Bachelor's Thesis

Racing Line Detection - Recording and Comparison of Racing Lines

Ideallinienerkennung - Aufnahme und Vergleich von Ideallinien

by

Tim Oesterreich

Potsdam, July 2016

Supervisor

Prof. Dr. Christoph Meinel,
Philipp Berger, Patrick Hennig

Internet-Technologies and Systems Group

Disclaimer

I certify that the material contained in this dissertation is my own work and does not contain significant portions of unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation.

Hiermit versichere ich, dass diese Arbeit selbständig verfasst wurde und dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt wurden. Diese Aussage trifft auch für alle Implementierungen und Dokumentationen im Rahmen dieses Projektes zu.

Potsdam, July 31, 2016

(Tim Oesterreich)

Kurzfassung

deutsche Zusammenfassung...

Abstract

//TODO

Motor racing is all about getting around a track as fast as possible. One of the most important and most driver-influenced steps in achieving best times is the racing line, especially hitting the ideal racing line through corners. Our race analysis system collects different car-based data to evaluate and compare the driving style of drivers which helps them gain an advantage over the competitors.

This paper especially focuses on the recording and comparison of racing lines, using different methods, like Visual Odometry and lane detection and use the extracted data to compare and analyze the driving behavior of different drivers.

Contents

1 Introduction

1.1 Motivation

The trajectory around a track that allows a given vehicle to traverse the circuit in the minimum amount of time is called the ideal racing line. There are a lot of parameters that determine the minimum lap time. In general it is a trade-off between the length of the taken line and the speed carried around the track, which in most cases turns out to be the line with the least curvature. Needless to say, in a race this is the path every driver wants to take. In reality, however, this isn't an easy task. In order to achieve best times, a racing driver has to remember exactly which corner of the track comes next, how fast they can take this corner, see how the car is positioned on the track and process much more information.
To assist aspiring drivers feel like professionals, we implemented a racing driver analysis system, that is capable of recording, comparing and evaluating race related data. The system saves a complete history of the many sensor data values, like speed, throttle position and safety relevant features, like engine temperature, the car produces. Also it tracks the car on the course using a combination of different methods. This car tracking will be the main focus of this thesis.
The comparison of two drivers requires very robust measurements, that return the same result for every given input. The most common tracking method, GPS, relies on external sources, namely satellites that communicate with the GPS device. The biggest problem with this traditional approach is that GPS isn't fully available everywhere. Especially on remote areas, which is where most race tracks are, there tends not to be complete coverage. This would result in faulty results and incomparable racing lines.
In order to break away from these dependencies, we used a camera-based approach. This allows us to track the relative translation and rotation of

the vehicle independent of its speed, location or surroundings. Apart from the recording of these racing lines, this paper also discusses how to process the obtained data in order to provide a useful support for racing drivers.

1.2 Project Scope

During the one year long bachelor project "Feel the Car - Automatic Anomaly Detection for Unstructured Test Drive Data" at the Hasso-Plattner-Institute in Potsdam, I worked together with four fellow students and in cooperation with Mercedes-AMG on analyzing video and sensor data from car test drives. Part of the project was to find a possibility to retrospectively evaluate a race for a given driver. This is especially useful for AMGs Driving Academy [1], which is a service that lets non-professional drivers drive on race tracks and become a racing driver.
To achieve this goal, and other requirements, such as detecting potholes on the street or security relevant warning signs in races, we used a stereo camera and a mini-computer that processed the incoming raw data. Additionally, an On-Board-Diagnostic 2 (OBD-II[2]) dongle was used to extract the sensor data that cars have already integrated. The analyzed result was then displayed in our web application, which also provided a driver profile page that provides a detailed overview of their recorded laps.

[1] http://www.mercedes-amg.com/driving-academy/
[2] http://www.obdii.com/

1.3 Problem Context

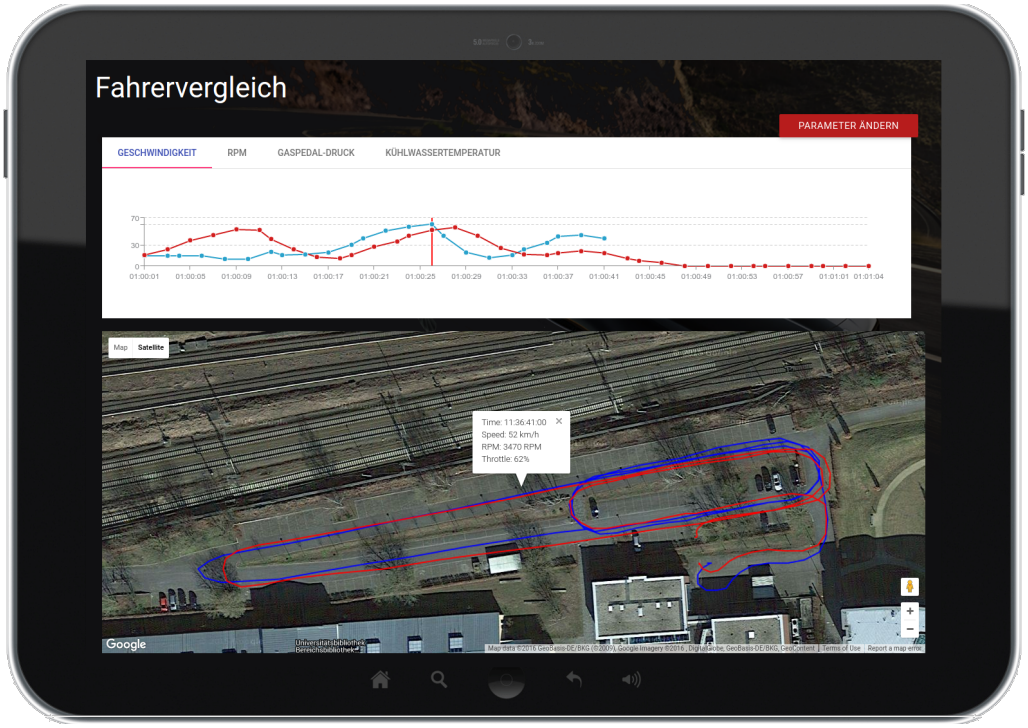

Figure 1: Driver Comparison

Initially, our task was to analyze existing video material from test drives made by AMG. The exact specifications of this analysis were defined at a meeting with AMG and developed to mainly consist of video- and sensor-based driving aids for drivers at the AMG Driving Academy. Besides the detection of security-relevant events, like warning flags, cars leaving the track into the gravel bed or potholes on the street, we also wanted to find a way to compare and evaluate the driving behavior of two drivers. During our first tests, which started with a mobile phone camera, we found out that there are a lot of challenges and prerequisites in video analysis. For example, every camera has to be calibrated first, to compute for the individual distortion a camera lens causes. This is not only a fairly complex task that requires high accuracy, it also prevents an

easy replacement of camera hardware.
This made us look into alternatives and we decided to use a stereo camera. A stereo (or 3D) camera is able to determine metric measurements from the camera image, which enables it to calibrate itself. Also the additional information we can extract from a 3D image made a more accurate and faster detection possible.

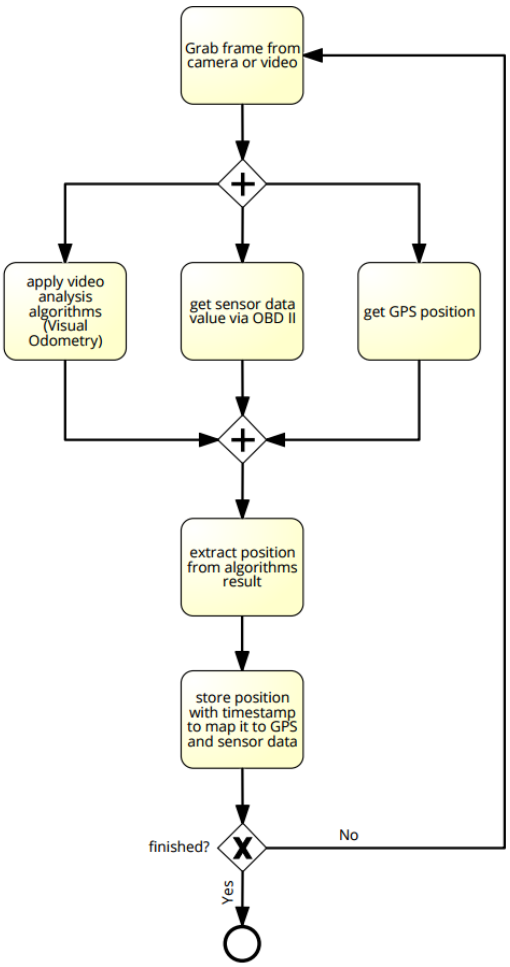

Figure 2: Process Diagram Racing Line Recording

Using techniques that will be described in section 3, we were able to

record the racing line of a car based on a stereo video and GPS. Together with the extracted sensor data from the OBD II interface, we stored the points in a database. The process is depicted in Figure 2.

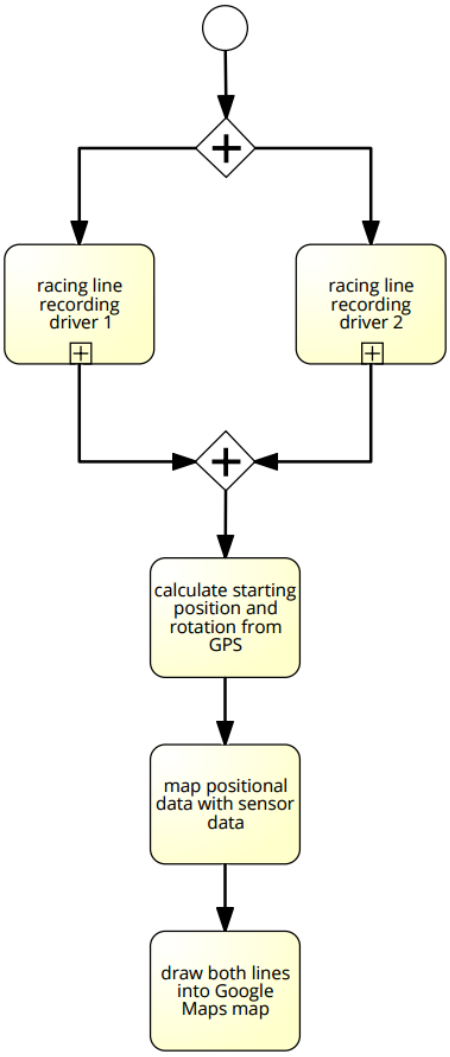

Figure 3: Process Diagram Racing Line Comparison

Figure 3 shows how the recorded racing lines can then be compared. We decided to use a Google Maps integration to properly determine where on the track potential improvements can be found. Because of the nature

of our recording algorithms, calculating a relative position that can't on itself be mapped to a geographic location, a few preparations, like calculating the starting point and rotation, have to be done. These are further described in Section 3.2. Interesting data values were displayed by coloring in the racing line in case of the detailed driver view (figure 4) or using an info window which gives an overview of the data at a given point (figure 1).

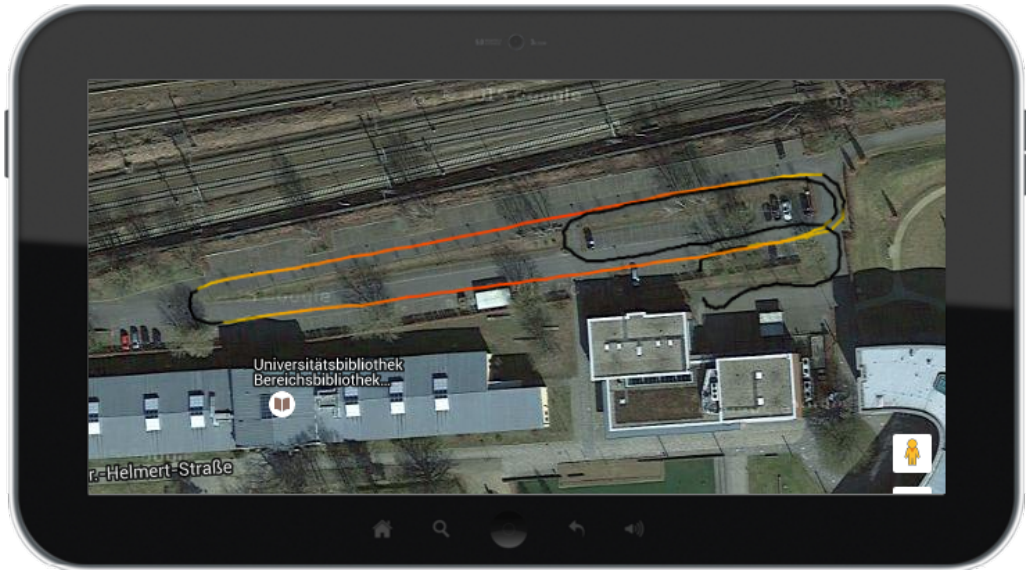

Figure 4: Detailed Driver Overview

2 Related Work

Contrary to most racing sports, where a racing line is usually drawn by an expert, video games, especially from independent developers, tend to concentrate on calculating the racing line for the computer-controlled cars.

2.1 Vamos Racing Simulator

One example is the *Vamos racing simulator*[3]. It uses an iterative curvature-minimization technique, simulating spring-loaded hinges that are placed in the middle of the track. The lateral forces a car produces during cornering are used to simulate the opening or closing of these hinges, iteratively shaping the racing line. After a certain number of iterations the curve stabilizes. This happens when the force across all hinges and therefore the curvature of the racing line is close to minimal. After the calculation is done, the possible speed of the racing cars at every point on the track can be calculated.

2.2 RaceOptimal

RaceOptimal[4] is a website that offers calculated ideal racing lines for 4 different vehicles on a variety of tracks. The approach is, similarly to that of Vamos, iterative. However, their focus lies on the cars physics. They use a Bézier-curve to approximate a smooth line across a circuit, based on predefined control points. Then the fastest possible speed for every point on the track is calculated based on the curvature of the turn, the friction coefficient and mass of the car and limited by the cars top speed. After that the acceleration is adjusted to not exceed the power of the engine and

[3] http://vamos.sourceforge.net/
[4] http://www.raceoptimal.com/

capabilities of the tires. At last, the graph is adjusted to also include the capacity of the brakes and aerodynamic drag. This algorithm is used on a certain number of possible lines, the initial population. These solution then breed offspring, which are combinations of the initial parents, and are modified randomly, to get as many different racing lines as possible. The best children are kept and breed again, bad solutions will be thrown away. This process is repeated until the result, being the lap time on the given track, stays consistently quick.

A similar genetic approach is used in this paper: [1]. Here the track is decomposed into several segments and for each segment a algorithm is applied, that searches the best trade-off between the minimization of the length and curvature of the racing line. To assess the fitness of each genetic iteration, the solution is tested in the TORCS racing simulator[5].

2.3 Optimal Control

Optimal control is an optimization method that, in general, tries to minimize a cost function of control variables in a dynamic system. In case of racing line optimization, the cost function is the lap time, the control variables are steering angle and throttle/break input and the dynamic system is a vehicle model restricted to the boundaries of the road. [2, 3]

2.4 Euler Spiral Method

An Euler Spiral (figure 5) is a curve whose curvature changes linearly with its curve length. They can be used to connect a tangent to a circular curve, which can be translated to connecting a straight with a curve on a track.
[4]

[5] https://sourceforge.net/projects/torcs/

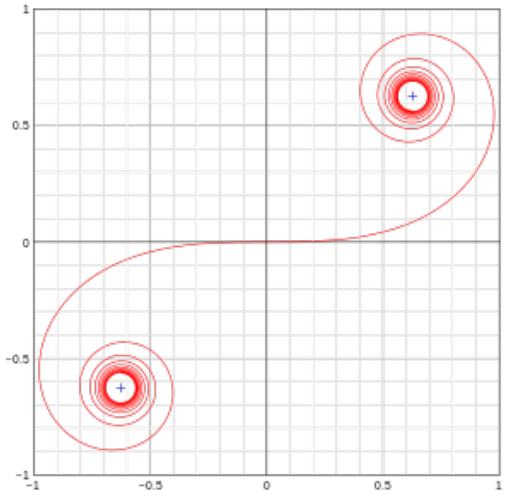

Figure 5: A double-ended Euler spiral

The racing line follows a segment of the Euler spiral that connects a starting point with an end point. This comes close to minimizing the curvature, but does not completely reach the minimum. Also there are multiple segments that connect two points, so as, usually, the least curvature is wanted, the segment with the biggest radius is taken.

Most of these methods either need exact knowledge of the track or a simulation to work. This paper discusses possibilities to compare racing lines on an arbitrary track with no additional software other than a web browser.

3 Concept

As this Bachelor Thesis tackles two problems, the algorithms can be divided into two categories. The first category consists of racing line recording concepts and the second one of ways to compare these recordings.

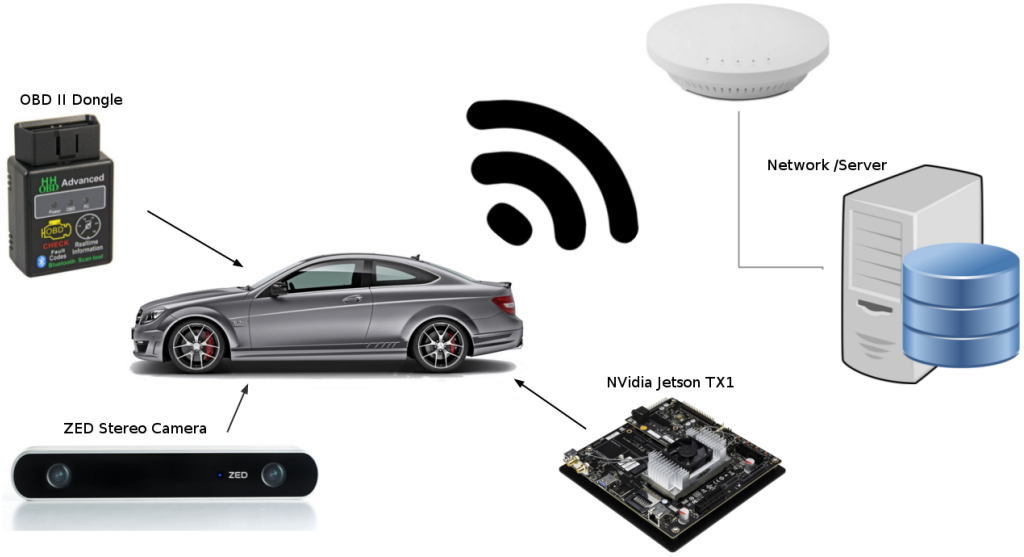

Figure 6: Hardware Setup

Figure 6 show our hardware setup, including all the devices we used to achieve our solution.

3.1 Recording

At first we have to find out which racing line a car has taken. For that a variety of methods were used that work hand in hand to record the racing line itself, the track the car was driving in and data about the car, like speed, throttle position and usage of the brakes. The general concept is explained in Section 1 Figure 2.

3.1.1 Visual Odometry

Visual Odometry (VO) is a concept that most camera-based robots use for navigation. It uses two consecutive camera frames to calculate the rotation and translation of the camera between them. The advantage over traditional tracking systems like GPS is that it is not dependent on any external sources (like satellites or radio towers) to determine the position of the object.

It is also capable of much higher polling rates than most GPS receivers, which normally poll at 1 Hz, so they get 1 positional update per second. The potential speed of Visual Odometry is linked to the frame rate of the camera. That way, a camera that records at 30 frames per second enables Visual Odometry to estimate a position 30 times a second. The crude algorithm behind VO determines and stores recognizable features in one frame, using a feature detection algorithm and tracks them to a consecutive frame. The displacement of these features determines the direction the camera is traveling. At first we need two consecutive, rectified camera images. It is important that the images are rectified, because otherwise the edges of the image would be curved and produce wrong results.

Our solution uses the Harris corner detection method [5] for determining features. Finding corners in an image is a very important prerequisite for the final position determination, because it allows us to calculate a direction vector to another point. This other point is the same feature in the following camera frame, where the Harris detector is used as well.
These raw data have to be refined for the final step.

One important step for the refinement is outlier removal. External movements, like those of other cars on the track, might cause the algorithm to think the camera was moving when in fact the surroundings moved. To filter these unwanted movements an iterative method called *Random Sample Consensus* (RANSAC) is used. The RANSAC algorithm works by

taking a random subset of all the available data, in our case the vectors obtained by the corner detector, and fitting a linear model with a specified neighborhood which contains this subset. All other vectors are then tested against this model and those who fit the model are considered part of the *consensus set*. If enough vectors lie within this model, it is considered as a viable candidate for a linear approximation. Optionally, the model can be re-evaluated using all vectors in the consensus set.

This sequence of events is repeated a fixed amount of times, each time producing a model which is rejected, because it contains too few inliers, or a refined model, if the model consists of more inliers than the best model so far.

Having two sets of viable corners from each image we can track each feature from one frame to another.

The Lucas-Kanade method [6, 7] is a way to estimate optical flow from these sets. It observes a local neighborhood around the detected corners and solves the optical flow equation by the least-squares-criterion.

The optical flow equation for a 2D case states that a pixel at position (x, y, t), with t being the time, and intensity $I(x, y, t)$ will have the same intensity after a displacement by $\Delta x, \Delta y$ and Δt:

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t)$$

Applying the Taylor series, we get:

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + \frac{\delta I}{\delta x}\Delta x + \frac{\delta I}{\delta y}\Delta y + \frac{\delta I}{\delta t}\Delta t$$

It follows that:

$$\frac{\delta I}{\delta x}\Delta x + \frac{\delta I}{\delta y}\Delta y + \frac{\delta I}{\delta t}\Delta t = 0$$

or

$$\frac{\delta I}{\delta x}\frac{\Delta x}{\Delta t} + \frac{\delta I}{\delta y}\frac{\Delta y}{\Delta t} + \frac{\delta I}{\delta t}\frac{\Delta t}{\Delta t} = 0$$

which results in

$$\frac{\delta I}{\delta x}\Delta V_x + \frac{\delta I}{\delta y}\Delta V_y + \frac{\delta I}{\delta t} = 0$$

where V_x and V_y are the velocity in x and y direction, respectively. $\frac{\delta I}{\delta x}$, $\frac{\delta I}{\delta y}$ and $\frac{\delta I}{\delta t}$ are the derivatives of the original pixel (x, y, t), which can be written as I_x, I_y and I_t. Thus the optical flow equation turns out to be

$$I_x V_x + I_y V_y = -I_t$$

This equation is under-determined. However, Lucas-Kanade creates an equation system from all detected points, to determine the optical flow vector (V_x, V_y). It can be written as:

$$\begin{aligned} I_x(q_1)V_x + I_y(q_1)V_y &= -I_t(q_1) \\ I_x(q_2)V_x + I_y(q_2)V_y &= -I_t(q_2) \\ &\dots \\ I_x(q_n)V_x + I_y(q_n)V_y &= -I_t(q_n) \end{aligned}$$

where $q_1, q_2, ..., q_n$ are the detected corners.

If these equations are written in matrix form Av = b, the result is:

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \dots & \dots \\ I_x(q_n) & I_y(q_n) \end{bmatrix}, v = \begin{bmatrix} V_x \\ V_y \end{bmatrix}, and\ b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \dots \\ -I_t(q_n) \end{bmatrix}$$

This system is usually over-determined. Lucas-Kanade now obtains a solution by minimizing the sum of the squared differences of the results to the determined model. In that it solves the 2x2 system

$$v = (A^T A)^{-1} A^T b$$

Where A^T is the transpose matrix of A.

The final computation is:

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} I_x(q_i)^2 & \sum_{i=1}^{n} I_x(q_i) I_y(q_i) \\ \sum_{i=1}^{n} I_y(q_i) I_x(q_i) & \sum_{i=1}^{n} I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_{i=1}^{n} I_x(q_i) I_t(q_i) \\ -\sum_{i=1}^{n} I_y(q_i) I_t(q_i) \end{bmatrix}$$

V_x and V_y are now the approximate displacements of every viable corner from one frame to another and can be used to determine the translation and rotation of the vehicle.

If the taken path includes loops it is possible to optimize the result by using loop detection. If any features are re-detected at a later point in time and the current position is close to the position the features were re-detected, the path can be stretched, so that it connects to the earlier path. This can reduce faulty results that accumulated due to small estimation errors in the previous steps.

3.1.2 Lane Detection

Lane Detection uses markings on a road to determine its lateral boundaries in relation to the recording camera. It is often used in autonomous driving and driving assistance systems, because the position of the car in between lane markings gives a lot of information about the direction the car is traveling. In our case, not only can we determine a traveling direction, but also the position on the lane. By tracking the distance to the left and right lane, a deviation to the middle of the lane can be calculated. If the driven track is known or the camera is capable of determining a scale (e.g. by using a stereo camera), the deviation can even be expressed as a metric length, which is helpful for extracting additional information, like speed and acceleration, later on.
Additionally, not only the racing line can be recorded this way, but also

the layout of the track. This is especially useful to determine an ideal racing line later on, because this often requires knowledge of the exact appearance of the race track.

3.1.3 Recording OBD2 Data

OBD II (On-board diagnostics 2) is an interface which all cars built after 1996 in the USA or after 2003 in the EU, respectively, have to have built-in. It implements the ELM 327 standard which provides PIDs that return specific car-based sensor data, like the current speed, current motor revolution (rpm), throttle position and more. Connections to the interface are made via a Bluetooth- or USB-adapter.
The standard also determines the formatting of the request and return value.
The request consists of the required mode and a Parameter ID (PID), which stands for one specific data value. In mode 1 the dongle returns the current values, which is what we need. Other modes, for example return the values since the last engine failure or information about the car. This request is written to the adapter via a serial port.

$$\underbrace{4}_{statuscode}\ \underbrace{1}_{mode}\ \underbrace{0D}_{PID(hex)}\ \underbrace{37}_{payload(hex)} \tag{1}$$

OBDII example response, returning 55 km/h as the vehicle speed

To make sure that the request arrived correctly, the answer, which will be returned immediately after the request was received, contains some extra data (1). The first character is the status. Status 4 means the request was understood and a answer could be delivered. The second, third and fourth characters repeat the obtained request. After that, the payload containing the data is appended. With a formula, which is also specified in the ELM 327 standard, this return sentence can be decoded to an

integer number, which represents the real result.

PID	Size	Description	Min value	Max value	Unit	Formula
0C	2	Engine RPM	0	16 383.75	rpm	$\frac{256A+B}{4}$
0D	1	Vehicle speed	0	255	km/h	A

Table 1: Excerpt of the available OBD II PIDs, A stands for the first, B for the second Byte

A data request is possible about 10 times per second. This gives us a good basis for further detailed comparisons combined with the racing line recording, as we can determine additional information like acceleration and braking points, which gear to shift to or at which parts on the track the engine is under especially heavy load, which might be interesting in an endurance race.

3.2 Comparing

The second part is the comparison of the previously recorded data. In order to provide an analysis system that has the potential to help racing drivers improve their lap times and beat their opponents, we implemented a comparison method that takes all the available data in consideration and displays them in an understandable way. The process is described in Section 1 Figure 3.

3.2.1 Visual Comparison of Racing Line with help of sensor data

A fairly simple but effective way to find out where time is lost on the track is the visual comparison of racing lines from different drivers. The idea is to overlay the previously recorded lines over the race track. To actually determine at which point a driver was faster than another one, we need

an obvious optical representation of the speed at any given point on the track and the possibility to receive further information on demand. In our system, the speed is represented by coloring in the racing line on a gradient scale reaching from green (standing still) to red (>250 km/h).

To do that, we use the HSV color space. Due to the circular structure of HSV (see figure 7), all we need to do normalize the hue part to green (0 km/h) and red (250+ km/h). The color is then simply a function $f(v) = HSV_{hue}$.
The HSV value has to then be transfered into RGB color space again.

Besides speed data, different values like acceleration, braking behavior or lap time, can be displayed in a similar fashion. That way the fastest line for a given corner can be determined and the slower drivers can find out, why their line was inferior.

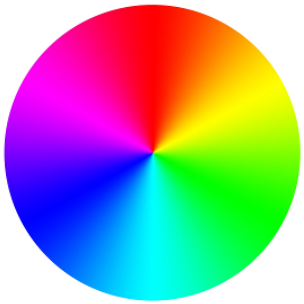

Figure 7: The Hue portion of the HSV color space

3.2.2 Mathematical Calculation of ideal line given a certain track

Besides comparing two driver-generated racing lines, it is also possible to calculate a fast, almost ideal racing line.

Bézier curves are a way to depict an entire race track, i.e. a curved path, with a set of control points, $\{P_0, P_1, ..., P_n\}$ (figure 8. In this definition, n is the order of the curve. P_0 and P_n stand for the beginning and the end

point of the curve, respectively, while the intermediate points define the curvature and don't usually lie on the curve.

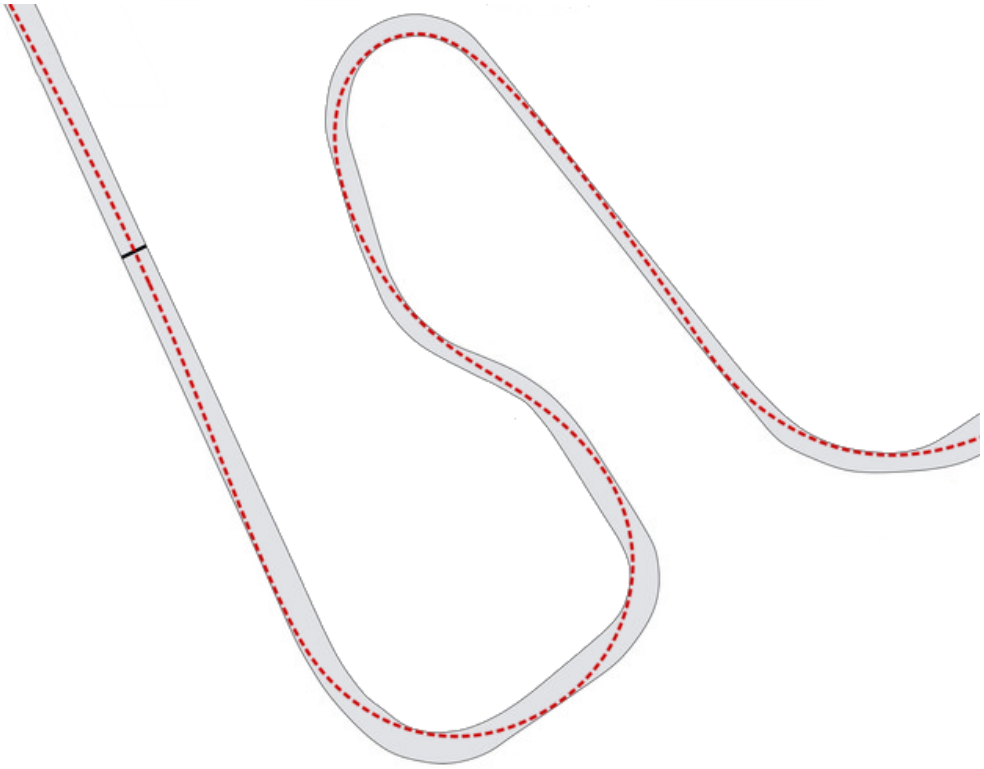

Figure 8: A racing line approximated with a Bézier curve

As we want to get the line with the least curvature, the Bézier curve is capable of creating a fair approximation, especially suitable for racing beginners. Algorithms and methods described in section 2 can be used in order to find the best possible curve around a track.

3.2.3 Mathematical Comparison of two paths

To gain additional and more detailed information about the driven racing line, it is helpful to get mathematical values for specific areas (e.g. the entry and exit angle of curves). This can tell a driver that their line was

too shallow or too wide and provide the potential to improve. For this technique we can use either racing line detection algorithm, even GPS. Every pair of consecutive points is turned into a vector, consisting of the distance between the points and the angle θ to the x-axis (the equator in case of GPS).

$$\Delta x = endPoint.x - startPoint.x$$
$$\Delta y = endPoint.y - startPoint.y$$
$$\overline{xy} = \sqrt{\Delta x^2 * \Delta y^2}$$
$$\theta = \arctan(\frac{\Delta x}{\Delta y})$$

To speed up subsequent calculations all consecutive vectors with the same θ can be combined into one vector by generating a new vector with the sum of the magnitude of the old ones. This indicates a straight where the individual sections are not that interesting. Now the angle of each section of a corner can be compared. Again, the coloring-approach may be used as a simple way to determine the angle at a glance.

4 Implementation

Most of the image processing steps were implemented using OpenCV [6], an open-source C++ computer vision library.
Our system was built in a modular way. Each module fulfilled a specific task. The structure can be seen in figure 9. The `Helper` class handles all the communication with the server, in that it sends the data packets, tells the server which hardware is connected to it, receives information about the current session and performs some more helping procedures. Camera handling is done by the `CameraHelper`.

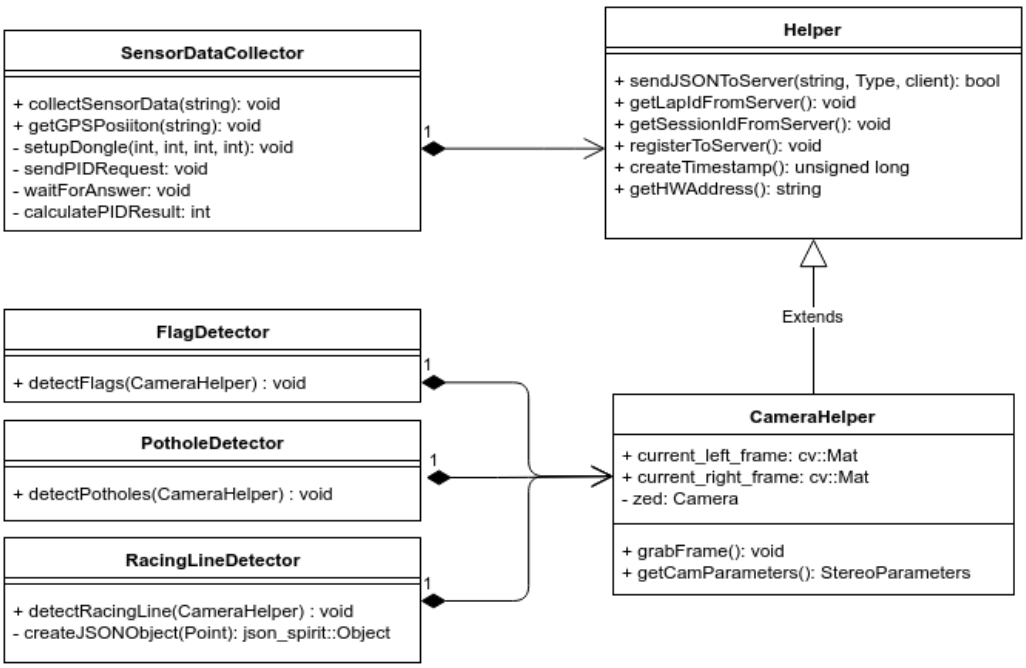

Figure 9: Architecture Diagram

[6] `http://opencv.org/`

The functions that are most important to us are the `SensorDataCollector`, which is responsible for receiving the cars sensor data via the OBD II dongle and the `RacingLineDetector`, which records the racing line of the car with the help of our stereo camera.
`FlagDetector` and `PotholeDetector` are classes that handle the additional features described in Section 1.

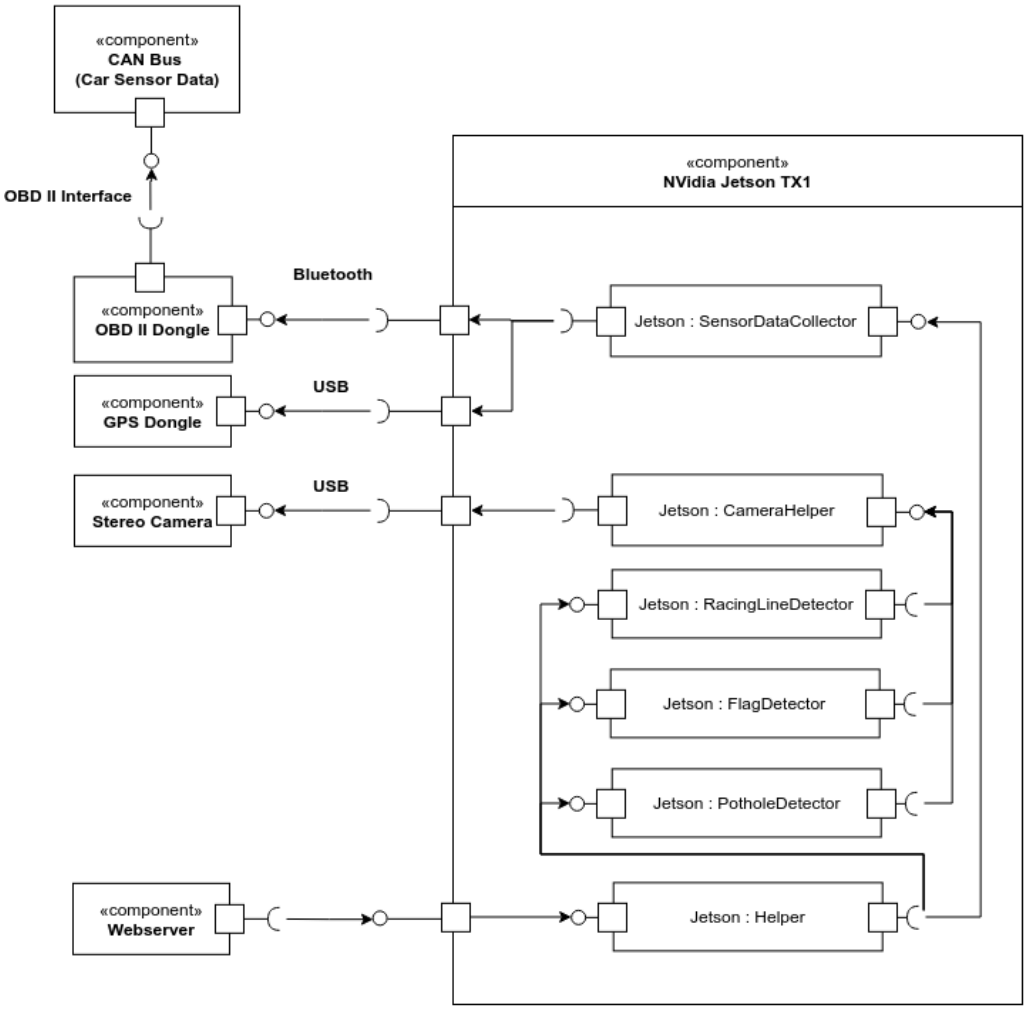

Figure 10: Component Diagram

The communication to our sensors and the server is depicted in the component diagram (figure 10). All the sensor data is received via the cars

CAN Bus and transfered to the OBD II dongle via the OBD II interface. The dongle sets up a bluetooth connection to our Jetson processing unit, where the data is grabbed by our `SensorDataCollector`. Also the GPS Dongle sends its data to that component via a USB connection.
Our `CameraHelper` receives the video data transmitted by the stereo camera via USB and distributes them to the video analysis modules.
Lastly, our `Helper` class collects all the data we extracted and sends them to the web server.

4.1 Recording

We implemented some of the algorithms described in Section 3 in OpenCV and C++, with special focus on the extraction of the racing line from the camera image and the sensor data collection.

4.1.1 Visual Odometry

To extract the Visual Odometry data from the video stream, we used a library called libviso2[7], which was developed at the Karlsruhe Institute of Technology (KIT). The library needs a left and right hand rectified image from a stereo camera. The ZED camera from Stereolabs we used already rectifies the images, so we only need to convert the received stereo image into two data buffers. For that the left and right image is stored in one OpenCV Mat each, converted into grayscale and then transformed into a uchar buffer. To receive the images from the camera, we created a `CameraHelper` class, which centrally grabs the frames and distributes them to the different modules, like the `RacingLineDetector`, `FlagDetector`, etc. Now that the prerequisites are met, we can feed the images to our library. After analyzing the input as described in section 3.1.1, the output will be a 4x4-matrix, the so called pose.

[7] `http://www.cvlibs.net/software/libviso/`

R_{11}	R_{21}	R_{31}	T_x
R_{12}	R_{22}	R_{32}	T_y
R_{13}	R_{23}	R_{33}	T_z
0	0	0	1

Table 2: Pose as returned by Visual Odometry algorithm

The pose consists of the XYZ 3x3 rotation matrix as defined by all R_{nm} and the XYZ translation vector $T_{x/y/z}$.

```
Mat pose; //position relative to last position
Mat motionMat; //position relative to starting point

while(!finished){
  //library processing
  pose = pose * motionMat;

  float diffX = (pose.at<float>(0, 3));
  float diffZ = (pose.at<float>(1, 3));
  Point2d position = Point2d(diffX, diffY);
  //JSON-object creation
}
```

For our purposes the rotation matrix is not that important, because the racing line is only a series of dots. The rotation of the car doesn't influence the result, so all we need is to extract the translation vector, store it in a vector object and write it into a list. This object contains the summation of all translation vectors, making up the position relative to the starting point. At this point it is simply a matter of connecting consecutive dots in the list and drawing the resulting lines. The result will be the recorded racing line.
This racing line can be used on its own, or combined with GPS, as discussed in the Bachelors Thesis "Precise Racing Car Localisation - Improving GPS through Video and Sensors" [8].

4.1.2 Sensordata

The OBD-II dongle uses a bluetooth connection to communicate with our analysis software. Via this connection, which is created automatically at startup by the operating system of our processing unit, we can open a serial port to write and read data from.

Once the connection is established, the serial connection has to be prepared for the specific settings of the dongle. A couple of control commands put it into the state that we can work with:

```
void SensorDataCollector::initialize(int fd) {
  waitForAnswer(fd, "ATZ"); //reset device
  waitForAnswer(fd, "ATL1");  //enable linefeed
  waitForAnswer(fd, "ATSP0"); //autodetect protocol
}
```

The function *waitForAnswer* writes the request to the dongle and returns the answer to an optional buffer. For the initialization this buffer is irrelevant, but as soon as we want to receive data, it is necessary.

```
void SensorDataCollector::
  waitForAnswer(int fd, char *buf, string request) {
  int nbuf = 0;   int i = 0;  const int TIMEOUT = 30;
  bool end_of_answer = false;

  request.append("\r"); //tells the dongle where the request ends
  long writtenBytes = write(fd, request.c_str(), strlen(request.c_str()));
  if (writtenBytes < 0) {
    //errorhandling
  }
  while (!end_of_answer && i < TIMEOUT) {
    long n = read(fd, buf + nbuf, sizeof buf); //read one char at a time
    nbuf += n;
    i++;
    end_of_answer = receivedAnswer(buf); //receivedAnswer checks if last character was a >
  }
}
```

Now, after determining which PIDs are supported by the car (using PID 0100), we constantly loop through all of them and calculate the corresponding value. Together with the description, this value is then sent to our server as a JSON[8] object.

4.2 Comparison

We compared our racing lines using a Google Maps integration in our JavaScript web page. This section describes how the prerequisites mentioned in Section 1.3 are calculated and implemented.

4.2.1 Google Maps

In order to properly identify important parts of the track, we decided to connect our recorded racing line with Google Maps. As our recording approaches can not extract any geographic information, we need at least two GPS points as a reference. One GPS point isn't enough, because that wouldn't specify the direction the car traveled. As the points returned by VO are in meters, we can calculate the corresponding geo-coordinates in relation to that reference point.

For that we need the starting latitude ϕ_1, the starting longitude λ_1, the bearing θ, which is the angle between two VO-points plus the displacement bearing set by the two reference GPS-coordinates, and the distance δ traveled, which is the distance between the two VO-points.

The resulting GPS-coordinates are calculated like this:

```
1 for (var k = 1; k < voData.length; k++) {
2   calculatedGPS[k] = {};
3
4   var d = Math.sqrt(Math.pow((voData[k].x - voData[0].x - xCorrection), 2) + Math.pow((
      voData[k].y - voData[0].y - yCorrection), 2)) * voScaleFactor / 1000;
```

[8] JavaScript Object Notation

```
5   var stdVec = [0, 1];
6   var pointVec = [voData[k].x - voData[0].x - xCorrection, voData[k].y - voData[0].y -
      yCorrection];
7
8   var bearingOfVoPoint = ((stdVec[0]*pointVec[0])+(stdVec[1]*pointVec[1]))/(Math.sqrt(Math.
      pow(stdVec[0], 2)+Math.pow(stdVec[1], 2))*Math.sqrt(Math.pow(pointVec[0], 2)+Math.pow(
      pointVec[1], 2)));
9   bearingOfVoPoint = Math.acos(bearingOfVoPoint);
10  bearingOfVoPoint = (pointVec[0] >= 0) ? bearingOfVoPoint : (2*Math.PI)-bearingOfVoPoint;
11  bearingOfVoPoint += rotation;
12
13  var lat2 = Math.asin( Math.sin(lat1)*Math.cos(d/R) + Math.cos(lat1)*Math.sin(d/R)*Math.
      cos(bearingOfVoPoint) );
14  var lng2 = lng1 + Math.atan2(Math.sin(bearingOfVoPoint)*Math.sin(d/R)*Math.cos(lat1),
      Math.cos(d/R)-Math.sin(lat1)*Math.sin(lat2));
15  lat2 = deg(lat2);
16  lng2 = deg(lng2);
17
18  calculatedGMapsPoints.push(new google.maps.LatLng(lat2, lng2));
19 }
```

In order to depict the sensor data, we color in every line segment according to a two-color gradient. Because of that, we get a lot of line objects, which make further processing difficult. Our solution was to use an additional hidden polyline, that contains all the line segments. This polyline is wider than the normal lines, to make hovering on it easier and the fact, that it is one single object, improves performance and enables us to easily calculate the length of the racing line as well.

5 Evaluation

In this section the results we observed using camera based racing line detection in comparison to the traditional method with GPS are discussed. The second part investigates whether and why it is sensible and useful to have a comparison of two drivers racing lines or a calculated ideal line.

5.1 Visual Odometry to record Racing Lines

Visual Odometry has many theoretical advantages over GPS. The amount of data points received is a lot higher, as it produces 1 position per frame, which could realistically result in 30 - 60 positional updates per second, wheres GPS usually produces 1 point per second. This high amount of points makes a detailed comparison and analysis a lot easier, as curves appear mostly smooth instead of having only a couple of straight lines, as seen in figure 11.

Consequently, VO is capable of detecting very small movements. A racing car driving at 100 km/h travels 28 meters forward every second. The lateral movement, i.e. driving from one side of the track to the other or through corners, is much slower. Assuming a driver takes 2 seconds to cross a 10 m wide track, we would get one positional update every 17 cm at 30 frames per second.

Obviously, this is 30 times better than GPS because of the higher speed alone, but also the GPS standard (NMEA 0183) is only accurate to $\frac{1}{60000}$ of a degree. The distance between two degrees of latitude is 110 km, the distance between two meridians at 51° (position of Berlin) 70 km. That means the accuracy in North-South direction is 1.8 m and in East-West direction 1.2 m. If the Region of Interest (ROI) of our camera image has a real-world width of 30 m and the recording has Full-HD resolution, the accuracy of VO would be 3 cm.

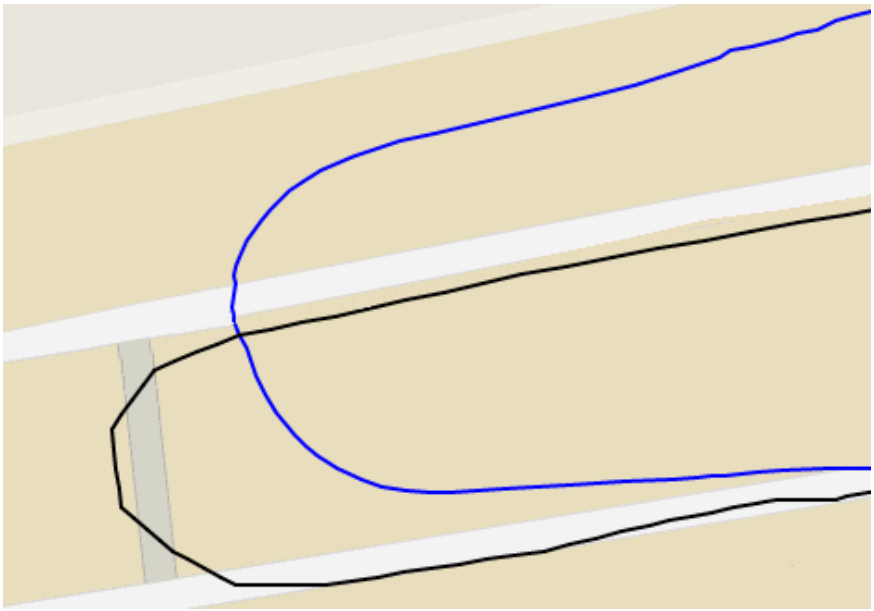

Figure 11: Visual Odometry (blue) produces much smoother curves than GPS (black), but can be erroneous

Another advantage of Visual Odometry is that it doesn't rely on external sources, like satellites, and is in general mostly independent of the area it is used. After all, even the Mars Rovers [9] used this algorithm to navigate. Especially in forests, rural areas and tunnels, GPS is either inaccurate or not usable at all. The camera-based approach still works when GPS fails.
So all in all, VO is able of extracting a very detailed depiction of the racing line, even in remote areas. However, the algorithms used are fairly expensive. A mobile-range computer (like the NVidia Jetson TX1) can achieve an average frame rate of about 8 frames per second at a resolution of 672 x 376 px. This is in theory still better than GPS, but it is an average value, potentially becoming a lot slower in feature-dense areas, like forests, and especially corners. The reason VO slows down in corners is that, rather than tracking already existent features points it has to search new points much more frequently than on straights, because the

scenery changes much quicker.
Additionally, VO can only determine a relative difference to the last calculated position. This means, that potential errors, like too shallow corner angles, will result in faulty results throughout the entire path. This can also be seen in figure 11. The incoming and outgoing straights should be close to parallel, but the VO image detected them at an angle. If this turn would be repeated, the error would cause the line to not be consistently at the same position every time, without further processing.

Most of these limitations are caused by the desire to achieve a real-time capable solution. If speed is not an issue, the error rate can be reduced a lot. So, for a quick draft it is sensible to use GPS, however if it is possible to record the drive and sufficient to analyze it at a later time, VO creates more accurate results.

5.2 Usefulness of Racing Line Comparison

Professional drivers can usually handle their cars very well, however only the best drivers have the feeling and intuition for the fastest racing line. Studies showed, that techniques like trail braking, which means that the driver breaks into the corner, until they reached the apex, in contrast to doing all the braking on the straight, improve lap times by up to 5%, which makes a difference of 4 seconds on an 80 second lap [2]. Being able to compare braking points between drivers and a calculated optimal line can therefore make the difference between winning and losing.
In addition to the possibility of improving ones lap time, especially amateur and intermediate drivers can experience an even more competitive race. Now the award for driving fast is not only a fast time, but also the visual confirmation, that shows where and why a driver was the best, plus it increases the challenge, as weaker drivers can improve faster.
A gamification element, like a level that reflects the drivers skill, could be introduced to further expand this criterion, as described in Section 6.1.

6 Future Work

6.1 Gamification

So far drivers already have a simple and accessible way of comparing each other, however our method only provides a direct relation between two drivers. Unless a driver looks at all other drivers racing lines, they can't certainly asses if they drove a particularly fast lap or if the opponent was just slow in general.
Thats why a form of gamification could be implemented. Based on different factors, like cornering, control and boldness, a score could be determined that assigns a level to a driver. That way drivers could compare, even if they didn't drive on the same track and races could include drivers of similar skill levels to make them as intense as possible.

6.2 Real-Time Comparison

Currently, the recording of the racing line can not be achieved completely in real-time with hardware that is power-conscious enough to run with the power provided by a cars battery, as the most powerful hardware that can be considered to fulfill this requirement is the NVidia Jetson TX1, which we used. However, current implementations of Visual Odometry tend to not use the full capabilities of the device, namely they only run on the CPU, while the GPU[9] is running idle. If, either using more powerful hardware or very specifically optimized algorithms, VO could be implemented to run in real-time, i.e. at 30-60 frames per second, this would make it possible to determine an exact location of the car at any point of the race, without the delay of sending the video data to a central, powerful processing unit or the inaccuracy of having to skip frames.
The position could then be transmitted directly to other cars and their

[9] Graphics Processing Unit

drivers to inform them about how they compete in relation to all other drivers. Such an overview could even be shown in a heads-up display directly on the windscreen or, in order to not disturb the driver too much, by informing the team manager of the driver, who can then tell the driver. The advantage of this process is, that no expensive measurement equipment and, especially, no external dependencies (apart from the communication between the cars) are needed, so it could be used anywhere.

6.3 Improving Accuracy

The biggest problem of Visual Odometry is that errors can not easily be removed, as only a relative position to the last point is calculated. If there is an error in this calculation, all other points will also be faulty. One idea is to use occasional GPS measures to get an idea of how far off the recorded racing line is from the actual path. For example, one GPS measurement at the beginning and end of a curve could be used to calculate the actual corner angle and warp the path so that it fits the GPS measurements.

7 Conclusion

In order to develop an accessible racing driver analysis we explored a variety of methods to record and assess the driving behavior of racers. As our system should have as little external dependencies as possible, we looked for methods to do all of the analysis inside the car, which is why camera-based methods were used to record the racing line and race track.

We used the cars on-board sensor data to obtain further information about the driving and consequently enable us to answer complex questions, like the ideal braking and acceleration points on the track.

This made it possible to compare two drivers and discover their strengths and weaknesses and how to tackle them to become a better racing driver. Alternatively, an ideal racing line could be calculated, even without prior knowledge of the track with the help of lane detection. This could help even professional drivers improve and achieve best times.

Our test drives showed that the Visual Odometry algorithm we used to record the racing line had its advantages and disadvantages. The fact that it runs on video material means, that the analysis doesn't have to take place at the time of the drive. That makes it possible to do the calculations after a race. Consequently we are able to get a lot more positional updates than with GPS, because one update can be calculated per frame, which leads to about 30 positions per second, in contrast to 1 position per second using GPS. However, the fact that the calculations are fairly expensive makes the algorithm only partly usable for a real time use case.

Further improvements can be made through the addition of a gamification system, that gives a level to a driver based on their skill. Also with the help of a more optimized implementation a real-time comparison could be achieved, that could tell a driver at any time how they compare to another driver, e.g. displaying it on a heads-up display.

References

[1] L. Cardamone, D. Loiacono, P. L. Lanzi, and A. P. Bardelli, "Searching for the Optimal Racing Line using Genetic Algorithms," in *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, pp. 388–394, Aug 2010.

[2] T. Gustafsson, "Computing The Ideal Racing Line Using Optimal Control," Master's thesis, Linköpings tekniska högskola, 2008.

[3] D. L. Brayshaw and M. F. Harrison, "A Quasi Steady State Approach to Race Car Lap Simulation in Order to Understand the Effects of Racing Line and Centre of Gravity Location," in *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, pp. 725–739, 2005.

[4] Y. Xiong, "Racing Line Optimization," Master's thesis, Shanghai Jiao Tong University, 2009.

[5] C. Harris and M. Stephens, "A combined corner and edge detector.," Citeseer, 1988.

[6] B. D. Lucas, T. Kanade, *et al.*, "An iterative image registration technique with an application to stereo vision.," in *IJCAI*, vol. 81, pp. 674–679, 1981.

[7] C. Tomasi and T. Kanade, *Detection and tracking of point features*. School of Computer Science, Carnegie Mellon Univ. Pittsburgh, 1991.

[8] N. Hoffmann, "Precise Racing Car Localisation - Improving GPS through Video and Sensors," 2016. Bachelor's Thesis.

[9] M. Maimone, Y. Cheng, and L. Matthies, "Two years of visual odometry on the mars exploration rovers," *Journal of Field Robotics*, vol. 24, no. 3, pp. 169–186, 2007.