Aidan Fischer 3/1/2022

CS 532 HW 2 I pledge my honor that I have abided by the Stevens Honor System

Code

main.py – Executes the functions that solve the homework

```
def main():
    GroundTruth = Image.get_image("../teddy/disp2.pgm")
    L = Image.get_image("../teddy/teddyL.pgm")
    R = Image.get_image("../teddy/teddyR.pgm")
    RankTL = RankTransform.rank_transform(L)
    RankTR = RankTransform.rank_transform(R)
    DispMap3, PKRN3 = SAD_Stereo.SAD(RankTL, RankTR, 3)
    DispMap15, PKRN15 = SAD_Stereo.SAD(RankTL, RankTR, 15)
    Image.save_image(SAD_Stereo.normalize_SAD(DispMap3),
"../Output/P1/DispMap3.pgm")
    print(f"Error 3x3 Window: {GroundTruthError.GroundTruthErr(GroundTruth,
DispMap3)}")
    print(f"Error 15x15 Window: {GroundTruthError.GroundTruthErr(GroundTruth,
DispMap15)}")
    PKRN3Err, PKRNKept = GroundTruthError.GroundTruthErrPKRN(GroundTruth,
DispMap3, PKRN3)
    print(f"PKRN Error 3x3 Window: {PKRN3Err}, Kept Pixels = {PKRNKept}")
if __name__ == "__main__":
    main()
```

Image.py – wrapper around PIL and numpy that I made for the Computer Vision course homework that is also useful for this course

```
def get_image(path, gray=False):
    image = Image.open(path)
    if gray:
        image = ImageOps.grayscale(image)
    return np.array(image)


def save_image(image_array, name):
    image = Image.fromarray(image_array)
    image.convert("RGB").save(name)
    return


def copy_image(image):
    return np.copy(image)
```

```
def to_array(PILimage):
    return np.array(PILimage)
```

RankTransform.py – Implements the rank transform algorithm.

```
def rank_transform(image):
    rank_t = np.zeros(image.shape)
    for r in trange(2,image.shape[0]-2, desc="Rank transform"):
        for c in range(2,image.shape[1]-2):
            # Gotta love numpy's syntax sugar.
            # This does an element-wise compare with the center of the window,
            # then since Python treats True as 1, and False as 0, the result is
            # as simple as summing the boolean values in the new array.
            rank_t[r, c] = (image[r-2:r+3,c-2:c+3] < image[r, c]).sum()
    return rank_t
```

SAD_Stereo.py – Implements the simple stereo algorithm using the SAD Disparity measure. Makes assumptions based on problem definition (disparity between 0 and 63) and course assumptions (slide that states to taking rectification for granted – assumed horizontal epipolar lines)

```
import numpy as np
from tqdm import trange

def SAD(RankTransform1, RankTransform2, windowSize):
    SAD_dispmap = np.zeros(RankTransform1.shape)
    PKRN = np.zeros(RankTransform1.shape)
    offset = int((windowSize-1)/2)
    for r in trange(offset, RankTransform1.shape[0]-offset, desc="Computing SAD
on rank transforms"):
        for c in range(offset, RankTransform1.shape[1]-offset):
            min_SAD = np.nan
            min_SAD_2 = np.nan
            disparity = 0
            # Uses the problem assumption that disparity is from 0 to 63
            for c2 in range(max(offset, c - 63), c+1):
                SAD = abs(RankTransform1[r-offset:r+offset+1, c-
offset:c+offset+1] - RankTransform2[r-offset:r+offset+1, c2-
offset:c2+offset+1]).sum()
                disparity = c - c2 if np.isnan(min_SAD) or SAD < min_SAD else
disparity
                min_SAD_2 = min_SAD if np.isnan(min_SAD) or SAD < min_SAD else
min_SAD_2
                min_SAD = SAD if np.isnan(min_SAD) or SAD < min_SAD else min_SAD
            SAD_dispmap[r, c] = disparity
```

```
            PKRN[r, c] = min_SAD_2/min_SAD if min_SAD > 0 and not
np.isnan(min_SAD_2) else np.inf # Will be closer to 1 the closer 2nd smallest is
to smallest.

    return SAD_dispmap, PKRN


#Normalizes to range of 0-255
#To be divided by 4 when processed.

def normalize_SAD(SAD):
    norm = 4
    return (SAD*norm).astype(np.uint8)
```

GroundTruthError.py – Implements the error measure, including base error and PKRN measure error.

```
def GroundTruthErr(GroundTruth, DispMap):
    GroundTruthDiv = np.round((GroundTruth / 4))
    Diff = np.abs(GroundTruthDiv-DispMap)
    return (Diff > 1).sum()/(GroundTruth.shape[0]*GroundTruth.shape[1])

def GroundTruthErrPKRN(GroundTruth, DispMap, PKRN):
    GroundTruthDiv = np.round((GroundTruth / 4))
    Median = np.median(PKRN)
    Diff = np.abs(GroundTruthDiv-DispMap)
    Keep = (PKRN > Median)
    Kept = np.count_nonzero(Keep)
    return np.logical_and(Diff > 1, Keep).sum() / Kept, Kept
```

Results

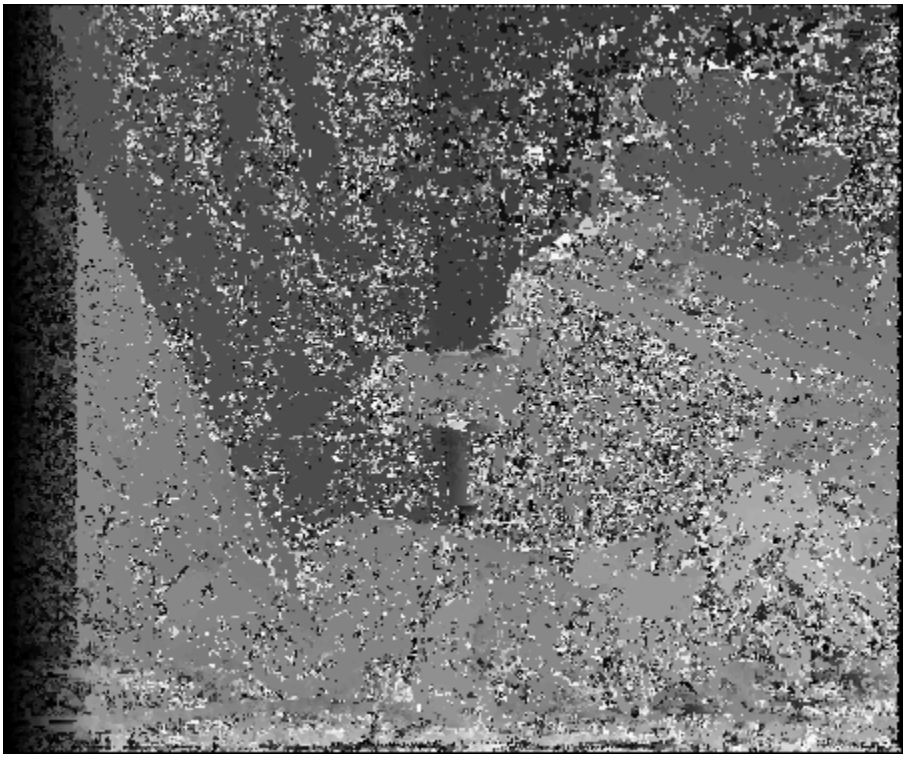
Error 3x3 Window: 0.47930666666666666

Error 15x15 Window: 0.25346962962962966

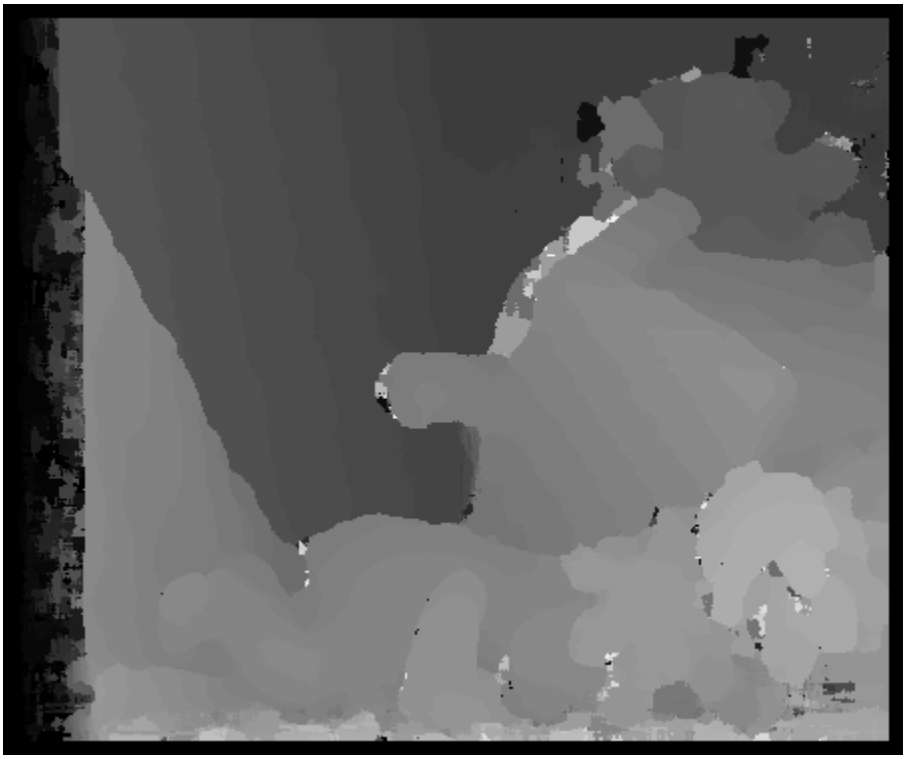
PKRN Error 3x3 Window: 0.34565819011081683, Kept Pixels = 84193

Disparity Maps

3x3:



15x15



Extra Notes:

As stated earlier, my implementation of the stereo algorithm makes assumptions based on the provided assumptions for the course and homework. The homework states that disparity is between 0 and 63, and because we have known left and right images, it also assumes that the matching disparity is strictly in the movement direction. This significantly improved the quality of my disparity maps. I also assumed that the epipolar lines were horizontal because Notes 3 stated to take rectification for granted.

Tqdm is a python progress bar library. I just use it to see the progress the algorithm is making. I usually use it if a program will take a long time to run to make sure it's doing stuff.

If PKRN is impossible to calculate (the smallest cost is 0) then I set the measure to infinity.