Map Generation in Autonomous Racing

A Comparision of a Classic Heuristical Algorithm and Machine Leaning

Alexander Seidler

Bachelor's Thesis

Department of Computer Science
Multimedia Information Processing Group
Kiel University

Advised by: Prof. Dr. Reinhard Koch
Lars Schmarje, M.Sc.

March 2022

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig und ohne fremde Hilfe angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Die eingereichte schriftliche Fassung der Arbeit entspricht der auf dem elektronischen Speichermedium.
Weiterhin versichere ich, dass diese Arbeit noch nicht als Abschlussarbeit an anderer Stelle vorgelegen hat.

Alexander Seidler
01. 03. 2022

Abstract

- advancing technology in automation of driving and in controlled environment racing - reconstruction of abstract racing map from camera and lidar input, using slam output or using direct output - implemented in two ways a classical approch using foo bar and heuristics - and machine learning approach using mlp, cnn, etc.

Acknowledgements

Optionale Danksagungen

Table of contents

Todo list

Chapter 1

Introduction

1.1 Motivation

Automation plays an essential role in the development of modern transport, as automation is the natural direction to take on in the seek of increased safety, efficiency and passenger comfort [11]. Autonomous racing sets a competition driven framework for the exploration of autonomous driving which incentivizes new innovations to take place. Thereby racing often sets the starting point for innovation to take over the whole industry pushing progress further [6]. One example of such competition is Formula Student Driverless (FSD).[1] FSD challanges teams across several countries to build cars that can atonomously drive around fixed tracks that are defined by different colored cones. One car is racing at a time and is competing for the fastests lap rounds.

The Problem of autonomous racing in this context can be split into three main parts, landmark detection and tracking, map generation and trajectory planning, and controlling the vehicle. The first step in autonomously driving a vehicle is to generate an abstract representation of its surrounding, to do this sensory input such as camera images, LIDAR data and odometric input from inertial measurement units is used to create and track landmarks in a virtual space and locate them relative to the vehicle. This task can be accomplished by simultaneous localization and mapping (SLAM) algorithms [14] and is not part of this thesis. On the other side the controlling of the vehicle uses specific driving parameters such as desired velocity and steering angle to control the various actuators, e.g. motors, that move the vehicle. This problem is very similar to the controlling of non-autonomous manually driven vehicles, since the main difference is the driving parameters coming from sensors like the acceleration pedal

[1]https://www.formulastudent.de/teams/fsd/

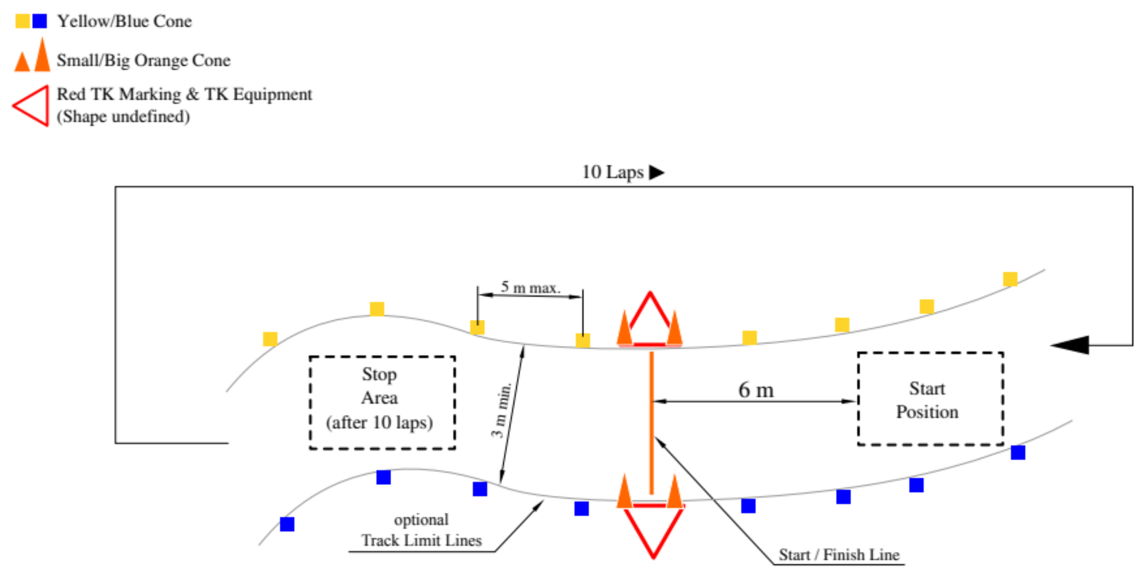

Yellow/Blue Cone
Small/Big Orange Cone
Red TK Marking & TK Equipment (Shape undefined)
10 Laps ▶
5 m max.
Stop Area (after 10 laps)
3 m min.
6 m
Start Position
optional Track Limit Lines
Start / Finish Line

Fig. 1.1 Layout of an FSD track (Source: FSG21 Competition Handbook, p.14, "Figure 2: Trackdrive")

and steering wheel in manual driving as opposed to the output of a processing pipeline in autonomous driving. This is also not part of this Thesis. The problem that is left to solve is using the virtual space provided by the SLAM to determine the driving parameters velocity and steering angle. This problem can be split into two parts. Map generation, which focuses on transforming information about landmarks into an abstract map of the racing track. And trajectory planning, which uses the abstract map to plan actions that will lead the vehicle to move along the track. This thesis looks at an extension of a previously worked on classical algorithm for map generation and a novel machine learning approach to solve map generation and trajectory planning in one step and systematically compares these two approaches.

1.2 Goals

Raceyard is a Kieler Team aiming to compete in FSD and sets the framework for the implementation and application of the ideas presented in this thesis. As of writing this thesis a simplistic classical approach to map generation is used at Raceyard which imposes several problems which make the algorithm not yet useable for autonomous driving. For three of these problems this theses suggests an improvement. These are:

- Robustness against the incorrect detection of the color of landmarks (misdetection), missing landmarks completely (non-detection) and detection of landmarks twice or more with one detection being at the wrong place (over-detection): Using the current approach only some misdetections can be automatically corrected, any misdetection that can't be corrected renders the resulting map completely unusable. Also, non-detections are completely ignored, with leads to problems especially in narrow curves while over-detections are handled like normal landmarks leading to wrong predictions as well.

- Using the certainty the SLAM provides: The SLAM assigns covariances representing the certainty in x and y direction to each landmark detected, this covariance is completely ignored by the current algorithm, although it could be beneficial to use.

- Runtime: The current approach takes orders of magnitudes too long to be used in real time

1.3 Related Work

Given the ever-growing field of autonomous driving an abundance of literature can be found about autonomous racing. However, all those works focus on key aspects that differ from this thesis in one or more ways.
With regards to the classical approach to map generations several techniques have been documented. The following Papers apply a classical algorithm specifically to the Problem of autonomous racing in FSD. AMZ Driverless [9] as well as Andresen et al. [2] focus on an architecture using an ordinary SLAM in conjunction with a Delaney triangulation do to path planning. Zeilinger et al. [18] as well as KIT19d [13] use an extended Kalman filter (EKF)-SLAM to derive the center line for trajectory planning directly. Also, these papers do not take a look at Machine learning as an alternative for path planning.

In Machine Learning some approaches to autonomous racing can be found, however none of those apply machine leaning (ML) to the problem of map generation and path planning in FSD specifically. Dewing [4] applied used a convolutional neural network (CNN) solve autonomous driving in a racing game. While Dziubiński [5] used a CNN for steering a toy car in free terrain without cones to mark the path.

One notable exception that applied machine learning to the problem presented in FSD specifically is the work of Georgiev [7]. Georgiev implemented Williams et al. [16] model predictive path integral (MPPI) in the Formula Student racing environment. MPPI uses a path integral over several possible trajectories to derive the best possible future trajectory in path planning. A Neural Network is used to train the parameters of the MPPI.

To the knowledge of the author, no full ML approach has been made specifically in the context of map generation in FSD. Also, no comparison to a classical approach in FSD has been conducted. This work evaluates a modified classic heuristic Algorithm in comparison to a ML approach in the context of FSD racing.

1.4 Thesis Structure

In the following chapter basics and technical background is explained surrounding the two approaches and autonomous racing in general.
Thereafter, in the third chapter the details of the classical and ML approach, as well as their implementation is presented.
In the 4th chapter the approaches are evaluated and compared, and in the last chapter the results are summarized and several improvement ideas and ideas for future work are listed.

Chapter 2

Foundations and Technologies

2.1 Raceyard and Formula Student

Formular Student is global competition for building racing cars. The subclass FSD is focused on autonomous driving and is spit into different disciplines. Whereof Autocross is the most relevant for this thesis. The goal in Autocross is to drive a previously unknown track for one lap as fast as possible, so all data about the track must be gathered and processed in real time with no prior map. Since 2005 Raceyard is the Team from Kiel for Formula Student and aims to compete in FSD in the upcoming competitions.

Fig. 2.1 The T-Kiel A CE, one of Raceyards latest cars (Source: https://raceyard.de/autos/)

2.1.1 The Rosyard Pipeline

The software that is to be used in FSD by the Raceyard car is called "Rosyard" which is build on the Robot Operation System (ROS) [15]. In ROS processing takes place in nodes which can communicate with each other using data channels called topics. The Nodes can be written in python or C++ and are connected in a way that forms a pipeline in a feed forward fashion. The pipeline reads sensory data in the beginning and outputs in the end control data for the vehicle. The pipeline consists of 5 stages which are each represented by a node:

1. detection: sensory input from cameras
2. slam: extract landmarks and locate them in virtual map
3. estimation: estimate centerline to follow in map
4. driving: given map data decide steering and velocity
5. lowlevel: hardware controlling

Missing figure

ros pipeline visualized with topics

This thesis focuses on the implementation of the 3rd stage. Given the landmarks located in a virtual map from the SLAM this node should estimate the course of the track, such that the 4th stage can successfully drive the car along the track. The pipeline is fully dockerized and runs in 4 different docker containers: the master node coordinating everything in ROS, an optional visualization container, a simulation container for providing fake sensory input, and a container running all 5 pipeline nodes.

2.2 Machine Leaning

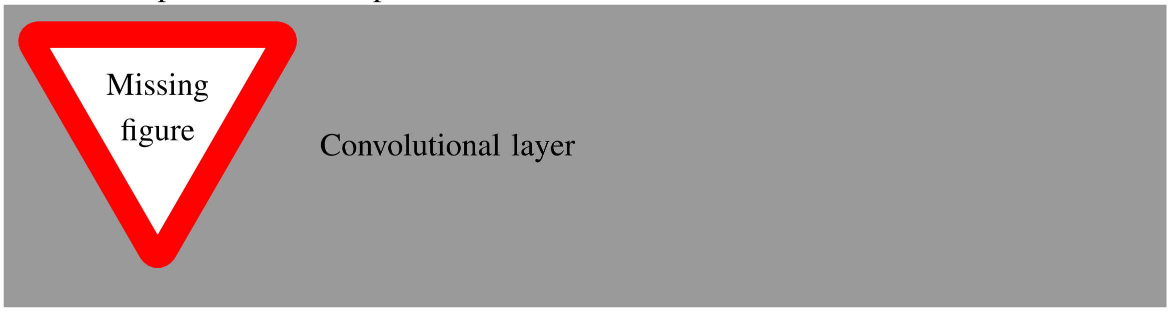
ML describes a class of algorithms that have the ability to improve automatically, this process of improving is known as learning. Three different categories of learning can be distinguished, supervised learning, unsupervised learning and reinforcement learning. In supervised learning a set of labeled data, called training data is used to improve the parameters of the algorithm to make it predict labels better without explicit programming. Supervised learning can be used to train artificial neural networks. A neural network (NN) can be modelled as a directed graph consisting of artificial neuron as nodes and connection between neurons as edges. One example for artificial neurons are perceptrons. A perceptron is an abstract and mathematically easy to compute model of a biological neuron. A perceptron receives a number of inputs x and using the weights of the inputs w calculates their weighed sum $z = w \cdot x$, and passes it through an activation function f. This leads to the output $y = f(z)$ which is called the activation of the perceptron. Common activation functions include linear $f_{linear}(x) = a \cdot x$ for some factor $a \in \mathbb{R}^+$ (commonly 1) and Rectified Linear Unit (ReLU) $f_{ReLU}(x) = max(a,x)$.

2.2.1 Deep Learning and Multilayer Perceptions

Multiple Perceptrons can be arranged in layers to form a special kind of NN, called multi layer perceptron (MLP). In such a layer a perceptron may only have a connection to perceptrons in the directly succeeding layer. A layer that has the maximum number of connections to the previous layer, such that each neuron is connected to each neuron in the previous layer is called fully connected layer. A MLP consists of an input layer an output layer and a variable number of so-called hidden layers in between the input and output layer. By having at least 2 hidden layers the decision boundary of a MLP can take an arbitrary form, allowing it in theory to solve arbitrarily complex problems as opposed to a single perceptron which can only solve linearly separable problems [10]. In recent years the research primarily focuses on networks with an even greater number of hidden layers. Such networks, with a big number of layers are called deep networks. Since Deep networks most often use non-linear activation functions the optimal weights cannot be found analytically, other algorithms for learning must be used, called deep learning. One of those algorithms is backpropagation which uses gradient descent to learn the weights as an optimization problem of the weights in respect to the desired output.

2.2.2 Convolutional Neural Networks

In CNNs the concept of MLPs is extended by adding convolutional and pooling layers in a NN. Convolutional layers allow for processing a big number of inputs while not imposing a huge number of learnable parameters as a fully connected layer would. Having this property convolutional layers are ideal for processing images, as even small images e.g. a 32x32 RGB image already has 3072 inputs. A convolutional layer uses a number of weights matrices called kernels of a fixed small size (e.g. 5x5). These kernels are convolved across the inputs width and height, meaning the dot product of the filter and a specific local region is computed for each input thereby computing a two-dimensional map of that kernel. The weights of the kernels can be learned using backpropagation, while certain hyperparameters must be set when designing the NN. One of such parameters is the size and number of kernels used. Another hyperparameter is by how many pixels the kernel is "moved" after each calculation, thereby skipping pixels as center for the kernel. This hyperparameter is called stride. Around the edges the input needs to be padded (usually with zeros) so that the edges of the input can be processed as well. A pooling layer reduces the number of inputs by partitioning the input along the width and height into equal size chunks (e.g. 2x2) and computing an output for each of these chunks. Some commonly used pooling is max pooling, calculating the maximum of its inputs, and average pooling, calculating the arithmetical mean. Often, convolutional and pooling layers are succeeded by fully connected layers which are then used to compute the final output of a network.



2.3 Discrete Curvature

Discrete curvature applies the concept of curvature from a continuous curve to a discrete curve called a polyline.

A polyline is a series of line segments and is determined by a sequence of points $(P_0, ..., P_n)$ $n \in \mathbb{N}$ where each line segment connecting a pair of adjacent points $[P_i, P_{i+1}]$ $i \in \mathbb{N}_{\leq n}$ forms a vertex in the polyline.

In the continuum [source:wiki] "the curvature at a point of a differentiable curve is the curvature of its osculating circle" which more formally can be defined in terms of the unit tangent $\vec{T}$ and the arc length s: [1]

$$\kappa = \left\| \frac{d\vec{T}}{ds} \right\|$$

This definition however cannot be used directly to determine the curvature of points in a polyline, given its non-continuous nature. All straight segments would have a curvature of 0 while the curvature in the edges would diverge to infinity. A new definition must be used to determine the curvature of a series line segments, which in turn can be used to approximate this series. A different definition can be derived from the quotient of the circular angle and the arc length:

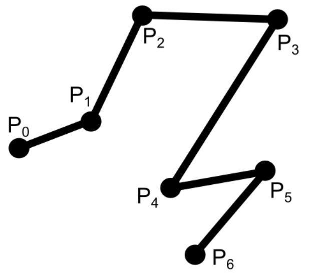


Fig. 2.2 A polyline over the vertices P_0 to P_6

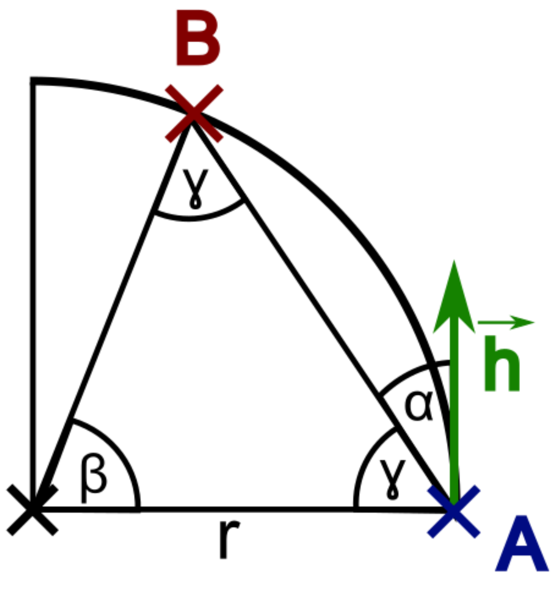


Fig. 2.3 Points A with heading $\vec{h}$ and B in circle with radius r, implying a curvature in point A of $1/r$. The circle center and B and A form an isosceles triangle with base angle γ and vertex angle β

$$\kappa = \frac{d\varphi}{ds}$$

Using this idea we can define the curvature from a point A, a heading $\vec{h}$ in that point and a point B as the reciprocal of the radius of the circle passing though A and B and being tangent to $\vec{h}$ in A.

Thus, we can calculate the reciprocal of the radius of this circle as follows:

Since $\vec{h}$ is tangent $\gamma = 90° - \alpha$ and $180° = 2\gamma + \beta$ thus $\beta = 2\alpha$

The length of the secant $s := |\vec{AB}|$ can be calculated as $s = 2r \cdot sin(\frac{\beta}{2}) \Leftrightarrow \frac{1}{r} = \frac{2sin(\alpha)}{s}$

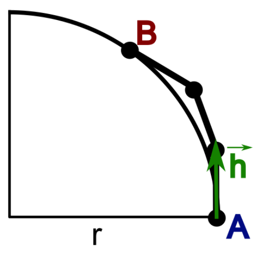


Fig. 2.4 Example curvature of $1/r$ *approximating a polyline leading from A to B, the circle corresponding to the curvature has the radius r*

Using this method we can calculate the average curvature of the curve that is tangent in A to $\vec{h}$ and passing though B, which approximates the polyline connecting these points. The heading $\vec{h}$ can also be derived using the next point after A leading to B. Doing this for different distant points B on a polyline gives us a suitable approximation for the course of a polyline starting from point A. Of course this neglects the shape of the polyline completely, which fails to detect S-curves between point A and B, this however imposes no problem if we choose a fairly small distance between point A and B such that the variance of the curvature for intermediate points is non-significant.

2.4 Simultaneous Localization and Mapping

SLAM algorithms solve the chicken-and-egg problem localizing an agent in a map and mapping the environment surrounding an agent. Since for localization seemingly a map is needed and for creating a map of the surrounding the position of an agent needs to be known, the natural solution is to solve both simultaneous. While an exact solution is often not possible / or desirable computation cost wise, several methods exists that can approximate the problem. These Approximations for example use EKF, graphs, or particle filters. The SLAM used as input for the approaches in this thesis is an implementation of FastSLAM [12] which is based on particle filters. In FastSLAM particles are used as potential positions for the agent, at each time step a weight is assigned to the particles according to their likelihood of being consistent with the sensed nearby landmarks. Next new particles are created according to the spacial distribution of weight thereby converging to the actual position. In any time step the particle with the biggest weight is guessed as the actual current position and reported as such. This leads to the problem of jumping in the virtual space when the particles diverge to two or more different positions and the previous most likely position becomes less likely than another distant position. When this occurs the generated map along the estimated position jumps in a non-continuous way. This also imposes the problem that landmarks cannot be identified consistently across time, since every particle keeps track of its own landmarks and once the estimated position jumps the landmarks cannot be associated to the previous landmarks because the transformation is non-continuous. The output of FastSLAM is the incrementally build map of landmarks in relation to the estimated position of the agent. The landmarks have an uncertainty in the x and y dimension associated with them in form of a covariance matrix. This can later be used to filter for accidental detection of landmarks.

2.5 Development Environment Used

For developing the approaches web technologies were used a development environment, as this allows for fast prototyping and easy building of a visual interface and visual output. Additionally, this makes the prototypes easily sharable as they can be hosted on a web server and be accessed via browser. Specifically the JavaScript model-view-viewmodel framework vue.js[1] was used. Since the main source code of raceyard as well as the previous algorithm is written in python the ability to run python code was crucial for the development as well. While one possibility was to use a dedicated server run python code with specific parameters that reports the result back to the web application, a web integrated solution would be more desirable.

2.5.1 Pyodide

Pyodide is a port of CPython to WebAssembly [3] which allows the execution of python code directly within a browser using WebAssembly. As Opposed to other systems Pyodide doesn't cross compile python to JavaScript but uses a python runtime to execute python code on demand. Also, many of the most used scientific python libraries, e.g. NumPy, SciPy, Pandas and Mathplotlib are supported out of the box, which makes it useable for many python scripts without modification. The non-native execution, however, comes at a performance cost of running at about 2x to 10x slower than native python, depending on the amount of C code used in packages [17][8]. Adding Pyodide to the dev environment allowed the Web

[1]https://vuejs.org/

application to be served completely statically, which meant that it could be published on a static website hosting service such as GitHub Pages[2].

[2]https://dsalex1.github.io/BachelorThesisRaceyard/

Chapter 3

Methods

3.1 Classical Approach

3.1.1 Basis - Master Project by Vaishnav/Agrawal

The basis for the classical algorithm is the Master Project of Vaishnav and Agrawal based on master project Vaishnav/Agrawal

Schwächen bsheriger Algorithmus, Rot anstatt Gelb, Schwarz nicht klassifizierte Punkte, Grau gänzlich nicht detektierte Punkte, schwarze Linie ground truth, Der Algorithmus macht die grüne Linie daraus. wo die Pylonen perfekt erkannt wurden, grüne Linie exakt auf ground truth, da wo die Farbe nicht erkannt wurde, passieren ganz komische dinge und da wo oben rechts gar keine Roten erkannt wurden weicht es sehr vom eigentlichen Track ab. Eingabekarte: https://gyazo.com/645d505ed8a7b1ae5bd6ac8d6a867348 ERgebnis: https://gyazo.com/43237dba0fb02314743c860a9360e520

0.5p

3.1.2 First Improvement - Guessing Missing Points

1. readding missing points

erster Ansatz, Punkte auf einer Seite fehlen, also orthogonal zur Tangente eines Pylons mit dem Abstand der Trackbreite kein andersfarbiger Pylon, dann ergänzt https://gyazo.com/385776842 78cf937f5

2p

3.1.3 Second Improvement - Covariance Filtering

2. ansatz Covarianzen rumgespielt, nur Punkte mit entsprechend kleiner Covarianz mit covarianz filter: https://gyazo.com/8c58c259adede0fb5b1eb977704b9655 echte daten: https://gyazo.com/60c4223b

1p

3.2 Machine Learning Approach

3.2.1 Idea and Input/Output Design

curevature to points 2-8m further down the midline using immediate environment to predict the curvature of the line segment current on

3.2.2 Modelling as Image Regressing Problem Using an CNN

mapping of input values to have a flatter distribution of values

Chapter 4

Discussion

4.1 Evaluation

4.1.1 Classical Approach

uncerternity threshhold in covariances auswirkungen analzsieren
sicherheit in der karte vs rauschen

zeitlicher verlauf nicht verfolgbar weil slam ids nicht matchen können beschrieben, weil particles getrennt kann man nicht gut mathcen, zu inperformant/wenn springen dann gar nicht

evaluation wie weit voraus notwendign sinnvoll etc 1p

Metric - Deviation From ground truth (GT)

examples for successful failed detection of the centerline 1p

Metric - Driving Test

letting driver test according to algorithm 1p

4.1.2 Machine Learning Approach

low number of training samples already great results
different parameters different learning results 2p

Metric - Driving Test

letting driver test according to algorithm 1p

4.2 Comparison of Approaches

ml more useful in first when there is no map data available, more robust for less accurate map data

less plannig ahead possible, work needed to generate map afterwards

classical resulting in complete map where planning can be done extensively, but very fragile

latenz/laufzeit betrachten als metrik

Chapter 5

Conclusion

5.1 Summary

Fasse nochmal alle Ergebnisse der Arbeit zusammen. 0.5p

5.2 Future Work

5.2.1 SLAM

Probleme des SLAMs wären wahrscheinlich noch verrauschte Position, Drifts und Doppelterkennungen. 0.5p

5.2.2 Classical Algorithm

- usage in first round - use proximity to car to add points Idee: neue Punkte nur in der Nähe des Autos zur Karte hinzufügen, - use orientation relative to car to recolor 0.5p

5.2.3 Machine Learning Algorithm

- factoring in derivation from track center - aus bild direk tkarte schätzem 0.5p

5.2.4 Other Improvements

cnn als preprocessing vor slam, bounding box von cones zu ist cone ja nein und farbe 0.5p

5.3 Outlook

where this work leads to

References

[1] Definition of curvature, 2016. URL https://tutorial.math.lamar.edu/classes/calciii/curvature.aspx. accessed on 07.01.2022.

[2] Leiv Andresen, Adrian Brandemuehl, Alex Honger, Benson Kuan, Niclas Vodisch, Hermann Blum, Victor Reijgwart, Lukas Bernreiter, Lukas Schaupp, Jen Jen Chung, and et al. Accurate mapping and planning for autonomous racing. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2020. doi: 10.1109/iros45743.2020.9341702. URL http://dx.doi.org/10.1109/IROS45743.2020.9341702.

[3] The Pyodide development team. pyodide/pyodide. August 2021. doi: 10.5281/zenodo.5156931. URL https://doi.org/10.5281/zenodo.5156931.

[4] A. Dewing. Now this is podracing-driving with neural networks. *Stanford CS231n Reports*, 2016. URL http://cs231n.stanford.edu/reports/2016/pdfs/100_Report.pdf.

[5] Maciek Dziubiński. Training a neural network for driving an autonomous rc car. *medium*, Sep 2019. URL https://medium.com/asap-report/training-a-neural-network-for-driving-an-autonomous-rc-car-3906db91f3e.

[6] Gordon R. Foxall and Brian R. Johnston. Innovation in grand prix motor racing: the evolution of technology, organization and strategy. *Technovation*, 11(7):387–402, 1991. ISSN 0166-4972. doi: https://doi.org/10.1016/0166-4972(91)90020-5. URL https://www.sciencedirect.com/science/article/pii/0166497291900205.

[7] Ignat Georgiev. Path planning and control for an autonomous race car. *Edinburgh UG4 Project Reports*, 2019. URL https://project-archive.inf.ed.ac.uk/ug4/20191552/ug4_proj.pdf.

[8] Abhinav Jangda, Bobby Powers, Emery D. Berger, and Arjun Guha. Not so fast: Analyzing the performance of WebAssembly vs. native code. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 107–120, Renton, WA, July 2019. USENIX Association. ISBN 978-1-939133-03-8. URL https://www.usenix.org/conference/atc19/presentation/jangda.

[9] Juraj Kabzan, Miguel I. Valls, Victor J. F. Reijgwart, Hubertus F. C. Hendrikx, Claas Ehmke, Manish Prajapat, Andreas Bühler, Nikhil Gosala, Mehak Gupta, Ramya Sivanesan, Ankit Dhall, Eugenio Chisari, Napat Karnchanachari, Sonja Brits, Manuel Dangel, Inkyu Sa, Renaud Dubé, Abel Gawel, Mark Pfeiffer, Alexander Liniger, John Lygeros, and Roland Siegwart. Amz driverless: The full autonomous racing system. *Journal of*

Field Robotics, 37(7):1267–1294, 2020. doi: https://doi.org/10.1002/rob.21977. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21977.

[10] Alan Lapedes and Robert Farber. How neural nets work. In D. Anderson, editor, *Neural Information Processing Systems*. American Institute of Physics, 1988. URL https://proceedings.neurips.cc/paper/1987/file/093f65e080a295f8076b1c5722a46aa2-Paper.pdf.

[11] Jerome M Lutin. Not if, but when: Autonomous driving and the future of transit. *Journal of Public Transportatio*, 21 (1), 2018. doi: 10.5038/2375-0901.21.1.10. URL https://digitalcommons.usf.edu/jpt/vol21/iss1/10.

[12] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. Fastslam: A factored solution to the simultaneous localization and mapping problem. 11 2002.

[13] Sherif Nekkah, Josua Janus, Mario Boxheimer, Lars Ohnemus, Stefan Hirsch, Benjamin Schmidt, Yuchen Liu, David Borbély, Florian Keck, Katharina Bachmann, and Lukasz Bleszynski. The autonomous racing software stack of the kit19d. *CoRR*, abs/2010.02828, 2020. URL https://arxiv.org/abs/2010.02828.

[14] Ashutosh Singandhupe and Hung Manh La. A review of slam techniques and security in autonomous driving. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 602–607, 2019. doi: 10.1109/IRC.2019.00122.

[15] Stanford Artificial Intelligence Laboratory et al. Robotic operating system, 2018. URL https://www.ros.org.

[16] Grady Williams, Paul Drews, Brian Goldfain, James M. Rehg, and Evangelos A. Theodorou. Aggressive driving with model predictive path integral control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440, 2016. doi: 10.1109/ICRA.2016.7487277.

[17] Roman Yurchak. Performance of python code, 2021. URL https://github.com/pyodide/pyodide/issues/1120.

[18] Marcel Zeilinger, Raphael Hauk, Markus Bader, and Alexander Hofmann. Design of an autonomous race car for the formula student driverless (fsd). In *Proceedings of the OAGM and ARW Joint Workshop*, Vienna, 05 2017.

Appendix A

Abbreviations

GT	ground truth	15
EKF	extended Kalman filter	3
MLP	multi layer perceptron	7
ML	machine leaning	3
CNN	convolutional neural network	3
SLAM	simultaneous localization and mapping	1
FSD	Formula Student Driverless	1
ROS	Robot Operation System	6
MPPI	model predictive path integral	3
ReLU	Rectified Linear Unit	7
NN	neural network	7

Appendix B

TrackVisualizerJS Documentation