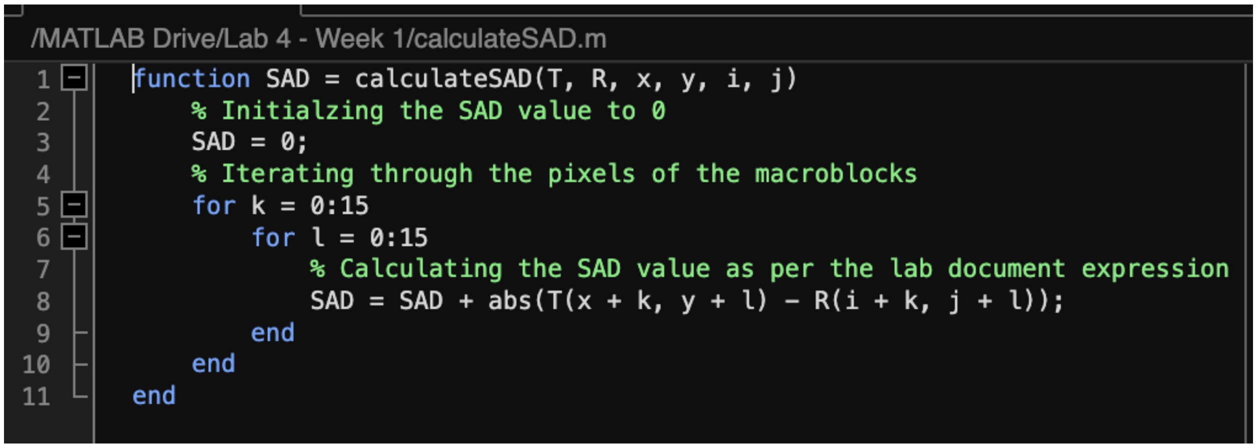
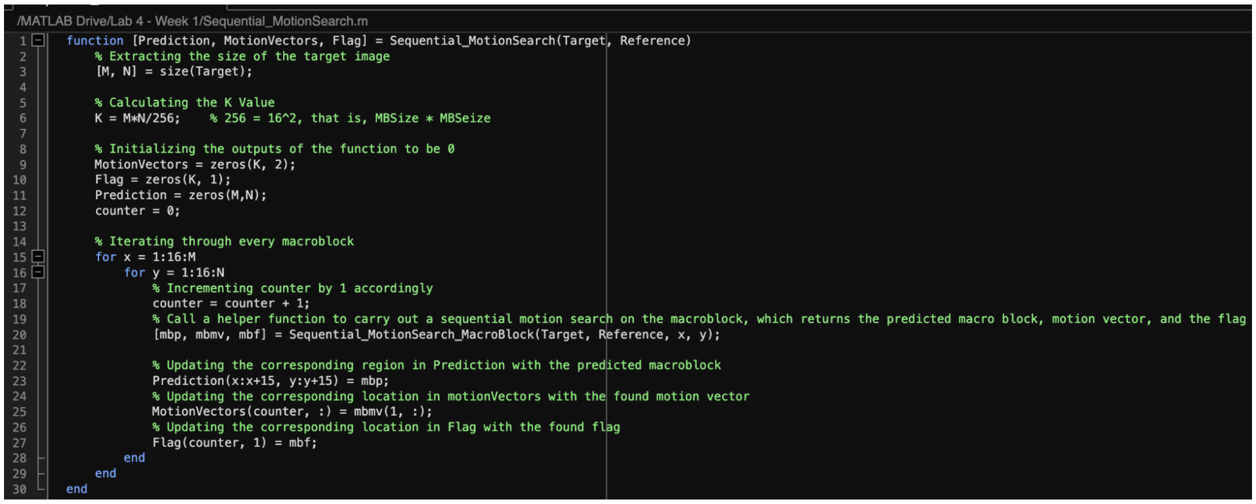
CalculateSAD:


/MATLAB Drive/Lab 4 - Week 1/calculateSAD.m

```
function SAD = calculateSAD(T, R, x, y, i, j)
    % Initialzing the SAD value to 0
    SAD = 0;
    % Iterating through the pixels of the macroblocks
    for k = 0:15
        for l = 0:15
            % Calculating the SAD value as per the lab document expression
            SAD = SAD + abs(T(x + k, y + l) - R(i + k, j + l));
        end
    end
end
```

This code represents the mathematical expression for the SAD calculation provided in the lab document. We first initialize the variable that will hold the final SAD value to 0. We use two for loops, one iterating k from 0 to 15 and the other iterating l from 0 to 15 to represent the double summation in the expression. We iterate through the target frame by adding k and l to x and y, respectively. We iterate through the reference frame by adding k and l to i and j, respectively. The x and y that needs to be added to the reference frame is done in the function which calls calculateSAD, previous to the function call. We take the difference as we iterate, then take the absolute value. We save the final value in SAD.

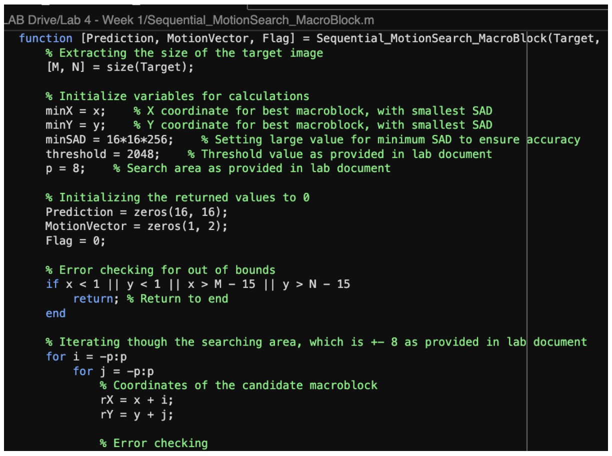
Sequential_MotionSearch:


/MATLAB Drive/Lab 4 - Week 1/Sequential_MotionSearch.m

```
function [Prediction, MotionVectors, Flag] = Sequential_MotionSearch(Target, Reference)
    % Extracting the size of the target image
    [M, N] = size(Target);

    % Calculating the K Value
    K = M*N/256;    % 256 = 16^2, that is, MBSize * MBSeize

    % Initializing the outputs of the function to be 0
    MotionVectors = zeros(K, 2);
    Flag = zeros(K, 1);
    Prediction = zeros(M,N);
    counter = 0;

    % Iterating through every macroblock
    for x = 1:16:M
        for y = 1:16:N
            % Incrementing counter by 1 accordingly
            counter = counter + 1;
            % Call a helper function to carry out a sequential motion search on the macroblock, which returns the predicted macro block, motion vector, and the flag
            [mbp, mbmv, mbf] = Sequential_MotionSearch_MacroBlock(Target, Reference, x, y);

            % Updating the corresponding region in Prediction with the predicted macroblock
            Prediction(x:x+15, y:y+15) = mbp;
            % Updating the corresponding location in motionVectors with the found motion vector
            MotionVectors(counter, :) = mbmv(1, :);
            % Updating the corresponding location in Flag with the found flag
            Flag(counter, 1) = mbf;
        end
    end
end
```

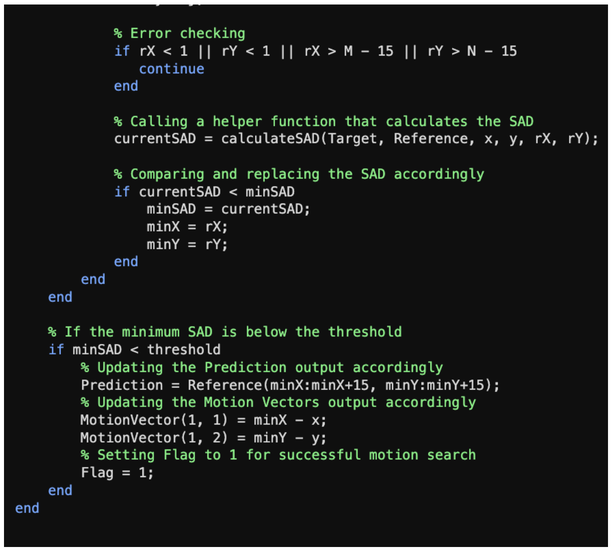
After initializing, calculating, and defining all necessary constants and variables, we iterate through every macro block. While loop iterating, we have a counter to help determine location for motion vectors and flag arrays. We then call a helper function, which completes a sequential motion search on the macro block corresponding to the current iteration. This function returns a predicted macro block, a motion vector, and a flag. The predicted macro block is the best one with the lowest SAD and meeting the threshold value. We then update the final predicted image with the predicted macro block, add the motion vector to the motion vectors, and the flag on whether the motion search for the macro block was successful or not to the final flag array.

Sequential_MotionSearch_Macroblock:



LAB Drive/Lab 4 - Week 1/Sequential_MotionSearch_MacroBlock.m

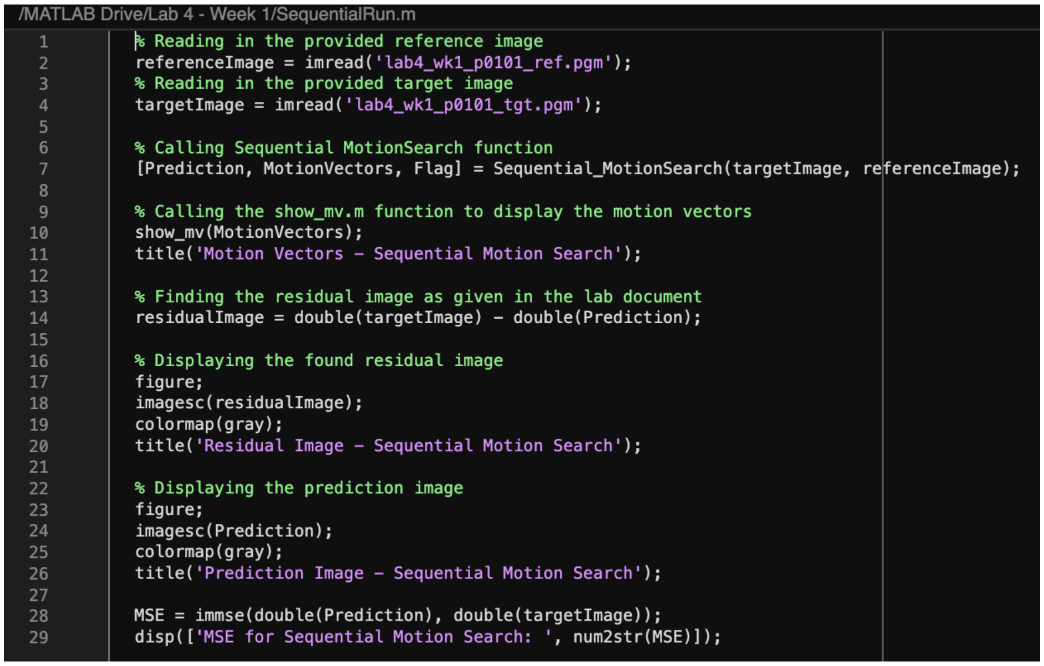
```
function [Prediction, MotionVector, Flag] = Sequential_MotionSearch_MacroBlock(Target,
    % Extracting the size of the target image
    [M, N] = size(Target);

    % Initialize variables for calculations
    minX = x;    % X coordinate for best macroblock, with smallest SAD
    minY = y;    % Y coordinate for best macroblock, with smallest SAD
    minSAD = 16*16*256;    % Setting large value for minimum SAD to ensure accuracy
    threshold = 2048;    % Threshold value as provided in lab document
    p = 8;    % Search area as provided in lab document

    % Initializing the returned values to 0
    Prediction = zeros(16, 16);
    MotionVector = zeros(1, 2);
    Flag = 0;

    % Error checking for out of bounds
    if x < 1 || y < 1 || x > M - 15 || y > N - 15
        return; % Return to end
    end

    % Iterating though the searching area, which is +- 8 as provided in lab document
    for i = -p:p
        for j = -p:p
            % Coordinates of the candidate macroblock
            rX = x + i;
            rY = y + j;

            % Error checking
```



```
            % Error checking
            if rX < 1 || rY < 1 || rX > M - 15 || rY > N - 15
                continue
            end

            % Calling a helper function that calculates the SAD
            currentSAD = calculateSAD(Target, Reference, x, y, rX, rY);

            % Comparing and replacing the SAD accordingly
            if currentSAD < minSAD
                minSAD = currentSAD;
                minX = rX;
                minY = rY;
            end
        end
    end

    % If the minimum SAD is below the threshold
    if minSAD < threshold
        % Updating the Prediction output accordingly
        Prediction = Reference(minX:minX+15, minY:minY+15);
        % Updating the Motion Vectors output accordingly
        MotionVector(1, 1) = minX - x;
        MotionVector(1, 2) = minY - y;
        % Setting Flag to 1 for successful motion search
        Flag = 1;
    end
end
```

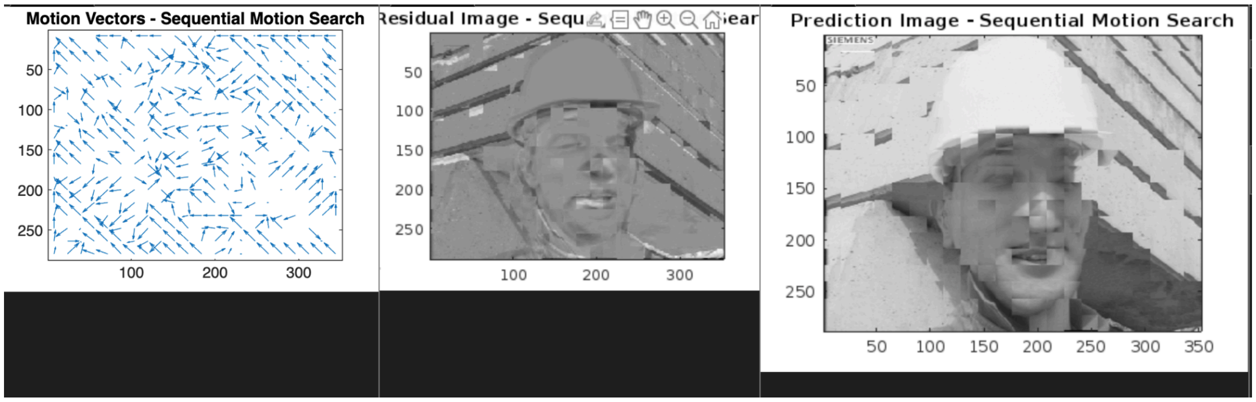
We start with initializing, calculating, and defining all necessary variables and constants. We conduct error checking to ensure all values are in correct bounds. We then iterate throughout the search area, which is given in the lab document. As we iterate, we increase the coordinates by the looping variables to go through the candidate macro block. Again, we perform error checking to ensure everything is in bound. As we have decided that SAD would be our cost measure, we call the helper function calculateSAD to calculate the SAD. If the calculated SAD value is smaller than the previously calculated minimum SAD value, we save the coordinates into minX and minY as these are our best coordinates. Finally, once looping is complete, we check if the minimum SAD value is smaller than the threshold value given in the document. If so, we update the outputs accordingly. We return the prediction macro block, motion vector, and the flag set to 1.

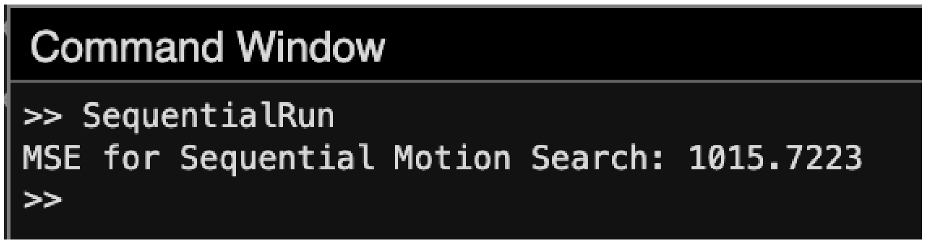
SequentialRun:



/MATLAB Drive/Lab 4 - Week 1/SequentialRun.m

```
% Reading in the provided reference image
referenceImage = imread('lab4_wk1_p0101_ref.pgm');
% Reading in the provided target image
targetImage = imread('lab4_wk1_p0101_tgt.pgm');

% Calling Sequential MotionSearch function
[Prediction, MotionVectors, Flag] = Sequential_MotionSearch(targetImage, referenceImage);

% Calling the show_mv.m function to display the motion vectors
show_mv(MotionVectors);
title('Motion Vectors - Sequential Motion Search');

% Finding the residual image as given in the lab document
residualImage = double(targetImage) - double(Prediction);

% Displaying the found residual image
figure;
imagesc(residualImage);
colormap(gray);
title('Residual Image - Sequential Motion Search');

% Displaying the prediction image
figure;
imagesc(Prediction);
colormap(gray);
title('Prediction Image - Sequential Motion Search');

MSE = immse(double(Prediction), double(targetImage));
disp(['MSE for Sequential Motion Search: ', num2str(MSE)]);
```



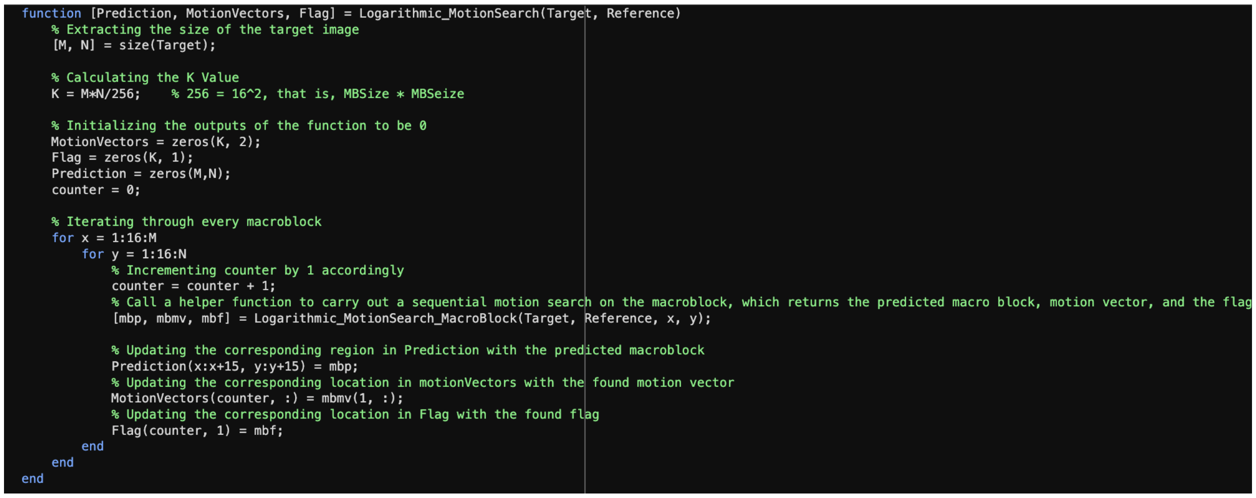


Command Window

```
>> SequentialRun
MSE for Sequential Motion Search: 1015.7223
>>
```

We first load in the target image and reference image that was provided. We then call the Sequential_MotionSearch function to run a sequential motion search. To first display the motion vectors, we use the provided show_mv function to display the motion vectors returned by Sequential_MotionSearch. To display the residual image, we take the difference of the initially provided target image and the returned predicted image, ensuring we convert them into double before taking the difference as outlined in the lab document. We then use imagesc() to display both the residual image and predicted image. Finally, we use the in-built immse() MATLAB function to calculate the MSE of the predicted image and display it in the terminal.

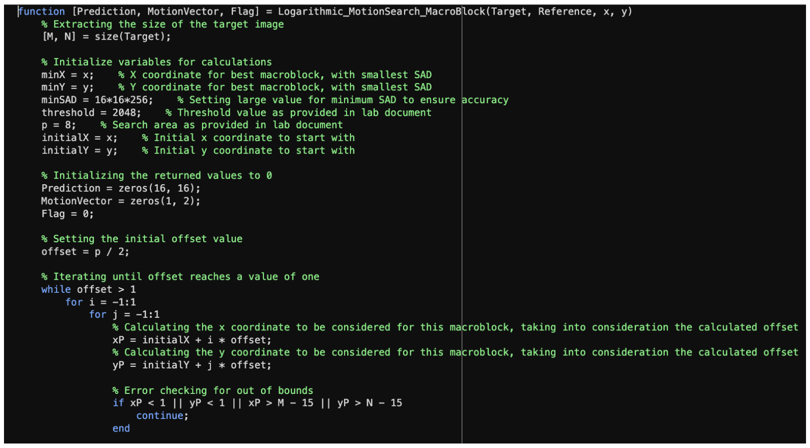
Logarithmic_MotionSearch:



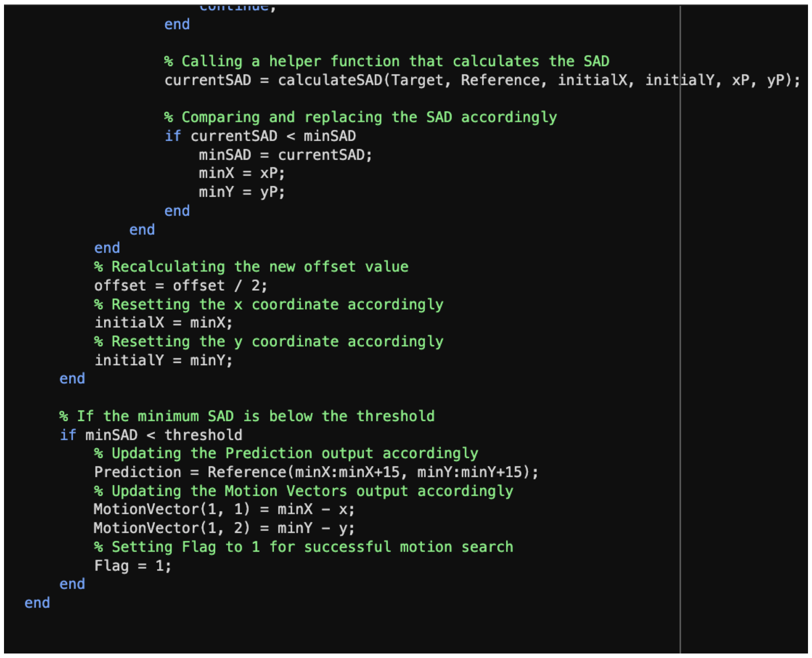
```
function [Prediction, MotionVectors, Flag] = Logarithmic_MotionSearch(Target, Reference)
    % Extracting the size of the target image
    [M, N] = size(Target);

    % Calculating the K Value
    K = M*N/256;    % 256 = 16^2, that is, MBSize * MBSeize

    % Initializing the outputs of the function to be 0
    MotionVectors = zeros(K, 2);
    Flag = zeros(K, 1);
    Prediction = zeros(M,N);
    counter = 0;

    % Iterating through every macroblock
    for x = 1:16:M
        for y = 1:16:N
            % Incrementing counter by 1 accordingly
            counter = counter + 1;
            % Call a helper function to carry out a sequential motion search on the macroblock, which returns the predicted macro block, motion vector, and the flag
            [mbp, mbmv, mbf] = Logarithmic_MotionSearch_MacroBlock(Target, Reference, x, y);

            % Updating the corresponding region in Prediction with the predicted macroblock
            Prediction(x:x+15, y:y+15) = mbp;
            % Updating the corresponding location in motionVectors with the found motion vector
            MotionVectors(counter, :) = mbmv(1, :);
            % Updating the corresponding location in Flag with the found flag
            Flag(counter, 1) = mbf;
        end
    end
end
```

After initializing, calculating, and defining all necessary constants and variables, we iterate through every macro block. While iterating, we have a counter to help determine location for motion vectors and flag arrays. We then call a helper function, which completes a logarithmic motion search on the macro block corresponding to the current iteration. This function returns a predicted macro block, a motion vector, and a flag. The predicted macro block is the best one with the lowest SAD and meeting the threshold value. We then update the final predicted image with the predicted macro block, add the motion vector to the motion vectors, and the flag on whether the motion search for the macro block was successful or not to the final flag array.

Logarithmic_MotionSearch_Macroblock:



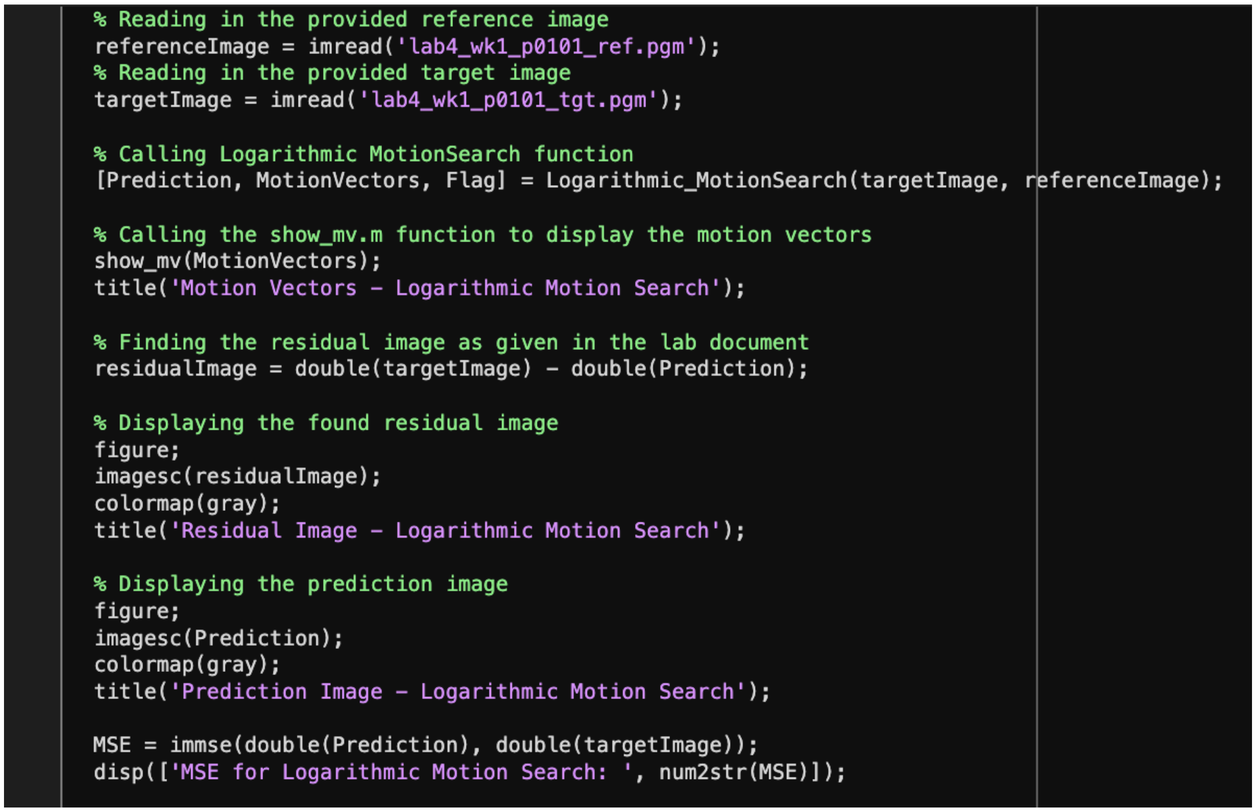
```matlab
function [Prediction, MotionVector, Flag] = Logarithmic_MotionSearch_MacroBlock(Target, Reference, x, y)
    % Extracting the size of the target image
    [M, N] = size(Target);

    % Initialize variables for calculations
    minX = x;    % X coordinate for best macroblock, with smallest SAD
    minY = y;    % Y coordinate for best macroblock, with smallest SAD
    minSAD = 16*16*256;    % Setting large value for minimum SAD to ensure accuracy
    threshold = 2048;    % Threshold value as provided in lab document
    p = 8;    % Search area as provided in lab document
    initialX = x;    % Initial x coordinate to start with
    initialY = y;    % Initial y coordinate to start with

    % Initializing the returned values to 0
    Prediction = zeros(16, 16);
    MotionVector = zeros(1, 2);
    Flag = 0;

    % Setting the initial offset value
    offset = p / 2;

    % Iterating until offset reaches a value of one
    while offset > 1
        for i = -1:1
            for j = -1:1
                % Calculating the x coordinate to be considered for this macroblock, taking into consideration the calculated offset
                xP = initialX + i * offset;
                % Calculating the y coordinate to be considered for this macroblock, taking into consideration the calculated offset
                yP = initialY + j * offset;

                % Error checking for out of bounds
                if xP < 1 || yP < 1 || xP > M - 15 || yP > N - 15
                    continue;
                end
```



```matlab
                    continue;
                end

                % Calling a helper function that calculates the SAD
                currentSAD = calculateSAD(Target, Reference, initialX, initialY, xP, yP);

                % Comparing and replacing the SAD accordingly
                if currentSAD < minSAD
                    minSAD = currentSAD;
                    minX = xP;
                    minY = yP;
                end
            end
        end
        % Recalculating the new offset value
        offset = offset / 2;
        % Resetting the x coordinate accordingly
        initialX = minX;
        % Resetting the y coordinate accordingly
        initialY = minY;
    end

    % If the minimum SAD is below the threshold
    if minSAD < threshold
        % Updating the Prediction output accordingly
        Prediction = Reference(minX:minX+15, minY:minY+15);
        % Updating the Motion Vectors output accordingly
        MotionVector(1, 1) = minX - x;
        MotionVector(1, 2) = minY - y;
        % Setting Flag to 1 for successful motion search
        Flag = 1;
    end
end
```

We start with initializing, calculating, and defining all necessary variables and constants. We then calculate the initial offset, which is p/2, as explained in the lab document. We then iterate through until the offset value hits a value of 1 or lower. We iterate through and search neighbouring areas, through the loops i = -1:1 and j = -1:1. Every time we loop, we calculate the new coordinate by multiplying the incremented initial coordinates with the newly calculated offset value. Every time we loop, the previous offset value is halved again, leading to logarithmic decrease. After each coordinate is calculated, we do error checking to ensure values are in bounds. As we have decided that SAD would be our cost measure, we call the helper function calculateSAD to calculate the SAD. If the calculated SAD value is smaller than the previously calculated minimum SAD value, we save the coordinates into minX and minY as these are our best coordinates. Finally, once looping is complete, we check if the minimum SAD value is smaller than the threshold value given in the document. If so, we update the outputs accordingly. We return the prediction macro block, motion vector, and the flag set to 1.

LogarithmicRun:



```
% Reading in the provided reference image
referenceImage = imread('lab4_wk1_p0101_ref.pgm');
% Reading in the provided target image
targetImage = imread('lab4_wk1_p0101_tgt.pgm');

% Calling Logarithmic MotionSearch function
[Prediction, MotionVectors, Flag] = Logarithmic_MotionSearch(targetImage, referenceImage);

% Calling the show_mv.m function to display the motion vectors
show_mv(MotionVectors);
title('Motion Vectors - Logarithmic Motion Search');

% Finding the residual image as given in the lab document
residualImage = double(targetImage) - double(Prediction);

% Displaying the found residual image
figure;
imagesc(residualImage);
colormap(gray);
title('Residual Image - Logarithmic Motion Search');

% Displaying the prediction image
figure;
imagesc(Prediction);
colormap(gray);
title('Prediction Image - Logarithmic Motion Search');

MSE = immse(double(Prediction), double(targetImage));
disp(['MSE for Logarithmic Motion Search: ', num2str(MSE)]);
```

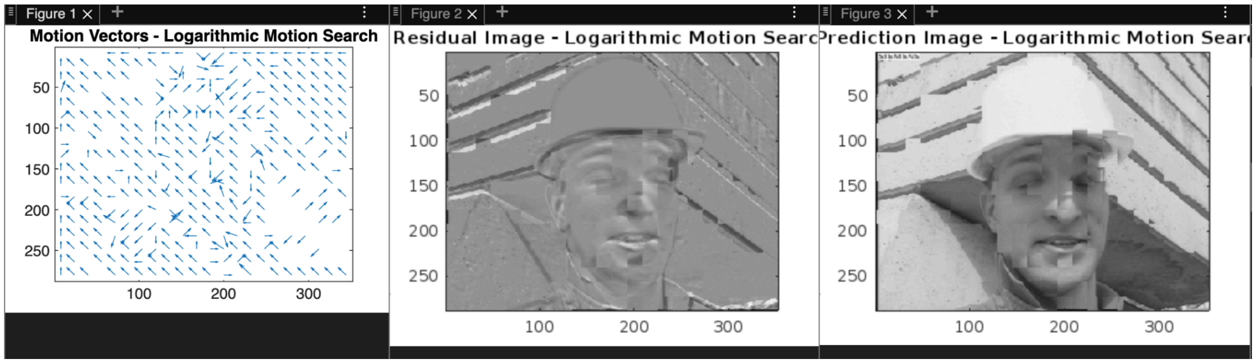
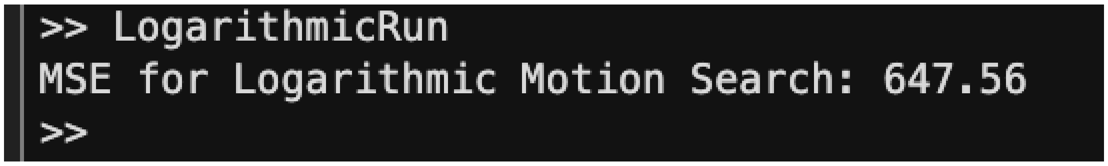


Figure 1
Motion Vectors - Logarithmic Motion Search
50
100
150
200
250
100
200
300
Figure 2
Residual Image - Logarithmic Motion Searc
Figure 3
Prediction Image - Logarithmic Motion Sear



```
>> LogarithmicRun
MSE for Logarithmic Motion Search: 647.56
>>
```

We first load in the target image and reference image that was provided. We then call the Logarithmic_MotionSearch function to run a logarithmic motion search. To first display the motion vectors, we use the provided show_mv function to display the motion vectors returned byLogarithmic_MotionSearch. To display the residual image, we take the difference of the initially provided target image and the returned predicted image, ensuring we convert them into double before taking the difference as outlined in the lab document. We then use imagesc() to display both the residual image and predicted image. Finally, we use the in-built immse() MATLAB function to calculate the MSE of the predicted image and display it in the terminal.