ASSIGNMENT - 2
CBIVR

PRIYANSHU SHARMA
15BCE1282

QUESTION - 1

Motion Based Multiple Object Tracking

Theory

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:

1. Detecting moving objects in each frame
2. Associating the defections 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.

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.

CODE

```
function MotionBasedMultiObjectTrackingExample()

obj = setupSystemObjects();

tracks = initializeTracks();
nextId = 1;
while ~isDone(obj.reader)
    frame = readFrame();
    [centroids, bboxes, mask] = detectObjects(frame);
    predictNewLocationsOfTracks();
    [assignments, unassignedTracks, unassignedDetections] = ...
        detectionToTrackAssignment();

    updateAssignedTracks();
    updateUnassignedTracks();
    deleteLostTracks();
    createNewTracks();

    displayTrackingResults();
end



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.mp4');

        % Create two video players, one to display the video,
        % and one to display the foreground mask.
        obj.maskPlayer = vision.VideoPlayer('Position', [740, 400, 700, 400]);
        obj.videoPlayer = vision.VideoPlayer('Position', [20, 400, 700, 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.
```

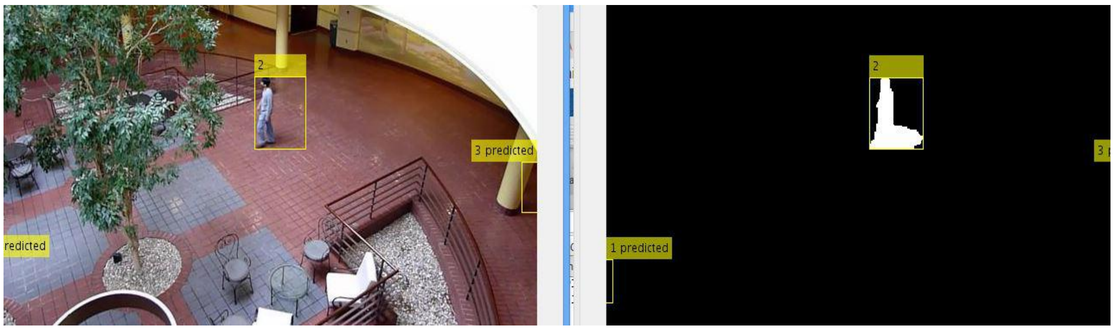
```
                obj.detector = vision.ForegroundDetector('NumGaussians', 3, ...
                        'NumTrainingFrames', 40, 'MinimumBackgroundRatio', 0.7);

                % 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

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

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 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)];
```

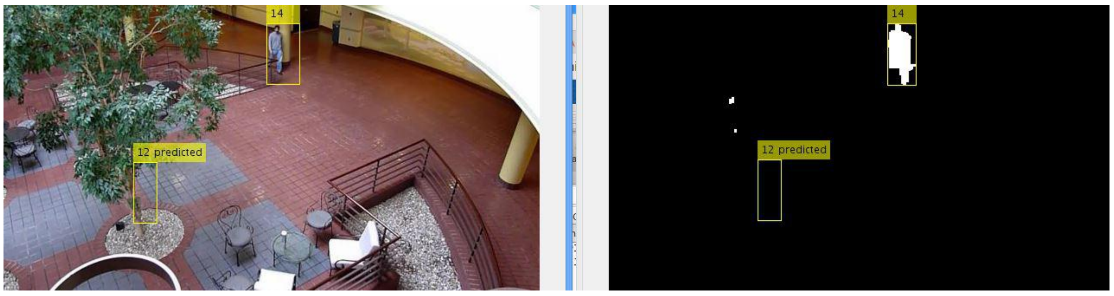
```
            end
    end
     function [assignments, unassignedTracks, unassignedDetections] = ...
                  detectionToTrackAssignment()


            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 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


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

```matlab
        end
end
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
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
```

```matlab
        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);
                end
            end

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

OUTPUT





QUESTION - 2

Object Tracking and Motion Estimation

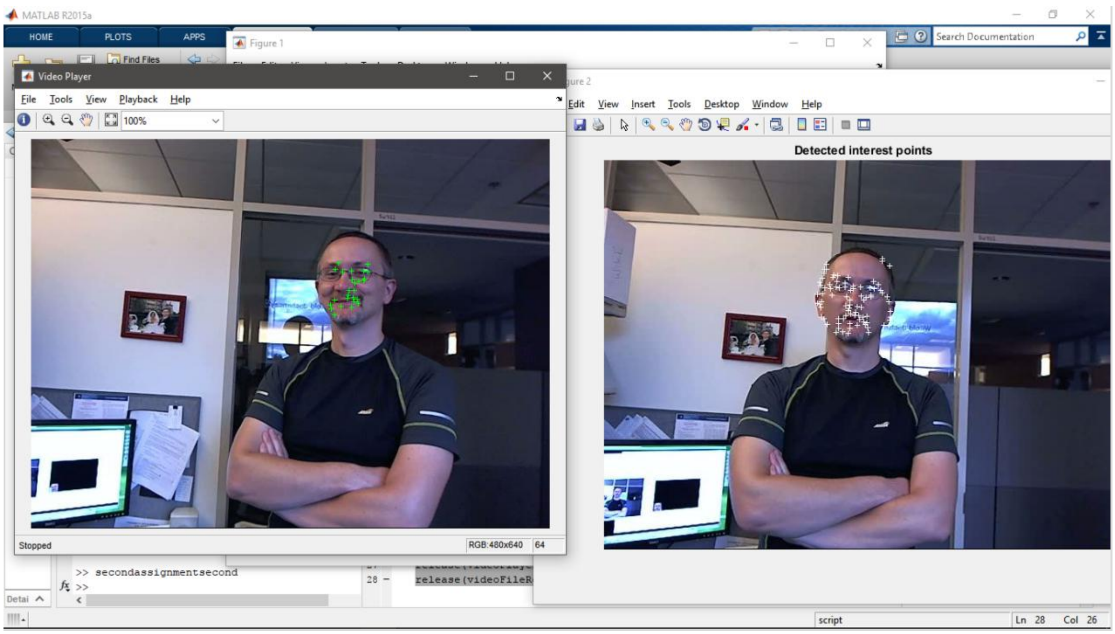
Theory

Motion estimation is the process of determining motion vectors that describe the transformation from one 2D image to another; usually from adjacent frames in a video sequence. It is an ill-posed problem as the motion is in three dimensions but the images are a projection of the 3D scene onto a 2D plane. The motion vectors may relate to the whole image (global motion estimation) or specific parts, such as rectangular blocks, arbitrary shaped patches or even per pixel. The motion vectors may be represented by a translational model or many other models that can approximate the motion of a real video camera, such as rotation and translation in all three dimensions and zoom.

CODE

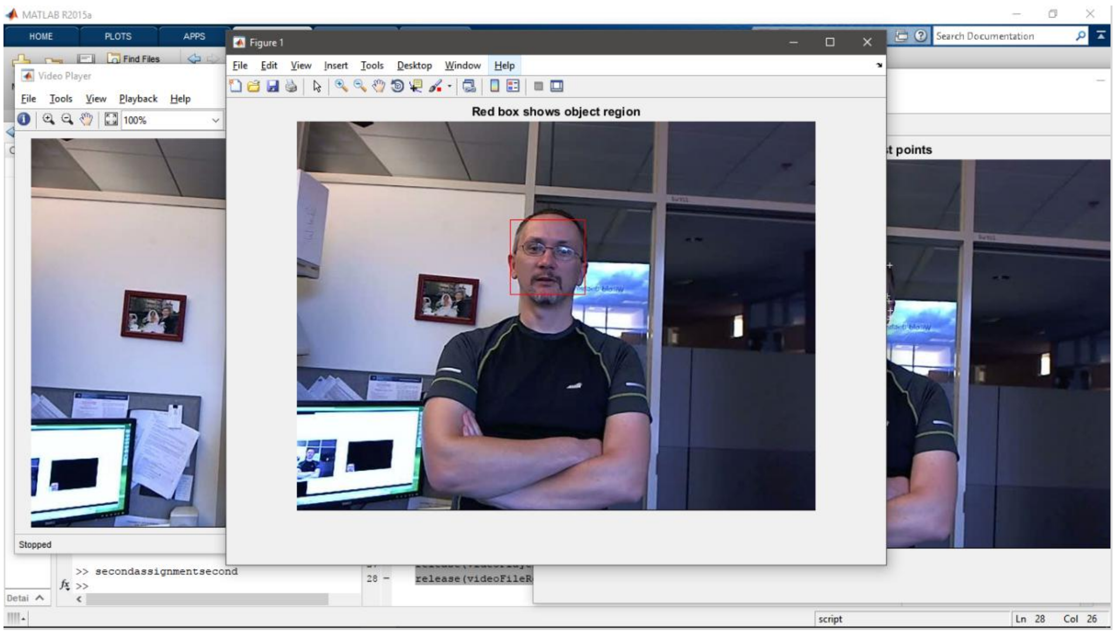
```
videoFileReader = vision.VideoFileReader('visionface.avi');
videoPlayer = vision.VideoPlayer('Position',[100,100,680,520]);

objectFrame = step(videoFileReader);
objectRegion = [264,122,93,93];

objectImage = insertShape(objectFrame,'Rectangle',objectRegion,'Color','red');
figure;
imshow(objectImage);
title('Red box shows object region');
points = detectMinEigenFeatures(rgb2gray(objectFrame),'ROI',objectRegion);

pointImage = insertMarker(objectFrame,points.Location,'+','Color','white');
figure;
imshow(pointImage);
title('Detected interest points');
tracker = vision.PointTracker('MaxBidirectionalError',1);

initialize(tracker,points.Location,objectFrame);
while ~isDone(videoFileReader)
        frame = step(videoFileReader);
        [points, validity] = step(tracker,frame);
        out = insertMarker(frame,points(validity, :),'+');
        step(videoPlayer,out);
end

release(videoPlayer);
release(videoFileReader);
```

OUTPUT



```
>> secondassignmentsecond
>>
```

```
28 -    release(videoFileR
```


MATLAB R2015a
HOME
PLOTS
APPS
Figure 1
File Edit View Insert Tools Desktop Window Help
Red box shows object region
Video Player
File Tools View Playback Help
100%
Stopped
Search Documentation
script
Ln 28 Col 26

```
>> secondassignmentsecond
>>
```

```
28 -    release(videoFileR
```

QUESTION - 3

Retrieval Image command

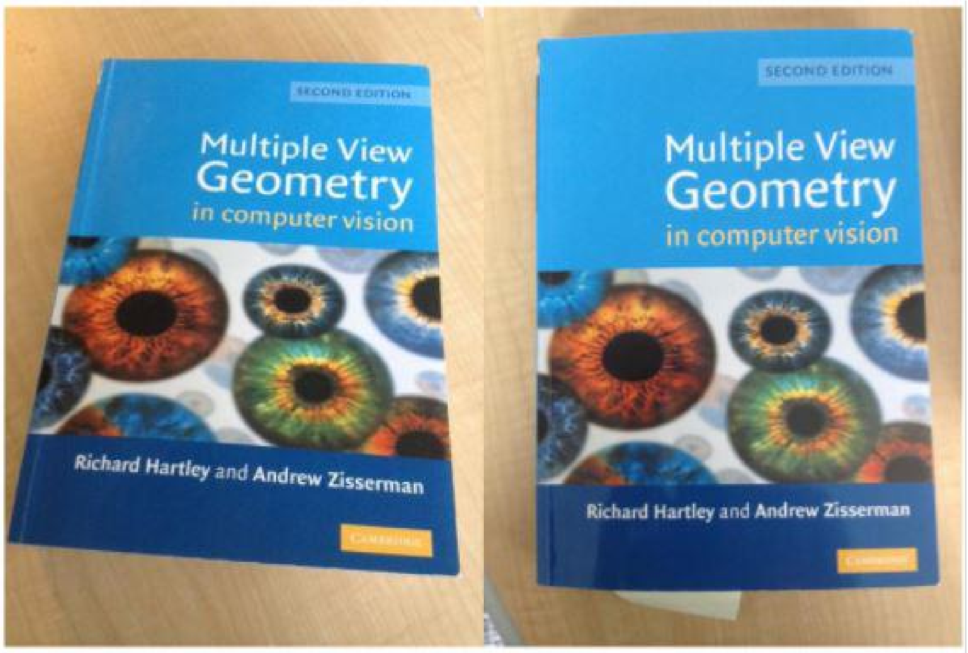
Theory

It will search set for similar images

CODE

```
dataDir = fullfile(toolboxdir('vision'),'visiondata','bookCovers');
bookCovers = imageDatastore(dataDir);
thumbnailGallery = [];
for i = 1:length(bookCovers.Files)
        I = readimage(bookCovers,i);
        thumbnail = imresize(I,[300 300]);
        thumbnailGallery = cat(4,thumbnailGallery,thumbnail);
end

figure
montage(thumbnailGallery);
imageIndex = indexImages(bookCovers);
queryDir = fullfile(dataDir,'queries',filesep);
queryImage = imread([queryDir 'query3.jpg']);

imageIDs = retrieveImages(queryImage,imageIndex);
bestMatch = imageIDs(1);
bestImage = imread(imageIndex.ImageLocation{bestMatch});

figure
imshowpair(queryImage,bestImage,'montage')
```

OUTPUT


SECOND EDITION
Multiple View Geometry in computer vision
Richard Hartley and Andrew Zisserman
CAMBRIDGE