ONE-SHOT, UNSUPERVISED LEARNING FOR IMPROVED HUMAN-ROBOT COLLABORATION

by

Joshua S. Vahala

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science
( Electrical Engineering)

at the

UNIVERSITY OF WISCONSIN-MADISON

2017
APPROVED

by

Advisor Signature: 

Advisor Title: Professor

Date: May 11, 2017
Joshua Vahala, Bilge Mutlu, and Robert Nowak

Abstract
As robots become commonplace in human environments such as households and manufacturing facilities, they require new control methods for effectively collaborating with their human partners in common pick-and-place tasks, such as unloading a dishwasher or providing parts for assembly. In this paper, we propose an online, generalizable task-modeling technique that enables robots to infer user task progress and determine their own path progress accordingly in order to optimize user experience and maintain good task performance. Furthermore, this method allows robots to actively build new task models when users temporarily or permanently switch to unknown tasks. We evaluate the algorithm’s effectiveness at building and updating task models using data collected on human performances of various pick-and-place tasks. We then implement the algorithm on a robot arm in a user study and show that the algorithm performs as well or better than the current state-of-the-art algorithm but with much less manual modeling effort.

Keywords
Human-Robot Collaboration; Collaborative Manipulation; User Experience; One-shot Learning; Unsupervised Learning; Task Recognition

1 Introduction
Robotic collaborators are becoming common in manufacturing, food service, home assistance, and entertainment where they must perform specific tasks consistently and reliably. Now that these systems are actually entering day-to-day human environments and interacting with non-experts, the needs of users, including safety, comfort, and preferences, become even more important. These non-expert users often perform the same tasks inconsistently across users and across time, so we need to allow robots to adapt accordingly. By teaching robots how to observe user tasks, we can build task models that allow them to adapt to changes in user actions and preferences over time and across tasks. Pick-and-place tasks are simple tasks, that can have significant user variability. These collaborative task models enable the robot to match the user’s task progress, thereby mimicking how a human collaborator would perform the task and improving user experience.

When a human-human team collaborates on a task, each individual is aware of all actions that occur within the task regardless of who is physically performing them (Sebanz et al. 2006). For instance, consider a scenario in which two individuals are collaborating to unload a dish rack, which was considered in prior work in robotics (Huang et al. 2015): a giver hands a dish to a receiver who places the dish in a cabinet. The giver understands how the receiver brings the dish to the cabinet. Further, the giver can adapt to unexpected receiver actions such as drying the dish with a towel or placing the dish in a new location. Because the giver knows the context of the task, the giver can figure out how to adapt to these types of variations.

Prior work (Huang et al. 2015) has shown that adaptive models that automatically match robotic actions to human task-progress provide better user-experience compared to proactive robotic models that aim to complete each task as quickly as possible, which may leave human collaborators feeling rushed by the purely proactive robot. While prior work has shown the promise of the use of such adaptive models, the construction of these models usually involves laborious manual processes for a single pick-and-place task type, including the hand-labeling of data collected from human interactions. In this paper, we present a method to automatically build adaptive models in a one-shot,
unsupervised fashion. We use the same pick-and-place task framework (see Figure 1) as the prior work, but in a mailroom sorting scenario. Given only the start times of each task, our technique generalizes to any pick-and-place task given only the task start times that can be gathered automatically such as when a pressure sensor signals that an object has left the robot’s gripper. We evaluate this approach on a new skeletal-joint-position dataset of several pick-and-place tasks with variable intermediate states. The goal of our evaluation is to measure the algorithm’s ability to adapt to new seen and unseen tasks as well as to new users. We then use these results to add sufficient heuristics for balancing user experience and task-completion speed in a user study with a robot arm.

This work contributes to the growing literature on human-robot collaboration through (1) the development of an unsupervised, generalizable task-modeling method for common pick-and-place tasks, (2) an implementation of the method as a python library for the robotics community to use, and (3) an evaluation of the effectiveness of the method off-line as well as on-line in a study with human subjects.

2 Background

Prior work on human action recognition and classification has explored the use of various modeling approaches, including Hidden Markov Models (HMMs) (Nava et al. 2014), bag-of-words methods (Wu et al. 2015), motion primitive methods (Zhao et al. 2013; Park and Howard 2010), and spectral clustering with dynamic time warping (Zhou et al. 2013). However, with the exception of methods such as semi-HMMs, these methods are designed without considering high task and individual variability inherent in real-world human actions and issues of data availability. In terms of online recognition, these otherwise powerful methods such as dynamic time warping can have high latency.

Alternative efforts to recognize human actions using online methods typically consist of various forms of augmented k-nearest-neighbors (KNN) matching to a set of states (Nava et al. 2014; Huang et al. 2015; Wu et al. 2012; Zanfir et al. 2013). We propose a new method of augmenting KNN that to our knowledge has not been applied in the literature.

Online human action recognition has been used to improve user experience without sacrificing task speed in several human-robot collaborative tasks. Most control methods derived from these methods involve studying human speed and intent within a motion path. HMMs have been used to segment future robot arm positions based on a user’s motions toward a known goal (Rozo et al. 2016), robot motion trajectories have been altered to match or avoid expected human hand trajectories (Lasota and Shah 2015; Pérez-D’Arpino and Shah 2015; Mainprice and Berenson 2013) or with respect to the path’s effect on human comfort levels (Mainprice et al. 2011), and robot end effector position has been matched to a user’s progress along an intuitively segmented task path (Huang et al. 2015). These methods all rely on expected actions of their users, whether those be known positions that the user may reach for in work by Lasota and Shah (2015) or Pérez-D’Arpino and Shah (2015), or from a set of supervised states as in work by Huang.
et al. (2015). However, these examples are constrained to specific, well-understood task models.

Despite these promising efforts, there is a need for more generalizable online methods that can learn and update models under limited data. We focus on robot adaptation to user timing similar to Huang et al. (2015), while most other work in this field adapts the robot motion path for collaborative environments (Dragan et al. 2015; Lasota and Shah 2015; Mainprice and Berenson 2013; Mainprice et al. 2011; Pérez-D’Arpino and Shah 2015).

3 Approach

In this paper, we propose a new method of building adaptive models that addresses our design goals of (1) allowing the robot to adapt with minimal training, (2) creating experiences for individuals, and (3) defining a framework that is simple to implement on a robotic collaborator. Our algorithm uses one-shot, unsupervised learning to define task models for the individual whenever a recently completed task is determined to be yet unseen. These task models are composed of a series of states and expected times that the user remains in each state. The robot adapts its own progress along its task path to the same progress of the user (i.e., if the algorithm determines the user to be 75% finished with the current task, the robot will move 75% of the way to the handover position). Thus, robots may quickly balance both user experience and task speed regardless of the task being performed.

We focus our application on pick-and-place tasks and a collaborative scenario where a robot is assisting its user in such a task. We provide the following definitions that will be used throughout this paper:

Definition 3.1. A pick-and-place task is a series of motions performed by the user that both begin and end with the user collecting the subsequent object. In this paper, this task is to be performed collaboratively as a handover between two parties, a user and a robot.

Definition 3.2. A state is a categorical pose that a user enters during a pick-and-place task. Each state is dependent upon the task being performed and the features being considered.

Definition 3.3. A task model is the ordered set of states and expected times associated with each state that a user follows to complete an instance of a particular pick-and-place task. The task model also includes the the most relevant features for the particular task.

More explicitly, the proposed approach contains two main steps: an online percent complete determination step, and a task-model update step.

Definition 3.4. The percent complete of an ongoing task refers to how much of the task the user has completed. For example, if the user is performing a handover task, then the user is 0 percent complete when collecting the object and 100 percent complete when reaching to collect the next object after processing it. Only 0 and 100 percent are well defined, and the rest of the task (e.g., placing the object in a bin) comprises all other percents.

The online step processes incoming skeletal-joint-based features defined for known task models through a probabilistic KNN and outputs an estimated percent complete of the current user task. The update step compares the newly completed task model to known task models through Dynamic Time Warping (DTW) (Müller 2007) and defines a new task model or updates a known task model through spectral clustering (von Luxburg 2007) or an interpolation-based subspace projection method. For each successive task, the online percent complete value is relayed to the robot to control its position. We explicitly detail our approach in two parts: Task Recognition and Robot Control. Our task recognition algorithm can be visualized in full as an online process that determines an estimated percent complete value (see Figure 2), and an offline process that updates our known task models (see Figure 3).
Figure 2. The online portion of our algorithm processes each incoming frame of joint data to estimate a current percent complete for the user’s current task. At least one known task model is required to output a percent complete, otherwise the algorithm will wait until the first task complete signal.

Figure 3. Once a task complete signal is received (e.g., the robot force sensor triggers due to the user pulling the object from the gripper), we compare the newly completed task to our known task models and either create a new task model for our dictionary of models, or we update a known task model.

3.1 Task Recognition: Task Modeling and Model Refinement

At a high level, upon each completion of a pick-and-place task (see Figure 4), the new task is compared to known task models and the task dictionary is either appended or updated. We use Dynamic Time Warping (DTW) to determine how similar a new task is to previously seen tasks that reside in our task models. If a new task model is to be created, then we use our feature determination method to define that task’s key features, and we use a spectral clustering method to define series of states that those features follow. If the new task was classified as a previously seen task type, then that task model is updated via an interpolation and redistribution method. The following subsections describe specifically what features are used, how they are mapped onto time, and how they are clustered into states.

3.1.1 Feature Selection

In order to learn and adapt to user and task models, we employ real-time skeletal tracking with the Microsoft Kinect* v2 RGBD sensor. The sensor returns 25 different joint positions as \((x,y,z)\)-coordinates (Head, SpineMid, ShoulderLeft, KneeRight, HandLeft, etc.) accurate to within 50–100 mm at a 30 frames per second (fps) rate (Wang et al. 2015).

We consider \(n\) of the 25 possible joints to form the set of joint positions \(\mathbf{J} \in \mathbb{R}^{n \times 3}\). The selection of which \(n\) joints to use as incoming data should be

informed by the expected tasks that this algorithm will be used on, but the algorithm is generalizable enough to ignore any selected joints that are not used in the task. For example, the HandLeft and HandRight joints are obviously very useful for a pick-and-place task. HipLeft is less important to include, but if the implementer does include it in the n selected joints, the algorithm will likely not include it in determining features. For each timestep, we consider two different sets of features: Interjoint features (I) and Joint-to-Midpoint features (M):

Definition 3.5. Interjoint features $I_k \in \mathbb{R}$ are defined as all pairwise euclidean distances of the n joints:

$$I_k = \|J_i - J_j\|_2 \quad \forall i \neq j; \quad j \geq i; \quad i, j \in \{1, ..., n\}$$  \hspace{1cm} (1)

Definition 3.6. Joint-to-Midpoint $M_k \in \mathbb{R}$ features are defined as the euclidean distance between all n joints to a fixed point $m \in \mathbb{R}^3$ (potentially the point of interaction between the user and robot):

$$M_k = \|J_i - m\|_2 \quad \forall i \in \{1, ..., n\}$$  \hspace{1cm} (2)

We normalize each feature value by a distance proportional to the human body (e.g., torso length) in order to help with inter-person classification. Then, we further normalize all features by their max value within the first seen instance a task to better compare larger features such as FootLeft-to-Head to shorter features such as ShoulderLeft-to-Head. This setup allows us to generalize across users and across tasks using the same features.

This number of total features $(N = |I| + |M| = \binom{n}{2} + n)$ can easily outnumber the total frame-count captured by the sensor for a generic task, thereby not being useful for classification after only a single task’s worth of data is collected. For example, consider a general pick-and-place task that lasts five seconds, including $T = 150$ total frames of data. If we have selected $n = 20$ joint positions as our inputs, there are $N = 210$ features to consider. Using such a large number of features would make it difficult to quickly compute new task models or perform online task recognition. In order to reduce the dimensionality of our high dimensional features from $\mathbb{R}^N$ to a reasonable size that can be quickly processed, we use a two-step method on each feature set $I$ and $M$ (for ease of notation, we denote a generic feature set with $\Delta$ and the number of features within the set with $|\Delta|$):

1. **Group similar features**: Comprise individual features into groups of similar features. We define similarity matrices $S^\Delta \in \mathbb{R}^{|\Delta| \times |\Delta|}$ on each feature set $\Delta$ such that

   $$S^\Delta_{i,j} = \exp(-\|\delta_i - \delta_j\|^2)$$  \hspace{1cm} (3)

   where $\delta_i$ denotes the $i$-th feature of the feature set $\Delta$. If two features $\delta_i$ and $\delta_j$ are similar, then $S^\Delta_{i,j}$ will be close to 1. We define two features $i, j$ to a subset if $S_{i,j} > \tau$, where $\tau$ is determined empirically as the threshold for considering features as redundant. An optimal value of $\tau$ is not important and the same $\tau$ can be used for any implementation of this algorithm. These subsets are then closed such that for features A, B, C, and D in subsets $\{A,B\}, \{B,C\}, \text{and } \{D\}$, the resulting set of closed sets, $G^\Delta = \{\{A,B,C\}, \{D\}\}$, is length $l^\Delta = 2$.

2. **Select active features**: Measure each group’s activity over the full task by comparing it to the feature’s “resting state,” which we approximate using the median. A candidate feature $G^\Delta_i$ from each closed group is tested for changes over the task timeframe to get an activity value

   $$A_i = \sum_{k=1}^{T} |G^\Delta_i(k) - \text{median}(G^\Delta_i)|$$  \hspace{1cm} (4)

where $T$ is the number of task frames and $G^\Delta_i(k)$ denotes the $k$-th frame of the candidate feature from the $i$-th group of $G^\Delta$. Candidate features with large $A_i$ are added as the task’s base features. In order to avoid defining what “large” means in this context, we instead select the four features with largest $A_i$ from each feature set $\Delta$ for a total of eight features.
Grouping is especially important to avoid cases of poor joint selection where redundant features such as \textit{HandLeft-to-HandTipRight} and \textit{WristLeft-to-ThumbRight} are both found to be highly active.

This method of feature selection is generalizable to any choice of base features in that it will select a subset of independent time-varying features. Thus, many other feature choices including dynamic features and joint-angle features are possible in this framework.

### 3.1.2 Task Similarity Determination

Dynamic Time Warping (DTW) is a well known time-series matching technique that scales the time axes of two signals, minimizing the euclidean distance cost of matching points (Müller 2007). By placing a relatively strict constraint that the time axis of each signal not vary by more than four percent of the other, the cost primarily depends on the time matching of signals and not small variations within. This four percent constraint is consistent with peak accuracy results for DTW and allows for quicker calculation than the typical 10 percent constraint inherited from the audio processing community (Ratanamahatana and Keogh 2005).

Dynamic Time Warping (DTW) is used to compare two sequences $X = \{x_0, x_1, \ldots, x_n\}$ and $Y = \{y_0, y_1, \ldots, y_m\}$. We define a cost measure $c_{ij} = |x_i - y_j|$ for all sequence indices $i$ and $j$. Our goal is to find the minimum cost path from $(0, 0)$, the start of each sequence, to $(n, m)$, the end of each sequence. The path of indices cannot go backwards in time. In other words, our path of indices could not be: $(0,0), (0,1), (1,2), (1,1)$. Each index must increase or stay the same. In this formulation, similar sequences will result in smaller costs.

We perform this minimization under a window constraint such that $i$ and $j$ cannot be more than four percent of the max sequence length from each other, as recommended by Ratanamahatana and Keogh (2005). In other words, if each sequence $X$ and $Y$ were of length 100, then the position $(10, 20)$ would not be valid.

We use DTW to compare a newly complete task’s feature signal to our known task model’s \textit{Genesis Signal}.

![Figure 5. Gamma random variable probability distribution functions (black) are fit to same-task and different-task DTW cost histograms (green and blue). The vertical axis shows the values of the fitted probability functions. The histograms have a y-axis that is normalized by the total number of comparisons. When the same task is compared (blue), we calculate low DTW costs. When different tasks are compared (green), we calculate higher DTW costs depending on how different the tasks seemed. We compute the intersection of the two fitted distributions to acquire a reasonable cost threshold of 1.79.](image)

\textbf{Definition 3.7.} A task model’s \textit{Genesis Signal} is defined as the feature signal that led to the creation of the task. It is comprised of $k$ base features chosen based on independence and activity seen throughout the task.

For each known task model, we record the DTW cost between each of the $k$ known task model base features and the corresponding feature of the newly completed task. We then average each base feature DTW cost to obtain the mean feature cost $\bar{C}$ and compare it to an empirically derived threshold $\gamma$:

$$\bar{C} = \frac{1}{k} \sum_{i=1}^{k} C_i$$

This threshold defines how sensitive this algorithm is to small task variations. Too small of a $\gamma$ will result in similar tasks being defined as unique task models, while too large of a $\gamma$ will result in disparate tasks being grouped into the same model.

We select an appropriate task differentiation cost threshold $\gamma$ by collecting joint-position data for 10 iterations of 10 unique pick-and-place tasks performed...
by one of the authors. For each task type, we handpicked six features to consider (three interjoint features, three joint-to-midpoint features) as a small and constant set of features to consider for many different task types. By using the same features for all task types instead of the features that would be selected by our algorithm, we achieve a more conservative empirical DTW cost threshold. Then, we calculated each feature’s DTW costs comparing different iterations of the same task type as well as comparing iterations of different task types. By constructing same-task and different-task histograms of the DTW costs (actually, the median DTW cost of the six feature comparisons), we can see a clear difference between task types (see Figure 5). We then fit a Gamma random variable probability distribution function to each histogram and found the point of intersection. This point maximizes the likelihood of successfully classifying two task iterations as the same type or different types.

In practice, a somewhat higher threshold is preferred. The data used to define this threshold, described in Section 4, was in a controlled environment and had very limited variance between task iterations. In other words, each task was relatively simple and short in length, so variations between iterations were minimal. Some task types also include very similar timings with only slight variations in movements, so some task differentiations assumed by this empirical threshold are unnecessary. Thus, in practice we push our threshold to a slightly larger value (around 3.0) to not needlessly add new task models for slight variations of known task models.

3.1.3 State-Transition Path Determination

Each task model is composed of a series of states that the user follows to complete the task, called a state-transition path. Each state can be thought of as a categorical pose that the user assumes throughout the process for a specified amounts of time. When we need to determine a new state-transition path from the time series task data, we use spectral clustering.

Spectral clustering is an unsupervised algorithm similar to k-means clustering, but specializes in finding connectedness rather than spatial groups. The general approach is to perform k-means clustering on a few eigenvectors of the graph Laplacian basis of an enumerated dataset. Spectral clustering first requires creating a symmetric similarity matrix $W \in \mathbb{R}^{m \times m}$ from our $m$ frames of task features $f_i \in \mathbb{R}^n$, where $n$ is the total number of features, such that

$$W_{ij} = e^{-\frac{1}{2} \|f_i - f_j\|^2}.$$  

This formulation gives us a dense graph from which to find structure. We make this graph sparser by keeping only the largest $r = 6$ values in each row of $W$. Next we
calculate the diagonal degree matrix $D \in \mathbb{R}^{n \times n}$ such that

$$D_{ii} = \sum_j W_{ij}$$

which simply says that each entry along the diagonal is the sum of each row of the similarity matrix. Finally we calculate the Laplacian $L \in \mathbb{R}^{n \times n}$ of the similarity matrix $W$ with degree matrix $D$ such that

$$L = D - W$$

We refer to the Graph Laplacian Basis (GLB) as the singular vector associated with the smallest $d = 2$ singular values (one singular value will be zero). Lastly, the GLB is clustered using the k-means++ algorithm to define labels for each of the $m$ frames in the task.

3.1.4 Add Task Model. When $\bar{C} > \gamma$ (see Equation 5), a new task is added. We add task models to allow the robot to seamlessly adapt between both completely new task types and high variations of known task types. The procedure to add a new task model goes as follows:

1. New active features are determined for the task (see Section 3.1; different pick-and-place tasks such as place-object-in-cabinet and place-object-on-floor may have uniquely expressive features).
2. Spectral clustering, a well known unsupervised clustering algorithm (von Luxburg 2007), groups features into a small set of clusters, thereby defining an initial state-transition path (STP) in the graph Laplacian basis (GLB). See Figure 6(top).
3. The GLB is updated to potentially add or remove new states based on the unique basis value concentration locations, thus allowing the implementer to avoid testing many potential cluster quantities in the prior step. See Figure 6(bottom).

3.1.5 Update Known Task. When $\bar{C} < \gamma$ (see Equation 5), the known task model associated with $\bar{C}$ is updated with the new task features. We update a known task model if a newly completed task is sufficiently similar to a known task. This process allows our algorithm to average out all task examples of a particular type into a single expected task type. The procedure for updating a known task goes as follows:

1. The GLB is calculated from the new task features.
2. Uniformly selected point-interpolation (or point-removal, if the new task took more frames to complete than the known task) is used to match the length of the new GLB to that of the known task, taking advantage of the connectedness of the GLB.
3. The new GLB is projected onto the subspace spanned by the known GLB.
4. Previously interpolated points are then removed (or previously removed points are re-interpolated) from the correct indicies of the projected GLB.
5. States are then redistributed as in Step 3 of “Add Task Model” to determine the new state-transition path. Expected times in each state are averaged among all seen task examples.

Thus, more examples of a task results in the averaging of the STP and expected times of each state, enabling a more consistent online understanding of the task.

3.2 Task Recognition: Online Task Comprehension

We create a new method named Rayleigh Probabilistic KNN (RP-KNN) to map incoming real-time features to the appropriate states within each known task model. Each known task model is treated separately to output individual percent complete estimates, assuming that the individual known task model is correct. We can weight these individual percent complete estimates by a task probability measure to determine a robust task-progress estimate. Thus, the estimated ongoing task is presented as a linear combination of the known task models that look most similar to the ongoing task.

3.2.1 Rayleigh Probabilistic KNN. Due to the “one-shot” execution of this algorithm, each known task is not guaranteed enough data points to successfully
A mixed Rayleigh distribution is defined as each new state is passed. The x-axis shows the number frames that have passed since the last state change. As time continues, the current state becomes proportionately less likely to occur relative to later expected states within a state-transition path.

We define a mixed Rayleigh distribution for the $z$ remaining states in an STP with individual pdfs

$$f_i(x; \sigma_i) = \frac{x}{\sigma_i^2} \exp\left(-\frac{x^2}{2\sigma_i^2}\right)$$

where $i \in \{1, ..., z\}$ and scale parameters $\sigma_i$ chosen such that $f_i$ intersects with $f_{i-1}$ at the expected time since the last state change. This expected time is determined by the expected times of each state in the STP. When a state changes, the mixed Rayleigh is redefined for whichever states remain in the STP.

KNN initializes a state-guess-distribution $U = \{u_1, ..., u_z\}$ such that $\sum_i u_i = 1$ for each state $i \in \{1, ..., z\}$ defined in the STP:

$$u_i := \frac{d_i}{k}$$

with $d_i$ being the count of the $i$-th state showing up in the $k$ nearest neighbors of the new feature, where $k = \sum_i d_i$ is the total number of nearest-neighbors in consideration. Note that $k$ should be chosen with respect to the shortest possible task length to ensure there are enough examples to consider for one-shot learning.

Then, as the current time advances, the proportions of each $f_i$ are used to alter $U$ (see Figure 7), thereby weighting future states more heavily than tasks that should have been completed at this time frame. However, any $u_i = 1$ (meaning all nearest neighbors of the input feature are in state $i$) will not be affected by the weighting, as expected. For each known task model $v$, the percent complete $p_v$ is determined from the current position $t$ within a state $s_i$ and the expected times of all future states. Note that $s_i$ denotes the $i$-th state within the task model $v$. For an $n$ state process, this amounts to

$$p_v = 1 - \frac{\max\{0, T(s_i) - t\} + \sum_{j=i+1}^n T(s_j)}{\sum_{j=1}^n T(s_j)}$$

where $T(s_i)$ is the expected time that state $i$ of task $v$ takes to complete and $t$ is the time since entering this state $s_i$. See Figure 10 for the online percent complete values associated with three known tasks models.
We now have a new task model, Task 1, in our model library. An unseen task type (a phone call interrupts the typical handover task) is completed. The new task type has highly dissimilar movements compared to the known task models 1 and 2, thus the error metric quickly increases and an error-correcting heuristic activates to smoothly send the robot to its next handover position. DTW later confirms that an e wt a s km o d e ls h o u l db ea d e do n c et h et a s ki s completed.

Figure 10. We now have a new task model, Task 3, in our model library. A new example of the same task type that generated Task 0 is completed. As the task progresses, the algorithm drops both models 1 and 2 from consideration and the estimated percent complete converges to the Task 0 percent complete value.

3.2.2 PID Controlled Error and Task Probability. In order to determine the probable current task of the m known models, we define an error-per-frame $e_v^v(k)$ for each task model $v \in \{1, \ldots, m\}$:

$$e_v^v(k) = \frac{1}{e_w} \sum_{i=k-w}^{k} [E[p_v^v(i)] - p_v^v(i)]^2 \quad (9)$$

where $k$ is the current frame number, $e$ controls errors from diverging too quickly, $w$ is the window to average over, and $E[p]$ is the expected percent complete if the task were to follow its STP perfectly (the percent complete should increase linearly with time). See Figures 8 through 10 for an example of how the percent complete estimate changes as the model progresses over three consecutive task types.

We take the softmax proportion of errors between tasks to determine the associated probability $\lambda^v(k)$ that task $v$ is ongoing at frame $k$:

$$\lambda^v(k) = \exp \left[ -\alpha e_v^v(k) \right] \frac{1}{\sum_{i=1}^{m} \exp \left[ -\alpha e_i^i(k) \right]} \quad (10)$$

where $\alpha$ is chosen as a scaling parameter based on the expected differences in errors. We then weight the individual percent complete estimates by these probabilities at each timeframe to acquire an estimated percent complete.

We further separate $e(i)$ values through PID control to more precisely weight each $p_v^v$ and differentiate similar error values by how they are actively changing and how they have behaved earlier in the task. So for each $e_v^v(k)$ denoted as $e_k$:

$$e_k^v = Pe_k + Ne_i + De_k e_{k-1} \quad (11)$$

The proportional term, $Pe_v^v(k)$, sends large errors to extremes. The integral term, $Ne_i \sum_{i=1}^{k} e_v^v(i)$, gives task models with recent errors precedence over those with consistent historical errors. The derivative term, $De_k e_v^v(k)$, penalizes increasing errors and supports decreasing errors.

Applying soft-thresholding on $e_v^v(k)$ can limit extravagant changes in early time frames, thus allowing for STPs to diverge or converge more naturally than without the soft-threshold. The proportional term above becomes $[P \max\{1, e_v^v(k) - \rho\}]$. This addition works especially well when the soft-threshold value $\rho$ starts large and ends small.
The final estimated percent complete \( p^* \) at frame \( k \) is then defined as

\[
p^*(k) = \sum_{i=1}^{m} \lambda^{(i)} p^{(i)}
\]

where \( \lambda^{(i)} \) and \( p^{(i)} \) are the associated probabilities and percent complete estimates of task model \( i \) at the current frame \( k \).

### 3.3 Robot Control

From this percent complete estimate, a robot can alter its own pace to match that of the user. We propose a linear actuation control method in which \( p^*(k) \) is mapped directly to a predefined robot motion path where for a generic handover task, 0% maps to “release object,” 50% maps to “collect new object,” and 100% maps to “release object,” which resets to 0% complete. This motion can be adapted if the release point should be different for different tasks, and the individual task probabilities can weight specific end effector positions in 3D space.

Each path is represented as a series of robot joint waypoints linearly interpolated throughout the base path. For a 50-waypoint path, each waypoint represents a two percent change in path progress.

The robot accepts a new percent complete value at a set rate from the task modeling algorithm. The robot then travels to this percent complete value via the waypoint path. If velocity control is available for the robot, then the speed of advancement can be mapped to the change in current percent complete to the new percent complete. In general, this method allows for the robot to take on the “slowing down” and “waiting” methods of collaboration that were used by Huang et al. (2015).

Further, the percentage estimate allows the robot to effectively collaborate in complex processes that involve the robot completing auxiliary tasks while the user completes the main task. Consider a cooking scenario in which a robot arm is tasked with handing a series of ingredients to the chef. While the chef processes each ingredient (e.g., stirs, puts in oven, etc.), the robot could utilize the 0% – 30% range of task-progress to prepare an ingredient needed in the future (e.g., cutting carrots to be added later in the recipe). When the user is considered to be out of this acceptable auxiliary task range, the robot can collect the next object and continue its adaptive handover strategy. Thus, this process of task-progress estimation allows for a simple addition of auxiliary tasks.

### 4 Algorithm Evaluation

We evaluate this algorithm’s performance through its ability to both provide a reasonable online metric for percent complete and to properly update or append known task models for both \textit{intra-user} and \textit{inter-user} scenarios. The \textit{intra-user} evaluation tests task models defined for a single user, while the \textit{inter-user} evaluation tests task models trained on one user, then tested on another.

In a human-subjects study approved by our Institutional Review Board, eight (8) individuals (four female, four male) ages 21 to 28 (\( \mu = 24.25 \), \( \sigma = 2.33 \)) completed ten (10) unique task types for ten (10) iterations each. Thus, a total of 800 task iterations were completed for this dataset. Each task is recorded as the \( x,y,z \) coordinates of fifteen (15) joints along with the timestamp and participant ID number “1” through “8” as the participant completed the ten iterations of the task type.

Each task can be described as a three step process in which the participants pick up plates from a table to their left, perform an intermediate task, then place the plate to their right before picking up the next plate. The intermediate tasks vary from low to high complexity and are based in kitchen or manufacturing scenarios. Each task was recorded using the sensor to collect the following joints’ coordinates:

- Head
- Neck
- SpineShoulder
- SpineMid
- SpineBase
- ShoulderLeft

Prepared using sagej.cls
• ShoulderRight
• ElbowLeft
• ElbowRight
• HandLeft
• HandRight
• KneeLeft
• KneeRight
• FootLeft
• FootRight

4.1 Low Complexity Task Types

Low complexity tasks do not have an explicitly defined intermediate task. Instead, the variation comes from where the plate is to be placed on the participant’s right hand side.

1 Floor Handover - The plate is placed into a bin located on the floor, which requires the participant to bend at the knees and hips to access.

2 Counter Handover - The plate is placed into a bin located at waist height. This task type is considered the basic handover process.

3 Cabinet Handover - The plate is placed into a bin located at cabinet height, which requires the participant to reach high with at least one arm to drop the plate.

4.2 Moderate Complexity Task Types

Moderate complexity tasks include a relatively simple intermediate task that is repeatable without any concentration by the participant. Without this additional intermediate task, the process is the same as task type 2 described above as the basic handover process.

4 Spot Inspection - The participant holds the plate up to the ceiling light with both hands to check for spots before placing it on the counter.

5 Towel Wipe - The participant, holding a paper towel, wipes the dish in a circular manner to clean the plate before placing it on the counter.

6 Object Detail Identification - The participant turns the plate over and vocal explains to the experimenter what shape they see on the reverse side before placing it on the counter.

7 Phone Call - The participant picks up a phone from the counter in front of them, says they are busy into the phone, and hangs it up before continuing to place the plate on the right hand side counter.

8 Signature - The participant sets down the plate on the counter in front of them to unc Cap a pen, sign their name on a sheet of paper, and recap the pen before continuing to place the plate on the right hand side counter.

4.3 High Complexity Task Types

High complexity tasks include a more complicated intermediate task than the moderate complexity tasks, but also follow the same basic handover process.

9 Cooking Scenario - The participants pretend the plate holds the next ingredient they need to add to their cooking pot. They take the plate, dump the imaginary contents into a bowl on the counter in front of them, stir the bowl with a spoon for a few revolutions, and then continue to place the plate on the right hand side counter.

10 Assembly Scenario - The participants pretend the plate is a kitting box that must be assembled. They take the plate, place it on the counter in front of them, collect a yellow and red wire from small bags that hold the respective colors and place each wire on the plate before continuing to place the plate on the right hand side counter.

4.4 Evaluation Goals

We base the algorithm evaluation on the final percent complete value, which is the estimated percent complete returned by the algorithm at the frame when the task should be completed. For example, assume a participant started the fifth iteration of Task 3 at frame 500 of the dataset and finished the iteration at frame 600. The algorithm is evaluated starting with frame 500 as 0% complete, and after 100 frames have elapsed, the algorithm should ideally be at 100%. The
actual percent complete value at frame 600 is recorded for evaluation. The individual tasks frames at each beginning and end were marked by hand as the point at which a new plate was picked up by the participant. We believe this to be an accurate measure of the success of this algorithm for two reasons:

1. The tasks are assumed to be collaborations with robotic partners, so the interaction point (i.e., when the new objects are acquired) is the most important point for measuring subjective collaboration.

2. The algorithm must progress through the individual states of each known task model’s state-transition path in order to reach a satisfactory final percent complete value. Thus, this measure of success naturally includes how well state-transition paths are followed.

We also treat all task types equally by randomizing the choice of which task types are seen together when the number of unique task types $q$ is greater than one. Thus, we show the generalizability of the algorithm to any combination of task types, which is more realistic to a practical implementation.

Also, we demonstrate through the “New Task Model Add Rate” for inter-participant task instances that the algorithm can generalize to new users when the tasks themselves are similar.

4.5 Interpreting Evaluation Plots

The general setup of our intra-user evaluation starts with a blank model being trained on $q$ unique task type instances for $n$ instances each (e.g., $\{n = 3, q = 2\}$ could mean that Task Type 4 is trained for three instances then Task Type 7 is trained for three instances). These $q$ unique task types comprise the set of known tasks to this model. The unknown task types include the $(10 - q)$ remaining task types in our dataset.

Once the model is trained, the model is split into two identical versions. One version of the model receives a new example from the set of $q$ known task types, and the other version receives a new example from the set of $(10 - q)$ unknown task types. The final percent complete after these known or unknown task examples are processed is saved along with whether or not a new task model would be added to the model library (according to our Dynamic Time Warping threshold).

For each participant’s dataset, this process is completed thirty (30) times for each $(q, n)$ pair for $q \in \{1, 2, 3, 4\}$ and $n \in \{1, 2, 3, 4\}$. The final values plotted in the main body of this work are the 95% confidence intervals around the median percent complete estimate. Each point’s median and confidence are computed from 240 examples of this process (8 participants × 30 random task type choices).

For the intra-user “New Task Model Add Rate” plot, we show the one standard deviation confidence for how often a new task model was added for the average participant after the Known or Unknown task type was processed.

For the inter-user evaluation, we iterate through each task type and train a model using a single user. We then split that model into seven identical versions and process an example of the same task type performed by the seven other participants. The choice of participant to train the model on is random, the choice of order of the individual task examples for the training is random, and the choice of the iteration of the new participant’s task are all random. We perform this test for thirty (30) different randomized initializations.

We plot both the rate at which a new task model is added and the final percent complete estimate in the main body of this work. A low “New Task Model Add Rate” suggests that the algorithm is able to consider different users as performers of the same task type. A high percent complete estimate suggests that the algorithm is able to properly estimate the new user’s motions as one or more of its known state-transition paths.

4.6 Intra-user Evaluation

For a randomly selected participant, we initialize the algorithm with $q$ unique tasks trained with $n$ instances of the task. For instance, if $q = 2$ and $n = 2$, then the
Figure 11. The median final percent complete estimate (with 95% confidence bounds) is shown for a generic task that has been trained $n$ times. This method reaches an end-position ($p^* > 80\%$) particularly well for one-shot learning when the number of unique task types, $q$, is small.

Figure 12. When an unknown task type is encountered after training, this method can perform well in median by using error correcting to advance the robot when all task models have high errors.

Figure 13. Unknown tasks are consistently added as new task models while known tasks are updated, especially when many examples of the known task have been seen. As the number of unique known tasks increases ($q$), the rate that models add unknown tasks decreases, suggesting that models are less specific to task than to task context.

task initialization could be [Task 1-instance 2; T1-i6; T8-i4; T8-i5]. Then, a new instance of a known (e.g., T8-i2) or unknown (e.g., T6-i1) task is processed. The final percent complete (as this is where the robot and user would interact) is output for the new task instance as well as whether or not a new task has been added. For each $(q,n)$ pair, we perform 30 experiments on each participant’s data for a total of 240 experiments. Each experiment is performed on a random task set (i.e., for $q = 2$, thirty random pairs of task types are considered). We then calculate the median final percent complete from all experiments and calculate the 95% confidence interval. We consider this median final percent complete value as an estimate of this algorithm’s ability to perform on generic tasks.

As shown in Figure 11, the algorithm consistently reaches an end-position ($p^* > 80\%$) with 95% confidence particularly well when limited training has occurred, thereby showing the capability of RP-KNN to match individual STPs. However, as $q$ increases, the algorithm shows weakness in differentiating among many task error signals, and additional work is needed in differentiating unique tasks or including a task-consolidation method to focus on frequently performed task types. As shown in Figure 12, the median final percent complete value often surpasses the success threshold with 95% confidence. This capability is mostly due to treating the new unknown task type as a linear combination of the known task types. The remaining success is due to an heuristic error-correcting advancement of 0.5% per frame when all task error metrics are above a high threshold. When all error metrics are high, the probabilities cannot be trusted, and the robot should simply advance. The threshold for this heuristic is not typically surpassed until late into a task, so the heuristic does not add a significant amount of percent, but does improve the overall results.
Figure 14. Tasks trained on one user and tested on another are mostly able to reach an end-position ($p^* > 80\%$), showing good generalizability across users. However, not all users complete tasks the same way, so especially for more complex tasks, personalized tasks models are generated for the new user.

As shown in Figure 13, as $q$ increases, the rate that new task models are added for new unknown tasks decreases, indicating that this method generalizes some unique tasks into the same context, which helps with online computation speed. Also, when known tasks are seen, new task models are added at a less than 10% rate which decreases as the model sees more examples of the known task due to the averaging effect of the task update process.

4.7 Inter-user Evaluation

For a randomly selected participant, we train ten task models with ten iterations of a single task type. Then, we choose a random iteration of each task of each other user to test on the associated task model. Figure 14 shows the averaged final percent complete values as the end of the online task processing as well as the rate at which a new task is added for each unique task.

In general, each task is generalizable across users in that the end-position ($p^* > 80\%$) is attained. However, the algorithm still determines that new task models should be created when tasks are more complex and can be done in different ways, thereby showing strong adaptability to individual users despite having pre-trained task models.

Figure 15. RP-KNN shows consistently lower final error metrics than a non-probabilistic KNN implementation in median with 95% confidence. Thus, the state transition path is followed better and the percent complete estimate is more accurate when using RP-KNN.

5 Rayleigh Probabilistic KNN Evaluation

In order to evaluate the effectiveness of RP-KNN, we consider how well a state transition path of a known task is followed relative to a more simple KNN implementation. The error metric (see Section 3.2.2) is directly related to a task’s deviation from known task models. Thus, we can look at the smallest error metric value among the known task models. We follow the same procedure and same random seed as the intra-user evaluation (Section 4.6) for both an RP-KNN procedure, and a simple KNN procedure. Each method uses the same $k$ nearest-neighbors values for each ($q$, $n$) pair. We plot the results in Figure 15.

RP-KNN performs consistently better than the non-probabilistic KNN for all update cases. KNN is quick to jump states in an incorrect order due to the limited amount of data for making decisions. Thus, the state-transition path is poorly followed in general.

6 User Study

We use a Kinova Mico 6-DOF\(^1\) robot arm for our implementation in a user study that seeks to determine subjective and objective measures for how

\(^1\)http://www.kinovarobotics.com/service-robotics/products/robot-arms/
In order to process an envelope, participants (1) Collect an envelope from the robot, (2) Solve the equation located beneath the envelope flap, and (3) Perform the necessary steps to process the classified mail type.

We seek to understand the robot’s success in adapting to a user’s changing tasks along with the user’s perception of the robot’s adaptation.

For the remainder of this paper, we denote our algorithm as JVA and the state-of-the-art algorithm from Huang et al. (2015) as CMA.

6.1 Hypotheses

6.1.1 H1 When the user performs the same task repeatedly, JVA will perform objectively and subjectively similar to CMA specifically designed for the task.

6.1.2 H2 When the user task varies from object to object, JVA will adapt objectively and subjectively better than CMA designed for only one of the tasks.

6.2 Experimental Setup

We base our implementation on a mail sorting scenario. The user is working with the robot to sort parcels of mail that contain coupons sent in from the public. The user must determine the validity of the coupon. We generalize this task by having the user solve a seven-, eight-, or nine-digit equation (e.g., \( x = 8 + 2 \times 3 - 6 \div 2 - 7 \times 2 + 4 \times 1 \)) using order of operations with four operators: \( \{\div, \times, +, -\} \). Any \( x = 0 \) is considered Junk and does not contain a coupon. Any \( x < 0 \) is considered Normal and contains a coupon will give the sender a fixed return, similar to a rebate coupon. Any \( x > 0 \) is considered Priority and must be immediately processed to determine the return, similar to how a raffle ticket would need to be verified before a prize were given. The three mail types are then processed as follows (also see Figure 16):

- **Junk**: User immediately places the piece of mail in a box (shredder) located at knee height.
- **Normal**: User drops the envelope in a box labeled “-” on the counter.
- **Priority**: User stamps envelope in specified location before placing it in a specific spot in a box labeled “+” on the counter.

This setup naturally amends itself to an inconsistent user task that would not be immediately successful to the current state-of-the-art adaptive method by Huang et al. (2015). We found that by changing both the number of digits and the number of particular operators in the math problems, we could change the amount of time a particular task type should take on average. Seven-digit math problems with \( \{2\times, 2+, 2\div\} \) operators took the authors 9.02 ± 1.14 seconds. Eight-digit math problems with \( \{1\div, 2\times, 2+, 2\div\} \) operators took the authors 12.17 ± 2.25 seconds. Nine-digit math problems with \( \{2\div, 2\times, 2+, 2\div\} \) operators took the authors 14.42 ± 1.69 seconds. An additional digit added approximately 2.7 seconds to the solution time. We assigned these types of math problems to Junk, Normal, and Priority mail types, respectively. Because of the likely non-math background of the typical participant, we assumed that the average calculation time for each task type would be somewhat longer. We thus expected Junk to take 10 seconds, Normal to take 15 seconds, and Priority to take 20 seconds.
The human-subjects study approved by our Institutional Review Board was designed as a within-user 2x2 with variables as the number of unique task types and the algorithm choice. The number of unique task types varies between “one” and “three”, where the “one” case has the participant perform only the Normal task type for all iterations and the “three” case has the participant perform a randomized set of all three types. We refer to these task counts as Single and Multi, respectively. The algorithm choices are JVA and CMA. The study protocol went as follows:

1. Get informed consent.
2. Explain task types and participant’s role.
3. Train user to advance the robot to subsequent envelopes.
4. Train user to perform various task types.
5. Allow user to refamiliarize with math principles by solving five example problems.
6. Complete randomized experiment number, then allow user to complete questionnaire and answer open questions. Repeat this step until all experiments are finished.
7. Provide payment for time at a rate of five dollars per 30 minute session for two sessions.

We had 24 individuals (13 female, 11 male) ages 19 to 30 (μ = 21.9, σ = 3.1) complete the four experiments in a computer-randomized order. All of one participant’s data was rejected according to criteria detailed in Section 6.5. Each experiment requires the participant to perform eight full handover processes with the robot. On a scale of 1=“Not at all” to 7=“Very familiar,” individuals reported a low familiarity with robots (μ = 2.44, σ = 1.29). On a scale from 1=“Weak” to 7=“Strong,” individuals reported an average strength at mental math (μ = 4.04, σ = 1.74).

6.3 Implementation Considerations

Our algorithm, JVA, is trained on one iteration of each task type, and the state-of-the-art model, CMA, expects only the Normal task type. The practical implementation has a well-defined handover location and pose that participants are asked to assume when they are ready for the next object. Thus, a JVA ready pose is defined ahead of time that acts similarly to the CMA handover pose. The ready pose and handover pose both tell the robot that the user is ready for the next object (i.e., the robot should move to the handover position).

Unlike during the algorithm evaluation, we do not apply an error-correcting heuristic to slowly advance the robot when all task model errors are high. Instead, we look for the user to be in a ready position to mark the end of a task. If the robot is not yet at the 100% progress position, then it will move to that position when the user enters the ready state. Thus, the user will not wait excessively long if the algorithm believes the task is not near complete. The error-correction heuristic was necessary in the evaluation to mimic the case where the robot can do nothing but guess at when the task will end. In the practical implementation, we do not need to guess.

JVA-single is pre-trained on one example of only the Normal task-type. JVA-multi is pre-trained on one example of each of the Normal, Priority, and Junk task-types. The examples were all performed by one of the authors. We decided to pre-train these models because of difficulties in quickly teaching participants to properly train the model. Participants in our pilot studies showed high variability in the amount of time the process took, which made scheduling participants difficult. Each task is also relatively simple and does not have room for extremely different motions throughout the task. Also, we demonstrated in Section 4.7 that models created on one user perform sufficiently well on another user performing the same tasks. The only part of the task that is participant dependent is the equation solving segment. We used the expectation that the typical participant would take 10, 15, and 20 seconds to solve Junk, Normal, and Priority mail types, respectively. The author pretended to calculate each equation for these amounts of times for each of the single task examples on which JVA was trained.
Our implementation does not include the task updating portion of the algorithm. We do this for two reasons. First, this method allows for better comparison with the state-of-the-art method which is pre-trained. By pre-training our own models, but with only a single instance, we demonstrate the ease with which new task models can be added. This ease is contrary to the extensive modeling effort required of the state-of-the-art. Second, due to the long nature of these tasks (≈ 25 seconds) relative to those that the algorithm was evaluated on (≈ 4 seconds), spectral clustering and DTW are not a quick enough methods for an update-after-every-task scheme. However, we could have allowed the algorithm to update in the background, but the small number of handovers per experiment (eight) renders the updates only mildly useful. If each experiment required a full workday’s worth of handovers, then this type of implementation would be more reasonable.

Our pilot studies showed us that JVA’s attempts to match the user’s task progress exactly were non-ideal. Participants suggested that the robot should favor being in the handover position too early rather than falling significantly behind the user’s pace, thereby requiring the user to wait. In order to implement a simple version of this behavior we added a scalar multiplier to the user’s percent complete estimate:

\[ p^* = 1.3p \]

where \( p \) is the current percent complete estimate of the user’s progress, and \( p^* \) is the updated percent complete estimate sent to the robot controller. By adding this linear adjustment, we make the assumption that the robot should move to the handover position if the participant is approximately 75% finished with the task. This assumption is reasonable in our application because this last 25% of a task mostly includes placing objects in their final positions and moving into position to collect the next object.

6.4 Differences Between Algorithm Implementations

Original code from the Huang et al. (2015) implementation of CMA was adapted to our envelope passing scenario. This implementation included a force-sensing feature that allowed for users to pull the envelope directly from the robot’s gripper. Because of the small size of envelopes, the threshold for sensing this force needed to be relatively low, but not so low that the robot could be triggered by sensor noise.

This same Force method of interaction could not be implemented in JVA because of our choice of overarching library for controlling the Mico arm. CMA was implemented directly using the Kinova API. In order to create a similar experience, we chose to use a Ready-Pose method in which the participant would enter a pre-defined pose that calls the robot to come to the handover position and open its gripper regardless of its current position. If the robot were already in the handover position, then it would simply open its gripper. The Force method is not substantially different from the Ready-Pose method because the same general user pose causes both algorithms to send the robot to the final handover position.

Participants were taught how to interact with each method of interaction during the training phase of the study. Before beginning the first experiment, we stressed that the method of interaction was not the driving difference between any algorithms in the four experiments. The participants were told that the Ready-Pose and Force methods should only be thought of as means to advance the robot to collect the next object. Unfortunately, several participants made comments during the open questioning that these methods were a defining difference they felt between the algorithms. For example, Participant 17 explained, “I like the ready-react method. It makes me feel like I have more control over the robot, and it makes it feel as if we are working more as a team ... and in terms of the actual handover itself, I like the force method better.” Most participants saw very little difference between algorithms though, so these comments could
be due to participants not noticing other differences. See Section 6.7.

The two algorithms also had slightly different trajectories. Again because of the use of different Mico control libraries, joint angles at different waypoints of CMA and JVA were not the same values at the same absolute positions. Still, we attempted to use similar trajectories. Further, CMA and JVA also move through their trajectories differently. CMA takes a wait-then-advance approach that only moves to its next major position once the user has reached a certain state. JVA advances to the current percent complete estimate of the user in a linear manner. One issue we had with this approach is that robot speed was not controllable, so the robot could behave in a jerky manner if the updated percent complete estimate were only slightly higher than the previous position. This jerkiness should be removed and a smooth advancement should be implemented in future iterations. A few participants such as Participant 11 noticed that “It (JVA-multi) was a little jerky, the movements weren’t as fluid as (in previous experiments).” However, the movements mostly went unnoticed to other participants. For example, Participant 12 said, “I don’t really notice the sounds at all when im doing the math or sorting.”

Lastly, the speeds of each algorithm were nearly but not exactly the same because velocity control was not implemented on a per-movement basis. However, the maximum allowable speeds of each algorithm were set such that neither algorithm had an advantage in delays due only to movement speed. These speeds were set to be relatively slow so that differences between algorithms could be more easily noticed by participants.

2. Robot fails to process an envelope (e.g., fails to pick up envelope but continues as if it was successful) more than one in the experiment.

3. Incorrect algorithm used for a particular experiment (e.g., JVA-multi mistakenly used for a JVA-single experiment).

4. Previous experiments for an individual participant were rejected according to above criteria (e.g., if the participant’s second experiment is rejected, then the third and fourth experiments will also be rejected).

Criterion number four is applied because subjective data for future experiments is likely substantially biased by the errors of previous experiments.

We also rejected individual handovers under a specific condition: for JVA, we reject waiting data if the task takes the participant longer than three times the average participant’s full task time (reject if greater than $3 \times 27.75 = 83.25$ seconds). This exclusion is reasonable because this implementation assumes a typical task time for a user. These outliers can be thought of as times when the participant had significant struggles solving an equation. In a full implementation, the algorithm would classify this sort of outlier as it’s own task type.

These criteria resulted in rejecting six individual JVA handovers (1.6% of all JVA handovers), two participants’ final experiments (1 x CMA-single, 1 x CMA-multi), and one participant’s complete dataset. In full, we had 23 experiment examples of JVA-single and JVA-multi, and 22 experiment examples CMA-single and CMA-multi.

### 6.5 Data Exclusion Criteria

We rejected whole experiments of data for three reasons:

1. User performs incorrect actions (e.g., processes Normal mail as Priority) more than one time in the experiment.

6.6 Objective Evaluation

The ideal handover process would have the robot and user waiting a minimal amount of time for their partner to finish the current task. If the robot waits in the handover position for the user, then any time spent in that position could hypothetically be applied to non-user-dependent subtasks. If the user waits in the handover position for the robot, then the task is not as efficient with the user’s time. Our goal in implementing
These adaptive algorithms is to minimize this overall waiting time.

We define the “overall waiting time” of an individual handover as the amount of time either party is in the handover position waiting for the other party to complete the handover. Each handover will include either “robot waiting time” (i.e., the robot waits in handover position for the user to collect the object) or “user waiting time” (i.e., the user waits for the robot to bring the object to the handover position), but not both. See Figure 17.

We most care about how well JVA performs relative to CMA for each task count (single or multi). Figure 18 shows the various average overall, robot, and user waiting times for the entire group of participants.

For both single and multi task scenarios, we performed t-tests on the Overall, Robot, and User waiting times for each algorithm while controlling for each participant’s average task completion time during the particular experiment. These average task completion times $\bar{t}$ were calculated as follows:

$$\bar{t} = \frac{T_{full} - W_{user}}{N}$$

where $N$ is the number of handovers completed (typically eight, but sometimes fewer in accordance with Section 6.5), $W_{user}$ is the total amount of time the user waited for the robot during the experiment, and $T$ is the total time from when the participant grabs the first envelope to when the participant grabs the last envelope (the participant does not process the final envelope). We remove the user waiting time because it is not indicative of how quickly the participant could hypothetically complete each task. Tests of the two a priori hypotheses were conducted using Bonferroni adjusted alpha levels of 0.025 per test (0.05/2). For the single-task scenario, only the user waiting time comparison resulted in rejecting the null hypothesis ($p = 0.0029$). For the multi-task scenario, both the overall waiting time ($p = 0.0006$) and user waiting time ($p < 0.0001$) showed significant differences.

For this group of participants, both JVA and CMA tended to require the user to wait for the robot to finish the handover. User waiting occurred in 81.5% of tasks for CMA-single and 58.6% of tasks for JVA-single. The nearly 50% robot-to-user waiting split for JVA reflects the idea that the robot would perform best for the average equation solvers in our set of participants. Ideally the robot would be trained to the individual’s pace and not this expected pace.
However, JVA required the user to wait for consistently shorter times. This detail is partially due to the progress matching feature of JVA’s control. As the task continues, the robot is further along the handover trajectory compared to CMA, which typically waits until the user finishes placing the envelope in the bin to start moving from its mid-trajectory position.

User waiting occurred in 58.6% of tasks for CMA-multi and 77.8% of tasks for JVA-multi. For CMA, the single task percentage of user waiting scenarios decreased because of the addition of the Priority tasks. The robot advanced to the handover position early in 50 of 53 Priority tasks (94.3%). This “too soon” motion (see Figure 19) was triggered by the participant moving to mark the envelope as priority, which resembles the algorithms handover state. CMA-multi saw the user wait in both Normal and Junk task types wait in 81% and 84% of handovers, respectively. These rates are consistent with the CMA-single rates.

For JVA-multi, the algorithm is much more cautious advancing because of uncertainty caused by its knowledge of all three task types. The robot showed restraint in moving to the handover position unless it was sure the user was ready or the task had been, in its opinion, excessively long and possibly unknown. Overall, this method resulted in the user waiting for 82% of Normal task types, 50% of Priority task types, and 98% of Junk task types. The algorithm performed best on the Priority task types which typically last longer than the others, which suggests that our task differentiation heavily favors longer tasks and improvements could be applied to this portion of the algorithm.

CMA-multi performed worse than JVA-multi in part due to user waiting times during the Junk task which skipped CMA’s expected “place” state. The lack of this state made it so the robot’s movements to the handover position started while the participant was already near the ready position rather than across the task space placing the envelope in its bin. JVA-multi out performed CMA-multi by an average of 3.8 seconds on user waiting time for the Junk task.

One additional benefit of JVA over CMA is that CMA can mistakenly advance the robot to the handover position if the participant does not stand in the expected poses that the task was initialized to follow. Two participants in particular caused CMA to fail in unexpected ways, which demonstrated the algorithms dependence on expected poses. One participant consistently stood with his torso at a slight angle towards the robot though his feet were pointed toward the sensor as he was instructed during the task training phase. CMA would classify this pose as the “handover” pose and the robot would advance to the final position though the participant had just started the task. Another participant solved the math problems with his forearms perpendicular to his body, which CMA often classified as the “idle” state which is meant to occur only when the participant is waiting on the robot to give the next object. So, the robot would pre-maturely move to the handover position in this case as well.

Our results support our first hypothesis, H1, in that JVA-single performs objectively similarly to the state-of-the-art CMA-single. In terms of overall waiting time in the system, our algorithm does not perform significantly worse even though the more modeling-intensive CMA was explicitly trained on the single task type. Interestingly, our implementation did result in a significant decrease in user waiting time.

Our results also support our second hypothesis, H2, in that we see significant objective improvement in both overall waiting time and user waiting time when the task can include multiple task types. The modeling effort required to acquire models for all three of the example task types is substantially less than that required to define the states of, and collect clustering data for, even a single explicit task model for a CMA implementation.

6.7 Subjective Evaluation

Participants filled out a questionnaire after completing each of the four experiments with the robot. Questions were grouped into three categories and their scores...
Figure 19. A Priority task type is completed by two participants at a similar rate. JVA (top) is not immediately triggered to go to the handover position by the participant’s motion toward the priority marking station as in CMA (bottom). Instead, the robot knows to wait until the user is closer to the task’s end to continue its movement.

Figure 20. The single task scenario saw JVA perform significantly better than CMA ($p = 0.0236$), but other measures suggested the algorithms performed equivalently.

were averaged to measure the robot’s performance in each category. The categories include:

- **Confidence** (five questions, Cronbach’s $\alpha = 0.816$):
  How confident the user was in the robot performing its required task. Scaled from “1=Low Confidence” to “7=High Confidence”.

- **Stress** (twelve questions, $\alpha = 0.759$):
  How the user felt about task load, pressure, stress, annoyance, and feeling of control. Scaled from “1=Low Stress” to “7=High Stress”.

- **Teamwork** (thirteen questions, $\alpha = 0.820$):
  How the user felt the human-robot team performed in terms of efficiency, enjoyability, and general collaboration. Scaled from “1=Poor Teamwork” to “7=Strong Teamwork”.

For both single and multi task scenarios, we performed t-tests for Confidence, Stress, and Teamwork measures of each algorithm while controlling for each participant’s average task completion time during the particular experiment. Tests of the two a priori hypotheses were conducted using Bonferroni adjusted alpha levels of 0.025 per test (0.05/2). For the single-task scenario, only the Teamwork measure resulted in rejecting the null hypothesis ($p = 0.0236$). For the multi-task scenario, no subjective measures showed significant differences.

6.7.1 Single-Task Scenario

For the single-task scenario, our subjective results (see Figure 20) support $H1$ in that participants scored JVA as subjectively equivalent or better (in the case of Teamwork) even though JVA required much less manual modeling effort.

Based on open questions, participants seemed reasonably split about whether they preferred JVA-single or CMA-single. Many participants expressed positive thoughts about JVA-single:

- “I felt good. I didn’t feel stressed or pressured or anything.” (Participant 13)
- “I felt like I had less control in the first two (CMA-multi and CMA-single), but in the last two (JVA-multi and JVA-single) I felt like I had more control.” (Participant 3)
- “This one (JVA-single) for whatever reason, felt like a more friendly collaboration. But I guess it wasn’t super different from either of the previous two (CMA-single and JVA-multi).” (Participant 5)
The multi task scenario saw CMA and JVA as subjectively equivalent for each of our measures.

- “The math was hard, but the robot part was interesting. It’s really cool it can detect my movements, but I did feel pressured like if it was grabbing something and I was still doing the math, but the part that was hardest for me was the math. When it was already waiting I felt like I was being too slow, but it was a good feeling when it was still working because it meant I was fast.” (Participant 22)

Many others seemed to prefer CMA-single for similar reasons:

- “There was maybe one time when I was waiting for it, when it was still doing its thing. I felt like it was more collaborative than it was for the first round (JVA-single). A little more so (I was in control of the pace). Not completely but definitely more so.” (Participant 22)
- “It seemed like you (the user) were controlling it. If I took a longer time to calculate something, the robot took its time.” (Participant 4)
- “It was kind of going at my pace rather than a pace that’s faster than mine.” (Participant 11)

This split committee suggests that each algorithm has benefits that are important to different users.

6.7.2 Multi-Task Scenario

For the multi-task scenario, our subjective results (see Figure 21) do not support H2 in that our algorithm does not subjectively outperform the state-of-the-art. We found that users were typically inattentive to differences in the robot’s behavior for different task types, and instead, they focused on the robot only during the actual handover when they were completely finished with their task.

Participants generally did not notice any differences in robot behavior in the multi-task scenarios for both JVA and CMA. Most participants were focused on solving the math problems and were not distracted by robot movements and noises in the background. Participant 11 explained, “I kind of zoned out on the robot. I was literally focused on the math.” Participant 14 agreed, “I didn’t really notice it during the task, I was more focused on the math.”

Some participants did notice that CMA-multi had a unique behavior on Priority tasks. The priority task was designed to trigger the robot to prematurely move to the handover position during only CMA, showing that the algorithm was not generalizeable to all handover tasks. Participant 18 was one of the few participants to pick up on this intentional fault of the algorithm: “When I put the (post-it notes on the priority) envelopes it would move. On the (normals) it didn’t. I don’t remember about the junk. I preferred its reaction during the (normals). Because we have to remember to place it in the box, the robot moving during the priority writing was a little distracting.”

Participant 6 also noticed the early movements: “There was that one time that it got screwed up... it thought I was done with the task, but I was just writing the priority thing.”

However, more participants found this movement of CMA-multi to be a feature that improved the collaboration. For example, Participant 5 had a good experience: “It felt like it was working to my pace. It felt like it figured me out. Whenever it figured out what I was going to do, it would go for the next one. As soon as I moved to priority it ... there was definitely a feeling like me and the robot were on the same pace during this one.” Many participants noticed that CMA had very deliberate motions that correlated well with their own motions. This belief coincides with CMA-multi having slightly better, though not significant, improvements over JVA-multi in each of the subjective measures.
even though \textit{JVA-multi} performed significantly better in objective measures.

Some participants also thought \textit{JVA-multi} was attuned to their own pace:

- “I liked that one a lot better (than \textit{CMA-single} or \textit{CMA-multi}) ... I felt like it was already moving towards me as I was walking back towards it.” (Participant 3)
- “I still think it’s doing good. It knows where I’m at. I didn’t feel like there was a difference between tasks.” (Participant 7)
- “For the most part, 80\% it was right on point of hey he’s about to get done, I should be on my way for the hand off. There were very few instances where the robot was lagging behind.” (Participant 11)

Some participants also noted the temporal expectation they felt the robot had of them. Participant 6 explained it as a benefit: “It was pretty similar to last time (\textit{CMA-multi}), I would say the one time it was particularly helpful was when I had spent a long time on a math equation it was already ready to hand me the next piece.” Participant 4, however, thought the timing added extra pressure: “The priority method took longer than the (junk) or (normal). I wanted to do it quicker because the robot was already further (along). (I) felt like I was going the same pace as the robot during trash and negative, but felt behind on the priority method.”

6.7.3 General Subjective Comments Overall, the differences between algorithms with regard to the robot’s movement during the task were mostly indistinguishable. Participant 18 noted, “I wasn’t paying much attention to the robot besides getting the envelope from it.” The most important factor for participant enjoyment seemed to be that the robot did not make them wait. Some participants did not find any benefit to an adaptive strategy:

- “I preferred it when it was already waiting for me for awhile.” (Participant 8)
- “My biggest gripe was that when I was done, it wasn’t ready. As soon as I grab the envelope I want it to be in that handoff position as soon as possible for the robot. It should be waiting on me 100\% of the time.” (Participant 11)
- “It should just take the envelope and give you the next one no matter what you are doing.” (Participant 10)

We believe these comments are mostly due to the relatively slow speed of the robot and not the adaptive nature of the algorithms. The original work by Huang et al. (2015) demonstrated the user experience improvement of the adaptive strategy over the proactive strategy that these participants desire. This prior work used significantly faster movements than those used here. When prompted about whether a faster robot would improve their experience, Participant 6 said, “In general, I felt like I could have finished the task faster without the robot except for the third one (\textit{CMA-single}). If the robot were faster that probably would be different.”

An additional comment shared by several participants was that they would like the robot to change its trajectory to match where they put their hand. After using \textit{JVA-multi}, Participant 8 suggested, “I would like it to predict the reach of my arm more over time so I wouldn’t have to make adjusting movements.”

7 Discussion and Future Work

In this paper, we have shown that our method of task modeling with online progress matching provides strong results especially when the number of known tasks is low. We have also demonstrated the algorithm’s generalizability to variable tasks through its ability to differentiate between known and unknown tasks, even across different users.

In comparison to the state of the art algorithms in this space, our algorithm allows for similar performance in both objective and subjective measures. Our algorithm benefits by an approximation of the current percent complete and one-shot learning of new tasks. In practice, the knowledge of current task progress allows
a robot to complete such tasks during the early stages of a task, while still benefiting the user’s experience with the robot when compared to a non-adaptive strategy. For example, in a cooking scenario, the robot subtask could be to collect ingredients required later in the recipe while the user stirs a stew. When the user is nearly finished stirring, the robot could begin the handover process of the next ingredient in the recipe. The handover could be useful in this case to assure the user does not select the wrong ingredient to add to the stew next. Our algorithm also allows for tasks to be learned in a one-shot manner, so the stew making motions would need to be performed a single time to collect the necessary information to build the human-robot team.

More work is necessary to allow for vastly different task timings within the same modeling setup. We needed to alter parameters in the PID task differentiation (see Section 3.2.2) to allow the algorithm to function properly in the robot implementation tasks, which were much longer than the algorithm evaluation tasks (25 seconds vs. 4 seconds). DTW was shown to be an effective method for task differentiation, but in combination with spectral clustering, proved to be most effective for shorter task lengths. Longer task lengths required too much computational time to efficiently (a) determine if new task models should be created and (b) perform spectral clustering to calculate new task models. We found sufficient, but likely non-optimal, PID parameters to perform our task differentiation. Advanced methods for automatically determining PID parameters based on the lengths of known tasks would be ideal.

Acknowledgements

National Science Foundation Research Traineeship Program LUCID (award 1545481) and National Science Foundation Research Award 1149970 supported this research. We would like to thank Chien-Ming Huang, Daniel Rakita, Kwang-Sung Jun, Marshall Mueller, Majid Askari, and Amanda Sarsha for their help with conducting the research.

References


