Tampere University of Applied Sciences

Visual odometry for a stable AR rendering in the HUD

Ruslan Ershov

BACHELOR'S
May 2023

Software Engineering

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Software Engineering

Ruslan Ershov:
Visual odometry for a stable AR rendering in the HUD

Bachelor's thesis 31 pages
May 2023

Recent developments in the automotive industry have emerged the importance of Augmented Reality (AR) applications. AR enhances the driving experience by providing a real-time visual representation of driving assistance features. Generally, AR elements are displayed in a Head-Up Display (HUD) or in an Augmented Reality HUD (AR-HUD). Although the HUD is the most popular choice, the AR-HUD offers richer feature set and remains mainly under the development due to the complexity of the topic.

This work discusses the data stability issues and proposes the approach to reliably track the vehicle on a map using the camera feed. By applying the concepts and solvers of projective geometry and computer vision, the visual odometry algorithm produces a precise 3D pose. The resultant pose is the core of a visual positioning system with which an accurate and stable car positioning is achieved. This system then provides the reliable data required for an immersive AR rendering in the HUD.

Key words: ar, odometry, poi, navigation, hud

CONTENTS

1 INTRODUCTION 5
2 DATA ISSUES 7
2.1. GPS 7
2.2. IMU 8
2.3. Camera 9
3 BACKGROUND 11
3.1. Essense of projective geometry 11
3.1.1 Projective plane 11
3.1.2 Triangulation 12
3.2. Keypoint detection 13
3.3. Optical flow 15
3.3.1 Lukas-Kanade method 16
3.4. Pose reconstruction 16
3.4.1 Essential matrix 17
3.4.2 Relative pose 17
3.5. Kalman filter 18
3.5.1 State update 19
3.5.2 State prediction 19
4 APPROACH 21
4.1. Optical flow odometry 21
4.1.1 Vehicle detection 22
4.1.2 Lane detection 23
5 EXPERIMENTS 25
5.1. Development environment 25
5.2. Benchmark 25
5.3. Optical flow odometry 26
6 DISCUSSION 30
REFERENCES 31

GLOSSARY

POI	Point of Interest
ODD	Operation Design Domain
AR	Augmented Reality
HUD	Head-Up Display
HD Map	High-Definition Map
Odometry	An estimation of change in position over time
IMU	Inertial measurement unit, electronic device measuring and reporting inertial data of the body
GPS	A network of satellites, ground stations, and receivers which provide location and time information
FPS	Frames per Second. Unit of measure for a rate at which frames arrive in a discrete video stream
Trilateration	A method of surveying in which angles of a triangle are computed based on the measured lengths of its sides
Quaternon	An axis in 3D space with an angle of rotation around the axis. Three first values, x, y, z, represent the 3D axis, and the last value w represents the angle of rotation around the axis
Euler angle	Angles which describe the orientation of a rigid body with respect to a fixed coordinate system
Triangulation	A process of determining a 3D point given its projections onto multiple images
Homogeneous space	An extension of a Euclidian space. Each point in a homogeneous space has an additional component w
Keypoint	A spatial location, or a point in an image that defines what stands out in an image

1 INTRODUCTION

AR technology allows the integration of digital information with the physical environment, creating an immersive experience for a user in real-time (Figure 1). Despite the vast potential of AR in the automotive industry, there are several challenges that must be addressed. The main challenge originates from the necessity of supporting the real-time robust rendering of AR elements to the HUD. The rendering algorithms should always have reliable data. To acquire the necessary data, cars are being equipped with the high-definition (HD) maps and sensors. However, both the inertial data of a car (e.g., speed, acceleration) and the GPS data are noisy and therefore tend to be inaccurate. This thesis will discuss these problems in more details in the next section.

Figure 1 AR technology with various AR elements

In general, to integrate AR technology in cars the necessary hardware should be present. Namely, the dedicated processor (e.g., Qualcomm devices such as Snapdragon line) for background computations, the camera system (e.g., front-facing camera, rear-facing camera) to capture images of the surrounding environment, and the HUD display (AR-HUD is the special case when the display is "integrated" into the windshield) that displays the real-time image augmented with AR elements.

Essentially, AR elements are special elements displayed in the HUD (Figure 2) by augmenting a discrete video stream. They are designed to provide users with a real-time driving environment information further improving the situation awareness, safety and navigation experience, and they are bound to specific route

points. Consider navigation AR elements such as chevrons and POIs, arguably the most important elements. Chevrons are arrows helping to perform a maneuver. POIs are specific landmarks or locations augmented with additional information.

Figure 2 HUD with AR view and AR elements in a BMW car

Due to the nature of AR elements – concretely, the route points boundness – they require an uninterrupted reliable data stream, so that they could be placed to the specific route points and remain locked to the target during the rapid scene changes because of the car movement. This serves as a motivation for a visual positioning system, which is the result of this work.

2 DATA ISSUES

2.1 GPS

Global Positioning System is a network of satellites, ground stations, and GPS receivers which, combined, provide location (latitude, longitude, altitude) and time by utilizing a constellation of orbiting satellites which transmit signals to GPS receivers on the Earth. The process of determining a location can be described as follows: the GPS receiver receives signals with the timestamps to calculate the distance between the receiver and a satellite. With three distances to each satellite a three-dimensional position can be reconstructed using the trilateration approach (Figure 3). The fourth satellite is necessary to correct for the clock error of the GPS receiver.

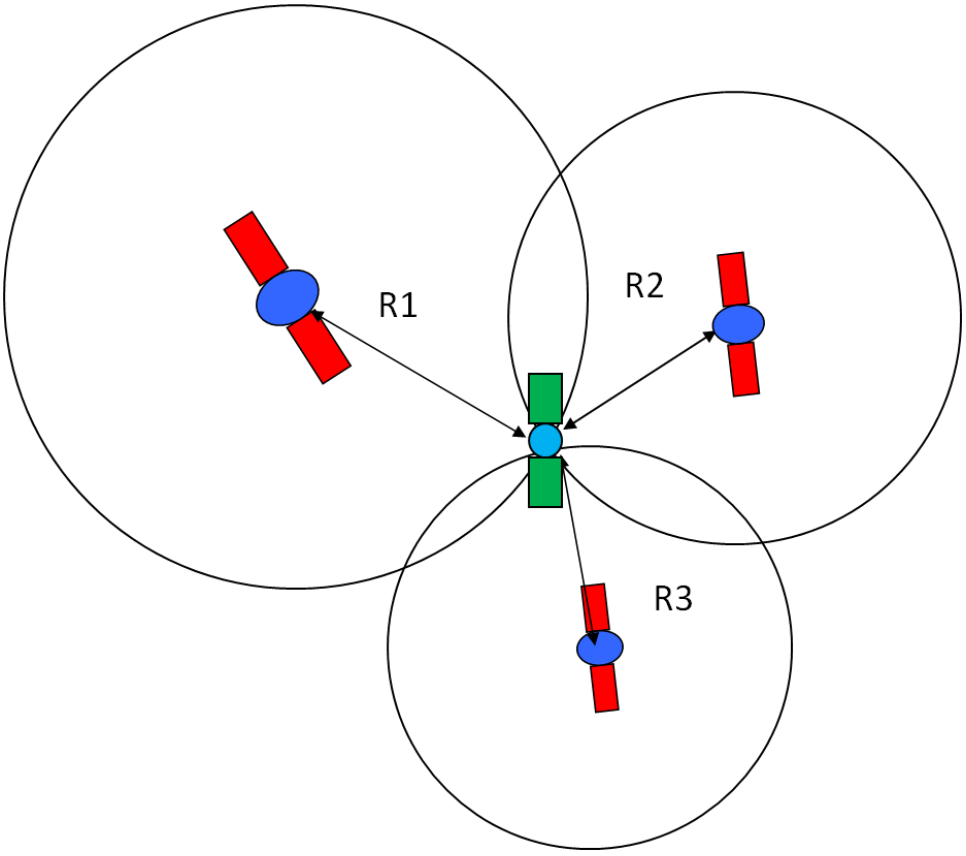

Figure 3 Position reconstruction using trilateration

The reasons for unreliable GPS signals in cars during driving are due to various conditions, such as: bad weather conditions (rain, snow, dense clouds), obstructions (e.g., tunnels, high buildings, mountains, dense forests), electromagnetic interference with other electronic devices in the car, the number and position of GPS satellites, and the receiver quality.

To minimize the loss of GPS signal while driving, the sky should be clear, obstructed areas should be avoided, the GPS receiver should be of a high-quality, the software of the receiver should be frequently updated, and the multiple position technologies should be employed. Among all the variants that can be done by the engineers, only the latter makes sense. Therefore, combining GPS with other positioning technologies like visual odometry can significantly improve stability and performance in difficult conditions.

2.2 IMU

Inertial Measurement Unit is an electronic device, which consists of multiple sensors, and capable of measuring and reporting inertial data of the body. Inertial data includes motion, orientation, and spatial position of the body in 3D space. An IMU measures changes in motion and position over time. The measurements are done by the sensors within the IMU by measuring the forces acting on the device, and further converting them to the motion and orientation of the body. The provided output is raw inertial data usually consisting of quaternions and Euler angles.

The sources of inaccuracies in IMU data are typical for sensors in general and they include: sensor errors, cross-axis coupling, drift and environmental factors. First, sensor errors are due to manufacturing tolerances or vibrations in the sensor properties. Second, sensors can be sensitive to motion in other axes, causing data noise. Lastly, drift, or gradual accumulation of errors over time due to intrinsic (sensor noise) and extrinsic (e.g., temperature, humidity, pressure changes) factors, causes inaccuracies in IMU data, too.

To avoid inaccuracies in IMUs, one can employ various calibration techniques to reduce sensor errors and cross-axis coupling. Furthermore, signal processing approaches, such as filtering or averaging, can help mitigate drift issues, as well as designing IMUs specifically for extreme extrinsic factors. Finally, IMUs can be combined with other sensors and algorithms to improve accuracy. One such algorithm is a visual odometry.

2.3 Camera

The camera system in a car typically consists of a front-facing camera and a rear-facing camera (Figure 4). It might include an infrared camera and very rarely side-facing with depth-sensing cameras. While a front-facing camera used for capturing images of the road ahead, a rear-facing camera used for supporting parking or reversing. An infrared camera can significantly improve the driving in poor environmental conditions (low light, rain, snow, fog). Side-facing cameras help providing a 360-degree view, and a depth-sensing camera provides precise measurements of the distance between the car and nearby objects. Although there are many types of cameras available, usually only front-facing and rear-facing cameras are used to moderate the cost and complexity of the system. Many useful features such as depth estimation are usually implemented with computer vision techniques on a video stream from the front-facing camera.

Figure 4 A typical camera system with front- and rear-facing cameras

Generally, the camera system issues are attributed to hardware, software and natural factors. Among the outstanding hardware and software issues: Field of View (FOV), latency and image stabilization. A narrow FOV limits the amount of real-world information available, while a wide FOV causes distortions and reduces the quality of the image. FOV issues are primarily caused by lens design, or sensor size. Latency is the delay between the camera stream and the displayed AR content, and can be caused by slow camera sensors, processing delays, or slow graphics rendering. An image stabilization issue is due to constant vibrations from the car which causes blur and greatly reduces the image quality. Lastly, natural factors such as poor weather conditions - most significantly poor lighting - severely reduce the image quality and are sometimes caused by the camera sensor design or the lens aperture.

Luckily, all the issues can be mitigated to a reasonable extent by utilizing image processing algorithms. Latency can be reduced by optimizing the processing and rendering stages, image stabilization issues can be compensated via the dedicated stabilization algorithms, and lighting with FOV issues can be mitigated by utilizing appropriate computer vision approaches.

3 BACKGROUND

This section is intended to give a short introduction to, or an overview of, the relevant mathematical background needed for understanding of the implemented approach and the discussion in the last section.

3.1 Essense of projective geometry

3.1.0 Projective plane

A projective plane is an extension of a Euclidian plane. It supports points, lines and planes at infinity. To be more specific, any two lines in the projective plane intersect at a unique point. Parallel lines included which intersect at a point at infinity. Conversely, any two distinct points form a unique line. Points at infinity included. The similar logic applies to planes. Planes can be formed by either two non-collinear points and a line not containing any of the two points, or by three non-collinear points. These interchangeabilities are a result of the concept called duality.

To explain the concept of entities at infinity we first should discuss the main difference between the projective and Euclidian planes. The projective plane operates in the homogeneous coordinates. The definition is as follows. Given a point in an *n*-dimensional space represented with *n* coordinates in the Euclidian plane, the corresponding point will be represented as the $(n+1)$-dimensional point in the homogeneous coordinates (Richard Hartley & Andrew Zisserman 2004, 26-32). For example, if we have a point in 2D Euclidian plane it will be represented as (x, y), and the corresponding point in the homogenous coordinates will be (x, y, w), where the last coordinate is usually set to one (Figure 5). Thanks to the additional coordinate, we are then able to represent entities (e.g., points) at infinity. With such a powerful tool, all transformations can be represented as linear transformations, thus enabling us for a development of projective geometry altogether.

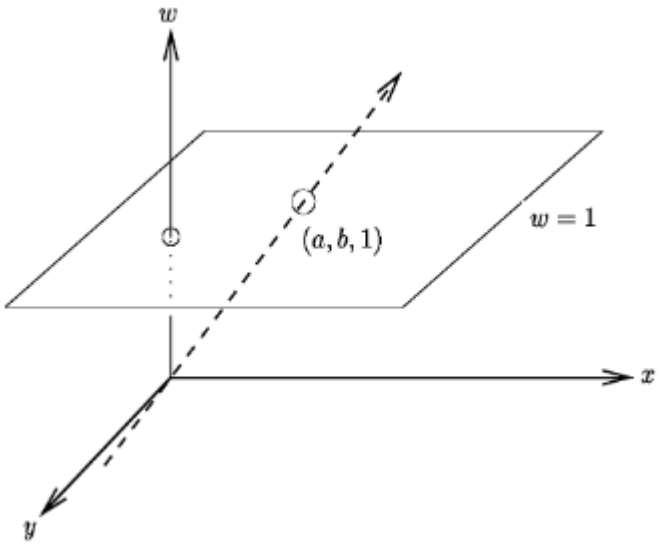

Figure 5 2D homogeneous coordinate system

Having homogeneous coordinates defined, it is easy to imagine a point at infinity lying in a separate plane (with $w = 0$ in homogeneous coordinates) like the imaginary part of complex numbers occupying a dedicated imaginary axis.

The homogeneous coordinate system is the foundation for all computer vision techniques used in the resultant approach.

3.1.1 Triangulation

In computer vision, triangulation is the process of determining a 3D point given its projections onto multiple images. To triangulate points properly we will need camera matrices that perform the projection transformation. However, for the sake of simplicity we will omit the implementation details and discuss only the idea behind this process.

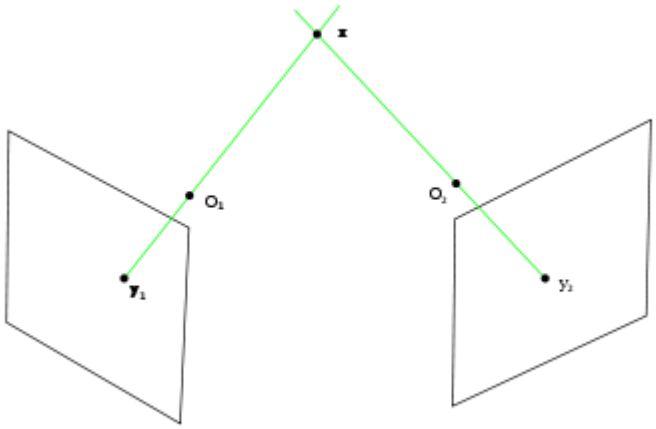

Figure 6 Triangulation process where y_1 and y_2 are the points in the camera planes triangulated to reconstruct a 3D point **x**

Assume that the 3D point is projected onto two camera planes. We know that a point in a camera plane corresponds to a line in 3D space. Then, if we have a pair of corresponding points in two camera images, we call them the reprojections of a common 3D point **x** (Figure 6). This completes the approach. Note that the real-world data contains noise, which causes point correspondences to be inaccurate, thus requiring the reprojection distance-based optimization (Richard Hartley & Andrew Zisserman 2004, 312-321).

The triangulation process is extensively used to recover relative poses needed for the approach described in this thesis.

3.2 Keypoint detection

Image features are distinct recognizable parts of an image used to identify, localize, or classify objects within the image. To detect such feature, we therefore should give it a proper definition. Image features are regions with large variation in brightness or color intensity in all directions. Therefore, they can be detected. The following fundamental algorithm demonstrates the idea.

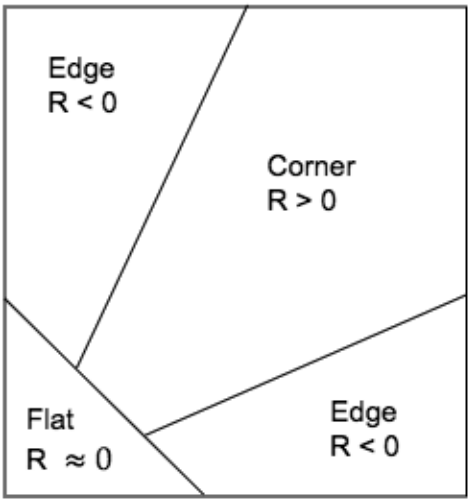

Figure 7 The Harris response score to classify image features

Let *I* be a grayscale 2D image. Next consider a sliding window region $(x, y) \in W$ and the same region shifted by $(\Delta x, \Delta y)$. Then assume the window function $W(x, y)$ is the least squares function, then the difference between two windows is given by: $f(\Delta x, \Delta y) = \sum \big(I(x_k, y_k) - I(x_k + \Delta x, y_k + \Delta y)\big)^2$, where $(x_k, y_k) \in W$. Next note that $I(x_k + \Delta x, y_k + \Delta y)$ can be approximated by Taylor expansion. Let I_x, I_y be the partial derivatives of *I* with respect to position x, y. This produces the following approximation: $f(\Delta x, \Delta y) \approx \sum \big(I_x(x, y)\Delta x + I_y(x, y)\Delta y\big)^2$. In the matrix form: $f(\Delta x, \Delta y) \approx (\Delta x\ \Delta y) M (\Delta x\ \Delta y)^T$, and M is the structure tensor holding partial derivatives of *I*. Then the Harris response score, R, is calculated: $R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2 = \det(M) - k \cdot tr(M)^2$. Essentially, magnitudes of λ_1, λ_2 classify a region. When $|R|$ is small (λ_1, λ_2 are small), the region is flat. When $R < 0$ ($\lambda_1 \ll \lambda_2$ or $\lambda_1 \gg \lambda_2$), the region is an edge. Otherwise, when R is large, the region is a corner (Harrison & Stephens 1988). The classification is depicted in Figure 7.

Now, understanding the general idea behind the corner detection, the keypoint detection is realized in a much similar way, employing slightly more sophisticated detectors such as SIFT, SURF, FAST. Keypoints are essential for the approach presented in this work. With keypoints for two consecutive frames it is possible to calculate various transformation matrices.

3.3 Optical flow

Optical flow is a technique from computer vision to estimate the motion of objects between two consecutive frames of a video stream. The principle behind this technique is to compute the displacement vector for each pixel in an image by comparing brightness and color of corresponding pixels in two (or more) consecutive frames. It is visualized in Figure 8.

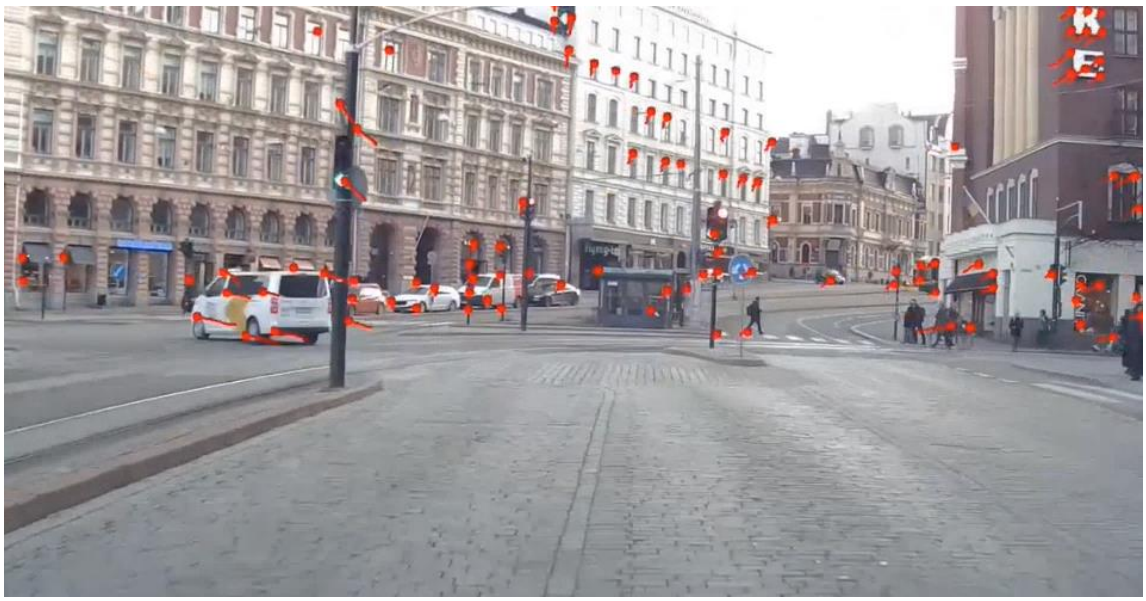

Figure 8 Optical flow computed on the car front-camera feed

There are various algorithms for optical flow computation, but we should limit our discussion to the linear equation approximation given by Lukas-Kanade method. The core assumption of this algorithm is that the displacement of pixels between two frames is sufficiently small, which holds since the front-facing camera video stream usually comes at the rate of approximately thirty frames per second (FPS). The optical flow algorithm solves for the motion vector by minimizing the difference between the image gradient in the current frame and the estimated gradient in the next frame.

3.3.0 Lukas-Kanade method

Let *I* be a grayscale 2D image. Then let I_x, I_y, I_t be the partial derivatives of *I* with respect to position x, y and time t. Assume the displacement between two consecutive frames is sufficiently small and approximately constant within a neighborhood of some point p under consideration. Therefore, the optical flow equation is assumed to hold for all pixels within a window with the center at p. Specifically, the image velocity vector (V_x, V_y) must satisfy the following system of linear equations: $I_x(q_i)V_x + I_y(q_i)V_y = -I_t(q_i)$, where q_i is a pixel inside the window with $i = 1,2,\dots,n$. We can rewrite equations in a matrix form: $Av = b$, where $A = [I_x(q_i)\ I_y(q_i)]$, $v = [V_x\ V_y]^T$, $b = [-I_t(q_i)]$. The system is thus over-determined since the number of equations is bigger than the number of unknowns. The Lukas-Kanade method solves 2×2 system of equations by the least squares principle: $A^T Av = A^T b \equiv v = (A^T A)^{-1} A^T b$, where $A^T A$ is the structure tensor of the image at a point p (Lukas & Kanade 1981, 121-130).

3.4 Pose reconstruction

Whenever we have a pair of cameras each observing a certain scene in the world, we might want to find the relation between the corresponding points of a scene projected onto each camera plane. Having this relation, we can compute relative 3D poses from 2D images.

There are two matrices that relate the camera poses of two views of a scene: fundamental and essential. Let us limit our discussion to the essential matrix since the fundamental matrix is a generalization of the essential matrix when the camera intrinsics are unknown, i.e., when the camera is uncalibrated.

3.4.0 Essential matrix

The essential matrix can be calculated from a set of corresponding keypoints between the two views. We use David Nistér's algorithm for essential matrix computation, and therefore shall describe it here. The main advantage of the algorithm is its efficiency, making it a practical choice for real-time applications often required in the automotive industry.

After the keypoints were detected in both images, they are normalized by subtracting the mean and scaling by the standard deviation. Next, $A_{5\times9}$ matrix is constructed, where each row is a correspondence between a pair of normalized keypoints. Note, however, that this will still work for the number of rows $m \geq 5$. Then the matrix A is decomposed using SVD (Singular Value Decomposition): $A = USV^*$, where U and V are orthogonal matrices, and S is a diagonal matrix. Use U and V matrices to construct the polynomial matrix $B_{3\times13}$, which first ten columns are polynomials of degree $n = \{1,2,3\}$ of the elements of U and V and the last three columns store a null vector to *B*, $v = [x\ y\ 1]^T$. Then construct a polynomial equation of degree ten using the determinant of B which in turn should vanish since the vector v is a null vector. Solve the equation using any root-finding algorithm to obtain up to ten solutions for the essential matrix. Lastly, the reprojection error between the keypoints is minimized using the Levenberg–Marquardt algorithm or similar (Nistér 2004, 756-770). Note that the real-world data always contain noise and thus require the number of keypoint correspondences$N \gg 5$, including a robust estimation algorithm (e.g., RANSAC) to remove outliers.

3.4.1 Relative pose

With the essential matrix E estimated, we can further determine the rotation matrix R and translation vector t between two coordinate systems of the cameras. The essential matrix can be decomposed with SVD, but we prefer to follow David Nistér's approach and decompose the essential matrix by doing the cheirality check described next.

In simple terms, the cheirality check is a technique to ensure that the estimated camera poses are physically valid, i.e., the cameras face the same scene and are not behind it. First, compute the distances between each 3D point and each camera center and check their signs. If the distance signs are the same, we conclude that the point is in front of both cameras, and hence keep it. Otherwise, the point is invalid, or an outlier, and is discarded. The distances calculations are carried out using the triangulation technique.

The resultant relative pose is the core element of the approach in this work. In fact, this pose is an output of the described algorithm.

3.5 Kalman filter

Numerous modern systems consist of sensors that estimate unknown or hidden states based on series of continuous measurements. One of the greatest challenges for such systems is to provide an accurate estimation of the hidden states in presence of uncertainty. The source of uncertainty can be many things, such as clock precision, vibrations in the sensor properties, and external factors. The data issues were discussed in Section 2. To estimate the hidden states in presence of uncertainty, the Kalman Filter was developed. Additionally, the Kalman Filter can predict the future state of the system based on the earlier estimations. We now aim at simply introducing the Kalman Filter algorithm and therefore examine it in one dimension only. We wish to introduce the Kalman Filter algorithm to discuss its potential applications for the visual positioning system algorithm in the last section.

In total there are two main components in the Kalman Filter algorithm: state update and state prediction. We will discuss state update in more details by covering the following equations: state update, covariance update, Kalman Gain (Q. Li, R. Li, K. Ji and W. Dai 2015, 74-77). We will then discuss the state prediction part by covering the following equations in general form: dynamic model (or state extrapolation) and covariance extrapolation. Consider a system doing successive measurements, so that both stages operate in the iterative fashion.

3.5.0 State update

We will start with the state update equation. Let the update state factor be a given constant value K and let the measurement value on a step $n \neq 0$ be z_n. Then, having an estimated state of a current iteration step from a previous iteration (i.e., an estimation of the state for the step n made on the previous step $n-1$), $\chi_{n,n-1}$, we can predict the state of the current iteration, $\chi_{n,n}$, as follows: $\chi_{n,n} = \chi_{n,n-1} + K(z_n - \chi_{n,n-1})$. Few remarks are necessary. First, the update state factor is called Kalman Gain and it is dynamic and updated iteratively. Second, for the step $n = 0$ we initialize $\chi_{n,n-1}$ with a value called the Initial Guess, which may be an educated guess or an arbitrary random value (e.g., from a normal distribution).

Now let us discuss the Kalman Gain equation. Since Kalman Filter treats the estimate as a random value (we do not know the estimate error), we should extrapolate the estimation uncertainty, too. Additionally, the measurements errors are random, so they can be described by variance σ^2 which is called a measurement uncertainty. Given an uncertainty in the estimate $p_{n,n-1}$ and the measurement uncertainty r_n (provided as an input together with z_n), the Kalman Gain is calculated as follows: $K_n = \frac{p_{n,n-1}}{p_{n,n-1}+r_n}$.

With the Kalman Gain equation presented we can now calculate the Kalman Gain factor for the update state equation. However, we do not yet know how to update the state uncertainty. Therefore, the state uncertainty is defined as follows: $p_{n,n} = (1 - K_n)p_{n,n-1}$.

3.5.1 State prediction

Typically, the dynamic model is unique for each problem, and it is hard to describe it in general form. We can consider a case of a simple linear motion with a constant velocity. Knowing the position x of an object at a step n, we can express velocity: $\dot{x} = v = \frac{\partial x}{\partial t}$. Then the state on the next iteration $n+1$ can be predicted:

$x_{n+1} = x_n + \dot{x}\Delta t$, $\dot{x_{n+1}} = \dot{x_n}$. Lastly, the covariance extrapolation (constant velocity case) is given by: $p_{n+1,n} = p_{n,n}$.

4 APPROACH

4.1 Optical flow odometry

In this approach we utilize an optical flow algorithm to calculate an optical flow between two consecutive frames of a video stream. Based on the calculated optical flow we compute the essential matrix which we further decompose into rotation matrix and translation vector. Lastly, a relative 3D pose is recovered using rotation matrix and translation vector.

As initialization, we extract keypoints using the FAST corner detection algorithm. We consider a circle of 16 pixels to classify whether a candidate point p is a corner. If a set of N contiguous pixels in the circle are all brighter or darker than the intensity of the candidate I_p, then the candidate point is a corner (Rosten, Porter, and Drummond 2010, 105-119). The brightness comparison uses a threshold $t = 20$ for a brightness difference.

Next, we calculate an optical flow using the Lukas-Kanade optical flow algorithm. For that we need to have two consecutive frame images and keypoints of the first image frame. The employed Lukas-Kanade algorithm additionally refines optical flow by utilizing a Gaussian pyramid to calculate the optical flow at different scales. We therefore solve for a velocity vector $v = [V_x\ V_y]^T$. We pick a square window of size $n = 21$. We iteratively recalculate the velocity vector until either the number of iterations exceeds the maximum value $m = 30$ or the search window moves by less than the threshold value $\epsilon = 0.01$ for each successive tracking window. We use keypoints calculated with FAST on the first iteration, and later we dynamically use the optical flow keypoints from the previous iteration for the first frame leaving the FAST keypoints aside.

With the optical flow successfully computed and the keypoints tracked on the second image, we can calculate the essential matrix and further recover a relative pose. Calculating the essential matrix E and further applying the cheirality check, we obtain the rotation matrix R and translation vector t with which we can reconstruct the position. We first construct the translation matrix $T = R * t$ and then

extract the first three entries of T to construct a vector p representing a relative position of the camera (and consequently, of the car) in 3D space.

4.1.0 Vehicle detection

When driving a car on a road we usually encounter many moving objects. Since the scene is highly dynamic the feature matches might be compromised. This significantly reduces the accuracy of the keypoint tracking. We therefore propose a solution to mitigate this by employing a vehicle detector.

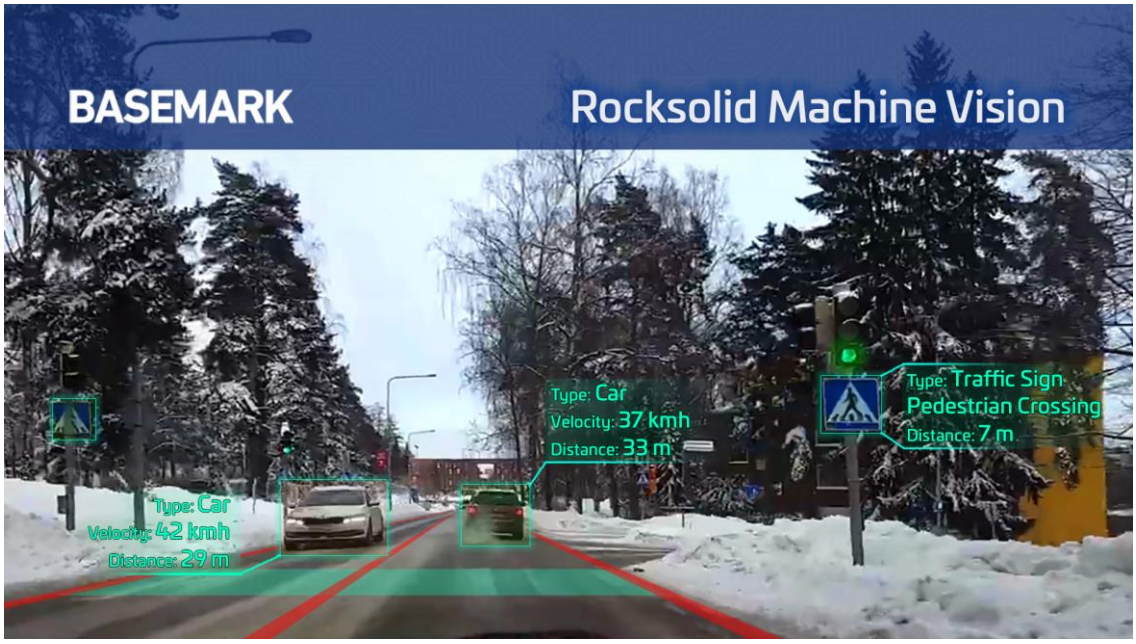

Figure 9 Object detection

We use YOLOv5 object detection model for a vehicle detection. It is a state-of-the-art real-time object detection algorithm. It is based on a deep convolutional neural network (DCNN) that uses anchor boxes and feature pyramids for scale and size invariant object detection (Redmon, Divvala, Girshick and Farhadi 2016, 779-788). A typical object detection is depicted in Figure 9. We now omit the details of the object detection process and discuss its purpose in context of the optical flow.

Assume the only moving objects on road are vehicles and assume that they are always in motion. Then, for each vehicle on a given image, detect a bounding box around a vehicle. We call this bounding box a moving object mask and it is

represented as a rectangle with the vertex point (x_m, y_m) and with the width and height (w_m, h_m). We therefore discard every tracked keypoint from the first image frame which is within the moving object mask. Namely, every tracked keypoint (x_p, y_p) such that $x_p \geq x_m$ and $x_p < x_m + w_m$, $y_p \geq y_m$ and $y_p < y_m + h_m$.

4.1.1 Lane detection

The resultant algorithm still lacks stability. It performs poorly on turns, or putting it differently, the optical flow odometry does not properly handle lateral movements. To improve the lateral movements sensitivity, the lane detection algorithm is introduced.

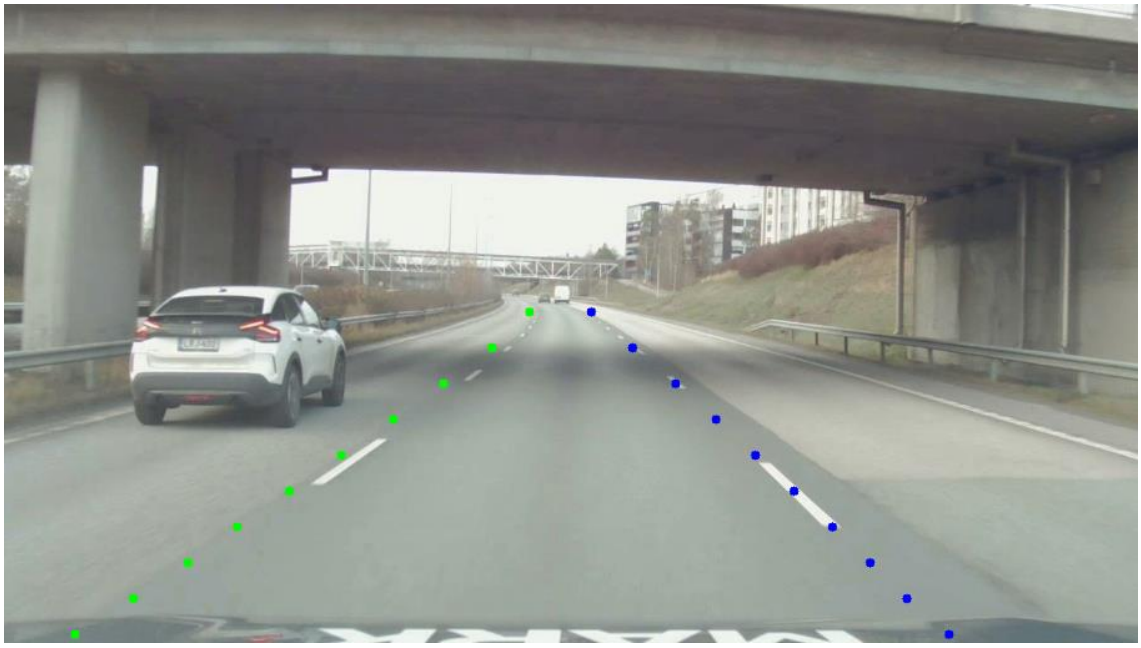

Figure 10 Lane detection

We use an internal lane detection model. Given a road image it produces 28 lane points x_l, y_l, with 14 points for both left and right lanes, respectively (Figure 10). These points are produced by a basic Kalman Filter based on the raw detected lanes by mapping raw data to original image pixels. We now switch to the discussion of the lane detection in context of the optical flow.

We therefore assume the car is always on a road, meaning there is always going to be at least one distinctive marked lane on the ground. If these conditions hold, we always have two sets of 2D points corresponding to two road lanes. With

simple trigonometry we can find an angle between two lanes, θ_l. Next recall that we work with two consecutive frames, so we can also calculate an angle between two points in each point pair (first, second frame), α_i, where $i = 1,2,\dots,n$ with n being the number of points in either of two frames. Lastly, we can compute the angle difference, $\gamma_i = \theta_l - \alpha_i$, and rotate all the points in the second frame by the corresponding angle difference γ_i.

5 EXPERIMENTS

5.1 Development environment

The core algorithms were implemented with C++17, while the various visualizations were created with Python using Matplotlib library. The C++ code heavily utilized the OpenCV library. All experiments took place in the ROS 2 Foxy environment. The code was organized as groups of independent ROS packages built by ROS colcon build tool with CMake. All packages were represented as ROS nodes with multiple subscribers and publishers. The communication between nodes was carried out by the underlying ROS inter-process communication (IPC) protocols. All the experiments were done on Linux Ubuntu 20.04.

5.2 Benchmark

The benchmark for a visual odometry was chosen to be a simple comparison graph. Although not as accurate as numerical benchmark, it allows to compare different versions of a visual odometry at an acceptable level. The benchmark is formulated as follows: given a video stream from the front-facing camera in a car and having a corresponding egomotion trajectory (represented as a 3D point) at every frame, plot a graph of all the 3D egomotion trajectory points (ground truth) against a graph of all the 3D reconstructed poses from the visual odometry (prediction). The better the visual odometry trajectory (orange curve) matches the egomotion trajectory (blue curve), the better the positioning system performs. Additionally, plot graphs of the lateral and longitudinal deviations of the visual odometry against the egomotion. The more closely the deviations match, the better the positioning system performs.

The egomotion trajectory is constructed using IMU data from a car. The validation video is a recorded drive on a highway near Helsinki consisting of 4500 unique frames.

5.3 Optical flow odometry

In total twelve graphs were created for the optical flow odometry. Three for the pure optical flow odometry (Figure 11), three for the vehicle detection-aided optical flow odometry (Figure 12), three for the lane detection-aided optical flow odometry (Figure 13), three for the vehicle and lane detection-aided optical flow odometry (Figure 14).

The detection-aided systems alone did not improve the result. In fact, they performed worse than the pure odometry. There are large differences between the visual odometry and egomotion trajectories. Additionally, both lateral and longitudinal deviations are significant for each version of the optical flow odometry.

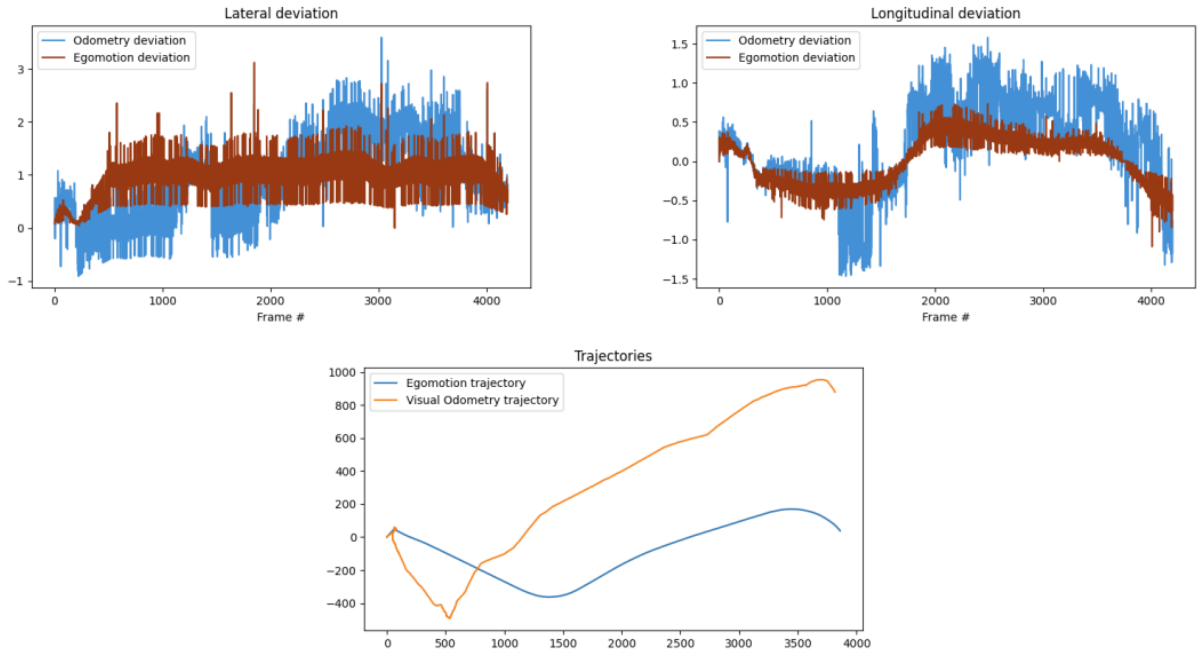

Figure 11 Pure optical flow odometry

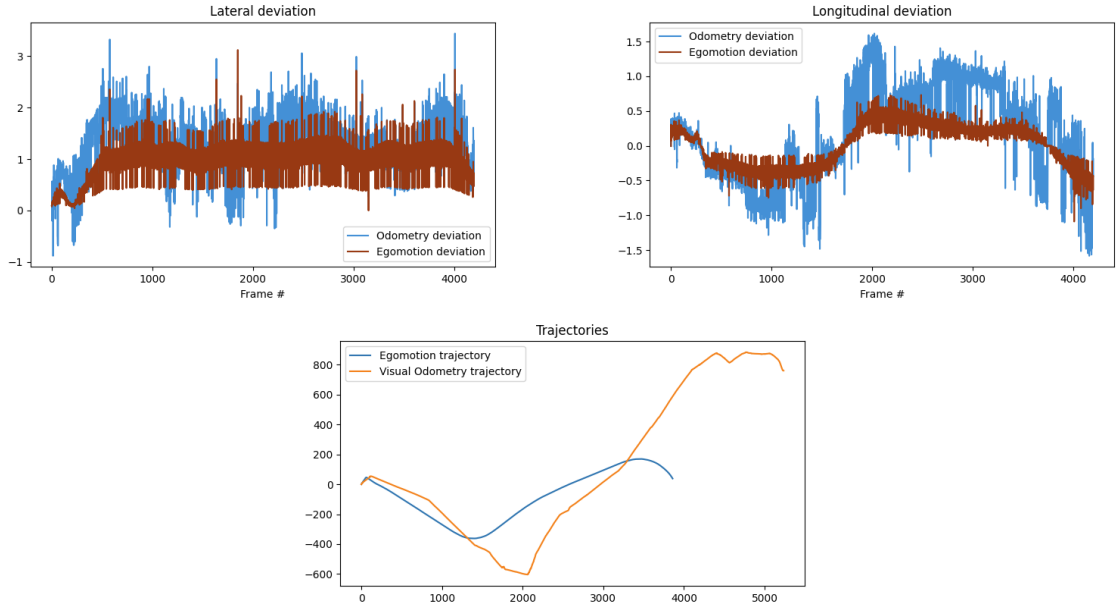

Figure 12 Optical flow odometry with vehicle detection support

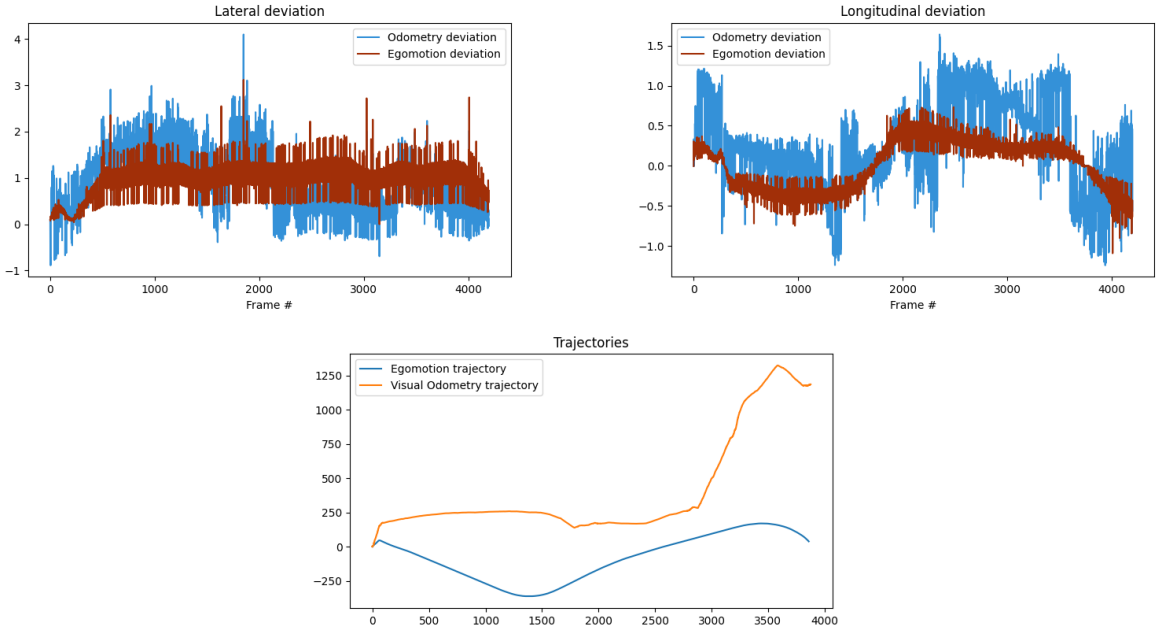

Figure 13 Optical flow odometry with lane detection support

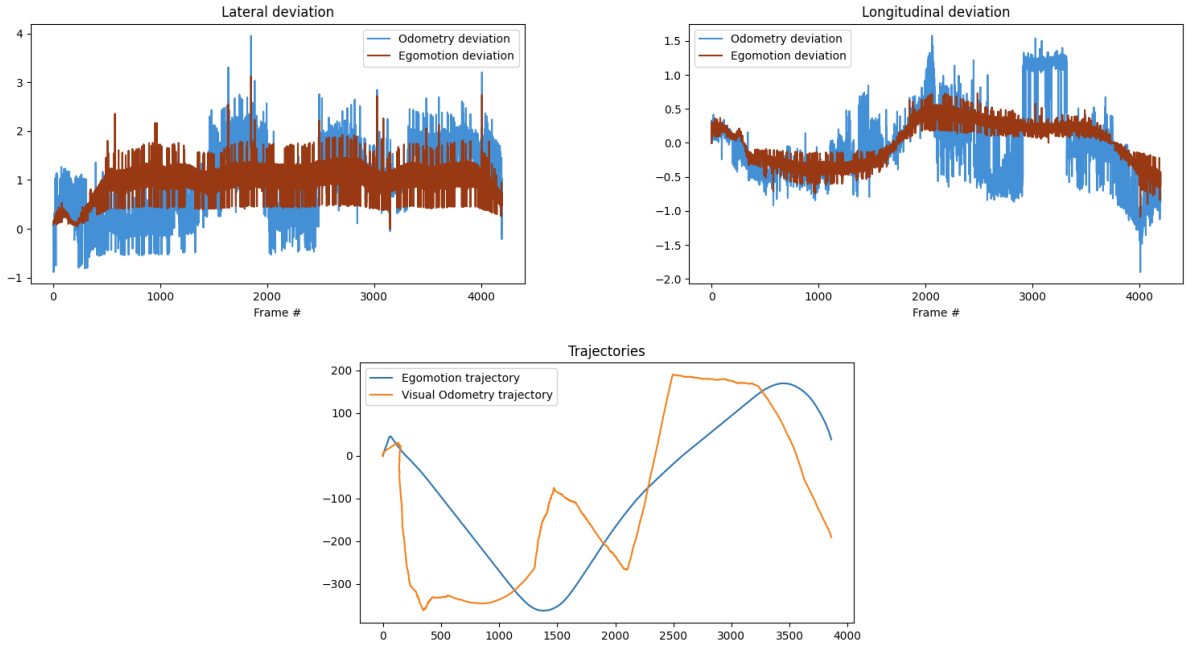

Figure 14 Optical flow odometry with lane and vehicle detection support

Generally, lateral deviations are similar for all versions, though the versions backed up by the detection-aided systems show slightly better matches of the distributions for lateral deviations. However, the lateral deviations distribution of the optical flow odometry with the vehicle and lane detection support (Figure 14) fails completely on the first frames, leading to a large deviation in the trajectories in the beginning. It is not clear why this happens, as the deviations along the positive y-axis of the vehicle detection-aided odometry (Figure 12) and the lane detection-aided odometry (Figure 13) combined should have cancelled out the deviations in the negative y-axis of the pure visual odometry (Figure 11).

Longitudinal deviations, however, look different. There are significantly more deviations. For the first three plots the deviations mainly occur along the positive y-axis. The last plot (Figure 14) demonstrates, to our surprise, a different behavior with the amount of deviations roughly balanced in all directions. With such behavior the longitudinal deviations introduced by the lane and vehicle detection-aided odometry arguably match the egomotion longitudinal deviations better. This is, we think, the key to a better match between two trajectories.

In conclusion, it is evident that the optical flow odometry supported by the vehicle and lane detection systems produces a better positioning system than the pure

optical flow odometry. However, the nature of improvements is widely counterintuitive. With the introduction of the lane detection system, we aimed at lateral improvements, but it made the lateral deviations larger while reducing the longitudinal deviations. We find it compelling and unsettling that there are such significant improvements with both the vehicle and lane detection systems enabled, while, when not combined, the results are worse than the pure optical flow odometry.

6 DISCUSSION

As a result of this work the visual positioning system has been created. The optical flow alone did not provide the desired accuracy, so we had to aid it with various algorithms. During an extensive process of trial and error, we have managed to develop a reasonably accurate positioning system by assisting the optical flow odometry with vehicle and lane detectors.

Due to the tight schedule, we did not have time to test other approaches, but there is clearly a room for improvement. Mainly there are many ways we can still improve the optical flow odometry alone. The approach presented in this work is rather hacky, because the odometry relies heavily on vehicle and lane detection systems. Additionally, the egomotion trajectory was created using IMU data. In section 2.2 we discussed its various data issues. So even though much more reliable than the trajectory produced by the purely visual approaches, we should not trust the egomotion trajectory completely.

First, the optical flow algorithm simply tracked feature points and then estimated the essential matrix between two views from point correspondences to recover a relative pose. Naturally, optical flow algorithms are capable of tracking feature points over more than two images.

Second, we did not use IMU data with which it is possible to predict the next state (or pose in our case) with Kalman Filter. However, it could greatly improve the performance of the system if we could predict the next state and then compared it against the optical flow. Or the predicted state could be used as a penalty to restrict sudden sharp movements in the generated trajectory.

Finally, a relative pose from a regular optical flow algorithm is usually a starting point for a more sophisticated relative pose estimation algorithm. We could triangulate the point correspondences to a 3D point cloud and optimized camera poses by minimizing the reprojection error in a process called bundle adjustment.

REFERENCES

Boboc, Razvan & Florin, Girbacia & Butila, Eugen. (2020). The Application of Augmented Reality in the Automotive Industry: A Systematic Literature Review. Applied Sciences. 10. 4259. 10.3390/app10124259. Read on 02.03.2023.

J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91. Read on 05.03.2023.

Baykut, Akgul and Ergintav, "GPS Data Modeling and GPS Noise Analysis," 2006 IEEE 14th Signal Processing and Communications Applications, Antalya, Turkey, 2006, pp. 1-4, doi: 10.1109/SIU.2006.1659765. Read on 12.03.2023.

Kj, Nirmal & Sreejith, A. & Mathew, Joice & Sarpotdar, Mayuresh & Suresh, Ambily & Prakash, Ajin & Safonova, Margarita & Murthy, Jayant. (2016). Noise modeling and analysis of an IMU-based attitude sensor: improvement of performance by filtering and sensor fusion. 99126W. 10.1117/12.2234255. Read on 12.03.2023.

Multiple View Geometry in Computer Vision Second Edition. Richard Hartley and Andrew Zisserman, Cambridge University Press, March 2004. Read on 01.02.2023.

Harris, Christopher G. and M. J. Stephens. “A Combined Corner and Edge Detector.” Alvey Vision Conference (1988). Read on 15.03.2023.

B. D. Lucas and T. Kanade, An iterative image registration technique with an application to stereo vision. Proceedings of Imaging Understanding Workshop, 1981, pp. 121-130. Read on 16.03.2023.

Edward Rosten, Reid Porter, and Tom Drummond, "Faster and better: a machine learning approach to corner detection" in IEEE Trans. Pattern Analysis and Machine Intelligence, 2010, vol 32, pp. 105-119. Read on 17.03.2023.

D. Nister, "An efficient solution to the five-point relative pose problem," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 26, no. 6, pp. 756-770, June 2004, doi: 10.1109/TPAMI.2004.17. Read on 18.03.2023.

Q. Li, R. Li, K. Ji and W. Dai, "Kalman Filter and Its Application," 2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS), Tianjin, China, 2015, pp. 74-77, doi: 10.1109/ICINIS.2015.35. Read on 21.03.2023.