Computer vision - Problem set 5

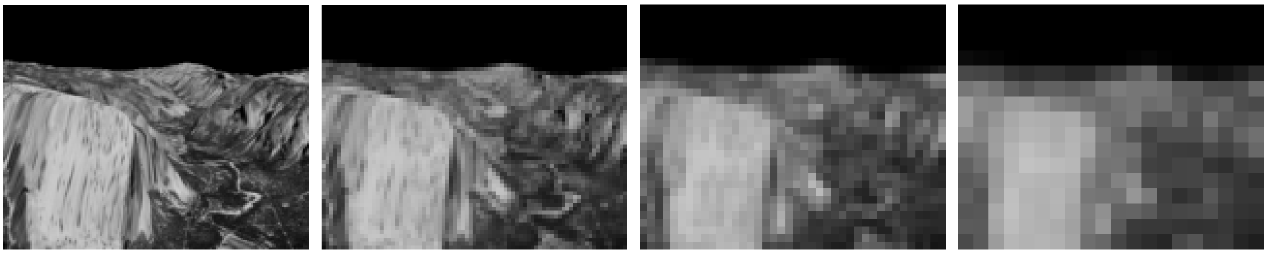
Nicolas Six

April 5, 2018

Contents

Question 1 Gaussian and Laplacian Pyramids

1.1 Gaussian pyramids

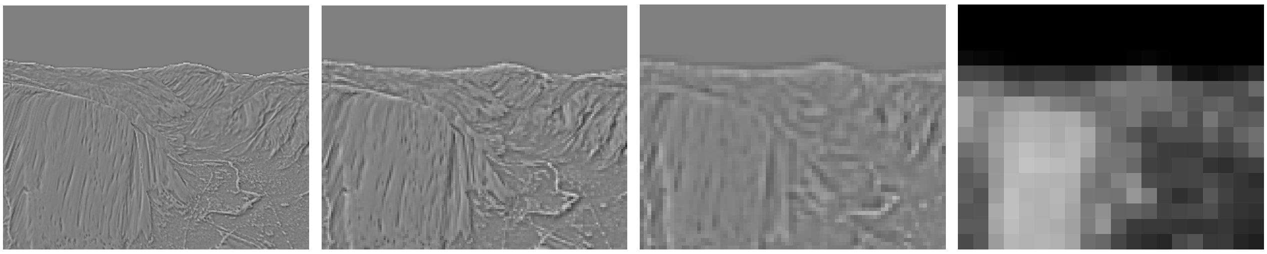


(a) Level 1 (b) level 2 (c) level 3 (d) level 4

Figure 1: 4 levels Gaussian pyramid

```python
def gaussian_reduce(im: np.ndarray):
    im = cv2.GaussianBlur(im, (3, 3), 0)
    im = cv2.resize(im, (0, 0), fx=0.5, fy=0.5,
       interpolation=cv2.INTER_NEAREST)
    return im
```

1.2 Laplacian pyramid



(a) Level 1 (b) level 2 (c) level 3 (d) level 4 (Gaussian)

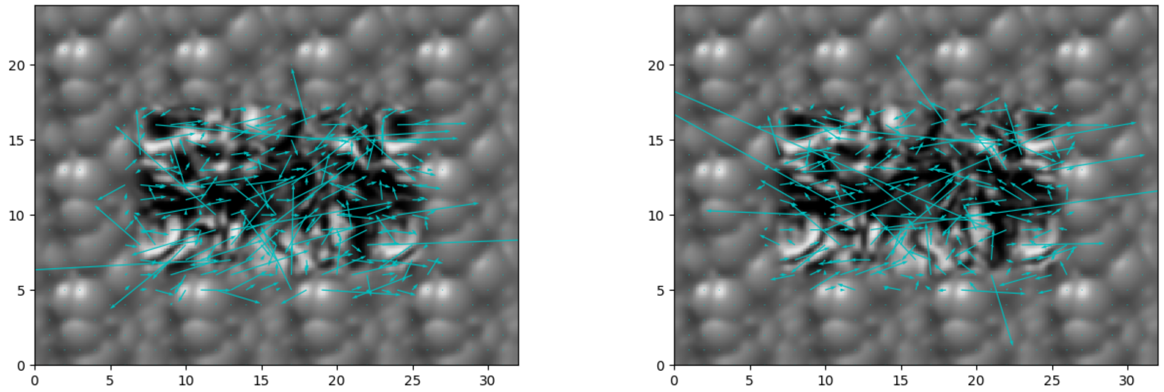
Figure 2: 4 levels Laplacian pyramid

```python
def gaussian_expend(im: np.ndarray):
    im = cv2.resize(im, (0, 0), fx=2.0, fy=2.0,
        interpolation=cv2.INTER_NEAREST)
    im = cv2.GaussianBlur(im, (3, 3), 0)
    return im


def laplacian_reduce(im: np.ndarray):
    r = gaussian_reduce(im.copy())
    e = gaussian_expend(r.copy())
    im = cv2.resize(im, (e.shape[1], e.shape[0]))
    return r, im.astype(np.int) - e.astype(np.int)
```

Question 2 Lucas Kanade optic flow

2.1 On small movement

```python

def lucas_kanade_optic_flow(im1, im2, win_size, blur_size=None):
    if blur_size is not None:
        im1 = cv2.GaussianBlur(im1, (blur_size, blur_size), 1,
            borderType=cv2.BORDER_REPLICATE)
        im2 = cv2.GaussianBlur(im2, (blur_size, blur_size), 1,
            borderType=cv2.BORDER_REPLICATE)

    i_x = compute_gradient(im1, 'X')
    i_y = compute_gradient(im1, 'Y')
    i_t = im1.astype(np.int32) - im2.astype(np.int32)

    win = cv2.getGaussianKernel(win_size, 1)

    i_xx_sum = convolve2d(np.power(i_x, 2), win, 'same', boundary='symm')
    i_yy_sum = convolve2d(np.power(i_y, 2), win, 'same', boundary='symm')
    i_xy_sum = convolve2d(i_x * i_y, win, 'same', boundary='symm')
    i_xt_sum = convolve2d(i_x * i_t, win, 'same', boundary='symm')
    i_yt_sum = convolve2d(i_y * i_t, win, 'same', boundary='symm')

```



(a) ShiftR2

(b) ShiftR5U5

Figure 3: Lucas Kanade displacement arrows draw over original image, computed after adding a 51px Gaussian blur

```
    det = np.power(i_xx_sum * i_yy_sum - i_xy_sum**2, -1)

    inds = np.where(np.isinf(det))
    det[inds] = 0.0

    vx = (-i_yy_sum * i_xt_sum + i_xy_sum * i_yt_sum) * det
    vy = ( i_xy_sum * i_xt_sum - i_xx_sum * i_yt_sum) * det
```

2.2 On larger movement

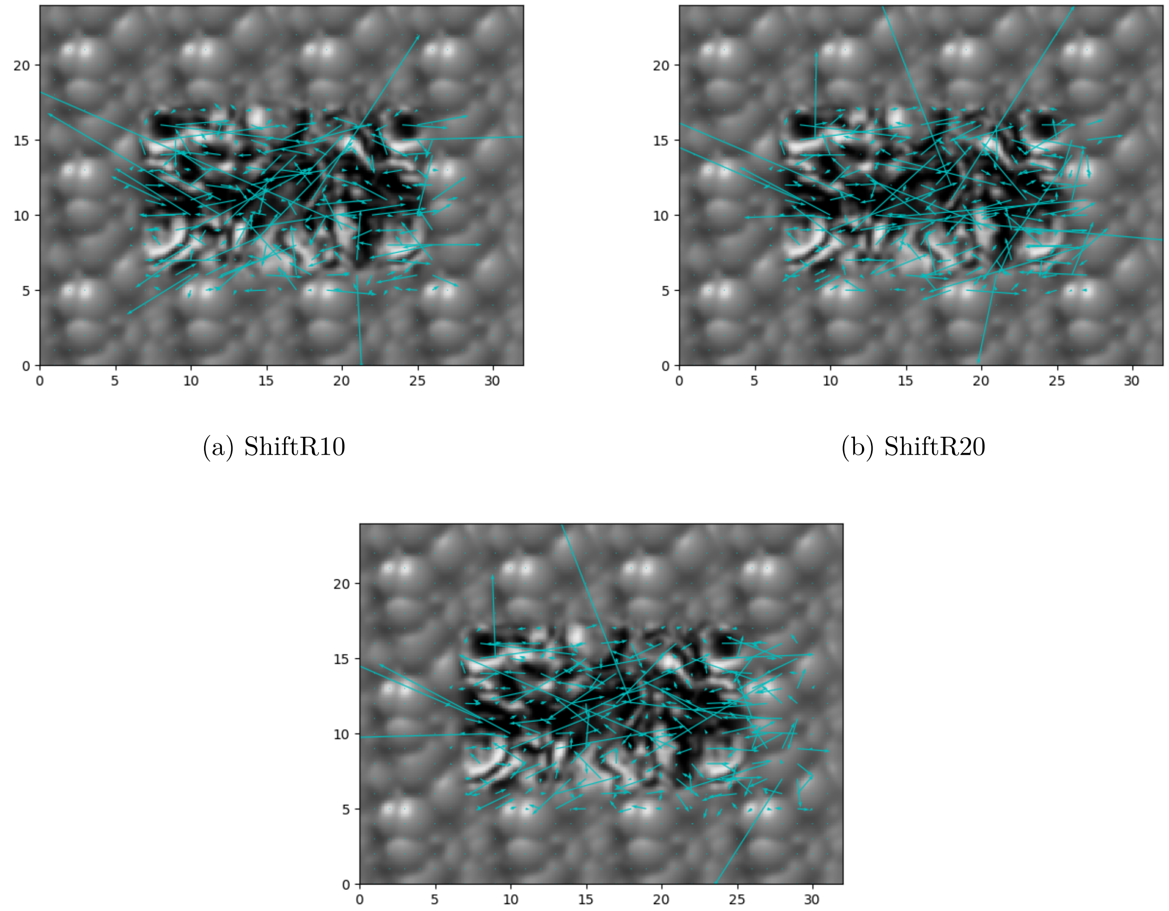
We verify, here, that Lucas Kanade does not work well on big movement. If the general movement is still globally in the right direction, the bigger is the shift, the bigger is the noise.

2.3 Gaussian pyramid

```
def wrap(im: np.ndarray, vx: np.ndarray, vy: np.ndarray):
    map1 = vx + np.array(range(vx.shape[0])).reshape((vx.shape[0], 1))
    map2 = vy + np.array(range(vy.shape[1])).reshape((1, vy.shape[1]))
    im = im[:vx.shape[0], :vx.shape[1]]
    ret = cv2.remap(im, map2.astype(np.float32), map1.astype(np.float32),
       cv2.INTER_NEAREST)
    return ret
```

On Figure 6, you can see that the wrapped image does not allow to recover the destination image. We were expecting better results, but did not fund how to get them.

Question 3 Hierarchical LK optic flow



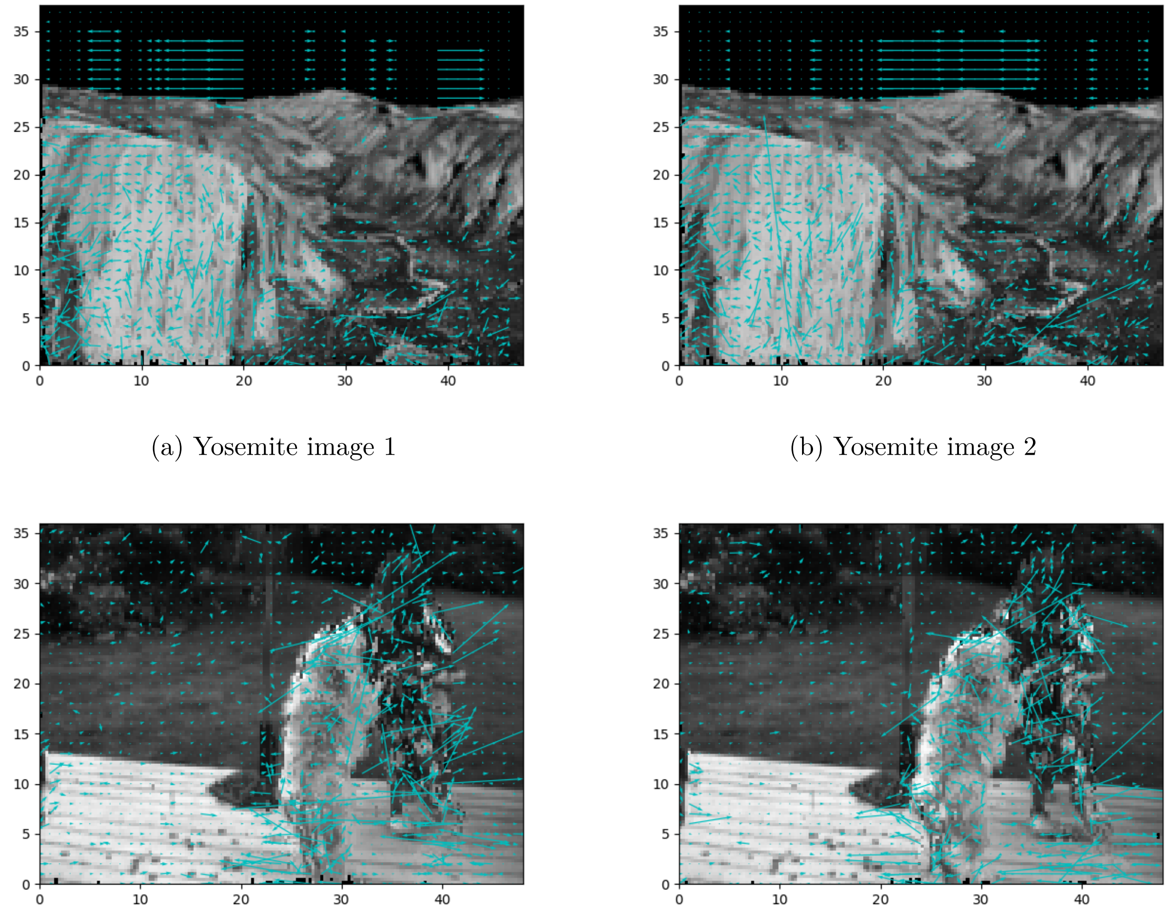
(a) ShiftR10

(b) ShiftR20

(c) ShiftR40

Figure 4: Lucas Kanade displacement arrows draw over original image, computed after adding a 51px Gaussian blur

```python
def hierarchical_lk(im1, im2, level, win_size=5, blur_size=None):
    sr = []
    imgs = (im1, im2)
    sr.append(imgs)
    for lvl in range(level):
        imgs = gaussian_reduce(imgs[0]), gaussian_reduce(imgs[1])
        sr.append(imgs)

    vx = np.zeros(sr[-1][0].shape, dtype=np.float64)
    vy = np.zeros(sr[-1][0].shape, dtype=np.float64)
    for lvl in range(level, -1, -1):
        w, h = sr[lvl][0].shape
        vx = vx[:w, :h]
        vy = vy[:w, :h]
        im_l0 = wrap(sr[lvl][0], vx, vy)
        new_vx, new_vy = lucas_kanade_optic_flow(im_l0, sr[lvl][1],
            win_size, blur_size)
```



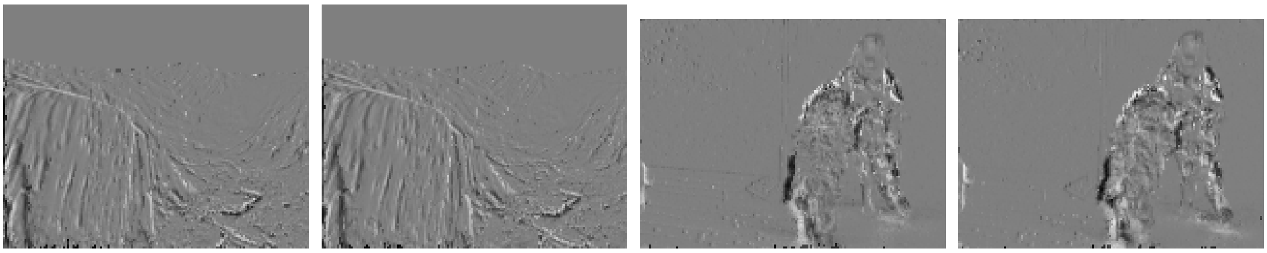
(a) Yosemite image 1

(b) Yosemite image 2

(c) Dog 0

(d) Dog 1

Figure 5: Lucas Kanade displacement arrows draw over original image



(a) Yosemite image 1

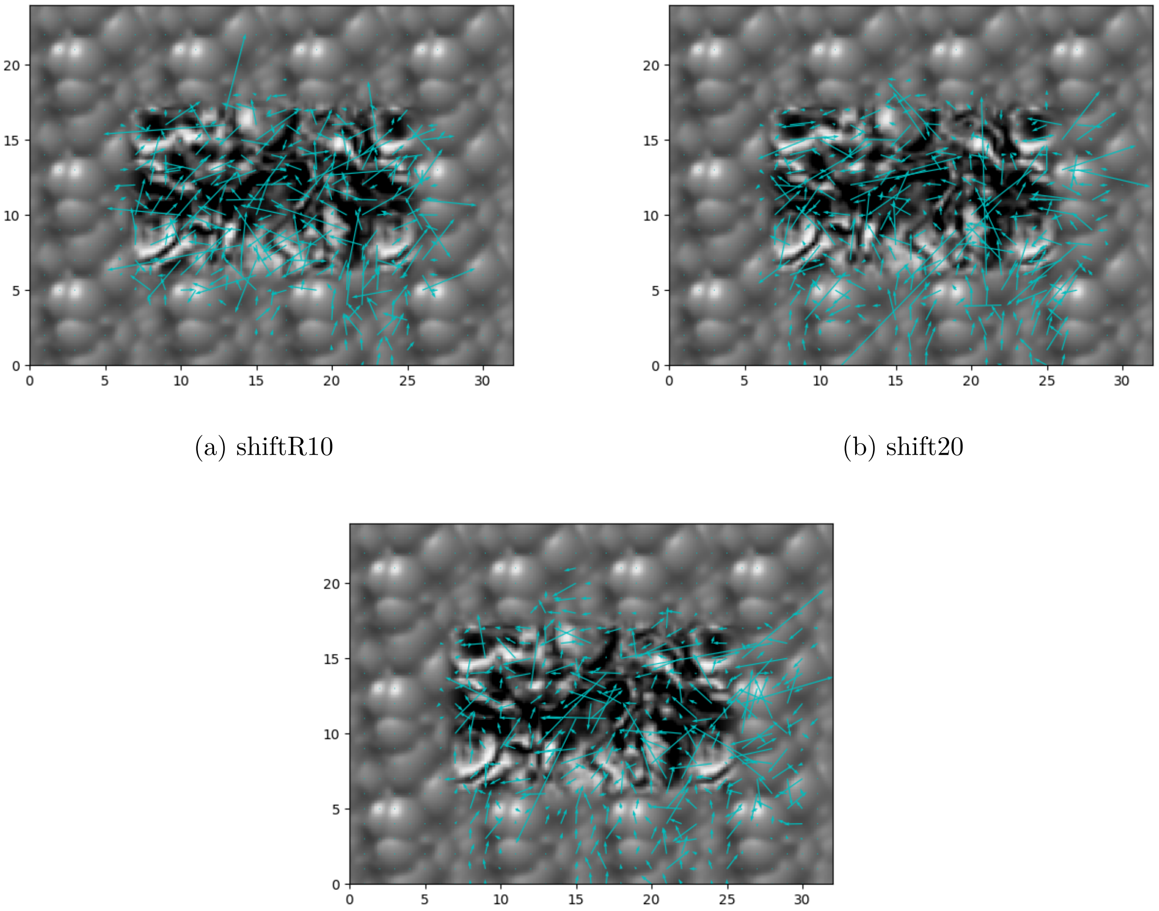
(b) Yosemite image 2

(c) Dog 0

(d) Dog 1

Figure 6: Difference between the wrapped and destination image

```
        vx += new_vx
        vy += new_vy
        if lvl != 0:
            vx, vy = gaussian_expend(vx), gaussian_expend(vy)

    return vx, vy
```

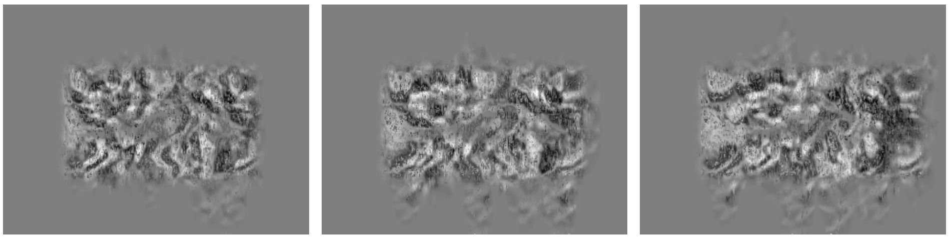


(a) shiftR10

(b) shift20

(c) shiftR40

Figure 7: Lucas Kanade displacement arrows draw over original image



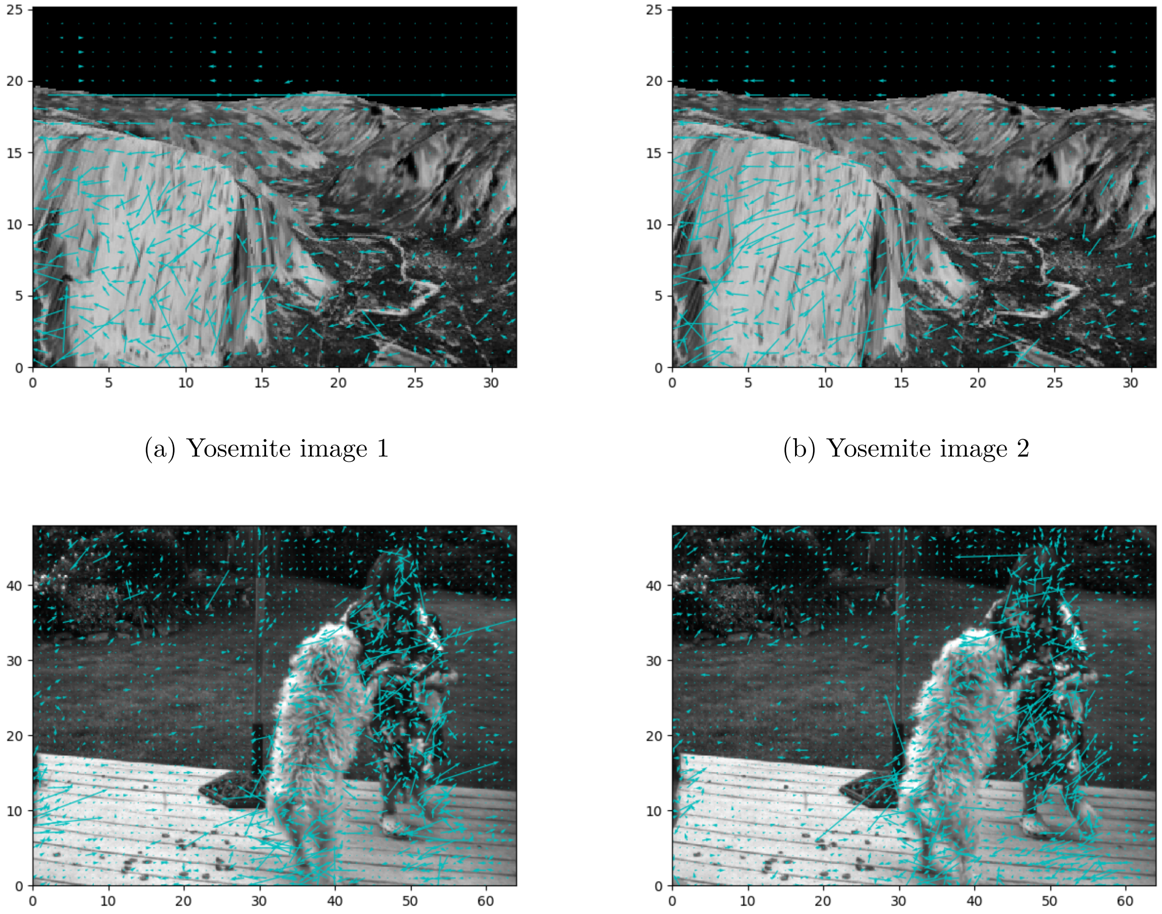
(a) shiftR10

(b) shift20

(c) shiftR40

Figure 8: Difference between the wrapped and destination image

Question 4 The Juggle Sequence



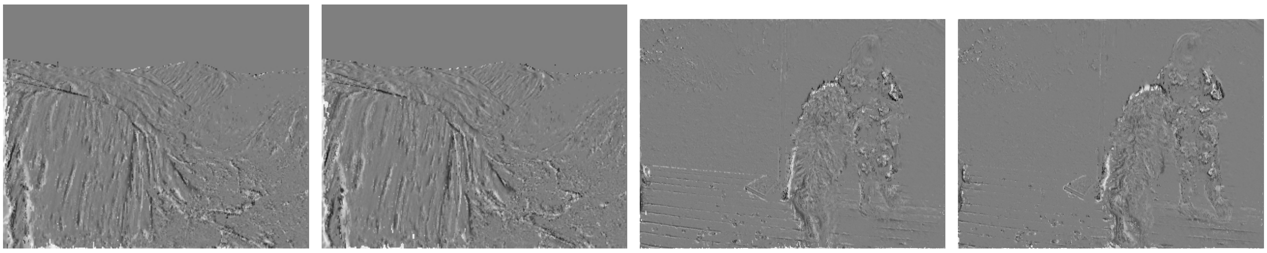
(a) Yosemite image 1

(b) Yosemite image 2

(c) Dog 0

(d) Dog 1

Figure 9: Lucas Kanade displacement arrows draw over original image



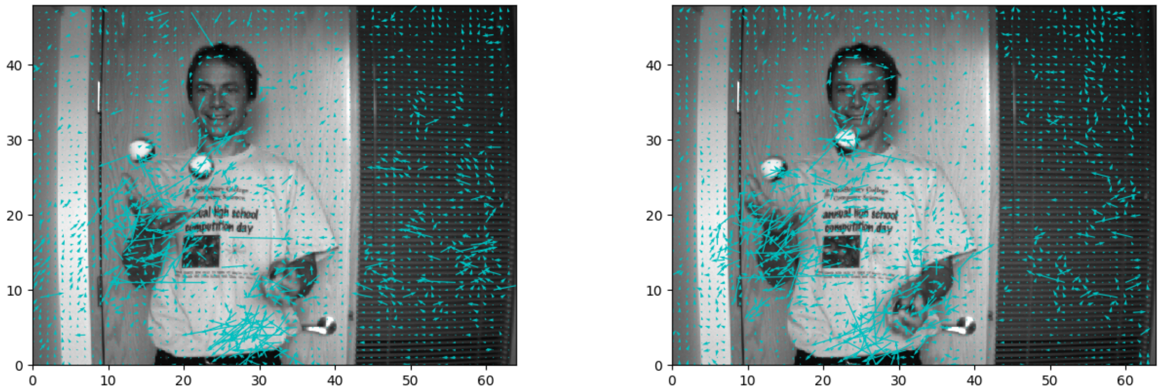
(a) Yosemite image 1

(b) Yosemite image 2

(c) Dog 0

(d) Dog 1

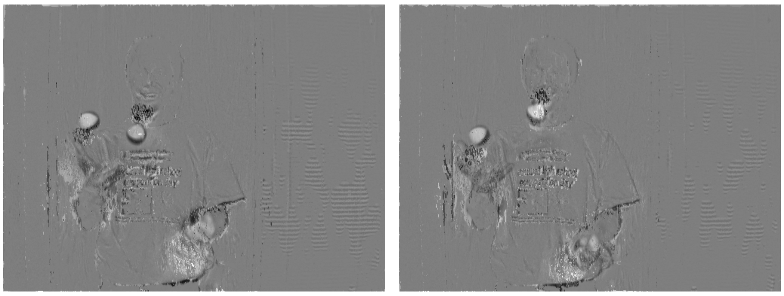
Figure 10: Difference between the wrapped and destination image



(a) Juggle 1

(b) Juggle 2

Figure 11: Lucas Kanade displacement arrows draw over original image



(a) Juggle 1

(b) Juggle 2

Figure 12: Difference between the wrapped and destination image