2016-04-20

# Spatial and Temporal Learning in Robotic Pick-and-Place Domains via Demonstrations and Observations

Russell C. Toris
*Worcester Polytechnic Institute*

# Spatial and Temporal Learning in Robotic Pick-and-Place Domains via Demonstrations and Observations

by

Russell Toris - rctoris@wpi.edu

A PhD Dissertation

Presented at

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

DOCTOR OF PHILOSOPHY

in

Computer Science

April 2016

Approved

# ABSTRACT

Traditional methods for Learning from Demonstration require users to train the robot through the entire process, or to provide feedback throughout a given task. These previous methods have proved to be successful in a selection of robotic domains; however, many are limited by the ability of the user to effectively demonstrate the task. In many cases, noisy demonstrations or a failure to understand the underlying model prevent these methods from working with a wider range of non-expert users. My insight is that in many mobile pick-and-place domains, teaching is done at a too fine grained level. In many such tasks, users are solely concerned with the end goal. This implies that the complexity and time associated with training and teaching robots through the entirety of the task is unnecessary. The robotic agent needs to know (1) a probable search location to retrieve the task's objects and (2) how to arrange the items to complete the task. This thesis work develops new techniques for obtaining such data from high-level spatial and temporal observations and demonstrations which can later be applied in new, unseen environments.

This thesis makes the following contributions: (1) This work is built on a crowd robotics platform and, as such, we contribute the development of efficient data streaming techniques to further these capabilities. By doing so, users can more easily interact with robots on a number of platforms. (2) The presentation of new algorithms that can learn pick-and-place tasks from a large corpus of goal templates. My work contributes algorithms that produce a metric which ranks the appropriate frame of reference for each item based solely on spatial demonstrations. (3) An algorithm which can enhance the above templates with ordering constraints using coarse and noisy temporal information. Such a method eliminates the need for a user to explicitly specify such constraints and searches for an optimal ordering and placement of items. (4) A novel algorithm which is able to learn probable search locations of objects based solely on sparsely made temporal observations. For this, we introduce persistence models of objects customized to a user's environment.

# ACKNOWLEDGMENTS

First and foremost I would like to thank all of my professors, mentors, and advisors throughout my higher-education. I would like to thank Prof. Grant Braught (Dickinson College) for guiding me into the world of research and providing an invaluable foundation in Computer Science. I also thank Prof. Sonia Chernova (WPI / Georgia Tech) for her guidance, teaching, and academic freedom as I explored my area of research and ultimately her support for this dissertation work. Sonia continuously encouraged me through the highs and the lows and exposed me to new opportunities which ultimately shaped me as a person and a research. In addition, I would like to thank Prof. Chad Jenkins (University of Michigan) for not only serving as a committee member, but also as a mentor through an internship, the Robot Web Tools project, and joint research projects. Furthermore, I thank Prof. Charles Rich (WPI) and Prof. Carolina Ruiz (WPI) for their feedback as committee members whose advice and discussions are always much appreciated. Finally, I would like to thank my family and fiancée Gwendolyn for their continued encouragement and support throughout this amazing process.

My ability to complete this work would also not have been possible without the support of grants which help to fund me and my work including National Science Foundation award number 1149876, CAREER: Towards Robots that Learn from Everyday People.

# Contents

# List of Figures

# List of Tables

# LIST OF ALGORITHMS

# CHAPTER 1
# INTRODUCTION

## 1.1 Motivation

Defining or programming complex robotic tasks is a time consuming and challenging process that is not accessible to untrained users. In order for robotics to move out of the laboratory setting and into personal, business, and consumer environments, great strides must be made to overcome such limiting factors. If robots are to be useful in the workplace and around the home, *non-expert* users must be able to intuitively and effectively interact with and teach *high-level tasks* such as setting a table or putting items away.

In recent years, significant progress has been made in both the areas of Learning from Demonstration (LfD) [11, 5] and Programming by Demonstration (PbD) [2, 4] in enabling users to teach robot behaviors; however, many existing techniques require users to procedurally demonstrate the sequences of actions that make up a task in order to learn the policy for behaviors such as table setting [57, 25], single item placement [2, 4], and item stacking [4]. Unfortunately, this form of teaching is time consuming, often requires tedious repetition, and can be difficult to perform correctly [53, 73, 84]. Furthermore, we believe that learning at the task-level may be done at a too fine-grained level, suggesting that perhaps masking these models by learning higher-level or semantically grounded information would be beneficial [77, 78].

An additional limitation of recent work is the reliance on complete task demon-

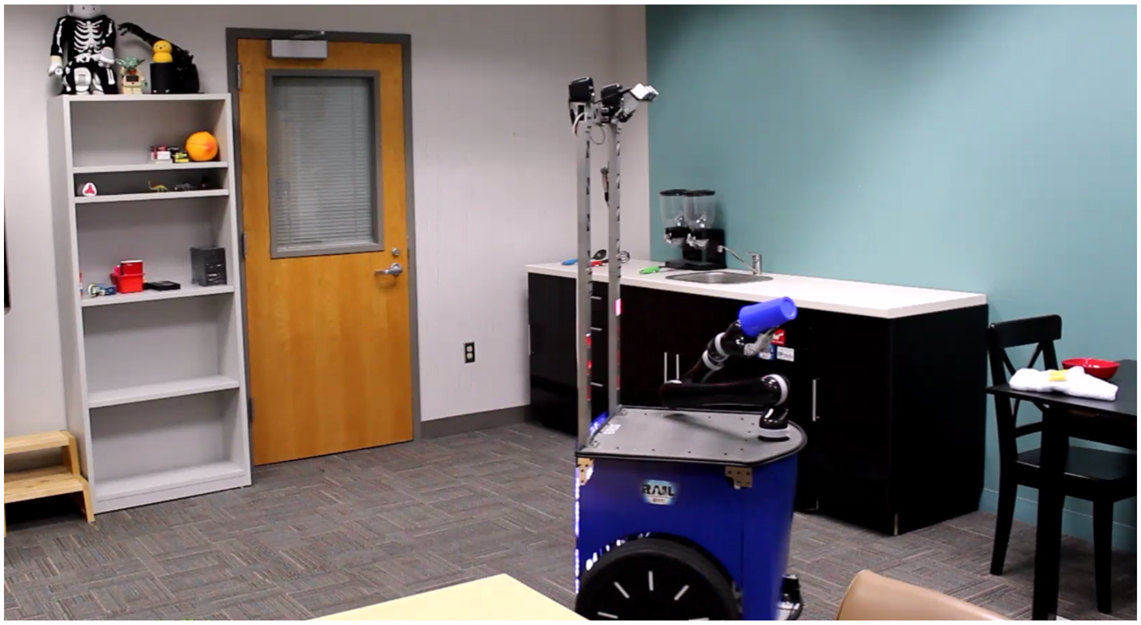


Figure 1.1: A mobile robot completing the high-level task "Set the Table."

strations. By relying on complete demonstrations, users are often faced with performing repetitive and often non-reusable low-level actions in order for the underlying learning method to take advantage of the raw data generated. Our insight is that for many of the tasks being studied in the robotic LfD field today, it is not the sequence of actions or executed trajectory, but the world state at the end of the task that matters most (e.g., setting the table, putting away items in a drawer, or stacking dishes in a cupboard). Such tasks typically fall into a sub-domain of manipulation tasks we define as *wide-area pick-and-place (WAPP)* tasks. We define a WAPP task to be any task which requires the placement of one or more manipulable objects from one area of the world to another, such that the start and end state of the placement may be outside of the robot's workspace (i.e., the robot may move throughout the world). Such a distinction prevents the naive solution of assuming a fixed reference frame such as the robot's base. In such a case, a robotic agent must know 1) where to look for and retrieve the items for a given task, and 2) how and where to arrange the items according to the task.

For such tasks, we believe instead of demonstrating *how* to complete the task, it is more suitable to enable humans to provide the *goal state* of the task. In the case of WAPP tasks, such a goal state corresponds directly to the final placement arrangement. In this context, many challenges emerge such that we are able to obtain sufficient training data and learn a generalized model that enables the robot to perform the task in a wide range of unstructured domains. If such challenges are overcome, the learned task templates can then be executed through the use of existing autonomous planning and grasping methods.

In addition to developing algorithms which address the above, we also note an additional challenge faced in intuitive robot interaction. As explored in earlier work [79, 3] the ability to create web-based interfaces allows for a broader range of users to interact with robots as well as share data. Since the benefits have been explored showing the ability to bring robots to non-expert users (either remotely, crowdsourced, or locally), this work makes use of such techniques for all of its components. A notable limitation in current systems is the lack of research in allowing for bandwidth efficient streaming of robot data to a remote client or server. As we will explore and exemplify later in this work, without improvements in this area, interaction and learning techniques such as those contributed in this thesis are not practical in real-world settings. In is essential to not only develop the underlying learning methods with robotics and LfD, but also exposes them to the masses. As such, we present improvements and new compression techniques developed to enhance the widely used Robot Web Tools project [3, 81] used throughout this work.

## 1.2 Contributions

Throughout this work we aim to show how a robot can leverage **goal state** input from crowdsourced users and sparse observations of its environment to learn and execute WAPP tasks with little input required from a local user. As such, in this thesis work, we introduces a novel framework, called the Spatial and Temporal (SPAT) learning framework, for goal-based LfD which allows a robotic agent to complete WAPP tasks from start-to-finish based on high-level semantically grounded spatial and temporal observations and goal state demonstrations. Such a framework is built using cloud robotics techniques allowing for data collection/processing/visualization, the sharing of high-level data, and interaction for robotic task execution. Our framework makes the following contributions:

1. **Efficient Data Transportation and Messaging for Cloud Robotics** (Chapter 3) As mentioned earlier, since SPAT is built on the foundations of cloud robotics, one of the contributions of our work are methods for dealing with high-bandwidth data and sensor streams to remote clients and servers within robotic domains. Our work builds on the Robot Web Tools[1] project, much of which has been developed during and in complement to this thesis work [79, 82, 80, 3, 81]. Since its official introduction in 2012, the Robot Web Tools project has grown tremendously as an open-source community, enabling new levels of interoperability and portability across heterogeneous robot systems, devices, and front-end user interfaces. At the heart of Robot Web Tools is the *rosbridge* protocol [18, 63] as a general means for messaging ROS (Robot Operating System) [68] data topics in a client-server paradigm suitable for wide area networks and human-robot interaction at a global scale through modern web browsers. Building from *rosbridge*, this thesis describes our contributions within Robot Web Tools to advance cloud robotics through more efficient methods of transporting high-bandwidth topics (e.g., kinematic transforms, image streams, and point clouds). We further discuss the significant impact of Robot Web Tools through a diverse set of use cases that showcase the importance of a generic messaging protocol and front-end development systems for human-robot interaction. This work is invaluable to the development and broadness of our implementation of the SPAT learning framework,

[1] `http://robotwebtools.org/`

allowing for a wider range of interfaces and data collection methods which previously required higher bandwidth restrictions. We note the deviance from the other core contributions of this thesis work, but choose to present it for a more complete representation of the foundation, deployment, and application of such a framework.

2. **Unsupervised Learning of Multi-Hypothesis Goal-State Templates from Spatial Demonstrations** (Chapter 5) In order to tackle the problem of collecting and learning the initial global knowledge base, we contribute a novel method for goal-based LfD which focuses solely on learning from a corpus of potential end states for WAPP tasks. It is important to note that a single example of a goal state is not enough to generalize a solution, as noise from human input or differences in preferences for a given task exist across user groups. Thus, it is critical to rely on a set of gathered data which is collected prior to execution which can be looked up quickly by the robot. Since this task is processed offline and pre-computed, crowdsourced users are utilized to gather large amounts of data. Such a system asks users to place a series of items in a simulated world inside of a web browser according to a predefined task (e.g., “set the table”). This raw placement data is then used to generate multi-hypothesis models in an unsupervised manner via Expectation Maximization (EM) clustering [20] and a novel ranking heuristic based on data aggregated across users for a given task. Such a method ultimately creates a foundational, global knowledge base which can be used during robot execution. This step can be thought of as creating a prior, or initial probability distribution, over the set of possible placements and associated reference frames.

3. **Inference of Inter-Template Ordering Constraints from Coarse Temporal Information** (Chapter 6) The above step results in multi-hypothesis models which rank the strength of the placement of a particular item with

respect to a reference frame. By querying the database for such models, the robotic agent can then determine a suitable *template* for completing the task. In many cases, such information is sufficient for producing the correct result; however, a requirement which breaks this assumption is the need for *ordering constraints*. Therefore, we show how weighting the above heuristic for the clusters by a ratio of coarsely available temporal data can solve such a problem in a number of cases. Furthermore, we show how reducing the problem into a weighted-graph search can provide reasonable execution ordering while maintaining strong spatial relationships. We present strengths and limitations of this approach in both a block-world domain as well as a crowdsourced household setting.

4. **Active Visual Search via Temporal Persistence Models** (Chapter 7) In addition to knowing where and how to place the objects, the robotic agent must also know *where to find the objects*. Therefore, we present a novel solution to the object search problem for domains in which object permanence cannot be assumed and other agents may move objects between locations without the robot's knowledge. We formalize object search as a failure analysis problem and contribute *temporal persistence modeling (TPM)*, an algorithm for probabilistic prediction of the time that an object is expected to remain at a given location given sparse prior observations. We show that probabilistic exponential distributions augmented with a Gaussian component can accurately represent probable object locations and search suggestions based entirely on sparsely made visual observations. We evaluate our work in two domains, a large scale GPS location data set for person tracking, and multi-object tracking on a mobile robot operating in a small-scale household environment over a 2-week period. TPM performance exceeds four baseline methods across all study conditions.

# CHAPTER 2
# BACKGROUND AND RELATED WORK

## 2.1 Learning from Demonstration

To begin, we put the scope of this work into context with a traditional look at Learning from Demonstration. We present LfD in the traditional sense in order to more clearly note the differences in our approach to the problem. Learning from Demonstration, or LfD, is a relatively new field of research within robotics which attempts to gain knowledge from human experts and make use of such knowledge in order for robots to learn new or improve upon existing tasks. One early appearance of the term appeared in 1997 with the work presented by Atkeson and Schaal [6] which established the notion of learning a reward function based on a human demonstration (in their case, for teaching a robotic arm), and computing a policy to reproduce the demonstrated task. The complexity of some machine learning methods were masked by allowing input to come from a human in a more natural manner.

This early work represents only a small fraction of the ideas that have come forth since. Over time, these ideas were expanded upon within the robotics field. One canonical paper that gave way to recent advancements came in 2010 [11]. Here, a more rigorous and formal definition of the methods and terms used in LfD was presented.

Another notable paper is the work by Argall et. al. [5] which presents a large survey of the work in the field of LfD. The ideas of policy learning via human teachers are deeply explored. In [5], demonstration, or input, methods such as teleoperation and imitation via observations and/or sensors are discussed. More importantly, the methods for policy formulation are shown to encompass techniques such as classification or regression.

In contrast to SPAT learning, traditional LfD focuses on the problem of learning state-action pairings. Within our work, instead of utilizing policy learning, we present methods for human LfD which maximize likelihood while maintaining constraints. To see this contrast, we present a brief background on policy-based LfD.

### 2.1.1 States and Actions

As with many machine learning problems, let us consider a world which consists of a state-space $S$. Within this world, are a set of actions $A$, which the agent, a robot in the case of robot LfD, can take. Actions can be represented as low level primitives, or higher level actions, depending on the context of the problem.

Given a current state $s \in S$, and a chosen action $a \in A$, the resulting state $s'$ is determined by the probabilistic transition function $T$ given by:

$$T(s'|s, a) : S \times A \times S \to [0, 1] \tag{2.1}$$

In all real world robotic domains, actions and state observations (via sensors) are non-deterministic. Therefore, we typically rely on statistical and probabilistic models. The agent can never truly know $S$, and therefore we define $Z$ to be the *observable* state [15].

### 2.1.2 Policy Learning

A *policy*, typically denoted by $\pi$, specifies actions that can be taken based on the current *observable* state. A more formal definition of $\pi$ is defined as:

$$\pi : Z \to A \tag{2.2}$$

The goal of traditional LfD is to somehow *learn* this policy for some task $\pi(t)$. In traditional machine learning, a policy is learned via some corpus of data generated, or mined, in a variety of manners. In LfD specifically, we rely on human input and demonstrations.

### 2.1.3 Demonstrations

In order for this policy to be learned, the agent must be able to extract information from a human teacher based on their demonstrations. While many demonstration input techniques exist, such as teleoperation or kinesthetic teaching (i.e., direct movement of the arm [51]), the theoretical, mathematical model remains a constant. Simply put, a single demonstration denoted as $d$, is part of an entire set of demonstrations $D$. Each $d \in D$ is in-turn its own set of $j$ pairings between actions and observable states. This model is defined more formally as:

$$D = \{d_1, \dots d_i\}, \quad d_i = \{\{z_{i1}, a_{i1}\}, \dots, \{z_{ij}, a_{ij}\}\}, \quad z_{ij} \in Z, \quad a_{ij} \in A \tag{2.3}$$

The overall goal of LfD is, when given a set of human demonstrated state-action pairings, how can a policy be learned such that the agent can act semi-autonomously in future situations. In all LfD work, one concept is made clear: in order for robots to learn more meaningful and robust tasks, information from a human expert must be utilized.

## 2.2 Naïve End Users in LfD

Research on robot learning from demonstration has seen significant growth in recent years, but existing evaluations have focused exclusively on algorithmic performance and not on usability factors, especially with respect to naïve users. In this thesis work, we propose methods to abstract the learning problem into high-level or semantically grounded input types.

Previous work done in [73, 84] exemplifies this need for better learning algorithms by conducting an in-depth comparison of three well founded LfD algorithms:

- **Interactive Reinforcement Learning:** A system model policy learning technique in which the reward signal is provided by the human user at runtime. The teacher interacts with the robot by providing reward and guidance input through an on-screen interface [76].
- **Behavior Networks:** A planning-based policy learning technique. The teacher interacts with the robot through kinesthetic teaching, a type of *experienced demonstration*, in which the user physically guides the robot through the task [61].
- **Confidence Based Autonomy:** A mapping approximation technique based on a classification algorithm. The teacher interacts with the robot by selecting the next action or correcting a past action choice through an on-screen interface [16].

Using implementations of the above methods, a simple 3-action task was given to 31 participants who claimed little-to-no robotics experience. Their task was to teach an Nao robot to pick-up randomly walking robotic bugs moving around a table. The users were given 3 preprogrammed actions: picking up a bug at a center pickup location (marked on the table with an '$\times$'), sweeping a bug inwards towards the pickup location, and waiting for 1 second (i.e. doing nothing). The wait action was an important action to include as it was useful in teaching the robot when it was not appropriate to either sweep or pickup. Upon arrival, each participant was given a broad overview of the task that was being trained as well as the capabilities of the Nao. Following the introduction, the participant used each algorithm for 10 minute intervals in a randomized order to train the robot. After each training session the participant was asked to fill out a brief questionnaire relating to the method that they had just used. At the conclusion of their session, a final general questionnaire was given which asked comparative questions about the three methods.

The results show that users were most proficient in and satisfied with teaching using the Confidence-Based Autonomy algorithm. Based on our participants' comments, this is mainly because naïve teachers like to see which actions are available to them and give commands directly to the robot. Although a lot of the participants liked the way they interacted with the robot for Behavior Networks (kinesthetic teaching), they were not happy with the performance of the result. In case of Interactive Reinforcement Learning, participants indicated that they could not see a satisfactory improvement in the action decision of the agent. This was mostly caused by the inherently slow learning rate of the algorithm; however, to the participants, this factor was not obvious. These results tell us that it is important to both keep the transparency of the robot's knowledge to its teacher during the process as well as provide high-level, easy to understand commands as a form of teaching input.

## 2.3 Semantic Mapping

Recent years have seen in increasing growth in semantic mapping within the field of robotics. At its core, semantic mapping deals with mapping human spatial concepts to objects within the world [66]. Many approaches have been made to tackle such a broad and wide scale problem, particularly within the robotics and perception communities [55, 56, 66, 67, 89].

Seminal work by Pronobis breaks down objects to be what he refers to as "spatial concepts" [66]. Spatial concepts can be observational properties (e.g., the item is green) or relationship properties (e.g., the item is in the kitchen). At is core, then, semantic mapping or label becomes assigning some kind of semantic (e.g., human-readable and understandable) label to a spacial entity within the world. Mason goes on to expand and note that by this means, object recognition algorithms themselves are thus one example of a kind of semantic mapping algorithm [55].

Semantic map information can also be used to determine item placement. For example, Mason and Marthi [56] build a semantic world model that enables them to detect and track objects over time, resulting in the ability to model where objects are in the world (e.g., where did you see item $x$), or how objects move around (e.g., where do instances of $x$ go throughout the day). This requires an accurate perception system and ample time for post-processing.

Furthermore, many works in semantic mapping attempt to infer further semantic, or common sense, knowledge about the world using information gained from semantic labels (e.g., inferring the room you are in based on the items you see in the room [67]).

In this thesis work, we present a new contribution in the field of semantic mapping to provide a solution to the object retrieval problem. In contrast to previous works, instead of attempting to infer spatial concepts in regards to locations or rooms, we attempt to make sparse spatial observations of known surfaces in order to determine a semantically grounded model of temporal persistence. These ideas are fully presented in Chapter 7.

## 2.4 EM Clustering of Gaussian Mixture Models

A core component of SPAT learning is based around clustering demonstrations in a spatial domain. That is, the work presented in this thesis makes use of well founded clustering techniques. While numerous techniques exist for clustering [86, 71, 42, 12, 20], this work makes use of EM (Expectation-Maximization) clustering of Gaussian Mixture Models (GMMs) [20, 74, 88]. The decision to use such a method comes from its ability to probabilistically determine the parameter set (in our case, the mean $\mu$ and covariance matrix $\Sigma$ of a set of mulitvariate Gaussian distributions) based on a set of observations. While EM itself is a general algorithm for iteratively finding the maximum likelihood estimate of unknown model parameters, we discuss its use in the domain of GMMs.

Let us assume we have some dataset $X$ such that each element $\{x_1, \ldots, x_N\} \in X$ is a $d$-dimensional real-valued data point. Formally, this means

$$\forall \; x_i \in X \mid i = \{1, \ldots, N\} : x \in \mathbb{R}^d$$

Such a set $X$ is called our *observation set* and each $x \in X$ is called an *instance*. Let us also assume for now that we know our observations come from $K$ different Gaussian models which are referred to as the *components* of the mixture model. Later we present how we can probabilistically determine a guess for an unknown $K$. Each of the $K$ components also has an associated weight $w_k$ used to define the priors of each component (discussed below during probability calculations). Therefore, the parameter set $\Theta$ we wish to estimate via EM are the means $\mu_k$ and covariance matrices $\Sigma_k$ for each of our $K$ Gaussian models. Formally, this gives us

$$\forall \;\; \theta_k \in \Theta \;\mid\; k = \{1, \ldots, K\} : \theta_k = \{\mu_k, \Sigma_k, w_k\}$$

When dealing with GMMs, Equation 2.4 defines the probability density function as a weighted sum of the $K$ component multivariate Gaussian distributions.

$$\begin{aligned} P(x|\Theta) &= \sum_{k=1}^{K} w_k P(x|\theta_k) \\ &= \sum_{k=1}^{K} w_k \mathcal{N}(x; \mu_k, \Sigma_k) \\ &= \sum_{k=1}^{K} w_k \frac{1}{(2\pi)^{D/2}|\Sigma_k|^{1/2}} e^{-\frac{(x-\mu_k)'\Sigma_k^{-1}(x-\mu_k)}{2}} \end{aligned} \tag{2.4}$$

Note that since the weights above are priors and used to normalize, it must be true that $\sum_{k=1}^{K} w_k = 1$.

Furthermore, in addition to our observed data, we also now have unobserved, or *latent*, variables $Z$. In the case of GMMs, our set $Z$ is the "labels" indicating which cluster (i.e., Gaussian distribution) each corresponding instance came from. Therefore, each element $z \in Z$ is a $K$-ary value such that

$$\forall \;\; z_i \in Z \;\mid\; i = \{1, \ldots, N\} : z \in \{1, \ldots, K\}$$

In general terms, the goal of EM then is to find the values of $\Theta$ which maximize the likelihood function $L$ given $X$ and $Z$. Therefore, EM aims to provide a solution to Equation 2.5 since no closed-form solution exists [20].

$$\begin{aligned}\hat{\Theta} &= \operatorname*{argmax}_{\Theta} L(\Theta; X, Z) \\ &= \operatorname*{argmax}_{\Theta} P(X, Z|\Theta)\end{aligned} \tag{2.5}$$

The algorithm works in an iterative manner until convergence occurs. First, the parameter set $\Theta$ is initialized to random values within the domain. The next step that occurs in the *expectation step*. This step (discussed in detail below) determines the likelihood that the observation set $X$ came from the GMM defined by $\Theta$. The next step, the *maximization step* (again, discussed in detail below), then uses the values from the previous step to better estimate new parameter values. These two steps continue iteratively until the values inside $\Theta$ do not change by some pre-defined epsilon value.

### 2.4.1 Expectation Step

In the expectation step of the EM algorithm, the goal is to determine the probability that each instance came from a given component. Therefore, we are attempting to find the probability that latent variable $z_i = k$ given the corresponding instance $x_i$ and the current parameter estimations at iteration $t$ denoted as $\Theta^t$. We denote this membership probability as $\tau_{i,k}$ and is computed for each combination of $i$ and $k$. Equation 2.6 gives the definition of this calculation which follows from Bayes rule [71] and can be further expanded using the logic in Equation 2.4.

$$\begin{aligned}\tau_{i,k} &= P(z_i = k|x_i, \Theta^t) \\ &= \frac{P(x_i|z_i = k, \Theta^t)P(z_i = k, \Theta^t)}{P(x_i|\Theta^t)} \\ &= \frac{\mathcal{N}(x_i; \mu_k^t, \Sigma_k^t)w_k^t}{\sum_{m=1}^{K} w_m^t \mathcal{N}(x_i; \mu_m^t, \Sigma_m^t)}\end{aligned} \tag{2.6}$$

### 2.4.2 Maximization Step

The maximization step utilizes the probabilities found above to increase the likelihood of the observation set by changing the values within $\Theta^t$ at iteration $t+1$. We note that the algebraic proof of this derivation is beyond the scope of this thesis and is available via [20]. Instead, we list the update equations below. At an intuitive level, new weights are calculated in Equation 2.7 by computing the fraction of instances from the observation set which belong to component $k$. The new mean is calculated in Equation 2.8 by computing a weighted average of the points for each component weighted by their membership probabilities from the expectation step. Similarly, Equation 2.9 computes the covariance matrix vary based on how a normal covariance matrix is computed for a multivariate Gaussian (again with similar weighting).

$$\begin{aligned} w_k^{t+1} &= \frac{1}{N}\sum_{i=1}^{N} P(z_i = k|x_i, \Theta^t) \\ &= \frac{1}{N}\sum_{i=1}^{N} \tau_{i,k} \end{aligned} \tag{2.7}$$

$$\begin{aligned} \mu_k^{t+1} &= \frac{\sum_{i=1}^{N} P(z_i = k|x_i, \Theta^t)x_i}{\sum_{i=1}^{N} P(z_i = k|x_i, \Theta^t)} \\ &= \frac{\sum_{i=1}^{N} \tau_{i,k}x_i}{\sum_{i=1}^{N} \tau_{i,k}} \end{aligned} \tag{2.8}$$

$$\begin{aligned} \Sigma_k^{t+1} &= \frac{\sum_{i=1}^{N} P(z_i = k|x_i, \Theta^t)(x_i - \mu_k^{t+1})(x_i - \mu_k^{t+1})'}{\sum_{i=1}^{N} P(z_i = k|x_i, \Theta^t)} \\ &= \frac{\sum_{i=1}^{N} \tau_{i,k}(x_i - \mu_k^{t+1})(x_i - \mu_k^{t+1})'}{\sum_{i=1}^{N} \tau_{i,k}} \end{aligned} \tag{2.9}$$

### 2.4.3 Estimation of $K$

One missing component for this method involves knowing the number of Gaussian components the instances originally came from. Since in many cases it cannot be assumes that the value of $K$ is known ahead of time, methods must be used to provide an estimation of $K$. This work makes use of the techniques presented in [88].

The main idea is to utilize $k$-fold cross-validation to incrementally build GMMs via EM from some training set and compute the likelihood of the data. At each step of the iteration, the number of clusters is increased. Once the likelihood fails to increase (by some pre-defined threshold $\delta$), the number of clusters is determined and EM is run on the entire data set.

The reduce the chance of confusing our $k$ terms in this discussion ($k$-fold cross-validation versus $k$ clusters), we refer to the validation term as $n$-fold cross validation. $n$-fold cross-validation works by randomly splitting the input instances into $n$ equally sized partitions. Of these partitions, $n - 1$ are grouped together as the training set, while the remaining partition is kept aside as the validation set. The training set is then used to compute the GMMs models via the techniques described above. Once finished, the validation set is then used to compute the likelihood (Equation 2.6) of each instance. This process is repeated $n$ times. A full algorithm of this process is given in Algorithm 1. Note that this process may not seem computationally efficient for many problems; however, as presented in Chapter 5, this process is done offline which allows us to utilize such a technique.

## 2.5 Cloud Robotics

Many robotics researchers have utilized Internet and Web technologies for remote teleoperation, time sharing of robot experimentation, data collection through crowdsourcing, and the study of new modes of human-robot interaction. As early as 1995, researchers enabled remote users to control a robotic arm to manipulate a set of colored blocks over the Web [75]. Goldberg enabled web users to maintain and monitor a garden, in which they could control a robot to perform tasks such as plant seeds and water plants [36]. Crick et al. presented remote users with a maze

**Algorithm 1** Cross-Validation for $K$ Cluster Estimation

```
Input: Set of instances $X$
       Number of folds $n$
Output: Optimal number of clusters $k$
 1: $k \leftarrow 1$
 2: $L \leftarrow 0$
 3: $increased \leftarrow$ true
 4: while $increased$ do
 5:     $L' \leftarrow 0$
 6:     for $i \leftarrow [1, \ldots, n]$ do
 7:         $\hat{X}_v \leftarrow \frac{|X|}{n}$ random instances from $X$
 8:         $\hat{X}_t \leftarrow$ the remaining $\frac{|X|(n-1)}{n}$ random instances from $X$
 9:         $\Theta_i \leftarrow$ EM($\hat{X}_t, k$)
10:         $L' \leftarrow L' +$ likelihood($\hat{X}_v, \Theta_i$)
11:     end for
12:     if $L' > L$ then
13:         $L \leftarrow L'$
14:         $increased \leftarrow$ true
15:         $k \leftarrow k + 1$
16:     else
17:         $increased \leftarrow$ false
18:     end if
19: end while
20: return $k - 1$
```

through which they were able to navigate using a robot, enabling the researchers to study how different levels of visual information affected successful navigation [18]. Building on these earlier efforts, a growing number of projects have recently emerged to allow researchers and end-users access to complex mobile manipulation systems via the web for remote experimentation [64, 65], human-robot interaction studies [82, 79], and imitation learning [17].

More broadly, the field of *cloud robotics* research focuses on systems that rely on any form of data or computation from a network to support their operation [47]. Using this definition, a number of cloud robotics architectures have been proposed [38, 46, 87] aimed to offload as much computation as possible onto a “remote brain” unbounded by normal resource limits. One example is the RoboEarth project [87] which utilizes the Rapyuta cloud engine [40]. Rapyuta was developed as a way of running remote ROS processes in the cloud, using a *rosbridge*-inspired protocol of JSON-contained ROS topics.

While the above definition creates a convenient vision of “cloud-enabled robots”, it casts cloud robotics purely as a collection of proprietary back-end services without regard to broader interoperability and human accessibility. In contrast, we consider a more inclusive definition for “robot-enabled clouds” that also considers human-accessible endpoints and interfaces in a network, such as web browsers. With continued advances in robotics-oriented protocols, cloud robotics has the potential to further both human-robot interaction and expanded off-board computation in concert.

As a primary component in our work for both backend and frontend services, this thesis work contributes great improvements to these technologies in the form of efficient messaging. Such a research area has been neglected by many researchers and developers alike preventing benefits of cloud-based solutions. In particular, if such challenges are not overcome, then the benefits of brining HRI and robot LfD to non-expert users is moot.

# CHAPTER 3
# EFFICIENT MESSAGING FOR ENABLING CLOUD ROBOTICS

In this chapter, we present our contributions for efficient data transportation of robotic data in order to further enable cloud robotics. As much of the reminder of this thesis work is built on this framework, we present both an overview of the system as a whole as well as novel contributions within. Furthermore, we present a brief survey of external use cases of this work within the robotics, and more particularly human-robot interaction, research communities.

## 3.1 Introduction

The recent rise of robot middleware and systems software has dramatically advanced the science and practice of robotics. Similar to a computing operating system, robot middleware manages the interface between robot hardware and software modules and provides common device drivers, data structures, visualization tools, peer-to-peer message-passing, and other resources. Advances in robot middleware have greatly improved the speed at which researchers and engineers can create new systems applications for robots by bringing a "plug-and-play" level of interoperability and code reuse. Projects in this space have supported a variety of cross-language and cross-platform tools that promote common functionality and best practices of component-based software design.

Among these efforts, the Robot Operating System (ROS) [68] has become the *de facto* standard middleware for enabling local area network control of robot systems. Developed initially for use with the Willow Garage PR2 robot, ROS is an open-source effort to provide the network protocols, build environment, and distribution release pipelines for researchers and developers to build robot systems and applications without re-writing standard functionality for every application or robot. Since its introduction, the number of individuals, laboratories, companies, and projects using ROS has grown tremendously [29]. This ROS community has also developed important general-use libraries for collision-free motion planning, task execution, 2D navigation, and visualization that have proved invaluable standards in robotics research.

Despite its many advantages, ROS and robot middleware in general has many shortcomings with regards to usability and accessibility, security, portability, and platform dependencies. For ROS, the only officially supported operating system is the popular Linux distribution Ubuntu. Since Ubuntu is used by less than 2% of worldwide computer users[1], ROS remains largely aimed for prototyping lower-level software systems. While there are projects that provide experimental support for operating systems other than Ubuntu, such as OS X and Arch Linux[2], those platforms are not fully stable and require specific support for the intricacies of each OS. As such, ROS in its current form is neither a standard nor sufficient for the development of end-user robot applications for human-robot interaction research [82].

In terms of cloud robotics, ROS largely assumes distributed computing over a local area network (LAN). Though distributed in theory[3], the ROS middleware uses a centralized system for keeping track of peer-to-peer handshaking and glob-

[1]`http://marketshare.hitslink.com/operating-system-market-share.aspx`

[2]`http://wiki.ros.org/indigo/Installation`

[3]From the ROS documentation (`http://wiki.ros.org/ROS/NetworkSetup`): “there must be complete, bi-directional connectivity between all pairs of machines, on all ports.”

ally relevant information, such as kinematic transforms, parameters, robot state, etc. Consequently, typical ROS operation proved problematic for cases that require remote communications over wide area and bandwidth limited networks. Furthermore, recent work such as *rosbridge* [18], remote robotics labs [82, 64], and robotic cloud computing frameworks such as Rapyuta [40] have focused on laying the framework and addressing security concerns [83] and less on efficiency of the underlying compression, filtering, and protocol strategies.

Addressing these shortcomings and others, this thesis contributes analysis of the Robot Web Tools (RWT) project [81, 3] in terms of its ongoing uses and applications across robotics, a presentation of our contributions in advancements for communication of robotics-oriented big data, and discuss future directions addressing improved network efficiency. Initially proposed in [3], RWT is an ongoing effort to realize seamless, general, and implementation-independent applications layer network communications for robotics through the design of open-source network protocols and client libraries. A principal goal of RWT is to converge robot middleware (i.e. ROS) with modern web and network technologies to enable a broadly accessible environment for robot front-end development, cloud robotics, and human-interaction research suitable for use over public wide area networks. Instead of relying on the complex network configuration required by ROS and other cloud robotics paradigms, we have extended the generic *rosbridge* protocol [18] of RWT (described more formally in Section 3.2) that can operate over a variety of network transports, including the web-accessible *WebSockets* and considerably improved the interoperability of robot systems (see Figure 3.1).

In regards to the contribution of this thesis work, this chapter presents: 1) the design of the RWT client-server architecture and use for developing web-accessible user interfaces for robots, 2) technical challenges and improved methods for network transport of high-bandwidth ROS topics, such as for articulated kinematics transforms, point clouds, and image streams, and 3) a survey of broader adoption and use of RWT in applications across robotics research and development.

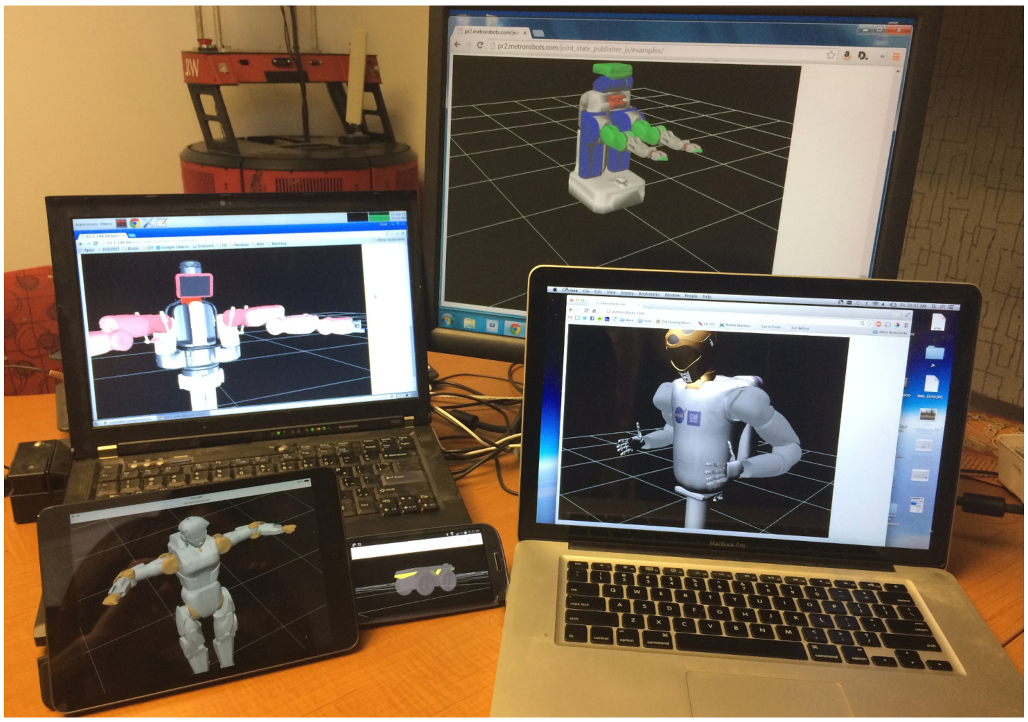


Figure 3.1: A standard Robot Web Tools 3D user interface on multiple operating systems with multiple robots and cloud simulation instances. Clockwise from top right, PR2 on Windows, NASA Robonaut 2 on Mac OS X, Clearpath Jackal on Android, NASA Valkyrie on iOS iPad, and Rethink Baxter on Ubuntu.

## 3.2 Design and Architecture

In contrast to the distributed publish-subscribe architecture of ROS, RWT is designed to be a client-server architecture. The RWT client-server architecture emulates the design of the earlier Player/Stage robot middleware [85, 33] as well as the dominant paradigm for web development. Client-server systems separate a computational workload and resources into services. These services have providers (servers) and consumers (clients), where client processes request services from a server process. A wide variety of services can be provided such as to interface with a device, query for certain information, compute special routines or functions, and more, forming a cascade of services that ultimately lead to the user interface. Similar to the remote procedure call, client-server processes are commonly request-reply interactions communicated by message passing over networks. Broadcast and stream-oriented services, common to publish-subscribe in ROS, can also be supported in the client-server design.

The core element of client-server architectures is an established communications protocol that allows the implementation specifics of services to be abstracted. Such protocols enable clients and services to effectively speak the same language purely through messages. It is this message-based interoperability that has been the engine of growth for the Internet (via the IP protocol) and the World Wide Web (via the HTTP protocol). In contrast, ROS uses an informal protocol (TCPROS) with typed data messages (ROS topics) but no established definition. As such, ROS is more suited to being a build and distribution system for software instead of a general message passing architecture.

The established protocol for RWT is *rosbridge* [18, 64]. *rosbridge* communicates ROS data messages contained in the JavaScript Object Notation (JSON) for straightforward marshalling and demarshalling. Below is a JSON encoded example *rosbridge* message for a 6-DOF velocity command used to move a robot forward:

```
{
  "op": "publish",
  "topic": "/cmd_vel",
  "msg": {"linear":{"x":1.0,"y":0,"z":0},
          "angular":{"x":0,"y":0,"z":0}
}
```

Through its use of WebSockets (a protocol built on top of HTTP), *rosbridge* can be readily used with modern web browsers without the need for installation. This fact, combined with its portability and pervasive use, makes the web browser an ideal platform for human-robot interaction. Working leading up to RWT included a simple lightweight client library known as `ros.js` [63] and a user interface and visualization package known as WViz (web-visualizer) [65, 64]. `ros.js` provided basic ROS functionality to browser-based programs via the early version of the *rosbridge* protocol, known now as *rosbridge* v1.

At RWT's official release, the *rosbridge* v2 protocol[4] added the ability to communicate more efficiently (e.g., specification of compression types) and provide near-complete access to ROS functionality. Previous to this work, RWT allowed for more elaborate, responsive, and reliable interfaces to be built [3] and stricter security measures regarding access by unauthorized remote clients [83].

In this work, to address a wide range of use cases and applications development for HRI, three JavaScript libraries were developed to facilitate web-based human-robot interaction: the *roslibjs* client library and *ros2djs* and *ros3djs* visualization libraries. The design of these libraries enables easy and flexible development of robot user interfaces that are fast and responsive. These libraries avoid both a large monolithic structure (which can be bandwidth intensive like WViz) and a large package system (which breaks out functionality into too many separate libraries like ROS). These cases were avoided principally due to the design of *roslibjs* to be used entirely without any visualization dependencies. Standalone widgets and tools, such as for navigation and mobile manipulation, can be built on top of these libraries and distributed as such, as described below:

*roslibjs* is the client library for communication of ROS topics, services, and actions. *roslibjs* includes utilities for common ROS and robotics tasks such as transform (TF) clients, URDF (Unified Robot Description Format) parsers, and common matrix/vector operations, as well as fully compatibility with server-side JavaScript environments (such as *node.js*).

*ros2djs* and *ros3djs* are the visualization libraries for ROS-related 2D and 3D data types, such as robot models, maps, laser scans, and point clouds, and interaction modes, such as interactive markers [37]. Both libraries build on existing libraries for 2D (*EaselJS*) and 3D (*three.js*) rendering on the HTML5 `<canvas>` element.

[4]`github.com/RobotWebTools/rosbridge_suite/blob/master/ROSBRIDGE_PROTOCOL.md`

To make releases of the libraries easy and efficient to use, pre-built and compressed JavaScript files are uploaded to a public CDN (content delivery network). Such a system makes use of multiple cache servers around the globe to serve clients the libraries as efficiently as possible. These servers manage thousands of hits a week without putting a load on a single failure point. Figure 3.2 showcases typical worldwide hits for the hosted RWT libraries for 2015.

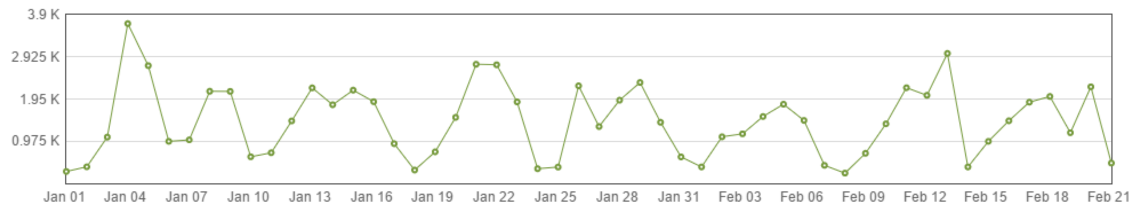


Figure 3.2: The number of HTTP requests to the CDN Libraries for the year 2015.

## 3.3 Efficient Messaging of High-Bandwidth Topics

Although RWT is designed for ease of comprehension and use, the creation of robust and reliable systems has many challenges. Most prominent is the challenge of real-time control. While new emergent technologies, such as WebRTC (real-time communication), are helping to improve efficient communication to and from standard web browsers [10], these technologies are still experimental, in constant flux, and widely unsupported at the time of this writing (these ideas are further discussed in Section 3.5).

The main challenge to real-time communication stems from the amount of data typically associated with streaming robot sensors and visualization information in order to minimize latency and permitting usable supervisory control. Support for "real-time" communication (e.g., for live streaming) in modern web browsers trades-off latency due to buffering for smoother streams. Such buffers add an extra delay (typically in the magnitude of seconds) which cause the robot interface to seem non-responsive and confusing to operators.

Further, due to security and other design constraints, it is not possible to create raw TCP or UDP sockets from JavaScript in a browser. As such, the most common communication approach for bi-directional client-server communication is with a WebSocket connection. WebSockets are built on-top of HTTP and leave an open communication channel to a WebSocket server. As a consequence to this, communication is limited to the efficiency and robustness of HTTP and, thus, unable to send raw or binary data.

Given these constraints, efficient communication of robot data is critical. We discuss advances made by RWT to overcome these limitations for several high-bandwidth message types for transform subscriptions, image streaming, and point cloud streaming, as well as message compression.

### 3.3.1 Transform Subscriptions

In ROS, information about the multitude of reference frames in robot environments is communicated via a stream of transform data, specifying how to translate one frame into another, creating a large, constantly updated tree of geometric relations. This behavior is regulated by a library known as *tf* and *tf2* [28]. Since many of the transforms are likely to be constantly changing and operating on stale data could lead to problematic behaviors, the *tf* library requires that all relevant transforms be constantly published. Any component of the system can publish additional transforms, making the system neatly decentralized.

While the constant stream of data creates a robust environment for many applications, it is not a low-bandwidth-friendly solution. The sheer amount of data published can be overwhelming. As Foote notes, the transforms for the PR2 robot can take up to 3Mbps of bandwidth [28]. Furthermore, much of this data is redundant. For example, if two components of a robot are fixed in relation to one another, the transform between them still needs to be published at the same rate as the rest of the TF tree in order to keep the entire tree up to date.

To reduce the amount of bandwidth devoted to *tf*, we contribute the package *tf2_web_republisher*. This is a ROS node that uses ROS' action interface to pre-compute requested transforms on demand, which can then be published via *rosbridge* with a smaller bandwidth usage. Transforms are published in reference to some specified global frame at a reduced rate, and only when a link is changed within some delta. The inclusion of this node moves computations that usually happen within a specific node to happen within a central authority. The result is the same effective information being provided to remote clients, without the same bandwidth usage.

To illustrate this point, we captured bandwidth usage of TF data over a one minute period using both the traditional ROS TF subscription and the web-based subscription. While the PR2 robot mentioned earlier has been shown to utilize 3Mbps of bandwidth, it can be argued this is due to the complexity of number of DOFs the PR2 contains. Here, a more simplified mobile robot was used with a 6-DOF arm which contained 52 frames. During the capture, the robot's base and arm were teleoperated to trigger TF republishes. For this capture, the web republisher was set to a default rate of 10.0Hz with change delta values of 0.01 meters and 0.01 radians. Figure 3.3 visualizes the results from the capture. Note that the ROS TF rate had an average of 208.5 KB/s ($\sigma = 1.5$) while the web based TF rate was had an average of 96.0 KB/s ($\sigma = 40.6$), resulting in a 54% reduction in bandwidth on average. A $t$-test on this data reports $p \ll 0.001$.

### 3.3.2 Image Streams

Another bandwidth-heavy data stream is any sort of video. In ROS, three main image streaming techniques are used as part of the image stream pipeline: raw images which utilize a ROS image message, compressed images which utilize JPEG or PNG image compression, and Theora streams which utilize the Theora video codec. As mentioned earlier, transport types to the browser are limited. While it would be possible in theory to send raw ROS images over *rosbridge*, the overhead and bandwidth of transporting images in their text JSON representation would be

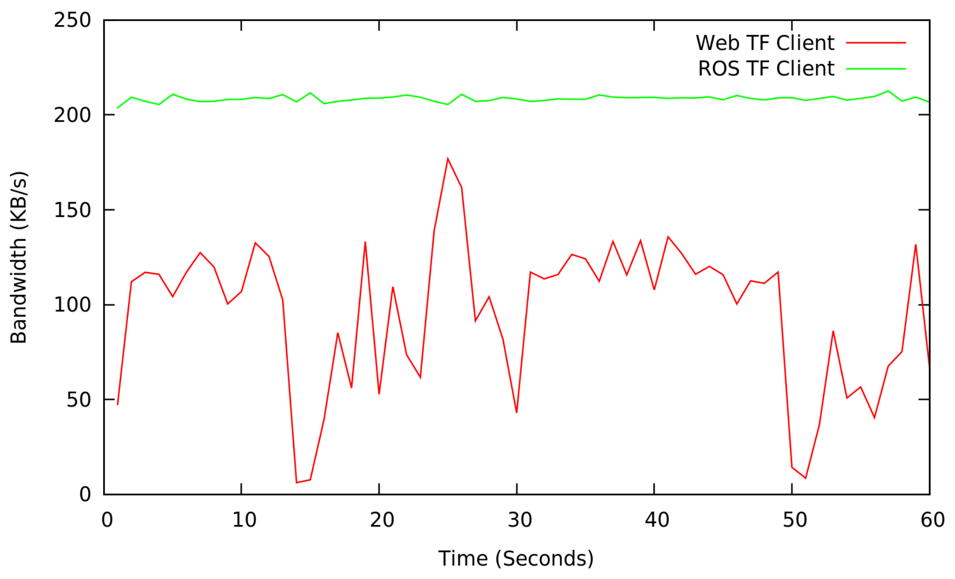


Figure 3.3: The average TF bandwidth from a ROS and web based client.

impracticable.

To overcome this limitation, we introduce two transport options from ROS to the web which are available to all, or a large subset of browsers via the *web_video_server* ROS package. The first option utilizes MJPEG streams which can be embedded in any HTML `<img>` tag. ROS images are transformed into series of JPEG images and streamed directly to the browser. While more efficient options exist, this transport type is available to all modern web browsers. The second streaming option utilizes the newer HTML5 `<video>` tag. This tag allows for more efficient video codecs to be streamed to a browser. In particular, the VP8 codec standard is used within WebM streams. The main limitation here is its limited support on modern browsers (typically well supported on Chrome and Firefox). Furthermore, intentional buffering within the video codecs results in an unavoidable delay or lag in transmission.

To illustrate bandwidth requirements from ROS to a browser, we provide an average bandwidth for multiple transport types across a 2 minute capture using a 640x480 pixel image at 30Hz from a USB camera. MJPEG compression is set to the default value of 90% while VP8 was set to a default bitrate of 100,000 kilobits/second. Results are presented in Table 3.1 which shows an increased performance rate over standard ROS images as well as comparable performance as compared to ROS' natively provided compression streams.

Table 3.1: Bandwidth Usage of Image Streams via Various Transport Types in KB/s

| | Web | | ROS Internal | | |
|---|---|---|---|---|---|
| | MJPEG | VP8 | RAW | Compressed | Theora |
| $\mu$ | 1317.0 | 342.2 | 12052.5 | 1309.3 | 19.4 |
| $\sigma$ | 11.1 | 18.3 | 1.7 | 11.6 | 8.9 |

### 3.3.3 Point Cloud Streams

One of the most bandwidth intensive data types in ROS, robotics, and computer vision are 3D point clouds. This depth information, typically overlayed with corresponding RGB data to form RGB-D data, contains much more information than a video or image stream, making it extremely useful for both computational algorithms as well as visualization. Standard compression types for point clouds are limited and certainly not available to web browsers. As such, new techniques are needed to allow point clouds to be streamed to the browser efficiently for visualization purposes.

As with image streams, while it would be possible to stream raw point cloud data across *rosbridge* in a JSON encoded message, the overhead and bandwidth requirements would be too costly to use effectively, even with message compression. In order to make full use of the built in compression and streaming features available in modern web browsers, we present a method for encoding RGB-D depth maps as a single image stream which can be sent across the wire using HTML5 video codecs and extracted in the browser for visualization.

The key to this idea is that point clouds are not just used for computation, but also visualization by people. As such, lossy compression types can be used to dramatically reduce bandwidth while still maintaining a visually useful display. This compression is done by streaming a single image composed of three separate "images" (shown in Figure 3.4a): encoded depth information from 0 to 3 meters, encoded depth information from 3 to 6 meters, a binary mask indicating valid sample points, and the RBG image itself. Encoded depth information is split into two frames (0-3m and 3-6m) in order increase the range of depth information that can be stored in an image (i.e., a finer discretization). To increase compression rates, compression artifacts are reduced by filling areas of unknown depth (typically represented by `NaN`) with interpolated sample data. On the client side, we then use the embedded binary mask to omit these samples during decoding. This single image is streamed to the web browser over a VP8 codec (via the *web_video_server* mentioned previously) and is assigned to a WebGL texture object with a vertex shader which allows for fast rendering of the point cloud via the clients GPU. An example of the resulting compressed point cloud image is shown in Figure 3.4b.

(a) The Encoded Point Cloud Image

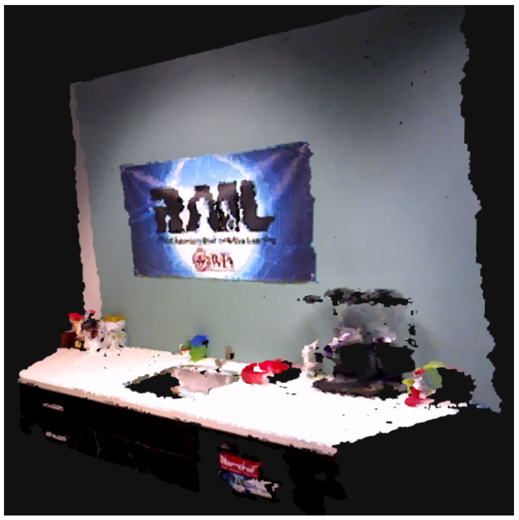

(b) The Resulting Rendering

Figure 3.4: An example of point cloud streaming.

To illustrate bandwidth requirements, we provide an average bandwidth for multiple transport types across a 2 minute capture using an ASUS Xtion Pro RGB-D camera. We again use the default compression values stated earlier. Results are presented in Table 3.2 which show a 90% decrease in bandwidth requirements for web-based point cloud visualization (again with a $t$-test reporting $p \ll 0.001$.

Table 3.2: Bandwidth Usage of Point Cloud Streaming in KB/s

| | ROS Internal | Web Streaming |
|---|---|---|
| $\mu$ | 5591.6 | 568.4 |
| $\sigma$ | 70.5 | 133.5 |

### 3.3.4 Generic Message Compression

Besides transform, image, and point cloud data, there exists many other formats of high bandwidth data in ROS. High resolution map data used for robot navigation is often larger than 2MB per message. 3D visualization and interactive markers [37] often contain large sets of points needed to render 3D models in a visualizer. Furthermore, since user-defined data structures are supported in ROS, a potentially unbounded amount of data could be sent through a ROS message.

To support large data streams in the general manner, we developed RWT to allow for compression options to be set. In particular, we utilize the web browsers built in PNG de-compression to efficiently package and send large messages in a lossless manner. If a client specifies that a data stream be sent using PNG compression, the *rosbridge* server starts by taking the raw bytes of the JSON data (in its ASCII representation) and treat them as the bytes of an RGB image. Padding is added with null values to create a square image. This image data is then compressed using standard PNG compression libraries and sent across the wire using base 64 encoding. This data can then be read internally by the browser and used to extract the original RBG values which, in tern, represent the ASCII JSON encoding of the message itself.

To illustrate the effect this has on message compression, we transmit a series of typical large format ROS messages using PNG compression. Results are presented in Table 3.3 which show the original size in KBs, the compressed size in KBs, and the time to compress and transmit in milliseconds. As a result, it reduces data less than 30% of its original size with negligible time loss.

Table 3.3: Performance of PNG compression for ROS Messages

| | Original (KB) | Compressed (KB) | Time (ms) |
|---|---|---|---|
| Map (732x413) | 2725 | 39 | 40.0 |
| 3D Point Array Marker | 42.1 | 5.1 | 0.8 |
| 3D Line Strip Marker | 42.1 | 4.8 | 0.9 |
| 3D Line List Marker | 78.1 | 7.7 | 1.0 |

## 3.4 Survey of Use Cases

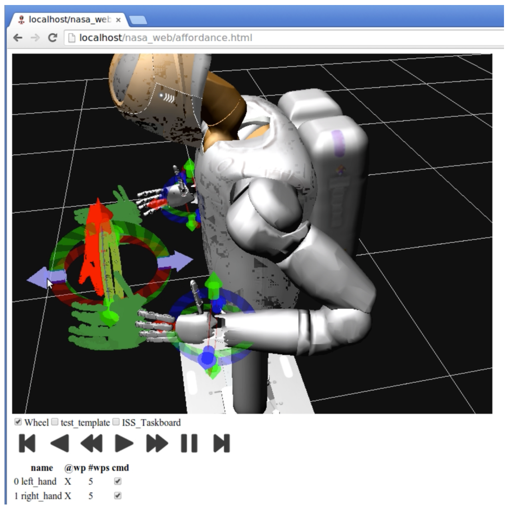
To illustrate the importance of these improvements in progressing robotics, and in particular HRI, research, we contribute a recent survey of recent external use cases outside the scope of this thesis work. With these improvements, RWT has been successfully deployed across multiple robot platforms and a wide range of applications. We posit such significant use demonstrates the need for and impact of open-source network protocols for robotics, human-robot interaction, and cloud robotics. The following is a small sampling of projects the RWT open-source releases in compelling research and development projects.

One of the first widely known applications was for the Robots for Humanity project, which seeks to develop robotic technologies to help people overcome physical disabilities [14]. For such applications, users often require custom interfaces that work on a wide variety of platforms and several design iterations to work around physical constraints, which can more readily be done through RWT than other alternatives. In this project, a person with a mute quadriplegic disability collaborated with a team of researchers to develop interfaces that utilized both the native ROS visualizer and RWT to enable accessible control of a PR2 and a quad rotor to people with disabilities (Figure 3.5a). Through these tools, this user was able operate a robot over the web to shave himself, fetch a towel, open a refrigerator to retrieve yogurt, and inspect his yard. Another independent group of researchers has also applied RWT to the problem of developing a web-based robot control interface for quadriplegics [45]. Utilizing the Baxter robot, this project explores the design of single-switch scanning interfaces for controlling the robot end effector.

RWT has also been applied to general purpose mobile manipulation tasks, with both simulated and real-world robots. Ratner et al. have developed a web-based interface that is capable of quickly recording large numbers of high-dimensional demonstrations of mobile manipulation tasks from non-experts [69]. Coupled with a light weight simulated environment, the tool is capable of supporting 10 concurrent demonstrators on a single server, leveraging existing crowdsourcing platforms to perform large scale data collection. Knowledge obtained in simulation is then transferred onto the physical robot, such as from a simulated to a real-world PR2. We argue that as explored in [82], without the improvements made to cloud and web robotics, acquiring crowdsourced data from non-expert users would be impracticable.

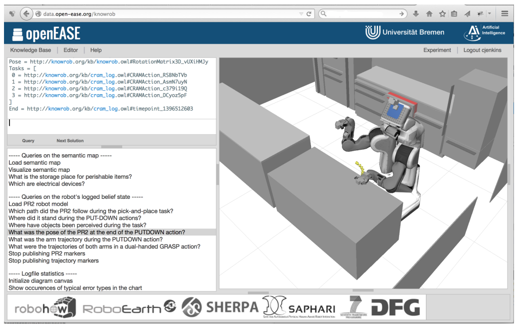
Similarly, the OpenEASE project [9] also makes use of simulation and web-based interface to provide a remote knowledge representation and processing service that aims at facilitating the use of Artificial Intelligence technology for equipping robots with knowledge and reasoning capabilities. Unlike [69], which seeks to advance the mobile manipulation capabilities of a single robotic system from demonstration data, OpenEASE analyzes experiences of manipulation episodes and reasons about what the robot saw, what it did, how it did that, why, and what effects it caused. This information is then stored in a platform-independent way, and can then be retrieved by queries formulated in PROLOG, a general-purpose logic programming language. The system can be used both by humans via a web-based graphical interface, or by robots that use OpenEASE as a cloud-based knowledge base via a WebService API (Figure 3.5c).

(a) Browser-based control of a quad rotor.



```
  name     @wp #wps cmd
0 left_hand  X   5   ✓
1 right_hand X   5   ✓
```

(b) A simulation of the NASA Robonaut 2.



```
Pose = http://knowrob.org/kb/knowrob.owl#RotationMatrix3D_vUXiHMJy
Tasks = [
 0 = http://knowrob.org/kb/cram_log.owl#CRAMAction_RSBNbTVb
 1 = http://knowrob.org/kb/cram_log.owl#CRAMAction_AsmN7uyN
 2 = http://knowrob.org/kb/cram_log.owl#CRAMAction_c379i19Q
 3 = http://knowrob.org/kb/cram_log.owl#CRAMAction_DCyoz5pF
]
End = http://knowrob.org/kb/cram_log.owl#timepoint_1396512603
```

----- Queries on the semantic map -----
Load semantic map
Visualize semantic map
What is the storage place for perishable items?
Which are electrical devices?

----- Queries on the robot's logged belief state -----
Load PR2 robot model
Which path did the PR2 follow during the pick-and-place task?
Where did it stand during the PUT-DOWN actions?
Where have objects been perceived during the task?
What was the pose of the PR2 at the end of the PUTDOWN action?
What was the arm trajectory during the PUTDOWN action?
What were the trajectories of both arms in a dual-handed GRASP action?
Stop publishing PR2 markers
Stop publishing trajectory markers

----- Logfile statistics -----
Initialize diagram canvas
Show occurences of typical error types in the chart

(c) The openEASE project is an example of robot operation and semantic mapping through RWT.

Figure 3.5: Examples of external uses of the RWT methods and technologies.

Additional use cases have been found in commercial applications. Savioke[5], a startup creating a robot hotel butler, uses RWT to bridge between ROS, used to

[5]http://www.savioke.com/

control/monitor the robot, and a web layer, which implements the application logic (e.g. the delivery application) as well as the user interfaces. Rethink Robotics[6] runs *rosbridge* automatically, which enables users to interface with the robot without needing a ROS installed platform.

Furthermore, while JavaScript and the web browser are the chief clients used in conjunction with *rosbridge*, the protocol *rosbridge* provides also creates opportunities for communicating with other platforms that do not have robust ROS implementations of their own. For example, the *jrosbridge*[7] standalone library provides communication with the ROS platform in native Java allowing for ROS communication on a number of different operating systems and architectures. There are also separate projects for using the *rosbridge* protocol to communicate with MATLAB and LabView code. It also provides a low-footprint communication platform for Arudino projects.

## 3.5 Future Technologies

With the continued development of new web standards, the capabilities of web applications continue to grow. New standards allow for web applications to interact more with the hardware of the device that it is running on, create more complex interfaces, and communicate using new robust and efficient protocols. Utilizing these standards can allow for building ROS web applications with advanced features, while still offering the platform independence and ease of use that comes from the web. Additionally, the continued development of these standards reduces the need for plug-ins that users would previously have needed to install (such as Adobe Flash or Microsoft Silverlight) in order to obtain more access to the underlying computer or perform more complex computational and network operations.

APIs that allow for access to the sensors and input devices on a device allows

[6] `http://www.rethinkrobotics.com/`

[7] `https://github.com/WPI-RAIL/jrosbridge`

for interfaces to be built that feel more like a native application. Through the use of new sensor APIs a web application can now have access to the device acceleration and orientation. These sensors are becoming available as the number of portable electronics (smart phones, tablets, etc.) increases. This can allow for interfaces where the user can move their device around to interact, instead of only clicking or touching on the screen. Other APIs also exist to get the state of a device such as battery status, geolocation, and improved touch screen support. These new APIs also open up access to audio and video devices from the web browser. A web application can now stream audio and video from the user's computer's webcam and microphone in addition to streaming video to the browser.

In addition to allowing for more information to be accessed from the browser, new APIs also allow for an application to do things more efficiently and concisely. Some of the APIs that is already used by the RWT client libraries are the canvas and WebGL APIs which allow for high performance 2D and 3D graphics in the browser. Web Workers is another API that allows for running tasks in the background. JavaScript usually runs in a single thread which means that performing a complex operation in JavaScript can cause the UI to lag. Currently if *roslibjs* receives a large number of messages or very large messages it must serialize/deserialize them in the main JavaScript thread, which can block the UI. If Web Workers were used then these operation could be performed in the background and potentially even in parallel allowing for a more responsive UI. Additionally, because web applications can do things in the background it means that they do not have to be a thin client any more and can do processor intensive work.

Furthermore, new web technologies are not just allowing a better web application, but they also allow for better communication with a robot. Initial web technologies were not built for the high bandwidth low latency applications that are being developed today. In order to build an immersive interface for the user, a lot of information (real-time video, point clouds, etc) must be streamed to the user with a minimal delay. As discussed earlier, current solutions for streaming this kind of data in RWT involve streaming the information as a compressed video (MJPEG or VP8) to the client over HTTP. However, because TCP (which HTTP is built on) is a reliable transport protocol, all of the information must be successfully transferred to the client which is not always necessary for responsive visualization purposes. New web standards, such as WebRTC [10], are looking to enable developers to build more media intensive applications. WebRTC allows for the transport of video over UDP and provides automatic adjustment of the video bitrate in response to changing available bandwidth. WebRTC not only allows for the transmission of real time video, but also supports audio, which opens up the possibilities for more feedback to the robot operator. In order to create more interactive applications WebRTC can also support streaming video and audio from the web browser, which could allow for the creation of a telepresence robot controlled purely from the browser.

A critical advance in WebRTC is that it can use direct peer to peer communication even if one of the peers is behind a router. This means that the robot could be hidden behind a NAT device with a server hosting the web application; that is, the robot's computer does not have to be directly exposed to the Internet. The standard not only supports the delivery of media, but also introduces a high-performance data channel API which is specifically designed for low-latency real-time communication. These channels can be either reliable or unreliable and implement congestion and flow control as well as encryption. The WebRTC standard has already been mostly implemented in Chrome, Firefox and Opera and support is planned for more browsers. It is also not just limited to desktop browsers, but is also supported by the mobile versions as well, which allows for the same technologies to be used on any form factor device. Current efforts are under-way to bring such standards into the core RWT libraries and tools allowing for responsive, immersive, and power portable robot web applications.

## 3.6 Conclusion

This chapter presents an analysis, efficient cloud messaging contributions, and discussion on how the Robot Web Tools project has enabled advances in cloud robotics and human-robot interaction research through the use of efficient and easy to use protocols. By overcoming challenges of high-bandwidth data flow for robot applications to the web, researchers are able to better develop and deploy applications and interaction modes to a wider audience on a wider range of platforms and devices as shown in a brief survey of known use cases.

One of the greatest benefits of open source software is the freedom to extend the code to do things that it was never originally planned to do. The work in this chapter has enabled an unprecedented level of freedom for visualizing and interacting with ROS-based systems on a wide variety of new platforms. Running Robot Web Tools through the browser enables easy and quick interfaces on new platforms without the steep learning curve required for writing native ROS clients for each individual operating system.

It should not be understated how valuable having the interface design tool be a language that is as pervasive as JavaScript. With interactive websites growing in prevalence over the past decade, the number of interface developers using JavaScript, HTML5 and the related tools has grown as well. Coupled with the growing capabilities of the modern web browser, web interfaces have become a new standard.

RWT is a continuing and ongoing effort, like ROS and many open-source projects in robotics. We have discussed a number of applications for RWT, many of which were not initially imagined when designing the *rosbridge* protocol or the RWT client libraries. Possible future directions and web technologies were also described that can help RWT improve both cloud robotics and HRI. We encourage further extension and contributions to RWT to help broaden the accessibility and improve the interoperability of robotics for new web and network-based applications.

## 3.7 Acknowledgments

We note that this work was funded in part by the National Science Foundation with grants IIS-1149876, IIS-1208497, and IIS-0844486, the Office of Naval Research with grant N00014-08-1-0910, NASA with grant NNX13AN07A, and the Ministry of Trade, Industry, & Energy (MOTIE, KOREA) Robot Industry Original Technology Development Project NO.10041789. We also acknowledge and thank the various companies and individuals who helped to support and develop portions of this project including Brandon Alexander, Julius Kammerl, Jihoon Lee, Odest Chadwicke Jenkins, Sarah Osentoski, David Gossow, Kaijen Hsiao, Alexander Sorokin, Nils Berg, Ben Pitzer, Trevor Jay, Jonathan Mace, Mitchell Wills, David V. Lu, Sonia Chernova, Willow Garage Inc., and Robert Bosch LLC.

# CHAPTER 4
# SYSTEM OVERVIEW

Before diving into our contributions of the learning components of SPAT, we provide a brief overview of our overall system. Although we provide certain implementation-level details in this overview, in particular how we frame it in a cloud robotics environment, it is important to note that the algorithms and methodologies developed within SPAT are not constrained to cloud systems.

## 4.1 Architecture

Our system as a whole can be broken into several different components. For this discussion, we refer to the high-level system overview depicted in Figure 4.1. At the bottom sit the physical user, the robot, and the environment. Via a browser based interface, the user can visualize the state of the robot and the world (e.g., as seen in Figure 4.2a) and send commands to the robot. Within our system we utilize the tools developed in the Robot Web Tools project for communication [81]. The robot therefore runs *rosbridge* [18] which enables the streaming and receiving of data via a JSON based protocol. We note that, while not done in our experiments, the robot control need not be in a co-present domain. That is, it has been shown that users can alternatively interact with a robot remotely in order to complete certain tasks [82].

Next, we discuss the SPAT computational and storage components which operate in the cloud. By performing these steps in the cloud, we allow both for less processing

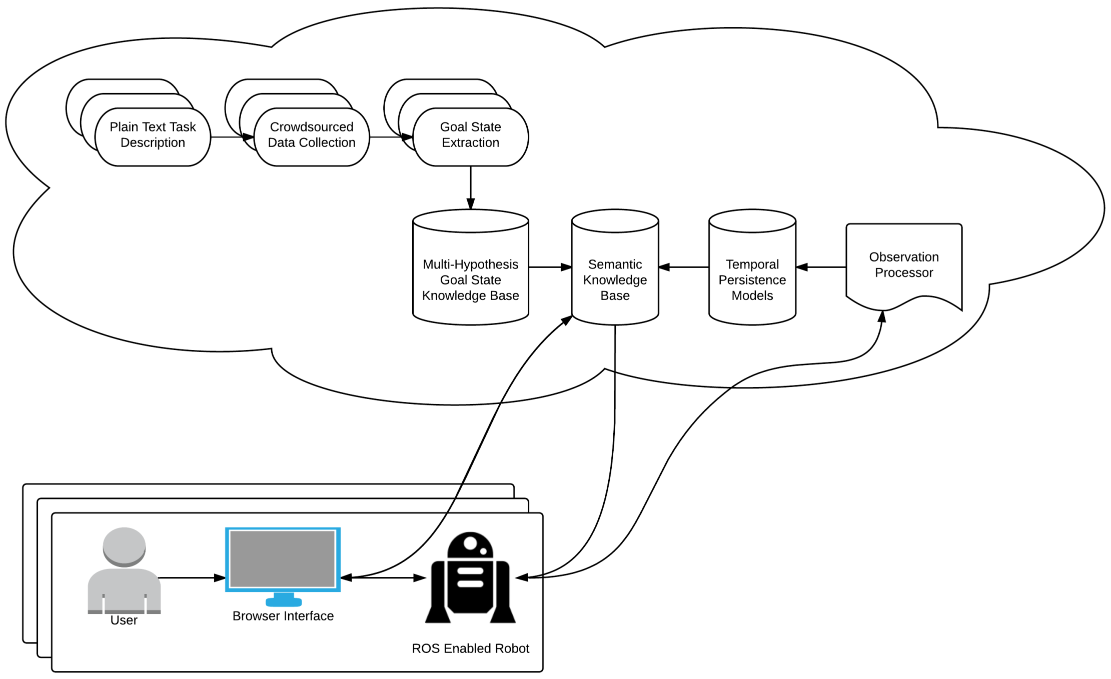
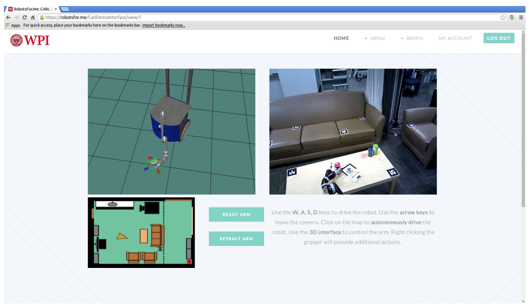

Plain Text Task Description
Crowdsourced Data Collection
Goal State Extraction
Multi-Hypothesis Goal State Knowledge Base
Semantic Knowledge Base
Temporal Persistence Models
Observation Processor
User
Browser Interface
ROS Enabled Robot

Figure 4.1: A high-level system diagram of the SPAT learning framework in the cloud.

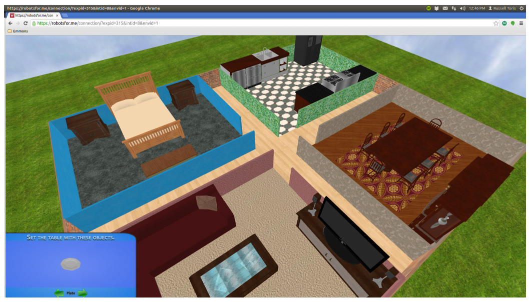
power on the client (e.g., robot) side, as well as the ability to share and re-use learned data across domains and robotic platforms.

To start, we discuss the goal-state learning portion (shown in the top left of Figure 4.1). For this component, we define a general method which starts with the set of loosely-defined tasks (e.g., set the table, put the dishes away, etc.). These tasks are then presented to a large group of crowdsourced users and data is collected on where these users place certain items based on the task description. An example interface, again built on top of the Robot Web Tools platform, is shown in Figure 4.2b. Once a large set of data has been collected, pruning, noise reduction, data transformation, and model training occurs in an *unsupervised* manner via EM clustering and ranking. The resulting templates are then refined to incorporate ordering constraint information before being saved in a JSON encoded format in a remote semantic database. Examples of this format are discussed in Section 4.2.


RobotsFor.Me: CARL
https://robotsfor.me/CarlDemoInterface/view/1
Apps For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...
WPI
HOME
MENU
ADMIN
MY ACCOUNT
LOG OUT
READY ARM
RETRACT ARM

Use the **W, A, S, D** keys to drive the robot. Use the **arrow keys** to move the camera. Click on the map to **autonomously drive** the robot. Use the **3D interface** to control the arm. Right clicking the gripper will provide additional actions.

(a) An example web-based robot control interface.



(b) An example crowd-based demonstration interface.

Figure 4.2: Examples of Robot Web Tools based remote interfaces.

Such template information can then be accessed publicly and remotely via a REST API[26].

The final piece of the system is shown on the bottom right portion of the remote cloud component in Figure 4.1. In order to provide a system which can perform task WAPP task execution from start to finish, the robotic agent must be able to infer probable search locations. To do so, we allow the robotic agent to make periodic spatial observations of its environment. Such observations can occur coarsely throughout the day as it complete its daily duties. We assume that the robot is able to take a "snapshot" of the environment which can then be sent off to some observation processor in an efficient manner. This processor, the main perception component of our system, is able to segment out detected objects in the scene and the relevant surface they are located on [48]. This spatial information, as well as associated timestamp information, is then fed into a set of temporal persistence models in order to create new or update existing priors of where items are located. As presented later in this thesis, we define temporal persistence models to be a model capable of probabilistically determining how long an item remains on a given surface since its last known observation.

With the above information collected and processed, a user can then command the robot to execute one of the pre-defined tasks. The robot can then query the cloud databases for information on how to complete a task for execution. The details of the goal-state learning and a greedy-based execution strategy is presented in Chapter 5, the ordering constraint refinement an an informed execution strategy in Chapter 6, and the object search and permanence modeling in Chapter 7.

## 4.2 Domain

In this section, we present and rigorously define our domain. This representation is used throughout the remainder of this thesis. This also servers as a useful reference point throughout the document.

### 4.2.1 The World

In this work, we are concerned with indoor wide-area-pick-and-place (WAPP) tasks. In this context, let a world be denoted as $W$, which is composed of and defined as a set of pre-defined *areas* (i.e., rooms) $A$ and *items* $I$ defined in a hierarchical manner below. Not that $W$ is not consistent across domains. That is, we do not assume (or rather, we take advantage of) that all real or virtual worlds are defined the same way. We later discuss the representation of a single world instance configuration, but for now, we define $W$ in the general sense.

Specifically, $W$ is defined to contain a set of areas $A$ where each $a_i \in A$ represents some logical area (e.g., a bedroom). It then follows that each area is further decomposed into static sets of *surfaces* or placement locations $S_a$ where each $s \in S_a$ represents some placement location (e.g., a counter-top). Finally, for each $s \in S_a$, there may be a set of points-of-interest (POIs) $P_s$ where each $p \in P_s$ represents a point of interest for the given surface. Examples of POIs are closely related sub-parts of a surface, such as chairs around a table, burners on a stove, or compartments in a shipping crate. By being static in a particular *instance of the world* (defined in the subsequent sub-section), it follows that these entities cannot be moved around. The immutability assumption prevents SPAT from providing solutions that require a surface to be moved. Furthermore, this approach further assumes that $A$, $S$, and $P$ are indeed sets. That is, if there are duplicates, such as multiple end-tables in a bedroom setting, they must be treated independently (e.g., with unique IDs).

Within $W$ are also a set of dynamic items $I$ where each $i \in I$ represents a physical item in the world (e.g., a plate). These dynamic items represent the movable objects within $W$. Finally, we define the set of all *objects* in the world to be $O = S \cup I$.

### 4.2.2 Instance Configurations

For a given $W$, we need to know the current configuration of the world. For our domain, we are concerned with the three-space Cartesian coordinates in terms of $x$, $y$, and $z$ in meters, and the rotation about each axis $\theta_x$, $\theta_y$, $\theta_z$ in radians for each element of $A$, $S$, and $P$. In order for these positions to make sense, a global, fixed coordinate frame must be imposed. We consider the origin of the coordinate frame $\Theta_W$ to be located at the center ($x$, $y$) point in the world located on the ground plane (i.e., $z = 0$). For the given world configuration $W$, each area $a \in A$ is thus expressed in reference to the coordinate frame $\Theta_W$, each $s \in S$ in terms of $\Theta_a$, and each $p \in P$ in terms of $\Theta_s$. Each pose $T$ of any source entity within $source \in W$ in reference to a target entity $target \in W$ can therefore be expressed in a $4 \times 4$ matrix denoted as $T_{source}^{target}$ and defined in Equation 4.1 with rotations in the form of [roll, pitch, yaw] = $[\gamma, \beta, \alpha]$.

$$T_{source}^{target} = \begin{bmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\cos\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma & x \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma & y \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.1)$$

In the case of a 2D domain (i.e., $\gamma = 0$, $\beta = 0$, and $z = 0$ with a single rotation defined as $\theta$), a simplified $3 \times 3$ matrix can be used as defined in Equation 4.2. In any case, references to entities within $W$ can then be made via product combinations of the resulting transformation matrices. This idea is more formally known in the domains of computer graphics as a *scene graph* [39] and within robotics as a *transform tree* [28].

$$T_{source}^{target} = \begin{bmatrix} \cos\theta & \sin\theta & x \\ -\sin\theta & \cos\theta & y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

We note that in a simulated domain (e.g., one used for data collection), this instance configuration information is readily available. In a real-world scenario, this information is obtained from either a pre-defined static or dynamic semantic map of the space in which the location of the various rooms and surfaces is know. Our approach assumes this information is available statically, as obtaining this data is not the focus of this research. Chapter 7, however, tackles the problem of searching for items in $I$ across $W$ at a semantic level.

For compatibility with the *rosbridge* protocol, we represent this data in JSON format. For reference, an example, simple 2D world instance configuration of a single room with a bed in it is defined below.

```
{
  "rooms": [{
    "name": "Bedroom",
    "position": {"x": -2.925, "y": 2.475, "theta": 0},
    "surface": [{
      "name": "Bed",
      "position": {"x": -1.55, "y": -0.875, "theta": -3.14},
      "poi": [{
        "name": "pillow",
        "position": {"x": 0.35, "y": 0.76, "theta": 0}
      }
    }]
  }]
}
```

### 4.2.3 Placements

As with above, each demonstrated placement must be recorded in terms of either its $[x,\ y,\ z,\ \gamma,\ \beta,\ \alpha]$ or $[x,\ y,\ \theta]$ values. For each item $i \in I$ that is placed on a surface $s \in S$ by a user, its transformation matrix $T_i^s$ is stored. That is, the position is stored *only with respect to the surface it was placed on.* We denote the coordinate frame of the surface to be denoted as $\Theta_s$. Furthermore, we denote this single placement demonstration matrix pose $P_i^s$ as a single data point $d$. It then follows that the set of *all* placement demonstrations for a given task is denoted as a set $D$.

We note that the origin of each coordinate frame is arbitrary. That is, the origin could refer to the corner of the object or the center, to name some common examples. The important fact is that these coordinate frames are defined before task execution and can be used across the simulated and real world environments. For our work, we define the origin to be the center of the bounding box about the $X$ and $Y$ dimensions, and on the top surface for the $Z$ value. Positive $Y$ is pointing toward the front of the object and positive $Z$ is pointing up.

Again, for reference, a simplified example of a placement $d$ (in this case a cup placed on a table) in JSON format is provided below.

```
{
  "surface": {"name": "Dining Table"},
  "item":
  {
    "name": "Cup",
    "position": {"x": 0.883, "y": 0.142, "theta": 0.157},
  }
}
```

# CHAPTER 5
# UNSUPERVISED LEARNING OF WIDE-AREA PICK-AND-PLACE GOAL STATES AND REFERENCE FRAMES

In this chapter we present our first contribution in SPAT learning. Based solely on spatial demonstrations of the final goal-state for a pre-defined task, we are able to output a set of multi-hypothesized templates for the task in an unsupervised manner. More importantly, we show how from these spatial demonstrations, the important frame of reference for each item in the demonstration can be inferred without the need of manual annotation from a human teacher.

## 5.1 Introduction

Programming complex robotic tasks is a time-consuming and challenging process that is not accessible to untrained users. Thus users must be able to intuitively and effectively teach higher-level tasks, such as clearing a table or putting items away, in order to effectively customize robot behavior. Existing LfD techniques require users to procedurally demonstrate the sequences of actions that make up a task in order to learn the policy for behaviors such as table setting [57, 25], single item placement [2], and item stacking [4]. Unfortunately, this form of teaching is time consuming, often requires tedious repetition, and can be difficult to perform correctly [53, 84, 73].

Our insight is that for many of the tasks being studied in the LfD field today, it is not the sequence of actions, but the end state of the task itself that matters most (e.g., setting the table, putting away items in a drawer, or stacking dishes in a cupboard). Such tasks typically fall into a sub-domain of manipulation tasks we refer to as *wide-area pick-and-place (WAPP)*. We define a WAPP task to be any task which requires the placement of one or more objects from one area of the world to another, such that the start and end state of the placement may be outside of the robot's workspace (i.e., the robot may move throughout the world). For such tasks, instead of demonstrating *how* to complete the task, we instead rely on the human to provide the *goal state* of the task, that is, the final placement arrangement. We therefore are attempting to learn solely from spatial data. In this context, the challenge becomes to obtain sufficient training data and learn a generalized model that enables the robot to perform the task in a wide range of unstructured domains. The learned task templates can then be executed through the use of existing autonomous planning methods.

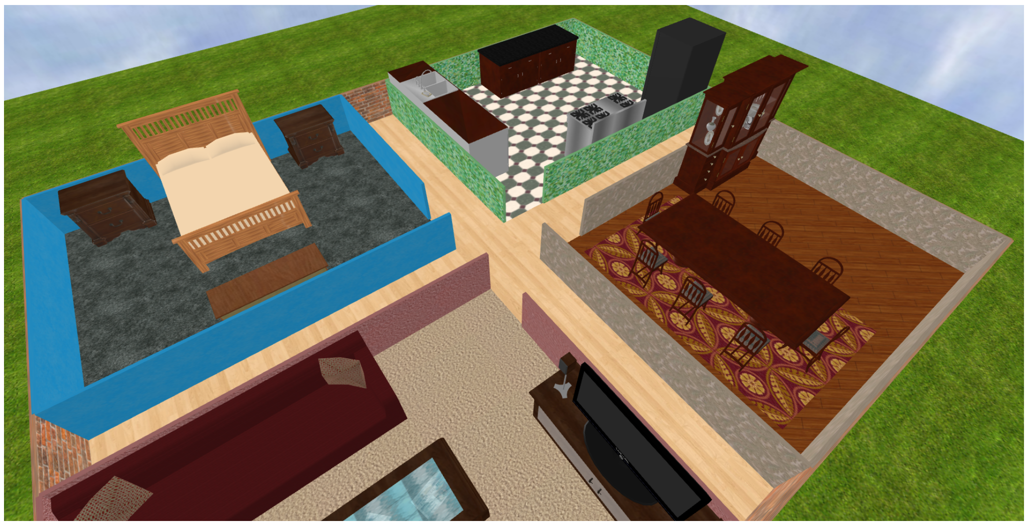


Figure 5.1: A view of the virtual simulated household environment.

We emphasize several sub-problems which are addressed in this work. The first deals with uncertainty within the hypotheses we are trying to learn. Consider, for example, the task of putting away a stack of magazines in a house. Given such a description, where should a robot, or even a human, place the magazines? Depending on the habits and lifestyle of an individual, this could be on the coffee table, the nightstand, or the kitchen counter. A subset or all of the above are valid hypotheses. Therefore, we do not wish to represent just one hypothesis, but to instead provide a multi-hypothesis solution for a task. Selecting only a single hypothesis and disregarding others in such a domain may leave the robot unable to complete the task to the user's satisfaction. For example, a single-hypothesis model that selects the coffee table in the living room would be unable to complete a task such as "put the magazines away in my bedroom" since its only known hypothesis is invalid.

The second sub-problem is a direct result of a solution to the first. In order to collect enough diverse data to learn from, we must rely on input from a large set of users. This ensures that data encompasses many different preferences and configurations. The resulting data, of course, is full of noise (both malicious and bogus input, as well as human errors made in placements). Therefore, the computational method for learning templates must be able to deal with and ignore such noise in the data.

A final sub-problem we wish to address is that of frames of reference, or coordinate frames. During data collection, it is unknown which frame of reference the human is placing the object in respect to (e.g., is the fork being placed next to the cup, or the plate, or the chair?). Therefore, we develop a method for automatically inferring these frames of reference given raw input from the user corpus.

In this chapter, we contribute a novel method for LfD where we focus solely on learning from a corpus of potential end states in WAPP tasks. To allow for multiple placement hypotheses and to overcome uncertainty and error in the human input, we rely on large sets of data obtained from crowdsourced users using a simulated household environment built in the cloud using the RWT framework (Figure 5.1). We ask users to place a series of items in the simulated world according to a predefined task (e.g., set the table). We provide a method in which multi-hypothesis models are generated in an unsupervised manner based on data aggregated across users for a given task. The end result is a system that can take a task description, collect a raw set of placement locations, and provide a set of plausible goal states with appropriate frames of reference for the task. These templates can then be applied to arbitrary domains (e.g., a physical household setting) that need not be identical to the simulated world. A high level outline of this method is shown in Figure 5.2.

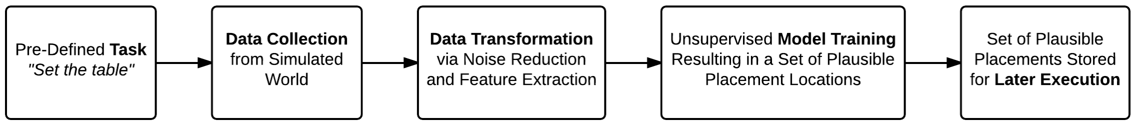

Pre-Defined Task
"Set the table"
Data Collection from Simulated World
Data Transformation via Noise Reduction and Feature Extraction
Unsupervised Model Training Resulting in a Set of Plausible Placement Locations
Set of Plausible Placements Stored for Later Execution

Figure 5.2: A high-level outline of the desired problem solution.

We demonstrate our system on three tasks in a household setting. For each task we collected hundreds of example item placements, and 94% of crowdsourced survey responses agreed with the item placements generated by our algorithm. Additionally, by learning semantically grounded reference frames in a non-domain specific we, demonstrate that task templates learned through crowdsourcing in simulation can be applied directly to a real-world setting.

## 5.2 Background

In general, this work is related to a growing area of research in robotics related to object *affordances*. Originally coined by James Gibson [35], affordances of an object relate to the physical manipulative properties of objects (e.g., turning a doorknob). In our case, we are interested in how items can be manipulated in order to finally be placed in an configuration to form a goal state *template* for a given task.

The idea of learning where to place items within the world has received far less work than that of grasping or picking objects up in a scene. In [41], the authors note the lack of work directly tackling the placement location problem and focus on a method to determine stable placements, in which objects do not collapse or tumble. Furthermore, their work introduces the notion of arranging items that follow *human usage preferences*. The authors collect data from 3D scenes and model object placement using Dirichlet process mixture models. In order to learn usage preferences, human poses were "hallucinated", or envisioned within the scene to estimate how objects in scenes might be related to human poses. Unlike our work, the referenced work focuses on the idea of stable placements, for example how to arrange items on a shelf such that they remain stationary. Our work instead deals with generalizable arrangement templates.

Semantic map information can also be used to determine item placement. For example, Mason and Marthi[56] build a semantic world model that enables the tracking and detection of objects over time, resulting in the ability to model where objects are located in the world (e.g., where did you see coffee cups), or how they move around (e.g., where do coffee cups go throughout the day). This requires an accurate perception system and ample time for post-processing. Such a system learns location properties, while our approach is designed to learn goal states for particular tasks which can later be executed.

Human demonstrations have also been used to teach robots where to place objects in an interactive manner [2, 4]. In these works, human teachers guide a robot manipulator through the execution of a manipulation task. The user requests the recording of either key-frames during the sequence or full continuous joint trajectories. These sequences can then be replayed at a later time to execute the learned task. Many other approaches have tackled manipulation problems in a similar manner and have proven to be both intuitive and useful [5]. Unlike our approach, such methods require the human demonstrator to guide the robot through an entire task. In contrast, our insight is that in the case of *pick-and-place*, a subset of the problems that the above work can handle, many motion planners can automatically generate the trajectories required to execute the task, eliminating the need for repeated demonstrations of lengthy trajectories.

Work done by Joho et. al. [43] attempts to reconstruct scene information based on 3D observation data. The authors develop a method for creating a probabilistic generative model of the scene which can be used to compare against partially constructed scenes. That is, given information obtained via several observations of a given scene, the system is able to infer given an incomplete scene which objects are missing and where. Unlike our work, we are further interested in not only understanding what might be missing from a scene, but multiple definitions of the scene itself.

The work which most closely resembles our own is that of Chung et. al. [17]. While not explicitly dealing with WAPP tasks, the authors enable a robot to reconstruct 2D block patterns from a corpus of crowdsourced end-state configurations. The goal of their work is to develop an online learning technique that gathers crowdsoured data during task execution which is used to assist the robot when learning from a single co-present user. An example scenario discussed in the paper includes having a human label the classes of the model (i.e., what each color of block represents), building an initial model, collecting and learning from a large set of crowdsourced configurations, and presenting the user with three possible final configurations of the pattern which the user can select from. Our work seeks to both learn from unlabeled data in an offline manner, as well as target the much broader problem of WAPP tasks in human domains.

## 5.3 Methodology

We now present our approach which is composed of three main tasks (a high-level flowchart depicting these steps is shown in Figure 5.3):

- **Data Collection**: The processes of collecting the possible hypotheses for a task description.
- **Data Transformation**: The processes of extracting important information and autonomously eliminating irrelevant data from the collected data.

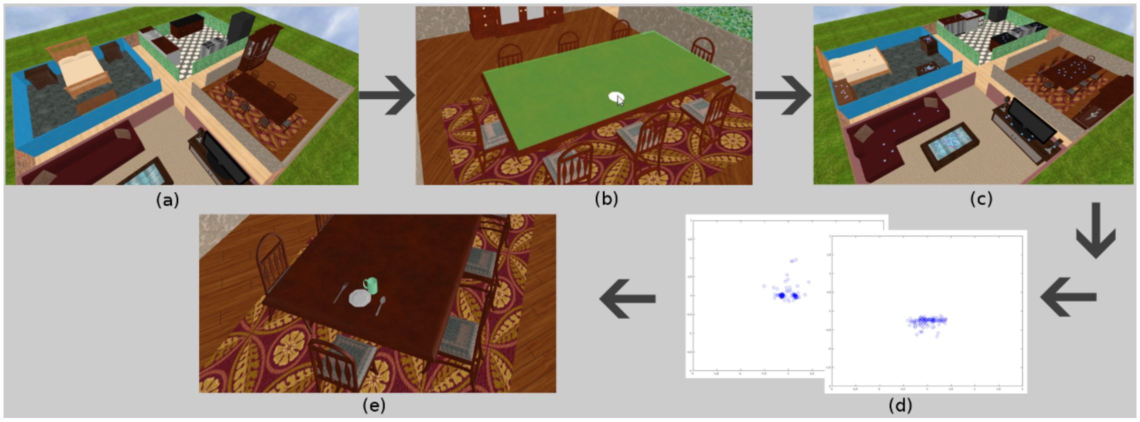


Figure 5.3: Methodology Flow Chart: Any empty world **(a)** is first populated with placements via crowdsourced users **(b)**. The noisy set of all placements **(c)** is then transformed to form candidate models against multiple possible frames of reference **(d)**. Using unsupervised learning, we then extract a multi-hypothesis model for the task **(e)**.

- **Model Training**: The processes of training from the set of transformed data to form a set of possible goal states for a task.

### 5.3.1 Data Collection

For each task, our goal is to collect a set of item placement examples that is diverse enough to capture the preferences of a wide variety of potential users. To achieve this goal, we crowdsource our data through the CrowdFlower micro-task market and use the Robot Management System (RMS) [79, 80, 82] for managing the users and data collection. We presented each user with a set of items and a brief textual task description, such as setting the table (actual task descriptions are given in Section 5.4). We did not provide any example item arrangements to avoid biasing the users.

Using a web browser, each user could select an item in set $I$ associated with the given task. Hovering over a possible placement surface $S$ caused the surface to appear green to indicate that a placement could be made (Figure 5.3b). Clicking on a location on the highlighted surface resulted in a recorded item placement demonstration $d$ in a cloud database. Each user was required to place a given number of each item somewhere in the world. Once all data has been collected, the data is loaded, transformed, and trained, as described below.

### 5.3.2 Data Transformation

At the end of the data collection step, we now have a *set of demonstration datasets* which we denote $D$ where each dataset represents a user's placement points for an item on a surface. We can further break up subsets of $D$ and refer to them as $D_i^s$ where each $d \in D_i^s$ is the transformation matrix representing the placement of an item $i \in I$ on a surface $s \in S$. At this point, we note that there exists a dataset $D_i^s \, \forall \, (i, s) \in I \times S$ (with the possibility that a given $D_i^s = \emptyset$). That is, we know each placement in reference to the surface it was placed on; however, it is possible that the coordinate frame of the surface is not the frame of interest. Consider, for example, a table setting task. Here, it is likely that the fork placement would be relative to the plate instead of the table. If we only model data based on $i$ with respect to $\Theta_s$, this information would never present itself. Therefore, we enumerate over the possible combinations of reference frames to expand and form multiple *datasets* of placements.

More formally, we want to create a new set of datasets $\hat{D} \mid D_i^o \in \hat{D} \, \forall \, (i, o) \in I \times O$. Each data set consists of the position of all instances of a given item (e.g., all spoons) with respect to a single object (e.g, the table) aggregated across all users.

We thus have $|S|+|P|+|I|$ possible reference frames and therefore $\frac{|S|+|P|+|I|!}{(|S|+|P|+|I|-2)!} - \frac{|S|+|P|}{(|S|+|P|-2)!}$ permutations to generate and evaluate, each of dimensionality 6 ($x$, $y$, $z$, $\gamma$, $\alpha$, and $\beta$). Note that the $\frac{|S|+|P|}{(|S|+|P|-2)!}$ term comes from the fact that we do not need to generate datasets of surfaces and POIs in reference to each other since these objects are considered static. To avoid extracting too much unnecessary data, we make a set of constraint assumptions:

1. *References cannot be across areas.* It is unlikely that the placement of an item

in one area is relative to the location of another object in a different area. For example, if a user places a spoon in the kitchen sink, it is unlikely this was in relation to the stack of magazines on the living room coffee table.

2. *References cannot be across surfaces.* Building upon the logic from the above assumption, we also eliminate inter-surface relationships. While it is possible to think of a relationship that would be across a surface in the world (e.g., placing a remote on the coffee table since the table itself is in-front of the TV), we argue that the relationship here is actually between the two surfaces. That is, typically the coffee table is between the couch and the TV, which is why the remote (which is placed on the coffee table) appears to have a relationship with the TV. Since surfaces are immutable in our domain, we thus only look at the relationships on a given surface.

3. *References are only with respect to the nearest POI type.* Each surface $s$ contains a set of POIs $P_s$ (with the possibility that $P_s = \emptyset$). While each POI $p \in P_s$ is unique, it is possible there are multiple similar-typed POIs (e.g., $P_{\texttt{kitchen-table}} = \{\texttt{chair}_1, \texttt{chair}_1, \ldots, \texttt{chair}_n\}$). Instead of looking at the relationship of each placement to each $p \in P_s$, we instead look at the nearest POI of a given type (e.g., the nearest chair).

4. *Rotations can only be made about the Z axis.* For many problems, rotations about other axes are irrelevant. Therefore, we ignore rotations about any other axis thus reducing our dimensionality to 4 ($x$, $y$, $z$, and single rotation $\theta$). Note that our user interface also made use of this assumption and disallowed rotations about the remaining two axes.

5. *Objects are placed on level surfaces.* If we assume each placement surface is level, then the $z$ coordinate becomes irrelevant. We note that this may violate certain ordering constraints for execution (e.g., a napkin under a fork); however, we explore this problem more deeply in Chapter 6.

The above assumptions thus reduces our dimensionality to 3 ($x$, $y$, and $\theta$). Furthermore, our permutation reduces to $\frac{1+|P_s|_{\text{unique}}+|I|!}{(|P_s|_{\text{unique}}+|I|-1)!} - \frac{1+|P_s|_{\text{unique}}}{(|P_s|_{\text{unique}}-1)!}$ possibilities. We note the important fact that the above assumptions are made merely to reduce the cardinality on the number of datasets to generate and evaluate in further steps as well as the dimensionality. In some domains, these assumptions may be invalid. Thus, this pruning step can be ignored without affecting any of the prior or subsequent steps in our methodology. That is, they can be removed without loss of generality on the described methods and instead the complexity of the models is increased potentially requiring more data and processing time for reasonable convergence.

We present an algorithm for our expanded dataset generation in Algorithm 2 which takes the set of raw placements stored in $D$ and transforms them into a set $\hat{D}$ where each $\hat{d} \in \hat{D}$ is a 3-dimensional *dataset* (e.g., representing all spoon placements with respect to the plate). Note that each placement $d \in D$ contains the surface it was placed on, $s_d \in S$, the item that was placed, $i_d$, and the transformation matrix recording the pose of the placement in the $\Theta_{s_d}$ coordinate frame. Furthermore, we use the notation $\hat{D}_i^o$ to denote the data set that represents all placements of item $i \in I$ with respect to object $o \in O$ (i.e., in the $\Theta_o$ coordinate frame).

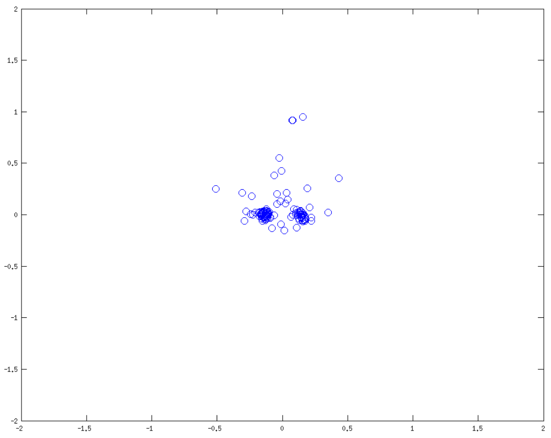
The algorithm starts by taking each $d \in D$, that is each raw placement demonstration of an item for all placements across all users, and extracts the item $i_d$ and surface $s_d$ from $d$. In the first for loop (lines 6-10), we iterate through each additional type of item $i \in I$ such that an instance of $i$ exists on $s_d$. For example, consider $i_d$ to be a spoon and $s_d$ be the table. If our set of raw placements $D$ contains instances of forks and plates on the table, but not cups, this for loop would only consider $i_{fork}$ and $i_{plate}$. Within this loop, we store the $x$, $y$, and $\theta$ of $d$ (e.g., our spoon placement) with respect to the closest instance of $i$ (e.g., the spoon with respect to the closest fork and the closest plate). This location is added to the data set $\hat{D}_{i_d}^i$. That is, the data set representing all spoons with respect to plates. Figure 5.4 shows examples of what such data sets look like in the $x - y$ plane.

**Algorithm 2** Data Generation and Pruning

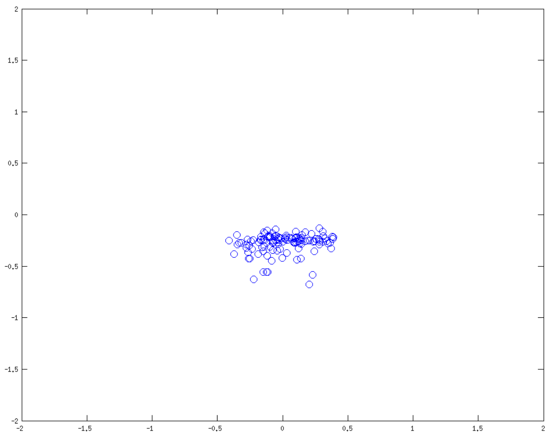
```
Input: Set of demonstration datasets $D \mid d \in D$
Output: Dataset of transformed placement datasets $\hat{D}$
1: $\hat{D} \leftarrow \emptyset$
2: for $d \in D$ do
3:     $s_d \leftarrow$ surface $d$ is placed on
4:     $i_d \leftarrow$ item placed in $d$
5:     $T_{i_d}^{s_d} \leftarrow$ transformation matrix of $i_d$ with respect to $s_d$
6:     for $i \in I | i \neq i_d$ and $i$ has been placed on $s_d$ at least once do
7:         $T_i^{s_d} \leftarrow$ closest placement of $i$ to $i_d$ on $s_d$
8:         $T_{i_d}^{i} \leftarrow (T_i^{s_d})^{-1} \; T_{i_d}^{s_d}$
9:         $\hat{D}_{i_d}^{i} \leftarrow \{\hat{D}_{i_d}^{i} \cup T_{i_d}^{i}\}$
10:    end for
11:    for $p \in P_{s_d}$ do
12:        $T_p^{s_d} \leftarrow$ closest instance $p$ to $i_d$ with respect to $s_d$
13:        $T_{i_d}^{p} \leftarrow (T_p^{s_d})^{-1} \; T_{i_d}^{s_d}$
14:        $\hat{D}_{i_d}^{p} \leftarrow \{\hat{D}_{i_d}^{p} \cup T_{i_d}^{p}\}$
15:    end for
16:    $\hat{D}_{i_d}^{s_d} \leftarrow \{\hat{D}_{i_d}^{s_d} \cup T_{i_d}^{s_d}\}$
17: end for
18: return $\{\hat{d} \in \hat{D} | (|\hat{d}| \geq \delta * |D|)\}$
```

In the second for loop (lines 11-15), we repeat the above technique in reference to the points of interest related to the current surface $s_d$. For example, we consider the spoon placement $d$ with respect to the closest chair $p_{chair}$ from the table. On line 16, we copy and store the actual placement location of $i_d$ with respect to $s_d$ (i.e., the spoon with respect to the table). Our final step on line 18 provides further pruning by discarding any sets in which the number of points is not within some $\delta$ percentage of the total number of placements. For our implementation, we set $\delta = 0.05 = 5\%$. This enables us to eliminate outliers, such as a spoon placed on the bed.

In summary, these processing steps have expanded our set of placements by considering additional frames of reference, and pruned any under-represented data sets. The final set $\hat{D}$ represents the set of potentially valid item placements with respect to candidate references $\Theta_o$. Again, figure 5.4 shows examples of these models transposed into their $(x, y)$ values.



(a) $i_{\texttt{Fork}}$ in $\Theta_{\texttt{Plate}}$ Space



(b) $i_{\texttt{Fork}}$ in $\Theta_{\texttt{Chair}}$ Space

Figure 5.4: Data sets from $\hat{D}$ for a table setting task.

### 5.3.3 Model Training

Given the data in $\hat{D}$, our goal now is to determine the possible hypotheses for placements and their frame of reference for the task at hand. To achieve this goal, we introduce a new method which makes use of a combination of clustering and ranking to produce a priority of possible hypotheses. The hypotheses are trained using unsupervised learning and ranked using a ranking function $\pi$, where smaller values denote a tighter fitting model.

We start by training a model $\lambda_{\hat{d}}$ based on each corresponding data set in $\hat{D}$. In our evaluation, we use Gaussian mixture models trained via EM where the number of clusters is determined via $k$-fold cross validation with $k = 10$ [20, 88]; however, others could be used as well. This transforms each data set $\hat{d} \in \hat{D}$ into a probabilistic GMM $\lambda_{\hat{d}}$. We let $\lambda_{\hat{d}}^{\Theta_o}$ and $\lambda_{\hat{d}}^{i}$ denote the frame of reference and the item placed in this model, respectively. Using a GMM makes the assumption that placements are placed with noise being representing by some hidden Gaussian model. As such, we can use the mean of our resulting Gaussian models to be the most-likely placement location of $\lambda_{\hat{d}}^{i}$ with respect to $\lambda_{\hat{d}}^{\Theta_o}$.

Now, given the set of all trained models $\Lambda$, we then can make the following query: given the current state of objects of the world $\bar{O} \subseteq O$ and an item we wish to place $i \in I$, where should the item be placed? The solution is calculated by finding the corresponding GMM in Equation 5.1 where $\lambda'$ is the model with the smallest rank value for all models which represent the target item $i$ with respect to potential target object $o$ already present in the world. This step can be thought of as a form of an entropy calculation within the models themselves [88].

$$\lambda' = \operatorname*{argmin}_{\lambda \in \Lambda} \ \pi(\lambda) \ \mid \ \lambda^{\Theta_O} \in \bar{O}, \ \ \lambda^{i} = i \tag{5.1}$$

For this work, our ranking function $\pi$ (Equation 5.2) is a heuristic devised to favor dense, populated clusters where $\Delta_c$ is the distance between all points to each other in cluster $c$ within the GMM. Again, by making the Gaussian assumption, to determine the placement location within each model, we choose $\mu_x, \mu_y, \mu_\theta$ inside the cluster $c$ which minimized Equation 5.2.

$$\begin{aligned} \Delta_c &= \sum_{i,j|i \neq j} |c_i c_j| \quad i,j = 1, \ldots, |c| \\ \pi(\lambda_{\hat{x}}) &= \underset{\text{cluster } c \in \lambda_{\hat{x}}}{\operatorname{argmin}} \frac{\Delta_c}{|c| * (|c| - 1)} * \frac{|\lambda_{\hat{x}}|}{|c|} \end{aligned} \tag{5.2}$$

The above technique produces a single hypothesis. That is, the most likely placement location for the most likely frame of reference. If instead we want to allow for multiple hypotheses, instead of returning $\lambda'$, we return an ordered list of $\lambda \in \Lambda$ where $\pi(\lambda) \leq \delta$ for some pre-defined threshold $\delta$.

### 5.3.4 Lack of Symmetry

A common question asked is if there is redundant information in the modeling technique. More specifically, one might wonder if the model for item $i_1$ in reference to $i_2$ is the same as the reference for item $i_2$ in reference to $i_1$. While in a noise free environment this is likely to be true (results from Section 6.3.1 actually show this), in a noisy environment this is not the case. The main reason behind this is due to the assumption on line 7 of Algorithm 2.

To illustrate this, consider the simple placement example in Figure 5.5 in a 1D domain. According to Algorithm 2, for each item, we add a data point for the location of that item in relation to the closest unique item. For our data set $D_{i_1}^{i_2}$, we would have two data points: one with a value of 2 units, and one with a value of 1 unit. A Gaussian model would likely converge around the midpoint of 1.5. For our data set $D_{i_1}^{i_2}$, we would have the single point of 1 unit length which is different than its inverse relationship model. Given larger and larger data sets and demonstrations, this effect is likely to occur more drastically.

We note that while techniques could be conceived to attempt to address this directly, we show how our methodology can overcome noise within the system resulting in appropriate final results.

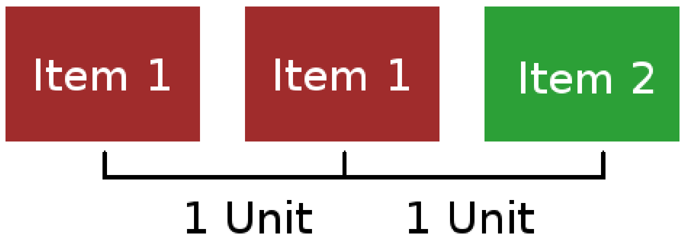


Figure 5.5: An example of a non-symmetrical placement model.

## 5.4 Evaluation

To verify our methodology, we present a set of three experiments meant to encompass a set of possible task types:

1. **Single Item Placement** In the first, simplistic case, we define a task in which a single item is placed somewhere in the world. In this experiment, we asked users to place a stack of magazines in an appropriate location.

2. **Multi-Item Uncoupled Placements** The second case consists of a set of items that should be placed, but their inter-item relationships are unimportant (as opposed to a templated solution). For this, we asked users to place a set of dirty dishes away.

3. **Multi-Item Template Placements** The third case concerns multiple items that should form some kind of template. Not only is their placement in the world important, but relationships between items are also important. For this, we asked users to set the table.

For each user, the state of the world $W$ (i.e., area and surface locations) was randomly assigned to one of 16 possible configurations. Our world consisted of 4 areas, 13 surfaces, and 18 POIs (Figure 5.1). At the start of the session, remote users logged into the simulated environment and were presented with a general task description. Users could then navigate around the world and place items onto a surface by using their mouse. Each experiment is now presented in detail.

Furthermore, with these results, we exemplify our ability to filter out erroneous demonstrations. For example, Figure 5.6 shows the entire demonstration set $D$ for the Multi-Item Template Placements (i.e., table setting) task. Solely from this data, and in an unsupervised manner, the methods discussed above is able to provide a reasonable grounding for a typical table setting in a completely unsupervised manner.

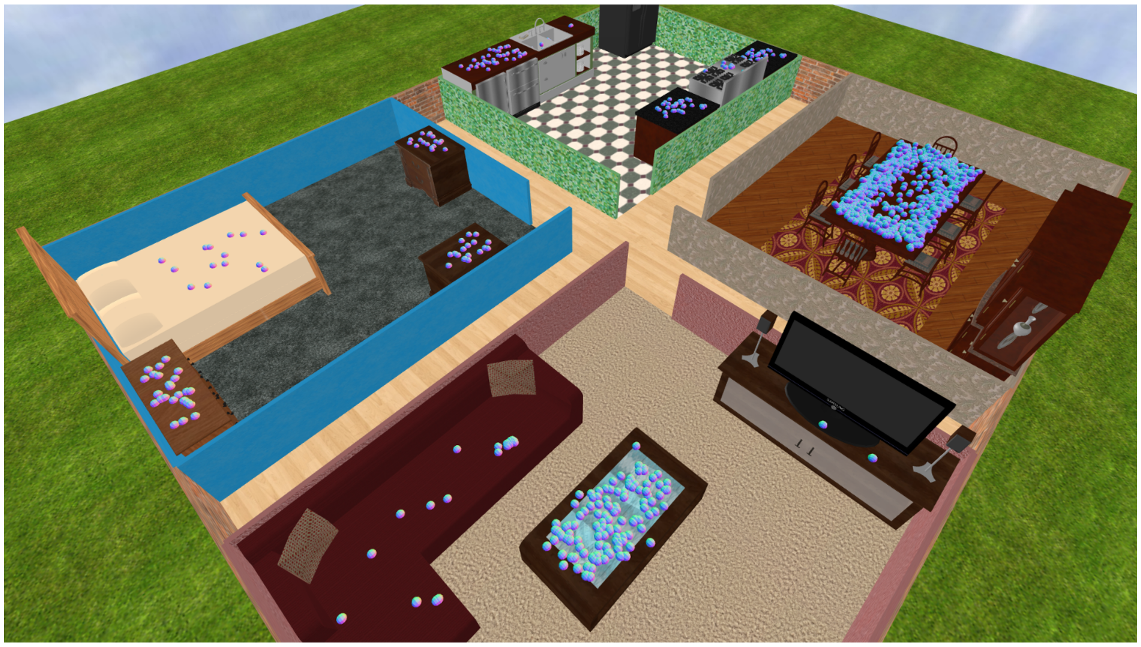


Figure 5.6: The noisy demonstration set of all placements for the table setting task.

### 5.4.1 Single Item Placement

Remote users were presented with the following task description: *Your job is to place the given object in an appropriate location in the house.* The item provided was the magazines ($I = \{I_{\texttt{Magazines}}\}$). We collected 516 placements for this task, and, after transforming and pruning the data with Algorithm 2, obtained 4 data sets in $\hat{D}$. We then ran the training process defined in Section 5.3.3 and used a threshold of $\delta = 1.1$ for our ranking function (found via iterative tuning).

Table 5.1: Models Obtained from the Magazine Task

| $\lambda^i$ | $\lambda^{\Theta_o}$ | $\pi(\lambda)$ |
|---|---|---|
| $I_{\texttt{Magazines}}$ | $S_{\texttt{NightStand}}$ | **0.2373** |
| $I_{\texttt{Magazines}}$ | $S_{\texttt{CoffeeTable}}$ | **0.6669** |
| $I_{\texttt{Magazines}}$ | $S_{\texttt{DiningTable}}$ | 1.3198 |
| $I_{\texttt{Magazines}}$ | $P_{\texttt{Chair}}$ | 2.7769 |

Table 5.1 presents the hypotheses obtained for this task. Two valid hypotheses were found (bold) relative to the nightstand and the coffee table. Extracting the placement locations for these models obtains the visualization shown in Figure 5.7.

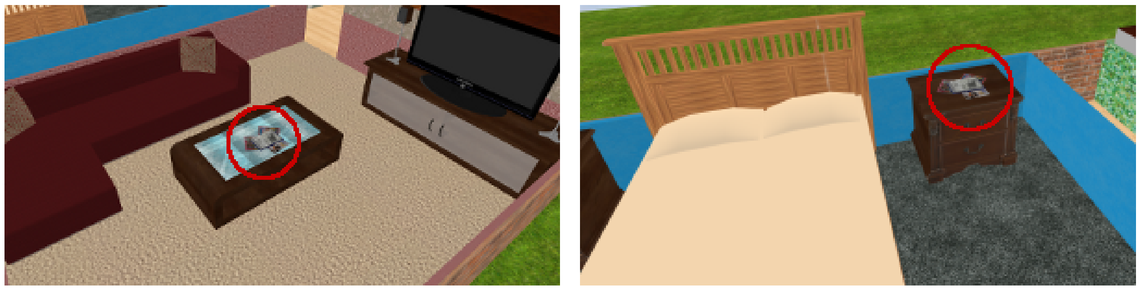


Figure 5.7: Hypotheses extracted from the magazine task.

### 5.4.2 Multi-Item Uncoupled Placements

For this task, users were presented with the following task description: *Your job is to place the dirty dishes in an appropriate location in the house.* The following items were provided for placement: $I = \{I_{\texttt{DirtyMug}}, I_{\texttt{DirtyPlate}}, I_{\texttt{DirtySpoon}}, I_{\texttt{DirtyFork}}\}$. We recorded 1744 item placements, which resulted in 22 models after transformation and training. Table 5.2 lists all valid hypotheses found.

Table 5.2: Valid Hypotheses from the Dishes Task

| $\lambda^i$ | $\lambda^{\Theta_o}$ | $\pi(\lambda)$ |
|---|---|---|
| $I_{\texttt{DirtyMug}}$ | $P_{\texttt{SinkTub}}$ | **0.3934** |
| $I_{\texttt{DirtyMug}}$ | $S_{\texttt{SinkUnit}}$ | **0.3934** |
| $I_{\texttt{DirtySpoon}}$ | $S_{\texttt{SinkUnit}}$ | **0.4468** |
| $I_{\texttt{DirtySpoon}}$ | $P_{\texttt{SinkTub}}$ | **0.4468** |
| $I_{\texttt{DirtyFork}}$ | $I_{\texttt{DirtySpoon}}$ | **0.7651** |
| $I_{\texttt{DirtySpoon}}$ | $I_{\texttt{DirtyFork}}$ | **0.8975** |
| $I_{\texttt{DirtyPlate}}$ | $I_{\texttt{DirtyMug}}$ | **1.0096** |
| $I_{\texttt{DirtyMug}}$ | $I_{\texttt{DirtyFork}}$ | **1.0329** |

Note here that there could be multiple starting points for this placement pattern. That is, for any of the models whose $\pi(\lambda)$ value is less than our threshold $\delta$, we could choose to place that item first and continue from there. This introduces a notion of planning that could be used with these models during task execution (i.e., item placement). We further explore this idea in Section 5.6 and for now consider the model with the smallest $\pi(\lambda)$ value to be our starting point. This results in the model depicted in Figure 5.8.

We note an important benefit of our approach at this point. As can be seen in Figure 5.8, many of the items overlap in the final result. As once might expect with multi-item uncoupled placements, this is a likely occurrence. Of course, when given to a robot for execution, it is likely the robot would fail to plan such placements as collisions would be unavoidable. However, since each resulting placement is modeled by a cluster within a GMM, instead of feed a single point to the robot during placement execution, we can instead provide the entire Gaussian distribution to the robot. This would allow the robot to sample around this distribution for a

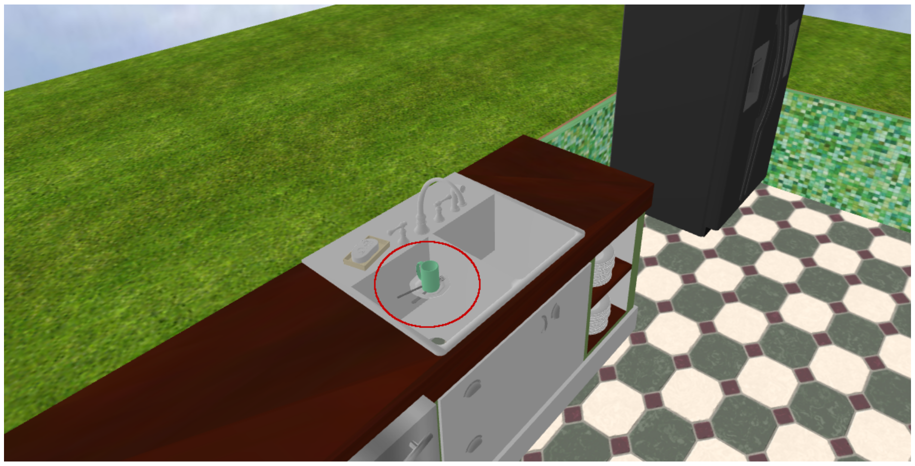


Figure 5.8: A Hypothesis for the Dishes Task

collision-free placement.

### 5.4.3 Multi-Item Template Placements

For this task, remote users were presented with the following task description: *Your job is to set the table with the given objects.* The objects provided were as follows: $I = \{I_{\texttt{Cup}}, I_{\texttt{Plate}}, I_{\texttt{Spoon}}, I_{\texttt{Fork}}\}$. We recorded 678 placements, which resulted in 20 models after transformation and training. Valid hypotheses are listed in Table 5.3.

Using the same method discussed during the dishes task for a starting point, an interesting result emerges: no objects in $S$ or $P$ are selected for reference frames. That is, the resulting placements for all items are solely related to the other items in $I$. As expected from the problem formulation of this task, this is the template of items we were hoping for. The template, shown in the left of Figure 5.9a, clearly depicts a typical table setting.

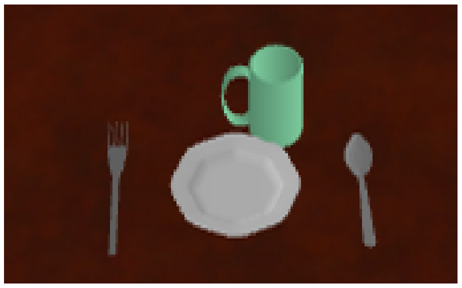
The results here relate to the idea that a table setting itself is really independent of a surface. Perhaps you eat at your counter, or even on a TV-dinner tray in your living room. No matter where the set of objects is placed, it still can be identified as a table setting arrangement.

To build upon this idea, suppose we enforce the above selection method to select

Table 5.3: Valid Hypotheses from the Table Setting Task

| $\lambda^i$ | $\lambda^{\Theta_o}$ | $\pi(\lambda)$ | $\lambda^i$ | $\lambda^{\Theta_o}$ | $\pi(\lambda)$ |
|---|---|---|---|---|---|
| $I_{\texttt{Spoon}}$ | $I_{\texttt{Plate}}$ | **0.2077** | $I_{\texttt{Cup}}$ | $I_{\texttt{Plate}}$ | **0.7585** |
| $I_{\texttt{Fork}}$ | $I_{\texttt{Plate}}$ | **0.2470** | $I_{\texttt{Plate}}$ | $I_{\texttt{Spoon}}$ | **0.7745** |
| $I_{\texttt{Spoon}}$ | $I_{\texttt{Fork}}$ | **0.4045** | $I_{\texttt{Cup}}$ | $I_{\texttt{Spoon}}$ | **0.7769** |
| $I_{\texttt{Spoon}}$ | $I_{\texttt{Cup}}$ | **0.4183** | $I_{\texttt{Plate}}$ | $I_{\texttt{Cup}}$ | **0.7777** |
| $I_{\texttt{Fork}}$ | $I_{\texttt{Spoon}}$ | **0.4548** | $I_{\texttt{Cup}}$ | $I_{\texttt{Fork}}$ | **0.9169** |
| $I_{\texttt{Plate}}$ | $I_{\texttt{Fork}}$ | **0.4992** | $I_{\texttt{Fork}}$ | $P_{\texttt{Chair}}$ | **0.9736** |
| $I_{\texttt{Fork}}$ | $I_{\texttt{Cup}}$ | **0.5530** | $I_{\texttt{Plate}}$ | $P_{\texttt{Chair}}$ | **1.0084** |

some object $o \in O | o \in \{S \cup P\}$. If we re-run the selection with the restriction, we see that typically table settings, in particular the plate, are in reference to a chair around the table. Once the plate has been placed with respect to the chair, the template then follows as normal. The result is shown in the right Figure 5.9b. The template could also be further reproduced around other objects in the world (e.g., all chairs), to further complete a more complex task.


(a) The standalone table setting template.

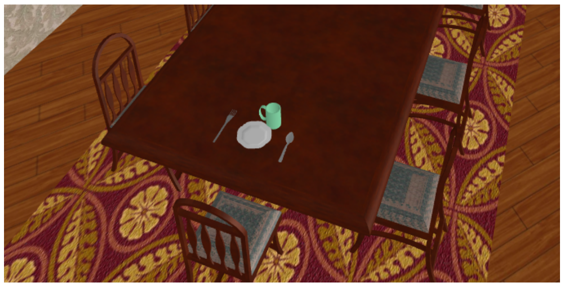

(b) The table setting template positioned about the chair reference.

Figure 5.9: The resulting table setting template.

### 5.4.4 Validation Results

To further verify our results, we ran a small user study in which we presented 20 anonymous users with three separate questions: one for each task. In each question, the user was given a set of images, some of which were the results shown above, and some of which were incorrect (e.g., placing the magazines on a stove burner). Given the original task descriptions presented to the crowdsourced users during data collection, we asked users to select which, *if any*, of the images accurately depicted the completed task. Note that users were not required to select any models as valid if they did not find them to be so.

For the magazine task, users were given an image of the two valid hypotheses (night stand and coffee table), and two invalid answers (stove burner and sink). For the dishes task, users were presented with the first possible hypothesis (Figure 5.8), as well as two incorrect possibilities (random arrangements of the dishes on the dresser and on the couch). Finally, for the table setting task, users were given the stand-alone template (Figure 5.9a), the template in reference to the chair (Figure 5.9b), and a random arrangement of the dishes on the TV unit. The results from the questionnaire are presented in Table 5.4 (choices representing algorithm-generated hypotheses are shown in bold, italic font).

As we can see from the results above, the resulting hypotheses from the proposed methodology match the human expectation of the task description 94% of the time. What is interesting to look at as well is the magazines task. Here, we see that the users typically selected *both* of the hypotheses derived from this work, supporting our use of multi-hypothesis methods.

## 5.5 Failed Clustering Heuristic

For the sake of completeness of this document, we also present a brief description of a failed alternative to our ranking function $\pi$ in Equation 5.2. Notable in clustering research is the use of scatter separability criterion [31]. The idea is to define two

Table 5.4: Validation Questionnaire Results

| **Task** | **Choice** | **# Resp.** |
|---|---|---|
| Magazine | ***Coffee Table*** | 19 |
| | Sink | 0 |
| | ***Nightstand*** | 15 |
| | Stove | 2 |
| Dishes | Dresser | 3 |
| | Couch | 1 |
| | ***Sink*** | 20 |
| Table Setting | ***Template & Chair*** | 20 |
| | ***Template*** | 17 |
| | TV Unit | 0 |

matrices, $S_w$ defined in Equation 5.3 which measures the within-cluster scatter, and $S_b$ defined in Equation 5.4 which measures the between-cluster scatter. Here, we define $\nu_j$ to be the prior probability of an instance belonging to cluster $\omega_j$, and $\Sigma_j$ is the sample covariance matrix for cluster $\omega_j$ [22].

$$S_w = \sum_{i=1}^{k} \nu_j \mathbb{E}[(\mathbf{X} - \mu_j)(\mathbf{X} - \mu_j)^T)|\omega_j] = \sum_{i=1}^{k} \nu_j \Sigma_j \tag{5.3}$$

$$S_b = \sum_{i=1}^{k} \nu_j (\mu_j - M_o)(\mu_j - M_o)^T \tag{5.4}$$

$$M_o = \mathbb{E}[\mathbf{X}] = \sum_{i=1}^{k} \nu_j \mu_j \tag{5.5}$$

The general idea is to reward low inter-cluster scatter (i.e., to favor tight clusters) and high between-cluster scatter (i.e., to favor clusters which are far apart). Typically, the trace function, shown in Equation 5.6 for an $n$-by-$n$ matrix $A$, is used to measure this desire. In particular, the value returned by the evaluation function is defined as $\mathrm{tr}(S_w^{-1}S_b)$. This evaluation function therefore was the return value for our ranking function $\pi$.

$$\mathrm{tr}(A) = a_{11} + a_{22} + \ldots + a_{nn} = \sum_{i=1}^{n} a_{ii} \tag{5.6}$$

We note that even though such a ranking is probabilistically sound and shown to work in many domains, our methodology failed to converge on consistently reasonable results. While the idea of within-cluster scatter is analogous to our ranking method, the addition of between-cluster scatter into the ranking deters from a solution in our domain. That is, for our ranking, we need not weight against models where two clusters are located near each other (e.g, different arrangements of the cup around the plate reference frame).

### 5.5.1 Robot Execution

We note that in order to complete the task execution, the robotic agent must be able to formulate not only where the items should be placed, but in what order. As shown in the following Chapter, a more informed search problem can be formulated to solve this problem, but without any type of ordering constraint, a greedy algorithm is able to provide this ordering. A general greedy execution algorithm is given in 3.

The algorithm starts at line 1 by getting the set of all items needed in the task definition followed by the current state of the world in line 2. Note that the state of the world initially consists of all surface locations within the world as no items have been placed yet. Line 4 then searches for the model with the smallest ranking value such that the models item are in the item set $I$ and the models reference frame are in the world $W$. In other words, we have yet to place $\lambda^i$ and we are able to place it in reference to $\lambda^{\Theta_o}$. We then search for and retrieve $\lambda^i$ before placing it. We then remove $\lambda^i$ from the set of items to be placed (line 7) and add the item to the world state as a potential reference frame in the future (line 8). This process continues until all of the items have been placed.

**Algorithm 3** Greedy Template Execution

```
Input: Set of placement models $\Lambda_t \mid \lambda \in \Lambda_t$
1: $I \leftarrow$ set of items for the task
2: $W \leftarrow$ state of the world
3: while $I \neq \emptyset$ do
4:     $\lambda \leftarrow \text{argmin}_{\lambda \in \Lambda_t} \pi(\lambda) \mid \lambda^{\Theta_o} \in W \ \wedge \ \lambda^i \in I$
5:     retrieve($\lambda^i$)
6:     place($\lambda^i, \lambda^{\Theta_o}$)
7:     $I \leftarrow I \setminus \{\lambda^i\}$
8:     $W \leftarrow \{W \cup \{\lambda^i\}\}$
9: end while
```

In our final form of evaluation we executed the resulting table setting task template on a physical robot. For our robot, we utilized the CARL (Crowdsourced Autonomy and Robot Learning) robot seen in Figure 5.10. CARL consists of a Segway RMP base with a 6 degree-of-freedom JACO arm. 3D object recognition and manipulation for the task's objects were previously trained on the robot such that it knew how to detect and manipulate the objects required for the task. Since information from the above methods is expressed in a high-level, semantic, and spatial way, new environments can make use of such information to complete task execution. During execution, CARL successfully queries the system for what a table setting template is, finds the necessary objects on the kitchen counter, and sets the table appropriately. A video of this task is available at `https://www.youtube.com/watch?v=Pqjgd33ZVAk`.

## 5.6 Conclusion

In this chapter, we have presented a new methodology and algorithm for learning item placements, templates, and reference frames from a corpus of noisy human

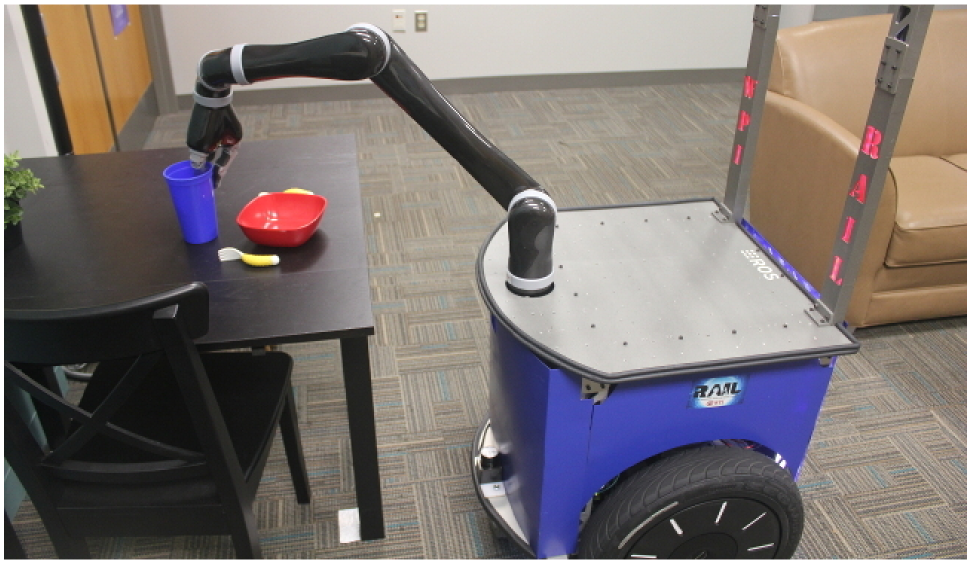


Figure 5.10: The result of the robotic table setting task.

demonstrations based solely on spatial data. Furthermore, no prior knowledge about the important frames of reference were given to the models, adding to the complexity of possibilities that must be considered during unsupervised learning.

Using our proposed methodology to transform and learn on the data generated from crowdsourced users, we were able to validate our framework for three different pick-and-place tasks: single item placement, multi-item placement with no inter-object relationships, and multi-item placement with a possible templated solution.

This work paves the way for a new form of robot PbD or LfD in which end states can be leveraged as spatial descriptions with learned reference frames instead of explicit demonstrations of task execution. The learning method is intuitive and easy to use since all data was collected from non-expert, anonymous crowdsourced users using intuitive, web-based interfaces. We note that extensive background research in robot learning has lead to little similar work. As discussed in Section 5.2, recent work has begun to notice and address this obvious gap.

We now move on to expand this work to deal with the notably missing ordering constraints.

## 5.7 Acknowledgments

We would like to acknowledge and thank David Kent for his support for enabling CARL with reliable pick-and-place capabilities in the robot execution portion of this work.

# CHAPTER 6
# UNSUPERVISED ORDERING INFERENCE VIA TEMPORAL WEIGHTING

In this chapter we expand our SPAT framework to incorporate its first form of temporal data in order to strengthen the goal state templates. In particular, we use coarsely represented temporal data from the demonstration set itself to infer ordering constraints within a template. Furthermore, we present a method for producing an informed execution order of the template for the agent during runtime. These methods are explored in both a series of block-world domains as well as a noisy household domain.

While the effectiveness of the spatial learning method above has been explored in Chapter 5, this chapter explores how the model $\Lambda$ can be further weighted to incorporate temporal ordering constraints (e.g., the napkin must be placed *before* the fork is placed on-top of it).

Ordering constraints, or more broadly, constraint satisfaction, is a widely explored field within Computer Science and Artificial Intelligence research [60, 71, 27, 34]. In its most general form, constraint satisfaction is a method in which one can find a solution to problem given a set of conditions that must be met. In many cases, these constraints are known or inferred ahead of time.

In the context of the spatially defined template models developed in Chapter 5, a constraint would need to be defined if some target item must be placed either before or after a particular reference item. Intuitively, it might then follow that, given a particular final template, the agent should execute the template in the same order that the user placed the items. However, such a claim then must come from the assumption that the user, or in the case of large datasets, *any given user*, placed the items both correctly, and in the correct order. That is, given the unsupervised nature of the collection process, combined with the lack of raw-demonstration data lost during the aggregation and transformation process in Algorithm 2, we do not know the template and thus the potential order of placements until the end of the process.

To give a concrete example, consider the table setting task from earlier, however, now we assume there is a napkin that must be placed under the fork. Given the level of noise seen in 5.6, once a reasonable template has been found with the appropriate spatial reference frames, which user's demonstration order should we use during execution? What if that user's demonstration does not perfectly match the resulting template? What if there is no particular order that must be followed (e.g., fork or spoon first).

Similar to previous work in programming by demonstration, one could manually annotate a template or demonstration with constraints such as a reference frame or ordering of placements [2, 15]. This, however, both complicates the demonstration process and removes our ability to provide solutions to WAPP problems in an unsupervised manner. Therefore, in this chapter, we contribute a method for weighting models from earlier with a coarse temporal weight. Such a weight is derived entirely from the unlabeled data collected from users. We then show both the cases in which this method can properly define temporal and spatial constraints as well as its known failure cases.

## 6.1 Background

### 6.1.1 Classical Planning

In many senses the problem of ordering constraints can be translated into a classical planning problem. One of the most well known planning styles is STRIPS (Stanford Research Institute Problem Solver) [27, 71, 60]. STRIPS was the foundation of many later works, including PDDL (Planning Domain Definition Language) [34].

In the original STRIPS literature, the term *world model* is introduced. The world model is a representation of the world based on a set of facts, or truths. This world model is considered to be the *initial state*. In contrast to this, STRIPS also specifies a *goal state*. These are the states of the world that we wish to achieve. For example, if we want to have the boxes $B$ and $C$ moved from room $y$ to room $x$, we could use the following world model goal state:

$$\mathrm{At}(B, x) \wedge \mathrm{At}(C, x)$$

In addition to world models, STRIPS instances also consist of a set of *operators*. Operators consist of three sets. The first is a set of pre-conditions (in the form of propositional variables). These conditions must be true in order for the operator to be executed. The second and third sets are sets of *effects*, one for which values become true (or added to the world model) by the execution of the operator, and the second being which values become false (or removed from the world model).

Note that operators are analogous to *actions*. The goal of STRIPS, therefore, is to find a plan, or path, from the starting state to the goal state by following the necessary pre and post conditions through the execution of a sequence of actions.

Many variants of programming languages have been defined around STRIPS to allow for powerful and efficient planning. In 1998, the Planning Domain Definition Language (PDDL) was introduced as a standard for the Artificial Intelligence Planning Systems 1998 (AIPS-98) competition [34]. While not the first formal language definition for STRIPS-style problems, it has become one of the most prominent languages for planning and has been revised into several different versions: PDDL1.2 [34], PDDL2.1 [30], PDDL2.2 [23], PDDL3.0 [32], and PDDL3.1 [52]. Each variant of PDDL introduced extensions such as allowing for non-binary, numeric values to represent a distance in a constraint.

We note that while a direct analogy can be made between WAPP goal state templates and classical planning, the lack of known pre-conditions makes the translation inadequate. Furthermore, STRIPS makes use of the closed world assumption (CWA) which tells us that if something is not stated in the world model, then it is assumed to be false. During WAPP execution, this assumption cannot hold. In fact, CWA, in its entirety, does not hold in the real world and is the most fundamental limitation of STRIPS in terms of converting WAPP goal state execution into a planning environment. As shown later in Chapter 7, we knowingly violate this assumption and allow users to modify the world state during task execution.

### 6.1.2 Robotic Pick-and-Place Ordering

Ordering constraints have also been explored explicitly within the domain of robotic pick-and-place. Early work in programming by demonstration [2, 15] allowed users to define tasks as a series of actions. These actions, in the form of keyframe based locations or low-level joint trajectory playback, could then be arranged in sequences manually according to the user's preference.

Work done by Mohseni-Kabir et. al [57, 58] introduced an interactive based approach. In this work, the user could guide the robot through a task (in their work, a tire rotation domain) using a series of available action primitives. This one-shot learning approach framed the problem as a hierarchical task network (HTN) in which partial ordering constraints were inferred. The agent would then make suggestions to the user asking if they wanted to group common actions (e.g., unscrewing a series of nuts) as a high-level sub-task (e.g., unscrewing the entire hub).

Finally, as a canonical example within pick-and-place, Ekvall and Kragic [25, 24] infer ordering constraints for tasks such as table setting from multiple human demonstrations. Unlike our approach, task demonstrations were done in a continuous manner and segmented based on distance thresholds as task items moved throughout the demonstration. As tasks were executed, ordering constraints were inferred by initially assuming the task must be executed in the exact order of the first demonstration. As further demonstrations were added, any constraint that was violated were immediately removed from the set of possible constraints. Work done by Niekum et. al. [62, 1] furthered this approach by utilizing Bayesian Nonparametric Time Series Analysis and motion primitives to significantly improve the segmentation of tasks into actions. It is noted however that a certain level of expertise is needed by the human teacher in order to reach reasonable results.

## 6.2 Methodolgy

In this section, we define and present a method for weighting plausible template models based on the coarsely defined temporal placement information available from the high-level goal-state demonstration database. We say that the temporal, or in this case, ordering information is coarsely defined since we only know for a particular user if the demonstration was placed either *before* or *after* a given instance of a reference. This concept can be thought of as a binary informational component and is more formally defined throughout this section.

### 6.2.1 Temporal and Spatial Weighting

Remember that a spatial template is defined as a set of models $\lambda \in \Lambda$ for a given task. Each model has an item $\lambda^i$ for an item $i \in I$ and a reference frame $\lambda^{\theta_o}$ for an object $o \in O$ within the world. Furthermore, $\lambda$ also contains an associated Gaussian distribution $\mathcal{N}(\mu_k, \Sigma_k)$ obtained via EM clustering where the mean represents the spatial location of placing $i$ with respect to $o$. The model $\lambda$ also has a weight defined by a ranking function $\pi$ denoting the strength of the associated cluster and thus the model (again, where smaller values represent a better fitting model).

Prior to this chapter, $\pi$ (defined in Equation 5.2) is expressed solely in terms of *spatial* data. With the introduction of ordering constraints, we want to define a new function, $\Phi$, which combines both the spatial component in $\pi$ and a temporal component $\psi$. These two components are then weighted via some weighting term $\alpha$ in the range [0, 1]. Intuitively, the weighting parameter $\alpha$ decides how heavily to weight temporal data $\psi$ over spatial data $\pi$ and can be tuned based on domain knowledge prior to execution. A general form of this method given in Equation 6.1.

$$\Phi = \alpha\pi + (1 - \alpha)\psi \tag{6.1}$$

In order to provide a reasonable bounds and value for $\Phi$, we normalize each component in the function to be in the range [0, 1] (and thus $\Phi$ is also in the range [0, 1]). For $\pi$, remembering from earlier the threshold value $\delta$ used to determine which models and templates were acceptable, we know an upper bound of the values. Therefore, we can normalize $\pi$ by this terms giving us $\frac{\pi}{\delta}$.

### 6.2.2 The Temporal Ratio Term

In this subsection we define the temporal term $\psi$. Given the data collection method described in Chapter 5, we are able to determine a coarse binary temporal value in the form of $before$ or $after$; that is, was the item in the demonstration placed before or after the current reference frame.

More formally, consider the data set $\hat{D}_i^o$ being generated in Algorithm 2 (Section 5.3.2) which represents all aggregated placement demonstrations of item $i$ with respect to object $o$. For a particular demonstration instance $d$ from a user, during this generation step we know both when the user placed $i$ and when (if ever, i.e., surfaces are always present) the user placed $o$. We denote this as a binary term $\hat{d}_\psi \in \{0, 1\}$ where 0 indicates the item was placed after the current reference frame, and 1 indicates that it was placed prior to the reference frame.

Given this information, we can then compute a ratio indicating the number of placement points inside of a given cluster that were placed after the associated reference frame. Equation 6.2 gives us this ratio in terms of each data point $\hat{d}$ in a given cluster $c$ inside of a model $\lambda$.

$$\psi(\text{cluster } c \ \in \lambda) = \frac{\sum_{i=1}^{|c|} \hat{d}_{i_\psi}}{|c|} \tag{6.2}$$

We note that the above equation may be susceptible to noise in the demonstration. To address this, we weight the binary term $\hat{d}_\psi$ by the probability that that demonstration comes from the given cluster. That is, if a demonstration point is far away from the mean of the given cluster, we assume that the demonstration point is less reliable. Therefore, each point is weighted by the probability density function associated with the Gaussian distribution for that cluster (Equation 6.3 where $n$ is the dimensionality of the distribution or 3 in our case).

$$\psi(\text{cluster } c \ \in \lambda) = \frac{\sum_{i=1}^{|c|} \hat{d}_{i_\psi} \frac{1}{\sqrt{(2\pi)^n |\Sigma_c|}}^{-\frac{1}{2}(\hat{d}_i - \mu_c)^\intercal \Sigma_c^{-1} (\hat{d}_i - \mu_c)}}{|c|} \tag{6.3}$$

### 6.2.3 Graph Reduction for Execution Order

With the above definitions, we now assume our new ranking function $\Phi$. Note that this term is a real-valued number in the range $[0,\ 1]$ and contains a metric on how well a placing item $i$ fits against reference frame $o$ both spatially and temporally. However, unlike earlier when we only used spatial criteria $\pi$, temporal data informs the model of a sense of execution-level ordering that must take place. In Chapter 5, when executing the template in a real-world domain, we used a best-first greedy approach (as discussed in Section 5.5.1 with Algorithm 3). While appropriate when dealing with unconstrained spatial data, this approach cannot be maintained with temporal data.

As an example, let us consider a simple two-item domain in which item $A$ should be placed prior to $B$. Let us also define some additional fixed reference frame in the world $F$. Assume that the spatial metric for $i_B$ with respect to $o_F$ is extremely well fitting (i.e., $\pi$ is small). Furthermore, assume that $A$ is always placed before $B$. Depending on $\alpha$, our greedy approach from earlier would likely choose this small $\pi$ value for $i_B$ with respect to $o_F$ first since it seems like a great starting point. However, now that the $B$ has been placed, the value of $\psi$ for $i_B$ with respect to $i_A$ is 1 thus introducing a notion of "error" into the execution (since $A$ should be placed before $B$). This means that execution cannot simply be defined in a greedy approach.

Therefore, given our new ranking metric $\Phi$, we compose our template execution strategy as a minimization problem. That is, we want to find the sequence of placements across all possible models $\Lambda$, such that the sum of $\Phi$ across all chosen placements is smallest.

In doing so, we translate the problem into a graph problem. Formally, we define each node in the graph to represent the current set of remaining items that must be placed according to the task description. Each edge in the graph therefore represents placing an available item from the target node according to some reference frame. The new node connected to this edge thus represents the set of available items from the target node minus the placed item. The weight of the edge is defined as $\Phi$. A form algorithm for this problem reduction is given in Algorithm 4.

We start by constructing the vertex set which represents all possible item set combinations (i.e., the power-set of $I$) in lines 4-6. This is analogous to creating a vertex for any possible state of items left to place throughout task execution. Next, for each of these states, we consider all possible edges. Line 8 looks at each item within each vertex and looks at creating a edge from that vertex $v$ to a destination vertex $u$ associated with placing item $i$ (line 10). Note that $i$ can be placed according to many different reference frames. Therefore, we search through the set of models $\Lambda$ in line 11 in which $i$ is the item being placed. Of these resulting models, we obtain the coordinate frame for which $i$ is placed in reference to (line 12). Lines 13-17 then check if this reference frame is both an item *and* if the reference frame is an item from the source node $v$. This an important check because if $o \in \hat{i}_v$,

**Algorithm 4** Reducing $\Lambda$ into a Weighted Graph

```
Input: Set of plausible placement models $\Lambda$ for the current task
       Set of task items $I$
Output: A weight graph $G = (V, E)$, a set of vertices and edges
1: $V \leftarrow \emptyset$
2: $E \leftarrow \emptyset$
3: $\hat{I} \leftarrow \mathcal{P}(I)$ (i.e., the power-set of $I$)
4: for each item set $\hat{i}$ in $\hat{I}$ (i.e., $\forall\ \hat{i} \in \hat{I}$) do
5:     $V \leftarrow \{V \cup \texttt{Vertex}(\hat{i})\}$
6: end for
7: for each vertex $v$ in $V$ (i.e., $\forall\ v \in V$) do
8:     for each item $i$ in $v$ do
9:         $\hat{i}_v \leftarrow$ item set associated with $v$
10:        $u \leftarrow$ vertex in $V$ with associated item set $\hat{i}_v - i$
11:        for each model $\lambda$ in $\Lambda$ such that $\lambda^i = i$ do
12:            $o \leftarrow \lambda^{\Theta_o}$
13:            if $o \in I \land o \in \hat{i}_v$ then
14:                $w \leftarrow \infty$
15:            else
16:                $w \leftarrow \Phi(\lambda)$
17:            end if
18:            $e \leftarrow \texttt{Edge}(v, u, w)$
19:            $E \leftarrow \{E \cup e\}$
20:        end for
21:    end for
22: end for
23: return $G = (V, E)$
```

then the reference item has yet to be placed. Therefore, it is impossible to use this transition and we thus set its weight to $\infty$. Finally, the resulting weighted edge is added to the edge list on line 19 before returning the completed graph on line 23. An example illustration of a graph is shown in Figure 6.1 with a simplified domain of $I = \{fork, plate$ and $S = \{table\}$ where "w.r.t." denotes "with respect to."

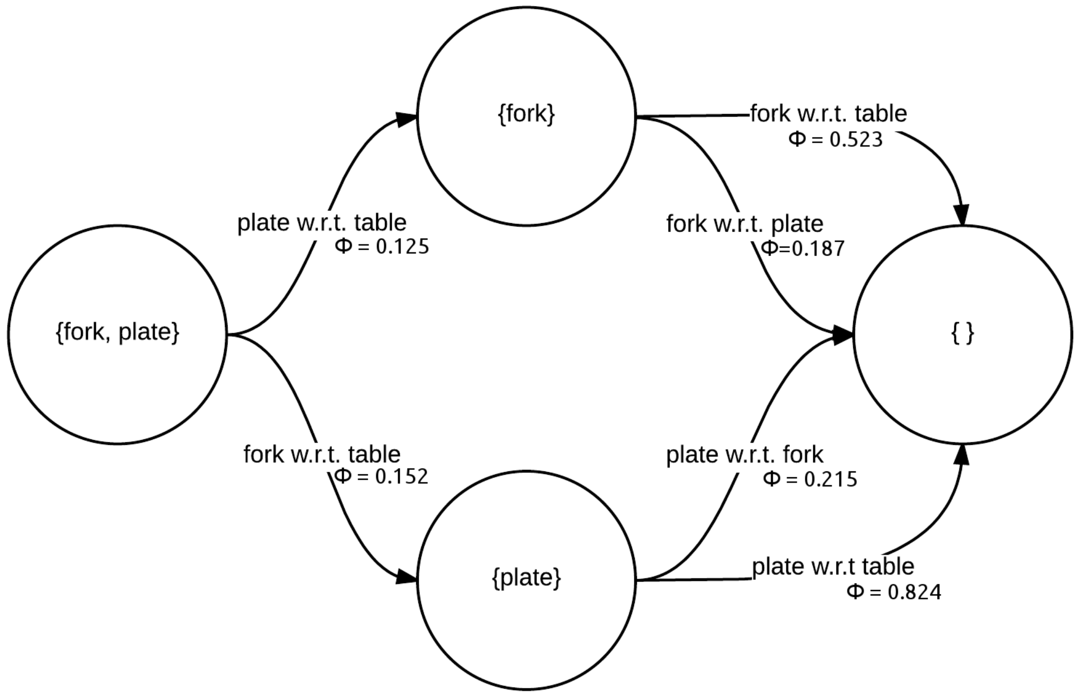


Figure 6.1: An example of a placement graph.

Given the resulting graph, an ordering of $\lambda$ placement models can then be made via a lowest-cost path search from the start state of $I$ to the goal state of $\emptyset$. That is, from the full set of items that need to be placed until the goal state of no-more-items-to-place. Such a path can therefore be computed via Dijkstra's algorithm[71]. This path therefore minimizes the cost of placing all items with respect to both spatial and temporal strength.

## 6.3 Results

This section describes two forms of evaluation for the above methods. In the first section, we deal with a constrained block domain which allows us to formalize different conditions for templated ordering constraints. In the second section, we look at real-world results from the same household domain presented in Chapter 5.

### 6.3.1 Variable Size Block Domain

To begin, we consider a simplistic case of ordering constraints. Let us consider a small block-world domain consisting of a set of 5 different sized blocks. The goal is to demonstrate to the system how to properly stack the blocks in order starting with the largest block ($\texttt{block}_1$) on the bottom and ending with the smallest block ($\texttt{block}_5$) on the top (Figure 6.2). We also assume a single "surface" frame in the world which we call the world-frame ($S_{\texttt{world}}$).

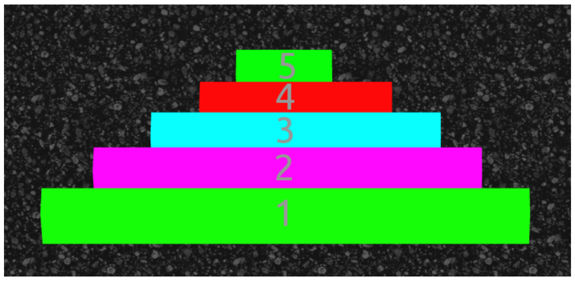


Figure 6.2: The simple tower stacking domain.

For the purposes of exploring the strengths and weaknesses of the developed methods, this section utilizes non-crowdsourced demonstrations. That is, we manually provide the system with 25 temporally noise-free demonstrations in order to see if an reasonable ordering emerges. We consider demonstrations to be temporally noise-free if each demonstration is placed in the correct order but allow for perturbations in the $x, y, \theta$ values.

As a baseline, we process the results using the existing methods developed in Chapter 5. The resulting models are presented in Table 6.1 and the resulting tem-

Table 6.1: $\pi$-Based Models Obtained from the Simple Tower Task

| $\lambda^i$ | $\lambda^{\Theta_o}$ | $\pi(\lambda)$ | $\lambda^i$ | $\lambda^{\Theta_o}$ | $\pi(\lambda)$ | $\lambda^i$ | $\lambda^{\Theta_o}$ | $\pi(\lambda)$ |
|---|---|---|---|---|---|---|---|---|
| $\texttt{block}_1$ | $\texttt{block}_2$ | **0.0273** | $\texttt{block}_2$ | $\texttt{block}_1$ | **0.0273** | $\texttt{block}_3$ | $\texttt{block}_1$ | **0.0502** |
| $\texttt{block}_1$ | $\texttt{block}_3$ | **0.0502** | $\texttt{block}_2$ | $\texttt{block}_3$ | **0.0230** | $\texttt{block}_3$ | $\texttt{block}_2$ | **0.0230** |
| $\texttt{block}_1$ | $\texttt{block}_4$ | **0.0749** | $\texttt{block}_2$ | $\texttt{block}_4$ | **0.0479** | $\texttt{block}_3$ | $\texttt{block}_4$ | **0.0258** |
| $\texttt{block}_1$ | $\texttt{block}_5$ | **0.0995** | $\texttt{block}_2$ | $\texttt{block}_5$ | **0.0725** | $\texttt{block}_3$ | $\texttt{block}_5$ | **0.0501** |
| $\texttt{block}_1$ | $S_{\texttt{world}}$ | **0.0478** | $\texttt{block}_2$ | $S_{\texttt{world}}$ | **0.0444** | $\texttt{block}_3$ | $S_{\texttt{world}}$ | **0.0492** |
| $\texttt{block}_4$ | $\texttt{block}_1$ | **0.0749** | $\texttt{block}_5$ | $\texttt{block}_1$ | **0.0995** | $\texttt{block}_4$ | $\texttt{block}_2$ | **0.0479** |
| $\texttt{block}_5$ | $\texttt{block}_2$ | **0.0725** | $\texttt{block}_4$ | $\texttt{block}_3$ | **0.0258** | $\texttt{block}_5$ | $\texttt{block}_3$ | **0.0501** |
| $\texttt{block}_4$ | $\texttt{block}_5$ | **0.0262** | $\texttt{block}_5$ | $\texttt{block}_4$ | **0.0262** | $\texttt{block}_4$ | $S_{\texttt{world}}$ | **0.0607** |
| | | | $\texttt{block}_5$ | $S_{\texttt{world}}$ | **0.0860** | | | |

plate is depicted in Figure 6.2. As can be seen, while the blocks are indeed arranged in a correct spatial arrangement, utilizing the greedy execution strategy in Algorithm 3 results in the following ordering of the placements:

$$\texttt{block}_2\text{-w.r.t.-}S_{\texttt{world}} \rightarrow$$
$$\texttt{block}_3\text{-w.r.t.-}\texttt{block}_2 \rightarrow$$
$$\texttt{block}_4\text{-w.r.t.-}\texttt{block}_3 \rightarrow$$
$$\texttt{block}_5\text{-w.r.t.-}\texttt{block}_4 \rightarrow$$
$$\texttt{block}_1\text{-w.r.t.-}\texttt{block}_2$$

Not only is this not the desired ordering, but given physical constraints in the real world, it would be impossible to place a block in mid-air without having the supporting block underneath it.

Next, we run the same data through Equation 6.3 to compute our ranking metric $\Phi$ for each model using an value of $\alpha = 0.5$. The results of these values are presented in Table 6.2. Utilizing the graph reduction technique outlined in Algorithm

4 provides the following execution solution with negligible affects on the spatial layout:

$$\texttt{block}_1\text{-w.r.t.-}S_{\texttt{world}} \rightarrow$$
$$\texttt{block}_2\text{-w.r.t.-}\texttt{block}_1 \rightarrow$$
$$\texttt{block}_3\text{-w.r.t.-}\texttt{block}_2 \rightarrow$$
$$\texttt{block}_4\text{-w.r.t.-}\texttt{block}_3 \rightarrow$$
$$\texttt{block}_5\text{-w.r.t.-}\texttt{block}_4$$

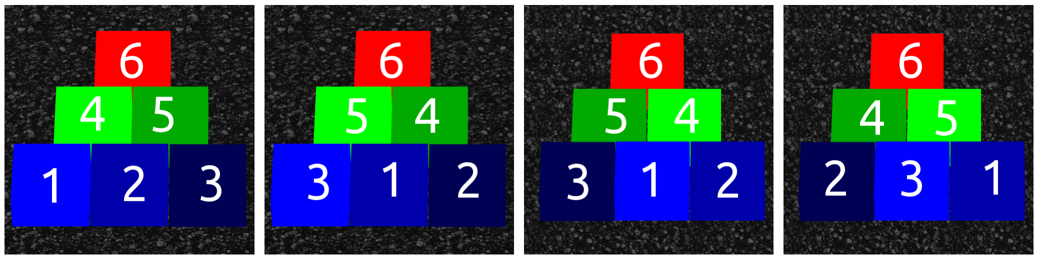
Table 6.2: Φ-Based Models Obtained from the Simple Tower Task

| $\lambda^i$ | $\lambda^{\Theta_o}$ | $\Phi(\lambda)$ | $\lambda^i$ | $\lambda^{\Theta_o}$ | $\Phi(\lambda)$ | $\lambda^i$ | $\lambda^{\Theta_o}$ | $\Phi(\lambda)$ |
|---|---|---|---|---|---|---|---|---|
| $\texttt{block}_1$ | $\texttt{block}_2$ | **0.5124** | $\texttt{block}_2$ | $\texttt{block}_1$ | **0.0124** | $\texttt{block}_3$ | $\texttt{block}_1$ | **0.0228** |
| $\texttt{block}_1$ | $\texttt{block}_3$ | **0.5228** | $\texttt{block}_2$ | $\texttt{block}_3$ | **0.5104** | $\texttt{block}_3$ | $\texttt{block}_2$ | **0.0104** |
| $\texttt{block}_1$ | $\texttt{block}_4$ | **0.5340** | $\texttt{block}_2$ | $\texttt{block}_4$ | **0.5217** | $\texttt{block}_3$ | $\texttt{block}_4$ | **0.5117** |
| $\texttt{block}_1$ | $\texttt{block}_5$ | **0.5452** | $\texttt{block}_2$ | $\texttt{block}_5$ | **0.5329** | $\texttt{block}_3$ | $\texttt{block}_5$ | **0.5228** |
| $\texttt{block}_1$ | $S_{\texttt{world}}$ | **0.0217** | $\texttt{block}_2$ | $S_{\texttt{world}}$ | **0.0202** | $\texttt{block}_3$ | $S_{\texttt{world}}$ | **0.0223** |
| $\texttt{block}_4$ | $\texttt{block}_1$ | **0.0340** | $\texttt{block}_5$ | $\texttt{block}_1$ | **0.0452** | $\texttt{block}_4$ | $\texttt{block}_2$ | **0.0217** |
| $\texttt{block}_5$ | $\texttt{block}_2$ | **0.0329** | $\texttt{block}_4$ | $\texttt{block}_3$ | **0.0117** | $\texttt{block}_5$ | $\texttt{block}_3$ | **0.0228** |
| $\texttt{block}_4$ | $\texttt{block}_5$ | **0.5119** | $\texttt{block}_5$ | $\texttt{block}_4$ | **0.0228** | $\texttt{block}_4$ | $S_{\texttt{world}}$ | **0.0276** |
| | | | $\texttt{block}_5$ | $S_{\texttt{world}}$ | **0.0391** | | | |

As we can see from these results, adding the temporal component $\psi$ into the weighting metric significantly improves our ability to represent a template in both a spatial and temporal manner. Block placements which violate our ordering constraint have noticeably higher Φ values than those which adhere to the desired ordering constraint. Furthermore, an important result of this case is that by adding these new values into our models, we still are able to maintain the spatial structure of the template itself. This result is further strengthened in the remainder of this chapter.

### 6.3.2 Uniform Size Colored Block Domain

The above results represent the effectiveness of combining both spatial and temporal information within the ranking heuristic as well as execution via the graph-reduction technique. While the presented case is the most simplistic of cases, we now present 4 classes of problems where we run our algorithm. For these 4 classes of problems, we also change our domain from a 5-block different size domain into a 6-block domain depicted in Figure 6.3. The 4 classes are as follows:



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

Figure 6.3: Example placement demonstrations for the 4 block-world conditions.

1. **Consistent Ordering, Consistent Placement** – This case is analogous to the variable size block domain. In this case we place the blocks in the same order starting from the bottom left and build the bottom row, then the middle row, then the top row. Blocks were placed with starting with the lightest shade of each color per row from left to right. The exact ordering and placement locations are shown in Figure 6.3a.

2. **Random Ordering, Consistent Placement** In this case we randomly place the blocks in a different order one row at a time. (e.g., if the first row has blocks 1, 2, 3, we would do demonstrations in the order 1, 2, 3, – 1, 3, 2 – etc...). In this case the ordering is randomized, but the locations are still fixed

(i.e., lightest to darkest). An example ordering with the exact locations of the placements is shown in Figure 6.3b.

3. **Consistent Ordering, Random Placement** This case is the opposite of the *Random Ordering, Consistent Placement* case. Here we randomize the order on each row but keep the ordering temporally consistent. The ordering of each block with an example placement location is shown in Figure 6.3c.

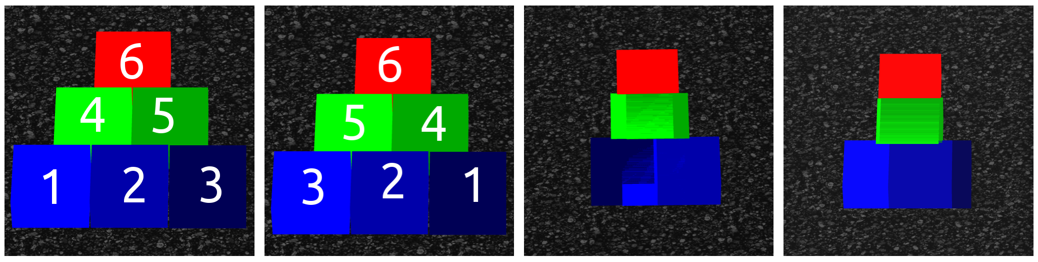
4. **Random Ordering, Random Placement** Here, we randomly place blocks in any order at any location within the row. An example is shown in Figure 6.3d.

Once again, we provide 25 demonstrations for each case. Instead of listing and enumerating results for all possible reference frame combinations in this domain (64 possible values for each condition), we present the ordering and $\Phi$ values returned by the graph search for each case. For a listing of these values, see the Appendix at the end of this thesis. For each case, we refer to each color block as the following:

- $\texttt{block}_{lb}$ - The light-blue block (number 1 in Figure 6.3a).
- $\texttt{block}_{mb}$ - The medium-blue block (number 2 in Figure 6.3a).
- $\texttt{block}_{db}$ - The dark-blue block (number 3 in Figure 6.3a).
- $\texttt{block}_{lg}$ - The light-green block (number 4 in Figure 6.3a).
- $\texttt{block}_{dg}$ - The dark-green block (number 5 in Figure 6.3a).
- $\texttt{block}_{r}$ - The red block (number 6 in Figure 6.3a).

### Consistent Ordering, Consistent Placement

Since this condition is analogous to the 5-block domain presented in Section 6.3.1, it is unsurprising to once again find a reasonable placement and ordering of the blocks.



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

Figure 6.4: Resulting templates for the 4 block-world conditions.

After running the training data through the process, the algorithm returns to us the following (shown visually in Figure 6.4a).

$$\begin{aligned}
\texttt{block}_{lb}\text{-w.r.t.-}S_{\texttt{world}}\ (\Phi = 0.3481)\ &\rightarrow \\
\texttt{block}_{mb}\text{-w.r.t.-}\texttt{block}_{lb}\ (\Phi = 0.0063)\ &\rightarrow \\
\texttt{block}_{db}\text{-w.r.t.-}\texttt{block}_{mb}\ (\Phi = 0.0099)\ &\rightarrow \\
\texttt{block}_{lg}\text{-w.r.t.-}\texttt{block}_{db}\ (\Phi = 0.0125)\ &\rightarrow \\
\texttt{block}_{dg}\text{-w.r.t.-}\texttt{block}_{lb}\ (\Phi = 0.0172)\ &\rightarrow \\
\texttt{block}_{r}\text{-w.r.t.-}\texttt{block}_{lg}\ (\Phi = 0.0226)&
\end{aligned}$$

Unsurprisingly, we do obtain a spatially and temporally consistent template as we found earlier.

### Random Ordering, Consistent Placement

This case is an important case in verifying that the temporal component $\psi$ does not overpower the spatial component $\pi$. Furthermore, we also hope to see that strong spatial relationships do not overpower the temporal relationships. By keeping the placement locations consistent, we hope to still achieve the same template spatially as shown in the previous case. Here, for each demonstration we randomly place the blocks within each row. After running the training data through the process, the algorithm returns to us the following (shown visually in Figure 6.4b).

$$\texttt{block}_{db}\text{-w.r.t.-}S_{\texttt{world}}\ (\Phi = 0.3330)\ \rightarrow$$
$$\texttt{block}_{mb}\text{-w.r.t.-}\texttt{block}_{db}\ (\Phi = 0.2519)\ \rightarrow$$
$$\texttt{block}_{lb}\text{-w.r.t.-}\texttt{block}_{mb}\ (\Phi = 0.2553)\ \rightarrow$$
$$\texttt{block}_{dg}\text{-w.r.t.-}\texttt{block}_{lb}\ (\Phi = 0.0060)\ \rightarrow$$
$$\texttt{block}_{lg}\text{-w.r.t.-}\texttt{block}_{lb}\ (\Phi = 0.0072)\ \rightarrow$$
$$\texttt{block}_{r}\text{-w.r.t.-}\texttt{block}_{lg}\ (\Phi = 0.0069)$$

This case exemplifies the effectiveness of our methodology. Within each row, ordering does not matter. In fact, in the Appendix we can see similar values across the the various components within each row. However, we do not overpower the spatial relationship and still achieve the desired template. Furthermore, while the ordering was random within each row, since the ordering of each row was held consistent, we can see that we do not place blocks from any of the rows above until each row is complete.

### Consistent Ordering, Random Placement

This case is similar to the dishes example in Section 5.4.2 (Multi-Item Uncoupled Placements). Since the block placement locations are randomized in each row, we do not expect our core algorithm to produce a template in the form of a tower. This is not unreasonable since we are never truly providing a unique spatial template of the unique blocks. After running the training data through the process, the algorithm returns to us the following (shown visually in Figure 6.4c).

$$\texttt{block}_{lb}\text{-w.r.t.-}S_{\texttt{world}}\ (\Phi = 0.4998)\ \rightarrow$$

$$\texttt{block}_{mb}\text{-w.r.t.-}\texttt{block}_{lb}\ (\Phi = 0.0892)\ \rightarrow$$

$$\texttt{block}_{db}\text{-w.r.t.-}\texttt{block}_{mb}\ (\Phi = 0.0294)\ \rightarrow$$

$$\texttt{block}_{dg}\text{-w.r.t.-}\texttt{block}_{db}\ (\Phi = 0.0495)\ \rightarrow$$

$$\texttt{block}_{r}\text{-w.r.t.-}\texttt{block}_{dg}\ (\Phi = 0.0285)\ \rightarrow$$

$$\texttt{block}_{lg}\text{-w.r.t.-}\texttt{block}_{db}\ (\Phi = 0.0470)$$

As with the dishes tasks, we wind up with overlapping elements in the result which again can be overcome at execution time by utilizing the resulting Gaussian distributions within each model. We note that given the randomness within the spatial template, even the ordering of placements is lost after the first row. While it is possible to tune $\alpha$ to more heavily weigh $\psi$ over $\pi$, we present this result as is in order to showcase both strengths and weaknesses of the approach. We note as well that we still obtain the correct ordering and placement of each row from top to bottom. This is due to the fact that each row is still placed under the conditions of the *Consistent Ordering, Consistent Placement* condition. That is, each row can be though of as a sub-template of the problem.

### Random Ordering, Random Placement

For the sake of completeness, we present results from a completely random set of demonstrations both spatially and temporally. As with the previous conditions, we keep randomness to within each respective row. While it would be possible to run experiments where ordering and location were randomized across all 6 blocks, we note that such an example is in no form a template – that is, little information would be obtained from the data set as there is no ground truth to be learned. After running the training data through the process, the algorithm returns to us the following (shown visually in Figure 6.4c).

$$\texttt{block}_{mb}\text{-w.r.t.-}S_{\texttt{world}}\ (\Phi = 0.4137)\ \rightarrow$$
$$\texttt{block}_{lb}\text{-w.r.t.-}\texttt{block}_{mb}\ (\Phi = 0.3205)\ \rightarrow$$
$$\texttt{block}_{dg}\text{-w.r.t.-}\texttt{block}_{lb}\ (\Phi = 0.0516)\ \rightarrow$$
$$\texttt{block}_{r}\text{-w.r.t.-}\texttt{block}_{dg}\ (\Phi = 0.0321)\ \rightarrow$$
$$\texttt{block}_{lg}\text{-w.r.t.-}\texttt{block}_{mb}\ (\Phi = 0.0383)\ \rightarrow$$
$$\texttt{block}_{db}\text{-w.r.t.-}\texttt{block}_{lb}\ (\Phi = 0.2404)$$

As with the previous condition, we still maintain our "sub-templates" on a per-row basis, but all temporal and spatial information is lost within each row.

### 6.3.3 Household Domain

Up until now, the presented results show both the effectiveness and limitations of our methodology across different types of temporally and spatially constructed problems. In this section we report results from a real-world dataset from the crowdsourced household domain. In particular, we re-frame our table setting task to incorporate the placement of a napkin within the template. The goal would be a template in which the napkin was placed under the fork and spoon.

In order to not bias users to placing items in any particular ordering of the items, we randomize the order in which items are presented to users. As before, we present the full list of items in the task description, and users are then allowed to scroll through the list when placing the items.

Perhaps the most important result from this experiment is that, as with the experiments from Chapter 5, the developed methodology is able to handle an abundance of noise. Figure 6.5 once again depicts this noise by showing the entire placement set within an instance of the world.

For this task, remote users were presented with the following task description: *Your job is to set the table with the given objects.* The objects provided were as

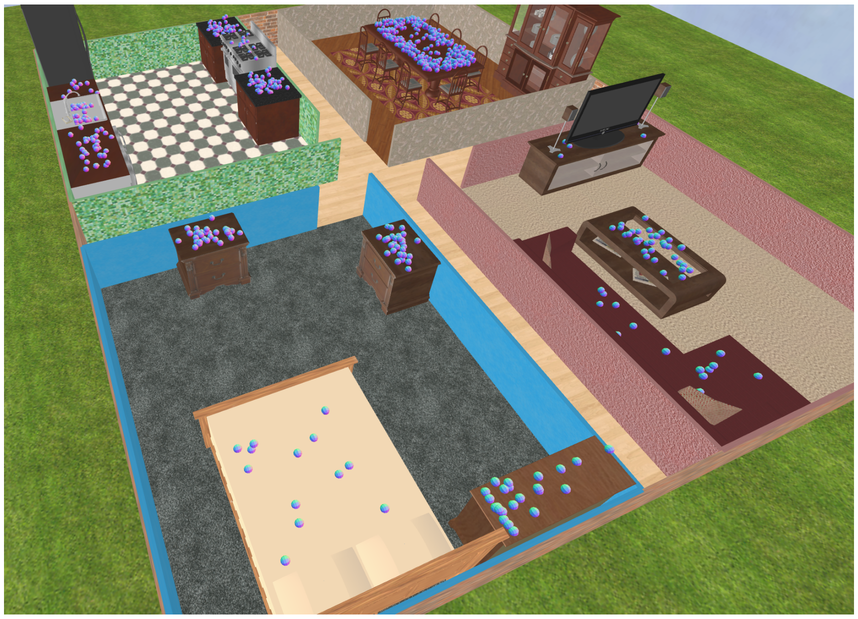


Figure 6.5: The noisy demonstration set of all placements for the napkin task.

follows: $I = \{I_{\texttt{Cup}}, I_{\texttt{Plate}}, I_{\texttt{Spoon}}, I_{\texttt{Fork}}, I_{\texttt{Napkin}}\}$. We recorded 900 placements, which resulted in 37 valid models (i.e., where $\pi(\lambda) \leq \delta$) after transformation and training. We refer readers to the Appendix for a full table of values. Here we present the final template ordering returned from the graph reduction step:

$$\begin{aligned}
I_{\texttt{Plate}}\text{-w.r.t.-}P_{\texttt{chair}}\ (\Phi = 0.2533)\ &\rightarrow \\
I_{\texttt{Napkin}}\text{-w.r.t.-}I_{\texttt{Plate}}\ (\Phi = 0.3001)\ &\rightarrow \\
I_{\texttt{Fork}}\text{-w.r.t.-}I_{\texttt{Napkin}}\ (\Phi = 0.0942)\ &\rightarrow \\
I_{\texttt{Spoon}}\text{-w.r.t.-}I_{\texttt{Napkin}}\ (\Phi = 0.1223)\ &\rightarrow \\
I_{\texttt{Cup}}\text{-w.r.t.-}I_{\texttt{Plate}}\ (\Phi = 0.2852)&
\end{aligned}$$

Shown visually in Figure 6.6, these results verify that even in a noisy domain from untrained users, we are able to provide a valid final grounding for a real-world task template with ordering constraints.

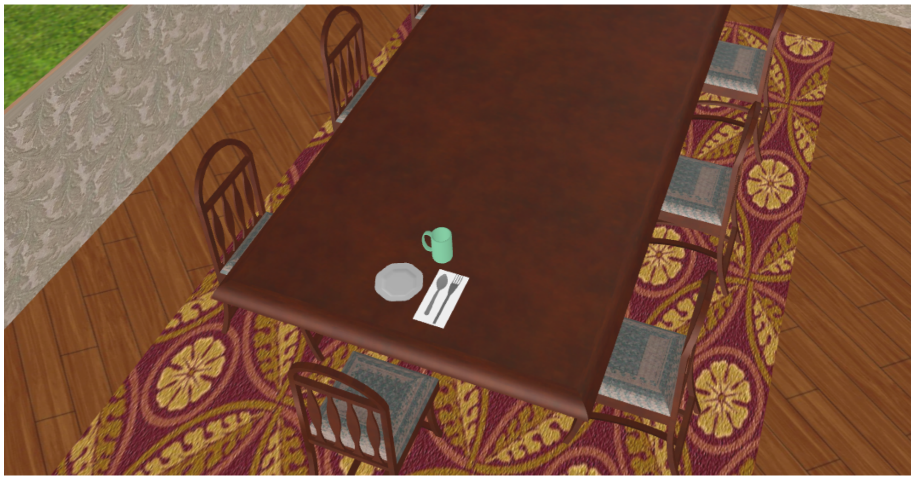


Figure 6.6: The resulting napkin template.

## 6.4 Conclusion

In this chapter, we expanded our general templating algorithm from Chapter 5 to adhere to potential unknown ordering constraints. In particular, we developed a method which kept to our goal of remaining an unsupervised approach and required no additional input methods or data from the human user. By looking at the coarse ordering information from each item-object pairing in the data set, we were able to compute and utilize a ratio weighted by the likelihood of each data point. This temporal component was then combined with our spatial weighting function from Chapter 5 to form our new weighting function $\Phi$. Finally, by using our reduction algorithm, we were able to reduce the problem of searching for an execution order by framing the problem as a lowest-cost path graph problem.

We presented results in two forms. In the first form, we looked at different types of constrained problem types within a simplified block-world domain. In particular, we showed how we were able to reconstruct a tower adhering to both spatial and temporal constraints in both the *Consistent Ordering, Consistent Placement* and *Random Ordering, Consistent Placement* conditions. We also showed in the *Consistent Ordering, Random Placement* how a rough ordering could be maintained even with randomness from the spatial placements. More interesting though from both the *Consistent Ordering, Random Placement* and *Random Ordering, Random Placement* conditions is the persistence of the "sub-templates" (i.e., each row as a component). In the second form, we were able to return to the household crowd-sourced domain and show the ability to overcome noisy environments from untrained users in order to set a table in the appropriate order.

By combining both spatial and temporal information from the users, we consider this to be the first complete form of SPAT learning for our framework. Since we are concerned with full task completion from start-to-end, we show how deriving a second form of SPAT learning from observations can provide a solution to actively searching for items at the start of task execution.

# CHAPTER 7
# TEMPORAL PERSISTENCE MODELING FROM SPARSE OBSERVATIONS

At this point in this thesis work, we have enabled a mobile manipulator with the ability utilize crowd-sourced spatial goal templates and reference frames to complete tasks and allowed for refinement of these templates using coarse temporal ordering information. These steps, however, are only half of the problem in WAPP tasks. In order for robotic agents to complete a wide range of tasks in mobile environments, they must, at some level, be able to search effectively for objects. That is, the robotic agent must be able to know where to search for and retrieve each required object for the task in order to place it according to the goal template.

## 7.1 Introduction

The ability to effectively locate objects is critical for robotic tasks across a wide range of assistive, household and service applications. Commonplace tasks, such as pick and place or manipulation, require the robot to navigate to different locations to retrieve objects, often given only uncertain information about the object's current location. Within this context, the problem of active visual search (AVS) is that of effectively predicting object locations in a large-scale environment using primarily visual sensing [7, 8].

Recent work in semantic mapping has explored a wide range of AVS solutions, considering search for both known [56, 54] and previously unknown [8, 72] objects. Many of the presented techniques leverage semantic information, such as object co-occurrence probabilities [49] or typical object locations [7]. One limitation of existing methods is that they typically operate under the assumption of object permanence; in other words, once found, objects are assumed to remain in their relative position. That is, existing methodologies lack a robust representation of *temporal memory.*

In this chapter, we address the problem of AVS within a domain where other agents (e.g., people) may move objects around without the robot's knowledge. For example, in a household setting, a person getting a snack is likely to carry a plate from the kitchen to the coffee table, and then back half an hour later. Similarly, in an office setting, shared objects such as staplers or computer accessories are often moved between work areas. We are interested in the problem of effectively predicting the permanence of an object at a given location in order to enable improved visual search.

We present a novel solution to the object search problem based on *temporal persistence modeling*, a probabilistic prediction of the time that an object is expected to remain at a given location given sparse prior observations of the object locations in the environment. Leveraging concepts from reliability theory [70], we show that probabilistic exponential distributions augmented with a Gaussian component can accurately represent probable object locations and search suggestions based entirely on sparsely made visual observations. We evaluate our work in two domains, a large scale GPS location data set for person tracking, and multi-object tracking on a mobile robot operating in a small-scale household environment over a 2-week period. Our results show that the performance of our algorithm exceeds four baseline methods across all study conditions.

## 7.2 Background

At its core, semantic mapping deals with mapping human spatial concepts to objects within the world [66]. Many approaches have been developed for tackling this broad research problem within the robotics and perception communities [55, 56, 66, 67, 89]. Seminal work by Pronobis breaks down objects to be what he refers to as "spatial concepts" [66], which can be observational properties (e.g., the object is green) or relationship properties (e.g., the object is in the kitchen). Thus, at its core, the problem of semantic mapping is that of assigning some type of semantic (e.g., human-readable and understandable) label to a spacial entity within the world.

Semantic map information can also be used to determine object placement. For example, Mason and Marthi [56] build a semantic world model that enables the detection and tracking of objects over time, resulting in the ability to model where objects are in the world (e.g., where did you see object $x$), or how objects move around (e.g., where do instances of $x$ go throughout the day). The major limitation of the proposed system is that it requires extensive time for post-processing.

In some senses, this problem is related to that of simultaneous localization and mapping (SLAM). In particular, fastSLAM [59] presents an approach of mapping and localizing within that environment dealing entirely with features instead of complete sensor data (e.g., occupancy grids from a laser scan). In our environment, the items themselves can be though of as features. Therefore, given information about how long they persist on a surface throughout the day, the item features can provide additional information on the likelihood or weighting a feature should take when helping to localize the robot.

In an alternate approach to object search, Kollar and Roy [49] predict the location of previously unseen objects given the probability of their co-occurrence with other known objects in the environment. This approach assumes a map and the locations of known objects in the environment are given to the robot. Then, given a novel object that has not been previously observed in the environment, the system leverages an online object co-occurrence database derived from Flickr to model the co-occurrence probabilities used to determine the likely new object location based on what it already knows about the environment. The work presented in this chapter is complimentary to Kollar and Roy's approach in that it seeks to generate and maintain the model of *known* object locations, which may then be used to predict the location of novel objects through their co-occurrence method.

In [72, 50], the authors present an approach that enables the robot to search the web in order to identify an object's most likely location. The approach, called *ObjectEval*, searches the Web to infer the probability that an object, such as "coffee", can be found in a location, such as a "kitchen". The algorithm is able to dynamically instantiate a utility function for different locations using this probability, with the benefit that the robot is able to effectively search for arbitrary objects is may never have previously encountered. Again, we view this work as complimentary to the contributions of this thesis; the technique presented in our work utilizes memory and online learning to track the locations of known objects, as opposed to focusing on the ability to find novel objects. Once objects are known, recognized and seen multiple times, it is more effective for a robot to track their likely location in a particular environment than to rely on a general notion of likely locations mined from the web, since the web-based suggestions perform well only in domains that fit common stereotypes.

Another popular approach to solving the object search problem involves augmenting the environment with additional sensors. Most popular is the use of radio-frequency identification (RFID) tags attached to key objects and reference points. For example, in [21], Deyle et al. enable a PR2 robot to locate objects in a highly-cluttered household environment by placing ultra-high frequency radio-frequency identification (UHF RFID) tags on objects. In their work, the problem of inferring object poses and the variability in signal strength is addressed in order to robustly search for a wide site of objects. However, even with the use of RFID tags, an object search model is still required to enable the robot to determine which area to scan for objects.

Lorbach et al. [54] present an approach in which the robot navigates around the environment and constructs a scene graph from its observations both at a semantic and spatial level. The robot then uses this information to develop a series of knowledge bases: the *Scene Structure* which contains the observed scene graph, the *Domain Knowledge* containing co-occurrence information (similar to the work in [49]), *Physical Constraints* which encode information such as large objects not being inside of smaller objects, *Logical Consistency* which ensures an object can only be in one place at a time, and *Search History* which serves as short term memory for spatial relationships. While this approach begins to explore the idea of history in guiding its search, we argue that simply remembering the past observations ignores the inherent temporal information about the object (i.e., how long it remains there). This idea is explored explicitly in our analysis. Furthermore, the need for hand-coded domain knowledge (which the authors say is to remove noise) makes such an approach less general.

Of course, the problem of persistence estimation extends beyond the field of robotics. One such example which remains relevant is the work of Brandt et. al. [13]. The referenced work presented as a method for event count predictions in the research field of Political Science. Researchers use statistical methods to attempt to predict events relating to US Supreme Court cases and international relations. In [13], a Kalman filter is used to estimate a moving average of a Poisson distribution. Unlike the work presented in this thesis, work in these fields typically deal with event counts, that is, directly with the number of events which occurred over a certain time (typically in the past). In our work, we incrementally build up a model from scratch using a coarse set of observations of the world state.

## 7.3 Methodology

In this chapter, we contribute *temporal persistence modeling (TPM)*, an algorithm for probabilistically modeling object locations based on sparse temporal observations. We define *temporal persistence* as the time an object is expected to be in a particular location before moving to another location (i.e., "disappearing"). More formally, let us consider the case of searching for an item $i \in I$. Given the current time $t_c$, and the last time item $i$ was seen at each surface $s \in S$, denoted $t_{i,s}$, our goal is to calculate the probability that $i$ is currently located on $s$.

To address this problem, we leverage concepts from the fields of *reliability theory* [70] and *failure analysis* [44]. Widely applied in biomedical studies, industrial life testing, psychology, economics and engineering disciplines, reliability theory is a branch of statistics used for modeling the expected duration of time until a particular event occurs, such as death in biological organisms and failure in mechanical systems.

Within the context of our work, the event time distribution we are interested in modeling is the time at which an object is removed from one location and placed at another. We formalize the object search problem using reliability theory as a maximization of the probabilistic temporal persistence ($P_{\mathtt{tp}}$) given in Equation 7.1.

$$\underset{s \in S}{\operatorname{argmax}} \, P_{\mathtt{tp}}(i \ \mathtt{located\text{-}on} \ s | t_c, t_{i,s}) \tag{7.1}$$

We now present how $P_{\mathtt{tp}}$ is derived. The first step in defining our temporal persistence model utilizes the exponential distribution [19]. In general, exponential distributions model the time between events and contain a single rate parameter, $\lambda$.[1] In order to accurately define $\lambda$, we utilize the fact that the mean $\mu$, or expected value of the distribution, is defined as $\mu = \frac{1}{\lambda}$. In TPM, $\mu_{i,s}$ represents the average or expected time that elapses between the last known observation of item $i$ on $s$,

[1]We note the use of $\lambda$ here should not be confused with the WAPP template models $\lambda$ from previous sections. We choose to reuse $\lambda$ in this chapter as to match the majority of work utilizing exponential distributions.

and the time at which $i$ is removed from $s$. We note that a problem which must be addressed is that observations at the actual time of removal are unlikely. That is, as the robot goes around the environment making observations, it most likely does not observe the actual time an object moves locations (unless, of course, the robot moves the object itself). Such an assumption would require a global visual sensing system, a case which we assume is not available. We later address how to probabilistically sample and calculate $\mu_{i,s}$, but for now we assume the value is known.

Given our exponential model for a given object-location pairing $\texttt{exp}(\lambda)_{i,s}$ with associated expected value $\mu_{i,s}$, we can calculate the probability that $i$ has been removed from $s$ on or before the current time $t_c$ from the cumulative distribution function (CDF) given in Equation 7.2.

$$\texttt{cdf}_{\texttt{exp}}(t, \lambda) = \begin{cases} 1 - e^{-\lambda t} & t \geq 0 \\ 0 & t < 0 \end{cases} \tag{7.2}$$

Therefore, to answer our initial question, the probability that $i$ is still located on $s$ at time $t_c$ is analogous to 1 minus the probability $i$ has be removed from $s$ at or before $t_c$. Our equation for this persistence model is given in Equation 7.3; a visualization is also provided in Figure 7.1.

$$\begin{aligned} P_{\texttt{tp}}(i \texttt{ located-on } s | t_c, t_{i,s}) &= 1 - \texttt{cdf}_{\texttt{exp}}((t_c - t_{i,s}), \frac{1}{\mu_{i,s}}) \\ &= e^{-\frac{1}{\mu_{i,s}}(t_c - t_{i,s})} \end{aligned} \tag{7.3}$$

The final missing component of the equation is determining $\mu_{i,s}$ for each model. As stated earlier, if we could assume a large-scale continuous perception system in the environment, we could assume we know exactly when $i$ was removed from $s$; however, if this assumption were true, our system would have perfect information about item locations, eliminating the need for this entire approach. Since such a system is unlikely to be widely available, is currently infeasible, and is costly to deploy, we assume that no such data is available.

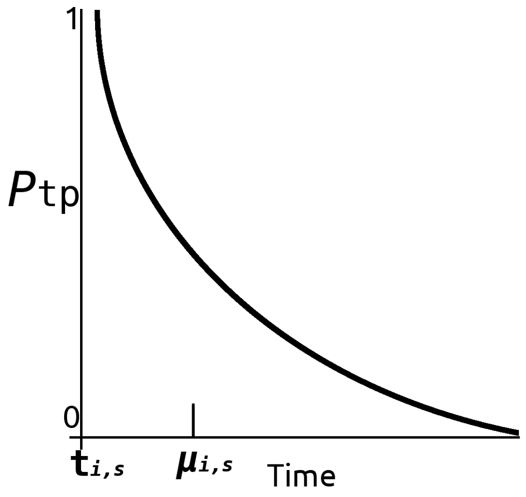


Figure 7.1: An example persistence model for item $i$ on surface $s$. Probabilities represent the probability $i$ is still located on $s$ at the given time.

Instead, we assume that the robot periodically records observations at random time steps throughout the day as it goes about its daily tasks and duties. It is also possible to program the robot to perform regularly scheduled sweeps of the environment to observe item locations. However, such behavior could be distracting and intrusive to other occupants of the space, and so we do not rely on such functionality in our system. Instead, we perform TPM updates each time items are recognized by the robot in its field of view during the course of normal operation. In this scenario, one of three possibilities can occur:

1. Item $i$ was observed on surface $s$ and it was not there before.

2. Item $i$ was previously observed on surface $s$ and it is still there.

3. Item $i$ was previously observed on surface $s$ and it is no longer there.

In order to create samples for each $\mu_{i,s}$, consider the example presented in Figure 7.2 which represents observations of $i$ on $s$. Here, we assume the item was first observed at time $t_1$, re-observed each time at times $t_2$, $t_3$, and $t_4$, and then observed missing at time $t_5$. For this work, we assume, with high probability, that the item was present in between times $t_1$ and $t_4$.

The question then becomes, when did the item actually disappear? Theoretically, the actual time could be anywhere in the range $[t_4, t_5]$; however, we assume that it is more likely that the disappearance happened somewhere in between the two and not at the extremities. Therefore, we fit a Gaussian distribution over $\Delta_t$ and randomly choose a value from inside that distribution as a sample for updating $\mu_{i,s}$.

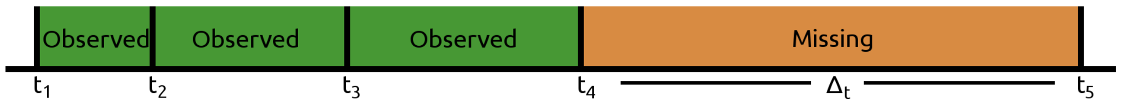


Figure 7.2: An example observation timeline of an item on a given surface.

In order to generate our estimated time of removal, we fit a Gaussian distribution over the time frame of $\Delta_t$ such that there is a 99.7% chance that a value chosen at random is between $t_4$ and $t_5$. We therefore want $t_4$ and $t_5$ to each be $3\sigma$ away from the mean, which gives our Gaussian distribution the value of $\mathcal{N}(\frac{\Delta_t}{2}, \frac{\Delta_t}{3})$. As illustrated in Figure 7.3, we then add the random value from the Gaussian to the difference between $t_4$ and $t_5$. This value is then sampled for updating $\mu_{i,s}$. Furthermore, as shown in the upcoming algorithm, current observations are more heavily weighted over past observations allowing for the models to change overtime as location preferences change.

The complete algorithm for updating $\mu_{i,s}$ in TPM is shown in Algorithm 5. The algorithm is called each time a robot makes a new observation of a surface $s$ which is denoted as $\Omega$. This set $\Omega$ contains a list of all item from $I$ which are observed on surface $s$. At the end of the algorithm, the new value of $\mu_{i,s}$ for each item-surface pairing allows us to create our temporal persistence models and therefore we now have enough information to maximize Equation 7.1 during item search.

**Algorithm 5** Temporal Persistence Modeling $\mu$ Update

```
$s \leftarrow$ surface being observed
$t_c \leftarrow$ current time
$\Omega \leftarrow$ makeObservation($s$)
for each item $i$ observed in $\Omega$ (i.e., $\forall\ i \in \Omega$) do
    if $i$ was not previously observed on $s$ then
        mark $i$ as observed on $s$ at time $t_c$
    else
        mark $i$ as still observed on $s$ at time $t_c$
    end if
end for
for each item $i$ not observed in $\Omega$ (i.e., $\forall\ i \in \{O \setminus \Omega\}$) do
    $t_s \leftarrow$ first observation of $i$ on $s$
    $t_l \leftarrow$ latest observation of $i$ on $s$
    $t_f \leftarrow t_l - t_s +$ rand($\mathcal{N}(\frac{t_c - t_l}{2}, \frac{t_c - t_l}{3})$)
    $\bar{\mu} \leftarrow \{\mu_1, \ldots, \mu_n\}$ previous estimates of $\mu_{i,s}$
    $\mu_{i,s} \leftarrow \frac{(\sum_{j=1}^{n}(j/n)\bar{\mu_j}) + t_f}{(\sum_{j=1}^{n}(j/n)) + 1}$
    mark $i$ as not observed on $s$
end for
```

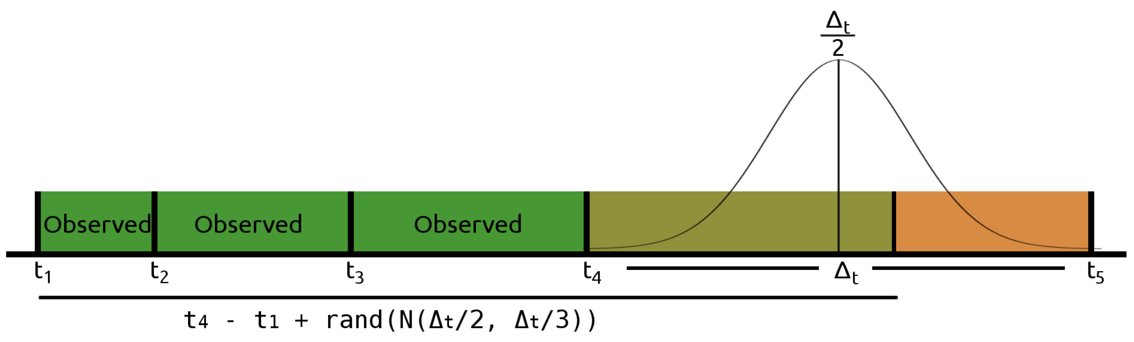


Figure 7.3: The method for probabilistically determining the sample removal time of an object on a location given its observation timeline.

## 7.4 Analysis

In this section, we present analysis of our temporal persistence modeling approach in two applications, 1) large-scale tracking of a single target based on real-world GPS cell phone data, and 2) multi-object tracking using a mobile robot over a 2-week period in a small-scale household setting. In both applications we compare the performance of TPM in predicting the location for a given item $i$ against four baseline methods:

1. **TP Random:** We select a random value for $\mu_{i,s}$ and then apply the remainder of the TPM algorithm as described above. The $\mu_{i,s}$ value is chosen in the range from the minimum to the maximum time difference between two observations in the training set. This experimental condition aims to show that the $\mu$ value within TPM is intelligently chosen, as well as the impact of a poor value.

2. **Last Seen:** We predict the surface for $i$ based on the last surface where $i$ was observed.

3. **Most Frequent:** We predict the surface for $i$ based on that object's most frequently observed surface.

4. **Random:** We randomly select a surface for $i$ from among the known locations.

### 7.4.1 Single Item Tracking from GPS Data

In our first experiment, we utilize GPS tracking data from the Microsoft Research GeoLife [90] dataset[2] to verify the ability of our temporal persistence models for learning probable object locations based on sparse temporal observational data. A GPS trajectory of this dataset is represented by a sequence of time-stamped points, each of which contains the latitude, longitude and altitude of a single cell phone user. The recordings span the full range of activities of everyday life, including regular work commutes, as well as entertainment and sports activities, such as shopping, sightseeing, dining, hiking, and cycling, making it an compelling dataset for temporal tracking.

For the purposes of our work, we are interested in modeling the temporal persistence of each individual cell phone user. Specifically, given a sparse set of observations of prior user locations, we seek to predict the user's current location. It is important to note that, when dealing with people, context aware methods are likely more suitable for location tracking than temporal persistence modeling. Information such as a person's place of work, members of their social circle, favorite places to eat, or even the current direction of motion are helpful in aiding in the analysis and prediction of a person's activities. In our work, however, we are primarily interested in tracking inanimate objects using *sparely observed spatial and temporal observations*, and thus we utilize the location information in this dataset as a useful benchmark due to its size and diversity of users. As we show in the results section, temporal persistence modeling performs extremely well on this dataset, even without the use of context information.

#### Experimental Setup

For our analysis we discretize the user trajectory data by superimposing a uniform $10 \times 10$ grid over the range of latitude and longitude values for a given user. There-

[2] http://research.microsoft.com/en-us/downloads/b16d359d-d164-469e-9fd4-daa38f2b2e13/

fore, each cell in the grid becomes a "location" and the user becomes the "object" for which we want to predict the location. Trajectories within the GeoLife dataset are logged in a dense representation, every 1-5 seconds. We subsample this data in order to create a relatively sparse training and test set that more closely resemble the frequency of object location observations a robot is likely to obtain during typical operation. The test set is generated by sampling 10% of a given user's data uniformly and without replacement. We then sort the resulting data points according to their timestamps and train either TPM or one of the baseline methods (in the case of TPM, we iteratively train the model using Algorithm 5). We generate the test set by sampling 10% of the remaining data from that user, again uniformly and without replacement. Each data point in the test set is used to query the trained model (in the case of TPM, by applying Equation 7.1). We then compare the returned value to the ground-truth of the actual discretized location to obtain an accuracy measure for the approach.

Note that throughout the verification process, we update the last seen time for the temporal persistence models $t_{i,s}$ and last seen location for the last-seen method as verification observations are made. Further note that this is *not* the same as updating the TPM parameter $\mu_{i,s}$.

### Results

We run the described analysis on 10 randomly selected users' datasets. Note that throughout the verification process, we update the last seen time for the TP models $t_{i,s}$ and last seen surface for the last-seen method as verification observations are made. Further note that this is *not* the same as updating the TP model parameter $\mu_{i,s}$.

Each user is run 50 times. The average success rate for each method across the 50 trials ($\mu$), standard deviation across the trials ($\sigma$), and number of samples chosen to create the models ($N$) for each user are presented in Table 7.1. Note again that $N$ is analogous to 10% of the entire dataset size per user.

Table 7.1: Results of searching for the user in the discretized grid.

| | $\mu$ | $\sigma$ | $p$ | | $\mu$ | $\sigma$ | $p$ |
|---|---|---|---|---|---|---|---|
| | **Experiment 1 ($N$=17,387)** | | | | **Experiment 2 ($N$=8,748)** | | |
| TP Modeling | **99.358%** | 0.048% | – | | **89.192%** | 0.357% | – |
| Random TP | 29.811% | 2.040% | $\ll 0.001$ | | 5.698% | 2.618% | $\ll 0.001$ |
| Last Seen | 96.819% | 1.140% | $\ll 0.001$ | | 64.600% | 6.217% | $\ll 0.001$ |
| Most Frequent | 88.903% | 1.140% | $\ll 0.001$ | | 39.465% | 0.494% | $\ll 0.001$ |
| Random | 32.348% | 2.906% | $\ll 0.001$ | | 3.784% | 0.130% | $\ll 0.001$ |
| | **Experiment 3 ($N$=29,118)** | | | | **Experiment 4 ($N$=2,477)** | | |
| TP Modeling | **98.660%** | 0.058% | – | | **82.944%** | 4.107% | – |
| Random TP | 42.1178% | 7.250% | $\ll 0.001$ | | 46.235% | 11.014% | $\ll 0.001$ |
| Last Seen | 97.051% | 0.997% | $\ll 0.001$ | | 78.922% | 4.754% | $\ll 0.001$ |
| Most Frequent | 97.051% | 0.250% | $\ll 0.001$ | | 50.477% | 0.935% | $\ll 0.001$ |
| Random | 10.838% | 0.529% | $\ll 0.001$ | | 2.940% | 0.265% | $\ll 0.001$ |
| | **Experiment 5 ($N$=883)** | | | | **Experiment 6 ($N$=4,556)** | | |
| TP Modeling | **99.404%** | 1.906% | – | | **93.210%** | 0.864% | – |
| Random TP | 55.962% | 17.434% | $\ll 0.001$ | | 11.841% | 3.993% | $\ll 0.001$ |
| Last Seen | 99.395% | 2.195% | $\ll 0.001$ | | 89.050% | 2.753% | $\ll 0.001$ |
| Most Frequent | 98.779% | 8.093% | $\ll 0.001$ | | 70.208% | 2.753% | $\ll 0.001$ |
| Random | 40.896% | 10.206% | $\ll 0.001$ | | 5.263% | 0.290% | $\ll 0.001$ |
| | **Experiment 7 ($N$=17,570)** | | | | **Experiment 8 ($N$=1,551)** | | |
| TP Modeling | **96.200%** | 0.440% | – | | **80.955%** | 2.576% | – |
| Random TP | 43.516% | 2.058% | $\ll 0.001$ | | 58.763% | 8.490% | $\ll 0.001$ |
| Last Seen | 86.399% | 5.417% | $\ll 0.001$ | | 70.243% | 6.022% | $\ll 0.001$ |
| Most Frequent | 52.478% | 0.333% | $\ll 0.001$ | | 42.457% | 1.149% | $\ll 0.001$ |
| Random | 2.857% | 0.075% | $\ll 0.001$ | | 8.558% | 0.455% | $\ll 0.001$ |
| | **Experiment 9 ($N$=79)** | | | | **Experiment 10 ($N$=958)** | | |
| TP Modeling | **95.088%** | 0.411% | – | | **97.124%** | 0.387% | – |

*Continued on next page*

Table 7.1 – *Continued from previous page*

| | $\mu$ | $\sigma$ | $p$ | | $\mu$ | $\sigma$ | $p$ |
|---|---|---|---|---|---|---|---|
| Random TP | 40.481% | 17.005% | $\ll 0.001$ | | 62.014% | 4.249% | $\ll 0.001$ |
| Last Seen | 82.835% | 7.629% | $\ll 0.001$ | | 88.574% | 6.217% | $\ll 0.001$ |
| Most Frequent | 16.785% | 4.629% | $\ll 0.001$ | | 59.367% | 1.831% | $\ll 0.001$ |
| Random | 4.964% | 2.238% | $\ll 0.001$ | | 40.133% | 5.498% | $\ll 0.001$ |

We report evaluation results for 10 users randomly selected from the GeoLife dataset. Figure 7.4 presents the average prediction success rate for all 10 users across all five algorithms. Due to the nature of the GeoLife dataset, the number of datapoints available for each user varies greatly; the $N$ value next to each user label in the figure indicates the number of training/testing samples available per user (i.e., 10% of the entire dataset for that user). All performance values are averaged over 50 trials per user, with training and testing samples randomly drawn for each trial. Points along the outside of the graph indicate performance close to 100%, whereas points near the center of the graph indicate predictive performance close to 0%.

As can be seen in the graph, the performance of all baseline methods varies greatly across users, while the performance of TPM (dashed blue line) consistently outperforms all other methods for all 10 users. Based on the data findings, we can observe certain patterns among the user data. Users 1, 3 and 5 appear to be sedentary, remaining in the same place for long periods of time. This can be observed from the fact that both the Last Seen and Most Frequent baselines perform very well for these users. The remaining users change location to varying degrees, with User 9 exhibiting the least repetitive behavior[3]. The Last Seen metric performs

[3]Given that this user has significantly less data than all other users, it is likely that we do not get the chance to observe the person's behavior long enough to see repeated patterns. One of the strengths of TPM is that its model can be constructed from relatively little data, which is discussed further during the second experiment.

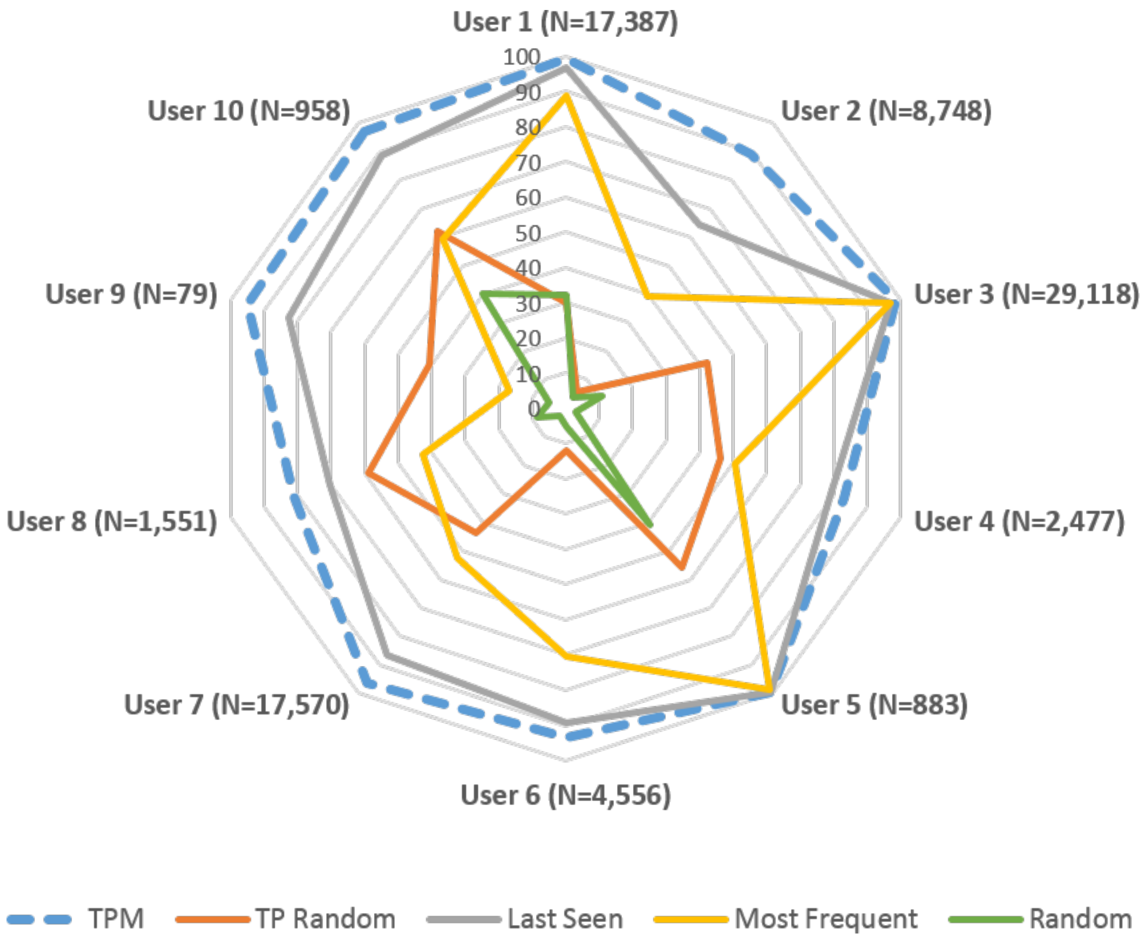


Figure 7.4: Performance of TPM and four baseline methods in predicting the location of 10 independent users in the GeoLife datase.

well across the full range of users. This is to be expected since multiple consecutive readings typically fall within the same discrete grid cell of our discretized state space. However, TPM dominates the performance of Last Seen (and all other baselines) with statistical significance of $p \leq 0.001$ as calculated by a $t$-test performed against each method independently. Additionally, we note that TP Random performs significantly more poorly than the complete TPM algorithm, indicating that customizing $\mu_{i,s}$ with the respect to an individual user's behavior significantly improves the performance of the algorithm.

From these experiments we have shown that using Algorithm 5 to generate

TP models based on sparse temporal observations provides a reasonable prediction method for object tracking. Results from the $t$-tests show that our method out-performs other reasonable guessing strategies used in previous literature (mainly last-seen and most-frequent). Furthermore, we also out-perform a random $\mu_{i,s}$ showing that our method for determining such a value is reasonable.

### 7.4.2 Multi-Item Tracking in a Robotic Apartment Environment

The GPS data evaluation above showed that TPM can effectively be used to track a single item (in that case, a person) over time. In this evaluation we evaluate our approach in tracking multiple physical objects in the context of a real-time, real-world robotics domain. We utilize a mobile manipulation robotic platform operating in a small-scale household environment consisting of a kitchen area, dining area and a seating area. Figure 7.5 presents an abstract map of the physical space, and Figure 7.6 shows the robot located in the kitchen area. Within the context of this experiment the robot does not manipulate any objects, but it is able to navigate the space and perform object recognition [48] to identify the objects of interest. The robot is provided with a map of the space that contains semantic labels associated with different locations in that space (e.g., *kitchen table, coffee table, couch).* Beyond that, we assume the robot has no additional information about any prior object locations when it begins.

Now that we have shown the effectiveness of utilizing sparse temporal based observations for tracking items in a discrete search space, we now show its effectiveness inside a real-time, real-world robotic domain. Here, we utilize a mobile manipulator which is able to do autonomous navigation in a apartment scenario as well as 3D based tabletop object recognition via the methods described by Kent et. al. [48]. For this experiment, we assume a robot has been placed into a new environment and has made no initial observations.

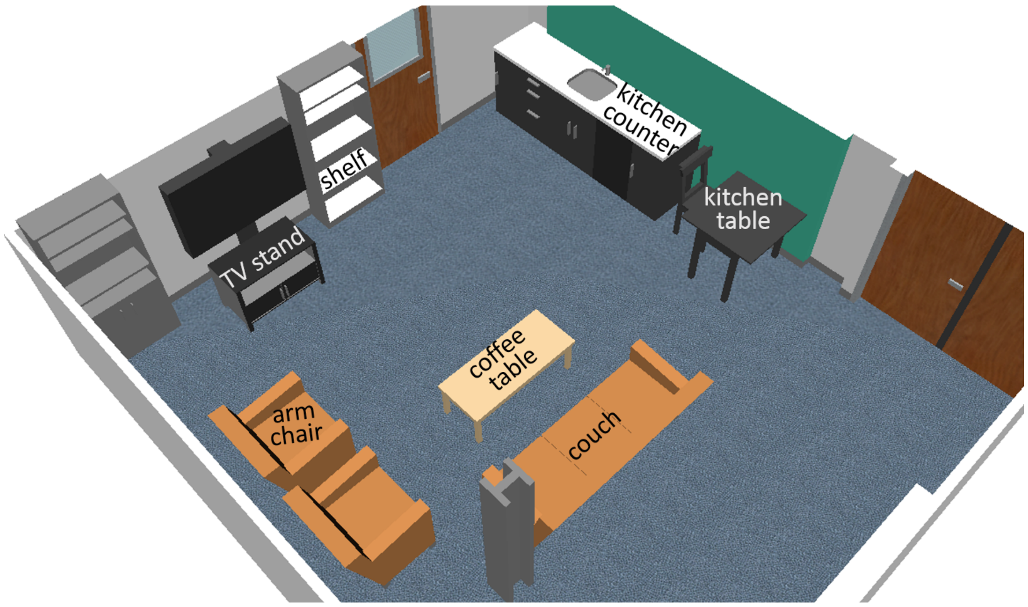


Figure 7.5: Map of the search space.

**Experimental Setup**

The dataset for this experiment was collected over a 2-week period, during which the robot was operated 6 days per week, 5 hours a day. The time restriction is due in part to the robot's battery limitations, but it also represents a reasonable estimate of the time a mobile robot may actively operate inside a typical household setting. During its operation time, the robot adhered to the following procedure:

1. Select a target surface $s \in S$ uniformly at random.
2. Navigate to the selected surface and make an observation of all items located on that surface.
3. Update TPM as defined in Algorithm 5.
4. Select a random wait time from a uniform distribution in the range of $[1, 20]$ minutes; return to Step 1.

This behavior pattern was selected because it mimics an observation pattern the robot might obtain while going about daily tasks in a household environment. For

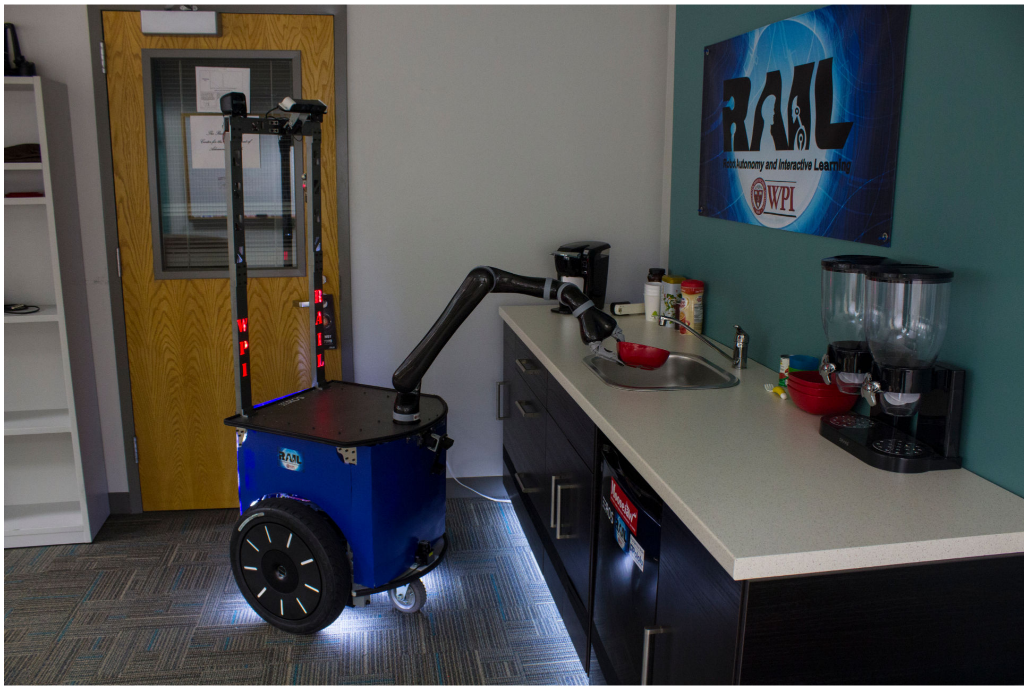


Figure 7.6: The robot, located near the kitchen counter.

example, the robot may go to tidy up the coffee table and in the process observe the objects that are currently placed there. Then it may go to the kitchen to clean up the dishes and in the process observe the objects placed there. Thus our work specifically seeks to target the scenario in which the robot makes incidental observations over the course of its normal operation, as opposed to a targeted sweeping search that records the location of every object in the environment. In this study the robot does not move any objects between locations itself, modeling object movement in that scenario would require a straightforward update to a world model or semantic map. However, TPM does support this scenario; if the robot were to move an object, the sample value to add to $\mu_{i,s}$ would be explicitly known (since the robot knows the exact time it moved the item), and therefore it would not need to be sampled from the Gaussian distribution.

In this chapter, we are instead interested in the more challenging case in which objects are moved by other agents (in our case, people) in the environment. We used the following set of surfaces in the experiment $S$={*TV stand, arm chair, shelf, couch, coffee table, kitchen counter, kitchen table*}. The items tracked in the study included $I$={*book, cup, plate, fork, candle, potted plant*}. Each of the objects was used to evaluate a different temporal persistence behavior:

1. **Regular Alternating 1 (RA1, plate)** The location of the plate alternated between the kitchen table and counter. The plate was on the kitchen counter for the first 3 hours of the day, then one hour at the kitchen table before being returned to the counter for the remainder of the day.

2. **Regular Alternating 2 (RA2, fork)** The location of the fork alternated between two hours on the kitchen counter and 30 minutes on the coffee table.

3. **Regular Alternating 3 (RA3, candle)** The location of the candle alternated between one hour on the coffee table and one hour on the kitchen table.

4. **Pattern Change (PC, book)** During the first week, the book was located on the kitchen table for the first hour of the day and then moved to the bookshelf for the remainder of the day. During the second week, the book remained on the TV unit stand for the first 3 hours of the day before moving to the bookshelf for the remainder of the day.

5. **Stationary (S, plant)** The plant remained on the bookshelf throughout the two-week study.

6. **Random (R, cup)** The location of the cup was changed at random over the course of the study by selecting $s \in S$ uniformly at random, and selecting a time period uniformly from the range $[1, 20]$ minutes, then repeatedly moving the cup to the selected surface after the designated period of time elapsed.

Each of the above objects can be viewed as a different study condition, ranging from random movement to stationary objects. RA 1-3 represent the most interesting temporal persistence variant for our target use case, one in which objects are moved based on some implicit customs or patterns that are unknown to the robot but that it is able to indirectly observe. The PC condition was selected to evaluate the scenario in which the use pattern of an object changed significantly during the period of observation.

In total, our 72 hours of robot operation included 412 observation snapshots taken across the seven surfaces in the domain, resulting in 506 object observations. Below, we use the resulting dataset of timestamped observations to evaluate the robot's ability to effectively predict the location of an item.

### Results

In order to evaluate the performance of TPM, we train a temporal permanence model for each object based on the observations obtained during the two-week data collection period. We then generate a test set by sampling an equal number of data points over a future two week period, following the same object placement schedule as previously described. The ground truth of each data point in the test set is determined based on the originally defined schedule.

Table 7.2 presents the evaluation results for each of the study conditions; $N$, the number of observations of each object, is also provided. As can be seen in the table, the performance of TMP dominates or matches the performance of the baseline methods for all six study conditions. RA1 and RA2 behave similarly in that both have a single location where the object of interest remains for the majority of the time; thus, both Last Seen and Most Frequent baselines perform well in these conditions. What enables TMP to improve over Last Seen and Most Frequent is its ability to predict that the object has most likely been moved from its last seen location after some period of time elapses.

RA3 helps to demonstrate the benefit a temporal model has over the heuristic baselines. In this study condition, the object alternates between two positions, spending equal time at both. As a result, both Last Seen and Most Frequent are able to guess the correct location of the object with only 50% accuracy, while TPM

Table 7.2: Results from the real-world robotic experiment.

| | **RA1** ($N = 89$) | **RA2** ($N = 87$) | **RA3** ($N = 79$) |
|---|---|---|---|
| TPM | **96.360%** | **83.021%** | **57.362%** |
| TP Random | 93.187%* | 75.932%* | 52.861%* |
| Last Seen | 96.044% | 80.114% | 50.460% |
| Most Frequent | 96.097% | 80.166% | 49.539% |
| Random | 18.4599% | 19.678% | 18.762% |
| | **PC** ($N = 86$) | **S** ($N = 78$) | **R** ($N = 87$) |
| TPM | **92.743%** | **100.000%** | **24.071%** |
| TP Random | 28.021%* | 17.970%* | 20.433%* |
| Last Seen | 91.250% | **100.000%** | 21.652% |
| Most Frequent | 91.250% | **100.000%** | 17.804% |
| Random | 20.895% | 21.014% | 19.445% |

*Indicates an average of 10 runs each with a different random value for $\mu_{i,s}$

is again able to predict the object's movement to a greater degree. By adding a temporal decay to the prediction strategy, the robot is able to gain an improvement in its success rate. However, this condition exemplifies a difficult case even for TPM. Consider the following:

Note that each TPM is independent of the others. Assume that both the locations $s_1$ and $s_2$ in the alternating case example have perfect $\mu_{i,s}$ values each of 1 hour. Now assume we have observed $i$ on $s_1$ 2 hours ago and $i$ on $s_2$ 1 hour and 59 minutes ago. Without any higher-level reasoning or knowledge of the other locations, the robot picks $s_2$ as the location regardless of the fact that it has most likely switched. This is a direct result of $\mu_{i,s}$ being the same for each location. Note that, however, by increasing the observation frequency at each location we could improve the success rate of such a case.

The PC condition is similar to RA1 in that the object alternates between two locations, spending more time at one than the other. However, the difference is that one of the locations changes part way through the study. As a result, the predictive performance of all methods in this condition is slightly lower than in RA1. The Stationary condition has perfect predictive performance, as expected, across TPM, Last Seen and Most Frequent. Finally, all methods perform relatively poorly in the Random condition, with the only highlight in that, again, TPM is able to explicitly model and predict that this object moves frequently, and as a result this method has a higher-than-chance record in guessing the correct location of the object.

In summary, the above evaluation demonstrates that temporal persistence modeling enables the robot to predict object locations with greater accuracy than any of the baseline heuristic methods for a wide range of object displacement patterns. In our second evaluation, we examine how much data is required to obtain a reliable temporal persistence model.

In this study condition, we incremented the amount of training data available to TPM, beginning with 1 observation, then 2, 3, and so on. At each step we evaluated the performance of the resulting model using the same evaluation metric as previously described. Figure 7.7 presents the results from this analysis. As can be seen, all six TPM models converge, but following a different number of observations. As expected, the Stationary condition converges the fastest, requiring only a handful of observations. The remaining models require 35-85 data samples before a sudden jump in performance. Note that each observation indicates a "snapshot" the robot took at any location, and not necessarily an observation of that object which helps to explain the discrepancy for the number of observations.

This observed trend indicates that the performance of TPM is relatively low early on, but that $\mu_{i,s}$ converges quickly once it has enough data to come close to its optimal value. Given that $\mu_{i,s}$ is calculated based on a weighted average, more data only serves to strengthen (i.e., stabilize) $\mu_{i,s}$, not necessarily provide a better estimate. For example, with the plate (RA1), early observations gave reasonable

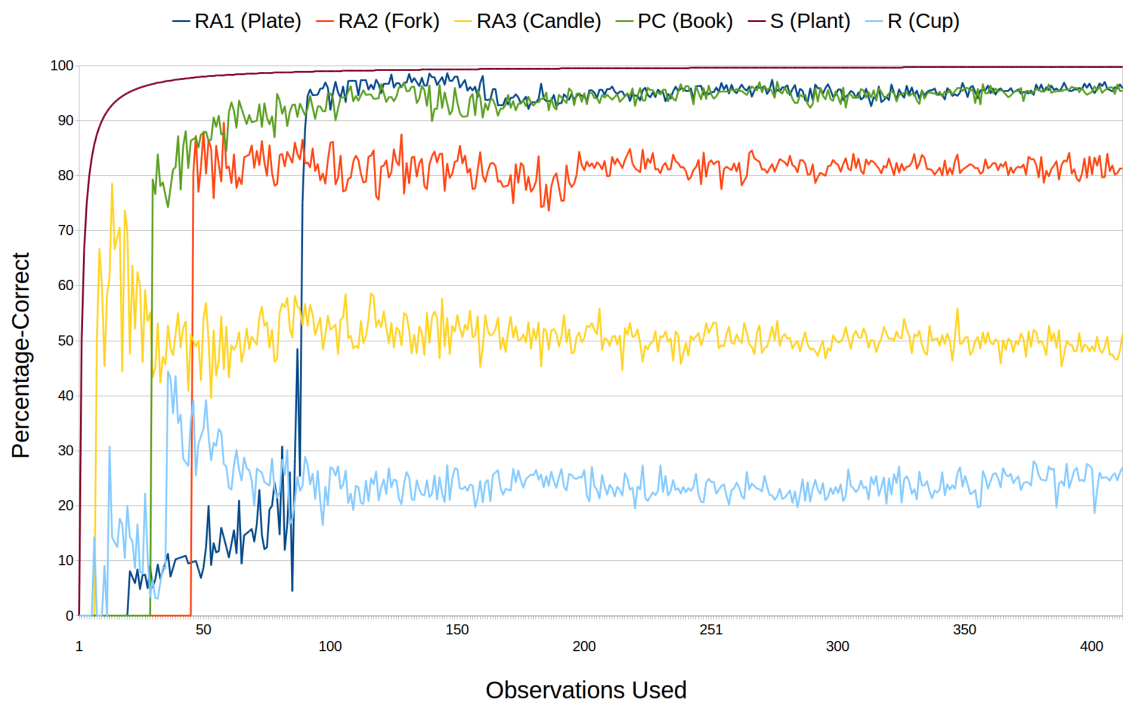


Figure 7.7: Evaluation results of TPM on increasing number of training samples.

estimates of the kitchen counter leading to the initial boost in performance, but once it was able to reason that $\mu_{\texttt{plate,kitchen-table}}$ was larger than $\mu_{\texttt{plate,counter}}$, the large jump was made. Additional observations only strengthen these values by pulling them farther apart.

What this analysis indicates is that a heuristic method, particularly Last Seen, performs better than TPM until the robot has obtained sufficient training data. Thus, from a practical deployment standpoint, it may be beneficial for the robot to utilize Last Seen at initial deployment, and then switch to TPM following convergence. In fact, this approach is highly analogous to human behavior in a new environment, where we initially rely on our raw observations and only over time develop a reliable predictive model of object temporal persistence (e.g., how long before coffee runs out in the common room, or where people usually put the shared stapler).

## 7.5 Conclusion

In summary, this chapter contributes a novel temporal persistence modeling algorithm that enables a robot to mode effectively model and predict likely object locations in its environment. Unlike past approaches in the areas of both active visual search and semantic mapping, we focus solely on temporally based relationships between objects and search locations. Our approach builds probabilistic exponential models in order to model when an object is no longer located on a given location given the last known time the object was observed on that location. In doing so, we compute a weighted average guess on what the rate parameter $\mu_{i,s}$ of the models are by sampling from a Gaussian distribution over the last known observation time and the current time at which the item was no longer seen on the surface.

We evaluated our approach using two data sets, evaluating both single-item tracking in a large-scale GPS location data set, and multi-item tracking in a real-world robotics deployment lasting two weeks. The performance of our algorithm dominates that of four baseline methods in both domains, and shows the importance of correctly estimating the $\mu_{i,s}$ parameter.

# CHAPTER 8
# CONCLUSION

## 8.1 Discussion

This thesis work presents a series of algorithms and methods for creating a goal-based learning framework based on high-level spatial and temporal (SPAT) observations and demonstrations. The SPAT framework is motivated by the lack of performance many LfD algorithms have when presented to non-expert end users. In particular, early work presented in [84, 73] exemplified this by asking non-expert users to train a set of traditional policy-based LfD methods with little success. This thesis also makes note of the benefits stated in [82, 79, 80, 81] in developing cloud and web-based solutions in order to facilitate brining robot interaction, control, and training to end users. As such, SPAT was developed in a cloud-oriented manner.

After an introduction into the background and related methods used in this work, we began with a presentation of novel efficient messaging techniques for streaming and receiving robotic data to and from the cloud. In particular, after examining the improved protocol and framework for Robot Web Tools, we looked at efficient techniques for streaming the high-bandwidth sensor streams related to kinematic transforms, video streams, 3D RGB-D point cloud streams, and generic message compression. We showed how a change-based transformation library, server-side web-based video compression, discretization and video encoding of 3D point clouds, and PNG compression of JSON ASCII based message formats added significant improvements in messaging across real-world data streams as compared to standard

ROS techniques. Furthermore, these techniques were developed in a way which is optimized for web-based streaming. In addition, we showcased a number of external examples of how this work has been utilized across the robotics research field to both allow for human-robot interaction techniques as well as data collection for robotic domains.

To begin tackling the problem of constructing goal states, we presented an algorithm for learning in an unsupervised manner from a set of data collected from non-expert users. Using Expectation Maximization, we were able to estimate Gaussian Mixture Models for each item and an associated potential reference frame. Each of the resulting models was then ranked using a heuristic designed to favor dense, populated clusters. Using non-expert crowdsourced users, we were able to show results across three different types of tasks: single-item, multi-item uncoupled, and multi-item templates. Given that the resulting models were both semantically grounded as well as spatially defined, the templates could then be used for execution by a mobile manipulated in a different environment.

After the above initial work in the spatial domain, we then expanded these templates to incorporate coarse temporal constraints. Adding this temporal component gave us our first complete component in the SPAT learning framework. As with the previous spatial method, the training was done in a completely unsupervised manner. The benefits of keeping the method as such enabled us to keep the teaching method the same requiring little expertise. For this task, we returned to the resulting clusters from before and added a new ranking heuristic which took a ratio of before/after binary values weighted by their likelihood within the cluster itself. The problem was then reduced into a weighted graph problem for which the minimal path was used as the execution order for the template. From this, we were able to explore its effectiveness across four block-world conditions: consistent order/consistent placement, random order/consistent placement, consistent order/random placement, and random order/random placement. Results indicated that the first two conditions were successfully captured by the methodology while the third condition showed partial success with the emergence of sub-templates. A further experiment returned back to the cloud-based household environment where we could verify the methodology using non-expert users in a noisy environment.

Next, to add the final component into the SPAT framework, we tackled the problem of searching for items for retrieval during task execution. By using only observations of semantically grounded spatial locations and temporal information, this final component would enable SPAT to complete a task from start to finish. For this, we presented persistence models which modeled each item/surface pairing as a exponential distribution. To overcome the problem of unobservable "failure cases" (i.e., not knowing when an item was removed from a surface), we augmented the models with a Gaussian component used to create estimates for the rate parameter. Initial testing in a real-world cell phone tracking environment showed its ability to reliably track a person within the world using only a sparse set of observations. When compared to other tracking and guessing strategies, we were consistently able to outperform each metric. Further experiments from a robotic apartment domain also showed similar results.

## 8.2 End Result

With the above contributions, we end with a final showcase of the process through its entirety. Again using the CARL robot, we show how using the SPAT framework with a cloud based database can enable CARL to set the table from start to finish without explicate training. For this demo, we require that CARL have:

1. A working perception system capable of detecting known objects [48].
2. A semantic map of the known surface locations within the world.
3. The ability to grasp and place items within the world[1].

[1]We thank and acknowledge David Kent for his assistance in training the robot for grasping.

First, we allowed CARL to make observations periodically throughout the day prior to task execution. This data was then used to create persistence models of where items are within the world. Next, using the crowd-interface, training data and models were generated for the table setting task with the single ordering constraint of the napkin. This information was processed and left for public access in a remote database. Using a browser based interface, we were then able to instruct CARL to begin setting the table. CARL then pulled down the existing template information from the online database and began searching for items within the world and placing them on the kitchen table. A video demonstration of this process is available at `https://www.youtube.com/watch?v=SY78VpU5l7c`. We note within the video how CARL was able to recover from a failed retrieval due to humans moving items within the world by updating its persistence models throughout execution.

This demonstration shows how using SPAT enables robots to learn from high-level, spatial and temporal information collected from both observations and non-expert users. The ability to train robots at a higher level enables users to easily provide training data for which many other approaches of LfD have failed to do.

## 8.3 Future Work

This work makes way for a number of potential extensions. In particular, we believe that extensions to the system could be made to build a much larger knowledge base. By constructing a large-scale database of task data, one potential extension could be to begin exploring the common relationships across tasks. For example, in many kitchen related tasks, there would likely always be some kind of relationship between forks and knives. If such relationships do exist consistently across different task domains, it may be possible to use these relationships as priors for weighting the reference frame. If this is true, it may be possible to learn more quickly, with few demonstrations, or overcome stronger noise.

Secondly, while our main contribution focused mainly on enabling a robot to act autonomously for task execution, one possible area of future work would be to explore ways in which human robot interaction can further the system. In particular, an interesting extension would be to allow the user to choose from different hypotheses from the database which could be stored as a local preference. Furthermore, the idea of template refinement, or re-declaration in its entirety, is left unexplored at this point. Methods in which a user could modify a template based on their own needs and propagate this information back to the global knowledge-base would further SPAT techniques to be more adaptive and general across environments.

Finally, in addition to expanding the work in the template generation component, future extensions are also possible in the persistence models as well. For example, an interesting area to explore is the ability to expand the models to not only predict how long an item remains on a surface and when it leaves, but also where it may have gone too. This extension would need to track movements of items across time but would allow for better failure recovery during execution. As with the above cases, the ability to feed this information back to a global database is also left as a potential area for further exploration.

# BIBLIOGRAPHY

[1] *Incremental Semantically Grounded Learning from Demonstration*, Berlin, Germany, 2013 2013.

[2] B. Akgun, M. Cakmak, K. Jiang, and A. Thomaz. Keyframe-based learning from demonstration. *International Journal of Social Robotics*, 4(4):343–355, 2012.

[3] B. Alexander, K. Hsiao, C. Jenkins, B. Suay, and R. Toris. Robot web tools [ROS topics]. *Robotics Automation Magazine, IEEE*, 19(4):20–23, December 2012.

[4] S. Alexandrova, M. Cakmak, K. Hsiao, and L. Takayama. Robot programming by demonstration with interactive action visualizations. In *Proceedings of the Tenth Annual Robotics: Science and Systems*, RSS '14, Berkeley, CA, USA, 2014.

[5] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57:469–483, May 2009.

[6] C. G. Atkeson and S. Schaal. Robot learning from demonstration. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 12–20, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[7] A. Aydemir, A. Pronobis, M. Gobelbecker, and P. Jensfelt. Active visual object search in unknown environments using uncertain semantics. *Robotics, IEEE Transactions on*, 29(4):986–1002, Aug 2013.

[8] A. Aydemir, K. Sjoo, J. Folkesson, A. Pronobis, and P. Jensfelt. Search in the real world: Active visual object search based on spatial relations. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2818–2824, May 2011.

[9] M. Beetz, M. Tenorth, and J. Winkler. Open-ease a knowledge processing service for robots and robotics/ai researchers. In *IEEE International Conference on Robotics and Automation (ICRA)*, page in press, Seattle, Washington, USA, 2015.

[10] A. Bergkvist, D. C. Burnett, C. Jennings, and A. Narayanan. WebRTC 1.0: Real-time communication between browsers. *Working Draft, W3C*, 91, 2012.

[11] E. A. Billing and T. Hellström. A formalism for learning from demonstration. *Paladyn, Journal of Behavioral Robotics*, 1(1):1–13, 2010.

[12] D. Birant and A. Kut. St-dbscan: An algorithm for clustering spatial–temporal data. *Data & Knowledge Engineering*, 60(1):208–221, 2007.

[13] P. T. Brandt, J. T. Williams, B. O. Fordham, and B. Pollins. Dynamic Modeling for Persistent Event-Count Time Series. *American Journal of Political Science*, 44(4):823–843, 2000.

[14] T. Chen, M. Ciocarlie, S. Cousins, P. M. Grice, K. Hawkins, K. Hsiao, C. Kemp, C.-H. King, D. Lazewatsky, A. E. Leeper, H. Nguyen, A. Paepcke, C. Pantofaru, W. Smart, and L. Takayama. Robots for humanity: A case study in assistive mobile manipulation. *IEEE Robotics & Automation Magazine, Special issue on Assistive Robotics*, 20, 2013.

[15] S. Chernova and A. Thomaz. *Robotic Learning from Human Teachers*. Morgan & Claypool Publishers, 2014.

[16] S. Chernova and M. M. Veloso. Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research (JAIR)*, 34(1):1–25, 2009.

[17] M. J.-Y. Chung, M. Forbes, M. Cakmak, and R. P. N. Rao. Accelerating imitation learning through crowdsourcing. In *IEEE International Conference on Robotics and Automation*, ICRA '14. IEEE, 2014.

[18] C. Crick, G. Jay, S. Osentoski, B. Pitzer, and O. C. Jenkins. rosbridge: ROS for non-ros users. In *Proceedings of the 15th international symposium on robotics research (ISRR)*, 2011.

[19] M. DeGroot and M. Schervish. *Probability and Statistics*. Addison-Wesley series in statistics. Addison-Wesley, 2002.

[20] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–38, 1977.

[21] T. Deyle, M. Reynolds, and C. Kemp. Finding and navigating to household objects with uhf rfid tags by optimizing rf signal strength. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 2579–2586, Sept 2014.

[22] J. G. Dy and C. E. Brodley. Feature selection for unsupervised learning. *Journal of Machine Learning Research*, 5:845–889, 2004.

[23] S. Edelkamp and J. Hoffmann. PDDL2.2: The language for the classical part of the 4th international planning competition. Technical Report 195, Institut für Informatik, Jan. 2004.

[24] S. Ekvall and D. Kragic. Learning task models from multiple human demonstrations. In *Robot and Human Interactive Communication, 2006. ROMAN 2006. The 15th IEEE International Symposium on*, pages 358–363, Sept 2006.

[25] S. Ekvall and D. Kragic. Robot learning from demonstration: A task-level planning approach. *International Journal of Advanced Robotic Systems*, 5(3):223–234, 2008.

[26] R. T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000. AAI9980887.

[27] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, IJCAI '71, pages 608–620, San Francisco, CA, USA, 1971. Morgan Kaufmann Publishers Inc.

[28] T. Foote. tf: The transform library. In *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, Open-Source Software workshop, pages 1–6, April 2013.

[29] T. Foote. ROS Community Metrics. 2014. `http://download.ros.org/downloads/metrics/metrics-report-2014-07.pdf`.

[30] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20, 2003.

[31] K. Fukunaga. *Introduction to Statistical Pattern Recognition (2nd Edition)*. Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[32] A. Gerevini and D. Long. Preferences and soft constraints in PDDL3. *Proceedings of ICAPS workshop on Planning with Preferences and Soft Constraints*, pages 46–53, 2006.

[33] B. Gerkey, R. T. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th international conference on advanced robotics*, volume 1, pages 317–323, 2003.

[34] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the planning domain definition language. Technical Report TR-98-003, Yale Center for Computational Vision and Control, 1998.

[35] J. J. Gibson. *The Ecological Approach To Visual Perception*. Psychology Press, new ed edition, 1986.

[36] K. Goldberg, editor. *The Robot in the Garden: Telerobotics and Telepistemology in the Age of the Internet*. MIT Press, Cambridge, MA, USA, 2001.

[37] D. Gossow, A. Leeper, D. Hershberger, and M. T. Ciocarlie. Interactive markers: 3-d user interfaces for ros applications [ROS topics]. *Robotics Automation Magazine, IEEE*, 18(4):14–15, 2011.

[38] G. Hu, W. P. Tay, and Y. Wen. Cloud robotics: architecture, challenges and applications. *Network, IEEE*, 26(3):21–28, 2012.

[39] J. F. Hughes, A. Van Dam, J. D. Foley, and S. K. Feiner. *Computer Graphics: Principles and Practice*. Pearson Education, 2013.

[40] D. Hunziker, M. Gajamohan, M. Waibel, and R. D'Andrea. Rapyuta: The roboearth cloud engine. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 438–444, May 2013.

[41] Y. Jiang, M. Lim, and A. Saxena. Learning object arrangements in 3D scenes using human context. In *Proceedings of the 29th International Conference on Machine Learning*, ICML '12, 2012.

[42] S. C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.

[43] D. Joho, G. D. Tipaldi, N. Engelhard, C. Stachniss, and W. Burgard. Nonparametric bayesian models for unsupervised scene analysis and reconstruction. In *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2012.

[44] J. D. Kalbfleisch and R. L. Prentice. *The statistical analysis of failure time data*, volume 360. John Wiley & Sons, 2011.

[45] V. Kalokyri, R. Shome, I. Yochelson, and K. E. Bekris. A single-switch scanning interface for robot control by quadriplegics. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems Workshop on Assistive Robotics for Individuals with Disabilities: HRI Issues and Beyond*, 2014.

[46] K. Kamei, S. Nishio, N. Hagita, and M. Sato. Cloud networked robotics. *Network, IEEE*, 26(3):28–34, 2012.

[47] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg. A survey of research on cloud robotics and automation. *Automation Science and Engineering, IEEE Transactions on*, PP(99):1–12, 2015.

[48] D. Kent and S. Chernova. Construction of an object manipulation database from grasp demonstrations. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 3347–3352, Sept 2014.

[49] T. Kollar and N. Roy. Utilizing object-object and object-scene context when planning to find things. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 2168–2173, May 2009.

[50] T. Kollar, M. Samadi, and M. M. Veloso. Enabling robots to find and fetch objects by querying the web. In *International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012 (3 Volumes)*, pages 1217–1218, 2012.

[51] P. Kormushev, S. Calinon, and D. G. Caldwell. Imitation learning of positional and force skills demonstrated via kinesthetic teaching and haptic input. *Advanced Robotics*, 25(5):581–603, 2011.

[52] D. L. Kovacs. BNF definition of PDDL3.1. Technical report, International Planning Competition, 2011.

[53] A. E. Leeper, K. Hsiao, M. Ciocarlie, L. Takayama, and D. Gossow. Strategies for human-in-the-loop robotic grasping. In *Proceedings of the Seventh Annual ACM/IEEE International Conference on Human-Robot Interaction*, HRI '12, pages 1–8, New York, NY, USA, 2012. ACM.

[54] M. Lorbach, S. Höfer, and O. Brock. Prior-assisted propagation of spatial information for object search. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, USA, September 14-18, 2014*, pages 2904–2909, 2014.

[55] J. Mason. *Object Discovery with a Mobile Robot*. PhD thesis, Duke University, Durham, North Carolina, USA, 2014.

[56] J. Mason and B. Marthi. An object-based semantic world model for long-term change detection and semantic querying. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, IROS '12, pages 3851–3858. IEEE, 2012.

[57] A. Mohseni-Kabir, C. Rich, and S. Chernova. Learning partial ordering constraints from a single demonstration (late-breaking report). In *Proceedings of the 2014 ACM/IEEE International Conference on Human-robot Interaction*, HRI '14, pages 248–249, New York, NY, USA, 2014. ACM.

[58] A. Mohseni-Kabir, C. Rich, S. Chernova, C. L. Sidner, and D. Miller. Interactive hierarchical task learning from a single demonstration. In *Proceedings*

*of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*, HRI '15, pages 205–212, New York, NY, USA, 2015. ACM.

[59] M. Montemerlo. *FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association.* PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, July 2003.

[60] D. Nau, M. Ghallab, and P. Traverso. *Automated Planning: Theory & Practice.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[61] M. N. Nicolescu and M. J. Mataric. Natural methods for robot task learning: instructive demonstrations, generalization and practice. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, AAMAS '03, pages 241–248, New York, NY, USA, 2003. ACM.

[62] S. D. Niekum. *Semantically Grounded Learning from Unstructured Demonstrations.* PhD thesis, University of Massachusetts Amherst, 2013.

[63] S. Osentoski, G. Jay, C. Crick, B. Pitzer, C. DuHadway, and O. C. Jenkins. Robots as web services: Reproducible experimentation and application development using rosjs. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 6078–6083. IEEE, 2011.

[64] S. Osentoski, B. Pitzer, C. Crick, G. Jay, S. Dong, D. H. Grollman, H. B. Suay, and O. C. Jenkins. Remote robotic laboratories for learning from demonstration - enabling user interaction and shared experimentation. *International Journal of Social Robotics*, 4(4):449–461, 2012.

[65] B. Pitzer, S. Osentoski, G. Jay, C. Crick, and O. Jenkins. Pr2 remote lab: An environment for remote development and experimentation. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3200–3205, May 2012.

[66] A. Pronobis. *Semantic Mapping with Mobile Robots.* PhD thesis, KTH Royal Institute of Technology, Stockholm, Sweden, 2011.

[67] A. Pronobis and P. Jensfelt. Large-scale semantic mapping and reasoning with heterogeneous modalities. In *2012 IEEE International Conference on Robotics and Automation (ICRA)*, ICRA '12, pages 3515–3522, May 2012.

[68] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. B. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[69] E. Ratner, B. Cohen, M. Phillips, and M. Likhachev. A web-based infrastructure for recording user demonstrations of mobile manipulation tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[70] M. Rausand and A. Høyland. *System reliability theory: models, statistical methods, and applications*, volume 396. John Wiley & Sons, 2004.

[71] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[72] M. Samadi, T. Kollar, and M. Veloso. Using the web to interactively learn to find objects. In *in Proceedings of the Twenty-Sixth Conference on Artificial Intelligence AAAI '12*, 2012.

[73] H. B. Suay, R. Toris, and S. Chernova. A practical comparison of three robot learning from demonstration algorithm. *International Journal of Social Robotics*, 4(4):319–330, 2012.

[74] P.-N. Tan, M. Steinbach, V. Kumar, et al. *Introduction to data mining*, volume 1. Pearson Addison Wesley Boston, 2006.

[75] K. Taylor and J. Trevelyan. A Telerobot On The World Wide Web. In *1995 National Conference of the Australian Robot Association*, Melbourne, July 1995. Australian Robotics Association.

[76] A. L. Thomaz and C. Breazeal. Adding guidance to interactive reinforcement learning. In *In Proceedings of the Twentieth Conference on Artificial Intelligence (AAAI)*, 2006.

[77] R. Toris. Evolving robotic desires: A new approach to bridging the reality gap. In *In Proceedings of PACISE 2011: Pennsylvania Association of Computer and Information Science Educators*, 2011.

[78] R. Toris. Evolving robotic desires: A new approach to bridging the reality gap. Dickinson College Institute Undergraduate Honors Thesis, May 2011. Honors Thesis.

[79] R. Toris. Bringing human-robot interaction studies online via the robot management system. Worcester Polytechnic Institute Masters Thesis, October 2013. Masters Thesis.

[80] R. Toris and S. Chernova. RobotsFor.Me and robots for you. In *Interactive Machine Learning Workshop, Intelligent User Interfaces Conference*, IUI '13, pages 10–12, March 2013.

[81] R. Toris, J. Kammerl, D. V. Lu, J. Lee, O. C. Jenkins, S. Osentoski, M. Wills, and S. Chernova. Robot web tools: Efficient messaging for cloud robotics. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 4530–4537, Sept 2015.

[82] R. Toris, D. Kent, and S. Chernova. The robot management system: A framework for conducting human-robot interaction studies through crowdsourcing. *Journal of Human-Robot Interaction*, 3(2):25, June 2014.

[83] R. Toris, C. Shue, and S. Chernova. Message authentication codes for secure remote non-native client connections to ROS enabled robots. In *Technologies for Practical Robot Applications (TePRA), 2014 IEEE International Conference on*, pages 1–6, April 2014.

[84] R. Toris, H. B. Suay, and S. Chernova. A practical comparison of three robot learning from demonstration algorithms (late-breaking report). In *Proceedings of the Seventh Annual ACM/IEEE International Conference on Human-Robot Interaction*, HRI '12, pages 261–262, New York, NY, USA, 2012. ACM.

[85] R. T. Vaughan, B. P. Gerkey, and A. Howard. On device abstractions for portable, reusable robot code. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 3, pages 2421–2427. IEEE, 2003.

[86] K. Wagstaff, C. Cardie, S. Rogers, and S. Schrödl. Constrained k-means clustering with background knowledge. In *ICML*, volume 1, pages 577–584, 2001.

[87] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Galvez-Lopez, K. Haussermann, R. Janssen, J. Montiel, A. Perzylo, B. Schiessle, M. Tenorth, O. Zweigle, and R. Van De Molengraft. Roboearth. *Robotics Automation Magazine, IEEE*, 18(2):69–82, June 2011.

[88] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[89] L. L. S. Wong. Living and searching in the world: Object-based state estimation for mobile robots. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (SIGAI Doctoral Consortium)*, pages 3085–3086, 2014.

[90] Y. Zheng, Y. Chen, X. Xie, and W.-Y. Ma. Geolife2.0: A location-based social networking service. In *Mobile Data Management: Systems, Services and*

*Middleware, 2009. MDM '09. Tenth International Conference on*, pages 357–358, May 2009.

# APPENDIX A

Table 1: Models for the “Consistent Order, Consistent Placement” Condition

| $\lambda^i$ | $\lambda^{\Theta_o}$ | $\Phi(\lambda)$ | $\lambda^i$ | $\lambda^{\Theta_o}$ | $\Phi(\lambda)$ |
|---|---|---|---|---|---|
| $\texttt{block}_{lb}$ | $P_{chair}$ | **0.3482** | $\texttt{block}_{lb}$ | $\texttt{block}_{mb}$ | **0.5063** |
| $\texttt{block}_{lb}$ | $\texttt{block}_{db}$ | **0.4542** | $\texttt{block}_{lb}$ | $\texttt{block}_{lg}$ | **0.5796** |
| $\texttt{block}_{lb}$ | $\texttt{block}_{dg}$ | **0.5209** | $\texttt{block}_{lb}$ | $\texttt{block}_{r}$ | **0.6390** |
| $\texttt{block}_{mb}$ | $P_{chair}$ | **0.3480** | $\texttt{block}_{mb}$ | $\texttt{block}_{lb}$ | **0.0063** |
| $\texttt{block}_{mb}$ | $\texttt{block}_{db}$ | **0.5100** | $\texttt{block}_{mb}$ | $\texttt{block}_{lg}$ | **0.5516** |
| $\texttt{block}_{mb}$ | $\texttt{block}_{dg}$ | **0.5173** | $\texttt{block}_{mb}$ | $\texttt{block}_{r}$ | **0.6551** |
| $\texttt{block}_{db}$ | $P_{chair}$ | **0.3499** | $\texttt{block}_{db}$ | $\texttt{block}_{lb}$ | **0.0992** |
| $\texttt{block}_{db}$ | $\texttt{block}_{mb}$ | **0.0636** | $\texttt{block}_{db}$ | $\texttt{block}_{lg}$ | **0.5125** |
| $\texttt{block}_{db}$ | $\texttt{block}_{dg}$ | **0.5533** | $\texttt{block}_{db}$ | $\texttt{block}_{r}$ | **0.6256** |
| $\texttt{block}_{lg}$ | $P_{chair}$ | **0.9019** | $\texttt{block}_{lg}$ | $\texttt{block}_{lb}$ | **0.0795** |
| $\texttt{block}_{lg}$ | $\texttt{block}_{mb}$ | **0.1112** | $\texttt{block}_{lg}$ | $\texttt{block}_{db}$ | **1.0000** |
| $\texttt{block}_{lg}$ | $\texttt{block}_{dg}$ | **0.5338** | $\texttt{block}_{lg}$ | $\texttt{block}_{r}$ | **0.5733** |
| $\texttt{block}_{dg}$ | $P_{chair}$ | **0.8976** | $\texttt{block}_{dg}$ | $\texttt{block}_{lb}$ | **0.0838** |
| $\texttt{block}_{dg}$ | $\texttt{block}_{mb}$ | **0.0689** | $\texttt{block}_{dg}$ | $\texttt{block}_{db}$ | **0.1144** |
| $\texttt{block}_{dg}$ | $\texttt{block}_{lg}$ | **0.0989** | $\texttt{block}_{dg}$ | $\texttt{block}_{r}$ | **0.5503** |
| $\texttt{block}_{r}$ | $P_{chair}$ | **0.8971** | $\texttt{block}_{r}$ | $\texttt{block}_{lb}$ | **0.1750** |
| $\texttt{block}_{r}$ | $\texttt{block}_{mb}$ | **0.5000** | $\texttt{block}_{r}$ | $\texttt{block}_{db}$ | **0.1724** |
| $\texttt{block}_{r}$ | $\texttt{block}_{lg}$ | **0.0741** | $\texttt{block}_{r}$ | $\texttt{block}_{dg}$ | **0.1133** |

Table 2: Models for the "Random Order, Consistent Placement" Condition

| $\lambda^i$ | $\lambda^{\Theta_o}$ | $\Phi(\lambda)$ | $\lambda^i$ | $\lambda^{\Theta_o}$ | $\Phi(\lambda)$ |
|---|---|---|---|---|---|
| $\texttt{block}_{lb}$ | $P_{chair}$ | **0.4414** | $\texttt{block}_{lb}$ | $\texttt{block}_{mb}$ | **0.2553** |
| $\texttt{block}_{lb}$ | $\texttt{block}_{db}$ | **0.2558** | $\texttt{block}_{lb}$ | $\texttt{block}_{lg}$ | **0.5072** |
| $\texttt{block}_{lb}$ | $\texttt{block}_{dg}$ | **0.5061** | $\texttt{block}_{lb}$ | $\texttt{block}_{r}$ | **0.6127** |
| $\texttt{block}_{mb}$ | $P_{chair}$ | **0.4419** | $\texttt{block}_{mb}$ | $\texttt{block}_{lb}$ | **0.2553** |
| $\texttt{block}_{mb}$ | $\texttt{block}_{db}$ | **0.2520** | $\texttt{block}_{mb}$ | $\texttt{block}_{lg}$ | **0.5079** |
| $\texttt{block}_{mb}$ | $\texttt{block}_{dg}$ | **0.5069** | $\texttt{block}_{mb}$ | $\texttt{block}_{r}$ | **0.6124** |
| $\texttt{block}_{db}$ | $P_{chair}$ | **0.3330** | $\texttt{block}_{db}$ | $\texttt{block}_{lb}$ | **0.2558** |
| $\texttt{block}_{db}$ | $\texttt{block}_{mb}$ | **0.2520** | $\texttt{block}_{db}$ | $\texttt{block}_{lg}$ | **0.5076** |
| $\texttt{block}_{db}$ | $\texttt{block}_{dg}$ | **0.5064** | $\texttt{block}_{db}$ | $\texttt{block}_{r}$ | **0.5940** |
| $\texttt{block}_{lg}$ | $P_{chair}$ | **0.6946** | $\texttt{block}_{lg}$ | $\texttt{block}_{lb}$ | **0.0072** |
| $\texttt{block}_{lg}$ | $\texttt{block}_{mb}$ | **0.0079** | $\texttt{block}_{lg}$ | $\texttt{block}_{db}$ | **0.0076** |
| $\texttt{block}_{lg}$ | $\texttt{block}_{dg}$ | **0.2554** | $\texttt{block}_{lg}$ | $\texttt{block}_{r}$ | **0.5070** |
| $\texttt{block}_{dg}$ | $P_{chair}$ | **0.8346** | $\texttt{block}_{dg}$ | $\texttt{block}_{lb}$ | **0.0061** |
| $\texttt{block}_{dg}$ | $\texttt{block}_{mb}$ | **0.0069** | $\texttt{block}_{dg}$ | $\texttt{block}_{db}$ | **0.0064** |
| $\texttt{block}_{dg}$ | $\texttt{block}_{lg}$ | **0.2554** | $\texttt{block}_{dg}$ | $\texttt{block}_{r}$ | **0.5045** |
| $\texttt{block}_{r}$ | $P_{chair}$ | **0.8862** | $\texttt{block}_{r}$ | $\texttt{block}_{lb}$ | **0.1139** |
| $\texttt{block}_{r}$ | $\texttt{block}_{mb}$ | **0.1136** | $\texttt{block}_{r}$ | $\texttt{block}_{db}$ | **0.0586** |
| $\texttt{block}_{r}$ | $\texttt{block}_{lg}$ | **0.0070** | $\texttt{block}_{r}$ | $\texttt{block}_{dg}$ | **0.0090** |

Table 3: Models for the "Consistent Order, Random Placement" Condition

| $\lambda^i$ | $\lambda^{\Theta_o}$ | $\Phi(\lambda)$ | $\lambda^i$ | $\lambda^{\Theta_o}$ | $\Phi(\lambda)$ |
|---|---|---|---|---|---|
| $\texttt{block}_{lb}$ | $P_{chair}$ | **0.4998** | $\texttt{block}_{lb}$ | $\texttt{block}_{mb}$ | **0.5893** |
| $\texttt{block}_{lb}$ | $\texttt{block}_{db}$ | **0.5859** | $\texttt{block}_{lb}$ | $\texttt{block}_{lg}$ | **0.5701** |
| $\texttt{block}_{lb}$ | $\texttt{block}_{dg}$ | **0.5687** | $\texttt{block}_{lb}$ | $\texttt{block}_{r}$ | **0.6189** |
| $\texttt{block}_{mb}$ | $P_{chair}$ | **0.8284** | $\texttt{block}_{mb}$ | $\texttt{block}_{lb}$ | **0.0893** |
| $\texttt{block}_{mb}$ | $\texttt{block}_{db}$ | **0.5294** | $\texttt{block}_{mb}$ | $\texttt{block}_{lg}$ | **0.5481** |
| $\texttt{block}_{mb}$ | $\texttt{block}_{dg}$ | **0.5532** | $\texttt{block}_{mb}$ | $\texttt{block}_{r}$ | **0.6058** |
| $\texttt{block}_{db}$ | $P_{chair}$ | **0.8268** | $\texttt{block}_{db}$ | $\texttt{block}_{lb}$ | **0.0859** |
| $\texttt{block}_{db}$ | $\texttt{block}_{mb}$ | **0.0294** | $\texttt{block}_{db}$ | $\texttt{block}_{lg}$ | **0.5471** |
| $\texttt{block}_{db}$ | $\texttt{block}_{dg}$ | **0.5495** | $\texttt{block}_{db}$ | $\texttt{block}_{r}$ | **0.5414** |
| $\texttt{block}_{lg}$ | $P_{chair}$ | **0.8728** | $\texttt{block}_{lg}$ | $\texttt{block}_{lb}$ | **0.0701** |
| $\texttt{block}_{lg}$ | $\texttt{block}_{mb}$ | **0.0481** | $\texttt{block}_{lg}$ | $\texttt{block}_{db}$ | **0.0471** |
| $\texttt{block}_{lg}$ | $\texttt{block}_{dg}$ | **0.5647** | $\texttt{block}_{lg}$ | $\texttt{block}_{r}$ | **0.5415** |
| $\texttt{block}_{dg}$ | $P_{chair}$ | **0.8766** | $\texttt{block}_{dg}$ | $\texttt{block}_{lb}$ | **0.0687** |
| $\texttt{block}_{dg}$ | $\texttt{block}_{mb}$ | **0.0532** | $\texttt{block}_{dg}$ | $\texttt{block}_{db}$ | **0.0495** |
| $\texttt{block}_{dg}$ | $\texttt{block}_{lg}$ | **0.0647** | $\texttt{block}_{dg}$ | $\texttt{block}_{r}$ | **0.5285** |
| $\texttt{block}_{r}$ | $P_{chair}$ | **0.6869** | $\texttt{block}_{r}$ | $\texttt{block}_{lb}$ | **0.0578** |
| $\texttt{block}_{r}$ | $\texttt{block}_{mb}$ | **0.0425** | $\texttt{block}_{r}$ | $\texttt{block}_{db}$ | **0.0414** |
| $\texttt{block}_{r}$ | $\texttt{block}_{lg}$ | **0.0415** | $\texttt{block}_{r}$ | $\texttt{block}_{dg}$ | **0.0285** |

Table 4: Models for the "Random Order, Random Placement" Condition

| $\lambda^i$ | $\lambda^{\Theta_o}$ | $\Phi(\lambda)$ | $\lambda^i$ | $\lambda^{\Theta_o}$ | $\Phi(\lambda)$ |
|---|---|---|---|---|---|
| $\texttt{block}_{lb}$ | $P_{chair}$ | **0.4137** | $\texttt{block}_{lb}$ | $\texttt{block}_{mb}$ | **0.3206** |
| $\texttt{block}_{lb}$ | $\texttt{block}_{db}$ | **0.4071** | $\texttt{block}_{lb}$ | $\texttt{block}_{lg}$ | **0.5509** |
| $\texttt{block}_{lb}$ | $\texttt{block}_{dg}$ | **0.5516** | $\texttt{block}_{lb}$ | $\texttt{block}_{r}$ | **0.6434** |
| $\texttt{block}_{mb}$ | $P_{chair}$ | **0.8392** | $\texttt{block}_{mb}$ | $\texttt{block}_{lb}$ | **0.3206** |
| $\texttt{block}_{mb}$ | $\texttt{block}_{db}$ | **0.3323** | $\texttt{block}_{mb}$ | $\texttt{block}_{lg}$ | **0.5383** |
| $\texttt{block}_{mb}$ | $\texttt{block}_{dg}$ | **0.5712** | $\texttt{block}_{mb}$ | $\texttt{block}_{r}$ | **0.6423** |
| $\texttt{block}_{db}$ | $P_{chair}$ | **0.7024** | $\texttt{block}_{db}$ | $\texttt{block}_{lb}$ | **0.2404** |
| $\texttt{block}_{db}$ | $\texttt{block}_{mb}$ | **0.3323** | $\texttt{block}_{db}$ | $\texttt{block}_{lg}$ | **0.5758** |
| $\texttt{block}_{db}$ | $\texttt{block}_{dg}$ | **0.5533** | $\texttt{block}_{db}$ | $\texttt{block}_{r}$ | **0.6521** |
| $\texttt{block}_{lg}$ | $P_{chair}$ | **0.8779** | $\texttt{block}_{lg}$ | $\texttt{block}_{lb}$ | **0.0509** |
| $\texttt{block}_{lg}$ | $\texttt{block}_{mb}$ | **0.0383** | $\texttt{block}_{lg}$ | $\texttt{block}_{db}$ | **0.0758** |
| $\texttt{block}_{lg}$ | $\texttt{block}_{dg}$ | **0.3096** | $\texttt{block}_{lg}$ | $\texttt{block}_{r}$ | **0.5372** |
| $\texttt{block}_{dg}$ | $P_{chair}$ | **0.8788** | $\texttt{block}_{dg}$ | $\texttt{block}_{lb}$ | **0.0516** |
| $\texttt{block}_{dg}$ | $\texttt{block}_{mb}$ | **0.0712** | $\texttt{block}_{dg}$ | $\texttt{block}_{db}$ | **0.0533** |
| $\texttt{block}_{dg}$ | $\texttt{block}_{lg}$ | **0.3096** | $\texttt{block}_{dg}$ | $\texttt{block}_{r}$ | **0.5321** |
| $\texttt{block}_{r}$ | $P_{chair}$ | **0.8234** | $\texttt{block}_{r}$ | $\texttt{block}_{lb}$ | **0.1314** |
| $\texttt{block}_{r}$ | $\texttt{block}_{mb}$ | **0.0931** | $\texttt{block}_{r}$ | $\texttt{block}_{db}$ | **0.1317** |
| $\texttt{block}_{r}$ | $\texttt{block}_{lg}$ | **0.0372** | $\texttt{block}_{r}$ | $\texttt{block}_{dg}$ | **0.0321** |

Table 5: Models for the Napkin Table Setting Task*

| $\lambda^i$ | $\lambda^{\Theta_o}$ | $\Phi(\lambda)$ | $\lambda^i$ | $\lambda^{\Theta_o}$ | $\Phi(\lambda)$ |
|---|---|---|---|---|---|
| $I_{Napkin}$ | $S_{DiningTable}$ | **0.8530** | $I_{Napkin}$ | $P_{chair}$ | **0.7309** |
| $I_{Napkin}$ | $I_{Plate}$ | **0.5713** | $I_{Napkin}$ | $I_{Cup}$ | **6.0543** |
| $I_{Napkin}$ | $I_{Spoon}$ | **0.7523** | $I_{Napkin}$ | $I_{Fork}$ | **0.6529** |
| $I_{Plate}$ | $P_{chair}$ | **0.2533** | $I_{Plate}$ | $S_{DiningTable}$ | **0.6967** |
| $I_{Plate}$ | $I_{Napkin}$ | **0.4612** | $I_{Plate}$ | $I_{Cup}$ | **0.4966** |
| $I_{Plate}$ | $I_{Spoon}$ | **0.4588** | $I_{Plate}$ | $I_{Fork}$ | **4.4767** |
| $I_{Cup}$ | $S_{DiningTable}$ | **0.8186** | $I_{Cup}$ | $P_{chair}$ | **0.7122** |
| $I_{Cup}$ | $I_{Napkin}$ | **0.4066** | $I_{Cup}$ | $I_{Plate}$ | **0.2852** |
| $I_{Cup}$ | $I_{Spoon}$ | **4.4341** | $I_{Cup}$ | $I_{Fork}$ | **2.7027** |
| $I_{Spoon}$ | $S_{DiningTable}$ | **0.8657** | $I_{Spoon}$ | $P_{chair}$ | **0.7288** |
| $I_{Spoon}$ | $I_{Napkin}$ | **7.9870** | $I_{Spoon}$ | $I_{Plate}$ | **0.2067** |
| $I_{Spoon}$ | $I_{Cup}$ | **5.9390** | $I_{Spoon}$ | $I_{Fork}$ | **0.7693** |
| $I_{Fork}$ | $S_{DiningTable}$ | **0.7776** | $I_{Fork}$ | $P_{chair}$ | **0.7264** |
| $I_{Fork}$ | $I_{Napkin}$ | **7.1259** | $I_{Fork}$ | $I_{Plate}$ | **0.2744** |
| $I_{Fork}$ | $I_{Cup}$ | **0.2980** | $I_{Fork}$ | $I_{Spoon}$ | **0.3721** |

*$\Phi(\lambda) > 1$ indicates a value of $\pi > \delta = 1.1$