Hierarchical Manipulation for Constructing Free Standing Structures

by

Kislay Kumar

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved October 2019 by the
Graduate Supervisory Committee:

Siddharth Srivastava, Chair
Yu Zhang
Yezhou Yang

ARIZONA STATE UNIVERSITY

December 2019

ABSTRACT

In order for a robot to solve complex tasks in real world, it needs to compute discrete, high-level strategies that can be translated into continuous movement trajectories. These problems become increasingly difficult with increasing numbers of objects and domain constraints, as well as with the increasing degrees of freedom of robotic manipulator arms.

The first part of this thesis develops and investigates new methods for addressing these problems through hierarchical task and motion planning for manipulation with a focus on autonomous construction of free-standing structures using precision-cut planks. These planks can be arranged in various orientations to design complex structures; reliably and autonomously building such structures from scratch is computationally intractable due to the long planning horizon and the infinite branching factor of possible grasps and placements that the robot could make.

An abstract representation is developed for this class of problems and show how pose generators can be used to autonomously compute feasible robot motion plans for constructing a given structure. The approach was evaluated through simulation and on a real ABB YuMi robot. Results show that hierarchical algorithms for planning can effectively overcome the computational barriers to solving such problems.

The second part of this thesis proposes a deep learning based algorithm to identify critical regions for motion planning. Further investigation is done whether these learned critical regions can be translated to learn high-level landmark actions for automated planning.

DEDICATION

This thesis is dedicated to my father. We miss you.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my thesis advisor Dr. Siddharth Srivastava for his guidance and patience throughout my thesis work. I am thankful to you for giving me opportunities to grow as a researcher. I would like to thank my thesis committee members, Dr. Yu 'Tony' Zhang and Dr. Yezhou Yang for their invaluable time.

And most importantly, I would like to thank my father, mother, sisters and friends for their constant love and support.

And lastly, I would like to thank Saurabh Animesh for guiding and supporting me throughout my masters studies.

TABLE OF CONTENTS

Page

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Chapter 1

INTRODUCTION

The new wave of Artificial Intelligence(AI) will enable robots to perform day-to-day tasks allowing humans to focus on more productive tasks. In order to perform these tasks, robots needs to manipulate objects in the real world. Combined task and motion planning allows us to do so by finding sequence of actions (high level actions) and their corresponding motion plans in order to complete a given task or reach a goal state. Automated planning finds these sequence of high level actions and guarantees complete solution i.e. the planners return a solution if there exists one. Though automated planning looks lucrative, it still needs a designer to design the domain or the rules of the universe as well a problem file written corresponding to the domain for each task. Though a manipulation task can have simple actions as Pick and Place, building structures using planks adds extra complexity for the task. This is due to multiple ways of stacking a plank and regions on the plank where another plank can be placed. As a result of this, the planning time increases exponentially with increase in number of planks in the structure. This is mitigated through abstracting regions of the planks and exploiting its symmetry. Moreover, these abstractions requires refined translation into motion plans or control sequence for controlling a robot in the real world. This thesis is targeted towards abstraction of these knowledge representation and generating low level sequence of control autonomously. We show that this abstract representation is independent of the robot and refinement of these plans are executable in simulation as well as on real robot by building generalised structures using Keva planks.

1.1 Keva Planks

Keva Planks are laser cut wooden planks used as a learning puzzle for kids. These planks holds symmetry and each of the planks are of same precise measurements which makes these planks easy to be stacked on top of each other to make plethora of 3D structures. Moreover, this allows us to model these planks easily in the simulation with precise geometry.

1.2 Task Planning

Task planning is an active area of research in the field of Artificial Intelligence which deals with finding sequence of actions to achieve a goal from a given start state. STRIPS (Fikes and Nilsson (1971)) and Planning Domain Definition Language(PDDL) (Ghallab *et al.* (1998)) are commonly used to define domains and problems. A planning problem defined as a tuple of set of finite *states* S, an *initial state* $s_0 \in S$, finite set of *actions* A, set of goal states $S_* \subseteq S$, *action cost:*$A \rightarrow \mathbb{R}_0^+$ and a *transition relation* $T \subseteq S \times A \times S$.

In this thesis we will focus on *Classical Planning* to find a plan to solve a given set of tasks. Moreover PDDL is used to define the world models for classical planning. A planning problem comprises of a domain file which describes the rules of the universe and a problem file which comprises of the objects in the universe as well as the initial state and goal state of the problem. For use with FF-Planner (Hoffmann and Nebel (2001)) we define the initial state as conjunction of literals. This assumes that grounded predicates not present in the initial state are negative literals. A domain file consists of specification of predicates whose conjunction is used to define a state, and action schemas of the set of actions applicable in the universe. An *Action Schema* comprises of parameters of the action, precondition for the action to be applied and

the effects that these action causes. These *Actions* provides change in fluents which changes the state. Every action has some *precondition* that are required to be met by the state for that action to be applied on that state. Applying an action on the state changes some of the fluents and leads to a new state. these are called *effects*. These sequence of actions taken from the initial state to reach a goal state is called a plan.

1.3 Motion Planning

Motion planning deals with low level of control of a robotic system which allows it to move along its degree of freedom. This can be considered as motion of a robotic system in continuous space of 2-dimension or 3-dimension avoiding obstacles in the environment. These motion planning algorithms generates trajectories consisting of waypoints in configuration space which is used to move a robot from an initial configuration to a goal configuration. A motion planning problem can be defined as a tuple $< C_S, f, s_I, s_G >$ where C_S is the configuration space of the robot within which it can move, f is a boolean function which returns whether the robot is in collision with itself or any other object in the environment, s_I and s_G are the start and goal configuration of the robot.

Since these motion planning algorithms has to deal with a vast continuous space, even a simple algorithm is NP-hard (Reif (1979)). Sampling based motion planners helps us maintain a probabilistic completeness of the algorithm where it guarantees to return a solution if there exists one as number of samples approaches infinity. Rapidly-exploring Random Trees (RRTs)(LaValle (1998)) and Probabilistic Roadmap (PRM) (Kavraki *et al.* (1994)) are general single query and muli-query motion planner correspondingly which are used to solve this challenging motion planning problems.

The motion planning problems becomes difficult to solve and takes a considerable amount of time with increasing degree of freedoms and obstacles in the environment.

Increasing degree of freedom requires more sampling permutations in order to solve a task. Increase in number of objects in the environment requires constant collision checking and higher number of waypoints to be generated.

1.4 Combined Task and Motion Planning

Combining these task and motion plans has always remain a challenging area of research in Artificial Intelligence. The task plans generated need to be translated into motion plans for complete solution of the task. Moreover, the task planners work independent of motion planners and are unaware of the geometry of the environment. A combined task and motion planning system targets at solving these translation and incorporating point of failure in the task planning problem to re-plan accordingly. Srivastava *et al.* (2014b) approaches this problem by introducing an interface layer to map high level actions to corresponding motion plans and translating point of failure into symbolic representations for the task planner. Shah and Srivastava (2019) extends this to stochastic environments and provides a framework for combined task and motion planning. We use this framework for our problem and extend the support of this framework to different robots, IK solvers and common bug fixing.

1.5 Manipulator Robots

Advancement in the field of robotics has produced a wide variety of robotic systems from a Roomba cleaning robot to self driving cars. These robotic system are either built for navigating in the environment or manipulating objects with an arm. Some robotic system are capable of navigating a space as well as manipulating objects in the environment. These types or robots are called mobile manipulators. We show that our system is independent of type of robot that can be used. We show construction of a various free standing structure on a single arm 7DOF mobile manipulator robot,

Figure 1.1: Fetch Robot

Fetch and a dual arm 14DOF manipulator robot ABB YuMi. Figure 1.1 shows Fetch robot and Figure 1.2 shows the ABB YuMi robot.

Fetch is a robot system build for traversing in a room and manipulating objects around it. Though this robot provides excellent freedom in navigation and manipulation, it is not built to handle small objects with precise accuracy. Thus for real world experiments, we use ABB YuMi to perform precise pick ups and placement. Since building these structures doesn't require navigation challenges, ABB YuMi proves to be an excellent choice in doing so. Our experiments in simulation are performed both on Fetch and YuMi and all the experiments on the real robot is performed on ABB YuMi.

1.6 Related Work

Blocks World has always reserved a special place in planning domains for task planning. Gupta and Nau (1992) has shown that optimal solution for blocks world and

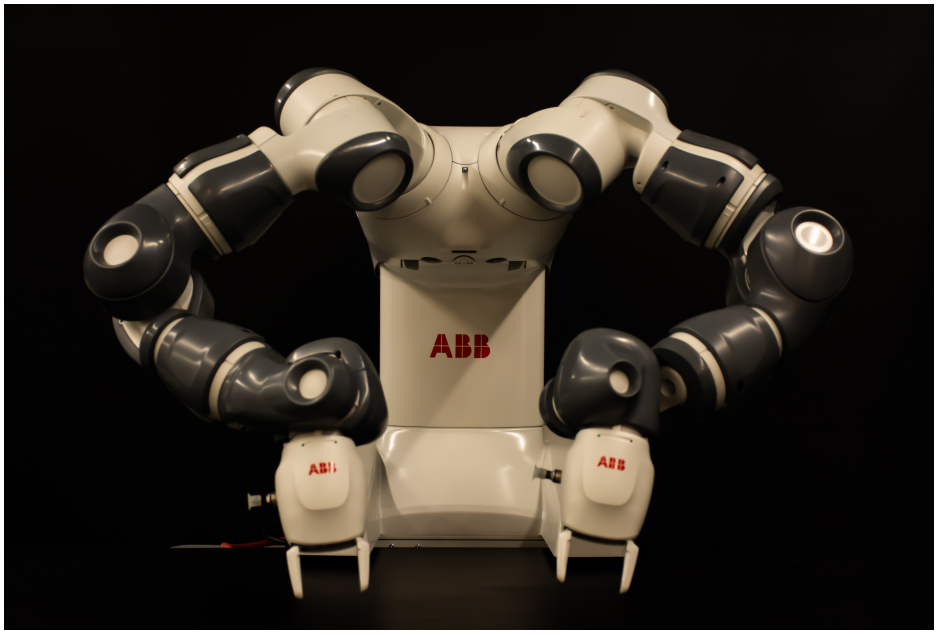


Figure 1.2: ABB YuMi Robot

related variants is NP-hard. Though this time complexities arises due to deadlocks. Fahlman (1974) presented an approach to solve the blocks word problem with complex structure. This work was dedicated towards build a system called BUILD which modelled the pyhsics, contacts and geometry of the problem. Moreover the author presented an approach to assemble blocks and wedges in desired configuration through sub-assembly of structure. Though this work presented a good approach to solve the blocks world problem in real life, it only focused towards generating high level plans to assemble these structures. Under-lying motion planning was considered to be out of scope for the paper. Our work not only provides high level plans but also the under-lying motion plans enabling a robot to perform the complete task. But our method doesn't include stability property while planning for the tasks.

Wismer *et al.* (2012) shows how a robot can build a structure based on a given plan. However, the high level plans for these constructions are pre-defined and lacks demonstration of variety of structures. Our method instead gives a complete solution

and is capable of building variety of structures. Similarly Suárez-Ruiz *et al.* (2018) presented a method to assemble IKEA chairs. Though this method provides good contact rich manipulation methods and collaborative planning for two arms in action, they lack in generating high level task plans. Instead, they use pre-defined task plans from the instruction manuals. Our method focuses on using combined task and motion planning for building structures with Keva planks.

Chapter 2

METHODOLOGY

This section of the thesis discusses the factored abstract representation of the domain for building Keva structures as well as discuses the methodology for autonomously generating target poses and motion planning for low level planning and control. We aim at building structures using combined task and motion planning framework developed at Autonomous Agents and Intelligent Robots (AAIR) lab at Arizona State University which involves my contribution in developing, testing and adding support for ABB YuMi robot, IK solvers and motion planners. More details on the framework, related paper and videos can be found at https://aair-lab.github.io/atam_full.html.

2.1 Abstract Representation

The Planning Domain Definition Language made it possible to hold the International Planning Competition (IPC) in 1998. This was aimed at creating a standardized planning language for creating new planners as well as defining problems and domains in this language. IPC introduced a variety of tasks for competing planners needed to solve a given set of tasks. Introduced in the 3rd IPC, Blocks world became one of the important domains in automated planning. Blocks world introduces a planning problem which requires to place finite set of wooden blocks of various colors or sizes to be placed in a desired configuration. These blocks can be moved one at a time and can be placed either atop table or on top of another block. Elementary Blocks World (EBW) is relaxed while variants introduces set of constraints in placement of the blocks. Gupta and Nau (1992) shows that these problems are NP-hard for finding optimal plans. IPC domain for blocks world is defined in the Figure 2.1. This shows

```
(define (domain BLOCKS)
(:requirements :strips :typing)
(:types block)
(:predicates (on ?x - block ?y - block)
             (ontable ?x - block)
             (clear ?x - block)
             (handempty)
             (holding ?x - block))
(:action pick-up
     :parameters (?x - block)
     :precondition (and (clear ?x) (ontable ?x) (
        handempty))
     :effect(and (not (ontable ?x))
           (not (clear ?x))
           (not (handempty))
           (holding ?x)))
(:action put-down
     :parameters (?x - block)
     :precondition (holding ?x)
     :effect(and (not (holding ?x))
           (clear ?x)
           (handempty)
           (ontable ?x)))
(:action stack
     :parameters (?x - block ?y - block)
     :precondition (and (holding ?x) (clear ?y))
     :effect(and (not (holding ?x))
           (not (clear ?y))
           (clear ?x)
           (handempty)
           (on ?x ?y)))
(:action unstack
     :parameters (?x - block ?y - block)
     :precondition (and (on ?x ?y) (clear ?x) (
        handempty))
     :effect
     (and (holding ?x)
           (clear ?y)
           (not (clear ?x))
           (not (handempty))
           (not (on ?x ?y)))))
```

Figure 2.1: Blocks World Domain

the predicates and action schema of the blocks world domain. Five predicates are defined to express a given state of the world as well as preconditions and effects for a given action. In the given blocks world domain, there are four actions: 1 for picking block from table, 1 for placing block on table, 1 for picking block from top of a block and 1 for placing a block atop another block. Since only one block can be placed on top of another and the orientation of the blocks is of no importance, these domains are relatively easier to solve. Moreover, blocks world has its limitation because of its simplicity, where it can't be used in any real life problems With increase in number of ways for a block that can be placed on the table and on top of a block including number of blocks and orientation, the problem becomes difficult to solve due to the involved constraints.

Our approach tries to mitigate some of these problems by abstracting the states allowing the geometric planning to refine the abstractions when required. Since a Keva plank can be placed in a infinite number of orientation or continuous value of roll, pitch and yaw from $-\pi$ to π, we abstract the orientation of a Keva plank with symbolic representation where a predicate `orientation(p,o)` defines the orientation of plank p. The predicate takes plank $p \in P$ where P is a finite set of planks, and o where $o \in \{horizontal, vertical, sideways\}$ as inputs to return True or False. This abstraction discretizes the orientation of a plank into three possible configuration. During refinement, the low level pose generators refers the orientation of the plank from reference structure which encapsulates the geometric constraint of the target structure to generate continuous orientation value for the placement of the plank.

Keva planks are designed to such measurements which allows 13 planks with thinnest side to be placed along the length of the plank, 5 planks along the medium thickness side to be placed along the length of the plank and 3 planks with thinnest side to be placed along the the medium thick side of the plank. This allows us to

divide the sides of the plank in multiple regions where a plank can be placed. Symbolic representation of such location will result in 58 such regions on the plank. With an increase in number of planks required for a structure, the number of regions will increase with an order of $\mathcal{O}(n)$. This will lead to an increased number of propositions in the state representation and an exponential increase in the search space. These regions are abstracted and planks are allowed to placed on a number of planks through number of actions involving different number of planks in the action parameter. Due to robotic gripper width and its limitation while placing a plank, only three planks can be placed on top of a plank or on the table next to each other. Thus, the proposition `region` is removed from the domain and an abstracted representation of the state is used to relax the model.

2.2 Pose Generators

An object is localised in the 3D world by representing the pose of the object as a 7 dimensional vector where 3 dimensions are used for translation of the object in x,y,z co-ordinate and 4 are used for quaternion representation. For working with OpenRave simulator, the poses of an object are expressed in the form of a 4×4 transformation matrix. The 3×3 matrix is the rotation matrix while 3×1 matrix is the translation matrix. The rest is used for padding.

We calculate the pose for the end effector based on the pose of the target object it is trying to manipulate. This is achieved through chain matrix multiplication. Let O_{OR} be the pose of the object with respect to origin and R_{OR} be the pose of the robot base with respect to origin. Then pose of the robot's gripper with respect robot base G_R is given by

$$G_R = R_{OR}^{-1} \times O_{OR} \tag{2.1}$$

Here O_{OR}, R_{OR} and G_R are matrices of size 4×4.

The calculated target pose of the end effector are susceptible to collision with obstacles. In order to mitigate backtracking in the combined task and motion planning, multiple target poses are calculated based on rotation around the plank. These poses are achieved by matrix multiplication of rotation matrix with the put down pose. Moreover, in error free mode, the inverse kinematic (IK) solutions to set the joint values of the manipulator arm are ran through a collision checker checking for self-collision and collision with the environment. Only IK solutions with no collision are generated to avoid re-planning or backtracking.

Chapter 3

IDENTIFYING CRITICAL REGIONS FOR MOTION PLANNING

This part of the thesis investigates whether methods of deep learning can be used to accelerate sampling-based motion planning. These motion planning problems are considered to be NP-Hard (Reif (1979)). Sampling-based motion planning has gained attention in planning community and till date most commonly used motion planners like RRT-Connect (Kuffner and LaValle (2000)) uses sampling to find solutions to a motion planning problem. These sampling based motion planners samples points in the configuration space of the robot and proves to be probabilistically complete. Though sampling-based motion planners shows potential in solving motion planning problems, they are generally time intensive and performs poorly in environments with tight spaces. These tight spaces have low probability of getting sampled under uniform sampling and samples around these regions are prone to rejection by collision checker. Though these regions have low probabilities of getting sampled, they still are critical to solutions for motion planning problems as most of the solution passes through these regions. We developed a deep learning based method to identify these critical regions and leverage them to find motion plans using Learn-and-Link (Molina *et al.* (2019)) suite of planners.

Given a robot R, an environment E, and a class of MP problems M, we define the *measure of criticality* of a region r, $\mu(r)$, as the ratio $\frac{f(r)}{v(r)}$, where *f(r)* is the fraction of observed MPs solving tasks from M that pass through r, and *v(r)* is the measure of r under a reference (usually uniform) probability density. Intuitively, regions with high criticality measures are those that are vital for solutions to problems in M, but have a low probability of exploration under a uniform density.

To learn critical regions, we construct a set D_{train} of N_{train} MP problem instances $\{\Pi_1, ..., \Pi_{N_{train}}\}$ and a corresponding set of solution trajectories $\{\tau_1, ..., \tau_{N_{train}}\}$ to construct the images, and a set D_{test} of N_{test} MP problem instances to evaluate the learned model.

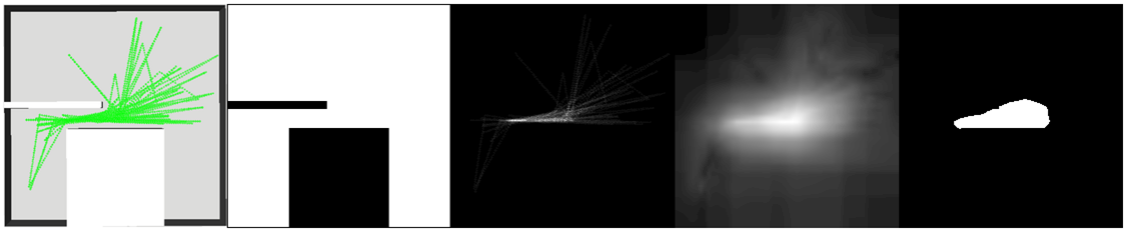
3.1 Data Generation

For each $\Pi_i \in D_{train}$, we use OMPL's RRT-Connect to generate a corresponding motion plan τ_i consisting of 50 MP problems from M.

We generate training images for each $\Pi_i \in D_{train}$ and use saliency model to generate corresponding labels. We describe the process for an *SE(2)* robot (see Figure 3.1), though it can be extended to mobile manipulators, such as the Barrett arm on a mobile base. We begin by creating a pixel-sized obstacle, based on the dimensions of the desired image and the bounds of a given environment, and scanning it across the environment. For the input images, if a collision is detected with an environment's obstacles, we select a black pixel, otherwise a white pixel is selected. For the motion trace images used by the saliency model, we assign a pixel value based on the μ-criticality of the region the pixel encompasses. We then use Itti's saliency model to extract relevant salient information and smooth out the salient areas from the motion trace images. The saliency maps are binned into two categories, high saliency (denoted by white pixels) and low saliency (denoted by black pixels), and are used as the labels.

3.2 Network Architecture

We propose a general structure for a convolutional encoder-decoder neural network which learns to detect critical regions.



(a) (b) (c) (d) (e)

Figure 3.1: (a) Example Environment Overlain with Motion Traces. (b) Model Input Obtained Post Raster Scan. (c) Saliency Model Input Based on μ-criticality of Each Pixel. (d) Saliency Map Obtained from c. (e) Input Label Obtained After Binning the Saliency Map Based on Pixel Intensity.

Our network, depicted in Figure 3.2, has 14 convolutional layers. 7 layers in the encoder network and 7 layers in the decoder network form the encoder-decoder architecture for pixel-wise classification. A pooling layer with stride 2 is introduced after each group of same number of filters to encode the learned representation. Similarly, an upsampling layer is added before each deconvolutional layer group of same number of filters. We draw inspiration from Badrinarayanan *et al.* (2015) for a learnable upsampling layer in the decoder network.

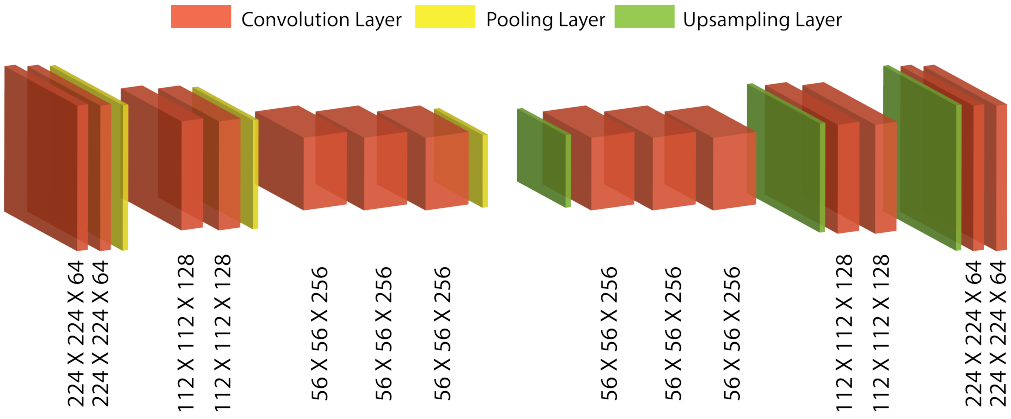


Figure 3.2: Network Architecture

The first two convolutional layers have 64 filters with a 3×3 kernel. Motivated by recent promising results by Simonyan and Zisserman (2014), we stack 3 layers with 3×3 kernel size to obtain a similar receptive field as a 7×7 kernel, with 81% less parameters, and more effective training owing to the added non-linearity after every layer. For the initial layer group of filter size 64 and 128, we stack only two layers of kernel size 3×3. Though the receptive field is smaller than a 7×7 kernel, we still stack only 2 layers as our problem statement doesn't require learning complex geometric features. The next 2 layers are of 128 filters with a 3×3 kernel. We add 3 layers of 256 filters each, with a 3×3 kernel, for a larger receptive field since deeper layers learn invariant complex features (Zeiler and Fergus (2014)).

In the decoder network, corresponding deconvolutional layers to the encoder network are used. The upsampled output is used for pixel-wise classification using a softmax cross entropy loss function. Each layer in the network is activated using ReLu non-linearity.

3.3 Training

The network was trained on a single Nvidia GTX 1080Ti using a mini-batch size of 16 and a data set of 10,024 images. Following Ioffe and Szegedy (2015), we did not train the network with dropout (Srivastava *et al.* (2014a)) since the output of every layer is batch-normalised, which also acts as a regularizer. We use Adam Optimizer (Kingma and Ba (2014)) with a 0.1 learning rate to train the network. The network was trained for 50,000 epochs since the loss converges at this point. The training images are shuffled before each epoch and trained with mini-batch to ensure that every input to the network is different from the previous. This assists the optimizer to exit local minima. We used a Github implementation of SegNet (Badrinarayanan

et al. (2015)) by `https://github.com/andreaazzini` for its data pipelines since they provide a fast and efficient input pipeline which reduces training time.

On average training for the full dataset using mini-batch takes approximately 3 hours. Single GPU training and shorter training time gives the advantage of using our method for fast motion planning.

Chapter 4

EXPERIMENTS

4.1 Keva Planks

Experiments for constructing free standing structure was performed in both simulation and on real robot. The task for the agent was to build a given structure provided as Collada model (3D model) of the target structure without human intervention. The robotic manipulator picks and places a plank at a time to build the structure. These structures were initially built using Fetch Robot in simulation. But due to thick gripper size and positional imperfection while handling small objects, we moved our experiments setting to a new manipulator robot, ABB YuMi. ABB YuMi's high precision controls helps us to build structure considering the low level environment as deterministic ones. This section is divided into two sub-sections explaining experimental setup in simulation and on real robot.

4.1.1 Simulation

The simulation environment is designed to match the real world environment. The task space around the robot is only modelled which only includes regions where the arms can reach. The environment contains 2 table, one of the table houses the robot while the other one is the playground for the robot where it builds the structures. In order to build complex structures, a large number of planks are required. The task space in front of the robot doesn't provide enough space to create a plank bank which robot uses to pick up new planks from. This problem disallows us to build structures involving large number of planks. To address this problem we introduce a

plank station where planks can be kept one at a time. Moreover we add an action `human_place_plank` in the planning domain which allows a user to place a plank at the plank station and task planner to plan accordingly. This eradicates the problem of plank storage as well as makes it easier to write the initial state of the universe for task planner.

We build a number of structures in the simulation to show the generalizability of the method to build different types of structures. Figure 4.1 shows the target structures which was required to be built. A high level problem file is written which expresses the goal configuration as the placement of planks on top of each other or table. Also, a collada file of the target structure is specified which encodes the relative poses of the plank to each other in a 3D model. This allows the system to respect the geometric constraints of the structure. Various off the self libraries are used to generate and refine the plans. FF planner is used for finding plans to build the structure. Underlying geometric environment uses OMPL's RRT-Connect and BIT* (Gammell *et al.* (2015)) for motion planning, Prximity Query Package (PQP) (Larsen *et al.* (1999)) and Flexible Collision Library (FCL) (Pan *et al.* (2012)) for collsion checking, OpenRave (Diankov and Kuffner (2008)) for simulation and Trac-IK (Beeson and Ames (2015)) for inverse kinematics (IK) solutions.

4.1.2 Real Robot

The simulation produces motion planning trajectories which can be executed in simulation as well as on real robot. These generated trajectories are timed joint state and velocity which can be used with position controllers as well as velocity controllers. We use Yumipy, an open source control interface for ABB YuMi robot, to control the movement of arms in real time on the robot. The yumipy implementation can be found at https://github.com/BerkeleyAutomation/yumipy. Though yumipy

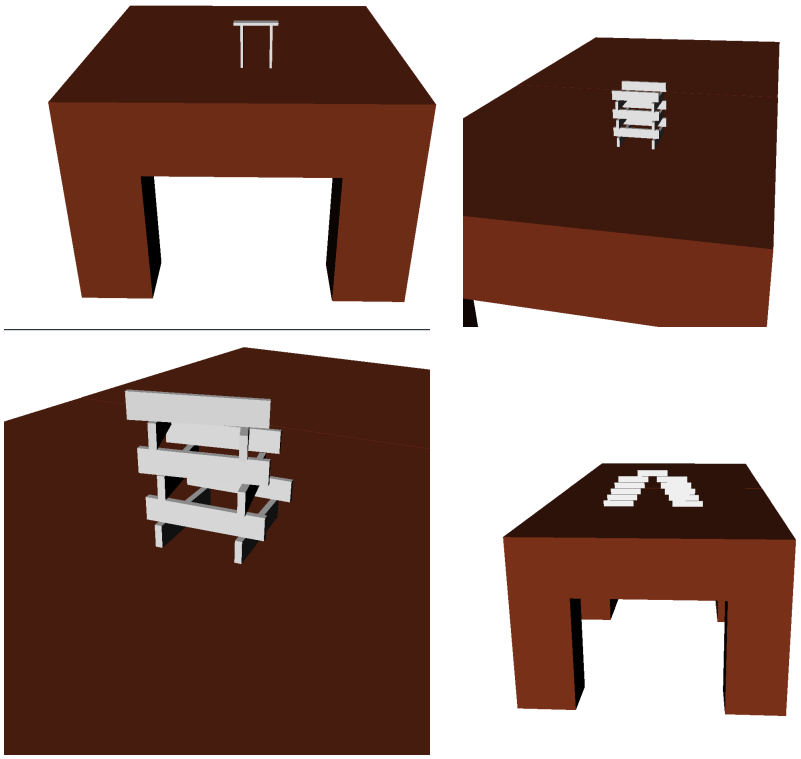


Figure 4.1: Target Keva Structures(Sim)

provides easy interface with the robot, it only supports position control as of last update. We use waypoints in the generated motion plan trajectories and throw away the velocity, time and acceleration information to just execute position control as these waypoints are the joint states. The interpolation between joint states happens inside the ABB YuMi controller. This shows flexibility of the framework to work with different motion planners. Figure 4.2 shows the target structures which was built using the real robot.

The execution on real robot is one of the most challenging task in robotics. These are prone to error in state estimation, collisions due to interpolation between the waypoints. Moreover precise placement requires precise movement of the arms while

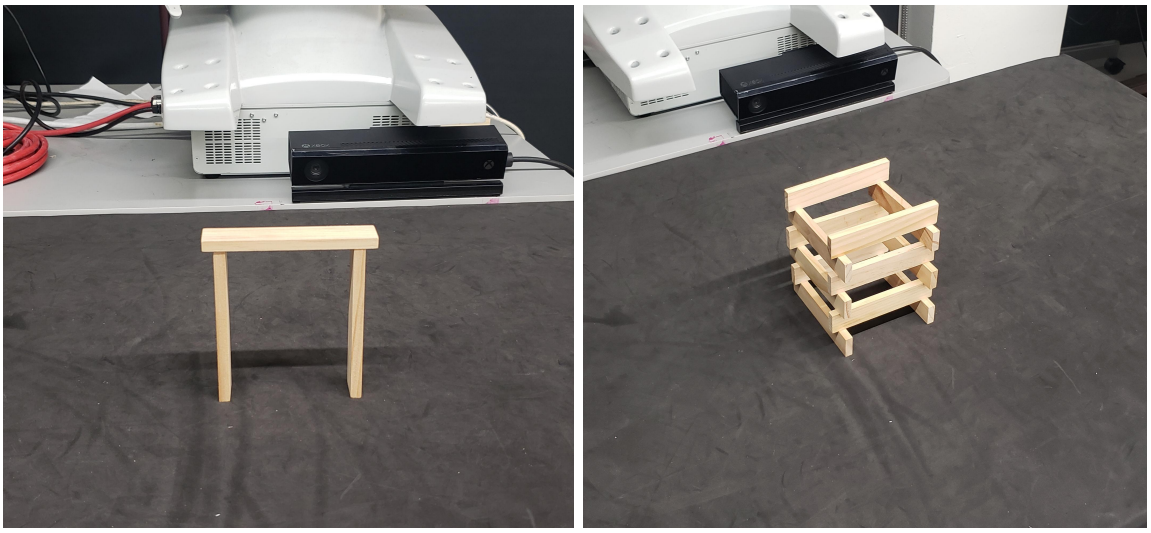


Figure 4.2: Target Keva Structures(Real)

picking and placing a plank. To mitigate this issue, pre and post grasp/putdown conditions are added which moves the arm few centimeters away from the actual pose and then move slowly to perform grasp or putdown. This reduces the collision between gripper and planks before and after picking and placing a plank. These trajectories from moving to a pre grasp to grasp requires the end-effector to move in a straight line. When these trajectories are computed in the simulation, it doesn't guarantee a linear path as the planning is done in the joint space and not in Cartesian space. Cartesian space planning together with velocity control allows the end effector to move in a straight line. The yumipy library only supports position control and not velocity control for the arm, though it supports moving the arm linearly for small change in translation using YuMi's inbuilt planner. Thus we use YuMi's inbuit planner to compute these linear trajectories as the Task and Motion Planning framework is planner agnostic and allows us to use hybrid-planning system.

The simulation matches the setting of real world environment which includes YuMi sitting atop a table and a table in front of the robot used as the task space.

Chapter 5

RESULTS

5.1 Keva Planks

The experiments were performed for 20 different runs for Tower, Pi and Spiral Tower structure. The experiments for the Bridge structure was performed for 10 runs. Table 5.1 shows the number of tasks completed for a given structure. These tasks were given a maximum time limit of 3000 seconds for planning and refinement. Figure 5.1 shows the construction steps for a pi structure.

Tasks	Tower	Pi	Spiral Tower	Bridge
Tasks Completed	19/20	20/20	20/20	10/10

Table 5.1: Task Completion Rate

Figure 5.2 shows the distribution of time taken for planning and refinement of each structure. The corresponding time logs for the experiments as shown in Table 5.1 includes total time taken to plan, refine those plans and generate motion trajectories.

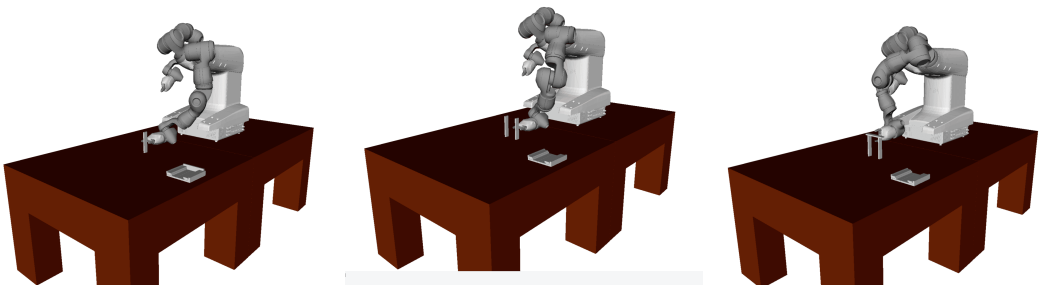


Figure 5.1: Construction Steps of a Pi Structure

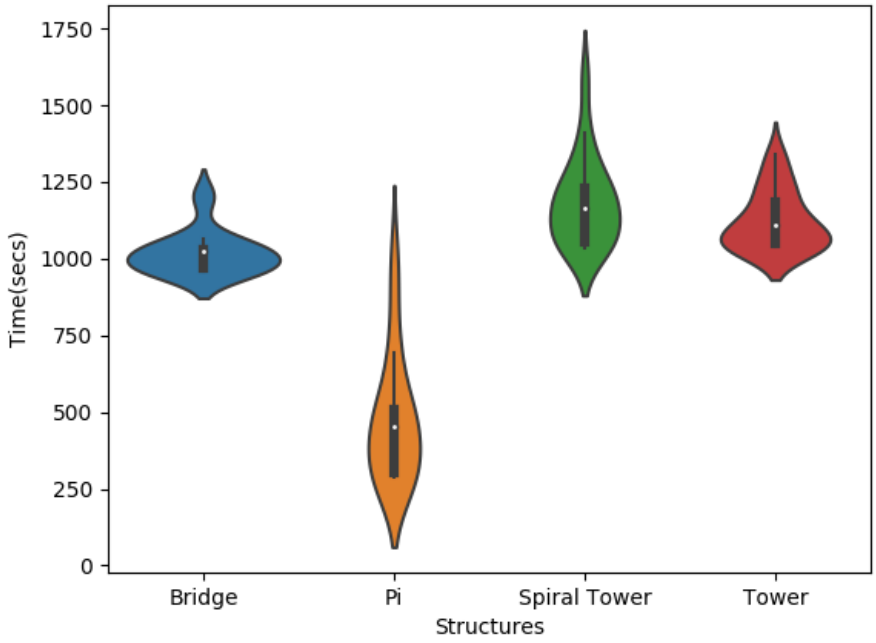


Figure 5.2: Planning and Refinement Time Distribution for Different Structures

Execution Run	1	2	3	4	5
Task Completion	0	0	0	1	0
Failure Mode	Unreachable Pose	Unreachable Pose	Unreachable Pose	-	Obstacle Collision

Table 5.2: Task Execution on Real YuMI

Table 5.2 shows execution success/failure(0/1) on real robot for Tower structure. For this experiment, 5 different runs of Tower structure were executed in simulation, each of these runs generating motion plans. These plans were further executed on the real robot. As discussed in Section 4.1.2, the motion plans for moving from post-grasp to pre-put down and post-put down to pre-grasp were computed using OpenRave motion planner whereas the motion plans for generating linear trajectories from pre-

grasp to grasp, grasp to post-grasp, pre-put down to put down, and put down to post-put down were computed using the YuMi's inbuilt motion planner. Three of these refined plans failed during execution on the real robot while computing the linear trajectories using YuMi's motion planner. These linear trajectory computations failed due to unreachable target pose of the end-effector for the linear movement. Furthermore, one plan led to collision of the gripper with the plank that was already placed. Only one out of 5 refined plan was successfully executed using the hybrid motion planning system. The plan leading to the successful execution on the robot was repeated 5 times. These repeated runs used the same refined plan which was computed in the simulation but computed the motion plans during execution. Table 5.3 shows the success rate of 5 runs for the repeated execution of plan which led to the successful execution in Table 5.2.

	Tower Structure
Tasks Completed	5/5

Table 5.3: Task Completion for Repeated Run of Successfully Executed Plan

5.2 Evaluating Identified Critical Regions

We evaluate the critical regions identified by our models using the ground truth motion trace image for an environment. We first cluster the model-identified critical regions using k-Nearest Neighbors (Altman (1992)). Then we evaluate each critical region cluster using the μ-criticality of the cluster, where we estimate $v(r)$ as the area of the pixels in the cluster. The metric values for each cluster are then summed to obtain an evaluation of the environment as a whole. The higher the value, the better the critical regions.

We use this metric instead of comparing pixel accuracy with the ground truth label since the motion trace image is embedded with much more information regarding the quality of the critical regions than solely being able to locate them.

Figure 5.3 shows a comparison of the critical regions identified by VGGNet, SegNet, and our parsimonious network using this metric.

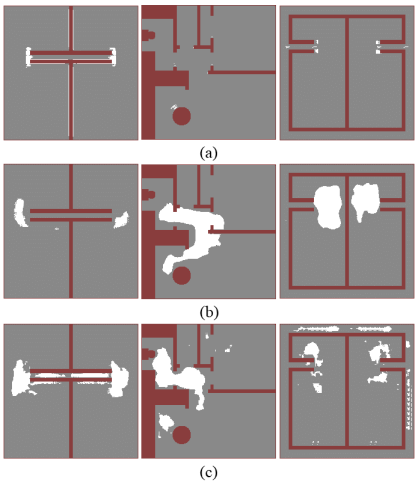


Figure 5.3: (a) Critical Regions Identified Using VGGNet. From Left to Right, μ-criticality is 0, 0, 0. (b) Critical Regions Identified Using SegNet. From Left to Right, μ-criticality is 0, 0.141, and 0.260. (c) Critical Regions Identified Using our Network. From Left to Right, μ-criticality is 0.604, 0.371, and 0.702.

Chapter 6

CONCLUSION

In this thesis we present a hierarchical approach in manipulation to overcome the intractability of these class of problems. Results shows how these algorithms are complete and able to cope up with the infinite branching in grasp and put down placements. The algorithms allows a robot to manipulate wooden planks to arrange in order which meets the goal state. The high level task planner generates sequence of actions i.e. the order in which these planks should be placed. The abstraction in the state representation allows to relax the problem and deal with increased number of literals in the state space. This accelerates the planning and are finds a solution in realizable time. The low level pose generators autonomously generates poses for a given action to manipulate the planks. The hierarchical approach here allows to generate solutions overcoming the infinite possible continuous values for these solutions.

The second part of this thesis aims at bridging the gap between planning and learning and tries to get best of the both worlds. An attempt was made in learning the critical region for motion planning which are regions having low probability of getting sampled under uniform sampling but most of the solutions pass through them. Results show hoe these learned regions accelerate motion planning and beats any state of the art motion planning algorithms in time required to solve a given problem. The results gives us good insights on how this can be used to learn landmark actions for task planning.

Chapter 7

FUTURE WORK

The framework for combined task and motion planning requires domain knowledge and user input in the form of pose generators. This work can be extended to reduce human interventions as much as possible to give complete autonomy. A pose generator can be learned for a class of problems which automatically generated poses for different actions based on the current and future actions. Moreover as investigated in Section 2.4.4, this work can be used to learn the landmarks actions for task planning to reduce the planning time.

The motion planning for a manipulator arm exhausts most of the available time for the whole planning problem. These motion planning algorithms contributes most to slow down the refinements. Learn and Link can thus be extended to high dimensional robots to accelerate the motion planning.

The methods discussed in this thesis is a step towards solving real world application of hierarchical manipulation. IKEA chair assembly is often considered as the moon landing for robotic automation. This thesis inches closer to have a complete autonomous agent which can generate plans to assemble IKEA chairs without human intervention.

REFERENCES

Altman, N. S., "An introduction to kernel and nearest-neighbor nonparametric regression", The American Statistician **46**, 3, 175–185 (1992).

Badrinarayanan, V., A. Kendall and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation", arXiv preprint arXiv:1511.00561 (2015).

Beeson, P. and B. Ames, "Trac-ik: An open-source library for improved solving of generic inverse kinematics", in "2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)", pp. 928–935 (IEEE, 2015).

Diankov, R. and J. Kuffner, "Openrave: A planning architecture for autonomous robotics", Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34 **79** (2008).

Fahlman, S. E., "A planning system for robot construction tasks", Artificial intelligence **5**, 1, 1–49 (1974).

Fikes, R. E. and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving", Artificial intelligence **2**, 3-4, 189–208 (1971).

Gammell, J. D., S. S. Srinivasa and T. D. Barfoot, "Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs", in "2015 IEEE International Conference on Robotics and Automation (ICRA)", pp. 3067–3074 (IEEE, 2015).

Ghallab, M., A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld and D. Wilkins, "Pddl—the planning domain definition language", AIPS-98 planning committee **3**, 14 (1998).

Gupta, N. and D. S. Nau, "On the complexity of blocks-world planning", Artificial Intelligence **56**, 2-3, 223–254 (1992).

Hoffmann, J. and B. Nebel, "The FF planning system: Fast plan generation through heuristic search", Journal of Artificial Intelligence Research **14**, 253–302 (2001).

Ioffe, S. and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", arXiv preprint arXiv:1502.03167 (2015).

Kavraki, L., P. Svestka and M. H. Overmars, *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*, vol. 1994 (Unknown Publisher, 1994).

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

Kuffner, J. J. and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning", in "Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)", vol. 2, pp. 995–1001 (IEEE, 2000).

Larsen, E., S. Gottschalk, M. C. Lin and D. Manocha, "Fast proximity queries with swept sphere volumes", Tech. rep., Technical Report TR99-018, Department of Computer Science, University of ... (1999).

LaValle, S. M., "Rapidly-exploring random trees: A new tool for path planning", (1998).

Molina, D., K. Kumar and S. Srivastava, "Identifying critical regions for motion planning using auto-generated saliency labels with convolutional neural networks", arXiv preprint arXiv:1903.03258 (2019).

Pan, J., S. Chitta and D. Manocha, "Fcl: A general purpose library for collision and proximity queries", in "2012 IEEE International Conference on Robotics and Automation", pp. 3859–3866 (IEEE, 2012).

Reif, J. H., "Complexity of the mover's problem and generalizations", in "20th Annual Symposium on Foundations of Computer Science (sfcs 1979)", pp. 421–427 (IEEE, 1979).

Shah, N. and S. Srivastava, "Anytime integrated task and motion policies for stochastic environments", arXiv preprint arXiv:1904.13006 (2019).

Simonyan, K. and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", arXiv preprint arXiv:1409.1556 (2014).

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting", The Journal of Machine Learning Research **15**, 1, 1929–1958 (2014a).

Srivastava, S., E. Fang, L. Riano, R. Chitnis, S. Russell and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer", in "2014 IEEE international conference on robotics and automation (ICRA)", pp. 639–646 (IEEE, 2014b).

Suárez-Ruiz, F., X. Zhou and Q.-C. Pham, "Can robots assemble an ikea chair?", Science Robotics **3**, 17, eaat6385 (2018).

Wismer, S., G. Hitz, M. Bonani, A. Gribovskiy and S. Magnenat, "Autonomous construction of a roofed structure: Synthesizing planning and stigmergy on a mobile robot", in "2012 IEEE/RSJ International Conference on Intelligent Robots and Systems", pp. 5436–5437 (IEEE, 2012).

Zeiler, M. D. and R. Fergus, "Visualizing and understanding convolutional networks", in "European conference on computer vision", pp. 818–833 (Springer, 2014).

APPENDIX A

NOTES ON CHAPTER 3

Please note that the chapter 3 of this document contains parts taken from the paper Molina *et al.* (2019) which was uploaded on arxiv and presented at ICAPS Workshop on Planning and Robotics 2019. This paper was co-authored with Daniel Molina and Siddharth Srivastava and co-authors approves of the inclusion of the above-stated work in this thesis.