ASSIGNMENT - 5

Experiments performed:

1) Finding suitable feature points: Not all the pixels were taken to find the optical flow, because it would become computationally expensive process. So best features were selected using `detectKAZEFeatures`.

2) Finding suitable window sizes: Different sizes of windows were used to generate outputs and the optimal size is 15.

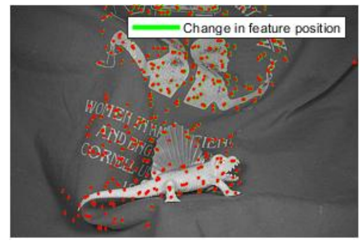
3) Segmentation using Optical flow: To obtain the segmentation of moving objects in a video, features that show some flow are considered. Next, closing operation is performed using a square structural element.

Results:

TRACKING OF MOVING OBJECTS:

First Image

Second Image



Optical Flow

First Image

Second Image

Optical Flow

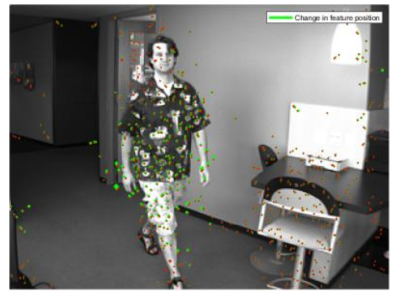
First Image

Second Image

Optical Flow

First Image

Second Image



Optical Flow

First Image

Second Image

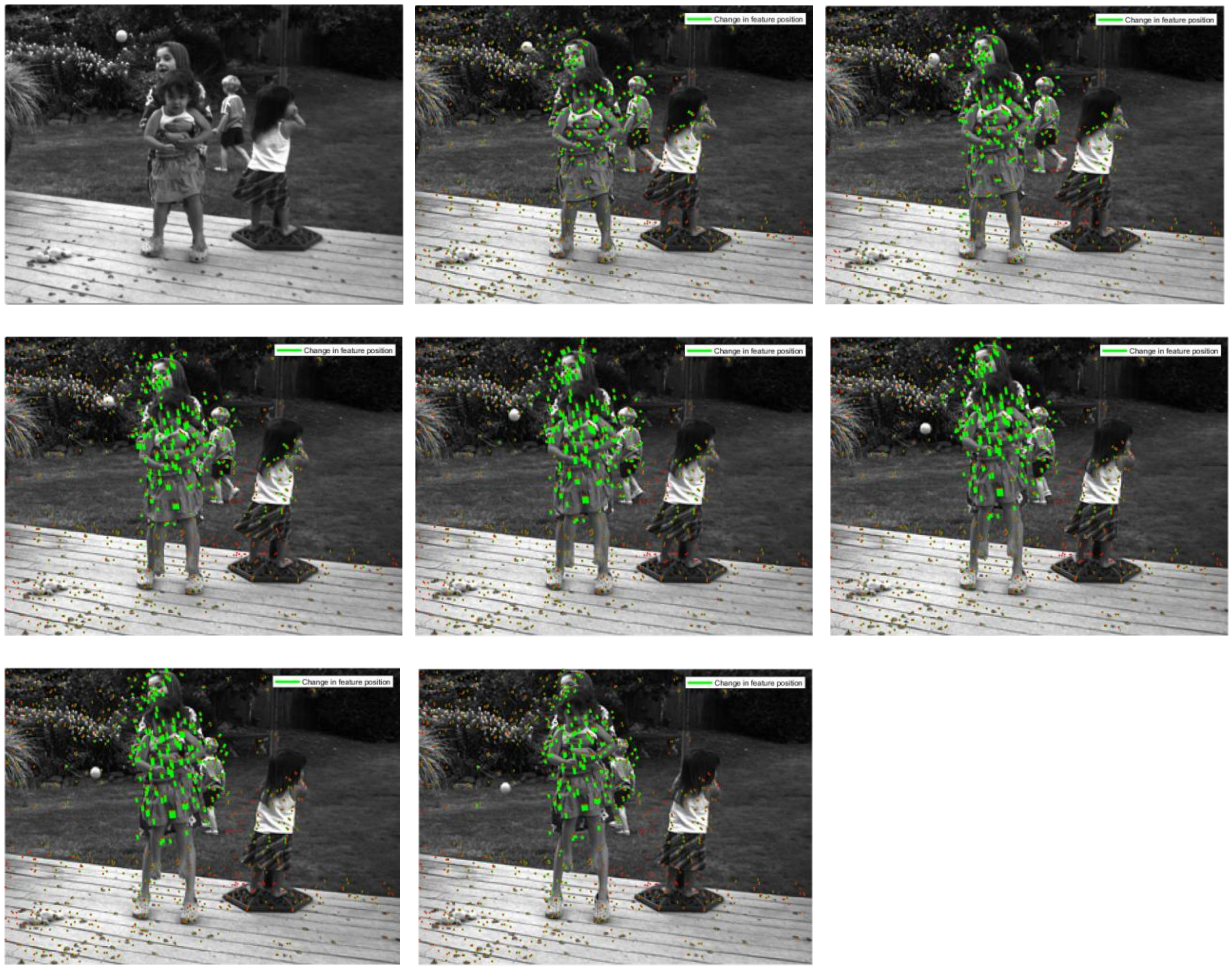
Optical Flow

First Image

Second Image

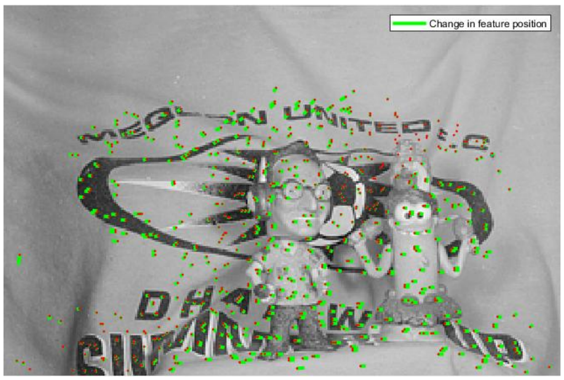
Optical Flow

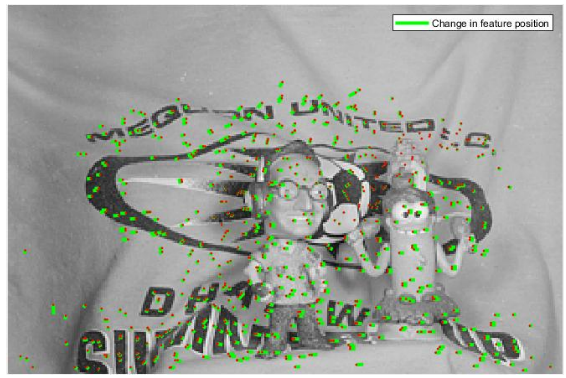
TRACKING OF MOVING OBJECTS IN VIDEO:

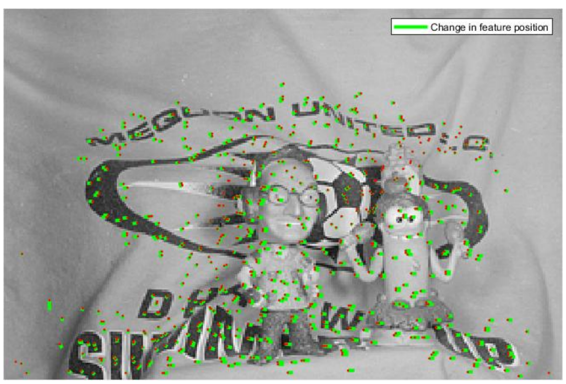


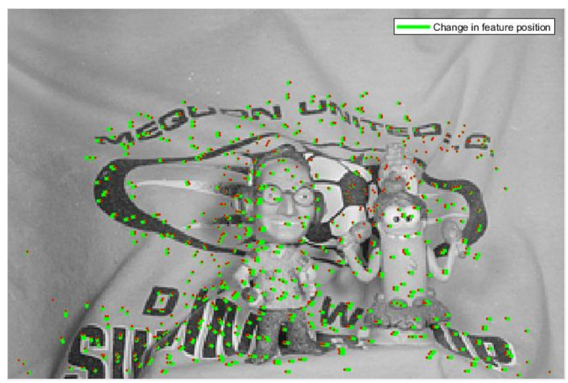

Change in feature position
Change in feature position
Change in feature position
Change in feature position
Change in feature position
Change in feature position
Change in feature position

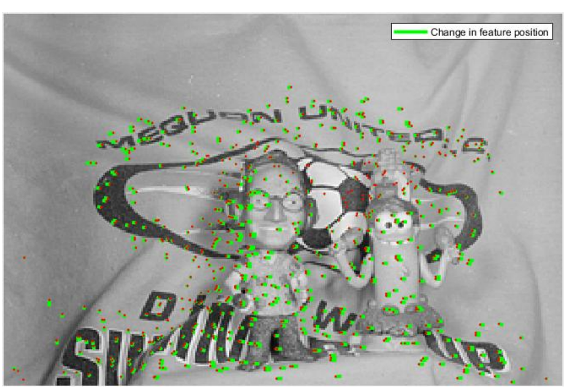

Change in feature position

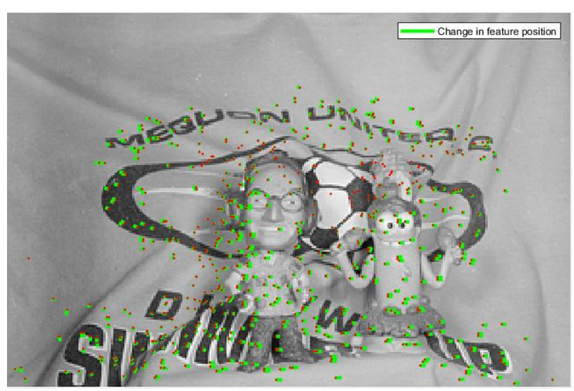

Change in feature position

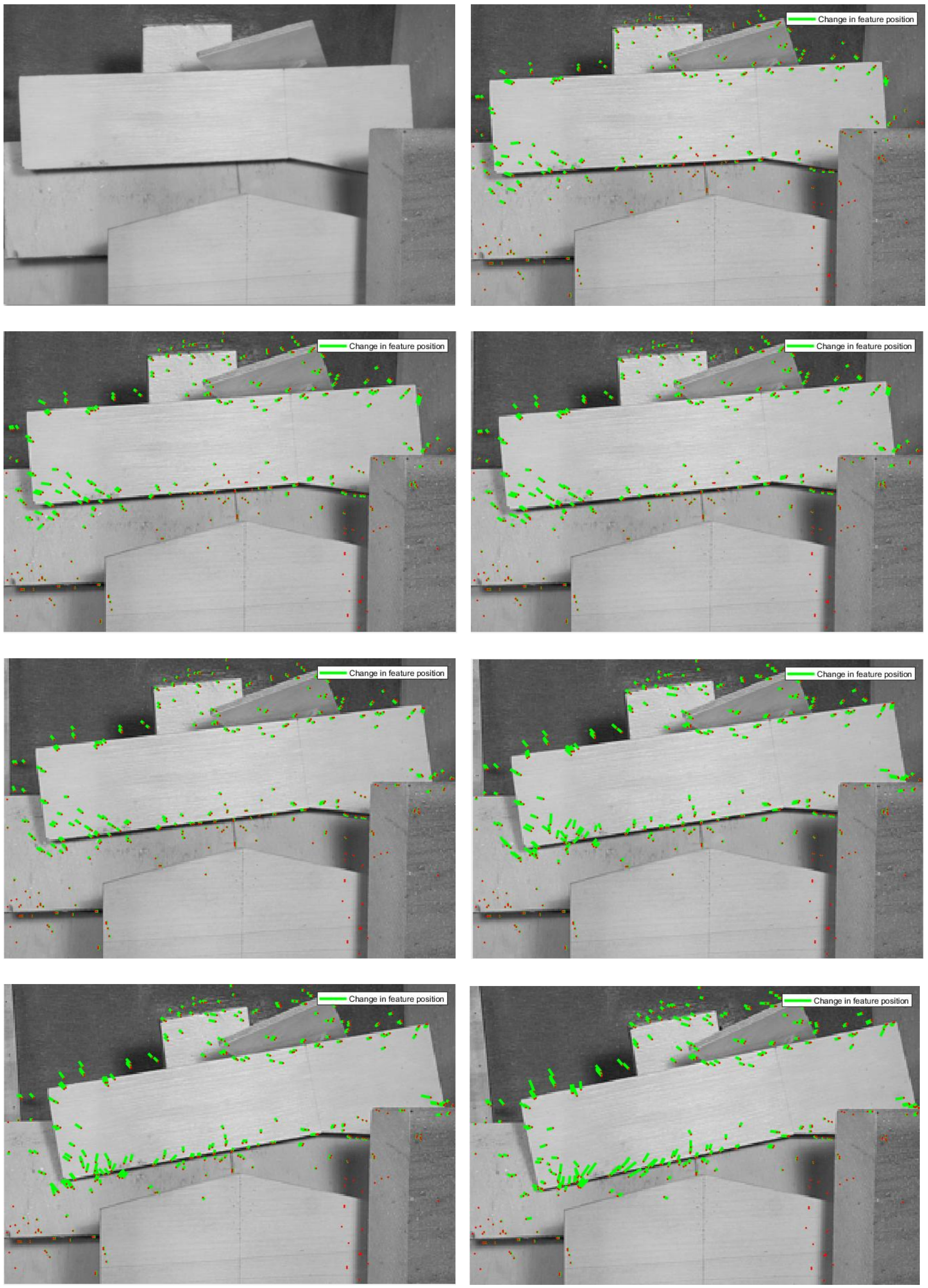

Change in feature position


Change in feature position

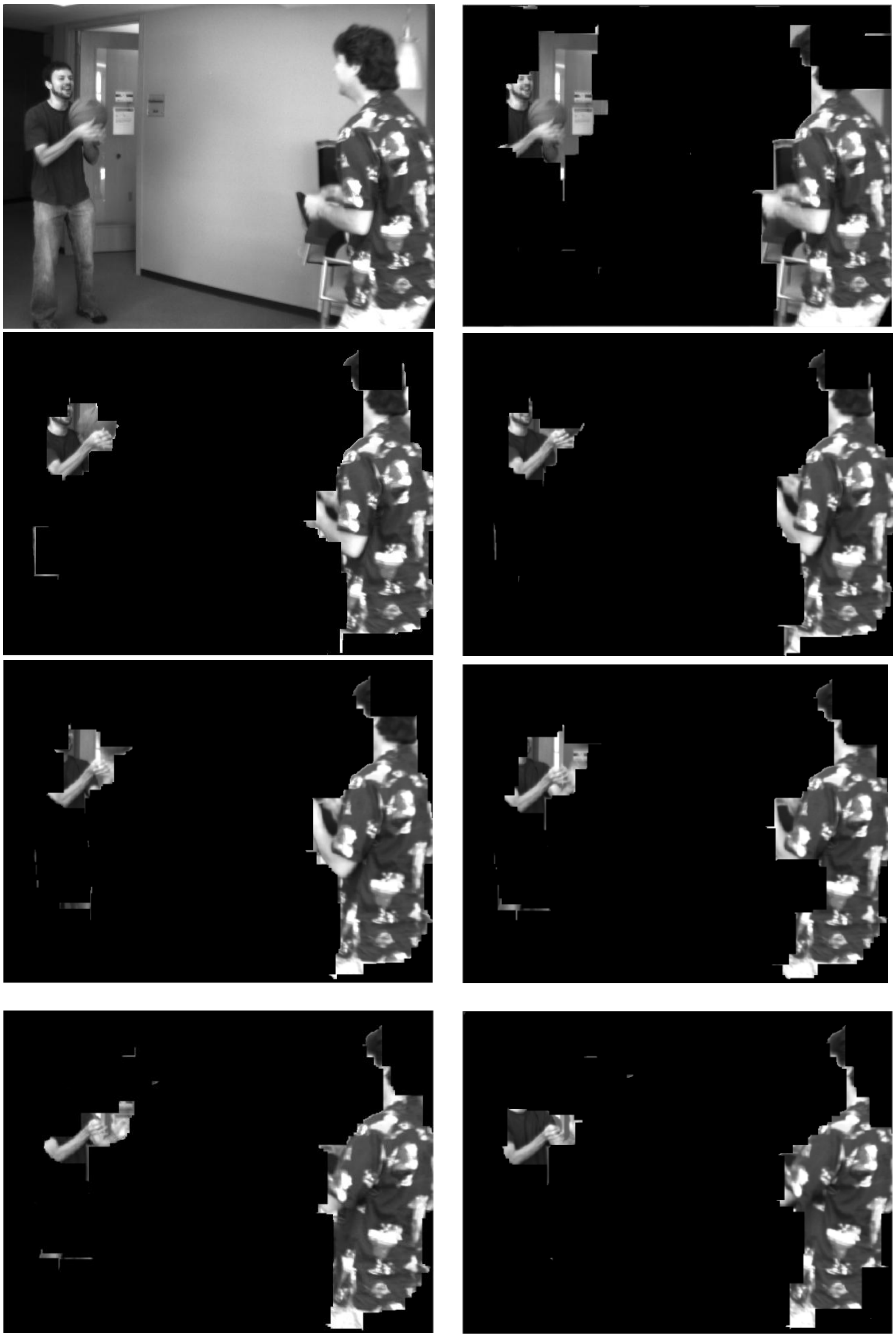

Change in feature position

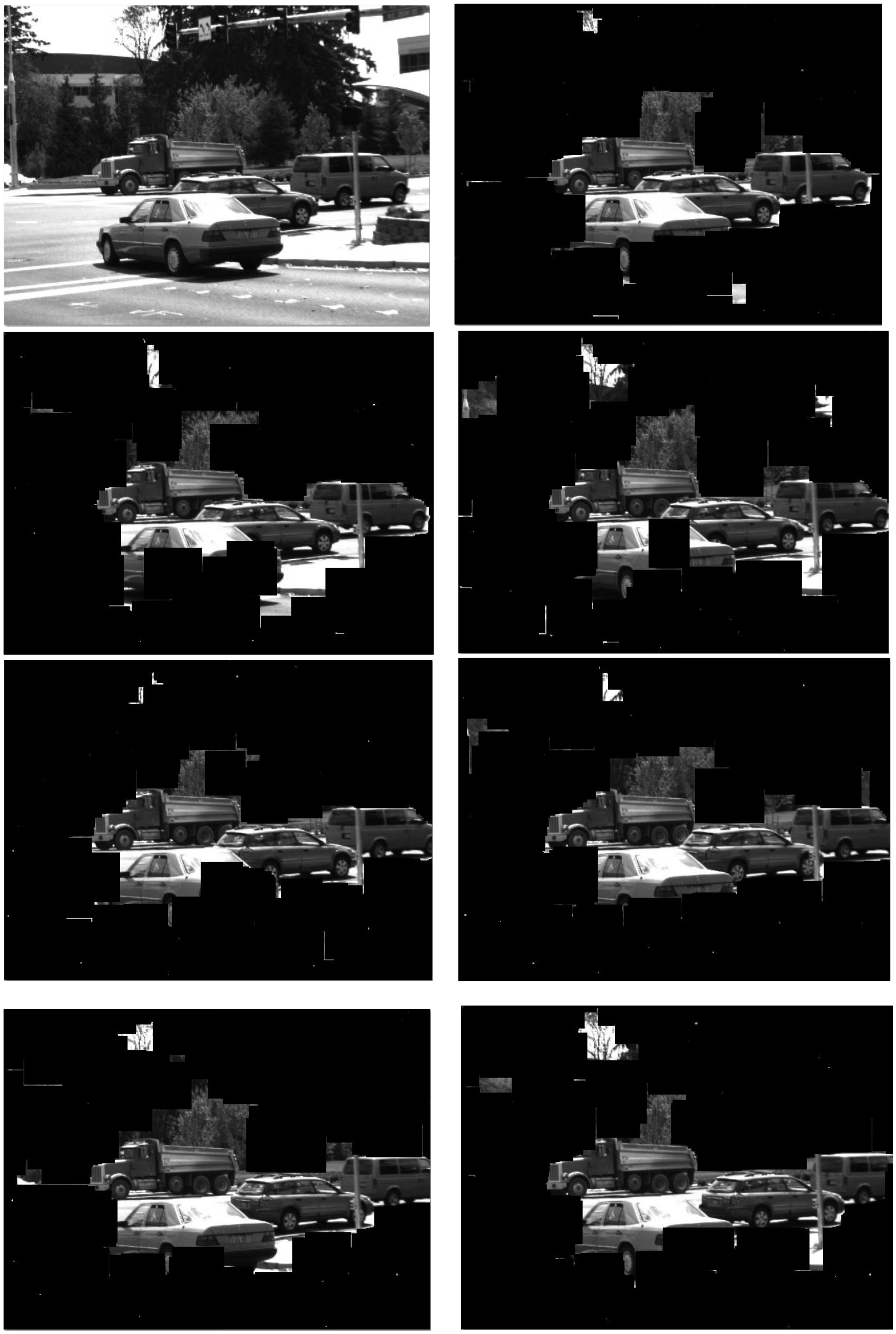

Change in feature position

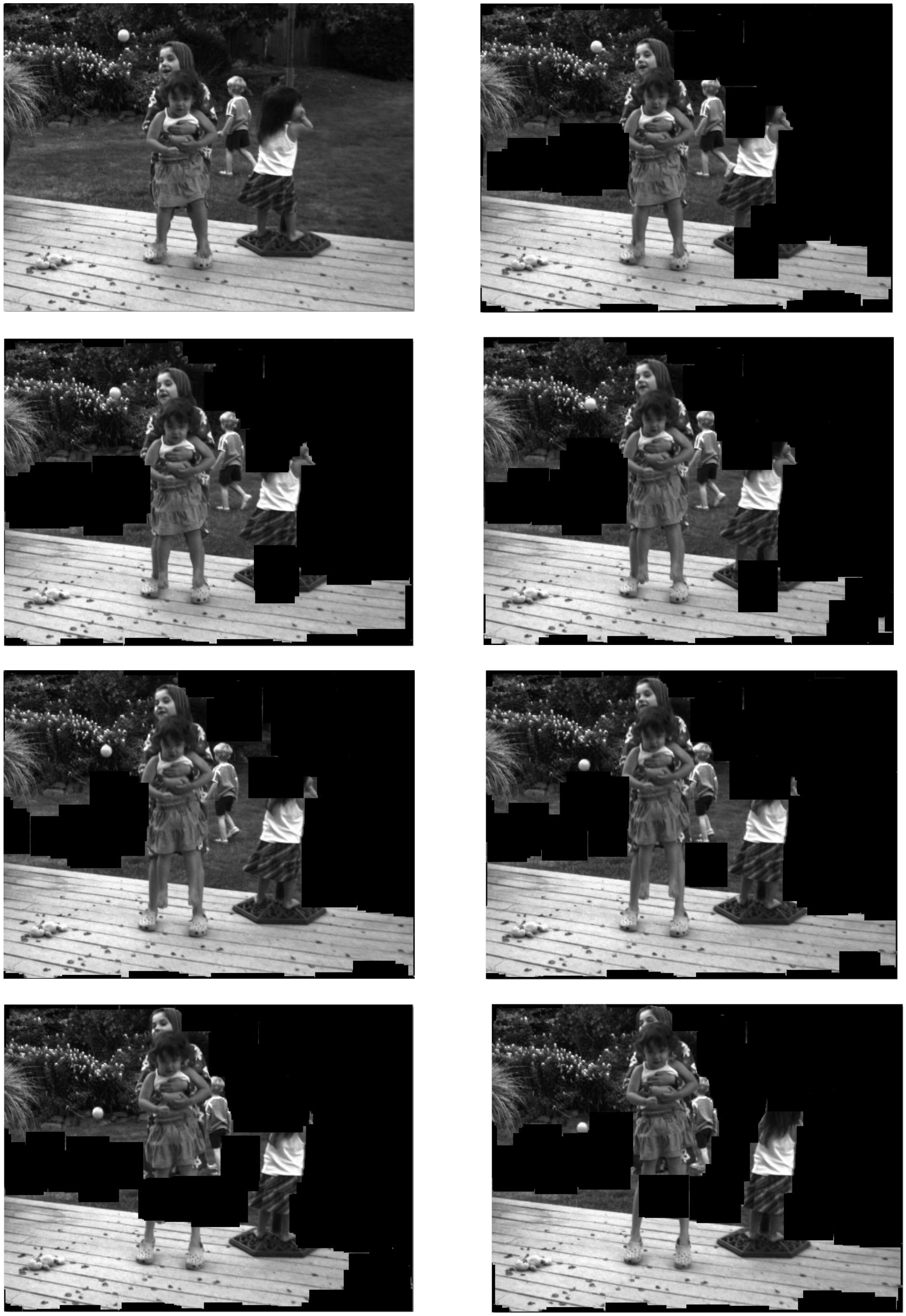

Change in feature position


Change in feature position
Change in feature position
Change in feature position
Change in feature position
Change in feature position
Change in feature position
Change in feature position

SEGMENTATION AND DETECTION OF MOVING OBJECTS:







CODE FOR OPTICAL FLOW:

```
close all;
clear all;
clc;
%% READ IMAGE
% directory = 'dataset/other-gray-twoframes/MiniCooper/';

directory = 'dataset/eval-data-gray/Basketball/';
r_T = 25;
num_iters = 50;

imgs = dir(fullfile(directory, '*.png'));
imgs_names = {imgs.name};

I_R = imresize(imsharpen(imsharpen(imread(string([directory,
imgs_names{1}])))),0.5);

%% DETECT FEATURE POINTS
pts = detectKAZEFeatures(I_R);
pts = pts.selectStrongest(800);
keypoints = floor(pts.Location()'); % (x,y)

figure(1);
imshow(I_R);
hold on;
plot(keypoints(1, :), keypoints(2, :), 'r.');
hold off;
title('Features on the reference image');

% OPTICAL FLOW REPRESENTATION
I_prev = I_R;
pause(0.05);

for img_idx = 1:size(imgs_names,2)
    I_curr = imresize(imsharpen(imsharpen(imread(string([directory,
imgs_names{img_idx}])))),0.5);
    dkp = zeros(size(keypoints));
    parfor j = 1:size(keypoints, 2)
        W = trackKLT(I_prev, I_curr, keypoints(:,j)', r_T, num_iters);
        dkp(:, j) = W(:, end);
    end
    kpold = keypoints;
    keypoints = keypoints + dkp;

    imshow(I_curr);
    hold on;
    plotMatches(1:size(keypoints, 2), flipud(keypoints), flipud(kpold));
    hold off;
    title('Optical Flow');
    legend('Change in feature position');
    I_prev = I_curr;
end

function [W, p_hist] = trackKLT(I_R, I, x_T, r_T, num_iters)
```

```matlab
% TRACKKLT Obtains best warping matrix.
% Inputs:
%   I_R         Reference image.
%   I           Image to track point in.
%   x_T         Points to track expressed as [x y]=[col row].
%   r_T         Radius of patch to track.
%   num_iters   Amount of iterations to run.
% Outputs:
%   W       Final W estimate. Size(2x3)
%   p_hist  History of p estimates, including the initial estimate.
%           Size(6x(num_iters+1)).

    % Initial estimate of warp.
    W = getSimWarp(0, 0, 0, 1);
    thrs = 0.01;

    p_hist = zeros(6, num_iters+1);
    p_hist(:, 1) = W(:);

    % Reference template
    I_RP = getWarpedPatch(I_R, W, x_T, r_T);
    i_r = I_RP(:);

    % Calculate dw_dx
    xs = -r_T:r_T;
    ys = -r_T:r_T;
    n = numel(xs);
    xy1 = [kron(xs, ones([1 n]))' kron(ones([1 n]), ys)' ones([n*n 1])];
    dw_dx = kron(xy1, eye(2));


    for iter = 1:num_iters
        % Getting more, for a valid convolution.
        IP = getWarpedPatch(I, W, x_T, r_T + 1);
        IWT = IP(2:end-1, 2:end-1);
        i = IWT(:);

        % getting di/dp
        IWTx = conv2(1, [1 0 -1], IP(2:end-1, :), 'valid');
        IWTy = conv2([1 0 -1], 1, IP(:, 2:end-1), 'valid');
        di_dw = [IWTx(:) IWTy(:)]; % as written in the statement
        di_dp = zeros(n*n, 6);
        for pixel_i = 1:n*n
            di_dp(pixel_i, :) = di_dw(pixel_i, :) * ...
                dw_dx(pixel_i*2-1:pixel_i*2, :);
        end

        % Hessian
        H = di_dp' * di_dp;

        if min(eig(H)) > thrs
            % Putting it together and incrementing
            delta_p = H^-1 * di_dp' * (i_r - i);
            W = W + reshape(delta_p, [2 3]);
        else
```

```
            delta_p = zeros(2,3);
        end

        p_hist(:, iter + 1) = W(:);

        if norm(delta_p) < 1e-3
           p_hist = p_hist(:, 1:iter+1);
           return
        end
    end

end


function W = getSimWarp(delta_x, delta_y, theta_d, lambda)
% GETSIMWARP Generates warp matrix.
% Inputs:
%   delta_x     Translation along x direction.
%   delta_y     Translation along y direction.
%   theta_d     Angle in degrees.
%   lambda      Scalar multiplier.
% Output:
%   W   Warp matrix. Size(2x3).
    R = [cosd(theta_d), -sind(theta_d);...
        sind(theta_d), cosd(theta_d)];

    t = [delta_x; delta_y];

    W = lambda*([R t]);
end


function patch = getWarpedPatch(I, W, x_T, r_T)
% GETWARPEDPATCH Gives warped image patch using W for points x_T within
% radius r_T.
% Inputs:
%   I       Image
%   W       Warping matrix. Size(2x3).
%   x_T     Contains [x_T y_T].Size(1x2)
%   r_T     Radius of patch.
% Output:
%   patch   Patch.Size((2*r_T+1)x(2*r_T+1))

    patch = zeros(2*r_T+1);

    max_coords = fliplr(size(I));
    WT = W';
    I = double(I);

    for x = -r_T:r_T
        for y = -r_T:r_T
            warped = x_T + [x y 1] * WT;

            if all(warped < max_coords & warped > [1 1])
```

```
                floors = floor(warped);
                weights = warped - floors;
                a = weights(1); b = weights(2);

                % Bilinear combination
                patch(y + r_T + 1, x + r_T + 1) = (1-b) * (...
                (1-a) * I(floors(2), floors(1)) +...
                a * I(floors(2), floors(1)+1))...
                + b * (...
                (1-a) * I(floors(2)+1, floors(1)) +...
                a * I(floors(2)+1, floors(1)+1));
            end

        end
    end

end
```

CODE FOR SEGMENTATION:

```
directory = 'dataset/eval-data-gray/Dumptruck/';
num_iters = 25;
r_T = 15;

imgs = dir(fullfile(directory, '*.png'));
imgs_names = {imgs.name};

opticFlow = opticalFlowLK('NoiseThreshold',0.009);

for img_idx = 1:size(imgs_names,2)
    I = imread(string([directory, imgs_names{img_idx}]));
    frameGray = imsharpen(imsharpen(I));
    flow = estimateFlow(opticFlow,frameGray);

    [r, c] = size(frameGray);
    mag = flow.Magnitude;
    mag(mag > 0.2) = 1;
    mag(mag < 0.2) = 0;
    vx = flow.Vx;
    vy = flow.Vy;
    mag(vx < 0.01) = 0;
    mag(vy < 0.01) = 0;
    coorx = (kron(ones(r, 1), 1:c) + vx) .* mag;
    coory = (kron(ones(c, 1), 1:r)' + vy) .* mag;
    coorx(coorx > c) = c; coorx(coorx < 1) = 1;
    coory(coory > r) = r; coory(coory < 1) = 1;

    figure(1);
    subplot(1,3,1);
    imshow(I);
    hold on
    plot(floor(coorx), floor(coory),'r.');
    hold off
```

```
    mask = zeros(r,c) + mag;
    mask2 = imclose(mask, strel('square',60));
    subplot(1,3,2); imshow(mask);
    seg = uint8(double(mask2) .* double(I));
    subplot(1,3,3); imshow(seg);

    quiver(vx.*mag, vy.*mag);



    pause(5);
end

function quiver_uv(u,v)
    scalefactor = 50/size(u,2);
    u_ = scalefactor * imresize(u, scalefactor, 'bilinear');
    v_ = scalefactor * imresize(v, scalefactor, 'bilinear');

    quiver(u_(end:-1:1,:), -v_(end:-1:1,:), 2);
    axis('tight');
end
```