Lab 4 : computer vision

Stereo Vision

May 15, 2019

Team

Dahlia Chehata 27

Fares Mehanna 52

Hesham Elsawaf 81

Overview

Stereo vision with block matching using 2 metrics : SDD and SAD

and with dynamic programming

Block matching

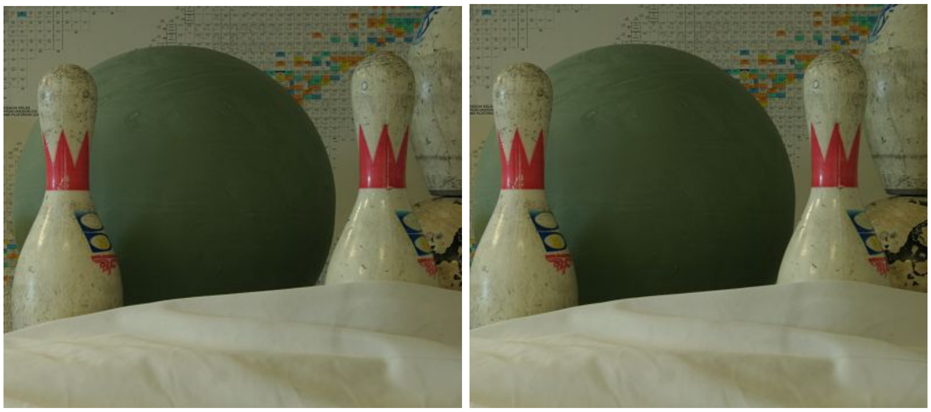
Code

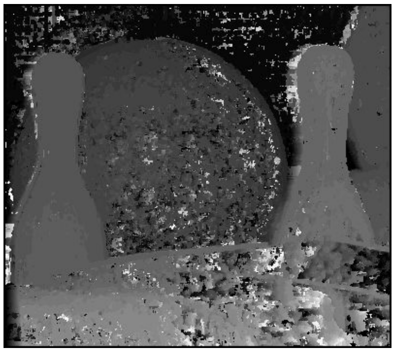
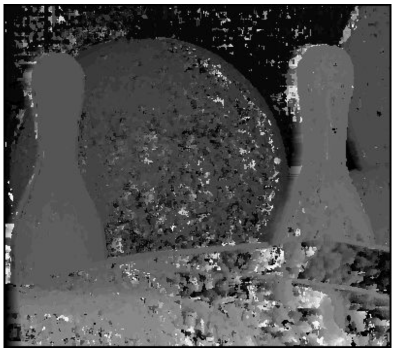
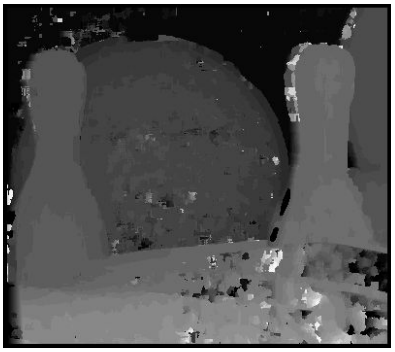
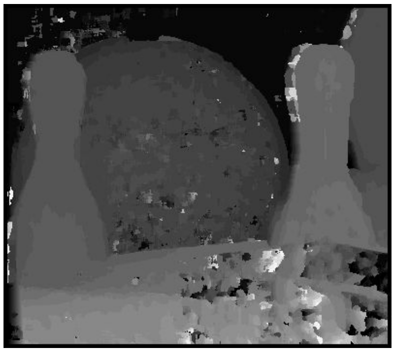
```
def stereo_match(left_img, right_img, kernel, max_offset,output_file):
   left_img = Image.open(left_img).convert('L')
   left = np.asarray(left_img)
   right_img = Image.open(right_img).convert('L')
   right = np.asarray(right_img)
   w, h = left_img.size  # assume that both images are same size

   # Depth (or disparity) map
   depth = np.zeros((w, h), np.uint8)
   depth.shape = h, w

   kernel_half = int(kernel / 2)
   offset_adjust = 255 / max_offset
   # this is used to map depth map output to 0-255 range
   for y in range(kernel_half, h - kernel_half):
       print(".", end="", flush=True)
       for x in range(kernel_half, w - kernel_half):
           best_offset = 0
           prev_ssd = 65534
           for offset in range(max_offset):
               ssd = 0
               ssd_temp = 0
              for v in range(-kernel_half, kernel_half):
                   for u in range(-kernel_half, kernel_half):
                      ssd_temp = int(left[y + v, x + u]) - int(right[y + v, (x + u) - offset])
                      if “SAD”: ssd += abs(ssd_temp)
                      elif “SSD” : ssd += ssd_temp * ssd_temp
               if ssd < prev_ssd:
                   prev_ssd = ssd
                   best_offset = offset
           result = best_offset * offset_adjust
           depth[y, x] = result
   # Convert to PIL and save it
   Image.fromarray(depth).save(output_file)
```

Results



	SSD	SAD
window size = 1 x 1		
window size = 5 x 5		
window size = 9 x 9		

Conclusion

- block matching takes a two photos, a left and right image of a subject taken from slightly different angles, and outputs a depth (disparity) map. Each pixel in the depth map indicates how far away it is from the camera - the darker it is, the further away it has been calculated to be, and vice versa.
- SSD is a better metric than SAD

Dynamic Programming

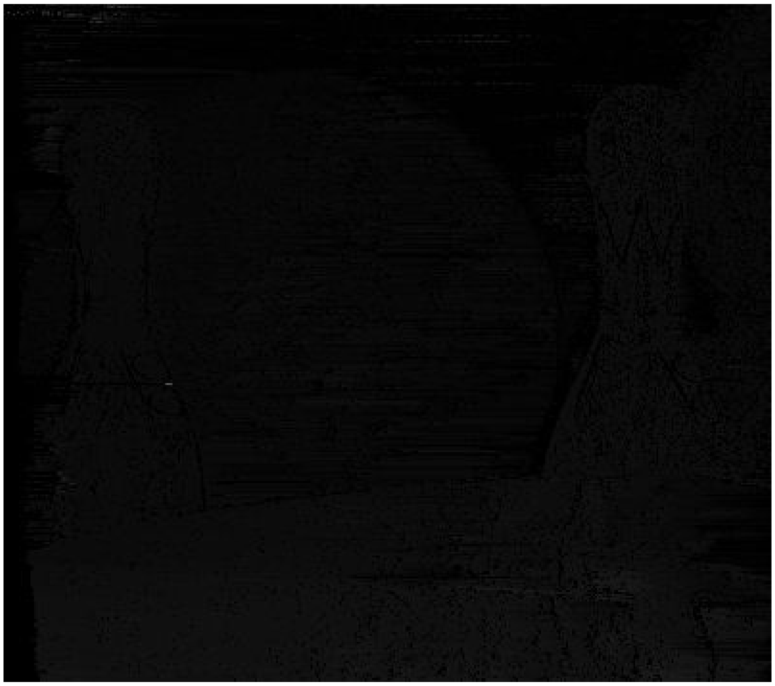
code

```
def stereoMatching(leftImg, rightImg):
   rows = leftImg.shape[0]
   cols = leftImg.shape[1]
   # Matrices to store disparities : left and right
   leftDisp = np.zeros((rows, cols))
   rightDisp = np.zeros((rows, cols))
   occlusion = 1
   # Pick a row in the image to be matched
   for c in range(0, rows):
       # Cost matrix
       colMat = np.zeros((cols, cols))
       # Disparity path matrix
       dispMat = [[None]*cols for i in range(cols)]
       # Initialize the cost matrix
       for i in range(0, cols):
           colMat[i][0] = i * occlusion
           colMat[0][i] = i * occlusion
       for k in range(0, cols):
           for j in range(0, cols):
               if leftImg[c][k] > rightImg[c][j]:
                   match_cost = leftImg[c][k] - rightImg[c][j]
               else:
                   match_cost = rightImg[c][j] - leftImg[c][k]
               dij=(match_cost/4 * match_cost)
               # Finding minimum cost
               min1 = colMat[k - 1][j - 1] + match_cost
               min2 = colMat[k - 1][j] + occlusion
               min3 = colMat[k][j - 1] + occlusion
               colMat[k][j] = cmin = min(min1, min2, min3)
               # Marking the path
               if (min1 == cmin):
                   dispMat[k][j] = "1"
```

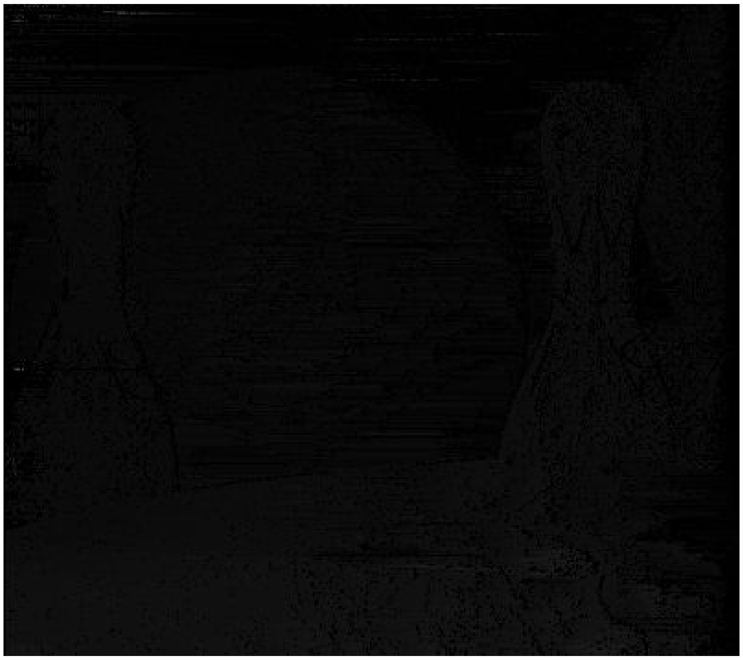
```
            if (min2 == cmin):
                dispMat[k][j] = '2'
            if (min3 == cmin):
                dispMat[k][j] = '3'
          i = cols - 1
    j = cols - 1
    while (i != 0) and (j != 0):
        if (dispMat[i][j] == 1):
            leftDisp[c][i] = np.absolute(i - j)
            rightDisp[c][j] = np.absolute(j - i)
            i = i - 1
            j = j - 1
        elif (dispMat[i][j] == 2):
            leftDisp[c][i] = 0
            i = i - 1
        elif (dispMat[i][j] == 3):
            rightDisp[c][j] = 0
            j = j - 1
cv2.imwrite("Left_Disparity_"+str(occlusion)+".png",leftDisp)
cv2.imwrite("Right_Disparity_"+str(occlusion)+".png",rightDisp)
```

Results

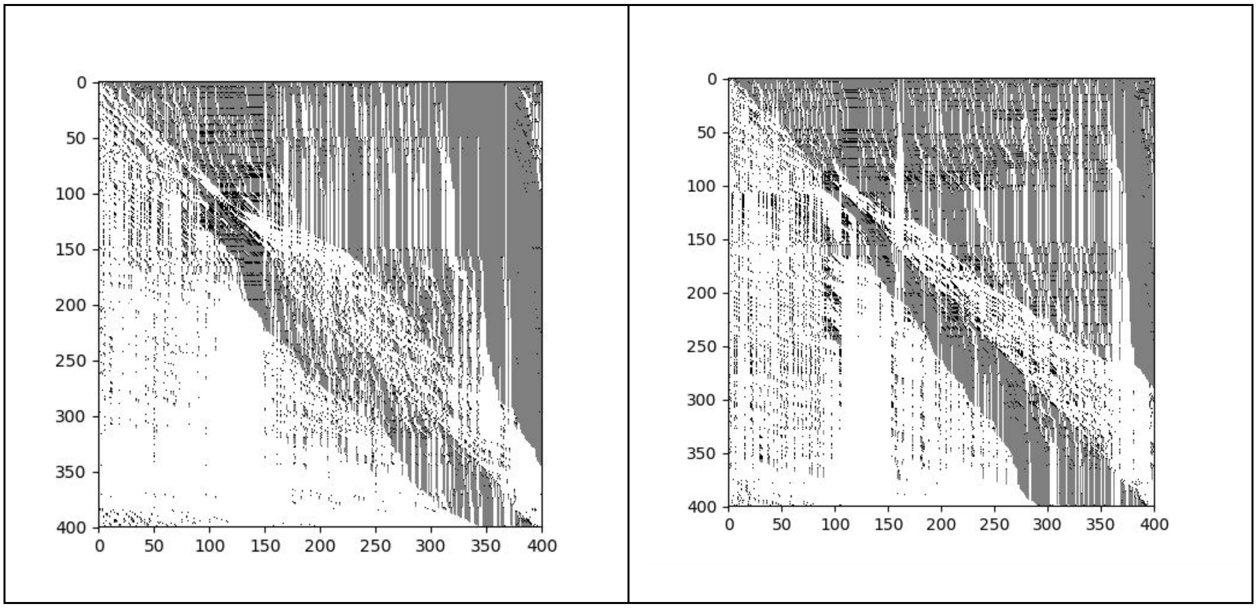
left disparity

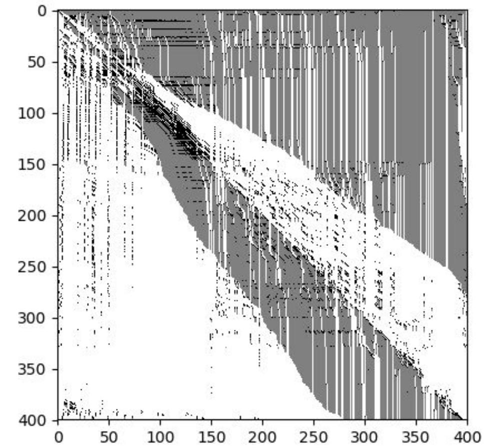


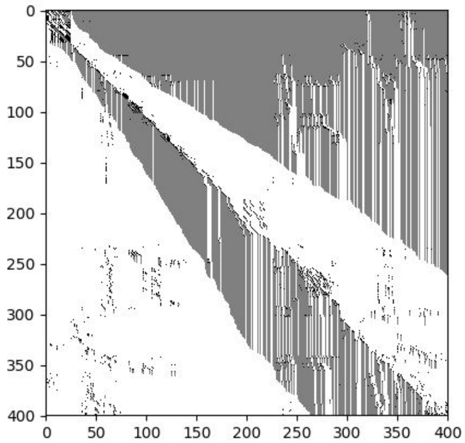
right disparity



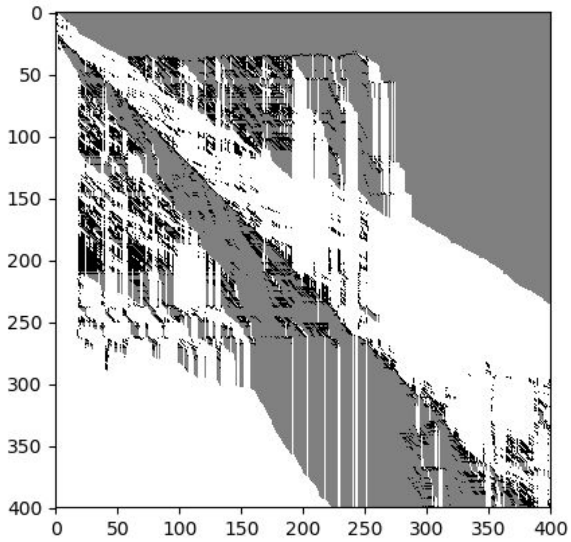
Alignment plot for a single scan line



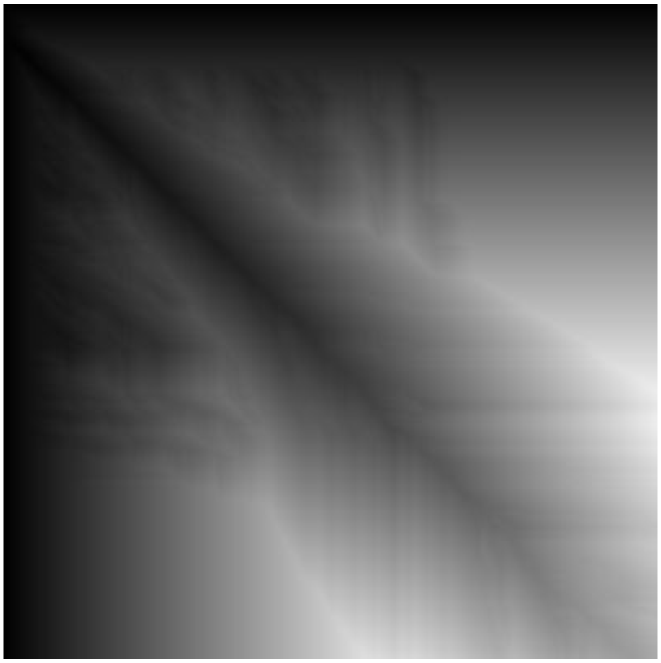




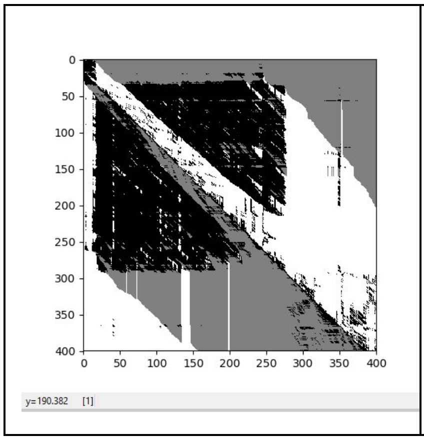
Alignment plot for whole matrix



cost matrix



occlusion = 20

 0 50 100 150 200 250 300 350 400 0 50 100 150 200 250 300 350 400 y=190.382 [1]		