# Real-time Evolving Swarms for Rapid Pattern Detection and Tracking

Christopher Middendorff and Matthias Scheutz
Artificial Intelligence and Robotics Laboratory
Department of Computer Science and Engineering
University of Notre Dame, Notre Dame, IN 46556
cmidden1@cse.nd.edu

**Abstract**

In this paper, we discuss a simple extension to the standard particle swarm optimization algorithm, inspired by genetic algorithms that allow swarms to cope better with dynamically changing fitness evaluations for a given parameter space. We demonstrate the utility of the extension in an application system for dynamical facial feature detection and tracking, which uses the proposed "real-time evolving swarms" for a continuous dynamic search of the best locations in a two-dimensional parameter space to improve upon feature detection with static parameters. We show in several experimental evaluations that the proposed method is robust to lighting changes and does not require any calibration. Moreover, the method works in real time, is computationally tractable, and not limited to the employed static feature detector, but can be applied to any n-dimensional search space.

## Introduction

Particle swarm optimization (PSO) has been successfully employed in a variety of applications to quickly find optimal or close-to-optimal parameters in partly high-dimensional parameters spaces (e.g., (Kennedy and Eberhart, 1995; Kennedy et al., 2001)). The idea behind PSO is to use $n$ particles or *agents* (that together form the *swarm*) to explore different regions of the parameter space in parallel. Agents, as in a biological swarm, attract each other to varying degrees dependent on the value of their location in the parameter space, thus causing agents in general to move towards better places in parameter space. If the value surface of the parameter space is smooth and there are pronounced peaks, agents will eventually gather around them. While this is desirable for static value evaluations, it can be problematic in the dynamic case where fitness surfaces change and peaks can turn into valleys, in which case swarm agents need to start their gradient ascent again.

In this paper, we propose a simple mechanism inspired by genetic algorithms that will allow swarm agents to dynamically evolve, thus helping them to find and track fitness maxima in parameter spaces very quickly based on dynamically changing fitness evaluations. We will demonstrate the proposed method by applying evolving swarms to the real-time vision problem of finding and tracking facial features.

## Real-time Evolving Swarms

We start with a brief review of swarm agents (or particles) in PSO systems. An *agent* or particle is characterized by a location $\bar{x}$ in an $n$-dimensional parameter space $\mathcal{P}$. The parameter space has an associated (static) *evaluation* function $\mathcal{E} : \mathcal{P}_D \mapsto \mathcal{R}$ (from the parameter space into the real numbers) that determines the quality or *fitness* of a given location (i.e., set of parameter values) for each location in $\mathcal{P}_D$ based on an evaluation domain $D$. Each agent (located in the parameter search space) has a velocity in each dimension that varies with time, depending on its own current and past position and the current and past positions of other agents.

The velocity $v_{i,j}(t)$ of agent $i$ in dimension $j$ at time $t$ is given by

$$\begin{aligned} v_{i,j}(t) = \omega \cdot v_{i,j}(t-1) + \\ c_1 \cdot \phi_1 \cdot (p_{i,j} - x_{i,j}(t-1)) + \\ c_2 \cdot \phi_2 \cdot (p_{g,j} - x_{i,j}(t-1)) \end{aligned} \quad (1)$$

This equation has three components: an *inertia* component $\omega \cdot v_{i,j}(t-1)$, which pulls the agent in its current direction; a *cognitive* component $p_{i,j} - x_{i,j}(t-1)$, which represents the agent's memory of its highest-scoring location $p_{i,j}$ in that dimension (based on $\mathcal{E}$) and its comparison to the current location; and a *social* component $p_{g,j} - x_{i,j}(t-1)$, which is the distance from the global best location $p_{g,j}$ discovered among all agents (Voss, 2003).

The position of each agent in each dimension is then updated by the equation

$$x_{i,j}(t) = x_{i,j}(t-1) + v_{i,j}(t)$$

Typically, swarm agents in a PSO system are initially placed in random locations in $\mathcal{P}$ and then updated for a certain number of iterations, either until a stable position of all agents has been obtained (e.g., all agents end up in the same location or close-by with little to no movement) or a solution has been found by at least one agent that is "good enough".

(Omran et al., 2005) assert that most prior work would give task-specific values to the coefficients $c_1$, $c_2$ and $\omega$.

However, van den Bergh (van den Bergh, 2002; van den Bergh and Engelbrecht, 2002) showed that these must solve the inequality

$$\frac{c_1 + c_2}{2} < \omega \tag{2}$$

to exhibit convergent behavior. Convergent behavior is desirable so swarms are attracted to the best determined value so that region can be more thoroughly examined for a maximum. However, in a non-smooth search space, this is not true; in a space composed of randomly-placed impulses (single points where the fitness function is high), a swarm requires more luck than ability in finding these areas.

We now extend the notion of static evaluation to *dynamic evaluation*, i.e., to a sequence $\mathcal{E}_I = \langle \mathcal{E}_{i_1}, \mathcal{E}_{i_2}, \ldots \rangle$ of functions $\mathcal{E}_{i_k} : \mathcal{P} \mapsto \mathcal{R}$, $I := \{i_1, i_2, ...\}$. Dynamic evaluations reflect situations where the quality of locations in parameter spaces can change over time (e.g., the success of a particular set of parameters that determines the foraging strategy of a simulated animal might change based on the food distribution in the environment). We call such a sequence of functions *real-time*, or *real-time evaluation* if a metric $\mathcal{M}$ is defined for the index set $I$, i.e., $\mathcal{M}$ is defined on $\{i_k | \exists \mathcal{E}_{i_k} \in \mathcal{E}_I\}$. In other words, real-time sequences can be used to model the temporal characteristics of changes of parameter evaluations in an environment (e.g., the shift of shaded locations under a tree on a sunny day).

Depending on the degree of change in the evaluation between $\mathcal{E}_{i_k}$ and $\mathcal{E}_{i_{k+1}}$, swarm agents will be able to cope to varying degrees: minor smooth changes in the fitness surface will lead to quick adaptations, but large transitions (e.g., from a value peak to a value valley) could render the swarm system largely immobile or lead to very slow adaptations. This can be especially problematic in real-time evaluations, where the number of iterations that swarms can use for adaptation is constrained by a real-time interval. Consequently, it might be useful to add mechanisms to swarms that would allow them to react to changes more quickly.

A very simple, yet highly effective mechanism is to treat a swarm system of $n$ agents as performing $m$-beam search (with $m < n$). In this case, as in genetic algorithms, $n$ agents are initialized in randomly generated states in the search space. The agents are ranked according to their performance based on the fitness evaluation $\mathcal{E}_{i_k}$. The best $m$ agents are kept, while the other agents are eliminated and are replaced by *offspring* based on a *selection strategy* (e.g., mutation of the locations of existing chosen agents, cross-over of their dimensions, or simply a random location drawn from a random distribution of the overall parameter space to reach relatively uniform coverage). This combines the maximum-seeking tendency of swarm-based search with the strengths of culling (i.e., faster convergence) and random searching (as implemented in GAs with mutations and cross-over, e.g. (Grefenstette, 1999)) to quickly find and converge on peaks in a multidimensional space.

It is the dual nature of the swarms in the system that allows for rapid feature detection. The highest-performing swarms, due to their preservation between iterations and between frames, seek optimal solutions in their current location (which corresponds to the best values for the frame). However, between frames it is possible that their current location may no longer be the best location. In these cases, it is the randomly-placed swarms that can quickly find the best locations, due to the fact that they are already distributed throughout the space; they do not need to seek because they are already in place.

## Previous Work

Static approaches to facial feature detection are unsuitable due to the number of constraints that must be placed on a subject and his environment, the amount of pre-processing required, or both. Color blob detection, for example, has been used to isolate the mouth (Hsu et al., 2002) in an image. The algorithm works on the assumption that the mouth has stronger red component (Cr) and weaker blue component (Cb) than the face. So by determining the average Cr and Cb of the face, the mouth region can be isolated by boosting the Cr component and taking the difference-squared between that and the Cr/Cb of each pixel in the face. This method is dependent on quality of lighting conditions, and the pre-processing step of skin isolation of the face. These pre-processing steps also take several seconds per frame, which is completely useless in real-time applications.

Swarm-based methods have also been used to analyze images. For example, the edge detection system in (Zhuang, 2004b; Zhuang, 2004a; Ramos and Almeida, 2000) or the color segmentation system in (White et al., 2004) featured a swarm system which allocate one agent per pixel in its method of edge detection. This method, however, is image-specific as the agents move over the pixels. Unlike parameter-based methods, where a search range can be applied to a series of images, the edges would have to be calculated over every frame.

(Ramos et al., 2005) presented a method using ant colonies to explore dynamic spaces, similar to the dynamic field of the parameter space (described in the following section) explored by the swarm agents. All of the memory of an ant's position is left on each discrete space as pheromones, with no knowledge of the actual positions of the other ants. This certainly provides a very thorough search; however, this requires thousands of ants to explore the search space so that pheromones can accumulate and be tracked before evaporating. (Grefenstette, 1999) offers a genetic algorithm approach to tracking maxima across dynamic fitness landscapes; the fixed hypermutation model is similar to the random replacement of the swarm agents in our system, and this model was shown to perform best over an abruptly-changing landscape.

We will now demonstrate the utility of evolving swarms for real-time evaluations in a practical domain that has been traditionally challenging: real-time facial feature detection and tracking under changing lighting conditions. In this task, it is critical to find evaluation peaks quickly (in parameter space), for lighting conditions can change from one image frame to the next, and will, thus, likely require ongoing adaptation. Here, evolving swarms can play out their strengths. Once an evaluation peak is found on a given frame, $m$ agents will reside in its vicinity (moving only slowly), while $n-m$ new agents will be placed in other locations in parameter space (in our case randomly), thus allowing a faster reaction of the swarm to changes in lighting conditions (for large enough $n-m$ based on the size of the parameter space), as it is likely that at least some of the $n-m$ new agents will be in a location with better fitness than the $n$ agents that lost the feature.

## Swarm-based Adaptive Feature Detection and Tracking

To make the domain-dependence of the evaluation functions more explicit, we consider the parameters of the visual feature detection system as variables to a function

$$features = F(img, k_1, ..., k_m, x_1, ..., x_n)$$

where *img* is the image to be processed, $k_1, ..., k_m$ are non-variant parameters of the system, $x_1, ..., x_n$ are parameters to this system, $F$ is the extraction function, and $features$ is the set of extracted features. Further, $score = g(feature)$ can apply evaluation function $g()$ to a feature that has been extracted, to provide feedback to the swarm system.

Determining values $x_1, ..., x_n$ such that *score* is maximized for $g$ requires searching an $n$-dimensional *parameter space*. While this can be done manually[1], this is not only tedious but also subject to changes in *img*.

Using swarms for feature detection grants the adaptability necessary to perform in a real-world environment. Variable parameters of a feature detection space can be represented as dimensions in this parameter space, and the position of each swarm agent represents values for each of these parameters; edge detection thresholds can be represented in 1-2 dimensions and color blob detection can be represented in 6 (a lower and upper bound for each of three color dimensions). As the algorithm progresses, a fitness function is applied to the resulting detected features. These are compared to determine the highest-performing swarm, which represents the best candidate for parameters.

An edge-detection algorithm was chosen over a color-based method to meet the speed constraints of a real-time system. Edge-detection can be performed on an intensity image (a black-and-white image obtained by color transformation) using two dimensions, where the dimensions in space represent the upper and lower thresholds of the Canny edge detector (Canny, 1986). Straightforward color detection systems (e.g. (Hsu et al., 2002)) which detect color regions, would require a six-dimensional search space. Intuitively, this will require more agents, more time, or both to determine the optimal value.

Rather, these will be used to explore a two-dimensional space representing the upper and lower thresholds of a Canny edge detector. As seen in (Voss, 2003) and (Omran et al., 2005), swarm agents can be used to quickly explore multi-dimensional spaces. Each agent is instantiated at a random point in this 2D space with a random velocity in each dimension, within the range of [0,1], and fixed attraction to global and local best values.

A search of parameter space is preferable to directly searching a specific image; moving between adjacent coordinates in parameter space should provide smooth transitions in the image processing (color blobs and edges should not suddenly vanish between adjacent points). This concept is critical for our real-time system, as it requires the re-use of the determined parameters.

The face is detected using the OpenCV library[2], and the eye region to be searched is defined as $\frac{1}{3}$ of this face, starting from the top $\frac{1}{6}$ down to the middle. Further, this could be halved by a vertical line through the center to allow the swarms to independently process the separate eyes. These rectangles are the regions searched for features.

```
FUNCTION LocateBestFeature(agents, face, eyebox, nIters)
for K from 1 to nIters do
  for J from 1 to length(agents) do
    feature ← DetermineBoundAndScore(agents[J], face)
    if feature.score ≥ agents[J].previousBestScore then
      agents[J].previousBestScore = feature.score
      agents[J].bestLocalPosition = agents[J].curPosition
    end if
    agents[J].modifyCumulative(feature.score)
  end for
  sort(agents) by cumulative scores
  for J from 1 to length(agents) do
    agents[J].updateSwarmAgent(agents[0])
    agents[J].previousWin ← false
  end for
  agents[0].previousWin ← true
  replace lowest performing replaceNum agents
end for
reward agents[0]
return DetermineBoundAndScore(agents[0], face)
```

Figure 1: The algorithm for locating the best features.

`LocateBestFeature`, shown in Figure 1, is the function responsible for determining the best agent to use for this frame, and subsequently using it to find the feature. Primarily, it submits agents to the function `DetermineBoundAndScore`, which determines the best

[1] Certainly, a parameter space may be so large that a "perfect solution" is impossible to pinpoint; however, it is possible to find a solution by trial-and-error that is sufficient for a specific feature-detection task.

[2] http://www.intel.com/research/mrl/research/opencv/

```
FUNCTION DetermineBoundAndScore(agent, image, boxratio)
params ← agent.currentPosition
edgemap ← canny(image, params[0], params[1])
contourlist ← findContours(edgemap)
for K from 1 to length(contourlist) do
  box ← boundingboxofcontour
  if box.width > boxratio * box.height then
    potentialboxes.push(box)
  end if
end for
prevscore ← −∞
boxToReturn ← 0
for K from 1 to size(potentialboxes) do
  score ← DetermineScore(potentialboxes[K])
  if score ≥ prevscore then
    boxToReturn ← potentialboxes[K]
    prevscore ← score
  end if
end for
return boxToReturn
```

Figure 2: The algorithm for determining boundary and score.

```
FUNCTION score(image, pReg, eReg, minW, maxW, minH, maxH)
shapeFlag ← 0
posFlag ← 0
if pReg.width > minW · eReg.width and pReg.width < maxW · eReg.width then
  shapeFlag ← shapeFlag + pReg.width / (maxW · eReg.width)
else
  shapeFlag ← shapeFlag − 1
end if
if pReg.height > minH · eReg.height and pReg.height < maxH · eReg.height then
  shapeFlag ← shapeFlag + pReg.height / (maxH · eReg.height)
else
  shapeFlag ← shapeFlag − 1
end if
shapeFlag ← shapeFlag / 2
posFlag ← 1 − (2 · pReg.y + pReg.height) / eReg.height
if posFlag > 0 then
  posFlag ← 1 − posFlag
end if
return posScale · posFlag + shapeScale · shapeFlag
```

Figure 3: The algorithm for determining score.

possible feature bound for that agent (as well as that bound's score) and returns it. The `LocateBestFeature` function determines for each agent whether this new location is its personal best, and then accumulates the score. Scores accumulate within an agent but decay with time, giving each agent a memory which extends several frames into the past. After each iteration, the agent positions are updated and the process repeated. The lowest-performing *replaceNum* agents are replaced after every iteration. The value *previousWin* is used as a tiebreaker when sorting the swarm agents.

After all iterations, the highest-performing agent is used to determine the bound for the image feature. This agent is also given a reward (a small increase to score) to help stabilize agent selection. If one agent does not clearly dominate, feature detection can produce unstable results since the agents will have differing positions in the parameter space.

The state of the agent-system is preserved between calls, though it can be reset if necessary. This allows a swarm to use previously-determined parameters for operation on subsequent frames.

The `DetermineBoundAndScore` algorithm, shown in Figure 2, implements the particular method for this feature-detection system. Canny edge detection is performed to determine the edges, using the coordinates in parameter space to determine the upper and lower thresholds of the edge detector. From the detected edges, the contours (edge points associated into connected curves) defining the feature can be found and bounded. Each bounding box is analyzed, comparing the size of the bounding box against the size of the entire eye region. The highest-scoring box is returned with its score. The *boxratio* (the width/height ratio) is used to eliminate the detection of sideburns which, like eyebrows, tend to contrast against skin.

For the feature evaluation, shown in Figure 3, the box that has been found is compared against the maximum box. Because the determined face can be at any distance, the system is stronger when performing comparisons to the face size, instead of fixed pixel dimensions. The values *minW*, *maxW*, *minH* and *maxH* (minimum and maximum width and height) scale the maximum eye region. In this case, features are rewarded for having dimensions within a certain size range, which is relative to the size of the maximum box.

By this method, the position is rewarded based on the location of its median. If the median of the detected box is above the median of the eye region, the reward is from [0,1], based on distance from the top (a reward of zero at the top, ranging to a reward of 1 at the center). In the bottom half, however, the reward ranges from [0,-1], with zero at the center, down to -1 at the bottom.

This is a somewhat unusual reward system, though not without meaning. In the bottom half of the region, a box bounding the *eye* tends to meet the geometric criteria, so the score needs to be decreased for falling below the height-wise median. In the top half of the region, however, a shadow above the eyebrow (on the eyebrow ridge) should not be rewarded over the eyebrow itself.

The values *posScale* and *shapeScale* are determined experimentally, to scale the contribution of each factor to the final score.

## Experiments and Results

To test the system, two subjects with differing facial attributes were used. One had dark skin and prominent facial features (for the purposes of this example, this means thick eyebrows). The other had light skin and less prominent features. The different subjects served several purposes for testing the system:

1. The different colors of the subjects' hair and skin will change the effect of light on both subjects. A change in lighting resulted in stronger changes in the contrast between the illuminated and darkened faces of the fair-skinned subject, than in the dark-skinned subject. This

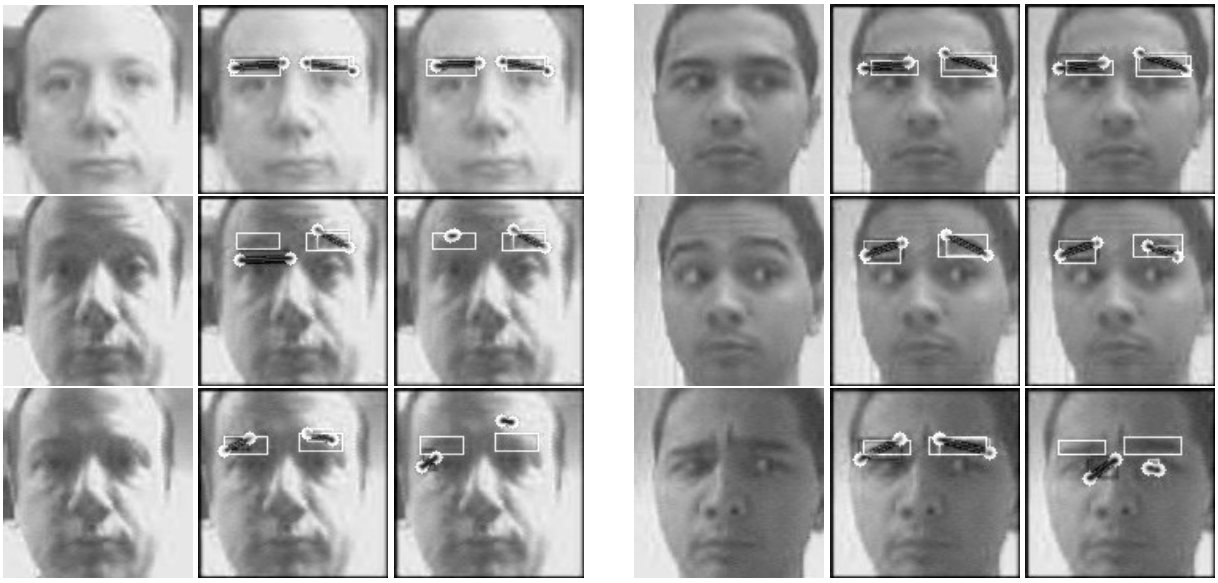

Figure 4: **Top row:** full lighting, **Middle row:** dimmed lighting, **Bottom row:** ambient lighting only.

stronger change results in a sharper change in the intensity difference between the skin and eyebrow, which in turn results in a change in the parameters necessary for edge detection.

2. Different facial-feature prominence has a twofold effect: the contrast between the feature and the skin, which affects the parameters of the edge detection; and the dimensions of the feature, which will affect the evaluation of eyebrows of differing shapes.

A gallery for each subject was recorded with a lamp shining to the left. Each subject raised and lowered his eyebrows throughout the recording. Approximately halfway through recording, the light was switched off. Afterwards, the location of each eyebrow in all pictures was determined for the purpose of comparison against the features found by the swarm system.

Two feature-detection systems were applied to each subject gallery, composed of 98 images of the dark-skinned subject and 93 of the fair-skinned subject. The first used the `DetermineBoundAndScore` algorithm for a fixed value (by submitting a single, immobile agent with a predetermined location to the algorithm). The second used the agent system, replacing the lowest-performing 80% of agents on each iteration. Other parameters were chosen to optimize the accuracy of the detector against its speed to ensure real-time performance for the vision system: $length(agentList) = 20$, $nIters = \{minIters, 30\}$, $minIters = 1$, $maxFC = 5$, $bottomFC = 5$, $thresh = 0.7$, $replaceNum = 0.8 \cdot length(agentList)$, $boxRatio = 0.8$, $minWidth = 0.05$, $maxWidth = 0.65$, $minHeight = 0.15$, $maxHeight = 0.4$, $posScale = 0.3$, $shapeScale = 0.7$, $replaceNum = 0.8 \cdot length(agentList)$.

- The number of agents ($length(agentList)$) and number of iterations ($nIters$) performed determine the speed of the algorithm, and the swarm's ability to explore the space. The maximum value of $nIters$ is the number of iterations the swarms will spend analyzing a frame when the optimal value is originally sought. For this implementation, it was determined that a small number of agents exploring the space for a small number of iterations can determine sufficient feature-tracking parameters. The value of $minIters$ is the number of iterations performed over $length(agentList)$ on each image when the values are being used; this value must be small for real-time operation.

- The values $maxFC$ and $bottomFC$ were defined to denote the number of consecutive frames to spend originally seeking the optimal, and the number of frames for which error (a returned score below $thresh$ was allowed. When the number of erroneous frames exceeded $bottomFC$, the swarm was re-initialized and the optimum re-sought.

- Inside the agent system, the replacement value was set at 80%, leaving the top 6 agents untouched on every iteration. To prevent these from completely dominating the other agents (due to their higher cumulative values), the cumulative values decrease by $\frac{1}{3}$ on every iteration.

- The values of the ratios between the eyebrow size and the eye region size were determined experimentally; the values have to be restricted in some way to prevent the bounding box from expanding to cover the entire region. Larger boxes tend to be superior to smaller ones, since smaller boxes mean that the edge detection values are too

high, and that the eyebrow has been eroded. However, very low threshold values will find contours over the entire area, which would result in the entire area being found as a feature.

- *positionScale* and *shapeScale* were decidedly weighed to emphasize shape more than position. Because eyebrow position in the image could vary so easily (head tilting, expression changes), it could not be the sole determinant. On the other hand, it needed enough weight to prevent an eye (with its similar geometry) from being found as an acceptable eyebrow.

Results can be seen in Figure 4. The top three rows illustrate the fair-skinned subject under changing lighting conditions; the first row is with a lamp on, the second row is just after the lamp was turned off, the third is 1-2 seconds after the lamp has been turned off. The bottom three rows are the dark-skinned subject under the same conditions. The first column is the original image, the second is that image after processing with the swarm-based system, and the third is the first image after being processed by a static detector using the same evaluation system as the swarm-based detector. The white box represents the actual eyebrow (as annotated by hand), and the black line is the eyebrow approximation detected by the swarm system.

To evaluate the performance of each system, the metric defined was $d_{sum} = \sum_{k=1}^{n}(t_k + b_k)$ , where $t_k$ is the distance from the top-left point of the determined eyebrow to the top-left point of the actual eyebrow, and $b_k$ is the distance from the bottom-right point of the determined eyebrow to the bottom-right point of the actual eyebrow. The sum is taken over the gallery, which contains $n$ images. By this metric, a lower $d_{sum}$ is desired because it represents a small difference from the actual eyebrow.

The static detector had a $d_{sum}$ of 977 for the left eye and 2013 for the right. By comparison, the dynamic detector had a $d_{sum}$ of 749, and 1321 for the right eye. The difference between the two detectors is significant ($t = 4.411$, $df = 182.937$, $p < .0001$).

## Discussion

The score was calculated for all possible agents on each image. This produced the surface the swarm agents explored for each image in the gallery. For an individual, this surface can change for two reasons: lighting, and motion. Lighting affects this feature surface because the peaks will shift, depending on the amount of light that is added/removed and the physical properties of the subjects. Motion will have a much more subtle effect, due to the evaluation function. The results earn their scores based on their shapes and positions; as these change, the scores will change slightly as well.

It is apparent in Figure 5 that this search space is not completely continuous, but has several discontinuous jumps throughout the space. This is a side-effect of trying to make

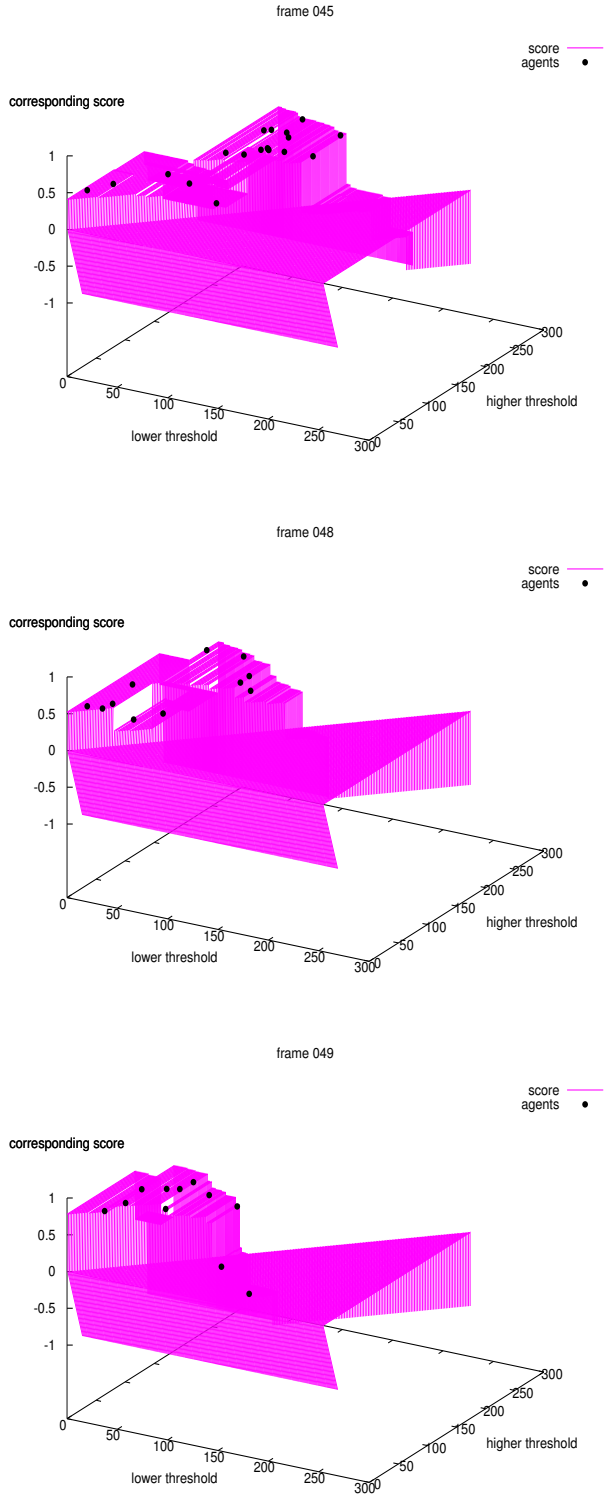


Figure 5: **Top:** full lighting, **Center:** dimmed lighting, **Bottom:** ambient lighting only.

discrete evaluations continuous; evaluations are discrete in this case since, for example, the dimensions of this eyebrow have certain size ranges, and anything inside this size range will be equally acceptable. Similarly, the sizes are integral because they are measured in pixels. In order to provide gradients for the swarms to follow, this had to be "smoothed". The result is a discrete approximation of a curve.

Because lighting changes will alter the parameter space, the system is augmented by the randomly-located swarms, located by the black points in Figure 5. Over the course of only a few frames, the shape of the graph changes dramatically, as the domain of the peaks decreases with the amount of light (as the difference between pixels will decrease with less lighting, the necessary threshold to separate them will decrease). The peak in the first frame, over the course of only a few frames (about 60-130 for the low threshold), became a low plateau. What was formerly a low plateau in the first frame (about 20-40 for the low threshold) became the peak.

The swarms in the region with a low "lower threshold", which are lower-ranked in the system when light is on, take priority almost immediately when the light is turned off. The result is that features are tracked consistently throughout the transition. In a traditional swarm system, however, there is no mechanism that will allow the swarms to easily move out of the valley they suddenly find themselves in. While these agents do have velocities and inertia to carry them through the space, they will lack the well-performing agents that can pull them towards a superior region.

## Conclusion

While swarms are suitable for exploring multidimensional space, they were not suited to the demands of seeking peaks in a dynamic space in real time. For the problem of real-time feature detection and tracking, it is important to have a system that can quickly adapt to changes in the environment. Static methods are insufficient due to time complexities, but the evolving swarms provide adaptability; while the best agents converge on the maximum point in the parameter space, randomly-placed agents throughout the parameter space will rapidly respond to a sudden field change in their favor, resulting in the seamless track of a feature over the environmental change. Biodiversity increases the performance of the system in two ways: the convergence on a high feature is helpful in the case of minor changes to the shape of the field, while the scattered agents quickly take advantage of sudden peaks caused by major changes in the field.

## References


Canny, J. (1986). A computational approach for edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698.

Grefenstette, J. J. (1999). Evolvability in dynamic fitness landscapes: A genetic algorithm approach. In *Proceeding of the 1999 Congress on Evolutionary Computation*, pages 2031–2038. IEEE Press.

Hsu, R.-L., Abdel-Mottaleb, M., and Jain, A. K. (2002). Face detection in color images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):696–706.

Kennedy, J. and Eberhart, R. C. (1995). Particle swarm optimization. In *Proc. IEEE Int. Conf. Neural Networks*, volume 4, pages 1942–1948.

Kennedy, J., Eberhart, R. C., and Shi, Y. (2001). *Swarm Intelligence*. Academic Press.

Omran, M., Engelbrecht, A. P., and Salman, A. (2005). Particle swarm optimization method for image clustering. *International Journal of Pattern Recognition and Artificial Intelligence*, 19(3):297–321.

Ramos, V. and Almeida, F. (2000). Artificial ant colonies in digital image habitats: A mass behavior effect study on pattern recognition. In Dorigo, M., Middendorf, M., and Stuzle, T., editors, *From Ant Colonies to Artificial Ants - 2nd Int. Wkshp. on Ant Algorithms*, pages 113–116.

Ramos, V., Fernandez, C., and Rosa, A. C. (2005). Social cognitive maps, swarm collective perception and distributed search on dynamic landscapes. *Brains, Minds and Media - Journal of New Media in Neural and Cognitive Science*.

van den Bergh, F. (2002). *An analysis of particle swarm optimizers*. PhD thesis, University of Pretoria.

van den Bergh, F. and Engelbrecht, A. P. (2002). A new locally convergent particle swarm optimizer. In *Proc. IEEE Conf. Systems, Man, and Cybernetics*.

Voss, M. S. (2003). Social programming using functional swarm optimization. In *Proceedings of the IEEE Swarm Intelligence Symposium 2003*, pages 103–109.

White, II, C. E., Tagliarini, G. A., and Narayan, S. (2004). An algorithm for swarm-based color image segmentation. In *Proceedings of the IEEE SOUTHEASTCON - 2004 "Engineering Connects"*, pages 84–89.

Zhuang, X. (2004a). Edge feature extraction in digital images with the ant colony system. In *IEEE International Conference on Computational Intelligence for Measurement Systems and Applications*, pages 133–136.

Zhuang, X. (2004b). Image feature extraction with the perceptual graph based on the ant colony system. In *2004 IEEE International Conference on Systems, Man and Cybernetics*, pages 6354–6359.