2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Chronos and CRS: Design of a miniature car-like robot and a software framework for single and multi-agent robotics and control

Andrea Carron*, Sabrina Bodmer, Lukas Vogel, René Zurbrügg, David Helm,
Rahel Rickenbach, Simon Muntwiler, Jerome Sieber and Melanie N. Zeilinger

***Abstract*— From both an educational and research point of view, experiments on hardware are a key aspect of robotics and control. In the last decade, many open-source hardware and software frameworks for wheeled robots have been presented, mainly in the form of unicycles and car-like robots, with the goal of making robotics accessible to a wider audience and to support control systems development. Unicycles are usually small and inexpensive, and therefore facilitate experiments in a larger fleet, but they are not suited for high-speed motion. Car-like robots are more agile, but they are usually larger and more expensive, thus requiring more resources in terms of space and money. In order to bridge this gap, we present Chronos, a new car-like 1/28th scale robot with customized open-source electronics, and CRS, an open-source software framework for control and robotics. The CRS software framework includes the implementation of various state-of-the-art algorithms for control, estimation, and multi-agent coordination. With this work, we aim to provide easier access to hardware and reduce the engineering time needed to start new educational and research projects.**

I. Introduction

In robotics and control development, experimental work is key to assess and test the effectiveness of algorithms. Car-like robots are great platforms to achieve this, as they mix agility with dynamic complexity. Furthermore, fleets of car-like vehicles can be used to simulate driving scenarios as well as multi-robot coordination strategies. However, the size and cost of available car-like platforms limit their usage, as they are comparably expensive and large in size. In this paper, we present Chronos and CRS. The former is an inexpensive, 1/28th scale car-like robot with open-source electronics. The small size of the vehicle enables experiments in small laboratories, as well as running multi-agent experiments in a confined environment. The overall cost of a single agent is below $300, which makes Chronos an ideal platform for research and educational projects. The CRS framework is an open-source software platform for testing control and robotics applications such as autonomous racing and single and multi-agent robotics. The current implementation comes with algorithms for control, safety, estimation, and multi-agent coordination, which reduces the implementation effort necessary to test new algorithms. The electronics schematics and PCBs, together with the software and firmware stacks are available online and open-source under a BSD 2-clause license in order to foster the wide-spread use of the platform, reduce the setup time, and encourage the exchange of code in the community.

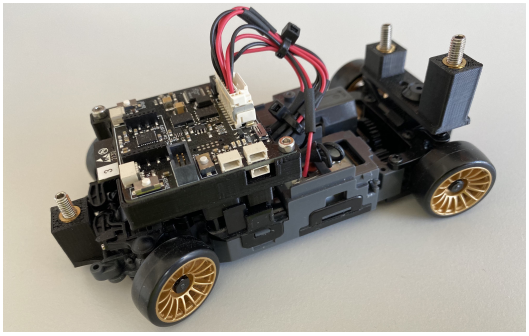

Fig. 1. The Chronos car. The chassis is taken from a Kyosho Mini-Z miniature car, while the PCB and the 3D printed mounts are custom made.

II. Related Work and Contributions

A wide range of wheeled robots are available on the market, for this reason, a complete survey of all available platforms is not within the scope of this paper. In this section, we list some of the most popular robots that are/were available in the last three decades and whose design is similar to the platform we propose. We categorize the robots as unicycle- and bicycle-type vehicles [1]. The former is a vehicle usually having two parallel-driven wheels, one mounted on each side of the center. The latter includes car-like robots with four wheels, where only the front two are steerable, usually via an Ackermann steering mechanism. The first successful small-scale unicycle was Khepera [2]. The first generation was designed at EPFL in 1991 and had widespread success in both research [3] and education. Another successful platform was the LEGO Mindstorms RCX [4], the first robotic platform from LEGO, whose combination with LEGO bricks made it popular for low-budget middle/high-school education [5] and research [6] projects. Another popular unicycle, also developed at EPFL, is the e-puck [7], proposed as a cheaper alternative to the bigger and more expensive Khepera III. Over the years, more powerful and expandable unicycle robots became available like iRobot Create [8], Pololu 3pi [9], Swarmrobot [10], AERobot [11], GRITSBot [12], Pheeno [13], Duckiebots [14], and Wheelbot [15]. More recently, remotely accessible unicycle swarm robotics platforms have been developed, among them, the Mobile Emulab [16], the HoTDeC testbed [17], the Robo-

The authors are members of the Institute of Dynamic Systems and Control, ETH Zurich, Sonneggstrasse 3, 8092 Zurich, Switzerland. The work of Simon Muntwiler was supported by the Bosch Research Foundation im Stifterverband.

*Corresponding author carrona@ethz.ch

tarium [18], and Duckietown [14]. A comprehensive list of multi-robot testbeds can be found in [19].

While these offer great platforms for applications related to trajectory planning, their main limitations for high performance controller development are their lack of agility and low maximum speed. This has motivated the use of car-like vehicles in research and education in recent years. The most common platforms are medium- to full-scale vehicles allowing the integration of powerful computers and to embed various sensors, like IMUs, CMOS/CCD motion sensors, LIDARs, wheel encoders, etc. Among the full-scale prototypes are the Stanford Shelley [20], Stanley [21], and Marty cars [22]; Formula Driverless custom cars [23] designed to compete in the Formula Student Driverless championship; and most recently Roborace DevBot 2.0 [24] and Dallara AV-21 [25]. Among the medium-scale vehicles, usually ranging between 1 to 10th and 18th scale, there are the Autorally [26], Donkey Car [27], F1-10 cars [28], [29], and the AWS Deep Racer [30]. The focus of these platforms is mainly oriented towards single-agent autonomous driving/racing and not multi-agent systems, as they require medium- to large-scale infrastructure and are thus comparably expensive and not used for experiments with many agents. The small-scale platforms presented in [31], [32], and [33] would fit the single- and multi-agent use-case, however, are only partially or not at all open-source, rendering them difficult to recreate.

On the software side, few toolboxes are currently available to solve control tasks. Among them: Drake [34], the Control Toolbox [35], which however is not maintained anymore, and OCS2 [36], which is a toolbox that includes only controllers, but not the full control pipeline including state estimators and safety filters.

The contribution of this paper is twofold: we present the design of Chronos, an inexpensive miniature car-like robot with open-source electronics and firmware. Thanks to its small-scale design, the robot can be used in confined spaces. The second contribution is the design of CRS, an open-source software framework for performing simulations and experiments. The software aims at supporting both researchers and practitioners by providing a modular framework to design control pipelines for both simulation and hardware experiments. It comes with implementations of state-of-the-art state estimators, controllers, and safety filters, which are non-trivial to set up. Furthermore, we show that CRS allows to quickly go from successful simulations to hardware experiments. Hardware, firmware, and the CRS software framework are available under a BSD clause-2 license[1]. Finally, the linked video showcases the platform[2].

III. Design Principles

In this section, we introduce the over-arching principles according to which Chronos and CRS are designed:

- *Fast dynamics*: We envision a 4-wheel agile and fast car-like robot that uses an Ackermann steering mechanism. The robot should produce highly dynamic motion, and reach comparably high speeds.
- *Small-scale*: Since space is a limited resource in many laboratories, the robots should be small-scale, allowing for operation in a small area.
- *Low-cost*: The cost of a single robot should be low, allowing research groups and students to perform experimental work with limited resources and facilitating the use for multi-agent applications.
- *User-friendliness*: The platform should be accessible to people with limited electronics knowledge and intermediate programming skills. This is a key requirement in order to become a widely-used platform for research and educational projects.
- *Open-source*: Electronics, firmware, and software should be open-source to facilitate their usage.

To the best of the authors' knowledge, there are currently no platforms that satisfy all these criteria, which motivated us to design Chronos and CRS in order to bridge this gap.

[1] https://gitlab.ethz.ch/ics/crs
[2] https://youtu.be/l1tMSebViZ0

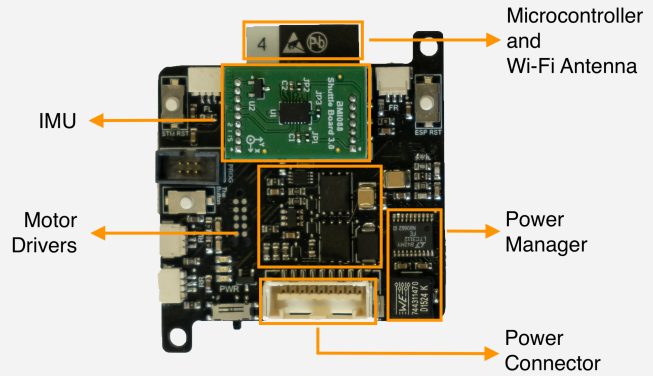

Fig. 2. The electronics board of Chronos. In the picture, the IMU, the motor driver, and the power manager are highlighted. The WiFi antenna is shown, while the microcontroller lies under the IMU.

IV. Chronos

In the following, we describe the hardware, electronics, and firmware stack used for Chronos.

A. Hardware

The following hardware requirements are derived from the previously stated design principles: reachable speeds of $3-4$ m/s; maximum size of 15 cm x 7 cm, thus allowing for operation in an area of about 3 m x 4 m; and costs below \$300. Based on these requirements, we choose the Kyosho Mini-z [37], a widely available 1/28th scale RC car, to be the base of Chronos. The Kyosho Mini-Z car is equipped with two DC motors, a drive-train and a steering actuator, and a potentiometer acting as a steering encoder.

We designed a custom electronics board, see Figure 2, composed of four modules: a microcontroller, a motor driver, sensors, and a power manager. At the core of the electronics board lies an Espressif ESP32-WROOM32D [38], a 32-bit microcontroller with a built-in antenna module, which supports Wi-Fi connectivity. Additionally, the microcontroller

also provides SPI connectivity, which allows the user to connect additional sensors and actuators. The motor driver is an STMicroelectronics STSPIN250 [39], which is used to drive both actuators - the drive-train and the steering actuator. The car's sensors include an inertial measurement unit (IMU) and a steering potentiometer. The used IMU is a Bosch BMI088 [40], which is connected to the microcontroller on an SPI bus. Chronos can be powered with four AAA batteries and it employs a buck-boost converter to maintain a constant supply of 5V and 2.2A to the motor. To protect the batteries, we use a low voltage monitoring circuit that turns Chronos off when the voltage level descents below a user-defined threshold.

B. Firmware

The microcontroller's firmware is based on the ESP-IDF development framework [41], with FreeRTOS real-time operating system installed on top. The firmware is task-oriented and the main tasks are communication with the CRS software and the execution of a low-level control loop for the steering angle and the drive-train acceleration. Chronos and the CRS framework communicate via a local Wi-Fi network. The microcontroller can connect to an access point and receive data through the Internet Protocol (IP). Data is serialized through the usage of Protocol Buffers (Protobuf) [42] and then sent as User Datagram Protocol (UDP) frames.

V. CRS Software Framework

The CRS framework aims to simplify simulation and transition to hardware experiments, as the same framework is used for both simulations and experiments. The framework is divided into two main parts: The first part (*crs*) contains the implementations of the main modules and it is possible to integrate this part of the framework into other software packages. The modules are in fact designed as toolboxes rather than as integrated applications in order to provide maximum flexibility to the users. Each module is built as a single catkin package enabling the user to compile only the required components. The second part (*ros4crs*), contains a wrapper around the provided modules, based on the robot operating system (ROS) [43] to enable communication among them and allow the user to run CRS in a distributed fashion.

Figure 3 shows a simplified overview[3] of the modules implemented in *crs* and their communication provided by *ros4crs*. The pipeline consists of a plant, which either represents a simulated dynamics model or real robotics hardware, e.g., Chronos; sensors to (partially) measure the state of the system; a state estimator, which estimates the system state from the sensor readings; a controller, which computes the next control input based on the current state estimate; and a safety filter, which checks if the proposed control input is safe to apply to the system. Given the modular structure of the framework, it is possible to adapt the implementation to custom models/algorithms and build custom control pipelines similar to the one shown in Figure 3. In the subsequent sections, we summarize the models/algorithms listed in Figure 3 for each module of the control pipeline.

[3]Two detailed UML diagrams are available at the following links: `https://gitlab.ethz.ch/ics/crs/-/wikis/crs.pdf` and `https://gitlab.ethz.ch/ics/crs/-/wikis/ros.pdf`

A. Dynamical Models

Models are a core component in the CRS framework and are of paramount importance to all elements in the pipeline, from simulation to control and estimation. In the following, we describe two widely used models: The kinematic and the dynamic bicycle models. For ease of presentation, both models are presented in their continuous-time formulation.

1) Kinematic Bicycle Model: The continuous-time kinematic bicycle model [44] is represented by the following differential equations

$$\dot{x}_p(t) = v(t)\cos(\psi(t) + \beta(t)), \tag{1}$$
$$\dot{y}_p(t) = v(t)\sin(\psi(t) + \beta(t)), \tag{2}$$
$$\dot{\psi}(t) = \frac{v(t)}{l_r}\sin(\beta(t)), \tag{3}$$
$$\dot{v}(t) = a(t), \tag{4}$$

where x_p and y_p are the world frame coordinates of the center of gravity, ψ is the heading angle, v is the total speed of the vehicle, and β is the slip angle at the center of mass and is defined as $\beta(t) = \arctan\left(\frac{l_r}{l_r+l_f}\tan\left(\delta(t)\right)\right)$. The model input $u = [\delta, a]$ consists of the steering angle δ and the drive-train acceleration a. The state is described by the vector $x = [x_p, y_p, \psi, v]$. The model has two parameters, l_f and l_r, which represent the distances from the center of mass to the front axle and rear axle, respectively.

2) Dynamic Bicycle Model: The continuous-time dynamic bicycle model [44] is governed by the following differential equations

$$\begin{aligned}
\dot{x}_p(t) &= v_x(t)\cos(\psi(t)) - v_y(t)\sin(\psi(t)),\\
\dot{y}_p(t) &= v_x(t)\sin(\psi(t)) + v_y(t)\cos(\psi(t)),\\
\dot{\psi}(t) &= \omega(t),\\
\dot{v}_x(t) &= \frac{1}{m}\big(F_x(t) - F_f(t)\sin(\delta(t)) + m v_y(t)\omega(t)\big),\\
\dot{v}_y(t) &= \frac{1}{m}\big(F_r(t) + F_f(t)\cos(\delta(t)) - m v_x(t)\omega(t)\big),\\
\dot{\omega}(t) &= \frac{1}{I_z}\big(F_f(t) l_f\cos(\delta(t)) - F_r(t) l_r\big),
\end{aligned} \tag{5}$$

where m is the mass of the vehicle, I_z is the inertia along the z-axis, x_p, y_p, ψ are the x-y coordinates and the yaw angle in the world frame, v_x and v_y are the longitudinal and lateral velocities in the body frame, and ω is the yaw rate. The lateral tire forces F_f and F_r are modeled with the simplified Pacejka tire model

$$\begin{aligned}
\alpha_f(t) &= \arctan\left(\frac{v_y(t) + \omega(t) l_f}{v_x(t)}\right) - \delta(t),\\
\alpha_r(t) &= \arctan\left(\frac{v_y(t) - \omega(t) l_r}{v_x(t)}\right),\\
F_f(t) &= D_f\sin(C_f\arctan(B_f\alpha_f(t))),\\
F_r(t) &= D_r\sin(C_r\arctan(B_r\alpha_r(t))),
\end{aligned} \tag{6}$$

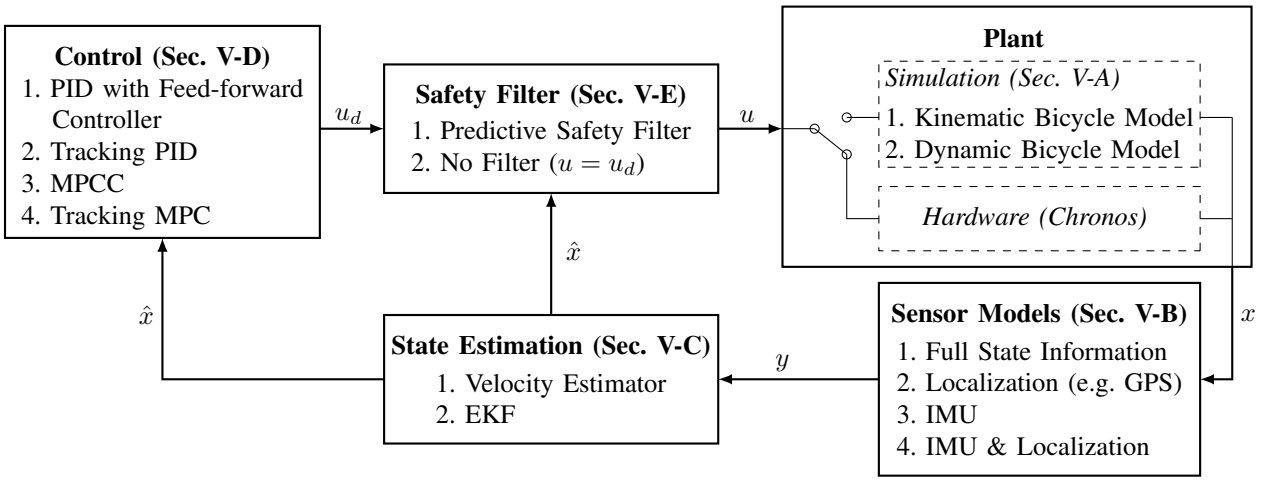

Fig. 3. Above schematic depicts the control pipeline established by CRS, including options for implemented algorithms described in Section V. The CRS framework can be used for both simulations and hardware experiments, since the plant block can switch between these two modes.

where α_f and α_r are the front and rear slip angles, and $B_f, B_r, C_f, C_r, D_f, D_r$ are the Pacjeka tire model parameters. The longitudinal force is modeled as a single force applied to the center of gravity of the vehicle and is computed as a combination of the drive-train command and the velocity

$$F_x(t) = \big(c_1 + c_2 v_x(t)\big)a(t) + c_3 v_x^2(t) + c_4, \tag{7}$$

where c_i with $i \in \{1, \dots, 4\}$ are appropriate parameters. The input to the system is the same as in the kinematic bicycle model, while the state is $x = [x_p,\ y_p,\ \psi,\ v_x,\ v_y,\ \omega]$.

B. Sensor Models

Sensor models are important for state estimation and simulation. In the following, we describe two sensor models.

1) Localization Systems: Localization systems, like, GPS or motion capture systems, usually only provide the pose of the system, i.e., the position and the orientation in the world frame. This results in the discrete-time sensor model

$$h(x(k)) = \begin{bmatrix} x_p(k)^\top & y_p(k)^\top & \psi(k)^\top \end{bmatrix}^\top.$$

2) Inertial Measurement Unit: Inertial measurement units (IMU) provide measurements of linear acceleration and angular velocities. For this reason, this sensor model can only be used with the dynamic bicycle model. The discrete-time measurement equation $h(x(k))$ is the following

$$\begin{bmatrix} \dot{v}_x(k) \\ \dot{v}_y(k) \\ \omega(k) \end{bmatrix} = \begin{bmatrix} \frac{1}{m}\big(F_x - F_f \sin(\delta(k)) + m v_y(k)\omega(k)\big) \\ \frac{1}{m}\big(F_r + F_f \cos(\delta(k)) - m v_x(k)\omega(k)\big) \\ \omega(k) \end{bmatrix},$$

where the forces can be described using the Pacejka tire model (6) and the longitudinal force model (7).

C. State Estimators

Most of the available localization systems only provide the spatial location and orientation of the vehicle, while linear and angular velocities have to be estimated. In this section, we present two methods for estimating those quantities: a velocity estimator and an extended Kalman filter.

1) Velocity Estimator: In the velocity estimator algorithm, the x- and y-velocities in world frame and the yaw-rate are first estimated via finite differences, i.e.,

$$v_x^w(k) = \frac{x_p(k) - x_p(k-1)}{T_s}, \quad v_y^w(k) = \frac{y_p(k) - y_p(k-1)}{T_s},$$
$$\omega(k) = \frac{\psi(k) - \psi(k-1)}{T_s}, \tag{8}$$

where T_s is the sampling time. These estimated quantities can be then filtered using a third-order Butterworth low-pass filter obtaining $\hat{v}_x^w(k), \hat{v}_y^w(k)$, and $\hat{\omega}(k)$, i.e., the estimated x- and y-velocities in world frame and the yaw-rate. To retrieve the estimate of the linear velocities in body frame the following transformation is performed

$$\begin{aligned} \hat{v}_x(k) &= \hat{v}_x^w(k)\cos(\hat{\omega}(k)) + \hat{v}_y^w(k)\sin(\hat{\omega}(k)), \\ \hat{v}_y(k) &= \hat{v}_y^w(k)\cos(\hat{\omega}(k)) - \hat{v}_x^w(k)\sin(\hat{\omega}(k)). \end{aligned} \tag{9}$$

2) Extended Kalman Filter: An alternative model-based approach is the extended Kalman filter that exploits a model of the system, e.g., the kinematic or dynamic bicycle model, and a model of the available sensors to estimate the state. The overall system model encompasses two discrete-time equations: the state space model and the output model, where the first model describes the dynamics and the second one the sensor model, i.e.,

$$\begin{aligned} x(k+1) &= f(x(k), u(k)) + w(k), \\ y(k) &= h(x(k), u(k)) + v(k), \end{aligned} \tag{10}$$

where $w(k)$ and $v(k)$ are the process and measurement noise and are usually modeled as Gaussian noise with zero-mean and variance Q and R, respectively. The standard extended Kalman filter update and prediction equations [45] are implemented.

D. Controllers

In the following, we describe three tracking controllers and one controller for autonomous racing, which are natively implemented in the CRS framework. For tracking, the control

objective is to accurately track a given reference or path, as e.g., the center line of the track or an optimal racing line. Whereas the objective for autonomous racing is to complete a lap on the track as quickly as possible.

1) PID: Two PID controllers are currently implemented: a tracking PID that tracks a constant set-point, and a path-following PID that tracks a given path. For the tracking PID, the drive-train and the steering commands depend on the position error e_p and the orientation error e_ψ, respectively, with the two errors defined as

$$e_p = \sqrt{e_x^2 + e_y^2}, \qquad e_\psi = \arctan 2(\frac{e_y}{e_x}) - \psi, \tag{11}$$

where $e_x = x_p - x_r$ and $e_y = y_p - y_r$ define the error in x- and y-direction, and $r = (x_r, y_r)$ is the set-point. The applied inputs follow from the standard PID controller structure, i.e.,

$$a(k) = k_{p,a} e_p(k) + k_{i,a} \sum_{j=0}^{k} e_p(j) + k_{d,a}(e_p(k) - e_p(k-1)),$$

$$\delta(k) = k_{p,\delta} e_\psi(k) + k_{i,\delta} \sum_{j=0}^{k} e_\psi(j) + k_{d,\delta}(e_\psi(k) - e_\psi(k-1)).$$

The path-following PID computes the predicted lateral distance to the center line e_l by using the kinematic model, and follows the standard PID controller structure to calculate the steering input. The applied torque is set to a constant value in order to achieve a desired steady state velocity.

2) PID with Feed-forward Controller: The implemented PID controller with feed-forward controller is based on the controller proposed in [46] and is intended for path following, e.g., a track's center line or a racing line. It makes use of a lateral feedback controller combined with a feed-forward controller to control the steering angle δ, and a longitudinal feed-forward controller to control the acceleration a. The lateral controller is designed as in [46, Chapter 3], where the steering input is computed as

$$\delta(x) = \delta_{\text{ff}}(x) + \delta_{\text{fb}}(x), \tag{12}$$

with feed-forward component $\delta_{\text{ff}}(x)$ computed as

$$\delta_{\text{ff}}(x) = \left(l_f + l_r + k_{\text{ug}}(x) \frac{v_x^\top v_x}{g} \right) \kappa(x),$$

where $\kappa(x)$ is the curvature of the reference and $k_{\text{ug}}(x)$ is the vehicle under-steer gradient as defined in [46, Equation (3.7)]. The lane keeping feedback term $\delta_{\text{fb}}(x)$ is computed as

$$\delta_{\text{fb}}(x) = \delta_{\text{control}}(x) + \delta_{\text{damping}}(x),$$

with

$$\begin{aligned} \delta_{\text{control}}(x) &= -k_c \left(e(x) + x_{\text{la}} \sin(\alpha_e(x)) \right), \\ \delta_{\text{damping}}(x) &= -k_d \left(\omega - \|v_x\|_2 \kappa(x) \cos \alpha_e(x) \right), \end{aligned}$$

where k_c, k_d are control gains, $e(x)$ is the lateral error, i.e., the distance of the center of gravity to the closest point on the path, x_{la} is the look-ahead distance, $\alpha_e(x)$ is the angle error, i.e., the difference between the car orientation and the tangent to the path, and ω is the yaw rate.

Additionally, a longitudinal controller is used to control the acceleration. Thereby, the objective is to track a constant target acceleration a_{target}, which is reduced whenever a turn is ahead. The heuristic longitudinal control law is computed as

$$a = a_{\text{target}} - k_a \bar{\kappa}(x) v_x^\top v_x, \tag{13}$$

where k_a is the control gain and $\bar{\kappa}(x)$ is the average curvature over a look-ahead horizon.

3) Tracking MPC: The goal of the tracking MPC controller is to track a given reference trajectory or set-point and is formalized as the following nonlinear optimization problem

$$\begin{aligned} \min_{[u_0,\dots,u_N]} & \sum_{i=0}^{N} \|x_i - x_i^r\|_Q^2 + \|u_i - u_i^r\|_R^2 \\ \text{s.t. } & x_{i+1} = f(x_i, u_i),\ x_0 = x(k), \\ & x_i \in \mathbb{X},\ u_i \in \mathbb{U}, \end{aligned} \tag{14}$$

where x_i^r and u_i^r are state and input references, the function f describes the dynamics of the system, using either the discretized kinematic or dynamic bicycle model, and $\mathbb{X}$ and $\mathbb{U}$ are state and input constraints. Finally, the integer N is the prediction horizon length.

4) MPCC: In the following, we briefly review the Model Predictive Contouring Controller (MPCC), one of the most commonly used planning and control algorithms for autonomous racing used for example in [47], [48]. The controller seeks to maximize the progress along a given reference path while satisfying the constraints imposed by the boundaries of the track. The control problem is formalized by the nonlinear optimization problem

$$\begin{aligned} \min_{[u_0,\dots,u_N]} & \sum_{i=0}^{N} \|\varepsilon(x_i)\|_Q^2 - Q_{\text{adv}} \gamma(x_i) + \|u_i\|_R^2 \\ \text{s.t. } & x_{i+1} = f(x_i, u_i),\ x_0 = x(k), \\ & x_i \in \mathbb{X},\ u_i \in \mathbb{U}, \gamma_i(x_i) > 0, \end{aligned} \tag{15}$$

where $\varepsilon(x) = [\varepsilon_l(x), \varepsilon_c(x)]$ denotes the lag and contouring error, i.e., the lateral and longitudinal error from a given reference path, which depends non-linearly on the state. The variable $\gamma(x_i)$ represents the progress of the vehicle along a given reference path, e.g., the center line, and is a function of the vehicle's position. The dynamical model f used in (15) can be either the discretized kinematic or dynamic bicycle model described in the previous sections. State and input constraints are defined by the sets $\mathbb{X}$ and $\mathbb{U}$, respectively. Finally, the integer N is the prediction horizon length.

E. Predictive Safety Filter

A predictive safety filter aims to keep a system within state and input constraints despite being controlled by a potentially unsafe control input. The implemented safety filter has a focus on autonomous racing [49], and guarantees safety by computing a sequence of safe backup control inputs that lead the agent towards a set of known safe states, where the first input of the sequence is as close as possible to the desired

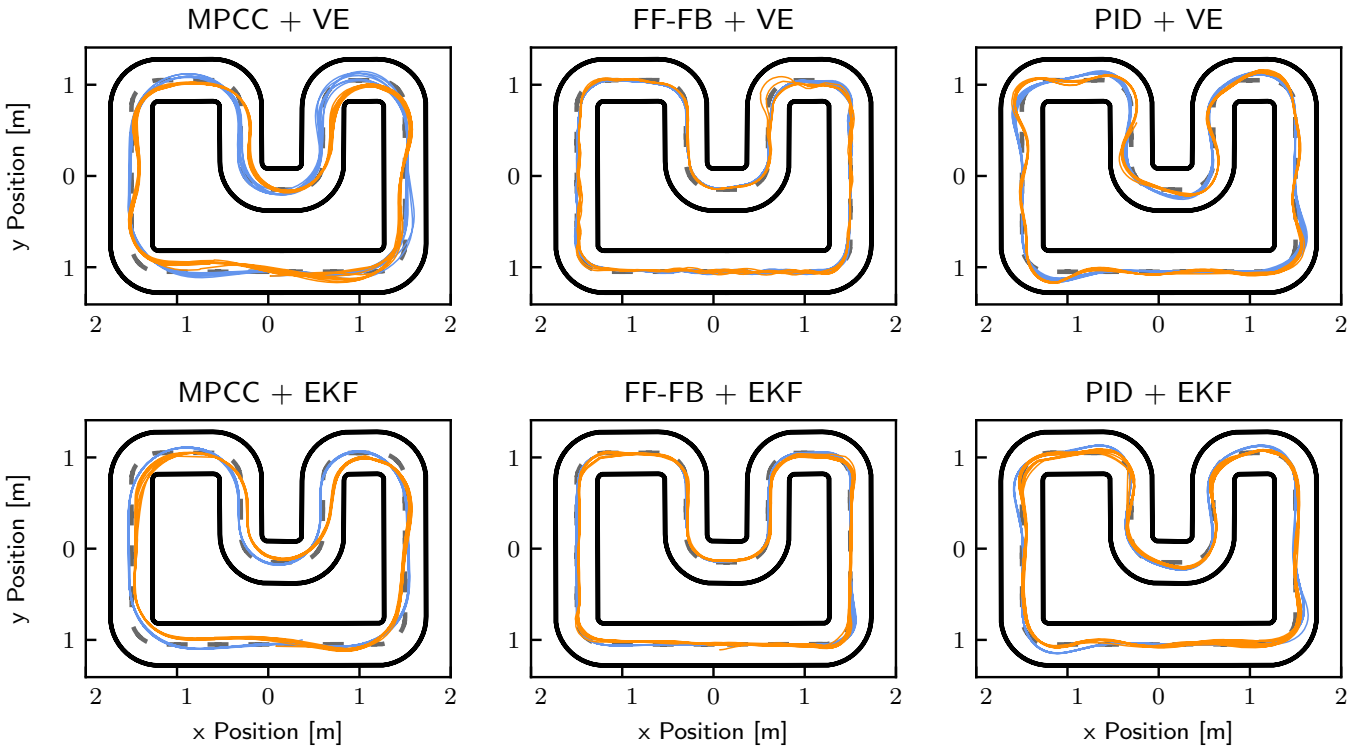

Fig. 4. Car trajectories collected from experiments for six control pipelines. The track boundaries are marked in solid black and the center-line in dashed grey. The simulation trajectories are marked in blue and hardware experiments in orange.

control signal u_d. The safe control input can be obtained by solving the following nonlinear optimization problem

$$\begin{aligned} \min_{[u_0,\dots,u_N]} & \|u_0 - u_d\|_R^2 + \sum_{i=0}^{N-1} \|\Delta u_i\|_{\Delta R}^2 \\ \text{s.t. } & x_{i+1} = f(x_i, u_i),\ x_0 = x(k), \\ & x_i \in \mathbb{X},\ u_i \in \mathbb{U},\ x_N \in \mathbb{X}_N, \end{aligned} \tag{16}$$

where $\Delta u_0 := u_0 - u(k-1)$, and $\Delta u_i := u_i - u_{i-1}$ for $i = 1, .., N-1$. The additional term in the cost function is a regularization that penalizes the rate of change of the inputs in order to encourage a smoother control trajectory. State, input, and terminal state constraints are defined by the sets $\mathbb{X}$, $\mathbb{U}$, and $\mathbb{X}_N$, respectively. Finally, the integer N is the prediction horizon length.

VI. Simulations and Experiments

In this section, we show the behaviour of the following six control pipelines: MPCC, PID with feed-forward controller, and PID, each combined with both the velocity estimator and the EKF as state estimators. The trajectories of multiple laps for each of the pipelines tested on simulation (blue) and hardware experiments (orange) are shown in Figure 4. We initialized both simulations and hardware experiments with same initial conditions. From the figure, we can see that the trajectories obtained in simulation mimic well the ones obtained from hardware experiments. This similarity is key in order to move from successful simulation to hardware experiments rapidly. Figure 4 also shows that using the EKF with any of the presented controllers increases the controller's performance since the time delay introduced in the state estimates is reduced compared to the velocity estimator. Among all presented controllers, the MPCC controller generates the fastest and most aggressive trajectories. The average lap time using the MPCC controller over 10 laps is $7.23\text{s} \pm 0.13\text{s}$ and $6.92\text{s} \pm 0.11\text{s}$ obtained with the velocity estimator and EKF, respectively. In this setting, we can generate a highly dynamic motion reaching a maximum speed of about $2.5\ \text{m/s}$ towards the end of the $2.4\ \text{m}$ long main-straight and a maximum yaw rate of about $5\ \text{rad/sec}$ in the two narrower U-turns. Finally, the PID with a feed-forward controller performs better than the PID controller tracking the center line. The predictive capabilities given by the feed-forward term ensure fewer oscillations and higher tracking accuracy.

VII. Conclusions

In this paper, we presented the design of Chronos and CRS. Unlike other car-like platforms, Chronos is an inexpensive small-scale robot that can be used for single and multi-agent control and robotics applications in a confined space. CRS is a flexible software framework that allows for the easy application of advanced control and estimation algorithms in simulation and hardware experiments. The electronics of Chronos and CRS are open-source under a BSD clause-2 license to foster their widespread use.

REFERENCES

[1] P. Corke, *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*. Springer Publishing Company, Incorporated, 1st ed., 2013.

[2] F. Mondada, E. Franzi, and P. Ienne, "Mobile robot miniaturisation: A tool for investigation in control algorithms," in *Experimental Robotics III* (T. Yoshikawa and F. Miyazaki, eds.), (Berlin, Heidelberg), pp. 501–513, Springer Berlin Heidelberg, 1994.

[3] P. F. M. J. Verschure, T. Voegtlin, and R. J. Douglas, "Environmentally mediated synergy between perception and behaviour in mobile robots," *Nature*, vol. 425, no. 6958, pp. 620–624, 2003.

[4] "Lego mindstorm." `https://www.lego.com/mindstorms`. Accessed: 2022-07-30.

[5] G. Kiss, "Using the lego-mindstorm kit in german computer science education," in *2010 IEEE 8th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pp. 101–104, 2010.

[6] A. Carron and E. Franco, "Receding horizon control of a two-agent system with competitive objectives," in *2013 American Control Conference*, pp. 2533–2538, 2013.

[7] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J. christophe Zufferey, D. Floreano, and A. Martinoli, "The e-puck, a robot designed for education in engineering," in *In Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, pp. 59–65, 2009.

[8] "irobot create." `https://www.irobot.com/`. Accessed: 2022-07-30.

[9] B. Thurský and A. Dubek, "Using pololu's 3pi robot in the education process," 2010.

[10] S. Kernbach, "Swarmrobot.org - open-hardware microrobotic project for large-scale artificial swarms," 2011.

[11] M. Rubenstein, B. Cimino, R. Nagpal, and J. Werfel, "Aerobot: An affordable one-robot-per-student system for early robotics education," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6107–6113, 2015.

[12] D. Pickem, M. Lee, and M. Egerstedt, "The gritsbot in its natural habitat - a multi-robot testbed," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4062–4067, 2015.

[13] S. Wilson, R. Gameros, M. Sheely, M. Lin, K. Dover, R. Gevorkyan, M. Haberland, A. Bertozzi, and S. Berman, "Pheeno, a versatile swarm robotic research and education platform," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 884–891, 2016.

[14] L. Paull, J. Tani, H. Ahn, J. Alonso-Mora, L. Carlone, M. Cap, Y. F. Chen, C. Choi, J. Dusek, Y. Fang, D. Hoehener, S.-Y. Liu, M. Novitzky, I. F. Okuyama, J. Pazis, G. Rosman, V. Varricchio, H.-C. Wang, D. Yershov, H. Zhao, M. Benjamin, C. Carr, M. Zuber, S. Karaman, E. Frazzoli, D. Del Vecchio, D. Rus, J. How, J. Leonard, and A. Censi, "Duckietown: An open, inexpensive and flexible platform for autonomy education and research," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1497–1504, 2017.

[15] A. R. Geist, J. Fiene, N. Tashiro, Z. Jia, and S. Trimpe, "The wheelbot: A jumping reaction wheel unicycle," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 9683–9690, 2022.

[16] D. Johnson, T. Stack, R. Fish, D. M. Flickinger, L. Stoller, R. Ricci, and J. Lepreau, "Mobile emulab: A robotic wireless and sensor network testbed," in *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pp. 1–12, 2006.

[17] A. Stubbs, V. Vladimerou, A. Fulford, D. King, J. Strick, and G. Dullerud, "Multivehicle systems control over networks: a hovercraft testbed for networked and decentralized control," *IEEE Control Systems Magazine*, vol. 26, no. 3, pp. 56–69, 2006.

[18] D. Pickem, P. Glotfelter, L. Wang, M. Mote, A. Ames, E. Feron, and M. Egerstedt, "The robotarium: A remotely accessible swarm robotics research testbed," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1699–1706, 2017.

[19] A. Jiménez-González, J. R. Martinez-de Dios, and A. Ollero, "Testbeds for ubiquitous robotics: A survey," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1487–1501, 2013.

[20] J. Funke, P. Theodosis, R. Hindiyeh, G. Stanek, K. Kritatakirana, C. Gerdes, D. Langer, M. Hernandez, B. Müller-Bessler, and B. Huhnke, "Up to the limits: Autonomous audi tts," in *2012 IEEE Intelligent Vehicles Symposium*, pp. 541–547, 2012.

[21] S. Thrun, "Toward robotic cars," *Commun. ACM*, vol. 53, p. 99–106, apr 2010.

[22] J. Y. Goh, T. Goel, and J. Christian Gerdes, "Toward Automated Vehicle Control Beyond the Stability Limits: Drifting Along a General Path," *Journal of Dynamic Systems, Measurement, and Control*, vol. 142, 11 2019. 021004.

[23] J. Kabzan, M. Valls, V. Reijgwart, H. Hendrikx, C. Ehmke, M. Prajapat, A. Bühler, N. Gosala, M. Gupta, R. Sivanesan, A. Dhall, E. Chisari, N. Karnchanachari, S. Brits, M. Dangel, I. Sa, R. Dube, A. Gawel, M. Pfeiffer, and R. Siegwart, "Amz driverless: The full autonomous racing system," 05 2019.

[24] "Roborace." `https://roborace.com`. Accessed: 2022-07-30.

[25] A. Wischnewski, M. Geisslinger, J. Betz, T. Betz, F. Fent, A. Heilmeier, L. Hermansdorfer, T. Herrmann, S. Huch, P. Karle, F. Nobis, L. Ögretmen, M. Rowold, F. Sauerbeck, T. Stahl, R. Trauth, M. Lienkamp, and B. Lohmann, "Indy autonomous challenge - autonomous race cars at the handling limits," in *12th International Munich Chassis Symposium 2021* (P. Pfeffer, ed.), (Berlin, Heidelberg), pp. 163–182, Springer Berlin Heidelberg, 2022.

[26] B. Goldfain, P. Drews, C. You, M. Barulic, O. D. Velev, P. Tsiotras, and J. M. Rehg, "Autorally: An open platform for aggressive autonomous driving," *IEEE Control Systems*, vol. 39, pp. 26–55, 2019.

[27] "Donkey car." `http://www.donkeycar.com`. Accessed: 2022-07-30.

[28] M. O'Kelly, H. Zheng, D. Karthik, and R. Mangharam, "F1tenth: An open-source evaluation environment for continuous control and reinforcement learning," in *Proceedings of the NeurIPS 2019 Competition and Demonstration Track* (H. J. Escalante and R. Hadsell, eds.), vol. 123 of *Proceedings of Machine Learning Research*, pp. 77–89, PMLR, 08–14 Dec 2020.

[29] G. M. Jon Gonzales, Ugo Rosolia, "Barc: Berkeley autonomous race car project."

[30] "Aws deep racer." `https://docs.aws.amazon.com/deepracer/`. Accessed: 2022-07-30.

[31] C. Hsieh, Y.-L. Chuang, Y. Huang, K. Leung, A. Bertozzi, and E. Frazzoli, "An economical micro-car testbed for validation of cooperative control strategies," in *2006 American Control Conference*, pp. 6 pp.–, 2006.

[32] K. K. Leung, C. H. Hsieh, Y. R. Huang, A. Joshi, V. Voroninski, and A. L. Bertozzi, "A second generation micro-vehicle testbed for cooperative control and sensing strategies," in *2007 American Control Conference*, pp. 1900–1907, 2007.

[33] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1:43 scale rc cars," *Optimal Control Applications and Methods*, vol. 36, no. 5, pp. 628–647, 2015.

[34] R. Tedrake and the Drake Development Team, "Drake: Model-based design and verification for robotics," 2019.

[35] M. Giftthaler, M. Neunert, M. Stäuble, and J. Buchli, "The control toolbox — an open-source c++ library for robotics, optimal and model predictive control," in *2018 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)* (H. Kurniawati, E. Drumwright, B. MacDonald, T. Fraichard, and N. Ye, eds.), (Piscataway, NJ), pp. 123 – 129, IEEE, 2018-06-11.

[36] R. S. L. E. Zurich, "Ocs2: Optimal control for switched systems," 2022.

[37] "Kyosho." `http://kyosho.com/mini-z-info/`. Accessed: 2022-08-12.

[38] "Espressif ESP32 WROOM datasheet, howpublished = `https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf`, note = Accessed: 2022-08-12."

[39] "STMicroelectronics STSPIN250 datasheet." `https://www.st.com/resource/en/datasheet/stspin250.pdf`. Accessed: 2022-08-12.

[40] "Bosch BMI088 datasheet." `https://download.mikroe.com/documents/datasheets/BMI088_Datasheet.pdf`. Accessed: 2022-08-12.

[41] "ESP-IDF development framework." `https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html`. Accessed: 2022-08-12.

[42] Google, "Protocol buffers." `http://code.google.com/apis/protocolbuffers/`.

[43] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source robot operating system," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, 2009.

[44] R. Rajamani, *Vehicle Dynamics and Control*. Mechanical Engineering Series, Springer US, 2011.
[45] L. A. McGee, S. F. Schmidt, and A. R. Center., *Discovery of the Kalman filter as a practical tool for aerospace and industry [microform] / Leonard A. McGee and Stanley F. Schmidt*. National Aeronautics and Space Administration, Ames Research Center Moffett Field, Calif, 1985.
[46] K. M. Kritayakirana, *Autonomous vehicle control at the limits of handling*. PhD thesis, 2012.
[47] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1: 43 scale rc cars," vol. 36, no. 5, pp. 628–647, 2015.
[48] L. P. Fröhlich, C. Küttel, E. Arcari, L. Hewing, M. N. Zeilinger, and A. Carron, "Model learning and contextual controller tuning for autonomous racing," 2021.
[49] B. Tearle, K. P. Wabersich, A. Carron, and M. N. Zeilinger, "A predictive safety filter for learning-based racing control," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7635–7642, 2021.