Optimized Algorithms for Prediction within Robotic Tele-Operative Interfaces*

Authors:
Rodney A. Martin†
Kevin R. Wheeler‡
Vytas SunSpiral§
Mark B. Allan¶

ABSTRACT
Robonaut, the humanoid robot developed at the Dexterous Robotics Laboratory at NASA Johnson Space Center serves as a testbed for human-robot collaboration research and development efforts. One of the primary efforts investigates how adjustable autonomy can provide for a safe and more effective completion of manipulation-based tasks. A predictive algorithm developed in previous work [9] was deployed as part of a software interface that can be used for long-distance tele-operation. In this paper we provide the details of this algorithm, how to improve upon the methods via optimization, and also present viable alternatives to the original algorithmic approach. We show that all of the algorithms presented can be optimized to meet the specifications of the metrics shown as being useful for measuring the performance of the predictive methods. Judicious feature selection also plays a significant role in the conclusions drawn.

Categories and Subject Descriptors
I.2.6 [**Artificial Intelligence**]: Learning—*parameter learning*; I.2.9 [**Artificial Intelligence**]: Robotics—*operator interfaces*; G.3 [**Mathematics of Computing**]: Probability and Statistics—*Markov processes*; I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis

General Terms
Algorithms, Performance

Keywords
optimization, prediction, practical implementation

1. INTRODUCTION
Humanoid robotic tele-operation has been shown to be of interest for space-related applications [1, 3]. NASA's Robonaut is clearly an excellent platform for performing human-robot collaboration research, and serves as a testbed for developing practical capabilities and interfaces. Potentially, a myriad of space-based construction and maintenance tasks can be performed remotely by Robonaut. Manual teleoperation allows for full control over the robot's trajectory throughout execution of a task. This is very important from the standpoint of safety in order for errant execution to be terminated as soon as possible to prevent damage to expensive equipment, or injury to personnel. However, teleoperation often incurs much more dedicated time and effort on the part of the human operator, with the task taking 3 to 4 times longer on average than if performed at normal human speeds. One reason for the extended task time is due to the fact the robot is being operated over a time delay, and it takes time to verify that the commands sent to the robot are the ones actually being executed. This "bump and wait" approach is tedious, and adds to the time to perform a task. Furthermore, for safety considerations, Robonaut's movement is rate limited, so that any movements made by the operator must match these rate limitations, naturally causing a slower execution.

In contrast, we may consider fully automating the operation of the robot, obviating the need for tele-operation, potentially decreasing the task burden and time. However, this would require building up a dictionary of commands based hierarchically upon waypoints to complete a simple task. The scalability of building such an interface is likely to present a natural practical limitation, not to mention that

*(Produces the permission block, copyright information and page numbering). For use with ACM_PROC_ARTICLE-SP.CLS V2.6SP. Supported by ACM.

†NASA Ames Research Center, M/S:269-1 Moffett Field, CA 94035-1000 email:rmartin@email.arc.nasa.gov

‡NASA Ames Research Center, M/S:269-1 Moffett Field, CA 94035-1000 email:kwheeler@email.arc.nasa.gov

§Formerly published as Thomas Willeke, QSS Group, Inc. NASA Ames Research Center, M/S:269-2 Moffett Field, CA 94035-1000 email: vytas@email.arc.nasa.gov

¶QSS Group, Inc. NASA Ames Research Center, M/S:269-2 Moffett Field, CA 94035-1000
email: mallan@email.arc.nasa.gov

Human Robot Interaction '06 Salt Lake City, Utah USA

operating in a fully autonomous mode prevents human intervention that is required for safety considerations. As a trade-off to operating in fully manual or fully autonomous mode, we take advantage of the sliding scale of adjustable autonomy. By allowing for both a manual tele-operation phase and an autonomous phase of operation, we can free the operator to perform other tasks, and mitigate their task burden, while at the same time retaining the ability to adhere to inherent safety constraints.

As part of the precursor to the work presented here, we previously developed predictive techniques and implemented them algorithmically for use during the manual tele-operation phase [9]. The prediction algorithm is run through a simulation as shown in Fig. 1, prior to the commands being sent to the actual robot as shown in Fig. 2.

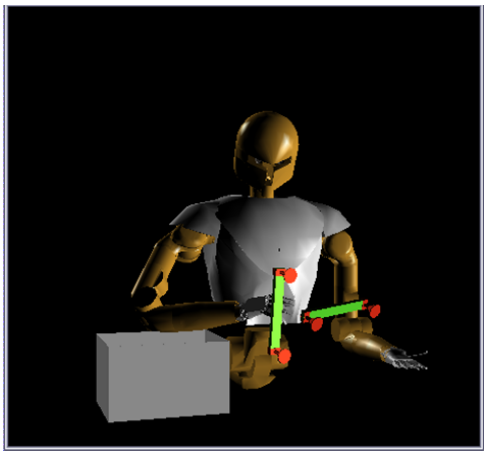

Figure 1: Robonaut Experimental Task Setup in Simulation

These prediction algorithms have previously been designed in a heuristic manner, without explicit optimization of the desired objectives. Those objectives are as follows:

- Probability of False Alarm – 0 %
- Probability of Missed Detection – ALARA (As low as reasonably achievable)

Figure 2: Robonaut Actual Experimental Task Setup

- Average time to prediction for correctly classified trials – ALARA

Other approaches will be examined as alternatives to prediction algorithms used in prior work [9]. Each of these techniques have their own nuances and pose unique challenges, yet they all share a common feature. In each method, there are sufficient available free parameters that can be optimized to achieve the listed objectives. As such, our goal is to improve upon previously generated results in order to meet the performance specifications listed above that define how well we are able to utilize the sliding scale of adjustable autonomy. In order to make the best use of the tele-operator's time, we would like to be able to accurately predict the task being performed as early as possible into its execution, prior to initiation of an autonomous action.

The main requirements are based upon error statistics that are normally used in decision theory, the probability of false alarm and missed detection. Normally there is an optimal tradeoff between false alarms and missed detections that can be achieved, where improvement in one metric can be achieved at the expense of impeding the other. There has been much work in the statistical literature on this topic, and several references are available [8]. However, for pragmatic purposes, we are also as interested in achieving the best tradeoff between false alarms and missed detections as we are of minimizing the average time to prediction for correctly classified trials.

For our experiment, an example of a false alarm is when the prediction algorithm indicates that the tele-operator is reaching for the vertical handrail, yet the "ground truth" is that the tele-operator is reaching for the horizontal one. A missed detection is the case in which the prediction algorithm fails to recognize that any handrail is being reached for at all. Minimizing the average time to prediction for correctly classified trials is important for the purposes of maximizing tele-operator "free time." That is, when the tele-operator grasps the object, this indicates the natural end of the window of useful time for prediction. For our application, false alarms are much more critical than missed detections, due to safety considerations. An autonomous grasp executed erroneously may potentially place the robot or astronauts working alongside the robot in a hazardous situation.

In previous work [9], we have trained and implemented Hidden Markov Models (HMMs) for prediction of operator intent using available data. The experimental setup used in this work will be the same as used in the previous work. The essence of the task is that the operator is reaching for a vertically or horizontally oriented handrail mounted on a vertical wall, prior to placing it in a box. Results have been shown to be very good in practice, using learning techniques more sophisticated than originally proposed for Robonaut [3]. Reports of using the same HMM methodology for gesture recognition are available in other studies as well [5]. However, it is possible to improve upon the results in [9] in order to more closely achieve the performance requirements stated above.

Both off-line (static) and on-line/real-time (dynamic) vali-

dation is performed by recalling on the trained models. "Recall" is a term that often refers to the use of the Viterbi algorithm [2], and we have used similar techniques for other applications in the past [10]. The Viterbi algorithm relies upon having a finite sequence buffer of data to be tested on. Off-line, or static validation refers to performing recall on a validation set, using the same sequence buffer segmentation that was used during training, to gain intuition about real-time performance. During real-time recall of the models, HMMs trained on both types of tasks (reaching for horizontal or reaching for vertical handrails) are arbitrated based upon an algorithm to determine the "winning model," or which model best describes the streaming data. Furthermore, a completely different set of data is validated during both types of recall than is used during training. This is performed by taking all of the data spanning multiple training sessions and training days, randomizing and partitioning the complete data set into mutually exclusive training and validation sets. Both the nature of how the Viterbi algorithm is implemented and the algorithmic details will be provided in the subsequent section.

The new methods which may allow us to achieve the performance specifications stated above are by modifying how we perform recall, using judicious feature selection, and optimizing both static model training and dynamic real-time recall parameters. In previous work, we used subsets of position (x-y-z) and orientation (roll-pitch-yaw) feature vectors to train and recall on the models. The feature vector acts as a template to form observation data sequences used both to train and recall the models. Here, we propose to either replace or augment these feature vectors with distance data, which provides the Euclidean distance to the object of interest being reached for. This will potentially add to the discriminatory ability of recall on the models.

Additional recall algorithms will also be examined, as alternatives to the Viterbi algorithm. One such algorithm is similarly based on a finite sequence buffer, however, its classification is based upon posterior probabilities. Regardless of the algorithmic method being implemented, there is a unique method to parameterize this sequence buffer and how to find an optimal design point for static model training. Finally, we will use the recall method of posterior probabilities without a sequence buffer, which has unique advantages and disadvantages. Because there is no buffer, optimization of this algorithm over the performance specifications of interest can only be performed over real-time recall parameters (not the model training parameters). We will perform dynamic optimization of the other competing methods as well, where applicable.

2. METHODOLOGY

2.1 Hidden Markov Model Implementation

Practical implementation of Hidden Markov Models has been covered at depth in the literature [7]. Here we aim to detail the most relevant facts pertaining to the nuances that we will exploit and have been presented in previous work [9]. Mathematical notation will be borrowed from Jordan [4]. We have chosen to implement a tied-mixture Hidden Markov Model with $M = 3$ states and $N = 6$ mixtures. Fig. 3 shows the graphical model topology and relevant parameters for this variant.

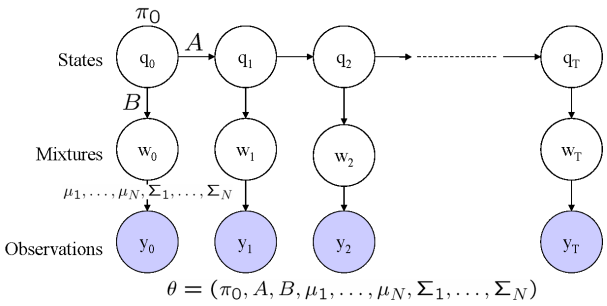

Figure 3: Tied Mixture Hidden Markov Model

where $t \in \{0, \ldots, T\}$ references the discrete-time steps within the sequence buffer, q_t : the state value at time t, w_t : the mixture value at time t, $y_t \in \mathbb{R}^n$: the observation vector at time t, and n : number of elements in the feature vector. The HMM parameters which are learned by Baum-Welch iteration, are grouped together as θ, and are defined as follows:

$$
\begin{aligned}
\text{Prior (initial) probability} & \\
\text{distribution} &: \pi_0 \\
\text{Transition probability matrix} &: A \\
\Rightarrow a_{ij} &= p(q_{t+1} = j | q_t = i) \\
\text{Mixture weights} &: B \\
\Rightarrow b_{ij} &= p(w_t = j | q_t = i) \\
\Rightarrow \pi_0(i) &= p(q_0 = i) \\
\text{Mean of Gaussian distribution} & \\
\text{for mixture j} &: \mu_j \\
\text{Covariance matrix of Gaussian} & \\
\text{distribution for mixture j} &: \Sigma_j
\end{aligned}
$$

Some other important probabilities with regards to the tied mixture HMM shown in Fig. 3 are as follows:

$$
\begin{aligned}
\text{Emission (output) probability} &: \\
p(y_t | w_t = j) &= \mathcal{N}(y_t; \mu_j, \Sigma_j) \\
\text{Gaussian mixture probability} &: \\
p(y_t | q_t = i) &= \sum_{j=1}^{N} b_{ij} \mathcal{N}(y_t; \mu_j, \Sigma_j)
\end{aligned}
$$

2.1.1 Viterbi-based recall

The Viterbi algorithm uses the idea of dynamic programming in a discrete-state context, and estimates the best state sequence described by the available data via the following mathematical formulation:

$$\delta_T(i) = \max_{q_0, \ldots, q_T} P(q_0, \ldots, q_T = i, y_0, \ldots, y_T | \theta)$$

Therefore, $\delta_T(i)$ is the probability that the state sequence

ends up in state i at final time T. The algorithm to solve this optimization problem is recursive in nature, and the termination step provides us with $\delta_T^* = \max_i \delta_T(i)$, the maximum probability over the possible state values, i, at the final time in the sequence buffer, T. The quantity $\log \delta_T^*$ is what we shall use as our primary indicator of the likelihood that the data being testing obeys the model being recalled on.

There are two models of interest, one trained on observation sequences recorded when reaching for the vertical handrail, and another trained when reaching for the horizontal handrail. We are interested in arbitrating between the likelihood of both models for a particular trial, whether it is horizontal or vertical. As such, a "confusion matrix" can be developed on static training data in order to gain intuition about the robustness of real-time recall. Errors in the confusion matrix are incremented when false alarms occur and placed on the off-diagonals of the matrix. An example of this is when we are reaching for the vertical handrail, but the recall arbitration indicates that it is the horizontal handrail.

To gain more clarity about the difference between static recall and dynamic real-time recall, refer to Fig. 4 , which shows an example of the y-yaw feature vector on the bottom 2 plots. Here, multiple validation trials are superimposed from a fixed time prior to the time that the handrail is grasped, or in some cases from the beginning of the trial until the time that the handrail is grasped. The most clarifying feature of this figure is that the "training" segments are demarcated as starting with circles ($\circ$), and ending with crosses ($\times$). The sequences to be recalled in real-time are shown as solid lines for the y variable, and dotted lines for the yaw variable. Because the data displayed in these plots are validation trials, the training demarcation is for illustrative purposes only. The demarcated portions of the trials indicate the length of the sequence buffer. When we are performing real-time recall, this buffer acts as a "sliding window" across the entire length of the trial. We can think about the demarcated "training" section shown as a snapshot of the sliding window at some time beyond the start of the trial. Notice that this is also the section that most clearly represents a divergence between the different types of trials.

Also shown in Fig. 4 is the dynamic counterpart to the confusion matrix, which illustrates the $\log \delta_T^*$ values for all the trials as they are recalled on different models. Trials of type 1 indicate the operator reached for the horizontal handrail, and trials of type 2 indicate that the operator reached for the vertical handrail. The variables $g(k,1)$ and $g(k,2)$ will be defined shortly. Because of the "sliding window" used during real-time recall, there should actually be a second time index to bookkeep the sequence length (T) as well as the time step (k). Therefore, hereafter we shall refer to the metric $\log \delta_T^*$ as $\log \delta_{k:k+T}^*$. The analogy of the confusion matrix to the results shown in Fig. 4 comes about due to the top half of the graphs shown. These 4 graphs represent a matrix of recalls on model types vs. trial types. The graphs along the diagonal represent the model recalling on trials of similar types, and the graphs on the off-diagonals represent models recalling on trials of opposite types.

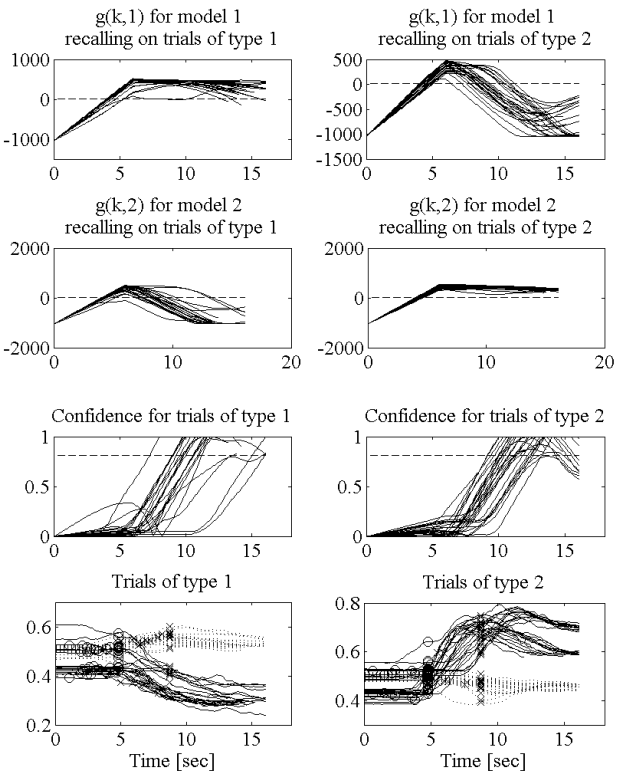

Figure 4: Y and Yaw validation trials shown with corresponding $\log \delta_{k:k+T}^*$ and confidence

As with the confusion matrix, we desire to have "0 on the off-diagonals." The interpretation of this is that we elicit as few false alarms as possible based upon the real-time $\log \delta_{k:k+T}^*$ results shown in each graph. In order to compute false alarms as well as missed detections and average time to prediction for correctly classified trials, we must perform real-time arbitration with the aid of many thresholds. As shown in the top 4 graphs of Fig. 4, there is a dashed line indicating an important threshold. This threshold plays a major part in the heuristically-driven prediction algorithm, which is outlined as follows:

For each time step k in the streaming real-time data set, perform the following:

1. Compute $\log \delta_{k:k+T}^*$ based upon the finite sequence length T by using the Viterbi algorithm (recall).

2. Subtract off model bias from trial:
   $g(k,m) \triangleq \log \delta_{k:k+T}^*(m) - \delta_m$,
   where $m \in \mathcal{M} \equiv \{\text{Horizontal}, \text{Vertical}\}$ indexes the model being recalled on.

3. For the current trial, determine and store the model yielding the maximum value between the quantities computed in the previous step, i.e. find
   $\hat{m} = \arg\max_m g(k,m)$.

4. Compute and store the confidence value, c, that indeed $\hat{m} = \arg\max_m g(k, m)$.
5. If $g(k, m) > T_d$, check if $\hat{m} = \arg\max_m g(k, m)$ from Step 3 references the same model as it did in the previous time step k. If so, increment a counter for model $\hat{m}$, otherwise, reset the counter to 1, for the first count of a newly arbitrated $\hat{m}$. If $g(k, m) \leq T_d$, reset the counter to 0. Note that T_d is *not* a model-specific threshold (shown as a dashed line in Fig. 4).
6. If the counter for model $\hat{m}$ exceeds a predetermined number (denoted as maximum hysteresis count), then use the confidence value computed in Step 4 for final arbitration.
7. If the confidence value, c, exceeds a predetermined threshold, T_c, then sound prediction alarm for model $\hat{m}$.

Step 4 in the algorithm is computed as follows, to yield a number $c \in [0, 1]$:

$$c = \frac{|g(k, \text{Horizontal}) - g(k, \text{Vertical})|}{C_s}$$

In the formula above, C_s is the confidence scale parameter. The idea of the confidence value, c, is to arbitrate between the models by computing the difference between the horizontal and vertical likelihoods, scaled by a judiciously selected factor, C_s, that will yield values $c \in [0, 1]$. If $c > 1$, then we set $c = 1$. The confidence values are shown on the third row of graphs in Fig. 4, with the corresponding confidence thresholds shown as dashed lines. All of the thresholds, biases, and scaling constants mentioned thus far for Viterbi recall are excellent candidates (i.e. "free parameters") for dynamic threshold optimization, and will be discussed in depth later.

Notice that the "off-diagonals" of the top 4 graphs in Fig. 4 have features that distinguish them from the graphs on the diagonals. The "off-diagonal" $g(k, m)$ values appear to rise, then fall with k, whereas the diagonal $g(k, m)$ values rise, then settle out above the thresholds for the most part. This distinguishing characteristic, in addition to the confidence values, provide the basis for the algorithmic construction outlined above. The reason for the initial pseudo-linear rise in all 4 graphs is due to the use of buffering required by the Viterbi algorithm. The buffer, $\{k : k + T\}$, is initialized with all zeros at first, and then the computation of the biased likelihood $g(k, m)$ grows exponentially until the buffer is full. At this point, the buffer slides forward with no leading zeros, allowing for the algorithmic construct to produce a robust model arbitration. Due to the delay in waiting for the buffer to fill, using buffer-based algorithms are potentially slow in meeting one of main performance specifications related to minimizing average time to prediction for correctly classified trials. Therefore, we will present an alternative to buffering methods shortly.

The bottom row of plots in Fig. 4 illustrate scaled feature vectors ranging between 0 and 1 (using min and max observed values for scaling), for y and yaw, in lieu of their original raw values. The scaled values are used as observation sequences for training the HMMs. This is done so that POR-based data can be comparable with each other (i.e. position is measured in cm, and orientation is measures in radians). This also becomes important when augmenting a POR-based feature vector with Euclidean distance data (to the handrails). In either case, it is still important to maintain a solid basis for comparison of the results.

2.1.2 Posterior probability-based recall

As an alternative to the Viterbi-based algorithm, we may use part of an algorithm that is traditionally used for inference during the Baum-Welch re-estimation procedure. Also referred to as the EM algorithm for *Estimate* (E-step) and *Maximize* (M-step), we borrow results from the E-step, in which inference amounts to computing the posterior probability:

$$p(q_t | y_0, \ldots, y_t) = \frac{\alpha(q_t)}{\sum_{q_t} \alpha(q_t)} = \frac{\alpha(q_t)}{p(y_0, \ldots, y_t)}$$

Notice that the formula is based upon some new quantities, specifically, $\alpha(q_t) \triangleq p(y_0, \ldots, y_t, q_t)$. A forwards recursive formula updates $\alpha(q_t)$ (α recursion), working only with data up to time t, and as such is real-time in nature. However, when used for the EM algorithm, this forwards algorithm complements a backwards recursive algorithm also run as part of the procedure, starting at final time T, and working backwards to time $t + 1$. This is called β recursion, where $\beta(q_t) \triangleq p(y_{t+1}, \ldots, y_T | q_t)$. Clearly this is performed offline in the context of the training that occurs with the EM algorithm. The details of these recursive formulae and the EM algorithm can be found in the literature [4, 7].

Due to the nature of α recursion, we can apply the algorithm across the same buffer of data operated on by the Viterbi algorithm outlined earlier. The buffer of data will still slide forwards in the same manner, accumulating new streaming observations with time. However, a major difference between implementation of the posterior-probability based recall is that the $\log \delta^*_{k:k+T}$ metric will be replaced with one that is based upon the state value. Specifically, at each time instant k, the recursive α formula will be run over the buffer of data $\{k : k + T\}$. Throughout the execution of the algorithm, the state corresponding to the maximum posterior probability value encountered over the current contents of the buffer will be stored. Mathematically, this can be represented as a two-step optimization problem as follows (where t: time in the buffer):

$$\begin{aligned} \hat{q}_t &= \arg\max_{q_t} p(q_t | y_0, \ldots, y_t) \\ \hat{p}_t &= \max_{q_t} p(q_t | y_0, \ldots, y_t) \\ t_{buf} &= \arg\max_t \hat{q}_t \\ \hat{q}_{t_{buf}} &= \max_t \hat{q}_t \end{aligned}$$

The result $\hat{q}_{t_{buf}}$ is then compared with canonical state values

to determine the progression of the data through a Markov chain. Because the tied mixture HMMs are trained as left-right models, the Markov chain should naturally proceed from the initial state, 1, to the final state, M. These are the "canonical" state values referred to previously. Clearly, they represent a causal link to progression of the Markov chain, and will aid us in determining the status and classification of the task at hand. Therefore, we will use the canonical state values to help with arbitration between models being recalled on differing trial types. The remainder of the algorithm pertaining to this arbitration can be summarized as follows:

1. Check to see if $\hat{q}_{t_{buf}}$ is equal to 1 or M, for both models.
2. If, for a particular model, m, the following conditions hold true, then sound prediction alarm for model m:
   (a) $\hat{q}_{t_{buf}} = M$ for model m.
   (b) $\hat{q}_{t_{buf}} = 1$ for all models other than m.
   (c) $\hat{p}_{t_{buf}} > 0.95$.

Notice that there is a condition, $\hat{p}_{t_{buf}} > 0.95$, corresponding to the maximum posterior probability value encountered over the contents of the buffer. In order to elicit an alarm, we require that this value be above 0.95. This is analogous to the confidence value used with Viterbi recall, and could possibly be used for dynamic threshold optimization. However, due to the nature of the algorithm's implicit "max" bias, performing such an optimization may not be necessary.

Finally, we can also implement real-time recall using posterior probability computations *without* the use of a buffer. There are computational advantages to using this method, in contrast to the previously discussed disadvantages of using algorithms based upon buffers. As such, we can use a slight modification of the buffered version of the algorithm. We can now reduce the mathematical representation of the two-step optimization problem to a single step optimization problem, as follows (where t: now refers to real-time):

$$\begin{aligned} \hat{q}_t &= \arg\max_{q_t} p(q_t|y_0,\ldots,y_t) \\ \hat{p}_t &= \max_{q_t} p(q_t|y_0,\ldots,y_t) \end{aligned}$$

The remainder of the algorithm pertaining to arbitration is very similar, with only very slight changes summarized as follows:

1. Check to see if $\hat{q}_t$ is equal to 1 or M, for both models.
2. If, for a particular model, m, the following conditions hold true, then sound prediction alarm for model m:
   (a) $\hat{q}_t = M$ for model m.
   (b) $\hat{p}_t > 0.95$.

Because this algorithm requires no buffering, there is hence no need for static optimization, and dynamic optimization is very computationally efficient. The optimization parameter is the confidence threshold, shown in the algorithmic summary above as 0.95. Previously, we hypothesized that the buffered version of the algorithm may exhibit very little sensitivity to this threshold. Although this is still the case when using the non-buffered method, marginal improvements in average time to prediction for correctly classified trials can be claimed by performing the dynamic optimization. Fig. 5 illustrates the striking transition of posterior probabilities for the final state only, i.e. $p(q_t|y_0,\ldots,y_t)$ shown for $i = M$, for models recalling on trials of the same type. For correctly classified trials on the "diagonals," the state transition occurs very rapidly, in contrast to the rather slow, linear transition of the $\log \delta^*_{k:k+T}$ metric shown in Fig. 4.

For certain trials on the lower diagonal, the posterior probability reverts back to a very small value just prior to grasp. This aberration would give us pause if the nature of the algorithm were to alarm based upon continuous observation. However, because our goal is to classify correctly as soon as possible, once the classification is performed and the alarm is triggered due to arbitration, there is no further need for monitoring the posterior values. This is especially true due to the fact that an optimal alarm triggered to initiate autonomous action moves us further along the sliding scale of autonomy than continuous monitoring of confidence values within the predictive interface as in [9].

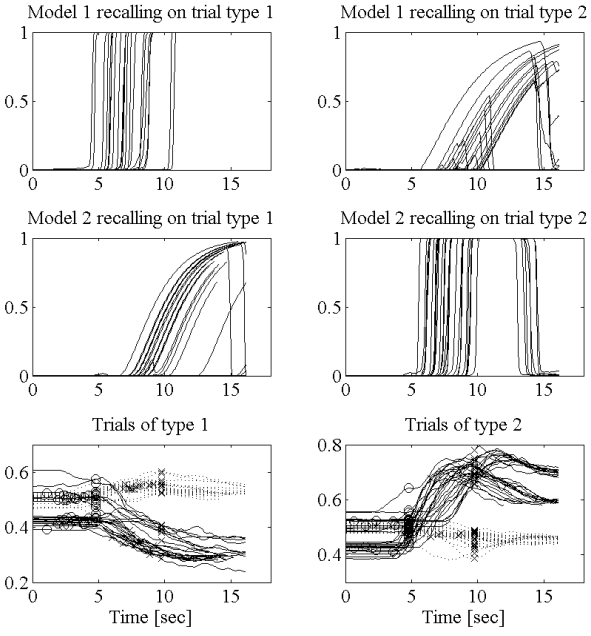

Figure 5: Y and Yaw validation trials shown with corresponding posterior probabilities, $p(q_t|y_0,\ldots,y_t)$

As mentioned before, only marginal improvements may be achieved by performing dynamic optimization over the confidence threshold for real-time non-buffered posterior probability-based recall. As seems clear by examining Fig. 5, using the

transition to the final state as a fundamental part of the algorithm, and optimizing to obtain marginal improvements may not be the optimal solution to our problem. Further improvements may be made by making use of some weighted combination of the posterior probability over all states as it exceeds some predetermined threshold. This intuitive concept can be formalized in the roots of decision theory, and is the subject of future work to be presented in a sequel paper by the lead author.

2.2 Optimization Methods

In order to find the optimal model training parameters when using a sequence buffer, the buffer can be parameterized to incur the fewest number of errors. As shown in Fig. 4, we have a fixed sequence buffer, which can be parameterized by the time prior to grasp and the sequence buffer length. Because we are interested in the fewest errors and maximizing the time *before* grasp, we propose this parametrization as an anecdote. In essence, this can be thought of as a "static optimization," where the cost function being optimized is the sum of the off-diagonals of the confusion matrix, $\mathbf{M}$, which quantify the errors (false alarms). In our case, since we only have two models over which to arbitrate, the confusion matrix is 2×2, and we can formally pose the optimization problem as follows:

$$\begin{aligned} \text{Solve} \quad & \arg\min_{\lambda_\mathbf{s}} tr(\mathbf{PM}(\lambda_\mathbf{s})) \\ \text{where } \mathbf{P} \quad &= \quad \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \end{aligned}$$

$\mathbf{P}$ is a permutation matrix, and $\lambda_\mathbf{s}$ is a vector containing the optimization parameters defined previously. The cost function, $h(\lambda_\mathbf{s}) \triangleq tr(\mathbf{PM}(\lambda_\mathbf{s}))$, is simply the sum of the off-diagonals of the confusion matrix, or the errors accumulated during static recall validation. As such, $h(\lambda_\mathbf{s})$ can be plotted as a function of the two optimization parameters via a simple two-dimensional grid search to see if there are any global or local minima. We will provide these illustrations in the results section. Clearly, there is no closed-form solution for this optimization problem since $\mathbf{M}(\lambda_\mathbf{s})$ is based upon empirical evidence. Therefore, from the plots mentioned above, we can determine the optimal design points. Our goal is to find the region of the parameter space that incurs no errors, but is constrained to having a maximum time before grasp (related to one of our performance specifications), and smallest possible sequence buffer length. The latter constraint is imposed because smaller sequence buffers allow for more "sliding window" time during real-time recall so that there is more time for correct arbitration between models.

The real-time recall thresholds can also be optimized in a formal manner, based upon the metrics previously introduced as performance specifications. Therefore, we may pose a "dynamic" optimization problem. In this case, there are three competing objectives, and the goal is to determine the optimization parameters that provide an optimal tradeoff among them. Whether using the Viterbi or the posterior-based recall method, we shall denote the vector of thresholds being optimized as $\lambda_\mathbf{d}$. In the case of the Viterbi recall method, $\lambda_\mathbf{d}$ contains the following thresholds and other recall parameters: $\delta_m, \forall m \in \mathcal{M}$ (model biases), T_d, the detection threshold against which we compare the log likelihood values $g(k, m)$ of both models, the maximum hysteresis count (MHC), the confidence threshold, T_c, and the confidence scale parameter, C_s.

Therefore, ${\lambda_\mathbf{d}}^T = \begin{bmatrix} \delta_1 & \delta_2 & T_d & MHC & T_c & C_s \end{bmatrix}$. For the posterior-based recall methods, $\lambda_d = T_c$, whether buffered or non-buffered techniques are being used.

Now that the details of the optimization parameters have been highlighted, we still have yet to capture the essence of how to deal with optimizing competing objectives over these parameters. There are several methods to choose from, but as first step, we focus on a simple one which will solve the optimization posed as follows:

$$\begin{aligned} \text{Solve} \quad & \arg\min_{\lambda_\mathbf{d}} \mathbf{w}^T \mathbf{x}(\lambda_\mathbf{d}) \\ \text{where } \mathbf{w}^T \quad &= \quad \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \\ \text{and } \mathbf{x}^T(\lambda_\mathbf{d}) \quad &= \quad \begin{bmatrix} P_{md}(\lambda_\mathbf{d}) & P_{fa}(\lambda_\mathbf{d}) & t_p(\lambda_\mathbf{d}) \end{bmatrix} \\ t_p(\lambda_\mathbf{d}) \quad &= \quad \frac{t_c(\lambda_\mathbf{d})}{t_c(\lambda_\mathbf{d}) + t_g(\lambda_\mathbf{d})} \end{aligned}$$

$P_{md}(\lambda_\mathbf{d})$ and $P_{fa}(\lambda_\mathbf{d})$, are the probability of missed detection and false alarm, respectively. $t_p(\lambda_\mathbf{d})$ is the scaled average time to prediction for correctly classified trials, computed as shown above, where $t_c(\lambda_\mathbf{d})$ is the average time to prediction for correct trials, and $t_g(\lambda_\mathbf{d})$ is the average "free time," or time before the grasp for correct trials. In this way, $t_p(\lambda_\mathbf{d}) \in [0, 1]$, and can be directly compared to $P_{md}(\lambda_\mathbf{d})$ and $P_{fa}(\lambda_\mathbf{d})$, which are also $\in [0, 1]$. It should be evident at this point that our cost function is essentially an equally weighted sum of all of the competing objectives. This method of solving a multi-objective optimization problem is a common approach, but suffers from having to make judicious selection of the weights, $\mathbf{w}$. These weights are not necessarily a measure of the relative magnitude of the metrics, and furthermore, certain solutions may be inaccessible if a nonconvex Pareto frontier exists. However, it can easily be posed as an unconstrained nonlinear optimization problem, where a simplex method [6] can be implemented. Although we cannot visualize the results for the Viterbi recall method due to optimization over a high dimensional space, it will be shown that the optimization routine converges to a local minimum for a particular set of initial conditions in the subsequent section.

We optimize the static and dynamic parameters separately in order to reduce the computational burden of parameterizing them simultaneously. Performing the static optimization as a two-dimensional grid search is equivalent to training a hidden Markov model for each $\lambda_\mathbf{s}$ point within the grid, which is computationally burdensome. Moreover, one of the optimization parameters in $\lambda_\mathbf{s}$ is the time prior to grasp. This is related to the average time to prediction for correct trials that is part of the cost function for dynamic optimization. As such, the problem would be ill-posed if both static and dynamic optimization problems were combined. Hav-

ing the two optimizations performed separately also allows us use the most well-trained model already statically pre-optimized with respect to a constraint (i.e. a lower bound) on the maximum time prior to grasp. Given that, we can improve upon the static validation performance by performing the secondary dynamic optimization. This further refines the validation performance with respect to the parameters in $\lambda_{\mathbf{d}}$, during real-time recall.

3. RESULTS

3.1 Static Optimization

An optimal design point can be found for static validation when using the y and yaw, POR-based feature vector. This design point reflects the best parametrization of the training segmentation for data used to train HMMs, using the Viterbi recall method. Shown in Fig. 6 is the sum of the off-diagonals of the confusion matrix, $h(\lambda_{\mathbf{s}})$, as a function of the optimization parameters: the time prior to grasp and the sequence buffer length.

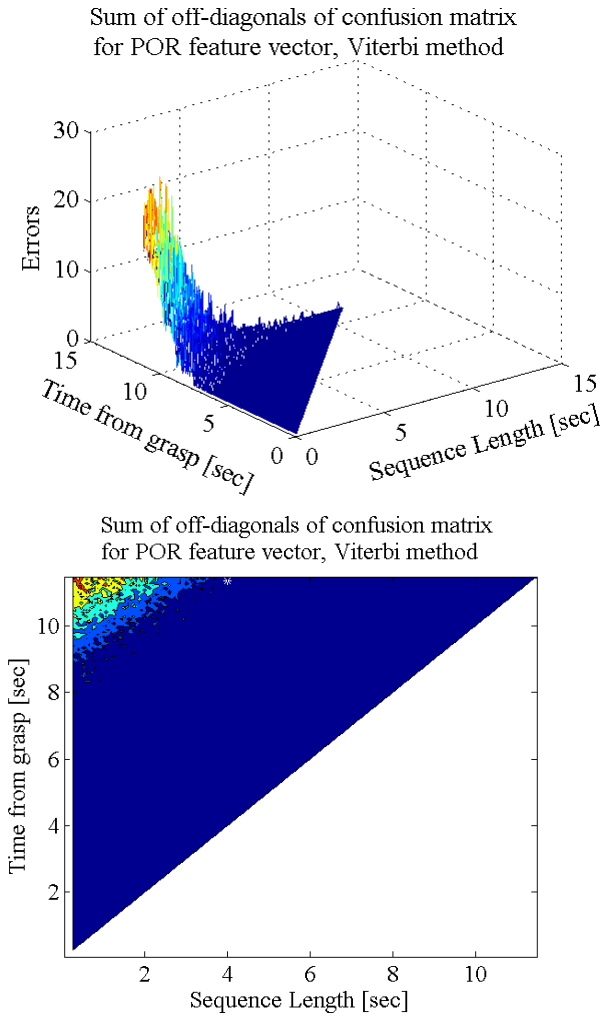

Figure 6: Optimal Design Point for Y and Yaw Feature Vector, Viterbi Method

As seen in Fig. 6, both the 3D and the contour plot view are given. It is clear that there is an area corresponding to a large accumulation of errors, for large times from grasp and small sequence lengths. Recall that our goal for static optimization is to find the region of the parameter space that incurs no errors, but is constrained to having a maximum time before grasp, and smallest possible sequence buffer length. As such, we have a conflict of interest between our goal and the constraints. However there is enough area in the parameter space where there is negligible error accrual to accommodate our requirements. The optimal design point has been marked with an asterisk ($*$) on the contour plot, for an optimal time from grasp of 11.3 sec and a sequence buffer length of 4 sec.

Similar contour plots can be presented for other cases of interest, when using other feature vectors, and for the posterior method, when using buffering. Static optimization clearly does not apply to the non-buffered posterior method, because there is no buffer that can be optimally parameterized. There are two feature vectors other than the POR-based one (y-yaw) to be studied: a feature vector based exclusively on distance data, and a feature vector based on y-yaw, and scaled distance data. Because of the scaling that is performed for comparability among variables, we may obtain varying results when considering the feature vector based solely on distance. Therefore, both the scaled and unscaled versions of the distance-only based feature vectors will be investigated.

It is therefore of interest to look at the contour plots for the methods or feature vectors that yield qualitatively significant differences. For the Viterbi recall method, the contour plots providing the static optimization results for feature vectors based on variables other than y-yaw are very similar to the one shown in Fig. 6. As such, and in the interest of conserving space, we will refrain from showing these plots. However, using the posterior (buffered) recall method illustrates a significant difference, especially when using the distance feature vector. The resulting contour plots are shown in Fig. 7, for both the scaled (top) and unscaled (bottom) distance-based feature vector.

Notice the difference between Figs. 6 and the top plot of 7: the results for the Viterbi method with POR-based feature vector are much cleaner than when using the posterior method with the distance-based (scaled) feature vector. One reason for this is that the errors are computed in a different manner for each method. Errors accumulated when using the Viterbi method are based on a log-likelihood metric and obtained by adding the off-diagonals of the confusion matrix. Errors accumulated with the posterior method are based on the respective algorithm described in the previous section, and by scaling the fraction of those correctly classified to the total number (not unlike summing off-diagonals of the confusion matrix). Another major difference is in the use of scaled distance as a feature vector rather than a POR-based feature vector. We may compare the results shown on the top plot of Fig. 7 to other results using the same scaled distance feature vector (not shown), using different recall methods. The results do vary, but not as significantly as seen between the POR-based results using the Viterbi recall method shown in Fig. 6 and the scaled distance results using the posterior recall method in Fig. 7.

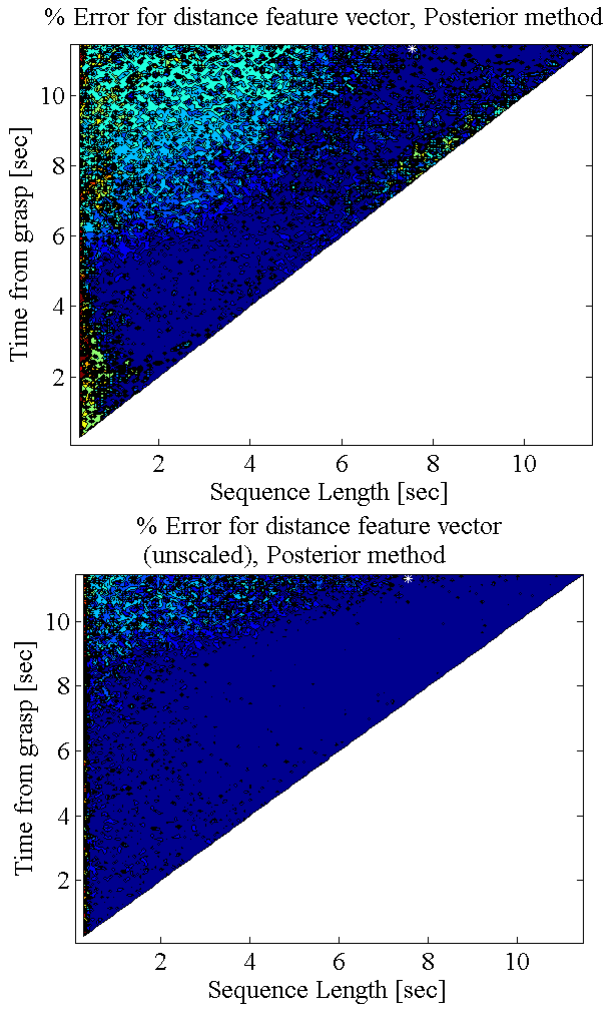

Figure 7: Optimal Design Point for Distance-Based Feature Vector, Posterior (Buffered) Method

As seen in Fig. 7, there is a "filtering" effect that cleans up the contour plot results when using the unscaled version of the distance feature vector (on bottom) as opposed to the scaled version (on top). This is due to the fact that the means of the unscaled observation sequences have more distinguishability when training an HMM than when they are scaled to values between 0 and 1. In summary, optimal design points for feature vectors and methods of all types are provided in Table 1. The results provided are for the optimal sequence buffer length only. For all feature vectors and methods, the optimal time prior to grasp can be set to 11.3 sec. Results provided for the "distance" only feature vector apply for both scaled and unscaled versions.

3.2 Optimization of Training/Validation Segmentation

Now that we have the optimal design points and best trained HMMs from a trial segmentation standpoint, we can perform dynamic threshold optimization. However, notice in Figs. 6 and 7 shown in the previous subsection, that there was a bit of noise present in the contour plots, and best seen in

Recall Method \ Feature Vector	POR (Y-yaw)	Distance to handrails	Combo (POR & distance)
Viterbi Method	4 sec.	5 sec.	6 sec.
Posterior Method	5 sec.	7.53 sec.	5 sec.

Table 1: Optimal sequence buffer length

the 3D plot from Fig. 6. This is due to the fact that the training and validation sets were randomly partitioned. This random partitioning may also account for some variance in the results obtained. Therefore, before proceeding further to the dynamic threshold optimization, we want to ensure that we study and understand the consequences of this behavior.

It is has been demonstrated that the results (average time to prediction for correctly classified trials, false alarms, missed detections) may vary considerably based upon using a particular method. As such, we can determine the best training/validation set segmentation for a particular recall method by continuously randomizing the training/validation segmentation based upon the statically optimized parameters from the previous subsection. This randomization continues until the performance specifications fall within a required preset tolerance. At this point, we may use the resulting training/validation segmentation as a basis for comparison when generating results using a different recall method. If the results vary greatly between the two recall methods, then we know there is a statistical bias for the feature vector under consideration.

Performing this test for all feature vectors studied will allow us to make a judicious feature selection. Furthermore, in performing this experiment, we can develop an intuitive sense of a "canonical" representation for a training/validation segmentation that works well for each method by examining the respective superimposed trials in a plot. This qualitative information is also very useful for building intuition on how to design future experiments. Of course, when randomizing the segmentation based upon a particular recall method, there is an inherent bias towards using that recall method. However, our aim is to find the recall method and feature selection that will yield the most desirable results. As such we will run randomization tests which are recall-biased for both posterior (non-buffered) and Viterbi methods. This will be followed by performing dynamic optimization based upon the feature vector that yields the most robust results across recall methods.

The main findings of our tests indicate that the combination POR/scaled distance feature vector exhibits the most robust behavior. In fact, when validating with the Viterbi recall-biased training/validation segmentation, both methods meet the required performance specifications. As such, we will use the combination POR/scaled distance feature vector as the candidate for study in the next subsection. There we will perform dynamic threshold optimization for all available recall methods, using hidden Markov models trained with biases towards both recall methods.

3.3 Dynamic Threshold Optimization

The results for dynamic threshold optimization provided in this subsection pertain only to the combination POR-based (y-yaw)/distance feature vector. We will first randomize the training/validation segmentation with a posterior recall method bias to achieve a zero probability of false alarm and missed detection, and a average time to prediction for correctly classified trials below 8 sec. The initial results and optimization parameters, as well as the optimized results and parameters for both the buffered and non-buffered posterior methods are shown in Table 2.

Table 2: Optimization Results for Posterior Method, training/validation segmentation with posterior recall bias

Buffering	Buffered	Non-Buffered
Initial T_c	0.95	0.95
Optimized T_c	0.95	0.7125
Initial P_{fa}	0%	0%
Optimized P_{fa}	0%	0%
Initial P_{md}	0%	0%
Optimized P_{md}	0%	0%
Initial t_c	9.46 sec	7.89 sec
Optimized t_c	9.46 sec	7.82 sec

We can verify a hypotheses stated in the previous section using the information in Table 2. Recall that for the buffered posterior recall method, in order to elicit an alarm we require that the confidence threshold be above 0.95. Our hypothesis was that due to the nature of the algorithm's implicit "max" bias, performing this optimization may not be necessary. As seen in the first column of Table 2, the optimized values don't change at all from the initial values, substantiating our hypothesis.

Note that a double line separates the table. The parameters above the double line are for optimization, and the parameters below the double line are the objectives. For the non-buffered posterior recall method, we hypothesized that the algorithm would exhibit very little sensitivity to the confidence threshold, and that only marginal improvements in average time to prediction for correctly classified trials would be claimed by performing the dynamic optimization. Evidence of this is also provided in Table 2, where we only lose hundredths of a second in reducing the average time to prediction by using an optimized confidence value of 0.71025 in lieu of 0.95.

Table 3 provides the optimization results when applying the Viterbi recall method on models that were based upon a training/validation segmentation with posterior recall bias. The table lists initial results and optimization parameters, as well as optimized results and parameters.

Here, the probability of false alarm and missed detection, and average time to prediction do not differ greatly than when we used the posterior recall methods. This was our expectation, due to the robustness of using the combination feature vector. Also evidenced in Table 3, we can improve even further upon those results, based in part on the initial set of optimization parameters selected. The initial set of optimization parameters listed in the column labeled "First Values" was generated by trial, error, and intuition. As seen, it is possible to improve upon the average time to prediction by fractions of a second.

Table 3: Optimization Results for Viterbi recall Method, training/validation segmentation with posterior recall bias

Parameters and Metrics	First Values	Second Values
Initial δ_1	0	0
Optimized δ_1	0.00035375	2.0833×10^{-5}
Initial δ_2	0	0
Optimized δ_2	0.0001068	$2.0833 \times^1 0^{-5}$
Initial T_d	-600	125
Optimized T_d	-629.32	125.52
Initial MHC	5	25
Optimized MHC	5	26
Initial T_c	0.81	0.81
Optimized T_c	0.79836	0.81338
Initial C_s	600	600
Optimized C_s	483.54	602.5
Initial P_{fa}	0%	0%
Optimized P_{fa}	0%	0%
Initial P_{md}	2%	2%
Optimized P_{md}	2%	0%
Initial t_c	7.6486 sec	7.9347 sec
Optimized t_c	7.3083 sec	7.985 sec

In the column labeled "Second Values," the initial optimization parameters are selected in contrast to other parameters in order to determine if there any sensitivities of the optimized results to the initial starting points. In essence, we would like to be able to determine if there is agreement or disagreement between optimum points found by using differing recall methods. As such, we can speak to the convergence points of the multi-objective optimization problem being global or local minima. It appears that the multi-objective optimization problem posed as an unconstrained nonlinear cost function clearly has local minima, and we may arrive at solutions that are very sensitive to starting location.

Similar conclusions can be surmised when randomizing the training/validation segmentation with a Viterbi recall method bias to achieve a zero probability of false alarm and missed detection, and a average time to prediction below 10 sec. The initial results and optimization parameters, as well as the optimized results and parameters for all recall methods are shown in Table 4 and 5.

This provides use with further evidence that *optimization yields improvements*, and in this case they appear to be more substantial, particularly for the Viterbi recall method. We see great reduction in average time to prediction, at the expense of a slight increase in the probability of missed detection. For the posterior recall methods, however, we see that the improvements are still only marginal. Therefore, for the posterior recall method, the truly effective steps in the optimization procedure come from static optimization and optimization of the training/validation segmentation. However, as stated earlier, the arbitration and machinery behind the

Table 4: Optimization Results for posterior recall method, training/validation segmentation with Viterbi recall bias

Buffering	Buffered	Non-Buffered
Initial T_c	0.95	0.95
Optimized T_c	0.95	0.86094
Initial P_{fa}	0%	0%
Optimized P_{fa}	0%	0%
Initial P_{md}	0%	0%
Optimized P_{md}	0%	0%
Initial t_c	8.7456 sec	7.9864 sec
Optimized t_c	8.7456 sec	7.9483 sec

Table 5: Optimization Results for Viterbi recall Method, training/validation segmentation with Viterbi recall bias

Parameters/Metrics	First Values	Second Values
Initial δ_1	0	0
Optimized δ_1	9.1681×10^{-5}	0.0010858
Initial δ_2	0	0
Optimized δ_2	0.0005762	0.0045668
Initial T_d	-600	125
Optimized T_d	-714.28	28.419
Initial MHC	5	25
Optimized MHC	5	10
Initial T_c	0.81	0.81
Optimized T_c	0.81181	0.99835
Initial C_s	600	600
Optimized C_s	243.7	268.3
Initial P_{fa}	0%	0%
Optimized P_{fa}	0%	0%
Initial P_{md}	0%	2%
Optimized P_{md}	2%	2%
Initial t_c	7.8354 sec	7.9944 sec
Optimized t_c	6.5083 sec	7.0722 sec

posterior recall algorithms may not be the optimal solution to the problem of minimizing average time to prediction, as well as adhering to the other performance specifications. Further improvements may be made by making use of some weighted combination of the posterior probability over all states as it exceeds some predetermined threshold, in attempt to improve the performance of the algorithm.

The multi-objective optimization problem again clearly has local minima, with the solutions exhibiting sensitivity to starting location as shown in Tables 4 and 5. At any rate, out of all of the cases investigated, there is no clear "winner," with regards to the recall method, or recall bias. However, we *can* conclude that the combination feature vector provides us with the most desirable solution. As expressed in previous work [9], it is possible to shave more time off of the average time to prediction by allowing for some missed detections. We can achieve an average time to prediction as low as 6.5 sec if we allow a 2% probability of missed detection, for the Viterbi recall method with a Viterbi recall bias shown in Table 5. However, if we desire a zero missed detection *and* zero false alarm probability, the best we can do on average time to prediction is 7.82 sec, for the posterior recall method (non-buffered) with a posterior recall bias shown in Table 2. These same tradeoffs can be achieved by using alternative dynamic threshold optimization weightings (i.e. not equally weighted as is performed currently with $\mathbf{w}^T = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$), or using a completely different approach to the multi-objective optimization problem.

4. CONCLUSION

Here we provide a summary of the most important findings:

- We can meet the requirements set by our performance specifications by choosing any of the appropriate recall methods or recall biases that provide for it.
- Depending on how strict the performance requirements are set, we can trade off minimizing the probability of missed detection for further reduction in the average time to prediction for correctly classified trials.
- Feature selection must take into account the effects of any statistical bias with respect to the randomizing the training/validation partitioning.
- Augmenting the feature vector with additional variables that provide more discriminatory power such as distances to handrails make the resulting hidden Markov models robust with respect to randomizing the training/validation segmentation.
- The optimized results are very sensitive to the initial starting points, and convergence to local minima is the best we can do with the currently implemented optimization approach. However, we've been able to document verifiable improvement over the initial starting points, particularly when using the Viterbi recall/arbitration method.
- Future work should include exploration of alternative optimization techniques, and ways to enhance the posterior recall method based upon more rigorous decision theoretic concepts.

5. ACKNOWLEDGMENTS

The authors would like to thank Bill Bluethmann and Kim Hambuchen for making the training and validation data available, Mike Goza for providing the tele-operation effort, and Rob Ambrose's vision in making this project possible at NASA JSC.

6. REFERENCES

[1] W. Bluethmann, R. O. Ambrose, M. A. Diftler, S. Askew, E. Huber, M. Goza, F. Rehnmark, C. Lovchik, and D. Magruder. Robonaut: A robot designed to work with humans in space. *Autonomous Robots*, 14(2-3):179–198, 2003.

[2] G. D. Forney. The viterbi algorithm. *Proceedings of The IEEE*, 61(3):268–278, 1973.

[3] R. A. P. II, C. L. Campbell, W. Bluethmann, and E. Huber. Robonaut task learning through teleoperation. In *ICRA*, pages 2806–2811. IEEE, 2003.

[4] M. I. Jordan. An introduction to probabilistic graphical models. Manuscript used for Class Notes of CS281A at UC Berkeley, Fall 2002.

[5] I.-C. Kim and S.-I. Chien. Analysis of 3D hand trajectory gestures using stroke-based composite hidden markov models. *Applied Intelligence*, 15(2):131–143, 2001.

[6] J. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[7] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. pages 267–296, 1990.

[8] H. L. VanTrees. *Detection, estimation, and modulation theory*. J. Wiley, 1992.

[9] K. Wheeler, R. A. Martin, V. SunSpiral, and M. Allan. Predictive interfaces for long-distance tele-operations. In *8th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, Munich, Germany, September 2005.

[10] K. R. Wheeler and C. C. Jorgensen. Gestures as input: Neuroelectric joysticks and keyboards. *IEEE Pervasive Computing*, 2(2):56–61, 2003.