Monocular Visual Odometry for an In-Vehicle Ego Motion

Tobias Hoelzer
Aug 2016

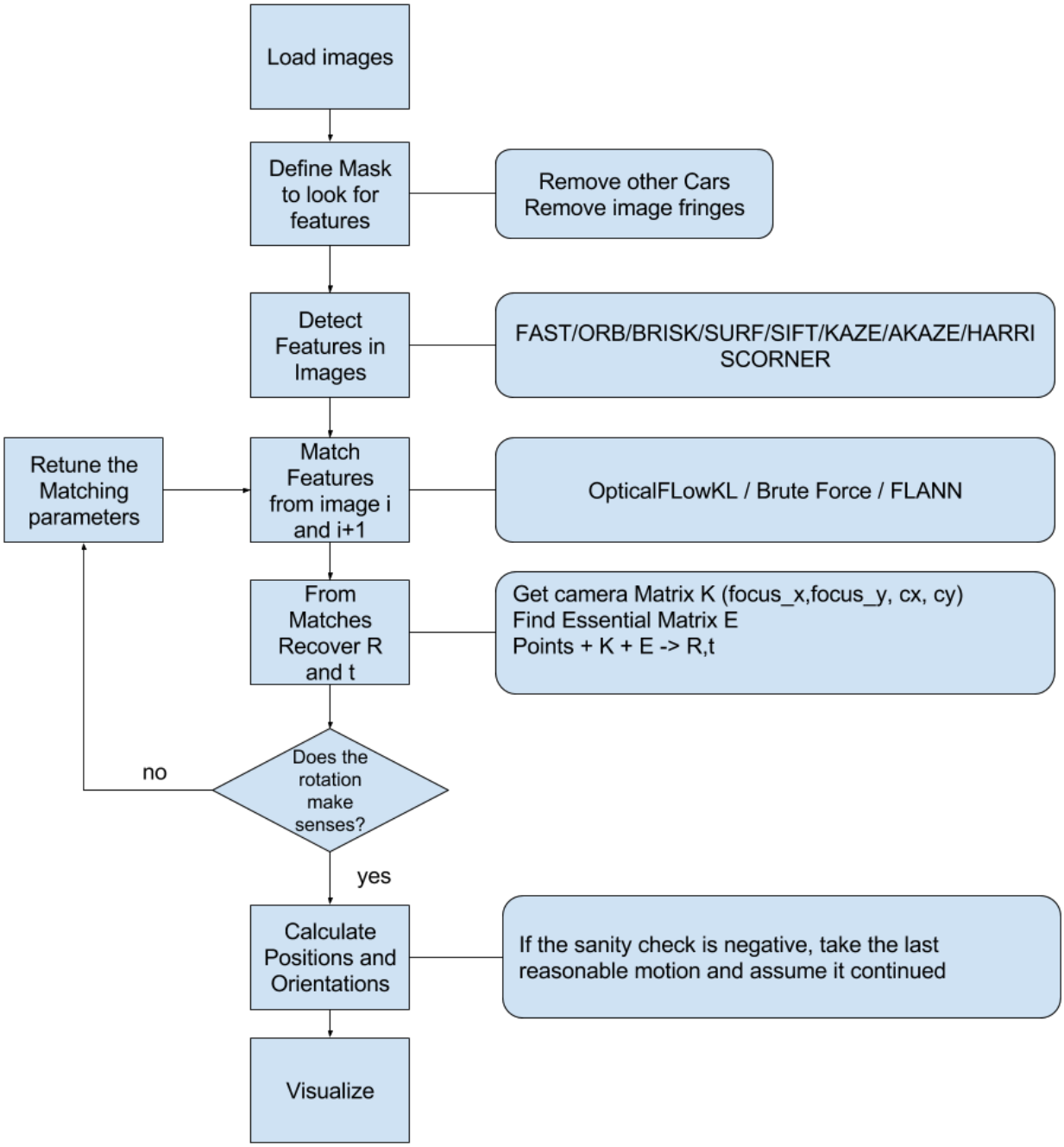
Problem Statement

Input/Output
We are given a stream of images from a monocular in-vehicle dashboard camera. For every pair of images, we want to find the translation vector **t** and the rotation matrix **R** describing the movement of the vehicle.

Restrictions
With a monocular camera, the translation vector can only be known up to a scale factor. (Taking 2 cameras or the accelerometer data or combine with aerial footage would solve this. In fact aerial footage might be the most robust for industry scale application)
For an accurate calculation, the camera intrinsics need to be known.

Ressources
http://research.microsoft.com/en-us/um/people/awf/bmvc03invariance/Torr03.pdf
http://answers.opencv.org/question/66839/units-of-rotation-and-translation-from-essential-matrix/
https://cseweb.ucsd.edu/classes/sp03/cse252/MaSKS_Ch5.pdf
http://research.microsoft.com/en-us/um/people/awf/bmvc03invariance/Torr03.pdf
http://answers.opencv.org/question/66839/units-of-rotation-and-translation-from-essential-matrix/
https://cseweb.ucsd.edu/classes/sp03/cse252/MaSKS_Ch5.pdf
http://isit.u-clermont1.fr/~ab/Classes/DIKU-3DCV2/Handouts/Lecture16.pdf
https://avisingh599.github.io/vision/monocular-vo/
http://www.cse.iitm.ac.in/~vplab/courses/CV_DIP/PDF/GEOM+PROJ+STEREO.pdf
http://imagine.enpc.fr/publications/papers/ARCHEOFOSS.pdf
http://drops.dagstuhl.de/opus/volltexte/2011/3096/pdf/7.pdf
http://www.robots.ox.ac.uk/~gk/PTAM/
https://static-content.springer.com/lookinside/art%3A10.1007%2FBF00133556/000.png
https://github.com/uzh-rpg/rpg_svo [this is the best solutioni found for high frame rates]

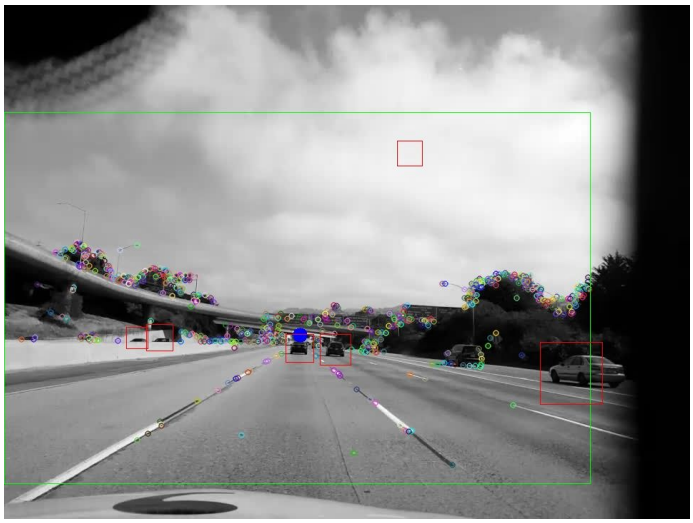
Algorithm


Load images
Define Mask to look for features
Remove other Cars
Remove image fringes
Detect Features in Images
FAST/ORB/BRISK/SURF/SIFT/KAZE/AKAZE/HARRISCORNER
Retune the Matching parameters
Match Features from image i and i+1
OpticalFLowKL / Brute Force / FLANN
From Matches Recover R and t
Get camera Matrix K (focus_x,focus_y, cx, cy)
Find Essential Matrix E
Points + K + E -> R,t
Does the rotation make senses?
no
yes
Calculate Positions and Orientations
If the sanity check is negative, take the last reasonable motion and assume it continued
Visualize

Load images

We use grayscale images for the feature detection algorithm

Define Mask to look for features



We don't want the hood or the mirror to be part of the image because they move with our coordinate system. The field of view (FOV) is shown in green. We also want to remove other cars because they are moving with our coordinate system. They are marked in red.

We use a pretrained Haar cascade to recognize cars. It's not amazing but a good start. Interestingly, the results are not much worse without removing other cars.

Detect Features in Images

There are three kinds of Feature Detectors specifically distinguished by their output:

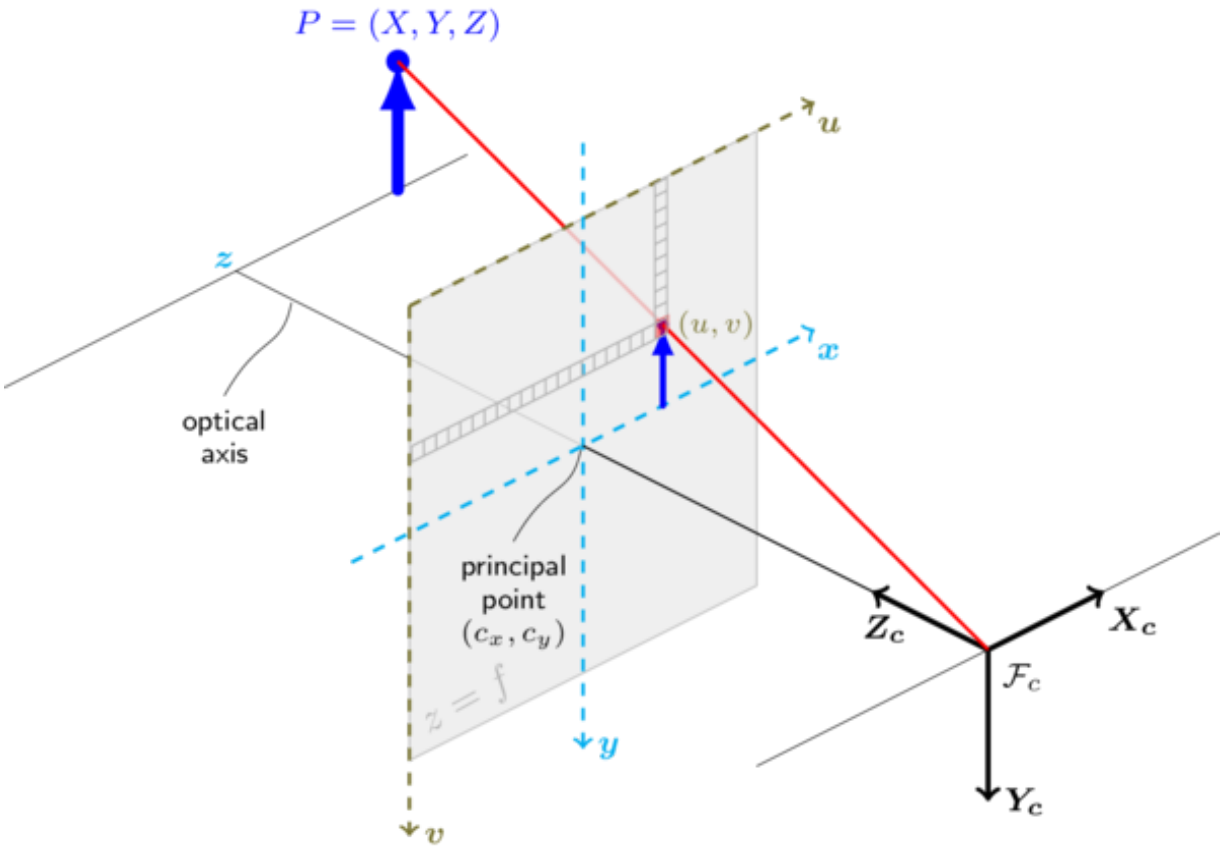
Output

1. Only keypoints [e.g. FAST, cv2.goodFeaturesToTrack]
2. Keypoints + Binary Descriptors [e.g. ORB]
3. Keypoints + Descriptors [e.g. SURF/SIFT]

The best turned out to be the FAST algorithm using corner detection using the hyperparameters FAST_hyper = [40,45,30,45,50] for the 5 different cars.

Match Feature from 2 Images

We put 0_0.jpg in the R^3 coordinate origin (0,0,0), with a direction into (0,0,1).
See picture below for our coordinate system in the pinhole camera model.

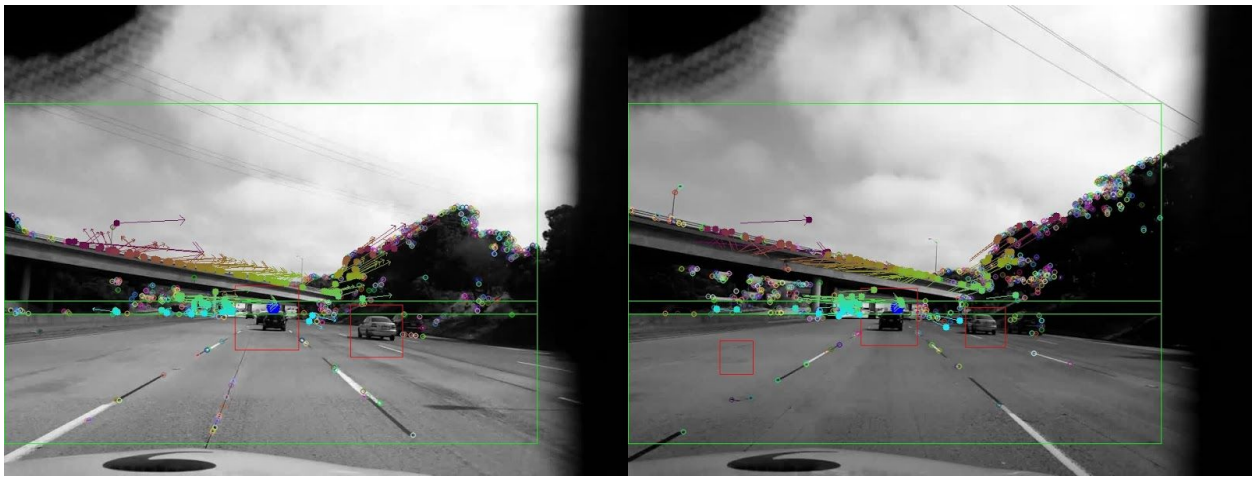


If it's the first image of a series, we compare to the 0_0.jpg image, otherwise we compare consecutive images in the series.
There are 3 different algorithms:

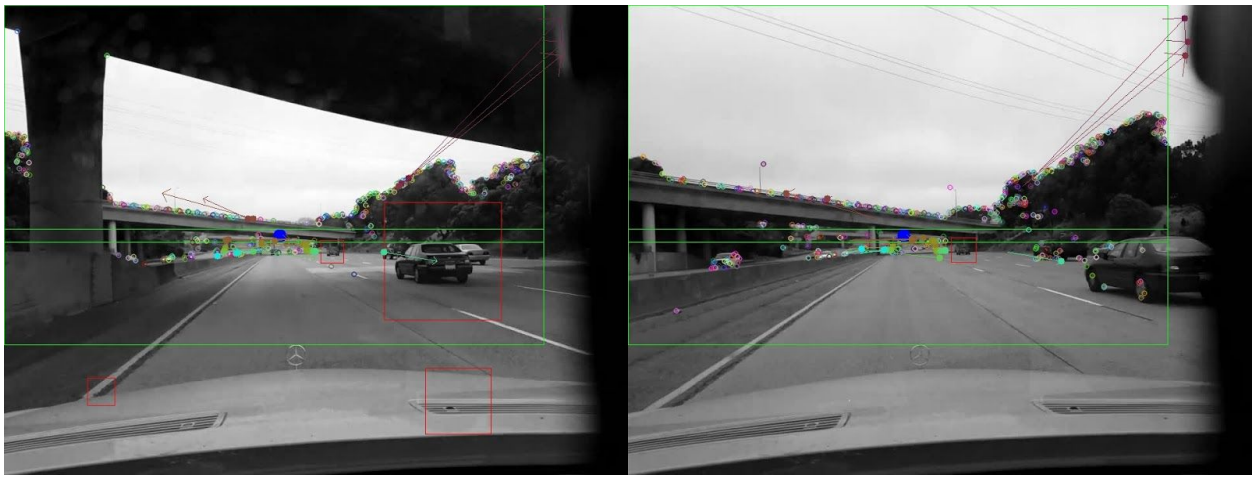
OpticalFlowLK, FLANN, Brute Force feature matching:

1. Optical Flow is cool because it only uses the keypoints and looks for similar ones in the other image, so it works with FAST algorithm.
2. Brute Force does a nearest neighbor matching of the keypoints according to the descriptors optained from algorithms like ORB (use cv2.NORM_HAMMING) or SIFT/SURF (use cv2.NORM_L1).
3. FLANN didn't work for me for some reason, uses knn matching of keypoints according to their descriptors.

We end up using OpticalFlowLK with lk_params = dict(winSize = (21,21), maxLevel = 3, criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 30, 0.01), minEigThreshold=0.001))



This is an example of a successful feature matching. The algorithm finds the corners at the top of the bridge and how they move from one image to the other shown in arrows.



This is an example of a failed matching of keypoints. This one is very hard because the bridge in the left image looks very similar to the bridge in the second image.

Solutions:

1. Sample images more densely
2. Match the keypoints not to consecutive images - instead match with aerial photographs

Recover R and t from Matches

1
First we need the camera matrix **K.**

K = [[f, s, cx]
[0, f, cy]
[0, 0, 1]]

With the focal length of the lens **f** , the skew **s** (0 in modern cameras), and the principal point **(cx,cy)** as show in the image above.

We estimated cx,cy by checking the Vanishing Point in each image. We estimated f by assuming a Oneplus 3 camera with an IMX 298 sensor. The technical documentation tells us that f = 29 mm, px size is 1.12 micrometers with a sensor size of 5.28 mm x 4.30 mm.

->

f [px] = $\frac{width[px]*f[mm]}{chipwidth[mm]} = \frac{960\ px * 29\ mm]}{5.28\ mm} = 5260\ px$

2
Then we calculate the Essential Matrix **E**

$E = R[t]_x$, $[t]_x$ being the matrix representation of the crossproduct with t.
It is important to use the RANSAC algorithm to detect outliers in the keypoint matches.

3

Recover R and t from E

There are 2 possible solutions for R and 2 possible solutions for t.
We therefore need to be careful with the output

Do a sanity check on R and t

R is the rotation matrix in Radians.
We calculate the Euler angles around x y and z axes and check if they make sense.

We allow a maximum x rotation (up/down hill) in between two images of 15°, a maximum y rotation (left / right) of 30° and a maximum z rotation (spin) of 15°.

If the rotations are bigger than there was obviously a mistake in the matching algorithm, so we repeat the algorithm with a bigger windowsize and a deeper triangulation. Explicitly, we increase the windowsize by (3,3) and maxlevel by 1.
We allow maximal 3 repetitions of the opticalflow matching algorithm.

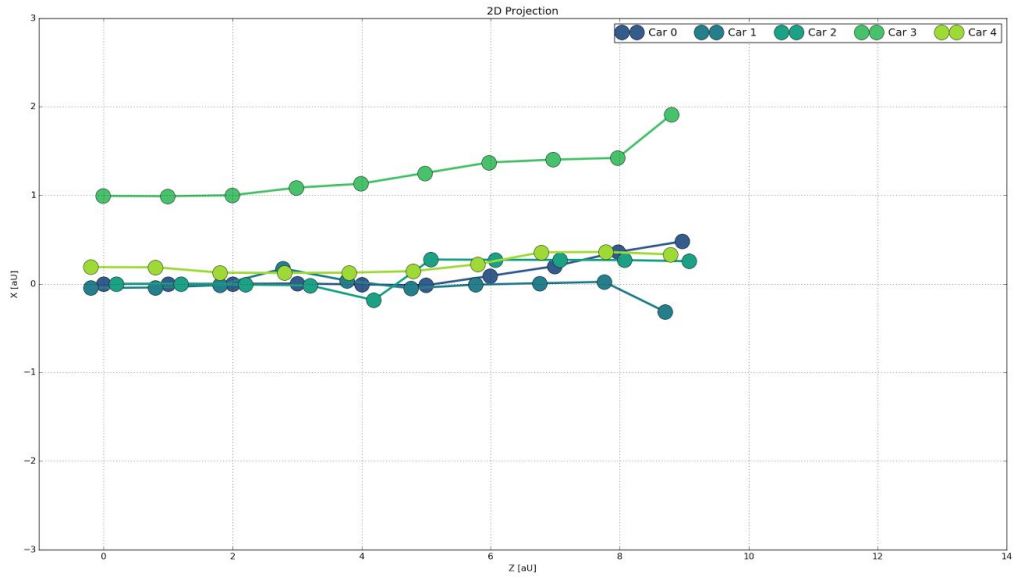
Calculate the actual positions from R and t

$$t_i = t_{i-1} + R_{i-1} * t_i$$
$$R_i = R_{i-1} * R_i$$

If the sanity check is negative, this means after 3 repetitions, no useful R and t could be found. We then assume no sudden changes in movement and use the previous ones. This happened 1 time for cars 1,2 and 3 with the current algorithm specifications.

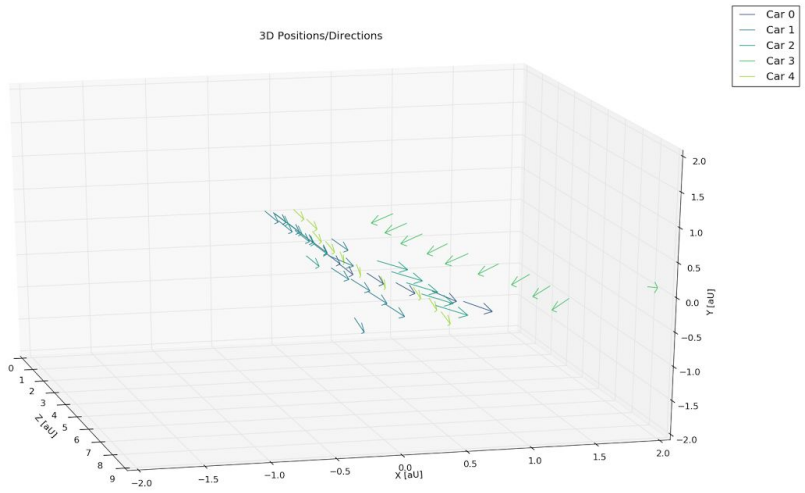
Visualize

3D visualisation is actually quite confusing, so we also use 2D mapping on the ground, realizing a bird eye view from above.



The 2D projection actually looks quite good, except that Car 1 should also be higher up the x axis on the other traffic lane. This is because the algorithm to recover **R** and **t** assumes the same camera matrix **K** for both images which is wrong (see ./output/matches/init_car_1.jpg). A custom function to recover **R** and **t** with different **K** would solve this problem.

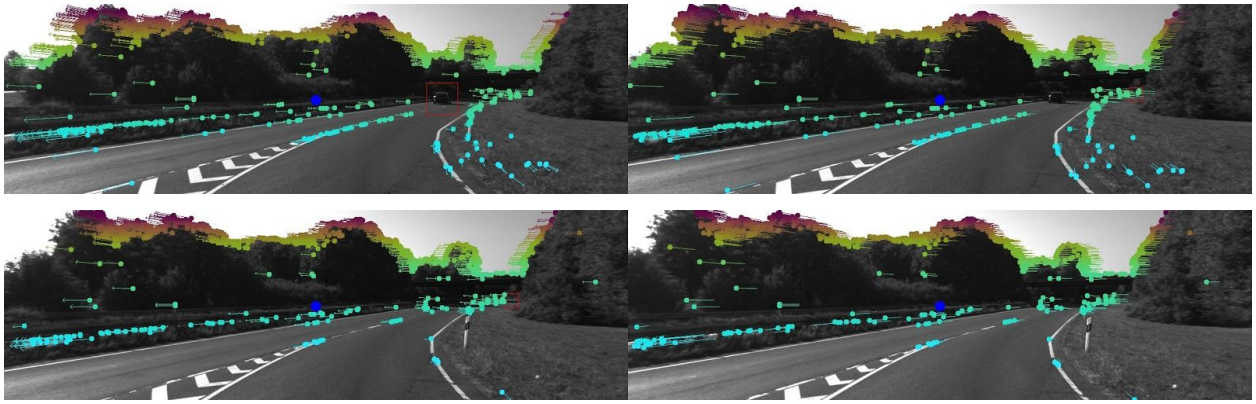
The direction is calculated by multiplying [0,0,1] with the current cumulated roation matrices of the respective step.



The 3D plot looks also pretty good. The camera direction is less stable than the translation information, which is a known property according to current literature.

External Validity

I used the algorithm to analyze the KITTI image dataset of vehicle motion.



Result

The algorithm is very good at finding the keypoints and matching them from image to image, even for other image sets.

The estimation of the rotation and translation is very unstable. This can have two reasons:

1. The camera calibration is very relevant to the problem
2. The opencv algorithm to recover R and t is flawed (which is indicated in a few forum discussions)

Improvement

To improve the recovery in next steps, I would use the algorithm of this paper. It includes e.g. tracking of keypoints over several images.