Tracking Multiple Objects in Non-Stationary Video

Hoang Nguyen, Bir Bhanu
Center for Research in Intelligent Systems
University of California, Riverside
Riverside, California 92521 USA
nthoang@cs.ucr.edu, bhanu@cris.ucr.edu

ABSTRACT

One of the key problems in computer vision and pattern recognition is tracking. Multiple objects, occlusion, and tracking moving objects using a moving camera are some of the challenges that one may face in developing an effective approach for tracking. While there are numerous algorithms and approaches to the tracking problem with their own shortcomings, a less-studied approach considers swarm intelligence. Swarm intelligence algorithms are often suited for optimization problems, but require advancements for tracking objects in video. This paper presents an improved algorithm based on Bacterial Foraging Optimization in order to track multiple objects in real-time video exposed to full and partial occlusion, using video from both fixed and moving cameras. A comparison with various algorithms is provided.

Track: Real World Application

Categories and Subject Descriptors

I.4.8 [**Image Processing and Computer Vision**]: Scene Analysis—*Tracking*; I.5.4 [**Pattern Recognition**]: Applications—*Computer vision*

General Terms

Algorithms

Keywords

multi-object tracking, swarm intelligence, bacterial foraging optimization, non-stationary video, occlusion

1. INTRODUCTION

Tracking in non-stationary video is a challenging problem and approaches to addressing it include image stabilization and motion segmentation [6, 9]. Non-stationary video may be the result of active camera movements (such as panning, tilting, or movement by a camera operator) or from cameras mounted on moving platforms (such as from aerial, ground, or sea-faring vehicles). This paper uses a different approach and presents an algorithm based on Bacterial Foraging Optimization (BFO) [12] which is used to track multiple objects in a single camera, treating camera motion as a part of the object motion.

GECCO'09, July 8–12, 2009, Montréal Québec, Canada.

2. RELATED WORK AND CONTRIBUTIONS

Popular approaches toward tracking in the computer vision literature include algorithms such as Mean Shift [5], CamShift [3], Kalman filters [7], and Particle filters [8]. Mean Shift and CamShift perform tracking based on distributions (such as tracking based on a color histogram or texture features) and centering the current search window at the mean of the previous window. Though simple to implement, it is not robust to factors such as fast movement or occlusion. Kalman filters are proved to be optimal in the sense of minimum square error, but only when the noise is Gaussian; the dynamical model used in Kalman filters is assumed to be linear and does not work well when the noise is multi-modal. Particle filters evolve from Kalman filters and focus on dealing with non-linear dynamical models and multi-modal densities by sampling the prior probability and weighting those samples based on the observation. Thus, it can recover from occlusion to some extent provided that there are enough particles on the image. In practice, however, the larger the number of particles, the higher the computational cost, thereby making real-time processing challenging to implement.

Other swarm intelligence algorithms have also been considered for tracking, however. For instance, Particle Swarm Optimization (PSO) [10] has been considered for object tracking, applicable to both single and multiple targets [11]. These approaches often involve running the algorithm to convergence for every single frame.

Unlike traditional optimization problems with stationary optima, tracking objects through video requires the algorithm to find the object not once, but in potentially every frame. With PSO, there are various approaches to dealing with moving objects, such as periodically giving a swarm amnesia [4]. Decaying the score of the best location by some percentage p_{decay} after every frame can also be used to force the swarm to continually search for a better location. Such methods are used to prevent the swarm from completely converging to a single point, allowing the swarm agents to be appropriately spaced in order to quickly reacquire a target in the event that it moves in the next frame.

The contributions of this paper are:

- the development of an improved Bacterial Foraging algorithm with improved performance for multiple object tracking in video
- performance comparison of various swarm intelligence algorithms for tracking multiple objects in video
- comparison of the tracking performance of the new algorithm against a standard computer vision algorithm such as CamShift [3]

3. TECHNICAL APPROACH

3.1 Tracking Multiple Objects in Video

Multi-object tracking is a challenging problem for a number of reasons. For instance, a tracker should be able to:

- handle partial and full occlusion
- scale to a potentially large number of objects
- operate in a real-time environment

Occlusion will likely occur as a result of object and scene interaction. In addition, the occlusion introduced by having multiple moving objects can occur arbitrarily; one cannot rely on pre-established entrance and exit points. Processing in real-time is also a highly desirable capability, as off-line analysis removes the ability of responding (such as panning or tilting to maintain the object in the camera's field of view). There are other approaches to the problem, but Bacterial Foraging Optimization (BFO) has yet to be considered for multi-object tracking in video.

3.2 Bacterial Foraging Optimization for Object Tracking

Bacterial Foraging Optimization possesses several traits which make it suitable to the problem. For instance, while tracking, objects can disappear at one location in a frame and appear in another location for arbitrary lengths of time. Similarly, BFO relocates members of its swarm to random points in the search space to address the need for adapting to a changing environment. In order to understand the proposed changes to the algorithm, however, it is necessary to discuss how BFO works.

Bacterial Foraging Optimization is a stochastic evolutionary search algorithm modeled after the behavior of *E.coli* bacteria [12]. A swarm consists of i bacteria particles or *agents* which "swim" and "tumble" through an environment looking for concentrations of food.

Swimming translates to each agent moving through the search space in steps of size C and a *run* consists of a series of (up to) N_s swims. At the end of each swim, an agent surveys the concentration of food at the current position, which is akin to evaluating the fitness $J^i_{current}$ of a position. At the end of each run, an agent tumbles, or uses its flagella to randomly rotate itself in a new direction. During a run, agents stop swimming when they notice that food concentration begins to decrease ($J^i_{current} < J^i_{previous}$), otherwise swimming in the same direction up to N_s times. In this way, the bacteria particles have programmed themselves to climb gradients.

```
for up to k Reproductions do
    J_best ← 0;
    foreach Agent i do
        J^i_health ← 0;
    for up to j Chemotactic steps do
        foreach Agent i do
            Calculate J^i_current (higher is better);
            J^i_health ← J^i_health + J^i_current;
            Tumble in random direction;
            for up to N_s swims do
                J^i_previous ← J^i_current;
                Move i forward in step size C;
                Calculate new J^i_current;
                J^i_health ← J^i_health + J^i_current;
                if J^i_current > J_best then
                    J_best ← J^i_current;
                    J_best_location ← current location;
                if J^i_current < J^i_previous then
                    break;
    Sort agents by J_health in descending order;
    Remove bottom S_r agents with worst health;
    foreach Agent j in top S_r do
        Create clone of j at same location;
foreach Agent i do
    Relocate i with probability p_ed to a random location
    in the search space;
```

Algorithm 1: Classical Bacterial Foraging Optimization (executed once per frame).

Swimming, however, consumes energy, and agents who are unable to find food eventually starve to death. This reflects in a reproduction step which models an agent's health J^i_{health} as a sum of how much food it has consumed in the current run and killing off the S_r worst bacteria due to malnutrition (relative to the swarm). Conversely, the S_r agents with the best health go on to reproduce. In practice, this reproduction step essentially relocates the S_r worst agents to the location of the S_r best agents and keeps the size of the swarm constant.

Finally, all agents are subjected to an elimination-dispersal step where each agent is suddenly relocated with probability p_{ed} to a random location in the search space. This is done to simulate a dynamic environment which can displace agents (such as being introduced to a liquid). This important step ensures that the swarm does not fully converge and that the search space remains covered. Algorithm 1 describes the traditional Bacterial Foraging Optimization algorithm.

When tracking t multiple objects, the algorithm is executed t times per frame (once for each target, each with a respective fitness function). The agent(s) with the best fitness scores at the end of an execution can be considered as the swarm's best guess of a target's location.

3.3 Details of Improved BFO Algorithm

While traditionally suited to searching for static or slow-moving optima, BFO can also be adapted to tracking fast-moving objects in video. We contribute 3 major modifications:

1. **Lookahead**. Note that it is only after agents have swam

```
if fitness at previous J_best_location ≤ T_t then
    return;
for up to k Reproductions do
    J_best ← 0;
    foreach Agent i do
        J^i_health ← 0;
    for up to j Chemotactic steps do
        foreach Agent i do
            if i_immunity ≠ true then
                Calculate J^i_current (higher is better);
                J^i_health ← J^i_health + J^i_current;
                Tumble in random direction;
                for up to N_s swims do
                    Look ahead one step forward of size
                    C;
                    Calculate J^i_potential;
                    if J^i_potential ≥ J^i_current then
                        Move i to new position;
                        J^i_previous ← J^i_current;
                        J^i_current ← J^i_potential;
                        J^i_health ← J^i_health + J^i_current;
                        if J^i_current > J_best then
                            J_best ← J^i_current;
                            J_best_location ← current
                            location;
                    else
                        break;
    Sort agents by J_health in descending order;
    Remove bottom S_r agents with worst health;
    foreach Agent i in top S_r do
        i_immunity ← true;
        Create clone of i at same location;
foreach Agent i do
    if i_immunity ≠ true then
        Relocate i with probability p_ed to a random
        location in the search space;
    i_immunity ← false;
```

Algorithm 2: Improved Bacterial Foraging Algorithm for tracking (executed once per frame).

one step of size C that they evaluate their current position. While it may be more realistic, it also leaves agents in a known suboptimal position (the previous position was better). This can be modified so that agents are given the ability of looking ahead and calculating the $J_{potential}$ of that position, and seeing whether or not they should move. This has the effect of agents preferring to stay within boundaries of higher fitness (which is useful when objects have a clear boundary or steep gradients; this is largely impacted by the step size C). Doing so allows the algorithm to move in bigger steps without worrying about overstepping or stepping out of a good fitness area.

2. **Elitism**. Note that all agents move at every reproduction step, including agents who were already deemed to have the best health on previous reproductions within the same frame. Since frames do not change across sequential reproduction steps, stopping these k best agents (one from each reproduction) from further movement in the current frame has been found to improve the algorithm's accuracy. In addition, the swarm's best guess is selected from this group of agents as opposed to the most recent best agent. Finally, granting these best agents with immunity to the elimination-dispersal step allows the swarm to keep its best agents on the target and having everyone else spread out.

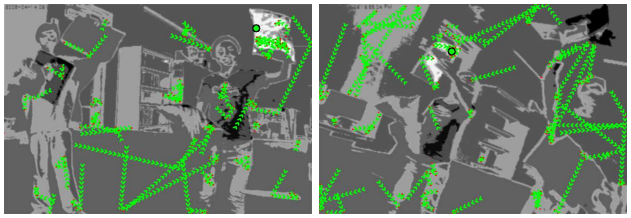


Figure 1: Example behavior of agents in the Bacterial Foraging Optimization algorithm at the end of a reproduction step. Green arrows depict agents swimming across the feature space and red dots show their destination. Agents systematically climb gradients by moving from darker regions (lower fitness) to brighter regions (higher fitness).

3. **Early Termination**. The swarm is inherently insatiable and thus always on the move; agents who are already on what would be considered to be the best location will continue to move in succeeding frames. This modification halts the swarm if the previous best location maintains a fitness score which exceeds threshold T_t (which can be determined dynamically with a 2-class Bayesian classifier on the previous best scores for the object).

Algorithm 2 incorporates the enhancements to the algorithm for video tracking. Figure 1 is an example of the swarm behavior in the feature space of a video frame.

4. EXPERIMENTAL RESULTS

4.1 Data

Three 30-second videos were recorded using an Axis PTZ-215 camera at 30 FPS (900 JPEG frames) and 704x480 resolution. Figures 2-4 shows samples of all three videos. The first video attempts to track 3 objects in a stationary camera (a red pillow, a green pillow, and a person wearing a dark blue jacket). The second video consists of 4 moving objects in another stationary setting (a red pillow, a green pillow, a yellow bag, and a person wearing a blue shirt). Finally, the third video consists of the same 4 objects with the camera in constant motion. All three videos include partial and full occlusion of all targets.

The fitness function used in this paper is a 20-bin color histogram [1] consisting of eight $Blue - Green$ and eight $Green - Red$ bins (chrominance) and four $Red + Green + Blue$ bins (luminance), sampled in a 9×9 circle centered on the evaluated point. That is, the RGB values of the pixels in the neighborhood around an evaluated point are decomposed into 20 bins (16 chrominance and 4 luminance bins) and fitness matching consists of enumerating the bins and measuring the percentage overlap between the evaluated histogram and the model histogram. In order to take advantage

Figure 2: Three targets in a stationary camera (red pillow, green pillow, and blue jacket and pillow)

Figure 3: Four targets in a stationary camera (red pillow, green pillow, person wearing a blue shirt, and yellow bag)

Figure 4: Four targets in a moving camera (red pillow, green pillow, person wearing a blue shirt, and yellow bag)

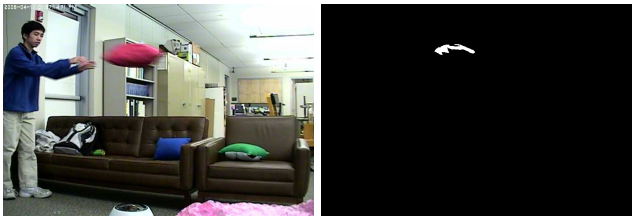


Figure 5: Ground truth location of a moving red object according to color classifier.

of all available frames, we manually initialize the model histogram on the center of an object at the start of a video, though it can also be done automatically using object detection (e.g., initializing the model on the centroid of detected moving objects).

4.2 Parameters

Due to the introduction of Elitism, it is beneficial to use a high elimination-dispersal probability $P_{ed} = 50\%$. This has the effect of freezing the best agents in place and spreading other agents out, thereby increasing the coverage of the search space. For the classic BFO algorithm however, a low $p_{ed} = 2\%$ is chosen to allow the best agents to maintain their locations (without immunity to the elimination-dispersal step, a swarm is easily dispersed from a target). Similarly, a larger step size $C = 10px$ is selected for the proposed algorithm to allow agents to climb gradients more quickly, whereas it is set to a lower $C = 2px$ in classical BFO since there is a higher risk of stepping out of a good area of fitness. Also, the number of chemotaxis steps $j = 1$ was experimentally determined after discovering higher tracking accuracy could consistently be achieved with the same computational cost by instead increasing the number of reproduction loops k.

For the experiments, the number of agents is set to 50 per target. Constant across the classic and proposed BFO tests, each reproduction step relocates $S_r = 1$ agent, the number of swims $N_s = 25$, and the number of reproduction steps $k = 10$. The parameters were experimentally chosen to enable the classic BFO configuration to achieve slightly above real-time performance of 30 FPS.

4.3 Implementation

The tested algorithms are implemented in C++ and are performed on a 2.4GHz Intel Core2 CPU with no additional multi-threading optimizations. Also tracking using a color histogram, the implementation of the CamShift algorithm is provided via the OpenCV library [2].

4.4 Evaluation Metrics

4.4.1 Tracking accuracy

Measured as the percentage of frames in which the swarm correctly locates the object. This is calculated as:

$$\text{accuracy} = \frac{\#\text{ of frames object correctly located}}{\text{total }\#\text{ of frames object is visible}} \quad (1)$$

Ground truth object locations are first established off-line. For all frames, the fitness function is exhaustively executed at every pixel and a Bayesian classifier is used on the fitness scores to segment the images. Figure 5 shows the resulting binary image produced for a single object.

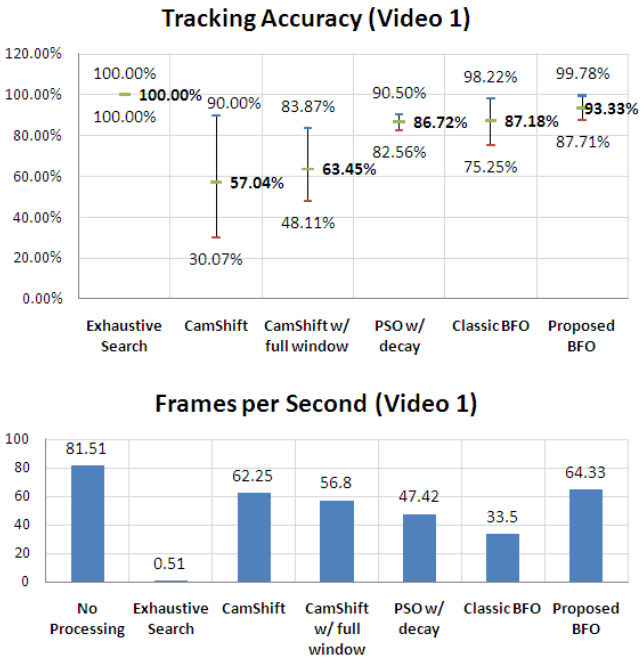


Figure 6: Performance comparison on tracking 3 targets in a stationary camera, averaged over 5 tests. Parameters: PSO (2,048 agents/target, 1% decay), Classic BFO (50 agents/target, $k = 10$, $j = 1$, $N_s = 25$, $p_{ed} = 1\%$, $C = 2px$), Proposed BFO (50 agents/target, $k = 10$, $j = 1$, $N_s = 25$, $p_{ed} = 50\%$, $C = 10px$)

4.4.2 Frames per second (FPS)

Measures frames processed per second. Note that experiments are executed on offline data in order to achieve frame rates unrestricted by limitations of the camera hardware.

4.5 Results

4.5.1 Comparison with PSO and standard BFO

Figures 6-8 show the tracking performance of the algorithms on the 3 respective datasets. The top and bottom whiskerbars represent the maximum and minimum and the green bar displays the mean.

We see that the proposed BFO algorithm maintains a higher tracking accuracy against classical BFO as well as PSO. Speed improvements can also be seen, with the proposed changes being capable of producing 65-92% speed improvements over classical BFO and 35-44% speed improvements over PSO. Figure 9 shows the incremental performance gain of each proposed improvement on an example configuration. Lookahead capabilities introduce a significant gain for accuracy as agents remain on their target easier (also resulting in a speed gain since fewer agents remain searching). Elitism similarly yields speed and accuracy gains (fewer agents continue searching and a larger pool of candidate positions to choose from, respectively). Early termination produces only modest accuracy gains (note that it maintains the large deviation in tracking accuracy) while speed improvements are slightly more substantial.

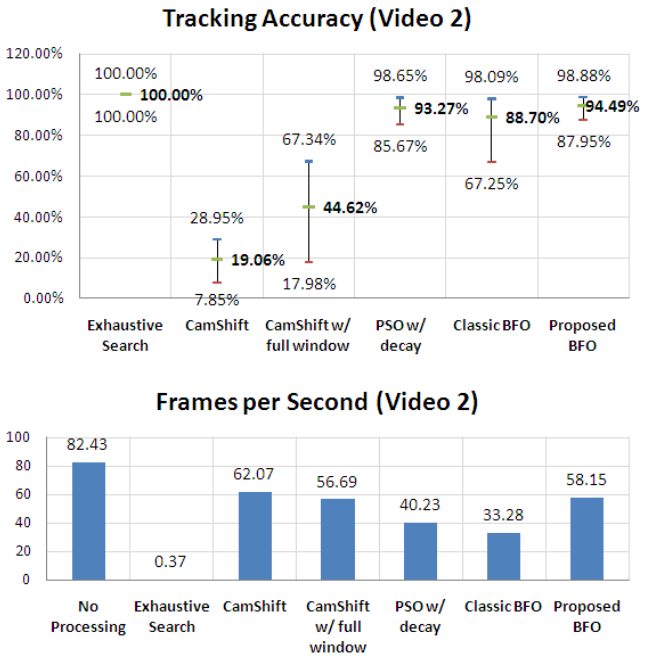


Figure 7: Performance comparison on tracking 4 targets in a stationary camera, averaged over 5 tests. Parameters: PSO (2,048 agents/target, 1% decay), Classic BFO (50 agents/target, $k = 10$, $j = 1$, $N_s = 25$, $p_{ed} = 1\%$, $C = 2px$), Proposed BFO (50 agents/target, $k = 10$, $j = 1$, $N_s = 25$, $p_{ed} = 50\%$, $C = 10px$)

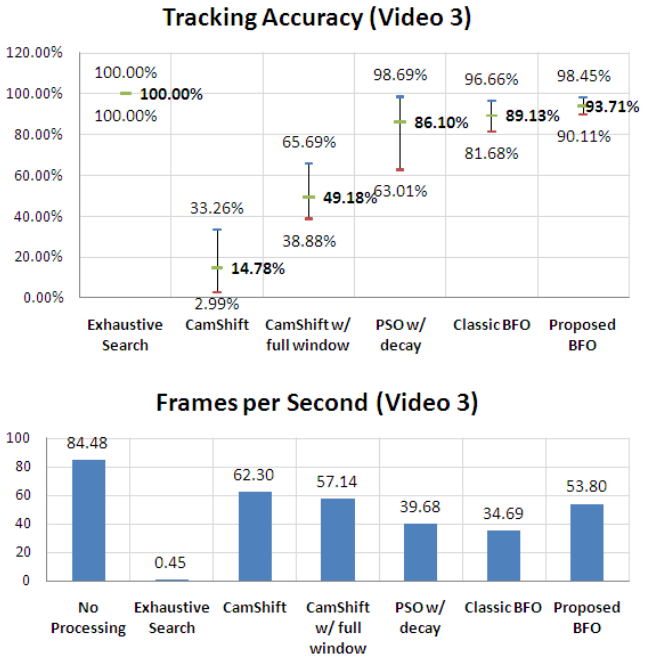
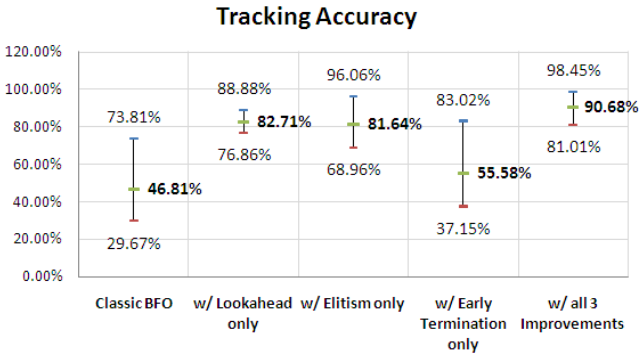


Figure 8: Performance comparison on tracking 4 targets in a moving camera, averaged over 5 tests. Parameters: PSO (2,048 agents/target, 1% decay), Classic BFO (50 agents/target, $k = 10$, $j = 1$, $N_s = 25$, $p_{ed} = 1\%$, $C = 2px$), Proposed BFO (50 agents/target, $k = 10$, $j = 1$, $N_s = 25$, $p_{ed} = 50\%$, $C = 10px$)



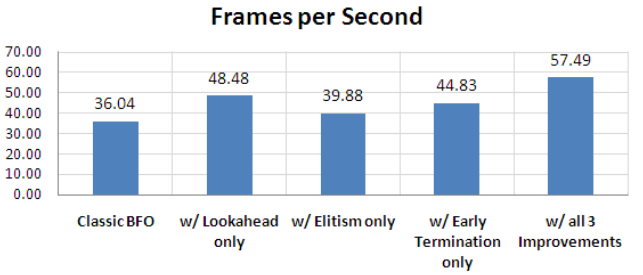


Figure 9: Incremental contribution of each improvement on a BFO configuration using Video 3. Parameters: 50 agents/target, $k = 10$, $j = 1$, $N_s = 25$, $p_{ed} = 15\%$, $C = 10px$

Figure 10 shows a comparison between the different tracks produced by the algorithms. In comparison, the proposed BFO algorithm produced tracks which are only marginally less noisy than classic BFO, but comparable to PSO.

4.5.2 *Scaling with Multiple Objects*

We have performed experiments on tracking 3 and 4 objects in both a stationary and moving camera. As the number of objects increases however, it is obvious that with limited resources (time and computational power), one must ultimately choose between tradeoffs in speed and accuracy. Indeed, with n frames in a video, both the classic BFO and proposed BFO algorithms have a time complexity of $O(itkjN_s)$ (where i = number of agents, t = number of targets, k = number of reproductions, j = number of chemotaxis steps, and N_s = maximum number of continuous swims in a run).

From a theoretical perspective, however, there is nothing inherent to swarm intelligence which limits its performance in tracking, but the limitation lies instead in the ability to distinguish between the different objects (e.g., the classifier, feature set, or fitness functions being evaluated). An extreme case involves tracking in a video in which every pixel in the image is an object to track. As the number of objects increases, it is easy to see that attempts to address scaling must produce a classifier which can scale in its ability to distinguish as well as in an algorithm's time complexity.

4.5.3 *Comparison of Proposed BFO with CamShift*

Figures 6-8 include performance comparisons to the CamShift algorithm. In order to improve its robustness to occlusion and loss of track, we also allow CamShift a full search window for each frame.

Note that since it is not a stochastic algorithm, repeated

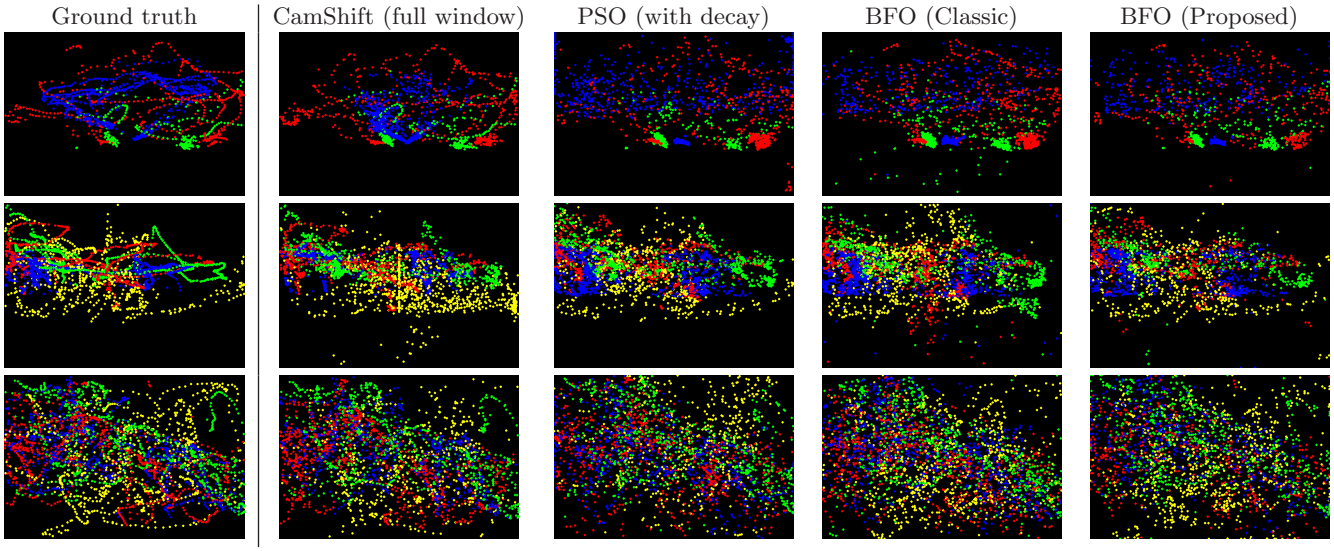


Figure 10: Comparison of target tracks for 900 frames. Each color represents its respective target. Note that while the proposed BFO achieves improved performance, the tracks are not smooth by default. This may be addressed by including a metric in the fitness function which encourages smooth tracks. The 4 columns from left to right: ground truth track, CamShift (full window), PSO w/ decay (2,048 agents/target, 1 step/frame, 1% decay), Classic BFO (50 agents/target, $k = 10$, $j = 1$, $N_s = 25$, $p_{ed} = 1\%$, $C = 2px$), Proposed BFO (50 agents/target, $k = 10$, $j = 1$, $N_s = 25$, $p_{ed} = 50\%$, $C = 10px$). The 3 rows from top to bottom: videos 1, 2, and 3.

tests of the CamShift tracker yield the same tracking accuracy results for the same targets. The spread in the mean, min, and max of the CamShift tests reflects the variance in performance between individual objects in the same video.

Performance-wise, the proposed BFO algorithm located targets more accurately than CamShift, particularly when CamShift uses only a local search window. This is because the algorithm expresses difficulty in reacquiring targets following brief periods of occlusion or once it has lost a target due to quick movement. Full window searching, however, enables it to work around occlusion and produces considerably better results. However, PSO, classic BFO, and the proposed BFO were able to produce more-accurate track locations; though CamShift was capable of generating smoother tracks (see Figure 10), it still has issues when presented with fitness regions which are multi-modal.

5. CONCLUSIONS

The results show that the proposed improvements to the BFO algorithm can be used to track objects through occlusion and uncalibrated camera movement, as well as outperform Particle Swarm Optimization using fewer resources. Future work extends this approach to track multiple objects through multiple cameras.

6. ACKNOWLEDGMENTS

The authors would like to thank Yiming Li for her helpful discussions and participation in data collection. This work was supported by NSF grants 0622176 and 0551741.

7. REFERENCES

[1] S. Birchfield. Elliptical head tracking using intensity gradients and color histograms. *CVPR 1998*, pages 232–237, 1998.

[2] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[3] G. R. Bradski. Computer vision face tracking for use in a perceptual user interface. *Intel Technology Journal Q2 '98*, 1998.

[4] A. J. Carlisle. *Applying the Particle Swarm Optimizer to Non-stationary Environments*. PhD thesis, Auburn University, 2002.

[5] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. *CVPR 2000*, 2:142–149 vol.2, 2000.

[6] W. Ding, Z. Gong, S. Xie, and H. Zou. Real-time vision-based object tracking from a moving platform in the air. *IEEE/RSJ Int. Conf. on Robots and Systems*, pages 681–685, 2006.

[7] D. A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, August 2002.

[8] M. Isard and A. Blake. CONDENSATION - conditional density propagation for visual tracking. *IJCV*, 29:5–28, 1998.

[9] J. Kang, I. Cohen, G. Medioni, and C. Yuan. Detection and tracking of moving objects from a moving platform in presence of strong parallax. *ICCV 2005*, 1:10–17 Vol. 1, 2005.

[10] J. Kennedy and R. Eberhart. Particle swarm optimization. *Neural Networks, IEEE Int. Conf.*, 4:1942–1948 vol.4, 1995.

[11] Y. Owechko and S. Medasani. Cognitive swarms for rapid detection of objects and associations in visual imagery. *Swarm Intelligence Symposium, IEEE*, pages 420–423, 2005.

[12] K. M. Passino. Biomimicry of bacterial foraging for distributed optimization and control. *IEEE Control Systems Magazine*, Vol. 22, No. 3:52–67, 2002.