Filter Based Matching

EE5532 Module 3 Assignment

Matthew Spencer

02/28/2021

CONTENTS

Introduction

This module explores programming a custom correlation match filter using Visual Studio C++ to find desired images on a user's desktop tracking them in real-time. It also goes into explaining a SAD block matching algorithm written in MATLAB to analyze the shifts between a pair of stereo images [1].

The correlation match filtering program starts by saving a bitmap RGB image of the user's desktop to memory using C++ and proprietary operating system paint functions supplied by Windows [2]. Once the bitmap is saved, a set of 32-bit Pac-Man ghost images from the video-game is stripped and saved within memory to be cross correlated with 32 x 32 pixel neighborhoods taken from the user's desktop bitmap. Finally once the desired ghost is found, a red rectangle around that neighborhood is drawn onto the frame in real-time, using again more of the proprietary OS functionality supplied by Windows.

The block matching MATLAB algorithm locates differences found in the positioning from one stereo image to the next by placing blocks over the images for reference, and then applying the Sum of Absolute Differences (SAD) equation to measure the differences found between them. The script then places arrows over the shifted image indicating which way, and by how much, one image is shifted over from the other.

This report outlines the procedures and results found experimenting using the concepts outlined in Module 3 of EE5532, and draws some conclusions and further research to look into for the future.

Theory

After implementing a correlation filter using template images and extracting the user's desktop image, a real-time marker can be drawn onto the found image that exists on the target computer's desktop image. The SAD algorithm on the other hand once applied to a block matching algorithm will allow the detection and movement shifts between a set of two stereo images.

Materials

- Windows 10 PC
    - 10400 i5 Intel Processor
    - 16 GB Ram
- MATLAB R2020a
- Visual Studio 2019 C++ [3]

Match Filtering

First, the 32x32 targets of the correlation filter (Pac-Man ghosts) are loaded into the program to be used for comparison against 32 x 32 pixel neighborhoods extracted from the user's desktop image [Fig. 1]. Each image was taken from an emulation of the game found on the Steam marketplace [4]. The ghost target to be found is then chosen by the user by setting the `ghosts` variable within the application.

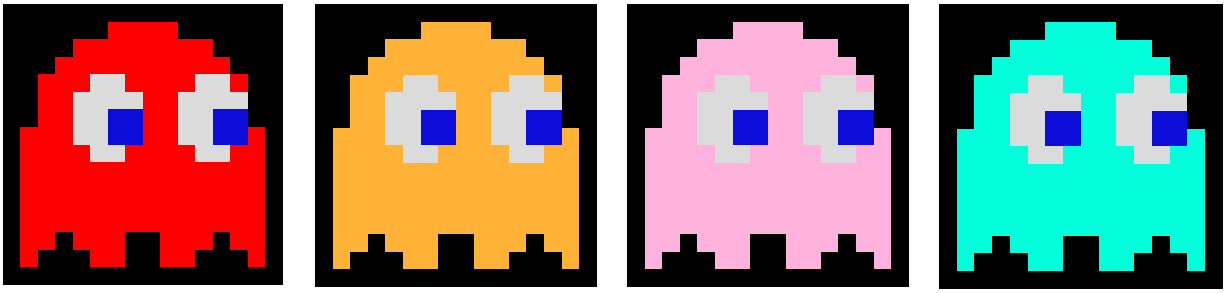

Figure 1: 32-bit Pac-Man Ghosts; Blinky, Clyde, Pinky, Inky [6]

Next, athe C++ script captures the desktop image running the application. The image, technically, is simply an unwritten bitmap (.bmp) stored within the RAM memory of the computer [5]. This internal format is used so the program can access and process the RGB pixel values quicker and more efficiently than writing it to a file, which then has to be manually imported into the application each time.

The desktop image then has the following Equation 1 correlation formula implemented within the program[1] to cross-correlate each 32x32 neighborhood of pixels from the desktop image. This correlation produces an absolute value correlation coefficient between 0 and 1 for each RGB pixel value, which is then averaged. A higher number indicates that the neighborhood being examined has a higher correlation with the original target image, with the highest correlation being chosen.

$$r(x,y) = \frac{\Sigma x_i y_i - n^* \bar{x}\,\bar{y}}{\sqrt{\Sigma x^2{}_i - n^* \bar{x}^{-2}} * \sqrt{\Sigma y^2{}_i - n^* \bar{y}^{-2}}}$$

Equation 1: Pearson's Correlation Coefficient [6]

x= Target Image

y- Input Image

n= Size of Image

[1] The C++ coding implementation can be found at the end of this report in the Appendix.

Once the computer finds the best correlated 32x32 sub-image, a red rectangle is drawn onto the screen to be viewed by the user. This is accomplished by using the Windows OS paint commands mentioned earlier, and an example is shown below in Fig. 2 below of the red ghost "Blinky" target being found and tracked.

The process of iterating through the desktop image and applying the correlation formula found in Equation 1 for the match filter required iterating through the pixels of every column of every row, and then *again* iterating through each of the 32x32 pixels found from every pixel's sub-image. The time complexity of keeping this method was very terrible, reaching $O(n^4)$for every 1920x1080 image captured. This time complexity may be forgivable for some applications, but if there was any hope of making the application detect a desired ghost target on the user's screen in real-time, changes had to be made.

To increase the speed of the match filter, compromises were decided upon. The simplicity of the Pac-Man ghosts was levered, and their common color attributes were used to prioritize which neighborhoods of pixels to correlate using the aforementioned formula. Once a desired threshold was met using the correlation coefficient, the search was abandoned and the target was marked on the user's screen. This cut down on the time needed to run each search, making the average time complexity $O(n^2)$. This is still quite costly, and there are some lower-level C optimization that still can be made, but the target could consistently be targeted and marked for real-time visualization purposes. This method also introduced a higher chance of the algorithm missing the desired target.

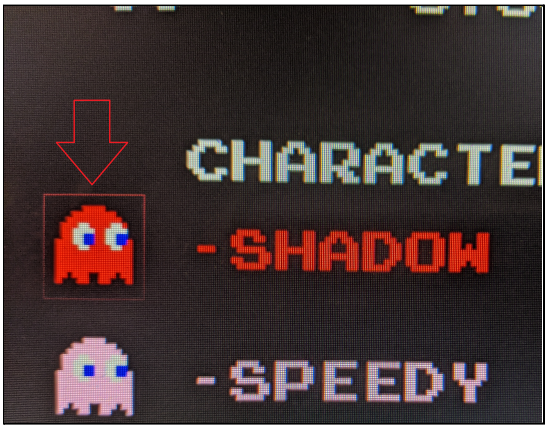

Figure 2: Marked Target Found by Match Filter

SAD

The Sum of Absolute Differences MATLAB algorithm takes an image and divides it into non-overlapping blocks, like a checkerboard. These blocks are used then to find the same locations located within the shifted opposite stereo image. Equation 2 can be used to find each SAD shift.

$$SAD(x, y, u, v) = \sum_{dx=-W}^{W} \sum_{dy=-W}^{W} | I_c(x + dx, y + dy) - I_r(x + u + dx, y + v + dy) |$$

Equation 2: Sum of Absolute Differences [1]

Each square of the reference image has a search region around it in which the correlating pixels are found [Fig. 5]. Once the shift is detected, MATLAB draws an arrow indicating the shift from one stereo image to the next. The results of this algorithm can be seen in Figures 6 - 7 below.

Figure 3: Left Stereo Input Image

Figure 4: Right Stereo Input Image

Figure 5: Block Divided Reference Image with Sub-regions

Figure 6: Detected Shifts in the Left Stereo Input Image

As shown by Figure 6 above, the correct shifts were detected in the left stereo input image of a peanut butter container. These formulas can be used in future work regarding realigning warped or aliased images [7].

Conclusion

The match filtering algorithm applied to the Pac-Man ghosts is quite time inefficient. The process of creating neighborhoods for each individual pixel prevents the program from finding the target in an acceptable time-frame which can be seen by the user. Other problems exist at a lower-level, as the methods used to mark the targets were also using inefficient operating system functions, wherein directly accessing the computer's graphics card buffer would have been much faster. More research is needed to explore these optimization options for further projects, but successfully tracks for this module demonstration.

The SAD algorithm successfully detected the shifts in stereo images, correctly placing arrows wherein the referenced image was shifted from. Applying these techniques to future image processing problems, such as anti-aliasing images in video games, could be an area of interest for research.

References

1. M. Roggemann, “EE5532 Module 3, Lesson 2”, *Michigan Technological University*, 2021, [Online] Available: https://mtu.instructure.com/courses/1347065/pages/module-3-lesson-2?module_item_id=17249969, [Accessed: Feb. 28, 2021]

2. “Bitmaps::Lockbits Method”, *Microsoft*, Dec. 2018, [Online] Available: https://docs.microsoft.com/en-us/windows/win32/api/gdiplusheaders/nf-gdiplusheaders-bitmap-lockbits, [Accessed: Feb. 28, 2021]

3. “Visual Studio 2019”, Microsoft, 2019, [Online] Available: https://visualstudio.microsoft.com/vs/, [Accessed: Feb. 28, 2021]

4. T. Iwatani, S. Ishimura, “Pac-Man”, *Bandai-Namco*, Jul. 1980, [Online] Available: https://store.steampowered.com/app/394160/ARCADE_GAME_SERIES_PACMAN/, [Accessed: Feb. 28, 2021]

5. “Bitmaps class (gdiplusheaders.h)”, *Microsoft*, Dec. 2018, [Online] Available: https://docs.microsoft.com/en-us/windows/win32/api/gdiplusheaders/nl-gdiplusheaders-bitmap, [Accessed: Feb. 28, 2021]

6. “Pearson correlation coefficient”, *Wikipedia the Free Encyclopedia*, Feb. 24, 2021, [Online] Available: https://en.wikipedia.org/wiki/Pearson_correlation_coefficient, [Accessed: Feb. 28, 2021]

7. M. Roggemann, W. Reynolds, “Block-matching algorithm for mitigating aliasing effects in undersampled image sequences”, *Optical Engineering*, Vol. 41. No. 2, Feb. 2002.

Appendix

```cpp
/*****************************************************************************
* Class : EE5532 Module 3
* Name  : Matthew Spencer
* Date  : 02/28/2021
* Pgrm  : Pac Man Ghost Detector
* Desc  : Choose which ghost on your monitor to scan for, draws a red rectangle
*         around it when found. Uses a correlation match filter to get better
*         results during searching.
* Notes : VERY inefficient. Memory leak problem when saving multiple bitmaps.
*****************************************************************************/
#include <Windows.h>
#include <vector>
#include <iostream>
#include <stdint.h>
#include <math.h>
#include <gdiplus.h>
#include <limits.h>
#include <cmath>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <cctype>
#include <fstream>
#include <thread>
#include <new>
#pragma comment( lib, "gdiplus" )
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"

using namespace std;

// Draw rectangle around found match
void drawRect(int x, int y)
{
    HDC screen = ::GetDC(0);
    const COLORREF red = 0x000000FF;
    HBRUSH brush = CreateSolidBrush(red);
    RECT frame;
    frame.left   = x-25;
    frame.top    = y-10;
    frame.right  = x+32;
    frame.bottom = y+48;
    ::FrameRect(screen, &frame, brush);
    ::ReleaseDC(0, screen);
}

```

Appendix 1: Match_Filter.cpp Headers and drawRect Function

```cpp
// Subtract the first image by the second image then add all values
vector<double> correlate(unsigned char* img1, unsigned char* img2, int width, int height, int bbp)
{
    // subtract image 1 by image 2
    vector<double> sumImg1(3, 0);
    vector<double> sumImg2(3, 0);
    vector<double> sumImgs(3, 0);
    vector<double> sqSum1(3, 0);
    vector<double> sqSum2(3, 0);
    int absSum = 0;
    for (int i = 0; i < height; i++)
    {
        for (int j = 0; j < width; j++)
        {
            size_t index = bbp * (i * width + j);
            sumImg1[0] = sumImg1[0] + (double)img1[index + 0];
            sumImg1[1] = sumImg1[1] + (double)img1[index + 1];
            sumImg1[2] = sumImg1[2] + (double)img1[index + 2];

            sumImg2[0] = sumImg2[0] + (double)img2[index + 0];
            sumImg2[1] = sumImg2[1] + (double)img2[index + 1];
            sumImg2[2] = sumImg2[2] + (double)img2[index + 2];

            sumImgs[0] = sumImgs[0] + ((double)img1[index + 0] * (double)img2[index + 0]);
            sumImgs[1] = sumImgs[1] + ((double)img1[index + 1] * (double)img2[index + 1]);
            sumImgs[2] = sumImgs[2] + ((double)img1[index + 2] * (double)img2[index + 2]);

            sqSum1[0] = sqSum1[0] + ((double)img1[index + 0] * (double)img1[index + 0]);
            sqSum1[1] = sqSum1[1] + ((double)img1[index + 1] * (double)img1[index + 1]);
            sqSum1[2] = sqSum1[2] + ((double)img1[index + 2] * (double)img1[index + 2]);

            sqSum2[0] = sqSum1[0] + ((double)img2[index + 0] * (double)img2[index + 0]);
            sqSum2[1] = sqSum2[1] + ((double)img2[index + 1] * (double)img2[index + 1]);
            sqSum2[2] = sqSum2[2] + ((double)img2[index + 2] * (double)img2[index + 2]);

        }
    }

    //Correlate
    vector<double> corr(3, 0);
    int size = height * width;
    corr[0] = abs(((size * sumImgs[0]) - (sumImg1[0] * sumImg2[0])) / sqrt(((size * sqSum1[0]) - (sumImg1[0] * sumImg1[0])) * ((size * sqSum2[0]) - (sumImg2[0] * sumImg2[0]))));
    corr[1] = abs(((size * sumImgs[1]) - (sumImg1[1] * sumImg2[1])) / sqrt(((size * sqSum1[1]) - (sumImg1[1] * sumImg1[1])) * ((size * sqSum2[1]) - (sumImg2[1] * sumImg2[1]))));
    corr[2] = abs(((size * sumImgs[2]) - (sumImg1[2] * sumImg2[2])) / sqrt(((size * sqSum1[2]) - (sumImg1[2] * sumImg1[2])) * ((size * sqSum2[2]) - (sumImg2[2] * sumImg2[2]))));
```

Appendix 1: Match_Filter.cpp C++ Correlation Function Part 1

```cpp
    //Correlate
    vector<double> corr(3, 0);
    int size = height * width;
    corr[0] = abs(((size * sumImgs[0]) - (sumImg1[0] * sumImg2[0])) / sqrt(((size * sqSum1[0]) - (sumImg1[0] * sumImg1[0])) * ((size * sqSum2[0]) - (sumImg2[0] * sumImg2[0]))));
    corr[1] = abs(((size * sumImgs[1]) - (sumImg1[1] * sumImg2[1])) / sqrt(((size * sqSum1[1]) - (sumImg1[1] * sumImg1[1])) * ((size * sqSum2[1]) - (sumImg2[1] * sumImg2[1]))));
    corr[2] = abs(((size * sumImgs[2]) - (sumImg1[2] * sumImg2[2])) / sqrt(((size * sqSum1[2]) - (sumImg1[2] * sumImg1[2])) * ((size * sqSum2[2]) - (sumImg2[2] * sumImg2[2]))));

    // Take care of the divide by zero
    if (isnan(corr[0]))
    {
        corr[0] = 0.0;
    }
    if (isnan(corr[1]))
    {
        corr[1] = 0.0;
    }
    if (isnan(corr[2]))
    {
        corr[2] = 0.0;
    }

    return corr;
}

// Get the correlation coefficient between the image and the current neighborhood of pixels
double get_correlation(int i, int j, byte* scan0, int height, int width, int screenW, unsigned char* img1, int bbp)
{
    // Get neighboorhood of pixels for sub image for correlation
    unsigned char* subImg = NULL;
    subImg = new unsigned char[3 * width * height];
    int x = 0;
    for (int p = i; p < i + height; p++)
    {
        int y = 0;
        for (int q = j; q < j + width; q++)
        {
            // Iterators
            size_t index1 = 3 * (x * width + y);
            size_t index2 = 3 * (p * screenW + q);

            // Save Neighboorhood
            subImg[index1 + 0] = scan0[index2 + 2]; // R
            subImg[index1 + 1] = scan0[index2 + 1]; // G
            subImg[index1 + 2] = scan0[index2 + 0]; // B
            y = y + 1;
        }
        x = x + 1;
    }
```

Appendix 2: Match_filter.cpp C++ Correlation Function Part 2

```
int main(void)
{
    // Choose which ghost to search for.
    // 0 = blinky, 1 = clyde, 2 = pinky, 3 = inky
    int ghost = 0;

    // Load the pac ghosts
    int width, height, bbp;
    unsigned char* blinky = stbi_load("./pac_ghosts/blinky.png", &width, &height, &bbp, 4);
    unsigned char* clyde  = stbi_load("./pac_ghosts/clyde.png", &width, &height, &bbp, 4);
    unsigned char* inky   = stbi_load("./pac_ghosts/inky.png", &width, &height, &bbp, 4);
    unsigned char* pinky  = stbi_load("./pac_ghosts/pinky.png", &width, &height, &bbp, 4);

    // Init coordinates for the boxes
    int blinkyCol   = 0;
    int clydeCol    = 0;
    int inkyCol     = 0;
    int pinkyCol    = 0;
    int blinkyRow   = 0;
    int clydeRow    = 0;
    int inkyRow     = 0;
    int pinkyRow    = 0;
    int blinkyFound = 0;
    int clydeFound  = 0;
    int inkyFound   = 0;
    int pinkyFound  = 0;

    // Init variables for the "game" loop
    using namespace Gdiplus;
    GdiplusStartupInput gdiplusStartupInput;
    ULONG_PTR gdiplusToken;
    GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

    // Get current monitor screenshot and dimensions, tranpose into a bitmap to read pixel values
    while (1)
    {
        // Init coordinates for the boxes
        int blinkyCol   = 0;
        int clydeCol    = 0;
        int inkyCol     = 0;
        int pinkyCol    = 0;
        int blinkyRow   = 0;
        int clydeRow    = 0;
        int inkyRow     = 0;
        int pinkyRow    = 0;
        int blinkyFound = 0;
        int clydeFound  = 0;
        int inkyFound   = 0;
        int pinkyFound  = 0;

```

Appendix 4: Match_Filter.cpp Main Function Part 1

```
191
192         // Thresholds
193         double blinkyThresh = 0.2;
194         double clydeThresh  = 0.1;
195         double pinkyThresh  = 0.3;
196         double inkyThresh   = 0.1;
197
198         // Init bitmap, Get current screenshot of the monitor
199         HDC scrdc, memdc;
200         HBITMAP membit;
201         scrdc = ::GetDC(0);
202         int Height = GetSystemMetrics(SM_CYSCREEN);
203         int Width = GetSystemMetrics(SM_CXSCREEN);
204         memdc = CreateCompatibleDC(scrdc);
205         membit = CreateCompatibleBitmap(scrdc, Width, Height);
206         HBITMAP hOldBitmap = (HBITMAP)SelectObject(memdc, membit);
207         BitBlt(memdc, 0, 0, Width, Height, scrdc, 0, 0, SRCCOPY);
208         Gdiplus::Bitmap bitmap(membit, NULL);
209         int screenW = bitmap.GetWidth();
210         int screenH = bitmap.GetHeight();
211
212         // Lock the bits
213         Gdiplus::Rect rectangle(0, 0, screenW, screenH);
214         UINT flags = ImageLockModeRead;
215         Gdiplus::PixelFormat format = PixelFormat24bppRGB;
216         BitmapData screenshotData;
217         bitmap.LockBits(&rectangle, flags, format, &screenshotData);
218         void* data  = NULL;
219         byte* scan0 = NULL;
220         int bytesPerPixel = 3;
221         scan0 = (byte*)screenshotData.Scan0;
222
223         // Move across the screen and search for ghosts using cross correlation.
224         // Cross correlate 32x32 chunks of the desktop monitor with the previous ghost
225         // correlations, find the closest match to guess where the ghost is.
226         int crop        = 64;
227         double blinkyCC = 0;
228         double clydeCC  = 0;
229         double pinkyCC  = 0;
230         double inkyCC   = 0;
231         int offset      = 5;
232
```

Appendix 5: Match_Filter Main Function Part 2

```
233         // Loop through the desktop image
234         int i = crop;
235         while (i < screenH - crop)
236         {
237             int j = crop;
238             while (j < screenW - crop)
239             {
240                 // Init index for desktop bitmap
241                 size_t index = 3 * (i * screenW + j);
242 
243                 if (blinkyFound == 0 || clydeFound == 0 || pinkyFound == 0 || inkyFound == 0)
244                 {
245                     // Check for blinky
246                     if (ghost == 0 && blinkyFound == 0 && scan0[index + 2] == 255 && scan0[index + 1] == 0 && scan0[index + 0] == 0)
247                     {
248                         double cc = get_correlation(i-offset, j-offset, scan0, height, width, screenW, blinky, bbp);
249                         if (cc <= 1 && cc >= blinkyThresh)
250                         {
251                             blinkyCol = j;
252                             blinkyRow = i;
253                             blinkyFound = 1;
254                         }
255                     }
256 
257                     // Check for clyde
258                     if (ghost == 1 && clydeFound == 0 && scan0[index + 2] == 255 && scan0[index + 1] == 178 && scan0[index + 0] == 54)
259                     {
260                         double cc = get_correlation(i-offset, j-offset, scan0, height, width, screenW, clyde, bbp);
261                         if (cc <= 1 && cc >= clydeThresh)
262                         {
263                             clydeCol = j;
264                             clydeRow = i;
265                             clydeFound = 1;
266                         }
267                     }
268 
269                     // Check for pinky
270                     if (ghost == 2 && pinkyFound == 0 && scan0[index + 2] == 255 && scan0[index + 1] == 178 && scan0[index + 0] == 219)
271                     {
272                         double cc = get_correlation(i-offset, j-offset, scan0, height, width, screenW, pinky, bbp);
273                         if (cc <= 1 && cc >= pinkyThresh)
274                         {
275                             pinkyCol = j;
276                             pinkyRow = i;
277                             pinkyFound = 1;
278                         }
279                     }
```

Appendix 6: Match_Filter Main Function Part 3

```
281                     // Check for inky
282                     if (ghost == 3 && inkyFound == 0 && scan0[index + 2] == 0 && scan0[index + 1] == 255 && scan0[index + 0] == 219)
283                     {
284                         double cc = get_correlation(i-offset, j-offset, scan0, height, width, screenW, inky, bbp);
285                         if (cc <= 1 && cc >= inkyThresh)
286                         {
287                             inkyCol = j;
288                             inkyRow = i;
289                             inkyFound = 1;
290                         }
291                     }
292                 }
293                 // Increase column
294                 j++;
295             }
296             // Increase row
297             i++;
298 
299             // Check to make sure we need to keep looping
300             if (blinkyFound == 1 && clydeFound == 1 && pinkyFound == 1 && inkyFound == 1)
301             {
302                 break;
303             }
304         }
305 
306         // Draw a ghost if they are found
307         if (blinkyFound == 1 || clydeFound == 1 || pinkyFound == 1 || inkyFound == 1)
308         {
309             // Draw Rectangle
310             if (blinkyFound == 1)
311             {
312                 drawRect(blinkyCol, blinkyRow);
313             }
314             if (clydeFound == 1)
315             {
316                 drawRect(clydeCol, clydeRow);
317             }
318             if (pinkyFound == 1)
319             {
320                 drawRect(pinkyCol, pinkyRow);
321             }
322             if (inkyFound == 1)
323             {
324                 drawRect(inkyCol, inkyRow);
```

Appendix 6: Match_Filter Main Function Part 4

```
            // Clean up the old image
            SelectObject(memdc, hOldBitmap);
            DeleteObject(memdc);
            DeleteObject(membit);
            ::ReleaseDC(0, scrdc);
        }
        // Free the pac ghosts
        stbi_image_free(blinky);
        stbi_image_free(clyde);
        stbi_image_free(inky);
        stbi_image_free(pinky);
        GdiplusShutdown(gdiplusToken);

        // Exit program
        return 0;
    }
```

Appendix 6: Match_Filter Main Function Part 5

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Pgrm   : Module 4 Part 2
% Author : Dr. M. Roggemann
% Edited : Matthew Spencer
% Class  : EE5367
% Date   : 02/26/2020
% Desc   : SAD image shift of peanut butter
% Input  : A pair of stereo images
% Output : Results of the block algorithms detecting the shifts.
% Notes  : None.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function Module_3()
%------------------------------------------------------------ Initializations
    % Clear all images and cmd window
    close all;
    clc;

    %BMA demo on a stereo pair
    img1 = imread("Peanut_1.jpg");
    img2 = imread("Peanut_2.jpg");
    t1   = double(rgb2gray(img1));
    t2   = double(rgb2gray(img2));

    % Crop the image
    [rlen,clen] = size(t1);
    rc          = rlen/2;
    cc          = clen/2;
    halflen     = 950;

    % Image 1 crop
    rstop  = rlen-600;
    rstart = rstop - 2*halflen + 1;
    cstart = cc - halflen;
    cstop  = cstart + 2*halflen - 1;
    leftim = t1(rstart:rstop,cstart:cstop);

    % Image 2 crop
    rstop   = rlen-600;
    rstart  = rstop - 2*halflen + 1;
    cstart  = cc - halflen;
    cstop   = cstart + 2*halflen - 1;
    rightim = t2(rstart:rstop,cstart:cstop);

    % side length in pixels of both images, which are now square
    len = 2*halflen;

```

Appendix 7: Module_3.m Part 1

```
%------------------------------------------------------------ Input Images
    figure
    subplot(2,1,1)
    imagesc(leftim)
    axis('image')
    colormap(gray(256))
    axis('off')
    title('left image')
    subplot(2,1,2)
    imagesc(rightim)
    axis('image')
    colormap(gray(256))
    axis('off')
    title('rightimage')

%------------------------------------------------------------ Calculations
    %set up for shift sensing
    sub_region_sz =input('What size subregion (default is 30)?');
    if isempty(sub_region_sz);
       sub_region_sz=30;
    end;

    search_wid = input('What size search region (default is 12)?');
    if isempty(search_wid);
       search_wid=12;
    end;

    %prevents search performed by bma from going outside bounds of
    % image and crashing
    procsz    = len - 2*sub_region_sz;
    proc_rcen = len/2 + 1;
    proc_ccen = len/2 + 1;

    len1 = procsz+2*sub_region_sz;  %subimage size to process
    cen1 = len1/2 + 1;  %center of arrays for performing ffts

    rstart1 = proc_rcen-len/2;
    rstop1  = proc_rcen+len/2-1;
    cstart1 = proc_ccen-len/2;
    cstop1  = proc_ccen+len/2-1;

    len2 = len1;
    cen2 = len2/2 + 1;

    upsamp_row = len1;
    upsamp_col = len1;
```

Appendix 8: Module_3.m Part 2

```
93
94      %setup for block matching algorithm
95      [sub_row_cen_arr, sub_col_cen_arr,...
96               rstart_mat, rstop_mat, cstart_mat,...
97               cstop_mat, sub_reg_lut, num_row_steps,...
98               num_col_steps] = ...
99               bma_setup(sub_region_sz, search_wid, len1, len1,...
100                        upsamp_row, upsamp_col);
101
102     %create visualization of the blocks
103     cnt = 0;
104     for m=1:num_row_steps
105         for n=1:num_col_steps
106             cnt = cnt + 1;
107             ulX = cstart_mat(m,n);
108             ulY = rstart_mat(m,n);
109             lrX = cstop_mat(m,n);
110             lrY = rstop_mat(m,n);
111             boxes{cnt}.X = [ulX lrX lrX ulX ulX]';
112             boxes{cnt}.Y = [ulY ulY lrY lrY ulY]';
113         end
114     end
115
116     % Block Visualization
117     figure
118     imagesc(leftim)
119     axis('image')
120     hold
121     for q=1:cnt
122         plot(boxes{q}.X, boxes{q}.Y, 'r');
123     end
124     title('visualization of the blocks')
125
126     %select left image as reference image
127     ref_img = leftim - min(min(leftim));
128     %normalize
129     ref_img = ref_img/max(max(ref_img));
130
131     %select right image as shifted image
132     sh_img = rightim - min(min(rightim));
133
134     %normalize
135     sh_img = sh_img/max(max(sh_img));
136
137     %can be used to indicate bad blocks.  not used here
138     good_reg_lut = ones(num_row_steps, num_col_steps);
139
140     % Call SAD functions
141     [row_sh_arr, col_sh_arr] = ...
142                 disp_sub3d_SAD(ref_img, sh_img, sub_region_sz,...
143                                search_wid, sub_row_cen_arr,...
144                                sub_col_cen_arr, rstart_mat, rstop_mat,...
145                                cstart_mat, cstop_mat, sub_reg_lut,...
146                                num_row_steps, num_col_steps, good_reg_lut);
```

Appendix 9: Module_3.m Part 3

```
147     % Get shifts
148     X = sub_col_cen_arr;
149     U = col_sh_arr;
150
151     Y = sub_row_cen_arr;
152     V = -row_sh_arr;
153
154 %------------------------------------------------------------- Merged Images
155     % Make one side-by-side image from the stereo image
156     sidebyside = zeros(len,2*len);
157     sidebyside(1:len,1:len) = leftim;
158     sidebyside(1:len,(len+1):(2*len)) = rightim;
159
160     % Shifted Merged images
161     figure
162     imagesc(sidebyside)
163     axis('image')
164     hold
165     %note that quiver draws arrows that are not perfectly to scale...
166     quiver(X,Y,U,V,'r')
167     title('detected shifts in right image relative to the left image')
168
169 end
170 % EOF ==========================================================================
```

Appendix 10: Module_3.m Part 4