 2014)"* and *"deep learning based detection period (after 2014)"* [70].

Every object or data in the world can be considered to be made of a set of representations called as features [36]. These can be as simple as straight line to as complex as splines. Traditional computer vision methods involve the use the hand crafted features for each type of object and finding the parts of the image where the relevant features can be found. The major disadvantage of hand crafted feature is that these features are not scalable, in the sense that they can't be extended to create very complex representation of the object without considerable human effort and time. This problem has been vastly overcome by the use of Deep Learning methods, where the network is able to learn a high level and complex hierarchy of features [69] with very less manual intervention. Although Neural Network based approaches are very powerful, it is still considered as a black-box and is difficult to understand and manage.

Histogram of Oriented Gradients (HoG) features or Haar features were widely used for detecting the handles [31] in the image. These methods involve finding the feature descriptor of a small localized areas in the image using a sliding window and are slightly invariant to geometric changes to the object and to some extent the lighting on the object.

There are supervised learning methods that make use of boosting for computing the Haar features[44] and use them to train a decision tree for the task of object recognition. But these methods also require large dataset (several thousands) of annotated positive and negative samples.

Some methods make use of the contextual properties of the objects to improve the classification accuracy of the detector algorithm. Rühr et al. [53] and] makes use of the spatial cues that handles are usually present in the centre of the door or away from the floor to reduce the computational requirements of the algorithm while also increasing the accuracy. [53] [44] makes use of the assumption based on the ADA (Americas with Disabilities Act) to make an assumption that the handle will always be present only within a fixed distance from the door plane. These kind of methods are logical in reducing the

overall processing required by extracting a ROI in the image. Realistically many of the spacial clues may fail for designs that are slightly different from the traditional norm and for non-general door designs. Assumptions based on standardized design rules like the ADA may become location locked and lose the generalizability.

In the case of object detection in 2D images, state of the art results[42][52] have recently been achieved after collection of vast amounts of data have become practically possible. Large amount of annotated data collected can be used to train a neural network for object detection. Due to the immense success in the robustness and generalizability, these methods have been used for both the task of detecting doors and handles in the camera frame.

Since training of detector is a computationally intensive task and require large amounts of datasets, some methods make use of markers to track the handle in the image then use the marker information obtained to grasp the handle. Although these methods simplify the perception subtask as no data-collection and training is required, the generalizability of these algorithms greatly reduce as it involves making modifications to the robot environment.

Few methods make use of the point cloud obtained from the RGB-D camera or laser scanners [57][11][45] to find the door candidates and eventually the handle. Unfortunately, these methods require several assumptions about the shape of the door and the position of the handle. Finding the door using point cloud data usually involves the assumption that the door and the wall surfaces present in the room are orthogonal to the floor plane[2] [45].

Figure 3.1: Using 2D object detection to extract the region of interest from the point cloud of the door. From the point cloud the cluster with the most different color is extracted outside the plane and assumed to be the handle.[43]

.

The noisy point cloud data obtained from the RGB-D camera is cleaned using a bilateral filter [45]. Few methods also use the 2D camera image for extracting the region of interest

[2]This is referred to as the expanded Manhattan World Hypothesis

to remove unwanted data in the point cloud and simplify the computation in the further steps. In many cases the preprocessing of the point cloud also involves removing the point cloud associate with the robot body itself [11].

Methods like Random sample consensus (RANSAC) algorithm are used to find the planes that are perpendicular to floor which also includes the door plane [53]. After extracting the planes, few techniques use color based growing methods[40] to find the largest cluster of points of the same color as the door. But this involved the assumption that there is a large difference in the color between the Door and the surrounding environment which is not always true. Other methods use line finding algorithms like Hough Transform to find all the vertical and horizontal lines in the point cloud[45]. Then these lines are use to create closed loops of quadrangle which are assumed to be doors.

Once the handle is extracted from the point cloud there are many methods that are use to classify the type of handle. [40] make use of Viewpoint Feature Histogram (VFH)[55] to classify the 3D object. VFH is a feature extraction method that can be used to obtain viewpoint invariant features. These are then use for object classification in point-clouds over a large variation of poses.

The major issue with RGB-D cameras is that they don't work very well under very bright sunlight or when the door is transparent. To compensate for the problems associated with individual types of sensors, some of the handle detection algorithms like the one designed by Meeussen et al. [44] make use of multiple sensor modalities to remove false positive while also improving the true positive. So for performing the robust handle detection, multiple sensors reading from the 2D camera, stereo camera and the lidar results must all show the same results to get successful final result. For the laser detection the authors make use of clustering methods based on the intensity difference of the point clouds between the handle and the door. For the 2D image "Viola and Jones classifier" is used along with a low recognition threshold and then the false positives are removed using a scoring system based on the data obtained from the stereo camera. These methods were shown to be very robust and was able to detect handles under a large type of lighting conditions.

3.1.2 Object pose and grasp estimation and for handle manipulation

Previous works

For the manipulation of the object the robot must be able to recognize the correct pose of the object and the best position so that it can correctly grasped. Different handle require

specific grasping point so that the object (door) attached to the handle can be manipulated by applying the least force based on the underlying model.

Few methods assume that the robot already has rough information about the location of the cabinets[45], doors along with shape and the pose of the handle in its knowledge base [53]. These methods have the model of the handle along with the articulation model and the pose as part of the environment map. The algorithm to find the precise orientation involves using the prior information available from the model along with fitting the best possible line onto the handle point-cloud cluster for performing the final grasping movement. The major problem with using explicit database of handle models is that it is practically not possible to obtain data about handles of all types.

Li et al. [40] make use of a Microsoft Kinect sensor to capture the point cloud of the robot environment. Then based on the different types of door handle that is recognized, specific types of 'grasp patterns' are associated to it. 'Grasp patterns' consists of the description and details about the grasp point and the best way to grasp. These grasp patters are manually provided for different types of handles.

Nagahama et al. [45] used the location of the door handles and the dimensions of the door as features to estimate the model of the door and pose of the handle. A random forest is first trained using the relationship between the features $\mathrm{f}(D_m)$ and door models using training images collected from the internet. The features are based on the simple geometric features of the door and are given as follows :

$$\mathrm{f}(D_m) = [\mathrm{f}_x(D_m), \mathrm{f}_y(D_m), \mathrm{f}_h(D_m)]^T \tag{3.1}$$

L_w and L_h are the door width and door height. H_x and H_z are the position of the handle in the horizontal and vertical direction respectively. f_h is the direction of the handle. Although this method provided a good prior for knowing the door model and the handle pose, the data collection and annotation is very time consuming.

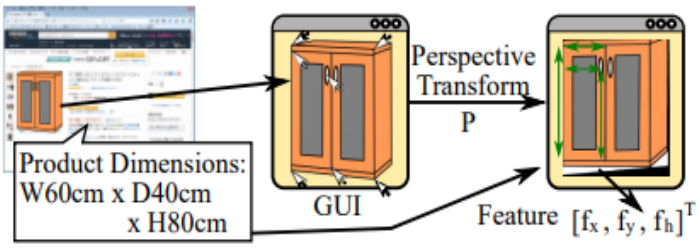


Figure 3.2: Using images from the internet for extracting the features and training random forest.[45]

.

Conclusion

Deep learning methods have given state of the art accuracy and the best generalizability on object detection tasks on RGB images. But these methods require very large number of annotated images and computational power especially for training the model. For common household items the datasets are easier to create but for other objects where it is difficult or impossible to collect image, traditional methods that make use of handmade features are better but with lesser generalizability.

Non-learning methods for object detection in 3D point clouds requires a model and doesn't generalize well to different object types. Although there are learning based methods that have shown good results, they have not been used for the handle detection task as it is very difficult to collect good datasets and these methods are very computationally expensive to run on general robot hardware.

Multimodal/sensor fusion approaches are helpful as they help in compensating for the problems faced by the individual methods. These methods are robust in a wider range of environments and can help in improving the results of both detection and pose estimation tasks.

3.2 Manipulation

Manipulation skill in humans is a complex synergy between the dexterous joints of the hand and fingers along with the planning capability of the brain. Similarly, Autonomous manipulation in robots is a very complex task due to the challenges involved in understanding the physical structure of the object and use the obtained object model for manipulating it. Intrinsic degrees of freedom of an object is defined as the relationship between the different components of the right body and are related to the actual functioning of the body. Extrensic degrees of freedom on the other hand are the relationship between the object to be manipulated and the robot degrees of freedom [30]. The manipulation algorithm must consider both the degrees of freedom of the robot and the object for successfully completing the task.

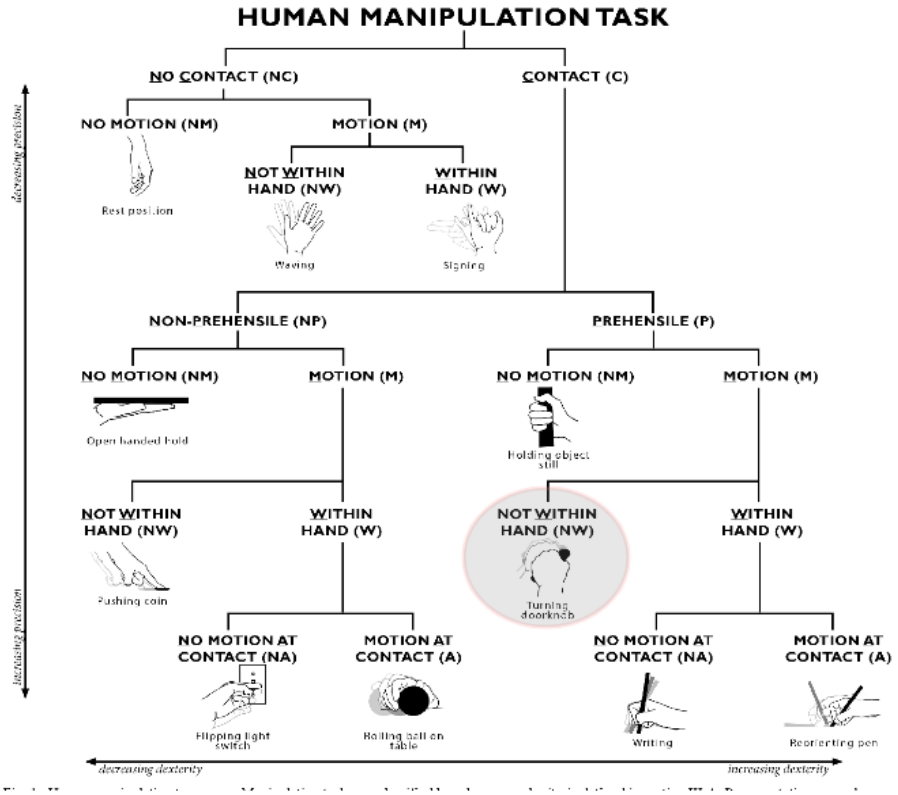


Figure 3.3: "Hand Centric" classification of human manipulation tasks. Classification is based on several categories depending on the contact of the hand with object, motion of the object or the hand, the type of grip [10].The circled diagram shows where "handle manipulation" task lies in terms of its complexity.

.

Based on the manipulation taxonomy given by Bullock and Dollar [10] the task of manipulation of handles can be considered as contact based, prehensile task where the object is grasped by the hand and the motion of the object doesn't take place within the hand. Prehensile means the object has multiple contact points with the hand and can is stable to small external forces. The grasp on the handle is a closed grasp and the motion of the handle doesn't occur within the hand (no fingers movement involved) but involves the hand and the body to apply forces on the object. From the given classification of the task it can be seen that the manipulating handles can be considered as a medium precision and medium dexterity manipulation[10].

This section will tabulate (Table 3.2) a broad classification of the types of methods in manipulation that have been used in the previous literature. A brief summary of the various algorithms will also be given along with the associated advantages and disadvantages if any.

Previous works

The various algorithms that have been used for the manipulation can be broadly classified as learning, compliance, control based methods and probabilistic approaches. There are other classification like graph based, fuzzy logic based but these methods have not been widely studied. Control based methods uses hardcoded motion trajectories of the manipulators for applying forces on an object or for reaching a specific pose. Unlike in control based methods, in learning based manipulation methods the robot is able to find the best possible trajectory for performing a given task without being explicitly programmed for the task. The robot is able to use its previous experiences for improving the parameters of the policy for solving the task. Compliant manipulation involves the use of external forces to guide the trajectory of the manipulator.

		Papers	*Study*	*Advantages*	*Disadvantages*
Control	***Compliant***	[5][45] [46][24] [15][28] [27]	Compliance scheme of manipulation follow the external force exerted by the environment to keep the internal forces within a limit.	Compliance is helpful for manipulating handles with different forces without damaging the end-effector. The inaccuracies in the modeled environment can also be handled robustly.	Robot must have a high degree of freedom joints and high quality force sensors for compliant based algorithms to work.
	Non-Compliant	[7][6] [38][29]	Non-compliant manipulator follows a predetermined trajectory regardless of the external forces applied on the end-effector.	Non-Compliant manipulation always take deterministic trajectory. These are helpful for robots that must perform the same task in a static environment.	Non-compliant manipulation can cause the robot to get damaged due to extreme external forces acting on it due to the model inaccuracies.
Reinforcement Learning	***Policy Search***	[61] [46]	These methods involves the use of RL for finding the best parameters of an underlying policy to complete a task. To reduce the parameter space of the policy, demonstrations are used to perform the task.	Only a small number of rollouts are required for the policy to converge.	Good quality initial demonstrations are required to help the policy parameters to converge faster.
	Deep RL	[23][66]	Deep RL use Neural Networks for learning the underlying policy of a task. These methods learn the policy from scratch by exploring the state or the action space.	No initial human demonstrations are required for initializing the policy parameters.	Very high number of rollouts are required for learning the best possible policy for complex tasks. Safe exploration of the state or action space is required especially in physical systems.
Probabilistic Methods		[5][16] [49]	Probabilistic methods usually involves encoding the trajectory or any relevant parameters as a probabilistic distribution and infer the required details by using it as a prior.	Probabilistic methods consider the uncertainty due to the noise and outlier in the data so provide a better generalizability.	Large number of rollouts are required to get a distribution that models the physical environment very realistically.
Graph based methods		[64] [63]	The states and actions of the robot are encoded as a graph and uses graph optimization to find the optimal path to imitate a trajectory.	Teaching a robot a new skill is simple and natural as it involves imitation by the data directly obtained from the camera. Highly accurate tracking is not required and only a simple apparatus is required for collecting the demonstrations	Trajectory imitation is very difficult in the cases where the marker might get occluded for a very long period of time or in cases which involves the rotation of the marker at a single position
Fuzzy logic		[58]	Fuzzy logic involves converting binary decisions into a continuous set of decisions with partial truths.	Compared to hardcoded values and strict rules in manipulation tasks, fuzzy logic can help in increasing the robustness of the algorithm.	Getting good/expected results involves tuning the controller logic.

Table 3.2: Methods of Manipulation used for the task of "Manipulation of handles"

There are many types of handle kinematic models that are available for the robot for manipulation. The algorithms must take into consideration the correct model of the object to make sure that the manipulation task is successful and the robot doesn't get damaged. Some algorithms are designed to open doors which don't involve any movable unlocking mechanisms [57][45] like cabinet doors and refrigerator doors. In [57] the robot is made to move backwards after grasping the handle correctly. The compliance of the robot arm in the yaw and the lateral direction is used to follow the best trajectory for opening the doors. The problem in this case is that if the robot doesn't stand sufficiently far from the door during manipulation it may lead to collision and subsequent failure of the task.

Su and Chen [58] designed their own mobile robot with a manipulator that can detect handles and open doors. For manipulating different types of handles which have different force profiles, the authors of the papers presented an algorithm that made use of fuzzy logic. To adapt the manipulator motors for different forces a min-max controller. Defuzzifiers are used on pressure sensor values and the rotated angles of the doors to get single quantifiable value for different pressure sensor values of the robot.

$$\mu_B(y) = max[min(\mu_A(x), \mu_R(x, y))] \tag{3.2}$$

Here x, y are the pressure sensor values and R denotes the rotation angle.

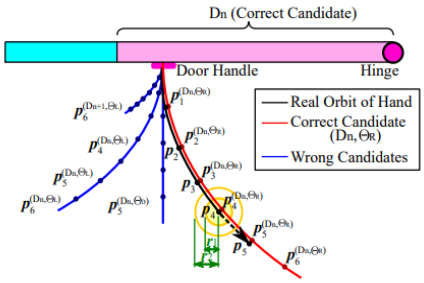


Figure 3.4: Learning the most appropriate trajectory for manipulating the handle using compliance.[45]

.

Nagahama et al. [45] also uses compliance in the robot arm for opening the door or cabinet with different models. The manipulation step is preceded by a perception step that provides a manipulation model (Φ_k) based on the classified door (D_m). Since many doors of similar looking handle might have the different manipulation models, a score ($s_0^{D_m,\Phi_k}$) is provided for the possibility of each model to be correct. A target trajectory is also given based on each of the possible door manipulation model. Based on the trajectory,

the robot uses very low frequency compliant control to manipulate the handle in small time-steps. The robot uses the compliance to follow a series of pre-computed target hand positions while making sure that the controller follows the door trajectory correctly. If the model is not correctly decided, the robot changes the trajectory to the next closest fitting trajectory. The compliance helps the manipulation tasks to be completed successfully even if there are errors in the perception model and the trajectory calculated is not correct. This method faces the limitation of not working correctly when the correct door model cannot be achieved by the perception task when the door is transparent or the door plane is correctly visible.

Endres et al. [15] presented an approach of opening door by explicitly modeling the dynamics and physical properties of the doors from the sensor readings. Firstly perception is used to find the geometry of the door, ie. the width of the door, the center of rotation of the door. For doing this a marker is attached to the door. A point on the marker is tracked along with the direction of the normal of the door. By tracking the two variables during the model training step, the centre of rotation of the door and the other geometric details about the door can be found in its closed form. This method has the advantage of having a faster execution speed compared to traditional compliance based methods that doesn't use the door dynamics and use slow quasi-static motion to push the door. The advantage of this method is that since the physical properties of the door is used, the algorithm can generalize over to different door start and end states. This method also showed accurate results for opening the door while not requiring a high DoF robot and high computational power.

Welschehold et al. [63] provides a novel approach of teaching the robot to manipulate objects by formulating the problem as a graph optimization problem. This method is a type of *Learning by Demonstration* which uses a human demonstration to teach the robot complex skills or policies. The goal of the authors is given as : "to adapt human demonstrations in a way that it allows the robot to learn a joint motion model for both the robot gripper and base enabling it to reproduce the intended action". The general steps of the algorithm involves using the RGB-D perception data to track markers attached to the various parts of human body. The markers are tracked to get an approximate joint trajectory followed by the hand and the object (X_h,X_o). Each observed state of the trajectory is stored in the form of a node in a hyper-graph where the links indicate the kinematic constraints between the various parts of the robot. The authors of the paper try to solve the problem associated with the missing data in the trajectory by using linear interpolation to solve the translation motion and slerp for the rotational motion. A few preprocessing steps are done for the data to make sure that the trajectories of the object

and the hand are of equal length or time intervals and that the data points are close enough to each other. Although this method shows a promising technique of using imitation to teach the robot manipulation, a lot of modification are required to the environment like adding markers to the human teacher and the door.

Collecting data for reinforcement learning on a real robot is a very time consuming task. Although it is easier to learn policies when model-free RL methods are used, model based methods have the advantage of being able to test new and novel ideas without damaging the actual robot hardware or causing danger to the human user. [61] created a simulation environment in the OpenAI Gym[9] environment to test various reinforcement learning algorithms designed to manipulate handles and open doors. For successfully learning robust models the DoorGym environment is designed to be highly configurable and with domain randomization capability. The different environment parameters that can be randomized are the handle types(lever handle, fixed handle and round handle), the lighting conditions, the height of the handle on the door, the forces on the door etc. The robot initial positions are also randomized to make the policy robust to different robot and handle positions. Along with the simulator the authors also provided two pretrained RL models; Proximal Policy Optimization (PPO) and Soft Actor Critic (SAC). The complete details about the reward function design is given in [chapter 4] .The obtained trained policy from the simulator is also ported on a real robot and the authors were able to achieve over 59% baseline score.

For improving the adoption speed of the Reinforcement Learning algorithm, Nemec et al. [46] gave a new policy parameter search method based on task space constraints. Like how humans try to follow the trajectory of the manipulated door after grasping the handle tightly, the authors of the paper used the compliance of the robot to follow the trajectory of the door with the goal to keep internal forces in the joints as low as possible, without reaching physical limits of the robot manipulator. The underlying controller is designed to move the manipulator in the direction of the door trajectory guided by the environment constraints. The controller must also make sure that the applied forces should not lead to inadmissible velocities in the robot joints. For extending the above framework using reinforcement learning, the controller is used as an exploration strategy for learning an underlying Radial Basis Functions(RBFs) with its weights and centers as the free parameters. Although this method had major advantage of being able to learn different types of policies for opening drawers, doors etc and converge within 20 rollouts, it requires very a compliant and highly flexible robot.

Deep Reinforcement learning methods use neural networks for learning the policy of a task from scratch, without requiring any hand-engineered policy. Gu et al. [23] used deep

reinforcement learning for learning manipulation policies for complex tasks like the door opening without any prior demonstrations. The method uses multiple robots for collecting rollouts in parallel [3] and then use the obtained data to train a Q-function approximator. The policy update is decoupled from the experience collection task so this method is considered to be asynchronous. Although Gu et al. [23] showed very promising results from Deep RL the method was extremely sample inefficient. The method is not scalable for many new tasks considering the large number of rollouts required for training the neural network.

Conclusion

Research on control based methods for the task of handle manipulation were helpful in very specific cases when the details about the environment can be properly perceived and there are no expected disturbances. Compliance based methods use the force cues from the environment to perform the manipulation task. Compared to pure control based methods, compliance can prevent unwanted stresses on the robot frame due to the environmental constraints and forces.

Reinforcement Learning can be divided into policy-search methods and Deep Reinforcement Learning based methods. Policy search methods are helpful to learn complex manipulation trajectories but require very good quality rollouts that can be obtained from human teacher by kinesthetic learning or imitation learning. DRL in robotics are not feasible due to the fact that very large number of rollouts (1 million) are required even for a simple task.

There are other novel algorithms that make use of graph or probabilistic distribution to store the trajectories and use them for better generalization to compensate for model errors.

[3] This experience pooling method is also called as Collective Robot Learning

3.3 Human Guidance for Robotics

When robots work in domestic environments, performing a task fully autonomously are still not possible due to computational, algorithmic constraints. Many algorithms require some initial instruction or prior information from humans so that the complex knowledge about the environment doesn't need to be hardcoded. The prior data that a robot may require from the task of manipulation of handle are the following :

Perception as mentioned in section 3.1 may require details about the location of the door in the camera frame for the robot to localize [45] [24], handle position, the type of handle based on the kinematic model, and the pose of the handle for the best grasp.

Manipulation of handles has many priors that may simplify the computational load for performing the task. The robot can be given the previous knowledge about the possible obstacles that may be present in the manipulation path along with the best possible trajectory to reach the object, in our case the handle.

Putting the human in the loop makes sure that the complex capabilities of the manipulator is not blocked by the limitation of the perception and intelligence of the algorithms. There are plethora of ways in which the prior data can be provided to the robots for performing a given task. Many industrial robots use a teaching pendant for teaching the robot to follow a specific trajectory or perform the required task. The teaching pendants may have joysticks, buttons or other forms of input that is used by the human teacher. Many modern teaching pendants also have touch screen to provide the perception related hints like region of interest based on the camera frame visible. There are other methods like LfD that are used to replace the time-consuming manual task of programming each and every position in a manipulation trajectory.

Previous works

For the task of manipulation of objects in domestic environments there have been a number of methods used to obtain the information from the human teacher. Since it is inefficient and impractical for the human to provide all the required information to the robot, only the important data that cannot be automatically detected by the robot, must require human intervention.

Capturing various user data requires different types of input devices. Teaching the movement usually involves the use of a teaching pendant, whereas manipulation task

	Paper	Method
GUI	[45][24] [39][7] [11][27]	These methods mainly involves bringing the human inside the perception and planning loop by providing a GUI based interface for the user to select the best possible action to perform or provide extra task related priors.
Human Teacher	[5][46] [64][16] [15]	Human teacher methods use a human to provide high quality rollouts by either using imitation based methods or kinesthetic learning. These rollouts can be further used for learning a policy or training the robot to directly perform manipulation.

Table 3.3: Methods of human guidance used for the task of "Manipulation of handles"

planning may also work well by using imitation based methods. Chen et al. [11] uses a multi-touch based method for providing the input required to manipulate furniture and electric equipment by a PR2 robot. The input can be used to perform push, pull and rotate actions for manipulating a large number of objects like microwave, refrigerator, closets and cupboards. Depending on the type of action to be performed, the input could be the position, direction to apply forces and the axis along with the forces should be applied.

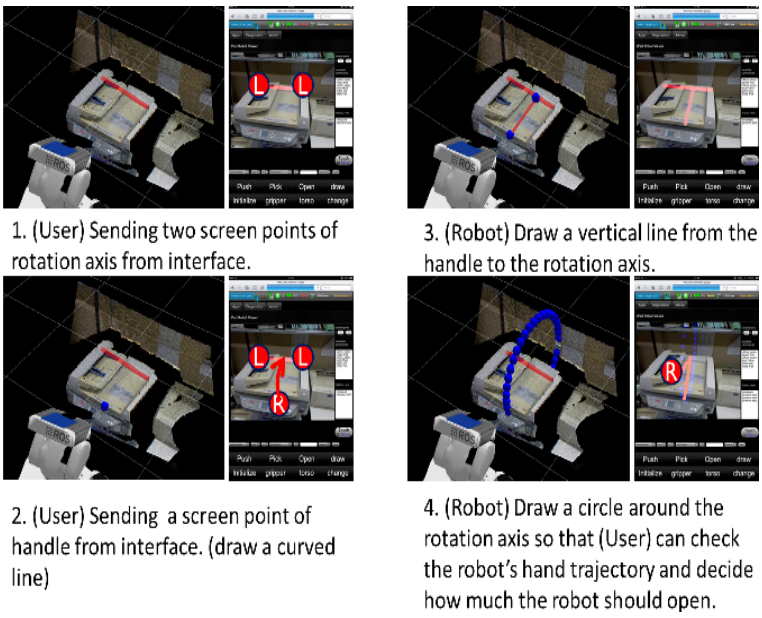


Figure 3.5: User providing the kinematic model as the prior information to the robot using a touch based GUI for improving manipulation planning. The user provides a set of points (L) denoting the axis along which the rotational force (R) should be applied. [11]

.

Simple GUI can help increase the speed of teaching the robot to manipulate new object and teach it new actions. Major problem with these systems is that a lot of extra code is required to generate a stable platform for joining the robot and the user input. Nagahama et al. [45] proposed a method where the user needs to give a robot only a single instruction to manipulate a door or a cabinet. Using a touch interface, the user presses the area where the door-handle is present in the image which is obtained from the robot camera. The input is use to extract the region of interest in the point-cloud data of the RGB-D camera, which is used to detect the pose of the handle and perform the manipulation.

Learning from demonstration (LfD) is another type of method that is used for teaching the robot simple manipulation skills where the robot uses the observations of the human actor demonstrating a skill and uses many data points like the human joint positions etc. to formulate the trajectories on the robot. These methods are particularly helpful in tasks where a particular trajectory is required for manipulating an object rather than using the trajectory given by the grasp planner. Methods similar to LfD has been used by [63] where the RGB-D observations of a human performing an action is used to teach the robot new capabilities like opening the cabinet door etc. The goal of the authors was to adopt the data obtained from the human demonstrations to learn the "joint motion model" for the robot gripper along with the base to perform the task successfully.

Conclusion

External inputs help the human teacher to provide lots of useful prior information about the environment and the task related knowledge to the robot. Inputs might be taken from key-presses on a teaching pendant or a touch screen GUI depending on the types of input required or the hardware architecture of the robot.

LfD methods like kinesthetic teaching and imitation learning are used to collect the trajectories for performing a given manipulation task. The collected trajectories can either be used as a control sequence for performing the task or as an initial dataset to train a new policy using reinforcement learning.

4

Methodology

Theory attracts practice as the magnet attracts iron

Carl Friedrich Gauss
German mathematician, 1777-1855

The task of "Manipulation of handles" can be logically decomposed into its constituent simpler but related tasks of perception and manipulation. This chapter of the report will give the methods that were selected by us for implementing the various subtasks for completing our task. In perception we will give a brief description about the deep learning models trained by us according to our requirements. In manipulation we will discuss about two methods tested by us : Simulation for learning a reinforcement learning policy and Maximum Likelihood Estimation for sampling the best grasping position.

4.1 Perception for detecting handles

From the literature survey, it was evident that Deep Learning methods are the best choice for performing object detection on RGB images [35] when a large dataset of images are present or can be collected and generalization over many types of objects is required.

Recent influx of research in the field of DL and CNN in particular, there are many types of architectures that are being studied. For the task of object detection the models can be broadly classified based on three stages [69] of the task that it performs : 1. *Region selection stage* : The size and aspect ratio of the object in an image may depend on a large number of factors like the object size, shape, distance to the camera, focal length of the camera etc. This stage of the model deals with finding the most efficient way to find objects

of different sizes and aspect ratios without having to scan the full image with multiple scales of sliding windows. Many strategies make use of fixed number of windows to reduce the computational complexity of the task. 2. *Feature extraction* : There are different hierarchy of visual features that the must be found so that the object detection can be performed. The model must be able to extract features of an object that are robust against different illuminations, backgrounds and poses. 3. *Classification stage* : The classifier must use the information gathered from the feature extraction stage to make informed decision about the probabilities of the classes that the features may belong to.

We selected three object detection architectures in particular, based on the ease of training, accuracy of the model and the inference time for processing a single image. Since the trained model will be used on the HSR-Robot, the models will be tested on the images taken from the various 2D cameras present in the robot.

4.1.1 Single Shot Detector (SSD)

SSD[42] makes use of a single neural network to detect objects of various aspect ratios and sizes. This is done by discretizing the bounding boxes into multiple bounding boxes of default set of aspect ratios and different scales. SSD is considered as one of the simpler architectures of object detection as it doesn't make use of any regional proposal networks unlike the R-CNN [21][20][52] family of object detection architectures. Since only a single network is present the training of the model can be done in a single stage.

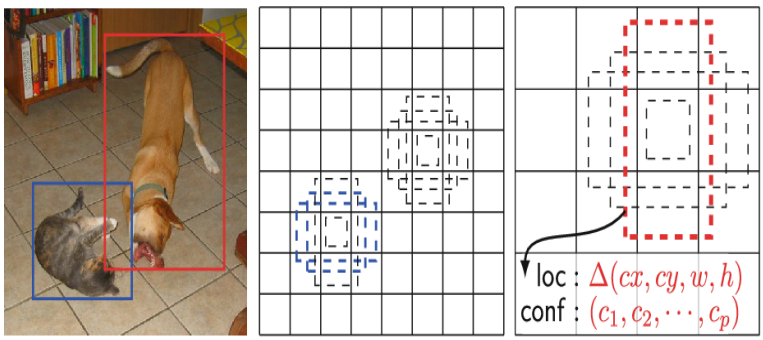


Figure 4.1: SSD is a single stage detector in which the prediction is done on bounding boxes of fixed sizes and aspect ratio on (8x8) or (4x4) feature maps. During training the size and the location of the bounding box is optimized with respect to the ground truth bounding boxes and the class probability

.

The basic steps in SSD involves dividing the input image into multiple cells called as the

feature map cells. For each of the feature map cells there are associated default bounding box of various different aspect ratios. The aim of the network is to predict the confidence score for all the object categories $(c_1, c_2...., c_n)$ along with the offset for the bounding boxes. Since multiple bounding boxes may have the same object, non-maximum suppression is used to remove the remove overlapping boxes and finally give a single prediction box. So the overall loss function is given by :

$$L(x, c, l, g) = 1/N(L_{\text{conf}}(x, c) + \alpha L_{\text{loc}}(x, l, g)) \tag{4.1}$$

where confidence loss, L_{conf}, is the confidence of presence of an object and localization loss, L_{loc}, is the offset between the predicted bounding box(l) and the ground truth box (g).

The SSD architecture uses a base network like VGG-16 or Inception net as the main feature extraction layer. A convolution feature layer is added to progressively decrease the image size inside the network so objects of multiple scales can be detected. SSD network is able to obtain very high accuracy (similar to FR-CNN) while being significantly faster.

4.1.2 Faster Region-Convolution Neural Network (FR-CNN)

FR-CNN[52] lies in the family of two staged detector models[21][20] for object detection. These methods have the first stage where regional proposals are given and these are used for finding the object confidence scores in the second stage.

In FR-CNN Regional proposals networks(RPN) are used to obtain the a rough location of the objects in the image. The RPM takes in the image and uses a base convolution network to create a feature map of the image. The input feature maps are sent to an RPM which uses the features to classify how likely the feature map is a background or foreground along with a set of bounding boxes of various aspect ratios called as the anchor boxes. The proposals that are obtained from the RPN are sent further down the model where object classification is performed.

The training of the model involves a multitask loss: a log loss over two classes L_{cls} which gives the probability of object vs not-object for classification. The regression loss for FR-CNN can be considered similar to a bounding box regression where the anchor box, with width, height and the box centre coordinates (x,y), is optimized with respect to a ground truth box.

FR-CNN are able to obtain high quality bounding boxes for the task of object detection while also improving the overall object detection accuracy.

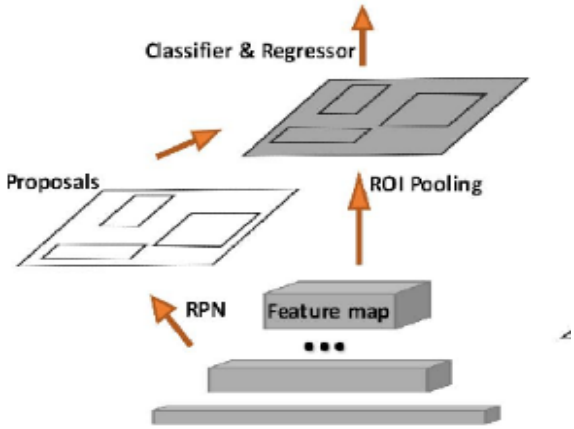


Figure 4.2: FR-CNN is a two stage detector where the first stage involves the approximate region selection and the classifier stage improves the results of the bounding-box.[52]

.

4.1.3 You Only Look Once(YOLO)

YOLO[51] lies in the family of single stage detectors like SSD that don't use any region proposals. So YOLO uses only a single network and considers the the whole global context of the image to perform the detection task. YOLO frames the problem of object detection as a regression problem unlike the two stage detector methods. The input image is first divided into $S * S$ grids. For each box in the grid the class confidence score is given along with a default number of bounding boxes.

For each of the box in the grid cell there are multiple bounding boxes that are predicted for each one of them called as the anchor boxes. These anchor boxes have different aspect ratios and can predict different objects that have their mid point in the same grid. For predicting the objects at different scales, a feature pyramid is created at three different scales for the object.

YOLO model has the advantage of being able to run realtime on GPU and can run at nearly realtime (10fps) on a CPU. Since YOLO divides the image into grids and then runs the object detection on each of the grid, it is not possible to detection small objects that lie together within a single grid correctly.

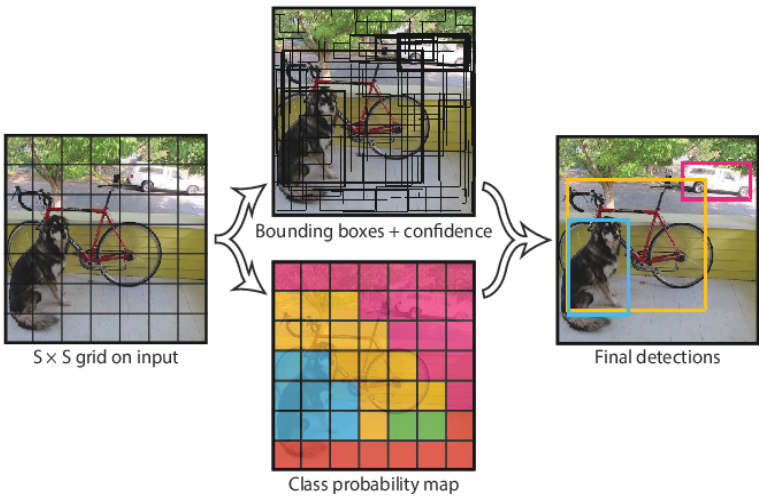


Figure 4.3: YOLO is also a single stage detector in which the input image is divided into a fixed number of grids and the class probability and the bounding box regressor are used to detect the objects in the image.[51]

4.2 Manipulation of different handles

In domestic environments, where the environment is not controlled and structured, the prior information and map of the environment is not enough for manipulation. Successful autonomous manipulation of handles likewise requires a tight coupling between the perception subtask along with the manipulation subtask.

Programming controller for robots is a highly complex job that requires a deep understanding about both the robot hardware and the environment in which the robot is working. Hard-coding controllers for very specific tasks are therefore robust in very specific conditions and cannot adapt to new environments and factors affecting the robot. These also require very accurate model of the robot and its environment, which is realistically very difficult. Learning methods on the other hand expect the robot to solve the task at hand by automatically learning all the relevant features associated with it.

4.2.1 Reinforcement learning for manipulation

The selection of the right RL algorithm for the task of robotics is very important as the final success and the failure of the algorithm depends on it. The task of robotics manipulation has its own set of constraints it sets on the type of algorithms that can be used. Robotics manipulation is a continuous control problem of very high dimensional

space (both state space and action space). In robotics due to the various sensor and hardware constraints it is also considered to be noisy state estimate [53]. Another major consideration in robotics is that it is very difficult take a large number of rollouts due to the time consumption and the possibility of the hardware getting damaged. The problem of robot RL in manipulation is also episodic in nature and many complex manipulation trajectories can be decomposed into smaller tasks. All the above reasons means that only a small subset of RL algorithms can work well for robotics.

Policy search RL algorithms have the required prerequisites for working well with robotics manipulation tasks as they are able to converge with a comparatively small number of rollouts ever for very high dimensional problem spaces. Safe exploration for training the policy is ensured by using a simulation environment.

Simulation for reinforcement learning

Collection of rollouts for training a reinforcement learning policy is a very difficult task considering the large number of samples required for the policy to converge. The major problem when running reinforcement learning on a actual hardware is that the real-world is not a perfect MDP. The major components of the MDP ie. the states, rewards, observations and the action are all noisy and can be difficult to model. The states of an environment can be very ambiguous and would require lot of simplifications so that the data can be compact and valid to be used with an algorithm. The state and observations will also not be fully observable due to the noisy sensors[2]. Similarly there are many actions that are performed in the real-world that have no logical reward generated or is very difficult to write a reward function. For example folding cloths, cooking etc. The actions in an MDP are difficult to model due to the complexity of designing controllers for very high dimensional manipulation tasks and constant degradation of hardware with time.

Taking real-world experiences on robot hardware is very time consuming as it is very difficult to automate all the steps involved. Due to the added randomness and errors in the sensors it is nearly impossible to set the same initial state for each rollout. For example in the task of manipulation of handles the the robot can be made to follow the trajectory, try to grasp the handle and pull the door without much human interaction involved. But to know if the door is open correctly requires a lot of modification to the environment. It is also very difficult to reset the training apparatus, the robot and door, back to its initial state perfectly without some human intervention.

Few of the problems can be alleviate by using an approximate simulation of the robot system and the environment. A good simulation environment makes it simpler to collect a

large number of training samples in a reasonable amount of time with random environment variables that can be changed for domain randomization. The simulation environment has its own set of problems that must be taken into consideration. The simulation environment designed must approximate the physical characteristics of the objects it is trying to simulate properly. This is a very difficult task as it is nearly impossible to model all the properties of the object perfectly and within reasonable computational limits. This difference between the real world and the simulation environment is often referred to as the *Sim-to-Real Gap*.

DoorGym environment

For testing the reinforcement learning we planned to perform the simulation of the task of opening the doors in the simulation environment called as DoorGym environment[61][60]. DoorGym environment is designed to be an environment for testing various RL algorithms for the task of manipulating handles to open new doors. DoorGym is designed for learning behaviours on physics simulators and using the policy to transfer to a real robot. DoorGym provides a vast number of handles like the pull, lever and round handles and knobs. The wide number of random handles and lighting conditions helps to use domain randomization so that the learnt model to be transferred to a real robot.

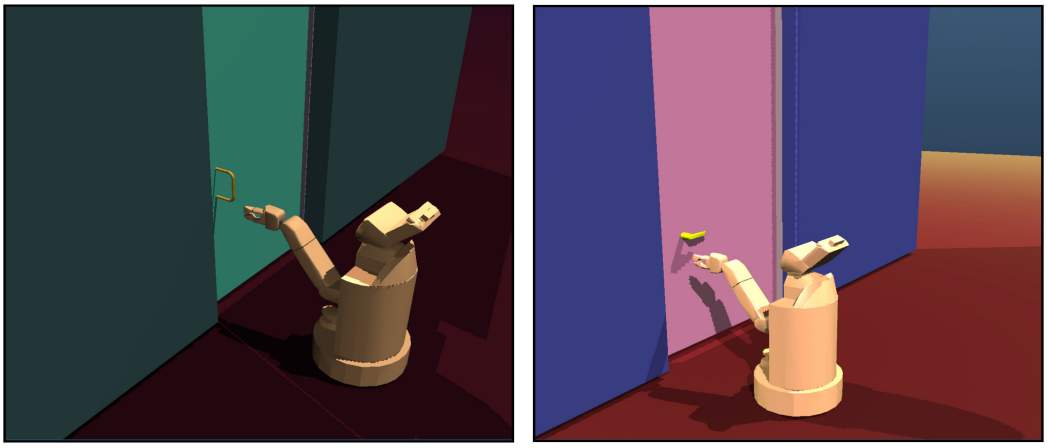


Figure 4.4: Toyota HSR ported to the DoorGym simulation environment.

Reward Function

The initial position of the robot is such that it is directly within the grasping length of the door handle. The robot arm is parallel to the door plane and the fingers form a grasp around the handle.

The total size of the action space that is used by the simulator is dependent on the DOF of the robot; 8 DoF combining the 3DOF of the base, 5 DOF of the Arm and the Torso lift.

The reward function that was used is the same as the one given by the DoorGym[61]:

$$r_t = -a_0 d_t - a_1 log(d_t + \alpha) - a_2 o_t - a_3 \sqrt{u_t^2} + a_4(\phi_t) + a_5 \psi_t \tag{4.2}$$

d_t is the distance between the fingertips of the robot end-effector and the centre of the doorknob. The log term (also referred to as Lorentzian -function[37]) is used to increase the precision as the end-effector of the robot reaches very close to the handle. The term o_t is dependent on the difference between the end-effector grip and the handle orientation. ϕ_t and ψ_t are the angles of the door and the handles respectively.

4.2.2 Maximum Likelihood Estimation for learning grasp distribution

The goal of using MLE for the task of manipulating handles is to model an appropriate distribution for sampling the best possible goal positions. The final goal position should lead to the successful grasping of the different type of handle and performing the final execution trajectory, like twisting the handle for lever knob or round knob and or pulling for a fixed knob.

Each collected rollout during the training phase has the end position of the robot end-effector stored as a coordinate {x,y,z} called as the goal pose/coordinates. The rollouts collected for the successful poses are assumed to be modeled as a weighted gaussian distribution where the mean (μ) and the variance (σ^2) of the distribution is selected using Maximum Likelihood Estimation (MLE). Sampling from the Gaussian distribution gives the output goal position as 3 variables {x,y,z} denoting the 3D goal position of the end effector.

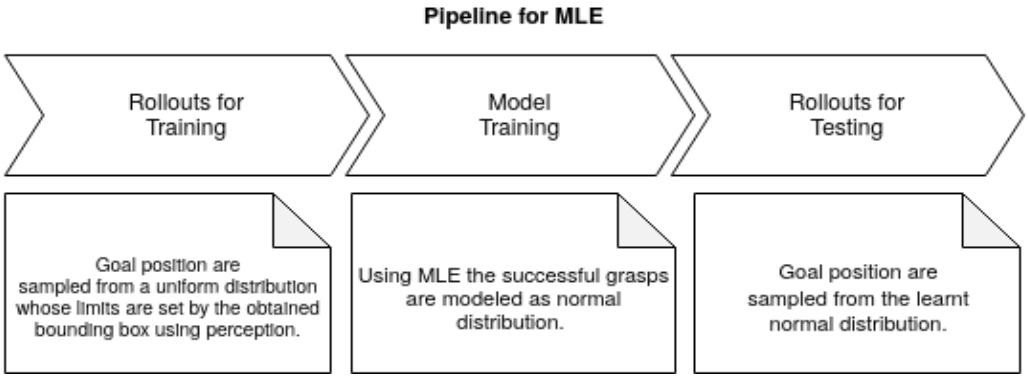


Figure 4.5: General pipeline for learning a distribution for sampling the best grasp positions.

Procedure for collecting rollouts

The steps for collecting the rollouts for subsequent training of the MLE model involves the following steps:

1. The robot is places in a fixed initial position such that the start position is the same in every iteration. The initial position should be within the grasping reach of the handle.

2. We initialize the distribution as a uniform distribution(prior) from which we sample the goal positions for training the model. The limits of the 3D position are fixed

within a given set of limits given as $x \in \{\alpha_0, \alpha_1\}$, $y \in \{\beta_0, \beta_1\}$ and $z \in \{\theta_0, \theta_1\}$. This 3D cuboid behaves as a restricted region for exploration of the goal positions.

3. For each sample of the goal position that is sampled from the initial distribution, the DMP implementation of the robot is used to accomplish the rollout. For each rollout the success of grasping the handle is manually noted.

4. After performing 50 such rollouts the initial distribution is updated using the new success and failure data such that the new distribution will lead to a higher chances of success during the subsequent sampled goal positions.

5. The training and the update step for the distribution is performed multiple times till we obtain a Gaussian distribution for sampling the goal position with very high rates of success.

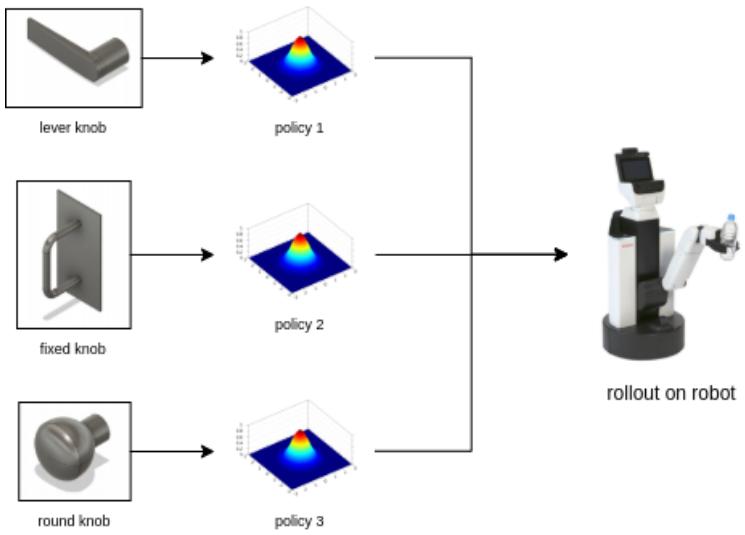


Figure 4.6: Based on the type of handle obtained from perception, the appropriate distribution is selected for sampling the best goal position. Using the sampled position from the learnt distribution the rollout is performed on the robot. Selection of the appropriates distribution can be done by leveraging the detection results.

5

Results

An approximate solution to the right problem is far better than an exact answer to an approximate problem.

John Wilder Tukey
American statistician

5.1 Perception - Deep Learning

For using deep learning, it is very important to have a vast dataset with a good diversity of images of different objects that we want to detect or classify. Since in most cases it is almost impossible to collect a very large dataset (1 million images) to train an entire Convolution Network from scratch, we instead use a method called as transfer learning.

5.1.1 Transfer Learning

Transfer learning is the technique of using a model which is pre-trained on a very large dataset for performing the task that is relevant for our requirement[3].

In case of image detection and image classification tasks transfer learning deals with using the pre-trained CNN and use it as a feature extractor for the new dataset. There are two common types of transfer learning methods that are used depending on the size and the similarity of the new dataset to the dataset on which the model is trained on :

1. **CNN as a fixed feature extractor** method involves training only the classifier of the CNN model. This is done by using a pre-trained model to extract the features

from an image as a vector and then use a linear or an SVM classifier to classify the new dataset. This method is preferred for cases when the size of the image dataset is very small while at the same time the features in the new dataset are very different from the one the model is trained on.

2. **Fine-tuning CNN** involves using backpropagation for training the various layers of the model using the new features. If the size of the dataset is very large and then all the layers of the CNN can be unfrozen to modify the weights. In case of a small dataset with similar features as the dataset of the pre-trained model, only the last few layers of the model can be unfrozen and the training can be used to fine-tune those layers.

5.1.2 Handle dataset

For performing the task of handle recognition the dataset provided by the Arduengo et al. [5] was used, this will be referred to as the *"Handle Dataset"*. This dataset consists of 1,213 annotated images of 4 classes of objects: Handle, Cabinet door, Refrigerator door, Door.

$$\text{Object classes} : \mathbf{c} \in \{Door, Handle, CabinetDoor, RefrigeratorDoor\} \tag{5.1}$$

It was observed that the models that were trained on the "Handle dataset" were unable to give the desired detected accuracy under the conditions of the lab in which our Toyota HSR robot was tested on. To improve the performance of the detection model on the images captured by the Toyota HSR, we have collected an extra 200 images from the various cameras of the robot. This will be called as the *"HSR Dataset"*. From this dataset, we will be using 150 images for training the model and the rest 50 images for evaluation of the trained model.

$$\text{Object classes} : \mathbf{c} \in \{LeverKnob, FixedKnob, RoundKnob\} \tag{5.2}$$

Since the "HSR Dataset" is not large enough to train a new model from scratch, transfer learning will be used to train the model for the above-mentioned class of objects. The common big datasets that are used for the task of object detection are Coco[41], Open-Image dataset(OID) to name a few. Coco dataset and OID already has the object "Door" as one of the categories so we will be using the models that have been trained on these two particular datasets.

Figure 5.1: Images taken from Toyota HSR for training the model. The different types of handles include lever handle, fixed handle, round handle.

5.1.3 Training results

The deep learning models were trained using Tensorflow Object Detection APIs [25]. The models were taken for each of 3 architectures: YOLO, SSD, FR-CNN architectures. Our models ware made to slightly overfit to our dataset and camera parameters as our requirement was for the model to work only for the Toyota HSR robot[1].

The models trained on our dataset were able to give very good detection accuracy specially for the lever handles and pull/fixed handles. Due to very less number of annotated round handles in our dataset, the generalization capabilities of the model especially in external images were quite limited. For very small pull handles(knobs), YOLO and SSD couldn't detect them due to very small image size after resizing the model resizes it for the input.

[1] https://github.com/njanirudh/Research-Development-HBRS

	AP	*AP-0.5*	*AP-0.75*	*AP small*	*AP medium*	*AP large*	*AR max=1*	*AR max=10*	*AR max=100*	*AR small*	*AR medium*	*AR large*
YOLO	0.64	-	-	-	-	-	-	-	-	-	-	
FR-CNN	0.904	0.980	0.977	0.646	0.894	0.936	0.574	0.923	0.924	0.700	0.911	0.956
SSD	0.702	0.96	0.781	-	0.715	0.708	0.503	0.752	0.752	-	0.757	0.751

Table 5.1: Object detection models used for detecting different types of handles in the robot camera frame

Figure 5.2: Different types of handles detected by the trained deep learning models.

Conclusion

Based on the accuracy required, ease of training, speed of detection and the simplicity of using the model in our system, we selected three common deep learning models. FR-CNN model showed the best accuracy for all the types of handles and for detecting small pull handles that were found in the lab environment. But FR-CNN models are very slow during inference and cannot be used for realtime tasks on slow hardware.

YOLO detector showed the best detection speed but the overall accuracy of detection and the detected bounding boxes are the lowest among the three models. YOLO model can be used in tasks which require reasonable accuracy but should run in realtime.

SSD architecture is a trade-off between the best accuracy and fast inference. This model can run nearly realtime on small hardware like the Toyota HSR robot, while also giving reasonably high detection accuracy and precise bounding box. SSD like YOLO also couldn't detect the very small pull handles present in the robot frame.

5.2 Manipulation

For manipulation, two learning-based methods are used for the task of grasping and manipulating different types of handles. The first method involves the use of simulation to train different policies and use them on the HSR robot in the simulated environment to open doors and manipulate different handles. The policy search algorithm used for learning the policy is Proximal Policy optimization. The second method involves the use of Toyota HSR robot to collect rollouts for grasping different types of handles. Maximum Likelihood Expectation (MLE) is then used for estimating the best distribution for sampling the goal position of the end-effector to successfully grasp handle. After grasping control based policies are used to manipulate the handle and open the door.

5.2.1 RL Simulation for learning to open doors

The simulation for the reinforcement algorithms were done in the DoorGym environment with the HSR robot model ported by us [2]. The porting of the robot URDF to the MuJoCo format involves adding the appropriate controls to the various joints present in the URDF file.

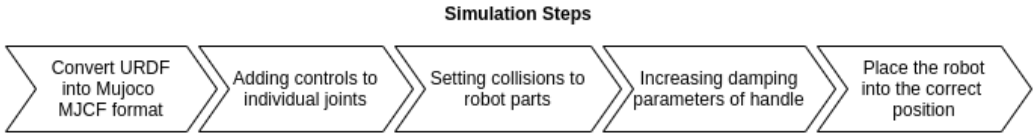


Figure 5.3: The images shows the steps involved in porting the URDF to the DoorGym simulation environment and running the simulation to obtain the policy.

The MJCF format is an XML file similar to the URDF format but with additional information required for the Mujoco simulation environment. Adding the appropriate controllers involves adding velocity and the position controls for each of the movable joints and setting appropriate controller parameters (kp,kv). To prevent one component from going through the other components in the simulators, the appropriate collision parameters should be added to the robot joints and the door and handle.

Few of the problems that were faced during the simulation and their status when writing the report are as follows:

- ✓ The handle would move on the door plane when the robot applied force on it. This problem was solved by increasing the damping parameters to very high values when generating the simulation environment.

[2] https://github.com/alex-mitrevski/DoorGym

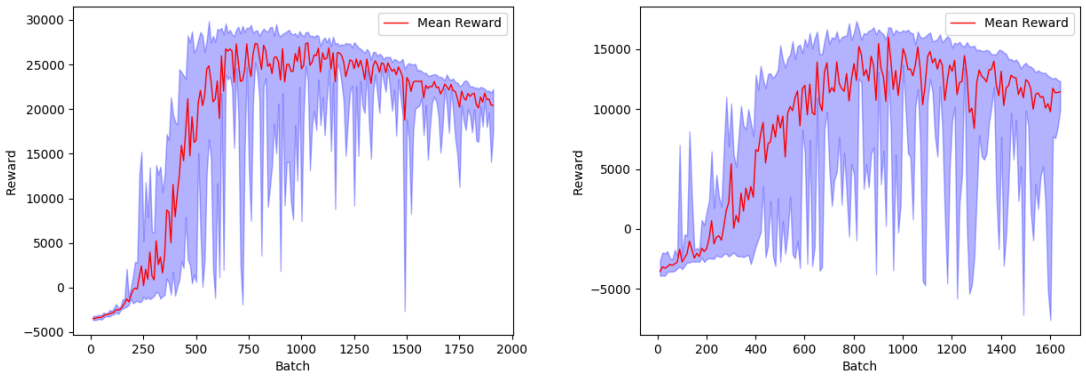
✓ The robot would not be placed correctly on the floor and would fly in the simulator. This solution to this problem involved changing the 'position' variable in the Toyota HSR XML file.

✓ The robot end-effector would detach from the robot body during the simulation. This was corrected by reducing the limit of the maximum height of the handle placed on the door in the simulated environment.

X The fingertips in the end-effectors of the robot couldn't be controlled in the simulator. This problem was due to a problem in the URDF and couldn't be solved.

X The robot learnt to push the handle but couldn't learn to open the door. This was a problem with the simulation environment which couldn't be solved.

The action space in the simulator is 5 DoF of the robot joint within the limits if the base movement is restricted or otherwise its is 8 DoF. The state space is restricted by the simulator to be all the possible states of the robot and the possible angles at which the door can be opened and the possible states depending on the handle.



(a) Lever handle - lr 10^{-4}

(b) Fixed handle - lr 10^{-4}

Figure 5.4: Graphs showing the increasing rewards with the number of iterations of training. The 'lr' represents the learning rate of the algorithm to learn the policy. The policy search algorithm used is PPO [4]

From the graph, it is observed that the reward obtained by the actor continuously increases with the number of rollouts. The red line indicates the mean reward archived by the robot and the blue region indicates the upper and lower range of the acquired reward. There were also cases when the simulator was not properly set up in terms of the door,

handle rigidity and collision but was still able to obtain positive rewards during training (section A.3).

The RL algorithm is able to learn a policy to grasp the handle in case of the fixed handle and grasp a handle and pull the lever downwards in case of a lever handle. It was observed that the robot in the simulation learnt to hook the handle using the fingers instead of grasping the handle correctly and pull/push the door. The source of this error can be pointed towards the physics setup of the simulator where the robot hand can slip between the narrow gap between the handle and the door plane and the reward function, which only rewards the grasping and pulling of the handle but not the actual trajectory followed by the end effector of the robot.

Conclusion

Reinforcement learning can be very helpful in robotics to learn generalizable manipulation policies. Training an RL policy is very hard due to the vast amount of data required to learn a good policy. To overcome the problems associated with the collection of rollouts on real robot hardware, high-quality simulation environments can be used. Simulation help in speeding up the data collection process and is safe to run as there is no real hardware involved. But simulations have its problems like it can't model all the physical forces acting on the robot accurately and doesn't consider the wear and tear on the robot hardware.

For the task of manipulating handles and opening doors, we have used a simulation environment called the DoorGym[61]. A model of Toyota HSR was ported and an RL policy was learnt in the simulation environment. It was observed that the robot was able to learn to push the handle down but was not able to pull/push the door open.

5.2.2 Maximum Likelihood Estimation for handle grasping

MLE was used for modelling the grasping end position of the end effector on the various kinds of handles. Few of the problem faced while collecting the rollouts are as following :

1. The robot couldn't open many spring-loaded drawers and doors which required very high force. To reduce the force that was required to open heavy drawers, a cloth or a small piece of plastic was placed such that the drawer would not close completely.

2. The robot gripper would often slip due to the smooth surface of the handles of the drawers. To reduce the smoothness and increase the friction between the robot gripper and the handle, a thin layer of rough masking tape was applied to the handle.

3. There were errors in placing the robot perfectly in its initial position during each rollout due to the manually moving the robot.

4. The robot perception failed to give the handle position correctly if multiple handles were detected in the camera frame. To prevent multiple handles from being detected in the camera frame, the extra handles were covered by a cloth.

From the collected rollouts Table 5.2 Table 5.3, the distance of the handle position in the 3D space x,y,z with respect to the robot base is obtained in meters. The final grasping position for the handle is sampled from a uniform distribution of the 3 coordinates with the limits obtained by the 3D bounding box around the handles.

The range within which the 3D points were sampled for a horizontal drawer handle (hh) when the robot is placed in approximately the same starting position is given by : $X^{hh}_{limits} = [0.8350, 0.8727]$, $Y^{hh}_{limits} = [-0.0192, 0.1378]$, $Z^{hh}_{limits} = [0.6588, 0.6663]$.

The range within which the 3D point were sampled for a vertical fridge handle (vh) when the robot is placed in approximately the same starting position is given by : $X^{vh}_{limits} = [0.8086, 0.8679]$, $Y^{vh}_{limits} = [0.0274, 0.1462]$, $Z^{vh}_{limits} = [0.6115, 0.6232]$.

Training rollouts by sampling from uniform distribution

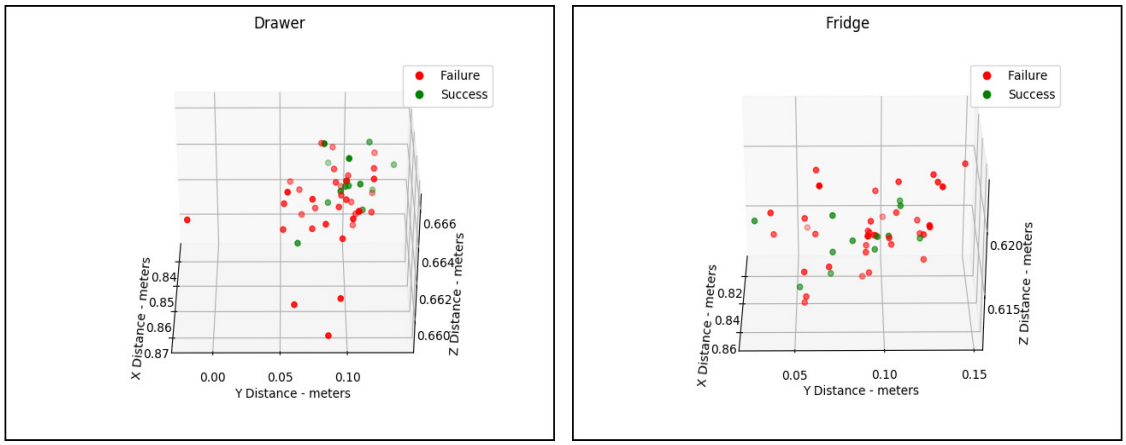


Figure 5.5: Scatter plot showing the various 3D grasping position with respect to the robot base for a horizontal drawer handle and vertical fridge handle.

It can be observed from the rollouts collected for training that in the case of a horizontal handle (Table 5.3) when the x-coordinate of goal position is very high, the robot end-effector cannot close as it is blocked by the drawer surface. This leads an unsuccessful handle grasp. When the value of 'x-coordinate' is very low, the end-effector of the robot either misses the handle completely or is unable to grasp the handle tightly and may lead to the handle slipping away during the backward movement. Similarly if the 'y-coordinate' is towards the extreme ends of the bounding-box, the end-effector may completely miss the handle or may not be able to grasp it tightly enough to open it. In the case of horizontal fixed handles, the robot grasp is quite robust to changes in the 'z-coordinate'. This is because if the handle lies between the open fingers of the robot end-effectors the probability of a successful grasp will only be dependent on the values of 'x' and 'y'.

Handle_X (meters)	Handle_Y (meters)	Handle_Z (meters)	Success	Notes on Success/Failure	Grasp	Opening
0.8611	0.1314	0.6224	0	x - failure , z - failure	-	-
0.8482	0.0826	0.6168	1	Drawer opened	✓	✓
0.8319	0.0958	0.6191	0	x - failure , y - failure , z - failure	-	-
0.8505	0.0369	0.6193	0	x - failure , z - failure	-	-
0.8456	0.1030	0.6169	1	Slip	✓	X
0.8480	0.1032	0.6170	0	z - failure	-	-
0.8487	0.1299	0.6215	0	y - failure	-	-
0.8679	0.0646	0.6232	0	y - failure	-	-
0.8403	0.0923	0.6166	0	z - failure	-	-
0.8364	0.1462	0.6217	0	z - failure	-	-
0.8494	0.0913	0.6173	0	x - failure , z - failure	-	-
0.8452	0.0274	0.6181	1	Slip	✓	X
0.8392	0.1075	0.6181	0	x - failure , z - failure	-	-
0.8384	0.1211	0.6163	0	z - failure	-	-
0.8124	0.1008	0.6149	0	z - failure	-	-
0.8449	0.1277	0.6217	0	x - failure , z - failure	-	-
0.8434	0.0556	0.6181	0	y - failure , z - failure	-	-
0.8347	0.1198	0.6170	0	x - failure	-	-
0.8301	0.1100	0.6181	1	Slip	✓	X
0.8086	0.0558	0.6136	0	x - failure	-	-
0.8585	0.1251	0.6190	0	y - failure	-	-
0.8653	0.0908	0.6183	0	z - failure	-	-
0.8463	0.0549	0.6115	0	x -failure , z - failure	-	-
0.8437	0.1046	0.6160	0	x -failure , z - failure	-	-
0.8415	0.1085	0.6208	0	z - failure	-	-
0.8476	0.0928	0.6183	0	x - failure , y - failure , z - failure	-	-
0.8620	0.1246	0.6195	0	Slip	✓	X
0.8378	0.0375	0.6162	0	x-failure	-	-
0.8502	0.1224	0.6175	0	z-failure	-	-
0.8389	0.0904	0.6154	0	x-failure	-	-
0.8435	0.1096	0.6191	1	Drawer opened	✓	✓
0.8446	0.0952	0.6157	1	Slip	✓	X
0.8392	0.1212	0.6160	1	Drawer opened	✓	✓
0.8404	0.0948	0.6164	0	x-failure , z-failure	-	-
0.8400	0.0914	0.6167	0	x-failure	-	-
0.8465	0.0919	0.6140	0	x-failure , z-failure	-	-
0.8540	0.0561	0.6129	0	y- failure , z-failure	-	-
0.8381	0.0622	0.6214	0	x - failure	-	-
0.8572	0.0693	0.6157	0	x - failure , z -failure	-	-
0.8488	0.0714	0.6189	1	Drawer opened	✓	✓
0.8489	0.0955	0.6173	0	x - failure , y -failure , z-failure	-	-
0.8423	0.1233	0.6146	0	x - failure , z -failure	-	-
0.8480	0.0524	0.6130	1	Slip	✓	X
0.8400	0.0700	0.6132	1	Drawer opened	✓	✓
0.8335	0.0883	0.6122	0	x - failure	-	-
0.8555	0.0553	0.6151	0	x - failure , z -failure	-	-
0.8395	0.0614	0.6164	0	z - failure	-	-
0.8458	0.0901	0.6156	0	x - failure	-	-
0.8498	0.0965	0.6173	1	Drawer opened	✓	✓
0.8474	0.0713	0.6158	1	Slip	✓	X

Table 5.2: Rollouts for training a model to grasp a vertically fixed handle of fridge.

Handle_X (meters)	Handle_Y (meters)	Handle_Z (meters)	Success	Notes on Success/Failure	Grasp	Opening
0.8419	0.1211	0.6625	0	y - failure	-	-
0.8480	0.1058	0.6615	0	x - failure	-	-
0.8382	0.1378	0.6636	1	Slipped	✓	X
0.8506	0.0830	0.6663	1	Drawer opened	✓	✓
0.8508	0.1000	0.6633	0	y - failure	-	-
0.8483	0.1006	0.6640	1	Slipped	✓	X
0.8466	0.0924	0.6637	0	x - failure	-	-
0.8549	0.0524	0.6622	0	x - failure	-	-
0.8475	0.0865	0.6627	1	Drawer opened	✓	✓
0.8443	0.0648	0.6630	0	x - failure	-	-
0.8394	0.1183	0.6650	1	Drawer opened	✓	✓
0.8432	0.0912	0.6640	0	x - failure	-	-
0.8485	0.1076	0.6622	0	x - failure	-	-
0.8488	0.1019	0.6638	1	Drawer opened	✓	✓
0.8579	0.0970	0.6621	0	x - failure	-	-
0.8450	0.1046	0.6624	0	y - failure	-	-
0.8498	0.1127	0.6626	1	Drawer opened	✓	✓
0.8513	0.1015	0.6656	1	Slip	✓	X
0.8445	0.1214	0.6642	0	x - failure	-	-
0.8398	0.1216	0.6624	1	Slip	✓	X
0.8516	0.0945	0.6630	0	x - failure , y -failure	-	-
0.8483	0.1100	0.6623	0	x - failure	-	-
0.8680	0.0952	0.6602	0	x - failure	-	-
0.8727	0.0860	0.6588	0	x - failure , y -failure	-	-
0.8546	0.0743	0.6622	0	x - failure , y -failure	-	-
0.8416	0.0578	0.6631	0	x - failure	-	-
0.8518	0.0956	0.6639	1	Drawer opened	✓	✓
0.8529	0.0745	0.6636	0	x-failure	-	-
0.8482	0.0965	0.6632	0	y-failure	-	-
0.8504	0.1104	0.6641	1	Slip	✓	X
0.8578	0.1090	0.6636	0	x - failure , y -failure	-	-
0.8383	0.0817	0.6648	0	x - failure , y - failure , z - failure	-	-
0.8500	0.1195	0.6625	0	x-failure	-	-
0.8483	0.0768	0.6625	0	x-failure	-	-
0.8350	0.0874	0.6633	1	Drawer opened	✓	✓
0.8368	0.0905	0.6644	0	x - failure , y -failure	-	-
0.8369	0.1228	0.6641	0	x - failure , y -failure	-	-
0.8472	0.0665	0.6620	0	x - failure	-	-
0.8545	0.0558	0.6642	0	x - failure , y -failure	-	-
0.8564	0.0844	0.6627	0	x - failure , y -failure	-	-
0.8568	0.0633	0.6617	1	Drawer opened	✓	✓
0.8437	0.1020	0.6637	0	x - failure , y -failure	-	-
0.8414	0.0969	0.6628	0	x - failure , y -failure	-	-
0.8683	0.0606	0.6599	0	x - failure , y -failure	-	-
0.8498	0.1207	0.6643	0	x - failure	-	-
0.8491	0.0993	0.6638	1	Slip	✓	X
0.8517	0.0531	0.6632	0	x - failure	-	-
0.8483	0.1001	0.6640	0	x - failure , y -failure	-	-
0.8572	0.1047	0.6631	0	y - failure	-	-
0.8568	-0.0192	0.6630	0	x - failure	-	-

Table 5.3: Rollouts for training a model to grasp a horizontal drawer handle.

In the case of the vertical fixed handles (Table 5.2) of the fridge, the final grasp is sensitive to the values along all the 3-axis. The sampled grasp position that lies in the extreme ends of the bounding box may lead to a complete failure in grasping.

Training a Weighted-Maximum Likelihood Estimation model

To find the best parameters of the underlying Gaussian distribution to model the grasp positions for the two types of handles horizontal fixed handle and vertical fixed handle, we are using a weighted version of the Maximum Likelihood Estimation. Weights have been assigned to the goal coordinates sample {x,y,z} based on the success or failure of grasping the handle successfully and pulling the handle. We use a simple extension of the "frequency weighting" where we try to increase the number of observations that give the expected results. The weighting for each sample is done manually by considering the coordinate that is leading to failure. For each coordinate of the goal sample, we will multiply it by a weight of 3 if it leads to a successful grasp or a 5 if it leads to a successful grasp along with a pulling action.

Let y_n be a single sample in the collected training set $\mathcal{Y}$. Where $y_n = \{x,y,z\}$ such that $y_n \in \mathcal{Y}$. $\forall y_n$ if y_n leads to a successful grasp add 3 samples same as y_n to the training set $\mathcal{Y}$. If y_n leads to a successful grasp and also a successful pulling action then add 5 samples of y_n to the training set $\mathcal{Y}$. If y_n leads to an unsuccessful grasp then consider the axis in which the grasp was unsuccessful. To each of the axis that was not the cause of failure (obtained manually by observation) assign, add a new sample with only the successful grasp coordinate and the values of the other of another axis to 0. Using the new weighted dataset ($\overline{\mathcal{Y}}$) MLE can be performed. The pseudo-code is given in Table 5.2.2.

Rollouts after sampling from the W-MLE model

Using the training rollouts we use MLE for learning the underlying gaussian distribution for sampling the best grasp position for horizontal and vertical handle. 50 rollouts were again taken, but this time with the goal coordinates sampled from the MLE learnt models and the results were compared to the goal positions sampled from the uniform distribution.

Handle_X (meters)	Handle_Y (meters)	Handle_Z (meters)	Success	Note	Grasp	Opening
0.6958	0.1695	0.6239	1	Slip	✓	X
0.7081	0.0628	0.6486	1	Slip	✓	X
0.7064	0.0940	0.6674	1	Slip	✓	X
0.6977	0.2441	0.6396	0	y - failure , z -failure	-	-
0.7288	0.1109	0.6662	0	x - failure	-	-
0.6903	0.0607	0.6655	0	x - failure	-	-
0.7077	0.1280	0.6212	1	Drawer opened	✓	✓
0.7012	0.1442	0.6318	1	Slip	✓	X
0.6955	0.1027	0.6556	0	x - failure	-	-
0.7045	0.1710	0.6634	1	Drawer opened	✓	✓
0.7089	0.1993	0.6487	0	x - failure	-	-
0.7135	0.0521	0.6379	1	Drawer opened	✓	✓
0.6929	0.1312	0.6418	0	x - failure	-	-
0.6904	0.1451	0.6490	0	x - failure	-	-
0.7252	0.1096	0.6435	0	x - failure	-	-
0.7182	0.1152	0.6487	1	Drawer opened	✓	✓
0.7168	0.1743	0.6499	1	Drawer opened	✓	✓
0.6814	0.0596	0.6419	0	x - failure	-	-
0.7122	0.0501	0.6640	1	Drawer opened	✓	✓
0.7108	0.1840	0.6635	1	Slip	✓	X
0.7184	0.1948	0.6449	1	Edge grip	✓	✓
0.7165	0.1411	0.6571	1	Drawer opened	✓	✓
0.7080	0.0891	0.6675	1	Drawer opened	✓	✓
0.6990	0.0581	0.6635	1	Slip	✓	X
0.6986	0.0962	0.6700	0	x - failure	-	-
0.6965	0.1762	0.6536	1	Drawer opened	-	-
0.7056	0.1139	0.6590	1	Slip	✓	X
0.6972	0.2339	0.6653	0	y - failure	-	-
0.7104	0.0319	0.6690	1	Drawer opened	✓	✓
0.7122	0.1574	0.6698	1	Slip	✓	X
0.7052	0.1349	0.6546	1	Drawer opened	✓	✓
0.7113	0.1959	0.6610	0	x - failure	-	-
0.6802	0.0506	0.6451	0	x - failure	-	-
0.7113	0.0888	0.6465	1	Slip	✓	X
0.7016	0.1003	0.6455	1	Slip	✓	X
0.6953	0.1230	0.6671	0	x - failure , z - failure	-	-
0.7047	0.1227	0.6541	1	Slip	✓	X
0.7057	0.0524	0.6578	1	Drawer opened	✓	✓
0.7012	0.0312	0.6764	1	Slip	✓	X
0.6967	0.0805	0.6119	0	y - failure , z -failure	-	-
0.7039	0.1255	0.6593	1	Slip	✓	X
0.6893	0.2050	0.6351	1	Slip	✓	X
0.6946	0.0736	0.6448	0	x - failure	-	-
0.6994	0.1210	0.6603	0	Slip	-	-
0.7088	0.1151	0.6510	1	Slip	✓	X
0.6779	0.0407	0.6443	0	x - failure	-	-
0.7148	0.0742	0.6483	1	Slip	✓	X
0.7318	0.1813	0.6588	0	x - failure	-	-
0.7008	0.1639	0.6449	1	Slip	✓	X
0.7240	0.2004	0.6384	0	x - failure , y - failure , z - failure	-	-

Table 5.4: Rollouts to grasp a drawer handle after sampling the goal position from MLE model.

Handle_X (meters)	Handle_Y (meters)	Handle_Z (meters)	Success	Note	Grasp	Opening
0.6724	0.1481	0.6321	0	x - failure, z - failure	-	-
0.7240	0.0975	0.7151	1	Slip	✓	X
0.6782	0.1327	0.5449	0	x - failure, y - failure	-	-
0.7234	0.0830	0.5773	0	✓- failure	-	-
0.6968	0.1737	0.6210	0	x - failure, y - failure, z - failure	-	-
0.6802	0.1063	0.6315	0	x - failure, y - failure, z - failure	-	-
0.7082	0.1135	0.6932	0	x - failure, z - failure	-	-
0.7189	0.1430	0.5191	1	Slip	✓	X
0.7043	0.1383	0.5986	1	Slip	✓	X
0.6941	0.1210	0.5308	1	Slip	✓	X
0.6885	0.1305	0.6412	1	Slip	✓	X
0.6722	0.1572	0.6016	1	Fridge opened	✓	✓
0.6968	0.1145	0.6481	1	Slip	✓	X
0.7120	0.1606	0.7196	0	y - failure	-	-
0.6964	0.1701	0.5432	0	x - failure , y- failure	-	-
0.7042	0.1066	0.6024	1	Slip	✓	X
0.7094	0.1085	0.5815	1	Slip	✓	X
0.7380	0.0768	0.5016	0	y - failure	-	-
0.7177	0.0683	0.5663	1	Slip	✓	X
0.7055	0.0998	0.6841	1	Slip	✓	X
0.6995	0.0245	0.6124	0	x - failure, y - failure	-	-
0.7155	0.1304	0.6042	1	Fridge opened	✓	✓
0.6896	0.1639	0.4928	0	x - failure, y - failure	-	-
0.7207	0.1387	0.6209	0	x - failure, y - failure	-	-
0.7028	0.1053	0.6135	1	Fridge opened	✓	✓
0.7089	0.0481	0.7274	1	Fridge opened	✓	✓
0.6627	0.0946	0.6692	0	x - failure, y - failure	-	-
0.6998	0.1490	0.6395	1	Fridge opened	✓	✓
0.7127	0.1451	0.5625	0	x - failure	-	-
0.7208	0.1309	0.8171	0	y - failure, z - failure	-	-
0.7165	0.0827	0.6746	0	x - failure	-	-
0.7087	0.1293	0.5720	1	Fridge opened	✓	✓
0.6833	0.1031	0.7535	0	x - failure, y - failure , z - failure	-	-
0.7361	0.0931	0.5766	0	x - failure	-	-
0.7137	0.1377	0.6283	0	x - failure	-	-
0.7207	0.1415	0.5029	0	x - failure	-	-
0.7133	0.1404	0.5929	1	Slip	✓	X
0.7071	0.0695	0.7235	0	x - failure, y - failure	-	-
0.7619	0.1422	0.6638	0	x - failure, y - failure	-	-
0.7168	0.0858	0.7068	0	x - failure	-	-
0.7069	0.0884	0.5661	1	Slip	✓	X
0.7246	0.1034	0.5853	1	Fridge opened	✓	✓
0.7460	0.0498	0.7056	0	x - failure, y - failure	-	-
0.7273	0.1757	0.5370	0	x - failure, y - failure, z - failure	-	-
0.7373	0.1037	0.5972	0	x - failure, y - failure	-	-
0.7614	0.1442	0.4404	0	x - failure, z - failure	-	-
0.7215	0.1540	0.6592	0	x - failure, y - failure	-	-
0.7317	0.0750	0.6515	0	x - failure	-	-
0.7265	0.1329	0.4491	0	z - failure	-	-
0.7121	0.1582	0.6287	1	Slip	✓	X

Table 5.5: Rollouts to grasp a fridge handle after sampling the goal position from MLE model.

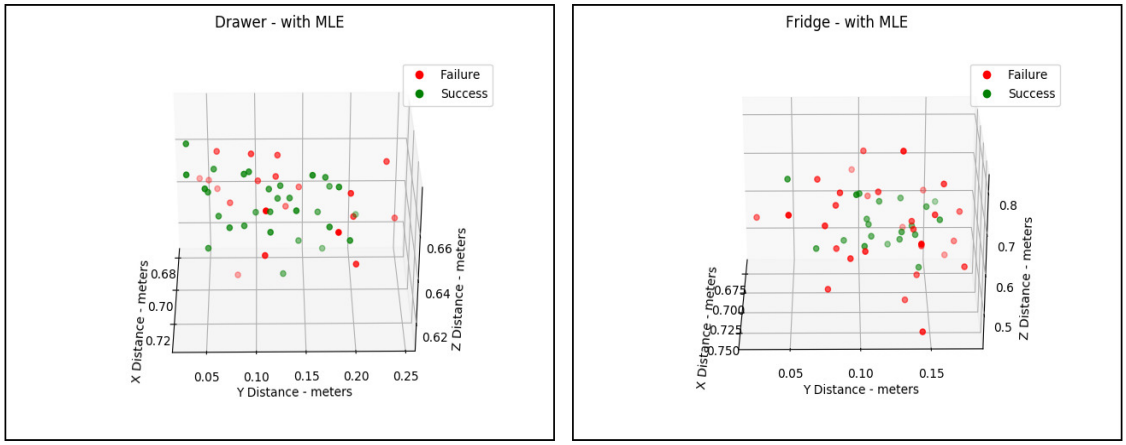


Figure 5.6: Scatter plot showing the various 3D grasping position with respect to the robot base for a horizontal drawer handle and vertical fridge handle.

	Sampling	**Successful grasp**	**Successful pull**	**Total Success (out of 50 rollouts)**	**Total Success (percentage)**
Drawer handle	Uniform distribution	14	7	14	28%
	Gaussian distribution	30	11	30	60%
Fridge handle	Uniform distribution	13	6	13	26%
	Gaussian distribution	20	7	20	40%

Table 5.6: Table showing the success of grasp and pull task by sampling goal position using a uniform distribution and a Gaussian distribution learnt by MLE. The total number of collected rollouts are 50 for each experiment.

The observed results show that the MLE model was able to improve the sampling of goal position giving a total success of 14/50 to 30/50 for the horizontal drawer handle and from 13/50 to 20/50 for the vertical drawer handle.

It was observed that the robot was able to learn the best value of the coordinates such that it doesn't overshoot the boundaries of the handle. In case of the horizontal drawer handle, most of the failure was in the 'x-axis'. The robot was not able to grasp the grasp the handle mainly due to the hardware constraints of the robot and the smooth handle which led to slipping. This was also the primary reason that in may cases the grasp was successful but the pulling action failed. In the case of vertical handle there was no particular trends observed that would be responsible for the failure. Some of the failure can be attributed to the errors in the bounding box obtained around the handle by the perception.

Algorithm 2 Weighted Maximum Likelihood Estimation

Data: List of grasp position coordinates, $y_i \in \mathcal{Y}$ where $y_i = \{x, y, z\}$
Data: Success of each sample in $\mathcal{Y}$, $\mathcal{S}$
Data: Weighted list of samples $\overline{\mathcal{Y}}$
Result: The parameters of gaussian distribution ($\mathcal{N}$), μ and σ^2
for *Each sample* $y_i \in \mathcal{Y}$ **do**
  if y_i *leads to a successful grasp or successful pull* **then**
    if y_i *leads to only a successful grasp* **then**
      Add 3 instances of y_i to $\overline{\mathcal{Y}}$;
    else
      Add 5 instances of y_i to $\overline{\mathcal{Y}}$;
    end
  else
    Consider the axis along which the failure occurred.
    Replace the value along that axis to zero in y_i.
    Add the modified y_i to $\overline{\mathcal{Y}}$.
  end
end
Use the $\overline{\mathcal{Y}}$ to perform MLE and find parameters of $\mathcal{N}$

Conclusion

Using the assumption that the underlying model of the successful grasp positions is a Gaussian distribution, we are able to train an MLE model to improve the success of sampling good goal positions.
The training prior is assumed to be sampled from a uniform distribution of x,y,z within the limits of the values obtained from the bounding box of the detected handle by perception. We collected 50 rollouts for each type of fixed handles for training the model on a Toyota HSR robot.
The MLE model was able to quantifiably improve the grasp position for horizontal handles from an initial value of 28% when sampled from a uniform distribution to 60% when sampled from gaussian distribution. In the case of vertical handles, the results we obtained saw an increase from the 26% to 40% after using the MLE model. The errors were mainly attributed to bounding-box obtained by perception.

6

Conclusions

The most exciting phrase to hear in science, the one that heralds new discoveries, is not Eureka! (I found it!) but Thats funny

Isaac Asimov
Scientist, Si-Fi Author

Although a lot of progress has been made in the domain of manipulation of handles, most of the systems are limited to work on a single robot and within very constrained and prior known environments. Most of the methods are not robust toward different types of lighting conditions or unexpected kinematic properties of the doors and cabinets. Frameworks like ROS have made it easier to make modular and reusable software, but there are still open-ended questions regarding the best algorithms to solve the task of perception and manipulation for complex tasks like manipulation of handles. This main questions that this report tries to address are :

1. What are the subtasks that "Manipulation of Handles" can be divided into?

2. Which perception algorithms can be used for the task of handle detection and handle pose estimation and what are the advantages and disadvantages of the methods?

3. What are the different classes of manipulation algorithms that have been used in the literature?

6.1 Contributions

The contribution of this report is multifold. The major contribution involves the survey and summarizing of the various techniques that are used previously for manipulating handles, a new deep learning model trained for detecting doors, handles in the image frame and porting the Toyota HSR robot URDF model to the simulation environment DoorGym [61]. Individually each of the tasks like perception and manipulation has a very vast literature, so to simplify the problem the main contributions are focused on the task of manipulation of handles. the contributions are summarized as follows :

Survey of papers related to manipulation of handles and doors The task of manipulation of handles is a very important skill for a robot to show autonomous capabilities. This problem has been studied for many decades with many types of algorithms that have been tried to tackle the problem. This report divides the task of manipulating handles into 2 distinct subtasks, perception [section 3.1] and manipulation subtask[section 3.2]. The subtasks are further divided logically based on the relevant literature and tabulated for perception task[Table 3.1], manipulation task[Table 3.2]. In addition to this, human guidance[Table 3.3] methods that can be used for solving the problem is also given. This tabulation will simplify the study of all the possible methods and give a brief description of the advantages and the disadvantages of each of the methods. This report can be used as a brief introduction for performing the literature survey and for finding new avenues of research in the mentioned topic.

Door and handle detection models for object detection Object detection is a primary requirement for the robot to detect the object in the camera frame. For the task of handle detection this report provides an extension of the "handle dataset" with labelled images taken from the Toyota HSR robot. Three deep learning models were also trained using the above-mentioned dataset. This model can be used on any robot for detecting handle in the perception subtask.

Porting the URDF model to DoorGym DoorGym is a useful simulation environment for trying out new reinforcement algorithms. Simulation environment that was given by the authors only had Baxter and PR2 models as part of the simulator. For extending the simulator for the Toyota HSR, we ported the URDF model to a format that was compatible for MuJoCo. The HSR simulator can be used to test various types of policy search algorithms without having the problems associated with running the algorithms on the actual robot hardware.

Training the HSR robot in the DoorGym The ported HSR robot was simulated in the DoorGym environment to train a policy to manipulate handles. The trained PPO policy was tested with the Toyota HSR in the DoorGym with partial success. The robot was able to learn to twist the lever handle in different initial robot pose and for lever handles at different heights on the door. Although the robot was able to manipulate the handle, it couldn't learn to grasp the handle and pull the door open as expected.

Training an MLE for grasping different types of handles To train an MLE model for sampling the best grasp position from the detected handle bounding box, we have collected training rollouts for different types of handles (vertical fixed handle, horizontal fixed handles). Using the dataset, MLE was used to learn the parameters of the gaussian distribution such that the sampled grasp pose will be the best for grasping the handle.

6.2 Future work

Manipulating handles is a complex task that requires knowledge and input from many different fields of engineering. The current algorithms are able to solve a highly simplified version of the problem with assumptions that make it impractical to run in the real-world dynamic human environment. This report tries to consolidate the methods that have been used in the past and opens up the possible combinations of methods that can be used in future to solve the problems robustly. The many promising methods that are tabulated can be used as a general reference for further research.

This report provides a solution for the individual subtasks of the handle manipulation pipeline. The annotated dataset of various types of handles and the trained deep learning models are provided. Future work can include training a more generalized deep learning model for the detecting handles which can run on a wider variety of robots.

The HSR robot ported into the DoorGym environment can be used in the future for training the robot on a wider variety of doors like sliding doors and vertically lifting doors. The simulated door can also be modified to have a lock or be spring-loaded to make it closer to the real-world environment. The simulation environment can also be extended for a wider variety of tasks like opening a drawer etc. The policy learnt from the simulator can be for "sim to real transfer"; ie. Running the policy learnt in the simulation on a real robot.

The rollouts collected for performing MLE can also be used for learning other types models like Gaussian processes and reinforcement learning based policies. Different models can then be compared for testing their robustness and obtaining the best performing model that runs well for all the handles.

6.3 Lessons learned

Over three decades worth of research has shown that the creativity of nature has still not been surpassed by the current engineering ingenuity. Although robot hardware and algorithms have vastly improved in the past couple of years, we are still bounded by the complexity of dynamic variables in a real-world scenario. There has been a drastic increase in the number of "learning" algorithms that are being developed to improve the generalizability of performing the task. These learning algorithms are designed to improve itself by learning better representation of the task or by improving the parameters to reach the required goal.

Deep Learning has shown immense success in solving many complex tasks especially in the field of computer vision with few results showing superhuman abilities. Similarly, Reinforcement Learning has shown very promising results in the field of manipulation but suffers from the problem of requiring a very high number of samples to train itself for a given task. There are many tricks like using data-augmentation in computer vision and simulation in RL to make the algorithms more practical.

The task of "Manipulating handles" if solved in the general sense can mean a nearly perfect understanding of the object detection, pose estimation and manipulation of articulated objects along with the intelligent coupling between the two subtasks. Although the task tackled in this report is far from solved, each new algorithm designed and any breakthrough in solving the subtasks will contribute to a more intelligent and autonomous robot in the future.

A

Design Details

A.1 Robot Hardware

The robot that is the focus for performing all the experiments in the simulation is the Toyota Human Service Robot(HSR) [67]. The total DoF of the robot without the base is 5 and considering movable base is 8.

size	ϕ 430 x 1,005 (1,350)
weight	37 Kg
arm length	600 mm
shoulder height	340 1,030 mm
grasped object	1.2 Kg Weight 130 mm width
maximum velocity	0.8 km/h
performance	5 mm difference in level 5 degree slope

Figure A.1: Details about the Toyota Human Support Robots (HSR)[67]

.

The robot also has many types of perception sensors like the stereo camera, an arm camera and the RGB-D sensor on top of the head which can be used for the perception and a laser sensor on the base to measure distance to obstacles and for path planning.

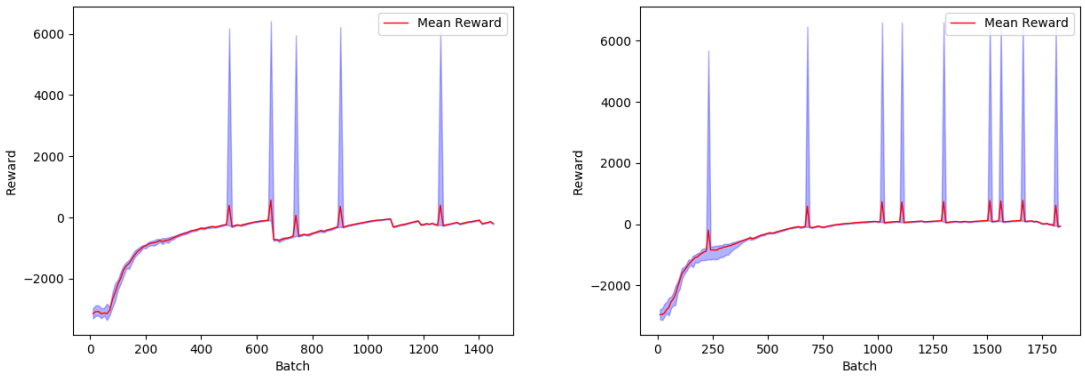
A.2 Joints with range for HSR simulation

Joint name	***Joint Type***	***Axis***	***Range***	***Units***
base_roll_joint	rotation	z	{-3.14 , 3.14}	radians
base_link_x_joint	slide	x	{-1000 , +1000}	meters
base_link_y_joint	slide	y	{-1000 , +1000}	meters
base_link_theta_joint	rotation	z	{-3.14 , 3.14}	radians
base_r_drive_wheel_joint	rotation	y	{-3.14 , 3.14}	radians
base_l_drive_wheel_joint	rotation	y	{-3.14 , 3.14}	radians
torso_lift_joint	slide	z	{0 , 0.345}	meters
arm_lift_joint	slide	z	{0 , 0.69}	meters
head_pan_joint	rotation	z	{-3.84,1.75}	radians
head_tilt_joint	rotation	-y	{-1.57 , 0.5}	radians
arm_flex_joint	rotation	-y	{-2.62 , 0}	radians
arm_roll_joint	rotation	z	{-2.09 , 3.84}	radians
wrist_flex_joint	rotation	-y	{-1.92 ,1.22}	radians
wrist_roll_joint	rotation	z	{-1.92 , 3.67}	radians
hand_r_distal_joint	rotation	-x	{-1.24 , 0.798}	radians
hand_l_distal_joint	rotation	x	{-1.24 , 0.798}	radians

Table A.1: Robot joints with axis of movement and range for the simulator

The above table shows the joints with the respective limits and joint type that must be added to the MJCF for controlling the robot in the mujoco simulation environment.

A.3 Unsuccessfully trained RL policy



(a) Lever handle - lr 10^{-3}

(b) Lever handle - lr 10^{-4}

Figure A.2: Increasing reward can be observed even in cases where the simulator is not setup correctly.

Figure A.2 shows that the rewards in the simulation increases even-though the simulation environment containing the Toyota HSR, the door and the handle are not setup correctly. In the above cases the collision parameters between the robot end effectors and the door handle. This cause the fingers of the robot to pass through the handle leading to a grasping failure. The increasing reward may be attributed to the distance parameter present in the reward function.

It is very important to run the final policy after training and observe if the robot is functioning as desired. This will be helpful in debugging the unrealistic behavior of the robot in the simulator.

A.4 MLE Graphs [1]

A.4.1 Fridge handle grasp samples.

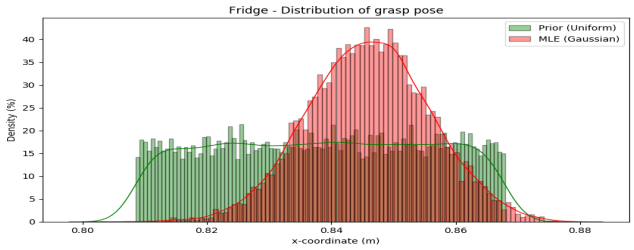


Figure A.3: Fridge: X-coordinates sampled from different distributions.

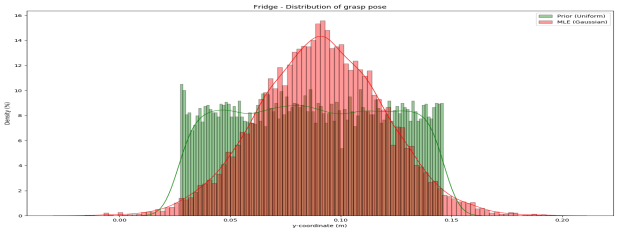


Figure A.4: Fridge: Y-coordinates sampled from different distributions.

.

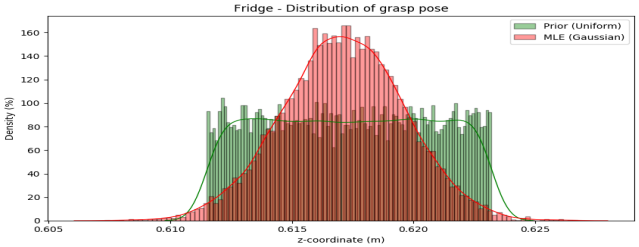


Figure A.5: Fridge: Z-coordinates sampled from different distributions.

[1] Higher quality graphs in https://github.com/njanirudh/Research-Development-HBRS/

A.4.2 Drawer handle grasp samples.

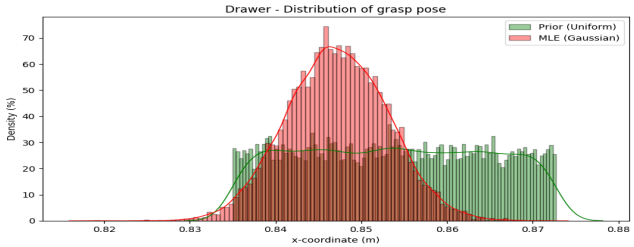


Figure A.6: Drawer: X-coordinates sampled from different distributions.

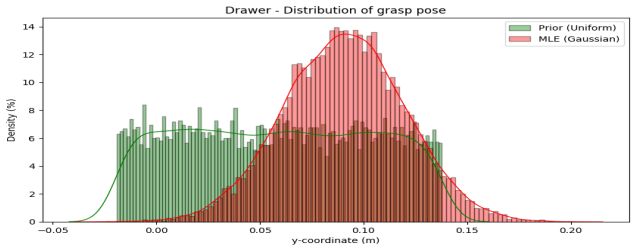


Figure A.7: Drawer: Y-coordinates sampled from different distributions.

.

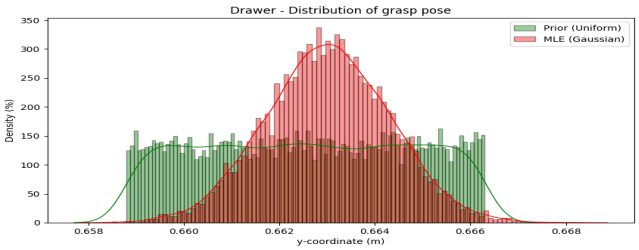


Figure A.8: Drawer: Z-coordinates sampled from different distributions.

B

Proofs

B.1 Policy gradient - Extended Proof

[1]

From the general definition of expectation

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

$$\mathbb{E}[g(x)] = \int g(x)p(x)dx$$

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}[R(\tau)]$$

$$\nabla_\theta J(\theta) = \nabla_\theta \int R(\tau)P(\tau|\theta)d\tau$$

$$\nabla_\theta J(\theta) = \int \nabla_\theta R(\tau)P(\tau|\theta)d\tau$$

Multiplying and Dividing by $P(\tau|\theta)$

$$\nabla_\theta J(\theta) = \int R(\tau)P(\tau|\theta)\frac{P(\nabla_\theta\tau|\theta)}{P(\tau|\theta)}d\tau^2$$

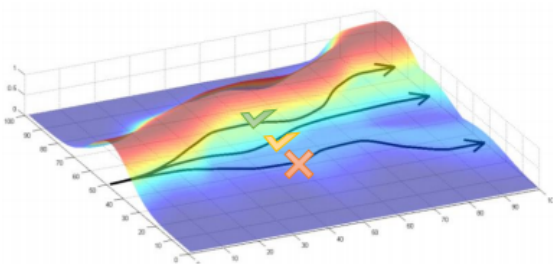


Figure B.1: Expectation of rewards improve chances of selecting good trajectory and reducing chances of bad trajectory[1]

[1]The steps of this proof were compiled along with the adviser(Alex Mitrevski)[4][1]

Using the definition of the differential $\frac{d}{dx}\log(x) = \frac{\log(x)}{x}$

$$\nabla_\theta J(\theta) = \int R(\tau)P(\tau|\theta)\nabla_\theta \log P(\tau|\theta)d\tau$$

$$\nabla_\theta J(\theta) = \int P(\tau|\theta)\nabla_\theta \log P(\tau|\theta)R(\tau)d\tau$$

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \log P(\tau|\theta)R(\tau)]$$

$$P(\tau|\theta) = P(s_0)\prod_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

Multiplying and Dividing by $P(\tau|\theta)$

$$\nabla_\theta J(\theta) = \int R(\tau)P(\tau|\theta)\frac{P(\nabla_\theta\tau|\theta)}{P(\tau|\theta)}d\tau$$

Using the definition of the differential $\frac{d}{dx}\log(x) = \frac{\log(x)}{x}$

$$\nabla_\theta J(\theta) = \int R(\tau)P(\tau|\theta)\nabla_\theta \log P(\tau|\theta)d\tau$$

$$\nabla_\theta J(\theta) = \int P(\tau|\theta)\nabla_\theta \log P(\tau|\theta)R(\tau)d\tau$$

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \log P(\tau|\theta)R(\tau)]$$

$$P(\tau|\theta) = P(s_0)\prod_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

(Taking log on both sides.) $\log(P(\tau|\theta)) = log(P(s_0)\prod_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t))$

We know that $\log(ab) = \log(a) + \log(b)$

$$\log P(\tau|\theta) = log(P(s_0)) + log(\prod_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t))$$

$$\log P(\tau|\theta) = log(P(s_0)) + log(P(s_1|s_0, a_0)\pi_\theta(a_0|s_0) * P(s_2|s_1, a_1)\pi_\theta(a_1|s_1) * .... * P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t))$$

$$\log P(\tau|\theta) = \log(P(s_0)) + \log(P(s_1|s_0, a_0)\pi_\theta(a_0|s_0)) + ... + \log(P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t))$$

$$\log P(\tau|\theta) = \log P(s_0) + \sum_{t=0}^{T} \log P(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t)$$

$$\therefore \qquad \nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \log P(\tau|\theta)R(\tau)]$$

References

[1] Deep reinforcement learning. URL `http://rail.eecs.berkeley.edu/deeprlcourse-fa18/`.

[2] Sim to reality. URL `http://josh-tobin.com/assets/pdf/randomization_and_the_reality_gap.pdf`.

[3] CS231n Convolutional Neural Networks for Visual Recognition. URL `http://cs231n.github.io/transfer-learning/`.

[4] Spinning up openai. URL `https://spinningup.openai.com/en/latest/algorithms/ppo.html`.

[5] Miguel Arduengo, Carme Torras, and Luis Sentis. A versatile framework for robust and adaptive door operation with a mobile manipulator robot. *ArXiv*, abs/1902.09051, 2019.

[6] Benjamin Axelrod and Wesley H. Huang. Autonomous door opening and traversal. *2015 IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*, pages 1–6, 2015.

[7] Nandan Banerjee, Xianchao Long, Ruixiang Du, Felipe Polido, Siyuan Feng, Christopher G. Atkeson, Michael A. Gennert, and Taskin Padir. Human-supervised control of the atlas humanoid robot for traversing doors. *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 722–729, 2015.

[8] Christopher M. Bishop and Nasser M. Nasrabadi. Pattern recognition and machine learning. *J. Electronic Imaging*, 16:49901, 2007.

[9] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *ArXiv*, abs/1606.01540, 2016.

[10] Ian M. Bullock and Aaron M. Dollar. Classifying human manipulation behavior. *2011 IEEE International Conference on Rehabilitation Robotics*, pages 1–6, 2011.

[11] Haseru Chen, Youhei Kakiuchi, Manabu Saito, Kei Okada, and Masayuki Inaba. View-based multi-touch gesture interface for furniture manipulation robots. *Advanced Robotics and its Social Impacts*, pages 39–42, 2011.

[12] Wei Chen, Ting Qu, Yimin Zhou, Kaijian Weng, Gang Wang, and Guoqiang Fu. Door recognition and deep learning algorithm for visual based robot navigation. *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*, pages 1793–1798, 2014.

[13] Sunglok Choi, Taemin Kim, and Wonpil Yu. Performance evaluation of ransac family. In *BMVC*, 2009.

[14] Anthony M. Dearden and Yiannis Demiris. Learning forward models for robots. In *IJCAI*, 2005.

[15] Felix Endres, Jeffrey C. Trinkle, and Wolfram Burgard. Learning the dynamics of doors for robotic manipulation. *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3543–3549, 2013.

[16] Peter Englert and Marc Toussaint. Combined optimization and reinforcement learning for manipulation skills. In *Robotics: Science and Systems*, 2016.

[17] Peter Englert and Marc Toussaint. Learning manipulation skills from a single demonstration. *I. J. Robotics Res.*, 37:137–154, 2018.

[18] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24:381–395, 1981.

[19] Doug Forsyth and Jean Ponce. Computer vision: A modern approach prentice hall. 2003.

[20] Ross B. Girshick. Fast r-cnn. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.

[21] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jagannath Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2013.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[23] Shixiang Gu, Ethan Holly, Timothy P. Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3389–3396, 2016.

[24] G. Clark Haynes, David Stager, Anthony Stentz, Michael Vande Weghe, Brian Zajac, Herman Herman, Alonzo Kelly, Eric Meyhofer, Dean M. Anderson, Dane Bennington, Jordan Brindza, David Butterworth, Christopher M. Dellin, Michael David George, Jose Gonzalez-Mora, Morgan Jones, Prathamesh Kini, Michel Laverne, Nick Letwin, Eric Perko, Chris Pinkston, David Rice, Justin Scheifflee, Kyle Strabala, Mark Waldbaum, and Randy Warner. Developing a robust disaster response robot: Chimp and the robotics challenge. *J. Field Robotics*, 34:281–304, 2017.

[25] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara Balan, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3296–3297, 2016.

[26] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: Learning attractor models for motor behaviors. *Neural Computation*, 25:328–373, 2013.

[27] Advait Jain and Charles C. Kemp. Behaviors for robust door opening and doorway traversal with a force-sensing mobile manipulator. 2008.

[28] Advait Jain and Charles C. Kemp. Pulling open novel doors and drawers with equilibrium point control. *2009 9th IEEE-RAS International Conference on Humanoid Robots*, pages 498–505, 2009.

[29] Yiannis Karayiannidis, Christian Smith, Francisco E. Vina, Petter Ögren, and Danica Kragic. Model-free robot manipulation of doors and drawers by means of fixed-grasps. *2013 IEEE International Conference on Robotics and Automation*, pages 4485–4492, 2013.

[30] Dov Katz, Yuri Pyuro, and Oliver Brock. Learning to manipulate articulated objects in unstructured environments using a grounded relational representation. In *Robotics: Science and Systems*, 2008.

[31] Ellen Klingbeil, Ashutosh Saxena, and Andrew Y. Ng. Learning to open new doors.

2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 2751–2757, 2010.

[32] Gerhard Kniewasser. Reinforcement learning with dynamic movement primitives-dmps. 2013.

[33] Jens Kober and Jan Peters. Policy search for motor primitives in robotics. In *NIPS 2008*, 2008.

[34] Petar Kormushev, Sylvain Calinon, Darwin G. Caldwell, and Barkan Ugurlu. Challenges for the policy representation when applying reinforcement learning in robotics. *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2012.

[35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60:84–90, 2012.

[36] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521: 436–444, 2015.

[37] Sergey Levine, Nolan Wagener, and Pieter Abbeel. Learning contact-rich manipulation skills with guided policy search. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 156–163, 2015.

[38] Jiajun Li, Jianguo Tao, Liang Ding, Haibo Gao, Zongquan Deng, and Ke rui Xia. Twisting door handles and pulling open doors with a mobile manipulator. *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 686–691, 2015.

[39] Jiajun Li, Jianguo Tao, Liang Ding, Haibo Gao, Zongquan Deng, and Yuehua Wu. Rgbd-based parameter extraction for door opening tasks with human assists in nuclear rescue. *2016 International Conference on Advanced Robotics and Mechatronics (ICARM)*, pages 101–106, 2016.

[40] Zhou Li, Zeyang Xia, Guodong Chen, Yangzhou Gan, Ying Hu, and Jianwei Zhang. Kinect-based robotic manipulation for door opening. *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 656–660, 2013.

[41] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014.

[42] Weiwei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In *ECCV*, 2016.

[43] Adrian Llopart, Ole Ravn, and Nils. A. Andersen. Door and cabinet recognition using convolutional neural nets and real-time method fusion for handle detection and grasping. *2017 3rd International Conference on Control, Automation and Robotics (ICCAR)*, pages 144–149, 2017.

[44] Wim Meeussen, Melonee Wise, Stuart Glaser, Sachin Chitta, Conor McGann, Patrick Mihelich, Eitan Marder-Eppstein, Marius Muja, Victor Eruhimov, Tully Foote, John M. Hsu, Radu Bogdan Rusu, Bhaskara Marthi, Gary R. Bradski, Kurt Konolige, Brian P. Gerkey, and Eric Berger. Autonomous door opening and plugging in with a personal robot. *2010 IEEE International Conference on Robotics and Automation*, pages 729–736, 2010.

[45] Kotaro Nagahama, Keisuke Takeshita, Hiroaki Yaguchi, Kimitoshi Yamazaki, Tomohide Yamamoto, and Masayuki Inaba. Estimating door shape and manipulation model for daily assistive robots based on the integration of visual and touch information. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7660–7666, 2018.

[46] Bojan Nemec, Leon Zlajpah, and Ale Ude. Door opening by joining reinforcement learning and intelligent control. *2017 18th International Conference on Advanced Robotics (ICAR)*, pages 222–228, 2017.

[47] Peter Pastor, Heiko Hoffmann, Tamim Asfour, and Stefan Schaal. Learning and generalization of motor skills by learning from demonstration. *2009 IEEE International Conference on Robotics and Automation*, pages 763–768, 2009.

[48] Georgia Peleka, Andreas Kargakos, Evangelos Skartados, Ioannis Kostavelis, Dimitrios Giakoumis, Iason Sarantopoulos, Zoe Doulgeri, Michalis Foukarakis, Margherita Antona, Sandra Hirche, Emanuele Ruffaldi, Bartlomiej Stanczyk, Anastasios Zompas, Joan Hernández-Farigola, Natalia Roberto, Konrad Rejdak, and Dimitrios Tzovaras. Ramcip - a service robot for mci patients at home. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9, 2018.

[49] Anna Petrovskaya and Andrew Y. Ng. Probabilistic mobile manipulation in dynamic environments, with application to opening doors. In *IJCAI*, 2007.

[50] Rahul Raguram, Jan-Michael Frahm, and Marc Pollefeys. A comparative analysis of ransac techniques leading to adaptive real-time random sample consensus. In *ECCV*, 2008.

[51] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2015.

[52] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:1137–1149, 2015.

[53] Thomas Rühr, Jürgen Sturm, Dejan Pangercic, Michael Beetz, and Daniel Cremers. A generalized framework for opening doors and drawers in kitchen environments. *2012 IEEE International Conference on Robotics and Automation*, pages 3852–3858, 2012.

[54] Radu Bogdan Rusu, Wim Meeussen, Sachin Chitta, and Michael Beetz. Laser-based perception for door and handle identification. *2009 International Conference on Advanced Robotics*, pages 1–8, 2009.

[55] Radu Bogdan Rusu, Gary R. Bradski, Romain Thibaux, and John M. Hsu. Fast 3d recognition and pose using the viewpoint feature histogram. *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2155–2162, 2010.

[56] Stefan Schaal, Jan Peters, Jun Nakanishi, and Auke Jan Ijspeert. Control, planning, learning, and imitation with dynamic movement primitives. 2003.

[57] Jörg Stückler, David Droeschel, Kathrin Gräve, Dirk Holz, Jochen Kläß, Michael Schreiber, Ricarda Steffens, and Seven Behnke. Towards robust mobility, flexible object manipulation, and intuitive multimodal interaction for domestic service robots. In *RoboCup*, 2011.

[58] Hong-Rui Su and Kuo-Yi Chen. Design and implementation of a mobile robot with autonomous door opening ability. *2017 International Conference on Fuzzy Theory and Its Applications (iFUZZY)*, pages 1–6, 2017.

[59] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL `http://incompleteideas.net/book/the-book-2nd.html`.

[60] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[61] Yusuke Urakami, Alec Hodgkinson, Casey Carlin, Randall Leu, Luca Rigazio, and Pieter Abbeel. Doorgym: A scalable door opening environment and baseline agent. *ArXiv*, abs/1908.01887, 2019.

[62] Toshihiko Watanabe. A fuzzy ransac algorithm based on reinforcement learning concept. *2013 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–6, 2013.

[63] Tim Welschehold, Christian Dornhege, and Wolfram Burgard. Learning manipulation actions from human demonstrations. *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3772–3777, 2016.

[64] Tim Welschehold, Christian Dornhege, and Wolfram Burgard. Learning mobile manipulation actions from human demonstrations. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3196–3201, 2017.

[65] Ping Wu, Yang Cao, Yuqing He, and Decai Li. Vision-based robot path planning with deep learning. In *ICVS*, 2017.

[66] Ali Yahya, Adrian Li, Mrinal Kalakrishnan, Yevgen Chebotar, and Sergey Levine. Collective robot reinforcement learning with distributed asynchronous guided policy search. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 79–86, 2016.

[67] Takashi Yamamoto, Koji Terada, Akiyoshi Ochiai, Fuminori Saito, Yoshiaki Asahara, and Kazuto Murase. Development of the research platform of a domestic mobile manipulator utilized for international competition and field test. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7675–7682, 2018.

[68] Michael Ying Yang and Wolfgang Förstner. Plane detection in point cloud data. 2010.

[69] Zhong-Qiu Zhao, Peng Zheng, Shou tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, 2018.

[70] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey. *ArXiv*, abs/1905.05055, 2019.