Visual Odometry for Localization in Autonomous Driving

Welcome to the assignment for Module 2: Visual Features - Detection, Description and Matching. In this assignment, you will practice using the material you have learned to estimate an autonomous vehicle trajectory by images taken with a monocular camera set up on the vehicle.

In this assignment, you will:

- Extract features from the photographs taken with a camera setup on the vehicle.
- Use the extracted features to find matches between the features in different photographs.
- Use the found matches to estimate the camera motion between subsequent photographs.
- Use the estimated camera motion to build the vehicle trajectory.

For most exercises, you are provided with a suggested outline. You are encouraged to diverge from the outline if you think there is a better, more efficient way to solve a problem.

You are only allowed to use the packages loaded bellow and the custom functions explained in the notebook. Run the cell bellow to import the required packages:

In [1]:
```
import numpy as np
import cv2
import sys
import os
from matplotlib import pyplot as plt
from m2bk import *

%matplotlib inline
%load_ext autoreload
%autoreload 2

np.random.seed(1)
np.set_printoptions(threshold=sys.maxsize)
```

0 - Loading and Visualizing the Data

We provide you with a convenient dataset handler class to read and iterate through samples taken from the CARLA simulator. Run the following code to create a dataset handler object.

In [2]:
```
dataset_handler = DatasetHandler()
```

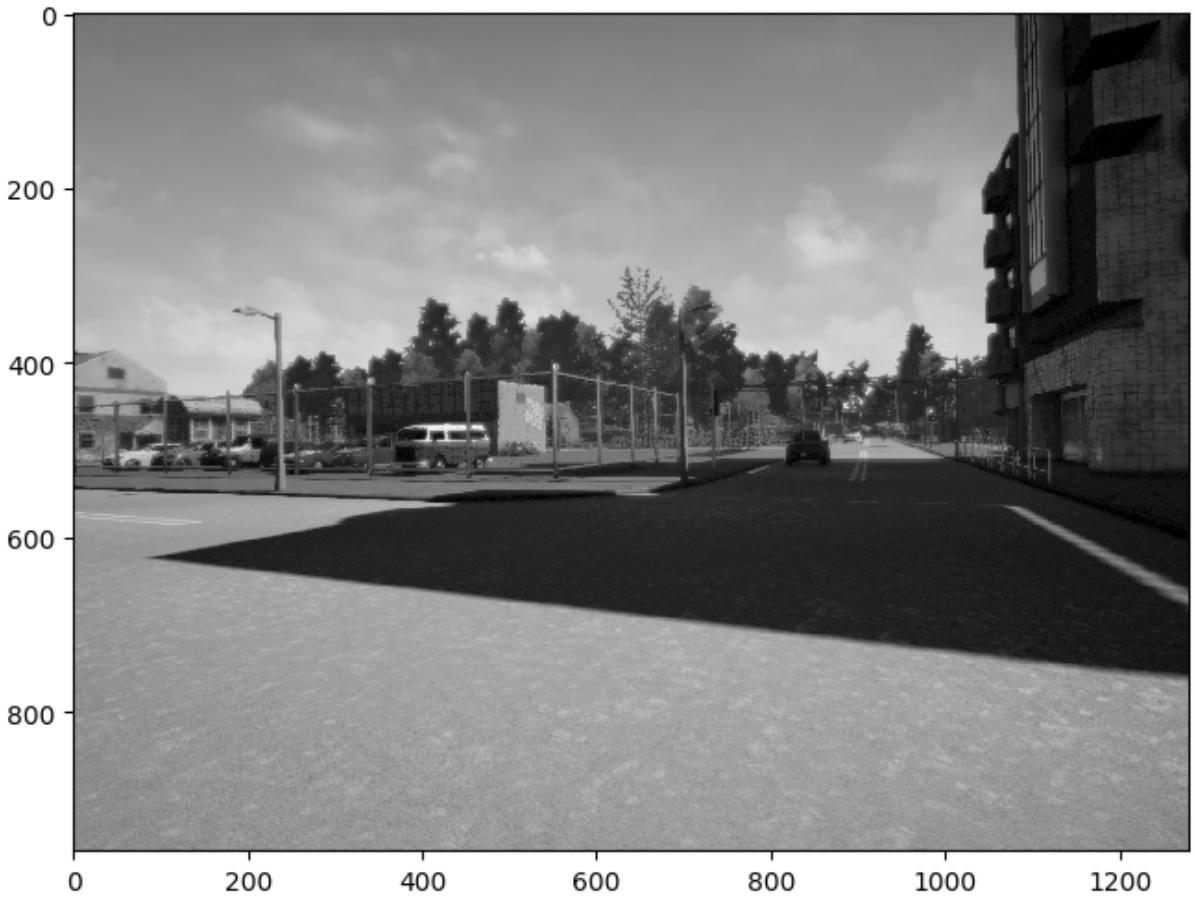
The dataset handler contains 52 data frames. Each frame contains an RGB image and a depth map taken with a setup on the vehicle and a grayscale version of the RGB image which will be used for computation. Furthermore, camera calibration matrix K is also provided in the dataset handler.

Upon creation of the dataset handler object, all the frames will be automatically read and loaded. The frame content can be accessed by using `images`, `images_rgb`, `depth_maps` attributes of the dataset handler object along with the index of the requested frame. See how to access the images (grayscale), rgb images (3-channel color), depth maps and camera calibration matrix in the example below.

Note (Depth Maps): Maximum depth distance is 1000. This value of depth shows that the selected pixel is at least 1000m (1km) far from the camera, however the exact distance of this pixel from the camera is unknown. Having this kind of points in further trajectory estimation might affect the trajectory precision.

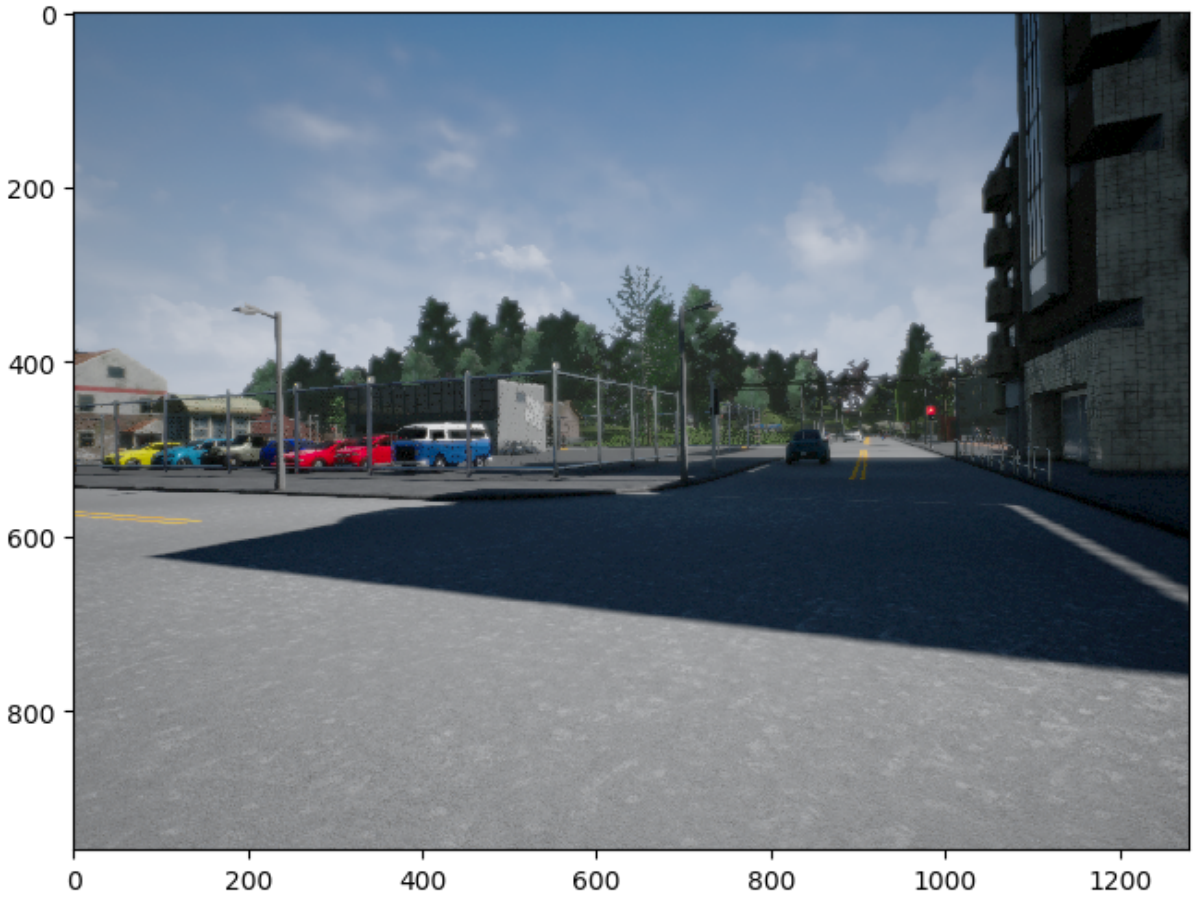
In [3]:
```
image = dataset_handler.images[0]

plt.figure(figsize=(8, 6), dpi=100)
plt.imshow(image, cmap='gray')
```

Out[3]: `<matplotlib.image.AxesImage at 0x7f499bf64828>`



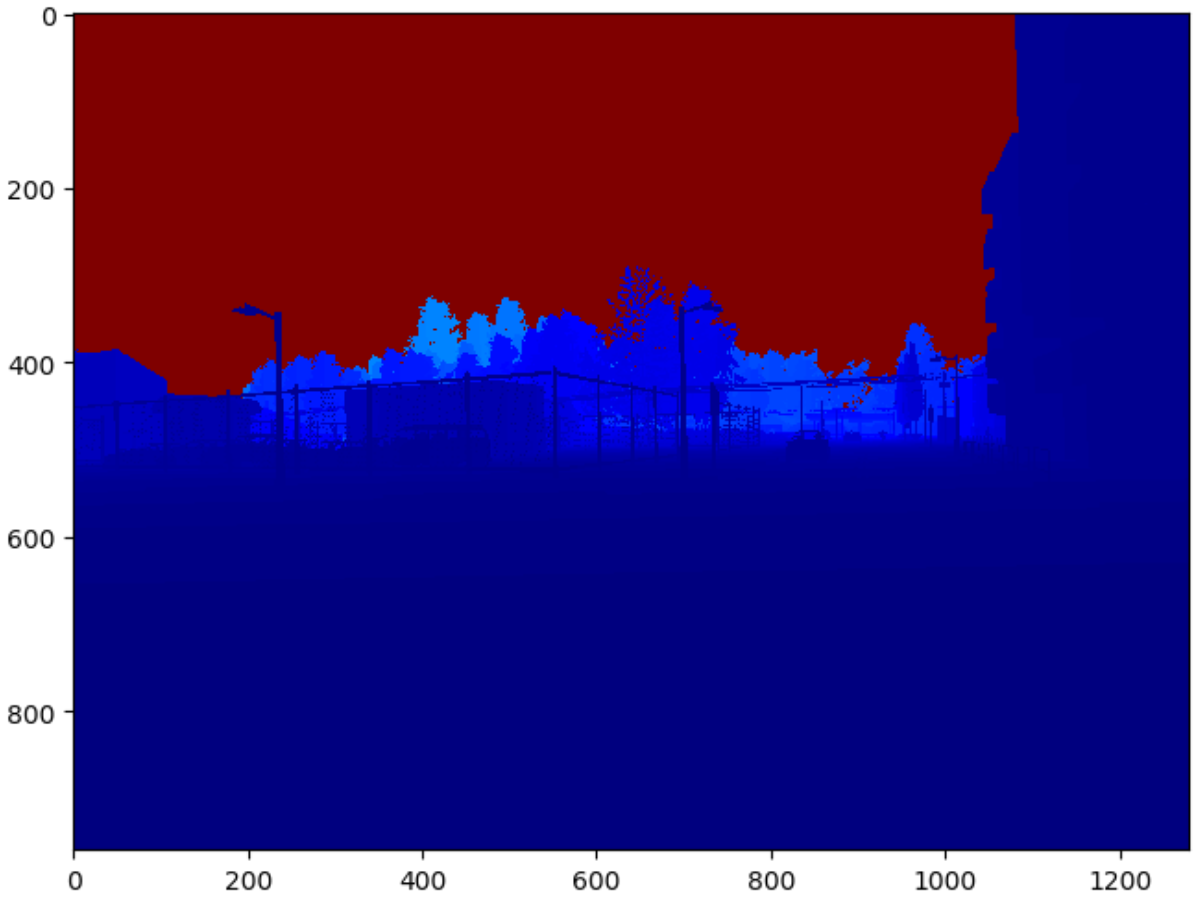
In [4]:
```
image_rgb = dataset_handler.images_rgb[0]

plt.figure(figsize=(8, 6), dpi=100)
plt.imshow(image_rgb)
```

Out[4]: <matplotlib.image.AxesImage at 0x7f499b86b978>



In [5]:
```
i = 0
depth = dataset_handler.depth_maps[i]

plt.figure(figsize=(8, 6), dpi=100)
plt.imshow(depth, cmap='jet')
```

Out[5]: <matplotlib.image.AxesImage at 0x7f499b3da4e0>



In [6]:
```
print("Depth map shape: {0}".format(depth.shape))

v, u = depth.shape
depth_val = depth[v-1, u-1]
print("Depth value of the very bottom-right pixel of depth map {0}
is {1:0.3f}".format(i, depth_val))
```

```
Depth map shape: (960, 1280)
Depth value of the very bottom-right pixel of depth map 0 is 1.862
```

In [7]:
```
dataset_handler.k
```

Out[7]:
```
array([[640.,   0., 640.],
       [  0., 480., 480.],
       [  0.,   0.,   1.]], dtype=float32)
```

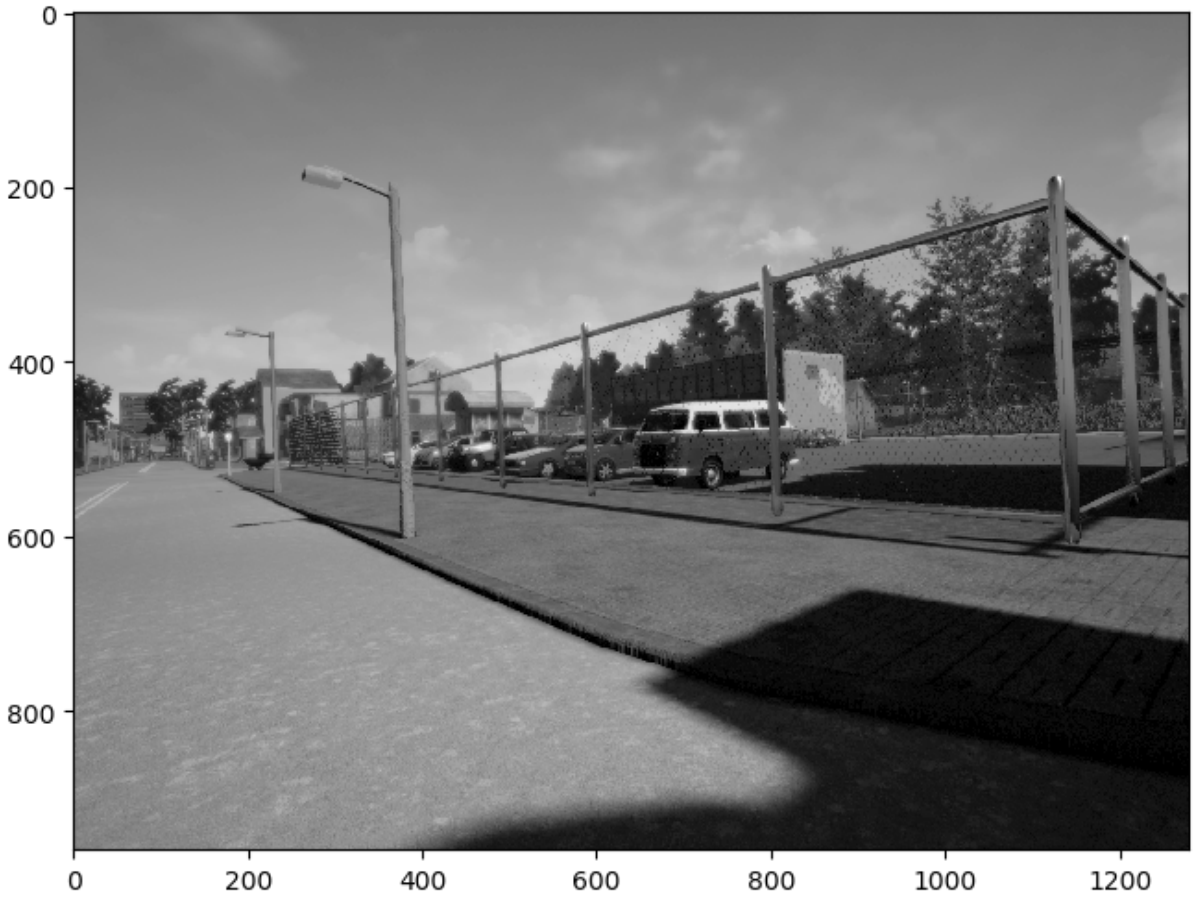
In order to access an arbitrary frame use image index, as shown in the examples below. Make sure the indexes are within the number of frames in the dataset. The number of frames in the dataset can be accessed with num_frames attribute.

In [8]:
```
# Number of frames in the dataset
print(dataset_handler.num_frames)
```

```
52
```

In [9]:
```
i = 30
image = dataset_handler.images[i]

plt.figure(figsize=(8, 6), dpi=100)
plt.imshow(image, cmap='gray')
```

Out[9]: <matplotlib.image.AxesImage at 0x7f499b1b2b00>



1 - Feature Extraction

1.1 - Extracting Features from an Image

Task: Implement feature extraction from a single image. You can use any feature descriptor of your choice covered in the lectures, ORB for example.

Note 1: Make sure you understand the structure of the keypoint descriptor object, this will be very useful for your further tasks. You might find OpenCV: Keypoint Class Description (https://docs.opencv.org/3.4.3/d2/d29/classcv_1_1KeyPoint.html) handy.

Note 2: Make sure you understand the image coordinate system, namely the origin location and axis directions.

Note 3: We provide you with a function to visualise the features detected. Run the last 2 cells in section 1.1 to view.

Optional: Try to extract features with different descriptors such as SIFT, ORB, SURF and BRIEF. You can also try using detectors such as Harris corners or FAST and pairing them with a descriptor. Lastly, try changing parameters of the algorithms. Do you see the difference in various approaches? You might find this link useful: OpenCV:Feature Detection and Description (https://docs.opencv.org/3.4.3/db/d27/tutorial_py_table_of_contents_feature2d.html).

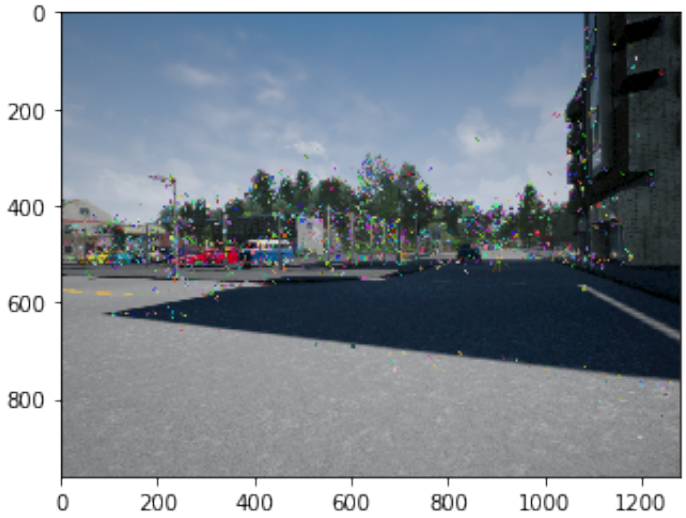
In [10]:
```
def extract_features(image):
    """
    Find keypoints and descriptors for the image

    Arguments:
    image -- a grayscale image

    Returns:
    kp -- list of the extracted keypoints (features) in an image
    des -- list of the keypoint descriptors in an image
    """
    ### START CODE HERE ###

    # Initiate SIFT detector
    surf = cv.xfeatures2d.SURF_create(400)

    # find the keypoints and descriptors with SIFT
    kp, des = surf.detectAndCompute(image,None)

    ### END CODE HERE ###

    return kp, des
```

In [11]:
```
i = 0
image = dataset_handler.images[i]
kp, des = extract_features(image)
print("Number of features detected in frame {0}: {1}\n".format(i, l
en(kp)))

print("Coordinates of the first keypoint in frame {0}: {1}".format(
i, str(kp[0].pt)))
```

```
Number of features detected in frame 0: 1580

Coordinates of the first keypoint in frame 0: (417.722900390625, 48
4.4891052246094)
```

In [12]:
```
def visualize_features(image, kp):
    """
    Visualize extracted features in the image

    Arguments:
    image -- a grayscale image
    kp -- list of the extracted keypoints

    Returns:
    """
    display = cv2.drawKeypoints(image, kp, None)
    plt.imshow(display)
```

In [13]:
```
# Optional: visualizing and experimenting with various feature desc
riptors
i = 0
image = dataset_handler.images_rgb[i]

visualize_features(image, kp)
```



1.2 - Extracting Features from Each Image in the Dataset

Task: Implement feature extraction for each image in the dataset with the function you wrote in the above section.

Note: If you do not remember how to pass functions as arguments, make sure to brush up on this topic. This Passing Functions as Arguments (https://www.coursera.org/lecture/program-code/passing-functions-as-arguments-hnmqD) might be helpful.

In [14]:
```
def extract_features_dataset(images, extract_features_function):
    """
    Find keypoints and descriptors for each image in the dataset

    Arguments:
    images -- a list of grayscale images
    extract_features_function -- a function which finds features (k
eypoints and descriptors) for an image

    Returns:
    kp_list -- a list of keypoints for each image in images
    des_list -- a list of descriptors for each image in images

    """
    kp_list = []
    des_list = []

    ### START CODE HERE ###
    for i in images:
        kp, des = extract_features_function(i)
        kp_list.append(kp)
        des_list.append(des)


    ### END CODE HERE ###

    return kp_list, des_list
```

In [15]:
```
images = dataset_handler.images
kp_list, des_list = extract_features_dataset(images, extract_featur
es)

i = 0
print("Number of features detected in frame {0}: {1}".format(i, len
(kp_list[i])))
print("Coordinates of the first keypoint in frame {0}: {1}\n".forma
t(i, str(kp_list[i][0].pt)))

# Remember that the length of the returned by dataset_handler lists
should be the same as the length of the image array
print("Length of images array: {0}".format(len(images)))
```

```
Number of features detected in frame 0: 1580
Coordinates of the first keypoint in frame 0: (417.722900390625, 48
4.4891052246094)

Length of images array: 52
```

2 - Feature Matching

Next step after extracting the features in each image is matching the features from the subsequent frames. This is what is needed to be done in this section.

2.1 - Matching Features from a Pair of Subsequent Frames

Task: Implement feature matching for a pair of images. You can use any feature matching algorithm of your choice covered in the lectures, Brute Force Matching or FLANN based Matching for example.

Optional 1: Implement match filtering by thresholding the distance between the best matches. This might be useful for improving your overall trajectory estimation results. Recall that you have an option of specifying the number best matches to be returned by the matcher.

We have provided a visualization of the found matches. Do all the matches look legitimate to you? Do you think match filtering can improve the situation?

In [30]:
```
def match_features(des1, des2):
    """
    Match features from two images

    Arguments:
    des1 -- list of the keypoint descriptors in the first image
    des2 -- list of the keypoint descriptors in the second image

    Returns:
    match -- list of matched features from two images. Each match[i
] is k or less matches for the same query descriptor
    """
    ### START CODE HERE ###

    bf = cv2.BFMatcher_create()
    match = bf.knnMatch(des1,des2, k=2)

    ### END CODE HERE ###

    return match
```

In [31]:
```
i = 0
des1 = des_list[i]
des2 = des_list[i+1]

match = match_features(des1, des2)
print("Number of features matched in frames {0} and {1}: {2}".forma
t(i, i+1, len(match)))

# Remember that a matcher finds the best matches for EACH descripto
r from a query set
```

```
Number of features matched in frames 0 and 1: 1580
```

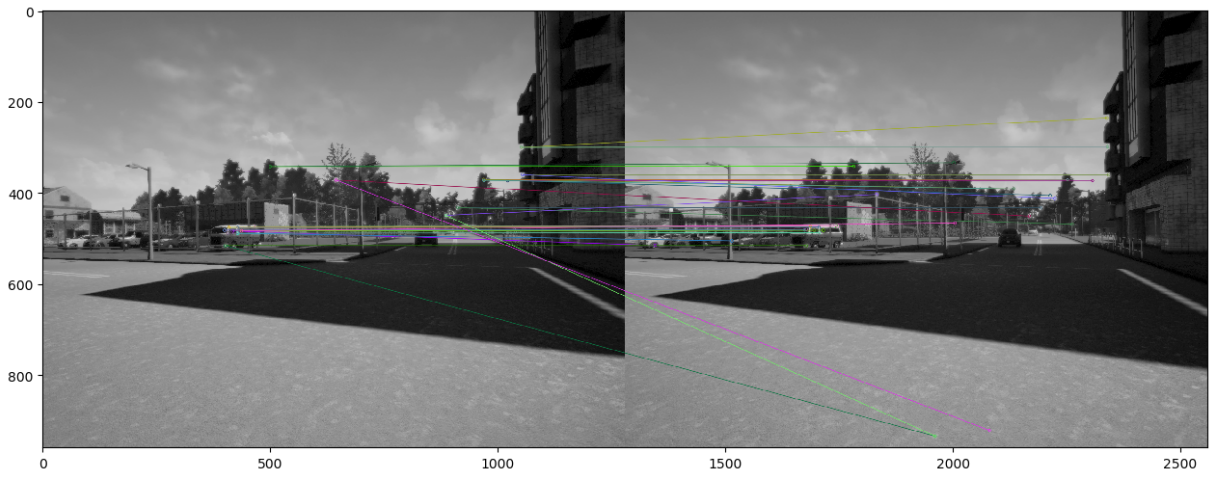
In [32]:
```
# Optional
def filter_matches_distance(match, dist_threshold):
    """
    Filter matched features from two images by distance between the
best matches

    Arguments:
    match -- list of matched features from two images
    dist_threshold -- maximum allowed relative distance between the
best matches, (0.0, 1.0)

    Returns:
    filtered_match -- list of good matches, satisfying the distance
threshold
    """
    filtered_match = []

    ### START CODE HERE ###
    filtered_match = []
    for m,n in match:
        if m.distance < dist_threshold * n.distance:
            filtered_match.append([m])

    ### END CODE HERE ###

    return filtered_match
```

In [33]:
```
# Optional
i = 0
des1 = des_list[i]
des2 = des_list[i+1]
match = match_features(des1, des2)

dist_threshold = 0.6
filtered_match = filter_matches_distance(match, dist_threshold)

print("Number of features matched in frames {0} and {1} after filte
ring by distance: {2}".format(i, i+1, len(filtered_match)))
```

```
Number of features matched in frames 0 and 1 after filtering by dis
tance: 533
```

In [42]:
```
def visualize_matches(image1, kp1, image2, kp2, match):
    """
    Visualize corresponding matches in two images

    Arguments:
    image1 -- the first image in a matched image pair
    kp1 -- list of the keypoints in the first image
    image2 -- the second image in a matched image pair
    kp2 -- list of the keypoints in the second image
    match -- list of matched features from the pair of images

    Returns:
    image_matches -- an image showing the corresponding matches on
both image1 and image2 or None if you don't use this function
    """

    # image_matches = cv2.drawMatches(image1,kp1,image2,kp2,match)
    # Always error, so i replay it with drawMatchesKnn
    # reference:https://stackoverflow.com/questions/31631352/typeer
ror-required-argument-outimg-pos-6-not-found

    image_matches = cv2.drawMatchesKnn(image1,kp1,image2,kp2,match,
None,flags=2)

    plt.figure(figsize=(16, 6), dpi=100)
    plt.imshow(image_matches)
```

In [43]:
```
# Visualize n first matches, set n to None to view all matches
# set filtering to True if using match filtering, otherwise set to
False
n = 20
filtering = False

i = 0
image1 = dataset_handler.images[i]
image2 = dataset_handler.images[i+1]

kp1 = kp_list[i]
kp2 = kp_list[i+1]

des1 = des_list[i]
des2 = des_list[i+1]

match = match_features(des1, des2)
if filtering:
    dist_threshold = 0.6
    match = filter_matches_distance(match, dist_threshold)

image_matches = visualize_matches(image1, kp1, image2, kp2, match[:
n])
```



2.2 - Matching Features in Each Subsequent Image Pair in the Dataset

Task: Implement feature matching for each subsequent image pair in the dataset with the function you wrote in the above section.

Optional: Implement match filtering by thresholding the distance for each subsequent image pair in the dataset with the function you wrote in the above section.

In [46]:
```
def match_features_dataset(des_list, match_features):
    """
    Match features for each subsequent image pair in the dataset

    Arguments:
    des_list -- a list of descriptors for each image in the dataset
    match_features -- a function which maches features between a pa
ir of images

    Returns:
    matches -- list of matches for each subsequent image pair in th
e dataset.
                Each matches[i] is a list of matched features from i
mages i and i + 1

    """
    matches = []

    ### START CODE HERE ###
    i = 0

    for i in range(len(des_list) - 1):
        des1 = des_list[i]
        des2 = des_list[i+1]
        match = match_features(des1, des2)
        matches.append(match)

    ### END CODE HERE ###

    return matches
```

In [47]:
```
matches = match_features_dataset(des_list, match_features)

i = 0
print("Number of features matched in frames {0} and {1}: {2}".forma
t(i, i+1, len(matches[i])))
```

```
Number of features matched in frames 0 and 1: 1580
```

In [48]:
```
# Optional
def filter_matches_dataset(filter_matches_distance, matches, dist_t
hreshold):
    """
    Filter matched features by distance for each subsequent image p
air in the dataset

    Arguments:
    filter_matches_distance -- a function which filters matched fea
tures from two images by distance between the best matches
    matches -- list of matches for each subsequent image pair in th
e dataset.
                Each matches[i] is a list of matched features from i
mages i and i + 1
    dist_threshold -- maximum allowed relative distance between the
best matches, (0.0, 1.0)

    Returns:
    filtered_matches -- list of good matches for each subsequent im
age pair in the dataset.
                        Each matches[i] is a list of good matches,
satisfying the distance threshold

    """
    filtered_matches = []

    ### START CODE HERE ###
    for match in matches:
        filtered_match = filter_matches_distance(match, dist_thresh
old)
        filtered_matches.append(filtered_match)

    ### END CODE HERE ###

    return filtered_matches
```

In [49]:
```
# Optional
dist_threshold = 0.6

filtered_matches = filter_matches_dataset(filter_matches_distance,
matches, dist_threshold)

if len(filtered_matches) > 0:

    # Make sure that this variable is set to True if you want to us
e filtered matches further in your assignment
    is_main_filtered_m = False
    if is_main_filtered_m:
        matches = filtered_matches

    i = 0
    print("Number of filtered matches in frames {0} and {1}: {2}".f
ormat(i, i+1, len(filtered_matches[i])))
```

```
Number of filtered matches in frames 0 and 1: 533
```

3 - Trajectory Estimation

At this point you have everything to perform visual odometry for the autonomous vehicle. In this section you will incrementally estimate the pose of the vehicle by examining the changes that motion induces on the images of its onboard camera.

3.1 - Estimating Camera Motion between a Pair of Images

Task: Implement camera motion estimation from a pair of images. You can use the motion estimation algorithm covered in the lecture materials, namely Perspective-n-Point (PnP), as well as Essential Matrix Decomposition.

- If you decide to use PnP, you will need depth maps of frame and they are provided with the dataset handler. Check out Section 0 of this assignment to recall how to access them if you need. As this method has been covered in the course, review the lecture materials if need be.
- If you decide to use Essential Matrix Decomposition, more information about this method can be found in Wikipedia: Determining R and t from E (https://en.wikipedia.org/wiki/Essential_matrix).

More information on both approaches implementation can be found in OpenCV: Camera Calibration and 3D Reconstruction (https://docs.opencv.org/3.4.3/d9/d0c/group__calib3d.html). Specifically, you might be interested in *Detailed Description* section of OpenCV: Camera Calibration and 3D Reconstruction (https://docs.opencv.org/3.4.3/d9/d0c/group__calib3d.html) as it explains the connection between the 3D world coordinate system and the 2D image coordinate system.

Optional: Implement camera motion estimation with PnP, PnP with RANSAC and Essential Matrix Decomposition. Check out how filtering matches by distance changes estimated camera movement. Do you see the difference in various approaches?

In [50]:
```
def estimate_motion(match, kp1, kp2, k, depth1=None):
    """
    Estimate camera motion from a pair of subsequent image frames

    Arguments:
    match -- list of matched features from the pair of images
    kp1 -- list of the keypoints in the first image
    kp2 -- list of the keypoints in the second image
    k -- camera calibration matrix

    Optional arguments:
    depth1 -- a depth map of the first frame. This argument is not
needed if you use Essential Matrix Decomposition

    Returns:
    rmat -- recovered 3x3 rotation numpy matrix
```

```
    tvec -- recovered 3x1 translation numpy vector
    image1_points -- a list of selected match coordinates in the fi
rst image. image1_points[i] = [u, v], where u and v are
                     coordinates of the i-th match in the image coo
rdinate system
    image2_points -- a list of selected match coordinates in the se
cond image. image1_points[i] = [u, v], where u and v are
                     coordinates of the i-th match in the image coo
rdinate system

    """
    rmat = np.eye(3)
    tvec = np.zeros((3, 1))
    image1_points = []
    image2_points = []

    ### START CODE HERE ###
    for m in match:
        m = m[0]
        query_idx = m.queryIdx
        train_idx = m.trainIdx

        # get first img matched keypoints
        p1_x, p1_y = kp1[query_idx].pt
        image2_points.append([p1_x, p1_y])

        # get second img matched keypoints
        p2_x, p2_y = kp2[train_idx].pt
        image1_points.append([p2_x, p2_y])

    # essential matrix
    E, mask = cv2.findEssentialMat(np.array(image1_points), np.arra
y(image2_points), dataset_handler.k)
    _, R, t, mask = cv2.recoverPose(E, np.array(image1_points), np.
array(image2_points), dataset_handler.k)

    rmat = R
    tvec = t


    ### END CODE HERE ###

    return rmat, tvec, image1_points, image2_points
```

In [51]:
```
i = 0
match = matches[i]
kp1 = kp_list[i]
kp2 = kp_list[i+1]
k = dataset_handler.k
depth = dataset_handler.depth_maps[i]

rmat, tvec, image1_points, image2_points = estimate_motion(match, k
p1, kp2, k, depth1=depth)

print("Estimated rotation:\n {0}".format(rmat))
print("Estimated translation:\n {0}".format(tvec))
```

```
Estimated rotation:
 [[ 9.99986714e-01  5.15249212e-03  1.50503128e-04]
 [-5.15256835e-03  9.99986595e-01  5.10548203e-04]
 [-1.47870515e-04 -5.11316897e-04  9.99999858e-01]]
Estimated translation:
 [[-0.15853136]
 [ 0.03644506]
 [ 0.98668109]]
```

Expected Output Format:

Make sure that your estimated rotation matrix and translation vector are in the same format as the given initial values

```
rmat = np.eye(3)
tvec = np.zeros((3, 1))

print("Initial rotation:\n {0}".format(rmat))
print("Initial translation:\n {0}".format(tvec))
```

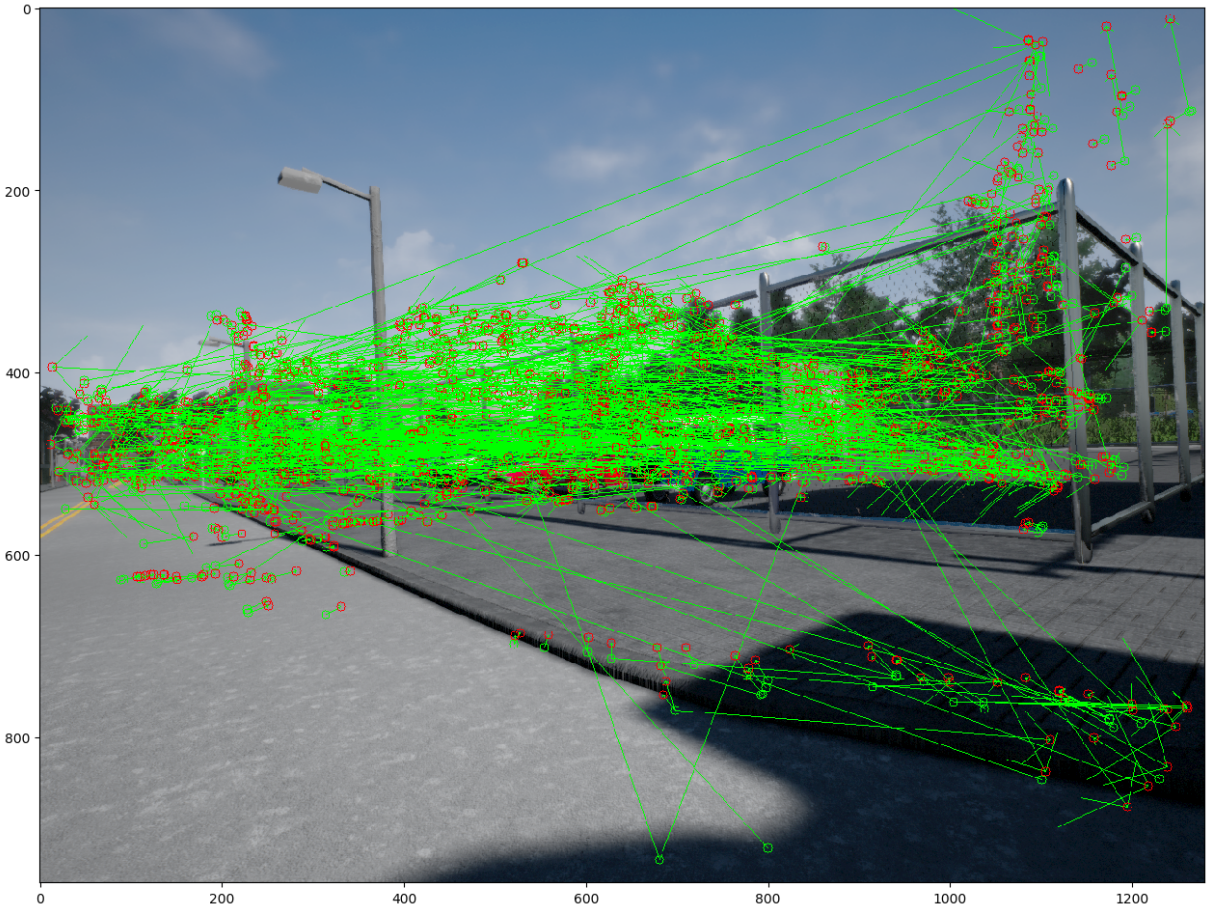
```
Initial rotation:
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
Initial translation:
 [[0.]
 [0.]
 [0.]]
```

Camera Movement Visualization: You can use `visualize_camera_movement` that is provided to you. This function visualizes final image matches from an image pair connected with an arrow corresponding to direction of camera movement (when `is_show_img_after_mov = False`). The function description:

```
Arguments:
image1 -- the first image in a matched image pair (RGB or grayscale)
image1_points -- a list of selected match coordinates in the first imag
e. image1_points[i] = [x, y], where x and y are
                coordinates of the i-th match in the image coordinate
system
image2 -- the second image in a matched image pair (RGB or grayscale)
image2_points -- a list of selected match coordinates in the second ima
ge. image1_points[i] = [x, y], where x and y are
                coordinates of the i-th match in the image coordinate
system
is_show_img_after_mov -- a boolean variable, controling the output (rea
d image_move description for more info)

Returns:
image_move -- an image with the visualization. When is_show_img_after_m
ov=False then the image points from both images are visualized on the f
irst image. Otherwise, the image points from the second image only are
visualized on the second image
```

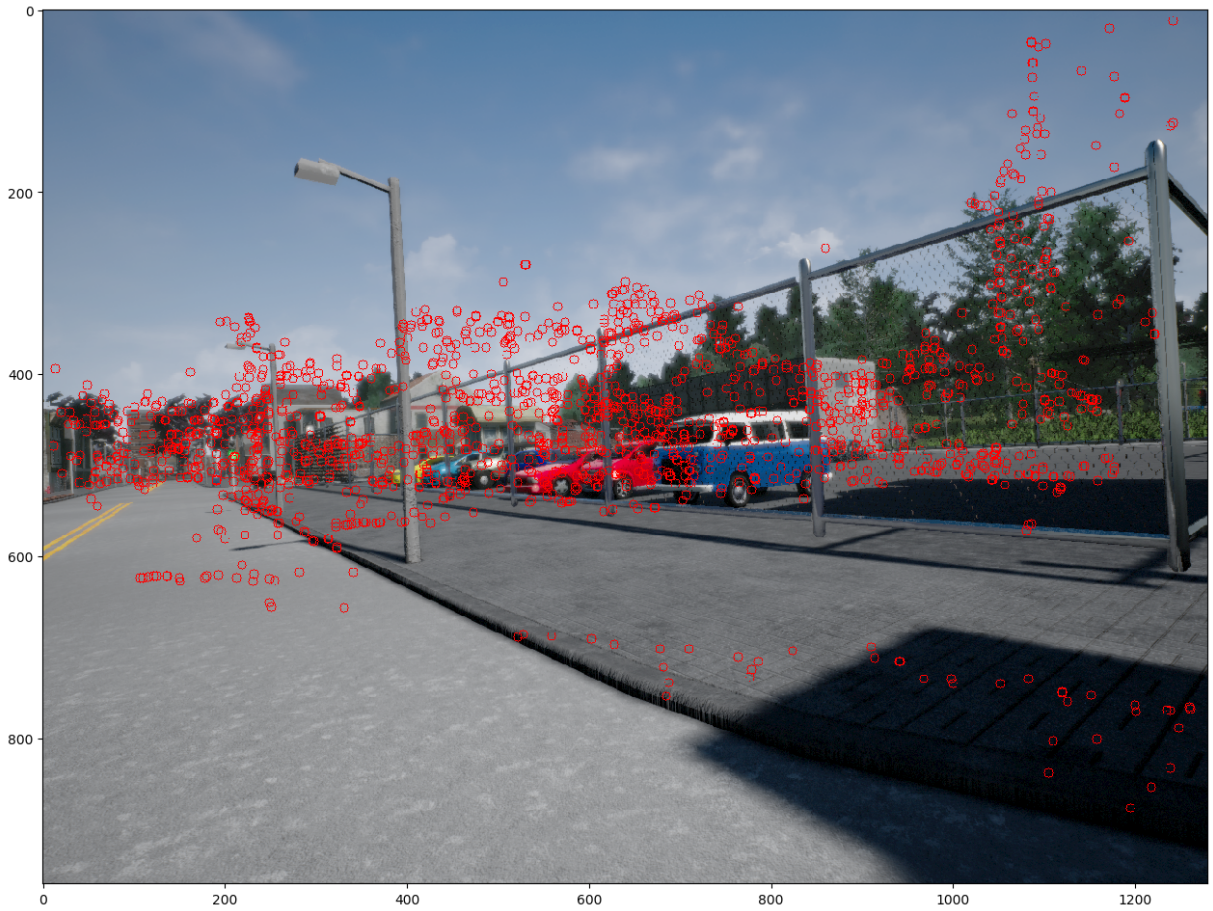
In [52]:
```
i=30
image1  = dataset_handler.images_rgb[i]
image2 = dataset_handler.images_rgb[i + 1]

image_move = visualize_camera_movement(image1, image1_points, image
2, image2_points)
plt.figure(figsize=(16, 12), dpi=100)
plt.imshow(image_move)
```

Out[52]: <matplotlib.image.AxesImage at 0x7f499ad86390>



In [53]:
```
image_move = visualize_camera_movement(image1, image1_points, image
2, image2_points, is_show_img_after_move=True)
plt.figure(figsize=(16, 12), dpi=100)
plt.imshow(image_move)
# These visualizations might be helpful for understanding the quali
ty of image points selected for the camera motion estimation
```

Out[53]: <matplotlib.image.AxesImage at 0x7f49945020f0>



3.2 - Camera Trajectory Estimation

Task: Implement camera trajectory estimation with visual odometry. More specifically, implement camera motion estimation for each subsequent image pair in the dataset with the function you wrote in the above section.

Note: Do not forget that the image pairs are not independent one to each other. i-th and (i + 1)-th image pairs have an image in common

In [58]:
```
def estimate_trajectory(estimate_motion, matches, kp_list, k, depth
_maps=[]):
    """
```

```
    Estimate complete camera trajectory from subsequent image pairs

    Arguments:
    estimate_motion -- a function which estimates camera motion fro
m a pair of subsequent image frames
    matches -- list of matches for each subsequent image pair in th
e dataset.
                Each matches[i] is a list of matched features from i
mages i and i + 1
    des_list -- a list of keypoints for each image in the dataset
    k -- camera calibration matrix

    Optional arguments:
    depth_maps -- a list of depth maps for each frame. This argumen
t is not needed if you use Essential Matrix Decomposition

    Returns:
    trajectory -- a 3xlen numpy array of the camera locations, wher
e len is the lenght of the list of images and
                  trajectory[:, i] is a 3x1 numpy vector, such as:

                  trajectory[:, i][0] - is X coordinate of the i-th
location
                  trajectory[:, i][1] - is Y coordinate of the i-th
location
                  trajectory[:, i][2] - is Z coordinate of the i-th
location

                  * Consider that the origin of your trajectory cor
dinate system is located at the camera position
                  when the first image (the one with index 0) was t
aken. The first camera location (index = 0) is geven
                  at the initialization of this function

    """
    trajectory = np.zeros((3, 1))

    ### START CODE HERE ###

    trajectory = [np.array([0, 0, 0])]
    P = np.eye(4)

    for i in range(len(matches)):
        match = matches[i]
        kp1 = kp_list[i]
        kp2 = kp_list[i+1]
        depth = depth_maps[i]

        rmat, tvec, image1_points, image2_points = estimate_motion(
match, kp1, kp2, k, depth)
```

```
            R = rmat
            t = np.array([tvec[0,0],tvec[1,0],tvec[2,0]])

            P_new = np.eye(4)
            P_new[0:3,0:3] = R.T
            P_new[0:3,3] = (-R.T).dot(t)
            P = P.dot(P_new)

            trajectory.append(P[:3,3])

        trajectory = np.array(trajectory).T
        trajectory[2,:] = -1*trajectory[2,:]


        ### END CODE HERE ###

        return trajectory
```

In [59]:
```
depth_maps = dataset_handler.depth_maps
trajectory = estimate_trajectory(estimate_motion, matches, kp_list,
k, depth_maps=depth_maps)

i = 1
print("Camera location in point {0} is: \n {1}\n".format(i, traject
ory[:, [i]]))

# Remember that the length of the returned by trajectory should be
the same as the length of the image array
print("Length of trajectory: {0}".format(trajectory.shape[1]))
```

```
Camera location in point 1 is:
 [[0.07258554]
 [0.03968014]
 [0.99657254]]

Length of trajectory: 52
```

Expected Output:

```
Camera location in point i is:
 [[locXi]
 [locYi]
 [locZi]]
```

In this output: locXi, locYi, locZi are the coordinates of the corresponding i-th camera location

4 - Submission:

Evaluation of this assignment is based on the estimated trajectory from the output of the cell below. Please run the cell bellow, then copy its output to the provided yaml file for submission on the programming assignment page.

Expected Submission Format:

```
Trajectory X:
 [[  0.          locX1        locX2        ...   ]]
Trajectory Y:
 [[  0.          locY1        locY2        ...   ]]
Trajectory Z:
 [[  0.          locZ1        locZ2        ...   ]]
```

In this output: locX1, locY1, locZ1; locX2, locY2, locZ2; ... are the coordinates of the corresponding 1st, 2nd and etc. camera locations

In [60]:
```
# Note: Make sure to uncomment the below line if you modified the o
riginal data in any ways
#dataset_handler = DatasetHandler()


# Part 1. Features Extraction
images = dataset_handler.images
kp_list, des_list = extract_features_dataset(images, extract_featur
es)


# Part II. Feature Matching
matches = match_features_dataset(des_list, match_features)

# Set to True if you want to use filtered matches or False otherwis
e
is_main_filtered_m = True
if is_main_filtered_m:
    dist_threshold = 0.75
    filtered_matches = filter_matches_dataset(filter_matches_distan
ce, matches, dist_threshold)
    matches = filtered_matches


# Part III. Trajectory Estimation
depth_maps = dataset_handler.depth_maps
trajectory = estimate_trajectory(estimate_motion, matches, kp_list,
k, depth_maps=depth_maps)


#!!! Make sure you don't modify the output in any way
# Print Submission Info
print("Trajectory X:\n {0}".format(trajectory[0,:].reshape((1,-1)))
)
print("Trajectory Y:\n {0}".format(trajectory[1,:].reshape((1,-1)))
)
print("Trajectory Z:\n {0}".format(trajectory[2,:].reshape((1,-1)))
)
```

```
Trajectory X:
 [[  0.          0.07258554  -0.01866342   0.04356773   0.07452287
    0.18422339   0.18101417   0.32716163   0.45120467   0.41416194
    0.31946978   0.27051919   0.36781204   0.44143243   0.58498294
    0.7440996    0.73537808   0.78396942   0.68951995   0.53506241
    0.36728056   0.26794613   0.29321121   0.18600511   0.04549436
   -0.09904935  -0.40620278  -0.76708534  -1.09714553  -1.42077381
   -1.69693657  -1.98821226  -2.22658393  -2.3755066   -2.44789434
   -2.817463    -3.21821526  -3.71639875  -4.21600181  -4.66996778
   -5.25494469  -5.86212167  -6.52599343  -7.30418351  -8.13672766
   -9.08328478  -9.97241815 -10.92657194 -11.85117244 -12.79983744
  -13.7585372  -14.70353401]]
Trajectory Y:
 [[ 0.          0.03968014  0.03160547 -0.04752961 -0.18293798 -0.2
2711792
  -0.2611457  -0.19815648 -0.2776846  -0.3075322  -0.33420589 -0.37
820284
  -0.4462568  -0.3925564  -0.38584391 -0.38432007 -0.40456053 -0.47
329385
  -0.48943935 -0.4551496  -0.42947718 -0.40997742 -0.40973008 -0.42
147976
  -0.39288524 -0.35978793 -0.28152535 -0.30958781 -0.30636276 -0.28
491917
  -0.35484603 -0.3756919  -0.37967115 -0.39005993 -0.42053418 -0.47
152681
  -0.48003593 -0.50329065 -0.51215733 -0.51141127 -0.47615742 -0.45
51339
  -0.32920182 -0.16997863  0.01641478  0.24880172  0.43787636  0.63
618195
   0.82745373  1.00853292  1.15527898  1.32809596]]
Trajectory Z:
 [[-0.          0.99657254  1.99236791  2.98728747  3.97759366  4.9
70576
   5.96999174  6.95724716  7.94633189  8.94519973  9.94034892 10.93
818062
  11.93110698 12.92694648 13.91656671 14.90382532 15.90358242 16.90
003343
  17.89543215 18.88283637 19.86832618 20.8631892  21.86286996 22.85
703735
  23.8467035  24.83564823 25.7840847  26.71627367 27.66022805 28.60
616933
  29.56473303 30.52114503 31.49231089 32.48110516 33.47801605 34.40
581924
  35.32196619 36.18872592 37.054935   37.9459537  38.7562371  39.55
052554
  40.28769268 40.8952025  41.41686114 41.6405259  42.05729039 42.28
148991
  42.61092735 42.87024383 43.11388369 43.39158012]]
```

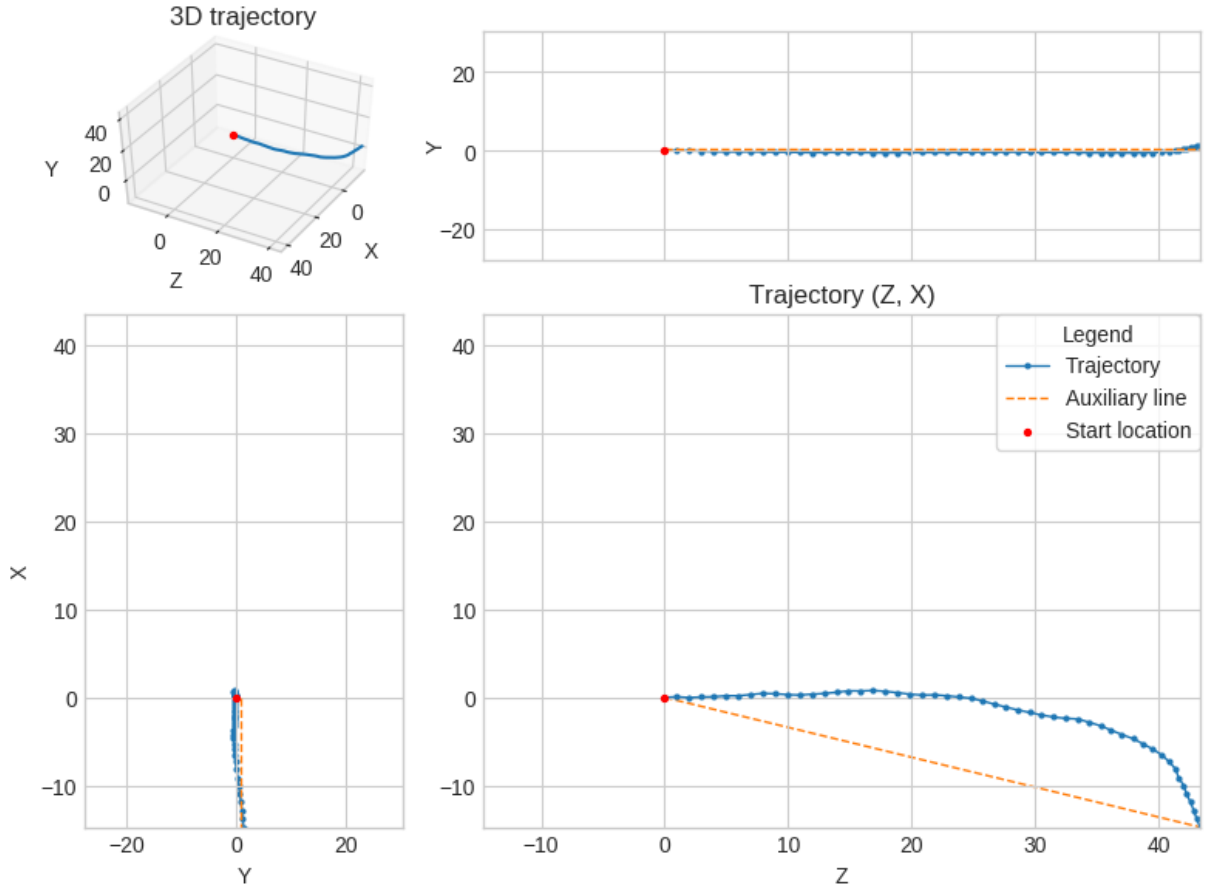
Visualize your Results

Important:

1) Make sure your results visualization is appealing before submitting your results. You might want to download this project dataset and check whether the trajectory that you have estimated is consistent to the one that you see from the dataset frames.

2) Assure that your trajectory axis directions follow the ones in *Detailed Description* section of OpenCV: Camera Calibration and 3D Reconstruction (https://docs.opencv.org/3.4.3/d9/d0c/group__calib3d.html).

```
In [61]: visualize_trajectory(trajectory)
```



Congrats on finishing this assignment!