f1tenth.dev - An Open-source ROS based F1/10 Autonomous Racing Simulator

Varundev Suresh Babu and Madhur Behl
Department of Computer Science
University of Virginia, Charlottesville, USA
{varundev and madhur.behl}@virginia.edu

***Abstract*—This paper presents the ROS F1/10 simulator - a ROS and Gazebo based autonomous racing simulator designed to mirror the behaviour and performance of the physical F1/10 platform. The simulator can be used to verify the performance of autonomous racing algorithms before testing on the real F1/10 racecar. The simulator supports most major ROS SLAM and navigation packages with tutorials on how to get started available on www.f1tenth.dev under open-source license and we continue to add support for more algorithms and features. The paper presents details on the simulator architecture, design, and features and presents several research use-cases including an example of developing in the sim and transferring the performance to the race racecar.**

I. Introduction

The F1/10 autonomous platform is a fully autonomous Ackermann-steering racing robot inspired by Formulae 1 (F1). It is one-tenth the scale of a full scale racecar. The F1/10 platform [1] is designed to make autonomy easily accessible, and it is fully capable of implementing the entire perception, planning & control pipeline using on-board components in real-time. The F1/10 platform project is open-source and made available on f1tenth.org. The F1/10 racecar is highly modular and is built on top of a realistic one-tenth scaled Traxxas Rally car chassis with a four-wheel drive brushless power train capable of rapid acceleration and fast cornering. The F1/10 racecar can be equipped with a varying sensor suite with a 2D scanning LiDAR as the primary navigation sensor and secondary sensors include Inertial Measurement Units (IMU), RGB cameras and 3D stereo depth cameras. The sensing and actuating components are controlled using an powerful onboard embedded computer (usually the NVIDIA Jetson TX2) with Robot Operating System (ROS) and associated packages as the default robot control software. The F1/10 platform has enabled research in robotics, machine learning, cyber-physical systems, and embedded systems across multiple research group and universities in the world [2]–[9].

Figure 1 provides an overview of the F1/10 simulator, and the project is made available at www.f1tenth.dev The F1/10 racecar makes vehicular autonomy easily accessible, and there is a lot of interest in the F1/10 racecar from students & researchers, but the slightly higher price tag (US$3000-4500) depending on the sensor configuration (e.g. high speed and range LIDAR), makes the physical platform still outside the reach of many researchers. In addition, sourcing all the parts to build the car has recently become a major hassle. The lack of a dedicated F1/10 simulator also makes the algorithm development cycle longer and ad-hoc, since the only way to test the racing algorithms is to implement them on the physical car. The ROS F1/10 simulator is designed to address these limitations of the F1/10 platform. The primary contributions of this paper are:

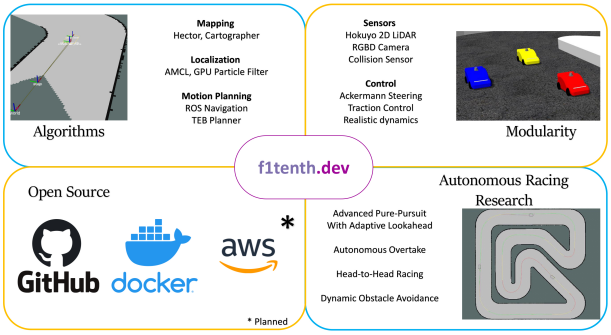

Fig. 1: Overview of F1/10 simulator - an autonomous racing simulator based on ROS. It is highly modular in terms of sensors and capabilities, works with most major ROS SLAM and navigation packages, and it is currently being used for research by major research universities

1) A ROS F1/10 simulator developed in Gazebo that is designed to mirror the performance and behavior of a real F1/10 autonomous racecar.
2) The simulator supports an extensive suite of perception, planning, and control research algorithms and provides `roslaunch` scripts to make them easy to implement.
3) We demonstrate the fidelity of our simulator by showing how one can transfer algorithms designed in simulation to the real F1/10 with ease.

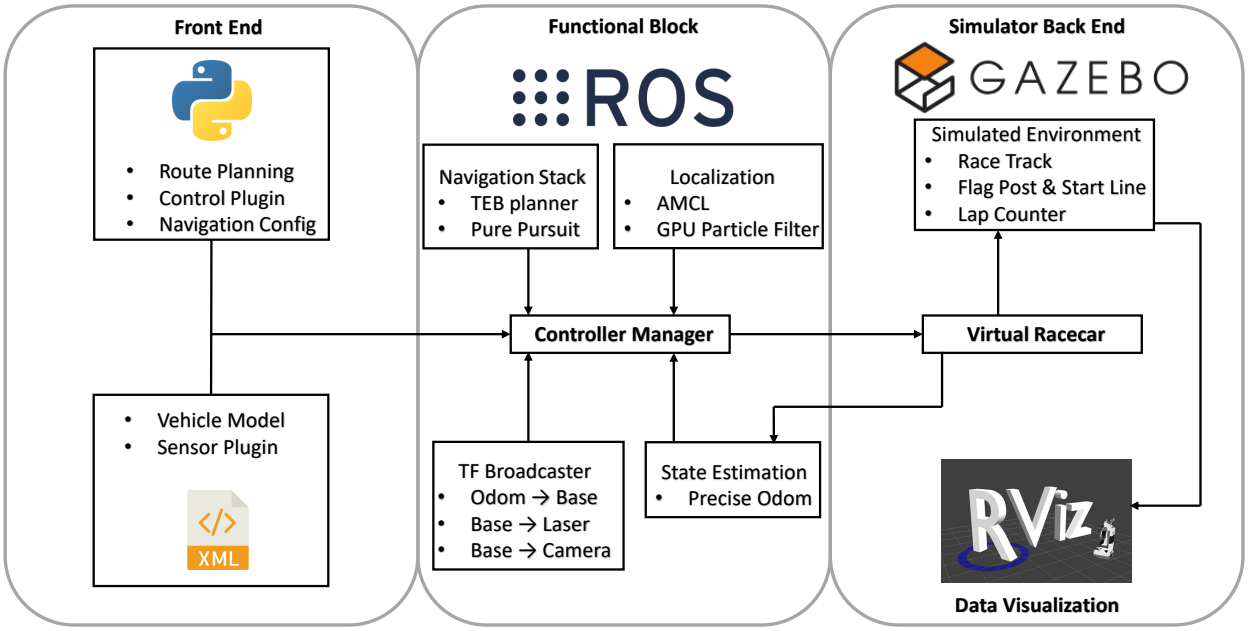

Fig. 2: The architecture of F1/10 simulator simulator [Left] front end with the ROS Python API and high level algorithmic parameters, desired racing behavior and a control plugin (with a command multiplexer to prioritize the desired navigation stacks), and the Xacro vehicle description, [Center] the ROS based functional block with feed forward and feed back mechanisms with the controller manager providing synchronization services, [Right] the simulator back end consisting of Gazebo, the racing environment, telemetry services and the visualization tools

II. Related Work

The core of the F1/10 simulator is ROS [10] and Gazebo [11] and their extensive list of supported libraries, and as such heavily borrows the design principles of those projects. Apart from the simulator and its back end components, the F10/10 simulator supports racing metrics and strategies employed in popular motor sports like Formulae 1, Indianapolis 500 etc. using a dedicated telemetry system. While there are many autonomous vehicle simulators available like Carla, Apollo, etc. and they can theoretically be used to test racing algorithms, that was never their primary purpose, and as such impose lots of barriers when testing high speed navigation and obstacle avoidance. Simulators like Apollo were not designed for autonomous racing, but rather for testing autonomous vehicles in an urban environment or interstate highways. Simulators like Carla [12] are open-source and relatively lightweight, but still focused on autonomous driving research in similar settings as Apollo. Autonomous racing differs from autonomous driving in certain key areas like higher speeds, lack of minimum safe following distance & lack of lanes, etc. The main limitations of the simulators above is the lack of a one-to-one correspondence with a physical testbed. The Amazon DeepRacer [13] is a similar but less capable hardware platform compared to the F1/10 racecar. It also has a simulated environment, but the entire DeepRacer platform is focused mostly on enabling Reinforcement Learning (RL). The DeepRacer is also a standalone platform with no out-of-box support for multi-vehicle operation. In addition to the complexity of the simulators themselves, the threshold to get started is very high both in terms of computer hardware and experience, and this is where F1/10 simulator tries to fill in the gap - by being a lightweight autonomous racing simulator and providing an easy to learn Python API, F1/10 simulator make vehicular autonomy easily accessible. The virtual racecar in F1/10 simulator is based on the specifications designed by the ROS Ackermann Interest Group, and depends on the GPU enabled fast particle filter [14].

III. Simulator Architecture

The F1/10 simulator has been built around the design principles of modularity and expandibility, and is based entirely on the open-source projects of Robot Operating System (ROS) and Gazebo physics simulator. The simulator architecture is contains three modules: simulation front end, functional-block and simulation back end.

A. Virtual Racecar

The virtual racecar in the F1/10 simulator is a collection of visual and functional components joined at a constant offset to a geometric tracking frame that simulates the sensing and performance capabilities of the F1/10 autonomous racecar platform. It is described using a single XML macros module and is designed to be highly modular with Boolean flags that can set or

reset all functional components of the virtual racecar. The components of the virtual racecar are divided into the following three sections, Figure 3 shows the features of the virtual racecar derived from the real F1/10 racecar:

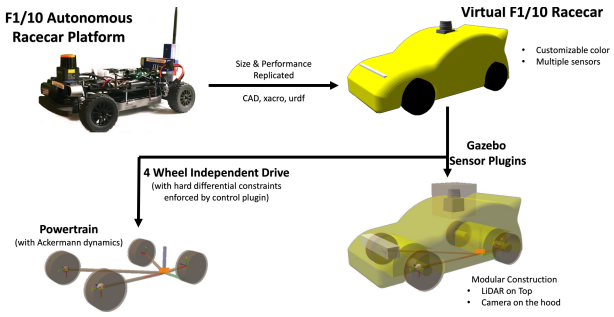

Fig. 3: Clockwise from top left - The real F1/10 racecar; the virtual F1/10 racecar as seen on the Gazebo GUI; the modular sensor suite with 2D LiDAR & stereo camera; the 4-wheel virtual drive train

1) Virtual Sensing Components: The F1/10 virtual racecar is equipped with a wide variety of sensors including 2D laser scanners, monocular and stereo RGBD (RGB + depth) cameras, IMUs (Inertial Measurement Units), rotational encoders and contact sensors. The sensors output a raw ROS standard message type and the simulator uses sensor filters to provide accurate estimates of the pose and velocity information of the virtual racecar using the ROS *robot_pose_EKF* (Extended Kalman Filter) [15] [16].

2) Virtual Actuation Components: The F1/10 virtual racecar contains six independent actuators with a governing Ackermann-steering vehicle controller. Four of the actuators are PID enabled torque controllers and the other two are PID steering hinge angle controllers [17]. The actuation control nodes can accept ROS *Twist* or *AckermannDrive* type input and enforces strict Ackermann-steering geometry constraints and the resulting differential wheel torque constraints.

3) Modular Components: Each component in the F1/10 virtual racecar can be enabled or disabled according to user needs. Amongst the sensor suite, the XML macros module describing the sensor and its operation consists of a Boolean flag to enable/disable that particular sensor, and amongst the actuation suite - the virtual racecar can be converted from four-wheel drive to two-wheel (front axle or rear axle) and the torque differential drive components can be enabled or disabled using the accompanying Boolean flag.

B. Simulator Front-End

The simulator front-end is divided into the F1/10 specific ROS Python API and the racecar description Xacros (XML Macros). The ROS Python API is used to perform the ROS publish and subscribe functions along with maintaining the ROS parameters server and ROS services, while the racecar description Xacros are used to determine the functional and visual parameters of the virtual racecar and its components.

1) ROS Python API: These are a set of ROS Python nodes that are used to communicate with the virtual racecar and perform a variety of functions including remote tele-operation using keyboard/joystick, read sensor data from the virtual racecar and publish the actuation commands from a higher level navigation stack. The API also enforces actuation constraints like steering wheel velocity and final drive wheel RPMs using realistic PID controllers, a collision avoidance node that helps protect the virtual racecar, and a $base\_footprint$ projection node that informs any high level navigation about the vehicle's non-zero geometric footprint.

2) Racecar Xacros: These are a set of XML Macros as defined in ROS that encode the functional components of the virtual racecar including (a) sensors: LiDAR, stereo camera, contact sensors and rotational encoders, and (b) actuators: torque controllers, position and joint controller, and steering hinge controllers, and also the visual or aesthetic components of the racecar like the shape and size of the racecar chassis, sensors and wheels. The entire virtual racecar has one unique identifiable geometric frame called its $base\_link$ and all components of the racecar are tethered to this link with finite geometric offsets. For ease of use with the higher level navigation stacks, the $base\_link$ is located at the geometric center of the rear drive-axle of the virtual racecar according to Ackermann-steering design [18].

C. Functional Block

The functional block module of the simulator is where most of the ROS based functions reside, the most important of which is the controller manager. In addition to the controller manager, the feed-forward control mechanisms (including navigation) and feed-back mechanisms (including state estimation filters) work in tandem to ensure the synchronization of the virtual racecars' command & control nodes.

1) Controller Manager: The controller manager is a ROS package used to enforce hard real-time constraints to the virtual racecar, and it ensures that the timing mechanisms governing the sensors and actuators control loops work synchronously. Within the F1/10 simulator, each racecar is equipped with an independent controller manager which further acts as a single point of synchronous command & control governor for each racecar.

2) Feed-Forward Mechanism: The feed-forward mechanism of the simulator is the section of the functional block that act as the navigation and control ROS nodes responsible for guiding the virtual racecar using a desired approach. High level navigation stacks are deployed in the feed-forward mechanism section along with the

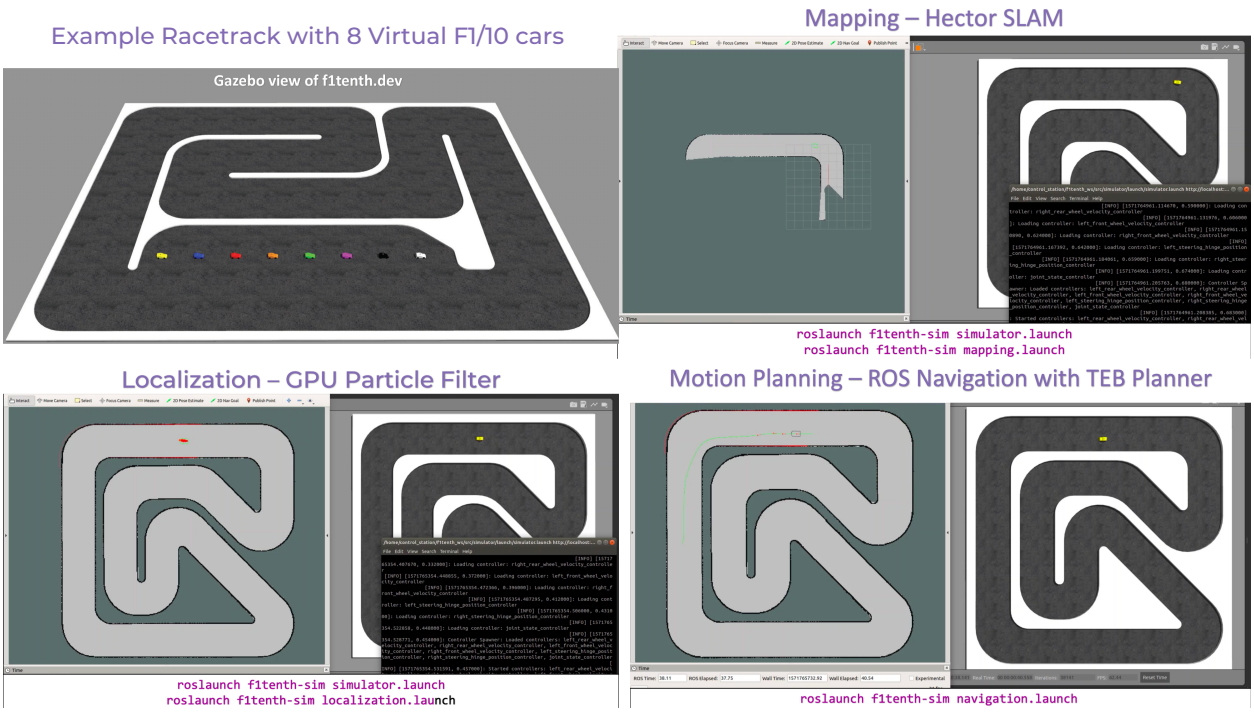

Fig. 4: [Top Left] The Gazebo GUI of the F1/10 simulator autonomous racing simulator showing multiple independent racecars.

default ROS navigation, Simultaneous Localization and Mapping (SLAM) [19] packages and robot-specific path planners like the TEB local planner [20].

3) Feed-Back mechanisms: The feed-back mechanism includes the state estimation nodes of the virtual racecar including the ego vehicle odometry calculator and opponent vehicle position and velocity estimation using ROS native linear and quadratic estimation filters.

Data generated using the feed-back mechanism is provided to the high level navigation navigation stack synchronously using the rate set by the controller-manager.

D. Simulation Back-End

1) Gazebo Physics Simulator: The open-source Gazebo physics is the mainstay of the F1/10 simulator and forms the majority of the back-end block. Gazebo was chosen for its high physics fidelity and seamless integration with ROS. The back-end physics simulator itself has two major functional blocks along with its primary purpose of simulating the vehicle physics:

- **Virtual Race-Track**
  The virtual race-track in the F1/10 simulator was carefully designed using CAD to represent a real race-track with equal assortments of sharp turns and straight stretches to allow testing a virtual racecar to its algorithmic limits.
- **Racing Metrics Telemetry**
  Each virtual racecar has an independent timing and tracking system that provides racing metrics telemetry like lap time, average lap speed and waypoint pose that are saved for every lap executed by the virtual racecar, and can be used to test the performance difference between racing algorithms.

A custom racetrack can be used in the F1/10 simulator by modifying the path to the 3D mesh in the world description file and changing the ground plane dimensions.

2) F1/10 rviz Visualizer: The F1/10 simulator is released with a set of *rviz* configuration files and associated ROS launch visualization nodes that aid in tracking a single virtual racecar as it traverses a virtual race track, providing critical visual information such as viewing the LiDAR data, the RGB camera data and IMU data etc., along with estimated and predicted data like the vehicle's instantaneous odometry and immediate goal pose.

IV. Multiple Autonomous Racecars

Figure 4 shows one of the unique features of the F1/10 simulator, which is the ability to independently spawn & control multiple virtual racecars (up to 16 racecars). This is achieved by leveraging the modular design of the virtual racecar and ROS name spaces. Autonomous racing also requires the ability of identifying and tracking the other autonomous agents in the race-track. Finally, the F1/10 simulator also provides a telemetry system to track the racing metrics of the virtual racecar.

A. ROS Namespace Delineation

Each virtual racecar is spawned under a unique ROS compliant string name space which eliminates unautho-

rized cross communication between multiple racecars. Name spaces also help eliminate concerns about some sensors (LiDARs, encoders, etc.) and actuators (torque controllers etc.) that have similar names across all virtual racecars, by forcing each component to publish and subscribe to the vehicles unique name space. The F1/10 simulator also maintains a set of global resources that are shared amongst all virtual racecars including global tf-frames [21] and navigation stack parameters including vehicle footprint and performance bounds.

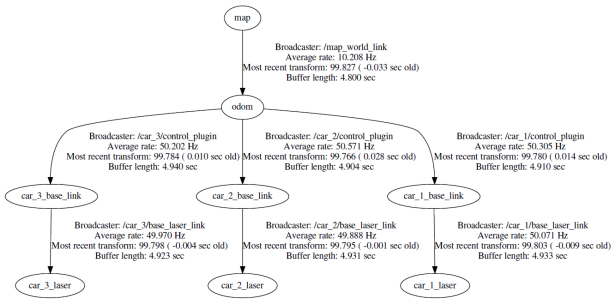

Fig. 5: The *rqt_graph* of three virtual racecars showing independent geometric frames (like *base_link* and *laser_link* etc.) under vehicle namespace and common geometric frames (like *map* and *odom*)

B. State Exchange Server

We have implemented a state exchange server that provides state estimation of other autonomous agents present in the race-track. This becomes especially necessary when addressing research questions such as dynamic obstacle avoidance autonomous overtaking. The F1/10 simulator can also estimate ego state for the purpose of measuring racing metrics.

1) Racecar State Estimation: Using ROS native linear and quadratic estimation techniques like EKF [16] and $AdaptiveMonteCarloLocalization(AMCL)$ [22], the F1/10 simulator is capable of identifying and tracking other autonomous agents in the race-track. To identify other autonomous agents, the ego vehicles $scanmatcher$-frame [23] from ROS Hector SLAM is employed to look for localized changes between two successive LiDAR scans and track the velocity of the changing cluster - providing information regarding pose and speed of a moving object in the LiDAR Field of View (FOV).

2) Racing Metrics: The F1/10 simulator is an autonomous racing simulator, and as such has the ability to measure racing metrics including lap_time: the time taken by the virtual racecar to traverse the race-track completely, lap_speed: the average speed of the virtual racecar during the lap and the highest & lowest instantaneous speeds, and $lap\_waypoints$: the actual path taken by the virtual racecar when traversing the lap.

C. V2V & V2X Communication

Each virtual racecar in the F1/10 simulator is equipped with the ability to communicate with a central node or with other virtual racecars. The communication standards can be described using an XML module and individual Vehicle2Vehicle (V2V) and Vehicle2Everything (V2X) communications can be enabled or disabled on the fly. The F1/10 simulator uses an overarching $Oracle$ (observation node) to control inbound and outbound communication using F1/10 specific UDP packets.

V. Research Enabled

The F1/10 simulator can be used to verify autonomous racing algorithms before implementing them on the real racecar. An online version is also being used for research and instruction into autonomy.

A. Sim2Real Pipeline

The F1/10 simulator is built to replicate the behavior & performance of the real F1/10 racecar, at most times mirroring the exact behavior of the real racecar, and is thus used to test & verify the performance of navigation algorithms. We provide an example for how the F1/10 simulator can be used to develop algorithm in simulation and how to translate that to results on the real testbed, thereby advancing the rapid development cycle. The following steps describe how the simulator was used to assign optimal lookahead distance for high speed pure-pursuit [24].

Fig. 6 provides an overview of the experiment workflow which includes the following sequential steps:

- **Mapping the Race Track**: The F1/10 racecar is manually driven around the race track to obtain an 2D occupancy grid map using the Hector SLAM [25] algorithm
- **ROSMap2Gazebo**: Using CAD, we extrude the map bounds by using a smoothing filter to filter out noises from the map point cloud and export the resulting 3D mesh to Gazebo as a world model.
- **Label Assignment**: The label assignment algorithm is run on the virtual race track in Gazebo. The lookahead list is set to the desired values.
- **Validation on Testbed**: The labels generated from the algorithm are exported to the F1/10 testbed and verified against simulation results.

In doing so, we can quickly go from a real racetrack, to a real map, to a 3D simulated racetrack and back to the real F1/10 testbed as described in Figure 6.

B. Research Enabled

Most of the research into autonomous racing algorithms enabled using F1/10 simulator are similar to those of the real F1/10 racecar, and can be broadly divided into the following two major categories:

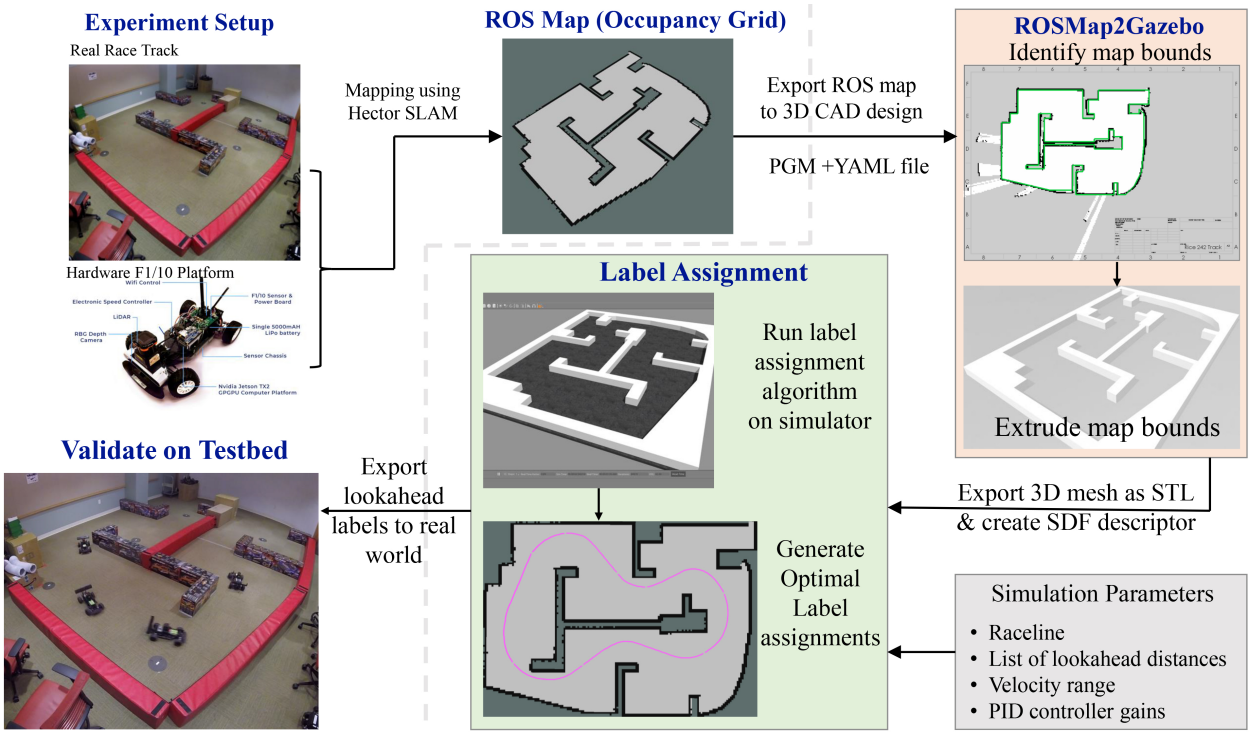

Fig. 6: An example of the Sim2Real used in Adaptive Lookahead Pure-Pursuit: Clockwise from Top Left: The F1/10 platform is manually driven around the race track to create a ROS map using traditional SLAM, the ROS map is exported to CAD where the map bounds are extruded and exported as a 3D mesh, the label assignment algorithm is performed on the new map using the set simulation parameters, & the labels are exported to be validated on the F1/10 platform

	F1/10 Simulator Research Features
Perception	LiDAR Scanmatcher [23], GPU Particle Filter [14] Visual Inertial Odometry [26] Ex: roslaunch f1tenth-sim localization.launch
Mapping	Hector SLAM [25], Gmapping [27] Google Cartographer [28] Ex: roslaunch f1tenth-sim mapping.launch
Navigation	Pure-Pursuit [24], ROS Navigation TEB Local Planner [20] Ex: roslaunch f1tenth-sim pure_pursuit.launch

TABLE I: Research enabled by the F1/10 simulator across disciplines: perception, planning and control, with accompanying easy to start launch commands (included with the simulator tutorial package)

1) Open-Loop Navigation: Algorithms in this category include Vector Field Histogram (VFH) [29] obstacle avoidance and dynamic window Follow the Gap (FTG) [30]. Optimizing the algorithms for high speeds requires significant tuning - which is made easy using the F1/10 simulator and the Sim2Real pipeline, and different speeds can be tested without worrying about collisions & damage to the real F1/10 racecar.

2) Closed-Loop Navigation: Pure-pursuit is a popular racing strategy that depends on a known map of the racetrack and highly accurate localization of the racecar pose & odometry information. The F1/10 simulator has been used to build on the advantages of the pure-pursuit by introducing a adaptive & variable lookahead distance that is optimal for a given stretch of the racetrack. The simulator supports autonomous racing using adaptive lookahead pure-pursuit & Timed Elastic Band (TEB) local planner plugin to base ROS navigation stack.

C. Classroom Instruction

The F1/10 simulator was scaled up and deployed on a GPU server at the University of Virginia to provide access to a total of 8 teams, each with an independent virtual racecar and dedicated workspace. Algorithms included wall-following, vector field histogram, follow the gap and Hector SLAM etc.

VI. Conclusion & Future Work

This paper presents F1/10 simulator - an open-source ROS & Gazebo based autonomous racing simulator modelled around the F1/10 autonomous racecar platform with an emphasis on modularity and ease of use. The simulator is currently being used for research into autonomous racing algorithms & as a tool of instruction for vehicular autonomy at major research universities. The project is released to the public and is currently maintained by the authors and other members of the community. The F1/10 simulator project will continue to be supported well into the future and upcoming releases include integrating machine learning libraries (including reinforcement learning) through dedicated APIs.

REFERENCES

[1] Matthew O'Kelly, Varundev Sukhil, Houssam Abbas, Jack Harkins, Chris Kao, Yash Vardhan Pant, Rahul Mangharam, Dipshil Agarwal, Madhur Behl, Paolo Burgio, et al. F1/10: An open-source autonomous cyber-physical platform. *arXiv preprint arXiv:1901.08567*, 2019.

[2] Martin Vajnar. *Model car for the f1/10 autonomous car racing competition*. PhD thesis, Master's thesis, Czech Technical University in Prague, 2017.

[3] Grady Williams, Brian Goldfain, Paul Drews, James M Rehg, and Evangelos A Theodorou. Autonomous racing with autorally vehicles and differential games. *arXiv preprint arXiv:1707.04540*, 2017.

[4] Daniel Gonçalves Almeida. *Implementing and Tuning an Autonomous Racing Car Testbed*. PhD thesis, 2019.

[5] Radoslav Ivanov, Taylor J Carpenter, James Weimer, Rajeev Alur, George J Pappas, and Insup Lee. Case study: Verifying the safety of an autonomous racing car with a neural network controller. *arXiv preprint arXiv:1910.11309*, 2019.

[6] Bernard Yett, Dimitrios Boursinos, and Feiyang Cai. Simulations of autonomous vehicle model using lidar sensing. 2016.

[7] Jan Dusil. Slip detection for f1/10 model car. 2019.

[8] Florian Curinga. Autonomous racing using model predictive control, 2017.

[9] Matthew O'Kelly, Hongrui Zheng, Achin Jain, Joseph Auckley, and Kim Luong. Tech report: Tunercar: A superoptimization toolchain for autonomous racing.

[10] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[11] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE, 2004.

[12] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*, 2017.

[13] Bharathan Balaji, Sunil Mallya, Sahika Genc, Saurabh Gupta, Leo Dirac, Vineet Khare, Gourav Roy, Tao Sun, Yunzhe Tao, Brian Townsend, et al. Deepracer: Educational autonomous racing platform for experimentation with sim2real reinforcement learning. *arXiv preprint arXiv:1911.01562*, 2019.

[14] Corey H Walsh and Sertac Karaman. Cddt: Fast approximate 2d ray casting for accelerated localization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[15] Greg Welch, Gary Bishop, et al. An introduction to the kalman filter. 1995.

[16] Edouard Ivanjko and I Petrovic. Extended kalman filter based mobile robot pose tracking using occupancy grid maps. In *Proceedings of the 12th IEEE Mediterranean Electrotechnical Conference (IEEE Cat. No. 04CH37521)*, volume 1, pages 311–314. IEEE, 2004.

[17] Sigeru Omatu, Michifumi Yoshioka, and Toshihisa Kosaka. Pid control of speed and torque of electric vehicle. In *2009 Third International Conference on Advanced Engineering Computing and Applications in Sciences*, pages 157–162. IEEE, 2009.

[18] Wm C Mitchell, Allan Staniforth, and Ian Scott. Analysis of ackermann steering geometry. Technical report, SAE Technical Paper, 2006.

[19] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[20] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. Online trajectory planning in ros under kinodynamic constraints with timed-elastic-bands. In *Robot Operating System (ROS)*, pages 231–261. Springer, 2017.

[21] Tully Foote. tf: The transform library. In *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*, pages 1–6. IEEE, 2013.

[22] Fitria Romadhona Quratul Aini, Agung Nugroho Jati, and Unang Sunarya. A study of monte carlo localization on robot operating system. In *2016 International Conference on Information Technology Systems and Innovation (ICITSI)*, pages 1–6. IEEE, 2016.

[23] Edwin B Olson. Real-time correlative scan matching. In *2009 IEEE International Conference on Robotics and Automation*, pages 4387–4393. IEEE, 2009.

[24] R Craig Coulter. Implementation of the pure pursuit path tracking algorithm. Technical report, Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, 1992.

[25] Stefan Kohlbrecher, Johannes Meyer, Thorsten Graber, Karen Petersen, Uwe Klingauf, and Oskar von Stryk. Hector open source modules for autonomous mapping and navigation with rescue robots. In *Robot Soccer World Cup*, pages 624–631. Springer, 2013.

[26] Michael Bloesch, Sammy Omari, Marco Hutter, and Roland Siegwart. Robust visual inertial odometry using a direct ekf-based approach. In *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 298–304. IEEE, 2015.

[27] BLEA Balasuriya, BAH Chathuranga, BHMD Jayasundara, NRAC Napagoda, SP Kumarawadu, DP Chandima, and AGBP Jayasekara. Outdoor robot navigation using gmapping based slam algorithm. In *2016 Moratuwa Engineering Research Conference (MERCon)*, pages 403–408. IEEE, 2016.

[28] Bo Xu, Zhengjun Liu, Yiran Fu, and Changsai Zhang. Research of cartographer laser slam algorithm. In *LIDAR Imaging Detection and Target Recognition 2017*, volume 10605, page 1060509. International Society for Optics and Photonics, 2017.

[29] Johann Borenstein, Yoram Koren, et al. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE transactions on robotics and automation*, 7(3):278–288, 1991.

[30] Volkan Sezer and Metin Gokasan. A novel obstacle avoidance algorithm:"follow the gap method". *Robotics and Autonomous Systems*, 60(9):1123–1134, 2012.