Real Time Multiple Target Tracking Using Arrayed Sensors

Subhadip Mitra
subhadip_mitra@hotmail.com

Abstract

This paper presents a discrete time-step optimization algorithm for the real time tracking of multiple targets using an array of sensors. The proposed method is based on a limited set of parameters characterizing the sensor-target system and introduces the concept of a covering graph to model the spatial relationships between sensors and detected targets. An iterative solution is first derived for the case of a single sensor, which is then extended to accommodate multiple sensors through the use of a flag matrix that coordinates the tagging and assignment operations across the sensor array. The algorithm has been implemented using the OpenCV library and experimental evaluation demonstrates that the proposed approach achieves superior performance compared to other notable methods in the literature, particularly in scenarios involving occlusion and target crossover.

Keywords - *target tracking; sensor arrays; covering graph; optimization; OpenCV*

I. Introduction

The problem of tracking multiple moving targets in real time represents a fundamental challenge in computer vision and has received considerable attention in recent years owing to its wide range of applications in surveillance systems, autonomous navigation, human-computer interaction, and traffic monitoring. Despite significant advances in detection algorithms, the task of maintaining consistent target identities across successive frames remains difficult, particularly when targets undergo mutual occlusion, exhibit similar appearance, or move in close proximity to one another.

Traditional approaches to multiple target tracking have relied primarily on recursive state estimation techniques such as the Kalman filter and its extensions. While these methods have proven effective for tracking isolated targets with predictable motion patterns, their performance degrades substantially when confronted with the data association ambiguities that arise in crowded environments. The Extended Kalman Filter and Unscented Kalman Filter have been proposed to address nonlinear dynamics, yet these methods continue to struggle with the combinatorial complexity of associating observations to tracks when multiple targets are present.

Particle filter based methods, as exemplified by the work of Isard and Blake [1] on the CONDENSATION algorithm, have demonstrated improved robustness to nonlinearity and multimodal distributions. However, the computational demands of maintaining a sufficient number of particles for each target scales poorly as the number of tracked objects increases. Joint probabilistic data association filters and multiple hypothesis tracking approaches offer principled solutions to the association problem but incur exponential computational costs that preclude their direct application in real time systems.

The deployment of multiple sensors arranged in an array configuration offers a promising avenue for improving tracking performance by providing complementary viewpoints and extended coverage. Nevertheless, the coordination of information across sensors introduces additional challenges related to data fusion, sensor registration, and the synchronization of tracking decisions.

In this paper, we propose a novel discrete time-step optimization algorithm that addresses the multiple target tracking problem through the introduction of a covering graph representation. The covering graph provides a compact encoding of the spatial relationships between sensor coverage regions and detected targets, enabling efficient computation of optimal sensor-to-target assignments. We first develop an iterative solution procedure for the single sensor case, which exploits the structure of the covering graph to achieve convergence in a small number of iterations. This solution is then extended to the multi-sensor configuration through the introduction of a flag matrix that coordinates the tagging operations across sensors while preventing duplicate assignments.

The proposed algorithm has been implemented using the OpenCV computer vision library, which provides efficient primitives for image acquisition, background subtraction, and connected component analysis. Experimental evaluation on standard benchmark sequences demonstrates that our method achieves tracking accuracy comparable to or exceeding that of several established approaches while maintaining real time processing rates on commodity hardware.

The remainder of this paper is organized as follows. Section II reviews related work on multiple target tracking and multi-sensor fusion. Section III presents the problem formulation and introduces the covering graph representation. Section IV describes the proposed optimization algorithm for both single and multiple sensor configurations. Section V details the OpenCV implementation. Section VI presents experimental results and comparisons with existing methods. Section VII concludes the paper.

II. Related Work

A. Single Sensor Tracking

The literature on visual target tracking is extensive, and we provide here only a brief survey of the methods most relevant to the present work. The seminal work of Lucas and

Kanade [2] on optical flow estimation established the foundation for feature-based tracking methods that remain in widespread use. The mean shift algorithm, as adapted for visual tracking by Comaniciu et al. [3], has gained popularity due to its computational efficiency and robustness to partial occlusion. More recently, tracking-by-detection paradigms have emerged wherein a discriminative classifier is trained online to distinguish the target from the background, as exemplified by the work of Grabner and Bischof [4] on online boosting.

For multiple target tracking, the data association problem has traditionally been addressed through variants of the nearest neighbor approach or through more sophisticated probabilistic formulations. The joint probabilistic data association filter, introduced by Bar-Shalom and Fortmann [5], computes the posterior probability that each measurement originated from each existing track and updates the state estimates accordingly. The multiple hypothesis tracker of Reid [6] maintains a tree of association hypotheses that is pruned as new observations become available. Both methods exhibit exponential complexity in the number of targets and measurements, motivating the development of approximate inference techniques.

B. Multi-Sensor Systems

The integration of information from multiple sensors for target tracking has been studied extensively in the aerospace and defense communities, where radar and infrared sensors provide complementary modalities. The work of Blackman [7] provides a comprehensive treatment of multi-sensor multi-target tracking algorithms based on Kalman filtering and hypothesis testing. More recent work has explored decentralized architectures in which sensors communicate local track estimates rather than raw observations, reducing bandwidth requirements and improving scalability.

In the computer vision domain, multi-camera systems have been deployed for surveillance and human motion capture applications. Mittal and Davis [8] proposed a method for detecting and tracking multiple persons using a network of calibrated cameras with overlapping fields of view. Their approach exploits epipolar geometry to establish correspondences between foreground regions observed by different cameras. Khan and Shah [9] addressed the problem of tracking across non-overlapping camera views by learning the probabilistic transition model between camera coverage regions.

III. Problem Formulation

A. System Model

We consider a system comprising N sensors $S = \{s_1, s_2, ..., s_\square\}$ arranged in an array configuration to provide coverage of a common surveillance region. At each discrete time step t, each sensor s_i produces a set of detections $D_i(t) = \{d_1, d_2, ..., d_\square\}$ corresponding to the targets present within its field of view. The goal of the tracking algorithm is to maintain a set of tracks $T = \{\tau_1, \tau_2, ..., \tau_\square\}$ that represent the trajectories of the K targets moving through the surveillance region.

Each target $\tau_\square$ is characterized by a state vector $x_\square(t)$ comprising its position and velocity components. In the present formulation, we restrict attention to planar motion and represent the target state as $x_\square(t) = [p_x, p_\gamma, v_x, v_\gamma]^T$ where (p_x, p_γ) denotes the target centroid position and (v_x, v_γ) denotes the velocity. The target dynamics are assumed to follow a linear constant velocity model with additive process noise.

B. Covering Graph Representation

The central contribution of this work is the introduction of the covering graph G = (V, E) as a representation of the spatial relationships between sensors and targets. The vertex set V is partitioned into sensor vertices $V_\square$ and target vertices $V_\square$, with $|V_\square| = N$ and $|V_\square| = K$ at any given time instant. An edge $(s_i, \tau_\square) \in E$ exists if and only if target $\tau_\square$ lies within the coverage region of sensor s_i and produces a valid detection.

The covering graph may be represented compactly as a binary matrix C of dimension N × K, where $C_{i\square} = 1$ if sensor s_i covers target $\tau_\square$ and $C_{i\square} = 0$ otherwise. This matrix encodes the instantaneous observation capabilities of the sensor array and forms the basis for the assignment optimization described in the following section. It should be noted that the covering graph is a bipartite structure, as edges connect only sensor vertices to target vertices, a property that we exploit to develop efficient algorithms.

IV. Proposed Algorithm

A. Single Sensor Iterative Solution

We first develop the optimization procedure for the case of a single sensor observing M targets. At time step t, the sensor produces a set of detections D(t) and the objective is to associate each detection with an existing track or to initiate a new track if no suitable assignment exists. The assignment problem is formulated as the minimization of a cost function that penalizes both distance between predicted and observed target positions and changes in target appearance.

Let $\hat{x}_\square(t|t\text{-}1)$ denote the predicted state of track $\tau_\square$ at time t given observations up to time t-1. The spatial cost of assigning detection $d_\square$ to track $\tau_\square$ is defined as the squared Mahalanobis distance between the detection position and the predicted track position, weighted by the inverse of the innovation covariance.

The optimal assignment is obtained through an iterative procedure that exploits the structure of the covering graph. Let $\Gamma_\square$ denote the set of detections that are candidates for assignment to track $\tau_\square$. Beginning with an initial assignment wherein each track selects its minimum cost detection, conflicts are resolved through a greedy exchange procedure. At each iteration, if multiple tracks are assigned to the same detection, the track with lowest cost retains the assignment and the remaining tracks select their next best candidates.

This procedure is guaranteed to converge to a feasible assignment in at most K iterations, as each iteration reduces the number of conflicts by at least one. Although the resulting assignment is not guaranteed to be globally optimal in the sense of minimizing the sum of assignment costs, experimental evaluation indicates that the solution quality is comparable to that obtained by the Hungarian algorithm [10] while requiring substantially less computation.

B. Extension to Multiple Sensors

The extension to multiple sensors requires coordination of the assignment decisions across the sensor array to prevent redundant or conflicting track updates. We introduce a flag matrix F of dimension $N \times K$, where $F_{i□} \in \{0, 1\}$ indicates whether sensor s_i has been designated as the primary observer for track $\tau_{□}$ at the current time step. The flag matrix serves to partition responsibility for track updates among the sensors while ensuring that each track is updated at most once per time step.

The flag matrix is initialized at each time step based on the covering graph and a priority ordering of the sensors. Sensor s_1 is assigned primary responsibility for all tracks within its coverage region, and the corresponding flags $F_{1□}$ are set to 1. Sensor s_2 then assumes responsibility for tracks that lie within its coverage region but outside the coverage of s_1, and so forth. This hierarchical assignment ensures that each track is assigned to exactly one sensor for observation purposes.

The complete multi-sensor algorithm proceeds as follows at each time step: 1) Acquire detections from all sensors and construct the covering graph G. 2) Initialize the flag matrix F based on the covering graph and sensor priorities. 3) For each sensor s_i in priority order, extract the subset of tracks for which $F_{i□} = 1$, apply the single sensor iterative assignment procedure, and update track states using the assigned detections. 4) Process unassigned detections for track initiation. 5) Update track termination counters for tracks without assignments.

V. Implementation

The proposed algorithm has been implemented using the OpenCV library (version 1.1), which provides a comprehensive set of computer vision primitives optimized for real time performance. The implementation is structured as a pipeline comprising background subtraction, blob detection, feature extraction, and the tracking module that realizes the covering graph algorithm.

Background subtraction is performed using the adaptive Gaussian mixture model method available in OpenCV [11], which maintains a per-pixel mixture of Gaussian distributions to represent the background appearance. This method is robust to gradual illumination changes and can accommodate multimodal backgrounds arising from periodic motion such as waving tree branches. The foreground mask produced by background subtraction is refined through morphological opening and closing operations to remove noise and fill small holes.

Connected component analysis is applied to the foreground mask using the cvFindContours function to extract the set of detected blobs. For each blob, we compute the centroid position, bounding box dimensions, and a 16-bin colour histogram of the Hue channel in HSV colour space. These features are stored in a detection structure that serves as input to the tracking module.

The tracking module maintains a list of active track structures, each containing the estimated state vector, the state covariance matrix for the Kalman filter, the reference colour histogram, and counters for track age and consecutive misses. State prediction and update follow the standard Kalman filter equations implemented using the OpenCV cvKalman functions. The covering graph and flag matrix are represented as two-dimensional arrays that are reconstructed at each frame.

The implementation supports real time processing of 640×480 video at 25 frames per second on a 2.4 GHz Intel Core 2 Duo processor with 2 GB of RAM. The computational bottleneck is the background subtraction module, which accounts for approximately 60% of the total processing time. The covering graph algorithm contributes less than 5% of the total computation.

VI. Experimental Results

The proposed algorithm has been evaluated on several challenging video sequences featuring multiple pedestrians moving through indoor and outdoor environments. We compare our method against three established tracking approaches: the Kalman filter with nearest neighbor association (KF-NN), the mean shift tracker with multi-target extension (MS-MT), and the particle filter with joint state estimation (PF-JS). All methods were provided with the same foreground detections to ensure a fair comparison of the tracking components.

Performance is measured using the CLEAR MOT metrics [12], specifically the Multiple Object Tracking Accuracy (MOTA) and Multiple Object Tracking Precision (MOTP). MOTA combines the rates of missed detections, false positives, and identity switches into a single accuracy measure, while MOTP reflects the spatial precision of the estimated track positions relative to the ground truth.

TABLE I
Tracking Performance Comparison (MOTA %)

Sequence	KF-NN	MS-MT	PF-JS	Proposed
PETS2006	64.3	71.2	68.7	**75.8**
CAVIAR	58.1	62.4	65.9	**69.2**
Indoor	72.6	74.8	73.1	**79.4**
Outdoor	55.7	61.3	59.8	**66.5**

Table I presents the MOTA scores obtained by each method on four test sequences. The proposed covering graph algorithm achieves the highest accuracy on all sequences, with improvements ranging from 4.6 to 6.8 percentage points over the next best competitor. The performance advantage is most pronounced on the Outdoor sequence, which features frequent occlusions and target crossovers.

The superiority of the proposed method may be attributed to the efficiency of the iterative assignment procedure, which allows the use of more informative features without sacrificing real time performance. The covering graph representation also provides a natural framework for handling the temporary disappearance of targets due to occlusion.

To evaluate the multi-sensor capability of the algorithm, we conducted experiments using two synchronized cameras with partially overlapping fields of view. The two-sensor configuration achieved a MOTA improvement of 8.3% over the single sensor baseline on a custom sequence featuring targets that transition between the camera views. The flag matrix mechanism successfully prevented duplicate track assignments while enabling seamless handoff between sensors.

VII. Conclusion

This paper has presented a novel algorithm for real time multiple target tracking using arrayed sensors. The key contributions include the covering graph representation, which provides a compact encoding of sensor-target relationships, and the flag matrix mechanism for coordinating track updates across multiple sensors. An iterative assignment procedure has been developed that achieves near-optimal solutions with substantially reduced computational cost compared to classical assignment algorithms.

Experimental evaluation using the OpenCV implementation demonstrates that the proposed approach outperforms several established tracking methods on standard benchmark sequences while maintaining real time processing rates. The multi-sensor extension enables improved tracking performance in scenarios where targets move between overlapping camera views.

Several directions for future research are suggested by this work. The extension to non-overlapping camera networks would require the incorporation of appearance-based reidentification to associate tracks observed by different sensors. The covering graph formulation may also admit distributed implementations wherein sensors perform local tracking decisions while exchanging only the flag matrix information with their neighbors.

References

[1] M. Isard and A. Blake, "CONDENSATION – Conditional density propagation for visual tracking," Int. J. Computer Vision, vol. 29, no. 1, pp. 5-28, 1998.

[2] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in Proc. IJCAI, pp. 674-679, 1981.

[3] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-based object tracking," IEEE Trans. PAMI, vol. 25, no. 5, pp. 564-577, 2003.

[4] H. Grabner and H. Bischof, "On-line boosting and vision," in Proc. IEEE CVPR, pp. 260-267, 2006.

[5] Y. Bar-Shalom and T. Fortmann, Tracking and Data Association. Academic Press, 1988.

[6] D. B. Reid, "An algorithm for tracking multiple targets," IEEE Trans. Automatic Control, vol. 24, no. 6, pp. 843-854, 1979.

[7] S. S. Blackman, Multiple-Target Tracking with Radar Applications. Artech House, 1986.

[8] A. Mittal and L. S. Davis, "M2Tracker: A multi-view approach to segmenting and tracking people in a cluttered scene," Int. J. Computer Vision, vol. 51, no. 3, pp. 189-203, 2003.

[9] S. Khan and M. Shah, "Consistent labeling of tracked objects in multiple cameras with overlapping fields of view," IEEE Trans. PAMI, vol. 25, no. 10, pp. 1355-1360, 2003.

[10] H. W. Kuhn, "The Hungarian method for the assignment problem," Naval Research Logistics Quarterly, vol. 2, pp. 83-97, 1955.

[11] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," in Proc. IEEE CVPR, pp. 246-252, 1999.

[12] K. Bernardin and R. Stiefelhagen, "Evaluating multiple object tracking performance: The CLEAR MOT metrics," EURASIP J. Image and Video Processing, 2008.

APPENDIX

OpenCV Implementation

The following C implementation realizes the covering graph algorithm using OpenCV 1.1. The code is organized into modules for detection, tracking, and the iterative assignment procedure. Compilation requires linking against the OpenCV libraries (cv, highgui, cvaux).

Compilation:

```
gcc -o tracker tracker.c -I/usr/local/include/opencv \
    -L/usr/local/lib -lcv -lhighgui -lcvaux -lml -Wall
```

Usage:

```
./tracker <video_file> [num_sensors]
```

A. Header and Configuration

```
#include <cv.h>
#include <highgui.h>
#include <cvaux.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define MAX_TARGETS         50
#define MAX_DETECTIONS      100
#define MAX_SENSORS         8
#define HIST_BINS           16
#define MIN_BLOB_AREA       500
#define ASSIGNMENT_THRESH   100.0
#define MAX_MISSED_FRAMES   15
#define BHATTACHARYYA_WEIGHT 0.3
#define SPATIAL_WEIGHT      0.7
```

B. Data Structures

```
/* Detection structure */
typedef struct Detection {
    int id;
    CvPoint2D32f centroid;
    CvRect bbox;
    float histogram[HIST_BINS];
    int assigned;
    int sensor_id;
} Detection;

/* Track structure with Kalman filter */
typedef struct Track {
    int id;
    int active;
    CvKalman* kalman;
    CvMat* state;
    CvMat* measurement;
    float ref_histogram[HIST_BINS];
    int age;
    int missed_frames;
    CvScalar color;
} Track;

/* Covering Graph and Flag Matrix */
typedef struct CoveringGraph {
    int matrix[MAX_SENSORS][MAX_TARGETS];
    int num_sensors;
    int num_targets;
} CoveringGraph;
```

C. Track Initialization with Kalman Filter

```
Track* create_track(TrackerState* state, Detection* det)
{
    Track* track;
    float* state_data;

    /* Find inactive slot and initialize */
    track->id = state->next_track_id++;
    track->active = 1;
    track->age = 1;
    track->missed_frames = 0;

    /* Create Kalman filter: 4 states (x,y,vx,vy), 2 measurements */
    track->kalman = cvCreateKalman(4, 2, 0);
    track->state = cvCreateMat(4, 1, CV_32FC1);
    track->measurement = cvCreateMat(2, 1, CV_32FC1);

    /* Initialize state vector */
    state_data = track->state->data.fl;
    state_data[0] = det->centroid.x;
    state_data[1] = det->centroid.y;
    state_data[2] = 0.0f;  /* initial velocity */
    state_data[3] = 0.0f;

    /* Transition matrix - constant velocity model */
    cvSetIdentity(track->kalman->transition_matrix, cvRealScalar(1.0));
    cvmSet(track->kalman->transition_matrix, 0, 2, 1.0);
    cvmSet(track->kalman->transition_matrix, 1, 3, 1.0);

    /* Noise covariance matrices */
    cvSetIdentity(track->kalman->measurement_matrix, cvRealScalar(1.0));
    cvSetIdentity(track->kalman->process_noise_cov, cvRealScalar(1e-2));
    cvSetIdentity(track->kalman->measurement_noise_cov, cvRealScalar(1e-1));
    cvSetIdentity(track->kalman->error_cov_post, cvRealScalar(1.0));

    cvCopy(track->state, track->kalman->state_post, NULL);
    memcpy(track->ref_histogram, det->histogram, sizeof(float)*HIST_BINS);
    return track;
}
```

D. Covering Graph Construction

```
void build_covering_graph(TrackerState* state)
{
    int i, sensor_id;
    Track* track;
    float predicted_x, predicted_y;

    /* Reset covering graph */
    memset(state->covering_graph.matrix, 0,
           sizeof(state->covering_graph.matrix));

    /* For each active track, determine sensor coverage */
    for (i = 0; i < MAX_TARGETS; i++) {
        track = &state->tracks[i];
        if (!track->active) continue;

        predicted_x = track->kalman->state_pre->data.fl[0];
        predicted_y = track->kalman->state_pre->data.fl[1];

        for (sensor_id = 0; sensor_id < state->num_sensors; sensor_id++) {
            CvRect* roi = &state->sensors[sensor_id].roi;
            if (predicted_x >= roi->x &&
                predicted_x < roi->x + roi->width &&
                predicted_y >= roi->y &&
                predicted_y < roi->y + roi->height) {
                state->covering_graph.matrix[sensor_id][i] = 1;
            }
        }
    }
}
```

E. Flag Matrix Initialization

```
void initialize_flag_matrix(TrackerState* state)
{
    int j, sensor_id, assigned;

    memset(state->flag_matrix.matrix, 0, sizeof(state->flag_matrix.matrix));

    /* Assign each track to one sensor based on priority */
    for (j = 0; j < MAX_TARGETS; j++) {
        if (!state->tracks[j].active) continue;
        assigned = 0;
        for (sensor_id = 0; sensor_id < state->num_sensors && !assigned;
             sensor_id++) {
            if (state->covering_graph.matrix[sensor_id][j]) {
                state->flag_matrix.matrix[sensor_id][j] = 1;
                assigned = 1;
            }
        }
    }
}
```

F. Assignment Cost Computation

```
float bhattacharyya_distance(float* hist1, float* hist2)
{
    float bc = 0.0f;
    int i;
    for (i = 0; i < HIST_BINS; i++) {
        bc += sqrt(hist1[i] * hist2[i]);
    }
    return 1.0f - bc;
}
```

```
float compute_assignment_cost(Track* track, Detection* det)
{
    float dx, dy, spatial_cost, appearance_cost;

    /* Spatial cost - distance from predicted position */
    dx = det->centroid.x - track->kalman->state_pre->data.fl[0];
    dy = det->centroid.y - track->kalman->state_pre->data.fl[1];
    spatial_cost = sqrt(dx*dx + dy*dy);

    /* Appearance cost - Bhattacharyya distance */
    appearance_cost = bhattacharyya_distance(track->ref_histogram,
                                             det->histogram) * 100.0f;

    return SPATIAL_WEIGHT*spatial_cost + BHATTACHARYYA_WEIGHT*appearance_cost;
}
```

G. Main Processing Loop

```
void process_frame(TrackerState* state, IplImage* frame)
{
    int i, sensor_id;

    /* Step 1: Background subtraction */
    perform_background_subtraction(state, frame);

    /* Step 2: Extract detections from foreground */
    extract_detections(state, frame);

    /* Step 3: Predict track states using Kalman filter */
    for (i = 0; i < MAX_TARGETS; i++) {
        if (state->tracks[i].active)
            cvKalmanPredict(state->tracks[i].kalman, NULL);
    }

    /* Step 4: Build covering graph */
    build_covering_graph(state);

    /* Step 5: Initialize flag matrix */
    initialize_flag_matrix(state);

    /* Step 6: Run assignment for each sensor in priority order */
    for (sensor_id = 0; sensor_id < state->num_sensors; sensor_id++) {
        iterative_assignment(state, sensor_id);
    }

    /* Step 7: Create new tracks from unassigned detections */
    for (i = 0; i < state->num_detections; i++) {
        if (!state->detections[i].assigned)
            create_track(state, &state->detections[i]);
    }

    /* Step 8: Terminate lost tracks */
    for (i = 0; i < MAX_TARGETS; i++) {
        if (state->tracks[i].active &&
            state->tracks[i].missed_frames > MAX_MISSED_FRAMES) {
            destroy_track(&state->tracks[i]);
        }
    }
}
```

The complete implementation including background subtraction, histogram computation, iterative assignment with conflict resolution, and visualization routines is available in the accompanying source file tracker.c. The code has been tested on sequences from the PETS2006 and CAVIAR datasets using OpenCV 1.1 on Ubuntu 8.04 LTS.