Learning to Perform Actions from Observations with Sequential Modeling and Reinforcement Learning

Ph.D. Dissertation Proposal

Brandeis University
Department of Computer Science
Dr. James Pustejovsky, Advisor

Committee Members:
Dr. Anthony G. Cohn (School of Computing, University of Leeds)
Dr. Pengyu Hong (Dept. of Computer Science, Brandeis University)
Dr. Timothy J. Hickey (Dept. of Computer Science, Brandeis University)

Submitted by Tuan Do
tuandn@brandeis.edu

January, 2018

ABSTRACT

Learning from (human) demonstration studies how computational and robotic agents can learn to perform complex task by observing humans. This closely resembles the way humans learn. For example, children can imitate adults in a variety of household tasks, such as pouring milk from a carton to a cup or cleaning a table, and after one or a few observation, they can replicate the action with relative accuracy. This research would test feasibility of a virtual agent that can process multimodal (linguistic and visual) captures of actions and learn to reenact them in a simulated environment. In particular, this thesis would focus on a small set of actions involving spatial primitives. The system is composed of the following components: a. an event capture annotation tool that captures human interaction with objects using Kinect sensor; b. an event representation learning method using recurrent neural network with QSR feature extraction; c. an action reenacting algorithm using reinforcement learning; d. an evaluation method using simulation tool VoxSim (based on Unity game engine) and Amazon Mechanical Turk to crowdsource judgments. .

List of Figures

List of Algorithms

Chapter 1

Introduction

The community surrounding "learning from (human) observation" (LfO) studies how computational and robotic agents can learn to perform complex task by observing humans (Young and Hawes, 2015). Work in this area can be traced back to reinforcement learning studies by Smart and Kaelbling (2002) or Asada et al. (1999), and closely resembles the way humans learn. Infants can imitate adults in a variety of household tasks, such as *pouring milk from a carton to a cup* or *cleaning a table*, by replicating actions after one or a few observations. This is in part an AI movement that aims to achieve *stronger* AI, which is AI systems that can adapt to new environment, to learn new skills and can cooperate with humans to perform tasks.

So far the development toward more adaptable AI has faced with many obstacles, leading to the fact that most robots developed in the previous decades have shipped with preinstalled programs, limited to a set of predefined functionalities. Learning approaches in the robotics community seek to move toward smarter and more adaptable robots, for, among others, the following reasons:

- Consumer desire for mobile or household assistant robots that can perform multiple tasks with flexible apparatus, such as multiple grasping arms (Bogue, 2017). Robots with behavioural robustness can learn from a wider range of experiences by running in a dynamic human environments (Hawes et al., 2017)
- Advances in deep learning have afforded robotic agents a high-level understanding of embedded semantics in multiple modalities, including language, gesture,

object recognition, and navigation. This increases the circumstances and modalities available for robotic learning.

- Robots that can communicate and cooperate with humans to perform tasks are desirable, as fully automated robots are far from able to perform complex tasks in novel environments or circumstances. Human can only communicate using qualitative semantics, such as ”put A closer to B“, while robots function in a lower quantitative level of semantics, such as ”put(A, coordination_XYZ)“. Understanding the nuances in human languages requires communicative robotic agents to learn various human concepts, such as the predicates ”put” or ”closer to”, in a form of action programs that are compositional with arguments.

For robots to achieve this level of autonomy and understandings, a scaffold of robot learning can be summarized as followings:

- Robots learn object forms (shape, color, size etc) and their mappings to human languages (round, square; red, green; big, small) (Alomari et al., 2017). Moreover, robots also need to learn relative relations among objects (such as whether one object could contain another object (Liang et al., 2015)) and between agents and objects (what is generally called object affordances, a.k.a what a human or robotic agent can do with the object (grasping, pouring) (Koppula et al., 2013)).
- Robots learn to perform simple actions by learning compositional programs, such as to learn programmatic form of action verbs (put, slide, push, roll etc), in addition to programmatic form of motion adjunctions specifying path of motion (on, in, next to etc.).
- Robots learn to perform more complex actions (actions that are generally represented by structured programs, such as procedural or repetitive programs), such as “make a row from given objects”.
- Robots learn to communicate with humans in a feed-back loop to make necessary correction, or to improve learned models. While human experts can give instructions or demonstrations, robots might not be able to recognize immediately very abstract concepts that are intended by experts. For example, with the same learned action as before, a smart robotic learner can recognize that the intended action is a repetitive programs of “move A next to B”, but it would be hard to recognize “direction of extension need to follow the longest axis”.

While the discussion so far has been about robots, it applies to virtual agents as well. Moreover, as most of the learning process could be simulated, this dissertation work will focus on learning capacity of a virtual agent that is co-situated with humans and participate in some collaborative tasks. The virtual agent can observe what the humans have done, perform action in a novel situation, and receive feedback from human partners.

Because of this dissertation scope, the focus of my work will be on the capacity of virtual agents to learn to perform action in regard to a set of spatial primitives that functioning as adjunctions specifying path of motions. The result of this dissertation work would be a feasibility study of a multi-step framework combining various components, including a module to learn action representation from video captures, one to generate different simulations of the virtual agents performing the action, and one to visualize those simulations for evaluation.

1.1 Background

1.1.1 Communicating with Computers

As a general term, communicating with computers can be understood as human-computer interface (HCI), in which there is an exchange of information between human and computers. Communicating with Computers (CwC) program, a DARPA project in specific, advance HCI to take into account multiple modalities, namedly vision, language and gestures. Moreover, it would enable humans to share complex concepts with computers, building upon more elemetary concepts. Quoting the program's statement, it *focuses on developing technology for assembling complex ideas from elementary ones given language and context*.

More importantly, for computers to be able to understand complex concepts from elementary concepts, it is necessary that we have to include **learning from demonstration** as one modality of communication. Not only that this kind of learning allows an agent to learn new concepts through interaction, incremental learning (which I would explain shortly) allows humans to provide immediate feedback to the computers to change its execution programs.

1.1.2 Learning from demonstration

Learning from demonstration (LfD) can be traced back to 1980s, in the form of automatic robot programming. The earliest form might be called *teaching by showing* or *guiding*, in which a robot's effectors could be moved to desired positions, and robot's controlling system would record their coordinations and rotations (controlling signals) so that when the robot is asked to execute the program, the controller would move its effectors according to the recorded controlling signals. This mechanism, though rudimentary, has proved successful with little memory or computational power in the eras before the popularity of general-purpose computers. It is, however, simply *record and replay* method, which is agnostic to the environment (Lozano-Perez, 1983).

This simple *guiding* framework has developed into robot Programming from demonstration (PbD) framework, which has several development in recent time. In basic, it still applies the same principles of using sensorimotor information of robot's joints to learn models of actions. However, it has been able to generalize from just *record and play* method, by allowing interpolation for novel configurations or new environments. Calinon et al. (2007) is an exemplary research for this framework, in which a teacher guides a humanoid robot's hands to move a chess piece forward through kinesthetics. Recently, the field starts to pick up newer machine learning tools such as Artificial Neural Networks (ANNs), Radial-Basis Function Networks (RBFs) etc. Moreover, the field progressively expands with other modalities of teaching-learning interface, such as vision, haptics etc.. A representative work that shows this development is the thesis of He (2017), in which robots observe multiple object manipulation (in-hand rotation of objects) and process them into visual and haptic features, and learning action policy by applying RBFs network over these feature vectors.

Another branch of the original line of research is its interdisciplinary variance, named **imitation learning**. Evidence from biology, neuro-science, cognitive sciences and human-computer interaction (HCI), has been used to guide its researching methods. For instance, neuroscientific research showed that primate imitation learning requires a conceptual model of object spatial and relative locations, resolved from visual sensory information (Maunsell and Van Essen, 1983). HCI contributes a framework for incremental learning, which makes interaction between human and computers an important component in the learning process. This effectively speeds up the learning rate, corrects learning errors, or guides learning agent to focus on certain part of learning. In

particular, deixis cues (pointing or gazing) could be used to limit the start and end of a demonstration, or to constraint the objects in an action (Calinon and Billard, 2006).

On the terminology used in this proposal, I will use the general term Learning from demonstration (LfD), following discussions in Argall et al. (2009). Programming by Demonstration will not be used because this term normally applies to robot PbD, and in this proposal, all experiments would be carried out in simulated environment (with embodied agent for visualization). Also following Argall et al's survey, this research proposal could be categorized as *imitation with external observation* approach, which means the **teacher execution** (motion of human body) could not make an Identity mapping to the **recorded execution** (input from 3-D sensors), and **recorded execution** could not make an Identity mapping to the **learner execution** (the agent's planning policy). However, this subcategorization system is not widely adopted by other authors, such as Billard et al. (2008). In fact, Billard et. al refers to the *imitation* as the reenacting part of the whole system, in contrast to *demonstration*. Without making more confusion, I would resort to use the most accepted term that could be found in the literature.

In the following two subsections, I will describe two machine learning techniques that I will use: temporal-sequential modeling learns model of the skills from demonstrations, and searching/reinforcement learning methods to reproduce the learned skill in a new context.

1.1.3 Temporal-sequential modeling

In basic, temporal-sequential models are any algorithmic, machine learning or logical models that allow us to represent changing of system states over time. There are many different types of models belonging to this category, with a wide variety of application, including classification (Do and Pustejovsky, 2017a), text/dialog generation (Serban et al., 2017) etc. Some popular sequential modeling systems are: finite-state machine or finite automaton (FSM) in either deterministic or non-deterministic form; its development with more explanatory power, Hidden Markov Model (HMM); interval temporal logic. More recently, development of neural methods has brought about several sequential neural network models, such as Recurrent neural networks (RNNs), with its flavors such as Long short term memory (LSTMs) or Gated recurrent unit (GRU).

Arguably, predictive sequential learning model is the most appropriate model for learning of trajectory level skills. What the learners observe in each action demonstration is a frame-by-frame sequence of feature vectors, and a predictive model could be used to produce different label of the sequence: which category of action it is, whether it has been terminated or not etc. A feed-forward network such as LSTM can be aptly trained to fit high level of complexity patterns in the feature vectors. In particular, I will use LSTM model to learn a function called *progress function*, that guides the agent to traverse through a good trajectory.

On the other hand, interval temporal logic paired with a graph-based or hierarchical approach, such as Dubba et al. (2015), provides a framework to describe more complex actions as compositions of elementary actions. Again, I will use the *progress function* to test whether the part-whole relationship between complex and elementary actions can be automatically recognized.

Moreover, the same predictive model could be used as a classifier to make distinction between different types of actions (or types of spatial primitives). Cognitively speaking, we, as humans, might learn each spatial primitive in action individually, but for machine learning methods, classifiers might help to boost learning performance. To test whether inclusion of a categorical classifier is necessary, I will compare performance of my system when models for each individual spatial primitive are trained independently, and when they are trained to make distinction against each other.

1.1.4 Reinforcement learning

Reinforcement learning (RL) is a set of methodologies used for robotic or virtual agents to learn by interacting with environment, trying to find an action policy to optimize a target value function while exploring the action search space. In this work, I will explore some combination of searching methods and reinforcement learning methods, especially ones belonging to Monte Carlo family.

A general formalization of reinforcement learning includes the following components: a *policy* dictates actions of agents given current state of the environment (*state* or *observation*), a *reward signal* dictates the goal of RL problem in a form of a single reward value returned from the environment after each action is performed, and a *value function* is the accumulated rewards over long run. In this problem, *state* refers to a feature vector

that describes the current configuration of simulated environment, which I would build upon qualitative spatial features between objects, *action* refers to movement of blocks (and which block to move) and *value function* refers to the aforementioned progress function.

A particular appealing framework that can be applied in learning policy in continuous space is one of policy gradient methods. In a nutshell, these methods learn a *parameterized policy* by optimizing a target value using *stochastic gradient descent* methods. In this dissertation work, I will try different Monte Carlo methods, including REINFORCE method (Williams (1992)) and some variances.

In this problem, I constraint my set of spatial primitives and actions so that all of them could be simulated in a 2-dimensional environment. This is not really a requirement for the idea to work, but it allows me to quickly develop a 2-dimensional simulator to try different RL algorithms. An extension to 3-dimensional space by employing a more sophisticated physical engine simulator that can generate, possibly in parallel, multiple simulations, without visualization expense (as seen in game engines like Unity).

1.1.5 Visualization using 3-D simulator

To evaluate, I will visualize the action being performed in an embodied 3D simulation environment, which allows evaluators to judge whether the system has successfully learned the novel concepts. In principles, while the 2-D simulator could also be used as an estimation of the system correctness, a 3-D simulator aligns better with human cognitive ability, allowing more precise and intuitive judgment.

Krisnaswamy's VoxSim (Voxicon Simulator) (Pustejovsky and Krishnaswamy (2014)) used for generating visual scenes from textual sentences will be used as the basis to generate visualizations. VoxSim is, in turned, developed using the popular Unity 3-D game engine. VoxSim setup has been configured specifically to test action simulations in a **Block World** environment, whereas a virtual humanoid agent named Diana can be commanded to move objects around on a flat table.

1.2 Related Prior Work

This work is a culmination of an effort to cross-pollinate different lines of research in our Brandeis's Computational Linguistic lab. Pustejovsky's rigorous framework of dynamic events and event participants (Pustejovsky (2013)) led to different lines of research: a top-down semantic framework called *Multimodal Semantic Simulations (MSS)*, which can be used to encode events as programs in a dynamic logic with an operational semantics, and in turn is used to create an conversational and grounding interface between humans and computational agents; a bottom-up approach that learning event representation through machine learning methods, using data from 3-dimensional video captures. The first approach requires programming of various predicative types, such as verbal-action types (**slide**, **roll**, **put**), and spatial adverbials (**on**, **in**, **next to**, **around**). The second approach, so far, can only make distinction between different verbal-action types and spatial adverbial types (Do and Pustejovsky (2017a)). This work is trying to bridge the gap between two approaches, which is to learn the programmatic form of action directly from the data.

This work is also related to a traditional treatment in computational linguistics, in particular, in lexical semantic analysis, in which, motion verbs can be divided into two classes *manner-* and *path-* oriented predicates, and adjuncts can be used to complement meaning on the other aspect of motion (Jackendoff, 1983). These classification correspond to answers *how* the movement is happening and *where* it is happening. Taking, for example, these two following sentences (Krishnaswamy, 2017):

(c) The ball rolled$_m$ across the room.

(d) The ball crossed$_p$ the room rolling.

In the first sentence, the predicative verb describes the *manner* of motion, while the adjunction refers to the motion's path and vice versa for the second sentence. While this work will focus on machine learning ability in regard to motion's path, the methodologies in this work can be applied to manner of motions as well, with a consideration that *manner* of motions generally refers to movement gradient of the object over time. For example, *slowly* refers to small change of movement, while *roll* refers to intrinsic change of object orientation over time. The difficulty of learning these *manners*

of motion lies in the fact that these intrinsic movements sometimes depend on object affordances. For instance, a ball can be rolled, but not a cube.

Other inspirations to my dissertation work come from interpretation of spatial cognitive reasoning and its application in spatial control for robotic systems (Randell et al., 1992; Bhatt and Loke, 2008; Bhatt, 2012), qualitative reasoning as a common knowledge interface between humans and AI systems (Bhatt et al., 2011; Cohn and Renz, 2008), usage of video captures to infer or recognize event/action structure (Dubba et al., 2015; Duckworth et al., 2016; Carreira and Zisserman, 2017), cognitive linguistic work regarding human ability to learn new concepts or to perform novel actions (Gergely and Bekkering).

Chapter 2

Framework

The framework for this research rests on three foundations:

1. **Mapping among linguistic, visual and programmatic representations of actions** — learning of actions from humans requires robots to understand multimodal representations of actions. Inputs that robots could take in include linguistic descriptions or commands from humans, visual features (including RGB, depth-field and infra-red etc.). Mapping between linguistic and visual representation allows robots to recognize and describe actions, whereas an addition step of mapping from linguistic and visual representation to programmatic representation (or program form of action) allows robots to perform the action given human linguistic commands.

   Linguistic event representation in my framework is modeled as a verbal subcategorization in a frame theory, a la Framenet (Baker et al., 1998), with thematic role arguments. In addition, I also account for *extra-verbal factors*, i.e. the aforementioned adjunctions describing path of motions. Therefore, I consider *A moves B toward C* and *A moves B around C* as different event types and aim to learn each event type as a separate *atomic* action. A future work that include learning *manner* of action would allow further distinction such as between *A rolls B toward C* vs *A slides B toward C* and combination of path and manner aspects of actions.

   Our visual event representation comprises visual features extracted from tracked objects in captured videos or virtual object positions saved from a simulation environment. Both types of feature represent information visible to humans and

observable by a machine in object state information. Using these data points and sequences, machines can observe humans performing actions through processing captured and annotated videos, while humans can observe machines performing actions through watching simulated scenes.

Programmatic event representation can be based on formal event semantics or on features that can direct simulated or robotic agents to perform an action with an object of known properties. From a human perspective, the distinction between learning to recognize and learning to perform an action might be obvious. However from a machine's perspective, these two tasks might require different learning methods. My work aims to demonstrate that given an appropriate framework, it is feasible to map between them, in a manner similar to the way humans actually learn: by matching actions to observations.

2. **Qualitative spatial reasoning used as the common feature representation** — Qualitative spatial reasoning (QSR), a sub-field of qualitative reasoning, is considered to be akin to the way humans understand geometry and space, due to the cognitive advantages of conceptual neighborhood relations and its ability to draw coarse inferences under uncertainty, and also analogous to the way humans map from continuous space of spatial perception to discrete space of linguistic description (Freksa, 1992, 1991).

   It is also considered a promising framework in robotic planning (Cohn and Renz, 2001). QSR allows formalization of many qualitative concepts, such as *near*, *toward*, *in*, *around*, and facilitates learning distinction between them (Do and Pustejovsky, 2017b). The use of qualitative predicates ensure that scenes which are semantically close have very similar feature descriptions.

   Moreover, from a machine learning perspective, as pointed out by Yang and Webb (2009) and Jiang et al. (2017), qualitative representation is a method of discretization, which makes data sparser, therefore easier to learn. Especially when taking the difference between features of two adjacent frames, as a qualitative feature strongly distinguishes between 0 and 1, the effect of sequential change is more pronounced.

3. **Visualization as evaluation method** — Visualization using 3-D scene has been used extensively for evaluation of many difficult questions in cognitive sciences and computational linguistics. For example, Krishnaswamy (2017)'s framework allows evaluators to judge different visualizations of under-specified actions with

different sets of constraint parameters, therefore to pick out the most accepted set of under-specified parameters. Liang et al. (2015) tests the alignment of human cognition in understanding containment relationship between two objects with the simulated result from a visualization/physical simulator. Smart et al. (2016) uses Unity game engine as visualization component for cognitive architecture (ACT-R), for example, to show spatial cognitive capability of a virtual robot to navigate mazes.

To evaluate simulations, generated scenes will be recorded and shown to Amazon Mechanical Turkers with priming questions. Details of my planned experiments will be provided at section .

To facilitate the process of teaching robots new actions and concept, we need an interface to teach robots the mapping between linguistic and visual representation of an action, in composition with participants (agents and objects). Whereas the ultimate target is an online mechanism that allows robots to learn new actions and concepts while continuously interacting with environment, to simplify the learning process, this dissertation work will still stick to the learning loop of:

$$Capture \rightarrow Process\ Data \rightarrow Machine\ Learning \rightarrow Evaluation$$

This learning process could be mapped to the following pipeline of:

1. **Capturing data using Event Capture and Annotation tool (ECAT)** — ECAT - a video annotation tool capable of extracting 3D positional and orientational information about objects annotated in video captured by Microsoft's Kinect® hardware. Originally devised as a generic tool to capture and annotate event structure (Do et al., 2016), therefore to learn event/action structure, ECAT has been used to capture and annotate data for fine-grained event classification of human-object interaction (Do and Pustejovsky, 2017a) and for action annotation in movie snippets (Do, 2016). ECAT functionality and its usage is summarized in 2.1. ECAT is implemented in **C# .NET** language.

2. **Data processing and feature extraction modules** — This actually encompasses various components in ECAT and some other components implemented in Python. Components in ECAT include an algorithm to synchronize data from various capturing streams, such as depth-field and RGB, and one to generate 3-D data from

2-D tracked frame-by-frame object data, an algorithm. Components implemented in Python include an algorithm to recover missing data (when there are 2-D coordinates that could not be mapped to 3-D coordinates), to project data to simulator space, to interpolate coordinates in frames that do not have corresponding tracked data etc. Feature extraction framework will be addressed at 2.2.

3. (a) **Supervised machine learning algorithms** — Algorithms that are used to predict progress of a learned action, and ones used to make distinction between different action classes. They would be detailed at 2.3

   (b) **Running simulation with 2-D simulators** — A 2-D simulator implemented in Python with various searching and reinforcement learning algorithms. Functionality and implementation of this simulator will be described in 2.4.1.

4. **Evaluation with visualizers** — Human evaluation based on watching the demonstration of action on 2-D and 3-D visualizers. 2-D visualizer is a component of 2-D simulator, whereas 3-D visualizer is a scene modified from Voxsim (Krishnaswamy and Pustejovsky). 2.5 will give a brief description of the 3-D visualizer, while 3.2.2 will elaborate on various experiments carried out on the visualizers. Finally, a *Feedback* step could be included to provide improvement to the learning loop (3.2.2.3).

2.1 ECAT - Video capture and annotation tool

2.1.1 ECAT GUI

1. Project management panel. Each project can hold multiple captured sessions.

2. Video display. For displaying either the color video or grayscale depth field video, and locating objects of interest in the scene—e.g., the table outlined in green in Figure 2.1.

3. Object annotation controller. Yellow time scrub bars show when each tracked object appears in the video. Black ticks mark frames where an annotator has drawn a bounding polygon around the object using the object toolbox (item 5). *Link to* links the selected object to another using a specified spatial configuration. *Generate3D* generates the selected object's tracking data using the depth field.

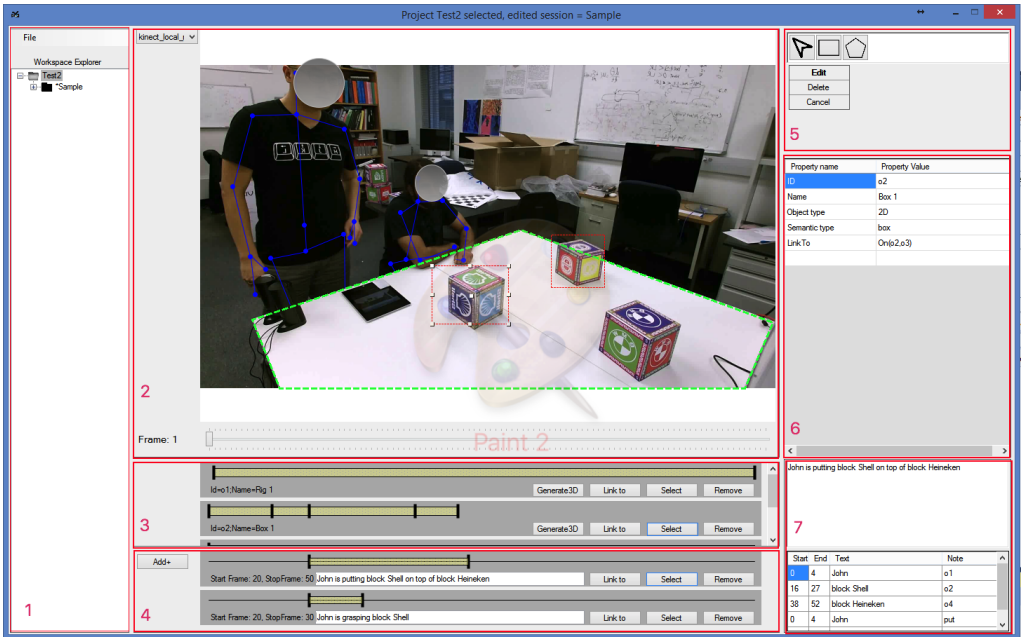


FIGURE 2.1: ECAT GUI. The left panel allows annotators to manage their captured and annotated sessions. Recognized human rigs are displayed as blue skeletons. Objects of interest are marked in color in the scene view.

4. Event annotation controller. Time scrub bars here show the duration of a marked event. Users provide a text description for the event, or use *Link to* to link the selected event to another captured event as a subevent. ECAT supports marking events that comprise multiple non-contiguous segments. Due to space constraints not all annotated subevents are visible in this screenshot.

5. Object toolbox. Annotators can manually mark an in-video object with a bounding rectangle or arbitrary polygon. Marked bounds can be moved across frames as the object moves.

6. Object property panel. Data about a selected object shows here, such as ID and name.

7. Event property panel. The selected event's properties, including type and participants, show here, and the event can also be linked to a VoxML event type.

2.1.2 Functionality

1. ECAT provides a tracking mechanism based on OpenCV implementation of detection algorithm for ARUCO markers (Garrido-Jurado et al., 2014). Users can put different ARUCO markers on different sides of a block so that ECAT can calculate its pose/orientation change over time. In a nutshell, the algorithm looks for parallelograms (demarcated by strong contrast pattern of black and white border) by searching through a gray image. Then it looks for checker pattern after transforming the parallelograms to squares. Searching over the whole image is carried out every 10 frames, whereas for remaining frames, the algorithm looks for space in the neighborhood of the previous frame.

2. ECAT provides mapping back and forth between 2-D boundary-based representation of objects, and 3-D boundary based representation. Currently it can infer corner coordinations of cubes in 3-D from detected ARUCO markers. It can also infer equation of the geometrical plane of a flat surface (such as the supporting table).

3. Users can mark the appearance of an object using the object toolbox. Marking the boundary of objects in 2-D can be used to infer object boundary in 3-dimensional by using depth-field data. I used this feature to mark the boundary of the table captured in the scene setup.

4. Each object has an `objectType` field that can be set to either *2D* or *3D*. One important tag that could be added to an object is `semanticType`, that can be linked to VoxML (Pustejovsky and Krishnaswamy, 2016). Based on `semanticType`, ECAT can infer different properties of an object, for example, if it is a marked_cube, it can infer the 3-d coordinations, whereas if it is a flat_surface, it can infer equation of geometrical plane.

5. Annotators may also mark spatial relations between objects. For example, at 2.1 two blocks are on top of the table. Users can create a link between a block object and the table object, using the *Link_to* button on the object tracking panel, and specify the relation between the objects as "on," resulting in the creation of a predicate $ON(Block\_1, Table)$ that is interpretable in a modeling language such as VoxML.

6. ECAT allows annotators to mark subevent relations between events. Thus, as in the example, the overarching event may be annotated as *put*, but it contains the

subevents *grasp*, *hold*, *move*, and *ungrasp*, which may overlap partially or entirely with some subsection of the main event and each other.

2.1.3 Data capture and annotation guidelines

Capture environment is as followings:

1. ECAT is installed on DARPA CwC Apparatus (Tamrakar, 2015). It includes the following components: a square table about 5' x 5', one flat screen mounted on one side of the table, and two Microsoft's Kinect® devices mounted on two sides next to the side mounted the screen (Figure 2.2).

2. Blocks are 6" cube blocks marked with ARUCO markers. The blocks belong to the original CwC setup, each printed with a brand name, such as *Pepsi*, *Stella Artois*). The ARUCO markers are stick on different sides of the blocks (Figure 2.3).

3. The apparatus is located so that performers can move freely around the performing table, and can observe the captured scenes on the flat screen, therefore to avoid moving out of the field of view.

FIGURE 2.2: Capture setup

Data capture guidelines are as followings:

1. Create a separate project for an action type. Under each project, for each performer, capture a long session of the performer doing multiple instances of an action.

2. The performers are given the linguistic form of an action, such as *slide A around B*, and are asked to perform according to their understanding of the action.

3. After each demonstration, the performers move the blocks to some initial configuration for the next demonstration. The performers are asked to vary the initial locations of blocks in each demonstration as much as they can, to vary their position with regard to the table, and change their performing hands. These strategies are mainly to add variation into training data.

FIGURE 2.3: Block with ARUCO markers

Annotation guidelines are as followings:

1. Annotators first mark the surface of the table throughout multiple frames of a session, by drawing a polygon boundary. Then the annotators click on *Generate 3D* to generate the 3-D planar equation for the table.

2. Annotators create or load ARUCO markers prototype file. This file map a block name to its ARUCO markers. In turn, each ARUCO marker is represented as 5x5 bitmap.

3. Annotators click on *Detect objects* for the ARUCO marker detection algorithm to run.

4. Annotators click on *Map to 3D* to map objects into 3-D coordinates using captured depth-field.

5. Annotators add events by marking event span (beginning and end frames) as well as adding a description of the event (e.g. The performer slides Pepsi around Stella Artois) while watching the replay of a session. Annotators are asked to mark the beginning of an event when the object starts to move, and the end when the performers stop moving object and release their grasp.

2.2 Feature extraction module

To facilitate event classification, it is necessary to present events in a learnable format. This introduces the question of how to represent events, namely the difficulty in defining their temporal and spatial extensions, as well as the difficulty in selecting an observational perspective.

Moreover, the feature representation for action needs to be shared between the real captured-data used for training and the artificial data generated from the simulator. The feature space in these two types of data are quite different: the real captured data is fuzzy and sometimes includes noisy outliers (represented as flicker when visualizing the captured session); the simulated data is smooth and interpolated frame-by-frame from planned actions.

This leads to difficulty in finding an appropriate feature set as common abstraction to bridge between these two kinds of data. I enumerate some different feature types I am experimenting in this work. They can be generally divided into two sets: quantitative and qualitative, whereas qualitative features are calculated based on quantitative features.

Both quantitative and qualitative features of the captured data are calculated based on projecting the blocks' coordinates on 2-d surface of the apparatus's table. The details

of the projecting algorithm is as following:

Input: 3-D coordinates of markers for each block (one block might have multiple visible markers)
Size of the block (different from size of a marker) **Purpose**: Estimate a 2-d square for each block on projecting surface
Result: Centroid and angle of projected 2D square
Estimate plane equation P of the table;
for *Each block* **do**
    Select the marker that has the most number of corners, and the largest size (most clearly observed marker) ;
    Estimate plane equation Q of the marker face;
    if *the cosine product of P and Q norm vertors* < 0.5 **then**
        Calculate the square T that perpendicular to Q, have the correct size, and have the edge on the middle line of marker P;
        Project T on P (T_P);
    else
        Rescale and project the marker on P (T_P);
    end
    Use a single point on P as the (0,0) origin, estimating the transform of the 2-d square from (T_P);
end

Algorithm 1: Algorithm to project blocks on 2-D simulator

2.2.1 Quantitative features

Firstly, in each demonstration of action, I rank the objects by their salience, i.e. the object that moves the most is the most salient. Quantitative features include the followings:

1. Location of the centroid of a block, and the block's angle made with regard to the Ox axis (the transform as estimated from Algorithm 3).
2. The difference between transforms of the most salient object to the second salient object.

3. The gradient of the transforms of the objects, i.e. the difference between two frames.

While quantitative features are not the focus in my discussion, we need include them to provide a more comprehensive picture, and to give comparison to the qualitative methods.

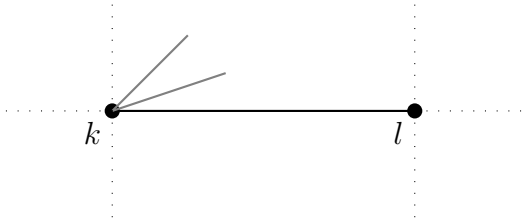
2.2.2 Qualitative features

There is a significant literature supporting the use of discretization for feature embedding. Yang and Webb (2009) shows that discretization is equivalent to using the true probability density function. More recently, Jiang et al. (2017) has used this method for classification of GPS trajectory. They studied three different approaches for discretization, including *equal-width binning*, Recursive Minimal Entropy Partitioning (RMEP) (Dougherty et al., 1995) and fuzzy discretization(Roy and Pal, 2003). Their finding is that the *equal-width binning* approach is both simple and effective, so I used this approach for quantization of both distance and orientation.

QSRLib Gatsoulis et al. (2016), a library that allows computation of Qualitative Spatial Relations and Calculi is employed to generate qualitative features. In particular, I use the following feature types from QSRLib:

- CARDINAL DIRECTION, (Andrew et al., 1991) (QSRLIb *cardir*) measures compass relations between two objects into canonical directions such as North, North East etc. In total, this qualitative relation give 9 different values, including one where two locations are identical.
- MOVING or STATIC (QSRLIb *mos*) measures whether a point is moving or not.
- QUALITATIVE DISTANCE CALCULUS (QSRLib *argd*) discretizes the distance between two moving points. Here I discretizes the distance between two centers of two squares.
- QUALITATIVE TRAJECTORY CALCULUS (Double Cross) (QSRLib *qtccs*): QTC_C is a representation of motions between two objects by considering them as two moving point objects (MPOs) (Delafontaine et al., 2011). The type C21 of QTC_C

(implemented in QSRLib) considers whether two points are moving toward each other or whether they are moving to the left or to the right of each other. The following diagram explains this:



QTC_C produces a tuple of 4 slots (A, B, C, D), where each could be given either -, + or 0, depending on the angle α. For example, C is $+$ if $\alpha > 0 \wedge \alpha < 180$, $-$ if $\alpha > 180 \wedge \alpha < 360$ and 0 otherwise. QSRLib also allows specification of a *quantisation factor* θ, which dictates whether the movement of a point is significant in comparison to the distance between k and l.

2.3 Supervised machine learning module

The main supervised learning method used in this dissertation is a version of Long-short term memory (LSTM) (Hochreiter and Schmidhuber, 1997) that processes sequential inputs to a sequence of output signals. LSTM has found utility in a range of problems involving sequential learning, such as speech and gesture recognition.

2.3.1 Event classifier

Event classifier is used to make distinction between different event types. In fact, it could be used to classify different event types that combine *manner*- and *path*- aspects of motions. In specific, using Conditional Random Field (CRF) as a constraint layer for output labels, *manner*- and *path*- motion aspects as reflected on predicates and conjunctions could be jointly classified.

The architecture of the event classifier is detailed in Do and Pustejovsky (2017a), but could be summarized as followings:

1. *manner*- and *path*- aspects of motions are reflected in the verb and adjunction slot in a frame-based representation of action description. For example, a description

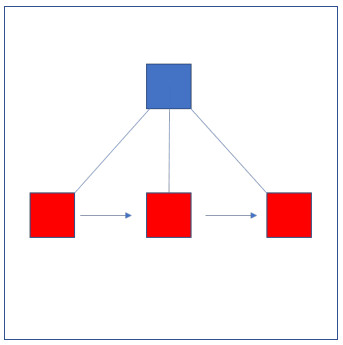


FIGURE 2.4: Red block moves past blue block

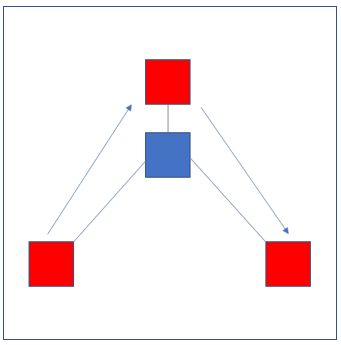


FIGURE 2.5: Red block moves around blue block

The performer pushes A toward B" could be mapped to the slot representation (Subject, Object, Locative, Verb, Adjunction) as (The performer, A, B, Push, Toward). An ensemble learner could learn all the slots at the same time by combing multiple sequential learners. In particular, multiple LSTM learners, one for each slot, could be constrained to produce appropriate outputs.

2. Event classifier could be used as an auxiliary component to make distinction to other learned actions. We can try to make robots to learn a novel action in isolation to other actions, but there are two reasons making distinction to other actions might be helpful. The first reason is that we could make reference to the way humans learning a new concept. We exhibit a strong *mutual exclusivity bias*, which entails Principle of Contrast (Merriman et al., 1989), i.e. different concepts should have different names and vice versa, different words should be associated with different things. The practical reason is that a naive learner could learn two actions "move A past B" and "move A around B" as "A move closer to B, then move further away from B" 2.4. The learner might not be able to make distinction between this two actions without referring to a classifier.

2.3.2 Progress learner

Inputs are sequence of feature vectors taken from capture-data or from the simplified simulator and outputs are a function that correspond to the progress of an event. In particular, it is a function that takes a sequence S of feature vectors, current frame i and action e: $f(S, i, e) = 0 \leq q_i \leq 1$

The training set of sequential captured data is passed through an LSTM network, which is fitted to predict a linear progressing function. At the start or outside of an event span, the network produces 0, whereas at the end, it produces 1.

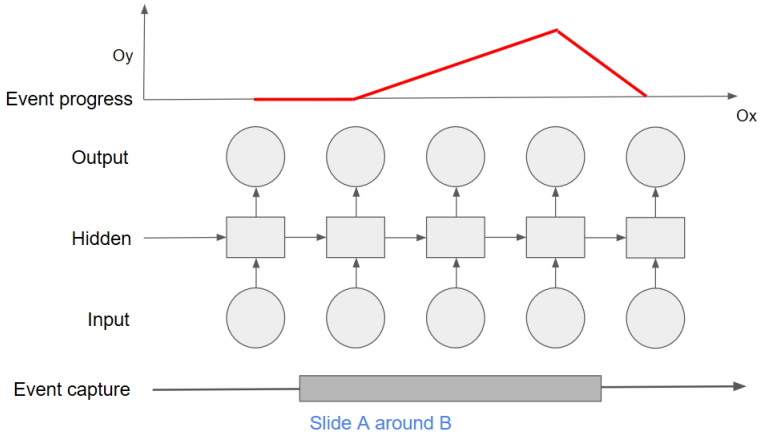


FIGURE 2.6: LSTM network producing event progress function

2.4 Simulation module

This module encompasses both a 2-D simulator that can quickly generate multiple demonstrations of an actions, as well as multiple algorithms to learn the best approach to generate an action. In implementation, it is also a wrapper around the predictors of the aforementioned supervised machine learning module 2.3, and also around the feature extractors 2.2.

Therefore, in my implementation, this module simulates the whole mechanism of a virtual agent's faculty of action learning. Section 2.4.1 will give a summary of the 2-D simulator, including its components and functionality. Section 2.4.3 will give a simple searching algorithm that search over the space of trajectory in a heuristic manner, however this algorithm will have to search over again for any new configuration. Section 2.4.2 will sketch out some reinforcement learning algorithms that can generate a strategy for new situation without resorting to searching over space.

2.4.1 2-D Simulator

For the updating loops in my reinforcement learning algorithm, I want to simulate observational data similar to the real captured data faster than real-time for effective computation. As a real-time, graphic-heavy simulator, VoxSim is not feasible for this portion of the task. While being aware of a few other physical simulation environments such as Gazebo[1], but as I do not focus on physical constraints in this study, I implemented my own simplified simulator in Python.

My set of learnable actions is limited to ones that can be easily approximated in 2D space. 3D captured data is transformed into simplified simulator space by projecting it onto a 2D plane defined by the surface of the table used for performing the captured interaction. My 2D simulator has the following features:

- It is equipped with aforementioned feature extractors. These extractors are used to generate features on a frame-by-frame basis from the simulator.
- It includes the aforementioned progress learner for each action type. This allows us to calculate an accumulated reward function after a move has been carried out.
- Each object is represented as a polygon (or square), is attached with a $transform$ object that stores its position, rotation, and scale.
- The space is constrained so that objects do not overlap, and objects could not be located outside of a square rectangle (table boundary).
- Object could be move to a new location (change of the attached *transform* object), by sending action command to the simulator. If the action is an illegal move (moving to outside of the table, or to an overlapped location), action is recorded, but not carried out.
- Speed can be specified so that object movement can be recorded as a sequence of feature vectors interpolated frame to frame.
- It supports step-by-step visualization of a sequence of moves, or locations of objects could be interpolated and the movement could be recorded into video. Therefore it provides a simple visualizing and debugging environment.

[1] http://gazebosim.org/

In details, it is implemented with the following components:

- An environment simulator class that supports adding of objects at specified locations. A movement command can be sent to the environment to change location of one object, as far as: a. destination location is not blocked, b. assuming that the object is moved with the original orientation, and only changes its orientation at the end of the movement, the corridor of space needs to be clear for movement to happen.
- An action environment class that inherits *gym.Env*, a class from OpenAI Gym (Brockman et al., 2016) Python package. It is basically an adapter for reinforcement learning methods to the aforementioned simulator, providing functionality to initiate environment with random configurations, and support recording and traversing through history of each episode.
- 2-D Visualizer/debugger that can show an episode to an interactive python notebook, or save it down as a video. It can also be used to visualize the captured data as they are projected onto 2-D space.

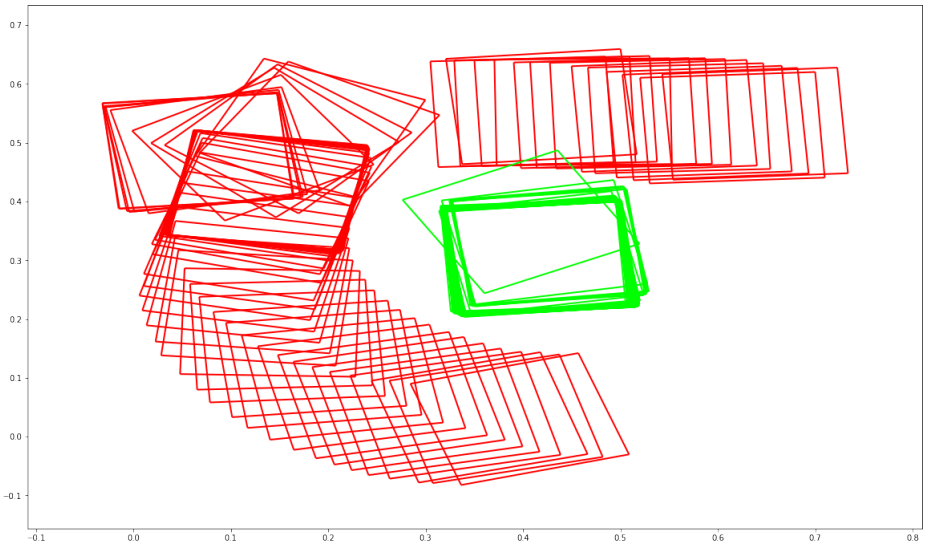


FIGURE 2.7: An event "Move A around B" projected into simulator. A is projected as a red square, B as a green square

2.4.2 Reinforcement learning algorithms

In this work, I will experiment with both running RL on continuous space and discretized space and we can evaluate the effectiveness of both methods in solving this problem. The selected algorithm for both cases are REINFORCE algorithm (Williams, 1992), for its effectiveness in policy gradient learning, which could be used for both continuous and discretized RL problems.

Input: Initial policy parameter θ, learning rate α;
Result: Learned policy θ_{final}
for *episode j* **do**
  Initialization $\pi_\theta \leftarrow \theta$, randomize initial state in simulator space X_o, reward r = 0;
  while *true* **do**
    $current\_step = k$;
    Draw set of an action from policy distribution $u_{ki} \leftarrow \pi(\theta)$;
    for *action* u_{ki} **do**
      do action $u_k i$ in simulator, record frame-to-frame sequential features, feed them into LSTM network to get progress output;
      calculate immediate reward as
      $r = \delta_f = f(after\_the\_action) - f(before\_the\_action)$
    end
    Select action u_k that leads to highest reward r_{max};
    if $r_{max} < 0$ **then**
      break;
    end
    do action u_k in simulator;
    get next state X_{k+1}
  end
  After episode terminates, obtain $X_{0:H}$, $u_{0:H}$, $r_{0:H}$;
  Estimate baseline b (Peters and Schaal, 2008);
  Estimate gradient
  $g = \langle (\sum_{h=0}^{H} \bigtriangledown_\theta log_{\pi_\theta}(u_h|x_h))(\sum_{h=0}^{H} a_h * r_h - b) \rangle$
  $\theta = \theta + g$;
end
Until: $j > Max\_episode$ or gradient converges

Algorithm 2: Williams Episodic REINFORCE algorithm

2.4.2.1 Continuous space

For the case when the action space is continuous, planning is carried out by selecting the action (coordinates in continuous space) at step k (u_k) based on the current state of the system ($X_k \in R^n$) (also continuous values). A stochastic planning step is parameterized by policy parameters θ : $u_k \sim \pi_\theta(u_k|x_k)$

Problem formulation is as followings:

1. **State**: *transforms* (X and Y coordinate and rotation) of two squares in continuous space.
2. **Action**: *transform* of the target location for the moving object.
3. **Reward**: gain of *progress function* over one move.
4. **Policy**: My *Policy estimator* is made by an artificial neural network (ANN), used to produce values μ (mean of the action's transform) and σ^2 (variance of the action's transform). The ANN will be parameterized by a set of weights θ. For simplicity, the dimensions of μ and σ are the same as the degrees of freedom in our simplified simulator (2 dimensions for position and 1 dimension for rotation). Action is generated by a Gaussian distribution $\pi_\theta(u|x) = Gaussian(\mu, \sigma)$.
5. **Baseline function**: Basically a function that estimate value of a state (expected accumulated reward from a state to the end of an episode). Here I use another ANN to predict this value.

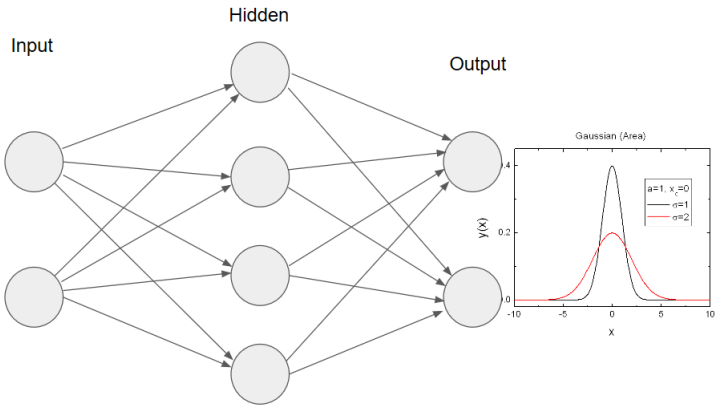


FIGURE 2.8: An ANN architecture producing Gaussian policy

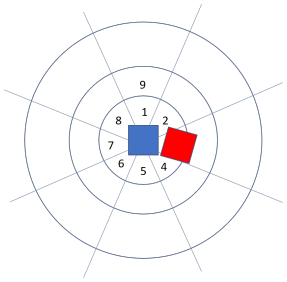


FIGURE 2.9: Discretizing 2-D searching space around the static object

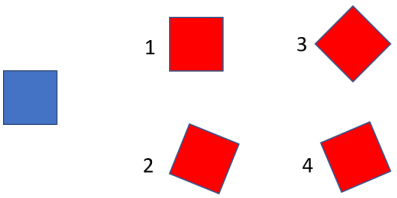


FIGURE 2.10: Discretizing rotation

2.4.2.2 Discretized space

I discretize the searching space using qualitative reasoning method. In specific, the searching space for the *transform* of the target location could be separated into space for (X, Y) coordinates and rotation r. The searching space for (X, Y) could be discretized as in Figure 2.9, the searching space for r could be discretized as in Figure 2.10. Notice the searching space for (X, Y) is divided so that the granularity of discretization is coarser when the moving object get further away from the static object.

Problem formulation is as followings:

1. **State**: Discretized state of the moving object with regard to the static object + discretized angle of the previous movement (or 0 if the object hasn't moved).
2. **Action**: Discretized transform of the target location.
3. **Reward**: gain of *progress function* over one move (same as for continuous case)
4. **Policy**: An artificial neural network (ANN) will be used to produce probability distribution of discretized action for each state. My implementation will use a simple fully connected layer for this ANN. An action is generated by randomizing from the discretized values.
5. **Baseline function**: Similar to continuous case.

2.4.3 Searching algorithms

Similar to the RL algorithms, my searching algorithms also have a continuous and a discretized versions. Following is the algorithm for continuous case:

Input: state of the 2-D simulator (locations of blocks, which block would be moved)
Purpose: generate a chain of movement for the moving block that maximizes the progress function
Result: A chain of movement;
while *# of steps* $\leq$ *max_step* **do**
    Random N location T_i on the searching space;
    for *Location* T_i **do**
        Move the moving square to T_i ;
        Calculate progress p_i;
        Back the moving square to previous location;
    end
    Select T_{best} according to progress values;
    Move the moving square to T_{best} ;
end

Algorithm 3: Searching for trajectory of action

For the discretized case, it would be simple enough to derive the algorithm based on my previous discussion on discretization. One noteworthy point is that when we generate N random locations, we can just generate one location for each discretized space slot. That reduces the computational expense, while keeping the coverage over the searching space. That also allows keeping deeper layer of backups (so we can plan two or three actions ahead, before deciding which one produces the best progress).

2.5 Visualization module

The visualizer is modified from our lab's internally developed environment for generating animated scenes in real time using real-world semantics of objects and events (Krishnaswamy and Pustejovsky). The whole framework is a robust system that can work with a large set of actions and objects, and backed by a solid theoretical foundation

(Pustejovsky and Krishnaswamy, 2016). In this dissertation work, I will only describe the particular scene used to generate visualization for testing my learning methodology:

1. The scene has an embodied agent, named Diana, playing the role of a communicative agent. Diana can move the block around using her hand, which makes the scene more realistic than the blocks are moved by themselves.

2. The scene provides functionality for an annotator to place blocks on any locations on the table. It also provides a method to randomize locations of two blocks on the table.

3. It supports input of an action command, such as "Move the yellow block toward the black block", and the Unity game engine would send the configuration of the environment, including blocks' location to the learning module (implemented in Python) to calculate trajectory of the movement and send back to the Unity engine to perform the movement.

4. Users can click on a button for Diana to do a random action from the list of available actions, then after the action is performed, the visualizer shows a panel for users to guess which action has been made.

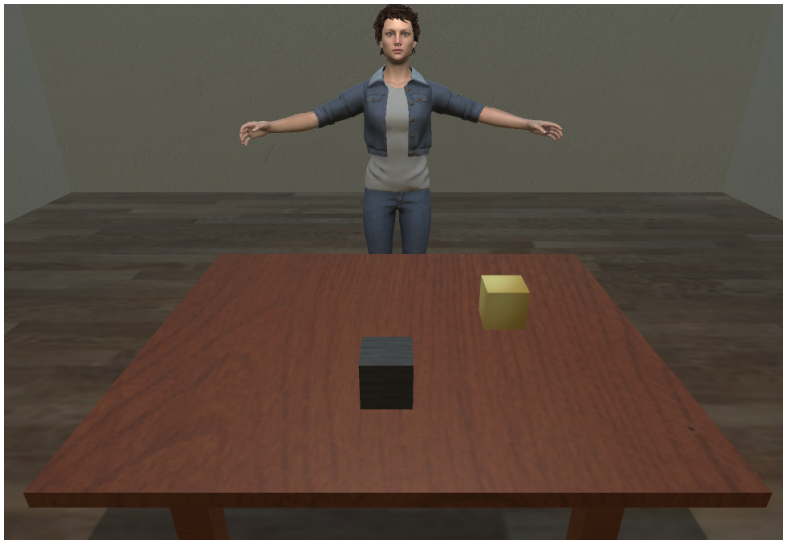


FIGURE 2.11: Visualizer is implemented as a Unity scene

Chapter 3

Proposed Work and Methodology

In this experiment, I will use the learning framework drafted above for learning agent to perform the set of following *actions*:

1. An agent moves {object A} **closer to** {object B}
2. An agent moves {object A} **away from** {object B}
3. An agent moves {object A} **past** {object B}
4. An agent moves {object A} **next to** {object B}
5. An agent moves {object A} **around** {object B}

This set of actions, on linguistic description, differ only in their adjunctions. These adjunctions are prepositions describing different trajectories of motions. For this set, the learning problem is reduced to learning trajectories, or path, of motions.

It is noted that the action types in this set are not homogeneous. Firstly, we would discuss about temporal extent of each action type, whether they are open-ended or close-ended, and whether they have a hard boundary or soft boundary. Secondly, we would discuss the way humans would recognize these action types, whether we base on the trajectory of movement, or we just base on the start and ending points of movement. Followings are the differences:

1. **closer to** differ from **next to** that A only gets **closer to** B but does not touch B. **closer to** is a close-ended action, as A can only move closer to B until certain point. **closer to** has a soft ending, which means that the action is satisfied for a time interval.

2. **away from** is an open-ended action, it also has a soft ending.

3. **past** is an open-ended action and has a soft ending. It combines the effect of **closer to** and **away from**.

4. **next to** has a hard ending and is close-ended.

5. **around** has a hard ending and is close-ended.

Arguably, all of the actions, excepts possibly for **move next to**, requires consideration of the trajectory as well as the start and ending points of movements. For example, **closer to** is conceptually just involve change of distance between the start and the ending position of the moving object in relative to the static object, but a complex moving trajectory would lead to misinterpretation of the action. **Closer to**, therefore, strongly indicates a trajectory of the moving object toward the static object.

By putting these event types to be learned altogether, I want to examine the capability of a single learning framework that can learn multiple event types. The reason is rather obvious: we, as humans, can learn all of these actions without prior knowledge of different action types, or without further input from the teachers.

3.1 Data preparation

Currently for each action type, I have data recorded with two performers, and each person performs the action 20 times, for a total of 40 demonstrations for each action. Even though this is just a small amount of demonstrations, increasing this number is rather simple, as all data so far was captured in a single evening. Annotation took a little bit longer, about 10 hours. I also hope that the feedback loop could compensate for the small number of demonstrations, as increasing the number of demonstrations might still not be able to give good negative samples to the learning module.

3.2 Evaluation

3.2.1 Machine-based Evaluation

Firstly I would evaluate the performance of the learner progress, comparing between qualitative and quantitative feature sets. Secondly, I would compare between the running time of searching algorithm and RL algorithms. We would like to know if the searching algorithms are fast enough. If they are fast, and perform equally well to RL algorithms (when evaluated by humans), it might be more reasonable to use searching algorithm to avoid overhead of running RL algorithms.

Thirdly, a rough estimation of the performance of RL algorithms could be measured by examining the average return for a span of consecutive runs. This figure would be increased overtime for a successful RL algorithm. Therefore we can use this method to avoid running expensive human-driven evaluation for any RL algorithm that could not converge to a good average return.

3.2.2 Human-Driven Evaluation

I will conduct one human evaluation based on watching the 2-D simulator, and one human evaluation based on watching simulations from the Unity 3-D visualizer. The human evaluation on 2-D simulator would be carried out on a smaller scale. The purpose of evaluation on 2-D simulator is two fold: firstly to reduce the number of control algorithms we would finally use for more expensive evaluation on the 3-D visualizer; secondly, that would help to answer if there is any disparity between the way humans judge the same action demonstration on 2-D simulator and 3-D visualizer.

Evaluation with 2-D visualizer will be performed by lab members or school students.

Evaluation with 3-D visualizer will be conducted using the Amazon Mechanical Turk platform (AMT). This platform has been proven to be a good source for cheap and reliable evaluation (Buhrmester et al., 2011), especially for cognitive ability tasks, such as testing adult ability to recognize words from uncertain auditory and visual inputs (Fourtassi and Frank), or human mental simulation in judging physical support (Gerstenberg et al.).

We will answer the following questions:

1. Which algorithm works the best, and what is its performance? Addressed at 3.2.2.1
2. Can the learner makes distinction between learned action types? Addressed at 3.2.2.2
3. Can we use the feedback from human evaluation to improve the learned model? Addressed at 3.2.2.3

3.2.2.1 Which algorithm works the best, and what is its performance?

For each evaluation round, we generate a random configuration of 2 blocks on the surface of the table. For each action type, the initializing algorithm will generate a text description of the action and generate a simulation corresponding to each algorithm. The annotator will be asked to answer the question: *"On the scale of 0 to 10, how do you grade the video shown with regard to the description?"*.

In this experiment, by collecting grades for each action type, we get average performance of an algorithm in generating simulations. Hereby, we can evaluate if one algorithm would work for all action types in our collection, or there are disparity between action types that leads to advantages or disadvantages of a certain algorithm.

As the ultimate purpose is to pick out a good algorithm that is indifferent to action types to be learned, we would average over the performance of an algorithm over all action types. As for later experiments, they would get increasingly expensive if we keep multiple algorithms, so I will only keep the best algorithm for other downstream experiments.

3.2.2.2 Can the learner makes distinction between learned action types?

This experiment will examine the ability of the learner to distinguish these action types. The annotator will click on a *Randomize action* button, and the system will pick one of the available action types for Diana to perform. After the demonstration, a multiple choice panels will be displayed, and the annotator will then be asked to answer

the question: *"Which of these sentences **best** describes the video shown?"*, with the aforementioned 5 descriptions, plus " None of these sentences describe the video" :

The task requires annotators to predict which sentence was used to generate the visualization in question. While the first experiment gives us a rough estimation of how well the learning algorithm is, only by juxtaposing different action types for annotators to select, we can measure if learning actions in isolated manner is enough. Some actions in this set are close enough for confusion, even when demonstrations are made by a human expert.

3.2.2.3 Can we use the feedback from human evaluation to improve the learned model?

As mentioned in 2, feedback from human evaluation could be used as training data. Demonstrations from the first experiment (3.2.2.1) are complementary to real, captured data, and in some sense are even better because typically in learning by demonstration, we do not have any rigorous way to include negative samples. Incorporating low-graded demonstrations is a good approach for updating the progress learner with negative samples. Moreover, another advantage is that data comes from generation would be easier and quicker to collect.

In this experiment, for the best performing algorithm, for each action, we will collect the generated demonstrations that have grades higher than 7 as additional positive samples, and ones that have grades lower than 3 as negative samples. The progress learner would be updated with those samples, and new demonstrations would be generated with the same initial configurations as all old demonstrations.

To test whether the newly learned model performs better to the original model, we carry out a simpler form of Experiment 3.2.2.1, in which the annotator will be asked to answer the question: *"Which one between these two videos **best** represents the sentence displayed above?"*. The old demonstration and the new demonstration would be shown for the annotators to choose.

3.3 Expected Results

On the first experiment (Exp. 3.2.2.1), I expect the discretized version of either searching or RL algorithm would perform the best overall. Especially for the case of "moving next to", without discretizing the rotation of moving object and the searching space, the two objects might not be able to align to be considered "next to each other". Moreover, running RL algorithm over continuous space might be too difficult for it to converge in a reasonable amount of time, so the continuous algorithm would be disadvantageous.

On the second experiment (Exp. 3.2.2.2), I expect high confusion between "moving next to" and "moving closer to", as the first action subsumes the second action. "Moving around" might be hard to recognize as well, as we generally think of the trajectory of "moving around" to be rather smooth.

I highly expect that Exp. 3.2.2.3 would improve the overall performance of the learner. Based on watching 2-D simulator, there is false demonstration of actions that could be accounted by lack of negative samples, such as the one in Fig 3.1.

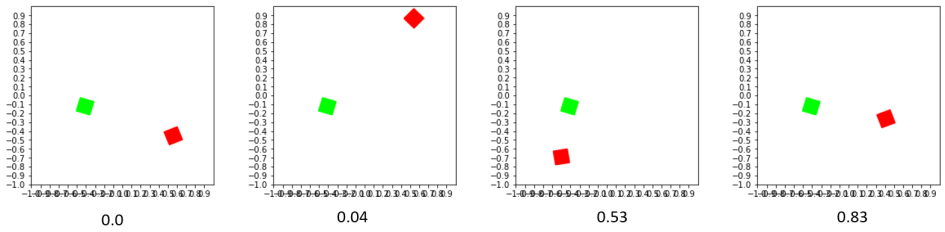


FIGURE 3.1: A wrong demonstration of "Move red block around green block". The values beneath each frame is value predicted by the progress learner.

3.4 Milestones

3.4.1 Progress to Date

Progress to date has been on the development of Event Capture and Annotation Tool and the machine learning algorithms used in this work:

- **September, 2015** — Start working on CwC project, devising the concept of an event learning toolkit.

- **December, 2015** — First built of ECAT, with simple annotation tools i.e. marking object boundary, event annotation.
- **March, 2016** — Second built of ECAT, integrated with 3-D tracking and mapping functionality for several object types (human rigs, blocks, table).
- **May, 2016** — Third built of ECAT, first attempt to integrate VoxML semantic structure with ECAT. ECAT is presented at ISA 2016.
- **August, 2016** — Used ECAT as an annotation tool for movie dataset (Do, 2016). I would not detail this project here, but the target is the same as in this project, which is to find event representations from video captures.
- **November, 2016** — Implementation of an event classifier with LSTM and CRF.
- **March, 2017** — Presented ECAT and the event classifier at AAAI-SS 2017 in the framework of learning object and event representation from multimodal resource (Pustejovsky et al., 2015).
- **April, 2017** — Presented ECAT and the event classifier at ESANN 2017 (Do and Pustejovsky, 2017a).
- **August, 2017** — Developed feature extractor based on qualitative reasoning method. Presented the event classifier with QSR features at Qualitive Reasoning workshop (Do and Pustejovsky, 2017b).
- **November, 2017** — Developed 2-D simulator, searching and RL algorithms.

3.4.2 Future Research Plans

Future research before the proposed completion date will be primarily concerned with completing the 3-D visualizer sketched in Section 2.5, and the experiments outlined in Section 3.2.2.

For the completion of this research, I propose the following timeline:

- **February, 2018** — Completion of the 3-D visualizer. Defense this proposal.
- **March, 2018** — Completion of lab-scale human evaluation on 2-D and 3-D visualizers

- **April, 2018** — Completion of experiments 3.2.2.1 and 3.2.2.2. Analysis of experimental results.
- **May, 2018** — Completion of experiment 3.2.2.3. Analysis of experimental results.
- **June, 2018** — Write-up complete with experimental data, results, and conclusions
- **July, 2018** — Final thesis defense

3.4.3 Post-Thesis

There are two different lines of research that can be considered extension from this framework. One line of research involves learning mechanism for more complex actions, such as "make a row from given objects", and one line of research involves learning manner aspect of actions.

Learning complex actions from simpler actions requires additional semantics framework for objects and actions. For example, to learn "make a row from given objects", and the learner observes rows of length 2 and length 3, the learner needs to be equipped with the concept of *repetition*, the concept of composite object made from elementary objects (e.g. size and shape of the composite object), and other abstract concepts, such as object axis.

Learning manner aspect of actions requires fine-grained treatment of object affordances. For example, for learner to make the distinction in performing "rolling a bottle" and "sliding a bottle", we need to equip it with reasoning mechanism related to how an object's pose and position dictating its affordances. These questions, hopefully, can be answered by making reference to a developing language for objects and events in our lab named VoxML (Pustejovsky and Krishnaswamy, 2016).

Bibliography

Muhannad Alomari, Paul Duckworth, Nils Bore, Majd Hawasly, David C Hogg, and Anthony G Cohn. Grounding of human environments and activities for autonomous robots. In *IJCAI-17 Proceedings*, 2017.

UF Andrew, D Mark, and D White. Qualitative spatial reasoning about cardinal directions. In *Proc. of the 7th Austrian Conf. on Artificial Intelligence. Baltimore: Morgan Kaufmann*, pages 157–167, 1991.

Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

Minoru Asada, Eiji Uchibe, and Koh Hosoda. Cooperative behavior acquisition for mobile robots in dynamically changing real worlds via vision-based reinforcement learning and development. *Artificial Intelligence*, 110(2):275–292, 1999.

Collin F Baker, Charles J Fillmore, and John B Lowe. The berkeley framenet project. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1*, pages 86–90. Association for Computational Linguistics, 1998.

Mehul Bhatt. Reasoning about space, actions, and change: A paradigm for applications. *Qualitative Spatio-Temporal Representation and Reasoning: Trends and Future Directions: Trends and Future Directions*, page 284, 2012.

Mehul Bhatt and Seng Loke. Modelling dynamic spatial systems in the situation calculus. *Spatial Cognition and Computation*, 2008.

Mehul Bhatt, Hans Guesgen, Stefan Wölfl, and Shyamanta Hazarika. Qualitative spatial and temporal reasoning: Emerging applications, trends, and directions. *Spatial Cognition & Computation*, 11(1):1–14, 2011.

Aude Billard, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. Robot programming by demonstration. In *Springer handbook of robotics*, pages 1371–1394. Springer, 2008.

Robert Bogue. Domestic robots: Has their time finally come? *Industrial Robot: An International Journal*, 44(2):129–136, 2017.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Michael Buhrmester, Tracy Kwang, and Samuel D. Gosling. Amazon's mechanical turk a new source of inexpensive, yet high-quality, data? *Perspectives on psychological science*, 6(1):3–5, 2011.

Sylvain Calinon and Aude Billard. Teaching a humanoid robot to recognize and reproduce social cues. In *Robot and Human Interactive Communication, 2006. ROMAN 2006. The 15th IEEE International Symposium on*, pages 346–351. IEEE, 2006.

Sylvain Calinon, Florent Guenter, and Aude Billard. On learning, representing, and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(2):286–298, 2007.

Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. *arXiv preprint arXiv:1705.07750*, 2017.

Anthony G Cohn and Jochen Renz. Qualitative spatial representation and reasoning. 46:1–2, 2001.

Anthony G Cohn and Jochen Renz. Qualitative spatial representation and reasoning. *Foundations of Artificial Intelligence*, 3:551–596, 2008.

Matthias Delafontaine, Anthony G Cohn, and Nico Van de Weghe. Implementing a qualitative calculus to analyse moving point objects. *Expert Systems with Applications*, 38(5):5187–5196, 2011.

Tuan Do. Event-driven movie annotation using mpii movie dataset. 2016.

Tuan Do and James Pustejovsky. Fine-grained event learning of human-object interaction with lstm-crf. *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, 2017a.

Tuan Do and James Pustejovsky. Learning event representation: As sparse as possible, but not sparser. *arXiv preprint arXiv:1710.00448*, 2017b.

Tuan Do, Nikhil Krishnaswamy, and James Pustejovsky. Ecat: Event capture annotation tool. *Proceedings of ISA-12: International Workshop on Semantic Annotation*, 2016.

James Dougherty, Ron Kohavi, Mehran Sahami, et al. Supervised and unsupervised discretization of continuous features. In *Machine learning: proceedings of the twelfth international conference*, volume 12, pages 194–202, 1995.

Krishna SR Dubba, Anthony G Cohn, David C Hogg, Mehul Bhatt, and Frank Dylla. Learning relational event models from video. *Journal of Artificial Intelligence Research*, 53:41–90, 2015.

Paul Duckworth, Muhannad Alomari, Yiannis Gatsoulis, David C Hogg, and Anthony G Cohn. Unsupervised activity recognition using latent semantic analysis on a mobile robot. In *IOS Press Proceedings*, number 285, pages 1062–1070, 2016.

Abdellah Fourtassi and Michael C Frank. Word identification under multimodal uncertainty.

Christian Freksa. Qualitative spatial reasoning. *Cognitive and linguistic aspects of geographic space*, 63:361–372, 1991.

Christian Freksa. *Using orientation information for qualitative spatial reasoning*. Springer, 1992.

Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco José Madrid-Cuevas, and Manuel Jesús Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, 2014.

Y Gatsoulis, M Alomari, C Burbridge, C Dondrup, P Duckworth, P Lightbody, M Hanheide, N Hawes, and AG Cohn. Qsrlib: a software library for online acquisition of qualitative spatial relations from video. In *Workshop on Qualitative Reasoning (QR16), at IJCAI-16*, 2016.

G Gergely and H Bekkering. Kirá ly, i.(2002). rational imitation in preverbal infants. *Nature*, 415(6873):755.

Tobias Gerstenberg, Liang Zhou, Kevin A Smith, and Joshua B Tenenbaum. Faulty towers: A hypothetical simulation model of physical support.

Nick Hawes, Christopher Burbridge, Ferdian Jovan, Lars Kunze, Bruno Lacerda, Lenka Mudrová, Jay Young, Jeremy Wyatt, Denise Hebesberger, Tobias Kortner, et al. The strands project: Long-term autonomy in everyday environments. *IEEE Robotics & Automation Magazine*, 24(3):146–156, 2017.

Junhu He. Robotic in-hand manipulation with push and support method. 2017.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Ray Jackendoff. *Semantics and Cognition*. MIT Press, 1983.

Xiang Jiang, Erico N de Souza, Xuan Liu, Behrouz Haji Soleimani, Xiaoguang Wang, Daniel L. Silver, and Stan Matwin. Partition-wise recurrent neural networks for point-based ais trajectory classification. In *25th European Symposium on Artificial Neural Networks, Computtional Intelligence and Machine Learning*, pages 529–534. ESANN, 2017.

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.

Nikhil Krishnaswamy. *Monte Carlo Simulation Generation Through Operationalization of Spatial Primitives*. PhD thesis, Brandeis University, 2017.

Nikhil Krishnaswamy and James Pustejovsky. Voxsim: A visual platform for modeling motion language.

Wei Liang, Yibiao Zhao, Yixin Zhu, and Song-Chun Zhu. Evaluating human cognition of containing relations with physical simulation. In *CogSci*, 2015.

Tomas Lozano-Perez. Robot programming. *Proceedings of the IEEE*, 71(7):821–841, 1983.

John H Maunsell and David C Van Essen. Functional properties of neurons in middle temporal visual area of the macaque monkey. ii. binocular interactions and sensitivity to binocular disparity. *Journal of neurophysiology*, 49(5):1148–1167, 1983.

William E Merriman, Laura L Bowman, and Brian MacWhinney. The mutual exclusivity bias in children's word learning. *Monographs of the society for research in child development*, pages i–129, 1989.

Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

James Pustejovsky. Dynamic event structure and habitat theory. In *Proceedings of the 6th International Conference on Generative Approaches to the Lexicon (GL2013)*, pages 1–10. ACL, 2013.

James Pustejovsky and Nikhil Krishnaswamy. Generating simulations of motion events from verbal descriptions. *Lexical and Computational Semantics (* SEM 2014)*, page 99, 2014.

James Pustejovsky and Nikhil Krishnaswamy. Voxml: A visual object modeling language. *Proceedings of LREC*, 2016.

James Pustejovsky, Nikhil Krishnaswamy, and Tuan Do. Object embodiment in a multimodal simulation. 2015.

David Randell, Zhan Cui, and Anthony Cohn. A spatial logic based on regions and connections. In Morgan Kaufmann, editor, *Proceedings of the 3rd Internation Conference on Knowledge Representation and Reasoning*, pages 165–176, San Mateo, 1992.

Amitava Roy and Sankar K Pal. Fuzzy discretization of feature space for a rough set classifier. *Pattern Recognition Letters*, 24(6):895–902, 2003.

Iulian Vlad Serban, Tim Klinger, Gerald Tesauro, Kartik Talamadupula, Bowen Zhou, Yoshua Bengio, and Aaron C Courville. Multiresolution recurrent neural networks: An application to dialogue response generation. In *AAAI*, pages 3288–3294, 2017.

Paul R Smart, Tom Scutt, Katia Sycara, and Nigel R Shadbolt. Integrating act-r cognitive models with the unity game engine. In *Integrating Cognitive Architectures into Virtual Character Design*, pages 35–64. IGI Global, 2016.

William D Smart and L Pack Kaelbling. Effective reinforcement learning for mobile robots. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 4, pages 3404–3410. IEEE, 2002.

Amir Tamrakar. Cwc blocks world (bw) apparatus, api reference manual. Technical report, 2015.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Ying Yang and Geoffrey I Webb. Discretization for naive-bayes learning: managing discretization bias and variance. *Machine learning*, 74(1):39–74, 2009.

Jay Young and Nick Hawes. Learning by observation using qualitative spatial relations. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 745–751. International Foundation for Autonomous Agents and Multiagent Systems, 2015.