2D Image Processing - Exercise Sheet 3: SIFT and Optical FLow

The notebook guides you through the exercise. Read all instructions carefully.

- **Deadline**: 13.06.2022 @ 23:59
- **Contact**: Michael.Fuerst@dfki.de
- Submission: As PDF Printout; filename is `ex03_2DIP_group_XY.pdf`, where XY is replaced by your group number.
- Allowed Libraries: Numpy, OpenCV and Matplotlib (unless a task specifically states differently).
- Copying or sharing code is NOT permitted and results in failing the exercise. However, you could compare produced outputs if you want to. (Btw, this includes copying code from the internet.)

Submission as PDF printout. You can generate a PDF directly from jupyterlab or if that does not work, export as HTML and then use your webbrowser to convert the HTML to a PDF. For the printout make sure, that all text/code is visible and readable. And that the figures have an appropriate size. (Check your file before submitting, without outputs you will not pass!)

0. Infrastructure: Cloud Image Loader

This is an image loader function, that loads the images needed for the exercise from the dfki-cloud into an opencv usable format. This allows for easy usage of colab, since you only need this notebook and no other files.

```
In [1]:
import cv2
import numpy as np
import matplotlib.pyplot as plt

import requests
from PIL import Image

def get_image(name, no_alpha=True):
    url = f'https://cloud.dfki.de/owncloud/index.php/s/THLirfoB6SYTetn/download?path=&files={name}'
    image = np.asarray(Image.open(requests.get(url, stream=True).raw))
    if no_alpha and len(image) > 2 and image.shape[2] == 4:
        image = image[:,:,:3]
    return image[:,:,::-1].copy()
```

1. SIFT and Feature Matching (8 Pts)

Theory (5 Pts)

Hint: Each 0.5 points expects one argument/step/etc.

1. List what methods are used for comparing two patches in the image (1.5 Pts).
2. Explain the ideas and steps of SIFT feature detection in detail. (2 Pts)
3. What are the advantages of SIFT compared to Harris? (0.5 Pts)
4. Also explain the idea of HOG as a descriptor. (1 Pts)

Solution:

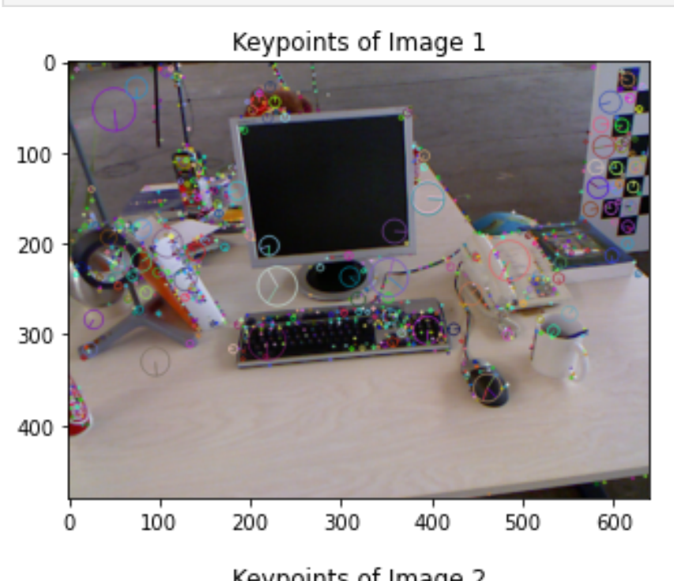
- **Patch Comparison**:
  - Cross-Correlation
  - Proximity - Search with Disparity Window
  - SIFT Desctiptor
- **SIFT**:
  1. Find the interest points that are scale and rotation invariant in an image using Blob Detection. It could be done using NLoG and DoG.
  2. After detecting the interest point determine the scale and location of each interest point and eliminate the weak keypoints.
  3. Assign orientations to each Keypoing using Histogram of Oriented Gradients (HoG) where histogram entries are weighed by the magnitude of the gradient and the gausian function with sigma equal to 1.5 times the scale of the interest point.
  4. Use local image gradients at the selected scale.
- **SIFT vs Harris**:
  - Harris is not invarient to changes in light, scale and viewpoint, whereas SIFT is robust to these changes
- **HOG**:
  1. HoG is used as a feature descriptor as it takes a window of a specific size around a keypoint and finds the features using magnitude and direction of the gradient.
  2. These features helps in finding the principal orientation of the keypoint. This rotation can be undoed to map the shifted image with the orignal image.

Programming Task (2 Pts)

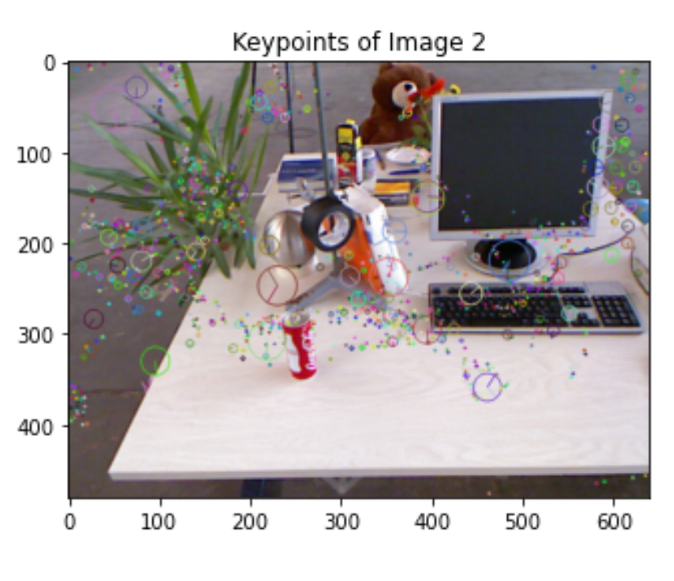
Given two RGB images, implement SIFT feature detection on both images, create proper feature descriptors and match the features between these two images and visualize the results.

```
In [6]:
# Solution
image1 = get_image("img1.png")
image2 = get_image("img2.png")


sift = cv2.SIFT_create()   #importing sift library


interestpoints_1, interestpoints_descriptors_1 = sift.detectAndCompute(image1,None) # finding interest points for image1
interestpoints_2, interestpoints_descriptors_2 = sift.detectAndCompute(image2,None) # finding interest points for image2



# Drawing Keypoints of both the images
img1KeyPoints=cv2.drawKeypoints(image1,interestpoints_1,image1,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
img2KeyPoints=cv2.drawKeypoints(image2,interestpoints_1,image2,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

plt.imshow(img1KeyPoints[:,:,::-1])
plt.title("Keypoints of Image 1")
plt.show()

plt.imshow(img2KeyPoints[:,:,::-1])
plt.title("Keypoints of Image 2")
plt.show()

# initializing the CV2 Brute Force matcher for comparing 2 images.
comparor = cv2.BFMatcher(cv2.NORM_L1, crossCheck=True)

# finding similar features
similar_features = comparor.match(interestpoints_descriptors_1,interestpoints_descriptors_2)
# sorting the filters based on there distace to each other
similar_features = sorted(similar_features, key = lambda x:x.distance)

image3 = cv2.drawMatches(image1, interestpoints_1, image2, interestpoints_2, similar_features[0:100], image2, flags=2)
# showing top 100 Features

# plt.subplot(1,2,1)
# plt.imshow(image1[:,:,::-1])
# plt.subplot(1,2,2)
# plt.imshow(image2[:,:,::-1])
# plt.show()


# kp = sift.detect(gray, None)



plt.figure(figsize = (15,15))
plt.imshow(image3[:,:,::-1])
plt.title("Top 100 Features Matching")
plt.show()
```

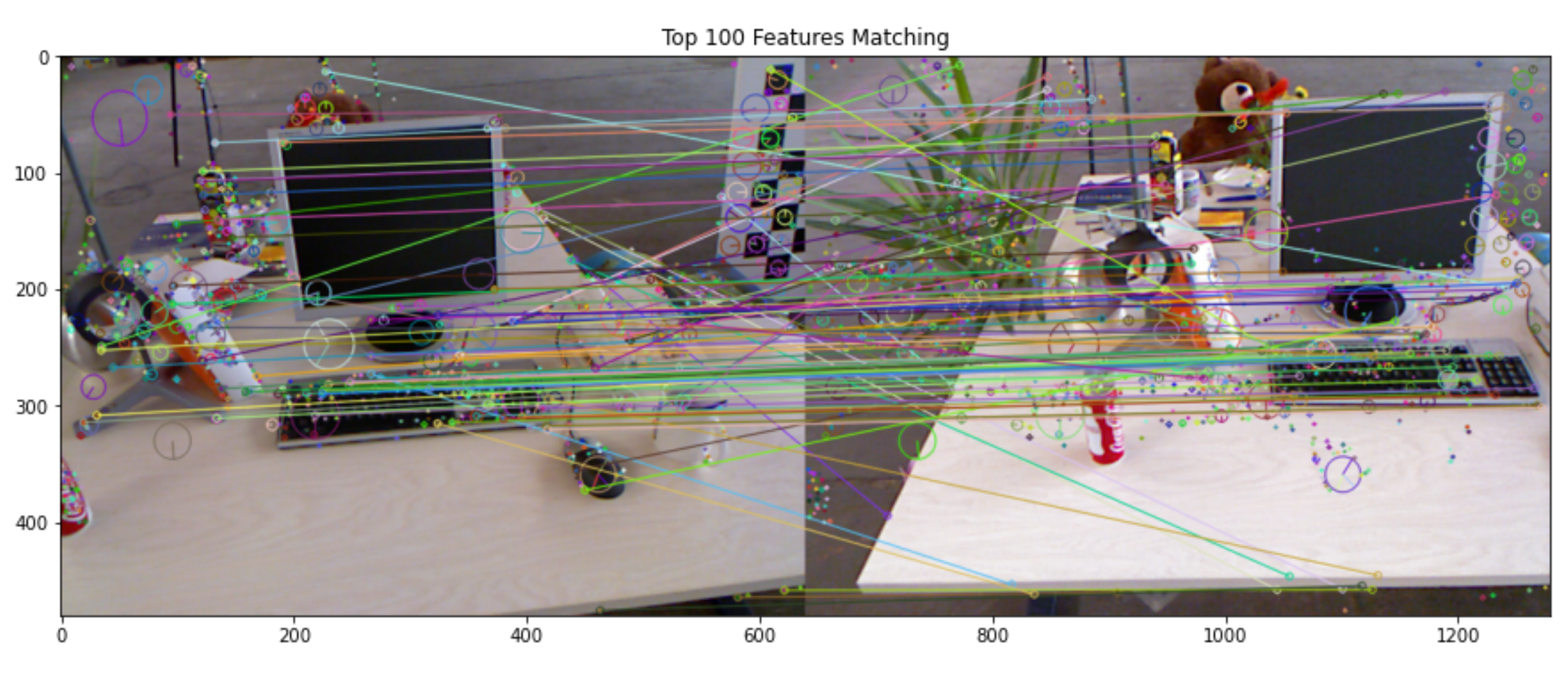
Keypoints of Image 1



Keypoints of Image 2



Top 100 Features Matching



Explanation (1 Pts):

We can analyse from the above image that SIFT is able to match some of the features in two images. Because image2 does not contain all the items in image1 and is shifted to left, some features are matched incorrectly. Also, we can notice the SIFT algorithm is immune to illumination and rotation changes by looking at how keypoints near the "Keyboard" and the "table" matches perfectly from image1 to image2.

2. Motion Estimation with Optical Flow (10 Pts)

Real time object tracking is a challenging task. Optical flow gives valuable information about the object movement. And moving object detection and tracking is an evolving research field due to its wide applications in traffic surveillance, 3D reconstruction, motion analysis (human and non-human), activity recognition, medical imaging, etc.

Theory (5 Pts):

Hint: Each 0.5 points expects one argument/step/etc.

1. Explain the difference between motion field and optical flow with example. (1.5 Pts)
2. Analyze if the three assumptions for optical flow tracking hold true when you would track bubbles in a glass of carbonated water. (1.5 Pts)
3. Explain the aperture problem in optical flow. How can one solve this problem by estimating the global geometric structure of the scene? (1 Pts)
4. The proposed Lucas-Kanade method has an issue with large movement. Why? How can you deal with large movement in the image? A conceptual explanation is sufficient. (1 Pts)

Solution:

1. **Motion Field vs Optical Flow**
   - Motion Field: It is defined as the image velocity of a point that is moving in a scene. It is a representation of 3D Motion as it is projected on a Image Camera. We are often unsure if we can calculate the velocity or not so we use Optical Flow.
   - Optical Flow: Optical Flow is the motion of brightness pattern in the image. We compare the brightness pattern in the corresponding images
   - Example: Let's consider a uniform surface sphere with a light source. If our sphere is moving but there is no change in light intensity then there exist only the motion field and no optical flow. Whereas if we consider a case where the light source is moving a sphere as stationary object then there exists only optical flow but no motion field. We can also thing of a Barber Pole illusion where optical flow is not equal to the motion field
2. **Tracking Bubbles**
   - Brightness consistency: This assumption will hold as there is no change in intensity for a give ntracking problem.
   - Spatial Coherence: Since every bubble will only be moving in the upwards direction, this assumption will hold.
   - Temporal persistence: This assumption will hold as bubbles will have small motion upwards over time, which will be easily tracked.
3. **Aperture Problem**
   - Problem: The aperture problem refers to the limitation that when an object in motion is viewed through an aperture, the direction of motion of the small feature of the structure maybe ambiguous.
   - Solution: The aperture problem can be solved by estimating global contraint of smootheness of the scene, this is Horn Schunch method. It tries to minimize the distortion in the flow over the image and prefers the smoothness factor of the image.

1. **Lucas-Kanade large Image Movement**
   - Problem: For the larger movement (larger than a pixel) in Lukas Kanade approach, we get anti aliasing problem, that is, distorted and jagged output. With a large displacement, there's a good possiblity to miss the similar intensity pixels within the vicinity of the previous position resulting to an abrupt change in the intensity due to large shift, causing distortion.
   - Solution: This can be overcome by coarse to fine estimation. Here, we solve the problem at the smaller level and then use that to solve problem at the larger level. The resolution is keep on halving until less than or equal to 1 pixel resolution is got. We now apply the Kanade algo at this level. We now use this flow to warp the image in the next higher resolution and so on.

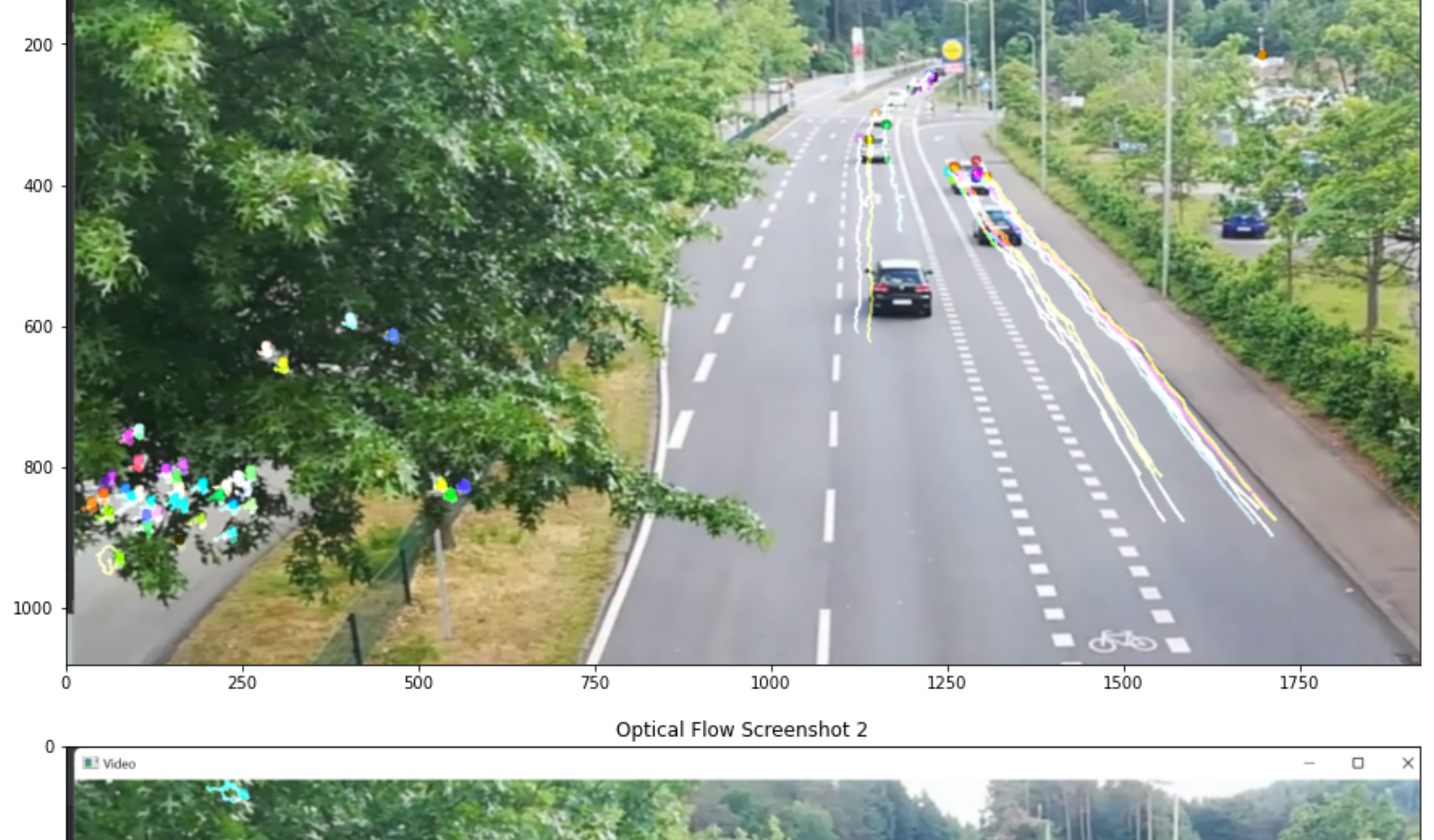
Programming Task (4 Pts)

Configure your OpenCV and read the provided mp4 video. Apply the Lucas-Kanade Optical Flow on it. The result should look like the given example.

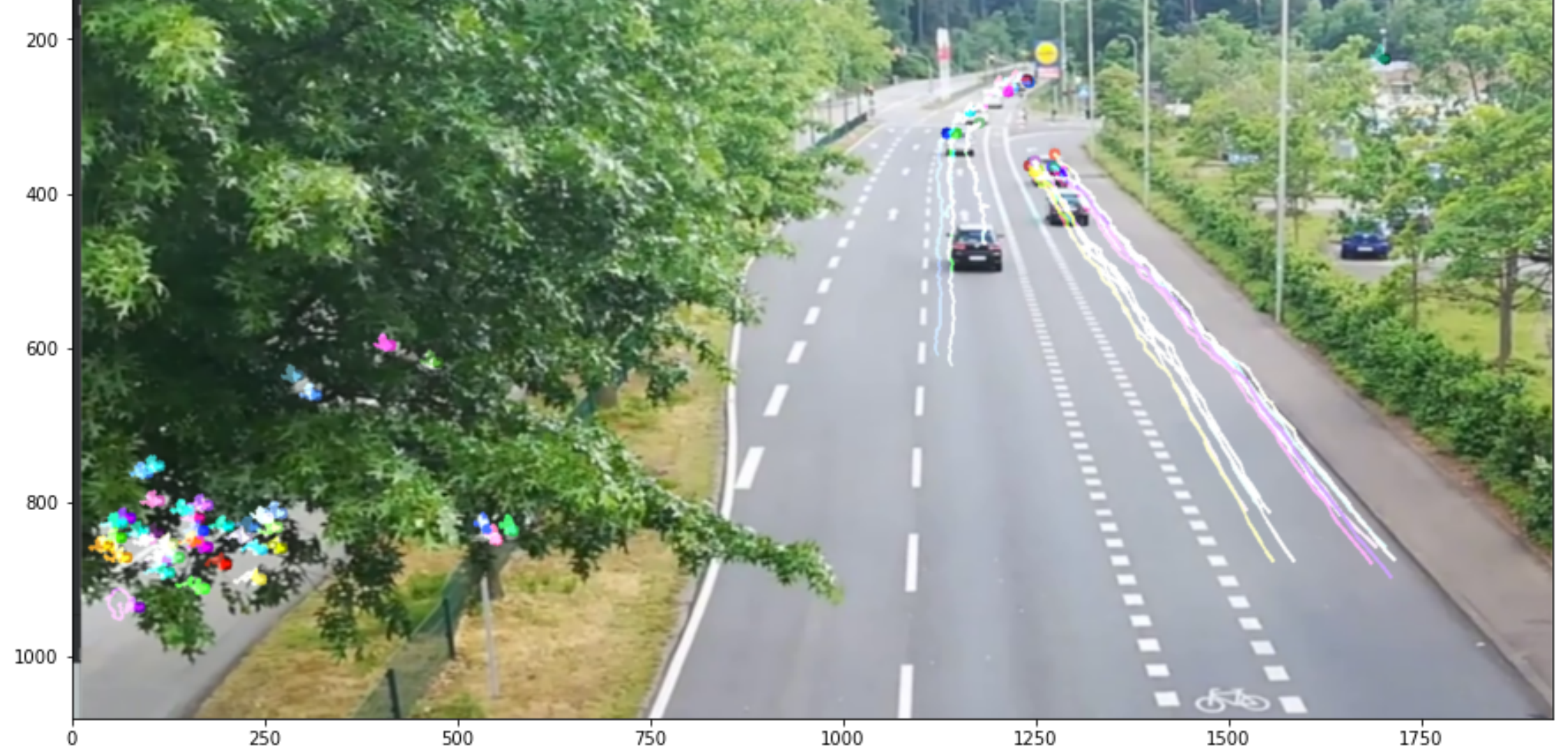
Video: https://cloud.dfki.de/owncloud/index.php/s/THLirfoB6SYTetn/download?path=&files=tracking.mp4

```
In [3]:
cap = cv2.VideoCapture('C:/Users/Dhruv/Videos/tracking.mp4')

fp = dict(maxCorners=100, qualityLevel=0.3, minDistance=7, blockSize=7)

lkp = dict(winSize=(15, 15), maxLevel=3, criteria=(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))

color = np.random.randint(0, 255, (100, 3))

_, prev_frame = cap.read()
old_gray = cv2.cvtColor(prev_frame, cv2.COLOR_BGR2GRAY)

# Shi-Tomashi Corner  Detection
kp1 = cv2.goodFeaturesToTrack(old_gray, mask=None, **fp)

mask = np.zeros_like(prev_frame)

while (1):

    try:
        _, frame = cap.read()
        frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # calculate the optical flows
        kp2, st, err = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray, kp1, None, **lkp)

        # Find the good points
        good_new = kp2[st == 1]
        good_old = kp1[st == 1]

        # place the trackers on the video
        for i, (new, old) in enumerate(zip(good_new, good_old)):
            a, b = new.ravel()
            c, d = old.ravel()
            mask = cv2.line(mask, (int(a), int(b)), (int(c), int(d)), color[i].tolist(), 2)

            frame = cv2.circle(frame, (int(a), int(b)), 5, color[i].tolist(), -1)
        img = cv2.add(frame, mask)

        # Showing the video
        cv2.imshow('Video', img)
        old_gray = frame_gray.copy()
        kp1 = good_new.reshape(-1, 1, 2)

        k = cv2.waitKey(30) & 0xff
        if k == 27:
            break
    except:
        break


# Added the screenshots because we are not able to play video in Jupyter Notebook

ss1=cv2.imread("C:/Users/Dhruv/OneDrive/Pictures/Screenshots/Screenshot (6).png")
plt.figure(figsize = (15,15))
plt.title("Optical Flow Screenshot 1")
plt.imshow(ss1[:,:,::-1])
plt.show()

ss2=cv2.imread("C:/Users/Dhruv/OneDrive/Pictures/Screenshots/Screenshot (7).png")
plt.figure(figsize = (15,15))
plt.title("Optical Flow Screenshot 2")
plt.imshow(ss2[:,:,::-1])
plt.show()
```

Optical Flow Screenshot 1



Optical Flow Screenshot 2



Explanation (1 Pts):

Lucas-Kanade Optical Flow works well on the given mp4 file. We are able to capture the objects with motion (in our case automobiles and tree on the left). It can be clearly seen from the image that the object without any motion does not have any optical flow for exp: street lights, trees on the right, road etc.