

# Optimized Algorithms for Prediction within Robotic Tele-Operative Interfaces

Rodney A. Martin<sup>1</sup>, Kevin R. Wheeler<sup>2</sup>, Mark B. Allan<sup>3</sup>, and Vytas SunSpiral<sup>3,4</sup>

<sup>1</sup> Intelligent Systems Division  
NASA Ames Research Center  
Mail Stop: 269-1  
Moffett Field, CA 94035-1000  
[rmartin@email.arc.nasa.gov](mailto:rmartin@email.arc.nasa.gov)

<sup>2</sup> Monterey Bay Aquarium Research Institute  
7700 Sandholdt Road  
Moss Landing, CA 94039  
[kevinwheeler@ieee.org](mailto:kevinwheeler@ieee.org)

<sup>3</sup> QSS Group, Inc.  
NASA Ames Research Center  
Mail Stop: 269-3  
Moffett Field, CA 94035-1000  
[{mallan,vytas}@email.arc.nasa.gov](mailto:{mallan,vytas}@email.arc.nasa.gov)

<sup>4</sup> Formerly published as Thomas Willeke

**Summary.** 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 recent 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 work, Hidden Markov Models (HMM's) were trained on data recorded during tele-operation of basic tasks. 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.

## 1.1 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, tele-operation 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 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. Scalability is also certainly an issue for sliding autonomy, and is an open question in the methods we describe here and in previous work [9]. However, this is a first attempt at optimizing these methods, and this paper is meant to focus on very simple tasks in order to set a precedent for future work. Increasingly complex tasks may benefit from more advanced probabilistic methods involving scalable, hierarchical architectures.

As part of the precursor to the work presented here, we have previously developed predictive techniques and implemented them algorithmically for use during the manual tele-operation phase [9]. These prediction algorithms have 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 – Minimize (as low as possible)
- Average time to prediction for correctly classified trials – Minimize (as early as possible)

Other approaches will be examined as alternatives to prediction algorithms used in prior work [9]. In our previous work, we have trained and implemented

Hidden Markov Models (HMMs) for prediction of operator intent using available data. The intention of the operator is conveyed to the robot via a predictive interface. This predictive interface allows for the tele-operator to execute remote commands in a completely simulated environment that runs on the tele-operator’s side of the time delay. The interested reader is referred to [9] for more detail on this interface. The available data mentioned previously is collected during the execution of pre-specified tele-operated tasks. This data is used as input into the selected algorithmic mechanism for predicting the operator’s intention. 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]. The results we will present are not directly comparable to those of [5], as we use a different approach in forming the feature vector used to train the HMM’s, and the application we present here is for trajectory prediction as opposed to gesture recognition. However, it is possible to improve upon the results in [9] to more closely achieve the performance requirements stated above.

Our new, alternative approaches as well as our previously implemented techniques all 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 interested in minimizing the average time to prediction for correctly classified trials.

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. 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.

Using the HMM paradigm, both off-line (static) and on-line/real-time (dynamic) validation 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.

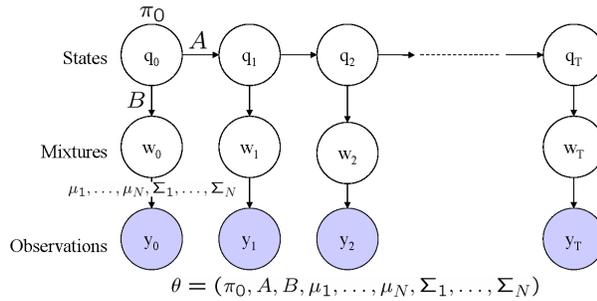
The new methods which may allow us to achieve the performance specifications stated above are by modifying how we perform recall, using judicious feature selection, and optimizing both static model training and dynamic real-time recall parameters. In previous work, we used subsets of a POR-based feature vector. POR stands for Point of Resolution, which is a 4x4 homogeneous transform matrix representing the commanded position and orientation of the back of the robot’s hand decomposed into position (x-y-z) and orientation (roll-pitch-yaw). These feature vectors act 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 target objects. 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. We will perform dynamic optimization of the other competing methods as well, where applicable.

## 1.2 Methodology

### 1.2.1 Hidden Markov Model Implementation

Practical implementation of Hidden Markov Models has been covered in 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]. We have chosen to implement a tied-mixture Hidden Markov Model with  $M = 3$  states and  $N = 6$  mixtures. Fig. 1.1 shows the graphical model topology and relevant parameters for this variant.



**Fig. 1.1.** Tied Mixture Hidden Markov Model

$t \in \{0, \dots, T\}$  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  $\theta$ , and are defined as follows:

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

### 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:

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

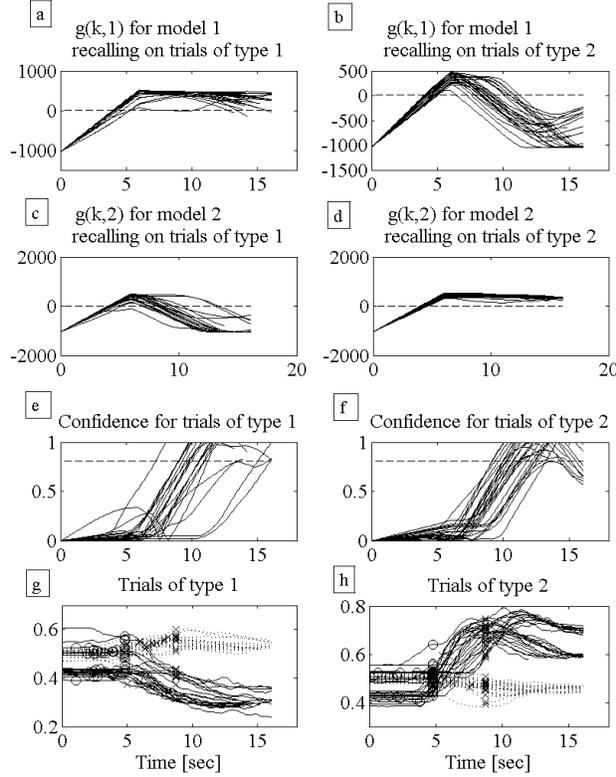
Therefore,  $\delta_T(i)$  is the probability that the state sequence ends up in state  $i$  at final time  $T$ . The algorithm to solve this optimization problem is recursive in nature, and the termination step provides us with  $\delta_T^* = \max_i \delta_T(i)$ , the maximum probability over the possible 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 tested obeys the model being recalled on.

To gain more clarity about the difference between static recall and dynamic real-time recall, refer to Fig. 1.2, which shows an example of the y-yaw feature vector on graphs “g” and “h”. The sequences shown as solid lines are for the y variable, and dotted lines are for the yaw variable. They represent portions of validation trials run for a fixed time prior to the time that a handrail is grasped. The “training” segments are demarcated as starting with circles ( $\circ$ ), and ending with crosses ( $\times$ ). Because the data displayed in these plots are validation trials, the training demarcation is for illustrative purposes only. However, 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. This is also the section that most clearly represents a divergence between the different types of trials.

Fig. 1.2 also illustrates the  $\log \delta_T^*$ -based values ( $g(k, 1)$  and  $g(k, 2)$ ) for all the trials as they are recalled on different models on graphs “a” through “d”. 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. 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  $\log \delta_{k:k+T}^*$ .

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 graphs “a” through “d” of Fig. 1.2, 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:



**Fig. 1.2.** Y and Yaw validation trials shown with corresponding  $\log \delta_{k:k+T}^*$  and confidence

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  $m \in \mathcal{M} \equiv \{\text{Horizontal, Vertical}\}$  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. 1.2).
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 graphs “e” and “f” in Fig. 1.2, 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 in Fig. 1.2, the graphs labelled “b” and “c” have features that distinguish them from the graphs labelled “a” and “d”. The  $g(k, m)$  values on graphs “b” and “c” appear to rise, then fall with  $k$ , whereas the  $g(k, m)$  values on graphs “a” and “d” 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 monotonic rise in all graphs labelled “a” through “d” 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 monotonically 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.

### 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:

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

Notice that the formula is based upon some new quantities, specifically,  $\alpha(q_t) \triangleq p(y_0, \dots, y_t, q_t)$ . A forwards recursive formula updates  $\alpha(q_t)$  ( $\alpha$  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  $\beta$  recursion, where  $\beta(q_t) \triangleq p(y_{t+1}, \dots, y_T|q_t)$ . Clearly this is performed off-line 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  $\alpha$  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  $\alpha$  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):

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

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):

$$\hat{q}_t = \arg \max_{q_t} p(q_t | y_0, \dots, y_t)$$

$$\hat{p}_t = \max_{q_t} p(q_t | y_0, \dots, y_t)$$

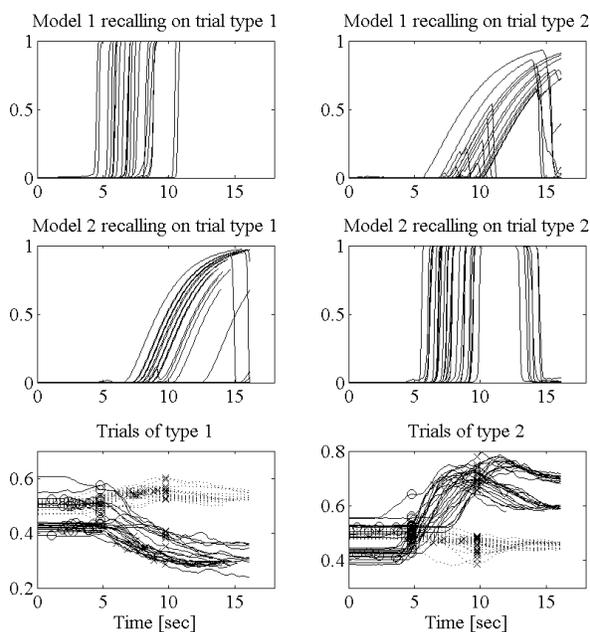
The remainder of the algorithm pertaining to arbitration is very similar, with only very slight changes summarized as follows:

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. 1.3 illustrates the striking transition of posterior probabilities for the final state only, i.e.  $p(q_t | y_0, \dots, y_t)$  shown for  $i = M$ , for models recalling on trials of the same

type. For correctly classified trials, the state transition occurs very rapidly, in contrast to the rather slow transition of the  $\log \delta_{k:k+T}^*$  metric shown in Fig. 1.2.

For certain trials, 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].



**Fig. 1.3.** Y and Yaw validation trials shown with corresponding posterior probabilities,  $p(q_t|y_0, \dots, y_t)$

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. 1.3, 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 may be investigated in future work.

### 1.2.2 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. 1.2, 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) accumulated during static validation. As such, it 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, where  $\lambda_{\mathbf{s}}$  is a vector containing the optimization parameters defined previously. Therefore, we can determine the optimal design points from the plots mentioned above. 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, but 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,  $C_h$ , the confidence threshold,  $T_c$ , and the confidence scale parameter,  $C_s$ .

Therefore,  $\lambda_{\mathbf{d}}^T = [\delta_1 \delta_2 T_d C_h T_c C_s]$ . For the posterior-based recall methods,  $\lambda_d = T_c$ , whether buffered or non-buffered techniques are being used. There are several optimization methods to choose from, but as first step, we focus on a simple one which will solve the problem posed as follows:

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

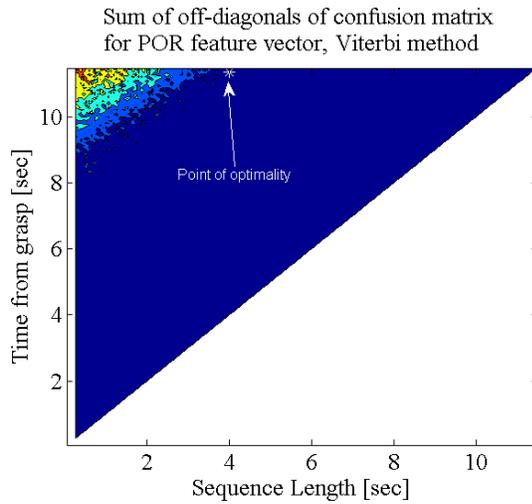
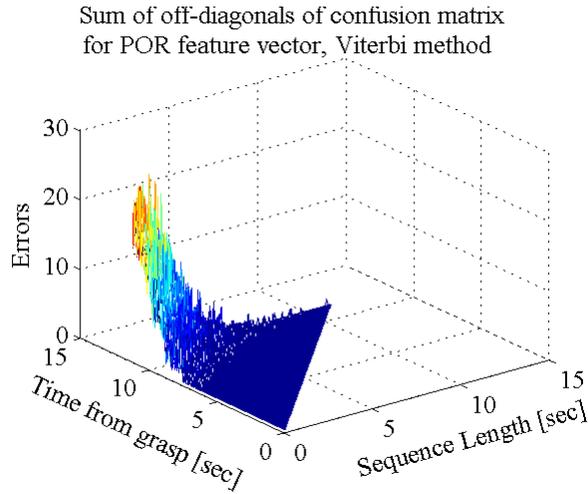
$P_{md}(\lambda_{\mathbf{d}})$  and  $P_{fa}(\lambda_{\mathbf{d}})$ , are the probability of missed detection and false alarm, respectively.  $t_p(\lambda_{\mathbf{d}})$  is the scaled average time to prediction for correctly classified trials, computed as shown above, where  $t_c(\lambda_{\mathbf{d}})$  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 the other competing objectives  $P_{md}(\lambda_{\mathbf{d}})$  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,  $\mathbf{w}$ . 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.

## 1.3 Results

### 1.3.1 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. 1.4 is the sum of the off-diagonals of the confusion matrix,  $\mathbf{M}(\lambda_{\mathbf{s}})$ , as a function of the optimization parameters ( $\lambda_{\mathbf{s}}$ ): the time prior to grasp and the sequence buffer length.

Both the 3D and the contour plot view are given in Fig. 1.4. 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



**Fig. 1.4.** Optimal Design Point for Y and Yaw Feature Vector, Viterbi Method

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 constructed for other cases of interest, i.e. 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. A combination feature vector based on y-yaw and scaled distance data is studied as well as the POR-based one (y-yaw). For the Viterbi recall method, the

contour plots providing the static optimization results for the combination feature vector is very similar to the one shown in Fig. 1.4. However, using the posterior (buffered) recall method results in a significant difference. For all feature vectors and methods, the optimal time prior to grasp can be set to 11.3 sec. The optimal sequence buffer length ranges between 4 and 7.53 sec, depending on the feature vector and recall method used.

### 1.3.2 Dynamic Threshold Optimization

The results for dynamic threshold optimization provided in this subsection pertain only to the combination POR-based (y-yaw)/scaled distance feature vector. This feature vector exhibits the most robust behavior with respect to random variations of the training/validation segmentation. For this feature vector, the results provide us with solid evidence that optimization yields substantial improvements, particularly when testing 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 only marginal.

Therefore, for the posterior recall method, the truly effective steps in the optimization procedure come from static optimization. However, as stated earlier, the arbitration and machinery behind the 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. Table 1.1 summarizes the results of the different methods and metrics. For the posterior method the (b) annotation refers to the buffered method, and the (n) annotation refers to the non-buffered method. For the Viterbi method, the two rows are shown correspond to two different optimization initializations with varied results.

**Table 1.1.** Optimization Results

Method	$t_c$	$P_{fa}$	$P_{md}$
Viterbi	6.51 sec	0%	2%
Viterbi	7.99 sec	0%	0%
Posterior (b)	8.75 sec	0%	0%
Posterior (n)	7.95 sec	0%	0%

Out of all of the cases investigated, there is no clear “winner,” with regards to the recall method. 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. However, if we desire a zero missed detection *and* zero false alarm probability, the best we can do on average time to prediction is 7.95 sec, for the posterior recall method (non-buffered).

## 1.4 Conclusion

In summary, we can meet the requirements set by our performance specifications by choosing any of the appropriate recall methods that provide for it. Furthermore, 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. Future work should include exploration of alternative optimization techniques, and ways to enhance the posterior recall method based upon more rigorous decision theoretic concepts.

## 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, 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. *Proceedings of the IEEE*, 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.