CSCI 4220U

Final Project Report

By Cody Macedo

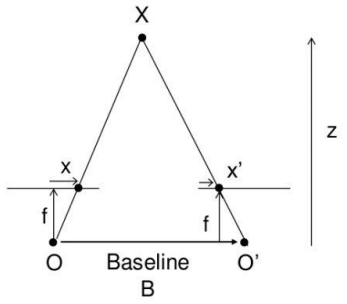
ID 100486136

The objective of my final project was to create a depth map mirror that took the depth of the objects in the scene, represented them as contours and displayed them on a screen. The original thought was to have a display external to the computer that would act as a mirror. One idea was to have an array of motors that would flip panels if the object in the scene met a certain depth threshold. Upon further research, the pure cost and amount of work required to make such a setup functional would be out of scope for this project. Another idea I had was to take an LED matrix and have that be the "mirror", having the LEDs shine varying luminosities of one colour depending on the depth of the image at each block of the image representing each pixel of the display. This was far more reasonable and is what I set my goal as for the completion of this project.

The first step to making this project work was to figure out how I would develop a depth map with opencv that was decent enough to be shown on the LED display. I originally thought to use only one camera, but upon researching the subject realized that this approach would be too cumbersome, with either having to use markers placed in the background/foreground of the scene to act as depth comparison markers. Another approach to the single image depth map was to identify objects in the scene upon which I actually knew the size of. For example, if there was a tree in the scene and I knew that the tree was 10 m tall, I could compare objects to the tree to determine their relative depth. I also learned that the topic of single image depth maps is currently a hot topic that is being studied in the field of computer vision, and the approaches proposed in scholarly papers would be quite difficult to implement in the time allotted to completing this project. In the end, I chose to go with stereo imaging as a way to allow me to determine a depth map to be used with the mirror.

The materials I needed for this project consisted of two web cams to take the stereo images, a printed out chessboard used to calibrate the cameras, an LED matrix and a microcontroller to receive signals from the computer and modify the LED matrix with the correct values.

The main idea about using stereo imaging to find the depth of pixels in the resulting images is that using both images produced by the stereo camera setup, you can calculate the disparity of a pixel and thus its depth. The following image is an illustration that shows the relationship between the cameras, O and O', the points being looked at, x and x' , and the proposed pixel, X, that is the same point as both x and x'.



disparity = x – x' = (B*f) / z

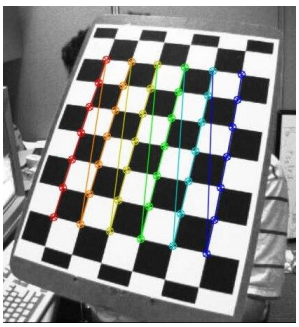
The value for disparity, as shown by the image above, is either the distance between points x and x' or the distance between the cameras times the camera focus length all divided by the proposes distance to the real point. With this equation, one can calculate a value for the disparity and eventually z, which will be used to determine the depth of objects in the scene.

Capturing the images is pretty simple, you just need two cameras. In my case, I used two USB web cams. The problems that I ran into was get the resolution and perspective of both web cams to match up. I also had to make sure the webcams remained at the same height so that the left and right images were only shifted along the horizontal, not the vertical. Once the web cams were fixed, I could take video with each one simultaneously and store each frame of the video to be used later to calibrate the system. One thing I did do, which I will explain later, is to crop the image from its native resolution of 720x1280 to 720x960. This change helped late to reduce the effects of radial distortion.

The next step to make the depth map work was to calibrate the cameras so that they would work in sync and rectify the two images into something that could be used for finding the depth map. As mentioned earlier, radial distortion from the cameras causes lines that are straight in real life to appear curved in the images. To reduce this effect, I cropped the input images to get rid of the extremes on the sides. To rectify the images, I used a function in opencv called calibrateCamera. This function solved this equation:

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Returning the camera matrix (left matrix), the rotational and translational vectors (right matrix) and the distortion coefficients which are the coefficients that were found to solve the second derivative of the linear equations created from the system above. Before we can call this function though, we need to calculate the points of interest/corners in each image and match them up to figure out how much distance the images are translated from one another. An easy way to do this is to use the function findChessboardCorners and use an actual chessboard to fin corners that are found in both images. An example of the corners on a chessboard for one camera is as follows:



This process would be done for all the images captured from the left and right camera. The set of points that match for both the left and right images at any moment in the captured video are used to calibrate the camera and rectify the left and right images. Once the calibrateCamera function is called, a series of other functions used to undistort, rectify and remap the image are called. The output images are undistorted and calibrated such that a depth map can be calculated with them.

Now that the cameras and the images are calibrated, I was able to call the function StereoBM_create to calculate the depth map in real time while the cameras record. The depth map uses model generated from the step before when the everything was calibrated. The resulting depth map determines the depth of the objects in the image and colours it a shade of grey depending on its depth. The lighter the grey the closer the item.

The results of this process didn't quite pan out perfectly for me as the depth map that I generated was quite scattered and messy. This could be because the calibration was not successful or there was too much noise collected from the depth map calculation. Either way, in the future I plan to fix this to get a resulting image that has grey contours representing objects at different depths. Once this is completed, I would like to have it streamed to an LED matrix and have it act as a sort of mirror to complete my original goal of this project.