Setting Low-Level Vision Parameters

Adrian Broadhurst and Simon Baker

CMU-RI-TR-04-202

Setting Low-Level Vision Parameters

CMU-RI-TR-04-20

Adrian Broadhurst and Simon Baker

The Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

Abstract

As vision systems become more and more complex there is an increasing need to understand the interaction between the various modules that these systems are composed of. In this paper we attempt to answer the question of how a high-level module can feed back its knowledge to a low-level module to improve the performance of the overall system. In particular we consider a system model consisting of a single low-level module that takes a set of low-level parameters as input and a single high-level module that estimates a set of high-level model parameters. We consider the task of setting the low-level parameters to maximize the performance of the overall system. Previous approaches to this problem include setting the parameters by hand, empirical evaluation, learning, and updating the parameters using the previous image in a video. We propose an approach based on simultaneous optimization of the high-level and low-level parameters. After outlining the approach, we demonstrate it on three examples: (1) color-blob tracking, (2) color-based lane tracking, and (3) edge-based lane tracking.

1 Introduction

Vision systems are becoming more and more complex. As this trend continues, there is an increasing need to understand the interaction between the various modules that are present in the system. How does a higher-level module feed back its knowledge to improve the performance of a lower-level module? How are the results of multiple modules integrated? With a few exceptions such as [1], there has been little study of these kind of questions since the pioneering work of [4].

In this paper we study the interaction of high-level and low-level vision modules. In particular, we attempt to answer the question of how a high-level module can feed back its knowledge to a low-level module to improve the performance of the overall system. Although this question is just one of many, and our system model is relatively simple, this paper represents an important first step towards developing a framework for the study of large, complex vision systems.

Our system model is illustrated in Figure 1. We assume that the system consists of two modules, a low-level module and a high-level module. The low-level module takes the input image and a set of low-level parameters and estimates a representation of the scene which we call the primal sketch, after [4]. The high-level module takes the primal sketch and computes the parameters of a high-level scene model. We assume that the high-level module operates by optimizing a model fitting error function with respect to the unknown high-level model parameters.

An illustrative example of such a system is a color-blob face tracker such as [8]. (We give a particularly simple example to show the benefits of the framework.) The low-level module is a color pixel classifier. The low-level parameters are the parameters of the color model; i.e. the definition of what is "face color". The primal sketch is the (possibly soft) pixel classification. The high-level model is a geometric model of the face such as an ellipse. The high-level module operates by fitting the ellipse to the classified face-color pixels. The high-level parameters are the parameters of the ellipse. Although this example is perhaps the simplest that could be thought of, it is illustrative of a wide range of more complex vision systems.

Given the system model in Figure 1, the specific question we are interested in is how to set the low-level vision parameters to maximize the performance of the complete system. The perfor-

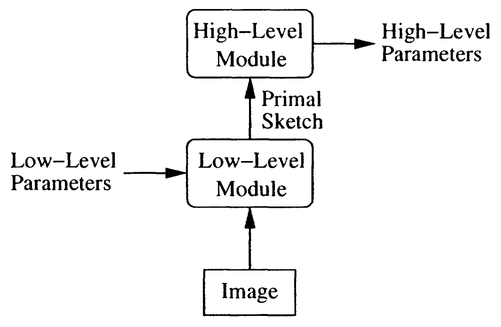


Figure 1: System Model. We assume that the vision system can be modeled as a low-level module and a high-level module. The low-level module takes the input image and a set of low-level parameters and outputs a representation of the image which we refer to as the primal sketch after [4]. The high-level module takes the primal sketch and estimates the high-level parameters.

mance of a color-blob tracker is very dependent on the parameters of the color model used. For example, as the illumination changes, and people with different skin colors are tracked with the system, the low-level skin color parameters may need to change. How do we use the knowledge embedded in the high-level module to help set the low-level parameters to maximize the performance of both the low-level module and the overall system?

A variety of approaches have been proposed for setting the low-level parameters. The most naive approach is to set the parameters *by hand,* using a trial and error approach to find a set of parameters that work well in practice. A more rigorous approach is to perform an *empirical evaluation* to find the best set of parameters for a particular application [7]. A closely related approach is to *learn* the low-level parameters [2]. Another approach that is commonly used in tracking applications is to *update* the low-level parameters from frame to frame based on the results in the previous frame [5, 6, 10].

In this paper we propose setting the low-level parameters by simultaneously optimizing the high-level model fitting error over the high-level and low-level parameters. Posing the problem in this way has number of advantages: (1) One criticism of the previous approaches (by hand, evaluation-based, learning, updating) is that the low-level parameters used for any particular input image do not depend on that image. The closest related approach is the updating approach in which the low-level parameters depend on the previous image (although not the current image.) On the other hand, with the simultaneous optimization approach the low-level parameters are implicit functions of the input image. (2) In the process of performing the simultaneous optimization, information is fed back from the high-level module to the low level module. (3) The simultaneous optimization approach provides a rigorous basis for many ad-hoc low-level parameter update algorithms used in the tracking literature. (4) Posing the problem as simultaneous optimization makes it easier to understand why the system works well or fails. Reasoning about simultaneous optimization is far easier than reasoning about the complex interactions between two modules. (5) The generality of the formulation leads to a variety of implementations including low-level parameter update strategies, parameter re-initialization strategies, and multi-hypothesis strategies.

The remainder of this paper is organized as follows. We begin in Section 2 by describing our framework. We proceed in Sections 3-5 to give three case studies of applying our framework. We end in Section 6 with a discussion.

2 Framework

2.1 System Model

We assume that the vision system under consideration can be modeled as in Figure 1. The model consists of a single low-level module and a single high-level module. Although many vision systems are more complex than Figure 1, if we can understand how to feed back information from the high-level module to the low-level module in Figure 1, the same techniques will likely be useful in more complex systems containing more modules and more layers.

The low-level module takes the input image $\mathbf{I}$ and a set of low-level parameters $\mathbf{llp}$. The output is an arbitrary representation of the input image which which we call the *primal sketch* ($\mathbf{PS}$), using the terminology of [4]. We denote the operation of the low-level module LLV as:

$$\mathbf{PS} = \mathrm{LLV}(\mathbf{I}; \mathbf{llp}). \tag{1}$$

The high-level module takes the primal sketch and computes the parameters of a high-level model.

We assume that this process can be formulated as a optimization problem:

$$\arg\min_{\mathbf{hlp}} \left[\mathrm{HLV}(\mathbf{PS}; \mathbf{hlp}) \right] \tag{2}$$

where $\mathrm{HLV}(\mathbf{PS}; \mathbf{hlp})$ measures the fit between the primal sketch $\mathbf{PS}$ and the high-level model.

Many computer vision systems match this simple model. The low-level module is a process such as edge detection, segmentation, or stereo, that estimates a (generally 3D, physically based) representation of the scene, the primal sketch [4]. The high-level module is a process that fits a (generally high-level, semantic, data driven) parametric model to the primal sketch to "understand" the scene by means of estimating a small number of high-level parameters. These parameters can then be fed into a classifier if a discrete interpretation of the scene is required.

Due to the relatively immature state of computer vision, much of the research literature is concerned with either: (1) a single low-level process such as edge detection, stereo, or shape-from-x, or (2) a single high-level modeling or recognition task that operates *directly* on the image as the primal sketch, for example eigenfaces [9] and other "appearance based" algorithm. There are relatively few research papers that discuss the combination of low-level and high-level processing. A few illustrative examples of systems that do have both a low-level and a high-level module are included below. Note, however, that due to the focused nature of most research, these "systems" are composed of relatively simple low-level and high-level components. Even so, these examples are sufficient to illustrate the main points of this paper.

Color-Blob Face Tracking: The low-level module is a color segmentation algorithm that maps pixels onto the likelihood that they are "face pixels" based on the low-level parameters of the color model. The high-level module is an algorithm that fits a geometric model (e.g. an ellipse) to the likelihood image. The high-level parameters are the geometric parameters.

Color-Based Lane Tracking: The low-level module is a color segmentation algorithm that maps pixels onto the likelihood that they are "road pixels" based on the low-level parameters of the color model. The high-level module is an algorithm that fits a geometric model of the road to the road likelihood image. The high-level parameters are the geometric model parameters.

Edge-Based Lane Tracking: The low-level module is an edge detector, the parameters of which are the gradient thresholds. The high-level module is an algorithm which fits a lane model to the edge map. The high-level parameters are the parameters of the lane model.

2.2 Setting the Low-Level Parameters

The only way the high-level module in Figure 1 can feed back information to the low-level module is by changing the low-level parameters. The question of how the low-level parameters are set and updated therefore becomes the key question. We now describe some of the methods that have been used in the literature. Note, however, that in the first three of these methods the low-level parameters, once set, are constant. The mechanism by which the high-level module can feed back information is therefore blocked.

2.2.1 Previous Approach 1: By Hand

Perhaps the most common (and ad-hoc) way to set low-level parameters is by hand. A trial and error approach is used to find a set of parameters that work well in practice for the (type of) images that the system is being tested on.

2.2.2 Previous Approach 2: Empirical Evaluation

A more principled approach is to perform an empirical evaluation for a variety of parameter settings and choose the best ones. This approach is often used for edge detectors. Ideally the evaluation should be performed based on the performance of the high-level module, as in [7].

2.2.3 Previous Approach 3: Learning

A related approach is to "learn" the best low-level parameters [2]. In this case, a large number of training examples are given in the form of pairs $(\mathbf{I}_i, \mathbf{PS}_i)$ where $\mathbf{PS}_i$ is the primal sketch for

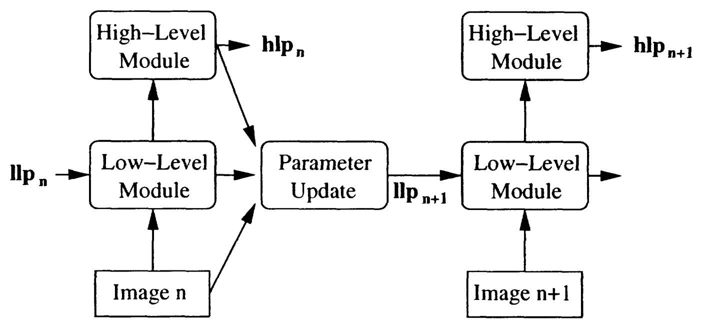


Figure 2: System diagram for the "updating" approach to setting low-level vision parameters (for videos.) The results for image n are used to update the low-level parameters for image $n + 1$.

the image $\mathbf{I}_i$. The optimal settings for the low-level parameters **llp** are then found using some sort of optimization process. The main use for this kind of approach is when the image formation process from primal sketch to input image is not invertible, or the inverse process ill-conditioned. In essence a data-driven prior is computed and coded using the low-level parameters.

2.2.4 Previous Approach 4: Updating

Another approach that is commonly used in video applications is to update the parameters from frame to frame based on the results on the previous image in the sequence. See, for example, [6,10] for two examples of color model update strategies, and [5] for an algorithm to update the entire template in a template tracking algorithm. The updating approach is illustrated in Figure 2. The information for image n (the high-level parameters, the low-level parameters, and the input image) are combined somehow to update the low-level parameters to make them suitable for image $n + 1$.

2.3 Simultaneous Optimization

A major criticism of all of the approaches listed above is that the low-level parameters do not depend on the input image (although in the "updating" approach they do depend on the previous image in the video.) Therefore, no information is passed from the high-level module back down to the low-level module. In this paper we propose "simultaneous optimization" as a framework for performing this feedback. We first present the general idea and then in the next section describe several specific implementations of the framework. The basic idea is conceptually very simple: set the low-level parameters by treating the overall system as optimizing:

$$\arg\min_{\mathbf{llp},\mathbf{hlp}} \left[\, \mathrm{HLV}(\mathrm{LLV}(\mathbf{I};\mathbf{llp});\mathbf{hlp}) \,\right] \tag{3}$$

simultaneously with respect to the low-level parameters **llp** and the high-level parameters **hlp**. Because the same optimization criterion is optimized simultaneously, the process enforces constraints between the two sets of parameters and feeds back information from the high-level to the low-level.

2.4 Implementations of the Framework

Equation (3) is a framework for setting the low-level parameters. It does not imply that the code for the two modules must be integrated. (It is often best if it is not.) There are various ways to implement the framework. These different methods correspond to the various ways in which the optimization can be performed: gradient descent, brute force combinatorial search, random sampling, etc.

2.4.1 Simultaneous Optimization

The most direct way to solve Equation (3) is as a simultaneous optimization over both the low-level and high-level parameters. One disadvantage of this approach is that it means that the low-level and high-level modules are in effect integrated into one. This reduces the modularity of the overall system which can make building it far more difficult.

2.4.2 Iterative Optimization

Another way to perform a gradient descent optimization with two sets of parameters is to optimize with respect to each set iteratively. This approach leads to the system diagram in Figure 3. First the low-level module is evaluated to compute $\mathrm{LLV}(\mathbf{I};\mathbf{llp})$ using an initial estimate of the low-level

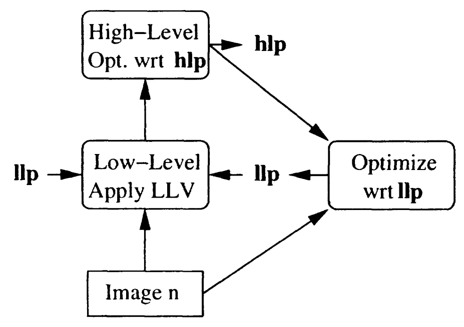


Figure 3: The iterative optimization implementation. After the low-level model has been evaluated using initial estimates of the low-level parameters, the high-level and low-level parameters are iteratively optimized until they converge. In the process the high-level information is fed back to the low-level module/parameters.

parameters. Next, the high-level model is fit using:

$$\arg\min_{\mathbf{hlp}} \left[\mathrm{HLV}(\mathrm{LLV}(\mathbf{I};\mathbf{llp});\mathbf{hlp}) \right]. \tag{4}$$

Then the low-level parameters are optimized via:

$$\arg\min_{\mathbf{llp}} \left[\mathrm{HLV}(\mathrm{LLV}(\mathbf{I};\mathbf{llp});\mathbf{hlp}) \right]. \tag{5}$$

These last two steps are then iterated until convergence. The advantage of this approach is that implementors of the high-level module can concentrate on optimizing Equation (4) and the implementors of the low-level module Equation (5).

2.4.3 Updating the Parameters in Video

If the input is a video sequence, a variant of the iterative optimization approach is to update the low-level parameters only between frames in the video (i.e. for each frame, run high-level fitting until convergence, then run low-level fitting until convergence.) This approach is illustrated in Figure 2 with the "parameter update" module performed using Equation (5); i.e. the "optimize with respect to **llp**" module.

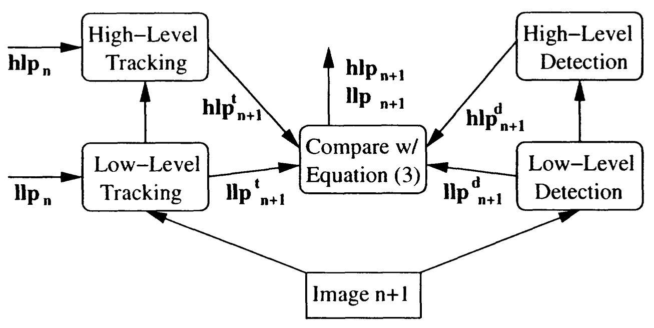


Figure 4: Combining detection and tracking. The detection and tracking processes are run in parallel and the results compared using Equation (3). The better set of parameters are the fi nal output.

2.4.4 Combining Detection and Tracking

Our framework also provides a rigorous way to integrate detection and tracking. By tracking, we mean any process that optimizes the quantity in Equation (3) at time $n + 1$ using any form of gradient descent, starting from the result at time n. By detection, we mean any process that optimizes the quantity in Equation (3) independently at each time using any form of brute-force search, combinatorial search, or random sampling. If the detection and tracking processing are run in parallel their outputs can be compared using Equation (3). The parameters which give the better value in Equation (3) are then chosen as the final result. This combination of detection and tracking is illustrated in Figure 4.

2.4.5 Multi-Hypothesis Detection and Tracking

The combination of detection and tracking can be extended to work with multiple hypotheses. The only difference is that there are then multiple sets of parameters that need to be tracked from time n and the detector must output multiple hypotheses. These hypotheses are then compared using Equation (3) and some combination of the best few hypotheses used as the final output. For example, the best single hypothesis could be used as the output. Another possibility is to use the average of the best few hypotheses. When there are a very large number of hypotheses, this approach is very similar to particle filtering or the CONDENSATION algorithm [3].

3 Color-Blob Face Tracking

Because of its speed, color-blob face tracking is a very commonly used technique, and has been studied by a variety of authors. Two early examples are [8, 10]. We now show how this problem can be mapped onto the system model in Figure 1. We then demonstrate using simultaneous optimization to perform the feedback from the high-level module to the low-level module. In particular, we show how our tracker can operate without a prior face color model. Instead, the low-level face color parameters are determined from the input image. Because the face color is determined in this way, our tracker is automatically robust to both continuous and discrete changes in face color and do not require special purpose color adaptation models such as the one in [10].

Note that our color-blob face tracker was implemented after we had already developed the framework and was built to fit the framework as well as possible. To demonstrate the versatility of our framework, the two other case studies in Sections 4 and 5 consider systems that had already been developed long before we conceived of the framework.

3.1 Mapping onto the System Model

Color-blob face tracking is mapped onto the system model of Figure 1 in the following way. The primal sketch $\mathbf{PS}$ is a soft labeling of the pixels as either face pixels or background pixels. Specifically, the primal sketch is a vector of two images $\mathbf{PS} = (\mathrm{PS}_{\mathbf{F}}(\mathbf{x}), \mathrm{PS}_{\mathbf{B}}(\mathbf{x}))$, the same size and shape as the input color image $\mathbf{I}(\mathbf{x}) = (R(\mathbf{x}), G(\mathbf{x}), B(\mathbf{x}))$. The first component is the probability a pixel is a face pixel:

$$\mathrm{PS}_{\mathbf{F}}(\mathbf{x}) = k_F e^{-(\mathbf{NC}(\mathbf{I}(\mathbf{x}))-\boldsymbol{\mu}_{\mathbf{F}})Q_{\mathbf{F}}(\mathbf{NC}(\mathbf{I}(\mathbf{x}))-\boldsymbol{\mu}_{\mathbf{F}})^{\mathrm{T}}} \tag{6}$$

where k_F is a constant and $\mathbf{NC}(\mathbf{I}(\mathbf{x}))$ is the normalized color:

$$\mathbf{NC}(\mathbf{I}(\mathbf{x})) = \frac{1}{\sqrt{R(\mathbf{x})^2 + G(\mathbf{x})^2 + B(\mathbf{x})^2}} \begin{bmatrix} R(\mathbf{x}) \\ B(\mathbf{x}) \end{bmatrix}. \tag{7}$$

In this equation, $\boldsymbol{\mu}_{\mathrm{F}}$ is the (2-parameter) mean normalized color, and Q_{F} is a (3-parameter) normalized color covariance matrix. Similarly, the second component is the probability that a pixel is a background pixel:

$$\mathrm{PS_B}(\mathbf{x}) \;=\; k_B e^{-\left(\mathrm{NC}(\mathrm{I}(\mathbf{x}))-\boldsymbol{\mu}_{\mathrm{B}}\right)Q_{\mathrm{B}}\left(\mathrm{NC}(\mathrm{I}(\mathbf{x}))-\boldsymbol{\mu}_{\mathrm{B}}\right)^{\mathrm{T}}} \tag{8}$$

where k_B is a constant, $\boldsymbol{\mu}_{\mathrm{B}}$ is the (2-parameter) mean normalized color, and Q_{B} is a (3-parameter) normalized color covariance matrix. In our experiments, the background parameters ($\boldsymbol{\mu}_{\mathrm{B}}$, Q_{B}) are learnt using a single training image and then fixed. The 5 parameters of the face color model ($\boldsymbol{\mu}_{\mathrm{F}}$, Q_{F}) are the low-level parameters **llp**.

The high-level model of the face is an ellipse specified by 5 geometry parameters, the center of the ellipse $\boldsymbol{\mu}_{\mathrm{G}}$ (2-parameters), and a covariance matrix Q_{G} (3-parameters). The high-level module maximizes the function:

$$\mathrm{HLV} = \prod_{\mathrm{dist}(\mathbf{x})\leq 1} \frac{\mathrm{PS_F}(\mathbf{x})}{\mathrm{PS_B}(\mathbf{x})+\mathrm{PS_F}(\mathbf{x})} \times \prod_{\mathrm{dist}(\mathbf{x})>1} \frac{1}{\mathrm{dist}(\mathbf{x})}\frac{\mathrm{PS_B}(\mathbf{x})}{\mathrm{PS_B}(\mathbf{x})+\mathrm{PS_F}(\mathbf{x})} \tag{9}$$

where: $\mathrm{dist}(\mathbf{x}) \;=\; (\mathbf{x}-\boldsymbol{\mu}_{\mathrm{G}})Q_{\mathrm{G}}(\mathbf{x}-\boldsymbol{\mu}_{\mathrm{G}})^{\mathrm{T}}$ is a Mahalanobis distance from the center of the ellipse. Informally, Equation (9) specifies the "probability" that all pixels inside the ellipse are face pixels (and not background pixels), and all of the pixels outside the ellipse are background pixels (and not face pixels). The background pixels are weighted with there distance from the ellipse. The high-level parameters **hlp** are the 5 ellipse parameters ($\boldsymbol{\mu}_{\mathrm{G}}$, Q_{G}).

3.2 Optimization Algorithms

The high-level optimization criterion HLV is a function of the 5 high-level parameters ($\boldsymbol{\mu}_{\mathrm{G}}$, Q_{G}), and indirectly a function of the 5 low-level parameters ($\boldsymbol{\mu}_{\mathrm{F}}$, Q_{F}). Although, for efficiency, ad-hoc optimization algorithms are usually used in color-blob tracker [8, 10], we implemented a full Gauss-Newton optimization algorithm to illustrate the power of our framework. Two variants of the algorithm were implemented. The first is a simultaneous optimization over all 10 parameters, with a 10×10 Hessian matrix. The second variant consists of separate optimizations over the high-level and low-level parameters. The high-level optimization is an optimization over the 5 high-level parameters, with a 5×5 Hessian. The low-level optimization is an optimization over the 5 low-level parameters, with a different 5×5 Hessian.

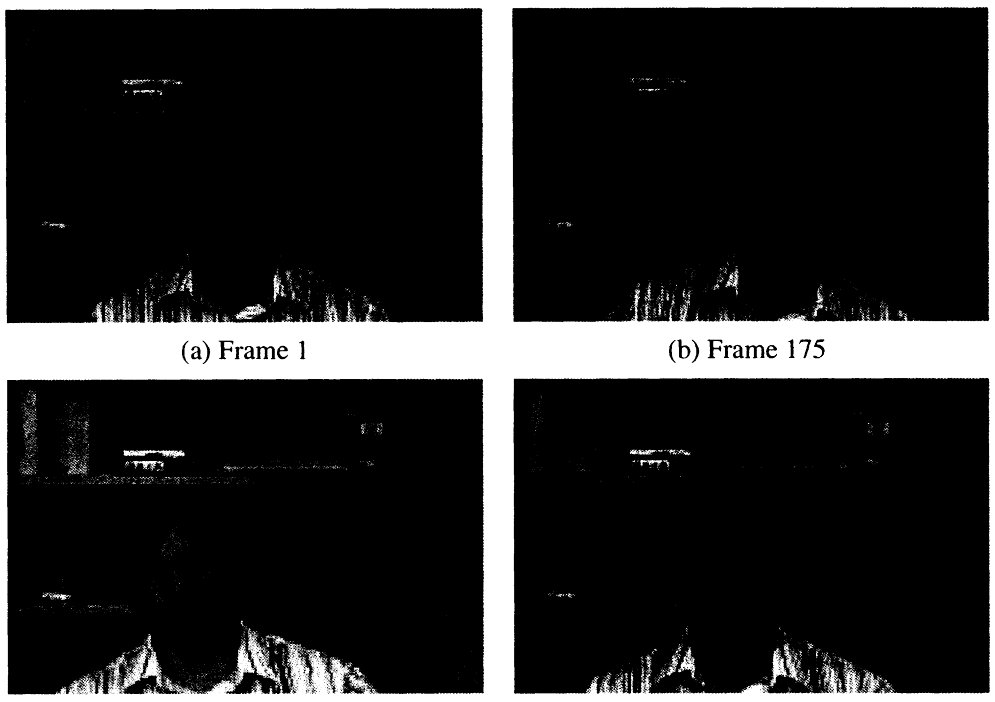
In addition to these gradient descent (tracking) algorithms, we also implemented a random-sampling based detection algorithm. This algorithm operates by randomly selecting multiple possible rectangles for the face region. From these regions, an estimate is made of the mean face color and covariance matrix. The background parameters are estimated as the mean and covariance of all of the pixels in the image. Given the low-level model, the pixels are then segmented as being either foreground or background depending on which is more likely given the color models. Connected components is then applied and the largest foreground component used to estimate the parameters of the face ellipse. The most promising estimates are then refined using the Gauss-Newton gradient descent algorithm.

Naturally, if the randomly selected face region is incorrect, the algorithm fails and a poor value is given for the optimality criterion in Equation (9). But, after enough tries the random-sampling algorithm will hit the face region and a good fit will be found for Equation (9).

3.3 Experiment 1: Continuous Color Change

We now demonstrate our tracker on a sequence with very large, but continuous, color change. The main point to note is that our tracker does not require a prior face color model; i.e. there is no initial estimate of the parameters of the color model $\boldsymbol{\mu}_{\mathrm{F}}$ and Q_{F}. Instead the color model is implicitly computed from the information fed back from the high-level module. The large color change in the input therefore does not pose a problem for our algorithm since effectively the color model is estimated separately for each input image.

To generate a sequence that would push our tracking algorithm to the limit we captured a sequence while manually changing the color balance setting on the camera. 4 frames from the 600


(a) Frame 1 (b) Frame 175

(c) Frame 455 (d) Frame 592

Figure 5: 4 frames from the 600 frame continuous color change sequence. The sequence was generated by manually changing the color balance on the camera during capture. To clearly see the color change, this fi gure must be viewed or printed in color.

frame sequence are shown in Figure 5. Notice how the color of the image changes from normal (a) to blue-ish (b) to pink-ish (c) and then back to normal (d).

We compared 4 different algorithms: (1) Simultaneous: Gauss-Newton gradient descent over all 10 parameters, (2) Iterative: an algorithm which iteratively performs separate Gauss-Newton gradient descents over the 5 high-level and 5 low-level parameters until they both converge, (3) Updating: an algorithm which performs the high-level Gauss-Newton gradient descent until convergence for each frame, and then performs the 5 parameter low-level Gauss-Newton gradient descent between frames to "update" the parameters, and (4) Fixed Low-Level Parameters (Fixed LLP): an algorithm that performs the Simultaneous algorithm on the first frame to initialize the color model, and from then on only performs the high-level 5 parameter Gauss-Newton gradient descent for each frame. The results for 3 frames for each of the 4 algorithms are included in Figure 6. The input image is overlayed with the pixels that are classified as most likely face pixels highlighted in green,

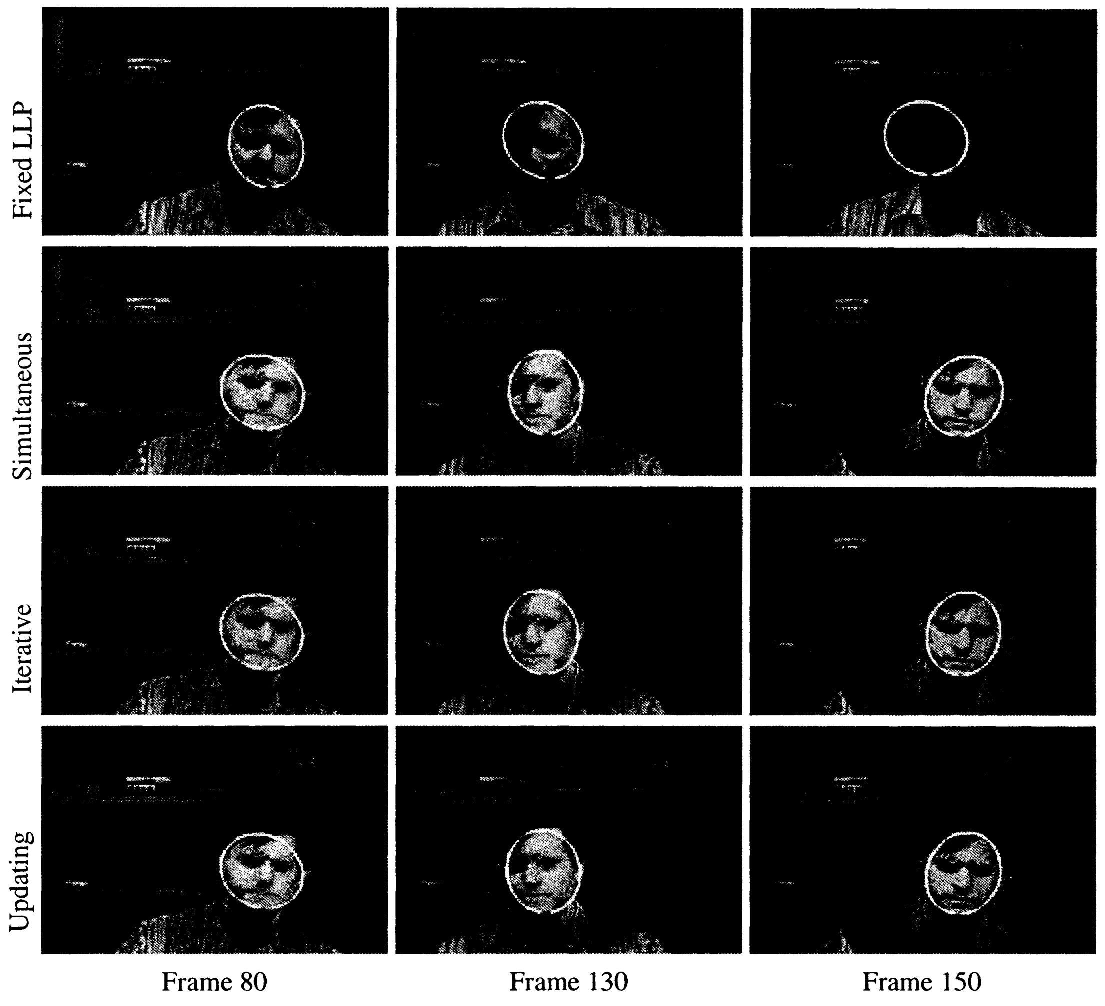
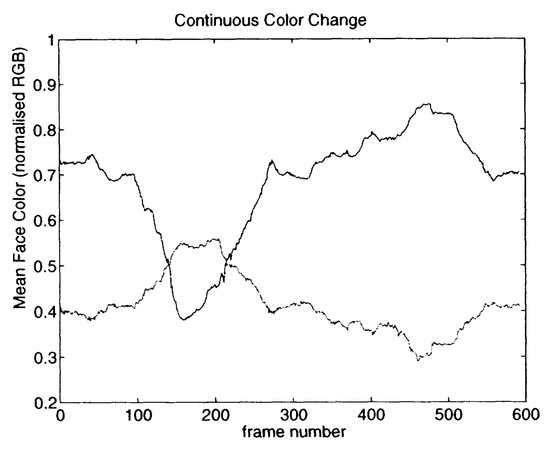
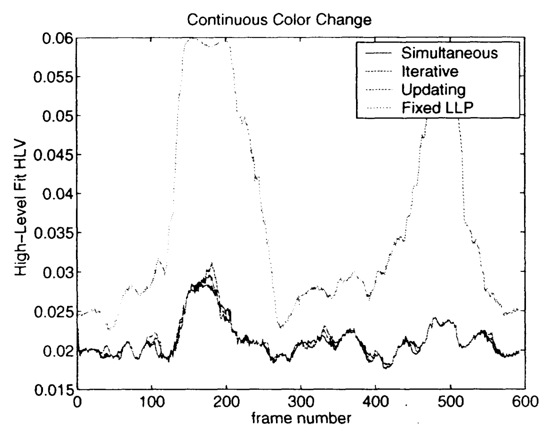


Figure 6: The results for 3 frames of the continuous color change sequence described in Figure 5 for 4 different algorithms described in the text (Simultaneous, Iterative, Updating, and Fixed Low-Level Parameters.) The figure shows the input image overlayed with the pixels that are classified as most likely face pixels highlighted in green, and the fit ellipse drawn on the figure in white. The 3 algorithms Simultaneous, Iterative, and Updating all perform well, and very similarly. The Fixed LLP algorithm begins to fail around frame 130 when the color balance begins to change. By frame 150 the algorithm has completely failed. See the project webpage `http://www.ri.cmu.edu/projects/project_529.html` for the results for the entire sequence.

and the high-level fit ellipse is displayed by drawing the curve $\mathrm{dist}(\mathbf{x}) = 1.0$. Overall, the three algorithms which do optimize over the low-level parameters (Simultaneous, Iterative, and Updating) all perform well, and similarly (more later). The Fixed LLP algorithm begins to fail around frame 130 as the color balance begins to change and completely fails by frame 150. Also see the movie on the project webpage `http://www.ri.cmu.edu/projects/project_529.html`.





(a) Mean Face Color μ_F (b) High-Level Fit HLV

Figure 7: Plots of (a) the 2 components of the mean face color μ_F and (b) the high-level fi t HLV for each frame in the continuous color change sequence in Figure 5. In (a) only the results for the Simultaneous algorithm are included to keep the fi gure simple.

More details of the results are included in Figure 7. In our framework, the low-level parameters are computed for each frame as the parameters which optimize the high-level fitting function HLV. We plot the two components of the recovered mean face color μ_F across frame number for the Simultaneous algorithm in Figure 7(a). Figure 7(a) clearly demonstrates the color variation in the sequence. The results for the Iterative and Updating algorithms are very similar and are omitted. Figure 7(b) plots the high-level fit HLV across the entire sequence for all 4 algorithms. The 3 algorithms which do optimize over the low-level parameters all perform very similarly. The 2 spikes for the Fixed Appearance algorithm show where it fails (and luckily recovers.)

A natural question at this point is whether there is any difference between the three algorithms, Simultaneous, Iterative, and Updating. In Table 1 we plot the average value of the high-level fit HLV for the continuous color change sequence. The Simultaneous optimization algorithm performs the best, followed by the Iterative algorithm, and then the Updating algorithm. The Fixed Appearance algorithm performs far worse. These results confirm that the Simultaneous algorithm is the best algorithm for optimizing HLV. The difference between the 3 algorithms that do optimize over the appearance is very little, however. Both the Iterative and Updating algorithms are good approximations to the Simultaneous algorithm (at least for this sequence.)

Table 1: The average value of the high-level fi t HLV for the 4 algorithms for the continuous color change sequence. The Simultaneous algorithm performs slightly better than the Iterative algorithm which in turn performs slightly better than the Updating algorithm. The Fixed LLP algorithm performs far worse.

Algorithm	Average HLV
Fixed LLP	0.030639
Simultaneous	0.021185
Iterative	0.021325
Updating	0.021525

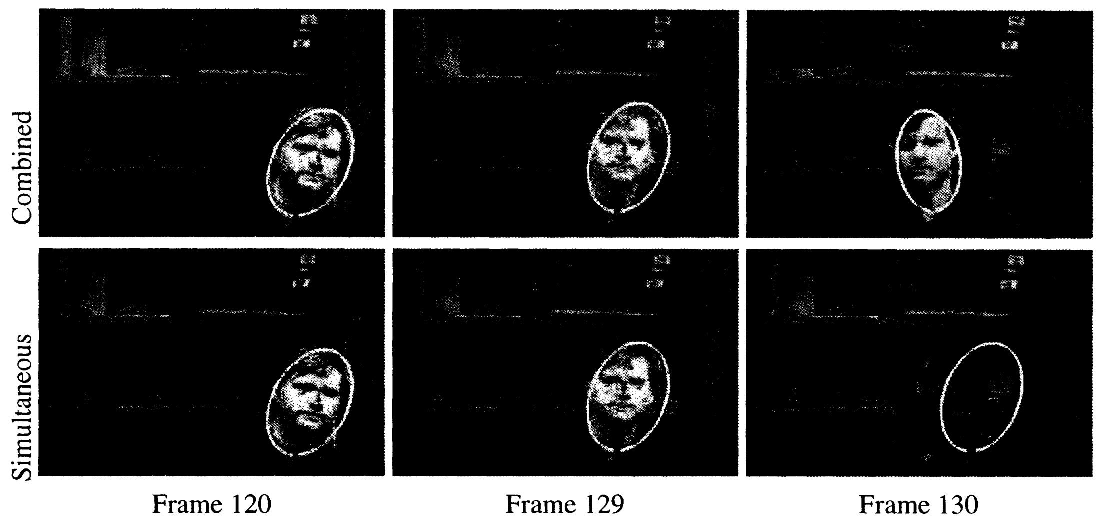
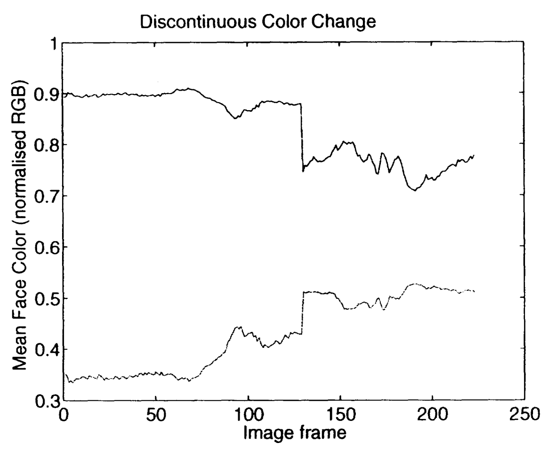


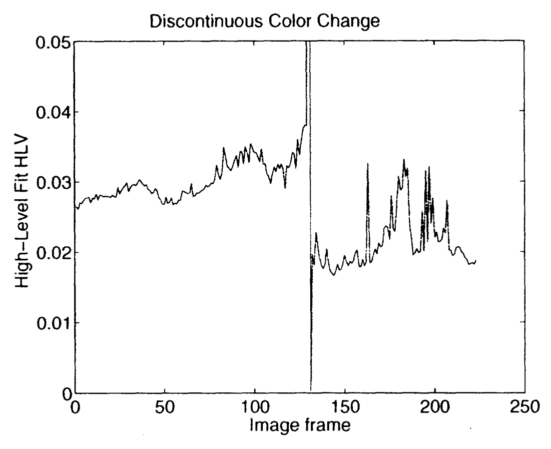
Figure 8: Results for 3 frames of the discontinuous color change sequence for the Simultaneous and Combined algorithm described in the text. There is a discontinuity in the color and appearance between frame 129 and frame 130. The combined algorithm is able to track correctly across the discontinuity because the optimization is a combined gradient-descent/random search. The Simultaneous algorithm is unable to track across the discontinuity because it relies only on a gradient-descent optimization.

3.4 Experiment 2: Discontinuous Color

In Experiment 1 we just ran the gradient descent optimization algorithms (after the first frame.) We now demonstrate the combined detection and tracking implementation of Section 2.4.4. We compare the Simultaneous algorithm with a Combined algorithm that performs both the full Gauss-Newton tracking and the random-sampling detection algorithm *in each frame*. See Section 3.2. We compare the algorithms on a discontinuous color change sequence which was created by splicing together two continuous color change sequences like the one in Figure 5. Note that at the color discontinuity, there is also a discontinuity in the position of the face so an algorithm that builds a



(a) Mean Face Color μ_F



(b) High-Level Fit HLV

Figure 9: (a) The 2 components of the mean face color μ_F and (b) the high-level fi t HLV for each frame in the discontinuous color change sequence in Figure 8 for the Combined algorithm.

new color model by assuming the face did not move would fail.

Some of the results are shown in Figures 8 and 9. In Figure 8 we include 3 frames for each of the two algorithms. There is a discontinuity in the face color and position between frame 129 and 130. The Combined algorithm is able to track across this discontinuity because it combines a random-sampling detection optimization algorithm with a gradient-descent tracking algorithm. On the other hand, the Simultaneous algorithm fails at the discontinuity. In Figure 9 we plot the mean face color μ_F and the high-level fit HLV for each frame in the sequence for the Combined algorithm. The plots clearly demonstrate the discontinuity.

4 Color-Based Lane Tracking

We now demonstrate how our framework can be applied to two systems that already existed before we conceived of the framework. The first such system is a color-based lane tracking system. The system is based on the same color pixel classification that the color-blob face tracker is, however, it also uses a variety of techniques. The road pixels are classified (removing all "edge pixels) and then connected components is applied. A simple geometric road model is then fit and used to select the connected components that are the road. Overall this system worked fairly well, but

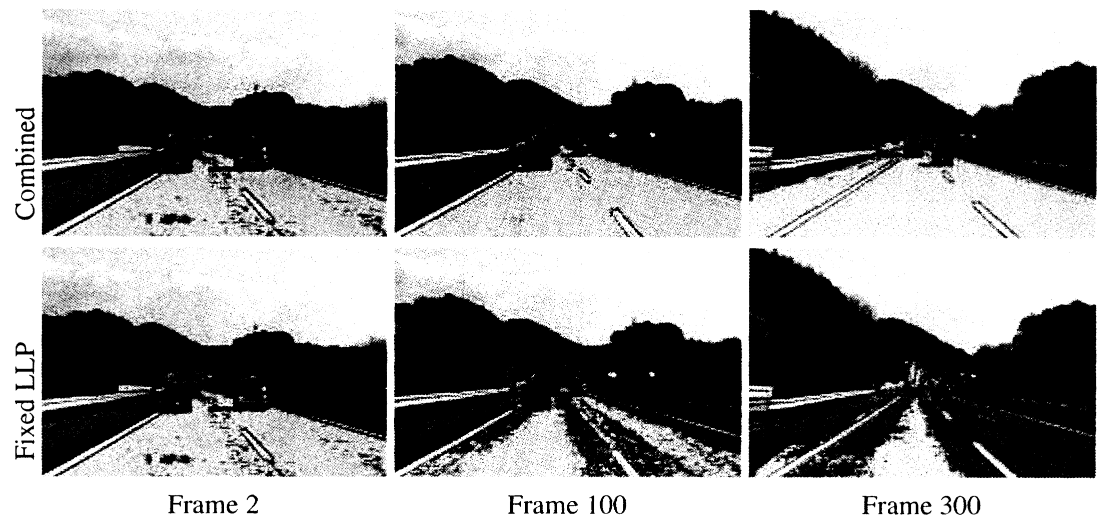
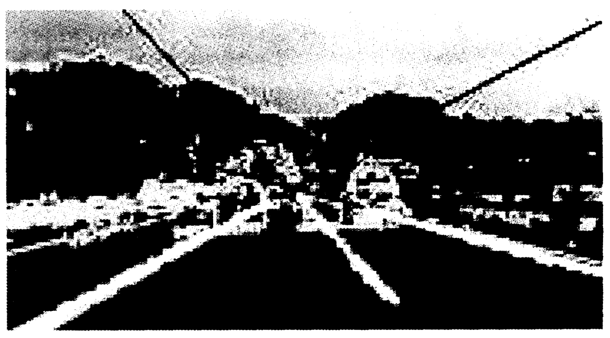


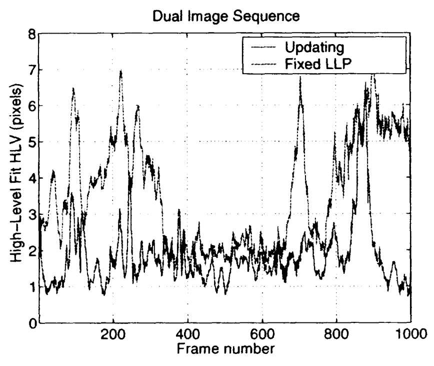
Figure 10: A comparison of the Combined algorithm and the Fixed LLP algorithm for the color-based lane tracking system. Because the Combined algorithm estimates the low-level road color parameters in each frame, it is able to cope with the slow change in the color of the road surface. On the other hand, the Fixed LLP algorithm begins to fail as soon as the road color changes. See the project webpage `http://www.ri.cmu.edu/projects/project_529.html` for the results for the entire sequence.

was always sensitive to the road color, and required hand initialization. To this system we added an algorithm that performed a Combined approximate gradient-descent and random-sampling on the low-level road-color parameters. A comparison of our system using this low-level parameter setting algorithm with the same system with a Fixed LLP, hand-initialized road color model is shown in Figure 10.

5 Edge-Based Lane Tracking

We applied our framework to a second legacy system, an edge-based lane tracker. This system operates by first applying edge detection and then filtering the edges based on their cross section to remove as many non-lane edges as possible. The primal sketch is the resulting edge map. The low-level parameters are the edge threshold and the edge cross section model. The high-level module fits a simple lane model to the filtered edge map. We took the legacy system and added a gradient-descent Updating algorithm on the low-level parameters. The result for one frame of the resulting Updating system in included in Figure 11(a). Figure 11(b) includes a plot of the final


(a) Example Frame using Updating



High-Level Fit HLV

Figure 11: (a) An example frame using our edge-based lane tracker with the Updating algorithm. The detected edges are displayed in yellow, the road edges in white, and the detected lanes in black. (b) A plot of the high-level fi t HLV across time. The value of HLV if signifi cantly better with the Updating algorithm (blue-line) than with the Fixed LLP algorithm (green line).

high-level fit HLV for the original Fixed LLP system (green) and the new Updating system (blue). The plot shows a clear improvement.

6 Discussion

The main contribution of this paper is to propose a framework for the feedback of information from high-level modules to low-level modules. Specifically, we posed this problem as setting the parameters of the low-level module. The usual approaches to setting the low-level parameters are by-hand, by empirical evaluation, by learning them, or by updating them from frame to frame. Instead, we posed the problem as solving a simultaneous optimization over both the low-level and high-level parameters. We also proposed a variety of different implementations of this framework.

We demonstrated the framework on 3 applications, 1 constructed with the framework in mind, and 2 implemented long before the framework was conceived of. The main point to note in the experiments is that the low-level parameters are estimated from the input image using the information fed back from the high-level module. They are not set by hand, empirically estimated, or learnt. Because the low-level parameters are computed from the input image, no prior knowledge or initial estimate of the low-level parameters (e.g. face/road color, edge detector threshold) is required. Moreover, when applied to a video sequence, the algorithms are automatically robust to changes in the low-level parameters, whether smooth or discontinuous.

Our framework puts a huge amount of importance on both the high-level error function HLV and the low-level vision function LLV. Designing a system with feedback from the high-level module to the low-level module consists of carefully deciding what these functions should be. We believe this is the way it should be. The system designer's effort is better devoted to carefully choosing HLV and LLV than to designing ad-hoc feedback mechanisms.

Acknowledgments

The research described in this paper was supported by Denso Corporation, Japan.

References

[1] Y. Aloimonos and D. Shulman. *Integration of Visual Modules: An Extension of the Marr Paradigm.* Academic Press, 1989.

[2] W.T. Freeman, E.C. Pasztor, and O.T. Carmichael. Learning low-level vision. *International Journal of Computer Vision*, 20(1):25–47, 2000.

[3] M. Isard and A. Blake. Contour tracking by sochastic propagation of conditional density. In *Proceedings of the European Conference on Computer Vision*, volume 1, pages 343–356, 1996.

[4] D. Marr. *Vision.* W.H. Freeman & Co., 1982.

[5] I. Matthews, T. Ishikawa, and S. Baker. The template update problem. In *Proceedings of the British Machine Vision Conference*, 2003.

[6] Y. Raja, S. McKenna, and S. Gong. Colour model selection and adaptation in dynamic scenes. In *Proceedings of the European Conference on Computer Vision*, 1998.

[7] M. Shin, D. Goldgof, K. Bowyer, and S. Nikiforou. Comparison of edge detection algorithms using a structure from motion task. *IEEE Transactions on Systems, Man, and Cybernetics*, 31(4):589–601, 2001.

[8] K. Toyama and G. Hager. Tracker fusion for robustness in visual feature tracking. *In SPIE International Symposium on Intelligent Systems and Advanced Manufacturing*, 2589, 1995.

[9] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[10] J. Yang, W. Lu, and A. Waibel. Skin-color modeling and adaptation. In *Proceedings of the Asian Conference on Computer Vision*, pages 687–694, 1998.