

Laboratory of Image Processing

Motion-Based Multiple Object Tracking

Pier Luigi Mazzeo
pierluigi.mazzeo@cnr.it

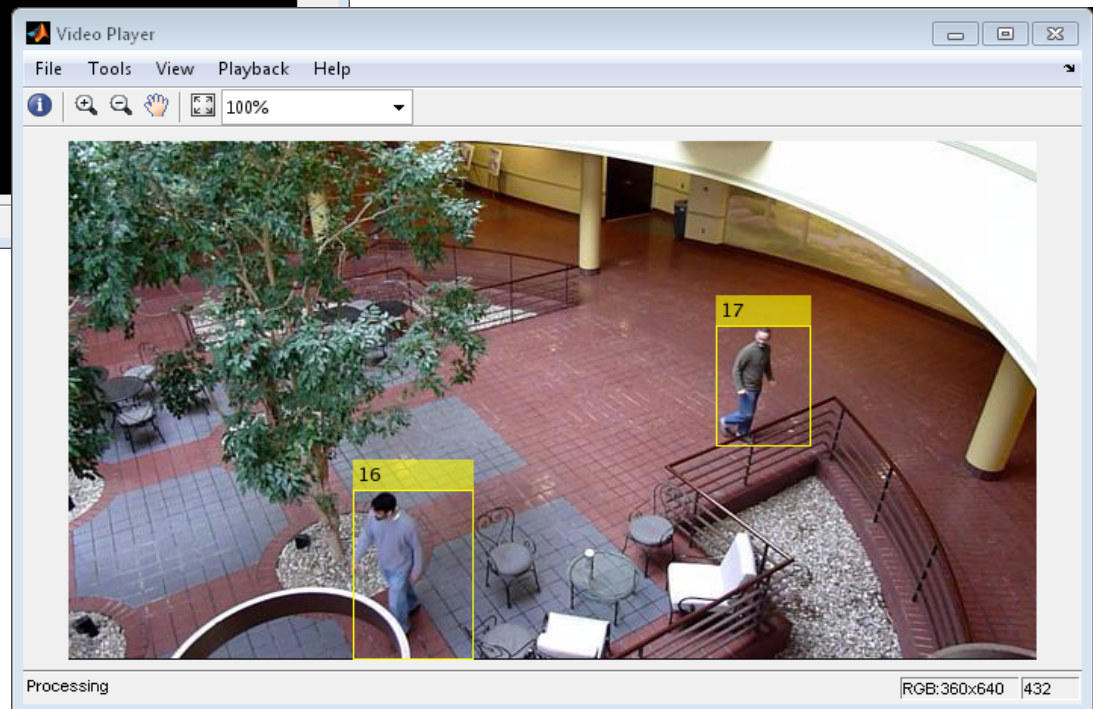
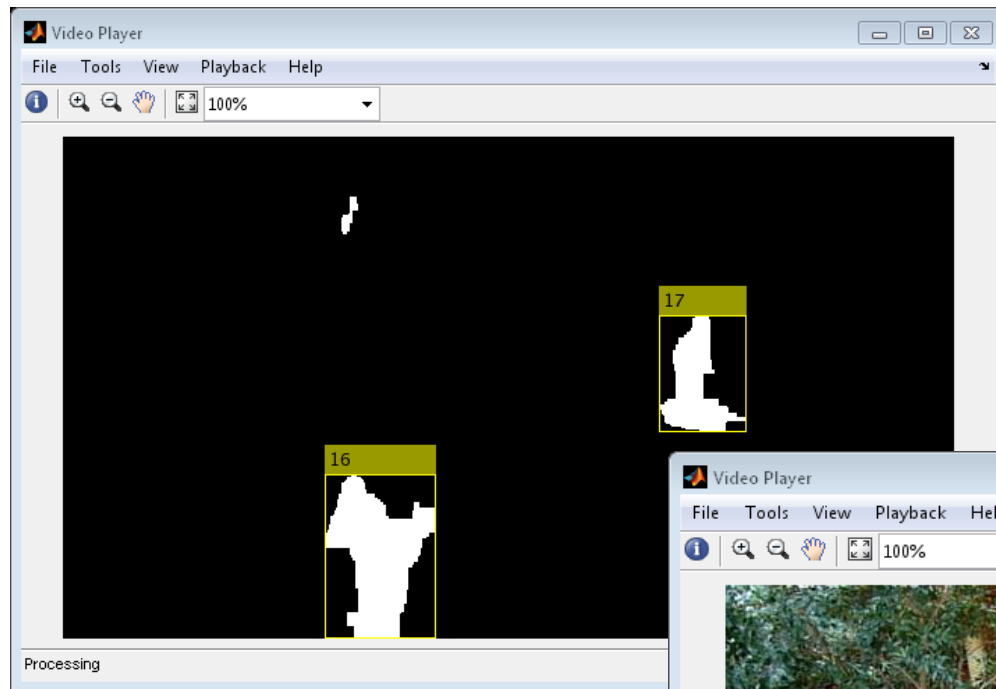
Multiple Object Tracking

- This example shows how to perform automatic detection and motion-based tracking of moving objects in a video from a stationary camera.
- Detection of moving objects and motion-based tracking are important components of many computer vision applications, including activity recognition, traffic monitoring, and automotive safety. The problem of motion-based object tracking can be divided into two parts:
 - detecting moving objects in each frame
 - associating the detections corresponding to the same object over time
- The detection of moving objects uses a background subtraction algorithm based on Gaussian mixture models. Morphological operations are applied to the resulting foreground mask to eliminate noise. Finally, blob analysis detects groups of connected pixels, which are likely to correspond to moving objects.
- The association of detections to the same object is based solely on motion. The motion of each track is estimated by a Kalman filter. The filter is used to predict the track's location in each frame, and determine the likelihood of each detection being assigned to each track.

Multiple Object Tracking

- Track maintenance becomes an important aspect of this example. In any given frame, some detections may be assigned to tracks, while other detections and tracks may remain unassigned. The assigned tracks are updated using the corresponding detections. The unassigned tracks are marked invisible. An unassigned detection begins a new track.
- Each track keeps count of the number of consecutive frames, where it remained unassigned. If the count exceeds a specified threshold, the example assumes that the object left the field of view and it deletes the track.
- This example is a function with the main body at the top and helper routines in the form of **nested functions**.

Multiple Object Tracking



Function multi Object Tracking

```
function multiObjectTracking()  
% create system objects used for reading video, detecting moving  
% objects, and displaying the results  
obj = setupSystemObjects();  
tracks = initializeTracks(); % create an empty array of tracks  
nextId = 1; % ID of the next track  
% detect moving objects, and track them across video frames  
while ~isDone(obj.reader)  
    frame = readFrame();  
    [centroids, bboxes, mask] = detectObjects(frame);  
    predictNewLocationsOfTracks();  
    [assignments, unassignedTracks, unassignedDetections] = ...  
        detectionToTrackAssignment();  
  
    updateAssignedTracks();  
    updateUnassignedTracks();  
    deleteLostTracks();  
    createNewTracks();  
    displayTrackingResults();  
end
```

Function Create System Objects

Create System objects used for reading the video frames, detecting foreground objects, and displaying results.

```
function obj = setupSystemObjects()  
% Initialize Video I/O  
%Create objects for reading a video from a file, drawing the  
%tracked objects in each frame, and playing the video.  
  
    % create a video file reader  
    obj.reader = vision.VideoFileReader('atrium.avi');  
  
    % create two video players, one to display the video,  
    % and one to display the foreground mask  
obj.videoPlayer = vision.VideoPlayer('Position', [20, 400, 700,  
400]);  
obj.maskPlayer = vision.VideoPlayer('Position', [740, 400, 700,  
400]);
```

Function Create System Objects (2)

Create system objects for foreground detection and blob analysis. The foreground detector is used to segment moving objects from the background. It outputs a binary mask, where the pixel value of 1 corresponds to the foreground and the value of 0 corresponds to the background.

```
obj.detector = vision.ForegroundDetector('NumGaussians', 3, ...  
    'NumTrainingFrames', 40, 'MinimumBackgroundRatio', 0.7);
```

% Connected groups of foreground pixels are likely to correspond to moving objects. The blob analysis system object is used to find such groups (called 'blobs' or 'connected components'), and compute their characteristics, such as area, centroid, and the bounding box.

```
obj.blobAnalyser =  
vision.BlobAnalysis('BoundingBoxOutputPort', ...  
    true, 'AreaOutputPort', true, 'CentroidOutputPort', true, ...  
        'MinimumBlobArea', 400);  
end
```

Initialize Tracks

The `initializeTracks` function creates an array of tracks, where each track is a structure representing a moving object in the video. The purpose of the structure is to maintain the state of a tracked object. The state consists of information used for detection to track assignment, track termination, and display.

The structure contains the following fields:

- `id` : the integer ID of the track
- `bbox` : the current bounding box of the object; used for display
- `kalmanFilter` : a Kalman filter object used for motion-based tracking
- `age` : the number of frames since the track was first detected
- `totalVisibleCount` the total number of frames in which the track was detected (visible)
- `consecutiveInvisibleCount` : the number of consecutive frames for which the track was not detected (invisible).

Initialize Tracks

Noisy detections tend to result in short-lived tracks. For this reason, the example only displays an object after it was tracked for some number of frames. This happens when `totalVisibleCount` exceeds a specified threshold.

When no detections are associated with a track for several consecutive frames, the example assumes that the object has left the field of view and deletes the track. This happens when `consecutiveInvisibleCount` exceeds a specified threshold. A track may also get deleted as noise if it was tracked for a short time, and marked invisible for most of the of the frames.

```
function tracks = initializeTracks()
    % create an empty array of tracks
    tracks = struct(...
        'id', {}, ...
        'bbox', {}, ...
        'kalmanFilter', {}, ...
        'age', {}, ...
        'totalVisibleCount', {}, ...
        'consecutiveInvisibleCount', {});
end
```

Read a Video Frame & Detect Objects

Read the next video frame from the video file.

```
function frame = readFrame()  
    frame = obj.reader.step();  
end
```

Detect Objects

The `detectObjects` function returns the centroids and the bounding boxes of the detected objects. It also returns the binary mask, which has the same size as the input frame. Pixels with a value of 1 correspond to the foreground, and pixels with a value of 0 correspond to the background.

The function performs motion segmentation using the foreground detector. It then performs morphological operations on the resulting binary mask to remove noisy pixels and to fill the holes in the remaining blobs.

Function Detect Objects

```
function [centroids, bboxes, mask] = detectObjects(frame)

    % detect foreground
    mask = obj.detector.step(frame);

    % apply morphological operations to remove noise and
    fill in holes
    mask = imopen(mask, strel('rectangle', [3,3]));
    mask = imclose(mask, strel('rectangle', [15, 15]));
    mask = imfill(mask, 'holes');

    % perform blob analysis to find connected components
    [~, centroids, bboxes] = obj.blobAnalyser.step(mask);
end
```

Function Predict New Locations of Existing Tracks

Use the Kalman filter to predict the centroid of each track in the current frame, and update its bounding box accordingly.

```
function predictNewLocationsOfTracks()  
    for i = 1:length(tracks)  
        bbox = tracks(i).bbox;  
  
        % predict the current location of the track  
        predictedCentroid =  
predict(tracks(i).kalmanFilter);  
  
        % shift the bounding box so that its center is at  
        % the predicted location  
        predictedCentroid = int32(predictedCentroid) -  
bbox(3:4) / 2;  
        tracks(i).bbox = [predictedCentroid, bbox(3:4)];  
    end  
end
```

Assign Detections to Tracks

- Assigning object detections in the current frame to existing tracks is done by minimizing cost. The cost is defined as the negative log-likelihood of a detection corresponding to a track.
- The algorithm involves two steps:
 - Step 1: Compute the cost of assigning every detection to each track using the `distance` method of the `vision.KalmanFilter` System object. The cost takes into account the Euclidean distance between the predicted centroid of the track and the centroid of the detection. It also includes the confidence of the prediction, which is maintained by the Kalman filter. The results are stored in an $M \times N$ matrix, where M is the number of tracks, and N is the number of detections.
 - Step 2: Solve the assignment problem represented by the cost matrix using the `assignDetectionsToTracks` function. The function takes the cost matrix and the cost of not assigning any detections to a track.

Function Assign Detections to Tracks

- The value for the cost of not assigning a detection to a track depends on the range of values returned by the `distance` method of the `vision.KalmanFilter`. This value must be tuned experimentally. Setting it too low increases the likelihood of creating a new track, and may result in track fragmentation. Setting it too high may result in a single track corresponding to a series of separate moving objects.
- The `assignDetectionsToTracks` function uses the Munkres' version of the Hungarian algorithm to compute an assignment which minimizes the total cost. It returns an $M \times 2$ matrix containing the corresponding indices of assigned tracks and detections in its two columns. It also returns the indices of tracks and detections that remained unassigned.

Function Assign Detections to Tracks (2)

```
function [assignments, unassignedTracks,  
unassignedDetections] = ...  
    detectionToTrackAssignment()  
    nTracks = length(tracks);  
    nDetections = size(centroids, 1);  
  
% compute the cost of assigning each detection to each track  
    cost = zeros(nTracks, nDetections);  
    for i = 1:nTracks  
        cost(i, :) = distance(tracks(i).kalmanFilter,  
centroids);  
    end  
    % solve the assignment problem  
    costOfNonAssignment = 20;  
    [assignments, unassignedTracks,  
unassignedDetections] = ...  
        assignDetectionsToTracks(cost,  
costOfNonAssignment);  
end
```

Function Update Assigned Tracks

The `updateAssignedTracks` function updates each assigned track with the corresponding detection. It calls the correct method of `vision.KalmanFilter` to correct the location estimate. Next, it stores the new bounding box, and increases the age of the track and the total visible count by 1. Finally, the function sets the invisible count to 0.

```
function updateAssignedTracks()  
    numAssignedTracks = size(assignments, 1);  
    for i = 1:numAssignedTracks  
        trackIdx = assignments(i, 1);  
        detectionIdx = assignments(i, 2);  
        centroid = centroids(detectionIdx, :);  
        bbox = bboxes(detectionIdx, :);  
  
        % correct the estimate of the object's location  
        % using the new detection  
        correct(tracks(trackIdx).kalmanFilter, centroid);  
  
        % replace predicted bounding box with detected  
        % bounding box  
        tracks(trackIdx).bbox = bbox;
```


Function Update Assigned Tracks(2)

```
% update track's age
    tracks(trackIdx).age = tracks(trackIdx).age + 1;

    % update visibility
    tracks(trackIdx).totalVisibleCount = ...
        tracks(trackIdx).totalVisibleCount + 1;
    tracks(trackIdx).consecutiveInvisibleCount = 0;
end
end
```

Function Update Unassigned Tracks

Mark each unassigned track as invisible, and increase its age by 1.

```
function updateUnassignedTracks()  
    for i = 1:length(unassignedTracks)  
        ind = unassignedTracks(i);  
        tracks(ind).age = tracks(ind).age + 1;  
        tracks(ind).consecutiveInvisibleCount = ...  
            tracks(ind).consecutiveInvisibleCount + 1;  
    end  
end
```

Function Delete Lost Tracks

The deleteLostTracks function deletes tracks that have been invisible for too many consecutive frames. It also deletes recently created tracks that have been invisible for too many frames overall.

```
function deleteLostTracks()  
    if isempty(tracks)  
        return;  
    end  
    invisibleForTooLong = 10;  
    ageThreshold = 8;  
    % compute the fraction of the track's age for which it was  
    % visible  
    ages = [tracks(:).age];  
    totalVisibleCounts = [tracks(:).totalVisibleCount];  
    visibility = totalVisibleCounts ./ ages;  
    % find the indices of 'lost' tracks  
    lostInds = (ages < ageThreshold & visibility < 0.6) | ...  
    [tracks(:).consecutiveInvisibleCount] >= invisibleForTooLong;  
    % delete lost tracks  
    tracks = tracks(~lostInds);  
end
```

Function Create New Tracks

Create new tracks from unassigned detections. Assume that any unassigned detection is a start of a new track. In practice, you can use other cues to eliminate noisy detections, such as size, location, or appearance.

```
function createNewTracks()  
    centroids = centroids(unassignedDetections, :);  
    bboxes = bboxes(unassignedDetections, :);  
  
    for i = 1:size(centroids, 1)  
  
        centroid = centroids(i,:);  
        bbox = bboxes(i, :);  
  
        % create a Kalman filter object  
        kalmanFilter = configureKalmanFilter('ConstantVelocity', ...  
            centroid, [200, 50], [100, 25], 100);
```

Function Create New Tracks(2)

```
% create a new track
newTrack = struct(...
    'id', nextId, ...
    'bbox', bbox, ...
    'kalmanFilter', kalmanFilter, ...
    'age', 1, ...
    'totalVisibleCount', 1, ...
    'consecutiveInvisibleCount', 0);

% add it to the array of tracks
tracks(end + 1) = newTrack;

% increment the next id
nextId = nextId + 1;
end
end
```

Function Display Tracking Results

The `displayTrackingResults` function draws a bounding box and label ID for each track on the video frame and the foreground mask. It then displays the frame and the mask in their respective video players.

```
function displayTrackingResults()  
    % convert the frame and the mask to uint8 RGB  
    frame = im2uint8(frame);  
    mask = uint8(repmat(mask, [1, 1, 3])) .* 255;  
  
    minVisibleCount = 8;  
    if ~isempty(tracks)  
  
        % noisy detections tend to result in short-lived tracks  
        % only display tracks that have been visible for more than  
        % a minimum number of frames.  
        reliableTrackInds = ...  
            [tracks(:).totalVisibleCount] > minVisibleCount;  
        reliableTracks = tracks(reliableTrackInds);
```

Function Display Tracking Results(2)

```
% display the objects. If an object has not been detected
% in this frame, display its predicted bounding box.
if ~isempty(reliableTracks)
    % get bounding boxes
    bboxes = cat(1, reliableTracks.bbox);

    % get ids
    ids = int32([reliableTracks(:).id]);

    % create labels for objects indicating the ones for
    % which we display the predicted rather than the actual
    % location
    labels = cellstr(int2str(ids'));
    predictedTrackInds = ...
        [reliableTracks(:).consecutiveInvisibleCount] > 0;
    isPredicted = cell(size(labels));
    isPredicted(predictedTrackInds) = {' predicted'};
    labels = strcat(labels, isPredicted);
```

Function Display Tracking Results(3)

```
% draw on the frame
frame = insertObjectAnnotation(frame, 'rectangle', ...
                               bboxes, labels);
    % draw on the mask
mask = insertObjectAnnotation(mask, 'rectangle', ...
                              bboxes, labels
    end
end

    % display the mask and the frame
obj.maskPlayer.step(mask);
obj.videoPlayer.step(frame);
end

% final end for the initial function multiObjectTracking
end
```


Conclusions

- This example created a motion-based system for detecting and tracking multiple moving objects. Try using a different video to see if you are able to detect and track objects. Try modifying the parameters for the detection, assignment, and deletion steps.
- The tracking in this example was solely based on motion with the assumption that all objects move in a straight line with constant speed. When the motion of an object significantly deviates from this model, the example may produce tracking errors. Notice the mistake in tracking the person labeled #12, when he is occluded by the tree.
- The likelihood of tracking errors can be reduced by using a more complex motion model, such as constant acceleration, or by using multiple Kalman filters for every object. Also, you can incorporate other cues for associating detections over time, such as size, shape, and color.