

Design of an Autonomous Racecar: Perception, State Estimation and System Integration

Miguel I. Valls*, Hubertus F.C. Hendrikx*, Victor J.F. Reijgwart*, Fabio V. Meier*
Inkyu Sa, Renaud Dubé, Abel Gawel, Mathias Bürki, and Roland Siegwart

Abstract—This paper introduces *flüela driverless*: the first autonomous racecar to win a Formula Student Driverless competition. In this competition, among other challenges, an autonomous racecar is tasked to complete 10 laps of a previously unknown racetrack as fast as possible and using only onboard sensing and computing. The key components of *flüela*'s design are its modular redundant sub-systems that allow robust performance despite challenging perceptual conditions or partial system failures. The paper presents the integration of key components of our autonomous racecar, i.e., system design, EKF-based state estimation, LiDAR-based perception, and particle filter-based SLAM. We perform an extensive experimental evaluation on real-world data, demonstrating the system's effectiveness by outperforming the next-best ranking team by almost half the time required to finish a lap. The autonomous racecar reaches lateral and longitudinal accelerations comparable to those achieved by experienced human drivers.

I. INTRODUCTION

On August 13th 2017, *flüela driverless* became the first car to ever win the Formula Student Driverless (FSD) competition. The competition requires the car to race fully autonomously and consists of 4 dynamic and 4 static disciplines [1]. The dynamic disciplines test the system's reliability under general race conditions and at high lateral and longitudinal speeds. The static disciplines evaluate the system's design under aspects of software, hardware, costs, and business. While *flüela driverless* performed well in all categories, we this paper focuses on software and hardware designs.

The hardware platform for the project is *flüela*, an electric 4WD car with a full aerodynamic package, high wheel torque, and a lightweight design developed by AMZ¹ for Formula Student Electric 2015. The sensor outfit for autonomous operation and the software system are developed from scratch.

In our autonomous design, system reliability under high performance operation is chosen as the main design goals, since the FSD regulations allow no human intervention.

This paper presents the state estimation, LiDAR SLAM, and localization systems that were integrated in *flüela*. The autonomous system perceives its surroundings using a 3D LiDAR and a self-developed visual-inertial stereo camera system. Furthermore, a velocity sensor and an Inertial Navigation System (INS) combining an IMU and a GPS were added for state estimation. All the information is processed



Fig. 1. *flüela driverless*, the winning electric, autonomous 4WD racecar at the Formula Student Germany 2017. The LiDAR, the GPS and the visual-inertial system are respectively marked by tags 1 to 3.

online by two computing units running Robot Operating System (ROS) and a real-time capable Electronic Control Unit (ECU). Figure 2 shows an overview of the hardware-software setup.

In order to reach *flüela*'s full potential when racing autonomously, the track must be known at least 2s ahead. At high speeds, this requires a perception horizon that exceeds the sensors' range. The car must thus drive carefully to discover and map the track. This mode will later be referred to as *SLAM Mode*. Once the map is known, the car can drive in *Localization Mode* which can exploit the advantage of planning on the previously mapped race-track.

The contributions of this paper are:

- A complete pipeline from perception to state estimation with on-board sensors and computation only, capable of driving an autonomous racecar close to a human driver's performance.
- Extensive experimental evaluation and demonstration in real-world racing scenarios.

The remainder of this paper is structured as follows. Section II introduces state-of-the-art work on autonomous racing, Section III describes the theoretical development for this project and Section IV the implementation details. We present our experimental results in Section V, and conclude in Section VI.

II. RELATED WORK/BACKGROUND

Autonomous racing is an emerging field within autonomous driving. In the last years, a few self-racing vehicles have been developed, both in academic and in the industrial research. The first known autonomous vehicle competition

* The authors contributed equally to this work.

Authors are with the Autonomous Systems Lab, ETH Zurich, Zurich, Switzerland. dmiguel@ethz.ch

¹<http://www.amzracing.ch/>

was the DARPA Grand Challenge, [2] which motivated the development of several autonomous cars in a two year period. These cars had to compete in a desert environment and drive through a way-point corridor given shortly before the race. In this sense, it is similar to FSD since a short period for mapping is allowed just before the race. They however differ in that the FSD track is asphalt, the vehicles are designed for racing and reached over 90km/h and 10m/s^2 accelerations.

Other autonomous racecars were developed afterwards [3], but their main goal was vehicle dynamic control and not SLAM and state estimation. In addition, several scaled racecars were developed [4] but they focus on control and have an external localization system. Others were developed with on-board sensors only [5] but the main focus also lied on control.

Finally, a look to the industry should not be forgotten, where an Audi RS7 and a Nio EP9 broke the speed record for autonomous cars in 2014 and 2017 respectively. Devbot from roborace also featured the first wheel to wheel autonomous race (2017).

III. FLÜELA DRIVERLESS

In this section an insight is given in the full setup of the state estimation system, including the LiDAR/camera based mapping & localization system. First a high level system overview is presented.

A. System architecture

The system architecture has been designed for reliability and performance. Reliability was given largest priority as the competition only grants one run, regardless of adverse weather conditions or software glitches.

The car is fitted with an Inertial Navigation System, an optical Ground Speed Sensor (GSS), a LiDAR and a self-developed visual-inertial stereo camera system. Furthermore, individual wheel speeds are determined by reading out each wheel encoder. Consumer-grade GPS is used (no differential GPS or RTK) as an absolute position sensor. Cones that mark the race-track are detected by both LiDAR and camera to create redundancy in the perception pipelines.

The chosen computing system consists of a high performance slave and an industrial master computer. The slave computer is dedicated to vision-based perception and the master computer runs all other software packages. Since vision-based perception is redundant with LiDAR, this solution ensures high reliability without limiting performance. The last important factor for reliable operation is the self-developed computing housing, presented in Sec. IV-B.

The designed software system runs on Ubuntu 14 LTS within the ROS Indigo framework. The distributed nature of ROS simplifies the integration of the slave computer. Chrony is used to synchronize the clocks of both computers over Ethernet.

Finally, a real-time capable ECU runs the low level controllers and low level state machine of the car. The torque vectoring and traction controllers developed for the original car are used to distribute individual torques to all 4 wheels

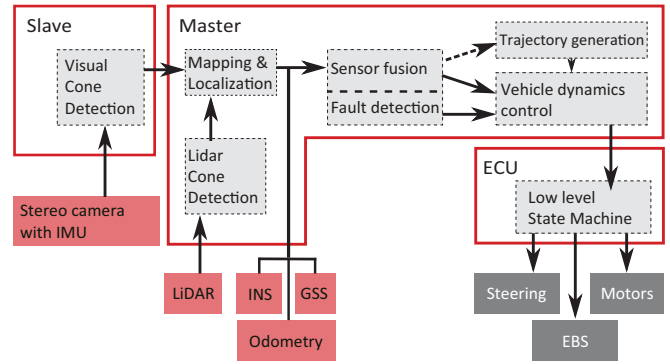


Fig. 2. Overview of the autonomous system's architecture.

at 200Hz. Their target is the desired throttle calculated on the master computer. The car relies on regenerative braking encoded as a negative throttle input during normal operation and the mechanical brakes are only used for emergency stops. Lastly, the ECU forwards the desired steering angle from the master computer to the internal controller of the steering actuator after a simple integrity check.

B. Pose and Velocity Estimation

State estimation is an essential part of any mobile robotic application as it enables the robust operation of other system components. Several sensors are fused to estimate the pose and velocity of the ground vehicle. To take advantage of redundancy in state estimation, the contribution of each sensor input to the overall estimated state has to be quantified in function of the sensor's accuracy and previous state knowledge. The Extended Kalman Filter (EKF) is the state-of-the-art estimator for fast, mildly non-linear systems. For systems with white zero-mean additive gaussian noise corrupting the sensors and the process model, it is a good approximation of the optimal solution.

1) *Process model*: The process model used is driven by the accelerometer as proposed in [6]. The vehicle body frame is chosen to coincide with the IMU frame. A constant velocity model is used with the accelerometer as a pseudo input to the system. Due to the application constraints, it is known that the vehicle will remain on the ground and not be substantially tilted. This assumption simplifies the state to a 2D state with only 6 elements. The state vector is defined as:

$$\begin{aligned} \mathbf{x} &= [\mathbf{p}^T, \theta, \mathbf{v}^T, r]^T \\ \mathbf{p} &= [x, y]^T \\ \mathbf{v} &= [v_x, v_y]^T \end{aligned} \quad (1)$$

where \mathbf{p} and θ are respectively the position and heading of the car (IMU) expressed in world reference frame. \mathbf{v} and r are respectively the linear and angular velocities of the car expressed in body reference frame. The process model is defined as:

$$\begin{aligned} \dot{\mathbf{p}} &= \mathbf{R}(\theta)^T \mathbf{v} \\ \dot{\theta} &= r \\ \dot{\mathbf{v}} &= \mathbf{a} + [v_y r, -v_x r]^T + \mathbf{n}_v \\ \dot{r} &= n_r \end{aligned} \quad (2)$$

where \mathbf{a} is the linear acceleration measured by the IMU, $\mathbf{R}(\theta)$ is the 2D rotation matrix between the vehicle body frame and the world reference frame, \mathbf{n}_v and n_r are i.i.d white noise distributed as $\mathbf{n}_v \sim \mathcal{N}(0, \Sigma_v)$, $n_r \sim \mathcal{N}(0, \Sigma_r)$.

2) *Sensor model*: The vehicle is equipped with multiple sensors (see Sec. I) which can be decomposed on the quantities being measured: position (\mathbf{z}_p), heading (z_θ), velocity (\mathbf{z}_v), and yaw rate (z_r). For instance, the GPS can be seen as a position sensor and the localization described in (Sec. III-D) as a position and a heading sensor. For this to hold, it is assumed that the noises of the decomposed measurements are uncorrelated. The measurements are introduced as

$$\begin{aligned} \mathbf{z}_p &= h_p(\mathbf{x}) = \mathbf{p} + \mathbf{R}(\theta)^T \mathbf{p}_s + \mathbf{n}_{z_p} \\ z_\theta &= h_\theta(\mathbf{x}) = \theta + \theta_s + n_{z_\theta} \\ \mathbf{z}_v &= h_v(\mathbf{x}) = \mathbf{R}(\theta_s)(\mathbf{v} + [-r \mathbf{p}_{s,y}, r \mathbf{p}_{s,x}]^T) + \mathbf{n}_{z_v} \\ z_r &= h_r(\mathbf{x}) = r + n_{z_r} \end{aligned} \quad (3)$$

where $h_{\{\cdot\}}(\mathbf{x})$ are the different measurement models, \mathbf{p}_s is the position of the sensor in body frame and θ_s is the sensor heading in body frame. $n_{\{\cdot\}}$ are gaussian i.i.d. noises that corrupt the sensor measurements.

3) *Observability analysis*: In order to determine for which states the system is observable, the observability matrix of the non-linear system must be analyzed. It can be constructed using the Lie derivatives of the sensor model presented in III-B.2. They are defined recursively as

$$L_f^{l+1} h(\mathbf{x}) = \frac{\partial L_f^l h}{\partial \mathbf{x}} f(\mathbf{x}, \mathbf{u}) \quad (4)$$

with $L_f^0 h(\mathbf{x}) = h(\mathbf{x})$.

The Observability matrix is defined as

$$\mathcal{O}(\mathbf{x}, \mathbf{u}) = \left[\frac{\partial L_f^0 h(\mathbf{x})}{\partial \mathbf{x}}, \frac{\partial L_f^1 h(\mathbf{x})}{\partial \mathbf{x}}, \dots \right]^T \quad (5)$$

By performing a rank-test on \mathcal{O} , it can be determined whether the system is weakly locally observable (in case of full column rank, [7]) or not observable. This analysis yields three scenarios:

- 1) The state is observable if there is at least one position and one heading sensor.
- 2) The state is not observable if there is no position sensor.
- 3) The state is observable except at stand-still if there is a position sensor but no heading sensor.

In the current setup, there always is a position sensor (GPS) but no heading sensor until the map is known and localization output is fed to state estimation, which means scenario 3) in *SLAM Mode* and 1) *Localization Mode*. To overcome the fact that the heading cannot be estimated at stand-still if the map is not known, a Frozen Pose Update (FPU) is implemented. It differs from the Zero-velocity update (ZUPT) since it assumes a constant pose instead of zero velocities. As long as zero-motion is detected, a virtual measurement is added that simulates a position and heading sensor with the value of the frozen pose. This prevents the system from drifting

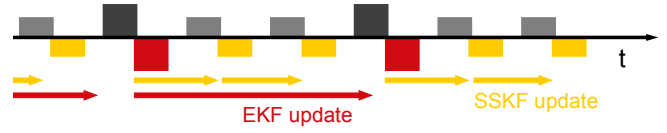


Fig. 3. Approximate delay compensation. The EKF accurately estimates the car state (red) up to the most recent low frequency measurement (black). Fast, non-delayed measurements (grey) are incorporated into a high frequency, temporary estimate (yellow) using the SSKF. This method locally approximates the model as an LTI system with stationary noise distributions.

even if the process model is biased or there is noise in the velocity sensors. In this application, the whole pose is frozen instead of only the heading as it is more important for the pose estimate to not drift at stand still than to drift towards the actual position.

4) *Delay compensation*: The EKF approach can only take into account measurements from the current state which is a problem with delayed measurements. A trivial solution is to keep a buffer of previous state distributions and measurements, and at every iteration the state is propagated forward and corrected with all the newer measurements up to the current time.

Although this approach is simple, consistently delayed measurements considerably increase the computational time of the filter. For other methods addressing discrete Kalman filters with delays, see [8]. We propose an approximate approach (see Fig. 3) where the EKF is calculated up to the most delayed measurement. A steady-state approximation of the EKF (SSKF) is then executed, taking into account all measurements newer than the most delayed one to keep a high-rate updated estimate for the control system. The SSKF is a simplified version of the EKF, where the covariance is assumed to be constant (or slowly varying) for the interval from the most delayed measurement to the current time. The measurement model is assumed to be close to linear and the measurement noise and process noise are assumed to be stationary for this interval. This leads to a constant Kalman gain, avoiding the matrix inversion step. There is also no need to calculate the covariance in this interval. This approach provides a trade-off for systems with delayed measurements that balances the accuracy of the EKF and runtime of SSKF.

5) *Outlier rejection and self-diagnosis*: Sensor faults are a major factor undermining the robustness of state estimation systems. We therefore use a probabilistic outlier detection method that works with any sensor. The idea was first presented by Brumback and Srinath [9] and later used by Hausman et al. [10]. This approach makes use of the innovation covariance calculated in the EKF. This allows one to assess the likelihood of a measurement belonging to innovation distribution. This approach intrinsically accounts for the uncertainty of the state and the sensor noise model:

$$\mathbf{r} = \mathbf{z} - h(\hat{\mathbf{x}}) \quad (6)$$

$$\mathbf{S} = \mathbf{H} \mathbf{P} \mathbf{H}^T + \mathbf{R} \quad (7)$$

where \mathbf{r} and \mathbf{S} are the residual (or innovation) and its covariance. \mathbf{z} and \mathbf{R} are the measurement and its covariance. $\hat{\mathbf{x}}$ and \mathbf{P} are the estimated state and its covariance. $h(\cdot)$ is

the measurement model and \mathbf{H} is its linearization around $\hat{\mathbf{x}}$.

If the Chi-Squared (χ^2) test fails, the measurement is considered an outlier. It fails when Eq. 8 does not hold:

$$\mathbf{r}^T \mathbf{S}^{-1} \mathbf{r} < \chi^2(\chi_i) \quad (8)$$

where $\chi^2(\cdot)$ is the Chi-Squared distribution of as many degrees of freedom as the size of \mathbf{r} and $\chi_i \in (0, 1)$ is the threshold to reject an outlier of the i^{th} sensor in the χ^2 test.

In this paper, the outlier detection is extended to *health* estimation and diagnosis based on the same idea as the outlier detector (the normalized sum of squares of the residuals). This normalized sum is scaled to reach 1 when it is considered an outlier and saturated to 1.

$$D_i = 1 - \max \left\{ \frac{r_i^T S_i^{-1} r_i}{\chi^2(\chi_i)}, 1 \right\} \quad (9)$$

$$D_T = \frac{\sum_{i=1}^p w_i D_i}{\sum_{i=1}^p w_i} \quad (10)$$

where $D_i \in [0, 1]$, r_i and S_i are the last diagnosis, last residual and last innovation matrix of the i^{th} sensor respectively. w_i is the weight of i^{th} sensor in the weighted sum that determines the overall diagnosis of the system (D_T). p is the number of sensors. $D_T \in [0, 1]$ where 0 means that every sensor is an outlier and 1 means that every sensor is perfectly predicted. The weights w_i are introduced to represent the impact of every sensor on the overall health diagnosis of the system.

C. LiDAR Cone Detection

3D LiDARs are used for detecting cones that mark the race track because of their robustness against variations in illumination. The model used is a Velodyne VLP 16 Puck. Left and right cones are colored blue and yellow respectively. Colors cannot be distinguished from the LiDAR point cloud at an acceptable range, therefore no color information is used. The cone locations are extracted using the pipeline depicted in Figure 4. The motion distortion is removed from the point cloud by using a velocity estimate provided by the state estimation module. The ground is then removed based on a local flatness assumption. Removal is performed by dividing the scan in segments, as seen in Fig. 5, [11]. Every point that is lower than the lowest point in its segment plus a threshold is removed. Two of these revolutions are accumulated and passed on to the cone detector.

The first step in detecting cones in the ground-free point clouds is Euclidean clustering. The clusters are then classified as cones depending on their size. In an additional filtering step, clusters are rejected using their distance to the LiDAR and the contained points within the cluster. Cones may not always appear in every scan because of pitching motions and distant cones can fall in between two Velodyne rays. Since multiple LiDAR scans are not fused, this is solved with a second clustering stage. The centroids of the clusters are exported to a new point cloud and the last 10 of these point clouds are stored in a rolling buffer. The combined content of this buffer is processed again with Euclidean clustering. The number of points in the resulting second stage

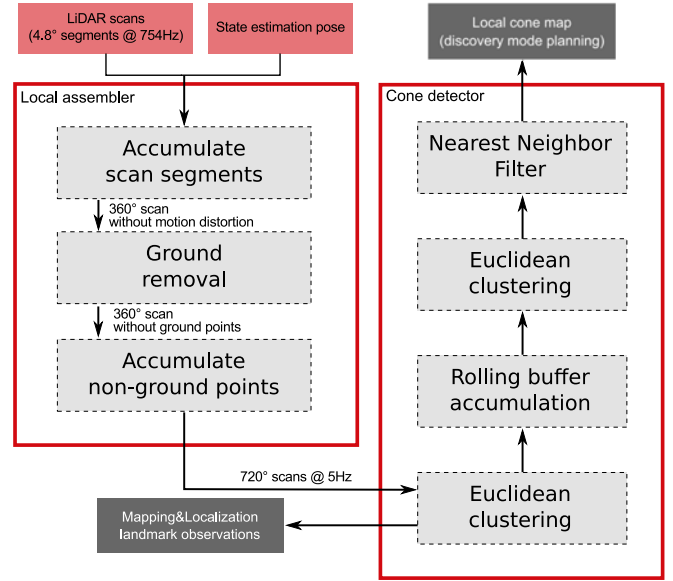


Fig. 4. Overview of the LiDAR processing pipeline.

clusters show how often the cone was observed in the last 10 scans. Cones observed twice or more are passed on to the last step.

False positive cones may be detected in areas with uneven terrain or tracks surrounded with high grass. These do not affect the path planning module as they mostly appear outside the track boundaries. However, the computation time of subsequent modules of our system scales with the number of detected cones. A Nearest Neighbour filter is applied to the observed cones to filter out areas with concentration of clusters that are higher than the expected concentration of cones.

D. Mapping & Localization

The maximum range of the perception sensors limits the length of the vehicles path planning horizon. This problem can be overcome by mapping the track and localizing the vehicle within it. As previously mentioned, the track is only marked with cones. The Simultaneous Localization and Mapping (SLAM) module is designed to accept input from either the LiDAR or camera processing pipeline which ensures safe operation in case of single sensor failure. The track is again assumed to be flat. Only cones are considered as landmarks and other potential features are rejected. There are two distinct phases, corresponding to the previously introduced in Sec. I, *SLAM* and *Localization Mode*. First, the SLAM phase in which the module builds a 2D landmark map of the race track and second, the localization phase where the map is fixed and used to estimate the vehicle pose. The switch from SLAM to localization is performed after a loop closure of the mapped race track is detected. In the following sections, a detailed description of both phases is given.

1) *SLAM Phase*: The cone observations provided by one of the perception pipelines (camera or LiDAR) are used as landmark inputs. Descriptors cannot be used to aid in data association since the cones are only distinguishable

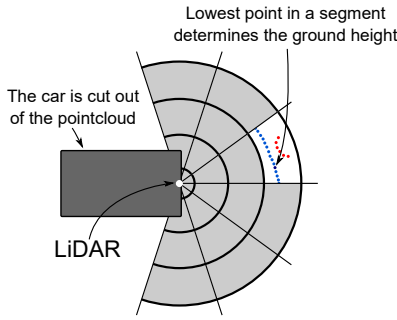


Fig. 5. Angular and radial segmentation of the LiDAR scan for ground removal. The number of segments is reduced for illustrational purposes. The experimental setup is run with 12 radial and 7 axial segments.

by color (the LiDAR cannot detect the color reliably), geometrically identical and all placed on similar looking asphalt. For this reason, we choose to use FastSLAM [12], a Rao-Blackwellized particle filter based SLAM method. Its ability to handle data association on a per particle basis makes it more robust than a SLAM approach that only considers one association hypothesis per time step (e.g., EKF-SLAM). The method also requires a odometry input. The full state estimation pose estimate (see Sec. III-B) is used while mapping which includes normal GPS. Note that the SLAM pose is not an input to the state estimation, so no information loops are induced.

The map \mathbf{m} is parameterized as a collection of N landmarks. The location of each landmark is estimated using a 2 dimensional EKF (μ , Σ). Additionally, we record each landmark's observation count n_o and missed observation count n_m within perception range.

$$\mathbf{m} = [[\mu_1, \Sigma_1, n_{o,1}, n_{m,1}], \dots, [\mu_N, \Sigma_N, n_{o,N}, n_{m,N}]] \quad (11)$$

In the particle filter, every particle $\mathbf{Y}^{[i,k]}$ with index i is a combined sample of the vehicle pose $\zeta^{[i,k]}$ and the map $\mathbf{m}^{[i,k]}$ at time k .

$$\mathbf{Y}^{[i,k]} = [\zeta^{[i,k]}, \mathbf{m}^{[i,k]}] \quad i = 1 \dots P, k \in \mathbb{N} \quad (12)$$

The particle filter is updated every time a new set of landmark observations \mathbf{z} becomes available. First, the particles poses are propagated using an odometry motion model [13, pp. 132-139], assuming zero mean uncorrelated Gaussian noise on translation and rotation respectively.

Then, observations are associated to existing landmarks in the map. This is done separately for each particle with the maximum likelihood principle. We define a likelihood function L that expresses the likelihood of an observation \mathbf{z}_i coming from a landmark \mathbf{m}_j .

$$L(\mathbf{z}_i, \mathbf{m}_j) = \frac{\exp(-\frac{1}{2}(\mathbf{z}_i - \mu_j)^T \Sigma_j^{-1}(\mathbf{z}_i - \mu_j))}{\sqrt{(2\pi)^2 |\Sigma_j|}} \quad (13)$$

Observations are assigned to known landmarks in an iterative manner. Mutual exclusion is enforced by using a queue mechanism. If a more likely observation-to-landmark association is found, the previous associated observation is

put back into the queue for reconsideration. If an observation cannot be associated with a likelihood of more than the threshold c , a new landmark will be initialized for that observation.

With the now known data association $\mathbf{a}^{[i,k]}$ for every particle, the EKF for each landmark is updated. Lastly, the weight w of each particle is calculated. The observation likelihoods are incorporated in this weight and the number of new landmarks $l^{[i,k]}$. A penalty $\beta^{[i,k]}$ is added for landmarks that were not observed, but are in the sensor's field of view.

$$w^{[i,k]} = c^{l^{[i,k]}} \cdot \beta^{[i,k]} \cdot \prod_{a_j \in \mathbf{a}^{[i,k]}} L(\mathbf{z}_{a_j}^{[k]}, \mathbf{m}_j^{[i,k]}) \quad (14)$$

The weights are then normalized.

Resampling of the particle filter is not done at every time step. To determine if resampling is necessary the effective sample size $N_{eff}^{[k]}$ is calculated. Only if this drops below $\frac{3}{4}P$ the particles are resampled using the systematic resampling method [14].

$$N_{eff}^{[k]} = \frac{1}{\sum_{i=1}^P (w^{[i,k]})^2} \quad (15)$$

2) *Loop Closure Detection*: The particle filter based SLAM method has no explicit loop closure detection. To detect the closing of the race track a simple finite state machine method is used. All particles include a loop closure state, which can take three states *Initialized*, *TravelledAway* and *ReturnedHome*. Particles start in the *Initialized* state and, when they move outside a 10m radius from their starting position, are transitioned to the *TravelledAway* state. The *ReturnedHome* state is triggered by coming back within a 5m radius of the starting position, with a heading not deviating more than an angle γ from the starting heading. When all particles reached the *ReturnedHome* state and the standard deviation of the pose estimated by all particles drops below 0.1m a closure is assumed. The system then switches to the localization phase.

3) *Localization Phase*: When the switch is made from mapping to localization the map of the highest weight particle is selected. First the landmarks in this map are pruned. This is done by removing the ones that have an observation ratio $n_o/(n_o + n_m)$ lower than 30%. Then the track boundaries are determined by linking the landmarks together. The track middle line is isolated from a Voronoi diagram constructed with the track boundaries.

The highest weight particle is now copied to all other particles. The landmark EKF update is disabled, this fixes the map, and the odometry input is switched from full state estimation to an integration of the velocity/wheel sensor and the gyroscope (without GPS). The estimate for the position and heading are extracted from the particle filter by taking the weighted average of all particles. This is fed to the state estimation module for further processing.

E. Safety

Without a driver, fluella can accelerate from 0 – 100km/h in under 2s and corner with up to 1.7g. This power unfortu-

nately also translates into potential danger. A safety system has thus been devised that guarantees robustness in case of a single mode failure and remains fail-safe in case of combined failures. The system combines safety mechanisms on all levels, from hardware up to the individual software modules.

Starting with hardware, the car has been extended with an Emergency Brake System (EBS) which brakes by default. It can only be released when the HV safety circuit is closed and additionally requires a continuous OK signal from the ECU. One level higher, the real-time ECU listens for errors on the car's CAN network, monitors the heartbeat of the autonomous master computer and the state of the Remote Emergency System (RES). If the RES is pressed, the car fully engages the brakes within 0.2s, resulting in a deceleration of at least 8m/s^2 until standstill. With these specifications, a system malfunction at 60km/h in a corner would result in the car travelling up to 14m out of track. This is assuming the safety operator pressed the RES within 0.5s. Eliminating the human reaction time would bring this distance down to 3m.

The autonomous master computer therefore runs a High Level Safety System (HLS), which monitors the heartbeats of each autonomous software module package. The heartbeats carry sequence IDs to detect package loss, time stamps for latency estimation, module load information and a health indicator. The HLS additionally tracks the system resource usage of each module.

On each iteration of the HLS, an anomaly detection algorithm classifies each subsystem as dead or alive. A decision tree then checks if every autonomous function is still covered by at least one package. The car would for example only keep driving in case of a LiDAR pipeline failure if the vision pipeline is still running. The second step is a calculation of the overall system health based on the individual package healths and system resource usage. When driving in *Localization mode*, the top speed is scaled according to the system health. If it falls below a threshold, the car is judged unstable and stopped. If both the decision tree and health threshold deem the car safe to drive, the HLS sends a heartbeat to the ECU and the process repeats.

At the highest level, certain software packages are also allowed to directly trigger the EBS. This would for example happen if the LiDAR detected a large obstacle directly in front of the car whilst racing.

IV. IMPLEMENTATION

A. Software setup

To ensure code quality while keeping the validation process efficient, special attention was paid to the testing methodology and tools.

1) *Version Control*: Git was used to efficiently code as a team and keep an integral record of the entire project history. A three-stage branching model was used to develop and validate functional components. For each new major feature, a Git branch was forked from master into the *Development* stage. Once the new code was written and ready, it was

scheduled for testing by moving it to a branch of the *Validation* stage corresponding to the test type and date. After the code had been tested and proven to be stable, it was merged back into the *Stable* stage's only branch: master.

2) *Continuous Integration*: A Jenkins build server was used to ensure continuous quality control and reveal integration errors early on. Every commit to the aforementioned Git repository was built in a clean workspace to reveal potential errors such as undeclared or clashing dependencies.

3) *Simulation*: Gazebo was used in combination with a dynamic model written in python to simulate new features. If new code passed this test, it was ready to be validated on the car. The simulation also proved to be a useful tool for preliminary controller tuning.

4) *Data Management*: Testing the autonomous system generated a considerable amount of data from different sources. A custom web browser based tool has been developed to efficiently manage all data through one interface. The information was structured as experiments with report annotation fields and nested test runs. Each test run contains a link to the source code (Git commit hash) that was run and the data (rosvbag) it generated.

5) *Telemetry and diagnostics*: A custom rqt plugin was designed to simplify Telemetry and Diagnostics. When launched in Telemetry mode the GUI would automatically connect the user's laptop to the car, provide a menu to launch and abort autonomous missions, and provide lightweight visuals to illustrate the car state. The GUI's Diagnostics mode allowed hybrid simulation and playback of rosvbags. The user could for instance use this to check if a change to path planning had no ill effects on control by replaying all topics excluding the planning and control packages, which would be rerun.

B. Computing assembly

A custom housing has been developed to safely embed the computing system in the car. It had to resist light debris and water sprayed up by the left front wheel. Additionally, it had to cool down electronics with a power rating of 170W and shield off EMI from the 480V 3 phase inverters running at 23KHz. These requirements were met with a hermetic, shielded and damped aluminum enclosure. Custom heat spreaders were used to channel the CPU heat from both computers to the walls by conduction. The air inside the box was cooled using 4 wall mounted forced convection heat exchangers. Passive heat sinks mounted on the outside of the box allowed the system to run at full power in steady state when driving at least 5 m/s. At last, the computing housing is mounted to the chassis with shock absorbing thermoplastic elastomer mounts to protect it against vibrations.

V. EXPERIMENTAL RESULTS

flüela was tested numerous times to guarantee robustness. It was tested on 8 different locations, on different closed racing tracks ranging from 100 to 500 meters. It was tested under heavy rain for several hours, including FSD, as well as under strong sun over 35°C . Throughout the testing season

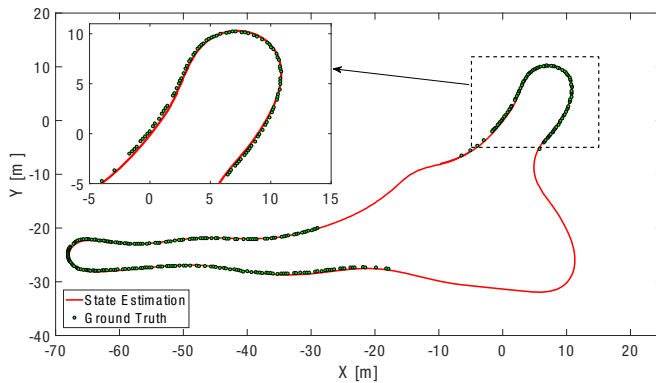


Fig. 6. The position from the state estimation (red) and the ground truth from the laser tracker (green circles) can be seen. The laser tracker could not follow the car at some locations due to its speed. The RMSE is 0.18m.

flüela reached over 90km/h and 1.7g lateral acceleration outperforming amateur drivers in some disciplines.

In this section, we present a performance evaluation based on filed experiments. A video of some experiments can be found at: https://youtu.be/NpLNJ5kC_G0 and a dataset used for some experiments is available at: https://github.com/AMZ-Driverless/fsd-resources#amz_driverless_2017

A. State Estimation

For the state estimation of the system four parts are evaluated and discussed: accuracy, robustness to outliers, self-diagnosis and delay compensation.

1) *Sensor fusion accuracy*: In order to validate the sensor fusion set up, the position estimate is compared to a ground truth provided by sub-mm precision Leica TotalStation 15i. When compared to ground truth the Root Mean Square Error (RMSE) of the estimated position is 0.18m. See Fig. 6. Note, that the TotalStation sometimes loses the target due to the high speed angular motion required to follow it. These locations are therefore not included in the evaluation.

2) *Robustness to outliers*: The outlier rejection method detailed in Sec. III-B.5 was developed to handle the different sensor faults which occurred during the testing phase. The first case of error (Fig. 7) is a velocity sensor which occasionally returns spikes when driving over reflecting surfaces such as water or some road markings. The χ^2 test could reject these cases without exception. The other case presented is the wheel odometry sensor. Due to the high accelerations of the car, one wheel is often blocked when braking and turning at the same time. This can be seen in Fig. 8. The χ^2 test is also used to reject this measurement in these scenarios. It has to be noted that, if wheel odometry is the only velocity source, and if the wheels are constantly blocked due to high accelerations, even with the χ^2 test the velocity estimate deteriorates.

3) *Self-diagnosis*: The self-diagnosis results are presented in Fig. 9 where 1 represents a perfect health and 0 all measurements being outliers. It can be seen that for the same track, the laps reaching 80km/h and 150°/s have a lower health diagnostic. This matches the fact that the medium speed laps have a 0.18m RMSE when compared to ground

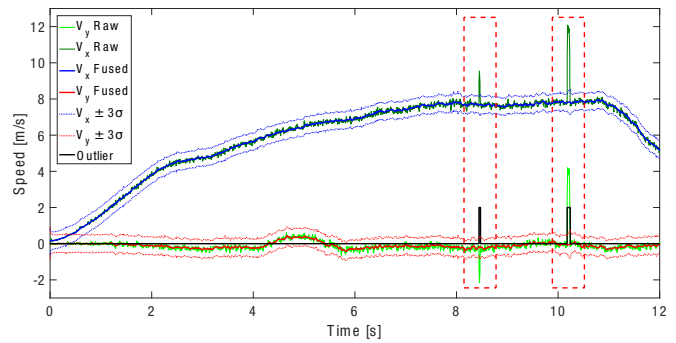


Fig. 7. Longitudinal and lateral velocity estimates (blue and red) and their 3- σ bounds are shown (dotted blue and dotted red). The raw velocity sensor (green) has two faults at $t \approx 8.45s$ and $t \approx 10.20s$. The outlier detector (black) spots these faults and rejects them based on the χ^2 test. Since these measurements are not introduced in the EKF, the velocity estimate is not corrupted.

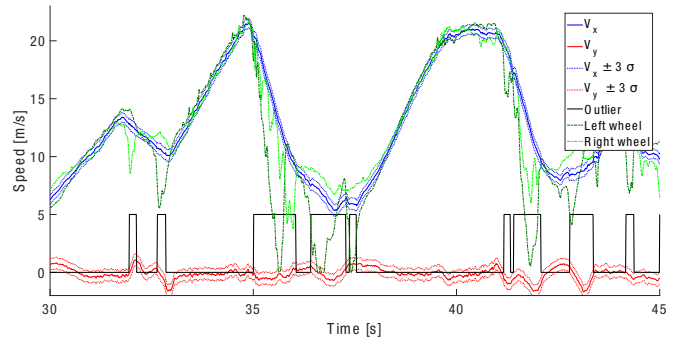


Fig. 8. Longitudinal and lateral velocity estimation (blue and red) and their 3- σ bounds are shown (dotted blue and dotted red). The rear wheels odometry (green) show that the wheels partially block when braking and turning. The χ^2 test can also be used to detect and reject these measurements (black) that do not represent the speed of the vehicle.

truth and an average health diagnostic of 0.987 whereas the fast laps have 0.39m RMSE and 0.896 average health diagnostic. This implies that the presented diagnosis method can provide an ad-hock qualitative estimate of the absolute error, without any ground truth information.

4) *Delay compensation*: In our experiments, the presented approximate delay compensation method is 5 times faster than the EKF intuitive compensation (Fig. 10). This varies depending on the most delayed measurement which in our case was from 40ms to 60ms (4 to 6 EKF iterations).

B. Mapping and Localization

The map built in real-time during the FSD competition can be seen in Fig. 11. The position estimated by the SLAM module is plotted within this map. Linking of the cones to form the boundaries is all done on-board and no manual changes were done to this map other than rotation and scaling for illustration purposes. With this data the particle filter was run at 5Hz using 500 particles with LiDAR cone observations as input. Wheel sensors combined with gyro integration was used as odometry input. The integrated gyro drifted almost 90° over the 10 plotted laps, while our localization on the loop-closed map performed robustly throughout. Note, that during the competition it rained heavily, yet our mapping and localization approach performed robustly under these conditions.

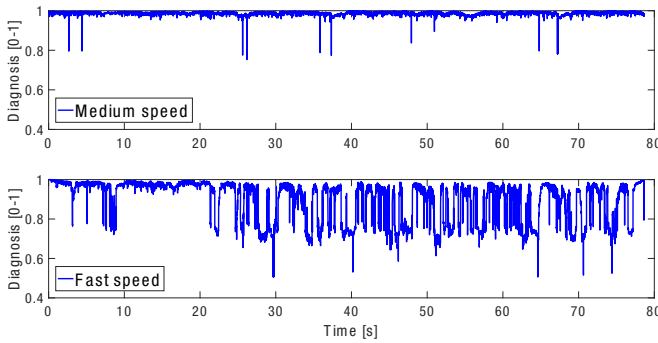


Fig. 9. The on-line self-diagnosis is shown for different laps in the same track. The medium speed laps result in a top speed of 30 km/h and maximum angular rate of $90^\circ/\text{s}$. The fast speed laps result in 80 km/h as top speed and $150^\circ/\text{s}$ as top angular rate. For the medium speed the mean diagnosis is 0.987 and lowest is 0.75. For the fast laps the mean is 0.896 and the lowest 0.5

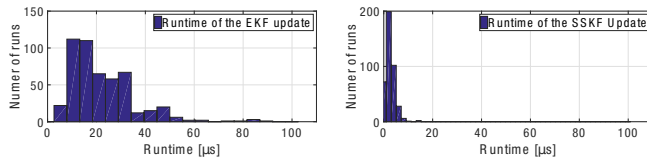


Fig. 10. The runtime of the EKF update (left) and the SSKF update (right) with delays are shown. For this setup with the most delayed measurement varying from 40ms to 60ms, the SSKF is around 5 times faster than the EKF.

On an Intel Core i7 7700HQ running at 2.8 GHz the filter update step takes in average 7ms during the mapping phase with a maximum of 29ms. During the localization phase the average computation time is 11ms with a maximum of 28ms. The computation time for the update step of the filter scales linearly with the amount of landmarks. This explains why the average time needed for the mapping phase is lower than for the localization phase.

As the update rate of 5Hz is too low for the control loops, the localization pose is fused with data from the other sensors at a higher rate, an example of this can be seen in Fig. 6.

VI. CONCLUSIONS

This paper presents the state estimation and system integration for an autonomous race car. It is capable of mapping a race track marked with cones using a landmark based SLAM system. Cones are detected with a 3D LiDAR using a two-stage clustering pipeline. The localization output of the SLAM system is used as a virtual position and heading sensor. Together with an INS and velocity sensor, it feeds an EKF based state estimator. Careful vehicle testing revealed the need to extend the state estimator with an outlier rejection and self-diagnosis system. The experiments show that the vehicle can race on unknown race tracks at competitive speeds, even when measurements are distorted due to adverse weather conditions. This perception and state estimation system shows potential to enable future autonomous race car generations to drive at lateral and longitudinal tire limits.

ACKNOWLEDGEMENT

The authors would like to thank the entire AMZ driverless team for their hard work and passion, as well as all sponsors for their financial and technical support. Furthermore

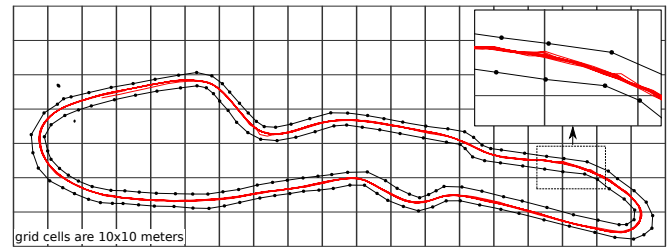


Fig. 11. A map built using the presented method with 500 particles. The estimated path coming from the mapping and localization module is depicted in red. The 10 laps were driven with a top speed limit of 8 m/s. Black dots are the estimated landmarks, in the top left the timing equipment was detected as a landmark, the rest are all cones. The zoom in shows the sharp edges due to the low update rate (5Hz)

we would like to thank the EU projects TRADR project No. FP7-ICT-609763, and European Unions Horizon 2020 research and innovation programme under grant agreement No 644227 (Flourish) and 688652 (UP-Drive) and from the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number 15.0029. and 15.0284 for their support.

REFERENCES

- [1] "Formula student rules 2017 v1.1," 2017.
- [2] M. B. K. Iagnemma, "Special issue on the darpa grand challenge, part 2," *Journal of Field Robotics*, vol. 23, pp. 661–692, September 2006.
- [3] J. Funke, P. Theodosis, R. Hindiyeh, G. Stanek, K. Kritatakirana, C. Gerdes, D. Langer, M. Hernandez, B. Mller-Bessler, and B. Huhnke, "Up to the limits: Autonomous audi tts," in *2012 IEEE Intelligent Vehicles Symposium*, pp. 541–547, June 2012.
- [4] A. Liniger, A. Domahidi, and M. Morari, "Optimization-Based Autonomous Racing of 1:43 Scale RC Cars," *Optimal Control Applications and Methods*, vol. 36, p. 628–647, July 2014.
- [5] U. Rosolia, A. Carvalho, and F. Borrelli, "Autonomous racing using learning model predictive control," in *2017 American Control Conference (ACC)*, pp. 5115–5120, May 2017.
- [6] S. Lynen, M. W. Achtelik, S. Weiss, M. Chli, and R. Siegwart, "A robust and modular multi-sensor fusion approach applied to mav navigation," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3923–3929, Nov 2013.
- [7] R. Hermann and A. J. Krener, "Nonlinear controllability and observability," *Automatic Control, IEEE Transactions on*, vol. 22, pp. 728–740, 11 1977.
- [8] T. D. Larsen, N. A. Andersen, O. Ravn, and N. K. Poulsen, "Incorporation of time delayed measurements in a discrete-time kalman filter," in *Proceedings of the 37th IEEE Conference on Decision and Control (Cat. No.98CH36171)*, vol. 4, pp. 3972–3977 vol.4, Dec 1998.
- [9] B. D. Brumback and M. Srinath, "A chi-square test for fault-detection in kalman filters," *Automatic Control, IEEE Transactions on*, vol. 32, pp. 552–554, 07 1987.
- [10] K. Hausman, S. Weiss, R. Brockers, L. Matthies, and G. S. Sukhatme, "Self-calibrating multi-sensor fusion with probabilistic measurement validation for seamless sensor switching on a uav," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4289–4296, May 2016.
- [11] M. Himmelsbach, F. v. Hundelshausen, and H. J. Wuensche, "Fast segmentation of 3d point clouds for ground vehicles," in *2010 IEEE Intelligent Vehicles Symposium*, pp. 560–565, June 2010.
- [12] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, et al., "Fastslam: A factored solution to the simultaneous localization and mapping problem," in *Aaai/iaai*, pp. 593–598, 2002.
- [13] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, Massachusetts: The MIT Press, 2006.
- [14] J. D. Hol, T. B. Schon, and F. Gustafsson, "On resampling algorithms for particle filters," in *Nonlinear Statistical Signal Processing Workshop, 2006 IEEE*, pp. 79–82, IEEE, 2006.