Comparison of Control Approaches for Autonomous Race Car Model

Jaroslav Klapálek*, Michal Sojka and Zdeněk Hanzálek

***Abstract*— Autonomous driving is an interesting but challenging field. To make it accessible to college students, several universities worldwide joined to create an F1/10 Autonomous Racing Competition, where students compete in a so-called battle of algorithms. When developing these algorithms or combining a few of them in more complex control approaches, it is not always easy to decide which approach is better. In this paper, we present a comprehensive methodology for comparing the control approaches based on four metrics: racing performance, obstacle avoidance slowdown, CPU load, and tuning effort. We apply the methodology to three conceptually different control approaches by performing an experimental evaluation with an autonomous race car model on two tracks. The results show that the difference between reactive and optimal approaches is small on a simple track, whereas on more complex tracks, the optimal approach has significant advantages. While these results were expected, the comparison shows the gap we will aim to close with our future control approaches.**

I. Introduction

Autonomous driving receives a lot of attention nowadays. According to the Washington Post article [1], Americans commute 52 minutes a day on average, and this number is increasing. Having self-driving cars would allow us to spend our traveling time more efficiently.

In their survey, Hussain and Zeadally reveal [2] that at least 14 large automakers, plus Google and Apple, are in the process of introducing an autonomous car. Each of these companies invests in the design, construction, and development of their self-driving car.

This focus of the automotive industry requires the universities to educate the students in the autonomous driving topics, to offer them a hands-on experience with related technologies, and to make them capable of designing, implementing, and evaluating such holistic experiments encompassing environment perception, data fusion, system modeling, control, decision making, optimization algorithms, and many technical issues. Therefore, several universities around the world chose to pursue autonomous driving in their way — to educate and train students in this challenging field using a scaled-down model having similar equipment like a real autonomous car. It substitutes the real car at a much lower cost, simpler safety requirement, and smaller space required to carry out the experiments. On the other hand, the scale-down model imposes strict constraints on the size, weight, and energy consumption of sensors and computer hardware.

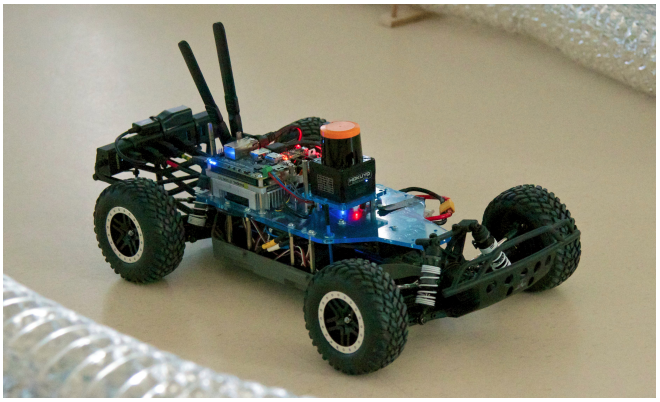


Fig. 1: Our F1/10 car model

Our colleagues from the University of Pennsylvania (USA), University of Virginia (USA), and the University of Modena and Reggio Emilia (Italy) chose to support this line of development by organizing a competition called *F1/10 Autonomous Racing Competition* [3], which is being held at least twice a year along with major conferences. The goal of the competition is not just to promote the scaled-down model cars but also to encourage student teams across the world to pursue autonomous driving while competing with others.

For the competition, the teams build the 1/10 scale model car, equip it with sensors, implement a control algorithm, and compete on the race track. The primary task is to drive fast to have the shortest lap time and drive safely, i.e., without crashing.

In this paper, we compare three control approaches to driving the F1/10 car autonomously. With the first approach, a simple reactive algorithm denoted as iFTG, we won the F1/10 competition in 2018. The second approach extends iFTG by exploiting the knowledge of the track map. The third approach relies on the off-line computation of close-to-optimal path, which is then followed at run time. The goal is to understand the strong and weak points of the approaches and to find directions for future improvements. We experimentally evaluate the approaches in four scenarios and compare them based on four metrics: racing performance, obstacle avoidance slowdown, CPU load ,and tuning effort.

The paper is organized as follows: in Section II, we discuss the related work. Section III describes the hardware and software components of our F1/10 car. In Section IV,

*The authors are with Faculty of Electrical Engineering, Department of Control Engineering, Czech Technical University in Prague, Technická 1902/2, 166 27 Prague, Czech Republic

All authors are with Czech Institute of Informatics, Robotics and Cybernetics (CIIRC), Czech Technical University in Prague, Jugoslávských partyzánů 1580/3, 160 00 Prague, Czech Republic (email: klapajar@fel.cvut.cz, michal.sojka@cvut.cz, zdenek.hanzalek@cvut.cz)

we describe the control approaches to be compared. The process of the evaluation as well the results are described in Section V. We conclude the paper in Section VI.

II. Related Work

The F1/10 competition [3] publishes a lot of information for the participating teams. The organizers describe [4], [5] a basic version of a scaled-down model of an autonomous car called *F1/10 Testbed* (or simply *F1/10 car*). The authors released instructions on building the car [6] and tutorials on the software stack design [7]. Supplementary materials also contain basic algorithms, such as remote control, reactive control, mapping, and localization. The organizers prefer using a specific sensor set and HW platform to keep equal conditions for the teams competing primarily on the algorithm level.

Control approaches for autonomous robots can be divided into three main categories [8]: (i) reactive, (ii) deliberative, and (iii) hybrid.

Reactive algorithms are memory-less and map-less methods for robot control. Every decision made by these algorithms depends only on the currently perceived situation. Reactive control approaches include, e.g., Potential Fields [9], Virtual Plane [10], or Follow the Gap [11]. In this paper, we build on top of the latter.

Deliberative algorithms use stored data (along with a map of the environment) to act with respect to past situations. These algorithms create a plan, and the actual control is performed according to this plan. This plan can be created using, e.g., RRT [12], or Collision-avoiding Potential Fields [13].

Hybrid algorithms are combining both previous approaches. After creating the plan, instead of strictly behaving according to it, some freedom is introduced to this process by means of a reactive algorithm. Therefore, the hybrid approach can react to sudden situations, which is useful when the planning phase takes a long time to compute. A hybrid approach was used, for example, in [14].

III. Car Hardware and Software

We have built our F1/10 car to comply with the rules of the competition [15] restricting some parameters. Specifically, we use the components described below, and, where appropriate, we comment on why these represent the best choice given the constraints in the rules.

While the operating system is given by the used onboard computer, the internal organization of the algorithms is unrestricted and, therefore, later described.

A. Hardware components

Our F1/10 car is built upon the chassis of *Traxxas Slash (TRA68086-24)* RC car. Slash is very similar to the *Traxxas Fiesta*, recommended by the competition organizers [16]. However, we argue that it is easier to start with the Slash as it already contains a brushless motor *Velineon 3500 (TRA3351R)* required by the rules. Slash is $568 \times 296 \times 324$ mm in dimension (stock), and, like Fiesta, it has a four-wheel drive.

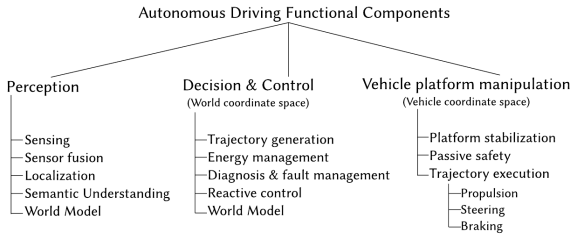


Fig. 2: Software component classification in an autonomous driving system (source [18])

We control the brushless motor by the VESC (Vedder's Electronic Speed Controller) [17], an open-source ESC created by Benjamin Vedder. Besides controlling the motor speed, VESC allows configuring many parameters that are vital for safe and reliable operation.

We can control the car either manually or automatically from the computer. Switching between the two modes is implemented in an on-car Arduino-like board *Teensy 3.2*. Board's firmware ensures that the manual control can always override automatic control.

For manual control, we use a 3-channel wireless transmitter *B3-STX Deluxe 2.4GHz F.H.S.S.*. It allows configuring various parameters, such as setting the boundaries for the speed or setting the left and right steering end-points independently. In contrast to the stock remote, *B3-STX* supports external charging, and it is less power demanding.

1) Sensors: The most important sensor of our car is the *Hokuyo UST-10LX* LiDAR. It measures the distance to the objects in $270°$ range up to 10 m with an angular resolution of $0.25°$ and with a frequency of 40 Hz. Assuming, that we drive at speed 4 m/s, the car receives LiDAR measurements every 10 cm.

Our car is also equipped with an inertial measurement unit *SparkFun 9DoF Razor IMU M0*. It helps to reduce the inaccuracies of the car localization by incorporating acceleration measurements into the localization process.

2) On-board computer: The algorithms run on the *NVIDIA Jetson TX2* embedded computer. We use the TX2 module with an *Orbitty Carrier* board, which is much smaller than NVIDIA's development carrier board.

Jetson runs *Ubuntu 16.04*, which makes it user-friendly for newer team members. Also, by selecting this Linux distribution, we can use all the support provided by the manufacturer.

B. Software Architecture

We use Robot Operating System (ROS) to run our control algorithms. The algorithms are written in a modular way, and each module is implemented as a separate component, called ROS node. ROS manages the data flow between the nodes.

The software architecture used to control our car is inspired by the architecture introduced in [18]. Our software components fall into three classes: *Perception*, *Decision and Control*, and *Vehicle Platform*. We describe the individual components in the rest of this section.

1) Perception: The goal of the Perception components is to provide all the required data for trajectory planning and car control. Perception ROS nodes perform several different tasks: Reading and filtering the data from the sensors or fusing the filtered data from multiple sensors. Other nodes perform obstacle detection, or Simultaneous localization and mapping (SLAM), for which we use Cartographer [19]. The outputs of the Perception nodes are fed to the Decision and Control nodes described next.

2) Decision and Control: Decision and Control nodes decide about car actions. In this work, we compare several different decision approaches and describe them in detail later in Section IV. Generally, these nodes comprise the path planners, reactive algorithms, path followers, and trajectory trackers.

Each chain (one or more nodes connected) outputs a desired speed and steering angle of the car that are processed by the Vehicle Platform components.

3) Vehicle Platform: Vehicle Platform components are distributed between the TX2 board and the Teensy board. We benefit from ROS's ability to automatically serialize and deserialize data to/from a serial line and process them both on a TX2 and Teensy.

The Vehicle Platform is composed of three main nodes – *Drive-API*, *VESC-driver*, and *Teensy-drive*. The former two runs on the TX2, whereas the latter runs on the Teensy board. The goal of these nodes is to control the car according to the control requests.

The *Drive-API* is a node providing a platform-independent interface (speed and steering angle) to the Decision and Control nodes. On each of our platforms (several cars or a simulator), this node converts the received data to the specific formats required by the actual platform, e.g., to the range of PWM signals.

The *Teensy-drive* node is a ROS node that runs on the Teensy board. Teensy-drive can run either in the manual mode or in the autonomous mode. When in the autonomous mode, Teensy-drive responds to an "emergency stop" triggered by touching the remote control transmitter. This action turns the car immediately off.

VESC-driver ROS node connects to the VESC via USB and controls the speed of the motor.

IV. Control Approaches

In this section, we describe approaches we compare in this paper. The first approach is the so-called *Follow the Gap* (FTG) [11], which won us the F1/10 competition in 2018, and we made its source codes public afterward [20]. Since then, other competitors have improved their algorithms and beat FTG. Therefore, in this paper, we try to understand the FTG, its weaknesses and compare it with different approaches. This should give us direction for the future development of our control strategies.

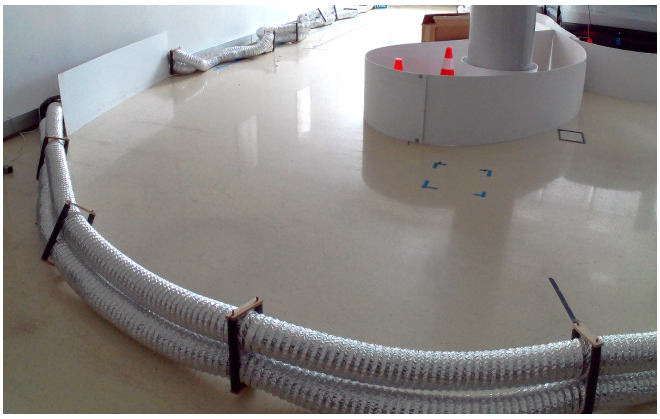


Fig. 3: Part of the testing track

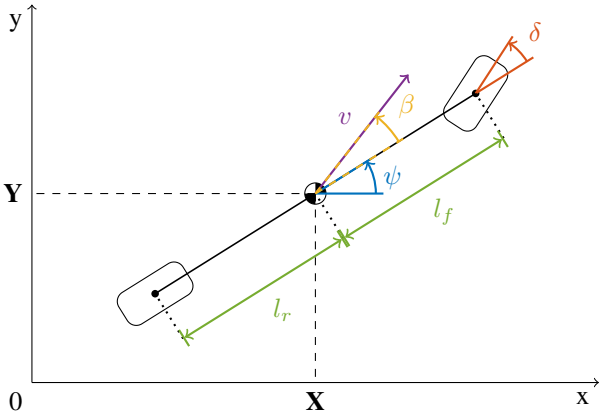


Fig. 4: Single-track model used to describe vehicle kinematics.

We compare the approaches in an environment that is as close as possible to the environment used at the competitions. That is, we have a flat track with no intersections. Track borders are marked with vertical boards and/or air-conditioning pipes with large diameters. Both options are easily movable and accurately perceivable by LiDARs. Our testing track is shown in Fig. 3.

A. Background and definitions

Before we describe the control approaches, we define few terms that are common to all of them.

a) Kinematic model: We describe the kinematics of the car using a *single-track* (also called bicycle) model [21] shown in Fig. 4. The model is represented by the following

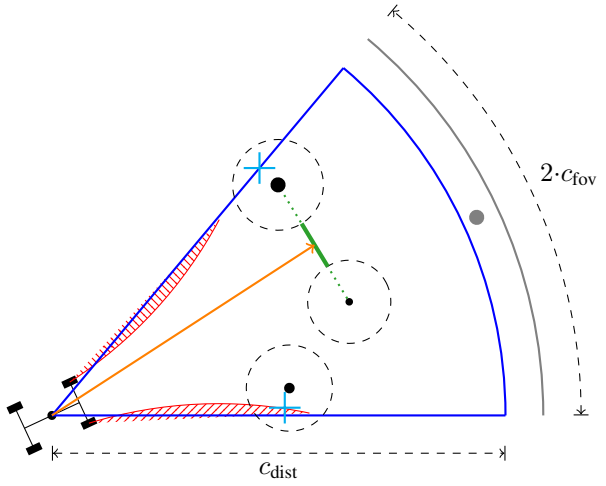


Fig. 5: Illustration of Follow the Gap algorithm. • obstacles in range and FoV (• outside),
- - enlarged obstacle (by car's size),
– active lidar range (– maximum range)
– nonholonomic constraints of the car,
+ border obstacles,
– largest gap,
– angle to the center of the largest gap.

non-linear equations:

$$\begin{aligned}
\dot{X} &= v \cdot \cos(\psi + \beta), \\
\dot{Y} &= v \cdot \sin(\psi + \beta), \\
\dot{\psi} &= \frac{v \cdot \cos\beta}{l_f + l_r} \cdot \tan\delta = v \cdot \cos\beta \cdot \kappa, \\
\dot{v} &= a, \\
\beta &= \tan^{-1}\left(\frac{l_r}{l_f + l_r} \cdot \tan\delta\right),
\end{aligned} \tag{1}$$

where v is the magnitude of car's velocity vector, l_f and l_r is the distance from the center to the front, and to the rear axle, respectively, δ is a *steering* angle, κ is an instantaneous curvature of the circle that would the car follow with current steering, and β is called *side-slip* angle.

b) Path and Trajectory: A sequence of car positions is called a *path*. *A trajectory* is a path where each position has associated a time when the car is at that position along with its speed, acceleration, and instantaneous path curvature.

B. Iterative Follow the Gap

The first control approach for the comparison is a reactive algorithm called *Iterative Follow the Gap* (iFTG), an extension of the Follow the Gap (FTG) algorithm proposed by Sezer and Gokasan [11].

We start by describing the FTG algorithm. Its principle is depicted in Fig. 5. Being a reactive algorithm, its output depends only on current information about the environment. It neither uses any information from the past nor tries to predict anything in the future.

The input to the algorithm is a sequence of obstacles detected by the LiDAR. Each obstacle is described using its position and size. The position is given in polar coordinates, d and ϕ relative to the car center. The size of each obstacle is enlarged by *car radius*, which is the maximal distance from the car center to its border. Therefore, the following steps can assume the car to be a single point.

The first step of the algorithm filters out the obstacles that are not important for the subsequent steps. Filtering is controlled by three parameters c_{dist}, c_{fov} and r. Specifically, the following classes of obstacles are discarded: (i) obstacles outside of the selected field of view ($|\phi| \geq c_{\text{fov}}$), (ii) obstacles further away from the vehicle ($d \geq c_{\text{dist}}$), and (iii) obstacles unreachable due to minimum steering radius r (nonholonomic constraints of the vehicle).

In the second step, so-called *border obstacles* are added as the first and last obstacle in the sequence. Their positions are given by distance of outer-most obstacles from the previous step and by parameters c_{fov} and r.

The last step of the FTG algorithm is to find the *largest gap* between obstacles. The gap is the distance between adjacent obstacles. The largest gap is depicted with a green line in Fig. 5. The algorithm returns the angle to the center of the largest gap (orange line in Fig. 5).

Our iFTG algorithm calls FTG iteratively with different values of c_{dist} and c_{fov}. We start with $c_{\text{dist}} = 4\,\text{m}$ and continue with $0.5\,\text{m}$ steps down to $2\,\text{m}$. The reason is that for large c_{dist}, there might be no gap visible in case of sharp turns. If we use only small c_{dist}, we might not have enough time to avoid the detected obstacle when driving fast.

The output of the iFTG algorithm is the steering angle, but to drive the vehicle, we also need the speed. We calculate the speed based on the steering angle. We use only three speed values called *slow*, *medium*, and *fast*. Fast speed is used on straight track segments and slow at sharp turns. These speeds are configurable as parameters. The hysteresis of switching between different speeds is also configurable. In total, tuning the iFTG algorithm for a particular track requires a change of 10 parameters.

C. Map-based Follow the Gap

The Iterative Follow the Gap algorithm suffers from the fact that it does not know the layout of the whole race track. Therefore, its speed output has to be conservative, and the car cannot drive at maximum speed at long straight segments. The Map-based Follow the Gap (MBFTG) algorithm extends the iFTG to take advantage of the knowledge of the track map. It combines the reactivity of the original Follow The Gap with a position-driven speed profile.

MBFTG works in two phases: initialization and execution. In the initialization phase, we drive once through the track with conservatively tuned iFTG and record the vehicle's path reported by the Cartographer SLAM algorithm [19]. SLAM algorithms work by creating a map of the environment and simultaneously localizing the car in the map. After the recorded path forms a loop, we calculate the speed profile of the path to obtain a trajectory and transition to the execution phase.

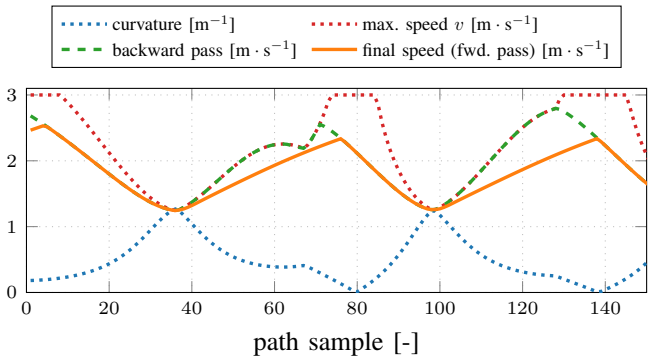


Fig. 6: Calculation of the speed profile

To calculate the speed profile, we need a twice-differentiable path. Therefore, we take a subset of points from the recorded path and interpolate it with a cubic spline. The resulting path has smooth curvature, which is a necessary condition for the path to be followed by a car. The curvature, κ, at each point of the path is computed as:

$$\kappa = \frac{x' \cdot y'' - y' \cdot x''}{\left[(x')^2 + (y')^2\right]^{\frac{3}{2}}}, \tag{2}$$

where x and y are coordinates of the points on the path.

Next, we compute the speed profile of the interpolated path using a two-pass iterative algorithm introduced in [22]. The algorithm first calculates the maximum speed v that avoids car side-slips as

$$v = \min(\sqrt{\frac{\mu \cdot g}{\kappa}}, v_{max}), \tag{3}$$

where μ is a tire friction coefficient, g is gravitational acceleration and v_{max} is maximum allowed speed. Then the speed is further decreased in so-called backward and forward passes to not violate car's maximum deceleration and acceleration. This is illustrated in Fig. 6.

In the execution phase of the MBFTG, we use the iFTG algorithm to compute only the steering angle. The speed is calculated as follows. The SLAM algorithm estimates the position of the car. We find the closest point on the trajectory constructed in the initialization phase and use the speed component of the trajectory.

D. Static Optimal Trajectory Tracking

Since the last competition showed that even MBFTG is not sufficient to win the race, better approaches are needed. Optimal control algorithms, e.g., those based on Model Predictive Control (MPC), are promising, but both identification of the needed model and online execution of the algorithm on an embedded computer are very challenging tasks. In this work, we are interested in understanding how far optimal trajectory planning can improve the lap times compared to the previously mentioned approaches. Therefore, we compare them with a deliberative approach, where an optimal trajectory is calculated off-line. At run-time, we just follow that optimal trajectory using a simple trajectory tracker. We call this approach Static Optimal Trajectory Tracking, or SOTT in short.

We construct the optimal trajectory by using a genetic algorithm similar to the one proposed by Braghin et al. [23]. To evaluate the quality of the paths generated by the genetic algorithm, we construct the speed profile in the same way as in the case of MBFTG (Section IV-C) and calculate the time needed to drive the trajectory. This way, the genetic algorithm finds the path with the lowest possible lap time.

Our car follows the planned trajectory using the *Pure Pursuit* algorithm [24], which controls the steering angle, and a P controller for speed control. The Cartographer SLAM algorithm estimates the actual position of the car.

V. Evaluation

To compare the control approaches introduced in Section IV, we define several metrics and experimentally evaluate the approaches based on these metrics in two different tracks with and without obstacles.

A. Metrics

Based on our years-long experience with the F1/10 platform and participation in the F1/10 competitions, we consider the following metrics essential for assessing the quality of the control approaches: *racing performance*, *obstacle avoidance*, *hardware load*, and *tuning effort*. Below, we describe the details of these metrics and how we measure them.

a) Racing Performance: Racing performance corresponds to the minimal lap times achieved by the control approach on the track without obstacles.

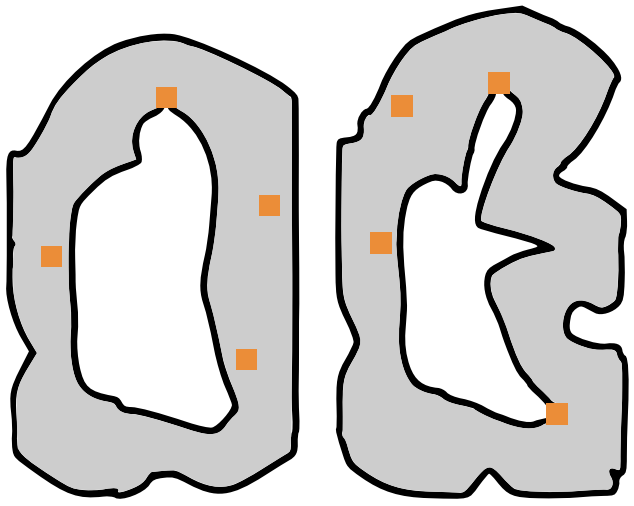
To measure the lap time, we rely on car localization by the SLAM algorithm. Whenever the car crosses a virtual starting line, we record the elapsed time. We always record several consecutive lap times and present their minimum, average, and standard deviation. Each control approach is manually tuned to perform the best on the given track.

b) Obstacle Avoidance Slowdown: One of the competition requirements is to successfully avoid obstacles that are placed on the racing track. We evaluate the ability to cope with obstacles by comparing the minimal lap times on the track with obstacles and without them (racing performance). Specifically, we define the obstacle avoidance slowdown metric as

$$\left(\frac{lt_{obst}}{rp} - 1\right) \cdot 100, \tag{4}$$

where lt_{obst} is the minimal lap time on the track with obstacles and rp is the racing performance.

The value of this metric is obtained by placing several static obstacles detectable by LiDAR on the track, optionally changing the control approach parameters to avoid the obstacles reliably, and measuring minimal lap time the same way as for the racing performance metric.



(a) Track 1 (T1) (b) Track 2 (T2)

Fig. 7: Tracks used for comparison. Each track is used both with and without obstacles (marked with ■)

c) CPU Load: This metric shows how the computations of the given control approach load the CPU cores of the system. It is essential that the system is not overloaded. Otherwise, the control performance is degraded, and the system becomes unreliable.

We use the 5-minute load average metric provided by Linux. This value is obtained after several minutes of driving the car with the given control approach. As the NVIDIA TX2 has six CPUs, a load greater than 6 means that the system is overloaded. Note that the load we report is generated by the control approach itself and by data gathering tools to we used in addition. Since we run those tools together with all control approaches, the differences in the CPU load metric correspond to the differences in the load generated by the control approaches.

d) Tuning Effort: Each control approach requires some manual tuning and tweaking to achieve good lap times without collisions. We express the effort of this tuning with two values. The first is the number of parameters that we configure for the given approach, and the second is a purely subjective rating of the difficulty of the tuning process.

B. Tracks

In this section, we describe the two tracks used for performing the experiments in our lab.

The tracks are depicted in Fig. 7. Track 1 (T1) represents a typical layout of an F1/10 competition racing track with an ellipse-like shape. Track 2 (T2) contains more turns, which usually causes trouble to reactive algorithms. Each track was used both with and without the obstacles (orange squares in Fig. 7). We denote the scenarios with obstacles as *obs*.

In summary, each control approach was evaluated in four different scenarios: T1, T1obs, T2, and T2obs.

C. Results

In this section, we describe the experimental results measured in the scenarios described above. The lap times measured for different approaches can be seen in Table I.

Scen.	Appr.	Lap time Min.	Avg.	SD
T1	iFTG-ref	12.12 s	12.29 s	0.11 s
	iFTG	8.84 s	8.96 s	0.11 s
	MBFTG	8.52 s	8.76 s	0.10 s
	SOTT	**8.20 s**	**8.26 s**	**0.05 s**
T1obs	iFTG-ref	12.64 s	12.83 s	0.09 s
	iFTG	8.72 s	9.18 s	0.34 s
	MBFTG	8.80 s	8.92 s	0.24 s
	SOTT	**7.88 s**	**7.97 s**	**0.08 s**
T2	iFTG-ref	13.20 s	14.03 s	0.38 s
	iFTG	12.20 s	12.65 s	0.38 s
	MBFTG	12.20 s	12.45 s	0.17 s
	SOTT	**9.77 s**	**9.91 s**	**0.10 s**
T2obs	iFTG-ref	14.25 s	14.78 s	0.50 s
	iFTG	13.41 s	13.82 s	0.28 s
	MBFTG	14.12 s	14.34 s	0.23 s
	SOTT	**10.56 s**	**10.76 s**	**0.13 s**

TABLE I: Lap time measurements for compared approaches – Iterative Follow the Gap (iFTG), Map-based Follow the Gap (MBFTG), and Static Optimal Trajectory Tracking (SOTT). Measurements for the baseline iFTG-ref are also shown. The best values are shown bold, and the worst are shown red.

Since each scenario is different, we set up a baseline for each of them. We call this baseline *iFTG-ref*, and it is the lap time obtained with a conservatively configured iFTG approach, where the iFTG configuration was the same in each scenario. The other approaches were configured differently for each scenario to achieve the best possible lap times.

The values of our metrics can be seen in Table II. The paths generated by different control approaches are shown in Fig. 8. In the following, we discuss the performance of the individual approaches.

a) Iterative Follow the Gap: We can see from Table II, iFTG has the worst racing performance on simple tracks, such as T1. The reason is an inability to reach maximum speed on straight track segments. For more complex tracks, iFTG can sometimes beat the MBFTG. However, standard deviations of lap times, shown in Table I, suggest that those good results are not delivered deterministically.

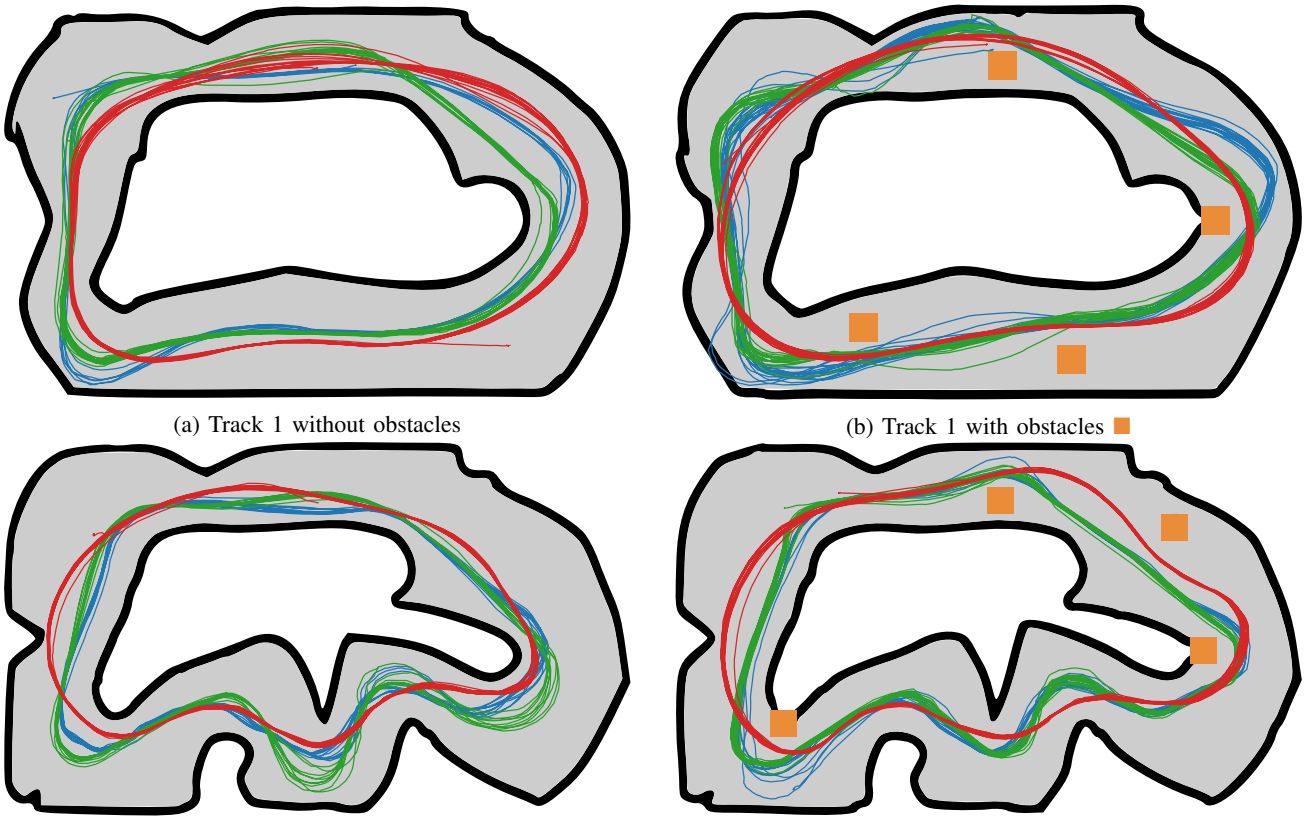
Concerning the CPU load, iFTG takes second place due to the execution of multiple iterations of the FTG.

Additionally, to achieve good performance, one has to put significant effort into tuning this algorithm to the particular track. In our implementation, we have 10 speed-related constants that have to be tuned manually. Their number could be reduced, but it would negatively affect iFTG's performance.

b) Map-based Follow the Gap: Table II shows that MBFTG loses in all metrics except for racing performance on simple track T1. Obstacle avoidance slowdown is quite significant and is caused by the conservativeness of the initial iFTG-ref, which is used for speed profile calculation. Comparison of iFTG and MBFTG paths in Fig. 8b shows that the reference algorithm generated the path with sharper

Approach	Racing Performance		Obst. Avoidance Slowdown		CPU load		Tuning effort (# parameters, comment)
	T1	T2	T1	T2	T1	T2	
iFTG	8.84 s	12.20 s	−1.33 %	9.94 %	3.61	3.61	10, time-consuming
MBFTG	8.52 s	12.20 s	3.33 %	15.74 %	3.92	4.13	2, tuning is quite straightforward, but
SOTT	**8.20 s**	**9.77 s**	**−4.06 %**	**−8.11 %**	**3.06**	**3.41**	requires information about the track

TABLE II: Comparison of the control approaches by means of our metrics



(a) Track 1 without obstacles

(b) Track 1 with obstacles ■

(c) Track 2 without obstacles

(d) Track 2 with obstacles ■

Fig. 8: Recordings of the vehicle path when driving counterclockwise. Colors represent different control approaches: iFTG - Iterative Follow the Gap (in blue), MBFTG Map-based Follow the Gap (in green) and SOTT Static Optimal Trajectory Tracking (in red). Collisions with the track borders visible in the recordings are caused by the localization inaccuracy.

turns, which led to lower speeds in the speed profile.

CPU load is the highest because the algorithm uses linear search to find the closest point on the reference trajectory. The higher load on longer track T2 is caused by more points on the reference trajectory and thus longer search times. CPU load could be decreased by implementing the search more efficiently.

The tuning effort required for MBFTG is, in comparison to iFTG, much lower. In this algorithm, only two parameters are tuned – friction coefficient and a path subset factor that adjusts the number of points used for the speed profile.

c) Static Optimal Trajectory Tracking: SOTT is a clear winner in all metrics. However, this comparison is not completely fair because the other algorithms can avoid at least some obstacles in apriori unknown positions contrary to SOTT, which blindly tracks the precomputed trajectory. On the other hand, SOTT shows that there is a big potential for improving the other algorithms.

Interestingly, the presence of obstacles decreases the lap time (i.e., the slowdown is negative). The reason can be seen in Fig. 8b where SOTT achieves smaller and more uniform curvature at the top left of the track, in comparison to Fig. 8a. This is probably caused by the genetic algorithm being stuck in the local minimum on the track without obstacles. Another possible explanation is that the search space of the track with the obstacles is smaller, helping to find better results faster.

CPU load is the lowest, as the algorithm (in the execution phase) runs only a trajectory tracker. Similarly to MBFTG, the load depends on the trajectory length.

The tuning effort for SOTT is the same as for MBFTG, as there are the same two parameters. Path subset factor is used to smoothen the path generated by the genetic algorithm.

VI. Conclusion

In this paper, we have described our scaled-down model car for participation in F1/10 Autonomous Racing Competition. We have presented and implemented three different control approaches, ranging from a simple reactive approach (iFTG) to the approach based on tracking a precomputed, close-to-optimal trajectory (SOTT). We experimentally evaluated the approaches using four metrics. SOTT is a clear winner by all of these metrics. Even though this approach is applicable to the time trial race, it is not suitable for the upcoming head-to-head race, with multiple cars on the track. The performance of reactive approaches is not satisfactory, especially on more complex tracks, and the presence of obstacles decreases the performance even more. While such results were expected, the comparison with the close-to-optimal approach shows the gap our future algorithms should aim to close. Therefore, the work presented in this paper will serve as a baseline for assessing the performance of control approaches developed in the future.

Acknowledgement

Research leading to these results has received funding from the EU ECSEL Joint Undertaking and the Ministry of Education of the Czech Republic under grant agreement 826452 and 8A19011 (project Arrowhead Tools).

References

[1] "The astonishing human potential wasted on commutes." https://www.washingtonpost.com/news/wonk/wp/2016/02/25/how-much-of-your-life-youre-wasting-on-your-commute/. Published: 25 February 2016.

[2] R. Hussain and S. Zeadally, "Autonomous Cars: Research Results, Issues, and Future Challenges," *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, pp. 1275–1313, 2019. Conference Name: IEEE Communications Surveys Tutorials.

[3] "F1Tenth website." http://f1tenth.org/. Accessed: April 2020.

[4] M. O'Kelly, V. Sukhil, H. Abbas, J. Harkins, C. Kao, Y. V. Pant, R. Mangharam, D. Agarwal, M. Behl, P. Burgio, and M. Bertogna, "F1/10: An Open-Source Autonomous Cyber-Physical Platform," *arXiv:1901.08567 [cs]*, Jan. 2019. arXiv: 1901.08567.

[5] M. O'Kelly, H. Zheng, A. Jain, J. Auckley, K. Luong, and R. Mangharam, "Tunercar: A superoptimization toolchain for autonomous racing," 2020.

[6] "F1Tenth website / build." https://f1tenth.org/build.html. Accessed: April 2020.

[7] "F1Tenth website / learn." https://f1tenth.org/learn.html. Accessed: April 2020.

[8] R. C. Arkin and D. C. MacKenzie, "Planning to Behave: A Hybrid Deliberative/Reactive Robot Control Architecture for Mobile Manipulation," 1994. Accepted: 2008-05-29T19:48:48Z Publisher: Georgia Institute of Technology.

[9] H. Haddad, M. Khatib, S. Lacroix, and R. Chatila, "Reactive navigation in outdoor environments using potential fields," in *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, vol. 2, pp. 1232–1237 vol.2, May 1998. ISSN: 1050-4729.

[10] F. Belkhouche, "Reactive Path Planning in a Dynamic Environment," *IEEE Transactions on Robotics*, vol. 25, pp. 902–911, Aug. 2009. Conference Name: IEEE Transactions on Robotics.

[11] V. Sezer and M. Gokasan, "A novel obstacle avoidance algorithm: "Follow the Gap Method"," *Robotics and Autonomous Systems*, vol. 60, pp. 1123–1134, Sept. 2012.

[12] S. M. Lavalle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," tech. rep., 1998.

[13] J. Barraquand and J.-C. Latombe, "Robot Motion Planning: A Distributed Representation Approach," *International Journal of Robotic Research - IJRR*, vol. 10, pp. 628–649, Dec. 1991.

[14] S. Stock, M. Mansouri, F. Pecora, and J. Hertzberg, "Hierarchical Hybrid Planning in a Mobile Service Robot," Sept. 2015.

[15] "F1Tenth Rules v2." http://f1tenth.org/race/rules-v2.pdf. Published: February 2019.

[16] "Bill of Materials – F1TENTH - build latest documentation." https://f1tenth.readthedocs.io/en/stable/getting_started/build_car/bom.html. Accessed: April 2021.

[17] "VESC Project." https://vesc-project.com/. Accessed: April 2020.

[18] S. Behere and M. Törngren, "A Functional Architecture for Autonomous Driving," in *Proceedings of the First International Workshop on Automotive Software Architecture*, WASA '15, (New York, NY, USA), pp. 3–10, ACM, 2015. event-place: Montréal, QC, Canada.

[19] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-Time Loop Closure in 2D LIDAR SLAM," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1271–1278, 2016.

[20] A. S. Pedersen and J. Klapálek, "Follow The Gap." https://github.com/CTU-IIG/f1t-ftg, 2020.

[21] R. Rajamani, *Vehicle Dynamics and Control*. Mechanical Engineering Series, Springer US, 2 ed., 2012.

[22] N. R. Kapania, J. Subosits, and J. Christian Gerdes, "A Sequential Two-Step Algorithm for Fast Generation of Vehicle Racing Trajectories," *J. Dyn. Sys., Meas., Control*, vol. 138, Sept. 2016. Publisher: American Society of Mechanical Engineers Digital Collection.

[23] F. Braghin, F. Cheli, S. Melzi, and E. Sabbioni, "Race driver model," *Computers & Structures*, vol. 86, pp. 1503–1516, July 2008.

[24] J. M. Snider, *Automatic Steering Methods for Autonomous Automobile Path Tracking*. PhD thesis, Carnegie Mellon University, Robotics Institute, Feb. 2009.