OBJECT AND RELATION-CENTRIC REASONING OF ACTION EFFECTS IN PUSH MANIPULATION TASKS

by

Ahmet Ercan Tekden

B.S., Computer Engineering, Boğaziçi University, 2017

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2020

ACKNOWLEDGEMENTS

I would like to thank my advisor Assist. Prof. Emre Uğur for his support and guidance throughout my master's student. His very kind support, advice, encouragement, and motivation helped me a lot to be able to complete this master thesis work. I would like to thank Assoc. Prof Aykut Erdem for cosupervising my research and giving many helpful advices. I also would like to thank Assoc. Prof. Erkut Erdem for his support and contribution to my research. My sincere gratitude to Assist. Prof. İnci Meliha Baytaş and Assist. Prof. Fatma Güney for kindly accepting to be in my thesis jury.

My sincerest gratitude to Mert İmre, Serkan Buğur, Melisa İdil Şener, Yunus Şeker, Tuna Han Salih Meral and Safa Andaç for their friendship, joyful coffee breaks and support through my masters degree. I feel privileged to be able to work with the members of the COLORS research group. Many thanks to Mete Tuluhan Akbulut, Alper Ahmetoğlu, Utku Bozdoğan, Ece Ada for the both academic, casual discussions.

Finally, I would like to thank my parents, my brothers for their genuine support and encouragement during my whole life. I would have never been able to persevere in my education without them.

This research has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 731761, IMAGINE; and supported by a Tubitak 2210-A scholarship awarded to me.

The numerical calculations reported in this thesis were partially performed at TUBITAK ULAKBIM, High Performance and Grid Computing Center (TRUBA resources).

ABSTRACT

OBJECT AND RELATION-CENTRIC REASONING OF ACTION EFFECTS IN PUSH MANIPULATION TASKS

In complex robotic systems, prediction of effects is a challenging problem when the number of objects varies, especially in the presence of rich and various interactions among these objects. To be able to model such systems, the representation of data should be able to sufficiently encode multiple objects and interactions between them. In this thesis, we first show our initial research on effect prediction on objects with various shapes. Then we propose a Graph Neural Network based framework, Belief Regulated Dual Propagation Networks (BRDPN), a general-purpose learnable physics engine. Our framework consists of two complementary components, a physics predictor and a belief regulator. While the former predicts the future states of the object(s) manipulated by the robot, the latter constantly corrects the robot's knowledge regarding the objects and their relations. Through our experiments, in complex environments consisting of different shaped objects and articulation types we have shown that by using this framework, the robot can reliably predict the consequences of its actions in object trajectory level and exploit its own interaction experience to correct its belief about the state of the environment. Furthermore, we have shown that we can use this framework in tool manipulation and planning.

ÖZET

İTME MANİPULASYONLU GÖREVLERDE AKSİYON-EFEKT KURAMININ NESNEL VE İLİŞKİSEL EKSENDE MUHAKEMESİ

Kompleks robotik sistemlerde, eylem-tepki tahmini çözmesi zor problemlerdendir, özellikle değişken sayıda obje ve bu objelerin arasında çeşitli etkileşimler varsa. Bu tarz sistemleri modellemek için, veriyi birden fazla obje ve etkileşimi tarif edebilecek şekilde kodlamak gerekir. Bu tezde önce bizim çeşitli şekillerdeki objeler üzerinde yaptığımız araştırmayı sunduk. Sonrasında, bahsettiğimizde tarz sistemleri modellemek için, grafik sinir ağı tabanlı bir yapı ve genel amaçlı bir fizik simülatörü olan Kendi İnancını Düzenleyen Çifte Yayılım Ağları(Belief Regulated Dual Propagation Networks)'nı öneriyoruz. Bu yapı birbirini destekleyen iki bileşenden oluşuyor, fizik tahminci ve inanç düzenleyici. Fizik tahminci robotun işlediği objelerin bir sonraki anını tahmin ederken, inanç düzenleyici robotun işlenen objelerle ilgili olan bilgisini güncelliyor. Deneylerimizde, farklı tip objeler ve eklemler bulunduran kompleks sistemlerde, robotun yaptığı eylemlerin sonuçlarını obje yörüngesi düzeyinde tahmin edebildiğini ve bu eylemlerden kazandığı tecrübelerden yararlanarak, ortam ile ilgili bildiklerini güncelleyebildiğini gösterdik. Aynı zamanda, bu sistemle nasıl alet kullanıp planlama yapılabileceğini gösterdik.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

a	Aggregation function
a_r^k	Relation Attributes of relation k
a_i^o	Physical Attributes of object i
$c_{k,t}^r$	Relation encoding of k relation at time t
$c_{i,t}^o$	Object encoding of i object at time t
C_t	Memory cell at time t
d_k	Displacement vector of relation k
$e_{i-j,t}$	Relation effect between i^{th} and j^{th} object at time t
E	Edge
f_O	Object function
f_O^l	Object propagation function
f_O^{enc}	Object encoder function
f_R	Relation function
f_R^l	Relation propagation function
f_R^{enc}	Relation encoder function
G	Graph
h_t	Message unit at time t
N^o	Cardinality of objects
N^j	Cardinality of relations
V	Node
o_i	Features of i^{th} object
O	Object
r_j	Features of j^{th} relation
R	Relation
$p_{i,t}$	Object effect of i^{th} object at time t
u	Global attributes
x_i	State of $i^{th} object$

$\mathcal{N}_i$	Relations of object i

LIST OF ACRONYMS/ABBREVIATIONS

2D	Two Dimensional
3D	Three Dimensional
ANN	Artifical Neural Network
ARTag	Artifical Reality Tag
BRDPN	Belief Regulated Dual Propagation Networks
CNN	Convolutional Neural Network
GNN	Graph Neural Network
HDD	Hard Drive Disk
Im-Cnn	Image CNN
K-NN	K-Nearest Neighbor
LSTM	Long-Short Term Memory
MLP	Multi Layer Perceptron
MSE	Mean Square Error
PCB	Printed Circuit Board
PropNets	Propagation Networks
RNN	Recurrent Neural Network
RMSE	Root Mean Square Error
ROS	Robot Operating System
Sc	Shape Context
Sc-CNN	Shape Context CNN
SP	Support Points

1. INTRODUCTION

Predicting effects in complex physical systems is a challenging problem, especially when the number of objects varies and there are rich and wide variety of interactions between objects. Besides, when the objects are linked with various physical connections like contact or joint, this further affects the object dynamics and makes the problem even more challenging. To be able to model such complex physical systems accurately, it should be possible to represent the system to appropriately handle the encoding of multiple objects and their interactions with each other. In addition, information regarding the objects and relation between them may not be known beforehand or be erroneous. In these cases, it should be possible to correct or to provide information regarding these objects and their relations.

In the context of recycling of electromechanical devices (An example object can be seen in Figure 1.1), to disassemble a hard-drive, a robot needs to take into account the relations between parts of the hard-drive in imagining the effects of its actions. The research done in this thesis is part of the larger research agenda studied in the

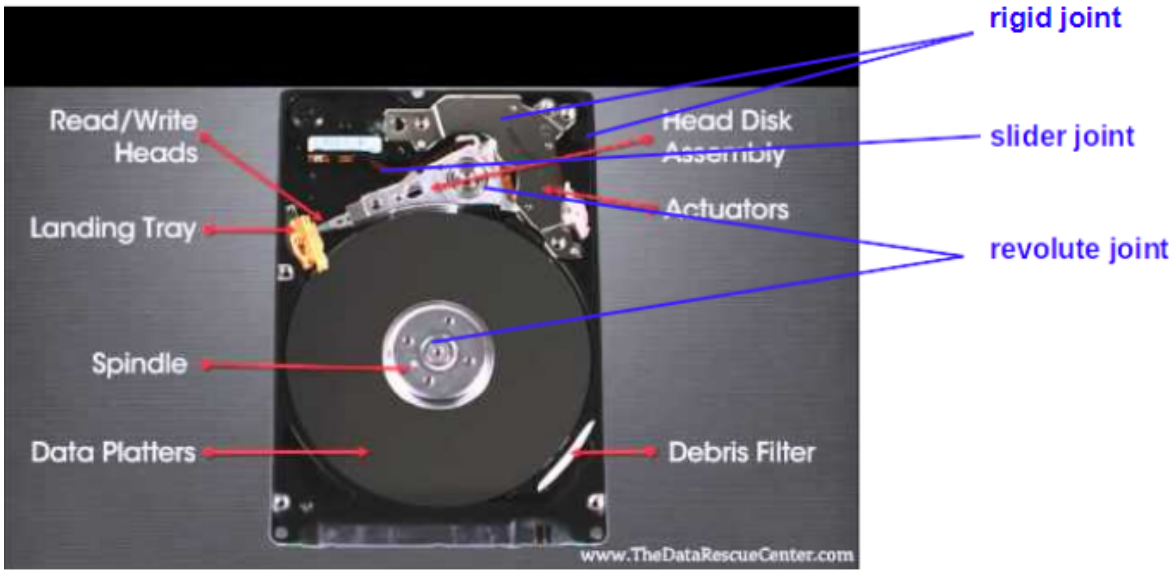


Figure 1.1. Robot needs to understand dynamics of part and joints between them to find proper disassembling routines for disassembling multi-part objects.

Imagine project [1] where the robot needs to imagine the effects of its action using machine learning methods.

A Graph is a structure consisting of nodes and edges that connect nodes to each other. By representing each object with a node and object-object interaction with edges, complex physical systems can be represented with graph structures. Graph neural networks [2] can exploit the graph structure of multi-objects systems by implementing and using object- and relation-centric representations. Graph neural networks employ shared object-wise and relation-wise computations for estimating the object dynamics which make this a very powerful approach.

Recently, a great amount of effort has put on the prediction of the dynamics via graph networks (e.g. [3–9]). These works can deal with varying number of objects and learn rich interaction dynamics among these objects. Some of these works have focused on developing learnable physics engines [3–7], while others were aimed at unsupervised learning [8, 9] . However, applying them to model robot-object interactions is not very straightforward as the active involvement of the robot was not taken into account in the previous studies.

In this thesis, our aim is to provide predictive models that allow us to reason about action-effects in push manipulation tasks. For this reason, we investigated the use of machine learning methods for the following two prediction problems: Action-effect prediction of single objects with various shapes [10, 11], and prediction of effects of robot actions on multi-object tasks [12]. In order to tackle the former problem, we took advantage of the use case of disassembling of electromechanical devices where different parts of the devices can have different shapes and geometries. To model the trajectory of these parts, we proposed using LSTM based effect prediction networks which allow to incorporate temporal information of the trajectory for effect prediction. For the latter, we investigate the use of graph neural network for action effect reasoning. Overall, our contribution to effect prediction can be summarized as followed:

- We have shown that LSTM models can be used for predicting low-level object motion trajectories that are generated by robots action. This was first demonstrated on single object [10] like printed circuit board(PCB), and next on objects with a wide variety of shapes and geometries [11].
- We have shown that we can use *Belief Regulated Dual Propagation Network (BRDPN)* [12], a network we proposed, for predicting the low-level trajectories of groups of articulated objects given push actions of a manipulator robot and for estimating joint relations between these objects based on interaction history of the objects and the robot. However, BRDPN was limited to cylindrical objects and required training of two independent networks.
- We allowed our networks to share weights, which allowed us to decrease the number of the learned parameter by about thirty per cent. We extended our network to be able to handle cuboid objects which in turn allowed us to predict action effects on environments with complex-shaped objects that can be built from cuboids and rigid joints. Besides, we further optimized the system with scheduled sampling [13] and decreased the errors for long-horizon predictions.
- Since our framework is very generic, we have shown that BRDPN can also be used for mass prediction and for tool manipulation. In the former, as robot interacts with objects, our system managed to do predict the weights of the objects. In the latter, by transferring the same representation and network trained on the aforementioned environments with complex shaped objects, we represented completely novel tools that were not experienced during training with graph neural networks. By representing them as objects attached with fixed joint relations, we used them in tool manipulation and planning to achieve goals.
- Finally, we have shown a simple example of how PropNets [5] can be used in 3D effect prediction. In this example, a manipulator robot levers-up or pushes a printed circuit board from the hard drive. This required prediction of both orientation and position of the printed circuit board.

2. BACKGROUND

In this chapter, background on machine learning networks used in this thesis will be explained.

2.1. Graph Neural Networks

Graph neural networks are networks that take graph structured input and give desired output. Both for input and output, this graph can have up to 3 attributes which are edge (E), node (V), and global (u) attributes.

Graph neural networks extend other neural network methods for processing the data represented in the graph domain. In this graph, each node and edge will have a set of features that identifies that node and edge. It can also support global attributes that define the graph. For each node and edge, it should be possible to learn about which information from other elements of the graph is relevant for estimating its output. This is done by message passing. This message passing is done as follows:

(i) First for each edge, a message is estimated.
(ii) For each node, messages coming from the edges belonging to the corresponding nodes will be aggregated. This aggregation operation can be done in many ways. Some of the popular aggregation methods are mean, sum and max operations.
(iii) Messages coming from nodes can be further aggregated to be used for estimating global level outputs, and messages.
(iv) Each edge message can be reused in corresponding edges, each node message can be reused in the corresponding node and its edges, and global message can be reused in all node and edges.
(v) By iterating these steps, information can be passed through all graph. Mostly, the number of iterations will be fixed and will depend on the task.

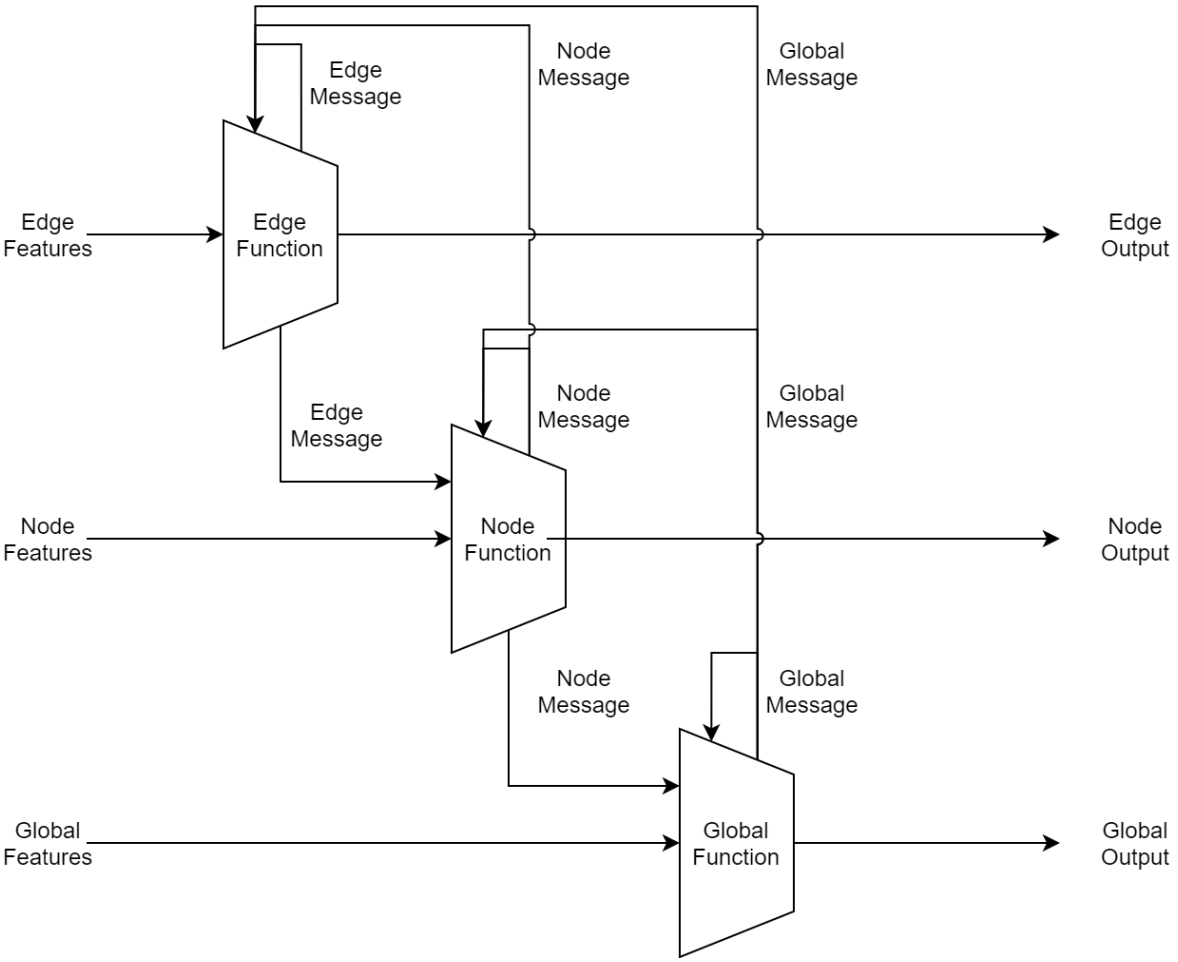


Figure 2.1. Graph neural network model.

After the message passing operation is done, each node and edge will know more about its neighbouring nodes and edges and can do better output estimations.

2.2. Interaction Networks

Interaction network [3] is a general-purpose learnable physics engine. It's a type of graph neural network which takes objects and relations between them as input. It can reason about objects dynamics and can predict the next state of objects. This is done by reasoning on relations and objects of the complex physical system.

For defining dynamics of a single object, one can use object function f_O to predict the next state of that object. However when multiple objects are present, this function will not be sufficient, information of interaction between objects need to be taken into

consideration.

This interaction information can be processed with a relation function f_R. For each object, its interaction with other objects will be processed with this relation function and will be sent to its object function. However, in case there are multiple interactions, this will not be sufficient and this relation information somehow needs to be combined. This can be done with aggregation function a. Aggregation function can be mean, sum, or any differentiable operation that can merge a variant number of feature vectors. Ideally, it should be permutation invariant.

For an object, process can be described as follows:

(i) Object's interactions with other objects will be processed with f_r.
(ii) Outputs of f_r will be merged with a.
(iii) Output of a will be used with object information to find next state of object.

With sum aggregation, the overall formula of the interaction network can be expressed with the following equations. $o_{i,t}$ corresponds to features of i^{th} object at time t, $r_{j,t}$ corresponds to features of j^{th} relation at time t. N^o and N^r corresponds to the cardinality of objects and their relations. $\mathcal{N}_i$ corresponds to relations of object i with other objects. A relation can be both expressed with its index or corresponding object indexes.

$$e_{i-j,t+1} = f_R(o_{i,t}, o_{j,t}, r_{k,t}) \quad k = 1 \dots N^r \tag{2.1}$$

$$p_{i,t+1} = f_O(o_{i,t}, \sum_{k \in \mathcal{N}_i} e_{k,t}) \tag{2.2}$$

Effect prediction capacity of this network was shown on of n-body problems, rigid-body collision, and non-rigid dynamics [3].

2.3. Propagation Networks

Since message passing capacity of interaction network is limited to single step (from relations to objects), it has certain shortcomings. One of them is when an object's movement has chain effects on other objects, interaction network fails to learn such dynamics.

Propagation network [5] is a general-purpose learnable physics engine that can handle instantaneous propagation of effects between objects. This is by increasing message passing capacity of the interaction network. The interaction network has message passing between relations to objects only, this can be called relation message. Propagation network defines an additional message on object level which can be passed to relations, this will be called object message. By doing this message passing between relation to object and object to relation few times, information of each object can be further passed to its non-immediate neighbours. This is called propagation.

Since propagation operation with several layers is costly, the relation network and object network is divided into two parts, encoder and propagator. Overall Propagation network can be expressed as follows:

Encoding step is as followed.$c^r_{k,t}$ and $c^o_{i,t}$ are latent encodings of objects and relations.

$$c^r_{k,t} = f^{enc}_R\left(r_{k,t}\right), \quad k = 1 \dots N^r \tag{2.3}$$

$$c^o_{i,t} = f^{enc}_O\left(o_{i,t}\right), \quad i = 1 \dots N^o \tag{2.4}$$

Propagation Step (iterated n times):

$$e^l_{k,t} = f^l_R\left(c^r_{k,t}, p^{l-1}_{i,t}, p^{l-1}_{j,t}\right), \quad k = 1 \dots N^r \tag{2.5}$$

$$p^l_{i,t} = f^l_O\left(c^o_{i,t}, p^{l-1}_{i,t}, \sum_{k \in \mathcal{N}_i} e^{l-1}_{k,t}\right), \quad i = 1 \dots N^o \tag{2.6}$$

In their experiments, it was shown that propagation network outperforms interaction network on tasks such as newton's cradle and rope manipulation where modelling instantaneous propagation of effects is necessary [5].

2.4. Long Short-Term Memory

Recurrent neural networks(RNN) are neural networks that can handle temporal sequences by allowing previous outputs to be used as inputs. Long-Short Term Memory [14] (LSTM) network LSTM network is a gated RNN architecture. They were invented since standard RNN can not handle long term dependencies. LSTM handles long term dependencies by using a gate mechanism. Similar to RNN, LSTM has hidden states (h_t) but besides, it uses internal memory and specific processing units, which are previously mentioned gates. Figure 2.2 provides the structure of LSTM. Purpose of each gate is as follows.

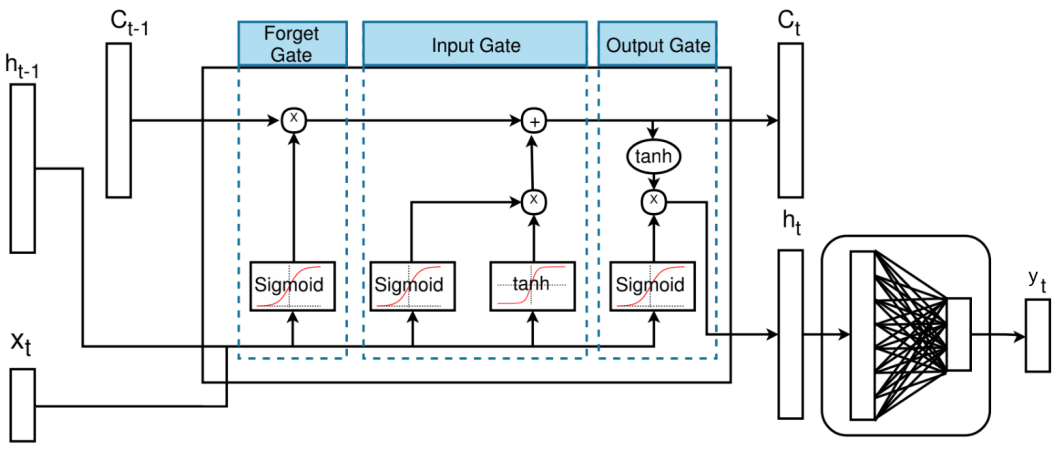


Figure 2.2. LSTM model.

(i) At Forget Gate, a sigmoid layer calculates a vector with values between 0 and 1 using the sigmoid activation function. This vector is multiplied with the memory cell (C_{t-1}) and the memory cell is updated. With this process, some information kept in the memory cell is forgotten.

(ii) At Input Gate, a sigmoid layer calculates a vector that decides which of the values in the candidate vector are added to the memory cell. At the same time, a *tanh* layer (with the hyperbolic tangent activation function) calculates a candidate vector to add to the memory cell. The output of the sigmoid layer(Input Gate) and tanh layer are multiplied and added to the memory cell (C_t).

(iii) At Output Gate, similarly, a sigmoid layer calculates a vector that decides which information to output. At the same time, hyperbolic tangent function is applied to values in the memory cell (C_{t-1}) previously updated by the input gate. It is multiplied with the output of output gate. This provides the output message(h_t).

At preparation, it is benefited from following tutorial [15].

3. EXPERIMENT PLATFORM

In this chapter, the physical components, frameworks, and software used in this research will be explained.

3.1. Physical Components

3.1.1. UR10 Robot

In the robotic experiments, both in simulation and real-world settings, we used the UR10 [16]. UR10 is an industrial robot arm designed by Universal Robots. It excels with its high precision and reliability. It has 10 kg payload with a reach radius of up to 1300 mm. It has 6 degrees of freedom (DOF) arm. In our experiments, the UR10 robot's motion was limited to planar motion and 6 degrees of freedom was sufficient.

3.1.2. Robot Gripper

In the robotic experiments, in real-world settings, we use Robotiq *3-Finger Adaptive Robot Gripper* [17]. In our experiments, the gripper is used just for holding push tool which is a mallet in our experiments. UR10 robot holding a mallet can be seen in Figure 3.1.

3.2. Frameworks

3.2.1. Robot Operating System

The Robot Operating System (ROS) is a middle-ware software framework which eases the process of writing applications for controlling robots [18]. It allows its users to effortlessly create software packages. Design of ROS allows software packages using ROS to easily communicate with each other. It allows the end-user to easily get frameworks necessary to run their robot and its, and then develop software to make

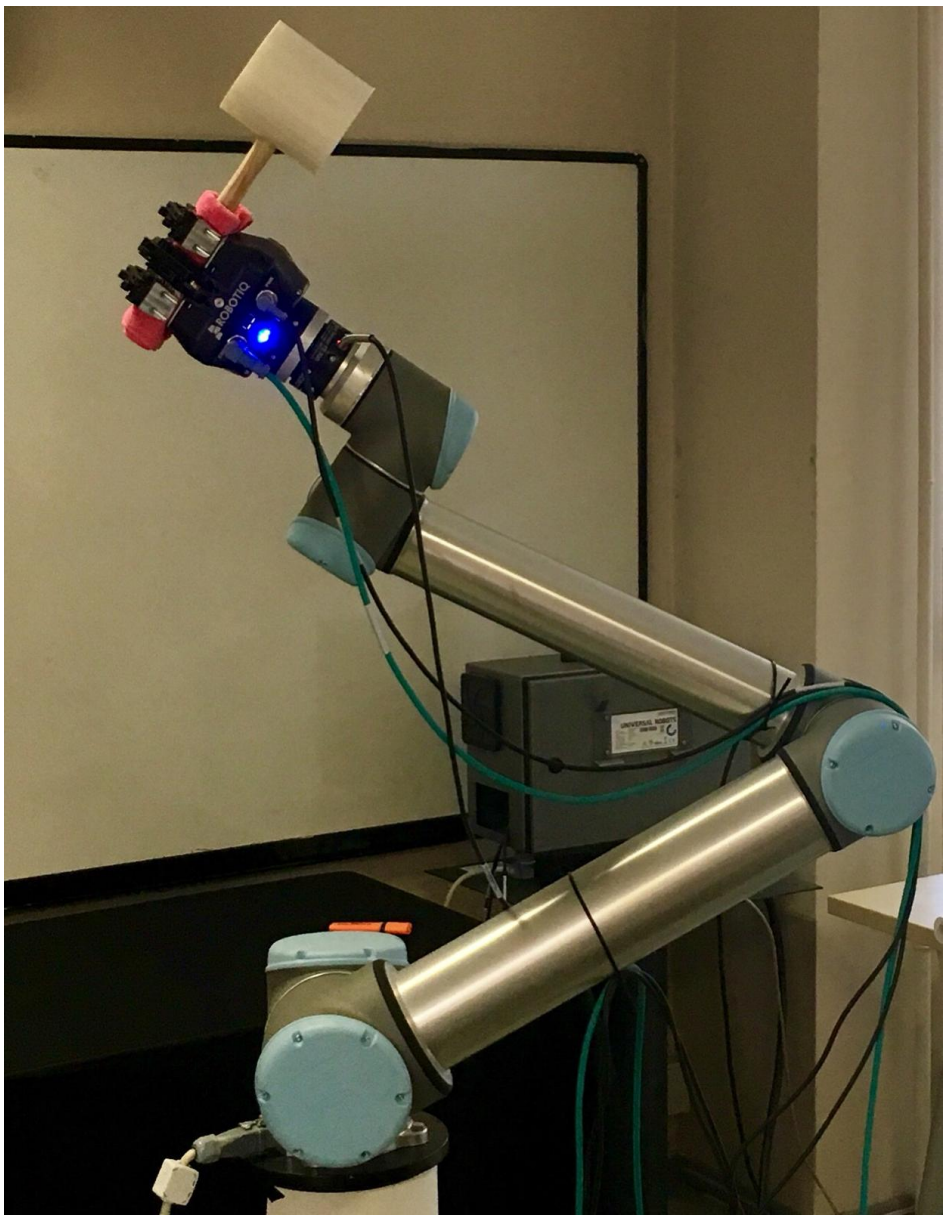


Figure 3.1. UR10 robot holding a mallet.

the robot do different tasks. There are many specifications on how frameworks for robot controllers, cameras, sensors should run, and this makes it easier for ROS programmers to move to different robots, cameras etc. Ros can be used with python and C++.

3.2.2. CoppeliaSim

CoppeliaSim is a robot simulator. It is an experimentation platform that allows to control, edit, modify different scenes and robots [19]. It is the successor of V-Rep [20] simulator and shares most of its features. It enables the users to write scripts which

further increase customizability of prepared experiments. It has an API for external control and has a ROS backend that end users can prototype their robotic applications in simulation. It allows usage of different physic engines like ODE, Bullet, etc. API is provided for many languages including C++ and Python.

3.2.3. Pyrep

Pyrep is a toolkit built on top of CoppeliaSim [21]. It provides a remote API similar to CoppeliaSim. However, it gets ahead with following improvements PyRep brings. It provides an object-oriented backend that eases defining simulation tasks and it speeds up running time of simulation by making the remote API faster, and by allowing multi-process usage of CoppeliaSim.

One of the main purposes of this toolkit was to accelerate deep learning research. In their Github page, they also demonstrate how their toolkit can be used for reinforcement learning.

3.2.4. Keras

Keras is a high-level deep learning framework [22]. It provides a consistent and simple neural network API written in Python. It allows rapid experiments and its one of the most used deep learning framework in Kaggle competitions since it is easy to test ideas with it.

3.2.5. PyTorch

Pytorch is another deep learning framework [23]. It provides a rich ecosystem with many research done on vision and natural language processing. It is easy to define custom layers and modules which makes it very good for research.

3.3. External Library and Packages

In addition, libraries like Scikit-learn [24], Matplotlib [25], Numpy [26] is used in experiments for data processing, acquiring results, and plotting. In addition collision [27] library is used in generation of scene for data collection to detect collisions.

4. RELATED WORK

Our task is closely related to learning the dynamics of objects and scenes, the effect of different relations between objects and understanding intrinsic information about the objects and their relations. Research done in these areas will be described. However, our main focus is doing action-effect prediction on the robotic domain and specifically using graph neural networks. For this reason, learning of dynamic specifically with graph neural networks will be mentioned. Then, more emphasis will be given to the research done in the robotics domain in these fields.

4.1. Learning of Dynamics - Modelling Physics

There have been a lot of interest on intuitive physics area [28]. For instance, Battaglia *et al.* [29] proposed a Bayesian model called Intuitive Physics Engine and showed that the physics of stacked cuboids can be modelled with this model. Similarly, Hamrick *et al.* [30] showed that humans can reason about object masses from their interactions, and modelled it with Bayesian models. Smith *et al.* [31] have modelled expectation violation in intuitive physics. They discuss how humans surprise when their physical expections mismatch with reality and they modelled this with deep learning methods. Deisenroth *et al.* [32] suggested a probabilistic dynamic model that depends on Gaussian Processes and that is capable of predicting the next state of a robot given the current state and its actions. Recently, some researchers extended these works by using deep learning methods to model physics. Lerer *et al.* [33] trained a deep network to predict the stability of the block towers given their raw images obtained from a simulator. Groth *et al.* [34] extended this idea by allowing stacking of objects with different geometry. They show that their network is capable of predicting stability of given tower even in this harder setup. The stacking of block tower task has continued to be an important intuitive physics environment [35].

A specific topic of interest within modelling physics with deep learning is motion prediction from images, which has gained increasing attention over the last few years. Mottaghi *et al.* [36] trained a CNN for motion prediction on static images by casting this problem as a classification problem. Mottaghi *et al.* [37] employed CNNs to predict movements of objects in a static image when some external forces are applied to them. Fragkiadaki *et al.* [38] suggested a deep architecture in which the outputs of a CNN are used as inputs to Long Short Term Memory (LSTM) cells [14] to predict movements of balls in simulated environments.

4.2. Graph Neural Networks For Learning Physics

As deep structured models, GNNs allows learning useful representations of entities and relations among them, providing a reasoning tool for solving structured learning problems. Hence, it has found particularly wide use in physics prediction. Interaction network by Battaglia *et al.* [3] and Neural Physics Engine by Chang *et al.* [4] are the earliest examples to general-purpose physic engines that depend on GNNs. These models do object-centric and relation-centric reasoning to predict movements of objects in a scene. Though they were successful in modelling dynamics of several systems such as n-body simulation and billiard balls, their models had certain shortcomings, especially when an object's movement has chain effects on other objects (e.g. a pushed object pushes another object (s) it is contacting with) or when the objects in motion have complex shapes. These shortcomings can be partly handled by including a message passing structure within GNNs as done in the recent works such as [5–7]. Most of these networks used simple neural networks for encoding object and relation information. Kipf *et al.* [39] showed that variational auto encoders can be used in encoding object and relation information. His network managed to encode object information directly from their trajectories in an unsupervised way.

Another approach was acquiring object information directly from images. Ye *et al.* [40] used image and detected location of objects to predict latent representation of next time step. This latent representation was then decoded do create image of next time step. Watters *et al.* [8] and van Steenkiste *et al.* [9] proposed hybrid network

models which encode object information directly from images via CNNs and which predict the next states of the objects with the use of GNNs. Lately, these networks have been extended to handle even more complex environments. Sanchez-Gonzales *et al.* [41] show that graph neural network can be used for learning particle based simulations consisting of more than 1000 particles and have acquired state of the art results.

4.3. Effect Prediction in Robotics

In [42], they trained forward and inverse models for learning how to poke an object to move it into a target position. This network uses latent vectors of CNN to train predictive models. The forward model tries to predict the latent representation of the final image using the current image, and the inverse model took latent representations of both final and initial images to find the parameters of the poke action.

Several studies have examined the action-effect prediction from videos in robotics. Finn *et al.* [43] proposed a convolutional recurrent neural network [44] to predict the future image frames using only the current image frame and actions of the robot. Byravan *et al.* [45] presented an encoder-decoder like architecture to predict SE(3) motions of rigid bodies in depth data. However, the output images get blurry over time or their predictions tend to drift away from the actual data due to the accumulated errors, making it not straightforward to use for long-term predictions in robotics.

Lately, after graph neural networks become much more popular, it has been started to be used robotics as well. Janner *et al.* [46] used graph neural networks to jointly learning of object factorizations from perception and physic prediction. This is done by using acquired object factorizations to predict the next successive object factorizations. Ye *et al.* [47] learned object-centric forward models to do planning and control. Their model takes object bounding boxes as input and use CNNs to encode object and learn to predict future states from this object encodings. They have shown that they can use this forward model to do long horizon plannings. Sanchez-Gonzales *et al.* [48] have used graph networks as learnable physics engines in robotic setups.

They shown that they can use this graph networks for inference and control.

4.4. Parameter Estimation

Wu *et al.* [49] proposed a deep approach for finding the parameters of a simulation engine which predicts the future positions of the objects that slide on various tilted surfaces. Zheng et al [50] used perception prediction networks, a type of graph neural network, for learning latent object properties from interaction experience to simulate system dynamics. Li et al [51] used recurrent neural networks to predict the center of mass from object mask and interaction experience. Xu *et al.* [52] used a deep learning architecture for learning object properties. In their settings, robot slides an object from an inclined surface and cause it to collide with another. By using a sequence of dynamic interactions, they showed that their model can learn to predict object representations.

Different from these approaches, there is also heavy interest on decomposing objects into primitive parts. In our system, we assumed that primitive parts of the objects are given, however this is another problem that is needed to be solved. In their work, Deng *et al.* [53] have shown that from input images, objects can be decomposed into convex hulls. In addition, they have shown that these convex hulls can be used for physics simulation. Similarly, Pashevich *et al.* [54] have shown that their framework can propose different part sets where objects can be divided into, and then reconstruct the divided object in real world with a robot using available primitives on the robots workspace.

5. SINGLE OBJECT ACTION-EFFECT PREDICTION

In this chapter, our initial paper [10] about this subject, *LSTM-based effect trajectory prediction of HDD* will be explained. This work is submitted to and presented in Turkish Robotic Conference 2018 (ToRK 2018).

In this work, we showed that LSTM models can predict 3D trajectories of different objects. This is demonstrated on a cylindrical object and printed circuit board. This work is limited to lever-up action and few simple-shaped objects, however, learned trajectory is a complex one and in future work, we extended this model with shape descriptors to transfer it to complex-shaped objects.

5.1. Motivation

People do their actions with purposes. For an action, if one can predict the effect of this action on the whole motion trajectory level, he can imagine possible future action plans and selects his action accordingly.

To predict the trajectory of an object, we need to consider the object, its interactions with surroundings and type of action that is applied to the object. In this paper, we limited the task to a single object and single parameterized action. This is demonstrated with lever-up action and two objects: a cylindrical object, and a printed circuit board (PCB). This is done with two LSTM models. The first model predicts the trajectory of the object given lever-up location, the other predicts the lever-up location given the trajectory. Using these two models together, we can start lever-upping object from a position, and from the feedback of motion of the object, we can fix original lever-up location and correct the predicted trajectory. To evaluate the performance of our models, V-Rep physic simulator [55] is used. We compared our method with K-NN and showed that we can do more generalizable predictions with LSTM.

5.2. Method

For single fixed object, and fixed action, trajectory prediction can be defined with where the action is applied, and the initial position of the object.

$$X = f(P_{\text{lever-up-point}}, x_o) \tag{5.1}$$

- $P_{\text{leverup-point}}$: The point of contact. This point has been sampled from the object's edges since the aim is to lift the body and make the opening motion.
- X: All positions in the object's trajectory.
- x_o: Initial Position of the object.

The aim here is to calculate the entire trajectory from the lever-up position and from the first position of the object. This can be done in an auto-regressive manner. Meaning, prediction at time t can be used for prediction at time $t + 1$.

$$x_{t+1} = f(P_{\text{lever-up-point}}, x_t) \tag{5.2}$$

Additionally, when the action starts, the latent state of the object starts to change as well. Since this can not be reliably provided to the system, the model should additionally learn to identify this state. This can be done by adding a unit h_t that learns to record this latent state at timestep t.

$$(x_{t+1}, h_{t+1}) = f(P_{\text{leverup-point}}, x_t, h_t) \tag{5.3}$$

- h_t: Internal parameters.

For this reason, we have used the long-short term memory (LSTM) model, which is a type of recurrent neural network (RNN). A visual illustration of this model can be seen in Figure 5.1

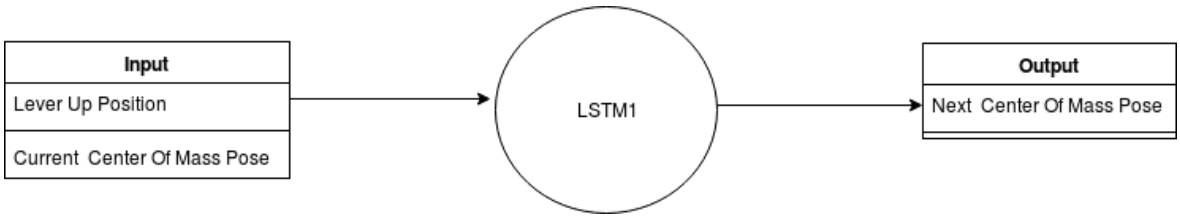


Figure 5.1. Visual illustration of Equation 5.3.

This model predicts trajectory from given lever-up location, in a way, it assumes that lever-up location is correct. But because of errors in visual perception, lever-up location may be gotten wrong, or it may not know it at all. For this, we can define another model that takes the trajectory of observed object motion and predict where the action is actually applied. This can be modelled using another LSTM model. A visual illustration of this model can be seen in Figure 5.2

$$P_{\text{leverup-point}} = f(x_{0:t}) \tag{5.4}$$

From the trajectory observed and predicted lever-up point, we can predict the remaining trajectory. This integrated model can be seen on Figure 5.3.

5.3. Experiment Setup

Experiments are done in V-Rep simulator. Robot lever-up action is done with normal forces from lever-up location.

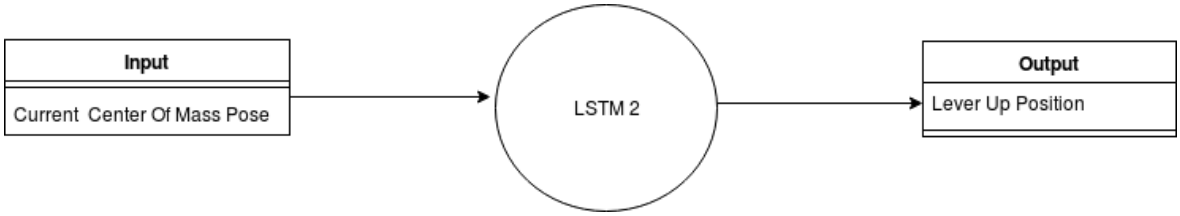


Figure 5.2. Visual Illustration of Equation 5.4. This network predicts lever-up location from observed trajectory.

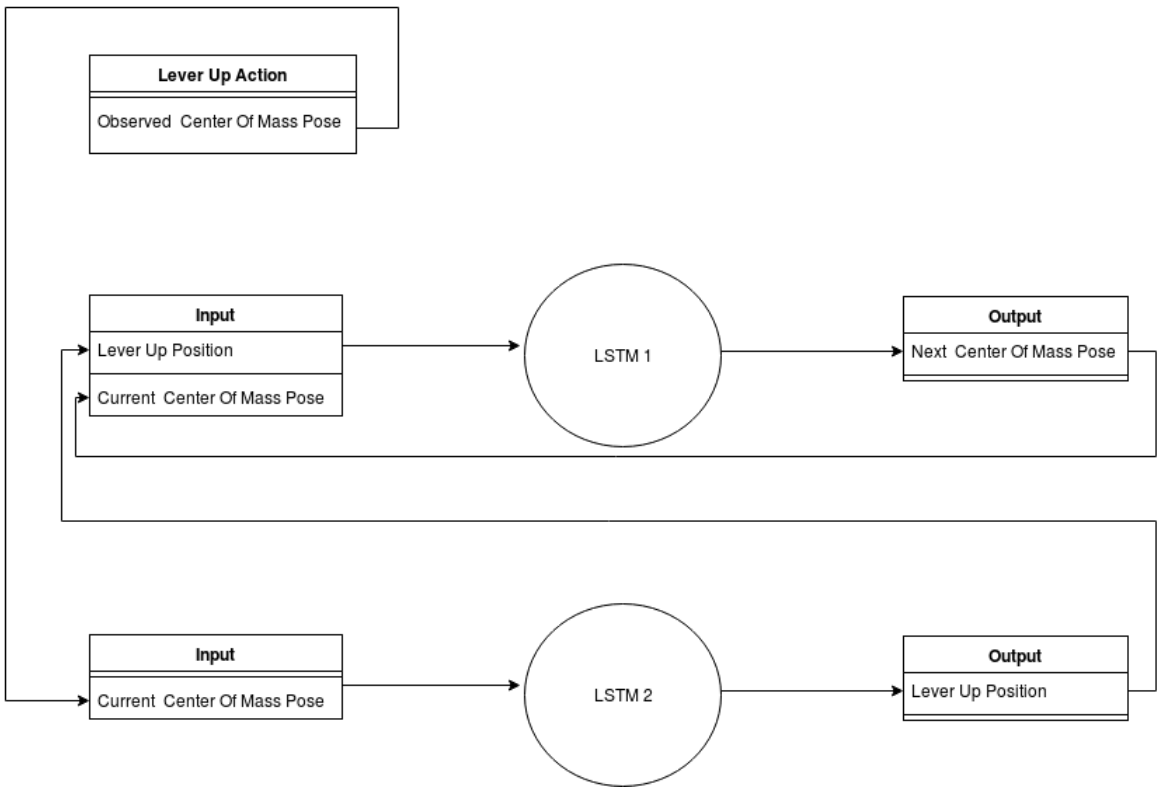


Figure 5.3. Integrated Model. Here, at timestep t, lever-up location is predicted by using first t timesteps. Remaining trajectory is predicted using this predicted lever-up location, trajectory observed until timestep t.

In the first experiment setup, in V-Rep, an empty environment with cylinder disk is used. This disk had 10 cm radius and 2 cm height. Time difference of simulation is 10 milliseconds and ODE is used as a physics engine.

Amount of force to apply is set experimentally calculated with trial and error. In every experiment, objects positions and orientations are saved with 40 ms time differences. With the lever-up location, position and orientation of objects, the dataset is created. This dataset contains lever-up action done from 100 different lever-up location. These lever-up locations are selected uniformly from edges of the disc.

For learning, Keras API which contains implemented LSTM design is used. For background engine, Tensorflow library is used.

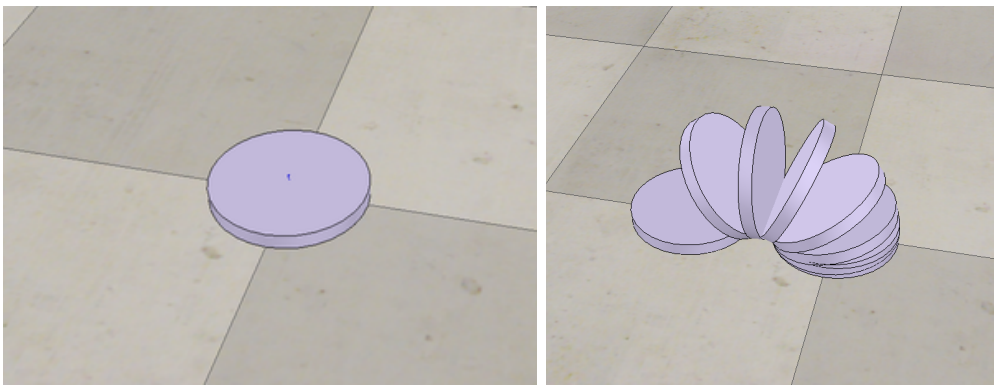


Figure 5.4. V-Rep Environment. Disk with 10 cm radius and 2 cm height, 10 ms time difference and ODE physics engine.

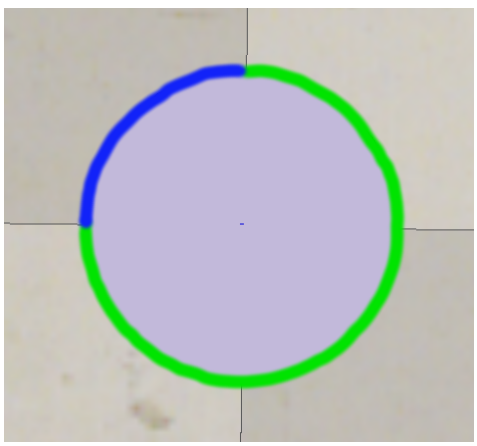


Figure 5.5. Training and Test set. Green is training set, Blue is test set.

To test how much LSTM based learning generalizes prediction, the training and test set are divided logically: they are selected completely from different parts. They are divided as %75-%25. It is shown in Figure 5.5.

5.4. Experiment Results

5.4.1. Trajectory Prediction Model

Figure 5.7 shows the result of the LSTM model (and the K-NN model) responsible for predicting the trajectory that will be generated when it is removed from the edge it did not see during learning. First two rows show the results of the K-NN model whereas the remaining rows show the results of LSTM network. Intuitively, for K-NN

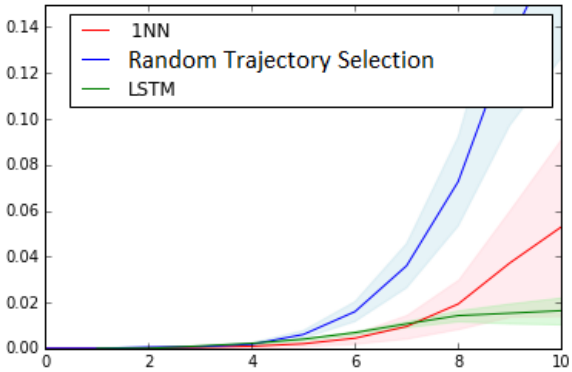


Figure 5.6. The error comparison between LSTM, KNN and random trajectory selection. Root mean square error (RMSE) is used as the measure of error.

it is difficult to predict the trajectory correctly because training data do not include test points, but the LSTM network can make better predictions because it learns a more general model.

The overall error rate from our model is given in Figure 5.6. In addition to 1-NN, random trajectory selection model which chooses a completely random trajectory sampled from our dataset is used. As it can be seen, as the motion continues, the error rate of 1-NN increases in high ratio compared to the error rate of LSTM during the prediction of the trajectory. In Figure 5.6, the lines show the error averages, and the shadows show the error variance.

5.4.2. Lever-up Location Prediction Model

The output of the lever-up location prediction model is generally similar to the first column of Figure 5.8. As can be seen here, even though these positions are at similar angles with respect to disc, they are not in correct positions. Since we know that all lever-up actions are done from edges of the disc, we can add a constraint that takes lever-up locations to edge of the disc. In a way, this found locations can be moved to edges of the disc. For this, an angle can be calculated from x, y values. Then using this angle, lever-up location can be recalculated. These new recalculated lever-up locations can be seen on the second column of Figure 5.8. Also, how post-processing

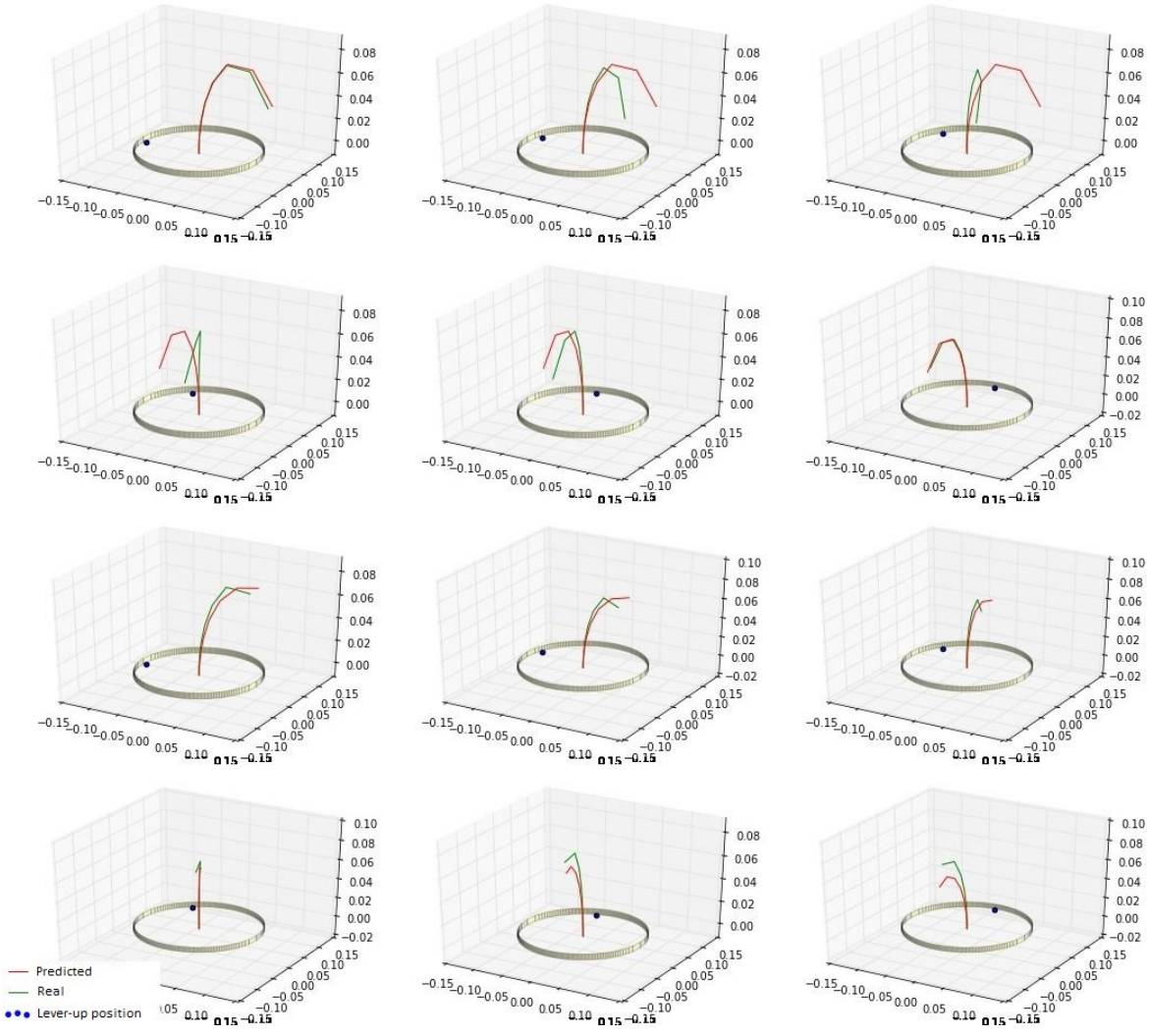


Figure 5.7. Results coming from the trajectory prediction model. Every column corresponds to same lever-up position.First two rows are 1-NN, other two rows corresponds to trajectories coming from LSTM

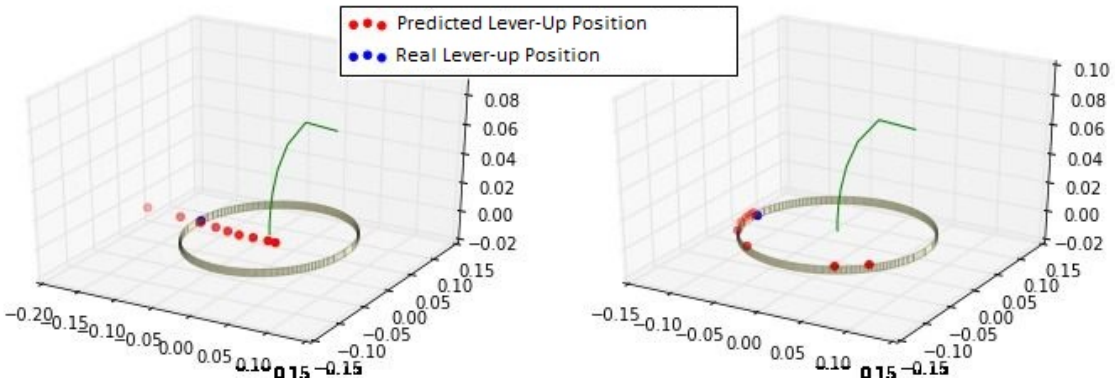


Figure 5.8. In left, lever-up locations found from model 2 and in right, points found after post processing

on data improved accuracy of the lever-up prediction model can be seen on Figure 5.9.

5.4.3. Integrated Model

The motion of the trajectory is initiated assuming that the lever-up motion has begun at the lifting points in the test set. As we observe the motion over this trajectory, new trajectories are predicted from the lever-up points and the poses in trajectory observed so far. As the number of poses increases, the error rate decreases. Figure 5.10 shows how much error model acquire after fixing the lever-up point during the motion on different timestep.

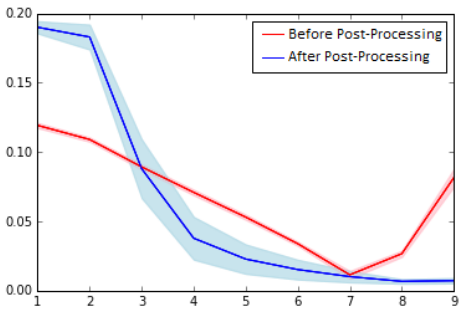


Figure 5.9. Mean squared error of lever-up locations before post processing and after post processing.

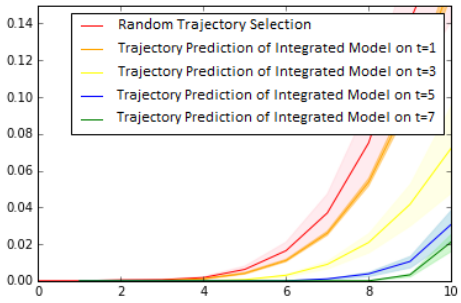


Figure 5.10. The average square root of the errors that the models give when they are processed together. Here you can see the amount of error in the trajectories that the model predicts from the lever-up point after experiencing the first 1, 3, 5, 7 poses.

5.4.4. Complex Shape: Printed Circuit Board

In this experiment, the same trajectory prediction task is applied to a printed circuit board (PCB). In this model, compared to cylinder disc, the shape is more complex, so more complex trajectories can be observed.

Since there are no features related to the shape of the object, the model can't learn it very well. Especially, in lever-up positions where there are discontinuities in the shape of the trajectory, the model's prediction is not very accurate.

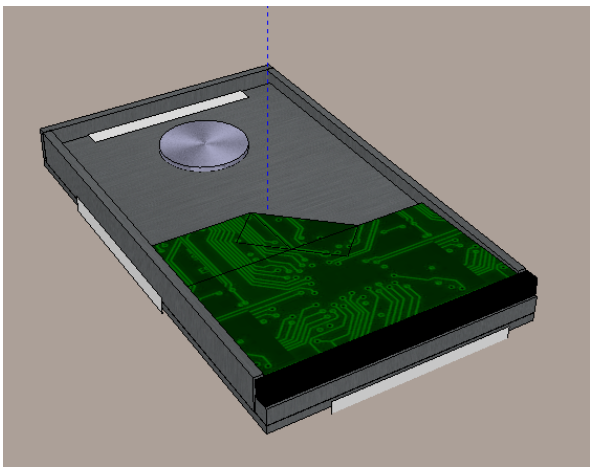


Figure 5.11. HDD model in V-REP.

5.5. Conclusion

In this work, we showed that LSTM models can predict motion trajectories. Even though this work is limited to lever-up action, we believe that model can be transferred to other parameterized effect prediction tasks with similar complexity.

As future work, we are aiming to include properties about the shape of the object and movement parameters about the robot's action to expand our model. Additionally, interaction with the objects is another potential research area.

6. ACTION-EFFECT PREDICTION ON COMPLEX SHAPED OBJECTS

In this chapter, our extension paper for the previous model [11], *Deep Effect Trajectory Prediction in Robot Manipulation*, will be explained. This work is published in Robotics and Autonomous Systems in 2019 (RAS 2019).

6.1. Motivation

For intelligent agents, predicting the effects of its action is an important requirement. Being able to do action-effect prediction would allow the system to possibly imagine and simulate any action on any object. Especially if the agent can do this on the low-level motion trajectory level. This can be used to train inverse-models, in action-selection and monitoring of execution performance.

In this work, our aim is investigating the effect of changing object shapes on low-level object motion trajectories and how we can model it using deep neural networks. This research is done in accord with IMAGINE project whose aim is to make robot reason about its action with its effect. This is implemented in the context of recycling and one of the most commonly used action which is lever-up action is selected as sample action in this research.

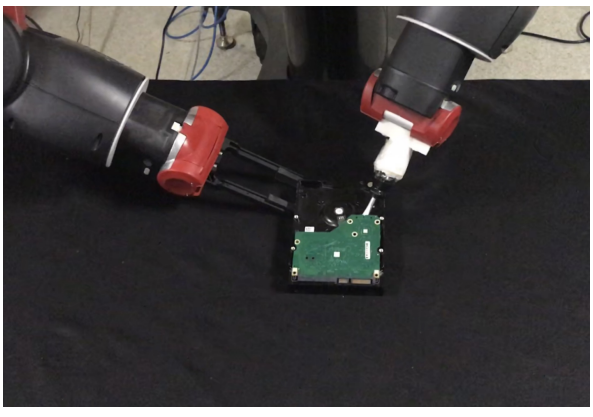


Figure 6.1. Baxter robot performing lever-up action in order to extract the PCB module of a hard drive.

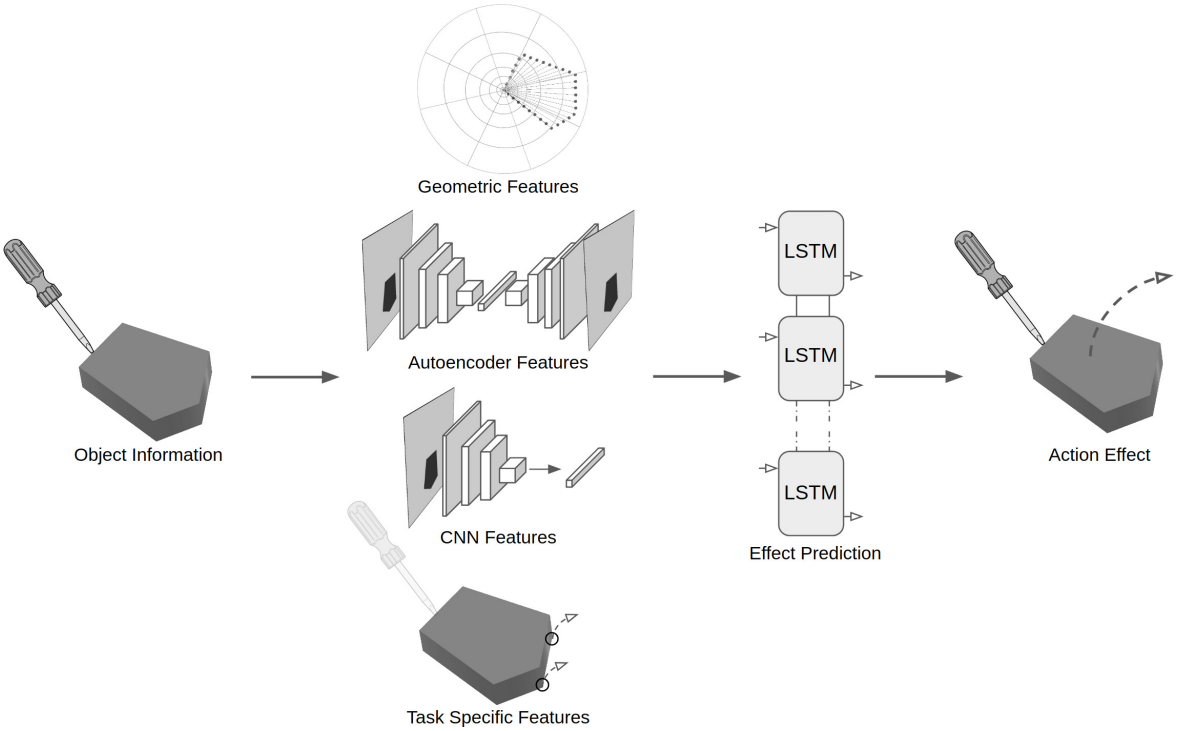


Figure 6.2. The robot applies an action on the given object. Our model can extract features from the top-down image of this object, and then can use this feature with an LSTM effect predictor to predict low-level trajectory of the object.

The effect of changing object shapes on low-level object motion trajectories is done in two steps: feature extraction and effect prediction. In the feature extraction step, various object descriptors such as shape context, autoencoder features and more task-specific features like support points are compared. In the next step, recurrent neural networks are used to predict the complete motion trajectory of the object given the object features. Given top-down 2D images of objects, our model can predict the low-level trajectory of applied action.

6.2. Method

Our model predicts the effect of a given action on the object in Figure 6.2. This prediction is done by first extracting relevant features from the image of the object and using them with a recurrent neural network (RNN) that outputs the position trajectory.

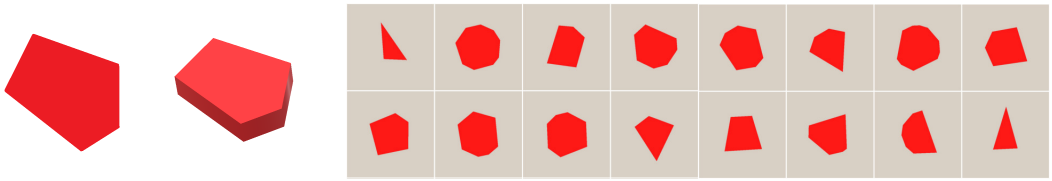


Figure 6.3. A number of randomly generated shapes on V-REP.

Depending on feature extraction, we propose following models:

- Generic unsupervised features based models: Features that can be independently estimated from the given task are used. For this models, shape context, which is a compact histogram representation of 2D geometry of the object, and auto-encoder, which is a dimensionality reduction technique to compresses high dimensional image data to low dimensional feature vectors, features are used.
- Supervised features based models: Feature extractor is conjointly trained with action effect predictor. A convolutional neural network that is trained together with an LSTM effect prediction model is used.
- Task-specific features based models: A specific set of features that are considered to well-represent the particular task and action are manually designed and used. For this, using support points of the lever-up trajectory which well represent this task is used.

6.3. Experimental Setup

We used V-REP [20] physics-based simulator with Bullet engine for the collection of data. To verify our system, we generated a set of random shaped objects. Applying lever-up action from each of its edges, we created a dataset. An object is generated by sampling 3 to 10 corner points from a circle with radius 10, and then by connecting the successive corner points (Figure 6.3). 160 objects are generated in total for interaction.

For each generated object, from each of its edges, lever-up action is applied to two positions sampled from that edge, and its effect is recorded. UR10 robot arm with a screwdriver attached to is hand is used for applying the lever-up action. This lever-up

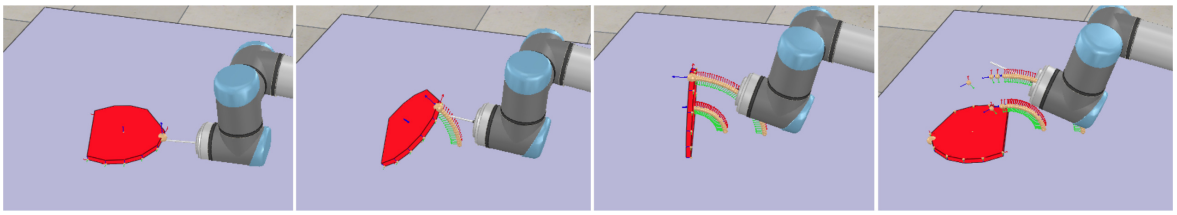


Figure 6.4. An example of lever-up experiment with a sample shape.

action is an open-loop action with the same relative trajectory with respect to where lever-up actions are applied as shown in Figure 6.4. 1800 trajectories are recorded. For each trajectory, the shape of the object, the top-down image is taken by a simulated Kinect camera, and the trajectory of the center-of-mass of the object is saved. For further supervision on task-specific features, support points are also recorded.

We shuffled the dataset into training (1400), validation (100) and test sets (300). For a better comparison between proposed feature extractors, the dataset division and experiments are repeated with 10 different seeds. The models were trained to predict with a sequence length of 15 . Number of hidden units of the LSTM was empirically selected as 256. The model is validated on full trajectory prediction. Poses obtained from intermediate steps are also used for estimation of loss which boosted the overall performance of the system. Validation error on the whole trajectory is used for selection of the best model. Keras framework [22] is used as a deep learning platform. ADAM optimizer [56] with default hyper-parameters and mean squared error loss is used. All models are trained for 100 epoch.

6.4. Results

In this section, we will first show the results of the support point prediction model and then compared errors of all proposed feature extractors.

6.4.1. Support Point Prediction

Support points are task-specific features that are selected for the prediction of lever-up action. On test time, since these support points are still needed to be predicted,

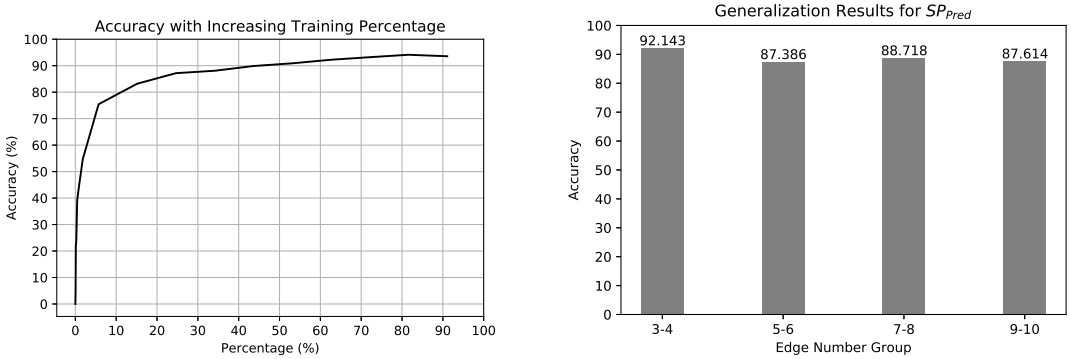


Figure 6.5. In left, change in support point prediction accuracy with increasing training size. In right, Generalization performance of the model when shapes with selected number of edges are used for testing while remaining are used for training.

it is important to have an accurate model so that our effect prediction error will not be high. Our results can be seen in Figure 6.5. Results show that our model can reach 93% accuracy on the prediction of support points. Even with using 50% of data, the model can still reach 90% accuracy which is a satisfactory result. Additionally, it can be seen that when the model is not trained on shapes with a certain number of edges, the model still manages to generalize to predict novel shapes which it has not seen before.

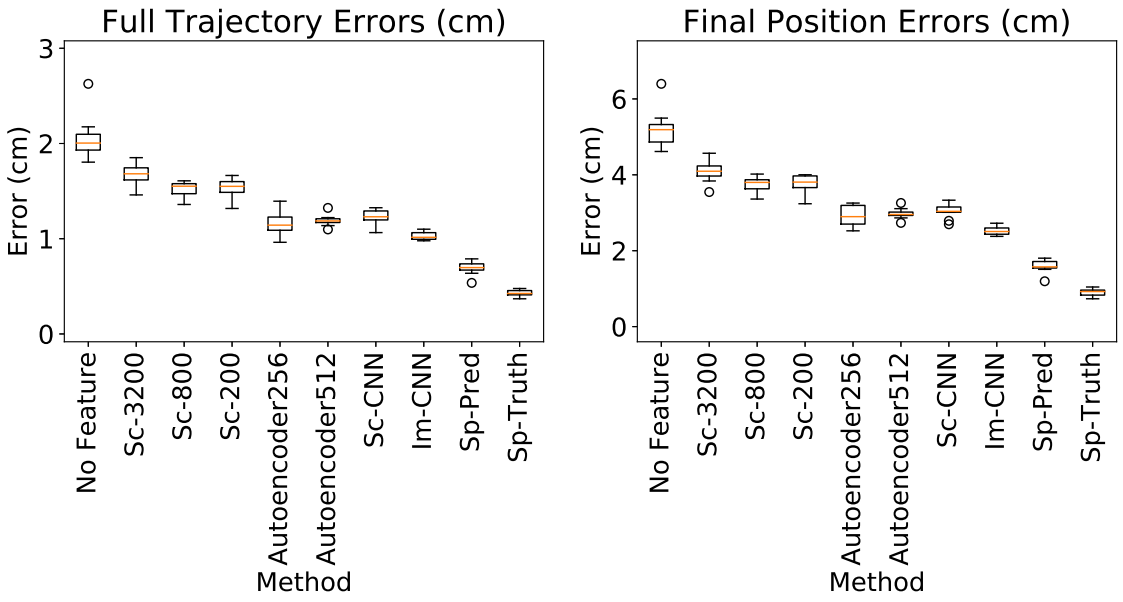


Figure 6.6. Prediction performance of different models are shown.

6.4.2. Object Movement Trajectory Prediction

Object movement trajectory prediction model is evaluated in this section. Figure 6.6 provides the errors comparison between different features. The first word stands for feature name, and the number stands for its hyperparameter. Sc corresponds to shape context, Im-CNN corresponds to image CNN, and Sc-CNN corresponds to shape context CNN (Shape context have a similar data structure to images.). SP stands for Support points. Error is RMSE between predicted and ground truth trajectories. The bars in the plot correspond to the mean error obtained by repeating the training and test 10 times with randomly shuffled data, and the lines correspond to the standard error.

Additionally, in Figure 6.7, how training size affected model performance can be seen. Overall, each model increases its accuracy as more training data is used at training. Support point-based models acquire good prediction results even with a low amount of data.

As can be seen, overall, supervised features perform better when compared to unsupervised features. Still, the performance of autoencoder based features are not bad compared to supervised methods, and they can be a good alternative if its hard to train CNN together with LSTM. Task-specific features perform well as expected.

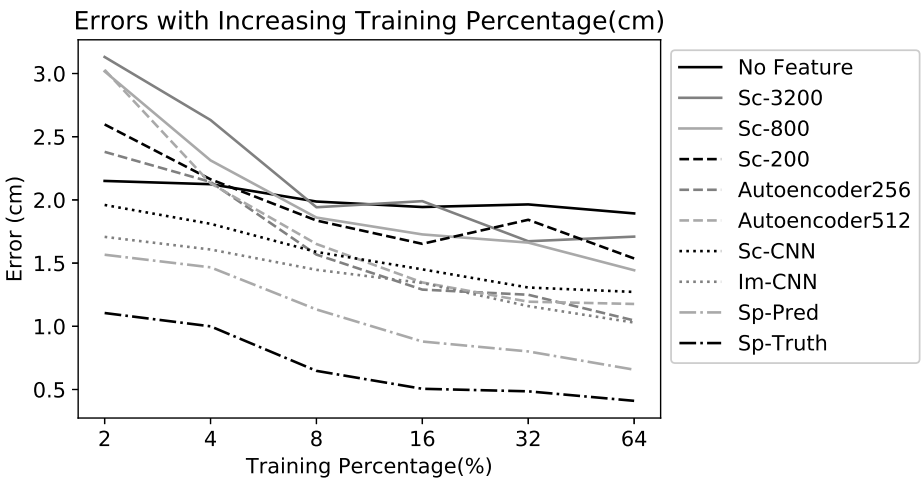


Figure 6.7. Performance of model with different percentage of training set usage.

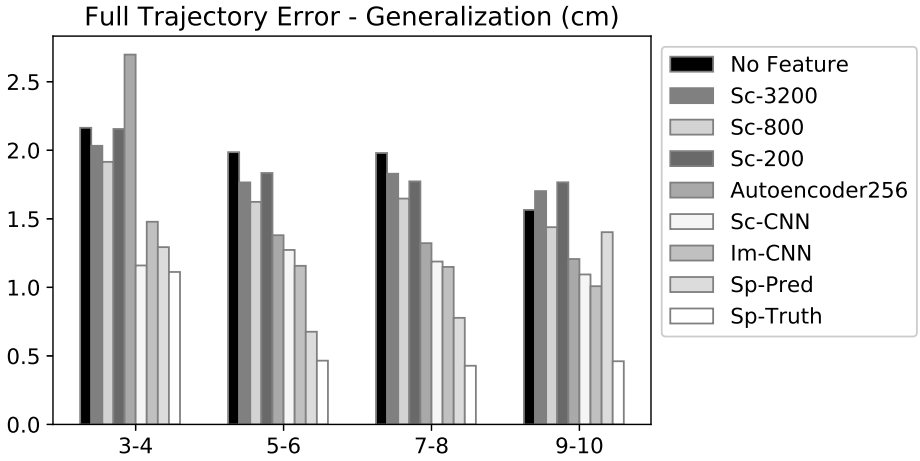


Figure 6.8. Generalization performance of trajectory prediction models.

However, they come at the cost of task-specific knowledge and manual engineering.

6.4.3. Model Generalization Analysis

In Figure 6.8 models generalization performance can be seen. The model is tested on shapes with a given number of edges while trained on the rest. This comparison is done so to see how different feature extractors perform when they need to do prediction on novel shapes that have not been seen before. We found that support point based methods and autoencoders do not perform as good as other methods. It can be seen that CNN based methods exhibit stable generalization performance.

6.4.4. Verification in The Real Robot

In this section, the real-world evaluation of our model is provided. The model on simulation is transferred to real-world and predicted trajectories are compared with actual trajectories. The robot is equipped with a screwdriver tool. Performed experiment is shown in Figure 6.9. A video of the experiment can be found on [57]. Pentagon and heptagon shaped objects are 3D printed and used in the experiment. These objects have similar shapes to objects from the simulation. The experiment is performed on each edge of these shapes. For tracking, an Intel real sense camera is used. Additionally, ARtags are placed on objects for real-time tracking. Im-CNN-based LSTM model

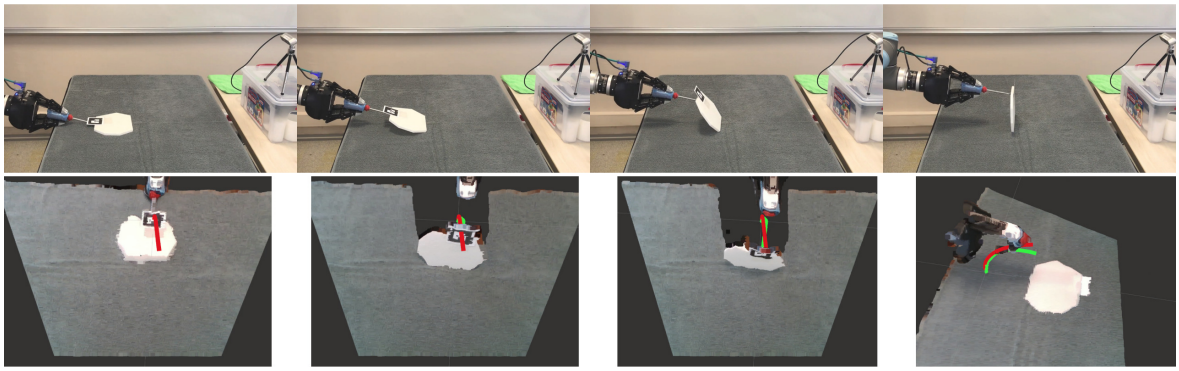


Figure 6.9. Real robot execution and its Rviz visualization. Steps of execution is shown left to right. In second row, trajectory prediction can be seen. Red line corresponds to predicted trajectory, while green line corresponds to actual.

is used in this experiment. Since the sampling rate may differ from the simulated system, dynamic time warping method is used to remove the timing discrepancy between predicted and observed trajectories. Euclidian distance between these trajectories are estimated and shown in Tables 6.1 and 6.2. These results show that our model can predict the effects of real robot actions successfully.

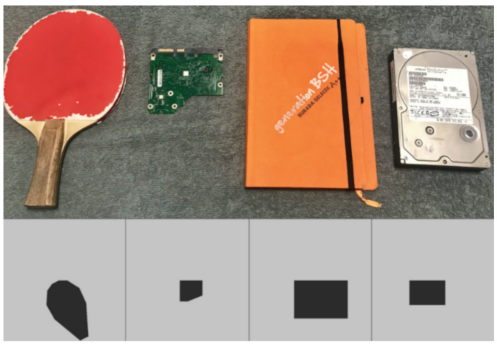
To be not limited to 3D printed objects, we enriched our experiments with a range of real-world objects namely, book, a PCB, an HDD and a table tennis racket. (Figure 6.10a). The masks of the objects were constructed by finding their corresponding convex hulls, and these masks were provided to the Im-CNN predictor as in the previous subsection. As shown in Figure 6.10b, the errors between the predicted and the real trajectories were around 0.5 cm for all objects, therefore indicating that our model can be successfully used to predict the effects of the actions on various daily objects.

Table 6.1. Error comparison of pentagon object. Error of trajectory generated from simulation and predicted trajectory are provided.

	Edge 1	Edge 2	Edge 3	Edge 4	Edge 5
Real vs Simulation	0.485	0.520	0.379	0.385	0.474
Real vs Predicted	0.981	0.446	0.480	0.611	0.401

Table 6.2. Error comparison of heptagon object. Error of trajectory generated from simulation and predicted trajectory are provided.

	Edge 1	Edge 2	Edge 3	Edge 4	Edge 5	Edge 6	Edge 7
Real vs Simulation	0.447	0.910	0.368	0.333	0.720	0.864	0.338
Real vs Predicted	0.601	0.633	0.460	0.316	0.531	0.499	0.471



(a) Real Objects

HDD	PCB	Book	Racket
0.123	0.344	0.415	0.479

(b) Prediction errors for real objects

Figure 6.10. (a) Some object from daily life and their masks. (b) Average errors (cm) between the predicted and observed trajectories

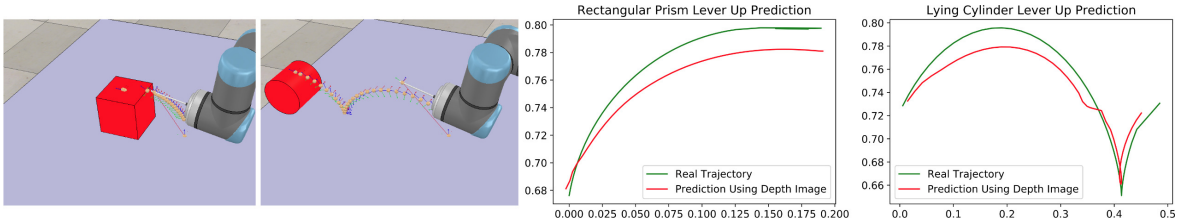


Figure 6.11. Objects used on the experiments. These objects have similar top-down image but different depth images. Their predicted trajectories are provided. It can be seen that our model can differentiate objects based on their depth images.

6.4.5. Non-flat Objects and Depth-Enriched Image Masks

In this section, we extended our model to depth data and experimented on non-flat objects. Instead of top-down masks, top-down depth images of objects are used. We created dataset containing lying various sized cylindrical and rectangular objects that can be distinguished only by their depth information. Originally, Im-CNN can not distinguish between cylindrical and rectangular objects, however by using depth information, Im-CNN can learn to predict effects of non-flat objects. Original Im-CNN has 7.23 cm error, whereas Im-CNN that uses depth images acquire error of 2.77 cm.

6.5. Conclusion

In this work, we investigated the effect of changing object shapes on low-level object motion trajectories and how we can model it with various feature extractors. Advantages and disadvantages of each model are experimentally shown and discussed. The trajectory prediction model that uses manually engineered features has shown the best performance out of other used feature descriptors. However, since for each new task and action, further engineering is needed, it is not feasible. For general purpose usage, we have found that Im-CNN based trajectory prediction model is very powerful and robust. In future work, we plan to study action-effect prediction with complex parametrized actions.

7. REASONING OF ACTION-EFFECTS ON ARTICULATED MULTI-OBJECT TASKS

In this chapter, main topic of the thesis, *Belief Regulated Dual Propagation Nets for Learning Action Effects on Groups of Articulated Objects* [12], will be explained. This work is accepted and to be presented in International Conference on Robotics and Automation in 2020 June (ICRA 2020).

7.1. Motivation

In complex robotic systems, predicting the effects is a challenging problem when the number of objects varies, especially when there is a presence of rich and various interactions among these objects. When several objects are linked with various physical connections like contact or joint, this would suggest some connections between them which results in propagation of forces and motion between objects. To be able to model such systems, the representation of data should be able to sufficiently encode multiple objects and interactions between them. In addition, the model should allow to correct or to provide information regarding the robot's belief about objects and relations between them.

Recently, a great amount of effort has put on the prediction of the dynamics via graph networks (e.g. [3–9]). These works can deal with varying number of objects and learn rich interaction dynamics among these objects. Some of these works have focused on unsupervised learning, while others were aimed at developing learnable physics engines [3–9]. However, applying them to model robot-object interactions is not very straightforward as the active involvement of the robot was not taken into account and, uncertainty in perception was not explicitly addressed.

In this work, we propose *Belief Regulated Dual Propagation Network (BRDPN)* which takes the actions of the robot and their effects into account in predicting the next

states. The network continuously regulates its belief about the environment based on its interaction history to correct its future predictions. For belief regulation, extending the recently proposed propagation networks (PropNets) [5] that handle instantaneous effect propagation, we propose a temporal propagation network that takes history of the motion of each object to predict unknown object or relation properties. Our system is verified on a table-top push setup that has cylindrical objects and joint relations between them. Our setup includes varying number of objects that might be connected with *rigid*, *revolute* or *prismatic joints*. The model definitions of these types of relations is not provided to the robot. Notice that in our settings the relations between objects cannot be perceived by the robot. From its interaction experience in the simulator, it learns to predict relations between objects given observed object motions, and exploits this information to predict future object trajectories. Furthermore, it was transferred to real world and verified in experiments that included around 100 interactions with 2 to 5 objects. Our system was shown to outperform the original PropNets, both in simulation and real-world, when the relations between objects were not reliably provided to the system. Our source code and experimental data are available [58].

Contribution of this work to the state of the art is two-fold: First, we introduced a deep neural network based method for learning how to exploit the interaction experience of the robot to extract values of otherwise unknown state variables in partially observable environments. Second, we implemented a learning based effect prediction robotic framework that can handle multiple interacting objects that might have different types of connections, and we verified this framework both in simulated and real robot experiments.

7.2. The Proposed Model

In this section, we introduce the *Belief Regulated Dual Propagation Networks (BRDPN)* and explain how it extends the propagation network framework for articulated multi-part multi-object settings to allow the regulation of the beliefs about environment state variables. Belief regulation corresponds to regulating the robot's belief about the environment through extracting or updating the values of state vari-

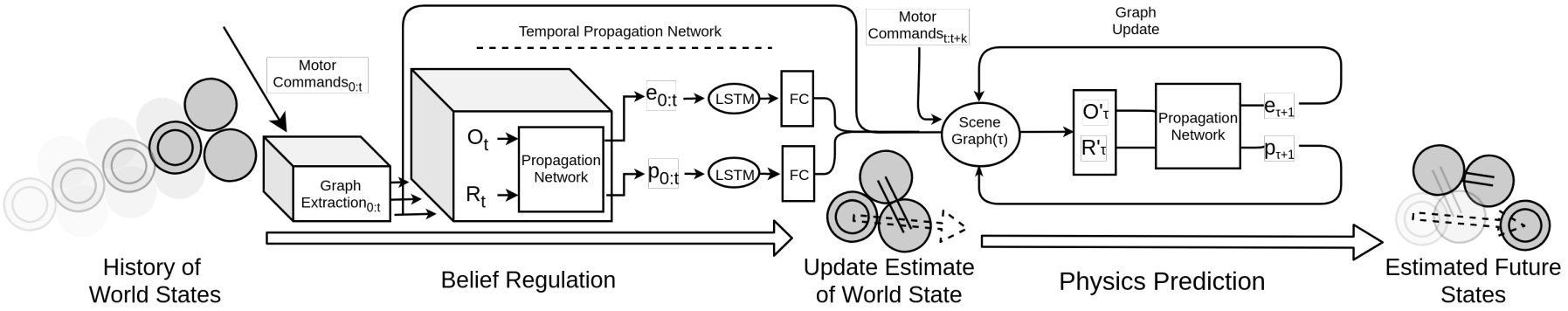


Figure 7.1. *Belief Regulated Dual Propagation Networks.* The belief regulation module is used to update the estimate of the state variables. The physics prediction module predicts the sequence of future states expected to be observed.

ables. Figure 7.1 shows a graphical illustration of our framework, which is composed of two main components: a *physics predictor* and a *belief regulator*. The physics predictor is based on propagation network and responsible for predicting future states of the manipulated objects. The belief regulation module is a propagation network with recurrent connections, which we call temporal propagation network. Belief regulation module is responsible of extracting/updating the knowledge of the robot about the environment through its observations of own-interaction experience. We explain the technical details of these models in the following parts.

7.2.1. Preliminaries

Assume that the robot is operating in a complex environment involving a set of multi-part objects O, we express the scene with a graph structure $G = \langle O, R \rangle$ where the nodes $O = \{o_i\}_{i=1:N^o}$ represent the set of objects (of cardinality N^o) and the edges $R = \{r_k\}_{k=1:N^r}$ represent the set of relations between them (of cardinality N^r). More formally, each node $o_i = \langle x_i, a_i^o \rangle$ stores object related information, where $x_i = \langle q_i, \dot{q}_i \rangle$ is the state of object i, consisting of its position q_i and velocity $\dot{q}_i$, and a_i^o denotes physical attributes such as its radius. Each edge $r_k = \langle d_k, s_k, a_k^r \rangle$ encodes the relation between objects i and j with $d_k = q_i - q_j$ representing the displacement vector, $s_k = \dot{q}_i - \dot{q}_j$ denoting the velocity difference between them, and a_k^r representing attributes of relation k such as the type of the joints connecting objects i and j.

7.2.2. Physics Prediction

Propagation networks encode the states of the objects and the relations between them separately. This encoding is carried out by two encoders, one for the relations denoted by f_R^{enc} and one for the objects denoted by f_O^{enc}, defined as follows:

$$c_{k,t}^r = f_R^{enc}\left(r_{k,t}\right), \quad k = 1 \dots N^r \tag{7.1}$$

$$c_{i,t}^o = f_O^{enc}\left(o_{i,t}\right), \quad i = 1 \dots N^o \tag{7.2}$$

where $o_{i,t}$ and $r_{k,t}$ represent the object i and the relation k at time t, respectively. $c^r_{k,t}$ and $c^o_{i,t}$ are latent encodings of objects and relations.

To predict the next state of the system, these encoders are used in the subsequent propagation steps within two different propagator functions, f^l_R for relations and f^l_O for objects, at the propagation step l, as follows:

$$e^l_{k,t} = f^l_R\left(c^r_{k,t}, p^{l-1}_{i,t}, p^{l-1}_{j,t}\right), \quad k = 1 \dots N^r \tag{7.3}$$

$$p^l_{i,t} = f^l_O\left(c^o_{i,t}, p^{l-1}_{i,t}, \sum_{k \in \mathcal{N}_i} e^{l-1}_{k,t}\right), \quad i = 1 \dots N^o \tag{7.4}$$

where $\mathcal{N}_i$ denotes the set of relations where object i is being a part of, and $e^l_{k,t}$ and $p^l_{i,t}$ represent the propagating effects from relation k and object i at propagation step l at time t, respectively. Here, the number of propagation steps can be decided depending on the complexity of the task. Through using the predicted states as inputs, it can chain the predictions and estimate the state of the objects at $t + T$. See [5] for more detailed description of this network.

7.2.3. Belief Regulation

The success of physics prediction step highly depends on how accurate the environment is encoded in the graph structure. Here we refer to the term belief as the estimated world state and given previous states and motor commands, the role of the belief regulation module is to constantly update this crucial part. In propagation network [5], the authors provide a method for estimating unknown parameters using gradient updates. In theory, it is possible to adapt this framework, however, since the relation types need to be represented as one-hot vectors, employing a continuous representation may lead to unreliable predictions. As the main theoretical contribution of this work, we propose a *temporal propagation network* architecture that augments a propagation network with a recurrent neural network (RNN) unit to regulate beliefs regarding object and relation information over time. More formally, it takes a sequence of a set of state variables during the action execution as input and employing a sec-

ondary special-purpose propagation network, it encodes these structured observations, which are then fed into an RNN cell to update the current world state, as follows:

$$r'_{k,t} = f_O^{blf}\left(e_{k,t}^L, r'_{k,t-1}\right), \quad k = 1 \dots N^r \tag{7.5}$$

$$o'_{i,t} = f_R^{blf}\left(p_{i,t}^L, o'_{i,t-1}\right), \quad i = 1 \dots N^o \tag{7.6}$$

where L denotes the propagation step, and f_O^{blf} and f_R^{blf} denote the RNN-based encoder functions for objects and relations, respectively. Feeding these functions with the sequence of encoding vectors $r'_{k,t-1}$ and $o'_{i,t-1}$ allows the temporal propagation network to consider the overall history of object and relation states from the previous timesteps. Hence, it continuously updates its belief regarding objects and relations states ($o_{i,t}$ and $r_{k,t}$), and eventually minimize the difference between the effect predicted by our physics prediction module and reality.

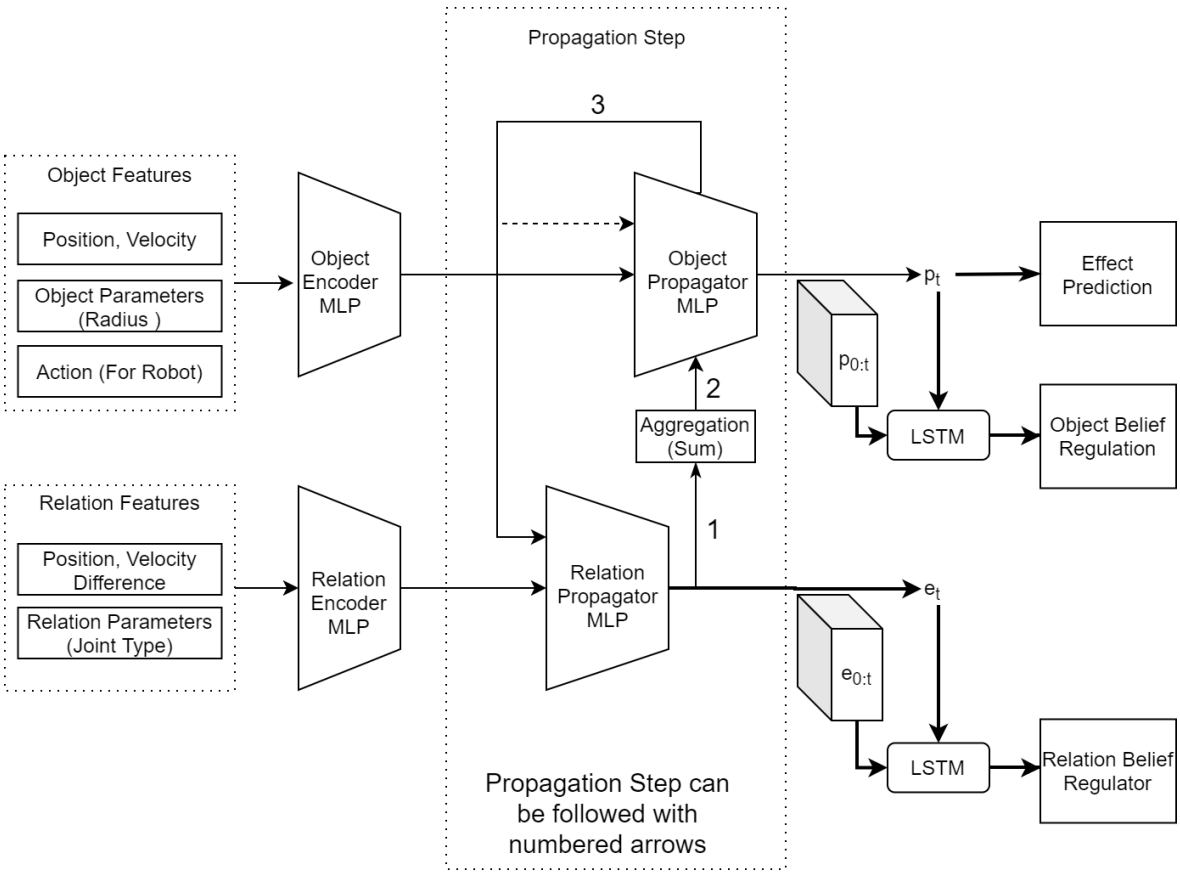


Figure 7.2. In-depth visualisation of Belief Regulated Dual Propagation Networks.

(a) Simulation (b) Real-World (c) Sparse (d) Dense

Figure 7.3. Sample setups (a-b), and initial configurations (c-d)

7.3. Experimental Setup

We evaluate our model in simulation and on a real robot through a set of experiments. In the following, we explain the details of the experimental setups designed to assess the generalization performance to the changing number of objects and timesteps, transferability of our model to different object-relation distributions and the real-world setting.

7.3.1. Robotic Setup

Our simulation and real world experiments included a 6 degrees of freedom UR10 arm and several cylindrical objects placed on a table as shown in Figure 7.3.a-b. The table-top settings were composed of objects of varying numbers and sizes. The objects might move independent of each other (no-joint) or connected through one of three different types of joints, namely *fixed*, *revolute* and *prismatic*. We prepared two sets of environments: fixed environments, which could contain only fixed joints, and mixed environments that may contain all three joint types. The robot learned to predict the effects of its actions by self-exploration and observation in the V-REP physics-based simulator with Bullet engine [20]. The simulated robot exercised its push action on a set of objects by moving a cylindrical object that was attached to its end-effector. After training, the performance of the prediction model was tested both in the simulated and real-world settings.

In our simulation experiments, we considered two different configurations for scene generation: a *sparse* configuration (Figure 7.3.c) where objects were initially scattered randomly in the scene, and a *dense* one (Figure 7.3.d) where the objects were initially grouped. Sparse configuration was specifically designed to maximize the contact time between the end-effector and the objects and to allow a rich set of interactions. The robot chooses 8 different linear motions of 30cm, maximizing contact time with the most diverse set of objects. While in the sparse configuration the objects were randomly scattered in the scene, in the dense configuration the objects were grouped in a grid structure. Sparse configuration was used for training, and dense configuration for testing the generalization performance of the model on novel environments, i.e. on instances drawn from a completely different distribution of objects and relations. A total of 900 different 9-object scenes were used for training the model. For testing, both sparse and dense configurations are used. The sparse test set was composed of 50 9-object, 25 6-object, and 25 12-object scenes. The dense test set was composed of 50 6-object, 50 8-object, and 50 9-object scenes. For each scene above, the robot arm approached from four different random directions. Each object in these scenes had radii between 8 cm to 16 cm. In the evaluations, separate models were trained and tested on scenes where only fixed joints and mixed types of joints exist.

7.3.2. Implementation Details

Our physic prediction module takes object position, velocity and radius as object features, and joint relation type between objects as relation features. More specifically, object encoder is a MLP with 3 hidden layers of 150 neurons, and it takes object radius and velocity as inputs. The relation encoder is a MLP with 1 hidden layer of 100 neurons. It takes radius, joint relation type, and position and velocity differences between the objects as input. While our relation propagator is an MLP with 2 hidden layers of 150 neurons, our object propagator is an MLP with one hidden layer of 100 neurons. During training, we validated our physics prediction module on the validation set containing instances from the sparse configuration and selected the model that has the lowest mean squared error (MSE) over 200 timestep trajectory roll-outs.

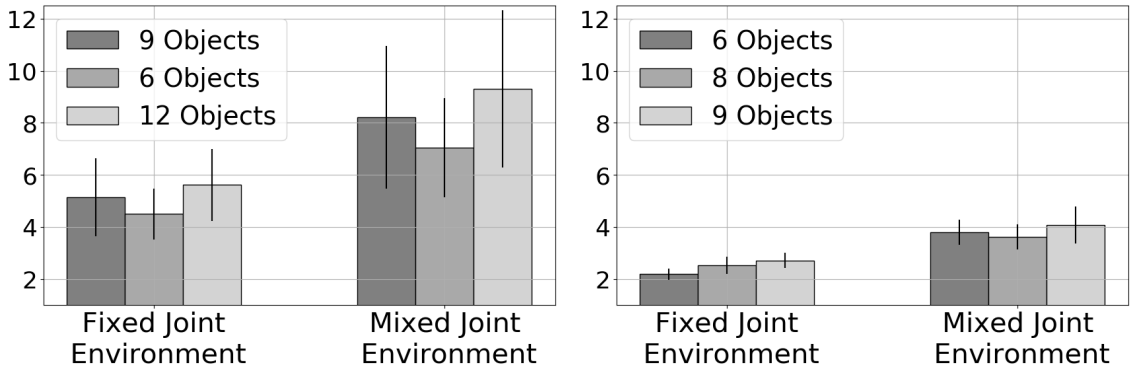


Figure 7.4. On left, error(cm) for object positions over 200 timestep trajectory roll-outs for sparse configuration, and on right, 50 timestep trajectory roll-outs for dense configuration.

Our belief regulation module uses the sequence of positions, velocities and radii of the objects, and predicts joint types between each pair of objects. The output of the relation propagator is connected to an LSTM with 100 hidden neurons. This LSTM is then connected to a fully connected layer to predict joint type between objects. This network was trained using sequences with 100 timesteps. During training, we optimized this network with the loss coming from the predicted joint types between the timesteps 50 and 100 to make sure that our model can generalize to the changing number of timesteps, while not over-fitting to the position information coming from the single timesteps.

For training our networks, we used batch size of 64, learning rate of 1e-3, and Adam optimizer [56]. The learning rate is decayed by 0.8 when there was no decrease in validation loss for 20 steps. Networks are trained for 500 epochs. For physics prediction module, at each epoch, half of the randomly shuffled data is used, and for belief regulation module, each epoch contained 100 batch of trajectories randomly sampled from training data. It took about one day to train the physic prediction module and one day to train belief regulation module using GTX 1080Ti GPU.

7.4. Results

For quantitative analysis, we compared our method with PropNets with alternative (hard-coded) relation assignments: As a strong baseline, PropNet$_{gt}$ uses ground-truth relations. PropNet$_f$ assumes all pairs of contacting objects have fixed relations between them. PropNet$_n$ assumes no joints between objects. Furthermore, to analyze the influence of temporal data in predicting relations within our model, we also report results with 1-step BRDPN that predicts object relations using only the observation from the previous step.

7.4.1. Quantitative Analysis of Separate Modules in Simulation

First, the physics prediction module is evaluated given ground-truth relation information. Figure 7.4 presents the performance on the test set for different object configurations. Each bar provides the mean error averaged over differences between predicted and observed trajectories. We evaluated for both the sparse and dense configuration settings in *fixed-joint* and *mixed-joint* environments separately. As shown, around 7 and 3 cm mean error is observed in sparse and dense object configurations. Furthermore, we observed that the error drops significantly (to 4 and 2 cm) in case only fixed joints are included. Given the average motion (including zero motion in many cases) in objects is $40cm$ sparse and $18cm$ in dense configuration, these results show that the model achieves high prediction performance if it uses the ground-truth relations; and with the increasing complexity of object relations, learning becomes more challenging.

Next, the performance of our belief regulation prediction module is evaluated on the sparse test set. As shown in Figure 7.5, the accuracy is already very high from the instant when the robot makes its first contact in *fixed-joint* environments. The accuracy increases in *mixed-joint* environments to over 98% as well, with the accumulated observations from the interactions of the robot.

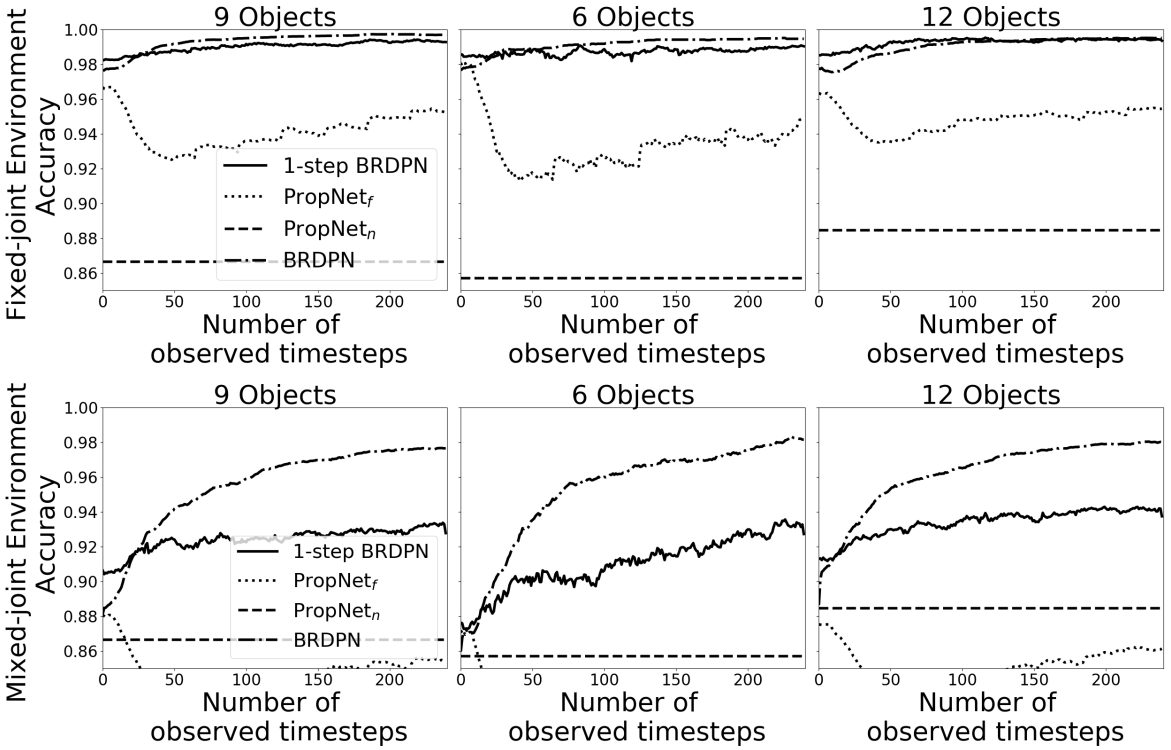
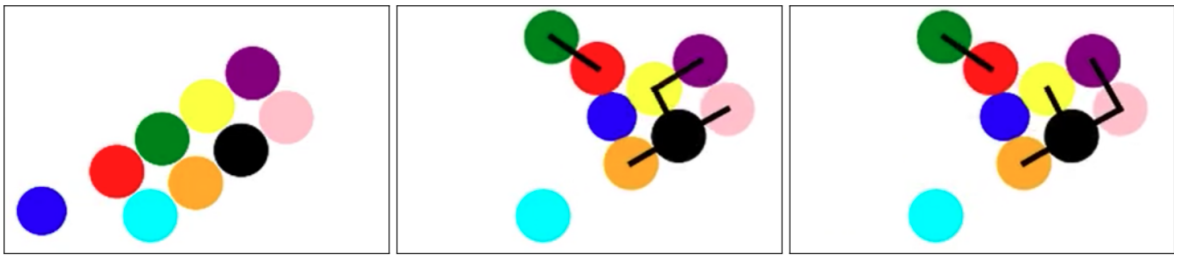


Figure 7.5. Relation prediction accuracies (sparse configuration).

We performed several experiments in the dense configuration as well. However, we observed that directly comparing real and predicted relations in this configuration might be misleading as different sets of joints that connect the objects in the same grid might generate identical effects in response to the robot interactions. The system might suffer from ambiguities in predicting joint relations from such interaction experience. For example, a group of objects that form a rigid body through a different set of connections would behave the same in response to the push action. Figure 7.6 provides a snapshot of such a case where the robot started interaction with 8 objects placed on a grid. In this case, even if the joint relations were incorrectly predicted for the sub-group of 5 objects, this was a plausible inference that enabled the system to make correct predictions about the object trajectories from that moment. While incorrect state predictions might not affect the effect prediction performance of the system in this particular extreme example, we might need intelligent exploration strategies that enable the robot to collect more reliable information in other ambiguous cases.



(a) Initial Scene (b) Predicted (c) Ground-Truth

Figure 7.6. (a) the setting is shown, where the end effector of the robot (shown in blue) moves towards an object group. (b) figure shows the predictions on fixed joint relations (black lines) and (c) provides the ground-truth relations.

7.4.2. Quantitative Analysis of the BRDPN in Simulation

In this section, the complete system is evaluated at different time-points during interactions. The belief regulation module predicts the relations between objects using the observations up to the corresponding time-points. Given the states of the objects, the robot actions, and the predicted relations between pairs of objects, the physics prediction module finds the trajectories of the objects that are expected to be observed for the rest of the motion. The results are presented in Figure 7.7 and Figure 7.8 where the errors on the remaining trajectories are computed with the predictions of the system at the reported timesteps. These results indicate that even if the relations are unknown, the proposed belief regulation improves the effect prediction performance of the system with more interaction experience. While BRDPN performs better for the sparse dataset, 1-step BRDPN performs similar to or better than BRDPN in dense configurations probably because the model was optimized for temporal information coming from sparse environments. Note that BRDPN outperforms the PropNets variants that do not utilize the ground-truth relations. PropNet$_{gt}$ has the best performance since it has access to the ground-truth relation assignments. However, in real world, joint relations may not have been provided to system.

7.4.3. Real World Experiments

In this section, we provide the results obtained in the real world. For this, the model trained in the simulator was directly transferred to the real world. A mallet that was grasped by the UR10 robot was used to push objects. Only one type of joint, namely fixed joint was used in this setup. Fixed joint relations are accomplished by placing customized card-boards under the specified objects, making all the group move together. A top-down oriented RGB camera with 1920×1080 pixels resolution was placed above the scene, ARTags were placed on the objects for tracking.

First, we present the results qualitatively over two example scenes. In the first example scene, 6 objects were placed as a group as shown in Figure 7.9, where the top left 3 objects and the bottom right pair were connected to each other. A straight push motion was executed by the robot and the object positions at timesteps 10 and 60 were provided. Solid and dashed lines show the real and predicted trajectories. As shown, given ground-truth joint information, the model made almost perfect trajectory

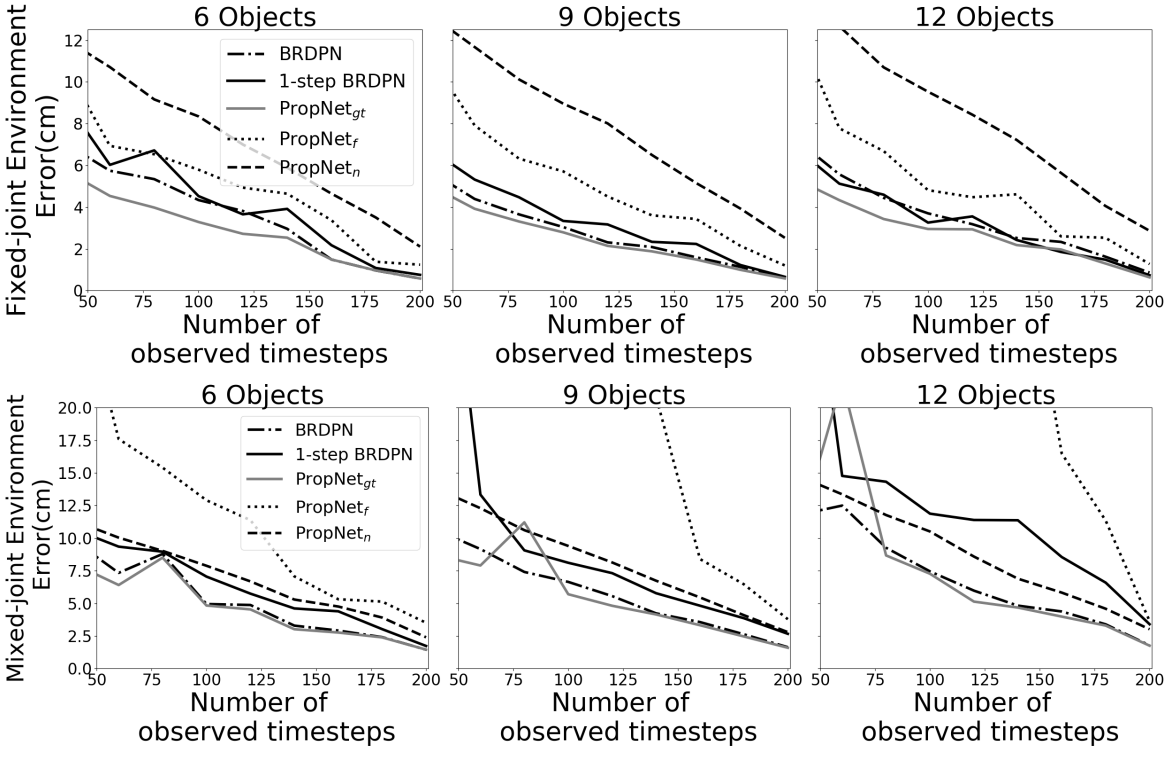


Figure 7.7. Error of the BRDPN in sparse configuration.

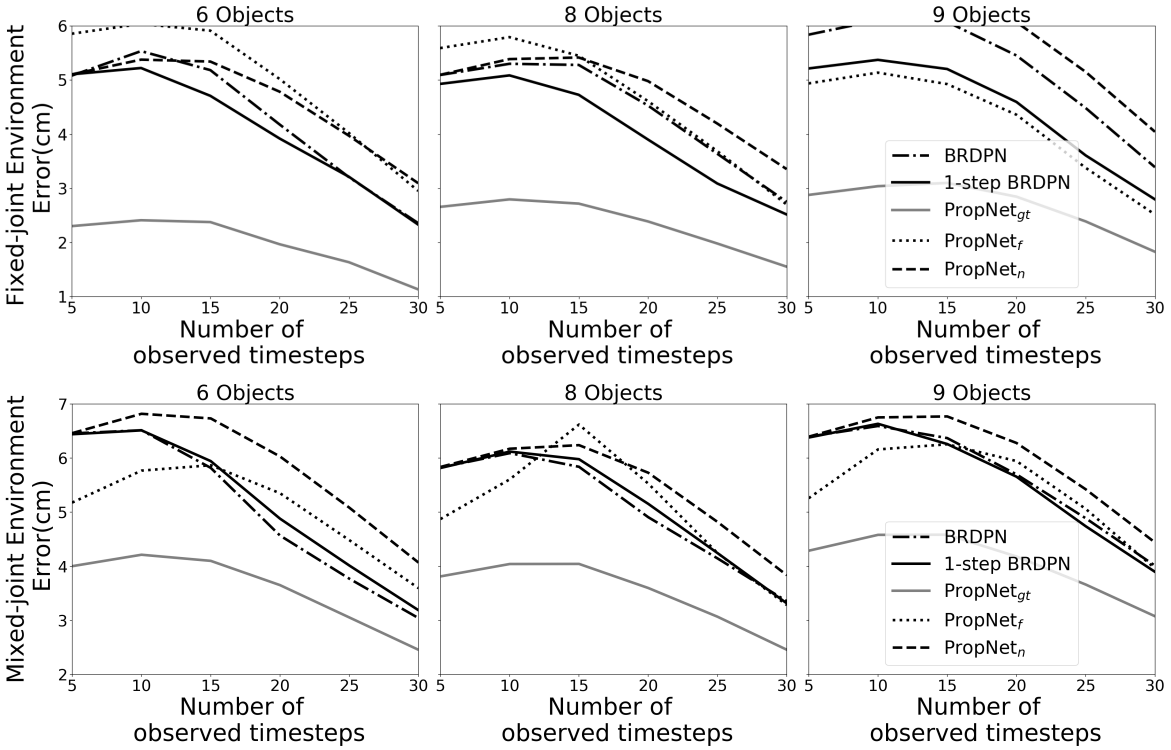


Figure 7.8. Error of the BRDPN in dense configuration.

predictions. When the ground-truth relations are not provided, as in PropNet$_n$ and PropNet$_f$, the model either predict that all objects are pushed aside or all contacting ones move together. Finally, when the relations are predicted, first the model predicts trajectories similar to PropNet$_f$ case, but after seeing the independent motions of upper three object group, it corrects the joint relations and predicts the correct trajectory successfully. In the second example scene, a more challenging configuration was used, where 7 objects were placed in two separate groups and objects in each group are attached to each other (Figure 7.10). The end-effector made a zigzag motion towards the objects. The relation prediction on the first group (the one closer to the robot) was correct at t : 50, since the robot had sufficient interaction with these objects. The indirect contact to the second group via the first group took place slightly before t : 140 and the robot correctly inferred that not all the objects in the second group had fixed relations. The prediction of the first group remained correct, but the robot made incorrect predictions in two cases: it incorrectly inferred that the first and second group was connected and that the top-right pair was also connected. With further interaction, these incorrect inferences were corrected at t : 150.

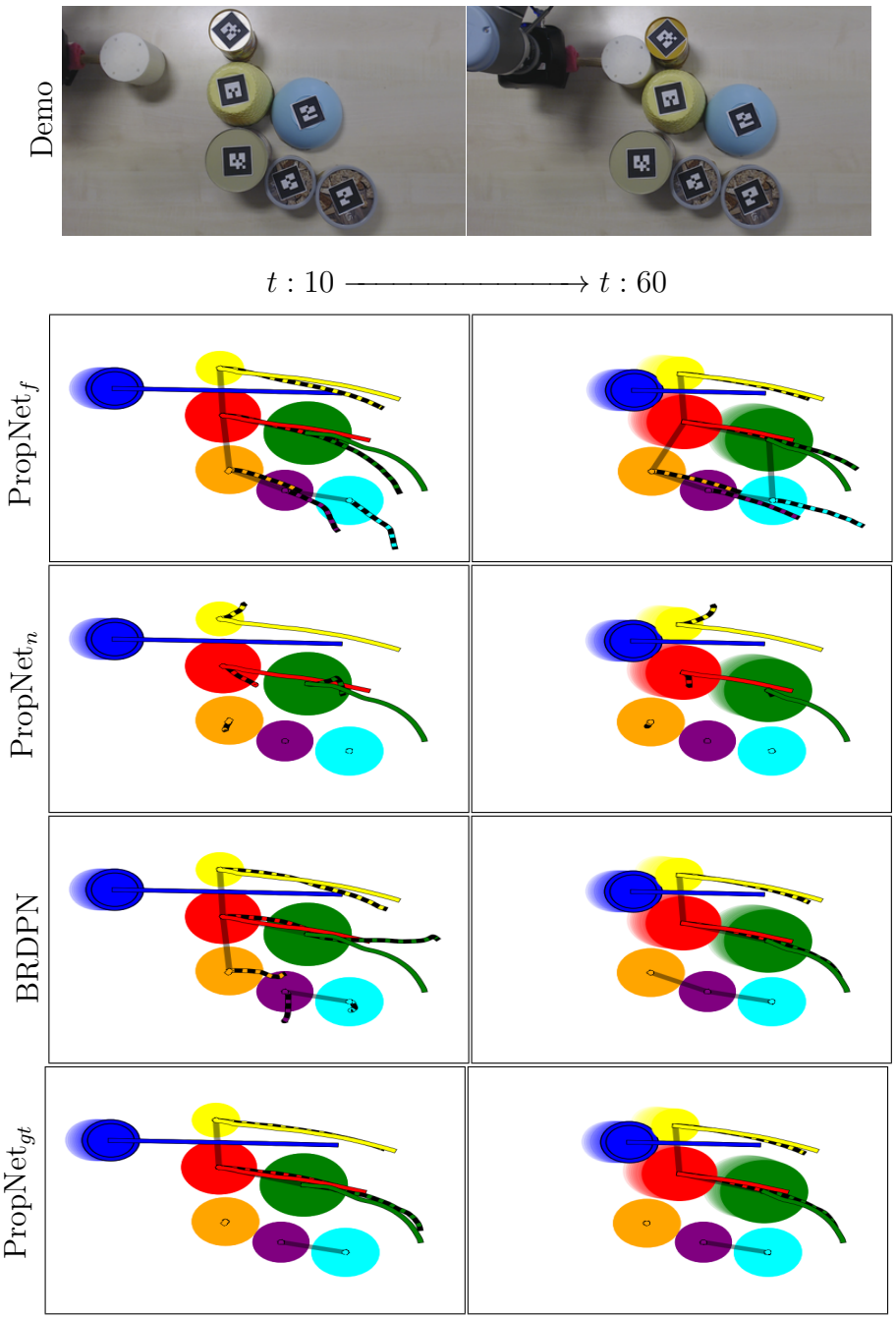


Figure 7.9. The first real-world interaction example. The relation assignments/predictions (black), the real (solid colored) and the predicted (dashed colored) trajectories are shown.

Finally, we evaluate our model quantitatively with a large number of interactions. We generated 102 different setups that include 2 to 5 objects with 1 to 3 connections. One of the 5 different predefined straight motions of 30 to 60 cm was applied towards these objects that were placed in different locations which results in objects moving 19.5 cm on average. Our model achieved an average error of 6.6 cm in predicting their

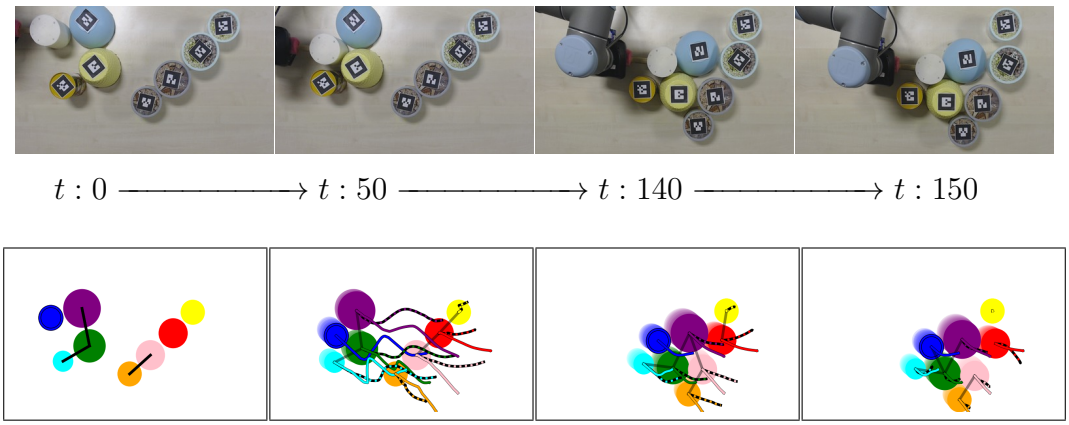


Figure 7.10. The second real-world interaction example.

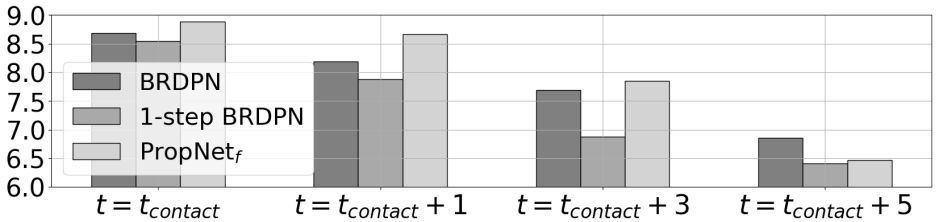


Figure 7.11. Average error (in cm) in the real world.

final positions. Although in some cases incorrect effect predictions caused failures in predicting the movement direction of interacted objects, our model performed well considering the average diameter of 12 cm of the objects and our direct transfer strategy from the simulation. Figure 7.11 provides a more detailed analysis of the results focusing on the time-point when the first contact with the objects occur. As shown, the prediction error of 1-step BRDPN quickly drops compared to the model that assigns fixed-joint to objects whose distances are smaller than 2.5 cm. Probably after the objects physically separated from each other, PropNet$_f$ does not consider those objects to be attached to each other and also start making predictions with similar accuracy. Note that PropNet$_n$ significantly under-performed and was not included in the figure, and ground-truth-relation model generated higher performance consistently, obtaining around 4 cm error at the end.

7.5. Conclusion

We presented *Belief Regulated Dual Propagation Networks (BRDPN)*, a general-purpose learnable physics engine that also continuously updates the estimated world

state through observing the consequences of its interactions. We demonstrated our network in setups containing articulated multi-part multi-objects settings. In these settings, we validated our network and its modules on several test cases which have shown both strengths and weaknesses of the proposed methods. While our system was validated in both simulation and real-world robotic experiments, we discussed that intelligent exploration strategies that resolve the inference problem in ambiguous situations are necessary. In the future, we aim to study on generating goal-directed action trajectories that balance the trade-off between exploration and exploitation. Furthermore, we plan to use the learned effect predictions in making multi-step plans in potentially sub-symbolic [59] or symbolic [60] spaces.

8. SCALING UP TO MORE COMPLEX TASKS AND FUTURE DIRECTIONS

The proposed framework is very generic and has a lot of points that could be further improved. In this section, we will discuss our improvements to BRDPN [12] model, will present our extensions to the environment and use cases. Finally, future directions will be explained and discussed.

8.1. Improvements on Training

In the training of the model, to reduce the total number of parameters, and to make future detection more robust, we have made some improvement on model training. These improvements will be discussed in this section.

8.1.1. Weight-Sharing

In our previous work, our method required two modules to be trained separately. This was the case since belief regulation module required partial state information since some of the parameters are needed to be predicted. In our experiments with model structure, we find that we can set a default value to these parameters, and can predict actual parameters. Our further experiments show that we can first train physic prediction than directly use propagation network in physic prediction to train LSTM used in belief regulation. This process decreased the number of the learned parameter by about thirty percent.

8.1.2. Scheduled Sampling

In our network, to increase the performance of physic prediction, we used scheduled sampling [13]. In sequence data, scheduled sampling is a curriculum learning strategy that allows the network to be trained not only from the true previous token

but also with the generated token. This makes the network more robust to situations where the network makes out of distribution predictions. For example, if the network predicts a state where an object is penetrating another object which contradicts with the rigidity of objects, it may make future predictions drift away from reality. However, with scheduled sampling, the model can do good predictions even in such situations.

8.1.3. Temporal Smoothing and Regularization

In belief regulation, it is important to increase accuracy in predictions as time progress. Normally, when applying loss to the output of the network, the model does not differentiate between first time-steps and later time-steps of the output. By applying different weights on different timesteps, we can acquire a smoother increase in accuracy as robot interacts with more objects. Besides, to make sure that predictions don't oscillate and jump between two very different predictions, we regularized changes between successive timesteps of LSTMs outputs.

8.2. Scaling to More Complex Domains

Previously, we used environments consisting of cylinders. Many objects in daily life can be said to be composed of cylinders(Like jars, cups) and cuboids(Like books, boxes) and other times, they can be a combination of both. A tennis racket can be represented by a cuboid and cylinder. By modelling cuboids in addition to cylinders in our network, we can represent and predict effects on many daily objects.

The cylinder can be represented with its radius and its state can be represented with its position. However, the cuboid is consists of two sides, have and angle that affects its contact with other objects. Simply representing it with its shape and its current angle will result in inconsistencies since, for the same situation, there can be multiple state representations. For example, if two sides of the cuboid are flipped, it will have the same effect but different representations. Same can be said for the angle. For propagation of motion between objects, their relative angle is more relevant. One of the benefits of graph neural networks is relational inductive biases, by allowing data

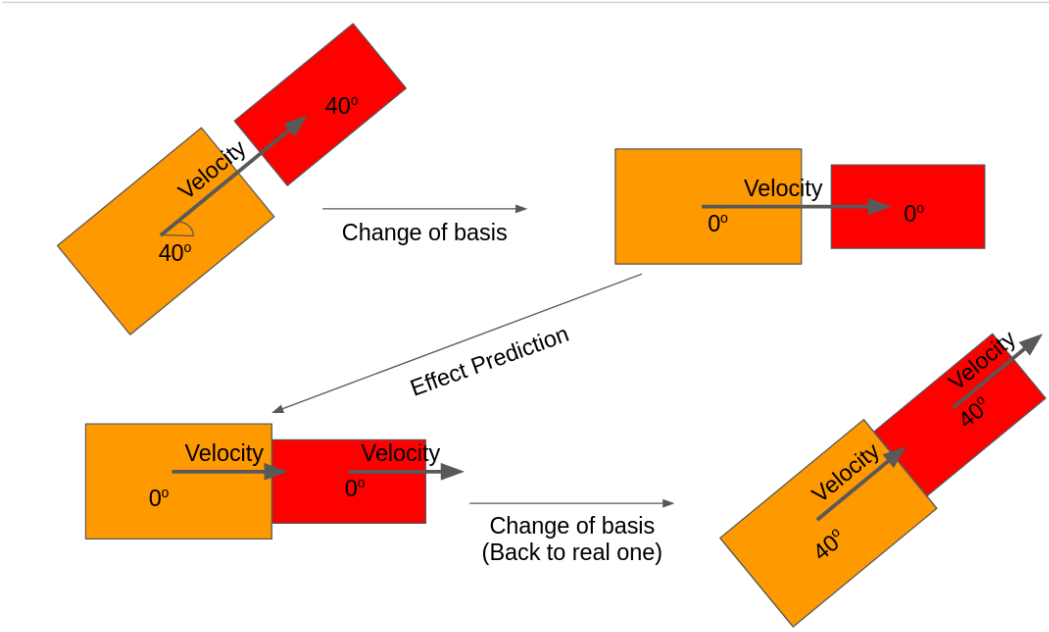


Figure 8.1. Visual illustration on how effect prediction is done on objects with different frame. By adding an relational inductive bias on objects frame, we can do prediction that is consistent with changing object frame

processing in node-edge level. This allows using relative angles, in addition to relative positions. We take inspiration from transformations widely used in robotics where objects, robots and their joints' frames are represented relative to each other. We represented each receiver and sender object-object relations with respect to receivers frame. A visual illustration of this is shown in Figure 8.1. In addition, since using raw angles results in discontinuities around change points (Between pi to -pi), to prevent gumball lock, we used sin and cos of said angles to make it continues.

We evaluated our networks in a dataset containing 30000 training trajectories. For testing, we used trajectories consisting of 9, 6, and 12 objects each with 1000 trajectories. While collecting trajectories, to make sure that the system learns multi-joint dynamics better, we changed probabilities of different object and joint generation every 25 trajectories. For example, in some runs, it is more likely to generate cuboids objects with a prismatic joint between them. This allowed the system to encounter more extreme setups, while at training preventing the system to discard these trajectories because they are outliers.

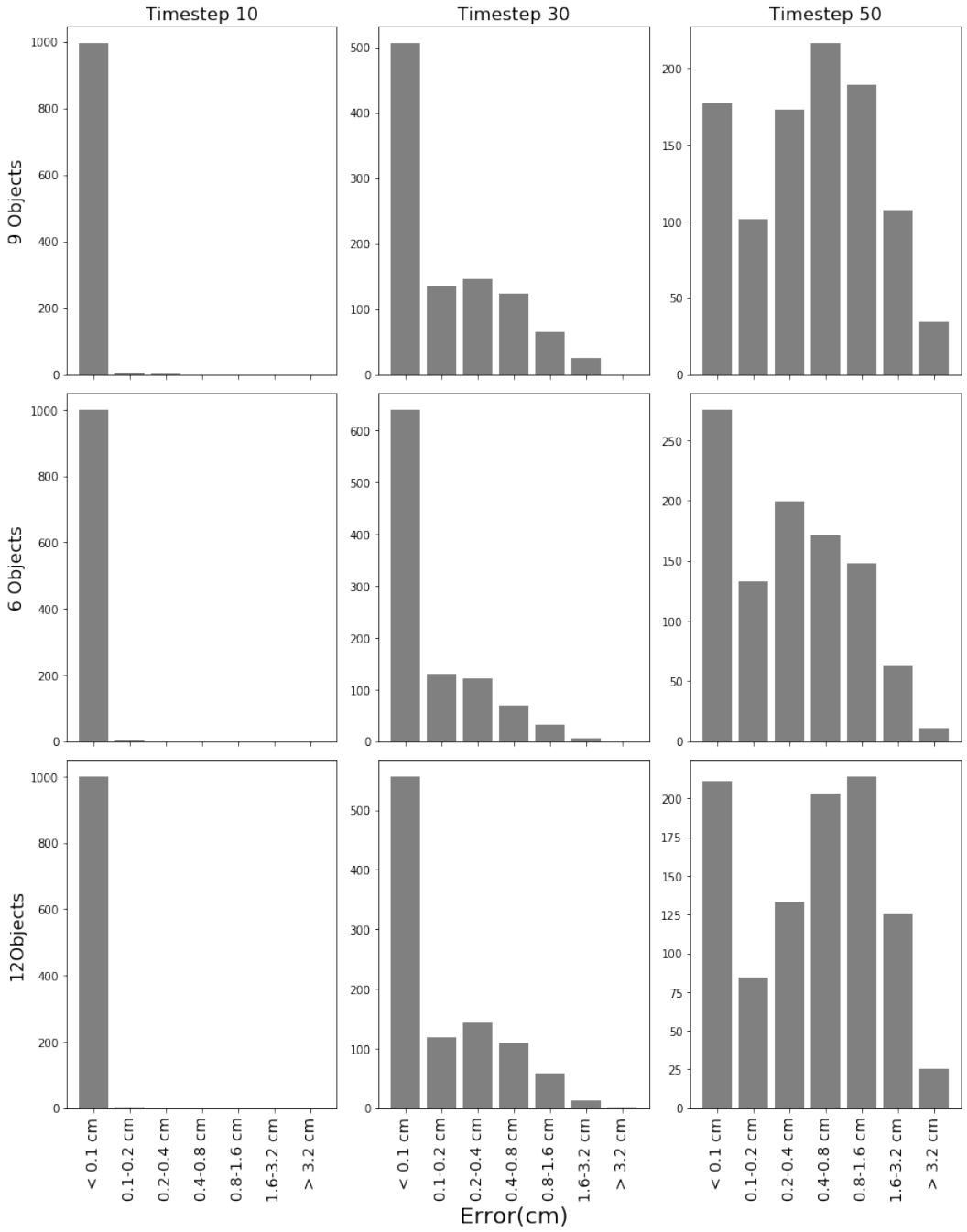


Figure 8.2. Physics prediction results of BRDPN on more complex environments.

We changed network size to handle more complex scene dynamics. Now, the object encoder is an MLP with 2 hidden layers of 256 neurons, and the relation encoder is an MLP with 3 hidden layers of 256 neurons. Our relation and object propagator is MLP with 1 hidden layer of 256 neurons. For training our networks, we used batch size of 16, learning rate of 3e-4, and Adam optimizer [56] with amsgrad [61] . The learning rate is decayed by 0.8 when there was no decrease in validation loss for 20

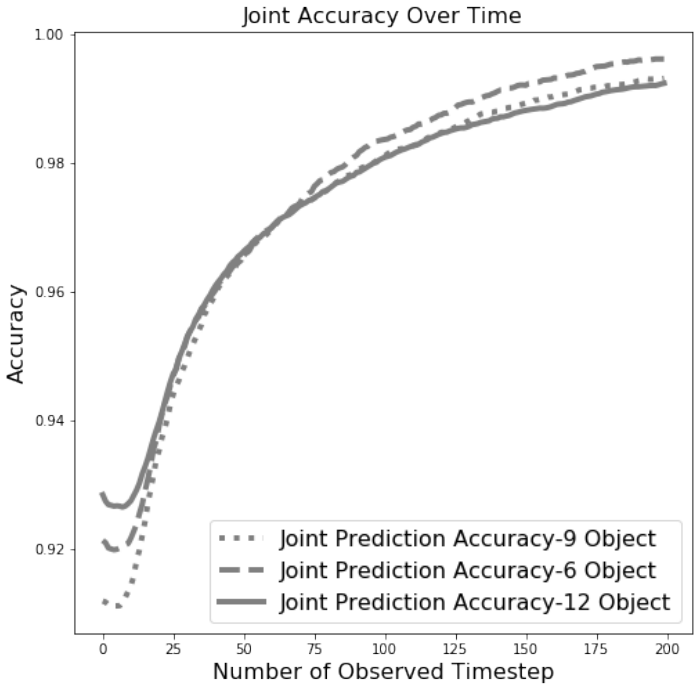


Figure 8.3. Belief regulation results of BRDPN on more complex environments.

steps. Networks are trained for 1000 epochs. For physics prediction module, at each epoch, 5000 were used. First, physics prediction part of the network was trained. After the training is complete, weights of shared part of the network are freeze and belief regulation module is trained. While training belief regulation, each epoch contained 200 batches of trajectories randomly sampled from training data. It took about two days to train the physic prediction module and one day to train belief regulation module with Nvidia P100 GPU.

Our results can be found in Figure 8.2 and Figure 8.3. As can be seen, our model acquired very good results on both physic prediction and belief regulation. As can be seen in Figure 8.2, most times, our model acquired predicted trajectories that are very close to the ground truth. However since the task is the autoregressive prediction of future states, one small error can result in trajectory to drift away from the ground truth. Similarly belief regulation succeeds in the prediction of joints between objects. As can be seen in the Figure 8.3, initially, joints between the objects are not reliably known and has the performance of predicting each joint relation as no-joint. But as robot interacts with them, joints between objects are predicted. Overall, our networks

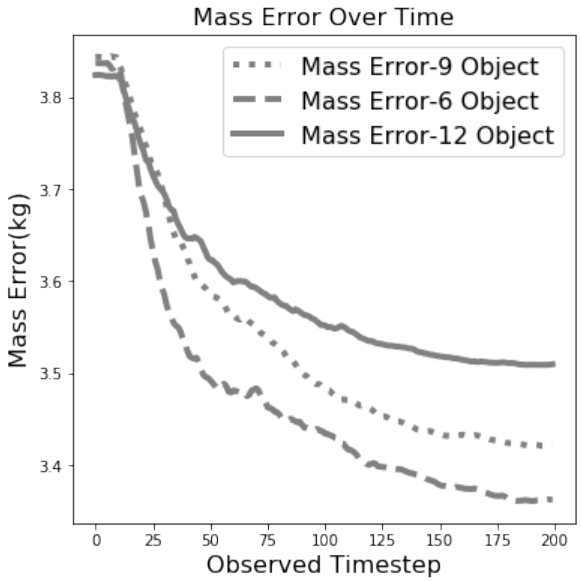


Figure 8.4. Mass prediction results.

have acquired very good performance both in physics prediction and belief regulation. Further results for physic prediction and belief regulation can be found in this links [62] [63].

8.3. Using BRDPN For Prediction of Object Parameters

In the previous work, we have shown BRDPN on relation prediction task. However, in the formulation of BRDPN, it was also possible to do reasoning on the object side. In this section, we design a task around showing capability of BRDPN on object reasoning side.

As a task, we selected mass prediction. Mass prediction just from pushing is very hard. For example, if the robot pushes a single object, it is very unlikely for the model to predict the mass of pushed object without using some kind of force sensor input. However, in multi-object case, from object-object interactions, the network can reason about objects relative masses. For example, if the robot object pushes a cylinder towards another cylinder object, if the former object is heavier than the other one, it will continue to push it forward. But for example, in case the opposite is true, the former object will flip to either side.

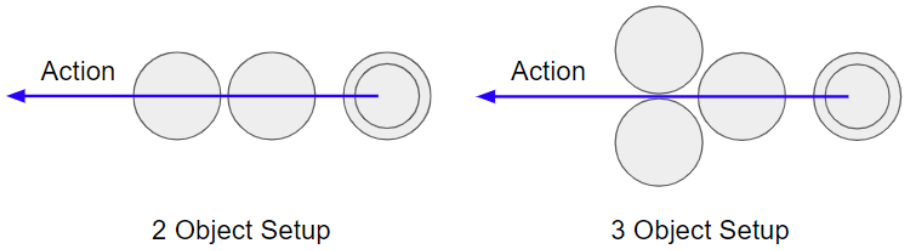


Figure 8.5. Controlled environment setups.

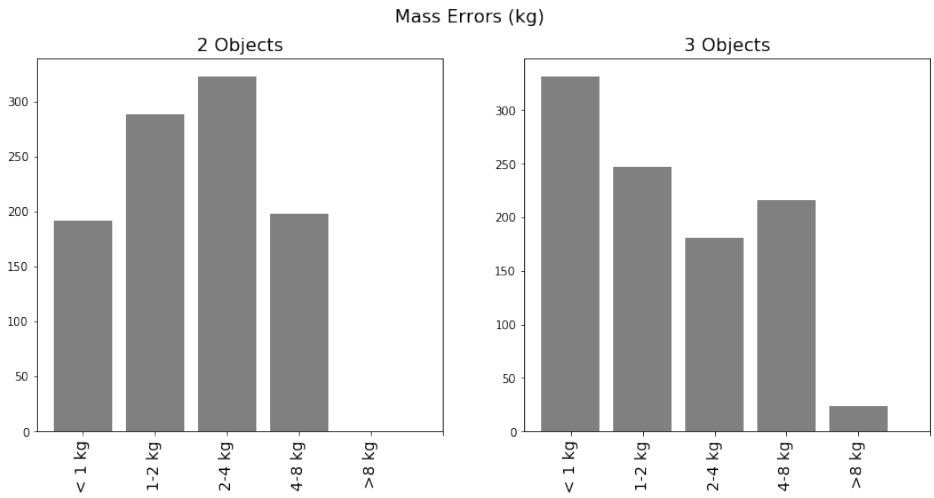


Figure 8.6. Mass prediction results in controlled environments.

From this observation, we created an environment where an object can have three mass ranges: 200-500 gr, 1000-2000 gr, 8000-10000 gr. Since this problem is quite hard as well, we limited the system to not have any joints between objects. Additionally, the effect of mass on cuboid objects are more discrete compared to cylindrical objects, and most of the time doesn't give any clue to network about the mass of objects. So we limited objects to be cylindrical. We generated 32000 9-object trajectories with previously told mass ranges. In this task, in the training setup, objects are placed in a more dense area. The robot does three linear push toward objects, while not selecting previously selected ones.

Result of mass prediction can be seen in Figure 8.4. In our tests, we see that as time goes, mass prediction become better, however it is not noticeably good. This is partly because the robot doesn't interact with most objects in this setup. To better see whether our network succeeded in mass prediction, we prepared 2 controlled

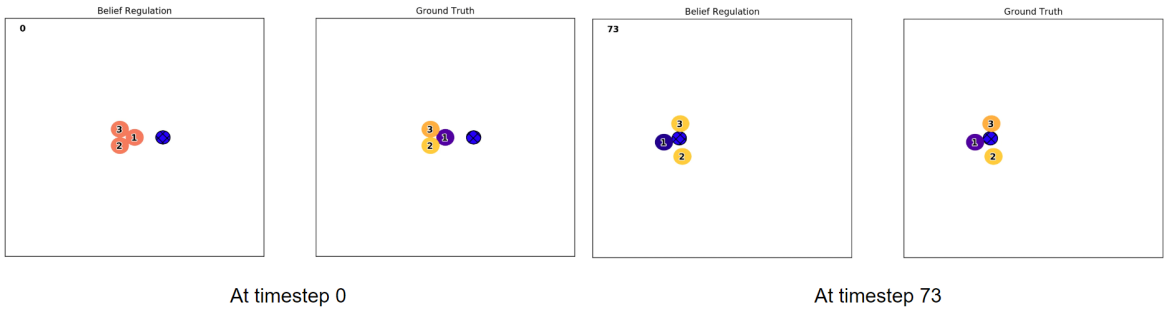


Figure 8.7. Correctly predicted ambiguous scene example.

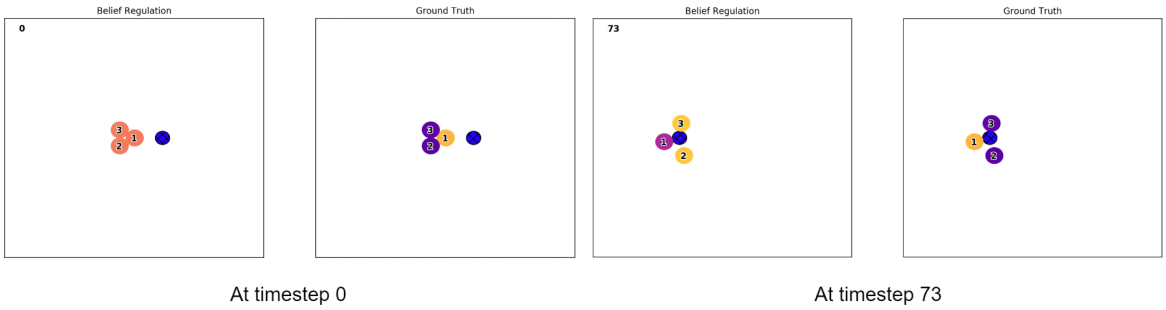


Figure 8.8. Incorrectly predicted ambiguous scene example.

environment setups. These setups can be seen in Figure 8.5. In this setups, we only change the mass of objects, however, all other parameters are the same. The robot has interaction with each object so it should be possible for the network to predict mass if it is predictable. We acquired results seen in Figure 8.6. As can be seen, our network acquired better results. However, these setups are still very ambiguous. Two of our result can be seen in Figure 8.7 and Figure 8.8. In these two scenes, the robot observes very similar trajectories with 0.15 cm difference between them, despite the interacted objects having very different masses. In these scenes, the network makes the same predictions. However, only in the first scene, it is correct. More of our mass predictions can be found in link [64].

8.4. Tool Usage

In Section 8.2, we have shown that we can represent complex multi-part scenes with graph neural networks. In this section, by using the same representation and network trained on aforementioned complex environments, we represented completely novel tools with graph neural networks and used them in tool manipulation and plan-

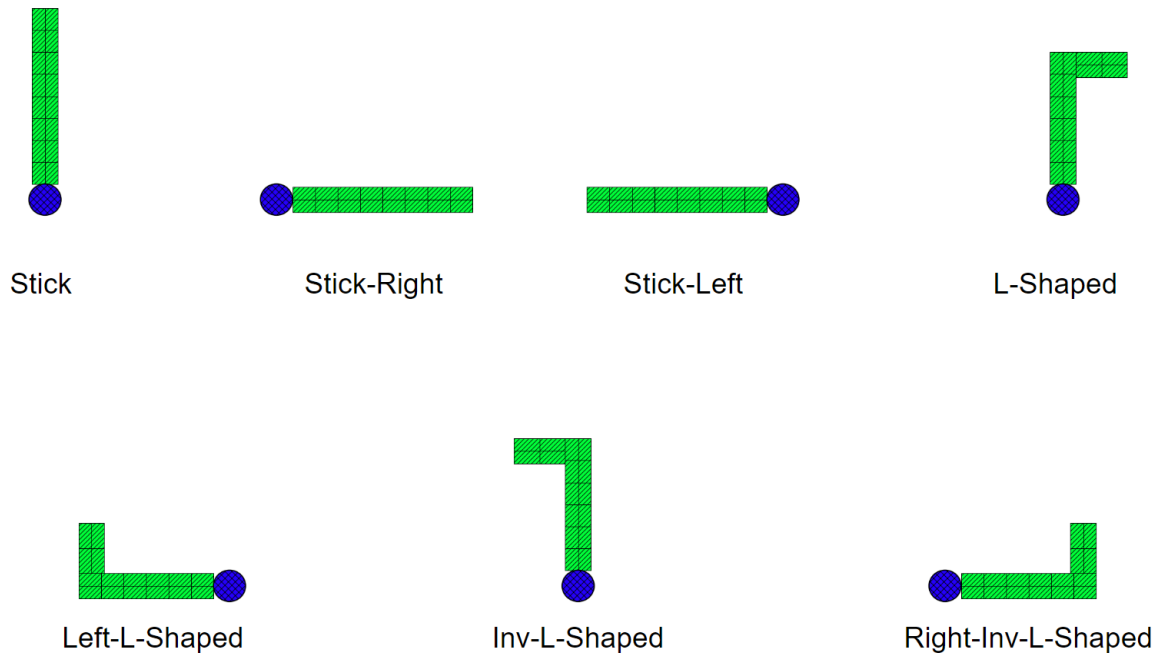


Figure 8.9. Tools used In tool selection and planning experiments.

ning to achieve goals. This is done by following.

We aimed to model widely used tools. For these, we divided into multiple parts and represented each part with a cuboid. These widely used tools can be seen in Figure 8.9.These tools are used as if they are attached to the robot end-effector. However, we didn't train the new network and tried to see the performance of the network trained in Section 8.2.

We defined actions as 20 cm pushes in principal directions. In these actions, tool motion is modelled kinematically, and not updated from network prediction but from. As we tested these actions, we find out that except forward push, the model can't generalize well to these actions. Instead, we modelled each of these actions as forward pushes, by changing the position of the robot end-effector. From our new observations, we see that network can now predict most pushes in each direction.

To show the capability of the system we defined, we created a tool selection experiment. We used tools shown in Figure 8.9. We generated one cylindrical object randomly in $3m^2$ reachable area in front of the robot end-effector and we generated

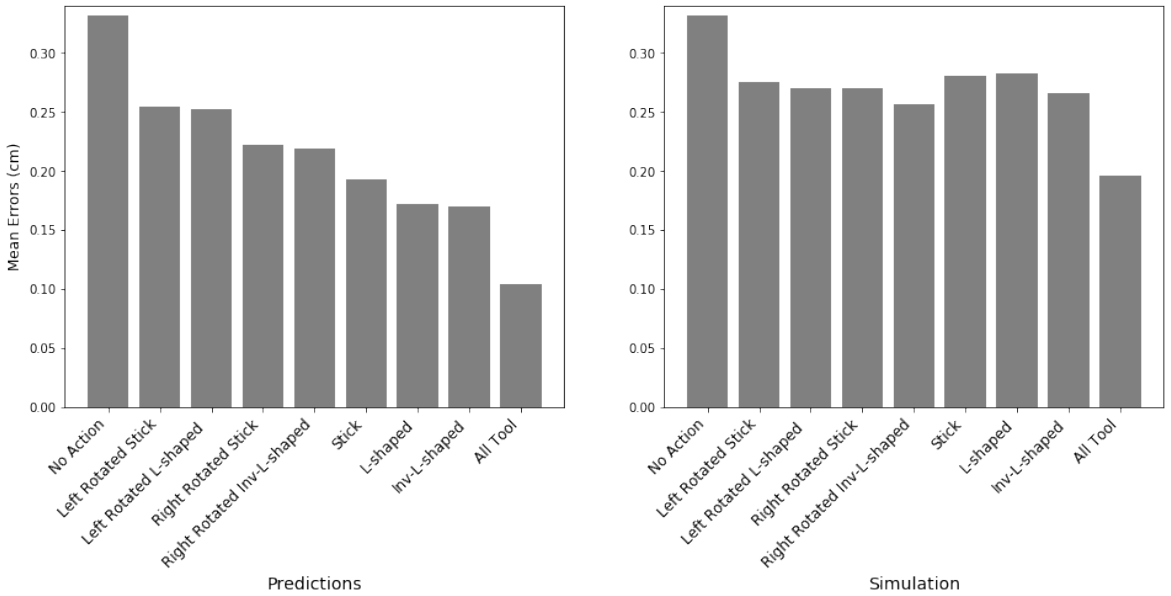


Figure 8.10. Comparison between acquired error after planning.

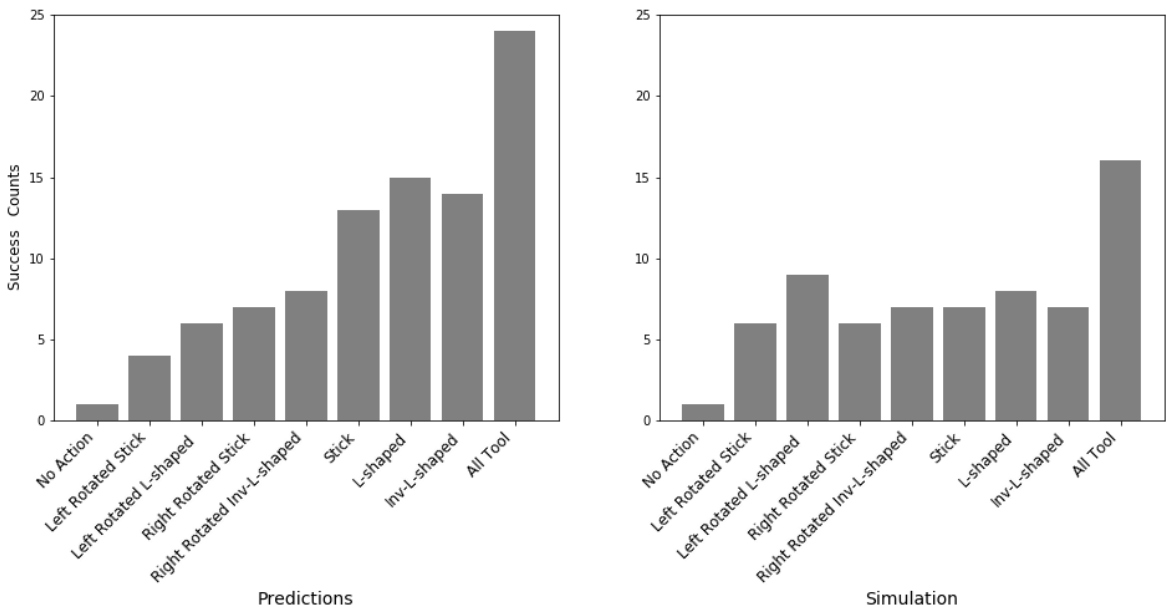


Figure 8.11. Comparison between number of successfully solved tasks for each tool.

a goal position in $16m^2$ area around of the generated object. We defined the task as the selection of the best tool and best action from all possible tools and actions. We transferred the found best tool and best action to test it in simulation. We acquired errors seen in Figure 8.10. Additionally, we decided on 10 cm threshold, and after any selected action, if the distance between object and target position is less than this threshold, we counted that actions as a success. Number of the successfully solved task for each tool can be seen in Figure 8.11. Note that in here, prediction results

can be thought as best acquirable results for each tool and can be thought of as an upper bound. In many cases, there can be no solution to the generated task, as can be seen in success counts on predictions in Figure 8.11. Overall, it can be seen that our model can reason about tool selection and when all tools are used, the system works significantly better. Videos related to transfer and comparison of selected tools, and how all tasks are solved with these tools can be found here [65].

8.5. Other Extensions

In addition to previously mentioned extensions, we managed to predict 6D rigid body motions that are very close to the ground truth. As a task, we selected levering up of PCB from HDD with a screwdriver tool. By representing HDD and PCB with multiple cuboids, our model managed to learn the effect of lever-up action. Edges between objects are created as they get close to each other. The network is trained using 500 lever-up interactions using 125 different procedurally generated hard-disks(Details on how they are generated can be seen in Figure 8.12). Results can be seen in this video [66]. For clarity, different parts of HDD are shown in different colours.

8.6. Future Directions

In this section, we explained our current extensions to the proposed BRDPN framework. Overall the proposed framework is very generic, and it can be transferred to many different tasks and use cases. We have discussed our improvements in the training of the framework. We have shown that BRDPN can learn the task even in more complex domains.

It can be noted that there are still many improvements that could be made. For example, in belief regulation, actions are not task-directed, and as environment complexity increase, doing random actions may not lead to correct predictions. This can be also partially seen in Section 8.3. The models of the system can be significantly enhanced by adding an action selection. This is an interesting research area since robots need to consider different objects in the scene and may have to do long term

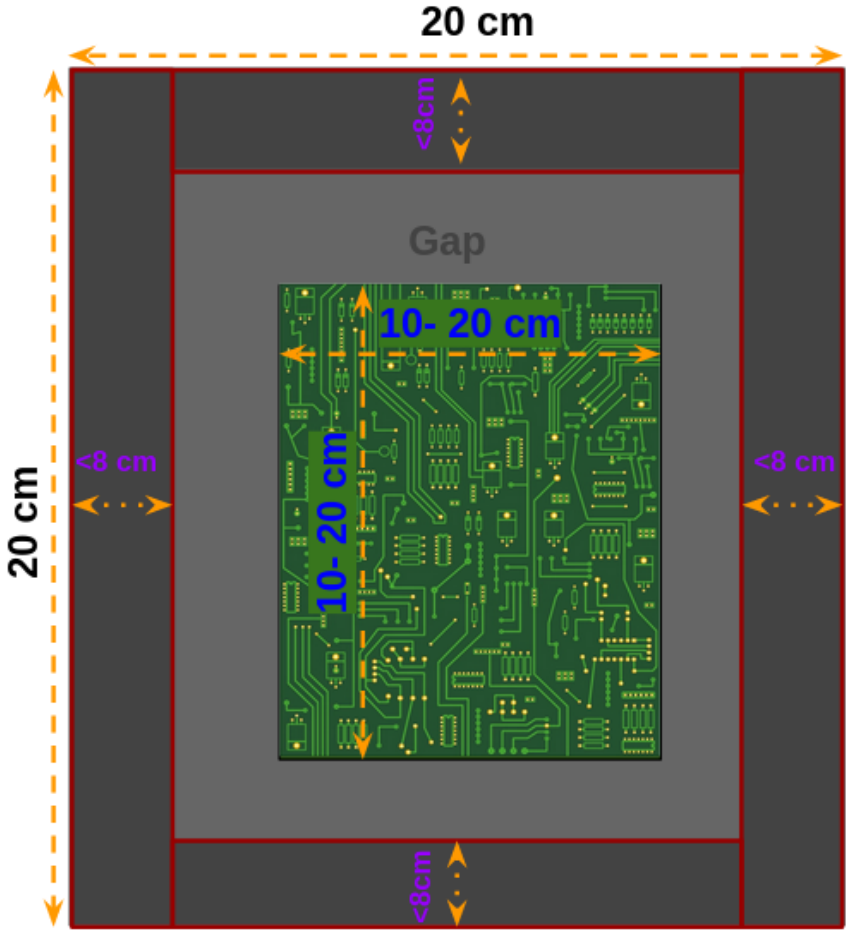


Figure 8.12. Scene generation of procedurally generated HDDs.

planning for hard to predict parameters of the system.

Another interesting area is 3D predictions. In 3D, predictions become significantly harder, and it introduces new problems like partial observability of the scene. Lately, 3D deep learning is becoming popular, and many tools are started to appear for researchers to use. Graph neural networks area is closely related to 3D deep learning and is used for modelling particle dynamics. For example, soft objects can be modelled with many particles and can be represented with graph neural networks. For such objects, interaction experience can be used as input for correcting belief about the shape of the object.

9. CONCLUSION

In this thesis, we have shown how we can use predictive models for reasoning about action-effects in push manipulation tasks. We investigated action-effect prediction of single objects with various shapes and prediction of the effects of robot actions on multi-object tasks. We validated all our approaches with in-depth analysis.

Firstly, in Chapter 5, we have shown that LSTM models can be used for prediction of effect trajectories in single objects. However, this work was limited to lever-up action and a single object. Nonetheless, it succeeds on the prediction of complex trajectories, which pushed us to improve networks capabilities. Then in Chapter 6, we extended the network to to be more general. In this work, we investigated how various shape descriptors can be used with our effect prediction LSTM model. We have thoroughly analyzed and compared the advantages and disadvantages of compared feature sets. We have found that prediction models with manually engineered features have the best performance out of all features. However, since manual engineering for each task is not feasible, we have found that for general purpose usage, Im-CNN based trajectory prediction model is the best performer.

In Chapter 7, we presented a general-purpose learnable physics engine that can do action-effect reasoning on articulated multi-object systems. This framework consisted of two complementary components, a physics predictor and a belief regulator. The former predicts the future states of the object(s) manipulated by the robot, the latter constantly corrects the robots knowledge regarding the objects and their relations. Our results showed that after training in a simulator, the robot can reliably predict the consequences of its actions in object trajectory level and exploit its own interaction experience to correct its belief about the state of the environment, enabling better predictions in partially observable environments. Furthermore, the trained model was transferred to the real world and verified in predicting trajectories of pushed interacting objects whose joint relations were initially unknown. We compared BRDPN against PropNets and showed that BRDPN performs consistently well. Moreover, BRDPN can

adapt its physic predictions, since the relations can be predicted online.

Finally, since our proposed framework explained in Chapter 7 is very generic, we have made many extensions to this system. These were shown in Chapter 8. In this chapter, we first show our improvement in model training. By using scheduled sampling [13], we improved network performance, and by allowing networks to share weights, we decreased the number of parameters to learn. Then we have shown how using a representation that considers objects frames, we can model environments with cuboid objects and much more interaction. By modelling cuboids in addition to cylinders in our network, many daily objects can be represented with our network. This is partially shown in tool manipulation and planning by representing tools with cuboids and rigid joints. Besides, we have shown by using object property prediction part of BRDPN, we can predict weights of objects as robots interactions are observed.

We believe our system can be further refined and extended. We have shown just a glimpse on how BRDPN can be used for 3D predictions, by further using advantages of graph neural network architecture, we can model more complex environments and systems. Our experiments are limited to random actions, and not all actions were providing the system with the necessary information. We discussed that as the complexity of environment increase, we may need a mechanism for action selection even in data collection. So, as another future work, we aim to improve our system to do reasoning on the action side as well.

REFERENCES

1. European Union's Horizon 2020 research and innovation programme, *IMAGINE - Robots Understanding Their Actions by Imagining Their Effects*, `https://imagine-h2020.eu/`, accessed at May 2020.

2. Battaglia, P., J. B. C. Hamrick, V. Bapst, A. Sanchez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. Allen, C. Nash, V. J. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li and R. Pascanu, "Relational inductive biases, deep learning, and graph networks", *arXiv*, 2018, `https://arxiv.org/pdf/1806.01261.pdf`.

3. Battaglia, P., R. Pascanu, M. Lai, D. J. Rezende *et al.*, "Interaction networks for learning about objects, relations and physics", *Advances in neural information processing systems*, pp. 4502–4510, 2016.

4. Chang, M. B., T. Ullman, A. Torralba and J. B. Tenenbaum, "A Compositional Object-Based Approach to Learning Physical Dynamics", *arXiv preprint arXiv:1612.00341*, 2016.

5. Li, Y., J. Wu, J.-Y. Zhu, J. B. Tenenbaum, A. Torralba and R. Tedrake, "Propagation Networks for Model-Based Control Under Partial Observation", *International Conference on Robotics and Automation*, pp. 1205–1211, 2019.

6. Mrowca, D., C. Zhuang, E. Wang, N. Haber, L. F. Fei-Fei, J. Tenenbaum and D. L. Yamins, "Flexible neural representation for physics prediction", *Advances in neural information processing systems*, pp. 8813–8824, 2018.

7. Li, Y., J. Wu, R. Tedrake, J. B. Tenenbaum and A. Torralba, "Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids", *arXiv preprint arXiv:1810.01566*, 2018.

8. Watters, N., D. Zoran, T. Weber, P. Battaglia, R. Pascanu and A. Tacchetti, "Visual interaction networks: Learning a physics simulator from video", *Advances in neural information processing systems*, pp. 4539–4547, 2017.

9. Van Steenkiste, S., M. Chang, K. Greff and J. Schmidhuber, "Relational neural expectation maximization: Unsupervised discovery of objects and their interactions", *arXiv preprint arXiv:1802.10353*, 2018.

10. Tekden, A. and E. Ugur, "Kaldırma aksiyonuyla oluşan yörüngenin uzun kısa dönem hafıza modeliyle tahmini", *Türkiye Robotbilim Konferansı (Turkish Robotics Conference)*, pp. 206–211, 2018.

11. Seker, M. Y., A. E. Tekden and E. Ugur, "Deep effect trajectory prediction in robot manipulation", *Robotics and Autonomous Systems*, Vol. 119, pp. 173 – 184, 2019, http://www.sciencedirect.com/science/article/pii/S0921889019300740.

12. Tekden, A. E., A. Erdem, E. Erdem, M. Imre, M. Y. Seker and E. Ugur, "Belief Regulated Dual Propagation Nets for Learning Action Effects on Articulated Multi-Part Objects", *arXiv preprint arXiv:1909.03785*, 2019.

13. Bengio, S., O. Vinyals, N. Jaitly and N. Shazeer, "Scheduled sampling for sequence prediction with recurrent neural networks", *Advances in Neural Information Processing Systems*, pp. 1171–1179, 2015.

14. Hochreiter, S. and J. Schmidhuber, "Long short-term memory", *Neural computation*, Vol. 9, No. 8, pp. 1735–1780, 1997.

15. Christopher Olah, *Understanding LSTM Network*, https://colah.github.io/posts/2015-08-Understanding-LSTMs/, accessed at May 2020.

16. Universal Robots, *UR10 Website*, https://www.universal-robots.com/products/ur10-robot/, accessed at May 2020.

17. Robotic, *3-Finger Adaptive Gripper*, `https://robotiq.com/products/3-finger-adaptive-robot-gripper`, accessed at May 2020.

18. Roswiki, *Robot Operating System*, `http://wiki.ros.org/`, accessed at May 2020.

19. Coppelia Robotics, *CoppeliaSim Robot Simulator*, `http://www.coppeliarobotics.com/`, accessed at May 2020.

20. Rohmer, E., S. P. N. Singh and M. Freese, "V-REP: A versatile and scalable robot simulation framework", *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1321–1326, 2013.

21. Stephen James, *PyRep - A toolkit for robot learning research*, `https://github.com/stepjam/PyRep`, accessed at May 2020.

22. Chollet, F. *et al.*, "Keras", `https://keras.io`, 2015.

23. Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library", H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox and R. Garnett (Editors), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, Curran Associates, Inc., 2019, `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

24. Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in Python", *Journal of machine learning research*, Vol. 12, No. Oct, pp. 2825–2830, 2011.

25. Hunter, J. D., "Matplotlib: A 2D graphics environment", *Computing in science &*

engineering, Vol. 9, No. 3, pp. 90–95, 2007.

26. Oliphant, T. E., *A guide to NumPy*, Vol. 1, Trelgol Publishing USA, 2006.

27. Qwertyquerty, *Collision*, `https://pypi.org/project/collision/`, accessed at May 2020.

28. Kubricht, J. R., K. J. Holyoak and H. Lu, "Intuitive physics: Current research and controversies", *Trends in cognitive sciences*, Vol. 21, No. 10, pp. 749–759, 2017.

29. Battaglia, P. W., J. B. Hamrick and J. B. Tenenbaum, "Simulation as an engine of physical scene understanding", *Proceedings of the National Academy of Sciences*, Vol. 110, No. 45, pp. 18327–18332, 2013.

30. Hamrick, J. B., P. W. Battaglia, T. L. Griffiths and J. B. Tenenbaum, "Inferring mass in complex scenes by mental simulation", *Cognition*, Vol. 157, pp. 61–76, 2016.

31. Smith, K., L. Mei, S. Yao, J. Wu, E. Spelke, J. Tenenbaum and T. Ullman, "Modeling Expectation Violation in Intuitive Physics with Coarse Probabilistic Object Representations", *Advances in Neural Information Processing Systems*, pp. 8983–8993, 2019.

32. Deisenroth, M. and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search", *Proceedings of the 28th International Conference on machine learning (International Conference on Machine Learning)*, pp. 465–472, 2011.

33. Lerer, A., S. Gross and R. Fergus, "Learning physical intuition of block towers by example", *International Conference on Machine Learning*, pp. 430–438, 2016.

34. Groth, O., F. B. Fuchs, I. Posner and A. Vedaldi, "Shapestacks: Learning vision-based physical intuition for generalised object stacking", *Proceedings of the Euro-*

pean Conference on Computer Vision (ECCV), pp. 702–717, 2018.

35. Li, W., S. Azimi, A. Leonardis and M. Fritz, "To fall or not to fall: A visual approach to physical stability prediction", *arXiv preprint arXiv:1604.00066*, 2016.

36. Mottaghi, R., H. Bagherinezhad, M. Rastegari and A. Farhadi, "Newtonian scene understanding: Unfolding the dynamics of objects in static images", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3521–3529, 2016.

37. Mottaghi, R., M. Rastegari, A. Gupta and A. Farhadi, ""What happens if..." Learning to Predict the Effect of Forces in Images", *European Conference on Computer Vision*, pp. 269–285, Springer, 2016.

38. Fragkiadaki, K., P. Agrawal, S. Levine and J. Malik, "Learning visual predictive models of physics for playing billiards", *arXiv preprint arXiv:1511.07404*, 2015.

39. Kipf, T., E. Fetaya, K.-C. Wang, M. Welling and R. Zemel, "Neural relational inference for interacting systems", *arXiv preprint arXiv:1802.04687*, 2018.

40. Ye, Y., M. Singh, A. Gupta and S. Tulsiani, "Compositional Video Prediction", *Proceedings of the IEEE International Conference on Computer Vision*, pp. 10353–10362, 2019.

41. Sanchez-Gonzalez, A., J. Godwin, T. Pfaff, R. Ying, J. Leskovec and P. W. Battaglia, "Learning to simulate complex physics with graph networks", *arXiv preprint arXiv:2002.09405*, 2020.

42. Agrawal, P., A. V. Nair, P. Abbeel, J. Malik and S. Levine, "Learning to poke by poking: Experiential learning of intuitive physics", *Advances in Neural Information Processing Systems*, pp. 5074–5082, 2016.

43. Finn, C., I. Goodfellow and S. Levine, "Unsupervised learning for physical interac-

tion through video prediction", *Advances in neural information processing systems*, pp. 64–72, 2016.

44. Xingjian, S., Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong and W.-c. Woo, "Convolutional LSTM network: A machine learning approach for precipitation nowcasting", *Advances in neural information processing systems*, pp. 802–810, 2015.

45. Byravan, A. and D. Fox, "SE3-Nets: Learning rigid body motion using deep neural networks", *International Conference on Robotics and Automation*, pp. 173–180, 2017.

46. Janner, M., S. Levine, W. T. Freeman, J. B. Tenenbaum, C. Finn and J. Wu, "Reasoning about physical interactions with object-oriented prediction and planning", *arXiv preprint arXiv:1812.10972*, 2018.

47. Ye, Y., D. Gandhi, A. Gupta and S. Tulsiani, "Object-centric Forward Modeling for Model Predictive Control", *arXiv preprint arXiv:1910.03568*, 2019.

48. Sanchez-Gonzalez, A., N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell and P. Battaglia, "Graph networks as learnable physics engines for inference and control", *arXiv preprint arXiv:1806.01242*, 2018.

49. Wu, J., I. Yildirim, J. J. Lim, B. Freeman and J. Tenenbaum, "Galileo: Perceiving physical object properties by integrating a physics engine with deep learning", *Advances in neural information processing systems*, pp. 127–135, 2015.

50. Zheng, D., V. Luo, J. Wu and J. B. Tenenbaum, "Unsupervised learning of latent physical properties using perception-prediction networks", *arXiv preprint arXiv:1807.09244*, 2018.

51. Li, J. K., W. S. Lee and D. Hsu, "Push-Net: Deep Planar Pushing for Objects with Unknown Physical Properties.", *Robotics: Science and Systems*, Vol. 14, Pittsburgh, Pennsylvania, June 2018.

52. Xu, Z., J. Wu, A. Zeng, J. B. Tenenbaum and S. Song, "Densephysnet: Learning dense physical object representations via multi-step dynamic interactions", *arXiv preprint arXiv:1906.03853*, 2019.

53. Deng, B., K. Genova, S. Yazdani, S. Bouaziz, G. Hinton and A. Tagliasacchi, "Cvxnet: Learnable convex decomposition", *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 31–44, 2020.

54. Pashevich, A., I. Kalevatykh, I. Laptev and C. Schmid, "Learning visual policies for building 3D shape categories", *arXiv preprint arXiv:2004.07950*, 2020.

55. Rohmer, E., S. P. Singh and M. Freese, "V-REP: A versatile and scalable robot simulation framework", *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 1321–1326, IEEE, 2013.

56. Kingma, D. P. and J. Ba, "Adam: A method for stochastic optimization", *arXiv preprint arXiv:1412.6980*, 2014.

57. Yunus Seker, Ahmet Tekden and Emre Uğur, *Deep Effect Trajectory Prediction in Robot Manipulation Video*, `https://youtu.be/dFPOH1C3DeY`, accessed at May 2020.

58. Ahmet Tekden, *BRDPN Project web page*, `https://fzaero.github.io/`, accessed at May 2020.

59. Ugur, E., E. Oztop and E. Sahin, "Goal emulation and planning in perceptual space using learned affordances", *Robotics and Autonomous Systems*, Vol. 59, No. 7-8, pp. 580–595, 2011.

60. Ugur, E. and J. Piater, "Bottom-up learning of object categories, action effects and logical rules: From continuous manipulative exploration to symbolic planning", *International Conference on Robotics and Automation*, pp. 2627–2633, IEEE, 2015.

61. Reddi, S. J., S. Kale and S. Kumar, "On the convergence of adam and beyond", *arXiv preprint arXiv:1904.09237*, 2019.

62. Ahmet Tekden, *Physic Prediction Result Videos*, https://drive.google.com/open?id=148jw8e1t7QatcfgMMabZWcAAJeCUhWvQ, accessed at May 2020.

63. Ahmet Tekden, *Belief Regulation Result Videos*, https://drive.google.com/open?id=1o8SuvLiMWqK1A8UYudInVmKxZPWZGucS, accessed at May 2020.

64. Ahmet Tekden, *Mass Prediction Result Videos*, https://drive.google.com/open?id=1k7378wa8TfR8Bex5Zo_SbEyLANiQ4uMO, accessed at May 2020.

65. Ahmet Tekden, *Tool Demonstration Videos*, https://drive.google.com/drive/folders/1Isk-CLS28totu4GOGKW09zPO9AdHHjM_, accessed at May 2020.

66. Ahmet Tekden, *BRDPN 3D demonstrations in PCB lever up task*, https://drive.google.com/open?id=1f2V9w5V7h7mOEApaVdEP_8mDpWE1rwtK, accessed at May 2020.