05_BlockMatching_HarrisCorners_final

November 27, 2019

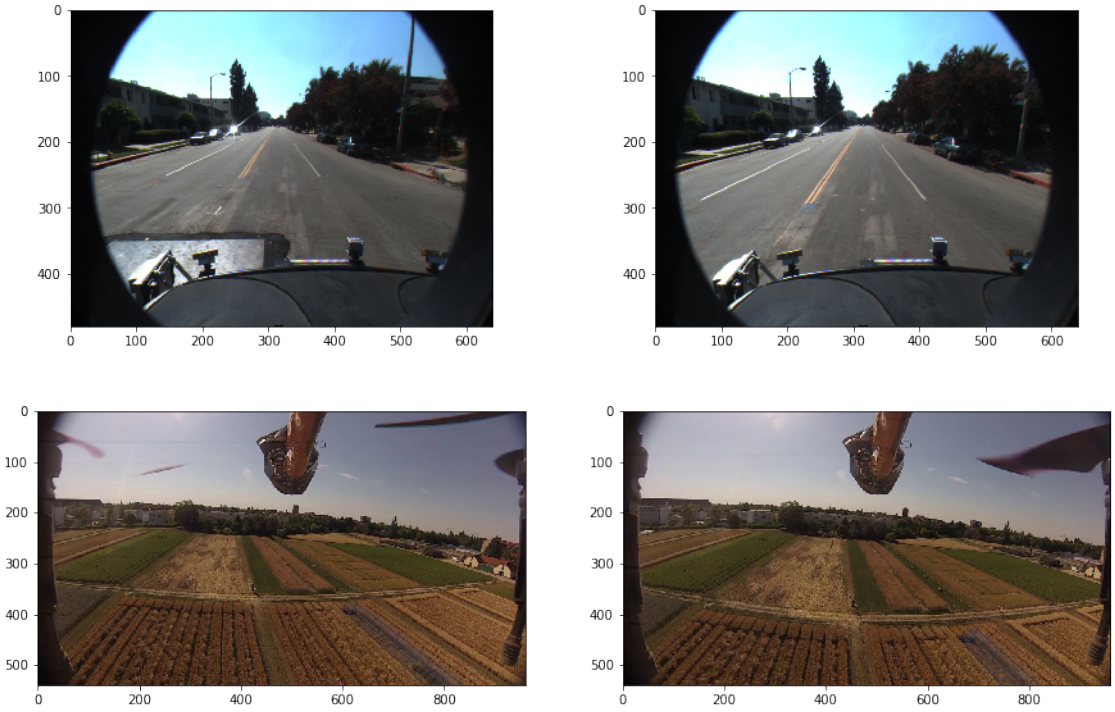
1 Assignment 5: Block Matching and Harris Corner Detection

1.1 Ex. 5.1 Dense Optical Flow by Block Matching

- implement the block matching method as shown in the lecture
- take two frames from the datasets "lane_detection" or "copter_flight" with variable distances in time (1, 2, x) and compute the vector flow field
- display a subset of flow vectors on the gray-value version of the first image, by drawing a respective line. adjust the grid density such that not too many vectors overlap (**RESULT**)

[3]:
```
%matplotlib inline
import matplotlib.pyplot as plt
from skimage import io, data, feature, color
import numpy as np

# Chose other images if you like
lane1 = io.imread('images/lane_detection/f00000.png')
lane2 = io.imread('images/lane_detection/f00001.png')

# Footage from our Neurocopter project:
# http://berlinbiorobotics.blog/projects/neurocopter/
copter1 = io.imread('images/copter_flight/frame050.jpg')
copter2 = io.imread('images/copter_flight/frame052.jpg')

fig = plt.figure(figsize=(15, 10))
ax11 = plt.subplot(2, 2, 1)
ax12 = plt.subplot(2, 2, 2)
ax21 = plt.subplot(2, 2, 3)
ax22 = plt.subplot(2, 2, 4)

ax11.imshow(lane1)
ax12.imshow(lane2)

ax21.imshow(copter1)
ax22.imshow(copter2)
```

[3]:
```
<matplotlib.image.AxesImage at 0x1079bdc50>
```



[108]:
```
from skimage import io, data, feature, color
import numpy as np


def get_window_coords(x, y, window_size, shape=None, clip=None, as_tuple=False):
    """
    :param window_size: (height, width) values should be odd

    Returned values are exclusive so ready for slicing.
    """

    delta_y, delta_x = window_size // 2
    if clip:
        clip_y, clip_x = clip
    else:
        assert shape, "'shape' argument required, when 'clip' is not given"
        height, width = shape
        clip_y = (0, height)
        clip_x = (0, width)
    y_min, y_max = np.clip(np.array([y - delta_y, y + delta_y + 1]), *clip_y)
    x_min, x_max = np.clip(np.array([x - delta_x, x + delta_x + 1]), *clip_x)
    if as_tuple:
        # No 'step' value here...
        return ((y_min, y_max), (x_min, x_max))
```

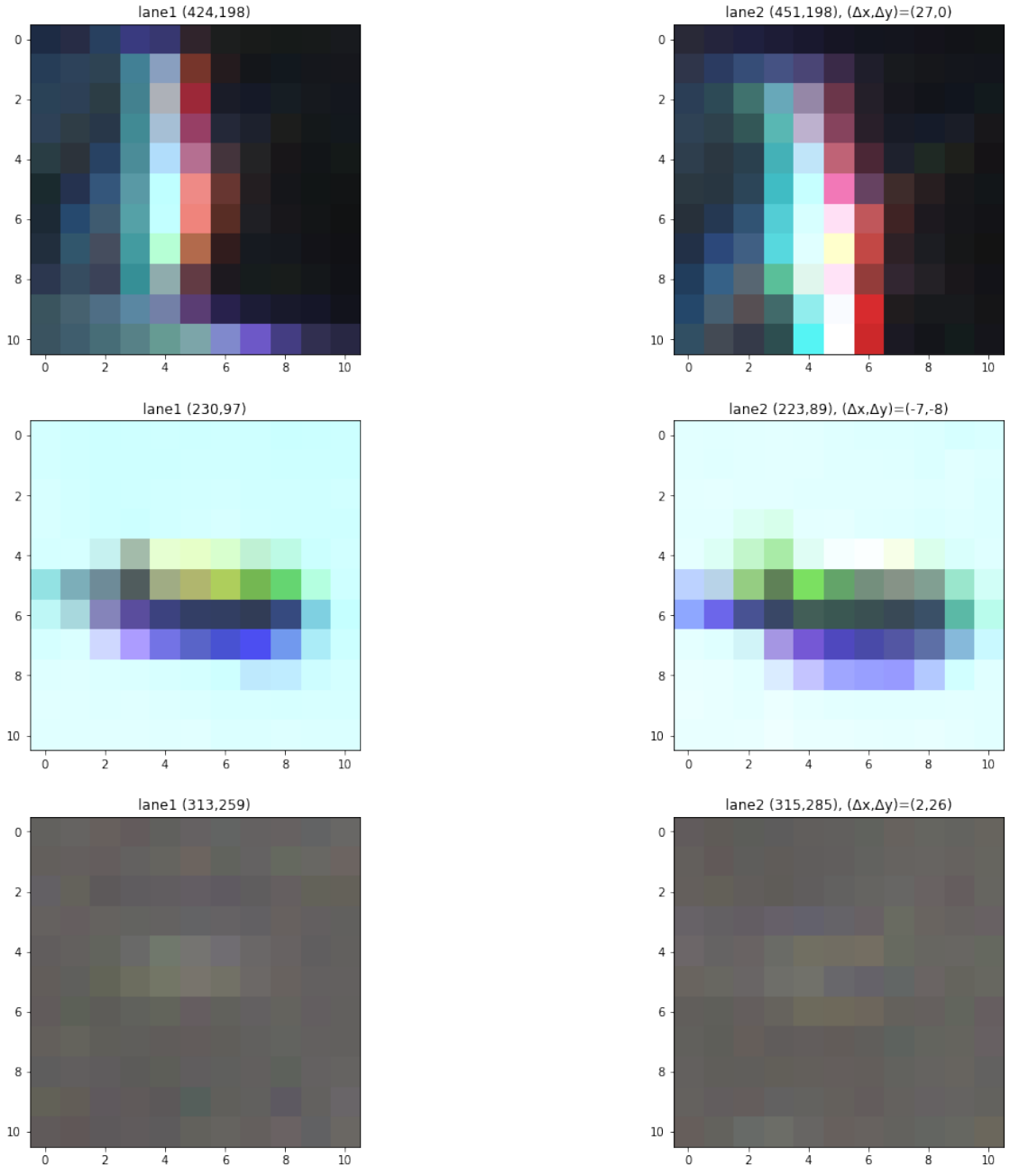
```
    return np.s_[y_min:y_max, x_min:x_max]


def get_window(image, x, y, window_size, shape):
    return image[get_window_coords(x, y, window_size, shape)]


def calc_ssd(image, x, y, dx, dy, block, shape):
    ty, tx = y + dy, x + dx
    shifted_block = get_window(image, tx, ty, BLOCK_SIZE, shape)
    return np.sum(np.square(block - shifted_block))


def calc_flow(image, x, y, block, shape):
    flow_vector = None
    min_ssd = None
    ref = np.array([x, y])

    height, width = shape
    pad_y, pad_x = block_size // 2
    clip = ((pad_y, height - pad_y), (pad_x, width - pad_x))
    (y_min, y_max), (x_min, x_max) = get_window_coords(
        x, y,
        SEARCH_WINDOW_SIZE,
        clip=clip,
        as_tuple=True,
    )
    for yi in range(y_min, y_max):
        for xi in range(x_min, x_max):
            dx, dy = np.array([xi, yi]) - ref
            ssd = calc_ssd(image, x, y, dx, dy, block, shape)
            if min_ssd is None or ssd < min_ssd:
                min_ssd = ssd
                flow_vector = np.array([dx, dy])
    return flow_vector


store = {}
BLOCK_SIZE = np.array([11, 11])
SEARCH_WINDOW_SIZE = np.array([71, 71])
sequence = [color.rgb2gray(image) for image in [lane1, lane2]]
shape = sequence[0].shape
print(shape)
for t, image in enumerate(sequence[1:], start=1):
    for x, y in [
        [424, 198],
        [230, 97],
```

```
        [313, 259],
        # Street sign...not working so well.
        # [496, 153],
    ]:
        print(f'at x={x}, y={y}. finding flow vector...')
        block = get_window(sequence[t - 1], x, y, BLOCK_SIZE, shape)
        search_window = get_window(image, x, y, SEARCH_WINDOW_SIZE, shape)
        flow_vector = calc_flow(image, x, y, block, shape)
        store[(x, y)] = flow_vector

print(store)
None
```

```
(480, 640)
at x=424, y=198. finding flow vector…
at x=230, y=97. finding flow vector…
at x=313, y=259. finding flow vector…
{(424, 198): array([27,  0]), (230, 97): array([-7, -8]), (313, 259): array([ 2,
26])}
```

[109]:
```
from skimage.draw import line, rectangle_perimeter as rect


N = len(store)
fig = plt.figure(figsize=(18, 10*N), dpi=72)
fig.tight_layout()
grid = fig.add_gridspec(N + 2, 2)

res_img = np.copy(lane2)
for i, ((x, y), (dx, dy)) in enumerate(store.items()):
    rr, cc = line(y, x, y + dy, x + dx)
    res_img[rr, cc] = np.array([255, 0, 0])
    (y_min, y_max), (x_min, x_max) = get_window_coords(x + dx, y + dy,␣
 ↪BLOCK_SIZE, shape, as_tuple=True)
    rr, cc = rect((y_min, x_min), (y_max, x_max), shape=shape)
    res_img[rr, cc] = np.array([0, 255, 0])

    ax1 = fig.add_subplot(grid[i, 0])
    ax2 = fig.add_subplot(grid[i, 1])

    ax1.imshow(get_window(lane1, x, y, block_size, shape))
    ax1.set_title(f'lane1 ({x},{y})')
    ax2.imshow(get_window(lane2, x + dx, y + dy, block_size, shape))
    ax2.set_title(f'lane2 ({x + dx},{y + dy}), (Δx,Δy)=({dx},{dy})')


fig = plt.figure(figsize=(18, 20), dpi=72)
```

```
ax1 = plt.subplot(2, 1, 1)
ax2 = plt.subplot(2, 1, 2)
ax1.set_title(f'lane1')
ax1.imshow(lane1)
ax2.set_title(f'lane2')
ax2.imshow(res_img)

plt.show()
None
```




lane1
0
100
200
300
400
0
100
200
300
400
500
600


lane2
0
100
200
300
400
0
100
200
300
400
500
600

[112]:
```
from skimage.draw import line


store_big = {}
STEP_SIZE = BLOCK_SIZE[0] * 2
for t, image in enumerate(sequence[1:], start=1):
    pad_y, pad_x = block_size // 2
    for x in range(pad_x, shape[1] - pad_x, STEP_SIZE):
        for y in range(pad_y, shape[0] - pad_y, STEP_SIZE):
            print(f'at x={x}, y={y}. finding flow vector...')
            block = get_window(sequence[t - 1], x, y, BLOCK_SIZE, shape)
            search_window = get_window(image, x, y, SEARCH_WINDOW_SIZE, shape)
            flow_vector = calc_flow(image, x, y, block, shape)
            store_big[(x, y)] = flow_vector

print(store_big)


fig = plt.figure(figsize=(18, 10), dpi=72)
res_img = np.copy(lane2)
for i, ((x, y), (dx, dy)) in enumerate(store_big.items()):
    rr, cc = line(y, x, y + dy, x + dx)
    res_img[rr, cc] = np.array([255, 0, 0])

io.imshow(res_img)
```

```
{(5, 5): array([0, 0]), (5, 27): array([  6, -12]), (5, 49): array([  5, -13]),
(5, 71): array([  9, -33]), (5, 93): array([  5, -34]), (5, 115): array([  1,
-33]), (5, 137): array([  0, -29]), (5, 159): array([0, 6]), (5, 181): array([
0, -10]), (5, 203): array([  0, -35]), (5, 225): array([ 0, 33]), (5, 247):
array([  0, -21]), (5, 269): array([ 2, 24]), (5, 291): array([ 1, 23]), (5,
313): array([0, 1]), (5, 335): array([  1, -19]), (5, 357): array([ 0, 27]), (5,
379): array([ 1, 28]), (5, 401): array([ 0, 21]), (5, 423): array([0, 3]), (5,
445): array([2, 5]), (5, 467): array([0, 1]), (27, 5): array([-17,  34]), (27,
27): array([  7, -21]), (27, 49): array([-3,  7]), (27, 71): array([ 0, -5]),
(27, 93): array([  2, -15]), (27, 115): array([ 1, 11]), (27, 137): array([-10,
-34]), (27, 159): array([-5, 26]), (27, 181): array([-4,  0]), (27, 203):
array([-14,  26]), (27, 225): array([ 0, 28]), (27, 247): array([0, 3]), (27,
269): array([-1, -7]), (27, 291): array([ 2, 13]), (27, 313): array([-10, -23]),
(27, 335): array([-1, -7]), (27, 357): array([-2, -2]), (27, 379): array([0,
0]), (27, 401): array([0, 0]), (27, 423): array([ 1, -6]), (27, 445): array([
2, -27]), (27, 467): array([0, 0]), (49, 5): array([-21,  22]), (49, 27):
array([ -4, -10]), (49, 49): array([ 1, -7]), (49, 71): array([-9, -2]), (49,
93): array([  5, -27]), (49, 115): array([-9, 20]), (49, 137): array([-3,  2]),
(49, 159): array([ 2, 14]), (49, 181): array([ 1, -7]), (49, 203): array([34,
```

1]), (49, 225): array([22, -3]), (49, 247): array([-3, 35]), (49, 269): array([
7, 24]), (49, 291): array([25, 35]), (49, 313): array([10, 35]), (49, 335):
array([7, 21]), (49, 357): array([-3, -5]), (49, 379): array([-1, 0]), (49,
401): array([-4, 13]), (49, 423): array([-6, 0]), (49, 445): array([-4, 11]),
(49, 467): array([-5, -35]), (71, 5): array([-30, 30]), (71, 27): array([-35,
25]), (71, 49): array([-12, 9]), (71, 71): array([3, -6]), (71, 93): array([
13, -25]), (71, 115): array([33, 12]), (71, 137): array([35, 7]), (71, 159):
array([-21, 35]), (71, 181): array([-30, 4]), (71, 203): array([-26, -17]),
(71, 225): array([-1, 2]), (71, 247): array([-26, 23]), (71, 269): array([10,
25]), (71, 291): array([-22, 15]), (71, 313): array([6, 34]), (71, 335):
array([0, 12]), (71, 357): array([35, 13]), (71, 379): array([29, -4]), (71,
401): array([7, 11]), (71, 423): array([-7, -8]), (71, 445): array([-13, -1]),
(71, 467): array([-13, -22]), (93, 5): array([-6, 0]), (93, 27): array([-6,
8]), (93, 49): array([-2, 2]), (93, 71): array([23, -18]), (93, 93): array([
14, -27]), (93, 115): array([-27, -5]), (93, 137): array([-28, -2]), (93,
159): array([3, 11]), (93, 181): array([-30, 3]), (93, 203): array([-31,
7]), (93, 225): array([-34, 10]), (93, 247): array([-31, 35]), (93, 269):
array([33, 31]), (93, 291): array([21, 32]), (93, 313): array([-29, -8]), (93,
335): array([-32, 25]), (93, 357): array([33, -29]), (93, 379): array([-5,
26]), (93, 401): array([1, 1]), (93, 423): array([-1, 0]), (93, 445):
array([10, 13]), (93, 467): array([-6, -10]), (115, 5): array([-4, 4]), (115,
27): array([7, -9]), (115, 49): array([0, 2]), (115, 71): array([2, -17]),
(115, 93): array([14, -30]), (115, 115): array([-26, -5]), (115, 137):
array([-33, 32]), (115, 159): array([-21, 1]), (115, 181): array([12, 0]),
(115, 203): array([-33, 7]), (115, 225): array([-1, 2]), (115, 247):
array([-9, 33]), (115, 269): array([32, 28]), (115, 291): array([-11, 30]),
(115, 313): array([-35, 35]), (115, 335): array([-28, 5]), (115, 357):
array([25, 8]), (115, 379): array([7, -1]), (115, 401): array([0, 0]), (115,
423): array([0, 1]), (115, 445): array([0, 0]), (115, 467): array([-1, 0]),
(137, 5): array([-1, 0]), (137, 27): array([22, -22]), (137, 49): array([23,
-35]), (137, 71): array([-10, -30]), (137, 93): array([-9, -24]), (137, 115):
array([-35, -21]), (137, 137): array([-17, 0]), (137, 159): array([-17, 0]),
(137, 181): array([-11, 2]), (137, 203): array([-14, 4]), (137, 225):
array([-33, 29]), (137, 247): array([12, 30]), (137, 269): array([-27, 28]),
(137, 291): array([-35, 34]), (137, 313): array([8, 16]), (137, 335):
array([35, 35]), (137, 357): array([35, -32]), (137, 379): array([0, 0]), (137,
401): array([0, 1]), (137, 423): array([-1, 0]), (137, 445): array([-10, 9]),
(137, 467): array([-1, 1]), (159, 5): array([31, 0]), (159, 27): array([32,
-22]), (159, 49): array([34, -35]), (159, 71): array([19, -18]), (159, 93):
array([-17, -21]), (159, 115): array([-30, -25]), (159, 137): array([-9, -2]),
(159, 159): array([-14, -1]), (159, 181): array([-13, 2]), (159, 203):
array([-1, 1]), (159, 225): array([33, 24]), (159, 247): array([-31, 16]),
(159, 269): array([-35, 31]), (159, 291): array([-35, 26]), (159, 313):
array([-17, 25]), (159, 335): array([21, 34]), (159, 357): array([35, -17]),
(159, 379): array([22, 24]), (159, 401): array([-29, -30]), (159, 423): array([
1, -1]), (159, 445): array([-17, 22]), (159, 467): array([-14, -7]), (181, 5):
array([28, 0]), (181, 27): array([28, -22]), (181, 49): array([35, -23]),
(181, 71): array([31, -22]), (181, 93): array([3, -22]), (181, 115):

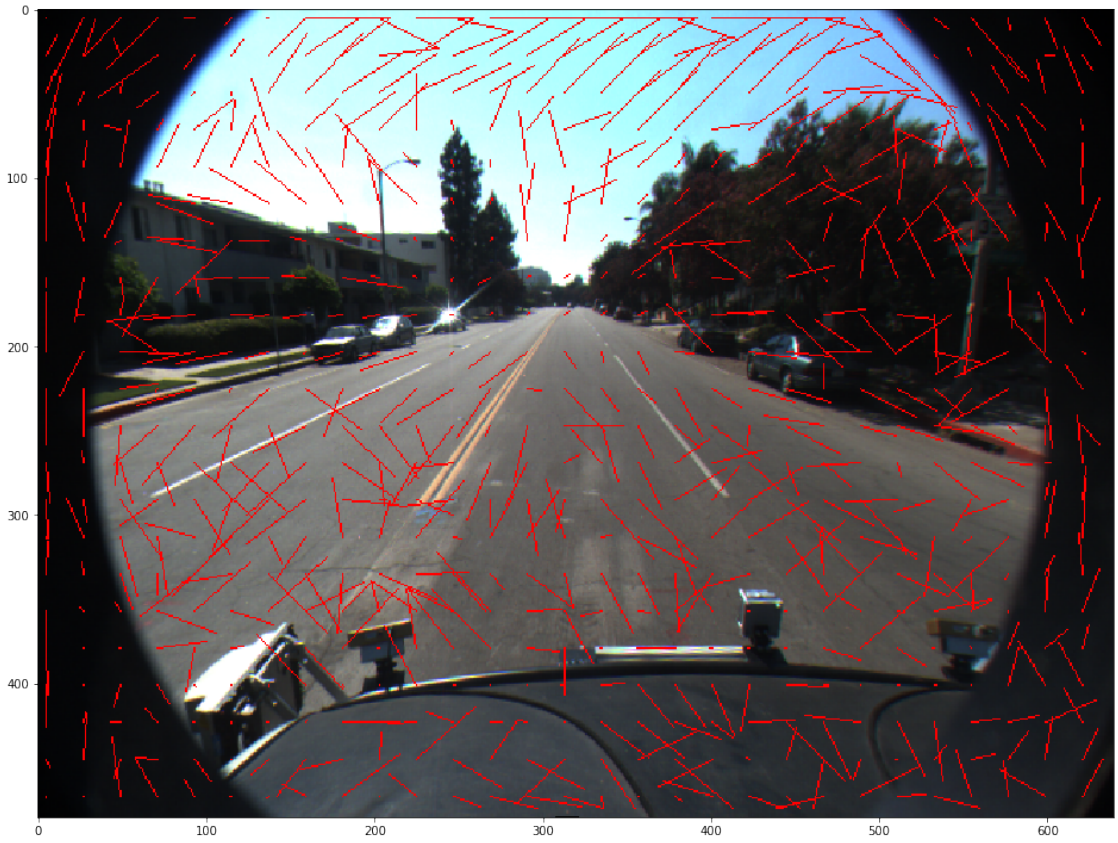
array([-22, -33]), (181, 137): array([-3, -1]), (181, 159): array([35, 6]),
(181, 181): array([-8, 1]), (181, 203): array([-16,   5]), (181, 225):
array([-11, 28]), (181, 247): array([31, 30]), (181, 269): array([33, 22]),
(181, 291): array([33, 3]), (181, 313): array([-6, -35]), (181, 335):
array([-3, 35]), (181, 357): array([17, -17]), (181, 379): array([33, -35]),
(181, 401): array([1, 1]), (181, 423): array([34, 0]), (181, 445): array([-25,
6]), (181, 467): array([34, -25]), (203, 5): array([35, 18]), (203, 27):
array([35, -4]), (203, 49): array([34, -23]), (203, 71): array([28, -27]),
(203, 93): array([-26, -31]), (203, 115): array([-7, -28]), (203, 137):
array([-24, -8]), (203, 159): array([-5, -1]), (203, 181): array([-10,   2]),
(203, 203): array([-12,   4]), (203, 225): array([-30, 15]), (203, 247):
array([23, 31]), (203, 269): array([3, 25]), (203, 291): array([4, 0]), (203,
313): array([6, -21]), (203, 335): array([34, 15]), (203, 357): array([-5,
-18]), (203, 379): array([-1, 0]), (203, 401): array([0, 0]), (203, 423):
array([10, 1]), (203, 445): array([1, -16]), (203, 467): array([0, 0]), (225,
5): array([33, 0]), (225, 27): array([34, -22]), (225, 49): array([35, -16]),
(225, 71): array([0, -33]), (225, 93): array([-8, -8]), (225, 115):
array([-28, -26]), (225, 137): array([-1, -2]), (225, 159): array([-2, -1]),
(225, 181): array([-8, 1]), (225, 203): array([-33, 11]), (225, 225):
array([-21, 20]), (225, 247): array([8, 22]), (225, 269): array([-14, 24]),
(225, 291): array([22, 9]), (225, 313): array([28, -33]), (225, 335):
array([31, -1]), (225, 357): array([-27, -18]), (225, 379): array([-6, -35]),
(225, 401): array([-19, -35]), (225, 423): array([-16,   1]), (225, 445):
array([35, -31]), (225, 467): array([-16, -13]), (247, 5): array([35, 8]),
(247, 27): array([35, -14]), (247, 49): array([34, -17]), (247, 71):
array([-1, -5]), (247, 93): array([-1, -4]), (247, 115): array([-1, -3]), (247,
137): array([-1, -2]), (247, 159): array([-1, 2]), (247, 181): array([-2, 0]),
(247, 203): array([-2, 2]), (247, 225): array([-33, 21]), (247, 247):
array([-27, 35]), (247, 269): array([-23,   3]), (247, 291): array([-32,   4]),
(247, 313): array([5, 1]), (247, 335): array([19, 22]), (247, 357): array([-27,
-13]), (247, 379): array([-27, -35]), (247, 401): array([1, 0]), (247, 423):
array([-16, -6]), (247, 445): array([29, 17]), (247, 467): array([16, 6]),
(269, 5): array([35, 0]), (269, 27): array([35, -22]), (269, 49): array([26,
-16]), (269, 71): array([12, -28]), (269, 93): array([0, -4]), (269, 115):
array([1, -3]), (269, 137): array([1, -2]), (269, 159): array([-7, 4]), (269,
181): array([1, -1]), (269, 203): array([-13, 10]), (269, 225): array([-20,
29]), (269, 247): array([-3, 6]), (269, 269): array([-14, 34]), (269, 291):
array([0, 16]), (269, 313): array([14, -25]), (269, 335): array([8, 9]), (269,
357): array([-32, -35]), (269, 379): array([-35, -35]), (269, 401): array([0,
0]), (269, 423): array([7, 0]), (269, 445): array([16, -23]), (269, 467):
array([28, 7]), (291, 5): array([35, 0]), (291, 27): array([35, -22]), (291,
49): array([31, -32]), (291, 71): array([15, -19]), (291, 93): array([-5,
-33]), (291, 115): array([4, -24]), (291, 137): array([-4, -33]), (291, 159):
array([2, -1]), (291, 181): array([1, -1]), (291, 203): array([-28, 21]),
(291, 225): array([8, 1]), (291, 247): array([-8, 33]), (291, 269): array([-13,
25]), (291, 291): array([6, 33]), (291, 313): array([19, -28]), (291, 335):
array([27, 20]), (291, 357): array([20, -1]), (291, 379): array([19, 2]), (291,
401): array([0, 0]), (291, 423): array([-5, -1]), (291, 445): array([34, 18]),

(291, 467): array([32, 6]), (313, 5): array([35, 0]), (313, 27): array([35,
-22]), (313, 49): array([35, -35]), (313, 71): array([27, -10]), (313, 93):
array([-9, -33]), (313, 115): array([31, -14]), (313, 137): array([5, -32]),
(313, 159): array([3, -2]), (313, 181): array([3, -2]), (313, 203): array([7,
10]), (313, 225): array([24, 25]), (313, 247): array([35, 0]), (313, 269):
array([9, -5]), (313, 291): array([-5, 19]), (313, 313): array([-11, -34]),
(313, 335): array([5, 20]), (313, 357): array([17, 31]), (313, 379): array([0,
28]), (313, 401): array([-2, 0]), (313, 423): array([0, 0]), (313, 445):
array([-27, 25]), (313, 467): array([23, 7]), (335, 5): array([35, 0]), (335,
27): array([35, -22]), (335, 49): array([35, -35]), (335, 71): array([35,
-35]), (335, 93): array([26, -15]), (335, 115): array([5, -32]), (335, 137):
array([3, -3]), (335, 159): array([3, -3]), (335, 181): array([3, -2]), (335,
203): array([1, 6]), (335, 225): array([10, 13]), (335, 247): array([10, 33]),
(335, 269): array([-18, -20]), (335, 291): array([31, 35]), (335, 313): array([
6, -1]), (335, 335): array([19, 35]), (335, 357): array([-12, 3]), (335, 379):
array([-1, 0]), (335, 401): array([14, -1]), (335, 423): array([28, 21]), (335,
445): array([20, 29]), (335, 467): array([1, 7]), (357, 5): array([35, 0]),
(357, 27): array([34, -22]), (357, 49): array([35, -26]), (357, 71): array([
35, -35]), (357, 93): array([19, -17]), (357, 115): array([5, -5]), (357,
137): array([4, -4]), (357, 159): array([4, -2]), (357, 181): array([5, -2]),
(357, 203): array([6, 13]), (357, 225): array([10, 8]), (357, 247): array([9,
19]), (357, 269): array([14, 27]), (357, 291): array([35, 35]), (357, 313):
array([34, 18]), (357, 335): array([12, 24]), (357, 357): array([7, 1]), (357,
379): array([-1, 0]), (357, 401): array([21, 25]), (357, 423): array([27, 21]),
(357, 445): array([34, -16]), (357, 467): array([0, 0]), (379, 5): array([35,
12]), (379, 27): array([35, -11]), (379, 49): array([28, -33]), (379, 71):
array([35, -35]), (379, 93): array([6, -8]), (379, 115): array([8, -9]),
(379, 137): array([-26, 20]), (379, 159): array([-3, -16]), (379, 181):
array([6, -2]), (379, 203): array([11, 1]), (379, 225): array([16, 28]), (379,
247): array([14, 13]), (379, 269): array([20, -14]), (379, 291): array([30,
-19]), (379, 313): array([12, 33]), (379, 335): array([2, 34]), (379, 357):
array([15, -13]), (379, 379): array([-21, 0]), (379, 401): array([18, 25]),
(379, 423): array([27, 10]), (379, 445): array([34, 29]), (379, 467): array([4,
3]), (401, 5): array([35, 0]), (401, 27): array([35, -22]), (401, 49): array([
35, -35]), (401, 71): array([27, -4]), (401, 93): array([7, -9]), (401, 115):
array([-34, 24]), (401, 137): array([-30, 4]), (401, 159): array([12, 1]),
(401, 181): array([33, -2]), (401, 203): array([-27, -28]), (401, 225):
array([17, 11]), (401, 247): array([28, 20]), (401, 269): array([29, -7]), (401,
291): array([16, -20]), (401, 313): array([23, -21]), (401, 335): array([-21,
34]), (401, 357): array([-7, -10]), (401, 379): array([-2, 0]), (401, 401):
array([13, 30]), (401, 423): array([12, 9]), (401, 445): array([-32, 16]),
(401, 467): array([-29, -9]), (423, 5): array([35, 0]), (423, 27): array([35,
-22]), (423, 49): array([35, -35]), (423, 71): array([35, -25]), (423, 93):
array([13, -8]), (423, 115): array([27, -18]), (423, 137): array([-29, 3]),
(423, 159): array([-26, -18]), (423, 181): array([14, -2]), (423, 203):
array([28, 4]), (423, 225): array([27, 8]), (423, 247): array([-1, 12]), (423,
269): array([19, 17]), (423, 291): array([31, -19]), (423, 313): array([35,
-21]), (423, 335): array([-27, 8]), (423, 357): array([-1, 0]), (423, 379):

array([-1, 0]), (423, 401): array([-5, 27]), (423, 423): array([27, -1]), (423, 445): array([-2, -1]), (423, 467): array([20, 7]), (445, 5): array([35, 0]), (445, 27): array([33, -21]), (445, 49): array([34, -32]), (445, 71): array([35, -22]), (445, 93): array([13, -18]), (445, 115): array([27, -15]), (445, 137): array([17, 19]), (445, 159): array([28, -7]), (445, 181): array([33, 31]), (445, 203): array([27, 3]), (445, 225): array([26, 8]), (445, 247): array([32, 17]), (445, 269): array([7, 19]), (445, 291): array([32, 25]), (445, 313): array([25, 35]), (445, 335): array([35, -32]), (445, 357): array([-1, 0]), (445, 379): array([-1, 0]), (445, 401): array([25, 1]), (445, 423): array([35, 35]), (445, 445): array([6, -19]), (445, 467): array([8, 7]), (467, 5): array([35, 9]), (467, 27): array([35, -12]), (467, 49): array([35, -4]), (467, 71): array([28, -20]), (467, 93): array([7, 9]), (467, 115): array([34, -16]), (467, 137): array([5, -16]), (467, 159): array([2, 2]), (467, 181): array([25, -3]), (467, 203): array([28, 0]), (467, 225): array([0, -15]), (467, 247): array([-35, -6]), (467, 269): array([16, 14]), (467, 291): array([30, 15]), (467, 313): array([-1, -6]), (467, 335): array([-2, -25]), (467, 357): array([35, -35]), (467, 379): array([22, 7]), (467, 401): array([3, 0]), (467, 423): array([16, 33]), (467, 445): array([-6, 29]), (467, 467): array([-28,   7]), (489, 5): array([35, 33]), (489, 27): array([35, 21]), (489, 49): array([35, -1]), (489, 71): array([31, -21]), (489, 93): array([31, -19]), (489, 115): array([-22, -11]), (489, 137): array([15, -20]), (489, 159): array([4, -29]), (489, 181): array([21, 23]), (489, 203): array([-27, -19]), (489, 225): array([33, 17]), (489, 247): array([21, 6]), (489, 269): array([-32,   2]), (489, 291): array([33, 35]), (489, 313): array([33, 10]), (489, 335): array([-27, -27]), (489, 357): array([-15, -10]), (489, 379): array([35, -30]), (489, 401): array([0, 0]), (489, 423): array([-33, -4]), (489, 445): array([-4, 12]), (489, 467): array([35, -18]), (511, 5): array([8, 8]), (511, 27): array([34, 34]), (511, 49): array([35, 9]), (511, 71): array([29, 13]), (511, 93): array([-2, -30]), (511, 115): array([5, -30]), (511, 137): array([29, -32]), (511, 159): array([-13, -31]), (511, 181): array([22, 3]), (511, 203): array([32, -22]), (511, 225): array([23, 13]), (511, 247): array([27, 9]), (511, 269): array([2, 8]), (511, 291): array([-31,   1]), (511, 313): array([5, 10]), (511, 335): array([30, 13]), (511, 357): array([4, 1]), (511, 379): array([17, 11]), (511, 401): array([1, 0]), (511, 423): array([16, 32]), (511, 445): array([-1, 0]), (511, 467): array([17, -1]), (533, 5): array([8, 6]), (533, 27): array([2, 2]), (533, 49): array([13, 18]), (533, 71): array([10, -4]), (533, 93): array([8, -27]), (533, 115): array([19, 25]), (533, 137): array([16, -7]), (533, 159): array([-18, -35]), (533, 181): array([2, 34]), (533, 203): array([30, 7]), (533, 225): array([17, 15]), (533, 247): array([34, 5]), (533, 269): array([27, 22]), (533, 291): array([-24,   7]), (533, 313): array([-6, 35]), (533, 335): array([22, 19]), (533, 357): array([34, -7]), (533, 379): array([-1, 0]), (533, 401): array([1, 0]), (533, 423): array([5, 35]), (533, 445): array([7, -8]), (533, 467): array([27, 7]), (555, 5): array([30, 35]), (555, 27): array([-2, -4]), (555, 49): array([5, 8]), (555, 71): array([-2, -4]), (555, 93): array([-10, -25]), (555, 115): array([5, -9]), (555, 137): array([0, -6]), (555, 159): array([-16, -35]), (555, 181): array([-4, 35]), (555, 203): array([-21, 13]), (555, 225): array([-19, 21]), (555, 247): array([25, 11]), (555, 269): array([31, 14]), (555, 291):

```
array([-11,  18]), (555, 313): array([-5,  4]), (555, 335): array([ 1, 20]),
(555, 357): array([12, -6]), (555, 379): array([-1, -1]), (555, 401): array([0,
0]), (555, 423): array([17, -2]), (555, 445): array([14, -3]), (555, 467):
array([ -9, -27]), (577, 5): array([1, 7]), (577, 27): array([12, 15]), (577,
49): array([-10, -16]), (577, 71): array([-12, -15]), (577, 93): array([-12,
-30]), (577, 115): array([4, 7]), (577, 137): array([-27, -35]), (577, 159):
array([ 2, 30]), (577, 181): array([25, 28]), (577, 203): array([-32,  15]),
(577, 225): array([-34,  20]), (577, 247): array([5, 5]), (577, 269): array([14,
1]), (577, 291): array([-26,   9]), (577, 313): array([-35,  34]), (577, 335):
array([-2,  3]), (577, 357): array([0, 1]), (577, 379): array([  6, -15]), (577,
401): array([  6, -22]), (577, 423): array([18, -4]), (577, 445): array([ 12,
-17]), (577, 467): array([ -3, -21]), (599, 5): array([ 7, 18]), (599, 27):
array([5, 0]), (599, 49): array([ -5, -19]), (599, 71): array([-14, -15]), (599,
93): array([-17, -30]), (599, 115): array([12, 21]), (599, 137): array([  1,
-10]), (599, 159): array([16, 27]), (599, 181): array([ -4, -13]), (599, 203):
array([  0, -22]), (599, 225): array([ 4, 35]), (599, 247): array([ 0, 11]),
(599, 269): array([3, 0]), (599, 291): array([  2, -10]), (599, 313): array([
6, -33]), (599, 335): array([  5, -11]), (599, 357): array([  2, -27]), (599,
379): array([ 13, -11]), (599, 401): array([-16, -26]), (599, 423): array([ 30,
-13]), (599, 445): array([  4, -25]), (599, 467): array([-27, -16]), (621, 5):
array([11, 16]), (621, 27): array([6, 9]), (621, 49): array([-11, -27]), (621,
71): array([-1, -6]), (621, 93): array([  0, -14]), (621, 115): array([ -1,
-21]), (621, 137): array([ 10, -25]), (621, 159): array([ -3, -25]), (621, 181):
array([ 1, -3]), (621, 203): array([2, 1]), (621, 225): array([-10, -14]), (621,
247): array([10, 11]), (621, 269): array([ -3, -28]), (621, 291): array([  2,
-20]), (621, 313): array([  2, -35]), (621, 335): array([ -2, -24]), (621, 357):
array([ 10, -25]), (621, 379): array([  4, -14]), (621, 401): array([-14, -25]),
(621, 423): array([-11, -20]), (621, 445): array([  9, -18]), (621, 467):
array([ 12, -22])}
```

[112]: <matplotlib.image.AxesImage at 0x1318af710>



1.2 Ex. 5.2 Harris Corner Detection

- implement the Harris Corner Detector as discussed in the lecture
- compute corners in the first image and track them with Lucas-Kanade (use e.g. the function "calcOpticalFlowPyrLK" in OpenCV)
- mark the positions of your Harris corners and draw the flow vectors found by Lucas-Kanade on the gray-value versions of the first image (**RESULT**)

[4]:
```
from skimage import color, filters

im = color.rgb2gray(copter1)
im_width, im_height = im.shape
H = np.empty([im_width, im_height, 4])
R = np.empty([im_width, im_height])
k = 0.04

print(im_width, im_height)

# 1) Compute derivations of I
I_x = filters.sobel_h(im)
I_y = filters.sobel_v(im)
```

```
# 2) Compute products of derivatives at every pixel
I_x2 = I_x*I_x
I_y2 = I_y*I_y
I_xy = I_x*I_y

# 3) Compute sums of products at every pixel
S_x2 = filters.sobel(I_x2)
S_y2 = filters.sobel(I_y2)
S_xy = filters.sobel(I_xy)

# 4) Define matrix at each pixel
for x in range(im_width):
    for y in range(im_height):
        H[x][y][0] = S_x2[x][y]
        H[x][y][1] = S_xy[x][y]
        H[x][y][2] = S_xy[x][y]
        H[x][y][3] = S_y2[x][y]

# 5) Compute detector response: R = det H - k* trace H **2            det=ad-bc
 ↪   trace=a+d
for x in range(im_width):
    for y in range(im_height):
        det = (H[x][y][0]*H[x][y][3]) - (H[x][y][1]*H[x][y][2]) # second term
 ↪always zero therefore obsolet, right?
        trace = H[x][y][0]+H[x][y][3]
        R[x][y] = det - k*(np.power(trace,2))
```

```
540 960
```

[5]:
```
# 6) Threshold the value of R. Compute nonmax suppression
R[R <= 0.005] = 0
rmax = R.max()
rmin = R.min()
print(rmax, rmin)
print(np.count_nonzero(R))

kernel_size = 3
result = np.zeros([im_width, im_height])

LC_vec = []

for x in range(im_width-(kernel_size-1)):
    for y in range(im_height-(kernel_size-1)):
        R_window = R[x:x+kernel_size,y:y+kernel_size]
        score = np.sum(R_window)
        result[x+1][y+1] = score
```

```
#second Treshold to reduce LC vector length
#result[result <= 0.04] = 0
#print(np.count_nonzero(result))

for x in range(im_width):
    for y in range(im_height):
        if(result[x][y] != 0.0):
            LC_vec.append((x,y))

# print(len(LC_vec))
# print(LC_vec)


fig = plt.figure(figsize=(20, 15))
plt.imshow(result, cmap='gray')
```

```
0.06350638982074053 0.0
180
```

[5]: <matplotlib.image.AxesImage at 0x12f103d50>

[6]:
```
# Get most valuable Harris Corner
corner = np.argmax(result)
corner_x = corner / im_height
corner_y = corner % im_height
```

```
HC = (corner_y, corner_x)
print(HC)
```

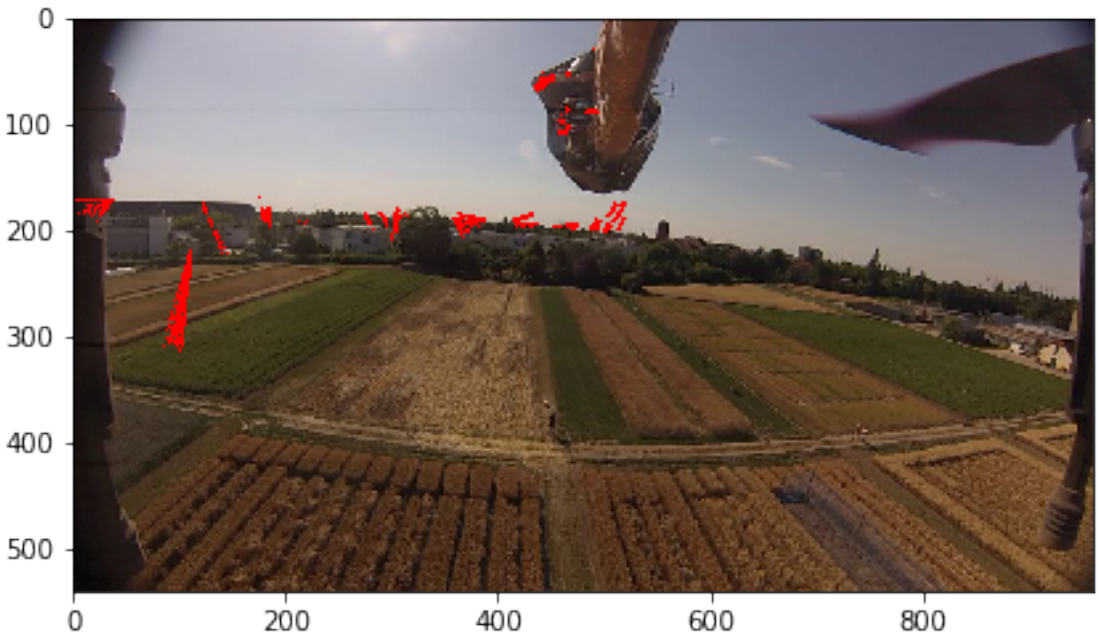
```
(491, 90.51145833333334)
```

[8]:
```
import cv2 as cv


copter1_gray = cv.imread('images/copter_flight/frame050.jpg', cv.
 ↪IMREAD_GRAYSCALE)
copter2_gray = cv.imread('images/copter_flight/frame052.jpg', cv.
 ↪IMREAD_GRAYSCALE)
print(type(copter2_gray))

lk_params = dict( winSize  = (15, 15),
                  maxLevel = 2,
                  criteria = (cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT,␣
 ↪10, 0.03))
prevPts = np.array(LC_vec, dtype=np.float32)
nextPts, status, err = cv.calcOpticalFlowPyrLK(
    copter1_gray, copter2_gray,
    prevPts,
    None,
#     [None for i in range(len(LC_vec))],
    **lk_params
)
print(nextPts.shape, '--', status.shape, err.shape)
```

```
<class 'numpy.ndarray'>
(1033, 2) -- (1033, 1) (1033, 1)
```

[9]:
```
from skimage.draw import line

res_img = np.copy(copter2)
for p, n, stat in zip(prevPts, nextPts, status):
    if stat == 1:
        y1, x1 = p.astype(np.uint)
        y2, x2 = n.astype(np.uint)
        rr, cc = line(y1, x1, y2, x2)
        res_img[rr, cc] = [255, 0, 0]

io.imshow(res_img)
```

[9]:
```
<matplotlib.image.AxesImage at 0x12d98c090>
```


0
100
200
300
400
500
0
200
400
600
800