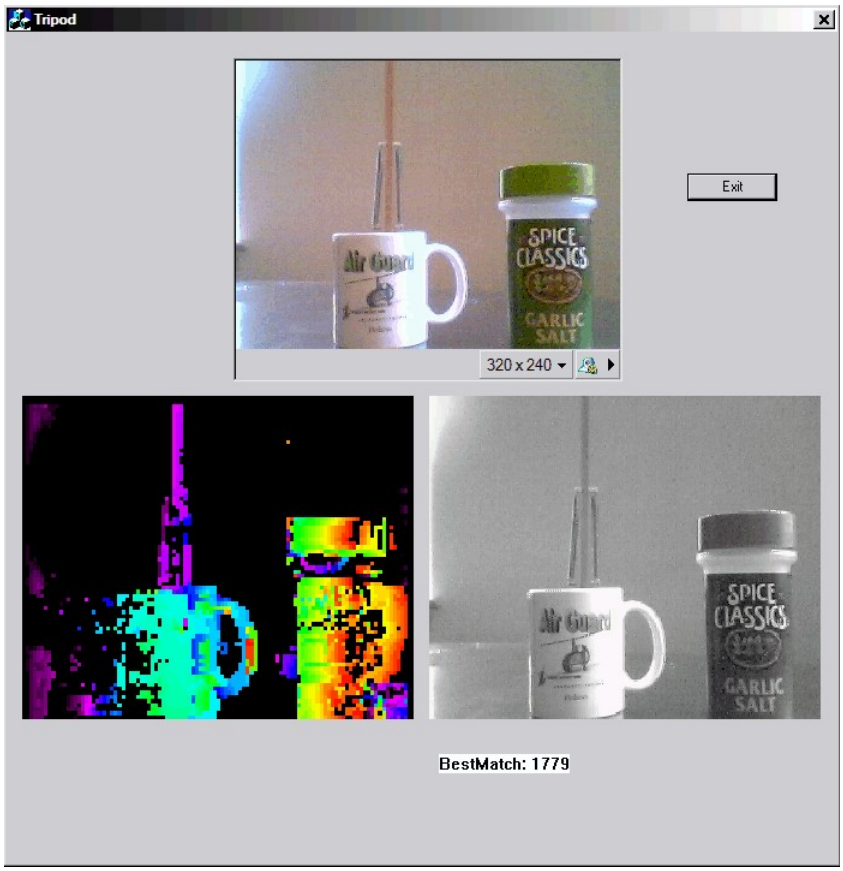
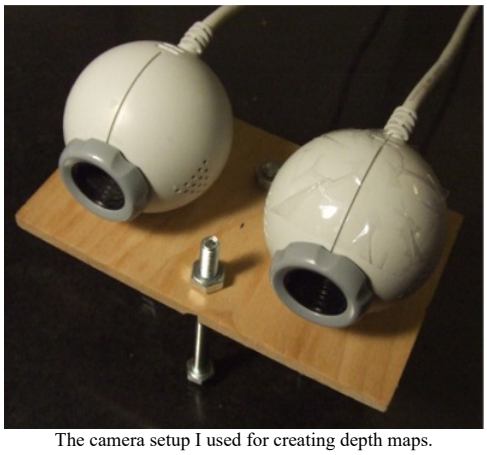
Depth Map Tutorial

Author: Noah Kuntz (2006)
Contact: nk752@drexel.edu

Keywords: Depth Map, Distance, TRIPOD, Binocular Vision



This image shows the result of this depth map generating program. Using two webcams the approximate depth of objects in a scene can be found. The first step in writing this program was adapting TRIPOD to use two cameras simultaneously. The focus of this tutorial is on the image processing nessicary to find depth maps with binocular vision. As far as the interface modifications you should download the entire project and examine it for more details of that change: multicamDistMap.zip In order to recompile the project you must put the standard QCSDK1 folder in the project folder, the same as with normal TRIPOD.



The camera setup I used for creating depth maps.

The image processing in this tutorial can be divided into four steps. First, the one camera's image is stored in an array. Next this is compared to the image of the other camera by taking square regions of pixels and comparing the intensity between the two camera's images. Third the depth of a given pixel region is computed and stored in an array. Finally this array of depths is converted into color for maximum clarity, with hotter colors being closer and cold colors being more distant. Detailed and well lit objects are the best to find the distance to. Large blocks of solid color will produce black or somethings nearly random results. The two cameras used should be identical and any brightness, contrast, and saturation adjustment should be done before using the program.

This tutorial assumes the reader:
(1) Knows how to use TRIPOD
(2) Has a basic knowledge of Visual C++
(3) Has a basic knowledge of image processing techniques

The rest of the tutorial is presented as follows:

Step 1: Store First Camera's Image

Variables are defined and the first camera's image is simply stored in an integer array.

To be compiled with Microsoft Visual C++
Note: download tripodDlg.cpp rather than cutting and pasting from below.

```
*NOTE*  This line must be added to the file tripodDlg.h, the rest goes
in tripodDlg.cpp's "doMyImageProcessing" function
int otherImage[240][320];
*End NOTE*

unsigned int    W, H;                           // Width and Height of current frame [pixels]
W = lpThisBitmapInfoHeader->biWidth;            // biWidth: number of columns
H = lpThisBitmapInfoHeader->biHeight;           // biHeight: number of rows
unsigned int    row, col;                       // Pixel's row and col positions
unsigned long   i;                              // Dummy variable for row-column vector
int rowOffset;                                  // Row offset from the current pixel
int colOffset;                                  // Col offset from the current pixel
int rowTotal = 0;                               // Row position of offset pixel
int colTotal = 0;                               // Col position of offset pixel
int tempIntensity[20][20];                      // Temporary array of pixels to compare one image to the other
int depthMapReal[240][320];                     // The resulting depth map
int row2;                                       // Row of other image
int col2;                                       // Col of other image
int tempDepth;                                  // Temporary storage of depth to convert to color
int depthRed;                                   // Red value for a given depth
int depthGreen;                                 // Green value for a given depth
int depthBlue;                                  // Blue value for a given depth
long intensityMatch;                            // Sum of differences between corresponding pixels in a region
int increment = 3;                              // Number of pixels to shift the current region
int rowStart;                                   // Lowest row to compare to
int bestCol2;                                   // Column with the closest match
int bestMatch;                                  // Intensity match that is the lowest
int offset = 6;                                 // Width of the region to compare

char                            str[80];        // To print message
CDC                             *pDC;           // Device context need to print message

// Clear Depth Map
for (row = 0; row < H; row++) {
        for (col = 0; col < W; col++) {
                depthMapReal[row][col] = 0;
        }
}

switch (side) {
case 4:

        // Store current image in an array to compare to the other camera's view
        for (row = 0; row < H; row++) {
                for (col = 0; col < W; col++) {
                        i = (unsigned long)(row*3*W + col*3);
                        otherImage[row][col] = *(m_destinationBmpRight + i);
                }
        }

        break;
```

Step 2: Compare Regions Between Each Camera

The similiarity between regions of the two images is compared by taking the difference between each region's pixel intensities. The starting point for comparison is the current region's location in the other image. That way there aren't false positives from shifts in the wrong direction. The size of the region (the offset) and how far the region is shifted over (the increment) can be adjusted for the application. The larger the offset the more accurate the resulting depths but speed decreases and detail is reduced. The smaller the increment the more detail is produced but speed is greatly diminished. I don't recommend an increment of less than 2 or an offset of less than 4, and the increment must evenly divide into the offset.

To be compiled with Microsoft Visual C++
Note: download tripodDlg.cpp rather than cutting and pasting from below.

```
        /* Determine depths */
        for (row = 0; row < H-offset; row+=increment) {
```

```
for (col = 0; col < W-offset; col+=increment) {
        // Set up temporary array of current region's pixels from the other camera
        for (rowOffset=0; rowOffset < offset; rowOffset++) {
                for (colOffset=0; colOffset < offset; colOffset++) {
                        i = (unsigned long)((row+rowOffset)*3*W + 3*(col+colOffset));
                        tempIntensity[rowOffset][colOffset] = otherImage[row+rowOffset][col+colOffset];
                }
        }
        // Decide where to start
        if(row < H - 5){
                row2 = row + 5;
        }else{
                row2 = H - row;
        }
        rowStart = row2;
        col2 = 0;
        bestCol2 = 0;
        bestMatch = 9999999;
        // Look for a match
        while (row2 > rowStart - 5) {
                while (col2 < W-offset) {
                        intensityMatch = 0;
                        for (rowOffset=0; rowOffset < offset; rowOffset++) {
                                for (colOffset=0; colOffset < offset; colOffset++) {
                                        i = (unsigned long)((row2+rowOffset)*3*W + 3*(col2+colOffset));
                                        intensityMatch += abs(tempIntensity[rowOffset][colOffset] - *(m_destination
                                }
                        }
                        // If the current match is better than the other matches, save it
                        if(intensityMatch < bestMatch){
                                bestMatch = intensityMatch;
                                bestCol2 = col2;
                        }
                        col2++;
                }
                row2--;
        }

        // Report the closest match per region
        pDC = GetDC();
        sprintf(str, "BestMatch: %d", bestMatch);
        pDC->TextOut(360, 600, str);
        ReleaseDC(pDC);
```

Step 3: Step 3: Calculate Depth

This simply checks to make sure the supposed shift isn't negative or greater than 127 pixels (absurdly close to the cameras), and that the sum of the differences between cameras for all the pixels in the region is less than a certain value which can be adjusted, but a higher value is usually better as this should just cut off really bad matches.

To be compiled with Microsoft Visual C++
Note: download tripodDlg.cpp rather than cutting and pasting from below.

```
// Set depth map value if it is a possible real depth
                if(col - bestCol2 > 0 && col - bestCol2 < 127 && bestMatch < 5000){
                        depthMapReal[row / increment][col / increment] = 2 * abs(col - bestCol2);
                }
        }
}
```

Step 4: Calculate Depth

Colors from close to far

Now the depth value (between 0 and 255) is converted to a color to better show how far away the object is. Red is the closest down in color temperature to black as the furthest. The actual depths these colors represent is dependent upon your cameras and their distance from each other, I suggest calibrating based on objects placed at known distances.

To be compiled with Microsoft Visual C++
Note: download tripodDlg.cpp rather than cutting and pasting from below.

```
// Convert the depth map to color and draw it on the left viewport
for (row = 0; row < H; row++) {
        for (col = 0; col < W; col++) {
                i = (unsigned long)(row*3*W + col*3);
                tempDepth = depthMapReal[row / increment][col / increment];
                if(tempDepth < 43){
                        depthRed = tempDepth * 6;
                        depthGreen = 0;
                        depthBlue = tempDepth * 6;
```

```
                        }
                        if(tempDepth > 42 && tempDepth < 85){
                                depthRed = 255 - (tempDepth - 43) * 6;
                                depthGreen = 0;
                                depthBlue = 255;
                        }
                        if(tempDepth > 84 && tempDepth < 128){
                                depthRed = 0;
                                depthGreen = (tempDepth - 85) * 6;
                                depthBlue = 255;
                        }
                        if(tempDepth > 127 && tempDepth < 169){
                                depthRed = 0;
                                depthGreen = 255;
                                depthBlue = 255 - (tempDepth - 128) * 6;
                        }
                        if(tempDepth > 168 && tempDepth < 212){
                                depthRed = (tempDepth - 169) * 6;
                                depthGreen = 255;
                                depthBlue = 0;
                        }
                        if(tempDepth > 211 && tempDepth < 254){
                                depthRed = 255;
                                depthGreen = 255 - (tempDepth - 212) * 6;
                                depthBlue = 0;
                        }
                        if(tempDepth > 253){
                                depthRed = 255;
                                depthGreen = 0;
                                depthBlue = 0;
                        }

                        *(m_destinationBmp + i) = depthBlue;
                        *(m_destinationBmp + i + 1) = depthGreen;
                        *(m_destinationBmp + i + 2) = depthRed;
                }
        }

        break;
}
```

Final Words

This tutorial's objective was to show how simple depth maps can be found with a modified version of TRIPOD. With this information a robot could do some basic collision avoidance and path planning. Note though that the lighting must be fairly even and bright and that the objects should be as detailed as possible for good results.

Click here to email me.
Click here to return to my Tutorials page.