Indy Autonomous Challenge

Karlsruhe Institute of Technology

Round One - Whitepaper

⋆

Team Contact:

Simon Schaefer
KA-RaceIng e.V.
Karlsruhe Institute of Technology

Building 70.03
Rintheimer Querallee 2
76131 Karlsruhe
Germany

E-Mail: simon.schaefer@ka-raceing.de

February 28, 2020

Contents

1 Introduction to KA-RaceIng

Founded in 2006 by students of the Karlsruhe Institute of Technology, this season KA-RaceIng is building its 27th and 28th race car to participate in the Formula Student competition. In many aspects similar to the North American Formula SAE, the goal of Formula Student is for university teams from all over the world to design and build a race car seating a single driver over the course of a year.
This year more than 70 students from different fields of study come together to form one team: KA-RaceIng. The team is structured like a company, with different subteams who are responsible for the different aspects of the endeavor, from marketing over suspension design up to the autonomous system. In those subteams, each teammember takes responsibility for a dedicated component of the respective package. The subteam leaders meet on a weekly basis to discuss and reach decisions covering topics that involve and concern the entire team and to ensure good communication.
In 2019, we at KA-RaceIng have built three cars: an internal combustion vehicle, an electric vehicle and a both electric and driverless vehicle. The electric and driverless vehicles have both scored a 2nd place overall, with 40 and 20 competitors respectively. As it is the most relevant one regarding the Indy Autonomous Challenge, the rest of this paper will focus on our driverless vehicle.
This year, we are currently building our 4th autonomous car, taking part in the Formula Student Driverless class since its foundation in 2017. Last year we have made an enormous progress, reducing our lap time for a 300m circuit including several curves from 124 to 24 seconds.[1]
To build the current driverless car, the KIT20d, the electric vehicle of the competition season 2015, the KIT15e, was used (again) as a basis and converted to have all the necessary components for a fully autonomous vehicle. More about the autonomous hardware used can be found in section 3.
In Formula Student Driverless, the car needs to excel in disciplines that require both good vehicle dynamics as well as a well-designed autonomous system. As opposed to the Indy series, the curve radii are much smaller, limiting the maximum achievable speeds.

2 Plans for the Indy Autonomous Challenge

The focus of the active team at KA-RaceIng is and will be the Formula Student competition. However, there are more than 800 alumni with a vast combined experience in building and programming race cars, and many of them are still pursuing their studies. We therefore believe that we can gather enough interest in partaking in the Indy Autonomous Challenge using current and future alumni while tapping into the knowledge and infrastructure of the active team.

[1] See youtube.com/watch?v=h22J8YzNdjo or youtube.com/watch?v=B3KfAhKuy-s for examples of our performance.

Building our current race cars is only possible with the help of over 100 sponsors, who provide things like parts, manufacturing, consulting support and financial help. We have different categories of sponsorship, with the highest one including companies such as Bosch, Daimler, Intel and Porsche.[2] We are positive that among these sponsors we could find some that support our venture of taking part in the Indy Autonomous Challenge.

3 KIT20d: Our Race Car

The vehicle platform which is used for testing our software and competing at the Formula Student competitions is a fully developed race car. Even though we drive autonomously at the competitions, the rules require that it still safely seats a single driver.
Our own lap time simulation provides us with comparisons between important sensitivities to build the fastest vehicle possible. The most important factors are tire to road coefficient of friction, weight, downforce and propulsion power. With a FEA (finite element analysis)-optimized CFRP (carbon fiber reinforced polymer) monocoque and our suspension design coming from our self-written multi-body parameter optimization, we created a lightweight and agile base that pushes the potential of the tire to its limit.
Our aerodynamic kit helps us to stay on track during high speed corners. The wings and body of the car are optimized for airflow in a curved air tunnel, all modeled with 60 million cells in our high resolution CFD simulation.
The 470V battery based on our own stack design and supervised by our Accumulator Management System (AMS) provides all the energy we need. The current, converted through an adjusted power electronic module, powers our electronically and mechanically self-developed electric motor. The latest version reaches a power to weight ratio of 7.14 kW/kg. Each of the four motors can be controlled separately which allows us the use of torque vectoring.
With a long list of sensors ranging from spring travel, hydraulic and pneumatic pressure, temperature, wheel speed, inertia, ground speed and many voltage measurements, the most import sensors are the ones guiding the car through the track. With a 40-channel mechanical LiDAR and 3 cameras, we can achieve an accurate perception range of up to 50 meters with a field of view covering 180 degrees. As a precise velocity estimation is key to a good pose estimation in SLAM, we use an optical ground speed sensor that is accurate to $<\pm$0,5 %FSO, calibrated to up to 400 km/h / 250 mph.

While the race cars for competing in the Indy Autonomous Challenge will be supplied by the organizers, having such deep knowledge in what makes a car fast will help us tremendously when designing a software that will push it to the limit.

4 Software Architecture and CI/CD

Software Architecture

For our software architecture, we set three main goals:

- **Modularity**: A simple modular structure with clean interfaces that serves as a basis for years to come, and even for projects such as this challenge.
- **Performance**: efficient and robust software for a performant vehicle. Consistent pursuit of improvement.

[2] A list of sponsors and supporting institutes can be found at ka-raceing.de/partner

- **Efficiency**: No duplicated work. Efficient, automated and independent testing. Using and extending existing resources.

In order to manage the growing complexity of the system and define separated modules we decided to use the open source framework ROS (Robot Operating System). The modular structure allows quick improvements of the overall performance by simply exchanging single blocks for enhanced versions as all interfaces are defined right in the beginning. ROS also facilitates a great amount of tools for debugging, system analysis and recording as well as replaying sets of data.

Continuous Integration / Continuous Delivery (CI/CD)

For the development of the autonomous system of the KIT20d we used a 3-stage CI/CD-pipeline as seen in figure 1. Our developers pushed the written code onto feature-branches in our git repository. On the completion of a certain feature the corresponding branch is merged into the protected master branch via a pull-request. This way we enforced the 4-eye-principle during merging. The pull-request also only gets accepted, if the master branch has already been successfully merged into the feature-branch. Thus we always have a functioning version on the master branch.

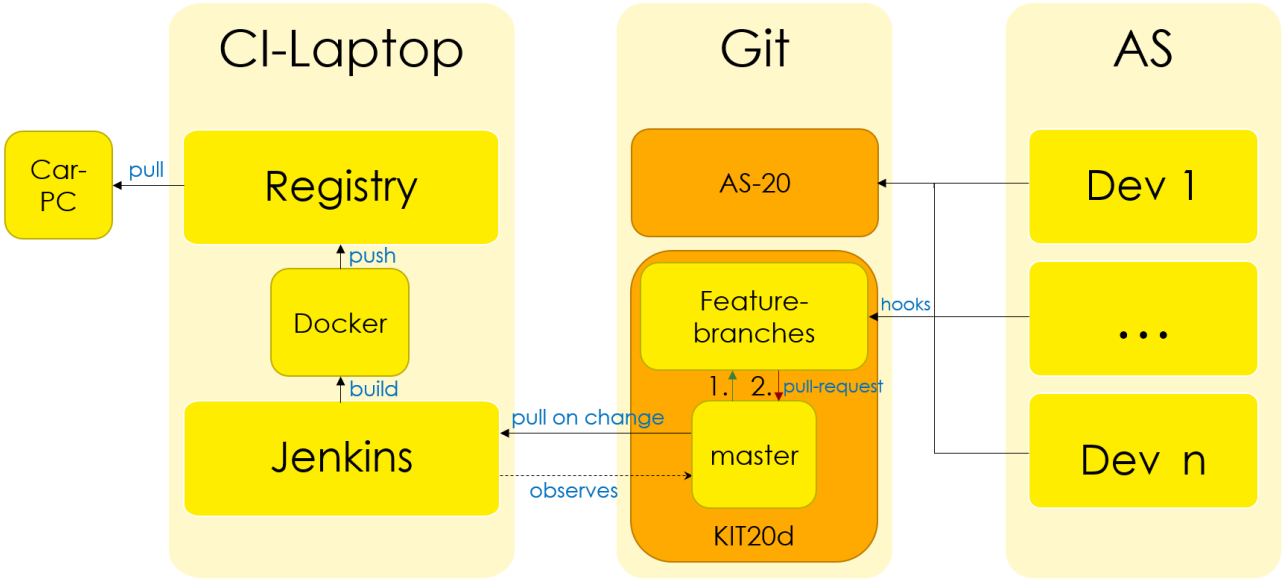

Figure 1: CI/CD pipeline

As our CI/CD tool we decided to use Jenkins, because it is developed under a MIT-license, so we are free to use it. Also it provides every necessary functionality, even the inclusion of Docker into the building process. Jenkins is configured to observe the master-branch on pull it, if it detects an update. It then uses Docker for compiling the code and stores the created docker-image in our self-hosted registry. We can then pull the image from the registry onto the Car-PC and run our software immediately without much preparation or configuration. This way the only necessary software on the Car-PC is the Docker-engine.
To ensure a high code-quality we implement test-driven development in our workflow. This ensures a high test-coverage and a very stable code after completion of a feature, as well as less time needed for debugging. We partially use pair-programming, if someone is stuck on a problem. For full-time pair-programming we don't have the resources or the time. Because our autonomous system is based on ROS' node-structure we cannot test everything with only 'normal' unit-tests. So besides the gtest framework for the C++ unit-tests we used the 'rostest'-package provided by ROS for a simple testing of the cross-node-communication.

Because of the good experiences with this approach, we are planning to continue this way for the Indy Autonomous Challenge.

5 Perception

In our KIT20d, we use a solution combining the strengths of LiDAR (precise measurements) and camera (semantic information). The LiDAR is used to detect objects by using filtering and clustering algorithms in the point cloud space. The position information of these detected objects are used to generate regions of interest (ROI) in the image space by using a bounding box. The categories and some other feature information of objects in the ROI are then identified by analyzing these cut-out sections of the camera image using an efficient convolutional neural network (the model is similar to *Faster R-CNN*[3].
In the Indy Autonomous Challenge, we would need to be able to perceive racetrack limits (fences, ground markings) in order to determine the drivable surface of the racetrack as well as rival cars on track, in order to be able to overtake them and avoid them in case any collisions occur in front us. There are other very important aspects to be taken care of such as flags (checkered, green, yellow, etc.), seeing how they dictate the car's speed and driving style.
There are several other perception approaches that could and need to be tested, such as doing semantic classification on the point cloud itself as well as having two separate pipelines (for the LiDAR and camera), which are then fused and give out the corresponding perception of the environment. Our plan is to expand on the concept we have developed for Formula Student Driverless depending on the sensors available and other frame conditions.

6 SLAM

Introduction

One of the great challenges of robotics is to know the robots environment and the robots location in it. The surroundings are needed as a basis for path planning, the location is necessary for the controller to be able to follow the path. While it is possible to just localize yourself in a given map, such a map is not always available. In that case, the robot faces the challenge of solving two opposing problems:

- Given your location, discover the surroundings and map the environment.
- Given a map of the environment, find your location in it.

This is called the problem of *Simultaneous Localization and Mapping (SLAM)*.
To add to this problem, measurement of the surroundings or the current vehicle state will, in reality, always be flawed. This has to be taken into account, when solving the SLAM problem. The general idea to solve this problem is to combine as much information as possible to the most likely estimation of the vehicles state and the map of the surroundings. Given a time delta, a velocity estimation can be utilized to get a first estimate of the vehicle's location. Such an estimated velocity is usually computed by combining odometry data from various sensors, e.g. wheels speeds, inertia measurements, GPS data or ground speed sensors. Localization based on odometry will always be affected by drift and will not give any information about the robots surroundings, which is necessary for path planning and collision avoidance.
That's where sensors like radar or LiDAR come into play. With these sensors it is possible to map undiscovered parts of the surroundings and use re-observed landmarks to correct the drift in the vehicles position. It is important to consider that these measurements, as well as the velocity estimation, are not without error.

[3] Shaoqing Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2015. arXiv: 1506.01497 [cs.CV].

Approaches

There are multiple ways to handle this problem. They can roughly be divided into three main approaches:

- Kalman filter
- Particle filter
- Graph-based approaches

All of these approaches have their respective pros and cons, so it is impossible to tell which approach is the best without looking at the specific use case.
In our case, the competition is based around tracks marked with cones. That is why our maps are landmark-based as opposed to dense maps that make more sense in a less defined environment. Also, the number of landmarks is very limited, which is the reason we are currently using an EKF (Extended Kalman Filter) approach. The computational complexity increases quadratically in the number of mapped landmarks[4], so this may not be the best solution for large scale SLAM.
We are also working on a graph-based method. This promises highly accurate maps and scalable complexity by using a sliding-window approach. We will compare these methods in performance and choose the one more capable of the specific challenge we're facing.
For the Indy Autonomous Challenge it is sensible to either use a dense map or to describe landmarks like the outer wall of the track as line-features. Also, Indianapolis Motor Speedway is around the tenfold in length when compared to a typical Formula Student track, on which we are competing. This points towards a graph-based approach to the SLAM problem, because the quadratic complexity of the EKF is very likely going to be too high to be real-time capable.

7 Trajectory Planning

In path planning, the landmarks information or/and the map from SLAM will be used in order to provide the controller with the orientation and velocity of the vehicle in the next step.
The basic steps of path planning are: firstly abstracting the configuration space or describing the collision free space; secondly finding way points in the defined space; thirdly generating a path based on the way points; and then optimizing the path and calculation the ideal velocity with a GGS-diagram. According to the optimization method, an iteration of the process could be executed. In this section, the emphasis will be laid on the GGS-diagram.

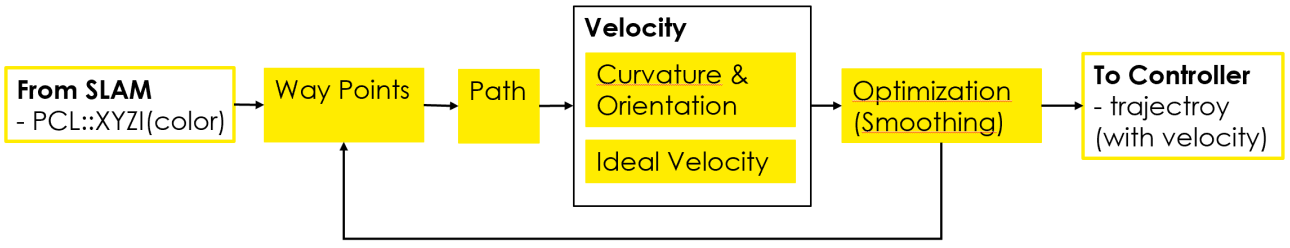

Figure 2: Subtasks in path planning with an iteration

A **GGS-diagram** shows the relation between the maximum lateral and maximum longitudinal ac-/deceleration at different velocities, which the vehicle can achieve at the same time. Thus, with a GGS-diagram, the full

[4] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. Cambridge, Mass.: MIT Press, 2005.

potential of the vehicle and its tires can be reached. For example, with a GGS-diagram, the vehicle can pass through the corner at an allowable maximum velocity, because the down-force from aerodynamic kit is already considered in this diagram.
Fig. 7 shows the process of how to use a GGS-diagram to obtain the velocity. Firstly, the points with local maximum curvature at the path will be found and labeled as apexes. Then, with an iterative process, the velocity of this apexes will be fixed. In the process it is assumed that the lateral ac-/deceleration of each apex achieves the maximum at its velocity. At the end, the velocity of each point between two apexes will be calculated. The first step is figuring out the velocity which increases from one apex to the next apex, and then working out the velocity which increases from the next apex to current apex, and the final velocity of each point is the smaller one in the two steps.

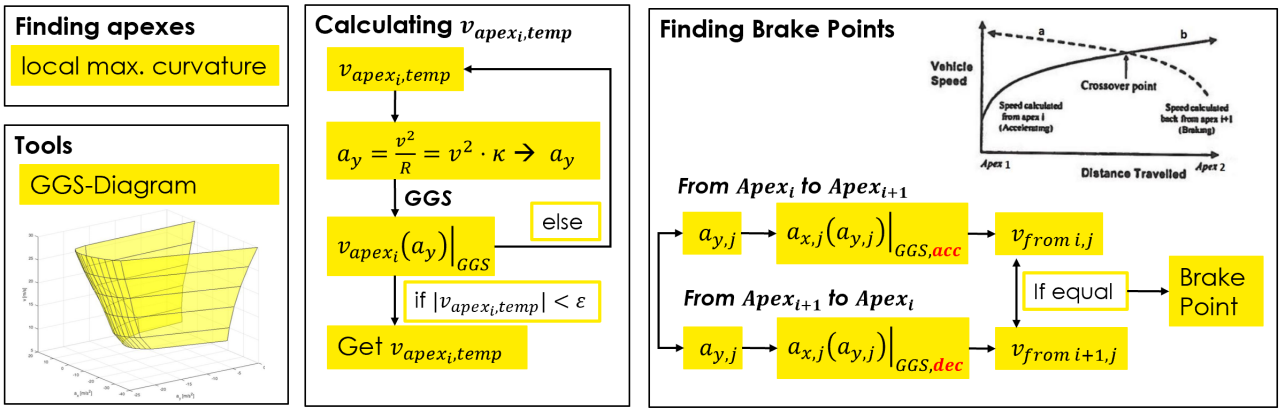

Figure 3: Using a GGS-diagram to calculate the velocity of each point at the path

8 Control

As shown in figure 4, our current setup is based on using a Proportional Integral (PI) controller for longitudinal control and an Model Predictive Control (MPC) for lateral control. The model used in the MPC controller is based on the model used for vehicle dynamics simulations.
An alternative concept would see the two controllers combined to a single, non-linear MPC. While having the potential to be better in teams of accuracy of control, the risk lies in high computational load generated by such a non-linear problem. It stands to be determined if a sufficiently high frequency can still be achieved.
To maximize the possible speed while retaining control, we use traction control (implemented as a PID controller) and are currently working on a friction coefficient estimation. Four separately powered wheels make it possible for us to use torque vectoring for optimally balancing forces on the wheels.

9 Simulation

For building and programming an autonomous vehicle it is essential to be able to test and debug your code without having the car to actually drive or even exist. In the first place this saves time and money since testing time is usually very expensive. Furthermore the risk of unexpected behavior which could even result in costly crashes can be reduced. Therefore having a well written simulation is key.
In our case the we are simulating the whole autonomous pipeline starting at the perception, over SLAM and planning up to the controller. The simulation is written in C++ based on the underlying ROS system and

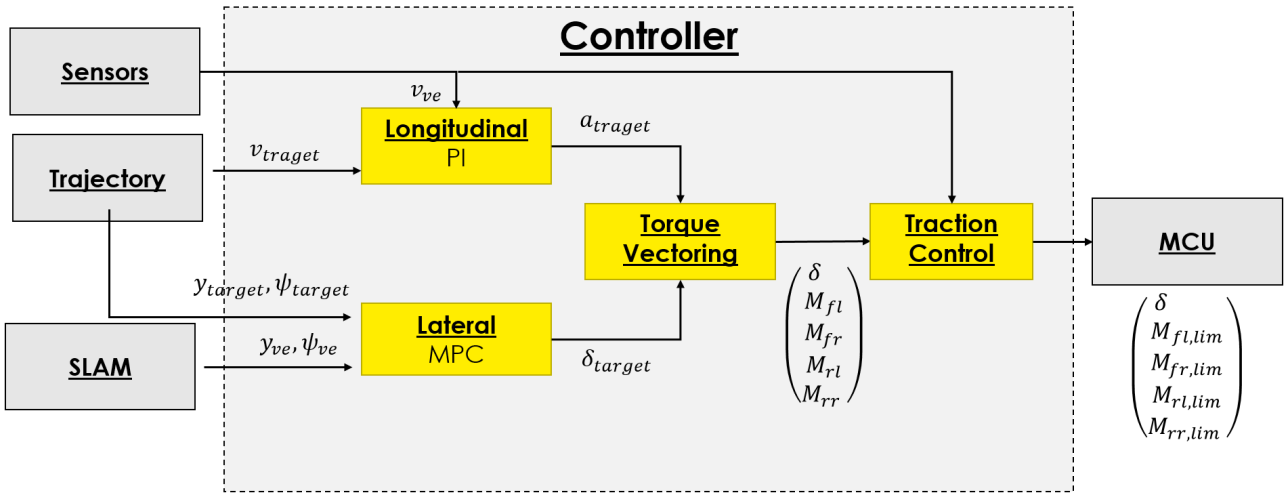

Figure 4: Concept for controlling the car

lets all nodes communicate in the same way as they do on the running car.
For perception we are currently developing a sensor simulation in Unreal engine. That way we can create picture and LiDAR data, which is then processed by the perception algorithms. To make this work, we are training the neural net with real pictures as well as the simulated data.
In addition simulating sensor data gives us the big advantage of having a ground truth, which is very important for validating the quality of the SLAM algorithms. In the simulation we know exactly where every landmark is located, therefore all deviations in the output of the SLAM are caused by the uncertainties of the perception and SLAM algorithms.
To make the simulation even more realistic, we can fit all sensor outputs with widely adjustable distortions or even completely block them out to simulate total failure of a sensor. That way the reaction of the autonomous system to critical situation can be tested beforehand and the safety of the vehicle is enhanced.
For having a response of the vehicle to the outputs of the controller, we are using a vehicle model with eight degrees of freedom. Since this model is more precise than the one used by the controller, the control algorithms can be tested fairly well.
In the future an even better vehicle model should make it possible to even simulate IMU data so that the velocity estimation can be tested as well. Overall the simulation is a very useful feature without which we would not be as successful in the Formula Student. It is a great basis for round 3 of the Indy Autonomous Challenge.

10 Summary

Having built a race car from scratch, constructed a sophisticated autonomous pipeline and having good software development practices in place, we think we are well prepared for taking part in this challenge.
We look very much forward to taking our maximum speed to a completely new level.