Towards Real-Time Cue Integration by Using Partial Results

Doug DeCarlo

Department of Computer Science and Center for Cognitive Science
Rutgers University, Piscataway, NJ 08854, USA,
decarlo@cs.rutgers.edu

Abstract. Typical cue integration techniques work by combining estimates produced by computations associated with each visual cue. Most of these computations are iterative, leading to *partial results* that are available upon each iteration, culminating in *complete results* when the algorithm finally terminates. Combining partial results upon each iteration would be the preferred strategy for cue integration, as early cue integration strategies are inherently more stable and more efficient. Surprisingly, existing cue integration techniques cannot correctly use partial results, but must wait for all of the cue computations to finish. This is because the intrinsic error in partial results, which arises entirely from the fact that the algorithm has not yet terminated, is not represented. While cue integration methods do exist which attempt to use partial results (such as one based on an iterated extended Kalman Filter), they make critical errors.

I address this limitation with the development of a probabilistic model of errors in estimates from partial results, which represents the error that remains in iterative algorithms prior to their completion. This enables existing cue integration frameworks to draw upon partial results correctly. Results are presented on using such a model for tracking faces using feature alignment, contours, and optical flow. They indicate that this framework improves accuracy, efficiency, and robustness over one that uses complete results.

The eventual goal of this line of research is the creation of a decision-theoretic meta-reasoning framework for cue integration—a vital mechanism for any system with real-time deadlines and variable computational demands. This framework will provide a means to decide how to best spend computational resources on each cue, based on how much it reduces the uncertainty of the combined result.

Keywords: Cue integration, real-time vision, meta-reasoning

1 Introduction

Robust vision systems cannot rely solely on one particular visual cue. With each cue comes limits on accuracy, limits on applicability, and variations in how much computation is required. Each individual cue reveals a separate aspect of the scene, but alone it will never suffice.

A. Heyden et al. (Eds.): ECCV 2002, LNCS 2353, pp. 327–342, 2002.

Vision systems will be successful only if they exploit the many sources of information in the image—it is crucial that we *integrate* the results from the various visual cues. This is, of course, well known. The human vision community has studied the problem of cue integration extensively [1,2,3], aiming to determine the strategy observers use to combine cues. These experiments suggest a mixture of statistical combination [1] and cue selection which can depend on viewing conditions [3] or context [2]. The sophistication of cue integration strategies in human observers casts doubt on whether computational vision systems can function without them. And in fact, research in computational vision has confirmed that combining multiple cues results in a more accurate and less ambiguous result, and a more robust system; typically the limits of only a few of the cues are taxed at any particular moment [4,5,6,7,8].

Such robustness is the only alternative to controlling the visual environment, and is almost certainly required for the development of a wide range of visual applications, including perceptual user interfaces and multi-modal interactive systems. These applications also bring real-time performance requirements. For there to be time to process and integrate multiple cues, we will need a principled approach that accounts for the information that each cue provides, while considering how much time is required to extract the information.

This paper launches a new approach to cue integration inspired by AI research on *meta-reasoning*, the problem of applying computational resources dynamically towards solving the problems in a developing time-critical situation. A complete real-time vision system using multiple cues would be able to use meta-reasoning to:

- schedule cue computations using decision-theoretic estimates of the value of their results;
- allow computations to be interrupted at any point yielding useful *partial results* for each cue;
- optimally combine all partial results computed in the time available, based on their uncertainty.

Such a strategy would yield the best possible estimate given the time restrictions.

This paper focuses on the last of these items: the problem of using partial results. Partial results inherently contain errors because their associated computation is not yet complete. Current cue integration strategies cannot use partial results because this error is not represented. This means that each cue computation must be allowed to finish, making it particularly difficult to build a real-time system that uses multiple sources of information, unless the associated computations are so simple that they always finish in a small and predictable interval. Furthermore, the advantage of using all of the information as early as possible is lost, making optimization problems more difficult and more likely to converge to the wrong answer. Finally, without the ability to terminate a cue computation in progress, computational resources must be wasted on cues that will not contribute to a robust estimate.

In this paper I develop a *probabilistic model of partial results* that can be used in existing cue integration frameworks. I evaluate this model using a model-based

face tracking system that uses three cues: features, contours and optical flow. This model leads to improved accuracy, efficiency and robustness.

The organization of this paper is as follows. Section 2 summarizes research on meta-reasoning and cue integration. This is followed by an informal discussion in Section 3 on the problems of using partial results in existing cue integration frameworks. This leads to the proposed solution, which is formalized and implemented in Section 4 and Section 5. Experiments using this model are presented in Section 6, followed by discussion of possible future investigations.

2 Background

2.1 Meta-reasoning

It is well understood that the design of real-time systems requires tradeoffs to be made between computation time and the quality of results. Research in meta-reasoning, in which systems take into account the resources required for their own computations in making decisions in order to judge the value of further deliberation, provides a formal approach to optimizing such tradeoffs in the face of real-time deadlines. Currently, this work is used in applications such as robot navigation under uncertainty [9] and rendering [10].

Anytime algorithms [11] form the basis of a widespread and successful approach to making real-time tradeoffs. Anytime algorithms produce improved results with more computation time, but provide some results regardless of how much of the computation has completed. This way, an appropriate amount of resources can be directed towards each computation, making the best possible partial computations available for use. For example, the real-time rendering application in [10] produced better renderings of an object given more time by simply rendering its constituent pieces one by one.

The performance of these algorithms is defined by a scalar measure of the utility of their output (or perhaps using a more abstract notion of "quality"). Such a measure is required for scheduling and making judgments on the validity of decisions made using these computations. With each algorithm comes a *performance profile* [11] as shown in Figure 1. A standard algorithm, whose profile is shown in Figure 1(a), produces useful results only after the completion of its entire computation—nothing is produced if such a computation is interrupted (before its completion, the quality is zero; afterwards, it is one). Figure 1(b) shows a profile for an idealized anytime algorithm, which shows quality smoothly increasing with computation time. In practice, improvements occur in jumps, such as after each iteration of a non-linear optimization, resulting in stepped profiles like (c). For example, the renderer in [10] assigned utilities to computations with a model of the perceptual quality gain for rendering individual scene components.

Computer vision algorithms that are iterative are *already* anytime algorithms. I take a model-based approach to vision and draw on many existing model-based algorithms as anytime algorithms. The reason is that each step in an optimization improves the result by aligning the model with the data. The computation can be

interrupted after each step if necessary. The model of partial results introduced in this paper serves as the performance profile for cue computations.

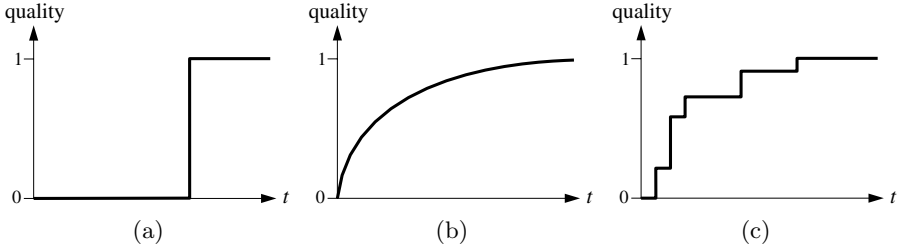


Fig. 1. Performance profiles indicate the quality of results as a function of computation time. A standard algorithm (a) provides results only after completion while (b) an idealized anytime algorithm provides results which continuously improve, although in practice (c) improvements come incrementally (after [12]).

Given expected utilities for each partial result, algorithms known as *deliberation schedulers* have been developed which optimally schedule the use of computational resources to maximize the expected utility of the result [13,12]. The scheduler used in [10] greedily rendered those scene components with the highest utility gain associated with them, that could be completed in the allotted time. When applied to cue integration, a deliberation scheduler must choose between the cues, to decide which will receive further computation.

2.2 Cue Integration

Vision systems must rely on a variety of visual cues to behave robustly, because individual cues do not supply accurate or reliable information in all situations. With the use of multiple visual cues comes the need of a method to *integrate* the results that come from each source. Such a method should improve the accuracy of estimates as well as reduce their uncertainties when compared to a method that uses only a single cue. In this section, I will briefly overview methods for cue integration. This work will build on these techniques, adapting them to allow for the use of partial computations.

Work on cue integration used for computer vision typically focuses on two interrelated problems. The first is the problem of integrating consistent yet noisy cues; the second is the selection of usable cues in those situations when they are corrupted or inconsistent. I take a probabilistic approach to these problems [14]. In the case of two cues, we have two sources of information $\mathbf{y}_1$ and $\mathbf{y}_2$ that result in two estimates $\hat{\mathbf{x}}_1$ and $\hat{\mathbf{x}}_2$ of an underlying state of the world $\mathbf{x}$ (parameters for a face, say). A reasonable assumption is that these sources are conditionally independent given $\mathbf{x}$ (this is, of course, a much weaker assumption than actually assuming the sources are independent). This corresponds to the Bayes net in Figure 2(a).

This graphical model specifies the problem of cue integration in terms of three parameters. $p(\mathbf{x})$ is the prior probability of different states of the world,

(a) (b)

Fig. 2. The state of the world $\mathbf{x}$ (a) determines the data sources $\mathbf{y}_1$ and $\mathbf{y}_2$ as described by this Bayes network, and (b) can be estimated as $\hat{\mathbf{x}}$ by statistically combining the estimates $\hat{\mathbf{x}}_1$ and $\hat{\mathbf{x}}_2$.

and $p(\mathbf{y}_1 \mid \mathbf{x})$ and $p(\mathbf{y}_2 \mid \mathbf{x})$ are the likelihoods of observing the cues as a function of the state of the world. By Bayes' theorem, the integration corresponds to multiplying the conditional probability distributions of each estimate:

$$p(\mathbf{x} \mid \mathbf{y}_1, \mathbf{y}_2) \propto p(\mathbf{y}_1 \mid \mathbf{x}) p(\mathbf{y}_2 \mid \mathbf{x}) p(\mathbf{x}) \tag{1}$$

The *combined estimate* $\hat{\mathbf{x}}$ is the maximum a posteriori estimate of $\mathbf{x}$ obtained from this distribution. Examples of frameworks for cue integration that are defined this way include Kalman filters [5,6] and physically-based frameworks [15, 16]. These probabilistic approaches are demonstrated pictorially in Figure 2(b), showing how the combined estimate $\hat{\mathbf{x}}$ weights together the individual results based on their reliabilities, and produces an estimate which has less uncertainty (depicted as an ellipsoid with its mean at its center) than using either source separately.

If given an accurate representation of the probability distributions of each estimate, this allows for failures in sensors and algorithms. Unfortunately, getting such a representation is typically infeasible, thus motivating us to address the second problem mentioned above—that of cue selection. Some approaches explicitly describe interactions between visual modules by modeling the flow and priority of information between a variety of modules to specify how various cues are combined [4,17,18]. Other methods use robust statistics [19,7] or voting approaches [8], which take advantage of consistency among those cues which have not failed in selecting between cues or alternative interpretations.

No aspect of current approaches for cue integration is geared towards time-critical implementations, which require all cue computations to be completed in real-time. As a result, today's real-time vision systems that use multiple cues must rely on simple properties of an image—allowing all computations to be completed in real-time. This is widespread in real-time systems for monitoring the activities of people. In these applications these cues only include image edges or "blob" parameters computed from color histograms, temporal image differences, or background subtractions [20,21,22,23,24].

3 Using Partial Results

This section describes my approach for extending cue integration methods. Existing methods cannot handle partial computations because they do not account for errors in partial results. This section explains why this causes difficulties and proposes a solution. For illustrative purposes, let's examine a simple example of fusing two cues as in Figure 2(b). To keep this discussion simple, assume that the uncertainty of the estimates that follow are well approximated as Gaussian (they will be depicted by an ellipsoid).

Suppose that using the first cue by itself produces an estimate of $\hat{\mathbf{x}}_1$. It involves a simple one-step computation, but results in a rather noisy estimate (a large uncertainty ellipsoid). The second cue produces an estimate of $\hat{\mathbf{x}}_2$. It involves an iterative solution of a non-linear equation, but is much less corrupted by noise, and has a much smaller uncertainty ellipsoid. A depiction of the result of these computations is given in Figure 2(b), along with their uncertainties. Keep in mind that there is a large discrepancy between the amount of computation time required for determining each result.

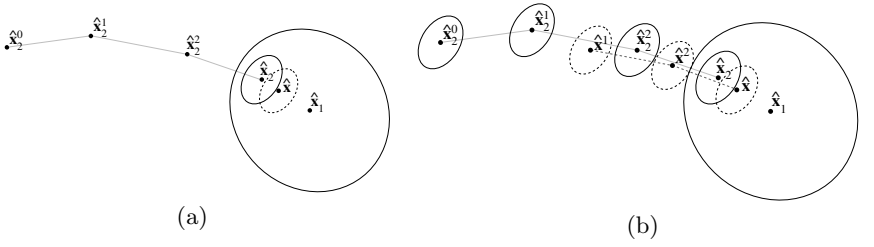


Fig. 3. With $\hat{\mathbf{x}}_1$ resulting from a one-step computation, and $\hat{\mathbf{x}}_2^i$ from an iterative algorithm (at step i), they can be combined at the very end in (a) late cue integration to produce $\hat{\mathbf{x}}$, or upon each iteration in (b) early cue integration to produce $\hat{\mathbf{x}}^i$ (which converges to $\hat{\mathbf{x}}$)

A late integration scheme [25] first computes the values of $\hat{\mathbf{x}}_1$ and $\hat{\mathbf{x}}_2$ (and their uncertainties), and then combines them statistically. (It is called late because the combination occurs late in the process.) In this case, $\hat{\mathbf{x}}_2$ is determined iteratively (in n iterations), producing the sequence $\left\{\hat{\mathbf{x}}_2^i \mid i \in 0 \ldots n\right\}$ where $\hat{\mathbf{x}}_2 := \hat{\mathbf{x}}_2^n$. This results in the fused estimate $\hat{\mathbf{x}}$, shown in Figure 3(a), which is obtained by weighting $\hat{\mathbf{x}}_1$ and $\hat{\mathbf{x}}_2$ together based on their uncertainties. An early integration scheme [25], such as an iterated extended Kalman filter, instead performs an integration upon each iteration. Figure 3(b) shows an example of such a computation. It combines $\hat{\mathbf{x}}_1$ with the current iterated estimate of $\hat{\mathbf{x}}_2$, until the combined estimate converges. In other words, $\hat{\mathbf{x}}_1$ is combined with $\hat{\mathbf{x}}_2^i$ to produce $\hat{\mathbf{x}}^i$. Generally, the iterates for $\hat{\mathbf{x}}_2$ will differ from those in the late integration scheme since $\hat{\mathbf{x}}^i$ is used as the next guess for $\hat{\mathbf{x}}_2$. This can often lead to faster convergence and stability, as richer information is used to form the next $\hat{\mathbf{x}}_2^i$. Additionally, depending on how sensitive the optimization problem is

to initial guesses, the converged value of $\hat{\mathbf{x}}$ may actually differ from that using late integration.

Typically, early integration is preferred, as more sources of information should help the optimization problem converge faster and perhaps more reliably (avoiding local minima), especially if certain computations (such as the one for $\hat{\mathbf{x}}_1$), require only a single iteration. However, there is a problem with the early integration computation in Figure 3(b), which becomes quite serious for when there is a large discrepancy between the rates of convergence for each cue. This is demonstrated by simply looking at the first iteration—which is equivalent to the situation that arises when there is only time for a single iteration to be computed—a partial computation. (This same situation arises had we interrupted the *late* integration computation of $\hat{\mathbf{x}}_2$).

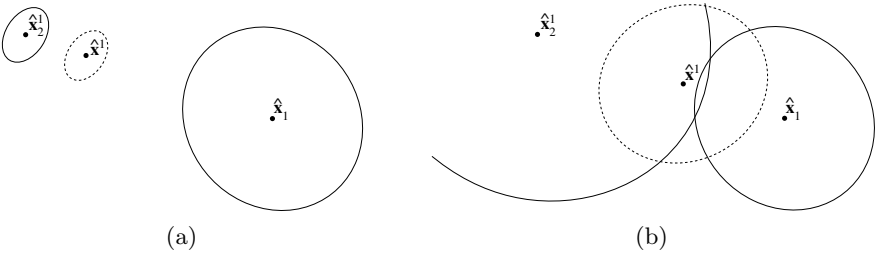


Fig. 4. (a) A serious problem with early cue integration is that the uncertainties do not represent that $\hat{\mathbf{x}}_2^1$ has not converged yet, producing an incorrect combined estimate of $\hat{\mathbf{x}}^1$. (b) A solution to this problem will appropriately inflate the uncertainty for $\hat{\mathbf{x}}_2^1$ *before* its combination.

Figure 4(a) shows what happens after a single iteration of the integration problems from above: $\hat{\mathbf{x}}_1$ is combined with $\hat{\mathbf{x}}_2^1$ to produce $\hat{\mathbf{x}}^1$. The uncertainty information for $\hat{\mathbf{x}}_2^1$ is being used inappropriately: the ellipsoid applies *only* to the truly converged value of $\hat{\mathbf{x}}_2$. (Actually, the parameters of the ellipsoid depend on $\hat{\mathbf{x}}_2^1$ and are also converging to their final values, but in a way that is only related to the non-linearity of the system, and not its convergence rate.) What is the consequence of using this uncertainty information this way? With luck, the result of this error could only be wasted computation, as it may still converge to the desired value (as it did in Figure 3(b)). In practice, however, such an error can lead to convergence to the wrong answer (a local minima), if early iterations are too distant from the actual answer.

A solution to this problem would be to inflate the uncertainty region for $\hat{\mathbf{x}}_2^1$ to account for the remaining error in the partial computation. This is shown in Figure 4(b). Such a solution would have two benefits. First, $\hat{\mathbf{x}}_2^1$ is now correctly influenced by $\hat{\mathbf{x}}_1$ (which is a complete computation). This can lead to faster convergence, as the value used in the next iteration will probably be closer to the convergence point. And second, $\hat{\mathbf{x}}^1$ has meaningful and useful uncertainty information associated with it, should there not be additional time for this com-

putation to continue. The next section formalizes this solution for the general case.

4 Probabilistic Models of Partial Results

Figure 4(b) illustrates a method for integrating partial results. This idea is formalized here by introducing a model of partial results. To start, we examine a method that uses completed computations, before moving on to partial results.

The estimate $\hat{\mathbf{x}}_c$ (of the state of the world $\mathbf{x}$) results from using visual cue c. Probabilistic approaches to estimation use models for noise introduced in the sensing process to describe the uncertainty in the estimate $\hat{\mathbf{x}}_c$ [26]. Maximum a posteriori estimation yields the state that maximizes the posterior $p(\mathbf{x}\,|\,\mathbf{y}_c)$ given the noisy source of information $\mathbf{y}_c$:

$$\hat{\mathbf{x}}_c = \operatorname{argmax}\, p(\mathbf{x} \mid \mathbf{y}_c) \tag{2}$$

We can think of $\mathbf{y}_c$ as resulting from a series of n computational steps (iterations of an optimization algorithm), $\mathbf{Y}_c^n := \left\{\mathbf{y}_c^i \mid i \in 0 \ldots n\right\}$, where $\mathbf{y}_c^0$ is the prior estimate (an initial guess), and the rest are based upon underlying data (perhaps an image). After n iterations, the computation is deemed complete (it converged to within a certain tolerance) so that $\mathbf{y}_c := \mathbf{y}_c^n$. The partial result $\mathbf{y}_c^i$ gives a *probability distribution* that results after i iterations of computation on cue c. For example, $\mathbf{y}_c^i$ would be described by the mean and covariance of a Gaussian distribution after i iterations of an iterated extended Kalman filter.

There are many sources of error that can exist in such estimates, including noise in the sensing process, sensor failures, violated assumptions about the world (assuming, for instance, that a light source is Lambertian in a shape from shading calculation), and computations resulting in inappropriate local minima (such as not finding the best alignment that maps model features to image features). These errors are modeled using the probability distribution $p(\mathbf{x} \mid \mathbf{y}_c)$, although robust statistical procedures [27,7] may be required for detection or exclusion of estimates that suffer from serious disruptions.

Fig. 5. (a) A model of errors in the partial results for cue c uses $\mathbf{e}_c^i$ to represent the deviation in the partial result $\mathbf{y}_c^i$ that will be eliminated by finishing the computation (which converges to $\mathbf{y}_c$). (b) This model is a probability distribution on convergence location that is a function of the partial results seen so far ($\mathbf{Y}_c^i$).

There is another source of error that is of interest here—errors in partial results (such as the deviation produced when an optimization is interrupted, that would have properly converged had enough time been available). While this source of error is not stochastic (since the actual deviation can be computed deterministically by finishing the computation), it is convenient to model it as follows:

$$\mathbf{y}_c = \mathbf{y}_c^i + \mathbf{e}_c^i \tag{3}$$

where the deviation from the convergence point that remains after i iterations is given by a random variable $\mathbf{e}_c^i$, as shown in Figure 5(a). The motivation for this assumption is the same as with the use of motion models in tracking [28]: while the motions of an observed object might be deterministic, they are modeled as stochastic, as a more detailed model requires unobservable information.

Another reasonable assumption is that this probability distribution is conditionally dependent on all of the computed iteration steps so far (after i iterations, that would be $\mathbf{y}_c^0$ through $\mathbf{y}_c^i$). An example of such a distribution is shown in Figure 5(b). While a Markovian assumption is also reasonable here, giving only conditional dependence on the most recent iteration (optimization procedures use only the latest iteration), the use of $\mathbf{e}_c^i$ in modeling the remaining deviation is only an approximation, and having the entire history provides helpful information to predict convergence. The probabilistic *model of partial results* at iteration i is written as:

$$p(\mathbf{y}_c \mid \mathbf{Y}_c^i) \quad i \in 0 \ldots n \tag{4}$$

which can be interpreted as the probability of converging to some configuration $\mathbf{y}_c$ having seen i iterations of computation. This model is used for estimation with partial results:

$$\hat{\mathbf{x}}_c^i = \operatorname{argmax} p(\mathbf{x} \mid \mathbf{Y}_c^i) \tag{5}$$

This is in contrast to (2), which required complete results. We can express $p(\mathbf{x} \mid \mathbf{Y}_c^i)$ in terms of the model of partial results (4) and the original posterior $p(\mathbf{x} \mid \mathbf{y}_c)$:

$$\begin{aligned} p(\mathbf{x} \mid \mathbf{Y}_c^i) &= \int p(\mathbf{x}, \mathbf{y}_c \mid \mathbf{Y}_c^i)\ d\mathbf{y}_c = \int p(\mathbf{y}_c \mid \mathbf{Y}_c^i) p(\mathbf{x} \mid \mathbf{y}_c, \mathbf{Y}_c^i)\ d\mathbf{y}_c \\ &= \int p(\mathbf{y}_c \mid \mathbf{Y}_c^i) p(\mathbf{x} \mid \mathbf{y}_c)\ d\mathbf{y}_c \end{aligned} \tag{6}$$

In the final manipulation, we reasonably assume that all dependence information in $p(\mathbf{x} \mid \mathbf{y}_c, \mathbf{Y}_c^i)$ on partial results is included in its dependence on $\mathbf{y}_c$. At this point, we can apply Bayes' theorem:

$$p(\mathbf{x} \mid \mathbf{Y}_c^i) \propto \int p(\mathbf{y}_c \mid \mathbf{Y}_c^i) p(\mathbf{y}_c \mid \mathbf{x}) p(\mathbf{x})\ d\mathbf{y}_c \tag{7}$$

to separate the model of noise and failures in sensing and computation $p(\mathbf{y}_c \mid \mathbf{x})$, from the priors $p(\mathbf{x})$.

Models are constructed by estimating their distributions by analyzing the data from typical runs (this data is gathered off-line so the computations are given time to fully converge). Similar learning approaches have been successfully used in meta-reasoning research [12], although in much lower dimensions (for a scalar measure of quality). We narrow our choice of models by taking into account one behavior this model must have. As the computation of a cue converges, the entropy (or average uncertainty) of the model should be decreasing, so that a completed computation is unaffected by the model. This rules out stochastic differential equations [28,29] that are often used as motion predictors, as they have constant covariance. The models developed in this section enable existing iterative vision algorithms to be used in the implemented system. The next description describes this implemented system, as well as its' specific model for partial computations.

5 Implementation

The domain of tracking a human face is used as a testbed for our system. This is constructed in part from a previously developed face tracking system [30], the relevant details of which follow.

5.1 Face Model

A three-dimensional face model is used to describe the shape, motion and appearance of faces in images. The model is formed by applying deformation functions to an underlying shape (a polygon mesh representing an average face). Within our face model, shape parameters describe an individual's appearance (there are about 80 shape parameters), while the motion parameters encode the location of their head, as well as their facial displays and expressions (there are 12 motion parameters). Figure 6 shows examples of the face model undergoing various shape deformations (showing four different individuals), motion deformations (showing brow raising and frowning, smiling, and mouth opening) and finally two examples of when several deformations are applied at once. Further detail about this model can be found in [30]. The next section describes the particular cue computations used by this system, to estimate shape and motion.

5.2 Cue Computations

Model-based optical flow. This computation is an iterative algorithm which solves a non-linear least-squares problem to determine the motion estimate of the face in terms of the motion parameters of the model. The specific optimization performed is a model-based version of the optical flow constraint equation [31], solved using the Gauss-Newton method. In other words, the resulting flow field is parameterized by the motion parameters of the face model (which include rigid head motion as well as expressions and facial displays).

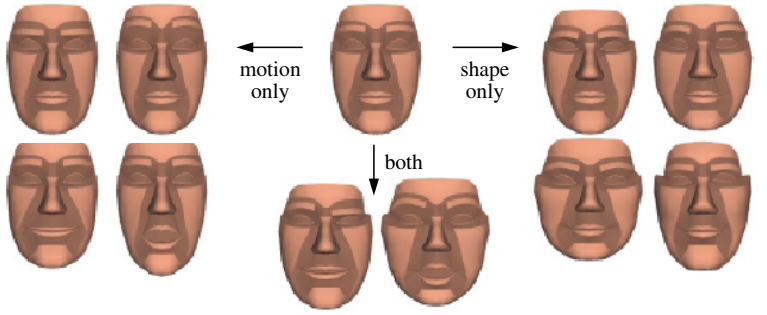


Fig. 6. Example parameterized deformations of the face model (with separate parameters for shape and motion)

Contour alignment. The parameters of the model are adjusted to improve the alignment between the predicted locations of contours and matched edges in the image [32,33,34]. This non-linear problem is solved using the Gauss-Newton method, and forms an estimate of the shape and motion parameters of the model.

Feature alignment. The face has a number of features which are reliably extracted, such as the corners or boundary curves of the lips, eyebrows and eyes. The computation used to determine the alignment between the predicted locations of features and their corresponding image locations is the same as that used for contours. This optimization problem is easier, as the features don't slide around the model (like contours do) when parameters are adjusted.

Each of the above computations results in an estimate of the model parameters represented by a Gaussian distribution. An ordinary approach to cue integration would simply use a Kalman filter to combine these estimates (perhaps upon each iteration). [30] investigated the use of constraints for cue integration, which provided a simple means to enslave one cue to another. This approach only made sense with two cues; the dominant cue was flow, which provided a constraint on the solution of contours and features (which were combined together into a single computation).

5.3 Implemented Model of Partial Results

Here I describe a specific instance of a model of partial results, built with these particular visual cues and associated computations in mind. The most concise model was chosen that is still sufficiently accurate for the application.

On the computation of these particular cues, the Gauss-Newton method is quite well-behaved. Specifically, the amount of change produced at the most recent iteration is a good indicator of the amount of convergence. As a first step in modeling, we put aside the model's explicit dependence on the current state, and instead use differences between iterations, as shown in Figure 5(b):

$$\mathbf{d}_c^i = \mathbf{y}_c^i - \mathbf{y}_c^{i-1} \quad i \in 1 \ldots n \tag{8}$$

Analysis of the example runs suggests that the amount of change at each iteration ($\left\|\mathbf{d}_c^i\right\|$) is good evidence for the amount of error that remains ($\left\|\mathbf{e}_c^i\right\|$), although the direction of $\mathbf{d}_c^i$ does not seem to indicate the direction of convergence. Given this, and the property of decreasing uncertainty mentioned above, I have chosen the model $p(\mathbf{y}_c \mid \left\|\mathbf{d}_c^i\right\|)$ described by (3) and:

$$\mathbf{e}_c^i = \left\|\mathbf{d}_c^i\right\| \mathbf{A}_c \mathbf{v} \tag{9}$$

where $\mathbf{v}$ is standard zero-mean Gaussian noise (the same dimension as $\mathbf{y}_c$) scaled appropriately by the square matrix $\mathbf{A}_c$. This model says that the error in a partial result has length roughly proportional to the iteration step size. The matrix $\mathbf{A}_c$ specifies the scaling of each component of $\mathbf{e}_c^i$ relative to $\left\|\mathbf{d}_c^i\right\|$, and is the maximum likelihood estimate using data gathered from example runs (in particular, each datum consists of $[\mathbf{d}_c^i, \mathbf{e}_c^i]$). These example runs are entirely distinct from those used for evaluation (in the next section). There is a separate model ($\mathbf{A}_c$) for each of the three cue algorithms mentioned above.

More complex models are possible (such as one which uses steps from additional iterations, or which uses $\mathbf{d}_c^i$ and not only its length), but require more data at the expense of the model having higher variance. More complex models did not show any significant improvement. Of course, with different optimization techniques or cues, alternative choices could have been made.

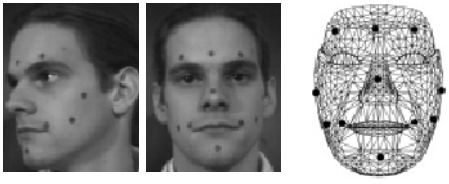
Using the model of partial results given by (9) within an iterated extended Kalman filter framework is quite straightforward. Since $\mathbf{e}_c^i$ is also Gaussian, the integral in (6) is easily solved in closed form [28]. This is used to inflate the estimated covariances upon each iteration.

6 Experiments

The use of the model of partial results was tested on the tracking sequence in Figure 8. To enable validation, markers were placed on the subject's face. The configuration of these markers, shown in Figure 7(a), was also determined with respect to the 3D face model, shown in (b). The computations on this sequence did not use pixels nearby these markers; this way, the presence of the markers had no effect on the estimates. The performance of the system on different image sequences was comparable.

Figure 9 shows the quantitative performance of seven different frameworks. This graph shows the average error in displacement (in pixels) between the actual marker locations in the image and that which the 3D face model predicted (as the marker locations on the 3D model were given). When cues were used in isolation, the systems lost track: the **flow** approach accumulated error until tracking failure as it only used velocity information, while **features** or **contours** alone were not robust enough to maintain track for this long sequence. Performance was better, although still failed when the two cues were combined using **late** integration. Without a model of partial results, **early** integration was also ineffective.

When the cues were combined early **using constraints**, the system tracked successfully, although lost accuracy when the flow estimate took longer to con-



(a) (b)

Fig. 7. Corresponding locations of markers on the subject (a) are determined on the 3D model (b)

verge when the subject moved rapidly (around frame 300). In our new system **using a model of partial results** the inaccuracies were eliminated—it maintained good track throughout the sequence.

Examination of the results indicate the model is indeed inflating the uncertainty of the estimates appropriately. Using the tracking sequence from Figure 8, the converged distributions (from the last iteration) are compared to those after the first iteration—with and without the model of partial results. This is demonstrated in Figure 10 using the cross-entropy H_{cross} between two distributions $p(\mathbf{x})$ and $q(\mathbf{x})$ [35], which can be used to measure "containment" of q in p (so that the most likely values in q are also reasonably likely in p). q is contained in p increasingly as $H_{\text{cross}}(p, q)$ becomes larger relative to $H_{\text{cross}}(q, p)$, as indicated by the "better containment" arrow in Figure 10(b). The quality of the "centering" of q within p is indicated by the "better alignment" arrow on this same plot; both cross-entropies decrease as centering improves. Looking to the data, the plot in Figure 10(a) indicates that without the model, the points are distributed along the diagonal, indicating that these distributions are in conflict (there is little containment in either direction). However, using the model to inflate the uncertainty after the first iteration enhances the containment, as seen in Figure 10(b). Even those points which are poorly aligned (which are distant from the origin) still exhibit this consistency.

The partial result framework is just as fast as the constraint framework (both took around 1 second per frame), and was significantly faster than early or late integration (which took over 4 seconds per frame). (Timing information was measured on a high end Sun Ultra 10.) This suggests that the appropriate weighting of computations reduces the number of iterations required. Unlike the previous constraint approach [30], the partial result framework additionally:

- yields consistently accurate results;
- can combine more than two cues (it is unclear if this can be done in [30]);
- enables real-time performance through the allocation of computational resources in a way that produces the best results in the allotted time, and in a way that allows for early termination of computations.

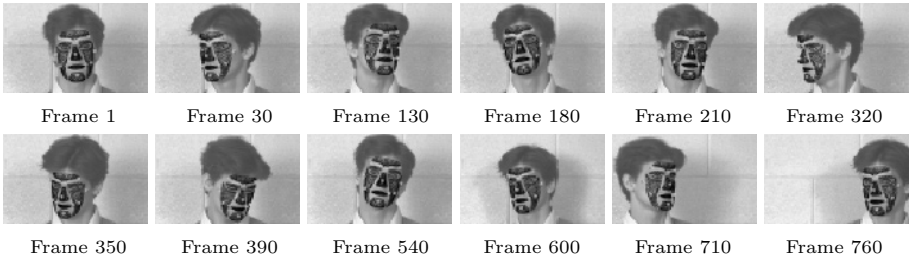


Fig. 8. Results from the face tracking validation experiment.

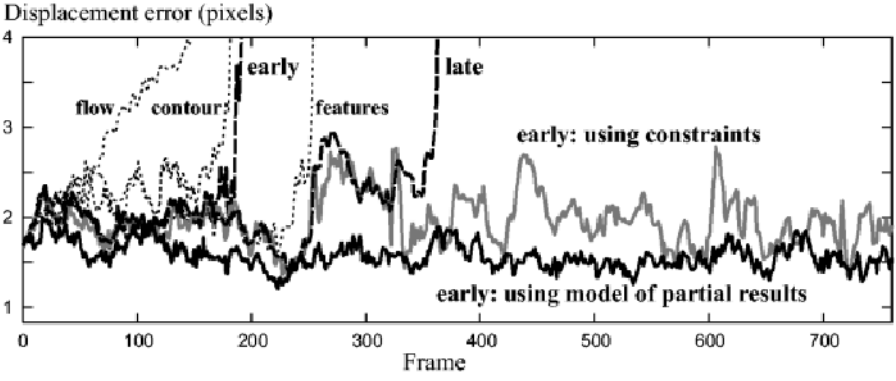


Fig. 9. Tracking performance of various frameworks (cues in isolation, and also integrated using different methods).

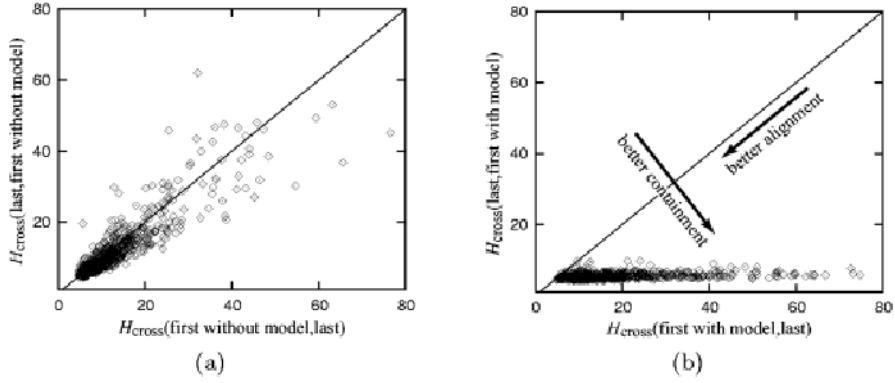


Fig. 10. Model evaluation: cross-entropy scatter plots enable comparison of the converged probability distributions with those after the first iteration. The distributions from an ordinary early integration approach (a) are often in conflict, while the use of a model of partial results (b) establishes consistency.

7 Conclusions and Further Investigations

This paper represents a significant step towards a real-time vision framework using multiple cues. A model of partial results enables early integration and

provides meaningful estimates even when interrupted. Using this idea in real-time implementations requires careful management of resources to get the most out of using each cue: meta-reasoning.

The next step is to develop a deliberation scheduler suitable for cue integration. To optimize the scheduling, meta-reasoning approaches use *expected utilities* [13] from each computation being considered to make the best use of resources. In other words, the scheduler attempts to predict which computation would provide the greatest benefit. In most meta-reasoning research [13,12], utilities of inputs are simply combined additively to determine the utility of the output. This design falls short for cue integration and requires alterations, as the contribution to the utility from one cue depends on the posteriors of all the other cues.

Acknowledgments. Thanks to Robert Jacobs, Peter Meer, Rick Wildes and Matthew Stone. Partially supported by NSF Instrumentation 9818322.

References

1. Dosher, B., Sperling, G., Wurst, S.: Tradeoffs between stereopsis and proximity luminance covariance as determinants of perceived 3D structure. Vision Research **26** (1986) 973–990
2. Jacobs, R., Fine, I.: Experience-dependent integration of texture and motion cues to depth. Vision Research **39** (1999) 4062–4075
3. Johnston, E.B., Cumming, B.G., Landy, M.: Integration of motion and steropsis cues. Vision Research **34** (1994) 2259–2275
4. Aloimonos, Y., Shulman, D.: Integration of Visual Modules: An Extension of the Marr Paradigm. Academic Press (1989)
5. Ayache, N., Faugeras, O.: Building, registering, and fusing noisy visual maps. International Journal of Robotics Research **7** (1988) 45–65
6. Kriegman, D., Triendl, E., Binford, T.: Stereo vision and navigation in buildings for mobile robots. IEEE Transactions on Robotics and Automation **5** (1989) 792–803
7. McKendall, R., Mintz, M.: Data fusion techniques using robust statistics. In Abidi, M., Gonzalez, R., eds.: Data Fusion in Robotics and Machine Intelligence. (1992) 211–244
8. Brautigam, C., Eklundh, J., Christensen, H.: A model-free voting approach for integrating multiple cues. In: ECCV '98. (1998) 734–750
9. Dean, T., Kaelbling, L., Kirman, J., Nicholson, A.: Planning under time constraints in stochastic domains. AI **76** (1995) 35–74
10. Horvitz, E., Lengyel, J.: Perception, attention, and resources: A decision-theoretic approach to graphics rendering. In: UAI '97. (1997) 238–249
11. Dean, T., Boddy, M.: An analysis of time-dependent planning. In: AAAI '88. (1988) 49–54
12. Zilberstein, S., Russell, S.: Optimal composition of real-time systems. AI **82** (1996) 181–213
13. Horvitz, E.: Reasoning under varying and uncertain resource constraints. In: AAAI '88. (1988) 111–116

14. Bulthoff, H., Yuille, A.: A bayesian framework for the integration of visual modules. In Inui, T., McClelland, J., eds.: Attention and Performance XVI: Information Integration in Perception and Communication. (1996) 49–70
15. Terzopoulos, D.: Physically-based fusion of visual data over space, time and scale. In Aggarwal, J., ed.: Multisensor Fusion for Computer Vision. Springer-Verlag (1993) 63–69
16. Zhang, G., Wallace, A.: Physical modeling and combination of range and intensity edge data. CVGIP **58** (1993) 191–220
17. Das, S., Ahuja, N.: Performance analysis of stereo, vergence, and focus as depth cues for active vision. PAMI **17** (1995) 1213–1219
18. Pankanti, S., Jain, A.: Integrating vision modules: Stereo, shading, grouping, and line labeling. PAMI **17** (1995) 831–842
19. Cho, K., Meer, P.: Image segmentation from consensus information. CVIU **68** (1997) 72–89
20. Azoz, Y., Devi, L., Sharma, R.: Reliable tracking of human arm dynamics by multiple cue integration and constraint fusion. In: CVPR '98. (1998) 905–910
21. Darrell, T., Gordon, G., Harville, M., Woodfill, J.: Integrated person tracking using stereo, color, and pattern detection. IJCV **37** (2000) 175–185
22. Graf, H., Cosatto, E., Gibbon, D., Kocheisen, M., Petajan, E.: Multi-modal system for locating heads and faces. In: AFGR '96. (1996) 88–93
23. Rasmussen, C., Hager, G.: Joint probabilistic techniques for tracking multi-part objects. In: CVPR '98. (1998) 16–21
24. Toyama, K., Horvitz, E.: Bayesian modality fusion: Probabilistic integration of multiple vision algorithms for head tracking. In: Fourth Asian Conference on Computer Vision. (2000)
25. Hennecke, M., Stork, D., Prasad, K.: Visionary speech: Looking ahead to practical speechreading system. In Stork, D., Hennecke, M., eds.: Speechreading by Humans and Machines: Models, Systems, and Applications (NATO ASI Series F: Computer and Systems Sciences, volume 150). (1996) 331–350
26. Szeliski, R.: Bayesian modeling of uncertainty in low-level vision. IJCV **5** (1990) 271–302
27. Meer, P., Mintz, D., Kim, D., Rosenfeld, A.: Robust regression methods for computer vision: A review. IJCV **6** (1991) 59–70
28. Maybeck, P.: Stochastic Models, Estimation and Control, Volume 1. Academic Press (1979)
29. Reynard, D., Wildenberg, A., Blake, A., Marchant, J.: Learning dynamics of complex motions from image sequences. In: ECCV '96. (1996) I:357–368
30. DeCarlo, D., Metaxas, D.: Optical flow constraints on deformable models with applications to face tracking. IJCV **32** (2000) 99–127
31. Bergen, J., Anandan, P., Hanna, K., Hingorani, R.: Hierarchical model-based motion estimation. In: ECCV '92. (1992) 237–252
32. Lowe, D.: Fitting parameterized three-dimensional models to images. PAMI **13** (1991) 441–450
33. Terzopoulos, D., Witkin, A., Kass, M.: Constraints on deformable models: Recovering 3D shape and nonrigid motion. AI **36** (1988) 91–123
34. Yuille, A., Cohen, D., Halliman, P.: Feature extraction from faces using deformable templates. IJCV **8** (1992) 104–109
35. Bishop, C.: Neural Networks for Pattern Recognition. Oxford University Press (1995)