

```
%===== flowme =====
%
% script flowme
%
% This is a script stub that will execute the optical flow solver to be
% written for this homework. The script code will not work as is, and
% must be adjusted. The dynamic range of the images definitely needs to
% be adjusted, as may the smoothness of the image data.
%
%
% Variables in the script:
%   source      - which image sequence source to use.
%   di          - vector spacing for quiver.
%   scale       - what scale to use for plotting vector field.
%   alphasqGD   - the value of alpha squared for optical flow.
%   iterGD      - the number of iterations in the solver.
%   dt         - the gradient update step size.
%
%===== flowme =====

figure_count = 1;
%==(0) Setup variables and get images ready.
close all
load('optflowData.mat');
sources = {'box','bonanza','fish3','fish4'};
for i=1:length(sources)
    source = sources(i);
    source = source{1};
    switch source
        case 'box'
            I1 = imsmooth(box(:,:,1),0.5);
            I2 = imsmooth(box(:,:,3),0.5);
            scale = 2;
            alphasqGD = 0.017;
            iterGD = 6000;
            dt = 0.005;
            di = 1;
        case 'bonanza'
            I1 = imsmooth(bonanza(:,:,1),3);
            I2 = imsmooth(bonanza(:,:,4),3);
            scale = 10;
            alphasqGD = 0.02;
            iterGD = 4000;
            dt = 0.00002;
            di = 5;
        case 'fish3'
            I1 = imsmooth(fish03(:,:,1),3);
            I2 = imsmooth(fish03(:,:,20),3);
            scale = 2.0;
            alphasqGD = 0.007;
            iterGD = 1000;
            dt = 0.0002;
            di = 5;
        case 'fish4'
            I1 = imsmooth(fish04(:,:,1),3);
            I2 = imsmooth(fish04(:,:,12),3);
            scale = 2.0;
            alphasqGD = 1.2;
            iterGD = 11000;
```

GD Method

```
dt = 0.00002;
di = 5;
end

%==(1) Actually compute the optical flow.
[X.u, X.v] = optflowGD(I1, I2, alphasqGD, iterGD, dt);

lenX = abs(X.u + i*X.v);
disp(['The max length optical vector is: ' num2str(max(lenX(:)))]);

%==(2) Display output of the optical flow.
j = 8 * (i - 1);
figure(j + 1);
    imagesc(I1);
    colormap('gray');
    axis image;
    title(strcat(source, ' I1'));
figure(j + 2);
    imagesc(I2);
    colormap('gray');
    axis image;
    title(strcat(source, ' I2'));
figure(j + 3);
    imagesc(I2-I1);
    colormap('gray');
    colorbar;
    axis image;
    title(strcat(source, ' I2 - I1'));
figure(j + 4);
    imagesc(X.u);
    colorbar;
    axis image;
    title(strcat(source, ' u'));
figure(j + 5);
    imagesc(X.v);
    colorbar;
    axis image;
    title(strcat(source, ' v'));
figure(j + 6);
    imagesc(I1);
    colormap('gray');
    axis image;
    [M, N] = size(I1);
    subx = [1:di:N];
    suby = [1:di:M];
    [gridx, gridy] = meshgrid(subx, suby);
    hold on;
    quiver(gridx, gridy, X.u(suby,subx),X.v(suby, subx),scale,'Color','r');
    hold off;
    title(strcat(source, ' I1 + Vector Field'));

intI = optflow_check(I1, I2, X);

figure(j + 7);
    imagesc(intI);
    colormap('gray');
    axis image;
    title(strcat(source, ' Warped I1'));

figure(j + 8);
    imagesc(I2 - intI);
```

GD Method

```
colorbar;
title(strcat(source,' Error Between I1 + Velocity and I2'));

end

%===== optflowGD =====
%
% [Xu, Xv] = optflowGD(I1, I2, alphasq, iter, dt)
%
%
% Implements the gradient descent optical flow algorithm based upon
% the energy functional of Horn and Schunck. Computes the optical flow
% between two images.
%
% Inputs:
%   I1      -the first image in time.
%   I2      -the second image in time.
%   alphasq -the parameter  $\alpha^2$ .
%   iter     -number of gradient descent iterations.
%   dt      -the timestep to use in gradient descent.
%
%===== optflowGD =====
function [Xu, Xv] = optflowGD(I1, I2, alphasq, iter, dt)

Xu = zeros(size(I1));           % This is our initial guess.
Xv = zeros(size(I2));

[Ix, Iy, It] = differentialsGD(I1, I2);

for i=1:iter
    lapXu = laplacian(Xu);
    lapXv = laplacian(Xv);

    grad = Ix .* Xu + Iy .* Xv + It;

    Xu = Xu - dt*(grad .* Ix - alphasq * lapXu);
    Xv = Xv - dt*(grad .* Iy - alphasq * lapXv);
end

end

%----- differentialsGD -----
%
% [Ix, Iy, It] = differentialsGD
%
%----- differentialsGD -----
function [Ix, Iy, It] = differentialsGD(I1, I2)

Kt = (1.0/4.0)*[1,1;1,1];
Kx = (1.0/4.0)*[-1,1;-1,1];
Ky = (1.0/4.0)*[-1,-1;1,1];

It = imfilter(I2, Kt) - imfilter(I1, Kt);
Ix = imfilter(I1, Kx) + imfilter(I2, Kx);
Iy = imfilter(I1, Ky) + imfilter(I2, Ky);

end

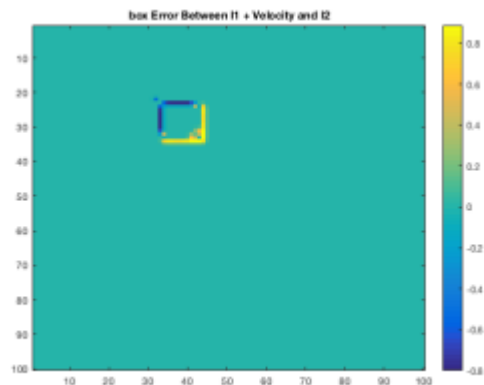
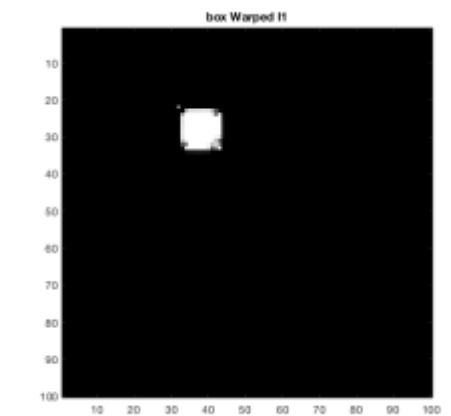
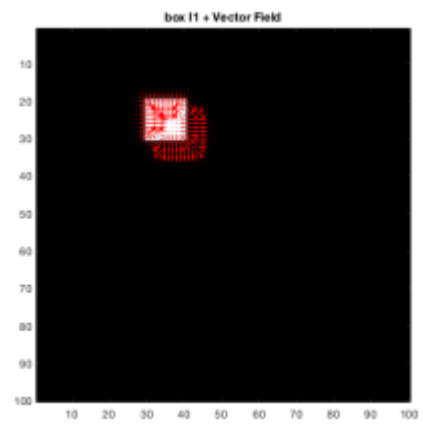
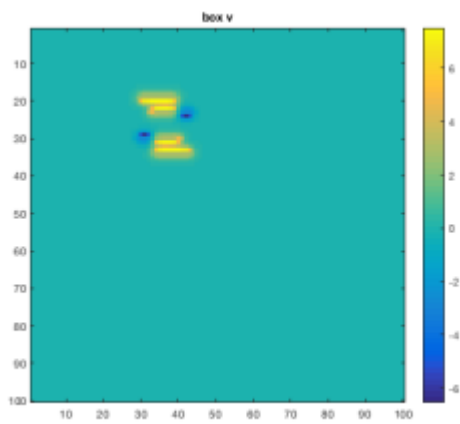
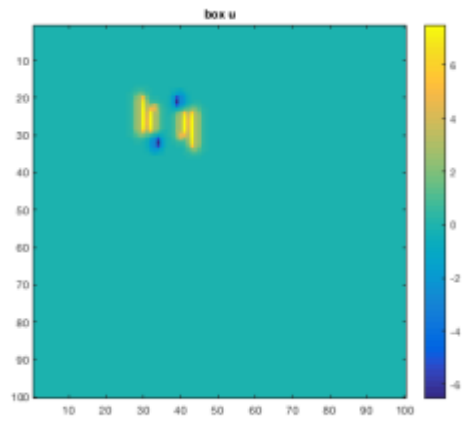
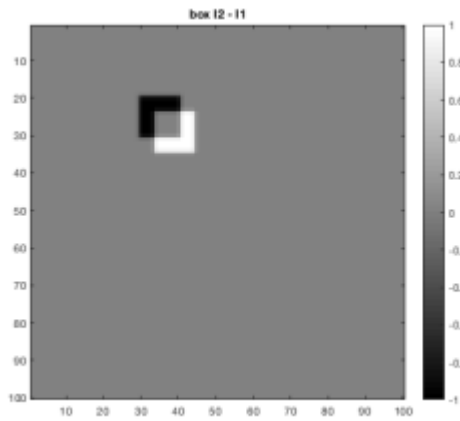
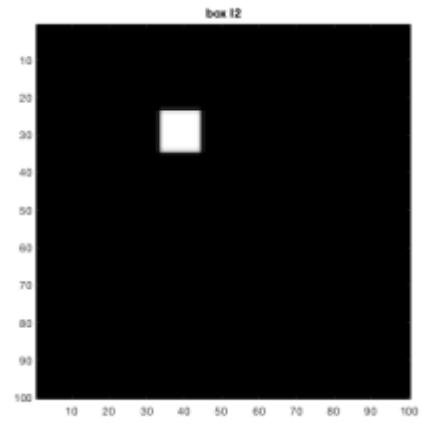
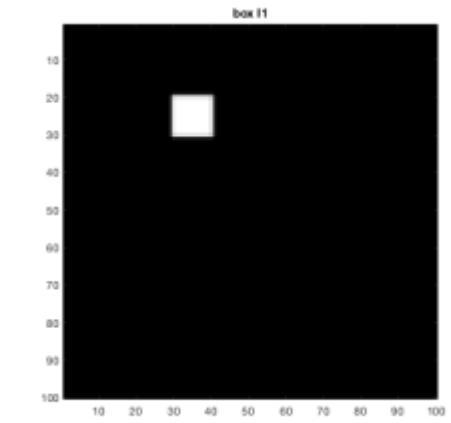
%----- laplacian -----
%
% [ubar] = laplacian(u)
```

GD Method

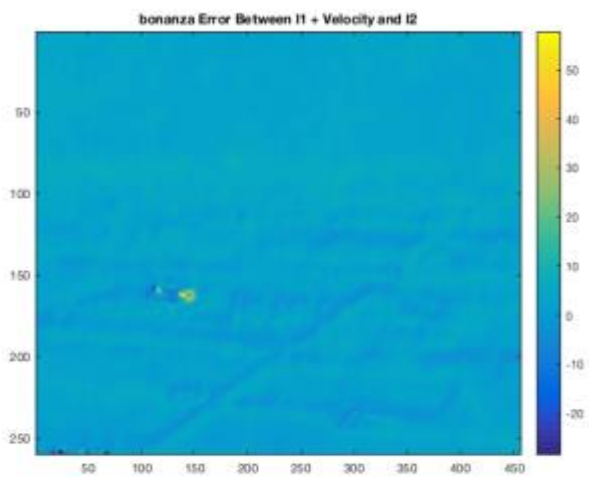
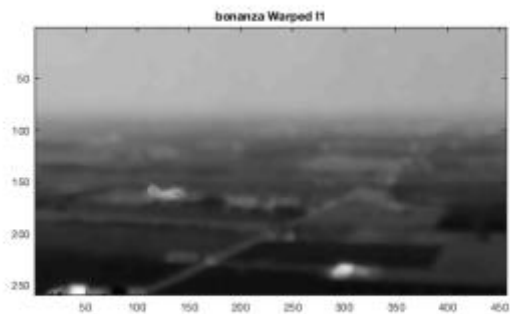
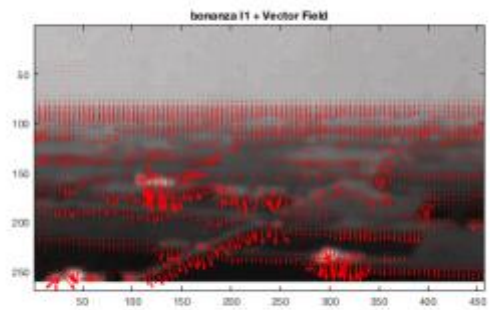
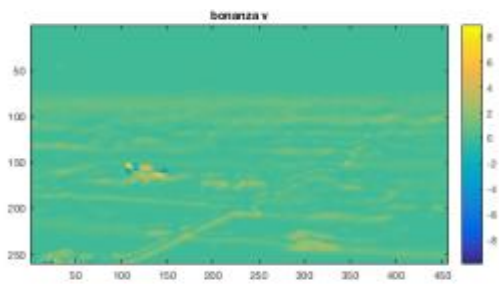
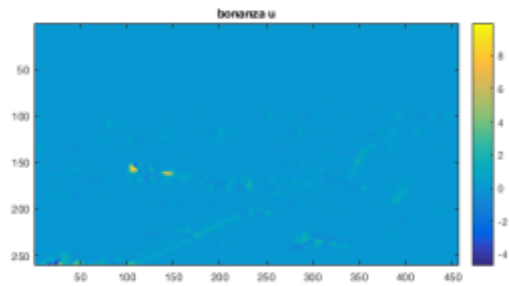
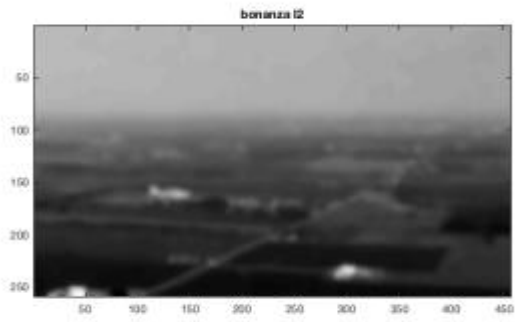
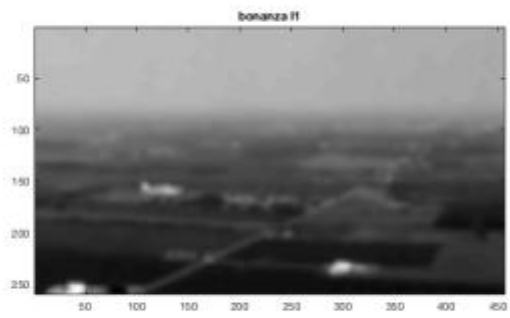
```
%  
%----- laplacian -----  
function [lapu] = laplacian(u)  
k = 3;  
A = (1/12.0)*[1,2,1;2,0,2;1,2,1];  
lapu = k * u + imfilter(u, k * A);  
end
```

The max length optical vector is: 10.9237
The max length optical vector is: 22.7509
The max length optical vector is: 44.9485
The max length optical vector is: 102.8549

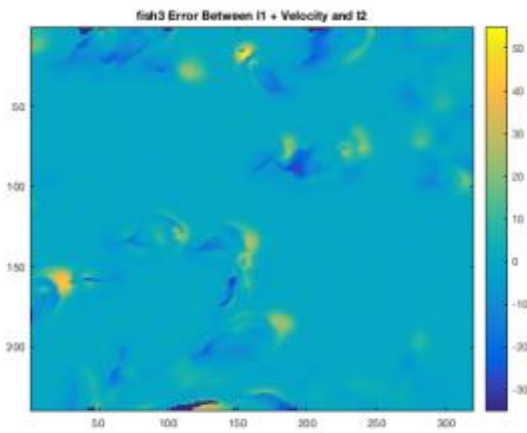
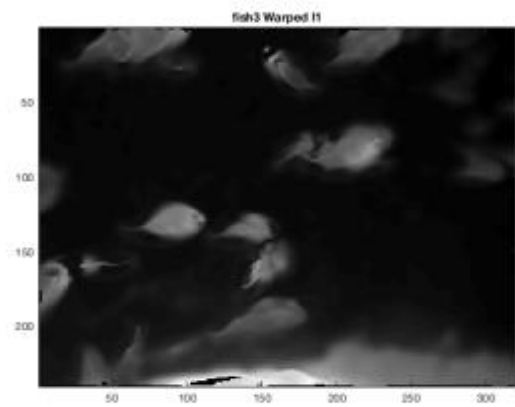
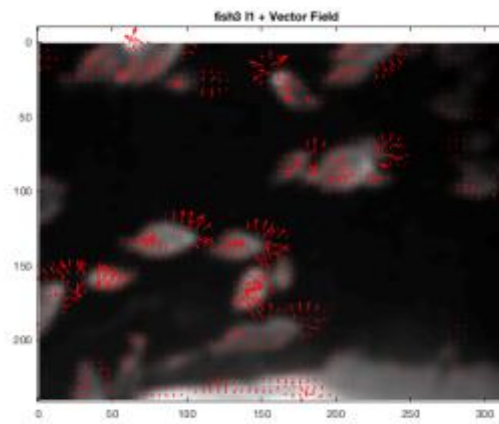
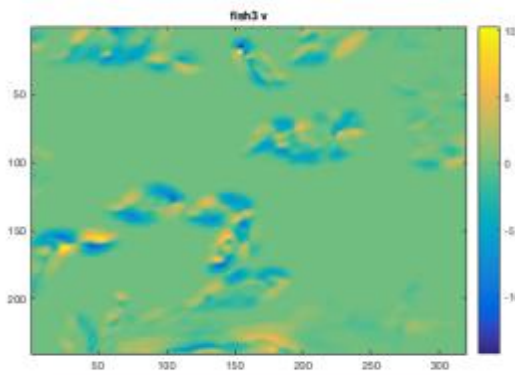
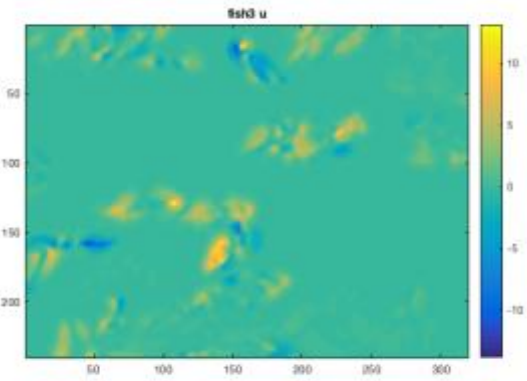
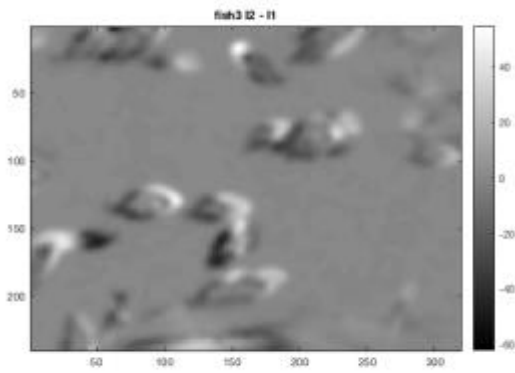
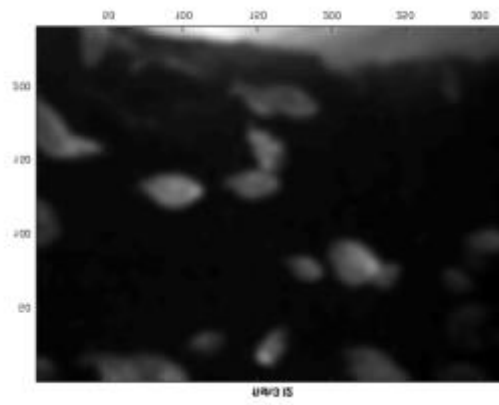
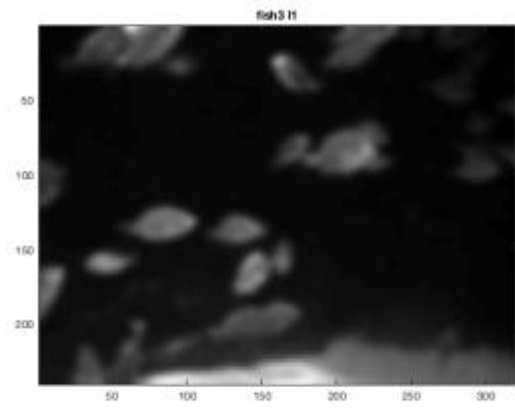
GD Method



GD Method

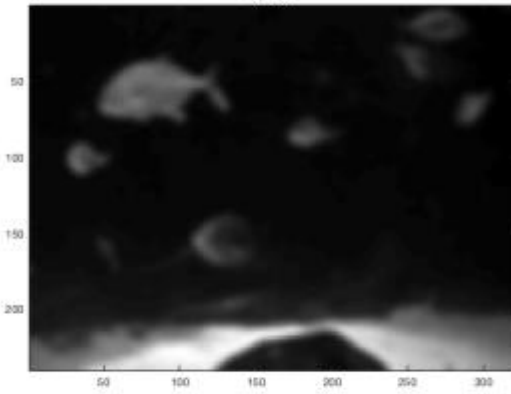


GD Method

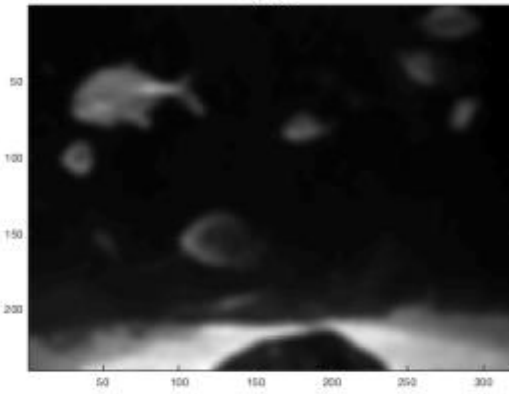


GD Method

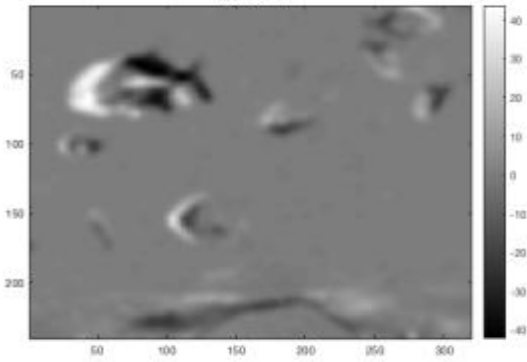
fish4 I1



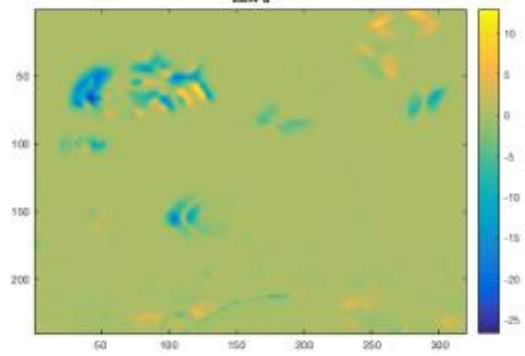
fish4 I2



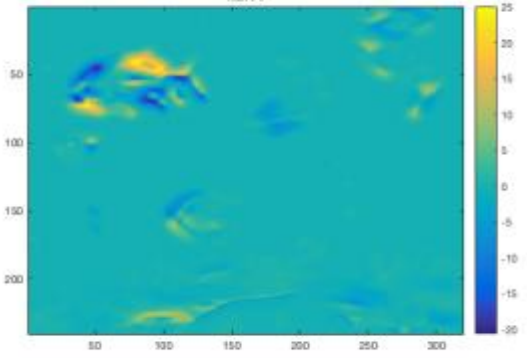
fish4 I2 - I1



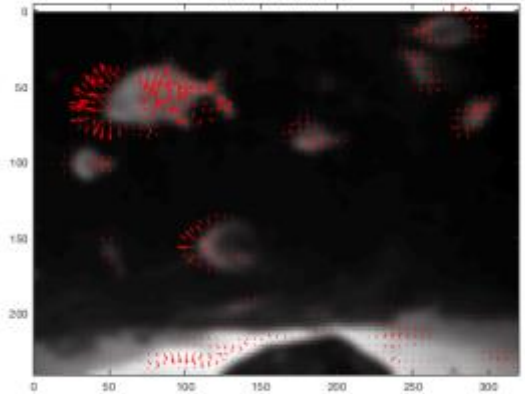
fish4 u



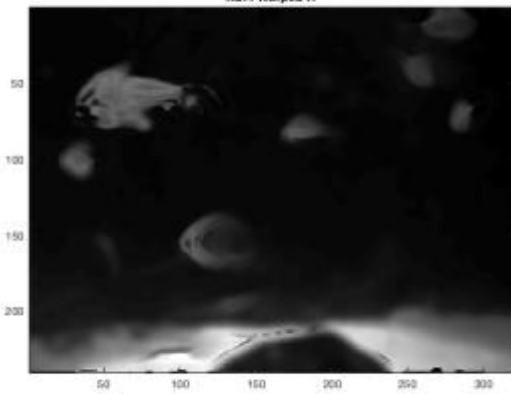
fish4 v



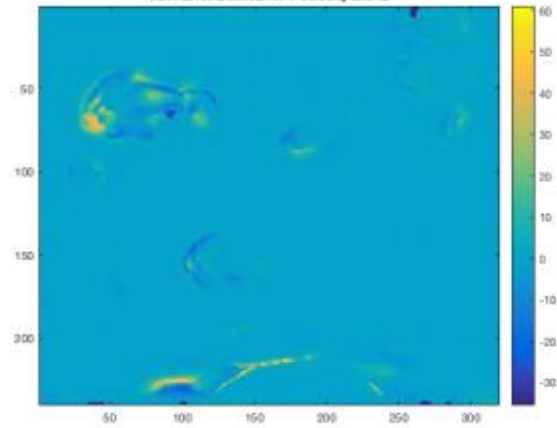
fish4 I1 + Vector Field



fish4 Warped I1



fish4 Error Between I1 + Velocity and I2



HS Method

```
%===== flowme =====
%
% script flowme
%
% This is a script stub that will execute the optical flow solver to be
% written for this homework. The script code will not work as is, and
% must be adjusted. The dynamic range of the images definitely needs to
% be adjusted, as may the smoothness of the image data.
%
%
% Variables in the script:
%   source          - which image sequence source to use.
%   di              - vector spacing for quiver.
%   scale           - what scale to use for plotting vector field.
%   alphasqHS       - the value of alpha squared for optical flow.
%   iterHS          - the number of iterations in the solver.
%
%===== flowme =====

%==(0) Setup variables and get images ready.
close all
load('optflowData.mat');
sources = {'box','bonanza','fish3','fish4'};
for i=1:length(sources)

    source = sources(i);
    source = source{1};
    switch source
    case 'box'
        I1 = imsmooth(box(:,:,1),0.5);
        I2 = imsmooth(box(:,:,3),0.5);
        scale = 1.5;
        alphasqHS = 0.02;
        iterHS = 4000;
        di = 1;
    case 'bonanza'
        I1 = imsmooth(bonanza(:,:,1),2);
        I2 = imsmooth(bonanza(:,:,4),2);
        scale = 2.5;
        alphasqHS = 10;
        iterHS = 100;
        di = 10;
    case 'fish3'
        I1 = imsmooth(fish03(:,:,1),3);
        I2 = imsmooth(fish03(:,:,20),3);
        scale = 2.5;
        alphasqHS = 10;
        iterHS = 100;
        di = 5;
    case 'fish4'
        I1 = imsmooth(fish04(:,:,1),3);
        I2 = imsmooth(fish04(:,:,12),3);
        scale = 2;
        alphasqHS = 10;
        iterHS = 100;
        di = 5;
    end

    %==(1) Actually compute the optical flow.
    [X.u, X.v] = optflowHS(I1, I2, alphasqHS, iterHS);
```

HS Method

```
lenX = abs(X.u + i*X.v);
disp(['The max length optical vector is: ' num2str(max(lenX(:)))]);

%==(2) Display output of the optical flow.
j = 8 * (i - 1);
figure(j + 1);
imagesc(I1);
colormap('gray');
axis image;
title(strcat(source, ' I1'));
figure(j + 2);
imagesc(I2);
colormap('gray');
axis image;
title(strcat(source, ' I2'));
figure(j + 3);
imagesc(I2-I1);
colormap('gray');
colorbar;
axis image;
title(strcat(source, ' I2 - I1'));
figure(j + 4);
imagesc(X.u);
colorbar;
axis image;
title(strcat(source, ' u'));
figure(j + 5);
imagesc(X.v);
colorbar;
axis image;
title(strcat(source, ' v'));
figure(j + 6);
imagesc(I1);
colormap('gray');
axis image;
[M, N] = size(I1);
subx = [1:di:N];
suby = [1:di:M];
[gridx, gridy] = meshgrid(subx, suby);
hold on;
quiver(gridx, gridy, X.u(suby,subx),X.v(suby, subx),scale,'Color','r');
hold off;
title(strcat(source, ' I1 + Vector Field'));

intI = optflow_check(I1, I2, X);

figure(j + 7);
imagesc(intI);
colormap('gray');
axis image;
title(strcat(source, ' Warped I1'));

figure(j + 8);
imagesc(I2 - intI);
colorbar;
title(strcat(source, ' Error Between I1 + Velocity and I2'));

end

%===== optflowHS =====
%
```

HS Method

```
% [Xu, Xv] = optflowHS(I1, I2, alphasq, iter)
%
%
% Implements the Horn and Schunck optical flow algorithm for determination
% of optical flow between two images.
%
% Source:
%   Horn, BKP and Schunck, BG, "Determining Optical Flow,"
%   Artificial Intelligence, vol. 17, pp. 185-204, 1981.
%
% Inputs:
%   I1           -the first image in time.
%   I2           -the second image in time.
%   alphasq      -the parameter  $\alpha^2$ .
%   iter         -number of Gauss-Siedel iterations.
%
%===== optflowHS =====
function [Xu, Xv] = optflowHS(I1, I2, alphasq, iter)

Xu = zeros(size(I1));           % This is the initial guess.
Xv = zeros(size(I2));

[Ix, Iy, It] = differentialsHS(I1, I2);

% Missing code here for the constant parts.
k = 1;
denom = (k * alphasq + Ix.^2 + Iy.^2);

for i=1:iter
    barXu = lapAverage(Xu);
    barXv = lapAverage(Xv);

    % Missing code here for iterative solver.
    gradConst = (Ix .* barXu + Iy .* barXv + It);
    Xu = barXu - (gradConst .* Ix) ./ denom;
    Xv = barXv - (gradConst .* Iy) ./ denom;
end

end

%----- differentialsHS -----
%
% [Ix, Iy, It] = differentialsHS
%
%----- differentialsHS -----
function [Ix, Iy, It] = differentialsHS(I1, I2)

% Missing code here. Make sure to take care of boundary conditions.

Kt = (1.0/4.0)*[1,1;1,1];
Kx = (1.0/4.0)*[-1,1;-1,1];
Ky = (1.0/4.0)*[-1,-1;1,1];

It = imfilter(I2, Kt) - imfilter(I1, Kt);
Ix = imfilter(I1, Kx) + imfilter(I2, Kx);
Iy = imfilter(I1, Ky) + imfilter(I2, Ky);

end

%----- lapAverage -----
%
```

HS Method

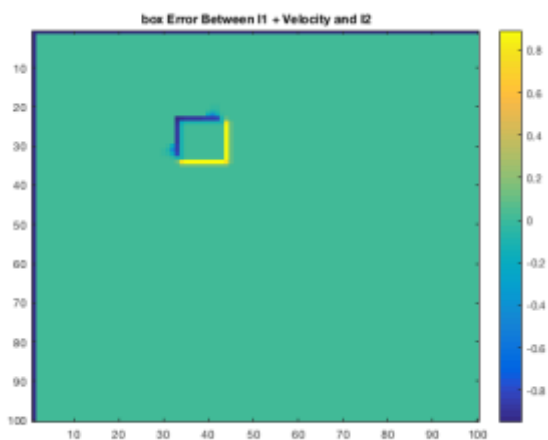
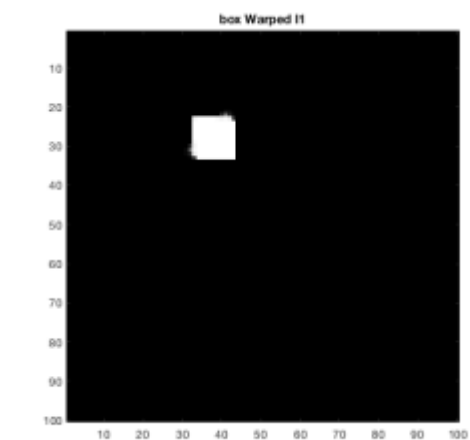
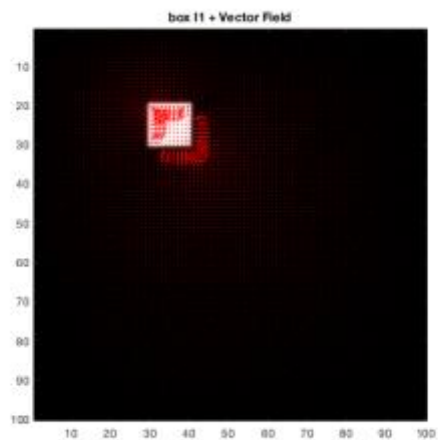
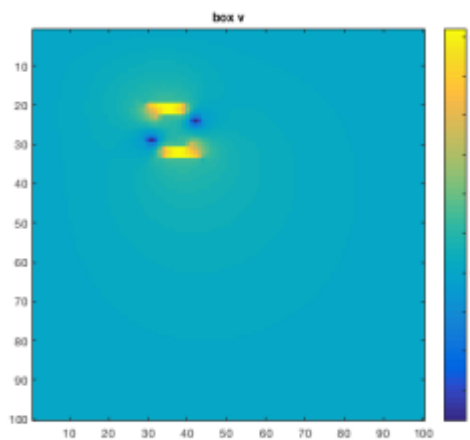
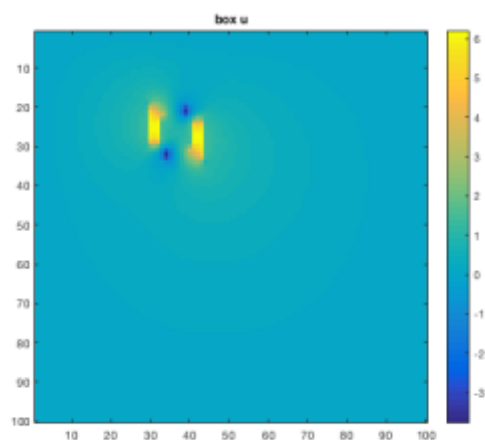
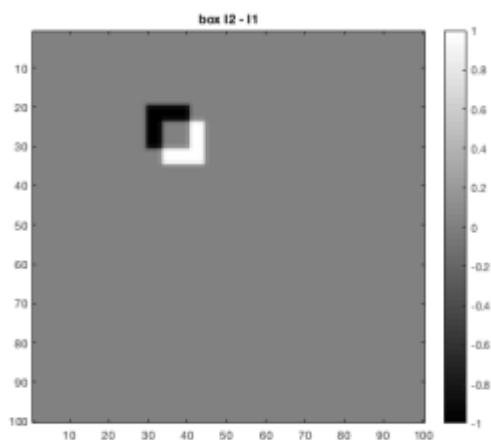
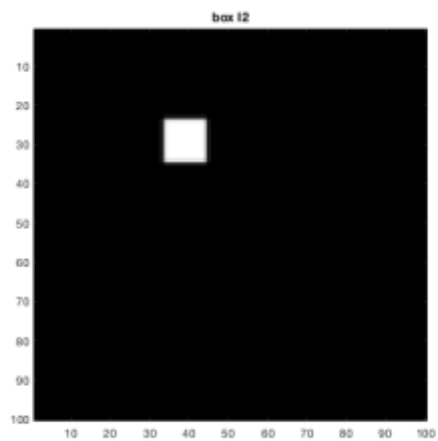
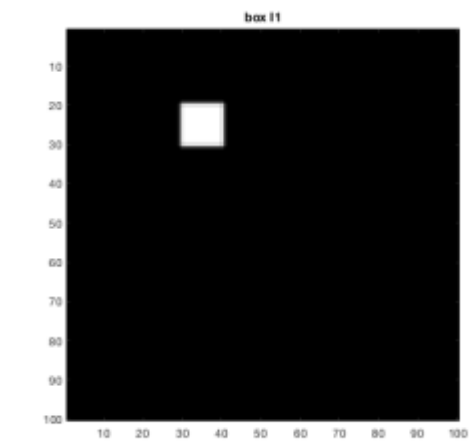
```
% [ubar] = lapAverage(u)
%
%----- lapAverage -----
function [ubar] = lapAverage(u)

% Missing code here. Make sure to take care of boundary conditions.

k = 3;
A = (1/12.0)*[1,2,1;2,0,2;1,2,1];
%lapU = k * u + imfilter(u, k * A);
%ubar = mean(mean(lapU));
ubar = imfilter(u, A);
end
```

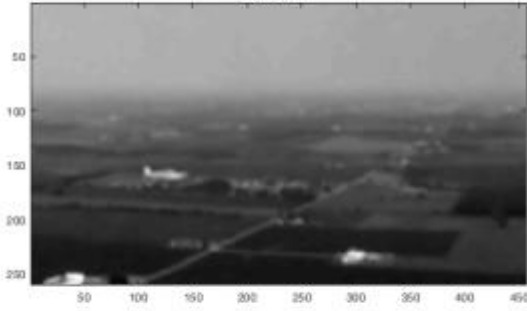
The max length optical vector is: 9.1339
The max length optical vector is: 35.5712
The max length optical vector is: 78.3867
The max length optical vector is: 125.093

HS Method

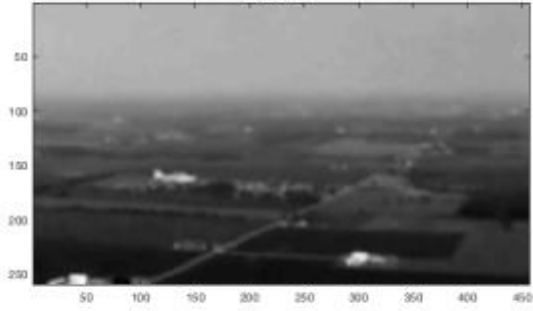


HS Method

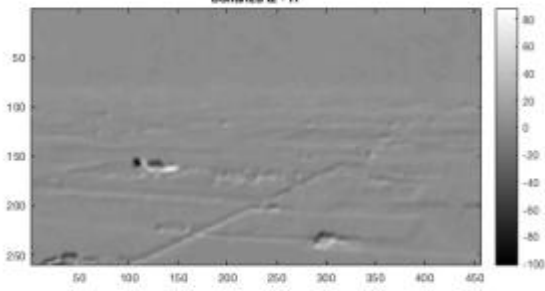
bonanza I1



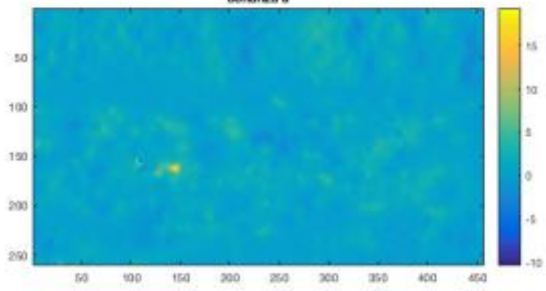
bonanza I2



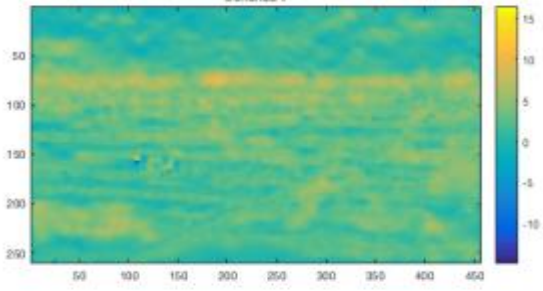
bonanza I2 - I1



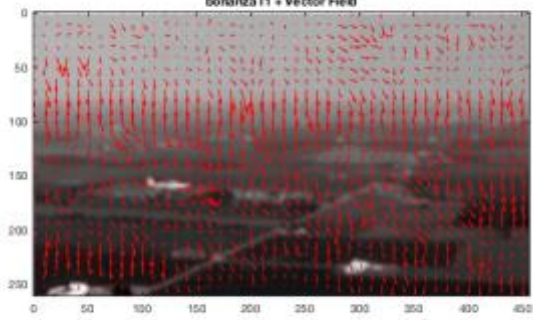
bonanza u



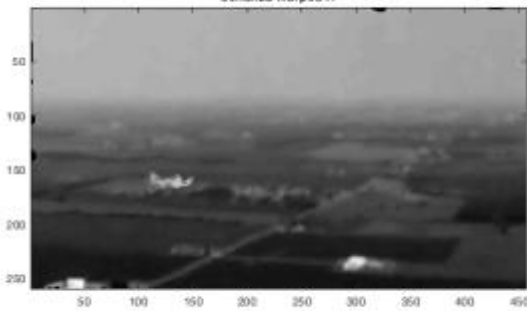
bonanza v



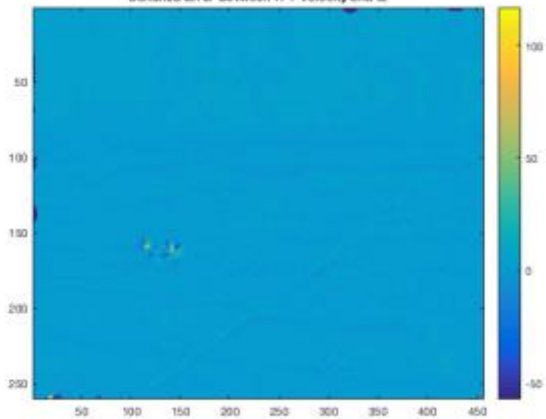
bonanza I1 + Vector Field



bonanza Warped I1

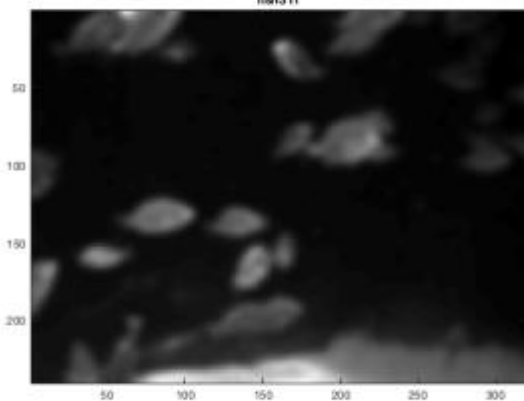


bonanza Error Between I1 + Velocity and I2

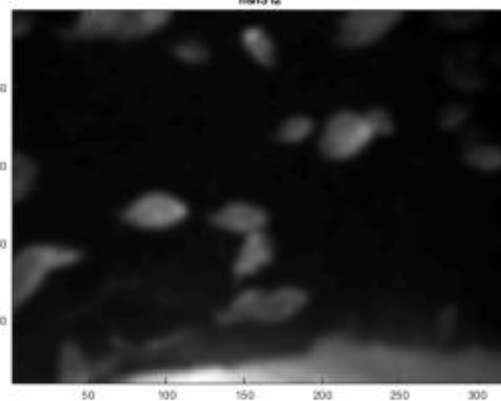


HS Method

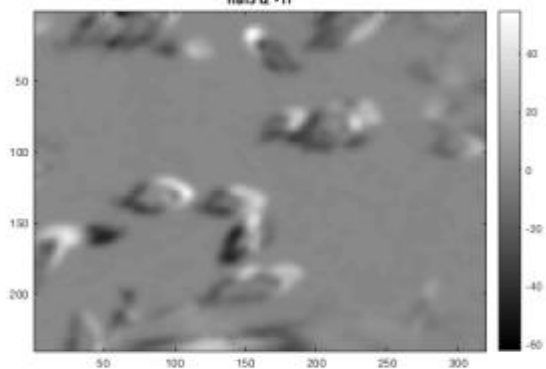
fish3 I1



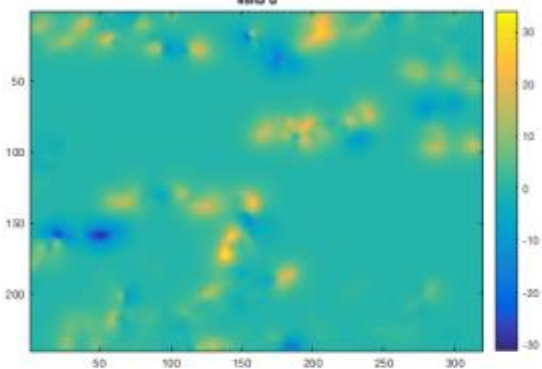
fish3 I2



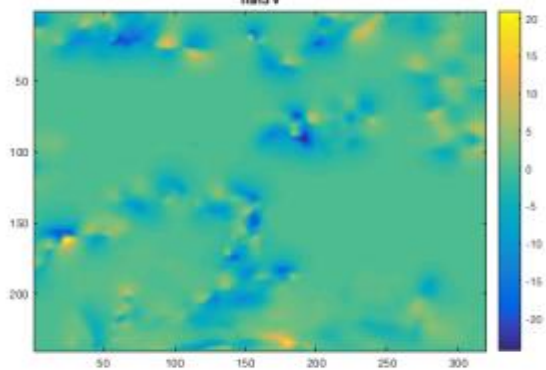
fish3 I2 - I1



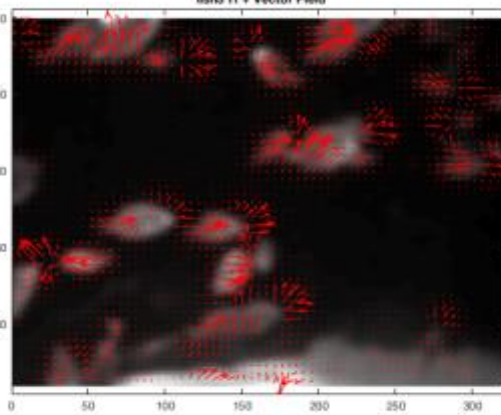
fish3 u



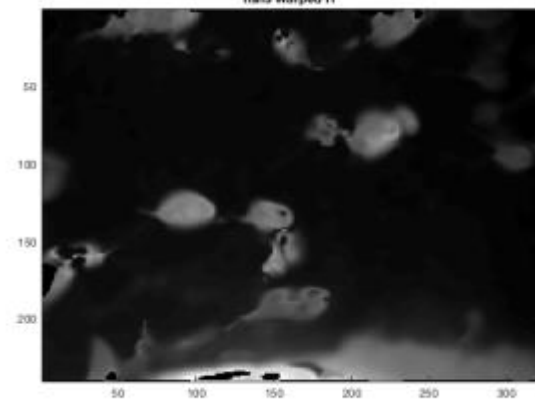
fish3 v



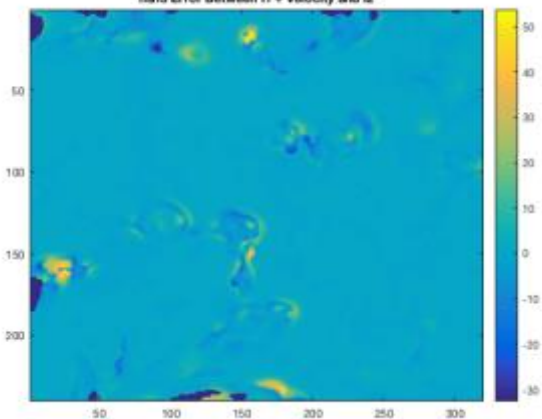
fish3 I1 + Vector Field



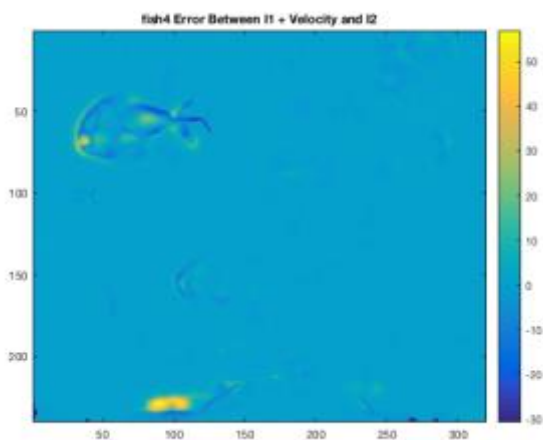
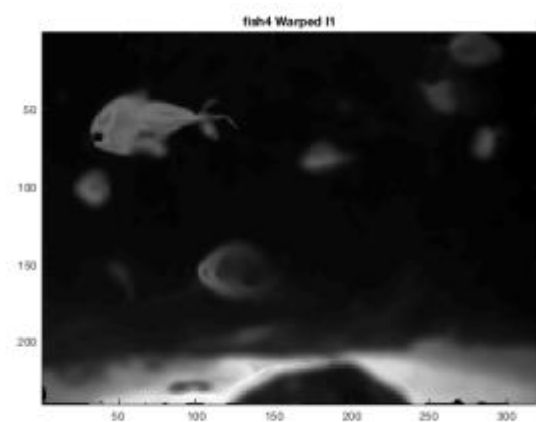
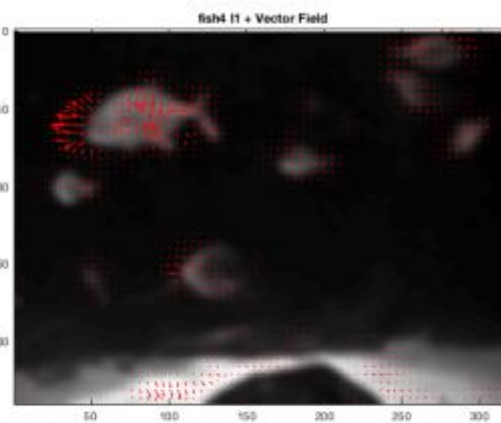
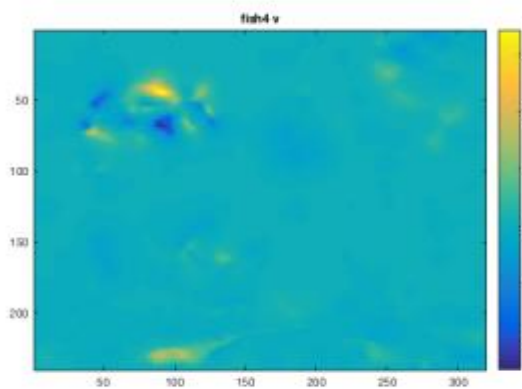
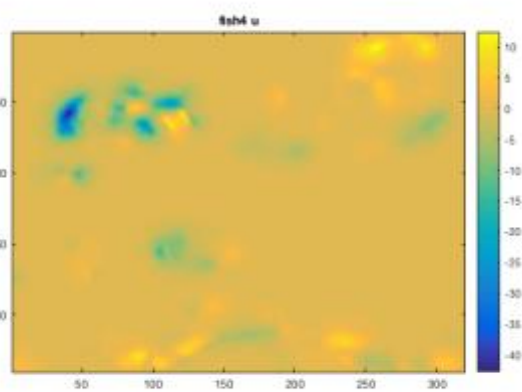
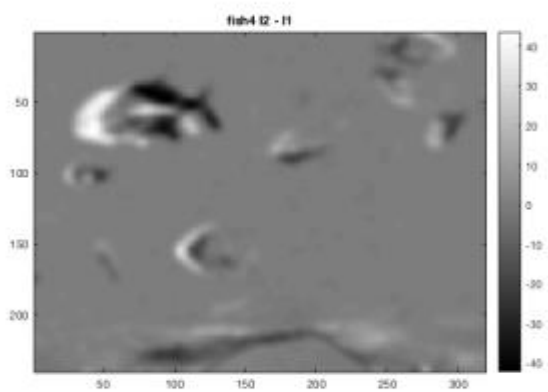
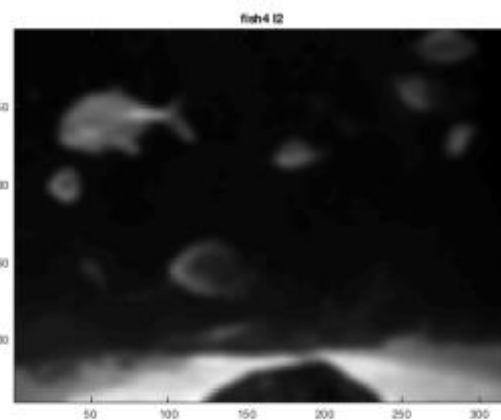
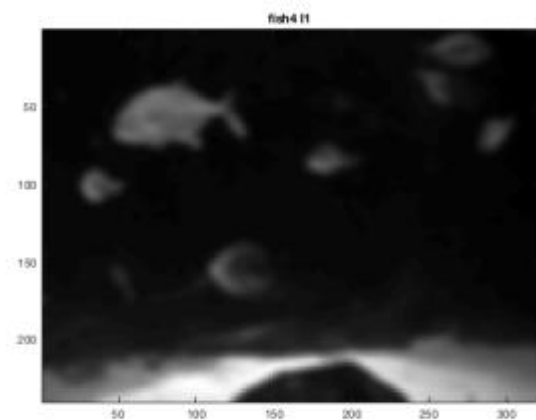
fish3 Warped I1



fish3 Error Between I1 + Velocity and I2



HS Method



The gradient descent solver was much more sensitive to changes in parameter, and was more difficult to determine the correct parameter values to obtain desired results. For example, changing the step size by one order of magnitude led to the maximum length optical vector changing by hundreds of degrees of magnitude. If the parameters were not set correctly, the output optical vectors were infinity or NaN.

The iterative solver was much more predictable, as changes in magnitude of the input produced similar changes in the output. Since there are only two parameters that affect the algorithm (iterations and alpha-squared), it was easier to find the values that produced the desired results.

Both algorithms improved when the images were smoothed with a Gaussian filter. While the simple box image could be smoothed with just a standard deviation of 0.5 and still result in the correct velocity vector, more complex images required a standard deviation of 3.0.