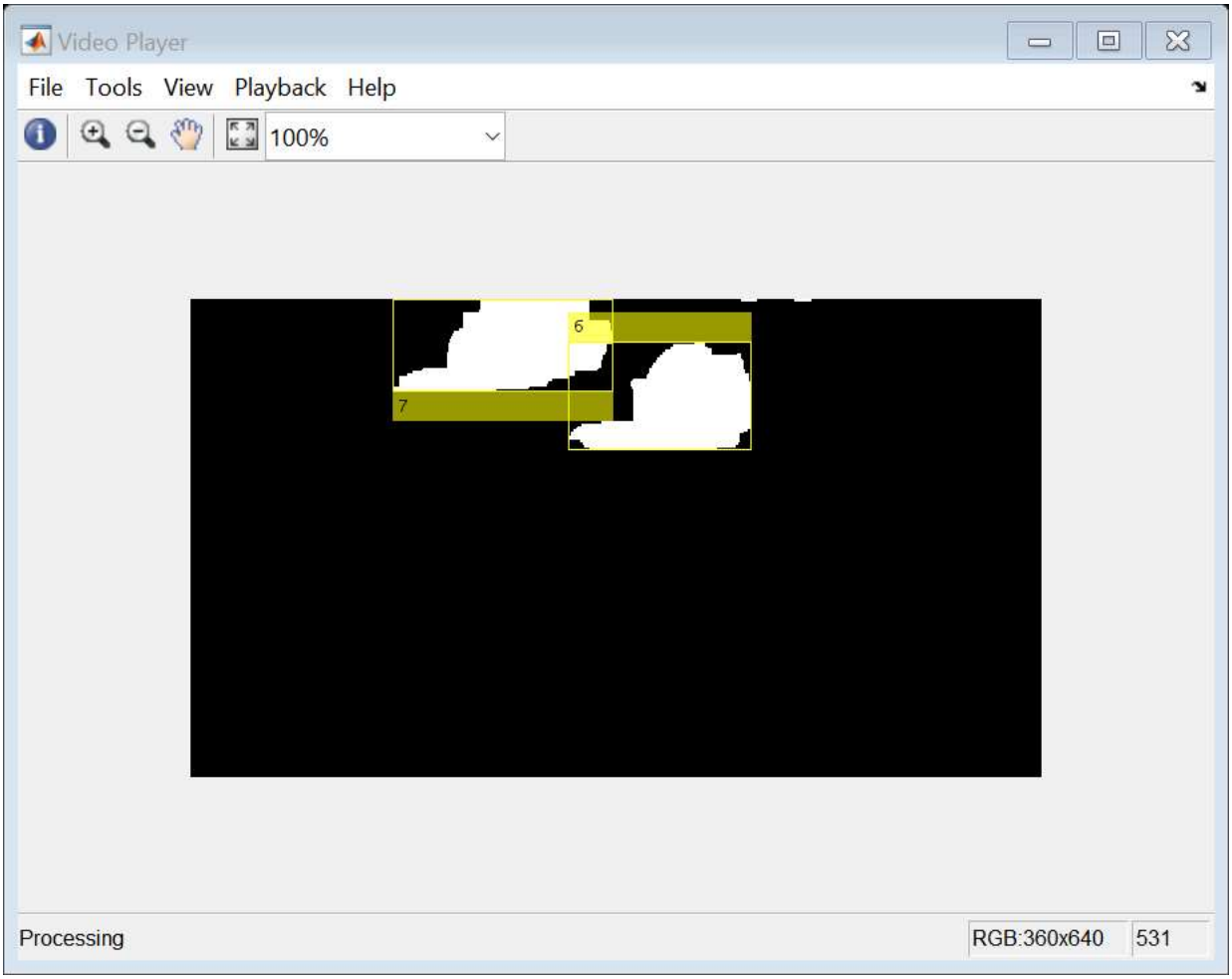
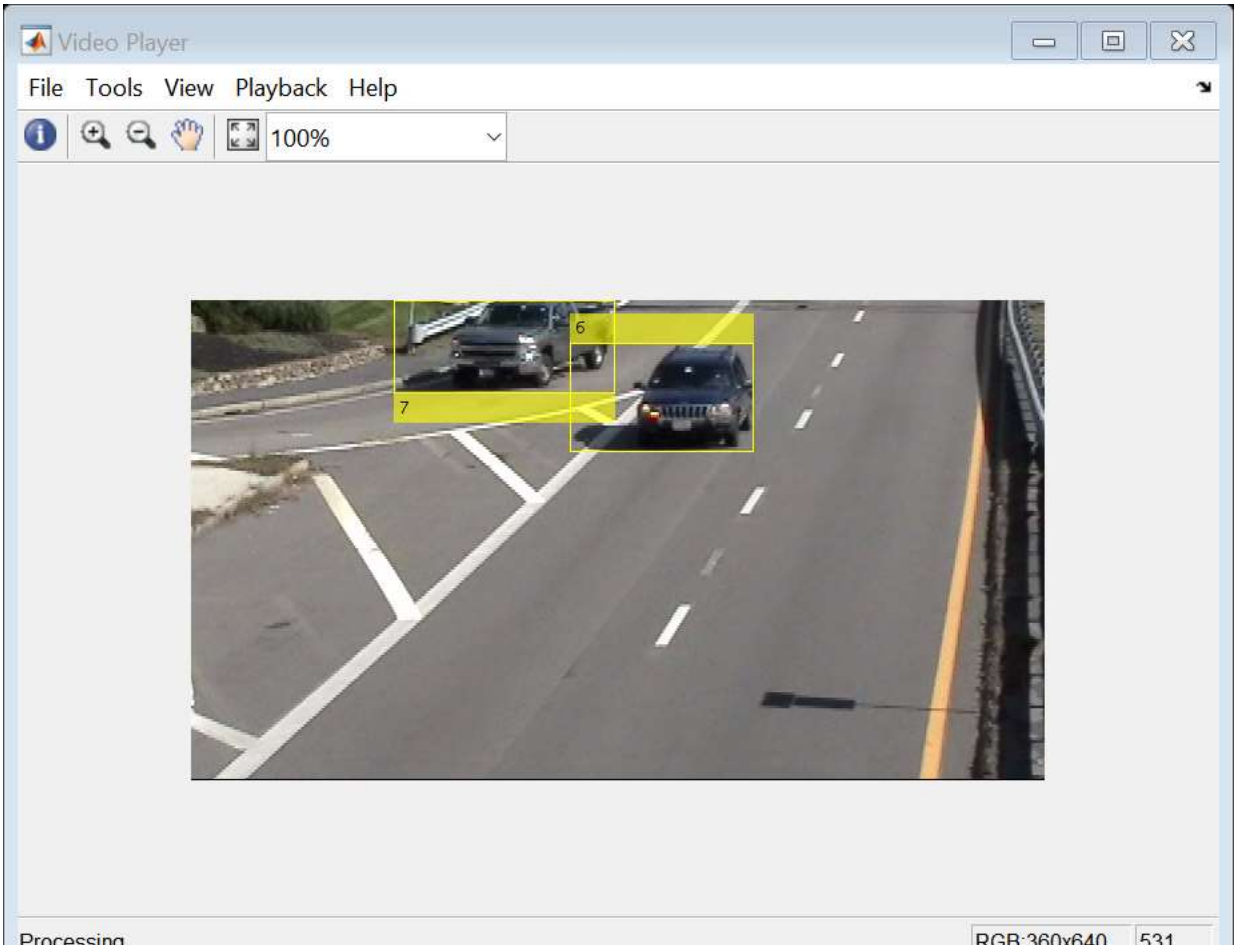
Contents

Motion-Based Multiple Object Tracking

```
function multiObjectTracking()
```

```
clc;  clear; close all;

% 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
```


Video Player
File Tools View Playback Help
100%
6
7
Processing
RGB:360x640
531


Video Player
File Tools View Playback Help
100%
6
7
Processing
RGB:360x640
531

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('singleball.mp4');
        obj.reader = vision.VideoFileReader('visiontraffic.avi');
        % Create two video players, one to display the video,
        % and one to display the foreground mask.
        obj.videoPlayer = vision.VideoPlayer('Position', [0, 200, 600, 400]);
        obj.maskPlayer = vision.VideoPlayer('Position', [600, 200, 600, 400]);

        % 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).

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

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 [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
```

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 MxN 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.

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 x 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 [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
```

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;

            % 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
```

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
```

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 = 20;
        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
```

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);

            % 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
```

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);

            % 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);

                % Draw the objects on the frame.
                frame = insertObjectAnnotation(frame, 'rectangle', ...
                    bboxes, labels);

                % Draw the objects on the mask.
                mask = insertObjectAnnotation(mask, 'rectangle', ...
                    bboxes, labels);

                % Draw the centroids
                for i = 1 : 1 : size(centroids,1)
                    mask = insertShape(mask, 'FilledCircle', [centroids(i,:) 5], ...
                        'LineWidth',5, 'Color','red');
                end

            end
        end

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

```
end
```

Published with MATLAB® R2016a