Bachelor's Thesis

Racing Line Detection - Recording and Comparison of Racing Lines

Ideallinienerkennung - Aufnahme und Vergleich von Ideallinien

by

Tim Oesterreich

Potsdam, June 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 21, 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 to 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 parameter 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 many 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 many data the car produces, like speed, throttle position and safety relevant features, like engine temperature and 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 observe specific areas and 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 to not 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 so that they 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, 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, 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) 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 overview so every driver had a detailed overview over their recorded laps.

1.3. Problem Introduction/Description/Context

2. Related Work

2.1. Racing Line calculation

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.

2.1.1. Vamos Racing Simulator

One example is the *Vamos racing simulator*. 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.

2.1.2. RaceOptimal

RaceOptimal 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 physics of the cars. 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 and 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 capabilities of the tires. Lastly, 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 consistent.

3. Algorithm/Concept

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

3.1. Recording

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 isn't dependent on any external sources, like satellites or radio towers, to determine the position of the object.

Also it is 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. 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 for determining features. Finding corners in an image is a very important prerequisite for the final position determination, because it allows us to calculate direc-

tion vector to another point. This other point being 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.

Outlier removal is an important stop, because otherwise external movements, like those of other cars on the track, might cause the algorithm to think the camera was moving in a different direction than it actually was. To do that 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 either image we can track each feature from one frame to another.

The Lucas-Kanade method 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 Lukas-Kanade creates an equation system from all detected points, to determine the optical flow vector (V_x, V_y), that 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:

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

This system is now (usually) over-determined. Lucas-Kanade now obtains a solution by minimizing the sum of the squared difference 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:

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

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 know 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.

- possibility to improve recording results on known track
- possibility to determine accurate horizontal position on track

3.1.3. Recording OBD2 Data

Definition OBD II (On-board diagnostics 2) is an interface which all cars build 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 standardized 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.

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

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 in Bytes	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.

3.2. Comparing

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 the other 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). 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 driver can find out, why their line was inferior.

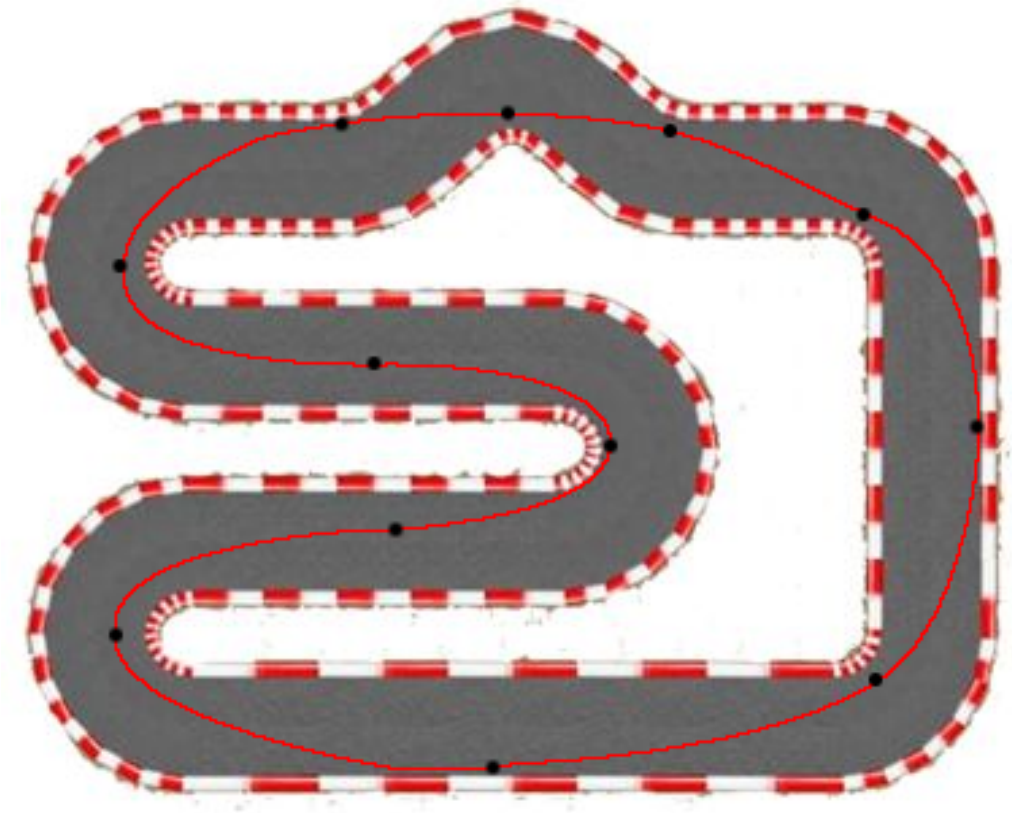
- highlight relevant locations, like corners, showing detailed information about throttle and brake behavior and curvature through the corner
- fastest way through corner is trade-off between distance traveled and curvature, with low curvature allowing the driver to carry more speed to exit of the corner

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 Bézier curves are a way to depict an entire race track, i.e. a curved path, with a set of control points, $\{P_0, P_1, ..., P_n\}$. 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 of the curve and don't usually lie on it.



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.

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).

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

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 aren't 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, an open-source C++ computer vision library.

4.1. Recording

4.1.1. Visual Odometry

To extract the Visual Odometry data from the video stream, we used a library called libviso2, 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 each one OpenCV Mat, 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.

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.

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 object.

4.2. Comparison

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't extract any geographic information, we need at least two GPS point 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:

```
for (var k = 1; k < voData.length; k++) {
  calculatedGPS[k] = {};

  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;
  var stdVec = [0, 1];
  var pointVec = [voData[k].x - voData[0].x -
     xCorrection, voData[k].y - voData[0].y -
```

```
    yCorrection];

  var bearingOfVoPoint =
    ((stdVec[0]*pointVec[0])+(stdVec[1]*pointVec[1]))/(Math.sqrt(Math.pow
    2)+Math.pow(stdVec[1],
    2))*Math.sqrt(Math.pow(pointVec[0],
    2)+Math.pow(pointVec[1], 2)));
  bearingOfVoPoint = Math.acos(bearingOfVoPoint);
  bearingOfVoPoint = (pointVec[0] >= 0) ?
    bearingOfVoPoint : (2*Math.PI)-bearingOfVoPoint;
  bearingOfVoPoint += rotation;

  var lat2 = Math.asin( Math.sin(lat1)*Math.cos(d/R) +
    Math.cos(lat1)*Math.sin(d/R)*Math.cos(bearingOfVoPoint)
    );
  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));
  lat2 = deg(lat2);
  lng2 = deg(lng2);

  calculatedGMapsPoints.push(new google.maps.LatLng(lat2,
    lng2));
}
```

$$\phi_2 = asin(sin\phi_1 * cos\delta + cos\phi_1 * sin\delta * cos\theta)$$

$$\lambda_2 = \lambda_1 + atan2(sin\theta * sin\delta * cos\phi_1, cos\delta - sin\phi_1 * sin\phi_2)$$

- Polyline Overlay
- hidden Polyline for calculations and hover function vs visible, colored Polyline

5. Evaluation

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 1.

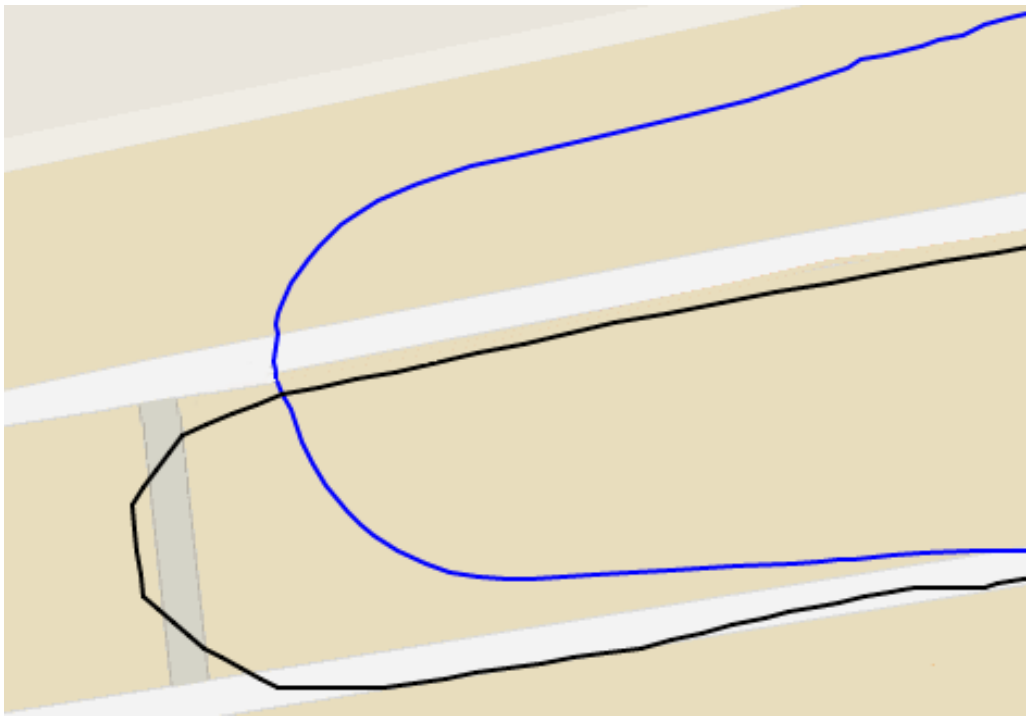


Figure 1: Visual Odometry (blue) produces much smoother curves than GPS (black), but can be erroneous

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. So, 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. 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 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 1. 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 achieve similar lap times with very different racing lines
- however amateur and intermediate drivers can learn a lot from more skilled drivers

6. Future Work

6.1. Open Research Questions

6.2. Extensions

7. Conclusion

References

A. Appendix