Observational Learning of Rules of Games

A Thesis

submitted in partial fulfilment of the

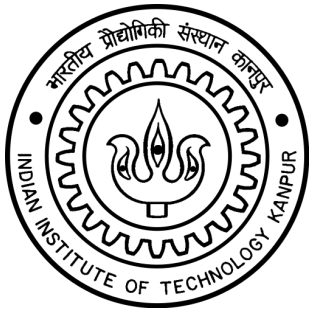
requirements for the Degree of

Master of Technology

by

Debidatta Dwibedi

Y9227188



to the

Department of Electrical Engineering

Indian Institute of Technology, Kanpur

May 2014

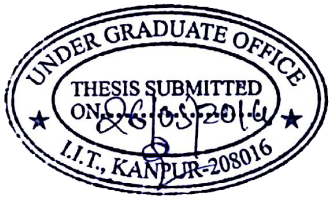
CERTIFICATE

It is certified that the work contained in this thesis entitled "**Observational Learning of Rules of Games**", by **Debidatta Dwibedi (Roll No. Y9227188)**, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

(Prof. Amitabha Mukerjee)
Department of Computer Science and Engineering,
Indian Institute of Technology Kanpur
Kanpur-208016

(Prof. K S Venkatesh)
Department of Electrical Engineering,
Indian Institute of Technology Kanpur
Kanpur-208016

26 May, 2014



TO

my parents and sister

for their love and support.

Abstract

People can learn to play games by observing others play. In the past, systems that attempted to discover game rules have used considerable background knowledge (e.g. game states, goals). In this work we use depth sensing technologies (RGBD), together with unsupervised transitions via Semantic Graphs to discover states and rules for games.

The system is based on our own multiple-object tracking method based on an octree overlap metric. It uses the Hungarian algorithm for the assignment problem to assign object labels from one frame to the next. Each frame is represented as a Semantic Graph whose nodes represent the objects and the edges encode spatial relationships. The changes in the structure of these graphs reveal typical moves in the game. We perform automatic discovery of board states by clustering game piece locations. Knowledge from these states and the semantic graphs is mapped to First-Order Logic descriptions of the states and transitions. Based on this, an Inductive Logic Programming (ILP) system is able to infer the valid moves and other rules of the game. Starting with no game-specific knowledge, induced rules are demonstrated for Towers of Hanoi (e.g. higher pegs must be smaller) and 1D Peg Solitaire (moves may be 1 square or two).

Acknowledgements

I would like to take this opportunity to thank the people who have helped me during the course of working on this thesis. Firstly, I would like to thank both my supervisors, Prof. Mukerjee and Prof. Venkatesh, for their support during the last couple of years. I was motivated a lot to pursue both this work and research because of their efforts and guidance. I sincerely appreciate the academic freedom that I have enjoyed at IIT Kanpur because of which I was able to pursue my diverse research interests.

The developers of PCL have produced a very efficient and well-documented library which has been used extensively in this work. Many a time I have been in a fix during the course of writing this thesis and found help from the online computer vision community, for which I am immensely grateful.

I would like to thank all my friends at IIT Kanpur for such an amazing stay here. I also thank my family and close friends for their immense love and constant support of my ambitions.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Games and AI

From its inception, the field of artificial intelligence has found great challenge in solving puzzles and playing games. It seemed to be a natural course of action to build game playing systems whose intelligence could be assessed by simply competing with humans. In the beginning, researchers in AI looked at discrete-state games and puzzles because they are easy to model and analyze, and their environment is fully observable, deterministic and discrete. This can be seen in contrast with the more difficult problems being solved in AI today like driving an autonomous car in which the environment is partially observable, stochastic and continuous. Hence, the real challenge for game playing agents was to defeat their human counterparts. In recent times, AI agents like Deep Blue(Chess) and Watson(Jeopardy!) have managed to successfully outperform humans. However, the General Game Playing problem, which involves the design of a single system that is able to play more than one game successfully, is still an active area of research. A General Game Player(GGP)[15] is able to take formal descriptions of a game and play it without human intervention. We will be attempting to build a prototype of a system that is able to model different games through visual observation. While the objective of a GGP is to be able to play the game well, our aim will be to be able to represent different kinds of games using the same system.

1.2 Learning of Game Rules

Some systems exist which attempt to learn from visual observation. Hazarika and Bhowmick[18] learn rules of card games from videos. They extract visual features by detecting blobs, edges and objects of interest which are used to classify the cards. They use incremental learning on a decision tree to learn the rules of the card game. Terminal states of win, lose or tie need to be fed also. A decision tree is a good way to look at card games because usually at the end of every move there is a comparison of the cards and a decision is taken. For spatial reconfiguration puzzles we are looking at, decision trees are not as effective. Barbu et al.[6] learn the rules of grid-based two player games and also transfer the gameplay to robot which is able to manipulate the pieces onto a wooden frame. They use PROGOL to learn valid moves of the game pieces and winning conditions in six games. While ILP is good at handling complexity, it needs to be provided with a lot of background knowledge to be able to give good results. Another approach that has been taken up for game rule learning uses descriptive complexity. Kaiser[19] presents a system which requires only a few demonstrations and minimal background knowledge to learn the rules. They compute formulas defining allowed moves and final positions in a game in different logics and select the most adequate ones. Their system is also capable of coming up with evaluation functions which can be used to later play the game competitively. Their game rule learning system has successfully learnt the rules of more complex games like Gomoku and Connect4. It generalizes what a winning state is from multiple demonstrations. Illegal moves need to be explicitly stated to their system.

A technique used to model transition from one state to the next is by transforming non-deterministic finite automata(NFA) from visual observations of people playing to deterministic finite automata(DFA). This method has been used to learn rules of simplified board games [7] and transfer assembly steps from humans to robots [10]. Bjornsson[7] looks at games played on rectangular boards made of squares. Preliminary knowledge that is provided to the system includes concepts like players that take turns and a terminal position which happens if a player reaches the goal state or can make no further legal moves which implies that the player has lost. Movement patterns of game pieces

are provided directly in terms of their coordinates on the board. Their system aims to come up with a DFA representation of the legal moves from the NFA of observations. Each move is represented in terms of change in x and y coordinates of a game piece. Rules learnt about the movement patterns of game pieces can be given as input in Game Description Language(GDL) to a General Game Playing(GGP) system that comes up with good strategies to play the game, not just correctly. We however decided to work with logical descriptions of state changes to try and learn the rules.

1.3 Objective

Our primary objective in this work is to build a system which takes a video of people playing a game as input and is able to learn its rules.

But understanding games by observation involves solving more than one problem. The visual system needs to segment objects(i.e. the game pieces, boards and players) from the scene, be able to track them robustly, use natural language processing or background knowledge to reason about the attributes of the pieces and the moves in the games, represent game states and plan for a winning strategy. All of the above are areas of active research, which makes their merging to learn rules of puzzles a challenging problem.

There are two main contributions by this thesis. First we present a object monitoring framework to build dynamic semantic graphs from point clouds. We also provide an unsupervised method to discover games states and valid game-piece positions. The other important contribution is the method proposed to represent game states of any game. Using that and changes in semantic graph structure of the scene, logical clauses are generated. These are used with inductive logic programming to learn the rules of puzzles and games. Finally test our system on two spatial reconfiguration games(Fig. 1.1), the Towers of Hanoi and the 1D Peg Solitaire, whose rules are very different from each other.

(a) The Towers of Hanoi (b) 1D Peg Solitaire

Figure 1.1: Games whose rules we attempt to learn.

1.4 Proposed Approach

1.4.1 Learning through Visual Observation

Roboticists and cognitive scientists are actively pursuing the idea of machines that learn by directly observing humans. This is different from supervised learning where the data is labeled by humans. Observational learning on the other hand involves looking at humans perform tasks. Availability of cheap and portable cameras and cheap storage has resulted in people generating tremendous amount of video data. New sensors for 3D processing like the Kinect have become popular and even portable. Effective systems now exist for the detection and tracking of humans from both videos and point cloud streams. As a result, machines learning by observing humans is becoming all the more popular.

Some of the diverse areas where observational learning is being applied in are semantic segmentation of scene[12], affordance modeling of objects[22] and manipulation planning[10]. Delaitre et al.[12] collected lots of time-lapse videos from YouTube of hours-lasting events like parties and house cleaning. Their system tracks the skeletons of humans interacting with different objects in the scene like the sofa, bed, cupboard etc. These categories are associated with both the human poses co-occurring in the vicinity of these objects as well as their appearance. These models significantly improved semantic segmentation of the scene by incorporating the functional properties of an object in terms of how humans interacted with it. Koppula et al.[22] aimed to both recognize object affordances and representing human activities in terms of pre-defined smaller subactivities by observing people in RGBD videos. They use a Markov random field where

the nodes represent objects and sub-activities, and the edges encode the relationships between object affordances, their relations with sub-activities, and their evolution over time. Dantam et al.[10] demonstrate how an assembly task performed by humans can be transferred to a robot. Their system looks at RGBD videos of a person connecting bars with screws to build an assembly. They build a graph of the assembly as it is being built. A linguistic representation is generated from these graphs. This is used by the robot to come up with a policy to create the same assembly on its own. More recently, tracking of hand from 3D videos has become robust[27]. This will provide great impetus to observing humans as now the visual system can not only look at the actions of humans in terms of changes in the joint angles of their tracked skeletons but also how they manipulate objects using their fingers. We too explore this new paradigm of learning to make inferences about the rules of different games by looking at how people manipulate the game pieces during gameplay. We look at manipulation recognition techniques to be able to identify frames which are game states out of all the frames in the game video. In other words, we don't look to learn the rules from the part of the video where the hand is manipulating the game pieces.

1.4.2 Representing Scenes with Semantic Graphs

One line of work that is of relevance is the representation of videos as dynamic graphs with nodes representing objects and edges encoding some semantic relationship between the nodes like contact. The advantage of using graphs is that it allows one to represent complex scenes in terms of their essence - the objects and their relationships with each other - just the way humans do. This method of generating graphs to represent scenes has already found application in recognition of manipulation actions[5][37] in terms of their primitives like merging and dividing and using that further to classify higher-order actions like making a sandwich, cutting a cucumber etc. We expect that these changes in graph structures and the properties of the nodes of these semantic graphs would help us to learn the rules being followed while a game is being played.

People can learn a lot about the rules of a game by observing other people play. In

spite of that, certain rules might still have to be explicitly stated to them to play that game. If robots can pick up rules the same way people do, it will provide great impetus to social robots that can interact with humans in a more natural manner. Agents with physical embodiment are more suited to teach children puzzles like the Tower of Hanoi as compared to animated agents[36]. Research is under way to train robots as co-therapists to help out children with autism[33] and robots that can understand rules of game will help them in making better mental models and understand social cues too. Robots like the Nao have proven to be more engaging because they do not overstimulate or overwhelm a child with autism. More people can teach robots by demonstration instead of programming.

1.4.3 Induction Logic Programming

Inductive reasoning is a method of generalization in which one seeks to derive general principles from detailed facts. In this kind of reasoning the premises seek to supply strong evidence for(not the absolute proof of) the truth of the conclusion. While the conclusion of a deductive argument is supposed to be certain, the truth of an inductive argument is supposed to be probable, based upon the evidence given[9]. This is the basis of many scientific theories like Darwinism and Big bang theory. People observe the moves being performed and try to come up with a hypothesis about the rule that might hold for all the cases they see. Inductive logic programming(ILP) is a method in AI which comes up with hypotheses given some positive examples and background knowledge. It has been used successfully to learn grammatical rules in natural language processing and elicit hidden knowledge from biological data. It is well-equipped to handle a lot of complexity in relational data which might not have a corresponding propositional representation. It is really powerful if domain specific structural information is provided as background knowledge. Srinivasan et al.[35] compared the performance of ILP systems in predicting the mutagenic activity of small molecules with those found out by regression, neural and tree-based methods. They showed that by providing the ILP system with structural information about the molecules from the domain knowledge of organic chemistry like

the definitions of bonds, rings, hetero-aromaticity etc. it was able to outperform the above mentioned feature-based algorithms.

1.4.4 Using ILP to Learn Rules of Games

One can formulate the problem of people learning rules of games by observation using inductive reasoning. ILP has been used to learns rules of boardgames like Tic-Tac-Toe and Hexapawn[6] and dice-based games[32] from visual observation. In both cases, it was able to successfully generalize from the demonstrations that were presented to it and induce the rules. Barbu et al.[6] described a system that learns to play games like tic-tac-toe, hexapawn given background knowledge of many concepts like board representation, linearity tests, frame axioms, turns and opponents, piece ownership and spatial predicates. The visual pipeline in their work was specific to learning to represent game states from grid-based games that their robot could play . They represented game states in form of clauses and use PROGOL to perform inductive logic programming to learn the initial and final game states and all possible legal moves. But there are many games especially spatial reconfiguration puzzles which do not require an explicit grid to be played. Our visual system had been successfully able to discover the game states. Using them, we aim to find the valid positions of the game pieces by clustering the positions they occupy during the game states. Without any board fitting algorithm, we attempt to come up with a general representation for any game that will help us induce its rules. Using these representations, we transfer the information contained in these semantic graphs to logical clauses in Prolog. We also try to minimize the background knowledge our system might need to come up with the rules.

We too will be using ILP to do the same. However, we intend to reduce the background knowledge required by the system. Our visual system which uses semantic graphs to discover game states is extended to find the clusters occupied by the game pieces without performing any grid fitting or using game specific assumptions. We are able to learn the rules of different kinds of games using the same system.

1.4.5 From Semantic Graphs to Logical Clauses

One important contribution of this work is the method of transfer of knowledge from the visual system to the rule learning system. We propose a method for representation of game states of different games. We discover the valid positions any game piece can take up during game play. Each of the game pieces gets assigned to one of the valid positions. Changes in this position are used to describe logical clauses like move. Outputs of classifiers for visual attributes like colour and size are also converted to clauses. These are provided as background knowledge and positive examples to the ILP system which is able to generalise from the observations to come up with the rules of the game.

1.5 Organization

The rest of this thesis is organized as follows. In the remaining part of this chapter we give an introduction to point clouds and the Point Cloud Library which we have used extensively. We also briefly describe the software BlenSor which has been used to create some of the datasets we used to initially test our algorithms. That is followed by a primer on inductive logic programming. We will be using ILP later on to learn the rules of puzzles. In chapter 2, we outline our framework which build semantic graphs by segmenting objects and tracking them. In Chapter 3 we present methods of generating clauses in Prolog from the semantic graphs. These are used to learn the rules of puzzles. We outline a method of general representation of game states for different games by performing unsupervised discovery of the valid positions a game piece can take. Finally in Chapter 4, we describe our experiments on two different puzzles: the Towers of Hanoi and the one-dimensional peg solitaire. We present our conclusions and possible directions for future work in Chapter 5.

1.6 Point Cloud Processing

A point cloud is a collection of data points in a coordinate system. In a three dimensional coordinate system, these points are usually defined by the tuple (X, Y, Z) which represents the coordinates of a point, usually present on the external surface of an object.

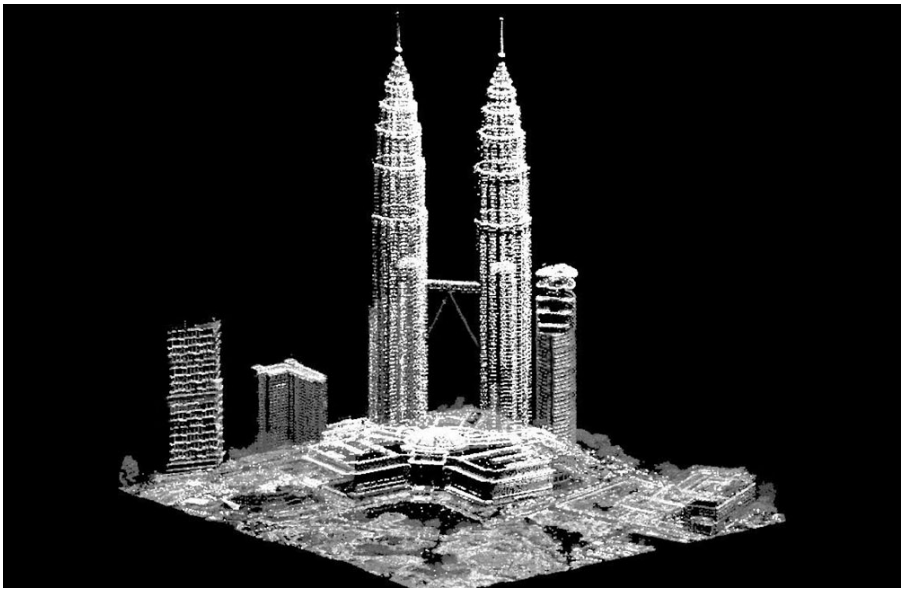


Figure 1.2: LiDaR generated Point Cloud of Kuala Lumpur City Centre, Malaysia. Taken from http://www.lidar.com

Initially, point clouds(Fig. 1.2) needed expensive 3D scanners such as LiDaR(Light Detection and Ranging) systems. But nowadays, they can also be created from RGBD images in real-time like the ones from the Microsoft Kinect or Asus Xtion(both are based on technology from PrimeSense)(Fig. 1.3). More modern devices like the Occiptal Structure Sensor can be attached to mobile devices making them portable. These can build point clouds of objects and even entire rooms at one go. Point clouds can also be built from 3d reconstruction by using lots of images of the same object taken from different viewpoints. Computationally, it is a very expensive process but it does not require an extra depth or ranging sensor. As the devices used to create these clouds get more ubiquitous, so do their applications.

One such area of application is that of robotics. If robots have to interact with humans they need to share a common sensory ground. The most useful sense in understanding

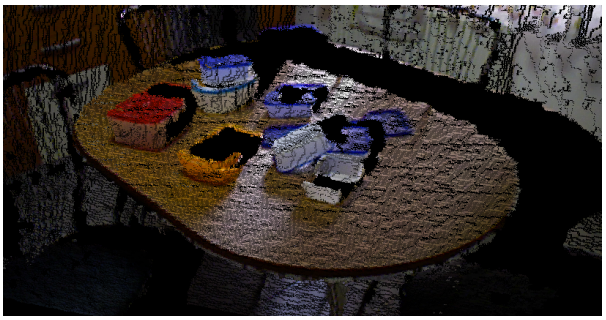


Figure 1.3: Point Cloud from a Kinect. Taken from http://pointclouds.org/

the world around us is that of vision. Ambiguities about the depth of objects often arise in image processing. But with a Kinect we get depth information from the points directly. This helps a lot in segmentation of objects and their tracking. Hence, we decided to use Kinect generated point clouds to conduct our experiments.

Since it was introduced a couple of years ago, Point Cloud Library(PCL)[31] has revolutionized point cloud processing by providing an open-source C++ library for efficient 3d processing. The PCL community has grown considerably and provides support on many issues on its forums. Currently, it boasts of many essential algorithms like filters, keypoint extraction, 3D feature descriptors, registration, segmentation, kd-trees and visualization of point clouds. We use PCL extensively in the implementation of our semantic graph framework.

1.6.1 BlenSor

BlenSor[17] is a modified version of the 3D graphics and animation software Blender which is used for simulation of Light Detection and Ranging (LIDAR/LADAR) and Kinect sensors. We have used BlenSor to create two datasets for our experiments: point clouds sequences of 4 block Towers of Hanoi and one-dimensional Peg Solitaire being played. The main reason we used BlenSor was that it would provide us with a quantitative estimate of how effective our algorithm was in an ideal setting. There is no hand/human that is manipulating the game pieces which allowed us to test and tweak our algorithm in a best case scenario.

1.7 Inductive Logic Programming

Inductive logic programming(ILP) is a subfield of machine learning which uses logic clauses as a uniform representation for examples, background knowledge and hypotheses[26]. The system is provided positive examples and background knowledge as clauses in Prolog. This can be thought of as a database of facts and observations from which an ILP system like ALEPH[34] or PROGOL[25] will be able to generate hypotheses that entails all the positive examples and none of the negative examples.

$$positive\ examples + negative\ examples + background\ knowledge \rightarrow hypothesis$$

PROGOL[26] is an implementation of an ILP system. It performs branch and bound top-down search to come up with hypotheses that cover as many positive examples as possible and as few as negative examples as possible. Outline of the PROGOL algorithm:

1. From a subset of positive examples, constructs the most specific rule r_s.
2. Based on r_s, it finds a generalized form r_g of r_s so that $score(r_g)$ has the highest value among all candidates.
3. Removes all positive examples that are covered by r_g.
4. Repeat from step 1 if there are still positive examples that are not yet covered.

In step 2, the term $score(r)$ is mentioned which is a measure of how well a rule r explains all the examples with preference given to shorter rules.
p_r = number of positive examples correctly deducible from r
n_r = number of negative examples correctly deducible from r
c_r = number of body literals in rule r
$score(r) = p_r - n_r - c_r$

The term c_r is the equivalent of regularization constant used in machine learning to prevent overfitting of the rule learnt to the dataset. Essentially, the *score* measure looks for the rule that can explain the positive and negative examples in terms of the simplest rule possible, simplicity of a rule being defined as the number of literals present in it.

1.7.1 The Family Example

To illustrate ILP more clearly, we use the example described in Wikipedia. They can be used to represent facts like $cat(tom)$. The above represents the fact *tom* is a *cat* is true. They can also be used to represent rules like `sibling(X,Y):-parent(Z,X),parent(Z,Y).` which is read as X and Y will have the *sibling* relation true if they have both the *parent* relation with the same Z is true.

As an example, we state the relationships in a family as background knowledge:

```
parent(h,m).
parent(h,t).
parent(g,m).
parent(t,e).
parent(n,e).
female(h).
female(m).
female(n).
female(e).
male(t).
```

We then input two positive examples where the relation *daughter* holds:

```
daughter(m,h).
daughter(e,t).
```

An ILP system is then able to induce the following relation for *daughter*:

```
daughter(X,Y) :- parent(Y,X), female(X).
```

Unlike deductive reasoning in which the conclusions are necessarily true, any rule learnt by the ILP system is a tentative one which might change if it is presented with more facts. In the above example, if `daughter(t,h)` is added as a positive example then the rule drops the `female(X)` clause. We have assumed that the person playing the game doesn't violate any of the rules. If the person who is demonstrating the game to

the visual system doesn't follow the rules it can result in the ILP messing up the rules it might have induced correctly till then.

The problem of induction can also be solved using probabilistic techniques. Bayesian learning can be used to infer what the relationship *daughter* meant by looking at how the probabilities of various evidences of the relations *parent* and $female$ was. Both these fields tend to solve the same problem in different manners mainly because they have assigned different terminology and assumptions for the same concepts. ILP fails in learning where uncertainty needs to be handled explicitly. There is, however, the advantage of expressibility of the rules it learns. While conventional statistical learning is good at soft reasoning and handling uncertainty. But it fails in cases where the learning problem cannot be elegantly described using attribute value representations. Probabilistic Inductive Logic Programming(PILP)[11] brings the best of both approaches together in learning structure and rules.

Inductive logic programming has proven particularly useful in discovering structures in bio-informatics and grammar rules in natural language processing. Our work attempts to discover structure from videos of people playing games and solving solves.

Chapter 2

Semantic Graphs of RGBD Scenes

2.1 Overview

A lot of research in computer vision has focused on object recognition and action recognition. Both these tasks are significant in the process of perception, which enables any intelligent agent to interpret sensory input(specifically visual), to be aware and represent its environment. Only recently has the area of manipulation recognition received the attention it deserves.

Manipulation is the process of handling objects with hands or similar manipulators to change their configuration in some manner. While action recognition is centered around the agent performing the action and involves 'larger' movements like running, kicking and fighting and might even involve movement of the entire body, manipulation recognition is centered around the object the agent is handling. Some examples of manipulation are transfer, rotate, divide and merge. There is usually more than one way to perform the same manipulation. There will be variability in motion trajectory, manipulator pose and relative semantic and spatial relations between the objects present. Hence, it becomes difficult to look at manipulations as if it was just another action. However, there is one thing common to manipulations performed by different people. It is the consequence of the manipulation on the objects involved.

One can look at longer manipulation actions like assembling a chair or making a

sandwich as a sequence of some primitive manipulations being done on objects. They are detected by looking for changes in configuration of the objects present. The primitive consequences might include changes in attributes like colour and shape, translation, rotation, merging with other objects or division of one object into more objects[37]. After having detected these consequences, the system needs to be able to represent them. One novel method of representation is the use of dynamic graphs with objects as nodes and semantic relationships between these objects encoded as edges. Both Semantic Event Chain(SEC)[5] and Visual Semantic Graph(VSG) [37] have similar motivation behind their slightly different approaches to manipulation recognition.

We propose to build a similar graph structure to represent the scene. Our work, however,attempts build these graphs from point clouds instead of images. They are used to detect changes in configuration of the game pieces to identify game states in an unsupervised manner.

2.2 Related Work

Aksoy et al.[5], extracted segments from the images using super-paramagnetic clustering in a spin-lattice model[13]. Doing this allowed them to perform robust markerless tracking of the segments. The segments represent objects in the scene. Spatial relationships are analyzed between each segment pair. There are four kind of relationships which can exist between each pair: *touching, overlapping, non-touching and absent.* Fig. 2.1 depicts how they build a SEC from image segments. Firstly, a matrix encoding all possible relation pairs is created and that is compressed to represent only the change in relation pairs. Using these changes, they model spatial and temporal similarity measure between different manipulation actions to classify them. Very deftly, they convert the problem of comparing dynamic graphs for similarity to one where they encode a dynamic graph as a string and compare strings for similarity.

Visual Semantic Graphs[37] also represent objects as nodes and relationships as edges. Edges exist only if the objects share parts of their borders in images(apparent contact) or

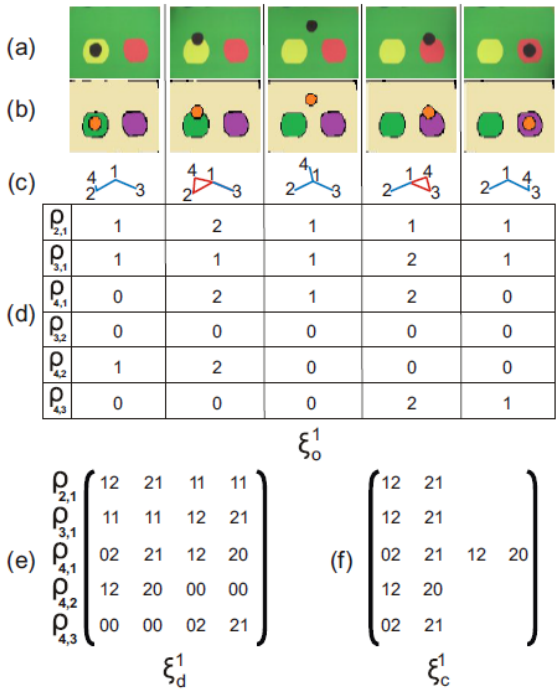


Figure 2.1: Semantic Event Chain for a *Moving Object*[5]

if one segment is contained in another. They also propose the use of depth information in Kinect to create a RGBD distribution model. Segmentation is then done using a weighted graph cut. Then active tracking of these segments is done to build the VSG for each frame. They also suggest the use of optical flow to get a better edge map of the objects in the frame. Using VSG, they run experiments on the Manipulation and Action Consequences dataset and get better results as compared to conventional action recognition techniques like Space-Time Interest Points[23] and Bag-of-Words used with both Linear SVM and Naive Bayes classifiers.

Both the works above provided encouraging results for the vision community as it established that the manipulation recognition problem should not be approached the same way as action recognition. Their results also validate the use of graphs to represent higher-order semantics.

The semantic relationships encoded in the edges between nodes can vary depending on the nature of task. Dantam et al.[10] build a *connection graph* of bars and screws from RGBD images. They use this graph to enable automatic transfer of an assembly

task from human to robot. Aksoy et al.[4] use Semantic Event Chain to build a cognitive system that learns new actions by observing. The system also decides whether to create a new action representation or to modify its existing model.

The main challenge in generating semantic graphs from images or point clouds is the object monitoring process which involves segmentation of the scene into objects and tracking them robustly. During the course of a manipulation many changes might occur. Objects might be occluded by the hand or by other objects, their appearance might be altered and the number of objects might change due to division or merging. Hence, tracking multiple objects during manipulation is a difficult problem. During games and puzzles however, some of the difficulties are removed as objects are usually rigid. There is an additional problem of tracking multiple pieces whose colour and shape are the same like the pawns in the game of chess.

2.3 Segmentation

Image segmentation remains an unsolved yet well-studied problem in computer vision. From detecting blobs of colours and textures as objects, researchers have come a long way on tackling image segmentation. Graph-based image segmentation[14] and its many variants provide decent segmentation of the scene but their parameters need to be fine-tuned. One method of segmenting 3D objects is to generate lots of candidate segments by changing the parameters of the segmentation algorithm and then project these segments onto 3D space and rank them according to the *objectness* score[8]. One alternative would be to use superpixels[3] to segment the scene because they are very good at detecting the boundaries of objects robustly.

Currently state-of-the-art methods in 3D object segmentation are based on alternative representations of the points in the cloud. For example, Koo et al.[20] represent objects are Gaussian Mixture Models. This enables them to identify dynamic objects, whose shape might change due to deformation or occlusion. Using a GMM allowed them to represent any shape of an object as a 3D probability distribution of the true positions.

Papon et al.[28] describe the 3D counterpart of superpixels called supervoxels. They oversegment the point cloud into perceptually similar regions based on colour, distance and normal similarity between the constituting pixels. A graph-cut based approach on the supervoxels then gives us object candidates. This method also encodes information about supervoxel connectivity that is whether two of them are neighbours in 3D space or not. Supervoxels effectively reduce the number of data points to work with by replacing the constituting points with one single point. This allows for real-time processing of point clouds using supervoxels.

With 3D data, object segmentation can be performed to cluster points close to each together based on Euclidean distance[30]. Our Algorithm 1 is a modified version of the above in which we perform filtering based on the colour in the HSV space before the clusters of points are discovered in the scene by doing Euclidean clustering based on distance. This is done because sometimes game pieces of different colours might be placed on top on another or in contact with each other like in the Towers of Hanoi. So our objective is to extract clusters of points as objects. These clusters should either have perceptually different colours or be separated above a particular threshold in space.

Algorithm 1 Pipeline to extract objects from scene

1. Use a Pass Through filter to focus on the table-top.
2. Use RANSAC to filter out points of the table-top from the cloud.
3. Perform Colour-based filtering of the point cloud in HSV space.
4. Do euclidean clustering of the different colour clouds to give objects that are either separated in space or have perceptually different colours.

Figure 2.2: Objects found in a scene from the animated Towers of Hanoi dataset

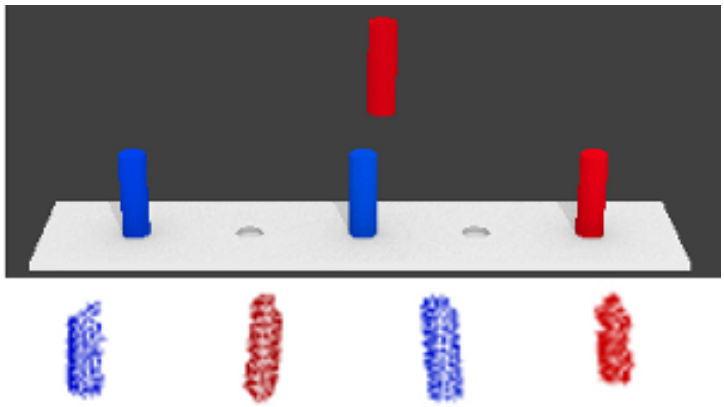


Figure 2.3: Objects found in a scene from animated 1D Peg Solitaire dataset

Figure 2.4: Objects found in a scene from the real Towers of Hanoi dataset

2.4 Multi-object Tracking

Supervoxels can prove useful in robustly tracking point clouds. Papon et al.[29] use a persistent supervoxel world-model which is updated for every new frame. This helps in tracking through occlusions. They use a particle filter to perform 6 DoF pose tracking. In another pioneering work, Koo et al.[21] present a novel model for tracking multiple objects by associating the points used to build the GMM object representation both spatially and temporal. Spatial association ensure object points are tracked in spite of occlusion. Both these works underline the importance of tracking the parts of the object instead of trying to track objects as a whole.

The multi-object tracking problem has been often looked at as a data association problem. More specifically, a label associated with an object needs to be linked with the same object in the next frame and this needs to be done with all objects present in

the scene. A model-based detection method to track objects was not used in our case as many of them shared the same shape and the correspondence matching algorithm needs the fine-tuning of many parameters.

We propose a new method to track multiple-objects in a point cloud video. It is based on the occupancy of voxels by an object in one frame and the next. The Hungarian method is used to solve the label assignment problem in polynomial time and as a result, track objects robustly. We also suggest a modified approach of the same that can improve tracking under full occlusion.

The assignment problem is a combinatorial optimization problem. It consists of finding a maximum weight matching in a weighted bipartite graph. In other words, there are two sets $A = \{a_1, a_2, .., a_n\}$ and $B = \{b_1, b_2, ..., b_n\}$. There is a certain cost for matching a a_i with a b_j. The assignment problem si to match each members of set A one member of set B such that the total cost of the assignments is minimized. Fig. 2.5 depicts the optimal solution of an assignment problem between $\{r_1, r_2, r_3\}$ and $\{t_1, t_2, t_3\}$.

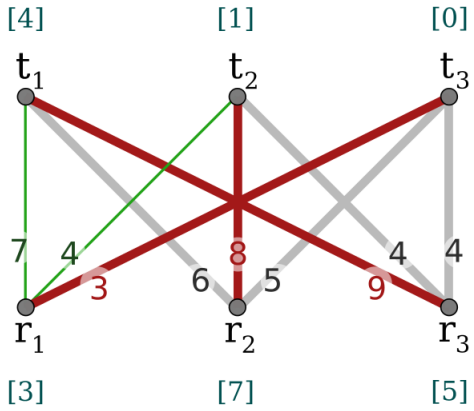


Figure 2.5: An example of the assignment problem and its optimal matching solution. Taken from http://www.frc.ri.cmu.edu/ lantao/

Multiple object tracking can be reduced to an assignment problem where the objects detected in frame i need to matched with themselves in frame $i+1$. The Euclidean distance between the centroids of the objects can be used as a distance metric to perform the matching[10]. But this might fail if there is are multiple objects moving simultaneously. Hence, we use octree overlap between point clouds to assign weights for matching

between two clusters, which is described in more detail below.

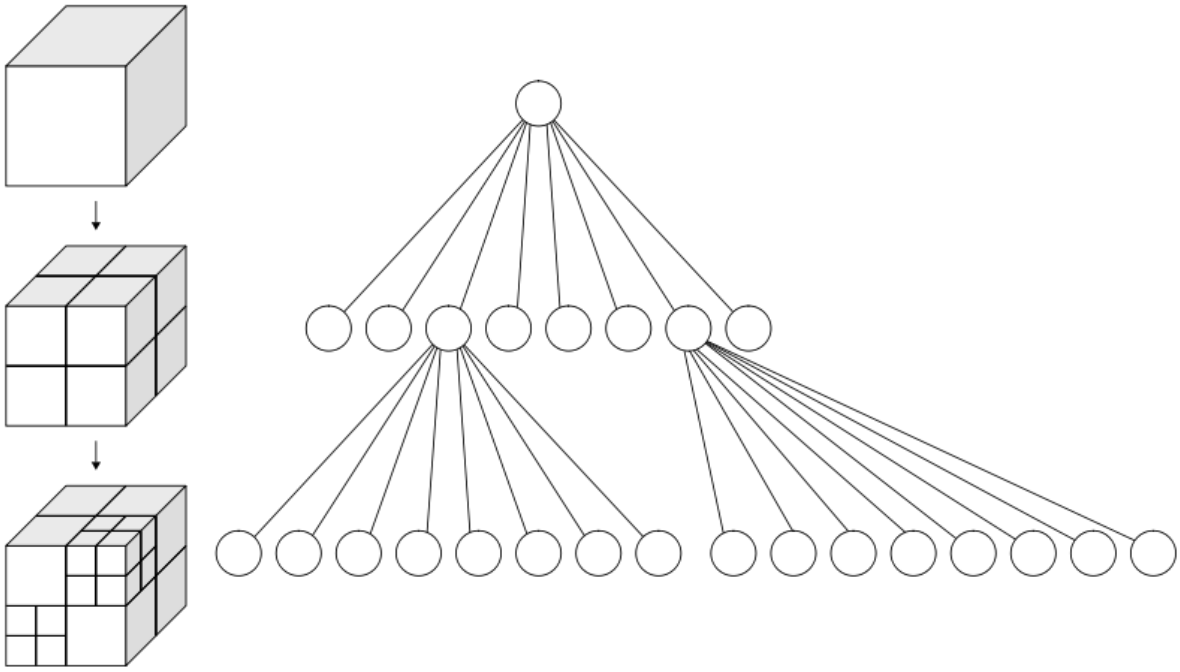


Figure 2.6: Division of a cube into octants and corresponding octree data-structure. Taken from http://en.wikipedia.org/wiki/Octree

Octree is a hierarchical tree data structure used to store points in point cloud data.[24] It is helpful in in spatial partitioning, downsampling of the point cloud into voxels and search operations on the point data set. Each node in the octree has either eight children or none. The root node describes a cubic bounding box which encapsulates all points. At every tree level, this space becomes subdivided by a factor of 2 which results in an increased voxel resolution.[2] The space will get divided till the voxel resolution reaches a given threshold. This is used to downsample the number of points to be processed. Fig. 2.6 shows an octree being built by recursive division of the octants of the cube. If there is no division of an octant it implies there is no point present in it.

We build the octree representation of the objects found by segmentation in two consecutive frames. If it moves, there is going to be a spatial overlap between the two objects in the two consecutive frames. This overlap will be zero with the other objects present in the scene. In Fig. 2.7 a cube is shown inf frame i and the same cube is translated in frame $i+1$. The gray area represents the overlap between these two cubes.

We propose to use this overlap in space to track objects.

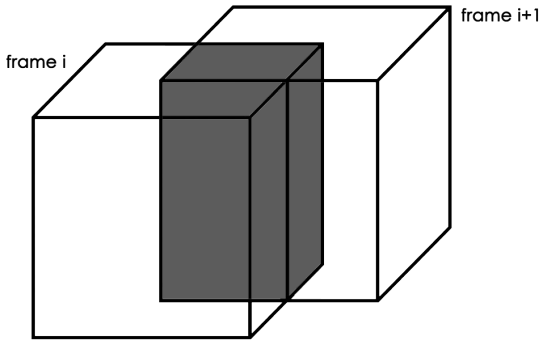
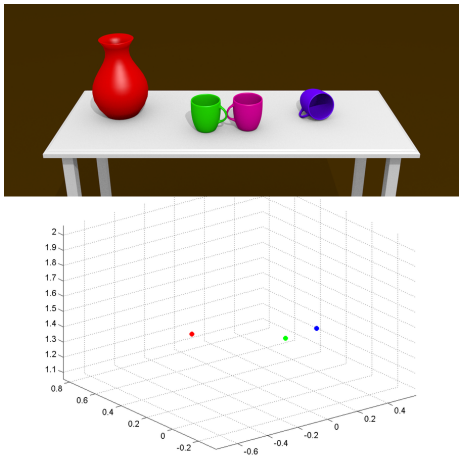


Figure 2.7: Overlap between an object in one frame and the same object in the next frame.

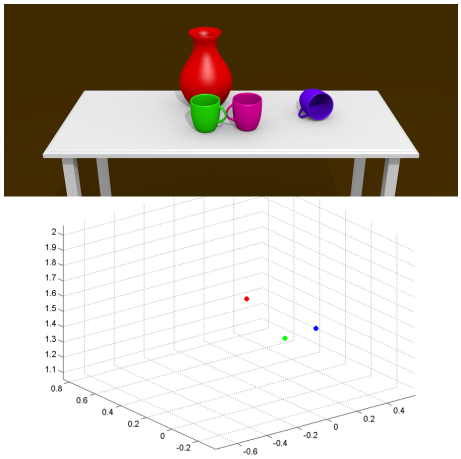
We maximize the sum of all overlaps while assigning labels from one frame to the next. There are two assumptions that make this tracking algorithm work. Our objects of interest are non-planar and rigid. Planar objects may have zero overlap with themselves in the next frame. The action performed by the player is slow enough for the Kinect to record the movement of the objects. If the frame-rate of recording the point clouds is slow there will be no overlap. In our case, however, we recorded gameplay at the usual pace a person plays and there was considerable overlap between the same objects in consecutive frames at normal Kinect recording rates.

Algorithm 2 Pipeline to track multiple objects in point cloud videos

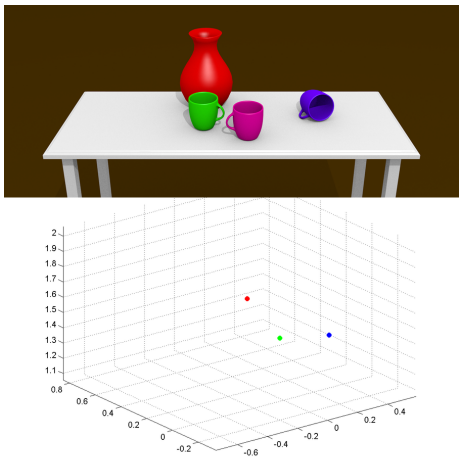
$P \leftarrow P_1, P_2, P_3..., P_N$ ▷ Sequence of point clouds in time
$i \leftarrow 1$ ▷ Frame number
$overlap \leftarrow [[]]$ ▷ Matrix to store overlap between two objects in consecutive frames
$label \leftarrow []$ ▷ k^{th} object in frame i is vector $label(k)^{th}$ object in frame $i+1$
while $i < N$ **do**
  $Objects_i \leftarrow HSVEuclideanClustering(P_i)$
  $Objects_{i+1} \leftarrow HSVEuclideanClustering(P_{i+1})$
  for all $object_i$ in $Objects_i$ **do**
    for all $object_{i+1}$ in $Objects_{i+1}$ **do**
      $getOctreeOverlap(overlap, object_i, object_{i+1})$
    end for
  end for
  $label \leftarrow HungarianAssignment(overlap)$
  $i \leftarrow i + 1$
end while



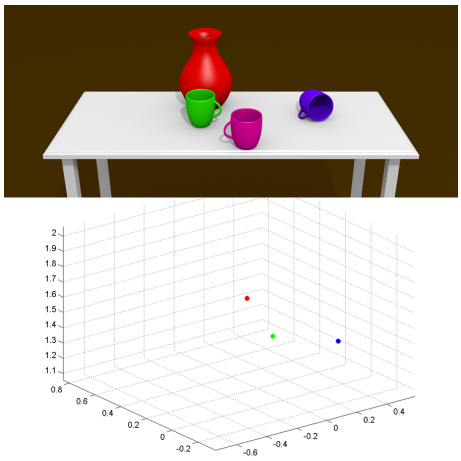
(a) Frame 1(Red vase moves)



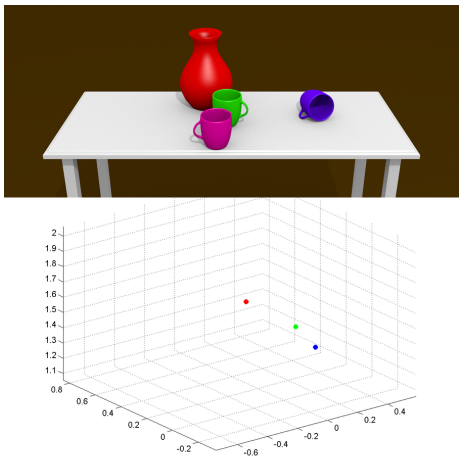
(b) Frame 40



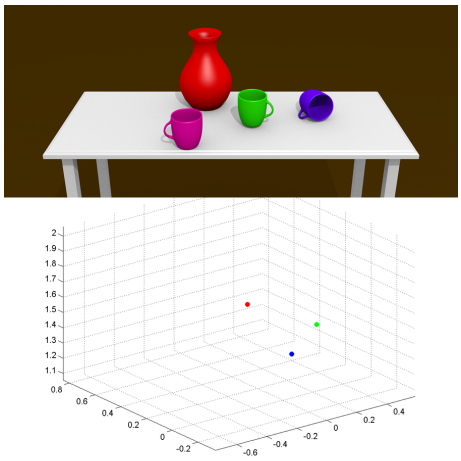
(c) Frame 50(Green and purple cups)



(d) Frame 60(Green and purple cups)



(e) Frame 70(Green and purple cups)



(f) Frame 80

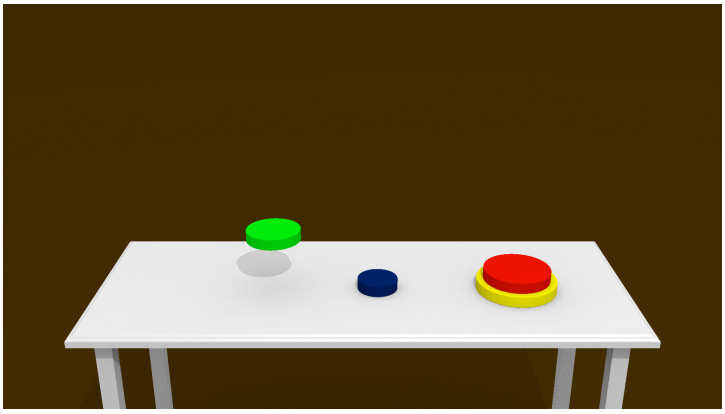
Figure 2.8: Multiple object tracking results on *Table* dataset

Along the lines of the MATLAB tutorial on *Motion-based Multiple Object Tracking*[1] from videos, we can adapt our method to be more robust to occlusions and missed detections by using a Kalman filter. Instead of associating objects of this frame to those present in the last frame, one can initialize tracks using objects in the first frame. From this point onwards, the Hungarian assignment algorithm is used to assign objects present in each frame to their corresponding tracks by minimizing the Euclidean distance of the predicted coordinates and the observed coordinates. In case, there is a missed detection of an object, the value predicted by the Kalman filter can be used instead. If the object is not detected for a given number of frames, it can be assumed to be lost and the track needs to be deleted. This is a work in progress and may later be used track objects in the scene.

2.5 Semantic Graphs

We have described above our the pipeline used to perform object monitoring. The process of building semantic graphs is not independent of the task one is looking at. Essentially, a semantic graph of the scene encodes certain meta-information about the relationships between the objects. Dantam et al.[10] encoded where on a bar is another one being attached by building a connection graph. Yang et al.[37] created an edge between two nodes if they were in contact in the images. While Aksoy et al.[5] looked at encoding the relationship between nodes as semantic label to edges between the nodes that is there is an edge even if they are not in contact. They also encoded the semantic relationship *overlapping* which meant one segment is included in another. Both of the above, had the objective of recognizing manipulation actions from these videos. Our aim is to learn the rules of puzzles from the semantic graph of scenes. Therefore, we must look for a semantic relationship that helps us to identify game states and compare them for changes.

After looking at some games, we came to the conclusion that the relationship *on* between two objects will be able to help us on our task. In most board games or puzzles



(a) Frame 93 from the animated *Towers of Hanoi* dataset

(b) Corresponding Semantic Graph for *contact*

Figure 2.9: An example of a Semantic Graph

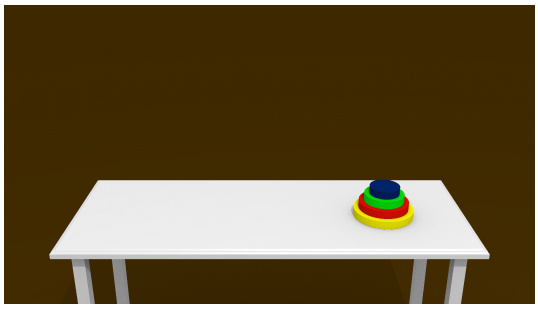
the game state is altered by picking up a piece and placing it somewhere else *on* the board. Sometimes, the pieces are not placed on the board but *on* top of each other as is the case in the Towers of Hanoi.

Fig. 2.9 depicts an example semantic graph. In the scene shown, the pieces have been labeled in decreasing order of size. The largest piece which is yellow in colour is labeled 1, the red one is labeled 2, the green one is 3 and the smallest blue one is 4. The board has been labeled as B. There exists an edge between any two node in the graph if they are in contact with each other. As is encoded in the graph, there exists no edge with the

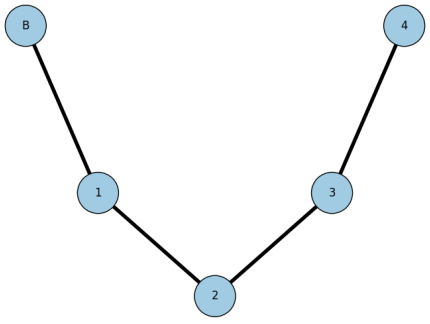
green piece as it is in the air and not in contact with anything. Two nodes 4 and 1 are in direct contact with the board. All this information is succinctly captured by the graph. Changes in this semantic graph will represent some action being performed in the game.

If the semantic relationship of *on* is encoded in the graph then we will be able to discover the states of the game by looking for configuration changes of the game pieces on the board. Every time a player lifts up a piece, an edge between the board and the piece disappears as it is no longer on the board. The moment the player places the piece back on the board or on another piece, a new edge is formed. Hence, game states can easily be discovered from the video by looking for states where the number of edges changes. Each node of the graph represents an object and also stores meta-information about it like the coordinates of its centroid, average colour of the object, number of visible points and the volume occupied by the bounding box of the object in the current frame. After the states have been detected, the change from one state to another can be found out by looking for changes in the meta-information.

For example, in the *Towers of Hanoi* dataset our method is able to identify the 16 game-state frames from about 750 frames. The ones in between them can now be discarded. The learning of the rules of the game will done from only the game-state frames. In Figs. 2.10 and 2.11, some game states that were discovered automatically are shown.

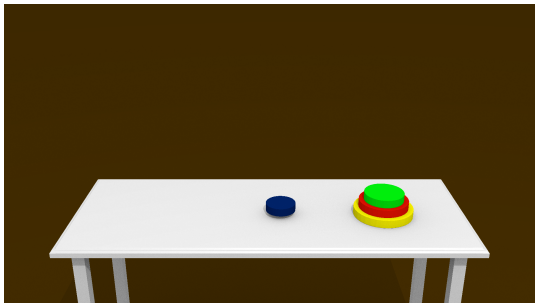


(a) Frame 1

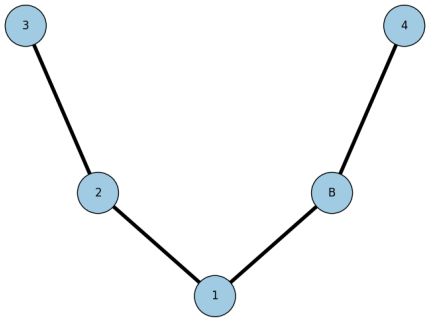


(b) Corresponding Semantic Graph

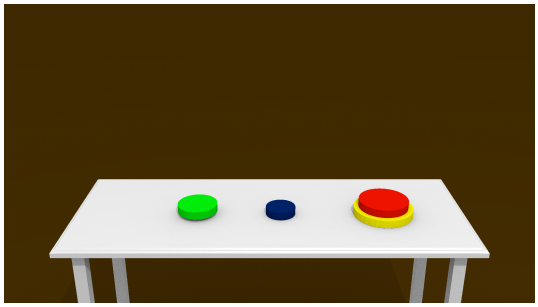
Figure 2.10: Automated Detection of Game States in the *Towers of Hanoi*(contd.)



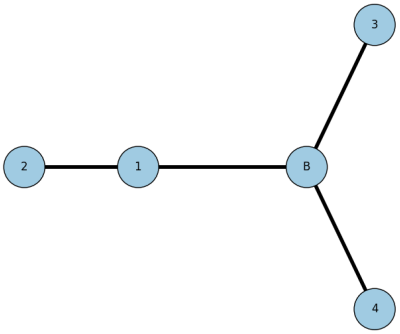
(a) Frame 46



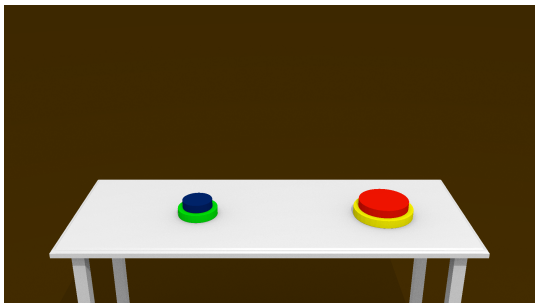
(b) Corresponding Semantic Graph



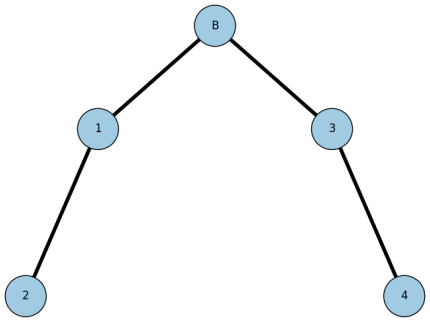
(c) Frame 108



(d) Corresponding Semantic Graph



(e) Frame 149



(f) Corresponding Semantic Graph

Figure 2.11: Automated Detection of Game States in the *Towers of Hanoi*

2.6 Summary

We presented a framework for monitoring of game pieces. Firstly, game pieces are segmented based on their colours in the HSV space followed by a Euclidean clustering algorithm to identify individual game pieces. An octree overlap metric is used with the Hungarian algorithm to assign labels from one frame to the next to track the game pieces. Semantic graphs which encode relationships between the objects are built. Selecting the relationship of *contact* helped us to automatically identify game states from a video. Using these game states, we aim to derive logical clauses from the video. In the next chapter, we describe how we go on to discover rules of puzzles from these clauses.

Chapter 3

Learning Rules from Semantic Graphs

3.1 Overview

Many puzzles and games involve iterations of two steps - spatial reconfiguration of the pieces and updating the state of the game. The previous chapter dealt with detecting these reconfigurations or game states. In the following sections we describe how we represent these game states in a manner that enables us to learn the rules.

Usually, there cannot be arbitrary updates in states. Certain rules need to be followed to reconfigure the game pieces. For example, the move of a knight in chess resembles the shape L and in the Towers of Hanoi a bigger piece cannot be placed on a smaller piece. The best way to represent these rules would be in form of logical clauses. Our objective is the inverse of playing a game in accordance with some rules that is we intend to guess the rules of the puzzles and games by observing people play.

3.2 Learning to Represent Game States

There can be almost as many game state representation as many there are games. These encodings of game states often enable a system to play the game intelligently. For

example, in the 3 block Towers of Hanoi a game state can be represented by simply telling in which pole each block is. The arrangement for the blocks in each pole is fixed because of the size rule that fixes their order. Say the names of the poles are a, b and c, the game state for Towers of Hanoi can be represented as the string aab. This means the first two blocks are in pole a while the third block is in pole b. In the 1D peg solitaire the game states can be represented as strings like $rregg$. This means that there are two red pegs followed by an empty space and then there are two green blocks. Consider games based on grids like Tic-Tac-Toe or Chess where the game states can be represented as which game piece has occupied which square in the grid. Hence, learning an effective representation game state is a difficult problem. But if we are able to get that then coupled with the rule that is learnt, we can have a General Game Learning and Playing system that is able to learn the rules by observation and then is ready to play with people by coming up with valid game moves on its own.

We try to use a representation that is general enough so that many different games can be incorporated. After observing for an extended period of time, our visual framework discovers the discrete positions on the board which game pieces occupy. These are found by clustering the positions of the game pieces during the game states not when they are being manipulated by the hand. Now for each game state we assign the game pieces to their respective cluster. This might leave us with a cluster that is unoccupied which can be represented as empty. Therefore, each game state can be encoded as a string of game piece labels at the cluster positions occupied by the game pieces. For example, the first game state in 1D peg solitaire will be `[{a},{b},{},{c},{d}]` where a, b, c and d are the labels given to the game pieces. The third hole is unoccupied in the beginning which is represented by the empty set. In Towers of Hanoi, the state corresponding to frame 108 shown in Fig. 2.11 will be represented by `[{a},{b},{c,d}]`. Notice that the third position has a set consisting of two game pieces. This representation is there to handle games where pieces can be placed one on top another occupying the same discrete cluster on the board. This can be extended to 2D games where a matrix of characters will represent the game state. We describe below how these clusters are discovered and

the game state representations are created.

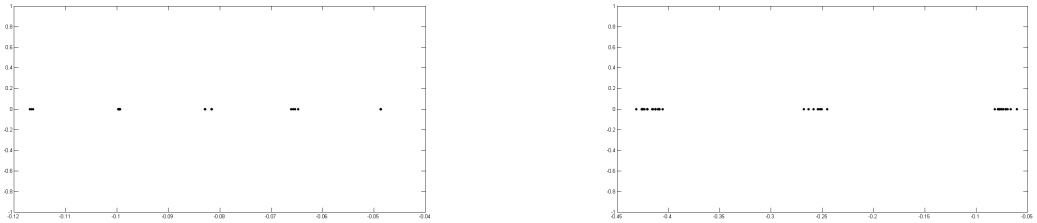
We need to change our frame of reference from the camera's frame to the board's to be able to represent the game states in a better manner. X_b, Y_b, Z_b are coordinates of the object in the frame of the board which will be used to find the clusters. These co-ordinates are obtained by the following operation on the camera coordinates X_c, Y_c, Z_c using the following equation:

$$\begin{bmatrix} X_b \\ Y_b \\ Z_b \end{bmatrix} = \begin{bmatrix} cos(\hat{x}_b, \hat{x}_c) & cos(\hat{x}_b, \hat{y}_c) & cos(\hat{x}_b, \hat{z}_c) \\ cos(\hat{y}_b, \hat{x}_c) & cos(\hat{y}_b, \hat{y}_c) & cos(\hat{y}_b, \hat{z}_c) \\ cos(\hat{z}_b, \hat{x}_c) & cos(\hat{z}_b, \hat{y}_c) & cos(\hat{z}_b, \hat{z}_c) \end{bmatrix} \times \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$$

In the above equation, $cos(x, y)$ is the cosine of the angle between the unit vectors x and y. $\hat{x}_b, \hat{y}_b$ and $\hat{z}_b$ represent the unit vectors of the axes in the frame of the board while $\hat{x}_c, \hat{y}_c$ and $\hat{z}_c$ represent the unit vectors of the axes in the frame of the camera. $\hat{z}_b$ is obtained as the average of normals of the points on the board. $\hat{x}_b$ and $\hat{y}_b$ are obtained by performing SVD on the coordinates of the game-pieces over all frames. One-dimensional games have only one significant eigenvalue. The eigenvector corresponding to that eigenvalue gives $\hat{x}_b$ if it doesn't coincide with $\hat{z}_b$. Similarly, In 2D games the second significant eigenvector gives $\hat{y}_b$. This can also be found as a cross product of $\hat{z}_b$ and $\hat{x}_b$. The above generalizations don't hold true when the game being played doesn't conform to an usual rectangular grid like triangular peg solitaire.

The system does not have any idea in the beginning whether the game is 1D or 2D or 3D. After it has discovered the game states by using the methods described in the previous chapter, it populates a list of the positions of all the game pieces across all the game-state frames. These are data points where game pieces have visited during the game play. By running SVD on this data we can discover the intrinsic dimensionality of the game. In this case, only one eigen value is significant(ratio based thresholding). Therefore, it can be induced that the game is based on one dimension. The next step is to look for clusters in the positions occupied by game pieces in those frames. While finding out the optimal number of clusters is an open problem, there are statistical methods

to estimate the optimum number of clusters in a dataset like ours. One method will be to look for an elbow or a bend in the sum of squared error(SSE) plot. The average silhouette index over all data of a cluster is a measure of how tightly grouped all the data in the cluster are. This can also be used to determine the number of natural clusters in the data. There are as many 30 different such metrics that can be used to estimate the number of clusters in the positions of the game pieces. We use the elbow detection method to find the possible number of positions each piece may occupy.



(a) Five clusters in animated *1D Peg Solitaire*. (b) Three clusters in real *Towers of Hanoi*.

Figure 3.1: Clusters formed in the significant dimension

In Fig. 3.1(a) there are five clusters(Fig. 3.2) that can be discovered from the data corresponding to the five holes in 1D Peg Solitaire. Similarly in Fig. 3.1(b) and there are three clusters(Fig. 3.3) discovered corresponding to the three poles in Towers of Hanoi.

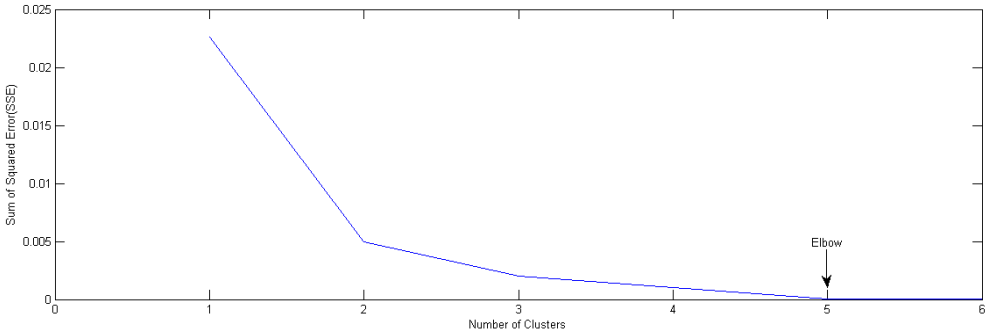


Figure 3.2: Elbow method to discover number of clusters in *1D Peg Solitaire*

The locations of the clusters are discovered by performing k-means clustering using the value of k found by using the elbow method. For each game state, each game piece is assigned to its nearest cluster. Doing so, gives us the string based representation

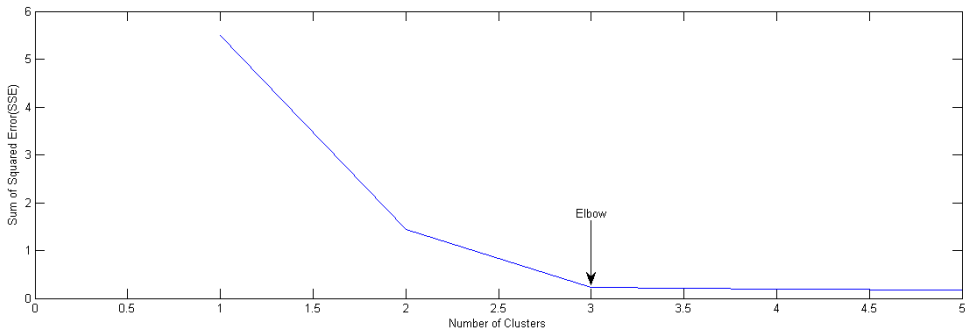


Figure 3.3: Elbow method to discover number of clusters in *Towers of Hanoi*

of the game state that we had aimed for. Having obtained these strings, we transfer information to a logic programming system which will be better domain to induce the rules of games.

3.3 From Semantic Graphs to Horn Clauses

Horn clauses are the basis of logic programming in Progol. Clauses in Prolog are generally written in the form `u :- p, q, ..., t.` which means to say u is true we need to prove that the clauses $p, q, ..., t$ are all true.

We use meta-information contained in the nodes of the graphs and changes in that from one game state to the next to generate logical clauses that will help us learn the rules. As described in the primer on ILP in Section 1.7 we will be needing background knowledge, positive examples and negative examples to be able to come up with hypotheses regarding the rules of the game. But we can't produce negative examples by observing people playing a game as it is assumed they play without violating any rules.

3.3.1 Background Knowledge

We focus our attention on reconfiguration puzzles like the Towers of Hanoi and 1D Peg Solitaire. In such puzzles, the game pieces are the objects that will be monitored to produce the semantic graphs representing the scene. Attributes of the game pieces like color, shape and size determine what sort of moves the pieces will be able to perform.

In Towers of Hanoi, it is the relative size of the pieces. In chess, the shape determines the moves a piece can perform. A child that is learning a game for the first time picks up rules by associating it with some pre-defined concepts like shape and color. Hence, it is also a viable assumption to provide our system with classifiers that can ascertain the shape, color and relative size of the game pieces.

We use classifiers for different colours and shapes to generate clauses for each object. But before that we declare each object discovered as a game piece. In a 4 block Towers of Hanoi, the following is the declaration of pieces a, b, c and d:

```
piece(a).
piece(b).
piece(c).
piece(d).
```

Both our games are one dimensional in nature. So we will be representing location cluster using one variable only. In the 1D Peg solitaires 5 clusters are discovered. Each cluster is also associated with a number which helps in comparing their position with other clusters. They are declared as follows:

```
x(l1).
x(l2).
x(l3).
x(l4).
x(l5).
```

For the colour classifier which uses non-overlapping HSV ranges to discriminate colours based on pre-defined values for each one, we declare each colour in the following manner:

```
colour(red).
colour(blue).
colour(yellow).
colour(green).
```

For each classifier, we declare more background knowledge regarding their colour:

```
colour(a,red).
colour(b,green).
colour(c,yellow).
colour(d,blue).
```

Numerical features like size are then declared:

```
size(a,1).
size(b,3).
size(c,9).
size(d,10).
```

Shape classifiers based on 3D features or models of game pieces can be added for some basic shapes like cylindrical, cuboid and spherical. Colour classifiers can be easily defined by setting ranges for perceptually different colours in HSV colour pace or LAB colour space. This can be directly incorporated in to the segmentation algorithm. This database of colours can be updated as the system sees more colours or can be pre-defined.

For each numerical feature there is a meta-clause generator that compares their values. For example the clause generated for size is shown below:

```
greatersize(A,B) :- piece(A),piece(B),size(A,NA),size(B,NB),NA>NB.
```

These comparators are generated for every feature that is numerical. Another feature that can be used to define the rule of the game is the height of the game piece. Currently, size is estimated by the average value of the bounding box of the game piece.

We also provide clauses *absDiff* which gives us how many steps has a game piece been moved in one direction. *diff* gives us also the direction of change. This would help us induce the rules regarding the movements of the game pieces. The final piece of background knowledge are the rules for *top* and *bottom* which give us the top and bottom pieces in a list.

```
diff(X1,X2,Diff):- x(X1),x(X2),project(X1,N1),project(X2,N2),
```

```
Diff is N1-N2.

abs(X,X) :- X>=0.
abs(X,Y) :- X<0, Y is -X.
absDiff(X1,X2,Diff) :- x(X1),x(X2),project(X1,N1),project(X2,N2),
Diff1 is X1-X2, abs(Diff1,Diff).

top(A,[A]).
top(A,[B|C]) :- top(A,C).

bottom(A,[A]).
bottom(A,[B|C]) :- bottom(A,B).
```

We expect the above background knowledge is enough to induce rules of many games.

3.3.2 Positive examples

The next course of action is to declare positive clauses. The most general relationships that we have identified by looking at some games and puzzles can be broadly divided into three types:

1. Between objects in the same frame. These are extracted from the semantic graph. We use the relationship *on* between two objects in one frame and only exists between objects which are physically on each other. It is not a symmetric relationship. The clause shown below is from the Towers of Hanoi game and represents a piece d being on piece a.

   ```
   on(d,a).
   ```

2. Between the same object in different frames. These will help us identify the valid moves a game piece can make during game-play. We use the relationship *move* for that purpose. It encodes the movement of an object from one cluster to another.

These clusters also have a number associated with them. The clause shown below is from the 1D Peg Solitaire which represents a piece called $p5$ goes from location $l4$ to $l5$.

```
move(p5,l4,l5).
```

3. Between the clusters detected and the game piece during gameplay. We use the relationship $fromto$ to encode the above. It encapsulates the state at the source cluster before the move and the final state at the target cluster. The clause shown below is from the Towers of Hanoi game and represents a piece d being moved from position *l1* to position *l2* and the initial set of game pieces at *l1* was `[a,b,c,d]` and the set of game pieces at *l2* after the move was `[d]`:

```
fromto(l1,[a,b,c,d],d,l2,[d]).
```

While it might appear that *move* and $fromto$ relationships are somewhat redundant, the former represents a move from the perspective of the game piece while the latter represents the move keeping the location clusters in mind. For some games, some of the relationships will not be useful. In Towers of Hanoi, the *move* relationship is not that useful. In 1D Peg Solitaire *on* is not useful. These relationships fail to get generalized when we run ILP using PROGOL.

3.4 Learning Rules of Puzzles

Both background knowledge and positive examples are fed to PROGOL. It is an ILP system that tries to generalize the clauses given in positive examples in terms of the relationships defined in the background knowledge.

For example, one rule that is learnt for the Towers of Hanoi:

```
on(A,B) :- greatersize(B,A).
```

The above rule is read as if there exists a game piece A on game piece B then B is bigger than A.

We used readable relationships in formulating clauses for background knowledge and positive examples. This allowed us to come up with rules that are readable and directly related to what they should mean to people who aren't aware of the game. In other words, the rules of the game come up in terms of relationships *on*, *move* and $fromto$ that have the same meaning that they have to a human observer because we have chosen such labels for the relationships. Classifiers for colour, shape and size also help in this symbol grounding exercise.

In the next chapter, we describe the games in our dataset, show the results of the visual system and enlist the steps leading up to the final learning of the game rules.

Chapter 4

Demonstrations

4.1 Towers of Hanoi

The Towers of Hanoi is a puzzle which consists of a number of disks of different sizes that can be placed in three rods or fixed centers. The puzzle starts with the all disks in a stack in ascending order of size, the smallest at the top in one center. The objective of the puzzle is to move the entire stack to another center or rod by obeying the following simple rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
3. No disk may be placed on top of a smaller disk.

Our system needs to come up with as many rules as it can with the backound knowledge that we have provided it with.

Figure 4.1: Objects found in a scene from the animated *Towers of Hanoi* dataset

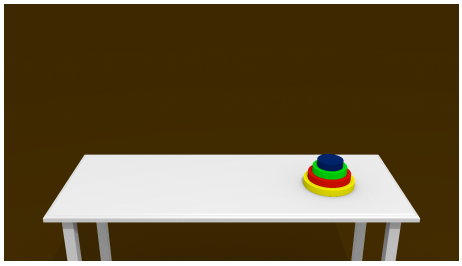
4.1.1 Animated Dataset

We used BlenSor to animate four differently sized blocks demonstrating the *Towers of Hanoi* puzzle being solved. It consisted of 740 frames of 640 × 480 RGBD images recorded on an artificial Kinect sensor in BlenSor.

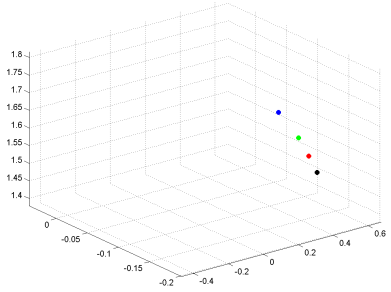
In Fig. 4.1 the clusters of points(that is the game pieces) are shown. Four clusters of different sizes are discovered. In Figs. 4.2 and 4.3, the results of our octree overlap based tracking method on the four clusters are shown. There is a shift in the centroid of the object on which another is placed because some points of the lower object get hidden. Figs. 4.4, 4.5 and 4.6 are the states that are discovered automatically by looking at how the game pieces have been reconfigured in the semantic graph.

We run the classifier for colour(from pre-defined HSV ranges) and relative size(volume of bounding box) in the first frame and make a safe assumption that no new pieces will enter the scene, a rule that most puzzles and games follow. From the first frame we get the following clauses for game pieces a, b, c and d:

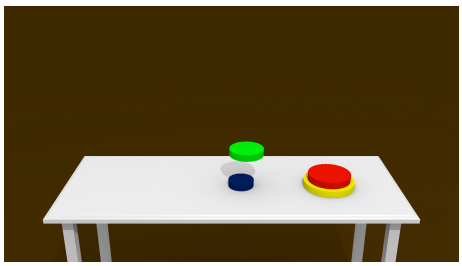
```
colour(a,yellow).
colour(b,red).
colour(c,green).
colour(d,blue).
size(a,10).
size(b,8).
size(c,4).
size(d,2).
```



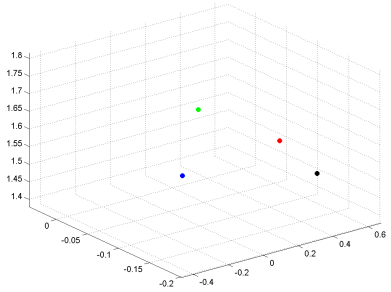
(a) Frame 1



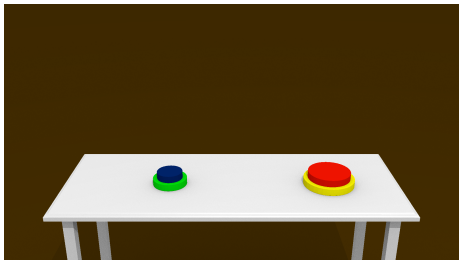
(b) Tracked Objects in Frame 1



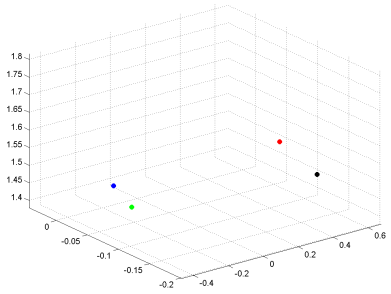
(c) Frame 75



(d) Tracked Objects in Frame 75

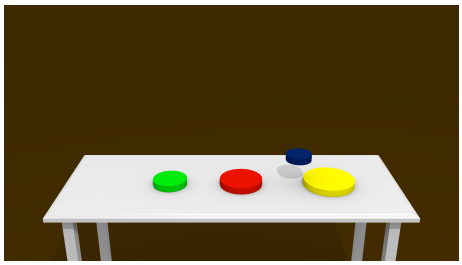


(e) Frame 150

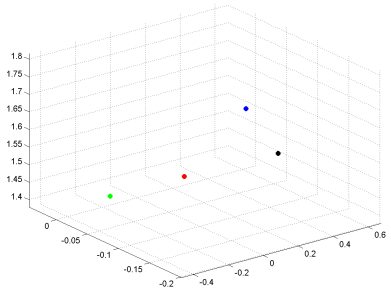


(f) Tracked Objects in Frame 150

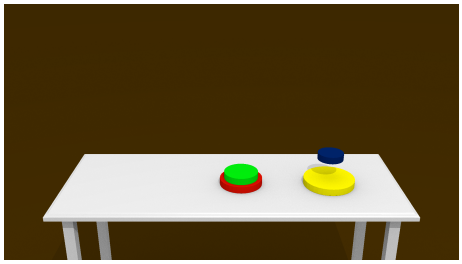
Figure 4.2: Tracking of game pieces in the *Towers of Hanoi* Animated dataset(contd.)



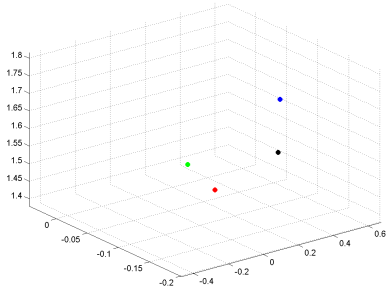
(a) Frame 225



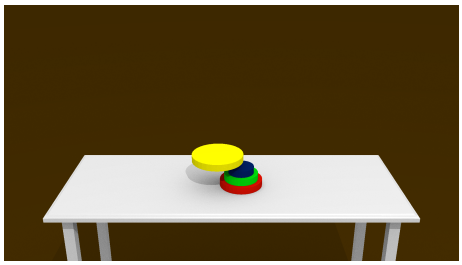
(b) Tracked Objects in Frame 225



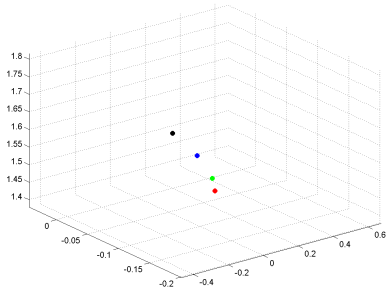
(c) Frame 300



(d) Tracked Objects in Frame 300)

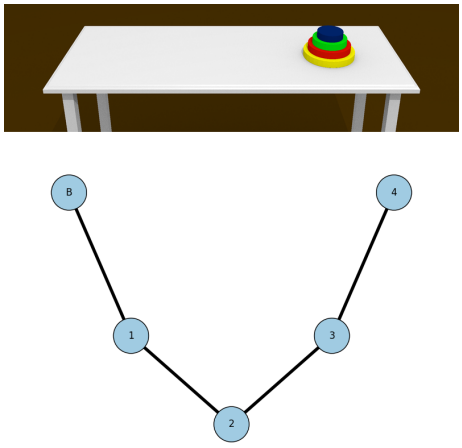


(e) Frame 375

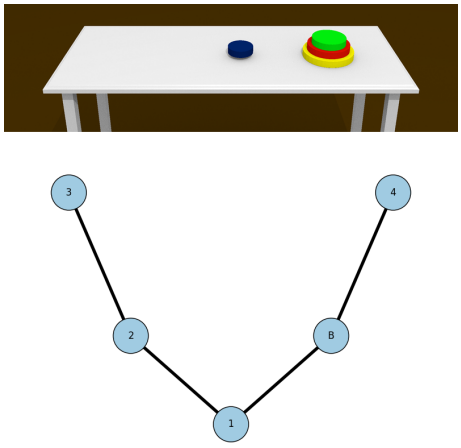


(f) Tracked Objects in Frame 375

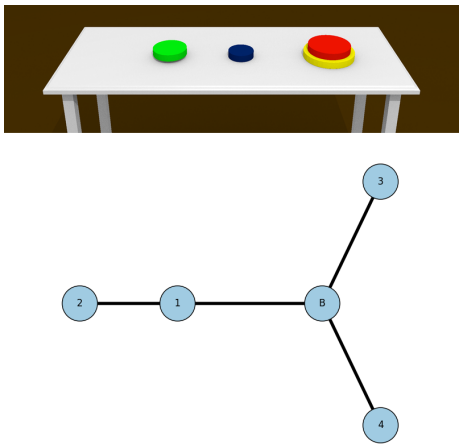
Figure 4.3: Tracking of game pieces in the *Towers of Hanoi* Animated dataset



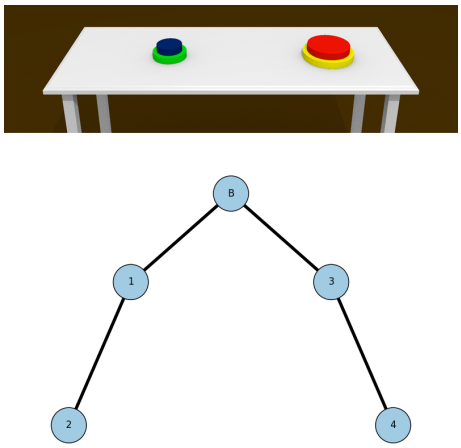
(a) Frame 1 and its Graph



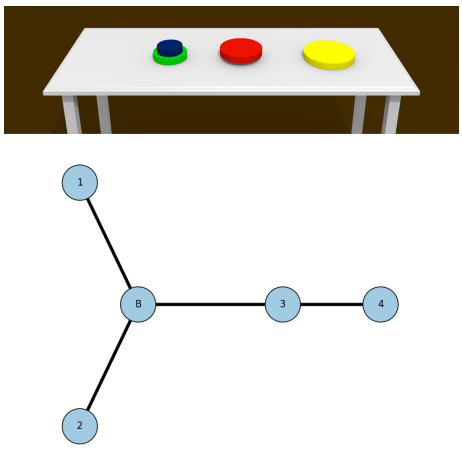
(b) Frame 46 and its Semantic Graph



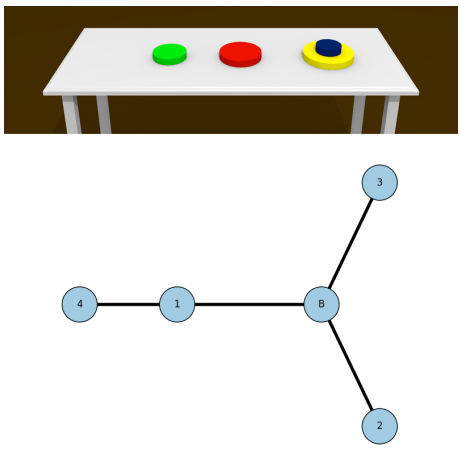
(c) Frame 108 and its Semantic Graph



(d) Frame 149 and its Semantic Graph

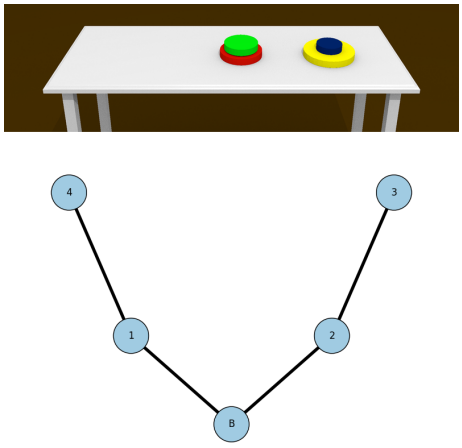


(e) Frame 182 and its Semantic Graph

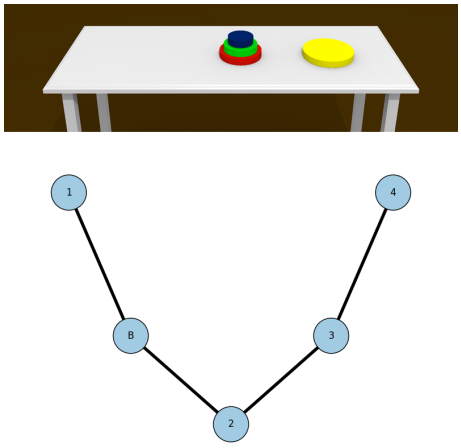


(f) Frame 249 and its Semantic Graph

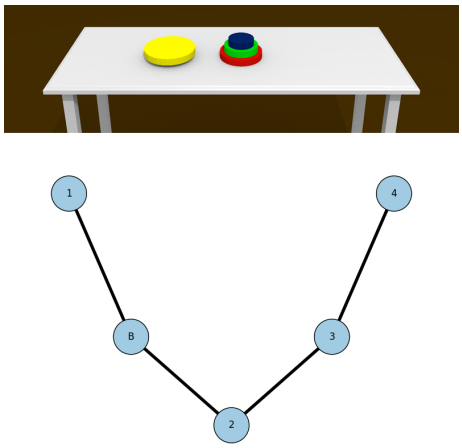
Figure 4.4: Automatic Discovery of Game States in the *Towers of Hanoi* Animated dataset(contd.)



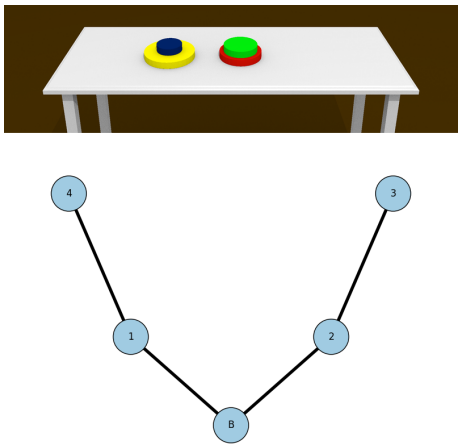
(a) Frame 288 and its Semantic Graph



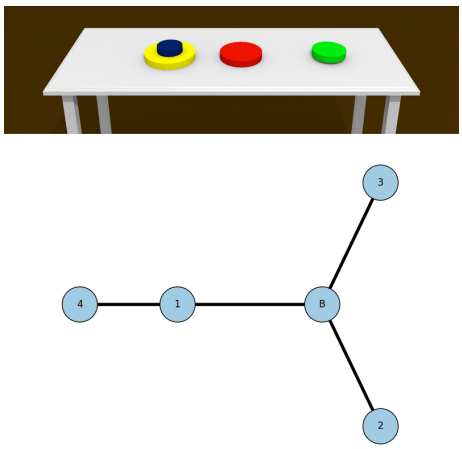
(b) Frame 328 and its Semantic Graph



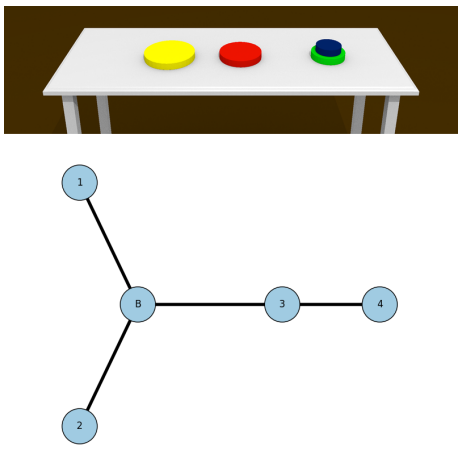
(c) Frame 405 and its Semantic Graph



(d) Frame 449 and its Semantic Graph

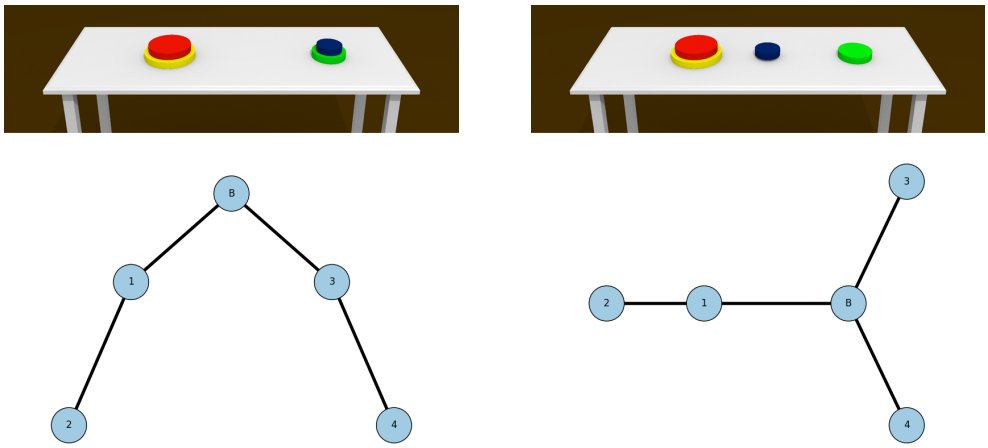


(e) Frame 488 and its Semantic Graph



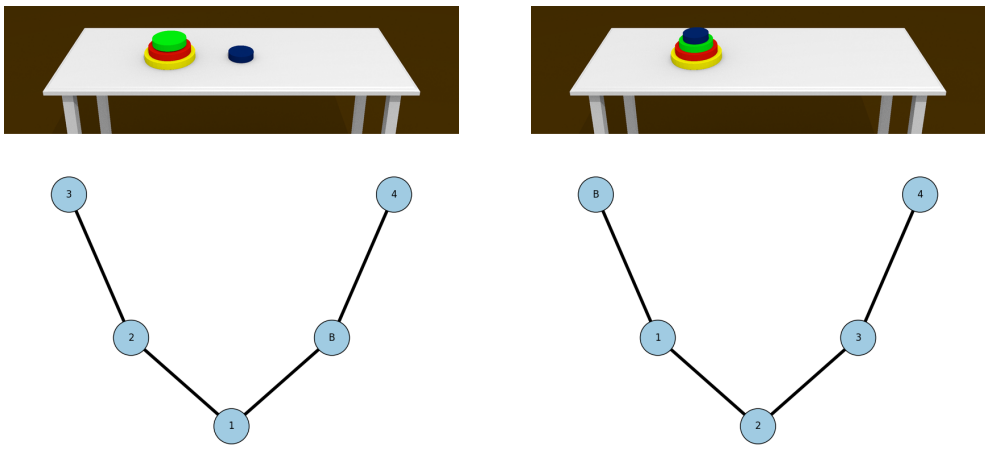
(f) Frame 549 and its Semantic Graph

Figure 4.5: Automatic Discovery of Game States in the *Towers of Hanoi* Animated dataset(contd.)



(a) Frame 589 and its Semantic Graph

(b) Frame 628 and its Semantic Graph



(c) Frame 687 and its Semantic Graph

(d) Frame 739 and its Semantic Graph

Figure 4.6: Automatic detection of game states in the *Towers of Hanoi* animated dataset

Label	Game Piece
B	Board
1	Yellow,largest disk
2	Red, 2nd largest disk
3	Green, 3rd largest disk
4	Blue, smallest disk

Table 4.1: Labels in the Semantic Graphs in Figs. 4.4, 4.5 and 4.6

For each of the game states, we run the *on* classifer to detect which objects are on which other objects. That gives us the following clauses, which serve as positive examples for the ILP system:

```
on(d,a).
on(d,b).
on(d,c).
on(c,b).
on(c,a).
on(b,a).

from(l1,[a,b,c,d],d,l2,[d]).
from(l1,[a,b,c],c,l3,[c]).
from(l2,[d],d,l3,[c,d]).
from(l1,[a,b],b,l2,[b]).
from(l3,[c,d],d,l1,[a,d]).
from(l3,[c],c,l2,[b,c]).
from(l1,[a,d],d,l2,[b,c,d]).
from(l1,[a],a,l3,[a]).
from(l2,[b,c,d],d,l3,[a,d]).
from(l2,[b,c],c,l1,[c]).
from(l3,[a,d],d,l1,[c,d]).
from(l2,[b],b,l3,[a,b]).
from(l1,[c,d],d,l2,[d]).
```

```
from(l1,[c],c,l3,[a,b,c]).
from(l2,[d],d,l3,[a,b,c,d]).
```

4.1.2 Real Dataset

The Towers of Hanoi is solved by a person while a Kinect records it in 1200 frames(Fig. 4.7). The results are similar to those obtained from the animated dataset(Figs. 4.8 and 4.9).

Figure 4.7: Objects found in a scene from the real Towers of Hanoi dataset

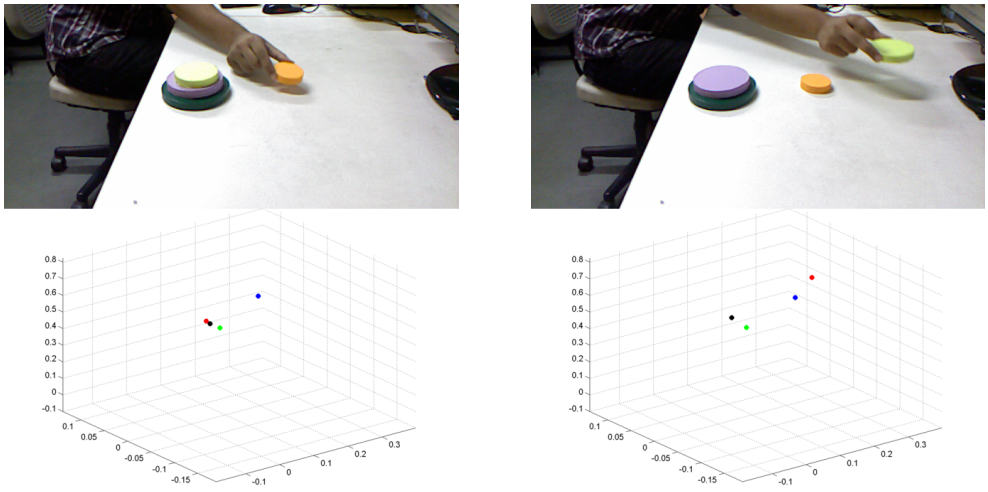
The hand is eliminated by using a HSV filter else it naturally occurs as an object that is tracked by our system. After doing so, we are able to plot the semantic graphs, detect game states and from the changes in the graph structure the clauses obtained are:

```
colour(e,green).
colour(f,violet).
colour(g,yellow).
colour(h,orange).
size(e,30).
size(f,28).
size(g,8).
size(h,2).
on(h,e).
on(h,f).
on(h,g).
```

```
on(g,e).
on(g,f).
on(f,e).
from(l1,[e,f,g,h],h,l2,[h]).
from(l1,[e,f,g],g,l3,[g]).
from(l2,[h],h,l3,[g,h]).
from(l1,[e,f],f,l2,[f]).
from(l3,[g,h],h,l1,[e,h]).
from(l3,[g],g,l2,[f,g]).
from(l1,[e,h],h,l2,[f,g,h]).
from(l1,[e],e,l3,[e]).
from(l2,[f,g,h],h,l3,[e,h]).
from(l2,[f,g],g,l1,[g]).
from(l3,[e,h],h,l1,[g,h]).
from(l2,[f],f,l3,[e,f]).
from(l1,[g,h],h,l2,[h]).
from(l1,[g],g,l3,[e,f,g]).
from(l2,[h],h,l3,[e,f,g,h]).
```

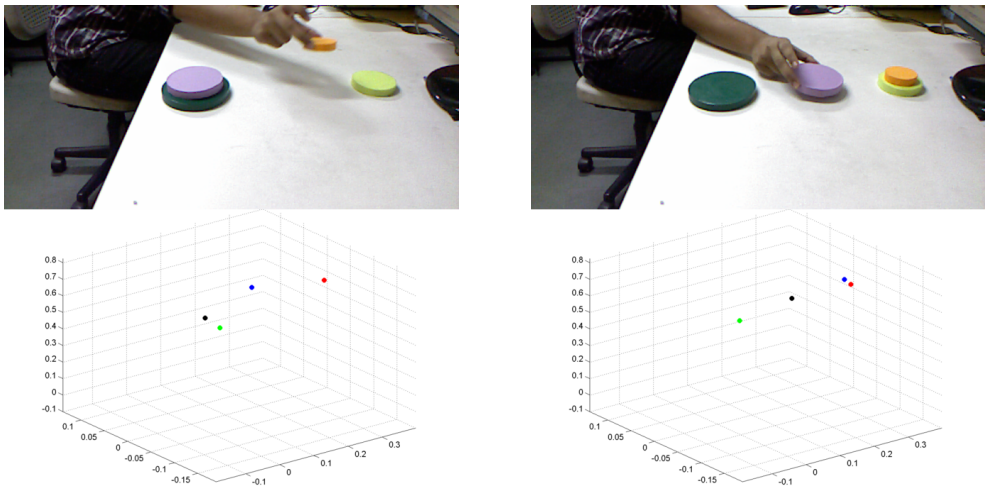
Label	Game Piece
B	Board
1	Violet,2nd smallest disk
2	Yellow, 3rd largest disk
3	Green, largest disk
4	Orange, smallest disk

Table 4.2: Labels in the Semantic Graphs in Fig. 4.9



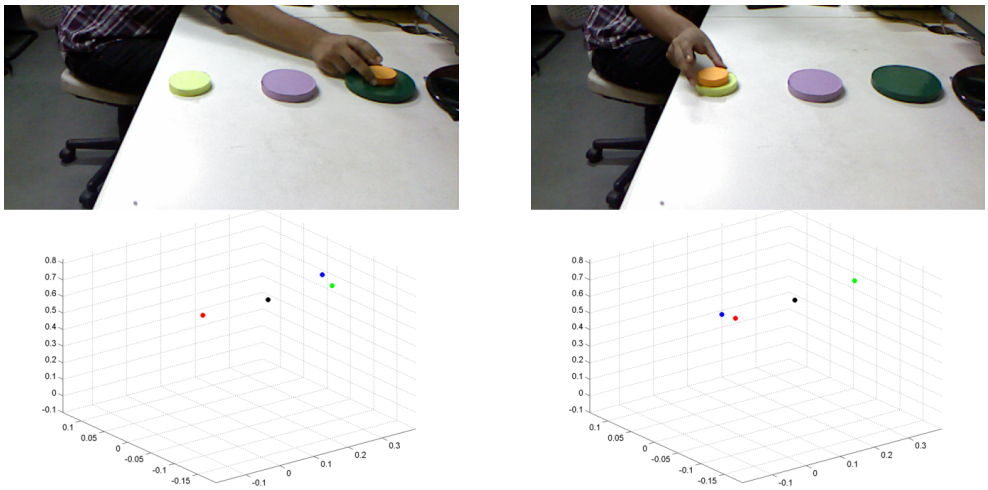
(a) Tracked Objects in Frame 1

(b) Tracked Objects in Frame 75



(c) Tracked Objects in Frame 150

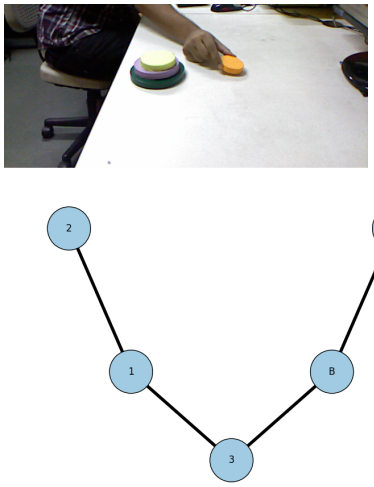
(d) Tracked Objects in Frame 250



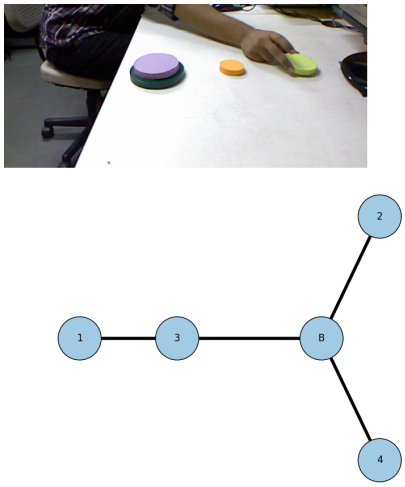
(e) Tracked Objects in Frame 750

(f) Tracked Objects in Frame 800

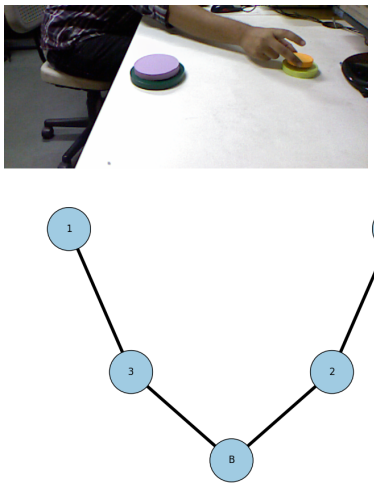
Figure 4.8: Tracking of game pieces in the *Towers of Hanoi* Real dataset



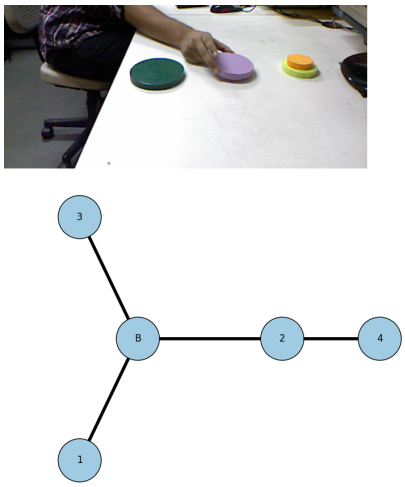
(a) Frame 4 and its Semantic Graph



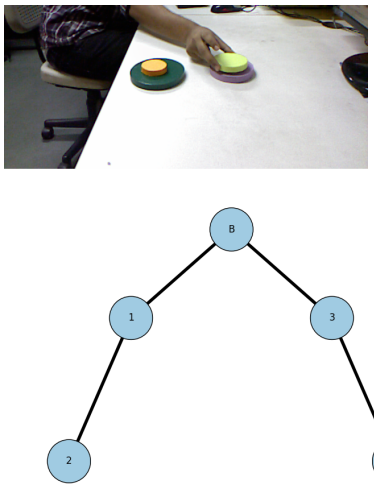
(b) Frame 83 and its Semantic Graph



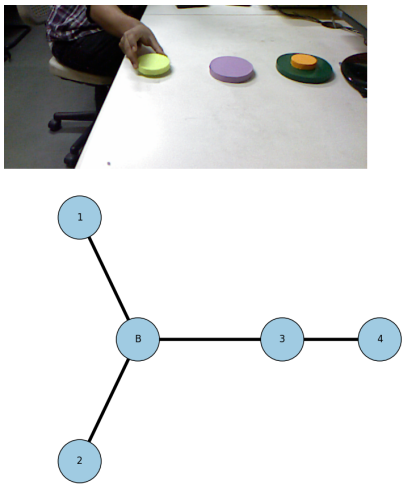
(c) Frame 173 and its Semantic Graph



(d) Frame 251 and its Semantic Graph



(e) Frame 421 and its Semantic Graph



(f) Frame 695 and its Semantic Graph

Figure 4.9: Automatic detection of game states in the *Towers of Hanoi* Real dataset

4.1.3 Rules Learnt by ILP

Clauses generated by observing both the animated and the real dataset are populated and provided as input to the ILP system. It attempts to generalize for the two kinds of clauses we have *on* and *move*. There is no simple logical generalization for *move* in the *Towers of Hanoi* and the only simple rule that is derived is for the clause *on*:

```
on(A,B) :- greatersize(B,A).
fromto(A,B,C,D,E) :- top(C,B), top(C,E).
```

The first rule is the equivalent of Rule 3 which states that "No disk may be placed on top of a smaller disk." The second rule says that piece C that is moves from any position A to any position D is at the top of the stack B and stack E in other words the piece that is moved must be from the top of one stack and also end up at the top of the other.

4.2 1D Peg Solitaire

This is another spatial reconfiguration puzzle. The game setup can be see in Fig. 4.10. There are five cylindrical holes in a row, two red pegs and two blue pegs. In the beginning(Fig. 4.11), the red pegs are placed in the two left holes and the blue pegs are placed in the two right holes leaving a hole in between that is empty. The goal of the game is to swap the positions of the red pegs with the blue pegs according to the following rules:

1. A red peg can step one position to the right into the empty space or jump two positions to the right into the empty space.
2. A blue peg can step one position to the left into the empty space or jump two positions to the left into the empty space.

4.2.1 Animated Dataset

BlenSor was used to animate four pegs - two red ones and two blue ones. The animation shows the successful solving of a 1D Peg Solitaire. It consists of 560 frames of 640 $\times$ 480 RGBD images. We expect to identify the nine game states and the underlying rules.

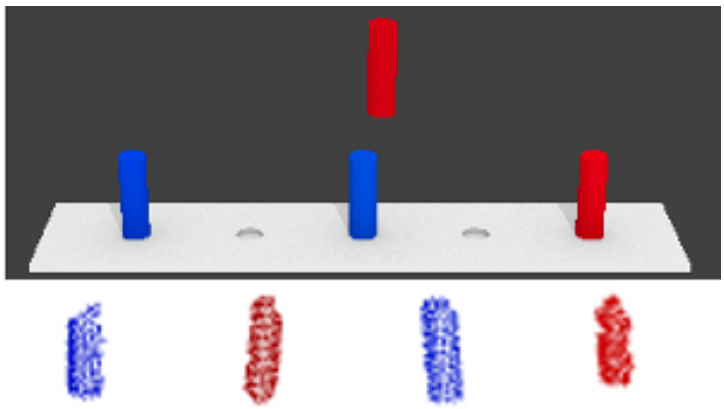
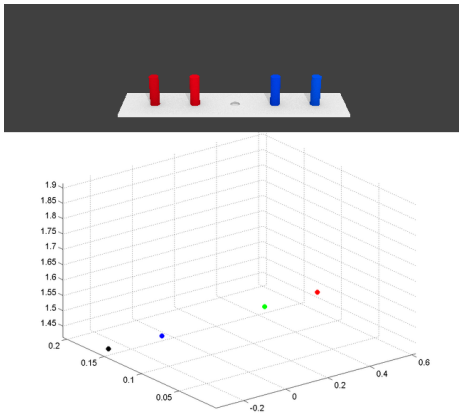
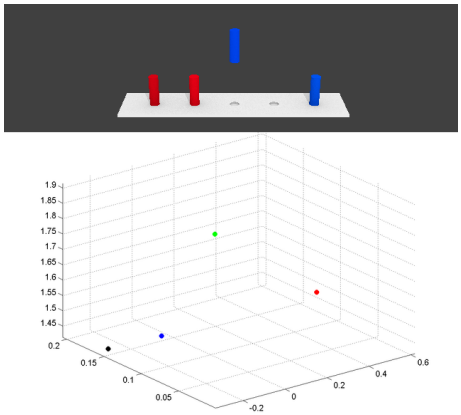


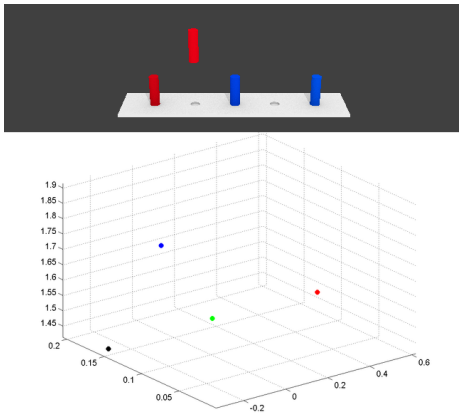
Figure 4.10: Objects found in a scene from animated 1D Peg Solitaire dataset



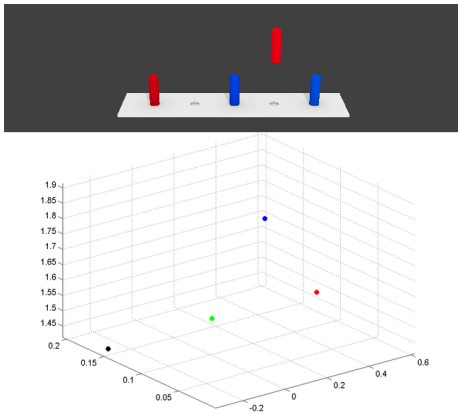
(a) Tracked Objects in Frame 1



(b) Tracked Objects in Frame 40

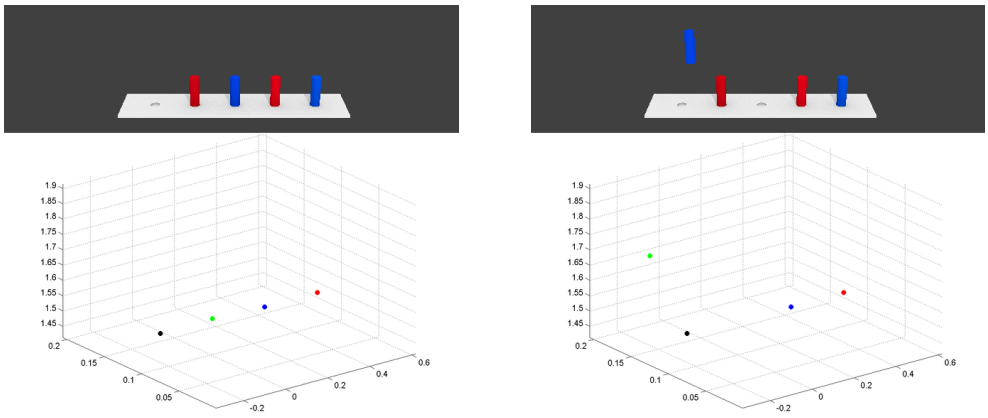


(c) Tracked Objects in Frame 80



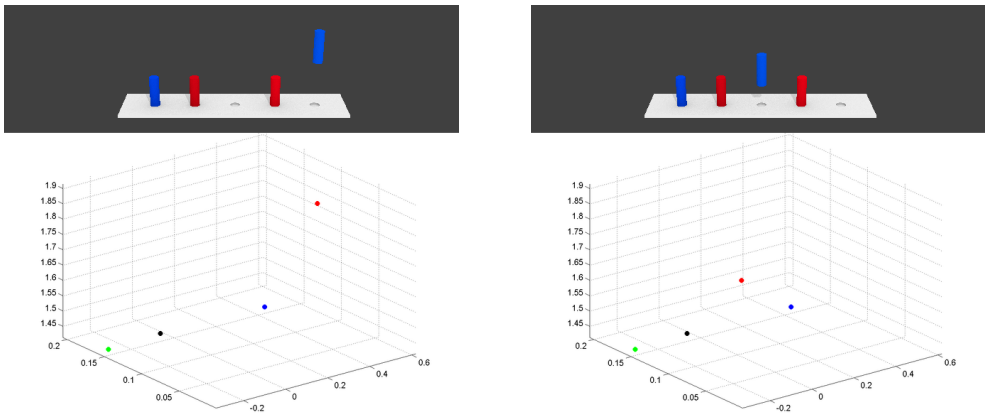
(d) Tracked Objects in Frame 120

Figure 4.11: Tracking of game pieces in the *1D Peg Solitaire* Animated dataset(contd.)



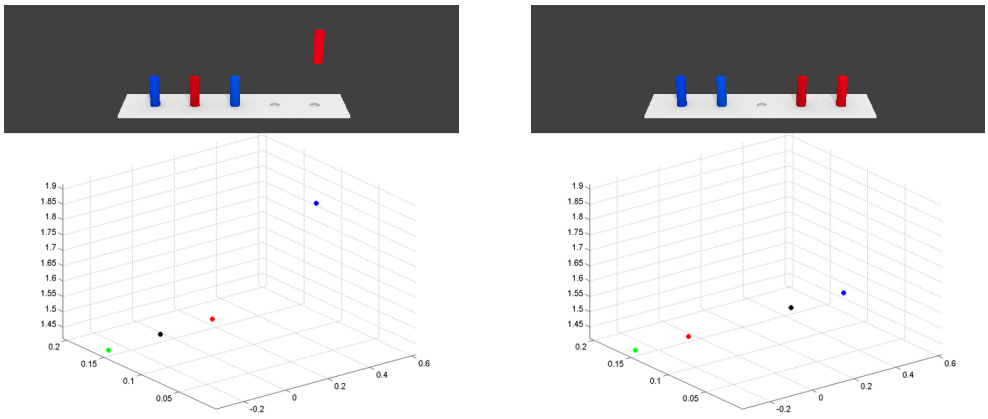
(a) Tracked Objects in Frame 200

(b) Tracked Objects in Frame 250



(c) Tracked Objects in Frame 300

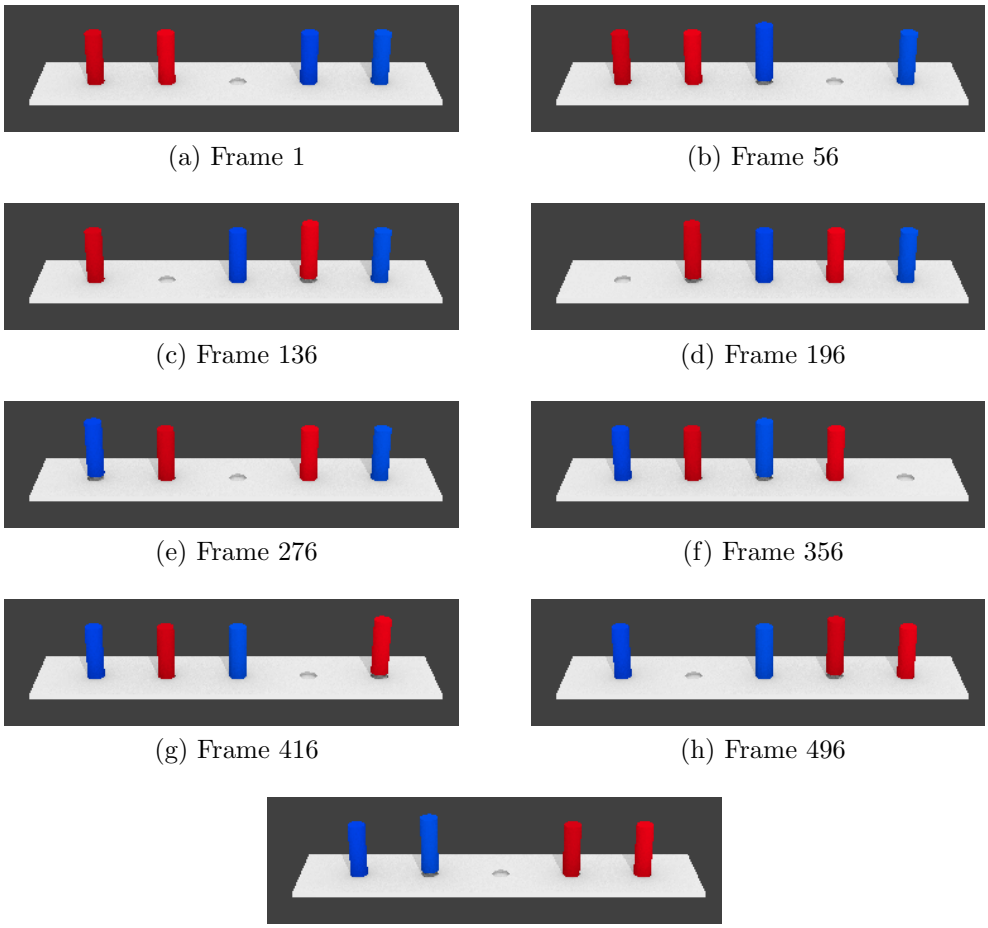
(d) Tracked Objects in Frame 350



(e) Tracked Objects in Frame 400

(f) Tracked Objects in Frame 560

Figure 4.12: Tracking of game pieces in the *1D Peg Solitaire* Animated dataset



(a) Frame 1 (b) Frame 56

(c) Frame 136 (d) Frame 196

(e) Frame 276 (f) Frame 356

(g) Frame 416 (h) Frame 496

(i) Frame 556

Figure 4.13: Automatic detection of game states in *1D Peg Solitaire* dataset

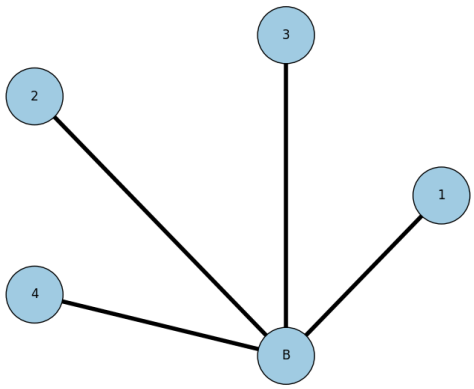


Figure 4.14: Semantic Graph of all the states in 1D Peg Solitaire

Label	Game Piece
B	Board
1	Red Peg 1
2	Red Peg 2
3	Blue Peg 1
4	Blue Peg 2

Table 4.3: Labels in the Semantic Graphs in Fig. 4.14

Figs. 4.11 and 4.12 show the results of octree based tracking of the game pieces. Based on that, the semantic graphs help us to identify the 9 states from 560 frames(Fig. 4.13 and Fig. 4.14). The semantic graph however is the same for all the game states. It is the meta-information contained in the nodes of the graph that changes. The following clauses are generated:

```
x(l1).
x(l2).
x(l3).
x(l4).
x(l5).
colour(p1,blue).
colour(p2,blue).
colour(p3,red).
colour(p4,red).
move(p1,l1,l2).
move(p1,l1,l3).
move(p1,l2,l4).
move(p1,l2,l3).
move(p1,l3,l4).
move(p1,l3,l5).
move(p1,l4,l5).
move(p2,l2,l3).
move(p2,l2,l4).
```

```
move(p2,l3,l4).
move(p2,l3,l5).
move(p2,l4,l5).
move(p3,l4,l3).
move(p3,l4,l2).
move(p3,l3,l1).
move(p3,l3,l2).
move(p3,l2,l1).
move(p4,l2,l1).
move(p4,l3,l1).
move(p4,l3,l2).
move(p4,l4,l3).
move(p4,l4,l2).
move(p4,l5,l3).
move(p4,l5,l4).
```

4.2.2 Rules Learnt by ILP

PROGOL generalizes for the clause *move* and comes up with four rules:

```
move(A,B,C) :- diff(B,C,-2), color(A,blue).
move(A,B,C) :- diff(B,C,-1), color(A,blue).
move(A,B,C) :- diff(B,C,1), color(A,red).
move(A,B,C) :- diff(B,C,2), color(A,red).
```

We learn that if there is an object that moves right its colour must be red and if there is one which moves left then its colour must be blue. More interestingly, the system discovers that there are two types of moves a piece is able to do that is one step and one jump which implies moving two steps at the same time.

4.3 Complex rule-based games

Both the games for which we learnt rules generalized a clause in terms of only one other relation from their background knowledge. For example, in *Towers of Hanoi* the rule was learn in terms of the clause *on* and in *1D Peg Solitaire* the rule was learnt in terms of the clause *colour*. To test how capable the ILP system was, we modified the Towers of Hanoi game to now play with an added rule that a smaller red coloured piece can only go on top of a larger green coloured piece. To possibly confound the system more information regarding shape was also provided. The ILP system gave following rule for that game:

```
on(A,B) :- colour(B,red),colour(A,green),greatersize(B,A).
```

In the previous chapter, we demonstrated how the squares of a grid-based game can be discovered and the positions of the poles in Towers of Hanoi and the holes in 1D Peg Solitaire were discovered. In games like Chess and Checkers, after prolonged visual observation the system will be able to find all the squares in the grid. With additional concepts like multiple players, turns in a game and more relations like *colinear* and *neighbour*, more complex game rules can be discovered. Models for classification of different shape of game pieces also need to be learnt. The ILP system has been proven to be really good at learning the rules of different pieces in chess if enough data is provided to it. Some of the rules that PROGOL has induced about chess pieces are:

```
move(bishop,pos(A,B),pos(C,D)) :- xdiff(B,D,E), ydiff(A,C,E).
move(knight,pos(A,B),pos(C,D)) :- xdiff(B,D,1), ydiff(A,C,2).
move(knight,pos(A,B),pos(C,D)) :- xdiff(B,D,2), ydiff(A,C,1).
```

Above *xdiff* represents the absolute difference in coordinates in x direction while *ydiff* is the absolute difference in coordinates in y direction. For the bishop it is able to induce that the change in x direction must equal to the change in y direction. The rule for a valid move of the knight also has been learnt successfully.

The framework will only need enhancement in what has been termed as background knowledge in ILP terminology to be able to handle games of more complexity.

Chapter 5

Conclusions and Discussion

We presented a framework that is able to represent RGBD scenes succinctly in the form of dynamic semantic graphs. Following which, we demonstrated its application in learning the rules of game and puzzles.

Our entire pipeline is modular. This allows for the use of better segmentation and tracking techniques in the future. As pointed out earlier, the major roadblock in using semantic graphs is the object monitoring process. Better segmentation methods like supervoxel-based clustering or random-walk segmentation of point clouds will enable the system to discover textured objects as well. This has already enabled better manipulation recognition in kitchen scenes. Certain assumptions of rigid single-coloured game pieces were taken during implementation. These can be relaxed with more advances in the perceptual system. Our current implementation of multi-object tracking needs to be more robust to handle occlusions that might occur when a game is being played with more game pieces. A persistent model of the game board and all pieces on it is needed which keeps track of objects even if they are hidden due to manipulations being performed by the hand or by game pieces in front of it. Recent research[21][29] on multi-object tracking has shown encouraging results which will be helpful in tracking in games with more game pieces. A saliency-based segmentation needs to be developed which estimates where the game is being played in a point cloud and filters out other parts in the background which are not useful to the learning of the rules.

As of now, we can successfully induce some of the rules of the game being played. However, to enable the system to actually start playing it needs to be equipped with many other concepts like that of turns in 2 or more player games and goal state recognition and generalization. There needs to be a module which comes up with useful representations for the game states that have been visually discovered. To have generalization for the start and end states we need a labeled set or use a heuristic for when a game has ended. In fact, this is the one question people ask explicitly when learning to play a game, even if they are able to induce the rules of the game perfectly. If the start and goal states are presented, we could use a logic based language to play the game. All the knowledge our system has about a game can be converted to Game Description Language(GDL) which will enable the system to start playing these games against humans too.

5.1 Future Work

We believe some ideas presented in this thesis will be relevant in the following areas:

5.1.1 Spatial Assembly

A robust 6-DoF 3d multiple object tracker will open up new areas of research. Spatial assembly problems like the 2D T Puzzle(Fig. 5.1) and the 3D Soma cube(Fig. 5.2) require tracking of not only the position of game pieces but also their relative orientation to each other. General rules of how two game pieces fit into each other can be easily made by looking at the semantic graph for the relation *contact*. These rules can be used to generate new shapes by a different assembly of the same set of pieces. Such a framework will be helpful in robotic therapy where an embodied agent that is able to identify how two game pieces get connected, can transfer this knowledge to children who have difficulty in doing so. A robot can discover certain common constructions in assembly toys like the Lincoln Logs or Lego blocks. These mini-assemblies can be hierarchically combined to form bigger assemblies.

The other obvious application would be a robot who is able to learn to assemble

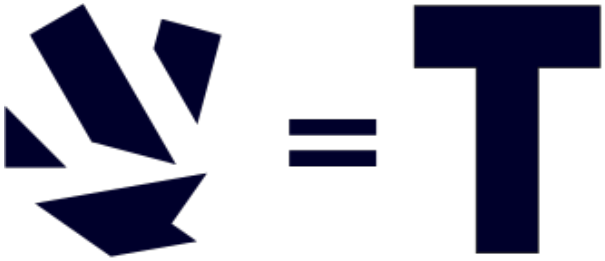


Figure 5.1: T-Puzzle Taken from http://www.wikipedia.org

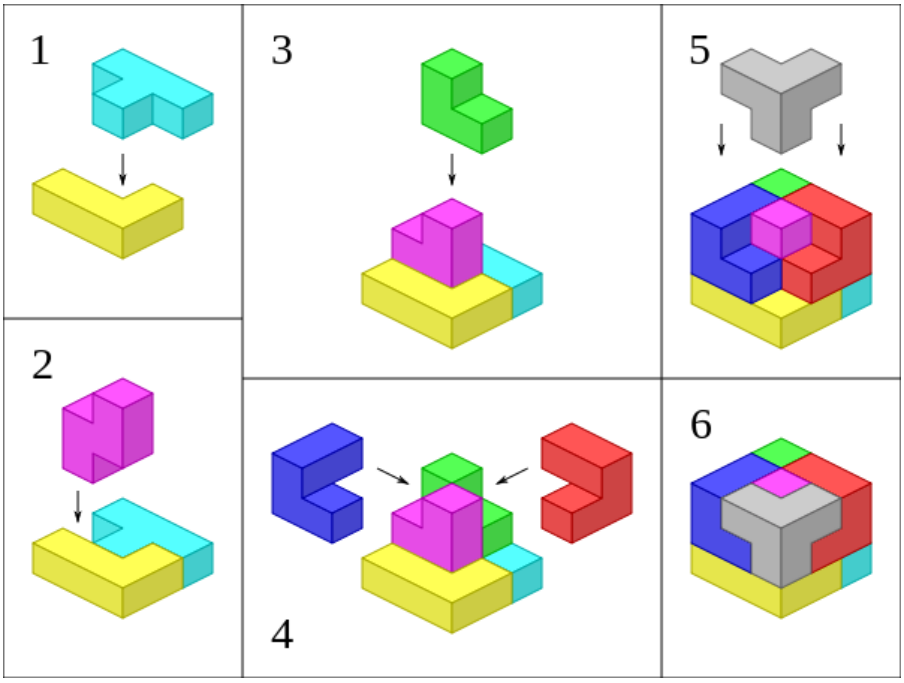


Figure 5.2: Soma Cube Assembly Taken from http://www.wikipedia.org

bigger machines using smaller parts after having observed people perform that assembly. One of the major obstacles in such systems is the tracking of multiple objects in 3D under occlusion. During manipulations, people might rotate the assembly or occlude it unknowingly. A persistent model of the world will help in keeping track of objects that are hidden from sight.

5.1.2 Object Affordances

The concept of object affordance[16] is very important in robotics. Given an action, a robot should be able to search in its environment for objects that would help him carry

that out. Sometimes the required object might not be present in its reach. It would be very exciting if the robot is able to still complete his action using another object with similar attributes. This work will involve manipulation recognition from semantic graphs and classification of objects based on their attributes. Consider an object like a scissor which has the attribute of being sharp. It gets associated with manipulation consequence divide which in terms of the semantic graph implies that a node has divided into two nodes in the next frame. From observing people, the robot deduces the presence of an active object say a pair of scissors and discovers that the probability of the consequence of a cut or a split happening is high. Next time, the robot might not be able to find the scissor but makes do something that is sharp to complete the action. This might fail a lot of the time because the consequence of divide is not the same across object categories. A scissor might divide a piece of paper into two but might fail to cut a cucumber into two pieces. Also affordances of certain objects depend on their orientation. For example, a wooden box with its open side facing up can *contain* other objects in itself. But if its open side is facing down, it can now provide *support* to other objects. There are other challenges regarding how a robot will carry out that action and how much supervision would be required to teach manipulations and affordances to it. That makes this a problem worthy of more research.

5.1.3 Grounding Natural Language in Semantic Graphs

Semantic graphs encode higher order manipulation consequence primitives like merge, divide, translate and rotate. More consequences can be created if the relationship encoded in the edges is different. Natural language descriptions of such scenes can be collected. Nouns, adjectives and verbs can then be grounded in various parts of the semantic graphs. The nouns are the nodes of the graph and the adjectives can be encoded as outputs of attribute classifiers run on the nodes. The verbs can be encoded as changes in the graph structure of the scene. For example, the verb *cut* will be correlated highly with the event of the number of nodes in the graph increasing by one.

5.1.4 The General Game Learning and Playing Machine

Learning new games comes naturally to people. By mere observation, they induce the rules of the game, form a mental representation of the game states, memorize the start and goal states and develop heuristics to win. As discussed previously, the representation space for each game is slightly different. We believe real intelligence lies in actually coming up with the relations like *on* and *move* to look for positive examples during game-play. While these classifiers can be trained on more data to be more accurate, a general game learning framework would involve coming up with these relations using which the rules of any game can be induced. Relations like *colinear* are useful in games like Tic-Tac-Toe and Chess. It would be interesting to either come up with a list of relations that can model many games or have th ability to learn classifiers for relations and background knowledge on the go.

Another interesting challenge is to come up with efficient representations for different games using the same framework. For example, in the 1D peg solitaire the rules are discovered only in terms of the *colour* relation. Therefore, it might hint that an efficient representation of the game state might also be in terms of the colour of the game pieces. This representation will allow us to formulate the start and end states better. Game state updates will be performed according to the rule that has been induced. In short, the machine can now play a game by observation and not by being programmed to do so. This can be integrated with a General Game Playing system to actually play games.

Successful implementation of the systems described above will result in a General Game Learning and Playing machine. This system has immense application in the context of human-robot interaction. Children who find it difficult to make mental models of games can be effectively tutored by these agents. More importantly, the therapists can program the robot to learn new games without having to study AI or know programming.

References

[1] Motion-based multiple object tracking. http://www.mathworks.in/help/vision/examples/motion-based-multiple-object-tracking.html.

[2] Octree documentation pcl. http://docs.pointclouds.org/1.7.0/group__octree.html.

[3] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Susstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(11):2274–2282, 2012.

[4] EE Aksoy, M Tamosiunaite, R Vuga, A Ude, C Geib, M Steedman, and F Wörgötter. Structural bootstrapping at the sensorimotor level for the fast acquisition of action knowledge for cognitive robots. 2013.

[5] Eren Erdal Aksoy, Alexey Abramov, Johannes Dörr, Kejun Ning, Babette Dellen, and Florentin Wörgötter. Learning the semantics of object–action relations by observation. *The International Journal of Robotics Research*, 30(10):1229–1249, 2011.

[6] Andrei Barbu, Siddharth Narayanaswamy, and Jeffrey Mark Siskind. Learning physically-instantiated game play through visual observation. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1879–1886. IEEE, 2010.

[7] Yngvi Björnsson. Learning rules of simplified boardgames by observing. In *ECAI*, pages 175–180, 2012.

[8] Alvaro Collet, Siddhartha S Srinivasa, and Martial Hebert. Structure discovery in multi-modal data: a region-based approach. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5695–5702. IEEE, 2011.

[9] Irving M Copi, Carl Cohen, and Daniel E Flage. *Essentials of logic*. Pearson/Prentice Hall, 2007.

[10] Neil Dantam, Irfan Essa, and Mike Stilman. Linguistic transfer of human assembly tasks to robots. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 237–242. IEEE, 2012.

[11] Luc De Raedt and Kristian Kersting. Probabilistic inductive logic programming. In *Algorithmic Learning Theory*, pages 19–36. Springer, 2004.

[12] Vincent Delaitre, David F Fouhey, Ivan Laptev, Josef Sivic, Abhinav Gupta, and Alexei A Efros. Scene semantics from long-term observation of people. *Computer Vision–ECCV 2012*, pages 284–298, 2012.

[13] Babette Dellen, Eren Erdal Aksoy, and Florentin Wörgötter. Segment tracking via a spatiotemporal linking process including feedback stabilization in an nd lattice model. *Sensors*, 9(11):9355–9379, 2009.

[14] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.

[15] Michael Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the aaai competition. *AI magazine*, 26(2):62, 2005.

[16] JJ Gibson. The concept of affordances. *Perceiving, acting, and knowing*, pages 67–82, 1977.

[17] Michael Gschwandtner, Roland Kwitt, Andreas Uhl, and Wolfgang Pree. BlenSor: Blender Sensor Simulation Toolbox Advances in Visual Computing. volume 6939 of *Lecture Notes in Computer Science*, chapter 20, pages 199–208. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2011.

[18] Shyamanta M Hazarika and Alexy Bhowmick. Learning rules of a card game from video. *Artificial Intelligence Review*, 38(1):55–65, 2012.

[19] Łukasz Kaiser. Learning games from videos guided by descriptive complexity. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[20] Seongyong Koo, Dongheui Lee, and Dong-Soo Kwon. Gmm-based 3d object representation and robust tracking in unconstructed dynamic environments. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1114–1121. IEEE, 2013.

[21] Seongyong Koo, Dongheui Lee, and Dong-Soo Kwon. Multiple object tracking using an rgb-d camera by hierarchical spatiotemporal data association. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1113–1118. IEEE, 2013.

[22] Hema Swetha Koppula, Rudhir Gupta, and Ashutosh Saxena. Learning human activities and object affordances from rgb-d videos. *The International Journal of Robotics Research*, 32(8):951–970, 2013.

[23] Ivan Laptev. On space-time interest points. *International Journal of Computer Vision*, 64(2-3):107–123, 2005.

[24] Donald Meagher. Geometric modeling using octree encoding. *Computer graphics and image processing*, 19(2):129–147, 1982.

[25] Stephen Muggleton. Inverse entailment and progol. *New generation computing*, 13(3-4):245–286, 1995.

[26] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679, 1994.

[27] Iason Oikonomidis, Nikolaos Kyriazis, and Antonis A Argyros. Efficient model-based 3d tracking of hand articulations using kinect. In *BMVC*, pages 1–11, 2011.

[28] Jeremie Papon, Alexey Abramov, Markus Schoeler, and Florentin Worgotter. Voxel cloud connectivity segmentation-supervoxels for point clouds. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 2027–2034. IEEE, 2013.

[29] Jeremie Papon, Tomas Kulvicius, Eren Erdal Aksoy, and Florentin Worgotter. Point cloud video object segmentation using a persistent supervoxel world-model. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 3712–3718. IEEE, 2013.

[30] Radu Bogdan Rusu. *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. PhD thesis, Computer Science department, Technische Universitaet Muenchen, Germany, October 2009.

[31] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE, 2011.

[32] Paulo Santos, Simon Colton, and Derek Magee. Predictive and descriptive approaches to learning game rules from vision data. In *Advances in Artificial Intelligence-IBERAMIA-SBIA 2006*, pages 349–359. Springer, 2006.

[33] Brian Scassellati. How social robots will help us to diagnose, treat, and understand autism. In *Robotics research*, pages 552–563. Springer, 2007.

[34] Ashwin Srinivasan. The aleph manual. *Machine Learning at the Computing Laboratory, Oxford University*, 2001.

[35] Ashwin Srinivasan, Stephen H Muggleton, Michael JE Sternberg, and Ross D King. Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence*, 85(1):277–299, 1996.

[36] Joshua Wainer, David J Feil-Seifer, Dylan A Shell, and Maja J Mataric. The role of physical embodiment in human-robot interaction. In *Robot and Human Interactive Communication, 2006. ROMAN 2006. The 15th IEEE International Symposium on*, pages 117–122. IEEE, 2006.

[37] Yezhou Yang, Cornelia Fermuller, and Yiannis Aloimonos. Detection of manipulation action consequences (mac). In *CVPR 2013*, 2013.